TEC | Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica

Sede Regional San Carlos

Segunda Tarea Programada

Prolog/Python – Juego Othello

Lenguajes de Programación

Profesor: Oscar Víquez Acuña

Estudiante: Justin Martínez

Fecha de entrega:

29 de abril de 2024 10:00 pm.

Síntesis del Problema

El objetivo del proyecto es implementar el juego Othello, también conocido como Reversi, en el cual dos jugadores compiten por tener más fichas de su color al final de la partida. Se requiere desarrollar un programa que permita jugar contra la computadora. La computadora debe evaluar las jugadas posibles y seleccionar la mejor opción según estrategias predefinidas. Además, se debe proporcionar una interfaz gráfica de usuario para facilitar la interacción.

Diseño - Estructuras y Soluciones Finales

Con Prolog:

Inicialización del Tablero: El predicado ***initialize_board/1*** se encarga de inicializar el tablero con el tamaño especificado. Crea una lista de listas que representa el tablero, donde cada sublista representa una fila y contiene ceros para indicar celdas vacías.

Actualización del Tablero: El predicado ***update_cell/3*** permite actualizar el valor de una celda en el tablero con un nuevo valor especificado. Utiliza nth0/3 para acceder a la fila y columna correspondientes en la lista de listas que representa el tablero, y luego utiliza ***replace/4*** para actualizar el valor de la celda.

Movimientos Posibles: El predicado ***get_possible_moves/3*** se encarga de obtener los movimientos posibles para un jugador en el tablero actual. Utiliza ***findall/3*** para encontrar todas las coordenadas ***(Row, Col)*** que representan movimientos válidos según las reglas del juego definidas en el predicado ***valid_move/4.***

Validación de Movimientos: El predicado ***valid_move/4*** determina si un movimiento es válido para un jugador en el tablero actual. Verifica si la celda está vacía, si el movimiento coloca una ficha adyacente al oponente y si hay una línea de fichas del oponente que se pueden voltear.

Direcciones de Movimiento: Se define el predicado ***move_direction/3*** para especificar las direcciones posibles en las que un jugador puede moverse en el

tablero. Cada dirección se representa como un par de deltas ***(DeltaRow, DeltaCol)*** que indican el cambio en la fila y columna, respectivamente.

Verificación de Límites del Tablero: El predicado ***inside_board/2*** se utiliza para verificar si una celda está dentro de los límites del tablero. Se asegura de que las coordenadas ***(Row, Col***) estén dentro del rango permitido para el tamaño del tablero.

Jugador Opuesto: El predicado ***opposite_player/2*** se utiliza para obtener el jugador opuesto dado un jugador actual. Esto es útil para determinar qué jugador es el oponente en la validación de movimientos.

Con Python:

Botones y Conexiones: La función ***start_game()*** se encarga de iniciar el juego cuando se presiona el botón ***"Start Game***".

El botón ***"Start Game"*** está conectado a la función ***start_game()*** mediante el atributo ***command*** de la clase ***Button***.

El botón "***Pass Turn***" permite al jugador humano pasar su turno y dejar que la computadora juegue.

El botón "***Pass Turn***" está conectado al método ***pass_turn()*** de la clase ***Othello*** mediante el atributo ***command***.

Validaciones:

- En la función ***start_game(),*** se valida que el tamaño del tablero ingresado sea un número entero y esté dentro de los tamaños válidos ***(4, 6, 8, 10).***
- En la clase ***Board***, los métodos ***is_on_board()*** y ***is_on_line()*** validan si una coordenada está dentro de los límites del tablero o en una línea del tablero, respectivamente.
- En la clase ***Othello***, el método ***is_legal_move()*** valida si un movimiento es legal verificando si la coordenada está dentro del tablero y si hay fichas del oponente que se puedan voltear en al menos una dirección.

- El método ***has_legal_move()*** verifica si hay algún movimiento legal disponible para el jugador activo.

Interacción con el Usuario:

- La clase ***Othello*** proporciona interacción con el usuario a través de la función ***play(),*** que permite al jugador humano realizar movimientos haciendo clic en el tablero.
- Después de cada turno del jugador humano, se muestra un mensaje indicando que es el turno de la computadora.
- Al final del juego, se muestra un mensaje con el resultado y se le pide al usuario que ingrese su nombre para guardar su puntuación.
- La función ***main()*** crea la interfaz de usuario utilizando la biblioteca ***Tkinter***, donde el usuario puede ingresar el tamaño del tablero y comenzar el juego.

Importaciones y Dependencias:

Se importan los siguientes módulos y paquetes:

- ***turtle*****:** Utilizado para la representación gráfica del tablero del juego.
- ***random*****:** Empleado para generar movimientos aleatorios para la computadora.
- ***pyswip.Prolog*****:** Usado para la comunicación entre Python y Prolog, permitiendo ejecutar consultas Prolog desde Python.
- ***tkinter*****:** Utilizado para crear la interfaz de usuario (GUI) donde se introduce el tamaño del tablero y se inicia el juego.
- ***messagebox de tkinter*****:** Empleado para mostrar mensajes de error o información al usuario.
- Además, se asume que el código Python depende de un archivo Prolog llamado ***prolog.pl,*** que contiene las reglas del juego de Othello implementadas en Prolog.

Instalación de Dependencias:

Se requieren las siguientes dependencias para ejecutar el código correctamente:

- ***Python (versión 3.x):*** El código está escrito en Python 3 y requiere un entorno de ejecución de Python para funcionar.
- ***PySWIP:*** Debe instalarse para permitir la comunicación entre Python y Prolog. Puede instalarse utilizando ***pip*** con el comando ***pip install pyswip.***
- ***Turtle:*** Es una biblioteca estándar de Python y no requiere instalación adicional.
- ***Tkinter:*** También es una biblioteca estándar de Python y no requiere instalación adicional.

Comparativa entre las dos formas de resolver el problema en Prolog.

Representación mediante matriz como hechos de listas de listas:

Ventajas:

- Facilidad de comprensión y visualización del tablero.
- Operaciones de acceso y actualización eficientes.

Desventajas:

- Requiere un almacenamiento adicional para representar el tablero.
- Mayor complejidad en la implementación de algunos predicados.

Representación mediante hechos "conectados":

Ventajas:

- Almacenamiento más eficiente, ya que solo se guardan las celdas ocupadas.
- Menor complejidad en la implementación de ciertos predicados.

Desventajas:

- Dificultad para visualizar el tablero en su totalidad.
- Operaciones de acceso y actualización pueden ser menos eficientes en algunos casos.

Bibliografía:

1. *Download a binary file*. (n.d.). Swi-prolog.org. Retrieved April 29, 2024, from https://www.swi-prolog.org/download/stable/bin/swipl-8.4.3-1.x64.exe.envelope
2. (*Pip install swipl?*, 2023)

   Pip install swipl? (2023, August 13). SWI-Prolog. https://swi-prolog.discourse.group/t/pip-install-swipl/6769
3. (*tkinter — Interface de Python para Tcl/Tk*, n.d.)

   tkinter — Interface de Python para Tcl/Tk. (n.d.). Python documentation. Retrieved April 29, 2024, from https://docs.python.org/es/3/library/tkinter.html
4. (*tkinter.messagebox — Tkinter message prompts*, n.d.)

   tkinter.messagebox — Tkinter message prompts. (n.d.). Python Documentation. Retrieved April 29, 2024, from https://docs.python.org/3/library/tkinter.messagebox.html
5. (*turtle — Gráficos con Turtle — documentación de Python - 3.9.18*, n.d.)

   turtle — Gráficos con Turtle — documentación de Python - 3.9.18. (n.d.). Python.org. Retrieved April 29, 2024, from https://docs.python.org/es/3.9/library/turtle.html

Nota: Se pasa una hoja pero es por la portada, además en el git dejé en el "Readme", el documento completo que se excede lo solicitado.