Inteligencia Artificial - Grado en Ingeniería Informática – 2019/2020

Actividad nº2 Ejercicio de búsqueda en heurística.- Ejercicio Min-Max inventado, con correlación con escenarios reales (ajedrez, damas …), en donde aplicar el algoritmo de suma cero MIN-MAX y en su caso poda alfa-beta.

Ejercicio de búsqueda heurística.

Inteligencia Artificial - Grado en Ingeniería Informática – 2019/2020

Estudiante: Borja López Pineda

Profesor: Luis Ignacio López Gómez

Índice

Resumen

El objetivo de este trabajo es codificar un algoritmo Minimax capaz de jugar al Conecta 4 con cierto grado de eficacia. Para ello deberá ser capaz de valorar cómo de favorable es un estado del tablero y decidir su siguiente movimiento que le permita llegar al mejor de los estados posibles.

Introducción

Conecta 4 es un juego donde dos personas compiten colocando fichas en un tablero para intentar formar una línea con sus fichas de longitud 4 o más.

Con la descripción anterior creo que el algoritmo Minimax es ideal para resolver este tipo de problema. Tenemos una acción a realizar que consiste en seleccionar una columna para colocar una ficha en ella, la combinación de movimientos se puede modelizar con un árbol dónde cada nodo es un estado del tablero. La puntuación de los nodos se determinará en función de qué jugador halla ganado o cómo de cerca esté de ganar. El objetivo de los jugadores es justo el contrario, encaja con la función maximizadora y minimizadora del algoritmo.

Algoritmo Minimax

Descripción y utilidad

Minimax es un algoritmo de toma de decisión cuya función es maximizar el resultado de un juego cuando el objetivo de los oponentes es completamente opuesto.

Dado que el rival intentará minimizar la función que el jugador quiere maximizar, sus movimientos son predecibles. En base a esta predicción de movimientos el algoritmo es capaz de encontrar el mejor estado al que podrá llegar si ambos jugasen de forma óptima y con el mismo criterio.

El gran problema de Minimax se evidencia cuando el número de estados posibles aumenta. Concretamente en el Conecta 4 con un tablero de 8x12 o 6x7 el tiempo de ejecución del algoritmo completo resulta inviable con la tecnología actual. Por esto, decidí limitar el número de jugadas a que intenta predecir

Pseudocódigo de Minimax

```
Función jugar(estado) : estado
	movimiento, nada <- max(estado, MAXIMA_PROFUNDIDAD)
	estado.aplicar(movimeinto)
	Devolver estado
FinFunción

Función max(estado, profundidad) : acción, entero
	máximo <- -infinito
	mejor <- nada

	Si profundidad = 0 || finJuego(estado) Haz
		Devolver nada, puntuar(estado)

	SiNo
		ParaCada movimiento En movimientos(estado) Haz
			estado.aplicar(movimeinto)
			nada, puntos = min(estado, profundidad - 1)
			Si puntos > máximo Haz
				máximo <- puntos
				mejor <- movimiento
			FinSi
			estado.revertir(movimiento)
		FinParaCada
		Devolver mejor, máximo
	FinSi
FinFunción
```

```
Función min(estado, profundidad) : acción, entero
  mínimo <- +infinito
  mejor <- nada

  Si profundidad = 0 || finJuego(estado) Haz
        Devolver nada, puntuar(estado)

  SiNo
        ParaCada movimiento En movimientos(estado) Haz
              estado.aplicar(movimeinto)
              nada, puntos = max(estado, profundidad - 1)
              Si puntos < mínimo Haz
                    mínimo <- puntos
                    mejor <- movimiento
              FinSi
              estado.revertir(movimiento)
        FinParaCada
        Devolver mejor, mínimo
  FinSi
FinFunción
```

Pseudocódigo de Minimax con poda Alfa Beta

La poda Alfa Beta es una optimización de Minimax que permite detectar cuando una búsqueda es inútil. Reduce el coste temporal ya que el número de nodos que se comprueban es menor.

Alfa es el mayor valor encontrado por max y Beta es el menor valor encontrado por min. Los valores de alfa y Beta se definen en las funciones max y min, se pasan de una a otra en cada llamada.

La condición de ruptura es que Alfa sea mayor o igual de Beta. Si estamos en max y el mayor valor es más grande que el menor de min, está claro que min no permitirá que lo obtengamos. Y si estamos en min y el menor valor es más pequeño que el mayor de max, max tampoco permitirá. En ambos casos tenemos asegurado un mejor valor y no tiene sentido continuar explorando esa rama.

La función jugar se modifica para pasarle unos valores por defectos de Alfa y Beta a max, max y min incorporan la actualización y paso de Alfa y Beta.

```
Función jugarPoda(estado) : estado
  movimiento, nada <- maxPoda(estado, MAXIMA_PROFUNDIDAD, -infinito, +infinito)
  estado.aplicar(movimeinto)
  Devolver estado
FinFunción
```

```
Función maxPoda(estado, profundidad, alfa, beta) : acción, entero
	máximo <- -infinito
	mejor <- nada

	Si profundidad = 0 || finJuego(estado) Haz
		Devolver nada, puntuar(estado)

	SiNo
		ParaCada movimiento En movimientos(estado) Haz
			estado.aplicar(movimeinto)
			nada, puntos <- minPoda(estado, profundidad – 1, alfa, beta)
			Si puntos > máximo Haz
				máximo <- puntos
				alfa <- puntos
				mejor <- movimiento
			FinSi
			estado.revertir(movimiento)
			Si alfa >= beta Haz
				Devolver mejor, máximo
			FinSi
		FinParaCada
		Devolver mejor, máximo
	FinSi
FinFunción

Función minPoda(estado, profundidad, alfa, beta) : acción, entero
	mínimo <- -infinito
	mejor <- nada

	Si profundidad = 0 || finJuego(estado) Haz
		Devolver nada, puntuar(estado)

	SiNo
		ParaCada movimiento En movimientos(estado) Haz
			estado.aplicar(movimeinto)
			nada, puntos <- maxPoda(estado, profundidad – 1, alfa, beta)
			Si puntos < mínimo Haz
				mínimo <- puntos
				beta <- puntos
				mejor <- movimiento
			FinSi
			estado.revertir(movimiento)
			Si alfa >= beta Haz
				Devolver mejor, mínimo
			FinSi
		FinParaCada
		Devolver mejor, máximo
	FinSi
FinFunción
```

Implementación

El lenguaje elegido es C++ por su velocidad. Se adjunta el fichero .cpp que puede ser compilado con GNU GCC configurado para la revisión 2011 del lenguaje.

Puntuación del tablero

Esta quizá sea la parte más importante, porque si el algoritmo no sabe cómo de favorable es un estado, tampoco podrá determinar cual es el mejor y, aunque persiga un estado concreto, este no le llevará a la victoria.

Con el tiempo he pasado por tres formas distintas de puntuar el tablero. Todas estas formas tienen algo en común, las estructuras de la máquina suman puntos y las del jugador restan.

La primera aproximación que realicé consistía en explorar cada posición del tablero y contar el número de fichas adyacentes en cualquier dirección. Solo me quedaba con la mayor de todas las direcciones y se elevaría a si misma para que el algoritmo priorice una gran línea antes que muchas pequeñas. Luego eliminaría esas fichas para no volver a contarlas. El problema de este método, a parte de su alto coste, ya que necesitaba copiar el tablero muchas veces, es su mal resultado. Logré mejorarlo un poco introduciendo una puntuación mucho más alta al estado en el que gana y una muy baja al estado en el que pierde.

El segundo intento me centré en lograr que la puntuación fuera la misma independientemente del punto de partida por el que se empezara a comprobar, al contrario de lo que ocurría en el método anterior. No solo tomaría en cuenta la mayor de las direcciones de una estructura, sino que tendría en cuenta todas las direcciones en las que se expanden todas las fichas. El rendimiento no mejoró demasiado y resultado seguía siendo insuficiente.

El tercer método consiste en contar todas las combinaciones posibles de 4 fichas que llevan a la victoria y puntuarlas en función del número de fichas que están rellenas. Si el número de fichas rellenas es 4, se detecta una victoria, la puntuación se hace máxima y no se buscan más posiciones. Esta vez las fichas adversarias no restan puntuación a menos que logren una victoria, solo se penaliza la derrota, no las estructuras rivales. Esto es así porque el algoritmo desempeña mejor si únicamente evita una derrota inminente y el resto del tiempo busca la victoria, en lugar de jugar de forma defensiva completamente.

Finalmente agregué una pequeña modificación para resolver un problema que noté. El algoritmo tendía a crear estructuras verticales que nunca podrían ser terminadas debido a la altura del tablero. Para solucionar este problema, y otros causados por el mismo motivo, decidí no puntuar las combinaciones que no pueden ser terminadas, es

decir, puntuar solo las combinaciones en las que la suma de los huecos libres y las fichas de la máquina suman 4.

Implementación de Minimax

Las funciones max y min no difieren demasiado de su pseudocódigo. Se ha implementado el código necesario para que los humanos puedan competir contra la máquina desde el teclado o desde el paso de parámetros.

Resultados

El programa resultante es coherente y juega con cierta eficacia. Aún así no es rival para el humano promedio. Cuando se ejecuta con una profundidad de 6 tiende a ganar el 30% de las veces. Los resultados con profundidad 4 y 8 no varían demasiado y una vez la profundidad aumenta por encima de 8 deja de mejorar.

Aparentemente se debe a la naturaleza impredecible humana, el algoritmo crea planes a futuro suponiendo que su adversario tiene el mismo criterio que él y que jugará de la forma más optima en base a ese criterio. Al no ser así, la estrategia de la máquina se ve frustrada y llega a un callejón sin salida.

La implementación del algoritmo Minimax no puede ser mejorada en resultado (en eficiencia seguro que hay formas de reducir el tiempo de cálculo), el gran factor limitante es la función puntuación. Parece que los humanos tenemos una forma de valorar la partida que no encaja del todo con la que he logrado codificar.

Interfaz Web

A fin de obtener datos sobre la eficacia del algoritmo, agregué la opción en el programa de jugar mediante paso de parámetros y construí un servidor web* que interactúa con él, haciendo de puente el usuario y el programa.

http://casabore.ddns.net:8082/

De esta forma, los interesados no tienen que descargar un ejecutable sin firmar. La web registra la profundidad a la que se configuró el algoritmo y quién fue el ganador.

*No es una página web, es un servidor únicamente diseñado para aceptar conexiones TCP, recibir los datos del protocolo http y contestarlos con un código html5 y Javascript que genera el propio servidor. Adjunto el programa en Autoit3.
Quizá exec de PHP resuelva mejor el problema, pero no me suele dar buenos resultados.