



UNIVERSIDAD DE MÁLAGA



Ingeniería del Software

Aprendizaje por Refuerzo en un Juego de Navegador Web

Reinforcement Learning in a Web Browser Game

Realizado por
Zoilo Hernández Duarte

Tutorizado por
Gabriel Jesús Luque Polo
José Ángel Morell Martínez

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, Enero 2023



UNIVERSIDAD
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

**Aprendizaje por Refuerzo en un Juego de Navegador
Web**

Reinforcement Learning in a Web Browser Game

Realizado por
Zoilo Hernández Duarte

Tutorizado por
Gabriel Jesús Luque Polo
José Ángel Morell Martínez

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, DICIEMBRE DE 2022

Fecha defensa: enero de 2023

Resumen

En la actualidad, la Inteligencia Artificial (IA) es un área multidisciplinaria de la ciencia cuyo objetivo es que los sistemas o máquinas puedan realizar operaciones que se consideran propias de la inteligencia humana e incluso puedan mejorar iterativamente a partir de la información que recopilan. Entre las muchas aplicaciones que tiene la IA se encuentra su aplicación en videojuegos. Los avances de esta disciplina en juegos tan complejos como el Go demuestran el potencial de esta rama para encontrar los mejores cursos de acción para cada situación.

Así, el objetivo de este proyecto es el desarrollo de juegos web, en los que se produzca un aprendizaje gradual por parte de los actores en el mismo, implementando correctamente algoritmos de aprendizaje por refuerzo, los cuales observarán la situación actual del juego en cada momento, y decidirán la mejor acción a realizar en cada una de estas.

Los juegos a implementar en este caso son el Pong (junto con alguna variación del mismo) y el tres en raya. Además, la aplicación web tendrá otras funcionalidades añadidas como acelerar el proceso de aprendizaje, descargar un documento que informa sobre el progreso de aprendizaje de los actores e incluso la posibilidad de descargar el modelo de aprendizaje del jugador en un momento determinado; así como poder cargarlo a un jugador en una iteración nueva.

Por último, se realizará una recolección de datos de las recompensas y castigos otorgados en distintas ejecuciones de dichos juegos para comprobar que los algoritmos se hayan implementado correctamente, así como poder comprobar su correcto funcionamiento, viendo si la tendencia de la recompensa media es crecer a lo largo del tiempo.

Palabras clave: Aprendizaje por refuerzo, Aplicación web, Javascript, Pong, Tres en raya.

Abstract

Nowadays, Artificial Intelligence (AI) is a prominent field of research in different areas, games among them. The advances these techniques achieve in games as complex as Go show their potential when trying to find the best action for every possible situation.

This project aims to create web browser games where the players go through a progressive learning process, correctly implementing reinforcement learning algorithms, which will observe the game's current state at that moment and decide the best action to take in every case.

Two basic games will be implemented in this final degree project: Pong (along with some variations) and Tic Tac Toe. The web application will also have some additional features: speeding up the learning process, downloading a text document reflecting the average reward throughout the process and the possibility of downloading the current learning model of one of the players, and allowing to load one as well so that a player can use the experience of that model immediately.

Lastly, there is a gathering of data consisting of an average of the rewards of punishments awarded in different executions of those games to check that the algorithm has been correctly implemented, as well as checking that it works correctly, observing if it trends upwards.

Keywords: Reinforcement learning, Web application, Javascript, Pong, Tic Tac Toe.

Índice

Resumen.....	1
Abstract	2
Índice.....	3
Introducción.....	5
1.1 Motivación.....	6
1.2 Objetivos	6
1.3 Metodología de trabajo.....	7
1.4 Estructura de la memoria.....	8
Antecedentes y Tecnologías	11
2.1 Antecedentes	12
2.2 Aprendizaje por refuerzo.....	13
2.2.1 Elementos del aprendizaje por refuerzo	16
2.2.2 Algoritmo Q-Learning	17
2.2.3 Algoritmo SARSA.....	18
2.2.4 Cuantificación	19
2.3 Tecnologías usadas	19
2.3.1 JavaScript	20
2.3.2 Phaser3	21
2.3.3 HTML	22
Análisis y Modelado.....	23
3.1 Análisis.....	24
3.1.1 Entorno para algoritmos de aprendizaje por refuerzo en el navegador web.....	24
3.1.2 Juego del Pong	25
3.1.3 Tres en raya.....	27
3.2 Modelado	28
Implementación	33
4.1 Desarrollo de los juegos Pong	33
4.1.1 Desarrollo del juego Pong principal	33
4.1.2 Variación del juego Pong.....	35
4.2 Desarrollo del juego tres en raya	36
4.3 Creación del algoritmo de aprendizaje por refuerzo	38
4.3.1 Implementación del algoritmo de aprendizaje en Pong.....	39
4.3.2 Implementación del algoritmo de aprendizaje por refuerzo en el tres en raya ..	42
4.4 Creación de la aplicación web y funcionalidades extras	42
Análisis Experimental.....	45
5.1 Pruebas del Pong	46
5.1.1 Pruebas en Pong con tamaño fijo	46
5.1.2 Pruebas de Pong con tamaño variable	48
5.1.3 Pruebas de Pong con el mismo algoritmo	49

5.1.4 Pruebas de Pong con el mismo algoritmo y tamaño variable.....	52
5.1.5 Pruebas de la variación del Pong.....	54
5.2 Pruebas del tres en raya.....	56
Conclusiones y Trabajos Futuros.....	57
6.1 Conclusiones	57
6.2 Trabajos futuros	58
Referencias	59
Manual de Uso	61

1

Introducción

La Inteligencia Artificial (IA), como bien indica su nombre, es toda inteligencia (o comportamiento inteligente) exhibida por entes no vivientes que en la mayoría de las ocasiones en las que se hable sobre los mismos serán referidos como agentes [1].

El término de Inteligencia Artificial surge en 1956, aunque el primer trabajo considerado dentro de este campo fue un modelo de neuronas artificiales de McCulloch y Pitt [2], pero por supuesto existen precedentes que permitieron el uso de computadores para procesar y computar situaciones y procesos complicados.

Hoy en día la Inteligencia Artificial está presente en una gran cantidad de elementos relacionados con el mundo de la tecnología; desde motores de búsqueda a sistemas usados para ajustar la experiencia de uso al usuario (recomendar ciertas cosas en base a sus acciones previas por ejemplo).

Por supuesto, esta tecnología es aplicable también a la toma de decisiones en un estado o situación determinados bajo una serie de reglas; por lo tanto, es obviamente aplicable a los juegos (tradicionales o electrónicos). Un ejemplo claro y bastante famoso en este ámbito sería el superordenador Deep Blue [1], creado específicamente para jugar al ajedrez y que fue el primero en ganar una partida y, más tarde, un encuentro a un campeón del mundo [3].

Asimismo, existen diversos aspectos en los que se puede centrar una IA para mostrar su comportamiento inteligente; algunos de estos serían: razonamiento, planificación, percepción o aprendizaje, siendo este último uno de los más importantes en el desarrollo de una IA [4].

Al mismo tiempo, el aprendizaje es probablemente el aspecto más importante de la IA en el proyecto, existiendo distintas formas de contribuir al proceso. En

nuestro caso; usaremos el aprendizaje por refuerzo [5], en el que un agente toma acciones de modo que siempre maximice una posible recompensa.

1.1 Motivación

Los desarrollos en Inteligencia Artificial y su amplio uso en una gran cantidad de ámbitos en la actualidad, demuestran por completo el gran valor que esta rama de aprendizaje tiene con respecto a las futuras tecnologías y desarrollos tecnológicos.

Cada vez son más los ámbitos y entornos en los que se busca la implementación de tecnologías de aprendizaje con el objetivo de mejorar la efectividad de los productos creados en ellos así como facilitar su uso.

Una de las tecnologías más recientes, que es una inspiración para este proyecto, es OpenAI Gym [17]. Este es una librería de Python que permite implementar algoritmos de aprendizaje por refuerzo y crear entornos con agentes que aprendan.

Al mismo tiempo, en tiempos recientes los navegadores web se han convertido en un entorno en el que es posible utilizar todo tipo de aplicaciones sin instalar otro tipo de herramientas, lo cual las convierte en un tipo de aplicación muy versátil y con pocos requisitos previos para su uso [6]. Entre ellas, el navegador web permite que cualquiera pueda utilizar algoritmos de IA sin requisitos adicionales, y la optimización actual de estos sistemas hace que su rendimiento se parezca cada vez más a las aplicaciones nativas de este tipo.

De esta manera, la motivación principal detrás de este proyecto es la combinación e integración de programas en los que se implemente un aprendizaje, con aplicaciones web; juntando la versatilidad de las aplicaciones web con el gran potencial y utilidad de las tecnologías de la rama del aprendizaje. Para ello, creamos en última instancia unos juegos usando tecnologías web, en los que se pueda producir y denotar un aprendizaje notable por parte de los jugadores/agentes y comprobando el grado de aprendizaje adquirido por estos de forma posterior.

Con todo esto, en definitiva queremos comprobar la viabilidad de unir dos tecnologías importantes en la actualidad de forma eficiente y práctica.

1.2 Objetivos

El objetivo principal de este trabajo fin de grado es la creación de juegos en aplicaciones web con aprendizaje por refuerzo implementado y el posterior análisis de los resultados que se obtienen tras entrenar a los agentes utilizando dicho proceso de aprendizaje. Para conseguir esto; es necesario lograr una serie de objetivos parciales, que en su conjunto nos llevarán a lograr dicho objetivo general:

- **Objetivo 1:** Desarrollo de varios juegos simples de manera que sea posible variar ciertas condiciones del mismo (tamaño, por ejemplo), y que pueda ser jugado por una inteligencia de manera independiente, pero también pueda ser jugado por una persona.

- **Pong:** con algunas variaciones.
 - **Tres en raya**
- Objetivo 2: Desarrollo de una implementación de algoritmos de aprendizaje por refuerzo, así como su integración en los juegos creados anteriormente.
- Objetivo 3: Creación de una aplicación web, que permita las siguientes operaciones sobre el juego:
 - Acelerar el proceso de juego: pudiendo así obtener los resultados de las distintas partidas de forma más eficiente.
 - Detener la aceleración: en caso de que el juego esté acelerado, esta función permitiría detenerlo y volver al ritmo de juego normal.
 - Descargar un historial de recompensas: descarga un archivo de texto en el que se muestren las medias de las recompensas otorgadas a los agentes cada cierto periodo de tiempo.
 - Descargar un modelo de aprendizaje: descarga un archivo de texto que contiene el modelo de aprendizaje de un agente del juego.
 - Cargar un modelo de aprendizaje: carga a un agente del juego un modelo de aprendizaje previamente seleccionado; esto permitirá al agente concreto actuar de acuerdo a dicho modelo.

Cabe destacar, que debido a la creación de varios juegos, es necesario llegar a estos objetivos en cada uno de ellos; en los casos del Pong y sus variaciones, las implementaciones son parecidas, pero con el tres en raya, es necesario cambiar el trabajo realizado un poco, para satisfacer las distintas situaciones que surgen entre ambos casos.

1.3 Metodología de trabajo

A la hora de realizar este proyecto, es necesario seguir una estructura de trabajo para obtener un resultado final que cumpla con los objetivos inicialmente planteados. Y, además, llegar a estos objetivos de una forma eficiente y organizada, facilitando el proceso de realización del proyecto, y mejorando la calidad del producto final.

Por ello, para completar este trabajo, se ha decidido seguir una adaptación a nuestra situación de la metodología Scrum [7]. En resumen, en Scrum existen varias características que son importantes destacar:

- En primer lugar, se debe tener una visión inicial de los requisitos del proyecto para comenzar a trabajarlos, aunque se trabaja bajo el pensamiento de que estos pueden cambiar a lo largo del tiempo. Estos requisitos serían la pila del producto (o *product backlog*).
- Una vez que tenemos esta recopilación inicial de requisitos o historias de usuario, vamos realizando una serie de ellos en cada uno de los ciclos de iteración. El objetivo de cada uno de estos ciclos es tener una parte completa y funcional del trabajo realizada, una especie de entregable. Estos ciclos serán también llamados *sprints* y las características implementadas durante dicho sprint recibirán el nombre de la pila del sprint.
- En Scrum, un aspecto muy importante son los roles asignados a los miembros del equipo, los cuales otorgan unas responsabilidades distintas:

- El **Product Owner** el cual representa los intereses de los potenciales clientes del producto, lo cual le convierte en la parte más importante de la pila del producto.
- El **equipo de desarrollo**, encargado de la realización e integración de los requisitos en cada ciclo de iteración.
- El **Scrum Master**, el cual se encarga de que se siga la metodología Scrum de forma correcta y cada uno de los elementos que intervienen durante el desarrollo sirvan para el beneficio de este.

De esta forma, la metodología que se ha seguido a la hora de desarrollar el proyecto es una adaptación de la metodología Scrum en la que cada ciclo de iteración abarcará uno de los aspectos del proyecto: estos pueden ser terminar uno de los juegos, implementar el aprendizaje en uno de ellos o realizar una recolección de los datos del entrenamiento y posteriormente analizarlos.

Asimismo, los roles que han estado relacionados con el desarrollo de la aplicación han sido asumidos por el propio estudiante ocupando el lugar del equipo de desarrollo.

Por otra parte, los tutores han asistido al estudiante en la toma de decisiones de ciertos aspectos del desarrollo y de algunos requisitos importantes mediante reuniones regulares con el estudiante por lo que podríamos decir que el rol del **Product Owner** ha sido llevado a cabo entre ambos.

Por último, el rol de **Scrum Master** ha sido representado por los tutores que han asegurado que el proceso de realización del proyecto se haya realizado de forma satisfactoria y de acuerdo a las planificaciones establecidas.

1.4 Estructura de la memoria

La memoria recoge toda la información relevante y de importancia del proyecto y está estructurada en los siguientes capítulos:

- **Capítulo 1:** este capítulo sirve como introducción a este trabajo. En él se detalla un poco de información previa sobre la Inteligencia Artificial, así como algunos motivos iniciales para la realización del trabajo y los objetivos a conseguir. También se describe la metodología usada.
- **Capítulo 2:** en este capítulo se explican los antecedentes del proyecto, así como las tecnologías utilizadas para su creación y desarrollo.
- **Capítulo 3:** en este apartado se recogen las fases de análisis y modelado del proyecto, en las que se organizan sus distintas partes y se establece su estructura para conseguir los objetivos del mismo de forma efectiva.
- **Capítulo 4:** en este capítulo se representa la fase de implementación de los juegos y la integración del aprendizaje en los mismos.
- **Capítulo 5:** capítulo en el que se expone el análisis experimental realizado posteriormente a la implementación representando la efectividad del aprendizaje a lo largo de la ejecución de los juegos.
- **Capítulo 6:** capítulo final que recoge las conclusiones sacadas de todo el proceso de realización del proyecto, así como las posibles líneas futuras de trabajo relacionadas con el mismo.

Adicionalmente, se incorpora un manual de usuario (Anexo I).

2

Antecedentes y Tecnologías

En la creación de todo proyecto software es necesario establecer las tecnologías y técnicas que se van a utilizar en el mismo para crear un resultado satisfactorio que cumpla los requisitos establecidos al inicio del trabajo.

Con esto no nos referimos únicamente a la forma de organizar y estructurar el trabajo realizado (desarrollados en el primer capítulo), sino también a las tecnologías utilizadas para la creación de los productos software; en términos generales estos elementos serían aquellos relacionados con el proceso de desarrollo en sí (mayormente lenguaje de programación y posibles complementos o *frameworks* del mismo).

Si queremos aplicar estos conceptos a nuestro proyecto más concretamente, los aspectos que tendremos que concretar al comienzo del proyecto no solo serán los que sean relevantes al acto de creación de los juegos puramente, en nuestro caso se incluiría además la inserción de los juegos en una página web y además la integración de un aprendizaje en estos.

La creación de una página web sería una cuestión parecida a la de la elección de un lenguaje de desarrollo, pues consistirá en elegir un lenguaje de marcado con el que incluir los juegos, así como una serie de modificaciones en distintas páginas.

Por otra parte, la inclusión del aprendizaje en estos juegos requiere decidir de qué forma se va a realizar, lo cual dictará como debemos integrar sus modelos de entrenamiento: más concretamente como funcionarán y como se comunicarán e interactuarán con los juegos.

Así, si tenemos los distintos aspectos del proyecto en cuenta, tendremos que especificar los siguientes elementos:

- **Lenguaje de desarrollo de los juegos:** En este caso, al lidiar con unos programas que serán integrados dentro de una página web, elegiremos JavaScript, uno de los lenguajes de programación más populares para desarrollo web actuales.
 - **Framework:** entre otros beneficios con los que cuenta, JavaScript cuenta con una gran cantidad de bibliotecas y frameworks a su disposición; en este caso hemos elegido el framework Phaser3, orientado a la creación de juegos web, ideal para nuestro propósito.
- **Lenguaje de marcado:** este será el lenguaje que estructurará el contenido de la página web; usaremos el extendido HTML.
- **Método del aprendizaje:** en el mundo del aprendizaje automático existen diversas ramas o metodologías que siguen distintos métodos con el objetivo de que una máquina aprenda. En nuestro caso hemos elegido el aprendizaje por refuerzo. Más concretamente, utilizaremos un tipo de algoritmo llamado SARSA que veremos más adelante.
- **Otras herramientas:** todos los elementos usados que no están relacionados directamente con los anteriores. Hemos usado la herramienta **Microsoft Teams**, para la comunicación entre profesor y estudiante; **Git y Github**, para el control de versiones del código, **Microsoft Word** se ha usado para la redacción de la memoria del trabajo y como editor de código fuente el estudiante ha usado **Visual Studio Code**.

En la primera parte del capítulo empezaremos describiendo los conceptos previos sobre aprendizaje por refuerzo que son necesarios para entender el desarrollo del proyecto y en la segunda analizaremos las tecnologías utilizadas.

2.1 Antecedentes

Si queremos estudiar los antecedentes de este proyecto, tendríamos que observar el recorrido del aprendizaje automático [18] y, además, el de la Inteligencia Artificial (IA) [2] a lo largo de la historia y destacar los sucesos más importantes y los que estén relacionados con el trabajo.

De esta forma, podemos establecer distintos antecedentes para la IA a lo largo de la historia, pero el nacimiento oficial del campo de la Inteligencia Artificial se produce en 1956 durante la conferencia de Dartmouth [2], un evento propuesto por el profesor John McCarthy para hablar sobre lo que era entonces el campo de las máquinas pensantes.

De esta conferencia surgió así el nombre Inteligencia Artificial y durante la misma los asistentes se dedicaron a la discusión de distintos temas sobre la misma a modo de lluvia de ideas con los distintos asistentes presentando y cuestionando distintos temas e ideas.

Por supuesto, junto a distintos antecedentes previos que fueron de gran importancia, fue de gran importancia las contribuciones de Alan Turing a lo largo

de su vida, por supuesto con sus aportes al desarrollo de la computación con la creación de la máquina de Turing.

Además, también es importante en nuestro contexto la concepción del Test de Turing en 1950 [2] que busca comprobar la capacidad de una máquina de emular la inteligencia de un humano. Esto claramente, refleja la examinación y comprobación de una Inteligencia Artificial, años antes de que el término en sí surgiera.

Tras la conferencia de Dartmouth se generó un gran entusiasmo dentro del campo y se crearon distintos programas con diferentes objetivos: existían aquellos que querían solucionar problemas matemáticos como el *General Problem Solver* [8] en 1957.

Otros tenían la meta de llegar a comprender idiomas como el inglés un ejemplo de estos es el programa ELIZA [1] que podía simular una conversación con un humano y por supuesto también existía la aplicación de la IA a robots físicos que pudieran realizar distintas tareas.

A lo largo de los años así, surgen períodos de auge como este último y períodos de menos entusiasmo en el campo de la IA y con una gran cantidad de sucesos importantes para la misma en todo este tiempo: desde los comienzos y desarrollos posteriores de las redes neuronales.

También, ocurrieron sucesos famosos en todo el mundo como la victoria del ordenador Deep Blue ante el campeón de ajedrez Garry Kasparov y en tiempos más recientes las victorias del programa AlphaGo en el juego Go [9].

Así, nos encontramos en una época en la que la IA es un campo de gran importancia y es un elemento integrado en una gran cantidad de tecnologías usadas actualmente.

En nuestro proyecto, lidiamos con una parte del campo de la inteligencia artificial: el aprendizaje automático, específicamente, el aprendizaje por refuerzo; siendo estos métodos ampliamente usados.

Por ejemplo, anteriormente hemos hecho referencia a usos de IA en juegos, como Deep Blue o AlphaGo, y es que el aprendizaje por refuerzo se puede usar (y se usa) en estos ámbitos, incluso en el de videojuegos electrónicos más comerciales: AlphaStar lo utilizó junto a otras técnicas para conseguir vencer a jugadores profesionales del videojuego StarCraft II [10] y OpenAI Five similarmente la usa en combinación a otras técnicas para jugar a Dota 2 [11].

En definitiva, podemos observar que existen aplicaciones de este tipo de aprendizaje en el mundo de los juegos, tanto los más tradicionales, como los más novedosos

2.2 Aprendizaje por refuerzo

Habiendo establecido algunos antecedentes a nuestro proyecto es importante explicar más detenidamente estos tipos de aprendizajes y sus funcionamientos.

Más concretamente, nos referimos al aprendizaje automático; el cual es en sí una rama de la Inteligencia Artificial. Así, en este se intentan crear métodos y técnicas para hacer que un ordenador pueda “aprender”.

En el aprendizaje automático, una máquina contiene una serie de datos a partir de los cuáles genera un modelo de actuación y posteriormente toma decisiones en su entorno basándose en este, pudiendo decir que gracias a los datos iniciales se ha aprendido a actuar de una forma concreta.

Dentro de esta rama, existen distintas formas de conseguir el objetivo deseado, distinguiendo generalmente tres variaciones:

- **Aprendizaje supervisado:** al algoritmo se le otorgan unos datos y las respuestas esperadas para cada uno de ellos; en otras palabras, una entrada y su respectiva salida.
- **Aprendizaje no supervisado:** a diferencia del método previo, en este caso solo se le otorga al algoritmo unos datos de entrada y este saca sus propias decisiones.
- **Aprendizaje por refuerzo:** en este caso, el algoritmo aprende a base de ensayo-error, tomando acciones en su entorno, observando los resultados y teniéndolos en cuenta para el futuro.

Como hemos mencionado anteriormente, en el caso de este trabajo fin de grado nos decantamos por usar el aprendizaje por refuerzo. Aquí es de gran importancia el entorno, pues a partir de este el algoritmo obtiene la información que necesita para conseguir un modelo de actuación.

Por eso, en estos tipos de aprendizajes el entorno se expresa mediante un proceso de decisión de Markov (MPD).

En primera instancia, esto quiere decir, que en el entorno se cumple la propiedad de Markov, en la que se establece que si nos encontramos en un estado presente, el estado futuro solo dependerá de este último y la acción tomada en el mismo y no dependerá de ningún estado pasado.

Más concretamente, si nos encontramos en el estado $\mathbf{S}_t$ es únicamente porque en el estado anterior $\mathbf{S}_{t-1}$ hemos tomado una acción $\mathbf{a}_{t-1}$. Asimismo, el nuevo futuro estado $\mathbf{S}_{t+1}$ solo dependerá del estado actual $\mathbf{S}_t$ y la acción $\mathbf{a}_t$ que tomemos en el mismo y no de ningún estado anterior al presente; como. $\mathbf{S}_{t-1}$ en nuestro caso.

En la Figura 2.1 podemos observar una formulación de la propiedad de estos estados:

$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \dots, S_t]$$

Figura 2.1: Formulación de la propiedad de Markov

Una vez tenemos estados que cumplan la propiedad de Markov, debemos establecer los procesos de Markov. Estos son así fenómenos en los que contamos con una serie de estados que cumplen la propiedad de Markov, pero con el añadido de que existen probabilidades para las transiciones entre estados.

Así, nos encontramos que desde un estado inicial tendremos distintas posibilidades de saltar a un estado u otro, por ejemplo, pudiendo haber un 80% de posibilidades de pasar a un estado concreto, y un 20% de posibilidades de transitar a otro distinto. Cabe señalar que esto se aplica a que el salto sea al mismo estado inicial.

Estas probabilidades de transición entre cada estado en las cadenas de Markov se ven reflejadas en la matriz de transición de estados la cual es una función de transición que nos muestra estas posibilidades.

De esta forma, el fenómeno de las cadenas de Markov puede contar con más elementos que intervengan en las transiciones entre estados además de las probabilidades, lo cual lo convierte en otros tipos de fenómenos. En específico, si a una cadena de Markov le añadimos recompensas y acciones concretas desde los estados; contamos con un proceso de decisión de Markov.

Así, en un proceso de decisión de Markov nos encontraremos en un estado desde **S** el que podemos tomar una serie de acciones: **A**. El elegir una de estas, determinará las probabilidades de acabar en distintos estados nuevos (**S'**), y además también determinará la recompensa esperada (**R**).

En definitiva, como hemos establecido, el aprendizaje por refuerzo tendrá que ocurrir en un entorno que siga el proceso de decisión de Markov, por lo que en este un algoritmo tomará decisiones sobre sus acciones en distintos estados y observando las recompensas que obtenga de su experiencia, aprenderá a intentar maximizarlas en un proceso de ensayo-error.

En este proceso de aprendizaje, es necesario que el algoritmo logre un equilibrio entre dos formas distintas de explorar las posibilidades de su entorno:

- **Exploración:** con este método, el algoritmo explora su entorno y realiza acciones de forma más aleatoria, ampliando su conocimiento sobre las distintas acciones que puede realizar. Esto está orientado a un beneficio a largo plazo.
- **Explotación:** en este caso, el algoritmo elige explotar el conocimiento que ya tiene sobre su entorno, eligiendo las acciones que de momento sabe que le van a dar la mayor recompensa posible, sin explorar otras opciones.

Ambas opciones tienen sus lados positivos y sus lados negativos, pero cuando se combinan de forma efectiva podemos beneficiarnos de un algoritmo que explora lo desconocido para él y al mismo tiempo se aprovecha de lo que ha experimentado hasta el momento para obtener la máxima recompensa posible.

Así, necesitamos una forma de poder equilibrar de forma eficiente ambos métodos, y para ello usaremos la **estrategia epsilon-greedy**, en la que se siguen los siguientes pasos:

- Establecemos un valor decimal entre 0 y 1, llamado **epsilon**, representado por el símbolo ϵ .
- Cuando el algoritmo quiera tomar una acción se genera un número aleatorio n entre 0 y 1.
- Dependiendo del valor de n , ocurrirá una de dos posibilidades:
 - Si $n < \epsilon$, se realizará una acción aleatoria.
 - Si $n > \epsilon$, se realizará la acción que otorgue mayor recompensa. Esta es llamada la acción *greedy*.
- Una vez hemos realizado la acción y nos encontremos en el nuevo estado, generamos un nuevo valor para n y realizamos una nueva comprobación.

En la Figura 2.2 podemos observar un esquema que representa esta política.

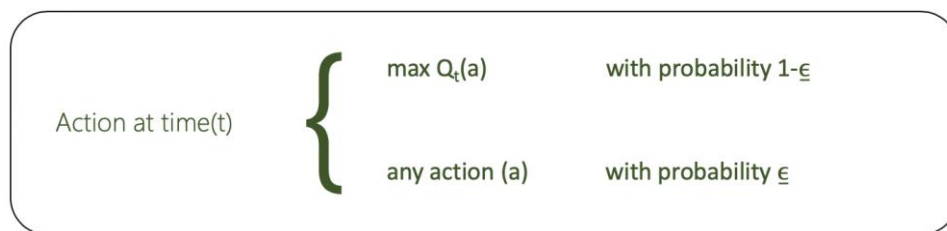


Figura 2.2: Política *epsilon-greedy*. Imagen extraída de [19]

En resumen, con todas las características que hemos establecido hasta ahora, tendríamos un algoritmo de aprendizaje por refuerzo, en el que se actúa en un entorno que sigue un proceso de decisión de Markov y en el que se seguirán acciones por exploración y explotación, balanceando el uso de ambos métodos mediante una **estrategia epsilon-greedy**.

2.2.1 Elementos del aprendizaje por refuerzo

Una vez que establecemos las características generales del aprendizaje por refuerzo, es importante exponer los diferentes elementos específicos que caracterizan estos algoritmos:

- **Agente:** es el elemento que aprende. El agente observa el estado en el que se encuentra, realiza las acciones y recibe las recompensas correspondientes.
- **Entorno:** el ambiente sobre el que actúa el agente y el que le comunica a este su estado actual y la recompensa que resulta de sus acciones.
- **Acción:** los actos que toma el agente en su entorno, y provocan cambios en el estado del entorno.
- **Recompensa:** la representación de lo positiva o negativa que ha sido una acción para el agente.
- **Política:** este elemento establece una relación entre estados y acciones, es decir, asocia el estado del agente a una o distintas acciones a tomar. Define así, el comportamiento del agente según su estado.

- **Función de refuerzo (o recompensa):** función que asocia la acción tomada por el agente en un estado concreto, a una recompensa o castigo.
- **Función de evaluación (o valor):** la función de evaluación establece la máxima recompensa esperada desde un estado siguiendo una política concreta.
- **Modelo del entorno:** un modelo que cuyo objetivo es intentar predecir la evolución del entorno y las transiciones entre estados según las acciones del agente.

Así, con todos estos elementos existen distintos tipos de algoritmos por refuerzo que pueden ser utilizados, en concreto hay dos que suelen ser utilizados en gran medida: **Q-Learning** y **SARSA**.

Al hablar de ambos algoritmos, es necesario denotar la presencia de otro elemento de importancia en ambos: la **Q-Table**. Es una tabla en la que las filas representan todos los estados del entorno y las columnas las acciones disponibles del agente (aunque también es habitual el uso de su transpuesta), y cada casilla representa la función de valor para el agente en el estado y acción que le correspondan.

Por ejemplo, en una Q-Table con tres filas y dos columnas, tendríamos un entorno en el que hay tres estados y dos acciones disponibles para el agente. Así, en la primera casilla de todas, tendríamos la función de valor de realizar la acción 0 en el estado 0, y así con todas las casillas.

De esta forma, en ambos algoritmos, cuando el agente explora los distintos estados tomando distintas acciones, modifica los valores de la Q-Table en base a las recompensas que va recibiendo, es decir, en base a su experiencia. La principal diferencia entre ambos será el modo en el que se modifiquen estos valores, esto lo veremos a continuación.

2.2.2 Algoritmo Q-Learning

El algoritmo Q-Learning [12] es un algoritmo de aprendizaje por refuerzo donde se intenta que un agente en un entorno elija las acciones más eficientes para su estado en todo momento. Este algoritmo dado un tiempo de exploración infinito y una política en parte aleatoria, seríamos capaces de encontrar una elección de acciones óptima para cada estado eventualmente.

Uno de los aspectos importantes de Q-Learning, es que es un algoritmo *off-policy*. Esto quiere decir que a la hora de actualizar los valores de su Q-Table tras actuar, en la fórmula no utiliza el valor de la nueva acción que vaya a realizar posteriormente sino el de la mejor acción que le sea posible, utilizando un punto de vista *greedy*. El funcionamiento general del algoritmo está representado en la Figura 2.3:

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  $Q(\text{terminal}, \cdot) = 0$ 
Loop for each episode:
  Initialize  $S$ 
  Loop for each step of episode:
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal

```

Figura 2.3: Funcionamiento del algoritmo Q-learning. Imagen extraída de [20]

Si observamos la tercera línea desde el final, encontramos la fórmula de actualización de los valores de la Q-Table (**$Q(S, A)$**) tras realizar la acción **A** en el estado **S** y llegar a **S'** . En este proceso se usa el máximo valor de la Q-Table en el nuevo estado **S'** por lo que toma una acción ***greedy***, lo cual no ocurre en otros tipos de algoritmo (esto lo veremos más adelante). Asimismo, podemos ver en la fórmula dos parámetros concretos representados por dos símbolos:

- **Tasa de aprendizaje:** representado por el símbolo alfa (α). Este es un número decimal entre 0 y 1, que representa la importancia que le da el agente a la nueva información que obtiene con respecto a la antigua. Con una tasa de 0, el agente no aprende, pues no usa el nuevo conocimiento, y con una tasa de 1 el agente solo usará los nuevos conocimientos que vaya obteniendo y sustituyendo a los antiguos.
- **Factor de descuento:** representado por el símbolo gamma (γ). Este número decimal entre 0 y 1 determina la importancia de recompensas futuras (a largo plazo). Un factor de 0 hace que el agente se centre en las recompensas más inmediatas y un factor de 1 hace que busque altas recompensas lejanas.

2.2.3 Algoritmo SARSA

El algoritmo SARSA [13] es también un algoritmo de aprendizaje por refuerzo al igual que Q-Learning. De hecho, son muy parecidos en la mayoría de sus aspectos, algo que se puede denotar si observamos su proceso de funcionamiento en la Figura 2.4:

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

```

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
  Repeat (for each step of episode):
    Take action  $A$ , observe  $R, S'$ 
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'; A \leftarrow A'$ 
  until  $S$  is terminal

```

Figura 2.4: Funcionamiento del algoritmo SARSA. Imagen extraída de [21]

Como podemos observar el mayor cambio surge en la fórmula de actualización de la Q-Table junto al factor de descuento. En este caso, SARSA utiliza el valor Q de realizar una nueva acción (**A'**) en el nuevo estado (**S'**), y no la acción más óptima de este nuevo estado.

Por lo que mantiene la misma política para elegir nuevas acciones y para actualizar su tabla, por lo que nos referimos a SARSA como un algoritmo *on-policy*.

2.2.4 Cuantificación

La cuantificación es una técnica utilizada en casos de aprendizaje en los que haya una cantidad de estados demasiado grande como para trabajar con ellos de forma eficiente con el aprendizaje por refuerzo.

Por ello, usamos la cuantificación, que descarta una gran cantidad de los estados para poder manejar el entorno más fácilmente.

Por ejemplo, si lidiamos con estados en los que se incluya una distancia entre objetos, el incluirla de forma específica (en metros, por ejemplo), aumentaría de forma exponencial el número de estados.

Con la cuantificación, podemos cambiar esta medida exacta a una aproximación que nos otorgue la información necesaria, manteniendo un número de estados manejable para el algoritmo.

2.3 Tecnologías usadas

Más allá de los aspectos relacionados con la teoría del aprendizaje automático que queremos aplicar a nuestro proyecto, es necesario la utilización de distintas tecnologías para la creación de nuestros juegos web, así como la posterior integración de dicha teoría de aprendizaje en los mismos.

Para ello, necesitamos tecnologías de desarrollo de software y un lenguaje de marcado para presentarlo en una página web de forma correcta. Para lo primero hemos elegido JavaScript un lenguaje de los más extendidos en el desarrollo web y con gran cantidad de soporte y herramientas.

Una de estas herramientas es el *framework* Phaser3 un motor con el que se pueden crear juegos en HTML5; el contar con este motor facilita la creación de los juegos y con una estructura adecuada la posterior implementación del aprendizaje en los mismos.

Para la presentación y organización de los juegos en forma de web, utilizaremos el lenguaje de marcado HTML, el más extendido y usado ampliamente en la actualidad.

Cabe destacar que este no es un lenguaje de programación, sus funciones son distintas: como hemos dicho en HTML se controla la presentación de distintos elementos, mientras que con lenguajes de programación se produce una salida a partir de una serie de instrucciones.

2.3.1 JavaScript

Así, como hemos mencionado anteriormente, el lenguaje de desarrollo escogido para la creación de los juegos es JavaScript [14]. En la actualidad, es de los lenguajes de desarrollo más extendidos y populares en el desarrollo de aplicaciones web y está además orientado a este tipo de producto.

Contrario a lo que pueda parecer, no tiene relación directa a Java (aunque parte de su sintaxis pueda parecerse) y JavaScript no era su nombre original, pero si fue el nombre con el que fue lanzado de forma oficial en diciembre de 1995 por Netscape para su navegador Netscape Navigator.

Originalmente, JavaScript fue creado con la intención de otorgar dinamismo a las páginas web. En los tiempos de los primeros navegadores web con interfaz de usuario estas páginas eran estáticas lo cual no permitían cambios en las mismas una vez eran cargadas por el navegador; si se querían mostrar cambios, era necesario volver a cargarla.

Así, el querer otorgar dinamismo a una página web una vez cargada, JavaScript es creado como un lenguaje que realiza todas sus operaciones del lado del cliente. Esto permite ejecutar instrucciones en la máquina local en el navegador sin tener que consultar al servidor, lo cual haría que se recargara la página en la que se encontrase.

Con el paso del tiempo surgieron distintas versiones de JavaScript (como JScript de Microsoft), lo cual a veces complicaba la compatibilidad de las páginas con distintos navegadores dependiendo de que tipo de lenguaje usara.

Finalmente, tras un periodo de intentos de estandarización de este lenguaje en 2009 se acordó entre las compañías relevantes trabajar entre todas para impulsar el lenguaje; lo cual llevó a la creación del estándar ECMAScript 5. Este estándar sigue actualizándose con nuevas versiones en la actualidad.

Así, desde la creación de este estándar JavaScript ha crecido de forma considerable, siendo uno de los lenguajes de desarrollo líder en el campo de las páginas web y recibiendo una gran atención. Incluso se han producido implementaciones de entornos de servidor basados en JavaScript algo que se había intentado en la infancia del lenguaje, pero no había tenido frutos. Un ejemplo importante de la utilización de JavaScript en un servidor sería Node.js.

Algunas de las características más importantes de JavaScript son:

- Es un lenguaje ligero entre otras cosas porque usa un intérprete en vez de un compilador. Además, usa una política de *just-in-time*, lo cual significa que es procesado justo en el momento de ejecución.
- Usa tipado dinámico, es decir, no es necesario declarar el tipo de cada variable y esta última puede cambiar su tipo cuando se requiera.
- Tanto las funciones como todos los datos que no sean de tipo primitivo son considerados objetos, es decir, heredan todas las características del prototipo Objeto de JavaScript, lo cual hace por ejemplo que una función pueda llamar a cualquiera de los métodos del prototipo.

Mayormente, como hemos mencionado con anterioridad, JavaScript es un lenguaje orientado al desarrollo web y con compatibilidad con una gran cantidad de navegadores en la actualidad.

Esto junto al gran soporte que se le da en la actualidad, así como las diferentes librerías con las que cuenta, nos lleva a elegirlo como el lenguaje de desarrollo para la creación de los juegos en la página web.

Habiendo elegido JavaScript, el siguiente paso sería la elección de alguna librería o framework que nos facilite el desarrollo de videojuegos en este lenguaje. Por suerte, existe un framework que cumple nuestros requisitos: Phaser3.

2.3.2 Phaser3

Phaser3 [15] es un framework gratuito de JavaScript que permite la creación de juegos en HTML5 para ordenadores y móviles desarrollado por la empresa Photon Storm. En principio está orientado mayormente a la creación de juegos 2d pero es posible la introducción de elementos 3d con la utilización de distintas técnicas y herramientas adicionales del framework.

Originalmente creado en 2013; Phaser3 es la tercera revisión general del motor, existiendo todavía una versión de Phaser2 que se sigue usando hoy en día, aunque su última versión oficial fue la 2.6.2 en agosto de 2016, siendo las más recientes versiones soportadas por una comunidad de usuarios.

Por otra parte; también está en desarrollo una versión Phaser4 que intenta trasladar los módulos que existen en Phaser3 al lenguaje TypeScript.

Para la creación de los juegos es necesario crear la lógica interna de los mismos, pero al mismo tiempo también es necesario poder mostrar los distintos elementos, formas y figuras que intervienen en el mismo y cómo cambian a lo largo del tiempo.

Para ello necesitamos el uso de un renderizador de gráficos que pueda mostrar todo esto en la página web. Phaser3 es compatible con dos renderizadores: WebGL una API de JavaScript y el elemento canvas de HTML5, pudiendo usar cualquiera de los dos.

Cabe destacar, que la creación de juegos directamente en el canvas de HTML5 es posible, sin embargo, con Phaser se simplifica el proceso de forma considerable al introducir métodos que automatizan acciones que serían más complejas al trabajar directamente en el canvas.

Así, una vez que ya podemos mostrar las distintas partes del juego en la página, el uso de Phaser3 en nuestro proyecto permite utilizar las características de JavaScript en un entorno para crear videojuegos.

De esta forma podemos crear una página en la que existan los distintos elementos del juego, y que cada uno de estos sea un objeto dentro del programa

con sus propios métodos y propiedades a los que podemos llamar y modificar y a través del renderizador que esté usando, mostramos la situación concreta del juego en cada momento.

En definitiva, Phaser3 es un framework de creación de juegos en la web en HTML5, gratis, de código abierto, sencillo de usar, con gran soporte, complementos y plugins y que utiliza JavaScript, por lo que es un framework ideal para la creación de nuestros juegos.

2.3.3 HTML

Por último, para la creación y correcta presentación de información en una página web es necesario el uso de un lenguaje de marcado; en nuestro caso utilizaremos el ampliamente usado en todo el mundo HTML [16].

Cuando un usuario está explorando una página web, el proceso que va ocurriendo es que el navegador recibe documentos HTML de un servidor y de acuerdo a este archivo presenta la información que contiene de una manera u otra.

Así, con HTML utilizamos una tecnología que nos permite mediante etiquetas presentar los juegos de forma correcta y además poder en caso de necesitarlo, insertar cualquier funcionalidad extra que queramos para poder obtener información y/o modificar en cierta manera el juego (como podría ser por ejemplo acelerarlo para que se ejecute más rápidamente).

3

Análisis y Modelado

Una vez tenemos un entendimiento de las técnicas y tecnologías usadas para la creación del proyecto, debemos intentar tener una visión general de las distintas partes del mismo antes de empezar a implementarlas. De lo contrario podríamos cometer errores iniciales que más adelante sería complicado corregir.

Así, para asegurar que la realización del proyecto es efectiva y eficiente, es necesario realizar un análisis de las distintas partes del mismo, observando como poder estructurarlas y como hacer que interactúen entre ellas.

En nuestro caso, tendremos que observar los distintos juegos y sus partes, viendo como estas tienen que comportarse para seguir las reglas del mismo y que este funcione correctamente.

Asimismo, debemos tener en cuenta que en los juegos tiene que haber un aprendizaje, por lo que debemos combinar el funcionamiento normal del juego, con algoritmos de aprendizaje que manejen la información de la partida y puedan responder correctamente a sus distintos estados.

Esto último quiere decir que los elementos de aprendizaje del proyecto y los miembros de cada uno de los juegos deben poder comunicarse entre sí correctamente para conseguir el objetivo del trabajo.

Una vez que hemos realizado un análisis del proyecto, podremos trabajar la creación e implementación de cada una de sus partes, siguiendo la estructura denotada por el análisis asegurándonos de maximizar el rendimiento de todas sus partes.

3.1 Análisis

Si queremos hacer un análisis del proyecto en general, tendremos que observar las distintas partes en las que puede ser dividido, y denotar las necesidades que tiene cada una de ellas para poder conseguir los objetivos que tenemos planteados.

Así, en nuestro trabajo contamos con:

- **Aprendizaje:** tenemos que crear los elementos necesarios para gestionar un aprendizaje por refuerzo.
- **Juegos:** crearemos la lógica y los miembros de los juegos de modo que funcionen correctamente.

Ambos elementos además, son desarrollados para ser reproducidos en un navegador web, y tendrán que comunicarse entre sí y responder efectivamente a lo que ocurra en el otro: si en el juego se da un estado negativo, el algoritmo de aprendizaje tendrá que informarle de su castigo, lo cual implica traslado de información por parte de los dos.

Así, tendremos que establecer la estructura de las dos partes y de ambos juegos posteriormente.

3.1.1 Entorno para algoritmos de aprendizaje por refuerzo en el navegador web

En primer lugar, un análisis de nuestro proyecto tiene que partir desde una perspectiva más general, por lo que tenemos que observar los distintos aspectos de desarrollar una aplicación de juegos con algoritmos de aprendizaje por refuerzo en un navegador, denotando sus partes más importantes, y posteriormente analizar los casos concretos.

Así, la primera de las partes que querríamos estructurar sería el **algoritmo**. Obviamente será de gran importancia, pues es el principal elemento del aprendizaje y el que se comunica con las demás partes para asegurar que se siguen las pautas de los algoritmos que pretendemos usar.

Entre otras cosas, algunas funciones importantes del algoritmo serían:

- Gestionar el uso de la política *epsilon-greedy*, determinando que tipo de acción hacer en cada momento.
- Administrar las recompensas y castigos, identificando las situaciones y estados del juego que llevan a unas u otras.
- Gestionar la Q-Table, y aplicar correctamente las actualizaciones sobre la misma.

Así, el algoritmo en sí en definitiva se encarga de aplicar correctamente las características expuestas anteriormente para asegurar el aprendizaje.

Por otra parte, en nuestro proyecto, el control de los juegos no sería exclusivo de un algoritmo, por lo que necesitamos un elemento **controlador**, que le otorgue el manejo de una de las entidades del juego a una persona, el jugador.

Posteriormente, tenemos los **agentes** los cuales llevan a cabo las acciones y sobre ellos sobre produce el aprendizaje.

En todo momento del juego, el agente se encontrará en un estado concreto y tendrá un número de acciones a su disposición. En nuestro caso, las acciones que realizará el agente serán decididas por el algoritmo o por un jugador y dichas acciones podrán cambiar el estado en el que se encuentre y tener consecuencias positivas o negativas.

Para poder interactuar con el algoritmo, un agente debe poder seguir las normas de la política epsilon-greedy que usamos en el proyecto. Por ello el agente tendrá que tener métodos que permitan elegir la acción más óptima que se conozca (greedy), o una aleatoria de las disponibles.

Por último tendríamos el entorno. El entorno es el espacio en el que se encuentran todos los elementos y en el que se toman todas las acciones y sobre el que se basan todos los estados.

En el trabajo, el entorno sería el juego en sí. Este está caracterizado por una serie de objetos, algunos de los cuales serán agentes y otros no. Las condiciones de cada uno de estos objetos, determinarán los estados del entorno, por lo que es necesario mantener actualizada la situación de cada uno.

En definitiva, si creamos una aplicación con aprendizaje de refuerzo en un juego de navegador tendremos un algoritmo y/o un controlador humano que indiquen a unos agentes qué acciones tomar en un entorno en el que existirán distintos objetos, que pueden ser a su vez agentes también.

Una vez establecidos estos elementos podemos observar los casos concretos que realizaremos.

3.1.2 Juego del Pong

El juego del Pong se desarrolla en un cuadrado o rectángulo en el que dos plataformas (o palas), golpean una pelota para enviarla al lado contrario en el que se encuentra otra plataforma e intentar sobrepasar a este contrario. Un ejemplo de este juego lo encontramos en la Figura 3.1.

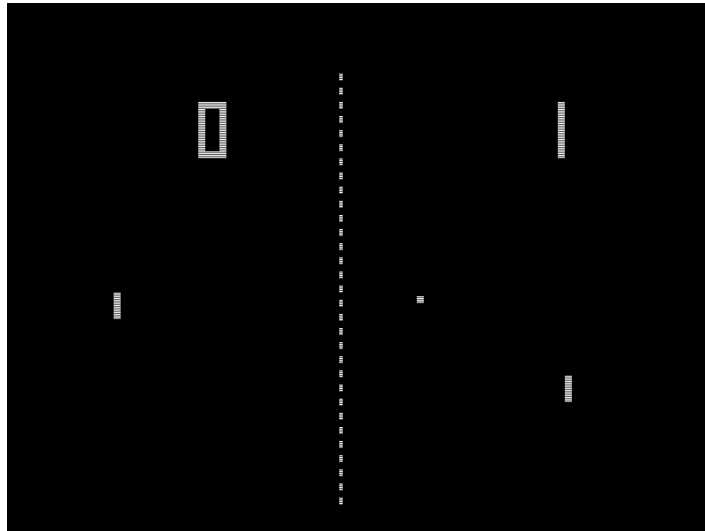


Figura 3.1: Ejemplo de juego Pong. Imagen extraída de [22]

Así en este caso contamos con dos tipos de elementos dentro del juego: las palas y la pelota. Las palas (o plataformas) serán agentes, por lo que podrán estar guiadas por un algoritmo y aprender de sus acciones o ser controladas por un jugador. Estos elementos dentro del proyecto recibirán el nombre de **Platform**.

Estos objetos tendrán tres acciones disponibles: moverse a la izquierda, derecha o permanecer quietos. Además los estados que tendrá en cuenta incluirán su propia posición, la del contrincante y la de la pelota, así como la velocidad y dirección de esta.

Por otra parte, la pelota es un elemento esencial para el funcionamiento del juego, pero que no es controlada por otro miembro, se mueve por un movimiento inicial y posteriormente por sus colisiones con las palas o los límites del juego. Podríamos referirnos a esta dentro del proyecto como **Ball**.

Adicionalmente, necesitaremos crear el entorno específico del Pong, en el que incluyamos a estos elementos y además cuidemos que se mantenga la lógica del mismo, estableciendo las acciones disponibles de las plataformas (moverse a un lado, al otro o quedarse quieto) y controlando los estados en los que se acaba una partida e informar a cada agente del resultado, entre otras cosas.

El entorno del juego será referido en este caso, y en sus variaciones, con el nombre de **Environment**.

Sin embargo, el número de estados a considerar si usamos los píxeles como coordenadas para las palas y la pelota es demasiado alto. Para reducir la complejidad del espacio de búsqueda, aplicamos un proceso de cuantificación, como el explicado en el capítulo 2.2.4. De esta forma, el modelo dividirá el campo de juego en cuadrículas de varios píxeles de alto y ancho, y visualizará las posiciones de los elementos a través de ellas. Así, las palas y la pelota se moverán por píxeles, pero los estados del aprendizaje serán determinados por la cuadrícula en la que este cada uno.

Por otra parte, tendremos que asegurarnos que la lógica del juego se adapte correctamente a tamaños distintos del campo de la partida, para poder ver como este cambio afecta al aprendizaje de los agentes.

3.1.3 Tres en raya

El juego del tres en raya es uno más simple al Pong. En este contamos con una tabla de 3x3, en la que cada casilla se puede rellenar con un símbolo, una **X** y **O** en nuestro caso. Cada símbolo representa a un jugador, y el primero que encadene sus tres símbolos de forma vertical, horizontal o diagonal, gana. Tenemos un ejemplo en la Figura 3.2:

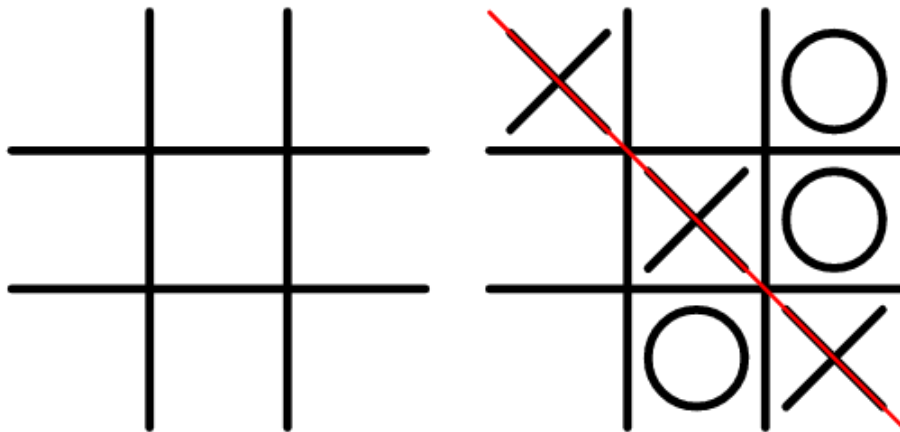


Figura 3.2: Ejemplo de juego de tres en raya, a la izquierda en su estado inicial, a la derecha un estado final de victoria para X. Imagen extraída de [23]

Así, en este caso, dentro del juego contamos solo con dos jugadores siendo ambos los agentes. Estos son más abstractos que en el Pong, pues solo actúan rellenando una de las casillas disponibles y no son un objeto permanente en el campo de juego.

Así, las acciones que puede realizar cada agente son distintas a las del Pong: en este, los agentes podían realizar cualquiera de las tres acciones en cualquier momento. Sin embargo, aquí cuando una casilla esté ocupada, no podrá tenerla ningún otro, lo cual limitará el número de acciones disponibles.

Por ejemplo, en un estado inicial, el agente que juegue tendrá nueve movimientos disponibles. En la jugada siguiente, habrá ocho movimientos disponibles y así sucesivamente.

Además, las acciones de ambos son alternas. Es decir, cuando un agente actúa el otro no lo puede hacer y viceversa.

Por otra parte, seguiremos contando con un entorno que deberá mantener la lógica del juego correctamente. En el tres en raya debemos asegurar que los jugadores rellenan sólo casillas válidas y detectar cuando se produce una victoria, derrota o empate, e informar debidamente a los agentes o a los algoritmos.

Cabe destacar que este **Environment** tendrá que ser distinto al de Pong. En este juego, existe un límite del número de movimientos que se pueden realizar en cada partida (9 entre los dos) y además puede acabar en empate, por lo que tendrá que detectar de forma distinta los estados finales del juego e informar a los agentes de dicho resultado.

3.2 Modelado

Siguiendo la estructura que hemos establecido con el análisis creamos un diagrama de clases para representar las distintas partes del proyecto. En la Figura 3.3 se muestra el diagrama de clases para el juego del Pong. Para otros juegos, la mayor parte es igual cambiando solo la parte inferior (clases heredadas de **Element** y sus relacionadas).

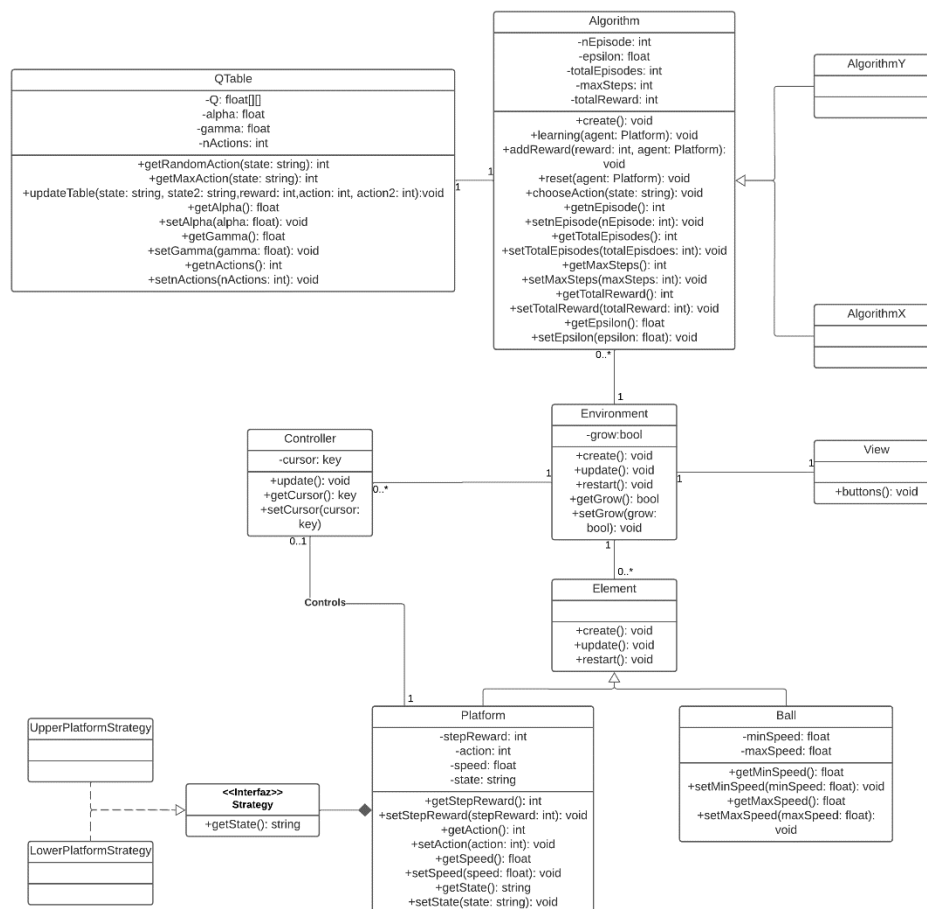


Figura 3.3: Diagrama de clases del proyecto

El primer elemento a destacar sería **Environment** que representa el entorno del juego (podríamos decir la instancia del juego en sí). Dentro del entorno destacamos el atributo de **grow**. Este es un atributo que determina si el juego actual es una versión en la que se aumentará el tamaño si es posible o no. Por otra parte contamos con una serie de funciones básicas para el funcionamiento de la partida:

- **create():** en este método se realizan las acciones necesarias para la generación del proyecto (creación de parámetros y distintos elementos que vayan a intervenir).
- **update():** método que se ejecuta en cada frame del juego para actualizar la situación de cada elemento del mismo (acción que esté haciendo, posición...)
- **restart():** método ejecutado entre episodios para restablecer cada elemento.
- **Getters y Setters():** estos métodos se realizarían para poder obtener y cambiar correspondientemente los valores de los atributos de esta clase. Estos métodos aparecen en muchos objetos y siguen la misma lógica en todos.

Una vez tenemos el entorno que tendrá la lógica de juego necesitamos desarrollar el resto de participantes del programa.

Primero, crearemos una vista para el juego, en la que se representará correctamente en todo momento el estado actual del juego, estando así relacionados. Los métodos con los que contará serán las funcionalidades que usaremos para facilitar nuestras comprobaciones (acelerar el juego, descargar información).

Después, tenemos los objetos que estarán dentro del juego y cuyas acciones o situaciones determinan los resultados y estados de la partida. Estos los llamamos **Element** (elementos). Todos los elementos tendrán los mismos métodos **create()**, **update()** y **restart()** que hemos establecido antes; y estos funcionan de la misma manera. Una vez establecemos estos, si nos fijamos en el ejemplo principal del Pong tendremos dos tipos de elementos (que heredan de esta clase):

- **Platform:** la plataforma, los cuales serán también los agentes del juego. Contará con los siguientes atributos:
 - **stepReward:** este número representa la recompensa (positiva o negativa) que se le debe al agente por la última acción hecha.
 - **action:** acción que está haciendo actualmente.
 - **speed:** velocidad con la que se moverá la pala en ambos sentidos.
 - **state:** estado en el que se encuentra el agente.
 - El resto de métodos de la plataforma serán Getters y Setters; sin embargo existe una ocasión en la que uno de ellos no será un getter normal, esto se explicará después.
- **Ball:** el representante de la pelota en el juego del Pong. Contará con los atributos **minSpeed** y **maxSpeed** que delimitarán las velocidades con las que se podrá mover en el juego. Los métodos son todos de nuevo Getters y Setters.

Dependiendo del juego Pong con el que estemos lidiando, es posible que las plataformas tengan que usar distintas formas de transmitir su estado. Para ello creamos una interfaz **Strategy** que pertenecerá cada plataforma cuando sea necesario.

Así, en la clase **Strategy** se define el método **getState()** sin especificar y las dos clases que implementan la interfaz, la definirán de forma distinta. Cuando instanciamos la plataforma, si es necesario, le diremos cuál de las dos estrategias usará.

El otro elemento de gran importancia en la creación del proyecto es la inclusión de los algoritmos de aprendizaje, que son representados en la figura por **Algorithm**. Como podemos observar, un juego podrá tener varios algoritmos y cada algoritmo existirá en un juego. Este objeto tendrá los siguientes parámetros:

- **nEpisode**: representa el episodio actual del juego.
- **epsilon**: el valor del número epsilon que se usa en la política *epsilon-greedy*.
- **totalEpisodes**: número a establecer en caso de que se quiera limitar el número de episodios de aprendizaje.
- **maxSteps**: número a establecer en caso de limitar el número de pasos por episodios.
- **totalReward**: número de la recompensa total administrada por un algoritmo.

Al mismo tiempo cuenta con una serie de métodos. Algunos son getters y setters, y también cuenta con un método **create()** para algunos parámetros pero existen otros que tienen funciones más específicas:

- **learning(agent)**: es el método al que se llamará cuando se requiera que el algoritmo decida qué acción el agente *agent* debe tomar. Dentro del mismo se incluye además el proceso de administrar recompensas, por lo que es el método central de un algoritmo.
- **addReward(reward, agent)**: con esta función se otorga una cantidad de *reward* al agente *agent* según lo que haya pasado en el juego.
- **reset(agent)**: se reinicia el algoritmo y también la recompensa del agente *agent*.

Por supuesto, podemos tratar con algoritmos distintos o que funcionen de manera algo distinta internamente. Por ello, podremos tener distintos algoritmos dentro del proyecto los cuales al mismo tiempo heredan de un algoritmo general, e implementan sus características de forma diferente.

Adicionalmente, los algoritmos que utilizamos en el proyecto, necesitan una Q-Table, para guardar la información de la experiencia hasta el momento. Para ello, creamos el objeto **QTable**, que estará dentro del algoritmo. Esta tabla contará con los siguientes atributos:

- **Q:** una matriz que contará con estados en las filas y acciones en las columnas (o viceversa), y que contará con la recompensa esperada de realizar una acción en un estado.
- **alpha:** el factor de descuento a usar en la fórmula de actualización.
- **gamma:** la tasa de aprendizaje que se usará en la fórmula de actualización.
- **nActions:** el número de acciones que puede tomar un agente en todo momento.

Por otra parte también tiene una serie de métodos destacables a parte de sus getters y setters:

- **getRandomAction(state):** obtener una acción aleatoria para el estado *state*.
- **getMaxAction(state):** obtener la acción óptima para el estado *state* según la información basada en Q.
- **updateTable(state,state2,reward,action,action2):** utilizar la fórmula del algoritmo para actualizar los valores Q, usando los estados acciones y recompensas involucrados.

Por último, existirá en el juego también un controlador en caso de que queramos que los jugadores sean controlados por una persona. Esta es la clase **Controller**. Esta tendrá un atributo *cursor* que representa las teclas que se usarán para controlar este movimiento. También contará con el método **update()**, en el que comprobará qué tecla está siendo presionada para determinar en qué dirección mover al jugador y su getter y setter para el atributo *cursor*.

De esta forma observando estas clases y sus relaciones, podemos ver como contaremos con un entorno el estado del cual se reflejará en una vista. En este entorno contaremos con un varios elementos los cuales pueden ser plataformas o pelotas, los cuales tienen atributos y comportamientos compartidos y otros distintos. Las plataformas a su vez pueden contar con distintas formas de obtener su estado según su estrategia.

Por otra parte, dentro de este entorno también podemos contar con una serie de algoritmos, los cuales pueden ser distintos entre sí, y que introducen aprendizaje en el juego a través del uso de su **QTable**. La relación entre estas dos clases es la que permite a los agentes tomar acciones y aprender de ellas para el futuro. Por último dentro del juego también hay controladores en caso de querer decidir manualmente las decisiones de los elementos jugadores.

En la Figura 3.4 podemos ver un pequeño diagrama de casos de uso sobre el juego de Pong. Vemos que los dos que actúan sobre el juego son el agente que controla la plataforma según las decisiones del algoritmo, o un jugador humano que controla la pala por decisión propia. A medida que avanza la partida y en función a este movimiento, pueden ocurrir dos cosas que afecten al resultado del juego:

- Impactar la pelota, lo cual puede causar a su vez que posteriormente se sobrepase al rival. En ambos casos, siempre se obtendrá una recompensa (aunque mayor en el segundo caso).
- La pelota nos sobrepasa: si la plataforma con su movimiento no es capaz de darle a la bola, se castigará al agente para reflejar este resultado.

De esta manera podemos ver que la elección en el movimiento de la pala, lleva a situaciones beneficiosas o perjudiciales para el agente que la controle, y por lo tanto a castigos o premios por parte el algoritmo para fomentar el aprendizaje.

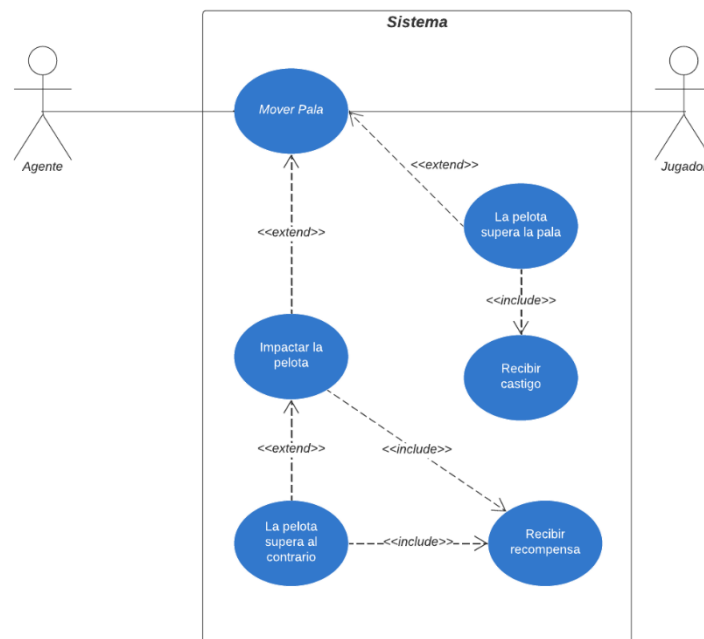


Figura 3.4: Diagrama de casos de uso.

4

Implementación

En este capítulo, se hablará sobre la implementación del trabajo; es decir hablaremos sobre el proceso de creación y programación de los juegos, la creación e implementación del algoritmo de aprendizaje por refuerzo y el desarrollo de la parte de aplicación web del trabajo con las funcionalidades que permitan realizar ciertas operaciones sobre el juego. En concreto contará con los siguientes puntos:

- Creación y desarrollo del juego de Pong y sus variaciones.
- Creación y desarrollo del juego del tres en raya; en este caso resaltarán las diferencias con el caso anterior, y se hará referencia al mismo cuando se apliquen técnicas similares.
- Desarrollo del algoritmo de aprendizaje por refuerzo e implementación en cada uno de los juegos.
- Creación de la funcionalidad de aplicación web.

4.1 Desarrollo de los juegos Pong

Empezamos primero con la descripción del desarrollo del juego del Pong.

4.1.1 Desarrollo del juego Pong principal

El juego del Pong es un videojuego en el que se intenta simular un tenis de mesa. En el juego hay dos jugadores en forma de palas paralelas al límite de la pantalla de donde se encuentren (si están a derecha/izquierda o arriba/abajo), y una pelota que se dirige en todo momento hacia uno de los lados donde se encuentra; de esta manera el objetivo de cada jugador es sencillo:

- Impactar la pelota para evitar que toque el límite de la pantalla donde se encuentra.
- Intentar dirigir la pelota de tal manera que al otro jugador le resulte difícil intentar darle a la misma.

Por lo tanto, ganaremos un punto si conseguimos que la pelota sobrepase el límite del otro jugador, y perderemos un punto si nos sobrepasa a nosotros mismos.

Así, para crear una partida de Pong necesitamos: un **campo de juego**, dos **palas** que serán los jugadores y una **pelota**.

El campo de juego se crea mediante una especificación de la configuración del juego en un archivo del proyecto. En este, tenemos que especificar la instancia del juego con la que tratamos (Pong, tres en raya, variaciones...), y los parámetros que nos interese. En nuestro caso, especificamos el tamaño del mismo y las físicas que se aplicarán; Phaser3 cuenta con unas físicas "arcade" que son las que usaremos en este juego.

Una vez especificamos estas propiedades, tendremos una instancia del juego (Environment) que se encarga de controlar la lógica del juego en un campo de juego que cumpla las condiciones establecidas. Cabe destacar dentro de esta clase **Environment** contaremos con unos métodos de inicialización que sirven para crear los elementos y parámetros que vaya a utilizar.

Tras los métodos de inicialización, se utiliza un método **update** que se ejecuta en cada frame de ejecución del juego y que sirve para actualizar los elementos que sean necesarios (mover un objeto, pararlo e incluso eliminarlo) y hacer comprobaciones sobre el estado del juego en general.

Así, únicamente creando la instancia del juego, tendríamos un recuadro del tamaño especificado en color negro, por lo que debemos añadir a la instancia del juego ambos jugadores y la pelota. Para ello, debemos crear clases para ambos tipos de objetos, una para las plataformas y otra para la pelota, siendo respectivamente **Platform** y **Ball**.

Ambos objetos así tendrán a su vez sus propias operaciones y métodos para su correcto funcionamiento en el juego. En ambos casos, tenemos también métodos de inicialización para establecer sus parámetros y otras características (como las colisiones con los límites del juego). También contamos con métodos para reiniciar el estado de cada objeto.

En el caso de la plataforma, tenemos métodos para cambiar la acción que esté realizando la pala (moverse a un lado u otro o quedarse quieto) y en la pelota habrá métodos para detectar su colisión con un jugador y modificar su velocidad y trayectoria adecuadamente en respuesta a esta.

Una vez hemos creado estos objetos; para hacer que aparezcan en el campo de la partida, declaramos ambos en los métodos de inicialización del juego correctamente, y después le asociamos la imagen que lo representará usando métodos propios de Phaser3.

Una vez creados los jugadores y la pelota; son posicionados en sus lugares correspondientes; en nuestro caso hemos posicionado los jugadores en los límites superiores e inferiores y la pelota en el centro como podemos observar en la Figura 4.1. Posteriormente, se otorga una velocidad aleatoria a la pelota, que puede dirigirse a cualquiera de los dos jugadores en distintos ángulos.

Además, cuando uno de los jugadores pierde, se reinicia a la situación inicial, con ambos jugadores en el centro de sus respectivos límites y la pelota de nuevo en el centro, aplicándole una nueva velocidad aleatoria, iniciando así otro episodio.

Así, lo único que falta para poder tener una partida de Pong, es que las palas se muevan. Como hemos dicho antes, las palas tienen métodos para cambiar su movimiento, pero hay que añadir elementos que le digan a la plataforma qué acción realizar.

Para ello, creamos un objeto **Controller**, el cual detecta cuando se presionan las teclas correspondientes a las distintas acciones y le traslada esta información a la plataforma para que se comporte de forma adecuada (por ejemplo, si presionamos la flecha izquierda, que se mueva a ese lado y si no se presiona nada no se mueve).

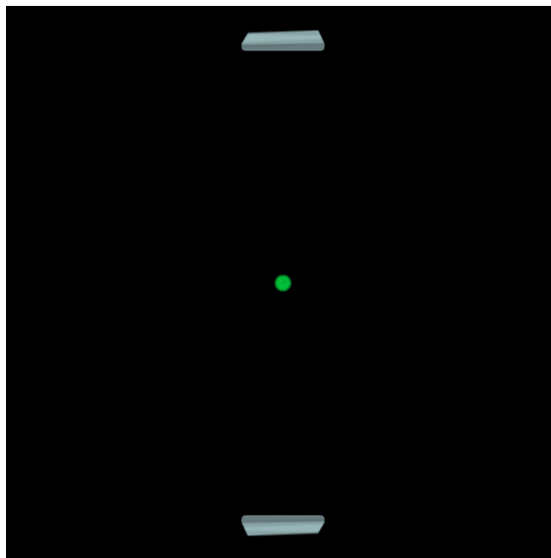


Figura 4.1: Ejemplo de estado inicial de nuestra versión de Pong.

Así, habiendo implementado todo lo anterior, contamos con un juego de un tamaño concreto (puede ser especificado), en el que dos palas se mueven a la izquierda o derecha para evitar que una pelota les sobrepase por arriba o abajo y cuando pasa a alguno de los dos, reinicia correctamente el estado del juego.

4.1.2 Variación del juego Pong

Por otra parte, hemos desarrollado una variación del Pong, en el que existe una estructura básica muy parecida pero con algunos cambios: el jugador de abajo mantiene su comportamiento del Pong normal, pero su objetivo para ganar cambia.

En esta versión, la pala de arriba aparece en una parte del límite superior de la pantalla y el jugador inferior debe hacer impactar la pelota el mayor número de veces posible en el superior sin que le sobrepase a sí mismo. Además, cada vez que el superior es impactado, aparece en otra parte aleatoria del límite superior.

Así, esta versión es bastante similar, pero la plataforma superior es estática y solo se reinicia por completo, cuando la pelota sobrepasa al jugador inferior; si se impacta al superior, solo se recoloca a este en otro lugar aleatorio. Por lo que podemos decir, que esta es una variación de un solo jugador.

4.2 Desarrollo del juego tres en raya

En el tres en raya, necesitamos crear un campo de juego dividido en cuadrículas, de modo que de forma similar al Pong, creamos una instancia de juego **Environment**, y dibujamos unas líneas que marquen los límites de cada casilla para representar visualmente las situaciones del juego como podemos observar en la Figura 4.2.

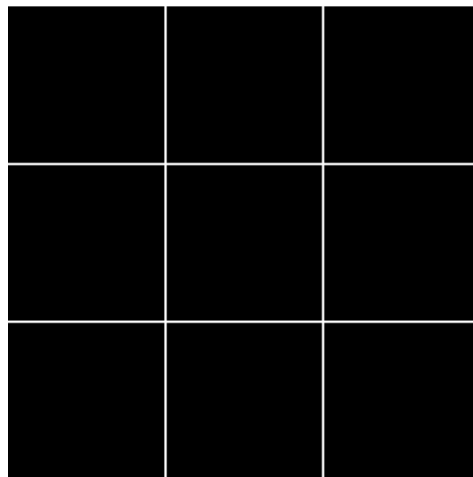


Figura 4.2: Ejemplo de estado inicial del tres en raya, antes de que ningún jugador haya actuado.

Una vez declaramos el juego, iniciamos los dos jugadores que van a participar por turnos sobre el tablero. En este caso, estos jugadores no se asocian a una imagen permanente sobre el campo de juego, pues no actúan sobre un objeto, sino que actúan modificando el terreno de la partida en sí. Para ello creamos la clase **Player** y los añadimos en los métodos de inicialización del juego.

De esta forma, cada jugador puede elegir uno de los nueve recuadros y ocuparlo, dejando un símbolo que lo represente sobre el espacio; en nuestro juego la X representa al jugador 1 y la O al 2. Obviamente, un jugador no podrá ocupar ningún espacio fuera del terreno de juego, ni ningún espacio ya ocupado por sí mismo o el contrario.

Así, como hemos establecido en el análisis, un jugador intentará encadenar tres símbolos propios, evitando que el contrario haga lo mismo, y rellenando los espacios de forma alterna. De esta forma podemos acabar con victoria de uno de ellos o un empate, en caso de que ninguno pueda conseguir su objetivo.

Teniendo todo esto en cuenta, tenemos que estructurar la instancia del juego de manera que se respeten las necesidades del mismo. Cabe destacar que a diferencia del Pong, en el tres en raya no hay ningún tipo de colisiones ni se

mueven objetos en el campo de juego. Los jugadores únicamente eligen una casilla y se coloca su símbolo en esta.

Así, en la clase del juego, tendremos que controlar tres factores importantes:

- Controlar correctamente los turnos de cada jugador, es decir que solo juegue el que tenga el turno en cada momento y el otro no pueda interrumpir.
- Comprobar si estamos en un estado final o no tras cada movimiento.
- Dibujar correctamente los símbolos de cada jugador, es decir, dibujar los símbolos en los sitios que le hayan señalado siempre que el movimiento sea aceptable. Si no lo es, no hacer nada.

De esta forma, una vez inicializado el juego e introducidos los jugadores, creamos los parámetros necesarios para controlar el estado del juego y dibujamos los límites del mismo para la representación visual para el usuario. El parámetro más propio en el caso del tres en raya en concreto es **board** un array de nueve números que se usará para controlar el estado del juego de forma interna; cada tres números se representa una fila del juego (la primera fila son los primeros tres números, la segunda los tres siguientes... etc). Los valores posibles para ese array van de 0 a 2, donde un 0 representa una casilla vacía, un 1 una casilla ocupada por el jugador 1 y 2 una ocupada por el jugador 2.

Una vez tenemos todos los elementos, se ejecuta en cada frame el método de actualización. En este, primero comprobamos si estamos en un estado final (victoria/derrota o empate), en cuyo caso reiniciamos el tablero borrando las casillas, y reiniciamos los parámetros correspondientes empezando una nueva partida.

Si no se ha terminado la partida, comprobamos de quién es el turno actual y esperamos a que el jugador realice una acción; en este caso, tenemos que esperar a que una persona elija una casilla con el ratón, pulsándola. Solo se continuará con la partida si se elige una casilla libre, el juego seguirá esperando si se intenta rellenar una casilla ocupada o si se pulsa fuera del campo de juego.

Una vez el jugador elige una casilla disponible, el juego utiliza unos métodos para transformar las coordenadas del clic en píxeles a filas y columnas y a su vez estas filas y columnas las traslada a un número entero para modificar correctamente el parámetro **board**. Asimismo, dibujamos los símbolos correspondientes sobre cada posición para reflejarlos visualmente en el juego.

Una vez realizada la acción, cambiamos el turno al otro jugador y volvemos a realizar la comprobación del final de juego, volviendo a repetir todo el proceso hasta que lleguemos a una victoria/derrota o a un empate. En las figuras 4.3 y 4.4 tenemos ejemplos de ambas conclusiones.

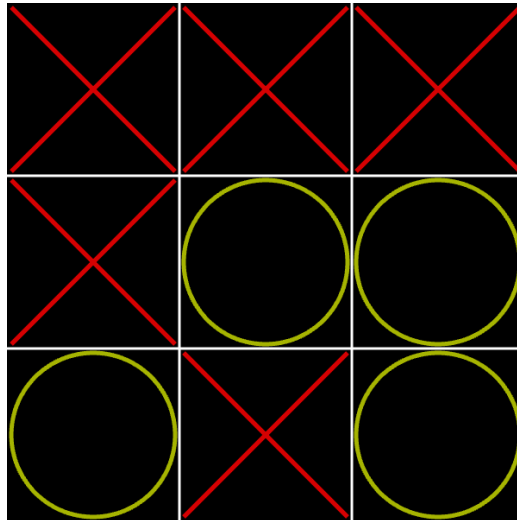


Figura 4.3: Ejemplo de estado de victoria del jugador 1 en el tres en raya.

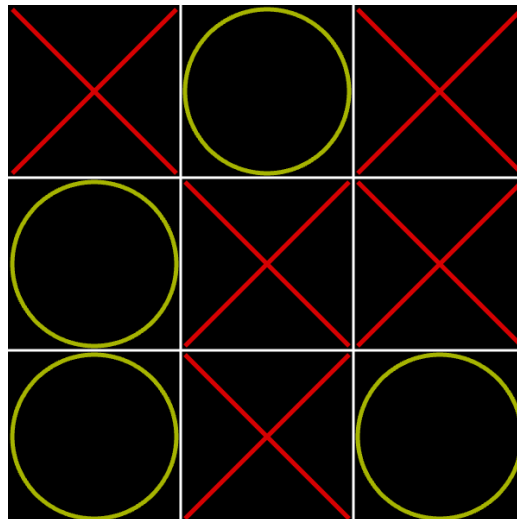


Figura 4.4: Ejemplo de estado de empate en el tres en raya.

4.3 Creación del algoritmo de aprendizaje por refuerzo

Como ya hemos establecido, para la creación de este proyecto hemos escogido el algoritmo de refuerzo SARSA. Para ello debemos crear un objeto que cumpla las pautas de este algoritmo para asegurar un correcto aprendizaje.

De este modo, creamos un objeto al que llamaremos **Algorithm** que representará a SARSA en nuestro proyecto y para ello deberá contar con las mismas características que el algoritmo. Para ello los elementos principales que necesitamos plasmar correctamente son la Q-Table y la fórmula de actualización de esta.

Así, para organizar el funcionamiento de este objeto, creamos una entidad dentro del algoritmo que represente a la tabla, llamada **QTable**. En este objeto contaremos con métodos que puedan explorar la tabla en busca de la mejor acción o una acción aleatoria en un estado (para la política *epsilon-greedy*) y también tendrán los parámetros y el método adecuados para la actualización de la tabla en base a la experiencia, siguiendo la fórmula de SARSA reflejada en la Figura 2.4.

Una vez contamos con la tabla, completamos el algoritmo con el objetivo de que sea un elemento que se comuniquen con el juego, los agentes y la tabla correctamente y administre las recompensas adecuadamente en respuesta a los resultados del juego.

El funcionamiento de esta clase girará en torno al método **learning**. Cada vez que un agente asociado a un algoritmo tenga que realizar una acción, se llamará a este método. Primeramente, comprobamos el estado del juego; dependiendo del mismo, podemos establecer una recompensa que recibirá más adelante por las acciones realizadas hasta el momento, si ha perdido se le asociará una recompensa negativa o positiva si ha ganado y, dependiendo del juego, también se le pueden dar pequeñas recompensas por distintas acciones.

Después, comprobamos si nos encontramos en el estado inicial del agente (no ha habido estados anteriores), en cuyo caso no podemos actualizar el conocimiento todavía, por lo que elegimos una acción siguiendo la política *epsilon-greedy* y se la trasladamos al agente, guardando el estado inicial y la acción que se ha decidido.

Cuando el agente tenga que volver a tomar una decisión volverá a recurrir a este aprendizaje. Volvemos a comprobar si hay una recompensa y, al contar ya con un estado y acción previos, podemos conseguir el estado actual del agente y la acción que vaya a tomar en el mismo. Combinando ambos grupos (estados y acciones previas/presentes) y la recompensa obtenida, podemos aplicar la fórmula de actualización de la QTable, para reflejar correctamente el conocimiento obtenido hasta el momento por el agente.

Una vez actualizada la tabla, se reinicia la recompensa a 0 y se actualizan los estados/acciones, eliminando los previos ya que hemos avanzado y ya no nos encontramos en la situación anterior. Posteriormente se le informa al agente de la acción que debe tomar, y la próxima vez que deba actuar, recurrirá de nuevo a su algoritmo.

Por supuesto, la implementación expuesta se trata de un punto de vista general, por lo que también es importante observar cómo aplicar este objeto a ambos juegos.

4.3.1 Implementación del algoritmo de aprendizaje en Pong

Para implementar el algoritmo dentro del juego de Pong, tendremos que primeramente introducir los objetos algoritmo al mismo tiempo que los agentes en los métodos de inicialización del juego. Una vez creados, tenemos que observar el funcionamiento de una partida y ver cuando es necesario llamar al método de aprendizaje (**learning()**).

En este caso, en el juego de Pong, los agentes podrán moverse en cualquier momento, no hay restricciones, por lo que podríamos llamar al algoritmo en cada frame del juego mediante el método **update**. Así, lo que habría que hacer es sustituir el manejo de la plataforma mediante el controlador, por una llamada al aprendizaje del algoritmo correspondiente con el agente.

Esto hará que el agente realice una acción siguiendo las indicaciones del algoritmo e irá aprendiendo a medida que transite entre estados distintos. En el Pong se le otorgará al agente una recompensa grande cuando consiga ganar una partida, una recompensa menor cuando consiga impactar la pelota y un castigo cuando le sobrepase la pelota y pierda.

Sin embargo, como hemos expuesto antes, al pasar la información al algoritmo, necesitamos incluir las posiciones de algunos elementos del juego. En el caso del Pong este será una cadena de texto que incluirá la posición horizontal y vertical de la pelota, la posición horizontal del agente del que obtengamos el estado y la dirección general a la que se dirige la pelota (derecha/izquierda y arriba/abajo).

Por otro lado, la inclusión de las posiciones, debe ser cuantificada, pues si estas fueran representadas por la localización en píxeles, nos encontraríamos con una cantidad de estados distintos demasiado alta. En otras palabras, si una plataforma se moviera un píxel hacia la izquierda, ya estaríamos en otro estado que tendríamos que guardar, a pesar de que a efectos prácticos, un desplazamiento de un píxel no diferencia a ambos estados lo suficiente como para tener que tomar otra decisión.

Para evitar este exceso de estados innecesarios, dividiremos el campo de juego en filas y columnas, obteniendo una cuadrícula. De esta forma, las posiciones de los elementos del juego, no se obtendrán por píxeles, sino por la fila y columna en la que se encuentre. Podemos ver un ejemplo en la Figura 4.5:

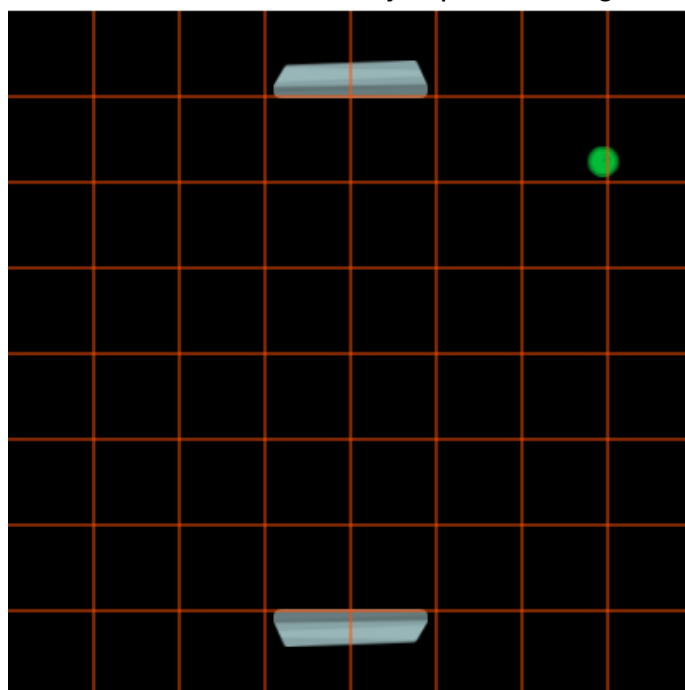


Figura 4.5: Ejemplo de la cuantificación en el Pong.

Así habiendo cuantificado el juego, podemos en cada frame consultar al algoritmo para saber qué acción debe realizar el agente, sin generar una gran

cantidad de estados. También cabe destacar, que en esta implementación, una vez que se ha decidido qué acción realizar en un estado, esta acción es la que se hará hasta que se deje el mismo. Es decir, si al entrar a un estado se decide que se moverá a la derecha, seguirá haciéndolo hasta que algún elemento del estado cambie (alguna posición o la dirección de la pelota).

Por otra parte, para el Pong estándar podemos considerar dos formas de incluir los algoritmos: podemos crear dos algoritmos distintos para cada agente, o podemos usar el mismo algoritmo para ambos, usando los dos agentes para entrenar al mismo algoritmo. Hasta ahora hemos hablado de la versión de dos algoritmos.

Para implementar el aprendizaje de esta otra forma, el mayor cambio que hay que incluir, es a la hora de obtener el estado del juego del agente. Antes, en el método de aprendizaje (**learning()**) se llamaba al agente para saber en qué estado estaba. Esto se hacía de forma normal, viendo el campo de juego de forma general y trasladándolo al algoritmo para guardarlo en su Q-Table.

Sin embargo, en este caso, ambos agentes comparten la tabla, por lo que si guardamos por ejemplo un estado en el que un jugador ha ganado y el otro ha perdido, la recompensa y el castigo se aplicarían en la misma posición de la tabla y no serviría de nada. Por ello, cuando un agente tenga que pasar su estado tendremos que hacer que cada uno lo pase desde su punto de vista los cuales serán simétricos.

Es decir, para cada estado, un agente tomará las posiciones y velocidad como se ven desde la perspectiva general y el otro lo tomará desde una simétrica: para ello cogemos la posición de la pelota y se la restamos al número de filas y la dirección vertical de la velocidad de la pelota es invertida. Esto lo hacemos mediante una aplicación del patrón de diseño Estrategia, teniendo así una estrategia para la plataforma inferior y otra para la superior.

De este modo, los estados de los dos agentes no son iguales por lo que el conocimiento no se solapa, pero también obtenemos conocimiento útil en todo momento.

Por último, el uso del algoritmo en la variación del Pong de un solo jugador, es bastante parecido al Pong normal, por supuesto, en este juego el único agente es el jugador inferior, y por tanto solo tenemos un algoritmo para este. Las recompensa mayor en este caso se otorga por impactar a la otra pala y las demás son iguales que en el Pong regular. La otra diferencia importante es que el agente debe incluir la posición de la otra plataforma en sus estados, ya que esta localización es el objetivo directo y es importante incluirlo (no es lo mismo intentar impactar la esquina izquierda que la derecha).

4.3.2 Implementación del algoritmo de aprendizaje por refuerzo en el tres en raya

De forma similar al Pong, la implementación del algoritmo en el tres en raya debe sustituir la intervención humana en el juego. Así, primeramente si la partida actual no se ha terminado, llamaremos al algoritmo del agente cuyo turno sea el actual para que elija la acción a tomar.

Así, dentro del aprendizaje de este algoritmo primero comprobamos si hace falta aplicar recompensas/castigos. Posteriormente, se elige la acción a tomar, sin embargo, en este juego tenemos que tener en cuenta que no en todo estado se puede tomar cualquier acción.

De este modo, el algoritmo elegirá una acción de las nueve casillas y en caso de que esté ocupada, volverá a elegir otra siguiendo su política hasta encontrar una válida. Posteriormente, si es posible, actualiza su tabla e informa al agente de su siguiente acción.

Como hemos establecido, después de cada movimiento se comprueba si hemos llegado a un estado final del tres en raya. Si este es el caso, llamamos al aprendizaje (**learning()**) de ambos agentes para comunicarles el resultado y que estos puedan administrar correctamente la recompensa actualizando su Q-Table pero esta vez sin darle una nueva acción a su agente, ya que el episodio ha terminado. En el tres en raya las recompensas serán: mayor por ganar la partida, menor por conseguir un empate y un castigo en caso de perder la partida.

Una vez aplicadas las recompensas reiniciamos los jugadores, el tablero y pasamos a un nuevo episodio para los algoritmos empezando una partida nueva con los conocimientos de las anteriores.

4.4 Creación de la aplicación web y funcionalidades extras

Al haber creado todos estos juegos, los insertamos en un entorno de aplicación web, insertándolos en una página creada con HTML. Teniendo en cuenta que más adelante queremos analizar el comportamiento de los algoritmos entrenados y su progresión queremos crear una serie de funcionalidades dentro de la aplicación web para asistir este objetivo:

- **Simular:** esta función deberá acelerar el proceso de funcionamiento de los juegos ampliamente, realizando las partidas mucho más rápido y por supuesto, entrenando a los algoritmos mucho antes, por lo que podemos observar los resultados de estos más fácilmente. Este proceso se puede comenzar y detener, volviendo a un ritmo de juego normal.
- **Tomar control:** el control del jugador 1 pasa a un jugador humano. En el caso del Pong usaremos las flechas para controlar las acciones y en el tres en raya usaremos el ratón seleccionando las casillas. Mientras el control del agente esté en manos de una persona, este no aprenderá, pero si lo hará el oponente.

- **Descargar información:** descarga un resumen de las recompensas del algoritmo del jugador 1. Cada cien episodios el agente guarda en un parámetro la media de las recompensas otorgadas durante esas partidas. Mediante esta funcionalidad podemos descargar un archivo de texto que contiene esta información y así podemos observar cómo ha evolucionado el comportamiento (cuanto mayor sea esta media, más recompensa ha obtenido).
- **Descargar/Cargar algoritmo:** mediante la descarga de un algoritmo, podemos obtener una representación del conocimiento obtenido por el algoritmo del jugador 1 en un archivo de texto y con la carga podemos sustituir el algoritmo del jugador 1 actual por el de un archivo. Es decir, podemos descargar un algoritmo entrenado ampliamente e introducirlo en una instancia nueva del juego recién empezada lo cual haría que el jugador 1 actuara de inmediato de forma efectiva en el juego, porque su algoritmo ya estaría entrenado.



Figura 4.6: Ejemplo de una ejecución del Pong con las funcionalidades extra encima.

Así, con todo esto tendríamos una aplicación web en la que contamos con un juego en el que se pueden enfrentar dos agentes o una persona contra un agente y del que podemos recabar información usando las funcionalidades adicionales. El siguiente paso sería usar esta información para comprobar la efectividad de los algoritmos creados y su implementación en los juegos.

5

Análisis Experimental

En este capítulo haremos una serie de pruebas con los juegos creados. En cada una de estas dejaremos que dos agentes jueguen un cierto tiempo. Pasado este, guardaremos la recompensa obtenida a lo largo de la ejecución y la intentaremos analizar.

Así examinaremos esta recompensa media obtenida con el objetivo de comprobar en qué tipo de juego los agentes aprenden mejor y qué condiciones afectan mejor o peor a este aprendizaje.

Todas la información se ha obtenido de las recompensas medias obtenidas por el jugador 1 cada cien episodios (la primera información que tendríamos sería la media de las recompensas de los 100 primeros episodios). Como ya hemos comentado, para acelerar el proceso hemos utilizado la función de simulación del juego y la información se obtiene del archivo de texto de la función de descargar información.

Cabe destacar los valores de los parámetros de los algoritmos: epsilon con 0,1; alfa con 0,1 y gamma con 0,9. Estos valores han sido determinados por unas pequeñas pruebas anteriores con las que los hemos ajustado.

Para reflejar esta información usaremos una serie de gráficas que representen la evolución de las recompensas conseguidas por el algoritmo conforme transcurren los episodios.

5.1 Pruebas del Pong

En este apartado exponemos los resultados del análisis de la versión tradicional del juego Pong. En este caso hemos realizado cuatro pruebas distintas, diferenciadas por el tamaño del campo de juego en píxeles:

- Pong de tamaño 200 de alto y ancho.
- Pong de tamaño 400 de alto y ancho
- Pong de tamaño 600 de alto y ancho.
- Pong en el que el tamaño cambia a lo largo del tiempo.

Por último, en estas pruebas se otorgan recompensas de tres tipos:

- **Premio mayor:** se le otorgan al agente 100 puntos si consigue hacer que la pelota sobrepase al contrario.
- **Premio menor:** se le otorgan al agente 50 puntos por impactar la pelota.
- **Castigo:** se le castiga al agente con 100 puntos menos si la pelota le sobrepasa.

5.1.1 Pruebas en Pong con tamaño fijo

En esta sección exponemos los resultados de las pruebas del Pong en un mapa de tamaño fijo. Realizaremos tres pruebas y entre cada una iremos aumentando el tamaño de forma considerable, comprobando así en qué medida afecta al aprendizaje el aumentar la distancia que tiene que cubrir un agente y por tanto la dificultad del juego. Así, cada uno tendrá las mismas condiciones, solo variando el tamaño del juego.

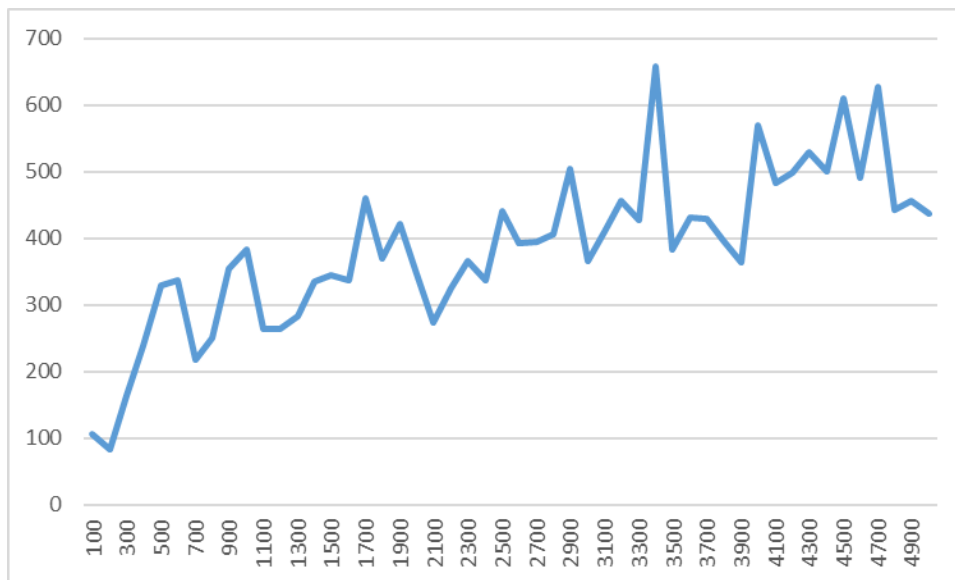


Figura 5.1: Evolución de las recompensas en un Pong de 200x200

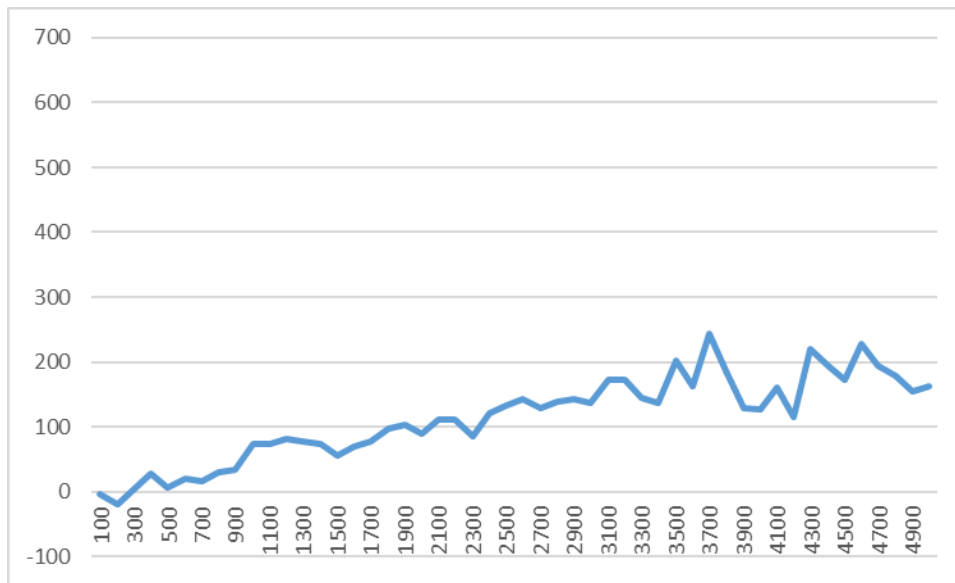


Figura 5.2: Evolución de las recompensas en un Pong de 400x400

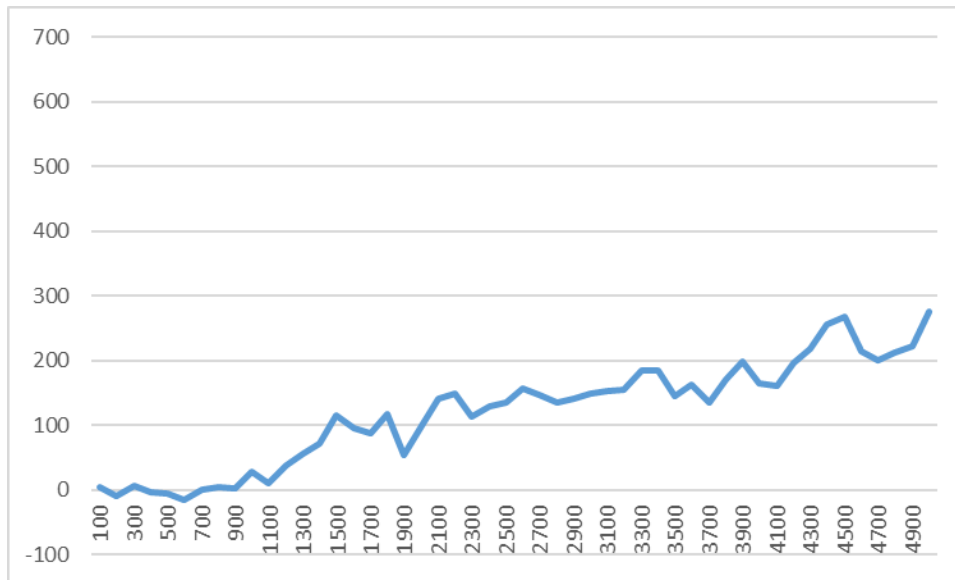


Figura 5.3: Evolución de las recompensas en un Pong de 600x600

Así, como podemos observar en las figuras 5.1, 5.2 y 5.3, en los tres casos existe una tendencia al crecimiento, por lo que se puede decir que el agente está aprendiendo. Al mismo tiempo, dicha tendencia es mucho más pronunciada en el juego más pequeño, que en los otros dos ya que es mucho más fácil para el jugador darle a la pelota.

Sin embargo, este no es un caso muy realista, pues el movimiento del jugador para impactar la pelota es básico en el juego de Pong y crear un entorno en el que el agente es recompensado en muchos casos con un mínimo movimiento no representa el juego normal.

Por otra parte, los casos de 400x400 y 600x600, son más parecidos, pudiendo ver que en el primero la tendencia inicial es algo más alta, aunque tras los 1000 primeros episodios es más parecida y al final la media se encuentra entre los

200 puntos para ambos, lo cual implica por lo menos impactar la pelota un número considerable de veces por partida.

5.1.2 Pruebas de Pong con tamaño variable

En esta sección realizaremos una prueba del Pong, pero en este caso, iremos aumentando el tamaño del campo de juego. Cada 100 episodios comprobaremos la media de la recompensa obtenida por el jugador 1. Si esta supera los 100 de valor, aumentaremos 10 píxeles de alto y ancho así hasta llegar a 600 por 600.

En caso de que no se superen los 100 de valor, tendremos que esperar a que vuelvan a producirse otros 100 episodios. De esta forma, con este experimento queremos comprobar si el hacer que el tamaño vaya aumentando progresivamente, beneficia más al agente a aprender y adaptarse a estos tamaños más grandes con respecto a hacer que se enfrente a estos entornos más difíciles desde el principio. Es decir, queremos ver que facilita más el aprendizaje del agente: el introducir dificultad progresivamente; o desde el principio.

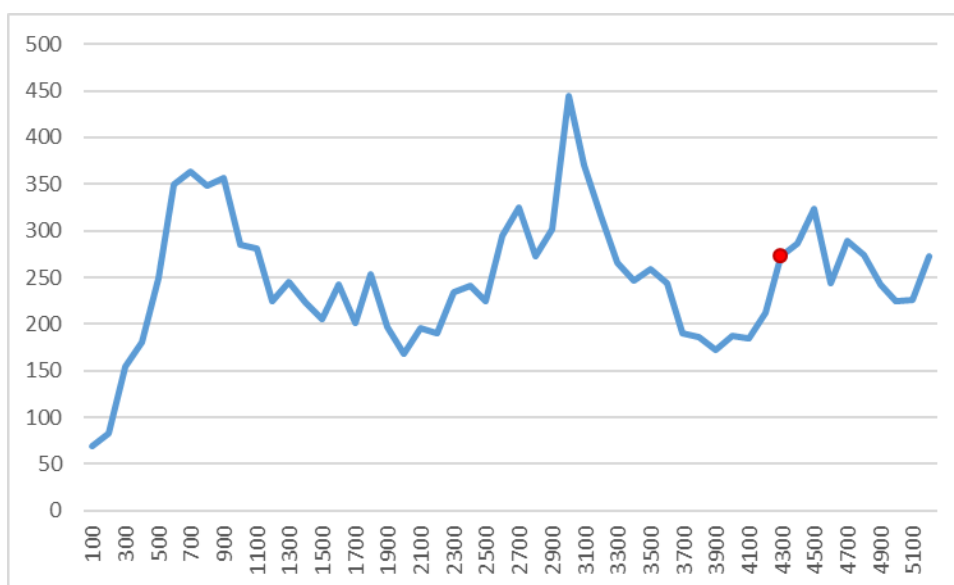


Figura 5.4: Evolución de las recompensas en un Pong con crecimiento (el punto rojo representa el momento en el que llega a un tamaño de 600x600).

En la Figura 5.4 se puede ver así el progreso de esta recompensa. Al principio, la tendencia es bastante alta a crecer al contar con un tamaño inicial pequeño sin embargo, a medida que crece le resulta más difícil mantenerlo alto. Podemos ver que va subiendo y bajando a medida que va cambiando de tamaños.

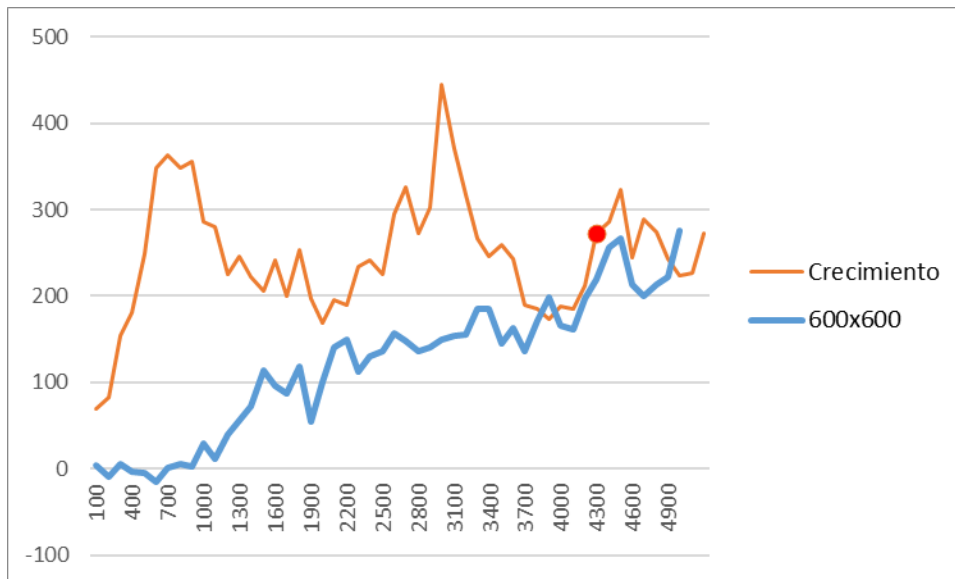


Figura 5.5: Comparación de las recompensas obtenidas en un juego con crecimiento y uno con tamaño fijo (el punto rojo representa el momento en el que el juego con crecimiento llega a un tamaño de 600x600).

En la Figura 5.5 podemos ver una comparación directa entre un juego de Pong con tamaño 600 y tamaño variable. Obviamente en los episodios iniciales el Pong con crecimiento obtiene mejores recompensas, pero cuando llegan a tener tamaños iguales sus tendencias son bastante parecidas. Por lo que podemos ver que con un campo creciente el agente conserva su conocimiento entre tamaños.

5.1.3 Pruebas de Pong con el mismo algoritmo

Hasta ahora, en las pruebas del Pong usábamos dos objetos algoritmo distintos. En este caso, los agentes usarán el mismo algoritmo, usando estados simétricos para poder distinguirlos como hemos explicado anteriormente. Las demás condiciones y asignaciones de recompensas de estas pruebas son iguales al Pong normal.

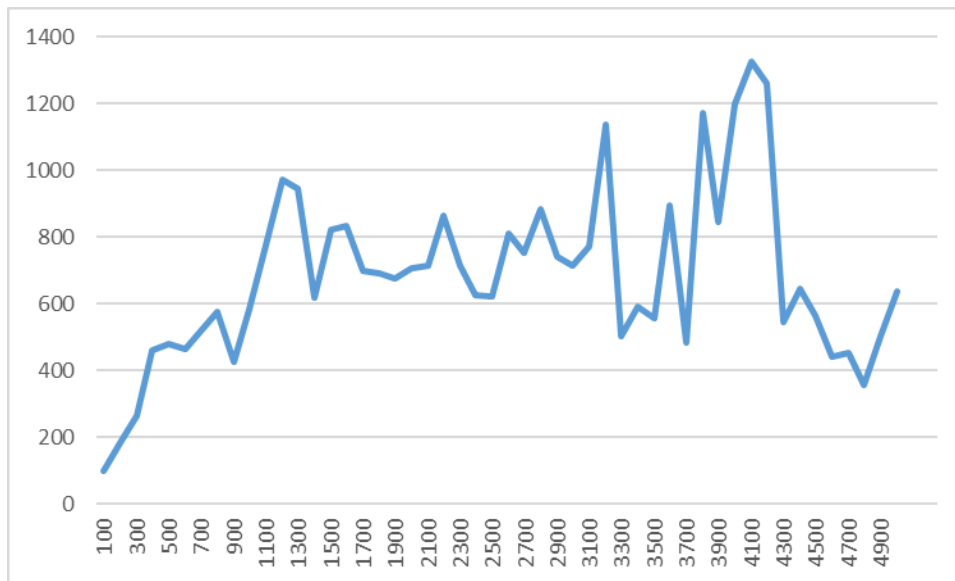


Figura 5.6: Evolución de las recompensas en un Pong con algoritmo compartido 200x200

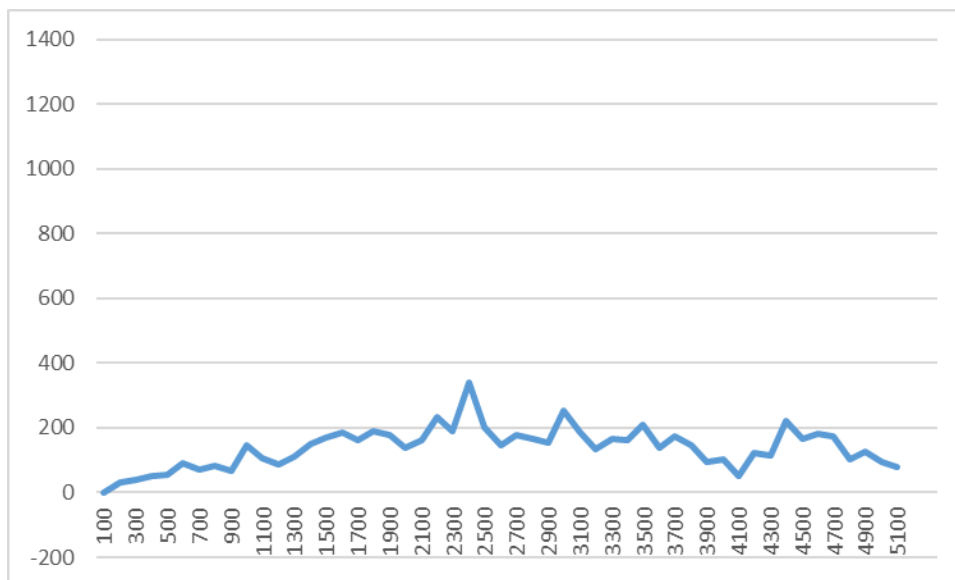


Figura 5.7: Evolución de las recompensas en un Pong con algoritmo compartido 400x400

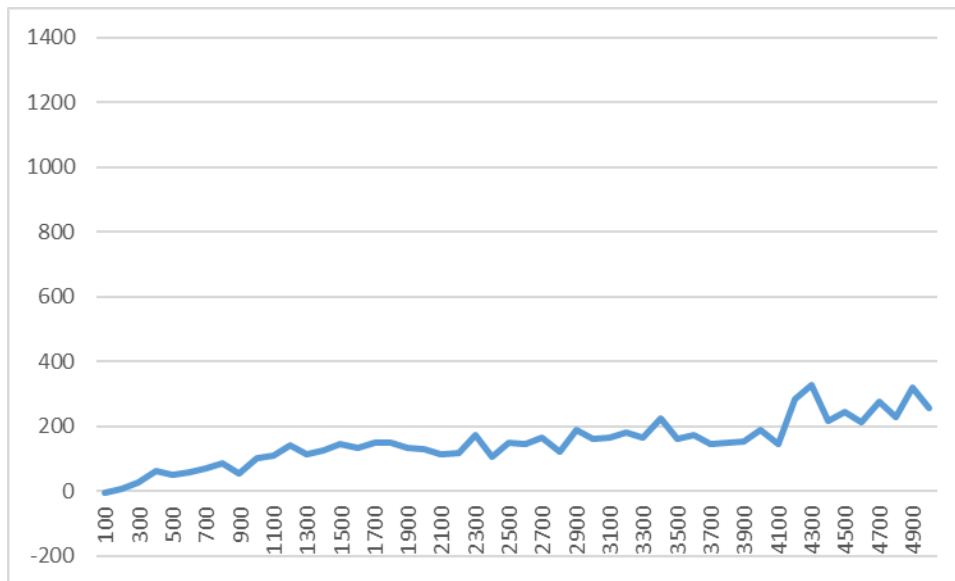


Figura 5.8: Evolución de las recompensas en un Pong con algoritmo compartido 600x600

De nuevo podemos ver como un tamaño pequeño lleva a una recompensa mucho mayor al ser más fáciles de obtener. Por otra parte los otros tamaños son más difíciles de aprender, pero las recompensas son bastante parecidas a los algoritmos separados, por lo que ambos enfoques son efectivos en el aprendizaje.

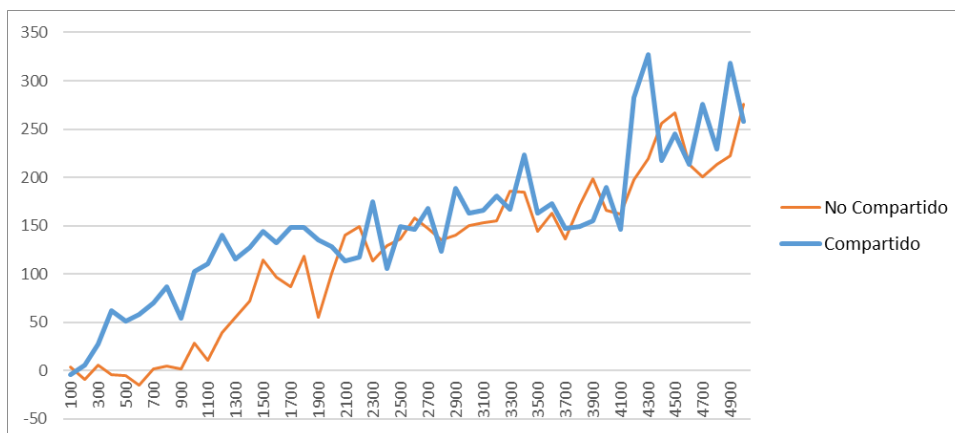


Figura 5.9: Comparación de recompensas en tamaño 600x600 con algoritmo compartido o no compartido

En la gráfica de la Figura 5.9 podemos ver una comparación entre el uso de algoritmos compartidos y separados en el Pong. Se puede observar, que en general parece que el uso de un algoritmo para ambos agentes resulta beneficioso, sobre todo al principio, donde la tendencia a crecer es mayor y además se alcanzan unos picos más altos.

Esto se puede observar en los otros tamaños, en los que también la tendencia a aumentar la recompensa y los valores obtenidos de la misma es algo mayor cuando se comparte el algoritmo que cuando no se hace.

5.1.4 Pruebas de Pong con el mismo algoritmo y tamaño variable

En este caso, usaremos el mismo algoritmo para ambos agentes en un Pong con campo de juego que aumenta.

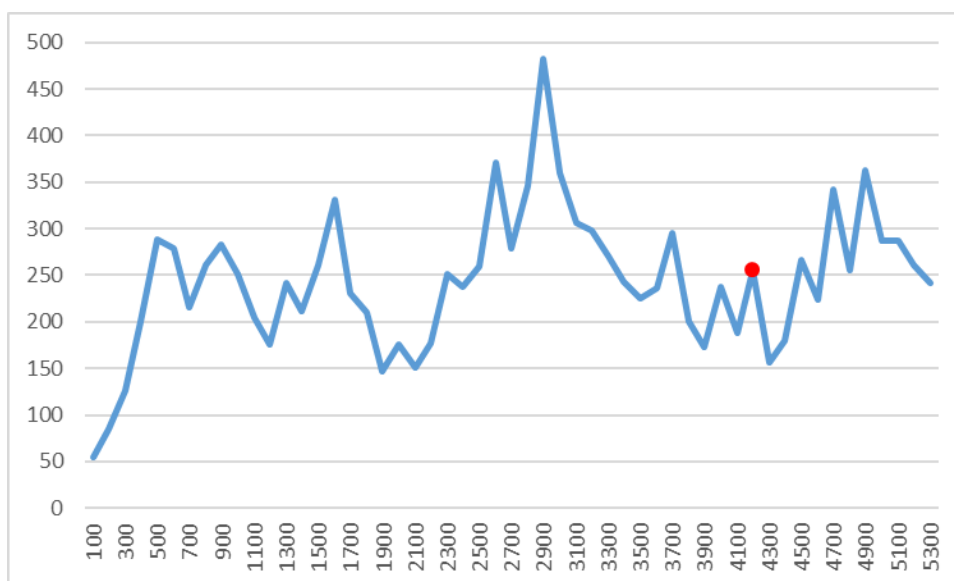


Figura 5.10: Evolución de las recompensas en un Pong con algoritmo compartido con crecimiento (el punto rojo indica la llegada a 600x600)

Similar al Pong normal, podemos ver como la tendencia va bajando y subiendo a medida que el tamaño cambia, y a excepción de al principio, consigue aumentar el tamaño del juego en todas las ocasiones.

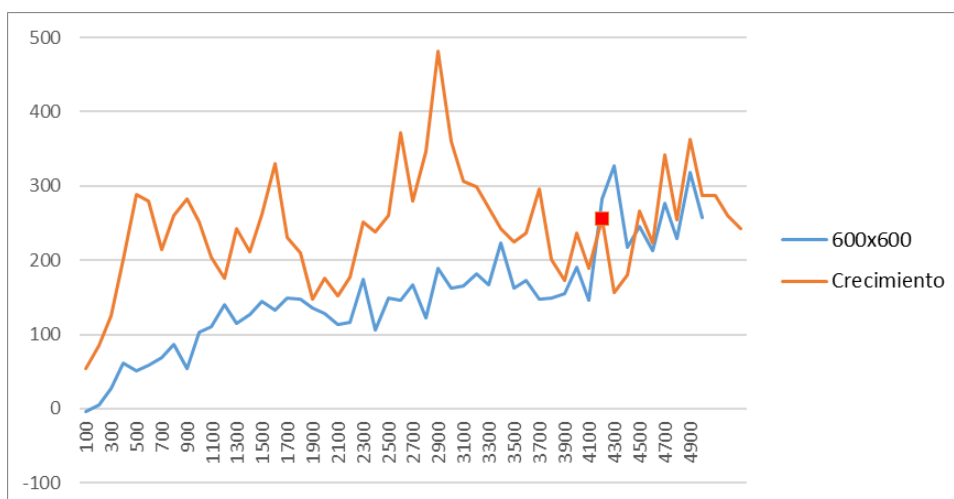


Figura 5.11: Comparación de la evolución de la recompensa en Pong con algoritmo compartido en tamaño estático o creciente (el punto rojo representa la llegada al tamaño 600x600 para el Pong con crecimiento).

Si nos fijamos en la Figura 5.11, una vez más, el Pong con tamaño variable tiene al principio campos de juegos más manejables, por lo que recibe mayores recompensas, pero una vez igualados sus tendencias son bastante parecidas.

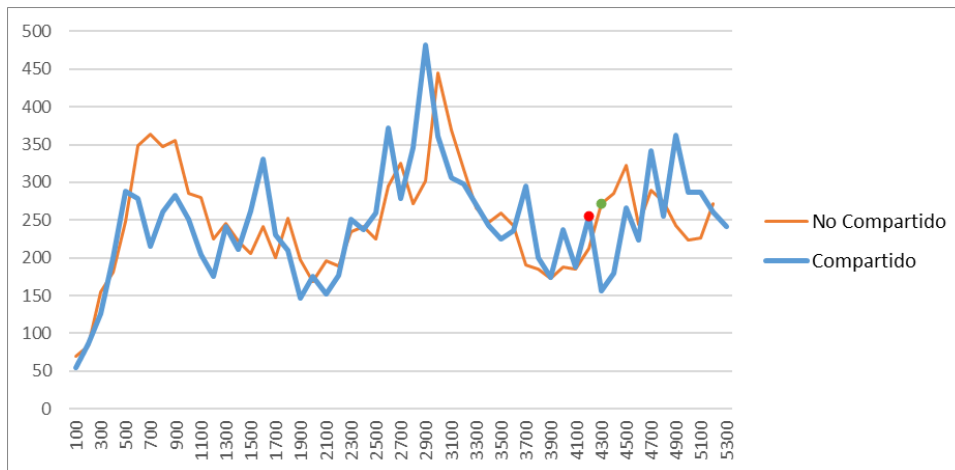


Figura 5.12: Comparación de la evolución de la recompensa en Pong con algoritmo o no compartido con crecimiento (los puntos rojo y verde representan el alcance al tamaño 600x600 en la versión con algoritmo compartido y no compartido respectivamente)

En la Figura 5.12 podemos ver una comparación entre usar algoritmos compartidos o no en un tamaño de juego con crecimiento. Podemos ver en este caso las tendencias son bastante parecidas, incluso llegan al tamaño de 600 por 600 en episodios muy cercanos, por lo que en este caso el rendimiento es muy similar aunque cabe destacar que el algoritmo compartido alcanza un poco antes el tamaño de 600 por 600 y sus picos mayores son más altos que los del algoritmo no compartido.

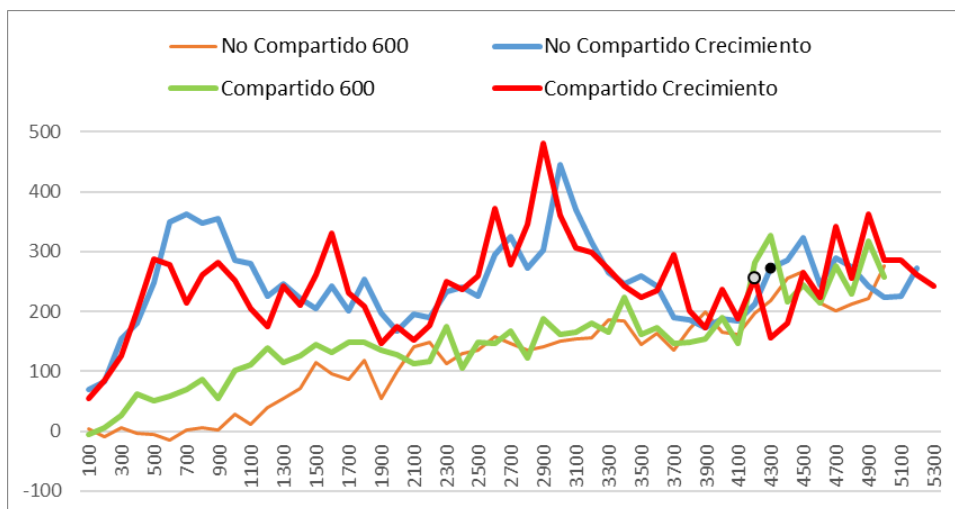


Figura 5.13: Comparación de la recompensa para todos los Pong de tamaño 600x600 o crecimiento y compartan o no algoritmo (los puntos gris y negro representan cuando llegan a 600x600 los de crecimiento compartido y no compartido respectivamente).

Por último, en la Figura 5.13, podemos ver una comparación entre todos los juegos Pong en tamaño 600 por 600 y con crecimiento, introduciendo además si comparten algoritmo o no. De esta forma, podemos ver como al principio, los juegos de crecimiento, se benefician al no tener un campo de juego tan grande, aunque sus valores se acercan más cuando llegan al tamaño máximo. A pesar de ello podríamos decir que usar algoritmos compartidos funciona algo mejor que usarlos por separado.

En definitiva, las mejores recompensas se encuentran una vez más en el Pong de 200 por 200, pero esto representa una situación incluso demasiado fácil para el agente. Las situaciones de 400 o 600 son más realistas para ver un progreso de las recompensas. Además añadir un factor de crecimiento le otorga una tendencia algo distinta a las de tamaño fijo.

5.1.5 Pruebas de la variación del Pong

En esta sección trataremos de realizar pruebas sobre la versión alterna del Pong. En este caso solo tratamos con un agente el jugador 1 del que sacaremos las recompensas observadas. En este además las recompensas se distribuyen de forma parecida pero cambiando ligeramente:

- **Premio mayor:** se le otorga una recompensa de 100 puntos si logra hacer impactar a la pelota con la pala superior.
- **Premio menor:** se le otorga una recompensa de 100 puntos si logra darle a la pelota, evitando que le pase.
- **Castigo:** se le da un castigo de 100 puntos si la pelota sobrepasa al jugador.

Así, el objetivo principal para el agente sería impactar la zona concreta en la que se encuentre la otra pala.

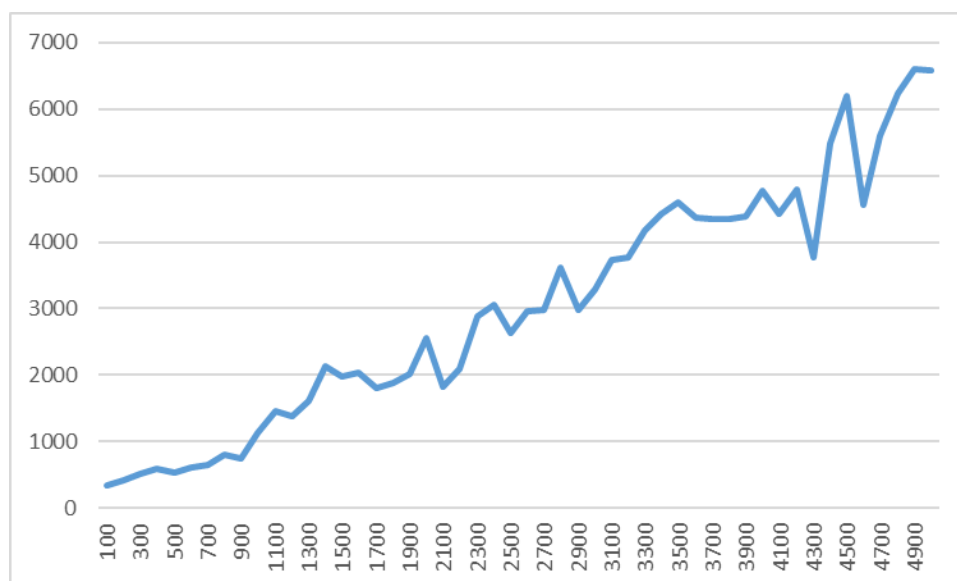


Figura 5.14: Evolución de la recompensa en la variación del Pong de tamaño 200x200

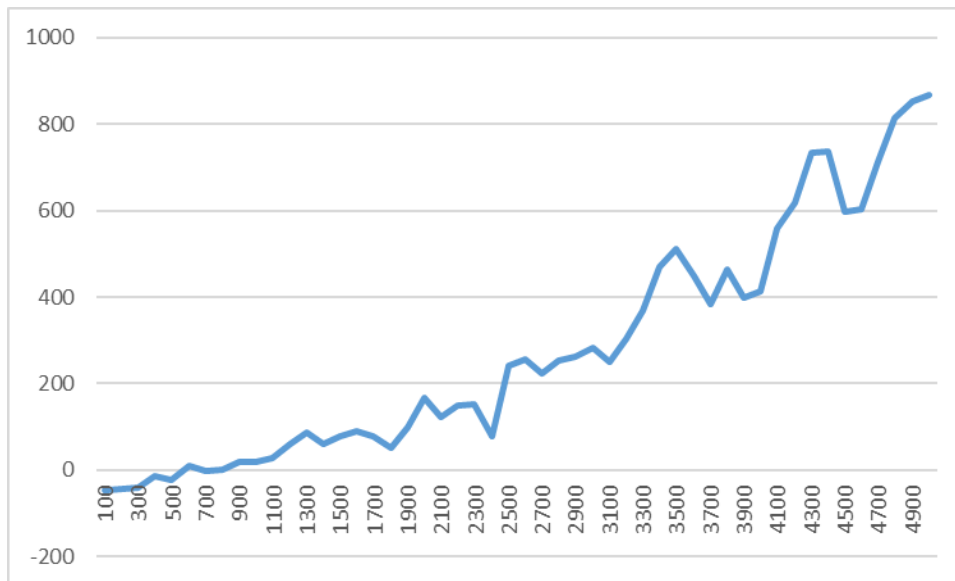


Figura 5.15: Evolución de la recompensa en la variación del Pong de tamaño 400x400

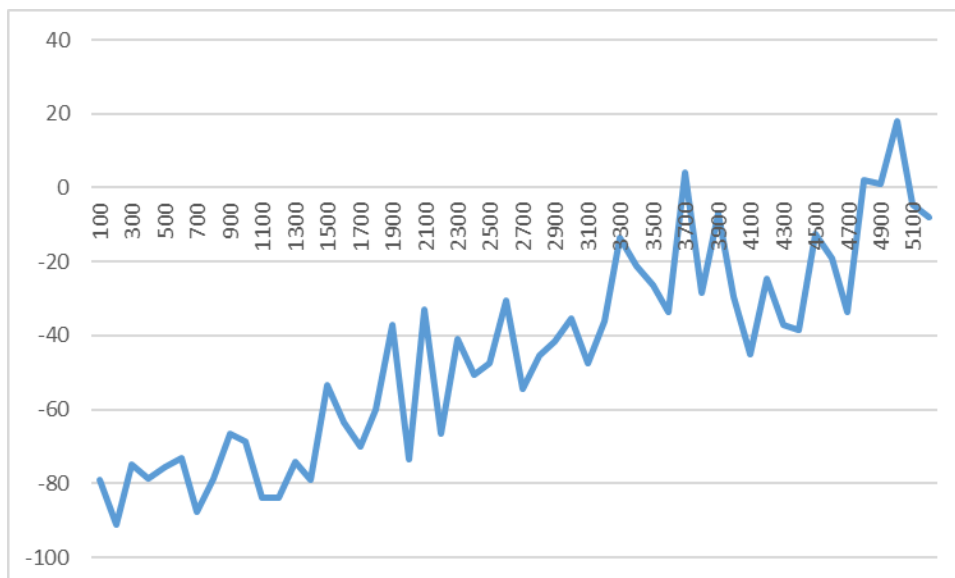


Figura 5.16: Evolución de la recompensa en la variación del Pong de tamaño 600x600

Una vez más, el juego en el tamaño más pequeño tiene una tendencia y unos resultados muchos más altos que los demás. De hecho, como es lógico, las instancias que usan un campo de juego más pequeño, consiguen recompensas mucho mayores incluso comparando un tamaño de 400 por 400 y 600 por 600.

Además, si comparamos estos resultados a los del Pong podemos observar que en el caso del tamaño 200, la versión alterna es mucho mayor en términos de recompensa y tendencia, algo que también se aplica al 400, pero no es el caso en el 600 por 600. En este último podemos ver que las recompensas son incluso negativas, y aunque la tendencia es a crecer, no se acerca a las recompensas finales de los Pongs normales de 600x600, por lo que en esta versión del Pong influye más que en el anterior el tamaño del juego.

5.2 Pruebas del tres en raya

Las últimas pruebas que queremos realizar, serán en un juego del tres en raya. De nuevo obtendremos las recompensas del jugador 1 durante una simulación asignando las recompensas de la siguiente manera:

- **Premio mayor:** se otorga una recompensa de 100 puntos si se gana una partida.
- **Premio menor:** se otorga una pequeña recompensa de 50 puntos si se empata la partida.
- **Castigo:** se le da un castigo al agente de 100 puntos si pierde la partida.

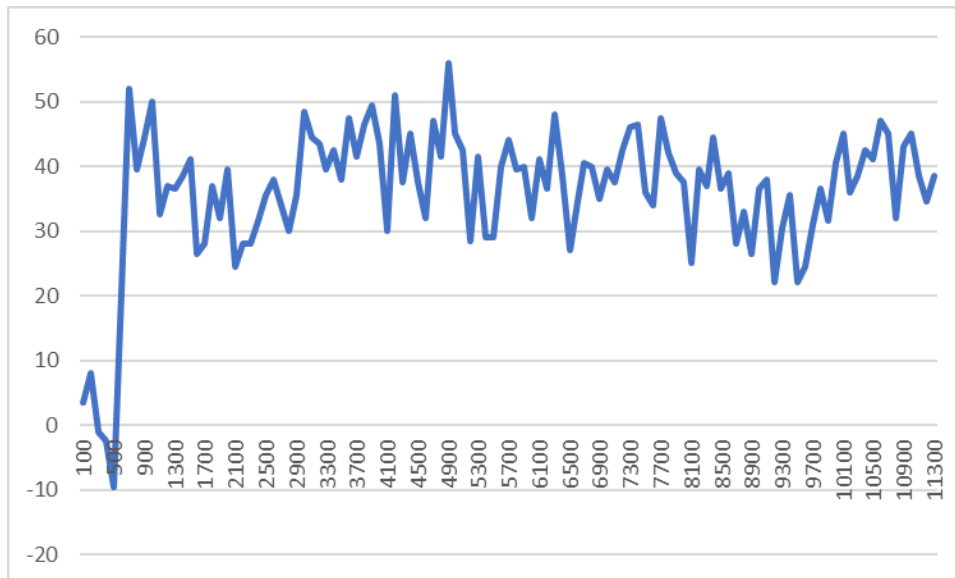


Figura 5.17: Evolución de la recompensa en el tres en raya

Así, en la Figura 5.17 podemos observar una rápida subida de las recompensas antes del episodio 1000, y por el resto del análisis se mantiene entre los 50 y 30 más o menos. Si ponemos esto en el contexto del juego es fácil de explicar.

En el tres en raya, es muy fácil empatar cuando se aplica estrategia. Por ello, una vez que el agente tiene un poco de conocimiento sobre el juego y lo aplica, aumentan bastante las probabilidades de obtener ese resultado. Es por ello que las recompensas, pasado el inicio, oscilan de forma constante ni crece ni desciende porque ambos agentes con conocimiento intentan obtener el mejor resultado posible. Además, al ser la recompensa depende únicamente del resultado (en el Pong aunque perdieras podías recoger recompensa si te defendías eficientemente anteriormente), las posibles recompensas que se pueden obtener son más limitadas.

6

Conclusiones y Trabajos Futuros

En este capítulo hablaremos sobre algunas conclusiones que podemos sacar del proyecto realizado, así como de posibles líneas de trabajo con las que se pueden expandir el contenido actual que hemos desarrollado.

6.1 Conclusiones

Con este trabajo, pretendíamos crear un entorno online en el que se pudiera producir un aprendizaje por parte de unos agentes que participan en distintos juegos y comprobar las efectividades de implementarlo de distintas formas y bajo reglas distintas.

Así, para comprobar el rendimiento de este aprendizaje, hemos realizado el Capítulo 5 en el que demostramos que las recompensas obtenidas a lo largo del tiempo van creciendo con el paso del tiempo en todos los casos con el Pong y sus versiones alternas; en el caso del tres en raya, se obtiene rápidamente suficiente conocimiento como para jugar eficientemente durante la mayoría de las partidas.

De esta forma, podemos decir que sí es posible realizar un aprendizaje en todos los juegos creados y en un entorno de página de web; con rendimientos distintos según su implementación lo cual permite contar con cierta flexibilidad para crear distintos tipos de aplicaciones según los objetivos perseguidos.

Por otra parte, todo este proyecto ha aportado mucho a la experiencia personal del alumno; a través de las fases y dificultades que se han pasado en el trabajo, se ha obtenido un conocimiento mucho más amplio sobre las distintas herramientas y áreas de aprendizaje que se han utilizado en el mismo.

Obviamente, el enfoque en el aprendizaje automático y concretamente el aprendizaje en refuerzo, ha hecho que el alumno tenga que familiarizarse con estos campos y aprender de forma más profunda sobre sus algoritmos. Es decir, el conocimiento básico que a lo mejor se tenía antes sobre estos aprendizajes, ahora es más detallado y profundo, conociendo cómo funcionan y como poder adaptarlos a distintos entornos.

Asimismo, al desarrollarse el programa usando JavaScript y la biblioteca Phaser, se ha tenido que familiarizarse con ambos de forma más extensa para la creación de la página con el juego en sí, y el manejo de la lógica del mismo. Este ha sido un proceso de estudio sobre todos estos elementos que ha sido en ocasiones complejo, pero ha aportado mucho personalmente.

También cabe destacar, que el campo de la Inteligencia Artificial no es un área en la que se haga un gran énfasis en el grado cursado; por ello, con la realización de este proyecto se ha obtenido una experiencia fundamental en áreas que aportan un gran valor a la formación completa del alumno.

6.2 Trabajos futuros

Una vez contamos con todo el trabajo hecho hasta el momento, debemos mirar a las posibles adiciones y desarrollos extra que se le pueden añadir que también ayuden a intentar conseguir los objetivos con los que comenzamos el proyecto.

Una adición sería la introducción de entrenamiento distribuido en el aprendizaje. Esto resulta lógico, mediante el uso de una página online, podemos repartir el entrenamiento del algoritmo entre distintos puntos usando Internet, obteniendo la información de distintos lugares separados en momentos distintos.

Por supuesto, podemos añadir también nuevos algoritmos: Q-learning, double Q-learning, expected SARSA, deep Q-learning o actor-critic entre otros. También se podrían crear nuevos juegos o diferentes versiones de los que ya tenemos (similar a la versión alterna del Pong que ya tenemos). Asimismo, sería posible usar alguna biblioteca de JavaScript para incluir más métodos de aprendizaje, como tensorflow.js una biblioteca con la que por ejemplo podríamos incluir redes neuronales.

Por otra parte, utilizar WebAssembly en combinación con JavaScript nos serviría para mejorar la eficiencia de los programas y así ejecutar los algoritmos con mejor rendimiento en una página web, ya que en ocasiones JavaScript por sí solo puede resultar algo más lento.

En definitiva, con el trabajo realizado hemos conseguido demostrar la viabilidad de introducir un aprendizaje por refuerzo en un entorno web, y la posibilidad de expandirlo a distintos juegos e implementar sus algoritmos de diferentes formas. Las posibilidades futuras del mismo expandirían y refinarían su funcionamiento para hacer que se consigan los objetivos de manera más extendida y eficiente

Referencias

- [1] McCorduck, Pamela (2004). *Machines who think* (2ª edición). Natick Massachussets: A.K. Peters, Ltd.
- [2] Russell, Stuart J. y Norvig, Peter (2009). *Artifical Intelligence: a modern approach* (3ª edición). Upper Saddle River, New Jersey: Prentice Hall
- [3] IBM (2011). *Deep Blue*. Nueva York. Recuperado de: <https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/> (último acceso: 12/12/2022)
- [4] Luger, George y Subblefield William (2004). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving* (5ª.edición). Benjamin/Cummings.
- [5] Sutton, Richard S. y Barto, Andrew G (2018). *Reinforcement Learning: An Introduction* (2ª edición). MIT Press.
- [6] Goh, H. A., Ho, C. K., & Abas, F. S. (2022). Front-end deep learning web apps development and deployment: a review. *Applied Intelligence*, 1-23.
- [7] Página oficial de Scrum: <https://www.scrum.org/> (último acceso: 12/12/2022)
- [8] Newquist, HP (1994). *The Brain Makers: Genius, Ego and Greed in the Quest For Machines That Think*. Nueva York: Macmillan/SAMS.
- [9] Google Deepmind (2016). *AlphaGo: Mastering the ancient game of Go with Machine Learning*. Recuperado de: <https://ai.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html> (último acceso: 13/12/2022).
- [10] Deepmind (2019). *AlphaStar: Mastering the real-time strategy game StarCraft II*. Recuperado de: <https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii> (último acceso: 13/12/2022)
- [11] OpenAI (2016). *OpenAI Five*. Recuperado de: <https://openai.com/five/> (último acceso: 13/12/2022).
- [12] DataScience (2020). *Q-Learning*. Recuperado de: <https://datascience.eu/machine-learning/q-learning/> (último acceso: 13/12/2022)
- [13] GeeksForGeeks (2021). *SARSA Reinforcement Learning*. Recuperado de: <https://www.geeksforgeeks.org/sarsa-reinforcement-learning/> (último acceso: 13/12/2022).

- [14] MDN Web Docs. *JavaScript*. Recuperado de: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (último acceso: 13/12/2022).
- [15] Photon Storm. *Phaser3*. Recuperado de: <https://phaser.io/phaser3> (último acceso 13/12/2022).
- [16] MDN Web Docs. *HTML*. Recuperado de: <https://developer.mozilla.org/es/docs/Web/HTML> (último acceso: 13/12/2022).
- [17] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *Openai gym*. arXiv preprint arXiv:
- [18] Mitchell, T. (1997) *Machine Learning*. Nueva York: McGraw Hill.
- [19] <https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/>
- [20] <https://zitaoshen.rbind.io/project/rl/1-min-of-reinforcement-learning-q-learning/>
- [21] <https://medium.com/swlh/introduction-to-reinforcement-learning-coding-sarsa-part-4-2d64d6e37617>
- [22] <https://es.wikipedia.org/wiki/Pong>
- [23] <https://mathworld.wolfram.com/Tic-Tac-Toe.html>

Apéndice A

Manual de Uso

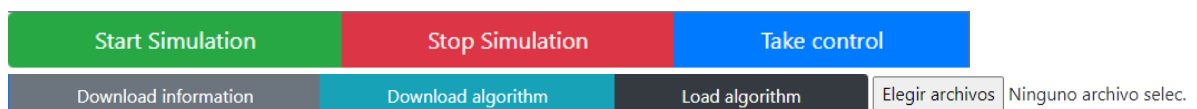
Este proyecto se trata de una aplicación web por lo que tiene que ser desplegado en un servidor. Esto se puede hacer utilizando un servidor remoto mediante algún tipo de servicio, o se puede ejecutar en una máquina local. En este manual consideramos la segunda opción.

Existen diferentes formas de desplegar una aplicación web localmente, en nuestro caso usaremos python en la consola del ordenador. Para ello existen diversos comandos según la versión de python que se tenga; para todas las versiones superiores a 3.0 se usan “`python3 -m http.server`” o “`py -m http.server`” en el directorio donde se encuentre el archivo html principal del programa.

De esta forma la aplicación se desplegará en nuestro ordenador en el puerto 8000, para ello en nuestro navegador podemos poner la dirección `localhost:8000` y nos encontraremos con la aplicación.

El tipo de aplicación que aparecerá dependerá de unos parámetros del código fuente. Primeramente, el que se ejecute Pong, Pong con algoritmos compartidos, la variación del Pong o el tres en raya, dependerá de qué tipo de juego se importe en el archivo `index.js`. En las primeras líneas de este archivo se importará 1 de 4 posibles archivos; `game.js` representa el Pong normal, `gameshared.js` el Pong con algoritmo compartido, `game2.js` la variación del Pong y `game3.js` el tres en raya. Así importando uno u otro elegimos el juego que se desplegará.

Si queremos que en el juego se produzca un crecimiento gradual hasta 600 por 600; existe un parámetro en `game.js` y `gameshared.js` llamado `grow` que define si se producirá durante su ejecución el aumento del tamaño. Además, si queremos ajustar el tamaño inicial de cualquier juego, tenemos que ajustar los valores `width` y `height` en `index.js` al valor que queramos en píxeles.



Dentro de cualquier versión de la aplicación contaremos con los siguientes botones; los cuales en resumidas cuentas actúan de la siguiente manera:

- **Start simulation:** acelerar el juego en gran medida.
- **Stop simulation:** parar una simulación.
- **Take control:** se da el control del jugador 1 a las flechas
- **Download information:** descarga un archivo de texto con las medias de las recompensas cada 100 episodios.

- **Download algorithm:** descarga un archivo de texto con el algoritmo actual de la pala jugador 1.
- **Load algorithm:** introducir un archivo de texto con formato de algoritmo y cargarlo a la pala jugador 1.

Por último, durante una partida, la consola del navegador reflejará una actualización después de cada episodio de la recompensa percibida por cada agente de la siguiente manera:

END episode = 93	reward = 0
END episode = 94	reward = 50
END episode = 95	reward = -100
END episode = 96	reward = -100
END episode = 97	reward = 50
END episode = 98	reward = -100
END episode = 99	reward = -100
END episode = 100	reward = -100

De esta forma podemos comprobar el progreso del agente y el resultado de cada episodio.



UNIVERSIDAD
DE MÁLAGA

| **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga