PROYECTO DE JUEGO MULTIJUGADOR

“THE MAZE RUNNER”

NOTA: Con la fabricación de este videojuego no se pretende nada más que llevar a las personas un instinto de supervivencia, lejos de incentivar la violencia, la simple idea de que por mucho que se pase siempre es posible encontrar una salida.

##DESCRIPCIÓN

Este proyecto es un juego multijugador con temática de escape de laberinto inspirado en la saga de películas de “The Maze Runner” y elaborado en UNITY_ENGINE (**FIG.0**),

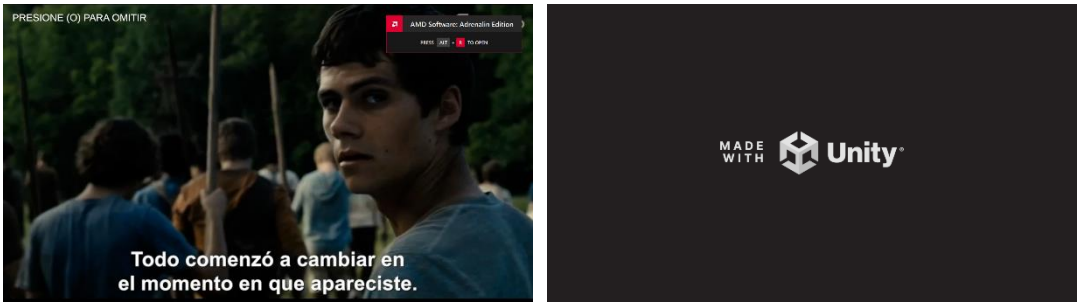

FIG.0

y con un ambiente dinámico que va desde un estilo Pixel-Art a un estilo realista. El objetivo principal es que los jugadores manejen héroes, ¿Héroes, qué clase de héroes?, pues sí , el videojuego cuenta con 6 héroes los cuales constan de una habilidad especial la cual se ve afectada por un tiempo de enfriamiento (k) que evita la utilización de la habilidad durante los siguientes k turnos luego de utilizar la habilidad, también consta con una velocidad que dicta la cantidad de casillas que podrá desplazarse dicho rol, el jugador tiene la posibilidad de escoger con que héroes va a jugar, como también introducirá su nombre **FIG.1** ,

FIG.1

pues cada jugador moverá sus héroes a través de un tablero generado dinámicamente, mientras enfrentan obstáculos, trampas o simplemente objetos de la vida diaria que penalizan o catalizan sus habilidades para moverse, y exploran el mapa para alcanzar el tele transportador que finalmente acaba la partida **FIG.2.**

FIG.2

El jugador que logre sacar al menos un héroe del tablero gana (**FIG.3**), parece sencillo, dado que cada jugador deberá trazar su propia estrategia para poder moverse por el tablero y así evitar de forma óptima los obstáculos para un mejor avance. **El juego está diseñado para ser jugado por exactamente dos jugadores quienes se irán alternando por un sistema de turnos, cada jugador tendrá la posibilidad de clicar un héroe al inicio de su turno y realizar alguna operación FIG.2. La descripción de cada héroe, trampa o ítem se muestra en pantalla como se muestra (**FIG.2**), también consta de una tienda en la cual con dinero colectado en el laberinto en el cual el jugador se podrá comprar potenciados que aumente su energía, así como artefactos mágicos que lo puedan sacar de una situación de peligro **FIG.3**.

FIG.3

##REQUISITOS: **El juego necesita de una resolución de pantalla de 1920x1080 o mayor.

##FUNCIONALIDADES E IMPLEMENTACIONES

-1 ****Generación dinámica del tablero****: El tablero es rectangular y de tamaño regular. Se generan casillas con objetos aleatorios, tales como:

-2 ****Obstáculos****: Impiden el paso de las fichas. Se implementaron alrededor de 4 tipos de obstáculos

-3 ****Trampas****: Imponen penalizaciones a los héroes que las pisan. Se implementaron alrededor de 10 tipos de trampas.

-4 ****Potenciadores****: Imponen bonificaciones a los héroes que las pisan. Se implementaron alrededor de 6 tipos de potenciadores

-5 ****Héroes****:

- El jugador tiene un conjunto de héroes (6) para escoger (hasta 3) antes de iniciar la partida.

- Los héroes tienen características definidas como **habilidad especial**, **tiempo de enfriamiento** (impide que la habilidad se use durante un número determinado de turnos) y **velocidad** (indica cuántas casillas se puede mover el héroe en un turno).

NOTA: Las habilidades pueden incluir poderes como moverse más rápido, saltar obstáculos, entre otros. Estas habilidades no son letales, es decir, las fichas no pueden morir permanentemente.

***Para la elaboración de los objetos se utilizaron clases que heredan de ScriptableObject, necesarias para tener objetos en físico y definirle propiedades especificas **FIG.4** y **FIG.4.1.**

```
public class Trap: ScriptableObject
{
    public string Name;
    public int Penalty;
    public Sprite TrapPhoto;
    public string Description;
    public AudioClip audioClip1;
    public AudioClip audioClip2;
}
```

FIG.4

```
public class Hero: ScriptableObject
{
    public new string name;
    public Sprite heroPhoto;
    public Hability hability;
    public int coolingTime;
    public int speed;
    public int life;
    public string habilityDescription;
    public AudioClip audioClip;
}
```

FIG.4.1

Se necesitó de prefabricados para poder instanciar los objetos(TRAMPAS, HÉROES , ITEMS , BLOQUES , HIERBA) desde el Backend. Para la generación del tablero se utiliza una estrategia de Backtrack con un principio de aleatoriedad para hacer de esta generación una generación dinámica y que cada vez que se juegue se tenga un tablero nuevo, para esto se dispuso de una clase

```
public class MazeGenerator
```

y un método principal llamado en cuanto se carga la escena del juego, encargado de asignar las tareas a los métodos auxiliares

```
public static void Starting()//preparar la escena del juego(laberinto)
{
   GenerateMaze();//generar el primer laberinto
   while(!IsValid()) //verificar si el laberinto es válido
   {
       maze = new bool[n,m];//volver a inicializar la matriz estática
       GenerateMaze();//generar otro laberinto
   }
   PrintMaze();//una vez el laberinto está listo se imprime en la escena
}
```

Una vez instanciado el laberinto en la escena se necesita instanciar los héroes correspondientes, trampas u objetos de manera aleatoria con los métodos

```
public static void GenerateTeleports(int x , int y , bool current)//salidas
```

```
public static void GenerateHeros() // Genera los héroes correspondientes
```

```
public static void PrepareTraps(int k, bool init) // instanciar las trampas
```

Una vez el tablero está del todo listo, en su turno, un jugador puede:

=> Elegir una héroe para moverlo dentro de su alcance según su velocidad. ¿Cómo se implementó esto?. Se necesita saber qué tecla se ha presionado una vez que se ha clicado un héroe e implementar una estrategia disponiendo de la clase

```
public class NPCMove : MonoBehaviour , IPointerDownHandler
```

, para esto se utiliza la característica fundamental del método

```
void Update() //se llama en cada frame
{
    DetectPressedKeys(); //detectar las teclas presionadas en cada frame
}
```

Con la implementación del método al que se llama en el Update() se verifica que tecla se ha presionado, en cada caso se llama al método particular para moverse acorde a la dirección correspondiente con cada tecla ej:

```
private void MoveW() //mover hacia arriba
```

con la implementación de cada uno de estos métodos solo se hace una verificación en la matriz instanciada en la escena y si es posible moverse, pues se mueve, una vez acaba el turno se debe presionar el botón de pasar turno, con el método

```
public void OnPassButtonPressed()
```

con esto se llama a otro método encargado de preparar el juego para el próximo jugador, ¿Qué método, en qué clase?, pues en la clase más importante de todo el juego

```
public class GameManager : MonoBehaviour
```

que se encarga de albergar en ella cada objeto de la escena, cada información y dirigir el flujo del juego en cada momento pues esta, para este preciso instante consta del método

```
public void PrepareGame() //preparar el juego para cada turno
```

Y cabe la pregunta de ¿Cómo se llegó hasta aquí?, pues el juego consta con la clase

```
public class ScenesController : MonoBehaviour
```

cada uno de los métodos de esta clase se encarga de cargar la escena correspondiente en cada momento del juego (PRESENTACIÓN , MENÚ , LABERINTO, VICTORIA) ej **FIG.5**

FIG.5

=> Usar una habilidad de la ficha (si no está en tiempo de enfriamiento) con el botón **APLY EFFECT**

¿Cómo funciona esto?, pues en cada momento se tiene la variable del GameManager

```
public GameObject clickedHero; //héroe  clicado
```

con esta variable se puede conocer el héroe correspondiente en cada momento del juego, una vez se presiona el botón de aplicar el efecto se llama al método correspondiente con el efecto de cada héroe y para manejar este tipo de cosas el juego consta con la clase

```
public class Effects : MonoBehaviour , IPointerDownHandler
```

la cuál maneja si es posible aplicar el efecto del héroe debido al enfriamiento, si se presionó el botón de aplicar el efecto, y de también activar los efectos de las [1] (TRAMPAS, POTENCIADORES, ITEMS y

etc), lo que se controla en **NPCMove** en cada movimiento una simple verificación y si hay [1] … pues se llama al método

```
public static void ColectObjects(int xpos , int ypos) //colectar objetos
```

que se encarga de aplicar el efecto de cada [1]

El tablero se genera de forma aleatoria al inicio de la partida, asegurando que no haya casillas inalcanzables. Se distribuyen los objetos (obstáculos, trampas, casillas especiales) de manera que no haya zonas bloqueadas y los jugadores puedan navegar por todo el tablero.

Lenguaje y Herramientas Utilizadas:

=> LENGUAJE: C#.

=> MOTOR GRÁFICO: UNITY_ENGINE.

=> SPRITES: www.kliparts.com , www.pngeggs.com .

=> SOUNDS: www.Pixabay.com .

Cómo Jugar

1. **Iniciar el Juego**: Ejecuta el archivo principal.

3. **Escoger Héroes**: Cada jugador selecciona sus fichas antes de que comience la partida.

3. **Jugar**: Los jugadores alternan turnos moviendo sus fichas y usando sus habilidades. El primero en lograr el objetivo de la partida será el ganador.

Instalación

1. Clona el repositorio en tu máquina local

   https://github.com/Leandro-Marquez/MAZE_RUNNER_LOGIC-GAME

2. Crea un proyecto de Unity 2D.

3. Importa el proyecto clonado al nuevo proyecto.

4. Compila el proyecto en un destino adecuado, esto generará un archivo te tipo .exe , con este se podrá jugar el juego finalmente.

Contribuciones

=> Las contribuciones son bienvenidas. Si encuentras algún error o tienes una mejora para el proyecto, no dudes en abrir un **Issue** o enviar un **Pull Request**.

=> Si tiene algún tema peculiar el cual le gustaría abordar directamente con el desarrollador puede contactar a la siguiente dirección de correo electrónico leandromarq0402@gmail.com .