**Tecnológico de Costa Rica**

**Campus Local San Carlos**

**Lenguajes de programación**

**Grupo: 50**

**Profesor: Óscar Víquez**

**Steven Chacón Zárate**

**Abril 2024**

## Síntesis del problema

El problema consiste en implementar un juego de othello utilizando Prolog para el análisis de las jugadas de la computadora y otro lenguaje de programación para la interfaz gráfica de usuario (GUI). Las reglas del juego dictan que la computadora debe seleccionar la jugada que resulte en la captura de la mayor cantidad de fichas del oponente.

Se deben cumplir algunas restricciones de implementación, como la representación del tablero en Prolog de dos formas diferentes (matriz como hechos de listas de listas o hechos "conectado"), el ingreso de datos desde la GUI a Prolog utilizando ASERT, la elección de la estrategia de la máquina antes de iniciar el juego, la configuración del tamaño del tablero, la posibilidad de que un jugador pase su turno, y la indicación del ganador al final del juego.

En resumen, se busca desarrollar un juego de othello con una interfaz gráfica que permita la interacción con el usuario y que utilice Prolog para el análisis de las jugadas de la computadora, cumpliendo con las restricciones mencionadas.

## Diseño - Estructuras y Soluciones finales

**Prolog(logic.pl)**

El código se utiliza para implementar la lógica de un juego de tablero, diseñado para un juego de othello en el cual un jugador es el usuario y el otro la computadora. El propósito principal del código es determinar los movimientos válidos que un jugador puede realizar en un tablero de juego dado.

El predicado `valid\_moves` es el punto de entrada principal. Toma como entrada el jugador actual, el estado actual del tablero y devuelve una lista de movimientos válidos que el jugador puede hacer. Utiliza el predicado `empty\_space` para identificar los espacios vacíos en el tablero y luego evalúa la validez de cada posible movimiento usando el predicado `valid\_move`.

El predicado `valid\_move` determina si un movimiento es válido para el jugador en una posición específica del tablero. Utiliza una lista predefinida de direcciones posibles, definida en el predicado `directions`, para explorar las diferentes direcciones en las que un jugador puede moverse. Llama al predicado `validate\_move` para confirmar si el movimiento es legal en la dirección dada.

**El predicado `validate\_move` realiza varias verificaciones para validar un** movimiento propuesto. Primero, calcula la nueva posición después del movimiento y luego verifica si esta nueva posición está dentro de los límites del tablero usando el predicado `inside\_board`. Luego, comprueba si la casilla en la nueva posición pertenece al oponente y, finalmente, utiliza el predicado `follow\_line` para determinar si hay una línea continua de piezas del jugador en la dirección especificada.

El predicado `follow\_line` se encarga de verificar si hay una línea de piezas del mismo jugador en una dirección específica en el tablero. Realiza esto de manera recursiva, avanzando en la dirección especificada y verificando si cada casilla visitada contiene una pieza del jugador. Si se encuentra una pieza del oponente o un espacio vacío, la verificación se detiene.

Este código implementa una serie de reglas y verificaciones para determinar qué movimientos son válidos para un jugador en un juego de tablero. Cada predicado desempeña un papel específico en la validación de movimientos, asegurando que el juego siga las reglas establecidas y proporcionando una base sólida para la lógica del juego.

**Python(link.py)**

El código utiliza la biblioteca `pyswip` para conectar Python con Prolog y así poder interactuar con la lógica del juego implementada en Prolog desde Python.

La configuración inicial del código implica la importación de la clase `Prolog` de `pyswip` y la creación de una instancia de esta clase para establecer una conexión con el intérprete de Prolog. Luego, se carga el archivo "logic.pl" mediante el método `consult()`, lo que permite acceder a las reglas y consultas lógicas definidas en ese archivo.

La función `pl\_board()` convierte la representación del tablero en Python en un formato compatible con Prolog. Itera sobre las filas de la matriz del tablero y las convierte en listas Prolog. Posteriormente, une todas estas listas Prolog en una cadena de texto, creando así una representación del tablero en un formato adecuado para Prolog.

La función `optimal\_play()` recibe el jugador actual y el estado actual del tablero como entrada. Utiliza la función `pl\_board()` para convertir el estado del tablero de Python a un formato Prolog comprensible. Luego, realiza una consulta Prolog utilizando `prolog.query()`, buscando todos los movimientos válidos para el jugador actual en el estado del tablero proporcionado. Itera sobre los resultados de la consulta y extrae las jugadas válidas, añadiéndolas a una lista de movimientos. Finalmente, retorna una lista de movimientos únicos, evitando duplicados, obtenidos de las consultas Prolog.

El código facilita la comunicación entre el entorno de Python y la lógica del juego implementada en Prolog, permitiendo que el programa Python obtenga movimientos óptimos consultando las reglas definidas en Prolog para el juego de tablero específico.

**Python(othelloUI)**

El código proporcionado crea una versión del juego Othello utilizando la biblioteca Tkinter para la interfaz gráfica. Aquí tienes una explicación en prosa:

La clase `Game` maneja la lógica del juego, comenzando con la inicialización del tablero y las fichas iniciales. Cada instancia de esta clase representa una partida de Othello. Proporciona métodos para jugar movimientos, verificar si un movimiento es válido, realizar los movimientos, contar las piezas en el tablero y determinar al ganador al final del juego.

Por otro lado, la clase `OthelloGUI` se encarga de la interfaz gráfica del juego. Cuando se inicializa, crea una ventana principal y un lienzo donde se dibuja el tablero. Esta clase maneja los clics del usuario en el tablero, interpretando las coordenadas del clic y ejecutando el movimiento correspondiente en el juego. Utiliza métodos de la clase `Game` para validar y realizar los movimientos, y para actualizar la interfaz gráfica en función del estado actual del juego.

La función `main()` es el punto de entrada del programa. Inicia la ventana principal del juego y muestra un diálogo para que el usuario elija el tamaño del tablero. Después de que el usuario selecciona el tamaño del tablero, se crea una instancia de la clase `Game` con ese tamaño y otra instancia de la clase `OthelloGUI` para iniciar la interfaz gráfica del juego. Luego, entra en el bucle principal de la aplicación.

Por último, la función `get\_board\_size()` muestra un diálogo que permite al usuario seleccionar el tamaño del tablero. Devuelve el tamaño del tablero seleccionado por el usuario.

El código combina la lógica del juego Othello con una interfaz gráfica de usuario, permitiendo a los jugadores jugar de manera interactiva en una ventana de aplicación.