Mendewing

Documentación Técnica

Desarrolladores:

Víctor Jopia

Diego contreras

Profesor:

José Luis Veas

Fecha:

20/06/2025

Documentación Videojuego Estructura De Datos

Descripción del juego

El juego consiste en un tablero de 3 en raya, con la particularidad de que cada celda del tablero es un minijuego por separado, se implementaran los siguientes minijuegos de manera aleatoria en el tablero:

- Hex: Consiste en lograr conectar los extremos opuestos de cada jugador en un tablero de 11x11, es un juego por turnos.

- Adivinar el numero: ambos jugadores eligen un número secreto del 1 al 100, luego por turnos cada jugador debe intentar adivinar el número del otro, en caso de fallar se darán pistas para que cierte (por ejemplo: "El número del jugador 2 es mayor").

- Batalla de Cartas: Cada jugador recibe un mano aleatorio de 5 cartas de un mazo personal (cartas con valores del 1 al 15), se juegan 5 rondas en las que por turnos cada jugador deberá elegir la carta que estime más conveniente para dicha ronda, luego se comparan y gana la ronda quien haya jugado la carta más alta.

Cuando se ganen dichos minijuegos, el ganador colocara su ficha en el tablero principal, la idea es que gane 3 minijuegos en raya para ganar el juego principal, en caso de que ningún jugador pueda realizar un 3 en raya, ganara el jugador que tenga más fichas colocadas en el tablero.

Se utilizo el lenguaje de programación C++ y la biblioteca SFML-3.0.0 para el desarrollo del videojuego y de la GUI.

Modos de juego

- JvJ:

  Modo de juego de jugador contra jugador, por turnos ambos jugadores elijen la casilla a jugar dentro del tablero principal, luego juegan los respectivos minijuegos por turnos. Gana quien obtenga un 3 en raya en el tablero principal, en caso de que no se logre hacer un 3 en raya, ganara quien tenga mayor cantidad de fichas en el tablero.

- JvIa:

  Modo de juego de jugador contra la maquina (IA), el jugador se enfrenta a la IA en los 3 minijuegos, por turnos el jugador y la IA juegan, en cada minijuego siempre empieza el jugador, este modo cuenta con ciertas modificaciones, en el Hex, el tamaño del tablero se redujo a 7x7 para mejorar el desempeño de la IA.

Línea de compilación

g++ -Iinclude src/GUI/gui2.cpp src/Nodo.cpp src/Tablero.cpp src/MiniJuegos/AdivinaNumero/AdivinaNumeroV2.cpp src/MiniJuegos/BatallaDeCartas/BatallaDeCartas.cpp src/MiniJuegos/Hex/Hex.cpp src/IA/IAHex2.cpp src/IA/IaAdivinaNumero.cpp src/IA/IaBatallaCartas.cpp -o bin/testTableroNodoIa.exe -I"lib/SFML-3.0.0/include" -L"lib/SFML-3.0.0/lib" -lsfml-audio-s -lsfml-graphics-s -lsfml-window-s -lsfml-system-s -lvorbisenc -lvorbisfile -lvorbis -logg -lFLAC -lopengl32 -lfreetype -lwinmm -lgdi32 -DSFML_STATIC

Línea de ejecución

.\bin\testTableroNodoIa.exe

Implementación del juego

Implementación de estructuras de datos

-Se utilizaron nodos para construir el tablero principal el cual es una matriz de nodos de 3x3, el nodo contiene un puntero al minijuego que se le asigno, su fila y columna y su estado.

-Se utilizaron vectores para almacenar los valores de las cartas de los jugadores(vector<int>mazoJ1 y vector<int>mazoJ2) o para almacenar objetos de la gui como por ejemplo vector<sf:text> cartasTexto.

-Se utilizaron arrays para almacenar cosas como las fichas o símbolos del tablero.

-Se utilizaron grafos para el tablero y la funcionalidad de hex, donde cada casilla es un nodo y las conexiones entre nodos representan los movimientos válidos en el juego. La búsqueda en profundidad (DFS) se utiliza para verificar si un jugador ha ganado al conectar sus lados del tablero.

Implementación de algoritmos para la IA en modo jugador vs IA

La implementación de la inteligencia artificial en juegos como el hex y batalla de cartas se basa en algoritmos de búsqueda y evaluación que permiten a la IA tomar decisiones estratégicas.

IA para el minijuego Hex

Minimax

El algoritmo Minimax es una técnica de decisión utilizada en teoría de juegos y decisiones, especialmente en juegos de dos jugadores con información perfecta. El objetivo es minimizar la posible pérdida máxima en un escenario de peor caso.

- **Funcionamiento**
    - **Nodos y Árbol de Juego**: Minimax opera sobre un árbol de juego donde cada nodo representa un estado del juego. Los nodos se alternan entre niveles de "maximización" y "minimización".
    - **Evaluación**: En cada nodo hoja, se aplica una función de evaluación que asigna un valor numérico al estado del juego desde la perspectiva de la IA.
    - **Propagación de Valores**: Los valores se propagan hacia arriba en el árbol. En los niveles de maximización, se elige el valor máximo de los nodos hijos, y en los niveles de minimización, se elige el valor mínimo.

Poda Alfa-Beta

La poda alfa-beta es una optimización del algoritmo Minimax que reduce el número de nodos evaluados en el árbol de búsqueda, lo que mejora significativamente la eficiencia sin afectar el resultado final.

- **Funcionamiento**
    - **Valores Alfa y Beta**: Alfa es el mejor valor (máximo) que el jugador maximizador actualmente puede garantizar en ese nivel o superior. Beta es el mejor valor (mínimo) que el jugador minimizador actualmente puede garantizar en ese nivel o superior.
    - **Poda**: Durante la búsqueda, si en un nodo de minimización se encuentra un movimiento cuyo valor es menor que Alfa, el resto de los movimientos en ese nodo pueden ser ignorados - podados, ya que el jugador maximizador no permitirá que se llegue a ese estado. De manera similar, en un nodo de maximización, si un movimiento tiene un valor mayor que Beta, los demás movimientos pueden ser podados.

Análisis de rendimiento de la IA

En el minijuego hex, la IA presenta un comportamiento agresivo, prioriza conectar su propio camino y además intenta bloquear al rival, además ante un evento crítico, la IA será capaz de bloquear nuestro paso, para evitar una inminente derrota.

Aquí se definen los valores dados a ataque y defensa respectivamente:

Evaluación de Defensa

- **Amenaza Crítica**: Si el oponente está a una distancia de 2 o menos de completar su camino, se asigna una penalización de -2000 puntos.
- **Amenaza Seria**: Si el oponente está a una distancia de 4 o menos, se asigna una penalización de -1000 puntos.
- **Amenaza Moderada**: Si el oponente está a una distancia de 6 o menos, se asigna una penalización de -300 puntos.

Evaluación de Ataque

- **Oportunidad de Victoria**: Si la IA está a una distancia de 2 o menos de completar su camino, se asigna una bonificación de +2000 puntos.

- **Buena Posición**: Si la IA está a una distancia de 4 o menos, se asigna una bonificación de +1000 puntos.
- **Posición Aceptable**: Si la IA está a una distancia de 6 o menos, se asigna una bonificación de +400 puntos.
- **Al menos hay camino**: Si la IA tiene un camino posible, se asigna una bonificación de +100 puntos.

Bonificación por Control del Centro

- **Control del Centro**: Si la IA controla el centro del tablero, se asigna una bonificación de +50 puntos por cada casilla controlada en el área central.

Evaluación de Prioridad de Movimiento

- **Reducción de Distancia**: Si un movimiento reduce la distancia al objetivo, se asigna una bonificación de 300 puntos por cada unidad de distancia reducida.
- **Bloqueo al Oponente**: Si un movimiento aumenta la distancia del oponente a su objetivo, se asigna una bonificación de 200 puntos por cada unidad de distancia aumentada.
- **Posición Estratégica**: Se asigna una bonificación basada en la cercanía al centro del tablero, con un máximo de 10 puntos por casilla cercana al centro.
- **Conectividad con Fichas Propias**: Se asigna una bonificación de 100 puntos por cada vecino controlado por la IA.

Evaluación General del Tablero

- **Balance entre Ataque y Defensa**: Se utiliza un factor de 0.6 para el ataque y 0.4 para la defensa para calcular la puntuación final del tablero

En promedio la IA presenta un tiempo de respuesta de 2007.25 ms (2 segundos).

Este tiempo se ve influenciado por la profundidad del árbol (3 en las pruebas realizadas) y por la dificultad (500 en las pruebas realizadas), con menores niveles de profundidades, los tiempos se reducen drásticamente.

IA para el minijuego Batalla de cartas

La IA está diseñada para tomar decisiones estratégicas sobre qué carta jugar en cada turno, evaluando el estado actual del juego y anticipando posibles movimientos futuros.

Componentes Clave

1. **Evaluación del Estado**:
    - La función evaluarEstado calcula un valor basado en la diferencia de puntos entre la IA y el jugador, así como la suma de las cartas restantes de la IA. Esto proporciona una evaluación cuantitativa del estado actual del juego desde la perspectiva de la IA.
2. **Algoritmo Minimax**:
    - La función minimax es una implementación del algoritmo Minimax que evalúa los posibles movimientos futuros. Este algoritmo explora recursivamente los posibles movimientos, simulando cómo podría desarrollarse el juego.
    - **Parámetros**:
        - estado: El estado actual del juego.
        - profundidad: La profundidad de la búsqueda, que limita cuán lejos en el futuro se exploran los movimientos.
        - alpha y beta: Valores utilizados para la poda alfa-beta, una optimización que reduce el número de nodos evaluados en el árbol de búsqueda.
        - esMaximizando: Un booleano que indica si el turno actual es de la IA (maximizador) o del jugador (minimizador).
3. **Selección de Cartas**:
    - La función elegirCarta utiliza el algoritmo Minimax para decidir qué carta jugar. Evalúa cada carta disponible, simulando cómo afectaría al estado del juego, y elige la carta que maximiza la puntuación de la IA.
4. **Poda Alfa-Beta**:
    - Durante la ejecución del algoritmo Minimax, se utiliza la poda alfa-beta para eliminar ramas del árbol de búsqueda que no afectarán el resultado final. Esto mejora la eficiencia del algoritmo al evitar explorar movimientos que claramente no son óptimos.

IA para el minijuego Adivina el número

La IA está diseñada para adivinar un número secreto elegido por el oponente. Utiliza una estrategia de búsqueda binaria para reducir el rango de posibles números y hacer predicciones más precisas con cada intento.

Componentes Clave

1. **Inicialización**:
    - La función reiniciar establece los límites iniciales del rango de búsqueda. El rango inicial es de 1 a 100, que son los valores mínimo y máximo posibles para el número secreto.
2. **Búsqueda Binaria**:
    - La función hacerPrediccion implementa la lógica de búsqueda binaria. La IA comienza prediciendo el número en el medio del rango actual.
    - Si la predicción ya ha sido intentada anteriormente, la IA ajusta ligeramente su predicción incrementando el valor. Si este ajuste excede el límite máximo, reinicia al valor mínimo.
3. **Actualización de Límites**:
    - La función actualizarLimites ajusta los límites del rango de búsqueda basado en la retroalimentación recibida. Si el número secreto es mayor que la última predicción, el límite inferior (min) se ajusta a un valor mayor que la última predicción. Si el número secreto es menor, el límite superior (max) se ajusta a un valor menor que la última predicción.
4. **Registro de Intentos**:
    - La IA mantiene un registro de sus intentos previos en el vector intentosAnteriores. Esto le permite evitar repetir predicciones y ajustar su estrategia de búsqueda de manera más eficiente.

Documentación clases y archivos

A continuación, se procederá a describir todos los métodos de las clases utilizadas.

Clase Tablero.cpp y .h

La clase Tablero representa la lógica principal de un juego tipo 3x3 (similar a Tic-Tac-Toe) con integración de minijuegos en cada celda. Controla el estado del juego, la alternancia de turnos, y determina al ganador.

Atributos

- matrizNodos (std::vector<std::vector<Nodo>>): Matriz 3x3 que contiene los objetos Nodo del tablero.
- turnoActual (int): Indica de qué jugador es el turno actual (1 o 2).
- estadoJuego (EstadoJuego): Enumeración que representa el estado del juego (EN_CURSO, GANADOR_J1, GANADOR_J2, EMPATE).
- fichasJ1 (int): Contador de nodos ganados por el jugador 1.
- fichasJ2 (int): Contador de nodos ganados por el jugador 2.

Métodos Públicos

- Tablero():
  - Constructor que inicializa el tablero, establece el estado inicial y configura la matriz de nodos.
- void reiniciar():
  - Reinicia el estado del tablero completamente, limpiando nodos y reiniciando contadores y turno.
- void reiniciarEstadoJuego():
  - Reinicia el estado de juego sin reconstruir la matriz y limpia los nodos.
- bool jugarNodo(int fila, int columna):
  - Permite que el jugador actual ocupe un nodo. Actualiza turnos, estado del juego y conteo de fichas.
  - Parámetros:
    - fila: Fila del nodo a jugar.

        - columna: Columna del nodo a jugar.
    - Retorna true si el movimiento fue válido y exitoso, false en caso contrario.
- int getTurnoActual() const:
    - Retorna el jugador actual (1 o 2).
- EstadoJuego getEstadoJuego() const:
    - Retorna el estado actual del juego (EN_CURSO, GANADOR_J1).
- int getFichasJ1() const:
    - Retorna el número de fichas ganadas por el jugador 1.
- int getFichasJ2() const:
    - Retorna el número de fichas ganadas por el jugador 2.
- Nodo& getNodo(int fila, int columna):
    - Parámetros:
        - fila: Fila del nodo.
        - columna: Columna del nodo.
    - Retorna referencia al nodo correspondiente. Si es inválido, retorna una referencia a un nodo vacío.
- const Nodo& getNodo(int fila, int columna) const:
    - Parámetros:
        - fila: Fila del nodo.
        - columna: Columna del nodo.
    - Retorna referencia constante al nodo correspondiente. Si es inválido, retorna una referencia a un nodo vacío.
- void mostrarTablero() const:
    - Muestra el estado actual del tablero en consola, incluyendo información de los nodos y el turno.
- void mostrarEstadoJuego() const:
    - Muestra por consola el estado del juego en forma textual.
- bool esMovimientoValido(int fila, int columna) const:

- Parámetros:
    - fila: Fila del movimiento.
    - columna: Columna del movimiento.
- Retorna true si el movimiento es legal (dentro de límites, casilla activa y vacía).

Métodos Privados

- void inicializarMatriz():
    - Crea e inicializa la matriz 3x3 de objetos Nodo.
- void cambiarTurno():
    - Alterna el turno entre jugador 1 y jugador 2.
- void verificarVictoria():
    - Verifica si alguno de los jugadores ha ganado por fila, columna o diagonal. Si el tablero está lleno, determina el ganador por mayoría de fichas.
- bool verificarFilas():
    - Evalúa si hay una línea completa para declarar un ganador.
    - Retorna true si se encontró una línea ganadora.
- bool verificarColumnas():
    - Evalúa si hay una línea completa para declarar un ganador.
    - Retorna true si se encontró una línea ganadora.
- bool verificarDiagonales():
    - Evalúa si hay una línea completa para declarar un ganador.
    - Retorna true si se encontró una línea ganadora.
- bool tableroLleno():
    - Retorna true si no hay más nodos vacíos.
- void determinarGanadorPorFichas():
    - Establece el ganador dependiendo de quién tenga más fichas ocupadas, en caso de empate en las líneas.

Clase Nodo.cpp y .h

La clase Nodo representa una celda del tablero principal. Cada nodo puede tiene asociado un minijuego, el cual los jugadores deben ganar para conquistar esa celda.

Atributos

- estado (EstadoNodo): Estado actual del nodo (VACIO, JUGADOR1, JUGADOR2).
- fila (int): Fila donde está ubicado el nodo en la matriz.
- columna (int): Columna donde está ubicado el nodo en la matriz.
- activo (bool): Indica si el nodo está habilitado para jugar.
- miniJuego (std::unique_ptr<MiniJuego>): Puntero inteligente a un objeto de minijuego.
- tipoMiniJuego (TipoMiniJuego): Enumeración que indica el tipo de minijuego asignado.
- tieneMiniJuego (bool): Indica si el nodo tiene un minijuego asignado.

Constructores

- Nodo():
  - Constructor por defecto. Inicializa el nodo como vacío, activo y sin minijuego.
- Nodo(int f, int c):
  - Constructor con posición. Inicializa el nodo en la posición dada.
  - Parámetros:
    - f: Fila del nodo.
    - c: Columna del nodo.

Métodos Getters

- EstadoNodo getEstado() const:
    - Devuelve el estado actual del nodo.
    - Retorna 3stado actual del nodo.
- int getFila() const:
    - Devuelve la fila del nodo.
    - Retorna fila del nodo.
- int getColumna() const:
    - Devuelve la columna del nodo.
    - Retorno columna del nodo.
- bool estaActivo() const:
    - Indica si el nodo está activo.
    - Retorno true si el nodo está activo, false en caso contrario.
- bool estaVacio() const:
    - Indica si el nodo está vacío.
    - Retorno true si el nodo está vacío, false en caso contrario.

Métodos Setters

- void setEstado(EstadoNodo nuevoEstado):
    - Establece el estado del nodo.
    - Parámetros:
        - nuevoEstado: Nuevo estado del nodo.
- void setPosicion(int f, int c):
    - Establece la posición del nodo.
    - Parámetros:
        - f: Nueva fila del nodo.
        - c: Nueva columna del nodo.

- void setActivo(bool nuevoActivo):
    - Activa o desactiva el nodo.
    - Parámetros:
        - nuevoActivo: Estado de actividad del nodo.

Métodos de Juego

- bool ocuparNodo(EstadoNodo jugador):
    - Permite que un jugador intente ocupar el nodo. Si tiene minijuego, lo ejecuta antes.
    - Parámetros:
        - jugador: Estado del jugador que intenta ocupar el nodo.
    - Retorna true si se ocupa correctamente, false en caso contrario.
- void reiniciar():
    - Reinicia el estado del nodo y su minijuego, si existe.

Métodos de Minijuego

- void asignarMiniJuego(TipoMiniJuego tipo):
    - Asigna un minijuego al nodo según el tipo especificado.
    - Parámetros:
        - tipo: Tipo de minijuego a asignar.

- bool tieneMiniJuegoAsignado() const:
    - Indica si hay un minijuego asignado.
    - Retorna true si hay un minijuego asignado, false en caso contrario.
- TipoMiniJuego getTipoMiniJuego() const:
    - Devuelve el tipo de minijuego asignado.
    - Retorna tipo de minijuego asignado.

- MiniJuego* getMiniJuego() const:
    - Devuelve un puntero al minijuego asignado.
    - Retorna puntero al minijuego asignado.
- EstadoNodo jugarMiniJuego():
    - Ejecuta el minijuego asignado y retorna el ganador como EstadoNodo.
    - Retorna estado del ganador del minijuego.

Utilidades

- void mostrarInfo() const:
    - Muestra información del nodo (estado, posición, tipo de minijuego).

Clase Gui2.cpp

Gui2.cpp es la función principal del juego. Este archivo contiene la lógica principal del juego, incluyendo la gestión de ventanas, la interacción con el usuario y la integración de varios minijuegos.

Enumeraciones

- GameState: Define los estados del juego, como menu y game.
- AdivinaState: Define los estados para el juego de Adivina el Número.
- OpcionJuego: Define las opciones disponibles después de una partida, como volver_a_jugar y salir.

Funciones Principales

Funciones de Ventanas

- OpcionJuego mostrarVentanaOpciones()
    - Muestra una ventana con opciones para volver a jugar o salir después de una partida.
    - Retorna OpcionJuego que indica la opción seleccionada por el usuario.
- void mostrarVentanaGanador(int puntosJ1, int puntosJ2)
    - Muestra una ventana con el resultado final de la Batalla de Cartas.
    - Parámetros:
        - puntosJ1: Puntos del Jugador 1.
        - puntosJ2: Puntos del Jugador 2.
- void mostrarVentanaVictoriaTablero(char simboloGanador)
    - Muestra una ventana de victoria para el tablero principal.
    - Parámetros:
        - simboloGanador: Símbolo del ganador ('X' o 'O').
- void mostrarVentanaVictoriaFichas(char simboloGanador)
    - Muestra una ventana de victoria por fichas en el tablero principal.
    - Parámetros:

    - simboloGanador: Símbolo del ganador ('X' o 'O')..
- void mostrarVentanaVictoria(int jugadorGanador, int numeroSecreto, Tablero& tablero)
    - Muestra una ventana de victoria para el minijuego.
    - Parámetros:
        - jugadorGanador: Jugador ganador.
        - numeroSecreto: Número secreto en el juego de Adivina el Número.
        - tablero: Referencia al tablero principal.
- void mostrarVentanaAyuda()
    - Muestra una ventana de ayuda con instrucciones generales del juego.
- void mostrarVentanaAyudaHex()
    - Muestra una ventana de ayuda específica para el juego Hex.
- void mostrarVentanaAyudaAdivinaNumero()
    - Muestra una ventana de ayuda específica para el juego Adivina el Número.
- void mostrarVentanaAyudaBatallaCartas()
    - Muestra una ventana de ayuda específica para el juego Batalla de Cartas.

Funciones de Minijuegos

- void abrirHex(int casilla, Tablero& tablero, sf::Music& musicaFondo)
    - Abre el minijuego Hex.
    - Parámetros:
        - casilla: Casilla del tablero principal donde se juega el minijuego.
        - tablero: Referencia al tablero principal.
- void abrirAdivinaNumero(int casilla, Tablero& tablero, sf::Music& musicaFondo)
    - Abre el minijuego Adivina el Número.
    - Parámetros:
        - casilla: Casilla del tablero principal donde se juega el minijuego.

    - tablero: Referencia al tablero principal.
    - musicaFondo: Referencia a la música de fondo.
- void abrirBatallaCartas(int casilla, Tablero& tablero, sf::Music& musicaFondo)
    - Abre el minijuego Batalla de Cartas.
    - Parámetros:
        - casilla: Casilla del tablero principal donde se juega el minijuego.
        - tablero: Referencia al tablero principal.
        - musicaFondo: Referencia a la música de fondo.
- void abrirHexVsIA(int casilla, Tablero& tablero, sf::Music& musicaFondo)
    - Abre el minijuego Hex en modo Jugador vs IA.
    - Parámetros:
        - casilla: Casilla del tablero principal donde se juega el minijuego.
        - tablero: Referencia al tablero principal.
        - musicaFondo: Referencia a la música de fondo.
- void abrirAdivinaNumeroVsIA(int casilla, Tablero& tablero, sf::Music& musicaFondo)
    - Abre el minijuego Adivina el Número en modo Jugador vs IA.
    - Parámetros:
        - casilla: Casilla del tablero principal donde se juega el minijuego.
        - tablero: Referencia al tablero principal.
        - musicaFondo: Referencia a la música de fondo.
- void abrirBatallaCartasVsIA(int casilla, Tablero& tablero, sf::Music& musicaFondo)
    - Abre el minijuego Batalla de Cartas en modo Jugador vs IA.
    - Parámetros:
        - casilla: Casilla del tablero principal donde se juega el minijuego.
        - tablero: Referencia al tablero principal.
        - musicaFondo: Referencia a la música de fondo.

Funciones Auxiliares

- void inicializarMinijuegosAleatorios(Tablero& tablero)
  - Inicializa los minijuegos en el tablero de manera aleatoria.
  - Parámetros:
    - tablero: Referencia al tablero principal.
- string obtenerSimboloMinijuego(TipoMiniJuego tipo)
  - Obtiene el símbolo asociado a un tipo de minijuego.
  - Parámetros:
    - tipo: Tipo de minijuego.
- CircleShape crearHexagono(float radio, sf::Vector2f posicion)
  - Crea un hexágono para el juego Hex.
  - Parámetros:
    - radio: Radio del hexágono.
    - posicion: Posición del hexágono.
  - Retorna sf::CircleShape que representa el hexágono.
- bool ventanaJugador(bool esJugador1, std::vector<int>& valoresCartas, int& cartaSeleccionada, int puntosJ1, int puntosJ2, bool& iaPensando)
  - Muestra la ventana para que un jugador seleccione una carta en el juego Batalla de Cartas.
  - Parámetros:
    - esJugador1: Indica si es el Jugador 1.
    - valoresCartas: Vector de cartas disponibles.
    - cartaSeleccionada: Referencia a la carta seleccionada.
    - puntosJ1: Puntos del Jugador 1.
    - puntosJ2: Puntos del Jugador 2.
    - iaPensando: Indica si la IA está pensando.
  - Retorna true si se seleccionó una carta, false en caso contrario.

Función Principal

- int main()
    - Función principal que inicializa el juego, maneja la ventana principal, los eventos y el flujo del juego.
    - Retorna 0 si el juego se ejecuta correctamente.

Clase Minijuego.h

La clase MiniJuego es una clase abstracta que sirve como interfaz base para todos los minijuegos del sistema.

Atributos

- estado (EstadoMiniJuego):
    - Representa el estado actual del minijuego. Puede ser EN_PROGRESO, GANADOR_JUGADOR1, o GANADOR_JUGADOR2.
- juegoTerminado (bool):
    - Indica si el minijuego ha concluido.

Constructores

- MiniJuego():
    - Constructor por defecto. Inicializa el minijuego en estado EN_PROGRESO y marca que no ha terminado.
- ~MiniJuego():
    - Destructor virtual por defecto. Permite el manejo adecuado de herencia polimórfica.

Métodos Comunes

- EstadoMiniJuego getEstado() const:
    - Retorna el estado actual del minijuego (EN_PROGRESO, GANADOR_JUGADOR1, GANADOR_JUGADOR2).
- bool estaTerminado() const:
    - Retorna true si el minijuego ha terminado, de lo contrario false.

Métodos Virtuales Puros

- bool procesarMovimiento(int jugador, int input):
    - Método abstracto que debe ser implementado por los minijuegos. Procesa un movimiento realizado por el jugador.
    - Parámetros:
        - jugador (int): El número del jugador (usualmente 1 o 2).
        - input (int): El valor introducido por el jugador (por ejemplo, intento de adivinanza o selección de carta).
    - Retorna true si el movimiento fue válido y aceptado; false en caso contrario.
- void mostrarEstado() const:
    - Método abstracto que debe mostrar el estado actual del minijuego por consola. Su implementación depende de la lógica de cada minijuego.
- void reiniciar():
    - Método abstracto que reinicia el minijuego a su estado inicial, permitiendo que se juegue nuevamente desde cero.

Métodos Protegidos

- void terminarJuego(EstadoMiniJuego ganador):
    - Marca el juego como terminado y asigna el estado correspondiente al jugador ganador.
    - Parámetros:
        - ganador (EstadoMiniJuego): Valor que representa quién ganó el minijuego.

Clase Hex.cpp y .h

La clase Hex representa el minijuego de conexión Hex, en el que dos jugadores deben formar un camino continuo entre lados opuestos del tablero hexagonal.

Atributos

- grafo (GrafoHex): Representa el tablero como un grafo de nodos hexagonales interconectados.
- jugadorActual (int): Indica qué jugador tiene el turno actual (1 o 2).
- juegoTerminado (bool): Indica si la partida ha finalizado.
- ganador (int): Jugador que ganó la partida (1 o 2).
- primerMovimiento (bool): Bandera que indica si es el primer turno de la partida.
- puedeRobar (bool): Indica si el jugador 2 puede aplicar la regla del robo.
- ultimaFila, ultimaColumna (int): Posición del último movimiento del jugador 1.
- modoIA (bool): Indica si el juego se está jugando contra la IA.

Constructores

- Hex(int tamaño = 11, bool _modoIA = false):
    - Constructor que inicializa el juego Hex con un tablero de tamaño personalizado y opcionalmente modo IA.

Métodos Derivados de MiniJuego

- bool procesarMovimiento(int jugador, int input):
    - Procesa un movimiento del jugador, convirtiendo el input en coordenadas y actualizando el estado del juego.
    - Parámetros:
        - jugador (int): El número del jugador (usualmente 1 o 2).
        - input (int): El valor introducido por el jugador.
    - Retorna true si el movimiento fue válido.
- void mostrarEstado() const:

    - Muestra el estado actual del tablero por consola.
- void reiniciar():
    - Reinicia el juego completamente, restaurando el grafo, el turno y los indicadores de victoria.

Métodos de Juego

- void jugar():
    - Inicia y gestiona el flujo completo del juego en modo consola interactiva.
- bool hacerMovimiento(int fila, int col):
    - Realiza un movimiento en la posición dada si es válida.
    - Parámetros:
        - fila (int): Fila del movimiento.
        - col (int): Columna del movimiento.
    - Retorna true si el movimiento fue exitoso.
- bool aplicarReglaRobo():
    - Permite al jugador 2 intercambiar colores con el jugador 1 si es el segundo turno.
    - Retorna true si la regla fue aplicada.
- int getGanador() const:
    - Retorna el número del jugador ganador (1 o 2).
- bool estaTerminado() const:
    - Retorna true si la partida ha concluido.
- void mostrarInstrucciones() const:
    - Muestra instrucciones básicas del juego Hex.

Métodos de Acceso

- EstadoCasilla getCasilla(int fila, int col) const:
    - Retorna el estado de una casilla específica del tablero.
    - Parámetros:
        - fila (int): Fila de la casilla.
        - col (int): Columna de la casilla.
- int getJugadorActual() const:
    - Retorna el número del jugador que debe mover.
- bool getPuedeRobar() const:
    - Retorna true si el jugador 2 puede aplicar la regla del robo.
- const GrafoHex& getGrafo() const:
    - Devuelve una referencia constante al grafo del tablero.
- int getTamañoTablero() const:
    - Devuelve el tamaño actual del tablero.
- bool esCasillaVacia(int fila, int col) const:
    - Indica si la casilla está vacía.
    - Parámetros:
        - fila (int): Fila de la casilla.
        - col (int): Columna de la casilla.

Clase BatallaDeCartas.cpp y .h

La clase BatallaDeCartas representa un minijuego de cinco rondas donde dos jugadores se enfrentan utilizando cartas con valores entre 1 y 15. Quien consiga mayor cantidad de puntos gana el juego.

Atributos

- mazoJugador1 (std::vector<int>): Mazo de cartas para el Jugador 1.
- mazoJugador2 (std::vector<int>): Mazo de cartas para el Jugador 2.
- rondasGanadasJ1 (int): Contador de rondas ganadas por el Jugador 1.
- rondasGanadasJ2 (int): Contador de rondas ganadas por el Jugador 2.
- totalRondas (int): Número total de rondas del juego (por defecto, 5).
- rng (std::mt19937): Generador de números aleatorios para barajar las cartas.

Constructores

- BatallaDeCartas():
    - Inicializa los mazos, mezcla las cartas y pone los contadores de rondas ganadas en 0.

Métodos Principales del Juego

- void inicializarMazos():
    - Crea y baraja los mazos con cartas del 1 al 5 (tres copias por valor).
- void mostrarTitulo():
    - Muestra en consola el nombre del minijuego y sus reglas.
- void mostrarMano(const std::vector<int>& mano, const std::string& jugador):
    - Muestra las primeras cinco cartas visibles de la mano de un jugador.
    - Parámetros:
        - mano (std::vector<int>): Vector de cartas del jugador.
        - jugador (std::string): Nombre del jugador.

- int seleccionarCarta(std::vector<int>& mano, const std::string& jugador):
    - Permite al jugador humano seleccionar una carta o elige una aleatoria para la IA. Devuelve el valor de la carta jugada y la elimina del mazo.
    - Parámetros:
        - mano (std::vector<int>): Vector de cartas del jugador.
        - jugador (std::string): Nombre del jugador.
    - Retorna el valor de la carta seleccionada.
- void mostrarResultadoRonda(int cartaJ1, int cartaJ2, int ronda):
    - Muestra los resultados de una ronda y actualiza las rondas ganadas.
    - Parámetros:
        - cartaJ1 (int): Valor de la carta del Jugador 1.
        - cartaJ2 (int): Valor de la carta del Jugador 2.
        - ronda (int): Número de la ronda actual.
- void mostrarResultadoFinal():
    - Muestra el resultado final de la partida indicando el ganador.
- void procesarRonda(int cartaJ1, int cartaJ2, int numeroRonda):
    - Elimina las cartas jugadas de los mazos y muestra el resultado de la ronda.
    - Parámetros:
        - cartaJ1 (int): Valor de la carta del Jugador 1.
        - cartaJ2 (int): Valor de la carta del Jugador 2.
        - numeroRonda (int): Número de la ronda actual.
- void jugar():
    - Controla el flujo completo del juego en consola, incluyendo la interacción por turnos, pausas y resultados.
- void reiniciarJuego():
    - Reinicia el estado del juego llamando al método reiniciar.

Métodos Derivados de MiniJuego

- bool procesarMovimiento(int jugador, int input):
    - Valida si el input del jugador está dentro del rango permitido para las cartas disponibles.
    - Parámetros:
        - jugador (int): Número del jugador (1 o 2).
        - input (int): Valor de la carta seleccionada.
    - Retorna true si el movimiento es válido.
- void mostrarEstado() const:
    - Muestra el estado actual del juego, incluyendo cartas restantes y rondas ganadas.
- void reiniciar():
    - Reinicia el juego restableciendo los contadores de rondas y los mazos.

Clase AdivinaElNumero.cpp y .h

La clase AdivinaNumero implementa un minijuego de dos jugadores en el que cada jugador elige un número secreto entre 1 y 100. Luego, se alternan turnos para adivinar el número secreto del oponente.

Atributos

- numeroJugador1 (int): Número secreto elegido por el Jugador 1.
- numeroJugador2 (int): Número secreto elegido por el Jugador 2.
- numeroJ1Elegido (bool): Indica si el Jugador 1 ha elegido su número secreto.
- numeroJ2Elegido (bool): Indica si el Jugador 2 ha elegido su número secreto.
- turnoActual (int): Indica de quién es el turno actual (1 para Jugador 1, 2 para Jugador 2).
- generador (std::mt19937): Generador de números aleatorios.
- intentosJugador1 (std::vector<int>): Vector que almacena los intentos del Jugador 1.
- intentosJugador2 (std::vector<int>): Vector que almacena los intentos del Jugador 2.

Constructor

- AdivinaNumero():
    - Inicializa el juego, estableciendo los números secretos de los jugadores a -1, indicando que no han sido elegidos, y configurando el turno actual al Jugador 1.

Métodos Principales

- bool establecerNumeroSecreto(int jugador, int numero):
    - Establece el número secreto para un jugador si es válido y no ha sido establecido previamente.
    - Parámetros:
        - jugador: Número del jugador (1 o 2).
        - numero: Número secreto a establecer.

    - Retorna true si el número fue establecido correctamente, false en caso contrario.
- bool validarNumero(int numero) const:
    - Valida si un número está dentro del rango permitido (1 a 100).
    - Parámetros:
        - numero: Número a validar.
    - Retorna true si el número es válido, false en caso contrario.
- bool ambosNumerosElegidos() const:
    - Verifica si ambos jugadores han elegido sus números secretos.
    - Retorna true si ambos jugadores han elegido sus números, false en caso contrario.
- bool procesarMovimiento(int jugador, int numeroAdivinado):
    - Procesa un intento de adivinar el número secreto del oponente. Si el número es correcto, termina el juego.
    - Parámetros:
        - jugador: Número del jugador que está adivinando.
        - numeroAdivinado: Número que el jugador intenta adivinar.
    - Retorna true si el número fue adivinado correctamente, false en caso contrario.
- int getNumeroJugador(int jugador) const:
    - Obtiene el número secreto de un jugador.
    - Parámetros:
        - jugador: Número del jugador (1 o 2).
    - Retorna Número secreto del jugador, o -1 si el jugador no es válido.

- void setNumeroJugador(int jugador, int numero):
    - Establece el número secreto para un jugador.
    - Parámetros:
        - jugador: Número del jugador (1 o 2).
        - numero: Número secreto a establecer.
- const std::vector<int>& getIntentosJugador(int jugador) const:
    - Obtiene los intentos de adivinación de un jugador.
    - Parámetros:
        - jugador: Número del jugador (1 o 2).
    - Retorna Vector de intentos del jugador.
- void mostrarEstado() const:
    - Muestra el estado actual del juego, incluyendo el turno actual, si ambos jugadores han elegido sus números secretos, y los números secretos si el juego ha terminado.
- void reiniciar():
    - Reinicia el juego, restableciendo el estado a "en progreso", limpiando los números secretos y los intentos de los jugadores.

Documentación de los algoritmos IA

Clase IaHex2.cpp

La clase IAHex implementa una inteligencia artificial para el juego Hex, utilizando el algoritmo Minimax con poda alfa-beta para tomar decisiones estratégicas. La IA evalúa el tablero y decide los movimientos basándose en la proximidad a la victoria y la defensa contra el oponente.

Atributos

- profundidadMaxima (int): Profundidad máxima que utilizará la IA al aplicar Minimax.
- dificultad (int): Valor de dificultad de la IA, afecta decisiones estratégicas.
- nodosExploradosTemp (int): Cantidad de nodos explorados durante la última búsqueda Minimax.
- tiempoTotalMovimientos (double): Tiempo total invertido en calcular movimientos.
- cantidadMovimientos (int): Contador de movimientos realizados por la IA.

Constructor

- IAHex(int profundidad, int dificultadIA):
    - Constructor de la clase IAHex.
    - Parámetros:
        - profundidad: Profundidad máxima deseada para Minimax.
        - dificultadIA: Nivel de dificultad inicial.
    - Inicializa los valores internos, limitando la profundidad a un máximo de 10. Muestra información de depuración en consola.

Métodos Públicos

- void setDificultad(int nuevaDificultad):
  - Establece la dificultad de la IA.
  - Parámetros:
    - nuevaDificultad: Nuevo valor de dificultad.
- void setProfundidadMaxima(int nuevaProfundidad):
  - Establece la profundidad máxima para Minimax.
  - Parámetros:
    - nuevaProfundidad: Nueva profundidad deseada.
- double getTiempoPromedioMovimiento() const:
  - Retorna el tiempo promedio de cálculo por movimiento.
- double getTiempoTotalMovimientos() const:
  - Retorna el tiempo total invertido en todos los movimientos de la IA.
- int getCantidadMovimientos() const:
  - Retorna la cantidad total de movimientos realizados por la IA.
- void resetEstadisticasTiempo():
  - Reinicia las estadísticas de tiempo y cantidad de movimientos.

Métodos Privados

- std::vector<Posicion> obtenerVecinosHex(int fila, int columna):
  - Devuelve los vecinos válidos de una celda hexagonal en el tablero 7x7.
  - Parámetros:
    - fila: Fila de la celda.
    - columna: Columna de la celda.
  - Retorna un vector con posiciones vecinas.

- int calcularDistanciaCamino(const Hex& estadoJuego, EstadoCasilla jugador, bool esVertical):
    - Calcula la distancia mínima entre bordes del tablero para un jugador usando BFS.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
        - jugador: Jugador a analizar.
        - esVertical: true para vertical (IA), false para horizontal (humano).
    - Retorna un entero con la distancia más corta o 999 si no hay camino.
- std::vector<Posicion> obtenerMovimientosDisponibles(const Hex& estadoJuego):
    - Genera una lista de movimientos válidos ordenados por prioridad estratégica.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
    - Retorna un vector con las mejores posiciones disponibles.
- int evaluarDefensa(const Hex& estadoJuego):
    - Evalúa el peligro de que el oponente gane y penaliza según proximidad.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
    - Retorna una puntuación negativa si hay amenaza.
- int evaluarAtaque(const Hex& estadoJuego):
    - Evalúa qué tan cerca está la IA de ganar y premia el control del centro.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
    - Retorna una puntuación positiva.

- int evaluarPrioridadMovimiento(const Hex& estadoJuego, const Posicion& mov):
    - Evalúa un movimiento en base a ataque, defensa, centro y conectividad.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
        - mov: Movimiento candidato.
    - Retorna un entero con la prioridad.
- int evaluarTablero(const Hex& estadoJuego):
    - Combina ataque y defensa con ponderaciones para evaluar el tablero.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
    - Retorna una puntuación final del estado del juego.
- Posicion buscarMovimientoOfensivo(const Hex& estadoJuego, const std::vector<Posicion>& movimientos):
    - Busca un movimiento que reduzca la distancia a la victoria.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
        - movimientos: Lista de movimientos válidos.
    - Retorna un movimiento ofensivo si existe, si no, (-1, -1).
- bool esAmenazaCritica(const Hex& estadoJuego, const Posicion& mov):
    - Verifica si dejar libre una celda permite al oponente ganar en su siguiente turno.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
        - mov: Movimiento a simular.
    - Retorna true si representa amenaza crítica, false si no.

- Posicion calcularMejorMovimiento(Hex& estadoJuego):
    - Determina el mejor movimiento para la IA considerando defensa, ataque y Minimax.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
    - Retorna la posición del mejor movimiento calculado.
- int minimax(Hex& estadoJuego, int profundidad, int alfa, int beta, bool esMaximizador):
    - Aplica Minimax con poda alfa-beta para evaluar posibles jugadas futuras.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
        - profundidad: Nivel de profundidad actual.
        - alfa: Valor mínimo que el maximizador puede garantizar.
        - beta: Valor máximo que el minimizador puede permitir.
        - esMaximizador: true si es turno de la IA, false si es del oponente.
    - Retorna el valor de evaluación del estado actual.
- int evaluarProgresoVictoria(const Hex& estadoJuego):
    - Compara la distancia a la victoria de ambos jugadores.
    - Parámetros:
        - estadoJuego: Estado actual del tablero.
    - Retorna una puntuación positiva si la IA está más cerca de ganar, negativa si el oponente lo está.

Clase IaBatallaCartas.cpp

La clase IaBatallaCartas implementa una inteligencia artificial para el juego de cartas "Batalla de Cartas". Utiliza el algoritmo Minimax con poda alfa-beta para tomar decisiones estratégicas sobre qué carta jugar.

Atributos

- puntosIA (int): Puntos acumulados por la IA.
- puntosJugador (int): Puntos acumulados por el jugador.
- dificultad (int): Nivel de dificultad de la IA.
- cartasDisponibles (std::vector<int>): Cartas disponibles para la IA.

Constructor

- IaBatallaCartas(int nivelDificultad):
    - Inicializa la IA con un nivel de dificultad y reinicia el estado del juego.
    - Parámetros:
        - nivelDificultad: Nivel de dificultad de la IA.

Métodos Principales

- int evaluarEstado(const Estado& estado):
    - Evalúa el estado actual del juego, calculando un valor basado en la diferencia de puntos y las cartas restantes.
    - Parámetros:
        - estado: Estado actual del juego.
    - Retorna valorEstado: Valor calculado del estado actual.
- int minimax(Estado estado, int profundidad, int alpha, int beta, bool esMaximizando):
    - Implementa el algoritmo Minimax con poda alfa-beta para evaluar movimientos futuros.
    - Parámetros:
        - estado: Estado actual del juego.

        - profundidad: Profundidad actual de la búsqueda.
        - alpha: Valor alfa para la poda alfa-beta.
        - beta: Valor beta para la poda alfa-beta.
        - esMaximizando: Indica si el jugador actual es el maximizador.
    - Retorna mejor valor calculado para el estado.
- int elegirCarta(int cartaJugador):
    - Elige la mejor carta para jugar utilizando el algoritmo Minimax.
    - Parámetros:
        - cartaJugador: Carta jugada por el jugador.
    - Retorna mejorCarta: Mejor carta calculada para jugar.
- void actualizarCartas(const std::vector<int>& nuevasCartas):
    - Actualiza las cartas disponibles para la IA.
    - Parámetros:
        - nuevasCartas: Vector de nuevas cartas disponibles.
- void actualizarPuntos(int pIA, int pJugador):
    - Actualiza los puntos de la IA y del jugador.
    - Parámetros:
        - pIA: Puntos de la IA.
        - pJugador: Puntos del jugador.
- void reiniciar():
    - Reinicia el estado del juego, limpiando las cartas disponibles y restableciendo los puntos.

Clase IaAdivinaNumero.cpp

La clase IaAdivinaNumero implementa una inteligencia artificial para el juego "Adivina el Número". Utiliza una estrategia de búsqueda binaria para adivinar el número secreto del oponente, ajustando sus predicciones basándose en las pistas recibidas.

Atributos

- min (int): Límite inferior del rango de búsqueda.
- max (int): Límite superior del rango de búsqueda.
- intentosAnteriores (std::vector<int>): Vector que almacena los intentos previos de la IA para evitar repeticiones.

Constructor

- IaAdivinaNumero():
    - Inicializa la IA y reinicia los límites de búsqueda y los intentos anteriores.

Métodos Principales

- void reiniciar():
    - Reinicia los límites de búsqueda y los intentos anteriores.
- int hacerPrediccion():
    - Realiza una predicción del número secreto utilizando una estrategia de búsqueda binaria. Si la predicción ya fue intentada, ajusta ligeramente el valor.
    - Retorna prediccion: Número predicho por la IA.
- void actualizarLimites(bool esMayor):
    - Actualiza los límites de búsqueda basándose en si el número a adivinar es mayor o menor que la última predicción.
    - Parámetros:
        - esMayor: Indica si el número a adivinar es mayor que la última predicción.

- bool numeroYaIntentado(int numero) const:
    - Verifica si un número ya ha sido intentado previamente por la IA.
    - Parámetros:
        - numero: Número a verificar.
    - Retorna true si el número ya fue intentado, false en caso contrario.

Clase JuegoPorconsola.cpp

Este programa permite a los jugadores interactuar con el juego a través de la consola, mostrando el estado del tablero, gestionando los turnos y determinando el ganador.

Metodos

- void mostrarMenu():
    - Muestra el menú principal del juego en la consola.
- void mostrarInstrucciones():
    - Muestra las instrucciones del juego en la consola, incluyendo cómo jugar y las reglas básicas.
- void inicializarMinijuegosAleatorios(Tablero& tablero):
    - Inicializa minijuegos aleatorios en cada casilla del tablero.
    - Parámetros:
        - tablero: Referencia al objeto Tablero donde se asignarán los minijuegos.
- void mostrarTableroConMinijuegos(const Tablero& tablero):
    - Muestra el estado actual del tablero en la consola, incluyendo los tipos de minijuegos y el estado de cada casilla.
    - Parámetros:
        - tablero: Referencia constante al objeto Tablero que se mostrará.
- int main():
    - Función principal que inicializa el juego, muestra el menú y las instrucciones, y gestiona el flujo del juego.
    - Permite a los jugadores ingresar sus movimientos y muestra el estado del juego después de cada turno.
    - Finaliza el juego cuando se determina un ganador o cuando el tablero está lleno, mostrando el resultado final.