Métodos de Modelado y Optimización
Proyecto

Descripción:

El árbol de búsqueda de Monte Carlo es un algoritmo que nos permite crear agentes que actúen en un espacio con base en simulaciones. El árbol de búsqueda de Monte Carlo consiste en un algoritmo iterativo, en el cuál en cada iteración se elige una rama de búsqueda a expandir, se ejecuta una simulación hasta el final del juego y se almacena el resultado. De esta manera se va adquiriendo más información de manera procedimental y, llegado el momento de realizar la acción, se elige la acción que tenga la mayor probabilidad de victoria.

Por otra parte, Quarto es un juego de mesa competitivo para dos jugadores. El juego se juega sobre un tablero pequeño de 4x4, y se juega con 16 piezas que representan cada una de las combinaciones posibles de mezclar las siguientes características binarias: las piezas son blancas o negras, son altas o bajas, son cuadradas o circulares, son planas o huecas.

En cada turno a un jugador le toca **seleccionar** la pieza, y al otro jugador le toca **colocarla**. Cada turno los roles cambian (si a un jugador le tocó **colocar** la pieza este turno, en el próximo le tocará **seleccionar** la pieza). El objetivo del juego consiste en ser el jugador que **coloca** una pieza logrando formar 4 piezas en línea que comparten al menos una característica (e.g: que las 4 piezas sean circulares, independiente del resto de atributos). Las 4 piezas en línea pueden ser por fila, por columna, o en las diagonales del tablero.

Su objetivo en este laboratorio consiste en implementar un agente que juegue Quarto de manera exitosa contra humanos utilizando árboles de búsqueda de Monte Carlo. Dado que el juego consiste de relativamente pocos movimientos (al iniciar el juego en máximo 32 acciones el juego acabará), es posible hacer un buen mapeo del árbol de búsqueda de Monte Carlo, pudiendo conservar el estado completo de las simulaciones.

Con el fin de facilitar su trabajo, su profesor le brinda una implementación lógica del juego Quarto, así como una interfaz gráfica para poder evaluar su agente jugando contra él. La clase `Quarto` posee los siguientes métodos de utilidad:

- `turn_player()`: Retorna si el turno le pertenece al jugador
- `turn_ai()`: Retorna si el turno le pertenece al agente
- `has_finished()`: Retorna si el estado actual es un estado final
- `get_winner()`: Retorna 1 si ganó el agente, 0 si ganó el jugador
- `get_available_actions()`: Retorna una lista con las posibles acciones
- `do_action(action)`: Retorna el objeto de clase Quarto que se obtiene al ejecutar la acción action en el estado actual

Implemente el siguiente método solicitado en lenguaje Python.

1. (100%) Método `mcts(root, time_limit = 0.25, exploitation=0.5)` que recibe el estado actual del sistema (un objeto de la clase `Quarto`) `root`, un valor flotante `time_limit` que representa el límite de tiempo que tendrá el agente para "pensar", y un valor flotante `exploitation` que se utilizará por el algoritmo para decidir si explorar una nueva rama o aprovechar una de las conocidas. El método debe retornar la mejor acción encontrada al finalizar el límite de tiempo.
    a. Para manejar el límite de tiempo deberá tomar el tiempo al inicio del algoritmo, cada vez que ejecute una iteración deberá revisar si ya superó el límite de tiempo, en cuyo caso no deberá realizar más iteraciones.
    b. Una vez que se acaba el límite de tiempo, su agente deberá elegir la mejor acción con base en el estado actual del árbol (aquella que le permita ganar el juego).
    c. Dado que este juego es un juego adversario, lo que ocurre es el surgimiento de lo que se conoce como un minmax tree, dónde cada jugador busca la opción que maximice sus posibilidades de ganar, al mismo tiempo que reduce las posibilidades de su oponente. Como nuestro árbol llega hasta los estados finales, eso significa que las probabilidades de ganar siempre serán un 1 o un 0 para cada jugador (todo o nada).
    d. Cada iteración del algoritmo consiste de las siguientes etapas:
        i. Inicie en el estado inicial (`root`).
        ii. Mientras no se haya llegado a un estado terminal, cicle en los siguientes pasos:
        iii. Si este estado ha sido explorado antes, entonces genere un número al azar, de ser mayor al `exploitation` entonces se elige la acción que lleva al mejor estado conocido.
        iv. Si el estado no ha sido explorado o el valor generado fue mejor al `exploitation`, se elige una acción al azar de entre todas las posibles. Note que si el estado nunca ha sido visitado, será necesario agregarlo a nuestro árbol.
        v. Una vez que se llega a un estado final se debe hacer backpropagation:, para cada nodo (en orden inverso de visita) se debe calcular si el resultado obtenido cambia sus posibilidades, sabiendo que un jugador **nunca** elegirá una acción que lo lleve a la

derrota (lo que significa que podría ser que el resultado obtenido no cambie las probabilidades).

e. Cuando se acaba el límite de tiempo se revisa la probabilidad de victorias de los hijos del nodo `root`, y se retorna la acción que lleve a un hijo con probabilidad de 1.

Nota: Se permite utilizar la biblioteca de soporte time, math, numpy, random, pero no se permite utilizar bibliotecas que realicen el trabajo por usted. Cualquier otra biblioteca que se desee usar debe ser consultada primero con el profesor.