INFORME PRIMER PROYECTO DE PROGRAMACIÓN

SUBDESARROLLO STUDIOS |

SHITPOST RUNNERS

Alejandro Garcia Ruiz C112

2025

El proyecto es un juego por turnos de recorrer un laberinto lleno de trampas. Utiliza el motor de Unity y tiene temática shitpost con distintos memes en las fichas, casillas y banda sonora.

Cuenta con 3 escenas en total:

Menú:

Escena con tres botones, el primero para empezar una partida, el segundo para ver el tutorial y el tercero para salir del juego.

Tutorial:

Escena donde se despliegan distintas instrucciones y explicaciones sobre las mecánicas del juego

Juego:

Escena donde se encuentra el juego en sí.

**Como Ejecutar**

El ejecutable se encuentra en la carpeta SR, no necesita de nada para ser instalado ni ejecutado, en esencia, si la computadora tiene RAM suficiente para abrir el Word, puede abrir el proyecto. En la carpeta Assets se encuentra todo lo utilizado para el proyecto, desde la banda sonora hasta el código (todo el código se encuentra dentro de la carpeta scripts).

**Instrucciones del juego:**

El juego comienza personalizando los jugadores, con su nombre y su avatar (máximo de cuatro jugadores). Una vez seleccionados la computadora repartirá 5 fichas al azar (cada una con una habilidad distinta) a cada jugador que serán con las que se moverá a lo largo de todo el laberinto. El objetivo es recorrer 8 letras repartidas por el laberinto. Una vez que todas hayan sido recogidas la partida terminará y el jugador con más letras será el ganador. En casi todas las casillas del laberinto hay alguna trampa o bonificación, el problema es que todas las casillas excepto los obstáculos están cubiertos y no se conoce que casilla contiene que hasta que se cae en ella por primera vez. En cada turno el jugador puede utilizar todas las habilidades que desee, así como mover una sola ficha (o todas las que quiera si el oponente no se da cuenta), el turno solo terminará una vez se haga click en el botón de pasar turno.

Fichas:

Hay 8 tipos de ficha en total y aunque cada una tiene una habilidad distinta todas comparten dos propiedades:

Velocidad: la cantidad de casillas que pueden recorrer en un turno.

Enfriamiento: la cantidad de turnos que se debe esperar para volver a utilizar su habilidad.

(Para ver más detalladamente la información de cada ficha recurra al tutorial dentro del juego)

Casillas:

Hay hasta 9 tipos de casillas distintas en todo el laberinto con distintos efectos cada una. Su mecánica principal es que no se ve que contienen en su interior hasta que una ficha cae en ella por primera vez, cambiando su foto y revelando el tipo

(Para ver más detalladamente la información de cada casilla recurra al tutorial dentro del juego)

**Descripción del juego desde la programación:**

El proyecto está dividido entre lo visual y la lógica que controla el juego. Comenzaré describiendo archivo por archivo

InterfacesYEnums:

Enum Casilla 🡪 Enum para clasificar el tipo de casilla.

Enum TipoFicha 🡪 Enum para clasificar el tipo de ficha.

Interface ICasilla 🡪 Es la interfaz que agrupa a los distintos tipos de casilla que hay en todo el laberinto. Contiene:

* Un int para la fila y otro para la columna.
* Un string donde se guarda la el mensaje que deja la casilla en pantalla cuando se cae en ella.
* Una lista de Fichas donde se guardan todas las fichas que están en esa casilla.
* Un método void donde se ejecuta la acción de la casilla sobre la ficha

public abstract class Ficha 🡪 Clase abstracta de la que heredarán todas las fichas. Contiene:

* Una propiedad string con la Descripción de la habilidad de la ficha.
* Una propiedad de tupla de int con la posición de la ficha en el laberinto (fila, columna).
* Un int con el enfriamiento base de la ficha.
* Un int con el enfriamiento actual de la ficha.
* Un int con la cantidad de turnos que no se puede jugar esta ficha.
* Una variable de tipo Jugador con el propietario de la ficha.
* Una propiedad abstracta de TipoFicha.
* Un método abstracto void con la habilidad de la ficha.
* Un método void que devuelve la menor cantidad de pasos posibles para realizar una jugada. Este método utiliza el método de backtracking.
* Un método bool que devuelve si una jugada es válida o no. En este se llama al método para saber la cantidad de pasos, luego se compara si es menor que la velocidad disponible y si es posible activa la acción de la casilla en las coordenadas dadas.

Jugador:

class Jugador 🡪 La clase donde se guarda el comportamiento de un jugador. Contiene:

* Un string con el nombre
* Un int con la cantidad de turnos que no puede jugar
* Una lista de las fichas que posee
* Una lista de las letras que posee
* Un método bool Jugar donde se controla la jugada. En este si la cantidad de turnos sin jugar es distinta de 0 devuelve falso, sino llama al método de jugar de la ficha y devuelve su valor

Casilla:

Todas las clases en este archivo heredan de ICasilla y en lo único que se diferencian es en los valores del tipo y la descripción; por tanto, describiré solo el método void Accion.

class LetraClave 🡪 Es la casilla que te da una de las 8 letras necesarias para ganar

* Accion: Si nadie ha caído en ella por primera vez le suma uno al contador general de letras obtenidas en el juego y le añade una de las letras al propietario de la última ficha en la lista de fichas en la casilla

class Vacia 🡪 Casilla vacía

* Accion: No hace nada

class Obstáculo 🡪 Casilla por la que no se puede pasar

* Accion: No hace nada

class ChillGuy 🡪 Trampa que disminuye la velocidad

* Accion: Resta 1 a la propiedad de velocidad de la ficha

class Abuelito 🡪 Trampa que impide el movimiento

* Accion: Suma 2 a la propiedad de turnos sin jugar de la ficha

class AmongUs 🡪 Trampa que regresa la ficha a la casilla anterior

* Accion: Lo único que hace es modificar el mensaje de la casilla en pantalla, pues mas que regresar la ficha, lo que hace es que directamente no avanza hasta ella.

class Morfeo 🡪 Trampa que tiene dos efectos, aumenta el enfriamiento o aumenta la velocidad

* Accion: Crea un numero random y dependiendo de cual sea aumenta la propiedad del enfriamiento actual o la velocidad.

class Honguito 🡪 Bonificación que restaura la habilidad de la ficha.

* Accion: Vuelve 0 la propiedad de enfriamiento actual de la ficha.

class Ojo 🡪 Bonificación que muestra las casillas cercanas.

* Accion: Tiene 2 int[] con las direcciones en las filas y las columnas. Luego itera por ellos para desbloquear las fotos de las casillas en esas coordenadas siempre que estén dentro del laberinto.

Ficha:

En este archivo todas las clases heredan de la clase abstracta Ficha por lo que solo varían en su constructor por los valores y en su habilidad. Por tanto, sólo describiré el método void Habilidad.

class McQueen

* Habilidad: Si el enfriamiento es 0 aumenta su propia velocidad en 1, luego aumenta el enfriamiento y luego cambia el texto de la habilidad

class CJ

* Habilidad: Si el enfriamiento es 0 va iterando por todas las fichas que se encuentren en su misma casilla y si su propietario tiene una letra la quita de su lista y la añade a la de su propietario, hace este proceso una sola vez, luego aumenta su enfriamiento y modifica el texto de la habilidad.

class UNE

* Habilidad: Si el enfriamiento es 0 genera un numero random entre 0 y la cantidad de jugadores para evaluarlo en la lista de jugadores. Con un while repite este proceso hasta que el jugador seleccionado no sea el propietario de esta ficha, luego le suma 2 a la cantidad de turnos sin jugar de este jugador. Por último, aumenta su enfriamiento y modifica el texto de la habilidad.

class Knuckles

* Habilidad: Si el enfriamiento es 0 accede a todas las fichas de su propietario y le resta 1 o 2 al enfriamiento actual dependiendo de lo que tengan.

class RickRoll

* Habilidad: Si el enfriamiento es 0 genera 4 números al azar, uno para el jugador, otro para la ficha, y los otros 2 las coordenadas. Con un while se encarga que el jugador no sea el propietario. Con otro while controla que la casilla en esas coordenadas no sea ni un obstáculo ni una letra. Luego mueve esa ficha de ese jugador a la casilla en esa coordenada, activa la acción de la casilla y la vuelve visible. Aumenta su enfriamiento y modifica el texto de la habilidad.

class Starman

* Habilidad: Modifica el texto de la habilidad, puesto que no tiene habilidad en sí sino que el resto de habilidades paran si es este tipo de ficha el objetivo.

class Dogue

* Habilidad: Si el enfriamiento es 0 itera por cada una de las fichas que se encuentren en su misma fila y les resta 2 o 1 de velocidad a cada una dependiendo de cuanto tengan para nunca dejarla en 0. Aumenta su enfriamiento y modifica el texto de su habilidad.

class ElChoco

* Habilidad: Modifica el texto de su habilidad puesto que no tiene habilidad en sí sino que las trampas paran si esta ficha es el objetivo.

Laberinto UN:

Este archivo es el que controla en comportamiento del laberinto. Contiene:

* Las 9 fotos de los distintos tipos de casillas
* Una matriz de 15x15 de ICasilla que sería el laberinto de las casillas en CSharp.
* Una matriz de 15x15 de GameObject que sería el laberinto compuesto por los objetos en la escena de unity.
* Un GameObject que es el prefab mediante el cual se instanciará en la escena cada casilla del laberinto.
* Un array de tuplas de int que simula las direcciones (utiliza 2, -2, 0 en vez de 1, -1, 0, para así asegurarme de que los bordes del laberinto siempre sean paredes)
* Un método void ShuffleDirections que modifica el orden del array de direcciones.
* Un método ICasilla GenerarCasilla que genera una casilla al azar. Usando un random genera un número y dependiendo de cual sea devuelve que casilla.
* Un método void AbrirCaminos que utilizando backtracking genera caminos de casillas.
* Un método void ReorganiceLetters que por cada una de las letras en MONDONGO escoge una casilla al azar del laberinto. Con un while se asegura que no sea ni un obstáculo ni una letra. Luego cambia esa casilla por una de tipo letra y le asigna una.
* Un método void GenerateLaberinth el cual instancia el laberinto con todas sus casillas en obstáculos, llama al método de abrir caminos y luego al de reorganizar letras
* Un método void InstanciarCasillas que recorre todas las casillas de la matriz de ICasilla, dependiendo del tipo instancia el prefab de esta en Unity y le asigna la imagen correspondiente.
* El método void Start que llama a los métodos GenerateLaberinth e InstanciarCasillas

GameController:

Esta clase es la que controla la jugabilidad y por la cual se regulan los procesos. Contiene:

* Un int que controla el turno
* Un int que lleva la cantidad de letras encontradas en la partida
* Un GameObject en el que se encuentran los avatares de los jugadores
* Una lista con los jugadores actuales.
* Un objeto de tipo LaberintoUN donde se contiene todo lo relacionado al laberinto
* Un Objeto de tipo VisualController que regula la mayoría del apartado visual
* Un objeto de tipo AudioController que regula el audio
* Un método bool para controlar la jugada que si no es válida devuelve false y si lo es, la realiza y llama al método de realizar una jugada en el VisualController.
* Un método void AvamzarTurno que regula dicha acción. Primero revisa si ya todas las letras han sido recogidas, si ya han recogido todas elige al ganador y llama al visual y la música para enseñar el ganador en pantalla. Luego avanza el contador de turno y lo vuelve 0 si es igual a la cantidad de jugadores. Si el turno es igual a 0 le resta 1 a los turnos sin jugar de cada jugador y al enfriamiento y turnos sin jugar de cada ficha (si es necesario). Por último, llama a la función de avanzar el tueno en el VisualController.
* Un método void GenerarFichas que dado un jugador elige con un random 5 fichas al azar para él, luego una casilla al azar también y con un while se asegura de que no sea ni un obstáculo ni una letra y esa será la casilla en la que se encuentre dicha ficha al inicio de la partida
* Un método void GenerarJugadores que dada la cantidad de hijos que tenga el GameObject de los avatares, genera los jugadores y los añade a la lista. Luego por cada hijo accede al componente JugadorUN y genera las fichas en el Unity.
* El método Start que busca el Laberinto en la escena, vuelve el turno 0 y busca el controlador de sonido.

JugadorUN:

Este archivo es el que adapta la clase jugador para poder usarla en el unity más cómodamente. Contiene:

* El objeto jugador instanciado en CSharp.
* Una propiedad con el nombre que devuelve el nombre del jugador de CSharp.
* Una lista de GameObject que serían las fichas ya instanciadas en la escena de Unity.
* Un bool que regula si el jugador ha sido seleccionado.
* Un GameObject que es el prefab mediante el cual se instancian las fichas en la escena
* Las 8 fotos de las fichas.
* Un método void para volver true el bool de seleccionar
* Un método void para comenzar la animación de daño del personaje
* Un método void para instanciar todas las fichas en la lista de fichas del jugador de CSharp en la escena y luego añadirlas a la lista de fichas de unity.

FichaUN:

Este archivo adapta la los objetos Ficha para poder utilizarlos en Unity. Contiene:

* Un objeto de tipo fica que contiene la ficha en CSharp.
* El Enum que indica el tipo de ficha.
* El texto en unity donde irá la descripción de la habilidad de la ficha.
* Un método void que activa la habilidad de la ficha. Si el unity detecta que se hizo click derecho sobre la ficha y el enfriamiento es 0 entonces llama a la habilidad de la ficha de CSharp.

CasillaUN:

Este archivo adapta el objeto de tipo casilla para poder utilizarlo en Unity. Contiene:

* El Objeto de tipo ICasilla.
* El Enum con el tipo de casilla.
* Un int con la fila
* Un int con la columna.
* Una lista de GameObject que serían las fichas que están en la casilla.
* Un método que vuelve visible la casilla modificando el componente imagen del objeto en Unity.
* Un método que hace que la casilla tome un tono verde.
* Un método que llama al que la hace visible, activa la acción de la casilla de CSharp y si es de tipo Letra llama al que la hace verde para evitar que los jugadores se confundan y sepan cuales letras han sido tomadas.

VisualController:

Este archivo es el que controla muchos de los procesos visuales de la jugabilidad del juego. Contiene:

* El texto donde va el nombre del jugador en turno.
* El texto donde se muestra la descripción de la habilidad.
* El texto donde se muestra la descripción de la casilla.
* El GameObject donde se enseñan las letras del jugador en turno.
* Una referencia al GameController.
* El GameObject de la pantalla de victoria.
* El prefab de las letras
* Un método void que se encarga únicamente de mover una ficha a una casilla.
* Un método void que actualiza el mensaje de la casilla y si esta no es de tipo AmongUs activa la animación de teletransporte y el sonido, luego mueve la ficha.
* Un método void que actualiza el nombre del jugador en turno.
* Un método que actualiza la pantalla del ganador y corre el audio.
* Un método void que se encarga de volver interactuables y visibles las fichas del jugador y caso contrario con el resto de fichas.
* Un método void que se encarga de instanciar una letra según el prefab.
* Un método que utiliza el método anterior para instanciar todas las letras del jugador en turno
* Un método para avanzar el turno en el que se actualiza el nombre, las letras, se transparentan las fichas enemigas, y se resetean los textos de la descripción de la habilidad y de la casilla.
* El método Start que desactiva la pantalla del ganador.

LogosLoader:

Este archivo se encarga únicamente de correr la animación del inicio del juego.

ShowInfo:

Este archivo instancia una pequeña ventana con la descripción de la ficha siempre que el mouse esté encima de esta.

ChangeButtons:

Este archivo se encarga de cambiar el color de los botones si el mouse está encima de estos.

AudioController:

Este archivo gestiona todo lo relacionado al audio en el juego. Contiene:

* Un objeto de tipo AudioClip (Componente de unity).
* Una lista de canciones.
* Un objeto de tipo AudioSource (Componente de unity).
* Un int que guarda el índice de la actual canción
* Un método que para la canción actual.
* Un método void que pasa la canción
* Un método void que para la canción actual y corre la de victoria.
* Un método void que reorganiza el orden de las canciones.
* El método Start que llama al de reorganizar y reproduce la música.
* El método Update que continuamente confirma si la canción ya se acabó y reproduce la siguiente, actualiza el índice y si se acaban las canciones las reorganiza y las vuelve a reproducir.

MakeMove:

Archivo que controla las jugadas y se las pasa al controlador. Contiene:

* Dos InputField (Componentes del Unity) que tienen las coordenadas de la jugada en fila, columna.
* GameObject que es el GameController en la escena de Unity.
* Un método que es el que revisa si hay una ficha seleccionada y luego le pasa esa ficha junto con las coordenadas al GameController. Si la jugada no es válida activa la animación de temblar indicando que el jugador debe cambiar su jugada.

ScenesManager:

Archivo que gestiona el cambio entre escenas. Contiene:

* Un método que cambia a la primera escena.
* Un método que activa la animación de cambio y llama a la primera escena.
* Un método que cambia a la segunda.
* Un método que activa la animación de cambio y llama a la segunda escena.
* Un método que cambia a la tercera.
* Un método que activa la animación de cambio y llama a la tercera escena.
* Un método que cierra el juego.

SelectPlayers:

Este archivo es el que gestiona la selección de jugadores. Contiene:

* Un InputField donde se escribe el nombre del jugador.
* Un GameObject donde se encuentran los avatares sin seleccionar.
* Un GameObject donde se encuentran los avatares seleccionados.
* 4 GameObject que serían el primero, segundo, tercero y cuarto avatar.
* Un método void para añadir un jugador y transportar el avatar seleccionado a donde corresponde.
* Un método void para eliminar a un jugador ya creado y mover su avatar a donde corresponde.

ToGame:

Este archivo es el que cambia del creador de personajes al juego. Contiene:

* Un GameObject que es el creador de personajes en sí.
* Un GameObject que son los avatares de los jugadores.
* Un GameObject que es donde deberían estar los jugadores en el juego.
* Un método void que mueve los avatares hasta donde deberían estar, desactiva el creador de personajes, llama al método de generar jugadores del GameController, llama a los métodos de cambiar el nombre y transparentar las fichas en el VisualController.

Tutos:

Gestiona las páginas que muestran los tutoriales. Contiene:

* Un GameObject que enseña las instrucciones del juego.
* Un GameObject que enseña lo relacionado a las casillas.
* Un GameObject que enseña lo relacionado a las fichas.