Trabajo práctico:

Estructura de datos y algorítmos

Eleccion de estructura de datos:

Para la representación del tablero use una matriz de ints, fue lo que me pareció mas eficiente, al principio había pensado en hacerlo un array de objetos (fichas), pero al final me di cuenta que la única necesidad de información en cada posición era a quien pertenecía la ficha, por lo tanto utilicé el id del jugador que pertenecia cada ficha en cada posición. El -1 representa que el lugar está vacio.
Si bien esta manera es poco expandible, es más eficiente que tener una matriz de objetos, por eso la elegí. Se podría optimizar mucho más utilizando por ejemplo una matriz de chars .

después cada jugador es una clase la cual contiene el color, los puntos, el id y si es o no un jugador controlado por computadora. Me hubiese gustado extender de la clase Player una clase que sea IAPlayer o algo por el estilo y tener ahí el comportamiento de la computadora, por ejemplo, con el algoritmo minimax.

Para las movidas también estaba entre dos opciones, guardar el tablero entero o guardar solo las piezas afectadas, al final me decidí por la segunda ya que me pareció más eficiente.

Para las jugadas que van pasando utilice un Stack, cada vez que se hacia una jugada se pushea en el stack. Esto me pareció lógico ya que para deshacer la última jugada, solo necesitamos eso, la última jugada.

Detalles del algoritmo Minimax:

Hice una implementación clásica en profundidad del algoritmo, lo único distinto es que en vez de hacer una copia del tablero, modificarla y pasarla a la siguiente iteración lo que hacia era aplicar una jugada y cuando terminaba la iteración la hacia inversa. Esto me pareció más eficiente que copiar un tablero entero y pasarlo, ya que las jugadas suelen ser de pocas fichas y solamente es cambiar un numero en la matriz.
Para el caso de tiempo use iterative deepening, que llama al algoritmo con distintas

profundidades hasta que se acaba el tiempo, me hubiese gustado implementar algún tipo de transpoting table para que sea mas eficiente pero no me dio el tiempo.

Heuristica:

Para la heurística utilice la cantidad de jugadas disponibles + el numero de piezas del jugador en ese momento. La primera la utilice porque me parece que es más probable ganar cuando tenés más posibilidades de mover (ya que tenés más abertura de mesa), y la segunda la utilice porque el ganador se define por quién tiene más fichas. Me gustaría haber probado con otras heuristicas, como por ejemplo, sumando la cantidad de fichas que un jugador tiene fijas (fichas que ya no se pueden comer).

Métricas de tiempo:

Lo siguiente fue probado en computadora contra computadora, es el tiempo en nano segundos que tardaron en terminar de llenar un tablero de 8x8.

parámetros:

poda: on

profundidad:5 tiempo en terminar tablero : 11.6 segundos

profundidad 6 tiempo en terminar tablero: 50.7 segundos

poda: off

profundidad 4 tiempo en terminar tablero: 11.3 segundos

profundidad 5 tiempo en terminar tablero: 161.3 segundos

Se puede observar bien la diferencia entre cuando hay poda y no.

Errores conocidos:

El dot tree se genera mal, creo que estoy haciendo bien el recorrido pero me está tomando los nodos que son de una jugada con las mismas coordenadas (pero distintas jugadas) como iguales, entonces se unen.

Esto no es un error pero me gustaría haberlo cambiado, la computadora hace la jugada en el momento que el usuario hace la suya, por lo tanto no se actualizan los colores hasta que se termina la jugada de al computadora, por lo que se hacen los cambios muy rápido y no se puede ver la jugada del jugador que no es la computadora.

Pude hacer que el tablero sea resizable pero no la parte de los comandos.

Pude compilarlo y andarlo en Windows, pero en la notebook que tengo Kali Linux no lo pude hacer funcionar (si compilaba, pero cuando ejecutaba el jar me tiraba error), trate de arreglarlo pero no hubo caso. Le pedí a un amigo que pruebe en su Windows y en su Ubuntu y le funciono perfecto, no se si será un problema de Kali en particular con javafx o algo por el estilo.