Rodrigo Cortés 21.594.212-0 Jaime Araya 21.652.246-K

(ACLARACION, en este repositorio puede que aparezcan 2 ramas, la default y la main, la más actual es la main ya que la otra fue antes de que supiéramos que se podía hacer un main y ahí se ven reflejados cambios anteriores) #Take your meds Bienvenido a Take your meds, un RPG por turnos Jugador vs Jugador (por ahora) donde podrás luchar contra tus demonios internos con una música de fondo emocionante

##Requisitos Visual Studio 2022 (recomendable) (Puede que al importar el repositorio puede que pida descargar algunas dependencias de Visual Studio las cuales no usamos en su mayoría, pero no sabíamos cómo quitarlas)

##Paso a paso para compilar

Para compilar correctamente este juego (recomendamos usar Visual Studio), probablemente tirara error de que no encuentra la librería de SFML/grafics...etc, también recomendamos exportarlo a Visual Studio desde Github Desktop

1-para solucionarlo hay que ir a propiedades en el proyecto en Propiedades de configuración---> C/C++ ---> General y dentro donde dice directorios de inclusión adicionales remplazar lo que haya con: $(ProjectDir)external\SFML\include y aplicar.

2-Luego también en Propiedades de configuración ---> Vinculador ---> General y dentro donde dice directorios de biblioteca adicionales remplazar lo que haya con: $(ProjectDir)external\SFML\lib y aplicar

3-Y por último ir a las propiedades del trabajoestructura.cpp y hacer lo mismo que en el paso 1 con eso debería estar listo y compilable

##Como Jugar -Al abrir el juego se mostrarán 2 opciones, Jugar, configuración (No funciona por ahora) y salir.

-Usa las flechas del teclado para moverte entre opción y pulsa entre para seleccionar una opción. -Al seleccionar la opción de "Jugar" se mostrará el tablero con todos los personajes (Verdes para el J1 y rojos para el J2), con el personaje en turno resaltado con una flecha en su barra de vida y su nombre escrito dentro del círculo. Como también 2 opciones, Atacar y Habilidad.

-al ser por turnos, el personaje con mayor velocidad atacara primero, puedes elegir entre un ataque básico que no consume Mana o una habilidad que gastara mana.

Para atacar debes seleccionar a quien quieres hacerlo (No puedes atacar personajes de tu mismo equipo, a ti mismo ni tampoco espacios en blanco), aunque las habilidades pueden no requerir un objetivo.

-El daño se muestra por consola

-Si te arrepientes y no quieres atacar si no, usar una habilidad o viceversa, pulsa la tecla ESC para volver atrás, así también para volver al menú inicial -El juego se acaba cuando todos los personajes de un equipo hayan muerto

##Estructura del código

###Clase Habilidad Esta clase es padre de la clase Buff y la clase Skill, contiene los atributos (Nombre, tipo y descripción) contiene tanto los setters como getters

###Clase Buff

Esta clase es una clase que hereda de la Clase Habilidad y contiene los atributos (Nombre, tipoBuff ,descripción y coste mana). Dentro contiene métodos para usar el buffo dependiendo del nombre.

###Clase Skill

Esta clase es la principal para las habilidades, y es hija de la clase Hbilidad, contiene los atributos (Nombre, Tipo, descripción, coste, efecto, potencia y acierto) con sus getters y setters, además del void usarSkill

###Clase Personaje

Esta clase es la que representa un personaje, contiene los atributos (nombre, tipo, control, vidaMax, ataque, defensa, velocidad, mana, sangrando, vivo, aturdido, vector<Habilidad*> habilidades, pair<int, int> posición, manaMax, string sprite)
###Clase MenuState

Esta clase es la encargada de inicializar el menú y cargar la fuente, creando las opciones (Jugar, Configuración y Salir).

-Void Handlevent: Maneja los eventos desde teclado en el menú, ya se pasar al estado GameState o cerrar el juego.

-Void update: Cambia el color de la opción actualmente seleccionada.

-void Render: dibuja el menú actual.

-Bool Debesalir: devuelve true si el usuario marco la opción salir.

- void reproducirCancionMenuAleatoria: Elige una canción aleatoria.

###Clase GameState

Esta clase administra la lógica principal del juego utilizando SFML. Controla el estado del juego, la interacción del usuario, el sistema de turnos, la ejecución de habilidades, la navegación por los menús y el renderizado de los elementos visuales. -Constructor tiene los siguientes atributos (RenderWindow& window, StateManager& stateManager, vector<Personaje*> personajes, Personaje* (&tablero)[8][8], queue<Personaje*> orden_Turno)

-En el constructor se cargan la mayor parte de los Sprite para poder dibujarlos luego, así como también la música y algunas posiciones iniciales para poder dibujar en ciertos lugares. También se llama al método reproducirCancion() y el inicializarOrdenTurno().

-void handlevent: Este metodo se encarga de reconocer los inputs que manda el jugador, como seleccionar la opción de ataque, habilidad y confirmar objetivo enemigo.

-void handleGameOverInput: maneja la entrada cuando el juego haya terminado

- void handleActionMenuInput: se encarga del movimiento por el menú de acción y de la confirmación de la selección de la opción.

-void handleAttackInput: Entrada en modo ataque: mover cursor y atacar.

-void handleSkillSelectInput: Entrada en menú de selección de habilidad.

-void handleSkillConfirmTargetInput: Confirmación del objetivo para skills tipo "Single" (de un solo objetivo)

-void actualizarObjetivosValidosBasico: Valida cada uno de los personajes para saber si es posible atacarlos

-void cargarSkillsDelPersonaje: Carga las habilidades del personaje en turno para que aparezcan en el menú.

-void verificarFinDelJuego: Verifica si quedan personajes vivos de ambos equipos, si no hay devuelve el ganador y termina el juego

-void pasarTurno: Este metodo se encarga de pasar el turno y si es turno de la IA llama a la IA para aplicar el algoritmo minimax.

Método render:

El render es el que se encarga de mostrar todo el apartado gráfico del juego. Funciona utilizando un framebuffer interno, el cual constantemente está transformando sprites, texturas, fuentes, etc., en instrucciones para que OpenGL se encargue del resto. Esto, dependiendo del framerate, ocurre x frames por segundo. Por ejemplo, Take Your Meds corre a 60 frames por segundo, lo que significa que se actualiza 60 veces por segundo. En nuestro código, el render trabaja los sprites, texturas, fuentes y demás.

Método usarSkill:

El método usarSkill se encarga de manejar el uso de habilidades, verificando si el lanzador tiene suficiente maná para utilizarlas y, en caso de ser así, restándole el maná correspondiente. También aplica el daño o efecto a los personajes afectados, según el tipo de habilidad, ya sea que impacte a un enemigo, a varios o a todos los personajes en pantalla. Además, gestiona las probabilidades de acertar o fallar y finalmente verifica si los enemigos sobreviven o mueren.

Método getObjetivosPorEfecto:

Este acompaña al método anterior. Se encarga de ver quiénes son los afectados por los distintos tipos de habilidades, si acaso afectan a un solo personaje, a varios o a todos. Básicamente funciona iterando sobre cada personaje, comprobando si son válidos, están vivos y no son compañeros (esto se ignora en el caso de una habilidad de tipo “Total”. Trabaja añadiendo los posibles objetivos a una lista del mismo nombre.

Método reproducirCancionDeCombateAleatoria:

Método muy simple que genera un número al azar, el cual sirve como índice para elegir una canción de la lista de canciones disponibles para el combate.

Método crearSimulacion:

Este método crea una versión alterna del estado actual del juego. Es una función const, por lo que no modifica nada realmente. Toma una versión extremadamente simplificada del estado actual y la copia creando un GameStateSim. Además, itera sobre los personajes del GameState para crear copias super simplificadas, que servirán en las simulaciones.

Método minimax (con poda alfa-beta):

Algoritmo minimax que se encarga de recorrer el árbol de estados posibles de manera recursiva hasta una profundidad definida o hasta el fin de la partida (simulada). Dentro de este árbol, cada nodo representa un estado posible del juego. Al algoritmo se le asigna este nombre debido a cómo funciona, simulando los posibles estados tanto de la IA como del oponente (jugador), donde:

- El maximizador busca maximizar su puntuación (en este caso, la IA).
- El minimizador intenta minimizarla (el oponente).

Los estados son evaluados por la función evaluarEstado, la cual, a rasgos simples, les asigna un puntaje para que así la IA pueda saber cuáles son más favorables. Por otro lado, la Poda Alfa-Beta se encarga de optimizar el minimax, asegurándose de que las opciones no favorables no sean exploradas, reduciendo así el tiempo que la IA tarda en encontrar su mejor opción. Para esto se utilizan dos variables de control:

- El alfa corresponde al mejor valor encontrado para la IA.
- El beta corresponde al mejor valor encontrado para el jugador.

Con esto, cuando se encuentra una rama que no ayudará al resultado final, se evita directamente su exploración. Finalmente, el método retorna un entero correspondiente a la mejor puntuación posible para la IA en su estado actual.

Método obtenerMejorAccion:

Este método es el más importante en la toma de decisiones de la IA. Es el método en el cual se generan todos los estados posibles del juego actual utilizando el GameStateSim. Estos se guardan en una lista; cada uno representa lo que pasaría si la IA realiza una acción específica. Cada estado posible es enviado al algoritmo minimax para que lo evalúe. Dependiendo de si mejora la puntuación, se guarda la

nueva mejor puntuación y el estado que la consiguió. Una vez se evalúan todos los casos posibles, se retorna el mejor estado.

Método evaluarEstado:

Este método recibe un GameStateSim, el cual evalúa, dándole puntaje o quitándoselo (recompensando o penalizando) dependiendo de factores como los enemigos (personajes del jugador) vivos, personajes de la IA restantes, puntos de vida restantes, velocidades, etc. Mientras mayor sea el puntaje conseguido, mejor para la IA; en caso contrario, peor. Finalmente, el método devuelve la puntuación que obtuvo ese estado.

Método actualizarEstadoConSimulacion:

Este método es probablemente el más simple en el proceso de toma de decisiones de la IA. Recibe un estado de juego simulado y se encarga de aplicar las decisiones de la IA, efectivamente atacando, disminuyendo vidas, etc. En esencia, lo que hace es efectuar el turno de la IA, para luego pasar el turno al siguiente.

-void inicializarOrdenTurnos: Ordena a los personajes en una queue según su velocidad y si está vivo.

###Clase GameStateSim

Esta clase es la copia simplificada del GameState con todo lo que necesita la IA para los datos del juego ayuda a la IA a simular escenarios para hacer la mejor elección de jugada para el mimimax

-void estaTerminada: pregunta si el estado del juego termino. -void

avanzarTurno: simula el avance de los turnos del juego.

###Clase main trabajoEstructura

En esta clase está el main que llama a todo el resto de las clases para inicializar el juego, también contiene 4 métodos.

1-Calcular Orden: Este método calcula el orden en que los personajes atacan según su velocidad y los ordena en una cola para el orden de turno.

2-Ubicar Personaje: Este método posiciona cada uno de los personajes en una matriz a través del pair del personaje

3-Crear Personajes: Este método se usa para crear los 8 personajes del juego con sus habilidades y estadísticas.

Análisis de la IA (Con poda alfa beta)

La IA tarda un promedio de 0,0005444 segundos (5444 microsegundos) en encontrar su mejor movimiento cuando todos los enemigos están vivos, a medida que la cantidad de personajes en el tablero va disminuyendo, el tiempo que tarda la IA en encontrar su mejor movimiento también disminuye alcanzando tiempos bajísimos como 0,000064 segundos (64 microsegundos), claramente estos números pueden variar dependiendo de la potencia del procesador, y además de como Windows decida manejar los hilos del procesador, considerando que durante la ejecución los procesos pelean por quien "se lleva el hilo".