MAESTRÍA EN INTELIGENCIA ARTIFICIAL

REPORTE DE TRABAJO PRÁCTICO

TP1: MONTE CARLO ES

Autor:
Maximiliano Torti

Docente:
Miguel Augusto Azar

Este trabajo fue realizado en abril de 2025.

Índice general

1 Desarrollo

1.1. Introducción

En el presente trabajo práctico, se desarrolla la implementación de los algoritmos Monte Carlo ES y Q-Learning en dos modelos que, luego de ser entrenado, son capaces de jugar al tres en linea. El objetivo es poder evaluar la implementación de los algoritmo y compararlos, en sus beneficios y defectos, y en su rendimiento en un juego simple y reducido.

Para lograrlo, se implementaron en lenguaje Python las clases *Board*, y *TicTacToe* que modelan el tablero y las reglas de juego respectivamente (entorno), y la clase abstracta *Player* que da soporte a las operaciones de los posibles jugadores (agentes). Heredando de *Player*, se crearon las clases *UserPlayer*, *BotPlayer*, *MonteCarloEsControl* y *QLearning* que permiten jugadores humanos, bots aleatorios, modelos guiados por el algoritmo de Monte Carlo ES y modelos guiados por el algoritmo de Q-Learning respectivamente.

Un script principal facilita la ejecución de los casos de uso principales: comenzar un juego entre 2 jugadores o entrenar un modelo utilizando uno de los 2 algoritmos disponibles.

1.2. Entorno

El juego comienza luego de definir 2 tipos de jugadores entre las 3 posibilidades, instanciar la clase *TicTacToe* y ejecutar el método *start*. En ese instante se imprime en pantalla un tablero de tres en línea como se muestra en la figura 1.1.

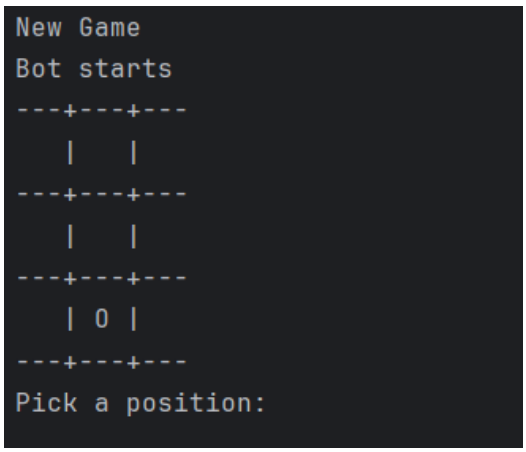

FIGURA 1.1. Tablero de tres en linea.

El comportamiento inicial dependerá de los jugadores seleccionados como jugador 1 y 2 y del azar, que le asignará a cualquiera de los anteriores la jugada inicial. Cada vez que un jugador realice una jugada valida, el tablero se refrescará y la

partida continuará hasta que haya un ganador o se produzca un empate (tablero lleno). En este momento se detiene el juego y se informa el resultado, como se muestra en la figura 1.2.

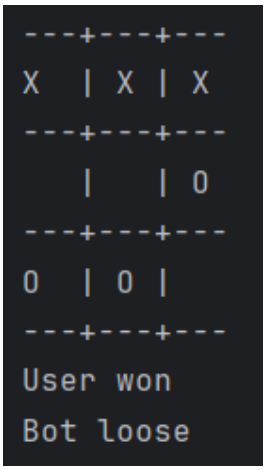

FIGURA 1.2. Juego finalizado.

1.3. Agentes

Cuando un jugador es humano y le llega su turno, este debe introducir por teclado un número del 0 al 8, correspondientes a las posiciones que se muestran en la figura 1.3. Si el jugador ingresa una posición inválida o una posición ya tomada, se deberá volver a introducir la jugada.

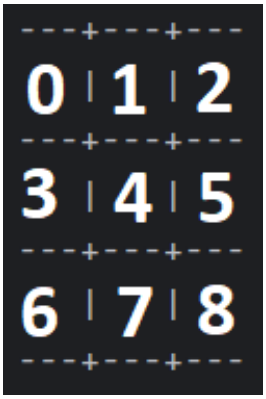

FIGURA 1.3. Posiciones en el tablero.

El bot aleatorio elige, en forma automática y aleatoria, cualquier posición disponible en el tablero. Este no sigue ninguna estrategia y resulta útil únicamente para probar el juego y para realizar el entrenamiento de los modelos disponibles.

Los agentes con modelos entrenables realizan movimientos en forma automática siguiendo una política aprendida. En cada turno observan el estado actual del tablero, y toman la acción que se evaluó como mejor opción para ese estado durante el entrenamiento.

1.4. Entrenamiento

1.4.1. Monte Carlo ES

Para entrenar el agente de Monte Carlo ES, se ejecutaron múltiples juegos consecutivos de tres en línea donde participaban el agente en entrenamiento y bots aleatorios. Dentro del agente de Monte Carlo ES se implementó el algoritmo de la figura 1.4, con los siguientes detalles:

- $A(s)$ contiene las acciones posibles, que son jugar las posiciones del 0 al 8.
- S contiene los estados posibles del tablero. El tablero de tres en línea tiene en total $3^9 = 19{,}683$ estados diferentes, la mayoría de los cuales son ilegales.
- Dado que resulta costoso generar las políticas iniciales aleatorias para todos los estados del tablero y también filtrar solo los estados posibles (alrededor de 5.478), se decidió comenzar con $\pi(s)$ y $Q(s, a)$ vacíos e inicializarlos en forma aleatoria a medida que el agente se encontraba con estados que nunca había visto antes.
- Para el comienzo exploratorio de Monte Carlo, se ejecutaron en el tablero un número aleatorio de jugadas aleatorias alternadas entre jugador 1 y 2 previo a comenzar el episodio. Cabe destacar que se realizó un chequeo del estado del tablero luego de estas inicializaciones para verificar la validez.
- Como recompensa se asignó -0.1 por cada jugada que no concluyera en el final del juego, -10 a una jugada perdedora, -2 a una jugada de empate y 10 a una jugada ganadora.
- Adicionalmente se limito el número de jugadas del agente en 20 y se agrego una penalización de -50 si se supera este valor. De esta forma se prioriza aprender a jugar posiciones válidas.
- Se fijo un factor de descuento de 0.9.
- El entrenamiento se repitió durante 500.000 episodios, demorando aproximadamente una hora.
- Los valores anteriores se obtuvieron en forma empírica, a partir de varios entrenamientos y analizando los resultados obtenidos.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
  $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
  $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
  Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$

FIGURA 1.4. Algoritmo de Mote Carlo ES.

1.4.2. Q-Learning

Para entrenar el agente de Q-Learning, se ejecutaron múltiples juegos consecutivos de tres en línea donde participaban el agente en entrenamiento y bots aleatorios. Dentro del agente de Q-Learning se implementó el algoritmo de la figura 1.5, con los siguientes detalles:

- $A(s)$ contiene las acciones posibles y S contiene los estados posibles del tablero, al igual que en el caso anterior.
- Q-Learning no necesita inicio aleatorio del estado del ambiente, por lo que los diferentes episodios de juego empezaron con el tablero vacío.
- Las recompensas se mantuvieron como en el caso anterior: -0.1 por cada jugada que no concluyera en el final del juego, -10 a una jugada perdedora, -2 a una jugada de empate, 10 a una jugada ganadora y -50 por superar las 20 jugadas.
- Se fijo un α (tasa de aprendizaje) de 0.1, un γ (factor de descuento) de 0.9 y un ϵ (factor de exploración) de 0.2.
- El entrenamiento se repitió durante 500.000 episodios, demorando aproximadamente 5 minutos.

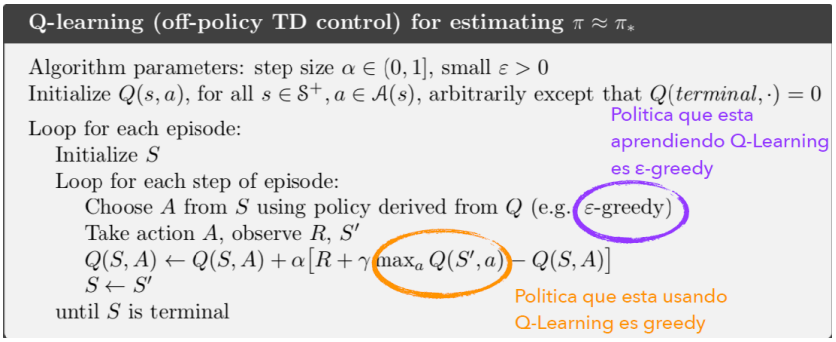

FIGURA 1.5. Algoritmo de Q-Learning.

1.5. Resultados

1.5.1. Monte Carlo ES

Luego de entrenar el agente Monte Carlo ES se obtuvo la curva de recompensa de la figura 1.6 y la curva de éxito (juegos ganados) de la figura 1.7.

En la figura 1.6 se observa que la media de la recompensa aumenta lentamente con el paso de los episodios desde un valor de 0 hasta aproximadamente 5, lo que nos indica que el modelo está entrenándose correctamente. También, observamos en la figura 1.7 que el modelo alcanza un porcentaje de juegos ganados de aproximadamente 0.7, es decir, el modelo gana el 70 % de las partidas contra un bot aleatorio, por lo que su política es mejor que elegir posiciones al azar.

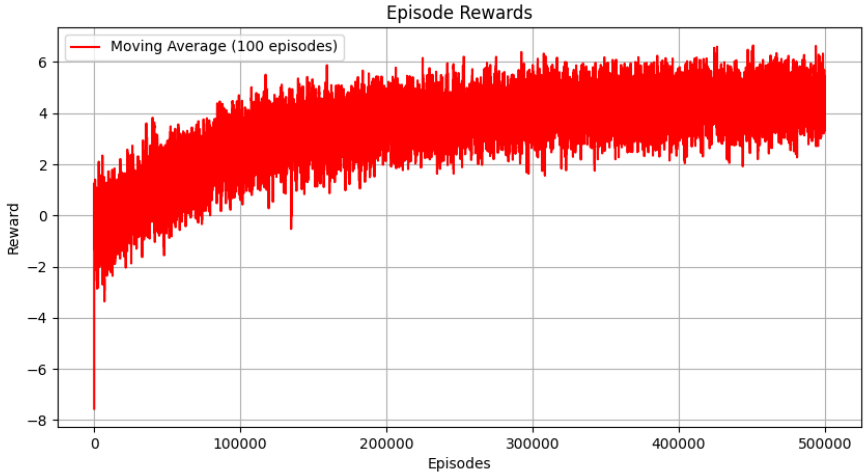

FIGURA 1.6. Recompensa por episodio para algoritmo de Monte Carlo ES, suavizado con una media móvil de 100 episodios.

FIGURA 1.7. Curva de éxito (porcentaje de juegos ganados en los últimos 100 episodios) para algoritmo de Monte Carlo ES.

Al jugar en contra del modelo, se observa que se obtuvo un modelo capaz de identificar las estrategias principales del tres en línea. Por ejemplo, si el modelo comienza el juego, este toma siempre el centro del tablero como se observa en la figura 1.8. También, si tiene posibilidad de ganar, toma la posición ganadora; o por el contrario, si el contrincante tiene posibilidad de ganar, intenta bloquear esa posibilidad, como se observa en las figuras 1.9 y 1.10.

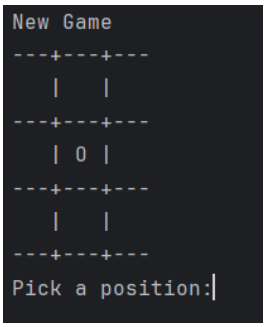

FIGURA 1.8. Agente tomando el centro.

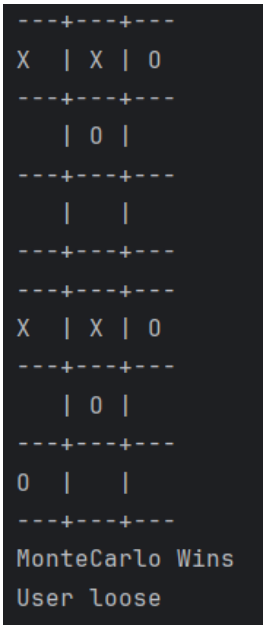

FIGURA 1.9. Agente tomando la posición ganadora.

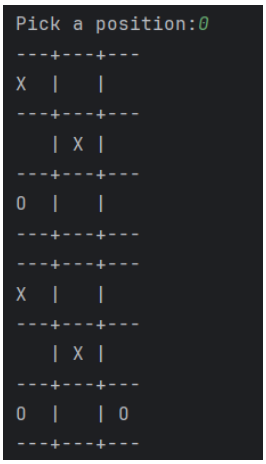

FIGURA 1.10. Agente bloqueando la posición ganadora del humano.

Lo sorprendente es que el modelo también aprende estrategias avanzadas, Por ejemplo, si teniendo tomado el centro el usuario elige posición 1, 3, 5 o 7, la victoria esta asegurada con el patrón de la figura 1.11. A la vez, si el jugador humano

intenta ejecutar la misma estrategia, el agente de Monte Carlo ES la evita eligiendo las esquinas como su primer movimiento.

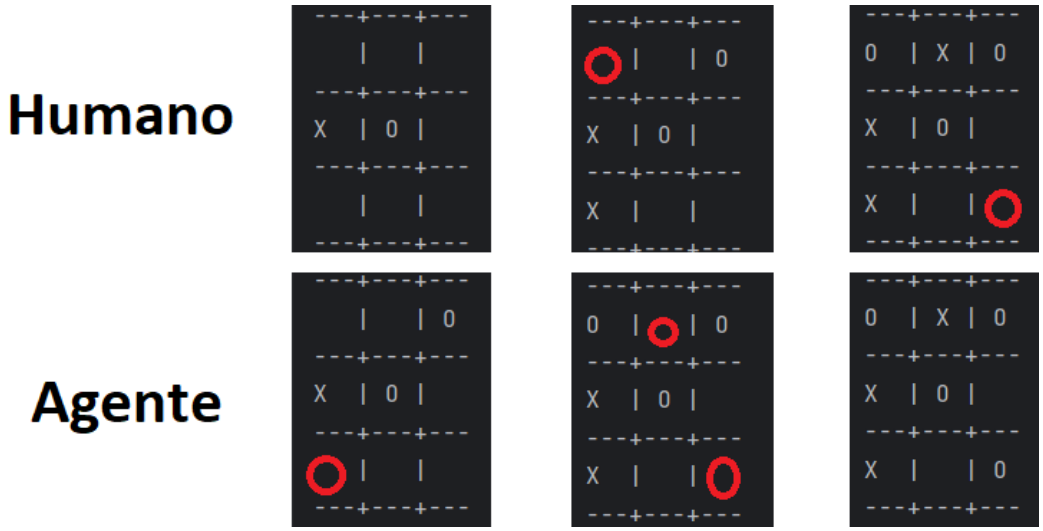

FIGURA 1.11. Agente ejecutando estrategia que asegura la victoria. En rojo posiciones ganadoras.

Como contrapartida el agente no es perfecto, cuando se intenta ejecutar sobre el mismo otra estrategia ganadora mostrada en la figura 1.12, este falla eligiendo repetidamente una posición ya tomada. Esto se debe a que en el entrenamiento no se exploró el espacio de estados y acciones lo suficiente como para encontrar una mejor política.

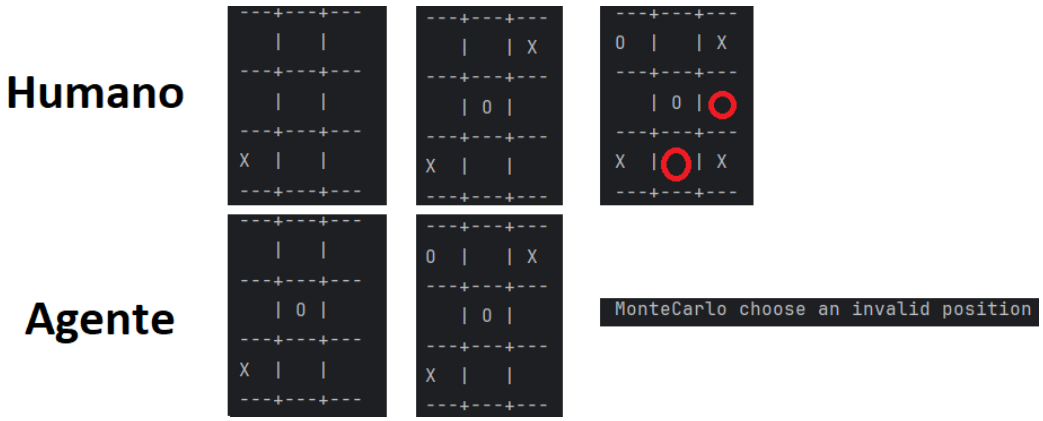

FIGURA 1.12. Agente fallando en tomar posiciones válidas.

1.5.2. Q-Learning

Luego de entrenar el agente Q-Learning se obtuvo la curva de recompensa de la figura 1.13 y la curva de éxito de la figura 1.14.

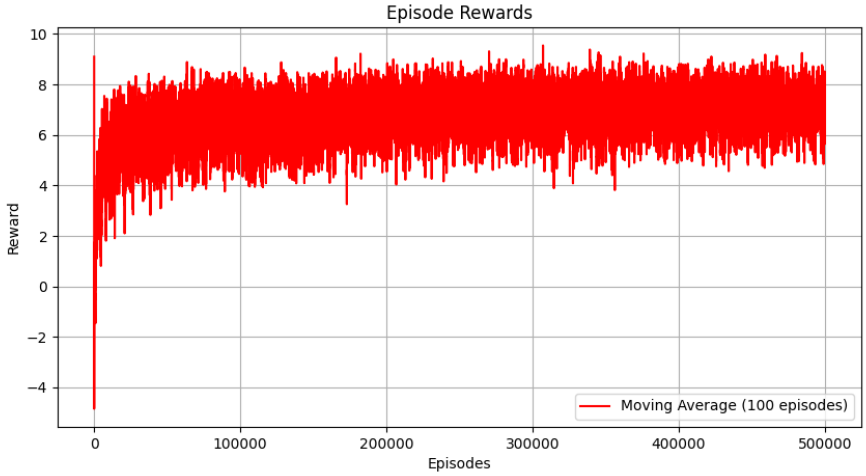

FIGURA 1.13. Recompensa por episodio para algoritmo de Q-Learning, suavizado con una media móvil de 100 episodios.

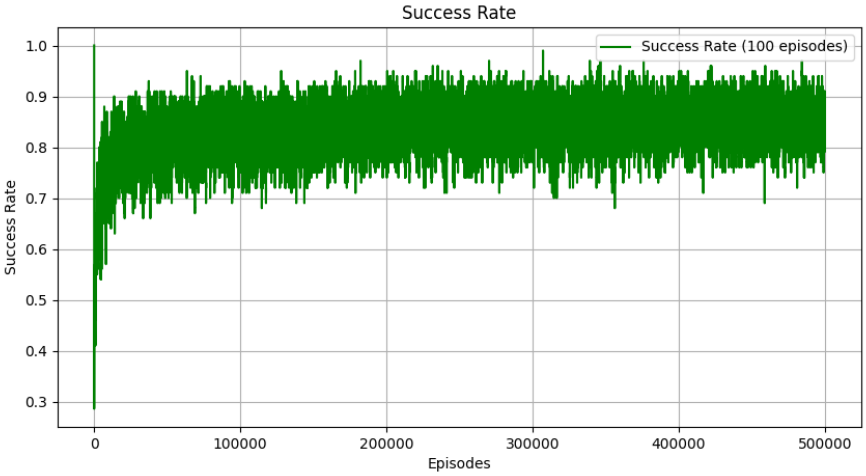

FIGURA 1.14. Curva de éxito (porcentaje de juegos ganados en los últimos 100 episodios) para algoritmo de Q-Learning.

En primer lugar se observa que el entrenamiento también converge y rápidamente, con solo 100.000 episodios el modelo ya estaría entrenado. Luego se observa que los valores de recompensa y de tasa de éxito alcanzados son mayores respecto al modelo anterior, aunque esto no es indicativo de mayor calidad ya que Monte Carlo ES requiere de inicios aleatorios que pueden llevar a estados que terminan siempre en jugadas perdedoras o empate.

Al jugar en contra del modelo, se obtienen los mismos resultados que en el modelo anterior, se observa que se aprenden estrategias básicas y también avanzadas. A diferencia del anterior, el modelo de Q-Learning parece ser mas consistente en sus resultados, teniendo menos situaciones donde elige posiciones ya tomadas.

1.6. Conclusiones

Ambos algoritmos mostraron resultados muy buenos en el desafío planteado, no solamente aprendiendo los movimientos básicos, sino también aprendiendo estrategias avanzadas lo cual fue inesperado.

Comparando, Q-Learning mostró un mejor desempeño con una convergencia en menor cantidad de episodios de entrenamiento, gracias a su aprendizaje off-policy y al algoritmo de exploración ϵ-greedy. Monte Carlo ES, al aprender solo en base a la política ejecutada y requerir de comienzos aleatorios para la exploración demora una mayor cantidad de episodios en explorar todos los pares estado-acción posibles. Más aún, los comienzos exploratorios de Monte Carlo ES agregan complejidad a la implementación, ya que requiere de código adicional que permita fijar el ambiente en un estado inicial determinado.

Como puntos débiles de ambos algoritmos, aún en un juego simple como el tres en línea, se deben mantener estructuras (como la tabla Q) para todos los estados posibles del tablero, que en este caso es del orden de los miles, lo que implica un costo computacional moderado. Otro punto débil de ambos algoritmos es que no pueden "extrapolar" conocimiento, es decir aprovechar lo aprendido para un estado para resolver estados similares, que en este caso sería una cualidad deseable ya que el tres en línea es un juego simétrico si se rota el tablero.

1.7. Trabajo futuro

Como trabajo futuro se plantea enfrentar ambos algoritmos en un entrenamiento a la par, para observar el comportamiento. Lo esperable sería que el modelo de Q-Learning comience ganando el mayor porcentaje de partidas y que, luego de muchos episodios, la mayoría de las partidas terminen en empate.

A Script principal

El script principal del proyecto contiene un procesador de líneas de comando que facilita la ejecución de los diferentes casos de uso. Los usos básicos del procesador se detallan a continuación:

- Mostrar la ayuda.

```
$ python src/main.py --help
```

- Jugar un humano contra el bot aleatorio.

```
$ python src/main.py play --games=<number_of_games> --bot
```

- Jugar un humano contra un modelo.

```
$ python src/main.py play --games=<number_of_games> \
    --model-file=<model-file>
```

- Entrenar un nuevo agente de Monte Carlo.

```
$ python src/main.py train --model=monte-carlo \
    --episodes=<number_of_episodes>
```

- Entrenar un nuevo agente de Q-Learning.

```
$ python src/main.py train --model=q-learning \
    --episodes=<number_of_episodes>
```

```
> python src/main.py --help
This script allows you to play or train a Tic Tac Toe game using different players.
You can play against a bot, a Monte Carlo model, or another human player.
It also allows you to train a Monte Carlo or a Q-learning model using reinforcement learning.

Usage:
    main.py play --games=<number_of_games> [--bot] [--model-file=<model-file>] [--human]
    main.py train --model=<model-type> --episodes=<number_of_episodes>
    main.py train2 --episodes=<number_of_episodes>

Options:
    -h --help                       Show this screen.
    --bot                           Play against a bot.
    --episodes=<number_of_episodes> Number of episodes to train the Monte Carlo model.
    --games=<number_of_games>       Number of games to play.
    --human                         Play against another human player.
    --model-file=<model-file>       Play against a trained model.
    --model=<model_type>            The type of model to train. Valid options are: monte-carlo or q-learning.
```

FIGURA A.1. Uso del script principal por línea de comando.

B Cobertura y calidad de código

Dada la complejidad del entorno, los agentes y los algoritmos de Monte Carlo ES y Q-Learning, se implementaron test unitarios de las diferentes clases a fin de verificar el comportamiento de los diferentes métodos. Estos test se pueden ejecutar mediante el comando:

```
$ pytest
```

También se puede analizar la cobertura de código lograda con los test unitarios desarrollados mediante el siguiente comando:

```
$ coverage run -m pytest
$ coverage report
```

Name	Stmts	Miss	Cover
src__init__.py	0	0	100%
src\game__init__.py	0	0	100%
src\game\board.py	38	0	100%
src\game\monte_carlo.py	112	0	100%
src\game\players.py	52	0	100%
src\game\q_learning.py	112	0	100%
src\game\tic_tac_toe.py	57	0	100%
src\game\utils.py	13	0	100%
tests__init__.py	0	0	100%
tests\game__init__.py	0	0	100%
tests\game\test_board.py	64	0	100%
tests\game\test_monte_carlo.py	146	0	100%
tests\game\test_players.py	67	0	100%
tests\game\test_q_learning.py	145	0	100%
tests\game\test_tic_tac_toe.py	132	0	100%
tests\game\test_utils.py	18	0	100%
TOTAL	956	0	100%

FIGURA B.1. Reporte de cobertura.

Para mantener una buena calidad de código, el repositorio cuenta con documentación extensiva y un archivo README.md donde se detalla el uso del proyecto. Adicionalmente se ejecutaron análisis estáticos de calidad de código mediante la herramienta SonarLint y se resolvieron las malas prácticas y los problemas de seguridad encontrados.

```
[2025-04-07T23:11:31.191] [sonarlint-analysis-engine] INFO sonarlint - Index files
[2025-04-07T23:11:31.191] [Report about progress of file indexation] INFO sonarlint - 7 files indexed
[2025-04-07T23:11:31.206] [sonarlint-analysis-engine] WARN sonarlint - No workDir in SonarLint
[2025-04-07T23:11:31.222] [sonarlint-analysis-engine] INFO org.sonar.plugins.python.Scanner - Starting rules execution
[2025-04-07T23:11:31.222] [rules execution progress] INFO org.sonarsource.analyzer.commons.ProgressReport - 6 source files to be analyzed
[2025-04-07T23:11:31.362] [rules execution progress] INFO org.sonarsource.analyzer.commons.ProgressReport - 6/6 source files have been analyzed
[2025-04-07T23:11:31.362] [sonarlint-analysis-engine] INFO org.sonar.plugins.python.PythonScanner - The Python analyzer was able to leverage cached data from previous analyses
[2025-04-07T23:11:31.362] [Progress of the Docker analysis] INFO org.sonarsource.analyzer.commons.ProgressReport - 0 source files to be analyzed
[2025-04-07T23:11:31.362] [Progress of the Docker analysis] INFO org.sonarsource.analyzer.commons.ProgressReport - 0/0 source files have been analyzed
[2025-04-07T23:11:31.362] [sonarlint-analysis-engine] INFO org.sonar.plugins.common.TextAndSecretsSensor - Available processors: 8
[2025-04-07T23:11:31.362] [sonarlint-analysis-engine] INFO org.sonar.plugins.common.TextAndSecretsSensor - Using 8 threads for analysis.
[2025-04-07T23:11:31.378] [sonarlint-analysis-engine] INFO org.sonar.plugins.common.TextAndSecretsSensor - Analyzing all except non binary files
[2025-04-07T23:11:31.378] [Progress of the text and secrets analysis] INFO org.sonar.plugins.common.MultiFileProgressReport - 7 source files to be analyzed
[2025-04-07T23:11:31.378] [Progress of the text and secrets analysis] INFO org.sonar.plugins.common.MultiFileProgressReport - 7/7 source files have been analyzed
[2025-04-07T23:11:31.378] [sonarlint-analysis-engine] INFO sonarlint - Analysis detected 0 issues and 0 Security Hotspots in 203ms
```

FIGURA B.2. Análisis estático de código.

C Repositorio

El repositorio con el código fuente, los test unitarios y archivos adicionales de utilidad se encuentra disponible en el siguiente link:

https://github.com/maxit1992/MIA_RL1/tree/main/tp1