MEMORIA PRÁCTICA 3: PARCHIS

-ELECIÓN DE ALGORITMO

Para realizar la práctica lo primero que debemos conocer son las técnicas de búsqueda con adversario, en este caso conocemos dos el algoritmo **MINIMAX** y el algoritmo de **PODA ALFA-BETA**.

MINMAX

Es una técnica utilizada en teoría de juegos y toma de decisiones. Su objetivo es encontrar la mejor jugada posible para un jugador, asumiendo que el oponente también juega de manera óptima.

Propósito:

-**Maximizar el Mínimo**: Para el jugador maximizado, el objetivo es maximizar el beneficio mínimo que puede obtener.

-**Minimizar el Máximo**: Para el jugador minimizador (el oponente), el objetivo es minimizar el beneficio máximo que el jugador maximizado puede obtener.

Características:

-**Juego de Suma Cero**: Ideal para juegos de suma cero, donde lo que un jugador gana, el otro lo pierde.

-**Turnos Alternos**: Cada jugador alterna turnos, evaluando y eligiendo movimientos.

Funcionamiento:

1.**Construcción del Árbol de Juego**:

El árbol de juego representa todos los movimientos posibles del juego, alternando entre los turnos del jugador maximizado y minimizador.

2.**Evaluación de las Hojas**:

Las hojas del árbol representan los estados finales del juego, evaluados con una función de evaluación que asigna un valor numérico.

3.**Propagación de Valores:**

Los valores se propagan hacia arriba desde las hojas del árbol:

-**Nodo Max:** Elige el valor máximo de sus hijos, representando la mejor jugada del jugador maximizador.

-**Nodo Min**: Elige el valor mínimo de sus hijos, representando la mejor respuesta del jugador minimizador.

4.**Selección de la Mejor Jugada**:

En la raíz del árbol, se elige el movimiento que conduce al valor óptimo determinado por el proceso de propagación.

PODA ALFA-BETA

La poda alfa-beta es una optimización del algoritmo Minimax. Su propósito es reducir el número de nodos que se evalúan en el árbol de decisiones, mejorando así la eficiencia del proceso sin afectar el resultado final.

Propósito:

-**Optimizar Minimax**: Reduce el número de nodos evaluados en el árbol de juego, manteniendo el mismo resultado que Minimax.

-**Poda de Nodos Innecesarios:** Elimina ramas del árbol de búsqueda que no pueden influir en la decisión final.

Características:

-**Valores Alfa y Beta:**

-**Alfa**: El valor máximo que el jugador maximizador (jugador que intenta maximizar su puntuación) puede asegurar hasta el momento.

-**Beta**: El valor mínimo que el jugador minimizador (jugador que intenta minimizar la puntuación del oponente) puede asegurar hasta el momento.

-**Podas**:

-**Poda Alfa**: Ocurre en nodos maximizadores. Si un nodo hijo de un nodo minimizador produce un valor menor o igual a alfa, no se evalúan más hijos de ese nodo minimizador.

-**Poda Beta**: Ocurre en nodos minimizadores. Si un nodo hijo de un nodo maximizador produce un valor mayor o igual a beta, no se evalúan más hijos de ese nodo maximizador.

Funcionamiento:

1.**Inicialización:**

Se llaman al algoritmo Minimax con poda alfa-beta con valores iniciales de alfa = -∞ y beta = ∞.

2.**Recorrido del Árbol**:

El árbol se recorre de manera similar a Minimax, alternando entre nodos maximizadores y minimizadores.

3.**Actualización de Alfa y Beta**:

En cada nodo, se actualizan los valores de alfa y beta.

-**Nodo Maximizador:**

Actualiza alfa al valor máximo de sus hijos.

Si alfa ≥ beta, se realiza la poda (se detiene la evaluación de los hijos restantes).

-**Nodo Minimizador:**

Actualiza beta al valor mínimo de sus hijos.

Si beta ≤ alfa, se realiza la poda (se detiene la evaluación de los hijos restantes).

CONCLUSIÓN: por temas de rendimiento he elegido el algoritmo PODA ALFA-BETA, ya que tardará mucho menos tiempo en realizar la partida de parchís ya que reduce los nodos evaluados podando las ramas del árbol q no influyen en la decisión final.

Para la realización del código me he fijado en pseudocódigo de las diapositivas de teoria:

Para calcular el valor V(J,alfa,beta), hacer lo siguiente:

1. Si J es un nodo terminal, devolver V(J)=f(J). En otro caso, sean $J_1,\ldots,J_k,\ldots,J_b$ los sucesores de J. Hacer k ←1 y, si J es un nodo MAX ir al paso 2; si J es un nodo MIN ir al paso 5.
2. Hacer alfa ← max(alfa, V(J_k,alfa,beta)).
3. Si alfa >= beta devolver beta; si no, continuar
4. Si k=b, devolver alfa; si no, hacer k ← k+1 y volver al paso 2.
5. Hacer beta ← min(beta, V(J_k,alfa,beta)).
6. Si beta <= alfa devolver alfa; si no, continuar
7. Si k=b, devolver beta; si no, hacer k ← k+1 y volver al paso 5.

-HEURÍSTICA

FACTORES PARA LA REALIZACIÓN DE LA HEURÍSTICA

-LA DISTANCIA DE CADA JUGADOR A LA META

Este es de lo puntos más importante de la heurística, lo que queremos representar es que cuanto más cerca este un jugador de la meta, este será el que se moverá con mayor probabilidad, esto para que llegue lo antes posible ha la meta y no pueda ser elimina en el proceso por una ficha contraria mediante el simple juego o con el dado especial, ya que cuanto más tiempo permanezca la ficha en las casillas de juego, más probabilidad de ser eliminada.

Para que la ficha más cercana sea la más puntuada, lo que he hecho es poner las casillas del tablero, 68, menos las casillas que le faltan por ir a la meta (mediante la función “distanceToGoal()”), así cuanto más cerca este de esta la las casillas del tablero se restarán por un número menor y obtendremos una mayor puntuación.

-DESTRUIR LA FICHA DE UN RIVAL

Destruir la ficha de un rival en Parchís es una estrategia eficaz porque retrasa significativamente el progreso del oponente al obligar a su ficha capturada a regresar a la base, desde donde debe comenzar su recorrido nuevamente. Este retraso puede dar una ventaja competitiva, ya que el oponente necesita gastar más turnos para avanzar nuevamente la ficha capturada, mientras que el jugador que destruye puede avanzar sus propias fichas sin competencia inmediata. Además, cada vez que una ficha es destruida, el jugador recibe una bonificación para mover otra ficha, lo que puede aprovechar para posicionar mejor sus propias fichas.

En el codigo lo que se hace es que al principio se comprueba si ha destruido una ficha, una vez vemos que ha sucedido eso miramos cual es la ficha destruida y ponemos la puntuacion según si ha sido una ficha rival o del mismo equipo, claramente destruir la ficha del enemigo resultará más ventajoso.

-LLEGAR A LA META

Realizaremos este movimiento ya que es el objetivo para ganar una partido , para ello sumaremos puntuación según si llega una ficha, que lo sabremos con la función “isGoalMove()”.

-ESTAR EN UNA CASILLA SEGURA

Este aspecto ya estaba en el código de la función “ValoraciónTest()” dada para derrotar al Ninja 0.

Estar en una casilla segura es una estrategia beneficiosa porque estas casillas protegen las fichas de ser capturadas por los oponentes. Cuando una ficha está en una casilla segura, no puede ser comida por las

fichas rivales, lo que evita que regrese a la base y tenga que empezar su recorrido nuevamente. Esto no solo protege el progreso acumulado, sino que también permite a los jugadores centrarse en avanzar otras fichas sin el riesgo inmediato de perder posiciones.

-FICHAS SIN ENTRAR EN EL TABLERO

En el juego del parchís, tener muchas fichas sin salir (es decir, aún en la "casa" o punto de partida) puede ser una estrategia doblemente interpretada según la situación del juego:

Ventajas de tener fichas sin salir:

-Menor riesgo: Las fichas en la casa están a salvo y no pueden ser capturadas por el oponente.

-Control del juego: Puedes esperar un momento estratégico para sacar tus fichas y moverlas de manera más segura.

Desventajas de tener fichas sin salir:

-Menor movilidad: Con pocas fichas en juego, tienes menos opciones de movimiento en cada turno.

-Retraso en el objetivo: El objetivo del juego es llevar todas las fichas a la meta, por lo que retrasar la salida de fichas puede dificultar alcanzar este objetivo si no se maneja con cuidado.

Po ello para ganar la más rápido posible he tomado la decisión de sacar las fichas en casa, ya que haciendo la comprobación en el código he notado una mejora significativa con esta última opción, ya que nos da más opción de movilidad.

Por ello utilizamos la función “pieceAtHome()” que nos indica las piezas en el punto de partida y le pondero sumándolo a “puntuación_oponente” para que al final de la heurística esta puntuación reste, es decir, halla menos puntuación cuantas más fichas en casa halla, y viceversa. Para sumarle puntos multiplicamos , ya que si no hay ninguna ficha en casa no se obtenga ninguna puntuación.

-FICHAS EN LA META

Lo que quiero decir con esto es que tendremos jugar con aquel color el cuál ya tenga el mayor número de piezas en la meta, ya que eso significa que jugando bien podremos acabar ganando el juego con menos movimientos que con otro color que todavía no tenga ninguna pieza en la meta.

Para ello usamos la función “pieceAtGoal()” que nos muestra el número de fichas de un color que están en la meta, por eso para sumarle puntos multiplicamos , ya que si no hay ninguna ficha en meta no se obtenga ninguna puntuación.

NOTAS

- Ha la hora de poner las puntuaciones que suman a la puntuación que le da la heurística a los nodos, he tenido en cuenta factores más importantes, dándole ha estos más puntuación como que nuestra ficha destruya a una rival, que se dé más puntuación a las fichas de color que tengan más fichas en la meta. En el dado especial igual le he dado más puntuación a las que creía más importante.
  Pero a la hora de especificar para determinar el valor de los puntos que le he puesto ha sido por PRUEBA y ERROR, después de muchos intentos con distintas valoraciones he puesto la que me da los mejores resultados.
- He obviado dar puntuaciones ha el oponente, que resta el valor en el return, porque lo he probado ha dale valor con fichas del color que más fichas tienen en la meta y los puntos por tener casillas al tenerlas en casa, pero me daba peores resultados, por lo que he optado en hacer solo los puntos del jugador.