Memoria práctica 3 Inteligencia Artificial.

-Análisis del problema-

Esta práctica trata sobre el juego del parchís , pero no un parchís cualquiera , se trata de un parchís en el que van a jugar dos jugadores , cada uno de ellos jugará con dos colores.
A pesar de que se enfrentan solo dos jugadores , hay que tener en cuenta que se enfrentan los 4 colores , esto quiere decir , que los colores pertenecientes a un mismo jugador se pueden comer entre sí.

Cada uno de estos colores cuenta con tres fichas , las cuales se irán moviendo con el principal objetivo de alcanzar la meta. Las fichas se irán moviendo según los valores de los dados (cada jugador puede mover la ficha que quiera de sus dos colores).

Los dados no son como los del parchís tradicional , se trata de dados con diferentes valores (los dados disponibles inicialmente son los números del 1 al 6 , salvo el 3) , por ejemplo, si se utiliza el número 6 , este dado no se podrá volver a utilizar hasta que todos los dados restantes se hayan utilizado, debido a que cuando todos los dados ya se hayan utilizado , se regeneran y estarán de nuevo disponibles.
Cada jugador también podrá tener en su posesión dados especiales (champiñón , bala , estrella , boo …etc) ,los cuales de darán una serie de poderes , estos dados se encontrarán repartidos por el tablero y el jugador solo podrá tener dos.
Los dados especiales pueden generar trampas que estarán en el mapa.

El jugador ganador será el que consiga llevar las tres fichas de alguno de sus colores a la meta.

El principal objetivo de la práctica es conseguir ganarle al máximo número de ninjas posibles , para ello debemos diseñar una buena heurística que nos lleve a conseguir el objetivo.
Todo este problema necesita de la función PODA-ALFA-BETA , la cual determinará cuál es el mejor movimiento .

-Explicación de la realización de la práctica-

-Poda-Alfa-Beta

En primer lugar, como ya comenté anteriormente , debemos realizar un algoritmo que recorra el árbol , para poder determinar el mejor movimiento , en mi caso he usado el algoritmo de PODA-ALFA-BETA.

Este algoritmo utiliza los valores de alpha y beta , para saber que ramas del árbol pueden descartarse.
Alpha al principio está inicializado a menos infinito y beta a más infinito , pero sus valores se irán actualizando conforme se vaya realizando la búsqueda.

Para poder realizar correctamente la búsqueda , nos apoyamos en un iterador , con el cual podremos obtener y recorrer los hijos.

```
ParchisBros iterador = actual.getChildren();
ParchisBros::Iterator it = iterador.begin();
Parchis siguiente_hijo = *it;
```

Alpha representa el valor máximo que el jugador maximizador puede garantizar hasta el momento en la rama actual del árbol .
Beta representa el valor mínimo que el jugador minimizador puede garantizar hasta el momento en la rama actual del árbol.

Si en alguna de las iteraciones nos encontramos con un valor de alpha mayor o igual que beta , realizamos un break y nos salimos del bucle (poda) , debido a que hay mejores opciones en otras ramas , por tanto evitamos tener que seguir explorando una rama que no determinará el resultado final.

```
if (alpha >= beta) {
        break;
}
```

De esta manera garantizamos explorar sólo los nodos que nos puedan llevar al movimiento más prometedor .

En el caso de que alpha sea menor que beta , la exploración continuará por la rama ,es decir, no se podará y se seguirá comprobando en iteraciones siguientes la misma condición (alpha > = beta).

-Heurísticas.

En mi caso he utilizado dos heurísticas valoraciónTest1 y valoraciónTest2 , cabe destacar que la mejor de ellas es valoracinTest1 , por tanto , es esta la que uso como solución al problema.

valoraciónTest2(no evaluable):

Para realizar esta heurística me he apoyado en la heurística que daba la práctica y a partir de esta he realizado modificaciones.
Para cada una de las fichas de cada color , en primer lugar he comprobado si el movimiento nos lleva a la meta , en el caso de que esto ocurriese , se suma a la puntuación del jugador 50.
Como siguiente condición , he contemplado que si el movimiento lleva a comernos un jugador , se sume a la puntuación del jugador 25.
Las dos siguientes condiciones las he cogido de valoraciónTest , he comprobado si la ficha está en una casilla segura , en este caso se sumará 10 a la posición del jugador , en el caso en que la ficha esté en la meta , sumaré 5 a la posición del jugador.
Por ultimo he contemplado la condicion de que si la ficha esta en la recta final para llegar a la meta , es decir en las casillas de color , se sumará 1 a la puntuación del jugador .
Para el oponente he utilizado exactamente las mismas condiciones , pero aplicadas a sus respectivos colores.

En cuanto a las ventajas de esta heurística , no podemos destacar ninguna debido a que no consigue ganarle a ningún ninja.

En cuanto a desventajas podemos destacar que solo afecta a la puntuación y de manera positiva , que la ficha esté en la recta final hacia la meta o que esté en la meta , pero no se contempla que la ficha pueda estar fuera de esta recta final, es decir, en otra parte del tablero.

Debido a que con valoraciónTest2 no obtuve un buen resultado , realice valoraciónTest1.

valoraciónTest1(SOLUCIÓN):

Para realizar esta heurística me he apoyado en valoraciónTest2 , pero he llevado a cabo algunos cambios , para poder encontrar un mejor resultado.
He mantenido las siguientes condiciones:

-La condición de que si el movimiento nos lleva a la meta sume 50 a la puntuación del jugador.
-La condición de que si el movimiento nos lleva a comernos una ficha sume 25 a la puntuación del jugador.
-La condición de que si la ficha está en una casilla segura sume 10 a la puntuación del jugador.

He quitado las siguientes condiciones:

-La condición de que si la pieza está en la meta sume 5 a la puntuación del jugador.

-La condición de que si la pieza está en la recta final para llegar a la meta sume 1 a la puntuación del jugador.

Estas dos últimas condiciones que he quitado las he reformulado antes de todas condiciones de la diferente manera:

```
puntuacion_jugador += 100;
Box casilla = estado.getBoard().getPiece(c, j).get_box();
puntuacion_jugador -= (estado.distanceToGoal(c,casilla) +
estado.piecesAtHome(c));
```

Con este trozo de código para cada ficha de cada color del jugador , se suma 100 a la puntuación del jugador. Una vez sumado estos 100 , se le va a restar a puntuación del jugador, la distancia hasta la meta y el número de fichas que tenga en la casa el color.
De esta manera el jugador se verá menos perjudicado cuanto más cerca tenga las fichas de la meta y cuanto menos fichas tenga en casa.

Para el oponente he utilizado exactamente el mismo código , pero aplicado todo a sus respectivos colores .

En cuanto a las ventajas de esta heurística , cabe destacar que he solucionado el problema que tenía en la heurística anterior , debido a que en esta nueva heurística afecta la posición de la ficha a la puntuación , de manera que cuanto más alejada esté la ficha de la meta , más afectará de manera negativa a la puntuación.
Otra ventaja es que esta heurística también contempla el número de fichas que tenemos en la casa , de modo que a mayor número de fichas en la casa más le perjudica a la puntuación , ya sea la del jugador o la del oponente.

FIN.

José Antonio Zamora Reyes