SAGRADA

Un juego de laberintos

JAVIER FONTES BASABE

1er Proyecto de Programación C-122

MATCOM 2024-2025

ÍNDICE

INTRODUCCIÓN

Sagrada es la solución ofrecida al problema de crear un juego multijugador con temática de laberinto donde el tablero se genera dinámicamente. Con el empleo de la consola como interfaz visual y añadiendo una experiencia sonora poco usual en este tipo de plataformas el proyecto presenta una respuesta creativa a la misión encomendada, ofreciendo al usuario una oportunidad de disfrutar de un juego diferente.

Inspirado en la arquitectura modernista de Antonio Gaudí y los demonios que lo acompañaron durante toda su vida creativa, el jugador deberá, a través de los ojos de uno de los demonios, obtener la pieza perdida por el arquitecto antes que se desmorone su obra cumbre, La Sagrada Familia.

DESCRIPCIÓN DEL PROYECTO

Sagrada es un juego de consola multijugador diseñado para desafiar a los jugadores a encontrar la mejor manera de obtener una pieza y retornarla a su base. El juego combina elementos de estrategia y habilidad, requiriendo el pensamiento creativo de los jugadores al enfrentarse a sus contrincantes y para evitar los obstáculos presentes en el laberinto así como las acciones aleatorias incluidas en el juego.

Género:

Estrategia o acción, con elementos multijugador y competitividad directa (PvP).

Tecnologías utilizadas:

Para el desarrollo del juego fueron utilizadas diferentes tecnologías para aportar diferentes funcionalidades. Estas en conjunto ofrecen una experiencia de usuario mucho más agradable. Entre las principales tecnologías encontramos:

- **C#**: Lenguaje de programación utilizado para elaborar toda la lógica del juego.
- **Spectre**.**Console:** Librería utilizada para realizar los gráficos del juego.
- **NAudio**: Librería utilizada para agregar música a la consola.
- **Git**: Utilizado para el control de versiones del proyecto.

Mecánicas de juego:

- **Objetivo principal:** Captura la bandera.
- **Interacción con otros jugadores:** Eliminar oponentes. (Al ser destruido un jugador aparece automáticamente en su posición de origen).
- **Trampas y obstáculos:** El juego cuenta con trampas con diversos efectos.
- **Visibilidad limitada de trampas:** Las trampas son visibles durante un período restringido de tiempo y con un costo de poder.
- **Turnos:** El juego se juega por turnos permitiendo planificar movimientos y estrategias.
- **Poderes:** Cada jugador posee un poder único que influye en la estrategia de juego, similar a los héroes en juegos de estrategia.

Características principales:

- **Objetivo**: el objetivo principal es capturar la pieza que se encuentra en el centro del laberinto y llevarlo de regreso a la base de cada jugador evitando ser destruido por trampas u otros jugadores.
- **Interfaz de usuario:** La aplicación cuenta con una interfaz de usuario sencilla permitiendo al usuario interactuar de manera agradable con el

juego, visualizar el laberinto en la pantalla y realizando los movimientos de manera intuitiva.

- **Laberinto dinámico:** El laberinto se genera de manera aleatoria cada vez que se inicia una nueva partida, lo que garantiza una experiencia única. El laberinto se compone por cinco sublaberintos interconectados como se muestra en la figura 1, con una dimensión de 13x13.

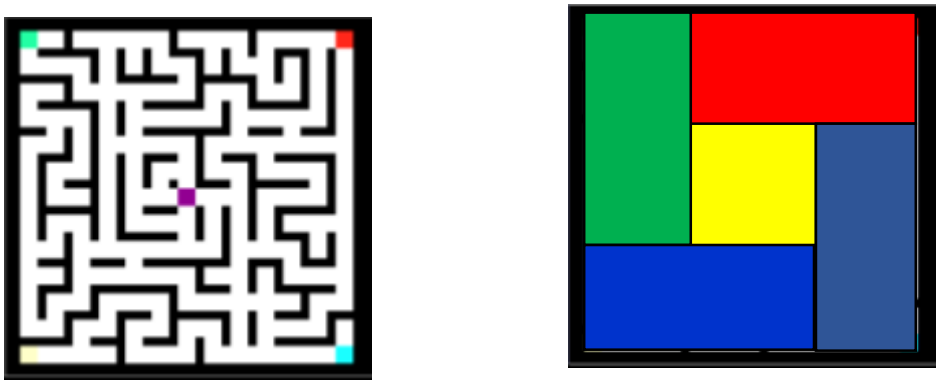
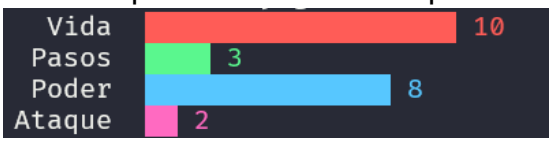


Figura 1

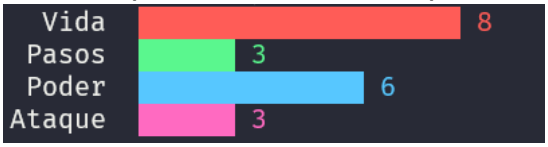
Diseño de personajes:

El juego cuenta con un total de siete personajes distintos para ser escogidos por el usuario al iniciar cada partida. Cada uno cuenta con un poder especial y características únicas, haciendo posible que las estrategias varíen en cada partida según los personajes que encontramos en el tablero. Así mismo todos los personajes tienen la posibilidad de revelar las trampas del tablero y atacar a sus oponentes. Los personajes son:

- **<u>Visión de Luz:</u>**
  Poder especial: Atravesar las paredes.

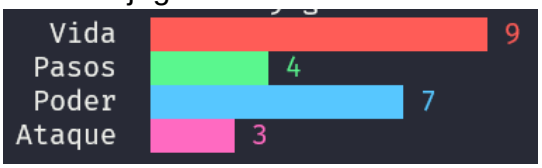


- **<u>Viento creativo:</u>**
  Poder especial: Aumenta sus pasos en 4.

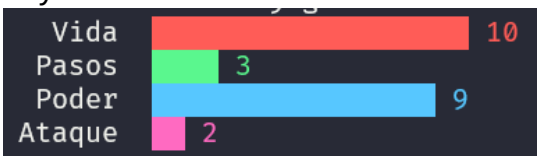


- **<u>Alma vital:</u>**
  Poder especial: Aumenta su vida en 3.

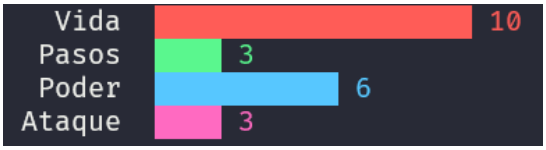
- **Mimetista de ideas:**
  Poder especial: Es capaz de intercambiarse con otro jugador.



- **Ruptura natural:**
  Poder especial: Destruye una trampa adyacente a él.



- **Espejo del tiempo:**
  Poder especial: Al activar el poder tiene un turno extra.



- **Mente Camaleón:**
  Poder especial: Imita el poder de uno de los jugadores en juego.

Diseño de trampas:

El juego cuenta con 3 trampas diferentes. A diferencia de muchas trampas que afectan al jugador que las activa, en este juego una trampa puede afectar a más de un jugador e incluso resultar beneficiosa en algunos casos. Además la aleatoriedad de su ubicación y de sus efectos permite tener una partida más dinámica. Las trampas no se destruyen luego de caer en ellas, por esto debes tener mucho cuidado de recordar por donde pasas.

Trampas:

- **Trampa de daño:** Reduce la vida del jugador que cae sobre ella. El valor de este efecto es aleatorio.
- **Cambio de laberinto:** Al caer en esta trampa cambia la estructura del laberinto.

- **Cambio de posición:** Al caer en esta trampa el laberinto enviará al jugador a una posición aleatoria cerca de la esquina opuesta a su posición actual en el tablero.

*Las trampas nunca se generan en el borde del tablero o las casillas centrales.

INSTALACIÓN Y EJECUCIÓN DEL PROYECTO

Para instalar y ejecutar el juego **Sagrada** en Windows, sigue estos pasos detallados:

1. **Clonar el repositorio**: Utiliza Git para clonar el repositorio Sagrada desde GitHub. Puedes hacer esto con el siguiente comando en tu terminal:

```
git clone https://github.com/FontesHabana/Sagrada-AMazeGame
```

2. **Abrir la carpeta "Program":** Navega a la carpeta "Program" dentro del directorio clonado. Se recomienda usar PowerShell para esto.

3. **Ampliar la ventana de la consola**: Ajusta el tamaño de la ventana de PowerShell para que ocupe toda la pantalla, lo que evitará distorsiones en los gráficos del juego.

4**. Ejecutar el proyecto:** Compila y ejecuta el proyecto utilizando el siguiente comando

```
dotnet run
```

Requisitos del Sistema

- **Sistema operativo:** Windows
- **Dotnet:** Necesitas tener instalado .NET 9.0 para poder compilar y ejecutar el juego.

Asegúrate de tener todos los requisitos instalados antes de proceder con la ejecución del juego.

* Es importante ejecutar el proyecto con la consola maximizada para evitar distorsiones de imagen durante la ejecución del juego.

** Es recomendable estar conectado a internet durante la primera compilación del proyecto. Así si alguna librería no se ha instalado se instalará automáticamente.

INSTRUCCIONES DE JUEGO

Controles del Juego:

- **Movimiento:** W: Moverse hacia arriba
  A: Moverse a la izquierda
  S: Moverse hacia abajo
  D: Moverse a la derecha
- **Menús:**
  - **Navegación:** Flecha Arriba: opción superior ↑
    Flecha Abajo: opción inferior ↓
  - **Seleccionar una opción**: Enter.

Al iniciar un turno podrás moverte por el tablero utilizando las teclas de movimiento o seleccionar activar uno de los poderes disponibles en el menú de juego ubicado a la derecha de la pantalla. Cuando termines tu turno selecciona la opción “Siguiente turno”.

El color de tu ficha en el tablero es el mismo de los bordes del recuadro con tu imagen.

Todos los textos se cierran presionando cualquier tecla.

ESTRUCTURA DEL PROYECTO

El proyecto está dividido principalmente en dos secciones:

1. Lógica del Juego (LogicGame).
2. Interfaz de Usuario (UserInterface).

Lógica del Juego:
Maneja todas las acciones y algoritmos necesarios para el funcionamiento de la aplicación sin importar como será representada al usuario.

La clase **Program** contiene a la aplicación. En esta se accede al menú de inicio con las funciones básicas como iniciar un nuevo juego, cambiar el lenguaje de la aplicación, ver las instrucciones o la historia.

La clase principal del juego es la clase **GameMaster**. Esta clase maneja la lógica central del juego y es la encargada de llamar a la parte visual para ser enlazada. Incluye entre sus funciones:

- La gestión de jugadores
- Configuración inicial del juego
- Flujo del juego
- Mecánicas de turno

Entre los métodos que podemos destacar en esta clase encontramos:

- **Game:** Método que ejecuta un nueva partida.
- **InitGame:** Método encargado de gestionar el inicio de cada partida. En este método se genera el tablero y la lista de los personajes que estarán presentes.
- **Turn:** Método encargado de gestionar el estado de cada turno durante el juego.
- **VictoryCondition:** Método encargado de comprobar si algún jugador cumple la condición de victoria.

Otros métodos auxiliares que podemos encontrar en esta clase son **NextTurn**, **Normalize** y **ValidateName**.

La clase **Menu** es la encargada de crear y gestionar los menus del juego. Un menú cuenta con una lista de **tuplas** de la forma (bool, string), que representan las posibles acciones a realizar por el menú y si está seleccionada, y un delegado **ActionMenu** que guarda el método que se ejecuta al seleccionar una opción del menú.

El método principal de esta clase es **ChangeOption**. Este método permite navegar utilizando el teclado por las diferentes opciones del menú.

Además cuenta con seis métodos encargados de generar las listas de opciones a utilizar y otros seis métodos para las acciones correspondientes a cada lista.

Los métodos de acción tienen todos la misma estructura. Esta consiste en comprobar si se presionó la tecla Enter que activa la opción. Luego contiene un switch con las opciones correspondientes. El método ejecuta la opción que en su tupla contenga el valor de verdad true.

Cada instancia de la clase **Character** que hereda de **Tiles** representa un personaje diferente. Cada personaje posee diferentes propiedades como: posición, posición inicial, color de la ficha, vida, ataque, poder, velocidad, aumento de poder, consumo de poder, nombre, imagen asociada, un enum que representa su poder especial, entre otras.

Esta clase posee los siguientes métodos:

- **Respawn:** método encargado de reubicar a un jugador si este muere.
- **AttackTo:** método encargado de disminuir la vida de un jugador aledaño consumiendo poder.
- **ShowTrap:** método encargado de mostrar las trampas del tablero consumiendo poder.
- **HaveFlag:** método encargado de comprobar si el jugador tiene la bandera y cambiar la posición de esta según quien la tenga.

La clase **Power** contiene los métodos referentes a los poderes de cada jugador. Para activar un poder el programa recurre a un switch que revisa el enum correspondiente a cada jugador y ejecuta el poder correspondiente. Los diferentes métodos relacionados con los poderes son:

- **JumpWall:** permite atravesar una pared.
- **IncreaseSpeed:** permite aumentar la cantidad de pasos.
- **IncreaseLife:** permite recuperar vida.
- **SwitchPlayer:** permite cambiar su posición con otro jugador.
- **DestroyTrap:** permite destruir una trampa.
- **NewTurn:** permite jugar un turno extra.
- **CopyPower:** permite copiar el poder de otro jugador.

El método **Maze** es el encargado de la generación y gestión de los laberintos. Cada laberinto es una matriz compuesta por **Cells** o **Trap.** Este método lo podemos dividir en dos secciones principales el algoritmo de generación de laberintos y la estructura del laberinto del juego.

Para el algoritmo de generación se utilizaron los métodos siguientes:

- **GenerateMaze:** método encargado de ejecutar el algoritmo para elaborar un laberinto de nxm.
- **GetUnvisitedNeighbors:** método encargado de añadir a una lista las celdas vecinas sin visitar,
- **RemoveWall:** método encargado de remover de manera aleatoria una pared generando un nuevo camino.

El algoritmo utilizado para elaborar el laberinto y asegurar que es completamente visitable se basa en recorrer todas las celdas generando los caminos en este recorrido. Al inicio todas las celdas se encuentran sin visitar y con todas sus paredes. Se selecciona una celda de inicio y se añade a una pila. Luego se selecciona una celda adyacente al azar y se rompe la pared. Se coloca esta celda en la cima de la pila y será desde donde repitamos el proceso. Si no es posible moverse a ninguna celda adyacente se elimina la celda de la pila y se repite en la celda superior. De esta forma cuando en la pila no quede ninguna celda podremos asegurar que se ha recorrido todo el tablero y todas las celdas están conectadas.

En la sección correspondiente al algoritmo del juego tenemos los métodos **MainMaze** que se encarga de construir cinco laberintos diferentes con el algoritmo anterior y juntarlos en un solo tablero y el método **StaticCell** que se encarga de fijar ciertas uniones entre estos laberintos con el fin de poder desplazarse entre ellos.

Cabe destacar que la clase **Trap**, es una clase que hereda de **Cell,** permitiendo que se encuentren las instancias de estas a la vez en el tablero. Dentro de la clase **Trap** encontramos un método para la acción de cada trampa y un método **ApplyEffect** que ejecuta esta acción si un jugador cae en ella.

Interfaz de Usuario

Esta sección se encarga de la presentación visual del juego y la experiencia del usuario. Para esto se utilizaron diferentes estructuras de la librería Spectre.Console como son los layouts, tablas, canvas y canvas image para poder representar el laberinto y los personajes en la consola. La sección presenta dos clases principales **GameDisplay** y **MazeCanvas**.

GameDisplay es la clase encargada de gestionar, crear y actualizar la interfaz gráfica del juego. En esta clase destacan métodos como:

- **GameScreen:** crea y contiene la estructura de la pantalla del juego.
- **PlayerStatus:** gestiona la forma de presentar los datos de cada jugador en la pantalla.
- **VerticalMenu, HorizontalMenu, VerticalMenuInit, PrintSelectionMenu:** son los métodos encargados de representar en pantalla los diferentes menús.
- **Start:** muestra el inicio del juego.
- **MainPage:** presenta la pantalla de inicio.
- **History:** muestra la historia del juego.
- **Victory:** muestra una pantalla de victoria.

En esta sección también debemos destacar la clase **Text**. Esta cuenta con un diccionario que hace referencia a los distintos lenguajes en los que es posible utilizar la aplicación (español, inglés, francés).

Así mismo en la clase **Audio** encontramos el método **PlayAudio** que es el encargado de la reproducción de los efectos sonoros dentro del juego.

La separación en dos secciones viene dada para mejorar el resultado de la aplicación y la posibilidad de escalarla a otras plataformas de manera más sencilla. Con esta estructura es posible realizar cambios en la lógica del juego sin afectar la visualización. Permite agregar nuevos componentes, trampas y personajes de manera modular. También admite el soporte multilingüe fácil de escalar a nuevos idiomas.