Índice

Enunciado

Objetivo

Desarrollar una aplicación de tipo "cliente-servidor" con interfaz gráfica que permita integrar y profundizar los conocimientos adquiridos en la materia, articulándolos en un planificado y organizado trabajo grupal.

Enunciado

Como se detalla en el apéndice, la búsqueda de experiencias más interactivas y dinámicas ha llevado a la incorporación de comportamientos físicos [1] a los juegos, realizando ciertos sacrificios en la precisión en la búsqueda de resultados interactivos [2][3]. El desarrollo del poder computacional ocurrido posibilita la simulación de sistemas complejos en forma relativamente simple, representando objetos compuestos por materiales flexibles como sistemas de partículas [4][5].

Teniendo en cuenta lo mencionado anteriormente, el objeto de este trabajo será desarrollar un juego similar a "The Incredible Machine" [6] o "Armadillo Run" [7] utilizando como soporte para la simulación física un sistema de masas y resortes (ver apéndice).

Reglas del Juego

Se dispondrá de dos modos de juego: modo solitario (un sólo jugador) y modo competitivo (dos jugadores).

En ambos modos, el objetivo del juego será intentar superar, uno tras otro, los escenarios que sean enviados por el Servidor (ver sección 4). Cada escenario será un campo bidimensional, que aparecerá inicialmente con una pelota, una o más áreas de salida y, dependiendo del escenario, elementos fijos. La forma de superarlos será la construcción de dispositivos que permitan a la pelota llegar a las áreas de salida designadas evitando que la pelota salga fuera del escenario. Los escenarios estarán sometidos a la acción de una fuerza de gravedad que actuará tanto sobre la pelota como sobre los dispositivos construidos por el jugador.

Para poder construir a los dispositivos, el jugador tendrá disponible una serie de elementos con distintas propiedades. Si bien pueden incluirse elementos adicionales a los aquí descritos, los siguientes elementos deberán estar presentes:

Barras metálicas: son elementos resistentes tanto a esfuerzos de compresión como a esfuerzos de tracción, y relativamente rígidos. No colisionan entre si ni con la pelota, aunque actúan restringiendo el movimiento de los elementos en sus extremos.

Sogas: son similares a las barras metálicas pero tienen la importante diferencia de no presentar resistencia alguna a la compresión.

Plataformas metálicas: son elementos que actúan en forma muy similar a las barras metálicas, pero presentan la importante diferencia de participar en colisiones.

Cintas de lona: actúan en forma equivalente a una soga pero participando en colisiones.

Masas: son elementos relativamente pequeños que concentran una masa considerable.

Cohetes: son similares a las plataformas metálicas pero desarrollan una fuerza en la dirección de su eje durante un cierto periodo establecido al crearlos.

Ruedas: desarrollan una fuerza de giro (torque) sobre su eje a establecer durante su creación.

Puntos fijos: son elementos que están "anclados" al escenario. Generalmente no podrán ser colocados por el jugador y serán los únicos objetos no sometidos a la acción de la gravedad.

Todos estos elementos tendrán un cierto número de puntos de conexión que permitirán unirlos entre sí para poder construir los dispositivos. Se tomará cada unión entre elementos como articulada, permitiendo el giro de los mismos sobre el punto de unión (ver Apéndice). La lista de elementos disponibles en cada escenario será determinada al crearlo, así como su costo en puntaje el jugador.

En modo competitivo, para la resolución de un determinado escenario se contará con un tiempo máximo. Superado dicho tiempo, no se permitirá la realización de más modificaciones.

El cálculo del puntaje conseguido en caso de resolver con éxito al escenario se realizará de acuerdo a la siguiente fórmula:

$$Puntaje = \frac{A}{TRes + B} + \frac{C}{TSim + D} - \sum_{e \in \text{Elementos}} Precio(e),$$

donde A, B, C y D serán constantes, TRes y TSim los tiempos de la construcción del dispositivo y de la simulación, respectivamente, y los precios de los elementos representarán un costo que se asignará al uso de los mismos. Los valores de todos estos parámetros serán establecidos al crear el escenario.

Una vez finalizada la construcción del dispositivo para resolver al escenario, se simulará y visualizará la solución, se informará al servidor del resultado, y se actualizará el puntaje.

Durante la fase de simulación no se permitirá la interacción por parte del jugador en modo alguno. En caso de que la máquina no funcione se volverá al modo de construcción para que el jugador pueda intentar resolver nuevamente el escenario.

En modo competitivo, ambas simulaciones se realizarán en paralelo y solo se volverá al modo de diseño en caso de que ambos jugadores fracasen en resolver el escenario.

Módulos de la Aplicación

La aplicación se deberá componer de 3 programas independientes pero que interactuarán entre sí: el *Cliente*, el *Servidor* y el *Editor de Escenarios*. A continuación se describirá la funcionalidad requerida de cada uno de ellos.

Cliente

Es el programa con la que los jugadores normalmente interactuarán. Para poder jugar éste se deberá conectar al *Servidor*. Deberá proveer a los jugadores con los medios para construir la máquina así como, en caso de estar jugando en modo competitivo, la posibilidad de visualizar la construcción de la máquina del adversario.

También deberá mostrarse el tiempo disponible y el puntaje actual, calculado asumiendo que la máquina funciona en forma correcta.

Servidor

El Servidor será un programa de consola que permitirá que dos jugadores puedan participar de una misma partida (en modo competitivo). Si un jugador desea jugar en modo competitivo, y no hay rival disponible, se informará la situación y se quedará a la espera de un oponente.

De desear jugarse en modo solitario, la partida comenzará inmediatamente tras la conexión del jugador. La cantidad máxima de partidas deberá ser limitada, a fin de garantizar el correcto funcionamiento del servicio.

Al iniciarse una partida se seleccionará un escenario de forma aleatoria (entre todos los disponibles), transmitiéndose el mismo al o los jugadores. Al finalizarse un escenario, comenzará automáticamente otro, nuevamente en forma aleatoria y sin repetición.

El servidor recibirá cada una de las acciones realizadas por los jugadores, transmitiendo dicha información a los rivales (modo competitivo). Se establecerá un tiempo de demora (delay) antes de comenzar a transmitir la jugada del oponente, a fin de penalizar "la copia".

Otras de las responsabilidades del servidor será controlar el ranking histórico de los máximos puntajes y controlar las conversaciones entre los participantes de una partida (chat).

Editor de Escenarios

El Editor permitirá la creación y edición de escenarios a ser utilizados dentro del juego.

Un escenario se compondrá de los siguientes elementos:

- Una imagen de fondo.
- Un área rectangular donde se permitirá al jugador colocar los elementos.
- Una o más áreas de salida rectangulares.
- Una pelota.
- Elementos fijos (que no podrán ser movidos ni modificados durante el juego).
- Precio de cada elemento y lista de elementos disponibles.
- Tiempo máximo disponible para la resolución en modo competitivo.
- Parámetros para el cálculo del puntaje.

Asimismo, antes de grabar se verificará la validez del escenario. No necesariamente tiene que ser "posible de resolver", pero si debe contar con los elementos mínimos para que el escenario tenga sentido (existencia de una pelota y de al menos un área de salida).

Físicamente, el escenario deberá almacenarse como un conjunto de archivos binarios y XML.

Restricciones Generales

Los jugadores deberán poder conversar (chat) durante el desarrollo de cada nivel. Un jugador podría inhabilitar esta característica (modo DND).

La comunicación entre los Clientes y el Servidor deberá ser realizada mediante un protocolo basado en texto plano, inclusive para el envío de archivos binarios.

Los niveles deberán ser descargados completamente desde el servidor. No se podrá asumir la preexistencia de ciertos archivos (ej. fondos) en el Cliente.

Con propósitos de debugging, el servidor deberá poseer una opción que permita la visualización de los paquetes transmitidos desde/hacia los Clientes.

Requerimientos Técnicos

Los clientes se conectarán al servidor mediante el protocolo TCP.

El editor y el cliente deberán contar con tecnología Drag & Drop.

En el servidor se deberá utilizar un thread para atender a cada cliente.

Salvo expresa autorización de la cátedra, los sockets a utilizar deben ser del tipo bloqueante.

A pesar de requerirse una interfaz 2D, de preferirse, se autoriza el uso de OpenGL [8][9].

El desarrollo puede ser realizado en Windows o en Linux. En caso de utilizar un entorno de desarrollo, la elección del mismo deberá consultarse con el ayudante designado.

Funcionalidad Opcional

Se apreciará que el desarrollo sea multiplataforma, es decir, que el mismo código fuente pueda ser compilado tanto para Windows como para Linux. Para conseguir esto, es importante recordar que, tanto GTK+ como SDL y pthreads, son multiplataforma.

Incorporación de nuevos elementos; al utilizarse dentro de un juego, el jugador recibiría una cierta penalidad sobre el puntaje y el tiempo disponible. Ejemplos: aceleradores, barras ásperas, saltos, etc.

Visualización del escenario en modo normal o en modo “masas y resortes”.

Bibliotecas Permitidas

A continuación, se indica una lista librerías permitidas. El uso de cualquier otra librería debe ser autorizada por la cátedra.

XML

- libxml2
- tinyxml
- expat

GRÁFICAS

- SDL
- OpenGL

Calendario Sugerido

Las fechas indicadas corresponden al inicio de la semana.

	Alumno 1 (Cliente gráfico)	**Alumno 2 (Servidor)**	**Alumno 3 (Editor de Escenarios)**
Semana 1 (13/10)	Análisis de la implementación de los elementos en base a masas y resortes.	Análisis del *Servidor* (diagramas y arquitectura)	Análisis del *Editor de Escenarios* (diagramas y arquitectura).
Semana 2 (20/10)	Implementación básica de elementos en base a masas y resortes.	Análisis del protocolo requerido. Implementación básica del *Servidor*.	Implementación básica del *Editor de Escenarios*, soportando colocación de elementos.
Semana 3 (27/10)	*Cliente* completo, excluyendo funcionalidad de red.	Implementación del protocolo requerido para *Cliente* y *Servidor*.	Persistencia de los escenarios.
Semana 4 (3/11)	Integración del protocolo en el *Cliente* y el *Servidor*. Integración con el *Editor de Escenarios*.		
Semana 5 (10/11)	Testing y documentación.		
Semana 6 (17/11)	**Preentrega**		
Semana 7 (24/11)	Realización de correcciones en base a los resultados de la preentrega		
Semana 8 (1/12)	**Entrega Final**		

Apéndice: Sistemas de masas y resortes

INTRODUCCIÓN

La búsqueda de un mayor realismo físico ha llevado a la incorporación de modelos con bases físicas para simular comportamientos que previamente los programadores manejaban "a mano". Si bien esto se ha visto en mayor medida en los ambientes interactivos que caracterizan a los juegos, la producción de animación 3D también viene utilizando en forma creciente métodos originalmente aplicados para simulaciones físicas. Esto se debe principalmente a tres factores: la búsqueda de un ahorro de tiempo por parte de los artistas en escenas complejas, la mayor naturalidad del comportamiento resultante y, en el caso de los juegos, la mayor flexibilidad de la interacción con el comportamiento del usuario.

Muchos de los primeros pasos realizados en búsqueda de este mayor realismo involucraron un incremento en la complejidad matemática de los programas. Por ejemplo, solo el tratar con cuerpos rígidos involucra manejar tensores de inercia para representar el comportamiento de los objetos frente a rotaciones y resolver complejos sistemas de ecuaciones para determinar las fuerzas de contacto que ocurren entre distintos objetos [10].

Podría pensarse que manejar objetos deformables sería aún más complejo. En efecto, así sería si se buscara modelar a un objeto flexible de acuerdo a los métodos de simulación física

tradicionales, ya que sería necesario resolver sistemas de ecuaciones en derivadas parciales para obtener la deformación, cuya solución numérica precisa requiere métodos notablemente complejos [11].

Pero, a diferencia de lo que ocurre en el caso en una simulación física, en un juego no se busca obtener como resultado un comportamiento "físicamente correcto"; simplemente se busca obtener un resultado que sea "visualmente correcto". Junto al gran incremento del poder computacional ocurrido en las últimas décadas, esto permite que la aplicación de métodos relativamente simples, como es el caso del que será descrito en esta sección, obtengan resultados visualmente atractivos.

¿QUÉ ES UN SISTEMA DE MASAS Y RESORTES?

Un sistema de masas y resortes puede pensarse como un conjunto de partículas con masa, denominadas "masas", unidas por vínculos que conectan a un par de masas cada uno, denominados "resortes". Cada uno de estos resortes obedece la denominada "Ley de Hooke" [12], o sea ejerce una fuerza sobre las partículas en su misma dirección y con una magnitud proporcional a la diferencia entre su longitud actual y una distancia denominada "longitud en reposo" del resorte. Como es de esperar, si la longitud del resorte en un momento dado es mayor que su longitud en reposo, el resorte "intentará" acercar a las masas y, en caso contrario, "tratará" de alejarlas. Esta ley puede representarse matemáticamente como $F = -k(l - l_0)$, donde F es la fuerza que aplica el resorte, l es la longitud actual del resorte, l0 su longitud en reposo y k una constante de proporcionalidad denominada "constante de elasticidad".

OBTENCIÓN DE SU COMPORTAMIENTO

En la simulación de un sistema de masas y resortes el tiempo avanza de modo discreto, a diferencia de como avanza el tiempo en un sistema físico real. Por lo tanto, el problema de determinar el comportamiento de un sistema de masas y resortes se reduce a obtener el estado de este sistema en instante t a partir de su estado en el instante t - Δt. El estado del sistema puede dividirse en dos partes: una que varía de acuerdo al paso del tiempo, incluyendo las posiciones y velocidades de las masas, y una que permanece inalterada, incluyendo entre otras cosas la conectividad entre masas y resortes.

Puede determinarse la variación de la posición de las masas a partir de su velocidad, pero la velocidad en sí no permanece constante, por la acción de los resortes sobre las masas. Sin embargo, si se determinan las fuerzas que los resortes ejercen sobre las masas en base a las posiciones de las partículas, puede aplicarse la Segunda Ley de Newton [13] para determinar la aceleración de las masas. Esto permite determinar la variación en las velocidades de las partículas y, por lo tanto, realizar la propagación del estado del sistema en un instante dado al estado siguiente. Este proceso de propagación es denominado "integración numérica del sistema".

La forma intuitivamente obvia de realizar esta propagación (denominada "Método de Euler"), $v(t + \Delta t) = a(t)\,\Delta t$ $x(t + \Delta t) = v(t)\,\Delta t$, tiene el inconveniente de ignorar por completo las posibles variaciones de a(t) y v(t) entre los instantes t y t + Δt. Este problema puede mitigarse aplicando dos aproximaciones distintas: disminuyendo el valor de Δt o utilizando algoritmos más sofisticados. Puede demostrarse que esta última elección es notablemente más eficiente en general y, por consiguiente, es común la utilización en la práctica de métodos de integración más sofisticados tales como el de "Runge-Kutta de cuarto orden" [14].

MÁS FUNCIONALIDAD E INCONVENIENTES

Como es de esperar, puede agregársele funcionalidad adicional a un sistema físico basado en el uso de masas y resortes. Por ejemplo:

Puede simularse la rotura de materiales permitiendo la rotura de un resorte cuando la fuerza supera un valor determinado.

Pueden agregarse fuentes externas de fuerza, tales como la gravedad o distintas clases de motores.

Pueden incorporarse masas inmóviles, útiles para representar anclajes con el mundo exterior.

También deben tenerse en cuenta algunos inconvenientes derivados del método de resolución empleado. Por ejemplo, podría pensarse que sería simple simular a un objeto rígido utilizando resortes que ejerzan fuerzas extraordinariamente altas frente a pequeñas deformaciones. De hecho, esto es lo que sucede con los objetos que denominamos como rígidos en la vida cotidiana: simplemente se deforman en forma imperceptible frente a las fuerzas a las que comúnmente los vemos sometidos.

Pero esta rigidez trae inconvenientes a la hora de simular el sistema: si el coeficiente de elasticidad de los resortes supera un cierto valor crítico, relacionado con el delta-t empleado y el método de integración utilizado, el sistema se volverá inestable y "explotará" (las partículas oscilarán con una velocidad exponencialmente creciente) [15][16]. Una forma de asegurarse de que esto no ocurra es limitar el coeficiente de elasticidad de los resortes a un valor dado y no permitir valores de delta-t superiores a un cierto límite relacionado con este coeficiente. Existen también otras opciones que involucran el uso de métodos de integración más sofisticados, denominados "métodos implícitos" [17]. Pero su aplicación en forma eficiente es relativamente compleja y excede el objeto de este trabajo práctico.

CONSTRUCCIÓN DE ELEMENTOS EN BASE AL USO DE MASAS Y RESORTES

Un sistema de masas y resortes asume por lo general que estos ejercen fuerzas solo en dirección axial, por lo que una "barra" construida como una sucesión de masas y resortes en línea (ver figura 1) presentará un comportamiento más similar al de una cuerda, ya que carecerá de resistencia a la flexión (tampoco presentará una resistencia apreciable a la compresión por razones de inestabilidad).

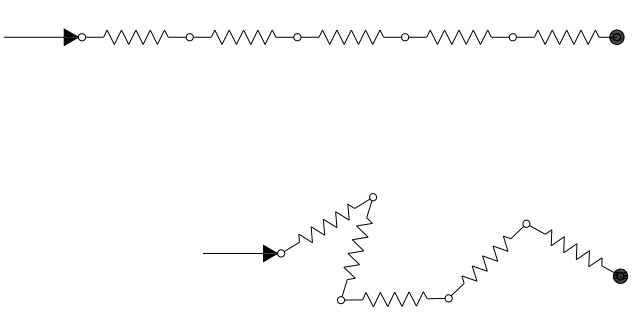

Figura 1. Ejemplo de comportamiento de una "barra" construida incorrectamente al ser sometida a un esfuerzo de compresión.

Para el análisis puede pensarse a cada resorte como una barra que se conecta en sus extremos mediante articulaciones. Por ello, para construir una barra capaz de deformarse en forma realista (o sea con múltiples elementos), puede recurrirse a reticulados tales como los utilizados en las estructuras metálicas [18].

De esta forma, la inherente rigidez de los triángulos permite la construcción de combinaciones de masas y partículas que presenten la resistencia que se espera a la flexión (ver figura 2).

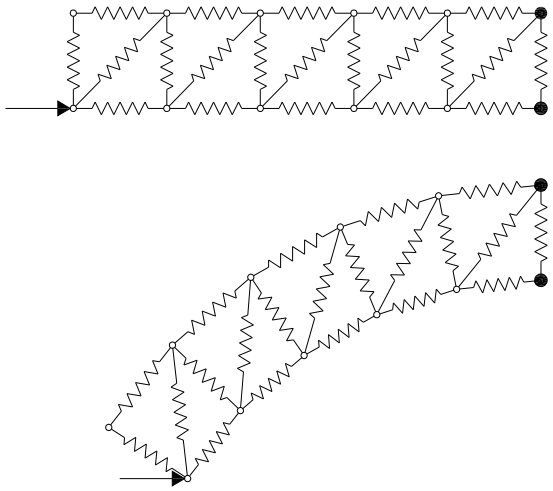

Figura 2. Sistema de masas y resortes simulando una barra sometida a compresión.

Las diferencias entre elementos más rígidos y elementos más flexibles residirán en la cantidad de resortes involucrados en su construcción, la forma en que se construya el reticulado y las constantes de elasticidad de estos resortes. Otras características, tales como la rotura, podrán simularse eliminando a los resortes en caso de que se superen ciertas tensiones máximas.

Para realizar una unión entre elementos bastará con hacer que compartan una masa en común, ya que la naturaleza axial de las fuerzas ejercidas por los resortes originará un comportamiento igual al esperado de una articulación.

REFERENCIAS

[1] http://gameplanets.blogspot.com/2007/06/physics-simulations.html

[2] http://www.codinghorror.com/blog/archives/001135.html

[3] http://www.gamasutra.com/view/feature/2798/physics_in_games_a_new_gameplay_.php

[4] http://channel9.msdn.com/posts/Charles/Brian-Beckman-The-Physics-in-Games-Real-Time-Simulation-Explained/

[5] http://repository.rigsofrods.com/

[6] http://en.wikipedia.org/wiki/The_Incredible_Machine

[7] http://www.armadillorun.com/

[8] http://www.opengl.org

[9] http://nehe.gamedev.net/

[10] http://www.cs.cmu.edu/~baraff/sigcourse/notesd1.pdf

[11] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.4074

[12] http://en.wikipedia.org/wiki/Hooke%27s_law

[13] http://ocw.mit.edu/OcwWeb/Physics/8-01Physics-IFall1999/VideoLectures/detail/embed06.htm

[14] http://farside.ph.utexas.edu/teaching/329/lectures/node35.html

[15] http://en.wikipedia.org/wiki/Stiff_equation

[16] http://www.math.buffalo.edu/306/Euler3a.html

[17] http://www.cs.cmu.edu/~baraff/sigcourse/notese.pdf

[18] http://www.jhu.edu/virtlab/bridge/bridge.htm

División de Tareas

A continuación expondremos una matriz en la que se verá plasmada la división de tareas por integrante del grupo.

	Arana, Andrés	**Illobre, Carlos**	**Raineri, Gabriel**
Definición de la Arquitectura de la Aplicación	X	X	X
Definición de Estándares de Programación	X	X	X
Diseño y Programación del Motor Físico	X		
Diseño y Programación de persistencia y serialización	O	X	
Diseño y Programación del Editor de Escenarios	O	X	O
Diseño y Programación de Biblioteca Común	O	O	X
Diseño y Programación de la Biblioteca de Juego	X		X
Diseño de la Biblioteca de Interfaz de Usuario	X		
Programación de la Biblioteca de Interfaz de Usuario	X		O
Desarrollo de Algoritmos Geométricos y Matemáticos	X		O
Diseño y Programación del Servidor			X
Diseño y Programación de Interfases Gráficas del Cliente			X
Configuración del *Build System* CMake	O	O	X

Referencias:

- X: Ejecutor Principal.
- O: Colaborador.

Si bien exponemos esta tabla de división de trabajo y tareas, es importante destacar la constante solidaridad del equipo sobre las necesidades de cada integrante. Siempre nos hemos mostrado muy abiertos y dispuestos a ayudarnos mutuamente.

Por otra parte, fue notorio el aumento de la sinergia cuando el grupo lograba reunirse en su totalidad. Lamentablemente esto no fue factible tan asiduamente como hubiésemos deseado debido a la distancia que nos separaba. De todas maneras, consideramos que hemos sabido resolver los problemas de la distancia, intentando estar en contacto tan frecuentemente como nos fue posible.

Evolución del Proyecto

Al iniciar el proyecto, hemos consensuado en trabajar con iteraciones cortas que concluyan en una versión incompleta pero funcional de la aplicación. Cada iteración duraría aproximadamente una semana y media.

En algunas ocasiones esto se pudo cumplir rigurosamente y en otras ocasiones no fue posible dado que la integración resultó más dificultosa a medida que se iba avanzando y, en muchos casos, hemos tenido que *refactorizar* porciones de código para que la calidad del diseño no se vea perjudicada.

Se comenzó el proyecto con la creación de la biblioteca de funcionalidades comunes dado que nos aportaría una vasta cantidad de utilidades que serían utilizadas en toda la aplicación.

Luego continuamos con la implementación del servidor y el cliente en lo que respecta a la funcionalidad de chat. Esto también nos daría la infraestructura necesaria para luego continuar con el resto de las funcionalidades. Esto es así dado que la comunicación entre cliente y servidor se realiza a través de comandos y el envío de mensajes de chat es simplemente uno de ellos. Esto nos sirvió de base para luego continuar agregando el resto de los comandos que constituirían las *conversaciones* entre el cliente y servidor.

Al mismo tiempo nos encontrábamos diseñando el nuevo motor físico[1] y el editor de escenarios. Finalizados ambos diseños se procedió a la implementación de cada uno de ellos.

A partir de aquí fuimos armando objetivos semanales que debían ser completados al finalizar cada domingo. Estos objetivos estaban relacionados con tareas muy puntuales y específicas acerca de lo que estabámos diseñando o implementando en cada momento.

Hemos hecho especial hincapié en las pruebas de cada porción de código implementada, para garantizar que su agregado no influya en el resto de la aplicación. Asimismo, cada prueba implicaba la utilización de la herramienta *Valgrind* para detectar posibles pérdidas de memoria o lecturas/escrituras de memoria inválidas.

[1] Decimos "nuevo" dado que se decidió reimplementar por completo la versión del motor provista por la cátedra.

Inconvenientes Encontrados

Toda funcionalidad compleja implementada trajo aparejados de alguna manera u otra ciertos inconvenientes, ya sea por integración, por liberación de memoria, por cálculos complejos que no devolvían lo que esperábamos, etc.

Uno de los primeros inconvenientes que se nos presentó fue el hecho de decidir la utilización de dos *sockets*, uno para el manejo de la comunicación exclusivamente del juego y el otro para el tráfico de *chat*. Si bien la idea nos pareció buena, el ayudante que estuvo acompañándonos en el desarrollo del presente trabajo discrepó con nuestra opinión y por eso decidimos dejar sólo un *socket*.

Muchos fueron los inconvenientes a la hora de investigar el funcionamiento del motor físico provisto por la cátedra. Una vez entendido su funcionamiento se procedió a reimplementarlo, lo cual a su vez trajo aparejado otros problemas. Esto requirió exhaustivas pruebas y la constante refactorización de ciertos aspectos para lograr mayor genericidad y prolijidad de tan complejo código.

Análisis de Puntos Pendientes

Carácter Separador

En el protocolo de comunicación se está utilizando un carácter ASCII no imprimible. Se trata del carácter '\30', *Record Separator*. Es probable que el uso de este carácter no sea seguro o bien, portable.

Nos hizo dudar el hecho de que al hacer un pequeño *debug* sobre la aplicación notamos que este carácter era convertido a '\24'. De todas maneras, no tuvimos inconvenientes porque la aplicación siempre se mantuvo coherente y consistente respecto al uso de este carácter.

La idea sería continuar con la investigación sobre la seguridad en el uso de este carácter y su eventual reemplazo en el protocolo de comunicación.

Historial de Puntajes

Es requisito funcional el hecho de que el servidor almacene un historial de los puntajes más altos.

Optimización del Motor Físico

Es nuestro deseo optimizar el motor físico y evaluar el cambio de Runge-Kutta de cuarto orden por la técnica Verlet o bien por métodos implícitos.

Validación de Elementos Visuales

La idea es validar que dos elementos colisionables no puedan ser posicionados de manera tal que se solapen.

Detención del Servidor

Proveer una manera más elegante de cerrar el servidor. Esto permitiría ejecutar el servidor en modo *background* sin mayores inconvenientes.

Implementación de Elementos Rompibles

El engine físico de la cátedra contempla la posibilidad de dar un valor máximo de esfuerzo al que puede ser sometido un resorte antes de romperse, lo que causa el colapso del mismo y la desvinculación de las partículas asociadas. Esto no es difícil de agregar al sistema físico desarrollado, de hecho es cuestión de agregar el control apropiado en el afector correspondiente (*SpringForceAffector*).

Renderers Llamativos

El *api* utilizada para dibujar los elementos en el area de juego es extremadamente versátil. Es posible mejorar muchísimo la presentación del juego implementando renderers nuevos que utilicen técnicas gráficas un poco más llamativas a las utilizadas actualmente, como gradientes de colores, texturas y sombreado.

Herramientas

Hemos utilizado diversas herramientas y bibliotecas con el objetivo de resolver distintas problemáticas.

Desarrollo

Para el desarrollo decidimos utilizar *Eclipse* como entorno integrado de desarrollo.

Dentro de lo que consideramos desarrollo, hemos utilizado la herramienta *Valgrind* para detectar posibles pérdidas de memoria y lecturas/escrituras inválidas.

Control de Versiones y Gestión del Proyecto

Para el control de versiones se utilizó el servidor SVN provisto por Assembla[2]. Asimismo, este sitio fue utilizado para la gestión integral del proyecto: creación de *Wikis*, comunicación entre los integrantes, creación de *tickets* y *bugs*, hitos, etc.

Generación de Código y Diagramas UML

Para la generación del esqueleto de las clases como así también para la creación de diagramas UML se utilizó *Enterprise Architect*.

Interfases Gráficas

Para el diseño de interfases gráficas se utilizaron un conjunto de bibliotecas multiplataforma que mencionaremos a continuación.

GLADE 3.4.5

Es una herramienta de desarrollo visual de interfases gráficas. Su utilización fue realmente muy beneficiosa puesto que nos permitió acelerar los tiempos de diseño sin necesidad de escribir código.

GTKMM 2.4

Es la versión orientada a objetos de la popular biblioteca GTK+ para la creación de entornos gráficos.

LIBGLADEMM 2.4

Esta biblioteca fue utilizada con el fin de leer los archivos XML provenientes de Glade. Estos archivos contienen el diseño de la interfaz gráfica y esta biblioteca es la que nos permite obtener objetos de Gtkmm a partir de lo definido en dichos archivos.

Otras Bibliotecas

TINYXML

Dado que el formato de intercambio entre el cliente y el servidor es XML fue necesario la lectura y escritura de este tipo de archivos. TinyXML es una biblioteca sencilla que nos proveyó las herramientas necesarias para el manejo de archivos XML. Vale aclarar que esta librería fue incluida dentro del código fuente.

[2] http://www.assembla.com

Otro requerimiento del enunciado es el hecho de permitir optar por una ejecución en modo de *debug*. Para ello utilizamos este logger que da la posibilidad de imprimir por pantalla o a un archivo de *log* aquellos eventos críticos.

Compilación y Sistemas de Construcción

Una de las problemáticas iniciales con las que el grupo debió lidiar fue el hecho de la colaboración a distancia. Es decir, desarrollar el presente trabajo sin necesidad de estar constantemente reunidos. Para ello se debió optar por utilizar un estándar en lo que respecta a la construcción de la aplicación.

Los sistemas de construcción[3] resultaron ser muy útiles a la hora de generar los ejecutables de la aplicación, logrando que la construcción sea relativamente portable, al menos entre ambientes Unix.

La herramienta que se eligió fue CMake en su versión 2.6.2. Su uso nos simplificó considerablemente los problemas típicos de compilación de una aplicación con cierta envergadura.

Pruebas

Inicialmente se conversó acerca de la posibilidad de utilizar algún *framework* de pruebas como por ejemplo *cppUnit*. Si bien las ventajas son innumerables, creímos conveniente por una cuestión de tiempo, el hecho de crear pequeños programas de prueba.

[3] En la jerga informática son más conocidos como *Build Systems*.

Conclusiones

Si bien el presente trabajo ha requerido de un constante esfuerzo por parte de todos los integrantes del grupo, no vacilamos a la hora de destacar todos los conceptos y conocimientos aprendidos durante el desarrollo del mismo.

No caben dudas de que sin el esfuerzo de todos los integrantes y de la sinergia del grupo, el presente trabajo no hubiese sido posible de ningún modo. Es gracias a esto que hoy podemos hablar de este trabajo como un producto cerrado y con una calidad adecuada para los plazos del proyecto.

Es realmente notorio el cambio de panorama desde que comenzó el proyecto hasta el día de la fecha. Comenzamos intentando bosquejar cómo sería nuestro trabajo, cómo iríamos dividiendo las tareas, con una vasta cantidad de interrogantes respecto a cuestiones técnicas, de diseño e, inclusive, lógicas. Luego fuimos avanzando lenta pero sostenidamente sobre los distintos desafíos que nos propuso este trabajo. Las dudas fueron disipándose a medida que investigábamos ciertos aspectos e íbamos plasmando en el código las ideas que surgían.

Por otra parte, los tiempos de desarrollo fueron acortándose a medida que pasaba el tiempo, fundamentalmente por dos razones: habíamos conseguido un buen conocimiento respecto del campo en el que nos encontrábamos y cada vez la infraestructura armada era mayor. Esto nos posibilitó efectuar pequeñas modificaciones pero con fuertes impactos funcionales y gráficos.

Una vez más, si bien el esfuerzo fue desmesurado, ningún integrante del grupo siente arrepentimiento alguno por haberlo hecho. Es más, sentimos que hemos aprendido en proporción directa al esfuerzo.