15 | Ajedrez

Verónica Esther Arriola Ríos

Este proyecto consiste en programar un jugador de ajedrez haciendo uso de las técncias de IA clásica.

Meta

El alumno implementará un agente inteligente capaz de jugar ajedrez contra un ser humano.

Objetivos

Que el alumno(a):implemente un agente inteligente para un juego, implemente un algoritmo de búsqueda con adversarios aplicándolo en un escenario real, transfiriendo sus conocimientos sobre *acciones*, *acciones aplicables*, *estados* y *nodos búsqueda*. diseñe su propia estructura para implementar el algoritmo, utilice un función heurística para truncar la profundidad a la que trabajará el algoritmo.

Código Auxiliar 15.1: Ajedrez

`https://github.com/computacion-ciencias/ia-ajedrez`

Desarrollo

En este paquete se te entrega:

- Un proyecto con `JavaFX` con la interfaz gráfica para mostrar un ajedrez.

Tareas a realizar

Las tareas que debes realizar son las siguientes:

Parte I: Reglas del ajedrez

1. Completa el proyecto que se te entrega. El paquete:

   `ia.ajedrez.modelo`

   ya incluye las clases para representar a las piezas del juego y sus movimientos legales, excepto por la **torre** y el **alfil**. Agrega estas clases faltantes.

2. En la clase `Juego` completa el método `jugadaEsVálida`, aquí deberás tomar en cuenta ya las posiciones de todas las piezas en el tablero. Observa que hay varios tipos de jugadas: movimiento, captura, enroque corto y enroque largo. Los caballos pueden saltar pero las otras piezas no.

   Añade métodos o lo que encuentres necesario. Recuerda que es tu proyecto. Observa que este paso corresponde a determinar si una acción es aplicable en un estado dado.

3. Completa el constructor por defecto de `Tablero` colocando todas las piezas en el estado inicial. Para ello deberás también agregar imágenes para las piezas del tablero, el programa las escala así que no tienes que preocuparte demasiado por la escala, puedes descargar las imágenes de internet o usar grafos de fuentes. Agrega las imágenes completando el método

   ```
   private void creaVistasDePiezas()
   ```

   de la clase `vistas.VistaTablero`.

4. Completa ahora el método `ejecutaJugada` en la misma clase para que los cambios sugeridos por la jugada se vean reflejados en el tablero. Recuerda almacenar qué jugada fue la que produjo el cambio en el tablero. En la segunda parte del proyecto este historial corresponderá a las *acciones aplicadas*. Así que observa bien cómo haces tu diseño, pensando en resolver la segunda parte.

5. Regresa ahora la clase `Tablero` y completa `hayJaqueMate`.

6. Finalmente prepara tu heurística: completa el método `calificación` de la clase `Tablero`. Como no será posible explorar todo el espacio de estados del ajedrez, habrá que truncar el algoritmo de búsqueda con adversarios que implementarás. Para ello, lo que harás es otorgar una calificación a cualquier tablero, para determinar *quién va ganado la partida*.

   Usa una función que pondere los valores siguientes:

a) Valores: suma puntos según el valor de cada pieza blanca en el tablero y réstalos por cada pieza negra.

b) Dominio del tablero: Por cada pieza, suma un punto por cada escaque libre al que pueda moverse dicha pieza si es blanca y resta un punto si la pieza es negra.

Devuelve un valor entero. Si uno de los jugadores está en jaque mate, devuelve el entero más grande si ganaron las bancas o el entero más pequeño si ganaron las negras.

Parte II

7. Crearás ahora a un agente para que mueva a las piezas negras. Modifica a la clase juego para que, cada vez que el turno pase a mano de las negras, se mande llamar una función que elija cómo mover la pieza y, una vez movida, regrese el turno a las blancas.

8. Para explorar el espacio de estados puede ser necesario crear copias de un estado y modificarlas en puntos estratégicos de la búsqueda. Completa el constructor copia de la clase `Tablero`. Para esto, dado que las piezas almacenan su posición, también deberás crear constructores copia para poder clonar las piezas en el tablero.

9. Implementa el algoritmo Min-max en su versión recursiva, de modo que vaya realizando una búsqueda en profundidad sobre el espacio de tableros. La mejor forma de aprovechar la memoria es no crear tantas copias del tablero, sino utilizar los objetos tipo `Jugada` para revertir las modificaciones al tablero cuando la recursividad deba regresar a un nodo búsqueda anterior. Crea copias únicamente cuando lo requieras para no perder información importante.

   Deberás definir una profundidad límite la cual tratarás como si fuera el fin del juego, aquí utilizarás la función `calificación` para asignar el puntaje al tablero.

   Mide cuál es la profundidad máxima que puede alcanzar tu algoritmo sin dejar esperando demasiado al jugador blanco.

10. Se otorgará un punto extra a quien sustituya Min-max por poda alfa-beta.