Análisis y diseño de algoritmos II

Proyecto de cursada

Amazonas

Integrantes: Chaju Yamil, Hoses Pedro.

E-mails: yamilchaju@gmail.com, pedrohoses@gmail.com

N° grupo: 16.

Índice

Introducción

En el siguiente informe se describe como se desarrolló el juego propuesto por la catedra, llamado amazonas, el cual consiste en un juego de estrategia de dos jugadores (dos personas o una persona vs el PC) donde cada jugador tiene 4 reinas sobre un único tablero de 10x10. Los movimientos permitidos son los mismos que las reinas de ajedrez con la diferencia de que la reina que se ha movido lanza una flecha desde la casilla de destino en dirección diagonal u ortogonal, a otra casilla. Las reinas no pueden moverse a las casillas alcanzadas por una flecha u ocupadas por otra reina, moverse a través de ellas, lanzar flechas a esas casillas, o lanzar flechas a través de ellas. El primer jugador en no poder hacer un movimiento pierde la partida.

Nuestra tarea es implementar un algoritmo adecuado para simular el comportamiento de un jugador inteligente (PC), evaluando cada movimiento posible con distintas funciones y así poder hacer el movimiento más beneficioso.

Métodos de la clase.

Esta clase cuenta con 8 métodos:

- Tablero (constructor básico): Este método crea el tablero de 10x10 en el cual coloca 8 fichas (4 reinas azules y 4 rojas) y guardas sus posiciones en un arreglo que se va a ir actualizando durante el juego.
- ~Tablero (destructor): Este método no lo implementamos ya que no borramos el tablero en ningún momento, por lo tanto, no lo creímos necesario.
- MoverFicha (modificador): Este método recibe por parámetros origen y destino los cuales, si el movimiento es válido, se usan para mover la ficha, luego se actualizan los arreglos con los nuevos datos.
- Color(observador): Este método recibe como parámetro una posición y devuelve el color de la ficha.
- Disparo (modificador): Este método recibe como parámetro el destino al cual se desea dispara la flecha y deja una cruz en el lugar.
- PonerFicha (modificador): Este método recibe como parámetro la posición en donde se coloca una ficha y su color.
- GameOver (observador): Este método recibe el color del jugador que desea verificar y devuelve un booleano informando si ese jugador tiene o no movimientos posibles.
- DameFicha (observador): Este método recibe el índice de una reina guardada en el arreglo y devuelve la posición actual de la misma.
- QuitarDisparo(modificador): Este método recibe la posición del disparo, lo borra y en su lugar lo ocupa con un guion “- “.

Implementación.

Utilizamos dos evaluaciones heurísticas.

Primer evaluación heurística:

```
int ValuacionPC(Tablero actual,char jugador)
{
   int valor=0;
   for (int i=0; i<4;i++)
   {
      char aux1;
      int aux2;
      actual.DameFicha(aux1,aux2,jugador,i);
      valor=valor+PuntuoCasillero(actual,aux1,aux2);
   }
   return valor;
}
```

La función recorre con un “for” la posición de cada reina del jugador actual y devuelve un valor correspondiente a cada tablero posible, utilizando la función PuntuoCasillero que devuelve la cantidad de movimientos posibles de la reina.

Segunda evaluación heurística:

```
int ValuacionPCP(Tablero actual,char jugador)
{
   int valor=0;
   for (int i=0; i<4;i++)
   {
      char aux1;
      int aux2;
      actual.DameFicha(aux1,aux2,jugador,i);
      valor=valor+PuntuoCasilleroProximidad(actual,aux1,aux2);
   }
   return valor;
}
```

La función recorre con un “for” la posición de cada reina del jugador actual y devuelve un valor correspondiente a cada tablero posible, utilizando la función PuntuoCasilleroProximidad que devuelve una puntuación evaluando los casilleros de dos a la redonda de la posición de la reina, si un casillero esta vacío suma un punto, si hay una cruz resta un punto y si hay otra reina (aliada o enemiga) resta dos puntos.

Función Negamax:

Este algoritmo funciona igual que MINIMAX con la diferencia que en vez de minimizar los valores de un jugador y maximizar los del otro, combina estas dos funciones en una sola, a partir de evaluar los nodos desde el punto de vista del jugador que va a recibir el turno.

La modificación que plantea el algoritmo Negamax es multiplicar los valores de los nodos por -1 en niveles alternados del espacio de búsqueda, para cambiar el punto de vista del jugador y así poder maximizar siempre los valores obtenidos. Esto es posible gracias a la propiedad matemática max(a,b) = -min(-a,-b).

Poda Alpha-Beta:

La poda Alpha-Beta se basa en la idea que, para obtener el valor Minimax del nodo raíz del espacio de búsqueda, no es necesario examinar todos los nodos de la frontera. Esta poda es una mejora muy importante que se le puede realizar al algoritmo Minimax para reducir notablemente el número de nodos que deberán ser examinados durante la búsqueda de una jugada posible.
El algoritmo Alpha-Beta mantiene dos valores adicionales (alpha y beta) que representan el intervalo de mejores valores que los jugadores pueden obtener durante la búsqueda. Entonces, se establece alpha (α) como el mejor valor posible y beta (β) como el peor valor posible. Si el algoritmo encuentra un nodo cuyo valor se encuentra por fuera del intervalo Alpha-Beta, dicho nodo y su sub-árbol puede ser podado, ya que se sabe con certeza que el valor encontrado no actualizará el valor del nodo raíz.

Heurísticas:

Valuación por movimientos:

Nuestra idea al pensar esta heurística consistía en puntuar el tablero según los movimientos que puede realizar el jugador actual menos los movimientos que puede realizar el enemigo. Siguiendo esta lógica uno espera que la maquina busque dejar sus reinas en la posición con mayor cantidad de movimientos y con el disparo bloquear las del otro jugador.

En un principio cuando llamábamos a la valuación del tablero solo evaluamos los movimientos que podía realizar el jugador actual (sin restar los del enemigo). Esto llevo a que la maquina solo buscara dejar sus reinas en lugares con muchos movimientos y disparaba a lugares que no le tapaban movimientos a ella, lo cual implicaba que no le tapaba movimientos al enemigo. Modificamos esta función para que reste los movimientos del jugador contrario y de esta forma logramos que la maquina no solo se mueva a un lugar favorable, sino que también tape los movimientos del enemigo.

Valuación por proximidad:

En este caso buscábamos evaluar la proximidad de cada reina de modo tal que si una reina estaba rodeada por cruces u otras reinas (aliadas o enemigas) buscara moverse a un casillero cuyo entorno este más libre. El problema que se nos presento fue que no ejecutaba los disparos más convenientes para la máquina y en ocasiones si el jugador la encerraba en un rango mayor a dos, la función no iba a evaluar que se estaba encerrando. Siguiendo un método parecido a la heurística anterior decidimos restarle la valuación del jugador enemigo consiguiendo así que realizara disparos más favorables para la máquina.

Resultados

Cuando comenzamos a realizar pruebas con la primera evaluación heurística, en profundidad 1, descubrimos que los movimientos realizados por la computadora no eran malos, simplemente no predecía que movimiento iba a hacer el jugador, por lo tanto, ganarle en esa profundidad resulto sencillo. Luego de esto comenzamos a realizar pruebas en profundidad 2, ya en esta instancia ganarle a la computadora fue difícil dado que los movimientos que realiza predicen nuestro mejor movimiento. Como era de esperarse cuando comenzamos a realizar pruebas en profundidad 3 nos fue casi imposible ganarle, de hecho, todavía no le pudimos ganar.

Esta heurística en los primeros movimientos obtiene un valor de tablero optimo con las primeras reinas optimizando a través de la poda Alpha-Beta una gran cantidad de tableros donde sus valuaciones no están dentro del rango de esa función (toma un valor muy grande al principio y en base a ese realiza la poda).

En cuanto a la calidad de la solución, la heurística que elegimos siempre busca moverse a lugares liberados y disparar en los casilleros más eficientes.

Analizando la segunda heurística los resultados no fueron tan agradables. En los primeros movimientos solo buscaba sacar la reina de los bordes del tablero, aun si eso implicaba poner sus reinas cerca de una cruz. Si se encontraba con dos reinas adyacentes no buscaba separarlas porque la valuación del tablero era similar para ambas reinas. Cuando realizamos pruebas en profundidad 2 la situación no cambio mucho salvo que ahora los disparos buscaban perjudicar más al jugador contrario, y finalmente en profundidad 3 se nos presentó un nuevo problema, el tiempo de cálculo del siguiente movimiento era constante. Luego de los análisis consideramos que esta heurística no es ideal ya que:

1- El tiempo de búsqueda se mantiene constante ya que siempre debe revisar todo el entorno de las 8 reinas.
2- No genera cambios significativos en el espacio de búsqueda.
3- Los movimientos obtenidos en ocasiones provocan que las reinas se encierren.

	Heurística 1	Heurística 2
Tiempo de respuesta	Mejora con el paso de los turnos, pero empeora cuanta más profundidad exiges	Se mantiene constante todo el juego
Primer piso	Busca principalmente mover sus reinas de zonas comprometidas a zonas más liberadas. No hay poda	Prioriza llevar sus reinas a zonas más liberadas. No hay poda
Segundo piso	Libera sus reinas y busca tapar movimientos al jugador. Realiza poda	Lleva sus reinas a zonas liberadas y dispara en las cercanías de las fichas del oponente. Poca poda.
Tercer piso	Mejora los resultados del piso anterior, no solo la calidad del movimiento sino también la cantidad de poda	No presenta mejorías, empeora el tiempo de búsqueda.
Calidad del resultado	Según nuestras pruebas la calidad es alta incluso en piso 2.	A nuestro parecer baja calidad.

Claramente, si debemos elegir entre las heurísticas realizadas, consideramos que la primera de ellas es la mejor, no solo por la alta calidad del resultado sino también por las mejorías que presenta conforme se desarrolla el juego: a medida que avanzan los turnos la respuesta es más rápida y la calidad del resultado es mayor (por la limitación del espacio).

Si bien la segunda heurística no es una mala idea en teoría, los resultados no fueron los esperados y el hecho de que el tiempo de búsqueda se mantuviera constante empeoro la calidad del juego. Consideramos que, si mejoramos el tiempo de búsqueda general del algoritmo y realizamos ciertas podas, con restricciones (que no calcule reinas encerradas), mejorara la calidad del resultado, pero no llegara a la calidad de la primera heurística.

Algoritmo de búsqueda

El algoritmo de búsqueda NEGAMAX consideramos que es eficaz en la calidad de la respuesta, pero es poco eficiente dado que debe generar una gran cantidad de estados del tablero. Al aplicar la poda Alpha-Beta se mejora considerablemente el espacio de búsqueda, aunque sigue siendo necesario generar toda la primera rama de búsqueda.

En nuestro caso decidimos guardar todas las jugadas posibles del jugador en dos listas (listaF para fichas y listaD para disparos), esto produce que el algoritmo se ralentice en profundidades altas.

Para mejorar la calidad del algoritmo se podría agregar una poda por estados similares cuyo peso heurístico sea el mismo a pesar de que la configuración es distinta (ejemplo cuando se permutan reinas), reduciendo así la generación de la primera rama del árbol de búsqueda, mejorando el tiempo general del algoritmo.

Conclusión

En general el NEGAMAX responde de forma deseada. Sin embargo, se podrían realizar mejoras que descubrimos luego de las pruebas finales: la principal mejora seria reemplazar las listas de movimientos las cuales ralentizan la búsqueda del mejor movimiento por una función que recorra el tablero en orden preguntando si el movimiento es válido. Esto mejorar notoriamente el tiempo de búsqueda, pero no la calidad del resultado. Otra mejora en la que pensamos fue almacenar ciertos tableros (los generados en los pisos múltiplos de 3,4 y 5) y cada vez que el NEGAMAX entre en esos pisos comparar el tablero actual con todos los guardados (en ese piso), si la configuración del tablero es similar (reinas permutadas o iguales, flechas en la misma posición) evitar generar esa rama agilizando así el tiempo de búsqueda.

Para las heurísticas no consideramos que existan más mejoras que las mencionadas anteriormente