**## Maze Run**

**Descripción:**

Maze Run es un juego por turnos con temática de escape de laberintos para varios jugadores. Los jugadores escogerán su ficha al inicio del juego y la guiarán a través de trampas y otros jugadores para llegar primero a la salida. Cada ficha tendrá una habilidad disponible que será de ayuda para escapar del laberinto. Con un laberinto distinto en cada juego, cada partida se sentirá diferente gracias a una generación de laberintos basada en el Algoritmo de Prim.

**Instrucciones:**

Una vez iniciado el juego se mostrara el Menú Principal con dos opciones: Iniciar Juego para pasar a la selección de jugadores y luego a la selección de fichas; y Salir para terminar el juego.

Una vez seleccionadas las fichas el juego iniciará. Los jugadores jugarán en el orden en que seleccionaron las fichas. Se mostrará el laberinto y el jugador en turno podrá moverse y si lo desea usar la habilidad especial de la ficha seleccionada al inicio. Los jugadores podrán caer en trampas y sufrirán efectos adversos que obstaculizaran su paso por el laberinto. El juego terminará cuando una ficha logre llegar a la salida

**Funcionamiento del juego:**

El juego utiliza la biblioteca Console.Spectre para ofrecer una interfaz más cómoda para el usuario.

Las clases principales del proyecto son Program, Game, Player, Token, Skill, MazeGenerator y Cell (y sus derivadas)

**Cell:**

La clase Cell se utiliza para representar una casilla del laberinto. Cell es una clase abstracta que es utilizada como base para la generación del laberinto y los tipos de casilla que van a existir en el laberinto. Tiene como propiedades X y Y para representar la posición en el laberinto que toma la casilla y un Emoji para representar la casilla cuando se imprima el laberinto. Ademas viene con un método Print que será sobreescrito por las clases hijas para imprimirse cada una.

**CellPath:**

CellPath es la clase utilizada para representar un camino en el laberinto, o sea es la celda por la que se puede caminar sin ningún efecto adverso. Es una clase que hereda de Cell y no tiene muchos cambios con ella mas alla del Emoji para imprimirla

**CellTrap:**

Esta clase representa una trampa en el laberinto, al igual que CellPath el jugador puede caminar sobre ella pero hacerlo le ocasionara un efecto adverso. Al igual que CellPath no tiene particularidades más allá del Emoji.

**CellWall:**

Es usada para representar una pared. El objetivo de esta celda es evitar que el jugador pueda salir del laberinto. No tiene particularidades más allá del Emoji

**CellObstacle:**

Esta celda es la que marca por donde es posible caminar en el laberinto. Originalmente esta clase era básicamente una pared pero como más adelante es posible modificar muy ligeramente el laberinto usamos esta clase para representar aquella ¨pared¨ que podemos modificar. Al igual que las demás difiere en su Emoji.

**CellExit:**

Esta celda marca simplemente la salida del laberinto. Una vez que una ficha llegue a esta casilla el juego terminará. Tiene Emoji característico.

(Todas las clases anteriores son hijas de la clase Cell)

**MazeGenerator:**

Esta clase es fundamental en el funcionamiento del juego. Utiliza como propiedad ´´auxiliar´´ una variable random, y diversos métodos que por sí mismos ayudan de una forma u otra a que el laberinto se genere correctamente. Usando una implementación iterativa del Algoritmo de Prim

**Skill:**

La clase Skill sirve para poder manejar de forma más focalizada las habilidades de las fichas. Tiene como propiedades un Name, CoolTime declarar el tiempo de enfriamiento de la habilidad y un Count para controlar el tiempo restante para poder usar la habilidad de nuevo. Posee un método para comprobar si es posible utilizarse (CanUseSkill)

**Token:**

Esta clase representa a la ficha. Es la que mas propiedades tiene debido a que es la que sufre de la mayoría de eventos posibles en el juego. Tiene un Name y una Speed para la cantidad de movimientos posibles en un turno; X y Y para representar la posición en el laberinto; una OriginalSpeed para manejar cambios en la velocidad de la ficha; un State para el estado en el que esta la ficha (por ahora None o Paralized) y un Count para manejar la cantidad de turnos restantes con el estado modificado; TrapTurns para los turnos que pueden durar el efecto de ciertas trampas; un Icon para imprimirlo en la consola; la Skill correspondiente al Token y dos arrays de direcciones (DirectionRow y DirectionCol) para ayudar con el movimiento de estas por el tablero. Posee al método MoveToken que modifica los valores X y Y en función del índice (dirección) pasado. Unos métodos GetCol y GetRow auxiliares y el método Update que será utilizado al cambiar de turno para actualizar el estado del Token o resetear propiedades que puedan haber sido modificadas por habilidades o trampas durante el juego.

**Player:**

La clase Player representa simplemente a los jugadores, van a tener un Token seleccionado al inicio del juego y un método Paralyze para ayudar en la aplicación de una habilidad especifica.

**Game:**

Esta clase es la encargada de controlar todos los procesos del juego. Esta clase va a tener a los Players seleccionados y al Maze como propiedades fundamentales. Tambien tiene una propiedad para manejar el turno (Turn) en que se realice el juego, una tupla de enteros para la salida del Maze que es importante porque es la que va a marcar el fin del juego, un bool GameStop para ayudar en la declaración del fin del juego y un random como apoyo a ciertas habilidades y otros eventos del juego.