Trabajo Práctico Especial

Estructuras de Datos y Algoritmos

Primer Cuatrimestre 2017

Objetivo

El objetivo del trabajo práctico es implementar el algoritmo **minimax** para el juego Go.

Requerimientos

Descripción del juego

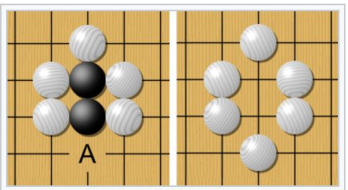
Reglas generales

Go es un juego milenario de 2 jugadores en el que se van ubicando fichas blancas y negras por turnos (un color para cada jugador) en un tablero de NxN. Para fines prácticos vamos a tomar un tablero de 13x13

En cada turno, el jugador debe ubicar una ficha de su color en una intersección vacía. Una vez puesta una pieza no puede moverse.

El objetivo del juego es hacer que tu "territorio" sea más grande que el del oponente. Un territorio es dominado por un jugador cuando está rodeado por piezas de este jugador
Una forma de hacer territorio es capturando fichas del oponente. Para capturar fichas de un oponente se las debe rodear en todas las direcciones.
Las fichas que están rodeadas se sacan del tablero y se guardan como prisioneras. Cada ficha prisionera vale un punto.

Ejemplo:

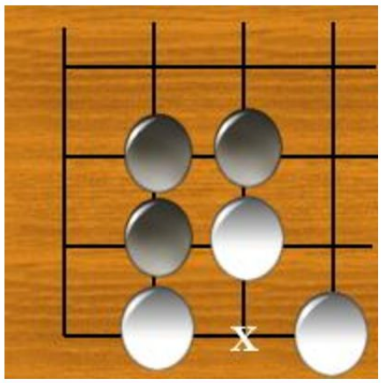


If White plays at A, the black chain loses its last liberty. It is captured and removed from the board.

Regla del suicidio

No puede ponerse fichas en algún lugar que signifique el suicidio automático de esa ficha.

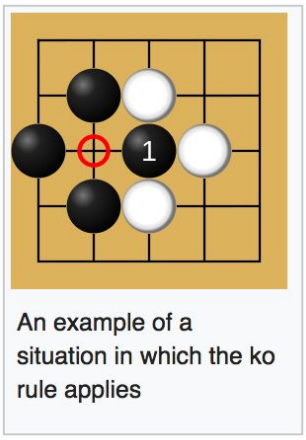
Ejemplo:



Regla Ko

No pueden ponerse fichas que se coman simultáneamente de forma cíclica infinita (un jugador no puede poner una ficha que haga que el tablero quede igual a como estaba al final de su último turno).
Ese jugador debe poner la ficha en cualquier otro lugar.

Ejemplo:



An example of a situation in which the ko rule applies

Fin del juego

El juego termina cuando ningún jugador quiere mover.

Gana el jugador que más puntos tiene. Los puntos se cuentan

- 1 por cada espacio capturado
- 1 por cada prisionero

Tutoriales

http://playgo.to/iwtg/en/
https://www.youtube.com/watch?v=JWdgqV-8yVg
https://en.wikipedia.org/wiki/Go_(game)

Implementación

Se pide implementar una aplicación en Java que pueda jugar a este juego, utilizando el algoritmo **minimax** como estrategia. La aplicación debe poder utilizarse de dos maneras distintas:

- A través de una interfaz gráfica que permita al usuario jugar una partida contra el programa.
- Por consola, leyendo de un archivo el estado actual del tablero e imprimiendo por salida estándar el movimiento a realizar.

En ambos casos se debe poder especificar el comportamiento del algoritmo en los siguientes aspectos:

- Limitar la profundidad máxima del árbol que se explora o bien indicar el tiempo máximo para calcular el movimiento a realizar.
- Habilitar opcionalmente la poda alfa-beta.
- Generar un archivo en formato **dot** con el árbol que fue explorado por el algoritmo.

Formato de los archivos de entrada

Cuando se utiliza la aplicación por consola, se toma como entrada un archivo de texto que contiene el estado del tablero. Este archivo contiene una matriz de caracteres de la dimensión del tablero, en donde un espacio en blanco indica una celda vacía, el carácter '1' indica una ficha del jugador 1, y el carácter '2' indica una ficha del jugador 2. El siguiente es un ejemplo de un posible archivo de tablero:

```
   1
   1
111121
   222
  21
```

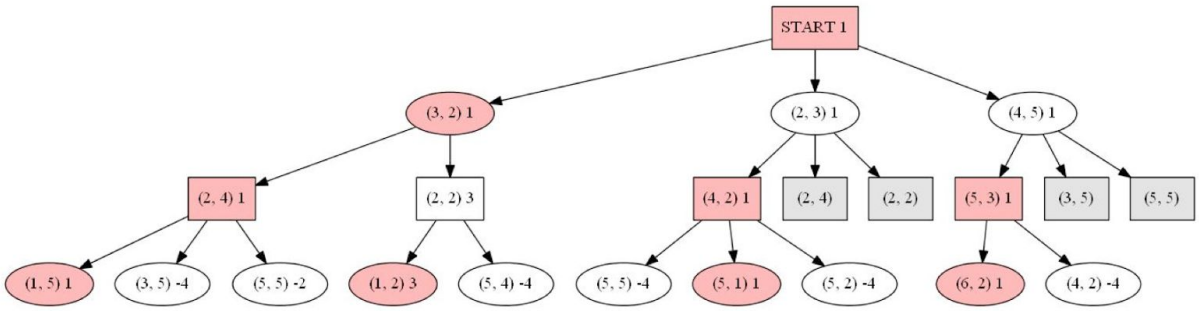
Árbol en formato dot

Cuando se habilite la generación del árbol, se debe crear un archivo llamado **tree.dot**, que contenga una representación del árbol de estados explorados por el algoritmo. Debe tener las siguientes características:

- En los nodos se debe indicar la jugada representada por el nodo en formato "(fila, columna)" junto con el valor heurístico de dicho nodo. Por ejemplo "(3, 2) -8".

- Los nodos correspondientes al jugador 1 deben tener una forma distinta a los del jugador 2, para poder distinguirlos fácilmente de manera visual.
- De los hijos de cada nodo, se debe resaltar aquel que es elegido por el algoritmo como el mejor (es decir, del cual se toma el valor heurístico para el padre).
- Cuando se aplique poda alfa-beta, se deben resaltar con un color de fondo distinto los nodos en los que se aplicó la poda (es decir, aquellos estados que no fueron explorados). En este caso, el texto de estos nodos no incluye el valor heurístico, sólo se muestra la fila y la columna de la jugada.
- En caso de limitar por tiempo la ejecución del algoritmo, el árbol que se debe generar es el que produjo la respuesta que se retorna.

A continuación se muestra un ejemplo simplificado de un posible árbol generado por el programa:



Sintaxis de la línea de comandos

Para ejecutar el programa, se debe respetar la siguiente sintaxis en la línea de comandos (los paréntesis indican opciones de las cuales se debe elegir una sí o sí, y los corchetes indican parámetros optativos):

java -jar tpe.jar (-visual | -file archivo -player n) (-maxtime n | -depth n) [-prune] [-tree]

Parámetro	**Descripción**
`-visual`	Ejecuta la aplicación en modo visual, permitiendo al usuario jugar una partida completa contra la computadora. Se muestra una ventana con el estado inicial del tablero, y comienza a jugar el usuario. Una vez que realiza una jugada, la computadora responde, y así continúan jugando alternadamente hasta que no se puedan realizar más jugadas, y se informa quién ganó.
`-file`	Ejecuta la aplicación para realizar una sola jugada. Lee de un archivo el estado del tablero, analiza la mejor jugada que se puede realizar, e imprime por

	consola dicha jugada en formato “fila, columna”. Si no se puede realizar ninguna jugada, imprime por consola el texto “PASS”.
`-player n`	Utilizado junto con el parámetro file, para indicar qué jugador es el que debe realizar la jugada. El valor de n puede ser 1 o 2.
`-maxtime n`	Indica el tiempo máximo para obtener una solución, expresado en segundos. Pasado este tiempo, el algoritmo debe retornar con la mejor jugada analizada hasta el momento. Si se utiliza con la opción “-visual”, es el tiempo máximo a utilizar por cada jugada de la computadora.
`-depth n`	Indica la profundidad del árbol que se desea explorar. Si se utiliza con la opción “-visual”, es la profundidad máxima a analizar en cada jugada de la computadora.
`-prune`	Habilita la poda alfa-beta en el algoritmo.
`-tree`	Genera un archivo llamado tree.dot con una representación del árbol explorado, luego de aplicar el algoritmo minimax. Sólo se puede utilizar con el parámetro –file.

Ejemplos de uso:

```
# java -jar tpe.jar -visual -maxtime 4 -prune
```

Ejecuta la aplicación en modo visual, tardando como máximo 4 segundos en responder cada vez que sea el turno del programa para jugar, y aplicando la poda alfa-beta.

```
# java -jar tpe.jar -visual -depth 5
```

Ejecuta la aplicación en modo visual, explorando el árbol de estados hasta 5 niveles de profundidad, sin aplicar ningún tipo de poda.

```
# java -jar tpe.jar -file tablero.txt -player 1 -maxtime 10 -prune -tree
```

Ejecuta la aplicación en modo consola, leyendo el estado del tablero del archivo tablero.txt. Analiza el mejor movimiento posible para el jugador 1, tardando como máximo 10 segundos, aplicando poda alfa-beta y creando el archivo tree.dot con la representación del árbol explorado para obtener la jugada a realizar. Imprime por consola dicha jugada. Para el ejemplo del archivo de tablero que se muestra en la sección anterior, una salida posible sería “5, 1”.

```
# java -jar tpe.jar -file tablero.txt -player 2 -depth 6 -prune -tree
```

Ejecuta la aplicación en modo consola, leyendo el estado del tablero del archivo tablero.txt. Analiza el mejor movimiento posible para el jugador 2, explorando el árbol hasta 6 niveles de profundidad, aplicando poda alfa-beta y creando el archivo tree.dot con la representación del árbol explorado. Imprime por consola dicha jugada. Para el ejemplo del archivo de tablero que se muestra en la sección anterior, una salida posible sería “6, 3”.

Entrega

El trabajo se realizará en grupos de hasta 5 integrantes.
Se contará con 3 entregables:

- Código fuente
- Informe
- Video

La fecha límite de entrega es el **martes 19 de junio a las 23:59 hs.**
Se cuenta con una opción de entrega tardía el **viernes 26 de junio a las 23:59 hs** con una penalidad de 2 puntos.

Código

La entrega del código fuente se deberá realizar a través del repositorio Git provisto por la cátedra. Cada grupo deberá enviar un mail a la cátedra indicando el número de revisión a considerar.

El código entregado debe contener un *buildfile* de Ant, cuyo target default genere un archivo **jar** con la aplicación desarrollada.

Informe

Cada grupo deberá enviar un **informe** que explique en forma clara:

- Algoritmos y estructuras principales utilizadas
- Problemas encontrados durante el desarrollo y decisiones tomadas
- Tablas de comparación de tiempos y resultados
- Conclusiones

Video

Y un **video** que ilustre todas las funcionalidades del programa (modo visual y no visual, con poda y sin poda, etc), demostrando ejemplos claros que sean consistentes con lo presentado en el informe