75:42 - Taller de Programación I

Ejercicio Nº ___ Padrón _____

Alumno _____ Firma _____

Nota:		**Corrige**:		**Entrega #1**
				Fecha de entrega
				Fecha de devolución

Nota:		**Corrige**:		**Entrega #2**
				Fecha de entrega
				Fecha de devolución

El presente trabajo, así como la entrega electrónica correspondiente al mismo, constituyen una obra de creación completamente personal, no habiendo sido inspirada ni siendo copia completa o parcial de ninguna fuente pública, privada, de otra persona o naturaleza.

Facultad de ingeniería de la UBA

75.42/95.08 Taller de Programación I
Cátedra Veiga

Trabajo Práctico Final
Micromachines

Segundo cuatrimestre de 2019

Integrante	Padrón	Correo electrónico
Calvo, Mateo Iván	98290	mate.-95@hotmail.com
Sandez, Jorge	92190	sandez.jorge@gmail.com
Ortiz, Javier	96598	ortizjavier160994@gmail.com

Link al repositorio:
https://github.com/mateoicalvo/micromachines/

Índice

1. Manual de Proyecto

1.1. Integrantes

Los integrantes del grupo de desarrollo y sus respectivas tareas fueron:

- Calvo, Mateo Iván: desarrollo del servidor
- Sandez, Jorge: desarrollo del cliente
- Ortiz, Javier: desarrollo de la grabación, bots y mods.

1.2. Enunciado

El enunciado del trabajo práctico se presenta en la sección final de apéndices.

1.3. División de tareas y evolución del proyecto

1.3.1. Para el Servidor

Se presenta aquí el cronograma real del desarrollo del proyecto

- Draft del modelo: Semanas 1 a 3.
- Pista exteriores y autos moviéndose libremente: Semanas 2 a 6.
- Autos interactuando con la pista y otros elementos: Semanas 2 a 7.
- Servidor Multipartidas con partidas multijugador. Victoria y derrota: Semanas 4 a 8.
- Armado del entregable: Semana 8.

1.3.2. Para el Cliente

- Mostrar una imagen, Mostrar una animación: semanas 1 a 2.
- Renderizado del escenario incluyendo la cámara: semanas 2 a 4.
- Animación de los elementos dinámicos: semanas 4.
- Comunicación con el servido, pantalla de conexión: semanas 4 a 8.
- Incorporación de scripts: semanas 4 a 8.
- Armado del entregable: semana 8.

1.4. Inconvenientes encontrados

A continuación se listan los inconvenientes encontrados durante el desarrollo del trabajo práctico.

1.4.1. En el servidor

- *Dispatching de eventos:* Encontrar la "mejor" manera de enviar, recibir, y coordinar eventos fue una de las complicaciones principales de todo el desarrollo. Para atacar este problema, se optó por un modelo del tipo orientado a eventos, mediante el uso de colas protegidas para comunicar hilos.

- *Coordinación de hilos:* Siendo el mutithreading un tópico central en el desarrollo del trabajo práctico, se debió prestar especial cuidado al manejo de hilos. Si bien las colas protegidas permitieron abstraer la comunicación de manera razonablemente cómoda, algunas estructuras o clases debieron protegerse. Para el lanzado y cerrado de hilos, el paradigma *RAII* resultó especialmente útil, ya que clarificó la manera (y el lugar) en que se utilizaron los hilos. Además, resultó crítico controlar el tiempo de ejecución de los hilos. Teniendo en cuenta que la simulación del tiempo no podía tener atrasos (pero si *saltos* que compensaran el tiempo perdido), se debió tener en cuenta el tiempo de ejecución del programa durante cada simulación. En este aspecto, la ayuda recibida de la cátedra fue de vital importancia para lograr ejecutar dichas secciones críticas a *"constant rate"*.

- *Identificación de entidades:* Para lograr la comunicación específica (y en algunos casos unívoca) entre entidades del modelo se optó por mantener identificadores globales y locales. Los identificadores globales permitieron referenciar entidades de manera unívoca. Esto fue necesario ya que ciertos eventos eran pertinentes a un usuario específico, no debiendo los demás enterarse de su ocurrencia. Por otro lado, los identificadores locales posibilitaron la optimización de los datos enviados debido a que se guardaron en tipos de datos más pequeños.

1.4.2. En el cliente

- Se perdio mucho tiempo para la instalación y utilización de `ffmpeg` para grabar el escenario. Debido a la poca documentación y poco entendimiento sobre la librería.
  También se perdió tiempo intentando integrar el sonido a las grabaciones, sin éxito alguno.

- Se debió ahondar en la documentación de SDL para comprender cómo se realizaban ciertas acciones específicas. Por ejemplo, se debió articular el renderizado sobre texturas y/o pantalla según fuera requerido, ya que para realizar una grabación fue requisito contar con estructuras auxiliares que permitieran copiar los píxeles renderizados a un *buffer* utilizable por el módulo grabador.

1.4.3. En la integración cliente-servidor

- No seguir la recomendación de la cátedra de confeccionar primero un módulo (internamente desacoplado) trajo la complicación de no tener hasta los momentos finales un programa ejecutable. Por lo tanto, se retrasó la resolución de problemas que de otra manera hubieran surgido al principio del desarrollo, tales como las conversiones entre sistemas de coordenadas. Por otro lado, tener un acoplamiento prácticamente nulo entre el cliente y el servidor derivó en no tener que solucionar interdependencias que hubieran escapado a la modularización.

1.5. Análisis de puntos pendientes

Al momento de la segunda entrega, restan implementar las siguientes características:

1.5.1. En el cliente

- Grabación con audio: no pudo realizarse, y tampoco se encontraron soluciones claras o factibles.

1.5.2. En el servidor

- Implementación de mods.
- Implementación de los modificadores boost y aceite: Si bien la colisión se detecta correctamente y el modificador desaparece, resta aplicar los efectos en el vehículo.
- Implementación de la disminución de velocidad al salirse de la pista: ídem punto anterior.

1.6. Herramientas

Herramientas auxiliares que utilizaron para hacer el proyecto, como editor de interfaz gráfica, control de versiones, herramienta para generar documentación, herramientas para debug, etcétera.

- Control de versiones: `git` mediante GitHub.
- Debugging: `GDB`, a través de los IDEs utilizados
- Persistencia de escenarios de juego: Json (librería: `nhlomann-json`).
- Físicas del juego: `Box2D`.
- Herramienta para parte gráfica: `SDL2`.
- Bot para el auto: *lua5.3*
- Grabación: *ffmpeg*
- Otras librerías usadas: `sdl-image`, `sdl-mixer`, `sdl-ttf`.
- Desarrollo: `Sublime Text`, `Visual Studio Code` y `CLion`.

1.7. Conclusiones

Se presentan a modo de conclusión los siguientes aspectos:

1.7.1. Servidor

Para el servidor, interesa destacar la dificultad del desarrollo, no tanto en la implementación, que fue bien cubierta durante las clases de la materia, sino en el tiempo requerido para idear una solución viable a cada problema que surgía, de modo que no implicara una reestructuración pesada del código y que resultara fácil de integrar con los demás módulos del servidor y el cliente.

1.7.2. Cliente

Como conclusión del cliente, se destaca que fue una tarea bastante larga con SDL, sobre todo con mucha dificultad a la hora de manejar botones en las pantallas que no sean partida. También fue un proceso de aprendizaje un poco vertiginoso en una herramienta que necesita un cierto tiempo de adaptación. Algunas cosas resultaron un poco más simples, pero requirió más programación con dicha librería. Se delegaron todas las tareas de dibujar a las escenas y quedaron muchas situaciones procedurales en la cual primero

se dibuja una capa, luego otra capa y se sigue ese patrón. Se cree que por el tiempo dado para el trabajo, no había otra opción para esto. Por lo que se verá mucho en cada escena que cada una va dibujando de esta forma y lo ultimo siempre se sobrepone a lo que se dibujo anteriormente. Con el tema de los hilos necesarios para cada cliente, creemos que se pudo lograr un diseño decente en temas de eventos y de objetos que se agregan al mapa o eventos que se envian al servidor. Por lo que quedo bastante facil el hecho de, por ejemplo, agregar un nuevo consumible, sonido, animacion, etc. Con AnimacionFactory esto se logro de buena manera. Hubo mucha dificultad a la hora de la cámara ya que con el tema de dibujar solamente lo que se ve en ese sector, se tardo mucho tiempo y se tomaron caminos erróneos hasta llegar al correcto, gracias al consejo del docente. También se pudo aplicar bien el constant rate loop para que se dibuje de forma constante y que las iteraciones no atrasen las animaciones, sino que en el caso que se pierdan frames, se sigue con el siguiente. Se llevo un tiempo largo hasta lograr un diseño que sea del agrado de los integrantes ya que al querer conectar la grabación con el dibujado de las escenas, hubo ciertos cambios que atrasaron un poco el avance, pero el resultado fue bueno ya que la grabación se realiza sin problemas.

2. Documentación Técnica

2.1. Requerimientos de software

Para el compilado, desarrollo, prueba y depurado del proyecto es necesario: (Basado en lo utilizado, podría funcionar con versiones inferiores/diferentes de las herramientas)

- C++11
- Linux Ubuntu 18.04.2 LTS
- SDL: libsdl2-dev, libsdl2-mixer-dev, libsdl2-image-dev, libsdl2-ttf-dev
- lua5.3 liblua5.3-dev
- libavutil-dev libavformat-dev libavcodec-dev libswscale-dev libswresample-dev

2.2. Descripción general

Globalmente se cuenta con dos aplicaciones, una para el servidor y otra para el cliente (jugador).

En el servidor se concentra la lógica del juego completa, siendo el cliente un "observador" de lo que ocurre en el modelo. El servidor cuenta, a grandes rasgos, con los siguientes módulos:

- Servidor: Concentra la lógica multipartida, y la aceptación y comunicación con los clientes. Aquí se manejan las partidas y la sala de espera.
- Modelo: Mantiene la lógica del juego. Para ello se vale de los módulos:
  - Entidades: Nuclea el comportamiento de las entidades del juego, tanto vehículos como modificadores.
  - Físicas: Abstrae el comportamiento físico de los objetos, maneja las colisiones y la simulación física
  - Superficies: Abarca la la interacción con el terreno, tanto en la pista como fuera de ella.

2.3. Módulo Servidor

2.3.1. Descripción General

En el módulo servidor se encuentran las clases responsables de controlar el agregado y eliminación de partidas, la recepción de eventos, la aceptación de clientes y el manejo de partidas y la Sala de espera

2.3.2. Clases

- CoordinadorPartidas: Es responsable de manejar los eventos de creación e inicio de partidas, además de la desconexión de los jugadores y el manejo del input del usuario recibido a través de la red.
- DistribuidorEventos: Se encarga de recibir todos los eventos de todos los clientes, y redirigirlos o actuar en consecuencia según el tipo de evento.
- HiloAceptador: Su responsabilidad radica en estar a la espera de nuevos clientes, agregándolos a la sala de espera cuando uno de ellos se conecta.
- Jugador: Coordina un socket TCP y dos hilos (uno para enviar eventos, otro para recibirlos).
- Partida: Es el hilo donde efectivamente corre la simulación física del juego, permite agregar jugadores y mantiene la lógica de inicio de partida (inicia cuando todos los jugadores están listos para iniciar). Se encarga de asignar los vehículos disponibles según el número de jugadores presentes.
- SalaDeEspera: Abstrae la coordinación de los jugadores aceptados, de manera protegida. Permite agregar jugadores y quitarlos, además de enviar eventos a los jugadores presentes en la sala, por ejemplo para notificarles de la creación de una nueva partida.
- Servidor: Clase principal del módulo servidor, coordina la ejecución de la sala de espera, el hilo aceptador de clientes, el distribuidor de eventos y el coordinador de partidas. Inicia todos los hilos del programa, y también los detiene y finaliza al salir. En su ejecución, se queda a la espera de recibir un caracter de escape.
- SocketTCPServidor: Permite enlazarse a un puerto y escuchar nuevas conexiones, devolviendo un SocketTCP aceptado.

- ConfigServidor: Abstracción de todos los parámetros de configuración del servidor.

2.3.3. Diagramas UML

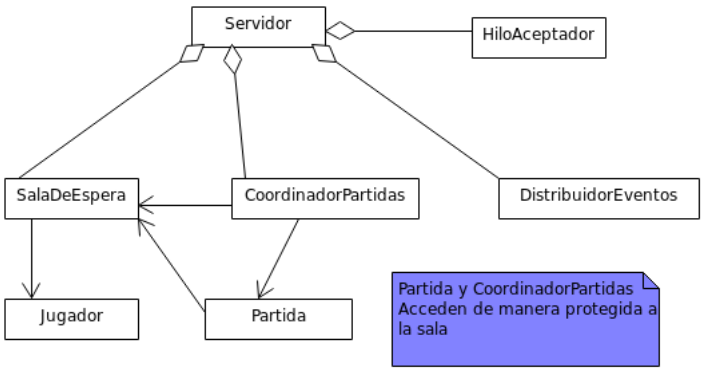


Figura 1: Modulo Servidor

2.3.4. Descripción de archivos y protocolos

2.4. Módulo Modelo

2.4.1. Descripción General

En el módulo modelo se encuentran las clases y abstracciones pertinentes al juego en sí.

2.4.2. Clases

- Modificador: Modelo abstracto de un modificador en la pista.
- Aceite: Modelo del modificador de aceite presente en la pista.
- Barro: Modelo del modificador de barro presente en la pista.
- Boost: Modelo del modificador de boost de velocidad presente en la pista.
- CajaVida: Modelo del modificador vida presente en la pista. Permite obtener una cantidad de puntos de vida a sumar en la salud del auto.
- Piedra: Modelo del modificador de piedra presente en la pista.

- Vehiculo: Modelo del vehiculo lógico (no físico). Mantiene atributos de salud, velocidad máxima, agarre, maniobrabilidad y aceleración, además de una referencia a su jugador dueño para permitir comunicarle eventos. Conoce su punto de reaparición, y permite asignarle uno nuevo.

- Checkpoint: Modela un punto de paso en la carrera, y permite registrar el paso de un vehículo si éste último cruzó por el checkpoint anterior correspondiente.

- Carrera: Modela la carrera a través de un conjunto de Checkpoints. Permite registrar vehículos, asignarles checkpoints y mantiene la lógica de finalización de la carrera a través del número de vueltas.

- Colisionable: Abstracción utilizada para manejar las colisiones entre entidades a través de un tipo.

- Identificable: Abstracción que permite identificar entidades del mismo tipo de manera unívoca.

- Mundo: Concentra abtracciones de la simulación del mundo físico, permitiendo además el manejo de los eventos de conducción de cada vehículo. Permite cargar una pista a partir de un archivo de persistencia.

- Fisicas: Abstrae los conceptos físicos del mundo, permitiendo agregar objetos y actuar sobre ellos, tanto para reubicarlos o para quitarlos. Funciona como interfaz con la librería externa Box2D, permitiendo actuar en los parámetros físicos de los cuerpos.

- ContactListener: se encarga de resolver todas las colisiones entre entidades. Es una implementación de la interfaz presentada por Box2D para manejar las colisiones entre cuerpos.

- B2DVehículo: abstracción de un vehículo simulado en Box2D, controla la fricción, y el torque producidos según las acciones recibidas. Permite actuar sobre el vehículo para cambiar su velocidad a través de fuerzas y torques.

- Transformacion: Interfaz presentada como *Functor* para permitir actuar luego de las colisiones detectadas por la librería Box2D, ya que la misma no permite alterar el mundo físico durante colisiones.

- Quitar: Transformación que quita un objeto del mundo físico.

- Reubicar: Transformación que reubica un objeto del mundo en una posición y ángulo determinados.
- Posicion: Modelo de posición 2D en el plano, con rotación agregada.
- Superficie: Modelo de un terreno abstracto de la pista.
- SuperficieFactory: Implementación del patrón *FactoryMethod* que permite instanciar superficies específicas.
- SuperficieArena: Modelo de un terreno hostil para el vehículo, que se destruye en contacto con la misma.
- SuperficiePista: Modelo de asfalto.
- SuperficieTierra: Modelo de la zona aledaña al asfalto, en la cual el vehículo pierde velocidad.

2.4.3. Diagramas UML

2.4.4. Descripción de archivos y protocolos

Este módulo se vale de archivos en formato *json* para cargar las pistas, los checkpoints y las superficies.

2.5. Módulos compartidos y Protocolo

2.5.1. Descripción General

En el módulo common se agruparon aquellas abstracciones que fueran de utilidad tanto para el cliente como para el servidor. Si bien algunas de ellas son meramente utilitarias, otras tales como los eventos sirvieron como comunicación entre el módulo cliente y servidor.

- ColaBloqueante: Utilizada para obtener eventos a través de la red mediante sockets.
- ColaProtegida: Utilizada para comunicar los hilos tanto en el cliente como en el servidor.
- Conversor: Utilizado para convertir magnitudes entre los puntos de vista cliente-mapa-servidor.
- Cronometro: Utilizado para medir el tiempo en los ciclos a *constant rate*.

- EnviadorEventos: Hilo utilizado para el envío de eventos via sockets TCP.
- Handler: interfaz que permite manejar eventos del juego.
- Hilo: Abstracción de un hilo de ejecución.
- RecibidorEventos: Permite recibir eventos vía sockets TCP.
- Tile: Modelo de un cuadrado en un mapa a cuadrículas.
- SocketTcp: Abstracción de un socket del lenguaje C, permite enviar y recibir tiras de bytes.
- Protocolo: Clase responsable de la comunicación, permite enviar y recibir distintos tipos de datos.
- Evento: Clase abstracta que representa un evento del juego; mediante la estrategia de dispatching permite el manejo de los mismos en las clases especializadas para reaccionar como corresponda.
- EventoFactory: Permite instanciar eventos concretos a partir de un identificador.

2.6. Módulo Cliente

2.6.1. Clases

A continuacion las clases principales del cliente.

- HiloDibujador: Procesa los eventos y deriva en el renderizador el dibujado de las escenas y/o el dibujo para el HiloGrabador. En esta clase vive el stack de escenas.
- HiloGrabador: Graba determinados frames por segundo y utiliza un DobleBuffer bloqueante para obtener nuevos frames.
- Renderizador: Levanta una textura y la renderiza.
- Escenas: Clase abstracta con las funciones del ciclo del cliente, a ser implementadas por cada escena concreta. Cada clase concreta se dibujará de una forma diferente ya que algunas tendran botones y otras el mapa del juego. Para cambiar de escena se utiliza el stack de escenas del hilo dibujador. En cada escena según la situación del juego, se apila la próxima pantalla a mostrar y para volver a la pantalla anterior, se hace un pop. A continuación una descripción de cada escena.

- EscenaMenu: Esta es la escena principal que tiene las opciones para jugar y salir. Es la primera escena que ve el usuario y la ultima que deberia ver en un ciclo de vida çorrecto"de crear/unirse a partida, jugar, ganar/perder y salir al menú completamente para luego cerrar la conexión.

- EscenaSala: La escena sala es donde se verán las partidas ya creadas y también se dara la opción de crear una nueva. en cualquier caso el usuario luego de esta escena ya estaria unido a una partida correctamente. Las partidas que ya estan llenas no se muestran y las que estan en juego tampoco.

- EscenaLobby: Aca es donde se pueden ver los usuarios que van a formar parte de la partida, a medida que van ingresando se van agregando a la lista de dicha escena en tiempo real.

- Escena Partida: esta es la escena "principal"de todo el juego, la que tiene el juego en sí mismo, aca es donde el usuario competirá con otros para ganar. Es la que tiene la información en tiempo real de como esta el juego, pero del lado cliente, es decir en que posición del mapa esta tal usuario, que consumibles hay, que sucesos fueron agregados al juego (con sucesos nos referimos a eventos como explosiones, choques, etc).

- Escena Podio: la escena final, la cual se activara cuando todos los jugadores hayan terminado la carrera. Esta escena tan solo mostrará un podio con el auto ganadorr, el 2do y el 3er puesto de dicha carrera. A partir de aca se puede volver al menú para volver a empezar otra partida si es necesario.

- AnimacionFactory: Factory para objetos del juego. Se crea una vista de objeto segun los parametros especificados.

- Camara: la camara es la encargada de çaluclar"lo que se debería mostrar en cada iteración dependiendo de la posición del auto. En la misma, tenemos los métodos correspondientes a dibujar cada parte del juego. Los métodos son: dibujarPista, dibujarObjetos y dibujarEventosTemporales. En cada método se hace los cálculos necesarios basandose en bloques de distancia desde el usuario hacia cada objeto. Por ejemplo para la pista se calculan los x e y correspondientes a los bloques que deben ser dibujados, para luego dibujar esa parte de la pista. Esa información de x e y luego sera utilizada en los otros dos métodos para dibujar los objetos y los eventos temporales dependiendo de esa "zona"de bloques. Cabe aclarar que en dibujarObjetos NO se dibuja el

auto asociado al cliente en cuestión, ya que el mismo siempre se mantendrá en el medio de la camara. Acá también se limita la cantidad de sonidos que se reproducirá en la camara del usuario. Obviamente dicha clase tiene una referencia a la pista.

- Pista: la pista es la que contiene la información del juego en si misma. Contiene un mapa el cual tiene como key las çapas"de la pista, esto es: como primera capa seria el pasto/arena/agua, como segunda, la pista. Esto fue hecho de esta manera para que al dibujar ciertas cosas esten por encima de otras. Luego por cada capa del mapa, se tiene una matriz asociada donde se incluyen las animaciones que serian las partes de la pista que se deben dibujar en dicho bloque. Para nuestro trabajo solo se utilizaron 2 capas, el terreno y la pista. Luego se tienen dos mapas asociados a los objetos dinamicos de la pista y a los eventos temporales. Los objetos dinamicos se refieren a los autos y los consumibles en cuestion, mientras que los eventos temporales son explosiones, choques y frenadas. Se permite desde afuera de esta clase: agregar, borrar, obtenerIds y objetos de ambos mapas. Esto se utiliza en la escena partida cuando se debe por ejemplo, agregar o quitar un consumible. También cuando se inicializan los autos, y para agregar explosiones, frenadas, etc.

- Sonido: Modela el audio de SDL, reproducira la musica de fondo y los efectos de sonido correspondiente a cada animación. Se puede manipular el volúmen y detener la reproducción. También tiene un flag en el cual se inidica si es un sonido en forma de loop o es solo una reproducción.

- ObjetoDinamico: Clase abstracta que sirve de wrapper para que una animación tenga una posición en la pista correspondiente. Contiene un sonido característico, una animación, posiciones, ángulos y también la vida(la cual es utilizada en el auto). Esta clase tambien es utilizada para agregar sucesos en el mapa como choques, explosiones o frenadas, en los caso de los choques y frenadas se utiliza una animacion "vacía"la cual es un png de 1x1. En el caso de los consumibles se utiliza un sonido "vacíoza que los mismos no tienen un sonido característico de momento. Se prefirió dejarle la opción abierta para que ambos casos puedan agregar tanto sonidos como animaciones.

- RecibidorEventos: Hilo que recibe eventos del socket y los encola en una cola no bloqueante para comunicar con el hilo principal.

- EnviadorEventos: Hilo que envia eventos al socket, obtenidos de una cola bloqueante donde inserta eventos el hilo principal.
- Boton: es un wrapper para una animación el cual tiene la particularidad que a través de un clic de mouse, devuelve si fue seleccionado o no, es útil para cuando se debe seleccionar una partida o ciertas opciones.
- EventoGUIHandler: interfaz que deben implementar todas las escenas. La misma se utiliza para manejar cada input en cada escena, en general son 3: clic mouse, key down y key up. Luego, cada escena tiene su propia funcionalidad dependiendo de lo que se haga. En algunos casos se enviara un evento al servidor, en otro se realizará algo sobre la misma escena.
- OutputStream: contiene las estructuras de la librería avformat para stremear audio o video. Se inicializa obteniendo una referencia a la clase OutputFormat, pidiéndole información sobre el Codec que se va a utilizar (ya sea audio o video) y la estructura AVStream. También contiene un Frame y un AVPacket que serán prescindibles para escribir audio/video.
- OutputFormat: encapsula la estructura AVFormatContext, que se inicializa con los datos obtenidos del filepath. Si no encuentra datos del formato, con mp4 por defecto. Intercala los paquetes recibidos por los streams y escribe tanto el header como el trailer en el archivo, para un correcto uso del formato.
- Codec: se encarga de la codificación de los paquetes de audio y video.
- VideoCodec: posee un rescalador de imagenes de la librería swscale. En esta clase se inicializan los parametros buscados para el video, ya sea bitrate, fps. Por defecto se utiliza el mejor formato de pixeles del codificador inicializado.
- Frame: esta clase encapsula los AVFrame que son frames decodificados con información. Cómo por ejemplo el formato, el tamaño y el pts (tiempo en que el mismo debe ser mostrado al usuario).

2.6.2. Diagramas UML

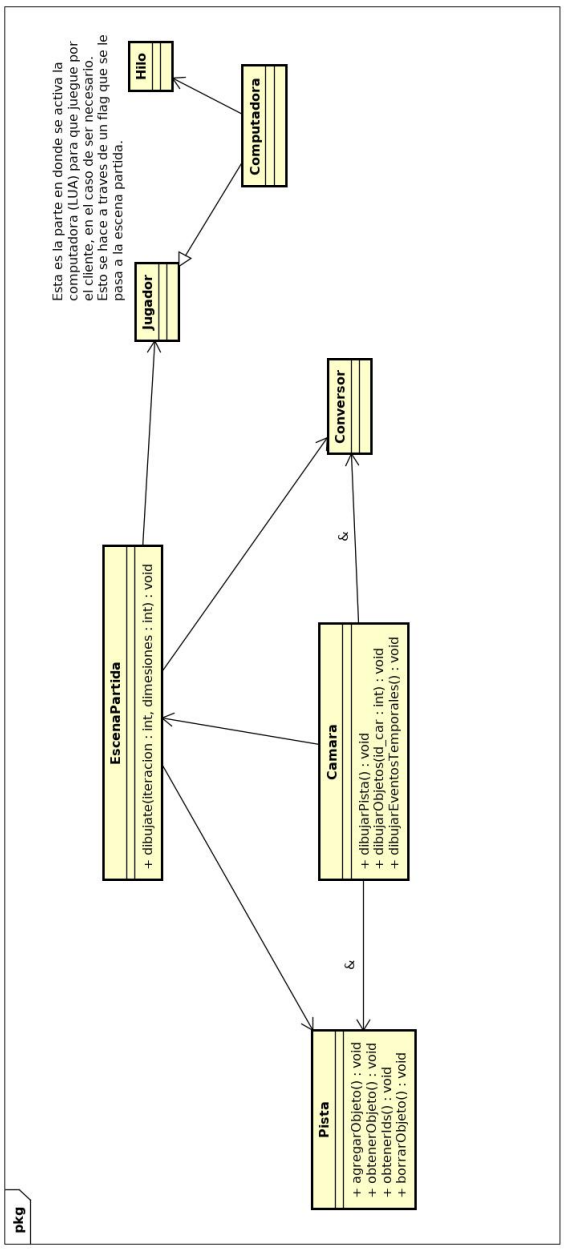


Figura 2: Diagrama de clases para las Escenas

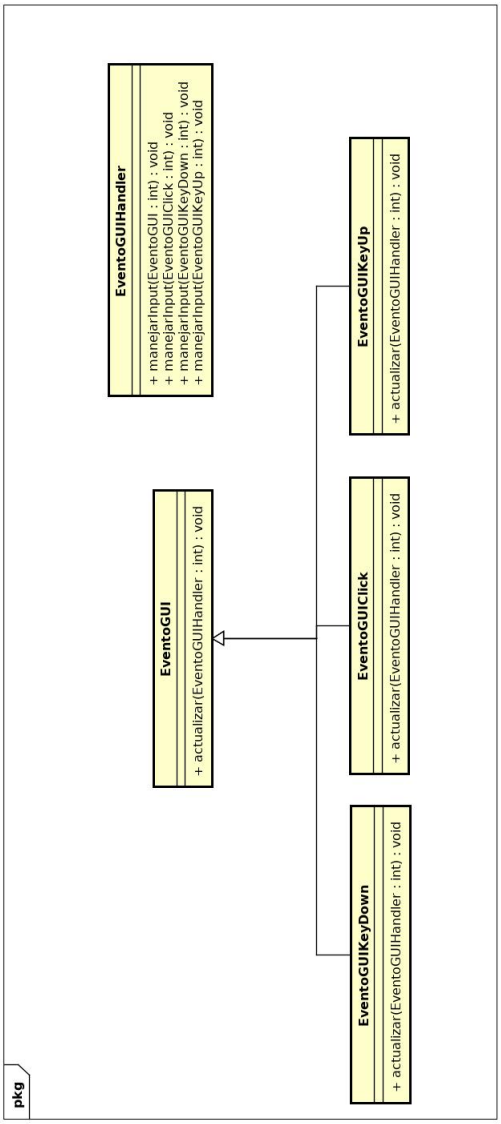


Figura 3: EventosGUI

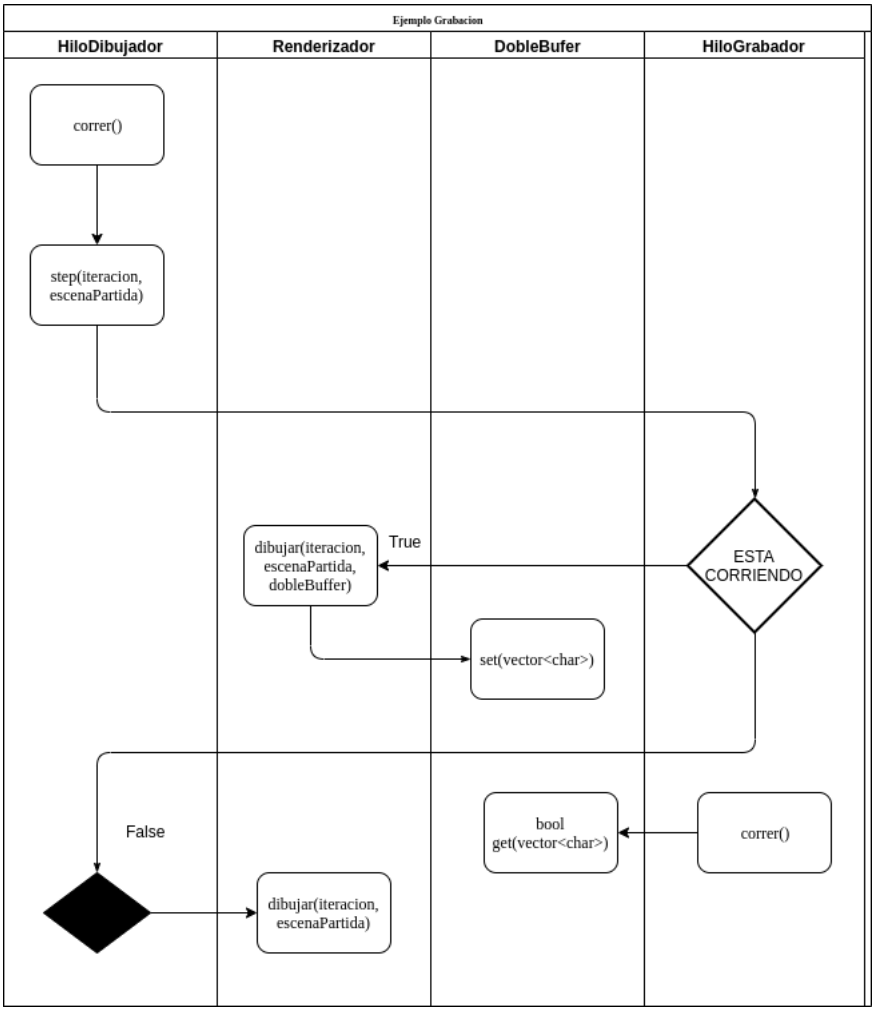


Figura 4: Funcionamiento del hilo grabador

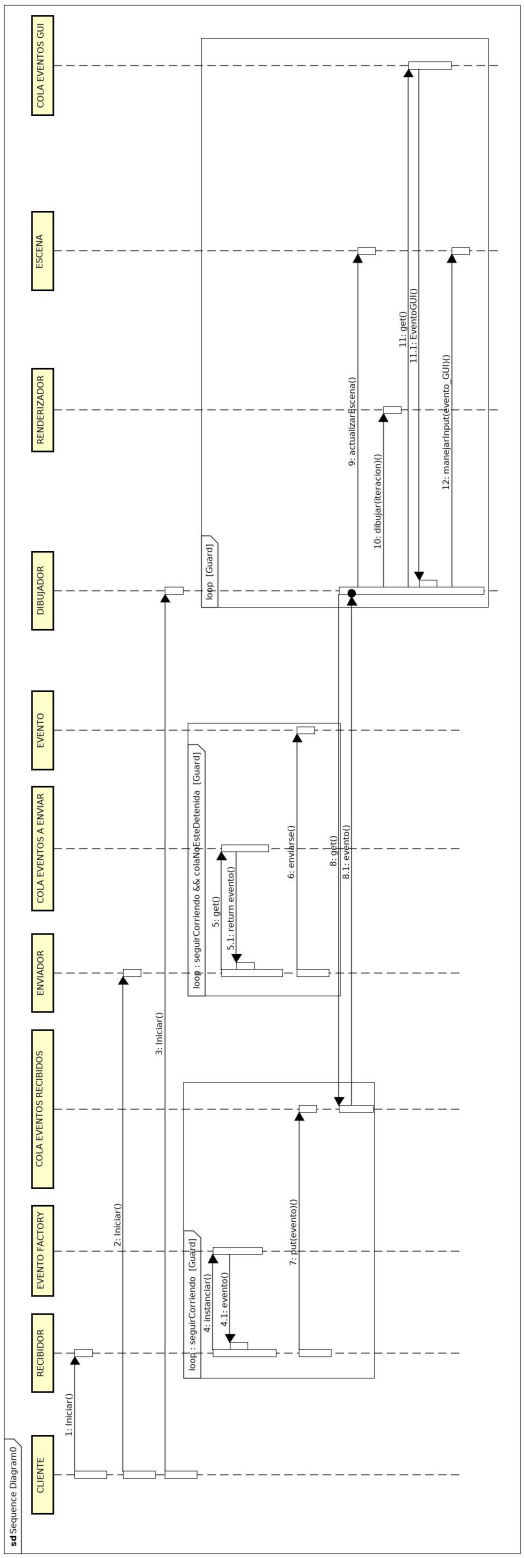


Figura 5: Funcionamiento de los hilos en el cliente

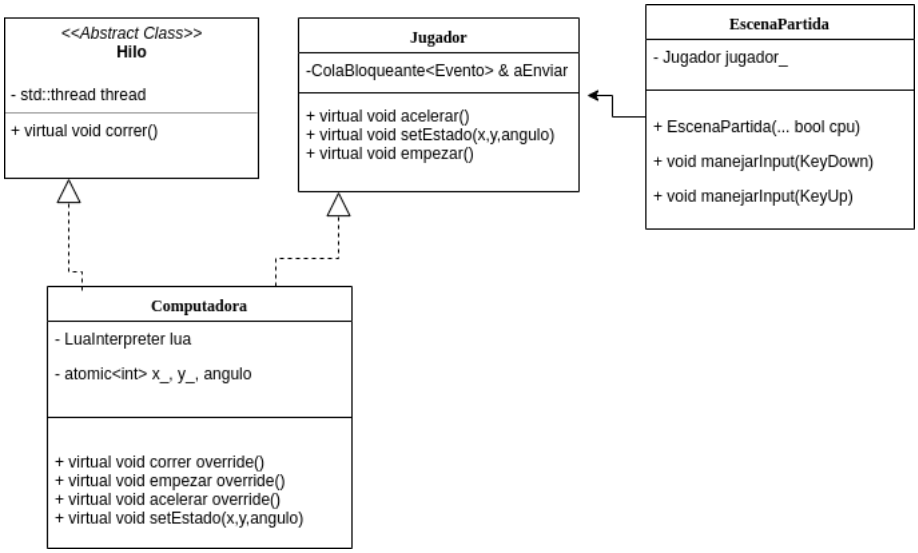


Figura 6: Integracion Con Lua

2.7. Programas intermedios y de prueba

Como programas intermedios y de prueba, se utilizó una script de envío de eventos al servidor para testear su respuesta, y la utilidad *TestBed*, provista por `Box2D` para familiarizarse con la librería. Además se utilizó la aplicacion *Tiled* para diseñar las pistas del juego.

2.8. Código fuente

El código fuente (tanto de la aplicación cliente como servidor) puede consultarse en los apéndices.

3. Manual de Usuario

Esta es una guía de instalación que permitirá obtener, compilar y ejecutar el juego *Micromachines*. Para su ejecución, el juego requiere que un servidor se esté ejecutando, y al menos un cliente para poder iniciar una partida. La instalación del cliente y el servidor, además de su ejecución, se detallan a continuación.

3.1. Instalación

3.1.1. Requerimientos de software

Las siguientes librerías o aplicaciones son indispensables para la instalación del juego. En el proceso de instalación se detalla cómo pueden obtenerse, y qué versiones se utilizaron para desarrollar el juego.

- Aplicación `cmake`.
- Aplicación `make`.
- Librería `SDL2-dev`.
- Librería `SDL2-image-dev`.
- Librería `SDL2-mixer-dev`.
- Librería `SDL2-ttf-dev`.
- Librería `liblua5.3`.
- Librería `lublua5.3-dev`.
- Librerías `FFMPEG`:

  `libavutil-dev`

  `libavformat-dev`

  `libavcodec-dev`

  `libswscale-dev`

Recomendados:

- Sistema Operativo: Ubuntu 18.04 LTS.
- Aplicación `git`.

Opcionales (En caso de errores):

- `ffmpeg`

3.1.2. Requerimientos de hardware

- 2 GB RAM.
- Conexión de red para partidas multijugador.
- Procesador Dual-Core 2.0GHz.

3.1.3. Proceso de instalación

A continuación se describe el proceso de instalación de las librerías. Luego, se muestra cómo instalar y ejecutar tanto el servidor como el cliente. Abriendo una terminal, se debe ejecutar:

Librerías de SDL2

```
sudo apt-get install libsdl2-dev libsdl2-image-dev
libsdl2-mixer-dev libsdl2-ttf-dev
```

Librerías de ffmpeg

```
sudo apt-get install libavutil-dev libavcodec-dev
libavformat-dev libswscale-dev
```

Librerías de lua

```
sudo apt-get liblua5.3 liblua5.3-dev
```

Make

```
sudo apt-get install build-essential
```

CMake

```
sudo apt-get install cmake
```

git

```
sudo apt-get install git
```

Una vez instaladas y/o actualizadas las aplicaciones anteriores, se debe obtener el código fuente. Esto se puede realizar de dos maneras: mediante `git` o descargando los archivos fuente desde la página del repositorio que se encuentra en la portada. Para obtener el código mediante `git`, crear un directorio y ejecutar en una terminal:

Para obtener los fuentes

```
git clone https://github.com/mateoicalvo/micromachines
```

Para instalar tanto el cliente como el servidor, ejecutar:

Instalación

```
mkdir build && cd build
cmake ..
make -j<N>install
```

En la carpeta`/home/user/micromachines` resultarán dos ejecutables, `Servidor` para el servidor y `Cliente` para el cliente.

3.2. Configuración

En la carpeta instalada en `/home/user/micromachines`, se encuentra la carpeta `config/`. Allí se manejan las configuraciones tanto para el servidor como para el cliente. Se pueden manejar los puertos, opciones de maniobrabilidad, scripts de lua, etcétera.
En el directorio instalado, se encontrarán las grabaciones hechas dentro del juego, por lo cuál también se puede configurar el formato y calidad de video deseado. Es importante destacar que debido a la complejidad en la grabación de video, se debe tener cuidado al seleccionar la calidad de grabación. Por otro lado, la resolución de grabación debe coincidir con la resolución elegida para ejecutar el juego.

Los parámetros físicos de los vehículos pueden modificarse con valores entre 0 y 100. Si bien no se provee un editor de escenarios, se puede editar manualmente el archivo json de la pista para modificar la misma. Se recomienda para ello utilizar la aplicación *Tiled*. Los nombres de los parámetros son en general bastante autodescriptivos, por lo que puede experimentarse cambiando sus valores siempre y cuando tengan sentido.

3.3. Forma de uso

Luego de realizar la instalación, desde la carpeta `/home/user/micromachines` ejecutar en una terminal:

Ejecución del servidor

```
./Servidor
```

y en una terminal nueva, en el mismo directorio:

Ejecución del cliente

```
./Cliente
```

3.3.1. Tutorial

Una vez dentro del juego, presionar el boton Jugar, que enviara al usuario a la escena de salas de partida.

Ahi puede crear una sala, unirse a una existente, y ver las demas partidas. En este caso seleccionaremos la partida que se encuentra creada por el Jugador 1.

Se puede jugar de dos maneras: manualmente o de manera automática, si se selecciona el botón circual CPU.

La partida comienza cuando todos los Jugadores hayan iniciado partida. Esto se denotará en color verde.

3.3.2. Comandos

- **A:** Acelerar.
- **Z:** Frenar.
- **⇒:** doblar a la derecha
- **⇐:** doblar a la izquierda
- **G:** Grabar.
- **F11:** Pantalla completa.
- **Esc:** Volver.

Apéndices

A. Enunciado

A continuación se encuentra anexado el enunciado del trabajo práctico, que contiene una descripción detallada del software a implementar.

Introducción

El presente trabajo consiste en una implementación multijugador en línea de un juego clásico de carreras: Micromachines
Esta variante del juego será implementada en 2D en un modo multijugador, y permitirá la incorporación de "mods" del lado del servidor y scripts del lado del cliente.
El juego deberá poder grabar un video de las partidas utilizando la biblioteca ffmpeg

Descripción

El juego consiste en una pista rodeada de pasto por la que los jugadores conducirán, esquivando obstaculos y utilizando bonificaciones ("powerups") para poder llegar a la meta antes que sus adversarios.
Gana la partida el jugador que complete primero un número determinado de vueltas alrededor de la pista, y la partida finaliza cuando todos los participantes activos completan ese numero de vueltas.

Autos

Los jugadores corren en autos de carrera que poseen una determinada máxima velocidad, aceleración, maniobrabilidad (respuesta al volante), agarre (a mayor agarre menor inercia del auto), y salud.
Los autos aceleran hasta llegar a su máxima velocidad, y pueden recibir órdenes de girar a la derecha o izquierda. El ángulo de giro del auto depende de su maniobrabilidad. Si un auto tiene poco agarre tenderá a mantener su cantidad de movimiento.

Cuando un auto llega a salud 0, explota y vuelve a aparecer en el centro de la pista más cercana, mostrando una animación para la explosión

Debe poder visualizarse el daño que posee el automovil (por ejemplo con una barra de vida). También, para darle más realismo a la animación, el auto debe estar animado como si estuviera vibrando por los motores.

Colisiones

Los autos pueden colisionar entre si, desviando uno al otro y provocándose daño.

Modificadores

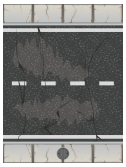
La tribuna a veces arroja ayuda a sus competidores favoritos. Estas ayudas pueden ser:

- Cajas de salud: Recuperan puntos de salud de los autos
- Boosts: elevan la velocidad máxima durante algunos segundos

Otras veces arrojan obstáculos para expresar su enojo en el evento. Estos obstáculos pueden ser:

- Piedras: Reducen la velocidad de los autos y parte de su salud.
- Aceite: Reducen el agarre de los autos.
- Barro: Reducen la visibilidad del jugador, reproduciendo una mancha de barro en la pantalla del cliente que afecta su visión en el campo.

Pista



La pista tiene cualquier forma con curvas y contracurvas pero es una sola pista continua, sin bifurcaciones ni intersecciones.
Obviamente la pista es cerrada: el principio de ella coincide con su fin.
Las pistas, asfaltadas, están rodeadas de pasto y tierra. Si el auto se va del pavimento, este reduce su velocidad a la mitad en un lapso de 500ms, y si se aleja más una determinada distancia de la pista, pierde todos sus puntos de vida y vuelve a aparecer en la misma altura de la pista pero en el centro del asfalto.

Cámara

La cámara muestra una porción de la pista (los escenarios pueden ser muy grandes y no entrar en la vista de la cámara) y debe enfocarse en el jugador y seguirlo a medida que se desplaza.

Sonidos

Como todo juego se debe reproducir sonidos para darle realismo a los eventos y acciones que suceden[5]:

- Cuando algún auto que está en pantalla está en movimiento.
- Cuando hay un choque.
- Cuando hay una frenada.

Los sonidos deben reproducirse solo si el evento es visible por el jugador.

Si la cantidad de eventos que suceden es muy grande, algunos sonidos pueden ser evitados para no saturar al jugador..

Musica ambiente

El juego debe reproducir una música ambiente, con un volumen relativamente bajo.

Interfaz del jugador

El juego debe poder dibujarse en pantalla completa y en modo ventana con el tamaño de esta configurable.
Cada jugador tendrá un color asociado de tal forma que se puedan distinguir los distintos autos.

Se debe mostrar el podio luego de cada carrera.

Aplicaciones Requeridas

Cliente

Se deberá implementar un cliente gráfico para que el usuario pueda conectarse al servidor, crear o unirse a una partida eligiendo el escenario a jugar.

Con la aplicación cliente el jugador podrá además de jugar iniciar o frenar la grabación en video de la partida actual usando ffmpeg.

Servidor

Se deberá implementar un servidor con soporte de múltiples partidas en simultáneo. Deberá poder indicarle a los clientes que se conecta qué escenarios hay disponibles así como también que partidas ya están creadas y están disponibles para que el usuario pueda unirse a alguna de ellas.

Todos los atributos del juego (velocidad máxima, aceleración, etc) deben ser configurables por archivo.

> *Es importante que todos los parámetros sean configurables: permite que se ajusten para tener un juego más balanceado y divertido a la vez que le permite a los docentes realizar pruebas.*

Autos controlados por computadora

El jugador tendrá la oportunidad de decidir si juega él o la computadora al momento de elegir una partida. En caso de elegir que juegue la computadora, elegirá un script escrito en lenguaje *Lua*[11] y lo cargará.
El script debe definir al menos una función que recibirá al menos el mapa y la posición del auto. La función deberá retornar qué acción debe realizar el auto que controla (acelerar, doblar, frenar, etc).

El cliente está escrito en C++ y para poder ejecutar un script en Lua deberá integrar un intérprete de Lua en él.
Entonces el cliente debe llamar esta función en Lua en cada iteración, como si procesara eventos del usuario.

Como es probable que la computadora sea más rápida que el humano, el cliente no deberá llamar a la función más de 10 veces por segundo.

Mods

Los mods modifican la lógica del juego de forma dinámica que se cargan en runtime en el servidor y no requieren recompilación. También son conocidos como plugins.

Estos mods o plugins alterarán la lógica de una manera muy específica: definirán una función que recibirá al menos una lista de autos y el mapa y modificará sus valores.

El servidor deberá llamar a estas funciones cada cierto tiempo.

Por ejemplo, un mod podría de forma aleatoria hacer que el último auto (el que está último en la carrera) reciba un boost de velocidad por cierto tiempo.

Otro ejemplo, si un auto está dañado su velocidad máxima se decrementa.

Otro ejemplo, un mod podría modificar ligeramente el mapa en runtime agregando o sacando zonas de barro o aceite.

Distribución de Tareas Propuesta

Con el objetivo de organizar el desarrollo de las tareas y distribuir la carga de trabajo, es necesario planificar las actividades y sus responsables durante la ejecución del proyecto. La siguiente tabla plantea una posible división de tareas de alto nivel que puede ser tomada como punto de partida para la planificación final del trabajo:

	Alumno 1 **Servidor - Modelo**	**Alumno 2** **Cliente - Modelo**	**Alumno 3** **Bibliotecas / scripts**
Semana 1 (08/10/2019)	- Draft del modelo (incluyendo lógica del juego y partidas multijugador)	- Mostrar una imagen. - Mostrar una animación. - Mostrar ambas en un lugar fijo o desplazándose por la pantalla (movimiento).	Prueba de concepto de LUA Prueba de concepto de bibliotecas dinámicas
Semana 2 (15/10/2019)	- Pista, sus exteriores y autos moviéndose libremente	- Renderizado del escenario incluyendo la cámara.	- Modelo de plugins en el servidor
Semana 3 (22/10/2019)	- Autos interactuando con la pista y otros elementos dinámicos.	- Animación de los elementos dinámicos.	- Modelo de scripts en el cliente.
Semana 4 (29/10/2019)	- Servidor multipartidas con partidas multijugador. Condiciones de victoria y derrota.	- Comunicación con el servidor. - Pantalla de conexión	- Scripts en lua para manejar los vehículos
Semana 5 (05/11/2019)	- Incorporación de plugins	- Incorporación de scripts	- Captura de video de una partida: inicio y frenado a voluntad.
Semana 6 (12/11/2019)	- Testing - Correcciones y *tuning* del Servidor - Documentación	- Testing - Correcciones y *tuning* del Cliente - Documentación	- Testing - Correcciones y *tuning* del Editor - Documentación
Entrega el 12/11/2019			
Semana 7 (19/11/2019)	- Testing y corrección de bugs - Documentación	- Testing y corrección de bugs - Documentación	- Testing y corrección de bugs - Documentación
Semana 8 (26/11/2019)	- Testing - Correcciones sobre la primer entrega - Armado del entregable	- Testing - Correcciones sobre primer entrega - Armado del entregable	- Testing - Correcciones sobre primer entrega - Armado del entregable
Reentrega el 26/11/2019			

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema se debe realizar en C++11 utilizando librerías *gtkmm, SDL y/o Qt*.
2. Los archivos de configuración deben ser almacenados en formato *YAML* [2] o *JSON*[1]. A tal fin, y con el objetivo de minimizar tiempos y posibles errores, se permiten distintas librerías externas (consultar sitio de la cátedra). No está permitido utilizar una implementación propia de lectura y escritura de *YAML/JSON*.
3. Para la simulación de la física del juego se puede utilizar el framework *Box2D* [2].
4. Para la grabación del video de las partidas se debe usar *ffmpeg* [3].
5. Es condición necesaria para la aprobación del trabajo práctico la entrega de la documentación mínima exigida (consultar sitio de la cátedra). Es importante recordar que cualquier elemento faltante o de dudosa calidad pone en riesgo la aprobación del ejercicio.
6. Entrega de uno o varios escenarios con la suficiente diversidad de elementos a tal fin que sea fácil mostrar las funcionalidades implementadas.
7. De forma opcional, se sugiere la utilización de alguna librería del estilo xUnit [7]. Si bien existen varias librerías disponibles en lenguaje C++ [8], se recomienda optar por CxxTest [9] o CppUnit [10].

Referencias

[1] Micromachines: https://en.wikipedia.org/wiki/Micro_Machines
[2] YAML: https://es.wikipedia.org/wiki/YAML
[3] JSON: https://es.wikipedia.org/wiki/JSON
[4] ffmpeg: https://ffmpeg.org/
[5] Sprites: https://opengameart.org/content/2d-race-cars
https://opengameart.org/content/race-track-tile-set
https://www.shutterstock.com/es/search/dust+animation
[6] Efectos y música ambiente: https://www.youtube.com/watch?v=Lp2SSiNu1qE
[7] Frameworks XUnit: http://en.wikipedia.org/wiki/XUnit
[8] Variantes XUnit para C/C++: http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#C.2B.2B
[9] CxxTest: http://cxxtest.com/
[10] CppUnit: http://sourceforge.net/apps/mediawiki/cppunit/index.php?title=Main_Page
[11] Lua: https://www.lua.org/

B. Código Fuente

```cpp
#include "includes/servidor/utils/ConfigServidor.h"

#include <fstream>

ConfigServidor& ConfigServidor::instancia() {
    static ConfigServidor config_(RUTA_CONFIG_SERVIDOR);
    return config_;
}

ConfigServidor::ConfigServidor(const std::string& rutaArchivo) {
    std::ifstream archivo(rutaArchivo);
    archivo >> json_;
    tilesTierra_ = json_["tiles"]["tilesTierra"].get<std::vector<int>>();
    tilesPista_ = json_["tiles"]["tilesPista"].get<std::vector<int>>();
}

std::string ConfigServidor::hostServidor() {
    return json_["red"]["host"].get<std::string>();
}

std::string ConfigServidor::puertoServidor() {
    return json_["red"]["puerto"].get<std::string>();
}

unsigned int ConfigServidor::maxClientesEnEspera() {
    return json_["red"]["maxClientesEnEspera"].get<unsigned int>();
}

unsigned int ConfigServidor::snapshotsEnviadosPorSegundo() {
    return json_["red"]["snapshotsEnviadosPorSegundo"].get<unsigned int>();
}

std::string ConfigServidor::rutaPistas() {
    return json_["rutaPistas"].get<std::string>();
}

std::vector<int>& ConfigServidor::tilesTierra() {
    return tilesTierra_;
}

std::vector<int>& ConfigServidor::tilesPista() {
    return tilesPista_;
}

int ConfigServidor::tileArena() {
    return json_["tiles"]["arena"].get<int>();
}

int ConfigServidor::tileBarro() {
    return json_["tiles"]["barro"].get<int>();
}

int ConfigServidor::tileAceite() {
    return json_["tiles"]["aceite"].get<int>();
}

int ConfigServidor::tileVacio() {
    return json_["tiles"]["vacio"].get<int>();
}

uint32_t ConfigServidor::simulacionesPorSegundo() {
    return json_["fisicas"]["simulacionesPorSegundo"].get<uint32_t>();
}

uint32_t ConfigServidor::iteracionesPosicion() {
    return json_["fisicas"]["iteracionesPosicion"].get<uint32_t>();
```



```cpp
}

uint32_t ConfigServidor::iteracionesVelocidad() {
    return json_["fisicas"]["iteracionesVelocidad"].get<uint32_t>();
}

unsigned int ConfigServidor::velocidadMaxVehiculoAdelante() {
    return json_["modelo"]["vehiculo"]["velocidadMaxAdelante"].get<unsigned int>();
}
unsigned int ConfigServidor::velocidadMaxVehiculoAtras() {
    return json_["modelo"]["vehiculo"]["velocidadMaxAtras"].get<unsigned int>();
}
unsigned int ConfigServidor::aceleracionVehiculo() {
    return json_["modelo"]["vehiculo"]["aceleracionVehiculo"].get<unsigned int>();
}
unsigned int ConfigServidor::maniobrabilidadVehiculo() {
    return json_["modelo"]["vehiculo"]["maniobrabilidadVehiculo"].get<unsigned int>();
}
unsigned int ConfigServidor::agarreVehiculo() {
    return json_["modelo"]["vehiculo"]["agarreVehiculo"].get<unsigned int>();
}
unsigned int ConfigServidor::saludVehiculo() {
    return json_["modelo"]["vehiculo"]["saludVehiculo"].get<unsigned int>();
}

float ConfigServidor::anchoVehiculo() {
    return json_["modelo"]["vehiculo"]["ancho"].get<float>();
}

float ConfigServidor::largoVehiculo() {
    return json_["modelo"]["vehiculo"]["largo"].get<float>();
}

float ConfigServidor::ladoSuperficie() {
    return json_["modelo"]["superficies"]["lado"].get<float>();
}

float ConfigServidor::anchoTile() {
    return json_["modelo"]["anchoTile"].get<float>();
}

uint8_t ConfigServidor::disminucionVidaChoqueConVehiculo() {
    return json_["modelo"]["disminucionVida"]["vsVehiculo"].get<uint8_t>();
}

int ConfigServidor::cantidadMaximaModificadores() {
    return json_["modelo"]["modificadores"]["cantidadMaxima"].get<int>();
}

int ConfigServidor::factorAparicionModificador() {
    return json_["modelo"]["modificadores"]["factorAparicion"].get<int>();
}
```

nov 26, 19 17:34 **Servidor.cpp** Page 1/1

```cpp
#include "includes/servidor/Servidor.h"

#include <iostream>

Servidor::Servidor(const std::string& unHost, const std::string& puerto) :
    salaDeEspera_(eventosRecibidos_),
    hiloAceptador_(unHost, puerto, salaDeEspera_),
    distribuidorEventos_(eventosRecibidos_, salaDeEspera_, coordinadorPartidas_)
,
    coordinadorPartidas_(salaDeEspera_) {
}

void Servidor::correr() {
    hiloAceptador_.iniciar();
    distribuidorEventos_.iniciar();
    char c;
    while ((c = std::cin.get()) ≠ CARACTER_SALIR) {
        // pass
    }
}

void Servidor::cerrar() {
    eventosRecibidos_.detener();

    hiloAceptador_.detener();
    hiloAceptador_.join();

    distribuidorEventos_.detener();
    distribuidorEventos_.join();
}
```

nov 26, 19 17:34 **SalaDeEspera.cpp** Page 1/1

```cpp
#include "includes/servidor/SalaDeEspera.h"

SalaDeEspera::SalaDeEspera(ColaBloqueante<std::shared_ptr<Evento>>& destinoEvent
os) :
    contadorJugadores_(0),
    destinoEventos_(destinoEventos) {
}

SalaDeEspera::~SalaDeEspera() {

}

void SalaDeEspera::agregarJugador(SocketTCP∧ socket) {
    std::lock_guard<std::mutex> lck(mtx_);
    contadorJugadores_++;
    jugadores_[contadorJugadores_] = std::make_shared<Jugador>(std::move(socket)
, contadorJugadores_, destinoEventos_);
}

void SalaDeEspera::agregarJugador(std::shared_ptr<Jugador> unJugador) {
    std::lock_guard<std::mutex> lck(mtx_);
    jugadores_[unJugador→uuid()] = unJugador;
}

std::shared_ptr<Jugador> SalaDeEspera::quitarJugador(uint32_t uuidJugador) {
    std::lock_guard<std::mutex> lck(mtx_);
    std::shared_ptr<Jugador> jugador = jugadores_.at(uuidJugador);
    jugadores_.erase(uuidJugador);
    return jugador;
}

std::shared_ptr<Jugador> SalaDeEspera::getJugador(uint32_t uuidJugador) {
    std::lock_guard<std::mutex> lck(mtx_);
    return jugadores_.at(uuidJugador);
}

void SalaDeEspera::ocurrio(std::shared_ptr<Evento> unEvento) {
    for (const auto& kv : jugadores_) {
        kv.second→ocurrio(unEvento);
    }
}

void SalaDeEspera::manejar(Evento& e) {
    e.actualizar(*this);
}

void SalaDeEspera::manejar(EventoDesconexion& e) {
    //FIXME: Mejorar esta lÃ³gica
    std::lock_guard<std::mutex> lck(mtx_);
    jugadores_.erase(e.uuidRemitente());
}
```

nov 26, 19 17:34 **SocketTCPServidor.cpp** Page 1/1

```cpp
#include "includes/servidor/red/SocketTCPServidor.h"

#include <stdexcept>

SocketTCPServidor::SocketTCPServidor(const std::string& unHost, const std::strin
g& unPuerto) :
    SocketTCP(unHost, unPuerto) {
}

void SocketTCPServidor::enlazar() {
    int opt_val = 1;
    int result_set_opt = setsockopt(fileDescripor_, SOL_SOCKET, \
        SO_REUSEADDR, &opt_val, sizeof(opt_val));
    if (result_set_opt ≡ -1) {
        throw std::runtime_error(ERROR_SET_SOCK_OPT);
    }
    int status = -1;
    bool ok = false;
    for (struct addrinfo* rp = hints_; rp ≠ NULL; rp = rp→ai_next) {
        status = bind(fileDescripor_, rp→ai_addr, rp→ai_addrlen);
        if (status ≡ 0) {
            ok = true;
            break;
        }
    }
    if (¬ok) {
        throw std::runtime_error(ERROR_BIND);
    }
}

void SocketTCPServidor::escuchar(unsigned int maxEnEspera) {
    int estado = listen(fileDescripor_, maxEnEspera);
    if (estado ≡ -1) {
        throw std::runtime_error(ERROR_LISTEN);
    }
}

SocketTCP SocketTCPServidor::aceptar() {
    int fdAceptado = accept(fileDescripor_, NULL, NULL);
    if (fdAceptado ≡ -1) {
        throw std::runtime_error(ERROR_ACEPTAR);
    }
    return std::move(SocketTCP(fdAceptado));
}
```

nov 26, 19 17:34 **Partida.cpp** Page 1/3

```cpp
#include "includes/servidor/Partida.h"

#include <cmath>

#include "includes/common/Cronometro.h"
#include "includes/common/Cola.h"
#include "includes/servidor/utils/ConfigServidor.h"
#include "includes/servidor/SalaDeEspera.h"
#include "includes/common/eventos/EventoPartidaIniciada.h"

Partida::Partida(uint16_t uuidPista, SalaDeEspera& salaDeEspera) :
    mundo_(uuidPista),
    salaDeEspera_(salaDeEspera),
    fueIniciada_(false) {
}

Partida::~Partida() {
}

void Partida::agregarJugador(std::shared_ptr<Jugador> jugador) {
    jugadores_[jugador→uuid()] = jugador;
    uuidJugadorAEstaListo_.emplace(jugador→uuid(), false);
}

bool Partida::todosListos() {
    for (auto& kv : uuidJugadorAEstaListo_) {
        if (kv.second ≡ false) {
            return false;
        }
    }
    return true;
}

void Partida::marcarListo(uint32_t uuidJugador) {
    //Nunca deberÃ-a pinchar porque el jugador fue agregado
    uuidJugadorAEstaListo_.at(uuidJugador) = true;
}

bool Partida::estaListo(uint32_t uuidJugador) {
    return uuidJugadorAEstaListo_.at(uuidJugador);
}

//TODO: CONST
std::map<uint32_t, std::shared_ptr<Jugador>>& Partida::jugadores() {
    return jugadores_;
}

void Partida::step(uint32_t nroIteracion) {
    bool obtenido = false;
    std::shared_ptr<Evento> evento;
    while((obtenido = eventosEntrantes_.get(evento))) {
        mundo_.manejar(*evento);
    }
    mundo_.step(nroIteracion);
    Cola<std::shared_ptr<Evento>>& eventosOcurridos = mundo_.eventosOcurridos();
    std::shared_ptr<Evento> eventoOcurrido;
    while((obtenido = eventosOcurridos.get(eventoOcurrido))) {
        for (auto& kv : jugadores_) {
            kv.second→ocurrio(eventoOcurrido);
        }
        //TODO: Manejar el evento, acÃ¡ me entero del fin partida
        manejar(*eventoOcurrido);
    }
}

void Partida::correr() {
```

```
67      fueIniciada_ = true;
68      asignarVehiculos();
69      //TODO: Asignar un auto a cada jugador presente, no poner autos vacios
70      double frecuencia = (double)1 / (double)CONFIG_SERVIDOR.simulacionesPorSegun
    do();
71      // Convierto a milisegundos
72      // TODO: Uniformizar esto, porque depende de como se usa aca, en el cronomet
    ro
73      // y en el dormir
74      frecuencia *= 1000;
75      Cronometro c;
76      double t1 = c.ahora();
77      uint32_t iteracion = 1;
78      while(seguirCorriendo_) {
79          step(iteracion);
80          double t2 = c.ahora();
81          double resto = frecuencia - (t2 - t1);
82          if (resto < 0) {
83              double atraso = -resto;
84              double perdidos = atraso - std::fmod(atraso, frecuencia);
85              resto = frecuencia - std::fmod(atraso, frecuencia);
86              t1 += perdidos;
87              iteracion += std::floor(perdidos / frecuencia);
88          }
89          dormir(resto);
90          t1 += frecuencia;
91          iteracion += 1;
92      }
93  }
94
95  void Partida::detener() {
96      seguirCorriendo_ = false;
97  }
98
99  void Partida::manejar(Evento& e) {
100     e.actualizar(*this);
101 }
102
103 void Partida::manejar(EventoFinCarrera& e) {
104     //FIXME: Ojo que el contenedor de jugadores ahora tiene que ser protegido.
105     detener();
106     for (auto& kv : jugadores_) {
107         salaDeEspera_.agregarJugador(kv.second);
108     }
109     jugadores_.clear();
110 }
111
112 void Partida::ocurrio(std::shared_ptr<Evento> unEvento) {
113     eventosEntrantes_.put(unEvento);
114 }
115
116 void Partida::asignarVehiculos() {
117     std::map <uint32_t, uint8_t> jugadoresAVehiculos;
118
119     for (const auto& kv : jugadores_) {
120         uint8_t idVehiculo = mundo_.agregarVehiculo(kv.second);
121         jugadoresAVehiculos.emplace(kv.first, idVehiculo);
122     }
123     for (const auto& kv : jugadores_) {
124         uint8_t idVehiculo = jugadoresAVehiculos.at(kv.first);
125         std::map<uint8_t, datosVehiculo_> estadoInicial = mundo_.getEstadoInicia
    l();
126         std::shared_ptr<Evento> eventoInicial = std::make_shared<EventoPartidaIn
    iciada>(idVehiculo, std::move(estadoInicial));
127         kv.second→ocurrio(eventoInicial);
128     }
```



```
129 }
130
131 bool Partida::aceptaJugadores() {
132     return ¬fueIniciada_;
133 }
```

nov 26, 19 17:34 **SuperficieTierra.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/superficies/SuperficieTierra.h"

int SuperficieTierra::getTipo() {
    return SUPERFICIE_TIERRA_;
}
```

nov 26, 19 17:34 **SuperficiePista.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/superficies/SuperficiePista.h"

int SuperficiePista::getTipo() {
    return SUPERFICIE_PISTA_;
}
```

nov 26, 19 17:34 **SuperficieFactory.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/superficies/SuperficieFactory.h"

#include <algorithm>

#include "includes/servidor/utils/ConfigServidor.h"

std::shared_ptr<Superficie> SuperficieFactory::instanciar(int uuid) {

    std::vector<int>& tilesTierra = CONFIG_SERVIDOR.tilesTierra();
    std::vector<int>& tilesPista = CONFIG_SERVIDOR.tilesPista();

    int tileArena = CONFIG_SERVIDOR.tileArena();

    std::vector<int>::iterator it;

    it = std::find(tilesTierra.begin(), tilesTierra.end(), uuid);
    if (it ≠ tilesTierra.end()) {
        return std::make_shared<SuperficieTierra>();
    }

    it = std::find(tilesPista.begin(), tilesPista.end(), uuid);
    if (it ≠ tilesPista.end()) {
        return std::make_shared<SuperficiePista>();
    }

    if (uuid ≡ tileArena) {
        return std::make_shared<SuperficieArena>();
    }

    throw std::runtime_error("No hay datos de superficie en el mapa");
}
```

nov 26, 19 17:34 **SuperficieArena.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/superficies/SuperficieArena.h"

int SuperficieArena::getTipo() {
    return SUPERFICIE_ARENA_;
}
```

```cpp
#include "includes/servidor/modelo/Mundo.h"

#include <string>
#include <fstream>
#include <map>
#include <vector>
#include <stdlib.h>


#include "includes/servidor/utils/ConfigServidor.h"
#include "includes/3rd−party/jsoncpp/json.hpp"
#include "includes/common/Tile.h"
#include "includes/common/Conversor.h"
#include "includes/servidor/modelo/superficies/SuperficieFactory.h"
#include "includes/servidor/modelo/movimiento/Posicion.h"
#include "includes/servidor/modelo/entidades/CajaVida.h"
#include "includes/servidor/modelo/entidades/Barro.h"
#include "includes/servidor/modelo/entidades/Boost.h"
#include "includes/servidor/modelo/entidades/Aceite.h"
#include "includes/servidor/modelo/entidades/Piedra.h"
#include "includes/common/eventos/EventoSnapshot.h"

//TODO: Crear conversor de coordenadas?
//Forward declaration
static void cargarSuelo(uint16_t largoX, uint16_t largoY, std::map<Tile, std::shared_ptr<Superficie>>& tilesASuelo, std::vector<Tile>& tilesConPista, Json& pistaJson);
static void cargarPosicionesIniciales(uint16_t largoX, uint16_t largoY, std::queue<Posicion>& tiles, Json& pistaJson);

Mundo::Mundo(uint16_t uuidPista) :
    fisicas_(eventosOcurridos_, contactListener_, *this),
    snapshotsEnviadosPorSegundo_(60/CONFIG_SERVIDOR.snapshotsEnviadosPorSegundo()),
    contactListener_(fisicas_),
    carrera_(eventosOcurridos_) {

    for (uint8_t id = 1; id < 255; ++id) {
        uuidsObjetos_.push(id);
    }

    //TODO: Es mejor cargar todas las pistas al inicio y luego hacer un get() para no tener que ir
    // siempre a disco.
    std::string rutaPista = CONFIG_SERVIDOR.rutaPistas() + std::to_string(uuidPista) + ".json";
    std::ifstream archivoPista(rutaPista);
    Json pistaJson;
    archivoPista >> pistaJson;
    uint16_t largoX = pistaJson["dimensiones"]["x"].get<uint16_t>();
    uint16_t largoY = pistaJson["dimensiones"]["y"].get<uint16_t>();

    cargarSuelo(largoX, largoY, tileASuelo_, tilesConPista_, pistaJson);
    cargarPosicionesIniciales(largoX, largoY, posicionesIniciales_, pistaJson);
    carrera_.cargarDesdeJson(pistaJson);

    fisicas_.generarSuelo(tileASuelo_);
    fisicas_.generarCheckpoints(carrera_.checkpoints());
    srand(time(NULL));
}

Mundo::~Mundo() {
    //TODO: IMPLEMENTAR
}

void Mundo::step(uint32_t numeroIteracion) {
```



```cpp
    fisicas_.step(numeroIteracion);
    for (auto& kv : jugadoresAVehiculos_) {
        kv.second.step();
    }
    //TODO: Chequear por la negativa?
    //FIXME: NO DEBIERA ESTAR LIGADO AL STEP DEL MUNDO, PERO MUNDO TENDRÃ−^MA QUE SER MONITOR
    if((numeroIteracion % snapshotsEnviadosPorSegundo_) ≡ 0) {
        std::map<uint8_t, datosVehiculo_> idsADatosVehiculo = serializarEstado();
        std::shared_ptr<Evento> snapshot = std::make_shared<EventoSnapshot>(std::move(idsADatosVehiculo));
        eventosOcurridos_.put(snapshot);
    }
    agregarModificadores(numeroIteracion);
}

Cola<std::shared_ptr<Evento>>& Mundo::eventosOcurridos() {
    return eventosOcurridos_;
}

void Mundo::recuperarUuid(uint8_t uuid) {
    uuidsObjetos_.push(uuid);
}

uint8_t Mundo::agregarVehiculo(std::shared_ptr<Jugador> unJugador) {
    //TODO: En cual de los casilleros?
    //FIXME: Nada impide top() de pila vacia si hay mas jugadores
    Posicion posicion = posicionesIniciales_.front();
    posicion.x_ = Conversor::tileAMetro(posicion.x_);
    posicion.y_ = Conversor::tileAMetro(posicion.y_);
    uint8_t uuid = uuidsObjetos_.front();
    jugadoresAVehiculos_.emplace(unJugador→uuid(), Vehiculo(uuid,
            CONFIG_SERVIDOR.velocidadMaxVehiculoAdelante(),
            CONFIG_SERVIDOR.velocidadMaxVehiculoAtras(),
            CONFIG_SERVIDOR.aceleracionVehiculo(),
            CONFIG_SERVIDOR.maniobrabilidadVehiculo(),
            CONFIG_SERVIDOR.agarreVehiculo(),
            CONFIG_SERVIDOR.saludVehiculo(),
            posicion,
            unJugador));

    jugadoresAIDVehiculo_[unJugador→uuid()] = uuid;
    fisicas_.agregarVehiculo(jugadoresAVehiculos_.at(unJugador→uuid()), posicion);
    posicionesIniciales_.pop();

    carrera_.registrarVehiculo(jugadoresAVehiculos_.at(unJugador→uuid()));

    uuidsObjetos_.pop();
    return uuid;
}

std::map<uint8_t, datosVehiculo_> Mundo::getEstadoInicial() {
    //FIXME No devuelve el estado inicial en llamadas sucesivas
    return serializarEstado();
}

void Mundo::agregarModificadores(uint32_t nroIteracion) {
    if (nroIteracion % CONFIG_SERVIDOR.factorAparicionModificador() ≠ 0) {
        return;
    }
    //SORTEAR EL QUE VA A APARECER
    if (uuidsObjetos_.size() ≡ 0) {
        return;
    }
```

```
123     int tile = rand() % tilesConPista_.size();
124     Tile& destino = tilesConPista_[tile];
125     Posicion posicion(Conversor::tileAMetro(destino.x_) + 0.5f*CONFIG_SERVIDOR.a
    nchoTile(),
126             Conversor::tileAMetro(destino.y_) + 0.5f*CONFIG_SERVIDOR.anchoTile(), 0)
    ;
127     uint8_t uuid = uuidsObjetos_.front();
128
129     int modificador = rand() % 5 + 1;
130     //TODO: REFACTOIRZAR ESTE HORROR
131     if (modificador ≡ UUID_VIDA) {
132         //TODO: AL CONFIG SERVIDOR LOS 20 DE VIDA
133         modificadores_.emplace(uuid, std::make_shared<CajaVida>(uuid, 20));
134         fisicas_.agregarModificador(modificadores_.at(uuid), UUID_VIDA, posicion
    );
135     } else if (modificador ≡ UUID_BARRO) {
136         modificadores_.emplace(uuid, std::make_shared<Barro>(uuid));
137         fisicas_.agregarModificador(modificadores_.at(uuid), UUID_BARRO, posicio
    n);
138     } else if (modificador ≡ UUID_PIEDRA) {
139         modificadores_.emplace(uuid, std::make_shared<Piedra>(uuid));
140         fisicas_.agregarModificador(modificadores_.at(uuid), UUID_PIEDRA, posici
    on);
141     } else if (modificador ≡ UUID_ACEITE) {
142         modificadores_.emplace(uuid, std::make_shared<Aceite>(uuid));
143         fisicas_.agregarModificador(modificadores_.at(uuid), UUID_ACEITE, posici
    on);
144     } else if (modificador ≡ UUID_BOOST) {
145         modificadores_.emplace(uuid, std::make_shared<Boost>(uuid));
146         fisicas_.agregarModificador(modificadores_.at(uuid), UUID_BOOST, posicio
    n);
147     }
148     uuidsObjetos_.pop();
149 }
150
151 void Mundo::manejar(Evento& e) {
152     e.actualizar(*this);
153 }
154
155 void Mundo::manejar(EventoAcelerar& e) {
156     uint32_t jugador = e.uuidRemitente();
157     fisicas_.acelerar(jugadoresAIDVehiculo_[jugador]);
158 }
159
160 void Mundo::manejar(EventoDesacelerar& e) {
161     uint32_t jugador = e.uuidRemitente();
162     fisicas_.desacelerar(jugadoresAIDVehiculo_[jugador]);
163 }
164
165 void Mundo::manejar(EventoFrenar& e) {
166     uint32_t jugador = e.uuidRemitente();
167     fisicas_.frenar(jugadoresAIDVehiculo_[jugador]);
168     Posicion p = fisicas_.getPosicionDe(jugadoresAIDVehiculo_.at(e.uuidRemitente
    ()));
169     std::shared_ptr<Evento> frenada = std::make_shared<EventoFrenada>(p.x_, p.y_
    );
170     eventosOcurridos_.put(frenada);
171 }
172
173 void Mundo::manejar(EventoDejarDeFrenar& e) {
174     uint32_t jugador = e.uuidRemitente();
175     fisicas_.dejarDeFrenar(jugadoresAIDVehiculo_[jugador]);
176 }
177
178 void Mundo::manejar(EventoDoblarIzquierda& e) {
179     uint32_t jugador = e.uuidRemitente();
```



```
180     fisicas_.doblarIzquierda(jugadoresAIDVehiculo_[jugador]);
181 }
182
183 void Mundo::manejar(EventoDejarDeDoblarIzquierda& e) {
184     uint32_t jugador = e.uuidRemitente();
185     fisicas_.dejarDeDoblarIzquierda(jugadoresAIDVehiculo_[jugador]);
186 }
187
188 void Mundo::manejar(EventoDoblarDerecha& e) {
189     uint32_t jugador = e.uuidRemitente();
190     fisicas_.doblarDerecha(jugadoresAIDVehiculo_[jugador]);
191 }
192
193 void Mundo::manejar(EventoDejarDeDoblarDerecha& e) {
194     uint32_t jugador = e.uuidRemitente();
195     fisicas_.dejarDeDoblarDerecha(jugadoresAIDVehiculo_[jugador]);
196 }
197
198 //FIXME: No hardcodear
199 // El sistema de referencia de la pista estÃ¡ arriba a la izquierda,
200 // mientras que en el servidor estÃ¡ abajo a la derecha.
201 static void cargarSuelo(uint16_t largoX, uint16_t largoY, std::map<Tile, std::sh
    ared_ptr<Superficie>>& tilesASuelo, std::vector<Tile>& tilesConPista, Json& pist
    aJson) {
202     for (int i = 0; i < largoX; ++i) {
203         for (int j = 0; j < largoY; ++j) {
204             int uuidTerreno = pistaJson["capas"]["terreno"][std::to_string(i)][std:
    :to_string(j)].get<int>();
205             int uuidPista = pistaJson["capas"]["pista"][std::to_string(i)][std::to_
    string(j)].get<int>();
206             // Hay pista
207             if (uuidPista ≠ CONFIG_SERVIDOR.tileVacio()) {
208                 tilesASuelo[Tile(i, largoY − j − 1)] = SuperficieFactory::instan
    ciar(uuidPista);
209                 tilesConPista.emplace_back(Tile(i, largoY − j − 1));
210
211             } else {
212                 tilesASuelo[Tile(i, largoY − j − 1)] = SuperficieFactory::instan
    ciar(uuidTerreno);
213             }
214         }
215     }
216 }
217
218 static void cargarPosicionesIniciales(uint16_t largoX, uint16_t largoY, std::que
    ue<Posicion>& tiles_, Json& pistaJson) {
219     int cupos = pistaJson["posicionesIniciales"]["cantidad"].get<int>();
220     for (int i = 0; i < cupos; ++i) {
221         float x = pistaJson["posicionesIniciales"][std::to_string(i)]["x"].get<float>(
    );
222         float y = largoY − pistaJson["posicionesIniciales"][std::to_string(i)]["y"].ge
    t<float>();
223         tiles_.emplace(Posicion(x, y, 0.0f));
224     }
225 }
226
227 std::map<uint8_t, datosVehiculo_> Mundo::serializarEstado() {
228     std::map<uint8_t, datosVehiculo_> idsADatosVehiculo;
229     for (const auto& kv : jugadoresAIDVehiculo_) {
230         uint8_t idVehiculo = jugadoresAIDVehiculo_.at(kv.first);
231         //FISICAS DE FISICAS
232         Posicion posicion = fisicas_.getPosicionDe(idVehiculo);
233         //LOGICA DE MUNDO(YO)
234         uint8_t salud = jugadoresAVehiculos_.at(kv.first).salud();
235         //FIXME: No debiera ser asÃ-
236         uint8_t visible = 1;
```

nov 26, 19 17:34 **Mundo.cpp** Page 5/5

```
237             idsADatosVehiculo.emplace(idVehiculo, datosVehiculo_{
238                 posicion.x_,
239                 posicion.y_,
240                 posicion.anguloDeg_,
241                 salud,
242                 visible
243             });
244         }
245         return std::move(idsADatosVehiculo);
246     }
```

nov 26, 19 17:34 **Posicion.cpp** Page 1/1

```
1   #include "includes/servidor/modelo/movimiento/Posicion.h"
2
3   Posicion::Posicion(float x, float y, uint16_t anguloDeg)
4       : x_(x)
5       , y_(y)
6       , anguloDeg_(anguloDeg) {
7   }
```

nov 26, 19 17:34 **Identificable.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/Identificable.h"

Identificable::Identificable(uint8_t uuid) :
    UUID_(uuid) {
}

Identificable::~Identificable() {
}

uint8_t Identificable::uuid() {
    return UUID_;
}
```

nov 26, 19 17:34 **Transformacion.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/fisicas/transformaciones/Transformacion.h"

Transformacion::Transformacion(Fisicas& fisicas) :
    fisicas_(fisicas) {

}

Transformacion::~Transformacion() {

}
```

nov 26, 19 17:34 **Reubicar.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/fisicas/transformaciones/Reubicar.h"

#include "includes/3rd-party/Box2D/Box2D.h"
#include "includes/servidor/modelo/movimiento/Posicion.h"

#ifndef DEGTORAD
#define DEGTORAD 0.0174532925199432957f
#define RADTODEG 57.295779513082320876f
#endif

Reubicar::Reubicar(Fisicas& fisicas, b2Body* cuerpo, Posicion& posicion) :
    Transformacion(fisicas),
    cuerpo_(cuerpo),
    posicion_(posicion) {

}

void Reubicar::aplicar() {
    cuerpo_->SetTransform(b2Vec2(posicion_.x_, posicion_.y_), (float)posicion_.anguloDeg_*DEGTORAD);
    cuerpo_->SetLinearVelocity(b2Vec2(0, 0));
    cuerpo_->SetAngularVelocity(0.0f);
}
```

nov 26, 19 17:34 **Quitar.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/fisicas/transformaciones/Quitar.h"

#include "includes/servidor/modelo/fisicas/Fisicas.h"
#include "includes/3rd-party/Box2D/Box2D.h"
#include "includes/common/eventos/EventoDesaparecioConsumible.h"

Quitar::Quitar(Fisicas& fisicas, b2Body* cuerpo, uint8_t uuidCuerpo) :
    Transformacion(fisicas),
    cuerpo_(cuerpo),
    uuidCuerpo_(uuidCuerpo) {
}

void Quitar::aplicar() {
    cuerpo_->GetWorld()->DestroyBody(cuerpo_);
    std::shared_ptr<Evento> desaparicion = std::make_shared<EventoDesaparecioConsumible>(uuidCuerpo_);
    fisicas_.ocurrio(desaparicion);
    fisicas_.nuevoUuidDisponible(uuidCuerpo_);
}
```

```
1   #include "includes/servidor/modelo/fisicas/Fisicas.h"
2
3   #include "includes/servidor/utils/ConfigServidor.h"
4   #include "includes/servidor/modelo/Mundo.h"
5
6   #include "includes/servidor/modelo/entidades/Vehiculo.h"
7   #include "includes/servidor/modelo/entidades/CajaVida.h"
8   #include "includes/servidor/modelo/entidades/Barro.h"
9   #include "includes/servidor/modelo/entidades/Boost.h"
10  #include "includes/servidor/modelo/entidades/Aceite.h"
11  #include "includes/servidor/modelo/entidades/Piedra.h"
12  #include "includes/servidor/modelo/fisicas/transformaciones/Reubicar.h"
13  #include "includes/servidor/modelo/fisicas/transformaciones/Quitar.h"
14
15  #include "includes/common/eventos/EventoAparecioConsumible.h"
16
17
18  //TODO: Fisicas debe conocer de eventos ocurridos?
19  //Tiene pinta de que no. Por ende tampoco de snapshots por segundo
20  Fisicas::Fisicas(Cola<std::shared_ptr<Evento>>& eventosOcurridos, ContactListene
    r& contactListener, Mundo& mundo) :
21      gravedad_(0, 0),
22      mundoBox2D_(std::make_shared<b2World>(gravedad_)),
23      frecuencia_((double)1 / (double)CONFIG_SERVIDOR.simulacionesPorSegundo()),
24      iteracion_(0),
25      eventosOcurridos_(eventosOcurridos),
26      mundo_(mundo) {
27
28      mundoBox2D_→SetContactListener(&contactListener);
29  }
30
31  Fisicas::~Fisicas() {
32  }
33
34  void Fisicas::ocurrio(std::shared_ptr<Evento> unEvento) {
35      eventosOcurridos_.put(unEvento);
36  }
37
38  void Fisicas::agregarModificador(std::shared_ptr<Modificador> modificador, uint8
    _t tipo, Posicion& posicion) {
39      float ladoModificador = 5.0f;//CONFIG_SEVIDOR.ladoModificador();
40      b2BodyDef bodyDef;
41      bodyDef.userData = modificador.get();
42      float x = posicion.x_;
43      float y = posicion.y_;
44      bodyDef.position.Set(x, y);
45
46      b2Body* cuerpo = mundoBox2D_→CreateBody(&bodyDef);
47      b2PolygonShape forma;
48      forma.SetAsBox(ladoModificador/2.0f, ladoModificador/2.0f);
49    b2FixtureDef caracteristicas;
50    caracteristicas.shape = &forma;
51      caracteristicas.isSensor = true;
52    cuerpo→CreateFixture(&caracteristicas);
53      colisionables_[modificador→uuid()] = cuerpo;
54      std::shared_ptr<Evento> aparicion = std::make_shared<EventoAparecioConsumibl
    e>(modificador→uuid(), tipo, x, y);
55      eventosOcurridos_.put(aparicion);
56  }
57
58  void Fisicas::generarSuelo(std::map<Tile, std::shared_ptr<Superficie>>& tileASue
    lo) {
59      //TODO: Implementar: es arena tierra y pista.
60      float anchoTile = CONFIG_SERVIDOR.anchoTile();
61      for (const auto& kv : tileASuelo) {
62          b2BodyDef bodyDef;
```



```
63          bodyDef.userData = kv.second.get();
64
65          float x = anchoTile*(float)kv.first.x_ + 0.5f*anchoTile;
66          float y = anchoTile*(float)kv.first.y_ + 0.5f*anchoTile;
67        bodyDef.position.Set(x, y);
68
69          b2Body* cuerpo = mundoBox2D_→CreateBody(&bodyDef);
70          b2PolygonShape forma;
71          forma.SetAsBox(CONFIG_SERVIDOR.ladoSuperficie()/2.0f, CONFIG_SERVIDOR.la
    doSuperficie()/2.0f);
72        b2FixtureDef caracteristicas;
73        caracteristicas.shape = &forma;
74          caracteristicas.isSensor = true;
75        cuerpo→CreateFixture(&caracteristicas);
76      }
77  }
78
79  /*void Fisicas::generarSuperficies(std::map<Tile, std::shared_ptr<Superficie>>&
    tileASuperficie) {
80      //TODO: Implementar
81  }*/
82
83  void Fisicas::generarCheckpoints(std::map<int, Checkpoint>& checkpoints) {
84      for (auto& kv : checkpoints) {
85          b2BodyDef bodyDef;
86          //https://stackoverflow.com/questions/5377434/does-stdmapiterator-return
    -a-copy-of-value-or-a-value-itself
87          bodyDef.userData = &kv.second;
88
89          float ancho = kv.second.ancho();
90          float largo = kv.second.alto();
91
92          float x = kv.second.posicion().x_;
93          float y = kv.second.posicion().y_;
94
95          bodyDef.position.Set(x, y);
96
97          b2Body* cuerpo = mundoBox2D_→CreateBody(&bodyDef);
98          b2PolygonShape forma;
99          forma.SetAsBox(ancho/2.0f, largo/2.0f);
100       b2FixtureDef caracteristicas;
101       caracteristicas.shape = &forma;
102         caracteristicas.isSensor = true;
103       cuerpo→CreateFixture(&caracteristicas);
104     }
105 }
106 void Fisicas::acelerar(uint8_t uuidVehiculo) {
107     vehiculos_.at(uuidVehiculo)→acelerando();
108 }
109
110 void Fisicas::desacelerar(uint8_t uuidVehiculo) {
111     vehiculos_.at(uuidVehiculo)→desacelerando();
112 }
113
114 void Fisicas::frenar(uint8_t uuidVehiculo) {
115     vehiculos_.at(uuidVehiculo)→frenando();
116 }
117
118 void Fisicas::dejarDeFrenar(uint8_t uuidVehiculo) {
119     vehiculos_.at(uuidVehiculo)→dejandoDeFrenar();
120 }
121
122 void Fisicas::doblarIzquierda(uint8_t uuidVehiculo) {
123     vehiculos_.at(uuidVehiculo)→doblandoIzquierda();
124 }
125
```

```
126 void Fisicas::dejarDeDoblarIzquierda(uint8_t uuidVehiculo) {
127     vehiculos_.at(uuidVehiculo)→dejandoDeDoblarIzquierda();
128 }
129
130 void Fisicas::doblarDerecha(uint8_t uuidVehiculo) {
131     vehiculos_.at(uuidVehiculo)→doblandoDerecha();
132 }
133
134 void Fisicas::dejarDeDoblarDerecha(uint8_t uuidVehiculo) {
135     vehiculos_.at(uuidVehiculo)→dejandoDeDoblarDerecha();
136 }
137
138 void Fisicas::agregarVehiculo(Vehiculo& vehiculo, Posicion& posicion) {
139     vehiculos_.emplace(vehiculo.uuid(), std::make_shared<B2DVehiculo>(mundoBox2D
    _.get(), vehiculo));
140     b2Vec2 posicionBox2D = {posicion.x_, posicion.y_};
141     vehiculos_.at(vehiculo.uuid())→getB2D()→SetTransform(posicionBox2D, (float
    )posicion.anguloDeg_);
142 }
143
144 Posicion Fisicas::getPosicionDe(uint8_t idCuerpo) {
145
146     b2Body* cuerpoFisico = vehiculos_.at(idCuerpo)→getB2D();
147     b2Vec2 posicion = cuerpoFisico→GetPosition();
148     float32 angulo = cuerpoFisico→GetAngle();
149     int anguloDeg = (int)(angulo*RADTODEG) % 360;
150     anguloDeg *= −1;
151     if (anguloDeg < 0) {
152         anguloDeg += 360;
153     }
154     return Posicion(posicion.x, posicion.y, anguloDeg);
155 }
156
157 void Fisicas::nuevoUuidDisponible(uint8_t uuid) {
158     mundo_.recuperarUuid(uuid);
159 }
160
161 void Fisicas::step(uint32_t numeroIteracion) {
162     //TODO: Todos haran step
163     //AcÃ¡ se alteran los cuerpos fÃ-sicos.
164     for (const auto& kv : vehiculos_) {
165         kv.second→step();
166     }
167     uint32_t escala = numeroIteracion − iteracion_;
168     float tiempoAtranscurrir = (float)escala * frecuencia_;
169     mundoBox2D_→Step(tiempoAtranscurrir, CONFIG_SERVIDOR.iteracionesVelocidad()
    , CONFIG_SERVIDOR.iteracionesPosicion());
170     iteracion_ = numeroIteracion;
171     //TODO: Aplicar transformaciones y encolar eventos pertinentes.
172     while(¬transformaciones_.empty()) {
173         std::shared_ptr<Transformacion> t = transformaciones_.front();
174         t→aplicar();
175         transformaciones_.pop();
176     }
177 }
178
179 //TODO: REFACTORIZAR, TODOS HACEN LO MISMO
180 void Fisicas::reubicar(Vehiculo& vehiculo, Posicion& posicion) {
181     b2Body* cuerpoVehiculo = vehiculos_.at(vehiculo.uuid())→getB2D();
182     std::shared_ptr<Transformacion> t = std::make_shared<Reubicar>(*this, cuerpo
    Vehiculo, posicion);
183     transformaciones_.push(t);
184 }
185
186 void Fisicas::quitar(Barro& barro) {
187     b2Body* cuerpo = colisionables_.at(barro.uuid());
```



```
188     std::shared_ptr<Transformacion> t = std::make_shared<Quitar>(*this, cuerpo,
    barro.uuid());
189     transformaciones_.push(t);
190 }
191
192 void Fisicas::quitar(Boost& boost) {
193     b2Body* cuerpo = colisionables_.at(boost.uuid());
194     std::shared_ptr<Transformacion> t = std::make_shared<Quitar>(*this, cuerpo,
    boost.uuid());
195     transformaciones_.push(t);
196 }
197
198 void Fisicas::quitar(Aceite& aceite) {
199     b2Body* cuerpo = colisionables_.at(aceite.uuid());
200     std::shared_ptr<Transformacion> t = std::make_shared<Quitar>(*this, cuerpo,
    aceite.uuid());
201     transformaciones_.push(t);
202 }
203
204 void Fisicas::quitar(Piedra& piedra) {
205     b2Body* cuerpo = colisionables_.at(piedra.uuid());
206     std::shared_ptr<Transformacion> t = std::make_shared<Quitar>(*this, cuerpo,
    piedra.uuid());
207     transformaciones_.push(t);
208 }
209
210 void Fisicas::quitar(CajaVida& cajaVida) {
211     b2Body* cuerpo = colisionables_.at(cajaVida.uuid());
212     std::shared_ptr<Transformacion> t = std::make_shared<Quitar>(*this, cuerpo,
    cajaVida.uuid());
213     transformaciones_.push(t);
214 }
```

```
#include "includes/servidor/modelo/fisicas/ContactListener.h"

#include "includes/servidor/modelo/Colisionable.h"
#include "includes/servidor/utils/ConfigServidor.h"

#include "includes/servidor/modelo/entidades/carrera/Checkpoint.h"
#include "includes/servidor/modelo/superficies/SuperficieArena.h"
#include "includes/servidor/modelo/entidades/Vehiculo.h"
#include "includes/servidor/modelo/entidades/CajaVida.h"
#include "includes/servidor/modelo/entidades/Aceite.h"
#include "includes/servidor/modelo/entidades/Barro.h"
#include "includes/servidor/modelo/entidades/Piedra.h"
#include "includes/servidor/modelo/entidades/Boost.h"
#include "includes/servidor/modelo/fisicas/Fisicas.h"

#include "includes/common/eventos/EventoChoque.h"
#include "includes/common/eventos/EventoExplosion.h"
#include "includes/common/eventos/EventoBarroPisado.h"
#include "includes/common/eventos/EventoFinBarro.h"

// MÃ©todos privados
static void ordenar(Colisionable** A, Colisionable** B);

ContactListener::ContactListener(Fisicas& fisicas) :
    fisicas_(fisicas) {
}

ContactListener::~ContactListener() {
}
#include <iostream>
void ContactListener::BeginContact(b2Contact* contact) {
    //Son raw pointers pero b2d garantiza que no se eliminan ni agregan cuerpos durante
    // las colisiones
    Colisionable* colisionableA = contact->GetFixtureA()->GetBody()->GetUserData();
    Colisionable* colisionableB = contact->GetFixtureB()->GetBody()->GetUserData();

    ordenar(&colisionableA, &colisionableB);

    // No deberÃa ocurrir porque no le puse data a los fixtures.
    //if(!colisionableA || !colisionableB) {
    //    return;
    //}

    if (colisionableA->getTipo() == Colisionable::tipos::VEHICULO_) {
        if (colisionableB->yaFueColisionado()) {
            //No queremos eliminar dos veces o revienta todo
            return;
        }
        if (colisionableB->getTipo() == Colisionable::tipos::VEHICULO_) {
            vehiculoVsVehiculo(*static_cast<Vehiculo*>(colisionableA), *static_cast<Vehiculo*>(colisionableB));
        }
        if (colisionableB->getTipo() == Colisionable::tipos::SUPERFICIE_ARENA_) {
            vehiculoVsArena(*static_cast<Vehiculo*>(colisionableA), *static_cast<SuperficieArena*>(colisionableB));
        }
        if (colisionableB->getTipo() == Colisionable::tipos::CHECKPOINT_) {
            vehiculoVsCheckpoint(*static_cast<Vehiculo*>(colisionableA), *static_cast<Checkpoint*>(colisionableB));
        }
        if (colisionableB->getTipo() == Colisionable::tipos::SUPERFICIE_TIERRA_) {
            //TODO: No se pueden modificar valores acÃ¡, hay que obtener el b2Ve
```



```
hiculo
        }
        if (colisionableB->getTipo() == Colisionable::tipos::SUPERFICIE_PISTA_) {
        }
        if (colisionableB->getTipo() == Colisionable::tipos::SALUD_) {
            vehiculoVsCajaVida(*static_cast<Vehiculo*>(colisionableA), *static_cast<CajaVida*>(colisionableB));
        }
        if (colisionableB->getTipo() == Colisionable::tipos::ACEITE_) {
            vehiculoVsAceite(*static_cast<Vehiculo*>(colisionableA), *static_cast<Aceite*>(colisionableB));
        }
        if (colisionableB->getTipo() == Colisionable::tipos::BARRO_) {
            vehiculoVsBarro(*static_cast<Vehiculo*>(colisionableA), *static_cast<Barro*>(colisionableB));
        }
        if (colisionableB->getTipo() == Colisionable::tipos::PIEDRA_) {
            vehiculoVsPiedra(*static_cast<Vehiculo*>(colisionableA), *static_cast<Piedra*>(colisionableB));
        }
        if (colisionableB->getTipo() == Colisionable::tipos::BOOST_) {
            vehiculoVsBoost(*static_cast<Vehiculo*>(colisionableA), *static_cast<Boost*>(colisionableB));
        }
    }
}

void ContactListener::EndContact(b2Contact* contact) {
}

void ContactListener::PreSolve(b2Contact* contact, const b2Manifold* oldManifold) {
    /*b2WorldManifold worldManifold;
    contact->GetWorldManifold(&worldManifold);
    b2PointState state1[2], state2[2];
    b2GetPointStates(state1, state2, oldManifold, contact->GetManifold());
    if (state2[0] == b2_addState) {
        const b2Body* bodyA = contact->GetFixtureA()->GetBody();
        const b2Body* bodyB = contact->GetFixtureB()->GetBody();
        b2Vec2 point = worldManifold.points[0];
        b2Vec2 vA = bodyA->GetLinearVelocityFromWorldPoint(point);
        b2Vec2 vB = bodyB->GetLinearVelocityFromWorldPoint(point);
        float32 approachVelocity = b2Dot(vB -vA, worldManifold.normal);    */

}

void ContactListener::PostSolve(b2Contact* contact, const b2ContactImpulse* impulse) {

}

static void ordenar(Colisionable** A, Colisionable** B) {

    Colisionable* Aptr = *A;
    Colisionable* Bptr = *B;

    int tipoDeA = Aptr->getTipo();
    int tipoDeB = Bptr->getTipo();

    if (tipoDeB < tipoDeA) {
        Colisionable* tmp = *B;
        *B = *A;
        *A = tmp;
    }
}
```

```
117 void ContactListener::vehiculoVsArena(Vehiculo& vehiculo, SuperficieArena& arena
    ) {
118     Posicion posicionVehiculo = fisicas_.getPosicionDe(vehiculo.uuid());
119     std::shared_ptr<Evento> explosion = std::make_shared<EventoExplosion>(posici
    onVehiculo.x_, posicionVehiculo.y_);
120     fisicas_.ocurrio(explosion);
121     fisicas_.reubicar(vehiculo, vehiculo.getPuntoRespawn());
122 }
123
124 void ContactListener::vehiculoVsCheckpoint(Vehiculo& vehiculo, Checkpoint& check
    point) {
125    checkpoint.registrarPaso(vehiculo);
126 }
127
128 void ContactListener::vehiculoVsVehiculo(Vehiculo& vehiculoA, Vehiculo& vehiculo
    B) {
129     Posicion posicionVehiculoA = fisicas_.getPosicionDe(vehiculoA.uuid());
130     std::shared_ptr<Evento> choque = std::make_shared<EventoChoque>(posicionVehi
    culoA.x_, posicionVehiculoA.y_);
131     fisicas_.ocurrio(choque);
132
133     uint8_t disminucionVida = CONFIG_SERVIDOR.disminucionVidaChoqueConVehiculo()
    ;
134     bool vehiculoAExploto = vehiculoA.disminuirSalud(disminucionVida);
135     if (vehiculoAExploto) {
136         std::shared_ptr<Evento> explosion = std::make_shared<EventoExplosion>(po
    sicionVehiculoA.x_, posicionVehiculoA.y_);
137         fisicas_.ocurrio(explosion);
138         fisicas_.reubicar(vehiculoA, vehiculoA.getPuntoRespawn());
139     }
140     bool vehiculoBExploto = vehiculoB.disminuirSalud(disminucionVida);
141     if (vehiculoBExploto) {
142         Posicion posicionVehiculoB = fisicas_.getPosicionDe(vehiculoB.uuid());
143         std::shared_ptr<Evento> explosion = std::make_shared<EventoExplosion>(po
    sicionVehiculoB.x_, posicionVehiculoB.y_);
144         fisicas_.ocurrio(explosion);
145         fisicas_.reubicar(vehiculoB, vehiculoB.getPuntoRespawn());
146     }
147 }
148
149 void ContactListener::vehiculoVsCajaVida(Vehiculo& vehiculo, CajaVida& cajaVida)
     {
150     int deltaVida = cajaVida.deltaVida();
151     vehiculo.sumarSalud(deltaVida);
152     fisicas_.quitar(cajaVida);
153 }
154
155 void ContactListener::vehiculoVsAceite(Vehiculo& vehiculo, Aceite& aceite) {
156     fisicas_.quitar(aceite);
157 }
158
159 void ContactListener::vehiculoVsBarro(Vehiculo& vehiculo, Barro& barro) {
160     std::shared_ptr<Evento> pisoBarro = std::make_shared<EventoBarroPisado>();
161     fisicas_.quitar(barro);
162     vehiculo.duenio()→ocurrio(pisoBarro);
163     std::shared_ptr<Evento> noMasBarro = std::make_shared<EventoFinBarro>();
164     //TODO: No hardcodear
165     vehiculo.ocurrira(noMasBarro, 300);
166 }
167
168 void ContactListener::vehiculoVsBoost(Vehiculo& vehiculo, Boost& boost) {
169     fisicas_.quitar(boost);
170 }
171
172 void ContactListener::vehiculoVsPiedra(Vehiculo& vehiculo, Piedra& piedra) {
173     //TODO: NO HARDCODEAR
```



```
174     bool exploto = vehiculo.disminuirSalud(30);
175     if (exploto) {
176         Posicion posicionVehiculo = fisicas_.getPosicionDe(vehiculo.uuid());
177         std::shared_ptr<Evento> explosion = std::make_shared<EventoExplosion>(po
    sicionVehiculo.x_, posicionVehiculo.y_);
178         fisicas_.ocurrio(explosion);
179         fisicas_.reubicar(vehiculo, vehiculo.getPuntoRespawn());
180     }
181     fisicas_.quitar(piedra);
182
183 }
```

```
#include "includes/servidor/modelo/fisicas/B2DVehiculo.h"

#include "includes/servidor/modelo/entidades/Vehiculo.h"
#include "includes/servidor/utils/ConfigServidor.h"

//TODO: Que la velocidad hacia atrÃ¡s dependa de la hacia delante
B2DVehiculo::B2DVehiculo(b2World* mundoBox2D, Vehiculo& vehiculo)
    : control_(0)
    , velocidadMaxAdelante_((float)vehiculo.velocidadMaximaAdelante() * AJUSTE_VELOCIDAD)
    , velocidadMaxAtras_((-1)*(float)vehiculo.velocidadMaximaAtras() * AJUSTE_VELOCIDAD) {

    b2BodyDef bodyDef;
    bodyDef.type = b2_dynamicBody;
    bodyDef.bullet = true;
    cuerpoBox2D_ = mundoBox2D->CreateBody(&bodyDef);

    b2PolygonShape polygonShape;
    float ancho = CONFIG_SERVIDOR.anchoVehiculo() / 2.0f;
    float largo = CONFIG_SERVIDOR.largoVehiculo() / 2.0f;
    polygonShape.SetAsBox(ancho, largo);
    b2Fixture* fixture = cuerpoBox2D_->CreateFixture(&polygonShape, DENSIDAD);
    //TODO: Propiedades del auto, sublcase que tiene datos (?)
    fixture->SetUserData(nullptr);
    cuerpoBox2D_->SetUserData(&vehiculo);

    //TODO: Revisar agarre
    traccion_ = (float)vehiculo.agarre() / 100.0f;
    fuerzaManejoMaxima_ = (float)vehiculo.aceleracion() * AJUSTE_ACELERACION;
}

B2DVehiculo::~B2DVehiculo() {
    cuerpoBox2D_->GetWorld()->DestroyBody(cuerpoBox2D_);
}

b2Vec2 B2DVehiculo::getVelocidadLateral() {
    b2Vec2 normal = cuerpoBox2D_->GetWorldVector(b2Vec2(1,0));
    return b2Dot(normal, cuerpoBox2D_->GetLinearVelocity()) * normal;
}

b2Body* B2DVehiculo::getB2D() {
    return cuerpoBox2D_;
}

b2Vec2 B2DVehiculo::getVelocidadFrontal() {
    b2Vec2 normal = cuerpoBox2D_->GetWorldVector(b2Vec2(0,1));
    return b2Dot(normal, cuerpoBox2D_->GetLinearVelocity()) * normal;
}

void B2DVehiculo::actualizarFriccion() {
    //Derrape
    //FIXME: Hacerlo f(velocidadActual) o agregar mas ruedas
    float maxImpulsoLateral = 25.0f / (CONFIG_SERVIDOR.agarreVehiculo() / 100.0f);
    b2Vec2 impulso = cuerpoBox2D_->GetMass() * -getVelocidadLateral();
    if (impulso.Length() > maxImpulsoLateral) {
        impulso *= maxImpulsoLateral / impulso.Length();
    }
    cuerpoBox2D_->ApplyLinearImpulse(traccion_*impulso, cuerpoBox2D_->GetWorldCenter(), true);

    cuerpoBox2D_->ApplyAngularImpulse(0.166f * -cuerpoBox2D_->GetAngularVelocity(), true);

    b2Vec2 normal = getVelocidadFrontal();
```



```
    float velocidadActual = normal.Normalize();
    float rozamientoFrenado = -(fuerzaManejoMaxima_/velocidadMaxAdelante_)*velocidadActual*0.000001f;
    cuerpoBox2D_->ApplyForce(rozamientoFrenado * normal, cuerpoBox2D_->GetWorldCenter(), true);
}

void B2DVehiculo::actualizarAceleracion() {
    // El control state llega de los eventos
    float velocidadObjetivo = 0;
    switch (control_ & (acelerador_|freno_)) {
        case acelerador_:
            velocidadObjetivo = velocidadMaxAdelante_;
            break;
        case freno_:
            velocidadObjetivo = velocidadMaxAtras_;
            break;
        default:
            break;
    }
    b2Vec2 normal = cuerpoBox2D_->GetWorldVector(b2Vec2(0,1));
    float velocidadActual = b2Dot(getVelocidadFrontal(),normal);
    float fuerza = 0;
    if (velocidadObjetivo > velocidadActual) {
        fuerza = fuerzaManejoMaxima_;
    } else if (velocidadObjetivo < velocidadActual) {
        fuerza = (-1)*fuerzaManejoMaxima_;
    } else {
        return;
    }
    cuerpoBox2D_->ApplyForce(traccion_ * fuerza * normal, cuerpoBox2D_->GetWorldCenter(), true);
}

void B2DVehiculo::actualizarVolante() {
    float torque = 0;
    switch (control_ & (volanteIzquierda_|volanteDerecha_)) {
        case volanteIzquierda_:
            torque = AJUSTE_VOLANTE*(float)CONFIG_SERVIDOR.maniobrabilidadVehiculo();
            break;
        case volanteDerecha_:
            torque = (-AJUSTE_VOLANTE)*(float)CONFIG_SERVIDOR.maniobrabilidadVehiculo();
            break;
        default:
            break;
    }
    cuerpoBox2D_->ApplyTorque(torque, true);
}

void B2DVehiculo::step() {
    actualizarFriccion();
    actualizarAceleracion();
    actualizarVolante();
}

void B2DVehiculo::acelerando() {
    control_ |= acelerador_;
}

void B2DVehiculo::desacelerando() {
    control_ &= ~acelerador_;
}

void B2DVehiculo::frenando() {
```

```
123     control_ |= freno_;
124 }
125
126 void B2DVehiculo::dejandoDeFrenar() {
127     control_ &= ~freno_;
128 }
129
130 void B2DVehiculo::doblandoIzquierda() {
131     control_ |= volanteIzquierda_;
132 }
133
134 void B2DVehiculo::dejandoDeDoblarIzquierda() {
135     control_ &= ~volanteIzquierda_;
136 }
137
138 void B2DVehiculo::doblandoDerecha() {
139     control_ |= volanteDerecha_;
140 }
141
142 void B2DVehiculo::dejandoDeDoblarDerecha() {
143     control_ &= ~volanteDerecha_;
144 }
```



```
1  #include "includes/servidor/modelo/entidades/Vehiculo.h"
2
3  #include <algorithm>
4
5  Vehiculo::Vehiculo(uint8_t uuid,
6              unsigned int velocidadMaximaAdelante,
7              unsigned int velocidadMaximaAtras,
8              unsigned int aceleracion,
9              unsigned int maniobrabilidad,
10             unsigned int agarre,
11             //FIXME: CAMBIAR SALUD A UINT8T
12             unsigned int salud,
13             Posicion respawn,
14             std::shared_ptr<Jugador> duenio) :
15             Identificable(uuid),
16             velocidadMaximaAdelante_(velocidadMaximaAdelante),
17             velocidadMaximaAtras_(velocidadMaximaAtras),
18             aceleracion_(aceleracion),
19             maniobrabilidad_(maniobrabilidad),
20             agarre_(agarre),
21             salud_(salud),
22             saludDefault_(salud),
23             respawn_(respawn),
24             duenio_(duenio) {
25 }
26
27 unsigned int Vehiculo::velocidadMaximaAdelante() {
28     return velocidadMaximaAdelante_;
29 }
30
31 unsigned int Vehiculo::velocidadMaximaAtras() {
32     return velocidadMaximaAtras_;
33 }
34
35 unsigned int Vehiculo::aceleracion() {
36     return aceleracion_;
37 }
38
39 unsigned int Vehiculo::maniobrabilidad() {
40     return maniobrabilidad_;
41 }
42
43 unsigned int Vehiculo::agarre() {
44     return agarre_;
45 }
46
47 unsigned int Vehiculo::salud() {
48     return salud_;
49 }
50
51 std::shared_ptr<Jugador> Vehiculo::duenio() {
52     return duenio_;
53 }
54
55 void Vehiculo::ocurrira(std::shared_ptr<Evento> unEvento, uint32_t steps) {
56     futuros_.push_back(futuro_t{unEvento, steps});
57 }
58
59 static bool termino(futuro_t& futuro) {
60     return (futuro.steps ≡ 0) ∧ (futuro.evento.use_count() > 2);
61 }
62
63 void Vehiculo::step() {
64     std::remove_if(futuros_.begin(), futuros_.end(), termino);
65     for (auto& futuro : futuros_) {
66         futuro.steps−−;
```

```
67          if (futuro.steps ≡ 0) {
68              duenio_→ocurrio(futuro.evento);
69          }
70      }
71  }
72
73  bool Vehiculo::disminuirSalud(uint8_t cantidad) {
74      int saludFinal = salud_ − cantidad;
75      if (saludFinal ≤ 0) {
76          salud_ = saludDefault_;
77          return true;
78      }
79      salud_ = saludFinal;
80      return false;
81  }
82
83  void Vehiculo::sumarSalud(int delta) {
84      unsigned int saludFinal = salud_ + delta;
85      if (saludFinal > saludDefault_) {
86          salud_ = saludDefault_;
87      } else {
88          salud_ = saludFinal;
89      }
90  }
91
92  int Vehiculo::getTipo() {
93      return VEHICULO_;
94  }
95
96  Posicion& Vehiculo::getPuntoRespawn() {
97      return respawn_;
98  }
99
100 void Vehiculo::setPuntoRespawn(Posicion& posicion) {
101     respawn_ = posicion;
102 }
```



```
1   #include "includes/servidor/modelo/entidades/Piedra.h"
2
3   Piedra::Piedra(uint8_t uuid) :
4       Modificador(uuid) {
5   }
6
7   int Piedra::getTipo() {
8       return PIEDRA_;
9   }
```

nov 26, 19 17:34 **Modificador.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/entidades/Modificador.h"

Modificador::Modificador(uint8_t uuid) :
    Identificable(uuid) {
}

Modificador::~Modificador() {

}
```

nov 26, 19 17:34 **Checkpoint.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/entidades/carrera/Checkpoint.h"

#include "includes/servidor/modelo/entidades/carrera/Carrera.h"
#include "includes/servidor/modelo/entidades/Vehiculo.h"

Checkpoint::Checkpoint(Carrera& carrera, int id, int idDelSiguiente, float ancho
, float alto, Posicion& posicion) :
    carrera_(carrera),
    id_(id),
    idDelSiguiente_(idDelSiguiente),
    ancho_(ancho),
    alto_(alto),
    puntoRespawn_(posicion) {
}

Checkpoint::~Checkpoint() {

}

int Checkpoint::getTipo() {
    return CHECKPOINT_;
}

Posicion& Checkpoint::posicion() {
    return puntoRespawn_;
}

float Checkpoint::ancho() {
    return ancho_;
}
float Checkpoint::alto() {
    return alto_;
}

int Checkpoint::id() {
    return id_;
}

void Checkpoint::registrarPaso(Vehiculo& vehiculo) {
    Checkpoint& ultimoCheckpoint = carrera_.ultimoCheckpointDe(vehiculo);
    if (ultimoCheckpoint.idDelSiguiente_ == id_) {
        //TODO: EN Meta serÃ¡ sumar vuelta
        vehiculo.setPuntoRespawn(puntoRespawn_);
        carrera_.setCheckpoint(vehiculo, *this);
    }
}
```

```
1  #include "includes/servidor/modelo/entidades/carrera/Carrera.h"
2
3  #include "includes/servidor/modelo/entidades/Vehiculo.h"
4  #include "includes/common/Conversor.h"
5  #include "includes/common/eventos/EventoFinCarrera.h"
6
7  Carrera::Carrera(ColaProtegida<std::shared_ptr<Evento>>& eventosMundo) :
8      numeroDeVueltas_(0),
9      eventosMundo_(eventosMundo) {
10 }
11
12 void Carrera::cargarDesdeJson(Json& pistaJson) {
13     int cantidadCheckpoints = pistaJson["checkpoints"]["cantidad"].get<int>();
14     uint16_t largoY = pistaJson["dimensiones"]["y"].get<uint16_t>();
15
16     //TODO: Cargo checkpoints, meta es aparte (?) Â¿Lo es?
17     for (int i = 0; i < cantidadCheckpoints; ++i) {
18         float x = pistaJson["checkpoints"][std::to_string(i)]["x"].get<float>();
19         float y = largoY - pistaJson["checkpoints"][std::to_string(i)]["y"].get<flo
   at>();
20         float ancho = pistaJson["checkpoints"][std::to_string(i)]["ancho"].get<float
   >();
21         float largo = pistaJson["checkpoints"][std::to_string(i)]["largo"].get<float
   >();
22         uint16_t angulo = pistaJson["checkpoints"][std::to_string(i)]["anguloRespawn"
   ].get<uint16_t>();
23         Posicion posicion(Conversor::tileAMetro(x), Conversor::tileAMetro(y), an
   gulo);
24         checkpoints_.emplace(i, Checkpoint(*this, i, (i+1) % cantidadCheckpoints
   , ancho, largo, posicion));
25     }
26     numeroDeVueltas_ = pistaJson["vueltas"].get<int>();
27 }
28
29 std::map<int, Checkpoint>& Carrera::checkpoints() {
30     return checkpoints_;
31 }
32
33 Checkpoint& Carrera::ultimoCheckpointDe(Vehiculo& vehiculo) {
34     return checkpoints_.at(idsVehiculosAidsCheckpoints_.at(vehiculo.uuid()));
35 }
36
37 void Carrera::setCheckpoint(Vehiculo& vehiculo, Checkpoint& checkpoint) {
38     idsVehiculosAidsCheckpoints_[vehiculo.uuid()] = checkpoint.id();
39     if (checkpoint.id() == ID_META) {
40         idsVehiculosAVueltas_[vehiculo.uuid()]++;
41         if (idsVehiculosAVueltas_[vehiculo.uuid()] == numeroDeVueltas_) {
42             podio_.emplace_back(vehiculo.uuid());
43         }
44     }
45     bool termino = finalizada();
46     if (termino) {
47         std::shared_ptr<Evento> fin = std::make_shared<EventoFinCarrera>(std::mo
   ve(podio_));
48         eventosMundo_.put(fin);
49     }
50 }
51
52 void Carrera::registrarVehiculo(Vehiculo& vehiculo) {
53     idsVehiculosAidsCheckpoints_[vehiculo.uuid()] = 0;
54     idsVehiculosAVueltas_[vehiculo.uuid()] = 0;
55 }
56
57 bool Carrera::finalizada() {
58     for (auto& kv : idsVehiculosAVueltas_) {
59         if (kv.second < numeroDeVueltas_) {
```



```
60             return false;
61         }
62     }
63     return true;
64 }
```

nov 26, 19 17:34 **CajaVida.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/entidades/CajaVida.h"

CajaVida::CajaVida(uint8_t uuid, int deltaVida) :
    Modificador(uuid),
    deltaVida_(deltaVida) {

}

int CajaVida::getTipo() {
    return SALUD_;
}

int CajaVida::deltaVida() {
    return deltaVida_;
}
```

nov 26, 19 17:34 **Boost.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/entidades/Boost.h"

Boost::Boost(uint8_t uuid) :
    Modificador(uuid) {
}

int Boost::getTipo() {
    return BOOST_;
}
```

nov 26, 19 17:34 **Barro.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/entidades/Barro.h"

Barro::Barro(uint8_t uuid) :
    Modificador(uuid) {
}

int Barro::getTipo() {
    return BARRO_;
}
```

nov 26, 19 17:34 **Aceite.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/entidades/Aceite.h"

Aceite::Aceite(uint8_t uuid) :
    Modificador(uuid) {
}

int Aceite::getTipo() {
    return ACEITE_;
}
```

nov 26, 19 17:34 **Colisionable.cpp** Page 1/1

```cpp
#include "includes/servidor/modelo/Colisionable.h"

Colisionable::~Colisionable() {
}

Colisionable::Colisionable() :
    yaColisionado_(false) {
}

void Colisionable::colisionado() {
    yaColisionado_ = true;
}

bool Colisionable::yaFueColisionado() {
    return yaColisionado_;
}
```

nov 26, 19 17:34 **main_servidor.cpp** Page 1/1

```cpp
#include <iostream>

#include "includes/servidor/Servidor.h"
#include "includes/servidor/utils/ConfigServidor.h"

int main(int argc, char const *argv[]) {
    Servidor servidor(CONFIG_SERVIDOR.hostServidor(), CONFIG_SERVIDOR.puertoServ
idor());
    try {
        servidor.correr();
    } catch(const std::exception& e) {
        std::cout << e.what() << '\n';
    }
    servidor.cerrar();
    return 0;
}
```

nov 26, 19 17:34 **Jugador.cpp** Page 1/1

```cpp
#include "includes/servidor/Jugador.h"

Jugador::Jugador(SocketTCP^ socket, uint32_t uuid, ColaBloqueante<std::shared_ptr<Evento>>& destinoEventos) :
    UUID_(uuid),
    socket_(std::move(socket)),
    destino_(destinoEventos),
    recibidorEventos_(socket_, destino_, UUID_),
    enviadorEventos_(socket_, eventosAEnviar_) {

    recibidorEventos_.iniciar();
    enviadorEventos_.iniciar();
}

Jugador::~Jugador() {
    eventosAEnviar_.detener();

    socket_.cerrarLectoEscritura();

    recibidorEventos_.detener();
    recibidorEventos_.join();

    enviadorEventos_.detener();
    enviadorEventos_.join();
}

uint32_t Jugador::uuid () {
    return UUID_;
}

void Jugador::ocurrio(std::shared_ptr<Evento> e) {
    eventosAEnviar_.put(e);
}
```

nov 26, 19 17:34 **HiloAceptador.cpp** Page 1/1

```cpp
#include "includes/servidor/HiloAceptador.h"

#include <iostream>

#include "includes/servidor/utils/ConfigServidor.h"

HiloAceptador::HiloAceptador(const std::string& unHost, const std::string& puerto, SalaDeEspera& salaDeEspera) :
    sktAceptador_(unHost, puerto),
    salaDeEspera_(salaDeEspera) {
}

void HiloAceptador::correr() {
    try {
    sktAceptador_.enlazar();
    sktAceptador_.escuchar(CONFIG_SERVIDOR.maxClientesEnEspera());
    }
    catch(const std::exception& e) {
        std::cerr << e.what() << '\n';
    }
    while (seguirCorriendo_) {
        try {
            SocketTCP aceptado = sktAceptador_.aceptar();
            salaDeEspera_.agregarJugador(std::move(aceptado));
        }
        catch(const std::exception& e) {
            std::cerr << e.what() << '\n';
        }
    }
}

void HiloAceptador::detener() {
    seguirCorriendo_ = false;
    sktAceptador_.cerrarLectoEscritura();
}

HiloAceptador::~HiloAceptador() {
}
```

```cpp
#include "includes/servidor/DistribuidorEventos.h"

DistribuidorEventos::DistribuidorEventos(ColaBloqueante<std::shared_ptr<Evento>>
& eventos, SalaDeEspera& salaDeEspera, CoordinadorPartidas& coordinadorPartidas)
 :
    eventos_(eventos),
    salaDeEspera_(salaDeEspera),
    coordinadorPartidas_(coordinadorPartidas) {
}

DistribuidorEventos::~DistribuidorEventos() {
}

void DistribuidorEventos::correr() {
    bool obtenido;
    std::shared_ptr<Evento> evento;
    while(seguirCorriendo_ ∧ (obtenido = eventos_.get(evento))) {
        manejar(*evento);
    }
}

void DistribuidorEventos::detener() {
    seguirCorriendo_ = false;
}

void DistribuidorEventos::manejar(Evento& e) {
    e.actualizar(*this);
}

void DistribuidorEventos::manejar(EventoAcelerar& e) {
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoDesacelerar& e) {
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoFrenar& e) {
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoDejarDeFrenar& e) {
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoDoblarIzquierda& e) {
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoDejarDeDoblarIzquierda& e) {
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoDoblarDerecha& e) {
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoDejarDeDoblarDerecha& e) {
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoCrearPartida& e) {
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoUnirseAPartida& e) {
```



```cpp
    uint32_t uuidJugador = e.uuidRemitente();
    uint16_t uuidPartida = e.uuidPartida_;
    std::shared_ptr<Jugador> jugador = salaDeEspera_.quitarJugador(uuidJugador);
    coordinadorPartidas_.agregarJugadorAPartida(jugador, uuidPartida);
}

void DistribuidorEventos::manejar(EventoDesconexion& e) {
    salaDeEspera_.manejar(e);
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoIniciarPartida& e) {
    coordinadorPartidas_.manejar(e);
}

void DistribuidorEventos::manejar(EventoUnirseASala& e) {
    std::shared_ptr<EventoSnapshotSala> snapshot = coordinadorPartidas_.getSnaps
hotSala();
    snapshot→setRemitente(e.uuidRemitente());
    salaDeEspera_.ocurrio(snapshot);
}
```

```cpp
#include "includes/servidor/CoordinadorPartidas.h"

CoordinadorPartidas::CoordinadorPartidas(SalaDeEspera& salaDeEspera) :
    contadorPartidas_(0),
    salaDeEspera_(salaDeEspera){
}

CoordinadorPartidas::~CoordinadorPartidas() {
    for (const auto& kv : partidas_) {
        if (kv.second→estaCorriendo()) {
            kv.second→detener();
        }
        kv.second→join();
    }
}

void CoordinadorPartidas::agregarJugadorAPartida(std::shared_ptr<Jugador> jugador, uint16_t uuidPartida) {
    partidas_.at(uuidPartida)→agregarJugador(jugador);
    jugadoresAPartidas_[jugador→uuid()] = uuidPartida;
    for (const auto& kv : partidas_.at(uuidPartida)→jugadores()) {
        kv.second→ocurrio(getSnapshotLobby(uuidPartida));
    }
}

std::shared_ptr<EventoSnapshotSala> CoordinadorPartidas::getSnapshotSala() {
    std::map<uint16_t, uint16_t> datosSnapshot;
    uint16_t ordinal = 1;
    for (const auto& kv : partidas_) {
        //FIXME: PREGUNTAR POR ESTACORRIENDO LA ELIMINACION "ORDENADA" TIENE QUE HACERSE EN UN SOLO LUGAR
        if (¬kv.second→aceptaJugadores()) {
            continue;
        }
        datosSnapshot.emplace(ordinal, kv.first);
        ordinal++;
    }
    return std::make_shared<EventoSnapshotSala>(std::move(datosSnapshot));
}

std::shared_ptr<EventoSnapshotLobby> CoordinadorPartidas::getSnapshotLobby(uint16_t uuidPartida) {
    std::map<uint32_t, bool> datosDelvento;
    std::shared_ptr<Partida> partida = partidas_.at(uuidPartida);
    std::map<uint32_t, std::shared_ptr<Jugador>> jugadoresEnPartida = partida→jugadores();
    for (const auto& kv : jugadoresEnPartida) {
        //FIXME: Agregar logica de estoy listo
        datosDelvento.emplace(kv.first, partida→estaListo(kv.first));
    }
    return std::make_shared<EventoSnapshotLobby>(std::move(datosDelvento));

}

void CoordinadorPartidas::manejar(Evento& e) {
    e.actualizar(*this);
}

void CoordinadorPartidas::manejar(EventoCrearPartida& e) {
    //FIXME: Si contadorPartidas da la vuelta a 0, lanzar error o soucionarlo
    contadorPartidas_++;
    // TODO: AcÃ¡ hay que decir que uuid de mapa se quiere cargar
    // FIXME: No hardcodear esto
    uint16_t uuidPista = 1;
    partidas_[contadorPartidas_] = std::make_shared<Partida>(uuidPista, salaDeEspera_);
```



```cpp
    std::shared_ptr<Evento> partidaCreada = std::make_shared<EventoPartidaCreada>(contadorPartidas_, e.uuidRemitente());
    salaDeEspera_.getJugador(e.uuidRemitente())→ocurrio(partidaCreada);
    salaDeEspera_.ocurrio(getSnapshotSala());
    //FIXME: Quitar partidas finalizadas, que no deben tener jugadores dentro (En realidad no iporta)

}

//FIXME: DeberÃ-a esperar que todos envÃ-en jugar.
void CoordinadorPartidas::manejar(EventoIniciarPartida& e) {
    uint32_t uuidJugador = e.uuidRemitente();
    uint16_t uuidPartida = jugadoresAPartidas_[uuidJugador];
    std::shared_ptr<Partida> partida = partidas_.at(uuidPartida);
    partida→marcarListo(uuidJugador);
    for (const auto& kv : partidas_.at(uuidPartida)→jugadores()) {
        kv.second→ocurrio(getSnapshotLobby(uuidPartida));
    }
    if (partida→todosListos()) {
        partidas_[uuidPartida]→iniciar();
    }
}

void CoordinadorPartidas::manejar(EventoDesconexion& e) {
    //TODO: enviar a la partida correspondiente el
    //evento desconexion para que se quite al jugador.
    //partidas_[1]->manejar(e);
    //partidas_.erase(1);
}

void CoordinadorPartidas::manejar(EventoAcelerar& e) {
    uint32_t uuidJugador = e.uuidRemitente();
    uint16_t uuidPartida = jugadoresAPartidas_[uuidJugador];
    std::shared_ptr<Evento> evento = std::make_shared<EventoAcelerar>(std::move(e));
    partidas_[uuidPartida]→ocurrio(evento);
}


void CoordinadorPartidas::manejar(EventoDesacelerar& e) {
    uint32_t uuidJugador = e.uuidRemitente();
    uint16_t uuidPartida = jugadoresAPartidas_[uuidJugador];
    std::shared_ptr<Evento> evento = std::make_shared<EventoDesacelerar>(std::move(e));
    partidas_[uuidPartida]→ocurrio(evento);
}

void CoordinadorPartidas::manejar(EventoFrenar& e) {
    uint32_t uuidJugador = e.uuidRemitente();
    uint16_t uuidPartida = jugadoresAPartidas_[uuidJugador];
    std::shared_ptr<Evento> evento = std::make_shared<EventoFrenar>(std::move(e));
    partidas_[uuidPartida]→ocurrio(evento);
}

void CoordinadorPartidas::manejar(EventoDejarDeFrenar& e) {
    uint32_t uuidJugador = e.uuidRemitente();
    uint16_t uuidPartida = jugadoresAPartidas_[uuidJugador];
    std::shared_ptr<Evento> evento = std::make_shared<EventoDejarDeFrenar>(std::move(e));
    partidas_[uuidPartida]→ocurrio(evento);
}

void CoordinadorPartidas::manejar(EventoDoblarIzquierda& e) {
    uint32_t uuidJugador = e.uuidRemitente();
    uint16_t uuidPartida = jugadoresAPartidas_[uuidJugador];
```

```
121     std::shared_ptr<Evento> evento = std::make_shared<EventoDoblarIzquierda>(std
    ::move(e));
122     partidas_[uuidPartida]→ocurrio(evento);
123 }
124
125 void CoordinadorPartidas::manejar(EventoDejarDeDoblarIzquierda& e) {
126     uint32_t uuidJugador = e.uuidRemitente();
127     uint16_t uuidPartida = jugadoresAPartidas_[uuidJugador];
128     std::shared_ptr<Evento> evento = std::make_shared<EventoDejarDeDoblarIzquier
    da>(std::move(e));
129     partidas_[uuidPartida]→ocurrio(evento);
130 }
131
132 void CoordinadorPartidas::manejar(EventoDoblarDerecha& e) {
133     uint32_t uuidJugador = e.uuidRemitente();
134     uint16_t uuidPartida = jugadoresAPartidas_[uuidJugador];
135     std::shared_ptr<Evento> evento = std::make_shared<EventoDoblarDerecha>(std::
    move(e));
136     partidas_[uuidPartida]→ocurrio(evento);
137 }
138
139 void CoordinadorPartidas::manejar(EventoDejarDeDoblarDerecha& e) {
140     uint32_t uuidJugador = e.uuidRemitente();
141     uint16_t uuidPartida = jugadoresAPartidas_[uuidJugador];
142     std::shared_ptr<Evento> evento = std::make_shared<EventoDejarDeDoblarDerecha
    >(std::move(e));
143     partidas_[uuidPartida]→ocurrio(evento);
144 }
```



```
1  #include "includes/common/Tile.h"
2
3  Tile::Tile(int x, int y) :
4      x_(x),
5      y_(y) {
6  }
7
8  bool Tile::operator<(const Tile& otro) const {
9      if (this→x_ < otro.x_) {
10         return true;
11     } else if (this→x_ ≡ otro.x_) {
12         if (this→y_ < otro.y_) {
13             return true;
14         }
15     }
16     return false;
17 }
```

```
1   #include "includes/common/red/SocketTCP.h"
2
3   //#define _POSIX_C_SOURCE 200112L
4
5   #include <unistd.h>
6   #include <stdexcept>
7   #include <string.h>
8
9   SocketTCP::SocketTCP(int unFileDescriptor) :
10      fileDescripor_(unFileDescriptor),
11      hints_(nullptr) {
12  }
13
14  SocketTCP::SocketTCP(const std::string& unHost, const std::string& unPuerto) :
15      fileDescripor_(-1),
16      hints_(nullptr) {
17      int status = 0;
18      addrinfo hints;
19      memset(&hints, 0, sizeof(hints));
20      hints.ai_family = IP_VERSION;
21      hints.ai_socktype = SOCKET_TYPE;
22      hints.ai_flags = AI_PASSIVE;
23      status = getaddrinfo(unHost.c_str(), unPuerto.c_str(), &hints, &hints_);
24      if (status ≠ 0) {
25          throw std::runtime_error(ERROR_GET_ADDRINFO);
26      }
27      fileDescripor_ = socket(hints_→ai_family, hints_→ai_socktype, hints_→ai_pr
    otocol);
28      if (fileDescripor_ ≡ -1) {
29          throw std::runtime_error(ERROR_CREAR);
30      }
31  }
32
33  SocketTCP::SocketTCP(SocketTCP∧ otro) {
34      hints_ = otro.hints_;
35      fileDescripor_ = otro.fileDescripor_;
36      otro.hints_ = nullptr;
37      otro.fileDescripor_ = -1;
38  }
39
40  SocketTCP& SocketTCP::operator=(SocketTCP∧ otro) {
41      if (this ≡ &otro) {
42          return *this;
43      }
44      if (hints_) {
45          freeaddrinfo(hints_);
46      }
47      if (fileDescripor_ ≠ -1) {
48          close(fileDescripor_);
49      }
50      hints_ = otro.hints_;
51      fileDescripor_ = otro.fileDescripor_;
52      otro.hints_ = nullptr;
53      otro.fileDescripor_ = -1;
54      return *this;
55  }
56
57  void SocketTCP::cerrarLectoEscritura() {
58      shutdown(fileDescripor_, SHUT_RDWR);
59  }
60
61  size_t SocketTCP::enviarN(const char* buffer, size_t nBytes) {
62      size_t enviados = 0;
63      int s = 0;
64      while (enviados < nBytes) {
65          s = send(fileDescripor_, &buffer[enviados],
```



```
66                  nBytes - enviados, MSG_NOSIGNAL);
67
68          if (s < 0) {
69              throw std::runtime_error(ERROR_SEND);
70          } else if (s ≡ 0) {
71              throw std::runtime_error(ERROR_CERRADO_S);
72          } else {
73              enviados += s;
74          }
75      }
76      return enviados;
77  }
78
79  size_t SocketTCP::recibirN(char* buffer, size_t nBytes) {
80      size_t recibidos = 0;
81      int  s = 0;
82      while (recibidos < nBytes) {
83          s = recv(fileDescripor_, &buffer[recibidos],
84              nBytes - recibidos, 0);
85          if (s < 0) {
86              throw std::runtime_error(ERROR_RECV);
87          } else if (s ≡ 0) {
88              throw std::runtime_error(ERROR_CERRADO_R);
89          } else {
90              recibidos += s;
91          }
92      }
93      return recibidos;
94  }
95
96  SocketTCP::~SocketTCP() {
97      if (fileDescripor_ ≠ -1) {
98          close(fileDescripor_);
99      }
100     if (hints_) {
101         freeaddrinfo(hints_);
102     }
103 }
```

nov 26, 19 17:34 **Protocolo.cpp** Page 1/1

```cpp
#include "includes/common/red/Protocolo.h"

Protocolo::Protocolo(SocketTCP& socket) :
    socket_(socket) {
}

uint8_t Protocolo::recibirNumUnsigned8() {
    uint8_t resultadoRed = 0;
    socket_.recibirN((char*) &resultadoRed, LEN_8);
    return resultadoRed;
}

uint16_t Protocolo::recibirNumUnsigned16() {
    uint16_t resultadoRed = 0;
    socket_.recibirN((char*) &resultadoRed, LEN_16);
    return ntohs(resultadoRed);
}

uint32_t Protocolo::recibirNumUnsigned32() {
    uint32_t resultadoRed = 0;
    socket_.recibirN((char*) &resultadoRed, LEN_32);
    return ntohl(resultadoRed);
}

void Protocolo::enviar(uint8_t unNumero) {
    socket_.enviarN((const char*) &unNumero, LEN_8);
}

void Protocolo::enviar(uint16_t unNumero) {
    uint16_t numeroRed = htons(unNumero);
    socket_.enviarN((const char*) &numeroRed, LEN_16);
}

void Protocolo::enviar(uint32_t unNumero) {
    uint32_t numeroRed = htonl(unNumero);
    socket_.enviarN((const char*) &numeroRed, LEN_32);
}
```

nov 26, 19 17:34 **RecibidorEventos.cpp** Page 1/1

```cpp
#include "includes/common/RecibidorEventos.h"

#include <iostream>

#include "includes/common/eventos/EventoFactory.h"
#include "includes/common/excepciones/EventoDesconocidoError.h"

RecibidorEventos::RecibidorEventos(SocketTCP& socketOrigen, Cola<std::shared_ptr<Evento>>& destino, uint32_t uuidRemitente) :
    destino_(destino),
    protocolo_(socketOrigen),
    UUIDRemitente_(uuidRemitente) {
}

void RecibidorEventos::correr() {
    while(seguirCorriendo_) {
        try {
            std::shared_ptr<Evento> eventoRecibido(EventoFactory::instanciar(UUIDRemitente_, protocolo_));
            destino_.put(eventoRecibido);
        }
        catch(EventoDesconocidoError& e) {
            std::cerr << e.what() << '\n';
            continue;
        }
        catch(const std::exception& e) {
            std::cerr << e.what() << '\n';
            std::shared_ptr<Evento> desconexion(std::make_shared<EventoDesconexion>(UUIDRemitente_));
            destino_.put(desconexion);
            break;
        }
    }
}

void RecibidorEventos::detener() {
    seguirCorriendo_ = false;
}
```

nov 26, 19 17:34 **Hilo.cpp** Page 1/1

```cpp
#include "includes/common/Hilo.h"

#include <thread>
#include <chrono>

Hilo::Hilo()
    : seguirCorriendo_(false) {
}

Hilo::~Hilo() {
}

bool Hilo::estaCorriendo() {
    return seguirCorriendo_;
}

Hilo::Hilo(Hilo∧ otroHilo) {
    hilo_ = std::move(otroHilo.hilo_);
}

Hilo& Hilo::operator=(Hilo∧ otroHilo) {
    hilo_ = std::move(otroHilo.hilo_);
    return *this;
}

void Hilo::iniciar() {
    seguirCorriendo_ = true;
    hilo_ = std::thread(&Hilo::correr, this);
}

void Hilo::dormir(double milisegundos) {
    std::this_thread::sleep_for(std::chrono::milliseconds((uint32_t)milisegundos
));
}

void Hilo::join() {
    if (hilo_.joinable()) {
        hilo_.join();
    }
}
```

nov 26, 19 17:34 **Handler.cpp** Page 1/2

```cpp
#include "includes/common/Handler.h"

Handler::~Handler() {

}

void Handler::manejar(EventoAcelerar& e) {
}

void Handler::manejar(EventoDesacelerar& e) {
}

void Handler::manejar(EventoFrenar& e) {
}

void Handler::manejar(EventoDejarDeFrenar& e) {
}

void Handler::manejar(EventoDoblarIzquierda& e) {
}

void Handler::manejar(EventoDejarDeDoblarIzquierda& e) {
}

void Handler::manejar(EventoDoblarDerecha& e) {
}

void Handler::manejar(EventoDejarDeDoblarDerecha& e) {
}

void Handler::manejar(EventoSnapshot& e) {
}

void Handler::manejar(EventoCrearPartida& e) {
}

void Handler::manejar(EventoUnirseAPartida& e) {
}

void Handler::manejar(EventoPartidaIniciada &e) {
}

void Handler::manejar(EventoIniciarPartida& e) {
}

void Handler::manejar(EventoFinCarrera& e) {
}

void Handler::manejar(EventoAparecioConsumible &e) {
}

void Handler::manejar(EventoDesaparecioConsumible &e) {
}

void Handler::manejar(EventoChoque &e) {
}

void Handler::manejar(EventoBarroPisado &e) {
}

void Handler::manejar(EventoFinBarro &e) {
}

void Handler::manejar(EventoExplosion &e) {
}

```

nov 26, 19 17:34 **Handler.cpp** Page 2/2

```
67  void Handler::manejar(EventoDesconexion &e) {
68  }
69
70  void Handler::manejar(EventoPartidaCreada &e) {
71  }
72
73  void Handler::manejar(EventoSnapshotLobby &e) {
74  }
75
76  void Handler::manejar(EventoSnapshotSala &e) {
77  }
78
79  void Handler::manejar(EventoUnirseASala &e) {
80  }
81
82  void Handler::manejar(EventoFrenada& e) {
83  }
```

nov 26, 19 17:34 **EventoDesconocidoError.cpp** Page 1/1

```
1  #include "includes/common/excepciones/EventoDesconocidoError.h"
2
3  EventoDesconocidoError::EventoDesconocidoError(const std::string& mensaje) :
4      std::runtime_error(mensaje.c_str()) {
5
6  }
```

nov 26, 19 17:34 **EventoUnirseASala.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoUnirseASala.h"

#include "includes/common/Handler.h"

EventoUnirseASala::EventoUnirseASala(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
}

EventoUnirseASala::EventoUnirseASala() :
    Evento(0) {

    UUIDEvento_ = UUID_EVENTO_UNIRSE_A_SALA;
}

void EventoUnirseASala::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoUnirseASala::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoUnirseAPartida.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoUnirseAPartida.h"

#include "includes/common/Handler.h"

EventoUnirseAPartida::EventoUnirseAPartida(uint32_t uuidRemitente, Protocolo &protocolo) :
    Evento(uuidRemitente) {
  uuidPartida_ = protocolo.recibirNumUnsigned16();
}

EventoUnirseAPartida::EventoUnirseAPartida(uint32_t uuidRemitente, uint16_t uuidPartida) :
    Evento(uuidRemitente),
    uuidPartida_(uuidPartida) {
}

EventoUnirseAPartida::EventoUnirseAPartida(uint16_t uuidPartida) : Evento(0) {
  this→UUIDEvento_ = UUID_EVENTO_UNIRSE_A_PARTIDA;
  this→uuidPartida_ = uuidPartida;
}
void EventoUnirseAPartida::enviarse(Protocolo &protocolo) {
  protocolo.enviar(UUIDEvento_);
  protocolo.enviar(uuidPartida_);
}

void EventoUnirseAPartida::actualizar(Handler &handler) {
  handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoSnapshotSala.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoSnapshotSala.h"

#include "includes/common/Handler.h"

EventoSnapshotSala::EventoSnapshotSala(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
    cantidadPartidas_ = protocolo.recibirNumUnsigned16();
    for (uint16_t ordinal = 0; ordinal < cantidadPartidas_; ordinal++) {
        uint16_t uuidPartida = protocolo.recibirNumUnsigned16();
        ordinalAuuidPartida_.emplace(ordinal, uuidPartida);
    }
}

EventoSnapshotSala::EventoSnapshotSala(std::map<uint16_t, uint16_t>∧ datos) :
    Evento(0),
    cantidadPartidas_(datos.size()) {

    UUIDEvento_ = UUID_EVENTO_SNAPSHOT_SALA;
    ordinalAuuidPartida_ = std::move(datos);
}

void EventoSnapshotSala::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
    protocolo.enviar(cantidadPartidas_);
    for (const auto& kv : ordinalAuuidPartida_) {
        protocolo.enviar(kv.second);
    }
}

void EventoSnapshotSala::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoSnapshotLobby.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoSnapshotLobby.h"

#include "includes/common/Handler.h"

EventoSnapshotLobby::EventoSnapshotLobby(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
    cantidadJugadores_ = protocolo.recibirNumUnsigned8();
    for (size_t i = 0; i < cantidadJugadores_; i++) {
        uint32_t idJugador = protocolo.recibirNumUnsigned32();
        //TODO: Que protocolo pueda enviar/recibir bool
        uint8_t estaListoRed = protocolo.recibirNumUnsigned8();
        bool estaListo = (estaListoRed ≡ 1 ? true : false);
        idJugadorAEstaListo_.emplace(idJugador, estaListo);
    }
}

EventoSnapshotLobby::EventoSnapshotLobby(std::map<uint32_t, bool>∧ datos) :
    Evento(0),
    cantidadJugadores_(datos.size()) {

    UUIDEvento_ = UUID_EVENTO_SNAPSHOT_LOBBY;
    idJugadorAEstaListo_ = std::move(datos);
}

void EventoSnapshotLobby::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
    protocolo.enviar(cantidadJugadores_);
    for (const auto& kv : idJugadorAEstaListo_) {
        protocolo.enviar(kv.first);
        uint8_t estaListoRed = (kv.second ? 1 : 0);
        protocolo.enviar(estaListoRed);
    }
}

void EventoSnapshotLobby::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoSnapshot.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoSnapshot.h"

#include "includes/common/Handler.h"

EventoSnapshot::EventoSnapshot(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
    cantidadVehiculos_ = protocolo.recibirNumUnsigned8();
    for (size_t i = 0; i < cantidadVehiculos_; i++) {
        uint8_t id = protocolo.recibirNumUnsigned8();
        uint16_t xCoordRed = protocolo.recibirNumUnsigned32();
        float xCoord = (float)xCoordRed / 100.0f;
        uint16_t yCoordRed = protocolo.recibirNumUnsigned32();
        float yCoord = (float)yCoordRed / 100.0f;
        uint16_t angulo = protocolo.recibirNumUnsigned16();
        uint8_t salud = protocolo.recibirNumUnsigned8();
        uint8_t visible = protocolo.recibirNumUnsigned8();
        idsADatosVehiculos_.emplace(id, datosVehiculo_{
            xCoord,
            yCoord,
            angulo,
            salud,
            visible
        });
    }
}

EventoSnapshot::EventoSnapshot(std::map<uint8_t, datosVehiculo_>∧ datos) :
    Evento(0),
    cantidadVehiculos_(datos.size()) {

    UUIDEvento_ = UUID_EVENTO_SNAPSHOT;
    idsADatosVehiculos_ = std::move(datos);
}

void EventoSnapshot::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
    enviarSoloDatos(protocolo);
}

void EventoSnapshot::actualizar(Handler& handler) {
    handler.manejar(*this);
}

void EventoSnapshot::enviarSoloDatos(Protocolo& protocolo) {
    protocolo.enviar(cantidadVehiculos_);
    for (const auto& kv : idsADatosVehiculos_) {
        //ID
        protocolo.enviar(kv.first);
        //DATOS
        uint32_t xCoordRed = 100 * kv.second.xCoord_;
        protocolo.enviar(xCoordRed);
        uint32_t yCoordRed = 100 * kv.second.yCoord_;
        protocolo.enviar(yCoordRed);
        protocolo.enviar(kv.second.angulo_);
        protocolo.enviar(kv.second.salud_);
        protocolo.enviar(kv.second.visible_);
    }
}
```

nov 26, 19 17:34 **EventoPartidaIniciada.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoPartidaIniciada.h"

#include "includes/common/Handler.h"

EventoPartidaIniciada::EventoPartidaIniciada(uint32_t uuidRemitente, Protocolo&
protocolo) :
    Evento(uuidRemitente),
    estadoInicial_(uuidRemitente, protocolo) {
        idDelVehiculo_ = protocolo.recibirNumUnsigned8();
}

EventoPartidaIniciada::EventoPartidaIniciada(uint8_t idDelVehiculo, std::map<uin
t8_t, datosVehiculo_>∧ datos) :
    Evento(0),
    idDelVehiculo_(idDelVehiculo),
    estadoInicial_(std::move(datos)) {
    UUIDEvento_ = UUID_EVENTO_PARTIDA_INICIADA;
}

void EventoPartidaIniciada::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
    estadoInicial_.enviarSoloDatos(protocolo);
    protocolo.enviar(idDelVehiculo_);
}

void EventoPartidaIniciada::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoPartidaCreada.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoPartidaCreada.h"

#include "includes/common/Handler.h"

EventoPartidaCreada::EventoPartidaCreada(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
    uuidPartida_ = protocolo.recibirNumUnsigned16();
    uuidCreador_ = protocolo.recibirNumUnsigned32();
}

EventoPartidaCreada::EventoPartidaCreada(uint16_t uuidPartida, uint32_t uuidCreador) :
    Evento(0),
    uuidPartida_(uuidPartida),
    uuidCreador_(uuidCreador) {

    UUIDEvento_ = UUID_EVENTO_PARTIDA_CREADA;
}

void EventoPartidaCreada::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
    protocolo.enviar(uuidPartida_);
    protocolo.enviar(uuidCreador_);
}

void EventoPartidaCreada::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoIniciarPartida.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoIniciarPartida.h"

#include "includes/common/Handler.h"

EventoIniciarPartida::EventoIniciarPartida(uint32_t uuidRemitente, Protocolo &protocolo) :
    Evento(uuidRemitente) {
}

EventoIniciarPartida::EventoIniciarPartida() : Evento(0) {
  this→UUIDEvento_ = UUID_EVENTO_INICIAR_PARTIDA;
}

void EventoIniciarPartida::enviarse(Protocolo &protocolo) {
  protocolo.enviar(UUIDEvento_);
}

void EventoIniciarPartida::actualizar(Handler &handler) {
  handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoFrenar.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoFrenar.h"

#include "includes/common/Handler.h"

EventoFrenar::EventoFrenar(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
}

EventoFrenar::EventoFrenar() :
    Evento(0) {
    UUIDEvento_ = UUID_EVENTO_FRENAR;
}

void EventoFrenar::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoFrenar::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoFrenada.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoFrenada.h"

#include "includes/common/Handler.h"

EventoFrenada::EventoFrenada(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {

    uint16_t xCoordRed = protocolo.recibirNumUnsigned32();
    xCoord_ = (float)xCoordRed / 100.0f;
    uint16_t yCoordRed = protocolo.recibirNumUnsigned32();
    yCoord_ = (float)yCoordRed / 100.0f;
}

EventoFrenada::EventoFrenada(float xCoord, float yCoord) :
    Evento(0),
    xCoord_(xCoord),
    yCoord_(yCoord) {

    UUIDEvento_ = UUID_EVENTO_FRENADA;
}

void EventoFrenada::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
    uint32_t xCoordRed = 100 * xCoord_;
    protocolo.enviar(xCoordRed);
    uint32_t yCoordRed = 100 * yCoord_;
    protocolo.enviar(yCoordRed);
}

void EventoFrenada::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoFinCarrera.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoFinCarrera.h"

#include "includes/common/Handler.h"

EventoFinCarrera::EventoFinCarrera(std::vector<uint8_t>^ podio) :
    Evento(0) {
    UUIDEvento_ = UUID_EVENTO_FIN_CARRERA;
    podio_ = std::move(podio);
}

EventoFinCarrera::EventoFinCarrera(uint32_t uuidRemitente, Protocolo &protocolo)
 :
    Evento(uuidRemitente) {
  uint8_t cantidad = protocolo.recibirNumUnsigned8();
  for (uint8_t i = 0; i < cantidad; ++i) {
    uint8_t idVehiculo = protocolo.recibirNumUnsigned8();
    podio_.emplace_back(idVehiculo);
  }
}

void EventoFinCarrera::enviarse(Protocolo &protocolo) {
  protocolo.enviar(UUIDEvento_);
  protocolo.enviar((uint8_t)podio_.size());
  for (uint8_t idVehiculo: podio_) {
    protocolo.enviar(idVehiculo);
  }
}

void EventoFinCarrera::actualizar(Handler &handler) {
  handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoFinBarro.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoFinBarro.h"

#include "includes/common/Handler.h"

EventoFinBarro::EventoFinBarro(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
}

EventoFinBarro::EventoFinBarro() :
    Evento(0) {

    UUIDEvento_ = UUID_EVENTO_FIN_BARRO;
}

void EventoFinBarro::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoFinBarro::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoFactory.cpp** Page 1/2

```cpp
#include "includes/common/eventos/EventoFactory.h"

#include "includes/common/excepciones/EventoDesconocidoError.h"

std::shared_ptr<Evento> EventoFactory::instanciar(uint32_t uuidRemitente, Protocolo &protocolo) {
  uint16_t UUIDEvento = protocolo.recibirNumUnsigned16();
  switch (UUIDEvento) {

    case UUID_EVENTO_CREAR_PARTIDA:
      return std::make_shared<EventoCrearPartida>(uuidRemitente, protocolo);

    case UUID_EVENTO_UNIRSE_A_PARTIDA:
      return std::make_shared<EventoUnirseAPartida>(uuidRemitente, protocolo);

    case UUID_EVENTO_INICIAR_PARTIDA:
      return std::make_shared<EventoIniciarPartida>(uuidRemitente, protocolo);

    case UUID_EVENTO_DESCONEXION:
      return std::make_shared<EventoDesconexion>(uuidRemitente, protocolo);

    case UUID_EVENTO_ACELERAR:
      return std::make_shared<EventoAcelerar>(uuidRemitente, protocolo);

    case UUID_EVENTO_DESACELERAR:
      return std::make_shared<EventoDesacelerar>(uuidRemitente, protocolo);

    case UUID_EVENTO_FRENAR:
      return std::make_shared<EventoFrenar>(uuidRemitente, protocolo);

    case UUID_EVENTO_DEJAR_DE_FRENAR:
      return std::make_shared<EventoDejarDeFrenar>(uuidRemitente, protocolo);

    case UUID_EVENTO_DOBLAR_IZQUIERDA:
      return std::make_shared<EventoDoblarIzquierda>(uuidRemitente, protocolo);

    case UUID_EVENTO_DEJAR_DE_DOBLAR_IZQUIERDA:
      return std::make_shared<EventoDejarDeDoblarIzquierda>(uuidRemitente, protocolo);

    case UUID_EVENTO_DOBLAR_DERECHA:
      return std::make_shared<EventoDoblarDerecha>(uuidRemitente, protocolo);

    case UUID_EVENTO_DEJAR_DE_DOBLAR_DERECHA:
      return std::make_shared<EventoDejarDeDoblarDerecha>(uuidRemitente, protocolo);

    case UUID_EVENTO_SNAPSHOT:
      return std::make_shared<EventoSnapshot>(uuidRemitente, protocolo);

    case UUID_EVENTO_PARTIDA_INICIADA:
      return std::make_shared<EventoPartidaIniciada>(uuidRemitente, protocolo);

    case UUID_EVENTO_FIN_CARRERA:
      return std::make_shared<EventoFinCarrera>(uuidRemitente, protocolo);

    case UUID_EVENTO_APARECIO_CONSUMIBLE:
      return std::make_shared<EventoAparecioConsumible>(uuidRemitente, protocolo);

    case UUID_EVENTO_BARRO_PISADO:
      return std::make_shared<EventoBarroPisado>(uuidRemitente, protocolo);

    case UUID_EVENTO_CHOQUE:
      return std::make_shared<EventoChoque>(uuidRemitente, protocolo);

```

nov 26, 19 17:34 **EventoFactory.cpp** Page 2/2

```cpp
    case UUID_EVENTO_DESAPARECIO_CONSUMIBLE:
      return std::make_shared<EventoDesaparecioConsumible>(uuidRemitente, protocolo);

    case UUID_EVENTO_EXPLOSION:
      return std::make_shared<EventoExplosion>(uuidRemitente, protocolo);

    case UUID_EVENTO_FIN_BARRO:
      return std::make_shared<EventoFinBarro>(uuidRemitente, protocolo);

    case UUID_EVENTO_PARTIDA_CREADA:
      return std::make_shared<EventoPartidaCreada>(uuidRemitente, protocolo);

    case UUID_EVENTO_SNAPSHOT_LOBBY:
      return std::make_shared<EventoSnapshotLobby>(uuidRemitente, protocolo);

    case UUID_EVENTO_SNAPSHOT_SALA:
      return std::make_shared<EventoSnapshotSala>(uuidRemitente, protocolo);

    case UUID_EVENTO_UNIRSE_A_SALA:
      return std::make_shared<EventoUnirseASala>(uuidRemitente, protocolo);

    case UUID_EVENTO_FRENADA:
      return std::make_shared<EventoFrenada>(uuidRemitente, protocolo);

    default:
      throw EventoDesconocidoError(ERROR_EVENTO_DESCONOCIDO);
  }
}
```

nov 26, 19 17:34 **EventoExplosion.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoExplosion.h"

#include "includes/common/Handler.h"

EventoExplosion::EventoExplosion(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {

    uint16_t xCoordRed = protocolo.recibirNumUnsigned32();
    xCoord_ = (float)xCoordRed / 100.0f;
    uint16_t yCoordRed = protocolo.recibirNumUnsigned32();
    yCoord_ = (float)yCoordRed / 100.0f;
}

EventoExplosion::EventoExplosion(float xCoord, float yCoord) :
    Evento(0),
    xCoord_(xCoord),
    yCoord_(yCoord) {

    UUIDEvento_ = UUID_EVENTO_EXPLOSION;
}

void EventoExplosion::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
    uint32_t xCoordRed = 100 * xCoord_;
    protocolo.enviar(xCoordRed);
    uint32_t yCoordRed = 100 * yCoord_;
    protocolo.enviar(yCoordRed);
}

void EventoExplosion::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoDoblarIzquierda.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoDoblarIzquierda.h"

#include "includes/common/Handler.h"

EventoDoblarIzquierda::EventoDoblarIzquierda(uint32_t uuidRemitente, Protocolo&
protocolo) :
    Evento(uuidRemitente) {
}

EventoDoblarIzquierda::EventoDoblarIzquierda() :
    Evento(0) {
    UUIDEvento_ = UUID_EVENTO_DOBLAR_IZQUIERDA;
}

void EventoDoblarIzquierda::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoDoblarIzquierda::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoDoblarDerecha.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoDoblarDerecha.h"

#include "includes/common/Handler.h"

EventoDoblarDerecha::EventoDoblarDerecha(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
}

EventoDoblarDerecha::EventoDoblarDerecha() :
    Evento(0) {
    UUIDEvento_ = UUID_EVENTO_DOBLAR_DERECHA;
}

void EventoDoblarDerecha::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoDoblarDerecha::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoDesconexion.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoDesconexion.h"

#include "includes/common/Handler.h"

EventoDesconexion::EventoDesconexion(uint32_t uuidRemitente) :
    Evento(uuidRemitente) {

}

EventoDesconexion::EventoDesconexion(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
}

void EventoDesconexion::enviarse(Protocolo& protocolo) {

}

void EventoDesconexion::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoDesaparecioConsumible.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoDesaparecioConsumible.h"

#include "includes/common/Handler.h"

EventoDesaparecioConsumible::EventoDesaparecioConsumible(uint32_t uuidRemitente,
 Protocolo& protocolo) :
    Evento(uuidRemitente) {
    uuidConsumible_ = protocolo.recibirNumUnsigned8();
}

EventoDesaparecioConsumible::EventoDesaparecioConsumible(uint8_t uuidConsumible)
 :
    Evento(0),
    uuidConsumible_(uuidConsumible) {

    UUIDEvento_ = UUID_EVENTO_DESAPARECIO_CONSUMIBLE;
}

void EventoDesaparecioConsumible::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
    protocolo.enviar(uuidConsumible_);
}

void EventoDesaparecioConsumible::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoDesacelerar.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoDesacelerar.h"

#include "includes/common/Handler.h"

EventoDesacelerar::EventoDesacelerar(uint32_t uuidRemitente, Protocolo& protocol
o) :
    Evento(uuidRemitente) {
}

EventoDesacelerar::EventoDesacelerar() :
    Evento(0) {
    UUIDEvento_ = UUID_EVENTO_DESACELERAR;
}

void EventoDesacelerar::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoDesacelerar::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoDejarDeFrenar.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoDejarDeFrenar.h"

#include "includes/common/Handler.h"

EventoDejarDeFrenar::EventoDejarDeFrenar(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
}

EventoDejarDeFrenar::EventoDejarDeFrenar() :
    Evento(0) {
    UUIDEvento_ = UUID_EVENTO_DEJAR_DE_FRENAR;
}

void EventoDejarDeFrenar::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoDejarDeFrenar::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoDejarDeDoblarIzquierda.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoDejarDeDoblarIzquierda.h"

#include "includes/common/Handler.h"

EventoDejarDeDoblarIzquierda::EventoDejarDeDoblarIzquierda(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
}

EventoDejarDeDoblarIzquierda::EventoDejarDeDoblarIzquierda() :
    Evento(0) {
    UUIDEvento_ = UUID_EVENTO_DEJAR_DE_DOBLAR_IZQUIERDA;
}

void EventoDejarDeDoblarIzquierda::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoDejarDeDoblarIzquierda::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoDejarDeDoblarDerecha.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoDejarDeDoblarDerecha.h"

#include "includes/common/Handler.h"

EventoDejarDeDoblarDerecha::EventoDejarDeDoblarDerecha(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
}

EventoDejarDeDoblarDerecha::EventoDejarDeDoblarDerecha() :
    Evento(0) {
    UUIDEvento_ = UUID_EVENTO_DEJAR_DE_DOBLAR_DERECHA;
}

void EventoDejarDeDoblarDerecha::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoDejarDeDoblarDerecha::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoCrearPartida.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoCrearPartida.h"

#include "includes/common/Handler.h"

EventoCrearPartida::EventoCrearPartida(uint32_t uuidRemitente, Protocolo &protocolo) :
    Evento(uuidRemitente) {
}

EventoCrearPartida::EventoCrearPartida() : Evento(0) {
  this→UUIDEvento_ = UUID_EVENTO_CREAR_PARTIDA;
}

void EventoCrearPartida::enviarse(Protocolo &protocolo) {
  protocolo.enviar(UUIDEvento_);
}

void EventoCrearPartida::actualizar(Handler &handler) {
  handler.manejar(*this);
}
```

nov 26, 19 17:34 **Evento.cpp** Page 1/1

```cpp
#include "includes/common/eventos/Evento.h"

Evento::Evento(uint32_t uuidRemitente) :
    UUIDRemitente_(uuidRemitente) {
}

uint32_t Evento::uuidRemitente() {
    return UUIDRemitente_;
}

void Evento::setRemitente(uint32_t uuidRemitente) {
    UUIDRemitente_ = uuidRemitente;
}
```

nov 26, 19 17:34 **EventoChoque.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoChoque.h"

#include "includes/common/Handler.h"

EventoChoque::EventoChoque(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {

    uint16_t xCoordRed = protocolo.recibirNumUnsigned32();
    xCoord_ = (float)xCoordRed / 100.0f;
    uint16_t yCoordRed = protocolo.recibirNumUnsigned32();
    yCoord_ = (float)yCoordRed / 100.0f;
}

EventoChoque::EventoChoque(float xCoord, float yCoord) :
    Evento(0),
    xCoord_(xCoord),
    yCoord_(yCoord) {

    UUIDEvento_ = UUID_EVENTO_CHOQUE;
}

void EventoChoque::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
    uint32_t xCoordRed = 100 * xCoord_;
    protocolo.enviar(xCoordRed);
    uint32_t yCoordRed = 100 * yCoord_;
    protocolo.enviar(yCoordRed);
}

void EventoChoque::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoBarroPisado.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoBarroPisado.h"

#include "includes/common/Handler.h"

EventoBarroPisado::EventoBarroPisado(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
}

EventoBarroPisado::EventoBarroPisado() :
    Evento(0) {

    UUIDEvento_ = UUID_EVENTO_BARRO_PISADO;
}

void EventoBarroPisado::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoBarroPisado::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoAparecioConsumible.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoAparecioConsumible.h"

#include "includes/common/Handler.h"

EventoAparecioConsumible::EventoAparecioConsumible(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
    //TODO: QUE EL PROTOCOLO ENVIE FLOATS
    uuidConsumible_ = protocolo.recibirNumUnsigned8();
    tipoConsumible_ = protocolo.recibirNumUnsigned8();
    uint16_t xCoordRed = protocolo.recibirNumUnsigned32();
    xCoord_ = (float)xCoordRed / 100.0f;
    uint16_t yCoordRed = protocolo.recibirNumUnsigned32();
    yCoord_ = (float)yCoordRed / 100.0f;
}

EventoAparecioConsumible::EventoAparecioConsumible(uint8_t uuidConsumible, uint8_t tipoConsumible, float xCoord, float yCoord) :
    Evento(0),
    uuidConsumible_(uuidConsumible),
    tipoConsumible_(tipoConsumible),
    xCoord_(xCoord),
    yCoord_(yCoord) {

    UUIDEvento_ = UUID_EVENTO_APARECIO_CONSUMIBLE;
}

void EventoAparecioConsumible::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
    protocolo.enviar(uuidConsumible_);
    protocolo.enviar(tipoConsumible_);
    uint32_t xCoordRed = 100 * xCoord_;
    protocolo.enviar(xCoordRed);
    uint32_t yCoordRed = 100 * yCoord_;
    protocolo.enviar(yCoordRed);
}

void EventoAparecioConsumible::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EventoAcelerar.cpp** Page 1/1

```cpp
#include "includes/common/eventos/EventoAcelerar.h"

#include "includes/common/Handler.h"

EventoAcelerar::EventoAcelerar(uint32_t uuidRemitente, Protocolo& protocolo) :
    Evento(uuidRemitente) {
}

EventoAcelerar::EventoAcelerar() :
    Evento(0) {
    UUIDEvento_ = UUID_EVENTO_ACELERAR;
}

void EventoAcelerar::enviarse(Protocolo& protocolo) {
    protocolo.enviar(UUIDEvento_);
}

void EventoAcelerar::actualizar(Handler& handler) {
    handler.manejar(*this);
}
```

nov 26, 19 17:34 **EnviadorEventos.cpp** Page 1/1

```cpp
#include "includes/common/EnviadorEventos.h"

#include <iostream>

EnviadorEventos::EnviadorEventos(SocketTCP& socketDestino, ColaBloqueante<std::shared_ptr<Evento>>& origen) :
    origen_(origen),
    protocolo_(socketDestino) {
}

void EnviadorEventos::correr() {
    bool obtenido;
    std::shared_ptr<Evento> eventoAEnviar;
    while(seguirCorriendo_ ∧ (obtenido = origen_.get(eventoAEnviar))) {
        try {
            eventoAEnviar→enviarse(protocolo_);
        }
        catch (const std::exception& e) {
            std::cerr << e.what() << '\n';
            break;
        }
    }
}

void EnviadorEventos::detener() {
    seguirCorriendo_ = false;
}
```

nov 26, 19 17:34 **Cronometro.cpp** Page 1/1

```cpp
#include "includes/common/Cronometro.h"

#include <chrono>

double Cronometro::ahora() {
    auto ahora = std::chrono::time_point_cast<std::chrono::milliseconds>(std::chrono::high_resolution_clock::now());
    return ahora.time_since_epoch().count();
}
```

nov 26, 19 17:34 **Conversor.cpp** Page 1/1

```cpp
#include "includes/common/Conversor.h"
#include <math.h>

Conversor::Conversor(float pixelPorMetro, int pixelPorBloque) :
    pixelPorMetro(pixelPorMetro), pixelPorBloque(pixelPorBloque) {}

int Conversor::metroAPixel(float coord) {
  return round(coord * pixelPorMetro);
}

float Conversor::pixelAMetro(int coord) {
  return (float) (coord / pixelPorMetro);
}

int Conversor::bloqueAPixel(int coord) {
  return coord * pixelPorBloque;
}

int Conversor::pixelABloque(int coord) {
  return trunc((float) coord / pixelPorBloque);
}
```

```cpp
#include "includes/cliente/utils/ConfigCliente.h"

#include <fstream>

ConfigCliente &ConfigCliente::instancia() {
  static ConfigCliente config_(RUTA_CONFIG_CLIENTE);
  return config_;
}

ConfigCliente::ConfigCliente(const std::string &rutaArchivo) {
  std::ifstream archivo(rutaArchivo);
  archivo >> json_;
}

unsigned int ConfigCliente::anchoVentana() {
  return json_["ventana"]["ancho"].get<unsigned int>();
}

unsigned int ConfigCliente::altoVentana() {
  return json_["ventana"]["alto"].get<unsigned int>();
}

float ConfigCliente::factorLejaniaCamara() {
  return json_["factorLejaniaPista"].get<float>();
}

bool ConfigCliente::pantallaCompleta() {
  return json_["ventana"]["pantallaCompleta"].get<bool>();
}

std::string ConfigCliente::tituloVentana() {
  return std::move(json_["ventana"]["titulo"].get<std::string>());
}

unsigned int ConfigCliente::fps() {
  return json_["ventana"]["fps"].get<unsigned int>();
}

std::string ConfigCliente::host() {
  return std::move(json_["red"]["host"].get<std::string>());
}

std::string ConfigCliente::puerto() {
  return std::move(json_["red"]["puerto"].get<std::string>());
}

std::string ConfigCliente::fuente() {
  return std::move(json_["fuente"]["path"].get<std::string>());
}

std::string ConfigCliente::musicaAmbiente() {
  return std::move(json_["musica"]["ambiente"].get<std::string>());
}

std::string ConfigCliente::musicaMotor() {
  return std::move(json_["musica"]["motor"].get<std::string>());
}

std::string ConfigCliente::musicaExplosion() {
  return std::move(json_["musica"]["explosion"].get<std::string>());
}

std::string ConfigCliente::musicaVacio() {
  return std::move(json_["musica"]["empty"].get<std::string>());
}

```



```cpp
std::string ConfigCliente::musicaChoque() {
  return std::move(json_["musica"]["choque"].get<std::string>());
}

std::string ConfigCliente::musicaFrenada() {
  return std::move(json_["musica"]["frenada"].get<std::string>());
}

std::string ConfigCliente::rutaLuaScriptUsuario() {
  return std::move(json_["lua"]["ruta2"].get<std::string>());
}

std::string ConfigCliente::rutaLuaScript() {
  return std::move(json_["lua"]["ruta"].get<std::string>());
}

int ConfigCliente::tiempoReaccionHumano() {
  return json_["lua"]["tiempoHumano"].get<int>();
}

unsigned int ConfigCliente::volumenAmbiente() {
  return json_["volumen"]["ambiente"].get<unsigned int>();
}

unsigned int ConfigCliente::anchoBloquesPista() {
  return json_["bloques"]["ancho"].get<unsigned int>();
}

unsigned int ConfigCliente::altoBloquesPista() {
  return json_["bloques"]["alto"].get<unsigned int>();
}

double ConfigCliente::pixelPorMetro() {
  return json_["conversor"]["pixelPorMetro"].get<double>();
}

unsigned int ConfigCliente::pixelPorBloque() {
  return json_["conversor"]["pixelPorBloque"].get<unsigned int>();
}

std::string ConfigCliente::texto(std::string sector) {
  return std::move(json_["interfaz"][sector]["texto"].get<std::string>());
}

unsigned int ConfigCliente::tamanioTexto(std::string sector) {
  return json_["interfaz"][sector]["tamaÃ±oTexto"].get<unsigned int>();
}

unsigned int ConfigCliente::anchoTexto(std::string sector) {
  return json_["interfaz"][sector]["anchoTexto"].get<unsigned int>();
}

double ConfigCliente::margenX(std::string sector) {
  return json_["interfaz"][sector]["margenX"].get<double>();
}

double ConfigCliente::margenY(std::string sector) {
  return json_["interfaz"][sector]["margenY"].get<double>();
}

unsigned int ConfigCliente::uuid(std::string nombreAnimacion) {
  return json_["animaciones"][nombreAnimacion]["uuid"].get<unsigned int>();
}

unsigned int ConfigCliente::ancho(std::string nombreAnimacion) {
  return json_["animaciones"][nombreAnimacion]["ancho"].get<unsigned int>();
```

```
133 }
134
135 unsigned int ConfigCliente::alto(std::string nombreAnimacion) {
136   return json_["animaciones"][nombreAnimacion]["alto"].get<unsigned int>();
137 }
138
139 std::vector<std::string> ConfigCliente::sprites(std::string nombreAnimacion) {
140   return std::move(json_["animaciones"][nombreAnimacion]["sprites"].get<std::vector<
141       std::string>>());
142 }
143
144 unsigned int ConfigCliente::anchoGrabadora() {
145   return json_["grabadora"]["resolucion"]["ancho"].get<unsigned int>();
146 }
147
148 unsigned int ConfigCliente::altoGrabadora() {
149   return json_["grabadora"]["resolucion"]["alto"].get<unsigned int>();
150 }
151
152 std::string ConfigCliente::formatoGrabadora() {
153   return std::move(json_["grabadora"]["formato"].get<std::string>());
154 }
155
156 unsigned int ConfigCliente::fpsGrabadora() {
157   return json_["grabadora"]["fps"].get<unsigned int>();
158 }
159
160 uint32_t ConfigCliente::bitrateGrabadora() {
161   return json_["grabadora"]["bitrate"].get<uint32_t>();
162 }
```



```
1  #include "includes/cliente/red/SocketTCPCliente.h"
2
3  #include <stdexcept>
4
5  SocketTCPCliente::SocketTCPCliente(const std::string& unHost, const std::string&
    unPuerto) :
6      SocketTCP(unHost, unPuerto) {
7  }
8
9  void SocketTCPCliente::conectar() {
10     bool conectado = false;
11     int estado = -1;
12     addrinfo* ptr = hints_;
13     while (ptr ≠ NULL ∧ ¬conectado) {
14         estado = connect(fileDescripor_, hints_→ai_addr, hints_→ai_addrlen);
15         conectado = (estado ≠ -1);
16         ptr = ptr→ai_next;
17     }
18     if (¬conectado) {
19         throw std::runtime_error(ERROR_CONEXION);
20     }
21
22 }
```

nov 26, 19 17:34 **main_cliente.cpp** Page 1/1

```cpp
#include <iostream>

#include "includes/cliente/Cliente.h"
#include "includes/cliente/utils/ConfigCliente.h"

int main(int argc, char *argv[]) {
    Cliente cliente(CONFIG_CLIENTE.anchoVentana(), CONFIG_CLIENTE.altoVentana(),
 CONFIG_CLIENTE.pantallaCompleta(), CONFIG_CLIENTE.tituloVentana(), CONFIG_CLIEN
TE.host(), CONFIG_CLIENTE.puerto());
    try {
        cliente.correr();
    } catch(const std::exception& e) {
        std::cout << e.what() << '\n';
        return -1;
    }
    cliente.cerrar();
    return 0;
}
```

nov 26, 19 17:34 **Jugador.cpp** Page 1/1

```cpp
#include "includes/cliente/jugadores/Jugador.h"

Jugador::Jugador(ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar)
                    : eventosAEnviar_(eventosAEnviar) {}

 void Jugador::setEstado(float x, float y, uint16_t angulo){}
 void Jugador::empezar(){}
 void Jugador::terminar(){}


 void Jugador::acelerar(){
   std::shared_ptr<Evento>
      eventoAcelerar = std::make_shared<EventoAcelerar>();
  eventosAEnviar_.put(eventoAcelerar);
}

 void Jugador::desacelerar(){
    std::shared_ptr<Evento>
      eventoDesacelerar = std::make_shared<EventoDesacelerar>();
  eventosAEnviar_.put(eventoDesacelerar);
}

 void Jugador::frenar(){
  std::shared_ptr<Evento>
        eventoFrenar = std::make_shared<EventoFrenar>();
  eventosAEnviar_.put(eventoFrenar);
}

 void Jugador::dejarDeFrenar(){
  std::shared_ptr<Evento>
    eventoDejarDeFrenar = std::make_shared<EventoDejarDeFrenar>();
  eventosAEnviar_.put(eventoDejarDeFrenar);
}

 void Jugador::doblarDerecha(){
  std::shared_ptr<Evento>
        eventoDoblarDerecha = std::make_shared<EventoDoblarDerecha>();
  eventosAEnviar_.put(eventoDoblarDerecha);
}

 void Jugador::dejarDeDoblarDerecha(){
  std::shared_ptr<Evento>
    eventoDejarDeDoblarDerecha = std::make_shared<EventoDejarDeDoblarDerecha>();
  eventosAEnviar_.put(eventoDejarDeDoblarDerecha);
}

 void Jugador::doblarIzquierda(){
  std::shared_ptr<Evento>
        eventoDoblarIzquierda = std::make_shared<EventoDoblarIzquierda>();
  eventosAEnviar_.put(eventoDoblarIzquierda);
}

 void Jugador::dejarDeDoblarIzquierda(){
  std::shared_ptr<Evento>
    eventoDejarDeDoblarIzquierda = std::make_shared<EventoDejarDeDoblarIzquierda
>();
  eventosAEnviar_.put(eventoDejarDeDoblarIzquierda);
}
```

```
1  #include <fstream>
2
3  #include "includes/cliente/jugadores/Computadora.h"
4
5  #include "includes/3rd−party/jsoncpp/json.hpp"
6  #include "includes/cliente/utils/ConfigCliente.h"
7  #include "includes/common/Cronometro.h"
8
9
10 Computadora::Computadora(ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar
   ,
11                          std::string fileName) :
12                              Jugador(eventosAEnviar){
13     std::string rutaPista = std::move(fileName);
14   std::ifstream archivoPista(rutaPista);
15   Json pistaJson;
16   archivoPista >> pistaJson;
17
18     int size_x = pistaJson["dimensiones"]["x"].get<uint16_t>();
19     int size_y = pistaJson["dimensiones"]["y"].get<uint16_t>();
20
21   lua.init_script(CONFIG_CLIENTE.rutaLuaScript().c_str());
22     for (uint16_t i = 0; i < size_x; i++) {
23       for (uint16_t j = 0; j < size_y; j++) {
24           int x = i;
25           int y = size_y − j − 1;
26           if (pistaJson["capas"]["pista"][std::to_string(i)][std::to_string(j)].get
   <int>() > 0){
27         lua.get_function_name("insert_road_block");
28             lua << x;
29             lua << y;
30           lua.call_function("insert_road_block", 2, 0);
31           } else {
32         lua.get_function_name("insert_bad_block");
33             lua << x;
34             lua << y;
35           lua.call_function("insert_bad_block", 2, 0);
36           }
37     }
38   }
39
40 }
41
42 void Computadora::setEstado(float x, float y, uint16_t angulo){
43   x_ = (uint16_t) x * 100;
44   y_ = (uint16_t) y * 100;
45   angulo_ = angulo;
46 }
47 void Computadora::empezar(){
48   Hilo::iniciar();
49 }
50
51 void Computadora::terminar(){
52   detener();
53   Hilo::join();
54 }
55
56 void Computadora::detener(){
57   this→seguirCorriendo_ = false;
58 }
59
60 void Computadora::correr(){
61   lua.init_script(CONFIG_CLIENTE.rutaLuaScriptUsuario().c_str());
62   double frecuencia = (double) 1 / (double) CONFIG_CLIENTE.tiempoReaccionHumano(
   );
63     int iteracion = 0;
```



```
64     frecuencia *= 1000;
65     Cronometro c;
66     double t1 = c.ahora();
67
68   while (seguirCorriendo_){
69     lua.get_function_name("get_instruction");
70     int x = std::floor(x_ /1000.0f);
71     int y = std::floor(y_/1000.0f);
72     uint16_t angulo = angulo_;
73     lua << (int) x;
74     lua << (int) y;
75     lua << (int) angulo;
76     lua.call_function("get_instruction", 3, 1);
77     do_command();
78     double t2 = c.ahora();
79         double resto = frecuencia − (t2 − t1);
80         if (resto < 0) {
81           double atraso = −resto;
82           double perdidos = atraso − std::fmod(atraso, frecuencia);
83           resto = frecuencia − std::fmod(atraso, frecuencia);
84           t1 += perdidos;
85           iteracion += std::floor(perdidos / frecuencia);
86         }
87         Hilo::dormir(resto);
88         t1 += frecuencia;
89         iteracion += 1;
90   }
91 }
92
93 void Computadora::do_command(){
94   int command  = lua.get<int>();
95   if (command ≠ last_command_){
96     leave_command();
97   }
98   if (command ≡ 0){
99     Jugador::doblarDerecha();
100  } else if (command ≡ 1){
101    Jugador::acelerar();
102  } else if (command ≡ 2){
103    Jugador::doblarIzquierda();
104  } else if (command ≡ 3){
105    Jugador::frenar();
106  }
107  last_command_ = command;
108 }
109
110
111 void Computadora::leave_command(){
112   if (last_command_ ≡ 0){
113     Jugador::dejarDeDoblarDerecha();
114   } else if (last_command_ ≡ 1){
115     Jugador::desacelerar();
116   } else if (last_command_ ≡ 2){
117     Jugador::dejarDeDoblarIzquierda();
118   } else if (last_command_ ≡ 3){
119     Jugador::dejarDeFrenar();
120   } else {}
121 }
122
123
124
125 // eventos humano
126 void Computadora::acelerar(){}
127 void Computadora::desacelerar(){}
128 void Computadora::frenar(){}
129 void Computadora::dejarDeFrenar(){}
```

```
130 void Computadora::doblarDerecha(){}
131 void Computadora::dejarDeDoblarDerecha(){}
132 void Computadora::doblarIzquierda(){}
133 void Computadora::dejarDeDoblarIzquierda(){}
```



```
1  #include "includes/cliente/GUI/Ventana.h"
2
3  #include <SDL2/SDL.h>
4  #include <SDL2/SDL_video.h>
5
6  #include "includes/cliente/excepciones/SDLException.h"
7
8  Ventana::Ventana(unsigned int ancho, unsigned int alto, bool pantallaCompleta, c
   onst std::string& tituloVentana) :
9      ancho_(ancho),
10     alto_(alto),
11     fullscreen_(false) {
12
13     int errCode = SDL_Init(SDL_INIT_VIDEO);
14     if (errCode) {
15         throw SDLException("Error en la inicializaciÃ³n", SDL_GetError());
16     }
17     if (pantallaCompleta) {
18         ventanaSDL_ = crearConFullScreen(ancho, alto, tituloVentana);
19     } else {
20         ventanaSDL_ = crearSinFullScreen(ancho, alto, tituloVentana);
21     }
22 }
23
24 Ventana::~Ventana() {
25     SDL_DestroyWindow(ventanaSDL_);
26     SDL_Quit();
27 }
28
29 //TODO: No tiene mucho sentido el 0, 0
30 Area Ventana::dimensiones() {
31     return Area(0, 0, ancho_, alto_);
32 }
33
34 unsigned int Ventana::ancho() {
35     return ancho_;
36 }
37
38 unsigned int Ventana::alto() {
39     return alto_;
40 }
41
42 void Ventana::toggleFullScreen() {
43     if (¬fullscreen_) {
44         SDL_SetWindowFullscreen(ventanaSDL_, SDL_WINDOW_FULLSCREEN);
45     } else {
46         SDL_SetWindowFullscreen(ventanaSDL_, 0);
47     }
48     fullscreen_ = ¬fullscreen_;
49 }
50
51 SDL_Window* Ventana::getSDL() {
52     return ventanaSDL_;
53 }
54
55 SDL_Window* Ventana::crearSinFullScreen(unsigned int ancho, unsigned int alto, c
   onst std::string& tituloVentana) {
56     ventanaSDL_ = SDL_CreateWindow(
57         tituloVentana.c_str(),
58         SDL_WINDOWPOS_UNDEFINED,
59         SDL_WINDOWPOS_UNDEFINED,
60         ancho,
61         alto,
62         SDL_WINDOW_OPENGL
63     );
64     if (¬ventanaSDL_) {
```

```
65          throw SDLException("Error en la creaciÃ³n de la ventana", SDL_GetError());
66      }
67      return ventanaSDL_;
68  }
69
70  SDL_Window* Ventana::crearConFullScreen(unsigned int ancho, unsigned int alto, c
    onst std::string& tituloVentana) {
71      ventanaSDL_ = SDL_CreateWindow(
72          tituloVentana.c_str(),
73          SDL_WINDOWPOS_UNDEFINED,
74          SDL_WINDOWPOS_UNDEFINED,
75          ancho,
76          alto,
77          SDL_WINDOW_OPENGL | SDL_WINDOW_FULLSCREEN
78      );
79      if (¬ventanaSDL_) {
80          throw SDLException("Error en la creaciÃ³n de la ventana", SDL_GetError());
81      }
82      return ventanaSDL_;
83  }
```



```
1   #include "includes/cliente/GUI/Textura.h"
2
3   #include <SDL2/SDL.h>
4   #include <SDL2/SDL_image.h>
5
6   #include "includes/cliente/GUI/Renderizador.h"
7   #include "includes/cliente/excepciones/SDLException.h"
8
9   Textura::Textura(const std::string& rutaArchivo, Renderizador& renderizador) {
10
11      texturaSDL_ = IMG_LoadTexture(renderizador.getSDL(), rutaArchivo.c_str());
12      if (¬texturaSDL_) {
13          throw SDLException("Error al cargar la textura", SDL_GetError());
14      }
15  }
16
17  Textura::Textura(Textura∧ otraTextura) {
18      this→texturaSDL_ = otraTextura.texturaSDL_;
19      otraTextura.texturaSDL_ = nullptr;
20  }
21
22  Textura& Textura::operator=(Textura∧ otraTextura) {
23      this→texturaSDL_ = otraTextura.texturaSDL_;
24      otraTextura.texturaSDL_ = nullptr;
25      return *this;
26  }
27
28  Textura::Textura(Renderizador& renderizador, Area dimensiones) {
29      texturaSDL_ = SDL_CreateTexture(renderizador.getSDL(),
30          SDL_PIXELFORMAT_RGB24,
31          SDL_TEXTUREACCESS_TARGET,
32          dimensiones.ancho(),
33          dimensiones.alto());
34      if (¬texturaSDL_) {
35          throw SDLException("Error al crear textura vacÃ-a", SDL_GetError());
36      }
37      renderizador.clear();
38  }
39
40  Textura::~Textura() {
41      if (texturaSDL_ ≠ nullptr) {
42          SDL_DestroyTexture(texturaSDL_);
43      }
44  }
45
46  SDL_Texture* Textura::getSDL() {
47      return texturaSDL_;
48  }
```

```
#include "includes/cliente/GUI/Texto.h"
#include "includes/cliente/utils/ConfigCliente.h"
#include "includes/cliente/excepciones/SDLException.h"
#include "includes/cliente/GUI/Renderizador.h"

SDL_Color Texto::getColorRGB(int uuidColor) {
  switch (uuidColor) {
    case UUID_TEXTO_BLANCO:return {255, 255, 255}; break;
    case UUID_TEXTO_NEGRO:return {0, 0, 0}; break;
    case UUID_TEXTO_ROJO:return {255, 0, 0}; break;
    case UUID_TEXTO_AMARILLO:return {255, 255, 0}; break;
    case UUID_TEXTO_VERDE: return {0, 255, 0}; break;
    default:return {255, 255, 255};
  }
}

SDL_Texture *Texto::createFromText(const std::string texto,
                                   Renderizador &renderizador,
                                   int uuidColor) {
  SDL_Surface
      *surface =
      TTF_RenderText_Blended(this→font, texto.c_str(), getColorRGB(uuidColor));
  if (¬surface)
    throw SDLException("Error con TTF_RenderText_Blended:", SDL_GetError());
  SDL_Texture
      *texture = SDL_CreateTextureFromSurface(renderizador.getSDL(), surface);
  if (¬texture) {
    throw SDLException("Error al cargar la textura", SDL_GetError());
  }
  SDL_FreeSurface(surface);
  return texture;
}

Texto::Texto(const std::string texto,
             const int size,
             Renderizador &renderizador, int uuidColor) {
  if (TTF_Init() ≡ -1)
    throw SDLException("Error al iniciar TTF:", SDL_GetError());
  this→font = TTF_OpenFont(CONFIG_CLIENTE.fuente().c_str(), size);
  if (font ≡ NULL)
    throw SDLException("Error al cargar font:", SDL_GetError());
  this→texturaSDL_ = createFromText(texto, renderizador, uuidColor);
}

SDL_Texture *Texto::getSDL() {
  return texturaSDL_;
}

void Texto::setColor(int uuidColor) {
  SDL_Color color = getColorRGB(uuidColor);
  SDL_SetTextureColorMod(this→texturaSDL_, color.r, color.g, color.b);
}

Texto::Texto(Texto ∧other) {
  this→texturaSDL_ = other.texturaSDL_;
  this→font = other.font;
  other.texturaSDL_ = nullptr;
  other.font = nullptr;
}

Texto &Texto::operator=(Texto ∧other) {
  if (this ≡ &other) {
    return *this;
  }
  if (this→texturaSDL_) {
    SDL_DestroyTexture(texturaSDL_);
```



```
  }
  if (this→font) {
    TTF_CloseFont(this→font);
    TTF_Quit();
  }
  this→texturaSDL_ = other.texturaSDL_;
  this→font = other.font;

  other.texturaSDL_ = nullptr;
  other.font = nullptr;
  return *this;
}

Texto::~Texto() {
  if (texturaSDL_ ≠ nullptr) {
    SDL_DestroyTexture(texturaSDL_);
  }
  if (font ≠ nullptr) {
    TTF_CloseFont(this→font);
    TTF_Quit();
  }
}
```

nov 26, 19 17:34 **Sonido.cpp** Page 1/1

```cpp
#include "includes/cliente/GUI/Sonido.h"

#include "includes/cliente/excepciones/SDLException.h"

Sonido::Sonido(std::string filename,bool loop): loop(loop) {
  if (SDL_Init(SDL_INIT_AUDIO) < 0) {
    throw SDLException("Error al iniciar audio con SDL", SDL_GetError());
  }
  if (Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 2048) < 0) {
    throw SDLException("Error al iniciar audio con mixer", SDL_GetError());
  }
  this→efectoSonido = Mix_LoadWAV(filename.c_str());
  if (¬this→efectoSonido) {
    throw SDLException("Error cargando sonido. ", Mix_GetError());
  }
}

void Sonido::setVolume(int volume) {
  this→efectoSonido→volume = volume;
}

void Sonido::play() {
  Mix_PlayChannel(−1, this→efectoSonido, −1 * loop);
}

void Sonido::stop() {
  Mix_HaltChannel(−1);
}

Sonido::~Sonido() {
  if (this→efectoSonido ≠ NULL)
    Mix_FreeChunk(this→efectoSonido);
  Mix_CloseAudio();
  SDL_QuitSubSystem(SDL_INIT_AUDIO);
}
```

nov 26, 19 17:34 **Renderizador.cpp** Page 1/2

```cpp
#include "includes/cliente/GUI/Renderizador.h"

#include <SDL2/SDL_render.h>
#include <SDL2/SDL_image.h>

#include "includes/cliente/GUI/Ventana.h"
#include "includes/cliente/GUI/escenas/Escena.h"

Renderizador::Renderizador(Ventana &ventana) :
    ventana_(ventana) {
  renderizadorSDL_ = SDL_CreateRenderer(ventana.getSDL(),
                                        SDL_PRIMER_DISPONIBLE,
                                        SDL_RENDERER_ACCELERATED | SDL_RENDERER_
TARGETTEXTURE);
  clear();
}

Renderizador::~Renderizador() {
  SDL_DestroyRenderer(renderizadorSDL_);
}

void Renderizador::clear() {
  SDL_SetRenderDrawColor(renderizadorSDL_, 0x33, 0x33, 0x33, 0xFF);
  SDL_RenderClear(renderizadorSDL_);
}

void Renderizador::setDestino(Textura &textura) {
  SDL_SetRenderTarget(renderizadorSDL_, textura.getSDL());
  clear();
}

void Renderizador::resetDestino() {
  SDL_SetRenderTarget(renderizadorSDL_, NULL);
  clear();
}

void Renderizador::dibujar(Textura &textura, Area &area) {
  SDL_Rect SDLDestino = {
      (int) area.x(),
      (int) area.y(),
      (int) area.ancho(),
      (int) area.alto()
  };
  SDL_RenderCopy(renderizadorSDL_, textura.getSDL(), NULL, &SDLDestino);
}

void Renderizador::dibujarTexto(Texto &texto, Area &area) {
  SDL_Rect SDLDestino = {
      (int) area.x(),
      (int) area.y(),
      (int) area.ancho(),
      (int) area.alto()
  };
  SDL_RenderCopy(renderizadorSDL_, texto.getSDL(), NULL, &SDLDestino);
}


void Renderizador::dibujar(Textura &textura, Area &area, double grados, bool fli
pVertical) {
  SDL_Rect SDLDestino = {
      (int) area.x(),
      (int) area.y(),
      (int) area.ancho(),
      (int) area.alto()
  };
  if (flipVertical) {
```

nov 26, 19 17:34 **Renderizador.cpp** Page 2/2

```
65      SDL_RenderCopyEx(renderizadorSDL_, textura.getSDL(), NULL, &SDLDestino, grad
    os, NULL, SDL_FLIP_VERTICAL);
66    } else {
67      SDL_RenderCopyEx(renderizadorSDL_, textura.getSDL(), NULL, &SDLDestino, grad
    os, NULL, SDL_FLIP_NONE);
68    }
69  }
70
71  void Renderizador::dibujar(uint32_t numeroIteracion, Escena &escena) {
72    Textura textura = escena.dibujate(numeroIteracion, ventana_.dimensiones());
73    SDL_Rect SDLDestino = {
74        0,
75        0,
76        (int) ventana_.ancho(),
77        (int) ventana_.alto()};
78    resetDestino();
79    SDL_RenderCopy(renderizadorSDL_, textura.getSDL(), NULL, &SDLDestino);
80    SDL_RenderPresent(renderizadorSDL_);
81  }
82
83  void Renderizador::dibujar(uint32_t numeroIteracion, Escena &escena, DobleBuffer
    <std::vector<char>>& buffer) {
84    Textura textura = escena.dibujate(numeroIteracion, ventana_.dimensiones());
85    SDL_Rect SDLDestino = {
86        0,
87        0,
88        (int) ventana_.ancho(),
89        (int) ventana_.alto()};
90
91    int anchoRGB = ventana_.ancho() * 3;
92    std::vector<char> pixeles(anchoRGB * ventana_.alto());
93
94    resetDestino();
95    SDL_RenderCopy(renderizadorSDL_, textura.getSDL(), NULL, &SDLDestino);
96    SDL_RenderReadPixels(renderizadorSDL_, NULL, SDL_PIXELFORMAT_RGB24, pixeles.da
    ta(), anchoRGB);
97    buffer.set(std::move(pixeles));
98    SDL_RenderPresent(renderizadorSDL_);
99  }
100
101 void Renderizador::toggleFullScreen() {
102   ventana_.toggleFullScreen();
103 }
104
105 SDL_Renderer *Renderizador::getSDL() {
106   return renderizadorSDL_;
107 }
```

nov 26, 19 17:34 **Pista.cpp** Page 1/3

```
1   #include "includes/cliente/GUI/Pista.h"
2   #include <fstream>
3
4   void Pista::agregarBloque(int capa,
5                             int x,
6                             int y,
7                             std::shared_ptr<Animacion> animacion) {
8     mapa[capa][x][y] = animacion;
9   }
10
11  void Pista::crearPista(nlohmann::json pistaJson) {
12    for (uint16_t i = 0; i < size_x; i++) {
13      for (uint16_t j = 0; j < size_y; j++) {
14        int bloqueTerreno =
15            pistaJson["capas"]["terreno"][std::to_string(i)][std::to_string(j)].get<
16                int>();
17        if (bloqueTerreno ≠ -1) {
18          if (texturas.find(bloqueTerreno) ≡ texturas.end()) {
19            texturas.insert(std::pair<int, std::shared_ptr<Animacion>>(
20                bloqueTerreno,
21                std::make_shared<Animacion>(AnimacionFactory::instanciar(
22                    bloqueTerreno,
23                    renderizador))));
24          }
25          agregarBloque(0, i, j, texturas.at(bloqueTerreno));
26        }
27      }
28      for (uint16_t i = 0; i < size_x; i++) {
29        for (uint16_t j = 0; j < size_y; j++) {
30          int bloqueTerreno =
31              pistaJson["capas"]["pista"][std::to_string(i)][std::to_string(j)].get<
32                  int>();
33          if (bloqueTerreno ≠ -1) {
34            if (texturas.find(bloqueTerreno) ≡ texturas.end()) {
35              texturas.insert(std::pair<int, std::shared_ptr<Animacion>>(
36                  bloqueTerreno,
37                  std::make_shared<Animacion>(AnimacionFactory::instanciar(
38                      bloqueTerreno,
39                      renderizador))));
40            }
41            agregarBloque(1, i, j, texturas.at(bloqueTerreno));
42          }
43        }
44      }
45    }
46  }
47
48  Pista::Pista(std::string
49               fileName, Renderizador &renderizador)
50      : renderizador(renderizador), objetosDinamicos() {
51    std::string rutaPista = fileName;
52    std::ifstream archivoPista(rutaPista);
53    Json pistaJson;
54    archivoPista >> pistaJson;
55    this→capas = pistaJson["dimensiones"]["capas"].get<uint16_t>();
56    this→size_x = pistaJson["dimensiones"]["x"].get<uint16_t>();
57    this→size_y = pistaJson["dimensiones"]["y"].get<uint16_t>();
58    for (int i = 0; i < this→capas; i++) {
59      std::vector<std::vector<std::shared_ptr<Animacion>>> matrix;
60      for (int j = 0; j < this→size_x; j++) {
61        std::vector<std::shared_ptr<Animacion>> array;
62        for (int k = 0; k < this→size_y; k++) {
63          array.emplace_back(nullptr);
64        }
65        matrix.push_back(array);
66      }
```

```
67      mapa.insert(std::pair<int,
68                            std::vector<std::vector<std::shared_ptr<Animacion>>>>(
69          i,
70          matrix));
71    }
72    crearPista(pistaJson);
73    idEventoTemporal = 0;
74  }
75
76  std::shared_ptr<Animacion> Pista::getBloque(int capa, int x, int y) const {
77    return mapa.at(capa).at(x).at(y);
78  }
79
80  void Pista::agregarObjeto(int id,
81                            std::shared_ptr<ObjetoDinamico> objetoDinamico) {
82    objetosDinamicos.insert(std::pair<int, std::shared_ptr<ObjetoDinamico>>(id,
83                                                                            objeto
    Dinamico));
84  }
85
86  void Pista::agregarEventoTemporal(std::shared_ptr<ObjetoDinamico> eventoTemporal
    ) {
87    eventosTemporales.insert(std::pair<int, std::shared_ptr<ObjetoDinamico>>(
88        idEventoTemporal,
89        eventoTemporal));
90    this→idEventoTemporal++;
91  }
92
93  std::shared_ptr<ObjetoDinamico> Pista::obtenerObjeto(int id) {
94    if (objetosDinamicos.find(id) ≠ objetosDinamicos.end()) {
95      return objetosDinamicos.at(id);
96    }
97    return nullptr;
98  }
99
100 std::shared_ptr<ObjetoDinamico> Pista::obtenerEventoTemporal(int id) {
101   return eventosTemporales.at(id);
102 }
103
104 void Pista::obtenerIds(std::vector<int> &ids) {
105   for (std::map<int, std::shared_ptr<ObjetoDinamico>>::iterator
106            it = objetosDinamicos.begin();
107        it ≠ objetosDinamicos.end(); ++it) {
108     ids.push_back(it→first);
109   }
110 }
111
112 void Pista::obtenerIdsEventosTemporales(std::vector<int> &ids) {
113   for (std::map<int, std::shared_ptr<ObjetoDinamico>>::iterator
114            it = eventosTemporales.begin();
115        it ≠ eventosTemporales.end(); ++it) {
116     ids.push_back(it→first);
117   }
118 }
119
120 void Pista::borrarObjeto(int id) {
121   if (objetosDinamicos.find(id) ≠ objetosDinamicos.end()) {
122     objetosDinamicos.erase(id);
123   }
124 }
125
126 void Pista::borrarEventoTemporal(int id) {
127   eventosTemporales.erase(id);
128 }
129
130 int Pista::getCapas() const {
```



```
131   return capas;
132 }
133
134 int Pista::getSizeX() const {
135   return size_x;
136 }
137
138 int Pista::getSizeY() const {
139   return size_y;
140 }
```

nov 26, 19 17:34 **ObjetoDinamico.cpp** Page 1/1

```cpp
#include "includes/cliente/GUI/ObjetoDinamico.h"

ObjetoDinamico::ObjetoDinamico(int uuid,
                               Renderizador &renderizador,
                               std::string sonido,
                               bool loopSonido) :
    animacion_(AnimacionFactory::instanciar(uuid, renderizador)), sonido(sonido, loopSonido) {
  this→x = 0;
  this→y = 0;
  this→angulo = 0;
  this→vida = 100;
  this→sonido.setVolume(0);
  this→sonido.play();
}

Animacion &ObjetoDinamico::getAnimacion() {
  return this→animacion_;
}

void ObjetoDinamico::mover(uint16_t x, uint16_t y, uint16_t angulo) {
  this→x = x;
  this→y = y;
  this→angulo = angulo;
}

uint16_t ObjetoDinamico::getX() const {
  return this→x;
}

uint16_t ObjetoDinamico::getY() const {
  return this→y;
}

uint16_t ObjetoDinamico::getAngulo() const {
  return this→angulo;
}

void ObjetoDinamico::setVida(uint16_t vida) {
  this→vida = vida;
}

uint16_t ObjetoDinamico::getVida() const {
  return this→vida;
}

Sonido &ObjetoDinamico::getSonido() {
  return this→sonido;
}
```

nov 26, 19 17:34 **HiloDibujador.cpp** Page 1/2

```cpp
#include "includes/cliente/GUI/HiloDibujador.h"

#include "includes/cliente/GUI/escenas/EscenaMenu.h"

#include "includes/cliente/utils/ConfigCliente.h"

#include <SDL2/SDL.h>
#include <includes/common/Cronometro.h>
#include <iostream>

void HiloDibujador::step(uint32_t iteracion, Escena &escena) {
  bool obtenido = false;
  std::shared_ptr<Evento> evento;
  while ((obtenido = eventos_.get(evento))) {
    // ACA SE PROCESAN LOS EVENTOS
    evento.get()→actualizar((Handler &) escena);
  }
  if (grabador_.estaCorriendo()) {
    renderizador_.dibujar(iteracion, escena, grabador_.getBuffer());
  } else {
    renderizador_.dibujar(iteracion, escena);
  }
}

HiloDibujador::HiloDibujador(Ventana &ventana,
                             Renderizador &renderizador,
                             HiloGrabador &grabador,
                             ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
                             ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
                             bool &seguirCorriendo)
    :
    ventana_(ventana),
    renderizador_(renderizador),
    grabador_(grabador),
    eventosGUI_(eventosGUI),
    eventosAEnviar_(eventosAEnviar_),
    musicaAmbiente(CONFIG_CLIENTE.musicaAmbiente(), true),
    seguirCorriendoCliente(seguirCorriendo) {
  escenas_.emplace(std::make_shared<EscenaMenu>(renderizador_,
                                                eventosGUI_,
                                                escenas_,
                                                eventosAEnviar_,
                                                musicaAmbiente,
                                                seguirCorriendoCliente));
}

void HiloDibujador::correr() {
  double frecuencia = (double) 1 / (double) CONFIG_CLIENTE.fps();
  frecuencia *= 1000;
  Cronometro c;
  double t1 = c.ahora();
  //TODO: Resetear cada vez que se cambia de escena
  uint32_t iteracion = 0;
  while (seguirCorriendo_) {
    Escena &escenaActual = *escenas_.top().get();
    step(iteracion, escenaActual);
    //FIXME: Se arregla haciendo que el metodo manejar evento devuelva true si hay que continuar.
    // TambiÃ©n se puede chequear al hacer pop que no se estÃ© quedando sin escenas
    bool obtenido = false;
    std::shared_ptr<EventoGUI> evento;
    while ((obtenido = eventosGUI_.get(evento))) {
      escenaActual.manejarInput(*evento.get());
```

```
63      }
64      double t2 = c.ahora();
65      double resto = frecuencia - (t2 - t1);
66      if (resto < 0) {
67        double atraso = -resto;
68        double perdidos = atraso - std::fmod(atraso, frecuencia);
69        resto = frecuencia - std::fmod(atraso, frecuencia);
70        t1 += perdidos;
71        iteracion += std::floor(perdidos / frecuencia);
72      }
73      dormir(resto);
74      t1 += frecuencia;
75      iteracion += 1;
76
77    }
78  }
79
80  void HiloDibujador::detener() {
81    seguirCorriendo_ = false;
82  }
83
84  ColaProtegida<std::shared_ptr<Evento>> &HiloDibujador::eventosEntrantes() {
85    return eventos_;
86  }
```



```
1   #include "includes/cliente/GUI/eventos/EventoGUIKeyUp.h"
2
3   #include "includes/cliente/GUI/EventoGUIHandler.h"
4
5   EventoGUIKeyUp::EventoGUIKeyUp(const std::string &tecla)
6       : tecla_(std::move(tecla)) {
7   }
8
9   void EventoGUIKeyUp::actualizar(EventoGUIHandler &handler) {
10    handler.manejarInput(*this);
11  }
12
13  std::string &EventoGUIKeyUp::getTecla() {
14    return tecla_;
15  }
16
```

nov 26, 19 17:34 **EventoGUIKeyDown.cpp** Page 1/1

```cpp
#include "includes/cliente/GUI/eventos/EventoGUIKeyDown.h"

#include "includes/cliente/GUI/EventoGUIHandler.h"

EventoGUIKeyDown::EventoGUIKeyDown(const std::string& tecla)
    : tecla_(std::move(tecla)) {
}

void EventoGUIKeyDown::actualizar(EventoGUIHandler& handler) {
  handler.manejarInput(*this);
}

std::string& EventoGUIKeyDown::getTecla() {
    return tecla_;
}
```

nov 26, 19 17:34 **EventoGUIClick.cpp** Page 1/1

```cpp
#include "includes/cliente/GUI/eventos/EventoGUIClick.h"

#include "includes/cliente/GUI/EventoGUIHandler.h"

EventoGUIClick::EventoGUIClick(unsigned int x, unsigned int y)
    : x_(x)
    , y_(y) {
}

void EventoGUIClick::actualizar(EventoGUIHandler& handler) {
  handler.manejarInput(*this);
}
```

```
1   #include "includes/cliente/GUI/escenas/EscenaSala.h"
2
3   #include "includes/cliente/GUI/escenas/EscenaLobby.h"
4   #include "includes/cliente/GUI/AnimacionFactory.h"
5   #include "includes/cliente/utils/ConfigCliente.h"
6   #include "includes/cliente/GUI/Area.h"
7   #include "includes/cliente/GUI/Texto.h"
8
9   void EscenaSala::inicializarBotones() {
10    botones.clear();
11    int anchoVentana = CONFIG_CLIENTE.anchoVentana();
12    int altoVentana = CONFIG_CLIENTE.altoVentana();
13    botones.clear();
14    this→botones.emplace(UUID_BOTON_CREAR_PARTIDA,
15      std::make_shared<Boton>(UUID_BOTON_CREAR_PARTIDA,
16                              renderizador_,
17                              0.10 * anchoVentana,
18                              0.60 * altoVentana));
19
20    this→botones.emplace(UUID_BOTON_UNIRSE_A_PARTIDA,
21      std::make_shared<Boton>(UUID_BOTON_UNIRSE_A_PARTIDA,
22                              renderizador_,
23                              0.10 * anchoVentana,
24                              0.65 * altoVentana));
25
26    this→botones.emplace(UUID_BOTON_ATRAS,
27      std::make_shared<Boton>(UUID_BOTON_ATRAS,
28                              renderizador_,
29                              0.10 * anchoVentana,
30                              0.70 * altoVentana));
31
32    // BOTONES VACIOS PARA LAS PARTIDAS
33    float posicionRelativaX = 0.45f;
34    float posicionRelativaY = 0.42f;
35
36    int restanDibujar = partidasId.size() > 4 ? 4 : partidasId.size();
37
38    for (size_t i = 0; i < partidasId.size(); ++i) {
39      if (restanDibujar ≡ 0) {
40        break;
41      }
42      this→botones.emplace(i, std::make_shared<Boton>(UUID_BOTON_VACIO,
43        renderizador_,
44        posicionRelativaX * anchoVentana,
45        posicionRelativaY * altoVentana
46        ));
47      posicionRelativaY += 0.10f;
48      restanDibujar--;
49    }
50
51    this→botones.emplace(4, std::make_shared<Boton>(UUID_BOTON_UP,
52                              renderizador_,
53                              0.80 * anchoVentana,
54                              0.42 * altoVentana));
55
56    this→botones.emplace(5, std::make_shared<Boton>(UUID_BOTON_DOWN,
57                              renderizador_,
58                              0.80 * anchoVentana,
59                              0.72 * altoVentana));
60  }
61
62  void EscenaSala::inicializarTextoPartidas() {
63    textoPartidas.clear();
64    if (finVentana_ > partidasId.size()) {
65      inicioVentana_ = 0;
66      finVentana_ = 4;
```



```
67    }
68    int tamanioFuente = 30;
69    int restanDibujar = partidasId.size() > 4 ? 4 : partidasId.size();
70    uint16_t skips = inicioVentana_;
71    int dibujadas = 0;
72    for (auto i = partidasId.begin(); i ≠ partidasId.end(); i++) {
73      if (skips > 0) {
74        skips--;
75        continue;
76      }
77      if (restanDibujar ≡ 0) {
78        break;
79      }
80      std::string texto = "Partida " + std::to_string(i→second);
81      textoPartidas.emplace(dibujadas, std::make_shared<Texto>(texto,
82        tamanioFuente,
83        renderizador_,
84        UUID_TEXTO_BLANCO));
85      restanDibujar--;
86      dibujadas++;
87    }
88  }
89
90  void EscenaSala::dibujarBotones(int nroIteracion) {
91    for (const auto &boton: botones) {
92      Animacion &animacion = boton.second.get()→getAnimacion();
93      Area areaBoton = Area(boton.second.get()→getX(),
94                            boton.second.get()→getY(),
95                            animacion.ancho(),
96                            animacion.alto());
97      renderizador_.dibujar(animacion.get(nroIteracion), areaBoton);
98    }
99  }
100
101 void EscenaSala::dibujarTextoPartidas(int iteracion) {
102   for (const auto &textoPartida: textoPartidas) {
103     std::shared_ptr<Boton> botonAsociado = botones.at(textoPartida.first);
104     Animacion &animacion = botonAsociado.get()→getAnimacion();
105     Area areaTexto = Area(botonAsociado.get()→getX(),
106                           botonAsociado.get()→getY(),
107                           animacion.ancho(),
108                           animacion.alto());
109     renderizador_.dibujarTexto(*(textoPartida.second.get()), areaTexto);
110   }
111 }
112
113 void EscenaSala::handlerBotones(int uuid) {
114   switch (uuid) {
115     case UUID_BOTON_CREAR_PARTIDA: {
116       std::shared_ptr<Evento> crearPartida = std::make_shared<EventoCrearPartida
    >();
117       eventosAEnviar_.put(crearPartida);
118       break;
119     }
120     case UUID_BOTON_UNIRSE_A_PARTIDA: {
121       if (partidaSeleccionada ≠ -1) {
122         std::shared_ptr<Evento>
123             eventoUnirseAPartida =
124             std::make_shared<EventoUnirseAPartida>(partidasId.at(
125                 partidaSeleccionada));
126       eventosAEnviar_.put(eventoUnirseAPartida);
127       }
128       break;
129     }
130     case 0: {
131       if (¬partidasId.empty()) {
```

```
132           partidaSeleccionada = 0;
133           for (const auto &texto:textoPartidas){
134             texto.second.get()→setColor(UUID_TEXTO_BLANCO);
135           }
136           textoPartidas.at(0).get()→setColor(UUID_TEXTO_AMARILLO);
137           break;
138         }
139       }
140       case 1: {
141         if (¬partidasId.empty()) {
142           partidaSeleccionada = 1;
143           for (const auto &texto:textoPartidas){
144             texto.second.get()→setColor(UUID_TEXTO_BLANCO);
145           }
146           textoPartidas.at(1).get()→setColor(UUID_TEXTO_AMARILLO);
147           break;
148         }
149       }
150       case 2: {
151         if (¬partidasId.empty()) {
152           partidaSeleccionada = 2;
153           for (const auto &texto:textoPartidas){
154             texto.second.get()→setColor(UUID_TEXTO_BLANCO);
155           }
156           textoPartidas.at(2).get()→setColor(UUID_TEXTO_AMARILLO);
157           break;
158         }
159       }
160       case 3: {
161         if (¬partidasId.empty()) {
162           partidaSeleccionada = 3;
163           for (const auto &texto:textoPartidas){
164             texto.second.get()→setColor(UUID_TEXTO_BLANCO);
165           }
166           textoPartidas.at(3).get()→setColor(UUID_TEXTO_AMARILLO);
167           break;
168         }
169       }
170       case UUID_BOTON_ATRAS: {
171         escenas_.pop();
172       }
173       //TOCO BOTON UP
174       case 4:
175         if (inicioVentana_ ≤ 0) {
176           return;
177         }
178         inicioVentana_−−;
179         finVentana_−−;
180         inicializarBotones();
181         inicializarTextoPartidas();
182         break;
183       //TOCO BOTON DOWN
184       case 5:
185         if (finVentana_ ≡ 0xFFFF ∨ finVentana_ ≡ partidasId.size()) {
186           return;
187         }
188         inicioVentana_++;
189         finVentana_++;
190         inicializarBotones();
191         inicializarTextoPartidas();
192         break;
193       default:break;
194     }
195     partidaSeleccionada += inicioVentana_;
196   }
197
```



```
198 EscenaSala::EscenaSala(Renderizador &renderizador,
199                         ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
200                         std::stack<std::shared_ptr<Escena>> &escenas,
201                         ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
202                         Sonido &musicaAmbiente,
203                         EventoSnapshotSala& e) :
204     Escena(escenas, renderizador, eventosAEnviar_, musicaAmbiente),
205     fondoMenu_(AnimacionFactory::instanciar(CONFIG_CLIENTE.uuid("fondoSala"),
206                                             renderizador)),
207     eventosGUI_(eventosGUI),
208     inicioVentana_(0),
209     finVentana_(4) {
210
211   for (const auto& kv : e.ordinalAuuidPartida_) {
212     partidasId.emplace(kv.first, kv.second);
213   }
214   partidaSeleccionada = −1;
215   inicializarBotones();
216   inicializarTextoPartidas();
217 }
218
219 Textura EscenaSala::dibujate(uint32_t numeroIteracion, Area dimensiones) {
220   Textura miTextura(renderizador_, dimensiones);
221   renderizador_.setDestino(miTextura);
222   Area areaFondo = Area(0, 0, dimensiones.ancho(), dimensiones.alto());
223   renderizador_.dibujar(fondoMenu_.get(numeroIteracion), areaFondo);
224   dibujarBotones(numeroIteracion);
225   dibujarTextoPartidas(numeroIteracion);
226   return std::move(miTextura);
227 }
228
229 void EscenaSala::manejarInput(EventoGUI &evento) {
230   evento.actualizar(*this);
231 }
232
233 void EscenaSala::manejarInput(EventoGUIClick &evento) {
234   int x, y;
235   SDL_GetMouseState(&x, &y);
236   for (const auto &boton: botones) {
237     if (boton.second.get()→estaSeleccionado(x, y)) {
238       handlerBotones(boton.first);
239       break;
240     }
241   }
242 }
243
244 void EscenaSala::manejarInput(EventoGUIKeyDown &evento) {
245   if (evento.getTecla() ≡ TECLA_ESC) {
246     escenas_.pop();
247   }
248   if (evento.getTecla() ≡ TECLA_FULLSCREEN) {
249     renderizador_.toggleFullScreen();
250   }
251 }
252
253 void EscenaSala::manejarInput(EventoGUIKeyUp &evento) {}
254
255 void EscenaSala::manejar(Evento &e) {
256   e.actualizar(*this);
257 }
258
259 void EscenaSala::manejar(EventoSnapshotSala& e) {
260   partidasId.clear();
261   for (const auto& kv : e.ordinalAuuidPartida_) {
262     partidasId.emplace(kv.first, kv.second);
263   }
```

```
264   inicioVentana_ = 0;
265   finVentana_ = 4;
266   inicializarBotones();
267   inicializarTextoPartidas();
268 }
269
270 void EscenaSala::manejar(EventoPartidaCreada& e) {
271   //TODO: EL uuid partida puede servir para mostrar por texto cual es
272   std::shared_ptr<Evento> unirme = std::make_shared<EventoUnirseAPartida>(e.uuid
    Partida_);
273   eventosAEnviar_.put(unirme);
274   escenas_.emplace(std::make_shared<EscenaLobby>(renderizador_,
275                                                  eventosGUI_,
276                                                  escenas_,
277                                                  eventosAEnviar_,
278                                                  this→musicaAmbiente,
279                                                  e));
280 }
281
282 void EscenaSala::manejar(EventoSnapshotLobby& e) {
283   escenas_.emplace(std::make_shared<EscenaLobby>(renderizador_,
284                                                    eventosGUI_,
285                                                    escenas_,
286                                                    eventosAEnviar_,
287                                                    this→musicaAmbiente,
288                                                    e));
289 }
```



```
1  #include <includes/cliente/GUI/AnimacionFactory.h>
2  #include <includes/cliente/utils/ConfigCliente.h>
3  #include "includes/cliente/GUI/escenas/EscenaPodio.h"
4
5  void EscenaPodio::dibujarAutos(int nroIteracion) {
6    for (auto &kv: this→mapaAutos) {
7      kv.second.get()→getSonido().setVolume(0);
8      Animacion &animacion = kv.second.get()→getAnimacion();
9      Area areaAuto =
10         Area(areasPodio.at(kv.first).x_ * CONFIG_CLIENTE.anchoVentana(),
11              areasPodio.at(kv.first).y_ * CONFIG_CLIENTE.altoVentana(),
12              animacion.ancho(),
13              animacion.alto());
14     renderizador_.dibujar(animacion.get(nroIteracion),
15                           areaAuto,
16                           0,
17                           false);
18   }
19 }
20
21 EscenaPodio::EscenaPodio(Renderizador &renderizador,
22                          ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
23                          std::stack<std::shared_ptr<Escena>> &escenas,
24                          ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar
    _,
25                          Sonido &musicaAmbiente,
26                          std::map<int,
27                                   std::shared_ptr<ObjetoDinamico>> &mapaAuto) :
28     Escena(escenas, renderizador, eventosAEnviar_, musicaAmbiente
29     ),
30     fondoMenu_(AnimacionFactory::instanciar(CONFIG_CLIENTE.uuid("fondoPodio"),
31                                             renderizador)
32     ),
33     eventosGUI_(eventosGUI), mapaAutos(mapaAuto) {
34   this→musicaAmbiente.play();
35   this→musicaAmbiente.setVolume(CONFIG_CLIENTE.volumenAmbiente());
36   areasPodio.insert(std::pair<int, area_t>(0, {0.46, 0.52}));
37   areasPodio.insert(std::pair<int, area_t>(1, {0.30, 0.56}));
38   areasPodio.insert(std::pair<int, area_t>(2, {0.64, 0.60}));
39 }
40
41 Textura EscenaPodio::dibujate(uint32_t numeroIteracion, Area dimensiones) {
42   Textura miTextura(renderizador_, dimensiones);
43   renderizador_.setDestino(miTextura);
44   Area areaFondo = Area(0, 0, dimensiones.ancho(), dimensiones.alto());
45   renderizador_.dibujar(fondoMenu_.get(numeroIteracion), areaFondo);
46   dibujarAutos(numeroIteracion);
47   return std::move(miTextura);
48 }
49
50 void EscenaPodio::manejarInput(EventoGUI &evento) {
51   evento.actualizar(*this);
52 }
53
54 void EscenaPodio::manejarInput(EventoGUIClick &evento) {
55   /*int x, y;
56   SDL_GetMouseState(&x, &y);
57   for (const auto &boton: botones) {
58     if (boton.second.get()->estaSeleccionado(x, y)) {
59       handlerBotones(boton.first);
60       break;
61     }
62   }*/
63 }
64
65 void EscenaPodio::manejarInput(EventoGUIKeyDown &evento) {
```

```
66    if (evento.getTecla() ≡ TECLA_ESC) {
67      while (escenas_.size() > 1) {
68        escenas_.pop();
69      }
70    }
71    if (evento.getTecla() ≡ TECLA_FULLSCREEN) {
72      renderizador_.toggleFullScreen();
73    }
74  }
75
76  void EscenaPodio::manejarInput(EventoGUIKeyUp &evento) {}
77
78  void EscenaPodio::manejar(Evento &e) {
79    e.actualizar(*this);
80  }
```



```
1   #include <iostream>
2   #include <includes/cliente/utils/ConfigCliente.h>
3   #include <includes/cliente/GUI/Texto.h>
4   #include "includes/cliente/GUI/escenas/EscenaPartida.h"
5   #include "includes/cliente/GUI/escenas/EscenaPodio.h"
6   #include "includes/cliente/GUI/Area.h"
7
8   // TODO: Refactorizar
9   void EscenaPartida::dibujarInterfaz(int iteracion) {
10    Texto vida(CONFIG_CLIENTE.texto("salud"),
11               CONFIG_CLIENTE.tamanioTexto("salud"),
12               renderizador_,
13               UUID_TEXTO_BLANCO);
14    Animacion salud
15        (AnimacionFactory::instanciar(UUID_ANIMACION_SALUD, this→renderizador_));
16    Area areaVida =
17        Area(CONFIG_CLIENTE.margenX("salud") * CONFIG_CLIENTE.anchoVentana(),
18             CONFIG_CLIENTE.margenY("salud") * CONFIG_CLIENTE.altoVentana(),
19             CONFIG_CLIENTE.anchoTexto("salud"),
20             salud.alto());
21    renderizador_.dibujarTexto(vida, areaVida);
22    std::shared_ptr<ObjetoDinamico> principalCar = pista.obtenerObjeto(id_car);
23    Area areaSalud = Area(CONFIG_CLIENTE.anchoTexto("salud") + 20,
24                          CONFIG_CLIENTE.margenY("salud")
25                              * CONFIG_CLIENTE.altoVentana(),
26                          round(principalCar.get()→getVida() * salud.ancho()
27                                  / 100),
28                          salud.alto());
29    renderizador_.dibujar(salud.get(iteracion), areaSalud);
30  }
31
32  void EscenaPartida::dibujarBarro(int iteracion) {
33    Area areaBarro = Area(0,
34                          0,
35                          CONFIG_CLIENTE.anchoVentana()
36                              * CONFIG_CLIENTE.factorLejaniaCamara(),
37                          CONFIG_CLIENTE.altoVentana()
38                              * CONFIG_CLIENTE.factorLejaniaCamara());
39    renderizador_.dibujar(barro.getAnimacion().get(iteracion), areaBarro);
40  }
41
42  //TODO: Cargar la pista json una sola vez. Para la computadora y para la Pista
43  EscenaPartida::EscenaPartida(Renderizador &renderizador,
44                               ColaProtegida<std::shared_ptr<EventoGUI>> &eventosG
    UI,
45                               std::stack<std::shared_ptr<Escena>> &escenas,
46                               ColaBloqueante<std::shared_ptr<Evento>> &eventosAEn
    viar_,
47                               EventoPartidaIniciada &estadoInicial,
48                               Sonido &musicaAmbiente,
49                               bool juegaComputadora) :
50      Escena(escenas, renderizador, eventosAEnviar_, musicaAmbiente),
51      eventosGUI_(eventosGUI),
52      pista("assets/pistas/1.json", renderizador),
53      conversor(CONFIG_CLIENTE.pixelPorMetro(), CONFIG_CLIENTE.pixelPorBloque()),
54      camara(conversor, pista, renderizador),
55      barro(UUID_ANIMACION_BARRO_GRANDE,
56            renderizador,
57            CONFIG_CLIENTE.musicaVacio(),
58            true) {
59    this→musicaAmbiente.stop();
60    const std::map<uint8_t, datosVehiculo_>
61        &idsADatosVehiculos = estadoInicial.estadoInicial_.idsADatosVehiculos_;
62    //FIXME: Esto no esta bueno
63    int vehiculoActual = 800;
64    for (const auto &kv : idsADatosVehiculos) {
```

```
65      uint8_t id = kv.first;
66      std::shared_ptr<ObjetoDinamico> vehiculo =
67          std::make_shared<ObjetoDinamico>(vehiculoActual,
68                                           renderizador,
69                                           CONFIG_CLIENTE.musicaMotor(),
70                                           true);
71      pista.agregarObjeto(id, vehiculo);
72      //FIXME: ESTO TAMBIEN ESTA FEO, LA PISTA DEBERIA DEJAR AGREGAR EN X; Y; ANGU
    LO; TAMBIEN CON VIDA?
73      int xCoord = conversor.metroAPixel(kv.second.xCoord_);
74      int yCoord = conversor.metroAPixel(kv.second.yCoord_);
75      uint16_t angulo = kv.second.angulo_;
76
77      pista.obtenerObjeto(id)→mover(xCoord, yCoord, angulo);
78      vehiculoActual += 10;
79
80      if (juegaComputadora) {
81        jugador_ = std::make_shared<Computadora>(eventosAEnviar_, "assets/pistas/1.json")
    ;
82      } else {
83        jugador_ =std::make_shared<Jugador>(eventosAEnviar_);
84      }
85    }
86    camara.setCar(pista.obtenerObjeto(estadoInicial.idDelVehiculo_));
87    this→id_car = estadoInicial.idDelVehiculo_;
88    jugador_→empezar();
89    this→barroActivo = false;
90  }
91
92  Textura EscenaPartida::dibujate(uint32_t numeroIteracion, Area dimensiones) {
93    float reescalado = CONFIG_CLIENTE.factorLejaniaCamara();
94    Area nueva = Area(0,
95                      0,
96                      dimensiones.ancho() * reescalado,
97                      dimensiones.alto() * reescalado);
98    Textura miTextura(renderizador_, nueva);
99    renderizador_.setDestino(miTextura);
100   this→camara.setWidthHeight(nueva.ancho(), nueva.alto());
101   camara.dibujarPista(numeroIteracion);
102   std::shared_ptr<ObjetoDinamico> principalCar = pista.obtenerObjeto(id_car);
103   Animacion &animacion = principalCar.get()→getAnimacion();
104   Area areaFondo =
105       Area(nueva.ancho() / 2 − (float) animacion.ancho() / 2.0f,
106            nueva.alto() / 2 − (float) animacion.alto() / 2.0f,
107            animacion.ancho(),
108            animacion.alto());
109   renderizador_.dibujar(animacion.get(numeroIteracion),
110                         areaFondo,
111                         principalCar.get()→getAngulo(),
112                         false);
113   camara.dibujarObjetos(id_car, numeroIteracion);
114   camara.dibujarEventosTemporales(numeroIteracion);
115   if (barroActivo)
116     dibujarBarro(numeroIteracion);
117   dibujarInterfaz(numeroIteracion);
118   return std::move(miTextura);
119 }
120
121 void EscenaPartida::manejarInput(EventoGUI &evento) {
122   evento.actualizar((EventoGUIHandler &) (*this));
123 }
124
125 void EscenaPartida::manejarInput(EventoGUIClick &evento) {}
126
127 void EscenaPartida::manejarInput(EventoGUIKeyDown &evento) {
128   if (evento.getTecla() ≡ TECLA_FULLSCREEN) {
```



```
129     renderizador_.toggleFullScreen();
130   } else if (evento.getTecla() ≡ TECLA_ESC) {
131     escenas_.pop();
132   } else if (evento.getTecla() ≡ TECLA_A) {
133     jugador_→acelerar();
134   } else if (evento.getTecla() ≡ TECLA_Z) {
135     jugador_→frenar();
136   } else if (evento.getTecla() ≡ TECLA_IZQ) {
137     jugador_→doblarIzquierda();
138   } else if (evento.getTecla() ≡ TECLA_DER) {
139     jugador_→doblarDerecha();
140   }
141 }
142
143 void EscenaPartida::manejarInput(EventoGUIKeyUp &evento) {
144   if (evento.getTecla() ≡ TECLA_A) {
145     jugador_→desacelerar();
146   } else if (evento.getTecla() ≡ TECLA_Z) {
147     jugador_→dejarDeFrenar();
148   } else if (evento.getTecla() ≡ TECLA_IZQ) {
149     jugador_→dejarDeDoblarIzquierda();
150   } else if (evento.getTecla() ≡ TECLA_DER) {
151     jugador_→dejarDeDoblarDerecha();
152   }
153 }
154
155 void EscenaPartida::manejar(Evento &e) {
156   e.actualizar((Handler &) (*this));
157 }
158
159 void EscenaPartida::manejar(EventoSnapshot &e) {
160   std::map<uint8_t, datosVehiculo_> datos = e.idsADatosVehiculos_;
161   for (const auto &kv : datos) {
162     float posX = this→conversor.metroAPixel(kv.second.xCoord_);
163     float posY = this→conversor.bloqueAPixel(pista.getSizeY())
164         − this→conversor.metroAPixel(kv.second.yCoord_);
165     this→pista.obtenerObjeto(kv.first).get()→mover(posX,
166                                                    posY,
167                                                    kv.second.angulo_);
168     this→pista.obtenerObjeto(kv.first).get()→setVida(kv.second.salud_);
169   }
170   jugador_→setEstado(datos[this→id_car].xCoord_,
171                      datos[this→id_car].yCoord_,
172                      datos[this→id_car].angulo_);
173 }
174
175 void EscenaPartida::manejar(EventoChoque &e) {
176   float posX = this→conversor.metroAPixel(e.xCoord_);
177   float posY = this→conversor.bloqueAPixel(pista.getSizeY())
178       − this→conversor.metroAPixel(e.yCoord_);
179   std::shared_ptr<ObjetoDinamico> choque =
180       std::make_shared<ObjetoDinamico>(UUID_ANIMACION_VACIA,
181                                        renderizador_,
182                                        CONFIG_CLIENTE.musicaChoque(),
183                                        false);
184   choque.get()→mover(posX, posY, 0);
185   pista.agregarEventoTemporal(choque);
186 }
187
188 void EscenaPartida::manejar(EventoFrenada &e) {
189   float posX = this→conversor.metroAPixel(e.xCoord_);
190   float posY = this→conversor.bloqueAPixel(pista.getSizeY())
191       − this→conversor.metroAPixel(e.yCoord_);
192   std::shared_ptr<ObjetoDinamico> frenada =
193       std::make_shared<ObjetoDinamico>(UUID_ANIMACION_VACIA,
194                                        renderizador_,
```

```
195                                     CONFIG_CLIENTE.musicaFrenada(),
196                                     false);
197     frenada.get()→mover(posX, posY, 0);
198     pista.agregarEventoTemporal(frenada);
199   }
200
201   void EscenaPartida::manejar(EventoExplosion &e) {
202     float posX = this→conversor.metroAPixel(e.xCoord_);
203     float posY = this→conversor.bloqueAPixel(pista.getSizeY())
204         - this→conversor.metroAPixel(e.yCoord_);
205     std::shared_ptr<ObjetoDinamico> explosion =
206         std::make_shared<ObjetoDinamico>(UUID_ANIMACION_EXPLOSION,
207                                     renderizador_,
208                                     CONFIG_CLIENTE.musicaExplosion(),
209                                     false);
210     explosion.get()→mover(posX, posY, 0);
211     pista.agregarEventoTemporal(explosion);
212   }
213
214   void EscenaPartida::manejar(EventoBarroPisado &e) {
215     this→barroActivo = true;
216   }
217
218   void EscenaPartida::manejar(EventoFinBarro &e) {
219     this→barroActivo = false;
220   }
221
222   void EscenaPartida::manejar(EventoFinCarrera &e) {
223     std::map<int, std::shared_ptr<ObjetoDinamico>> mapaAutos;
224     for (uint8_t i = 0; i < e.podio_.size(); i++) {
225       mapaAutos.insert(std::pair<int, std::shared_ptr<ObjetoDinamico>>(i,
226                                                                   pista.obten
    erObjeto(
227                                                                       e.podio
    _[i])));
228     }
229     escenas_.emplace(std::make_shared<EscenaPodio>(renderizador_,
230                                                    eventosGUI_,
231                                                    escenas_,
232                                                    eventosAEnviar_,
233                                                    this→musicaAmbiente,
234                                                    mapaAutos));
235   }
236
237   void EscenaPartida::manejar(EventoAparecioConsumible& e) {
238     int idAnimacion = 0;
239     if (e.tipoConsumible_ ≡ UUID_VIDA) {
240       idAnimacion = UUID_ANIMACION_CAJAS_SALUD;
241     } else if (e.tipoConsumible_ ≡ UUID_BOOST) {
242       idAnimacion = UUID_ANIMACION_BOOST;
243     } else if (e.tipoConsumible_ ≡ UUID_BARRO) {
244       idAnimacion = UUID_ANIMACION_BARRO;
245     } else if (e.tipoConsumible_ ≡ UUID_PIEDRA) {
246       idAnimacion = UUID_ANIMACION_PIEDRA;
247     } else if (e.tipoConsumible_ ≡ UUID_ACEITE) {
248       idAnimacion = UUID_ANIMACION_ACEITE;
249     }
250
251     std::shared_ptr<ObjetoDinamico> consumible =
252           std::make_shared<ObjetoDinamico>(idAnimacion,
253                                       this→renderizador_,
254                                       CONFIG_CLIENTE.musicaVacio(),
255                                       false);
256
257
258       float posX = this→conversor.metroAPixel(e.xCoord_);
```



```
259       float posY = this→conversor.bloqueAPixel(pista.getSizeY())
260           - this→conversor.metroAPixel(e.yCoord_);
261
262       consumible→mover(posX, posY, 0);
263
264       pista.agregarObjeto(e.uuidConsumible_, consumible);
265   }
266
267   void EscenaPartida::manejar(EventoDesaparecioConsumible& e) {
268     pista.borrarObjeto(e.uuidConsumible_);
269   }
270
271   EscenaPartida::~EscenaPartida() {
272     jugador_→terminar();
273   }
```

```
1   #include <iostream>
2   #include "includes/cliente/GUI/escenas/EscenaMenu.h"
3   #include "includes/cliente/GUI/escenas/EscenaPartida.h"
4   #include "includes/cliente/GUI/escenas/EscenaSala.h"
5   #include "includes/cliente/GUI/AnimacionFactory.h"
6   #include "includes/cliente/utils/ConfigCliente.h"
7   #include "includes/cliente/GUI/Area.h"
8
9   void EscenaMenu::inicializarBotones() {
10    this→botones.insert(std::pair<int, std::shared_ptr<Boton>>(
11        UUID_BOTON_JUGAR,
12        std::make_shared<Boton>(UUID_BOTON_JUGAR,
13                                renderizador_,
14                                0.41
15                                    * CONFIG_CLIENTE.anchoVentana(),
16                                0.60
17                                    * CONFIG_CLIENTE.altoVentana())));
18    this→botones.insert(std::pair<int, std::shared_ptr<Boton>>(
19        UUID_BOTON_SALIR,
20        std::make_shared<Boton>(UUID_BOTON_SALIR,
21                                renderizador_,
22                                0.41
23                                    * CONFIG_CLIENTE.anchoVentana(),
24                                0.70
25                                    * CONFIG_CLIENTE.altoVentana())));
26  }
27
28  void EscenaMenu::dibujarBotones(int nroIteracion) {
29    for (const auto &boton: botones) {
30      Animacion &animacion = boton.second.get()→getAnimacion();
31      Area areaBoton = Area(boton.second.get()→getX(),
32                            boton.second.get()→getY(),
33                            animacion.ancho(),
34                            animacion.alto());
35      renderizador_.dibujar(animacion.get(nroIteracion), areaBoton);
36    }
37  }
38
39  void EscenaMenu::handlerBotones(int uuid) {
40    switch (uuid) {
41      case UUID_BOTON_JUGAR: {
42        std::shared_ptr<Evento> unirme = std::make_shared<EventoUnirseASala>();
43        eventosAEnviar_.put(unirme);
44        quiereEntrarASala = true;
45        break;
46      }
47      case UUID_BOTON_SALIR: {
48        seguirCorriendoCliente = false;
49        break;
50      }
51      default:break;
52    }
53  }
54
55  EscenaMenu::EscenaMenu(Renderizador &renderizador,
56                         ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
57                         std::stack<std::shared_ptr<Escena>> &escenas,
58                         ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
59                         Sonido &musicaAmbiente, bool &seguirCorriendo) :
60      Escena(escenas, renderizador, eventosAEnviar_, musicaAmbiente),
61      fondoMenu_(AnimacionFactory::instanciar(CONFIG_CLIENTE.uuid("fondoMenu"),
62                                              renderizador)),
63      eventosGUI_(eventosGUI),
64      quiereEntrarASala(false),
65      seguirCorriendoCliente(seguirCorriendo) {
66    inicializarBotones();
```



```
67    this→musicaAmbiente.setVolume(CONFIG_CLIENTE.volumenAmbiente());
68    this→musicaAmbiente.play();
69  }
70
71  Textura EscenaMenu::dibujate(uint32_t numeroIteracion, Area dimensiones) {
72    Textura miTextura(renderizador_, dimensiones);
73    renderizador_.setDestino(miTextura);
74    Area areaFondo = Area(0, 0, dimensiones.ancho(), dimensiones.alto());
75    renderizador_.dibujar(fondoMenu_.get(numeroIteracion), areaFondo);
76    dibujarBotones(numeroIteracion);
77    return std::move(miTextura);
78  }
79
80  void EscenaMenu::manejarInput(EventoGUI &evento) {
81    evento.actualizar(*this);
82  }
83
84  void EscenaMenu::manejarInput(EventoGUIClick &evento) {
85    int x, y;
86    SDL_GetMouseState(&x, &y);
87    for (const auto &boton: botones) {
88      if (boton.second.get()→estaSeleccionado(x, y)) {
89        handlerBotones(boton.first);
90        break;
91      }
92    }
93  }
94
95  void EscenaMenu::manejarInput(EventoGUIKeyDown &evento) {
96    if (evento.getTecla() ≡ TECLA_FULLSCREEN) {
97      renderizador_.toggleFullScreen();
98    }
99  }
100
101 void EscenaMenu::manejarInput(EventoGUIKeyUp &evento) {}
102
103 void EscenaMenu::manejar(Evento &e) {
104   e.actualizar(*this);
105 }
106
107 void EscenaMenu::manejar(EventoSnapshotSala &e) {
108   if (¬quiereEntrarASala) {
109     return;
110   }
111   escenas_.emplace(std::make_shared<EscenaSala>(renderizador_,
112                                                 eventosGUI_,
113                                                 escenas_,
114                                                 eventosAEnviar_,
115                                                 this→musicaAmbiente,
116                                                 e));
117 }
```

```
1   #include "includes/cliente/GUI/escenas/EscenaLobby.h"
2
3   #include "includes/cliente/GUI/escenas/EscenaPartida.h"
4   #include "includes/cliente/GUI/AnimacionFactory.h"
5   #include "includes/cliente/utils/ConfigCliente.h"
6   #include "includes/cliente/GUI/Area.h"
7
8   void EscenaLobby::inicializarBotones() {
9     int anchoVentana = CONFIG_CLIENTE.anchoVentana();
10    int altoVentana = CONFIG_CLIENTE.altoVentana();
11
12    this→botones.emplace(UUID_BOTON_INICIAR_PARTIDA,
13                         std::make_shared<Boton>(UUID_BOTON_INICIAR_PARTIDA,
14                                                 renderizador_,
15                                                 0.10 * anchoVentana,
16                                                 0.60 * altoVentana));
17
18    this→botones.emplace(UUID_BOTON_ATRAS,
19                         std::make_shared<Boton>(UUID_BOTON_ATRAS,
20                                                 renderizador_,
21                                                 0.10 * anchoVentana,
22                                                 0.70 * altoVentana));
23    this→botones.emplace(UUID_BOTON_CIRCULAR,
24                         std::make_shared<Boton>(UUID_BOTON_CIRCULAR,
25                                                 renderizador_,
26                                                 0.40 * anchoVentana,
27                                                 0.80 * altoVentana));
28  }
29
30  void EscenaLobby::inicializarTextoJugadores() {
31    int tamanioFuente = 30;
32    textoJugadores.clear();
33    for (size_t i = 0; i < jugadoresId.size(); ++i) {
34      std::string texto = "Jugador " + std::to_string(jugadoresId.at(i));
35      int color = UUID_TEXTO_AMARILLO;
36      if (jugadoresEstaListo.at(i)) {
37        std::cout << "Es true\n";
38        color = UUID_TEXTO_VERDE;
39      }
40      textoJugadores.emplace(i, std::make_shared<Texto>(texto,
41                                                        tamanioFuente,
42                                                        renderizador_,
43                                                        color));
44    }
45    int color = UUID_TEXTO_BLANCO;
46    if (cpu) {
47      color = UUID_TEXTO_VERDE;
48    }
49    textoJugadores.emplace(99, std::make_shared<Texto>("CPU",
50                                                       tamanioFuente,
51                                                       renderizador_,
52                                                       color));
53  }
54
55  void EscenaLobby::dibujarBotones(int nroIteracion) {
56    for (const auto &boton: botones) {
57      Animacion &animacion = boton.second.get()→getAnimacion();
58      Area areaBoton = Area(boton.second.get()→getX(),
59                            boton.second.get()→getY(),
60                            animacion.ancho(),
61                            animacion.alto());
62      renderizador_.dibujar(animacion.get(nroIteracion), areaBoton);
63    }
64  }
65
66  void EscenaLobby::handlerBotones(int uuid) {
```



```
67    switch (uuid) {
68      case UUID_BOTON_INICIAR_PARTIDA: {
69        std::shared_ptr<Evento> jugar = std::make_shared<EventoIniciarPartida>();
70        eventosAEnviar_.put(jugar);
71        break;
72      }
73      case UUID_BOTON_ATRAS: {
74        escenas_.pop();
75      }
76      case UUID_BOTON_CIRCULAR: {
77        cpu = ¬cpu;
78        inicializarTextoJugadores();
79      }
80      default:break;
81    }
82  }
83
84  void EscenaLobby::dibujarTextoJugadores(int iteracion) {
85    double i = 0;
86    int anchoVentana = CONFIG_CLIENTE.anchoVentana();
87    int altoVentana = CONFIG_CLIENTE.altoVentana();
88    for (const auto &textoJugador: textoJugadores) {
89      if (textoJugador.first ≠ 99) {
90        Area areaTexto = Area(0.45 * anchoVentana,
91                              (0.42 + i) * altoVentana,
92                              247,
93                              31);
94        renderizador_.dibujarTexto(*(textoJugador.second.get()), areaTexto);
95        i = i + 0.10;
96      } else {
97        Area areaTextoCPU = Area(0.41 * anchoVentana,
98                                 0.80 * altoVentana,
99                                 50,
100                                50);
101       renderizador_.dibujarTexto(*(textoJugadores.at(99)), areaTextoCPU);
102     }
103   }
104
105 }
106
107 EscenaLobby::EscenaLobby(Renderizador &renderizador,
108                          ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
109                          std::stack<std::shared_ptr<Escena>> &escenas,
110                          ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar
    _,
111                          Sonido &musicaAmbiente,
112                          EventoPartidaCreada &e) :
113     Escena(escenas, renderizador, eventosAEnviar_, musicaAmbiente),
114     fondoMenu_(AnimacionFactory::instanciar(CONFIG_CLIENTE.uuid("fondoSala"),
115                                             renderizador)),
116     eventosGUI_(eventosGUI),
117     cpu(false) {
118
119   jugadoresId.emplace(0, e.uuidCreador_);
120   jugadoresEstaListo.emplace(0, false);
121   inicializarBotones();
122   inicializarTextoJugadores();
123 }
124
125 EscenaLobby::EscenaLobby(Renderizador &renderizador,
126                          ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
127                          std::stack<std::shared_ptr<Escena>> &escenas,
128                          ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar
    _,
129                          Sonido &musicaAmbiente,
130                          EventoSnapshotLobby &e) :
```

```
131     Escena(escenas, renderizador, eventosAEnviar_, musicaAmbiente),
132     fondoMenu_(AnimacionFactory::instanciar(CONFIG_CLIENTE.uuid("fondoSala"),
133                                             renderizador)),
134     eventosGUI_(eventosGUI) {
135
136   int ordinal = 0;
137   for (const auto &kv : e.idJugadorAEstaListo_) {
138     jugadoresId.emplace(ordinal, kv.first);
139     jugadoresEstaListo.emplace(ordinal, kv.second);
140     ordinal++;
141   }
142   inicializarBotones();
143   inicializarTextoJugadores();
144 }
145
146 Textura EscenaLobby::dibujate(uint32_t numeroIteracion, Area dimensiones) {
147   Textura miTextura(renderizador_, dimensiones);
148   renderizador_.setDestino(miTextura);
149   Area areaFondo = Area(0, 0, dimensiones.ancho(), dimensiones.alto());
150   renderizador_.dibujar(fondoMenu_.get(numeroIteracion), areaFondo);
151   dibujarBotones(numeroIteracion);
152   dibujarTextoJugadores(numeroIteracion);
153   return std::move(miTextura);
154 }
155
156 void EscenaLobby::manejarInput(EventoGUI &evento) {
157   evento.actualizar(*this);
158 }
159
160 void EscenaLobby::manejarInput(EventoGUIClick &evento) {
161   int x, y;
162   SDL_GetMouseState(&x, &y);
163   for (const auto &boton: botones) {
164     if (boton.second.get()→estaSeleccionado(x, y)) {
165       handlerBotones(boton.first);
166       break;
167     }
168   }
169 }
170
171 void EscenaLobby::manejarInput(EventoGUIKeyDown &evento) {
172   if (evento.getTecla() ≡ TECLA_ESC) {
173     escenas_.pop();
174   }
175   if (evento.getTecla() ≡ TECLA_FULLSCREEN) {
176     renderizador_.toggleFullScreen();
177   }
178 }
179
180 void EscenaLobby::manejarInput(EventoGUIKeyUp &evento) {}
181
182 void EscenaLobby::manejar(Evento &e) {
183   e.actualizar(*this);
184 }
185
186 void EscenaLobby::manejar(EventoPartidaIniciada &estadoInicial) {
187   escenas_.emplace(std::make_shared<EscenaPartida>(renderizador_,
188                                                    eventosGUI_,
189                                                    escenas_,
190                                                    eventosAEnviar_,
191                                                    estadoInicial,
192                                                    this→musicaAmbiente,
193                                                    cpu));
194 }
195
196 void EscenaLobby::manejar(EventoSnapshotLobby &e) {
```



```
197   jugadoresId.clear();
198   jugadoresEstaListo.clear();
199   int ordinal = 0;
200   for (auto &kv : e.idJugadorAEstaListo_) {
201     jugadoresId.emplace(ordinal, kv.first);
202     jugadoresEstaListo.emplace(ordinal, kv.second);
203     ordinal++;
204   }
205   inicializarTextoJugadores();
206 }
```

```cpp
#include "includes/cliente/GUI/escenas/Escena.h"

Escena::Escena(std::stack<std::shared_ptr<Escena>> &escenas,
               Renderizador &renderizador,
               ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
               Sonido &musicaAmbiente) :
    escenas_(escenas),
    renderizador_(renderizador),
    eventosAEnviar_(eventosAEnviar_),
    musicaAmbiente(musicaAmbiente) {}

Escena::~Escena() {

}
```



```cpp
#include "includes/cliente/GUI/Camara.h"

Camara::Camara(Conversor &conversor, Pista &pista, Renderizador &renderizador) :
    conversor(conversor), pista(pista), renderizador_(renderizador) {
  xInicial = 0;
  xFinal = 0;
  yInicial = 0;
  yFinal = 0;
}

void Camara::setWidthHeight(int width, int height) {
  this→width = width;
  this→height = height;
}
void Camara::setCar(std::shared_ptr<ObjetoDinamico> car) {
  this→car = car;
  car.get()→getSonido().setVolume(30);
}

void Camara::dibujarPista(int iteracion) {
  int posCarX = conversor.pixelABloque(car.get()→getX());
  int posCarY = conversor.pixelABloque(car.get()→getY());
  xInicial = posCarX − (conversor.pixelABloque(width / 2) + 1);
  xFinal = posCarX + (conversor.pixelABloque(width / 2) + 2);
  yInicial = posCarY − (conversor.pixelABloque(height / 2) + 1);
  yFinal = posCarY + (conversor.pixelABloque(height / 2) + 2);
  if (xInicial < 0) {
    xInicial = 0;
  }
  if (yInicial < 0) {
    yInicial = 0;
  }
  if (xFinal > pista.getSizeX()) {
    xFinal = pista.getSizeX();
  }
  if (yFinal > pista.getSizeY()) {
    yFinal = pista.getSizeY();
  }

  int nroCapas = pista.getCapas();
  for (int k = 0; k < nroCapas; k++) {
    for (int i = xInicial; i < xFinal; i++) {
      for (int j = yInicial; j < yFinal; j++) {
        std::shared_ptr<Animacion> animacion = pista.getBloque(k, i, j);
        if (animacion ≠ nullptr) {
          Area areaFondo = Area(
              i * animacion.get()→ancho() − (car.get()→getX() − width / 2),
              j * animacion.get()→alto() − (car.get()→getY() − height / 2),
              animacion.get()→ancho(),
              animacion.get()→alto());
          renderizador_.dibujar(animacion.get()→get(iteracion), areaFondo);
        }
      }
    }
  }
}

void Camara::dibujarObjetos(int car_id, int iteracion) {
  int cantidadObjetos = 0;
  std::vector<int> idObjetos;
  pista.obtenerIds(idObjetos);
  for (uint16_t i = 0; i < idObjetos.size(); i++) {
    if (idObjetos[i] ≠ car_id) {
      std::shared_ptr<ObjetoDinamico>
          objeto = pista.obtenerObjeto(idObjetos[i]);
      objeto.get()→getSonido().setVolume(0);
```

```
67        if (objeto ≠ nullptr) {
68          int bloqueCarX = conversor.pixelABloque(objeto.get()→getX());
69          int bloqueCarY = conversor.pixelABloque(objeto.get()→getY());
70          if (bloqueCarX ≥ xInicial ∧
71              bloqueCarX ≤ xFinal ∧
72              bloqueCarY ≥ yInicial ∧
73              bloqueCarY ≤ yFinal) {
74            if (cantidadObjetos ≤ 3) {
75              objeto.get()→getSonido().setVolume(20);
76              cantidadObjetos++;
77            }
78            Animacion &animacion = objeto.get()→getAnimacion();
79            Area areaFondo = Area(
80                objeto.get()→getX() − (this→car.get()→getX() − width / 2)
81                    − (float) objeto→getAnimacion().ancho() / 2.0f,
82                objeto.get()→getY() − (this→car.get()→getY() − height / 2)
83                    − (float) objeto→getAnimacion().alto() / 2.0f,
84                animacion.ancho(),
85                animacion.alto());
86            renderizador_.dibujar(animacion.get(iteracion),
87                                  areaFondo,
88                                  objeto.get()→getAngulo(),
89                                  false);
90          }
91        }
92      }
93    }
94  }
95
96  void Camara::dibujarEventosTemporales(int iteracion) {
97    int cantidadEventos = 0;
98    std::vector<int> idEventos;
99    pista.obtenerIdsEventosTemporales(idEventos);
100   for (uint16_t i = 0; i < idEventos.size(); i++) {
101     std::shared_ptr<ObjetoDinamico>
102         objeto = pista.obtenerEventoTemporal(idEventos[i]);
103     objeto.get()→getSonido().setVolume(0);
104     if (objeto ≠ nullptr) {
105       int bloqueCarX = conversor.pixelABloque(objeto.get()→getX());
106       int bloqueCarY = conversor.pixelABloque(objeto.get()→getY());
107       if (bloqueCarX ≥ xInicial ∧
108           bloqueCarX ≤ xFinal ∧
109           bloqueCarY ≥ yInicial ∧
110           bloqueCarY ≤ yFinal) {
111         if (cantidadEventos ≤ 4) {
112           objeto.get()→getSonido().setVolume(30);
113           cantidadEventos++;
114         }
115         Animacion &animacion = objeto.get()→getAnimacion();
116         Area areaFondo = Area(
117             objeto.get()→getX() − (this→car.get()→getX() − width / 2)
118                 − (float) objeto→getAnimacion().ancho() / 2.0f,
119             objeto.get()→getY() − (this→car.get()→getY() − height / 2)
120                 − (float) objeto→getAnimacion().alto() / 2.0f,
121             animacion.ancho(),
122             animacion.alto());
123         renderizador_.dibujar(animacion.get(iteracion),
124                               areaFondo,
125                               objeto.get()→getAngulo(),
126                               false);
127       }
128     }
129     if (objeto.get()→getAnimacion().terminoPrimerIteracion()) {
130       pista.borrarEventoTemporal(idEventos[i]);
131     }
132   }
```



```
133 }
134
```

nov 26, 19 17:34 **Boton.cpp** Page 1/1

```cpp
#include <includes/cliente/GUI/AnimacionFactory.h>
#include "includes/cliente/GUI/Boton.h"

Boton::Boton(int uuid, Renderizador &renderizador, uint16_t x, uint16_t y) :
    x(x), y(y), animacion_(AnimacionFactory::instanciar(uuid, renderizador)) {}

Animacion &Boton::getAnimacion() {
  return this→animacion_;
}

bool Boton::estaSeleccionado(uint16_t x, uint16_t y) {
  return x ≥ this→x ∧
      x ≤ (this→x + this→animacion_.ancho()) ∧
      y ≥ this→y ∧
      y ≤ (this→y + this→animacion_.alto());
}

uint16_t Boton::getX() const {
  return x;
}

uint16_t Boton::getY() const {
  return y;
}
```

nov 26, 19 17:34 **Area.cpp** Page 1/1

```cpp
#include "includes/cliente/GUI/Area.h"

Area::Area(unsigned int x, unsigned int y, unsigned int ancho, unsigned int alto
) :
    x_(x),
    y_(y),
    ancho_(ancho),
    alto_(alto) {

}

unsigned int Area::ancho() {
    return ancho_;
}

unsigned int Area::alto() {
    return alto_;
}

unsigned int Area::x() {
    return x_;
}

unsigned int Area::y() {
    return y_;
}
```

```
1   #include "includes/cliente/GUI/AnimacionFactory.h"
2
3   #include <vector>
4   #include <string>
5
6   #include "includes/cliente/utils/ConfigCliente.h"
7   #include "includes/cliente/GUI/Textura.h"
8
9   Animacion AnimacionFactory::instanciar(unsigned int uuidAnimacion, Renderizador
    &renderizador) {
10    std::vector<Textura> frames_;
11    unsigned int ancho_ = 0;
12    unsigned int alto_ = 0;
13    switch (uuidAnimacion) {
14      case UUID_ANIMACION_SALUD:
15        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("salud")) {
16          frames_.emplace_back(Textura(rutaArchivo, renderizador));
17        }
18        ancho_ = CONFIG_CLIENTE.ancho("salud");
19        alto_ = CONFIG_CLIENTE.alto("salud");
20        break;
21      case UUID_BOTON_CREAR_PARTIDA:
22        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("crearPartida")) {
23          frames_.emplace_back(Textura(rutaArchivo, renderizador));
24        }
25        ancho_ = CONFIG_CLIENTE.ancho("crearPartida");
26        alto_ = CONFIG_CLIENTE.alto("crearPartida");
27        break;
28      case UUID_BOTON_UNIRSE_A_PARTIDA:
29        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("unirseAPartida")) {
30          frames_.emplace_back(Textura(rutaArchivo, renderizador));
31        }
32        ancho_ = CONFIG_CLIENTE.ancho("unirseAPartida");
33        alto_ = CONFIG_CLIENTE.alto("unirseAPartida");
34        break;
35      case UUID_BOTON_SALIR:
36        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("salir")) {
37          frames_.emplace_back(Textura(rutaArchivo, renderizador));
38        }
39        ancho_ = CONFIG_CLIENTE.ancho("salir");
40        alto_ = CONFIG_CLIENTE.alto("salir");
41        break;
42      case UUID_BOTON_JUGAR:
43        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("jugar")) {
44          frames_.emplace_back(Textura(rutaArchivo, renderizador));
45        }
46        ancho_ = CONFIG_CLIENTE.ancho("jugar");
47        alto_ = CONFIG_CLIENTE.alto("jugar");
48        break;
49      case UUID_BOTON_ATRAS:
50        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("atras")) {
51          frames_.emplace_back(Textura(rutaArchivo, renderizador));
52        }
53        ancho_ = CONFIG_CLIENTE.ancho("atras");
54        alto_ = CONFIG_CLIENTE.alto("atras");
55        break;
56      case UUID_BOTON_INICIAR_PARTIDA:
57        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("iniciarPartida")) {
58          frames_.emplace_back(Textura(rutaArchivo, renderizador));
59        }
60        ancho_ = CONFIG_CLIENTE.ancho("iniciarPartida");
61        alto_ = CONFIG_CLIENTE.alto("iniciarPartida");
62        break;
63      case UUID_BOTON_LISTO:
64        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("listo")) {
65          frames_.emplace_back(Textura(rutaArchivo, renderizador));
```



```
66        }
67        ancho_ = CONFIG_CLIENTE.ancho("listo");
68        alto_ = CONFIG_CLIENTE.alto("listo");
69        break;
70      case UUID_BOTON_MENU:
71        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("menu")) {
72          frames_.emplace_back(Textura(rutaArchivo, renderizador));
73        }
74        ancho_ = CONFIG_CLIENTE.ancho("menu");
75        alto_ = CONFIG_CLIENTE.alto("menu");
76        break;
77      case UUID_BOTON_VACIO:
78        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("vacio")) {
79          frames_.emplace_back(Textura(rutaArchivo, renderizador));
80        }
81        ancho_ = CONFIG_CLIENTE.ancho("vacio");
82        alto_ = CONFIG_CLIENTE.alto("vacio");
83        break;
84      case UUID_BOTON_UP:
85        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("arriba")) {
86          frames_.emplace_back(Textura(rutaArchivo, renderizador));
87        }
88        ancho_ = CONFIG_CLIENTE.ancho("arriba");
89        alto_ = CONFIG_CLIENTE.alto("arriba");
90        break;
91      case UUID_BOTON_DOWN:
92        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("abajo")) {
93          frames_.emplace_back(Textura(rutaArchivo, renderizador));
94        }
95        ancho_ = CONFIG_CLIENTE.ancho("abajo");
96        alto_ = CONFIG_CLIENTE.alto("abajo");
97        break;
98      case UUID_BOTON_CIRCULAR:
99        for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("circular")) {
100         frames_.emplace_back(Textura(rutaArchivo, renderizador));
101       }
102       ancho_ = CONFIG_CLIENTE.ancho("circular");
103       alto_ = CONFIG_CLIENTE.alto("circular");
104       break;
105     case UUID_ANIMACION_AUTO_ROJO:
106       for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("autoRojo")) {
107         frames_.emplace_back(Textura(rutaArchivo, renderizador));
108       }
109       ancho_ = CONFIG_CLIENTE.ancho("autoRojo");
110       alto_ = CONFIG_CLIENTE.alto("autoRojo");
111       break;
112     case UUID_ANIMACION_AUTO_AMARILLO:
113       for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("autoAmarillo")) {
114         frames_.emplace_back(Textura(rutaArchivo, renderizador));
115       }
116       ancho_ = CONFIG_CLIENTE.ancho("autoAmarillo");
117       alto_ = CONFIG_CLIENTE.alto("autoAmarillo");
118       break;
119     case UUID_ANIMACION_AUTO_NEGRO:
120       for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("autoNegro")) {
121         frames_.emplace_back(Textura(rutaArchivo, renderizador));
122       }
123       ancho_ = CONFIG_CLIENTE.ancho("autoNegro");
124       alto_ = CONFIG_CLIENTE.alto("autoNegro");
125       break;
126     case UUID_ANIMACION_AUTO_AZUL:
127       for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("autoAzul")) {
128         frames_.emplace_back(Textura(rutaArchivo, renderizador));
129       }
130       ancho_ = CONFIG_CLIENTE.ancho("autoAzul");
131       alto_ = CONFIG_CLIENTE.alto("autoAzul");
```

```
132         break;
133       case UUID_ANIMACION_AUTO_VERDE:
134         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("autoVerde")) {
135           frames_.emplace_back(Textura(rutaArchivo, renderizador));
136         }
137         ancho_ = CONFIG_CLIENTE.ancho("autoVerde");
138         alto_ = CONFIG_CLIENTE.alto("autoVerde");
139         break;
140
141       case UUID_ANIMACION_FONDO_MENU:
142         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("fondoMenu")) {
143           frames_.emplace_back(Textura(rutaArchivo, renderizador));
144         }
145         ancho_ = CONFIG_CLIENTE.ancho("fondoMenu");
146         alto_ = CONFIG_CLIENTE.alto("fondoMenu");
147         break;
148       case UUID_ANIMACION_FONDO_SALA:
149         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("fondoSala")) {
150           frames_.emplace_back(Textura(rutaArchivo, renderizador));
151         }
152         ancho_ = CONFIG_CLIENTE.ancho("fondoSala");
153         alto_ = CONFIG_CLIENTE.alto("fondoSala");
154         break;
155       case UUID_ANIMACION_FONDO_PODIO:
156         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("fondoPodio")) {
157           frames_.emplace_back(Textura(rutaArchivo, renderizador));
158         }
159         ancho_ = CONFIG_CLIENTE.ancho("fondoPodio");
160         alto_ = CONFIG_CLIENTE.alto("fondoPodio");
161         break;
162       case UUID_ANIMACION_EXPLOSION:
163         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("explosion")) {
164           frames_.emplace_back(Textura(rutaArchivo, renderizador));
165         }
166         ancho_ = CONFIG_CLIENTE.ancho("explosion");
167         alto_ = CONFIG_CLIENTE.alto("explosion");
168         break;
169
170       case UUID_ANIMACION_CAJAS_SALUD:
171         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("cajaSalud")) {
172           frames_.emplace_back(Textura(rutaArchivo, renderizador));
173         }
174         ancho_ = CONFIG_CLIENTE.ancho("cajaSalud");
175         alto_ = CONFIG_CLIENTE.alto("cajaSalud");
176         break;
177
178       case UUID_ANIMACION_BOOST:
179         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("boost")) {
180           frames_.emplace_back(Textura(rutaArchivo, renderizador));
181         }
182         ancho_ = CONFIG_CLIENTE.ancho("boost");
183         alto_ = CONFIG_CLIENTE.alto("boost");
184         break;
185
186       case UUID_ANIMACION_PIEDRA:
187         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("piedra")) {
188           frames_.emplace_back(Textura(rutaArchivo, renderizador));
189         }
190         ancho_ = CONFIG_CLIENTE.ancho("piedra");
191         alto_ = CONFIG_CLIENTE.alto("piedra");
192         break;
193
194       case UUID_ANIMACION_ACEITE:
195         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("aceite")) {
196           frames_.emplace_back(Textura(rutaArchivo, renderizador));
197         }
```



```
198         ancho_ = CONFIG_CLIENTE.ancho("aceite");
199         alto_ = CONFIG_CLIENTE.alto("aceite");
200         break;
201
202       case UUID_ANIMACION_BARRO:
203         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("barro")) {
204           frames_.emplace_back(Textura(rutaArchivo, renderizador));
205         }
206         ancho_ = CONFIG_CLIENTE.ancho("barro");
207         alto_ = CONFIG_CLIENTE.alto("barro");
208         break;
209       case UUID_ANIMACION_BARRO_GRANDE:
210         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("barroGrande")) {
211           frames_.emplace_back(Textura(rutaArchivo, renderizador));
212         }
213         ancho_ = CONFIG_CLIENTE.ancho("barroGrande");
214         alto_ = CONFIG_CLIENTE.alto("barroGrande");
215         break;
216       case UUID_ANIMACION_VACIA:
217         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("animacionVacia")) {
218           frames_.emplace_back(Textura(rutaArchivo, renderizador));
219         }
220         ancho_ = CONFIG_CLIENTE.ancho("animacionVacia");
221         alto_ = CONFIG_CLIENTE.alto("animacionVacia");
222         break;
223
224       case UUID_ANIMACION_PASTO:
225         for (std::string &rutaArchivo : CONFIG_CLIENTE.sprites("pasto")) {
226           frames_.emplace_back(Textura(rutaArchivo, renderizador));
227         }
228         ancho_ = CONFIG_CLIENTE.ancho("pasto");
229         alto_ = CONFIG_CLIENTE.alto("pasto");
230         break;
231
232       default:frames_.emplace_back(Textura("assets/pistas/" + std::to_string(uuidAnimac
    ion) + ".png", renderizador));
233         ancho_ = CONFIG_CLIENTE.anchoBloquesPista();
234         alto_ = CONFIG_CLIENTE.altoBloquesPista();
235         break;
236     }
237     return Animacion(frames_, ancho_, alto_);
238 }
```

nov 26, 19 17:34 **Animacion.cpp** Page 1/1

```cpp
#include <iostream>
#include "includes/cliente/GUI/Animacion.h"
#include "includes/cliente/utils/ConfigCliente.h"

void Animacion::loadFramesByIteration() {
  int frames_division = iterations_.size() / frames_.size();
  int frame_selected = 0;
  int frame = 0;
  for (size_t i = 0 ;i <iterations_.size();i++){
    if (frame > frames_division) {
      frame = 0;
      frame_selected++;
    }
    iterations_[i] = frame_selected;
    frame ++;
  }
}

Animacion::Animacion(std::vector<Textura> &texturas, unsigned int ancho, unsigned int alto) :
    ancho_(ancho),
    alto_(alto),
    iterations_(CONFIG_CLIENTE.fps(), 0) {
  this→primerIteracion = false;
  for (Textura &t : texturas) {
    frames_.push_back(std::move(t));
  }
  loadFramesByIteration();
}

Textura &Animacion::get(uint32_t numeroIteracion) {
  int resto = numeroIteracion % iterations_.size();
  if (numeroIteracion ≠ 0 ∧ resto ≡ 0){
    primerIteracion = true;
  }
  return frames_[iterations_[resto]];
}

bool Animacion::terminoPrimerIteracion() const {
  return primerIteracion;
}

unsigned int Animacion::ancho() {
  return ancho_;
}
unsigned int Animacion::alto() {
  return alto_;
}
```

nov 26, 19 17:34 **HiloGrabador.cpp** Page 1/1

```cpp
#include "includes/cliente/grabador/HiloGrabador.h"

#include "includes/cliente/utils/ConfigCliente.h"
#include "includes/cliente/grabador/ffmpeg/output_video.h"
#include "includes/cliente/grabador/ffmpeg/output_format.h"

#include <ctime>
#include "includes/common/Cronometro.h"

#include <thread>
#include <chrono>

void HiloGrabador::correr(){
    time_t t = std::time(0);
  long int ahora = static_cast<long int> (t);
  std::string  nombreGrabacion = std::to_string(ahora) + std::string(".") + CONFIG_CLIENTE.formatoGrabadora();
    OutputFormat fmt(nombreGrabacion);
    AVRational frame_rate = { 1, static_cast<int>(CONFIG_CLIENTE.fpsGrabadora()) };
    OutputVideo videoOutput(fmt, frame_rate, CONFIG_CLIENTE.anchoGrabadora(), CONFIG_CLIENTE.altoGrabadora(), AV_PIX_FMT_RGB24);
    fmt.open();
    double frecuencia = (double) 1 / (double) CONFIG_CLIENTE.fpsGrabadora();
    int iteracion = 0;
    frecuencia *= 1000;
    Cronometro c;
    double t1 = c.ahora();
  while (seguirCorriendo_){
        std::vector<char> linea;
        bool hayFrame = lineas_rgb_.get(linea);
        if (¬hayFrame) {
            break;
        }
        videoOutput.rgb_line_to_frame(linea.data());
        videoOutput.write_frame();
        // Tiempo por frame
        double t2 = c.ahora();
        double resto = frecuencia − (t2 − t1);
        if (resto < 0) {
          double atraso = −resto;
          double perdidos = atraso − std::fmod(atraso, frecuencia);
          resto = frecuencia − std::fmod(atraso, frecuencia);
          t1 += perdidos;
          iteracion += std::floor(perdidos / frecuencia);
        }
        Hilo::dormir(resto);
        t1 += frecuencia;
        iteracion += 1;
  }
  fmt.write_trailer();
  // Reinicio el doble buffer
    //FIXME: HACER UN VACIAR, no se pueden reasignar/mover mutexes. En realidad no es necesario vaciarlo
  //lineas_rgb_ = DobleBuffer<std::vector<char>>();
}


void HiloGrabador::detener(){
    seguirCorriendo_ = false;
    lineas_rgb_.detener();
}

DobleBuffer<std::vector<char>>& HiloGrabador::getBuffer() {
    return lineas_rgb_;
}
```

```
1  #include "includes/cliente/grabador/ffmpeg/video_codec.h"
2
3  #include "includes/cliente/utils/ConfigCliente.h"
4
5  #include <iostream>
6  #include <string>
7  #include <cmath>
8  #include <exception>
9  #include "includes/cliente/grabador/ffmpeg/codec.h"
10 #include "includes/cliente/grabador/ffmpeg/frame.h"
11
12 #include <iostream>
13
14 VideoCodec::VideoCodec(enum AVCodecID id, AVRational avr, Frame& fr, int width,
   int height, AVPixelFormat pix_fmt, bool header_flag) :
15   Codec(id) {
16   enc→codec_id = id;
17   enc→bit_rate = CONFIG_CLIENTE.bitrateGrabadora();
18
19   /* Resolution must be a multiple of two. */
20   enc→width    = width;
21   enc→height   = height;
22
23     /* timebase: This is the fundamental unit of time (in seconds) in terms
24      * of which frame timestamps are represented. For fixed-fps content,
25      * timebase should be 1/framerate and timestamp increments should be
26      * identical to 1. */
27   enc→time_base = avr;
28   enc→gop_size = 10; /* emit one intra frame every testing frames at most */
29   if (enc→codec_id ≡ AV_CODEC_ID_MPEG2VIDEO) {
30       /* just for testing, we also add B-frames */
31     enc→max_b_frames = 2;
32   }
33   if (enc→codec_id ≡ AV_CODEC_ID_MPEG1VIDEO) {
34   /* Needed to avoid using macroblocks in which some coeffs overflow.
35    * This does not happen with normal video, it just happens here as
36    * the motion of the chroma plane does not match the luma plane. */
37     enc→mb_decision = 2;
38   }
39
40   if (header_flag) {
41     enc→flags |= AV_CODEC_FLAG_GLOBAL_HEADER;
42   }
43
44   if (enc→codec_id ≡ AV_CODEC_ID_H264) {
45     enc→profile = FF_PROFILE_H264_BASELINE;
46     av_opt_set(enc→priv_data, "preset", "slow", 0);
47   }
48
49   enc→pix_fmt = codec→pix_fmts[0]; /* best quality format for codec*/
50
51   Codec::open();
52
53   sws_ctx = sws_getContext(width, height, pix_fmt, width, height, enc→pix_fmt,
   SWS_BICUBIC, NULL, NULL, NULL);
54   if (¬sws_ctx) {
55     throw std::runtime_error("No se pudo iniciar el contexto de converiÃ³n de video");
56   }
57
58   fr.VideoFrame(enc→pix_fmt, width, height);
59 }
60
61
62 void VideoCodec::write_rgb_frame(Frame& dest, const char * data, int pts) {
63   dest.fill_rgb(sws_ctx, data, enc→width, pts);
64 }
```



```
65
66 VideoCodec::VideoCodec(VideoCodec∧ rhs) :
67   Codec(std::move(rhs)){
68   this→sws_ctx = rhs.sws_ctx;
69   rhs.sws_ctx = NULL;
70 }
71
72 VideoCodec::~VideoCodec() {
73   if (sws_ctx){
74     sws_freeContext(sws_ctx);
75   }
76 }
```

nov 26, 19 17:34 **output_video.cpp** Page 1/1

```cpp
#include "includes/cliente/grabador/ffmpeg/output_video.h"

#include <stdexcept>

OutputVideo::OutputVideo(OutputFormat &fmt, AVRational avr, int w, int h, AVPixelFormat pix) :
  OutputStream(fmt) {
  try {
    st→time_base = avr;
    enc = new VideoCodec(fmt.get_video_codec_id(), avr, frame, w, h, pix, fmt.is_flag_set(AVFMT_GLOBALHEADER));
    enc→copy_parameters(st);
  }
  catch (std::runtime_error &e) {
    throw std::runtime_error(e);
  }
}

void OutputVideo::rgb_line_to_frame(const char *v) {
  reinterpret_cast<VideoCodec*>(enc)→write_rgb_frame(frame, v, current_pts);
  current_pts++;
}

OutputVideo::~OutputVideo() {
  delete enc;
}
```

nov 26, 19 17:34 **output_stream.cpp** Page 1/1

```cpp
#include "includes/cliente/grabador/ffmpeg/output_stream.h"

#include <stdexcept>

#include "includes/cliente/grabador/ffmpeg/codec.h"

OutputStream::OutputStream(OutputFormat &fmt) :
  fmt(fmt){
  st = fmt.get_new_stream();
  pkt = av_packet_alloc();
}

void OutputStream::write_frame() {
  try {
    enc→encode_frame(frame);
    while (true) {
      AVRational time_base;
      if (¬enc→get_packet(pkt, &time_base)) {
        break;
      }
      av_packet_rescale_ts(pkt, time_base, st→time_base);
      pkt→stream_index = st→index;
      //write
      if (fmt.write_pkt(pkt)) {
        throw std::runtime_error("No se pudo escribir paquete");
      }
      av_packet_unref(pkt);
    }
  }
  catch (std::runtime_error &e) {
    throw std::runtime_error(e);
  }
}

OutputStream::~OutputStream() {
  av_packet_free(&pkt);
}
```

```cpp
#include "includes/cliente/grabador/ffmpeg/output_format.h"

#include <iostream>
#include <cmath>
#include <exception>

OutputFormat::OutputFormat(const std::string &file) : filename(file) {
  av_register_all();
  avformat_alloc_output_context2(&ctx, NULL, NULL, filename.c_str());
  if (¬ctx) {
    // Si no se encuentra el formato del archivo, se usa MP4
    avformat_alloc_output_context2(&ctx, NULL, "mp4", filename.c_str());
  }
  if (¬ctx) {
    throw std::runtime_error("No se pudo alocar el contexto del formato del archivo");
  }
}

AVStream* OutputFormat::get_new_stream() {
  AVStream *st = avformat_new_stream(ctx, NULL);
  st→id = ctx→nb_streams − 1;
  return st;
}

bool OutputFormat::is_flag_set(int flag) {
  return ctx→oformat→flags & AVFMT_GLOBALHEADER;
}

int OutputFormat::write_pkt(AVPacket *pkt) {
  return av_interleaved_write_frame(ctx, pkt);
}

void OutputFormat::open() {
  av_dump_format(ctx, 0, filename.c_str(), 1);
  if (¬(ctx→oformat→flags & AVFMT_NOFILE)) {
    if (avio_open(&ctx→pb, filename.c_str(), AVIO_FLAG_WRITE) < 0) {
      throw std::runtime_error("No se pudo abrir el archivo");
    }
  }
  // EN VEZ DE NULL SE PUEDEN AGREGAR OPCIONES
  if (avformat_write_header(ctx, NULL) < 0) {
    throw std::runtime_error("No se pudo escribir el header");
  }
}

enum AVCodecID OutputFormat::get_video_codec_id() {
  return ctx→oformat→video_codec;
}

enum AVCodecID OutputFormat::get_audio_codec_id() {
  return ctx→oformat→audio_codec;
}

void OutputFormat::write_trailer() {
  if (av_write_trailer(ctx) < 0) {
    throw std::runtime_error("No se pudo escribir el trailer");
  }
}

OutputFormat::~OutputFormat() {
  if (¬(ctx→oformat→flags & AVFMT_NOFILE)) {
    avio_closep(&ctx→pb);
  }
  if (ctx) {
    avformat_free_context(ctx);
  }
```



```cpp
}
```

```
#include "includes/cliente/grabador/ffmpeg/frame.h"

#include <iostream>
#include <string>
#include <cmath>
#include <exception>
#include <vector>

Frame::Frame() {
  fr = av_frame_alloc();
  if (¬fr){
    throw std::runtime_error("No se pudo inicializar frame");
  }
}

void Frame::VideoFrame(enum AVPixelFormat pix_fmt, int width, int height) {
  fr→format = pix_fmt;
  fr→width  = width;
  fr→height = height;
  if (av_frame_get_buffer(fr, 0) < 0){
      throw std::runtime_error("No se pudo obtener buffer para el audio");
  }
}

void Frame::make_writable() {
  if (av_frame_make_writable(fr) < 0){
    throw std::runtime_error("No se pudo asegurar la escritura del Frame");
  }
}

const AVFrame* Frame::get_frame() const {
  return fr;
}

void Frame::fill_rgb(SwsContext * ctx, const char * data, int width, int pts) {
  int w = width * 3;
  sws_scale(ctx, (const uint8_t * const *) &data, &w, 0, this→fr→height, this→
fr→data, this→fr→linesize);
  this→fr→pts = pts;
}

void Frame::AudioFrame(enum AVSampleFormat sample_fmt, uint64_t channel_layout,
int sample_rate, int nb_samples) {
  fr→format = sample_fmt;
  fr→channel_layout = channel_layout;
  fr→sample_rate = sample_rate;
  fr→nb_samples = nb_samples;
  if (av_frame_get_buffer(fr, 0) < 0){
      throw std::runtime_error("No se pudo obtener buffer para el video");
  }
}

Frame::Frame(Frame∧ rhs){
  rhs.fr = this→fr;
  this→fr = NULL;
}

Frame& Frame::operator=(Frame∧ rhs){
  rhs.fr = this→fr;
  this→fr = NULL;
  return *this;
}
Frame::~Frame(){
  if (fr){
    av_frame_free(&fr);
  }
```



```
}
```

nov 26, 19 17:34 **codec.cpp** Page 1/1

```cpp
#include "includes/cliente/grabador/ffmpeg/codec.h"

#include <stdexcept>

Codec::Codec(enum AVCodecID id){
  codec = avcodec_find_encoder(id);
  if (¬codec) {
    throw std::runtime_error("Encoder no encontrado.");
  }
  enc = avcodec_alloc_context3(codec);
  if (¬enc) {
    throw std::runtime_error("No se pudo alocar el contexto del encoder!");
  }
}

void Codec::open() {
  // NULL pero se puede agregar un diccionario de opciones
  if (avcodec_open2(enc, codec, NULL) < 0){
    throw std::runtime_error("No se puedo abrir el encoder");
  }
}

void Codec::encode_frame(const Frame& f) {
  if (avcodec_send_frame(enc, f.get_frame())) {
      throw std::runtime_error("Error al enviar frame");
  }
}

int Codec::get_packet(AVPacket * pkt, AVRational *time_base) {
  int ret = avcodec_receive_packet(enc, pkt);
  if (ret ≡ AVERROR(EAGAIN) ∨ ret ≡ AVERROR_EOF) {
    return 0;
  } else if (ret < 0) {
    throw std::runtime_error("Error al codificar");
  }
  time_base→num = enc→time_base.num;
  time_base→den = enc→time_base.den;
  return 1;
}

void Codec::copy_parameters(AVStream * st) {
  if (avcodec_parameters_from_context(st→codecpar, enc) < 0) {
    throw std::runtime_error("No se pudieron copiar los parametros del contexto al stream");
  }
}


Codec::Codec(Codec∧ rhs){
  this→enc = rhs.enc;
  rhs.enc = NULL;
}

Codec& Codec::operator=(Codec∧ rhs) {
  this→enc = rhs.enc;
  rhs.enc = NULL;
  return *this;
}

Codec::~Codec() {
  if (enc ≠ nullptr) {
    avcodec_free_context(&enc);
  }
}
```

nov 26, 19 17:34 **SDLException.cpp** Page 1/1

```cpp
#include "includes/cliente/excepciones/SDLException.h"

SDLException::SDLException(const char* descripcion, const char* errorSDL) :
    std::exception(),
    descripcion_(descripcion) {

    descripcion_.append("\n |−ERROR_SDL: ").append(errorSDL);
}

const char* SDLException::what() const noexcept {
    return descripcion_.c_str();
}
```

```
1  #include "includes/cliente/Cliente.h"
2
3  #include <iostream>
4  #include <includes/cliente/GUI/eventos/EventoGUIKeyUp.h>
5
6  #include "includes/cliente/GUI/eventos/EventoGUIClick.h"
7  #include "includes/cliente/GUI/eventos/EventoGUIKeyDown.h"
8
9  Cliente::Cliente(unsigned int anchoVentana,
10                  unsigned int altoVentana,
11                  bool pantallaCompleta,
12                  const std::string &tituloVentana,
13                  const std::string &host,
14                  const std::string &puerto) :
15     seguirCorriendo(false),
16     ventana_(anchoVentana, altoVentana, pantallaCompleta, tituloVentana),
17     renderizador_(ventana_),
18     dibujador_(ventana_, renderizador_, grabador_, eventosGUI_, eventosAEnviar_,
   seguirCorriendo),
19     socket_(host, puerto),
20     recibidor_(socket_, dibujador_.eventosEntrantes(), 0),
21     enviador_(socket_, eventosAEnviar_) {
22 }
23
24 Cliente::~Cliente() {
25   dibujador_.join();
26   enviador_.join();
27   recibidor_.join();
28   grabador_.join();
29 }
30
31 void Cliente::correr() {
32     try {
33       socket_.conectar();
34     }
35     catch (const std::exception &e) {
36       std::cerr << e.what() << '\n';
37     }
38     this→seguirCorriendo = true;
39     recibidor_.iniciar();
40     enviador_.iniciar();
41     dibujador_.iniciar();
42     //TODO: Mover a inputhandler, que serÃ¡ de teclas o LUA
43     SDL_Event evento;
44     while (SDL_WaitEvent(&evento) ∧ seguirCorriendo) {
45       switch (evento.type) {
46         case SDL_KEYDOWN:manejarKeyDown(evento);
47           break;
48         case SDL_KEYUP:manejarKeyUp(evento);
49           break;
50         case SDL_MOUSEBUTTONDOWN:manejarMouseDown(evento);
51           break;
52         case SDL_QUIT:seguirCorriendo = false;
53           break;
54         default:break;
55       }
56     }
57 }
58
59 void Cliente::cerrar() {
60   dibujador_.detener();
61   eventosAEnviar_.detener();
62   enviador_.detener();
63   recibidor_.detener();
64   socket_.cerrarLectoEscritura();
65   if (grabador_.estaCorriendo()){
```



```
66     grabador_.detener();
67   }
68 }
69
70 void Cliente::manejarKeyDown(SDL_Event &eventoSDL) {
71   SDL_KeyboardEvent &keyEvent = (SDL_KeyboardEvent &) eventoSDL;
72   if (eventoSDL.key.repeat ≠ 0) {
73     return;
74   }
75   std::shared_ptr<EventoGUI> evento;
76   switch (keyEvent.keysym.sym) {
77     case SDLK_c:evento = std::make_shared<EventoGUIKeyDown>(TECLA_C);
78       eventosGUI_.put(evento);
79       break;
80     case SDLK_a:evento = std::make_shared<EventoGUIKeyDown>(TECLA_A);
81       eventosGUI_.put(evento);
82       break;
83     case SDLK_z:evento = std::make_shared<EventoGUIKeyDown>(TECLA_Z);
84       eventosGUI_.put(evento);
85       break;
86     case SDLK_LEFT:evento = std::make_shared<EventoGUIKeyDown>(TECLA_IZQ);
87       eventosGUI_.put(evento);
88       break;
89     case SDLK_RIGHT:evento = std::make_shared<EventoGUIKeyDown>(TECLA_DER);
90       eventosGUI_.put(evento);
91       break;
92     case SDLK_ESCAPE:evento = std::make_shared<EventoGUIKeyDown>(TECLA_ESC);
93       eventosGUI_.put(evento);
94       break;
95     case SDLK_F11:evento = std::make_shared<EventoGUIKeyDown>(TECLA_FULLSCREEN);
96       eventosGUI_.put(evento);
97       break;
98     case SDLK_g:
99       if (grabador_.estaCorriendo()){
100          grabador_.detener();
101      } else {
102          grabador_.join();
103          grabador_.iniciar();
104      }
105      break;
106    default:
107      break;
108  }
109 }
110
111 void Cliente::manejarKeyUp(SDL_Event &eventoSDL) {
112   SDL_KeyboardEvent &keyEvent = (SDL_KeyboardEvent &) eventoSDL;
113   if (eventoSDL.key.repeat ≠ 0) {
114     return;
115   }
116   std::shared_ptr<EventoGUI> evento;
117   switch (keyEvent.keysym.sym) {
118     case SDLK_a:evento = std::make_shared<EventoGUIKeyUp>(TECLA_A);
119       eventosGUI_.put(evento);
120       break;
121     case SDLK_z:evento = std::make_shared<EventoGUIKeyUp>(TECLA_Z);
122       eventosGUI_.put(evento);
123       break;
124     case SDLK_LEFT:evento = std::make_shared<EventoGUIKeyUp>(TECLA_IZQ);
125       eventosGUI_.put(evento);
126       break;
127     case SDLK_RIGHT:evento = std::make_shared<EventoGUIKeyUp>(TECLA_DER);
128       eventosGUI_.put(evento);
129       break;
130     default:break;
131   }
```

nov 26, 19 17:34 **Cliente.cpp** Page 3/3

```
132 }
133
134 void Cliente::manejarMouseDown(SDL_Event &eventoSDL) {
135   if (eventoSDL.button.button ≠ SDL_BUTTON_LEFT) {
136     return;
137   }
138   int x, y;
139   SDL_GetMouseState(&x, &y);
140   std::shared_ptr<EventoGUI> eventoClick = std::make_shared<EventoGUIClick>(x, y
    );
141   eventosGUI_.put(eventoClick);
142 }
```

nov 26, 19 17:34 **LuaInterprete.cpp** Page 1/1

```
1  #include "includes/3rd−party/lua/LuaInterprete.hpp"
2
3
4  LuaInterpreter::LuaInterpreter(){
5      L = luaL_newstate();
6      luaL_openlibs(L);
7  }
8
9
10 void LuaInterpreter::init_script(const char * filepath){
11   int ret = luaL_dofile(L, filepath);
12   if (ret){
13     throw std::runtime_error("Error en el script: " + std::string(lua_tostring(L, ret))
     );
14   }
15 }
16
17 void LuaInterpreter::call_function(const char * fname, int params, int outparams
    ){
18   lua_call(L, params, outparams);
19 }
20
21 void LuaInterpreter::get_function_name(const char * fname){
22   lua_getglobal(L, fname);
23 }
24
25
26
27 void LuaInterpreter::operator<<(const std::string &str){
28   lua_pushlstring(L, str.c_str(), str.size());
29 }
30
31 void LuaInterpreter::operator<<(int num){
32     lua_pushnumber(L, num);
33 }
34
35 void LuaInterpreter::operator<<(float num){
36     lua_pushnumber(L, num);
37 }
38
39 void LuaInterpreter::operator<<(bool b){
40     lua_pushboolean(L, b);
41 }
42
43 LuaInterpreter::~LuaInterpreter(){
44     if(L){
45       lua_close(L);
46   }
47 }
```

```
/*
* Copyright (c) 2011 Erin Catto http://box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Rope/b2Rope.h"
#include "Box2D/Common/b2Draw.h"

b2Rope::b2Rope()
{
  m_count = 0;
  m_ps = nullptr;
  m_p0s = nullptr;
  m_vs = nullptr;
  m_ims = nullptr;
  m_Ls = nullptr;
  m_as = nullptr;
  m_gravity.SetZero();
  m_k2 = 1.0f;
  m_k3 = 0.1f;
}

b2Rope::~b2Rope()
{
  b2Free(m_ps);
  b2Free(m_p0s);
  b2Free(m_vs);
  b2Free(m_ims);
  b2Free(m_Ls);
  b2Free(m_as);
}

void b2Rope::Initialize(const b2RopeDef* def)
{
  b2Assert(def→count ≥ 3);
  m_count = def→count;
  m_ps = (b2Vec2*)b2Alloc(m_count * sizeof(b2Vec2));
  m_p0s = (b2Vec2*)b2Alloc(m_count * sizeof(b2Vec2));
  m_vs = (b2Vec2*)b2Alloc(m_count * sizeof(b2Vec2));
  m_ims = (float32*)b2Alloc(m_count * sizeof(float32));

  for (int32 i = 0; i < m_count; ++i)
  {
    m_ps[i] = def→vertices[i];
    m_p0s[i] = def→vertices[i];
    m_vs[i].SetZero();

    float32 m = def→masses[i];
    if (m > 0.0f)
    {
      m_ims[i] = 1.0f / m;
    }
    else
```



```
    {
      m_ims[i] = 0.0f;
    }
  }

  int32 count2 = m_count - 1;
  int32 count3 = m_count - 2;
  m_Ls = (float32*)b2Alloc(count2 * sizeof(float32));
  m_as = (float32*)b2Alloc(count3 * sizeof(float32));

  for (int32 i = 0; i < count2; ++i)
  {
    b2Vec2 p1 = m_ps[i];
    b2Vec2 p2 = m_ps[i+1];
    m_Ls[i] = b2Distance(p1, p2);
  }

  for (int32 i = 0; i < count3; ++i)
  {
    b2Vec2 p1 = m_ps[i];
    b2Vec2 p2 = m_ps[i + 1];
    b2Vec2 p3 = m_ps[i + 2];

    b2Vec2 d1 = p2 - p1;
    b2Vec2 d2 = p3 - p2;

    float32 a = b2Cross(d1, d2);
    float32 b = b2Dot(d1, d2);

    m_as[i] = b2Atan2(a, b);
  }

  m_gravity = def→gravity;
  m_damping = def→damping;
  m_k2 = def→k2;
  m_k3 = def→k3;
}

void b2Rope::Step(float32 h, int32 iterations)
{
  if (h ≡ 0.0)
  {
    return;
  }

  float32 d = expf(- h * m_damping);

  for (int32 i = 0; i < m_count; ++i)
  {
    m_p0s[i] = m_ps[i];
    if (m_ims[i] > 0.0f)
    {
      m_vs[i] += h * m_gravity;
    }
    m_vs[i] *= d;
    m_ps[i] += h * m_vs[i];

  }

  for (int32 i = 0; i < iterations; ++i)
  {
    SolveC2();
    SolveC3();
    SolveC2();
  }

```

```
133   float32 inv_h = 1.0f / h;
134   for (int32 i = 0; i < m_count; ++i)
135   {
136     m_vs[i] = inv_h * (m_ps[i] - m_p0s[i]);
137   }
138 }
139
140 void b2Rope::SolveC2()
141 {
142   int32 count2 = m_count - 1;
143
144   for (int32 i = 0; i < count2; ++i)
145   {
146     b2Vec2 p1 = m_ps[i];
147     b2Vec2 p2 = m_ps[i + 1];
148
149     b2Vec2 d = p2 - p1;
150     float32 L = d.Normalize();
151
152     float32 im1 = m_ims[i];
153     float32 im2 = m_ims[i + 1];
154
155     if (im1 + im2 == 0.0f)
156     {
157       continue;
158     }
159
160     float32 s1 = im1 / (im1 + im2);
161     float32 s2 = im2 / (im1 + im2);
162
163     p1 -= m_k2 * s1 * (m_Ls[i] - L) * d;
164     p2 += m_k2 * s2 * (m_Ls[i] - L) * d;
165
166     m_ps[i] = p1;
167     m_ps[i + 1] = p2;
168   }
169 }
170
171 void b2Rope::SetAngle(float32 angle)
172 {
173   int32 count3 = m_count - 2;
174   for (int32 i = 0; i < count3; ++i)
175   {
176     m_as[i] = angle;
177   }
178 }
179
180 void b2Rope::SolveC3()
181 {
182   int32 count3 = m_count - 2;
183
184   for (int32 i = 0; i < count3; ++i)
185   {
186     b2Vec2 p1 = m_ps[i];
187     b2Vec2 p2 = m_ps[i + 1];
188     b2Vec2 p3 = m_ps[i + 2];
189
190     float32 m1 = m_ims[i];
191     float32 m2 = m_ims[i + 1];
192     float32 m3 = m_ims[i + 2];
193
194     b2Vec2 d1 = p2 - p1;
195     b2Vec2 d2 = p3 - p2;
196
197     float32 L1sqr = d1.LengthSquared();
198     float32 L2sqr = d2.LengthSquared();
```



```
199
200     if (L1sqr * L2sqr == 0.0f)
201     {
202       continue;
203     }
204
205     float32 a = b2Cross(d1, d2);
206     float32 b = b2Dot(d1, d2);
207
208     float32 angle = b2Atan2(a, b);
209
210     b2Vec2 Jd1 = (-1.0f / L1sqr) * d1.Skew();
211     b2Vec2 Jd2 = (1.0f / L2sqr) * d2.Skew();
212
213     b2Vec2 J1 = -Jd1;
214     b2Vec2 J2 = Jd1 - Jd2;
215     b2Vec2 J3 = Jd2;
216
217     float32 mass = m1 * b2Dot(J1, J1) + m2 * b2Dot(J2, J2) + m3 * b2Dot(J3, J3);
218     if (mass == 0.0f)
219     {
220       continue;
221     }
222
223     mass = 1.0f / mass;
224
225     float32 C = angle - m_as[i];
226
227     while (C > b2_pi)
228     {
229       angle -= 2 * b2_pi;
230       C = angle - m_as[i];
231     }
232
233     while (C < -b2_pi)
234     {
235       angle += 2.0f * b2_pi;
236       C = angle - m_as[i];
237     }
238
239     float32 impulse = - m_k3 * mass * C;
240
241     p1 += (m1 * impulse) * J1;
242     p2 += (m2 * impulse) * J2;
243     p3 += (m3 * impulse) * J3;
244
245     m_ps[i] = p1;
246     m_ps[i + 1] = p2;
247     m_ps[i + 2] = p3;
248   }
249 }
250
251 void b2Rope::Draw(b2Draw* draw) const
252 {
253   b2Color c(0.4f, 0.5f, 0.7f);
254
255   for (int32 i = 0; i < m_count - 1; ++i)
256   {
257     draw->DrawSegment(m_ps[i], m_ps[i+1], c);
258   }
259 }
```

```
/*
* Copyright (c) 2006-2007 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Joints/b2WheelJoint.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2TimeStep.h"

// Linear constraint (point-to-line)
// d = pB - pA = xB + rB - xA - rA
// C = dot(ay, d)
// Cdot = dot(d, cross(wA, ay)) + dot(ay, vB + cross(wB, rB) - vA - cross(wA, rA))
//      = -dot(ay, vA) - dot(cross(d + rA, ay), wA) + dot(ay, vB) + dot(cross(rB, ay), vB)
// J = [-ay, -cross(d + rA, ay), ay, cross(rB, ay)]

// Spring linear constraint
// C = dot(ax, d)
// Cdot = = -dot(ax, vA) - dot(cross(d + rA, ax), wA) + dot(ax, vB) + dot(cross(rB, ax), vB)
// J = [-ax -cross(d+rA, ax) ax cross(rB, ax)]

// Motor rotational constraint
// Cdot = wB - wA
// J = [0 0 -1 0 0 1]

void b2WheelJointDef::Initialize(b2Body* bA, b2Body* bB, const b2Vec2& anchor, const b2Vec2& axis)
{
  bodyA = bA;
  bodyB = bB;
  localAnchorA = bodyA->GetLocalPoint(anchor);
  localAnchorB = bodyB->GetLocalPoint(anchor);
  localAxisA = bodyA->GetLocalVector(axis);
}

b2WheelJoint::b2WheelJoint(const b2WheelJointDef* def)
: b2Joint(def)
{
  m_localAnchorA = def->localAnchorA;
  m_localAnchorB = def->localAnchorB;
  m_localXAxisA = def->localAxisA;
  m_localYAxisA = b2Cross(1.0f, m_localXAxisA);

  m_mass = 0.0f;
  m_impulse = 0.0f;
  m_motorMass = 0.0f;
  m_motorImpulse = 0.0f;
  m_springMass = 0.0f;
  m_springImpulse = 0.0f;

```



```
  m_maxMotorTorque = def->maxMotorTorque;
  m_motorSpeed = def->motorSpeed;
  m_enableMotor = def->enableMotor;

  m_frequencyHz = def->frequencyHz;
  m_dampingRatio = def->dampingRatio;

  m_bias = 0.0f;
  m_gamma = 0.0f;

  m_ax.SetZero();
  m_ay.SetZero();
}

void b2WheelJoint::InitVelocityConstraints(const b2SolverData& data)
{
  m_indexA = m_bodyA->m_islandIndex;
  m_indexB = m_bodyB->m_islandIndex;
  m_localCenterA = m_bodyA->m_sweep.localCenter;
  m_localCenterB = m_bodyB->m_sweep.localCenter;
  m_invMassA = m_bodyA->m_invMass;
  m_invMassB = m_bodyB->m_invMass;
  m_invIA = m_bodyA->m_invI;
  m_invIB = m_bodyB->m_invI;

  float32 mA = m_invMassA, mB = m_invMassB;
  float32 iA = m_invIA, iB = m_invIB;

  b2Vec2 cA = data.positions[m_indexA].c;
  float32 aA = data.positions[m_indexA].a;
  b2Vec2 vA = data.velocities[m_indexA].v;
  float32 wA = data.velocities[m_indexA].w;

  b2Vec2 cB = data.positions[m_indexB].c;
  float32 aB = data.positions[m_indexB].a;
  b2Vec2 vB = data.velocities[m_indexB].v;
  float32 wB = data.velocities[m_indexB].w;

  b2Rot qA(aA), qB(aB);

  // Compute the effective masses.
  b2Vec2 rA = b2Mul(qA, m_localAnchorA - m_localCenterA);
  b2Vec2 rB = b2Mul(qB, m_localAnchorB - m_localCenterB);
  b2Vec2 d = cB + rB - cA - rA;

  // Point to line constraint
  {
    m_ay = b2Mul(qA, m_localYAxisA);
    m_sAy = b2Cross(d + rA, m_ay);
    m_sBy = b2Cross(rB, m_ay);

    m_mass = mA + mB + iA * m_sAy * m_sAy + iB * m_sBy * m_sBy;

    if (m_mass > 0.0f)
    {
      m_mass = 1.0f / m_mass;
    }
  }

  // Spring constraint
  m_springMass = 0.0f;
  m_bias = 0.0f;
  m_gamma = 0.0f;
  if (m_frequencyHz > 0.0f)
  {
    m_ax = b2Mul(qA, m_localXAxisA);
```

```
129     m_sAx = b2Cross(d + rA, m_ax);
130     m_sBx = b2Cross(rB, m_ax);
131
132     float32 invMass = mA + mB + iA * m_sAx * m_sAx + iB * m_sBx * m_sBx;
133
134     if (invMass > 0.0f)
135     {
136       m_springMass = 1.0f / invMass;
137
138       float32 C = b2Dot(d, m_ax);
139
140       // Frequency
141       float32 omega = 2.0f * b2_pi * m_frequencyHz;
142
143       // Damping coefficient
144       float32 damp = 2.0f * m_springMass * m_dampingRatio * omega;
145
146       // Spring stiffness
147       float32 k = m_springMass * omega * omega;
148
149       // magic formulas
150       float32 h = data.step.dt;
151       m_gamma = h * (damp + h * k);
152       if (m_gamma > 0.0f)
153       {
154         m_gamma = 1.0f / m_gamma;
155       }
156
157       m_bias = C * h * k * m_gamma;
158
159       m_springMass = invMass + m_gamma;
160       if (m_springMass > 0.0f)
161       {
162         m_springMass = 1.0f / m_springMass;
163       }
164     }
165   }
166   else
167   {
168     m_springImpulse = 0.0f;
169   }
170
171   // Rotational motor
172   if (m_enableMotor)
173   {
174     m_motorMass = iA + iB;
175     if (m_motorMass > 0.0f)
176     {
177       m_motorMass = 1.0f / m_motorMass;
178     }
179   }
180   else
181   {
182     m_motorMass = 0.0f;
183     m_motorImpulse = 0.0f;
184   }
185
186   if (data.step.warmStarting)
187   {
188     // Account for variable time step.
189     m_impulse *= data.step.dtRatio;
190     m_springImpulse *= data.step.dtRatio;
191     m_motorImpulse *= data.step.dtRatio;
192
193     b2Vec2 P = m_impulse * m_ay + m_springImpulse * m_ax;
194     float32 LA = m_impulse * m_sAy + m_springImpulse * m_sAx + m_motorImpulse;
```



```
195     float32 LB = m_impulse * m_sBy + m_springImpulse * m_sBx + m_motorImpulse;
196
197     vA -= m_invMassA * P;
198     wA -= m_invIA * LA;
199
200     vB += m_invMassB * P;
201     wB += m_invIB * LB;
202   }
203   else
204   {
205     m_impulse = 0.0f;
206     m_springImpulse = 0.0f;
207     m_motorImpulse = 0.0f;
208   }
209
210   data.velocities[m_indexA].v = vA;
211   data.velocities[m_indexA].w = wA;
212   data.velocities[m_indexB].v = vB;
213   data.velocities[m_indexB].w = wB;
214 }
215
216 void b2WheelJoint::SolveVelocityConstraints(const b2SolverData& data)
217 {
218   float32 mA = m_invMassA, mB = m_invMassB;
219   float32 iA = m_invIA, iB = m_invIB;
220
221   b2Vec2 vA = data.velocities[m_indexA].v;
222   float32 wA = data.velocities[m_indexA].w;
223   b2Vec2 vB = data.velocities[m_indexB].v;
224   float32 wB = data.velocities[m_indexB].w;
225
226   // Solve spring constraint
227   {
228     float32 Cdot = b2Dot(m_ax, vB - vA) + m_sBx * wB - m_sAx * wA;
229     float32 impulse = -m_springMass * (Cdot + m_bias + m_gamma * m_springImpulse
);
230     m_springImpulse += impulse;
231
232     b2Vec2 P = impulse * m_ax;
233     float32 LA = impulse * m_sAx;
234     float32 LB = impulse * m_sBx;
235
236     vA -= mA * P;
237     wA -= iA * LA;
238
239     vB += mB * P;
240     wB += iB * LB;
241   }
242
243   // Solve rotational motor constraint
244   {
245     float32 Cdot = wB - wA - m_motorSpeed;
246     float32 impulse = -m_motorMass * Cdot;
247
248     float32 oldImpulse = m_motorImpulse;
249     float32 maxImpulse = data.step.dt * m_maxMotorTorque;
250     m_motorImpulse = b2Clamp(m_motorImpulse + impulse, -maxImpulse, maxImpulse);
251     impulse = m_motorImpulse - oldImpulse;
252
253     wA -= iA * impulse;
254     wB += iB * impulse;
255   }
256
257   // Solve point to line constraint
258   {
259     float32 Cdot = b2Dot(m_ay, vB - vA) + m_sBy * wB - m_sAy * wA;
```

```
260       float32 impulse = -m_mass * Cdot;
261       m_impulse += impulse;
262
263       b2Vec2 P = impulse * m_ay;
264       float32 LA = impulse * m_sAy;
265       float32 LB = impulse * m_sBy;
266
267       vA -= mA * P;
268       wA -= iA * LA;
269
270       vB += mB * P;
271       wB += iB * LB;
272     }
273
274     data.velocities[m_indexA].v = vA;
275     data.velocities[m_indexA].w = wA;
276     data.velocities[m_indexB].v = vB;
277     data.velocities[m_indexB].w = wB;
278   }
279
280   bool b2WheelJoint::SolvePositionConstraints(const b2SolverData& data)
281   {
282     b2Vec2 cA = data.positions[m_indexA].c;
283     float32 aA = data.positions[m_indexA].a;
284     b2Vec2 cB = data.positions[m_indexB].c;
285     float32 aB = data.positions[m_indexB].a;
286
287     b2Rot qA(aA), qB(aB);
288
289     b2Vec2 rA = b2Mul(qA, m_localAnchorA - m_localCenterA);
290     b2Vec2 rB = b2Mul(qB, m_localAnchorB - m_localCenterB);
291     b2Vec2 d = (cB - cA) + rB - rA;
292
293     b2Vec2 ay = b2Mul(qA, m_localYAxisA);
294
295     float32 sAy = b2Cross(d + rA, ay);
296     float32 sBy = b2Cross(rB, ay);
297
298     float32 C = b2Dot(d, ay);
299
300     float32 k = m_invMassA + m_invMassB + m_invIA * m_sAy * m_sAy + m_invIB * m_sB
    y * m_sBy;
301
302     float32 impulse;
303     if (k ≠ 0.0f)
304     {
305       impulse = - C / k;
306     }
307     else
308     {
309       impulse = 0.0f;
310     }
311
312     b2Vec2 P = impulse * ay;
313     float32 LA = impulse * sAy;
314     float32 LB = impulse * sBy;
315
316     cA -= m_invMassA * P;
317     aA -= m_invIA * LA;
318     cB += m_invMassB * P;
319     aB += m_invIB * LB;
320
321     data.positions[m_indexA].c = cA;
322     data.positions[m_indexA].a = aA;
323     data.positions[m_indexB].c = cB;
324     data.positions[m_indexB].a = aB;
```



```
325
326     return b2Abs(C) ≤ b2_linearSlop;
327   }
328
329   b2Vec2 b2WheelJoint::GetAnchorA() const
330   {
331     return m_bodyA→GetWorldPoint(m_localAnchorA);
332   }
333
334   b2Vec2 b2WheelJoint::GetAnchorB() const
335   {
336     return m_bodyB→GetWorldPoint(m_localAnchorB);
337   }
338
339   b2Vec2 b2WheelJoint::GetReactionForce(float32 inv_dt) const
340   {
341     return inv_dt * (m_impulse * m_ay + m_springImpulse * m_ax);
342   }
343
344   float32 b2WheelJoint::GetReactionTorque(float32 inv_dt) const
345   {
346     return inv_dt * m_motorImpulse;
347   }
348
349   float32 b2WheelJoint::GetJointTranslation() const
350   {
351     b2Body* bA = m_bodyA;
352     b2Body* bB = m_bodyB;
353
354     b2Vec2 pA = bA→GetWorldPoint(m_localAnchorA);
355     b2Vec2 pB = bB→GetWorldPoint(m_localAnchorB);
356     b2Vec2 d = pB - pA;
357     b2Vec2 axis = bA→GetWorldVector(m_localXAxisA);
358
359     float32 translation = b2Dot(d, axis);
360     return translation;
361   }
362
363   float32 b2WheelJoint::GetJointLinearSpeed() const
364   {
365     b2Body* bA = m_bodyA;
366     b2Body* bB = m_bodyB;
367
368     b2Vec2 rA = b2Mul(bA→m_xf.q, m_localAnchorA - bA→m_sweep.localCenter);
369     b2Vec2 rB = b2Mul(bB→m_xf.q, m_localAnchorB - bB→m_sweep.localCenter);
370     b2Vec2 p1 = bA→m_sweep.c + rA;
371     b2Vec2 p2 = bB→m_sweep.c + rB;
372     b2Vec2 d = p2 - p1;
373     b2Vec2 axis = b2Mul(bA→m_xf.q, m_localXAxisA);
374
375     b2Vec2 vA = bA→m_linearVelocity;
376     b2Vec2 vB = bB→m_linearVelocity;
377     float32 wA = bA→m_angularVelocity;
378     float32 wB = bB→m_angularVelocity;
379
380     float32 speed = b2Dot(d, b2Cross(wA, axis)) + b2Dot(axis, vB + b2Cross(wB, rB)
    - vA - b2Cross(wA, rA));
381     return speed;
382   }
383
384   float32 b2WheelJoint::GetJointAngle() const
385   {
386     b2Body* bA = m_bodyA;
387     b2Body* bB = m_bodyB;
388     return bB→m_sweep.a - bA→m_sweep.a;
389   }
```

```
390
391 float32 b2WheelJoint::GetJointAngularSpeed() const
392 {
393   float32 wA = m_bodyA→m_angularVelocity;
394   float32 wB = m_bodyB→m_angularVelocity;
395   return wB − wA;
396 }
397
398 bool b2WheelJoint::IsMotorEnabled() const
399 {
400   return m_enableMotor;
401 }
402
403 void b2WheelJoint::EnableMotor(bool flag)
404 {
405   if (flag ≠ m_enableMotor)
406   {
407     m_bodyA→SetAwake(true);
408     m_bodyB→SetAwake(true);
409     m_enableMotor = flag;
410   }
411 }
412
413 void b2WheelJoint::SetMotorSpeed(float32 speed)
414 {
415   if (speed ≠ m_motorSpeed)
416   {
417     m_bodyA→SetAwake(true);
418     m_bodyB→SetAwake(true);
419     m_motorSpeed = speed;
420   }
421 }
422
423 void b2WheelJoint::SetMaxMotorTorque(float32 torque)
424 {
425   if (torque ≠ m_maxMotorTorque)
426   {
427     m_bodyA→SetAwake(true);
428     m_bodyB→SetAwake(true);
429     m_maxMotorTorque = torque;
430   }
431 }
432
433 float32 b2WheelJoint::GetMotorTorque(float32 inv_dt) const
434 {
435   return inv_dt * m_motorImpulse;
436 }
437
438 void b2WheelJoint::Dump()
439 {
440   int32 indexA = m_bodyA→m_islandIndex;
441   int32 indexB = m_bodyB→m_islandIndex;
442
443   b2Log(" b2WheelJointDef jd;\n");
444   b2Log(" jd.bodyA = bodies[%d];\n", indexA);
445   b2Log(" jd.bodyB = bodies[%d];\n", indexB);
446   b2Log(" jd.collideConnected = bool(%d);\n", m_collideConnected);
447   b2Log(" jd.localAnchorA.Set(%.15lef, %.15lef);\n", m_localAnchorA.x, m_localAnchorA.y);
448   b2Log(" jd.localAnchorB.Set(%.15lef, %.15lef);\n", m_localAnchorB.x, m_localAnchorB.y);
449   b2Log(" jd.localAxisA.Set(%.15lef, %.15lef);\n", m_localXAxisA.x, m_localXAxisA.y);
450   b2Log(" jd.enableMotor = bool(%d);\n", m_enableMotor);
451   b2Log(" jd.motorSpeed = %.15lef;\n", m_motorSpeed);
452   b2Log(" jd.maxMotorTorque = %.15lef;\n", m_maxMotorTorque);
453   b2Log(" jd.frequencyHz = %.15lef;\n", m_frequencyHz);
454   b2Log(" jd.dampingRatio = %.15lef;\n", m_dampingRatio);
455   b2Log(" joints[%d] = m_world->CreateJoint(&jd);\n", m_index);
```



```
456 }
```

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Joints/b2WeldJoint.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2TimeStep.h"

// Point-to-point constraint
// C = p2 - p1
// Cdot = v2 - v1
//      = v2 + cross(w2, r2) - v1 - cross(w1, r1)
// J = [-I -r1_skew I r2_skew ]
// Identity used:
// w k % (rx i + ry j) = w * (-ry i + rx j)

// Angle constraint
// C = angle2 - angle1 - referenceAngle
// Cdot = w2 - w1
// J = [0 0 -1 0 0 1]
// K = invI1 + invI2

void b2WeldJointDef::Initialize(b2Body* bA, b2Body* bB, const b2Vec2& anchor)
{
  bodyA = bA;
  bodyB = bB;
  localAnchorA = bodyA->GetLocalPoint(anchor);
  localAnchorB = bodyB->GetLocalPoint(anchor);
  referenceAngle = bodyB->GetAngle() - bodyA->GetAngle();
}

b2WeldJoint::b2WeldJoint(const b2WeldJointDef* def)
: b2Joint(def)
{
  m_localAnchorA = def->localAnchorA;
  m_localAnchorB = def->localAnchorB;
  m_referenceAngle = def->referenceAngle;
  m_frequencyHz = def->frequencyHz;
  m_dampingRatio = def->dampingRatio;

  m_impulse.SetZero();
}

void b2WeldJoint::InitVelocityConstraints(const b2SolverData& data)
{
  m_indexA = m_bodyA->m_islandIndex;
  m_indexB = m_bodyB->m_islandIndex;
  m_localCenterA = m_bodyA->m_sweep.localCenter;
  m_localCenterB = m_bodyB->m_sweep.localCenter;
  m_invMassA = m_bodyA->m_invMass;
  m_invMassB = m_bodyB->m_invMass;
  m_invIA = m_bodyA->m_invI;
```



```
  m_invIB = m_bodyB->m_invI;

  float32 aA = data.positions[m_indexA].a;
  b2Vec2 vA = data.velocities[m_indexA].v;
  float32 wA = data.velocities[m_indexA].w;

  float32 aB = data.positions[m_indexB].a;
  b2Vec2 vB = data.velocities[m_indexB].v;
  float32 wB = data.velocities[m_indexB].w;

  b2Rot qA(aA), qB(aB);

  m_rA = b2Mul(qA, m_localAnchorA - m_localCenterA);
  m_rB = b2Mul(qB, m_localAnchorB - m_localCenterB);

  // J = [-I -r1_skew I r2_skew]
  //     [ 0       -1 0       1]
  // r_skew = [-ry; rx]

  // Matlab
  // K = [ mA+r1y^2*iA+mB+r2y^2*iB,  -r1y*iA*r1x-r2y*iB*r2x,          -r1y*iA-r2y*iB]
  //     [  -r1y*iA*r1x-r2y*iB*r2x, mA+r1x^2*iA+mB+r2x^2*iB,           r1x*iA+r2x*iB]
  //     [          -r1y*iA-r2y*iB,           r1x*iA+r2x*iB,                     iA+iB]

  float32 mA = m_invMassA, mB = m_invMassB;
  float32 iA = m_invIA, iB = m_invIB;

  b2Mat33 K;
  K.ex.x = mA + mB + m_rA.y * m_rA.y * iA + m_rB.y * m_rB.y * iB;
  K.ey.x = -m_rA.y * m_rA.x * iA - m_rB.y * m_rB.x * iB;
  K.ez.x = -m_rA.y * iA - m_rB.y * iB;
  K.ex.y = K.ey.x;
  K.ey.y = mA + mB + m_rA.x * m_rA.x * iA + m_rB.x * m_rB.x * iB;
  K.ez.y = m_rA.x * iA + m_rB.x * iB;
  K.ex.z = K.ez.x;
  K.ey.z = K.ez.y;
  K.ez.z = iA + iB;

  if (m_frequencyHz > 0.0f)
  {
    K.GetInverse22(&m_mass);

    float32 invM = iA + iB;
    float32 m = invM > 0.0f ? 1.0f / invM : 0.0f;

    float32 C = aB - aA - m_referenceAngle;

    // Frequency
    float32 omega = 2.0f * b2_pi * m_frequencyHz;

    // Damping coefficient
    float32 d = 2.0f * m * m_dampingRatio * omega;

    // Spring stiffness
    float32 k = m * omega * omega;

    // magic formulas
    float32 h = data.step.dt;
    m_gamma = h * (d + h * k);
    m_gamma = m_gamma ≠ 0.0f ? 1.0f / m_gamma : 0.0f;
    m_bias = C * h * k * m_gamma;

    invM += m_gamma;
```

```
130       m_mass.ez.z = invM ≠ 0.0f ? 1.0f / invM : 0.0f;
131     }
132     else if (K.ez.z ≡ 0.0f)
133     {
134       K.GetInverse22(&m_mass);
135       m_gamma = 0.0f;
136       m_bias = 0.0f;
137     }
138     else
139     {
140       K.GetSymInverse33(&m_mass);
141       m_gamma = 0.0f;
142       m_bias = 0.0f;
143     }
144
145     if (data.step.warmStarting)
146     {
147       // Scale impulses to support a variable time step.
148       m_impulse *= data.step.dtRatio;
149
150       b2Vec2 P(m_impulse.x, m_impulse.y);
151
152       vA −= mA * P;
153       wA −= iA * (b2Cross(m_rA, P) + m_impulse.z);
154
155       vB += mB * P;
156       wB += iB * (b2Cross(m_rB, P) + m_impulse.z);
157     }
158     else
159     {
160       m_impulse.SetZero();
161     }
162
163     data.velocities[m_indexA].v = vA;
164     data.velocities[m_indexA].w = wA;
165     data.velocities[m_indexB].v = vB;
166     data.velocities[m_indexB].w = wB;
167   }
168
169   void b2WeldJoint::SolveVelocityConstraints(const b2SolverData& data)
170   {
171     b2Vec2 vA = data.velocities[m_indexA].v;
172     float32 wA = data.velocities[m_indexA].w;
173     b2Vec2 vB = data.velocities[m_indexB].v;
174     float32 wB = data.velocities[m_indexB].w;
175
176     float32 mA = m_invMassA, mB = m_invMassB;
177     float32 iA = m_invIA, iB = m_invIB;
178
179     if (m_frequencyHz > 0.0f)
180     {
181       float32 Cdot2 = wB − wA;
182
183       float32 impulse2 = −m_mass.ez.z * (Cdot2 + m_bias + m_gamma * m_impulse.z);
184       m_impulse.z += impulse2;
185
186       wA −= iA * impulse2;
187       wB += iB * impulse2;
188
189       b2Vec2 Cdot1 = vB + b2Cross(wB, m_rB) − vA − b2Cross(wA, m_rA);
190
191       b2Vec2 impulse1 = −b2Mul22(m_mass, Cdot1);
192       m_impulse.x += impulse1.x;
193       m_impulse.y += impulse1.y;
194
195       b2Vec2 P = impulse1;
```



```
196
197       vA −= mA * P;
198       wA −= iA * b2Cross(m_rA, P);
199
200       vB += mB * P;
201       wB += iB * b2Cross(m_rB, P);
202     }
203     else
204     {
205       b2Vec2 Cdot1 = vB + b2Cross(wB, m_rB) − vA − b2Cross(wA, m_rA);
206       float32 Cdot2 = wB − wA;
207       b2Vec3 Cdot(Cdot1.x, Cdot1.y, Cdot2);
208
209       b2Vec3 impulse = −b2Mul(m_mass, Cdot);
210       m_impulse += impulse;
211
212       b2Vec2 P(impulse.x, impulse.y);
213
214       vA −= mA * P;
215       wA −= iA * (b2Cross(m_rA, P) + impulse.z);
216
217       vB += mB * P;
218       wB += iB * (b2Cross(m_rB, P) + impulse.z);
219     }
220
221     data.velocities[m_indexA].v = vA;
222     data.velocities[m_indexA].w = wA;
223     data.velocities[m_indexB].v = vB;
224     data.velocities[m_indexB].w = wB;
225   }
226
227   bool b2WeldJoint::SolvePositionConstraints(const b2SolverData& data)
228   {
229     b2Vec2 cA = data.positions[m_indexA].c;
230     float32 aA = data.positions[m_indexA].a;
231     b2Vec2 cB = data.positions[m_indexB].c;
232     float32 aB = data.positions[m_indexB].a;
233
234     b2Rot qA(aA), qB(aB);
235
236     float32 mA = m_invMassA, mB = m_invMassB;
237     float32 iA = m_invIA, iB = m_invIB;
238
239     b2Vec2 rA = b2Mul(qA, m_localAnchorA − m_localCenterA);
240     b2Vec2 rB = b2Mul(qB, m_localAnchorB − m_localCenterB);
241
242     float32 positionError, angularError;
243
244     b2Mat33 K;
245     K.ex.x = mA + mB + rA.y * rA.y * iA + rB.y * rB.y * iB;
246     K.ey.x = −rA.y * rA.x * iA − rB.y * rB.x * iB;
247     K.ez.x = −rA.y * iA − rB.y * iB;
248     K.ex.y = K.ey.x;
249     K.ey.y = mA + mB + rA.x * rA.x * iA + rB.x * rB.x * iB;
250     K.ez.y = rA.x * iA + rB.x * iB;
251     K.ex.z = K.ez.x;
252     K.ey.z = K.ez.y;
253     K.ez.z = iA + iB;
254
255     if (m_frequencyHz > 0.0f)
256     {
257       b2Vec2 C1 =  cB + rB − cA − rA;
258
259       positionError = C1.Length();
260       angularError = 0.0f;
261
```

```
262     b2Vec2 P = −K.Solve22(C1);
263
264     cA −= mA * P;
265     aA −= iA * b2Cross(rA, P);
266
267     cB += mB * P;
268     aB += iB * b2Cross(rB, P);
269   }
270   else
271   {
272     b2Vec2 C1 =  cB + rB − cA − rA;
273     float32 C2 = aB − aA − m_referenceAngle;
274
275     positionError = C1.Length();
276     angularError = b2Abs(C2);
277
278     b2Vec3 C(C1.x, C1.y, C2);
279
280     b2Vec3 impulse;
281     if (K.ez.z > 0.0f)
282     {
283       impulse = −K.Solve33(C);
284     }
285     else
286     {
287       b2Vec2 impulse2 = −K.Solve22(C1);
288       impulse.Set(impulse2.x, impulse2.y, 0.0f);
289     }
290
291     b2Vec2 P(impulse.x, impulse.y);
292
293     cA −= mA * P;
294     aA −= iA * (b2Cross(rA, P) + impulse.z);
295
296     cB += mB * P;
297     aB += iB * (b2Cross(rB, P) + impulse.z);
298   }
299
300   data.positions[m_indexA].c = cA;
301   data.positions[m_indexA].a = aA;
302   data.positions[m_indexB].c = cB;
303   data.positions[m_indexB].a = aB;
304
305   return positionError ≤ b2_linearSlop ∧ angularError ≤ b2_angularSlop;
306 }
307
308 b2Vec2 b2WeldJoint::GetAnchorA() const
309 {
310   return m_bodyA→GetWorldPoint(m_localAnchorA);
311 }
312
313 b2Vec2 b2WeldJoint::GetAnchorB() const
314 {
315   return m_bodyB→GetWorldPoint(m_localAnchorB);
316 }
317
318 b2Vec2 b2WeldJoint::GetReactionForce(float32 inv_dt) const
319 {
320   b2Vec2 P(m_impulse.x, m_impulse.y);
321   return inv_dt * P;
322 }
323
324 float32 b2WeldJoint::GetReactionTorque(float32 inv_dt) const
325 {
326   return inv_dt * m_impulse.z;
327 }
```



```
328
329 void b2WeldJoint::Dump()
330 {
331   int32 indexA = m_bodyA→m_islandIndex;
332   int32 indexB = m_bodyB→m_islandIndex;
333
334   b2Log(" b2WeldJointDef jd;\n");
335   b2Log(" jd.bodyA = bodies[%d];\n", indexA);
336   b2Log(" jd.bodyB = bodies[%d];\n", indexB);
337   b2Log(" jd.collideConnected = bool(%d);\n", m_collideConnected);
338   b2Log(" jd.localAnchorA.Set(%.15lef, %.15lef);\n", m_localAnchorA.x, m_localAnchorA.y);
339   b2Log(" jd.localAnchorB.Set(%.15lef, %.15lef);\n", m_localAnchorB.x, m_localAnchorB.y);
340   b2Log(" jd.referenceAngle = %.15lef;\n", m_referenceAngle);
341   b2Log(" jd.frequencyHz = %.15lef;\n", m_frequencyHz);
342   b2Log(" jd.dampingRatio = %.15lef;\n", m_dampingRatio);
343   b2Log(" joints[%d] = m_world−>CreateJoint(&jd);\n", m_index);
344 }
```

```
/*
* Copyright (c) 2007-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Joints/b2RopeJoint.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2TimeStep.h"


// Limit:
// C = norm(pB - pA) - L
// u = (pB - pA) / norm(pB - pA)
// Cdot = dot(u, vB + cross(wB, rB) - vA - cross(wA, rA))
// J = [-u -cross(rA, u) u cross(rB, u)]
// K = J * invM * JT
//   = invMassA + invIA * cross(rA, u)^2 + invMassB + invIB * cross(rB, u)^2

b2RopeJoint::b2RopeJoint(const b2RopeJointDef* def)
: b2Joint(def)
{
  m_localAnchorA = def->localAnchorA;
  m_localAnchorB = def->localAnchorB;

  m_maxLength = def->maxLength;

  m_mass = 0.0f;
  m_impulse = 0.0f;
  m_state = e_inactiveLimit;
  m_length = 0.0f;
}

void b2RopeJoint::InitVelocityConstraints(const b2SolverData& data)
{
  m_indexA = m_bodyA->m_islandIndex;
  m_indexB = m_bodyB->m_islandIndex;
  m_localCenterA = m_bodyA->m_sweep.localCenter;
  m_localCenterB = m_bodyB->m_sweep.localCenter;
  m_invMassA = m_bodyA->m_invMass;
  m_invMassB = m_bodyB->m_invMass;
  m_invIA = m_bodyA->m_invI;
  m_invIB = m_bodyB->m_invI;

  b2Vec2 cA = data.positions[m_indexA].c;
  float32 aA = data.positions[m_indexA].a;
  b2Vec2 vA = data.velocities[m_indexA].v;
  float32 wA = data.velocities[m_indexA].w;

  b2Vec2 cB = data.positions[m_indexB].c;
  float32 aB = data.positions[m_indexB].a;
  b2Vec2 vB = data.velocities[m_indexB].v;
  float32 wB = data.velocities[m_indexB].w;

```



```
  b2Rot qA(aA), qB(aB);

  m_rA = b2Mul(qA, m_localAnchorA - m_localCenterA);
  m_rB = b2Mul(qB, m_localAnchorB - m_localCenterB);
  m_u = cB + m_rB - cA - m_rA;

  m_length = m_u.Length();

  float32 C = m_length - m_maxLength;
  if (C > 0.0f)
  {
    m_state = e_atUpperLimit;
  }
  else
  {
    m_state = e_inactiveLimit;
  }

  if (m_length > b2_linearSlop)
  {
    m_u *= 1.0f / m_length;
  }
  else
  {
    m_u.SetZero();
    m_mass = 0.0f;
    m_impulse = 0.0f;
    return;
  }

  // Compute effective mass.
  float32 crA = b2Cross(m_rA, m_u);
  float32 crB = b2Cross(m_rB, m_u);
  float32 invMass = m_invMassA + m_invIA * crA * crA + m_invMassB + m_invIB * cr
B * crB;

  m_mass = invMass != 0.0f ? 1.0f / invMass : 0.0f;

  if (data.step.warmStarting)
  {
    // Scale the impulse to support a variable time step.
    m_impulse *= data.step.dtRatio;

    b2Vec2 P = m_impulse * m_u;
    vA -= m_invMassA * P;
    wA -= m_invIA * b2Cross(m_rA, P);
    vB += m_invMassB * P;
    wB += m_invIB * b2Cross(m_rB, P);
  }
  else
  {
    m_impulse = 0.0f;
  }

  data.velocities[m_indexA].v = vA;
  data.velocities[m_indexA].w = wA;
  data.velocities[m_indexB].v = vB;
  data.velocities[m_indexB].w = wB;
}

void b2RopeJoint::SolveVelocityConstraints(const b2SolverData& data)
{
  b2Vec2 vA = data.velocities[m_indexA].v;
  float32 wA = data.velocities[m_indexA].w;
  b2Vec2 vB = data.velocities[m_indexB].v;
  float32 wB = data.velocities[m_indexB].w;
```

```
132
133   // Cdot = dot(u, v + cross(w, r))
134   b2Vec2 vpA = vA + b2Cross(wA, m_rA);
135   b2Vec2 vpB = vB + b2Cross(wB, m_rB);
136   float32 C = m_length - m_maxLength;
137   float32 Cdot = b2Dot(m_u, vpB - vpA);
138
139   // Predictive constraint.
140   if (C < 0.0f)
141   {
142     Cdot += data.step.inv_dt * C;
143   }
144
145   float32 impulse = -m_mass * Cdot;
146   float32 oldImpulse = m_impulse;
147   m_impulse = b2Min(0.0f, m_impulse + impulse);
148   impulse = m_impulse - oldImpulse;
149
150   b2Vec2 P = impulse * m_u;
151   vA -= m_invMassA * P;
152   wA -= m_invIA * b2Cross(m_rA, P);
153   vB += m_invMassB * P;
154   wB += m_invIB * b2Cross(m_rB, P);
155
156   data.velocities[m_indexA].v = vA;
157   data.velocities[m_indexA].w = wA;
158   data.velocities[m_indexB].v = vB;
159   data.velocities[m_indexB].w = wB;
160 }
161
162 bool b2RopeJoint::SolvePositionConstraints(const b2SolverData& data)
163 {
164   b2Vec2 cA = data.positions[m_indexA].c;
165   float32 aA = data.positions[m_indexA].a;
166   b2Vec2 cB = data.positions[m_indexB].c;
167   float32 aB = data.positions[m_indexB].a;
168
169   b2Rot qA(aA), qB(aB);
170
171   b2Vec2 rA = b2Mul(qA, m_localAnchorA - m_localCenterA);
172   b2Vec2 rB = b2Mul(qB, m_localAnchorB - m_localCenterB);
173   b2Vec2 u = cB + rB - cA - rA;
174
175   float32 length = u.Normalize();
176   float32 C = length - m_maxLength;
177
178   C = b2Clamp(C, 0.0f, b2_maxLinearCorrection);
179
180   float32 impulse = -m_mass * C;
181   b2Vec2 P = impulse * u;
182
183   cA -= m_invMassA * P;
184   aA -= m_invIA * b2Cross(rA, P);
185   cB += m_invMassB * P;
186   aB += m_invIB * b2Cross(rB, P);
187
188   data.positions[m_indexA].c = cA;
189   data.positions[m_indexA].a = aA;
190   data.positions[m_indexB].c = cB;
191   data.positions[m_indexB].a = aB;
192
193   return length - m_maxLength < b2_linearSlop;
194 }
195
196 b2Vec2 b2RopeJoint::GetAnchorA() const
197 {
```



```
198   return m_bodyA->GetWorldPoint(m_localAnchorA);
199 }
200
201 b2Vec2 b2RopeJoint::GetAnchorB() const
202 {
203   return m_bodyB->GetWorldPoint(m_localAnchorB);
204 }
205
206 b2Vec2 b2RopeJoint::GetReactionForce(float32 inv_dt) const
207 {
208   b2Vec2 F = (inv_dt * m_impulse) * m_u;
209   return F;
210 }
211
212 float32 b2RopeJoint::GetReactionTorque(float32 inv_dt) const
213 {
214   B2_NOT_USED(inv_dt);
215   return 0.0f;
216 }
217
218 float32 b2RopeJoint::GetMaxLength() const
219 {
220   return m_maxLength;
221 }
222
223 b2LimitState b2RopeJoint::GetLimitState() const
224 {
225   return m_state;
226 }
227
228 void b2RopeJoint::Dump()
229 {
230   int32 indexA = m_bodyA->m_islandIndex;
231   int32 indexB = m_bodyB->m_islandIndex;
232
233   b2Log(" b2RopeJointDef jd;\n");
234   b2Log(" jd.bodyA = bodies[%d];\n", indexA);
235   b2Log(" jd.bodyB = bodies[%d];\n", indexB);
236   b2Log(" jd.collideConnected = bool(%d);\n", m_collideConnected);
237   b2Log(" jd.localAnchorA.Set(%.15lef, %.15lef);\n", m_localAnchorA.x, m_localAnchorA.y);
238   b2Log(" jd.localAnchorB.Set(%.15lef, %.15lef);\n", m_localAnchorB.x, m_localAnchorB.y);
239   b2Log(" jd.maxLength = %.15lef;\n", m_maxLength);
240   b2Log(" joints[%d] = m_world->CreateJoint(&jd);\n", m_index);
241 }
```

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Joints/b2RevoluteJoint.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2TimeStep.h"

// Point-to-point constraint
// C = p2 - p1
// Cdot = v2 - v1
//      = v2 + cross(w2, r2) - v1 - cross(w1, r1)
// J = [-I -r1_skew I r2_skew ]
// Identity used:
// w k % (rx i + ry j) = w * (-ry i + rx j)

// Motor constraint
// Cdot = w2 - w1
// J = [0 0 -1 0 0 1]
// K = invI1 + invI2

void b2RevoluteJointDef::Initialize(b2Body* bA, b2Body* bB, const b2Vec2& anchor
)
{
  bodyA = bA;
  bodyB = bB;
  localAnchorA = bodyA->GetLocalPoint(anchor);
  localAnchorB = bodyB->GetLocalPoint(anchor);
  referenceAngle = bodyB->GetAngle() - bodyA->GetAngle();
}

b2RevoluteJoint::b2RevoluteJoint(const b2RevoluteJointDef* def)
: b2Joint(def)
{
  m_localAnchorA = def->localAnchorA;
  m_localAnchorB = def->localAnchorB;
  m_referenceAngle = def->referenceAngle;

  m_impulse.SetZero();
  m_motorImpulse = 0.0f;

  m_lowerAngle = def->lowerAngle;
  m_upperAngle = def->upperAngle;
  m_maxMotorTorque = def->maxMotorTorque;
  m_motorSpeed = def->motorSpeed;
  m_enableLimit = def->enableLimit;
  m_enableMotor = def->enableMotor;
  m_limitState = e_inactiveLimit;
}

void b2RevoluteJoint::InitVelocityConstraints(const b2SolverData& data)
{
```



```
  m_indexA = m_bodyA->m_islandIndex;
  m_indexB = m_bodyB->m_islandIndex;
  m_localCenterA = m_bodyA->m_sweep.localCenter;
  m_localCenterB = m_bodyB->m_sweep.localCenter;
  m_invMassA = m_bodyA->m_invMass;
  m_invMassB = m_bodyB->m_invMass;
  m_invIA = m_bodyA->m_invI;
  m_invIB = m_bodyB->m_invI;

  float32 aA = data.positions[m_indexA].a;
  b2Vec2 vA = data.velocities[m_indexA].v;
  float32 wA = data.velocities[m_indexA].w;

  float32 aB = data.positions[m_indexB].a;
  b2Vec2 vB = data.velocities[m_indexB].v;
  float32 wB = data.velocities[m_indexB].w;

  b2Rot qA(aA), qB(aB);

  m_rA = b2Mul(qA, m_localAnchorA - m_localCenterA);
  m_rB = b2Mul(qB, m_localAnchorB - m_localCenterB);

  // J = [-I -r1_skew I r2_skew]
  //     [ 0       -1 0       1]
  // r_skew = [-ry; rx]

  // Matlab
  // K = [ mA+r1y^2*iA+mB+r2y^2*iB,  -r1y*iA*r1x-r2y*iB*r2x,          -r1y*iA-r2
y*iB]
  //     [  -r1y*iA*r1x-r2y*iB*r2x, mA+r1x^2*iA+mB+r2x^2*iB,           r1x*iA+r2
x*iB]
  //     [          -r1y*iA-r2y*iB,           r1x*iA+r2x*iB,                   i
A+iB]

  float32 mA = m_invMassA, mB = m_invMassB;
  float32 iA = m_invIA, iB = m_invIB;

  bool fixedRotation = (iA + iB == 0.0f);

  m_mass.ex.x = mA + mB + m_rA.y * m_rA.y * iA + m_rB.y * m_rB.y * iB;
  m_mass.ey.x = -m_rA.y * m_rA.x * iA - m_rB.y * m_rB.x * iB;
  m_mass.ez.x = -m_rA.y * iA - m_rB.y * iB;
  m_mass.ex.y = m_mass.ey.x;
  m_mass.ey.y = mA + mB + m_rA.x * m_rA.x * iA + m_rB.x * m_rB.x * iB;
  m_mass.ez.y = m_rA.x * iA + m_rB.x * iB;
  m_mass.ex.z = m_mass.ez.x;
  m_mass.ey.z = m_mass.ez.y;
  m_mass.ez.z = iA + iB;

  m_motorMass = iA + iB;
  if (m_motorMass > 0.0f)
  {
    m_motorMass = 1.0f / m_motorMass;
  }

  if (m_enableMotor == false || fixedRotation)
  {
    m_motorImpulse = 0.0f;
  }

  if (m_enableLimit && fixedRotation == false)
  {
    float32 jointAngle = aB - aA - m_referenceAngle;
    if (b2Abs(m_upperAngle - m_lowerAngle) < 2.0f * b2_angularSlop)
    {
      m_limitState = e_equalLimits;
```

```
129       }
130       else if (jointAngle ≤ m_lowerAngle)
131       {
132         if (m_limitState ≠ e_atLowerLimit)
133         {
134           m_impulse.z = 0.0f;
135         }
136         m_limitState = e_atLowerLimit;
137       }
138       else if (jointAngle ≥ m_upperAngle)
139       {
140         if (m_limitState ≠ e_atUpperLimit)
141         {
142           m_impulse.z = 0.0f;
143         }
144         m_limitState = e_atUpperLimit;
145       }
146       else
147       {
148         m_limitState = e_inactiveLimit;
149         m_impulse.z = 0.0f;
150       }
151     }
152     else
153     {
154       m_limitState = e_inactiveLimit;
155     }
156
157     if (data.step.warmStarting)
158     {
159       // Scale impulses to support a variable time step.
160       m_impulse *= data.step.dtRatio;
161       m_motorImpulse *= data.step.dtRatio;
162
163       b2Vec2 P(m_impulse.x, m_impulse.y);
164
165       vA −= mA * P;
166       wA −= iA * (b2Cross(m_rA, P) + m_motorImpulse + m_impulse.z);
167
168       vB += mB * P;
169       wB += iB * (b2Cross(m_rB, P) + m_motorImpulse + m_impulse.z);
170     }
171     else
172     {
173       m_impulse.SetZero();
174       m_motorImpulse = 0.0f;
175     }
176
177     data.velocities[m_indexA].v = vA;
178     data.velocities[m_indexA].w = wA;
179     data.velocities[m_indexB].v = vB;
180     data.velocities[m_indexB].w = wB;
181   }
182
183   void b2RevoluteJoint::SolveVelocityConstraints(const b2SolverData& data)
184   {
185     b2Vec2 vA = data.velocities[m_indexA].v;
186     float32 wA = data.velocities[m_indexA].w;
187     b2Vec2 vB = data.velocities[m_indexB].v;
188     float32 wB = data.velocities[m_indexB].w;
189
190     float32 mA = m_invMassA, mB = m_invMassB;
191     float32 iA = m_invIA, iB = m_invIB;
192
193     bool fixedRotation = (iA + iB ≡ 0.0f);
194
```



```
195     // Solve motor constraint.
196     if (m_enableMotor ∧ m_limitState ≠ e_equalLimits ∧ fixedRotation ≡ false)
197     {
198       float32 Cdot = wB − wA − m_motorSpeed;
199       float32 impulse = −m_motorMass * Cdot;
200       float32 oldImpulse = m_motorImpulse;
201       float32 maxImpulse = data.step.dt * m_maxMotorTorque;
202       m_motorImpulse = b2Clamp(m_motorImpulse + impulse, −maxImpulse, maxImpulse);
203       impulse = m_motorImpulse − oldImpulse;
204
205       wA −= iA * impulse;
206       wB += iB * impulse;
207     }
208
209     // Solve limit constraint.
210     if (m_enableLimit ∧ m_limitState ≠ e_inactiveLimit ∧ fixedRotation ≡ false)
211     {
212       b2Vec2 Cdot1 = vB + b2Cross(wB, m_rB) − vA − b2Cross(wA, m_rA);
213       float32 Cdot2 = wB − wA;
214       b2Vec3 Cdot(Cdot1.x, Cdot1.y, Cdot2);
215
216       b2Vec3 impulse = −m_mass.Solve33(Cdot);
217
218       if (m_limitState ≡ e_equalLimits)
219       {
220         m_impulse += impulse;
221       }
222       else if (m_limitState ≡ e_atLowerLimit)
223       {
224         float32 newImpulse = m_impulse.z + impulse.z;
225         if (newImpulse < 0.0f)
226         {
227           b2Vec2 rhs = −Cdot1 + m_impulse.z * b2Vec2(m_mass.ez.x, m_mass.ez.y);
228           b2Vec2 reduced = m_mass.Solve22(rhs);
229           impulse.x = reduced.x;
230           impulse.y = reduced.y;
231           impulse.z = −m_impulse.z;
232           m_impulse.x += reduced.x;
233           m_impulse.y += reduced.y;
234           m_impulse.z = 0.0f;
235         }
236         else
237         {
238           m_impulse += impulse;
239         }
240       }
241       else if (m_limitState ≡ e_atUpperLimit)
242       {
243         float32 newImpulse = m_impulse.z + impulse.z;
244         if (newImpulse > 0.0f)
245         {
246           b2Vec2 rhs = −Cdot1 + m_impulse.z * b2Vec2(m_mass.ez.x, m_mass.ez.y);
247           b2Vec2 reduced = m_mass.Solve22(rhs);
248           impulse.x = reduced.x;
249           impulse.y = reduced.y;
250           impulse.z = −m_impulse.z;
251           m_impulse.x += reduced.x;
252           m_impulse.y += reduced.y;
253           m_impulse.z = 0.0f;
254         }
255         else
256         {
257           m_impulse += impulse;
258         }
259       }
260
```

```
261       b2Vec2 P(impulse.x, impulse.y);
262
263       vA -= mA * P;
264       wA -= iA * (b2Cross(m_rA, P) + impulse.z);
265
266       vB += mB * P;
267       wB += iB * (b2Cross(m_rB, P) + impulse.z);
268     }
269     else
270     {
271       // Solve point-to-point constraint
272       b2Vec2 Cdot = vB + b2Cross(wB, m_rB) - vA - b2Cross(wA, m_rA);
273       b2Vec2 impulse = m_mass.Solve22(-Cdot);
274
275       m_impulse.x += impulse.x;
276       m_impulse.y += impulse.y;
277
278       vA -= mA * impulse;
279       wA -= iA * b2Cross(m_rA, impulse);
280
281       vB += mB * impulse;
282       wB += iB * b2Cross(m_rB, impulse);
283     }
284
285     data.velocities[m_indexA].v = vA;
286     data.velocities[m_indexA].w = wA;
287     data.velocities[m_indexB].v = vB;
288     data.velocities[m_indexB].w = wB;
289   }
290
291   bool b2RevoluteJoint::SolvePositionConstraints(const b2SolverData& data)
292   {
293     b2Vec2 cA = data.positions[m_indexA].c;
294     float32 aA = data.positions[m_indexA].a;
295     b2Vec2 cB = data.positions[m_indexB].c;
296     float32 aB = data.positions[m_indexB].a;
297
298     b2Rot qA(aA), qB(aB);
299
300     float32 angularError = 0.0f;
301     float32 positionError = 0.0f;
302
303     bool fixedRotation = (m_invIA + m_invIB ≡ 0.0f);
304
305     // Solve angular limit constraint.
306     if (m_enableLimit ∧ m_limitState ≠ e_inactiveLimit ∧ fixedRotation ≡ false)
307     {
308       float32 angle = aB - aA - m_referenceAngle;
309       float32 limitImpulse = 0.0f;
310
311       if (m_limitState ≡ e_equalLimits)
312       {
313         // Prevent large angular corrections
314         float32 C = b2Clamp(angle - m_lowerAngle, -b2_maxAngularCorrection, b2_max
    AngularCorrection);
315         limitImpulse = -m_motorMass * C;
316         angularError = b2Abs(C);
317       }
318       else if (m_limitState ≡ e_atLowerLimit)
319       {
320         float32 C = angle - m_lowerAngle;
321         angularError = -C;
322
323         // Prevent large angular corrections and allow some slop.
324         C = b2Clamp(C + b2_angularSlop, -b2_maxAngularCorrection, 0.0f);
325         limitImpulse = -m_motorMass * C;
```



```
326       }
327       else if (m_limitState ≡ e_atUpperLimit)
328       {
329         float32 C = angle - m_upperAngle;
330         angularError = C;
331
332         // Prevent large angular corrections and allow some slop.
333         C = b2Clamp(C - b2_angularSlop, 0.0f, b2_maxAngularCorrection);
334         limitImpulse = -m_motorMass * C;
335       }
336
337       aA -= m_invIA * limitImpulse;
338       aB += m_invIB * limitImpulse;
339     }
340
341     // Solve point-to-point constraint.
342     {
343       qA.Set(aA);
344       qB.Set(aB);
345       b2Vec2 rA = b2Mul(qA, m_localAnchorA - m_localCenterA);
346       b2Vec2 rB = b2Mul(qB, m_localAnchorB - m_localCenterB);
347
348       b2Vec2 C = cB + rB - cA - rA;
349       positionError = C.Length();
350
351       float32 mA = m_invMassA, mB = m_invMassB;
352       float32 iA = m_invIA, iB = m_invIB;
353
354       b2Mat22 K;
355       K.ex.x = mA + mB + iA * rA.y * rA.y + iB * rB.y * rB.y;
356       K.ex.y = -iA * rA.x * rA.y - iB * rB.x * rB.y;
357       K.ey.x = K.ex.y;
358       K.ey.y = mA + mB + iA * rA.x * rA.x + iB * rB.x * rB.x;
359
360       b2Vec2 impulse = -K.Solve(C);
361
362       cA -= mA * impulse;
363       aA -= iA * b2Cross(rA, impulse);
364
365       cB += mB * impulse;
366       aB += iB * b2Cross(rB, impulse);
367     }
368
369     data.positions[m_indexA].c = cA;
370     data.positions[m_indexA].a = aA;
371     data.positions[m_indexB].c = cB;
372     data.positions[m_indexB].a = aB;
373
374     return positionError ≤ b2_linearSlop ∧ angularError ≤ b2_angularSlop;
375   }
376
377   b2Vec2 b2RevoluteJoint::GetAnchorA() const
378   {
379     return m_bodyA→GetWorldPoint(m_localAnchorA);
380   }
381
382   b2Vec2 b2RevoluteJoint::GetAnchorB() const
383   {
384     return m_bodyB→GetWorldPoint(m_localAnchorB);
385   }
386
387   b2Vec2 b2RevoluteJoint::GetReactionForce(float32 inv_dt) const
388   {
389     b2Vec2 P(m_impulse.x, m_impulse.y);
390     return inv_dt * P;
391   }
```

```
392
393 float32 b2RevoluteJoint::GetReactionTorque(float32 inv_dt) const
394 {
395   return inv_dt * m_impulse.z;
396 }
397
398 float32 b2RevoluteJoint::GetJointAngle() const
399 {
400   b2Body* bA = m_bodyA;
401   b2Body* bB = m_bodyB;
402   return bB→m_sweep.a − bA→m_sweep.a − m_referenceAngle;
403 }
404
405 float32 b2RevoluteJoint::GetJointSpeed() const
406 {
407   b2Body* bA = m_bodyA;
408   b2Body* bB = m_bodyB;
409   return bB→m_angularVelocity − bA→m_angularVelocity;
410 }
411
412 bool b2RevoluteJoint::IsMotorEnabled() const
413 {
414   return m_enableMotor;
415 }
416
417 void b2RevoluteJoint::EnableMotor(bool flag)
418 {
419   if (flag ≠ m_enableMotor)
420   {
421     m_bodyA→SetAwake(true);
422     m_bodyB→SetAwake(true);
423     m_enableMotor = flag;
424   }
425 }
426
427 float32 b2RevoluteJoint::GetMotorTorque(float32 inv_dt) const
428 {
429   return inv_dt * m_motorImpulse;
430 }
431
432 void b2RevoluteJoint::SetMotorSpeed(float32 speed)
433 {
434   if (speed ≠ m_motorSpeed)
435   {
436     m_bodyA→SetAwake(true);
437     m_bodyB→SetAwake(true);
438     m_motorSpeed = speed;
439   }
440 }
441
442 void b2RevoluteJoint::SetMaxMotorTorque(float32 torque)
443 {
444   if (torque ≠ m_maxMotorTorque)
445   {
446     m_bodyA→SetAwake(true);
447     m_bodyB→SetAwake(true);
448     m_maxMotorTorque = torque;
449   }
450 }
451
452 bool b2RevoluteJoint::IsLimitEnabled() const
453 {
454   return m_enableLimit;
455 }
456
457 void b2RevoluteJoint::EnableLimit(bool flag)
```



```
458 {
459   if (flag ≠ m_enableLimit)
460   {
461     m_bodyA→SetAwake(true);
462     m_bodyB→SetAwake(true);
463     m_enableLimit = flag;
464     m_impulse.z = 0.0f;
465   }
466 }
467
468 float32 b2RevoluteJoint::GetLowerLimit() const
469 {
470   return m_lowerAngle;
471 }
472
473 float32 b2RevoluteJoint::GetUpperLimit() const
474 {
475   return m_upperAngle;
476 }
477
478 void b2RevoluteJoint::SetLimits(float32 lower, float32 upper)
479 {
480   b2Assert(lower ≤ upper);
481
482   if (lower ≠ m_lowerAngle ∨ upper ≠ m_upperAngle)
483   {
484     m_bodyA→SetAwake(true);
485     m_bodyB→SetAwake(true);
486     m_impulse.z = 0.0f;
487     m_lowerAngle = lower;
488     m_upperAngle = upper;
489   }
490 }
491
492 void b2RevoluteJoint::Dump()
493 {
494   int32 indexA = m_bodyA→m_islandIndex;
495   int32 indexB = m_bodyB→m_islandIndex;
496
497   b2Log(" b2RevoluteJointDef jd;\n");
498   b2Log(" jd.bodyA = bodies[%d];\n", indexA);
499   b2Log(" jd.bodyB = bodies[%d];\n", indexB);
500   b2Log(" jd.collideConnected = bool(%d);\n", m_collideConnected);
501   b2Log(" jd.localAnchorA.Set(%.15lef, %.15lef);\n", m_localAnchorA.x, m_localAnchorA.y);
502   b2Log(" jd.localAnchorB.Set(%.15lef, %.15lef);\n", m_localAnchorB.x, m_localAnchorB.y);
503   b2Log(" jd.referenceAngle = %.15lef;\n", m_referenceAngle);
504   b2Log(" jd.enableLimit = bool(%d);\n", m_enableLimit);
505   b2Log(" jd.lowerAngle = %.15lef;\n", m_lowerAngle);
506   b2Log(" jd.upperAngle = %.15lef;\n", m_upperAngle);
507   b2Log(" jd.enableMotor = bool(%d);\n", m_enableMotor);
508   b2Log(" jd.motorSpeed = %.15lef;\n", m_motorSpeed);
509   b2Log(" jd.maxMotorTorque = %.15lef;\n", m_maxMotorTorque);
510   b2Log(" joints[%d] = m_world->CreateJoint(&jd);\n", m_index);
511 }
```

```
/*
* Copyright (c) 2007 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Joints/b2PulleyJoint.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2TimeStep.h"

// Pulley:
// length1 = norm(p1 - s1)
// length2 = norm(p2 - s2)
// C0 = (length1 + ratio * length2)_initial
// C = C0 - (length1 + ratio * length2)
// u1 = (p1 - s1) / norm(p1 - s1)
// u2 = (p2 - s2) / norm(p2 - s2)
// Cdot = -dot(u1, v1 + cross(w1, r1)) - ratio * dot(u2, v2 + cross(w2, r2))
// J = -[u1 cross(r1, u1) ratio * u2  ratio * cross(r2, u2)]
// K = J * invM * JT
//   = invMass1 + invI1 * cross(r1, u1)^2 + ratio^2 * (invMass2 + invI2 * cross(
r2, u2)^2)

void b2PulleyJointDef::Initialize(b2Body* bA, b2Body* bB,
        const b2Vec2& groundA, const b2Vec2& groundB,
        const b2Vec2& anchorA, const b2Vec2& anchorB,
        float32 r)
{
  bodyA = bA;
  bodyB = bB;
  groundAnchorA = groundA;
  groundAnchorB = groundB;
  localAnchorA = bodyA->GetLocalPoint(anchorA);
  localAnchorB = bodyB->GetLocalPoint(anchorB);
  b2Vec2 dA = anchorA - groundA;
  lengthA = dA.Length();
  b2Vec2 dB = anchorB - groundB;
  lengthB = dB.Length();
  ratio = r;
  b2Assert(ratio > b2_epsilon);
}

b2PulleyJoint::b2PulleyJoint(const b2PulleyJointDef* def)
: b2Joint(def)
{
  m_groundAnchorA = def->groundAnchorA;
  m_groundAnchorB = def->groundAnchorB;
  m_localAnchorA = def->localAnchorA;
  m_localAnchorB = def->localAnchorB;

  m_lengthA = def->lengthA;
  m_lengthB = def->lengthB;

  b2Assert(def->ratio != 0.0f);
```



```
  m_ratio = def->ratio;

  m_constant = def->lengthA + m_ratio * def->lengthB;

  m_impulse = 0.0f;
}

void b2PulleyJoint::InitVelocityConstraints(const b2SolverData& data)
{
  m_indexA = m_bodyA->m_islandIndex;
  m_indexB = m_bodyB->m_islandIndex;
  m_localCenterA = m_bodyA->m_sweep.localCenter;
  m_localCenterB = m_bodyB->m_sweep.localCenter;
  m_invMassA = m_bodyA->m_invMass;
  m_invMassB = m_bodyB->m_invMass;
  m_invIA = m_bodyA->m_invI;
  m_invIB = m_bodyB->m_invI;

  b2Vec2 cA = data.positions[m_indexA].c;
  float32 aA = data.positions[m_indexA].a;
  b2Vec2 vA = data.velocities[m_indexA].v;
  float32 wA = data.velocities[m_indexA].w;

  b2Vec2 cB = data.positions[m_indexB].c;
  float32 aB = data.positions[m_indexB].a;
  b2Vec2 vB = data.velocities[m_indexB].v;
  float32 wB = data.velocities[m_indexB].w;

  b2Rot qA(aA), qB(aB);

  m_rA = b2Mul(qA, m_localAnchorA - m_localCenterA);
  m_rB = b2Mul(qB, m_localAnchorB - m_localCenterB);

  // Get the pulley axes.
  m_uA = cA + m_rA - m_groundAnchorA;
  m_uB = cB + m_rB - m_groundAnchorB;

  float32 lengthA = m_uA.Length();
  float32 lengthB = m_uB.Length();

  if (lengthA > 10.0f * b2_linearSlop)
  {
    m_uA *= 1.0f / lengthA;
  }
  else
  {
    m_uA.SetZero();
  }

  if (lengthB > 10.0f * b2_linearSlop)
  {
    m_uB *= 1.0f / lengthB;
  }
  else
  {
    m_uB.SetZero();
  }

  // Compute effective mass.
  float32 ruA = b2Cross(m_rA, m_uA);
  float32 ruB = b2Cross(m_rB, m_uB);

  float32 mA = m_invMassA + m_invIA * ruA * ruA;
  float32 mB = m_invMassB + m_invIB * ruB * ruB;

  m_mass = mA + m_ratio * m_ratio * mB;
```

```
132
133   if (m_mass > 0.0f)
134   {
135     m_mass = 1.0f / m_mass;
136   }
137
138   if (data.step.warmStarting)
139   {
140     // Scale impulses to support variable time steps.
141     m_impulse *= data.step.dtRatio;
142
143     // Warm starting.
144     b2Vec2 PA = -(m_impulse) * m_uA;
145     b2Vec2 PB = (-m_ratio * m_impulse) * m_uB;
146
147     vA += m_invMassA * PA;
148     wA += m_invIA * b2Cross(m_rA, PA);
149     vB += m_invMassB * PB;
150     wB += m_invIB * b2Cross(m_rB, PB);
151   }
152   else
153   {
154     m_impulse = 0.0f;
155   }
156
157   data.velocities[m_indexA].v = vA;
158   data.velocities[m_indexA].w = wA;
159   data.velocities[m_indexB].v = vB;
160   data.velocities[m_indexB].w = wB;
161 }
162
163 void b2PulleyJoint::SolveVelocityConstraints(const b2SolverData& data)
164 {
165   b2Vec2 vA = data.velocities[m_indexA].v;
166   float32 wA = data.velocities[m_indexA].w;
167   b2Vec2 vB = data.velocities[m_indexB].v;
168   float32 wB = data.velocities[m_indexB].w;
169
170   b2Vec2 vpA = vA + b2Cross(wA, m_rA);
171   b2Vec2 vpB = vB + b2Cross(wB, m_rB);
172
173   float32 Cdot = -b2Dot(m_uA, vpA) - m_ratio * b2Dot(m_uB, vpB);
174   float32 impulse = -m_mass * Cdot;
175   m_impulse += impulse;
176
177   b2Vec2 PA = -impulse * m_uA;
178   b2Vec2 PB = -m_ratio * impulse * m_uB;
179   vA += m_invMassA * PA;
180   wA += m_invIA * b2Cross(m_rA, PA);
181   vB += m_invMassB * PB;
182   wB += m_invIB * b2Cross(m_rB, PB);
183
184   data.velocities[m_indexA].v = vA;
185   data.velocities[m_indexA].w = wA;
186   data.velocities[m_indexB].v = vB;
187   data.velocities[m_indexB].w = wB;
188 }
189
190 bool b2PulleyJoint::SolvePositionConstraints(const b2SolverData& data)
191 {
192   b2Vec2 cA = data.positions[m_indexA].c;
193   float32 aA = data.positions[m_indexA].a;
194   b2Vec2 cB = data.positions[m_indexB].c;
195   float32 aB = data.positions[m_indexB].a;
196
197   b2Rot qA(aA), qB(aB);
```



```
198
199   b2Vec2 rA = b2Mul(qA, m_localAnchorA - m_localCenterA);
200   b2Vec2 rB = b2Mul(qB, m_localAnchorB - m_localCenterB);
201
202   // Get the pulley axes.
203   b2Vec2 uA = cA + rA - m_groundAnchorA;
204   b2Vec2 uB = cB + rB - m_groundAnchorB;
205
206   float32 lengthA = uA.Length();
207   float32 lengthB = uB.Length();
208
209   if (lengthA > 10.0f * b2_linearSlop)
210   {
211     uA *= 1.0f / lengthA;
212   }
213   else
214   {
215     uA.SetZero();
216   }
217
218   if (lengthB > 10.0f * b2_linearSlop)
219   {
220     uB *= 1.0f / lengthB;
221   }
222   else
223   {
224     uB.SetZero();
225   }
226
227   // Compute effective mass.
228   float32 ruA = b2Cross(rA, uA);
229   float32 ruB = b2Cross(rB, uB);
230
231   float32 mA = m_invMassA + m_invIA * ruA * ruA;
232   float32 mB = m_invMassB + m_invIB * ruB * ruB;
233
234   float32 mass = mA + m_ratio * m_ratio * mB;
235
236   if (mass > 0.0f)
237   {
238     mass = 1.0f / mass;
239   }
240
241   float32 C = m_constant - lengthA - m_ratio * lengthB;
242   float32 linearError = b2Abs(C);
243
244   float32 impulse = -mass * C;
245
246   b2Vec2 PA = -impulse * uA;
247   b2Vec2 PB = -m_ratio * impulse * uB;
248
249   cA += m_invMassA * PA;
250   aA += m_invIA * b2Cross(rA, PA);
251   cB += m_invMassB * PB;
252   aB += m_invIB * b2Cross(rB, PB);
253
254   data.positions[m_indexA].c = cA;
255   data.positions[m_indexA].a = aA;
256   data.positions[m_indexB].c = cB;
257   data.positions[m_indexB].a = aB;
258
259   return linearError < b2_linearSlop;
260 }
261
262 b2Vec2 b2PulleyJoint::GetAnchorA() const
263 {
```

```
264   return m_bodyA→GetWorldPoint(m_localAnchorA);
265 }
266
267 b2Vec2 b2PulleyJoint::GetAnchorB() const
268 {
269   return m_bodyB→GetWorldPoint(m_localAnchorB);
270 }
271
272 b2Vec2 b2PulleyJoint::GetReactionForce(float32 inv_dt) const
273 {
274   b2Vec2 P = m_impulse * m_uB;
275   return inv_dt * P;
276 }
277
278 float32 b2PulleyJoint::GetReactionTorque(float32 inv_dt) const
279 {
280   B2_NOT_USED(inv_dt);
281   return 0.0f;
282 }
283
284 b2Vec2 b2PulleyJoint::GetGroundAnchorA() const
285 {
286   return m_groundAnchorA;
287 }
288
289 b2Vec2 b2PulleyJoint::GetGroundAnchorB() const
290 {
291   return m_groundAnchorB;
292 }
293
294 float32 b2PulleyJoint::GetLengthA() const
295 {
296   return m_lengthA;
297 }
298
299 float32 b2PulleyJoint::GetLengthB() const
300 {
301   return m_lengthB;
302 }
303
304 float32 b2PulleyJoint::GetRatio() const
305 {
306   return m_ratio;
307 }
308
309 float32 b2PulleyJoint::GetCurrentLengthA() const
310 {
311   b2Vec2 p = m_bodyA→GetWorldPoint(m_localAnchorA);
312   b2Vec2 s = m_groundAnchorA;
313   b2Vec2 d = p − s;
314   return d.Length();
315 }
316
317 float32 b2PulleyJoint::GetCurrentLengthB() const
318 {
319   b2Vec2 p = m_bodyB→GetWorldPoint(m_localAnchorB);
320   b2Vec2 s = m_groundAnchorB;
321   b2Vec2 d = p − s;
322   return d.Length();
323 }
324
325 void b2PulleyJoint::Dump()
326 {
327   int32 indexA = m_bodyA→m_islandIndex;
328   int32 indexB = m_bodyB→m_islandIndex;
329
```



```
330   b2Log(" b2PulleyJointDef jd;\n");
331   b2Log(" jd.bodyA = bodies[%d];\n", indexA);
332   b2Log(" jd.bodyB = bodies[%d];\n", indexB);
333   b2Log(" jd.collideConnected = bool(%d);\n", m_collideConnected);
334   b2Log(" jd.groundAnchorA.Set(%.15lef, %.15lef);\n", m_groundAnchorA.x, m_groundAnchorA.y)
    ;
335   b2Log(" jd.groundAnchorB.Set(%.15lef, %.15lef);\n", m_groundAnchorB.x, m_groundAnchorB.y)
    ;
336   b2Log(" jd.localAnchorA.Set(%.15lef, %.15lef);\n", m_localAnchorA.x, m_localAnchorA.y);
337   b2Log(" jd.localAnchorB.Set(%.15lef, %.15lef);\n", m_localAnchorB.x, m_localAnchorB.y);
338   b2Log(" jd.lengthA = %.15lef;\n", m_lengthA);
339   b2Log(" jd.lengthB = %.15lef;\n", m_lengthB);
340   b2Log(" jd.ratio = %.15lef;\n", m_ratio);
341   b2Log(" joints[%d] = m_world−>CreateJoint(&jd);\n", m_index);
342 }
343
344 void b2PulleyJoint::ShiftOrigin(const b2Vec2& newOrigin)
345 {
346   m_groundAnchorA −= newOrigin;
347   m_groundAnchorB −= newOrigin;
348 }
```

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Joints/b2PrismaticJoint.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2TimeStep.h"

// Linear constraint (point-to-line)
// d = p2 - p1 = x2 + r2 - x1 - r1
// C = dot(perp, d)
// Cdot = dot(d, cross(w1, perp)) + dot(perp, v2 + cross(w2, r2) - v1 - cross(w1, r1))
//      = -dot(perp, v1) - dot(cross(d + r1, perp), w1) + dot(perp, v2) + dot(cross(r2, perp), v2)
// J = [-perp, -cross(d + r1, perp), perp, cross(r2,perp)]
//
// Angular constraint
// C = a2 - a1 + a_initial
// Cdot = w2 - w1
// J = [0 0 -1 0 0 1]
//
// K = J * invM * JT
//
// J = [-a -s1 a s2]
//     [0  -1  0  1]
// a = perp
// s1 = cross(d + r1, a) = cross(p2 - x1, a)
// s2 = cross(r2, a) = cross(p2 - x2, a)


// Motor/Limit linear constraint
// C = dot(ax1, d)
// Cdot = = -dot(ax1, v1) - dot(cross(d + r1, ax1), w1) + dot(ax1, v2) + dot(cross(r2, ax1), v2)
// J = [-ax1 -cross(d+r1,ax1) ax1 cross(r2,ax1)]

// Block Solver
// We develop a block solver that includes the joint limit. This makes the limit stiff (inelastic) even
// when the mass has poor distribution (leading to large torques about the joint anchor points).
//
// The Jacobian has 3 rows:
// J = [-uT -s1 uT s2] // linear
//     [0   -1   0  1] // angular
//     [-vT -a1 vT a2] // limit
//
// u = perp
// v = axis
// s1 = cross(d + r1, u), s2 = cross(r2, u)
// a1 = cross(d + r1, v), a2 = cross(r2, v)
```



```

// M * (v2 - v1) = JT * df
// J * v2 = bias
//
// v2 = v1 + invM * JT * df
// J * (v1 + invM * JT * df) = bias
// K * df = bias - J * v1 = -Cdot
// K = J * invM * JT
// Cdot = J * v1 - bias
//
// Now solve for f2.
// df = f2 - f1
// K * (f2 - f1) = -Cdot
// f2 = invK * (-Cdot) + f1
//
// Clamp accumulated limit impulse.
// lower: f2(3) = max(f2(3), 0)
// upper: f2(3) = min(f2(3), 0)
//
// Solve for correct f2(1:2)
// K(1:2, 1:2) * f2(1:2) = -Cdot(1:2) - K(1:2,3) * f2(3) + K(1:2,1:3) * f1
//                       = -Cdot(1:2) - K(1:2,3) * f2(3) + K(1:2,1:2) * f1(1:2) + K(1:2,3) * f1(3)
// K(1:2, 1:2) * f2(1:2) = -Cdot(1:2) - K(1:2,3) * (f2(3) - f1(3)) + K(1:2,1:2) * f1(1:2)
// f2(1:2) = invK(1:2,1:2) * (-Cdot(1:2) - K(1:2,3) * (f2(3) - f1(3))) + f1(1:2)
//
// Now compute impulse to be applied:
// df = f2 - f1

void b2PrismaticJointDef::Initialize(b2Body* bA, b2Body* bB, const b2Vec2& anchor, const b2Vec2& axis)
{
  bodyA = bA;
  bodyB = bB;
  localAnchorA = bodyA->GetLocalPoint(anchor);
  localAnchorB = bodyB->GetLocalPoint(anchor);
  localAxisA = bodyA->GetLocalVector(axis);
  referenceAngle = bodyB->GetAngle() - bodyA->GetAngle();
}

b2PrismaticJoint::b2PrismaticJoint(const b2PrismaticJointDef* def)
: b2Joint(def)
{
  m_localAnchorA = def->localAnchorA;
  m_localAnchorB = def->localAnchorB;
  m_localXAxisA = def->localAxisA;
  m_localXAxisA.Normalize();
  m_localYAxisA = b2Cross(1.0f, m_localXAxisA);
  m_referenceAngle = def->referenceAngle;

  m_impulse.SetZero();
  m_motorMass = 0.0f;
  m_motorImpulse = 0.0f;

  m_lowerTranslation = def->lowerTranslation;
  m_upperTranslation = def->upperTranslation;
  m_maxMotorForce = def->maxMotorForce;
  m_motorSpeed = def->motorSpeed;
  m_enableLimit = def->enableLimit;
  m_enableMotor = def->enableMotor;
  m_limitState = e_inactiveLimit;

  m_axis.SetZero();
  m_perp.SetZero();
}
```

```
125
126 void b2PrismaticJoint::InitVelocityConstraints(const b2SolverData& data)
127 {
128   m_indexA = m_bodyA→m_islandIndex;
129   m_indexB = m_bodyB→m_islandIndex;
130   m_localCenterA = m_bodyA→m_sweep.localCenter;
131   m_localCenterB = m_bodyB→m_sweep.localCenter;
132   m_invMassA = m_bodyA→m_invMass;
133   m_invMassB = m_bodyB→m_invMass;
134   m_invIA = m_bodyA→m_invI;
135   m_invIB = m_bodyB→m_invI;
136
137   b2Vec2 cA = data.positions[m_indexA].c;
138   float32 aA = data.positions[m_indexA].a;
139   b2Vec2 vA = data.velocities[m_indexA].v;
140   float32 wA = data.velocities[m_indexA].w;
141
142   b2Vec2 cB = data.positions[m_indexB].c;
143   float32 aB = data.positions[m_indexB].a;
144   b2Vec2 vB = data.velocities[m_indexB].v;
145   float32 wB = data.velocities[m_indexB].w;
146
147   b2Rot qA(aA), qB(aB);
148
149   // Compute the effective masses.
150   b2Vec2 rA = b2Mul(qA, m_localAnchorA − m_localCenterA);
151   b2Vec2 rB = b2Mul(qB, m_localAnchorB − m_localCenterB);
152   b2Vec2 d = (cB − cA) + rB − rA;
153
154   float32 mA = m_invMassA, mB = m_invMassB;
155   float32 iA = m_invIA, iB = m_invIB;
156
157   // Compute motor Jacobian and effective mass.
158   {
159     m_axis = b2Mul(qA, m_localXAxisA);
160     m_a1 = b2Cross(d + rA, m_axis);
161     m_a2 = b2Cross(rB, m_axis);
162
163     m_motorMass = mA + mB + iA * m_a1 * m_a1 + iB * m_a2 * m_a2;
164     if (m_motorMass > 0.0f)
165     {
166       m_motorMass = 1.0f / m_motorMass;
167     }
168   }
169
170   // Prismatic constraint.
171   {
172     m_perp = b2Mul(qA, m_localYAxisA);
173
174     m_s1 = b2Cross(d + rA, m_perp);
175     m_s2 = b2Cross(rB, m_perp);
176
177     float32 k11 = mA + mB + iA * m_s1 * m_s1 + iB * m_s2 * m_s2;
178     float32 k12 = iA * m_s1 + iB * m_s2;
179     float32 k13 = iA * m_s1 * m_a1 + iB * m_s2 * m_a2;
180     float32 k22 = iA + iB;
181     if (k22 ≡ 0.0f)
182     {
183       // For bodies with fixed rotation.
184       k22 = 1.0f;
185     }
186     float32 k23 = iA * m_a1 + iB * m_a2;
187     float32 k33 = mA + mB + iA * m_a1 * m_a1 + iB * m_a2 * m_a2;
188
189     m_K.ex.Set(k11, k12, k13);
190     m_K.ey.Set(k12, k22, k23);
```



```
191     m_K.ez.Set(k13, k23, k33);
192   }
193
194   // Compute motor and limit terms.
195   if (m_enableLimit)
196   {
197     float32 jointTranslation = b2Dot(m_axis, d);
198     if (b2Abs(m_upperTranslation − m_lowerTranslation) < 2.0f * b2_linearSlop)
199     {
200       m_limitState = e_equalLimits;
201     }
202     else if (jointTranslation ≤ m_lowerTranslation)
203     {
204       if (m_limitState ≠ e_atLowerLimit)
205       {
206         m_limitState = e_atLowerLimit;
207         m_impulse.z = 0.0f;
208       }
209     }
210     else if (jointTranslation ≥ m_upperTranslation)
211     {
212       if (m_limitState ≠ e_atUpperLimit)
213       {
214         m_limitState = e_atUpperLimit;
215         m_impulse.z = 0.0f;
216       }
217     }
218     else
219     {
220       m_limitState = e_inactiveLimit;
221       m_impulse.z = 0.0f;
222     }
223   }
224   else
225   {
226     m_limitState = e_inactiveLimit;
227     m_impulse.z = 0.0f;
228   }
229
230   if (m_enableMotor ≡ false)
231   {
232     m_motorImpulse = 0.0f;
233   }
234
235   if (data.step.warmStarting)
236   {
237     // Account for variable time step.
238     m_impulse *= data.step.dtRatio;
239     m_motorImpulse *= data.step.dtRatio;
240
241     b2Vec2 P = m_impulse.x * m_perp + (m_motorImpulse + m_impulse.z) * m_axis;
242     float32 LA = m_impulse.x * m_s1 + m_impulse.y + (m_motorImpulse + m_impulse.
    z) * m_a1;
243     float32 LB = m_impulse.x * m_s2 + m_impulse.y + (m_motorImpulse + m_impulse.
    z) * m_a2;
244
245     vA −= mA * P;
246     wA −= iA * LA;
247
248     vB += mB * P;
249     wB += iB * LB;
250   }
251   else
252   {
253     m_impulse.SetZero();
254     m_motorImpulse = 0.0f;
```

```
255    }
256
257    data.velocities[m_indexA].v = vA;
258    data.velocities[m_indexA].w = wA;
259    data.velocities[m_indexB].v = vB;
260    data.velocities[m_indexB].w = wB;
261  }
262
263  void b2PrismaticJoint::SolveVelocityConstraints(const b2SolverData& data)
264  {
265    b2Vec2 vA = data.velocities[m_indexA].v;
266    float32 wA = data.velocities[m_indexA].w;
267    b2Vec2 vB = data.velocities[m_indexB].v;
268    float32 wB = data.velocities[m_indexB].w;
269
270    float32 mA = m_invMassA, mB = m_invMassB;
271    float32 iA = m_invIA, iB = m_invIB;
272
273    // Solve linear motor constraint.
274    if (m_enableMotor ∧ m_limitState ≠ e_equalLimits)
275    {
276      float32 Cdot = b2Dot(m_axis, vB − vA) + m_a2 * wB − m_a1 * wA;
277      float32 impulse = m_motorMass * (m_motorSpeed − Cdot);
278      float32 oldImpulse = m_motorImpulse;
279      float32 maxImpulse = data.step.dt * m_maxMotorForce;
280      m_motorImpulse = b2Clamp(m_motorImpulse + impulse, −maxImpulse, maxImpulse);
281      impulse = m_motorImpulse − oldImpulse;
282
283      b2Vec2 P = impulse * m_axis;
284      float32 LA = impulse * m_a1;
285      float32 LB = impulse * m_a2;
286
287      vA −= mA * P;
288      wA −= iA * LA;
289
290      vB += mB * P;
291      wB += iB * LB;
292    }
293
294    b2Vec2 Cdot1;
295    Cdot1.x = b2Dot(m_perp, vB − vA) + m_s2 * wB − m_s1 * wA;
296    Cdot1.y = wB − wA;
297
298    if (m_enableLimit ∧ m_limitState ≠ e_inactiveLimit)
299    {
300      // Solve prismatic and limit constraint in block form.
301      float32 Cdot2;
302      Cdot2 = b2Dot(m_axis, vB − vA) + m_a2 * wB − m_a1 * wA;
303      b2Vec3 Cdot(Cdot1.x, Cdot1.y, Cdot2);
304
305      b2Vec3 f1 = m_impulse;
306      b2Vec3 df =  m_K.Solve33(−Cdot);
307      m_impulse += df;
308
309      if (m_limitState ≡ e_atLowerLimit)
310      {
311        m_impulse.z = b2Max(m_impulse.z, 0.0f);
312      }
313      else if (m_limitState ≡ e_atUpperLimit)
314      {
315        m_impulse.z = b2Min(m_impulse.z, 0.0f);
316      }
317
318      // f2(1:2) = invK(1:2,1:2) * (−Cdot(1:2) − K(1:2,3) * (f2(3) − f1(3))) + f1(
   1:2)
319      b2Vec2 b = −Cdot1 − (m_impulse.z − f1.z) * b2Vec2(m_K.ez.x, m_K.ez.y);
```



```
320      b2Vec2 f2r = m_K.Solve22(b) + b2Vec2(f1.x, f1.y);
321      m_impulse.x = f2r.x;
322      m_impulse.y = f2r.y;
323
324      df = m_impulse − f1;
325
326      b2Vec2 P = df.x * m_perp + df.z * m_axis;
327      float32 LA = df.x * m_s1 + df.y + df.z * m_a1;
328      float32 LB = df.x * m_s2 + df.y + df.z * m_a2;
329
330      vA −= mA * P;
331      wA −= iA * LA;
332
333      vB += mB * P;
334      wB += iB * LB;
335    }
336    else
337    {
338      // Limit is inactive, just solve the prismatic constraint in block form.
339      b2Vec2 df = m_K.Solve22(−Cdot1);
340      m_impulse.x += df.x;
341      m_impulse.y += df.y;
342
343      b2Vec2 P = df.x * m_perp;
344      float32 LA = df.x * m_s1 + df.y;
345      float32 LB = df.x * m_s2 + df.y;
346
347      vA −= mA * P;
348      wA −= iA * LA;
349
350      vB += mB * P;
351      wB += iB * LB;
352    }
353
354    data.velocities[m_indexA].v = vA;
355    data.velocities[m_indexA].w = wA;
356    data.velocities[m_indexB].v = vB;
357    data.velocities[m_indexB].w = wB;
358  }
359
360  // A velocity based solver computes reaction forces(impulses) using the velocity
      constraint solver.Under this context,
361  // the position solver is not there to resolve forces.It is only there to cope w
   ith integration error.
362  //
363  // Therefore, the pseudo impulses in the position solver do not have any physica
   l meaning.Thus it is okay if they suck.
364  //
365  // We could take the active state from the velocity solver.However, the joint mi
   ght push past the limit when the velocity
366  // solver indicates the limit is inactive.
367  bool b2PrismaticJoint::SolvePositionConstraints(const b2SolverData& data)
368  {
369    b2Vec2 cA = data.positions[m_indexA].c;
370    float32 aA = data.positions[m_indexA].a;
371    b2Vec2 cB = data.positions[m_indexB].c;
372    float32 aB = data.positions[m_indexB].a;
373
374    b2Rot qA(aA), qB(aB);
375
376    float32 mA = m_invMassA, mB = m_invMassB;
377    float32 iA = m_invIA, iB = m_invIB;
378
379    // Compute fresh Jacobians
380    b2Vec2 rA = b2Mul(qA, m_localAnchorA − m_localCenterA);
381    b2Vec2 rB = b2Mul(qB, m_localAnchorB − m_localCenterB);
```

```
382   b2Vec2 d = cB + rB − cA − rA;
383
384   b2Vec2 axis = b2Mul(qA, m_localXAxisA);
385   float32 a1 = b2Cross(d + rA, axis);
386   float32 a2 = b2Cross(rB, axis);
387   b2Vec2 perp = b2Mul(qA, m_localYAxisA);
388
389   float32 s1 = b2Cross(d + rA, perp);
390   float32 s2 = b2Cross(rB, perp);
391
392   b2Vec3 impulse;
393   b2Vec2 C1;
394   C1.x = b2Dot(perp, d);
395   C1.y = aB − aA − m_referenceAngle;
396
397   float32 linearError = b2Abs(C1.x);
398   float32 angularError = b2Abs(C1.y);
399
400   bool active = false;
401   float32 C2 = 0.0f;
402   if (m_enableLimit)
403   {
404     float32 translation = b2Dot(axis, d);
405     if (b2Abs(m_upperTranslation − m_lowerTranslation) < 2.0f * b2_linearSlop)
406     {
407       // Prevent large angular corrections
408       C2 = b2Clamp(translation, −b2_maxLinearCorrection, b2_maxLinearCorrection)
    ;
409       linearError = b2Max(linearError, b2Abs(translation));
410       active = true;
411     }
412     else if (translation ≤ m_lowerTranslation)
413     {
414       // Prevent large linear corrections and allow some slop.
415       C2 = b2Clamp(translation − m_lowerTranslation + b2_linearSlop, −b2_maxLine
    arCorrection, 0.0f);
416       linearError = b2Max(linearError, m_lowerTranslation − translation);
417       active = true;
418     }
419     else if (translation ≥ m_upperTranslation)
420     {
421       // Prevent large linear corrections and allow some slop.
422       C2 = b2Clamp(translation − m_upperTranslation − b2_linearSlop, 0.0f, b2_ma
    xLinearCorrection);
423       linearError = b2Max(linearError, translation − m_upperTranslation);
424       active = true;
425     }
426   }
427
428   if (active)
429   {
430     float32 k11 = mA + mB + iA * s1 * s1 + iB * s2 * s2;
431     float32 k12 = iA * s1 + iB * s2;
432     float32 k13 = iA * s1 * a1 + iB * s2 * a2;
433     float32 k22 = iA + iB;
434     if (k22 ≡ 0.0f)
435     {
436       // For fixed rotation
437       k22 = 1.0f;
438     }
439     float32 k23 = iA * a1 + iB * a2;
440     float32 k33 = mA + mB + iA * a1 * a1 + iB * a2 * a2;
441
442     b2Mat33 K;
443     K.ex.Set(k11, k12, k13);
444     K.ey.Set(k12, k22, k23);
```



```
445     K.ez.Set(k13, k23, k33);
446
447     b2Vec3 C;
448     C.x = C1.x;
449     C.y = C1.y;
450     C.z = C2;
451
452     impulse = K.Solve33(−C);
453   }
454   else
455   {
456     float32 k11 = mA + mB + iA * s1 * s1 + iB * s2 * s2;
457     float32 k12 = iA * s1 + iB * s2;
458     float32 k22 = iA + iB;
459     if (k22 ≡ 0.0f)
460     {
461       k22 = 1.0f;
462     }
463
464     b2Mat22 K;
465     K.ex.Set(k11, k12);
466     K.ey.Set(k12, k22);
467
468     b2Vec2 impulse1 = K.Solve(−C1);
469     impulse.x = impulse1.x;
470     impulse.y = impulse1.y;
471     impulse.z = 0.0f;
472   }
473
474   b2Vec2 P = impulse.x * perp + impulse.z * axis;
475   float32 LA = impulse.x * s1 + impulse.y + impulse.z * a1;
476   float32 LB = impulse.x * s2 + impulse.y + impulse.z * a2;
477
478   cA −= mA * P;
479   aA −= iA * LA;
480   cB += mB * P;
481   aB += iB * LB;
482
483   data.positions[m_indexA].c = cA;
484   data.positions[m_indexA].a = aA;
485   data.positions[m_indexB].c = cB;
486   data.positions[m_indexB].a = aB;
487
488   return linearError ≤ b2_linearSlop ∧ angularError ≤ b2_angularSlop;
489 }
490
491 b2Vec2 b2PrismaticJoint::GetAnchorA() const
492 {
493   return m_bodyA→GetWorldPoint(m_localAnchorA);
494 }
495
496 b2Vec2 b2PrismaticJoint::GetAnchorB() const
497 {
498   return m_bodyB→GetWorldPoint(m_localAnchorB);
499 }
500
501 b2Vec2 b2PrismaticJoint::GetReactionForce(float32 inv_dt) const
502 {
503   return inv_dt * (m_impulse.x * m_perp + (m_motorImpulse + m_impulse.z) * m_axi
    s);
504 }
505
506 float32 b2PrismaticJoint::GetReactionTorque(float32 inv_dt) const
507 {
508   return inv_dt * m_impulse.y;
509 }
```

```
510
511 float32 b2PrismaticJoint::GetJointTranslation() const
512 {
513   b2Vec2 pA = m_bodyA→GetWorldPoint(m_localAnchorA);
514   b2Vec2 pB = m_bodyB→GetWorldPoint(m_localAnchorB);
515   b2Vec2 d = pB − pA;
516   b2Vec2 axis = m_bodyA→GetWorldVector(m_localXAxisA);
517
518   float32 translation = b2Dot(d, axis);
519   return translation;
520 }
521
522 float32 b2PrismaticJoint::GetJointSpeed() const
523 {
524   b2Body* bA = m_bodyA;
525   b2Body* bB = m_bodyB;
526
527   b2Vec2 rA = b2Mul(bA→m_xf.q, m_localAnchorA − bA→m_sweep.localCenter);
528   b2Vec2 rB = b2Mul(bB→m_xf.q, m_localAnchorB − bB→m_sweep.localCenter);
529   b2Vec2 p1 = bA→m_sweep.c + rA;
530   b2Vec2 p2 = bB→m_sweep.c + rB;
531   b2Vec2 d = p2 − p1;
532   b2Vec2 axis = b2Mul(bA→m_xf.q, m_localXAxisA);
533
534   b2Vec2 vA = bA→m_linearVelocity;
535   b2Vec2 vB = bB→m_linearVelocity;
536   float32 wA = bA→m_angularVelocity;
537   float32 wB = bB→m_angularVelocity;
538
539   float32 speed = b2Dot(d, b2Cross(wA, axis)) + b2Dot(axis, vB + b2Cross(wB, rB)
     − vA − b2Cross(wA, rA));
540   return speed;
541 }
542
543 bool b2PrismaticJoint::IsLimitEnabled() const
544 {
545   return m_enableLimit;
546 }
547
548 void b2PrismaticJoint::EnableLimit(bool flag)
549 {
550   if (flag ≠ m_enableLimit)
551   {
552     m_bodyA→SetAwake(true);
553     m_bodyB→SetAwake(true);
554     m_enableLimit = flag;
555     m_impulse.z = 0.0f;
556   }
557 }
558
559 float32 b2PrismaticJoint::GetLowerLimit() const
560 {
561   return m_lowerTranslation;
562 }
563
564 float32 b2PrismaticJoint::GetUpperLimit() const
565 {
566   return m_upperTranslation;
567 }
568
569 void b2PrismaticJoint::SetLimits(float32 lower, float32 upper)
570 {
571   b2Assert(lower ≤ upper);
572   if (lower ≠ m_lowerTranslation ∨ upper ≠ m_upperTranslation)
573   {
574     m_bodyA→SetAwake(true);
```



```
575     m_bodyB→SetAwake(true);
576     m_lowerTranslation = lower;
577     m_upperTranslation = upper;
578     m_impulse.z = 0.0f;
579   }
580 }
581
582 bool b2PrismaticJoint::IsMotorEnabled() const
583 {
584   return m_enableMotor;
585 }
586
587 void b2PrismaticJoint::EnableMotor(bool flag)
588 {
589   if (flag ≠ m_enableMotor)
590   {
591     m_bodyA→SetAwake(true);
592     m_bodyB→SetAwake(true);
593     m_enableMotor = flag;
594   }
595 }
596
597 void b2PrismaticJoint::SetMotorSpeed(float32 speed)
598 {
599   if (speed ≠ m_motorSpeed)
600   {
601     m_bodyA→SetAwake(true);
602     m_bodyB→SetAwake(true);
603     m_motorSpeed = speed;
604   }
605 }
606
607 void b2PrismaticJoint::SetMaxMotorForce(float32 force)
608 {
609   if (force ≠ m_maxMotorForce)
610   {
611     m_bodyA→SetAwake(true);
612     m_bodyB→SetAwake(true);
613     m_maxMotorForce = force;
614   }
615 }
616
617 float32 b2PrismaticJoint::GetMotorForce(float32 inv_dt) const
618 {
619   return inv_dt * m_motorImpulse;
620 }
621
622 void b2PrismaticJoint::Dump()
623 {
624   int32 indexA = m_bodyA→m_islandIndex;
625   int32 indexB = m_bodyB→m_islandIndex;
626
627   b2Log(" b2PrismaticJointDef jd;\n");
628   b2Log(" jd.bodyA = bodies[%d];\n", indexA);
629   b2Log(" jd.bodyB = bodies[%d];\n", indexB);
630   b2Log(" jd.collideConnected = bool(%d);\n", m_collideConnected);
631   b2Log(" jd.localAnchorA.Set(%.15lef, %.15lef);\n", m_localAnchorA.x, m_localAnchorA.y);
632   b2Log(" jd.localAnchorB.Set(%.15lef, %.15lef);\n", m_localAnchorB.x, m_localAnchorB.y);
633   b2Log(" jd.localAxisA.Set(%.15lef, %.15lef);\n", m_localXAxisA.x, m_localXAxisA.y);
634   b2Log(" jd.referenceAngle = %.15lef;\n", m_referenceAngle);
635   b2Log(" jd.enableLimit = bool(%d);\n", m_enableLimit);
636   b2Log(" jd.lowerTranslation = %.15lef;\n", m_lowerTranslation);
637   b2Log(" jd.upperTranslation = %.15lef;\n", m_upperTranslation);
638   b2Log(" jd.enableMotor = bool(%d);\n", m_enableMotor);
639   b2Log(" jd.motorSpeed = %.15lef;\n", m_motorSpeed);
640   b2Log(" jd.maxMotorForce = %.15lef;\n", m_maxMotorForce);
```

```
641     b2Log(" joints[%d] = m_world->CreateJoint(&jd);\n", m_index);
642 }
```



```
1  /*
2  * Copyright (c) 2006-2007 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as-is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #include "Box2D/Dynamics/Joints/b2MouseJoint.h"
20 #include "Box2D/Dynamics/b2Body.h"
21 #include "Box2D/Dynamics/b2TimeStep.h"
22
23 // p = attached point, m = mouse point
24 // C = p - m
25 // Cdot = v
26 //      = v + cross(w, r)
27 // J = [I r_skew]
28 // Identity used:
29 // w k % (rx i + ry j) = w * (-ry i + rx j)
30
31 b2MouseJoint::b2MouseJoint(const b2MouseJointDef* def)
32 : b2Joint(def)
33 {
34   b2Assert(def->target.IsValid());
35   b2Assert(b2IsValid(def->maxForce) ∧ def->maxForce ≥ 0.0f);
36   b2Assert(b2IsValid(def->frequencyHz) ∧ def->frequencyHz ≥ 0.0f);
37   b2Assert(b2IsValid(def->dampingRatio) ∧ def->dampingRatio ≥ 0.0f);
38
39   m_targetA = def->target;
40   m_localAnchorB = b2MulT(m_bodyB->GetTransform(), m_targetA);
41
42   m_maxForce = def->maxForce;
43   m_impulse.SetZero();
44
45   m_frequencyHz = def->frequencyHz;
46   m_dampingRatio = def->dampingRatio;
47
48   m_beta = 0.0f;
49   m_gamma = 0.0f;
50 }
51
52 void b2MouseJoint::SetTarget(const b2Vec2& target)
53 {
54   if (target ≠ m_targetA)
55   {
56     m_bodyB->SetAwake(true);
57     m_targetA = target;
58   }
59 }
60
61 const b2Vec2& b2MouseJoint::GetTarget() const
62 {
63   return m_targetA;
64 }
65
66 void b2MouseJoint::SetMaxForce(float32 force)
```

```
67  {
68    m_maxForce = force;
69  }
70
71  float32 b2MouseJoint::GetMaxForce() const
72  {
73    return m_maxForce;
74  }
75
76  void b2MouseJoint::SetFrequency(float32 hz)
77  {
78    m_frequencyHz = hz;
79  }
80
81  float32 b2MouseJoint::GetFrequency() const
82  {
83    return m_frequencyHz;
84  }
85
86  void b2MouseJoint::SetDampingRatio(float32 ratio)
87  {
88    m_dampingRatio = ratio;
89  }
90
91  float32 b2MouseJoint::GetDampingRatio() const
92  {
93    return m_dampingRatio;
94  }
95
96  void b2MouseJoint::InitVelocityConstraints(const b2SolverData& data)
97  {
98    m_indexB = m_bodyB→m_islandIndex;
99    m_localCenterB = m_bodyB→m_sweep.localCenter;
100   m_invMassB = m_bodyB→m_invMass;
101   m_invIB = m_bodyB→m_invI;
102
103   b2Vec2 cB = data.positions[m_indexB].c;
104   float32 aB = data.positions[m_indexB].a;
105   b2Vec2 vB = data.velocities[m_indexB].v;
106   float32 wB = data.velocities[m_indexB].w;
107
108   b2Rot qB(aB);
109
110   float32 mass = m_bodyB→GetMass();
111
112   // Frequency
113   float32 omega = 2.0f * b2_pi * m_frequencyHz;
114
115   // Damping coefficient
116   float32 d = 2.0f * mass * m_dampingRatio * omega;
117
118   // Spring stiffness
119   float32 k = mass * (omega * omega);
120
121   // magic formulas
122   // gamma has units of inverse mass.
123   // beta has units of inverse time.
124   float32 h = data.step.dt;
125   b2Assert(d + h * k > b2_epsilon);
126   m_gamma = h * (d + h * k);
127   if (m_gamma ≠ 0.0f)
128   {
129     m_gamma = 1.0f / m_gamma;
130   }
131   m_beta = h * k * m_gamma;
132
```



```
133   // Compute the effective mass matrix.
134   m_rB = b2Mul(qB, m_localAnchorB − m_localCenterB);
135
136   // K    = [(1/m1 + 1/m2) * eye(2) − skew(r1) * invI1 * skew(r1) − skew(r2) * i
    nvI2 * skew(r2)]
137   //      = [1/m1+1/m2     0    ] + invI1 * [r1.y*r1.y −r1.x*r1.y] + invI2 * [r1
    .y*r1.y −r1.x*r1.y]
138   //        [    0     1/m1+1/m2]           [−r1.x*r1.y r1.x*r1.x]           [−r
    1.x*r1.y r1.x*r1.x]
139   b2Mat22 K;
140   K.ex.x = m_invMassB + m_invIB * m_rB.y * m_rB.y + m_gamma;
141   K.ex.y = −m_invIB * m_rB.x * m_rB.y;
142   K.ey.x = K.ex.y;
143   K.ey.y = m_invMassB + m_invIB * m_rB.x * m_rB.x + m_gamma;
144
145   m_mass = K.GetInverse();
146
147   m_C = cB + m_rB − m_targetA;
148   m_C *= m_beta;
149
150   // Cheat with some damping
151   wB *= 0.98f;
152
153   if (data.step.warmStarting)
154   {
155     m_impulse *= data.step.dtRatio;
156     vB += m_invMassB * m_impulse;
157     wB += m_invIB * b2Cross(m_rB, m_impulse);
158   }
159   else
160   {
161     m_impulse.SetZero();
162   }
163
164   data.velocities[m_indexB].v = vB;
165   data.velocities[m_indexB].w = wB;
166 }
167
168 void b2MouseJoint::SolveVelocityConstraints(const b2SolverData& data)
169 {
170   b2Vec2 vB = data.velocities[m_indexB].v;
171   float32 wB = data.velocities[m_indexB].w;
172
173   // Cdot = v + cross(w, r)
174   b2Vec2 Cdot = vB + b2Cross(wB, m_rB);
175   b2Vec2 impulse = b2Mul(m_mass, −(Cdot + m_C + m_gamma * m_impulse));
176
177   b2Vec2 oldImpulse = m_impulse;
178   m_impulse += impulse;
179   float32 maxImpulse = data.step.dt * m_maxForce;
180   if (m_impulse.LengthSquared() > maxImpulse * maxImpulse)
181   {
182     m_impulse *= maxImpulse / m_impulse.Length();
183   }
184   impulse = m_impulse − oldImpulse;
185
186   vB += m_invMassB * impulse;
187   wB += m_invIB * b2Cross(m_rB, impulse);
188
189   data.velocities[m_indexB].v = vB;
190   data.velocities[m_indexB].w = wB;
191 }
192
193 bool b2MouseJoint::SolvePositionConstraints(const b2SolverData& data)
194 {
195   B2_NOT_USED(data);
```

```
196   return true;
197 }
198
199 b2Vec2 b2MouseJoint::GetAnchorA() const
200 {
201   return m_targetA;
202 }
203
204 b2Vec2 b2MouseJoint::GetAnchorB() const
205 {
206   return m_bodyB→GetWorldPoint(m_localAnchorB);
207 }
208
209 b2Vec2 b2MouseJoint::GetReactionForce(float32 inv_dt) const
210 {
211   return inv_dt * m_impulse;
212 }
213
214 float32 b2MouseJoint::GetReactionTorque(float32 inv_dt) const
215 {
216   return inv_dt * 0.0f;
217 }
218
219 void b2MouseJoint::ShiftOrigin(const b2Vec2& newOrigin)
220 {
221   m_targetA −= newOrigin;
222 }
```



```
1  /*
2  * Copyright (c) 2006−2012 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as−is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #include "Box2D/Dynamics/Joints/b2MotorJoint.h"
20 #include "Box2D/Dynamics/b2Body.h"
21 #include "Box2D/Dynamics/b2TimeStep.h"
22
23 // Point−to−point constraint
24 // Cdot = v2 − v1
25 //      = v2 + cross(w2, r2) − v1 − cross(w1, r1)
26 // J = [−I −r1_skew I r2_skew ]
27 // Identity used:
28 // w k % (rx i + ry j) = w * (−ry i + rx j)
29 //
30 // r1 = offset − c1
31 // r2 = −c2
32
33 // Angle constraint
34 // Cdot = w2 − w1
35 // J = [0 0 −1 0 0 1]
36 // K = invI1 + invI2
37
38 void b2MotorJointDef::Initialize(b2Body* bA, b2Body* bB)
39 {
40   bodyA = bA;
41   bodyB = bB;
42   b2Vec2 xB = bodyB→GetPosition();
43   linearOffset = bodyA→GetLocalPoint(xB);
44
45   float32 angleA = bodyA→GetAngle();
46   float32 angleB = bodyB→GetAngle();
47   angularOffset = angleB − angleA;
48 }
49
50 b2MotorJoint::b2MotorJoint(const b2MotorJointDef* def)
51 : b2Joint(def)
52 {
53   m_linearOffset = def→linearOffset;
54   m_angularOffset = def→angularOffset;
55
56   m_linearImpulse.SetZero();
57   m_angularImpulse = 0.0f;
58
59   m_maxForce = def→maxForce;
60   m_maxTorque = def→maxTorque;
61   m_correctionFactor = def→correctionFactor;
62 }
63
64 void b2MotorJoint::InitVelocityConstraints(const b2SolverData& data)
65 {
66   m_indexA = m_bodyA→m_islandIndex;
```

```
67    m_indexB = m_bodyB→m_islandIndex;
68    m_localCenterA = m_bodyA→m_sweep.localCenter;
69    m_localCenterB = m_bodyB→m_sweep.localCenter;
70    m_invMassA = m_bodyA→m_invMass;
71    m_invMassB = m_bodyB→m_invMass;
72    m_invIA = m_bodyA→m_invI;
73    m_invIB = m_bodyB→m_invI;
74
75    b2Vec2 cA = data.positions[m_indexA].c;
76    float32 aA = data.positions[m_indexA].a;
77    b2Vec2 vA = data.velocities[m_indexA].v;
78    float32 wA = data.velocities[m_indexA].w;
79
80    b2Vec2 cB = data.positions[m_indexB].c;
81    float32 aB = data.positions[m_indexB].a;
82    b2Vec2 vB = data.velocities[m_indexB].v;
83    float32 wB = data.velocities[m_indexB].w;
84
85    b2Rot qA(aA), qB(aB);
86
87    // Compute the effective mass matrix.
88    m_rA = b2Mul(qA, m_linearOffset − m_localCenterA);
89    m_rB = b2Mul(qB, −m_localCenterB);
90
91    // J = [−I −r1_skew I r2_skew]
92    // r_skew = [−ry; rx]
93
94    // Matlab
95    // K = [ mA+r1y^2*iA+mB+r2y^2*iB,  −r1y*iA*r1x−r2y*iB*r2x,          −r1y*iA−r2
    y*iB]
96    //     [  −r1y*iA*r1x−r2y*iB*r2x, mA+r1x^2*iA+mB+r2x^2*iB,           r1x*iA+r2
    x*iB]
97    //     [          −r1y*iA−r2y*iB,           r1x*iA+r2x*iB,                   i
    A+iB]
98
99
100
101   float32 mA = m_invMassA, mB = m_invMassB;
102   float32 iA = m_invIA, iB = m_invIB;
103
104   // Upper 2 by 2 of K for point to point
105   b2Mat22 K;
106   K.ex.x = mA + mB + iA * m_rA.y * m_rA.y + iB * m_rB.y * m_rB.y;
107   K.ex.y = −iA * m_rA.x * m_rA.y − iB * m_rB.x * m_rB.y;
108   K.ey.x = K.ex.y;
109   K.ey.y = mA + mB + iA * m_rA.x * m_rA.x + iB * m_rB.x * m_rB.x;
110
111   m_linearMass = K.GetInverse();
112
113   m_angularMass = iA + iB;
114   if (m_angularMass > 0.0f)
115   {
116     m_angularMass = 1.0f / m_angularMass;
117   }
118
119   m_linearError = cB + m_rB − cA − m_rA;
120   m_angularError = aB − aA − m_angularOffset;
121
122   if (data.step.warmStarting)
123   {
124     // Scale impulses to support a variable time step.
125     m_linearImpulse *= data.step.dtRatio;
126     m_angularImpulse *= data.step.dtRatio;
127
128     b2Vec2 P(m_linearImpulse.x, m_linearImpulse.y);
129     vA −= mA * P;
```



```
130     wA −= iA * (b2Cross(m_rA, P) + m_angularImpulse);
131     vB += mB * P;
132     wB += iB * (b2Cross(m_rB, P) + m_angularImpulse);
133   }
134   else
135   {
136     m_linearImpulse.SetZero();
137     m_angularImpulse = 0.0f;
138   }
139
140   data.velocities[m_indexA].v = vA;
141   data.velocities[m_indexA].w = wA;
142   data.velocities[m_indexB].v = vB;
143   data.velocities[m_indexB].w = wB;
144 }
145
146 void b2MotorJoint::SolveVelocityConstraints(const b2SolverData& data)
147 {
148   b2Vec2 vA = data.velocities[m_indexA].v;
149   float32 wA = data.velocities[m_indexA].w;
150   b2Vec2 vB = data.velocities[m_indexB].v;
151   float32 wB = data.velocities[m_indexB].w;
152
153   float32 mA = m_invMassA, mB = m_invMassB;
154   float32 iA = m_invIA, iB = m_invIB;
155
156   float32 h = data.step.dt;
157   float32 inv_h = data.step.inv_dt;
158
159   // Solve angular friction
160   {
161     float32 Cdot = wB − wA + inv_h * m_correctionFactor * m_angularError;
162     float32 impulse = −m_angularMass * Cdot;
163
164     float32 oldImpulse = m_angularImpulse;
165     float32 maxImpulse = h * m_maxTorque;
166     m_angularImpulse = b2Clamp(m_angularImpulse + impulse, −maxImpulse, maxImpul
    se);
167     impulse = m_angularImpulse − oldImpulse;
168
169     wA −= iA * impulse;
170     wB += iB * impulse;
171   }
172
173   // Solve linear friction
174   {
175     b2Vec2 Cdot = vB + b2Cross(wB, m_rB) − vA − b2Cross(wA, m_rA) + inv_h * m_co
    rrectionFactor * m_linearError;
176
177     b2Vec2 impulse = −b2Mul(m_linearMass, Cdot);
178     b2Vec2 oldImpulse = m_linearImpulse;
179     m_linearImpulse += impulse;
180
181     float32 maxImpulse = h * m_maxForce;
182
183     if (m_linearImpulse.LengthSquared() > maxImpulse * maxImpulse)
184     {
185       m_linearImpulse.Normalize();
186       m_linearImpulse *= maxImpulse;
187     }
188
189     impulse = m_linearImpulse − oldImpulse;
190
191     vA −= mA * impulse;
192     wA −= iA * b2Cross(m_rA, impulse);
193
```

```
194     vB += mB * impulse;
195     wB += iB * b2Cross(m_rB, impulse);
196   }
197
198   data.velocities[m_indexA].v = vA;
199   data.velocities[m_indexA].w = wA;
200   data.velocities[m_indexB].v = vB;
201   data.velocities[m_indexB].w = wB;
202 }
203
204 bool b2MotorJoint::SolvePositionConstraints(const b2SolverData& data)
205 {
206   B2_NOT_USED(data);
207
208   return true;
209 }
210
211 b2Vec2 b2MotorJoint::GetAnchorA() const
212 {
213   return m_bodyA→GetPosition();
214 }
215
216 b2Vec2 b2MotorJoint::GetAnchorB() const
217 {
218   return m_bodyB→GetPosition();
219 }
220
221 b2Vec2 b2MotorJoint::GetReactionForce(float32 inv_dt) const
222 {
223   return inv_dt * m_linearImpulse;
224 }
225
226 float32 b2MotorJoint::GetReactionTorque(float32 inv_dt) const
227 {
228   return inv_dt * m_angularImpulse;
229 }
230
231 void b2MotorJoint::SetMaxForce(float32 force)
232 {
233   b2Assert(b2IsValid(force) ∧ force ≥ 0.0f);
234   m_maxForce = force;
235 }
236
237 float32 b2MotorJoint::GetMaxForce() const
238 {
239   return m_maxForce;
240 }
241
242 void b2MotorJoint::SetMaxTorque(float32 torque)
243 {
244   b2Assert(b2IsValid(torque) ∧ torque ≥ 0.0f);
245   m_maxTorque = torque;
246 }
247
248 float32 b2MotorJoint::GetMaxTorque() const
249 {
250   return m_maxTorque;
251 }
252
253 void b2MotorJoint::SetCorrectionFactor(float32 factor)
254 {
255   b2Assert(b2IsValid(factor) ∧ 0.0f ≤ factor ∧ factor ≤ 1.0f);
256   m_correctionFactor = factor;
257 }
258
259 float32 b2MotorJoint::GetCorrectionFactor() const
```



```
260 {
261   return m_correctionFactor;
262 }
263
264 void b2MotorJoint::SetLinearOffset(const b2Vec2& linearOffset)
265 {
266   if (linearOffset.x ≠ m_linearOffset.x ∨ linearOffset.y ≠ m_linearOffset.y)
267   {
268     m_bodyA→SetAwake(true);
269     m_bodyB→SetAwake(true);
270     m_linearOffset = linearOffset;
271   }
272 }
273
274 const b2Vec2& b2MotorJoint::GetLinearOffset() const
275 {
276   return m_linearOffset;
277 }
278
279 void b2MotorJoint::SetAngularOffset(float32 angularOffset)
280 {
281   if (angularOffset ≠ m_angularOffset)
282   {
283     m_bodyA→SetAwake(true);
284     m_bodyB→SetAwake(true);
285     m_angularOffset = angularOffset;
286   }
287 }
288
289 float32 b2MotorJoint::GetAngularOffset() const
290 {
291   return m_angularOffset;
292 }
293
294 void b2MotorJoint::Dump()
295 {
296   int32 indexA = m_bodyA→m_islandIndex;
297   int32 indexB = m_bodyB→m_islandIndex;
298
299   b2Log(" b2MotorJointDef jd;\n");
300   b2Log(" jd.bodyA = bodies[%d];\n", indexA);
301   b2Log(" jd.bodyB = bodies[%d];\n", indexB);
302   b2Log(" jd.collideConnected = bool(%d);\n", m_collideConnected);
303   b2Log(" jd.linearOffset.Set(%.15lef, %.15lef);\n", m_linearOffset.x, m_linearOffset.y);
304   b2Log(" jd.angularOffset = %.15lef;\n", m_angularOffset);
305   b2Log(" jd.maxForce = %.15lef;\n", m_maxForce);
306   b2Log(" jd.maxTorque = %.15lef;\n", m_maxTorque);
307   b2Log(" jd.correctionFactor = %.15lef;\n", m_correctionFactor);
308   b2Log(" joints[%d] = m_world->CreateJoint(&jd);\n", m_index);
309 }
```

```
1  /*
2  * Copyright (c) 2006-2007 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as-is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #include "Box2D/Dynamics/Joints/b2Joint.h"
20 #include "Box2D/Dynamics/Joints/b2DistanceJoint.h"
21 #include "Box2D/Dynamics/Joints/b2WheelJoint.h"
22 #include "Box2D/Dynamics/Joints/b2MouseJoint.h"
23 #include "Box2D/Dynamics/Joints/b2RevoluteJoint.h"
24 #include "Box2D/Dynamics/Joints/b2PrismaticJoint.h"
25 #include "Box2D/Dynamics/Joints/b2PulleyJoint.h"
26 #include "Box2D/Dynamics/Joints/b2GearJoint.h"
27 #include "Box2D/Dynamics/Joints/b2WeldJoint.h"
28 #include "Box2D/Dynamics/Joints/b2FrictionJoint.h"
29 #include "Box2D/Dynamics/Joints/b2RopeJoint.h"
30 #include "Box2D/Dynamics/Joints/b2MotorJoint.h"
31 #include "Box2D/Dynamics/b2Body.h"
32 #include "Box2D/Dynamics/b2World.h"
33 #include "Box2D/Common/b2BlockAllocator.h"
34
35 #include <new>
36
37 b2Joint* b2Joint::Create(const b2JointDef* def, b2BlockAllocator* allocator)
38 {
39   b2Joint* joint = nullptr;
40
41   switch (def->type)
42   {
43   case e_distanceJoint:
44     {
45       void* mem = allocator->Allocate(sizeof(b2DistanceJoint));
46       joint = new (mem) b2DistanceJoint(static_cast<const b2DistanceJointDef*>(d
   ef));
47     }
48     break;
49
50   case e_mouseJoint:
51     {
52       void* mem = allocator->Allocate(sizeof(b2MouseJoint));
53       joint = new (mem) b2MouseJoint(static_cast<const b2MouseJointDef*>(def));
54     }
55     break;
56
57   case e_prismaticJoint:
58     {
59       void* mem = allocator->Allocate(sizeof(b2PrismaticJoint));
60       joint = new (mem) b2PrismaticJoint(static_cast<const b2PrismaticJointDef*>
   (def));
61     }
62     break;
63
64   case e_revoluteJoint:
```



```
65     {
66       void* mem = allocator->Allocate(sizeof(b2RevoluteJoint));
67       joint = new (mem) b2RevoluteJoint(static_cast<const b2RevoluteJointDef*>(d
   ef));
68     }
69     break;
70
71   case e_pulleyJoint:
72     {
73       void* mem = allocator->Allocate(sizeof(b2PulleyJoint));
74       joint = new (mem) b2PulleyJoint(static_cast<const b2PulleyJointDef*>(def))
   ;
75     }
76     break;
77
78   case e_gearJoint:
79     {
80       void* mem = allocator->Allocate(sizeof(b2GearJoint));
81       joint = new (mem) b2GearJoint(static_cast<const b2GearJointDef*>(def));
82     }
83     break;
84
85   case e_wheelJoint:
86     {
87       void* mem = allocator->Allocate(sizeof(b2WheelJoint));
88       joint = new (mem) b2WheelJoint(static_cast<const b2WheelJointDef*>(def));
89     }
90     break;
91
92   case e_weldJoint:
93     {
94       void* mem = allocator->Allocate(sizeof(b2WeldJoint));
95       joint = new (mem) b2WeldJoint(static_cast<const b2WeldJointDef*>(def));
96     }
97     break;
98
99   case e_frictionJoint:
100    {
101      void* mem = allocator->Allocate(sizeof(b2FrictionJoint));
102      joint = new (mem) b2FrictionJoint(static_cast<const b2FrictionJointDef*>(d
   ef));
103    }
104    break;
105
106  case e_ropeJoint:
107    {
108      void* mem = allocator->Allocate(sizeof(b2RopeJoint));
109      joint = new (mem) b2RopeJoint(static_cast<const b2RopeJointDef*>(def));
110    }
111    break;
112
113  case e_motorJoint:
114    {
115      void* mem = allocator->Allocate(sizeof(b2MotorJoint));
116      joint = new (mem) b2MotorJoint(static_cast<const b2MotorJointDef*>(def));
117    }
118    break;
119
120  default:
121    b2Assert(false);
122    break;
123  }
124
125  return joint;
126 }
127
```

```
128 void b2Joint::Destroy(b2Joint* joint, b2BlockAllocator* allocator)
129 {
130   joint→~b2Joint();
131   switch (joint→m_type)
132   {
133   case e_distanceJoint:
134     allocator→Free(joint, sizeof(b2DistanceJoint));
135     break;
136
137   case e_mouseJoint:
138     allocator→Free(joint, sizeof(b2MouseJoint));
139     break;
140
141   case e_prismaticJoint:
142     allocator→Free(joint, sizeof(b2PrismaticJoint));
143     break;
144
145   case e_revoluteJoint:
146     allocator→Free(joint, sizeof(b2RevoluteJoint));
147     break;
148
149   case e_pulleyJoint:
150     allocator→Free(joint, sizeof(b2PulleyJoint));
151     break;
152
153   case e_gearJoint:
154     allocator→Free(joint, sizeof(b2GearJoint));
155     break;
156
157   case e_wheelJoint:
158     allocator→Free(joint, sizeof(b2WheelJoint));
159     break;
160
161   case e_weldJoint:
162     allocator→Free(joint, sizeof(b2WeldJoint));
163     break;
164
165   case e_frictionJoint:
166     allocator→Free(joint, sizeof(b2FrictionJoint));
167     break;
168
169   case e_ropeJoint:
170     allocator→Free(joint, sizeof(b2RopeJoint));
171     break;
172
173   case e_motorJoint:
174     allocator→Free(joint, sizeof(b2MotorJoint));
175     break;
176
177   default:
178     b2Assert(false);
179     break;
180   }
181 }
182
183 b2Joint::b2Joint(const b2JointDef* def)
184 {
185   b2Assert(def→bodyA ≠ def→bodyB);
186
187   m_type = def→type;
188   m_prev = nullptr;
189   m_next = nullptr;
190   m_bodyA = def→bodyA;
191   m_bodyB = def→bodyB;
192   m_index = 0;
193   m_collideConnected = def→collideConnected;
```



```
194   m_islandFlag = false;
195   m_userData = def→userData;
196
197   m_edgeA.joint = nullptr;
198   m_edgeA.other = nullptr;
199   m_edgeA.prev = nullptr;
200   m_edgeA.next = nullptr;
201
202   m_edgeB.joint = nullptr;
203   m_edgeB.other = nullptr;
204   m_edgeB.prev = nullptr;
205   m_edgeB.next = nullptr;
206 }
207
208 bool b2Joint::IsActive() const
209 {
210   return m_bodyA→IsActive() ∧ m_bodyB→IsActive();
211 }
```

```
/*
* Copyright (c) 2007-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Joints/b2GearJoint.h"
#include "Box2D/Dynamics/Joints/b2RevoluteJoint.h"
#include "Box2D/Dynamics/Joints/b2PrismaticJoint.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2TimeStep.h"

// Gear Joint:
// C0 = (coordinate1 + ratio * coordinate2)_initial
// C = (coordinate1 + ratio * coordinate2) - C0 = 0
// J = [J1 ratio * J2]
// K = J * invM * JT
//   = J1 * invM1 * J1T + ratio * ratio * J2 * invM2 * J2T
//
// Revolute:
// coordinate = rotation
// Cdot = angularVelocity
// J = [0 0 1]
// K = J * invM * JT = invI
//
// Prismatic:
// coordinate = dot(p - pg, ug)
// Cdot = dot(v + cross(w, r), ug)
// J = [ug cross(r, ug)]
// K = J * invM * JT = invMass + invI * cross(r, ug)^2

b2GearJoint::b2GearJoint(const b2GearJointDef* def)
: b2Joint(def)
{
  m_joint1 = def->joint1;
  m_joint2 = def->joint2;

  m_typeA = m_joint1->GetType();
  m_typeB = m_joint2->GetType();

  b2Assert(m_typeA == e_revoluteJoint || m_typeA == e_prismaticJoint);
  b2Assert(m_typeB == e_revoluteJoint || m_typeB == e_prismaticJoint);

  float32 coordinateA, coordinateB;



  m_bodyC = m_joint1->GetBodyA();
  m_bodyA = m_joint1->GetBodyB();

  // Get geometry of joint1
  b2Transform xfA = m_bodyA->m_xf;
  float32 aA = m_bodyA->m_sweep.a;
  b2Transform xfC = m_bodyC->m_xf;
```



```
  float32 aC = m_bodyC->m_sweep.a;

  if (m_typeA == e_revoluteJoint)
  {
    b2RevoluteJoint* revolute = (b2RevoluteJoint*)def->joint1;
    m_localAnchorC = revolute->m_localAnchorA;
    m_localAnchorA = revolute->m_localAnchorB;
    m_referenceAngleA = revolute->m_referenceAngle;
    m_localAxisC.SetZero();

    coordinateA = aA - aC - m_referenceAngleA;
  }
  else
  {
    b2PrismaticJoint* prismatic = (b2PrismaticJoint*)def->joint1;
    m_localAnchorC = prismatic->m_localAnchorA;
    m_localAnchorA = prismatic->m_localAnchorB;
    m_referenceAngleA = prismatic->m_referenceAngle;
    m_localAxisC = prismatic->m_localXAxisA;

    b2Vec2 pC = m_localAnchorC;
    b2Vec2 pA = b2MulT(xfC.q, b2Mul(xfA.q, m_localAnchorA) + (xfA.p - xfC.p));
    coordinateA = b2Dot(pA - pC, m_localAxisC);
  }

  m_bodyD = m_joint2->GetBodyA();
  m_bodyB = m_joint2->GetBodyB();

  // Get geometry of joint2
  b2Transform xfB = m_bodyB->m_xf;
  float32 aB = m_bodyB->m_sweep.a;
  b2Transform xfD = m_bodyD->m_xf;
  float32 aD = m_bodyD->m_sweep.a;

  if (m_typeB == e_revoluteJoint)
  {
    b2RevoluteJoint* revolute = (b2RevoluteJoint*)def->joint2;
    m_localAnchorD = revolute->m_localAnchorA;
    m_localAnchorB = revolute->m_localAnchorB;
    m_referenceAngleB = revolute->m_referenceAngle;
    m_localAxisD.SetZero();

    coordinateB = aB - aD - m_referenceAngleB;
  }
  else
  {
    b2PrismaticJoint* prismatic = (b2PrismaticJoint*)def->joint2;
    m_localAnchorD = prismatic->m_localAnchorA;
    m_localAnchorB = prismatic->m_localAnchorB;
    m_referenceAngleB = prismatic->m_referenceAngle;
    m_localAxisD = prismatic->m_localXAxisA;

    b2Vec2 pD = m_localAnchorD;
    b2Vec2 pB = b2MulT(xfD.q, b2Mul(xfB.q, m_localAnchorB) + (xfB.p - xfD.p));
    coordinateB = b2Dot(pB - pD, m_localAxisD);
  }

  m_ratio = def->ratio;

  m_constant = coordinateA + m_ratio * coordinateB;

  m_impulse = 0.0f;
}

void b2GearJoint::InitVelocityConstraints(const b2SolverData& data)
{
```

```
133     m_indexA = m_bodyA→m_islandIndex;
134     m_indexB = m_bodyB→m_islandIndex;
135     m_indexC = m_bodyC→m_islandIndex;
136     m_indexD = m_bodyD→m_islandIndex;
137     m_lcA = m_bodyA→m_sweep.localCenter;
138     m_lcB = m_bodyB→m_sweep.localCenter;
139     m_lcC = m_bodyC→m_sweep.localCenter;
140     m_lcD = m_bodyD→m_sweep.localCenter;
141     m_mA = m_bodyA→m_invMass;
142     m_mB = m_bodyB→m_invMass;
143     m_mC = m_bodyC→m_invMass;
144     m_mD = m_bodyD→m_invMass;
145     m_iA = m_bodyA→m_invI;
146     m_iB = m_bodyB→m_invI;
147     m_iC = m_bodyC→m_invI;
148     m_iD = m_bodyD→m_invI;
149
150     float32 aA = data.positions[m_indexA].a;
151     b2Vec2 vA = data.velocities[m_indexA].v;
152     float32 wA = data.velocities[m_indexA].w;
153
154     float32 aB = data.positions[m_indexB].a;
155     b2Vec2 vB = data.velocities[m_indexB].v;
156     float32 wB = data.velocities[m_indexB].w;
157
158     float32 aC = data.positions[m_indexC].a;
159     b2Vec2 vC = data.velocities[m_indexC].v;
160     float32 wC = data.velocities[m_indexC].w;
161
162     float32 aD = data.positions[m_indexD].a;
163     b2Vec2 vD = data.velocities[m_indexD].v;
164     float32 wD = data.velocities[m_indexD].w;
165
166     b2Rot qA(aA), qB(aB), qC(aC), qD(aD);
167
168     m_mass = 0.0f;
169
170     if (m_typeA ≡ e_revoluteJoint)
171     {
172       m_JvAC.SetZero();
173       m_JwA = 1.0f;
174       m_JwC = 1.0f;
175       m_mass += m_iA + m_iC;
176     }
177     else
178     {
179       b2Vec2 u = b2Mul(qC, m_localAxisC);
180       b2Vec2 rC = b2Mul(qC, m_localAnchorC − m_lcC);
181       b2Vec2 rA = b2Mul(qA, m_localAnchorA − m_lcA);
182       m_JvAC = u;
183       m_JwC = b2Cross(rC, u);
184       m_JwA = b2Cross(rA, u);
185       m_mass += m_mC + m_mA + m_iC * m_JwC * m_JwC + m_iA * m_JwA * m_JwA;
186     }
187
188     if (m_typeB ≡ e_revoluteJoint)
189     {
190       m_JvBD.SetZero();
191       m_JwB = m_ratio;
192       m_JwD = m_ratio;
193       m_mass += m_ratio * m_ratio * (m_iB + m_iD);
194     }
195     else
196     {
197       b2Vec2 u = b2Mul(qD, m_localAxisD);
198       b2Vec2 rD = b2Mul(qD, m_localAnchorD − m_lcD);
```



```
199       b2Vec2 rB = b2Mul(qB, m_localAnchorB − m_lcB);
200       m_JvBD = m_ratio * u;
201       m_JwD = m_ratio * b2Cross(rD, u);
202       m_JwB = m_ratio * b2Cross(rB, u);
203       m_mass += m_ratio * m_ratio * (m_mD + m_mB) + m_iD * m_JwD * m_JwD + m_iB *
    m_JwB * m_JwB;
204     }
205
206     // Compute effective mass.
207     m_mass = m_mass > 0.0f ? 1.0f / m_mass : 0.0f;
208
209     if (data.step.warmStarting)
210     {
211       vA += (m_mA * m_impulse) * m_JvAC;
212       wA += m_iA * m_impulse * m_JwA;
213       vB += (m_mB * m_impulse) * m_JvBD;
214       wB += m_iB * m_impulse * m_JwB;
215       vC −= (m_mC * m_impulse) * m_JvAC;
216       wC −= m_iC * m_impulse * m_JwC;
217       vD −= (m_mD * m_impulse) * m_JvBD;
218       wD −= m_iD * m_impulse * m_JwD;
219     }
220     else
221     {
222       m_impulse = 0.0f;
223     }
224
225     data.velocities[m_indexA].v = vA;
226     data.velocities[m_indexA].w = wA;
227     data.velocities[m_indexB].v = vB;
228     data.velocities[m_indexB].w = wB;
229     data.velocities[m_indexC].v = vC;
230     data.velocities[m_indexC].w = wC;
231     data.velocities[m_indexD].v = vD;
232     data.velocities[m_indexD].w = wD;
233 }
234
235 void b2GearJoint::SolveVelocityConstraints(const b2SolverData& data)
236 {
237     b2Vec2 vA = data.velocities[m_indexA].v;
238     float32 wA = data.velocities[m_indexA].w;
239     b2Vec2 vB = data.velocities[m_indexB].v;
240     float32 wB = data.velocities[m_indexB].w;
241     b2Vec2 vC = data.velocities[m_indexC].v;
242     float32 wC = data.velocities[m_indexC].w;
243     b2Vec2 vD = data.velocities[m_indexD].v;
244     float32 wD = data.velocities[m_indexD].w;
245
246     float32 Cdot = b2Dot(m_JvAC, vA − vC) + b2Dot(m_JvBD, vB − vD);
247     Cdot += (m_JwA * wA − m_JwC * wC) + (m_JwB * wB − m_JwD * wD);
248
249     float32 impulse = −m_mass * Cdot;
250     m_impulse += impulse;
251
252     vA += (m_mA * impulse) * m_JvAC;
253     wA += m_iA * impulse * m_JwA;
254     vB += (m_mB * impulse) * m_JvBD;
255     wB += m_iB * impulse * m_JwB;
256     vC −= (m_mC * impulse) * m_JvAC;
257     wC −= m_iC * impulse * m_JwC;
258     vD −= (m_mD * impulse) * m_JvBD;
259     wD −= m_iD * impulse * m_JwD;
260
261     data.velocities[m_indexA].v = vA;
262     data.velocities[m_indexA].w = wA;
263     data.velocities[m_indexB].v = vB;
```

```
264   data.velocities[m_indexB].w = wB;
265   data.velocities[m_indexC].v = vC;
266   data.velocities[m_indexC].w = wC;
267   data.velocities[m_indexD].v = vD;
268   data.velocities[m_indexD].w = wD;
269 }
270
271 bool b2GearJoint::SolvePositionConstraints(const b2SolverData& data)
272 {
273   b2Vec2 cA = data.positions[m_indexA].c;
274   float32 aA = data.positions[m_indexA].a;
275   b2Vec2 cB = data.positions[m_indexB].c;
276   float32 aB = data.positions[m_indexB].a;
277   b2Vec2 cC = data.positions[m_indexC].c;
278   float32 aC = data.positions[m_indexC].a;
279   b2Vec2 cD = data.positions[m_indexD].c;
280   float32 aD = data.positions[m_indexD].a;
281
282   b2Rot qA(aA), qB(aB), qC(aC), qD(aD);
283
284   float32 linearError = 0.0f;
285
286   float32 coordinateA, coordinateB;
287
288   b2Vec2 JvAC, JvBD;
289   float32 JwA, JwB, JwC, JwD;
290   float32 mass = 0.0f;
291
292   if (m_typeA == e_revoluteJoint)
293   {
294     JvAC.SetZero();
295     JwA = 1.0f;
296     JwC = 1.0f;
297     mass += m_iA + m_iC;
298
299     coordinateA = aA - aC - m_referenceAngleA;
300   }
301   else
302   {
303     b2Vec2 u = b2Mul(qC, m_localAxisC);
304     b2Vec2 rC = b2Mul(qC, m_localAnchorC - m_lcC);
305     b2Vec2 rA = b2Mul(qA, m_localAnchorA - m_lcA);
306     JvAC = u;
307     JwC = b2Cross(rC, u);
308     JwA = b2Cross(rA, u);
309     mass += m_mC + m_mA + m_iC * JwC * JwC + m_iA * JwA * JwA;
310
311     b2Vec2 pC = m_localAnchorC - m_lcC;
312     b2Vec2 pA = b2MulT(qC, rA + (cA - cC));
313     coordinateA = b2Dot(pA - pC, m_localAxisC);
314   }
315
316   if (m_typeB == e_revoluteJoint)
317   {
318     JvBD.SetZero();
319     JwB = m_ratio;
320     JwD = m_ratio;
321     mass += m_ratio * m_ratio * (m_iB + m_iD);
322
323     coordinateB = aB - aD - m_referenceAngleB;
324   }
325   else
326   {
327     b2Vec2 u = b2Mul(qD, m_localAxisD);
328     b2Vec2 rD = b2Mul(qD, m_localAnchorD - m_lcD);
329     b2Vec2 rB = b2Mul(qB, m_localAnchorB - m_lcB);
```



```
330     JvBD = m_ratio * u;
331     JwD = m_ratio * b2Cross(rD, u);
332     JwB = m_ratio * b2Cross(rB, u);
333     mass += m_ratio * m_ratio * (m_mD + m_mB) + m_iD * JwD * JwD + m_iB * JwB *
    JwB;
334
335     b2Vec2 pD = m_localAnchorD - m_lcD;
336     b2Vec2 pB = b2MulT(qD, rB + (cB - cD));
337     coordinateB = b2Dot(pB - pD, m_localAxisD);
338   }
339
340   float32 C = (coordinateA + m_ratio * coordinateB) - m_constant;
341
342   float32 impulse = 0.0f;
343   if (mass > 0.0f)
344   {
345     impulse = -C / mass;
346   }
347
348   cA += m_mA * impulse * JvAC;
349   aA += m_iA * impulse * JwA;
350   cB += m_mB * impulse * JvBD;
351   aB += m_iB * impulse * JwB;
352   cC -= m_mC * impulse * JvAC;
353   aC -= m_iC * impulse * JwC;
354   cD -= m_mD * impulse * JvBD;
355   aD -= m_iD * impulse * JwD;
356
357   data.positions[m_indexA].c = cA;
358   data.positions[m_indexA].a = aA;
359   data.positions[m_indexB].c = cB;
360   data.positions[m_indexB].a = aB;
361   data.positions[m_indexC].c = cC;
362   data.positions[m_indexC].a = aC;
363   data.positions[m_indexD].c = cD;
364   data.positions[m_indexD].a = aD;
365
366
367   return linearError < b2_linearSlop;
368 }
369
370 b2Vec2 b2GearJoint::GetAnchorA() const
371 {
372   return m_bodyA->GetWorldPoint(m_localAnchorA);
373 }
374
375 b2Vec2 b2GearJoint::GetAnchorB() const
376 {
377   return m_bodyB->GetWorldPoint(m_localAnchorB);
378 }
379
380 b2Vec2 b2GearJoint::GetReactionForce(float32 inv_dt) const
381 {
382   b2Vec2 P = m_impulse * m_JvAC;
383   return inv_dt * P;
384 }
385
386 float32 b2GearJoint::GetReactionTorque(float32 inv_dt) const
387 {
388   float32 L = m_impulse * m_JwA;
389   return inv_dt * L;
390 }
391
392 void b2GearJoint::SetRatio(float32 ratio)
393 {
394   b2Assert(b2IsValid(ratio));
```

```
395   m_ratio = ratio;
396 }
397
398 float32 b2GearJoint::GetRatio() const
399 {
400   return m_ratio;
401 }
402
403 void b2GearJoint::Dump()
404 {
405   int32 indexA = m_bodyA->m_islandIndex;
406   int32 indexB = m_bodyB->m_islandIndex;
407
408   int32 index1 = m_joint1->m_index;
409   int32 index2 = m_joint2->m_index;
410
411   b2Log(" b2GearJointDef jd;\n");
412   b2Log(" jd.bodyA = bodies[%d];\n", indexA);
413   b2Log(" jd.bodyB = bodies[%d];\n", indexB);
414   b2Log(" jd.collideConnected = bool(%d);\n", m_collideConnected);
415   b2Log(" jd.joint1 = joints[%d];\n", index1);
416   b2Log(" jd.joint2 = joints[%d];\n", index2);
417   b2Log(" jd.ratio = %.15lef;\n", m_ratio);
418   b2Log(" joints[%d] = m_world->CreateJoint(&jd);\n", m_index);
419 }
```



```
1  /*
2  * Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as-is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #include "Box2D/Dynamics/Joints/b2FrictionJoint.h"
20 #include "Box2D/Dynamics/b2Body.h"
21 #include "Box2D/Dynamics/b2TimeStep.h"
22
23 // Point-to-point constraint
24 // Cdot = v2 - v1
25 //      = v2 + cross(w2, r2) - v1 - cross(w1, r1)
26 // J = [-I -r1_skew I r2_skew ]
27 // Identity used:
28 // w k % (rx i + ry j) = w * (-ry i + rx j)
29
30 // Angle constraint
31 // Cdot = w2 - w1
32 // J = [0 0 -1 0 0 1]
33 // K = invI1 + invI2
34
35 void b2FrictionJointDef::Initialize(b2Body* bA, b2Body* bB, const b2Vec2& anchor
   )
36 {
37   bodyA = bA;
38   bodyB = bB;
39   localAnchorA = bodyA->GetLocalPoint(anchor);
40   localAnchorB = bodyB->GetLocalPoint(anchor);
41 }
42
43 b2FrictionJoint::b2FrictionJoint(const b2FrictionJointDef* def)
44 : b2Joint(def)
45 {
46   m_localAnchorA = def->localAnchorA;
47   m_localAnchorB = def->localAnchorB;
48
49   m_linearImpulse.SetZero();
50   m_angularImpulse = 0.0f;
51
52   m_maxForce = def->maxForce;
53   m_maxTorque = def->maxTorque;
54 }
55
56 void b2FrictionJoint::InitVelocityConstraints(const b2SolverData& data)
57 {
58   m_indexA = m_bodyA->m_islandIndex;
59   m_indexB = m_bodyB->m_islandIndex;
60   m_localCenterA = m_bodyA->m_sweep.localCenter;
61   m_localCenterB = m_bodyB->m_sweep.localCenter;
62   m_invMassA = m_bodyA->m_invMass;
63   m_invMassB = m_bodyB->m_invMass;
64   m_invIA = m_bodyA->m_invI;
65   m_invIB = m_bodyB->m_invI;
```

```
66
67    float32 aA = data.positions[m_indexA].a;
68    b2Vec2 vA = data.velocities[m_indexA].v;
69    float32 wA = data.velocities[m_indexA].w;
70
71    float32 aB = data.positions[m_indexB].a;
72    b2Vec2 vB = data.velocities[m_indexB].v;
73    float32 wB = data.velocities[m_indexB].w;
74
75    b2Rot qA(aA), qB(aB);
76
77    // Compute the effective mass matrix.
78    m_rA = b2Mul(qA, m_localAnchorA - m_localCenterA);
79    m_rB = b2Mul(qB, m_localAnchorB - m_localCenterB);
80
81    // J = [-I -r1_skew I r2_skew]
82    //     [ 0       -1 0       1]
83    // r_skew = [-ry; rx]
84
85    // Matlab
86    // K = [ mA+r1y^2*iA+mB+r2y^2*iB,  -r1y*iA*r1x-r2y*iB*r2x,          -r1y*iA-r2
    y*iB]
87    //     [  -r1y*iA*r1x-r2y*iB*r2x, mA+r1x^2*iA+mB+r2x^2*iB,           r1x*iA+r2
    x*iB]
88    //     [          -r1y*iA-r2y*iB,           r1x*iA+r2x*iB,                   i
    A+iB]
89
90    float32 mA = m_invMassA, mB = m_invMassB;
91    float32 iA = m_invIA, iB = m_invIB;
92
93    b2Mat22 K;
94    K.ex.x = mA + mB + iA * m_rA.y * m_rA.y + iB * m_rB.y * m_rB.y;
95    K.ex.y = -iA * m_rA.x * m_rA.y - iB * m_rB.x * m_rB.y;
96    K.ey.x = K.ex.y;
97    K.ey.y = mA + mB + iA * m_rA.x * m_rA.x + iB * m_rB.x * m_rB.x;
98
99    m_linearMass = K.GetInverse();
100
101   m_angularMass = iA + iB;
102   if (m_angularMass > 0.0f)
103   {
104     m_angularMass = 1.0f / m_angularMass;
105   }
106
107   if (data.step.warmStarting)
108   {
109     // Scale impulses to support a variable time step.
110     m_linearImpulse *= data.step.dtRatio;
111     m_angularImpulse *= data.step.dtRatio;
112
113     b2Vec2 P(m_linearImpulse.x, m_linearImpulse.y);
114     vA -= mA * P;
115     wA -= iA * (b2Cross(m_rA, P) + m_angularImpulse);
116     vB += mB * P;
117     wB += iB * (b2Cross(m_rB, P) + m_angularImpulse);
118   }
119   else
120   {
121     m_linearImpulse.SetZero();
122     m_angularImpulse = 0.0f;
123   }
124
125   data.velocities[m_indexA].v = vA;
126   data.velocities[m_indexA].w = wA;
127   data.velocities[m_indexB].v = vB;
128   data.velocities[m_indexB].w = wB;
```



```
129 }
130
131 void b2FrictionJoint::SolveVelocityConstraints(const b2SolverData& data)
132 {
133   b2Vec2 vA = data.velocities[m_indexA].v;
134   float32 wA = data.velocities[m_indexA].w;
135   b2Vec2 vB = data.velocities[m_indexB].v;
136   float32 wB = data.velocities[m_indexB].w;
137
138   float32 mA = m_invMassA, mB = m_invMassB;
139   float32 iA = m_invIA, iB = m_invIB;
140
141   float32 h = data.step.dt;
142
143   // Solve angular friction
144   {
145     float32 Cdot = wB - wA;
146     float32 impulse = -m_angularMass * Cdot;
147
148     float32 oldImpulse = m_angularImpulse;
149     float32 maxImpulse = h * m_maxTorque;
150     m_angularImpulse = b2Clamp(m_angularImpulse + impulse, -maxImpulse, maxImpul
    se);
151     impulse = m_angularImpulse - oldImpulse;
152
153     wA -= iA * impulse;
154     wB += iB * impulse;
155   }
156
157   // Solve linear friction
158   {
159     b2Vec2 Cdot = vB + b2Cross(wB, m_rB) - vA - b2Cross(wA, m_rA);
160
161     b2Vec2 impulse = -b2Mul(m_linearMass, Cdot);
162     b2Vec2 oldImpulse = m_linearImpulse;
163     m_linearImpulse += impulse;
164
165     float32 maxImpulse = h * m_maxForce;
166
167     if (m_linearImpulse.LengthSquared() > maxImpulse * maxImpulse)
168     {
169       m_linearImpulse.Normalize();
170       m_linearImpulse *= maxImpulse;
171     }
172
173     impulse = m_linearImpulse - oldImpulse;
174
175     vA -= mA * impulse;
176     wA -= iA * b2Cross(m_rA, impulse);
177
178     vB += mB * impulse;
179     wB += iB * b2Cross(m_rB, impulse);
180   }
181
182   data.velocities[m_indexA].v = vA;
183   data.velocities[m_indexA].w = wA;
184   data.velocities[m_indexB].v = vB;
185   data.velocities[m_indexB].w = wB;
186 }
187
188 bool b2FrictionJoint::SolvePositionConstraints(const b2SolverData& data)
189 {
190   B2_NOT_USED(data);
191
192   return true;
193 }
```

```
194
195 b2Vec2 b2FrictionJoint::GetAnchorA() const
196 {
197   return m_bodyA→GetWorldPoint(m_localAnchorA);
198 }
199
200 b2Vec2 b2FrictionJoint::GetAnchorB() const
201 {
202   return m_bodyB→GetWorldPoint(m_localAnchorB);
203 }
204
205 b2Vec2 b2FrictionJoint::GetReactionForce(float32 inv_dt) const
206 {
207   return inv_dt * m_linearImpulse;
208 }
209
210 float32 b2FrictionJoint::GetReactionTorque(float32 inv_dt) const
211 {
212   return inv_dt * m_angularImpulse;
213 }
214
215 void b2FrictionJoint::SetMaxForce(float32 force)
216 {
217   b2Assert(b2IsValid(force) ∧ force ≥ 0.0f);
218   m_maxForce = force;
219 }
220
221 float32 b2FrictionJoint::GetMaxForce() const
222 {
223   return m_maxForce;
224 }
225
226 void b2FrictionJoint::SetMaxTorque(float32 torque)
227 {
228   b2Assert(b2IsValid(torque) ∧ torque ≥ 0.0f);
229   m_maxTorque = torque;
230 }
231
232 float32 b2FrictionJoint::GetMaxTorque() const
233 {
234   return m_maxTorque;
235 }
236
237 void b2FrictionJoint::Dump()
238 {
239   int32 indexA = m_bodyA→m_islandIndex;
240   int32 indexB = m_bodyB→m_islandIndex;
241
242   b2Log(" b2FrictionJointDef jd;\n");
243   b2Log(" jd.bodyA = bodies[%d];\n", indexA);
244   b2Log(" jd.bodyB = bodies[%d];\n", indexB);
245   b2Log(" jd.collideConnected = bool(%d);\n", m_collideConnected);
246   b2Log(" jd.localAnchorA.Set(%.15lef, %.15lef);\n", m_localAnchorA.x, m_localAnchorA.y);
247   b2Log(" jd.localAnchorB.Set(%.15lef, %.15lef);\n", m_localAnchorB.x, m_localAnchorB.y);
248   b2Log(" jd.maxForce = %.15lef;\n", m_maxForce);
249   b2Log(" jd.maxTorque = %.15lef;\n", m_maxTorque);
250   b2Log(" joints[%d] = m_world->CreateJoint(&jd);\n", m_index);
251 }
```



```
1  /*
2  * Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as-is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #include "Box2D/Dynamics/Joints/b2DistanceJoint.h"
20 #include "Box2D/Dynamics/b2Body.h"
21 #include "Box2D/Dynamics/b2TimeStep.h"
22
23 // 1-D constrained system
24 // m (v2 - v1) = lambda
25 // v2 + (beta/h) * x1 + gamma * lambda = 0, gamma has units of inverse mass.
26 // x2 = x1 + h * v2
27
28 // 1-D mass-damper-spring system
29 // m (v2 - v1) + h * d * v2 + h * k *
30
31 // C = norm(p2 - p1) - L
32 // u = (p2 - p1) / norm(p2 - p1)
33 // Cdot = dot(u, v2 + cross(w2, r2) - v1 - cross(w1, r1))
34 // J = [-u -cross(r1, u) u cross(r2, u)]
35 // K = J * invM * JT
36 //   = invMass1 + invI1 * cross(r1, u)^2 + invMass2 + invI2 * cross(r2, u)^2
37
38 void b2DistanceJointDef::Initialize(b2Body* b1, b2Body* b2,
39                                     const b2Vec2& anchor1, const b2Vec2& anchor2)
40 {
41   bodyA = b1;
42   bodyB = b2;
43   localAnchorA = bodyA→GetLocalPoint(anchor1);
44   localAnchorB = bodyB→GetLocalPoint(anchor2);
45   b2Vec2 d = anchor2 - anchor1;
46   length = d.Length();
47 }
48
49 b2DistanceJoint::b2DistanceJoint(const b2DistanceJointDef* def)
50 : b2Joint(def)
51 {
52   m_localAnchorA = def→localAnchorA;
53   m_localAnchorB = def→localAnchorB;
54   m_length = def→length;
55   m_frequencyHz = def→frequencyHz;
56   m_dampingRatio = def→dampingRatio;
57   m_impulse = 0.0f;
58   m_gamma = 0.0f;
59   m_bias = 0.0f;
60 }
61
62 void b2DistanceJoint::InitVelocityConstraints(const b2SolverData& data)
63 {
64   m_indexA = m_bodyA→m_islandIndex;
65   m_indexB = m_bodyB→m_islandIndex;
66   m_localCenterA = m_bodyA→m_sweep.localCenter;
```

```
67    m_localCenterB = m_bodyB→m_sweep.localCenter;
68    m_invMassA = m_bodyA→m_invMass;
69    m_invMassB = m_bodyB→m_invMass;
70    m_invIA = m_bodyA→m_invI;
71    m_invIB = m_bodyB→m_invI;
72
73    b2Vec2 cA = data.positions[m_indexA].c;
74    float32 aA = data.positions[m_indexA].a;
75    b2Vec2 vA = data.velocities[m_indexA].v;
76    float32 wA = data.velocities[m_indexA].w;
77
78    b2Vec2 cB = data.positions[m_indexB].c;
79    float32 aB = data.positions[m_indexB].a;
80    b2Vec2 vB = data.velocities[m_indexB].v;
81    float32 wB = data.velocities[m_indexB].w;
82
83    b2Rot qA(aA), qB(aB);
84
85    m_rA = b2Mul(qA, m_localAnchorA − m_localCenterA);
86    m_rB = b2Mul(qB, m_localAnchorB − m_localCenterB);
87    m_u = cB + m_rB − cA − m_rA;
88
89    // Handle singularity.
90    float32 length = m_u.Length();
91    if (length > b2_linearSlop)
92    {
93      m_u *= 1.0f / length;
94    }
95    else
96    {
97      m_u.Set(0.0f, 0.0f);
98    }
99
100   float32 crAu = b2Cross(m_rA, m_u);
101   float32 crBu = b2Cross(m_rB, m_u);
102   float32 invMass = m_invMassA + m_invIA * crAu * crAu + m_invMassB + m_invIB *
    crBu * crBu;
103
104   // Compute the effective mass matrix.
105   m_mass = invMass ≠ 0.0f ? 1.0f / invMass : 0.0f;
106
107   if (m_frequencyHz > 0.0f)
108   {
109     float32 C = length − m_length;
110
111     // Frequency
112     float32 omega = 2.0f * b2_pi * m_frequencyHz;
113
114     // Damping coefficient
115     float32 d = 2.0f * m_mass * m_dampingRatio * omega;
116
117     // Spring stiffness
118     float32 k = m_mass * omega * omega;
119
120     // magic formulas
121     float32 h = data.step.dt;
122     m_gamma = h * (d + h * k);
123     m_gamma = m_gamma ≠ 0.0f ? 1.0f / m_gamma : 0.0f;
124     m_bias = C * h * k * m_gamma;
125
126     invMass += m_gamma;
127     m_mass = invMass ≠ 0.0f ? 1.0f / invMass : 0.0f;
128   }
129   else
130   {
131     m_gamma = 0.0f;
```



```
132     m_bias = 0.0f;
133   }
134
135   if (data.step.warmStarting)
136   {
137     // Scale the impulse to support a variable time step.
138     m_impulse *= data.step.dtRatio;
139
140     b2Vec2 P = m_impulse * m_u;
141     vA −= m_invMassA * P;
142     wA −= m_invIA * b2Cross(m_rA, P);
143     vB += m_invMassB * P;
144     wB += m_invIB * b2Cross(m_rB, P);
145   }
146   else
147   {
148     m_impulse = 0.0f;
149   }
150
151   data.velocities[m_indexA].v = vA;
152   data.velocities[m_indexA].w = wA;
153   data.velocities[m_indexB].v = vB;
154   data.velocities[m_indexB].w = wB;
155 }
156
157 void b2DistanceJoint::SolveVelocityConstraints(const b2SolverData& data)
158 {
159   b2Vec2 vA = data.velocities[m_indexA].v;
160   float32 wA = data.velocities[m_indexA].w;
161   b2Vec2 vB = data.velocities[m_indexB].v;
162   float32 wB = data.velocities[m_indexB].w;
163
164   // Cdot = dot(u, v + cross(w, r))
165   b2Vec2 vpA = vA + b2Cross(wA, m_rA);
166   b2Vec2 vpB = vB + b2Cross(wB, m_rB);
167   float32 Cdot = b2Dot(m_u, vpB − vpA);
168
169   float32 impulse = −m_mass * (Cdot + m_bias + m_gamma * m_impulse);
170   m_impulse += impulse;
171
172   b2Vec2 P = impulse * m_u;
173   vA −= m_invMassA * P;
174   wA −= m_invIA * b2Cross(m_rA, P);
175   vB += m_invMassB * P;
176   wB += m_invIB * b2Cross(m_rB, P);
177
178   data.velocities[m_indexA].v = vA;
179   data.velocities[m_indexA].w = wA;
180   data.velocities[m_indexB].v = vB;
181   data.velocities[m_indexB].w = wB;
182 }
183
184 bool b2DistanceJoint::SolvePositionConstraints(const b2SolverData& data)
185 {
186   if (m_frequencyHz > 0.0f)
187   {
188     // There is no position correction for soft distance constraints.
189     return true;
190   }
191
192   b2Vec2 cA = data.positions[m_indexA].c;
193   float32 aA = data.positions[m_indexA].a;
194   b2Vec2 cB = data.positions[m_indexB].c;
195   float32 aB = data.positions[m_indexB].a;
196
197   b2Rot qA(aA), qB(aB);
```

```
198
199   b2Vec2 rA = b2Mul(qA, m_localAnchorA − m_localCenterA);
200   b2Vec2 rB = b2Mul(qB, m_localAnchorB − m_localCenterB);
201   b2Vec2 u = cB + rB − cA − rA;
202
203   float32 length = u.Normalize();
204   float32 C = length − m_length;
205   C = b2Clamp(C, −b2_maxLinearCorrection, b2_maxLinearCorrection);
206
207   float32 impulse = −m_mass * C;
208   b2Vec2 P = impulse * u;
209
210   cA −= m_invMassA * P;
211   aA −= m_invIA * b2Cross(rA, P);
212   cB += m_invMassB * P;
213   aB += m_invIB * b2Cross(rB, P);
214
215   data.positions[m_indexA].c = cA;
216   data.positions[m_indexA].a = aA;
217   data.positions[m_indexB].c = cB;
218   data.positions[m_indexB].a = aB;
219
220   return b2Abs(C) < b2_linearSlop;
221 }
222
223 b2Vec2 b2DistanceJoint::GetAnchorA() const
224 {
225   return m_bodyA→GetWorldPoint(m_localAnchorA);
226 }
227
228 b2Vec2 b2DistanceJoint::GetAnchorB() const
229 {
230   return m_bodyB→GetWorldPoint(m_localAnchorB);
231 }
232
233 b2Vec2 b2DistanceJoint::GetReactionForce(float32 inv_dt) const
234 {
235   b2Vec2 F = (inv_dt * m_impulse) * m_u;
236   return F;
237 }
238
239 float32 b2DistanceJoint::GetReactionTorque(float32 inv_dt) const
240 {
241   B2_NOT_USED(inv_dt);
242   return 0.0f;
243 }
244
245 void b2DistanceJoint::Dump()
246 {
247   int32 indexA = m_bodyA→m_islandIndex;
248   int32 indexB = m_bodyB→m_islandIndex;
249
250   b2Log(" b2DistanceJointDef jd;\n");
251   b2Log(" jd.bodyA = bodies[%d];\n", indexA);
252   b2Log(" jd.bodyB = bodies[%d];\n", indexB);
253   b2Log(" jd.collideConnected = bool(%d);\n", m_collideConnected);
254   b2Log(" jd.localAnchorA.Set(%.15lef, %.15lef);\n", m_localAnchorA.x, m_localAnchorA.y);
255   b2Log(" jd.localAnchorB.Set(%.15lef, %.15lef);\n", m_localAnchorB.x, m_localAnchorB.y);
256   b2Log(" jd.length = %.15lef;\n", m_length);
257   b2Log(" jd.frequencyHz = %.15lef;\n", m_frequencyHz);
258   b2Log(" jd.dampingRatio = %.15lef;\n", m_dampingRatio);
259   b2Log(" joints[%d] = m_world−>CreateJoint(&jd);\n", m_index);
260 }
```



```
1  /*
2  * Copyright (c) 2006−2009 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as−is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #include "Box2D/Dynamics/Contacts/b2PolygonContact.h"
20 #include "Box2D/Common/b2BlockAllocator.h"
21 #include "Box2D/Collision/b2TimeOfImpact.h"
22 #include "Box2D/Dynamics/b2Body.h"
23 #include "Box2D/Dynamics/b2Fixture.h"
24 #include "Box2D/Dynamics/b2WorldCallbacks.h"
25
26 #include <new>
27
28 b2Contact* b2PolygonContact::Create(b2Fixture* fixtureA, int32, b2Fixture* fixtu
   reB, int32, b2BlockAllocator* allocator)
29 {
30   void* mem = allocator→Allocate(sizeof(b2PolygonContact));
31   return new (mem) b2PolygonContact(fixtureA, fixtureB);
32 }
33
34 void b2PolygonContact::Destroy(b2Contact* contact, b2BlockAllocator* allocator)
35 {
36   ((b2PolygonContact*)contact)→~b2PolygonContact();
37   allocator→Free(contact, sizeof(b2PolygonContact));
38 }
39
40 b2PolygonContact::b2PolygonContact(b2Fixture* fixtureA, b2Fixture* fixtureB)
41   : b2Contact(fixtureA, 0, fixtureB, 0)
42 {
43   b2Assert(m_fixtureA→GetType() ≡ b2Shape::e_polygon);
44   b2Assert(m_fixtureB→GetType() ≡ b2Shape::e_polygon);
45 }
46
47 void b2PolygonContact::Evaluate(b2Manifold* manifold, const b2Transform& xfA, co
   nst b2Transform& xfB)
48 {
49   b2CollidePolygons(  manifold,
50             (b2PolygonShape*)m_fixtureA→GetShape(), xfA,
51             (b2PolygonShape*)m_fixtureB→GetShape(), xfB);
52 }
```

nov 26, 19 17:34 **b2PolygonAndCircleContact.cpp** Page 1/1

```cpp
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Contacts/b2PolygonAndCircleContact.h"
#include "Box2D/Common/b2BlockAllocator.h"
#include "Box2D/Dynamics/b2Fixture.h"

#include <new>

b2Contact* b2PolygonAndCircleContact::Create(b2Fixture* fixtureA, int32, b2Fixture* fixtureB, int32, b2BlockAllocator* allocator)
{
  void* mem = allocator->Allocate(sizeof(b2PolygonAndCircleContact));
  return new (mem) b2PolygonAndCircleContact(fixtureA, fixtureB);
}

void b2PolygonAndCircleContact::Destroy(b2Contact* contact, b2BlockAllocator* allocator)
{
  ((b2PolygonAndCircleContact*)contact)->~b2PolygonAndCircleContact();
  allocator->Free(contact, sizeof(b2PolygonAndCircleContact));
}

b2PolygonAndCircleContact::b2PolygonAndCircleContact(b2Fixture* fixtureA, b2Fixture* fixtureB)
: b2Contact(fixtureA, 0, fixtureB, 0)
{
  b2Assert(m_fixtureA->GetType() == b2Shape::e_polygon);
  b2Assert(m_fixtureB->GetType() == b2Shape::e_circle);
}

void b2PolygonAndCircleContact::Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform& xfB)
{
  b2CollidePolygonAndCircle(  manifold,
                (b2PolygonShape*)m_fixtureA->GetShape(), xfA,
                (b2CircleShape*)m_fixtureB->GetShape(), xfB);
}
```

nov 26, 19 17:34 **b2EdgeAndPolygonContact.cpp** Page 1/1

```cpp
/*
* Copyright (c) 2006-2010 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Contacts/b2EdgeAndPolygonContact.h"
#include "Box2D/Common/b2BlockAllocator.h"
#include "Box2D/Dynamics/b2Fixture.h"

#include <new>

b2Contact* b2EdgeAndPolygonContact::Create(b2Fixture* fixtureA, int32, b2Fixture* fixtureB, int32, b2BlockAllocator* allocator)
{
  void* mem = allocator->Allocate(sizeof(b2EdgeAndPolygonContact));
  return new (mem) b2EdgeAndPolygonContact(fixtureA, fixtureB);
}

void b2EdgeAndPolygonContact::Destroy(b2Contact* contact, b2BlockAllocator* allocator)
{
  ((b2EdgeAndPolygonContact*)contact)->~b2EdgeAndPolygonContact();
  allocator->Free(contact, sizeof(b2EdgeAndPolygonContact));
}

b2EdgeAndPolygonContact::b2EdgeAndPolygonContact(b2Fixture* fixtureA, b2Fixture* fixtureB)
: b2Contact(fixtureA, 0, fixtureB, 0)
{
  b2Assert(m_fixtureA->GetType() == b2Shape::e_edge);
  b2Assert(m_fixtureB->GetType() == b2Shape::e_polygon);
}

void b2EdgeAndPolygonContact::Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform& xfB)
{
  b2CollideEdgeAndPolygon(  manifold,
                (b2EdgeShape*)m_fixtureA->GetShape(), xfA,
                (b2PolygonShape*)m_fixtureB->GetShape(), xfB);
}
```

nov 26, 19 17:34 **b2EdgeAndCircleContact.cpp** Page 1/1

```
/*
* Copyright (c) 2006-2010 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Contacts/b2EdgeAndCircleContact.h"
#include "Box2D/Common/b2BlockAllocator.h"
#include "Box2D/Dynamics/b2Fixture.h"

#include <new>

b2Contact* b2EdgeAndCircleContact::Create(b2Fixture* fixtureA, int32, b2Fixture*
 fixtureB, int32, b2BlockAllocator* allocator)
{
  void* mem = allocator→Allocate(sizeof(b2EdgeAndCircleContact));
  return new (mem) b2EdgeAndCircleContact(fixtureA, fixtureB);
}

void b2EdgeAndCircleContact::Destroy(b2Contact* contact, b2BlockAllocator* alloc
ator)
{
  ((b2EdgeAndCircleContact*)contact)→~b2EdgeAndCircleContact();
  allocator→Free(contact, sizeof(b2EdgeAndCircleContact));
}

b2EdgeAndCircleContact::b2EdgeAndCircleContact(b2Fixture* fixtureA, b2Fixture* f
ixtureB)
: b2Contact(fixtureA, 0, fixtureB, 0)
{
  b2Assert(m_fixtureA→GetType() ≡ b2Shape::e_edge);
  b2Assert(m_fixtureB→GetType() ≡ b2Shape::e_circle);
}

void b2EdgeAndCircleContact::Evaluate(b2Manifold* manifold, const b2Transform& x
fA, const b2Transform& xfB)
{
  b2CollideEdgeAndCircle( manifold,
                (b2EdgeShape*)m_fixtureA→GetShape(), xfA,
                (b2CircleShape*)m_fixtureB→GetShape(), xfB);
}
```

nov 26, 19 17:34 **b2ContactSolver.cpp** Page 1/13

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Contacts/b2ContactSolver.h"

#include "Box2D/Dynamics/Contacts/b2Contact.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2Fixture.h"
#include "Box2D/Dynamics/b2World.h"
#include "Box2D/Common/b2StackAllocator.h"

// Solver debugging is normally disabled because the block solver sometimes has
to deal with a poorly conditioned effective mass matrix.
#define B2_DEBUG_SOLVER 0

bool g_blockSolve = true;

struct b2ContactPositionConstraint
{
  b2Vec2 localPoints[b2_maxManifoldPoints];
  b2Vec2 localNormal;
  b2Vec2 localPoint;
  int32 indexA;
  int32 indexB;
  float32 invMassA, invMassB;
  b2Vec2 localCenterA, localCenterB;
  float32 invIA, invIB;
  b2Manifold::Type type;
  float32 radiusA, radiusB;
  int32 pointCount;
};

b2ContactSolver::b2ContactSolver(b2ContactSolverDef* def)
{
  m_step = def→step;
  m_allocator = def→allocator;
  m_count = def→count;
  m_positionConstraints = (b2ContactPositionConstraint*)m_allocator→Allocate(m_
count * sizeof(b2ContactPositionConstraint));
  m_velocityConstraints = (b2ContactVelocityConstraint*)m_allocator→Allocate(m_
count * sizeof(b2ContactVelocityConstraint));
  m_positions = def→positions;
  m_velocities = def→velocities;
  m_contacts = def→contacts;

  // Initialize position independent portions of the constraints.
  for (int32 i = 0; i < m_count; ++i)
  {
    b2Contact* contact = m_contacts[i];

    b2Fixture* fixtureA = contact→m_fixtureA;
```

```
64      b2Fixture* fixtureB = contact→m_fixtureB;
65      b2Shape* shapeA = fixtureA→GetShape();
66      b2Shape* shapeB = fixtureB→GetShape();
67      float32 radiusA = shapeA→m_radius;
68      float32 radiusB = shapeB→m_radius;
69      b2Body* bodyA = fixtureA→GetBody();
70      b2Body* bodyB = fixtureB→GetBody();
71      b2Manifold* manifold = contact→GetManifold();
72
73      int32 pointCount = manifold→pointCount;
74      b2Assert(pointCount > 0);
75
76      b2ContactVelocityConstraint* vc = m_velocityConstraints + i;
77      vc→friction = contact→m_friction;
78      vc→restitution = contact→m_restitution;
79      vc→tangentSpeed = contact→m_tangentSpeed;
80      vc→indexA = bodyA→m_islandIndex;
81      vc→indexB = bodyB→m_islandIndex;
82      vc→invMassA = bodyA→m_invMass;
83      vc→invMassB = bodyB→m_invMass;
84      vc→invIA = bodyA→m_invI;
85      vc→invIB = bodyB→m_invI;
86      vc→contactIndex = i;
87      vc→pointCount = pointCount;
88      vc→K.SetZero();
89      vc→normalMass.SetZero();
90
91      b2ContactPositionConstraint* pc = m_positionConstraints + i;
92      pc→indexA = bodyA→m_islandIndex;
93      pc→indexB = bodyB→m_islandIndex;
94      pc→invMassA = bodyA→m_invMass;
95      pc→invMassB = bodyB→m_invMass;
96      pc→localCenterA = bodyA→m_sweep.localCenter;
97      pc→localCenterB = bodyB→m_sweep.localCenter;
98      pc→invIA = bodyA→m_invI;
99      pc→invIB = bodyB→m_invI;
100     pc→localNormal = manifold→localNormal;
101     pc→localPoint = manifold→localPoint;
102     pc→pointCount = pointCount;
103     pc→radiusA = radiusA;
104     pc→radiusB = radiusB;
105     pc→type = manifold→type;
106
107     for (int32 j = 0; j < pointCount; ++j)
108     {
109       b2ManifoldPoint* cp = manifold→points + j;
110       b2VelocityConstraintPoint* vcp = vc→points + j;
111
112       if (m_step.warmStarting)
113       {
114         vcp→normalImpulse = m_step.dtRatio * cp→normalImpulse;
115         vcp→tangentImpulse = m_step.dtRatio * cp→tangentImpulse;
116       }
117       else
118       {
119         vcp→normalImpulse = 0.0f;
120         vcp→tangentImpulse = 0.0f;
121       }
122
123       vcp→rA.SetZero();
124       vcp→rB.SetZero();
125       vcp→normalMass = 0.0f;
126       vcp→tangentMass = 0.0f;
127       vcp→velocityBias = 0.0f;
128
129       pc→localPoints[j] = cp→localPoint;
```



```
130     }
131   }
132 }
133
134 b2ContactSolver::~b2ContactSolver()
135 {
136   m_allocator→Free(m_velocityConstraints);
137   m_allocator→Free(m_positionConstraints);
138 }
139
140 // Initialize position dependent portions of the velocity constraints.
141 void b2ContactSolver::InitializeVelocityConstraints()
142 {
143   for (int32 i = 0; i < m_count; ++i)
144   {
145     b2ContactVelocityConstraint* vc = m_velocityConstraints + i;
146     b2ContactPositionConstraint* pc = m_positionConstraints + i;
147
148     float32 radiusA = pc→radiusA;
149     float32 radiusB = pc→radiusB;
150     b2Manifold* manifold = m_contacts[vc→contactIndex]→GetManifold();
151
152     int32 indexA = vc→indexA;
153     int32 indexB = vc→indexB;
154
155     float32 mA = vc→invMassA;
156     float32 mB = vc→invMassB;
157     float32 iA = vc→invIA;
158     float32 iB = vc→invIB;
159     b2Vec2 localCenterA = pc→localCenterA;
160     b2Vec2 localCenterB = pc→localCenterB;
161
162     b2Vec2 cA = m_positions[indexA].c;
163     float32 aA = m_positions[indexA].a;
164     b2Vec2 vA = m_velocities[indexA].v;
165     float32 wA = m_velocities[indexA].w;
166
167     b2Vec2 cB = m_positions[indexB].c;
168     float32 aB = m_positions[indexB].a;
169     b2Vec2 vB = m_velocities[indexB].v;
170     float32 wB = m_velocities[indexB].w;
171
172     b2Assert(manifold→pointCount > 0);
173
174     b2Transform xfA, xfB;
175     xfA.q.Set(aA);
176     xfB.q.Set(aB);
177     xfA.p = cA − b2Mul(xfA.q, localCenterA);
178     xfB.p = cB − b2Mul(xfB.q, localCenterB);
179
180     b2WorldManifold worldManifold;
181     worldManifold.Initialize(manifold, xfA, radiusA, xfB, radiusB);
182
183     vc→normal = worldManifold.normal;
184
185     int32 pointCount = vc→pointCount;
186     for (int32 j = 0; j < pointCount; ++j)
187     {
188       b2VelocityConstraintPoint* vcp = vc→points + j;
189
190       vcp→rA = worldManifold.points[j] − cA;
191       vcp→rB = worldManifold.points[j] − cB;
192
193       float32 rnA = b2Cross(vcp→rA, vc→normal);
194       float32 rnB = b2Cross(vcp→rB, vc→normal);
195
```

```
196       float32 kNormal = mA + mB + iA * rnA * rnA + iB * rnB * rnB;
197
198       vcp->normalMass = kNormal > 0.0f ? 1.0f / kNormal : 0.0f;
199
200       b2Vec2 tangent = b2Cross(vc->normal, 1.0f);
201
202       float32 rtA = b2Cross(vcp->rA, tangent);
203       float32 rtB = b2Cross(vcp->rB, tangent);
204
205       float32 kTangent = mA + mB + iA * rtA * rtA + iB * rtB * rtB;
206
207       vcp->tangentMass = kTangent > 0.0f ? 1.0f /  kTangent : 0.0f;
208
209       // Setup a velocity bias for restitution.
210       vcp->velocityBias = 0.0f;
211       float32 vRel = b2Dot(vc->normal, vB + b2Cross(wB, vcp->rB) - vA - b2Cross(
    wA, vcp->rA));
212       if (vRel < -b2_velocityThreshold)
213       {
214         vcp->velocityBias = -vc->restitution * vRel;
215       }
216     }
217
218     // If we have two points, then prepare the block solver.
219     if (vc->pointCount == 2 && g_blockSolve)
220     {
221       b2VelocityConstraintPoint* vcp1 = vc->points + 0;
222       b2VelocityConstraintPoint* vcp2 = vc->points + 1;
223
224       float32 rn1A = b2Cross(vcp1->rA, vc->normal);
225       float32 rn1B = b2Cross(vcp1->rB, vc->normal);
226       float32 rn2A = b2Cross(vcp2->rA, vc->normal);
227       float32 rn2B = b2Cross(vcp2->rB, vc->normal);
228
229       float32 k11 = mA + mB + iA * rn1A * rn1A + iB * rn1B * rn1B;
230       float32 k22 = mA + mB + iA * rn2A * rn2A + iB * rn2B * rn2B;
231       float32 k12 = mA + mB + iA * rn1A * rn2A + iB * rn1B * rn2B;
232
233       // Ensure a reasonable condition number.
234       const float32 k_maxConditionNumber = 1000.0f;
235       if (k11 * k11 < k_maxConditionNumber * (k11 * k22 - k12 * k12))
236       {
237         // K is safe to invert.
238         vc->K.ex.Set(k11, k12);
239         vc->K.ey.Set(k12, k22);
240         vc->normalMass = vc->K.GetInverse();
241       }
242       else
243       {
244         // The constraints are redundant, just use one.
245
246         vc->pointCount = 1;
247       }
248     }
249   }
250 }
251
252 void b2ContactSolver::WarmStart()
253 {
254   // Warm start.
255   for (int32 i = 0; i < m_count; ++i)
256   {
257     b2ContactVelocityConstraint* vc = m_velocityConstraints + i;
258
259     int32 indexA = vc->indexA;
260     int32 indexB = vc->indexB;
```



```
261     float32 mA = vc->invMassA;
262     float32 iA = vc->invIA;
263     float32 mB = vc->invMassB;
264     float32 iB = vc->invIB;
265     int32 pointCount = vc->pointCount;
266
267     b2Vec2 vA = m_velocities[indexA].v;
268     float32 wA = m_velocities[indexA].w;
269     b2Vec2 vB = m_velocities[indexB].v;
270     float32 wB = m_velocities[indexB].w;
271
272     b2Vec2 normal = vc->normal;
273     b2Vec2 tangent = b2Cross(normal, 1.0f);
274
275     for (int32 j = 0; j < pointCount; ++j)
276     {
277       b2VelocityConstraintPoint* vcp = vc->points + j;
278       b2Vec2 P = vcp->normalImpulse * normal + vcp->tangentImpulse * tangent;
279       wA -= iA * b2Cross(vcp->rA, P);
280       vA -= mA * P;
281       wB += iB * b2Cross(vcp->rB, P);
282       vB += mB * P;
283     }
284
285     m_velocities[indexA].v = vA;
286     m_velocities[indexA].w = wA;
287     m_velocities[indexB].v = vB;
288     m_velocities[indexB].w = wB;
289   }
290 }
291
292 void b2ContactSolver::SolveVelocityConstraints()
293 {
294   for (int32 i = 0; i < m_count; ++i)
295   {
296     b2ContactVelocityConstraint* vc = m_velocityConstraints + i;
297
298     int32 indexA = vc->indexA;
299     int32 indexB = vc->indexB;
300     float32 mA = vc->invMassA;
301     float32 iA = vc->invIA;
302     float32 mB = vc->invMassB;
303     float32 iB = vc->invIB;
304     int32 pointCount = vc->pointCount;
305
306     b2Vec2 vA = m_velocities[indexA].v;
307     float32 wA = m_velocities[indexA].w;
308     b2Vec2 vB = m_velocities[indexB].v;
309     float32 wB = m_velocities[indexB].w;
310
311     b2Vec2 normal = vc->normal;
312     b2Vec2 tangent = b2Cross(normal, 1.0f);
313     float32 friction = vc->friction;
314
315     b2Assert(pointCount == 1 || pointCount == 2);
316
317     // Solve tangent constraints first because non-penetration is more important
318     // than friction.
319     for (int32 j = 0; j < pointCount; ++j)
320     {
321       b2VelocityConstraintPoint* vcp = vc->points + j;
322
323       // Relative velocity at contact
324       b2Vec2 dv = vB + b2Cross(wB, vcp->rB) - vA - b2Cross(wA, vcp->rA);
325
326       // Compute tangent force
```

```
327       float32 vt = b2Dot(dv, tangent) - vc->tangentSpeed;
328       float32 lambda = vcp->tangentMass * (-vt);
329
330       // b2Clamp the accumulated force
331       float32 maxFriction = friction * vcp->normalImpulse;
332       float32 newImpulse = b2Clamp(vcp->tangentImpulse + lambda, -maxFriction, m
    axFriction);
333       lambda = newImpulse - vcp->tangentImpulse;
334       vcp->tangentImpulse = newImpulse;
335
336       // Apply contact impulse
337       b2Vec2 P = lambda * tangent;
338
339       vA -= mA * P;
340       wA -= iA * b2Cross(vcp->rA, P);
341
342       vB += mB * P;
343       wB += iB * b2Cross(vcp->rB, P);
344     }
345
346     // Solve normal constraints
347     if (pointCount == 1 || g_blockSolve == false)
348     {
349       for (int32 j = 0; j < pointCount; ++j)
350       {
351         b2VelocityConstraintPoint* vcp = vc->points + j;
352
353         // Relative velocity at contact
354         b2Vec2 dv = vB + b2Cross(wB, vcp->rB) - vA - b2Cross(wA, vcp->rA);
355
356         // Compute normal impulse
357         float32 vn = b2Dot(dv, normal);
358         float32 lambda = -vcp->normalMass * (vn - vcp->velocityBias);
359
360         // b2Clamp the accumulated impulse
361         float32 newImpulse = b2Max(vcp->normalImpulse + lambda, 0.0f);
362         lambda = newImpulse - vcp->normalImpulse;
363         vcp->normalImpulse = newImpulse;
364
365         // Apply contact impulse
366         b2Vec2 P = lambda * normal;
367         vA -= mA * P;
368         wA -= iA * b2Cross(vcp->rA, P);
369
370         vB += mB * P;
371         wB += iB * b2Cross(vcp->rB, P);
372       }
373     }
374     else
375     {
376       // Block solver developed in collaboration with Dirk Gregorius (back in 01
    /07 on Box2D_Lite).
377       // Build the mini LCP for this contact patch
378       //
379       // vn = A * x + b, vn >= 0, x >= 0 and vn_i * x_i = 0 with i = 1..2
380       //
381       // A = J * W * JT and J = ( -n, -r1 x n, n, r2 x n )
382       // b = vn0 - velocityBias
383       //
384       // The system is solved using the "Total enumeration method" (s. Murty). T
    he complementary constraint vn_i * x_i
385       // implies that we must have in any solution either vn_i = 0 or x_i = 0. S
    o for the 2D contact problem the cases
386       // vn1 = 0 and vn2 = 0, x1 = 0 and x2 = 0, x1 = 0 and vn2 = 0, x2 = 0 and
    vn1 = 0 need to be tested. The first valid
387       // solution that satisfies the problem is chosen.
```



```
388       //
389       // In order to account of the accumulated impulse 'a' (because of the iter
    ative nature of the solver which only requires
390       // that the accumulated impulse is clamped and not the incremental impulse
    ) we change the impulse variable (x_i).
391       //
392       // Substitute:
393       //
394       // x = a + d
395       //
396       // a := old total impulse
397       // x := new total impulse
398       // d := incremental impulse
399       //
400       // For the current iteration we extend the formula for the incremental imp
    ulse
401       // to compute the new total impulse:
402       //
403       // vn = A * d + b
404       //    = A * (x - a) + b
405       //    = A * x + b - A * a
406       //    = A * x + b'
407       // b' = b - A * a;
408
409       b2VelocityConstraintPoint* cp1 = vc->points + 0;
410       b2VelocityConstraintPoint* cp2 = vc->points + 1;
411
412       b2Vec2 a(cp1->normalImpulse, cp2->normalImpulse);
413       b2Assert(a.x >= 0.0f && a.y >= 0.0f);
414
415       // Relative velocity at contact
416       b2Vec2 dv1 = vB + b2Cross(wB, cp1->rB) - vA - b2Cross(wA, cp1->rA);
417       b2Vec2 dv2 = vB + b2Cross(wB, cp2->rB) - vA - b2Cross(wA, cp2->rA);
418
419       // Compute normal velocity
420       float32 vn1 = b2Dot(dv1, normal);
421       float32 vn2 = b2Dot(dv2, normal);
422
423       b2Vec2 b;
424       b.x = vn1 - cp1->velocityBias;
425       b.y = vn2 - cp2->velocityBias;
426
427       // Compute b'
428       b -= b2Mul(vc->K, a);
429
430       const float32 k_errorTol = 1e-3f;
431       B2_NOT_USED(k_errorTol);
432
433       for (;;)
434       {
435         //
436         // Case 1: vn = 0
437         //
438         // 0 = A * x + b'
439         //
440         // Solve for x:
441         //
442         // x = - inv(A) * b'
443         //
444         b2Vec2 x = - b2Mul(vc->normalMass, b);
445
446         if (x.x >= 0.0f && x.y >= 0.0f)
447         {
448           // Get the incremental impulse
449           b2Vec2 d = x - a;
450
```

```
451             // Apply incremental impulse
452             b2Vec2 P1 = d.x * normal;
453             b2Vec2 P2 = d.y * normal;
454             vA -= mA * (P1 + P2);
455             wA -= iA * (b2Cross(cp1->rA, P1) + b2Cross(cp2->rA, P2));
456
457             vB += mB * (P1 + P2);
458             wB += iB * (b2Cross(cp1->rB, P1) + b2Cross(cp2->rB, P2));
459
460             // Accumulate
461             cp1->normalImpulse = x.x;
462             cp2->normalImpulse = x.y;
463
464 #if B2_DEBUG_SOLVER == 1
465             // Postconditions
466             dv1 = vB + b2Cross(wB, cp1->rB) - vA - b2Cross(wA, cp1->rA);
467             dv2 = vB + b2Cross(wB, cp2->rB) - vA - b2Cross(wA, cp2->rA);
468
469             // Compute normal velocity
470             vn1 = b2Dot(dv1, normal);
471             vn2 = b2Dot(dv2, normal);
472
473             b2Assert(b2Abs(vn1 - cp1->velocityBias) < k_errorTol);
474             b2Assert(b2Abs(vn2 - cp2->velocityBias) < k_errorTol);
475 #endif
476             break;
477           }
478
479           //
480           // Case 2: vn1 = 0 and x2 = 0
481           //
482           //   0 = a11 * x1 + a12 * 0 + b1'
483           // vn2 = a21 * x1 + a22 * 0 + b2'
484           //
485           x.x = - cp1->normalMass * b.x;
486           x.y = 0.0f;
487           vn1 = 0.0f;
488           vn2 = vc->K.ex.y * x.x + b.y;
489           if (x.x >= 0.0f && vn2 >= 0.0f)
490           {
491             // Get the incremental impulse
492             b2Vec2 d = x - a;
493
494             // Apply incremental impulse
495             b2Vec2 P1 = d.x * normal;
496             b2Vec2 P2 = d.y * normal;
497             vA -= mA * (P1 + P2);
498             wA -= iA * (b2Cross(cp1->rA, P1) + b2Cross(cp2->rA, P2));
499
500             vB += mB * (P1 + P2);
501             wB += iB * (b2Cross(cp1->rB, P1) + b2Cross(cp2->rB, P2));
502
503             // Accumulate
504             cp1->normalImpulse = x.x;
505             cp2->normalImpulse = x.y;
506
507 #if B2_DEBUG_SOLVER == 1
508             // Postconditions
509             dv1 = vB + b2Cross(wB, cp1->rB) - vA - b2Cross(wA, cp1->rA);
510
511             // Compute normal velocity
512             vn1 = b2Dot(dv1, normal);
513
514             b2Assert(b2Abs(vn1 - cp1->velocityBias) < k_errorTol);
515 #endif
516             break;
```



```
517           }
518
519
520           //
521           // Case 3: vn2 = 0 and x1 = 0
522           //
523           // vn1 = a11 * 0 + a12 * x2 + b1'
524           //   0 = a21 * 0 + a22 * x2 + b2'
525           //
526           x.x = 0.0f;
527           x.y = - cp2->normalMass * b.y;
528           vn1 = vc->K.ey.x * x.y + b.x;
529           vn2 = 0.0f;
530
531           if (x.y >= 0.0f && vn1 >= 0.0f)
532           {
533             // Resubstitute for the incremental impulse
534             b2Vec2 d = x - a;
535
536             // Apply incremental impulse
537             b2Vec2 P1 = d.x * normal;
538             b2Vec2 P2 = d.y * normal;
539             vA -= mA * (P1 + P2);
540             wA -= iA * (b2Cross(cp1->rA, P1) + b2Cross(cp2->rA, P2));
541
542             vB += mB * (P1 + P2);
543             wB += iB * (b2Cross(cp1->rB, P1) + b2Cross(cp2->rB, P2));
544
545             // Accumulate
546             cp1->normalImpulse = x.x;
547             cp2->normalImpulse = x.y;
548
549 #if B2_DEBUG_SOLVER == 1
550             // Postconditions
551             dv2 = vB + b2Cross(wB, cp2->rB) - vA - b2Cross(wA, cp2->rA);
552
553             // Compute normal velocity
554             vn2 = b2Dot(dv2, normal);
555
556             b2Assert(b2Abs(vn2 - cp2->velocityBias) < k_errorTol);
557 #endif
558             break;
559           }
560
561           //
562           // Case 4: x1 = 0 and x2 = 0
563           //
564           // vn1 = b1
565           // vn2 = b2;
566           x.x = 0.0f;
567           x.y = 0.0f;
568           vn1 = b.x;
569           vn2 = b.y;
570
571           if (vn1 >= 0.0f && vn2 >= 0.0f )
572           {
573             // Resubstitute for the incremental impulse
574             b2Vec2 d = x - a;
575
576             // Apply incremental impulse
577             b2Vec2 P1 = d.x * normal;
578             b2Vec2 P2 = d.y * normal;
579             vA -= mA * (P1 + P2);
580             wA -= iA * (b2Cross(cp1->rA, P1) + b2Cross(cp2->rA, P2));
581
582             vB += mB * (P1 + P2);
```

```
583             wB += iB * (b2Cross(cp1→rB, P1) + b2Cross(cp2→rB, P2));
584
585             // Accumulate
586             cp1→normalImpulse = x.x;
587             cp2→normalImpulse = x.y;
588
589             break;
590           }
591
592           // No solution, give up. This is hit sometimes, but it doesn't seem to m
    atter.
593           break;
594         }
595       }
596
597       m_velocities[indexA].v = vA;
598       m_velocities[indexA].w = wA;
599       m_velocities[indexB].v = vB;
600       m_velocities[indexB].w = wB;
601     }
602   }
603
604   void b2ContactSolver::StoreImpulses()
605   {
606     for (int32 i = 0; i < m_count; ++i)
607     {
608       b2ContactVelocityConstraint* vc = m_velocityConstraints + i;
609       b2Manifold* manifold = m_contacts[vc→contactIndex]→GetManifold();
610
611       for (int32 j = 0; j < vc→pointCount; ++j)
612       {
613         manifold→points[j].normalImpulse = vc→points[j].normalImpulse;
614         manifold→points[j].tangentImpulse = vc→points[j].tangentImpulse;
615       }
616     }
617   }
618
619   struct b2PositionSolverManifold
620   {
621     void Initialize(b2ContactPositionConstraint* pc, const b2Transform& xfA, const
     b2Transform& xfB, int32 index)
622     {
623       b2Assert(pc→pointCount > 0);
624
625       switch (pc→type)
626       {
627       case b2Manifold::e_circles:
628         {
629           b2Vec2 pointA = b2Mul(xfA, pc→localPoint);
630           b2Vec2 pointB = b2Mul(xfB, pc→localPoints[0]);
631           normal = pointB − pointA;
632           normal.Normalize();
633           point = 0.5f * (pointA + pointB);
634           separation = b2Dot(pointB − pointA, normal) − pc→radiusA − pc→radiusB;
635         }
636         break;
637
638       case b2Manifold::e_faceA:
639         {
640           normal = b2Mul(xfA.q, pc→localNormal);
641           b2Vec2 planePoint = b2Mul(xfA, pc→localPoint);
642
643           b2Vec2 clipPoint = b2Mul(xfB, pc→localPoints[index]);
644           separation = b2Dot(clipPoint − planePoint, normal) − pc→radiusA − pc→r
    adiusB;
645           point = clipPoint;
```



```
646         }
647         break;
648
649       case b2Manifold::e_faceB:
650         {
651           normal = b2Mul(xfB.q, pc→localNormal);
652           b2Vec2 planePoint = b2Mul(xfB, pc→localPoint);
653
654           b2Vec2 clipPoint = b2Mul(xfA, pc→localPoints[index]);
655           separation = b2Dot(clipPoint − planePoint, normal) − pc→radiusA − pc→r
    adiusB;
656           point = clipPoint;
657
658           // Ensure normal points from A to B
659           normal = −normal;
660         }
661         break;
662       }
663     }
664
665     b2Vec2 normal;
666     b2Vec2 point;
667     float32 separation;
668   };
669
670   // Sequential solver.
671   bool b2ContactSolver::SolvePositionConstraints()
672   {
673     float32 minSeparation = 0.0f;
674
675     for (int32 i = 0; i < m_count; ++i)
676     {
677       b2ContactPositionConstraint* pc = m_positionConstraints + i;
678
679       int32 indexA = pc→indexA;
680       int32 indexB = pc→indexB;
681       b2Vec2 localCenterA = pc→localCenterA;
682       float32 mA = pc→invMassA;
683       float32 iA = pc→invIA;
684       b2Vec2 localCenterB = pc→localCenterB;
685       float32 mB = pc→invMassB;
686       float32 iB = pc→invIB;
687       int32 pointCount = pc→pointCount;
688
689       b2Vec2 cA = m_positions[indexA].c;
690       float32 aA = m_positions[indexA].a;
691
692       b2Vec2 cB = m_positions[indexB].c;
693       float32 aB = m_positions[indexB].a;
694
695       // Solve normal constraints
696       for (int32 j = 0; j < pointCount; ++j)
697       {
698         b2Transform xfA, xfB;
699         xfA.q.Set(aA);
700         xfB.q.Set(aB);
701         xfA.p = cA − b2Mul(xfA.q, localCenterA);
702         xfB.p = cB − b2Mul(xfB.q, localCenterB);
703
704         b2PositionSolverManifold psm;
705         psm.Initialize(pc, xfA, xfB, j);
706         b2Vec2 normal = psm.normal;
707
708         b2Vec2 point = psm.point;
709         float32 separation = psm.separation;
710
```

```
711         b2Vec2 rA = point − cA;
712         b2Vec2 rB = point − cB;
713
714         // Track max constraint error.
715         minSeparation = b2Min(minSeparation, separation);
716
717         // Prevent large corrections and allow slop.
718         float32 C = b2Clamp(b2_baumgarte * (separation + b2_linearSlop), −b2_maxLi
    nearCorrection, 0.0f);
719
720         // Compute the effective mass.
721         float32 rnA = b2Cross(rA, normal);
722         float32 rnB = b2Cross(rB, normal);
723         float32 K = mA + mB + iA * rnA * rnA + iB * rnB * rnB;
724
725         // Compute normal impulse
726         float32 impulse = K > 0.0f ? − C / K : 0.0f;
727
728         b2Vec2 P = impulse * normal;
729
730         cA −= mA * P;
731         aA −= iA * b2Cross(rA, P);
732
733         cB += mB * P;
734         aB += iB * b2Cross(rB, P);
735       }
736
737       m_positions[indexA].c = cA;
738       m_positions[indexA].a = aA;
739
740       m_positions[indexB].c = cB;
741       m_positions[indexB].a = aB;
742     }
743
744     // We can't expect minSpeparation >= −b2_linearSlop because we don't
745     // push the separation above −b2_linearSlop.
746     return minSeparation ≥ −3.0f * b2_linearSlop;
747   }
748
749   // Sequential position solver for position constraints.
750   bool b2ContactSolver::SolveTOIPositionConstraints(int32 toiIndexA, int32 toiInde
    xB)
751   {
752     float32 minSeparation = 0.0f;
753
754     for (int32 i = 0; i < m_count; ++i)
755     {
756       b2ContactPositionConstraint* pc = m_positionConstraints + i;
757
758       int32 indexA = pc→indexA;
759       int32 indexB = pc→indexB;
760       b2Vec2 localCenterA = pc→localCenterA;
761       b2Vec2 localCenterB = pc→localCenterB;
762       int32 pointCount = pc→pointCount;
763
764       float32 mA = 0.0f;
765       float32 iA = 0.0f;
766       if (indexA ≡ toiIndexA ∨ indexA ≡ toiIndexB)
767       {
768         mA = pc→invMassA;
769         iA = pc→invIA;
770       }
771
772       float32 mB = 0.0f;
773       float32 iB = 0.;
774       if (indexB ≡ toiIndexA ∨ indexB ≡ toiIndexB)
```



```
775       {
776         mB = pc→invMassB;
777         iB = pc→invIB;
778       }
779
780       b2Vec2 cA = m_positions[indexA].c;
781       float32 aA = m_positions[indexA].a;
782
783       b2Vec2 cB = m_positions[indexB].c;
784       float32 aB = m_positions[indexB].a;
785
786       // Solve normal constraints
787       for (int32 j = 0; j < pointCount; ++j)
788       {
789         b2Transform xfA, xfB;
790         xfA.q.Set(aA);
791         xfB.q.Set(aB);
792         xfA.p = cA − b2Mul(xfA.q, localCenterA);
793         xfB.p = cB − b2Mul(xfB.q, localCenterB);
794
795         b2PositionSolverManifold psm;
796         psm.Initialize(pc, xfA, xfB, j);
797         b2Vec2 normal = psm.normal;
798
799         b2Vec2 point = psm.point;
800         float32 separation = psm.separation;
801
802         b2Vec2 rA = point − cA;
803         b2Vec2 rB = point − cB;
804
805         // Track max constraint error.
806         minSeparation = b2Min(minSeparation, separation);
807
808         // Prevent large corrections and allow slop.
809         float32 C = b2Clamp(b2_toiBaugarte * (separation + b2_linearSlop), −b2_max
    LinearCorrection, 0.0f);
810
811         // Compute the effective mass.
812         float32 rnA = b2Cross(rA, normal);
813         float32 rnB = b2Cross(rB, normal);
814         float32 K = mA + mB + iA * rnA * rnA + iB * rnB * rnB;
815
816         // Compute normal impulse
817         float32 impulse = K > 0.0f ? − C / K : 0.0f;
818
819         b2Vec2 P = impulse * normal;
820
821         cA −= mA * P;
822         aA −= iA * b2Cross(rA, P);
823
824         cB += mB * P;
825         aB += iB * b2Cross(rB, P);
826       }
827
828       m_positions[indexA].c = cA;
829       m_positions[indexA].a = aA;
830
831       m_positions[indexB].c = cB;
832       m_positions[indexB].a = aB;
833     }
834
835     // We can't expect minSpeparation >= −b2_linearSlop because we don't
836     // push the separation above −b2_linearSlop.
837     return minSeparation ≥ −1.5f * b2_linearSlop;
838   }
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Contacts/b2Contact.h"
#include "Box2D/Dynamics/Contacts/b2CircleContact.h"
#include "Box2D/Dynamics/Contacts/b2PolygonAndCircleContact.h"
#include "Box2D/Dynamics/Contacts/b2PolygonContact.h"
#include "Box2D/Dynamics/Contacts/b2EdgeAndCircleContact.h"
#include "Box2D/Dynamics/Contacts/b2EdgeAndPolygonContact.h"
#include "Box2D/Dynamics/Contacts/b2ChainAndCircleContact.h"
#include "Box2D/Dynamics/Contacts/b2ChainAndPolygonContact.h"
#include "Box2D/Dynamics/Contacts/b2ContactSolver.h"

#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Collision/b2TimeOfImpact.h"
#include "Box2D/Collision/Shapes/b2Shape.h"
#include "Box2D/Common/b2BlockAllocator.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2Fixture.h"
#include "Box2D/Dynamics/b2World.h"

b2ContactRegister b2Contact::s_registers[b2Shape::e_typeCount][b2Shape::e_typeCount];
bool b2Contact::s_initialized = false;

void b2Contact::InitializeRegisters()
{
  AddType(b2CircleContact::Create, b2CircleContact::Destroy, b2Shape::e_circle, b2Shape::e_circle);
  AddType(b2PolygonAndCircleContact::Create, b2PolygonAndCircleContact::Destroy, b2Shape::e_polygon, b2Shape::e_circle);
  AddType(b2PolygonContact::Create, b2PolygonContact::Destroy, b2Shape::e_polygon, b2Shape::e_polygon);
  AddType(b2EdgeAndCircleContact::Create, b2EdgeAndCircleContact::Destroy, b2Shape::e_edge, b2Shape::e_circle);
  AddType(b2EdgeAndPolygonContact::Create, b2EdgeAndPolygonContact::Destroy, b2Shape::e_edge, b2Shape::e_polygon);
  AddType(b2ChainAndCircleContact::Create, b2ChainAndCircleContact::Destroy, b2Shape::e_chain, b2Shape::e_circle);
  AddType(b2ChainAndPolygonContact::Create, b2ChainAndPolygonContact::Destroy, b2Shape::e_chain, b2Shape::e_polygon);
}

void b2Contact::AddType(b2ContactCreateFcn* createFcn, b2ContactDestroyFcn* destoryFcn,
                        b2Shape::Type type1, b2Shape::Type type2)
{
  b2Assert(0 ≤ type1 ∧ type1 < b2Shape::e_typeCount);
  b2Assert(0 ≤ type2 ∧ type2 < b2Shape::e_typeCount);

  s_registers[type1][type2].createFcn = createFcn;
```



```
  s_registers[type1][type2].destroyFcn = destoryFcn;
  s_registers[type1][type2].primary = true;

  if (type1 ≠ type2)
  {
    s_registers[type2][type1].createFcn = createFcn;
    s_registers[type2][type1].destroyFcn = destoryFcn;
    s_registers[type2][type1].primary = false;
  }
}

b2Contact* b2Contact::Create(b2Fixture* fixtureA, int32 indexA, b2Fixture* fixtureB, int32 indexB, b2BlockAllocator* allocator)
{
  if (s_initialized ≡ false)
  {
    InitializeRegisters();
    s_initialized = true;
  }

  b2Shape::Type type1 = fixtureA→GetType();
  b2Shape::Type type2 = fixtureB→GetType();

  b2Assert(0 ≤ type1 ∧ type1 < b2Shape::e_typeCount);
  b2Assert(0 ≤ type2 ∧ type2 < b2Shape::e_typeCount);

  b2ContactCreateFcn* createFcn = s_registers[type1][type2].createFcn;
  if (createFcn)
  {
    if (s_registers[type1][type2].primary)
    {
      return createFcn(fixtureA, indexA, fixtureB, indexB, allocator);
    }
    else
    {
      return createFcn(fixtureB, indexB, fixtureA, indexA, allocator);
    }
  }
  else
  {
    return nullptr;
  }
}

void b2Contact::Destroy(b2Contact* contact, b2BlockAllocator* allocator)
{
  b2Assert(s_initialized ≡ true);

  b2Fixture* fixtureA = contact→m_fixtureA;
  b2Fixture* fixtureB = contact→m_fixtureB;

  if (contact→m_manifold.pointCount > 0 ∧
    fixtureA→IsSensor() ≡ false ∧
    fixtureB→IsSensor() ≡ false)
  {
    fixtureA→GetBody()→SetAwake(true);
    fixtureB→GetBody()→SetAwake(true);
  }

  b2Shape::Type typeA = fixtureA→GetType();
  b2Shape::Type typeB = fixtureB→GetType();

  b2Assert(0 ≤ typeA ∧ typeB < b2Shape::e_typeCount);
  b2Assert(0 ≤ typeA ∧ typeB < b2Shape::e_typeCount);

  b2ContactDestroyFcn* destroyFcn = s_registers[typeA][typeB].destroyFcn;
```

```
123   destroyFcn(contact, allocator);
124 }
125
126 b2Contact::b2Contact(b2Fixture* fA, int32 indexA, b2Fixture* fB, int32 indexB)
127 {
128   m_flags = e_enabledFlag;
129
130   m_fixtureA = fA;
131   m_fixtureB = fB;
132
133   m_indexA = indexA;
134   m_indexB = indexB;
135
136   m_manifold.pointCount = 0;
137
138   m_prev = nullptr;
139   m_next = nullptr;
140
141   m_nodeA.contact = nullptr;
142   m_nodeA.prev = nullptr;
143   m_nodeA.next = nullptr;
144   m_nodeA.other = nullptr;
145
146   m_nodeB.contact = nullptr;
147   m_nodeB.prev = nullptr;
148   m_nodeB.next = nullptr;
149   m_nodeB.other = nullptr;
150
151   m_toiCount = 0;
152
153   m_friction = b2MixFriction(m_fixtureA→m_friction, m_fixtureB→m_friction);
154   m_restitution = b2MixRestitution(m_fixtureA→m_restitution, m_fixtureB→m_rest
    itution);
155
156   m_tangentSpeed = 0.0f;
157 }
158
159 // Update the contact manifold and touching status.
160 // Note: do not assume the fixture AABBs are overlapping or are valid.
161 void b2Contact::Update(b2ContactListener* listener)
162 {
163   b2Manifold oldManifold = m_manifold;
164
165   // Re-enable this contact.
166   m_flags |= e_enabledFlag;
167
168   bool touching = false;
169   bool wasTouching = (m_flags & e_touchingFlag) ≡ e_touchingFlag;
170
171   bool sensorA = m_fixtureA→IsSensor();
172   bool sensorB = m_fixtureB→IsSensor();
173   bool sensor = sensorA ∨ sensorB;
174
175   b2Body* bodyA = m_fixtureA→GetBody();
176   b2Body* bodyB = m_fixtureB→GetBody();
177   const b2Transform& xfA = bodyA→GetTransform();
178   const b2Transform& xfB = bodyB→GetTransform();
179
180   // Is this contact a sensor?
181   if (sensor)
182   {
183     const b2Shape* shapeA = m_fixtureA→GetShape();
184     const b2Shape* shapeB = m_fixtureB→GetShape();
185     touching = b2TestOverlap(shapeA, m_indexA, shapeB, m_indexB, xfA, xfB);
186
187     // Sensors don't generate manifolds.
```



```
188     m_manifold.pointCount = 0;
189   }
190   else
191   {
192     Evaluate(&m_manifold, xfA, xfB);
193     touching = m_manifold.pointCount > 0;
194
195     // Match old contact ids to new contact ids and copy the
196     // stored impulses to warm start the solver.
197     for (int32 i = 0; i < m_manifold.pointCount; ++i)
198     {
199       b2ManifoldPoint* mp2 = m_manifold.points + i;
200       mp2→normalImpulse = 0.0f;
201       mp2→tangentImpulse = 0.0f;
202       b2ContactID id2 = mp2→id;
203
204       for (int32 j = 0; j < oldManifold.pointCount; ++j)
205       {
206         b2ManifoldPoint* mp1 = oldManifold.points + j;
207
208         if (mp1→id.key ≡ id2.key)
209         {
210           mp2→normalImpulse = mp1→normalImpulse;
211           mp2→tangentImpulse = mp1→tangentImpulse;
212           break;
213         }
214       }
215     }
216
217     if (touching ≠ wasTouching)
218     {
219       bodyA→SetAwake(true);
220       bodyB→SetAwake(true);
221     }
222   }
223
224   if (touching)
225   {
226     m_flags |= e_touchingFlag;
227   }
228   else
229   {
230     m_flags &= ~e_touchingFlag;
231   }
232
233   if (wasTouching ≡ false ∧ touching ≡ true ∧ listener)
234   {
235     listener→BeginContact(this);
236   }
237
238   if (wasTouching ≡ true ∧ touching ≡ false ∧ listener)
239   {
240     listener→EndContact(this);
241   }
242
243   if (sensor ≡ false ∧ touching ∧ listener)
244   {
245     listener→PreSolve(this, &oldManifold);
246   }
247 }
```

nov 26, 19 17:34 **b2CircleContact.cpp** Page 1/1

```cpp
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Contacts/b2CircleContact.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2Fixture.h"
#include "Box2D/Dynamics/b2WorldCallbacks.h"
#include "Box2D/Common/b2BlockAllocator.h"
#include "Box2D/Collision/b2TimeOfImpact.h"

#include <new>

b2Contact* b2CircleContact::Create(b2Fixture* fixtureA, int32, b2Fixture* fixtureB, int32, b2BlockAllocator* allocator)
{
  void* mem = allocator->Allocate(sizeof(b2CircleContact));
  return new (mem) b2CircleContact(fixtureA, fixtureB);
}

void b2CircleContact::Destroy(b2Contact* contact, b2BlockAllocator* allocator)
{
  ((b2CircleContact*)contact)->~b2CircleContact();
  allocator->Free(contact, sizeof(b2CircleContact));
}

b2CircleContact::b2CircleContact(b2Fixture* fixtureA, b2Fixture* fixtureB)
  : b2Contact(fixtureA, 0, fixtureB, 0)
{
  b2Assert(m_fixtureA->GetType() == b2Shape::e_circle);
  b2Assert(m_fixtureB->GetType() == b2Shape::e_circle);
}

void b2CircleContact::Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform& xfB)
{
  b2CollideCircles(manifold,
          (b2CircleShape*)m_fixtureA->GetShape(), xfA,
          (b2CircleShape*)m_fixtureB->GetShape(), xfB);
}
```

nov 26, 19 17:34 **b2ChainAndPolygonContact.cpp** Page 1/1

```cpp
/*
* Copyright (c) 2006-2010 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Contacts/b2ChainAndPolygonContact.h"
#include "Box2D/Common/b2BlockAllocator.h"
#include "Box2D/Dynamics/b2Fixture.h"
#include "Box2D/Collision/Shapes/b2ChainShape.h"
#include "Box2D/Collision/Shapes/b2EdgeShape.h"

#include <new>

b2Contact* b2ChainAndPolygonContact::Create(b2Fixture* fixtureA, int32 indexA, b2Fixture* fixtureB, int32 indexB, b2BlockAllocator* allocator)
{
  void* mem = allocator->Allocate(sizeof(b2ChainAndPolygonContact));
  return new (mem) b2ChainAndPolygonContact(fixtureA, indexA, fixtureB, indexB);
}

void b2ChainAndPolygonContact::Destroy(b2Contact* contact, b2BlockAllocator* allocator)
{
  ((b2ChainAndPolygonContact*)contact)->~b2ChainAndPolygonContact();
  allocator->Free(contact, sizeof(b2ChainAndPolygonContact));
}

b2ChainAndPolygonContact::b2ChainAndPolygonContact(b2Fixture* fixtureA, int32 indexA, b2Fixture* fixtureB, int32 indexB)
: b2Contact(fixtureA, indexA, fixtureB, indexB)
{
  b2Assert(m_fixtureA->GetType() == b2Shape::e_chain);
  b2Assert(m_fixtureB->GetType() == b2Shape::e_polygon);
}

void b2ChainAndPolygonContact::Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform& xfB)
{
  b2ChainShape* chain = (b2ChainShape*)m_fixtureA->GetShape();
  b2EdgeShape edge;
  chain->GetChildEdge(&edge, m_indexA);
  b2CollideEdgeAndPolygon(  manifold, &edge, xfA,
                (b2PolygonShape*)m_fixtureB->GetShape(), xfB);
}
```

```cpp
/*
* Copyright (c) 2006-2010 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/Contacts/b2ChainAndCircleContact.h"
#include "Box2D/Common/b2BlockAllocator.h"
#include "Box2D/Dynamics/b2Fixture.h"
#include "Box2D/Collision/Shapes/b2ChainShape.h"
#include "Box2D/Collision/Shapes/b2EdgeShape.h"

#include <new>

b2Contact* b2ChainAndCircleContact::Create(b2Fixture* fixtureA, int32 indexA, b2Fixture* fixtureB, int32 indexB, b2BlockAllocator* allocator)
{
  void* mem = allocator→Allocate(sizeof(b2ChainAndCircleContact));
  return new (mem) b2ChainAndCircleContact(fixtureA, indexA, fixtureB, indexB);
}

void b2ChainAndCircleContact::Destroy(b2Contact* contact, b2BlockAllocator* allocator)
{
  ((b2ChainAndCircleContact*)contact)→~b2ChainAndCircleContact();
  allocator→Free(contact, sizeof(b2ChainAndCircleContact));
}

b2ChainAndCircleContact::b2ChainAndCircleContact(b2Fixture* fixtureA, int32 indexA, b2Fixture* fixtureB, int32 indexB)
: b2Contact(fixtureA, indexA, fixtureB, indexB)
{
  b2Assert(m_fixtureA→GetType() ≡ b2Shape::e_chain);
  b2Assert(m_fixtureB→GetType() ≡ b2Shape::e_circle);
}

void b2ChainAndCircleContact::Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform& xfB)
{
  b2ChainShape* chain = (b2ChainShape*)m_fixtureA→GetShape();
  b2EdgeShape edge;
  chain→GetChildEdge(&edge, m_indexA);
  b2CollideEdgeAndCircle( manifold, &edge, xfA,
              (b2CircleShape*)m_fixtureB→GetShape(), xfB);
}
```



```cpp
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/b2World.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2Fixture.h"
#include "Box2D/Dynamics/b2Island.h"
#include "Box2D/Dynamics/Joints/b2PulleyJoint.h"
#include "Box2D/Dynamics/Contacts/b2Contact.h"
#include "Box2D/Dynamics/Contacts/b2ContactSolver.h"
#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Collision/b2BroadPhase.h"
#include "Box2D/Collision/Shapes/b2CircleShape.h"
#include "Box2D/Collision/Shapes/b2EdgeShape.h"
#include "Box2D/Collision/Shapes/b2ChainShape.h"
#include "Box2D/Collision/Shapes/b2PolygonShape.h"
#include "Box2D/Collision/b2TimeOfImpact.h"
#include "Box2D/Common/b2Draw.h"
#include "Box2D/Common/b2Timer.h"
#include <new>

b2World::b2World(const b2Vec2& gravity)
{
  m_destructionListener = nullptr;
  m_debugDraw = nullptr;

  m_bodyList = nullptr;
  m_jointList = nullptr;

  m_bodyCount = 0;
  m_jointCount = 0;

  m_warmStarting = true;
  m_continuousPhysics = true;
  m_subStepping = false;

  m_stepComplete = true;

  m_allowSleep = true;
  m_gravity = gravity;

  m_flags = e_clearForces;

  m_inv_dt0 = 0.0f;

  m_contactManager.m_allocator = &m_blockAllocator;

  memset(&m_profile, 0, sizeof(b2Profile));
}

b2World::~b2World()
```

```
67  {
68    // Some shapes allocate using b2Alloc.
69    b2Body* b = m_bodyList;
70    while (b)
71    {
72      b2Body* bNext = b→m_next;
73
74      b2Fixture* f = b→m_fixtureList;
75      while (f)
76      {
77        b2Fixture* fNext = f→m_next;
78        f→m_proxyCount = 0;
79        f→Destroy(&m_blockAllocator);
80        f = fNext;
81      }
82
83      b = bNext;
84    }
85  }
86
87  void b2World::SetDestructionListener(b2DestructionListener* listener)
88  {
89    m_destructionListener = listener;
90  }
91
92  void b2World::SetContactFilter(b2ContactFilter* filter)
93  {
94    m_contactManager.m_contactFilter = filter;
95  }
96
97  void b2World::SetContactListener(b2ContactListener* listener)
98  {
99    m_contactManager.m_contactListener = listener;
100 }
101
102 void b2World::SetDebugDraw(b2Draw* debugDraw)
103 {
104   m_debugDraw = debugDraw;
105 }
106
107 b2Body* b2World::CreateBody(const b2BodyDef* def)
108 {
109   b2Assert(IsLocked() ≡ false);
110   if (IsLocked())
111   {
112     return nullptr;
113   }
114
115   void* mem = m_blockAllocator.Allocate(sizeof(b2Body));
116   b2Body* b = new (mem) b2Body(def, this);
117
118   // Add to world doubly linked list.
119   b→m_prev = nullptr;
120   b→m_next = m_bodyList;
121   if (m_bodyList)
122   {
123     m_bodyList→m_prev = b;
124   }
125   m_bodyList = b;
126   ++m_bodyCount;
127
128   return b;
129 }
130
131 void b2World::DestroyBody(b2Body* b)
132 {
```



```
133   b2Assert(m_bodyCount > 0);
134   b2Assert(IsLocked() ≡ false);
135   if (IsLocked())
136   {
137     return;
138   }
139
140   // Delete the attached joints.
141   b2JointEdge* je = b→m_jointList;
142   while (je)
143   {
144     b2JointEdge* je0 = je;
145     je = je→next;
146
147     if (m_destructionListener)
148     {
149       m_destructionListener→SayGoodbye(je0→joint);
150     }
151
152     DestroyJoint(je0→joint);
153
154     b→m_jointList = je;
155   }
156   b→m_jointList = nullptr;
157
158   // Delete the attached contacts.
159   b2ContactEdge* ce = b→m_contactList;
160   while (ce)
161   {
162     b2ContactEdge* ce0 = ce;
163     ce = ce→next;
164     m_contactManager.Destroy(ce0→contact);
165   }
166   b→m_contactList = nullptr;
167
168   // Delete the attached fixtures. This destroys broad-phase proxies.
169   b2Fixture* f = b→m_fixtureList;
170   while (f)
171   {
172     b2Fixture* f0 = f;
173     f = f→m_next;
174
175     if (m_destructionListener)
176     {
177       m_destructionListener→SayGoodbye(f0);
178     }
179
180     f0→DestroyProxies(&m_contactManager.m_broadPhase);
181     f0→Destroy(&m_blockAllocator);
182     f0→~b2Fixture();
183     m_blockAllocator.Free(f0, sizeof(b2Fixture));
184
185     b→m_fixtureList = f;
186     b→m_fixtureCount -= 1;
187   }
188   b→m_fixtureList = nullptr;
189   b→m_fixtureCount = 0;
190
191   // Remove world body list.
192   if (b→m_prev)
193   {
194     b→m_prev→m_next = b→m_next;
195   }
196
197   if (b→m_next)
198   {
```

```
199       b→m_next→m_prev = b→m_prev;
200     }
201
202     if (b ≡ m_bodyList)
203     {
204       m_bodyList = b→m_next;
205     }
206
207     −−m_bodyCount;
208     b→~b2Body();
209     m_blockAllocator.Free(b, sizeof(b2Body));
210   }
211
212   b2Joint* b2World::CreateJoint(const b2JointDef* def)
213   {
214     b2Assert(IsLocked() ≡ false);
215     if (IsLocked())
216     {
217       return nullptr;
218     }
219
220     b2Joint* j = b2Joint::Create(def, &m_blockAllocator);
221
222     // Connect to the world list.
223     j→m_prev = nullptr;
224     j→m_next = m_jointList;
225     if (m_jointList)
226     {
227       m_jointList→m_prev = j;
228     }
229     m_jointList = j;
230     ++m_jointCount;
231
232     // Connect to the bodies' doubly linked lists.
233     j→m_edgeA.joint = j;
234     j→m_edgeA.other = j→m_bodyB;
235     j→m_edgeA.prev = nullptr;
236     j→m_edgeA.next = j→m_bodyA→m_jointList;
237     if (j→m_bodyA→m_jointList) j→m_bodyA→m_jointList→prev = &j→m_edgeA;
238     j→m_bodyA→m_jointList = &j→m_edgeA;
239
240     j→m_edgeB.joint = j;
241     j→m_edgeB.other = j→m_bodyA;
242     j→m_edgeB.prev = nullptr;
243     j→m_edgeB.next = j→m_bodyB→m_jointList;
244     if (j→m_bodyB→m_jointList) j→m_bodyB→m_jointList→prev = &j→m_edgeB;
245     j→m_bodyB→m_jointList = &j→m_edgeB;
246
247     b2Body* bodyA = def→bodyA;
248     b2Body* bodyB = def→bodyB;
249
250     // If the joint prevents collisions, then flag any contacts for filtering.
251     if (def→collideConnected ≡ false)
252     {
253       b2ContactEdge* edge = bodyB→GetContactList();
254       while (edge)
255       {
256         if (edge→other ≡ bodyA)
257         {
258           // Flag the contact for filtering at the next time step (where either
259           // body is awake).
260           edge→contact→FlagForFiltering();
261         }
262
263         edge = edge→next;
264       }
```



```
265     }
266
267     // Note: creating a joint doesn't wake the bodies.
268
269     return j;
270   }
271
272   void b2World::DestroyJoint(b2Joint* j)
273   {
274     b2Assert(IsLocked() ≡ false);
275     if (IsLocked())
276     {
277       return;
278     }
279
280     bool collideConnected = j→m_collideConnected;
281
282     // Remove from the doubly linked list.
283     if (j→m_prev)
284     {
285       j→m_prev→m_next = j→m_next;
286     }
287
288     if (j→m_next)
289     {
290       j→m_next→m_prev = j→m_prev;
291     }
292
293     if (j ≡ m_jointList)
294     {
295       m_jointList = j→m_next;
296     }
297
298     // Disconnect from island graph.
299     b2Body* bodyA = j→m_bodyA;
300     b2Body* bodyB = j→m_bodyB;
301
302     // Wake up connected bodies.
303     bodyA→SetAwake(true);
304     bodyB→SetAwake(true);
305
306     // Remove from body 1.
307     if (j→m_edgeA.prev)
308     {
309       j→m_edgeA.prev→next = j→m_edgeA.next;
310     }
311
312     if (j→m_edgeA.next)
313     {
314       j→m_edgeA.next→prev = j→m_edgeA.prev;
315     }
316
317     if (&j→m_edgeA ≡ bodyA→m_jointList)
318     {
319       bodyA→m_jointList = j→m_edgeA.next;
320     }
321
322     j→m_edgeA.prev = nullptr;
323     j→m_edgeA.next = nullptr;
324
325     // Remove from body 2
326     if (j→m_edgeB.prev)
327     {
328       j→m_edgeB.prev→next = j→m_edgeB.next;
329     }
330
```

```
331   if (j→m_edgeB.next)
332   {
333     j→m_edgeB.next→prev = j→m_edgeB.prev;
334   }
335
336   if (&j→m_edgeB ≡ bodyB→m_jointList)
337   {
338     bodyB→m_jointList = j→m_edgeB.next;
339   }
340
341   j→m_edgeB.prev = nullptr;
342   j→m_edgeB.next = nullptr;
343
344   b2Joint::Destroy(j, &m_blockAllocator);
345
346   b2Assert(m_jointCount > 0);
347   --m_jointCount;
348
349   // If the joint prevents collisions, then flag any contacts for filtering.
350   if (collideConnected ≡ false)
351   {
352     b2ContactEdge* edge = bodyB→GetContactList();
353     while (edge)
354     {
355       if (edge→other ≡ bodyA)
356       {
357         // Flag the contact for filtering at the next time step (where either
358         // body is awake).
359         edge→contact→FlagForFiltering();
360       }
361
362       edge = edge→next;
363     }
364   }
365 }
366
367 //
368 void b2World::SetAllowSleeping(bool flag)
369 {
370   if (flag ≡ m_allowSleep)
371   {
372     return;
373   }
374
375   m_allowSleep = flag;
376   if (m_allowSleep ≡ false)
377   {
378     for (b2Body* b = m_bodyList; b; b = b→m_next)
379     {
380       b→SetAwake(true);
381     }
382   }
383 }
384
385 // Find islands, integrate and solve constraints, solve position constraints
386 void b2World::Solve(const b2TimeStep& step)
387 {
388   m_profile.solveInit = 0.0f;
389   m_profile.solveVelocity = 0.0f;
390   m_profile.solvePosition = 0.0f;
391
392   // Size the island for the worst case.
393   b2Island island(m_bodyCount,
394           m_contactManager.m_contactCount,
395           m_jointCount,
396           &m_stackAllocator,
```



```
397           m_contactManager.m_contactListener);
398
399   // Clear all the island flags.
400   for (b2Body* b = m_bodyList; b; b = b→m_next)
401   {
402     b→m_flags &= ~b2Body::e_islandFlag;
403   }
404   for (b2Contact* c = m_contactManager.m_contactList; c; c = c→m_next)
405   {
406     c→m_flags &= ~b2Contact::e_islandFlag;
407   }
408   for (b2Joint* j = m_jointList; j; j = j→m_next)
409   {
410     j→m_islandFlag = false;
411   }
412
413   // Build and simulate all awake islands.
414   int32 stackSize = m_bodyCount;
415   b2Body** stack = (b2Body**)m_stackAllocator.Allocate(stackSize * sizeof(b2Body
    *));
416   for (b2Body* seed = m_bodyList; seed; seed = seed→m_next)
417   {
418     if (seed→m_flags & b2Body::e_islandFlag)
419     {
420       continue;
421     }
422
423     if (seed→IsAwake() ≡ false ∨ seed→IsActive() ≡ false)
424     {
425       continue;
426     }
427
428     // The seed can be dynamic or kinematic.
429     if (seed→GetType() ≡ b2_staticBody)
430     {
431       continue;
432     }
433
434     // Reset island and stack.
435     island.Clear();
436     int32 stackCount = 0;
437     stack[stackCount++] = seed;
438     seed→m_flags |= b2Body::e_islandFlag;
439
440     // Perform a depth first search (DFS) on the constraint graph.
441     while (stackCount > 0)
442     {
443       // Grab the next body off the stack and add it to the island.
444       b2Body* b = stack[--stackCount];
445       b2Assert(b→IsActive() ≡ true);
446       island.Add(b);
447
448       // Make sure the body is awake (without resetting sleep timer).
449       b→m_flags |= b2Body::e_awakeFlag;
450
451       // To keep islands as small as possible, we don't
452       // propagate islands across static bodies.
453       if (b→GetType() ≡ b2_staticBody)
454       {
455         continue;
456       }
457
458       // Search all contacts connected to this body.
459       for (b2ContactEdge* ce = b→m_contactList; ce; ce = ce→next)
460       {
461         b2Contact* contact = ce→contact;
```

```
462
463           // Has this contact already been added to an island?
464           if (contact→m_flags & b2Contact::e_islandFlag)
465           {
466             continue;
467           }
468
469           // Is this contact solid and touching?
470           if (contact→IsEnabled() ≡ false ∨
471             contact→IsTouching() ≡ false)
472           {
473             continue;
474           }
475
476           // Skip sensors.
477           bool sensorA = contact→m_fixtureA→m_isSensor;
478           bool sensorB = contact→m_fixtureB→m_isSensor;
479           if (sensorA ∨ sensorB)
480           {
481             continue;
482           }
483
484           island.Add(contact);
485           contact→m_flags |= b2Contact::e_islandFlag;
486
487           b2Body* other = ce→other;
488
489           // Was the other body already added to this island?
490           if (other→m_flags & b2Body::e_islandFlag)
491           {
492             continue;
493           }
494
495           b2Assert(stackCount < stackSize);
496           stack[stackCount++] = other;
497           other→m_flags |= b2Body::e_islandFlag;
498         }
499
500         // Search all joints connect to this body.
501         for (b2JointEdge* je = b→m_jointList; je; je = je→next)
502         {
503           if (je→joint→m_islandFlag ≡ true)
504           {
505             continue;
506           }
507
508           b2Body* other = je→other;
509
510           // Don't simulate joints connected to inactive bodies.
511           if (other→IsActive() ≡ false)
512           {
513             continue;
514           }
515
516           island.Add(je→joint);
517           je→joint→m_islandFlag = true;
518
519           if (other→m_flags & b2Body::e_islandFlag)
520           {
521             continue;
522           }
523
524           b2Assert(stackCount < stackSize);
525           stack[stackCount++] = other;
526           other→m_flags |= b2Body::e_islandFlag;
527         }
```



```
528       }
529
530       b2Profile profile;
531       island.Solve(&profile, step, m_gravity, m_allowSleep);
532       m_profile.solveInit += profile.solveInit;
533       m_profile.solveVelocity += profile.solveVelocity;
534       m_profile.solvePosition += profile.solvePosition;
535
536       // Post solve cleanup.
537       for (int32 i = 0; i < island.m_bodyCount; ++i)
538       {
539         // Allow static bodies to participate in other islands.
540         b2Body* b = island.m_bodies[i];
541         if (b→GetType() ≡ b2_staticBody)
542         {
543           b→m_flags &= ~b2Body::e_islandFlag;
544         }
545       }
546     }
547
548     m_stackAllocator.Free(stack);
549
550     {
551       b2Timer timer;
552       // Synchronize fixtures, check for out of range bodies.
553       for (b2Body* b = m_bodyList; b; b = b→GetNext())
554       {
555         // If a body was not in an island then it did not move.
556         if ((b→m_flags & b2Body::e_islandFlag) ≡ 0)
557         {
558           continue;
559         }
560
561         if (b→GetType() ≡ b2_staticBody)
562         {
563           continue;
564         }
565
566         // Update fixtures (for broad-phase).
567         b→SynchronizeFixtures();
568       }
569
570       // Look for new contacts.
571       m_contactManager.FindNewContacts();
572       m_profile.broadphase = timer.GetMilliseconds();
573     }
574   }
575
576   // Find TOI contacts and solve them.
577   void b2World::SolveTOI(const b2TimeStep& step)
578   {
579     b2Island island(2 * b2_maxTOIContacts, b2_maxTOIContacts, 0, &m_stackAllocator
      , m_contactManager.m_contactListener);
580
581     if (m_stepComplete)
582     {
583       for (b2Body* b = m_bodyList; b; b = b→m_next)
584       {
585         b→m_flags &= ~b2Body::e_islandFlag;
586         b→m_sweep.alpha0 = 0.0f;
587       }
588
589       for (b2Contact* c = m_contactManager.m_contactList; c; c = c→m_next)
590       {
591         // Invalidate TOI
592         c→m_flags &= ~(b2Contact::e_toiFlag | b2Contact::e_islandFlag);
```

```
593         c→m_toiCount = 0;
594         c→m_toi = 1.0f;
595       }
596     }
597
598     // Find TOI events and solve them.
599     for (;;)
600     {
601       // Find the first TOI.
602       b2Contact* minContact = nullptr;
603       float32 minAlpha = 1.0f;
604
605       for (b2Contact* c = m_contactManager.m_contactList; c; c = c→m_next)
606       {
607         // Is this contact disabled?
608         if (c→IsEnabled() ≡ false)
609         {
610           continue;
611         }
612
613         // Prevent excessive sub−stepping.
614         if (c→m_toiCount > b2_maxSubSteps)
615         {
616           continue;
617         }
618
619         float32 alpha = 1.0f;
620         if (c→m_flags & b2Contact::e_toiFlag)
621         {
622           // This contact has a valid cached TOI.
623           alpha = c→m_toi;
624         }
625         else
626         {
627           b2Fixture* fA = c→GetFixtureA();
628           b2Fixture* fB = c→GetFixtureB();
629
630           // Is there a sensor?
631           if (fA→IsSensor() ∨ fB→IsSensor())
632           {
633             continue;
634           }
635
636           b2Body* bA = fA→GetBody();
637           b2Body* bB = fB→GetBody();
638
639           b2BodyType typeA = bA→m_type;
640           b2BodyType typeB = bB→m_type;
641           b2Assert(typeA ≡ b2_dynamicBody ∨ typeB ≡ b2_dynamicBody);
642
643           bool activeA = bA→IsAwake() ∧ typeA ≠ b2_staticBody;
644           bool activeB = bB→IsAwake() ∧ typeB ≠ b2_staticBody;
645
646           // Is at least one body active (awake and dynamic or kinematic)?
647           if (activeA ≡ false ∧ activeB ≡ false)
648           {
649             continue;
650           }
651
652           bool collideA = bA→IsBullet() ∨ typeA ≠ b2_dynamicBody;
653           bool collideB = bB→IsBullet() ∨ typeB ≠ b2_dynamicBody;
654
655           // Are these two non−bullet dynamic bodies?
656           if (collideA ≡ false ∧ collideB ≡ false)
657           {
658             continue;
```



```
659           }
660
661           // Compute the TOI for this contact.
662           // Put the sweeps onto the same time interval.
663           float32 alpha0 = bA→m_sweep.alpha0;
664
665           if (bA→m_sweep.alpha0 < bB→m_sweep.alpha0)
666           {
667             alpha0 = bB→m_sweep.alpha0;
668             bA→m_sweep.Advance(alpha0);
669           }
670           else if (bB→m_sweep.alpha0 < bA→m_sweep.alpha0)
671           {
672             alpha0 = bA→m_sweep.alpha0;
673             bB→m_sweep.Advance(alpha0);
674           }
675
676           b2Assert(alpha0 < 1.0f);
677
678           int32 indexA = c→GetChildIndexA();
679           int32 indexB = c→GetChildIndexB();
680
681           // Compute the time of impact in interval [0, minTOI]
682           b2TOIInput input;
683           input.proxyA.Set(fA→GetShape(), indexA);
684           input.proxyB.Set(fB→GetShape(), indexB);
685           input.sweepA = bA→m_sweep;
686           input.sweepB = bB→m_sweep;
687           input.tMax = 1.0f;
688
689           b2TOIOutput output;
690           b2TimeOfImpact(&output, &input);
691
692           // Beta is the fraction of the remaining portion of the .
693           float32 beta = output.t;
694           if (output.state ≡ b2TOIOutput::e_touching)
695           {
696             alpha = b2Min(alpha0 + (1.0f − alpha0) * beta, 1.0f);
697           }
698           else
699           {
700             alpha = 1.0f;
701           }
702
703           c→m_toi = alpha;
704           c→m_flags |= b2Contact::e_toiFlag;
705         }
706
707         if (alpha < minAlpha)
708         {
709           // This is the minimum TOI found so far.
710           minContact = c;
711           minAlpha = alpha;
712         }
713       }
714
715       if (minContact ≡ nullptr ∨ 1.0f − 10.0f * b2_epsilon < minAlpha)
716       {
717         // No more TOI events. Done!
718         m_stepComplete = true;
719         break;
720       }
721
722       // Advance the bodies to the TOI.
723       b2Fixture* fA = minContact→GetFixtureA();
724       b2Fixture* fB = minContact→GetFixtureB();
```

```
725       b2Body* bA = fA→GetBody();
726       b2Body* bB = fB→GetBody();
727
728       b2Sweep backup1 = bA→m_sweep;
729       b2Sweep backup2 = bB→m_sweep;
730
731       bA→Advance(minAlpha);
732       bB→Advance(minAlpha);
733
734       // The TOI contact likely has some new contact points.
735       minContact→Update(m_contactManager.m_contactListener);
736       minContact→m_flags &= ~b2Contact::e_toiFlag;
737       ++minContact→m_toiCount;
738
739       // Is the contact solid?
740       if (minContact→IsEnabled() ≡ false ∨ minContact→IsTouching() ≡ false)
741       {
742         // Restore the sweeps.
743         minContact→SetEnabled(false);
744         bA→m_sweep = backup1;
745         bB→m_sweep = backup2;
746         bA→SynchronizeTransform();
747         bB→SynchronizeTransform();
748         continue;
749       }
750
751       bA→SetAwake(true);
752       bB→SetAwake(true);
753
754       // Build the island
755       island.Clear();
756       island.Add(bA);
757       island.Add(bB);
758       island.Add(minContact);
759
760       bA→m_flags |= b2Body::e_islandFlag;
761       bB→m_flags |= b2Body::e_islandFlag;
762       minContact→m_flags |= b2Contact::e_islandFlag;
763
764       // Get contacts on bodyA and bodyB.
765       b2Body* bodies[2] = {bA, bB};
766       for (int32 i = 0; i < 2; ++i)
767       {
768         b2Body* body = bodies[i];
769         if (body→m_type ≡ b2_dynamicBody)
770         {
771           for (b2ContactEdge* ce = body→m_contactList; ce; ce = ce→next)
772           {
773             if (island.m_bodyCount ≡ island.m_bodyCapacity)
774             {
775               break;
776             }
777
778             if (island.m_contactCount ≡ island.m_contactCapacity)
779             {
780               break;
781             }
782
783             b2Contact* contact = ce→contact;
784
785             // Has this contact already been added to the island?
786             if (contact→m_flags & b2Contact::e_islandFlag)
787             {
788               continue;
789             }
790
```



```
791             // Only add static, kinematic, or bullet bodies.
792             b2Body* other = ce→other;
793             if (other→m_type ≡ b2_dynamicBody ∧
794               body→IsBullet() ≡ false ∧ other→IsBullet() ≡ false)
795             {
796               continue;
797             }
798
799             // Skip sensors.
800             bool sensorA = contact→m_fixtureA→m_isSensor;
801             bool sensorB = contact→m_fixtureB→m_isSensor;
802             if (sensorA ∨ sensorB)
803             {
804               continue;
805             }
806
807             // Tentatively advance the body to the TOI.
808             b2Sweep backup = other→m_sweep;
809             if ((other→m_flags & b2Body::e_islandFlag) ≡ 0)
810             {
811               other→Advance(minAlpha);
812             }
813
814             // Update the contact points
815             contact→Update(m_contactManager.m_contactListener);
816
817             // Was the contact disabled by the user?
818             if (contact→IsEnabled() ≡ false)
819             {
820               other→m_sweep = backup;
821               other→SynchronizeTransform();
822               continue;
823             }
824
825             // Are there contact points?
826             if (contact→IsTouching() ≡ false)
827             {
828               other→m_sweep = backup;
829               other→SynchronizeTransform();
830               continue;
831             }
832
833             // Add the contact to the island
834             contact→m_flags |= b2Contact::e_islandFlag;
835             island.Add(contact);
836
837             // Has the other body already been added to the island?
838             if (other→m_flags & b2Body::e_islandFlag)
839             {
840               continue;
841             }
842
843             // Add the other body to the island.
844             other→m_flags |= b2Body::e_islandFlag;
845
846             if (other→m_type ≠ b2_staticBody)
847             {
848               other→SetAwake(true);
849             }
850
851             island.Add(other);
852           }
853         }
854       }
855
856       b2TimeStep subStep;
```

```
857       subStep.dt = (1.0f − minAlpha) * step.dt;
858       subStep.inv_dt = 1.0f / subStep.dt;
859       subStep.dtRatio = 1.0f;
860       subStep.positionIterations = 20;
861       subStep.velocityIterations = step.velocityIterations;
862       subStep.warmStarting = false;
863       island.SolveTOI(subStep, bA→m_islandIndex, bB→m_islandIndex);
864
865       // Reset island flags and synchronize broad−phase proxies.
866       for (int32 i = 0; i < island.m_bodyCount; ++i)
867       {
868         b2Body* body = island.m_bodies[i];
869         body→m_flags &= ~b2Body::e_islandFlag;
870
871         if (body→m_type ≠ b2_dynamicBody)
872         {
873           continue;
874         }
875
876         body→SynchronizeFixtures();
877
878         // Invalidate all contact TOIs on this displaced body.
879         for (b2ContactEdge* ce = body→m_contactList; ce; ce = ce→next)
880         {
881           ce→contact→m_flags &= ~(b2Contact::e_toiFlag | b2Contact::e_islandFlag
    );
882         }
883       }
884
885       // Commit fixture proxy movements to the broad−phase so that new contacts ar
    e created.
886       // Also, some contacts can be destroyed.
887       m_contactManager.FindNewContacts();
888
889       if (m_subStepping)
890       {
891         m_stepComplete = false;
892         break;
893       }
894     }
895   }
896
897   void b2World::Step(float32 dt, int32 velocityIterations, int32 positionIteration
    s)
898   {
899     b2Timer stepTimer;
900
901     // If new fixtures were added, we need to find the new contacts.
902     if (m_flags & e_newFixture)
903     {
904       m_contactManager.FindNewContacts();
905       m_flags &= ~e_newFixture;
906     }
907
908     m_flags |= e_locked;
909
910     b2TimeStep step;
911     step.dt = dt;
912     step.velocityIterations = velocityIterations;
913     step.positionIterations = positionIterations;
914     if (dt > 0.0f)
915     {
916       step.inv_dt = 1.0f / dt;
917     }
918     else
919     {
```



```
920       step.inv_dt = 0.0f;
921     }
922
923     step.dtRatio = m_inv_dt0 * dt;
924
925     step.warmStarting = m_warmStarting;
926
927     // Update contacts. This is where some contacts are destroyed.
928     {
929       b2Timer timer;
930       m_contactManager.Collide();
931       m_profile.collide = timer.GetMilliseconds();
932     }
933
934     // Integrate velocities, solve velocity constraints, and integrate positions.
935     if (m_stepComplete ∧ step.dt > 0.0f)
936     {
937       b2Timer timer;
938       Solve(step);
939       m_profile.solve = timer.GetMilliseconds();
940     }
941
942     // Handle TOI events.
943     if (m_continuousPhysics ∧ step.dt > 0.0f)
944     {
945       b2Timer timer;
946       SolveTOI(step);
947       m_profile.solveTOI = timer.GetMilliseconds();
948     }
949
950     if (step.dt > 0.0f)
951     {
952       m_inv_dt0 = step.inv_dt;
953     }
954
955     if (m_flags & e_clearForces)
956     {
957       ClearForces();
958     }
959
960     m_flags &= ~e_locked;
961
962     m_profile.step = stepTimer.GetMilliseconds();
963   }
964
965   void b2World::ClearForces()
966   {
967     for (b2Body* body = m_bodyList; body; body = body→GetNext())
968     {
969       body→m_force.SetZero();
970       body→m_torque = 0.0f;
971     }
972   }
973
974   struct b2WorldQueryWrapper
975   {
976     bool QueryCallback(int32 proxyId)
977     {
978       b2FixtureProxy* proxy = (b2FixtureProxy*)broadPhase→GetUserData(proxyId);
979       return callback→ReportFixture(proxy→fixture);
980     }
981
982     const b2BroadPhase* broadPhase;
983     b2QueryCallback* callback;
984   };
985
```

```
986 void b2World::QueryAABB(b2QueryCallback* callback, const b2AABB& aabb) const
987 {
988   b2WorldQueryWrapper wrapper;
989   wrapper.broadPhase = &m_contactManager.m_broadPhase;
990   wrapper.callback = callback;
991   m_contactManager.m_broadPhase.Query(&wrapper, aabb);
992 }
993
994 struct b2WorldRayCastWrapper
995 {
996   float32 RayCastCallback(const b2RayCastInput& input, int32 proxyId)
997   {
998     void* userData = broadPhase->GetUserData(proxyId);
999     b2FixtureProxy* proxy = (b2FixtureProxy*)userData;
1000    b2Fixture* fixture = proxy->fixture;
1001    int32 index = proxy->childIndex;
1002    b2RayCastOutput output;
1003    bool hit = fixture->RayCast(&output, input, index);
1004
1005    if (hit)
1006    {
1007      float32 fraction = output.fraction;
1008      b2Vec2 point = (1.0f - fraction) * input.p1 + fraction * input.p2;
1009      return callback->ReportFixture(fixture, point, output.normal, fraction);
1010    }
1011
1012    return input.maxFraction;
1013  }
1014
1015  const b2BroadPhase* broadPhase;
1016  b2RayCastCallback* callback;
1017 };
1018
1019 void b2World::RayCast(b2RayCastCallback* callback, const b2Vec2& point1, const b
     2Vec2& point2) const
1020 {
1021   b2WorldRayCastWrapper wrapper;
1022   wrapper.broadPhase = &m_contactManager.m_broadPhase;
1023   wrapper.callback = callback;
1024   b2RayCastInput input;
1025   input.maxFraction = 1.0f;
1026   input.p1 = point1;
1027   input.p2 = point2;
1028   m_contactManager.m_broadPhase.RayCast(&wrapper, input);
1029 }
1030
1031 void b2World::DrawShape(b2Fixture* fixture, const b2Transform& xf, const b2Color
     & color)
1032 {
1033   switch (fixture->GetType())
1034   {
1035   case b2Shape::e_circle:
1036     {
1037       b2CircleShape* circle = (b2CircleShape*)fixture->GetShape();
1038
1039       b2Vec2 center = b2Mul(xf, circle->m_p);
1040       float32 radius = circle->m_radius;
1041       b2Vec2 axis = b2Mul(xf.q, b2Vec2(1.0f, 0.0f));
1042
1043       m_debugDraw->DrawSolidCircle(center, radius, axis, color);
1044     }
1045     break;
1046
1047   case b2Shape::e_edge:
1048     {
1049       b2EdgeShape* edge = (b2EdgeShape*)fixture->GetShape();
```



```
1050       b2Vec2 v1 = b2Mul(xf, edge->m_vertex1);
1051       b2Vec2 v2 = b2Mul(xf, edge->m_vertex2);
1052       m_debugDraw->DrawSegment(v1, v2, color);
1053     }
1054     break;
1055
1056   case b2Shape::e_chain:
1057     {
1058       b2ChainShape* chain = (b2ChainShape*)fixture->GetShape();
1059       int32 count = chain->m_count;
1060       const b2Vec2* vertices = chain->m_vertices;
1061
1062       b2Color ghostColor(0.75f * color.r, 0.75f * color.g, 0.75f * color.b, colo
     r.a);
1063
1064       b2Vec2 v1 = b2Mul(xf, vertices[0]);
1065       m_debugDraw->DrawPoint(v1, 4.0f, color);
1066
1067       if (chain->m_hasPrevVertex)
1068       {
1069         b2Vec2 vp = b2Mul(xf, chain->m_prevVertex);
1070         m_debugDraw->DrawSegment(vp, v1, ghostColor);
1071         m_debugDraw->DrawCircle(vp, 0.1f, ghostColor);
1072       }
1073
1074       for (int32 i = 1; i < count; ++i)
1075       {
1076         b2Vec2 v2 = b2Mul(xf, vertices[i]);
1077         m_debugDraw->DrawSegment(v1, v2, color);
1078         m_debugDraw->DrawPoint(v2, 4.0f, color);
1079         v1 = v2;
1080       }
1081
1082       if (chain->m_hasNextVertex)
1083       {
1084         b2Vec2 vn = b2Mul(xf, chain->m_nextVertex);
1085         m_debugDraw->DrawSegment(v1, vn, ghostColor);
1086         m_debugDraw->DrawCircle(vn, 0.1f, ghostColor);
1087       }
1088     }
1089     break;
1090
1091   case b2Shape::e_polygon:
1092     {
1093       b2PolygonShape* poly = (b2PolygonShape*)fixture->GetShape();
1094       int32 vertexCount = poly->m_count;
1095       b2Assert(vertexCount ≤ b2_maxPolygonVertices);
1096       b2Vec2 vertices[b2_maxPolygonVertices];
1097
1098       for (int32 i = 0; i < vertexCount; ++i)
1099       {
1100         vertices[i] = b2Mul(xf, poly->m_vertices[i]);
1101       }
1102
1103       m_debugDraw->DrawSolidPolygon(vertices, vertexCount, color);
1104     }
1105     break;
1106
1107     default:
1108       break;
1109   }
1110 }
1111
1112 void b2World::DrawJoint(b2Joint* joint)
1113 {
1114   b2Body* bodyA = joint->GetBodyA();
```

```
1115    b2Body* bodyB = joint→GetBodyB();
1116    const b2Transform& xf1 = bodyA→GetTransform();
1117    const b2Transform& xf2 = bodyB→GetTransform();
1118    b2Vec2 x1 = xf1.p;
1119    b2Vec2 x2 = xf2.p;
1120    b2Vec2 p1 = joint→GetAnchorA();
1121    b2Vec2 p2 = joint→GetAnchorB();
1122
1123    b2Color color(0.5f, 0.8f, 0.8f);
1124
1125    switch (joint→GetType())
1126    {
1127    case e_distanceJoint:
1128      m_debugDraw→DrawSegment(p1, p2, color);
1129      break;
1130
1131    case e_pulleyJoint:
1132    {
1133      b2PulleyJoint* pulley = (b2PulleyJoint*)joint;
1134      b2Vec2 s1 = pulley→GetGroundAnchorA();
1135      b2Vec2 s2 = pulley→GetGroundAnchorB();
1136      m_debugDraw→DrawSegment(s1, p1, color);
1137      m_debugDraw→DrawSegment(s2, p2, color);
1138      m_debugDraw→DrawSegment(s1, s2, color);
1139    }
1140    break;
1141
1142    case e_mouseJoint:
1143    {
1144      b2Color c;
1145      c.Set(0.0f, 1.0f, 0.0f);
1146      m_debugDraw→DrawPoint(p1, 4.0f, c);
1147      m_debugDraw→DrawPoint(p2, 4.0f, c);
1148
1149      c.Set(0.8f, 0.8f, 0.8f);
1150      m_debugDraw→DrawSegment(p1, p2, c);
1151
1152    }
1153    break;
1154
1155    default:
1156      m_debugDraw→DrawSegment(x1, p1, color);
1157      m_debugDraw→DrawSegment(p1, p2, color);
1158      m_debugDraw→DrawSegment(x2, p2, color);
1159    }
1160  }
1161
1162  void b2World::DrawDebugData()
1163  {
1164    if (m_debugDraw ≡ nullptr)
1165    {
1166      return;
1167    }
1168
1169    uint32 flags = m_debugDraw→GetFlags();
1170
1171    if (flags & b2Draw::e_shapeBit)
1172    {
1173      for (b2Body* b = m_bodyList; b; b = b→GetNext())
1174      {
1175        const b2Transform& xf = b→GetTransform();
1176        for (b2Fixture* f = b→GetFixtureList(); f; f = f→GetNext())
1177        {
1178          if (b→IsActive() ≡ false)
1179          {
1180            DrawShape(f, xf, b2Color(0.5f, 0.5f, 0.3f));
```



```
1181          }
1182          else if (b→GetType() ≡ b2_staticBody)
1183          {
1184            DrawShape(f, xf, b2Color(0.5f, 0.9f, 0.5f));
1185          }
1186          else if (b→GetType() ≡ b2_kinematicBody)
1187          {
1188            DrawShape(f, xf, b2Color(0.5f, 0.5f, 0.9f));
1189          }
1190          else if (b→IsAwake() ≡ false)
1191          {
1192            DrawShape(f, xf, b2Color(0.6f, 0.6f, 0.6f));
1193          }
1194          else
1195          {
1196            DrawShape(f, xf, b2Color(0.9f, 0.7f, 0.7f));
1197          }
1198        }
1199      }
1200    }
1201
1202    if (flags & b2Draw::e_jointBit)
1203    {
1204      for (b2Joint* j = m_jointList; j; j = j→GetNext())
1205      {
1206        DrawJoint(j);
1207      }
1208    }
1209
1210    if (flags & b2Draw::e_pairBit)
1211    {
1212      b2Color color(0.3f, 0.9f, 0.9f);
1213      for (b2Contact* c = m_contactManager.m_contactList; c; c = c→GetNext())
1214      {
1215        //b2Fixture* fixtureA = c->GetFixtureA();
1216        //b2Fixture* fixtureB = c->GetFixtureB();
1217
1218        //b2Vec2 cA = fixtureA->GetAABB().GetCenter();
1219        //b2Vec2 cB = fixtureB->GetAABB().GetCenter();
1220
1221        //g_debugDraw->DrawSegment(cA, cB, color);
1222      }
1223    }
1224
1225    if (flags & b2Draw::e_aabbBit)
1226    {
1227      b2Color color(0.9f, 0.3f, 0.9f);
1228      b2BroadPhase* bp = &m_contactManager.m_broadPhase;
1229
1230      for (b2Body* b = m_bodyList; b; b = b→GetNext())
1231      {
1232        if (b→IsActive() ≡ false)
1233        {
1234          continue;
1235        }
1236
1237        for (b2Fixture* f = b→GetFixtureList(); f; f = f→GetNext())
1238        {
1239          for (int32 i = 0; i < f→m_proxyCount; ++i)
1240          {
1241            b2FixtureProxy* proxy = f→m_proxies + i;
1242            b2AABB aabb = bp→GetFatAABB(proxy→proxyId);
1243            b2Vec2 vs[4];
1244            vs[0].Set(aabb.lowerBound.x, aabb.lowerBound.y);
1245            vs[1].Set(aabb.upperBound.x, aabb.lowerBound.y);
1246            vs[2].Set(aabb.upperBound.x, aabb.upperBound.y);
```

```
1247            vs[3].Set(aabb.lowerBound.x, aabb.upperBound.y);
1248
1249            m_debugDraw→DrawPolygon(vs, 4, color);
1250          }
1251        }
1252      }
1253    }
1254
1255    if (flags & b2Draw::e_centerOfMassBit)
1256    {
1257      for (b2Body* b = m_bodyList; b; b = b→GetNext())
1258      {
1259        b2Transform xf = b→GetTransform();
1260        xf.p = b→GetWorldCenter();
1261        m_debugDraw→DrawTransform(xf);
1262      }
1263    }
1264  }
1265
1266  int32 b2World::GetProxyCount() const
1267  {
1268    return m_contactManager.m_broadPhase.GetProxyCount();
1269  }
1270
1271  int32 b2World::GetTreeHeight() const
1272  {
1273    return m_contactManager.m_broadPhase.GetTreeHeight();
1274  }
1275
1276  int32 b2World::GetTreeBalance() const
1277  {
1278    return m_contactManager.m_broadPhase.GetTreeBalance();
1279  }
1280
1281  float32 b2World::GetTreeQuality() const
1282  {
1283    return m_contactManager.m_broadPhase.GetTreeQuality();
1284  }
1285
1286  void b2World::ShiftOrigin(const b2Vec2& newOrigin)
1287  {
1288    b2Assert((m_flags & e_locked) ≡ 0);
1289    if ((m_flags & e_locked) ≡ e_locked)
1290    {
1291      return;
1292    }
1293
1294    for (b2Body* b = m_bodyList; b; b = b→m_next)
1295    {
1296      b→m_xf.p −= newOrigin;
1297      b→m_sweep.c0 −= newOrigin;
1298      b→m_sweep.c −= newOrigin;
1299    }
1300
1301    for (b2Joint* j = m_jointList; j; j = j→m_next)
1302    {
1303      j→ShiftOrigin(newOrigin);
1304    }
1305
1306    m_contactManager.m_broadPhase.ShiftOrigin(newOrigin);
1307  }
1308
1309  void b2World::Dump()
1310  {
1311    if ((m_flags & e_locked) ≡ e_locked)
1312    {
```



```
1313      return;
1314    }
1315
1316    b2Log("b2Vec2 g(%.15lef, %.15lef);\n", m_gravity.x, m_gravity.y);
1317    b2Log("m_world->SetGravity(g);\n");
1318
1319    b2Log("b2Body** bodies = (b2Body**)b2Alloc(%d * sizeof(b2Body*));\n", m_bodyCount);
1320    b2Log("b2Joint** joints = (b2Joint**)b2Alloc(%d * sizeof(b2Joint*));\n", m_jointCount);
1321    int32 i = 0;
1322    for (b2Body* b = m_bodyList; b; b = b→m_next)
1323    {
1324      b→m_islandIndex = i;
1325      b→Dump();
1326      ++i;
1327    }
1328
1329    i = 0;
1330    for (b2Joint* j = m_jointList; j; j = j→m_next)
1331    {
1332      j→m_index = i;
1333      ++i;
1334    }
1335
1336    // First pass on joints, skip gear joints.
1337    for (b2Joint* j = m_jointList; j; j = j→m_next)
1338    {
1339      if (j→m_type ≡ e_gearJoint)
1340      {
1341        continue;
1342      }
1343
1344      b2Log("{\n");
1345      j→Dump();
1346      b2Log("}\n");
1347    }
1348
1349    // Second pass on joints, only gear joints.
1350    for (b2Joint* j = m_jointList; j; j = j→m_next)
1351    {
1352      if (j→m_type ≠ e_gearJoint)
1353      {
1354        continue;
1355      }
1356
1357      b2Log("{\n");
1358      j→Dump();
1359      b2Log("}\n");
1360    }
1361
1362    b2Log("b2Free(joints);\n");
1363    b2Log("b2Free(bodies);\n");
1364    b2Log("joints = nullptr;\n");
1365    b2Log("bodies = nullptr;\n");
1366  }
```

```cpp
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/b2WorldCallbacks.h"
#include "Box2D/Dynamics/b2Fixture.h"

// Return true if contact calculations should be performed between these two shapes.
// If you implement your own collision filter you may want to build from this implementation.
bool b2ContactFilter::ShouldCollide(b2Fixture* fixtureA, b2Fixture* fixtureB)
{
  const b2Filter& filterA = fixtureA→GetFilterData();
  const b2Filter& filterB = fixtureB→GetFilterData();

  if (filterA.groupIndex ≡ filterB.groupIndex ∧ filterA.groupIndex ≠ 0)
  {
    return filterA.groupIndex > 0;
  }

  bool collide = (filterA.maskBits & filterB.categoryBits) ≠ 0 ∧ (filterA.categoryBits & filterB.maskBits) ≠ 0;
  return collide;
}
```



```cpp
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Collision/b2Distance.h"
#include "Box2D/Dynamics/b2Island.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2Fixture.h"
#include "Box2D/Dynamics/b2World.h"
#include "Box2D/Dynamics/Contacts/b2Contact.h"
#include "Box2D/Dynamics/Contacts/b2ContactSolver.h"
#include "Box2D/Dynamics/Joints/b2Joint.h"
#include "Box2D/Common/b2StackAllocator.h"
#include "Box2D/Common/b2Timer.h"

/*
Position Correction Notes
=========================
I tried the several algorithms for position correction of the 2D revolute joint.
I looked at these systems:
- simple pendulum (1m diameter sphere on massless 5m stick) with initial angular velocity of 100 rad/s.
- suspension bridge with 30 1m long planks of length 1m.
- multi-link chain with 30 1m long links.

Here are the algorithms:

Baumgarte - A fraction of the position error is added to the velocity error. There is no
separate position solver.

Pseudo Velocities - After the velocity solver and position integration,
the position error, Jacobian, and effective mass are recomputed. Then
the velocity constraints are solved with pseudo velocities and a fraction
of the position error is added to the pseudo velocity error. The pseudo
velocities are initialized to zero and there is no warm-starting. After
the position solver, the pseudo velocities are added to the positions.
This is also called the First Order World method or the Position LCP method.

Modified Nonlinear Gauss-Seidel (NGS) - Like Pseudo Velocities except the
position error is re-computed for each constraint and the positions are updated
after the constraint is solved. The radius vectors (aka Jacobians) are
re-computed too (otherwise the algorithm has horrible instability). The pseudo
velocity states are not needed because they are effectively zero at the beginning
of each iteration. Since we have the current position error, we allow the
iterations to terminate early if the error becomes smaller than b2_linearSlop.

Full NGS or just NGS - Like Modified NGS except the effective mass are re-computed
each time a constraint is solved.
```

```
63  Here are the results:
64  Baumgarte - this is the cheapest algorithm but it has some stability problems,
65  especially with the bridge. The chain links separate easily close to the root
66  and they jitter as they struggle to pull together. This is one of the most commo
    n
67  methods in the field. The big drawback is that the position correction artificia
    lly
68  affects the momentum, thus leading to instabilities and false bounce. I used a
69  bias factor of 0.2. A larger bias factor makes the bridge less stable, a smaller
70  factor makes joints and contacts more spongy.
71
72  Pseudo Velocities - the is more stable than the Baumgarte method. The bridge is
73  stable. However, joints still separate with large angular velocities. Drag the
74  simple pendulum in a circle quickly and the joint will separate. The chain separ
    ates
75  easily and does not recover. I used a bias factor of 0.2. A larger value lead to
76  the bridge collapsing when a heavy cube drops on it.
77
78  Modified NGS - this algorithm is better in some ways than Baumgarte and Pseudo
79  Velocities, but in other ways it is worse. The bridge and chain are much more
80  stable, but the simple pendulum goes unstable at high angular velocities.
81
82  Full NGS - stable in all tests. The joints display good stiffness. The bridge
83  still sags, but this is better than infinite forces.
84
85  Recommendations
86  Pseudo Velocities are not really worthwhile because the bridge and chain cannot
87  recover from joint separation. In other cases the benefit over Baumgarte is smal
    l.
88
89  Modified NGS is not a robust method for the revolute joint due to the violent
90  instability seen in the simple pendulum. Perhaps it is viable with other constra
    int
91  types, especially scalar constraints where the effective mass is a scalar.
92
93  This leaves Baumgarte and Full NGS. Baumgarte has small, but manageable instabil
    ities
94  and is very fast. I don't think we can escape Baumgarte, especially in highly
95  demanding cases where high constraint fidelity is not needed.
96
97  Full NGS is robust and easy on the eyes. I recommend this as an option for
98  higher fidelity simulation and certainly for suspension bridges and long chains.
99  Full NGS might be a good choice for ragdolls, especially motorized ragdolls wher
    e
100 joint separation can be problematic. The number of NGS iterations can be reduced
101 for better performance without harming robustness much.
102
103 Each joint in a can be handled differently in the position solver. So I recommen
    d
104 a system where the user can select the algorithm on a per joint basis. I would
105 probably default to the slower Full NGS and let the user select the faster
106 Baumgarte method in performance critical scenarios.
107 */
108
109 /*
110 Cache Performance
111
112 The Box2D solvers are dominated by cache misses. Data structures are designed
113 to increase the number of cache hits. Much of misses are due to random access
114 to body data. The constraint structures are iterated over linearly, which leads
115 to few cache misses.
116
117 The bodies are not accessed during iteration. Instead read only data, such as
118 the mass values are stored with the constraints. The mutable data are the constr
    aint
119 impulses and the bodies velocities/positions. The impulses are held inside the
```



```
120 constraint structures. The body velocities/positions are held in compact, tempor
    ary
121 arrays to increase the number of cache hits. Linear and angular velocity are
122 stored in a single array since multiple arrays lead to multiple misses.
123 */
124
125 /*
126 2D Rotation
127
128 R = [cos(theta) -sin(theta)]
129     [sin(theta) cos(theta) ]
130
131 thetaDot = omega
132
133 Let q1 = cos(theta), q2 = sin(theta).
134 R = [q1 -q2]
135     [q2  q1]
136
137 q1Dot = -thetaDot * q2
138 q2Dot = thetaDot * q1
139
140 q1_new = q1_old - dt * w * q2
141 q2_new = q2_old + dt * w * q1
142 then normalize.
143
144 This might be faster than computing sin+cos.
145 However, we can compute sin+cos of the same angle fast.
146 */
147
148 b2Island::b2Island(
149   int32 bodyCapacity,
150   int32 contactCapacity,
151   int32 jointCapacity,
152   b2StackAllocator* allocator,
153   b2ContactListener* listener)
154 {
155   m_bodyCapacity = bodyCapacity;
156   m_contactCapacity = contactCapacity;
157   m_jointCapacity  = jointCapacity;
158   m_bodyCount = 0;
159   m_contactCount = 0;
160   m_jointCount = 0;
161
162   m_allocator = allocator;
163   m_listener = listener;
164
165   m_bodies = (b2Body**)m_allocator->Allocate(bodyCapacity * sizeof(b2Body*));
166   m_contacts = (b2Contact**)m_allocator->Allocate(contactCapacity   * sizeof(b2Co
    ntact*));
167   m_joints = (b2Joint**)m_allocator->Allocate(jointCapacity * sizeof(b2Joint*));
168
169   m_velocities = (b2Velocity*)m_allocator->Allocate(m_bodyCapacity * sizeof(b2Ve
    locity));
170   m_positions = (b2Position*)m_allocator->Allocate(m_bodyCapacity * sizeof(b2Pos
    ition));
171 }
172
173 b2Island::~b2Island()
174 {
175   // Warning: the order should reverse the constructor order.
176   m_allocator->Free(m_positions);
177   m_allocator->Free(m_velocities);
178   m_allocator->Free(m_joints);
179   m_allocator->Free(m_contacts);
180   m_allocator->Free(m_bodies);
181 }
```

```
182
183 void b2Island::Solve(b2Profile* profile, const b2TimeStep& step, const b2Vec2& g
    ravity, bool allowSleep)
184 {
185   b2Timer timer;
186
187   float32 h = step.dt;
188
189   // Integrate velocities and apply damping. Initialize the body state.
190   for (int32 i = 0; i < m_bodyCount; ++i)
191   {
192     b2Body* b = m_bodies[i];
193
194     b2Vec2 c = b→m_sweep.c;
195     float32 a = b→m_sweep.a;
196     b2Vec2 v = b→m_linearVelocity;
197     float32 w = b→m_angularVelocity;
198
199     // Store positions for continuous collision.
200     b→m_sweep.c0 = b→m_sweep.c;
201     b→m_sweep.a0 = b→m_sweep.a;
202
203     if (b→m_type ≡ b2_dynamicBody)
204     {
205       // Integrate velocities.
206       v += h * (b→m_gravityScale * gravity + b→m_invMass * b→m_force);
207       w += h * b→m_invI * b→m_torque;
208
209       // Apply damping.
210       // ODE: dv/dt + c * v = 0
211       // Solution: v(t) = v0 * exp(-c * t)
212       // Time step: v(t + dt) = v0 * exp(-c * (t + dt)) = v0 * exp(-c * t) * exp
    (-c * dt) = v * exp(-c * dt)
213       // v2 = exp(-c * dt) * v1
214       // Pade approximation:
215       // v2 = v1 * 1 / (1 + c * dt)
216       v *= 1.0f / (1.0f + h * b→m_linearDamping);
217       w *= 1.0f / (1.0f + h * b→m_angularDamping);
218     }
219
220     m_positions[i].c = c;
221     m_positions[i].a = a;
222     m_velocities[i].v = v;
223     m_velocities[i].w = w;
224   }
225
226   timer.Reset();
227
228   // Solver data
229   b2SolverData solverData;
230   solverData.step = step;
231   solverData.positions = m_positions;
232   solverData.velocities = m_velocities;
233
234   // Initialize velocity constraints.
235   b2ContactSolverDef contactSolverDef;
236   contactSolverDef.step = step;
237   contactSolverDef.contacts = m_contacts;
238   contactSolverDef.count = m_contactCount;
239   contactSolverDef.positions = m_positions;
240   contactSolverDef.velocities = m_velocities;
241   contactSolverDef.allocator = m_allocator;
242
243   b2ContactSolver contactSolver(&contactSolverDef);
244   contactSolver.InitializeVelocityConstraints();
245
```



```
246   if (step.warmStarting)
247   {
248     contactSolver.WarmStart();
249   }
250
251   for (int32 i = 0; i < m_jointCount; ++i)
252   {
253     m_joints[i]→InitVelocityConstraints(solverData);
254   }
255
256   profile→solveInit = timer.GetMilliseconds();
257
258   // Solve velocity constraints
259   timer.Reset();
260   for (int32 i = 0; i < step.velocityIterations; ++i)
261   {
262     for (int32 j = 0; j < m_jointCount; ++j)
263     {
264       m_joints[j]→SolveVelocityConstraints(solverData);
265     }
266
267     contactSolver.SolveVelocityConstraints();
268   }
269
270   // Store impulses for warm starting
271   contactSolver.StoreImpulses();
272   profile→solveVelocity = timer.GetMilliseconds();
273
274   // Integrate positions
275   for (int32 i = 0; i < m_bodyCount; ++i)
276   {
277     b2Vec2 c = m_positions[i].c;
278     float32 a = m_positions[i].a;
279     b2Vec2 v = m_velocities[i].v;
280     float32 w = m_velocities[i].w;
281
282     // Check for large velocities
283     b2Vec2 translation = h * v;
284     if (b2Dot(translation, translation) > b2_maxTranslationSquared)
285     {
286       float32 ratio = b2_maxTranslation / translation.Length();
287       v *= ratio;
288     }
289
290     float32 rotation = h * w;
291     if (rotation * rotation > b2_maxRotationSquared)
292     {
293       float32 ratio = b2_maxRotation / b2Abs(rotation);
294       w *= ratio;
295     }
296
297     // Integrate
298     c += h * v;
299     a += h * w;
300
301     m_positions[i].c = c;
302     m_positions[i].a = a;
303     m_velocities[i].v = v;
304     m_velocities[i].w = w;
305   }
306
307   // Solve position constraints
308   timer.Reset();
309   bool positionSolved = false;
310   for (int32 i = 0; i < step.positionIterations; ++i)
311   {
```

```
312       bool contactsOkay = contactSolver.SolvePositionConstraints();
313
314       bool jointsOkay = true;
315       for (int32 j = 0; j < m_jointCount; ++j)
316       {
317         bool jointOkay = m_joints[j]→SolvePositionConstraints(solverData);
318         jointsOkay = jointsOkay ∧ jointOkay;
319       }
320
321       if (contactsOkay ∧ jointsOkay)
322       {
323         // Exit early if the position errors are small.
324         positionSolved = true;
325         break;
326       }
327     }
328
329     // Copy state buffers back to the bodies
330     for (int32 i = 0; i < m_bodyCount; ++i)
331     {
332       b2Body* body = m_bodies[i];
333       body→m_sweep.c = m_positions[i].c;
334       body→m_sweep.a = m_positions[i].a;
335       body→m_linearVelocity = m_velocities[i].v;
336       body→m_angularVelocity = m_velocities[i].w;
337       body→SynchronizeTransform();
338     }
339
340     profile→solvePosition = timer.GetMilliseconds();
341
342     Report(contactSolver.m_velocityConstraints);
343
344     if (allowSleep)
345     {
346       float32 minSleepTime = b2_maxFloat;
347
348       const float32 linTolSqr = b2_linearSleepTolerance * b2_linearSleepTolerance;
349       const float32 angTolSqr = b2_angularSleepTolerance * b2_angularSleepToleranc
    e;
350
351       for (int32 i = 0; i < m_bodyCount; ++i)
352       {
353         b2Body* b = m_bodies[i];
354         if (b→GetType() ≡ b2_staticBody)
355         {
356           continue;
357         }
358
359         if ((b→m_flags & b2Body::e_autoSleepFlag) ≡ 0 ∨
360           b→m_angularVelocity * b→m_angularVelocity > angTolSqr ∨
361           b2Dot(b→m_linearVelocity, b→m_linearVelocity) > linTolSqr)
362         {
363           b→m_sleepTime = 0.0f;
364           minSleepTime = 0.0f;
365         }
366         else
367         {
368           b→m_sleepTime += h;
369           minSleepTime = b2Min(minSleepTime, b→m_sleepTime);
370         }
371       }
372
373       if (minSleepTime ≥ b2_timeToSleep ∧ positionSolved)
374       {
375         for (int32 i = 0; i < m_bodyCount; ++i)
376         {
```



```
377           b2Body* b = m_bodies[i];
378           b→SetAwake(false);
379         }
380       }
381     }
382   }
383
384   void b2Island::SolveTOI(const b2TimeStep& subStep, int32 toiIndexA, int32 toiInd
    exB)
385   {
386     b2Assert(toiIndexA < m_bodyCount);
387     b2Assert(toiIndexB < m_bodyCount);
388
389     // Initialize the body state.
390     for (int32 i = 0; i < m_bodyCount; ++i)
391     {
392       b2Body* b = m_bodies[i];
393       m_positions[i].c = b→m_sweep.c;
394       m_positions[i].a = b→m_sweep.a;
395       m_velocities[i].v = b→m_linearVelocity;
396       m_velocities[i].w = b→m_angularVelocity;
397     }
398
399     b2ContactSolverDef contactSolverDef;
400     contactSolverDef.contacts = m_contacts;
401     contactSolverDef.count = m_contactCount;
402     contactSolverDef.allocator = m_allocator;
403     contactSolverDef.step = subStep;
404     contactSolverDef.positions = m_positions;
405     contactSolverDef.velocities = m_velocities;
406     b2ContactSolver contactSolver(&contactSolverDef);
407
408     // Solve position constraints.
409     for (int32 i = 0; i < subStep.positionIterations; ++i)
410     {
411       bool contactsOkay = contactSolver.SolveTOIPositionConstraints(toiIndexA, toi
    IndexB);
412       if (contactsOkay)
413       {
414         break;
415       }
416     }
417
418   #if 0
419     // Is the new position really safe?
420     for (int32 i = 0; i < m_contactCount; ++i)
421     {
422       b2Contact* c = m_contacts[i];
423       b2Fixture* fA = c→GetFixtureA();
424       b2Fixture* fB = c→GetFixtureB();
425
426       b2Body* bA = fA→GetBody();
427       b2Body* bB = fB→GetBody();
428
429       int32 indexA = c→GetChildIndexA();
430       int32 indexB = c→GetChildIndexB();
431
432       b2DistanceInput input;
433       input.proxyA.Set(fA→GetShape(), indexA);
434       input.proxyB.Set(fB→GetShape(), indexB);
435       input.transformA = bA→GetTransform();
436       input.transformB = bB→GetTransform();
437       input.useRadii = false;
438
439       b2DistanceOutput output;
440       b2SimplexCache cache;
```

```
441       cache.count = 0;
442       b2Distance(&output, &cache, &input);
443
444       if (output.distance ≡ 0 ∨ cache.count ≡ 3)
445       {
446         cache.count += 0;
447       }
448     }
449   #endif
450
451     // Leap of faith to new safe state.
452     m_bodies[toiIndexA]→m_sweep.c0 = m_positions[toiIndexA].c;
453     m_bodies[toiIndexA]→m_sweep.a0 = m_positions[toiIndexA].a;
454     m_bodies[toiIndexB]→m_sweep.c0 = m_positions[toiIndexB].c;
455     m_bodies[toiIndexB]→m_sweep.a0 = m_positions[toiIndexB].a;
456
457     // No warm starting is needed for TOI events because warm
458     // starting impulses were applied in the discrete solver.
459     contactSolver.InitializeVelocityConstraints();
460
461     // Solve velocity constraints.
462     for (int32 i = 0; i < subStep.velocityIterations; ++i)
463     {
464       contactSolver.SolveVelocityConstraints();
465     }
466
467     // Don't store the TOI contact forces for warm starting
468     // because they can be quite large.
469
470     float32 h = subStep.dt;
471
472     // Integrate positions
473     for (int32 i = 0; i < m_bodyCount; ++i)
474     {
475       b2Vec2 c = m_positions[i].c;
476       float32 a = m_positions[i].a;
477       b2Vec2 v = m_velocities[i].v;
478       float32 w = m_velocities[i].w;
479
480       // Check for large velocities
481       b2Vec2 translation = h * v;
482       if (b2Dot(translation, translation) > b2_maxTranslationSquared)
483       {
484         float32 ratio = b2_maxTranslation / translation.Length();
485         v *= ratio;
486       }
487
488       float32 rotation = h * w;
489       if (rotation * rotation > b2_maxRotationSquared)
490       {
491         float32 ratio = b2_maxRotation / b2Abs(rotation);
492         w *= ratio;
493       }
494
495       // Integrate
496       c += h * v;
497       a += h * w;
498
499       m_positions[i].c = c;
500       m_positions[i].a = a;
501       m_velocities[i].v = v;
502       m_velocities[i].w = w;
503
504       // Sync bodies
505       b2Body* body = m_bodies[i];
506       body→m_sweep.c = c;
```



```
507       body→m_sweep.a = a;
508       body→m_linearVelocity = v;
509       body→m_angularVelocity = w;
510       body→SynchronizeTransform();
511     }
512
513     Report(contactSolver.m_velocityConstraints);
514   }
515
516   void b2Island::Report(const b2ContactVelocityConstraint* constraints)
517   {
518     if (m_listener ≡ nullptr)
519     {
520       return;
521     }
522
523     for (int32 i = 0; i < m_contactCount; ++i)
524     {
525       b2Contact* c = m_contacts[i];
526
527       const b2ContactVelocityConstraint* vc = constraints + i;
528
529       b2ContactImpulse impulse;
530       impulse.count = vc→pointCount;
531       for (int32 j = 0; j < vc→pointCount; ++j)
532       {
533         impulse.normalImpulses[j] = vc→points[j].normalImpulse;
534         impulse.tangentImpulses[j] = vc→points[j].tangentImpulse;
535       }
536
537       m_listener→PostSolve(c, &impulse);
538     }
539   }
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/b2Fixture.h"
#include "Box2D/Dynamics/Contacts/b2Contact.h"
#include "Box2D/Dynamics/b2World.h"
#include "Box2D/Collision/Shapes/b2CircleShape.h"
#include "Box2D/Collision/Shapes/b2EdgeShape.h"
#include "Box2D/Collision/Shapes/b2PolygonShape.h"
#include "Box2D/Collision/Shapes/b2ChainShape.h"
#include "Box2D/Collision/b2BroadPhase.h"
#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Common/b2BlockAllocator.h"

b2Fixture::b2Fixture()
{
  m_userData = nullptr;
  m_body = nullptr;
  m_next = nullptr;
  m_proxies = nullptr;
  m_proxyCount = 0;
  m_shape = nullptr;
  m_density = 0.0f;
}

void b2Fixture::Create(b2BlockAllocator* allocator, b2Body* body, const b2FixtureDef* def)
{
  m_userData = def->userData;
  m_friction = def->friction;
  m_restitution = def->restitution;

  m_body = body;
  m_next = nullptr;

  m_filter = def->filter;

  m_isSensor = def->isSensor;

  m_shape = def->shape->Clone(allocator);

  // Reserve proxy space
  int32 childCount = m_shape->GetChildCount();
  m_proxies = (b2FixtureProxy*)allocator->Allocate(childCount * sizeof(b2FixtureProxy));
  for (int32 i = 0; i < childCount; ++i)
  {
    m_proxies[i].fixture = nullptr;
    m_proxies[i].proxyId = b2BroadPhase::e_nullProxy;
  }
  m_proxyCount = 0;
```



```

  m_density = def->density;
}

void b2Fixture::Destroy(b2BlockAllocator* allocator)
{
  // The proxies must be destroyed before calling this.
  b2Assert(m_proxyCount == 0);

  // Free the proxy array.
  int32 childCount = m_shape->GetChildCount();
  allocator->Free(m_proxies, childCount * sizeof(b2FixtureProxy));
  m_proxies = nullptr;

  // Free the child shape.
  switch (m_shape->m_type)
  {
  case b2Shape::e_circle:
    {
      b2CircleShape* s = (b2CircleShape*)m_shape;
      s->~b2CircleShape();
      allocator->Free(s, sizeof(b2CircleShape));
    }
    break;

  case b2Shape::e_edge:
    {
      b2EdgeShape* s = (b2EdgeShape*)m_shape;
      s->~b2EdgeShape();
      allocator->Free(s, sizeof(b2EdgeShape));
    }
    break;

  case b2Shape::e_polygon:
    {
      b2PolygonShape* s = (b2PolygonShape*)m_shape;
      s->~b2PolygonShape();
      allocator->Free(s, sizeof(b2PolygonShape));
    }
    break;

  case b2Shape::e_chain:
    {
      b2ChainShape* s = (b2ChainShape*)m_shape;
      s->~b2ChainShape();
      allocator->Free(s, sizeof(b2ChainShape));
    }
    break;

  default:
    b2Assert(false);
    break;
  }

  m_shape = nullptr;
}

void b2Fixture::CreateProxies(b2BroadPhase* broadPhase, const b2Transform& xf)
{
  b2Assert(m_proxyCount == 0);

  // Create proxies in the broad-phase.
  m_proxyCount = m_shape->GetChildCount();

  for (int32 i = 0; i < m_proxyCount; ++i)
  {
```

```
131     b2FixtureProxy* proxy = m_proxies + i;
132     m_shape→ComputeAABB(&proxy→aabb, xf, i);
133     proxy→proxyId = broadPhase→CreateProxy(proxy→aabb, proxy);
134     proxy→fixture = this;
135     proxy→childIndex = i;
136   }
137 }
138
139 void b2Fixture::DestroyProxies(b2BroadPhase* broadPhase)
140 {
141   // Destroy proxies in the broad-phase.
142   for (int32 i = 0; i < m_proxyCount; ++i)
143   {
144     b2FixtureProxy* proxy = m_proxies + i;
145     broadPhase→DestroyProxy(proxy→proxyId);
146     proxy→proxyId = b2BroadPhase::e_nullProxy;
147   }
148
149   m_proxyCount = 0;
150 }
151
152 void b2Fixture::Synchronize(b2BroadPhase* broadPhase, const b2Transform& transfo
    rm1, const b2Transform& transform2)
153 {
154   if (m_proxyCount ≡ 0)
155   {
156     return;
157   }
158
159   for (int32 i = 0; i < m_proxyCount; ++i)
160   {
161     b2FixtureProxy* proxy = m_proxies + i;
162
163     // Compute an AABB that covers the swept shape (may miss some rotation effec
    t).
164     b2AABB aabb1, aabb2;
165     m_shape→ComputeAABB(&aabb1, transform1, proxy→childIndex);
166     m_shape→ComputeAABB(&aabb2, transform2, proxy→childIndex);
167
168     proxy→aabb.Combine(aabb1, aabb2);
169
170     b2Vec2 displacement = transform2.p − transform1.p;
171
172     broadPhase→MoveProxy(proxy→proxyId, proxy→aabb, displacement);
173   }
174 }
175
176 void b2Fixture::SetFilterData(const b2Filter& filter)
177 {
178   m_filter = filter;
179
180   Refilter();
181 }
182
183 void b2Fixture::Refilter()
184 {
185   if (m_body ≡ nullptr)
186   {
187     return;
188   }
189
190   // Flag associated contacts for filtering.
191   b2ContactEdge* edge = m_body→GetContactList();
192   while (edge)
193   {
194     b2Contact* contact = edge→contact;
```



```
195     b2Fixture* fixtureA = contact→GetFixtureA();
196     b2Fixture* fixtureB = contact→GetFixtureB();
197     if (fixtureA ≡ this ∨ fixtureB ≡ this)
198     {
199       contact→FlagForFiltering();
200     }
201
202     edge = edge→next;
203   }
204
205   b2World* world = m_body→GetWorld();
206
207   if (world ≡ nullptr)
208   {
209     return;
210   }
211
212   // Touch each proxy so that new pairs may be created
213   b2BroadPhase* broadPhase = &world→m_contactManager.m_broadPhase;
214   for (int32 i = 0; i < m_proxyCount; ++i)
215   {
216     broadPhase→TouchProxy(m_proxies[i].proxyId);
217   }
218 }
219
220 void b2Fixture::SetSensor(bool sensor)
221 {
222   if (sensor ≠ m_isSensor)
223   {
224     m_body→SetAwake(true);
225     m_isSensor = sensor;
226   }
227 }
228
229 void b2Fixture::Dump(int32 bodyIndex)
230 {
231   b2Log("  b2FixtureDef fd;\n");
232   b2Log("  fd.friction = %.15lef;\n", m_friction);
233   b2Log("  fd.restitution = %.15lef;\n", m_restitution);
234   b2Log("  fd.density = %.15lef;\n", m_density);
235   b2Log("  fd.isSensor = bool(%d);\n", m_isSensor);
236   b2Log("  fd.filter.categoryBits = uint16(%d);\n", m_filter.categoryBits);
237   b2Log("  fd.filter.maskBits = uint16(%d);\n", m_filter.maskBits);
238   b2Log("  fd.filter.groupIndex = int16(%d);\n", m_filter.groupIndex);
239
240   switch (m_shape→m_type)
241   {
242   case b2Shape::e_circle:
243     {
244       b2CircleShape* s = (b2CircleShape*)m_shape;
245       b2Log("  b2CircleShape shape;\n");
246       b2Log("  shape.m_radius = %.15lef;\n", s→m_radius);
247       b2Log("  shape.m_p.Set(%.15lef, %.15lef);\n", s→m_p.x, s→m_p.y);
248     }
249     break;
250
251   case b2Shape::e_edge:
252     {
253       b2EdgeShape* s = (b2EdgeShape*)m_shape;
254       b2Log("  b2EdgeShape shape;\n");
255       b2Log("  shape.m_radius = %.15lef;\n", s→m_radius);
256       b2Log("  shape.m_vertex0.Set(%.15lef, %.15lef);\n", s→m_vertex0.x, s→m_vertex0.y);
257       b2Log("  shape.m_vertex1.Set(%.15lef, %.15lef);\n", s→m_vertex1.x, s→m_vertex1.y);
258       b2Log("  shape.m_vertex2.Set(%.15lef, %.15lef);\n", s→m_vertex2.x, s→m_vertex2.y);
259       b2Log("  shape.m_vertex3.Set(%.15lef, %.15lef);\n", s→m_vertex3.x, s→m_vertex3.y);
260       b2Log("  shape.m_hasVertex0 = bool(%d);\n", s→m_hasVertex0);
```

```
261         b2Log("  shape.m_hasVertex3 = bool(%d);\n", s→m_hasVertex3);
262       }
263       break;
264
265     case b2Shape::e_polygon:
266       {
267         b2PolygonShape* s = (b2PolygonShape*)m_shape;
268         b2Log("  b2PolygonShape shape;\n");
269         b2Log("  b2Vec2 vs[%d];\n", b2_maxPolygonVertices);
270         for (int32 i = 0; i < s→m_count; ++i)
271         {
272           b2Log("  vs[%d].Set(%.15lef, %.15lef);\n", i, s→m_vertices[i].x, s→m_vertices[i]
    .y);
273         }
274         b2Log("  shape.Set(vs, %d);\n", s→m_count);
275       }
276       break;
277
278     case b2Shape::e_chain:
279       {
280         b2ChainShape* s = (b2ChainShape*)m_shape;
281         b2Log("  b2ChainShape shape;\n");
282         b2Log("  b2Vec2 vs[%d];\n", s→m_count);
283         for (int32 i = 0; i < s→m_count; ++i)
284         {
285           b2Log("  vs[%d].Set(%.15lef, %.15lef);\n", i, s→m_vertices[i].x, s→m_vertices[i]
    .y);
286         }
287         b2Log("  shape.CreateChain(vs, %d);\n", s→m_count);
288         b2Log("  shape.m_prevVertex.Set(%.15lef, %.15lef);\n", s→m_prevVertex.x, s→m_prevVert
    ex.y);
289         b2Log("  shape.m_nextVertex.Set(%.15lef, %.15lef);\n", s→m_nextVertex.x, s→m_nextVert
    ex.y);
290         b2Log("  shape.m_hasPrevVertex = bool(%d);\n", s→m_hasPrevVertex);
291         b2Log("  shape.m_hasNextVertex = bool(%d);\n", s→m_hasNextVertex);
292       }
293       break;
294
295     default:
296       return;
297     }
298
299     b2Log("\n");
300     b2Log("  fd.shape = &shape;\n");
301     b2Log("\n");
302     b2Log("  bodies[%d]->CreateFixture(&fd);\n", bodyIndex);
303   }
```



```
1   /*
2   * Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
3   *
4   * This software is provided 'as-is', without any express or implied
5   * warranty.  In no event will the authors be held liable for any damages
6   * arising from the use of this software.
7   * Permission is granted to anyone to use this software for any purpose,
8   * including commercial applications, and to alter it and redistribute it
9   * freely, subject to the following restrictions:
10  * 1. The origin of this software must not be misrepresented; you must not
11  * claim that you wrote the original software. If you use this software
12  * in a product, an acknowledgment in the product documentation would be
13  * appreciated but is not required.
14  * 2. Altered source versions must be plainly marked as such, and must not be
15  * misrepresented as being the original software.
16  * 3. This notice may not be removed or altered from any source distribution.
17  */
18
19  #include "Box2D/Dynamics/b2ContactManager.h"
20  #include "Box2D/Dynamics/b2Body.h"
21  #include "Box2D/Dynamics/b2Fixture.h"
22  #include "Box2D/Dynamics/b2WorldCallbacks.h"
23  #include "Box2D/Dynamics/Contacts/b2Contact.h"
24
25  b2ContactFilter b2_defaultFilter;
26  b2ContactListener b2_defaultListener;
27
28  b2ContactManager::b2ContactManager()
29  {
30    m_contactList = nullptr;
31    m_contactCount = 0;
32    m_contactFilter = &b2_defaultFilter;
33    m_contactListener = &b2_defaultListener;
34    m_allocator = nullptr;
35  }
36
37  void b2ContactManager::Destroy(b2Contact* c)
38  {
39    b2Fixture* fixtureA = c→GetFixtureA();
40    b2Fixture* fixtureB = c→GetFixtureB();
41    b2Body* bodyA = fixtureA→GetBody();
42    b2Body* bodyB = fixtureB→GetBody();
43
44    if (m_contactListener ∧ c→IsTouching())
45    {
46      m_contactListener→EndContact(c);
47    }
48
49    // Remove from the world.
50    if (c→m_prev)
51    {
52      c→m_prev→m_next = c→m_next;
53    }
54
55    if (c→m_next)
56    {
57      c→m_next→m_prev = c→m_prev;
58    }
59
60    if (c ≡ m_contactList)
61    {
62      m_contactList = c→m_next;
63    }
64
65    // Remove from body 1
66    if (c→m_nodeA.prev)
```

```
67    {
68      c→m_nodeA.prev→next = c→m_nodeA.next;
69    }
70
71    if (c→m_nodeA.next)
72    {
73      c→m_nodeA.next→prev = c→m_nodeA.prev;
74    }
75
76    if (&c→m_nodeA ≡ bodyA→m_contactList)
77    {
78      bodyA→m_contactList = c→m_nodeA.next;
79    }
80
81    // Remove from body 2
82    if (c→m_nodeB.prev)
83    {
84      c→m_nodeB.prev→next = c→m_nodeB.next;
85    }
86
87    if (c→m_nodeB.next)
88    {
89      c→m_nodeB.next→prev = c→m_nodeB.prev;
90    }
91
92    if (&c→m_nodeB ≡ bodyB→m_contactList)
93    {
94      bodyB→m_contactList = c→m_nodeB.next;
95    }
96
97    // Call the factory.
98    b2Contact::Destroy(c, m_allocator);
99    −−m_contactCount;
100 }
101
102 // This is the top level collision call for the time step. Here
103 // all the narrow phase collision is processed for the world
104 // contact list.
105 void b2ContactManager::Collide()
106 {
107   // Update awake contacts.
108   b2Contact* c = m_contactList;
109   while (c)
110   {
111     b2Fixture* fixtureA = c→GetFixtureA();
112     b2Fixture* fixtureB = c→GetFixtureB();
113     int32 indexA = c→GetChildIndexA();
114     int32 indexB = c→GetChildIndexB();
115     b2Body* bodyA = fixtureA→GetBody();
116     b2Body* bodyB = fixtureB→GetBody();
117
118     // Is this contact flagged for filtering?
119     if (c→m_flags & b2Contact::e_filterFlag)
120     {
121       // Should these bodies collide?
122       if (bodyB→ShouldCollide(bodyA) ≡ false)
123       {
124         b2Contact* cNuke = c;
125         c = cNuke→GetNext();
126         Destroy(cNuke);
127         continue;
128       }
129
130       // Check user filtering.
131       if (m_contactFilter ∧ m_contactFilter→ShouldCollide(fixtureA, fixtureB) ≡
    false)
```



```
132       {
133         b2Contact* cNuke = c;
134         c = cNuke→GetNext();
135         Destroy(cNuke);
136         continue;
137       }
138
139       // Clear the filtering flag.
140       c→m_flags &= ~b2Contact::e_filterFlag;
141     }
142
143     bool activeA = bodyA→IsAwake() ∧ bodyA→m_type ≠ b2_staticBody;
144     bool activeB = bodyB→IsAwake() ∧ bodyB→m_type ≠ b2_staticBody;
145
146     // At least one body must be awake and it must be dynamic or kinematic.
147     if (activeA ≡ false ∧ activeB ≡ false)
148     {
149       c = c→GetNext();
150       continue;
151     }
152
153     int32 proxyIdA = fixtureA→m_proxies[indexA].proxyId;
154     int32 proxyIdB = fixtureB→m_proxies[indexB].proxyId;
155     bool overlap = m_broadPhase.TestOverlap(proxyIdA, proxyIdB);
156
157     // Here we destroy contacts that cease to overlap in the broad−phase.
158     if (overlap ≡ false)
159     {
160       b2Contact* cNuke = c;
161       c = cNuke→GetNext();
162       Destroy(cNuke);
163       continue;
164     }
165
166     // The contact persists.
167     c→Update(m_contactListener);
168     c = c→GetNext();
169   }
170 }
171
172 void b2ContactManager::FindNewContacts()
173 {
174   m_broadPhase.UpdatePairs(this);
175 }
176
177 void b2ContactManager::AddPair(void* proxyUserDataA, void* proxyUserDataB)
178 {
179   b2FixtureProxy* proxyA = (b2FixtureProxy*)proxyUserDataA;
180   b2FixtureProxy* proxyB = (b2FixtureProxy*)proxyUserDataB;
181
182   b2Fixture* fixtureA = proxyA→fixture;
183   b2Fixture* fixtureB = proxyB→fixture;
184
185   int32 indexA = proxyA→childIndex;
186   int32 indexB = proxyB→childIndex;
187
188   b2Body* bodyA = fixtureA→GetBody();
189   b2Body* bodyB = fixtureB→GetBody();
190
191   // Are the fixtures on the same body?
192   if (bodyA ≡ bodyB)
193   {
194     return;
195   }
196
197
```

```
198   // bodies have a lot of contacts.
199   // Does a contact already exist?
200   b2ContactEdge* edge = bodyB→GetContactList();
201   while (edge)
202   {
203     if (edge→other ≡ bodyA)
204     {
205       b2Fixture* fA = edge→contact→GetFixtureA();
206       b2Fixture* fB = edge→contact→GetFixtureB();
207       int32 iA = edge→contact→GetChildIndexA();
208       int32 iB = edge→contact→GetChildIndexB();
209
210       if (fA ≡ fixtureA ∧ fB ≡ fixtureB ∧ iA ≡ indexA ∧ iB ≡ indexB)
211       {
212         // A contact already exists.
213         return;
214       }
215
216       if (fA ≡ fixtureB ∧ fB ≡ fixtureA ∧ iA ≡ indexB ∧ iB ≡ indexA)
217       {
218         // A contact already exists.
219         return;
220       }
221     }
222
223     edge = edge→next;
224   }
225
226   // Does a joint override collision? Is at least one body dynamic?
227   if (bodyB→ShouldCollide(bodyA) ≡ false)
228   {
229     return;
230   }
231
232   // Check user filtering.
233   if (m_contactFilter ∧ m_contactFilter→ShouldCollide(fixtureA, fixtureB) ≡ fal
    se)
234   {
235     return;
236   }
237
238   // Call the factory.
239   b2Contact* c = b2Contact::Create(fixtureA, indexA, fixtureB, indexB, m_allocat
    or);
240   if (c ≡ nullptr)
241   {
242     return;
243   }
244
245   // Contact creation may swap fixtures.
246   fixtureA = c→GetFixtureA();
247   fixtureB = c→GetFixtureB();
248   indexA = c→GetChildIndexA();
249   indexB = c→GetChildIndexB();
250   bodyA = fixtureA→GetBody();
251   bodyB = fixtureB→GetBody();
252
253   // Insert into the world.
254   c→m_prev = nullptr;
255   c→m_next = m_contactList;
256   if (m_contactList ≠ nullptr)
257   {
258     m_contactList→m_prev = c;
259   }
260   m_contactList = c;
261
```



```
262   // Connect to island graph.
263
264   // Connect to body A
265   c→m_nodeA.contact = c;
266   c→m_nodeA.other = bodyB;
267
268   c→m_nodeA.prev = nullptr;
269   c→m_nodeA.next = bodyA→m_contactList;
270   if (bodyA→m_contactList ≠ nullptr)
271   {
272     bodyA→m_contactList→prev = &c→m_nodeA;
273   }
274   bodyA→m_contactList = &c→m_nodeA;
275
276   // Connect to body B
277   c→m_nodeB.contact = c;
278   c→m_nodeB.other = bodyA;
279
280   c→m_nodeB.prev = nullptr;
281   c→m_nodeB.next = bodyB→m_contactList;
282   if (bodyB→m_contactList ≠ nullptr)
283   {
284     bodyB→m_contactList→prev = &c→m_nodeB;
285   }
286   bodyB→m_contactList = &c→m_nodeB;
287
288   // Wake up the bodies
289   if (fixtureA→IsSensor() ≡ false ∧ fixtureB→IsSensor() ≡ false)
290   {
291     bodyA→SetAwake(true);
292     bodyB→SetAwake(true);
293   }
294
295   ++m_contactCount;
296 }
```

```
/*
* Copyright (c) 2006-2007 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2Fixture.h"
#include "Box2D/Dynamics/b2World.h"
#include "Box2D/Dynamics/Contacts/b2Contact.h"
#include "Box2D/Dynamics/Joints/b2Joint.h"

b2Body::b2Body(const b2BodyDef* bd, b2World* world)
{
  b2Assert(bd→position.IsValid());
  b2Assert(bd→linearVelocity.IsValid());
  b2Assert(b2IsValid(bd→angle));
  b2Assert(b2IsValid(bd→angularVelocity));
  b2Assert(b2IsValid(bd→angularDamping) ∧ bd→angularDamping ≥ 0.0f);
  b2Assert(b2IsValid(bd→linearDamping) ∧ bd→linearDamping ≥ 0.0f);

  m_flags = 0;

  if (bd→bullet)
  {
    m_flags |= e_bulletFlag;
  }
  if (bd→fixedRotation)
  {
    m_flags |= e_fixedRotationFlag;
  }
  if (bd→allowSleep)
  {
    m_flags |= e_autoSleepFlag;
  }
  if (bd→awake)
  {
    m_flags |= e_awakeFlag;
  }
  if (bd→active)
  {
    m_flags |= e_activeFlag;
  }

  m_world = world;

  m_xf.p = bd→position;
  m_xf.q.Set(bd→angle);

  m_sweep.localCenter.SetZero();
  m_sweep.c0 = m_xf.p;
  m_sweep.c = m_xf.p;
  m_sweep.a0 = bd→angle;
  m_sweep.a = bd→angle;
```



```
  m_sweep.alpha0 = 0.0f;

  m_jointList = nullptr;
  m_contactList = nullptr;
  m_prev = nullptr;
  m_next = nullptr;

  m_linearVelocity = bd→linearVelocity;
  m_angularVelocity = bd→angularVelocity;

  m_linearDamping = bd→linearDamping;
  m_angularDamping = bd→angularDamping;
  m_gravityScale = bd→gravityScale;

  m_force.SetZero();
  m_torque = 0.0f;

  m_sleepTime = 0.0f;

  m_type = bd→type;

  if (m_type ≡ b2_dynamicBody)
  {
    m_mass = 1.0f;
    m_invMass = 1.0f;
  }
  else
  {
    m_mass = 0.0f;
    m_invMass = 0.0f;
  }

  m_I = 0.0f;
  m_invI = 0.0f;

  m_userData = bd→userData;

  m_fixtureList = nullptr;
  m_fixtureCount = 0;
}

b2Body::~b2Body()
{
  // shapes and joints are destroyed in b2World::Destroy
}

void b2Body::SetType(b2BodyType type)
{
  b2Assert(m_world→IsLocked() ≡ false);
  if (m_world→IsLocked() ≡ true)
  {
    return;
  }

  if (m_type ≡ type)
  {
    return;
  }

  m_type = type;

  ResetMassData();

  if (m_type ≡ b2_staticBody)
  {
    m_linearVelocity.SetZero();
```

```
133       m_angularVelocity = 0.0f;
134       m_sweep.a0 = m_sweep.a;
135       m_sweep.c0 = m_sweep.c;
136       SynchronizeFixtures();
137     }
138
139     SetAwake(true);
140
141     m_force.SetZero();
142     m_torque = 0.0f;
143
144     // Delete the attached contacts.
145     b2ContactEdge* ce = m_contactList;
146     while (ce)
147     {
148       b2ContactEdge* ce0 = ce;
149       ce = ce→next;
150       m_world→m_contactManager.Destroy(ce0→contact);
151     }
152     m_contactList = nullptr;
153
154     // Touch the proxies so that new contacts will be created (when appropriate)
155     b2BroadPhase* broadPhase = &m_world→m_contactManager.m_broadPhase;
156     for (b2Fixture* f = m_fixtureList; f; f = f→m_next)
157     {
158       int32 proxyCount = f→m_proxyCount;
159       for (int32 i = 0; i < proxyCount; ++i)
160       {
161         broadPhase→TouchProxy(f→m_proxies[i].proxyId);
162       }
163     }
164   }
165
166   b2Fixture* b2Body::CreateFixture(const b2FixtureDef* def)
167   {
168     b2Assert(m_world→IsLocked() ≡ false);
169     if (m_world→IsLocked() ≡ true)
170     {
171       return nullptr;
172     }
173
174     b2BlockAllocator* allocator = &m_world→m_blockAllocator;
175
176     void* memory = allocator→Allocate(sizeof(b2Fixture));
177     b2Fixture* fixture = new (memory) b2Fixture;
178     fixture→Create(allocator, this, def);
179
180     if (m_flags & e_activeFlag)
181     {
182       b2BroadPhase* broadPhase = &m_world→m_contactManager.m_broadPhase;
183       fixture→CreateProxies(broadPhase, m_xf);
184     }
185
186     fixture→m_next = m_fixtureList;
187     m_fixtureList = fixture;
188     ++m_fixtureCount;
189
190     fixture→m_body = this;
191
192     // Adjust mass properties if needed.
193     if (fixture→m_density > 0.0f)
194     {
195       ResetMassData();
196     }
197
198     // Let the world know we have a new fixture. This will cause new contacts
```



```
199     // to be created at the beginning of the next time step.
200     m_world→m_flags |= b2World::e_newFixture;
201
202     return fixture;
203   }
204
205   b2Fixture* b2Body::CreateFixture(const b2Shape* shape, float32 density)
206   {
207     b2FixtureDef def;
208     def.shape = shape;
209     def.density = density;
210
211     return CreateFixture(&def);
212   }
213
214   void b2Body::DestroyFixture(b2Fixture* fixture)
215   {
216     if (fixture ≡ NULL)
217     {
218       return;
219     }
220
221     b2Assert(m_world→IsLocked() ≡ false);
222     if (m_world→IsLocked() ≡ true)
223     {
224       return;
225     }
226
227     b2Assert(fixture→m_body ≡ this);
228
229     // Remove the fixture from this body's singly linked list.
230     b2Assert(m_fixtureCount > 0);
231     b2Fixture** node = &m_fixtureList;
232     bool found = false;
233     while (*node ≠ nullptr)
234     {
235       if (*node ≡ fixture)
236       {
237         *node = fixture→m_next;
238         found = true;
239         break;
240       }
241
242       node = &(*node)→m_next;
243     }
244
245     // You tried to remove a shape that is not attached to this body.
246     b2Assert(found);
247
248     // Destroy any contacts associated with the fixture.
249     b2ContactEdge* edge = m_contactList;
250     while (edge)
251     {
252       b2Contact* c = edge→contact;
253       edge = edge→next;
254
255       b2Fixture* fixtureA = c→GetFixtureA();
256       b2Fixture* fixtureB = c→GetFixtureB();
257
258       if (fixture ≡ fixtureA ∨ fixture ≡ fixtureB)
259       {
260         // This destroys the contact and removes it from
261         // this body's contact list.
262         m_world→m_contactManager.Destroy(c);
263       }
264     }
```

```
265
266   b2BlockAllocator* allocator = &m_world→m_blockAllocator;
267
268   if (m_flags & e_activeFlag)
269   {
270     b2BroadPhase* broadPhase = &m_world→m_contactManager.m_broadPhase;
271     fixture→DestroyProxies(broadPhase);
272   }
273
274   fixture→m_body = nullptr;
275   fixture→m_next = nullptr;
276   fixture→Destroy(allocator);
277   fixture→~b2Fixture();
278   allocator→Free(fixture, sizeof(b2Fixture));
279
280   --m_fixtureCount;
281
282   // Reset the mass data.
283   ResetMassData();
284 }
285
286 void b2Body::ResetMassData()
287 {
288   // Compute mass data from shapes. Each shape has its own density.
289   m_mass = 0.0f;
290   m_invMass = 0.0f;
291   m_I = 0.0f;
292   m_invI = 0.0f;
293   m_sweep.localCenter.SetZero();
294
295   // Static and kinematic bodies have zero mass.
296   if (m_type ≡ b2_staticBody ∨ m_type ≡ b2_kinematicBody)
297   {
298     m_sweep.c0 = m_xf.p;
299     m_sweep.c = m_xf.p;
300     m_sweep.a0 = m_sweep.a;
301     return;
302   }
303
304   b2Assert(m_type ≡ b2_dynamicBody);
305
306   // Accumulate mass over all fixtures.
307   b2Vec2 localCenter = b2Vec2_zero;
308   for (b2Fixture* f = m_fixtureList; f; f = f→m_next)
309   {
310     if (f→m_density ≡ 0.0f)
311     {
312       continue;
313     }
314
315     b2MassData massData;
316     f→GetMassData(&massData);
317     m_mass += massData.mass;
318     localCenter += massData.mass * massData.center;
319     m_I += massData.I;
320   }
321
322   // Compute center of mass.
323   if (m_mass > 0.0f)
324   {
325     m_invMass = 1.0f / m_mass;
326     localCenter *= m_invMass;
327   }
328   else
329   {
330     // Force all dynamic bodies to have a positive mass.
```



```
331     m_mass = 1.0f;
332     m_invMass = 1.0f;
333   }
334
335   if (m_I > 0.0f ∧ (m_flags & e_fixedRotationFlag) ≡ 0)
336   {
337     // Center the inertia about the center of mass.
338     m_I −= m_mass * b2Dot(localCenter, localCenter);
339     b2Assert(m_I > 0.0f);
340     m_invI = 1.0f / m_I;
341
342   }
343   else
344   {
345     m_I = 0.0f;
346     m_invI = 0.0f;
347   }
348
349   // Move center of mass.
350   b2Vec2 oldCenter = m_sweep.c;
351   m_sweep.localCenter = localCenter;
352   m_sweep.c0 = m_sweep.c = b2Mul(m_xf, m_sweep.localCenter);
353
354   // Update center of mass velocity.
355   m_linearVelocity += b2Cross(m_angularVelocity, m_sweep.c − oldCenter);
356 }
357
358 void b2Body::SetMassData(const b2MassData* massData)
359 {
360   b2Assert(m_world→IsLocked() ≡ false);
361   if (m_world→IsLocked() ≡ true)
362   {
363     return;
364   }
365
366   if (m_type ≠ b2_dynamicBody)
367   {
368     return;
369   }
370
371   m_invMass = 0.0f;
372   m_I = 0.0f;
373   m_invI = 0.0f;
374
375   m_mass = massData→mass;
376   if (m_mass ≤ 0.0f)
377   {
378     m_mass = 1.0f;
379   }
380
381   m_invMass = 1.0f / m_mass;
382
383   if (massData→I > 0.0f ∧ (m_flags & b2Body::e_fixedRotationFlag) ≡ 0)
384   {
385     m_I = massData→I − m_mass * b2Dot(massData→center, massData→center);
386     b2Assert(m_I > 0.0f);
387     m_invI = 1.0f / m_I;
388   }
389
390   // Move center of mass.
391   b2Vec2 oldCenter = m_sweep.c;
392   m_sweep.localCenter =  massData→center;
393   m_sweep.c0 = m_sweep.c = b2Mul(m_xf, m_sweep.localCenter);
394
395   // Update center of mass velocity.
396   m_linearVelocity += b2Cross(m_angularVelocity, m_sweep.c − oldCenter);
```

```
397 }
398
399 bool b2Body::ShouldCollide(const b2Body* other) const
400 {
401   // At least one body should be dynamic.
402   if (m_type ≠ b2_dynamicBody ∧ other→m_type ≠ b2_dynamicBody)
403   {
404     return false;
405   }
406
407   // Does a joint prevent collision?
408   for (b2JointEdge* jn = m_jointList; jn; jn = jn→next)
409   {
410     if (jn→other ≡ other)
411     {
412       if (jn→joint→m_collideConnected ≡ false)
413       {
414         return false;
415       }
416     }
417   }
418
419   return true;
420 }
421
422 void b2Body::SetTransform(const b2Vec2& position, float32 angle)
423 {
424   b2Assert(m_world→IsLocked() ≡ false);
425   if (m_world→IsLocked() ≡ true)
426   {
427     return;
428   }
429
430   m_xf.q.Set(angle);
431   m_xf.p = position;
432
433   m_sweep.c = b2Mul(m_xf, m_sweep.localCenter);
434   m_sweep.a = angle;
435
436   m_sweep.c0 = m_sweep.c;
437   m_sweep.a0 = angle;
438
439   b2BroadPhase* broadPhase = &m_world→m_contactManager.m_broadPhase;
440   for (b2Fixture* f = m_fixtureList; f; f = f→m_next)
441   {
442     f→Synchronize(broadPhase, m_xf, m_xf);
443   }
444 }
445
446 void b2Body::SynchronizeFixtures()
447 {
448   b2Transform xf1;
449   xf1.q.Set(m_sweep.a0);
450   xf1.p = m_sweep.c0 − b2Mul(xf1.q, m_sweep.localCenter);
451
452   b2BroadPhase* broadPhase = &m_world→m_contactManager.m_broadPhase;
453   for (b2Fixture* f = m_fixtureList; f; f = f→m_next)
454   {
455     f→Synchronize(broadPhase, xf1, m_xf);
456   }
457 }
458
459 void b2Body::SetActive(bool flag)
460 {
461   b2Assert(m_world→IsLocked() ≡ false);
462
```



```
463   if (flag ≡ IsActive())
464   {
465     return;
466   }
467
468   if (flag)
469   {
470     m_flags |= e_activeFlag;
471
472     // Create all proxies.
473     b2BroadPhase* broadPhase = &m_world→m_contactManager.m_broadPhase;
474     for (b2Fixture* f = m_fixtureList; f; f = f→m_next)
475     {
476       f→CreateProxies(broadPhase, m_xf);
477     }
478
479     // Contacts are created the next time step.
480   }
481   else
482   {
483     m_flags &= ~e_activeFlag;
484
485     // Destroy all proxies.
486     b2BroadPhase* broadPhase = &m_world→m_contactManager.m_broadPhase;
487     for (b2Fixture* f = m_fixtureList; f; f = f→m_next)
488     {
489       f→DestroyProxies(broadPhase);
490     }
491
492     // Destroy the attached contacts.
493     b2ContactEdge* ce = m_contactList;
494     while (ce)
495     {
496       b2ContactEdge* ce0 = ce;
497       ce = ce→next;
498       m_world→m_contactManager.Destroy(ce0→contact);
499     }
500     m_contactList = nullptr;
501   }
502 }
503
504 void b2Body::SetFixedRotation(bool flag)
505 {
506   bool status = (m_flags & e_fixedRotationFlag) ≡ e_fixedRotationFlag;
507   if (status ≡ flag)
508   {
509     return;
510   }
511
512   if (flag)
513   {
514     m_flags |= e_fixedRotationFlag;
515   }
516   else
517   {
518     m_flags &= ~e_fixedRotationFlag;
519   }
520
521   m_angularVelocity = 0.0f;
522
523   ResetMassData();
524 }
525
526 void b2Body::Dump()
527 {
528   int32 bodyIndex = m_islandIndex;
```

```
529
530   b2Log("{\n");
531   b2Log(" b2BodyDef bd;\n");
532   b2Log(" bd.type = b2BodyType(%d);\n", m_type);
533   b2Log(" bd.position.Set(%.15lef, %.15lef);\n", m_xf.p.x, m_xf.p.y);
534   b2Log(" bd.angle = %.15lef;\n", m_sweep.a);
535   b2Log(" bd.linearVelocity.Set(%.15lef, %.15lef);\n", m_linearVelocity.x, m_linearVelocity.y
    );
536   b2Log(" bd.angularVelocity = %.15lef;\n", m_angularVelocity);
537   b2Log(" bd.linearDamping = %.15lef;\n", m_linearDamping);
538   b2Log(" bd.angularDamping = %.15lef;\n", m_angularDamping);
539   b2Log(" bd.allowSleep = bool(%d);\n", m_flags & e_autoSleepFlag);
540   b2Log(" bd.awake = bool(%d);\n", m_flags & e_awakeFlag);
541   b2Log(" bd.fixedRotation = bool(%d);\n", m_flags & e_fixedRotationFlag);
542   b2Log(" bd.bullet = bool(%d);\n", m_flags & e_bulletFlag);
543   b2Log(" bd.active = bool(%d);\n", m_flags & e_activeFlag);
544   b2Log(" bd.gravityScale = %.15lef;\n", m_gravityScale);
545   b2Log(" bodies[%d] = m_world->CreateBody(&bd);\n", m_islandIndex);
546   b2Log("\n");
547   for (b2Fixture* f = m_fixtureList; f; f = f->m_next)
548   {
549     b2Log(" {\n");
550     f->Dump(bodyIndex);
551     b2Log(" }\n");
552   }
553   b2Log("}\n");
554 }
```



```
1  /*
2  * Copyright (c) 2011 Erin Catto http://box2d.org
3  *
4  * This software is provided 'as-is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #include "Box2D/Common/b2Timer.h"
20
21 #if defined(_WIN32)
22
23 float64 b2Timer::s_invFrequency = 0.0f;
24
25 #ifndef WIN32_LEAN_AND_MEAN
26 #define WIN32_LEAN_AND_MEAN
27 #endif
28
29 #include <windows.h>
30
31 b2Timer::b2Timer()
32 {
33   LARGE_INTEGER largeInteger;
34
35   if (s_invFrequency == 0.0f)
36   {
37     QueryPerformanceFrequency(&largeInteger);
38     s_invFrequency = float64(largeInteger.QuadPart);
39     if (s_invFrequency > 0.0f)
40     {
41       s_invFrequency = 1000.0f / s_invFrequency;
42     }
43   }
44
45   QueryPerformanceCounter(&largeInteger);
46   m_start = float64(largeInteger.QuadPart);
47 }
48
49 void b2Timer::Reset()
50 {
51   LARGE_INTEGER largeInteger;
52   QueryPerformanceCounter(&largeInteger);
53   m_start = float64(largeInteger.QuadPart);
54 }
55
56 float32 b2Timer::GetMilliseconds() const
57 {
58   LARGE_INTEGER largeInteger;
59   QueryPerformanceCounter(&largeInteger);
60   float64 count = float64(largeInteger.QuadPart);
61   float32 ms = float32(s_invFrequency * (count - m_start));
62   return ms;
63 }
64
65 #elif defined(__linux__) ∨ defined (__APPLE__)
66
```

```
#include <sys/time.h>

b2Timer::b2Timer()
{
    Reset();
}

void b2Timer::Reset()
{
    timeval t;
    gettimeofday(&t, 0);
    m_start_sec = t.tv_sec;
    m_start_usec = t.tv_usec;
}

float32 b2Timer::GetMilliseconds() const
{
    timeval t;
    gettimeofday(&t, 0);
  time_t start_sec = m_start_sec;
  suseconds_t start_usec = m_start_usec;

  // http://www.gnu.org/software/libc/manual/html_node/Elapsed-Time.html
  if (t.tv_usec < start_usec)
  {
    int nsec = (start_usec - t.tv_usec) / 1000000 + 1;
    start_usec -= 1000000 * nsec;
    start_sec += nsec;
  }

  if (t.tv_usec - start_usec > 1000000)
  {
    int nsec = (t.tv_usec - start_usec) / 1000000;
    start_usec += 1000000 * nsec;
    start_sec -= nsec;
  }
  return 1000.0f * (t.tv_sec - start_sec) + 0.001f * (t.tv_usec - start_usec);
}

#else

b2Timer::b2Timer()
{
}

void b2Timer::Reset()
{
}

float32 b2Timer::GetMilliseconds() const
{
  return 0.0f;
}

#endif
```



```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Common/b2StackAllocator.h"
#include "Box2D/Common/b2Math.h"

b2StackAllocator::b2StackAllocator()
{
  m_index = 0;
  m_allocation = 0;
  m_maxAllocation = 0;
  m_entryCount = 0;
}

b2StackAllocator::~b2StackAllocator()
{
  b2Assert(m_index == 0);
  b2Assert(m_entryCount == 0);
}

void* b2StackAllocator::Allocate(int32 size)
{
  b2Assert(m_entryCount < b2_maxStackEntries);

  b2StackEntry* entry = m_entries + m_entryCount;
  entry->size = size;
  if (m_index + size > b2_stackSize)
  {
    entry->data = (char*)b2Alloc(size);
    entry->usedMalloc = true;
  }
  else
  {
    entry->data = m_data + m_index;
    entry->usedMalloc = false;
    m_index += size;
  }

  m_allocation += size;
  m_maxAllocation = b2Max(m_maxAllocation, m_allocation);
  ++m_entryCount;

  return entry->data;
}

void b2StackAllocator::Free(void* p)
{
  b2Assert(m_entryCount > 0);
  b2StackEntry* entry = m_entries + m_entryCount - 1;
  b2Assert(p == entry->data);
  if (entry->usedMalloc)
```

```
67    {
68      b2Free(p);
69    }
70    else
71    {
72      m_index -= entry->size;
73    }
74    m_allocation -= entry->size;
75    --m_entryCount;
76
77    p = nullptr;
78  }
79
80  int32 b2StackAllocator::GetMaxAllocation() const
81  {
82    return m_maxAllocation;
83  }
```



```
1   /*
2   * Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
3   *
4   * This software is provided 'as-is', without any express or implied
5   * warranty.  In no event will the authors be held liable for any damages
6   * arising from the use of this software.
7   * Permission is granted to anyone to use this software for any purpose,
8   * including commercial applications, and to alter it and redistribute it
9   * freely, subject to the following restrictions:
10  * 1. The origin of this software must not be misrepresented; you must not
11  * claim that you wrote the original software. If you use this software
12  * in a product, an acknowledgment in the product documentation would be
13  * appreciated but is not required.
14  * 2. Altered source versions must be plainly marked as such, and must not be
15  * misrepresented as being the original software.
16  * 3. This notice may not be removed or altered from any source distribution.
17  */
18
19  #include "Box2D/Common/b2Settings.h"
20  #include <stdio.h>
21  #include <stdarg.h>
22  #include <stdlib.h>
23
24  b2Version b2_version = {2, 3, 2};
25
26  // Memory allocators. Modify these to use your own allocator.
27  void* b2Alloc(int32 size)
28  {
29    return malloc(size);
30  }
31
32  void b2Free(void* mem)
33  {
34    free(mem);
35  }
36
37  // You can modify this to use your logging facility.
38  void b2Log(const char* string, ...)
39  {
40    va_list args;
41    va_start(args, string);
42    vprintf(string, args);
43    va_end(args);
44  }
```

```
/*
* Copyright (c) 2007-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Common/b2Math.h"

const b2Vec2 b2Vec2_zero(0.0f, 0.0f);

/// Solve A * x = b, where b is a column vector. This is more efficient
/// than computing the inverse in one-shot cases.
b2Vec3 b2Mat33::Solve33(const b2Vec3& b) const
{
  float32 det = b2Dot(ex, b2Cross(ey, ez));
  if (det ≠ 0.0f)
  {
    det = 1.0f / det;
  }
  b2Vec3 x;
  x.x = det * b2Dot(b, b2Cross(ey, ez));
  x.y = det * b2Dot(ex, b2Cross(b, ez));
  x.z = det * b2Dot(ex, b2Cross(ey, b));
  return x;
}

/// Solve A * x = b, where b is a column vector. This is more efficient
/// than computing the inverse in one-shot cases.
b2Vec2 b2Mat33::Solve22(const b2Vec2& b) const
{
  float32 a11 = ex.x, a12 = ey.x, a21 = ex.y, a22 = ey.y;
  float32 det = a11 * a22 - a12 * a21;
  if (det ≠ 0.0f)
  {
    det = 1.0f / det;
  }
  b2Vec2 x;
  x.x = det * (a22 * b.x - a12 * b.y);
  x.y = det * (a11 * b.y - a21 * b.x);
  return x;
}

///
void b2Mat33::GetInverse22(b2Mat33* M) const
{
  float32 a = ex.x, b = ey.x, c = ex.y, d = ey.y;
  float32 det = a * d - b * c;
  if (det ≠ 0.0f)
  {
    det = 1.0f / det;
  }

  M→ex.x =  det * d;  M→ey.x = -det * b; M→ex.z = 0.0f;
  M→ex.y = -det * c;  M→ey.y =  det * a; M→ey.z = 0.0f;
```



```
  M→ez.x = 0.0f; M→ez.y = 0.0f; M→ez.z = 0.0f;
}

/// Returns the zero matrix if singular.
void b2Mat33::GetSymInverse33(b2Mat33* M) const
{
  float32 det = b2Dot(ex, b2Cross(ey, ez));
  if (det ≠ 0.0f)
  {
    det = 1.0f / det;
  }

  float32 a11 = ex.x, a12 = ey.x, a13 = ez.x;
  float32 a22 = ey.y, a23 = ez.y;
  float32 a33 = ez.z;

  M→ex.x = det * (a22 * a33 - a23 * a23);
  M→ex.y = det * (a13 * a23 - a12 * a33);
  M→ex.z = det * (a12 * a23 - a13 * a22);

  M→ey.x = M→ex.y;
  M→ey.y = det * (a11 * a33 - a13 * a13);
  M→ey.z = det * (a13 * a12 - a11 * a23);

  M→ez.x = M→ex.z;
  M→ez.y = M→ey.z;
  M→ez.z = det * (a11 * a22 - a12 * a12);
}
```

```cpp
/*
* Copyright (c) 2011 Erin Catto http://box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Common/b2Draw.h"

b2Draw::b2Draw()
{
  m_drawFlags = 0;
}

void b2Draw::SetFlags(uint32 flags)
{
  m_drawFlags = flags;
}

uint32 b2Draw::GetFlags() const
{
  return m_drawFlags;
}

void b2Draw::AppendFlags(uint32 flags)
{
  m_drawFlags |= flags;
}

void b2Draw::ClearFlags(uint32 flags)
{
  m_drawFlags &= ~flags;
}
```



```cpp
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Common/b2BlockAllocator.h"
#include <limits.h>
#include <string.h>
#include <stddef.h>

int32 b2BlockAllocator::s_blockSizes[b2_blockSizes] =
{
  16,   // 0
  32,   // 1
  64,   // 2
  96,   // 3
  128,  // 4
  160,  // 5
  192,  // 6
  224,  // 7
  256,  // 8
  320,  // 9
  384,  // 10
  448,  // 11
  512,  // 12
  640,  // 13
};
uint8 b2BlockAllocator::s_blockSizeLookup[b2_maxBlockSize + 1];
bool b2BlockAllocator::s_blockSizeLookupInitialized;

struct b2Chunk
{
  int32 blockSize;
  b2Block* blocks;
};

struct b2Block
{
  b2Block* next;
};

b2BlockAllocator::b2BlockAllocator()
{
  b2Assert(b2_blockSizes < UCHAR_MAX);

  m_chunkSpace = b2_chunkArrayIncrement;
  m_chunkCount = 0;
  m_chunks = (b2Chunk*)b2Alloc(m_chunkSpace * sizeof(b2Chunk));

  memset(m_chunks, 0, m_chunkSpace * sizeof(b2Chunk));
  memset(m_freeLists, 0, sizeof(m_freeLists));

  if (s_blockSizeLookupInitialized == false)
```

```
67    {
68      int32 j = 0;
69      for (int32 i = 1; i ≤ b2_maxBlockSize; ++i)
70      {
71        b2Assert(j < b2_blockSizes);
72        if (i ≤ s_blockSizes[j])
73        {
74          s_blockSizeLookup[i] = (uint8)j;
75        }
76        else
77        {
78          ++j;
79          s_blockSizeLookup[i] = (uint8)j;
80        }
81      }
82
83      s_blockSizeLookupInitialized = true;
84    }
85  }
86
87  b2BlockAllocator::~b2BlockAllocator()
88  {
89    for (int32 i = 0; i < m_chunkCount; ++i)
90    {
91      b2Free(m_chunks[i].blocks);
92    }
93
94    b2Free(m_chunks);
95  }
96
97  void* b2BlockAllocator::Allocate(int32 size)
98  {
99    if (size ≡ 0)
100     return nullptr;
101
102   b2Assert(0 < size);
103
104   if (size > b2_maxBlockSize)
105   {
106     return b2Alloc(size);
107   }
108
109   int32 index = s_blockSizeLookup[size];
110   b2Assert(0 ≤ index ∧ index < b2_blockSizes);
111
112   if (m_freeLists[index])
113   {
114     b2Block* block = m_freeLists[index];
115     m_freeLists[index] = block→next;
116     return block;
117   }
118   else
119   {
120     if (m_chunkCount ≡ m_chunkSpace)
121     {
122       b2Chunk* oldChunks = m_chunks;
123       m_chunkSpace += b2_chunkArrayIncrement;
124       m_chunks = (b2Chunk*)b2Alloc(m_chunkSpace * sizeof(b2Chunk));
125       memcpy(m_chunks, oldChunks, m_chunkCount * sizeof(b2Chunk));
126       memset(m_chunks + m_chunkCount, 0, b2_chunkArrayIncrement * sizeof(b2Chunk
    ));
127       b2Free(oldChunks);
128     }
129
130     b2Chunk* chunk = m_chunks + m_chunkCount;
131     chunk→blocks = (b2Block*)b2Alloc(b2_chunkSize);
```



```
132 #if defined(_DEBUG)
133     memset(chunk→blocks, 0xcd, b2_chunkSize);
134 #endif
135     int32 blockSize = s_blockSizes[index];
136     chunk→blockSize = blockSize;
137     int32 blockCount = b2_chunkSize / blockSize;
138     b2Assert(blockCount * blockSize ≤ b2_chunkSize);
139     for (int32 i = 0; i < blockCount − 1; ++i)
140     {
141       b2Block* block = (b2Block*)((int8*)chunk→blocks + blockSize * i);
142       b2Block* next = (b2Block*)((int8*)chunk→blocks + blockSize * (i + 1));
143       block→next = next;
144     }
145     b2Block* last = (b2Block*)((int8*)chunk→blocks + blockSize * (blockCount −
    1));
146     last→next = nullptr;
147
148     m_freeLists[index] = chunk→blocks→next;
149     ++m_chunkCount;
150
151     return chunk→blocks;
152   }
153 }
154
155 void b2BlockAllocator::Free(void* p, int32 size)
156 {
157   if (size ≡ 0)
158   {
159     return;
160   }
161
162   b2Assert(0 < size);
163
164   if (size > b2_maxBlockSize)
165   {
166     b2Free(p);
167     return;
168   }
169
170   int32 index = s_blockSizeLookup[size];
171   b2Assert(0 ≤ index ∧ index < b2_blockSizes);
172
173 #ifdef _DEBUG
174   // Verify the memory address and size is valid.
175   int32 blockSize = s_blockSizes[index];
176   bool found = false;
177   for (int32 i = 0; i < m_chunkCount; ++i)
178   {
179     b2Chunk* chunk = m_chunks + i;
180     if (chunk→blockSize ≠ blockSize)
181     {
182       b2Assert( (int8*)p + blockSize ≤ (int8*)chunk→blocks ∨
183             (int8*)chunk→blocks + b2_chunkSize ≤ (int8*)p);
184     }
185     else
186     {
187       if ((int8*)chunk→blocks ≤ (int8*)p ∧ (int8*)p + blockSize ≤ (int8*)chunk
    →blocks + b2_chunkSize)
188       {
189         found = true;
190       }
191     }
192   }
193
194   b2Assert(found);
195
```

```
196   memset(p, 0xfd, blockSize);
197 #endif
198
199   b2Block* block = (b2Block*)p;
200   block→next = m_freeLists[index];
201   m_freeLists[index] = block;
202 }
203
204 void b2BlockAllocator::Clear()
205 {
206   for (int32 i = 0; i < m_chunkCount; ++i)
207   {
208     b2Free(m_chunks[i].blocks);
209   }
210
211   m_chunkCount = 0;
212   memset(m_chunks, 0, m_chunkSpace * sizeof(b2Chunk));
213
214   memset(m_freeLists, 0, sizeof(m_freeLists));
215 }
```



```
1  /*
2  * Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as-is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #include "Box2D/Collision/Shapes/b2PolygonShape.h"
20 #include <new>
21
22 b2Shape* b2PolygonShape::Clone(b2BlockAllocator* allocator) const
23 {
24   void* mem = allocator→Allocate(sizeof(b2PolygonShape));
25   b2PolygonShape* clone = new (mem) b2PolygonShape;
26   *clone = *this;
27   return clone;
28 }
29
30 void b2PolygonShape::SetAsBox(float32 hx, float32 hy)
31 {
32   m_count = 4;
33   m_vertices[0].Set(-hx, -hy);
34   m_vertices[1].Set( hx, -hy);
35   m_vertices[2].Set( hx,  hy);
36   m_vertices[3].Set(-hx,  hy);
37   m_normals[0].Set(0.0f, -1.0f);
38   m_normals[1].Set(1.0f, 0.0f);
39   m_normals[2].Set(0.0f, 1.0f);
40   m_normals[3].Set(-1.0f, 0.0f);
41   m_centroid.SetZero();
42 }
43
44 void b2PolygonShape::SetAsBox(float32 hx, float32 hy, const b2Vec2& center, floa
   t32 angle)
45 {
46   m_count = 4;
47   m_vertices[0].Set(-hx, -hy);
48   m_vertices[1].Set( hx, -hy);
49   m_vertices[2].Set( hx,  hy);
50   m_vertices[3].Set(-hx,  hy);
51   m_normals[0].Set(0.0f, -1.0f);
52   m_normals[1].Set(1.0f, 0.0f);
53   m_normals[2].Set(0.0f, 1.0f);
54   m_normals[3].Set(-1.0f, 0.0f);
55   m_centroid = center;
56
57   b2Transform xf;
58   xf.p = center;
59   xf.q.Set(angle);
60
61   // Transform vertices and normals.
62   for (int32 i = 0; i < m_count; ++i)
63   {
64     m_vertices[i] = b2Mul(xf, m_vertices[i]);
65     m_normals[i] = b2Mul(xf.q, m_normals[i]);
```

```
66    }
67  }
68
69  int32 b2PolygonShape::GetChildCount() const
70  {
71    return 1;
72  }
73
74  static b2Vec2 ComputeCentroid(const b2Vec2* vs, int32 count)
75  {
76    b2Assert(count ≥ 3);
77
78    b2Vec2 c; c.Set(0.0f, 0.0f);
79    float32 area = 0.0f;
80
81    // pRef is the reference point for forming triangles.
82    // It's location doesn't change the result (except for rounding error).
83    b2Vec2 pRef(0.0f, 0.0f);
84  #if 0
85    // This code would put the reference point inside the polygon.
86    for (int32 i = 0; i < count; ++i)
87    {
88      pRef += vs[i];
89    }
90    pRef *= 1.0f / count;
91  #endif
92
93    const float32 inv3 = 1.0f / 3.0f;
94
95    for (int32 i = 0; i < count; ++i)
96    {
97      // Triangle vertices.
98      b2Vec2 p1 = pRef;
99      b2Vec2 p2 = vs[i];
100     b2Vec2 p3 = i + 1 < count ? vs[i+1] : vs[0];
101
102     b2Vec2 e1 = p2 − p1;
103     b2Vec2 e2 = p3 − p1;
104
105     float32 D = b2Cross(e1, e2);
106
107     float32 triangleArea = 0.5f * D;
108     area += triangleArea;
109
110     // Area weighted centroid
111     c += triangleArea * inv3 * (p1 + p2 + p3);
112   }
113
114   // Centroid
115   b2Assert(area > b2_epsilon);
116   c *= 1.0f / area;
117   return c;
118 }
119
120 void b2PolygonShape::Set(const b2Vec2* vertices, int32 count)
121 {
122   b2Assert(3 ≤ count ∧ count ≤ b2_maxPolygonVertices);
123   if (count < 3)
124   {
125     SetAsBox(1.0f, 1.0f);
126     return;
127   }
128
129   int32 n = b2Min(count, b2_maxPolygonVertices);
130
131   // Perform welding and copy vertices into local buffer.
```



```
132   b2Vec2 ps[b2_maxPolygonVertices];
133   int32 tempCount = 0;
134   for (int32 i = 0; i < n; ++i)
135   {
136     b2Vec2 v = vertices[i];
137
138     bool unique = true;
139     for (int32 j = 0; j < tempCount; ++j)
140     {
141       if (b2DistanceSquared(v, ps[j]) < ((0.5f * b2_linearSlop) * (0.5f * b2_lin
    earSlop)))
142       {
143         unique = false;
144         break;
145       }
146     }
147
148     if (unique)
149     {
150       ps[tempCount++] = v;
151     }
152   }
153
154   n = tempCount;
155   if (n < 3)
156   {
157     // Polygon is degenerate.
158     b2Assert(false);
159     SetAsBox(1.0f, 1.0f);
160     return;
161   }
162
163   // Create the convex hull using the Gift wrapping algorithm
164   // http://en.wikipedia.org/wiki/Gift_wrapping_algorithm
165
166   // Find the right most point on the hull
167   int32 i0 = 0;
168   float32 x0 = ps[0].x;
169   for (int32 i = 1; i < n; ++i)
170   {
171     float32 x = ps[i].x;
172     if (x > x0 ∨ (x ≡ x0 ∧ ps[i].y < ps[i0].y))
173     {
174       i0 = i;
175       x0 = x;
176     }
177   }
178
179   int32 hull[b2_maxPolygonVertices];
180   int32 m = 0;
181   int32 ih = i0;
182
183   for (;;)
184   {
185     b2Assert(m < b2_maxPolygonVertices);
186     hull[m] = ih;
187
188     int32 ie = 0;
189     for (int32 j = 1; j < n; ++j)
190     {
191       if (ie ≡ ih)
192       {
193         ie = j;
194         continue;
195       }
196
```

```
197         b2Vec2 r = ps[ie] − ps[hull[m]];
198         b2Vec2 v = ps[j] − ps[hull[m]];
199         float32 c = b2Cross(r, v);
200         if (c < 0.0f)
201         {
202           ie = j;
203         }
204
205         // Collinearity check
206         if (c ≡ 0.0f ∧ v.LengthSquared() > r.LengthSquared())
207         {
208           ie = j;
209         }
210       }
211
212       ++m;
213       ih = ie;
214
215       if (ie ≡ i0)
216       {
217         break;
218       }
219     }
220
221     if (m < 3)
222     {
223       // Polygon is degenerate.
224       b2Assert(false);
225       SetAsBox(1.0f, 1.0f);
226       return;
227     }
228
229     m_count = m;
230
231     // Copy vertices.
232     for (int32 i = 0; i < m; ++i)
233     {
234       m_vertices[i] = ps[hull[i]];
235     }
236
237     // Compute normals. Ensure the edges have non−zero length.
238     for (int32 i = 0; i < m; ++i)
239     {
240       int32 i1 = i;
241       int32 i2 = i + 1 < m ? i + 1 : 0;
242       b2Vec2 edge = m_vertices[i2] − m_vertices[i1];
243       b2Assert(edge.LengthSquared() > b2_epsilon * b2_epsilon);
244       m_normals[i] = b2Cross(edge, 1.0f);
245       m_normals[i].Normalize();
246     }
247
248     // Compute the polygon centroid.
249     m_centroid = ComputeCentroid(m_vertices, m);
250   }
251
252   bool b2PolygonShape::TestPoint(const b2Transform& xf, const b2Vec2& p) const
253   {
254     b2Vec2 pLocal = b2MulT(xf.q, p − xf.p);
255
256     for (int32 i = 0; i < m_count; ++i)
257     {
258       float32 dot = b2Dot(m_normals[i], pLocal − m_vertices[i]);
259       if (dot > 0.0f)
260       {
261         return false;
262       }
```



```
263     }
264
265     return true;
266   }
267
268   bool b2PolygonShape::RayCast(b2RayCastOutput* output, const b2RayCastInput& inpu
      t,
269                                const b2Transform& xf, int32 childIndex) const
270   {
271     B2_NOT_USED(childIndex);
272
273     // Put the ray into the polygon's frame of reference.
274     b2Vec2 p1 = b2MulT(xf.q, input.p1 − xf.p);
275     b2Vec2 p2 = b2MulT(xf.q, input.p2 − xf.p);
276     b2Vec2 d = p2 − p1;
277
278     float32 lower = 0.0f, upper = input.maxFraction;
279
280     int32 index = −1;
281
282     for (int32 i = 0; i < m_count; ++i)
283     {
284       // p = p1 + a * d
285       // dot(normal, p − v) = 0
286       // dot(normal, p1 − v) + a * dot(normal, d) = 0
287       float32 numerator = b2Dot(m_normals[i], m_vertices[i] − p1);
288       float32 denominator = b2Dot(m_normals[i], d);
289
290       if (denominator ≡ 0.0f)
291       {
292         if (numerator < 0.0f)
293         {
294           return false;
295         }
296       }
297       else
298       {
299         // Note: we want this predicate without division:
300         // lower < numerator / denominator, where denominator < 0
301         // Since denominator < 0, we have to flip the inequality:
302         // lower < numerator / denominator <==> denominator * lower > numerator.
303         if (denominator < 0.0f ∧ numerator < lower * denominator)
304         {
305           // Increase lower.
306           // The segment enters this half−space.
307           lower = numerator / denominator;
308           index = i;
309         }
310         else if (denominator > 0.0f ∧ numerator < upper * denominator)
311         {
312           // Decrease upper.
313           // The segment exits this half−space.
314           upper = numerator / denominator;
315         }
316       }
317
318       // The use of epsilon here causes the assert on lower to trip
319       // in some cases. Apparently the use of epsilon was to make edge
320       // shapes work, but now those are handled separately.
321       //if (upper < lower − b2_epsilon)
322       if (upper < lower)
323       {
324         return false;
325       }
326     }
327
```

```
328   b2Assert(0.0f ≤ lower ∧ lower ≤ input.maxFraction);
329
330   if (index ≥ 0)
331   {
332     output→fraction = lower;
333     output→normal = b2Mul(xf.q, m_normals[index]);
334     return true;
335   }
336
337   return false;
338 }
339
340 void b2PolygonShape::ComputeAABB(b2AABB* aabb, const b2Transform& xf, int32 chil
    dIndex) const
341 {
342   B2_NOT_USED(childIndex);
343
344   b2Vec2 lower = b2Mul(xf, m_vertices[0]);
345   b2Vec2 upper = lower;
346
347   for (int32 i = 1; i < m_count; ++i)
348   {
349     b2Vec2 v = b2Mul(xf, m_vertices[i]);
350     lower = b2Min(lower, v);
351     upper = b2Max(upper, v);
352   }
353
354   b2Vec2 r(m_radius, m_radius);
355   aabb→lowerBound = lower − r;
356   aabb→upperBound = upper + r;
357 }
358
359 void b2PolygonShape::ComputeMass(b2MassData* massData, float32 density) const
360 {
361   // Polygon mass, centroid, and inertia.
362   // Let rho be the polygon density in mass per unit area.
363   // Then:
364   // mass = rho * int(dA)
365   // centroid.x = (1/mass) * rho * int(x * dA)
366   // centroid.y = (1/mass) * rho * int(y * dA)
367   // I = rho * int((x*x + y*y) * dA)
368   //
369   // We can compute these integrals by summing all the integrals
370   // for each triangle of the polygon. To evaluate the integral
371   // for a single triangle, we make a change of variables to
372   // the (u,v) coordinates of the triangle:
373   // x = x0 + e1x * u + e2x * v
374   // y = y0 + e1y * u + e2y * v
375   // where 0 <= u && 0 <= v && u + v <= 1.
376   //
377   // We integrate u from [0,1−v] and then v from [0,1].
378   // We also need to use the Jacobian of the transformation:
379   // D = cross(e1, e2)
380   //
381   // Simplification: triangle centroid = (1/3) * (p1 + p2 + p3)
382   //
383   // The rest of the derivation is handled by computer algebra.
384
385   b2Assert(m_count ≥ 3);
386
387   b2Vec2 center; center.Set(0.0f, 0.0f);
388   float32 area = 0.0f;
389   float32 I = 0.0f;
390
391   // s is the reference point for forming triangles.
392   // It’s location doesn’t change the result (except for rounding error).
```



```
393   b2Vec2 s(0.0f, 0.0f);
394
395   // This code would put the reference point inside the polygon.
396   for (int32 i = 0; i < m_count; ++i)
397   {
398     s += m_vertices[i];
399   }
400   s *= 1.0f / m_count;
401
402   const float32 k_inv3 = 1.0f / 3.0f;
403
404   for (int32 i = 0; i < m_count; ++i)
405   {
406     // Triangle vertices.
407     b2Vec2 e1 = m_vertices[i] − s;
408     b2Vec2 e2 = i + 1 < m_count ? m_vertices[i+1] − s : m_vertices[0] − s;
409
410     float32 D = b2Cross(e1, e2);
411
412     float32 triangleArea = 0.5f * D;
413     area += triangleArea;
414
415     // Area weighted centroid
416     center += triangleArea * k_inv3 * (e1 + e2);
417
418     float32 ex1 = e1.x, ey1 = e1.y;
419     float32 ex2 = e2.x, ey2 = e2.y;
420
421     float32 intx2 = ex1*ex1 + ex2*ex1 + ex2*ex2;
422     float32 inty2 = ey1*ey1 + ey2*ey1 + ey2*ey2;
423
424     I += (0.25f * k_inv3 * D) * (intx2 + inty2);
425   }
426
427   // Total mass
428   massData→mass = density * area;
429
430   // Center of mass
431   b2Assert(area > b2_epsilon);
432   center *= 1.0f / area;
433   massData→center = center + s;
434
435   // Inertia tensor relative to the local origin (point s).
436   massData→I = density * I;
437
438   // Shift to center of mass then to original body origin.
439   massData→I += massData→mass * (b2Dot(massData→center, massData→center) − b2
    Dot(center, center));
440 }
441
442 bool b2PolygonShape::Validate() const
443 {
444   for (int32 i = 0; i < m_count; ++i)
445   {
446     int32 i1 = i;
447     int32 i2 = i < m_count − 1 ? i1 + 1 : 0;
448     b2Vec2 p = m_vertices[i1];
449     b2Vec2 e = m_vertices[i2] − p;
450
451     for (int32 j = 0; j < m_count; ++j)
452     {
453       if (j ≡ i1 ∨ j ≡ i2)
454       {
455         continue;
456       }
457
```

```
458         b2Vec2 v = m_vertices[j] - p;
459         float32 c = b2Cross(e, v);
460         if (c < 0.0f)
461         {
462           return false;
463         }
464       }
465     }
466
467     return true;
468   }
```



```
1   /*
2   * Copyright (c) 2006-2010 Erin Catto http://www.box2d.org
3   *
4   * This software is provided 'as-is', without any express or implied
5   * warranty.  In no event will the authors be held liable for any damages
6   * arising from the use of this software.
7   * Permission is granted to anyone to use this software for any purpose,
8   * including commercial applications, and to alter it and redistribute it
9   * freely, subject to the following restrictions:
10  * 1. The origin of this software must not be misrepresented; you must not
11  * claim that you wrote the original software. If you use this software
12  * in a product, an acknowledgment in the product documentation would be
13  * appreciated but is not required.
14  * 2. Altered source versions must be plainly marked as such, and must not be
15  * misrepresented as being the original software.
16  * 3. This notice may not be removed or altered from any source distribution.
17  */
18
19  #include "Box2D/Collision/Shapes/b2EdgeShape.h"
20  #include <new>
21
22  void b2EdgeShape::Set(const b2Vec2& v1, const b2Vec2& v2)
23  {
24    m_vertex1 = v1;
25    m_vertex2 = v2;
26    m_hasVertex0 = false;
27    m_hasVertex3 = false;
28  }
29
30  b2Shape* b2EdgeShape::Clone(b2BlockAllocator* allocator) const
31  {
32    void* mem = allocator->Allocate(sizeof(b2EdgeShape));
33    b2EdgeShape* clone = new (mem) b2EdgeShape;
34    *clone = *this;
35    return clone;
36  }
37
38  int32 b2EdgeShape::GetChildCount() const
39  {
40    return 1;
41  }
42
43  bool b2EdgeShape::TestPoint(const b2Transform& xf, const b2Vec2& p) const
44  {
45    B2_NOT_USED(xf);
46    B2_NOT_USED(p);
47    return false;
48  }
49
50  // p = p1 + t * d
51  // v = v1 + s * e
52  // p1 + t * d = v1 + s * e
53  // s * e - t * d = p1 - v1
54  bool b2EdgeShape::RayCast(b2RayCastOutput* output, const b2RayCastInput& input,
55                            const b2Transform& xf, int32 childIndex) const
56  {
57    B2_NOT_USED(childIndex);
58
59    // Put the ray into the edge's frame of reference.
60    b2Vec2 p1 = b2MulT(xf.q, input.p1 - xf.p);
61    b2Vec2 p2 = b2MulT(xf.q, input.p2 - xf.p);
62    b2Vec2 d = p2 - p1;
63
64    b2Vec2 v1 = m_vertex1;
65    b2Vec2 v2 = m_vertex2;
66    b2Vec2 e = v2 - v1;
```

```
67    b2Vec2 normal(e.y, -e.x);
68    normal.Normalize();
69
70    // q = p1 + t * d
71    // dot(normal, q - v1) = 0
72    // dot(normal, p1 - v1) + t * dot(normal, d) = 0
73    float32 numerator = b2Dot(normal, v1 - p1);
74    float32 denominator = b2Dot(normal, d);
75
76    if (denominator == 0.0f)
77    {
78      return false;
79    }
80
81    float32 t = numerator / denominator;
82    if (t < 0.0f || input.maxFraction < t)
83    {
84      return false;
85    }
86
87    b2Vec2 q = p1 + t * d;
88
89    // q = v1 + s * r
90    // s = dot(q - v1, r) / dot(r, r)
91    b2Vec2 r = v2 - v1;
92    float32 rr = b2Dot(r, r);
93    if (rr == 0.0f)
94    {
95      return false;
96    }
97
98    float32 s = b2Dot(q - v1, r) / rr;
99    if (s < 0.0f || 1.0f < s)
100   {
101     return false;
102   }
103
104   output->fraction = t;
105   if (numerator > 0.0f)
106   {
107     output->normal = -b2Mul(xf.q, normal);
108   }
109   else
110   {
111     output->normal = b2Mul(xf.q, normal);
112   }
113   return true;
114 }
115
116 void b2EdgeShape::ComputeAABB(b2AABB* aabb, const b2Transform& xf, int32 childIn
    dex) const
117 {
118   B2_NOT_USED(childIndex);
119
120   b2Vec2 v1 = b2Mul(xf, m_vertex1);
121   b2Vec2 v2 = b2Mul(xf, m_vertex2);
122
123   b2Vec2 lower = b2Min(v1, v2);
124   b2Vec2 upper = b2Max(v1, v2);
125
126   b2Vec2 r(m_radius, m_radius);
127   aabb->lowerBound = lower - r;
128   aabb->upperBound = upper + r;
129 }
130
131 void b2EdgeShape::ComputeMass(b2MassData* massData, float32 density) const
```



```
132 {
133   B2_NOT_USED(density);
134
135   massData->mass = 0.0f;
136   massData->center = 0.5f * (m_vertex1 + m_vertex2);
137   massData->I = 0.0f;
138 }
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Collision/Shapes/b2CircleShape.h"
#include <new>

b2Shape* b2CircleShape::Clone(b2BlockAllocator* allocator) const
{
  void* mem = allocator→Allocate(sizeof(b2CircleShape));
  b2CircleShape* clone = new (mem) b2CircleShape;
  *clone = *this;
  return clone;
}

int32 b2CircleShape::GetChildCount() const
{
  return 1;
}

bool b2CircleShape::TestPoint(const b2Transform& transform, const b2Vec2& p) const
{
  b2Vec2 center = transform.p + b2Mul(transform.q, m_p);
  b2Vec2 d = p - center;
  return b2Dot(d, d) ≤ m_radius * m_radius;
}

// Collision Detection in Interactive 3D Environments by Gino van den Bergen
// From Section 3.1.2
// x = s + a * r
// norm(x) = radius
bool b2CircleShape::RayCast(b2RayCastOutput* output, const b2RayCastInput& input,
              const b2Transform& transform, int32 childIndex) const
{
  B2_NOT_USED(childIndex);

  b2Vec2 position = transform.p + b2Mul(transform.q, m_p);
  b2Vec2 s = input.p1 - position;
  float32 b = b2Dot(s, s) - m_radius * m_radius;

  // Solve quadratic equation.
  b2Vec2 r = input.p2 - input.p1;
  float32 c =  b2Dot(s, r);
  float32 rr = b2Dot(r, r);
  float32 sigma = c * c - rr * b;

  // Check for negative discriminant and short segment.
  if (sigma < 0.0f ∨ rr < b2_epsilon)
  {
    return false;
  }

  // Find the point of intersection of the line with the circle.
  float32 a = -(c + b2Sqrt(sigma));

  // Is the intersection point on the segment?
  if (0.0f ≤ a ∧ a ≤ input.maxFraction * rr)
  {
    a /= rr;
    output→fraction = a;
    output→normal = s + a * r;
    output→normal.Normalize();
    return true;
  }

  return false;
}

void b2CircleShape::ComputeAABB(b2AABB* aabb, const b2Transform& transform, int32 childIndex) const
{
  B2_NOT_USED(childIndex);

  b2Vec2 p = transform.p + b2Mul(transform.q, m_p);
  aabb→lowerBound.Set(p.x - m_radius, p.y - m_radius);
  aabb→upperBound.Set(p.x + m_radius, p.y + m_radius);
}

void b2CircleShape::ComputeMass(b2MassData* massData, float32 density) const
{
  massData→mass = density * b2_pi * m_radius * m_radius;
  massData→center = m_p;

  // inertia about the local origin
  massData→I = massData→mass * (0.5f * m_radius * m_radius + b2Dot(m_p, m_p));
}
```

```
/*
* Copyright (c) 2006-2010 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Collision/Shapes/b2ChainShape.h"
#include "Box2D/Collision/Shapes/b2EdgeShape.h"
#include <new>
#include <string.h>

b2ChainShape::~b2ChainShape()
{
  Clear();
}

void b2ChainShape::Clear()
{
  b2Free(m_vertices);
  m_vertices = nullptr;
  m_count = 0;
}

void b2ChainShape::CreateLoop(const b2Vec2* vertices, int32 count)
{
  b2Assert(m_vertices == nullptr && m_count == 0);
  b2Assert(count >= 3);
  if (count < 3)
  {
    return;
  }

  for (int32 i = 1; i < count; ++i)
  {
    b2Vec2 v1 = vertices[i-1];
    b2Vec2 v2 = vertices[i];
    // If the code crashes here, it means your vertices are too close together.
    b2Assert(b2DistanceSquared(v1, v2) > b2_linearSlop * b2_linearSlop);
  }

  m_count = count + 1;
  m_vertices = (b2Vec2*)b2Alloc(m_count * sizeof(b2Vec2));
  memcpy(m_vertices, vertices, count * sizeof(b2Vec2));
  m_vertices[count] = m_vertices[0];
  m_prevVertex = m_vertices[m_count - 2];
  m_nextVertex = m_vertices[1];
  m_hasPrevVertex = true;
  m_hasNextVertex = true;
}

void b2ChainShape::CreateChain(const b2Vec2* vertices, int32 count)
{
  b2Assert(m_vertices == nullptr && m_count == 0);
  b2Assert(count >= 2);
```



```
  for (int32 i = 1; i < count; ++i)
  {
    // If the code crashes here, it means your vertices are too close together.
    b2Assert(b2DistanceSquared(vertices[i-1], vertices[i]) > b2_linearSlop * b2_
linearSlop);
  }

  m_count = count;
  m_vertices = (b2Vec2*)b2Alloc(count * sizeof(b2Vec2));
  memcpy(m_vertices, vertices, m_count * sizeof(b2Vec2));

  m_hasPrevVertex = false;
  m_hasNextVertex = false;

  m_prevVertex.SetZero();
  m_nextVertex.SetZero();
}

void b2ChainShape::SetPrevVertex(const b2Vec2& prevVertex)
{
  m_prevVertex = prevVertex;
  m_hasPrevVertex = true;
}

void b2ChainShape::SetNextVertex(const b2Vec2& nextVertex)
{
  m_nextVertex = nextVertex;
  m_hasNextVertex = true;
}

b2Shape* b2ChainShape::Clone(b2BlockAllocator* allocator) const
{
  void* mem = allocator->Allocate(sizeof(b2ChainShape));
  b2ChainShape* clone = new (mem) b2ChainShape;
  clone->CreateChain(m_vertices, m_count);
  clone->m_prevVertex = m_prevVertex;
  clone->m_nextVertex = m_nextVertex;
  clone->m_hasPrevVertex = m_hasPrevVertex;
  clone->m_hasNextVertex = m_hasNextVertex;
  return clone;
}

int32 b2ChainShape::GetChildCount() const
{
  // edge count = vertex count - 1
  return m_count - 1;
}

void b2ChainShape::GetChildEdge(b2EdgeShape* edge, int32 index) const
{
  b2Assert(0 <= index && index < m_count - 1);
  edge->m_type = b2Shape::e_edge;
  edge->m_radius = m_radius;

  edge->m_vertex1 = m_vertices[index + 0];
  edge->m_vertex2 = m_vertices[index + 1];

  if (index > 0)
  {
    edge->m_vertex0 = m_vertices[index - 1];
    edge->m_hasVertex0 = true;
  }
  else
  {
    edge->m_vertex0 = m_prevVertex;
    edge->m_hasVertex0 = m_hasPrevVertex;
```

```
132   }
133
134   if (index < m_count - 2)
135   {
136     edge->m_vertex3 = m_vertices[index + 2];
137     edge->m_hasVertex3 = true;
138   }
139   else
140   {
141     edge->m_vertex3 = m_nextVertex;
142     edge->m_hasVertex3 = m_hasNextVertex;
143   }
144 }
145
146 bool b2ChainShape::TestPoint(const b2Transform& xf, const b2Vec2& p) const
147 {
148   B2_NOT_USED(xf);
149   B2_NOT_USED(p);
150   return false;
151 }
152
153 bool b2ChainShape::RayCast(b2RayCastOutput* output, const b2RayCastInput& input,
154                const b2Transform& xf, int32 childIndex) const
155 {
156   b2Assert(childIndex < m_count);
157
158   b2EdgeShape edgeShape;
159
160   int32 i1 = childIndex;
161   int32 i2 = childIndex + 1;
162   if (i2 == m_count)
163   {
164     i2 = 0;
165   }
166
167   edgeShape.m_vertex1 = m_vertices[i1];
168   edgeShape.m_vertex2 = m_vertices[i2];
169
170   return edgeShape.RayCast(output, input, xf, 0);
171 }
172
173 void b2ChainShape::ComputeAABB(b2AABB* aabb, const b2Transform& xf, int32 childI
    ndex) const
174 {
175   b2Assert(childIndex < m_count);
176
177   int32 i1 = childIndex;
178   int32 i2 = childIndex + 1;
179   if (i2 == m_count)
180   {
181     i2 = 0;
182   }
183
184   b2Vec2 v1 = b2Mul(xf, m_vertices[i1]);
185   b2Vec2 v2 = b2Mul(xf, m_vertices[i2]);
186
187   aabb->lowerBound = b2Min(v1, v2);
188   aabb->upperBound = b2Max(v1, v2);
189 }
190
191 void b2ChainShape::ComputeMass(b2MassData* massData, float32 density) const
192 {
193   B2_NOT_USED(density);
194
195   massData->mass = 0.0f;
196   massData->center.SetZero();
```



```
197   massData->I = 0.0f;
198 }
```

```
/*
* Copyright (c) 2007-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Collision/b2Distance.h"
#include "Box2D/Collision/b2TimeOfImpact.h"
#include "Box2D/Collision/Shapes/b2CircleShape.h"
#include "Box2D/Collision/Shapes/b2PolygonShape.h"
#include "Box2D/Common/b2Timer.h"

#include <stdio.h>

float32 b2_toiTime, b2_toiMaxTime;
int32 b2_toiCalls, b2_toiIters, b2_toiMaxIters;
int32 b2_toiRootIters, b2_toiMaxRootIters;

//
struct b2SeparationFunction
{
  enum Type
  {
    e_points,
    e_faceA,
    e_faceB
  };



  float32 Initialize(const b2SimplexCache* cache,
    const b2DistanceProxy* proxyA, const b2Sweep& sweepA,
    const b2DistanceProxy* proxyB, const b2Sweep& sweepB,
    float32 t1)
  {
    m_proxyA = proxyA;
    m_proxyB = proxyB;
    int32 count = cache->count;
    b2Assert(0 < count && count < 3);

    m_sweepA = sweepA;
    m_sweepB = sweepB;

    b2Transform xfA, xfB;
    m_sweepA.GetTransform(&xfA, t1);
    m_sweepB.GetTransform(&xfB, t1);

    if (count == 1)
    {
      m_type = e_points;
      b2Vec2 localPointA = m_proxyA->GetVertex(cache->indexA[0]);
      b2Vec2 localPointB = m_proxyB->GetVertex(cache->indexB[0]);
      b2Vec2 pointA = b2Mul(xfA, localPointA);
```



```
      b2Vec2 pointB = b2Mul(xfB, localPointB);
      m_axis = pointB - pointA;
      float32 s = m_axis.Normalize();
      return s;
    }
    else if (cache->indexA[0] == cache->indexA[1])
    {
      // Two points on B and one on A.
      m_type = e_faceB;
      b2Vec2 localPointB1 = proxyB->GetVertex(cache->indexB[0]);
      b2Vec2 localPointB2 = proxyB->GetVertex(cache->indexB[1]);

      m_axis = b2Cross(localPointB2 - localPointB1, 1.0f);
      m_axis.Normalize();
      b2Vec2 normal = b2Mul(xfB.q, m_axis);

      m_localPoint = 0.5f * (localPointB1 + localPointB2);
      b2Vec2 pointB = b2Mul(xfB, m_localPoint);

      b2Vec2 localPointA = proxyA->GetVertex(cache->indexA[0]);
      b2Vec2 pointA = b2Mul(xfA, localPointA);

      float32 s = b2Dot(pointA - pointB, normal);
      if (s < 0.0f)
      {
        m_axis = -m_axis;
        s = -s;
      }
      return s;
    }
    else
    {
      // Two points on A and one or two points on B.
      m_type = e_faceA;
      b2Vec2 localPointA1 = m_proxyA->GetVertex(cache->indexA[0]);
      b2Vec2 localPointA2 = m_proxyA->GetVertex(cache->indexA[1]);

      m_axis = b2Cross(localPointA2 - localPointA1, 1.0f);
      m_axis.Normalize();
      b2Vec2 normal = b2Mul(xfA.q, m_axis);

      m_localPoint = 0.5f * (localPointA1 + localPointA2);
      b2Vec2 pointA = b2Mul(xfA, m_localPoint);

      b2Vec2 localPointB = m_proxyB->GetVertex(cache->indexB[0]);
      b2Vec2 pointB = b2Mul(xfB, localPointB);

      float32 s = b2Dot(pointB - pointA, normal);
      if (s < 0.0f)
      {
        m_axis = -m_axis;
        s = -s;
      }
      return s;
    }
  }

  //
  float32 FindMinSeparation(int32* indexA, int32* indexB, float32 t) const
  {
    b2Transform xfA, xfB;
    m_sweepA.GetTransform(&xfA, t);
    m_sweepB.GetTransform(&xfB, t);

    switch (m_type)
    {
```

```
133       case e_points:
134         {
135           b2Vec2 axisA = b2MulT(xfA.q,  m_axis);
136           b2Vec2 axisB = b2MulT(xfB.q, -m_axis);
137
138           *indexA = m_proxyA->GetSupport(axisA);
139           *indexB = m_proxyB->GetSupport(axisB);
140
141           b2Vec2 localPointA = m_proxyA->GetVertex(*indexA);
142           b2Vec2 localPointB = m_proxyB->GetVertex(*indexB);
143
144           b2Vec2 pointA = b2Mul(xfA, localPointA);
145           b2Vec2 pointB = b2Mul(xfB, localPointB);
146
147           float32 separation = b2Dot(pointB - pointA, m_axis);
148           return separation;
149         }
150
151       case e_faceA:
152         {
153           b2Vec2 normal = b2Mul(xfA.q, m_axis);
154           b2Vec2 pointA = b2Mul(xfA, m_localPoint);
155
156           b2Vec2 axisB = b2MulT(xfB.q, -normal);
157
158           *indexA = -1;
159           *indexB = m_proxyB->GetSupport(axisB);
160
161           b2Vec2 localPointB = m_proxyB->GetVertex(*indexB);
162           b2Vec2 pointB = b2Mul(xfB, localPointB);
163
164           float32 separation = b2Dot(pointB - pointA, normal);
165           return separation;
166         }
167
168       case e_faceB:
169         {
170           b2Vec2 normal = b2Mul(xfB.q, m_axis);
171           b2Vec2 pointB = b2Mul(xfB, m_localPoint);
172
173           b2Vec2 axisA = b2MulT(xfA.q, -normal);
174
175           *indexB = -1;
176           *indexA = m_proxyA->GetSupport(axisA);
177
178           b2Vec2 localPointA = m_proxyA->GetVertex(*indexA);
179           b2Vec2 pointA = b2Mul(xfA, localPointA);
180
181           float32 separation = b2Dot(pointA - pointB, normal);
182           return separation;
183         }
184
185       default:
186         b2Assert(false);
187         *indexA = -1;
188         *indexB = -1;
189         return 0.0f;
190       }
191     }
192
193     //
194     float32 Evaluate(int32 indexA, int32 indexB, float32 t) const
195     {
196       b2Transform xfA, xfB;
197       m_sweepA.GetTransform(&xfA, t);
198       m_sweepB.GetTransform(&xfB, t);
```



```
199
200       switch (m_type)
201       {
202       case e_points:
203         {
204           b2Vec2 localPointA = m_proxyA->GetVertex(indexA);
205           b2Vec2 localPointB = m_proxyB->GetVertex(indexB);
206
207           b2Vec2 pointA = b2Mul(xfA, localPointA);
208           b2Vec2 pointB = b2Mul(xfB, localPointB);
209           float32 separation = b2Dot(pointB - pointA, m_axis);
210
211           return separation;
212         }
213
214       case e_faceA:
215         {
216           b2Vec2 normal = b2Mul(xfA.q, m_axis);
217           b2Vec2 pointA = b2Mul(xfA, m_localPoint);
218
219           b2Vec2 localPointB = m_proxyB->GetVertex(indexB);
220           b2Vec2 pointB = b2Mul(xfB, localPointB);
221
222           float32 separation = b2Dot(pointB - pointA, normal);
223           return separation;
224         }
225
226       case e_faceB:
227         {
228           b2Vec2 normal = b2Mul(xfB.q, m_axis);
229           b2Vec2 pointB = b2Mul(xfB, m_localPoint);
230
231           b2Vec2 localPointA = m_proxyA->GetVertex(indexA);
232           b2Vec2 pointA = b2Mul(xfA, localPointA);
233
234           float32 separation = b2Dot(pointA - pointB, normal);
235           return separation;
236         }
237
238       default:
239         b2Assert(false);
240         return 0.0f;
241       }
242     }
243
244     const b2DistanceProxy* m_proxyA;
245     const b2DistanceProxy* m_proxyB;
246     b2Sweep m_sweepA, m_sweepB;
247     Type m_type;
248     b2Vec2 m_localPoint;
249     b2Vec2 m_axis;
250   };
251
252   // CCD via the local separating axis method. This seeks progression
253   // by computing the largest time at which separation is maintained.
254   void b2TimeOfImpact(b2TOIOutput* output, const b2TOIInput* input)
255   {
256     b2Timer timer;
257
258     ++b2_toiCalls;
259
260     output->state = b2TOIOutput::e_unknown;
261     output->t = input->tMax;
262
263     const b2DistanceProxy* proxyA = &input->proxyA;
264     const b2DistanceProxy* proxyB = &input->proxyB;
```

```
265
266     b2Sweep sweepA = input→sweepA;
267     b2Sweep sweepB = input→sweepB;
268
269     // Large rotations can make the root finder fail, so we normalize the
270     // sweep angles.
271     sweepA.Normalize();
272     sweepB.Normalize();
273
274     float32 tMax = input→tMax;
275
276     float32 totalRadius = proxyA→m_radius + proxyB→m_radius;
277     float32 target = b2Max(b2_linearSlop, totalRadius − 3.0f * b2_linearSlop);
278     float32 tolerance = 0.25f * b2_linearSlop;
279     b2Assert(target > tolerance);
280
281     float32 t1 = 0.0f;
282     const int32 k_maxIterations = 20;
283     int32 iter = 0;
284
285     // Prepare input for distance query.
286     b2SimplexCache cache;
287     cache.count = 0;
288     b2DistanceInput distanceInput;
289     distanceInput.proxyA = input→proxyA;
290     distanceInput.proxyB = input→proxyB;
291     distanceInput.useRadii = false;
292
293     // The outer loop progressively attempts to compute new separating axes.
294     // This loop terminates when an axis is repeated (no progress is made).
295     for(;;)
296     {
297       b2Transform xfA, xfB;
298       sweepA.GetTransform(&xfA, t1);
299       sweepB.GetTransform(&xfB, t1);
300
301       // Get the distance between shapes. We can also use the results
302       // to get a separating axis.
303       distanceInput.transformA = xfA;
304       distanceInput.transformB = xfB;
305       b2DistanceOutput distanceOutput;
306       b2Distance(&distanceOutput, &cache, &distanceInput);
307
308       // If the shapes are overlapped, we give up on continuous collision.
309       if (distanceOutput.distance ≤ 0.0f)
310       {
311         // Failure!
312         output→state = b2TOIOutput::e_overlapped;
313         output→t = 0.0f;
314         break;
315       }
316
317       if (distanceOutput.distance < target + tolerance)
318       {
319         // Victory!
320         output→state = b2TOIOutput::e_touching;
321         output→t = t1;
322         break;
323       }
324
325       // Initialize the separating axis.
326       b2SeparationFunction fcn;
327       fcn.Initialize(&cache, proxyA, sweepA, proxyB, sweepB, t1);
328 #if 0
329       // Dump the curve seen by the root finder
330       {
```



```
331         const int32 N = 100;
332         float32 dx = 1.0f / N;
333         float32 xs[N+1];
334         float32 fs[N+1];
335
336         float32 x = 0.0f;
337
338         for (int32 i = 0; i ≤ N; ++i)
339         {
340           sweepA.GetTransform(&xfA, x);
341           sweepB.GetTransform(&xfB, x);
342           float32 f = fcn.Evaluate(xfA, xfB) − target;
343
344           printf("%g %g\n", x, f);
345
346           xs[i] = x;
347           fs[i] = f;
348
349           x += dx;
350         }
351       }
352 #endif
353
354       // Compute the TOI on the separating axis. We do this by successively
355       // resolving the deepest point. This loop is bounded by the number of vertic
    es.
356       bool done = false;
357       float32 t2 = tMax;
358       int32 pushBackIter = 0;
359       for (;;)
360       {
361         // Find the deepest point at t2. Store the witness point indices.
362         int32 indexA, indexB;
363         float32 s2 = fcn.FindMinSeparation(&indexA, &indexB, t2);
364
365         // Is the final configuration separated?
366         if (s2 > target + tolerance)
367         {
368           // Victory!
369           output→state = b2TOIOutput::e_separated;
370           output→t = tMax;
371           done = true;
372           break;
373         }
374
375         // Has the separation reached tolerance?
376         if (s2 > target − tolerance)
377         {
378           // Advance the sweeps
379           t1 = t2;
380           break;
381         }
382
383         // Compute the initial separation of the witness points.
384         float32 s1 = fcn.Evaluate(indexA, indexB, t1);
385
386         // Check for initial overlap. This might happen if the root finder
387         // runs out of iterations.
388         if (s1 < target − tolerance)
389         {
390           output→state = b2TOIOutput::e_failed;
391           output→t = t1;
392           done = true;
393           break;
394         }
395
```

```
396         // Check for touching
397         if (s1 ≤ target + tolerance)
398         {
399           // Victory! t1 should hold the TOI (could be 0.0).
400           output→state = b2TOIOutput::e_touching;
401           output→t = t1;
402           done = true;
403           break;
404         }
405
406         // Compute 1D root of: f(x) − target = 0
407         int32 rootIterCount = 0;
408         float32 a1 = t1, a2 = t2;
409         for (;;)
410         {
411           // Use a mix of the secant rule and bisection.
412           float32 t;
413           if (rootIterCount & 1)
414           {
415             // Secant rule to improve convergence.
416             t = a1 + (target − s1) * (a2 − a1) / (s2 − s1);
417           }
418           else
419           {
420             // Bisection to guarantee progress.
421             t = 0.5f * (a1 + a2);
422           }
423
424           ++rootIterCount;
425           ++b2_toiRootIters;
426
427           float32 s = fcn.Evaluate(indexA, indexB, t);
428
429           if (b2Abs(s − target) < tolerance)
430           {
431             // t2 holds a tentative value for t1
432             t2 = t;
433             break;
434           }
435
436           // Ensure we continue to bracket the root.
437           if (s > target)
438           {
439             a1 = t;
440             s1 = s;
441           }
442           else
443           {
444             a2 = t;
445             s2 = s;
446           }
447
448           if (rootIterCount ≡ 50)
449           {
450             break;
451           }
452         }
453
454         b2_toiMaxRootIters = b2Max(b2_toiMaxRootIters, rootIterCount);
455
456         ++pushBackIter;
457
458         if (pushBackIter ≡ b2_maxPolygonVertices)
459         {
460           break;
461         }
```



```
462       }
463
464       ++iter;
465       ++b2_toiIters;
466
467       if (done)
468       {
469         break;
470       }
471
472       if (iter ≡ k_maxIterations)
473       {
474         // Root finder got stuck. Semi−victory.
475         output→state = b2TOIOutput::e_failed;
476         output→t = t1;
477         break;
478       }
479     }
480
481     b2_toiMaxIters = b2Max(b2_toiMaxIters, iter);
482
483     float32 time = timer.GetMilliseconds();
484     b2_toiMaxTime = b2Max(b2_toiMaxTime, time);
485     b2_toiTime += time;
486   }
```

```
/*
* Copyright (c) 2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Collision/b2DynamicTree.h"
#include <string.h>

b2DynamicTree::b2DynamicTree()
{
  m_root = b2_nullNode;

  m_nodeCapacity = 16;
  m_nodeCount = 0;
  m_nodes = (b2TreeNode*)b2Alloc(m_nodeCapacity * sizeof(b2TreeNode));
  memset(m_nodes, 0, m_nodeCapacity * sizeof(b2TreeNode));

  // Build a linked list for the free list.
  for (int32 i = 0; i < m_nodeCapacity - 1; ++i)
  {
    m_nodes[i].next = i + 1;
    m_nodes[i].height = -1;
  }
  m_nodes[m_nodeCapacity-1].next = b2_nullNode;
  m_nodes[m_nodeCapacity-1].height = -1;
  m_freeList = 0;

  m_path = 0;

  m_insertionCount = 0;
}

b2DynamicTree::~b2DynamicTree()
{
  // This frees the entire tree in one shot.
  b2Free(m_nodes);
}

// Allocate a node from the pool. Grow the pool if necessary.
int32 b2DynamicTree::AllocateNode()
{
  // Expand the node pool as needed.
  if (m_freeList == b2_nullNode)
  {
    b2Assert(m_nodeCount == m_nodeCapacity);

    // The free list is empty. Rebuild a bigger pool.
    b2TreeNode* oldNodes = m_nodes;
    m_nodeCapacity *= 2;
    m_nodes = (b2TreeNode*)b2Alloc(m_nodeCapacity * sizeof(b2TreeNode));
    memcpy(m_nodes, oldNodes, m_nodeCount * sizeof(b2TreeNode));
    b2Free(oldNodes);

```



```
    // Build a linked list for the free list. The parent
    // pointer becomes the "next" pointer.
    for (int32 i = m_nodeCount; i < m_nodeCapacity - 1; ++i)
    {
      m_nodes[i].next = i + 1;
      m_nodes[i].height = -1;
    }
    m_nodes[m_nodeCapacity-1].next = b2_nullNode;
    m_nodes[m_nodeCapacity-1].height = -1;
    m_freeList = m_nodeCount;
  }

  // Peel a node off the free list.
  int32 nodeId = m_freeList;
  m_freeList = m_nodes[nodeId].next;
  m_nodes[nodeId].parent = b2_nullNode;
  m_nodes[nodeId].child1 = b2_nullNode;
  m_nodes[nodeId].child2 = b2_nullNode;
  m_nodes[nodeId].height = 0;
  m_nodes[nodeId].userData = nullptr;
  ++m_nodeCount;
  return nodeId;
}

// Return a node to the pool.
void b2DynamicTree::FreeNode(int32 nodeId)
{
  b2Assert(0 ≤ nodeId ∧ nodeId < m_nodeCapacity);
  b2Assert(0 < m_nodeCount);
  m_nodes[nodeId].next = m_freeList;
  m_nodes[nodeId].height = -1;
  m_freeList = nodeId;
  --m_nodeCount;
}

// Create a proxy in the tree as a leaf node. We return the index
// of the node instead of a pointer so that we can grow
// the node pool.
int32 b2DynamicTree::CreateProxy(const b2AABB& aabb, void* userData)
{
  int32 proxyId = AllocateNode();

  // Fatten the aabb.
  b2Vec2 r(b2_aabbExtension, b2_aabbExtension);
  m_nodes[proxyId].aabb.lowerBound = aabb.lowerBound - r;
  m_nodes[proxyId].aabb.upperBound = aabb.upperBound + r;
  m_nodes[proxyId].userData = userData;
  m_nodes[proxyId].height = 0;

  InsertLeaf(proxyId);

  return proxyId;
}

void b2DynamicTree::DestroyProxy(int32 proxyId)
{
  b2Assert(0 ≤ proxyId ∧ proxyId < m_nodeCapacity);
  b2Assert(m_nodes[proxyId].IsLeaf());

  RemoveLeaf(proxyId);
  FreeNode(proxyId);
}

bool b2DynamicTree::MoveProxy(int32 proxyId, const b2AABB& aabb, const b2Vec2& d
isplacement)
{
```

```
132   b2Assert(0 ≤ proxyId ∧ proxyId < m_nodeCapacity);
133
134   b2Assert(m_nodes[proxyId].IsLeaf());
135
136   if (m_nodes[proxyId].aabb.Contains(aabb))
137   {
138     return false;
139   }
140
141   RemoveLeaf(proxyId);
142
143   // Extend AABB.
144   b2AABB b = aabb;
145   b2Vec2 r(b2_aabbExtension, b2_aabbExtension);
146   b.lowerBound = b.lowerBound − r;
147   b.upperBound = b.upperBound + r;
148
149   // Predict AABB displacement.
150   b2Vec2 d = b2_aabbMultiplier * displacement;
151
152   if (d.x < 0.0f)
153   {
154     b.lowerBound.x += d.x;
155   }
156   else
157   {
158     b.upperBound.x += d.x;
159   }
160
161   if (d.y < 0.0f)
162   {
163     b.lowerBound.y += d.y;
164   }
165   else
166   {
167     b.upperBound.y += d.y;
168   }
169
170   m_nodes[proxyId].aabb = b;
171
172   InsertLeaf(proxyId);
173   return true;
174 }
175
176 void b2DynamicTree::InsertLeaf(int32 leaf)
177 {
178   ++m_insertionCount;
179
180   if (m_root ≡ b2_nullNode)
181   {
182     m_root = leaf;
183     m_nodes[m_root].parent = b2_nullNode;
184     return;
185   }
186
187   // Find the best sibling for this node
188   b2AABB leafAABB = m_nodes[leaf].aabb;
189   int32 index = m_root;
190   while (m_nodes[index].IsLeaf() ≡ false)
191   {
192     int32 child1 = m_nodes[index].child1;
193     int32 child2 = m_nodes[index].child2;
194
195     float32 area = m_nodes[index].aabb.GetPerimeter();
196
197     b2AABB combinedAABB;
```



```
198     combinedAABB.Combine(m_nodes[index].aabb, leafAABB);
199     float32 combinedArea = combinedAABB.GetPerimeter();
200
201     // Cost of creating a new parent for this node and the new leaf
202     float32 cost = 2.0f * combinedArea;
203
204     // Minimum cost of pushing the leaf further down the tree
205     float32 inheritanceCost = 2.0f * (combinedArea − area);
206
207     // Cost of descending into child1
208     float32 cost1;
209     if (m_nodes[child1].IsLeaf())
210     {
211       b2AABB aabb;
212       aabb.Combine(leafAABB, m_nodes[child1].aabb);
213       cost1 = aabb.GetPerimeter() + inheritanceCost;
214     }
215     else
216     {
217       b2AABB aabb;
218       aabb.Combine(leafAABB, m_nodes[child1].aabb);
219       float32 oldArea = m_nodes[child1].aabb.GetPerimeter();
220       float32 newArea = aabb.GetPerimeter();
221       cost1 = (newArea − oldArea) + inheritanceCost;
222     }
223
224     // Cost of descending into child2
225     float32 cost2;
226     if (m_nodes[child2].IsLeaf())
227     {
228       b2AABB aabb;
229       aabb.Combine(leafAABB, m_nodes[child2].aabb);
230       cost2 = aabb.GetPerimeter() + inheritanceCost;
231     }
232     else
233     {
234       b2AABB aabb;
235       aabb.Combine(leafAABB, m_nodes[child2].aabb);
236       float32 oldArea = m_nodes[child2].aabb.GetPerimeter();
237       float32 newArea = aabb.GetPerimeter();
238       cost2 = newArea − oldArea + inheritanceCost;
239     }
240
241     // Descend according to the minimum cost.
242     if (cost < cost1 ∧ cost < cost2)
243     {
244       break;
245     }
246
247     // Descend
248     if (cost1 < cost2)
249     {
250       index = child1;
251     }
252     else
253     {
254       index = child2;
255     }
256   }
257
258   int32 sibling = index;
259
260   // Create a new parent.
261   int32 oldParent = m_nodes[sibling].parent;
262   int32 newParent = AllocateNode();
263   m_nodes[newParent].parent = oldParent;
```

```
264   m_nodes[newParent].userData = nullptr;
265   m_nodes[newParent].aabb.Combine(leafAABB, m_nodes[sibling].aabb);
266   m_nodes[newParent].height = m_nodes[sibling].height + 1;
267
268   if (oldParent ≠ b2_nullNode)
269   {
270     // The sibling was not the root.
271     if (m_nodes[oldParent].child1 ≡ sibling)
272     {
273       m_nodes[oldParent].child1 = newParent;
274     }
275     else
276     {
277       m_nodes[oldParent].child2 = newParent;
278     }
279
280     m_nodes[newParent].child1 = sibling;
281     m_nodes[newParent].child2 = leaf;
282     m_nodes[sibling].parent = newParent;
283     m_nodes[leaf].parent = newParent;
284   }
285   else
286   {
287     // The sibling was the root.
288     m_nodes[newParent].child1 = sibling;
289     m_nodes[newParent].child2 = leaf;
290     m_nodes[sibling].parent = newParent;
291     m_nodes[leaf].parent = newParent;
292     m_root = newParent;
293   }
294
295   // Walk back up the tree fixing heights and AABBs
296   index = m_nodes[leaf].parent;
297   while (index ≠ b2_nullNode)
298   {
299     index = Balance(index);
300
301     int32 child1 = m_nodes[index].child1;
302     int32 child2 = m_nodes[index].child2;
303
304     b2Assert(child1 ≠ b2_nullNode);
305     b2Assert(child2 ≠ b2_nullNode);
306
307     m_nodes[index].height = 1 + b2Max(m_nodes[child1].height, m_nodes[child2].he
    ight);
308     m_nodes[index].aabb.Combine(m_nodes[child1].aabb, m_nodes[child2].aabb);
309
310     index = m_nodes[index].parent;
311   }
312
313   //Validate();
314 }
315
316 void b2DynamicTree::RemoveLeaf(int32 leaf)
317 {
318   if (leaf ≡ m_root)
319   {
320     m_root = b2_nullNode;
321     return;
322   }
323
324   int32 parent = m_nodes[leaf].parent;
325   int32 grandParent = m_nodes[parent].parent;
326   int32 sibling;
327   if (m_nodes[parent].child1 ≡ leaf)
328   {
```



```
329     sibling = m_nodes[parent].child2;
330   }
331   else
332   {
333     sibling = m_nodes[parent].child1;
334   }
335
336   if (grandParent ≠ b2_nullNode)
337   {
338     // Destroy parent and connect sibling to grandParent.
339     if (m_nodes[grandParent].child1 ≡ parent)
340     {
341       m_nodes[grandParent].child1 = sibling;
342     }
343     else
344     {
345       m_nodes[grandParent].child2 = sibling;
346     }
347     m_nodes[sibling].parent = grandParent;
348     FreeNode(parent);
349
350     // Adjust ancestor bounds.
351     int32 index = grandParent;
352     while (index ≠ b2_nullNode)
353     {
354       index = Balance(index);
355
356       int32 child1 = m_nodes[index].child1;
357       int32 child2 = m_nodes[index].child2;
358
359       m_nodes[index].aabb.Combine(m_nodes[child1].aabb, m_nodes[child2].aabb);
360       m_nodes[index].height = 1 + b2Max(m_nodes[child1].height, m_nodes[child2].
    height);
361
362       index = m_nodes[index].parent;
363     }
364   }
365   else
366   {
367     m_root = sibling;
368     m_nodes[sibling].parent = b2_nullNode;
369     FreeNode(parent);
370   }
371
372   //Validate();
373 }
374
375 // Perform a left or right rotation if node A is imbalanced.
376 // Returns the new root index.
377 int32 b2DynamicTree::Balance(int32 iA)
378 {
379   b2Assert(iA ≠ b2_nullNode);
380
381   b2TreeNode* A = m_nodes + iA;
382   if (A→IsLeaf() ∨ A→height < 2)
383   {
384     return iA;
385   }
386
387   int32 iB = A→child1;
388   int32 iC = A→child2;
389   b2Assert(0 ≤ iB ∧ iB < m_nodeCapacity);
390   b2Assert(0 ≤ iC ∧ iC < m_nodeCapacity);
391
392   b2TreeNode* B = m_nodes + iB;
393   b2TreeNode* C = m_nodes + iC;
```

```
394
395     int32 balance = C→height − B→height;
396
397     // Rotate C up
398     if (balance > 1)
399     {
400       int32 iF = C→child1;
401       int32 iG = C→child2;
402       b2TreeNode* F = m_nodes + iF;
403       b2TreeNode* G = m_nodes + iG;
404       b2Assert(0 ≤ iF ∧ iF < m_nodeCapacity);
405       b2Assert(0 ≤ iG ∧ iG < m_nodeCapacity);
406
407       // Swap A and C
408       C→child1 = iA;
409       C→parent = A→parent;
410       A→parent = iC;
411
412       // A's old parent should point to C
413       if (C→parent ≠ b2_nullNode)
414       {
415         if (m_nodes[C→parent].child1 ≡ iA)
416         {
417           m_nodes[C→parent].child1 = iC;
418         }
419         else
420         {
421           b2Assert(m_nodes[C→parent].child2 ≡ iA);
422           m_nodes[C→parent].child2 = iC;
423         }
424       }
425       else
426       {
427         m_root = iC;
428       }
429
430       // Rotate
431       if (F→height > G→height)
432       {
433         C→child2 = iF;
434         A→child2 = iG;
435         G→parent = iA;
436         A→aabb.Combine(B→aabb, G→aabb);
437         C→aabb.Combine(A→aabb, F→aabb);
438
439         A→height = 1 + b2Max(B→height, G→height);
440         C→height = 1 + b2Max(A→height, F→height);
441       }
442       else
443       {
444         C→child2 = iG;
445         A→child2 = iF;
446         F→parent = iA;
447         A→aabb.Combine(B→aabb, F→aabb);
448         C→aabb.Combine(A→aabb, G→aabb);
449
450         A→height = 1 + b2Max(B→height, F→height);
451         C→height = 1 + b2Max(A→height, G→height);
452       }
453
454       return iC;
455     }
456
457     // Rotate B up
458     if (balance < −1)
459     {
```



```
460       int32 iD = B→child1;
461       int32 iE = B→child2;
462       b2TreeNode* D = m_nodes + iD;
463       b2TreeNode* E = m_nodes + iE;
464       b2Assert(0 ≤ iD ∧ iD < m_nodeCapacity);
465       b2Assert(0 ≤ iE ∧ iE < m_nodeCapacity);
466
467       // Swap A and B
468       B→child1 = iA;
469       B→parent = A→parent;
470       A→parent = iB;
471
472       // A's old parent should point to B
473       if (B→parent ≠ b2_nullNode)
474       {
475         if (m_nodes[B→parent].child1 ≡ iA)
476         {
477           m_nodes[B→parent].child1 = iB;
478         }
479         else
480         {
481           b2Assert(m_nodes[B→parent].child2 ≡ iA);
482           m_nodes[B→parent].child2 = iB;
483         }
484       }
485       else
486       {
487         m_root = iB;
488       }
489
490       // Rotate
491       if (D→height > E→height)
492       {
493         B→child2 = iD;
494         A→child1 = iE;
495         E→parent = iA;
496         A→aabb.Combine(C→aabb, E→aabb);
497         B→aabb.Combine(A→aabb, D→aabb);
498
499         A→height = 1 + b2Max(C→height, E→height);
500         B→height = 1 + b2Max(A→height, D→height);
501       }
502       else
503       {
504         B→child2 = iE;
505         A→child1 = iD;
506         D→parent = iA;
507         A→aabb.Combine(C→aabb, D→aabb);
508         B→aabb.Combine(A→aabb, E→aabb);
509
510         A→height = 1 + b2Max(C→height, D→height);
511         B→height = 1 + b2Max(A→height, E→height);
512       }
513
514       return iB;
515     }
516
517     return iA;
518   }
519
520   int32 b2DynamicTree::GetHeight() const
521   {
522     if (m_root ≡ b2_nullNode)
523     {
524       return 0;
525     }
```

```
526
527   return m_nodes[m_root].height;
528 }
529
530 //
531 float32 b2DynamicTree::GetAreaRatio() const
532 {
533   if (m_root ≡ b2_nullNode)
534   {
535     return 0.0f;
536   }
537
538   const b2TreeNode* root = m_nodes + m_root;
539   float32 rootArea = root→aabb.GetPerimeter();
540
541   float32 totalArea = 0.0f;
542   for (int32 i = 0; i < m_nodeCapacity; ++i)
543   {
544     const b2TreeNode* node = m_nodes + i;
545     if (node→height < 0)
546     {
547       // Free node in pool
548       continue;
549     }
550
551     totalArea += node→aabb.GetPerimeter();
552   }
553
554   return totalArea / rootArea;
555 }
556
557 // Compute the height of a sub-tree.
558 int32 b2DynamicTree::ComputeHeight(int32 nodeId) const
559 {
560   b2Assert(0 ≤ nodeId ∧ nodeId < m_nodeCapacity);
561   b2TreeNode* node = m_nodes + nodeId;
562
563   if (node→IsLeaf())
564   {
565     return 0;
566   }
567
568   int32 height1 = ComputeHeight(node→child1);
569   int32 height2 = ComputeHeight(node→child2);
570   return 1 + b2Max(height1, height2);
571 }
572
573 int32 b2DynamicTree::ComputeHeight() const
574 {
575   int32 height = ComputeHeight(m_root);
576   return height;
577 }
578
579 void b2DynamicTree::ValidateStructure(int32 index) const
580 {
581   if (index ≡ b2_nullNode)
582   {
583     return;
584   }
585
586   if (index ≡ m_root)
587   {
588     b2Assert(m_nodes[index].parent ≡ b2_nullNode);
589   }
590
591   const b2TreeNode* node = m_nodes + index;
```



```
592
593   int32 child1 = node→child1;
594   int32 child2 = node→child2;
595
596   if (node→IsLeaf())
597   {
598     b2Assert(child1 ≡ b2_nullNode);
599     b2Assert(child2 ≡ b2_nullNode);
600     b2Assert(node→height ≡ 0);
601     return;
602   }
603
604   b2Assert(0 ≤ child1 ∧ child1 < m_nodeCapacity);
605   b2Assert(0 ≤ child2 ∧ child2 < m_nodeCapacity);
606
607   b2Assert(m_nodes[child1].parent ≡ index);
608   b2Assert(m_nodes[child2].parent ≡ index);
609
610   ValidateStructure(child1);
611   ValidateStructure(child2);
612 }
613
614 void b2DynamicTree::ValidateMetrics(int32 index) const
615 {
616   if (index ≡ b2_nullNode)
617   {
618     return;
619   }
620
621   const b2TreeNode* node = m_nodes + index;
622
623   int32 child1 = node→child1;
624   int32 child2 = node→child2;
625
626   if (node→IsLeaf())
627   {
628     b2Assert(child1 ≡ b2_nullNode);
629     b2Assert(child2 ≡ b2_nullNode);
630     b2Assert(node→height ≡ 0);
631     return;
632   }
633
634   b2Assert(0 ≤ child1 ∧ child1 < m_nodeCapacity);
635   b2Assert(0 ≤ child2 ∧ child2 < m_nodeCapacity);
636
637   int32 height1 = m_nodes[child1].height;
638   int32 height2 = m_nodes[child2].height;
639   int32 height;
640   height = 1 + b2Max(height1, height2);
641   b2Assert(node→height ≡ height);
642
643   b2AABB aabb;
644   aabb.Combine(m_nodes[child1].aabb, m_nodes[child2].aabb);
645
646   b2Assert(aabb.lowerBound ≡ node→aabb.lowerBound);
647   b2Assert(aabb.upperBound ≡ node→aabb.upperBound);
648
649   ValidateMetrics(child1);
650   ValidateMetrics(child2);
651 }
652
653 void b2DynamicTree::Validate() const
654 {
655 #if defined(b2DEBUG)
656   ValidateStructure(m_root);
657   ValidateMetrics(m_root);
```

```
658
659   int32 freeCount = 0;
660   int32 freeIndex = m_freeList;
661   while (freeIndex ≠ b2_nullNode)
662   {
663     b2Assert(0 ≤ freeIndex ∧ freeIndex < m_nodeCapacity);
664     freeIndex = m_nodes[freeIndex].next;
665     ++freeCount;
666   }
667
668   b2Assert(GetHeight() ≡ ComputeHeight());
669
670   b2Assert(m_nodeCount + freeCount ≡ m_nodeCapacity);
671 #endif
672 }
673
674 int32 b2DynamicTree::GetMaxBalance() const
675 {
676   int32 maxBalance = 0;
677   for (int32 i = 0; i < m_nodeCapacity; ++i)
678   {
679     const b2TreeNode* node = m_nodes + i;
680     if (node→height ≤ 1)
681     {
682       continue;
683     }
684
685     b2Assert(node→IsLeaf() ≡ false);
686
687     int32 child1 = node→child1;
688     int32 child2 = node→child2;
689     int32 balance = b2Abs(m_nodes[child2].height − m_nodes[child1].height);
690     maxBalance = b2Max(maxBalance, balance);
691   }
692
693   return maxBalance;
694 }
695
696 void b2DynamicTree::RebuildBottomUp()
697 {
698   int32* nodes = (int32*)b2Alloc(m_nodeCount * sizeof(int32));
699   int32 count = 0;
700
701   // Build array of leaves. Free the rest.
702   for (int32 i = 0; i < m_nodeCapacity; ++i)
703   {
704     if (m_nodes[i].height < 0)
705     {
706       // free node in pool
707       continue;
708     }
709
710     if (m_nodes[i].IsLeaf())
711     {
712       m_nodes[i].parent = b2_nullNode;
713       nodes[count] = i;
714       ++count;
715     }
716     else
717     {
718       FreeNode(i);
719     }
720   }
721
722   while (count > 1)
723   {
```



```
724     float32 minCost = b2_maxFloat;
725     int32 iMin = −1, jMin = −1;
726     for (int32 i = 0; i < count; ++i)
727     {
728       b2AABB aabbi = m_nodes[nodes[i]].aabb;
729
730       for (int32 j = i + 1; j < count; ++j)
731       {
732         b2AABB aabbj = m_nodes[nodes[j]].aabb;
733         b2AABB b;
734         b.Combine(aabbi, aabbj);
735         float32 cost = b.GetPerimeter();
736         if (cost < minCost)
737         {
738           iMin = i;
739           jMin = j;
740           minCost = cost;
741         }
742       }
743     }
744
745     int32 index1 = nodes[iMin];
746     int32 index2 = nodes[jMin];
747     b2TreeNode* child1 = m_nodes + index1;
748     b2TreeNode* child2 = m_nodes + index2;
749
750     int32 parentIndex = AllocateNode();
751     b2TreeNode* parent = m_nodes + parentIndex;
752     parent→child1 = index1;
753     parent→child2 = index2;
754     parent→height = 1 + b2Max(child1→height, child2→height);
755     parent→aabb.Combine(child1→aabb, child2→aabb);
756     parent→parent = b2_nullNode;
757
758     child1→parent = parentIndex;
759     child2→parent = parentIndex;
760
761     nodes[jMin] = nodes[count−1];
762     nodes[iMin] = parentIndex;
763     −−count;
764   }
765
766   m_root = nodes[0];
767   b2Free(nodes);
768
769   Validate();
770 }
771
772 void b2DynamicTree::ShiftOrigin(const b2Vec2& newOrigin)
773 {
774   // Build array of leaves. Free the rest.
775   for (int32 i = 0; i < m_nodeCapacity; ++i)
776   {
777     m_nodes[i].aabb.lowerBound −= newOrigin;
778     m_nodes[i].aabb.upperBound −= newOrigin;
779   }
780 }
```

```
/*
* Copyright (c) 2007-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Collision/b2Distance.h"
#include "Box2D/Collision/Shapes/b2CircleShape.h"
#include "Box2D/Collision/Shapes/b2EdgeShape.h"
#include "Box2D/Collision/Shapes/b2ChainShape.h"
#include "Box2D/Collision/Shapes/b2PolygonShape.h"

// GJK using Voronoi regions (Christer Ericson) and Barycentric coordinates.
int32 b2_gjkCalls, b2_gjkIters, b2_gjkMaxIters;

void b2DistanceProxy::Set(const b2Shape* shape, int32 index)
{
  switch (shape→GetType())
  {
  case b2Shape::e_circle:
    {
      const b2CircleShape* circle = static_cast<const b2CircleShape*>(shape);
      m_vertices = &circle→m_p;
      m_count = 1;
      m_radius = circle→m_radius;
    }
    break;

  case b2Shape::e_polygon:
    {
      const b2PolygonShape* polygon = static_cast<const b2PolygonShape*>(shape);
      m_vertices = polygon→m_vertices;
      m_count = polygon→m_count;
      m_radius = polygon→m_radius;
    }
    break;

  case b2Shape::e_chain:
    {
      const b2ChainShape* chain = static_cast<const b2ChainShape*>(shape);
      b2Assert(0 ≤ index ∧ index < chain→m_count);

      m_buffer[0] = chain→m_vertices[index];
      if (index + 1 < chain→m_count)
      {
        m_buffer[1] = chain→m_vertices[index + 1];
      }
      else
      {
        m_buffer[1] = chain→m_vertices[0];
      }

      m_vertices = m_buffer;
      m_count = 2;
```



```
      m_radius = chain→m_radius;
    }
    break;

  case b2Shape::e_edge:
    {
      const b2EdgeShape* edge = static_cast<const b2EdgeShape*>(shape);
      m_vertices = &edge→m_vertex1;
      m_count = 2;
      m_radius = edge→m_radius;
    }
    break;

  default:
    b2Assert(false);
  }
}

void b2DistanceProxy::Set(const b2Vec2* vertices, int32 count, float32 radius)
{
    m_vertices = vertices;
    m_count = count;
    m_radius = radius;
}

struct b2SimplexVertex
{
  b2Vec2 wA;    // support point in proxyA
  b2Vec2 wB;    // support point in proxyB
  b2Vec2 w;   // wB - wA
  float32 a;    // barycentric coordinate for closest point
  int32 indexA; // wA index
  int32 indexB; // wB index
};

struct b2Simplex
{
  void ReadCache( const b2SimplexCache* cache,
          const b2DistanceProxy* proxyA, const b2Transform& transformA,
          const b2DistanceProxy* proxyB, const b2Transform& transformB)
  {
    b2Assert(cache→count ≤ 3);

    // Copy data from cache.
    m_count = cache→count;
    b2SimplexVertex* vertices = &m_v1;
    for (int32 i = 0; i < m_count; ++i)
    {
      b2SimplexVertex* v = vertices + i;
      v→indexA = cache→indexA[i];
      v→indexB = cache→indexB[i];
      b2Vec2 wALocal = proxyA→GetVertex(v→indexA);
      b2Vec2 wBLocal = proxyB→GetVertex(v→indexB);
      v→wA = b2Mul(transformA, wALocal);
      v→wB = b2Mul(transformB, wBLocal);
      v→w = v→wB - v→wA;
      v→a = 0.0f;
    }

    // Compute the new simplex metric, if it is substantially different than
    // old metric then flush the simplex.
    if (m_count > 1)
    {
      float32 metric1 = cache→metric;
      float32 metric2 = GetMetric();
      if (metric2 < 0.5f * metric1 ∨ 2.0f * metric1 < metric2 ∨ metric2 < b2_ep
```

```
    silon)
133       {
134         // Reset the simplex.
135         m_count = 0;
136       }
137     }
138
139     // If the cache is empty or invalid ...
140     if (m_count == 0)
141     {
142       b2SimplexVertex* v = vertices + 0;
143       v->indexA = 0;
144       v->indexB = 0;
145       b2Vec2 wALocal = proxyA->GetVertex(0);
146       b2Vec2 wBLocal = proxyB->GetVertex(0);
147       v->wA = b2Mul(transformA, wALocal);
148       v->wB = b2Mul(transformB, wBLocal);
149       v->w = v->wB - v->wA;
150       v->a = 1.0f;
151       m_count = 1;
152     }
153   }
154
155   void WriteCache(b2SimplexCache* cache) const
156   {
157     cache->metric = GetMetric();
158     cache->count = uint16(m_count);
159     const b2SimplexVertex* vertices = &m_v1;
160     for (int32 i = 0; i < m_count; ++i)
161     {
162       cache->indexA[i] = uint8(vertices[i].indexA);
163       cache->indexB[i] = uint8(vertices[i].indexB);
164     }
165   }
166
167   b2Vec2 GetSearchDirection() const
168   {
169     switch (m_count)
170     {
171     case 1:
172       return -m_v1.w;
173
174     case 2:
175       {
176         b2Vec2 e12 = m_v2.w - m_v1.w;
177         float32 sgn = b2Cross(e12, -m_v1.w);
178         if (sgn > 0.0f)
179         {
180           // Origin is left of e12.
181           return b2Cross(1.0f, e12);
182         }
183         else
184         {
185           // Origin is right of e12.
186           return b2Cross(e12, 1.0f);
187         }
188       }
189
190     default:
191       b2Assert(false);
192       return b2Vec2_zero;
193     }
194   }
195
196   b2Vec2 GetClosestPoint() const
197   {
```



```
198     switch (m_count)
199     {
200     case 0:
201       b2Assert(false);
202       return b2Vec2_zero;
203
204     case 1:
205       return m_v1.w;
206
207     case 2:
208       return m_v1.a * m_v1.w + m_v2.a * m_v2.w;
209
210     case 3:
211       return b2Vec2_zero;
212
213     default:
214       b2Assert(false);
215       return b2Vec2_zero;
216     }
217   }
218
219   void GetWitnessPoints(b2Vec2* pA, b2Vec2* pB) const
220   {
221     switch (m_count)
222     {
223     case 0:
224       b2Assert(false);
225       break;
226
227     case 1:
228       *pA = m_v1.wA;
229       *pB = m_v1.wB;
230       break;
231
232     case 2:
233       *pA = m_v1.a * m_v1.wA + m_v2.a * m_v2.wA;
234       *pB = m_v1.a * m_v1.wB + m_v2.a * m_v2.wB;
235       break;
236
237     case 3:
238       *pA = m_v1.a * m_v1.wA + m_v2.a * m_v2.wA + m_v3.a * m_v3.wA;
239       *pB = *pA;
240       break;
241
242     default:
243       b2Assert(false);
244       break;
245     }
246   }
247
248   float32 GetMetric() const
249   {
250     switch (m_count)
251     {
252     case 0:
253       b2Assert(false);
254       return 0.0f;
255
256     case 1:
257       return 0.0f;
258
259     case 2:
260       return b2Distance(m_v1.w, m_v2.w);
261
262     case 3:
263       return b2Cross(m_v2.w - m_v1.w, m_v3.w - m_v1.w);
```

```
264
265       default:
266         b2Assert(false);
267         return 0.0f;
268       }
269     }
270
271     void Solve2();
272     void Solve3();
273
274     b2SimplexVertex m_v1, m_v2, m_v3;
275     int32 m_count;
276   };
277
278
279   // Solve a line segment using barycentric coordinates.
280   //
281   // p = a1 * w1 + a2 * w2
282   // a1 + a2 = 1
283   //
284   // The vector from the origin to the closest point on the line is
285   // perpendicular to the line.
286   // e12 = w2 - w1
287   // dot(p, e) = 0
288   // a1 * dot(w1, e) + a2 * dot(w2, e) = 0
289   //
290   // 2-by-2 linear system
291   // [1      1     ][a1] = [1]
292   // [w1.e12 w2.e12][a2] = [0]
293   //
294   // Define
295   // d12_1 =  dot(w2, e12)
296   // d12_2 = -dot(w1, e12)
297   // d12 = d12_1 + d12_2
298   //
299   // Solution
300   // a1 = d12_1 / d12
301   // a2 = d12_2 / d12
302   void b2Simplex::Solve2()
303   {
304     b2Vec2 w1 = m_v1.w;
305     b2Vec2 w2 = m_v2.w;
306     b2Vec2 e12 = w2 - w1;
307
308     // w1 region
309     float32 d12_2 = -b2Dot(w1, e12);
310     if (d12_2 ≤ 0.0f)
311     {
312       // a2 <= 0, so we clamp it to 0
313       m_v1.a = 1.0f;
314       m_count = 1;
315       return;
316     }
317
318     // w2 region
319     float32 d12_1 = b2Dot(w2, e12);
320     if (d12_1 ≤ 0.0f)
321     {
322       // a1 <= 0, so we clamp it to 0
323       m_v2.a = 1.0f;
324       m_count = 1;
325       m_v1 = m_v2;
326       return;
327     }
328
329     // Must be in e12 region.
```



```
330     float32 inv_d12 = 1.0f / (d12_1 + d12_2);
331     m_v1.a = d12_1 * inv_d12;
332     m_v2.a = d12_2 * inv_d12;
333     m_count = 2;
334   }
335
336   // Possible regions:
337   // - points[2]
338   // - edge points[0]-points[2]
339   // - edge points[1]-points[2]
340   // - inside the triangle
341   void b2Simplex::Solve3()
342   {
343     b2Vec2 w1 = m_v1.w;
344     b2Vec2 w2 = m_v2.w;
345     b2Vec2 w3 = m_v3.w;
346
347     // Edge12
348     // [1      1     ][a1] = [1]
349     // [w1.e12 w2.e12][a2] = [0]
350     // a3 = 0
351     b2Vec2 e12 = w2 - w1;
352     float32 w1e12 = b2Dot(w1, e12);
353     float32 w2e12 = b2Dot(w2, e12);
354     float32 d12_1 = w2e12;
355     float32 d12_2 = -w1e12;
356
357     // Edge13
358     // [1      1     ][a1] = [1]
359     // [w1.e13 w3.e13][a3] = [0]
360     // a2 = 0
361     b2Vec2 e13 = w3 - w1;
362     float32 w1e13 = b2Dot(w1, e13);
363     float32 w3e13 = b2Dot(w3, e13);
364     float32 d13_1 = w3e13;
365     float32 d13_2 = -w1e13;
366
367     // Edge23
368     // [1      1     ][a2] = [1]
369     // [w2.e23 w3.e23][a3] = [0]
370     // a1 = 0
371     b2Vec2 e23 = w3 - w2;
372     float32 w2e23 = b2Dot(w2, e23);
373     float32 w3e23 = b2Dot(w3, e23);
374     float32 d23_1 = w3e23;
375     float32 d23_2 = -w2e23;
376
377     // Triangle123
378     float32 n123 = b2Cross(e12, e13);
379
380     float32 d123_1 = n123 * b2Cross(w2, w3);
381     float32 d123_2 = n123 * b2Cross(w3, w1);
382     float32 d123_3 = n123 * b2Cross(w1, w2);
383
384     // w1 region
385     if (d12_2 ≤ 0.0f ∧ d13_2 ≤ 0.0f)
386     {
387       m_v1.a = 1.0f;
388       m_count = 1;
389       return;
390     }
391
392     // e12
393     if (d12_1 > 0.0f ∧ d12_2 > 0.0f ∧ d123_3 ≤ 0.0f)
394     {
395       float32 inv_d12 = 1.0f / (d12_1 + d12_2);
```

```
396       m_v1.a = d12_1 * inv_d12;
397       m_v2.a = d12_2 * inv_d12;
398       m_count = 2;
399       return;
400     }
401
402     // e13
403     if (d13_1 > 0.0f ∧ d13_2 > 0.0f ∧ d123_2 ≤ 0.0f)
404     {
405       float32 inv_d13 = 1.0f / (d13_1 + d13_2);
406       m_v1.a = d13_1 * inv_d13;
407       m_v3.a = d13_2 * inv_d13;
408       m_count = 2;
409       m_v2 = m_v3;
410       return;
411     }
412
413     // w2 region
414     if (d12_1 ≤ 0.0f ∧ d23_2 ≤ 0.0f)
415     {
416       m_v2.a = 1.0f;
417       m_count = 1;
418       m_v1 = m_v2;
419       return;
420     }
421
422     // w3 region
423     if (d13_1 ≤ 0.0f ∧ d23_1 ≤ 0.0f)
424     {
425       m_v3.a = 1.0f;
426       m_count = 1;
427       m_v1 = m_v3;
428       return;
429     }
430
431     // e23
432     if (d23_1 > 0.0f ∧ d23_2 > 0.0f ∧ d123_1 ≤ 0.0f)
433     {
434       float32 inv_d23 = 1.0f / (d23_1 + d23_2);
435       m_v2.a = d23_1 * inv_d23;
436       m_v3.a = d23_2 * inv_d23;
437       m_count = 2;
438       m_v1 = m_v3;
439       return;
440     }
441
442     // Must be in triangle123
443     float32 inv_d123 = 1.0f / (d123_1 + d123_2 + d123_3);
444     m_v1.a = d123_1 * inv_d123;
445     m_v2.a = d123_2 * inv_d123;
446     m_v3.a = d123_3 * inv_d123;
447     m_count = 3;
448   }
449
450   void b2Distance(b2DistanceOutput* output,
451           b2SimplexCache* cache,
452           const b2DistanceInput* input)
453   {
454     ++b2_gjkCalls;
455
456     const b2DistanceProxy* proxyA = &input→proxyA;
457     const b2DistanceProxy* proxyB = &input→proxyB;
458
459     b2Transform transformA = input→transformA;
460     b2Transform transformB = input→transformB;
461
```



```
462     // Initialize the simplex.
463     b2Simplex simplex;
464     simplex.ReadCache(cache, proxyA, transformA, proxyB, transformB);
465
466     // Get simplex vertices as an array.
467     b2SimplexVertex* vertices = &simplex.m_v1;
468     const int32 k_maxIters = 20;
469
470     // These store the vertices of the last simplex so that we
471     // can check for duplicates and prevent cycling.
472     int32 saveA[3], saveB[3];
473     int32 saveCount = 0;
474
475     // Main iteration loop.
476     int32 iter = 0;
477     while (iter < k_maxIters)
478     {
479       // Copy simplex so we can identify duplicates.
480       saveCount = simplex.m_count;
481       for (int32 i = 0; i < saveCount; ++i)
482       {
483         saveA[i] = vertices[i].indexA;
484         saveB[i] = vertices[i].indexB;
485       }
486
487       switch (simplex.m_count)
488       {
489       case 1:
490         break;
491
492       case 2:
493         simplex.Solve2();
494         break;
495
496       case 3:
497         simplex.Solve3();
498         break;
499
500       default:
501         b2Assert(false);
502       }
503
504       // If we have 3 points, then the origin is in the corresponding triangle.
505       if (simplex.m_count ≡ 3)
506       {
507         break;
508       }
509
510       // Get search direction.
511       b2Vec2 d = simplex.GetSearchDirection();
512
513       // Ensure the search direction is numerically fit.
514       if (d.LengthSquared() < b2_epsilon * b2_epsilon)
515       {
516         // The origin is probably contained by a line segment
517         // or triangle. Thus the shapes are overlapped.
518
519         // We can't return zero here even though there may be overlap.
520         // In case the simplex is a point, segment, or triangle it is difficult
521         // to determine if the origin is contained in the CSO or very close to it.
522         break;
523       }
524
525       // Compute a tentative new simplex vertex using support points.
526       b2SimplexVertex* vertex = vertices + simplex.m_count;
527       vertex→indexA = proxyA→GetSupport(b2MulT(transformA.q, −d));
```

```
528       vertex→wA = b2Mul(transformA, proxyA→GetVertex(vertex→indexA));
529       b2Vec2 wBLocal;
530       vertex→indexB = proxyB→GetSupport(b2MulT(transformB.q, d));
531       vertex→wB = b2Mul(transformB, proxyB→GetVertex(vertex→indexB));
532       vertex→w = vertex→wB − vertex→wA;
533
534       // Iteration count is equated to the number of support point calls.
535       ++iter;
536       ++b2_gjkIters;
537
538       // Check for duplicate support points. This is the main termination criteria
    .
539       bool duplicate = false;
540       for (int32 i = 0; i < saveCount; ++i)
541       {
542         if (vertex→indexA ≡ saveA[i] ∧ vertex→indexB ≡ saveB[i])
543         {
544           duplicate = true;
545           break;
546         }
547       }
548
549       // If we found a duplicate support point we must exit to avoid cycling.
550       if (duplicate)
551       {
552         break;
553       }
554
555       // New vertex is ok and needed.
556       ++simplex.m_count;
557     }
558
559     b2_gjkMaxIters = b2Max(b2_gjkMaxIters, iter);
560
561     // Prepare output.
562     simplex.GetWitnessPoints(&output→pointA, &output→pointB);
563     output→distance = b2Distance(output→pointA, output→pointB);
564     output→iterations = iter;
565
566     // Cache the simplex.
567     simplex.WriteCache(cache);
568
569     // Apply radii if requested.
570     if (input→useRadii)
571     {
572       float32 rA = proxyA→m_radius;
573       float32 rB = proxyB→m_radius;
574
575       if (output→distance > rA + rB ∧ output→distance > b2_epsilon)
576       {
577         // Shapes are still no overlapped.
578         // Move the witness points to the outer surface.
579         output→distance −= rA + rB;
580         b2Vec2 normal = output→pointB − output→pointA;
581         normal.Normalize();
582         output→pointA += rA * normal;
583         output→pointB −= rB * normal;
584       }
585       else
586       {
587         // Shapes are overlapped when radii are considered.
588         // Move the witness points to the middle.
589         b2Vec2 p = 0.5f * (output→pointA + output→pointB);
590         output→pointA = p;
591         output→pointB = p;
592         output→distance = 0.0f;
```



```
593       }
594     }
595   }
596
597   // GJK−raycast
598   // Algorithm by Gino van den Bergen.
599   // "Smooth Mesh Contacts with GJK" in Game Physics Pearls. 2010
600   bool b2ShapeCast(b2ShapeCastOutput * output, const b2ShapeCastInput * input)
601   {
602       output→iterations = 0;
603       output→lambda = 1.0f;
604       output→normal.SetZero();
605       output→point.SetZero();
606
607     const b2DistanceProxy* proxyA = &input→proxyA;
608     const b2DistanceProxy* proxyB = &input→proxyB;
609
610       float32 radiusA = b2Max(proxyA→m_radius, b2_polygonRadius);
611       float32 radiusB = b2Max(proxyB→m_radius, b2_polygonRadius);
612       float32 radius = radiusA + radiusB;
613
614     b2Transform xfA = input→transformA;
615     b2Transform xfB = input→transformB;
616
617     b2Vec2 r = input→translationB;
618     b2Vec2 n(0.0f, 0.0f);
619     float32 lambda = 0.0f;
620
621     // Initial simplex
622     b2Simplex simplex;
623     simplex.m_count = 0;
624
625     // Get simplex vertices as an array.
626     b2SimplexVertex* vertices = &simplex.m_v1;
627
628     // Get support point in −r direction
629     int32 indexA = proxyA→GetSupport(b2MulT(xfA.q, −r));
630     b2Vec2 wA = b2Mul(xfA, proxyA→GetVertex(indexA));
631     int32 indexB = proxyB→GetSupport(b2MulT(xfB.q, r));
632     b2Vec2 wB = b2Mul(xfB, proxyB→GetVertex(indexB));
633       b2Vec2 v = wA − wB;
634
635       // Sigma is the target distance between polygons
636       float32 sigma = b2Max(b2_polygonRadius, radius − b2_polygonRadius);
637     const float32 tolerance = 0.5f * b2_linearSlop;
638
639     // Main iteration loop.
640     const int32 k_maxIters = 20;
641     int32 iter = 0;
642     while (iter < k_maxIters ∧ b2Abs(v.Length() − sigma) > tolerance)
643     {
644       b2Assert(simplex.m_count < 3);
645
646           output→iterations += 1;
647
648       // Support in direction −v (A − B)
649       indexA = proxyA→GetSupport(b2MulT(xfA.q, −v));
650       wA = b2Mul(xfA, proxyA→GetVertex(indexA));
651       indexB = proxyB→GetSupport(b2MulT(xfB.q, v));
652       wB = b2Mul(xfB, proxyB→GetVertex(indexB));
653           b2Vec2 p = wA − wB;
654
655           // −v is a normal at p
656           v.Normalize();
657
658           // Intersect ray with plane
```

```
659       float32 vp = b2Dot(v, p);
660           float32 vr = b2Dot(v, r);
661       if (vp − sigma > lambda * vr)
662       {
663         if (vr ≤ 0.0f)
664         {
665           return false;
666         }
667
668         lambda = (vp − sigma) / vr;
669         if (lambda > 1.0f)
670         {
671           return false;
672         }
673
674               n = −v;
675               simplex.m_count = 0;
676       }
677
678           // Reverse simplex since it works with B − A.
679           // Shift by lambda * r because we want the closest point to the current
    clip point.
680           // Note that the support point p is not shifted because we want the plan
    e equation
681           // to be formed in unshifted space.
682       b2SimplexVertex* vertex = vertices + simplex.m_count;
683       vertex→indexA = indexB;
684       vertex→wA = wB + lambda * r;
685       vertex→indexB = indexA;
686       vertex→wB = wA;
687       vertex→w = vertex→wB − vertex→wA;
688       vertex→a = 1.0f;
689       simplex.m_count += 1;
690
691       switch (simplex.m_count)
692       {
693       case 1:
694         break;
695
696       case 2:
697         simplex.Solve2();
698         break;
699
700       case 3:
701         simplex.Solve3();
702         break;
703
704       default:
705         b2Assert(false);
706       }
707
708       // If we have 3 points, then the origin is in the corresponding triangle.
709       if (simplex.m_count ≡ 3)
710       {
711         // Overlap
712         return false;
713       }
714
715       // Get search direction.
716       v = simplex.GetClosestPoint();
717
718       // Iteration count is equated to the number of support point calls.
719       ++iter;
720     }
721
722     // Prepare output.
```



```
723     b2Vec2 pointA, pointB;
724     simplex.GetWitnessPoints(&pointB, &pointA);
725
726     if (v.LengthSquared() > 0.0f)
727     {
728           n = −v;
729       n.Normalize();
730     }
731
732       output→point = pointA + radiusA * n;
733     output→normal = n;
734     output→lambda = lambda;
735     output→iterations = iter;
736     return true;
737   }
```

```
/*
* Copyright (c) 2007-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Collision/b2Distance.h"

void b2WorldManifold::Initialize(const b2Manifold* manifold,
               const b2Transform& xfA, float32 radiusA,
               const b2Transform& xfB, float32 radiusB)
{
  if (manifold->pointCount == 0)
  {
    return;
  }

  switch (manifold->type)
  {
  case b2Manifold::e_circles:
    {
      normal.Set(1.0f, 0.0f);
      b2Vec2 pointA = b2Mul(xfA, manifold->localPoint);
      b2Vec2 pointB = b2Mul(xfB, manifold->points[0].localPoint);
      if (b2DistanceSquared(pointA, pointB) > b2_epsilon * b2_epsilon)
      {
        normal = pointB - pointA;
        normal.Normalize();
      }

      b2Vec2 cA = pointA + radiusA * normal;
      b2Vec2 cB = pointB - radiusB * normal;
      points[0] = 0.5f * (cA + cB);
      separations[0] = b2Dot(cB - cA, normal);
    }
    break;

  case b2Manifold::e_faceA:
    {
      normal = b2Mul(xfA.q, manifold->localNormal);
      b2Vec2 planePoint = b2Mul(xfA, manifold->localPoint);

      for (int32 i = 0; i < manifold->pointCount; ++i)
      {
        b2Vec2 clipPoint = b2Mul(xfB, manifold->points[i].localPoint);
        b2Vec2 cA = clipPoint + (radiusA - b2Dot(clipPoint - planePoint, normal)
) * normal;
        b2Vec2 cB = clipPoint - radiusB * normal;
        points[i] = 0.5f * (cA + cB);
        separations[i] = b2Dot(cB - cA, normal);
      }
    }
    break;
```



```

  case b2Manifold::e_faceB:
    {
      normal = b2Mul(xfB.q, manifold->localNormal);
      b2Vec2 planePoint = b2Mul(xfB, manifold->localPoint);

      for (int32 i = 0; i < manifold->pointCount; ++i)
      {
        b2Vec2 clipPoint = b2Mul(xfA, manifold->points[i].localPoint);
        b2Vec2 cB = clipPoint + (radiusB - b2Dot(clipPoint - planePoint, normal)
) * normal;
        b2Vec2 cA = clipPoint - radiusA * normal;
        points[i] = 0.5f * (cA + cB);
        separations[i] = b2Dot(cA - cB, normal);
      }

      // Ensure normal points from A to B.
      normal = -normal;
    }
    break;
  }
}

void b2GetPointStates(b2PointState state1[b2_maxManifoldPoints], b2PointState st
ate2[b2_maxManifoldPoints],
            const b2Manifold* manifold1, const b2Manifold* manifold2)
{
  for (int32 i = 0; i < b2_maxManifoldPoints; ++i)
  {
    state1[i] = b2_nullState;
    state2[i] = b2_nullState;
  }

  // Detect persists and removes.
  for (int32 i = 0; i < manifold1->pointCount; ++i)
  {
    b2ContactID id = manifold1->points[i].id;

    state1[i] = b2_removeState;

    for (int32 j = 0; j < manifold2->pointCount; ++j)
    {
      if (manifold2->points[j].id.key == id.key)
      {
        state1[i] = b2_persistState;
        break;
      }
    }
  }

  // Detect persists and adds.
  for (int32 i = 0; i < manifold2->pointCount; ++i)
  {
    b2ContactID id = manifold2->points[i].id;

    state2[i] = b2_addState;

    for (int32 j = 0; j < manifold1->pointCount; ++j)
    {
      if (manifold1->points[j].id.key == id.key)
      {
        state2[i] = b2_persistState;
        break;
      }
    }
  }
```

```
}

// From Real-time Collision Detection, p179.
bool b2AABB::RayCast(b2RayCastOutput* output, const b2RayCastInput& input) const
{
  float32 tmin = -b2_maxFloat;
  float32 tmax = b2_maxFloat;

  b2Vec2 p = input.p1;
  b2Vec2 d = input.p2 - input.p1;
  b2Vec2 absD = b2Abs(d);

  b2Vec2 normal;

  for (int32 i = 0; i < 2; ++i)
  {
    if (absD(i) < b2_epsilon)
    {
      // Parallel.
      if (p(i) < lowerBound(i) ∨ upperBound(i) < p(i))
      {
        return false;
      }
    }
    else
    {
      float32 inv_d = 1.0f / d(i);
      float32 t1 = (lowerBound(i) - p(i)) * inv_d;
      float32 t2 = (upperBound(i) - p(i)) * inv_d;

      // Sign of the normal vector.
      float32 s = -1.0f;

      if (t1 > t2)
      {
        b2Swap(t1, t2);
        s = 1.0f;
      }

      // Push the min up
      if (t1 > tmin)
      {
        normal.SetZero();
        normal(i) = s;
        tmin = t1;
      }

      // Pull the max down
      tmax = b2Min(tmax, t2);

      if (tmin > tmax)
      {
        return false;
      }
    }
  }

  // Does the ray start inside the box?
  // Does the ray intersect beyond the max fraction?
  if (tmin < 0.0f ∨ input.maxFraction < tmin)
  {
    return false;
  }

  // Intersection.
  output→fraction = tmin;
```



```
  output→normal = normal;
  return true;
}

// Sutherland-Hodgman clipping.
int32 b2ClipSegmentToLine(b2ClipVertex vOut[2], const b2ClipVertex vIn[2],
                          const b2Vec2& normal, float32 offset, int32 vertexIndexA)
{
  // Start with no output points
  int32 numOut = 0;

  // Calculate the distance of end points to the line
  float32 distance0 = b2Dot(normal, vIn[0].v) - offset;
  float32 distance1 = b2Dot(normal, vIn[1].v) - offset;

  // If the points are behind the plane
  if (distance0 ≤ 0.0f) vOut[numOut++] = vIn[0];
  if (distance1 ≤ 0.0f) vOut[numOut++] = vIn[1];

  // If the points are on different sides of the plane
  if (distance0 * distance1 < 0.0f)
  {
    // Find intersection point of edge and plane
    float32 interp = distance0 / (distance0 - distance1);
    vOut[numOut].v = vIn[0].v + interp * (vIn[1].v - vIn[0].v);

    // VertexA is hitting edgeB.
    vOut[numOut].id.cf.indexA = static_cast<uint8>(vertexIndexA);
    vOut[numOut].id.cf.indexB = vIn[0].id.cf.indexB;
    vOut[numOut].id.cf.typeA = b2ContactFeature::e_vertex;
    vOut[numOut].id.cf.typeB = b2ContactFeature::e_face;
    ++numOut;
  }

  return numOut;
}

bool b2TestOverlap( const b2Shape* shapeA, int32 indexA,
                    const b2Shape* shapeB, int32 indexB,
                    const b2Transform& xfA, const b2Transform& xfB)
{
  b2DistanceInput input;
  input.proxyA.Set(shapeA, indexA);
  input.proxyB.Set(shapeB, indexB);
  input.transformA = xfA;
  input.transformB = xfB;
  input.useRadii = true;

  b2SimplexCache cache;
  cache.count = 0;

  b2DistanceOutput output;

  b2Distance(&output, &cache, &input);

  return output.distance < 10.0f * b2_epsilon;
}
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Collision/Shapes/b2PolygonShape.h"

// Find the max separation between poly1 and poly2 using edge normals from poly1
.
static float32 b2FindMaxSeparation(int32* edgeIndex,
                 const b2PolygonShape* poly1, const b2Transform& xf1,
                 const b2PolygonShape* poly2, const b2Transform& xf2)
{
  int32 count1 = poly1->m_count;
  int32 count2 = poly2->m_count;
  const b2Vec2* n1s = poly1->m_normals;
  const b2Vec2* v1s = poly1->m_vertices;
  const b2Vec2* v2s = poly2->m_vertices;
  b2Transform xf = b2MulT(xf2, xf1);

  int32 bestIndex = 0;
  float32 maxSeparation = -b2_maxFloat;
  for (int32 i = 0; i < count1; ++i)
  {
    // Get poly1 normal in frame2.
    b2Vec2 n = b2Mul(xf.q, n1s[i]);
    b2Vec2 v1 = b2Mul(xf, v1s[i]);

    // Find deepest point for normal i.
    float32 si = b2_maxFloat;
    for (int32 j = 0; j < count2; ++j)
    {
      float32 sij = b2Dot(n, v2s[j] - v1);
      if (sij < si)
      {
        si = sij;
      }
    }

    if (si > maxSeparation)
    {
      maxSeparation = si;
      bestIndex = i;
    }
  }

  *edgeIndex = bestIndex;
  return maxSeparation;
}

static void b2FindIncidentEdge(b2ClipVertex c[2],
                const b2PolygonShape* poly1, const b2Transform& xf1, int32 edge1,
```



```
                const b2PolygonShape* poly2, const b2Transform& xf2)
{
  const b2Vec2* normals1 = poly1->m_normals;

  int32 count2 = poly2->m_count;
  const b2Vec2* vertices2 = poly2->m_vertices;
  const b2Vec2* normals2 = poly2->m_normals;

  b2Assert(0 ≤ edge1 ∧ edge1 < poly1->m_count);

  // Get the normal of the reference edge in poly2's frame.
  b2Vec2 normal1 = b2MulT(xf2.q, b2Mul(xf1.q, normals1[edge1]));

  // Find the incident edge on poly2.
  int32 index = 0;
  float32 minDot = b2_maxFloat;
  for (int32 i = 0; i < count2; ++i)
  {
    float32 dot = b2Dot(normal1, normals2[i]);
    if (dot < minDot)
    {
      minDot = dot;
      index = i;
    }
  }

  // Build the clip vertices for the incident edge.
  int32 i1 = index;
  int32 i2 = i1 + 1 < count2 ? i1 + 1 : 0;

  c[0].v = b2Mul(xf2, vertices2[i1]);
  c[0].id.cf.indexA = (uint8)edge1;
  c[0].id.cf.indexB = (uint8)i1;
  c[0].id.cf.typeA = b2ContactFeature::e_face;
  c[0].id.cf.typeB = b2ContactFeature::e_vertex;

  c[1].v = b2Mul(xf2, vertices2[i2]);
  c[1].id.cf.indexA = (uint8)edge1;
  c[1].id.cf.indexB = (uint8)i2;
  c[1].id.cf.typeA = b2ContactFeature::e_face;
  c[1].id.cf.typeB = b2ContactFeature::e_vertex;
}

// Find edge normal of max separation on A - return if separating axis is found
// Find edge normal of max separation on B - return if separation axis is found
// Choose reference edge as min(minA, minB)
// Find incident edge
// Clip

// The normal points from 1 to 2
void b2CollidePolygons(b2Manifold* manifold,
            const b2PolygonShape* polyA, const b2Transform& xfA,
            const b2PolygonShape* polyB, const b2Transform& xfB)
{
  manifold->pointCount = 0;
  float32 totalRadius = polyA->m_radius + polyB->m_radius;

  int32 edgeA = 0;
  float32 separationA = b2FindMaxSeparation(&edgeA, polyA, xfA, polyB, xfB);
  if (separationA > totalRadius)
    return;

  int32 edgeB = 0;
  float32 separationB = b2FindMaxSeparation(&edgeB, polyB, xfB, polyA, xfA);
  if (separationB > totalRadius)
    return;
```

```
132
133     const b2PolygonShape* poly1;  // reference polygon
134     const b2PolygonShape* poly2;  // incident polygon
135     b2Transform xf1, xf2;
136     int32 edge1;          // reference edge
137     uint8 flip;
138     const float32 k_tol = 0.1f * b2_linearSlop;
139
140     if (separationB > separationA + k_tol)
141     {
142       poly1 = polyB;
143       poly2 = polyA;
144       xf1 = xfB;
145       xf2 = xfA;
146       edge1 = edgeB;
147       manifold->type = b2Manifold::e_faceB;
148       flip = 1;
149     }
150     else
151     {
152       poly1 = polyA;
153       poly2 = polyB;
154       xf1 = xfA;
155       xf2 = xfB;
156       edge1 = edgeA;
157       manifold->type = b2Manifold::e_faceA;
158       flip = 0;
159     }
160
161     b2ClipVertex incidentEdge[2];
162     b2FindIncidentEdge(incidentEdge, poly1, xf1, edge1, poly2, xf2);
163
164     int32 count1 = poly1->m_count;
165     const b2Vec2* vertices1 = poly1->m_vertices;
166
167     int32 iv1 = edge1;
168     int32 iv2 = edge1 + 1 < count1 ? edge1 + 1 : 0;
169
170     b2Vec2 v11 = vertices1[iv1];
171     b2Vec2 v12 = vertices1[iv2];
172
173     b2Vec2 localTangent = v12 - v11;
174     localTangent.Normalize();
175
176     b2Vec2 localNormal = b2Cross(localTangent, 1.0f);
177     b2Vec2 planePoint = 0.5f * (v11 + v12);
178
179     b2Vec2 tangent = b2Mul(xf1.q, localTangent);
180     b2Vec2 normal = b2Cross(tangent, 1.0f);
181
182     v11 = b2Mul(xf1, v11);
183     v12 = b2Mul(xf1, v12);
184
185     // Face offset.
186     float32 frontOffset = b2Dot(normal, v11);
187
188     // Side offsets, extended by polytope skin thickness.
189     float32 sideOffset1 = -b2Dot(tangent, v11) + totalRadius;
190     float32 sideOffset2 = b2Dot(tangent, v12) + totalRadius;
191
192     // Clip incident edge against extruded edge1 side edges.
193     b2ClipVertex clipPoints1[2];
194     b2ClipVertex clipPoints2[2];
195     int np;
196
197     // Clip to box side 1
```



```
198     np = b2ClipSegmentToLine(clipPoints1, incidentEdge, -tangent, sideOffset1, iv1
    );
199
200     if (np < 2)
201       return;
202
203     // Clip to negative box side 1
204     np = b2ClipSegmentToLine(clipPoints2, clipPoints1,  tangent, sideOffset2, iv2)
    ;
205
206     if (np < 2)
207     {
208       return;
209     }
210
211     // Now clipPoints2 contains the clipped points.
212     manifold->localNormal = localNormal;
213     manifold->localPoint = planePoint;
214
215     int32 pointCount = 0;
216     for (int32 i = 0; i < b2_maxManifoldPoints; ++i)
217     {
218       float32 separation = b2Dot(normal, clipPoints2[i].v) - frontOffset;
219
220       if (separation <= totalRadius)
221       {
222         b2ManifoldPoint* cp = manifold->points + pointCount;
223         cp->localPoint = b2MulT(xf2, clipPoints2[i].v);
224         cp->id = clipPoints2[i].id;
225         if (flip)
226         {
227           // Swap features
228           b2ContactFeature cf = cp->id.cf;
229           cp->id.cf.indexA = cf.indexB;
230           cp->id.cf.indexB = cf.indexA;
231           cp->id.cf.typeA = cf.typeB;
232           cp->id.cf.typeB = cf.typeA;
233         }
234         ++pointCount;
235       }
236     }
237
238     manifold->pointCount = pointCount;
239 }
```

```
/*
 * Copyright (c) 2007-2009 Erin Catto http://www.box2d.org
 *
 * This software is provided 'as-is', without any express or implied
 * warranty.  In no event will the authors be held liable for any damages
 * arising from the use of this software.
 * Permission is granted to anyone to use this software for any purpose,
 * including commercial applications, and to alter it and redistribute it
 * freely, subject to the following restrictions:
 * 1. The origin of this software must not be misrepresented; you must not
 * claim that you wrote the original software. If you use this software
 * in a product, an acknowledgment in the product documentation would be
 * appreciated but is not required.
 * 2. Altered source versions must be plainly marked as such, and must not be
 * misrepresented as being the original software.
 * 3. This notice may not be removed or altered from any source distribution.
 */

#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Collision/Shapes/b2CircleShape.h"
#include "Box2D/Collision/Shapes/b2EdgeShape.h"
#include "Box2D/Collision/Shapes/b2PolygonShape.h"


// Compute contact points for edge versus circle.
// This accounts for edge connectivity.
void b2CollideEdgeAndCircle(b2Manifold* manifold,
              const b2EdgeShape* edgeA, const b2Transform& xfA,
              const b2CircleShape* circleB, const b2Transform& xfB)
{
  manifold->pointCount = 0;

  // Compute circle in frame of edge
  b2Vec2 Q = b2MulT(xfA, b2Mul(xfB, circleB->m_p));

  b2Vec2 A = edgeA->m_vertex1, B = edgeA->m_vertex2;
  b2Vec2 e = B - A;

  // Barycentric coordinates
  float32 u = b2Dot(e, B - Q);
  float32 v = b2Dot(e, Q - A);

  float32 radius = edgeA->m_radius + circleB->m_radius;

  b2ContactFeature cf;
  cf.indexB = 0;
  cf.typeB = b2ContactFeature::e_vertex;

  // Region A
  if (v <= 0.0f)
  {
    b2Vec2 P = A;
    b2Vec2 d = Q - P;
    float32 dd = b2Dot(d, d);
    if (dd > radius * radius)
    {
      return;
    }

    // Is there an edge connected to A?
    if (edgeA->m_hasVertex0)
    {
      b2Vec2 A1 = edgeA->m_vertex0;
      b2Vec2 B1 = A;
      b2Vec2 e1 = B1 - A1;
      float32 u1 = b2Dot(e1, B1 - Q);
```



```

      // Is the circle in Region AB of the previous edge?
      if (u1 > 0.0f)
      {
        return;
      }
    }

    cf.indexA = 0;
    cf.typeA = b2ContactFeature::e_vertex;
    manifold->pointCount = 1;
    manifold->type = b2Manifold::e_circles;
    manifold->localNormal.SetZero();
    manifold->localPoint = P;
    manifold->points[0].id.key = 0;
    manifold->points[0].id.cf = cf;
    manifold->points[0].localPoint = circleB->m_p;
    return;
  }

  // Region B
  if (u <= 0.0f)
  {
    b2Vec2 P = B;
    b2Vec2 d = Q - P;
    float32 dd = b2Dot(d, d);
    if (dd > radius * radius)
    {
      return;
    }

    // Is there an edge connected to B?
    if (edgeA->m_hasVertex3)
    {
      b2Vec2 B2 = edgeA->m_vertex3;
      b2Vec2 A2 = B;
      b2Vec2 e2 = B2 - A2;
      float32 v2 = b2Dot(e2, Q - A2);

      // Is the circle in Region AB of the next edge?
      if (v2 > 0.0f)
      {
        return;
      }
    }

    cf.indexA = 1;
    cf.typeA = b2ContactFeature::e_vertex;
    manifold->pointCount = 1;
    manifold->type = b2Manifold::e_circles;
    manifold->localNormal.SetZero();
    manifold->localPoint = P;
    manifold->points[0].id.key = 0;
    manifold->points[0].id.cf = cf;
    manifold->points[0].localPoint = circleB->m_p;
    return;
  }

  // Region AB
  float32 den = b2Dot(e, e);
  b2Assert(den > 0.0f);
  b2Vec2 P = (1.0f / den) * (u * A + v * B);
  b2Vec2 d = Q - P;
  float32 dd = b2Dot(d, d);
  if (dd > radius * radius)
  {
```

```
133       return;
134     }
135
136     b2Vec2 n(-e.y, e.x);
137     if (b2Dot(n, Q - A) < 0.0f)
138     {
139       n.Set(-n.x, -n.y);
140     }
141     n.Normalize();
142
143     cf.indexA = 0;
144     cf.typeA = b2ContactFeature::e_face;
145     manifold->pointCount = 1;
146     manifold->type = b2Manifold::e_faceA;
147     manifold->localNormal = n;
148     manifold->localPoint = A;
149     manifold->points[0].id.key = 0;
150     manifold->points[0].id.cf = cf;
151     manifold->points[0].localPoint = circleB->m_p;
152   }
153
154   // This structure is used to keep track of the best separating axis.
155   struct b2EPAxis
156   {
157     enum Type
158     {
159       e_unknown,
160       e_edgeA,
161       e_edgeB
162     };
163
164     Type type;
165     int32 index;
166     float32 separation;
167   };
168
169   // This holds polygon B expressed in frame A.
170   struct b2TempPolygon
171   {
172     b2Vec2 vertices[b2_maxPolygonVertices];
173     b2Vec2 normals[b2_maxPolygonVertices];
174     int32 count;
175   };
176
177   // Reference face used for clipping
178   struct b2ReferenceFace
179   {
180     int32 i1, i2;
181
182     b2Vec2 v1, v2;
183
184     b2Vec2 normal;
185
186     b2Vec2 sideNormal1;
187     float32 sideOffset1;
188
189     b2Vec2 sideNormal2;
190     float32 sideOffset2;
191   };
192
193   // This class collides and edge and a polygon, taking into account edge adjacenc
      y.
194   struct b2EPCollider
195   {
196     void Collide(b2Manifold* manifold, const b2EdgeShape* edgeA, const b2Transform
      & xfA,
```



```
197             const b2PolygonShape* polygonB, const b2Transform& xfB);
198     b2EPAxis ComputeEdgeSeparation();
199     b2EPAxis ComputePolygonSeparation();
200
201     enum VertexType
202     {
203       e_isolated,
204       e_concave,
205       e_convex
206     };
207
208     b2TempPolygon m_polygonB;
209
210     b2Transform m_xf;
211     b2Vec2 m_centroidB;
212     b2Vec2 m_v0, m_v1, m_v2, m_v3;
213     b2Vec2 m_normal0, m_normal1, m_normal2;
214     b2Vec2 m_normal;
215     VertexType m_type1, m_type2;
216     b2Vec2 m_lowerLimit, m_upperLimit;
217     float32 m_radius;
218     bool m_front;
219   };
220
221   // Algorithm:
222   // 1. Classify v1 and v2
223   // 2. Classify polygon centroid as front or back
224   // 3. Flip normal if necessary
225   // 4. Initialize normal range to [-pi, pi] about face normal
226   // 5. Adjust normal range according to adjacent edges
227   // 6. Visit each separating axes, only accept axes within the range
228   // 7. Return if _any_ axis indicates separation
229   // 8. Clip
230   void b2EPCollider::Collide(b2Manifold* manifold, const b2EdgeShape* edgeA, const
       b2Transform& xfA,
231                              const b2PolygonShape* polygonB, const b2Transform& xfB)
232   {
233     m_xf = b2MulT(xfA, xfB);
234
235     m_centroidB = b2Mul(m_xf, polygonB->m_centroid);
236
237     m_v0 = edgeA->m_vertex0;
238     m_v1 = edgeA->m_vertex1;
239     m_v2 = edgeA->m_vertex2;
240     m_v3 = edgeA->m_vertex3;
241
242     bool hasVertex0 = edgeA->m_hasVertex0;
243     bool hasVertex3 = edgeA->m_hasVertex3;
244
245     b2Vec2 edge1 = m_v2 - m_v1;
246     edge1.Normalize();
247     m_normal1.Set(edge1.y, -edge1.x);
248     float32 offset1 = b2Dot(m_normal1, m_centroidB - m_v1);
249     float32 offset0 = 0.0f, offset2 = 0.0f;
250     bool convex1 = false, convex2 = false;
251
252     // Is there a preceding edge?
253     if (hasVertex0)
254     {
255       b2Vec2 edge0 = m_v1 - m_v0;
256       edge0.Normalize();
257       m_normal0.Set(edge0.y, -edge0.x);
258       convex1 = b2Cross(edge0, edge1) >= 0.0f;
259       offset0 = b2Dot(m_normal0, m_centroidB - m_v0);
260     }
261
```

```
262   // Is there a following edge?
263   if (hasVertex3)
264   {
265     b2Vec2 edge2 = m_v3 − m_v2;
266     edge2.Normalize();
267     m_normal2.Set(edge2.y, −edge2.x);
268     convex2 = b2Cross(edge1, edge2) > 0.0f;
269     offset2 = b2Dot(m_normal2, m_centroidB − m_v2);
270   }
271
272   // Determine front or back collision. Determine collision normal limits.
273   if (hasVertex0 ∧ hasVertex3)
274   {
275     if (convex1 ∧ convex2)
276     {
277       m_front = offset0 ≥ 0.0f ∨ offset1 ≥ 0.0f ∨ offset2 ≥ 0.0f;
278       if (m_front)
279       {
280         m_normal = m_normal1;
281         m_lowerLimit = m_normal0;
282         m_upperLimit = m_normal2;
283       }
284       else
285       {
286         m_normal = −m_normal1;
287         m_lowerLimit = −m_normal1;
288         m_upperLimit = −m_normal1;
289       }
290     }
291     else if (convex1)
292     {
293       m_front = offset0 ≥ 0.0f ∨ (offset1 ≥ 0.0f ∧ offset2 ≥ 0.0f);
294       if (m_front)
295       {
296         m_normal = m_normal1;
297         m_lowerLimit = m_normal0;
298         m_upperLimit = m_normal1;
299       }
300       else
301       {
302         m_normal = −m_normal1;
303         m_lowerLimit = −m_normal2;
304         m_upperLimit = −m_normal1;
305       }
306     }
307     else if (convex2)
308     {
309       m_front = offset2 ≥ 0.0f ∨ (offset0 ≥ 0.0f ∧ offset1 ≥ 0.0f);
310       if (m_front)
311       {
312         m_normal = m_normal1;
313         m_lowerLimit = m_normal1;
314         m_upperLimit = m_normal2;
315       }
316       else
317       {
318         m_normal = −m_normal1;
319         m_lowerLimit = −m_normal1;
320         m_upperLimit = −m_normal0;
321       }
322     }
323     else
324     {
325       m_front = offset0 ≥ 0.0f ∧ offset1 ≥ 0.0f ∧ offset2 ≥ 0.0f;
326       if (m_front)
327       {
```



```
328         m_normal = m_normal1;
329         m_lowerLimit = m_normal1;
330         m_upperLimit = m_normal1;
331       }
332       else
333       {
334         m_normal = −m_normal1;
335         m_lowerLimit = −m_normal2;
336         m_upperLimit = −m_normal0;
337       }
338     }
339   }
340   else if (hasVertex0)
341   {
342     if (convex1)
343     {
344       m_front = offset0 ≥ 0.0f ∨ offset1 ≥ 0.0f;
345       if (m_front)
346       {
347         m_normal = m_normal1;
348         m_lowerLimit = m_normal0;
349         m_upperLimit = −m_normal1;
350       }
351       else
352       {
353         m_normal = −m_normal1;
354         m_lowerLimit = m_normal1;
355         m_upperLimit = −m_normal1;
356       }
357     }
358     else
359     {
360       m_front = offset0 ≥ 0.0f ∧ offset1 ≥ 0.0f;
361       if (m_front)
362       {
363         m_normal = m_normal1;
364         m_lowerLimit = m_normal1;
365         m_upperLimit = −m_normal1;
366       }
367       else
368       {
369         m_normal = −m_normal1;
370         m_lowerLimit = m_normal1;
371         m_upperLimit = −m_normal0;
372       }
373     }
374   }
375   else if (hasVertex3)
376   {
377     if (convex2)
378     {
379       m_front = offset1 ≥ 0.0f ∨ offset2 ≥ 0.0f;
380       if (m_front)
381       {
382         m_normal = m_normal1;
383         m_lowerLimit = −m_normal1;
384         m_upperLimit = m_normal2;
385       }
386       else
387       {
388         m_normal = −m_normal1;
389         m_lowerLimit = −m_normal1;
390         m_upperLimit = m_normal1;
391       }
392     }
393     else
```

```
394       {
395         m_front = offset1 ≥ 0.0f ∧ offset2 ≥ 0.0f;
396         if (m_front)
397         {
398           m_normal = m_normal1;
399           m_lowerLimit = −m_normal1;
400           m_upperLimit = m_normal1;
401         }
402         else
403         {
404           m_normal = −m_normal1;
405           m_lowerLimit = −m_normal2;
406           m_upperLimit = m_normal1;
407         }
408       }
409     }
410     else
411     {
412       m_front = offset1 ≥ 0.0f;
413       if (m_front)
414       {
415         m_normal = m_normal1;
416         m_lowerLimit = −m_normal1;
417         m_upperLimit = −m_normal1;
418       }
419       else
420       {
421         m_normal = −m_normal1;
422         m_lowerLimit = m_normal1;
423         m_upperLimit = m_normal1;
424       }
425     }
426
427     // Get polygonB in frameA
428     m_polygonB.count = polygonB→m_count;
429     for (int32 i = 0; i < polygonB→m_count; ++i)
430     {
431       m_polygonB.vertices[i] = b2Mul(m_xf, polygonB→m_vertices[i]);
432       m_polygonB.normals[i] = b2Mul(m_xf.q, polygonB→m_normals[i]);
433     }
434
435     m_radius = polygonB→m_radius + edgeA→m_radius;
436
437     manifold→pointCount = 0;
438
439     b2EPAxis edgeAxis = ComputeEdgeSeparation();
440
441     // If no valid normal can be found than this edge should not collide.
442     if (edgeAxis.type ≡ b2EPAxis::e_unknown)
443     {
444       return;
445     }
446
447     if (edgeAxis.separation > m_radius)
448     {
449       return;
450     }
451
452     b2EPAxis polygonAxis = ComputePolygonSeparation();
453     if (polygonAxis.type ≠ b2EPAxis::e_unknown ∧ polygonAxis.separation > m_radius
    )
454     {
455       return;
456     }
457
458     // Use hysteresis for jitter reduction.
```



```
459     const float32 k_relativeTol = 0.98f;
460     const float32 k_absoluteTol = 0.001f;
461
462     b2EPAxis primaryAxis;
463     if (polygonAxis.type ≡ b2EPAxis::e_unknown)
464     {
465       primaryAxis = edgeAxis;
466     }
467     else if (polygonAxis.separation > k_relativeTol * edgeAxis.separation + k_abso
    luteTol)
468     {
469       primaryAxis = polygonAxis;
470     }
471     else
472     {
473       primaryAxis = edgeAxis;
474     }
475
476     b2ClipVertex ie[2];
477     b2ReferenceFace rf;
478     if (primaryAxis.type ≡ b2EPAxis::e_edgeA)
479     {
480       manifold→type = b2Manifold::e_faceA;
481
482       // Search for the polygon normal that is most anti−parallel to the edge norm
    al.
483       int32 bestIndex = 0;
484       float32 bestValue = b2Dot(m_normal, m_polygonB.normals[0]);
485       for (int32 i = 1; i < m_polygonB.count; ++i)
486       {
487         float32 value = b2Dot(m_normal, m_polygonB.normals[i]);
488         if (value < bestValue)
489         {
490           bestValue = value;
491           bestIndex = i;
492         }
493       }
494
495       int32 i1 = bestIndex;
496       int32 i2 = i1 + 1 < m_polygonB.count ? i1 + 1 : 0;
497
498       ie[0].v = m_polygonB.vertices[i1];
499       ie[0].id.cf.indexA = 0;
500       ie[0].id.cf.indexB = static_cast<uint8>(i1);
501       ie[0].id.cf.typeA = b2ContactFeature::e_face;
502       ie[0].id.cf.typeB = b2ContactFeature::e_vertex;
503
504       ie[1].v = m_polygonB.vertices[i2];
505       ie[1].id.cf.indexA = 0;
506       ie[1].id.cf.indexB = static_cast<uint8>(i2);
507       ie[1].id.cf.typeA = b2ContactFeature::e_face;
508       ie[1].id.cf.typeB = b2ContactFeature::e_vertex;
509
510       if (m_front)
511       {
512         rf.i1 = 0;
513         rf.i2 = 1;
514         rf.v1 = m_v1;
515         rf.v2 = m_v2;
516         rf.normal = m_normal1;
517       }
518       else
519       {
520         rf.i1 = 1;
521         rf.i2 = 0;
522         rf.v1 = m_v2;
```

```
523         rf.v2 = m_v1;
524         rf.normal = -m_normal1;
525       }
526     }
527     else
528     {
529       manifold->type = b2Manifold::e_faceB;
530
531       ie[0].v = m_v1;
532       ie[0].id.cf.indexA = 0;
533       ie[0].id.cf.indexB = static_cast<uint8>(primaryAxis.index);
534       ie[0].id.cf.typeA = b2ContactFeature::e_vertex;
535       ie[0].id.cf.typeB = b2ContactFeature::e_face;
536
537       ie[1].v = m_v2;
538       ie[1].id.cf.indexA = 0;
539       ie[1].id.cf.indexB = static_cast<uint8>(primaryAxis.index);
540       ie[1].id.cf.typeA = b2ContactFeature::e_vertex;
541       ie[1].id.cf.typeB = b2ContactFeature::e_face;
542
543       rf.i1 = primaryAxis.index;
544       rf.i2 = rf.i1 + 1 < m_polygonB.count ? rf.i1 + 1 : 0;
545       rf.v1 = m_polygonB.vertices[rf.i1];
546       rf.v2 = m_polygonB.vertices[rf.i2];
547       rf.normal = m_polygonB.normals[rf.i1];
548     }
549
550     rf.sideNormal1.Set(rf.normal.y, -rf.normal.x);
551     rf.sideNormal2 = -rf.sideNormal1;
552     rf.sideOffset1 = b2Dot(rf.sideNormal1, rf.v1);
553     rf.sideOffset2 = b2Dot(rf.sideNormal2, rf.v2);
554
555     // Clip incident edge against extruded edge1 side edges.
556     b2ClipVertex clipPoints1[2];
557     b2ClipVertex clipPoints2[2];
558     int32 np;
559
560     // Clip to box side 1
561     np = b2ClipSegmentToLine(clipPoints1, ie, rf.sideNormal1, rf.sideOffset1, rf.i
    1);
562
563     if (np < b2_maxManifoldPoints)
564     {
565       return;
566     }
567
568     // Clip to negative box side 1
569     np = b2ClipSegmentToLine(clipPoints2, clipPoints1, rf.sideNormal2, rf.sideOffs
    et2, rf.i2);
570
571     if (np < b2_maxManifoldPoints)
572     {
573       return;
574     }
575
576     // Now clipPoints2 contains the clipped points.
577     if (primaryAxis.type == b2EPAxis::e_edgeA)
578     {
579       manifold->localNormal = rf.normal;
580       manifold->localPoint = rf.v1;
581     }
582     else
583     {
584       manifold->localNormal = polygonB->m_normals[rf.i1];
585       manifold->localPoint = polygonB->m_vertices[rf.i1];
586     }
```



```
587
588     int32 pointCount = 0;
589     for (int32 i = 0; i < b2_maxManifoldPoints; ++i)
590     {
591       float32 separation;
592
593       separation = b2Dot(rf.normal, clipPoints2[i].v - rf.v1);
594
595       if (separation <= m_radius)
596       {
597         b2ManifoldPoint* cp = manifold->points + pointCount;
598
599         if (primaryAxis.type == b2EPAxis::e_edgeA)
600         {
601           cp->localPoint = b2MulT(m_xf, clipPoints2[i].v);
602           cp->id = clipPoints2[i].id;
603         }
604         else
605         {
606           cp->localPoint = clipPoints2[i].v;
607           cp->id.cf.typeA = clipPoints2[i].id.cf.typeB;
608           cp->id.cf.typeB = clipPoints2[i].id.cf.typeA;
609           cp->id.cf.indexA = clipPoints2[i].id.cf.indexB;
610           cp->id.cf.indexB = clipPoints2[i].id.cf.indexA;
611         }
612
613         ++pointCount;
614       }
615     }
616
617     manifold->pointCount = pointCount;
618   }
619
620   b2EPAxis b2EPCollider::ComputeEdgeSeparation()
621   {
622     b2EPAxis axis;
623     axis.type = b2EPAxis::e_edgeA;
624     axis.index = m_front ? 0 : 1;
625     axis.separation = FLT_MAX;
626
627     for (int32 i = 0; i < m_polygonB.count; ++i)
628     {
629       float32 s = b2Dot(m_normal, m_polygonB.vertices[i] - m_v1);
630       if (s < axis.separation)
631       {
632         axis.separation = s;
633       }
634     }
635
636     return axis;
637   }
638
639   b2EPAxis b2EPCollider::ComputePolygonSeparation()
640   {
641     b2EPAxis axis;
642     axis.type = b2EPAxis::e_unknown;
643     axis.index = -1;
644     axis.separation = -FLT_MAX;
645
646     b2Vec2 perp(-m_normal.y, m_normal.x);
647
648     for (int32 i = 0; i < m_polygonB.count; ++i)
649     {
650       b2Vec2 n = -m_polygonB.normals[i];
651
652       float32 s1 = b2Dot(n, m_polygonB.vertices[i] - m_v1);
```

```
653       float32 s2 = b2Dot(n, m_polygonB.vertices[i] − m_v2);
654       float32 s = b2Min(s1, s2);
655
656       if (s > m_radius)
657       {
658         // No collision
659         axis.type = b2EPAxis::e_edgeB;
660         axis.index = i;
661         axis.separation = s;
662         return axis;
663       }
664
665       // Adjacency
666       if (b2Dot(n, perp) ≥ 0.0f)
667       {
668         if (b2Dot(n − m_upperLimit, m_normal) < −b2_angularSlop)
669         {
670           continue;
671         }
672       }
673       else
674       {
675         if (b2Dot(n − m_lowerLimit, m_normal) < −b2_angularSlop)
676         {
677           continue;
678         }
679       }
680
681       if (s > axis.separation)
682       {
683         axis.type = b2EPAxis::e_edgeB;
684         axis.index = i;
685         axis.separation = s;
686       }
687     }
688
689     return axis;
690   }
691
692   void b2CollideEdgeAndPolygon( b2Manifold* manifold,
693                 const b2EdgeShape* edgeA, const b2Transform& xfA,
694                 const b2PolygonShape* polygonB, const b2Transform& xfB)
695   {
696     b2EPCollider collider;
697     collider.Collide(manifold, edgeA, xfA, polygonB, xfB);
698   }
```



```
1   /*
2   * Copyright (c) 2007−2009 Erin Catto http://www.box2d.org
3   *
4   * This software is provided 'as−is', without any express or implied
5   * warranty.  In no event will the authors be held liable for any damages
6   * arising from the use of this software.
7   * Permission is granted to anyone to use this software for any purpose,
8   * including commercial applications, and to alter it and redistribute it
9   * freely, subject to the following restrictions:
10  * 1. The origin of this software must not be misrepresented; you must not
11  * claim that you wrote the original software. If you use this software
12  * in a product, an acknowledgment in the product documentation would be
13  * appreciated but is not required.
14  * 2. Altered source versions must be plainly marked as such, and must not be
15  * misrepresented as being the original software.
16  * 3. This notice may not be removed or altered from any source distribution.
17  */
18
19  #include "Box2D/Collision/b2Collision.h"
20  #include "Box2D/Collision/Shapes/b2CircleShape.h"
21  #include "Box2D/Collision/Shapes/b2PolygonShape.h"
22
23  void b2CollideCircles(
24    b2Manifold* manifold,
25    const b2CircleShape* circleA, const b2Transform& xfA,
26    const b2CircleShape* circleB, const b2Transform& xfB)
27  {
28    manifold→pointCount = 0;
29
30    b2Vec2 pA = b2Mul(xfA, circleA→m_p);
31    b2Vec2 pB = b2Mul(xfB, circleB→m_p);
32
33    b2Vec2 d = pB − pA;
34    float32 distSqr = b2Dot(d, d);
35    float32 rA = circleA→m_radius, rB = circleB→m_radius;
36    float32 radius = rA + rB;
37    if (distSqr > radius * radius)
38    {
39      return;
40    }
41
42    manifold→type = b2Manifold::e_circles;
43    manifold→localPoint = circleA→m_p;
44    manifold→localNormal.SetZero();
45    manifold→pointCount = 1;
46
47    manifold→points[0].localPoint = circleB→m_p;
48    manifold→points[0].id.key = 0;
49  }
50
51  void b2CollidePolygonAndCircle(
52    b2Manifold* manifold,
53    const b2PolygonShape* polygonA, const b2Transform& xfA,
54    const b2CircleShape* circleB, const b2Transform& xfB)
55  {
56    manifold→pointCount = 0;
57
58    // Compute circle position in the frame of the polygon.
59    b2Vec2 c = b2Mul(xfB, circleB→m_p);
60    b2Vec2 cLocal = b2MulT(xfA, c);
61
62    // Find the min separating edge.
63    int32 normalIndex = 0;
64    float32 separation = −b2_maxFloat;
65    float32 radius = polygonA→m_radius + circleB→m_radius;
66    int32 vertexCount = polygonA→m_count;
```

```
67    const b2Vec2* vertices = polygonA→m_vertices;
68    const b2Vec2* normals = polygonA→m_normals;
69
70    for (int32 i = 0; i < vertexCount; ++i)
71    {
72      float32 s = b2Dot(normals[i], cLocal − vertices[i]);
73
74      if (s > radius)
75      {
76        // Early out.
77        return;
78      }
79
80      if (s > separation)
81      {
82        separation = s;
83        normalIndex = i;
84      }
85    }
86
87    // Vertices that subtend the incident face.
88    int32 vertIndex1 = normalIndex;
89    int32 vertIndex2 = vertIndex1 + 1 < vertexCount ? vertIndex1 + 1 : 0;
90    b2Vec2 v1 = vertices[vertIndex1];
91    b2Vec2 v2 = vertices[vertIndex2];
92
93    // If the center is inside the polygon ...
94    if (separation < b2_epsilon)
95    {
96      manifold→pointCount = 1;
97      manifold→type = b2Manifold::e_faceA;
98      manifold→localNormal = normals[normalIndex];
99      manifold→localPoint = 0.5f * (v1 + v2);
100     manifold→points[0].localPoint = circleB→m_p;
101     manifold→points[0].id.key = 0;
102     return;
103   }
104
105   // Compute barycentric coordinates
106   float32 u1 = b2Dot(cLocal − v1, v2 − v1);
107   float32 u2 = b2Dot(cLocal − v2, v1 − v2);
108   if (u1 ≤ 0.0f)
109   {
110     if (b2DistanceSquared(cLocal, v1) > radius * radius)
111     {
112       return;
113     }
114
115     manifold→pointCount = 1;
116     manifold→type = b2Manifold::e_faceA;
117     manifold→localNormal = cLocal − v1;
118     manifold→localNormal.Normalize();
119     manifold→localPoint = v1;
120     manifold→points[0].localPoint = circleB→m_p;
121     manifold→points[0].id.key = 0;
122   }
123   else if (u2 ≤ 0.0f)
124   {
125     if (b2DistanceSquared(cLocal, v2) > radius * radius)
126     {
127       return;
128     }
129
130     manifold→pointCount = 1;
131     manifold→type = b2Manifold::e_faceA;
132     manifold→localNormal = cLocal − v2;
```



```
133     manifold→localNormal.Normalize();
134     manifold→localPoint = v2;
135     manifold→points[0].localPoint = circleB→m_p;
136     manifold→points[0].id.key = 0;
137   }
138   else
139   {
140     b2Vec2 faceCenter = 0.5f * (v1 + v2);
141     float32 s = b2Dot(cLocal − faceCenter, normals[vertIndex1]);
142     if (s > radius)
143     {
144       return;
145     }
146
147     manifold→pointCount = 1;
148     manifold→type = b2Manifold::e_faceA;
149     manifold→localNormal = normals[vertIndex1];
150     manifold→localPoint = faceCenter;
151     manifold→points[0].localPoint = circleB→m_p;
152     manifold→points[0].id.key = 0;
153   }
154 }
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#include "Box2D/Collision/b2BroadPhase.h"

b2BroadPhase::b2BroadPhase()
{
  m_proxyCount = 0;

  m_pairCapacity = 16;
  m_pairCount = 0;
  m_pairBuffer = (b2Pair*)b2Alloc(m_pairCapacity * sizeof(b2Pair));

  m_moveCapacity = 16;
  m_moveCount = 0;
  m_moveBuffer = (int32*)b2Alloc(m_moveCapacity * sizeof(int32));
}

b2BroadPhase::~b2BroadPhase()
{
  b2Free(m_moveBuffer);
  b2Free(m_pairBuffer);
}

int32 b2BroadPhase::CreateProxy(const b2AABB& aabb, void* userData)
{
  int32 proxyId = m_tree.CreateProxy(aabb, userData);
  ++m_proxyCount;
  BufferMove(proxyId);
  return proxyId;
}

void b2BroadPhase::DestroyProxy(int32 proxyId)
{
  UnBufferMove(proxyId);
  --m_proxyCount;
  m_tree.DestroyProxy(proxyId);
}

void b2BroadPhase::MoveProxy(int32 proxyId, const b2AABB& aabb, const b2Vec2& displacement)
{
  bool buffer = m_tree.MoveProxy(proxyId, aabb, displacement);
  if (buffer)
  {
    BufferMove(proxyId);
  }
}

void b2BroadPhase::TouchProxy(int32 proxyId)
{
```



```
  BufferMove(proxyId);
}

void b2BroadPhase::BufferMove(int32 proxyId)
{
  if (m_moveCount == m_moveCapacity)
  {
    int32* oldBuffer = m_moveBuffer;
    m_moveCapacity *= 2;
    m_moveBuffer = (int32*)b2Alloc(m_moveCapacity * sizeof(int32));
    memcpy(m_moveBuffer, oldBuffer, m_moveCount * sizeof(int32));
    b2Free(oldBuffer);
  }

  m_moveBuffer[m_moveCount] = proxyId;
  ++m_moveCount;
}

void b2BroadPhase::UnBufferMove(int32 proxyId)
{
  for (int32 i = 0; i < m_moveCount; ++i)
  {
    if (m_moveBuffer[i] == proxyId)
    {
      m_moveBuffer[i] = e_nullProxy;
    }
  }
}

// This is called from b2DynamicTree::Query when we are gathering pairs.
bool b2BroadPhase::QueryCallback(int32 proxyId)
{
  // A proxy cannot form a pair with itself.
  if (proxyId == m_queryProxyId)
  {
    return true;
  }

  // Grow the pair buffer as needed.
  if (m_pairCount == m_pairCapacity)
  {
    b2Pair* oldBuffer = m_pairBuffer;
    m_pairCapacity *= 2;
    m_pairBuffer = (b2Pair*)b2Alloc(m_pairCapacity * sizeof(b2Pair));
    memcpy(m_pairBuffer, oldBuffer, m_pairCount * sizeof(b2Pair));
    b2Free(oldBuffer);
  }

  m_pairBuffer[m_pairCount].proxyIdA = b2Min(proxyId, m_queryProxyId);
  m_pairBuffer[m_pairCount].proxyIdB = b2Max(proxyId, m_queryProxyId);
  ++m_pairCount;

  return true;
}
```

nov 26, 19 17:34 **ConfigServidor.h** Page 1/1

```
#ifndef _CONFIG_SERVIDOR_H_
#define _CONFIG_SERVIDOR_H_

#define RUTA_CONFIG_SERVIDOR "config/server_settings.json"

#define CONFIG_SERVIDOR ConfigServidor::instancia()

#include <string>
#include <vector>

#include "includes/3rd-party/jsoncpp/json.hpp"

class ConfigServidor {
private:
    ConfigServidor(const std::string& rutaArchivo);
    Json json_;
    std::vector<int> tilesTierra_;
    std::vector<int> tilesPista_;

public:
    static ConfigServidor& instancia();
    std::string puertoServidor();
    std::string hostServidor();
    unsigned int maxClientesEnEspera();
    unsigned int snapshotsEnviadosPorSegundo();

    float anchoTile();

    std::string rutaPistas();
    std::vector<int>& tilesTierra();
    std::vector<int>& tilesPista();
    int tileArena();
    int tileBarro();
    int tileAceite();
    int tileVacio();

    uint32_t simulacionesPorSegundo();
    uint32_t iteracionesPosicion();
    uint32_t iteracionesVelocidad();

    unsigned int velocidadMaxVehiculoAdelante();
    unsigned int velocidadMaxVehiculoAtras();
    unsigned int aceleracionVehiculo();
    unsigned int maniobrabilidadVehiculo();
    unsigned int agarreVehiculo();
    unsigned int saludVehiculo();
    float anchoVehiculo();
    float largoVehiculo();

    float ladoSuperficie();

    int cantidadMaximaModificadores();
    int factorAparicionModificador();

    uint8_t disminucionVidaChoqueConVehiculo();
};

#endif
```

nov 26, 19 17:34 **Servidor.h** Page 1/1

```
#ifndef _SERVIDOR_H_
#define _SERVIDOR_H_

#include "includes/servidor/HiloAceptador.h"
#include "includes/servidor/SalaDeEspera.h"
#include "includes/servidor/DistribuidorEventos.h"
#include "includes/servidor/CoordinadorPartidas.h"

#include <string>

#define CARACTER_SALIR 'q'

class Servidor {
private:
    ColaBloqueante<std::shared_ptr<Evento>> eventosRecibidos_;
    SalaDeEspera salaDeEspera_;
    HiloAceptador hiloAceptador_;
    DistribuidorEventos distribuidorEventos_;
    CoordinadorPartidas coordinadorPartidas_;

public:
    Servidor(const std::string& unHost, const std::string& puerto);
    void correr();
    void cerrar();
};

#endif
```

nov 26, 19 17:34 **SalaDeEspera.h** Page 1/1

```
#ifndef _SALA_DE_ESPERA_H_
#define _SALA_DE_ESPERA_H_

#include <map>
#include <memory>
#include <mutex>

#include "includes/common/Handler.h"
#include "includes/common/red/SocketTCP.h"
#include "includes/servidor/Jugador.h"

class SalaDeEspera : public Handler {
private:
    uint32_t contadorJugadores_;
    ColaBloqueante<std::shared_ptr<Evento>>& destinoEventos_;
    //FIXME: Proteger esto
    std::map<uint32_t, std::shared_ptr<Jugador>> jugadores_;
    std::mutex mtx_;

public:
    SalaDeEspera(ColaBloqueante<std::shared_ptr<Evento>>& destinoEventos);
    ~SalaDeEspera();
    void agregarJugador(SocketTCP^ socket);
    void agregarJugador(std::shared_ptr<Jugador> unJugador);
    std::shared_ptr<Jugador> quitarJugador(uint32_t uuidJugador);
    std::shared_ptr<Jugador> getJugador(uint32_t uuidJugador);

    void ocurrio(std::shared_ptr<Evento> evento);

    virtual void manejar(Evento& e) override;
    virtual void manejar(EventoDesconexion& e) override;
};

#endif
```

nov 26, 19 17:34 **SocketTCPServidor.h** Page 1/1

```
#ifndef _SOCKET_TCP_SERVIDOR_H_
#define _SOCKET_TCP_SERVIDOR_H_

#include "includes/common/red/SocketTCP.h"

#define ERROR_SET_SOCK_OPT "Error al llamar a setsockopt antes de enlazar."
#define ERROR_BIND "Error al intentar enlazar el socket."
#define ERROR_LISTEN "Error al llamar a listen()."
#define ERROR_ACEPTAR "Error al llamar a accept()."

class SocketTCPServidor : public SocketTCP {
public:
    SocketTCPServidor(const std::string& unHost, const std::string& unPuerto);

    void enlazar();

    void escuchar(unsigned int maxEnEspera);

    SocketTCP aceptar();
};

#endif
```

nov 26, 19 17:34 **Partida.h** Page 1/1

```
#ifndef _PARTIDA_H_
#define _PARTIDA_H_

#include "includes/common/Hilo.h"
#include "includes/common/Handler.h"
#include "includes/common/ColaProtegida.h"
#include "includes/servidor/Jugador.h"
#include "includes/servidor/modelo/Mundo.h"

#include <map>
#include <memory>

//Forward declaration
class SalaDeEspera;

class Partida : public Hilo, public Handler {
//TODO: Devolver al jugador a la sala de espera cuando finaliza la partida
private:
    std::map<uint32_t, std::shared_ptr<Jugador>> jugadores_;
    std::map<uint32_t, bool> uuidJugadorAEstaListo_;
    ColaProtegida<std::shared_ptr<Evento>> eventosEntrantes_;
    Mundo mundo_;
    SalaDeEspera& salaDeEspera_;
    bool fueIniciada_;

    Partida(const Partida& otra) = delete;
    Partida& operator=(const Partida& otra) = delete;
    Partida(Partida∧ otra) = delete;
    Partida& operator=(Partida∧ otra) = delete;

    void step(uint32_t iteracion);
    void asignarVehiculos();

public:
    Partida(uint16_t uuidPista, SalaDeEspera& salaDeEspera);
    ~Partida();

    void agregarJugador(std::shared_ptr<Jugador> jugador);
    std::map<uint32_t, std::shared_ptr<Jugador>>& jugadores();

    bool todosListos();
    bool estaListo(uint32_t uuidJugador);
    void marcarListo(uint32_t uuidJugador);

    virtual void correr() override;
    virtual void detener() override;

    virtual void manejar(Evento& e) override;
    virtual void manejar(EventoFinCarrera& e) override;

    void ocurrio(std::shared_ptr<Evento> unEvento);

    bool aceptaJugadores();
};

#endif
```

nov 26, 19 17:34 **SuperficieTierra.h** Page 1/1

```
#ifndef _TIERRA_H_
#define _TIERRA_H_

#include "includes/servidor/modelo/superficies/Superficie.h"

class SuperficieTierra : public Superficie {

public:
    virtual int getTipo() override;
};

#endif
```

nov 26, 19 17:34 **SuperficiePista.h** Page 1/1

```
#ifndef _SUPERFICIE_PISTA_H_
#define _SUPERFICIE_PISTA_H_

#include "includes/servidor/modelo/superficies/Superficie.h"

class SuperficiePista : public Superficie {

public:
    virtual int getTipo() override;
};

#endif
```

nov 26, 19 17:34 **Superficie.h** Page 1/1

```
#ifndef _SUPERFICIE_H_
#define _SUPERFICIE_H_

#include "includes/servidor/modelo/Colisionable.h"

class Superficie : public Colisionable{

};

#endif
```

nov 26, 19 17:34 **SuperficieFactory.h** Page 1/1

```
#ifndef _SUPERFICIE_FACTORY_H_
#define _SUPERFICIE_FACTORY_H_

#include <memory>

#include "includes/servidor/modelo/superficies/Superficie.h"
#include "includes/servidor/modelo/superficies/SuperficieTierra.h"
#include "includes/servidor/modelo/superficies/SuperficiePista.h"
#include "includes/servidor/modelo/superficies/SuperficieArena.h"

#define MENSAJE_ERROR_SUPERFICIE_DESCONOCIDA "Error al instanciar la superficie, se utilizÃ³ un UUID desconocido."

class SuperficieFactory {
private:
public:
    static std::shared_ptr<Superficie> instanciar(int uuid);
};

#endif
```

nov 26, 19 17:34 **SuperficieArena.h** Page 1/1

```
#ifndef _SUPERFICIE_ARENA_H_
#define _SUPERFICIE_ARENA_H_

#include "includes/servidor/modelo/superficies/Superficie.h"

class SuperficieArena : public Superficie {

public:
    virtual int getTipo() override;
};

#endif
```

nov 26, 19 17:34 **Mundo.h** Page 1/1

```
#ifndef _MUNDO_H_
#define _MUNDO_H_

#include <memory>
#include <queue>

#include "includes/common/Handler.h"
#include "includes/common/ColaProtegida.h"
#include "includes/servidor/modelo/fisicas/Fisicas.h"
#include "includes/servidor/modelo/fisicas/ContactListener.h"
#include "includes/servidor/modelo/entidades/Vehiculo.h"
#include "includes/servidor/modelo/entidades/Modificador.h"
#include "includes/common/Tile.h"
#include "includes/servidor/modelo/superficies/Superficie.h"
#include "includes/servidor/modelo/entidades/carrera/Carrera.h"
#include "includes/servidor/Jugador.h"

class Mundo : public Handler {
private:
    std::map<Tile, std::shared_ptr<Superficie>> tileASuelo_;
    std::vector<Tile> tilesConPista_;
    std::queue<Posicion> posicionesIniciales_;
    std::map<uint32_t, Vehiculo> jugadoresAVehiculos_;
    ColaProtegida<std::shared_ptr<Evento>> eventosOcurridos_;
    Fisicas fisicas_;
    std::queue<uint8_t> uuidsObjetos_;
    std::map<uint32_t, uint8_t> jugadoresAIDVehiculo_;
    unsigned int snapshotsEnviadosPorSegundo_;
    ContactListener contactListener_;
    Carrera carrera_;
    std::map<uint8_t, std::shared_ptr<Modificador>> modificadores_;

    std::map<uint8_t, datosVehiculo_> serializarEstado();

public:
    Mundo(uint16_t uuidPista);
    ~Mundo();

    void step(uint32_t numeroIteracion);
    Cola<std::shared_ptr<Evento>>& eventosOcurridos();
    uint8_t agregarVehiculo(std::shared_ptr<Jugador> unJugador);
    std::map<uint8_t, datosVehiculo_> getEstadoInicial();
    void agregarModificadores(uint32_t numeroIteracion);

    void recuperarUuid(uint8_t uuid);

    virtual void manejar(Evento& e) override;
    virtual void manejar(EventoAcelerar& e) override;
    virtual void manejar(EventoDesacelerar& e) override;
    virtual void manejar(EventoFrenar& e) override;
    virtual void manejar(EventoDejarDeFrenar& e) override;
    virtual void manejar(EventoDoblarIzquierda& e) override;
    virtual void manejar(EventoDejarDeDoblarIzquierda& e) override;
    virtual void manejar(EventoDoblarDerecha& e) override;
    virtual void manejar(EventoDejarDeDoblarDerecha& e) override;
};

#endif
```

nov 26, 19 17:34 **Posicion.h** Page 1/1

```
#ifndef _POSICION_H_
#define _POSICION_H_

#include <cstdint>

class Posicion {
public:
    float x_;
    float y_;
    uint16_t anguloDeg_;

    Posicion(float x, float y, uint16_t anguloDeg);
};

#endif
```

nov 26, 19 17:34 **Identificable.h** Page 1/1

```
#ifndef _IDENTIFICABLE_H_
#define _IDENTIFICABLE_H_

#include <cstdint>

class Identificable {
private:
    uint8_t UUID_;
public:
    Identificable(uint8_t uuid);
    virtual ~Identificable();
    uint8_t uuid();
};

#endif
```

nov 26, 19 17:34 **Transformacion.h** Page 1/1

```
#ifndef _TRANSFORMACION_H_
#define _TRANSFORMACION_H_

//Forward declaration
class Fisicas;

class Transformacion {
protected:
    Fisicas& fisicas_;

public:
    Transformacion(Fisicas& unasFisicas);
    virtual ~Transformacion();
    virtual void aplicar() = 0;
};

#endif
```

nov 26, 19 17:34 **Reubicar.h** Page 1/1

```
1  #ifndef _REUBICAR_H_
2  #define _REUBICAR_H_
3
4  #include "includes/servidor/modelo/fisicas/transformaciones/Transformacion.h"
5
6  #include "includes/servidor/modelo/Colisionable.h"
7
8  //Forward declarations
9  class b2Body;
10 class Posicion;
11
12 class Reubicar : public Transformacion {
13 private:
14     b2Body* cuerpo_;
15     Posicion& posicion_;
16 public:
17     Reubicar(Fisicas& fiscas, b2Body* cuerpo, Posicion& posicion);
18     virtual void aplicar() override;
19 };
20
21 #endif
```

nov 26, 19 17:34 **Quitar.h** Page 1/1

```
1  #ifndef _QUITAR_H_
2  #define _QUITAR_H_
3
4  #include "includes/servidor/modelo/fisicas/transformaciones/Transformacion.h"
5
6  #include <cstdint>
7
8  //Forward declarations
9  class b2Body;
10 class Fisicas;
11
12 class Quitar : public Transformacion {
13 private:
14     b2Body* cuerpo_;
15     uint8_t uuidCuerpo_;
16 public:
17     Quitar(Fisicas& fiscas, b2Body* cuerpo, uint8_t uuidCuerpo);
18     virtual void aplicar() override;
19 };
20
21 #endif
```

```
#ifndef _FISICAS_H_
#define _FISICAS_H_

#include <map>
#include <memory>

#include "includes/3rd-party/Box2D/Box2D.h"
#include "includes/common/Tile.h"
#include "includes/common/Cola.h"
#include "includes/common/eventos/Evento.h"
#include "includes/servidor/modelo/superficies/Superficie.h"
#include "includes/servidor/modelo/entidades/Vehiculo.h"
#include "includes/servidor/modelo/entidades/carrera/Checkpoint.h"
#include "includes/servidor/modelo/entidades/Modificador.h"
#include "includes/servidor/modelo/movimiento/Posicion.h"
#include "includes/servidor/modelo/fisicas/B2DVehiculo.h"
#include "includes/servidor/modelo/fisicas/ContactListener.h"
#include "includes/servidor/modelo/fisicas/transformaciones/Transformacion.h"

#ifndef DEGTORAD
#define DEGTORAD 0.0174532925199432957f
#define RADTODEG 57.295779513082320876f
#endif

//FD
class Mundo;

class Fisicas {
private:
    b2Vec2 gravedad_;
    std::shared_ptr<b2World> mundoBox2D_;
    std::map<uint8_t, b2Body*> colisionables_;
    std::map<uint8_t, std::shared_ptr<B2DVehiculo>> vehiculos_;
    double frecuencia_;
    uint32_t iteracion_;
    Cola<std::shared_ptr<Evento>>& eventosOcurridos_;
    std::queue<std::shared_ptr<Transformacion>> transformaciones_;
    Mundo& mundo_;

public:
    Fisicas(Cola<std::shared_ptr<Evento>>& eventosOcurridos, ContactListener& co
ntactListener, Mundo& mundo);
    ~Fisicas();
    void generarSuelo(std::map<Tile, std::shared_ptr<Superficie>>& tileASuelo);
    //void generarSuperficies(std::map<Tile, std::shared_ptr<Superficie>>& tileA
Superficie);
    void generarCheckpoints(std::map<int, Checkpoint>& checkpoints);
    void step(uint32_t numeroIteracion);

    void agregarVehiculo(Vehiculo& vehiculo, Posicion& posicion);
    void acelerar(uint8_t uuidVehiculo);
    void desacelerar(uint8_t uuidVehiculo);
    void frenar(uint8_t uuidVehiculo);
    void dejarDeFrenar(uint8_t uuidVehiculo);
    void doblarIzquierda(uint8_t uuidVehiculo);
    void dejarDeDoblarIzquierda(uint8_t uuidVehiculo);
    void doblarDerecha(uint8_t uuidVehiculo);
    void dejarDeDoblarDerecha(uint8_t uuidVehiculo);

    void agregarModificador(std::shared_ptr<Modificador> modificador, uint8_t ti
po, Posicion& posicion);

    void ocurrio(std::shared_ptr<Evento> evento);

    Posicion getPosicionDe(uint8_t idCuerpo);
    void nuevoUuidDisponible(uint8_t uuid);
```



```

    void reubicar(Vehiculo& vehiculo, Posicion& Posicion);
    void quitar(CajaVida& cajaVida);
    void quitar(Barro& barro);
    void quitar(Aceite& aceite);
    void quitar(Boost& boost);
    void quitar(Piedra& piedra);

};

#endif
```

nov 26, 19 17:34 **ContactListener.h** Page 1/1

```
#ifndef _CONTACT_LISTENER_H_
#define _CONTACT_LISTENER_H_

#include "includes/3rd-party/Box2D/Box2D.h"

//Forward declations
class Fisicas;
class SuperficieArena;
class Vehiculo;
class Checkpoint;
class CajaVida;
class Aceite;
class Barro;
class Boost;
class Piedra;

class ContactListener : public b2ContactListener {
private:
    Fisicas& fisicas_;

    void vehiculoVsArena(Vehiculo& vehiculo, SuperficieArena& arena);
    void vehiculoVsCheckpoint(Vehiculo& vehiculo, Checkpoint& checkpoint);
    void vehiculoVsVehiculo(Vehiculo& vehiculoA, Vehiculo& vehiculoB);
    void vehiculoVsCajaVida(Vehiculo& vehiculo, CajaVida& cajaVida);
    void vehiculoVsAceite(Vehiculo& vehiculo, Aceite& aceite);
    void vehiculoVsBarro(Vehiculo& vehiculo, Barro& barro);
    void vehiculoVsBoost(Vehiculo& vehiculo, Boost& boost);
    void vehiculoVsPiedra(Vehiculo& vehiculo, Piedra& piedra);

public:
    ContactListener(Fisicas& fisicas);
    virtual ~ContactListener();
    virtual void PreSolve(b2Contact* contact, const b2Manifold* oldManifold) ove
rride;
    virtual void BeginContact(b2Contact* contact) override;
    virtual void EndContact(b2Contact* contact) override;
    virtual void PostSolve(b2Contact* contact, const b2ContactImpulse* impulse)
override;
};

#endif
```

nov 26, 19 17:34 **B2DVehiculo.h** Page 1/1

```
#ifndef _B2D_VEHICULO_H_
#define _B2D_VEHICULO_H_

#include <cstdint>

#include "includes/3rd-party/Box2D/Box2D.h"

#define DENSIDAD 0.05f
#define MAX_IMPULSO_LATERAL 15.0f
#define CORRECCION_DERRAPE 0.03f
#define AJUSTE_VOLANTE 2.5f
#define AJUSTE_VELOCIDAD 0.45f
#define AJUSTE_ACELERACION 0.25f

//Forward declarations
class b2Body;
class b2World;
class Vehiculo;

class B2DVehiculo {
private:
    static const int volanteIzquierda_ = 0x1;
    static const int volanteDerecha_ = 0x2;
    static const int acelerador_ = 0x4;
    static const int freno_ = 0x8;
    int control_;

    b2Body* cuerpoBox2D_;

    float velocidadMaxAdelante_;
    float velocidadMaxAtras_;
    // Fuerza que se aplica para acelerar/frenar el auto
    float fuerzaManejoMaxima_;
    // AKA rozamiento, agarre
    float traccion_;

public:
    B2DVehiculo(b2World* mundoBox2D, Vehiculo& vehiculo);
    ~B2DVehiculo();
    b2Vec2 getVelocidadLateral();
    b2Vec2 getVelocidadFrontal();
    b2Body* getB2D();
    void actualizarFriccion();
    void actualizarAceleracion();
    void actualizarVolante();
    void step();
    void acelerando();
    void desacelerando();
    void frenando();
    void dejandoDeFrenar();
    void doblandoIzquierda();
    void dejandoDeDoblarIzquierda();
    void doblandoDerecha();
    void dejandoDeDoblarDerecha();
};

#endif
```

nov 26, 19 17:34 **Vehiculo.h** Page 1/1

```
#ifndef _VEHICULO_H_
#define _VEHICULO_H_

#include <vector>

#include "includes/servidor/modelo/Identificable.h"
#include "includes/servidor/modelo/Colisionable.h"
#include "includes/servidor/modelo/movimiento/Posicion.h"
#include "includes/servidor/Jugador.h"

typedef struct futuro {
    std::shared_ptr<Evento> evento;
    uint32_t steps;
} futuro_t;

class Vehiculo : public Identificable, public Colisionable {
private:
    unsigned int velocidadMaximaAdelante_;
    unsigned int velocidadMaximaAtras_;
    unsigned int aceleracion_;
    unsigned int maniobrabilidad_;
    unsigned int agarre_;
    unsigned int salud_;
    unsigned int saludDefault_;
    Posicion respawn_;
    std::shared_ptr<Jugador> duenio_;
    std::vector<futuro_t> futuros_;

public:
    Vehiculo(uint8_t uuid,
            unsigned int velocidadMaximaAdelante,
            unsigned int velocidadMaximaAtras,
            unsigned int aceleracion,
            unsigned int maniobrabilidad,
            unsigned int agarre,
            unsigned int salud,
            Posicion respawn,
            std::shared_ptr<Jugador> duenio);

    unsigned int velocidadMaximaAdelante();
    unsigned int velocidadMaximaAtras();
    unsigned int aceleracion();
    unsigned int maniobrabilidad();
    unsigned int agarre();
    unsigned int salud();

    std::shared_ptr<Jugador> duenio();

    void step();

    void ocurrira(std::shared_ptr<Evento> unEvento, uint32_t steps);

    bool disminuirSalud(uint8_t cantidad);
    void sumarSalud(int delta);

    virtual int getTipo() override;
    Posicion& getPuntoRespawn();
    void setPuntoRespawn(Posicion& Posicion);
};

#endif
```

nov 26, 19 17:34 **Piedra.h** Page 1/1

```
#ifndef _PIEDRA_H_
#define _PIEDRA_H_

#include "includes/servidor/modelo/entidades/Modificador.h"

class Piedra : public Modificador {
public:
    Piedra(uint8_t uuid);
    virtual int getTipo() override;
};

#endif
```

nov 26, 19 17:34 **Modificador.h** Page 1/1

```h
#ifndef _MODIFICADOR_H_
#define _MODIFICADOR_H_

#include "includes/servidor/modelo/Colisionable.h"
#include "includes/servidor/modelo/Identificable.h"

class Modificador : public Colisionable, public Identificable {
public:
    Modificador(uint8_t uuid);
    virtual ~Modificador();
};

#endif
```

nov 26, 19 17:34 **Checkpoint.h** Page 1/1

```h
#ifndef _CHECKPOINT_H_
#define _CHECKPOINT_H_

#include "includes/servidor/modelo/Colisionable.h"
#include "includes/servidor/modelo/movimiento/Posicion.h"

//Forward declarations
class Carrera;
class Vehiculo;

class Checkpoint : public Colisionable {
private:
    Carrera& carrera_;
    int id_;
    int idDelSiguiente_;
    float ancho_;
    float alto_;
    Posicion puntoRespawn_;

public:
    Checkpoint(Carrera& carrera, int id, int idDelSiguiente, float ancho, float alto, Posicion& posicion);
    virtual ~Checkpoint();
    virtual int getTipo() override;
    Posicion& posicion();
    float ancho();
    float alto();
    void registrarPaso(Vehiculo& vehiculo);
    int id();
};

#endif
```

nov 26, 19 17:34 **Carrera.h** Page 1/1

```
#ifndef _CARRERA_H_
#define _CARRERA_H_

#include <map>
#include <vector>

#include "includes/3rd-party/jsoncpp/json.hpp"
#include "includes/servidor/modelo/entidades/carrera/Checkpoint.h"
#include "includes/common/ColaProtegida.h"
#include "includes/common/eventos/Evento.h"

#define ID_META 0

//Forward declarations
class Vehiculo;

class Carrera {
private:
    std::map<int, Checkpoint> checkpoints_;
    std::map<uint8_t, int> idsVehiculosAidsCheckpoints_;
    std::map<uint8_t, int> idsVehiculosAVueltas_;
    int numeroDeVueltas_;
    std::vector<uint8_t> podio_;
    ColaProtegida<std::shared_ptr<Evento>>& eventosMundo_;

    bool finalizada();

public:
    Carrera(ColaProtegida<std::shared_ptr<Evento>>& eventosMundo);
    void cargarDesdeJson(Json& pistaJson);
    std::map<int, Checkpoint>& checkpoints();
    Checkpoint& ultimoCheckpointDe(Vehiculo& vehiculo);
    void setCheckpoint(Vehiculo& vehiculo, Checkpoint& checkpoint);
    void registrarVehiculo(Vehiculo& vehiculo);
};

#endif
```

nov 26, 19 17:34 **CajaVida.h** Page 1/1

```
#ifndef _CAJA_VIDA_H_
#define _CAJA_VIDA_H_

#include "includes/servidor/modelo/entidades/Modificador.h"

class CajaVida : public Modificador {
private:
    int deltaVida_;
public:
    CajaVida(uint8_t uuid, int deltaVida);
    virtual int getTipo() override;
    int deltaVida();
};

#endif
```

nov 26, 19 17:34 **Boost.h** Page 1/1

```
1  #ifndef _BOOST_H_
2  #define _BOOST_H_
3
4  #include "includes/servidor/modelo/entidades/Modificador.h"
5
6  class Boost : public Modificador {
7  public:
8      Boost(uint8_t uuid);
9      virtual int getTipo() override;
10 };
11
12 #endif
```

nov 26, 19 17:34 **Barro.h** Page 1/1

```
1  #ifndef _BARRO_H_
2  #define _BARRO_H_
3
4  #include "includes/servidor/modelo/entidades/Modificador.h"
5
6  class Barro : public Modificador {
7  public:
8      Barro(uint8_t uuid);
9      virtual int getTipo() override;
10 };
11
12 #endif
```

nov 26, 19 17:34 **Aceite.h** Page 1/1

```
#ifndef _ACEITE_H_
#define _ACEITE_H_

#include "includes/servidor/modelo/entidades/Modificador.h"

class Aceite : public Modificador {
public:
    Aceite(uint8_t uuid);
    virtual int getTipo() override;
};

#endif
```

nov 26, 19 17:34 **Colisionable.h** Page 1/1

```
#ifndef _COLISIONABLE_H_
#define _COLISIONABLE_H_

class Colisionable {
private:
    bool yaColisionado_;

public:
    Colisionable();
    virtual ~Colisionable();
    enum tipos {
        VEHICULO_ = 0,
        BARRO_ = 1,
        ACEITE_ = 2,
        PIEDRA_ = 3,
        SALUD_ = 4,
        BOOST_ = 5,
        SUPERFICIE_ARENA_ = 6,
        SUPERFICIE_PISTA_ = 7,
        SUPERFICIE_TIERRA_ = 8,
        CHECKPOINT_ = 9,
        META_ = 10
    };

    virtual int getTipo() = 0;
    void colisionado();
    bool yaFueColisionado();
};

#endif
```

nov 26, 19 17:34 **Jugador.h** Page 1/1

```
#ifndef _JUGADOR_H_
#define _JUGADOR_H_

#include <memory>

#include "includes/common/Handler.h"
#include "includes/common/red/SocketTCP.h"
#include "includes/common/ColaBloqueante.h"
#include "includes/common/eventos/Evento.h"
#include "includes/common/RecibidorEventos.h"
#include "includes/common/EnviadorEventos.h"

class Jugador {
private:
    uint32_t UUID_;
    SocketTCP socket_;
    ColaBloqueante<std::shared_ptr<Evento>>& destino_;
    ColaBloqueante<std::shared_ptr<Evento>> eventosAEnviar_;
    RecibidorEventos recibidorEventos_;
    EnviadorEventos enviadorEventos_;


public:
    Jugador(SocketTCP∧ socket, uint32_t uuid, ColaBloqueante<std::shared_ptr<Ev
ento>>& destinoEventos);
    ~Jugador();
    uint32_t uuid();
    //TODO: QUE ESTO SEA UNA ITNTERFAZ
    void ocurrio(std::shared_ptr<Evento> e);
};

#endif
```

nov 26, 19 17:34 **HiloAceptador.h** Page 1/1

```
#ifndef _HILO_ACEPTADOR_H_
#define _HILO_ACEPTADOR_H_

#include <string>

#include "includes/common/Hilo.h"
#include "includes/common/red/SocketTCP.h"
#include "includes/servidor/red/SocketTCPServidor.h"
#include "includes/servidor/SalaDeEspera.h"

class HiloAceptador : public Hilo {

private:
    SocketTCPServidor sktAceptador_;
    SalaDeEspera& salaDeEspera_;

    HiloAceptador(const HiloAceptador& otro) = delete;
    HiloAceptador(HiloAceptador∧ otro) = delete;
    HiloAceptador& operator=(HiloAceptador∧ otro) = delete;
    HiloAceptador& operator=(const HiloAceptador& otro) = delete;

public:
    HiloAceptador(const std::string& unHost, const std::string& puerto, SalaDeEs
pera& salaDeEspera);

    ~HiloAceptador();

    virtual void correr() override;
    virtual void detener() override;
};

#endif
```

nov 26, 19 17:34 **DistribuidorEventos.h** Page 1/1

```
#ifndef _DISTRIBUIDOR_EVENTOS_H_
#define _DISTRIBUIDOR_EVENTOS_H_

#include "includes/common/Hilo.h"
#include "includes/common/Handler.h"
#include "includes/servidor/SalaDeEspera.h"
#include "includes/servidor/CoordinadorPartidas.h"
#include "includes/common/ColaBloqueante.h"
#include "includes/common/eventos/Evento.h"

class DistribuidorEventos : public Hilo, public Handler {
private:
    ColaBloqueante<std::shared_ptr<Evento>>& eventos_;
    SalaDeEspera& salaDeEspera_;
    CoordinadorPartidas& coordinadorPartidas_;

    DistribuidorEventos(const DistribuidorEventos& otro) = delete;
    DistribuidorEventos(DistribuidorEventos∧ otro) = delete;
    DistribuidorEventos& operator=(DistribuidorEventos∧ otro) = delete;
    DistribuidorEventos& operator=(const DistribuidorEventos& otro) = delete;

public:
    DistribuidorEventos(ColaBloqueante<std::shared_ptr<Evento>>& eventos, SalaDeEspera& salaDeEspera, CoordinadorPartidas& coordinadorPartidas);

    ~DistribuidorEventos();

    virtual void correr() override;
    virtual void detener() override;

    virtual void manejar(Evento& e) override;

    virtual void manejar(EventoAcelerar& e) override;
    virtual void manejar(EventoDesacelerar& e) override;
    virtual void manejar(EventoFrenar& e) override;
    virtual void manejar(EventoDejarDeFrenar& e) override;
    virtual void manejar(EventoDoblarIzquierda& e) override;
    virtual void manejar(EventoDejarDeDoblarIzquierda& e) override;
    virtual void manejar(EventoDoblarDerecha& e) override;
    virtual void manejar(EventoDejarDeDoblarDerecha& e) override;
    virtual void manejar(EventoCrearPartida& e) override;
    virtual void manejar(EventoUnirseAPartida& e) override;
    virtual void manejar(EventoIniciarPartida& e) override;
    virtual void manejar(EventoUnirseASala& e) override;
    virtual void manejar(EventoDesconexion& e) override;
};

#endif
```

nov 26, 19 17:34 **CoordinadorPartidas.h** Page 1/1

```
#ifndef _COORDINADOR_PARTIDAS_H_
#define _COORDINADOR_PARTIDAS_H_

#include <map>
#include <memory>

#include "includes/servidor/Partida.h"
#include "includes/servidor/SalaDeEspera.h"
#include "includes/common/Handler.h"

class CoordinadorPartidas : public Handler{
private:
    uint16_t contadorPartidas_;
    SalaDeEspera& salaDeEspera_;
    std::map<uint16_t, std::shared_ptr<Partida>> partidas_;
    std::map<uint32_t, uint16_t> jugadoresAPartidas_;

public:
    CoordinadorPartidas(SalaDeEspera& salaDeEspera);
    ~CoordinadorPartidas();

    void agregarJugadorAPartida(std::shared_ptr<Jugador> jugador, uint16_t uuidPartida);

    std::shared_ptr<EventoSnapshotSala> getSnapshotSala();
    std::shared_ptr<EventoSnapshotLobby> getSnapshotLobby(uint16_t uuidPartida);

    virtual void manejar(Evento& e) override;
    virtual void manejar(EventoCrearPartida& e) override;
    virtual void manejar(EventoIniciarPartida& e) override;
    virtual void manejar(EventoDesconexion& e) override;
    virtual void manejar(EventoAcelerar& e) override;
    virtual void manejar(EventoDesacelerar& e) override;
    virtual void manejar(EventoFrenar& e) override;
    virtual void manejar(EventoDejarDeFrenar& e) override;
    virtual void manejar(EventoDoblarIzquierda& e) override;
    virtual void manejar(EventoDejarDeDoblarIzquierda& e) override;
    virtual void manejar(EventoDoblarDerecha& e) override;
    virtual void manejar(EventoDejarDeDoblarDerecha& e) override;
};

#endif
```

nov 26, 19 17:34 **Tile.h** Page 1/1

```
#ifndef _TILE_H_
#define _TILE_H_

class Tile {
public:
    int x_;
    int y_;

    Tile(int x, int y);
    bool operator <(const Tile& otroTile) const;
};

#endif
```

nov 26, 19 17:34 **SocketTCP.h** Page 1/1

```
#ifndef _SOCKET_TCP_H_
#define _SOCKET_TCP_H_

#include <cstddef>
#include <netdb.h>
#include <string>

#define IP_VERSION AF_INET
#define SOCKET_TYPE SOCK_STREAM
#define FLAGS 0

#define ERROR_CERRADO_S "Error al enviar. El socket estÃ¡ cerrado."
#define ERROR_CERRADO_R "Error al recibir. El socket estÃ¡ cerrado."
#define ERROR_SEND "Error al enviar. send() devolviÃ³ un valor menor a cero."
#define ERROR_RECV "Error al recibir. recv() devolviÃ³ un valor menor a cero."
#define ERROR_GET_ADDRINFO "Error al llamar a getaddrinfo."
#define ERROR_CREAR "Error al crear el socket."

class SocketTCP {
private:
    SocketTCP(const SocketTCP& otro) = delete;
    SocketTCP& operator=(const SocketTCP& otro) = delete;

protected:
    int fileDescripor_;
    addrinfo* hints_;

public:
    explicit SocketTCP(int unFileDescriptor);
    SocketTCP(const std::string& unHost, const std::string& unPuerto);
    SocketTCP(SocketTCP∧ otro);
    SocketTCP& operator=(SocketTCP∧ otro);
    ~SocketTCP();
    size_t enviarN(const char* buffer, size_t nBytes);
    size_t recibirN(char* buffer, size_t nBytes);
    void cerrarLectoEscritura();
};

#endif
```

nov 26, 19 17:34 **Protocolo.h** Page 1/1

```
1  #ifndef _PROTOCOLO_H_
2  #define _PROTOCOLO_H_
3
4  #include "includes/common/red/SocketTCP.h"
5
6  #define LEN_8 1
7  #define LEN_16 2
8  #define LEN_32 4
9
10 class Protocolo {
11 private:
12     SocketTCP& socket_;
13 public:
14     Protocolo(SocketTCP& socket);
15     uint8_t recibirNumUnsigned8();
16     uint16_t recibirNumUnsigned16();
17     uint32_t recibirNumUnsigned32();
18     void enviar(uint16_t numero);
19     void enviar(uint8_t numero);
20     void enviar(uint32_t numero);
21
22
23 };
24
25 #endif
```

nov 26, 19 17:34 **RecibidorEventos.h** Page 1/1

```
1  #ifndef _RECIBIDOR_EVENTOS_H_
2  #define _RECIBIDOR_EVENTOS_H_
3
4  #include "includes/common/Hilo.h"
5  #include "includes/common/red/SocketTCP.h"
6  #include "includes/common/Cola.h"
7  #include "includes/common/eventos/Evento.h"
8  #include "includes/common/red/Protocolo.h"
9
10 class RecibidorEventos : public Hilo {
11 private:
12     Cola<std::shared_ptr<Evento>>& destino_;
13     Protocolo protocolo_;
14     uint32_t UUIDRemitente_;
15
16 public:
17     RecibidorEventos(SocketTCP& origen, Cola<std::shared_ptr<Evento>>& destino,
   uint32_t uuidRemitente);
18     virtual void correr() override;
19     virtual void detener() override;
20 };
21
22 #endif
```

nov 26, 19 17:34 **Hilo.h** Page 1/1

```
#ifndef _THREAD_H_
#define _THREAD_H_

#include <thread>
#include <atomic>

class Hilo {
private:
    std::thread hilo_;

protected:
    std::atomic<bool> seguirCorriendo_;

public:
    Hilo();

    void iniciar();

    void dormir(double milisegundos);

    bool estaCorriendo();

    virtual void correr() = 0;

    // TODO: Refactorizar y hacer un detener default, virtual. Que solo lo overr
idee un hilo que lance mas hilos.
    virtual void detener() = 0;

    virtual void join();

    virtual ~Hilo();

    Hilo(const Hilo&) = delete;

    Hilo& operator=(const Hilo&) = delete;

    Hilo(Hilo∧ other);

    Hilo& operator=(Hilo∧ other);
};

#endif
```

nov 26, 19 17:34 **Handler.h** Page 1/1

```
#ifndef _HANDLER_H_
#define _HANDLER_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/eventos/EventoAcelerar.h"
#include "includes/common/eventos/EventoDesacelerar.h"
#include "includes/common/eventos/EventoFrenar.h"
#include "includes/common/eventos/EventoDejarDeFrenar.h"
#include "includes/common/eventos/EventoDoblarIzquierda.h"
#include "includes/common/eventos/EventoDejarDeDoblarIzquierda.h"
#include "includes/common/eventos/EventoDoblarDerecha.h"
#include "includes/common/eventos/EventoDejarDeDoblarDerecha.h"
#include "includes/common/eventos/EventoSnapshot.h"
#include "includes/common/eventos/EventoCrearPartida.h"
#include "includes/common/eventos/EventoUnirseAPartida.h"
#include "includes/common/eventos/EventoIniciarPartida.h"
#include "includes/common/eventos/EventoPartidaIniciada.h"
#include "includes/common/eventos/EventoDesconexion.h"
#include "includes/common/eventos/EventoFinCarrera.h"
#include "includes/common/eventos/EventoAparecioConsumible.h"
#include "includes/common/eventos/EventoDesaparecioConsumible.h"
#include "includes/common/eventos/EventoChoque.h"
#include "includes/common/eventos/EventoBarroPisado.h"
#include "includes/common/eventos/EventoExplosion.h"
#include "includes/common/eventos/EventoFinBarro.h"
#include "includes/common/eventos/EventoPartidaCreada.h"
#include "includes/common/eventos/EventoSnapshotLobby.h"
#include "includes/common/eventos/EventoSnapshotSala.h"
#include "includes/common/eventos/EventoUnirseASala.h"
#include "includes/common/eventos/EventoFrenada.h"

class Handler {
 public:
  virtual ~Handler();

  virtual void manejar(Evento &e) = 0;
  virtual void manejar(EventoAcelerar &e);
  virtual void manejar(EventoDesacelerar &e);
  virtual void manejar(EventoFrenar &e);
  virtual void manejar(EventoDejarDeFrenar &e);
  virtual void manejar(EventoDoblarIzquierda &e);
  virtual void manejar(EventoDejarDeDoblarIzquierda &e);
  virtual void manejar(EventoDoblarDerecha &e);
  virtual void manejar(EventoDejarDeDoblarDerecha &e);
  virtual void manejar(EventoSnapshot &e);
  virtual void manejar(EventoCrearPartida &e);
  virtual void manejar(EventoUnirseAPartida &e);
  virtual void manejar(EventoIniciarPartida &e);
  virtual void manejar(EventoPartidaIniciada &e);
  virtual void manejar(EventoFinCarrera &e);
  virtual void manejar(EventoAparecioConsumible &e);
  virtual void manejar(EventoDesaparecioConsumible &e);
  virtual void manejar(EventoDesconexion &e);
  virtual void manejar(EventoChoque &e);
  virtual void manejar(EventoBarroPisado &e);
  virtual void manejar(EventoExplosion &e);
  virtual void manejar(EventoFinBarro &e);
  virtual void manejar(EventoPartidaCreada &e);
  virtual void manejar(EventoSnapshotLobby &e);
  virtual void manejar(EventoSnapshotSala &e);
  virtual void manejar(EventoUnirseASala &e);
  virtual void manejar(EventoFrenada &e);
};

#endif
```

nov 26, 19 17:34 **EventoDesconocidoError.h** Page 1/1

```
#ifndef _EVENTO_DESCONOCIDO_ERROR_H_
#define _EVENTO_DESCONOCIDO_ERROR_H_

#include <stdexcept>
#include <string>

class EventoDesconocidoError : public std::runtime_error {
public:
    EventoDesconocidoError(const std::string& mensaje);
};

#endif
```

nov 26, 19 17:34 **EventoUnirseASala.h** Page 1/1

```
#ifndef _EVENTO_UNIRSE_A_SALA_H_
#define _EVENTO_UNIRSE_A_SALA_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoUnirseASala : public Evento {
public:
    EventoUnirseASala(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoUnirseASala();
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoUnirseAPartida.h** Page 1/1

```
#ifndef _EVENTO_UNIRSE_A_PARTIDA_H_
#define _EVENTO_UNIRSE_A_PARTIDA_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoUnirseAPartida : public Evento {
public:
    uint16_t uuidPartida_;

    EventoUnirseAPartida(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoUnirseAPartida(uint32_t uuidRemitente, uint16_t uuidPartida);
    EventoUnirseAPartida(uint16_t uuidPartida);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoSnapshotSala.h** Page 1/1

```
#ifndef _EVENTO_SNAPSHOT_SALA_H_
#define _EVENTO_SNAPSHOT_SALA_H_

#include <map>

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoSnapshotSala : public Evento {
public:
    uint16_t cantidadPartidas_;
    std::map<uint16_t, uint16_t> ordinalAuuidPartida_;

    EventoSnapshotSala(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoSnapshotSala(std::map<uint16_t, uint16_t>∧ datos);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoSnapshotLobby.h** Page 1/1

```
#ifndef _EVENTO_SNAPSHOT_LOBBY_H_
#define _EVENTO_SNAPSHOT_LOBBY_H_

#include <map>

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoSnapshotLobby : public Evento {
public:
    uint8_t cantidadJugadores_;
    std::map<uint32_t, bool> idJugadorAEstaListo_;

    EventoSnapshotLobby(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoSnapshotLobby(std::map<uint32_t, bool>∧ datos);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoSnapshot.h** Page 1/1

```
#ifndef _EVENTO_SNAPSHOT_H_
#define _EVENTO_SNAPSHOT_H_

#include <map>

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

typedef struct {
    float xCoord_;
    float yCoord_;
    uint16_t angulo_;
    uint8_t salud_;
    uint8_t visible_;
} datosVehiculo_;

class EventoSnapshot : public Evento {
public:
    //TODO: Esto queda por si queremos usar tamaÃ±o fijo para envio recepcion mas eficiente
    uint8_t cantidadVehiculos_;
    std::map<uint8_t, datosVehiculo_> idsADatosVehiculos_;

    EventoSnapshot(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoSnapshot(std::map<uint8_t, datosVehiculo_>∧ datos);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
    void enviarSoloDatos(Protocolo& protocolo);
};

#endif
```

nov 26, 19 17:34 **EventoPartidaIniciada.h** Page 1/1

```
#ifndef _EVENTO_PARTIDA_INICIADA_H_
#define _EVENTO_PARTIDA_INICIADA_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/eventos/EventoSnapshot.h"
#include "includes/common/red/Protocolo.h"

class EventoPartidaIniciada : public Evento {

public:
    uint8_t idDelVehiculo_;
    EventoSnapshot estadoInicial_;

    EventoPartidaIniciada(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoPartidaIniciada(uint8_t idDelVehiculo, std::map<uint8_t, datosVehiculo
_>^ datos);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoPartidaCreada.h** Page 1/1

```
#ifndef _EVENTO_PARTIDA_CREADA_H_
#define _EVENTO_PARTIDA_CREADA_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoPartidaCreada : public Evento {
public:
    uint16_t uuidPartida_;
    uint32_t uuidCreador_;

    EventoPartidaCreada(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoPartidaCreada(uint16_t uuidPartida, uint32_t uuidCreador);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoIniciarPartida.h** Page 1/1

```h
#ifndef _EVENTO_INICIAR_PARTIDA_H_
#define _EVENTO_INICIAR_PARTIDA_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoIniciarPartida : public Evento {
public:
    EventoIniciarPartida(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoIniciarPartida();
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **Evento.h** Page 1/1

```h
#ifndef _EVENTO_H_
#define _EVENTO_H_

#include "includes/common/red/Protocolo.h"

//Forward declarations
class Handler;

#define UUID_EVENTO_ACELERAR                    1
#define UUID_EVENTO_DESACELERAR                 2
#define UUID_EVENTO_FRENAR                      3
#define UUID_EVENTO_DEJAR_DE_FRENAR             4
#define UUID_EVENTO_DOBLAR_IZQUIERDA            5
#define UUID_EVENTO_DEJAR_DE_DOBLAR_IZQUIERDA   6
#define UUID_EVENTO_DOBLAR_DERECHA              7
#define UUID_EVENTO_DEJAR_DE_DOBLAR_DERECHA     8

#define UUID_EVENTO_CREAR_PARTIDA               9
#define UUID_EVENTO_PARTIDA_AGREGADA            10
#define UUID_EVENTO_UNIRSE_A_PARTIDA            11
#define UUID_EVENTO_INICIAR_PARTIDA             12
#define UUID_EVENTO_DESCONEXION                 13
#define UUID_EVENTO_PARTIDA_INICIADA            14
#define UUID_EVENTO_SNAPSHOT                    15
#define UUID_EVENTO_FIN_CARRERA                 16
#define UUID_EVENTO_APARECIO_CONSUMIBLE         17
#define UUID_EVENTO_BARRO_PISADO                18
#define UUID_EVENTO_CHOQUE                      19
#define UUID_EVENTO_DESAPARECIO_CONSUMIBLE      20
#define UUID_EVENTO_EXPLOSION                   21
#define UUID_EVENTO_FIN_BARRO                   22
#define UUID_EVENTO_PARTIDA_CREADA              23
#define UUID_EVENTO_SNAPSHOT_LOBBY              24
#define UUID_EVENTO_SNAPSHOT_SALA               25
#define UUID_EVENTO_UNIRSE_A_SALA               26
#define UUID_EVENTO_FRENADA                     27


// TODO: Crear constructor por movimiento
//TODO: Setear id en cada evento creado
// TODO: implementar envio y construccion
// TODO: achicar para que el envio sea mas eficiente segÃºn el mÃ¡ximo de eventos
class Evento {
private:
    uint32_t UUIDRemitente_;
protected:
    uint16_t UUIDEvento_;
public:
    Evento(uint32_t uuidRemitente);
    virtual ~Evento() {}
    virtual void enviarse(Protocolo& protocolo) = 0;
    virtual void actualizar(Handler& handler) = 0;
    uint32_t uuidRemitente();
    void setRemitente(uint32_t uuidRemitente);
};

#endif
```

nov 26, 19 17:34 **EventoFrenar.h** Page 1/1

```
#ifndef _EVENTO_FRENAR_H_
#define _EVENTO_FRENAR_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoFrenar : public Evento {
public:
    EventoFrenar(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoFrenar();
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoFrenada.h** Page 1/1

```
#ifndef _EVENTO_FRENADA_H_
#define _EVENTO_FRENADA_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoFrenada : public Evento {
public:

    float xCoord_;
    float yCoord_;

    EventoFrenada(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoFrenada(float xCoord, float yCoord);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoFinCarrera.h** Page 1/1

```
#ifndef _EVENTO_FIN_CARRERA_H_
#define _EVENTO_FIN_CARRERA_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

#include <vector>

class EventoFinCarrera : public Evento {
public:

    std::vector<uint8_t> podio_;

    EventoFinCarrera(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoFinCarrera(std::vector<uint8_t>^ podio);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoFinBarro.h** Page 1/1

```
#ifndef _EVENTO_FIN_BARRO_H_
#define _EVENTO_FIN_BARRO_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoFinBarro : public Evento {
public:
    EventoFinBarro(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoFinBarro();
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoFactory.h** Page 1/1

```
#ifndef _EVENTO_FACTORY_H_
#define _EVENTO_FACTORY_H_

#include <memory>

#include "includes/common/red/Protocolo.h"
#include "includes/common/eventos/Evento.h"
#include "includes/common/eventos/EventoPartidaIniciada.h"
#include "includes/common/eventos/EventoCrearPartida.h"
#include "includes/common/eventos/EventoUnirseAPartida.h"
#include "includes/common/eventos/EventoIniciarPartida.h"
#include "includes/common/eventos/EventoDesconexion.h"
#include "includes/common/eventos/EventoAcelerar.h"
#include "includes/common/eventos/EventoDesacelerar.h"
#include "includes/common/eventos/EventoFrenar.h"
#include "includes/common/eventos/EventoDejarDeFrenar.h"
#include "includes/common/eventos/EventoDoblarIzquierda.h"
#include "includes/common/eventos/EventoDejarDeDoblarIzquierda.h"
#include "includes/common/eventos/EventoDoblarDerecha.h"
#include "includes/common/eventos/EventoDejarDeDoblarDerecha.h"
#include "includes/common/eventos/EventoSnapshot.h"
#include "includes/common/eventos/EventoFinCarrera.h"
#include "includes/common/eventos/EventoAparecioConsumible.h"
#include "includes/common/eventos/EventoDesaparecioConsumible.h"
#include "includes/common/eventos/EventoChoque.h"
#include "includes/common/eventos/EventoBarroPisado.h"
#include "includes/common/eventos/EventoExplosion.h"
#include "includes/common/eventos/EventoFinBarro.h"
#include "includes/common/eventos/EventoPartidaCreada.h"
#include "includes/common/eventos/EventoSnapshotLobby.h"
#include "includes/common/eventos/EventoSnapshotSala.h"
#include "includes/common/eventos/EventoUnirseASala.h"
#include "includes/common/eventos/EventoFrenada.h"

#define ERROR_EVENTO_DESCONOCIDO "Error al instanciar evento, se utilizÃ³ un UUID desconocido."

class EventoFactory {
private:
public:
    static std::shared_ptr<Evento> instanciar(uint32_t uuidRemitente, Protocolo&
 protocolo);
};

#endif
```

nov 26, 19 17:34 **EventoExplosion.h** Page 1/1

```
#ifndef _EVENTO_EXPLOSION_H_
#define _EVENTO_EXPLOSION_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoExplosion : public Evento {
public:

    float xCoord_;
    float yCoord_;

    EventoExplosion(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoExplosion(float xCoord, float yCoord);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoDoblarIzquierda.h** Page 1/1

```
1  #ifndef _EVENTO_DOBLAR_IZQUIERDA_H_
2  #define _EVENTO_DOBLAR_IZQUIERDA_H_
3
4  #include "includes/common/eventos/Evento.h"
5  #include "includes/common/red/Protocolo.h"
6
7  class EventoDoblarIzquierda : public Evento {
8  public:
9      EventoDoblarIzquierda(uint32_t uuidRemitente, Protocolo& protocolo);
10     EventoDoblarIzquierda();
11     virtual void enviarse(Protocolo& protocolo) override;
12     virtual void actualizar(Handler& handler) override;
13 };
14
15 #endif
```

nov 26, 19 17:34 **EventoDoblarDerecha.h** Page 1/1

```
1  #ifndef _EVENTO_DOBLAR_DERECHA_H_
2  #define _EVENTO_DOBLAR_DERECHA_H_
3
4  #include "includes/common/eventos/Evento.h"
5  #include "includes/common/red/Protocolo.h"
6
7  class EventoDoblarDerecha : public Evento {
8  public:
9      EventoDoblarDerecha(uint32_t uuidRemitente, Protocolo& protocolo);
10     EventoDoblarDerecha();
11     virtual void enviarse(Protocolo& protocolo) override;
12     virtual void actualizar(Handler& handler) override;
13 };
14
15 #endif
```

nov 26, 19 17:34 **EventoDesconexion.h** Page 1/1

```
#ifndef _EVENTO_DESCONEXION_H_
#define _EVENTO_DESCONEXION_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoDesconexion : public Evento {
public:
    EventoDesconexion(uint32_t uuidRemitente);
    EventoDesconexion(uint32_t uuidRemitente, Protocolo& protocolo);
    virtual void enviarse(Protocolo& protocolo);
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoDesaparecioConsumible.h** Page 1/1

```
#ifndef _EVENTO_DESAPARECIO_CONSUMIBLE_H_
#define _EVENTO_DESAPARECIO_CONSUMIBLE_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoDesaparecioConsumible : public Evento {
public:
    uint8_t uuidConsumible_;

    EventoDesaparecioConsumible(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoDesaparecioConsumible(uint8_t uuidConsumible);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoDesacelerar.h** Page 1/1

```
1  #ifndef _EVENTO_DESACELERAR_H_
2  #define _EVENTO_DESACELERAR_H_
3
4  #include "includes/common/eventos/Evento.h"
5  #include "includes/common/red/Protocolo.h"
6
7  class EventoDesacelerar : public Evento {
8  public:
9      EventoDesacelerar(uint32_t uuidRemitente, Protocolo& protocolo);
10     EventoDesacelerar();
11     virtual void enviarse(Protocolo& protocolo) override;
12     virtual void actualizar(Handler& handler) override;
13 };
14
15 #endif
```

nov 26, 19 17:34 **EventoDejarDeFrenar.h** Page 1/1

```
1  #ifndef _EVENTO_DEJAR_DE_FRENAR_H_
2  #define _EVENTO_DEJAR_DE_FRENAR_H_
3
4  #include "includes/common/eventos/Evento.h"
5  #include "includes/common/red/Protocolo.h"
6
7  class EventoDejarDeFrenar : public Evento {
8  public:
9      EventoDejarDeFrenar(uint32_t uuidRemitente, Protocolo& protocolo);
10     EventoDejarDeFrenar();
11     virtual void enviarse(Protocolo& protocolo) override;
12     virtual void actualizar(Handler& handler) override;
13 };
14
15 #endif
```

nov 26, 19 17:34 **EventoDejarDeDoblarIzquierda.h** Page 1/1

```
1  #ifndef _EVENTO_DEJAR_DE_DOBLAR_IZQUIERDA_H_
2  #define _EVENTO_DEJAR_DE_DOBLAR_IZQUIERDA_H_
3
4  #include "includes/common/eventos/Evento.h"
5  #include "includes/common/red/Protocolo.h"
6
7  class EventoDejarDeDoblarIzquierda : public Evento {
8  public:
9      EventoDejarDeDoblarIzquierda(uint32_t uuidRemitente, Protocolo& protocolo);
10     EventoDejarDeDoblarIzquierda();
11     virtual void enviarse(Protocolo& protocolo) override;
12     virtual void actualizar(Handler& handler) override;
13 };
14
15 #endif
```

nov 26, 19 17:34 **EventoDejarDeDoblarDerecha.h** Page 1/1

```
1  #ifndef _EVENTO_DEJAR_DE_DOBLAR_DERECHA_H_
2  #define _EVENTO_DEJAR_DE_DOBLAR_DERECHA_H_
3
4  #include "includes/common/eventos/Evento.h"
5  #include "includes/common/red/Protocolo.h"
6
7  class EventoDejarDeDoblarDerecha : public Evento {
8  public:
9      EventoDejarDeDoblarDerecha(uint32_t uuidRemitente, Protocolo& protocolo);
10     EventoDejarDeDoblarDerecha();
11     virtual void enviarse(Protocolo& protocolo) override;
12     virtual void actualizar(Handler& handler) override;
13 };
14
15 #endif
```

nov 26, 19 17:34 **EventoCrearPartida.h** Page 1/1

```
#ifndef _EVENTO_CREAR_PARTIDA_H_
#define _EVENTO_CREAR_PARTIDA_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoCrearPartida : public Evento {
 public:
  EventoCrearPartida(uint32_t uuidRemitente, Protocolo &protocolo);
  EventoCrearPartida();
  virtual void enviarse(Protocolo &protocolo) override;
  virtual void actualizar(Handler &handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoChoque.h** Page 1/1

```
#ifndef _EVENTO_CHOQUE_H_
#define _EVENTO_CHOQUE_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoChoque : public Evento {
public:

    float xCoord_;
    float yCoord_;

    EventoChoque(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoChoque(float xCoord, float yCoord);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoBarroPisado.h** Page 1/1

```
#ifndef _EVENTO_BARRO_PISADO_H_
#define _EVENTO_BARRO_PISADO_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoBarroPisado : public Evento {
public:
    EventoBarroPisado(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoBarroPisado();
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoAparecioConsumible.h** Page 1/1

```
#ifndef _EVENTO_APARECIO_CONSUMIBLE_H_
#define _EVENTO_APARECIO_CONSUMIBLE_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

#define UUID_PIEDRA     1
#define UUID_VIDA       2
#define UUID_ACEITE     3
#define UUID_BARRO      4
#define UUID_BOOST      5

class EventoAparecioConsumible : public Evento {
public:
    uint8_t uuidConsumible_;
    uint8_t tipoConsumible_;
    float xCoord_;
    float yCoord_;

    EventoAparecioConsumible(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoAparecioConsumible(uint8_t uuidConsumible, uint8_t tipoConsumible, flo
at xCoord, float yCoord);
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoAcelerar.h** Page 1/1

```
#ifndef _EVENTO_ACELERAR_H_
#define _EVENTO_ACELERAR_H_

#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EventoAcelerar : public Evento {
public:
    EventoAcelerar(uint32_t uuidRemitente, Protocolo& protocolo);
    EventoAcelerar();
    virtual void enviarse(Protocolo& protocolo) override;
    virtual void actualizar(Handler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EnviadorEventos.h** Page 1/1

```
#ifndef _ENVIADOR_EVENTOS_H_
#define _ENVIADOR_EVENTOS_H_

#include "includes/common/Hilo.h"
#include "includes/common/red/SocketTCP.h"
#include "includes/common/ColaBloqueante.h"
#include "includes/common/eventos/Evento.h"
#include "includes/common/red/Protocolo.h"

class EnviadorEventos : public Hilo {
private:
    ColaBloqueante<std::shared_ptr<Evento>>& origen_;
    Protocolo protocolo_;

public:
    EnviadorEventos(SocketTCP& socketDestino, ColaBloqueante<std::shared_ptr<Eve
nto>>& origen);
    virtual void correr() override;
    virtual void detener() override;
};

#endif
```

nov 26, 19 17:34 **Cronometro.h** Page 1/1

```
1  #ifndef _CRONOMETRO_H_
2  #define _CRONOMETRO_H_
3
4  class Cronometro {
5  public:
6      double ahora();
7  };
8
9  #endif
```

nov 26, 19 17:34 **Conversor.h** Page 1/1

```
1  #ifndef _CONVERSOR_H_
2  #define _CONVERSOR_H_
3
4  /*
5   * Conversor que realiza el traspaso
6   * de metros a pixeles y a bloques
7   */
8  class
9  Conversor {
10  private:
11   float pixelPorMetro;
12   int pixelPorBloque;
13
14  public:
15   Conversor(float pixelPorMetro, int pixelPorBloque);
16   int metroAPixel(float coord);
17   float pixelAMetro(int coord);
18   int bloqueAPixel(int coord);
19   int pixelABloque(int coord);
20   //TODO: Refactor this
21   static float tileAMetro(float coordenada) {
22     return 10.0f * coordenada;
23   }
24
25 };
26 #endif
```

nov 26, 19 17:34 **Constantes.h** Page 1/1

```
#ifndef _CONSTANTES_H_
#define _CONSTANTES_H_

#define ID_TIERRA 100
#define ID_PASTO 101
#define ID_ASFALTO_RECTO 102
#define ID_ASFALTO_CURVA 104
#define ID_CAR 105
#define ID_SALUD 106
#define ID_BOOST 107
#define ID_PIEDRA 108
#define ID_ACEITE 109
#define ID_BARRO 110

#endif
```

nov 26, 19 17:34 **ColaProtegida.h** Page 1/1

```
#ifndef _COLA_PROTEGIDA_H_
#define _COLA_PROTEGIDA_H_

#include <mutex>

#include "includes/common/Cola.h"

template <class T>
class ColaProtegida : public Cola<T> {
private:
    std::mutex mtx_;

    ColaProtegida(ColaProtegida^ otra) = delete;

    ColaProtegida(const ColaProtegida& otra) = delete;

    ColaProtegida& operator=(const ColaProtegida& otra) = delete;

    ColaProtegida& operator=(ColaProtegida^ otra) = delete;

    std::queue<T> elementos_;

    public:
    ColaProtegida() {
    }

    ~ColaProtegida() {
    }

    void put(T unElemento) override {
        std::lock_guard<std::mutex> lck(mtx_);
        elementos_.push(unElemento);
    }

    bool get(T& unElemento) override {
        std::unique_lock<std::mutex> lck(mtx_);
        if (elementos_.empty()) {
            return false;
        }
        unElemento = elementos_.front();
        elementos_.pop();
        return true;
    }
};

#endif
```

nov 26, 19 17:34 **ColaNoProtegida.h** Page 1/1

```
#ifndef _COLA_NO_PROTEGIDA_H_
#define _COLA_NO_PROTEGIDA_H_

#include "includes/common/Cola.h"

template <class T>
class ColaNoProtegida : public Cola<T> {
private:
    ColaNoProtegida(ColaNoProtegida∧ otra) = delete;
    ColaNoProtegida(const ColaNoProtegida& otra) = delete;
    ColaNoProtegida& operator=(const ColaNoProtegida& otra) = delete;
    ColaNoProtegida& operator=(ColaNoProtegida∧ otra) = delete;

    std::queue<T> elementos_;

    public:
    ColaNoProtegida() {
    }

    ~ColaNoProtegida() {
    }

    void put(T unElemento) override {
        elementos_.push(std::move(unElemento));
    }

    bool get(T& unElemento) override {
        if (elementos_.empty()) {
            return false;
        }
        unElemento = std::move(elementos_.front());
        elementos_.pop();
        return true;
    }
};

#endif
```

nov 26, 19 17:34 **Cola.h** Page 1/1

```
#ifndef _COLA_H_
#define _COLA_H_

#include <queue>

template <class T>
class Cola {
private:
    Cola(Cola∧ otra) = delete;
    Cola(const Cola& otra) = delete;
    Cola& operator=(const Cola& otra) = delete;
    Cola& operator=(Cola∧ otra) = delete;

public:
    Cola() {
    }
    virtual ~Cola() {
    }

    virtual void put(T unElemento) = 0;

    virtual bool get(T& unElemento) = 0;
};

#endif
```

nov 26, 19 17:34 **ColaBloqueante.h** Page 1/1

```
#ifndef _COLA_BLOQUEANTE_H_
#define _COLA_BLOQUEANTE_H_

#include <mutex>
#include <condition_variable>
#include <atomic>

#include "includes/common/Cola.h"

template <class T>
class ColaBloqueante : public Cola<T> {
private:
    std::mutex mtx_;
    std::condition_variable cond_;
    std::atomic<bool> detenida_;
    std::queue<T> elementos_;

    ColaBloqueante(ColaBloqueante∧ otra) = delete;

    ColaBloqueante(const ColaBloqueante& otra) = delete;

    ColaBloqueante& operator=(const ColaBloqueante& otra) = delete;

    ColaBloqueante& operator=(ColaBloqueante∧ otra) = delete;

    public:
    ColaBloqueante() :
        detenida_(false) {
    }

    ~ColaBloqueante() {
    }

    void put(T unElemento) override {
        std::lock_guard<std::mutex> lck(mtx_);
        elementos_.push(unElemento);
        cond_.notify_one();
    }

    bool get(T& unElemento) override {
        std::unique_lock<std::mutex> lck(mtx_);
        cond_.wait(lck, [this]{return ¬elementos_.empty() ∨ detenida_ ;});
        if (detenida_) {
            return false;
        }
        unElemento = elementos_.front();
        elementos_.pop();
        return true;
    }

    void detener() {
        detenida_ = true;
        cond_.notify_all();
    }
};

#endif
```

nov 26, 19 17:34 **DobleBuffer.h** Page 1/1

```
#ifndef _DOBLE_BUFFER_H_
#define _DOBLE_BUFFER_H_


#include <mutex>
#include <condition_variable>
#include <atomic>

template <typename T>
class DobleBuffer {
private:
    T datos_[2];
    int actual_{0};
    std::mutex mutex_;
    std::condition_variable cond_;
    std::atomic<bool> detenido_, nuevo_dato_;
public:
    DobleBuffer() : detenido_(false), nuevo_dato_(false) {}
    ~DobleBuffer() {}

    void swap();
    void set(T∧ instance);
    bool get(T& destino);
    void detener();
};


template <typename T>
void DobleBuffer<T>::set(T∧ instance) {
    std::unique_lock<std::mutex> lock(mutex_);
    datos_[¬actual_] = std::move(instance);
    nuevo_dato_ = true;
    cond_.notify_one();
}


template <typename T>
bool DobleBuffer<T>::get(T& destino) {
    std::unique_lock<std::mutex> lck(mutex_);
    cond_.wait(lck, [this]{return nuevo_dato_ ∨ detenido_;});
    if (detenido_) {
        nuevo_dato_ = detenido_ = false;
        return false;
    }
    actual_ = ¬actual_;
    destino = datos_[actual_];
    nuevo_dato_ = false;
    return true;
}

template <typename T>
void DobleBuffer<T>::detener(){
    detenido_ = true;
    cond_.notify_all();
}


#endif
```

```
#ifndef _CONFIG_CLIENTE_H_
#define _CONFIG_CLIENTE_H_

#define RUTA_CONFIG_CLIENTE "config/client_settings.json"

#define CONFIG_CLIENTE ConfigCliente::instancia()

#include <string>
#include <vector>

#include "includes/3rd-party/jsoncpp/json.hpp"

class ConfigCliente {
 private:
  ConfigCliente(const std::string &rutaArchivo);
  Json json_;

 public:
  static ConfigCliente &instancia();

  unsigned int anchoVentana();
  unsigned int altoVentana();
  float factorLejaniaCamara();
  bool pantallaCompleta();
  std::string tituloVentana();
  unsigned int fps();

  std::string host();
  std::string puerto();

  std::string fuente();

  std::string musicaAmbiente();
  std::string musicaMotor();
  std::string musicaExplosion();
  std::string musicaVacio();
  std::string musicaChoque();
  std::string musicaFrenada();

  unsigned int volumenAmbiente();

  unsigned int anchoBloquesPista();
  unsigned int altoBloquesPista();

  double pixelPorMetro();
  unsigned int pixelPorBloque();

  std::string texto(std::string sector);
  unsigned int tamanioTexto(std::string sector);
  unsigned int anchoTexto(std::string sector);
  double margenX(std::string sector);
  double margenY(std::string sector);

  unsigned int uuid(std::string nombreAnimacion);
  unsigned int ancho(std::string nombreAnimacion);
  unsigned int alto(std::string nombreAnimacion);
  std::vector<std::string> sprites(std::string nombreAnimacion);

  unsigned int anchoGrabadora();
  unsigned int altoGrabadora();
  std::string formatoGrabadora();
  uint32_t fpsGrabadora();
  unsigned int bitrateGrabadora();


```



```
  std::string rutaLuaScriptUsuario();
  std::string rutaLuaScript();
  int tiempoReaccionHumano();
};

#endif
```

nov 26, 19 17:34 **SocketTCPCliente.h** Page 1/1

```
#ifndef _SOCKET_TCP_CLIENTE_H_
#define _SOCKET_TCP_CLIENTE_H_

#include "includes/common/red/SocketTCP.h"

#include <string>

#define ERROR_CONEXION "Error al intentar conectar el socket."

class SocketTCPCliente : public SocketTCP {
public:
    SocketTCPCliente(const std::string& unHost, const std::string& unPuerto);

    void conectar();
};

#endif
```

nov 26, 19 17:34 **Jugador.h** Page 1/1

```
#ifndef __JUGADOR_H__
#define __JUGADOR_H__

#include <memory>
#include "includes/common/ColaBloqueante.h"
#include "includes/common/eventos/Evento.h"
#include "includes/common/eventos/EventoAcelerar.h"
#include "includes/common/eventos/EventoDesacelerar.h"
#include "includes/common/eventos/EventoDoblarDerecha.h"
#include "includes/common/eventos/EventoDejarDeDoblarDerecha.h"
#include "includes/common/eventos/EventoDoblarIzquierda.h"
#include "includes/common/eventos/EventoDejarDeDoblarIzquierda.h"
#include "includes/common/eventos/EventoFrenar.h"
#include "includes/common/eventos/EventoDejarDeFrenar.h"


class Jugador {
private:
  ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_;
public:
  Jugador(ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar);

  virtual void setEstado(float x, float y, uint16_t angulo);
  virtual void empezar();
  virtual void terminar();


  virtual void acelerar();

  virtual void desacelerar();

  virtual void frenar();

  virtual void dejarDeFrenar();

  virtual void doblarDerecha();

  virtual void dejarDeDoblarDerecha();

  virtual void doblarIzquierda();

  virtual void dejarDeDoblarIzquierda();
};


#endif
```

nov 26, 19 17:34 **Computadora.h** Page 1/1

```
#ifndef __COMPUTADORA_H__
#define __COMPUTADORA_H__

#include <exception>
#include <string>
#include <atomic>

#include "includes/cliente/jugadores/Jugador.h"

#include "includes/common/Hilo.h"
#include "includes/3rd-party/lua/LuaInterprete.hpp"


class Computadora : public Jugador, Hilo {
private:
  LuaInterpreter lua;
  std::atomic<uint_least16_t> x_;
  std::atomic<uint_least16_t> y_;
  std::atomic<uint_least16_t> angulo_;
  int last_command_ = -1;
  void leave_command();
  void do_command();

public:
  Computadora(ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar,
                          std::string fileName);

  virtual void setEstado(float x, float y, uint16_t angulo) override ;
  virtual void empezar() override ;
  virtual void terminar() override ;


  virtual void detener() override ;
  virtual void correr() override ;

// eventos humano
  virtual void acelerar();
  virtual void desacelerar();
  virtual void frenar();
  virtual void dejarDeFrenar();
  virtual void doblarDerecha();
  virtual void dejarDeDoblarDerecha();
  virtual void doblarIzquierda();
  virtual void dejarDeDoblarIzquierda();
};

#endif
```

nov 26, 19 17:34 **Ventana.h** Page 1/1

```
#ifndef _VENTANA_H_
#define _VENTANA_H_

#include <string>

#include "includes/cliente/GUI/Area.h"

//Forward declarations:
class SDL_Window;

class Ventana {
private:
    SDL_Window* ventanaSDL_;
    unsigned int ancho_;
    unsigned int alto_;
    bool fullscreen_;

    Ventana(const Ventana&) = delete;
    Ventana& operator=(const Ventana&) = delete;

public:
    Ventana(unsigned int ancho, unsigned int alto, bool pantallaCompleta, const std::string& tituloVentana);
    ~Ventana();

    Area dimensiones();
    unsigned int ancho();
    unsigned int alto();
    void toggleFullScreen();


    SDL_Window* getSDL();

private:
    SDL_Window* crearConFullScreen(unsigned int ancho, unsigned int alto, const std::string& tituloPantalla);
    SDL_Window* crearSinFullScreen(unsigned int ancho, unsigned int alto, const std::string& tituloPantalla);
};

#endif
```

nov 26, 19 17:34 **Textura.h** Page 1/1

```
1  #ifndef _TEXTURA_H_
2  #define _TEXTURA_H_
3
4  #include <string>
5
6  #include "includes/cliente/GUI/Area.h"
7
8  // Forward Declarations:
9  class SDL_Texture;
10
11 class Renderizador;
12
13 class Textura {
14  protected:
15   SDL_Texture *texturaSDL_;
16  private:
17   Textura(const Textura &) = delete;
18   Textura &operator=(const Textura &) = delete;
19
20  public:
21   Textura(const std::string &rutaArchivo, Renderizador &renderizador);
22   Textura(Renderizador &renderizador, Area dimensiones);
23   Textura(Textura ∧otraTextura);
24   Textura &operator=(Textura ∧otraTextura);
25   ~Textura();
26
27   SDL_Texture *getSDL();
28 };
29
30 #endif
```

nov 26, 19 17:34 **Texto.h** Page 1/1

```
1  #ifndef _TEXTO_H_
2  #define _TEXTO_H_
3
4  #include <SDL2/SDL.h>
5  #include <SDL2/SDL_ttf.h>
6  #include "string"
7
8  #define UUID_TEXTO_BLANCO 1
9  #define UUID_TEXTO_NEGRO 2
10 #define UUID_TEXTO_ROJO 3
11 #define UUID_TEXTO_AMARILLO 4
12 #define UUID_TEXTO_VERDE 5
13
14 class Renderizador;
15
16 class Texto {
17  private:
18
19   TTF_Font *font;
20   SDL_Texture *texturaSDL_;
21
22   SDL_Color getColorRGB(int uuidColor);
23   SDL_Texture *createFromText(const std::string texto, Renderizador &renderizado
   r,int uuidColor);
24   Texto(const Texto &) = delete;
25   Texto &operator=(const Texto &) = delete;
26
27  public:
28   Texto(const std::string texto, const int size, Renderizador &renderizador,int
   uuidColor);
29   SDL_Texture *getSDL();
30   void setColor(int uuidColor);
31
32   Texto(Texto ∧other);
33   Texto &operator=(Texto ∧other);
34
35   ~Texto();
36 };
37 #endif
```

nov 26, 19 17:34 **Sonido.h** Page 1/1

```
#ifndef _SONIDO_H_
#define _SONIDO_H_

#include <SDL2/SDL.h>
#include <SDL2/SDL_mixer.h>

#include <string>

class Sonido {
 private:
  Mix_Chunk* efectoSonido;
  bool loop;
 public:
  Sonido(std::string filename,bool loop);
  void play();
  void stop();
  void setVolume(int volume);
  ~Sonido();
};
#endif
```

nov 26, 19 17:34 **Renderizador.h** Page 1/1

```
#ifndef _RENDERIZADOR_H_
#define _RENDERIZADOR_H_

#include "includes/cliente/GUI/Textura.h"
#include "includes/cliente/GUI/Texto.h"
#include "includes/cliente/utils/DobleBuffer.h"
#include <vector>

//Forward declarations:
class SDL_Renderer;
class Escena;
class Ventana;

#define SDL_PRIMER_DISPONIBLE -1

class Renderizador {
 private:
  SDL_Renderer *renderizadorSDL_;
  Ventana &ventana_;

  Renderizador(const Renderizador &) = delete;
  Renderizador &operator=(const Renderizador &) = delete;

 public:
  Renderizador(Ventana &ventana);
  ~Renderizador();
  void clear();
  void dibujar(uint32_t numeroIteracion, Escena &escena);
  void dibujar(uint32_t numeroIteracion, Escena &escena, DobleBuffer<std::vector
<char>>& buffer);
  void setDestino(Textura &textura);
  void resetDestino();
  void dibujar(Textura &textura, Area &destino);
  void dibujarTexto(Texto &texto, Area &destino);
  void dibujar(Textura &textura, Area &destino, double grados, bool flipVertical
);
  void toggleFullScreen();

  SDL_Renderer *getSDL();
};

#endif
```

nov 26, 19 17:34 **Pista.h** Page 1/1

```
#ifndef _PISTA_H_
#define _PISTA_H_
#include "includes/3rd-party/jsoncpp/json.hpp"
#include "includes/cliente/GUI/Animacion.h"
#include "includes/cliente/GUI/AnimacionFactory.h"
#include <map>
#include <memory>
#include <vector>
#include <mutex>
#include "includes/cliente/GUI/ObjetoDinamico.h"

class Pista {
 private:
  Renderizador &renderizador;
  uint16_t capas, size_x, size_y;
  std::map<int, std::vector<std::vector<std::shared_ptr<Animacion>>>> mapa;
  std::map<int, std::shared_ptr<Animacion>> texturas;
  std::map<int, std::shared_ptr<ObjetoDinamico>> objetosDinamicos;
  std::map<int, std::shared_ptr<ObjetoDinamico>> eventosTemporales;
  int idEventoTemporal;

  void agregarBloque(int capa, int x, int y, std::shared_ptr<Animacion> animacio
n);
  void crearPista(nlohmann::json pistaJson);

 public:
  Pista(std::string fileName, Renderizador &renderizador);
  std::shared_ptr<Animacion> getBloque(int capa, int x, int y) const;
  void agregarObjeto(int id, std::shared_ptr<ObjetoDinamico>);
  void agregarEventoTemporal(std::shared_ptr<ObjetoDinamico>);
  std::shared_ptr<ObjetoDinamico> obtenerObjeto(int id);
  std::shared_ptr<ObjetoDinamico> obtenerEventoTemporal(int id);
  void obtenerIds(std::vector<int> &ids);
  void obtenerIdsEventosTemporales(std::vector<int> &ids);
  void borrarObjeto(int id);
  void borrarEventoTemporal(int id);
  int getCapas() const;
  int getSizeX() const;
  int getSizeY() const;
};

#endif

```

nov 26, 19 17:34 **ObjetoDinamico.h** Page 1/1

```
#ifndef _OBJETO_DINAMICO_H_
#define _OBJETO_DINAMICO_H_
#include "includes/cliente/GUI/AnimacionFactory.h"
#include "includes/cliente/GUI/Animacion.h"
#include "includes/cliente/GUI/Renderizador.h"
#include "includes/cliente/GUI/Sonido.h"

class ObjetoDinamico {
 private:
  uint16_t x, y, angulo, vida;
  Animacion animacion_;
  Sonido sonido;

 public:
  ObjetoDinamico(int uuid,
                 Renderizador &renderizador,
                 std::string sonido, bool loopSonido);
  Animacion &getAnimacion();
  void mover(uint16_t x, uint16_t y, uint16_t angulo);
  void setVida(uint16_t vida);
  uint16_t getX() const;
  uint16_t getY() const;
  uint16_t getAngulo() const;
  uint16_t getVida() const;
  Sonido &getSonido();
};
#endif
```

nov 26, 19 17:34 **HiloDibujador.h** Page 1/1

```
#ifndef _HILO_DIBUJADOR_H_
#define _HILO_DIBUJADOR_H_

#include "includes/common/Hilo.h"

#include <stack>
#include <memory>
#include <includes/common/ColaBloqueante.h>

#include "includes/common/eventos/Evento.h"
#include "includes/common/ColaProtegida.h"
#include "includes/cliente/GUI/Sonido.h"
#include "includes/cliente/GUI/Ventana.h"
#include "includes/cliente/GUI/Renderizador.h"
#include "includes/cliente/GUI/eventos/EventoGUI.h"
#include "includes/cliente/grabador/HiloGrabador.h"

class HiloDibujador : public Hilo {
 private:
  Ventana &ventana_;
  Renderizador &renderizador_;
  HiloGrabador &grabador_;
  ColaProtegida<std::shared_ptr<Evento>> eventos_;
  ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI_;
  ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_;
  //TODO: Abstraer en "Escenario" ?
  std::stack<std::shared_ptr<Escena>> escenas_;
  Sonido musicaAmbiente;
  bool &seguirCorriendoCliente;
  void step(uint32_t nroIteracion, Escena &escena);
 public:
  HiloDibujador(Ventana &ventana,
                Renderizador &renderizador,
                HiloGrabador &grabador,
                ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
                ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
                bool &seguirCorriendo);
  virtual void correr() override;
  virtual void detener() override;

  ColaProtegida<std::shared_ptr<Evento>> &eventosEntrantes();
};

#endif
```

nov 26, 19 17:34 **EventoGUIKeyUp.h** Page 1/1

```
#ifndef _EVENTO_GUI_KEY_UP_H_
#define _EVENTO_GUI_KEY_UP_H_

#include <string>

#include "includes/cliente/GUI/eventos/EventoGUI.h"

#define TECLA_C "c"
#define TECLA_A "a"
#define TECLA_Z "z"
#define TECLA_IZQ "Izq"
#define TECLA_DER "Der"

class EventoGUIKeyUp: public EventoGUI {
 private:
  std::string tecla_;

 public:
  EventoGUIKeyUp(const std::string& tecla);
  virtual void actualizar(EventoGUIHandler& handler) override;
  std::string& getTecla();
};

#endif
```

nov 26, 19 17:34 **EventoGUIKeyDown.h** Page 1/1

```
#ifndef _EVENTO_GUI_KEY_DOWN_H_
#define _EVENTO_GUI_KEY_DOWN_H_

#include <string>

#include "includes/cliente/GUI/eventos/EventoGUI.h"

#define TECLA_ESC "Esc"
#define TECLA_C "c"
#define TECLA_A "a"
#define TECLA_Z "z"
#define TECLA_IZQ "Izq"
#define TECLA_DER "Der"
#define TECLA_FULLSCREEN "F11"

class EventoGUIKeyDown : public EventoGUI {
private:
    std::string tecla_;

public:
    EventoGUIKeyDown(const std::string& tecla);
    virtual void actualizar(EventoGUIHandler& handler) override;
    std::string& getTecla();
};

#endif
```

nov 26, 19 17:34 **EventoGUI.h** Page 1/1

```
#ifndef _EVENTO_GUI_H_
#define _EVENTO_GUI_H_

//Forward declarations
class EventoGUIHandler;

class EventoGUI {

public:
    virtual ~EventoGUI() {}
    virtual void actualizar(EventoGUIHandler& handler) = 0;

};

#endif
```

nov 26, 19 17:34 **EventoGUIClick.h** Page 1/1

```
#ifndef _EVENTO_GUI_CLICK_H_
#define _EVENTO_GUI_CLICK_H_

#include "includes/cliente/GUI/eventos/EventoGUI.h"

class EventoGUIClick : public EventoGUI {
private:
    unsigned int x_;
    unsigned int y_;

public:
    EventoGUIClick(unsigned int x, unsigned int y);
    virtual void actualizar(EventoGUIHandler& handler) override;
};

#endif
```

nov 26, 19 17:34 **EventoGUIHandler.h** Page 1/1

```
#ifndef _EVENTO_GUI_HANDLER_H_
#define _EVENTO_GUI_HANDLER_H_

#include "includes/cliente/GUI/eventos/EventoGUI.h"
#include "includes/cliente/GUI/eventos/EventoGUIClick.h"
#include "includes/cliente/GUI/eventos/EventoGUIKeyDown.h"
#include "includes/cliente/GUI/eventos/EventoGUIKeyUp.h"

class EventoGUIHandler {

public:
    virtual void manejarInput(EventoGUI& evento) = 0;
    virtual void manejarInput(EventoGUIClick& evento) = 0;
    virtual void manejarInput(EventoGUIKeyDown& evento) = 0;
    virtual void manejarInput(EventoGUIKeyUp& evento) = 0;
};

#endif
```

nov 26, 19 17:34 **EscenaSala.h** Page 1/1

```
#ifndef _ESCENA_SALA_H_
#define _ESCENA_SALA_H_

#include "includes/cliente/GUI/escenas/Escena.h"

#include <memory>

#include "includes/common/ColaProtegida.h"
#include "includes/cliente/GUI/Animacion.h"
#include "includes/cliente/GUI/Area.h"
#include "includes/cliente/GUI/Renderizador.h"
#include "includes/cliente/GUI/Textura.h"
#include "includes/cliente/GUI/Boton.h"

class EscenaSala : public Escena {
 private:
  Animacion fondoMenu_;
  //TODO: Mover a Escena
  ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI_;
  std::map<int, std::shared_ptr<Boton>> botones;
  std::map<int, std::shared_ptr<Texto>> textoPartidas;
  std::map<uint16_t, uint16_t> partidasId;
  int partidaSeleccionada;
  uint16_t inicioVentana_;
  uint16_t finVentana_;

  void inicializarBotones();
  void inicializarTextoPartidas();
  void dibujarBotones(int iteracion);
  void dibujarTextoPartidas(int iteracion);
  void handlerBotones(int uuid);

 public:
  EscenaSala(Renderizador &renderizador,
             ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
             std::stack<std::shared_ptr<Escena>> &escenas,
             ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
             Sonido &musicaAmbiente,
             EventoSnapshotSala& e);
  virtual Textura dibujate(uint32_t numeroIteracion, Area dimensiones) override;
  virtual void manejarInput(EventoGUI &evento) override;
  virtual void manejarInput(EventoGUIClick &evento) override;
  virtual void manejarInput(EventoGUIKeyDown &evento) override;
  virtual void manejarInput(EventoGUIKeyUp &evento) override;

  virtual void manejar(Evento &e) override;
  virtual void manejar(EventoSnapshotSala &e) override;
  virtual void manejar(EventoPartidaCreada& e) override;
  virtual void manejar(EventoSnapshotLobby& e) override;
};

#endif
```

nov 26, 19 17:34 **EscenaPodio.h** Page 1/1

```
#ifndef _ESCENA_PODIO_H_
#define _ESCENA_PODIO_H_

#include <includes/cliente/GUI/Animacion.h>
#include <includes/common/ColaProtegida.h>
#include <includes/cliente/GUI/ObjetoDinamico.h>
#include "includes/cliente/GUI/escenas/Escena.h"

typedef struct {
  double x_;
  double y_;
} area_t;

class EscenaPodio : public Escena {
 private:
  Animacion fondoMenu_;
  ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI_;
  std::map<int, std::shared_ptr<ObjetoDinamico>> mapaAutos;
  std::map<int,area_t> areasPodio;

  void dibujarAutos(int nroIteracion);

 public:
  EscenaPodio(Renderizador &renderizador,
              ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
              std::stack<std::shared_ptr<Escena>> &escenas,
              ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
              Sonido &musicaAmbiente,
              std::map<int, std::shared_ptr<ObjetoDinamico>> &mapaAutos);
  virtual Textura dibujate(uint32_t numeroIteracion, Area dimensiones) override;
  virtual void manejarInput(EventoGUI &evento) override;
  virtual void manejarInput(EventoGUIClick &evento) override;
  virtual void manejarInput(EventoGUIKeyDown &evento) override;
  virtual void manejarInput(EventoGUIKeyUp &evento) override;

  virtual void manejar(Evento &e) override;
};
#endif
```

nov 26, 19 17:34 **EscenaPartida.h** Page 1/1

```
#ifndef _ESCENA_PARTIDA_H_
#define _ESCENA_PARTIDA_H_

#include "includes/cliente/GUI/escenas/Escena.h"

#include <memory>

#include "includes/common/ColaProtegida.h"
#include "includes/common/Conversor.h"
#include "includes/cliente/GUI/Animacion.h"
#include "includes/cliente/GUI/Area.h"
#include "includes/cliente/GUI/Renderizador.h"
#include "includes/cliente/GUI/Textura.h"
#include "includes/cliente/GUI/Pista.h"
#include "includes/cliente/GUI/Camara.h"
#include "includes/cliente/jugadores/Jugador.h"
#include "includes/cliente/jugadores/Computadora.h"


class EscenaPartida : public Escena {
 private:
  ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI_;
  Pista pista;
  Conversor conversor;
  Camara camara;

  std::shared_ptr<Jugador> jugador_;
  int screenX, screenY;
  ObjetoDinamico barro;

  int id_car;
  bool barroActivo;
  void dibujarInterfaz(int iteracion);
  void dibujarBarro(int iteracion);
 public:
  EscenaPartida(Renderizador &renderizador,
                ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
                std::stack<std::shared_ptr<Escena>> &escenas,
                ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
                EventoPartidaIniciada& estadoInicial,
                Sonido& musicaAmbiente,
                bool juegaComputadora);
  ~EscenaPartida();
  virtual Textura dibujate(uint32_t numeroIteracion, Area dimensiones) override;
  virtual void manejarInput(EventoGUI &evento) override;
  virtual void manejarInput(EventoGUIClick &evento) override;
  virtual void manejarInput(EventoGUIKeyDown &evento) override;
  virtual void manejarInput(EventoGUIKeyUp &evento) override;
  virtual void manejar(Evento &e) override;
  virtual void manejar(EventoSnapshot &e) override;
  virtual void manejar(EventoChoque &e) override ;
  virtual void manejar(EventoFrenada &e) override;
  virtual void manejar(EventoExplosion &e) override ;
  virtual void manejar(EventoBarroPisado &e) override ;
  virtual void manejar(EventoFinBarro &e) override ;
  virtual void manejar(EventoFinCarrera &e) override;
  virtual void manejar(EventoAparecioConsumible& e) override;
  virtual void manejar(EventoDesaparecioConsumible& e) override;
};
#endif
```

nov 26, 19 17:34 **EscenaMenu.h** Page 1/1

```
#ifndef _ESCENA_MENU_H_
#define _ESCENA_MENU_H_

#include "includes/cliente/GUI/escenas/Escena.h"

#include <memory>

#include "includes/common/ColaProtegida.h"
#include "includes/cliente/GUI/Animacion.h"
#include "includes/cliente/GUI/Area.h"
#include "includes/cliente/GUI/Renderizador.h"
#include "includes/cliente/GUI/Textura.h"
#include "includes/cliente/GUI/Sonido.h"
#include "includes/cliente/GUI/Boton.h"

class EscenaMenu : public Escena {
 private:
  Animacion fondoMenu_;
  ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI_;
  std::map<int, std::shared_ptr<Boton>> botones;
  bool quiereEntrarASala;
  bool &seguirCorriendoCliente;

  void inicializarBotones();
  void dibujarBotones(int iteracion);
  void handlerBotones(int uuid);

 public:
  EscenaMenu(Renderizador &renderizador,
             ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
             std::stack<std::shared_ptr<Escena>> &escenas,
             ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
             Sonido &musicaAmbiente, bool &seguirCorriendo);
  virtual Textura dibujate(uint32_t numeroIteracion, Area dimensiones) override;
  virtual void manejarInput(EventoGUI &evento) override;
  virtual void manejarInput(EventoGUIClick &evento) override;
  virtual void manejarInput(EventoGUIKeyDown &evento) override;
  virtual void manejarInput(EventoGUIKeyUp &evento) override;

  virtual void manejar(Evento &e) override;
  virtual void manejar(EventoSnapshotSala &e) override;
};

#endif
```

nov 26, 19 17:34 **EscenaLobby.h** Page 1/1

```
#ifndef _ESCENA_LOBBY_H_
#define _ESCENA_LOBBY_H_

#include <includes/common/ColaProtegida.h>
#include "includes/cliente/GUI/escenas/Escena.h"
#include "includes/cliente/GUI/Boton.h"

class EscenaLobby : public Escena {
 private:
  Animacion fondoMenu_;
  ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI_;
  std::map<int, std::shared_ptr<Boton>> botones;
  std::map<int, std::shared_ptr<Texto>> textoJugadores;
  std::map<int,uint32_t> jugadoresId;
  std::map<int,bool> jugadoresEstaListo;
  bool cpu;

  void inicializarBotones();
  void inicializarTextoJugadores();
  void dibujarBotones(int iteracion);
  void handlerBotones(int uuid);
  void dibujarTextoJugadores(int iteracion);
 public:
  EscenaLobby(Renderizador &renderizador,
              ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
              std::stack<std::shared_ptr<Escena>> &escenas,
              ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
              Sonido &musicaAmbiente,
              EventoPartidaCreada& e);

  EscenaLobby(Renderizador &renderizador,
              ColaProtegida<std::shared_ptr<EventoGUI>> &eventosGUI,
              std::stack<std::shared_ptr<Escena>> &escenas,
              ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
              Sonido &musicaAmbiente,
              EventoSnapshotLobby& e);
  virtual Textura dibujate(uint32_t numeroIteracion, Area dimensiones) override;
  virtual void manejarInput(EventoGUI &evento) override;
  virtual void manejarInput(EventoGUIClick &evento) override;
  virtual void manejarInput(EventoGUIKeyDown &evento) override;
  virtual void manejarInput(EventoGUIKeyUp &evento) override;
  virtual void manejar(Evento &e) override;
  virtual void manejar(EventoPartidaIniciada &estadoInicial) override;
  virtual void manejar(EventoSnapshotLobby &e) override;
};
#endif
```

nov 26, 19 17:34 **Escena.h** Page 1/1

```
#ifndef _ESCENA_H_
#define _ESCENA_H_

#include <stack>
#include <memory>
#include <includes/common/ColaBloqueante.h>
#include <includes/cliente/GUI/Sonido.h>

#include "includes/cliente/GUI/EventoGUIHandler.h"

#include "includes/cliente/GUI/Renderizador.h"
#include "includes/cliente/GUI/Textura.h"

#include "includes/common/Handler.h"

class Escena : public EventoGUIHandler, public Handler {
 protected:
  std::stack<std::shared_ptr<Escena>> &escenas_;
  Renderizador &renderizador_;
  ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_;
  Sonido &musicaAmbiente;

 public:
  Escena(std::stack<std::shared_ptr<Escena>> &escenas,
         Renderizador &renderizador,
         ColaBloqueante<std::shared_ptr<Evento>> &eventosAEnviar_,
         Sonido &musicaAmbiente);
  virtual Textura dibujate(uint32_t numeroIteracion, Area dimensiones) = 0;
  virtual ~Escena();
};

#endif
```

nov 26, 19 17:34 **Camara.h** Page 1/1

```
1  #ifndef _CAMARA_H_
2  #define _CAMARA_H_
3  #include <memory>
4  #include <vector>
5  #include "includes/common/Conversor.h"
6  #include "includes/cliente/GUI/ObjetoDinamico.h"
7  #include "includes/cliente/GUI/Pista.h"
8
9  class Camara {
10  private:
11   Conversor &conversor;
12   Pista &pista;
13   Renderizador& renderizador_;
14   int width, height, xInicial, xFinal, yInicial, yFinal;
15   std::shared_ptr<ObjetoDinamico> car;
16
17  public:
18   Camara(Conversor &conversor, Pista &pista,Renderizador& renderizador);
19   void setWidthHeight(int width, int height);
20   void setCar(std::shared_ptr<ObjetoDinamico> car);
21   void dibujarPista(int iteracion);
22   void dibujarObjetos(int car_id, int iteracion);
23   void dibujarEventosTemporales(int iteracion);
24 };
25
26 #endif
```

nov 26, 19 17:34 **Boton.h** Page 1/1

```
1  #ifndef _BOTON_H_
2  #define _BOTON_H_
3
4  #include "includes/cliente/GUI/Animacion.h"
5  class Boton {
6   private:
7    uint16_t x, y;
8    Animacion animacion_;
9   public:
10   Boton(int uuid, Renderizador &renderizador, uint16_t x, uint16_t y);
11   Animacion &getAnimacion();
12   bool estaSeleccionado(uint16_t x, uint16_t y);
13   uint16_t getX() const;
14   uint16_t getY() const;
15 };
16 #endif
```

nov 26, 19 17:34 **Area.h** Page 1/1

```
1  #ifndef _AREA_H_
2  #define _AREA_H_
3
4  class Area {
5  private:
6      unsigned int x_;
7      unsigned int y_;
8      unsigned int ancho_;
9      unsigned int alto_;
10 public:
11     Area(unsigned int x, unsigned int y, unsigned int ancho, unsigned int alto);
12
13     unsigned int ancho();
14     unsigned int alto();
15
16     unsigned int x();
17     unsigned int y();
18 };
19
20 #endif
```

nov 26, 19 17:34 **Animacion.h** Page 1/1

```
1  #ifndef _ANIMACION_H_
2  #define _ANIMACION_H_
3
4  #include <vector>
5  #include <map>
6
7  #include "includes/cliente/GUI/Textura.h"
8
9  class Animacion {
10 private:
11     unsigned int ancho_;
12     unsigned int alto_;
13     bool primerIteracion;
14     std::vector<int> iterations_;
15     std::vector<Textura> frames_;
16
17     void loadFramesByIteration();
18
19 public:
20     Animacion(std::vector<Textura>& frames, unsigned int ancho, unsigned int alt
   o);
21     Textura& get(uint32_t numeroIteracion);
22     bool terminoPrimerIteracion() const;
23
24     unsigned int ancho();
25     unsigned int alto();
26 };
27
28 #endif
```

nov 26, 19 17:34 **AnimacionFactory.h** Page 1/1

```
#ifndef _ANIMACION_FACTORY_H_
#define _ANIMACION_FACTORY_H_

#include <vector>

#include "includes/cliente/GUI/Textura.h"
#include "includes/cliente/GUI/Renderizador.h"
#include "includes/cliente/GUI/Animacion.h"

#define UUID_BOTON_CREAR_PARTIDA 300
#define UUID_BOTON_UNIRSE_A_PARTIDA 310
#define UUID_BOTON_SALIR 320
#define UUID_BOTON_JUGAR 330
#define UUID_BOTON_ATRAS 340
#define UUID_BOTON_INICIAR_PARTIDA 350
#define UUID_BOTON_LISTO 360
#define UUID_BOTON_MENU 370
#define UUID_BOTON_VACIO 380
#define UUID_BOTON_UP 390
#define UUID_BOTON_DOWN 391
#define UUID_BOTON_CIRCULAR 392

#define UUID_ANIMACION_SALUD 400
#define UUID_ANIMACION_FONDO_MENU 700
#define UUID_ANIMACION_FONDO_SALA 710
#define UUID_ANIMACION_FONDO_PODIO 720
#define UUID_ANIMACION_AUTO_ROJO 800
#define UUID_ANIMACION_AUTO_AMARILLO 810
#define UUID_ANIMACION_AUTO_NEGRO 820
#define UUID_ANIMACION_AUTO_AZUL 830
#define UUID_ANIMACION_AUTO_VERDE 840
#define UUID_ANIMACION_AUTO_OTRO 850
#define UUID_ANIMACION_EXPLOSION 900
#define UUID_ANIMACION_CAJAS_SALUD 1000
#define UUID_ANIMACION_BOOST 1100
#define UUID_ANIMACION_PIEDRA 1200
#define UUID_ANIMACION_ACEITE 1300
#define UUID_ANIMACION_BARRO 1400
#define UUID_ANIMACION_BARRO_GRANDE 1500
#define UUID_ANIMACION_VACIA 9999

#define UUID_ANIMACION_PASTO 106

class AnimacionFactory {

public:
    static Animacion instanciar(unsigned int uuidAnimacion, Renderizador& render
izador);
};

#endif
```

nov 26, 19 17:34 **HiloGrabador.h** Page 1/1

```
#ifndef __HILO_GRABADOR_H__
#define __HILO_GRABADOR_H__


#include "includes/common/Hilo.h"
#include "includes/cliente/utils/DobleBuffer.h"
#include <vector>


class HiloGrabador : public Hilo {
public:
    virtual void correr() override;

    virtual void detener() override;

    DobleBuffer<std::vector<char>>& getBuffer();

private:
  DobleBuffer<std::vector<char>> lineas_rgb_;
};

#endif
```

nov 26, 19 17:34 **video_codec.h** Page 1/1

```
#ifndef _VIDEO_CODEC_H_
#define _VIDEO_CODEC_H_

#include "includes/cliente/grabador/ffmpeg/codec.h"

extern "C" {
  #include <libswscale/swscale.h>
}

class VideoCodec : public Codec {
private:
    struct SwsContext *sws_ctx = NULL;
    VideoCodec(const VideoCodec&) = delete;
  VideoCodec& operator=(const VideoCodec&) = delete;
    VideoCodec& operator=(VideoCodec∧ rhs) = delete;

public:
  VideoCodec(enum AVCodecID id, AVRational avr, Frame& fr, int width, int height
, AVPixelFormat pix_fmt, bool header_flag);

  void copy_parameters(AVStream * st);

  void write_rgb_frame(Frame& dest, const char * data, int pts);

  VideoCodec(VideoCodec∧ rhs);

  ~VideoCodec();
};

#endif
```

nov 26, 19 17:34 **output_video.h** Page 1/1

```
#ifndef _OUTPUT_VIDEO_H_
#define _OUTPUT_VIDEO_H_

#include "includes/cliente/grabador/ffmpeg/output_stream.h"
#include "includes/cliente/grabador/ffmpeg/video_codec.h"

class OutputVideo : public OutputStream {
public:
  OutputVideo(OutputFormat &fmt, AVRational avr, int w, int h, AVPixelFormat pix
);

    void rgb_line_to_frame(const char * v);

    ~OutputVideo();
};

#endif
```

nov 26, 19 17:34 **output_stream.h** Page 1/1

```
#ifndef _OUTPUT_STREAM_H_
#define _OUTPUT_STREAM_H_

extern "C" {
  #include <libavutil/avassert.h>
  #include <libavformat/avformat.h>
  #include <libavutil/opt.h>
  #include <libavutil/mathematics.h>
  #include <libavutil/timestamp.h>
}

#include "includes/cliente/grabador/ffmpeg/frame.h"
#include "includes/cliente/grabador/ffmpeg/codec.h"
#include "includes/cliente/grabador/ffmpeg/output_format.h"

#define STREAM_PIX_FMT    AV_PIX_FMT_YUV420P /* default pix_fmt */
#define SCALE_FLAGS SWS_BICUBIC

class OutputStream {
protected:
    AVStream* st;
    Codec* enc;
    AVPacket* pkt;
    /* pts of the next frame that will be generated */
    int64_t current_pts = 0;
    OutputFormat& fmt;
    Frame frame;

public:
  OutputStream(OutputFormat &fmt);

    virtual void write_frame();

  virtual ~OutputStream();
};

#endif
```

nov 26, 19 17:34 **output_format.h** Page 1/1

```
#ifndef _OUTPUT_FORMAT_H_
#define _OUTPUT_FORMAT_H_

extern "C" {
  #include <libavutil/avassert.h>
  #include <libavformat/avformat.h>
}

#include <string>

class OutputFormat {
private:
    AVFormatContext* ctx;
    std::string filename;

public:
  OutputFormat(const std::string& file);

  AVStream* get_new_stream();

  bool is_flag_set(int flag);

    int write_pkt(AVPacket * pkt);

    void open();

    enum AVCodecID get_video_codec_id();

    enum AVCodecID get_audio_codec_id();

    void write_trailer();

    ~OutputFormat();
};

#endif
```

nov 26, 19 17:34 **frame.h** Page 1/1

```
#ifndef _FRAME_H_
#define _FRAME_H_

extern "C" {
  #include <libavutil/avassert.h>
  #include <libavutil/channel_layout.h>
  #include <libavutil/opt.h>
  #include <libavutil/mathematics.h>
  #include <libavutil/timestamp.h>
  #include <libavformat/avformat.h>
  #include <libswscale/swscale.h>
  #include <libswresample/swresample.h>
}

class Frame {
private:
    AVFrame* fr;

    Frame(const Frame&) = delete;
  Frame& operator=(const Frame&) = delete;

public:
    Frame();

    void VideoFrame(enum AVPixelFormat pix_fmt, int width, int height);

    void make_writable();

    const AVFrame* get_frame() const;

    void fill_rgb(SwsContext* ctx, const char* data, int width, int pts);

    void AudioFrame(enum AVSampleFormat sample_fmt, uint64_t channel_layout, int
 sample_rate, int nb_samples);

    Frame(Frame∧ rhs);


  Frame& operator=(Frame∧ rhs);


  ~Frame();
};

#endif
```

nov 26, 19 17:34 **codec.h** Page 1/1

```
#ifndef _CODEC_H_
#define _CODEC_H_

extern "C" {
  #include <libavformat/avformat.h>
  #include <libavutil/opt.h>
  #include <libavutil/mathematics.h>
  #include <libavutil/timestamp.h>
  #include <libavcodec/avcodec.h>
}

#include "includes/cliente/grabador/ffmpeg/frame.h"

class Codec {
private:
    Codec(const Codec& rhs) = delete;
  Codec& operator=(const Codec& rhs) = delete;

protected:
    AVCodecContext *enc;
  AVCodec* codec;

public:
  Codec(enum AVCodecID id);

  void open();

  void copy_parameters(AVStream * st);

  void encode_frame(const Frame& f);

  int get_packet(AVPacket * pkt, AVRational *time_base);

  Codec(Codec∧ rhs);

  Codec& operator=(Codec∧ rhs);

  virtual ~Codec();
};

#endif
```

nov 26, 19 17:34 **SDLException.h** Page 1/1

```
1  #ifndef _SDL_EXCEPTION_H_
2  #define _SDL_EXCEPTION_H_
3
4  #include <string>
5  #include <exception>
6
7  class SDLException : public std::exception {
8  private:
9      std::string descripcion_;
10
11 public:
12     SDLException(const char* descripcion, const char* errorSDL);
13     const char* what() const noexcept;
14 };
15
16 #endif
```

nov 26, 19 17:34 **Cliente.h** Page 1/1

```
1  #ifndef _CLIENTE_H_
2  #define _CLIENTE_H_
3
4  #include <string>
5  #include <SDL2/SDL.h>
6
7  #include "includes/common/RecibidorEventos.h"
8  #include "includes/common/EnviadorEventos.h"
9  #include "includes/common/eventos/Evento.h"
10
11 #include "includes/cliente/GUI/Ventana.h"
12 #include "includes/cliente/GUI/Renderizador.h"
13 #include "includes/cliente/GUI/eventos/EventoGUI.h"
14 #include "includes/cliente/GUI/HiloDibujador.h"
15 #include "includes/cliente/red/SocketTCPCliente.h"
16 #include "includes/cliente/grabador/HiloGrabador.h"
17
18 class Cliente {
19  private:
20   ColaProtegida<std::shared_ptr<EventoGUI>> eventosGUI_;
21   bool seguirCorriendo;
22   Ventana ventana_;
23   Renderizador renderizador_;
24   HiloGrabador grabador_;
25   HiloDibujador dibujador_;
26   SocketTCPCliente socket_;
27   RecibidorEventos recibidor_;
28   ColaBloqueante<std::shared_ptr<Evento>> eventosAEnviar_;
29   EnviadorEventos enviador_;
30
31   void manejarKeyUp(SDL_Event &eventoSDL);
32   void manejarKeyDown(SDL_Event &eventoSDL);
33   void manejarMouseDown(SDL_Event &eventoSDL);
34
35  public:
36   Cliente(unsigned int anchoVentana,
37           unsigned int altoVentana,
38           bool pantallaCompleta,
39           const std::string &tituloVentana,
40           const std::string &host,
41           const std::string &puerto);
42   void correr();
43   void cerrar();
44   ~Cliente();
45 };
46
47 #endif
48
```

```
/*
* Copyright (c) 2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_ROPE_H
#define B2_ROPE_H

#include "Box2D/Common/b2Math.h"

class b2Draw;

///
struct b2RopeDef
{
  b2RopeDef()
  {
    vertices = nullptr;
    count = 0;
    masses = nullptr;
    gravity.SetZero();
    damping = 0.1f;
    k2 = 0.9f;
    k3 = 0.1f;
  }

  ///
  b2Vec2* vertices;

  ///
  int32 count;

  ///
  float32* masses;

  ///
  b2Vec2 gravity;

  ///
  float32 damping;

  /// Stretching stiffness
  float32 k2;

  /// Bending stiffness. Values above 0.5 can make the simulation blow up.
  float32 k3;
};

///
class b2Rope
{
public:
  b2Rope();
```



```
  ~b2Rope();

  ///
  void Initialize(const b2RopeDef* def);

  ///
  void Step(float32 timeStep, int32 iterations);

  ///
  int32 GetVertexCount() const
  {
    return m_count;
  }

  ///
  const b2Vec2* GetVertices() const
  {
    return m_ps;
  }

  ///
  void Draw(b2Draw* draw) const;

  ///
  void SetAngle(float32 angle);

private:

  void SolveC2();
  void SolveC3();

  int32 m_count;
  b2Vec2* m_ps;
  b2Vec2* m_p0s;
  b2Vec2* m_vs;

  float32* m_ims;

  float32* m_Ls;
  float32* m_as;

  b2Vec2 m_gravity;
  float32 m_damping;

  float32 m_k2;
  float32 m_k3;
};

#endif
```

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_WHEEL_JOINT_H
#define B2_WHEEL_JOINT_H

#include "Box2D/Dynamics/Joints/b2Joint.h"

/// Wheel joint definition. This requires defining a line of
/// motion using an axis and an anchor point. The definition uses local
/// anchor points and a local axis so that the initial configuration
/// can violate the constraint slightly. The joint translation is zero
/// when the local anchor points coincide in world space. Using local
/// anchors and a local axis helps when saving and loading a game.
struct b2WheelJointDef : public b2JointDef
{
  b2WheelJointDef()
  {
    type = e_wheelJoint;
    localAnchorA.SetZero();
    localAnchorB.SetZero();
    localAxisA.Set(1.0f, 0.0f);
    enableMotor = false;
    maxMotorTorque = 0.0f;
    motorSpeed = 0.0f;
    frequencyHz = 2.0f;
    dampingRatio = 0.7f;
  }

  /// Initialize the bodies, anchors, axis, and reference angle using the world
  /// anchor and world axis.
  void Initialize(b2Body* bodyA, b2Body* bodyB, const b2Vec2& anchor, const b2Vec2& axis);

  /// The local anchor point relative to bodyA's origin.
  b2Vec2 localAnchorA;

  /// The local anchor point relative to bodyB's origin.
  b2Vec2 localAnchorB;

  /// The local translation axis in bodyA.
  b2Vec2 localAxisA;

  /// Enable/disable the joint motor.
  bool enableMotor;

  /// The maximum motor torque, usually in N-m.
  float32 maxMotorTorque;

  /// The desired motor speed in radians per second.
  float32 motorSpeed;
```



```

  /// Suspension frequency, zero indicates no suspension
  float32 frequencyHz;

  /// Suspension damping ratio, one indicates critical damping
  float32 dampingRatio;
};

/// A wheel joint. This joint provides two degrees of freedom: translation
/// along an axis fixed in bodyA and rotation in the plane. In other words, it is a point to
/// line constraint with a rotational motor and a linear spring/damper.
/// This joint is designed for vehicle suspensions.
class b2WheelJoint : public b2Joint
{
public:
  b2Vec2 GetAnchorA() const override;
  b2Vec2 GetAnchorB() const override;

  b2Vec2 GetReactionForce(float32 inv_dt) const override;
  float32 GetReactionTorque(float32 inv_dt) const override;

  /// The local anchor point relative to bodyA's origin.
  const b2Vec2& GetLocalAnchorA() const { return m_localAnchorA; }

  /// The local anchor point relative to bodyB's origin.
  const b2Vec2& GetLocalAnchorB() const  { return m_localAnchorB; }

  /// The local joint axis relative to bodyA.
  const b2Vec2& GetLocalAxisA() const { return m_localXAxisA; }

  /// Get the current joint translation, usually in meters.
  float32 GetJointTranslation() const;

  /// Get the current joint linear speed, usually in meters per second.
  float32 GetJointLinearSpeed() const;

  /// Get the current joint angle in radians.
  float32 GetJointAngle() const;

  /// Get the current joint angular speed in radians per second.
  float32 GetJointAngularSpeed() const;

  /// Is the joint motor enabled?
  bool IsMotorEnabled() const;

  /// Enable/disable the joint motor.
  void EnableMotor(bool flag);

  /// Set the motor speed, usually in radians per second.
  void SetMotorSpeed(float32 speed);

  /// Get the motor speed, usually in radians per second.
  float32 GetMotorSpeed() const;

  /// Set/Get the maximum motor force, usually in N-m.
  void SetMaxMotorTorque(float32 torque);
  float32 GetMaxMotorTorque() const;

  /// Get the current motor torque given the inverse time step, usually in N-m.
  float32 GetMotorTorque(float32 inv_dt) const;

  /// Set/Get the spring frequency in hertz. Setting the frequency to zero disables the spring.
  void SetSpringFrequencyHz(float32 hz);
  float32 GetSpringFrequencyHz() const;
```

```
130
131    /// Set/Get the spring damping ratio
132    void SetSpringDampingRatio(float32 ratio);
133    float32 GetSpringDampingRatio() const;
134
135    /// Dump to b2Log
136    void Dump() override;
137
138  protected:
139
140    friend class b2Joint;
141    b2WheelJoint(const b2WheelJointDef* def);
142
143    void InitVelocityConstraints(const b2SolverData& data) override;
144    void SolveVelocityConstraints(const b2SolverData& data) override;
145    bool SolvePositionConstraints(const b2SolverData& data) override;
146
147    float32 m_frequencyHz;
148    float32 m_dampingRatio;
149
150    // Solver shared
151    b2Vec2 m_localAnchorA;
152    b2Vec2 m_localAnchorB;
153    b2Vec2 m_localXAxisA;
154    b2Vec2 m_localYAxisA;
155
156    float32 m_impulse;
157    float32 m_motorImpulse;
158    float32 m_springImpulse;
159
160    float32 m_maxMotorTorque;
161    float32 m_motorSpeed;
162    bool m_enableMotor;
163
164    // Solver temp
165    int32 m_indexA;
166    int32 m_indexB;
167    b2Vec2 m_localCenterA;
168    b2Vec2 m_localCenterB;
169    float32 m_invMassA;
170    float32 m_invMassB;
171    float32 m_invIA;
172    float32 m_invIB;
173
174    b2Vec2 m_ax, m_ay;
175    float32 m_sAx, m_sBx;
176    float32 m_sAy, m_sBy;
177
178    float32 m_mass;
179    float32 m_motorMass;
180    float32 m_springMass;
181
182    float32 m_bias;
183    float32 m_gamma;
184  };
185
186  inline float32 b2WheelJoint::GetMotorSpeed() const
187  {
188    return m_motorSpeed;
189  }
190
191  inline float32 b2WheelJoint::GetMaxMotorTorque() const
192  {
193    return m_maxMotorTorque;
194  }
195
```



```
196  inline void b2WheelJoint::SetSpringFrequencyHz(float32 hz)
197  {
198    m_frequencyHz = hz;
199  }
200
201  inline float32 b2WheelJoint::GetSpringFrequencyHz() const
202  {
203    return m_frequencyHz;
204  }
205
206  inline void b2WheelJoint::SetSpringDampingRatio(float32 ratio)
207  {
208    m_dampingRatio = ratio;
209  }
210
211  inline float32 b2WheelJoint::GetSpringDampingRatio() const
212  {
213    return m_dampingRatio;
214  }
215
216  #endif
```

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_WELD_JOINT_H
#define B2_WELD_JOINT_H

#include "Box2D/Dynamics/Joints/b2Joint.h"

/// Weld joint definition. You need to specify local anchor points
/// where they are attached and the relative body angle. The position
/// of the anchor points is important for computing the reaction torque.
struct b2WeldJointDef : public b2JointDef
{
  b2WeldJointDef()
  {
    type = e_weldJoint;
    localAnchorA.Set(0.0f, 0.0f);
    localAnchorB.Set(0.0f, 0.0f);
    referenceAngle = 0.0f;
    frequencyHz = 0.0f;
    dampingRatio = 0.0f;
  }

  /// Initialize the bodies, anchors, and reference angle using a world
  /// anchor point.
  void Initialize(b2Body* bodyA, b2Body* bodyB, const b2Vec2& anchor);

  /// The local anchor point relative to bodyA's origin.
  b2Vec2 localAnchorA;

  /// The local anchor point relative to bodyB's origin.
  b2Vec2 localAnchorB;

  /// The bodyB angle minus bodyA angle in the reference state (radians).
  float32 referenceAngle;

  /// The mass-spring-damper frequency in Hertz. Rotation only.
  /// Disable softness with a value of 0.
  float32 frequencyHz;

  /// The damping ratio. 0 = no damping, 1 = critical damping.
  float32 dampingRatio;
};

/// A weld joint essentially glues two bodies together. A weld joint may
/// distort somewhat because the island constraint solver is approximate.
class b2WeldJoint : public b2Joint
{
public:
  b2Vec2 GetAnchorA() const override;
  b2Vec2 GetAnchorB() const override;
```



```

  b2Vec2 GetReactionForce(float32 inv_dt) const override;
  float32 GetReactionTorque(float32 inv_dt) const override;

  /// The local anchor point relative to bodyA's origin.
  const b2Vec2& GetLocalAnchorA() const { return m_localAnchorA; }

  /// The local anchor point relative to bodyB's origin.
  const b2Vec2& GetLocalAnchorB() const  { return m_localAnchorB; }

  /// Get the reference angle.
  float32 GetReferenceAngle() const { return m_referenceAngle; }

  /// Set/get frequency in Hz.
  void SetFrequency(float32 hz) { m_frequencyHz = hz; }
  float32 GetFrequency() const { return m_frequencyHz; }

  /// Set/get damping ratio.
  void SetDampingRatio(float32 ratio) { m_dampingRatio = ratio; }
  float32 GetDampingRatio() const { return m_dampingRatio; }

  /// Dump to b2Log
  void Dump() override;

protected:

  friend class b2Joint;

  b2WeldJoint(const b2WeldJointDef* def);

  void InitVelocityConstraints(const b2SolverData& data) override;
  void SolveVelocityConstraints(const b2SolverData& data) override;
  bool SolvePositionConstraints(const b2SolverData& data) override;

  float32 m_frequencyHz;
  float32 m_dampingRatio;
  float32 m_bias;

  // Solver shared
  b2Vec2 m_localAnchorA;
  b2Vec2 m_localAnchorB;
  float32 m_referenceAngle;
  float32 m_gamma;
  b2Vec3 m_impulse;

  // Solver temp
  int32 m_indexA;
  int32 m_indexB;
  b2Vec2 m_rA;
  b2Vec2 m_rB;
  b2Vec2 m_localCenterA;
  b2Vec2 m_localCenterB;
  float32 m_invMassA;
  float32 m_invMassB;
  float32 m_invIA;
  float32 m_invIB;
  b2Mat33 m_mass;
};

#endif
```

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_ROPE_JOINT_H
#define B2_ROPE_JOINT_H

#include "Box2D/Dynamics/Joints/b2Joint.h"

/// Rope joint definition. This requires two body anchor points and
/// a maximum lengths.
/// Note: by default the connected objects will not collide.
/// see collideConnected in b2JointDef.
struct b2RopeJointDef : public b2JointDef
{
  b2RopeJointDef()
  {
    type = e_ropeJoint;
    localAnchorA.Set(-1.0f, 0.0f);
    localAnchorB.Set(1.0f, 0.0f);
    maxLength = 0.0f;
  }

  /// The local anchor point relative to bodyA's origin.
  b2Vec2 localAnchorA;

  /// The local anchor point relative to bodyB's origin.
  b2Vec2 localAnchorB;

  /// The maximum length of the rope.
  /// Warning: this must be larger than b2_linearSlop or
  /// the joint will have no effect.
  float32 maxLength;
};

/// A rope joint enforces a maximum distance between two points
/// on two bodies. It has no other effect.
/// Warning: if you attempt to change the maximum length during
/// the simulation you will get some non-physical behavior.
/// A model that would allow you to dynamically modify the length
/// would have some sponginess, so I chose not to implement it
/// that way. See b2DistanceJoint if you want to dynamically
/// control length.
class b2RopeJoint : public b2Joint
{
public:
  b2Vec2 GetAnchorA() const override;
  b2Vec2 GetAnchorB() const override;

  b2Vec2 GetReactionForce(float32 inv_dt) const override;
  float32 GetReactionTorque(float32 inv_dt) const override;

```



```
  /// The local anchor point relative to bodyA's origin.
  const b2Vec2& GetLocalAnchorA() const { return m_localAnchorA; }

  /// The local anchor point relative to bodyB's origin.
  const b2Vec2& GetLocalAnchorB() const  { return m_localAnchorB; }

  /// Set/Get the maximum length of the rope.
  void SetMaxLength(float32 length) { m_maxLength = length; }
  float32 GetMaxLength() const;

  b2LimitState GetLimitState() const;

  /// Dump joint to dmLog
  void Dump() override;

protected:

  friend class b2Joint;
  b2RopeJoint(const b2RopeJointDef* data);

  void InitVelocityConstraints(const b2SolverData& data) override;
  void SolveVelocityConstraints(const b2SolverData& data) override;
  bool SolvePositionConstraints(const b2SolverData& data) override;

  // Solver shared
  b2Vec2 m_localAnchorA;
  b2Vec2 m_localAnchorB;
  float32 m_maxLength;
  float32 m_length;
  float32 m_impulse;

  // Solver temp
  int32 m_indexA;
  int32 m_indexB;
  b2Vec2 m_u;
  b2Vec2 m_rA;
  b2Vec2 m_rB;
  b2Vec2 m_localCenterA;
  b2Vec2 m_localCenterB;
  float32 m_invMassA;
  float32 m_invMassB;
  float32 m_invIA;
  float32 m_invIB;
  float32 m_mass;
  b2LimitState m_state;
};

#endif
```

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_REVOLUTE_JOINT_H
#define B2_REVOLUTE_JOINT_H

#include "Box2D/Dynamics/Joints/b2Joint.h"

/// Revolute joint definition. This requires defining an
/// anchor point where the bodies are joined. The definition
/// uses local anchor points so that the initial configuration
/// can violate the constraint slightly. You also need to
/// specify the initial relative angle for joint limits. This
/// helps when saving and loading a game.
/// The local anchor points are measured from the body's origin
/// rather than the center of mass because:
/// 1. you might not know where the center of mass will be.
/// 2. if you add/remove shapes from a body and recompute the mass,
///    the joints will be broken.
struct b2RevoluteJointDef : public b2JointDef
{
  b2RevoluteJointDef()
  {
    type = e_revoluteJoint;
    localAnchorA.Set(0.0f, 0.0f);
    localAnchorB.Set(0.0f, 0.0f);
    referenceAngle = 0.0f;
    lowerAngle = 0.0f;
    upperAngle = 0.0f;
    maxMotorTorque = 0.0f;
    motorSpeed = 0.0f;
    enableLimit = false;
    enableMotor = false;
  }

  /// Initialize the bodies, anchors, and reference angle using a world
  /// anchor point.
  void Initialize(b2Body* bodyA, b2Body* bodyB, const b2Vec2& anchor);

  /// The local anchor point relative to bodyA's origin.
  b2Vec2 localAnchorA;

  /// The local anchor point relative to bodyB's origin.
  b2Vec2 localAnchorB;

  /// The bodyB angle minus bodyA angle in the reference state (radians).
  float32 referenceAngle;

  /// A flag to enable joint limits.
  bool enableLimit;

```



```
  /// The lower angle for the joint limit (radians).
  float32 lowerAngle;

  /// The upper angle for the joint limit (radians).
  float32 upperAngle;

  /// A flag to enable the joint motor.
  bool enableMotor;

  /// The desired motor speed. Usually in radians per second.
  float32 motorSpeed;

  /// The maximum motor torque used to achieve the desired motor speed.
  /// Usually in N-m.
  float32 maxMotorTorque;
};

/// A revolute joint constrains two bodies to share a common point while they
/// are free to rotate about the point. The relative rotation about the shared
/// point is the joint angle. You can limit the relative rotation with
/// a joint limit that specifies a lower and upper angle. You can use a motor
/// to drive the relative rotation about the shared point. A maximum motor torqu
e
/// is provided so that infinite forces are not generated.
class b2RevoluteJoint : public b2Joint
{
public:
  b2Vec2 GetAnchorA() const override;
  b2Vec2 GetAnchorB() const override;

  /// The local anchor point relative to bodyA's origin.
  const b2Vec2& GetLocalAnchorA() const { return m_localAnchorA; }

  /// The local anchor point relative to bodyB's origin.
  const b2Vec2& GetLocalAnchorB() const  { return m_localAnchorB; }

  /// Get the reference angle.
  float32 GetReferenceAngle() const { return m_referenceAngle; }

  /// Get the current joint angle in radians.
  float32 GetJointAngle() const;

  /// Get the current joint angle speed in radians per second.
  float32 GetJointSpeed() const;

  /// Is the joint limit enabled?
  bool IsLimitEnabled() const;

  /// Enable/disable the joint limit.
  void EnableLimit(bool flag);

  /// Get the lower joint limit in radians.
  float32 GetLowerLimit() const;

  /// Get the upper joint limit in radians.
  float32 GetUpperLimit() const;

  /// Set the joint limits in radians.
  void SetLimits(float32 lower, float32 upper);

  /// Is the joint motor enabled?
  bool IsMotorEnabled() const;

  /// Enable/disable the joint motor.
  void EnableMotor(bool flag);

```

```
132    /// Set the motor speed in radians per second.
133    void SetMotorSpeed(float32 speed);
134
135    /// Get the motor speed in radians per second.
136    float32 GetMotorSpeed() const;
137
138    /// Set the maximum motor torque, usually in N-m.
139    void SetMaxMotorTorque(float32 torque);
140    float32 GetMaxMotorTorque() const { return m_maxMotorTorque; }
141
142    /// Get the reaction force given the inverse time step.
143    /// Unit is N.
144    b2Vec2 GetReactionForce(float32 inv_dt) const override;
145
146    /// Get the reaction torque due to the joint limit given the inverse time step
     .
147    /// Unit is N*m.
148    float32 GetReactionTorque(float32 inv_dt) const override;
149
150    /// Get the current motor torque given the inverse time step.
151    /// Unit is N*m.
152    float32 GetMotorTorque(float32 inv_dt) const;
153
154    /// Dump to b2Log.
155    void Dump() override;
156
157  protected:
158
159    friend class b2Joint;
160    friend class b2GearJoint;
161
162    b2RevoluteJoint(const b2RevoluteJointDef* def);
163
164    void InitVelocityConstraints(const b2SolverData& data) override;
165    void SolveVelocityConstraints(const b2SolverData& data) override;
166    bool SolvePositionConstraints(const b2SolverData& data) override;
167
168    // Solver shared
169    b2Vec2 m_localAnchorA;
170    b2Vec2 m_localAnchorB;
171    b2Vec3 m_impulse;
172    float32 m_motorImpulse;
173
174    bool m_enableMotor;
175    float32 m_maxMotorTorque;
176    float32 m_motorSpeed;
177
178    bool m_enableLimit;
179    float32 m_referenceAngle;
180    float32 m_lowerAngle;
181    float32 m_upperAngle;
182
183    // Solver temp
184    int32 m_indexA;
185    int32 m_indexB;
186    b2Vec2 m_rA;
187    b2Vec2 m_rB;
188    b2Vec2 m_localCenterA;
189    b2Vec2 m_localCenterB;
190    float32 m_invMassA;
191    float32 m_invMassB;
192    float32 m_invIA;
193    float32 m_invIB;
194    b2Mat33 m_mass;     // effective mass for point-to-point constraint.
195    float32 m_motorMass;  // effective mass for motor/limit angular constraint.
196    b2LimitState m_limitState;
```



```
197  };
198
199  inline float32 b2RevoluteJoint::GetMotorSpeed() const
200  {
201    return m_motorSpeed;
202  }
203
204  #endif
```

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_PULLEY_JOINT_H
#define B2_PULLEY_JOINT_H

#include "Box2D/Dynamics/Joints/b2Joint.h"

const float32 b2_minPulleyLength = 2.0f;

/// Pulley joint definition. This requires two ground anchors,
/// two dynamic body anchor points, and a pulley ratio.
struct b2PulleyJointDef : public b2JointDef
{
  b2PulleyJointDef()
  {
    type = e_pulleyJoint;
    groundAnchorA.Set(-1.0f, 1.0f);
    groundAnchorB.Set(1.0f, 1.0f);
    localAnchorA.Set(-1.0f, 0.0f);
    localAnchorB.Set(1.0f, 0.0f);
    lengthA = 0.0f;
    lengthB = 0.0f;
    ratio = 1.0f;
    collideConnected = true;
  }

  /// Initialize the bodies, anchors, lengths, max lengths, and ratio using the
world anchors.
  void Initialize(b2Body* bodyA, b2Body* bodyB,
          const b2Vec2& groundAnchorA, const b2Vec2& groundAnchorB,
          const b2Vec2& anchorA, const b2Vec2& anchorB,
          float32 ratio);

  /// The first ground anchor in world coordinates. This point never moves.
  b2Vec2 groundAnchorA;

  /// The second ground anchor in world coordinates. This point never moves.
  b2Vec2 groundAnchorB;

  /// The local anchor point relative to bodyA's origin.
  b2Vec2 localAnchorA;

  /// The local anchor point relative to bodyB's origin.
  b2Vec2 localAnchorB;

  /// The a reference length for the segment attached to bodyA.
  float32 lengthA;

  /// The a reference length for the segment attached to bodyB.
  float32 lengthB;
```



```

  /// The pulley ratio, used to simulate a block-and-tackle.
  float32 ratio;
};

/// The pulley joint is connected to two bodies and two fixed ground points.
/// The pulley supports a ratio such that:
/// length1 + ratio * length2 <= constant
/// Yes, the force transmitted is scaled by the ratio.
/// Warning: the pulley joint can get a bit squirrelly by itself. They often
/// work better when combined with prismatic joints. You should also cover the
/// the anchor points with static shapes to prevent one side from going to
/// zero length.
class b2PulleyJoint : public b2Joint
{
public:
  b2Vec2 GetAnchorA() const override;
  b2Vec2 GetAnchorB() const override;

  b2Vec2 GetReactionForce(float32 inv_dt) const override;
  float32 GetReactionTorque(float32 inv_dt) const override;

  /// Get the first ground anchor.
  b2Vec2 GetGroundAnchorA() const;

  /// Get the second ground anchor.
  b2Vec2 GetGroundAnchorB() const;

  /// Get the current length of the segment attached to bodyA.
  float32 GetLengthA() const;

  /// Get the current length of the segment attached to bodyB.
  float32 GetLengthB() const;

  /// Get the pulley ratio.
  float32 GetRatio() const;

  /// Get the current length of the segment attached to bodyA.
  float32 GetCurrentLengthA() const;

  /// Get the current length of the segment attached to bodyB.
  float32 GetCurrentLengthB() const;

  /// Dump joint to dmLog
  void Dump() override;

  /// Implement b2Joint::ShiftOrigin
  void ShiftOrigin(const b2Vec2& newOrigin) override;

protected:

  friend class b2Joint;
  b2PulleyJoint(const b2PulleyJointDef* data);

  void InitVelocityConstraints(const b2SolverData& data) override;
  void SolveVelocityConstraints(const b2SolverData& data) override;
  bool SolvePositionConstraints(const b2SolverData& data) override;

  b2Vec2 m_groundAnchorA;
  b2Vec2 m_groundAnchorB;
  float32 m_lengthA;
  float32 m_lengthB;

  // Solver shared
  b2Vec2 m_localAnchorA;
  b2Vec2 m_localAnchorB;
```

```
132    float32 m_constant;
133    float32 m_ratio;
134    float32 m_impulse;
135
136    // Solver temp
137    int32 m_indexA;
138    int32 m_indexB;
139    b2Vec2 m_uA;
140    b2Vec2 m_uB;
141    b2Vec2 m_rA;
142    b2Vec2 m_rB;
143    b2Vec2 m_localCenterA;
144    b2Vec2 m_localCenterB;
145    float32 m_invMassA;
146    float32 m_invMassB;
147    float32 m_invIA;
148    float32 m_invIB;
149    float32 m_mass;
150  };
151
152  #endif
```



```
1   /*
2   * Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
3   *
4   * This software is provided 'as-is', without any express or implied
5   * warranty.  In no event will the authors be held liable for any damages
6   * arising from the use of this software.
7   * Permission is granted to anyone to use this software for any purpose,
8   * including commercial applications, and to alter it and redistribute it
9   * freely, subject to the following restrictions:
10  * 1. The origin of this software must not be misrepresented; you must not
11  * claim that you wrote the original software. If you use this software
12  * in a product, an acknowledgment in the product documentation would be
13  * appreciated but is not required.
14  * 2. Altered source versions must be plainly marked as such, and must not be
15  * misrepresented as being the original software.
16  * 3. This notice may not be removed or altered from any source distribution.
17  */
18
19  #ifndef B2_PRISMATIC_JOINT_H
20  #define B2_PRISMATIC_JOINT_H
21
22  #include "Box2D/Dynamics/Joints/b2Joint.h"
23
24  /// Prismatic joint definition. This requires defining a line of
25  /// motion using an axis and an anchor point. The definition uses local
26  /// anchor points and a local axis so that the initial configuration
27  /// can violate the constraint slightly. The joint translation is zero
28  /// when the local anchor points coincide in world space. Using local
29  /// anchors and a local axis helps when saving and loading a game.
30  struct b2PrismaticJointDef : public b2JointDef
31  {
32    b2PrismaticJointDef()
33    {
34      type = e_prismaticJoint;
35      localAnchorA.SetZero();
36      localAnchorB.SetZero();
37      localAxisA.Set(1.0f, 0.0f);
38      referenceAngle = 0.0f;
39      enableLimit = false;
40      lowerTranslation = 0.0f;
41      upperTranslation = 0.0f;
42      enableMotor = false;
43      maxMotorForce = 0.0f;
44      motorSpeed = 0.0f;
45    }
46
47    /// Initialize the bodies, anchors, axis, and reference angle using the world
48    /// anchor and unit world axis.
49    void Initialize(b2Body* bodyA, b2Body* bodyB, const b2Vec2& anchor, const b2Ve
      c2& axis);
50
51    /// The local anchor point relative to bodyA's origin.
52    b2Vec2 localAnchorA;
53
54    /// The local anchor point relative to bodyB's origin.
55    b2Vec2 localAnchorB;
56
57    /// The local translation unit axis in bodyA.
58    b2Vec2 localAxisA;
59
60    /// The constrained angle between the bodies: bodyB_angle - bodyA_angle.
61    float32 referenceAngle;
62
63    /// Enable/disable the joint limit.
64    bool enableLimit;
65
```

```
66    /// The lower translation limit, usually in meters.
67    float32 lowerTranslation;
68
69    /// The upper translation limit, usually in meters.
70    float32 upperTranslation;
71
72    /// Enable/disable the joint motor.
73    bool enableMotor;
74
75    /// The maximum motor torque, usually in N-m.
76    float32 maxMotorForce;
77
78    /// The desired motor speed in radians per second.
79    float32 motorSpeed;
80  };
81
82  /// A prismatic joint. This joint provides one degree of freedom: translation
83  /// along an axis fixed in bodyA. Relative rotation is prevented. You can
84  /// use a joint limit to restrict the range of motion and a joint motor to
85  /// drive the motion or to model joint friction.
86  class b2PrismaticJoint : public b2Joint
87  {
88  public:
89    b2Vec2 GetAnchorA() const override;
90    b2Vec2 GetAnchorB() const override;
91
92    b2Vec2 GetReactionForce(float32 inv_dt) const override;
93    float32 GetReactionTorque(float32 inv_dt) const override;
94
95    /// The local anchor point relative to bodyA's origin.
96    const b2Vec2& GetLocalAnchorA() const { return m_localAnchorA; }
97
98    /// The local anchor point relative to bodyB's origin.
99    const b2Vec2& GetLocalAnchorB() const  { return m_localAnchorB; }
100
101   /// The local joint axis relative to bodyA.
102   const b2Vec2& GetLocalAxisA() const { return m_localXAxisA; }
103
104   /// Get the reference angle.
105   float32 GetReferenceAngle() const { return m_referenceAngle; }
106
107   /// Get the current joint translation, usually in meters.
108   float32 GetJointTranslation() const;
109
110   /// Get the current joint translation speed, usually in meters per second.
111   float32 GetJointSpeed() const;
112
113   /// Is the joint limit enabled?
114   bool IsLimitEnabled() const;
115
116   /// Enable/disable the joint limit.
117   void EnableLimit(bool flag);
118
119   /// Get the lower joint limit, usually in meters.
120   float32 GetLowerLimit() const;
121
122   /// Get the upper joint limit, usually in meters.
123   float32 GetUpperLimit() const;
124
125   /// Set the joint limits, usually in meters.
126   void SetLimits(float32 lower, float32 upper);
127
128   /// Is the joint motor enabled?
129   bool IsMotorEnabled() const;
130
131   /// Enable/disable the joint motor.
```



```
132   void EnableMotor(bool flag);
133
134   /// Set the motor speed, usually in meters per second.
135   void SetMotorSpeed(float32 speed);
136
137   /// Get the motor speed, usually in meters per second.
138   float32 GetMotorSpeed() const;
139
140   /// Set the maximum motor force, usually in N.
141   void SetMaxMotorForce(float32 force);
142   float32 GetMaxMotorForce() const { return m_maxMotorForce; }
143
144   /// Get the current motor force given the inverse time step, usually in N.
145   float32 GetMotorForce(float32 inv_dt) const;
146
147   /// Dump to b2Log
148   void Dump() override;
149
150 protected:
151   friend class b2Joint;
152   friend class b2GearJoint;
153   b2PrismaticJoint(const b2PrismaticJointDef* def);
154
155   void InitVelocityConstraints(const b2SolverData& data) override;
156   void SolveVelocityConstraints(const b2SolverData& data) override;
157   bool SolvePositionConstraints(const b2SolverData& data) override;
158
159   // Solver shared
160   b2Vec2 m_localAnchorA;
161   b2Vec2 m_localAnchorB;
162   b2Vec2 m_localXAxisA;
163   b2Vec2 m_localYAxisA;
164   float32 m_referenceAngle;
165   b2Vec3 m_impulse;
166   float32 m_motorImpulse;
167   float32 m_lowerTranslation;
168   float32 m_upperTranslation;
169   float32 m_maxMotorForce;
170   float32 m_motorSpeed;
171   bool m_enableLimit;
172   bool m_enableMotor;
173   b2LimitState m_limitState;
174
175   // Solver temp
176   int32 m_indexA;
177   int32 m_indexB;
178   b2Vec2 m_localCenterA;
179   b2Vec2 m_localCenterB;
180   float32 m_invMassA;
181   float32 m_invMassB;
182   float32 m_invIA;
183   float32 m_invIB;
184   b2Vec2 m_axis, m_perp;
185   float32 m_s1, m_s2;
186   float32 m_a1, m_a2;
187   b2Mat33 m_K;
188   float32 m_motorMass;
189 };
190
191 inline float32 b2PrismaticJoint::GetMotorSpeed() const
192 {
193   return m_motorSpeed;
194 }
195
196 #endif
```

```
/*
* Copyright (c) 2006-2007 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_MOUSE_JOINT_H
#define B2_MOUSE_JOINT_H

#include "Box2D/Dynamics/Joints/b2Joint.h"

/// Mouse joint definition. This requires a world target point,
/// tuning parameters, and the time step.
struct b2MouseJointDef : public b2JointDef
{
  b2MouseJointDef()
  {
    type = e_mouseJoint;
    target.Set(0.0f, 0.0f);
    maxForce = 0.0f;
    frequencyHz = 5.0f;
    dampingRatio = 0.7f;
  }

  /// The initial world target point. This is assumed
  /// to coincide with the body anchor initially.
  b2Vec2 target;

  /// The maximum constraint force that can be exerted
  /// to move the candidate body. Usually you will express
  /// as some multiple of the weight (multiplier * mass * gravity).
  float32 maxForce;

  /// The response speed.
  float32 frequencyHz;

  /// The damping ratio. 0 = no damping, 1 = critical damping.
  float32 dampingRatio;
};

/// A mouse joint is used to make a point on a body track a
/// specified world point. This a soft constraint with a maximum
/// force. This allows the constraint to stretch and without
/// applying huge forces.
/// NOTE: this joint is not documented in the manual because it was
/// developed to be used in the testbed. If you want to learn how to
/// use the mouse joint, look at the testbed.
class b2MouseJoint : public b2Joint
{
public:

  /// Implements b2Joint.
  b2Vec2 GetAnchorA() const override;

```



```
  /// Implements b2Joint.
  b2Vec2 GetAnchorB() const override;

  /// Implements b2Joint.
  b2Vec2 GetReactionForce(float32 inv_dt) const override;

  /// Implements b2Joint.
  float32 GetReactionTorque(float32 inv_dt) const override;

  /// Use this to update the target point.
  void SetTarget(const b2Vec2& target);
  const b2Vec2& GetTarget() const;

  /// Set/get the maximum force in Newtons.
  void SetMaxForce(float32 force);
  float32 GetMaxForce() const;

  /// Set/get the frequency in Hertz.
  void SetFrequency(float32 hz);
  float32 GetFrequency() const;

  /// Set/get the damping ratio (dimensionless).
  void SetDampingRatio(float32 ratio);
  float32 GetDampingRatio() const;

  /// The mouse joint does not support dumping.
  void Dump() override { b2Log("Mouse joint dumping is not supported.\n"); }

  /// Implement b2Joint::ShiftOrigin
  void ShiftOrigin(const b2Vec2& newOrigin) override;

protected:
  friend class b2Joint;

  b2MouseJoint(const b2MouseJointDef* def);

  void InitVelocityConstraints(const b2SolverData& data) override;
  void SolveVelocityConstraints(const b2SolverData& data) override;
  bool SolvePositionConstraints(const b2SolverData& data) override;

  b2Vec2 m_localAnchorB;
  b2Vec2 m_targetA;
  float32 m_frequencyHz;
  float32 m_dampingRatio;
  float32 m_beta;

  // Solver shared
  b2Vec2 m_impulse;
  float32 m_maxForce;
  float32 m_gamma;

  // Solver temp
  int32 m_indexA;
  int32 m_indexB;
  b2Vec2 m_rB;
  b2Vec2 m_localCenterB;
  float32 m_invMassB;
  float32 m_invIB;
  b2Mat22 m_mass;
  b2Vec2 m_C;
};

#endif
```

```
/*
* Copyright (c) 2006-2012 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_MOTOR_JOINT_H
#define B2_MOTOR_JOINT_H

#include "Box2D/Dynamics/Joints/b2Joint.h"

/// Motor joint definition.
struct b2MotorJointDef : public b2JointDef
{
  b2MotorJointDef()
  {
    type = e_motorJoint;
    linearOffset.SetZero();
    angularOffset = 0.0f;
    maxForce = 1.0f;
    maxTorque = 1.0f;
    correctionFactor = 0.3f;
  }

  /// Initialize the bodies and offsets using the current transforms.
  void Initialize(b2Body* bodyA, b2Body* bodyB);

  /// Position of bodyB minus the position of bodyA, in bodyA's frame, in meters
.
  b2Vec2 linearOffset;

  /// The bodyB angle minus bodyA angle in radians.
  float32 angularOffset;

  /// The maximum motor force in N.
  float32 maxForce;

  /// The maximum motor torque in N-m.
  float32 maxTorque;

  /// Position correction factor in the range [0,1].
  float32 correctionFactor;
};

/// A motor joint is used to control the relative motion
/// between two bodies. A typical usage is to control the movement
/// of a dynamic body with respect to the ground.
class b2MotorJoint : public b2Joint
{
public:
  b2Vec2 GetAnchorA() const override;
  b2Vec2 GetAnchorB() const override;

  b2Vec2 GetReactionForce(float32 inv_dt) const override;
```



```
  float32 GetReactionTorque(float32 inv_dt) const override;

  /// Set/get the target linear offset, in frame A, in meters.
  void SetLinearOffset(const b2Vec2& linearOffset);
  const b2Vec2& GetLinearOffset() const;

  /// Set/get the target angular offset, in radians.
  void SetAngularOffset(float32 angularOffset);
  float32 GetAngularOffset() const;

  /// Set the maximum friction force in N.
  void SetMaxForce(float32 force);

  /// Get the maximum friction force in N.
  float32 GetMaxForce() const;

  /// Set the maximum friction torque in N*m.
  void SetMaxTorque(float32 torque);

  /// Get the maximum friction torque in N*m.
  float32 GetMaxTorque() const;

  /// Set the position correction factor in the range [0,1].
  void SetCorrectionFactor(float32 factor);

  /// Get the position correction factor in the range [0,1].
  float32 GetCorrectionFactor() const;

  /// Dump to b2Log
  void Dump() override;

protected:

  friend class b2Joint;

  b2MotorJoint(const b2MotorJointDef* def);

  void InitVelocityConstraints(const b2SolverData& data) override;
  void SolveVelocityConstraints(const b2SolverData& data) override;
  bool SolvePositionConstraints(const b2SolverData& data) override;

  // Solver shared
  b2Vec2 m_linearOffset;
  float32 m_angularOffset;
  b2Vec2 m_linearImpulse;
  float32 m_angularImpulse;
  float32 m_maxForce;
  float32 m_maxTorque;
  float32 m_correctionFactor;

  // Solver temp
  int32 m_indexA;
  int32 m_indexB;
  b2Vec2 m_rA;
  b2Vec2 m_rB;
  b2Vec2 m_localCenterA;
  b2Vec2 m_localCenterB;
  b2Vec2 m_linearError;
  float32 m_angularError;
  float32 m_invMassA;
  float32 m_invMassB;
  float32 m_invIA;
  float32 m_invIB;
  b2Mat22 m_linearMass;
  float32 m_angularMass;
};
```

```
132
133 #endif
```



```
1  /*
2  * Copyright (c) 2006-2007 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as-is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #ifndef B2_JOINT_H
20 #define B2_JOINT_H
21
22 #include "Box2D/Common/b2Math.h"
23
24 class b2Body;
25 class b2Joint;
26 struct b2SolverData;
27 class b2BlockAllocator;
28
29 enum b2JointType
30 {
31   e_unknownJoint,
32   e_revoluteJoint,
33   e_prismaticJoint,
34   e_distanceJoint,
35   e_pulleyJoint,
36   e_mouseJoint,
37   e_gearJoint,
38   e_wheelJoint,
39     e_weldJoint,
40   e_frictionJoint,
41   e_ropeJoint,
42   e_motorJoint
43 };
44
45 enum b2LimitState
46 {
47   e_inactiveLimit,
48   e_atLowerLimit,
49   e_atUpperLimit,
50   e_equalLimits
51 };
52
53 struct b2Jacobian
54 {
55   b2Vec2 linear;
56   float32 angularA;
57   float32 angularB;
58 };
59
60 /// A joint edge is used to connect bodies and joints together
61 /// in a joint graph where each body is a node and each joint
62 /// is an edge. A joint edge belongs to a doubly linked list
63 /// maintained in each attached body. Each joint has two joint
64 /// nodes, one for each attached body.
65 struct b2JointEdge
66 {
```

```
67    b2Body* other;      ///< provides quick access to the other body attached.
68    b2Joint* joint;     ///< the joint
69    b2JointEdge* prev;    ///< the previous joint edge in the body's joint list
70    b2JointEdge* next;    ///< the next joint edge in the body's joint list
71  };
72
73  /// Joint definitions are used to construct joints.
74  struct b2JointDef
75  {
76    b2JointDef()
77    {
78      type = e_unknownJoint;
79      userData = nullptr;
80      bodyA = nullptr;
81      bodyB = nullptr;
82      collideConnected = false;
83    }
84
85    /// The joint type is set automatically for concrete joint types.
86    b2JointType type;
87
88    /// Use this to attach application specific data to your joints.
89    void* userData;
90
91    /// The first attached body.
92    b2Body* bodyA;
93
94    /// The second attached body.
95    b2Body* bodyB;
96
97    /// Set this flag to true if the attached bodies should collide.
98    bool collideConnected;
99  };
100
101 /// The base joint class. Joints are used to constraint two bodies together in
102 /// various fashions. Some joints also feature limits and motors.
103 class b2Joint
104 {
105 public:
106
107   /// Get the type of the concrete joint.
108   b2JointType GetType() const;
109
110   /// Get the first body attached to this joint.
111   b2Body* GetBodyA();
112
113   /// Get the second body attached to this joint.
114   b2Body* GetBodyB();
115
116   /// Get the anchor point on bodyA in world coordinates.
117   virtual b2Vec2 GetAnchorA() const = 0;
118
119   /// Get the anchor point on bodyB in world coordinates.
120   virtual b2Vec2 GetAnchorB() const = 0;
121
122   /// Get the reaction force on bodyB at the joint anchor in Newtons.
123   virtual b2Vec2 GetReactionForce(float32 inv_dt) const = 0;
124
125   /// Get the reaction torque on bodyB in N*m.
126   virtual float32 GetReactionTorque(float32 inv_dt) const = 0;
127
128   /// Get the next joint the world joint list.
129   b2Joint* GetNext();
130   const b2Joint* GetNext() const;
131
132   /// Get the user data pointer.
```



```
133   void* GetUserData() const;
134
135   /// Set the user data pointer.
136   void SetUserData(void* data);
137
138   /// Short-cut function to determine if either body is inactive.
139   bool IsActive() const;
140
141   /// Get collide connected.
142   /// Note: modifying the collide connect flag won't work correctly because
143   /// the flag is only checked when fixture AABBs begin to overlap.
144   bool GetCollideConnected() const;
145
146   /// Dump this joint to the log file.
147   virtual void Dump() { b2Log("// Dump is not supported for this joint type.\n"); }
148
149   /// Shift the origin for any points stored in world coordinates.
150   virtual void ShiftOrigin(const b2Vec2& newOrigin) { B2_NOT_USED(newOrigin);  }
151
152 protected:
153   friend class b2World;
154   friend class b2Body;
155   friend class b2Island;
156   friend class b2GearJoint;
157
158   static b2Joint* Create(const b2JointDef* def, b2BlockAllocator* allocator);
159   static void Destroy(b2Joint* joint, b2BlockAllocator* allocator);
160
161   b2Joint(const b2JointDef* def);
162   virtual ~b2Joint() {}
163
164   virtual void InitVelocityConstraints(const b2SolverData& data) = 0;
165   virtual void SolveVelocityConstraints(const b2SolverData& data) = 0;
166
167   // This returns true if the position errors are within tolerance.
168   virtual bool SolvePositionConstraints(const b2SolverData& data) = 0;
169
170   b2JointType m_type;
171   b2Joint* m_prev;
172   b2Joint* m_next;
173   b2JointEdge m_edgeA;
174   b2JointEdge m_edgeB;
175   b2Body* m_bodyA;
176   b2Body* m_bodyB;
177
178   int32 m_index;
179
180   bool m_islandFlag;
181   bool m_collideConnected;
182
183   void* m_userData;
184 };
185
186 inline b2JointType b2Joint::GetType() const
187 {
188   return m_type;
189 }
190
191 inline b2Body* b2Joint::GetBodyA()
192 {
193   return m_bodyA;
194 }
195
196 inline b2Body* b2Joint::GetBodyB()
197 {
198   return m_bodyB;
```

```
}

inline b2Joint* b2Joint::GetNext()
{
  return m_next;
}

inline const b2Joint* b2Joint::GetNext() const
{
  return m_next;
}

inline void* b2Joint::GetUserData() const
{
  return m_userData;
}

inline void b2Joint::SetUserData(void* data)
{
  m_userData = data;
}

inline bool b2Joint::GetCollideConnected() const
{
  return m_collideConnected;
}

#endif
```



```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_GEAR_JOINT_H
#define B2_GEAR_JOINT_H

#include "Box2D/Dynamics/Joints/b2Joint.h"

/// Gear joint definition. This definition requires two existing
/// revolute or prismatic joints (any combination will work).
struct b2GearJointDef : public b2JointDef
{
  b2GearJointDef()
  {
    type = e_gearJoint;
    joint1 = nullptr;
    joint2 = nullptr;
    ratio = 1.0f;
  }

  /// The first revolute/prismatic joint attached to the gear joint.
  b2Joint* joint1;

  /// The second revolute/prismatic joint attached to the gear joint.
  b2Joint* joint2;

  /// The gear ratio.
  /// @see b2GearJoint for explanation.
  float32 ratio;
};

/// A gear joint is used to connect two joints together. Either joint
/// can be a revolute or prismatic joint. You specify a gear ratio
/// to bind the motions together:
/// coordinate1 + ratio * coordinate2 = constant
/// The ratio can be negative or positive. If one joint is a revolute joint
/// and the other joint is a prismatic joint, then the ratio will have units
/// of length or units of 1/length.
/// @warning You have to manually destroy the gear joint if joint1 or joint2
/// is destroyed.
class b2GearJoint : public b2Joint
{
public:
  b2Vec2 GetAnchorA() const override;
  b2Vec2 GetAnchorB() const override;

  b2Vec2 GetReactionForce(float32 inv_dt) const override;
  float32 GetReactionTorque(float32 inv_dt) const override;

  /// Get the first joint.
  b2Joint* GetJoint1() { return m_joint1; }
```

```
67
68    /// Get the second joint.
69    b2Joint* GetJoint2() { return m_joint2; }
70
71    /// Set/Get the gear ratio.
72    void SetRatio(float32 ratio);
73    float32 GetRatio() const;
74
75    /// Dump joint to dmLog
76    void Dump() override;
77
78  protected:
79
80    friend class b2Joint;
81    b2GearJoint(const b2GearJointDef* data);
82
83    void InitVelocityConstraints(const b2SolverData& data) override;
84    void SolveVelocityConstraints(const b2SolverData& data) override;
85    bool SolvePositionConstraints(const b2SolverData& data) override;
86
87    b2Joint* m_joint1;
88    b2Joint* m_joint2;
89
90    b2JointType m_typeA;
91    b2JointType m_typeB;
92
93    // Body A is connected to body C
94    // Body B is connected to body D
95    b2Body* m_bodyC;
96    b2Body* m_bodyD;
97
98    // Solver shared
99    b2Vec2 m_localAnchorA;
100   b2Vec2 m_localAnchorB;
101   b2Vec2 m_localAnchorC;
102   b2Vec2 m_localAnchorD;
103
104   b2Vec2 m_localAxisC;
105   b2Vec2 m_localAxisD;
106
107   float32 m_referenceAngleA;
108   float32 m_referenceAngleB;
109
110   float32 m_constant;
111   float32 m_ratio;
112
113   float32 m_impulse;
114
115   // Solver temp
116   int32 m_indexA, m_indexB, m_indexC, m_indexD;
117   b2Vec2 m_lcA, m_lcB, m_lcC, m_lcD;
118   float32 m_mA, m_mB, m_mC, m_mD;
119   float32 m_iA, m_iB, m_iC, m_iD;
120   b2Vec2 m_JvAC, m_JvBD;
121   float32 m_JwA, m_JwB, m_JwC, m_JwD;
122   float32 m_mass;
123 };
124
125 #endif
```



```
1   /*
2   * Copyright (c) 2006-2007 Erin Catto http://www.box2d.org
3   *
4   * This software is provided 'as-is', without any express or implied
5   * warranty.  In no event will the authors be held liable for any damages
6   * arising from the use of this software.
7   * Permission is granted to anyone to use this software for any purpose,
8   * including commercial applications, and to alter it and redistribute it
9   * freely, subject to the following restrictions:
10  * 1. The origin of this software must not be misrepresented; you must not
11  * claim that you wrote the original software. If you use this software
12  * in a product, an acknowledgment in the product documentation would be
13  * appreciated but is not required.
14  * 2. Altered source versions must be plainly marked as such, and must not be
15  * misrepresented as being the original software.
16  * 3. This notice may not be removed or altered from any source distribution.
17  */
18
19  #ifndef B2_FRICTION_JOINT_H
20  #define B2_FRICTION_JOINT_H
21
22  #include "Box2D/Dynamics/Joints/b2Joint.h"
23
24  /// Friction joint definition.
25  struct b2FrictionJointDef : public b2JointDef
26  {
27    b2FrictionJointDef()
28    {
29      type = e_frictionJoint;
30      localAnchorA.SetZero();
31      localAnchorB.SetZero();
32      maxForce = 0.0f;
33      maxTorque = 0.0f;
34    }
35
36    /// Initialize the bodies, anchors, axis, and reference angle using the world
37    /// anchor and world axis.
38    void Initialize(b2Body* bodyA, b2Body* bodyB, const b2Vec2& anchor);
39
40    /// The local anchor point relative to bodyA's origin.
41    b2Vec2 localAnchorA;
42
43    /// The local anchor point relative to bodyB's origin.
44    b2Vec2 localAnchorB;
45
46    /// The maximum friction force in N.
47    float32 maxForce;
48
49    /// The maximum friction torque in N-m.
50    float32 maxTorque;
51  };
52
53  /// Friction joint. This is used for top-down friction.
54  /// It provides 2D translational friction and angular friction.
55  class b2FrictionJoint : public b2Joint
56  {
57  public:
58    b2Vec2 GetAnchorA() const override;
59    b2Vec2 GetAnchorB() const override;
60
61    b2Vec2 GetReactionForce(float32 inv_dt) const override;
62    float32 GetReactionTorque(float32 inv_dt) const override;
63
64    /// The local anchor point relative to bodyA's origin.
65    const b2Vec2& GetLocalAnchorA() const { return m_localAnchorA; }
66
```

```
67    /// The local anchor point relative to bodyB's origin.
68    const b2Vec2& GetLocalAnchorB() const  { return m_localAnchorB; }
69
70    /// Set the maximum friction force in N.
71    void SetMaxForce(float32 force);
72
73    /// Get the maximum friction force in N.
74    float32 GetMaxForce() const;
75
76    /// Set the maximum friction torque in N*m.
77    void SetMaxTorque(float32 torque);
78
79    /// Get the maximum friction torque in N*m.
80    float32 GetMaxTorque() const;
81
82    /// Dump joint to dmLog
83    void Dump() override;
84
85  protected:
86
87    friend class b2Joint;
88
89    b2FrictionJoint(const b2FrictionJointDef* def);
90
91    void InitVelocityConstraints(const b2SolverData& data) override;
92    void SolveVelocityConstraints(const b2SolverData& data) override;
93    bool SolvePositionConstraints(const b2SolverData& data) override;
94
95    b2Vec2 m_localAnchorA;
96    b2Vec2 m_localAnchorB;
97
98    // Solver shared
99    b2Vec2 m_linearImpulse;
100   float32 m_angularImpulse;
101   float32 m_maxForce;
102   float32 m_maxTorque;
103
104   // Solver temp
105   int32 m_indexA;
106   int32 m_indexB;
107   b2Vec2 m_rA;
108   b2Vec2 m_rB;
109   b2Vec2 m_localCenterA;
110   b2Vec2 m_localCenterB;
111   float32 m_invMassA;
112   float32 m_invMassB;
113   float32 m_invIA;
114   float32 m_invIB;
115   b2Mat22 m_linearMass;
116   float32 m_angularMass;
117 };
118
119 #endif
```



```
1   /*
2   * Copyright (c) 2006-2007 Erin Catto http://www.box2d.org
3   *
4   * This software is provided 'as-is', without any express or implied
5   * warranty.  In no event will the authors be held liable for any damages
6   * arising from the use of this software.
7   * Permission is granted to anyone to use this software for any purpose,
8   * including commercial applications, and to alter it and redistribute it
9   * freely, subject to the following restrictions:
10  * 1. The origin of this software must not be misrepresented; you must not
11  * claim that you wrote the original software. If you use this software
12  * in a product, an acknowledgment in the product documentation would be
13  * appreciated but is not required.
14  * 2. Altered source versions must be plainly marked as such, and must not be
15  * misrepresented as being the original software.
16  * 3. This notice may not be removed or altered from any source distribution.
17  */
18
19  #ifndef B2_DISTANCE_JOINT_H
20  #define B2_DISTANCE_JOINT_H
21
22  #include "Box2D/Dynamics/Joints/b2Joint.h"
23
24  /// Distance joint definition. This requires defining an
25  /// anchor point on both bodies and the non-zero length of the
26  /// distance joint. The definition uses local anchor points
27  /// so that the initial configuration can violate the constraint
28  /// slightly. This helps when saving and loading a game.
29  /// @warning Do not use a zero or short length.
30  struct b2DistanceJointDef : public b2JointDef
31  {
32    b2DistanceJointDef()
33    {
34      type = e_distanceJoint;
35      localAnchorA.Set(0.0f, 0.0f);
36      localAnchorB.Set(0.0f, 0.0f);
37      length = 1.0f;
38      frequencyHz = 0.0f;
39      dampingRatio = 0.0f;
40    }
41
42    /// Initialize the bodies, anchors, and length using the world
43    /// anchors.
44    void Initialize(b2Body* bodyA, b2Body* bodyB,
45            const b2Vec2& anchorA, const b2Vec2& anchorB);
46
47    /// The local anchor point relative to bodyA's origin.
48    b2Vec2 localAnchorA;
49
50    /// The local anchor point relative to bodyB's origin.
51    b2Vec2 localAnchorB;
52
53    /// The natural length between the anchor points.
54    float32 length;
55
56    /// The mass-spring-damper frequency in Hertz. A value of 0
57    /// disables softness.
58    float32 frequencyHz;
59
60    /// The damping ratio. 0 = no damping, 1 = critical damping.
61    float32 dampingRatio;
62  };
63
64  /// A distance joint constrains two points on two bodies
65  /// to remain at a fixed distance from each other. You can view
66  /// this as a massless, rigid rod.
```

```
67  class b2DistanceJoint : public b2Joint
68  {
69  public:
70
71    b2Vec2 GetAnchorA() const override;
72    b2Vec2 GetAnchorB() const override;
73
74    /// Get the reaction force given the inverse time step.
75    /// Unit is N.
76    b2Vec2 GetReactionForce(float32 inv_dt) const override;
77
78    /// Get the reaction torque given the inverse time step.
79    /// Unit is N*m. This is always zero for a distance joint.
80    float32 GetReactionTorque(float32 inv_dt) const override;
81
82    /// The local anchor point relative to bodyA's origin.
83    const b2Vec2& GetLocalAnchorA() const { return m_localAnchorA; }
84
85    /// The local anchor point relative to bodyB's origin.
86    const b2Vec2& GetLocalAnchorB() const  { return m_localAnchorB; }
87
88    /// Set/get the natural length.
89    /// Manipulating the length can lead to non-physical behavior when the frequen
    cy is zero.
90    void SetLength(float32 length);
91    float32 GetLength() const;
92
93    /// Set/get frequency in Hz.
94    void SetFrequency(float32 hz);
95    float32 GetFrequency() const;
96
97    /// Set/get damping ratio.
98    void SetDampingRatio(float32 ratio);
99    float32 GetDampingRatio() const;
100
101   /// Dump joint to dmLog
102   void Dump() override;
103
104 protected:
105
106   friend class b2Joint;
107   b2DistanceJoint(const b2DistanceJointDef* data);
108
109   void InitVelocityConstraints(const b2SolverData& data) override;
110   void SolveVelocityConstraints(const b2SolverData& data) override;
111   bool SolvePositionConstraints(const b2SolverData& data) override;
112
113   float32 m_frequencyHz;
114   float32 m_dampingRatio;
115   float32 m_bias;
116
117   // Solver shared
118   b2Vec2 m_localAnchorA;
119   b2Vec2 m_localAnchorB;
120   float32 m_gamma;
121   float32 m_impulse;
122   float32 m_length;
123
124   // Solver temp
125   int32 m_indexA;
126   int32 m_indexB;
127   b2Vec2 m_u;
128   b2Vec2 m_rA;
129   b2Vec2 m_rB;
130   b2Vec2 m_localCenterA;
131   b2Vec2 m_localCenterB;
```



```
132   float32 m_invMassA;
133   float32 m_invMassB;
134   float32 m_invIA;
135   float32 m_invIB;
136   float32 m_mass;
137 };
138
139 inline void b2DistanceJoint::SetLength(float32 length)
140 {
141   m_length = length;
142 }
143
144 inline float32 b2DistanceJoint::GetLength() const
145 {
146   return m_length;
147 }
148
149 inline void b2DistanceJoint::SetFrequency(float32 hz)
150 {
151   m_frequencyHz = hz;
152 }
153
154 inline float32 b2DistanceJoint::GetFrequency() const
155 {
156   return m_frequencyHz;
157 }
158
159 inline void b2DistanceJoint::SetDampingRatio(float32 ratio)
160 {
161   m_dampingRatio = ratio;
162 }
163
164 inline float32 b2DistanceJoint::GetDampingRatio() const
165 {
166   return m_dampingRatio;
167 }
168
169 #endif
```

nov 26, 19 17:34 **b2PolygonContact.h** Page 1/1

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_POLYGON_CONTACT_H
#define B2_POLYGON_CONTACT_H

#include "Box2D/Dynamics/Contacts/b2Contact.h"

class b2BlockAllocator;

class b2PolygonContact : public b2Contact
{
public:
  static b2Contact* Create( b2Fixture* fixtureA, int32 indexA,
                b2Fixture* fixtureB, int32 indexB, b2BlockAllocator* allocator);
  static void Destroy(b2Contact* contact, b2BlockAllocator* allocator);

  b2PolygonContact(b2Fixture* fixtureA, b2Fixture* fixtureB);
  ~b2PolygonContact() {}

  void Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform&
 xfB) override;
};

#endif
```

nov 26, 19 17:34 **b2PolygonAndCircleContact.h** Page 1/1

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_POLYGON_AND_CIRCLE_CONTACT_H
#define B2_POLYGON_AND_CIRCLE_CONTACT_H

#include "Box2D/Dynamics/Contacts/b2Contact.h"

class b2BlockAllocator;

class b2PolygonAndCircleContact : public b2Contact
{
public:
  static b2Contact* Create(b2Fixture* fixtureA, int32 indexA, b2Fixture* fixture
B, int32 indexB, b2BlockAllocator* allocator);
  static void Destroy(b2Contact* contact, b2BlockAllocator* allocator);

  b2PolygonAndCircleContact(b2Fixture* fixtureA, b2Fixture* fixtureB);
  ~b2PolygonAndCircleContact() {}

  void Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform&
 xfB) override;
};

#endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_EDGE_AND_POLYGON_CONTACT_H
#define B2_EDGE_AND_POLYGON_CONTACT_H

#include "Box2D/Dynamics/Contacts/b2Contact.h"

class b2BlockAllocator;

class b2EdgeAndPolygonContact : public b2Contact
{
public:
  static b2Contact* Create( b2Fixture* fixtureA, int32 indexA,
                b2Fixture* fixtureB, int32 indexB, b2BlockAllocator* allocator);
  static void Destroy(b2Contact* contact, b2BlockAllocator* allocator);

  b2EdgeAndPolygonContact(b2Fixture* fixtureA, b2Fixture* fixtureB);
  ~b2EdgeAndPolygonContact() {}

  void Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform&
 xfB) override;
};

#endif
```



```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_EDGE_AND_CIRCLE_CONTACT_H
#define B2_EDGE_AND_CIRCLE_CONTACT_H

#include "Box2D/Dynamics/Contacts/b2Contact.h"

class b2BlockAllocator;

class b2EdgeAndCircleContact : public b2Contact
{
public:
  static b2Contact* Create( b2Fixture* fixtureA, int32 indexA,
                b2Fixture* fixtureB, int32 indexB, b2BlockAllocator* allocator);
  static void Destroy(b2Contact* contact, b2BlockAllocator* allocator);

  b2EdgeAndCircleContact(b2Fixture* fixtureA, b2Fixture* fixtureB);
  ~b2EdgeAndCircleContact() {}

  void Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform&
 xfB) override;
};

#endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_CONTACT_SOLVER_H
#define B2_CONTACT_SOLVER_H

#include "Box2D/Common/b2Math.h"
#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Dynamics/b2TimeStep.h"

class b2Contact;
class b2Body;
class b2StackAllocator;
struct b2ContactPositionConstraint;

struct b2VelocityConstraintPoint
{
  b2Vec2 rA;
  b2Vec2 rB;
  float32 normalImpulse;
  float32 tangentImpulse;
  float32 normalMass;
  float32 tangentMass;
  float32 velocityBias;
};

struct b2ContactVelocityConstraint
{
  b2VelocityConstraintPoint points[b2_maxManifoldPoints];
  b2Vec2 normal;
  b2Mat22 normalMass;
  b2Mat22 K;
  int32 indexA;
  int32 indexB;
  float32 invMassA, invMassB;
  float32 invIA, invIB;
  float32 friction;
  float32 restitution;
  float32 tangentSpeed;
  int32 pointCount;
  int32 contactIndex;
};

struct b2ContactSolverDef
{
  b2TimeStep step;
  b2Contact** contacts;
  int32 count;
  b2Position* positions;
  b2Velocity* velocities;
  b2StackAllocator* allocator;
```



```
};

class b2ContactSolver
{
public:
  b2ContactSolver(b2ContactSolverDef* def);
  ~b2ContactSolver();

  void InitializeVelocityConstraints();

  void WarmStart();
  void SolveVelocityConstraints();
  void StoreImpulses();

  bool SolvePositionConstraints();
  bool SolveTOIPositionConstraints(int32 toiIndexA, int32 toiIndexB);

  b2TimeStep m_step;
  b2Position* m_positions;
  b2Velocity* m_velocities;
  b2StackAllocator* m_allocator;
  b2ContactPositionConstraint* m_positionConstraints;
  b2ContactVelocityConstraint* m_velocityConstraints;
  b2Contact** m_contacts;
  int m_count;
};

#endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_CONTACT_H
#define B2_CONTACT_H

#include "Box2D/Common/b2Math.h"
#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Collision/Shapes/b2Shape.h"
#include "Box2D/Dynamics/b2Fixture.h"

class b2Body;
class b2Contact;
class b2Fixture;
class b2World;
class b2BlockAllocator;
class b2StackAllocator;
class b2ContactListener;

/// Friction mixing law. The idea is to allow either fixture to drive the friction to zero.
/// For example, anything slides on ice.
inline float32 b2MixFriction(float32 friction1, float32 friction2)
{
  return b2Sqrt(friction1 * friction2);
}

/// Restitution mixing law. The idea is allow for anything to bounce off an inelastic surface.
/// For example, a superball bounces on anything.
inline float32 b2MixRestitution(float32 restitution1, float32 restitution2)
{
  return restitution1 > restitution2 ? restitution1 : restitution2;
}

typedef b2Contact* b2ContactCreateFcn(  b2Fixture* fixtureA, int32 indexA,
                    b2Fixture* fixtureB, int32 indexB,
                    b2BlockAllocator* allocator);
typedef void b2ContactDestroyFcn(b2Contact* contact, b2BlockAllocator* allocator);

struct b2ContactRegister
{
  b2ContactCreateFcn* createFcn;
  b2ContactDestroyFcn* destroyFcn;
  bool primary;
};

/// A contact edge is used to connect bodies and contacts together
/// in a contact graph where each body is a node and each contact
/// is an edge. A contact edge belongs to a doubly linked list
```



```
/// maintained in each attached body. Each contact has two contact
/// nodes, one for each attached body.
struct b2ContactEdge
{
  b2Body* other;      ///< provides quick access to the other body attached.
  b2Contact* contact;   ///< the contact
  b2ContactEdge* prev;  ///< the previous contact edge in the body's contact list
  b2ContactEdge* next;  ///< the next contact edge in the body's contact list
};

/// The class manages contact between two shapes. A contact exists for each overlapping
/// AABB in the broad-phase (except if filtered). Therefore a contact object may exist
/// that has no contact points.
class b2Contact
{
public:

  /// Get the contact manifold. Do not modify the manifold unless you understand the
  /// internals of Box2D.
  b2Manifold* GetManifold();
  const b2Manifold* GetManifold() const;

  /// Get the world manifold.
  void GetWorldManifold(b2WorldManifold* worldManifold) const;

  /// Is this contact touching?
  bool IsTouching() const;

  /// Enable/disable this contact. This can be used inside the pre-solve
  /// contact listener. The contact is only disabled for the current
  /// time step (or sub-step in continuous collisions).
  void SetEnabled(bool flag);

  /// Has this contact been disabled?
  bool IsEnabled() const;

  /// Get the next contact in the world's contact list.
  b2Contact* GetNext();
  const b2Contact* GetNext() const;

  /// Get fixture A in this contact.
  b2Fixture* GetFixtureA();
  const b2Fixture* GetFixtureA() const;

  /// Get the child primitive index for fixture A.
  int32 GetChildIndexA() const;

  /// Get fixture B in this contact.
  b2Fixture* GetFixtureB();
  const b2Fixture* GetFixtureB() const;

  /// Get the child primitive index for fixture B.
  int32 GetChildIndexB() const;

  /// Override the default friction mixture. You can call this in b2ContactListener::PreSolve.
  /// This value persists until set or reset.
  void SetFriction(float32 friction);

  /// Get the friction.
  float32 GetFriction() const;

```

```
125   /// Reset the friction mixture to the default value.
126   void ResetFriction();
127
128   /// Override the default restitution mixture. You can call this in b2ContactLi
    stener::PreSolve.
129   /// The value persists until you set or reset.
130   void SetRestitution(float32 restitution);
131
132   /// Get the restitution.
133   float32 GetRestitution() const;
134
135   /// Reset the restitution to the default value.
136   void ResetRestitution();
137
138   /// Set the desired tangent speed for a conveyor belt behavior. In meters per
    second.
139   void SetTangentSpeed(float32 speed);
140
141   /// Get the desired tangent speed. In meters per second.
142   float32 GetTangentSpeed() const;
143
144   /// Evaluate this contact with your own manifold and transforms.
145   virtual void Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Tr
    ansform& xfB) = 0;
146
147 protected:
148   friend class b2ContactManager;
149   friend class b2World;
150   friend class b2ContactSolver;
151   friend class b2Body;
152   friend class b2Fixture;
153
154   // Flags stored in m_flags
155   enum
156   {
157     // Used when crawling contact graph when forming islands.
158     e_islandFlag    = 0x0001,
159
160     // Set when the shapes are touching.
161     e_touchingFlag    = 0x0002,
162
163     // This contact can be disabled (by user)
164     e_enabledFlag   = 0x0004,
165
166     // This contact needs filtering because a fixture filter was changed.
167     e_filterFlag    = 0x0008,
168
169     // This bullet contact had a TOI event
170     e_bulletHitFlag   = 0x0010,
171
172     // This contact has a valid TOI in m_toi
173     e_toiFlag     = 0x0020
174   };
175
176   /// Flag this contact for filtering. Filtering will occur the next time step.
177   void FlagForFiltering();
178
179   static void AddType(b2ContactCreateFcn* createFcn, b2ContactDestroyFcn* destro
    yFcn,
180             b2Shape::Type typeA, b2Shape::Type typeB);
181   static void InitializeRegisters();
182   static b2Contact* Create(b2Fixture* fixtureA, int32 indexA, b2Fixture* fixture
    B, int32 indexB, b2BlockAllocator* allocator);
183   static void Destroy(b2Contact* contact, b2Shape::Type typeA, b2Shape::Type typ
    eB, b2BlockAllocator* allocator);
184   static void Destroy(b2Contact* contact, b2BlockAllocator* allocator);
```



```
185
186   b2Contact() : m_fixtureA(nullptr), m_fixtureB(nullptr) {}
187   b2Contact(b2Fixture* fixtureA, int32 indexA, b2Fixture* fixtureB, int32 indexB
    );
188   virtual ~b2Contact() {}
189
190   void Update(b2ContactListener* listener);
191
192   static b2ContactRegister s_registers[b2Shape::e_typeCount][b2Shape::e_typeCoun
    t];
193   static bool s_initialized;
194
195   uint32 m_flags;
196
197   // World pool and list pointers.
198   b2Contact* m_prev;
199   b2Contact* m_next;
200
201   // Nodes for connecting bodies.
202   b2ContactEdge m_nodeA;
203   b2ContactEdge m_nodeB;
204
205   b2Fixture* m_fixtureA;
206   b2Fixture* m_fixtureB;
207
208   int32 m_indexA;
209   int32 m_indexB;
210
211   b2Manifold m_manifold;
212
213   int32 m_toiCount;
214   float32 m_toi;
215
216   float32 m_friction;
217   float32 m_restitution;
218
219   float32 m_tangentSpeed;
220 };
221
222 inline b2Manifold* b2Contact::GetManifold()
223 {
224   return &m_manifold;
225 }
226
227 inline const b2Manifold* b2Contact::GetManifold() const
228 {
229   return &m_manifold;
230 }
231
232 inline void b2Contact::GetWorldManifold(b2WorldManifold* worldManifold) const
233 {
234   const b2Body* bodyA = m_fixtureA→GetBody();
235   const b2Body* bodyB = m_fixtureB→GetBody();
236   const b2Shape* shapeA = m_fixtureA→GetShape();
237   const b2Shape* shapeB = m_fixtureB→GetShape();
238
239   worldManifold→Initialize(&m_manifold, bodyA→GetTransform(), shapeA→m_radius,
     bodyB→GetTransform(), shapeB→m_radius);
240 }
241
242 inline void b2Contact::SetEnabled(bool flag)
243 {
244   if (flag)
245   {
246     m_flags |= e_enabledFlag;
247   }
```

```
248   else
249   {
250     m_flags &= ~e_enabledFlag;
251   }
252 }
253
254 inline bool b2Contact::IsEnabled() const
255 {
256   return (m_flags & e_enabledFlag) == e_enabledFlag;
257 }
258
259 inline bool b2Contact::IsTouching() const
260 {
261   return (m_flags & e_touchingFlag) == e_touchingFlag;
262 }
263
264 inline b2Contact* b2Contact::GetNext()
265 {
266   return m_next;
267 }
268
269 inline const b2Contact* b2Contact::GetNext() const
270 {
271   return m_next;
272 }
273
274 inline b2Fixture* b2Contact::GetFixtureA()
275 {
276   return m_fixtureA;
277 }
278
279 inline const b2Fixture* b2Contact::GetFixtureA() const
280 {
281   return m_fixtureA;
282 }
283
284 inline b2Fixture* b2Contact::GetFixtureB()
285 {
286   return m_fixtureB;
287 }
288
289 inline int32 b2Contact::GetChildIndexA() const
290 {
291   return m_indexA;
292 }
293
294 inline const b2Fixture* b2Contact::GetFixtureB() const
295 {
296   return m_fixtureB;
297 }
298
299 inline int32 b2Contact::GetChildIndexB() const
300 {
301   return m_indexB;
302 }
303
304 inline void b2Contact::FlagForFiltering()
305 {
306   m_flags |= e_filterFlag;
307 }
308
309 inline void b2Contact::SetFriction(float32 friction)
310 {
311   m_friction = friction;
312 }
313
```



```
314 inline float32 b2Contact::GetFriction() const
315 {
316   return m_friction;
317 }
318
319 inline void b2Contact::ResetFriction()
320 {
321   m_friction = b2MixFriction(m_fixtureA→m_friction, m_fixtureB→m_friction);
322 }
323
324 inline void b2Contact::SetRestitution(float32 restitution)
325 {
326   m_restitution = restitution;
327 }
328
329 inline float32 b2Contact::GetRestitution() const
330 {
331   return m_restitution;
332 }
333
334 inline void b2Contact::ResetRestitution()
335 {
336   m_restitution = b2MixRestitution(m_fixtureA→m_restitution, m_fixtureB→m_rest
    itution);
337 }
338
339 inline void b2Contact::SetTangentSpeed(float32 speed)
340 {
341   m_tangentSpeed = speed;
342 }
343
344 inline float32 b2Contact::GetTangentSpeed() const
345 {
346   return m_tangentSpeed;
347 }
348
349 #endif
```

nov 26, 19 17:34 **b2CircleContact.h** Page 1/1

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_CIRCLE_CONTACT_H
#define B2_CIRCLE_CONTACT_H

#include "Box2D/Dynamics/Contacts/b2Contact.h"

class b2BlockAllocator;

class b2CircleContact : public b2Contact
{
public:
  static b2Contact* Create( b2Fixture* fixtureA, int32 indexA,
                b2Fixture* fixtureB, int32 indexB, b2BlockAllocator* allocator);
  static void Destroy(b2Contact* contact, b2BlockAllocator* allocator);

  b2CircleContact(b2Fixture* fixtureA, b2Fixture* fixtureB);
  ~b2CircleContact() {}

  void Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform&
 xfB) override;
};

#endif
```

nov 26, 19 17:34 **b2ChainAndPolygonContact.h** Page 1/1

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_CHAIN_AND_POLYGON_CONTACT_H
#define B2_CHAIN_AND_POLYGON_CONTACT_H

#include "Box2D/Dynamics/Contacts/b2Contact.h"

class b2BlockAllocator;

class b2ChainAndPolygonContact : public b2Contact
{
public:
  static b2Contact* Create( b2Fixture* fixtureA, int32 indexA,
                b2Fixture* fixtureB, int32 indexB, b2BlockAllocator* allocator);
  static void Destroy(b2Contact* contact, b2BlockAllocator* allocator);

  b2ChainAndPolygonContact(b2Fixture* fixtureA, int32 indexA, b2Fixture* fixture
B, int32 indexB);
  ~b2ChainAndPolygonContact() {}

  void Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform&
 xfB) override;
};

#endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_CHAIN_AND_CIRCLE_CONTACT_H
#define B2_CHAIN_AND_CIRCLE_CONTACT_H

#include "Box2D/Dynamics/Contacts/b2Contact.h"

class b2BlockAllocator;

class b2ChainAndCircleContact : public b2Contact
{
public:
  static b2Contact* Create( b2Fixture* fixtureA, int32 indexA,
                b2Fixture* fixtureB, int32 indexB, b2BlockAllocator* allocator);
  static void Destroy(b2Contact* contact, b2BlockAllocator* allocator);

  b2ChainAndCircleContact(b2Fixture* fixtureA, int32 indexA, b2Fixture* fixtureB
, int32 indexB);
  ~b2ChainAndCircleContact() {}

  void Evaluate(b2Manifold* manifold, const b2Transform& xfA, const b2Transform&
 xfB) override;
};

#endif
```



```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_WORLD_H
#define B2_WORLD_H

#include "Box2D/Common/b2Math.h"
#include "Box2D/Common/b2BlockAllocator.h"
#include "Box2D/Common/b2StackAllocator.h"
#include "Box2D/Dynamics/b2ContactManager.h"
#include "Box2D/Dynamics/b2WorldCallbacks.h"
#include "Box2D/Dynamics/b2TimeStep.h"

struct b2AABB;
struct b2BodyDef;
struct b2Color;
struct b2JointDef;
class b2Body;
class b2Draw;
class b2Fixture;
class b2Joint;

/// The world class manages all physics entities, dynamic simulation,
/// and asynchronous queries. The world also contains efficient memory
/// management facilities.
class b2World
{
public:
  /// Construct a world object.
  /// @param gravity the world gravity vector.
  b2World(const b2Vec2& gravity);

  /// Destruct the world. All physics entities are destroyed and all heap memory
 is released.
  ~b2World();

  /// Register a destruction listener. The listener is owned by you and must
  /// remain in scope.
  void SetDestructionListener(b2DestructionListener* listener);

  /// Register a contact filter to provide specific control over collision.
  /// Otherwise the default filter is used (b2_defaultFilter). The listener is
  /// owned by you and must remain in scope.
  void SetContactFilter(b2ContactFilter* filter);

  /// Register a contact event listener. The listener is owned by you and must
  /// remain in scope.
  void SetContactListener(b2ContactListener* listener);

  /// Register a routine for debug drawing. The debug draw functions are called
  /// inside with b2World::DrawDebugData method. The debug draw object is owned
```

```
66    /// by you and must remain in scope.
67    void SetDebugDraw(b2Draw* debugDraw);
68
69    /// Create a rigid body given a definition. No reference to the definition
70    /// is retained.
71    /// @warning This function is locked during callbacks.
72    b2Body* CreateBody(const b2BodyDef* def);
73
74    /// Destroy a rigid body given a definition. No reference to the definition
75    /// is retained. This function is locked during callbacks.
76    /// @warning This automatically deletes all associated shapes and joints.
77    /// @warning This function is locked during callbacks.
78    void DestroyBody(b2Body* body);
79
80    /// Create a joint to constrain bodies together. No reference to the definitio
    n
81    /// is retained. This may cause the connected bodies to cease colliding.
82    /// @warning This function is locked during callbacks.
83    b2Joint* CreateJoint(const b2JointDef* def);
84
85    /// Destroy a joint. This may cause the connected bodies to begin colliding.
86    /// @warning This function is locked during callbacks.
87    void DestroyJoint(b2Joint* joint);
88
89    /// Take a time step. This performs collision detection, integration,
90    /// and constraint solution.
91    /// @param timeStep the amount of time to simulate, this should not vary.
92    /// @param velocityIterations for the velocity constraint solver.
93    /// @param positionIterations for the position constraint solver.
94    void Step(  float32 timeStep,
95          int32 velocityIterations,
96          int32 positionIterations);
97
98    /// Manually clear the force buffer on all bodies. By default, forces are clea
    red automatically
99    /// after each call to Step. The default behavior is modified by calling SetAu
    toClearForces.
100   /// The purpose of this function is to support sub-stepping. Sub-stepping is o
    ften used to maintain
101   /// a fixed sized time step under a variable frame-rate.
102   /// When you perform sub-stepping you will disable auto clearing of forces and
     instead call
103   /// ClearForces after all sub-steps are complete in one pass of your game loop
    .
104   /// @see SetAutoClearForces
105   void ClearForces();
106
107   /// Call this to draw shapes and other debug draw data. This is intentionally
    non-const.
108   void DrawDebugData();
109
110   /// Query the world for all fixtures that potentially overlap the
111   /// provided AABB.
112   /// @param callback a user implemented callback class.
113   /// @param aabb the query box.
114   void QueryAABB(b2QueryCallback* callback, const b2AABB& aabb) const;
115
116   /// Ray-cast the world for all fixtures in the path of the ray. Your callback
117   /// controls whether you get the closest point, any point, or n-points.
118   /// The ray-cast ignores shapes that contain the starting point.
119   /// @param callback a user implemented callback class.
120   /// @param point1 the ray starting point
121   /// @param point2 the ray ending point
122   void RayCast(b2RayCastCallback* callback, const b2Vec2& point1, const b2Vec2&
    point2) const;
123
```



```
124   /// Get the world body list. With the returned body, use b2Body::GetNext to ge
    t
125   /// the next body in the world list. A nullptr body indicates the end of the l
    ist.
126   /// @return the head of the world body list.
127   b2Body* GetBodyList();
128   const b2Body* GetBodyList() const;
129
130   /// Get the world joint list. With the returned joint, use b2Joint::GetNext to
     get
131   /// the next joint in the world list. A nullptr joint indicates the end of the
     list.
132   /// @return the head of the world joint list.
133   b2Joint* GetJointList();
134   const b2Joint* GetJointList() const;
135
136   /// Get the world contact list. With the returned contact, use b2Contact::GetN
    ext to get
137   /// the next contact in the world list. A nullptr contact indicates the end of
     the list.
138   /// @return the head of the world contact list.
139   /// @warning contacts are created and destroyed in the middle of a time step.
140   /// Use b2ContactListener to avoid missing contacts.
141   b2Contact* GetContactList();
142   const b2Contact* GetContactList() const;
143
144   /// Enable/disable sleep.
145   void SetAllowSleeping(bool flag);
146   bool GetAllowSleeping() const { return m_allowSleep; }
147
148   /// Enable/disable warm starting. For testing.
149   void SetWarmStarting(bool flag) { m_warmStarting = flag; }
150   bool GetWarmStarting() const { return m_warmStarting; }
151
152   /// Enable/disable continuous physics. For testing.
153   void SetContinuousPhysics(bool flag) { m_continuousPhysics = flag; }
154   bool GetContinuousPhysics() const { return m_continuousPhysics; }
155
156   /// Enable/disable single stepped continuous physics. For testing.
157   void SetSubStepping(bool flag) { m_subStepping = flag; }
158   bool GetSubStepping() const { return m_subStepping; }
159
160   /// Get the number of broad-phase proxies.
161   int32 GetProxyCount() const;
162
163   /// Get the number of bodies.
164   int32 GetBodyCount() const;
165
166   /// Get the number of joints.
167   int32 GetJointCount() const;
168
169   /// Get the number of contacts (each may have 0 or more contact points).
170   int32 GetContactCount() const;
171
172   /// Get the height of the dynamic tree.
173   int32 GetTreeHeight() const;
174
175   /// Get the balance of the dynamic tree.
176   int32 GetTreeBalance() const;
177
178   /// Get the quality metric of the dynamic tree. The smaller the better.
179   /// The minimum is 1.
180   float32 GetTreeQuality() const;
181
182   /// Change the global gravity vector.
183   void SetGravity(const b2Vec2& gravity);
```

```
184
185   /// Get the global gravity vector.
186   b2Vec2 GetGravity() const;
187
188   /// Is the world locked (in the middle of a time step).
189   bool IsLocked() const;
190
191   /// Set flag to control automatic clearing of forces after each time step.
192   void SetAutoClearForces(bool flag);
193
194   /// Get the flag that controls automatic clearing of forces after each time st
    ep.
195   bool GetAutoClearForces() const;
196
197   /// Shift the world origin. Useful for large worlds.
198   /// The body shift formula is: position -= newOrigin
199   /// @param newOrigin the new origin with respect to the old origin
200   void ShiftOrigin(const b2Vec2& newOrigin);
201
202   /// Get the contact manager for testing.
203   const b2ContactManager& GetContactManager() const;
204
205   /// Get the current profile.
206   const b2Profile& GetProfile() const;
207
208   /// Dump the world into the log file.
209   /// @warning this should be called outside of a time step.
210   void Dump();
211
212 private:
213
214   // m_flags
215   enum
216   {
217     e_newFixture  = 0x0001,
218     e_locked    = 0x0002,
219     e_clearForces = 0x0004
220   };
221
222   friend class b2Body;
223   friend class b2Fixture;
224   friend class b2ContactManager;
225   friend class b2Controller;
226
227   void Solve(const b2TimeStep& step);
228   void SolveTOI(const b2TimeStep& step);
229
230   void DrawJoint(b2Joint* joint);
231   void DrawShape(b2Fixture* shape, const b2Transform& xf, const b2Color& color);
232
233   b2BlockAllocator m_blockAllocator;
234   b2StackAllocator m_stackAllocator;
235
236   int32 m_flags;
237
238   b2ContactManager m_contactManager;
239
240   b2Body* m_bodyList;
241   b2Joint* m_jointList;
242
243   int32 m_bodyCount;
244   int32 m_jointCount;
245
246   b2Vec2 m_gravity;
247   bool m_allowSleep;
248
```



```
249   b2DestructionListener* m_destructionListener;
250   b2Draw* m_debugDraw;
251
252   // This is used to compute the time step ratio to
253   // support a variable time step.
254   float32 m_inv_dt0;
255
256   // These are for debugging the solver.
257   bool m_warmStarting;
258   bool m_continuousPhysics;
259   bool m_subStepping;
260
261   bool m_stepComplete;
262
263   b2Profile m_profile;
264 };
265
266 inline b2Body* b2World::GetBodyList()
267 {
268   return m_bodyList;
269 }
270
271 inline const b2Body* b2World::GetBodyList() const
272 {
273   return m_bodyList;
274 }
275
276 inline b2Joint* b2World::GetJointList()
277 {
278   return m_jointList;
279 }
280
281 inline const b2Joint* b2World::GetJointList() const
282 {
283   return m_jointList;
284 }
285
286 inline b2Contact* b2World::GetContactList()
287 {
288   return m_contactManager.m_contactList;
289 }
290
291 inline const b2Contact* b2World::GetContactList() const
292 {
293   return m_contactManager.m_contactList;
294 }
295
296 inline int32 b2World::GetBodyCount() const
297 {
298   return m_bodyCount;
299 }
300
301 inline int32 b2World::GetJointCount() const
302 {
303   return m_jointCount;
304 }
305
306 inline int32 b2World::GetContactCount() const
307 {
308   return m_contactManager.m_contactCount;
309 }
310
311 inline void b2World::SetGravity(const b2Vec2& gravity)
312 {
313   m_gravity = gravity;
314 }
```

```
315
316 inline b2Vec2 b2World::GetGravity() const
317 {
318   return m_gravity;
319 }
320
321 inline bool b2World::IsLocked() const
322 {
323   return (m_flags & e_locked) ≡ e_locked;
324 }
325
326 inline void b2World::SetAutoClearForces(bool flag)
327 {
328   if (flag)
329   {
330     m_flags |= e_clearForces;
331   }
332   else
333   {
334     m_flags &= ~e_clearForces;
335   }
336 }
337
338 /// Get the flag that controls automatic clearing of forces after each time step
    .
339 inline bool b2World::GetAutoClearForces() const
340 {
341   return (m_flags & e_clearForces) ≡ e_clearForces;
342 }
343
344 inline const b2ContactManager& b2World::GetContactManager() const
345 {
346   return m_contactManager;
347 }
348
349 inline const b2Profile& b2World::GetProfile() const
350 {
351   return m_profile;
352 }
353
354 #endif
```



```
1  /*
2  * Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as-is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #ifndef B2_WORLD_CALLBACKS_H
20 #define B2_WORLD_CALLBACKS_H
21
22 #include "Box2D/Common/b2Settings.h"
23
24 struct b2Vec2;
25 struct b2Transform;
26 class b2Fixture;
27 class b2Body;
28 class b2Joint;
29 class b2Contact;
30 struct b2ContactResult;
31 struct b2Manifold;
32
33 /// Joints and fixtures are destroyed when their associated
34 /// body is destroyed. Implement this listener so that you
35 /// may nullify references to these joints and shapes.
36 class b2DestructionListener
37 {
38 public:
39   virtual ~b2DestructionListener() {}
40
41   /// Called when any joint is about to be destroyed due
42   /// to the destruction of one of its attached bodies.
43   virtual void SayGoodbye(b2Joint* joint) = 0;
44
45   /// Called when any fixture is about to be destroyed due
46   /// to the destruction of its parent body.
47   virtual void SayGoodbye(b2Fixture* fixture) = 0;
48 };
49
50 /// Implement this class to provide collision filtering. In other words, you can
    implement
51 /// this class if you want finer control over contact creation.
52 class b2ContactFilter
53 {
54 public:
55   virtual ~b2ContactFilter() {}
56
57   /// Return true if contact calculations should be performed between these two
   shapes.
58   /// @warning for performance reasons this is only called when the AABBs begin
   to overlap.
59   virtual bool ShouldCollide(b2Fixture* fixtureA, b2Fixture* fixtureB);
60 };
61
62 /// Contact impulses for reporting. Impulses are used instead of forces because
63 /// sub-step forces may approach infinity for rigid body collisions. These
```

```
64  /// match up one-to-one with the contact points in b2Manifold.
65  struct b2ContactImpulse
66  {
67    float32 normalImpulses[b2_maxManifoldPoints];
68    float32 tangentImpulses[b2_maxManifoldPoints];
69    int32 count;
70  };
71
72  /// Implement this class to get contact information. You can use these results f
    or
73  /// things like sounds and game logic. You can also get contact results by
74  /// traversing the contact lists after the time step. However, you might miss
75  /// some contacts because continuous physics leads to sub-stepping.
76  /// Additionally you may receive multiple callbacks for the same contact in a
77  /// single time step.
78  /// You should strive to make your callbacks efficient because there may be
79  /// many callbacks per time step.
80  /// @warning You cannot create/destroy Box2D entities inside these callbacks.
81  class b2ContactListener
82  {
83  public:
84    virtual ~b2ContactListener() {}
85
86    /// Called when two fixtures begin to touch.
87    virtual void BeginContact(b2Contact* contact) { B2_NOT_USED(contact); }
88
89    /// Called when two fixtures cease to touch.
90    virtual void EndContact(b2Contact* contact) { B2_NOT_USED(contact); }
91
92    /// This is called after a contact is updated. This allows you to inspect a
93    /// contact before it goes to the solver. If you are careful, you can modify t
    he
94    /// contact manifold (e.g. disable contact).
95    /// A copy of the old manifold is provided so that you can detect changes.
96    /// Note: this is called only for awake bodies.
97    /// Note: this is called even when the number of contact points is zero.
98    /// Note: this is not called for sensors.
99    /// Note: if you set the number of contact points to zero, you will not
100   /// get an EndContact callback. However, you may get a BeginContact callback
101   /// the next step.
102   virtual void PreSolve(b2Contact* contact, const b2Manifold* oldManifold)
103   {
104     B2_NOT_USED(contact);
105     B2_NOT_USED(oldManifold);
106   }
107
108   /// This lets you inspect a contact after the solver is finished. This is usef
    ul
109   /// for inspecting impulses.
110   /// Note: the contact manifold does not include time of impact impulses, which
     can be
111   /// arbitrarily large if the sub-step is small. Hence the impulse is provided
    explicitly
112   /// in a separate data structure.
113   /// Note: this is only called for contacts that are touching, solid, and awake
    .
114   virtual void PostSolve(b2Contact* contact, const b2ContactImpulse* impulse)
115   {
116     B2_NOT_USED(contact);
117     B2_NOT_USED(impulse);
118   }
119 };
120
121 /// Callback class for AABB queries.
122 /// See b2World::Query
123 class b2QueryCallback
```



```
124 {
125 public:
126   virtual ~b2QueryCallback() {}
127
128   /// Called for each fixture found in the query AABB.
129   /// @return false to terminate the query.
130   virtual bool ReportFixture(b2Fixture* fixture) = 0;
131 };
132
133 /// Callback class for ray casts.
134 /// See b2World::RayCast
135 class b2RayCastCallback
136 {
137 public:
138   virtual ~b2RayCastCallback() {}
139
140   /// Called for each fixture found in the query. You control how the ray cast
141   /// proceeds by returning a float:
142   /// return -1: ignore this fixture and continue
143   /// return 0: terminate the ray cast
144   /// return fraction: clip the ray to this point
145   /// return 1: don't clip the ray and continue
146   /// @param fixture the fixture hit by the ray
147   /// @param point the point of initial intersection
148   /// @param normal the normal vector at the point of intersection
149   /// @return -1 to filter, 0 to terminate, fraction to clip the ray for
150   /// closest hit, 1 to continue
151   virtual float32 ReportFixture(  b2Fixture* fixture, const b2Vec2& point,
152                                   const b2Vec2& normal, float32 fraction) = 0;
153 };
154
155 #endif
```

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_TIME_STEP_H
#define B2_TIME_STEP_H

#include "Box2D/Common/b2Math.h"

/// Profiling data. Times are in milliseconds.
struct b2Profile
{
  float32 step;
  float32 collide;
  float32 solve;
  float32 solveInit;
  float32 solveVelocity;
  float32 solvePosition;
  float32 broadphase;
  float32 solveTOI;
};

/// This is an internal structure.
struct b2TimeStep
{
  float32 dt;     // time step
  float32 inv_dt;   // inverse time step (0 if dt == 0).
  float32 dtRatio;  // dt * inv_dt0
  int32 velocityIterations;
  int32 positionIterations;
  bool warmStarting;
};

/// This is an internal structure.
struct b2Position
{
  b2Vec2 c;
  float32 a;
};

/// This is an internal structure.
struct b2Velocity
{
  b2Vec2 v;
  float32 w;
};

/// Solver Data
struct b2SolverData
{
  b2TimeStep step;
  b2Position* positions;
```



```
  b2Velocity* velocities;
};

#endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_ISLAND_H
#define B2_ISLAND_H

#include "Box2D/Common/b2Math.h"
#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2TimeStep.h"

class b2Contact;
class b2Joint;
class b2StackAllocator;
class b2ContactListener;
struct b2ContactVelocityConstraint;
struct b2Profile;

/// This is an internal class.
class b2Island
{
public:
  b2Island(int32 bodyCapacity, int32 contactCapacity, int32 jointCapacity,
      b2StackAllocator* allocator, b2ContactListener* listener);
  ~b2Island();

  void Clear()
  {
    m_bodyCount = 0;
    m_contactCount = 0;
    m_jointCount = 0;
  }

  void Solve(b2Profile* profile, const b2TimeStep& step, const b2Vec2& gravity,
bool allowSleep);

  void SolveTOI(const b2TimeStep& subStep, int32 toiIndexA, int32 toiIndexB);

  void Add(b2Body* body)
  {
    b2Assert(m_bodyCount < m_bodyCapacity);
    body->m_islandIndex = m_bodyCount;
    m_bodies[m_bodyCount] = body;
    ++m_bodyCount;
  }

  void Add(b2Contact* contact)
  {
    b2Assert(m_contactCount < m_contactCapacity);
    m_contacts[m_contactCount++] = contact;
  }

```



```
  void Add(b2Joint* joint)
  {
    b2Assert(m_jointCount < m_jointCapacity);
    m_joints[m_jointCount++] = joint;
  }

  void Report(const b2ContactVelocityConstraint* constraints);

  b2StackAllocator* m_allocator;
  b2ContactListener* m_listener;

  b2Body** m_bodies;
  b2Contact** m_contacts;
  b2Joint** m_joints;

  b2Position* m_positions;
  b2Velocity* m_velocities;

  int32 m_bodyCount;
  int32 m_jointCount;
  int32 m_contactCount;

  int32 m_bodyCapacity;
  int32 m_contactCapacity;
  int32 m_jointCapacity;
};

#endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_FIXTURE_H
#define B2_FIXTURE_H

#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Collision/Shapes/b2Shape.h"

class b2BlockAllocator;
class b2Body;
class b2BroadPhase;
class b2Fixture;

/// This holds contact filtering data.
struct b2Filter
{
  b2Filter()
  {
    categoryBits = 0x0001;
    maskBits = 0xFFFF;
    groupIndex = 0;
  }

  /// The collision category bits. Normally you would just set one bit.
  uint16 categoryBits;

  /// The collision mask bits. This states the categories that this
  /// shape would accept for collision.
  uint16 maskBits;

  /// Collision groups allow a certain group of objects to never collide (negati
ve)
  /// or always collide (positive). Zero means no collision group. Non-zero grou
p
  /// filtering always wins against the mask bits.
  int16 groupIndex;
};

/// A fixture definition is used to create a fixture. This class defines an
/// abstract fixture definition. You can reuse fixture definitions safely.
struct b2FixtureDef
{
  /// The constructor sets the default fixture definition values.
  b2FixtureDef()
  {
    shape = nullptr;
    userData = nullptr;
    friction = 0.2f;
    restitution = 0.0f;
```



```
    density = 0.0f;
    isSensor = false;
  }

  /// The shape, this must be set. The shape will be cloned, so you
  /// can create the shape on the stack.
  const b2Shape* shape;

  /// Use this to store application specific fixture data.
  void* userData;

  /// The friction coefficient, usually in the range [0,1].
  float32 friction;

  /// The restitution (elasticity) usually in the range [0,1].
  float32 restitution;

  /// The density, usually in kg/m^2.
  float32 density;

  /// A sensor shape collects contact information but never generates a collisio
n
  /// response.
  bool isSensor;

  /// Contact filtering data.
  b2Filter filter;
};

/// This proxy is used internally to connect fixtures to the broad-phase.
struct b2FixtureProxy
{
  b2AABB aabb;
  b2Fixture* fixture;
  int32 childIndex;
  int32 proxyId;
};

/// A fixture is used to attach a shape to a body for collision detection. A fix
ture
/// inherits its transform from its parent. Fixtures hold additional non-geometr
ic data
/// such as friction, collision filters, etc.
/// Fixtures are created via b2Body::CreateFixture.
/// @warning you cannot reuse fixtures.
class b2Fixture
{
public:
  /// Get the type of the child shape. You can use this to down cast to the conc
rete shape.
  /// @return the shape type.
  b2Shape::Type GetType() const;

  /// Get the child shape. You can modify the child shape, however you should no
t change the
  /// number of vertices because this will crash some collision caching mechanis
ms.
  /// Manipulating the shape may lead to non-physical behavior.
  b2Shape* GetShape();
  const b2Shape* GetShape() const;

  /// Set if this fixture is a sensor.
  void SetSensor(bool sensor);

  /// Is this fixture a sensor (non-solid)?
  /// @return the true if the shape is a sensor.
```

```
125   bool IsSensor() const;
126
127   /// Set the contact filtering data. This will not update contacts until the ne
    xt time
128   /// step when either parent body is active and awake.
129   /// This automatically calls Refilter.
130   void SetFilterData(const b2Filter& filter);
131
132   /// Get the contact filtering data.
133   const b2Filter& GetFilterData() const;
134
135   /// Call this if you want to establish collision that was previously disabled
    by b2ContactFilter::ShouldCollide.
136   void Refilter();
137
138   /// Get the parent body of this fixture. This is nullptr if the fixture is not
     attached.
139   /// @return the parent body.
140   b2Body* GetBody();
141   const b2Body* GetBody() const;
142
143   /// Get the next fixture in the parent body's fixture list.
144   /// @return the next shape.
145   b2Fixture* GetNext();
146   const b2Fixture* GetNext() const;
147
148   /// Get the user data that was assigned in the fixture definition. Use this to
149   /// store your application specific data.
150   void* GetUserData() const;
151
152   /// Set the user data. Use this to store your application specific data.
153   void SetUserData(void* data);
154
155   /// Test a point for containment in this fixture.
156   /// @param p a point in world coordinates.
157   bool TestPoint(const b2Vec2& p) const;
158
159   /// Cast a ray against this shape.
160   /// @param output the ray-cast results.
161   /// @param input the ray-cast input parameters.
162   bool RayCast(b2RayCastOutput* output, const b2RayCastInput& input, int32 child
    Index) const;
163
164   /// Get the mass data for this fixture. The mass data is based on the density
    and
165   /// the shape. The rotational inertia is about the shape's origin. This operat
    ion
166   /// may be expensive.
167   void GetMassData(b2MassData* massData) const;
168
169   /// Set the density of this fixture. This will _not_ automatically adjust the
    mass
170   /// of the body. You must call b2Body::ResetMassData to update the body's mass
    .
171   void SetDensity(float32 density);
172
173   /// Get the density of this fixture.
174   float32 GetDensity() const;
175
176   /// Get the coefficient of friction.
177   float32 GetFriction() const;
178
179   /// Set the coefficient of friction. This will _not_ change the friction of
180   /// existing contacts.
181   void SetFriction(float32 friction);
182
```



```
183   /// Get the coefficient of restitution.
184   float32 GetRestitution() const;
185
186   /// Set the coefficient of restitution. This will _not_ change the restitution
     of
187   /// existing contacts.
188   void SetRestitution(float32 restitution);
189
190   /// Get the fixture's AABB. This AABB may be enlarge and/or stale.
191   /// If you need a more accurate AABB, compute it using the shape and
192   /// the body transform.
193   const b2AABB& GetAABB(int32 childIndex) const;
194
195   /// Dump this fixture to the log file.
196   void Dump(int32 bodyIndex);
197
198 protected:
199
200   friend class b2Body;
201   friend class b2World;
202   friend class b2Contact;
203   friend class b2ContactManager;
204
205   b2Fixture();
206
207   // We need separation create/destroy functions from the constructor/destructor
     because
208   // the destructor cannot access the allocator (no destructor arguments allowed
     by C++).
209   void Create(b2BlockAllocator* allocator, b2Body* body, const b2FixtureDef* def
    );
210   void Destroy(b2BlockAllocator* allocator);
211
212   // These support body activation/deactivation.
213   void CreateProxies(b2BroadPhase* broadPhase, const b2Transform& xf);
214   void DestroyProxies(b2BroadPhase* broadPhase);
215
216   void Synchronize(b2BroadPhase* broadPhase, const b2Transform& xf1, const b2Tra
    nsform& xf2);
217
218   float32 m_density;
219
220   b2Fixture* m_next;
221   b2Body* m_body;
222
223   b2Shape* m_shape;
224
225   float32 m_friction;
226   float32 m_restitution;
227
228   b2FixtureProxy* m_proxies;
229   int32 m_proxyCount;
230
231   b2Filter m_filter;
232
233   bool m_isSensor;
234
235   void* m_userData;
236 };
237
238 inline b2Shape::Type b2Fixture::GetType() const
239 {
240   return m_shape->GetType();
241 }
242
243 inline b2Shape* b2Fixture::GetShape()
```

```
244 {
245   return m_shape;
246 }
247
248 inline const b2Shape* b2Fixture::GetShape() const
249 {
250   return m_shape;
251 }
252
253 inline bool b2Fixture::IsSensor() const
254 {
255   return m_isSensor;
256 }
257
258 inline const b2Filter& b2Fixture::GetFilterData() const
259 {
260   return m_filter;
261 }
262
263 inline void* b2Fixture::GetUserData() const
264 {
265   return m_userData;
266 }
267
268 inline void b2Fixture::SetUserData(void* data)
269 {
270   m_userData = data;
271 }
272
273 inline b2Body* b2Fixture::GetBody()
274 {
275   return m_body;
276 }
277
278 inline const b2Body* b2Fixture::GetBody() const
279 {
280   return m_body;
281 }
282
283 inline b2Fixture* b2Fixture::GetNext()
284 {
285   return m_next;
286 }
287
288 inline const b2Fixture* b2Fixture::GetNext() const
289 {
290   return m_next;
291 }
292
293 inline void b2Fixture::SetDensity(float32 density)
294 {
295   b2Assert(b2IsValid(density) ∧ density ≥ 0.0f);
296   m_density = density;
297 }
298
299 inline float32 b2Fixture::GetDensity() const
300 {
301   return m_density;
302 }
303
304 inline float32 b2Fixture::GetFriction() const
305 {
306   return m_friction;
307 }
308
309 inline void b2Fixture::SetFriction(float32 friction)
```



```
310 {
311   m_friction = friction;
312 }
313
314 inline float32 b2Fixture::GetRestitution() const
315 {
316   return m_restitution;
317 }
318
319 inline void b2Fixture::SetRestitution(float32 restitution)
320 {
321   m_restitution = restitution;
322 }
323
324 inline bool b2Fixture::TestPoint(const b2Vec2& p) const
325 {
326   return m_shape→TestPoint(m_body→GetTransform(), p);
327 }
328
329 inline bool b2Fixture::RayCast(b2RayCastOutput* output, const b2RayCastInput& in
    put, int32 childIndex) const
330 {
331   return m_shape→RayCast(output, input, m_body→GetTransform(), childIndex);
332 }
333
334 inline void b2Fixture::GetMassData(b2MassData* massData) const
335 {
336   m_shape→ComputeMass(massData, m_density);
337 }
338
339 inline const b2AABB& b2Fixture::GetAABB(int32 childIndex) const
340 {
341   b2Assert(0 ≤ childIndex ∧ childIndex < m_proxyCount);
342   return m_proxies[childIndex].aabb;
343 }
344
345 #endif
```

nov 26, 19 17:34 **b2ContactManager.h** Page 1/1

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_CONTACT_MANAGER_H
#define B2_CONTACT_MANAGER_H

#include "Box2D/Collision/b2BroadPhase.h"

class b2Contact;
class b2ContactFilter;
class b2ContactListener;
class b2BlockAllocator;

// Delegate of b2World.
class b2ContactManager
{
public:
  b2ContactManager();

  // Broad-phase callback.
  void AddPair(void* proxyUserDataA, void* proxyUserDataB);

  void FindNewContacts();

  void Destroy(b2Contact* c);

  void Collide();

  b2BroadPhase m_broadPhase;
  b2Contact* m_contactList;
  int32 m_contactCount;
  b2ContactFilter* m_contactFilter;
  b2ContactListener* m_contactListener;
  b2BlockAllocator* m_allocator;
};

#endif
```

nov 26, 19 17:34 **b2Body.h** Page 1/14

```
/*
* Copyright (c) 2006-2011 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_BODY_H
#define B2_BODY_H

#include "Box2D/Common/b2Math.h"
#include "Box2D/Collision/Shapes/b2Shape.h"
#include <memory>

class b2Fixture;
class b2Joint;
class b2Contact;
class b2Controller;
class b2World;
struct b2FixtureDef;
struct b2JointEdge;
struct b2ContactEdge;

//Modificacion
class Colisionable;

/// The body type.
/// static: zero mass, zero velocity, may be manually moved
/// kinematic: zero mass, non-zero velocity set by user, moved by solver
/// dynamic: positive mass, non-zero velocity determined by forces, moved by sol
ver
enum b2BodyType
{
  b2_staticBody = 0,
  b2_kinematicBody,
  b2_dynamicBody


  //b2_bulletBody,
};

/// A body definition holds all the data needed to construct a rigid body.
/// You can safely re-use body definitions. Shapes are added to a body after con
struction.
struct b2BodyDef
{
  /// This constructor sets the body definition default values.
  b2BodyDef()
  {
    userData = nullptr;
    position.Set(0.0f, 0.0f);
    angle = 0.0f;
    linearVelocity.Set(0.0f, 0.0f);
    angularVelocity = 0.0f;
    linearDamping = 0.0f;
```

```
65      angularDamping = 0.0f;
66      allowSleep = true;
67      awake = true;
68      fixedRotation = false;
69      bullet = false;
70      type = b2_staticBody;
71      active = true;
72      gravityScale = 1.0f;
73    }
74
75    /// The body type: static, kinematic, or dynamic.
76    /// Note: if a dynamic body would have zero mass, the mass is set to one.
77    b2BodyType type;
78
79    /// The world position of the body. Avoid creating bodies at the origin
80    /// since this can lead to many overlapping shapes.
81    b2Vec2 position;
82
83    /// The world angle of the body in radians.
84    float32 angle;
85
86    /// The linear velocity of the body's origin in world co-ordinates.
87    b2Vec2 linearVelocity;
88
89    /// The angular velocity of the body.
90    float32 angularVelocity;
91
92    /// Linear damping is use to reduce the linear velocity. The damping parameter
93    /// can be larger than 1.0f but the damping effect becomes sensitive to the
94    /// time step when the damping parameter is large.
95    /// Units are 1/time
96    float32 linearDamping;
97
98    /// Angular damping is use to reduce the angular velocity. The damping paramet
      er
99    /// can be larger than 1.0f but the damping effect becomes sensitive to the
100   /// time step when the damping parameter is large.
101   /// Units are 1/time
102   float32 angularDamping;
103
104   /// Set this flag to false if this body should never fall asleep. Note that
105   /// this increases CPU usage.
106   bool allowSleep;
107
108   /// Is this body initially awake or sleeping?
109   bool awake;
110
111   /// Should this body be prevented from rotating? Useful for characters.
112   bool fixedRotation;
113
114   /// Is this a fast moving body that should be prevented from tunneling through
115   /// other moving bodies? Note that all bodies are prevented from tunneling thr
      ough
116   /// kinematic and static bodies. This setting is only considered on dynamic bo
      dies.
117   /// @warning You should use this flag sparingly since it increases processing
      time.
118   bool bullet;
119
120   /// Does this body start out active?
121   bool active;
122
123   /// Use this to store application specific body data.
124   Colisionable* userData;
125
126   /// Scale the gravity applied to this body.
```



```
127   float32 gravityScale;
128 };
129
130 /// A rigid body. These are created via b2World::CreateBody.
131 class b2Body
132 {
133 public:
134   /// Creates a fixture and attach it to this body. Use this function if you nee
      d
135   /// to set some fixture parameters, like friction. Otherwise you can create th
      e
136   /// fixture directly from a shape.
137   /// If the density is non-zero, this function automatically updates the mass o
      f the body.
138   /// Contacts are not created until the next time step.
139   /// @param def the fixture definition.
140   /// @warning This function is locked during callbacks.
141   b2Fixture* CreateFixture(const b2FixtureDef* def);
142
143   /// Creates a fixture from a shape and attach it to this body.
144   /// This is a convenience function. Use b2FixtureDef if you need to set parame
      ters
145   /// like friction, restitution, user data, or filtering.
146   /// If the density is non-zero, this function automatically updates the mass o
      f the body.
147   /// @param shape the shape to be cloned.
148   /// @param density the shape density (set to zero for static bodies).
149   /// @warning This function is locked during callbacks.
150   b2Fixture* CreateFixture(const b2Shape* shape, float32 density);
151
152   /// Destroy a fixture. This removes the fixture from the broad-phase and
153   /// destroys all contacts associated with this fixture. This will
154   /// automatically adjust the mass of the body if the body is dynamic and the
155   /// fixture has positive density.
156   /// All fixtures attached to a body are implicitly destroyed when the body is
      destroyed.
157   /// @param fixture the fixture to be removed.
158   /// @warning This function is locked during callbacks.
159   void DestroyFixture(b2Fixture* fixture);
160
161   /// Set the position of the body's origin and rotation.
162   /// Manipulating a body's transform may cause non-physical behavior.
163   /// Note: contacts are updated on the next call to b2World::Step.
164   /// @param position the world position of the body's local origin.
165   /// @param angle the world rotation in radians.
166   void SetTransform(const b2Vec2& position, float32 angle);
167
168   /// Get the body transform for the body's origin.
169   /// @return the world transform of the body's origin.
170   const b2Transform& GetTransform() const;
171
172   /// Get the world body origin position.
173   /// @return the world position of the body's origin.
174   const b2Vec2& GetPosition() const;
175
176   /// Get the angle in radians.
177   /// @return the current world rotation angle in radians.
178   float32 GetAngle() const;
179
180   /// Get the world position of the center of mass.
181   const b2Vec2& GetWorldCenter() const;
182
183   /// Get the local position of the center of mass.
184   const b2Vec2& GetLocalCenter() const;
185
186   /// Set the linear velocity of the center of mass.
```

```
187    /// @param v the new linear velocity of the center of mass.
188    void SetLinearVelocity(const b2Vec2& v);
189
190    /// Get the linear velocity of the center of mass.
191    /// @return the linear velocity of the center of mass.
192    const b2Vec2& GetLinearVelocity() const;
193
194    /// Set the angular velocity.
195    /// @param omega the new angular velocity in radians/second.
196    void SetAngularVelocity(float32 omega);
197
198    /// Get the angular velocity.
199    /// @return the angular velocity in radians/second.
200    float32 GetAngularVelocity() const;
201
202    /// Apply a force at a world point. If the force is not
203    /// applied at the center of mass, it will generate a torque and
204    /// affect the angular velocity. This wakes up the body.
205    /// @param force the world force vector, usually in Newtons (N).
206    /// @param point the world position of the point of application.
207    /// @param wake also wake up the body
208    void ApplyForce(const b2Vec2& force, const b2Vec2& point, bool wake);
209
210    /// Apply a force to the center of mass. This wakes up the body.
211    /// @param force the world force vector, usually in Newtons (N).
212    /// @param wake also wake up the body
213    void ApplyForceToCenter(const b2Vec2& force, bool wake);
214
215    /// Apply a torque. This affects the angular velocity
216    /// without affecting the linear velocity of the center of mass.
217    /// @param torque about the z-axis (out of the screen), usually in N-m.
218    /// @param wake also wake up the body
219    void ApplyTorque(float32 torque, bool wake);
220
221    /// Apply an impulse at a point. This immediately modifies the velocity.
222    /// It also modifies the angular velocity if the point of application
223    /// is not at the center of mass. This wakes up the body.
224    /// @param impulse the world impulse vector, usually in N-seconds or kg-m/s.
225    /// @param point the world position of the point of application.
226    /// @param wake also wake up the body
227    void ApplyLinearImpulse(const b2Vec2& impulse, const b2Vec2& point, bool wake)
     ;
228
229    /// Apply an impulse to the center of mass. This immediately modifies the velo
    city.
230    /// @param impulse the world impulse vector, usually in N-seconds or kg-m/s.
231    /// @param wake also wake up the body
232    void ApplyLinearImpulseToCenter(const b2Vec2& impulse, bool wake);
233
234    /// Apply an angular impulse.
235    /// @param impulse the angular impulse in units of kg*m*m/s
236    /// @param wake also wake up the body
237    void ApplyAngularImpulse(float32 impulse, bool wake);
238
239    /// Get the total mass of the body.
240    /// @return the mass, usually in kilograms (kg).
241    float32 GetMass() const;
242
243    /// Get the rotational inertia of the body about the local origin.
244    /// @return the rotational inertia, usually in kg-m^2.
245    float32 GetInertia() const;
246
247    /// Get the mass data of the body.
248    /// @return a struct containing the mass, inertia and center of the body.
249    void GetMassData(b2MassData* data) const;
250
```



```
251    /// Set the mass properties to override the mass properties of the fixtures.
252    /// Note that this changes the center of mass position.
253    /// Note that creating or destroying fixtures can also alter the mass.
254    /// This function has no effect if the body isn't dynamic.
255    /// @param massData the mass properties.
256    void SetMassData(const b2MassData* data);
257
258    /// This resets the mass properties to the sum of the mass properties of the f
    ixtures.
259    /// This normally does not need to be called unless you called SetMassData to
    override
260    /// the mass and you later want to reset the mass.
261    void ResetMassData();
262
263    /// Get the world coordinates of a point given the local coordinates.
264    /// @param localPoint a point on the body measured relative the the body's ori
    gin.
265    /// @return the same point expressed in world coordinates.
266    b2Vec2 GetWorldPoint(const b2Vec2& localPoint) const;
267
268    /// Get the world coordinates of a vector given the local coordinates.
269    /// @param localVector a vector fixed in the body.
270    /// @return the same vector expressed in world coordinates.
271    b2Vec2 GetWorldVector(const b2Vec2& localVector) const;
272
273    /// Gets a local point relative to the body's origin given a world point.
274    /// @param a point in world coordinates.
275    /// @return the corresponding local point relative to the body's origin.
276    b2Vec2 GetLocalPoint(const b2Vec2& worldPoint) const;
277
278    /// Gets a local vector given a world vector.
279    /// @param a vector in world coordinates.
280    /// @return the corresponding local vector.
281    b2Vec2 GetLocalVector(const b2Vec2& worldVector) const;
282
283    /// Get the world linear velocity of a world point attached to this body.
284    /// @param a point in world coordinates.
285    /// @return the world velocity of a point.
286    b2Vec2 GetLinearVelocityFromWorldPoint(const b2Vec2& worldPoint) const;
287
288    /// Get the world velocity of a local point.
289    /// @param a point in local coordinates.
290    /// @return the world velocity of a point.
291    b2Vec2 GetLinearVelocityFromLocalPoint(const b2Vec2& localPoint) const;
292
293    /// Get the linear damping of the body.
294    float32 GetLinearDamping() const;
295
296    /// Set the linear damping of the body.
297    void SetLinearDamping(float32 linearDamping);
298
299    /// Get the angular damping of the body.
300    float32 GetAngularDamping() const;
301
302    /// Set the angular damping of the body.
303    void SetAngularDamping(float32 angularDamping);
304
305    /// Get the gravity scale of the body.
306    float32 GetGravityScale() const;
307
308    /// Set the gravity scale of the body.
309    void SetGravityScale(float32 scale);
310
311    /// Set the type of this body. This may alter the mass and velocity.
312    void SetType(b2BodyType type);
313
```

```
314    /// Get the type of this body.
315    b2BodyType GetType() const;
316
317    /// Should this body be treated like a bullet for continuous collision detecti
    on?
318    void SetBullet(bool flag);
319
320    /// Is this body treated like a bullet for continuous collision detection?
321    bool IsBullet() const;
322
323    /// You can disable sleeping on this body. If you disable sleeping, the
324    /// body will be woken.
325    void SetSleepingAllowed(bool flag);
326
327    /// Is this body allowed to sleep
328    bool IsSleepingAllowed() const;
329
330    /// Set the sleep state of the body. A sleeping body has very
331    /// low CPU cost.
332    /// @param flag set to true to wake the body, false to put it to sleep.
333    void SetAwake(bool flag);
334
335    /// Get the sleeping state of this body.
336    /// @return true if the body is awake.
337    bool IsAwake() const;
338
339    /// Set the active state of the body. An inactive body is not
340    /// simulated and cannot be collided with or woken up.
341    /// If you pass a flag of true, all fixtures will be added to the
342    /// broad-phase.
343    /// If you pass a flag of false, all fixtures will be removed from
344    /// the broad-phase and all contacts will be destroyed.
345    /// Fixtures and joints are otherwise unaffected. You may continue
346    /// to create/destroy fixtures and joints on inactive bodies.
347    /// Fixtures on an inactive body are implicitly inactive and will
348    /// not participate in collisions, ray-casts, or queries.
349    /// Joints connected to an inactive body are implicitly inactive.
350    /// An inactive body is still owned by a b2World object and remains
351    /// in the body list.
352    void SetActive(bool flag);
353
354    /// Get the active state of the body.
355    bool IsActive() const;
356
357    /// Set this body to have fixed rotation. This causes the mass
358    /// to be reset.
359    void SetFixedRotation(bool flag);
360
361    /// Does this body have fixed rotation?
362    bool IsFixedRotation() const;
363
364    /// Get the list of all fixtures attached to this body.
365    b2Fixture* GetFixtureList();
366    const b2Fixture* GetFixtureList() const;
367
368    /// Get the list of all joints attached to this body.
369    b2JointEdge* GetJointList();
370    const b2JointEdge* GetJointList() const;
371
372    /// Get the list of all contacts attached to this body.
373    /// @warning this list changes during the time step and you may
374    /// miss some collisions if you don't use b2ContactListener.
375    b2ContactEdge* GetContactList();
376    const b2ContactEdge* GetContactList() const;
377
378    /// Get the next body in the world's body list.
```



```
379    b2Body* GetNext();
380    const b2Body* GetNext() const;
381
382    /// Get the user data pointer that was provided in the body definition.
383    Colisionable* GetUserData() const;
384
385    /// Set the user data. Use this to store your application specific data.
386    void SetUserData(Colisionable* data);
387
388    /// Get the parent world of this body.
389    b2World* GetWorld();
390    const b2World* GetWorld() const;
391
392    /// Dump this body to a log file
393    void Dump();
394
395  private:
396
397    friend class b2World;
398    friend class b2Island;
399    friend class b2ContactManager;
400    friend class b2ContactSolver;
401    friend class b2Contact;
402
403    friend class b2DistanceJoint;
404    friend class b2FrictionJoint;
405    friend class b2GearJoint;
406    friend class b2MotorJoint;
407    friend class b2MouseJoint;
408    friend class b2PrismaticJoint;
409    friend class b2PulleyJoint;
410    friend class b2RevoluteJoint;
411    friend class b2RopeJoint;
412    friend class b2WeldJoint;
413    friend class b2WheelJoint;
414
415    // m_flags
416    enum
417    {
418      e_islandFlag    = 0x0001,
419      e_awakeFlag     = 0x0002,
420      e_autoSleepFlag   = 0x0004,
421      e_bulletFlag    = 0x0008,
422      e_fixedRotationFlag = 0x0010,
423      e_activeFlag    = 0x0020,
424      e_toiFlag     = 0x0040
425    };
426
427    b2Body(const b2BodyDef* bd, b2World* world);
428    ~b2Body();
429
430    void SynchronizeFixtures();
431    void SynchronizeTransform();
432
433    // This is used to prevent connected bodies from colliding.
434    // It may lie, depending on the collideConnected flag.
435    bool ShouldCollide(const b2Body* other) const;
436
437    void Advance(float32 t);
438
439    b2BodyType m_type;
440
441    uint16 m_flags;
442
443    int32 m_islandIndex;
444
```

```
445   b2Transform m_xf;   // the body origin transform
446   b2Sweep m_sweep;    // the swept motion for CCD
447
448   b2Vec2 m_linearVelocity;
449   float32 m_angularVelocity;
450
451   b2Vec2 m_force;
452   float32 m_torque;
453
454   b2World* m_world;
455   b2Body* m_prev;
456   b2Body* m_next;
457
458   b2Fixture* m_fixtureList;
459   int32 m_fixtureCount;
460
461   b2JointEdge* m_jointList;
462   b2ContactEdge* m_contactList;
463
464   float32 m_mass, m_invMass;
465
466   // Rotational inertia about the center of mass.
467   float32 m_I, m_invI;
468
469   float32 m_linearDamping;
470   float32 m_angularDamping;
471   float32 m_gravityScale;
472
473   float32 m_sleepTime;
474
475   Colisionable* m_userData;
476 };
477
478 inline b2BodyType b2Body::GetType() const
479 {
480   return m_type;
481 }
482
483 inline const b2Transform& b2Body::GetTransform() const
484 {
485   return m_xf;
486 }
487
488 inline const b2Vec2& b2Body::GetPosition() const
489 {
490   return m_xf.p;
491 }
492
493 inline float32 b2Body::GetAngle() const
494 {
495   return m_sweep.a;
496 }
497
498 inline const b2Vec2& b2Body::GetWorldCenter() const
499 {
500   return m_sweep.c;
501 }
502
503 inline const b2Vec2& b2Body::GetLocalCenter() const
504 {
505   return m_sweep.localCenter;
506 }
507
508 inline void b2Body::SetLinearVelocity(const b2Vec2& v)
509 {
510   if (m_type ≡ b2_staticBody)
```



```
511   {
512     return;
513   }
514
515   if (b2Dot(v,v) > 0.0f)
516   {
517     SetAwake(true);
518   }
519
520   m_linearVelocity = v;
521 }
522
523 inline const b2Vec2& b2Body::GetLinearVelocity() const
524 {
525   return m_linearVelocity;
526 }
527
528 inline void b2Body::SetAngularVelocity(float32 w)
529 {
530   if (m_type ≡ b2_staticBody)
531   {
532     return;
533   }
534
535   if (w * w > 0.0f)
536   {
537     SetAwake(true);
538   }
539
540   m_angularVelocity = w;
541 }
542
543 inline float32 b2Body::GetAngularVelocity() const
544 {
545   return m_angularVelocity;
546 }
547
548 inline float32 b2Body::GetMass() const
549 {
550   return m_mass;
551 }
552
553 inline float32 b2Body::GetInertia() const
554 {
555   return m_I + m_mass * b2Dot(m_sweep.localCenter, m_sweep.localCenter);
556 }
557
558 inline void b2Body::GetMassData(b2MassData* data) const
559 {
560   data→mass = m_mass;
561   data→I = m_I + m_mass * b2Dot(m_sweep.localCenter, m_sweep.localCenter);
562   data→center = m_sweep.localCenter;
563 }
564
565 inline b2Vec2 b2Body::GetWorldPoint(const b2Vec2& localPoint) const
566 {
567   return b2Mul(m_xf, localPoint);
568 }
569
570 inline b2Vec2 b2Body::GetWorldVector(const b2Vec2& localVector) const
571 {
572   return b2Mul(m_xf.q, localVector);
573 }
574
575 inline b2Vec2 b2Body::GetLocalPoint(const b2Vec2& worldPoint) const
576 {
```

```
577   return b2MulT(m_xf, worldPoint);
578 }
579
580 inline b2Vec2 b2Body::GetLocalVector(const b2Vec2& worldVector) const
581 {
582   return b2MulT(m_xf.q, worldVector);
583 }
584
585 inline b2Vec2 b2Body::GetLinearVelocityFromWorldPoint(const b2Vec2& worldPoint)
    const
586 {
587   return m_linearVelocity + b2Cross(m_angularVelocity, worldPoint − m_sweep.c);
588 }
589
590 inline b2Vec2 b2Body::GetLinearVelocityFromLocalPoint(const b2Vec2& localPoint)
    const
591 {
592   return GetLinearVelocityFromWorldPoint(GetWorldPoint(localPoint));
593 }
594
595 inline float32 b2Body::GetLinearDamping() const
596 {
597   return m_linearDamping;
598 }
599
600 inline void b2Body::SetLinearDamping(float32 linearDamping)
601 {
602   m_linearDamping = linearDamping;
603 }
604
605 inline float32 b2Body::GetAngularDamping() const
606 {
607   return m_angularDamping;
608 }
609
610 inline void b2Body::SetAngularDamping(float32 angularDamping)
611 {
612   m_angularDamping = angularDamping;
613 }
614
615 inline float32 b2Body::GetGravityScale() const
616 {
617   return m_gravityScale;
618 }
619
620 inline void b2Body::SetGravityScale(float32 scale)
621 {
622   m_gravityScale = scale;
623 }
624
625 inline void b2Body::SetBullet(bool flag)
626 {
627   if (flag)
628   {
629     m_flags |= e_bulletFlag;
630   }
631   else
632   {
633     m_flags &= ~e_bulletFlag;
634   }
635 }
636
637 inline bool b2Body::IsBullet() const
638 {
639   return (m_flags & e_bulletFlag) ≡ e_bulletFlag;
640 }
```



```
641
642 inline void b2Body::SetAwake(bool flag)
643 {
644   if (flag)
645   {
646     m_flags |= e_awakeFlag;
647     m_sleepTime = 0.0f;
648   }
649   else
650   {
651     m_flags &= ~e_awakeFlag;
652     m_sleepTime = 0.0f;
653     m_linearVelocity.SetZero();
654     m_angularVelocity = 0.0f;
655     m_force.SetZero();
656     m_torque = 0.0f;
657   }
658 }
659
660 inline bool b2Body::IsAwake() const
661 {
662   return (m_flags & e_awakeFlag) ≡ e_awakeFlag;
663 }
664
665 inline bool b2Body::IsActive() const
666 {
667   return (m_flags & e_activeFlag) ≡ e_activeFlag;
668 }
669
670 inline bool b2Body::IsFixedRotation() const
671 {
672   return (m_flags & e_fixedRotationFlag) ≡ e_fixedRotationFlag;
673 }
674
675 inline void b2Body::SetSleepingAllowed(bool flag)
676 {
677   if (flag)
678   {
679     m_flags |= e_autoSleepFlag;
680   }
681   else
682   {
683     m_flags &= ~e_autoSleepFlag;
684     SetAwake(true);
685   }
686 }
687
688 inline bool b2Body::IsSleepingAllowed() const
689 {
690   return (m_flags & e_autoSleepFlag) ≡ e_autoSleepFlag;
691 }
692
693 inline b2Fixture* b2Body::GetFixtureList()
694 {
695   return m_fixtureList;
696 }
697
698 inline const b2Fixture* b2Body::GetFixtureList() const
699 {
700   return m_fixtureList;
701 }
702
703 inline b2JointEdge* b2Body::GetJointList()
704 {
705   return m_jointList;
706 }
```

```
707
708 inline const b2JointEdge* b2Body::GetJointList() const
709 {
710   return m_jointList;
711 }
712
713 inline b2ContactEdge* b2Body::GetContactList()
714 {
715   return m_contactList;
716 }
717
718 inline const b2ContactEdge* b2Body::GetContactList() const
719 {
720   return m_contactList;
721 }
722
723 inline b2Body* b2Body::GetNext()
724 {
725   return m_next;
726 }
727
728 inline const b2Body* b2Body::GetNext() const
729 {
730   return m_next;
731 }
732
733 inline void b2Body::SetUserData(Colisionable* data)
734 {
735   m_userData = data;
736 }
737
738 inline Colisionable* b2Body::GetUserData() const
739 {
740   return m_userData;
741 }
742
743 inline void b2Body::ApplyForce(const b2Vec2& force, const b2Vec2& point, bool wa
    ke)
744 {
745   if (m_type ≠ b2_dynamicBody)
746   {
747     return;
748   }
749
750   if (wake ∧ (m_flags & e_awakeFlag) ≡ 0)
751   {
752     SetAwake(true);
753   }
754
755   // Don't accumulate a force if the body is sleeping.
756   if (m_flags & e_awakeFlag)
757   {
758     m_force += force;
759     m_torque += b2Cross(point − m_sweep.c, force);
760   }
761 }
762
763 inline void b2Body::ApplyForceToCenter(const b2Vec2& force, bool wake)
764 {
765   if (m_type ≠ b2_dynamicBody)
766   {
767     return;
768   }
769
770   if (wake ∧ (m_flags & e_awakeFlag) ≡ 0)
771   {
```



```
772     SetAwake(true);
773   }
774
775   // Don't accumulate a force if the body is sleeping
776   if (m_flags & e_awakeFlag)
777   {
778     m_force += force;
779   }
780 }
781
782 inline void b2Body::ApplyTorque(float32 torque, bool wake)
783 {
784   if (m_type ≠ b2_dynamicBody)
785   {
786     return;
787   }
788
789   if (wake ∧ (m_flags & e_awakeFlag) ≡ 0)
790   {
791     SetAwake(true);
792   }
793
794   // Don't accumulate a force if the body is sleeping
795   if (m_flags & e_awakeFlag)
796   {
797     m_torque += torque;
798   }
799 }
800
801 inline void b2Body::ApplyLinearImpulse(const b2Vec2& impulse, const b2Vec2& poin
    t, bool wake)
802 {
803   if (m_type ≠ b2_dynamicBody)
804   {
805     return;
806   }
807
808   if (wake ∧ (m_flags & e_awakeFlag) ≡ 0)
809   {
810     SetAwake(true);
811   }
812
813   // Don't accumulate velocity if the body is sleeping
814   if (m_flags & e_awakeFlag)
815   {
816     m_linearVelocity += m_invMass * impulse;
817     m_angularVelocity += m_invI * b2Cross(point − m_sweep.c, impulse);
818   }
819 }
820
821 inline void b2Body::ApplyLinearImpulseToCenter(const b2Vec2& impulse, bool wake)
822 {
823   if (m_type ≠ b2_dynamicBody)
824   {
825     return;
826   }
827
828   if (wake ∧ (m_flags & e_awakeFlag) ≡ 0)
829   {
830     SetAwake(true);
831   }
832
833   // Don't accumulate velocity if the body is sleeping
834   if (m_flags & e_awakeFlag)
835   {
836     m_linearVelocity += m_invMass * impulse;
```

```
837   }
838 }
839
840 inline void b2Body::ApplyAngularImpulse(float32 impulse, bool wake)
841 {
842   if (m_type ≠ b2_dynamicBody)
843   {
844     return;
845   }
846
847   if (wake ∧ (m_flags & e_awakeFlag) ≡ 0)
848   {
849     SetAwake(true);
850   }
851
852   // Don't accumulate velocity if the body is sleeping
853   if (m_flags & e_awakeFlag)
854   {
855     m_angularVelocity += m_invI * impulse;
856   }
857 }
858
859 inline void b2Body::SynchronizeTransform()
860 {
861   m_xf.q.Set(m_sweep.a);
862   m_xf.p = m_sweep.c − b2Mul(m_xf.q, m_sweep.localCenter);
863 }
864
865 inline void b2Body::Advance(float32 alpha)
866 {
867   // Advance to the new safe time. This doesn't sync the broad−phase.
868   m_sweep.Advance(alpha);
869   m_sweep.c = m_sweep.c0;
870   m_sweep.a = m_sweep.a0;
871   m_xf.q.Set(m_sweep.a);
872   m_xf.p = m_sweep.c − b2Mul(m_xf.q, m_sweep.localCenter);
873 }
874
875 inline b2World* b2Body::GetWorld()
876 {
877   return m_world;
878 }
879
880 inline const b2World* b2Body::GetWorld() const
881 {
882   return m_world;
883 }
884
885 #endif
```



```
1  /*
2  * Copyright (c) 2011 Erin Catto http://box2d.org
3  *
4  * This software is provided 'as−is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #ifndef B2_TIMER_H
20 #define B2_TIMER_H
21
22 #include "Box2D/Common/b2Settings.h"
23
24 /// Timer for profiling. This has platform specific code and may
25 /// not work on every platform.
26 class b2Timer
27 {
28 public:
29
30   /// Constructor
31   b2Timer();
32
33   /// Reset the timer.
34   void Reset();
35
36   /// Get the time since construction or the last reset.
37   float32 GetMilliseconds() const;
38
39 private:
40
41 #if defined(_WIN32)
42   float64 m_start;
43   static float64 s_invFrequency;
44 #elif defined(__linux__) ∨ defined (__APPLE__)
45   unsigned long long m_start_sec;
46   unsigned long long m_start_usec;
47 #endif
48 };
49
50 #endif
```

nov 26, 19 17:34 **b2StackAllocator.h** Page 1/1

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_STACK_ALLOCATOR_H
#define B2_STACK_ALLOCATOR_H

#include "Box2D/Common/b2Settings.h"

const int32 b2_stackSize = 100 * 1024;  // 100k
const int32 b2_maxStackEntries = 32;

struct b2StackEntry
{
  char* data;
  int32 size;
  bool usedMalloc;
};

// This is a stack allocator used for fast per step allocations.
// You must nest allocate/free pairs. The code will assert
// if you try to interleave multiple allocate/free pairs.
class b2StackAllocator
{
public:
  b2StackAllocator();
  ~b2StackAllocator();

  void* Allocate(int32 size);
  void Free(void* p);

  int32 GetMaxAllocation() const;

private:

  char m_data[b2_stackSize];
  int32 m_index;

  int32 m_allocation;
  int32 m_maxAllocation;

  b2StackEntry m_entries[b2_maxStackEntries];
  int32 m_entryCount;
};

#endif
```

nov 26, 19 17:34 **b2Settings.h** Page 1/3

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_SETTINGS_H
#define B2_SETTINGS_H

#include <stddef.h>
#include <assert.h>
#include <float.h>

#if ¬defined(NDEBUG)
  #define b2DEBUG
#endif

#define B2_NOT_USED(x) ((void)(x))
#define b2Assert(A) assert(A)

typedef signed char int8;
typedef signed short int16;
typedef signed int int32;
typedef unsigned char uint8;
typedef unsigned short uint16;
typedef unsigned int uint32;
typedef float float32;
typedef double float64;

#define b2_maxFloat   FLT_MAX
#define b2_epsilon    FLT_EPSILON
#define b2_pi     3.14159265359f

/// @file
/// Global tuning constants based on meters-kilograms-seconds (MKS) units.
///

// Collision

/// The maximum number of contact points between two convex shapes. Do
/// not change this value.
#define b2_maxManifoldPoints  2

/// The maximum number of vertices on a convex polygon. You cannot increase
/// this too much because b2BlockAllocator has a maximum object size.
#define b2_maxPolygonVertices 8

/// This is used to fatten AABBs in the dynamic tree. This allows proxies
/// to move by a small amount without triggering a tree adjustment.
/// This is in meters.
#define b2_aabbExtension    0.1f

/// This is used to fatten AABBs in the dynamic tree. This is used to predict
/// the future position based on the current displacement.
```

```
67  /// This is a dimensionless multiplier.
68  #define b2_aabbMultiplier   2.0f
69
70  /// A small length used as a collision and constraint tolerance. Usually it is
71  /// chosen to be numerically significant, but visually insignificant.
72  #define b2_linearSlop     0.005f
73
74  /// A small angle used as a collision and constraint tolerance. Usually it is
75  /// chosen to be numerically significant, but visually insignificant.
76  #define b2_angularSlop      (2.0f / 180.0f * b2_pi)
77
78  /// The radius of the polygon/edge shape skin. This should not be modified. Maki
    ng
79  /// this smaller means polygons will have an insufficient buffer for continuous
    collision.
80  /// Making it larger may create artifacts for vertex collision.
81  #define b2_polygonRadius    (2.0f * b2_linearSlop)
82
83  /// Maximum number of sub-steps per contact in continuous physics simulation.
84  #define b2_maxSubSteps      8
85
86
87  // Dynamics
88
89  /// Maximum number of contacts to be handled to solve a TOI impact.
90  #define b2_maxTOIContacts     32
91
92  /// A velocity threshold for elastic collisions. Any collision with a relative l
    inear
93  /// velocity below this threshold will be treated as inelastic.
94  #define b2_velocityThreshold    1.0f
95
96  /// The maximum linear position correction used when solving constraints. This h
    elps to
97  /// prevent overshoot.
98  #define b2_maxLinearCorrection    0.2f
99
100 /// The maximum angular position correction used when solving constraints. This
    helps to
101 /// prevent overshoot.
102 #define b2_maxAngularCorrection   (8.0f / 180.0f * b2_pi)
103
104 /// The maximum linear velocity of a body. This limit is very large and is used
105 /// to prevent numerical problems. You shouldn't need to adjust this.
106 #define b2_maxTranslation     2.0f
107 #define b2_maxTranslationSquared  (b2_maxTranslation * b2_maxTranslation)
108
109 /// The maximum angular velocity of a body. This limit is very large and is used
110 /// to prevent numerical problems. You shouldn't need to adjust this.
111 #define b2_maxRotation        (0.5f * b2_pi)
112 #define b2_maxRotationSquared   (b2_maxRotation * b2_maxRotation)
113
114 /// This scale factor controls how fast overlap is resolved. Ideally this would
    be 1 so
115 /// that overlap is removed in one time step. However using values close to 1 of
    ten lead
116 /// to overshoot.
117 #define b2_baumgarte        0.2f
118 #define b2_toiBaugarte        0.75f
119
120
121 // Sleep
122
123 /// The time that a body must be still before it will go to sleep.
124 #define b2_timeToSleep        0.5f
125
```



```
126 /// A body cannot sleep if its linear velocity is above this tolerance.
127 #define b2_linearSleepTolerance   0.01f
128
129 /// A body cannot sleep if its angular velocity is above this tolerance.
130 #define b2_angularSleepTolerance  (2.0f / 180.0f * b2_pi)
131
132 // Memory Allocation
133
134 /// Implement this function to use your own memory allocator.
135 void* b2Alloc(int32 size);
136
137 /// If you implement b2Alloc, you should also implement this function.
138 void b2Free(void* mem);
139
140 /// Logging function.
141 void b2Log(const char* string, ...);
142
143 /// Version numbering scheme.
144 /// See http://en.wikipedia.org/wiki/Software_versioning
145 struct b2Version
146 {
147   int32 major;    ///< significant changes
148   int32 minor;    ///< incremental changes
149   int32 revision;   ///< bug fixes
150 };
151
152 /// Current version.
153 extern b2Version b2_version;
154
155 #endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_MATH_H
#define B2_MATH_H

#include "Box2D/Common/b2Settings.h"
#include <math.h>

/// This function is used to ensure that a floating point number is not a NaN or
 infinity.
inline bool b2IsValid(float32 x)
{
  return isfinite(x);
}

#define b2Sqrt(x) sqrtf(x)
#define b2Atan2(y, x) atan2f(y, x)

/// A 2D column vector.
struct b2Vec2
{
  /// Default constructor does nothing (for performance).
  b2Vec2() {}

  /// Construct using coordinates.
  b2Vec2(float32 xIn, float32 yIn) : x(xIn), y(yIn) {}

  /// Set this vector to all zeros.
  void SetZero() { x = 0.0f; y = 0.0f; }

  /// Set this vector to some specified coordinates.
  void Set(float32 x_, float32 y_) { x = x_; y = y_; }

  /// Negate this vector.
  b2Vec2 operator -() const { b2Vec2 v; v.Set(-x, -y); return v; }

  /// Read from and indexed element.
  float32 operator () (int32 i) const
  {
    return (&x)[i];
  }

  /// Write to an indexed element.
  float32& operator () (int32 i)
  {
    return (&x)[i];
  }

  /// Add a vector to this vector.
  void operator += (const b2Vec2& v)
```



```
  {
    x += v.x; y += v.y;
  }

  /// Subtract a vector from this vector.
  void operator -= (const b2Vec2& v)
  {
    x -= v.x; y -= v.y;
  }

  /// Multiply this vector by a scalar.
  void operator *= (float32 a)
  {
    x *= a; y *= a;
  }

  /// Get the length of this vector (the norm).
  float32 Length() const
  {
    return b2Sqrt(x * x + y * y);
  }

  /// Get the length squared. For performance, use this instead of
  /// b2Vec2::Length (if possible).
  float32 LengthSquared() const
  {
    return x * x + y * y;
  }

  /// Convert this vector into a unit vector. Returns the length.
  float32 Normalize()
  {
    float32 length = Length();
    if (length < b2_epsilon)
    {
      return 0.0f;
    }
    float32 invLength = 1.0f / length;
    x *= invLength;
    y *= invLength;

    return length;
  }

  /// Does this vector contain finite coordinates?
  bool IsValid() const
  {
    return b2IsValid(x) ∧ b2IsValid(y);
  }

  /// Get the skew vector such that dot(skew_vec, other) == cross(vec, other)
  b2Vec2 Skew() const
  {
    return b2Vec2(-y, x);
  }

  float32 x, y;
};

/// A 2D column vector with 3 elements.
struct b2Vec3
{
  /// Default constructor does nothing (for performance).
  b2Vec3() {}

  /// Construct using coordinates.
```

```
132   b2Vec3(float32 xIn, float32 yIn, float32 zIn) : x(xIn), y(yIn), z(zIn) {}
133
134   /// Set this vector to all zeros.
135   void SetZero() { x = 0.0f; y = 0.0f; z = 0.0f; }
136
137   /// Set this vector to some specified coordinates.
138   void Set(float32 x_, float32 y_, float32 z_) { x = x_; y = y_; z = z_; }
139
140   /// Negate this vector.
141   b2Vec3 operator -() const { b2Vec3 v; v.Set(-x, -y, -z); return v; }
142
143   /// Add a vector to this vector.
144   void operator += (const b2Vec3& v)
145   {
146     x += v.x; y += v.y; z += v.z;
147   }
148
149   /// Subtract a vector from this vector.
150   void operator -= (const b2Vec3& v)
151   {
152     x -= v.x; y -= v.y; z -= v.z;
153   }
154
155   /// Multiply this vector by a scalar.
156   void operator *= (float32 s)
157   {
158     x *= s; y *= s; z *= s;
159   }
160
161   float32 x, y, z;
162 };
163
164 /// A 2-by-2 matrix. Stored in column-major order.
165 struct b2Mat22
166 {
167   /// The default constructor does nothing (for performance).
168   b2Mat22() {}
169
170   /// Construct this matrix using columns.
171   b2Mat22(const b2Vec2& c1, const b2Vec2& c2)
172   {
173     ex = c1;
174     ey = c2;
175   }
176
177   /// Construct this matrix using scalars.
178   b2Mat22(float32 a11, float32 a12, float32 a21, float32 a22)
179   {
180     ex.x = a11; ex.y = a21;
181     ey.x = a12; ey.y = a22;
182   }
183
184   /// Initialize this matrix using columns.
185   void Set(const b2Vec2& c1, const b2Vec2& c2)
186   {
187     ex = c1;
188     ey = c2;
189   }
190
191   /// Set this to the identity matrix.
192   void SetIdentity()
193   {
194     ex.x = 1.0f; ey.x = 0.0f;
195     ex.y = 0.0f; ey.y = 1.0f;
196   }
197
```



```
198   /// Set this matrix to all zeros.
199   void SetZero()
200   {
201     ex.x = 0.0f; ey.x = 0.0f;
202     ex.y = 0.0f; ey.y = 0.0f;
203   }
204
205   b2Mat22 GetInverse() const
206   {
207     float32 a = ex.x, b = ey.x, c = ex.y, d = ey.y;
208     b2Mat22 B;
209     float32 det = a * d - b * c;
210     if (det ≠ 0.0f)
211     {
212       det = 1.0f / det;
213     }
214     B.ex.x =  det * d;  B.ey.x = -det * b;
215     B.ex.y = -det * c;  B.ey.y =  det * a;
216     return B;
217   }
218
219   /// Solve A * x = b, where b is a column vector. This is more efficient
220   /// than computing the inverse in one-shot cases.
221   b2Vec2 Solve(const b2Vec2& b) const
222   {
223     float32 a11 = ex.x, a12 = ey.x, a21 = ex.y, a22 = ey.y;
224     float32 det = a11 * a22 - a12 * a21;
225     if (det ≠ 0.0f)
226     {
227       det = 1.0f / det;
228     }
229     b2Vec2 x;
230     x.x = det * (a22 * b.x - a12 * b.y);
231     x.y = det * (a11 * b.y - a21 * b.x);
232     return x;
233   }
234
235   b2Vec2 ex, ey;
236 };
237
238 /// A 3-by-3 matrix. Stored in column-major order.
239 struct b2Mat33
240 {
241   /// The default constructor does nothing (for performance).
242   b2Mat33() {}
243
244   /// Construct this matrix using columns.
245   b2Mat33(const b2Vec3& c1, const b2Vec3& c2, const b2Vec3& c3)
246   {
247     ex = c1;
248     ey = c2;
249     ez = c3;
250   }
251
252   /// Set this matrix to all zeros.
253   void SetZero()
254   {
255     ex.SetZero();
256     ey.SetZero();
257     ez.SetZero();
258   }
259
260   /// Solve A * x = b, where b is a column vector. This is more efficient
261   /// than computing the inverse in one-shot cases.
262   b2Vec3 Solve33(const b2Vec3& b) const;
263
```

```
264   /// Solve A * x = b, where b is a column vector. This is more efficient
265   /// than computing the inverse in one-shot cases. Solve only the upper
266   /// 2-by-2 matrix equation.
267   b2Vec2 Solve22(const b2Vec2& b) const;
268
269   /// Get the inverse of this matrix as a 2-by-2.
270   /// Returns the zero matrix if singular.
271   void GetInverse22(b2Mat33* M) const;
272
273   /// Get the symmetric inverse of this matrix as a 3-by-3.
274   /// Returns the zero matrix if singular.
275   void GetSymInverse33(b2Mat33* M) const;
276
277   b2Vec3 ex, ey, ez;
278 };
279
280 /// Rotation
281 struct b2Rot
282 {
283   b2Rot() {}
284
285   /// Initialize from an angle in radians
286   explicit b2Rot(float32 angle)
287   {
288
289     s = sinf(angle);
290     c = cosf(angle);
291   }
292
293   /// Set using an angle in radians.
294   void Set(float32 angle)
295   {
296
297     s = sinf(angle);
298     c = cosf(angle);
299   }
300
301   /// Set to the identity rotation
302   void SetIdentity()
303   {
304     s = 0.0f;
305     c = 1.0f;
306   }
307
308   /// Get the angle in radians
309   float32 GetAngle() const
310   {
311     return b2Atan2(s, c);
312   }
313
314   /// Get the x-axis
315   b2Vec2 GetXAxis() const
316   {
317     return b2Vec2(c, s);
318   }
319
320   /// Get the u-axis
321   b2Vec2 GetYAxis() const
322   {
323     return b2Vec2(-s, c);
324   }
325
326   /// Sine and cosine
327   float32 s, c;
328 };
329
```



```
330 /// A transform contains translation and rotation. It is used to represent
331 /// the position and orientation of rigid frames.
332 struct b2Transform
333 {
334   /// The default constructor does nothing.
335   b2Transform() {}
336
337   /// Initialize using a position vector and a rotation.
338   b2Transform(const b2Vec2& position, const b2Rot& rotation) : p(position), q(ro
    tation) {}
339
340   /// Set this to the identity transform.
341   void SetIdentity()
342   {
343     p.SetZero();
344     q.SetIdentity();
345   }
346
347   /// Set this based on the position and angle.
348   void Set(const b2Vec2& position, float32 angle)
349   {
350     p = position;
351     q.Set(angle);
352   }
353
354   b2Vec2 p;
355   b2Rot q;
356 };
357
358 /// This describes the motion of a body/shape for TOI computation.
359 /// Shapes are defined with respect to the body origin, which may
360 /// no coincide with the center of mass. However, to support dynamics
361 /// we must interpolate the center of mass position.
362 struct b2Sweep
363 {
364   /// Get the interpolated transform at a specific time.
365   /// @param beta is a factor in [0,1], where 0 indicates alpha0.
366   void GetTransform(b2Transform* xfb, float32 beta) const;
367
368   /// Advance the sweep forward, yielding a new initial state.
369   /// @param alpha the new initial time.
370   void Advance(float32 alpha);
371
372   /// Normalize the angles.
373   void Normalize();
374
375   b2Vec2 localCenter; ///< local center of mass position
376   b2Vec2 c0, c;   ///< center world positions
377   float32 a0, a;    ///< world angles
378
379   /// Fraction of the current time step in the range [0,1]
380   /// c0 and a0 are the positions at alpha0.
381   float32 alpha0;
382 };
383
384 /// Useful constant
385 extern const b2Vec2 b2Vec2_zero;
386
387 /// Perform the dot product on two vectors.
388 inline float32 b2Dot(const b2Vec2& a, const b2Vec2& b)
389 {
390   return a.x * b.x + a.y * b.y;
391 }
392
393 /// Perform the cross product on two vectors. In 2D this produces a scalar.
394 inline float32 b2Cross(const b2Vec2& a, const b2Vec2& b)
```

```
395 {
396   return a.x * b.y − a.y * b.x;
397 }
398
399 /// Perform the cross product on a vector and a scalar. In 2D this produces
400 /// a vector.
401 inline b2Vec2 b2Cross(const b2Vec2& a, float32 s)
402 {
403   return b2Vec2(s * a.y, −s * a.x);
404 }
405
406 /// Perform the cross product on a scalar and a vector. In 2D this produces
407 /// a vector.
408 inline b2Vec2 b2Cross(float32 s, const b2Vec2& a)
409 {
410   return b2Vec2(−s * a.y, s * a.x);
411 }
412
413 /// Multiply a matrix times a vector. If a rotation matrix is provided,
414 /// then this transforms the vector from one frame to another.
415 inline b2Vec2 b2Mul(const b2Mat22& A, const b2Vec2& v)
416 {
417   return b2Vec2(A.ex.x * v.x + A.ey.x * v.y, A.ex.y * v.x + A.ey.y * v.y);
418 }
419
420 /// Multiply a matrix transpose times a vector. If a rotation matrix is provided
    ,
421 /// then this transforms the vector from one frame to another (inverse transform
    ).
422 inline b2Vec2 b2MulT(const b2Mat22& A, const b2Vec2& v)
423 {
424   return b2Vec2(b2Dot(v, A.ex), b2Dot(v, A.ey));
425 }
426
427 /// Add two vectors component−wise.
428 inline b2Vec2 operator + (const b2Vec2& a, const b2Vec2& b)
429 {
430   return b2Vec2(a.x + b.x, a.y + b.y);
431 }
432
433 /// Subtract two vectors component−wise.
434 inline b2Vec2 operator − (const b2Vec2& a, const b2Vec2& b)
435 {
436   return b2Vec2(a.x − b.x, a.y − b.y);
437 }
438
439 inline b2Vec2 operator * (float32 s, const b2Vec2& a)
440 {
441   return b2Vec2(s * a.x, s * a.y);
442 }
443
444 inline bool operator ≡ (const b2Vec2& a, const b2Vec2& b)
445 {
446   return a.x ≡ b.x ∧ a.y ≡ b.y;
447 }
448
449 inline bool operator ≠ (const b2Vec2& a, const b2Vec2& b)
450 {
451   return a.x ≠ b.x ∨ a.y ≠ b.y;
452 }
453
454 inline float32 b2Distance(const b2Vec2& a, const b2Vec2& b)
455 {
456   b2Vec2 c = a − b;
457   return c.Length();
458 }
```



```
459
460 inline float32 b2DistanceSquared(const b2Vec2& a, const b2Vec2& b)
461 {
462   b2Vec2 c = a − b;
463   return b2Dot(c, c);
464 }
465
466 inline b2Vec3 operator * (float32 s, const b2Vec3& a)
467 {
468   return b2Vec3(s * a.x, s * a.y, s * a.z);
469 }
470
471 /// Add two vectors component−wise.
472 inline b2Vec3 operator + (const b2Vec3& a, const b2Vec3& b)
473 {
474   return b2Vec3(a.x + b.x, a.y + b.y, a.z + b.z);
475 }
476
477 /// Subtract two vectors component−wise.
478 inline b2Vec3 operator − (const b2Vec3& a, const b2Vec3& b)
479 {
480   return b2Vec3(a.x − b.x, a.y − b.y, a.z − b.z);
481 }
482
483 /// Perform the dot product on two vectors.
484 inline float32 b2Dot(const b2Vec3& a, const b2Vec3& b)
485 {
486   return a.x * b.x + a.y * b.y + a.z * b.z;
487 }
488
489 /// Perform the cross product on two vectors.
490 inline b2Vec3 b2Cross(const b2Vec3& a, const b2Vec3& b)
491 {
492   return b2Vec3(a.y * b.z − a.z * b.y, a.z * b.x − a.x * b.z, a.x * b.y − a.y *
    b.x);
493 }
494
495 inline b2Mat22 operator + (const b2Mat22& A, const b2Mat22& B)
496 {
497   return b2Mat22(A.ex + B.ex, A.ey + B.ey);
498 }
499
500 // A * B
501 inline b2Mat22 b2Mul(const b2Mat22& A, const b2Mat22& B)
502 {
503   return b2Mat22(b2Mul(A, B.ex), b2Mul(A, B.ey));
504 }
505
506 // A^T * B
507 inline b2Mat22 b2MulT(const b2Mat22& A, const b2Mat22& B)
508 {
509   b2Vec2 c1(b2Dot(A.ex, B.ex), b2Dot(A.ey, B.ex));
510   b2Vec2 c2(b2Dot(A.ex, B.ey), b2Dot(A.ey, B.ey));
511   return b2Mat22(c1, c2);
512 }
513
514 /// Multiply a matrix times a vector.
515 inline b2Vec3 b2Mul(const b2Mat33& A, const b2Vec3& v)
516 {
517   return v.x * A.ex + v.y * A.ey + v.z * A.ez;
518 }
519
520 /// Multiply a matrix times a vector.
521 inline b2Vec2 b2Mul22(const b2Mat33& A, const b2Vec2& v)
522 {
523   return b2Vec2(A.ex.x * v.x + A.ey.x * v.y, A.ex.y * v.x + A.ey.y * v.y);
```

```
}

/// Multiply two rotations: q * r
inline b2Rot b2Mul(const b2Rot& q, const b2Rot& r)
{
  // [qc −qs] * [rc −rs] = [qc*rc−qs*rs −qc*rs−qs*rc]
  // [qs  qc]   [rs  rc]   [qs*rc+qc*rs −qs*rs+qc*rc]
  // s = qs * rc + qc * rs
  // c = qc * rc − qs * rs
  b2Rot qr;
  qr.s = q.s * r.c + q.c * r.s;
  qr.c = q.c * r.c − q.s * r.s;
  return qr;
}

/// Transpose multiply two rotations: qT * r
inline b2Rot b2MulT(const b2Rot& q, const b2Rot& r)
{
  // [ qc qs] * [rc −rs] = [qc*rc+qs*rs −qc*rs+qs*rc]
  // [−qs qc]   [rs  rc]   [−qs*rc+qc*rs qs*rs+qc*rc]
  // s = qc * rs − qs * rc
  // c = qc * rc + qs * rs
  b2Rot qr;
  qr.s = q.c * r.s − q.s * r.c;
  qr.c = q.c * r.c + q.s * r.s;
  return qr;
}

/// Rotate a vector
inline b2Vec2 b2Mul(const b2Rot& q, const b2Vec2& v)
{
  return b2Vec2(q.c * v.x − q.s * v.y, q.s * v.x + q.c * v.y);
}

/// Inverse rotate a vector
inline b2Vec2 b2MulT(const b2Rot& q, const b2Vec2& v)
{
  return b2Vec2(q.c * v.x + q.s * v.y, −q.s * v.x + q.c * v.y);
}

inline b2Vec2 b2Mul(const b2Transform& T, const b2Vec2& v)
{
  float32 x = (T.q.c * v.x − T.q.s * v.y) + T.p.x;
  float32 y = (T.q.s * v.x + T.q.c * v.y) + T.p.y;

  return b2Vec2(x, y);
}

inline b2Vec2 b2MulT(const b2Transform& T, const b2Vec2& v)
{
  float32 px = v.x − T.p.x;
  float32 py = v.y − T.p.y;
  float32 x = (T.q.c * px + T.q.s * py);
  float32 y = (−T.q.s * px + T.q.c * py);

  return b2Vec2(x, y);
}

// v2 = A.q.Rot(B.q.Rot(v1) + B.p) + A.p
//    = (A.q * B.q).Rot(v1) + A.q.Rot(B.p) + A.p
inline b2Transform b2Mul(const b2Transform& A, const b2Transform& B)
{
  b2Transform C;
  C.q = b2Mul(A.q, B.q);
  C.p = b2Mul(A.q, B.p) + A.p;
  return C;
```



```
}

// v2 = A.q' * (B.q * v1 + B.p − A.p)
//    = A.q' * B.q * v1 + A.q' * (B.p − A.p)
inline b2Transform b2MulT(const b2Transform& A, const b2Transform& B)
{
  b2Transform C;
  C.q = b2MulT(A.q, B.q);
  C.p = b2MulT(A.q, B.p − A.p);
  return C;
}

template <typename T>
inline T b2Abs(T a)
{
  return a > T(0) ? a : −a;
}

inline b2Vec2 b2Abs(const b2Vec2& a)
{
  return b2Vec2(b2Abs(a.x), b2Abs(a.y));
}

inline b2Mat22 b2Abs(const b2Mat22& A)
{
  return b2Mat22(b2Abs(A.ex), b2Abs(A.ey));
}

template <typename T>
inline T b2Min(T a, T b)
{
  return a < b ? a : b;
}

inline b2Vec2 b2Min(const b2Vec2& a, const b2Vec2& b)
{
  return b2Vec2(b2Min(a.x, b.x), b2Min(a.y, b.y));
}

template <typename T>
inline T b2Max(T a, T b)
{
  return a > b ? a : b;
}

inline b2Vec2 b2Max(const b2Vec2& a, const b2Vec2& b)
{
  return b2Vec2(b2Max(a.x, b.x), b2Max(a.y, b.y));
}

template <typename T>
inline T b2Clamp(T a, T low, T high)
{
  return b2Max(low, b2Min(a, high));
}

inline b2Vec2 b2Clamp(const b2Vec2& a, const b2Vec2& low, const b2Vec2& high)
{
  return b2Max(low, b2Min(a, high));
}

template<typename T> inline void b2Swap(T& a, T& b)
{
  T tmp = a;
  a = b;
  b = tmp;
```

```
656 }
657
658 /// "Next Largest Power of 2
659 /// Given a binary integer value x, the next largest power of 2 can be computed
    by a SWAR algorithm
660 /// that recursively "folds" the upper bits into the lower bits. This process yi
    elds a bit vector with
661 /// the same most significant 1 as x, but all 1's below it. Adding 1 to that val
    ue yields the next
662 /// largest power of 2. For a 32-bit value:"
663 inline uint32 b2NextPowerOfTwo(uint32 x)
664 {
665   x |= (x >> 1);
666   x |= (x >> 2);
667   x |= (x >> 4);
668   x |= (x >> 8);
669   x |= (x >> 16);
670   return x + 1;
671 }
672
673 inline bool b2IsPowerOfTwo(uint32 x)
674 {
675   bool result = x > 0 ∧ (x & (x − 1)) ≡ 0;
676   return result;
677 }
678
679 inline void b2Sweep::GetTransform(b2Transform* xf, float32 beta) const
680 {
681   xf→p = (1.0f − beta) * c0 + beta * c;
682   float32 angle = (1.0f − beta) * a0 + beta * a;
683   xf→q.Set(angle);
684
685   // Shift to origin
686   xf→p −= b2Mul(xf→q, localCenter);
687 }
688
689 inline void b2Sweep::Advance(float32 alpha)
690 {
691   b2Assert(alpha0 < 1.0f);
692   float32 beta = (alpha − alpha0) / (1.0f − alpha0);
693   c0 += beta * (c − c0);
694   a0 += beta * (a − a0);
695   alpha0 = alpha;
696 }
697
698 /// Normalize an angle in radians to be between −pi and pi
699 inline void b2Sweep::Normalize()
700 {
701   float32 twoPi = 2.0f * b2_pi;
702   float32 d =  twoPi * floorf(a0 / twoPi);
703   a0 −= d;
704   a −= d;
705 }
706
707 #endif
```



```
1  /*
2  * Copyright (c) 2010 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as-is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #ifndef B2_GROWABLE_STACK_H
20 #define B2_GROWABLE_STACK_H
21 #include "Box2D/Common/b2Settings.h"
22 #include <string.h>
23
24 /// This is a growable LIFO stack with an initial capacity of N.
25 /// If the stack size exceeds the initial capacity, the heap is used
26 /// to increase the size of the stack.
27 template <typename T, int32 N>
28 class b2GrowableStack
29 {
30 public:
31   b2GrowableStack()
32   {
33     m_stack = m_array;
34     m_count = 0;
35     m_capacity = N;
36   }
37
38   ~b2GrowableStack()
39   {
40     if (m_stack ≠ m_array)
41     {
42       b2Free(m_stack);
43       m_stack = nullptr;
44     }
45   }
46
47   void Push(const T& element)
48   {
49     if (m_count ≡ m_capacity)
50     {
51       T* old = m_stack;
52       m_capacity *= 2;
53       m_stack = (T*)b2Alloc(m_capacity * sizeof(T));
54       memcpy(m_stack, old, m_count * sizeof(T));
55       if (old ≠ m_array)
56       {
57         b2Free(old);
58       }
59     }
60
61     m_stack[m_count] = element;
62     ++m_count;
63   }
64
65   T Pop()
66   {
```

```
67      b2Assert(m_count > 0);
68      --m_count;
69      return m_stack[m_count];
70    }
71
72    int32 GetCount()
73    {
74      return m_count;
75    }
76
77  private:
78    T* m_stack;
79    T m_array[N];
80    int32 m_count;
81    int32 m_capacity;
82  };
83
84
85  #endif
```



```
1   /*
2   * Copyright (c) 2011 Erin Catto http://box2d.org
3   *
4   * This software is provided 'as-is', without any express or implied
5   * warranty.  In no event will the authors be held liable for any damages
6   * arising from the use of this software.
7   * Permission is granted to anyone to use this software for any purpose,
8   * including commercial applications, and to alter it and redistribute it
9   * freely, subject to the following restrictions:
10  * 1. The origin of this software must not be misrepresented; you must not
11  * claim that you wrote the original software. If you use this software
12  * in a product, an acknowledgment in the product documentation would be
13  * appreciated but is not required.
14  * 2. Altered source versions must be plainly marked as such, and must not be
15  * misrepresented as being the original software.
16  * 3. This notice may not be removed or altered from any source distribution.
17  */
18
19  #ifndef B2_DRAW_H
20  #define B2_DRAW_H
21
22  #include "Box2D/Common/b2Math.h"
23
24  /// Color for debug drawing. Each value has the range [0,1].
25  struct b2Color
26  {
27    b2Color() {}
28    b2Color(float32 rIn, float32 gIn, float32 bIn, float32 aIn = 1.0f)
29    {
30      r = rIn; g = gIn; b = bIn; a = aIn;
31    }
32
33    void Set(float32 rIn, float32 gIn, float32 bIn, float32 aIn = 1.0f)
34    {
35      r = rIn; g = gIn; b = bIn; a = aIn;
36    }
37
38    float32 r, g, b, a;
39  };
40
41  /// Implement and register this class with a b2World to provide debug drawing of
     physics
42  /// entities in your game.
43  class b2Draw
44  {
45  public:
46    b2Draw();
47
48    virtual ~b2Draw() {}
49
50    enum
51    {
52      e_shapeBit        = 0x0001, ///< draw shapes
53      e_jointBit        = 0x0002, ///< draw joint connections
54      e_aabbBit       = 0x0004, ///< draw axis aligned bounding boxes
55      e_pairBit       = 0x0008, ///< draw broad-phase pairs
56      e_centerOfMassBit   = 0x0010  ///< draw center of mass frame
57    };
58
59    /// Set the drawing flags.
60    void SetFlags(uint32 flags);
61
62    /// Get the drawing flags.
63    uint32 GetFlags() const;
64
65    /// Append flags to the current flags.
```

```
66    void AppendFlags(uint32 flags);
67
68    /// Clear flags from the current flags.
69    void ClearFlags(uint32 flags);
70
71    /// Draw a closed polygon provided in CCW order.
72    virtual void DrawPolygon(const b2Vec2* vertices, int32 vertexCount, const b2Co
      lor& color) = 0;
73
74    /// Draw a solid closed polygon provided in CCW order.
75    virtual void DrawSolidPolygon(const b2Vec2* vertices, int32 vertexCount, const
       b2Color& color) = 0;
76
77    /// Draw a circle.
78    virtual void DrawCircle(const b2Vec2& center, float32 radius, const b2Color& c
      olor) = 0;
79
80    /// Draw a solid circle.
81    virtual void DrawSolidCircle(const b2Vec2& center, float32 radius, const b2Vec
      2& axis, const b2Color& color) = 0;
82
83    /// Draw a line segment.
84    virtual void DrawSegment(const b2Vec2& p1, const b2Vec2& p2, const b2Color& co
      lor) = 0;
85
86    /// Draw a transform. Choose your own length scale.
87    /// @param xf a transform.
88    virtual void DrawTransform(const b2Transform& xf) = 0;
89
90    /// Draw a point.
91    virtual void DrawPoint(const b2Vec2& p, float32 size, const b2Color& color) =
      0;
92
93  protected:
94    uint32 m_drawFlags;
95  };
96
97  #endif
```



```
1   /*
2   * Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
3   *
4   * This software is provided 'as-is', without any express or implied
5   * warranty.  In no event will the authors be held liable for any damages
6   * arising from the use of this software.
7   * Permission is granted to anyone to use this software for any purpose,
8   * including commercial applications, and to alter it and redistribute it
9   * freely, subject to the following restrictions:
10  * 1. The origin of this software must not be misrepresented; you must not
11  * claim that you wrote the original software. If you use this software
12  * in a product, an acknowledgment in the product documentation would be
13  * appreciated but is not required.
14  * 2. Altered source versions must be plainly marked as such, and must not be
15  * misrepresented as being the original software.
16  * 3. This notice may not be removed or altered from any source distribution.
17  */
18
19  #ifndef B2_BLOCK_ALLOCATOR_H
20  #define B2_BLOCK_ALLOCATOR_H
21
22  #include "Box2D/Common/b2Settings.h"
23
24  const int32 b2_chunkSize = 16 * 1024;
25  const int32 b2_maxBlockSize = 640;
26  const int32 b2_blockSizes = 14;
27  const int32 b2_chunkArrayIncrement = 128;
28
29  struct b2Block;
30  struct b2Chunk;
31
32  /// This is a small object allocator used for allocating small
33  /// objects that persist for more than one time step.
34  /// See: http://www.codeproject.com/useritems/Small_Block_Allocator.asp
35  class b2BlockAllocator
36  {
37  public:
38    b2BlockAllocator();
39    ~b2BlockAllocator();
40
41    /// Allocate memory. This will use b2Alloc if the size is larger than b2_maxBl
      ockSize.
42    void* Allocate(int32 size);
43
44    /// Free memory. This will use b2Free if the size is larger than b2_maxBlockSi
      ze.
45    void Free(void* p, int32 size);
46
47    void Clear();
48
49  private:
50
51    b2Chunk* m_chunks;
52    int32 m_chunkCount;
53    int32 m_chunkSpace;
54
55    b2Block* m_freeLists[b2_blockSizes];
56
57    static int32 s_blockSizes[b2_blockSizes];
58    static uint8 s_blockSizeLookup[b2_maxBlockSize + 1];
59    static bool s_blockSizeLookupInitialized;
60  };
61
62  #endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_SHAPE_H
#define B2_SHAPE_H

#include "Box2D/Common/b2BlockAllocator.h"
#include "Box2D/Common/b2Math.h"
#include "Box2D/Collision/b2Collision.h"

/// This holds the mass data computed for a shape.
struct b2MassData
{
  /// The mass of the shape, usually in kilograms.
  float32 mass;

  /// The position of the shape's centroid relative to the shape's origin.
  b2Vec2 center;

  /// The rotational inertia of the shape about the local origin.
  float32 I;
};

/// A shape is used for collision detection. You can create a shape however you like.
/// Shapes used for simulation in b2World are created automatically when a b2Fixture
/// is created. Shapes may encapsulate a one or more child shapes.
class b2Shape
{
public:

  enum Type
  {
    e_circle = 0,
    e_edge = 1,
    e_polygon = 2,
    e_chain = 3,
    e_typeCount = 4
  };

  virtual ~b2Shape() {}

  /// Clone the concrete shape using the provided allocator.
  virtual b2Shape* Clone(b2BlockAllocator* allocator) const = 0;

  /// Get the type of this shape. You can use this to down cast to the concrete shape.
  /// @return the shape type.
  Type GetType() const;

```



```
  /// Get the number of child primitives.
  virtual int32 GetChildCount() const = 0;

  /// Test a point for containment in this shape. This only works for convex shapes.
  /// @param xf the shape world transform.
  /// @param p a point in world coordinates.
  virtual bool TestPoint(const b2Transform& xf, const b2Vec2& p) const = 0;

  /// Cast a ray against a child shape.
  /// @param output the ray-cast results.
  /// @param input the ray-cast input parameters.
  /// @param transform the transform to be applied to the shape.
  /// @param childIndex the child shape index
  virtual bool RayCast(b2RayCastOutput* output, const b2RayCastInput& input,
            const b2Transform& transform, int32 childIndex) const = 0;

  /// Given a transform, compute the associated axis aligned bounding box for a child shape.
  /// @param aabb returns the axis aligned box.
  /// @param xf the world transform of the shape.
  /// @param childIndex the child shape
  virtual void ComputeAABB(b2AABB* aabb, const b2Transform& xf, int32 childIndex) const = 0;

  /// Compute the mass properties of this shape using its dimensions and density.
  /// The inertia tensor is computed about the local origin.
  /// @param massData returns the mass data for this shape.
  /// @param density the density in kilograms per meter squared.
  virtual void ComputeMass(b2MassData* massData, float32 density) const = 0;

  Type m_type;

  /// Radius of a shape. For polygonal shapes this must be b2_polygonRadius. There is no support for
  /// making rounded polygons.
  float32 m_radius;
};

inline b2Shape::Type b2Shape::GetType() const
{
  return m_type;
}

#endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_POLYGON_SHAPE_H
#define B2_POLYGON_SHAPE_H

#include "Box2D/Collision/Shapes/b2Shape.h"

/// A convex polygon. It is assumed that the interior of the polygon is to
/// the left of each edge.
/// Polygons have a maximum number of vertices equal to b2_maxPolygonVertices.
/// In most cases you should not need many vertices for a convex polygon.
class b2PolygonShape : public b2Shape
{
public:
  b2PolygonShape();

  /// Implement b2Shape.
  b2Shape* Clone(b2BlockAllocator* allocator) const override;

  /// @see b2Shape::GetChildCount
  int32 GetChildCount() const override;

  /// Create a convex hull from the given array of local points.
  /// The count must be in the range [3, b2_maxPolygonVertices].
  /// @warning the points may be re-ordered, even if they form a convex polygon
  /// @warning collinear points are handled but not removed. Collinear points
  /// may lead to poor stacking behavior.
  void Set(const b2Vec2* points, int32 count);

  /// Build vertices to represent an axis-aligned box centered on the local orig
in.
  /// @param hx the half-width.
  /// @param hy the half-height.
  void SetAsBox(float32 hx, float32 hy);

  /// Build vertices to represent an oriented box.
  /// @param hx the half-width.
  /// @param hy the half-height.
  /// @param center the center of the box in local coordinates.
  /// @param angle the rotation of the box in local coordinates.
  void SetAsBox(float32 hx, float32 hy, const b2Vec2& center, float32 angle);

  /// @see b2Shape::TestPoint
  bool TestPoint(const b2Transform& transform, const b2Vec2& p) const override;

  /// Implement b2Shape.
  bool RayCast(b2RayCastOutput* output, const b2RayCastInput& input,
          const b2Transform& transform, int32 childIndex) const override;

  /// @see b2Shape::ComputeAABB
```



```
  void ComputeAABB(b2AABB* aabb, const b2Transform& transform, int32 childIndex)
 const override;

  /// @see b2Shape::ComputeMass
  void ComputeMass(b2MassData* massData, float32 density) const override;

  /// Validate convexity. This is a very time consuming operation.
  /// @returns true if valid
  bool Validate() const;

  b2Vec2 m_centroid;
  b2Vec2 m_vertices[b2_maxPolygonVertices];
  b2Vec2 m_normals[b2_maxPolygonVertices];
  int32 m_count;
};

inline b2PolygonShape::b2PolygonShape()
{
  m_type = e_polygon;
  m_radius = b2_polygonRadius;
  m_count = 0;
  m_centroid.SetZero();
}

#endif
```

```
/*
* Copyright (c) 2006-2010 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_EDGE_SHAPE_H
#define B2_EDGE_SHAPE_H

#include "Box2D/Collision/Shapes/b2Shape.h"

/// A line segment (edge) shape. These can be connected in chains or loops
/// to other edge shapes. The connectivity information is used to ensure
/// correct contact normals.
class b2EdgeShape : public b2Shape
{
public:
  b2EdgeShape();

  /// Set this as an isolated edge.
  void Set(const b2Vec2& v1, const b2Vec2& v2);

  /// Implement b2Shape.
  b2Shape* Clone(b2BlockAllocator* allocator) const override;

  /// @see b2Shape::GetChildCount
  int32 GetChildCount() const override;

  /// @see b2Shape::TestPoint
  bool TestPoint(const b2Transform& transform, const b2Vec2& p) const override;

  /// Implement b2Shape.
  bool RayCast(b2RayCastOutput* output, const b2RayCastInput& input,
        const b2Transform& transform, int32 childIndex) const override;

  /// @see b2Shape::ComputeAABB
  void ComputeAABB(b2AABB* aabb, const b2Transform& transform, int32 childIndex)
 const override;

  /// @see b2Shape::ComputeMass
  void ComputeMass(b2MassData* massData, float32 density) const override;

  /// These are the edge vertices
  b2Vec2 m_vertex1, m_vertex2;

  /// Optional adjacent vertices. These are used for smooth collision.
  b2Vec2 m_vertex0, m_vertex3;
  bool m_hasVertex0, m_hasVertex3;
};

inline b2EdgeShape::b2EdgeShape()
{
  m_type = e_edge;
  m_radius = b2_polygonRadius;
```



```
  m_vertex0.x = 0.0f;
  m_vertex0.y = 0.0f;
  m_vertex3.x = 0.0f;
  m_vertex3.y = 0.0f;
  m_hasVertex0 = false;
  m_hasVertex3 = false;
}

#endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_CIRCLE_SHAPE_H
#define B2_CIRCLE_SHAPE_H

#include "Box2D/Collision/Shapes/b2Shape.h"

/// A circle shape.
class b2CircleShape : public b2Shape
{
public:
  b2CircleShape();

  /// Implement b2Shape.
  b2Shape* Clone(b2BlockAllocator* allocator) const override;

  /// @see b2Shape::GetChildCount
  int32 GetChildCount() const override;

  /// Implement b2Shape.
  bool TestPoint(const b2Transform& transform, const b2Vec2& p) const override;

  /// Implement b2Shape.
  bool RayCast(b2RayCastOutput* output, const b2RayCastInput& input,
        const b2Transform& transform, int32 childIndex) const override;

  /// @see b2Shape::ComputeAABB
  void ComputeAABB(b2AABB* aabb, const b2Transform& transform, int32 childIndex)
 const override;

  /// @see b2Shape::ComputeMass
  void ComputeMass(b2MassData* massData, float32 density) const override;

  /// Position
  b2Vec2 m_p;
};

inline b2CircleShape::b2CircleShape()
{
  m_type = e_circle;
  m_radius = 0.0f;
  m_p.SetZero();
}

#endif
```



```
/*
* Copyright (c) 2006-2010 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_CHAIN_SHAPE_H
#define B2_CHAIN_SHAPE_H

#include "Box2D/Collision/Shapes/b2Shape.h"

class b2EdgeShape;

/// A chain shape is a free form sequence of line segments.
/// The chain has two-sided collision, so you can use inside and outside collisi
on.
/// Therefore, you may use any winding order.
/// Since there may be many vertices, they are allocated using b2Alloc.
/// Connectivity information is used to create smooth collisions.
/// WARNING: The chain will not collide properly if there are self-intersections
.
class b2ChainShape : public b2Shape
{
public:
  b2ChainShape();

  /// The destructor frees the vertices using b2Free.
  ~b2ChainShape();

  /// Clear all data.
  void Clear();

  /// Create a loop. This automatically adjusts connectivity.
  /// @param vertices an array of vertices, these are copied
  /// @param count the vertex count
  void CreateLoop(const b2Vec2* vertices, int32 count);

  /// Create a chain with isolated end vertices.
  /// @param vertices an array of vertices, these are copied
  /// @param count the vertex count
  void CreateChain(const b2Vec2* vertices, int32 count);

  /// Establish connectivity to a vertex that precedes the first vertex.
  /// Don't call this for loops.
  void SetPrevVertex(const b2Vec2& prevVertex);

  /// Establish connectivity to a vertex that follows the last vertex.
  /// Don't call this for loops.
  void SetNextVertex(const b2Vec2& nextVertex);

  /// Implement b2Shape. Vertices are cloned using b2Alloc.
  b2Shape* Clone(b2BlockAllocator* allocator) const override;

  /// @see b2Shape::GetChildCount
```

```
65    int32 GetChildCount() const override;
66
67    /// Get a child edge.
68    void GetChildEdge(b2EdgeShape* edge, int32 index) const;
69
70    /// This always return false.
71    /// @see b2Shape::TestPoint
72    bool TestPoint(const b2Transform& transform, const b2Vec2& p) const override;
73
74    /// Implement b2Shape.
75    bool RayCast(b2RayCastOutput* output, const b2RayCastInput& input,
76                 const b2Transform& transform, int32 childIndex) const override;
77
78    /// @see b2Shape::ComputeAABB
79    void ComputeAABB(b2AABB* aabb, const b2Transform& transform, int32 childIndex)
      const override;
80
81    /// Chains have zero mass.
82    /// @see b2Shape::ComputeMass
83    void ComputeMass(b2MassData* massData, float32 density) const override;
84
85    /// The vertices. Owned by this class.
86    b2Vec2* m_vertices;
87
88    /// The vertex count.
89    int32 m_count;
90
91    b2Vec2 m_prevVertex, m_nextVertex;
92    bool m_hasPrevVertex, m_hasNextVertex;
93  };
94
95  inline b2ChainShape::b2ChainShape()
96  {
97    m_type = e_chain;
98    m_radius = b2_polygonRadius;
99    m_vertices = nullptr;
100   m_count = 0;
101   m_hasPrevVertex = false;
102   m_hasNextVertex = false;
103 }
104
105 #endif
```



```
1   /*
2   * Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
3   *
4   * This software is provided 'as-is', without any express or implied
5   * warranty.  In no event will the authors be held liable for any damages
6   * arising from the use of this software.
7   * Permission is granted to anyone to use this software for any purpose,
8   * including commercial applications, and to alter it and redistribute it
9   * freely, subject to the following restrictions:
10  * 1. The origin of this software must not be misrepresented; you must not
11  * claim that you wrote the original software. If you use this software
12  * in a product, an acknowledgment in the product documentation would be
13  * appreciated but is not required.
14  * 2. Altered source versions must be plainly marked as such, and must not be
15  * misrepresented as being the original software.
16  * 3. This notice may not be removed or altered from any source distribution.
17  */
18
19  #ifndef B2_TIME_OF_IMPACT_H
20  #define B2_TIME_OF_IMPACT_H
21
22  #include "Box2D/Common/b2Math.h"
23  #include "Box2D/Collision/b2Distance.h"
24
25  /// Input parameters for b2TimeOfImpact
26  struct b2TOIInput
27  {
28    b2DistanceProxy proxyA;
29    b2DistanceProxy proxyB;
30    b2Sweep sweepA;
31    b2Sweep sweepB;
32    float32 tMax;   // defines sweep interval [0, tMax]
33  };
34
35  /// Output parameters for b2TimeOfImpact.
36  struct b2TOIOutput
37  {
38    enum State
39    {
40      e_unknown,
41      e_failed,
42      e_overlapped,
43      e_touching,
44      e_separated
45    };
46
47    State state;
48    float32 t;
49  };
50
51  /// Compute the upper bound on time before two shapes penetrate. Time is represe
    nted as
52  /// a fraction between [0,tMax]. This uses a swept separating axis and may miss
    some intermediate,
53  /// non-tunneling collisions. If you change the time interval, you should call t
    his function
54  /// again.
55  /// Note: use b2Distance to compute the contact point and normal at the time of
    impact.
56  void b2TimeOfImpact(b2TOIOutput* output, const b2TOIInput* input);
57
58  #endif
```

```
/*
* Copyright (c) 2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_DYNAMIC_TREE_H
#define B2_DYNAMIC_TREE_H

#include "Box2D/Collision/b2Collision.h"
#include "Box2D/Common/b2GrowableStack.h"

#define b2_nullNode (-1)

/// A node in the dynamic tree. The client does not interact with this directly.
struct b2TreeNode
{
  bool IsLeaf() const
  {
    return child1 == b2_nullNode;
  }

  /// Enlarged AABB
  b2AABB aabb;

  void* userData;

  union
  {
    int32 parent;
    int32 next;
  };

  int32 child1;
  int32 child2;

  // leaf = 0, free node = -1
  int32 height;
};

/// A dynamic AABB tree broad-phase, inspired by Nathanael Presson's btDbvt.
/// A dynamic tree arranges data in a binary tree to accelerate
/// queries such as volume queries and ray casts. Leafs are proxies
/// with an AABB. In the tree we expand the proxy AABB by b2_fatAABBFactor
/// so that the proxy AABB is bigger than the client object. This allows the client
/// object to move by small amounts without triggering a tree update.
///
/// Nodes are pooled and relocatable, so we use node indices rather than pointers.
class b2DynamicTree
{
public:
  /// Constructing the tree initializes the node pool.
```



```
  b2DynamicTree();

  /// Destroy the tree, freeing the node pool.
  ~b2DynamicTree();

  /// Create a proxy. Provide a tight fitting AABB and a userData pointer.
  int32 CreateProxy(const b2AABB& aabb, void* userData);

  /// Destroy a proxy. This asserts if the id is invalid.
  void DestroyProxy(int32 proxyId);

  /// Move a proxy with a swepted AABB. If the proxy has moved outside of its fattened AABB,
  /// then the proxy is removed from the tree and re-inserted. Otherwise
  /// the function returns immediately.
  /// @return true if the proxy was re-inserted.
  bool MoveProxy(int32 proxyId, const b2AABB& aabb1, const b2Vec2& displacement);

  /// Get proxy user data.
  /// @return the proxy user data or 0 if the id is invalid.
  void* GetUserData(int32 proxyId) const;

  /// Get the fat AABB for a proxy.
  const b2AABB& GetFatAABB(int32 proxyId) const;

  /// Query an AABB for overlapping proxies. The callback class
  /// is called for each proxy that overlaps the supplied AABB.
  template <typename T>
  void Query(T* callback, const b2AABB& aabb) const;

  /// Ray-cast against the proxies in the tree. This relies on the callback
  /// to perform a exact ray-cast in the case were the proxy contains a shape.
  /// The callback also performs the any collision filtering. This has performance
  /// roughly equal to k * log(n), where k is the number of collisions and n is the
  /// number of proxies in the tree.
  /// @param input the ray-cast input data. The ray extends from p1 to p1 + maxFraction * (p2 - p1).
  /// @param callback a callback class that is called for each proxy that is hit by the ray.
  template <typename T>
  void RayCast(T* callback, const b2RayCastInput& input) const;

  /// Validate this tree. For testing.
  void Validate() const;

  /// Compute the height of the binary tree in O(N) time. Should not be
  /// called often.
  int32 GetHeight() const;

  /// Get the maximum balance of an node in the tree. The balance is the difference
  /// in height of the two children of a node.
  int32 GetMaxBalance() const;

  /// Get the ratio of the sum of the node areas to the root area.
  float32 GetAreaRatio() const;

  /// Build an optimal tree. Very expensive. For testing.
  void RebuildBottomUp();

  /// Shift the world origin. Useful for large worlds.
  /// The shift formula is: position -= newOrigin
  /// @param newOrigin the new origin with respect to the old origin
```

```
124   void ShiftOrigin(const b2Vec2& newOrigin);
125
126 private:
127
128   int32 AllocateNode();
129   void FreeNode(int32 node);
130
131   void InsertLeaf(int32 node);
132   void RemoveLeaf(int32 node);
133
134   int32 Balance(int32 index);
135
136   int32 ComputeHeight() const;
137   int32 ComputeHeight(int32 nodeId) const;
138
139   void ValidateStructure(int32 index) const;
140   void ValidateMetrics(int32 index) const;
141
142   int32 m_root;
143
144   b2TreeNode* m_nodes;
145   int32 m_nodeCount;
146   int32 m_nodeCapacity;
147
148   int32 m_freeList;
149
150   /// This is used to incrementally traverse the tree for re-balancing.
151   uint32 m_path;
152
153   int32 m_insertionCount;
154 };
155
156 inline void* b2DynamicTree::GetUserData(int32 proxyId) const
157 {
158   b2Assert(0 ≤ proxyId ∧ proxyId < m_nodeCapacity);
159   return m_nodes[proxyId].userData;
160 }
161
162 inline const b2AABB& b2DynamicTree::GetFatAABB(int32 proxyId) const
163 {
164   b2Assert(0 ≤ proxyId ∧ proxyId < m_nodeCapacity);
165   return m_nodes[proxyId].aabb;
166 }
167
168 template <typename T>
169 inline void b2DynamicTree::Query(T* callback, const b2AABB& aabb) const
170 {
171   b2GrowableStack<int32, 256> stack;
172   stack.Push(m_root);
173
174   while (stack.GetCount() > 0)
175   {
176     int32 nodeId = stack.Pop();
177     if (nodeId ≡ b2_nullNode)
178     {
179       continue;
180     }
181
182     const b2TreeNode* node = m_nodes + nodeId;
183
184     if (b2TestOverlap(node→aabb, aabb))
185     {
186       if (node→IsLeaf())
187       {
188         bool proceed = callback→QueryCallback(nodeId);
189         if (proceed ≡ false)
```



```
190         {
191           return;
192         }
193       }
194       else
195       {
196         stack.Push(node→child1);
197         stack.Push(node→child2);
198       }
199     }
200   }
201 }
202
203 template <typename T>
204 inline void b2DynamicTree::RayCast(T* callback, const b2RayCastInput& input) con
    st
205 {
206   b2Vec2 p1 = input.p1;
207   b2Vec2 p2 = input.p2;
208   b2Vec2 r = p2 − p1;
209   b2Assert(r.LengthSquared() > 0.0f);
210   r.Normalize();
211
212   // v is perpendicular to the segment.
213   b2Vec2 v = b2Cross(1.0f, r);
214   b2Vec2 abs_v = b2Abs(v);
215
216   // Separating axis for segment (Gino, p80).
217   // |dot(v, p1 − c)| > dot(|v|, h)
218
219   float32 maxFraction = input.maxFraction;
220
221   // Build a bounding box for the segment.
222   b2AABB segmentAABB;
223   {
224     b2Vec2 t = p1 + maxFraction * (p2 − p1);
225     segmentAABB.lowerBound = b2Min(p1, t);
226     segmentAABB.upperBound = b2Max(p1, t);
227   }
228
229   b2GrowableStack<int32, 256> stack;
230   stack.Push(m_root);
231
232   while (stack.GetCount() > 0)
233   {
234     int32 nodeId = stack.Pop();
235     if (nodeId ≡ b2_nullNode)
236     {
237       continue;
238     }
239
240     const b2TreeNode* node = m_nodes + nodeId;
241
242     if (b2TestOverlap(node→aabb, segmentAABB) ≡ false)
243     {
244       continue;
245     }
246
247     // Separating axis for segment (Gino, p80).
248     // |dot(v, p1 − c)| > dot(|v|, h)
249     b2Vec2 c = node→aabb.GetCenter();
250     b2Vec2 h = node→aabb.GetExtents();
251     float32 separation = b2Abs(b2Dot(v, p1 − c)) − b2Dot(abs_v, h);
252     if (separation > 0.0f)
253     {
254       continue;
```

```
255       }
256
257       if (node→IsLeaf())
258       {
259         b2RayCastInput subInput;
260         subInput.p1 = input.p1;
261         subInput.p2 = input.p2;
262         subInput.maxFraction = maxFraction;
263
264         float32 value = callback→RayCastCallback(subInput, nodeId);
265
266         if (value ≡ 0.0f)
267         {
268           // The client has terminated the ray cast.
269           return;
270         }
271
272         if (value > 0.0f)
273         {
274           // Update segment bounding box.
275           maxFraction = value;
276           b2Vec2 t = p1 + maxFraction * (p2 − p1);
277           segmentAABB.lowerBound = b2Min(p1, t);
278           segmentAABB.upperBound = b2Max(p1, t);
279         }
280       }
281       else
282       {
283         stack.Push(node→child1);
284         stack.Push(node→child2);
285       }
286     }
287   }
288
289   #endif
```



```
1
2   /*
3   * Copyright (c) 2006−2009 Erin Catto http://www.box2d.org
4   *
5   * This software is provided 'as−is', without any express or implied
6   * warranty.  In no event will the authors be held liable for any damages
7   * arising from the use of this software.
8   * Permission is granted to anyone to use this software for any purpose,
9   * including commercial applications, and to alter it and redistribute it
10  * freely, subject to the following restrictions:
11  * 1. The origin of this software must not be misrepresented; you must not
12  * claim that you wrote the original software. If you use this software
13  * in a product, an acknowledgment in the product documentation would be
14  * appreciated but is not required.
15  * 2. Altered source versions must be plainly marked as such, and must not be
16  * misrepresented as being the original software.
17  * 3. This notice may not be removed or altered from any source distribution.
18  */
19
20  #ifndef B2_DISTANCE_H
21  #define B2_DISTANCE_H
22
23  #include "Box2D/Common/b2Math.h"
24
25  class b2Shape;
26
27  /// A distance proxy is used by the GJK algorithm.
28  /// It encapsulates any shape.
29  struct b2DistanceProxy
30  {
31    b2DistanceProxy() : m_vertices(nullptr), m_count(0), m_radius(0.0f) {}
32
33    /// Initialize the proxy using the given shape. The shape
34    /// must remain in scope while the proxy is in use.
35    void Set(const b2Shape* shape, int32 index);
36
37      /// Initialize the proxy using a vertex cloud and radius. The vertices
38      /// must remain in scope while the proxy is in use.
39      void Set(const b2Vec2* vertices, int32 count, float32 radius);
40
41    /// Get the supporting vertex index in the given direction.
42    int32 GetSupport(const b2Vec2& d) const;
43
44    /// Get the supporting vertex in the given direction.
45    const b2Vec2& GetSupportVertex(const b2Vec2& d) const;
46
47    /// Get the vertex count.
48    int32 GetVertexCount() const;
49
50    /// Get a vertex by index. Used by b2Distance.
51    const b2Vec2& GetVertex(int32 index) const;
52
53    b2Vec2 m_buffer[2];
54    const b2Vec2* m_vertices;
55    int32 m_count;
56    float32 m_radius;
57  };
58
59  /// Used to warm start b2Distance.
60  /// Set count to zero on first call.
61  struct b2SimplexCache
62  {
63    float32 metric;   ///< length or area
64    uint16 count;
65    uint8 indexA[3];  ///< vertices on shape A
66    uint8 indexB[3];  ///< vertices on shape B
```

```
};

/// Input for b2Distance.
/// You have to option to use the shape radii
/// in the computation. Even
struct b2DistanceInput
{
  b2DistanceProxy proxyA;
  b2DistanceProxy proxyB;
  b2Transform transformA;
  b2Transform transformB;
  bool useRadii;
};

/// Output for b2Distance.
struct b2DistanceOutput
{
  b2Vec2 pointA;    ///< closest point on shapeA
  b2Vec2 pointB;    ///< closest point on shapeB
  float32 distance;
  int32 iterations; ///< number of GJK iterations used
};

/// Compute the closest points between two shapes. Supports any combination of:
/// b2CircleShape, b2PolygonShape, b2EdgeShape. The simplex cache is input/output.
/// On the first call set b2SimplexCache.count to zero.
void b2Distance(b2DistanceOutput* output,
        b2SimplexCache* cache,
        const b2DistanceInput* input);

/// Input parameters for b2ShapeCast
struct b2ShapeCastInput
{
  b2DistanceProxy proxyA;
  b2DistanceProxy proxyB;
  b2Transform transformA;
  b2Transform transformB;
  b2Vec2 translationB;
};

/// Output results for b2ShapeCast
struct b2ShapeCastOutput
{
  b2Vec2 point;
  b2Vec2 normal;
  float32 lambda;
  int32 iterations;
};

/// Perform a linear shape cast of shape B moving and shape A fixed. Determines the hit point, normal, and translation fraction.
bool b2ShapeCast(b2ShapeCastOutput* output, const b2ShapeCastInput* input);

//////////////////////////////////////////////////////////////////////

inline int32 b2DistanceProxy::GetVertexCount() const
{
  return m_count;
}

inline const b2Vec2& b2DistanceProxy::GetVertex(int32 index) const
{
  b2Assert(0 ≤ index ∧ index < m_count);
  return m_vertices[index];
}
```



```

inline int32 b2DistanceProxy::GetSupport(const b2Vec2& d) const
{
  int32 bestIndex = 0;
  float32 bestValue = b2Dot(m_vertices[0], d);
  for (int32 i = 1; i < m_count; ++i)
  {
    float32 value = b2Dot(m_vertices[i], d);
    if (value > bestValue)
    {
      bestIndex = i;
      bestValue = value;
    }
  }

  return bestIndex;
}

inline const b2Vec2& b2DistanceProxy::GetSupportVertex(const b2Vec2& d) const
{
  int32 bestIndex = 0;
  float32 bestValue = b2Dot(m_vertices[0], d);
  for (int32 i = 1; i < m_count; ++i)
  {
    float32 value = b2Dot(m_vertices[i], d);
    if (value > bestValue)
    {
      bestIndex = i;
      bestValue = value;
    }
  }

  return m_vertices[bestIndex];
}

#endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef B2_COLLISION_H
#define B2_COLLISION_H

#include "Box2D/Common/b2Math.h"
#include <limits.h>

/// @file
/// Structures and functions used for computing contact points, distance
/// queries, and TOI queries.

class b2Shape;
class b2CircleShape;
class b2EdgeShape;
class b2PolygonShape;

const uint8 b2_nullFeature = UCHAR_MAX;

/// The features that intersect to form the contact point
/// This must be 4 bytes or less.
struct b2ContactFeature
{
  enum Type
  {
    e_vertex = 0,
    e_face = 1
  };

  uint8 indexA;   ///< Feature index on shapeA
  uint8 indexB;   ///< Feature index on shapeB
  uint8 typeA;    ///< The feature type on shapeA
  uint8 typeB;    ///< The feature type on shapeB
};

/// Contact ids to facilitate warm starting.
union b2ContactID
{
  b2ContactFeature cf;
  uint32 key;         ///< Used to quickly compare contact ids.
};

/// A manifold point is a contact point belonging to a contact
/// manifold. It holds details related to the geometry and dynamics
/// of the contact points.
/// The local point usage depends on the manifold type:
/// -e_circles: the local center of circleB
/// -e_faceA: the local center of cirlceB or the clip point of polygonB
/// -e_faceB: the clip point of polygonA
/// This structure is stored across time steps, so we keep it small.
```



```
/// Note: the impulses are used for internal caching and may not
/// provide reliable contact forces, especially for high speed collisions.
struct b2ManifoldPoint
{
  b2Vec2 localPoint;    ///< usage depends on manifold type
  float32 normalImpulse;  ///< the non-penetration impulse
  float32 tangentImpulse; ///< the friction impulse
  b2ContactID id;     ///< uniquely identifies a contact point between two shapes
};

/// A manifold for two touching convex shapes.
/// Box2D supports multiple types of contact:
/// - clip point versus plane with radius
/// - point versus point with radius (circles)
/// The local point usage depends on the manifold type:
/// -e_circles: the local center of circleA
/// -e_faceA: the center of faceA
/// -e_faceB: the center of faceB
/// Similarly the local normal usage:
/// -e_circles: not used
/// -e_faceA: the normal on polygonA
/// -e_faceB: the normal on polygonB
/// We store contacts in this way so that position correction can
/// account for movement, which is critical for continuous physics.
/// All contact scenarios must be expressed in one of these types.
/// This structure is stored across time steps, so we keep it small.
struct b2Manifold
{
  enum Type
  {
    e_circles,
    e_faceA,
    e_faceB
  };

  b2ManifoldPoint points[b2_maxManifoldPoints]; ///< the points of contact
  b2Vec2 localNormal;               ///< not use for Type::e_points
  b2Vec2 localPoint;                ///< usage depends on manifold type
  Type type;
  int32 pointCount;              ///< the number of manifold points
};

/// This is used to compute the current state of a contact manifold.
struct b2WorldManifold
{
  /// Evaluate the manifold with supplied transforms. This assumes
  /// modest motion from the original state. This does not change the
  /// point count, impulses, etc. The radii must come from the shapes
  /// that generated the manifold.
  void Initialize(const b2Manifold* manifold,
          const b2Transform& xfA, float32 radiusA,
          const b2Transform& xfB, float32 radiusB);

  b2Vec2 normal;                ///< world vector pointing from A to B
  b2Vec2 points[b2_maxManifoldPoints];    ///< world contact point (point of intersection)
  float32 separations[b2_maxManifoldPoints];  ///< a negative value indicates overlap, in meters
};

/// This is used for determining the state of contact points.
enum b2PointState
{
  b2_nullState,   ///< point does not exist
  b2_addState,    ///< point was added in the update
```

```
130   b2_persistState,  ///< point persisted across the update
131   b2_removeState    ///< point was removed in the update
132 };
133
134 /// Compute the point states given two manifolds. The states pertain to the tran
    sition from manifold1
135 /// to manifold2. So state1 is either persist or remove while state2 is either a
    dd or persist.
136 void b2GetPointStates(b2PointState state1[b2_maxManifoldPoints], b2PointState st
    ate2[b2_maxManifoldPoints],
137                       const b2Manifold* manifold1, const b2Manifold* manifold2);
138
139 /// Used for computing contact manifolds.
140 struct b2ClipVertex
141 {
142   b2Vec2 v;
143   b2ContactID id;
144 };
145
146 /// Ray-cast input data. The ray extends from p1 to p1 + maxFraction * (p2 - p1)
    .
147 struct b2RayCastInput
148 {
149   b2Vec2 p1, p2;
150   float32 maxFraction;
151 };
152
153 /// Ray-cast output data. The ray hits at p1 + fraction * (p2 - p1), where p1 an
    d p2
154 /// come from b2RayCastInput.
155 struct b2RayCastOutput
156 {
157   b2Vec2 normal;
158   float32 fraction;
159 };
160
161 /// An axis aligned bounding box.
162 struct b2AABB
163 {
164   /// Verify that the bounds are sorted.
165   bool IsValid() const;
166
167   /// Get the center of the AABB.
168   b2Vec2 GetCenter() const
169   {
170     return 0.5f * (lowerBound + upperBound);
171   }
172
173   /// Get the extents of the AABB (half-widths).
174   b2Vec2 GetExtents() const
175   {
176     return 0.5f * (upperBound - lowerBound);
177   }
178
179   /// Get the perimeter length
180   float32 GetPerimeter() const
181   {
182     float32 wx = upperBound.x - lowerBound.x;
183     float32 wy = upperBound.y - lowerBound.y;
184     return 2.0f * (wx + wy);
185   }
186
187   /// Combine an AABB into this one.
188   void Combine(const b2AABB& aabb)
189   {
190     lowerBound = b2Min(lowerBound, aabb.lowerBound);
```



```
191     upperBound = b2Max(upperBound, aabb.upperBound);
192   }
193
194   /// Combine two AABBs into this one.
195   void Combine(const b2AABB& aabb1, const b2AABB& aabb2)
196   {
197     lowerBound = b2Min(aabb1.lowerBound, aabb2.lowerBound);
198     upperBound = b2Max(aabb1.upperBound, aabb2.upperBound);
199   }
200
201   /// Does this aabb contain the provided AABB.
202   bool Contains(const b2AABB& aabb) const
203   {
204     bool result = true;
205     result = result ∧ lowerBound.x ≤ aabb.lowerBound.x;
206     result = result ∧ lowerBound.y ≤ aabb.lowerBound.y;
207     result = result ∧ aabb.upperBound.x ≤ upperBound.x;
208     result = result ∧ aabb.upperBound.y ≤ upperBound.y;
209     return result;
210   }
211
212   bool RayCast(b2RayCastOutput* output, const b2RayCastInput& input) const;
213
214   b2Vec2 lowerBound;  ///< the lower vertex
215   b2Vec2 upperBound;  ///< the upper vertex
216 };
217
218 /// Compute the collision manifold between two circles.
219 void b2CollideCircles(b2Manifold* manifold,
220                       const b2CircleShape* circleA, const b2Transform& xfA,
221                       const b2CircleShape* circleB, const b2Transform& xfB);
222
223 /// Compute the collision manifold between a polygon and a circle.
224 void b2CollidePolygonAndCircle(b2Manifold* manifold,
225                                const b2PolygonShape* polygonA, const b2Transform& xfA,
226                                const b2CircleShape* circleB, const b2Transform& xfB);
227
228 /// Compute the collision manifold between two polygons.
229 void b2CollidePolygons(b2Manifold* manifold,
230                        const b2PolygonShape* polygonA, const b2Transform& xfA,
231                        const b2PolygonShape* polygonB, const b2Transform& xfB);
232
233 /// Compute the collision manifold between an edge and a circle.
234 void b2CollideEdgeAndCircle(b2Manifold* manifold,
235                             const b2EdgeShape* polygonA, const b2Transform& xfA,
236                             const b2CircleShape* circleB, const b2Transform& xfB);
237
238 /// Compute the collision manifold between an edge and a circle.
239 void b2CollideEdgeAndPolygon(b2Manifold* manifold,
240                              const b2EdgeShape* edgeA, const b2Transform& xfA,
241                              const b2PolygonShape* circleB, const b2Transform& xfB);
242
243 /// Clipping for contact manifolds.
244 int32 b2ClipSegmentToLine(b2ClipVertex vOut[2], const b2ClipVertex vIn[2],
245                           const b2Vec2& normal, float32 offset, int32 vertexIndexA);
246
247 /// Determine if two generic shapes overlap.
248 bool b2TestOverlap( const b2Shape* shapeA, int32 indexA,
249                     const b2Shape* shapeB, int32 indexB,
250                     const b2Transform& xfA, const b2Transform& xfB);
251
252 // ---------------- Inline Functions ------------------------------------------
253
254 inline bool b2AABB::IsValid() const
255 {
256   b2Vec2 d = upperBound - lowerBound;
```

```
257   bool valid = d.x ≥ 0.0f ∧ d.y ≥ 0.0f;
258   valid = valid ∧ lowerBound.IsValid() ∧ upperBound.IsValid();
259   return valid;
260 }
261
262 inline bool b2TestOverlap(const b2AABB& a, const b2AABB& b)
263 {
264   b2Vec2 d1, d2;
265   d1 = b.lowerBound − a.upperBound;
266   d2 = a.lowerBound − b.upperBound;
267
268   if (d1.x > 0.0f ∨ d1.y > 0.0f)
269     return false;
270
271   if (d2.x > 0.0f ∨ d2.y > 0.0f)
272     return false;
273
274   return true;
275 }
276
277 #endif
```



```
1  /*
2  * Copyright (c) 2006−2009 Erin Catto http://www.box2d.org
3  *
4  * This software is provided 'as−is', without any express or implied
5  * warranty.  In no event will the authors be held liable for any damages
6  * arising from the use of this software.
7  * Permission is granted to anyone to use this software for any purpose,
8  * including commercial applications, and to alter it and redistribute it
9  * freely, subject to the following restrictions:
10 * 1. The origin of this software must not be misrepresented; you must not
11 * claim that you wrote the original software. If you use this software
12 * in a product, an acknowledgment in the product documentation would be
13 * appreciated but is not required.
14 * 2. Altered source versions must be plainly marked as such, and must not be
15 * misrepresented as being the original software.
16 * 3. This notice may not be removed or altered from any source distribution.
17 */
18
19 #ifndef B2_BROAD_PHASE_H
20 #define B2_BROAD_PHASE_H
21
22 #include "Box2D/Common/b2Settings.h"
23 #include "Box2D/Collision/b2Collision.h"
24 #include "Box2D/Collision/b2DynamicTree.h"
25 #include <algorithm>
26
27 struct b2Pair
28 {
29   int32 proxyIdA;
30   int32 proxyIdB;
31 };
32
33 /// The broad−phase is used for computing pairs and performing volume queries an
    d ray casts.
34 /// This broad−phase does not persist pairs. Instead, this reports potentially n
    ew pairs.
35 /// It is up to the client to consume the new pairs and to track subsequent over
    lap.
36 class b2BroadPhase
37 {
38 public:
39
40   enum
41   {
42     e_nullProxy = −1
43   };
44
45   b2BroadPhase();
46   ~b2BroadPhase();
47
48   /// Create a proxy with an initial AABB. Pairs are not reported until
49   /// UpdatePairs is called.
50   int32 CreateProxy(const b2AABB& aabb, void* userData);
51
52   /// Destroy a proxy. It is up to the client to remove any pairs.
53   void DestroyProxy(int32 proxyId);
54
55   /// Call MoveProxy as many times as you like, then when you are done
56   /// call UpdatePairs to finalized the proxy pairs (for your time step).
57   void MoveProxy(int32 proxyId, const b2AABB& aabb, const b2Vec2& displacement);
58
59   /// Call to trigger a re−processing of it's pairs on the next call to UpdatePa
    irs.
60   void TouchProxy(int32 proxyId);
61
62   /// Get the fat AABB for a proxy.
```

```
63    const b2AABB& GetFatAABB(int32 proxyId) const;
64
65    /// Get user data from a proxy. Returns nullptr if the id is invalid.
66    void* GetUserData(int32 proxyId) const;
67
68    /// Test overlap of fat AABBs.
69    bool TestOverlap(int32 proxyIdA, int32 proxyIdB) const;
70
71    /// Get the number of proxies.
72    int32 GetProxyCount() const;
73
74    /// Update the pairs. This results in pair callbacks. This can only add pairs.
75    template <typename T>
76    void UpdatePairs(T* callback);
77
78    /// Query an AABB for overlapping proxies. The callback class
79    /// is called for each proxy that overlaps the supplied AABB.
80    template <typename T>
81    void Query(T* callback, const b2AABB& aabb) const;
82
83    /// Ray-cast against the proxies in the tree. This relies on the callback
84    /// to perform a exact ray-cast in the case were the proxy contains a shape.
85    /// The callback also performs the any collision filtering. This has performan
   ce
86    /// roughly equal to k * log(n), where k is the number of collisions and n is
   the
87    /// number of proxies in the tree.
88    /// @param input the ray-cast input data. The ray extends from p1 to p1 + maxF
   raction * (p2 - p1).
89    /// @param callback a callback class that is called for each proxy that is hit
    by the ray.
90    template <typename T>
91    void RayCast(T* callback, const b2RayCastInput& input) const;
92
93    /// Get the height of the embedded tree.
94    int32 GetTreeHeight() const;
95
96    /// Get the balance of the embedded tree.
97    int32 GetTreeBalance() const;
98
99    /// Get the quality metric of the embedded tree.
100   float32 GetTreeQuality() const;
101
102   /// Shift the world origin. Useful for large worlds.
103   /// The shift formula is: position -= newOrigin
104   /// @param newOrigin the new origin with respect to the old origin
105   void ShiftOrigin(const b2Vec2& newOrigin);
106
107 private:
108
109   friend class b2DynamicTree;
110
111   void BufferMove(int32 proxyId);
112   void UnBufferMove(int32 proxyId);
113
114   bool QueryCallback(int32 proxyId);
115
116   b2DynamicTree m_tree;
117
118   int32 m_proxyCount;
119
120   int32* m_moveBuffer;
121   int32 m_moveCapacity;
122   int32 m_moveCount;
123
124   b2Pair* m_pairBuffer;
```



```
125   int32 m_pairCapacity;
126   int32 m_pairCount;
127
128   int32 m_queryProxyId;
129 };
130
131 /// This is used to sort pairs.
132 inline bool b2PairLessThan(const b2Pair& pair1, const b2Pair& pair2)
133 {
134   if (pair1.proxyIdA < pair2.proxyIdA)
135   {
136     return true;
137   }
138
139   if (pair1.proxyIdA == pair2.proxyIdA)
140   {
141     return pair1.proxyIdB < pair2.proxyIdB;
142   }
143
144   return false;
145 }
146
147 inline void* b2BroadPhase::GetUserData(int32 proxyId) const
148 {
149   return m_tree.GetUserData(proxyId);
150 }
151
152 inline bool b2BroadPhase::TestOverlap(int32 proxyIdA, int32 proxyIdB) const
153 {
154   const b2AABB& aabbA = m_tree.GetFatAABB(proxyIdA);
155   const b2AABB& aabbB = m_tree.GetFatAABB(proxyIdB);
156   return b2TestOverlap(aabbA, aabbB);
157 }
158
159 inline const b2AABB& b2BroadPhase::GetFatAABB(int32 proxyId) const
160 {
161   return m_tree.GetFatAABB(proxyId);
162 }
163
164 inline int32 b2BroadPhase::GetProxyCount() const
165 {
166   return m_proxyCount;
167 }
168
169 inline int32 b2BroadPhase::GetTreeHeight() const
170 {
171   return m_tree.GetHeight();
172 }
173
174 inline int32 b2BroadPhase::GetTreeBalance() const
175 {
176   return m_tree.GetMaxBalance();
177 }
178
179 inline float32 b2BroadPhase::GetTreeQuality() const
180 {
181   return m_tree.GetAreaRatio();
182 }
183
184 template <typename T>
185 void b2BroadPhase::UpdatePairs(T* callback)
186 {
187   // Reset pair buffer
188   m_pairCount = 0;
189
190   // Perform tree queries for all moving proxies.
```

```
191   for (int32 i = 0; i < m_moveCount; ++i)
192   {
193     m_queryProxyId = m_moveBuffer[i];
194     if (m_queryProxyId ≡ e_nullProxy)
195     {
196       continue;
197     }
198
199     // We have to query the tree with the fat AABB so that
200     // we don't fail to create a pair that may touch later.
201     const b2AABB& fatAABB = m_tree.GetFatAABB(m_queryProxyId);
202
203     // Query tree, create pairs and add them pair buffer.
204     m_tree.Query(this, fatAABB);
205   }
206
207   // Reset move buffer
208   m_moveCount = 0;
209
210   // Sort the pair buffer to expose duplicates.
211   std::sort(m_pairBuffer, m_pairBuffer + m_pairCount, b2PairLessThan);
212
213   // Send the pairs back to the client.
214   int32 i = 0;
215   while (i < m_pairCount)
216   {
217     b2Pair* primaryPair = m_pairBuffer + i;
218     void* userDataA = m_tree.GetUserData(primaryPair→proxyIdA);
219     void* userDataB = m_tree.GetUserData(primaryPair→proxyIdB);
220
221     callback→AddPair(userDataA, userDataB);
222     ++i;
223
224     // Skip any duplicate pairs.
225     while (i < m_pairCount)
226     {
227       b2Pair* pair = m_pairBuffer + i;
228       if (pair→proxyIdA ≠ primaryPair→proxyIdA ∨ pair→proxyIdB ≠ primaryPair→
    proxyIdB)
229       {
230         break;
231       }
232       ++i;
233     }
234   }
235
236   // Try to keep the tree balanced.
237   //m_tree.Rebalance(4);
238 }
239
240 template <typename T>
241 inline void b2BroadPhase::Query(T* callback, const b2AABB& aabb) const
242 {
243   m_tree.Query(callback, aabb);
244 }
245
246 template <typename T>
247 inline void b2BroadPhase::RayCast(T* callback, const b2RayCastInput& input) cons
    t
248 {
249   m_tree.RayCast(callback, input);
250 }
251
252 inline void b2BroadPhase::ShiftOrigin(const b2Vec2& newOrigin)
253 {
254   m_tree.ShiftOrigin(newOrigin);
```



```
255 }
256
257 #endif
```

```
/*
* Copyright (c) 2006-2009 Erin Catto http://www.box2d.org
*
* This software is provided 'as-is', without any express or implied
* warranty.  In no event will the authors be held liable for any damages
* arising from the use of this software.
* Permission is granted to anyone to use this software for any purpose,
* including commercial applications, and to alter it and redistribute it
* freely, subject to the following restrictions:
* 1. The origin of this software must not be misrepresented; you must not
* claim that you wrote the original software. If you use this software
* in a product, an acknowledgment in the product documentation would be
* appreciated but is not required.
* 2. Altered source versions must be plainly marked as such, and must not be
* misrepresented as being the original software.
* 3. This notice may not be removed or altered from any source distribution.
*/

#ifndef BOX2D_H
#define BOX2D_H

/**
\mainpage Box2D API Documentation

\section intro_sec Getting Started

For documentation please see http://box2d.org/documentation.html

For discussion please visit http://box2d.org/forum
*/

// These include files constitute the main Box2D API

#include "Box2D/Common/b2Settings.h"
#include "Box2D/Common/b2Draw.h"
#include "Box2D/Common/b2Timer.h"

#include "Box2D/Collision/Shapes/b2CircleShape.h"
#include "Box2D/Collision/Shapes/b2EdgeShape.h"
#include "Box2D/Collision/Shapes/b2ChainShape.h"
#include "Box2D/Collision/Shapes/b2PolygonShape.h"

#include "Box2D/Collision/b2BroadPhase.h"
#include "Box2D/Collision/b2Distance.h"
#include "Box2D/Collision/b2DynamicTree.h"
#include "Box2D/Collision/b2TimeOfImpact.h"

#include "Box2D/Dynamics/b2Body.h"
#include "Box2D/Dynamics/b2Fixture.h"
#include "Box2D/Dynamics/b2WorldCallbacks.h"
#include "Box2D/Dynamics/b2TimeStep.h"
#include "Box2D/Dynamics/b2World.h"

#include "Box2D/Dynamics/Contacts/b2Contact.h"

#include "Box2D/Dynamics/Joints/b2DistanceJoint.h"
#include "Box2D/Dynamics/Joints/b2FrictionJoint.h"
#include "Box2D/Dynamics/Joints/b2GearJoint.h"
#include "Box2D/Dynamics/Joints/b2MotorJoint.h"
#include "Box2D/Dynamics/Joints/b2MouseJoint.h"
#include "Box2D/Dynamics/Joints/b2PrismaticJoint.h"
#include "Box2D/Dynamics/Joints/b2PulleyJoint.h"
#include "Box2D/Dynamics/Joints/b2RevoluteJoint.h"
#include "Box2D/Dynamics/Joints/b2RopeJoint.h"
#include "Box2D/Dynamics/Joints/b2WeldJoint.h"
#include "Box2D/Dynamics/Joints/b2WheelJoint.h"
```



```

#endif
```

Table of Contents

nov 26, 19 17:34 **Table of Content** Page 3/5

nov 26, 19 17:34 **Table of Content** Page 4/5

nov 26, 19 17:34 **Table of Content** Page 5/5