GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN EJERCICIO COOPERATIVO CON HEARTHSTONE PARA ESTUDIANTES DE IA

Autor: JESÚS LÓPEZ LÓPEZ - NEIRA

Directores: LUIS FRANCISCO SÁNCHEZ MERCHANTE

Madrid, 5 de julio de 2023

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Diseño e implementación de un ejercicio cooperativo

con Hearthtstone para estudiantes de IA.

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022/23 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

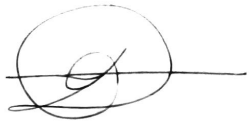
El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.:  Fecha: 05/07/2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Firmado digitalmente por Luis Francisco Sánchez Merchante
Fecha: 2023.07.05 11:04:07 +02'00'

Fdo.: Luis Francisco Sánchez Merchante Fecha: 05/07/2023

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN EJERCICIO COOPERATIVO CON HEARTHSTONE PARA ESTUDIANTES DE IA

Autor: JESÚS LÓPEZ LÓPEZ - NEIRA

Directores: LUIS FRANCISCO SÁNCHEZ MERCHANTE

Madrid, 5 de julio de 2023

DISEÑO E IMPLEMENTACIÓN DE UN EJERCICIO COOPERATIVO CON HEARTHSTONE PARA ESTUDIANTES DE IA

Autor: López López - Neira, Jesús.
Director: Sánchez Merchante, Luis Francisco.
Entidad Colaboradora: ICAI – (Universidad Pontificia Comillas).

RESUMEN DEL PROYECTO

Este proyecto expone un estudio holístico de la implementación de un desafío cooperativo de inteligencia artificial usando el videojuego Hearthstone. En el texto, se contemplan todas las fases desde la investigación y diseño académico hasta la implementación con la adaptación de simuladores, la creación de nuevos entornos y la aplicación de algoritmos de aprendizaje por refuerzo.

Los resultados demuestran un éxito en la creación del entorno y los métodos de estudio para estudiantes universitarios. En cuanto a la implementación del ejercicio, este texto, solo supone un primer acercamiento que, marca las bases, y trata de dirigir tanto al profesorado como a los alumnos en la exploración del entorno.

Palabras clave: *Hearthstone*, *Inteligencia Artificial*, *Gamificación, Aprendizaje por Refuerzo, Videojuegos*

1. Introducción

El Desarrollo de los estudiantes es el objetivo primordial de la actividad docente. Con el desarrollo de nuevas tecnologías, se ha comprobado que existen nuevas técnicas educativas de gran impacto para el alumnado. Una de las metodologías con mayor impacto es la gamificación del entorno académico (Dicheva, 2015).

Por otro lado, la inteligencia artificial (IA) en la que se basa este estudio tiene una historia fuertemente ligada a los videojuegos y el concepto de la gamificación. A lo largo de los últimos 30 años hemos presenciado modelos que han batido récords y ganado a numerosos campeones en juegos de mesa y videojuegos ((Hsu, 2002); (Silver, 2016); (Vinyals, 2019)).

En los últimos años, el videojuego Hearthstone ha ganado una importante popularidad y ha sido objeto de competiciones en el ámbito de la inteligencia artificial (Dockhorn, 2019). Este interés prevalece por su alto grado de complejidad y su componente estocástica.

2. Definición del Proyecto

Con este objetivo en mente, se ha desarrollado la manera óptima de implementar el videojuego Hearthstone para esta herramienta con la búsqueda y el análisis de los simuladores existentes, puesto que, Hearthstone es un videojuego online dónde no se pueden aplicar bots sin autorización.

Además, se ha creado un entorno sencillo y adaptado a las necesidades de un alumno que pueda ser lego en la materia. Más allá, del desarrollo del entorno, se propone una guía y metodología de estudio y se estudian diversos agentes y parámetros del entorno.

3. Descripción del modelo/sistema/herramienta

Al comienzo del proyecto, se ha realizado un profundo estudio de los beneficios de la gamificación, así como, de ejemplos y trabajos previos en el ámbito de la inteligencia artificial y Hearthstone.

Posteriormente se ha comparado la funcionalidad de varios simuladores, principalmente Fireplace y Sabberstone como base para la creación de un entorno. Aunque, Sabberstone ha sido objeto de más proyectos de IA se ha optado por el uso de un simulador en Python dónde resulte más sencillo para el alumno el lenguaje de programación y la aplicación de algoritmos mediante librerías como Stable Baselines.

Tras la elección del simulador y la librería de apoyo para el desarrollo de agentes ha sido preciso aunar las tecnologías en un entorno propio mediante el desarrollo de funciones necesarias en los procesos de aprendizaje por refuerzo. Para la estandarización de estas funciones y, de esta forma, conseguir su implementación en una amplia gama de algoritmos sea usado la librería Gymnasium para crear el entorno.

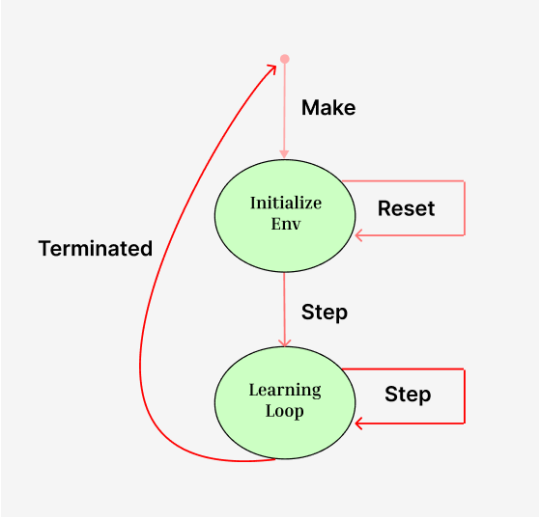


Figura 1. Diagrama del flujo de un entorno Gymnasium.

Para el testeo y la exploración del entorno se han implementado una variedad de agentes sencillos que han permitido la investigación del entorno; demonstración de la aplicación de distintos algoritmos de aprendizaje por refuerzo, parametrización de argumentos, creación de agentes que sirvan de base para la competición, la proposición de nuevas técnicas y objetos de estudio dentro del entorno.

Por último, se ha desarrollado una guía docente y una metodología de estudio para los alumnos y profesores que busquen la implementación de la enseñanza gamificada en sus universidades o de forma autodidáctica.

4. Resultados

En los resultados del proyecto se han discutido los distintos algoritmos y técnicas empelados. Mediante la parametrización de argumentos como la puntuación se ha conseguido una ratio de victorias que ha ascendido desde los primeros modelos en un 52,7 % de victorias hasta un 58,2%.

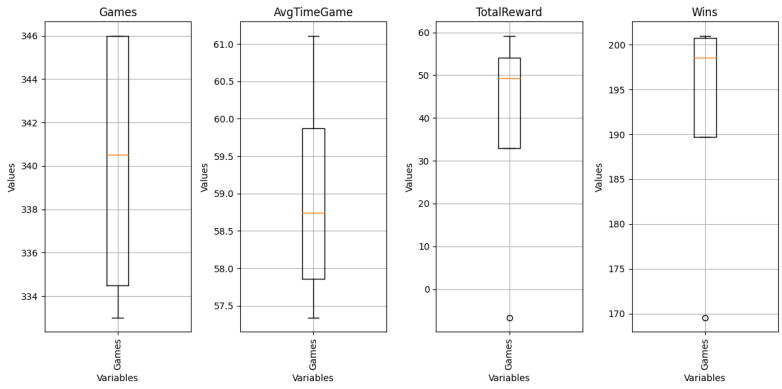


Figura 2. Diagrama de cajas de un agente (Type, Complex(0.0005), Classic) – 58,2% de Win Rate

Se ha analizado la capacidad de un bot aleatorio que es capaz de ganar partidas con una ratio de victorias del 60,1%. Este resultado deja palpable los distintos niveles de complejidad del Hearthstone dónde conviven estrategias esenciales y básicas sencillas ejecutar y otras más complejas que pueden decantar la partida hacia los jugadores. Se estima, por tanto, que los algoritmos desarrollados en este proyecto pueden beneficiarse de espacios de observación de menor complejidad que permitan, inicialmente, el aprendizaje de las estrategias esenciales.

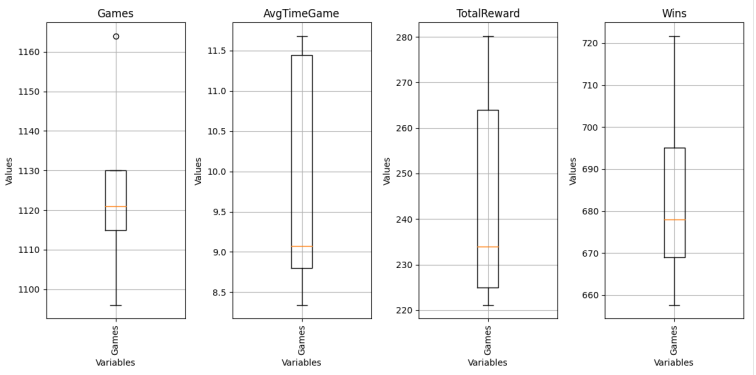


Figura 3. Diagrama de cajas de un bot aleatorio.

5. Conclusiones

Con el desarrollo de este trabajo se pretende fomentar a las universidades a utilizar la metodología de gamificación de la enseñanza y, aportar, herramientas gratuitas y documentadas abiertas a la comunidad, con el objetivo de ser accesible a cualquier estudiante e identidad. También, supone un nuevo formato para la academia en el videojuego Hearthstone que se ha visto en auge los últimos años por el interés que aportan las observaciones incompletas y la estocástica de las acciones.

6. Referencias

[1] Dicheva, D. a. (2015). *Gamification in education: A systematic mapping study. Journal of educational technology & society, 75--88*.

[2] Hsu, F.-H. (2002). *Behind Deep Blue: Building the computer that defeated the world chess champion.* Princeton University Press.

[3] Silver, D. a. (2016). *Mastering the game of Go with deep neural networks and tree search. nature, 484--489.*

[4] Vinyals, O. a. (2019). *Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 350--354.*

[5] Dockhorn, A. a. (2019). *Introducing the hearthstone-ai competition. arXiv preprint arXiv:1906.04238.*

DESING AND IMPLEMENTATION OF A GROUP CHALLENGE USING HEARTHSTONE FOR AI UNDERGRADUATE STUDENTS

Author: López López-Neira, Jesús.
Supervisor: Sánchez Merchante, Luis Francisco.
Collaborating Entity: ICAI – (Universidad Pontificia Comillas).

ABSTRACT

This project presents a holistic study of the implementation of an artificial intelligence cooperative challenge using the Hearthstone video game. In the text, all phases from academic research and design to implementation with the adaptation of simulators, the creation of new environments and the application of reinforcement learning algorithms are covered.

The results demonstrate success in the creation of the environment and study methods for university students. As for the implementation of the exercise, this text is only a first approach that lays the groundwork and tries to guide both teachers and students in the exploration of the environment.

Keywords: Hearthstone, Artificial Intelligence, Gamification, Reinforcement Learning, Videogames.

1. Introduction

Student development is the primary objective of the teaching activity. With the development of new technologies, it has been proven that there are new educational techniques with great impact on students. One of the methodologies with the greatest impact is the gamification of the academic environment (Dicheva, 2015).

On the other hand, artificial intelligence (AI) on which this study is based has a history strongly linked to video games and the concept of gamification. Over the last 30 years, we have witnessed record-breaking models and winning numerous champions in board games and video games ((Hsu, 2002); (Silver, 2016); (Vinyals, 2019)).

In recent years, the video game Hearthstone has gained significant popularity and has been the subject of competitions in the field of artificial intelligence (Dockhorn, 2019). This interest prevails because of its high degree of complexity and its stochastic component.

2. Project definition

With this objective in mind, we have developed the optimal way to implement the Hearthstone video game for this tool by searching and analyzing existing simulators, since Hearthstone is an online video game where bots cannot be applied without authorization.

In addition, a simple environment has been created and adapted to the needs of a student who may be a layman in the field. Beyond the development of the environment, a

study guide and methodology is proposed and various agents and parameters of the environment are studied.

3. Model/system/tool description

At the beginning of the project, an in-depth study of the benefits of gamification was carried out, as well as examples and previous work in the field of artificial intelligence and Hearthstone.

Subsequently, the functionality of several simulators was compared, mainly Fireplace and Sabberstone as a basis for the creation of an environment. Although Sabberstone has been the subject of more AI projects, we have opted for the use of a Python simulator where it is easier for the student to learn the programming language and the application of algorithms through libraries such as Stable Baselines.

After choosing the simulator and the support library for the development of agents, it has been necessary to combine the technologies in a proprietary environment by developing the necessary functions in the reinforcement learning processes. For the standardization of these functions and, in this way, to achieve their implementation in a wide range of algorithms, the Gymnasium library was used to create the environment.

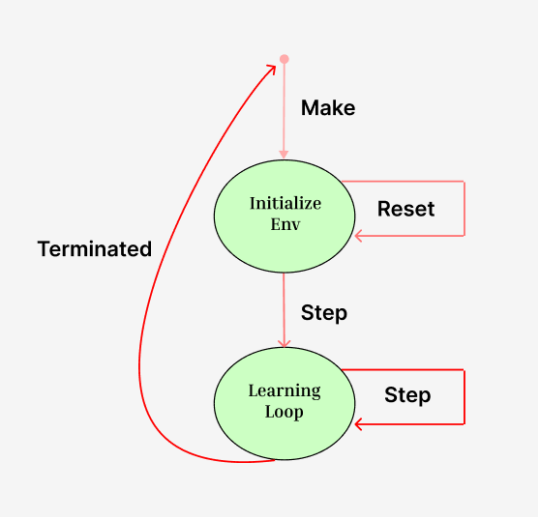


Figure 1. Functional flow diagram of a Gymnasium environment.

For the testing and exploration of the environment, a variety of simple agents have been implemented that have allowed the investigation of the environment; demonstration of the application of different reinforcement learning algorithms, parameterization of arguments, creation of agents that serve as a basis for competition, the proposal of new techniques and objects of study within the environment.

Finally, a teaching guide and a study methodology have been developed for students and teachers who are looking for the implementation of gamified teaching in their universities or in a self-taught way.

4. Results

In the results of the project the different algorithms and techniques used have been discussed. By parameterizing arguments such as scoring, a win ratio has been achieved that has risen from the first models at 52.7% to 58.2% of wins.

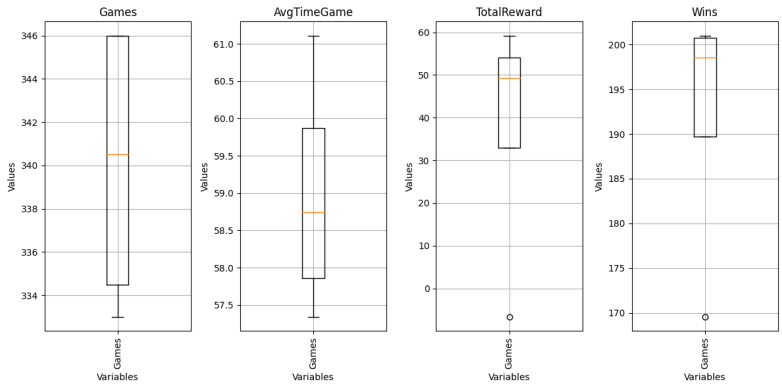


Figure 2. Boxplot visualization of agent (Type, Complex(0.0005), Classic) – 58,2% de Win Rate

Se ha analizado la capacidad de un bot aleatorio que es capaz de ganar partidas con una ratio de victorias del 60,1%. Este resultado deja palpable los distintos niveles de complejidad del Hearthstone dónde conviven estrategias esenciales y básicas sencillas ejecutar y otras más complejas que pueden decantar la partida hacia los jugadores. Se estima, por tanto, que los algoritmos desarrollados en este proyecto pueden beneficiarse de espacios de observación de menor complejidad que permitan, inicialmente, el aprendizaje de las estrategias esenciales.

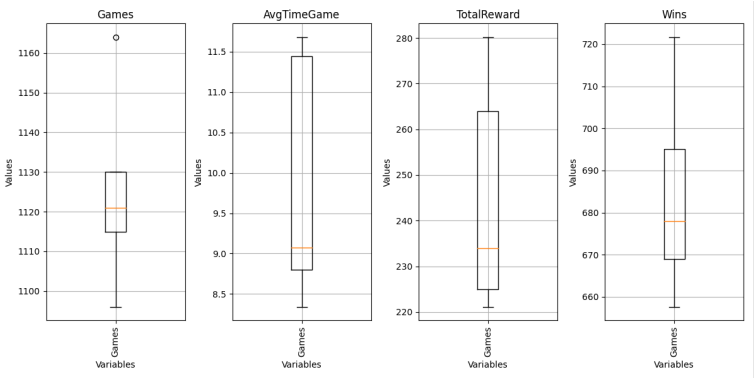


Figure 3. Boxplot visualization of a Random Agent.

5. Conclusions

With the development of this work, we intend to encourage universities to use the methodology of gamification of teaching and provide free and documented tools open to the community, with the aim of being accessible to any student and identity. Furthermore, it also represents a new format for academia in the Hearthstone video game, which has been booming in recent years due to the interest provided by incomplete observations and stochastic actions.

6. References

[1] Dicheva, D. a. (2015). *Gamification in education: A systematic mapping study. Journal of educational technology & society, 75--88*.

[2] Hsu, F.-H. (2002). *Behind Deep Blue: Building the computer that defeated the world chess champion.* Princeton University Press.

[3] Silver, D. a. (2016). *Mastering the game of Go with deep neural networks and tree search. nature, 484--489.*

[4] Vinyals, O. a. (2019). *Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 350--354.*

[5] Dockhorn, A. a. (2019). *Introducing the hearthstone-ai competition. arXiv preprint arXiv:1906.04238.*

Índice de la memoria

Índice de figuras

Índice de ecuaciones

Capítulo 1. INTRODUCCIÓN

Los videojuegos CCG (Collectible Card Games) presentan unas características interesantes para el desarrollo de algoritmos de inteligencia artificial. Los desafíos y competiciones de modelos fomentan el aprendizaje y permiten generar aplicaciones prácticas de la materia que se estudia.

Este trabajo propone diseñar un ejercicio colaborativo para el desarrollo de un bot para el juego Hearthstone. Esta tarea es especialmente interesante para desarrollar competencias en el ámbito de inteligencia artificial porque el diseño de un bot de estas características puede plantearse desde diferentes puntos de vista, desde el uso de algoritmos evolutivos, cadenas de Márkov, aprendizaje por refuerzo hasta algoritmos de Frequent Pattern Matching.

De la misma manera, la evaluación de los algoritmos implementados por los participantes en el desafío podría realizarse de manera explícita mediante la competición en un entorno de juego real. Permitiendo generar un ranking de victorias distribuyendo la puntación del ejercicio en función del éxito del bot. Por ello, aprovechando las plataformas existentes, en este trabajo, se pretende crear un desafío cooperativo para estudiantes que cursen asignaturas de inteligencia artificial.

1.1 MOTIVACIÓN DEL PROYECTO

En el contexto actual de la educación universitaria y el creciente uso de la inteligencia artificial (IA), la gamificación ha surgido como una poderosa herramienta pedagógica que promueve la participación de los estudiantes, el aprendizaje autónomo y la motivación intrínseca. La gamificación combina elementos y dinámicas propias de los juegos en

entornos educativos, con el objetivo de potenciar el compromiso, la colaboración y el logro de objetivos académicos.

La importancia de la gamificación en el ámbito educativo no puede subestimarse, ya que proporciona un enfoque innovador y efectivo para abordar los desafíos de la enseñanza y el aprendizaje en la era digital. Según estudios recientes ((Dicheva, 2015); (Burguillo, 2010)), la integración de elementos de juego en el entorno educativo ha demostrado impactos positivos en el rendimiento académico, la retención de conocimientos y la satisfacción de los estudiantes.

La aplicación de la gamificación en la educación universitaria brinda numerosos beneficios. En primer lugar, fomenta la participación activa de los estudiantes al convertir el proceso de aprendizaje en una experiencia interactiva y divertida. Los estudiantes se sienten motivados y comprometidos al tener objetivos claros, recibir recompensas y competir de manera saludable con sus compañeros.

Además, la gamificación promueve el aprendizaje autónomo al permitir que los estudiantes exploren, descubran y adquieran conocimientos de forma autodirigida. Los juegos educativos ofrecen oportunidades de práctica, retroalimentación inmediata y la posibilidad de aprender de los errores, lo que facilita la comprensión y el dominio de los conceptos.

La colaboración también se ve fortalecida mediante la gamificación. Los juegos educativos pueden fomentar el trabajo en equipo, el intercambio de ideas y la resolución de problemas de manera conjunta. Esto promueve habilidades sociales y mejora la capacidad de los estudiantes para comunicarse y colaborar de manera efectiva.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Con el propósito de mejorar la legibilidad y la comprensión de este proyecto, se presentarán a continuación las tecnologías que se han utilizado en el mismo, algunas de las cuales son exclusivas del ámbito de la programación y otras del ámbito de la seguridad.

2.1 PYTHON

Python es ampliamente reconocido como uno de los lenguajes de programación más populares y versátiles en la actualidad. Su utilidad para el desarrollo de proyectos es innegable, gracias a una serie de ventajas que ofrece a los programadores.

Una de las principales ventajas de Python es su legibilidad. Su sintaxis clara y concisa facilita la comprensión del código, lo que reduce el tiempo y esfuerzo necesarios para el desarrollo y mantenimiento de proyectos. Además, cuenta con una amplia variedad de bibliotecas y módulos de terceros, lo que acelera el proceso de desarrollo al permitir reutilizar código existente.

Python también destaca por su portabilidad y compatibilidad multiplataforma. Esto significa que un programa escrito en Python puede ejecutarse en diferentes sistemas operativos sin necesidad de realizar cambios significativos en el código. Esto resulta especialmente útil en entornos de desarrollo heterogéneos.

Además, Python es un lenguaje interpretado, lo que implica que el código se ejecuta línea por línea, lo que facilita la depuración y prueba de aplicaciones. Además, su naturaleza dinámica permite una mayor flexibilidad en tiempo de ejecución, lo que facilita la manipulación y transformación de datos.

Sin embargo, también existen algunas desventajas asociadas con el uso de Python. En comparación con lenguajes de programación compilados, como C++ o Java, Python puede ser más lento en términos de rendimiento. Esto se debe a su naturaleza interpretada y a que algunas operaciones requieren más tiempo de ejecución.

Otra desventaja es que, al ser un lenguaje de alto nivel, Python puede no ser la mejor opción para aplicaciones que requieren un control preciso sobre el hardware o donde el rendimiento extremo sea crucial.

2.2 *HEARTHSTONE*

Hearthstone es un juego de cartas en el que dos jugadores se enfrentan utilizando mazos de cartas personalizados. El objetivo del juego es reducir los puntos de vida del oponente a cero. Hearthstone es un juego de cartas por turnos, en cada turno los jugadores roban cartas de su mazo y las utilizan para invocar criaturas, lanzar hechizos y usar armas para atacar al oponente.

2.2.1 LOS HÉROES Y LAS CLASES

Cada jugador debe elegir un "héroe", cada héroe está asociado a una clase única. Esta clase definirá el tipo de cartas que puede usar y le permitirá al jugador usar un poder de héroe único. Cada héroe tiene una cantidad de 30 puntos de vida. Si uno de los jugadores ve sus puntos de vida reducidos a cero perderá la partida.

Al comienzo de cada turno, el jugador roba una carta de su mazo que comienza con 30 cartas. Existen unos mazos preestablecidos, pero, los jugadores pueden crear nuevos mazos para buscar estrategias innovadoras. Muchas de las cartas son de tipo neutral y están disponibles para todas las clases, sin embargo, existe un conjunto de cartas que son únicos de un héroe o clase. De este modo, los héroes tienen estrategias únicas, así como, ventajas y desventajas contra otras clases.

Ilustración 1. Héroe de Hearthstone, Clase de Mago

Durante el turno, los jugadores pueden tomar una gran variedad de decisiones; usar su poder de héroe único, utilizar a sus esbirros para atacar a otros esbirros enemigos o al héroe oponente, atacar con el propio héroe si tiene un arma equipada, jugar cartas desde la mano al campo de batalla.

Ilustración 2. Poder de Héroe, clase de Mago

Para realizar estas acciones los jugadores necesitan pagar un coste, este coste se conoce como cristales de maná. Al principio de la partida comienzan con un cristal de maná y cada turno ganan un cristal más hasta llegar al límite de 10 cristales de maná. Todos los turnos, el maná se regenera, por tanto, los cristales que no son utilizados en un turno se desperdician. Cuantos más cristales de maná pueda usar un jugador, más poderosas y complejas jugadas podrá realizar con sus cartas.

Ilustración 3. Componentes de una partida de Hearthstone, por Hearthsone Fandom

2.2.2 COMPONENTES DE LA PARTIDA

Como se puede ver en la imagen, lo primero que el jugador encuentra en la parte inferior es su mano. Aquí el jugador almacena las cartas que roba del mazo, puede llegar a tener hasta diez cartas en la mano. Encima de las cartas nos encontramos el héroe y el poder de héroe que son únicos de la clase a la que pertenecen. Justo a la derecha, podemos ver los

cristales de maná. En este caso el jugador tiene tres cristales de maná, de los cuales, dos han sido gastados y, por eso, se ven en negro.

En el centro de la imagen, nos encontramos el campo de batalla, que se divide en dos partes, una para cada jugador. En el campo de batalla se juegan los esbirros. Si nos fijamos a la derecha del oponente, en la parte superior, veremos un arma que tiene equipada. Esa arma le permitirá atacar igual que lo hacen los esbirros. Por último, observamos los mazos a la derecha de la imagen y las cartas del oponente en la parte superior, estas cartas están boca abajo y son información desconocida para el jugador.

A pesar de que el objetivo del juego consiste en bajar los puntos de vida de tu oponente a cero, existen un gran número de estrategias y otros pequeños objetivos que permiten a los jugadores con experiencia sobreponerse a sus rivales. El uso de los esbirros, los hechizos, las sinergias y la componente estocástica del juego son algunas de las dinámicas que un jugador principiante tendrá que aprender para mejorar en Hearthstone.

2.2.3 EXPLICACIÓN DE LOS TURNOS

Al comenzar el turno, el jugador recupera los cristales de maná gastados y gana un cristal si tiene menos de diez cristales. Antes de que pueda ejecutar acciones el jugador roba una carta de su mano, si su mano está llena (si tiene diez cartas en la mano) perderá la carta. A partir de ese momento, el jugador puede tomar tantas acciones como sean posibles, sin embargo, puede terminar el turno en cualquier momento, si no tiene pendiente una acción de tipo elección. A continuación, trataremos de explicar las acciones que puede tomar el jugador durante su turno:

- Jugar cartas desde la mano: las cartas se pueden jugar siempre que el jugador tenga suficientes cristales de maná y, se cumplan las condiciones necesarias para cada carta (Ahondaremos en las cartas en la próxima sección).

- Atacar con esbirros: cuando un esbirro ataca a otro, estos se hacen daño mutuamente, y pierden puntos de vida. Sin embargo, cuando los esbirros atacan al héroe enemigo, es el héroe el único que pierde vida.
- Atacar con el héroe: los héroes solo pueden atacar cuando ganan ataque, lo más común, es que ataquen cuando se equipan una carta de tipo arma. De nuevo si los héroes atacan a los esbirros se harán daño mutuamente, sin embargo, si se ataca al héroe enemigo, solo recibirá daño el héroe que está siendo atacado.
- Usar el poder héroe: Los poderes de héroes son poderes especiales de cada clase y se pueden usar una vez por turno pagando dos cristales de maná.
- Acciones de elección: En algunas situaciones el jugador activará una acción en la que deberá elegir entre varias opciones. Por ejemplo, al inicio de la partida los jugadores roban tres cartas y pueden elegir si intercambiarlas por otras o quedárselas para el comienzo de la partida.

2.2.4 LAS CARTAS

Las cartas de Hearthstone se dividen en tres tipos: criaturas, hechizos y armas. Las criaturas son personajes que se utilizan para atacar al oponente o defenderse de sus ataques. Los hechizos son habilidades que se utilizan para infligir daño, curar, otorgar bonificaciones o eliminar cartas del oponente. Las armas son herramientas que se utilizan para atacar al oponente o sus criaturas.

Una de las características más interesantes de Hearthstone es que cada carta tiene una habilidad única que la hace diferente de las demás. Los jugadores pueden construir mazos personalizados utilizando cartas de diferentes tipos y clases, cada una con sus propias fortalezas y debilidades. Esto significa que hay una gran variedad de estrategias posibles en el juego. Las cartas se pueden dividir en tres tipos.

2.2.4.1 Los esbirros

Los esbirros se caracterizan por tener estadísticas de ataque y vida. Además, muchos esbirros tienen habilidades únicas. Estas habilidades se pueden ejecutar, dependiendo de la carta, de muchas maneras, por ejemplo, al jugarla, al destruir el esbirro o cada vez que se usa el poder de héroe. Todos los esbirros ocupan un espacio del campo de batalla, pudiendo tener como máximo siete esbirros al mismo tiempo.

Ilustración 4. Carta de esbirro

2.2.4.2 Los hechizos

Los hechizos, en general no ocupan espacio, pero interactúan con el entorno, los esbirros o con los héroes. Algunos ejemplos de habilidades de los hechizos son provocar daño, cambiar estadísticas de los esbirros o robar cartas del mazo. Los únicos hechizos que ocupan "espacio" son los secretos y los que invocan esbirros. Los secretos son hechizos que el oponente no puede ver y que ejecutan una habilidad cuando son desvelados (Solo se puede tener cinco secretos activos al mismo tiempo).

Ilustración 5. Carta de hechizo

2.2.4.3 Las armas

Las armas tienen estadísticas de ataque y durabilidad. El ataque funciona igual que en los esbirros. La durabilidad es el número de veces que puedes atacar con esa arma. Por tanto, cada vez que se ataca con un arma, esta pierde un punto de durabilidad.

Ilustración 6. Carta de arma

2.2.5 INTERÉS PARA LA INTELIGENCIA ARTIFICIAL

Otra característica interesante de Hearthstone es que los jugadores no siempre tienen toda la información sobre las cartas del oponente. Algunas cartas se mantienen ocultas hasta que se juegan, lo que significa que los jugadores deben valorar numerosas y diversas situaciones. Esta información incompleta es lo que hace que el juego sea de mayor interés a la hora de aplicar modelos de inteligencia artificial.

El Hearthstone tiene un espacio de observación y de acción muy grande en comparación con otros juegos. Como veremos en futuras secciones el espacio de observación es mucho mayor que el ajedrez o que el número de átomos que existen en el universo.

Existen varias estrategias comunes en Hearthstone, de hecho, muchos mazos que son creados por los jugadores suelen seguir alguna de estas estrategias. Las principales estrategias de los mazos son las agresivas, las de rango medio, las de combo y las de control. Ahora explicaremos superficialmente, los objetivos que persigue cada una:

- Estrategia Agresiva: Los mazos que siguen esta estrategia suelen tener cartas que cuestan pocos cristales de maná, son mazos baratos. El objetivo es jugar muchas cartas al principio de la partida y conseguir derrotar al rival antes de que pueda jugar suficientes cartas y no pueda defenderse. Por tanto, priorizará bajar los puntos de vida del enemigo antes que intentar realizar combinaciones más codiciosas. La desventaja de estos mazos es la falta de recursos cuándo las partidas se extienden. Un ejemplo de estos mazos es el Cazador FaceHunter.
- Estrategia Control: Estos mazos son el opuesto a los mazos agresivos. Tratan de hacer combinaciones muy caras y poderosas. Durante los primeros turnos, usan hechizos para eliminar las amenazas del oponente. Estos mazos suelen ganar contra el resto cuándo las partidas se alargan. La gran dificultad de estos mazos se debe a la complicada gestión de recursos al comienzo de la partida y en

conseguir no perder en los primeros turnos de la partida dónde no jugarán muchas cartas. Un mazo típico que sigue esta estrategia es el guerrero control.

- Estrategias de Rango Medio: En este caso los mazos tratan de buscar un punto intermedio, entre una vertiente agresiva y la de control. Estos mazos tienen cartas que cuestan pocos cristales de maná y otras que cuestan más. Su objetivo es robar las cartas, en orden de coste para poder jugar las cartas más poderosas en cada momento. Además, suelen permitir a los jugadores seguir estrategias más agresivas o de control para contrarrestar la estrategia del rival. Sin embargo, por la componente estocástica del juego, esta estrategia puede complicarse, puesto que, si no se juegan las cartas en orden serán demasiado lentos para ganar los enfrentamientos. Un mazo de rango Medio es el Mago clásico.
- Estrategias de Combo: Estos mazos tratan de realizar combos muy poderosos juntando muchas cartas poderosas. Suelen tener una alta ventaja contra mazos de control, pero, suelen perder contra mazos agresivos antes de conseguir todas las cartas que necesitan para realizar sus combos. Un mazo típico de Combo es el Pícaro Mill.

En resumen, Hearthstone es un juego de cartas emocionante y desafiante en el que los jugadores utilizan estrategias y habilidades para reducir los puntos de vida del oponente a cero. La gran variedad de cartas y mazos personalizados, junto con la información incompleta sobre las cartas del oponente, hacen que el juego sea interesante y difícil de dominar. Esto lo convierte en un excelente espacio de problemas para implementar algoritmos de RL y explorar nuevas estrategias de juego.

2.3 DESCRIPCIÓN DE ENTORNOS GYMNASIUM

Gymnasium es una librería de APIs estandarizadas para el aprendizaje por refuerzo. Además, esta librería contiene un amplio conjunto de entornos preparados para aplicar dichos algoritmos. Los entornos que contiene son una colección de videojuegos que han tenido un gran impacto en la investigación de la inteligencia artificial y el aprendizaje por refuerzo.

Los entornos que contiene la herramienta se actualizan junto con la propia librería y tienen documentación clara para conseguir implementar algoritmos eficaces. Dependiendo del entorno algunos algoritmos no son una buena opción, esto implica que se debe tener un conocimiento inicial sobre los métodos de inteligencia artificial para saber cuál aplicar en cada caso. Por tanto, esta herramienta permite no solo probar nuevos algoritmos de aprendizaje por refuerzo, pero, obtener un primer acercamiento y conocimiento sobre los mismos.

2.3.1 ESTRUCTURA DE LOS ENTORNOS

Un entorno de Gymnasium tiene cuatro funciones principales que forman la estructura del entorno. Estas funciones son obligatorias para crear un objeto de entorno y muchos algoritmos de Stable Baselines usan estas funciones de manera predeterminada en sus procesos de aprendizaje por refuerzo.

2.3.1.1 Funciones principales

- Make: esta función permite crear una instancia del entorno. El entorno puede tener varias versiones. Esta función debe inicializar las variables y el entorno. Además, en la mayoría de los casos, esta función admite varios argumentos para personalizar las características del entorno dónde se entrenará al agente.

- Reset: esta función tiene como objetivo reiniciar el entorno a un estado inicial. Como hemos visto anteriormente, los entornos pueden ser continuos o episódicos. Sin embargo, en el caso de los videojuegos no se suele trabajar con entornos continuos, es decir, que siempre existe un momento de inicio y otro de final. Cada vez que se llega al final del episodio el entorno debe reiniciarse.
- Step: los "Steps" son las iteraciones de los algoritmos. En cada iteración se realiza una acción, además, se recibe una observación y una puntuación que refleja la nueva situación del entorno. Las iteraciones se consideran la menor medida de tiempo en el aprendizaje de los algoritmos. Como vimos en la sección de aprendizaje por refuerzo, para algunos entornos es preferible actualizar el valor de la política o la función de valor cada iteración, mientras que, otros entornos aprenden mejor con actualizaciones episódicas de estas funciones.
- Render: Esta última función permite visualizar el entorno en aquellos videojuegos dónde está funcionalidad esta desarrollada.

El flujo lógico de un entorno de Gymnasium comienza por crear el entorno con la función "Make" junto con los argumentos que se prefieran en cada caso. Después de iniciar el entorno, se debe usar la función de reiniciado para obtener la observación inicial y, partir de esta observación, se podrá comenzar el bucle de aprendizaje del algoritmo.

Cada una de las iteraciones de este bucle de aprendizaje se ejecuta con la función "Step" y el bucle se ejecuta de forma infinita. Dentro de cada iteración se analizará la observación y el agente tomará una acción. Después, se ejecutará esta acción en el entorno, esta acción cambiará el entorno, devolverá una nueva observación y una puntuación. Con la puntuación el algoritmo será capaz de aprender y cambiar la política o la función del valor de los estados. Finalmente, la función "Step" también debe devolver dos expresiones booleanas que son la de final de episodio y la de error. La primera se devuelve cuando la

partida se termina y, la segunda, cuando ocurre un error en el entorno o el número de iteraciones excede un límite establecido.

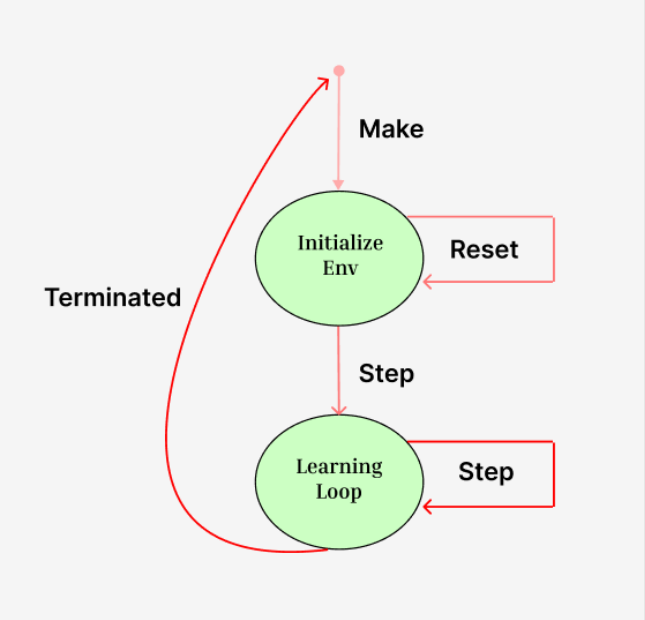


Ilustración 7. Flujo lógico de los entornos de Gymnasium

Los espacios de observación y de acción forman parte de la estructura del entorno y deben ser definidos para poder ejecutar los algoritmos. Estos espacios permiten la interacción de los agentes con los entornos, tanto, en procesos de inferencia, como, de en entrenamiento. La diferencia entre el proceso de inferencia y entrenamiento viene dada por la puntuación que en el caso del entrenamiento será usada para actualizar la política o la función de valores de los estados. Estos espacios de observación y acción son, respectivamente, la entrada y la salida de la función que define al agente.

Por un lado, el espacio de observación define toda la información que el agente puede visualizar y que usará para definir su política de actuación o para detectar el estado en el que se encuentra. Por otro lado, el espacio de acción define todas las posibilidades que el agente tiene para interaccionar con el entorno. Estos espacios en Gymnasium pueden

definirse con distintas funciones según los posibles valores que puedan tomar las observaciones o acciones.

- Box: Este espacio permite definir un conjunto de valores continuos con una cota superior y una inferior.
- Discrete: Este espacio define un conjunto de n acciones discretas que puede ser usado para el espacio de acción y el espacio de observación.
- Dict: Este espacio es un contenedor de otros espacios simples. Todos los demás espacios pueden contenerse dentro del diccionario.
- Tuple: Este espacio define tuplas de valores.
- Multibinary: Este espacio define una lista de n valores binarios, dónde el argumento n puede ser un número o una lista de números.
- MultiDiscrete: Contiene un conjunto de espacios discretos, dónde cada uno de dichos espacios tiene un número de acciones distintos.

2.3.1.2 Wrappers

Para modificar los entornos también se pueden usar los "Wrappers" que permiten modificar entornos de forma sencilla y reduciendo el número de líneas de código. Estos "Wrappers" se pueden usar en cadena y muchos entornos aplican varios por defecto. Cabe destacar, que los "Wrappers" solo pueden aplicarse una vez a sido iniciado un entorno. Existen muchas funcionalidades, pero, los "Wrappers" más comunes que conviene conocer son:

- TimeLimit: Genera una señal de error cuándo el entorno supera un número máximo de iteraciones. Respeta otras señales de error generadas por el entorno.
- ClipAction: Transforma la acción elegida para que se ajuste a un espacio de acción del tipo "Box".

- RescaleAction: Cambia la escala de un espacio de acción para que se ajuste a un intervalo determinado.
- TimeAwareObservation: Añade información sobre la iteración en la que se encuentra el agente a la observación. Especialmente útil para los algoritmos que aplican métodos de Markov.

2.4 *HEARTHSTONE API*

Para generar las cartas de Hearthstone se usa un servicio llamado Hearthstone API que permite obtener un archivo XML con la información de todas las cartas. Sin embargo, algunas de estas no están implementadas y, debe hacerse un tratamiento de limpieza de estos datos antes de ser usados. Para más información es aconsejable visitar el repositorio de la Api (HearthSim, 2023).

2.5 *FIREPLACE API*

Fireplace es un simulador de Hearthstone escrito en Python. Es la implementación más avanzada y fiel del motor de Hearthstone, e incluye un lenguaje declarativo en Python. Fireplace incluye un servidor Kettle incorporado que le permite comunicarse con el cliente real de Hearthstone a través de Stove, o simular partidas. Para profundizar en el conocimiento de Fireplace se recomienda clonar el repositorio (Fireplace, 2021).

2.6 *STABLE BASELINES*

La biblioteca incluye varios algoritmos de RL populares, como DQN, A2C, PPO, TRPO y SAC, que se pueden usar para entrenar agentes en diferentes entornos de RL. También incluye una amplia gama de funciones y herramientas útiles, como funciones de preprocesamiento de datos, políticas personalizadas y herramientas de visualización para analizar el desempeño de los agentes.

"Stable Baselines" está diseñado para ser fácil de usar y personalizable, lo que lo hace ideal para investigadores y desarrolladores que desean implementar y experimentar con diferentes algoritmos de RL en sus proyectos. Además, también es compatible con una amplia gama de entornos de RL, lo que permite a los usuarios entrenar agentes en diferentes aplicaciones y escenarios, como juegos, robótica y simulaciones. Stable Baselines es una librería dinámica que sigue en continuo desarrollo y mantenimiento. Esto hace que sea una gran elección para el desarrollo de proyectos.

En resumen, "Stable Baselines" es una poderosa biblioteca de aprendizaje por refuerzo que proporciona una interfaz fácil de usar y herramientas útiles para entrenar y evaluar agentes de RL en diferentes entornos. Su amplia gama de algoritmos de RL y funciones personalizables lo hacen una opción popular para investigadores y desarrolladores que trabajan en proyectos de RL. Si quieres revisar la implementación de sus algoritmos puedes revisar su repositorio (Baselines, 2023).

Capítulo 3. ESTADO DE LA CUESTIÓN

La experiencia académica mejora con la experimentación y los desafíos, especialmente, en aquellas actividades que son cooperativas. Consecuentemente, la realización de este tipo de proyectos en entornos de aprendizaje es común y, en el ámbito de la inteligencia artificial, está ligada asiduamente a los videojuegos (Burguillo, 2010).

Debido a la dificultad de la temática de la inteligencia artificial, los desafíos se suelen desarrollar en centros universitarios. Por ejemplo, en la Universidad de Bolonia, se desarrolló un torneo basado en Nine Men's Morris consiguiendo mejorar la motivación y el número de horas de estudio de los estudiantes (Chesani, 2017). Este resultado también se obtuvo en la competición de la Universidad de Oporto que utilizó el juego Ataxx como entorno del desafío para fomentar el aprendizaje de los estudiantes en el campo de Logic Programming (Ribeiro, 2009). En la Universidad de Huelva se fomentó la participación del alumnado en competiciones online de inteligencia artificial durante el horario lectivo para perseguir el mismo objetivo (Carpio Cañada, 2015). Otras competiciones han desarrollado juegos completamente nuevos como NERO, en la Universidad de Austin en Texas, dónde en el propio juego se debe diseñar un entrenamiento y usar distintos parámetros para entrenar robots que, más tarde, se enfrentarán en una batalla (Stanley, 2005).

Los Collectable Card Games (CCG) o juegos de cartas coleccionables no han estado muy presentes en las competiciones de inteligencia artificial. Sin embargo, estos juegos de estrategia suponen un desafío interesante debido al enorme espacio de posibilidades que suponen (todas las diferentes posibles partidas) y, por otro lado, a su característica de información incompleta, es decir, no toda la información relevante está disponible al tomar decisiones. La mayoría de los CCG tienen similitudes; son juegos de un jugador contra un jugador, las cartas del oponente se desconocen, cada jugador tiene la posibilidad de crear su propio mazo con las cartas que ha coleccionado dentro del juego. Entre los juegos CCG

más famosos encontramos el Hearthstone, Magic the Gathering Arena, Pokémon Tradeable Card Game, Yu-Gi-Oh y Gwent.

El Hearthstone es un juego que fue creado por la compañía Blizzard en 2014. Ese mismo año fue nominado a mejor juego del año en los GOTY y ganó el premio al mejor juego para móvil (TheGameAwards, 2022). Además, en 2015, ganó el premio al mejor juego multijugador en los BAFTA Games Awards (BAFTA, 2022). En 2020 Hearthstone alcanzó un número superior a 23,5 millones de jugadores activos en el mundo (Blizzard, 2021). El reciente crecimiento de este juego y de otros CCG, así y cómo, sus interesantes características han motivado varios artículos en el ámbito de la inteligencia artificial (Hoover, 2020).

Las partidas en este videojuego son de un jugador contra un jugador. Ambos jugadores empiezan la partida con un mazo de 30 o 40 cartas que previamente han diseñado juntando cartas de sus colecciones. Cada mazo debe tener un "héroe" asignado, dicho héroe comienza con un número de puntos de vida iguales al número de cartas en el mazo y tiene un poder único. El juego se desarrolla por turnos con un límite de tiempo dentro del cual debe pensar una estrategia y jugar las cartas.

El papel de los mazos es muy importante a la hora de ganar al oponente, puesto que, de ellos dependen las sinergias que haya entre las cartas, su poder y su coste. Por tanto, algunos artículos se han centrado en la creación de mazos con inteligencia artificial (Fontaine, 2019), incluso, empresas como HSReplay han creado modelos de negocios basados en el uso de datos de jugadores para encontrar los mazos con mayor proporción de victorias, así como, que mazo puede contrarrestar a otro (HSReplay, 2022). También se han propuesto modelos para la creación de cartas en CCG, cuya aplicación puede ser de gran utilidad para los diseñadores (Summerville, 2016).

Los algoritmos que tratan de optimizar en número de victorias dado uno, o varios mazos, son los más recurrentes dentro de los CCG porque permiten aplicar diversas

técnicas de inteligencia artificial y son más desafiantes. Para la implementación de competiciones de inteligencia artificial usando Hearthstone se ha hecho uso de un simulador llamado "Sabberstone" (Dockhorn, 2019). Este simulador se ha usado en competiciones académicas como en la universidad de Magdeburgo (OVGU, 2020).

Capítulo 4. FUNDAMENTOS TEÓRICOS

4.1 APRENDIZAJE POR REFUERZO

El aprendizaje por refuerzo es un enfoque de la inteligencia artificial y del aprendizaje automático que se basa en el concepto de recompensa y retroalimentación para enseñar a un agente cómo tomar decisiones óptimas en un entorno determinado. A diferencia del aprendizaje supervisado, donde se le proporciona al agente un conjunto de ejemplos etiquetados, y del aprendizaje no supervisado, donde no se le proporciona ninguna etiqueta, el aprendizaje por refuerzo se basa en el principio de prueba y error.

En el aprendizaje por refuerzo, un agente interactúa con un entorno y toma acciones en función de su estado actual. Cada acción tiene asociada una recompensa que puede ser positiva o negativa. El objetivo del agente es maximizar las recompensas acumuladas a lo largo del tiempo. Para lograr esto, el agente utiliza una política, que es una estrategia que determina qué acción tomar en función del estado actual.

A medida que el agente interactúa con el entorno, aprende a través de la retroalimentación que recibe en forma de recompensas. Utiliza esta retroalimentación para ajustar su política y mejorar su rendimiento en el entorno. El aprendizaje por refuerzo se basa en algoritmos que buscan encontrar la mejor política posible para maximizar las recompensas a largo plazo.

El aprendizaje por refuerzo tiene aplicaciones en una amplia gama de campos, desde robótica y control de procesos hasta juegos y optimización de recursos. Permite que los agentes aprendan a tomar decisiones en entornos complejos y dinámicos, adaptándose a nuevas situaciones y mejorando su rendimiento con la experiencia.

Sin embargo, el aprendizaje por refuerzo también presenta desafíos. La exploración del agente en el entorno puede llevar a un proceso de prueba y error prolongado. Además, el diseño de la función de recompensa puede resultar crucial, ya que puede influir en el comportamiento aprendido del agente.

Esta familia de algoritmos se utiliza comúnmente en el mundo de los videojuegos. A lo largo de la historia son muchos los hitos que la academia ha conseguido en el ámbito de los juegos. Desde 1952, cuándo A.S.Douglas creo un software capaz de jugar al "Tres en Raya" al nivel de los humanos y, unos años después, dio forma a lo que hoy conocemos como aprendizaje por refuerzo haciendo que un agente jugará contra sí mismo en el juego de las damas (Samuel, 1959). Más tarde, en 1992 Gerald Tesauro uso una red neuronal y diferencias temporales para crear un modelo que pudiese jugar al "Backgammon" con jugadores de primer nivel (Tesauro G. , 1991) (Tesauro G. a., 1995). Antes de los 2000, el agente Chinook Checkers ya había derrotado a la que era campeona mundial de las damas (Schaeffer J. a., 1996). Poca más de diez años más tarde, se probaría que la campeona Marion Tinsley no podría haber obtenido nada mejor que un empate jugando contra Chinook (Schaeffer J. a., 2007).

Otros agentes han ganado a campeones mundiales de ajedrez (Hsu, 2002), han competido y ganado en tiempo real durante programas de televisión tipo trivial (Ferrucci, 2010). En 2016 y 2017, Google demostró que juegos tan complicados como el Go podían ser dominados completamente por la inteligencia artificial (Levinovitz, 2014) (Silver, 2016). La inteligencia artificial se ha llegado a usar en test de Turings para simular el comportamiento humano (Hingston, 2010) (haker, 2013) y para la creación de juegos basados en el entrenamiento de agentes en tiempo real como NERO (Stanley, 2005).

La gamificación del aprendizaje se ha usado asiduamente en el sector educativo con grandes resultados (Lee, 2011) (Dicheva, 2015). Se ha llegado a usar previamente con

entornos de Hearthstone (Dockhorn, 2019). Por ello, para explicar cómo funciona el aprendizaje por refuerzo nos centraremos en esta clase de ejemplos.

4.1.1 APRENDIZAJE POR REFUERZO APLICADO A VIDEOJUEGOS

En la mayoría de los videojuegos existen unas reglas y unos objetivos claros, que por lo que veremos, son dos factores muy importantes para la eficacia de este algoritmo.

El aprendizaje por refuerzo asume dos partes diferenciadas, estas son, un entorno de entrenamiento (el juego) y un agente (el bot que estamos enseñando a jugar). El entorno viene definido por unas reglas y, en él, el agente puede obtener información (observaciones) y basándose en la información obtenida, puede analizar la situación e interactuar con el entorno (acción). Por tanto, el objetivo del aprendizaje por refuerzo es aprender a analizar las distintas situaciones y escoger la acción que vaya a resultar más beneficiosa para el agente (obtener más puntuación).

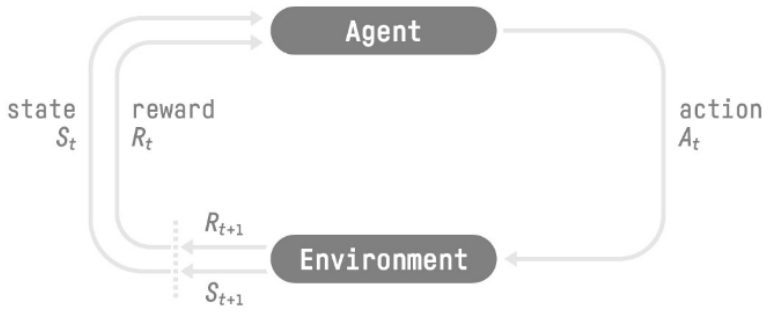


Ilustración 8. Esquema general de aprendizaje de un agente mediante RL,

Fuente: Reinforcement Learning: An Introduction, Richard Sutton and Andrew G. Barto

En esta foto podemos ver el flujo del proceso de aprendizaje. Inicialmente el agente entra en contacto con el entorno y lo observa, después de analizar la situación, ejecuta una acción en el entorno. Esta interacción, puede cambiar el entorno y la puntuación que recibe

el agente. En este momento, comienza una nueva iteración del bucle en la que el agente recibe una nueva observación y una puntuación.

Gracias a esta información el agente puede aprender si la acción ha sido beneficiosa, ha tenido puntuación positiva, o si ha sido una acción perjudicial, es decir, que ha recibido una puntuación negativa.

4.1.2 COMPONENTES DEL ENTORNO

Los componentes principales son el espacio de acciones, las observaciones y las tareas.

Dentro del espacio de acciones, podemos diferenciar entre acciones discretas y continuas. Por ejemplo, cuando tratamos el movimiento de un agente en un videojuego, este puede implementarse de diversas formas. Puede ser discreto si el agente solo puede moverse a la derecha y la izquierda. Sin embargo, en juegos 3D, dónde el movimiento viene dado por un ratón o un joystick, las acciones son infinitas.

Las observaciones pueden ser completa e incompleta. A una observación completa se le conoce cómo un estado. Un estado es una situación única en la que se puede encontrar un agente dentro de un entorno. Sin embargo, cuando las observaciones son incompletas, es posible que no se pueda saber el estado en el que se encuentra el agente. En estos casos, tendremos una distribución de probabilidad de los distintos estados.

Las Tareas pueden ser episódicas o continuas. En las tareas episódicas se puede definir un episodio, para ello, se necesita un punto de inicio y de final. Sin embargo, en las continuas no existe un punto de final, por ejemplo, si se intenta predecir el tiempo, estaríamos ante una tarea continua porque el entorno no tiene un punto final.

4.1.3 PUNTUACIÓN

El objetivo de los juegos suele ser conseguir la mayor puntuación posible o ganar al resto de tus rivales. Desde el enfoque matemático con el que vemos el problema, nosotros

veremos todos los casos cómo una puntuación que se pueda medir de forma numérica. Por ejemplo, en un juego de carreras dónde compiten diez jugadores, el primero se podría llevar diez puntos, el segundo nueve y así podríamos crear un sistema de puntuación.

Es importante entender que esta puntuación es arbitraria y la elegimos, bien nosotros o, bien el propio juego. De esta forma, en el caso del juego de carreras, si damos una puntuación muy alta por hacer derrapes (porque creemos que así el jugador será más rápido) puede que nuestro algoritmo priorice hacer derrapes a ganar la carrera. En conclusión, el aprendizaje por refuerzo simplemente buscará conseguir la máxima puntuación y nosotros tendremos que intentar guiar al algoritmo para conseguir el objetivo que buscamos.

4.1.3.1 Fórmula de la puntuación

Vamos a definir la fórmula de la puntuación para nuestro algoritmo. Queremos maximizar la puntuación a lo largo del tiempo, es decir, que queremos obtener una fórmula para la puntuación total. Nuestro agente recibe puntos cada vez que toma una acción. Llamaremos a la puntuación R, siguiendo su nomenclatura en inglés, y sumaremos todas las veces que nuestro agente recibe puntos.

$$R(\tau) = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + \cdots$$

Ecuación 1. Fórmula de la puntuación

Para escribir esta ecuación de forma más general:

$$R(\tau) = \sum_{k=0}^{\infty} r_{t+k+1}$$

Ecuación 2. Fórmula de la puntuación (General)

La Tau (τ) que observamos en esta fórmula es el conjunto de estados y acciones que sucederán en el futuro. Lógicamente, esto es una suposición, puesto que, no siempre

podemos saber a ciencia cierta cómo se desarrollará la partida. De hecho, lo normal es que estemos más seguros de aquello que sucederá pronto que de lo que vendrá más tarde.

El descuento se utiliza para contrarrestar la incertidumbre de los futuros estados de la partida. De esta forma, reduciremos el valor de la puntuación cuánto más tarde ocurra. Una acción que sucederá pronto es más probable que ocurra y se descontará un poco en comparación con una situación más cercana al final de la partida que se descontará en mayor medida.

Para traducir esto de forma matemática, añadiremos un valor gamma. La fórmula que vimos anteriormente quedaría de la siguiente manera:

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$$

Ecuación 3. Fórmula de la puntuación con descuento

Y la forma general sería:

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Ecuación 4. Fórmula de la puntuación con descuento (General)

El valor de gamma debe estar restringido entre cero y uno, en caso contrario, la puntuación tendería a infinito para valores muy lejanos en el futuro. Para hacernos a la idea, cuanto mayor sea el valor (más cercano a uno) mayor importancia tendrán las acciones futuras y cuanto menor sea el valor de gamma más descontadas estarán las acciones futuras.

4.1.4 EXPLOTACIÓN Y EXPLORACIÓN

Basándonos en esta fórmula todavía podemos caer en una trampa común. Tal y como lo hemos definido, el algoritmo de aprendizaje por refuerzo tratará de explotar los recursos

que conoce al máximo. Esto implica que según vaya aprendiendo, siempre tomará las acciones que le den una puntuación más alta. Sin embargo, en muchos juegos de estrategia, tienes que tomar decisiones que son peores que otras en el momento pero que te permiten obtener una mejor puntuación a la larga.

En el juego del ajedrez imagina una ocasión en la que tienes la opción de comer una pieza del contrincante. Comer la pieza aportará a nuestro agente mayor puntuación que no hacerlo, pero, es posible que, a lo largo de varios turnos, la puntuación sea peor que otra en la que decidió no comer la pieza del contrincante. Para poder descubrir nuevas formas de mejorar la puntuación, a veces, es necesario probar nuevas opciones, aunque, estas no sean las óptimas. A este proceso se le conoce como exploración.

Para implementar esta funcionalidad se suele usar un parámetro ϵ que permitirá generar la exploración. Por ejemplo, si queremos que nuestro agente explore el entorno un 20% por ciento de las veces definiremos épsilon, tal que, $\epsilon = 0.2$. Cuando el algoritmo escoja la acción que según su función maximice la puntuación, la ejecutará con una probabilidad de $1-\epsilon$ y, en el resto de los casos el agente realizará una acción aleatoria para explorar el entorno. Esta política se conoce como "épsilon greedy" o "$\epsilon - greedy$".

4.1.5 LA POLÍTICA

La función que devuelve la acción que debe ejecutar el agente basándose en la observación se conoce como la política. Usando el aprendizaje por refuerzo, el algoritmo tratará de encontrar la mejor función posible. El objetivo es conseguir encontrar la función que consiga la mayor puntuación esperada.

Existen dos métodos principales para desarrollar la política del agente. Estos se conocen como los métodos basados en la política y los métodos basados en el estado de la acción. El primero trata de aprender cuál es la acción que puede tomar el agente para maximizar su puntuación en cada caso. El segundo aprende que estados son más valiosos y luego elige la

acción que lleve al agente a esos estados. Cabe destacar que en muchos casos el agente tendrá que aprender que acciones se deben usar para llegar de un estado a otro, pero, especialmente en problemas de pequeña envergadura, el mapeo entre las acciones y los estados puede estar mapeado por el propio programador o usuario.

En los métodos basados en la política se aprende directamente la función que maximiza la puntuación esperada. Las funciones pueden ser de dos tipos, deterministas y estocásticas. Las deterministas devuelven una única acción, mientras que las estocásticas devuelven una distribución estadística con las posibles acciones y la probabilidad de elegir cada una.

En cuanto a los métodos basados en el valor de los estados, el agente aprenderá una función que consiga mapear los estados con su respectivo valor. El valor de cada estado se calculará teniendo en cuenta la puntuación por llegar a ese estado más la puntuación descontada de los estados futuros, asumiendo que el agente escoge la función óptima en cada caso.

4.1.6 TIPOS DE MÉTODOS BASADOS EN EL VALOR DE LOS ESTADOS

Cuando se usan métodos basados en el valor de los estados, siempre habrá que predefinir una política. Una política habitual consiste en escoger la acción que produzca la mayor puntuación basándose en los valores de los estados. Esta política se conoce como la política "greedy". Otra política muy común que veremos más adelante se conoce como la función "greedy épsilon" que permite gestionar el balance entre explotación y exploración.

El objetivo de los métodos que usan el valor de los estados es, por tanto, conseguir un valor óptimo de la función de valor y, al usar esta función, conseguir aplicar la política óptima para el problema en cuestión. Para representar esta conexión de forma matemática, llamaremos π a la política y Q o V a la función que mapea el valor de los estados.

$$\pi^*(s) = argmax_a Q^*(s,a)$$

Ecuación 5. Definición de la política basada en la función del valor de los estados

Las dos funciones que vamos a distinguir son la del valor de los estados y la función del par acción y valor. La primera calcula el valor del estado teniendo en cuenta que se sigue la política óptima desde ese momento. Sin embargo, el par valor y acción calcula la puntuación de la política óptima después de escoger la acción en dicho estado inicial. El problema de ambos métodos es que pueden ser computacionalmente muy costosos porque se debe calcular cada uno de los valores de los estados, que, en su mayoría, dependen de otros.

La función del valor de los estados se puede escribir como:

$$V_\pi(s) = E_\pi Q^*[G_t \mid S_t = s]$$

Ecuación 6. Definición de la función del valor de los estados

Y la función del par valor y acción es:

$$Q_\pi(s,a) = E_\pi Q^*[G_t \mid S_t = s, A_t = a]$$

Ecuación 7. Definición de la función del valor de los pares acción y estado

4.1.7 LA ECUACIÓN DE BELLMAN

Para poder mejorar el coste computacional que hablábamos en el apartado anterior usaremos la ecuación de Bellman. Esta ecuación usa los métodos de la programación dinámica para no realizar cálculos redundantes y, de esta forma, reducir el coste computacional. La ecuación se puede expresar como:

$$V_\pi(s) = E_\pi[R_{t+1} + yV_\pi(S_{t+1}) \mid S_t = s]$$

Ecuación 8. La ecuación de Bellman

Se puede observar que hay un componente recurrente $V_\pi(S_{t+1})$ que viene descontado por el factor gamma. Esta definición nos permitirá iterar por toda la matriz de valores de manera eficiente.

4.1.8 MÉTODOS DE APRENDIZAJE

Como hemos visto el objetivo del aprendizaje por refuerzo es conseguir una política óptima con la que se consiga maximizar la puntuación. Para poder conseguir esta política óptima, se necesita realizar un entrenamiento con métodos basados en el valor de los estados o en la política. Por último, para entrenar el agente, también, distinguiremos dos métodos.

4.1.8.1 Montecarlo

El método de Montecarlo sigue un formato de aprendizaje episódico. En este caso se usa el termino para identificar la puntuación exacta del episodio. Una vez finalizado el episodio, se procede a actualizar los valores de la función.

$$V(S_t) = V(S_t) + \propto [G_t - V(S_t)]$$

Ecuación 9. Actualización del valor de los estados usando el método de Montecarlo (Episódico)

4.1.8.2 Diferencia Temporal

La diferencia temporal, en cambio, utiliza un aprendizaje por pasos o iteraciones. Para ello, usa la puntuación descontada del siguiente estado. En este caso, la puntuación es solo una aproximación porque no es posible saber la puntuación que obtendrá el agente al final del episodio.

$$V(S_t) = V(S_t) + \propto [R_{t+1} + yV(S_{t+1}) - V(S_t)]$$

Ecuación 10. Actualización del valor de los estados usando el método de Diferencia Temporal (Iterativo)

4.1.9 ALGORITMOS DE APRENDIZAJE

4.1.9.1 Las dos políticas de los algoritmos

Los modelos de inteligencia artificial tienen dos fases, el entrenamiento y la inferencia. Mientras un modelo está en la fase de aprendizaje función de política o de del valor de los estados cambia. Sin embargo, durante la inferencia no ocurre el aprendizaje, simplemente, el modelo obtiene siempre el mismo resultado para una misma situación.

Algunos algoritmos de aprendizaje usan una política distinta para el entrenamiento y la inferencia. Estos algoritmos se conocen como "Off policy algorithms" y, aquellos que usan la misma política para las dos fases se llaman "On policy algorithms".

4.1.9.2 Q Learning

El "Q Learning" es un algoritmo de aprendizaje por refuerzo. Además, lo podemos categorizar como un algoritmo basado en el valor de los estados que utiliza una función que recibe a la entrada pares valor y acción. Respecto a los métodos de aprendizaje, este algoritmo utiliza la diferencia temporal, es decir, que aprende después de cada iteración. Por último, es un algoritmo que utiliza una acción diferente para el entrenamiento y la inferencia, por tanto, es un algoritmo "Off policy".

Este algoritmo crea una matriz que tiene como dimensiones los estados y las acciones, es decir, tiene un tamaño (SxA); siendo S el número de estados y A el número de acciones. De esta forma el "Q Learning" guarda en memoria el valor de cada acción en cada estado. Con el suficiente entrenamiento este algoritmo es capaz de aprender una política óptima. Sin embargo, este algoritmo tiene problemas para la escalabilidad porque en los entornos donde el espacio de acción o el número de estados es amplio supone una cantidad de memoria insostenible o ineficiente.

4.1.9.3 El algoritmo de Q Learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

Ilustración 9. Pseudocódigo del algoritmo Q-Learning,

Fuente: [Reinforcement Learning: An Introduction, Richard Sutton and Andrew G. Barto]

En la primera línea de código se definen los posibles valores que puede tomar el parámetro alfa, que veremos a continuación. Sin embargo, el primer paso del algoritmo (segunda línea de código) es inicializar la tabla o matriz de dos dimensiones donde vamos a guardar los valores de cada par estado y acción, a esta tabla la llamaremos $Q(s, a)$. Lo común es inicializar los valores de la matriz a cero. Después vemos como se inicia un bucle que ejecutaremos en cada episodio y dónde iniciaremos el entorno.

El Segundo paso de gran importancia será elegir la acción usando la política "$\in -greedy$". De esta forma podremos explorar el entorno para poder encontrar la política óptima. Sin embargo, en este caso añadiremos un matiz importante a esta política. Empezaremos con un valor de $\in = 1.0$ y, poco a poco, iremos reduciendo este valor para que una vez descubiertas distintas opciones que ofrece el entorno, nuestro agente se centre en encontrar cuales de estas puede maximizar su puntuación.

El último paso, que estudiaremos será la actualización de nuestra matriz de valores. Si recordamos la ecuación que vimos en el método de aprendizaje de diferencia temporal,

veremos el parecido que tiene con está aplicación para una tabla de pares estado y acción. Esta actualización de los valores viene definida por la siguiente fórmula:

$$Q(S_t, A_t) = Q(S_t, A_t) + \propto [R_{t+1} + ymaxQ(S_{t+1}, a) - Q(S_t, A_t)]$$

Ecuación 11. Actualización de los valores de la matriz acción y estado

Cabe destacar que este algoritmo de aprendizaje es de tipo "Off policy", como podemos ver usa una formula distinta en la fase de inferencia ($\in -greedy$) y de entrenamiento:

$$ymaxQ(S_{t+1}, a)$$

Ecuación 12. Estimación del valor del estado del Q-Learning en la fase de entrenamiento

4.1.9.4 Deep Q Learning

Este tipo de algoritmos de aprendizaje surge para poder tener opciones escalables del "Q Learning". El espacio de acción de muchos juegos es inmenso y se necesita una forma de aproximar el valor de los estados, puesto que, aprender una función que pueda mapear el espacio de estado y acción completo resulta imposible en la práctica. Esta aproximación se puede conseguir mediante una función parametrizada. El "Deep Q Learning" es distinto al "Q Learning" porque usa redes neuronales profundas para desarrollar la función que determine el valor de los estados.

4.1.9.5 El algoritmo del Deep Q Learning

En el caso del "Deep Q Learning" no podemos usar el mismo método de entrenamiento que en el "Q Learning" porque ahora no estamos actualizando una tabla, sino, una red neuronal. Para poder actualizar la red neuronal necesitaremos calcular el "Q-Loss" que nos servirá para actualizar los pesos de las neuronas usando el método del descenso de gradiente. Como vimos en el "Q Learning" es la puntuación que recibiremos al tomar una acción más nuestra estimación de la máxima puntuación que podamos recibir en el

siguiente estado (Q-Target), después de restar, la predicción del valor de la función para el estado y la acción actual ($Q(S_t, A_t)$).

$$Q_{Loss} = R_{t+1} + ymaxQ(S_{t+1}, a) - Q(S_t, A_t)$$

Ecuación 13. Definición del valor de la función Q-Loss

Donde el Q-Target sería,

$$Q_{Target} = R_{t+1} + ymaxQ(S_{t+1}, a)$$

Ecuación 14. Definición del valor de la función Q-Target

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Ilustración 10. Pseudocódigo del algoritmo Deep Q-Learning 1,

Fuente: https://huggingface.co/learn/deep-rl-course/unit3/deep-q-algorithm?fw=pt

4.1.9.5.1. Las Fases del Deep Q Learning

Este algoritmo tiene dos fases diferenciadas, la de muestreo y de entrenamiento. Podemos ver las líneas de código que definen cada una de ellas en la ilustración 4. A grandes rasgos, el apartado de muestreo guarda en la memoria las observaciones previas. Luego en el entrenamiento usa esta memoria para aprender. Esto tiene varias ventajas, primero, se

puede sacar más provecho a las iteraciones del entorno porque se puede aprender de la misma situación varias veces (Esto puede ser realmente importante para procesos de entrenamiento largos porque el modelo puede olvidar lo que aprendió al comienzo). Segundo, permite al algoritmo aprender en un orden distinto al recopilado que puede generar robustez en algunos casos.

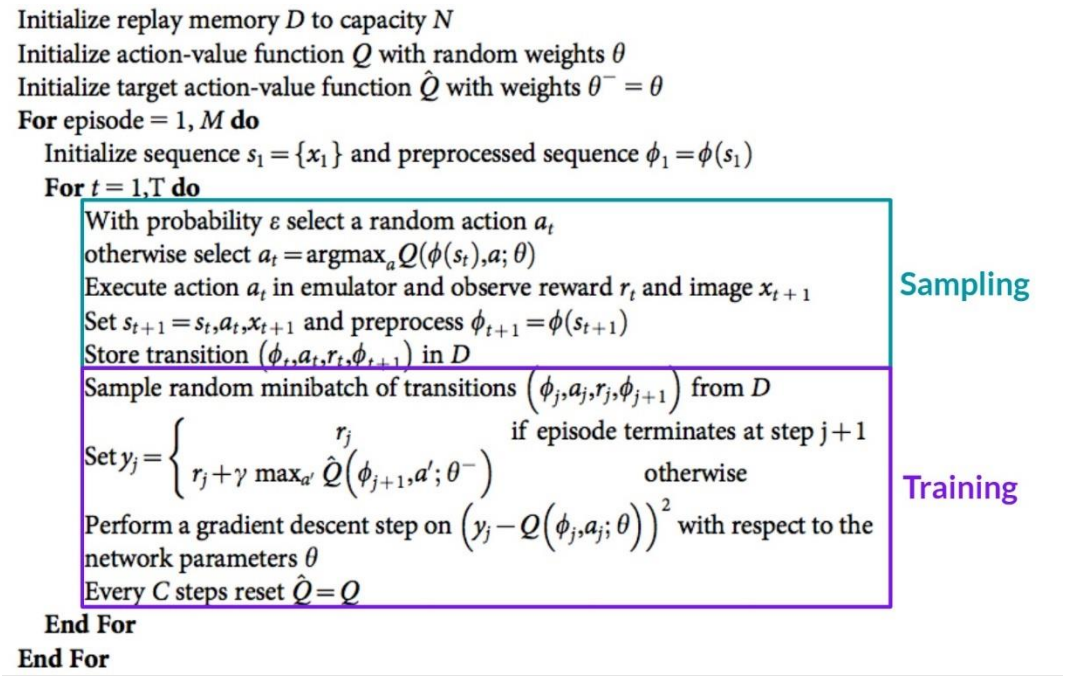


Ilustración 11. Pseudocódigo del algoritmo Deep Q-Learning 2,

Fuente: https://huggingface.co/learn/deep-rl-course/unit3/deep-q-algorithm?fw=pt

4.1.9.5.2. Inestabilidades

En este caso, el fallo de las aproximaciones con las que aprende el DQN se extiende por la red neuronal que es una función no linear. Para evitar las inestabilidades que esto pueda generar se usan un par de métodos.

Por un lado, usando el buffer de memorias que comentamos en la sección anterior podemos prevenir que el algoritmo aprenda únicamente de los acontecimientos más cercanos. En muchos casos, este método puede evitar una divergencia de o una continua oscilación de nuestra función. Dependiendo del entorno y la complejidad del entrenamiento se puede definir parámetros para modelar la cantidad de memoria desplegada para las repeticiones.

Por otro lado, se debe fijar el "Q-Target" para estabilizar el entrenamiento. Cuando calculamos este componente nos basamos en una aproximación calculada por la red neuronal y, este mismo valor, es el que usamos para actualizar los pesos de la red neuronal. Esta correlación generará inestabilidades.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Ilustración 12. Pseudocódigo del algoritmo Deep Q-Learning 3,

Fuente: https://huggingface.co/learn/deep-rl-course/unit3/deep-q-algorithm?fw=pt

Para poder fijar el valor del "Q-Target" haremos una copia de la red neuronal cada N iteraciones y usaremos esta red para calcular las aproximaciones. De esta forma, eliminamos la correlación directa entre el valor del "Q-Target" y la red neuronal.

Por último, encontramos el doble DQN, al comienzo del entrenamiento existe otro factor que también provocará inestabilidad o entrenamientos muy largos. Al principio, el valor de la mejor acción es desconocido y, basarnos en la acción con mejor puntuación, será un método fiable. Por ello, una buena solución es usar dos medidas del valor, la del DQN que elige aquella que tenga el mayor valor y una "Target network" que mida el valor de tomar esa acción en el siguiente estado.

-

Capítulo 5. DEFINICIÓN DEL TRABAJO

5.1 JUSTIFICACIÓN

Actualmente, ha surgido una gran acogida de la inteligencia artificial por su amplia implementación en el mercado laboral, así como, los avances en la investigación de esta. Por otro lado, la alta necesidad de digitalización de los servicios ha generado una gran demanda alrededor de la inteligencia artificial.

En suma, los avances de investigación y la creciente demandan provocan que un gran número de personas decidan estudiar y aprender sobre estos temas. Este proyecto trata de sumarse a una ola de proyectos que tienen como objetivo crear una educación diversa y profunda sobre algoritmos de inteligencia artificial. En específico este proyecto se centra en los algoritmos de aprendizaje por refuerzo, aunque, en el futuro el proyecto podría ampliarse a otras ramas de la inteligencia artificial.

La principal diferencia que aporta este trabajo, frente a otros previos es la búsqueda de una aplicación docente en el ámbito de estudiantes universitarios, así como, de integrar nuevos entornos creciente complejidad para el estudio del aprendizaje por refuerzo. Aporta también, un modelo de aprendizaje en formato de competición, usando un juego y librerías de programación actuales.

En conclusión, este trabajo permite un acercamiento a estudiantes y universidades, para la adopción de métodos docentes sobre la inteligencia artificial. Además, como proyecto de código abierto y gratuito, también, tiene como objetivo disminuir las barreras de las universidades y estudiantes que quieran implantar o aprender sobre este tipo de modelos.

5.2 OBJETIVOS

En este apartado se exponen y detallan los objetivos que se perseguirán durante la vida de este proyecto.

- Estudio de algoritmos de aprendizaje por refuerzo para el videojuego Hearthstone.
- Generar un entorno adecuado para una competición de inteligencias artificiales.
    - Comprender y utilizar los simuladores de código abierto para ejecutar el videojuego Hearthstone (Sabberstone y Fireplace). Decidir el simulador que se utilizará en el proyecto
    - Analizar los algoritmos existentes en la herramienta y en trabajos previos relacionados con Sabberstone.
    - Implementar un entorno de inteligencia de artificial.
    - Desarrollo de uno o varios algoritmos para la competición.
    - Análisis del o de los algoritmos

- Diseñar las instrucciones y la documentación para los alumnos.
    - Documentación sobre cómo descargar, ejecutar y comprender el entorno del desafío.
    - Desarrollar las bases de la competición y el marco de trabajo de los estudiantes.

De forma complementaria, en este trabajo se plantean otros objetivos secundarios para enriquecer el aprendizaje de los alumnos y su entendimiento de la temática.

- Creación de mazos utilizando algoritmos de inteligencia artificial
    - Conseguir base de datos que contenga porcentajes de victoria, partidas y cartas de los mazos.
    - Desarrollo de modelos para la creación de mazos

La consecución de los objetivos secundarios dependerá fuertemente de la dificultad en implementar los bots, así como, de la implementación del simulador usado para la competición.

5.3 METODOLOGÍA

Con el fin de lograr un proyecto eficiente y bien gestionado, se ha seguido la metodología de *SCRUM*, aunque se omitieron las reuniones diarias, ya que se consideró que una periodicidad semanal era suficiente para llevar a cabo los controles necesarios, es decir, definir nuevas tareas, marcar aquellas que ya han sido completadas y corregir errores. A continuación, se presentan los diferentes *Sprints* que se siguieron.

5.3.1 REVISIÓN BIBLIOGRÁFICA

Para comenzar el proyecto se estudiarán los artículos relacionados con inteligencia artificial y Hearthstone, especialmente aquellos que tengan como objetivo crear un bot inteligente mediante el simulador Sabberstone. Así como, los proyectos de docencia en el ámbito de la inteligencia artificial.

5.3.2 ESTUDIAR LAS TÉCNICAS DE IA A USAR

En este paso del proyecto se profundizará la comprensión de técnicas de inteligencia artificial que se pueden aplicar en la tarea. Se dedicará una mayor atención al estudio de técnicas de aprendizaje por refuerzo, puesto que, esta será la técnica usada para el bot definitivo en el proyecto.

5.3.3 INSTALACIÓN Y ESTUDIO DE SABBERSTONE

A continuación, para iniciar la implementación del trabajo se comenzará a trabajar con el simulador Sabberstone; herramienta que se usará como entorno para la competición. Este simulador ha sido usado en competiciones de Hearthstone anteriores, pero actualmente, no se encuentra actualizado. Por ello, el estudio del simulador es de vital importancia tanto para la creación del Bot como para una correcta implementación del posterior desafío.

5.3.4 INSTALACIÓN Y ESTUDIO DE FIREPLACE

Por los problemas generados en implementación de Sabberstone, así como, la posibilidad de utilizar un simulador de Python que permite un acceso mayor para la comunidad que el lenguaje de C#. Este simulador llamado Fireplace ha sido usado en proyectos de entornos de inteligencia artificial con Gymnasium. Se instalará y estudiará este simulador para el resto del proyecto.

5.3.5 IMPLEMENTACIÓN DE ENTORNO CON GYMNASIUM

A continuación, se creará un entorno que permita un acceso más sencillo a los estudiantes, en el que se pueda tomar contacto con los principales algoritmos estudiados sin tener que desarrollarlos desde cero. De esta forma, se puede conseguir un formato gradual del aprendizaje con una parte práctica a lo largo del proceso lectivo.

5.3.6 IMPLEMENTACIÓN DE UN PRIMER BOT INTELIGENTE

A continuación, se creará un Bot inteligente. Este bot deberá ser capaz de superar al bot aleatorio en porcentaje de victorias.

5.3.7 IMPLEMENTACIÓN DE UN BOT DEFINITIVO MEDIANTE RL

Se culminará el desarrollo de Bots diseñando un último modelo con técnicas de aprendizaje con refuerzo. El objetivo es que este bot tenga mayor porcentaje de victorias

que los bots anteriormente creados. Además, se pretende que este modelo sea usado en desafío cooperativo.

5.3.8 COMPARACIÓN DEL BOT CON OTROS BOTS PUBLICADOS

Después, de comparar el bot definitivo con aquellos más sencillos creados previamente, se comparará el modelo con otros algoritmos que hayan sido publicados. Este paso permitirá comparar y valorar la efectividad del Bot creado en este proyecto, así como, de otros que ya hayan sido publicados.

5.3.9 CONSTRUIR Y REDACTAR EL DESAFÍO

El último paso en el proyecto consistirá en construir un desafío que sea implementable en asignaturas que traten el tema de la inteligencia artificial. El diseño de este desafío tendrá como objetivo mejorar la participación y motivación del alumnado.

5.3.10 REDACCIÓN DE LA MEMORIA DEL TFG

Para la presentación del proyecto y su posterior evaluación se redactará una memoria que exponga y analice el trabajo y los resultados conseguidos durante el proyecto. Además, esta memoria servirá como documentación del proyecto, proporcionando un contexto y una guía apoyo para la implementación de este proyecto en un curso de inteligencia artificial.

5.4 PLANIFICACIÓN

En este apartado exponemos un diagrama de Gantt que se ha usado como base para el desarrollo de las tareas del proyecto. Se ha utilizado una metodología SCRUM por sprints que ha permitido adaptarse a los cambios que han sido necesarios durante el trabajo como el cambio de simulador o la implementación de un entorno con Gymnasium.

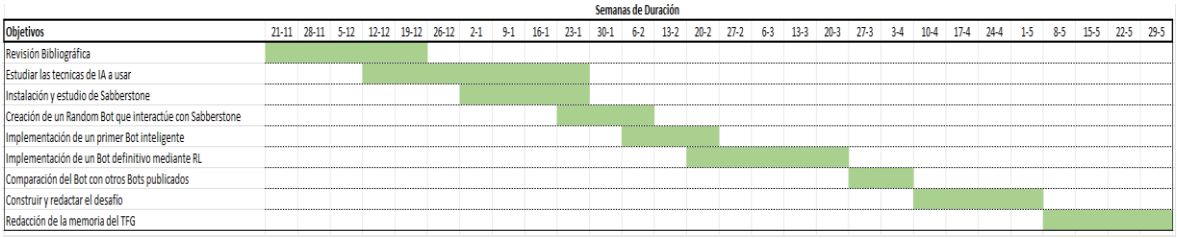


Ilustración 13. Diagrama de Gantt del proyecto

5.5 ESTIMACIÓN ECONÓMICA

La estimación económica realizada mediante un análisis de gastos tiene como objetivo, guiar en una futura implementación. De esta forma, este desglose de gastos incurridos puede servir para la creación de un presupuesto de un proyecto similar.

El principal coste de este proyecto es el trabajo de horas realizado, que puede asemejarse al trabajo realizado por un desarrollador de software junior. Tomando un salario medio de esta posición de 2000€ al mes, tenemos un coste por hora de 12,5€.

Se estima que en este trabajo ha necesitado alrededor de 10 horas a la semana durante 9 meses, 400 horas para su elaboración y, por tanto, el coste inducido por la mano de obra sería de un total de 5000€.

A pesar de no ser contingente para la realización de este proyecto, es recomendable el uso de herramientas que agilicen y mejoren la capacidad del trabajo como Google Collab o Kraggle para ejecutar procesos en GPUs o el uso de herramientas de proyectos agiles como Notion o Trello. La suscripción de Google Collab es de 11,9€ al mes, el coste total de esta suscripción sería de 120€ para los 10 meses del proyecto, alrededor de un 2.5% del coste de la mano de obra. Por otro lado, existen una gran variedad de herramientas para el desarrollo ágil gratuitas.

Capítulo 6. SISTEMA/MODELO DESARROLLADO

6.1 ANÁLISIS DEL SISTEMA Y CARACTERÍSTICAS RELEVANTES

A lo largo de este proyecto, se ha desarrollado un entorno del videojuego Hearthstone dónde se puedan aplicar algoritmos de inteligencia artificial. El objetivo de este proyecto es de carácter docente y, por tanto, se pretende realizar una implementación sencilla para futuros estudiante que estén introduciéndose en la materia.

Para desarrollar el entorno se ha hecho uso de tres herramientas diferenciadas; el simulador del videojuego Hearthstone llamado Fireplace, la librería Stable Baselines para la implementación de algoritmos de aprendizaje por refuerzo, la librería Gymnasium para crear una estructura sencilla del entorno que permita la interacción entre Fireplace y Stable Baselines, así como, las futuras implementaciones de inteligencia artificial que se puedan plantear.

El simulador de Fireplace está desarrollado en Python, esto ofrece una gran ventaja para el proyecto porque permite fusionar el simulador con la librería para aplicar algoritmos de aprendizaje por refuerzo. Sin embargo, los simuladores de Hearthstone, como en el caso de Fireplace, no están actualizados y contienen algunos fallos en la implementación de cartas. Este hecho se debe a que Hearthstone es un juego que actualiza sus cartas tres o cuatro veces al año y, la opción de un simulador de código abierto que se encuentre al día con el juego se vuelve un proyecto de extremada dificultad.

6.2 CAMBIOS EN FIREPLACE Y STABLE BASELINES

Durante el desarrollo del proyecto se ha intentado evitar en la medida de lo posible los cambios en las herramientas externas por dos simples razones. Primero por conseguir una

unificación del código y la herramienta dónde se pueda encontrar toda la posible información en el mismo repositorio. Y segundo, por mantener una sencillez de instalación y de uso, para los estudiantes a los que está dirigido el proyecto, de los cuales se asume, que se están iniciando en la materia de la inteligencia artificial.

Por el lado de Fireplace, se ha mantenido todo el código intacto, aunque se ha usado algunas funciones retocadas que se han implementado en el entorno, entre ellas, la separación de cartas implementadas y no implementadas. Este código de la herramienta Fireplace permite eliminar las cartas no implementadas y prevenir errores que surgen del simulador.

Por el lado de Stable Baselines, este proyecto se ha centrado en la implementación del algoritmo DQN por su valor en términos de sencillez y potencia. Para la implementación de este algoritmo se tuvo que corregir una función que procesaba el espacio de observación. Dado que el espacio de observación del videojuego es altamente complejo fueron necesarios unos cambios en esta función de la librería que se han mantenido con la mayor sencillez posible.

6.3 DESARROLLO DE UN ENTORNO CON GYMNASIUM

6.3.1 EXPLICACIÓN DEL ESPACIO DE OBSERVACIÓN

El espacio de observación del videojuego Hearthstone es muy extenso. En rasgos generales, la mayor parte de la información de la partida es observable y tiene los mismos campos para ambos jugadores. También hay información no observable que define el estado del juego y, además, existe una alta complejidad debido a las habilidades únicas de las cartas que se encuentran en el juego. Por tanto, para realizar un análisis exhaustivo del espacio observación, lo dividiremos en información observable y no observable. También, haremos una sección específica para desglosar la complejidad de las cartas de Hearthstone.

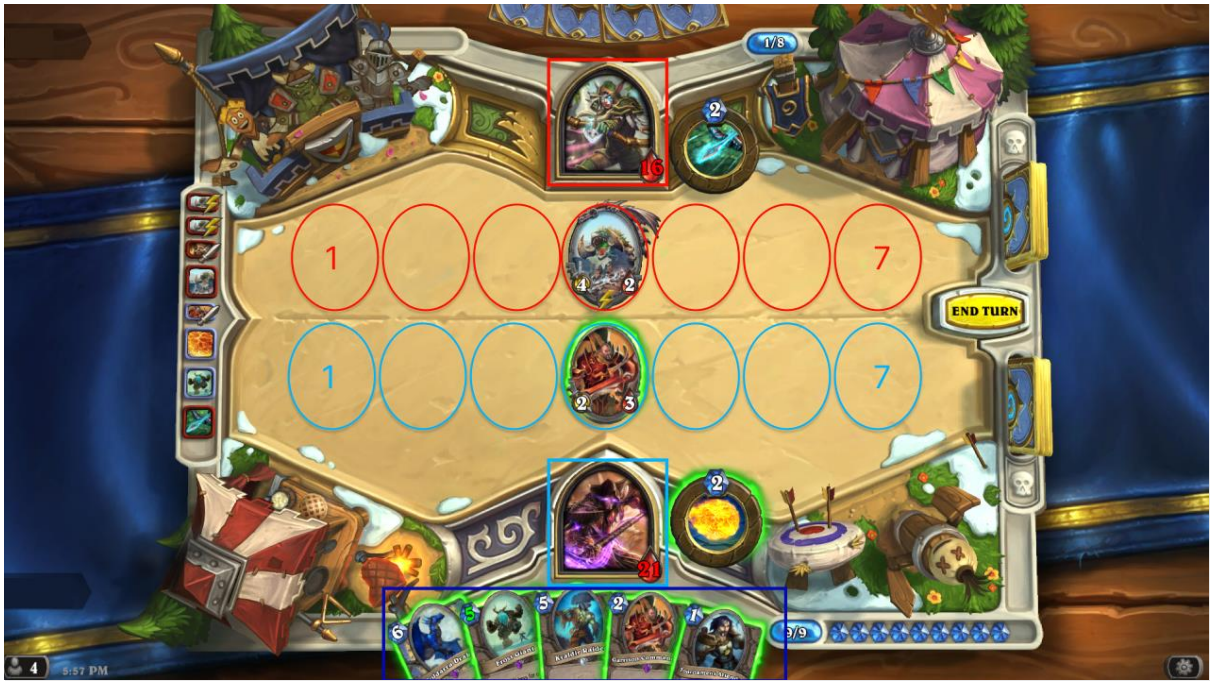


Ilustración 14. Personajes y cartas de una partida de Hearthstone.

- Espacio Observable:
  - Héroes: cada jugador puede elegir entre nueve héroes distintos (2 variables con 9 opciones).
    - Los poderes de héroe están asociados a los héroes, pero, es posible que estos se mejoren durante la partida, en total, 36 opciones que, por el momento, no están implementadas en el entorno.
    - Por otro lado, la vida y la armadura que acotaremos en 30 puntos como máximo (4 variables 30 opciones)
  - Mana: el maná total de cada jugador puede variar entre uno y diez cristales. Además, durante el turno se podrá observar también el maná utilizado (4 variables con 10 opciones).
  - Cantidad de cartas en la mano: la cantidad de cartas en la mano puede variar entre cero y diez cartas para ambos jugadores (2 variables con 11 opciones).

- Cantidad de cartas en el mazo: la cantidad de cartas en el mazo puede variar entre cero y treinta cartas para cada jugador (2 variables con 31 opciones)
- Acciones "posibles": cómo se puede ver en la imagen, las acciones que se pueden ejecutar se destacan en verde. Cada esbirro en el campo de batalla, cada carta en la mano, el héroe y el poder de héroe tienen una opción binaria que indica si la acción se puede realizar o no (19 variables con 2 opciones).
- Cartas
  - Hay más de 26 mil cartas en el simulador Fireplace, sin embargo, teniendo las solo las cartas que están correctamente implementadas nos quedamos con más de 8 mil cartas.
  - Además, las cartas implementadas tienen una variable de coste que puede tomar hasta 10 valores y, al menos, una característica que puede tomar cualquier valor positivo (ataque, vida, durabilidad, poder de hechizos), también lo acotaremos a 20.
  - En los esbirros en el campo de batalla sí podemos hablar de ataque, defensa y coste. (7 variables con 10 opciones, otras 7 con 20 opciones y otras 7 con 8 mil opciones)
  - Cartas Propias (10 variables con 8 mil opciones)

Podemos asegurar, por tanto, que el espacio de observación de este videojuego es mayor que 10^{103}.

- Espacio no observable
  - Cartas del oponente: las cartas que tiene el oponente en la mano no son visibles para el jugador.
  - Cartas de ambos mazos: ambos mazos son opacos y no sé puede saber el orden en el que se encuentran las cartas.

6.3.2 EXPLICACIÓN DEL ESPACIO DE ACCIÓN

Para definir el espacio de acciones estudiaremos cada uno de los tipos de acciones:

- Elección: Las acciones de elección obligan al jugador a escoger entre dos otre opciones (3 acciones).
- Poder de Héroe: Esta acción se puede usar por sí solo o aplicarla a cada uno de los 16 personajes que vemos en la ilustración 13 (17 acciones).
- Atacar con esbirros: Un esbirro puede atacar a los 8 personajes enemigos y se pueden controlar hasta 7 esbirros en el campo de batalla (56 acciones)
- Atacar con el héroe: Al igual que los esbirros el héroe puede atacar a los 8 personajes enemigos (8 opciones)
- Jugar cartas desde la mano: Las cartas pueden no tener efecto, o tener un efecto aplicable a cualquiera de los 14 personajes, por tanto, nos encontramos con 15 acciones por carta (150 acciones).
- Terminar turno: Esta es la acción que realiza el jugador cuando da por terminado su turno. Al terminar un turno, comienza el turno del oponente. (1 acción)

En total tenemos 227 acciones en el entorno. El problema con el entorno consiste en hacer un mapeo de las acciones elegidas por el modelo y las acciones posibles. Se puede encontrar las definiciones de los subespacios que se han usado como espacios de acciones en este proyecto.

6.3.3 ANÁLISIS Y COMPARACIÓN DE LOS ESPACIOS

Hemos visto que el espacio de observación de este entorno está acotado inferiormente por 10^{103}.Para comparar la complejidad de este entorno, juegos famosos que han supuesto un gran desafío para la inteligencia como el ajedrez (menor que 10^{12}). De hecho, el espacio de acción es mayor que el número de átomos en el universo (10^{80})

6.3.4 IMPLEMENTACIÓN DE PARÁMETROS

Para dar mayor complejidad al entorno y acercarlo en mayor mediad al juego original se han implementado una variedad de parámetros. Estos parámetros permiten crean entornos distinguidos para probar la capacidad de los algoritmos y mejorar el aprendizaje de distintas formas. Los parámetros que se han implementado son los listados a continuación:

6.3.4.1 Action type

Con este argumento se puede elegir dos espacios de acción distintos.

- Random: Esta opción permite crear un entorno en el que el agente actúe de forma aleatoria.
- Type: Esta opción separa las acciones en seis grupos de sub-acciones. El agente elegirá un grupo de sub-acciones y se lo pasará como entrada al entorno. El entorno comprobará si esa acción es legal o ilegal. La sub-acción será legal si existe alguna acción dentro de esta, que sea posible ejecutar en el estado actual, en caso contrario, será una sub-acción ilegal. Si la sub-acción es legal, se cogerán todas las acciones que pertenecen a esta y se realizará una de ellas aleatoriamente. Si la sub-acción es ilegal, el entorno no ejecutará ninguna acción y devolverá la misma observación.

6.3.4.2 Reward mode

Este argumento permite elegir distintas formas de aplicar la puntuación.

- Simple: la puntuación será +1 cuándo el agente gane una partida y -1 cuándo pierda (esta puntuación es episódica).
- Penalize: Este método aplicará la puntuación “Simple” y, además, devolverá una puntuación de -0.01 en cada iteración que el agente escoja una acción ilegal.
- Incentive: Este método aplicará la puntuación “Simple” y, además, devolverá una puntuación de +0.01 en cada iteración que el agente escoja una acción legal.

- Complex: Este método aplicará la puntuación "Simple", "Penalize" y "Incentive" juntas.

6.3.4.3 Opponent model

Con este argumento se puede elegir el modelo que se enfrentará al agente. Por defecto, nuestro agente se enfrentará a un oponente aleatorio.

- Random: El oponente del agente será un agente aleatorio.
- Ruta del modelo: este argumento permitirá que el oponente sea un modelo entrenado previamente.

6.3.4.4 Decks

Con este argumento se puede elegir con qué tipo de mazos jugaran los agentes. Por defecto, el entorno usará los mazos clásicos.

- Classic: Mazos predeterminados con una basados en una estrategia.
- Random: Mazos creado aleatoriamente.

6.3.4.5 Seed

Con este argumento se puede elegir una semilla. La semilla es de vital importancia para comparar métodos distintos.

- Int: Se puede elegir una semilla para la generación de números aleatorios en el entorno.

6.3.4.6 Ejemplo de código

Este código permite ejecutar 100 mil iteraciones de un algoritmo DQN usando subacciones y mazos clásicos.

```
Steps = 100000 env = gym.make('Hearthstone-v1', action_type="type",
reward_mode="simple", decks="classic")
```

```
model = DQN("MultiInputPolicy", env, batch_size=1, verbose=1)
model.learn(total_timesteps=steps, log_interval=4)
env.display_stats()
model.save("dqn_type_100k") env.close()
```

6.4 APLICACIÓN A LA DOCENCIA

En la actualidad, la gamificación se ha convertido en una herramienta fundamental para potenciar el aprendizaje en diversas áreas, y el campo de la inteligencia artificial no es una excepción. La incorporación de elementos lúdicos y mecánicas de juego en el proceso de enseñanza de la inteligencia artificial tiene un impacto significativo en el compromiso y la participación de los alumnos. Al gamificar el aprendizaje de la inteligencia artificial, se crea un entorno interactivo y estimulante que fomenta la curiosidad y la experimentación activa por parte de los estudiantes. A través de desafíos, recompensas y competencias, los alumnos se sienten motivados a explorar conceptos complejos y a desarrollar habilidades prácticas en la implementación de algoritmos y modelos de IA. Esta metodología no solo les permite adquirir conocimientos teóricos, sino que también promueve el pensamiento crítico, la resolución de problemas y el trabajo en equipo, habilidades esenciales en el mundo actual. Al involucrar a los alumnos de manera activa y divertida, la gamificación del aprendizaje de la inteligencia artificial se convierte en un factor clave para un proceso educativo enriquecedor y efectivo.

En esta sección se va a detallar el proyecto docente propuesto que se articula en torno al entorno desarrollado en este documento. El propósito es presentar a los alumnos un desafío que irán mejorando a lo largo del curso. El primer día de clase ya debería ser capaces de entregar y validar su primer agente y obtener una primera posición en el ranking general. A partir de ese momento, los alumnos irán mejorando esos agentes a medida que el temario les vaya "desbloqueando" nuevos conocimientos que podrán implementar en sus agentes para obtener mejoras en las métricas de victorias.

6.4.1 PRE-REQUISITOS

Para afrontar con éxito un curso de estas características se requerían los siguientes conocimientos previos:

- Conocimiento sólido de programación en Python, incluyendo el manejo de estructuras de datos, funciones y bucles
- Estar familiarizado con conceptos básicos de aprendizaje automático, como algoritmos de optimización y técnicas de regresión y clasificación
- Conocimientos de matemáticas y estadística, ya que estos conceptos son esenciales para comprender las ecuaciones que soportan los algoritmos de Reinforcement Learning.
- Habilidad en el manejo de entornos de desarrollo como Jupyter Notebook, PyCharm, Visual Code o similar para escribir y ejecutar el código.
- Sería muy recomendable además tener experiencia previa en el uso de bibliotecas populares de aprendizaje automático y procesamiento de datos, como NumPy y Pandas, así como, conocimientos de programación orientada a objetos para poder entender el entorno.

6.4.2 COMPETENCIAS

El proyecto de aprendizaje debería permitir adquirir las siguientes competencias generales:

- Adquisición de conocimientos sólidos y tecnologías relacionadas que permitan el aprendizaje autónomo de nuevos métodos y herramientas, y la adaptación a entornos en constante evolución.
- Habilidad para abordar problemas complejos, tomar decisiones con iniciativa, ser creativo y comunicar de manera efectiva conocimientos, habilidades y destrezas, demostrando un sólido entendimiento de la responsabilidad ética y profesional en el ámbito del Reinforcement Learning.

Y las siguientes competencias específicas:

- Capacidad de programación aplicada a algoritmos y aplicaciones de Reinforcement Learning en entornos Python.
- Dominio de las técnicas y metodologías fundamentales en Reinforcement Learning para el diseño e implementación de sistemas de toma de decisiones automatizados.
- Conocimiento profundo de los entornos Gym y otras herramientas relacionadas, y habilidad para utilizarlos en la resolución de problemas de Reinforcement Learning.

6.4.3 RESULTADOS DE APRENDIZAJE

El resultado del mecanismo propuesto debería proporcionar al alumno los siguientes resultados de aprendizaje:

- Comprender las bases teóricas y conceptuales del Reinforcement Learning, identificando sus principios fundamentales y sus aplicaciones en diversos campos.
- Desarrollar habilidades para diseñar, implementar y evaluar algoritmos de Reinforcement Learning, considerando aspectos como la exploración y explotación, la función de recompensa y el aprendizaje a través de la interacción con el entorno.
- Aplicar técnicas de Reinforcement Learning para resolver problemas específicos, como la toma de decisiones secuenciales, la planificación y control de sistemas autónomos, y la optimización de políticas en entornos complejos.
- Evaluar y comparar el rendimiento de distintos algoritmos y enfoques de Reinforcement Learning, utilizando métricas adecuadas y métodos de experimentación rigurosos.
- Comunicar de manera clara y efectiva los resultados obtenidos, tanto de forma oral como escrita, demostrando un sólido dominio de los conceptos y técnicas del Reinforcement Learning, y su aplicación en casos prácticos.

6.4.4 BLOQUE TEMÁTICO

La extensión del temario depende fuertemente del grado y del curso al que se oriente una experiencia docente de este tipo. A continuación, se proporciona un posible temario orientado a alumnos en segundo o tercero de un Grado en Ingeniería de Telecomunicaciones o Ingeniería Matemática:

1. Introducción al Reinforcement Learning y entornos Gym:
   - Conceptos básicos de Reinforcement Learning.
   - Presentación del entorno Gym y sus características.
   - Agentes, entornos y recompensas.
2. Exploración y explotación:
   - Dilema de exploración y explotación.
   - Métodos de selección de acciones: ε-greedy, Annealing, UCB, entre otros.
   - Balanceando la exploración y la explotación en el aprendizaje por refuerzo.
3. Políticas y funciones de valor:
   - Definición y tipos de políticas: determinísticas y estocásticas.
   - Funciones de valor: valor de estado, valor de acción y función Q.
   - Relación entre políticas y funciones de valor.
4. Aprendizaje basado en modelos:
   - Aproximación y utilización de la ecuación de Bellman.
   - Aprendizaje mediante métodos de Monte Carlo.
   - Aprendizaje mediante diferencia temporal (TD Learning).
5. Algoritmos clásicos de Reinforcement Learning:
   - Q-learning y su aplicación en entornos Gym.
   - SARSA y su comparación con Q-learning.
   - Control de la tasa de aprendizaje y la exploración.
6. Aprendizaje profundo y Reinforcement Learning:
   - Introducción al aprendizaje profundo (Deep Learning).
   - Aplicación de redes neuronales en Reinforcement Learning.

- Deep Q-learning (DQN) y sus mejoras.

7. Algoritmos avanzados de Reinforcement Learning:
   - Policy Gradient: gradientes de la política y teorema del gradiente.
   - Métodos de búsqueda estocástica: REINFORCE, Actor-Critic, A2C, PPO.
   - Comparación y selección de algoritmos en base a distintos escenarios.
8. Reinforcement Learning en entornos continuos:
   - Espacios de acción y observación continuos.
   - Algoritmos de Reinforcement Learning para entornos continuos: DDPG, TD3, SAC.
   - Funciones de aproximación y redes neuronales en entornos continuos.
9. Aplicaciones avanzadas de Reinforcement Learning:
   - Reinforcement Learning jerárquico y aprendizaje por imitación.
   - Aprendizaje por refuerzo en juegos de estrategia.
   - Otros enfoques y tendencias actuales en Reinforcement Learning.

6.4.5 IMPLANTACIÓN

Para la implementación docente de este entorno, se proponen las siguientes sesiones:

- Sesiones magistrales sobre el aprendizaje por refuerzo
- Sesiones de introducción a los entornos de Gymnasium
- Sesiones de instalación y de prueba del entorno
- Sesiones de aplicabilidad a un ejemplo sencillo de entrenamiento de un agente para un juego con un espacio de acciones y un espacio de observaciones reducido
- Sesiones de explicación de la competición y entorno de validación de los modelos realizados
- Sesiones de presentación de resultados y entrega de premios.

6.4.5.1 Sesiones Magistrales sobre RL

En una sesión magistral sobre aprendizaje por refuerzo diseñada para alumnos que se enfrentan por primera vez a esta materia, el enfoque principal sería brindar una introducción clara sobre los conceptos fundamentales que les permitan, más adelante, afrontarse al desafío práctico.

El objetivo sería proporcionar una comprensión sólida de los principios del aprendizaje por refuerzo y cómo se aplica en el contexto de la inteligencia artificial. La sesión comenzaría con una explicación de conceptos como el de agente, el entorno, las acciones y las recompensas. Se ilustrarían ejemplos concretos y se presentarían casos de uso de aprendizaje por refuerzo en diversos campos, como la robótica y los videojuegos. A medida que avance la sesión, se profundizaría en los algoritmos y las estrategias utilizadas en el aprendizaje por refuerzo, como la política epsilon-greedy y el algoritmo Q-Learning. Se haría especial hincapié en la importancia de la exploración y la explotación, así como en la noción de descuento y el equilibrio entre la recompensa inmediata y la recompensa a largo plazo. Este bloque finalizaría con una sesión interactiva donde los estudiantes tendrían la oportunidad de plantear preguntas y participar en ejercicios prácticos para aplicar los conceptos aprendidos.

6.4.5.2 Sesiones de Instalación del entorno

El objetivo de esta sesión sería guiarlos en el proceso de configuración y preparación de un entorno de trabajo adecuado. La sesión comenzaría con una introducción a los requisitos de software y librerías necesarios para trabajar con Python y OpenAI Gimnasyum. A continuación, se proporcionarían instrucciones detalladas sobre cómo instalar y configurar estos componentes, asegurándose de que los estudiantes comprendan cada paso. Se presentarían alternativas y opciones según las preferencias y el sistema operativo de cada estudiante. Durante la sesión, se abordarían posibles desafíos y soluciones comunes que puedan surgir en el proceso de instalación. Además, se brindarían recursos adicionales y

enlaces útiles para facilitar la comprensión y el acceso a la documentación relevante. Al finalizar la sesión, se realizarían ejercicios prácticos sencillos para verificar que el entorno de desarrollo se haya configurado correctamente, permitiendo a los alumnos ejecutar y familiarizarse con los entornos Gymnasium en su propio sistema.

6.4.5.3 Sesiones de Introducción a Entornos Gymnasium

El enfoque principal de este módulo sería familiarizar a los alumnos con los conceptos y la implementación práctica de estos entornos. La sesión comenzaría con una descripción general de los entornos Gymnasium, resaltando su importancia y utilidad como herramienta para desarrollar y probar algoritmos de aprendizaje por refuerzo. Se explicarían los componentes esenciales de un entorno Gymnasium, como los estados, las acciones y las recompensas. A medida que avance la sesión, se presentarían ejemplos de entornos Gymnasium populares y ampliamente utilizados, como el CartPole y el MountainCar, que permitirían a los estudiantes comprender cómo se definen y configuran estos entornos. Se proporcionaría una visión general de las características y limitaciones de los entornos, así como de las métricas de evaluación utilizadas en el aprendizaje por refuerzo. Además, se presentarían recursos y bibliotecas disponibles para trabajar con los entornos Gymnasium.

6.4.5.4 Sesiones prácticas sobre el uso de entornos Gymnasium

La sesión consistiría en ejercicios prácticos en los que los alumnos tendrían la oportunidad de interactuar con un entorno Gymnasium sencillo, con un espacio de observaciones y de acciones limitado. El objetivo es que los alumnos prueben a implementar sus primeras versiones de agentes de aprendizaje por refuerzo.

6.4.5.5 Sesiones de explicación de la competición y entorno de validación

El propósito de este proyecto es gamificar el aprendizaje de las técnicas de RL y para ello se plantea una competición en grupos de dos personas. En la competición el objetivo

principal consistirá en maximizar una serie de métricas en las que la más importante podría ser el número de victorias del modelo. Otro ejemplo de métricas que demostrarían la adquisición de los conocimientos por parte de los alumnos sería maximizar el número de victorias relativas al número de pasos realizados en el entrenamiento, significando que el modelo realizado tiene una mayor capacidad de aprendizaje además de suponer una menor huella de carbono al requerir menos aporte energético para su entrenamiento. Otra métrica que podría ser considerada es el modelo con menor consumo de memoria que maximice el número de victorias.

Dentro de esta sesión se explicaría el mecanismo de entrega, para ello se debería constituir un entorno de evaluación que enfrente los agentes realizados por los alumnos contra una baseline potente que permita generar un ranking de agentes en función de las métricas consideradas. Se recomienda que cada grupo pueda entregar su agente tantas veces como considere o incluso ver la posición que ocupa en el ranking general para que sirva de motivación adicional a mejorar su modelo.

6.4.5.6 Sesiones de presentación de resultados

En último lugar es de vital importancia cerrar la competición con presentaciones de resultados para que el alumno sea consciente del impacto de su trabajo y para que los modelos más eficientes. Se recomienda a ser posible, hacer una presentación para cada tarea realizada durante el curso.

6.4.6 EVALUACIÓN

Para evaluar la adquisición de las competencias y los resultados de aprendizaje propuestos anteriormente se propone el siguiente sistema de evaluación dividida en una parte individual y una parte grupal:

1. Evaluación Individual:

- Exámenes escritos: Evaluación teórica mediante exámenes escritos que aborden conceptos fundamentales del Reinforcement Learning y su aplicación.
- Trabajos individuales: Los alumnos deberán realizar trabajos individuales sobre temas específicos relacionados con el Reinforcement Learning, como la investigación de un algoritmo avanzado, la revisión de un artículo científico o la implementación de un agente en un entorno Gymnasium Sencillo. Se evaluará la comprensión de los conceptos, la capacidad de investigación y la presentación del trabajo.

2. Evaluación Grupal:
   - Desarrollo de un desafío: Los estudiantes se agruparán y deberán desarrollar un proyecto práctico que aplique técnicas de Reinforcement Learning a un videojuego específico utilizando el entorno Gymnasium. En este caso el entorno propuesto en este trabajo. La presentación del desafío se realizará a principio de curso de manera que los alumnos pueden ir creando agentes desde el primer día de clase y mejorándolo con los conocimientos que vayan adquiriendo. Se evaluará la planificación, implementación y resultados obtenidos, así como la creatividad y originalidad en la solución propuesta.
   - Presentación y defensa del proyecto: Cada grupo realizará una presentación oral de su proyecto ante el resto de la clase. Se evaluará la claridad, coherencia y calidad de la presentación, así como la capacidad de comunicación y argumentación. Además, se llevará a cabo una sesión de preguntas y respuestas para evaluar la comprensión profunda del proyecto.

Capítulo 7. ANÁLISIS DE RESULTADOS

7.1 ASPECTOS DESTACADOS

Durante todo el desarrollo e implementación del proyecto, se consiguieron importantes avances en el diseño del entorno que permitirá entrenar agentes inteligentes basados en diferentes técnicas de aprendizaje por refuerzo. En esta sección se muestran algunos experimentos preliminares que permiten establecer unas *baselines* para su aprovechamiento académico.

7.2 RESULTADOS DEL BOT ALEATORIO

Para comenzar y valorar el estudio de modelos de inteligencia artificial es de vital importancia estudiar el comportamiento de un agente aleatorio, puesto que, este supone una implementación muy sencilla y más liviana a nivel computacional.

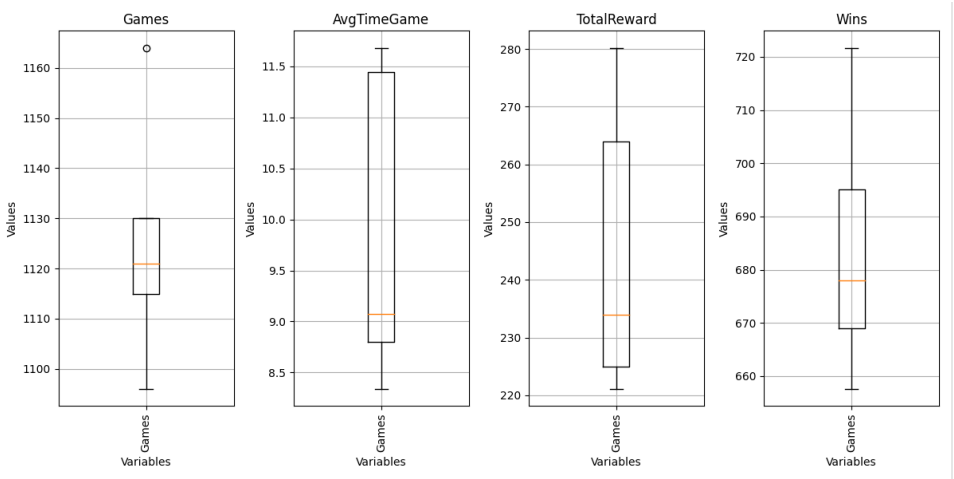


Ilustración 15. Diagrama de cajas, de un bot random.

En este caso, se ha realizado 5 pruebas de 100 mil iteraciones dónde el modelo aleatorio ha conseguido una ratio de victorias del 60.09%. Ha tardado una media de 10 segundos en ejecutar cada partida, consiguiendo ejecutar alrededor de 1120 partidas (~3horas) en cada ejecución. El valor medio de la puntuación conseguida durante el proceso ha sido 235.

Sorprendentemente, a pesar de la sencillez de este método, funciona muy bien en muchos juegos de estrategia que, en general, pretenden enganchar a los usuarios generando formas sencillas de ganar. En relación con el Hearthstone, hay varias mecánicas sencillas que pueden marcar una diferencia abismal entre ganar y perder victorias, sin embargo, para jugadores experimentados que ejecutan estas estrategias sin fallos tienen mucho más valor otras mecánicas más complejas que pueden torcer la balanza a su favor.

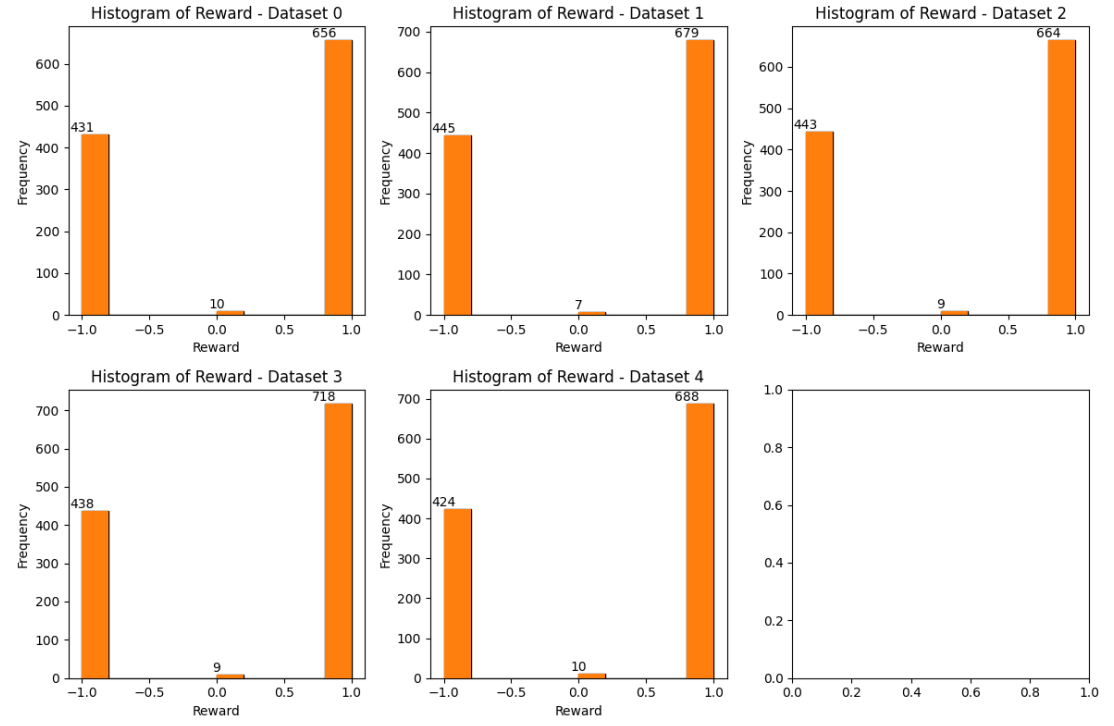


Ilustración 16. Puntuación obtenida por un bot Random.

Como se observa en la *ilustración 17*, el agente recibe una puntuación positiva en todas las ejecuciones. En este caso, el agente recibía 1 punto en los episodios que terminaban victoria,

perdía un punto en los que terminaban en derrota y obtenía 0 o 0.1 puntos cuando ocurría un error o un empate en el episodio.

7.3 RESULTADOS DE LOS BOTS "TYPE"

El bot aleatorio se caracteriza por tener un espacio de acción mínimo (1 acción). El entorno con el que estamos trabajando tiene 227 acciones y la complejidad de que dependiendo del estado las acciones que se pueden ejecutar cambian. El desarrollo del bot "Type" reduce el espacio de acciones a 6 ordenando las acciones en 6 grupos o tipos (los tipos de acción vienen explicados en la sección 6.3.2). De esta forma, se ha podido estudiar algunos modelos distintos al aleatorio.

Se ha optado por centrarse en este espacio de acciones por tres limitaciones. Primero, el desarrollo de un espacio de acciones completo resulta altamente complejo pues se debe comprobar las acciones propuestas por el agente con las acciones posibles. Segundo, hay acciones mucho más comunes que otras lo que provocaría que el agente valúe peor las acciones que aparecen con menor asiduidad. Tercero se necesitaría muchas más iteraciones para que la red neuronal converja.

7.3.1 PUNTUACIÓN SIMPLE Y MAZOS CLÁSICOS

Todos los modelos de esta sección han sido estudiados con 5 ejecuciones de 100 mil iteraciones, probando distintos argumentos. Para comenzar estudiaremos una puntuación simple como la del bot aleatorio y el uso de mazos preestablecidos.

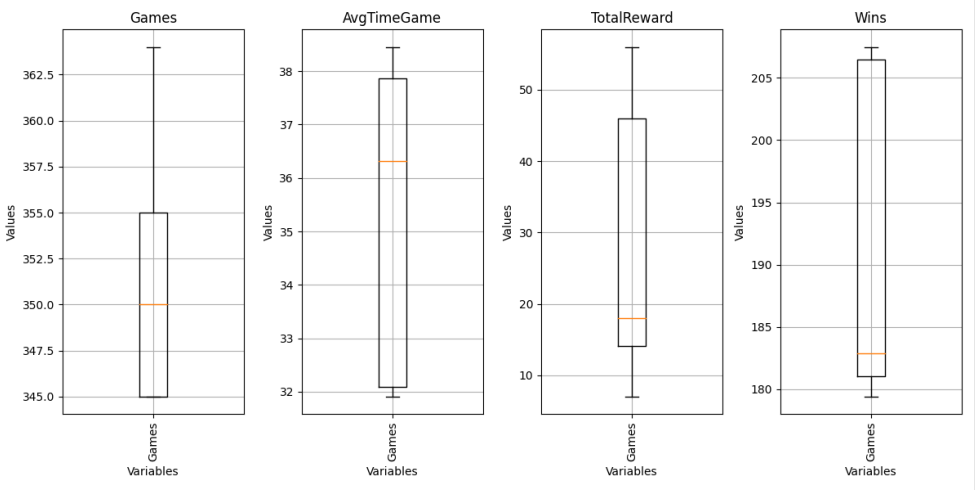


Ilustración 17. Diagrama de cajas de un modelo con argumentos Type, Simple, Classic.

En la *ilustración 18* podemos ver que el número medio de partidas jugadas es 350, un 31% de las partidas jugadas por el bot aleatorio, además, tarda una media de 36 segundos por partida que es más de tres veces el valor de la sección anterior (~3.5 horas). La puntuación media es de 18 y 183 la media de victorias. El porcentaje de victorias es de 52.3%.

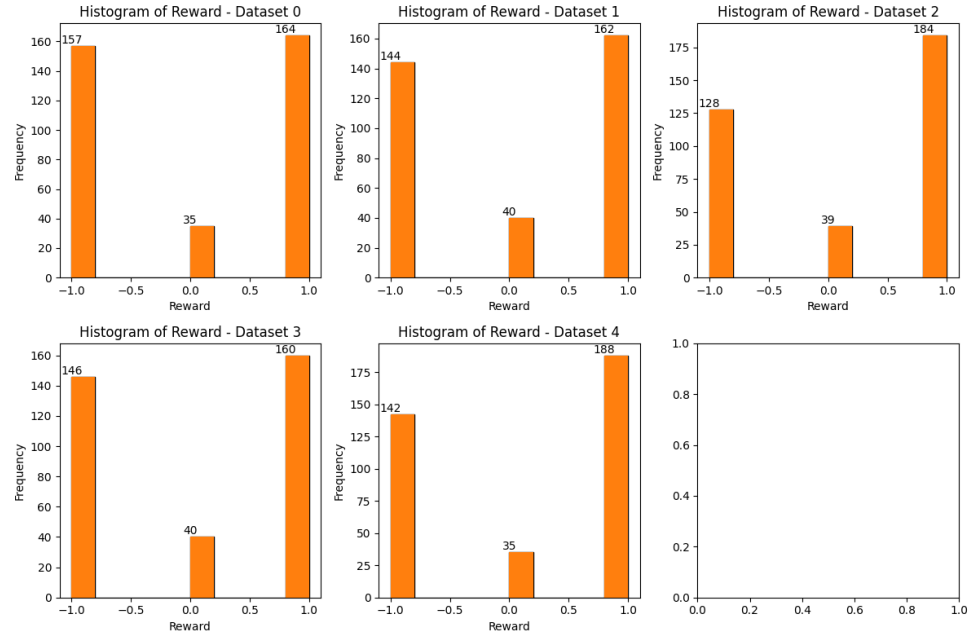


Ilustración 18. Puntuación obtenida por un modelo con argumentos Type, Simple, Classic.

En este caso vemos que la puntuación es superficialmente mayor que cero. También, cabe destacar, el aumento de los errores que surgen, en su mayoría, porque se necesitan muchas más iteraciones por episodio y hay un número de iteraciones máximas establecido. Los errores suponen aproximadamente un 10%.

7.3.2 PUNTUACIÓN COMPLEJA Y MAZOS ALEATORIOS

En este caso compararemos la versión "Type" del espacio de acciones con puntuación compleja. En la puntuación compleja el entorno premiará que el agente escoja una acción que se pueda ejecutar en el entorno, una acción legal y castigará con puntuación negativa las acciones que sean ilegales en el estado actual. El valor predeterminado de las acciones legales e ilegales es 0.1 puntos en valor absoluto.

El número medio de partidas es 382, el tiempo medio 59,3 segundos (~6.3 horas), la puntuación -370 y el número de victorias 210. Como podemos ver, al modelo le cuesta aprender la diferencia entre acciones legales e ilegales y eso supones una gran disminución de la puntuación. El porcentaje de victoria es 55.0%

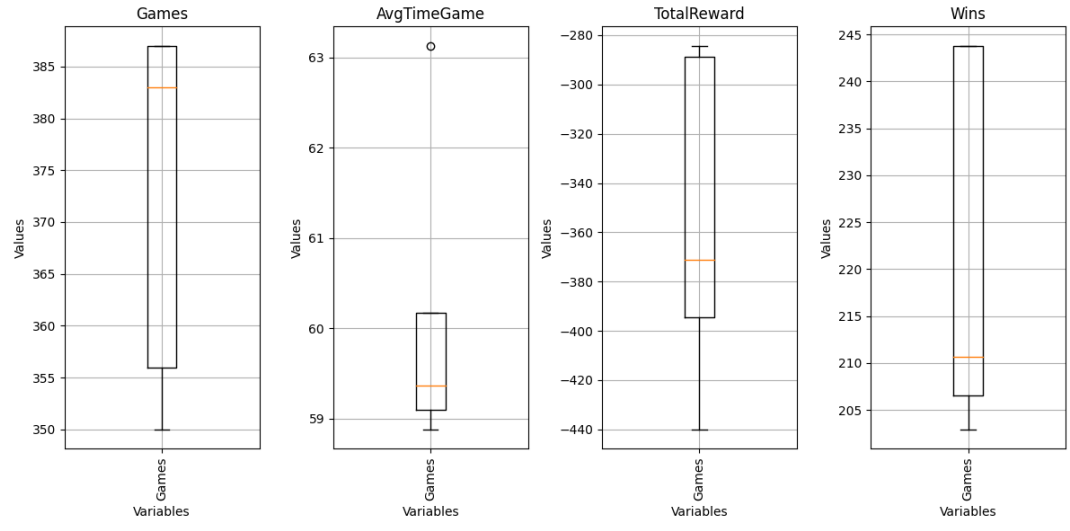


Ilustración 19. Diagrama de cajas de un modelo con argumentos Type, Complex, Random.

En la *ilustración 21*, se puede observar un suceso muy interesante. En este caso, el modelo suele mantener una puntuación media durante los 200 primeros episodios y, a partir de este momento, empieza a recibir puntuaciones muy negativas. Sería interesante ver si esta tendencia converge o se recupera después de más episodios. Sin embargo, uno de los grandes problemas es el aumento sustancial del tiempo que tarda en ejecutarse el episodio cuándo el modelo elige una gran cantidad de acciones ilegales.

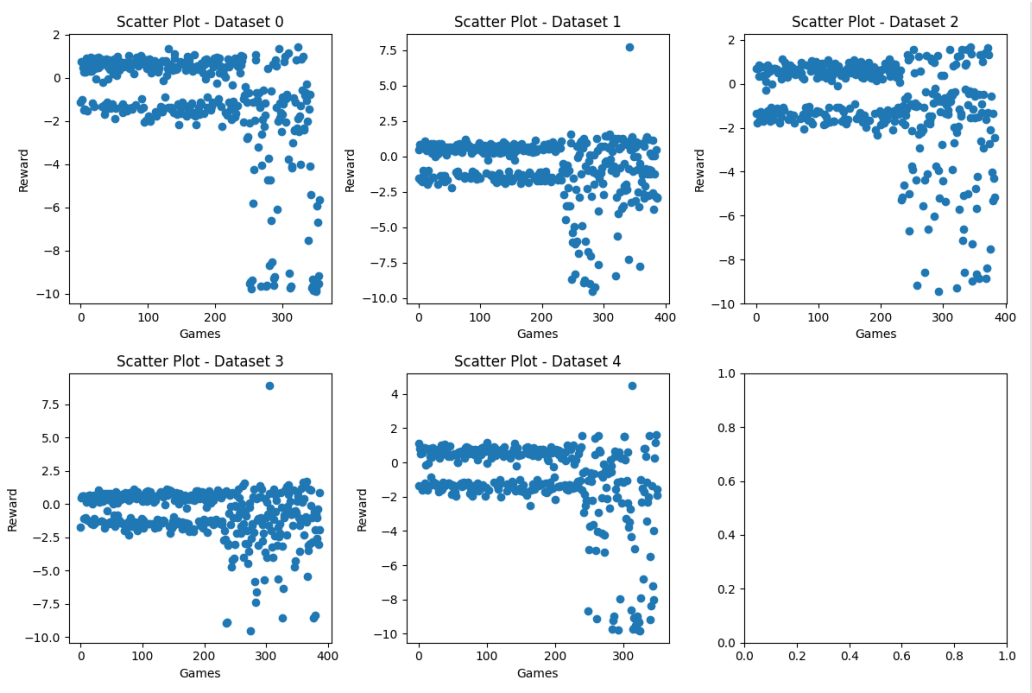


Ilustración 20. Puntuación obtenida por un modelo con argumentos Type, Complex, Random.

7.3.3 PUNTUACIÓN COMPLEJA Y MAZOS CLÁSICOS

En esta sección estudiaremos como influyen los cambios en la puntuación que reciben los agentes. Estudiaremos varios modelos "Type" con mazos clásicos para distintos valores absolutos de la puntuación (0.01, 0.005, 0.0005).

7.3.3.1 Valor absoluto de la puntuación (0.01)

En el primer caso observamos una media de 455 partidas, con 42,2 segundos de tiempo medio (~5.3 horas). La puntuación media es -405 y el número de partidas 240. El porcentaje de victorias es 52.7%.

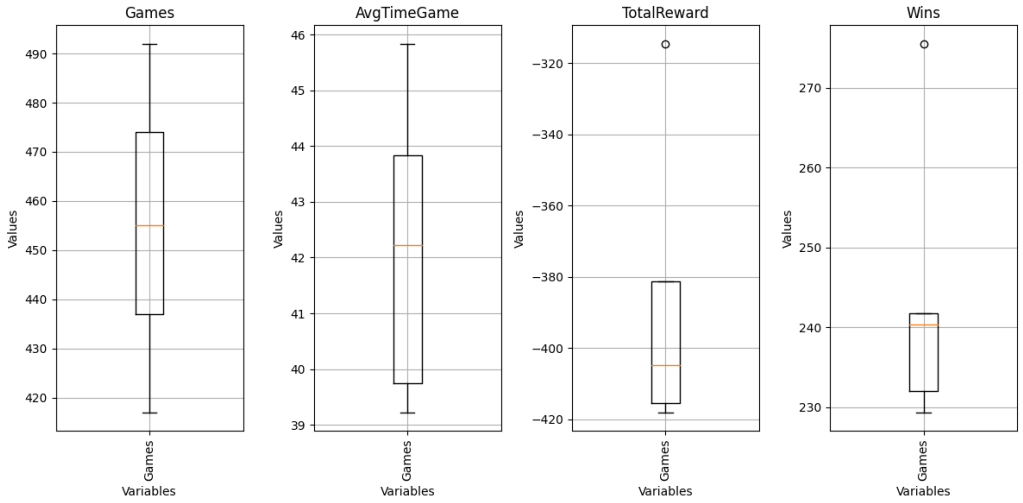


Ilustración 21. Diagrama de cajas de un modelo con argumentos Type, Complex (0.01), Classic.

En el caso de la puntuación seguimos viendo una tendencia clara a obtener una puntuación muy baja a partir de los 300 episodios.

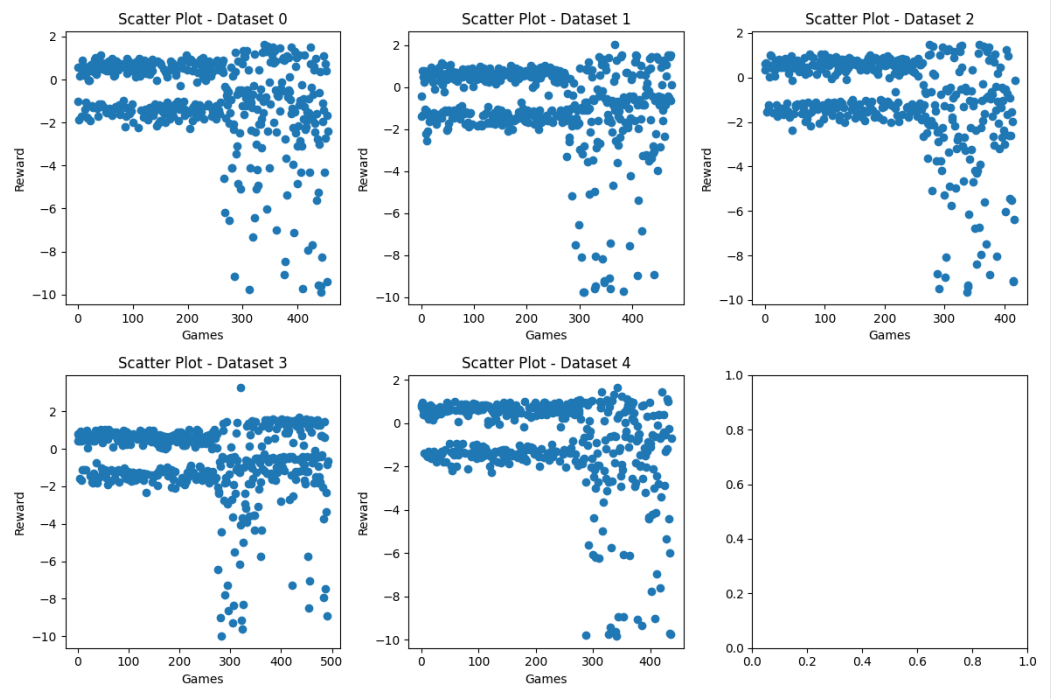


Ilustración 22. puntuación obtenida por un modelo con argumentos Type, Complex (0.01), Classic.

7.3.3.2 Valor absoluto de la puntuación (0.005)

En el segundo caso observamos una media de 380 partidas, con 51,2 segundos de tiempo medio (~ 5,4 horas). La puntuación media es -218 y el número de partidas 208. El porcentaje de victorias es 54.7%.

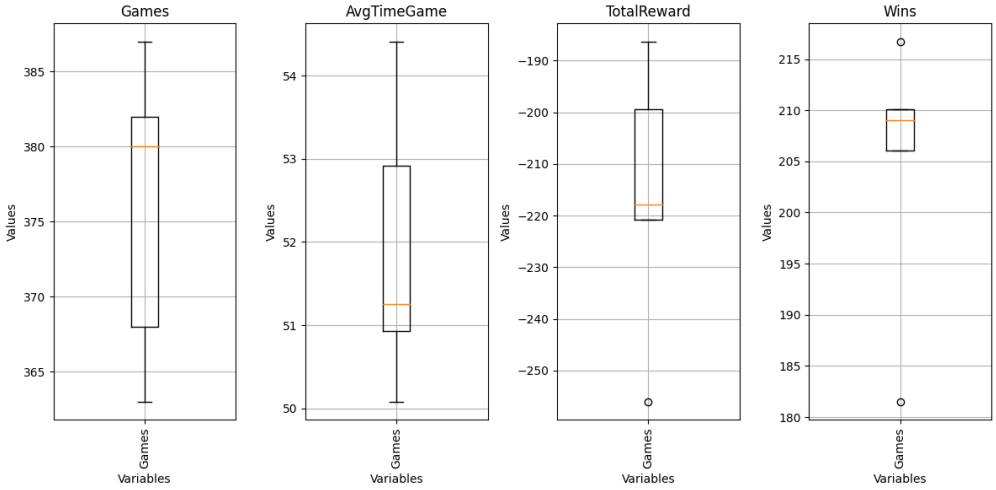


Ilustración 23. Diagrama de cajas de un modelo con argumentos Type, Complex (0.005), Classic

Aunque la tendencia sigue siendo equiparable a los casos anteriores, reducimos el valor negativo de los episodios con menor puntuación, es decir, les damos menos importancia. Esto en comparación se traduce en una pequeña mejora de la ratio de victorias, sin embargo, el tiempo de ejecución aumenta.

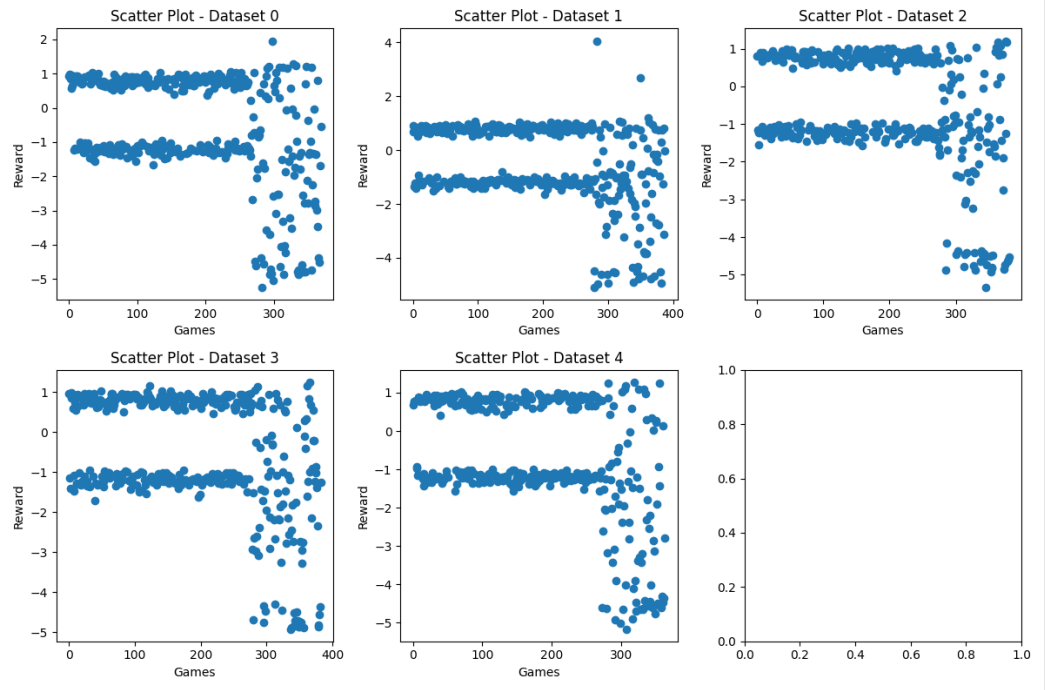


Ilustración 24. Puntuación obtenida por un modelo con argumentos Type, Complex (0.005), Classic.

7.3.3.3 Valor absoluto de la puntuación (0.0005)

En el primer caso observamos una media de 340 partidas, con 58.7 segundos de tiempo medio (~5.6 horas). La puntuación media es 50 y el número de partidas 198. El porcentaje de victorias es 58.2%. Como desventaja este modelo es el más costoso en términos de inferencia, pero, esta pequeña modificación de la puntuación ayuda con la ratio siendo la mejor opción tras el bot random.

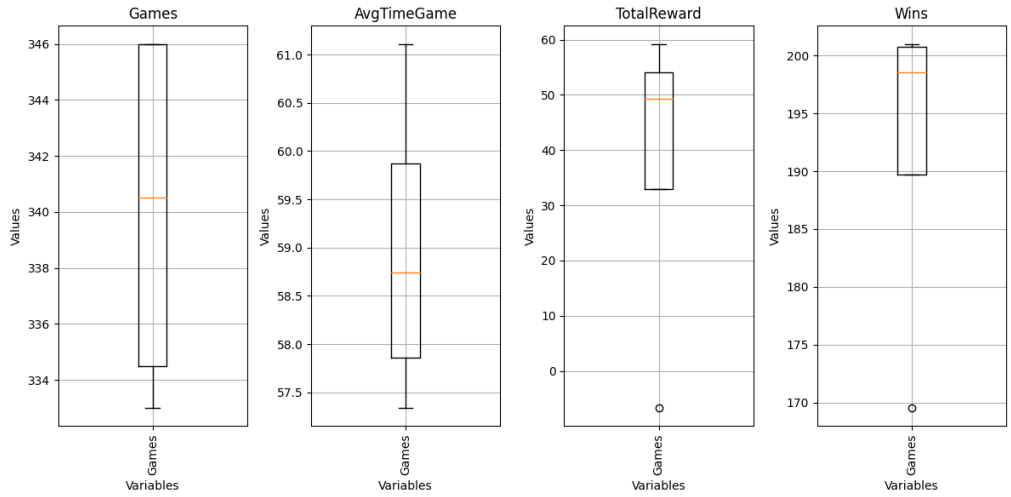


Ilustración 25. Diagrama de cajas de un modelo con argumentos Type, Complex (0.0005), Classic.

En el siguiente gráfico vemos variabilidad del porcentaje de victoria en uno de los casos. También, es interesante el elevado número de errores que se siguen sucediendo en el entorno.

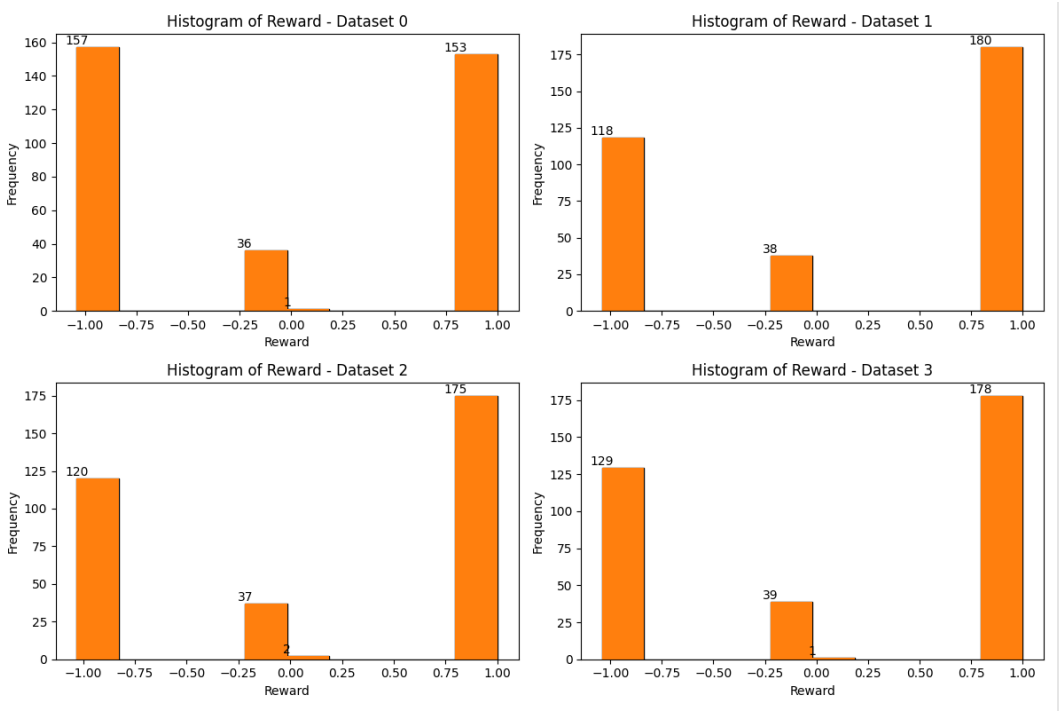


Ilustración 26. Puntuación obtenida por un modelo con argumentos Type, Complex (0.0005), Classic.

7.3.4 ESTUDIO DE LAS SEEDS

Por último, se ha realizado un estudio de la variabilidad del entorno, midiendo la diferencia de los resultados con distintos tipos de semillas para los algoritmos de generación de números aleatorios. Hemos utilizado 7 semillas distintas para realizar este estudio y un número de iteraciones que nos han permitido valorar entre 80 y 90 partidas por ejecución.

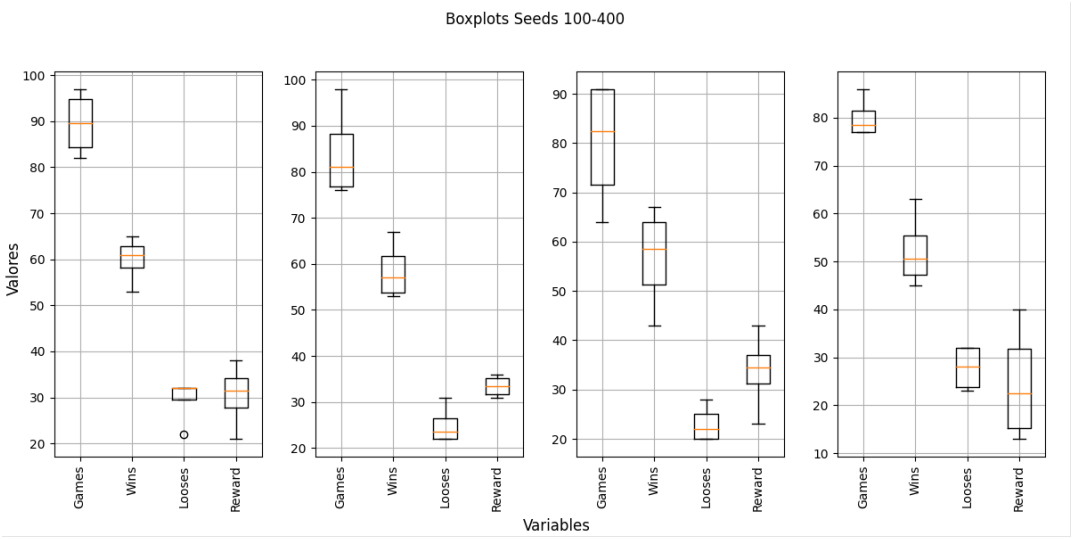


Ilustración 27. Diagrama de cajas de entornos con valores de Seed, 100, 200, 300 y 400.

Vemos que el número de victorias varía siendo este mayor en los casos en los que hay un mayor número de partidas, tendencia que se repite para las derrotas. Con este estudio podemos extrapolar un error aproximado de 5 partidas cada 90, en términos porcentuales estaríamos ante un 5,5%. La puntuación se mantiene superior a 30 para los casos de 90 partidas y entre 20-30 puntos para el resto de casos.

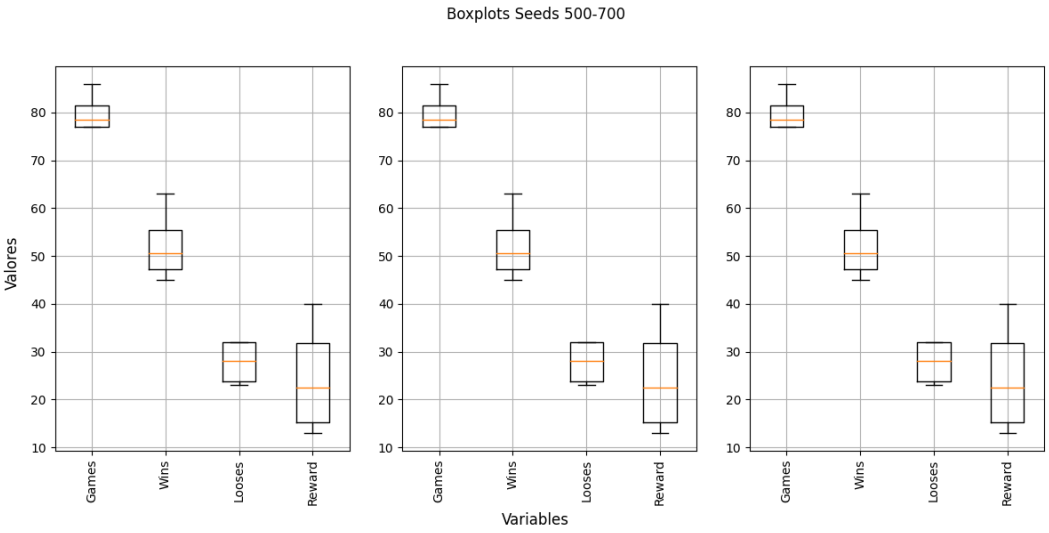


Ilustración 28. Diagrama de cajas de entornos con valores de Seed, 500, 600, 700.

Capítulo 8. CONCLUSIONES Y TRABAJOS FUTUROS

En trabajos futuros se recomienda investigare esta lista de posibilidades:

- Otros algoritmos complejos: Como la Política de gradiente y el PPO.
- Hacer pruebas con el espacio de acciones completo.
- Disminuir espacio de observación
- Parametrizar la exploración y explotación
- Probar nuevos mazos
- Enfrentar modelos contra otros y versiones previas de sí mismo.
- Probar puntuación ligadas a estrategias (Agresivas, Rango Medio y Control)

8.1 OTROS ALGORITMOS COMPLEJOS: COMO LA POLÍTICA DE GRADIENTE Y EL PPO.

Existen diversos algoritmos que pueden ser de gran ayuda para encontrar modelos más adecuados al entorno y al juego de Hearthstone. Entre ellos, destaca la política de gradiente, que resulta especialmente útil gracias a sus acciones de distribución estocástica. Este enfoque evita llamar al modelo cada vez que se elige una acción que no se puede ejecutar en el entorno. La política de gradiente permite adaptar de manera eficiente las decisiones del agente a las circunstancias cambiantes del juego, optimizando así su rendimiento y capacidad de respuesta. El estudio de otros algoritmos puede ser una interesante vertiente de estudio.

8.2 HACER PRUEBAS CON EL ESPACIO DE ACCIONES COMPLETO.

El uso de un espacio de acciones completo puede llegar a tener gran impacto en el desarrollo de modelo. Sin embargo, para el estadio de este proyecto y sin tener un algoritmo que pueda

generar distribuciones probabilísticas de acciones a tomar, resulta extremadamente ineficiente.

8.3 DISMINUIR ESPACIO DE OBSERVACIÓN

Del mismo modo que se ha reducido el espacio de acción, también, se puede probar a reducir el espacio de observación. Actualmente, el espacio de observación es muy grande y, es posible, que mucha de la información contenida en él no sea de gran importancia. Esto puede hacer que se generé ruido, impidiendo y atrasando la convergencia del modelo.

8.4 PARAMETRIZAR LA EXPLORACIÓN Y EXPLOTACIÓN

Una vez contenidos los problemas que pueden surgir por la falta de convergencia del agente actualmente y, con la capacidad computacional suficiente, se debería hacer un estudio de la exploración y explotación parametrizando el argumento épsilon.

8.5 PROBAR NUEVOS MAZOS

Otra forma de aumentar la complejidad del modelo y de prevenir el overfitting, puede ser la introducción de mazos nuevos. Algunos podrían usarse solo en la fase de inferencia. Nuevos mazos implican nuevas cartas y estrategias que los agentes deben aprender. Además, al usar los mazos previos que ya han sido estudiados deberán desarrollar estrategias personalizadas cuando se enfrente a los mazos nuevos para aumentar el número de victorias.

8.6 ENFRENTAR MODELOS CONTRA OTROS Y VERSIONES PREVIAS DE SÍ MISMO.

Un recurso típico del aprendizaje por refuerzo es enfrentar el modelo contra sí mismo utilizando versiones previas del modelo como oponente. Se recomienda encarecidamente el

uso de esta técnica porque permite enfrentarse a un modelo más realista que el bot aleatorio, así como, que el algoritmo aprenda a vencer, explotar y defenderse de sus propios errores. Nuevamente, para ejecutar este recurso satisfactoriamente será necesario mejorar los recursos computacionales y la latencia de los modelos.

8.7 PROBAR PUNTUACIÓN LIGADAS A ESTRATEGIAS (AGRESIVAS, RANGO MEDIO Y CONTROL)

Un recurso que ha demostrado ser eficiente es adaptar el sistema de puntuación a las estrategias de cada mazo (Dockhorn, 2019), usando un tipo de puntuación distinta para los mismos estados y creando variaciones en la política según el mazo.

8.8 PROBAR CON REPETICIONES DE PARTIDAS

Implementar la posibilidad de entrenar usando repeticiones de partidas puede ser complicado de desarrollar, sin embargo, se ha visto que puede ser muy eficiente para simuladores y juegos dónde el desarrollo de la partida puede generar una gran latencia (Dockhorn, 2019).

BIBLIOGRAFÍA

BAFTA. (17 de 11 de 2022). *Awards BAFTA*. Obtenido de 2015 Games: https://awards.bafta.org/award/2015/games

Barto, R. S. (2018). *Reinforcement Learning An Introduction.* Bradford Books.

Baselines, S. (2023). *Stable baselines*. Obtenido de Github: https://github.com/DLR-RM/stable-baselines3/tree/master

Blizzard. (2 de 11 de 2021). *Hearthstone Blizzard*. Obtenido de Year of the Phoenix in review: https://hearthstone.blizzard.com/en-us/news/23625669/year-of-the-phoenix-in-review

Burguillo, J. C. (2010). Using game theory and competition-based learning to stimulate student motivation and performance. *Computers & education*, 566--575.

Carpio Cañada, J. a. (2015). Open classroom: enhancing student achievement on artificial intelligence through an international online competition. *Journal of Computer Assisted Learning*, 14--31.

Chesani, F. a. (2017). A game-based competition as instrument for teaching artificial intelligence. En Springer, *Conference of the Italian Association for Artificial Intelligence* (págs. 72--84).

Dicheva, D. a. (2015). Gamification in education: A systematic mapping study. *Journal of educational technology & society*, 75--88.

Dockhorn, A. a. (2019). Introducing the hearthstone-ai competition. *arXiv preprint arXiv:1906.04238*.

Dormann, A. R. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 1-8.

Fandom, H. (2023). *Gameplay*. Obtenido de Hearthstone Fandom: https://hearthstone.fandom.com/wiki/Gameplay

Farama. (2023). *Basic Usage*. Obtenido de Gymnasium Farama: https://gymnasium.farama.org/content/basic_usage/

Ferrucci, D. a.-C. (2010). Building Watson: An overview of the DeepQA project. *AI magazine*, 59--79.

Fireplace. (2021). *Fireplace*. Obtenido de Github: https://github.com/jleclanche/fireplace/tree/master

Fontaine, M. C. (2019). Mapping hearthstone deck spaces through map-elites with sliding boundaries. En *Proceedings of The Genetic and Evolutionary Computation Conference* (págs. 161--169).

haker, N. a. (2013). The turing test track of the 2012 mario ai championship: entries and evaluation. *IEEE*, (págs. 1--8).

HearthSim. (2023). *Python hearthstone*. Obtenido de Github: https://github.com/HearthSim/python-hearthstone

Hingston, P. (2010). A new design for a turing test for bots. *IEEE*, (págs. 345--350).

Hoover, A. K. (2020). The many ai challenges of hearthstone. *KI-Kü}nstliche Intelligenz*, 33--43.

HSReplay. (17 de 11 de 2022). Obtenido de HSReplay: https://hsreplay.net/

Hsu, F.-H. (2002). *Behind Deep Blue: Building the computer that defeated the world chess champion.* Princeton University Press.

IEEE. (17 de 11 de 2022). *IEEE Organization*. Obtenido de Conferences: https://www.ieee.org/conferences/index.html

Jesuslln. (2023). *Gymnasium Hearthstone*. Obtenido de Github: https://github.com/jesuslln/gymnasium_hearthstone_env

Lee, J. J. (2011). Gamification in education: What, how, why bother? *Academic exchange quarterly*.

Levinovitz, A. (2014). The mystery of Go, the ancient game that computers still can't win. *Wired Magazine*.

OVGU. (2 de 12 de 2020). *OVGU*. Obtenido de Computational Intelligence in Games: https://www.is.ovgu.de/Teaching/SS+20/CIG.html

Ribeiro, P. a. (2009). Teaching artificial intelligence and logic programming in a competitive environment. *Informatics in Education*, 85--100.

Russell, S. a. (1994). *Artificial Intelligence: A Modern Approach, Fourth Edition.* Prentice Hall.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 210--229.

Schaeffer, J. a. (1996). Chinook the world man-machine checkers champion. *AI magazine*, 21--21.

Schaeffer, J. a. (2007). Checkers is solved. *science*, 1518--1522.

Silver, D. a. (2016). Mastering the game of Go with deep neural networks and tree search. *nature*, 484--489.

Stanley, K. O. (2005). Real-time neuroevolution in the NERO video game. *IEEE transactions on evolutionary computation*, 653--668.

Summerville, A. J. (2016). Mystical tutor: A magic: The gathering design assistant via denoising sequence-to-sequence learning. En *Twelfth artificial intelligence and interactive digital entertainment conference.*

Tesauro, G. (1991). Practical issues in temporal difference learning. *Advances in neural information processing systems*.

Tesauro, G. a. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 58--68.

TheGameAwards. (17 de 11 de 2022). *The Game Awards*. Obtenido de Year 2014: https://thegameawards.com/rewind/year-2014

Vinyals, O. a. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 350--354.

w. (w). *w*. w: w.

Yellorambo. (2021). *Tier list of classic decks*. Obtenido de Hearthstone Top Decks: https://www.hearthstonetopdecks.com/hearthstone-classic-tier-list-ranking-the-best-classic-decks/

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

INTRODUCCIÓN

Los Objetivos de Desarrollo Sostenible (ODS) son una serie de 17 metas establecidas por las Naciones Unidas en 2015 como parte de la Agenda 2030 para el Desarrollo Sostenible. Estos objetivos abordan los desafíos más apremiantes a los que se enfrenta el mundo en términos de desarrollo económico, social y ambiental, y buscan guiar a los países hacia un futuro más sostenible.

Los ODS abarcan una amplia gama de temas, que incluyen la erradicación de la pobreza, la igualdad de género, la educación de calidad, la acción climática, la protección del medio ambiente, la salud y el bienestar, la paz y la justicia, entre otros. Cada objetivo tiene metas específicas y se basa en los logros y desafíos del anterior conjunto de objetivos conocidos como los Objetivos de Desarrollo del Milenio (ODM).

Ilustración 29. Objetivos de Desarrollo Sostenible (ODS), Naciones Unidas

2. *APOYO DE LOS OBJETIVOS*

El proyecto desarrollado se alinea, principalmente, con los objetivos de educación de calidad y reducción de las desigualdades.

2.1 EDUCACIÓN DE CALIDAD

Este proyecto tiene como objetivo esencial crear una metodología de docencia para estudiantes. Pretende crear una educación extensa y diversa para estudiantes de ámbito universitario. Aunque este objetivo suele fijarse en educación esencial, no conviene olvidar la importancia de la educación superior y la investigación.

Ilustración 30. Educación de calidad (ODS 4).

2.2 REDUCCIÓN DE LAS DESIGUALDADES

Otra de las características esenciales de este trabajo es el desarrollo de herramientas de código abierto. La educación superior suele suponer mayores dificultades de acceso. El carácter público y gratuito de este proyecto promueve que universidades y estudiantes con acceso a un ordenador puedan encontrar herramientas de docencia para ampliar sus conocimientos.

Ilustración 31. Reducción de las desigualdades (ODS 10).

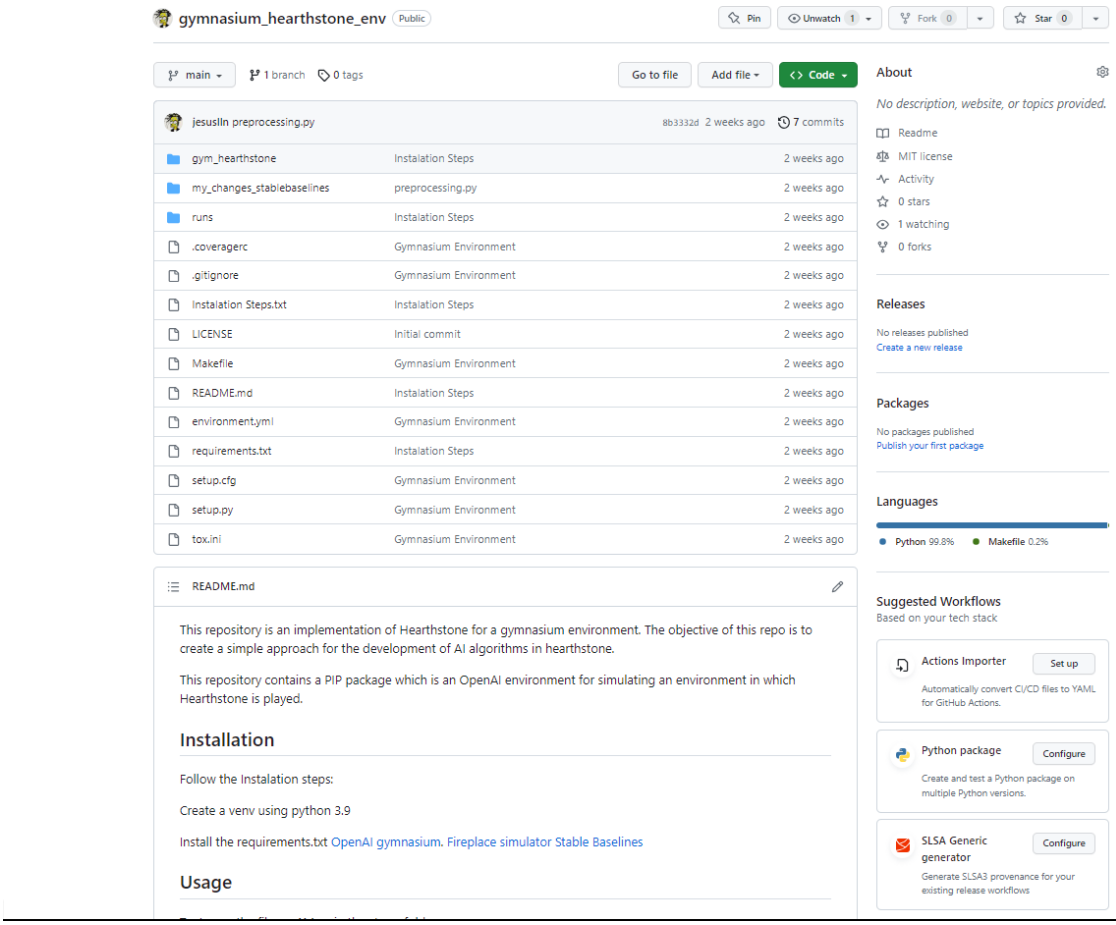
3. POSIBLE INTERFERENCIA CON LOS OBJETIVOS

Este proyecto de investigación se alinea en esencia con los ODS como hemos comentado en la sección anterior. Sin embargo, cabe recordar el alto consumo que suponen los modelos de inteligencia de artificial y nuestro consumo de energía. Esto no nos debería frenar a seguir investigando, pero, debería hacernos replantearnos los usos que le damos a los modelos de inteligencia artificial, así como, la investigación que trata de crear modelos eficientes que sirvan para democratizar las herramientas y mejorar el consumo que suponen.

Ilustración 32. Acción por el clima (ODS 13)

ANEXO II: REPOSITORIO DEL PROYECTO

Para más información sobre el proyecto y el código utilizado se puede acceder al repositorio de Github. En este, se puede encontrar una guía de instalación, una wiki y el código fuente del proyecto (Jesuslln, 2023).



Además, para los mazos utilizados en los modelos de este proyecto, he escogido mazos clásicos por su sencillez y número de cartas implementadas. Si desea ahondar en los mazos implementados puede revisar esta página (Yellorambo, 2021).