# Búsqueda adversaria para aprender a jugar a Otelo

Eloy Sancho Cebrero
Universidad de Sevilla
Sevilla, España
elosanceb@alum.us.es

Iván Fernández Limárquez
Universidad de Sevilla
Sevilla, España
ivaferlim@alum.us.es

***Abstract*—Este es el resumen del trabajo. Aquí puedes describir brevemente el objetivo, el método utilizado y los resultados más importantes. Suele ocupar unas 5–6 líneas.**

***Index Terms*—Monte Carlo Tree Search, MCTS, Upper Confidence Bound for Trees, Othello, Otelo, Reversi, Python, Inteligencia Artificial**

## I. Introducción

Describe aquí el contexto del proyecto, motivación y objetivos.

## II. Preliminares

Preliminares

## III. Implementación

Para la implementación del proyecto se han realizado los siguientes pasos: se ha creado una versión inicial de Otelo utilizando Pygame, se ha creado un agente básico utilizando el algoritmo de Monte Carlo Tree Search, se han generado datos a partir de ese agente básico, se ha creado y, posteriormente, entrenado una red neuronal a partir de los datos generados, se ha utilizado la red neuronal como sustituta de una de las funciones de las que hace uso la implementación inicial del algoritmo MCTS y, finalmente, se ha adaptado la versión inicial de Otelo (en la que no se podía jugar contra ningún adversario) para que el agente desarrollado pudiese jugar contra el usuario.

A continuación, desarrollaremos la implementación de cada uno de los pasos mencionados.

### A. Otelo en `pygame`

El primer paso a seguir en la implementación del proyecto fue la creación del juego Otelo (sin presencia de un agente contra el que jugar) que nos serviría como base para crear tanto el agente MCTS como el agente final que haría uso de la red neuronal.

El juego se planteó como una matriz de ocho filas y ocho columnas en la que los jugadores realizan los cambios permitidos por las restricciones del reglamento de Otelo. En el contexto de Python, esta matriz se traduce como una lista que contiene ocho listas que, a su vez, contienen ocho números cada una y representan las ocho filas en las que se divide el tablero, siendo cada número almacenado en esa lista de listas la representación de lo que hay en cada casilla (no hay ficha, una ficha blanca o una ficha negra), haciendo un total de sesenta y cuatro números. En esta implementación de Otelo sólo hay tres posibles números que representan el color de la ficha que hay en cada casilla del tablero, el resultado final de la partida y, adicionalmente, se utilizan también para saber el turno actual de una partida en curso. Estos números son: el cero (no hay ficha en la casilla o el juego ha terminado en empate), el uno (hay una ficha blanca en una casilla, el jugador de las blancas ha ganado la partida o es el turno de dicho jugador) y el dos (hay una ficha negra en la casilla, el jugador de las blancas ha ganado la partida o es el turno de dicho jugador).

Sin embargo, esta lógica es la base que hay detrás de la interfaz gráfica mediante la cual una persona puede jugar a esta implementación de Otelo, una interfaz gráfica creada gracias a los recursos de la librería Pygame. Si bien es cierto que, entre las alternativas posibles para la implementación de este juego estaba la opción de crear un script que permitiera jugar a Otelo mediante texto (es decir, representando directamente la lógica que se ha mencionado anteriormente), valoramos más la opción de hacer uso de la librería Pygame, pues no sólo el juego sería mejor desde el punto de vista visual (aunque, dentro de los mínimos a cumplir en el proyecto, no desde el punto de vista técnico), sino que también se concibió el uso de Pygame como una manera muy intuitiva de empezar la implementación del proyecto partiendo desde cero.

Se construyó el script otelo.py para que, al ser ejecutado, inicializase una pantalla de 640 píxeles de alto y ancho en la que el color de fondo es el verde y varias líneas negras dividen las casillas cuadradas de 80 píxeles de lado. Adicionalmente, se representan las fichas como círculos blancos o negros de diámetro un poco menor al lado de las casillas y de centro situado en el centro de la casilla en la que está cada uno. Este script, como todos los videojuegos creados a partir de Pygame, contiene un bucle principal que comprueba si el usuario ha cerrado la ventana, comprueba si el usuario ha hecho clic en alguna casilla, etc. En este bucle, dentro de la implementación inicial que se trata en este apartado, sencillamente se comprueba en qué lugares de la pantalla el usuario hace clic para colocar la ficha correspondiente (en caso de que la disposición del tablero lo permita) y se alternan los turnos ("turno = 1" y "turno = 2"). Cada vez que ocurre esto, se pinta el tablero en base a la nueva matriz que lo representa y que contiene los cambios provocados por la acción del jugador (tanto la posición de la nueva ficha colocada como las fichas volteadas).

El proceso para hacer esto consiste en lo siguiente: primero

es necesario convertir el píxel en el que el usuario ha hecho clic en una casilla. Para ello, se hace una división entera de cada coordenada del píxel entre la longitud en píxeles de cada casilla. Posteriormente, teniendo las coordenadas de la casilla, se comprueba si es posible colocar ficha en la casilla seleccionada. Si no lo es, no ocurre nada, pero si lo es, se coloca la ficha en la casilla y se voltean las fichas que deberán darse la vuelta tras la aparición de la nueva ficha. Para realizar tanto la comprobación de movimientos disponibles como el volteo de las fichas, se utilizan funciones que comprueban si existen fichas que serán volteadas tras la colocación de la nueva ficha (aunque, anteriormente, se comprueba si la matriz contiene un cero en el lugar seleccionado) y se guardan las coordenadas de dichas fichas para, en la modificación del tablero previa al cambio de turno, cambiar el color de las fichas a voltear; en caso de que el turno actual sea igual a 1 (blancas) se asigna el valor 1 a las celdas de la matriz equivalentes a las casillas afectadas y en caso de que el turno actual sea el del jugador de las fichas negras (igual a 2), se hace exactamente lo mismo pero asignando un 2 a las celdas correspondientes.

Por último, es necesario recalcar varios puntos: primero, antes del proceso anteriormente descrito, se comprueba si el jugador tiene movimientos disponibles. Si no los tiene, se cambia el turno. Segundo (como ya se ha mencionado) en esta implementación no existe ningún agente, el usuario coloca fichas tanto para el jugador de las blancas como para el jugador de las negras. Y tercero, es necesario recalcar que todo lo explicado es fundamental para el desarrollo del agente MCTS, pues implementan la lógica que seguirá el agente para poder jugar correctamente a Otelo.

### *B. Motor MCTS con UCT*

### *C. Agente básico generador de datos*

### *D. Diseño de la red neuronal*

### *E. Entrenamiento de la red neuronal*

### *F. Agente adversario*

## IV. Pruebas y experimentación

Explica qué datos has obtenido, cómo se comporta el agente, etc. Puedes incluir tablas, gráficos, o descripciones.

## V. Conclusiones

Resume lo aprendido, dificultades enfrentadas, mejoras posibles.