Micromachines

Ejercicio Final

Objetivos	● Afianzar los conocimientos adquiridos durante la cursada. ● Poner en práctica la coordinación de tareas dentro de un grupo de trabajo. ● Realizar un aplicativo de complejidad media con niveles aceptables de calidad y usabilidad.
Instancias de Entrega	**Pre-Entrega:** clase 14 (12/11/2019). **Entrega:** clase 16 (26/11/2019).
Temas de Repaso	● Aplicaciones Cliente-Servidor multi-threading. ● Interfaces gráficas con *gtkmm/cairo/SDL/qt* ● Manejo de errores en C++
Criterios de Evaluación	● Criterios de ejercicios anteriores. ● Construcción de un sistema Cliente-Servidor de complejidad media. ● Empleo de buenas prácticas de programación en C++. ● Coordinación de trabajo grupal. ● Posibilidad de utilizar y generar bibliotecas de terceros ● Planificación y distribución de tareas para cumplir con los plazos de entrega pautados. ● Cumplimiento de todos los requerimientos técnicos y funcionales. ● Facilidad de instalación y ejecución del sistema final. ● Calidad de la documentación técnica y manuales entregados. ● Buena presentación del trabajo práctico y cumplimiento de las normas de entrega establecidas por la cátedra (revisar criterios en sitio de la materia).

Índice

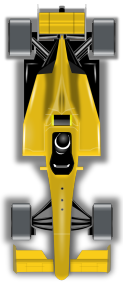
Introducción

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

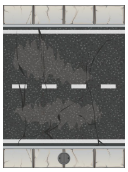
La tribuna a veces arroja ayuda a sus competidores favoritos. Estas ayudas pueden ser:

- 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/