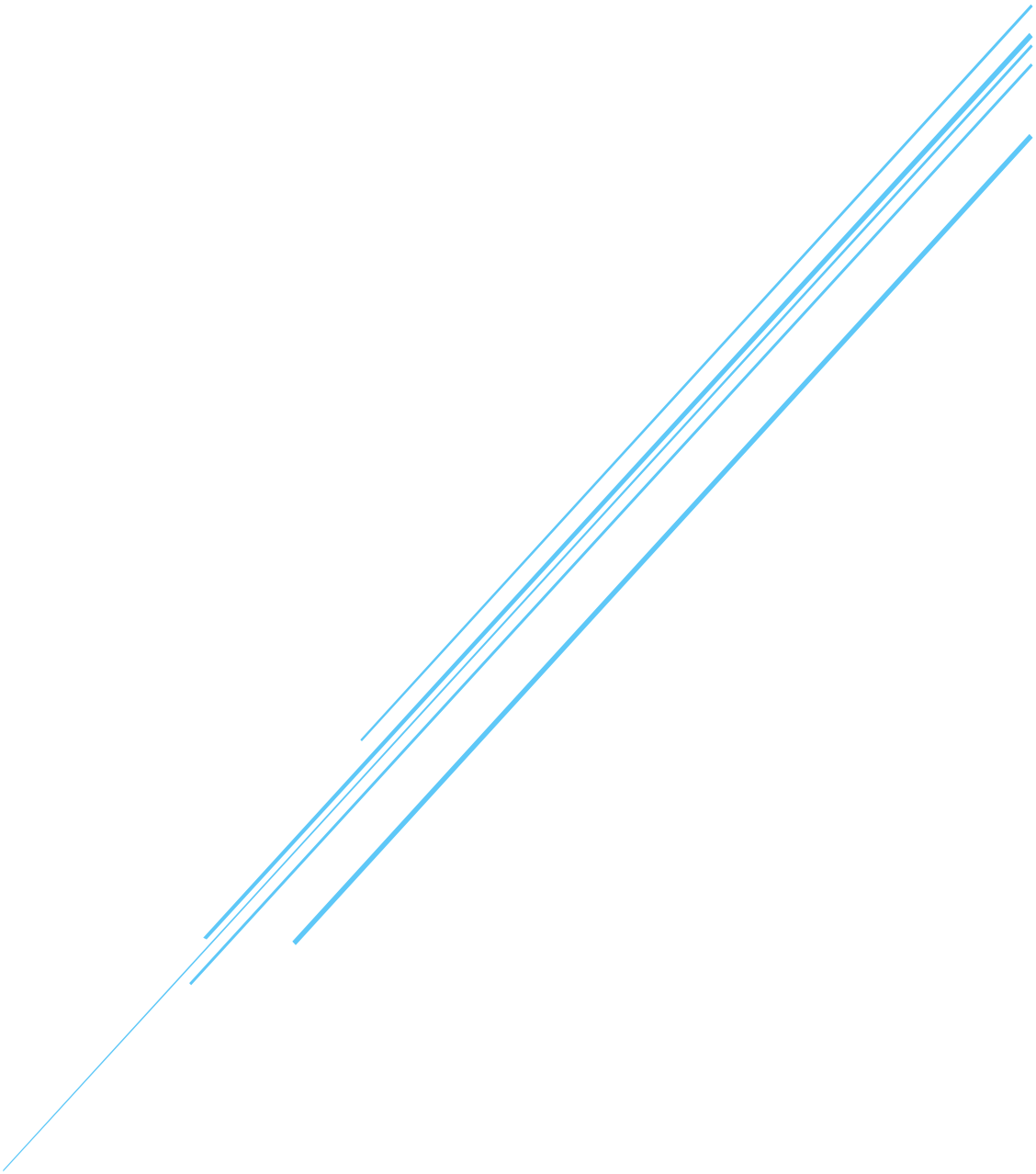
PROYECTO 1

GRUPO 4



ETSII

INTELIGENCIA ARTIFICIAL

TABLA DE CONTENIDO

MANCALA

1 INTRODUCCIÓN

En este proyecto se nos encargó realizar el juego Mancala en el lenguaje NetLogo e implementar un jugador automático basado en el algoritmo de Monte Carlo o Minimax.

Nuestro grupo ha optado por elegir el algoritmo de Monte Carlo ya que se adapta mejor a nuestras necesidades.

El juego Mancala es un juego de origen árabe, cuyo nombre significa mover. Este juego consiste en dos filas, cada una de ellas con 6 huecos y dos kalahas, situadas en los laterales del tablero. A cada jugador le corresponde una fila y la kalaha que esté a su derecha. Dentro de cada uno de los huecos de las filas encontramos de manera inicial cuatro semillas.

2 JUEGO

El objetivo del juego es conseguir el mayor número de semillas en la kalaha, de forma que el único movimiento posible es repartir todas las semillas de un hueco en sentido antihorario.

Tras realizar este movimiento pueden darse tres casos posibles:

1. Si la última semilla cae en un hueco en el que ya había semillas, sea del jugador que sea, es turno del siguiente jugador
2. Si la última semilla cae en la kalaha correspondiente al jugador que ha realizado el movimiento, dicho jugador obtiene un turno extra
3. Si la última semilla cae en un hueco vacío correspondiente a la fila del jugador que ha realizado el movimiento y además, hay semillas en el hueco justo delante de este, el jugador roba todas esas semillas y las introduce en su kalaha

La partida terminará cuando una de las dos filas se encuentre totalmente vacía y ganará el jugador que posea más semillas en su kalaha, para ello se usa la función **finpartida?**

Se ha implementado dos formas de jugar:

- **Player vs Player**: Dos jugadores humanos, ninguna máquina
- **Player vs IA**: Un jugador humano, una máquina

Ambos modos de juego se pueden seleccionar en la interfaz.

El turno de cada jugador se expresa en una variable global llamada "**turno**". Esta variable solo toma los valores cero y uno, de forma que, cuando su valor es cero, representa al jugador rojo (jugador 1) y cuando su valor es uno representa al jugador azul (jugador 2). La variable se incrementa en uno y posteriormente se realiza el módulo 2, cada vez que algún jugador realiza algún movimiento.

Como hemos dicho antes, puede darse el caso en el que la última semilla caiga en la kalaha, por lo que el jugador tiene un turno extra. Esto se refleja en el código mediante una variable global llamada "**turno-extra**", la cual puede tomar los valores cierto o falso. En el caso en el que esta variable se sea cierta, se vuelve a incrementar la variable "**turno**", para que, de esta forma, vuelva a tocarle al mismo jugador.

3 MODELIZACIÓN

Para este juego, hemos modelizado el tablero con una matriz de dos filas.

En la primera fila de la matriz se encuentra la línea superior del tablero, además, en la primera posición se encuentra la kalaha correspondiente al jugador que juegue en la parte superior del tablero.

En la segunda fila de la matriz se encuentra la línea inferior del tablero, además, en la última posición se encuentra la kalaha correspondiente al jugador que juegue en la parte inferior del tablero.

Hemos redimensionado el mundo para que los patches nos hagan de tablero, de forma que los patches superiores, los cuales tienen pycor = 1, corresponden a la línea superior del tablero y los patches inferiores, los cuales tienen pxcor = 0, corresponden a la línea inferior del tablero. Esto nos servirá cuando vayamos a aplicar una jugada, ya que así sabremos que el jugador ha seleccionado un hueco correspondiente a su fila y además nos servirá para representar ambas filas del tablero.

Además, la coordenada x de los patches nos sirve para determinar el hueco del tablero sobre el que vamos a trabajar. En el caso de la línea superior, si el jugador selecciona el hueco x, trabajaremos sobre la posición x en la fila superior de la matriz ya que corresponde con el índice del hueco en el que están las semillas que se quieren repartir. Por otro lado, en la línea inferior del tablero, si el jugador selecciona el hueco x, trabajaremos sobre la posición x-1 en la fila inferior de la matriz ya que, en este caso, dicha fila se encuentra desplazada una posición a la derecha ya que la kalaha del jugador superior toma dos partches:

ls1 = [0 4 4 4 4 4 4]

pxcor = 0 1 2 3 4 5 6

ls2 = [4 4 4 4 4 4 0]

pxcor = 1 2 3 4 5 6 7

Aquí podemos ver de una forma más visual la relación entre posiciones y coordenadas.

Una vez que sabemos la posición de la lista sobre la que vamos a trabajar, mediante las coordenadas x e y seleccionadas con el ratón, cargamos el contenido del hueco en una variable y empezamos a repartir semillas por el tablero sumando uno a cada hueco de la lista de la manera correspondiente a cada una de las filas (en la fila superior de derecha a izquierda y en la fila inferior de izquierda a derecha) y una vez que hemos dado una vuelta completa, las semillas que hayan sobrado las añadimos a nuestra kalaha, si es que han sobrado algunas. Para realizar todas estas acciones se usa la función **aplicaJugada**, la cual recibe como parámetros las coordenadas xr e yr y además el jugador que realiza el movimiento.

En cada movimiento, se mira también la última semilla, para saber si se activa el doble turno o la posibilidad de robar al otro jugador.

Todo esto se ha modelizado usando una variable llamada **matriz-global.**

4 PARTE GRÁFICA

El tablero, como dijimos anteriormente, está formado por dos filas, cada una de ellas con seis huecos y dos kalahas en los laterales, además, se ha añadido una columna adicional para representar el turno de una forma más visual. Para representar este tablero se han usado los patches del mundo, cambiándoles el color según lo necesitemos.

Para representar las semillas se han usado tortugas, cada una de ellas representa una semilla y se encuentra en uno de los huecos correspondientes al tablero. Las tortugas del jugador que juegue en la línea inferior del tablero son de color rojo, mientras que las del otro jugador son de color azul. Además del valor del patch para representar de forma más precisa el número exacto de semillas, ya que a veces no se puede distinguir a simple vista el número de semillas que hay.

Para realizar esta representación gráfica se han creado dos funciones:

- **representaTablero:** La cual representa el estado actual del tablero basándose en la variable **matriz-global**. Para ellos, se crean tantas tortugas como semillas haya y se colocan por el tablero en su correspondiente lugar.
- **representaTurno**: La cual representa el turno del jugador al que le toca jugar usando dos patches de colores, iluminando el patch del color correspondiente al jugador que tiene el turno.

5 MONTE CARLO

Como se ha dicho en la introducción, para implementar el jugador automático se ha usado el algoritmo de Monte Carlo.

Las funciones más relevantes que se han implementado para que el algoritmo de Monte Carlo pueda funcionar son:

MCTS:GET-RULES

Esta función recibe como parámetro un estado y devuelve como posibles jugadas una lista con los índices de las filas de la matriz cuyo contenido en la lista no está vacío.

Dependiendo si el jugador que acaba de jugar ha sido el jugador 1 o el jugador 2, se devuelve la lista dicha anteriormente para la parte superior o inferior del tablero respectivamente.

MCTS:APPLY

Esta función recibe un estado y una regla y en función de esos dos parámetros crea estados en los que se haya aplicado dicha regla.

Para ello se usa la función **aplicaJugadaMC** la cual es una adaptación de la función **aplicaJugada**, ya que **aplicaJugada** trabaja directamente sobre la **matriz-global** y no nos interesa eso, ya que nos interesa una función que devuelva el valor de la **matriz-global** pero sin llegar a cambiarlo de forma real

MCTS:GET-RESULT

Esta función recibe un estado y un jugador y en función de esos dos parámetros se comprueba si es un nodo final del árbol o si no lo es.

Esto es así ya que al algoritmo busca ganar, de modo que busca el camino más corto hasta el nodo que consiga hacerle vencedor. Para ello, hemos modelizado el get-result haciendo que devuelva el número de semillas que hay en la kalaha, de esta forma, busca siempre maximizar ese número, lo que se traduce en que busca ganar con la mayor puntuación posible. Además, se ha recortado el árbol debido a que se toma como nodo final, aquél en el que la kalaha tiene 24 semillas o más, ya que una vez llegado a ese estado, haga las jugadas que haga va a ganar puesto que hay 48 semillas en total y en ningún momento varía el número de semillas.

6 RESUMEN FUNCIONES

Aquí vamos a ver un breve resumen de las funciones implementadas, los parámetros que reciben y su funcionamiento:

- **setup**: Esta función no recibe ningún parámetro.
  Se usa para poder crear el tablero e inicializar la matriz global
- **creaTablero**: Esta función no recibe ningún parámetro.
  Se usa para crear el tablero inicial, cargando la matriz global y cargando el color de los diferentes patches del tablero
- **representaTablero**: Esta función no recibe ningún parámetro.
  Se usa para representar el estado actual del tablero basándose en la variable **matriz-global**, para que así sea más visual el estado actual del mismo
- **representaTurno**: Esta función no recibe ningún parámetro.
  Se usa para representar el turno actual. Para ello utiliza la variable global **turno**
- **aplicaJugada [xr yr jugador]:** Esta función recibe como parámetro la coordenada x e y del puntero del ratón y el jugador que aplica la jugada. Devuelve true o false, dependiendo de si se ha podido realizar o no la jugada
  Se usa para aplicar la jugada correspondiente en función del hueco que se haya seleccionado del tablero.
- **finPartida?**: Esta función no recibe ningún parámetro. Devuelve true o false dependiendo de si se ha terminado o no la partida
  Se usa para determinar si la partida ha finalizado y anunciar quien es el ganador
- **jugar**: Esta función no recibe ningún parámetro.
  Se usa para aplicar las jugadas, para ello usamos una espera activa, de la cual se sale cuando se haya completado la jugada. Esto es así ya que en el caso en el que no la usáramos, la función **representaTablero** no funcionaría de forma correcta
- **jugar2**: Esta función no recibe ningún parámetro.
  Esta función es similar a la función jugar, solo que en esta se implementan los comandos necesarios para poder jugar contra la máquina

- **jugar3**: Esta función no recibe ningún parámetro.
  Esta función es similar a la función jugar, solo que en esta se implementan los comandos necesarios para poder ver cómo la máquina se enfrenta consigo misma, por si queremos ver un ejemplo de juego habitual

Trabajo realizado por:

Frankford, Eduard

Honores Murray, Gustavo Ariel

Fernández Cuevas, Plácido