Dibujo con letras blancas

Descripción generada automáticamente con confianza media

**CUATRO EN LÍNEA** **– PARTE 2**

**Proyecto para el curso de Introducción a la Ciencia de la Computación**

**SECCIÓN**

LABORATORIO 2.02

**DOCENTE**

GALLEGOS VELGARA, HENRY GIOVANNY

|  |  |
| --- | --- |
| **Integrantes** | **Código** |
| Joel Modesto Cayllahua Hilario | 202410731 |
| Garro Arana, Micaela Soledad | 202410675 |
| Falí Ferdinand Araoz Arana | 202410721 |

**Resumen**

Este informe describe la segunda parte del desarrollo del juego "4 en línea" programado en Python para el curso de Programación 1 en la UTEC, que amplía las funcionalidades del juego permitiendo partidas contra la computadora. Se añadieron distintas funciones al código presentado en la primera parte. Se añade al menú principal la opción de jugar contra la computadora, solicitando el nombre del jugador y la elección de quién comienza el juego. La lógica de la computadora se diseña para realizar movimientos estratégicos que buscan ganar y evitar que el jugador gane, analizando el tablero y detectando patrones.

Además, se implementa el almacenamiento y recuperación de juegos realizados en un archivo JSON para permitir la persistencia de datos y la visualización de estadísticas, como la cantidad de juegos totales y por jugador, partidas ganadas por la computadora, lista de jugadores ordenados por cantidad de juegos, tiempo promedio de los juegos y cantidad de movimientos por juego. Ejemplos de ejecución del programa demuestran su funcionalidad, incluyendo la interacción con la computadora y la visualización de estadísticas, y se presentan conclusiones sobre el aprendizaje adquirido y recomendaciones para futuras mejoras.

**Introducción**

El juego "4 en línea" es un clásico juego de estrategia. Su simplicidad en las reglas contrasta con la profundidad estratégica necesaria para dominarlo, lo que lo convierte en un tema ideal para un proyecto de programación educativa. Este proyecto busca implementar este juego utilizando el lenguaje de programación Python, permitiendo a los jugadores competir para alinear cuatro fichas consecutivas en todas las posibles direcciones dentro de un tablero de 6x7.

En esta segunda parte del proyecto, se ha añadido la opción de jugar contra la computadora. La inteligencia artificial diseñada para este propósito no solo realiza movimientos estratégicos, sino que también puede analizar el tablero en busca de patrones, lo que añade una capa adicional de complejidad y aprendizaje. Un dato interesante es que este tipo de juegos contra la computadora puede mejorar significativamente las habilidades de resolución de problemas y pensamiento crítico en los jugadores.

**Definiciones**

Para entender el código del programa, debemos definir las funciones clave que se utilizan. A continuación, se presentan las definiciones de estos elementos:

* Función guardar\_partidas(): Guarda las partidas realizadas en un archivo JSON para su posterior consulta. Utiliza el módulo `json` para serializar la lista de partidas y almacenarlas en un archivo.
* Función cargar\_partidas(): Carga las partidas guardadas desde un archivo JSON al inicio del programa. Si el archivo no existe, inicializa la lista de partidas vacía.
* Función heurístico(): Evalúa el estado actual del tablero sumando las puntuaciones de todas las líneas posibles (filas, columnas y diagonales). Esta evaluación se utiliza para determinar la jugada más favorable para la computadora.
* Función heurístico\_linea(linea): Evalúa una línea específica (ya sea fila, columna o diagonal) para determinar su puntuación basándose en la presencia de fichas del jugador y la computadora.
* Función evaluar\_4\_elementos(linea): Evalúa un conjunto de cuatro elementos en una línea para asignar una puntuación basada en la cantidad de fichas del jugador y la computadora.
* Función get\_diagonales(): Genera todas las diagonales posibles del tablero, tanto de izquierda a derecha como de derecha a izquierda. Estas diagonales son importantes para evaluar posibles victorias diagonales.
* Función get\_abiertos(): Retorna una lista de las posiciones de las columnas que están abiertas (no están llenas). Verifica la primera fila de cada columna para determinar si está vacía.
* Función minimax\_alpha\_beta(profundidad, alpha, beta, jugador): Implementa el algoritmo Minimax con poda alfa-beta para optimizar la búsqueda de la mejor jugada posible. Este algoritmo evalúa todas las posibles jugadas hasta una profundidad determinada y utiliza la función heurística para asignar una puntuación a cada estado del tablero.

**Funcionamiento del Programa**

Se implementaron todas estas funciones para permitir la toma de decisiones de la IA, la gestión de tablero y para facilitar la interacción del jugador con el programa. A continuación, se detallan cada una de estas funciones y cómo se relacionan y contribuyen con el funcionamiento del juego.

**Guardar las partidas**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Función guardar\_partidas(): Esta función guarda las partidas realizadas en un archivo JSON. Utiliza el módulo `json` para serializar la lista de partidas `partidas\_hechas` y almacenarlas en un archivo especificado por `archivo\_partidas`.

Apertura del archivo: Abre el archivo especificado en modo escritura.

Serialización y almacenamiento: Utiliza `json.dump()` para escribir la lista de partidas en el archivo con un formato indentado para facilitar la lectura.

**Cargar las partidas**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**Función cargar\_partidas():** Esta función carga las partidas guardadas desde un archivo JSON al inicio del programa. Si el archivo no existe, inicializa la lista de partidas como una lista vacía.

Intento de carga: Intenta abrir y leer el archivo especificado.

Manejo de excepciones: Si el archivo no se encuentra, inicializa `partidas\_hechas` como una lista vacía.

**Función heurística para evaluar el tablero**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Función heuristico(): Evalúa el estado actual del tablero sumando las puntuaciones de todas las líneas posibles (filas, columnas y diagonales). Esta evaluación se utiliza para determinar la jugada más favorable para la computadora.

Obtención de filas: Utiliza la variable global `get\_filas` para obtener todas las filas del tablero.

Obtención de diagonales: Llama a la función `get\_diagonales()` para obtener todas las diagonales posibles del tablero.

Conversión de columnas: Convierte las filas en columnas para evaluarlas.

Evaluación de todas las líneas: Combina filas, columnas y diagonales, y evalúa cada línea utilizando la función `heuristico\_linea()`. Si la puntuación es muy alta o muy baja (indicando una victoria clara), se retorna inmediatamente.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Función heuristico\_linea(linea): Evalúa una línea específica (ya sea fila, columna o diagonal) para determinar su puntuación basándose en la presencia de fichas del jugador y la computadora.

Evaluación de segmentos de cuatro: Recorre la línea evaluando segmentos de cuatro elementos utilizando la función `evaluar\_4\_elementos()`.

Detección de victorias: Si se detecta una victoria (puntuación muy alta o muy baja), se retorna inmediatamente.

**Evaluar un conjunto de elementos**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Función evaluar\_4\_elementos(): Evalúa un conjunto de cuatro elementos en una línea para asignar una puntuación basada en la cantidad de fichas del jugador y la computadora.

Contar fichas: Cuenta las fichas del jugador ('X') y de la computadora ('O').

Asignación de puntuación: Asigna una puntuación muy alta si una de las partes tiene cuatro fichas y la otra ninguna, y puntuaciones menores para otras configuraciones.

**Funciones auxiliares para obtener filas, diagonales y posiciones abiertas**

****

**Función get\_filas():** Obtiene todas las filas del tablero actual, devolviéndolas en un formato adecuado para su evaluación.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Función get\_diagonales(): Genera todas las diagonales posibles del tablero, tanto de izquierda a derecha como de derecha a izquierda. Estas diagonales son importantes para evaluar posibles victorias diagonales.**

Texto

Descripción generada automáticamente

Función get\_abiertos(): Retorna una lista de las posiciones de las columnas que están abiertas (no están llenas).

**Algoritmo Minimax con poda alfa-beta**

**Imagen que contiene Aplicación

Descripción generada automáticamente**

Función minimax\_alpha\_beta(profundidad, alpha, beta, jugador): Implementa el algoritmo Minimax con poda alfa-beta para optimizar la búsqueda de la mejor jugada posible. Este algoritmo evalúa todas las posibles jugadas hasta una profundidad determinada y utiliza la función heurística para asignar una puntuación a cada estado del tablero.

Evaluación de fin de juego: Si se alcanza la profundidad máxima, se evalúa el tablero con `heuristico()`.

Búsqueda del mejor movimiento: Para cada jugador ('O' y 'X'), se generan todos los estados hijos posibles y se evalúan utilizando recursivamente `minimax\_alpha\_beta()`.

Poda alfa-beta: Se aplica poda alfa-beta para reducir la cantidad de estados evaluados, mejorando la eficiencia del algoritmo.

**Inicializar el juego y guardar partidas**

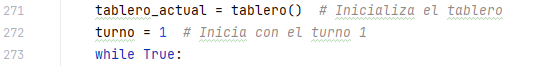
**Inicialización del Juego**

**Texto

Descripción generada automáticamente**

* Hace que tablero\_actual sea una variable global para poder ser accedida y modificada dentro de la función.
* Obtiene la fecha y hora actuales utilizando datetime.now().strftime('%Y-%m-%d %H:%M').
* Solicita los nombres del jugador 1 y del jugador 2 (o "IA" si se desea jugar contra la computadora).
* Imprime los nombres de los jugadores y la fecha y hora del juego

**Configuración del Tablero**



* Inicializa el tablero llamando a la función tablero().
* Inicia el contador de turnos en 1.
* Entra en un bucle while que continuará hasta que se gane el juego o se alcance un empate.

**Turno de Jugadores**

Texto

Descripción generada automáticamente

* Imprime el tablero actual llamando a imprimir\_tablero(tablero\_actual).
* Determina de quién es el turno basado en si turno es par o impar.
* Si es el turno del jugador 1, solicita una columna usando pedir\_columna(tablero\_actual).
* Si es el turno de la IA, calcula la mejor columna usando minimax\_alpha\_beta().

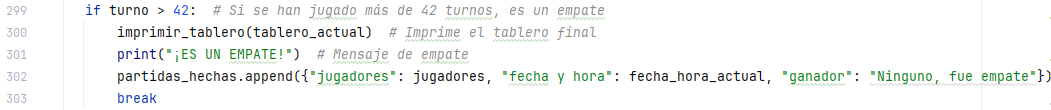
**Colocación de Piezas y Verificación de Ganador**

Texto

Descripción generada automáticamente

* Coloca la pieza en la columna seleccionada usando colocarla(columna, turno, tablero\_actual).
* Verifica si hay un ganador usando verificar\_ganador(tablero\_actual).
* Si hay un ganador, imprime el tablero final, anuncia el ganador y guarda la partida.
* Incrementa el contador de turnos

**Manejo de Movimiento Inválido y Empate**



* Si el movimiento es inválido (columna llena), imprime un mensaje de error.
* Si se han jugado más de 42 turnos, declara un empate y guarda la partida

**Función main**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Bucle Principal del Programa**

Inicio del Bucle: La función entra en un bucle `while True` que continuará ejecutándose hasta que el usuario decida salir.

Mostrar Menú: En cada iteración del bucle, se llama a la función `mostrar\_menu()` que despliega las opciones del menú principal y solicita una opción al usuario.

**Gestión de Opciones del Menú**

* Opción 1 - Iniciar Juego: Si el usuario selecciona la opción '1', se llama a la función `iniciar\_juego()` para iniciar un nuevo juego.
* Opción 2 - Ver Juegos Realizados: Si el usuario selecciona la opción '2', se llaman a las funciones `juegos\_realizados()` y `mostrar\_estadisticas()` para mostrar el historial de juegos y las estadísticas.
* Opción 3 - Salir del Programa: Si el usuario selecciona la opción '3', se imprime un mensaje de salida y se rompe el bucle, terminando la ejecución del programa.
* Opción Inválida: Si el usuario introduce una opción que no es válida, se imprime un mensaje de error.

Llamada a la Función Principal: Si la condición es verdadera, se llama a la función `main()` para iniciar el programa.

**Ejemplos de ejecución**

**Tabla

Descripción generada automáticamenteTabla

Descripción generada automáticamente**

**Conclusiones y Recomendaciones**

Nuestro proyecto "4 en línea" resultó exitoso, logrando desarrollar un juego interactivo que incluye una IA competitiva basada en el algoritmo Minimax con poda alfa-beta. La función heurística permitió a la IA evaluar el tablero y tomar decisiones estratégicas, compitiendo eficazmente contra un jugador humano. Además, se añadieron funcionalidades para guardar y cargar partidas, permitiendo llevar un registro detallado de todas las partidas jugadas.

El juego cuenta con una interfaz de usuario simple y funcional que facilita la interacción del jugador, mejorando la experiencia de juego. La implementación del manejo de turnos, la verificación de victorias y la gestión de los resultados se realizaron de manera eficiente, proporcionando un juego fluido. El proyecto también nos ofreció una valiosa oportunidad para aplicar conocimientos de programación, además de experimentar con algoritmos de inteligencia artificial.

Para mejorar el proyecto, se recomienda optimizar la función heurística para aumentar el rendimiento de la IA y considerar el desarrollo de una interfaz gráfica de usuario utilizando diferentes bibliotecas enfocadas en ello, lo que mejoraría significativamente la experiencia del usuario. También es esencial continuar realizando pruebas de depuración del código para asegurar la robustez y estabilidad del juego. Estas mejoras y optimizaciones permitirán que el juego "4 en línea" sea más atractivo y desafiante para los usuarios.