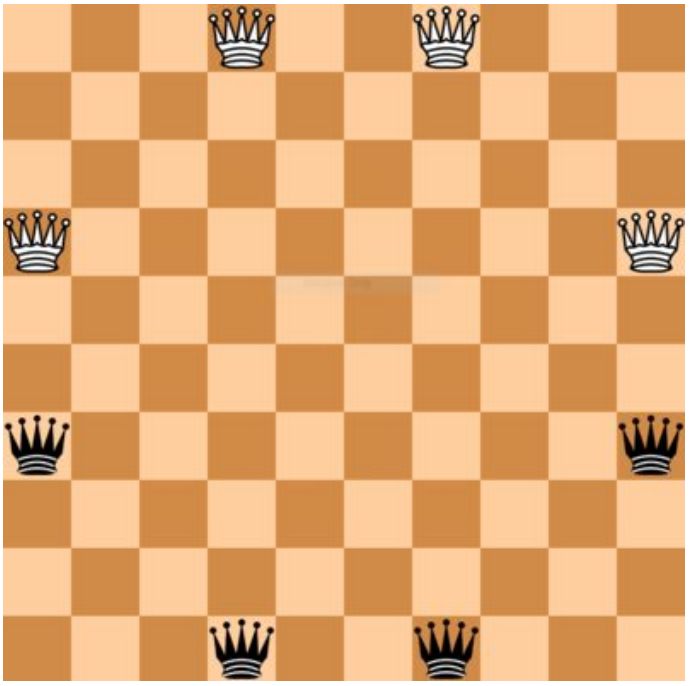


The game of Amazons

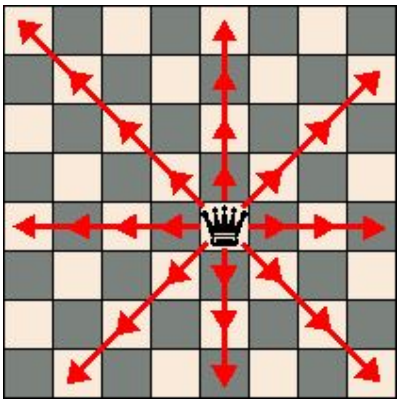
Projecte de Programació 20-21

Índice

Introducción

El objetivo del proyecto es desarrollar un jugador que siguiendo técnicas de IA, compita en el **juego de las amazonas** y resulte ganador, en tres tipos de tablero y con dos versiones de juego, con profundidad fija y con profundidad iterada.
El juego de las amazonas fue inventado por Walter Zamkauskas en el año 1988, se trata de un juego de complejidad pero con unas reglas muy simples.
Es un juego de dos jugadores, los cuales alternan sus turnos, eligiendo una de las 4 amazonas de las que dispone cada jugador (blancas o negras), para moverse por un tablero de 10x10.

En cada turno, cada jugador realiza **dos** movimientos:
1. Mueve una de sus amazonas con movimientos disponibles, hacia una posición libre, siguiendo el mismo movimiento que la reina en el ajedrez



2. Lanza una flecha a cualquier casilla vacía del tablero

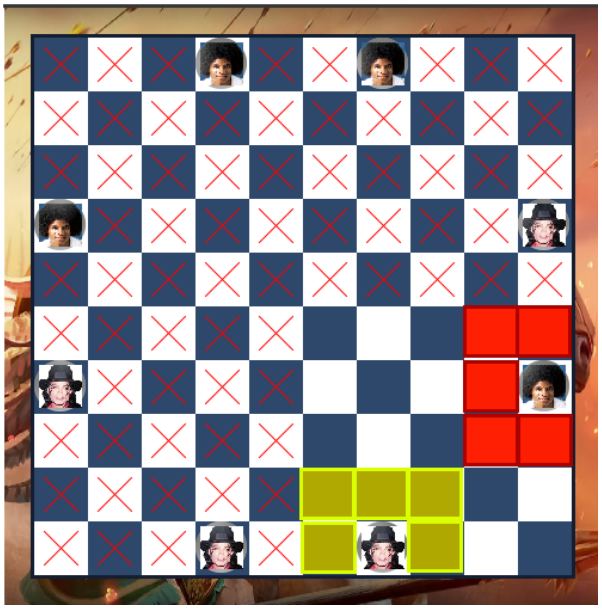
El primer jugador en quedar acorralado y no poder moverse, pierde.
En este juego no se puede empatar, es un juego territorial.

Evaluación de heurísticas

Durante el transcurso del proyecto, hemos valorado diferentes maneras de implementar una buena heurística, a continuación detallaremos cada una de ellas con un nombre descriptivo.

Celdas libres:

Inicialmente valoramos la opción de realizar una heurística básica para poder empezar a testear el juego, y de hecho, llegamos a implementarla.
Consiste en algo sencillo, pero que dio buenos resultados pese a no explorar muchos nodos. Por cada amazona aliada, la idea era realizar una comprobación de las casillas libres que tenía a su alrededor. haciendo un sumatorio de todas estas.
Posteriormente, lo mismo para cada amazona enemiga, realizando un sumatorio también. Finalmente la función devuelve la diferencia entre el sumatorio de las casillas libres enemigas, y las aliadas. Era un método rápido, ya que no exploraba todo el tablero, sólo las casillas adyacentes a cada una de las amazonas del tablero.



En la imagen podemos ver que ambos jugadores, tienen 5 celdas libres adyacentes inicialmente

g(n) = casillas libres alrededor de las amazonas alidas
h(n) = casillas libres alrededor de las amazonas enemigas

$$f(h) = g(n) - h(n)$$

Movilidad:

La segunda opción valorada fue el conteo del número de movimientos que podía ejecutar cada jugador y calcular la diferencia de nuevo (con el parámetro **restricted = false,** para devolver todos los movimientos posibles, durante todo el juego se utilizará en este estado). Siendo una opción con mejores resultados, resultó válida, sin embargo, creimos que aún podría mejorar.

En la imagen podemos los movimientos del jugador 1 (amarillos)
y los movimientos del jugador 2 (rojos) y algunos movimientos que comparten ambos.

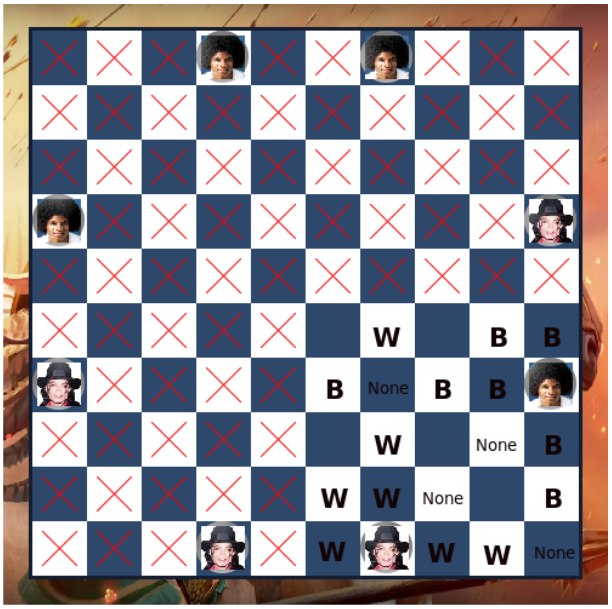
```
g(n) = posibles movimientos de las amazonas alidas
h(n) = posibles movimientos de las amazonas enemigas

                    f(h) = g(n) - h(n)
```

Territorio:

La tercera opción valorada fue realizar un contador territorial, donde cada jugador se asigna las celdas a las que puede llegar en un movimiento, con un booleano. Después, revisa desde el destino de cada movimiento, los segundos movimientos a los que puede llegar, y marca las casillas sumando un 5 a un contador propio en dicha casilla.
Esto último se añade para que, en caso de que una casilla coincida y pertenezca a un segundo movimiento de los dos jugadores, la casilla pertenezca al jugador que más movimientos involucran a dicha casilla.
Es decir, si el jugador 1 no llega a la casilla X en un movimiento, y el jugador 2 tampoco, pero el jugador 1 llega a la casilla X desde un segundo movimiento 3 veces, y el jugador 2 llega también pero sólo 2 veces, esa casilla pertenece al jugador 1.

También indicar, que si ambos jugadores llegan una casilla en un sólo movimiento, esta no tendrá propietario, así como si ambos jugadores llegan a una casilla en el segundo movimiento y los contadores empatan, tampoco pertenecerá a ninguno.



En caso de un primer movimiento, el escenario sería el de la imagen anterior.
Siendo W=White, B=Black y None para las casillas sin propietario.

En caso de que las posiciones actuales pertenecieran a un segundo movimiento, cada jugador marcaría las casillas a las que puede llegar con un 5 y el resto por defecto quedarían marcadas con un 10 (no se ha indicado en la imagen por falta de espacio), pero al valer lo mismo para cada jugador, la casilla no pertenecería a ninguno.

```
g(n) = casillas que son propiedad de las amazonas aliadas
h(n) = casillas que son propiedad de las amazonas enemigas

                    f(h) = g(n) - h(n)
```

Fusión movilidad y territorio:

Es la opción utilizada finalmente, una combinación de los dos últimos puntos explicados.
Realizamos un recorrido por todo el tablero dando un propietario a cada casilla libre, la propiedad se la dará al jugador que llegue en menos movimientos a dicha casilla.
Al final se harán dos sumatorios, el primero será de las casillas que tienen en propiedad el Jugador A y el Jugador B. Se hará una suma de +4 a cada jugador por cada casilla que tenga en propiedad.
El segundo será de la suma total de los posibles movimientos que tiene cada jugador, pero en este caso solo aumentará en 1 por cada movimiento posible de cada jugador.

Siendo la función heurística:

```
g(n) = (número de casillas en propiedad del jugador A)*4 + posibles movimientos del jugador A

h(n) = (número de casillas en propiedad del jugador B)*4 + posibles movimientos del jugador A

f(n) = g(n) + h(n)
```

Implementación de la heurística utilizada:

Guardamos en una lista de puntos las posiciones de las amazonas aliadas y en otra la posición de las amazonas enemigas.
Generamos una matriz de 10x10 de la clase Casilla para indicar la propiedad de cada casilla.

Por cada amazona aliada dado el punto en el tablero donde se encuentra, podemos ver todos los posibles movimientos de esta utilizando la función getAmazonsMoves() de la clase GameStatus, estas casillas son las que podemos alcanzar en un solo movimiento. Ahora que sabemos que casillas podemos alcanzar en un movimiento, utilizamos la función buscarJugadas() para averiguar qué casillas podemos alcanzar, así averiguamos las casillas que se pueden alcanzar en dos movimientos.

Todas las demás casillas a las que no se puede llegar en 1 o 2 movimientos, las marcamos como que llegan en 3 o más, y por tanto, no tienen propietario.

A continuación, hacemos lo mismo que el punto anterior para las amazonas enemigas, marcando en la matriz de 10x10.

En este punto las casillas que hayan sido marcadas de ambos jugadores como que llegan en un solo movimiento se pondrán en “None”, indicando que no tienen dueño y que en el contador o sumatorio de casillas con dueño no se sumarán.

Al final hacemos un recorrido de la matriz de 10x10 de clase Casilla, para averiguar el propietario de la casilla, haciendo dos contadores, uno que cuente las casillas que tiene en propiedad el aliado y otra para el enemigo. Con esto obtenemos el número de casillas en propiedad de cada jugador.

A cada casilla en propiedad de cada jugador le asignamos un valor de 4 y a cada posible movimiento de cada jugador se le asigna un valor de 1.

Siendo así que la territorialidad tenga más peso que un posible movimiento, con esto nuestra heurística tomará en cuenta ambos valores pero dando prioridad a la territorialidad.

Estrategias

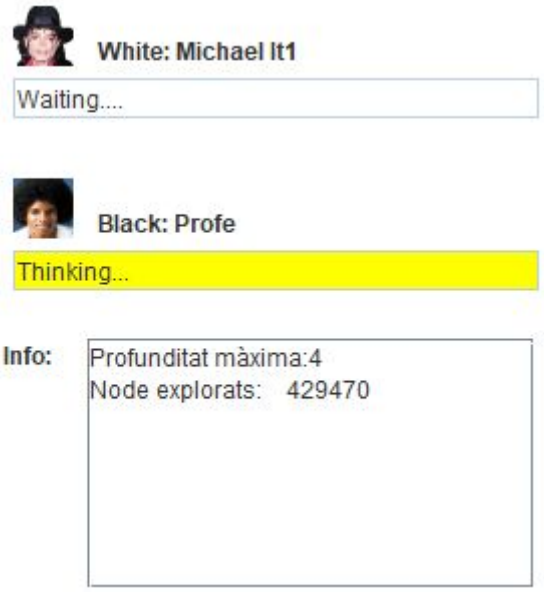
Se han utilizado dos estrategias para tirar flechas, en base al número de celdas vacías que queden en el tablero, el cual podemos obtenerlo gracias al método getEmptyCellsCount(). Para el movimiento, siempre se utilizará la heurística explicada anteriormente.
1- Si el número de casillas vacías es superior al cuarto de tablero (es decir, mayor a 23, 25 casillas y dos jugadores), utilizaremos una estrategia agresiva, dónde, gracias al método buscarMejorTiro(), detallado en el JavaDoc, se tirará una flecha a la mejor posición adyacente libre del enemigo.
2- En cambio, si el número de casillas es reducido (igual o inferior al cuarto de tablero), se revisará todo el tablero y será la función min_max() quién escogerá la mejor opción para tirar una flecha.

Este procedimiento es el que se ha seguido para las dos versiones del jugador.

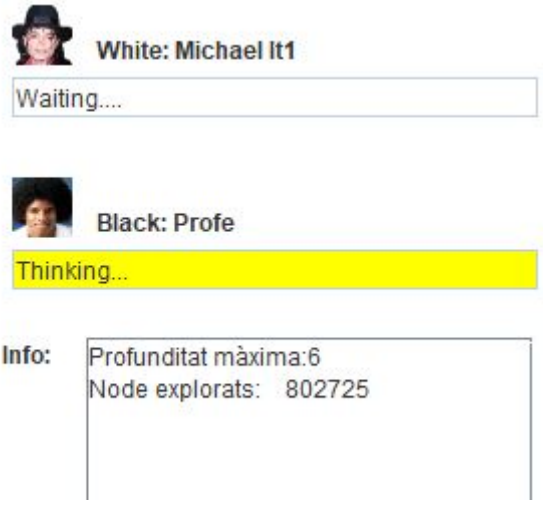
Con esta estrategia conseguimos aumentar la profundidad con el tablero completo y con medio tablero. También nos da más ventaja porque empezamos siendo muy agresivos tirando las flechas cerca de las reinas enemigas intentando encerrarlas en lugar de dejar que la heurística la elija y ponga las flechas aleadas de ellas porque no llega a explorar todas las posibles jugadas.

Estadísticas poda Alpha-Beta

- Profundidad iterada, sin poda Alpha-Beta con timeout de 5 segundos en HALFBOARD:

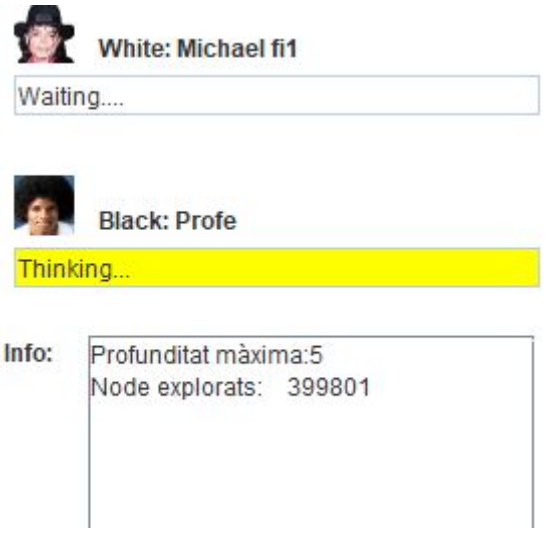


- Profundidad iterada, con poda Alpha-Beta con timeout de 5 segundos en HALFBOARD:



Al tener la poda Alpha-Beta, observamos que puede bajar a mayor profundidad por que deja de explorar ramas que no le sirven. Al contrario que sin ella, dónde tiene que explorar todas las ramas.

- Profundidad fija de 5, sin poda Alpha-Beta con timeout de 5 segundos en HALFBOARD:



- Profundidad fija de 5, con poda Alpha-Beta con timeout de 5 segundos en HALFBOARD:

En el caso de la versión fija, podemos observar que sin la poda Alpha-Beta, explora más nodos y tarda más tiempo en dar un buen movimiento, al contrario que con la poda, dónde explorará menos nodos y devolverá el movimiento en menos tiempo.

Implicación del equipo

Nombre	Código	Documentación	Git
Héctor	50%	50%	50%
César	50%	50%	50%

Esta tabla refleja el trabajo que creemos haber hecho en conjunto, cabe destacar que el proyecto lo trabajamos siempre juntos y no cada uno por su parte y luego juntado. Por eso creemos que los porcentajes de trabajo hecho de la tabla son los más adecuados.

Enlace al proyecto en GitHub

https://github.com/hmonpa/AmazonsTheGame

Conclusiones

Haciendo todas las pruebas posibles con el la versión iterada y la versión con profundidad fija, tenemos una conclusión para cada versión:

Profundidad fija:

Este jugador generalmente gana casi siempre con una profundidad de 3, después de hacer decenas de pruebas, parece que en esta profundidad es cuando llega a elegir los mejores movimientos para ganar al jugador Carlinhos, si vamos aumentando la profundidad baja la probabilidad de victoria.
Tanto en cuarto de tablero como en medio tablero, tiene un porcentaje de victoria muy alto y en tablero completo, prácticamente no pierde nunca.

Profundidad iterada:

Cuanto más grande es el tablero, más puede bajar, llegando a bajar a los mismos o más niveles que Carlinhos, lo cual nos brinda una ventaja al inicio de la partida. Esto es debido a la estrategia utilizada cuando hay más de un cuarto de tablero con celdas libres.
En tablero medio dependiendo de cómo de agresivas sean las jugadas de nuestro jugador, podría darnos ventaja o desventaja ya que el tiro de flecha no lo elige nuestra heurística, si no la función implementada para ello.

Con esto podemos concluir que nuestro jugador tiene una probabilidad moderada de victoria en un cuarto de tablero, y tiene una probabilidad alta en cuanto más grande es el tablero.

Bibliografía:

Información adicional donde buscamos información de las posibles heurísticas.

[An evaluation function for the game of amazons]

[Efficient Implementation of Game Trees]

[A Knowledge-based Approach of the Game of Amazons]