Títol de l'Article (màxim 3 línies)

Ivan Sanchez Resina

Resumen– La idea de este proyecto es implementar un juego de Connect 4 y añadirle varios algoritmos de IA que sirvan como oponentes a un jugador real y entre ellos mismos. De estos algoritmos implementados realitzar varias pruebas sobre los parámetros que regulan su comportamiento con le objectivo de estudiar cómo interactua cada uno con los otros y cual es el más eficiente.

Los algoritmos implementados son Minimax, Minimax con poda alfabeta, Monte Carlo Tree Search y una implementación parcial del algoritmo de Reinforcement Learning conocido como AlphaZero, el cual aplica el MCTS pero utilizando una red neuronal entrenada para predecir los mejores movimientos
Palabras clave– Algortimos IA, Monte Carlo Tree Search, Teoría de juegos, Minimax, Minimax AlphaBeta, Connect4, AlphaZero, Red Neuronal

Abstract– The idea of this project això no m'agradais to implement a ~~fully-functional game~~ of Connect 4 and add different IA algorithms that can act as opponents to a human Player and between themselves. Of these implemented algorithms, realise several tests on the parameters that regulate their behaviour in order to study how each interacts with the others and which one is the most effective.

The implemented algorithms are Minimax, Minimax with alpha-beta pruning, Monte Carlo Tree Search and a partial implementation of the Reinforcement Learning algorithm known as AlphaZero, that applies the MCTS but using a trained neural network to predict the bests movements.
Keywords– IA-driven algorithm, Monte Carlo Tree Search, Game Theory, Minimax, Minimax AlphaBeta, Connect4, AlphaZero, Neural Network

1 Introducción

Los juegos constituyen una parte importantísima en la vida de las personas. Especialmente cuando somos pequeños, ya que nos entretienen y nos divierten mientras nos permiten **desarrollar varias de nuestras capacidades** sin darnos cuenta. De las varias formas que pueden tomar, la que más me ha llamado la atención siempre, y no solo como jugador si no también a un nivel puramente conceptual, son los juegos de mesa.

Hay muchos tipos de juegos de mesa, pero creo que los más interesantes son los que, con un mínimo de materiales y un puñado escaso de normas permiten desarrollar tus habilidades cognitivas mientras te diviertes junto a otras personas. Desde luego hay mucho mérito en crear un juego complejo, con varias páginas de normativa, mil casuísticas y excepciones, una caja enorme con materiales para montar el escenario y que requiere un gran número de jugadores. Pero siempre he pensado que los juegos que más mérito tienen son precisamente los que huyen de esto. Un tablero y un puñado de fichas, varias decenas de cartas permiten tener horas de diversión con un tiempo de aprendizaje ridículo y sin necesitar materiales, ya que juegos como la oca, el parchís o el 3 en ralla se pueden realizar incluso sobre una hoja de papel, con dos garabatos y varias piedrecillas como fichas.

- E-mail de contacto: isanres@gmail.com
- Mención realizada: Computación
- Trabajo tutorizado por: Maria Vanrell (departament)
- Curs 20xx/xx

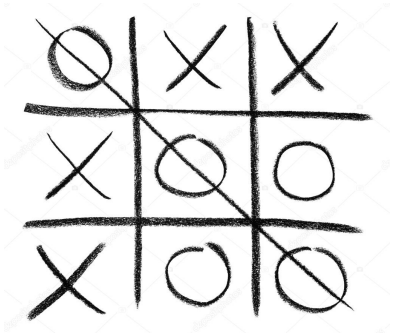

Fig. 1: No se requiere más que una superfície donde dibujar

Pero no solo en la sencillez radica su atractivo, o al menos no para mí. Lo que más me intriga es cómo estos juegos, en principio tan simples, permiten como comentaba antes desarrollar distintas habilidades. Rapidez mental, pensamiento lateral, capacidad de cómputo o memoria son solo algunos ejemplos [1] de las habilidades que puedes mejorar jugando de forma asidua a estos juegos.

Siendo una parte tan importante de nuestra sociedad, era evidente que llegaría un momento en el que se introduciría la **automatización de los oponentes**, para permitir más opciones y dar flexibilidad a estos juegos, mediante los algoritmos de IA oponentes. Sin embargo... ¿es realmente eso todo lo que pueden aportar los algoritmos y la automatización a los juegos de mesa? ¿Es la sustitución de el oponente humano todo a lo que han de aspirar estos agentes artificiales? Es evidente que si se limitan a eso los juegos pierden parte de su gracia, ya que eliminas la parte comunitaria, el aliciente de competir contra otras personas y, más importante, de mejorar a un juego con otras personas. Y es aquí donde entran las técnicas de Reinforcement Learning, o RL, que permiten a los agentes artificiales no solo ser buenos a un juego si no **aprender a jugarlo** por sí mismos.

Estas técnicas permiten principalmente dos cosas, siendo la más llamativa sin duda el hecho de que permite a un agente artificial mejorar por su propia cuenta para saber adaptarse a distintos tipos de jugadores y superar así a los mejores jugadores humanos de cada juego. Sin embargo hay una segunda aplicación de estas técnicas que a menudo pasa desapercibida: la utilización de un agente artificial que puede aprender a jugar a un juego nos permite desgranar cada pequeño detalle de él. No sólo creo que sea importante el resultado final de estos algoritmos (es decir, un jugador muy competente del juego) si no que creo que es casi más interesante el hecho de que podamos **ver y entender cómo ha aprendido a jugar** al juego, qué procesos hay detrás del juego, clasificar tipos de jugadores e identificar cómo potenciar esas habilidades que comentaba al principio que estos juegos nos permiten adquirir. Porque de esta forma, se abre un nuevo mundo de posibilidades. Agentes que permiten realizar un seguimiento intensivo de pacientes con Alzheimer, que permiten identificar los puntos flacos de un estudiante que está en pleno proceso de aprendizaje y actúa en consecuencia para potenciarlos o que incluso pueden **actuar no ya como diagnóstico si no como remedio parcial** también. Hoy en día ya se crean juegos que se pueden utilizar como refuerzo para la recuperación de pacientes con discapacidades cognitivas [2], pero la utilización de los datos de estos agentes artificiales, la utilización de algoritmos de RL como AlphaZero o QLearning nos puede permitir dar un paso más allá.

Así pues, mi intención al desarrollar este proyecto era utilizarlo como una aproximación al mundo de los agentes artificiales aplicados a los juegos de mesa, con la ambición de entender cómo funcionan y ser capaces de progresar hacia una aplicación más óptima de las técnicas que se han usado, que permitan la creación de agentes que puedan utilizarse como soporte para personas con problemas de salud mentales o infantes en desarrollo.

2 Estado del arte

El uso de Reinforcement Learning para juegos de mesa está en auge ahora mismo. Constantemente aparecen modificaciones a algoritmos existentes y algoritmos nuevos que superan a los anteriores por mucho. De hecho, cuando se inició este proyecto, en septiembre de 2019, uno de los algoritmos de RL aplicado a juegos de mesa más importante, por no decir el más importante y eficaz era AlphaZero, mientras que hace apenas un mes, apareció el algoritmo MuZero, hermano menor de AlphaZero, cuya principal característica es que puede aprender sin un simulador perfecto. Un simulador perfecto es como se conoce a los juegos como Go, Ajedrez o Connect4, los cuales te permiten saber exactamente qué estado tendrás después de realizar cada movimiento o incluso varios movimientos. MuZero, sin embargo, permite aprender en entornos donde no sabe exactamente qué efectos tendrán sus acciones, lo cual abre un nuevo mundo de entornos a los que se pueden aplicar estos algoritmos.

Volviendo a AlphaZero, este algoritmo es a su vez una evolución de AlphaGo, el cual, entre sus varias versiones, llegó a destrozar a varios de los mejores jugadores de Go y incluso a ganar al mejor jugador del mundo [3]. La evolución es tal que AZ fue capaz de vencer a la versión de AG que venció a uno de los mejores jugadores de Go del mundo en 2016... tras solo 3 días de entrenamiento y sin tener **ningún tipo de conocimiento previo** sobre el Go, simplemente extrayendo la información de las partidas. 21 días después, fue capaz de vencer al mejor jugador del mundo y a otros 60 jugadores profesionales sin problemas. Poco después fue capaz de obliterar por completo a Stockfish, la herramienta de entrenamiento de ajedrez que utilizada por la gran mayoría de profesionales del ajedrez... tras solo 4 horas de entrenamiento y sin ningún conocimiento previo sobre ajedrez.[4][5]

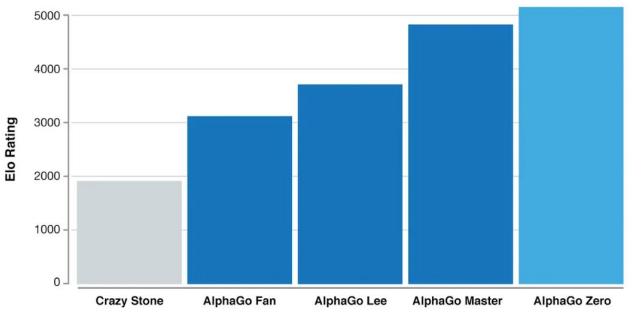

Fig. 2: Las victorias de AZ contra las versiones de AG. La última empezó con 0 conocimiento

Por si fuera poco, todas estas mejoras, actualizaciones y victorias han sucedido en los últimos 3 años, lo que demuestra que estamos ante un campo en pleno auge y que promete bastante. Aún así, lo más impresionante en mi opinión sobre la evolución en este campo de los últimos años es que las mejoras no solo han sido respecto a el número de victorias. Los avances se han realizado en tiempo de computación, en entornos a los que aplicar los algoritmos y también para la eficacia del algoritmo. No es precisamente un campo con pocas oportunidades para innovar, si no uno que permite a todo el mundo aplicar su aproximación al mismo y contribuir al desarrollo de esta tecnología.

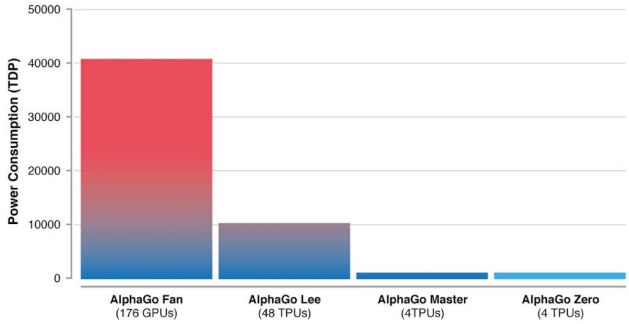

Fig. 3: Cada una de las nuevas versiones mejoraba la eficiencia además de la eficacia

3 Objetivos

A nivel general y sin entrar en requisitos para considerar completo el proyecto este trabajo tiene principalmente un objetivo: servir como herramienta que me permita introducirme en el mundo de los algoritmos de RL aplicados a juegos de mesa y permitirme entender cómo funcionan para seguir explorando este campo en un futuro.

Esta idea se delimitó más al inicio del proyecto con el objetivo de tener una carga de trabajo asequible y que proporcionara un **equilibrio correcto entre aplicación de conocimientos que ya poseía e investigación y aplicación de algoritmos desconocidos**. Así pues, se decidió lo siguiente:

- El juego que se desarrollaría sería el Connect4 o Conecta 4 con gravedad, un juego sencillo con normas básicas que simplificaba la parte del proyecto centrada en la representación del juego y las funciones básicas como mover ficha o las posibilidades en cada ronda.
- Antes de iniciar la aplicación de un algoritmo de RL se implementaría un algoritmo conocido que no pudiese lastrar mucho el desarrollo
- Como algoritmo de RL se utilizaría un esqueleto de código que adaptaba el AlphaZero a Connect4, así que no se tendría que implementar de 0, solo entenderlo y variar los parámetros para entender su funcionamiento
- En caso de que el desarrollo avanzara de forma correcta se podrían tomar dos caminos de mejora: modificar el juego base e implementar el ajedrez o aplicar un segundo algoritmo de RL

Con estas directivas en mente se establecieron los siguientes objetivos:

- Implementar un juego funcional de Connect4
- Implementar el algoritmo Minimax como algoritmo conocido
- Entender el funcionamiento de AZ y adaptar el código para que pudiese jugar contra el Minimax
- Realizar variaciones en ambos algoritmos para estudiar la interacción entre ellos

4 Metodología y herramientas

A lo largo del proyecto se ha usado un número reducido de herramientas, debido principalmente a que las tres fases que se han ido alternando a lo largo del desarrollo (investigación, implementación y gestión) se han podido llevar a cabo utilizando una o dos herramientas distintas únicamente.
Así pues, haré en esta sección una breve mención a las herramientas utilizadas y después explicaré las fases que se han ido realizando durante el desarrollo.

- Python 3.7 como único lenguaje de programación para todo el código implementado
- Pycharm como entorno de desarrollo principal
- Spyder (distribución de Anacondoa) se ha utilizado para realizar el entrenamiento básico de la NN de AlphaZero y también se ha utilizado para crear el entorno que se ha usado luego desde Pycharm, ya que desde este último entorno no fue posible realizar la instalación de todas las librerías que eran necesarias para ejecutar el código utilizado de AlphaZero
- Google Scholar para investigación sobre los distintos módulos implementados
- Trello como método de organización y visualización de tareas a realizar
- Overleaf para desarrollar el LaTeX del Informe Final
- GitHub se ha utilizado para subir el código al final del desarrollo y también de manera local para la implementación de cada algoritmo que se ha intentado implementar a partir de Minimax

El desarrollo de este proyecto se ha estructurado en 4 períodos, el final de cada uno de los cuales está marcado por la entrega de uno de los informes (inicial, seguimiento 1 y 2 y final). La metodología que se ha utilizado consta, como he comentado antes, de 3 fases, que se han ejecutado siguiendo la misma estructura en los 3 últimos períodos. Estas fases son las siguientes:

- Fase de investigación: Durante esta fase se ha recopilado información sobre los distintos algoritmos y se ha intentado entender cuál es su funcionamiento básico y cómo se pueden implementar con la estructura de tablero propia. Dado que el principal objetivo de este proyecto era familiarizarse con distintos algoritmos, ha tenido un papel crucial y se ha llevado a cabo en las 4 fases.
- Fase de desarrollo: Durante esta fase se implementaban los algoritmos estudiados durante la fase de investigación. Esta fase ha tenido su mayor importancia en los períodos 2 y 3, mientras que tuvo un papel menor en el 4 y no estuvo presente en el 1.
- Fase de gestión: Durante esta fase se escoge qué tareas se iban a implementar durante un período y se estimaba el tiempo de investigación y desarrollo que iba a llevar. También se diseñaban planes de contingencia y el mínimo de tareas necesarias para considerar un período satisfactorio. Se realizaba al inicio de cada período y se plasmaban los resultados de esta fase en los informes. Además, se utilizaban 5 tablas en la aplicación Trello para llevar un control de las tareas realizadas y por realizar [6].

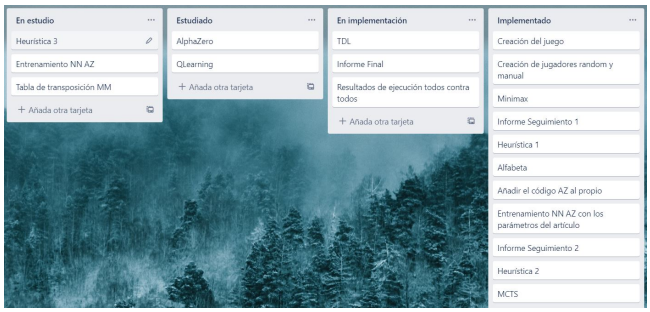

Fig. 4: Imagen de las 5 tablas con las que se ha trabajado en el estado final del proyecto

5 DESARROLLO

El objetivo de esta sección es hacer una breve presentación de los distintos módulos implementados, con el objetivo de entender qué estructura tienen y en el caso de los algoritmos una breve explicación con pseudocódigo de su funcionamiento. En los anexos se incluirán informaciones más detalladas de su funcionamiento

5.1. El juego

El primer elemento desarrollado fue el juego en sí, implementando los jugadores Random y Manual para poder realizar pruebas rápidas y sencillas sobre si el funcionamiento del flujo de la partida era el correcto. Además, se podía utilizar el jugador Random para probar la eficacia y eficiencia de los algoritmos y heurísticas implementadas. Este código está formado por 3 archivos de código: board (incluye la definición de la representación del tablero), players (incluye la definición de los jugadores) y main (incluye el flujo principal del juego)

5.1.1. Tablero

El tablero es la parte más importante del código, concretamente la representación que se ha implementado. Después de consultar algunos artículos explicando en profundidad el funcionamiento del juego [7] decidí implementar lo siguiente:

- Como estructura de datos se impelmentó, en vez de una matriz de X*Y, una lista de listas vacías, cada una de las cuales representa una columna. Así, para realizar un movimiento simplemente hay que poner un 1 o un 0 en la lista que toque y no hay que jugar con los índices ni pasar varios datos. Además, es más rápido comprobar si en una columna podemos poner ficha o no o el estado actual de la partida.
- Como funciones asociadas hay varias, siendo las más importantes makeMove (coloca una ficha en la columna que toque), isTerminal (devuelve el estado de la partida actual) y children (devuelve una lista con todos los tableros que podemos tener a partir del tablero actual)

Además, tanto el tamaño del tablero como el número de fichas que hay que encadenar para ganar son parametrizables, para poder realizar distintas pruebas sobre el rendimiento de los algoritmos.

5.1.2. Jugadores

Para los jugadores se creó una clase principal, que recibe varios parámetros (el tipo de heurística que usan, si usan NN o no, la profundidad a la que buscan, etc) y luego una subclase para cada algortimo implementado que tiene una función findMove para decidir el siguiente movimiento a hacer. En los siguientes apartados se profundizará en los tipos de jugadores que hay y en cómo funcionan.

5.1.3. main

El bucle principal no tiene ningún misterio, es el bucle que simula una partida normal y sigue el siguiente esquema:

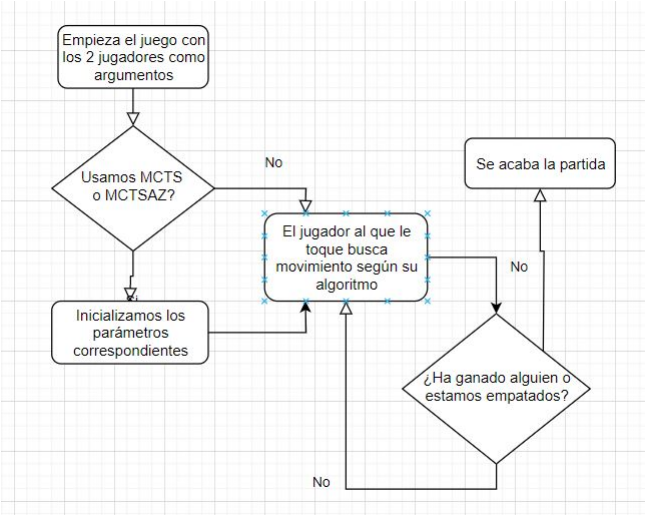

Fig. 5: Estructura principal del juego

5.2. Minimax

5.2.1. Algoritmo

El primer jugador implementado fue Minimax básico. Este algoritmo, está explicado en más profundidad en otros artículos y páginas [8] [9], por lo que simplemente explicaré lo implementado por mí, que son las heurísticas que usa el algoritmo. Por resumir, la heurística es la fórmula matemática que utiliza el algoritmo MiniMax para evaluar un tablero concreto y determinar cuán bueno o malo es para si mismo (siendo el Minimax uno de los jugadores de la partida). Durante este proyecto se han desarrollado 2 heurísticas, con 2 mejoras cada una.

5.2.2. Heurísticas

- H1: En esta heurística se busca el número de piezas juntas que hay de cada jugador y se suma o resta al valor total en función de si son nuestras o de nuestro oponente. Cuantas más juntas más valor tienen. En un principio la escala definida era 10 para 2 fichas juntas, 100 para 3 y 10000 para 4, pero después se corrigió a 5, 12 y 50 para evitar que 6 parejas valieran menos que 1 trío. Este cambio dió buenos resultados, por lo que se convirtió en la primera versión definitiva.

- H2: Esta heurística es algo más exhaustiva. Para cada posición del tablero se calcula cuantas posibilidades de ganar tenemos (en un tablero sin piezas hay 4 posibilidades para cada casilla: horizontal, vertical, diagonal derecha y diagonal izquierda). Si hay una pieza que no es nuestra en alguna de las 4 direcciones dentro del alcance esa posibilidad no se contabiliza. Para el número total se suman las posibilidades propias y se restan las del oponente. Debido al gran cómputo que exige esta opción se limitó a buscar en las casillas hasta la máxima fila ocupada en el tablero actual, ya que si no el tiempo para tomar una decisión es demasiado, incluso a profundidades bajas.

- Mejora: Después de implementar ambas heurísticas se introdujo un nuevo elemento a ambas que dió buenos resultados: pesos a las columnas. Para cada heurística se sumaban puntos por las columnas centrales (5 para la 4, 3 para la 3 y la 5, 1 para la 2 y la 6) y se ignoraban las marginales.

5.2.3. Alfabeta

Al igual que con Minimax, no entraré mucho en detalle con la implementación de la mejora AlfaBeta, ya que existen otros artículos que tratan el algoritmo en más profundidad [10] [11], por lo que simplemente explicaré que es una mejora del algoritmo Minimax que no afecta a la eficacia del mismo (gana lo mismo) pero si al tiempo de cómputo, ya que evita explorar las ramas menos prometedoras, mientras que el Minimax base las explora todas. Esto hace que en el Minimax base solo pudiera usar hasta 4-5 de profundidad sin que el tiempo fuera demasiado elevado mientras que con AlfaBeta he podido llegar hasta 8 sin problemas. Después de hacer varias pruebas decidí utilizar el jugador Alfabeta para todas las pruebas contra el resto de algoritmos:

Tiempo en s AB/MM		Profundidad de búsqueda			
		2	3	4	5
Partidas jugadas	5	0.07/0.15	0.32/0.79	0.86/4.83	2.56/70.34
	10	0.18/0.23	0.38/1.23	1.33/10.14	4.64/130.44
	15	0.21/0.41	0.76/2.24	2.37/13.73	
	20	0.3/0.56	1.17/3.94	3.89/24.02	
	25	0.48/0.76	1.41/4.45	5.22/28.08	

TAULA 1: LOS RESULTADOS ERAN IDÉNTICOS EN AMBOS CASOS O CON DIFERENCIAS MÍNIMAS, PERO LA DIFERENCIA DE TIEMPOS ERA ABISMAL. ESTAS PRUEBAS SE REALIZARON CONTRA EL RANDOM PLAYER Y SACANDO LA MEDIA DE 3 EJECUCIONES PARA CADA CELDA

5.3. Monte Carlo Tree Search

Mini intro, gran descubrimiento de AZ

5.3.1. Fases del algoritmo

Funciona por nodos. Cada nodo representa un estado del tablero actual. Cada rama representa un movimiento. El funcionamiento básicamente es el siguiente: se ejecuta número predefinido de simulaciones para determinar el mejor movimiento posible. Cada una de estas simulacones sucede en 4 fases y resumido al máximo consiste en escoger un tablero y simular una partida desde ese tablero en concreto haciendo movimientos aleatorios hasta ver quien gana. Las 4 fases son las siguientes: Cada simulación sigue el siguiente esquema
nodo hoja = nodo con todos los hijos explorados (esta definición cambia, en mi caso es todos los hijos explorados + no quedan opciones por realizar porque solo hago una expansión cada vez)

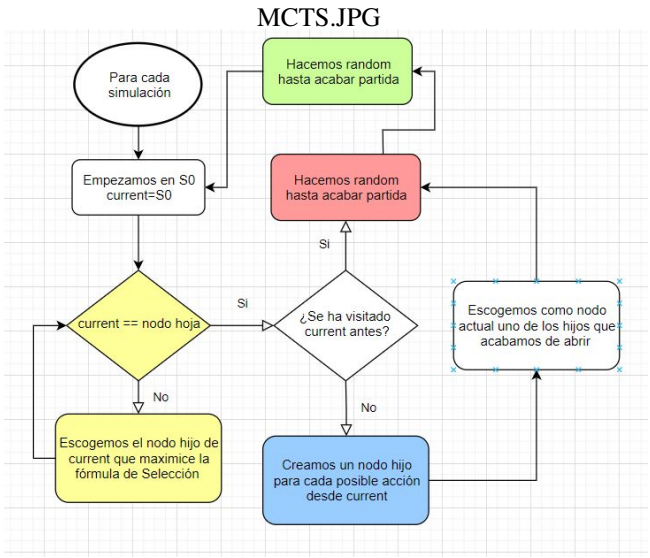

Fig. 6: Esquema de cada una de las simulaciones. Las fases amarillas corresponden a la Selección, la azul a la Expansión, la roja a la Simulación/Rollout y la verde a la Backpropagation

En mi caso, como he comentado antes, la fase de Expansión es algo diferente, y la definición de nodo hoja para la fase de Selección también. Este es el esquema que siguen esas fases:

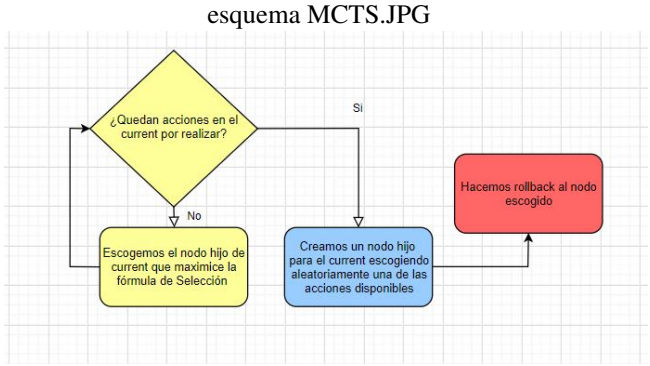

Fig. 7: Solo damos un nodo por explorado si no le quedan movimientos por hacer. Cuando expandimos solo lo hacemos con uno de los movimientos disponibles al azar en vez de todos a la vez.

Esta es una implementación perfectamente válida del MCTS y es la que he escogido realizar por el hecho de que si se expanden todos a la vez al intentar aplicar la fórmula de Selección se intenta calcular la media entre victorias y visitas, y para los hijos no escogidos eso es 0/0, lo cual intorduce dificultades en el código. Así nos aseguramos de que todos los nodos a los que les aplicamos la fórmula han sido visitados al menos una vez.

5.3.2. Fórmula de selección

Explicación fórmula

5.4. AlphaZero

5.4.1. Algoritmo

5.4.2. Modificación MCTS

6 Resultados

7 Conclusiones

Agradecimientos

Por mi parte me gustaría agradecer principalmente a mi familia, especialmente mi madre, que han entendido la importancia que este proyecto tenía para mi y me han apoyado y animado a seguir en todo momento.
También a mis compañeros y amigos, tanto de la universidad como de fuera de ella, que han sabido ver cuando necesitaba un respiro y me han apoyado y cuidado de mi durante el desarrollo. Gracias por estar ahí cuando lo neecsitaba y cuando creía que no lo necesitaba
Y agradecer, por supuesto, a la mayoría de los profesores de la carrera de Ingeniería Informática, especialmente a los de la mención de Computación y especialmente a los de las asignaturas Conocimiento Raciocinio e Incertidumbre, Inteligencia Artificial y Aprendizaje Computacional por presentarme los conceptos y las herramientas que me han llevado a escoger este tema y encontrar mi verdadera vocación.
También a los profesores de otras asignaturas que se suelen subestimar como Ingeniería del Software, Gestión y Desarrollo de Software y gestión de Proyectos por presentarme las herramientas necesarias y familiarizarme con el desarrollo de un proyecto de software como este.
Finalmente, pero no por ello menos importante, a las personas detrás de los artículos adjuntados en la bibliografía, especialmente a los autores de *Towards Data Science* por esas expliaciones sobre los distintos algoritmos que me han ayudado a comprender más los entresijos de las herramientas que espero utilizar en el futuro.
Gracias a todos y a todas por todo, espero que este sea solo el inicio de una larga carrera en este campo y espero hacer honor a todos los conocimientos que he adquirido durante el desarrollo de este proyecto para hacer del mundo un lugar mejor.

Referencias

[1] Second Nature Academy. 8 ways board games teach like skills, 2016.

[2] Miriam Llorens Monzó. Sistematizacion de un juego de mesa para la rehabilitaciÓn cognitiva de pacientes con daÑo cerebral adquirido. 2016.

[3] DeepMind Blog. Alphago china, 2017.

[4] DeepMind Blog. Alphago zero: Starting from scratch, 2017.

[5] Mike Klein. Google's alphazero destroys stockfish in 100-game match, 2017.

[6] Ivan Sanchez Resina. Tableros utilizados durante el desarrollo del proyecto, 2019.

[7] Victor Allis. A knowledge-based approach of connect-four. 1988.

[8] Wikipedia. Minimax, 2019.

[9] Marissa Eppes. Game theory — the minimax algorithm explained, 2019.

[10] Wikipedia. Alpha-beta pruning, 2019.

[11] JavaTpoint. Alpha-beta pruning, 2018.

APÉNDICES

1.1. Secció d'Apèndix

... .