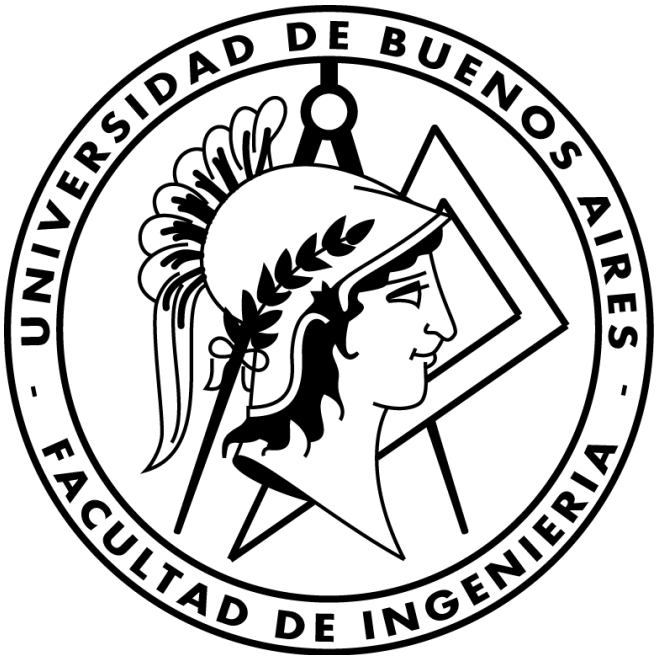
Trabajo Práctico Final

Remake de juego Worms en C++

75.42 - Taller de Programación

Manual de Proyecto



Integrantes:

ALVAREZ WINDEY ARIEL JUSTO – 97893

DIZ GONZALO – 98546

ROBLES GABRIEL – 95897

Índice

1 División de Tareas....3
Responsabilidades particulares....3
Responsabilidades compartidas....3
2 Evolución del Proyecto....3
Semana 1 (01/05/2018)....3
Semana 2 (08/05/2018)....3
Semana 3 (15/05/2018)....3
Semana 4 (22/05/2018)....4
Semana 5 (29/05/2018)....4
Semana 6 (05/06/2018)....5
Semana 7 (12/06/2018)....5
Semana 8 (19/06/2018)....6
3 Inconvenientes encontrados....7
Lag en el cliente....7
Alto consumo de cpu en servidor....7
4 Herramientas utilizadas....8
IDE....8
Dibujos y animaciones del juego....8
Interfaces de usuario (lobby y editor)....8
Parser....8
Control de versiones....8
5 Análisis de puntos pendientes....9
6 Conclusiones....10

1 División de Tareas

Responsabilidades particulares

ALVAREZ WINDEY ARIEL JUSTO	Entendimiento y uso de SDL. Responsable de todos los dibujos, animaciones y sonidos (client-side).
DIZ GONZALO	Entendimiento y uso de Box2D. Responsable de la física (server-side).
GABRIEL ROBLES	Entendimiento y uso de QT5. Responsable de integración cliente-servidor.

Responsabilidades compartidas

Entendimiento y uso de parser *yaml-cpp*. Testing y reporte de bugs.

2 Evolución del Proyecto

Semana 1 (01/05/2018)

- Creación de repositorio Github
- Instalación de Box2D
- Creación de checklist a partir de enunciado

Semana 2 (08/05/2018)

- Diagrama de ventanas para el cliente
- Creación de Lobby con QT5
- Instalación de SDL2
- Instalación de yaml-cpp
- Pruebas de concepto con SDL y Box2D
- Organización de folders del proyecto en client – common – editor y server. Creación de archivos cmake para compilar las aplicaciones por separado

Semana 3 (15/05/2018)

- Creación del protocolo para enviar archivos .yml por socket
- Implementación de clase blocking queue
- Modelado de worms y vigas en Box2D
- Implementación de cámara y dibujos

- Integración de cámara y dibujos con lógica de cliente
- Implementación de cola bloqueante de eventos en client-side
- Definición de estructura yml de mapas y snapshots
- Implementación de threads con sus respectivas tareas tanto en client-side como en server-side
- Dibujo de worms, inventario, vigas y agua en client-side

Semana 4 (22/05/2018)

- Testing, fixes y optimizaciones de cpu y memoria
- Renderizado de nombres de worms y vidas
- Implementación de animaciones
- Primera integración cliente-servidor. El cliente logra dibujar lo que el servidor le indica mediante snapshots
- Modelado de clase Evento (que envía el cliente al servidor)
- Creación de lógica de equipos y turnos en server-side
- Creación de editor de mapa
- Modelado de agua en Box2d
- Agregado rozamiento en vigas
- Agregadas funciónes de caminata y salto frontal y posterior de gusanos
- Creada primera arma: dinamita
- Implementado daño y muerte de gusanos

Semana 5 (29/05/2018)

- Refactors y fixes tanto en cliente como en servidor
- Selección de armas con click en el inventario
- Dibujo de cronómetro
- Detección de equipo ganador, equipo perdedor o empate al finalizar una partida
- Integración de editor a pantalla de QT5
- Agregado de sonidos al editor
- Implementación de bate y granadas
- Se dibujan explosiones

- Explosiones con sonidos
- Logica de countdown en proyectiles para que el cliente lo dibuje
- Implementación de la mira
- Clase global de configuración del servidor, cargada con valores hardcodeados
- Los gusanos mueren al tocar el agua

Semana 6 (05/06/2018)

- Logicas de cambio de turno al disparar, gusano protagónico herido o timeout
- Agregados límites físicos al mapa (a lo ancho) que matan al gusano
- Dibujos de granadas y holy bomb
- Lógica en la fisica de la intensidad de disparo
- Mapas generados como archivos .tar.gz
- Física de rebote a las granadas
- Implementación de Bazooka
- Implementación de partidas en espera en el lobby
- Dibujo de potencia variable en disparo
- Mejoras para mayor eficiencia de servidor. No se envían snapshots repetidas
- Dibujo de ataque aéreo
- Integración de cliente con la lógica de lobby
- Integración de lógica de juego en server-side con lógica de servidor propiamente dicho (aceptar conexiones, crear partida, administrar jugadores en la partida, lanzar juego, etc)
- Dibujo de nuevas animaciones (worms afectados por explosión, worms cayendo)

Semana 7 (12/06/2018)

- Testing, factor y fixes
- Mensajes de victoria, empate y derrota al finalizar el juego
- Dibujo de vida total de cada equipo
- Regreso ordenado al lobby tras finalización de una partida
- Nuevas animaciones (gusanos caminando en superficies inclinadas)
- Finalización ordenada de la partida si un jugador abandona la misma
- Cámara enfoca proyectiles y sigue a gusanos afectados por explosión

- Cámara enfoca a worm protagónico
- Animación y sonido de gusanos muriendo
- Lógica de descontar municiones al disparar
- Implementado cálculo correcto para el daño de las explosiones
- Configuracion global del servidor se puede cargar desde un archivo yml
- Dibujo actualiza nuevas cantidades de munición. Borra dibujo de arma si no tiene munición
- Servidor conoce el arma pickeada por un protagonic worm

Semana 8 (19/06/2018)

- Generación de manuales de documentación
- Optimizacion de uso de cpu y memoria del editor
- Servidor se puede detener apretando la ‘q’
- Clase global de configuración para el cliente
- Editor permite crear mapas con hasta 99 gusanos por team
- Teams con menos gusanos tendrán mas vida por gusanos
- Instalador de juego
- Musica de ambiente de fondo
- Botón de save y botón de exit en el editor
- Sonido de worm caminando
- Sonido de worm cuando cambia el turno
- Editor permite abrir un mapa y continuar con su edición
- Clase de flash messages dibujables en SDL
- Editor sabe si hay cambios sin guardar durante la edición

3 Inconvenientes encontrados

Lag en el cliente

Se percibía lag en el cliente, dado que al ejecutar una acción (ejemplo, caminar o saltar), la misma se dibujaba luego de un intervalo de tiempo inaceptable (y cada vez mayor). Se llegó a la conclusión de que el cliente demoraba más en dibujar que en recibir por socket los snapshots que enviaba el servidor (que se acolaban).

Para ello, se implementaron delays dinámicos en el main-loop del juego del cliente. La idea fué renderizar 60 veces por segundo, y los delays se hicieron en forma dinámica, midiendo los tiempos de dibujado y de sleep (el SO no siempre devuelve el control en un sleep tras el intervalo indicado). Esto fué acompañado con una estratégia adicional: El main-loop del cliente procesa en un ciclo todos los snapshots de la cola hasta que no haya más, y luego renderiza el último estado procesado. De esta forma, todos los snapshots son procesados y no se pierde ninguna información.

Con esta combinación de estrategicas logramos corregir el problema de lag en el cliente.

Alto consumo de cpu en servidor

Se percibía un alto consumo de cpu en el aplicativo del servidor, del orden de 50%. Considerando que el servidor unicamente administra partidas y ejecuta la simulación de la física, estos valores son considerados altos.

Se tenía que los snapshots (fotografías que toma el servidor de todos los objetos físicos dinámicos) se guardaban y actualizaban en memoria como objetos YAML (de la librería yaml-cpp, ver herramientas utilizadas), realizando iteraciones sobre nodos de este tipo de objeto. Esto se hacía 60 veces por segundo (que es la cantidad de snapshots que toma el servidor).

En lugar del anterior esquema, se crearon mapas clave valor para tener en memoria los estados de los objetos dinámicos, y las iteraciones se realizan sobre este tipo de contenedores. A la hora de tomar el snapshot, se vuelca la información de estos contenedores a un objeto YAML mediante YAML Emitter, una herramienta de la librería que encontramos mucho mas performante que iterar y editar objetos YAML.

Además se modificó el servidor para que únicamente saque snapshots cuando haya algun cuerpo (entidad física) que se haya cambiado su posición, explotado, etc. No se sacan ni envían snapshots repetidas.

Tras realizar estas modificaciones, el servidor consume picos de 10% de cpu.

4 Herramientas utilizadas

IDE

Para el proceso de desarrollo todos los miembros utilizamos el entorno de desarrollo Visual Studio Code.

Dibujos y animaciones del juego

Todos los dibujos y animaciones del juego que pueden apreciarse en el cliente, fueron realizados con la ayuda de la biblioteca SDL2.

Interfaces de usuario (lobby y editor)

Toda interfáz de usuario (ventanas, botones, etc) fueron realizadas con la ayuda de la biblioteca QT5 y su herramienta de edición con interfáz gráfica QT5 Creator 4.6.1.

Parser

Todos los archivos de configuración, así como los mensajes de cliente a servidor y de servidor a cliente son de formato YAML. Para su creación, lectura y edición se utilizó una bibliteca denominada yaml-cpp creada por el usuario jbeder de github.

Control de versiones

Como herramienta de control de versiones utilizamos Github. El repositorio donde puede verse el desarrollo desde el día cero es https://github.com/gabyrobles93/tp-final-taller

5 Análisis de puntos pendientes

- Equipos personalizados

Que cada jugador tenga su o sus equipos personalizados, con los nombres de worms que él configure.

- Guardar mapa jugado

Actualmente cuando nos unimos a una partida, el servidor nos envia el mapa para que podamos dibujarlo y jugarlo. Pero dicho mapa es considerado un “archivo temporal”, que es borrado tras la finalización de la partida. Vemos como un punto pendiente hacer que dicho mapa pueda guardarse en una carpeta junto con el resto de los mapas del jugador, para que pueda ser editado y usado en otra partida.

- Dibujado de equipamiento de arma

Los worms no se dibujan con su arma seleccionada. Vemos como un punto pendiente hacer que un gusano protagonista, equipado con determinada arma, se dibuje con esa arma equipada. Para eso hay que hacer ciertos cambios al protocolo, pues todos los clientes jugadores de la partida deben enterarse qué arma está portando el gusano protagonista.

- Proceso de QA mas intenso

Por los tiempos que se manejan en el desarrollo de este trabajo, no hemos podido dedicar abundante tiempo a un proceso de testeo y depuración. Esto hace que existan algunos crasheos, mas que nada cuando finaliza la partida y se debe regresar al lobby, pero en lineas generales se pueden jugar partidas muy divertidas sin mayor problema.

6 Conclusiones

Consideramos factores clave el sistema de control de versiones github, que nos permitía integrar nuestro código como así dejar documentados cambios, issues, fixes, progresos y demás, como así también la biblioteca yaml-cpp empleada, que nos facilitó el parseo de mensajes entre cliente y servidor.

La división de tareas que propusimos, fué diferente a la sugerida en el enunciado, pero creemos que fué adecuada y repartimos equitativamente el trabajo.

Asímismo, creemos que para un próximo desarrollo de estas dimensiones es conveniente separar el proyecto en etapas, y realizar una planificación de cada etapa. Por no realizar esto, nos hemos tenido que enfrentar a muchos refactors en la lógica de algunos procesos.

Los integrantes de este equipo estamos de acuerdo en que la experiencia de realizar este trabajo fue muy satisfactoria. En particular, fue nuestro primer trabajo en equipo de desarrollo de software de tanta complejidad y que requiera tanta coordinación.