BARBIE'S SECRET GARDEN

Primer Proyecto de programación

Sabrina Socarras Arnaiz | Primer Año | Ciencias de la Computacion
Universidad de La Habana
Facultad de Matemática y Computación

Introducción

Este informe tiene como objetivo proporcionar una visión general del proyecto Barbie's Secret Garden, un juego de mesa digital implementado en C#. El informe se divide en tres secciones principales: instalación del programa, cómo jugar, y un desglose del código importante para entender cómo funciona el juego.

1. Instalacion del programa

Para poder ejecutar el proyecto Barbie Secret Garden, es necesario seguir los siguientes pasos:

Requisitos Previos:

.NET SDK: Asegúrate de tener instalado el SDK de .NET (versión 6.0 o superior) en tu sistema.

Sistema Operativo: Solo se ejecutara con Windows.

Editor de Código: Se recomienda utilizar Visual Studio o Visual Studio Code para una mejor experiencia de desarrollo y depuración.

Spectre.Console: El proyecto utiliza la librería Spectre.Console para la interfaz de usuario en la consola. Esta librería se instala automáticamente al restaurar los paquetes NuGet del proyecto.

Pasos para Instalar y Ejecutar el Proyecto:

1. **Clonar el Repositorio**:
    - Clona el repositorio del proyecto desde GitHub o descarga los archivos directamente.
    - Abre una terminal en la carpeta del proyecto.
2. **Restaurar Dependencias**:
    - Ejecuta el siguiente comando para restaurar las dependencias del proyecto:

    – dotnet restore

3. **Compilar el Proyecto**:
    - Compila el proyecto con el siguiente comando:

    – dotnet build

4. **Ejecutar el Juego**:
    - Finalmente, ejecuta el juego con:

- dotnet run

Recomendable ejecutar en Windows PowerShell, y ajustar el tamaño de la ventana para que ocupe todo el espacio de la pantalla.

2. ¿Cómo jugar?

Inicio del Juego:

1. **Número de Jugadores**:
    - Al iniciar el juego, se solicita el número de jugadores que participarán.
2. **Selección de Piezas**:
    - Cada jugador selecciona una pieza (ficha) para representarse en el tablero. Las piezas tienen habilidades especiales que pueden activarse durante el juego.

Mecánicas del Juego:

- **Movimiento**:
    - Los jugadores se mueven por el tablero usando las flechas del teclado (arriba, abajo, izquierda, derecha).
    - Cada pieza tiene una velocidad que determina cuántos movimientos puede realizar en su turno.
- **Trampas**:
    - **Reducción de Velocidad**: Reduce a la mitad la velocidad del jugador.
    - **Volver al Inicio**: El jugador es enviado de vuelta a la casilla de inicio.
    - **Quedarse en el Lugar**: El jugador no puede moverse durante un turno.
- **Habilidades Especiales**:
    - Cada pieza tiene una habilidad especial que puede activarse una vez por turno. Estas habilidades pueden dar ventajas estratégicas. Pero que tienen un tiempo de enfriamiento de un turno.
- **Final del Juego**:
    - El juego termina cuando un jugador llega a la celda de salida (**Final**). Se muestra un menú de victoria con el nombre del jugador ganador.

3. Desglose del código mas importante

Las clases más importantes de mi código desde mi punto de vista son:

- **Clase Application:** Esta clase es el núcleo del juego y maneja la lógica principal. Contiene diferentes métodos como el **método Start** que inicializa el tablero, las piezas y los jugadores. Crea un tablero de 31x31 celdas y genera el laberinto, la salida, el inicio y las trampas y además cada jugador selecciona una pieza y se coloca en la casilla de inicio que es elegida al azar. **El método PlayGame** que controla el flujo del juego en un bucle hasta que un jugador gana. Maneja las acciones del jugador, como moverse, activar habilidades o salir del juego. **El método Movement** que controla el movimiento de las piezas en el tablero.Verifica si el jugador cae en una trampa y aplica los efectos correspondientes (reducción de velocidad, volver al inicio, etc.).Actualiza la posición de la pieza en el tablero y muestra el estado actual. **El método PrintBoard** que dibuja el tablero en la consola, mostrando las paredes, las trampas, la salida y las posiciones de los jugadores.

- **Clase Board:** Esta clase representa el tablero del juego y maneja la generación del laberinto y las trampas. Contiene **el método GenerateMaze** que genera un laberinto utilizando el algoritmo de Prim. Este algoritmo asegura que el laberinto sea perfecto, es decir, un laberinto en el que hay exactamente un camino entre cualquier par de celdas, sin áreas inaccesibles. Yo lo implemente para que primero se creara una cuadricula de celdas, donde todas inicialmente son paredes (CellType.Wall) Luego se crea una lista de paredes adyacentes a las celdas visitadas. Estas paredes son candidatas para ser "derribadas" y convertirse en caminos. Mientras haya paredes en la lista: Se selecciona una pared aleatoria de la lista y se verifica si la celda al otro lado de la pared no ha sido visitada. Si no ha sido visitada: Se derriba la pared (se convierte en un camino, CellType.Path) y se marca la celda como visitada. Luego se agregan las paredes adyacentes a la nueva celda a la lista de candidatos. Si ya ha sido visitada, se elimina la pared de la lista.El proceso termina cuando no quedan paredes en la lista.

  Además esta clase tiene **métodos como GenerateExit y GenerateStart** que colocan aleatoriamente la salida y el inicio en el laberinto, asegurándose de que no estén en una pared. Y el **método GenerateTraps** que coloca trampas aleatorias en el laberinto, como reducción de velocidad, volver al inicio y quedarse en el lugar.

- **Clase Game:** Esta clase maneja la lógica del juego, como el cambio de turnos mediante el **método ChangePlayer** y la verificación de si un jugador ha ganado mediante el **método Winner** que verifica si el jugador actual ha llegado a la celda de salida.

- **Clase Piece:** Representa las piezas (fichas) de los jugadores. Contiene la **propiedad Skill**: que permite que cada pieza tenga una habilidad especial que puede activarse durante el juego. También el **método Movement** que controla el movimiento de la pieza en el tablero, verificando si el movimiento es válido (no choca con una pared).