FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Argentum - Trabajo Final
Manual de Proyecto

Taller de Programación 1
Primer cuatrimestre de 2020

Alumno	Número de Padrón	Email
Ruiz, Francisco	99429	fran7ruiz9@gmail.com
Belosevich, Victor	97757	victor.belosevich94@gmail.com

Índice

1. División de Tareas

Francisco Ruiz

- Interfaz gráfica.
- Clase RAII de los elementos básicos de SDL.
- Manejo de Cámara.
- Lógica de estados de los personajes del lado del cliente, inputs permitidos en cada uno.
- Manejo de eventos de usuario.
- Animación de armas, cuerpos, cabezas, sombreros y escudos.
- Sonidos y Animación de Ataques.
- Lógica de Fair Play.
- Equipación y desequipación de Items del lado del cliente.
- Profesiones de los NPC en el server.
- Sistema de comunicación en Cliente.
- Protocolo de comunicación, encoding y decoding de comandos, e información del estado del juego.

Victor Belosevich

- Carga de Mapas.
- Lógica de movimiento de los personajes y NPC.
- Lógica de estados de los personajes del lado del servidor.
- Lógica de celdas, posiciones y colisiones.
- Carga de configuración desde archivo json.
- Lógica de las criaturas, spawneo, ataque, persecución y estados.
- Lógica de Ataque y Defensa. Drops al morir.
- Equipación y desequipación de Items del lado del servidor.
- Sistema de comunicación en Servidor.
- Protocolo de comunicación, encoding y decoding del mapa.
- Lógica de Resurrección, regeneración de vida/mana, meditación.

2. Inconvenientes encontrados

2.1. Tablero de Posiciones en Server

Al principio se optó por un sistema de posiciones, que no hacía uso de un tablero. Esto hizo difícil obtener las posiciones adyacentes a la actual, determinar el rango en el cual se podía atacar o interactuar y el proceso de determinar las colisiones resultaba demasiado costoso ya que había que verificar las mismas contra todos los objetos del mapa, sean estáticos o no. Al ver todos estos problemas, se optó por no seguir con este desarrollo y avanzar hacia un refactor, en el cual estas operaciones no fueran tan costosas. La solución fue cambiar a un sistema de tablero en el que el mapa está dividido en fracciones fijas de terreno y que cada fracción sea mapeada a una celda, la cual contiene información relevante del terreno al cual fue asignada. Utilizando este formato, las detección de colisiones pasó a ser un proceso mucho más simple ya que solo debemos chequear la celda a la cual se quería mover el personaje. Los demás problemas antes mencionados también se solucionaron de una manera eficaz, permitiendo terminar el resto de los features de forma más amena.

2.2. Interfaz de Usuario

La inexperiencia de ambos en la interfaz de usuario generó que por momentos el avance sea muy lento. Las animaciones de los personajes y los distintos items generó complicaciones hasta que el movimiento observado sea fluido y que la actualización de las mismas se haga de manera correcta. Lo mismo sucede al realizar el resize del juego que la interfaz se ve levemente afectada.

3. Análisis de puntos pendientes

3.1. Redimensión de la UI

Como puntos pendientes tenemos la corrección del renderizado cuando se juega fullscreen. La cámara al agrandar el tamaño de la ventana, nos muestra más porción del mapa que al tener una ventana más pequeña. Se debería haber generado una escala para que se grafique correctamente, pero al intentar implementarlo, los tiles del mapa, quedaban muy distorsionados, así que se prefirió evitar esa implementación.

3.2. Características de Raza y Clase

En la UI, no se puede apreciar las características propias según la raza y la clase que fueron seleccionadas al iniciar el juego y como va variando según su avance en el juego. Este feature no fue agregado por falta de tiempo, se priorizaron corregir y completar otras áreas del trabajo.

4. Herramientas

4.1. Tiled

Editor de mapa utilizado para generar el mapa estático con tiles. El archivo json resultante es el que el servidor levanta para crear el mapa. Además esta herramienta fue utilizada para generar los objetos estático, como los árboles, casas, paredes, NPC y el nido de los cuales las criaturas se van spawneando.

4.2. Visual Studio Code y CLion

CLion es un IDE de *JetBrains* y **Visual Studio Code** es un editor de *Microsoft*. Ambos permiten compilar y debuggear la aplicación. En este caso utilizamos de base un archivo CMake.

4.3. Control de Versiones

Para realizar el control de versiones se utilizó **git**. El repositorio está alojado en GitHub.

4.4. Documentación

La presente documentación fue generada con LaTex, desde la web de Overleaf. Además los diagramas UML fueron hechos con *draw.io*.