Práctica 2: Algoritmo Expect Mini-Max

Carol Álvarez
Ignacio Tapia
Alicia Touris

Índice:

Descripción del algoritmo. Expect MiniMax: 3
Diseño e implementación: 4
Análisis de los resultados obtenidos: 6
Diferencia de profundidad (la primera vez que se ejecuta el algoritmo) 6
Profundidad: 15 6
Profundidad: 10 6
Profundidad: 5 6
Profundidad: 2 6
Diferencia de valores de ataques 6
Rest 6

Descripción del algoritmo. Expect MiniMax:

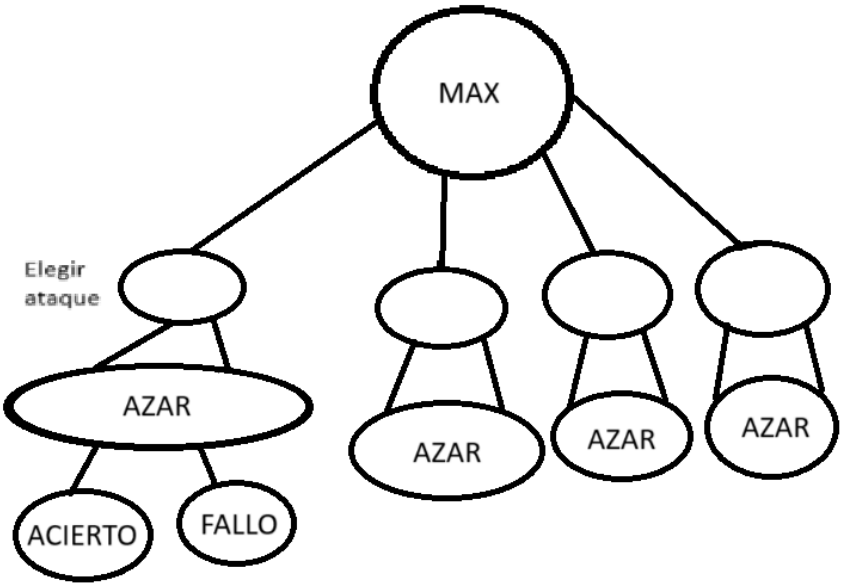
El algoritmo MiniMax se trata de un algoritmo de toma de decisiones comúnmente usado en juegos como el ajedrez o el tres en raya. Son juegos en los cuales los participantes compiten, maximizando o minimizando sus resultados con el fin de determinar la mejor jugada posible para uno de los jugadores en el estado de juego dado.

El algoritmo Expect MiniMax es una variante del algoritmo MiniMax común. Mientras que el algoritmo MiniMax simple considera que el adversario siempre va a escoger la jugada más óptima, el algoritmo Expect MiniMax no lo hace. Se aplica a juegos donde los adversarios no son óptimos o sus acciones son al azar.

De esta forma, el algoritmo Expect MiniMax busca la mejor acción teniendo en cuenta las probabilidades que generan las jugadas de azar junto con las posibles respuestas del contrincante. Para ello, se asignan valores a los nodos del árbol del juego en función de las probabilidades con las que se cuenta y los resultados que se esperan obtener, como la utilidad esperada del movimiento.

El algoritmo Expect Minimax realiza un recorrido en profundidad del árbol de juego, evaluando en cada nivel los nodos. Cuando llega a los nodos de azar, toma en cuenta todas las jugadas posibles y calcula la utilidad esperada considerando las probabilidades. En los nodos de los contrincantes, dependiendo de si el contrincante es maximizador o minimizador, elige la acción que maximice o minimice.

Diseño e implementación:



El algoritmo que hemos desarrollado se basa principalmente en la recursividad. Tenemos tres funciones que representan cada uno de los tres tipos de nodos diferentes, y estas se llaman entre sí. Con esto, abrimos el árbol rama por rama hasta llegar al final, almacenando esa información en la caché del proyecto hasta que se completa el proceso y nos quedamos con el último dato necesario para determinar qué acción realizar.

En la función de los nodos Max, a la que se le pasa un movimiento (que en la primera llamada será el estado inicial), accedemos a un bucle por cada ataque posible que tenga Max en función de su energía. Por cada uno de estos ataques, llamamos a la función Chance() con el movimiento correspondiente, que contiene el estado actual del juego junto con el ataque que vamos a ejecutar en ese movimiento, para que nos dé el valor de esos nodos. Luego, obtenemos el máximo de esos valores y ese será el ataque que vamos a realizar o pasaremos al siguiente nodo.

En la función Min, hacemos lo mismo que en Max, solo que en lugar de seleccionar el máximo de los valores, seleccionamos el mínimo.

En la función Chance, lo primero que hacemos es identificar de quién viene la llamada, si de Min o de Max. Si proviene de Max, por cada una de las posibilidades de ataque que se nos pasan en el movimiento, que son el fallo más cada una de las posibles cantidades de daño que se pueden hacer en ese ataque, llamamos a la función Min() para obtener el movimiento que debemos evaluar. Evaluamos este movimiento llamando a la función heurística que hemos diseñado,

```
    }

    return ((9 * state.maxHealth) - (10 * state.minHealth)) + (state.maxEnergy/20);
}
```

y multiplicamos el valor resultante por el porcentaje de fallo. Luego sumamos estos valores y eso es lo que devuelve la función.

Análisis de los resultados obtenidos:

Diferencia de profundidad (la primera vez que se ejecuta el algoritmo)

Profundidad: 15

Tiempo de ejecución: 7,3442890 segundos
Nodos abiertos: 198736675 nodos

Profundidad: 10

Tiempo de ejecución: 0.0102175 segundos
Nodos abiertos: 260879 nodos

Profundidad: 5

Tiempo de ejecución: 0.0007308 segundos
Nodos abiertos: 304 nodos

Profundidad: 2

Tiempo de ejecución: 0.0005452 segundos
Nodos abiertos: 4 nodos

Diferencia de valores de ataques

Rest

Hit Chance: 1
Min Damage: 20
Max Damage: 20
Energy: -20

Al cambiar los valores del ataque, va a coger siempre el Rest porque al hacer daño y dar energía, el valor del movimiento es mucho mayor que el resto.

En caso de cambiar los valores de cualquier otro ataque, la funcionalidad de la IA cambiaría respecto a ellos.