Universidad de Castilla- La Mancha

Diseño de Algoritmos

El Juego del Molino con MCTS

Autor

Iago García Suárez

18 de Mayo, 2022

Contents

1 Introducción

En este documento se explicará la implementación en Python de **El Juego del Molino**, un juego de tablero, por turnos y totalmente determinista y observable.

El juego es muy similar al tres en raya, con el añadido de que cada jugador tiene un número de fichas disponibles y una vez están todas dispuestas, deberá moverlas por el tablero para conseguir alinear tres fichas y así ir eliminando las del jugador opuesto.

El objetivo es implementar una inteligencia artificial que utilice árboles de búsqueda de Monte Carlo para valorar las jugadas que ejecutará. Posteriormente, se simularán varias partidas contra un jugador que haga movimientos totalmente aleatorios y se evaluará la proporción de victorias.

2 Reglas del juego

Existen distintas variantes. Para este caso se ha elegido la variante de 9 piezas. Los requisitos serán los siguientes:

- El tablero consta de 24 posiciones distintas.
- Hay dos jugadores (uno contra uno).
- Cada jugador dispone de 9 fichas.
- Inicialmente, el tablero está vacío.
- Cada jugador podrá realizar un movimiento por turno.

Las reglas, basadas en la versión original, serán:

- Al inicio, cada jugador puede situar una ficha por turno en cualquier posición libre del tablero, hasta haber puesto las 9.
- Sólo cuando estén todas las fichas colocadas en el tablero, se podrán mover de posición.
- El movimiento se puede realizar a las posiciones adyacentes conectadas. Cuando el jugador que tiene el turno disponga de 3 fichas, podrá mover cualquier ficha a cualquier posición libre.
- Cada vez que se consigan alinear 3 fichas del mismo color, el jugador correspondiente a esas fichas podrá elegir una ficha del jugador contrario para eliminarla.

- Un jugador pierde cuando disponga de menos de 3 fichas o no tenga ningún movimiento disponible.

3 Componentes del juego

Para representar cada estado y acción posibles en una partida, se ha representado como un objeto cada elemento principal.

3.1 Tablero

El tablero será una representación del tablero físico mostrado en la figura 1. Para acceder a sus posiciones, se ha interpretado como una matriz de 7x7 y se han definido las posiciones adyacentes para cada una de ellas.

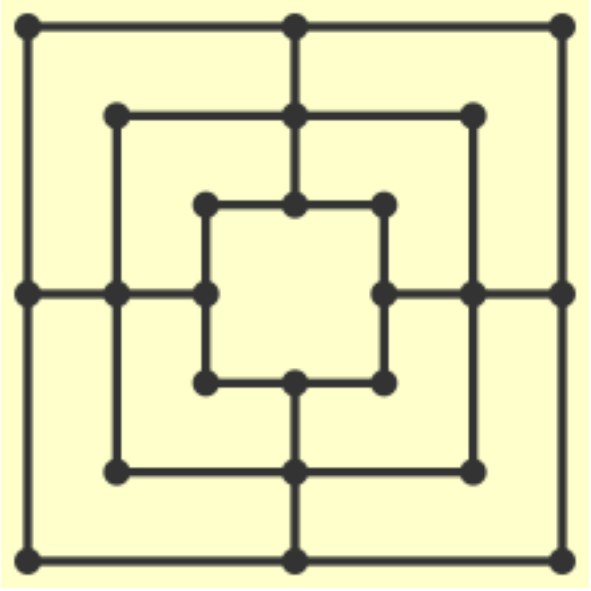

Figure 1: Tablero del Juego del Molino

Dado que el tablero es la representación más directa del estado de una partida, se ha implementado desde la misma perspectiva, de manera que en el propio objeto se definen los siguientes elementos:

- **Identificador**: Una cadena de texto única para identificar una partida.
- **Listas de posiciones de los jugadores**: Una lista donde se guardarán todas las posiciones del tablero donde haya una ficha colocada por el jugador 1 y otra para las fichas del jugador 2.

- **Número de fichas restantes**: Se controla por separado el número de fichas que le queda a cada jugador por situar en el tablero. La suma de este valor y la longitud de la lista de posiciones corresponde al número total de fichas que tiene un jugador.
- **Turno**: Comenzando en el turno 0, cada movimiento efectuado incrementa el valor en 1. De esta manera, el turno del jugador 1 tendrá lugar cuando el turno sea par y el del jugador 2 cuando sea impar.
- **Estado de la partida**: Representará si la partida sigue en curso, si se ha llegado a un empate o si ha ganado algún jugador.

Puesto que el identificador representa una partida y el turno tendrá siempre un valor distinto, es posible diferenciar dos estados diferentes en los que las posiciones de las fichas y el número de fichas sean iguales.

3.2 Movimiento

Cada acción posible está representada como un movimiento utilizando tres elementos:

- **Posición inicial**: Posición de la ficha en el tablero antes de moverla. En la fase inicial donde sólo se pueden colocar las fichas, este valor será nulo.
- **Posición final**: Posición a donde se quiere mover la ficha.
- **Matar ficha**: En caso de lograr alinear tres fichas con el movimiento, se deberá elegir una ficha del oponente para eliminarla. Por el contrario, este valor será nulo.

En caso de pedir tablas, también se utilizará un movimiento el cual tendrá los tres valores nulos.

3.3 Ficha

La representación de la ficha, como se ha visto, no se utiliza en el estado ni en el movimiento más allá de su posición. Su finalidad principal es la de validar una acción, de manera que al seleccionar una ficha para moverla, una posición final y, si procede, una ficha del oponente a eliminar, se construirá el objeto correspondiente para determinar el jugador al que pertenece cada ficha, si está viva o no, y la posición.

4 Árboles de búsqueda de Monte Carlo

Mediante árboles de búsqueda de Monte Carlo se ha implementado una inteligencia artificial que simule el comportamiento de un jugador. La forma de jugar vendrá definida por el valor UCT (Upper Confidence Bound applied to Trees) calculado sobre cada nodo del árbol que se genera a partir de un nodo raíz, correspondiente al estado actual.

4.1 Nodo

El componente principal de cualquier árbol de búsqueda será el nodo, generado por la aplicación de una acción a un estado. En este caso, el nodo cuenta con los siguientes parámetros:

- **Movimiento**: Movimiento con el que se ha generado el nodo.
- **Estado**: Estado en el que se encuentra la partida tras aplicar el movimiento.
- **Padre**: Nodo padre a partir del cual se ha generado el nodo actual.
- **Victorias**: Contador de las victorias de cada jugador así como de los empates, actualizado al alcanzar un nodo terminal en la simulación.
- **Visitas**: Número de veces que se ha pasado por el nodo, ya sea para seleccionarlo o para seleccionar a un hijo.
- **Hijos**: Nodos expandidos tras aplicar uno de los posibles movimientos al estado actual.
- **Posibles movimientos**: Movimientos posibles que puede efectuar el jugador con el turno en el nodo actual, descartando aquellos que ya se hayan utilizado para generar un nodo hijo.
- **UCT**: Valor con el que se determina el criterio de selección de un nodo entre los demás.

4.2 Fases de MCTS

El algoritmo de Monte Carlo aplicado a árboles de búsqueda consta de cuatro fases principales, que se ejecutarán en bucle hasta que no queden recursos (en el caso de esta implementación, se utiliza el tiempo como recurso):

1. **Selección**: A partir de un nodo raíz, el cual correspondería al estado actual de la partida, se selecciona recursivamente uno de los hijos hasta alcanzar un nodo terminal (es decir, cuyo estado corresponda a una partida en la que

haya ganado uno de los jugadores) o un nodo que no esté completamente expandido. Si los hijos del nodo han sido todos expandidos, se seleccionará el mejor, criterio representado mediante el valor UCT (Upper Confidence Bound applied to Trees). Este valor se calcula con la siguiente fórmula:

$$\frac{victorias - derrotas}{visitas} + c\sqrt{\frac{ln(visitas_{padre})}{visitas}} \tag{1}$$

donde c es una constante, que se ha establecido en $\frac{2}{\sqrt{2}}$. Su significado es que el nodo con el que se haya llegado a un número mayor de victorias y menor de derrotas por visita será más apto para ser seleccionado. Para mejorar la exploración, se añade la segunda parte de la fórmula. Con ella se pretende hacer que los nodos menos visitados sean más propensos a ser elegidos, de manera que se puedan evitar óptimos locales. Por lo tanto, al aumentar las visitas al padre con la selección de los demás hijos, la proporción con los nodos menos visitados será mayor y como consecuencia el valor UCT también aumentará.

2. **Expansión**: Una vez seleccionado un nodo no terminal y que no haya sido totalmente expandido, se elige aleatoriamente uno de los movimientos posibles que no se hayan utilizado previamente y se genera el nuevo nodo hijo.

3. **Simulación**: Cuando el siguiente nuevo nodo a analizar sea seleccionado, ya sea porque es terminal o porque se ha generado tras una expansión, se comienza a simular una partida. Durante esta simulación se van eligiendo movimientos aleatoriamente y aplicándolos a cada estado que se vaya obteniendo, hasta alcanzar un nodo terminal o que se hayan terminado los recursos. Si se ha alcanzado un nodo terminal, el valor resultante será el estado de la partida, ya que hace referencia al jugador que ha ganado. En caso de que los recursos se hayan agotado y no se haya alcanzado un nodo terminal, el valor será el correspondiente al empate.

4. **Retropropagación**: Se vuelve al nodo del que partió la simulación y, usando el valor obtenido por la misma, se incrementa el número de visitas al nodo así como el valor de las victorias correspondiente al valor de la simulación. Si el nodo tiene padre, se continúa la retropropagación del valor recursivamente hasta llegar a un nodo que no tenga padre, es decir, el raíz.

Cuando el algoritmo llega al límite de recursos, se calcula el valor UCT de cada nodo hijo del nodo raíz y se devuelve el movimiento asociado al nodo con mayor UCT.

4.3 Comprobación de MCTS

Para verificar que el algoritmo funciona correctamente, se ha diseñado un segundo jugador, al que se le denominará *Monoloco*, que seleccionará un movimiento totalmente aleatorio cada vez que sea su turno. De esta manera, Monte Carlo debería ganar en una alta proporción a este jugador para demostrar que tiene un comportamiento inteligente.

Una vez implementado, se hace una prueba con 50 partidas, y en los resultados Monte Carlo ha conseguido 47 victorias mientras que Monoloco ha conseguido 3. Esto se traduce en un porcentaje de victorias de Monte Carlo del 94%, con lo que se puede afirmar que el algoritmo es inteligente.

5 Implementación

5.1 Interfaz de la partida

Se ha utilizado la librería *Pygame* para implementar la interfaz visual que permitirá interactuar con el tablero para simular una partida de la manera más fiel a la realidad posible.

Según el modo seleccionado, el jugador correspondiente podrá ver las fichas seleccionables así como las casillas disponibles para moverlas, y en caso de conseguir alinear tres fichas, se marcarán las fichas del oponente para seleccionar una y completar así la construcción de un movimiento. Cuando el movimiento está completo, se comprueba en el estado actual si es válido y, si es así, se efectúa y se actualiza en la interfaz.

Por lo tanto, la comunicación entre la interfaz y el dominio se efectúa de manera bidireccional, enviando movimientos desde la interfaz al dominio y respondiendo éste con el nuevo estado (o el mismo estado en caso de que el movimiento no sea válido).

5.2 Menú principal

Para poder jugar una partida se deberá acceder al menú principal, una interfaz por línea de comandos implementada con *cmd*.

En este menú se puede iniciar una simulación de 10 partidas entre Monte Carlo y Monoloco, que mostrará el número de victorias totales de cada uno una vez haya terminado.

Para acceder a cualquier otro modo de juego que incluya un jugador humano, se debe iniciar sesión, por lo que habrá que desplegar el servidor.

5.3 Servidor y comunicación

Para centralizar la ejecución del sistema se ha implementado una API REST para el servidor, que utiliza *FastAPI*. En la misma librería se incluye lo necesario para la integración de *WebSockets*, que hará posible ejecutar una partida en línea.

De manera paralela a los clientes, el servidor mantiene el estado actual de la partida, de forma que lo que recibe de los jugadores serán los movimientos, los cuales valida y efectúa si son correctos. Cuando termina, envía mediante los *WebSockets* el nuevo estado de la partida a ambos jugadores, que actualizarán su interfaz correspondiente con el mismo.

5.4 Persistencia

Los datos persistentes que almacena el servidor serán los relativos a los usuarios, tanto sus credenciales como sus estadísticas, entre las que estarán el número de partidas jugadas, ganadas y perdidas y la proporción de victorias y derrotas.

Esta base de datos se gestiona con el módulo SQLite3.

6 Manual de usuario

6.1 Instalación

Para hacer posible la ejecución del programa es necesario instalar previamente los paquetes necesarios. Para ello, se abre una terminal en el directorio raíz del programa y se ejecuta el siguiente comando:

```
> pip install -r requirements.txt
```

6.2 Ejecución

Cuando se haya completado la instalación, se despliega el servidor para permitir acceder a todas las opciones.

```
> uvicorn server_api:app
```

En otra terminal, se ejecuta el cliente:

```
> python game_p1.py
```

A partir de aquí se pueden utilizar los comandos definidos. Escribiendo `help` se mostrarán todos los comandos disponibles y con `help<comando>` se mostrará la información detallada del comando indicado.

Para iniciar una partida, primero se deberá iniciar sesión. Dado que este sistema es un prototipo, se puede crear un usuario con:

```
> createuser
```

que pedirá un nombre de usuario y una contraseña. Una vez creado se puede iniciar sesión con:

```
> login
```

que, de manera similar a la creación de usuario, pedirá las credenciales.

Una vez se haya iniciado sesión, se puede crear una partida con el comando:

```
> newgame <modo>
```

donde `<modo>` corresponde al tipo de partida que se quiere iniciar. Para más detalles se puede consultar la ayuda del comando.

En caso de elegir el modo de jugador contra jugador en línea, se pedirá un nombre para la partida y una contraseña, la cual se puede dejar en blanco para que la partida sea pública.

Desde un segundo cliente, habiendo iniciado sesión, es posible ver las partidas creadas escribiendo:

```
> listgames
```

Con la partida a la que se desea unirse identificada, se puede utilizar:

```
> joingame <index | uid>
```

Es decir, unirse a una partida indicando el índice en la lista mostrada anteriormente o directamente el identificador de la partida si se conoce con anterioridad.

Cuando la partida haya finalizado, independientemente del modo, se podrá salir del sistema escribiendo `exit`.

En caso de que se desee configurar un servidor público, se deberá configurar apropiadamente el *Port Forwarding* en el router para las peticiones HTTP, y cambiar la dirección IP y el puerto en el archivo *const.py* para la constante *SERVER*.

7 Conclusión

Para finalizar, cabe mencionar algún detalle respectivo a la implementación de MCTS. Utilizando el criterio de selección UCT con la fórmula definida, se puede

advertir un patrón común en la mayoría de jugadas. Durante la fase inicial de colocación de las fichas, Monte Carlo no busca alinear tres fichas lo antes posible, sino que las dispone en alineaciones de dos fichas preparadas para conseguir el número máximo de eliminaciones en el menor tiempo posible. Este comportamiento es interesante ya que, en cierto modo, menosprecia la posibilidad del oponente para frustrar su jugada, provocando que al jugar contra un humano sea muy fácil ganarle en las jugadas iniciales.

Cuando se presentan simultáneamente un movimiento que le permita eliminar una ficha del oponente y un movimiento que le permita salvar una suya, suele optar por eliminar una ficha del oponente que a su vez le impida completar la formación, lo cual demuestra que ha estudiado la mejor solución para ese estado.

Por otra parte, al final de la partida, cuando el oponente tiene una ventaja superior que no permita a Monte Carlo encontrar un número mínimo de victorias, elige movimientos en cierto modo aleatorios, ya que prácticamente cualquiera llevará a una derrota, y los UCTs de los nodos se mantienen muy bajos y similares.

Posiblemente, ajustando la explotación y los pesos de las victorias o derrotas, se podría evitar la primera situación. Por el contrario, la última situación podría requerir una mejora de la exploración para contrarrestar el efecto. Sin embargo, llegado al punto en que sólo encuentra derrotas, no es especialmente relevante si intenta ganar o no, pero sí podría ser beneficioso mejorar la explotación para que al principio también valore eliminar las fichas del oponente lo antes posible, en lugar del mayor número de eliminaciones en el mínimo tiempo.