Laboratorio 3: Agentes Inteligentes en un Laberinto

Descripción General

Desarrollar un programa en C++ para un agente (un soldado) que, ubicado en una posición aleatoria dentro de un laberinto, deberá encontrar la salida automáticamente. El laberinto es de 10x10 celdas, donde cada celda tiene un valor de **1** o **0**. Un **1** representa un camino y un **0** un precipicio, por lo que el soldado debe evitar los precipicios mientras busca una ruta que lo conduzca a la salida, ubicada en la posición **[10,10]**.

Aplica la **programación orientada a objetos** en C++, como el uso de punteros, la depuración de código, el manejo de versiones, y la colaboración en equipo. Además, utilizaremos **referencias** y **punteros** para el paso de mensajes entre objetos, aplicando relaciones entre ellos para coordinar el movimiento del soldado.

Objetivos

1. **Desarrollar habilidades en el uso de punteros en C++** mediante el manejo de referencias y punteros entre objetos en el contexto de un laberinto.
2. **Aplicar la arquitectura modelo, controlador, vista**.
3. **Aplicar conceptos de relaciones entre clases**.
4. **Aplicar conceptos de relaciones entre objetos**
5. **Usar herramientas de depuración** promoviendo el análisis de errores de código.
6. **Usar herramientas de control de versiones**, promoviendo la colaboración en equipo y el análisis de errores de código.
7. **Implementar un algoritmo de búsqueda** que permita al agente encontrar la salida de forma automática.

Actividades

1. **Definición de Clases y Objetos**
    - Crea una clase llamada `Tablero` que representa el laberinto de 10x10 celdas. Este laberinto debe ser una matriz bidimensional donde cada celda tiene un valor de **0** o **1**.
    - Define la clase `Avatar`, que representará al agente en el laberinto. El avatar podrá moverse en las direcciones **arriba, abajo, izquierda, derecha**, siempre que el movimiento no lo lleve fuera del laberinto ni a una celda con valor **0**.
2. Implementa la arquitectura modelo, controlador, vista al juego
3. Implementa **al menos dos tipos de relaciones entre clases**
4. Implementa **al menos dos tipos de relaciones entre objetos**
5. **Implementación del Movimiento**
    - Utiliza **punteros** o **referencias** para actualizar la posición del soldado en el laberinto.
    - El soldado debe comenzar en una posición aleatoria del laberinto, como **[4,1]**, y debe buscar la salida en **[10,10]**.
    - El movimiento del avatar puede ser aleatorio entre las cuatro direcciones posibles.

    - Para encontrar la salida el avatar hace varios movimientos aleatorios.
6. **Manejo de Versiones y Depuración**
    - Colabora en equipo mediante un repositorio de control de versiones (GitHub) para manejar el progreso del programa.
    - Usa herramientas de **depuración** para identificar y corregir errores en el algoritmo de búsqueda.
    - Documenta tus cambios y coordina con tus compañeros de equipo, usando **ramas** y **commits** descriptivos para mejorar la colaboración y el rastreo de errores.

Criterios de Evaluación

- **Funcionalidad del Algoritmo:** El soldado debe encontrar la salida sin caer en precipicios mediante movimiento aleatorio.
- **Implementación de Punteros y Referencias:** Se evaluará el uso adecuado de punteros y referencias para la comunicación entre objetos.
- **Manejo de Versiones y Colaboración:** Se valorará el uso adecuado de Git para la colaboración en equipo.
- **Depuración Efectiva:** La habilidad para encontrar y corregir errores en el programa será evaluada durante las pruebas del programa.

Recomendaciones

- Asegúrese de manejar de manera adecuada los límites del laberinto para evitar errores de acceso a memoria.
- Utilice `const` donde sea posible para proteger referencias y punteros de modificación accidental.
- Colabore estrechamente con su equipo y utilice GitHub para organizar los cambios y mejorar la eficiencia en el desarrollo.