# APLICACIÓN DE LA INTELIGENCIA ARTIFICIAL EN VIDEOJUEGOS Y DESARROLLO DE UN AGENTE BASADO EN DEEP REINFORCEMENT LEARNING

Marcos Esteve Casademunt
David Gimeno Gómez

## Introducción

La industria de los videojuegos ha ido tomado un papel de gran relevancia en la economía mundial a medida que se han ido produciendo avances tecnológicos, proporcionando el auge de este sector. Es por ello por lo que tradicionalmente, esta industria se ha centrado en mejorar aspectos como los motores gráficos o el modelado 3D entre otros. No obstante, desde hace tiempo la Inteligencia Artificial en el ámbito de los videojuegos es un área de investigación abierta, donde el objetivo principal se basa en la idea de que un sistema informático pueda jugar tal y como lo haría un ser humano. Por otro lado, los videojuegos proporcionan un gran abanico de escenarios y desafíos interesantes para la investigación de la inteligencia computacional, permitiendo extrapolar las técnicas desarrolladas a otros dominios.

El objetivo principal de esta memoria consiste en realizar una revisión sobre la aplicación de técnicas de Inteligencia Artificial sobre la industria de los videojuegos, así como el desarrollo de un agente inteligente a través de la herramienta *OpenAI*. Respecto a la revisión cabe destacar que con el objetivo de encapsular el mayor número de desafíos presentes en el ámbito, se ha focalizado sobre la rama relacionada con los juegos de Estrategia en Tiempo Real (ETR), aunque gran parte de estos desafíos pueden ser compartidos, en ciertos momentos, con otros tipos de videojuegos.

En cuanto la estructura, comenzaremos con la revisión de las diferentes aplicaciones que ha aportado la Inteligencia Artificial en los juegos ETR [1], describiendo los distintos desafíos y las principales propuestas de solución. De esta manera, daremos pie al desarrollo de un agente mediante la herramienta *OpenAI Gym* [3]. No obstante, antes de abordar este apartado, se comentarán los avances realizados en videojuegos como AlphaStar, desarrollado por *DeepMind* [4]; o el caso de *Google Football* [5]. Por último, expondremos una serie de conclusiones finales respecto al proyecto realizado.

## Revisión de la Aplicación de Inteligencia Computacional en juegos ETR

Los juegos de Estrategia en Tiempo Real (ETR) es un género de videojuegos donde se requiere, por norma general, la gestión de distintas unidades de recursos en tiempo real, así como la coordinación y cooperación con los componentes del equipo. Normalmente, podemos encontrar una gran multitud de jugadores (humanos o no) que basan sus decisiones en la información incompleta que perciben en su entorno, la cual varía durante el transcurso del juego. Todas estas características, como se ha mencionado anteriormente, hacen de los juegos ETR un escenario perfecto para la investigación de la inteligencia computacional.

## Desafíos presentes en juegos ERT

Los juegos ERT proporcionan, como se ha mencionado anteriormente, un gran abanico de desafíos de gran interés en la investigación de la inteligencia computacional [6]. Entre ellos podemos distinguir los siguientes:

- **Planificación en tiempo real:** el jugador debe tomar sus decisiones bajo restricciones de tiempo severas, así como ser capaz de gestionar múltiples acciones simultáneamente. Estas decisiones abarcan un amplio abanico desde la gestión de recursos hasta las estrategias de batalla. Por otro lado, el entorno donde transcurre el juego se encuentra en continuo cambio debido a diversos factores, como pueden ser las interacciones del resto de jugadores. Además, se trata de un entorno parcialmente observable por lo que las decisiones deberán tomarse bajo el marco de la incerteza.
- **Modelado del oponente:** en ocasiones resulta interesante analizar el comportamiento del enemigo y predecir, en base a estas observaciones, sus futuras acciones o puntos flacos. Una aproximación basada en la minería de datos es presentada en el artículo [22] donde técnicas de *machine learning* son aplicadas sobre *logs* de juegos guardados.
- ***Pathfinding*:** hace referencia a la tarea de encontrar la ruta más adecuada para llegar a un punto de destino en el mapa. Dado que el entorno es dinámico y pueden aparecer multitud de factores a tener en cuenta, la tarea se complica. No obstante, resultados de gran naturalidad fueron conseguidos por los autores del artículo [18].
- **Colaboración:** Usualmente los jugadores se dividen en distintos equipos con los que deben cooperar y colaborar para alcanzar la victoria. Por ejemplo, en una batalla decidir a qué oponente atacar teniendo en cuenta lo que están haciendo sus compañeros. En definitiva, será necesario tratar con técnicas relacionadas con Sistemas Multi-Agente.
- **Generación de contenido:** con este desafío se pretende construir modelos capaces de generar, con suficiente variedad, todo contenido relacionado con el juego como pudiera ser el mapa. Una gran multitud de técnicas se han aplicado sobre este problema, siendo ampliamente descritas en el artículo [11].

Por último, consideramos necesario indicar que resolver este tipo de desafíos puede proporcionar grandes avances en otros ámbitos de la inteligencia computacional, tal y como se ha mencionado en el apartado anterior.

## Técnicas empleadas en juegos ERT

Previamente hemos descrito los desafíos relacionados con la inteligencia computacional que plantean los juegos ERT. Este apartado se centra en comentar las distintas soluciones que se han planteado en la literatura para resolver estos desafíos. Entre ellas destacamos las siguientes:

- **Planificadores:** la estrategia que debe adoptar el NPC puede interpretarse básicamente como un plan. De ahí que hayan surgido números artículos, como es el caso de [15, 16, 17], donde se emplean planificadores conocidos como por ejemplo, PDDL. Algunos de estos artículos no tratan una estrategia global, sino que se centran en niveles más concretos como puede ser la construcción de estructuras para mejorar la defensa ante un ataque o la gestión de recursos. Teniendo en cuenta la limitación de que, en todo momento, el agente se encuentra en un entorno parcialmente observable.
- **Algoritmos Evolutivos:** estos métodos han sido ampliamente utilizadas en el ámbito y, a menudo, se combinan con otras técnicas para mejorar la calidad de sus resultados. Como es el

caso de los artículos [7, 8] cuyo propósito era proporcionar estrategias de juego automáticas a los NPC. Por otro lado, [9, 14] compartían el mismo objetivo pero introdujeron mejoras en el tiempo de respuesta de estos algoritmos evolutivos. Respecto a la generación de contenido, los autores de [10] desarrollaron un algoritmo genético multiobjetivo capaz de producir el mapa del videojuego *Starcraft,* tanto de forma automática como para proporcionar un apoyo a la generación asistida. Otros artículos basados en la misma filosofía son [12, 13].

- **Esquemas Multi-Agente:** la cooperación y coordinación son puntos clave para los NPC que pertenezcan a un mismo equipo. Estos desafíos requieren el uso de técnicas relacionadas con los Sistemas Multi-Agente que les permitan una cierta comunicación entre los NPC. Varios artículos como [23, 24] se apoyan en estos esquemas.
- ***Reinforcement Learning*:** este tipo de técnicas normalmente se emplean para la selección del plan o estrategia del NPC. A grandes rasgos, el agente debe refinar una política que determina cuáles son las acciones más adecuadas en función del estado en el que se encuentre [28]. Como en muchos otras ocasiones, en los últimos años esta filosofía ha gravitado hacia el ámbito *Deep Learning*. Una revisión completa al respecto se refleja en la publicación [2] de la bibliografía.

  No obstante, debido a que el agente desarrollado mediante *OpenAI Gym* en nuestro proyecto se fundamenta en este tipo de técnicas, se ha optado por detallar ciertos aspectos de esta filosofía en el apartado correspondiente. Por otro lado, numerosos avances en estas técnicas también son mencionados en los casos de uso descritos posteriormente, donde se se comentan arquitecturas novedosas como son DQN [25] e IMPALA [26], descritas en mayor detalle en el artículo [2].

- **Redes Neuronales:** uno de los principales modelos fue rtNEAT presentado en [19] y posteriormente aplicado en [20, 21]. El objetivo era dotar a los NPC del videojuego con la capacidad de adaptarse al jugador a medida que avanza el juego y prescindir de comportamientos predefinidos. Todo esto mediante el empleo de arquitecturas profundas. Por otro lado, esta filosofía tiene gran presencia en el ámbito del *Reinforcement Learning.*

En definitiva, los juegos ERT proporcionan un escenario perfecto, así como un abanico de desafíos, donde poder desarrollar técnicas de gran utilidad que puedan extrapolarse posteriormente a otros ámbitos fuera de la industria de los videojuegos. Si se desea indagar más en el tema, se puede encontrar una revisión más exhaustiva en el artículo [1], el cual ha sido, en este apartado, la principal referencia.

## Casos de Uso

### *AlphaStar: Multiagent Reinforcement Learning*

*AlphaStar* [4] es un sistema desarrollado por *DeepMind* basado en *Multi-agent Reinforcement Learning*, un ámbito que se encuentra en auge debido a numerosos avances [27]. Se trata de la primera IA en aprender, con un nivel de considerable calidad y sin ningún tipo de restricción, juegos de gran complejidad como pueden ser los ERT, ya descritos anteriormente. Más concretamente, este *software* fue probado sobre el popular juego *StarCraft II*, llegando a desafiar a los mejores jugadores, venciendolos en numerosas ocasiones.

El sistema comprende diversas técnicas de *machine learning* entre las que destacamos: Redes Neuronales, *Reinforcement Learning*, *Multi-Agent Learning*, así como Aprendizaje por Imitación. Todas ellas capaces de aprender directamente de la información generada en el juego. Por otro lado, en combinación, se empleó una técnica conocida como *Self-play Learning* donde el agente debe enfrentarse con distintas

versiones de sí mismo para ir aprendiendo progresivamente aunque esta filosofía presenta numerosos inconvenientes, tal y como comentan los autores [4]. El más destacado se conoce como *forgetting*, es decir, el agente que juega contra sí mismo irá mejorando a medida que juega pero puede llegar a olvidar cómo vencer a anteriores versiones del oponente. Esto, además, conduce el aprendizaje del agente a través de un ciclo sin que logre generalizar. Una forma de resolver este problema fue enfrentar al agente contra una mixtura de versiones distintas de sí mismo, lo que se denominó *Fictitious Self-playing.*

Sin embargo, el *Fictitious Self-play* es insuficiente si se desean resultados considerables en un ERT, ya que es necesario integrar la interacción o cooperación entre los distintos agentes que forman un equipo. A grandes rasgos, entre un mismo equipo los agentes pueden compartir conocimiento de forma que todos vayan aprendiendo y mejorando conjuntamente. Esto es posible a la introducción de una especie de agentes encargados de identificar los puntos flacos del agente principal, conocidos como *exploradores*.. Todo esto claro, bajo el paradigma del *Reinforcement Learning* de modo que interactuando con el entorno y a base de prueba y error los agentes irán refinando sus parámetros logrando un aprendizaje robusto. No obstante, en un entorno con las características de un ERT, el agente debe tomar decisiones entre un gran abanico de posibilidades; estamos hablando del alrededor de $10^{26}$ posibles acciones en un instante de tiempo. Encontrar una estrategia ganadora es una tarea compleja en un espacio de búsqueda como tal. Es por ello necesario introducir cierto conocimiento experto. Aquí entran en juego las técnicas Imitativas con el objetivo de replicar ciertas estrategias que pueda observar en otros agentes, que en combinación con redes neuronales logramos establecer una política inicial de gran calidad. Para una inspección en detalle se referencia el artículo [4].

De este modo, tal y como se ha avanzado previamente, se obtuvieron grandes resultados en *StarCraft II* pero se mantiene la idea de aplicar este desarrollo sobre otros dominios y ámbitos del mundo real, tras los avances que ha supuesto la investigación.

### *Google Football*

Como hemos podido observar a lo largo de la memoria, el *Reinforcement Learning* trata de entrenar agentes con el propósito de que éstos sean capaces de interactuar con su entorno y aprender de sus experiencias para resolver tareas de gran complejidad. Aunque el propósito principal sería su aplicación en el mundo real, hemos visto como se ha fomentado el empleo de estas técnicas en escenarios seguros como pueden ser los videojuegos. En este apartado centramos nuestra atención en el juego del fútbol, donde podemos encontrar desafíos al tener que realizar un balance entre un control de corto alcance, aprendizaje de conceptos (como puede ser el regate o pasar el balón) y una estrategia de alto nivel con el resto del equipo, la cual en muchas ocasiones dependerá de las acciones que tome el oponente.

*Google Football* [5] consiste en un entorno de *Reinforcement Learning* donde se proporciona una simulación de fútbol en 3D con todas las físicas necesarias. En ella, los agentes controlan a uno o todos los jugadores de fútbol de su equipo, aprenden a pasar el balón entre ellos y logran superar la defensa rival para marcar los goles que les llevarán a la victoria; tal y como podemos observar en la Figura 1.


**Figura 1.** Distintos escenarios donde se desarrolla *Google Football*

Todo esto es posible gracias al funcionamiento conjunto de los principales componentes en los que se divide *Google Research Football Environment*. Entre ellos destacamos los siguientes:

- ***Football Engine*:** es el núcleo principal de la sistema, proporcionando una simulación de alta optimización escrita en código C++ que permite su ejecución tanto si se posee GPU como si no. A partir de las acciones tomadas por los agentes de ambos equipos, logra simular un partido de fútbol incluyendo goles, faltas, penaltis o saques de esquina. Respecto a las características orientadas al *Reinforcement Learning* que este motor proporciona, destacamos las técnicas que permiten representar cada uno de los estados de forma que los agentes puedan aprender de éstos. En ellos podemos encontrar la posición de cada uno de los jugadores o información semántica. No obstante, también se trata el aprendizaje a partir de los píxeles en crudo, es decir, sin ningún tipo de proceso. Por otro lado, el desarrollador puede decidir si desea un entorno estocástico, donde se aplique aleatoriedad en múltiples aspectos; o, por el contrario, un entorno determinista. Por último, presenta compatibilidad con la herramienta *OpenAI Gym*, la cual es empleada en este proyecto.
- ***Football Benchmarks:*** el sistema proporciona un conjunto de problemas de referencia, también conocidos como *benchmarks*. El principal objetivo de estos problemas es que el agente aprenda a jugar un partido de fútbol contra un oponente basado en reglas, el cual se encuentra disponibles en tres modos de dificultad. Para ello, se emplearon dos algoritmos del estado del arte en *Deep Reinforcement Learning*: DQN [25] e IMPALA [26], cuyo *reward* se centraba en marcar goles o, al menos, generar una jugada de gran peligro. Tal y como muestra la Figura 2, el DQN entrenada en 20 millones de *steps* es capaz de vencer en el modo *easy*, mientras que si deseamos vencer al resto de niveles debemos emplear el algoritmo IMPALA con un entrenamiento mucho mayor.

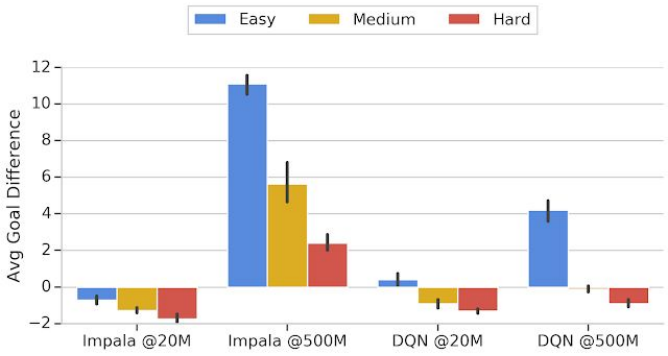


**Figura 2.** Resultados de los algoritmos DQN e IMPALA en el *benchmark* planteado en *Google Football*

- ***Football Academy*:** aprender a jugar un partido completo puede resultar muy complicado. Por ello, se proporcionan una serie de escenarios con desafíos diversos de dificultad variante. Esto permite a los desarrolladores aplicar técnicas de *curriculum learning* donde planteamos a nuestro agente, de forma gradual, distintos desafíos hasta que es capaz de enfrentarse a un partido completo. Cada uno de estos escenarios se centra en aspectos como los saques de esquina, pases o penaltis. Además, mediante una sencilla API, se permite desarrollar escenarios propios enfocados a las pruebas que deseemos llevar a cabo.

Un último aspecto a destacar es que el desarrollador puede probar su proyecto enfrentando su agente frente a otros o jugando él mismo contra su creación. Por otro lado, si se desea una inspección más exhaustiva del sistema desarrollado por Google, puede inspeccionarse el artículo [5] además de que toda la implementación queda plasmada en su *Github*[1] como código libre.

## Desarrollo de un agente mediante la herramienta *OpenAI Gym*

El objetivo de este apartado consiste en comentar la herramienta proporcionada por *OpenAI* para el entrenamiento de sistemas de *Reinforcement Learning*. Esta herramienta permite obtener acceso a una abstracción de alto nivel para la construcción de dichos sistemas, donde, como podemos observar en la figura Figura 3, un sistema inteligente, desde ahora llamado agente, realiza una acción $A_t$ en un entorno específico, como puede ser un videojuego o una simulación. Una vez realizada la acción el entorno cambia y por tanto el agente observa el nuevo estado $s_t$ y recoge una recompensa $r_t$.

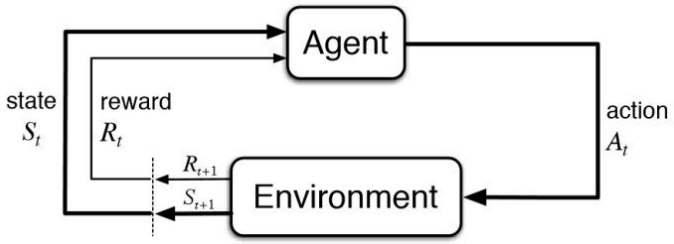


**Figura 3.** Esquema General de un sistema *Reinforcement Learning*

[1] https://github.com/google-research/football

El objetivo se reduce por tanto a especificar un entorno y definir el algoritmo que se encargue de ir aprendiendo la política de dicho entorno, es decir, cuales son las acciones que, dado un estado determinado, le lleven a conseguir el objetivo perseguido, por ejemplo maximizar la recompensa obtenida.

Con el objetivo de evaluar el *software* desarrollado, se ha optado por emplear el juego *Car-Mountain-v0* propuesto por la herramienta *OpenAI* entre otras alternativas. Tal y como sugiere la Figura 4, se pretende que el agente aprenda la forma de alcanzar la meta situada en la cima de la montaña. Para ello dispone de tres acciones: moverse, tanto a la izquierda como derecha, con una cierta velocidad o no realizar ningún acto. Con la ayuda de estas decisiones y los impulsos que pueda tomar gracias a la gravedad implementada en la simulación, el agente refinará su política en base a sus experiencias para conseguir cumplir el objetivo planteado.

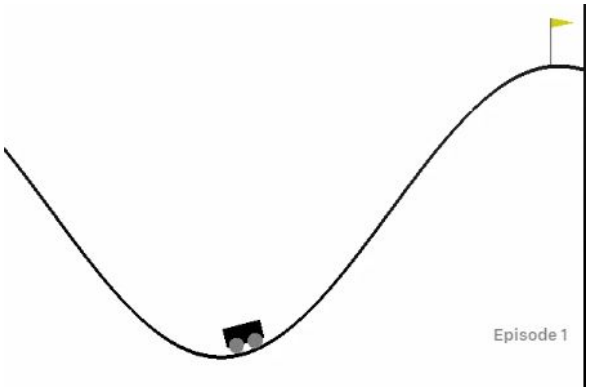


**Figura 4.** Desafío *Car-Mountain-v0* propuesto en la herramienta *OpenAI*

Para ello se va a evaluar dos técnicas de *Reinforcement Learning*. Por una parte, el *Q-Learning* clásico, también conocido como la ecuación de Bellman; y por otra parte técnicas más avanzadas y basadas en redes neuronales con Deep-Q-learning.

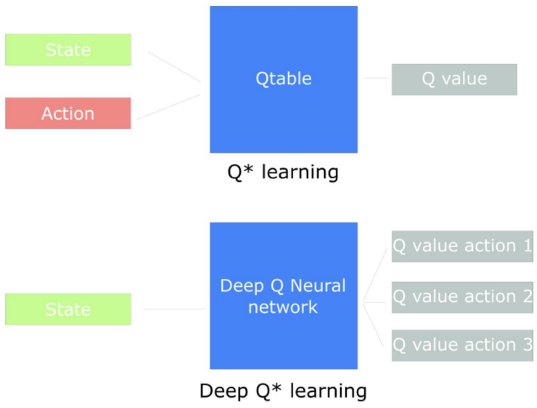


**Figura 5.** Comparación de *Q* learning* clasico con *Deep Q* learning*

Tal y como puede observarse en la Figura 5, la principal diferencia entre las dos técnicas radica en que mientras el Algoritmo *Q * Learning* recibe como entradas el estado y la acción tomada por el agente y devuelve su valor *Q value,* también conocida como la bonanza de esa acción, el *Deep Q Learning* recibe

un estado como entrada y emplea una red neuronal para predecir los *Q values* de todas las posibles acciones. En los sucesivos puntos comentaremos en mayor detalle cada una de las implementaciones realizadas.

### *Q-Learning* Clásico

Para el desarrollo del primer experimento se ha optado por realizar una implementación en Python del algoritmo, comentado en pseudocódigo en la Figura 6. Este algoritmo consiste en ir aprendiendo una tabla Q donde para cada posible estado se almacenará cual es la puntuación de cada posible acción. Finalmente, tal y como se puede observar en el pseudocódigo, el objetivo de dicho algoritmo es maximizar la puntuación obtenida.

```
Output: Q
  initialize Q arbitrarily, e.g., to 0
  Repeat
    select s as an initial state
    while(state s is not terminal) do
      a = action for s derived by Q (epsilon-greedy)
      take action a, observe r, sn
      Q(s,a)=Q(s,a)+alfa*[r+gamma*max_an(Q(sn, an))-Q(s,a)]
      s=sn
    end
  Until convergence
end
```

**Figura 6.** Pseudocódigo basado en la Ecuación de Bellman para técnicas de *Reinforcement Learning*

Nuestro caso de estudio tiene un conjunto de 2 posibles estados determinados por la posición y la velocidad del vehículo en cada instante de tiempo. Por otra parte se disponen de tres posibles acciones, no hacer nada, moverse a la izquierda y moverse a la derecha. Discretizando los posibles valores de posición y velocidad que puede tomar nuestro agente, se tiene una tabla de 19 observaciones de posición * 15 observaciones de velocidad * 3 posibles acciones. Por último, el algoritmo deberá de aprender para cada una de las posiciones de la tabla comentada, su *Q value* realizando una experimentación fundamentada en la prueba y error. La implementación de este algoritmo queda reflejada en el *notebook* adjunto en la entrega de esta memoria.

### *Deep Q-Learning*

En los últimos años, esta filosofía ha gravitado hacia el ámbito del *Deep Learning*, sustituyendo la tabla Q*, anteriormente empleada en las técnicas clásicas, por una red neuronal capaz de predecir los *Q values* de cada posible acción en el dominio. Dependiendo, en cada momento, del estado en el que se encuentre el agente, tal y como se refleja en la Figura 5. Por ello, guiados por la documentación de *OpenAI*, se ha optado por implementar un sistema *Deep Q-Learning*, donde identificamos dos etapas principalmente:

- En primer lugar, el agente interactúa con el entorno decidiendo en cada momento qué acción tomar en función de las predicciones proporcionadas en la red. Mientras tanto, se almacenan todas estas experiencias en lo que denominamos *memoria*. Esta memoria contiene el estado donde nos encontramos, la acción tomada, la recompensa recibida tras esa acción y el nuevo estado, consecuencia de las decisiones del agente. Además se almacena si esa "jugada" lleva a un estado objetivo.

- La segunda etapa, consiste en aprovechar esta *memoria* o experiencias de modo que se refinen los parámetros de nuestra red, la cual puede interpretarse como la política comentada anteriormente. En definitiva, se trata de entrenar la red en base a las experiencias vividas por el agente. Este aprendizaje se lleva a cabo a través de una función *loss* similar a la ecuación de Bellman y mediante una optimización por el error de mínimos cuadrados.

Estas dos etapas se ejecutan durante un número predefinido de *epochs* o episodios de forma que vaya perfeccionándose el aprendizaje de nuestro agente. De igual manera, toda la implementación puede inspeccionarse en el *notebook* adjunto en la entrega de la memoria.

## Conclusiones y trabajo futuro

A modo de conclusiones se podría destacar que en este trabajo se ha realizado una exploración exhaustiva de las distintas tecnologías empleadas para problemas de *Reinforcement Learning* en videojuegos. Además, se han implementado *Q learning* y *Deep Q learning* para la creación de un agente sencillo en el entorno de simulación de videojuegos empleando, para ello, la herramienta OpenAI.

Por otra parte, y a modo de reflexión, destacar que el uso de tecnologías relacionadas con el *Reinforcement Learning* está teniendo cada vez mayor presencia en diversas áreas como la conducción autónoma o la robótica. Esto es gracias al proceso de maduración que han sufrido estas tecnologías con el transcurso de los años. Como ya se ha comentado a lo largo de la memoria, los entornos proporcionados por los videojuegos suponen un perfecto escenario para el desarrollo y mejora de este tipo de técnicas antes de abordar ámbitos más ambiciosos como los mencionados previamente.

Por último,  y a modo de trabajo futuro, se podría plantear hacer uso de herramientas como Keras-rl[2], la cual permite obtener una abstracción de los algoritmos de Aprendizaje Reforzado Profundo y, por ejemplo, aplicarlos sobre un entorno como podría ser Google Football.

[2] https://github.com/keras-rl/keras-rl

## Bibliografía


[1] Lara-Cabrera, R., Cotta, C., & Fernández-Leiva, A. J. (2013, April). A review of computational intelligence in RTS games. In *2013 IEEE Symposium on Foundations of Computational Intelligence (FOCI)* (pp. 114-121). IEEE.

[2] Shao, K., Tang, Z., Zhu, Y., Li, N., & Zhao, D. (2019). A Survey of Deep Reinforcement Learning in Video Games. *arXiv preprint arXiv:1912.10944*.

[3] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*. https://gym.openai.com

[4] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., ... & Oh, J. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*(7782), 350-354. https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforcement-learning

[5] Kurach, K., Raichuk, A., Stańczyk, P., Zając, M., Bachem, O., Espeholt, L., ... & Gelly, S. (2019). Google research football: A novel reinforcement learning environment. *arXiv preprint arXiv:1907.11180*. https://ai.googleblog.com/2019/06/introducing-google-research-football.html

[6] Buro, M., & Furtak, T. M. (2004, May). RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)* (Vol. 6370).

[7] M. J. V. Ponsen, H. Muñoz-Avila, P. Spronck, and D. W. Aha, "Automatically generating game tactics through evolutionary learning", *AI Magazine*, vol. 27, no. 3, pp.75-84, 2006.

[8] C. Miles, S. Louis, N. Cole, and J. McDonnell, "Learning to play like a human: case injected genetic algorithms for strategic computer gaming", in *Congress on Evolutionary Computation*, vol. 2, 2004, pp. 1441-1448.

[9] C. Miles, S. J. Louis, "Case-injection improves response time for a real-time strategy game", in *IEEE Symposium on Computational Intelligence and Games*. IEEE, 2005.

[10] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, "Multiobjective exploration of the Starcraft map space", in *IEEE Conference on Computational Intelligence and Games*, G. N. Yannakakis and J. Togelius, Eds. IEEE, 2010, pp. 265-272.

[11] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172-186, 2011.

[12] "Evolution of artificial terrains for video games based on accessibility", in *Applications of Evolutionary Computation 2010*, ser. Lecture Notes in Computer Science, C. D. Chio *et* al., Eds., vol. 6024. Springer-Verlag, 2010, pp. 90-99.

[13] "Evolution of artificial terrains for video games based on obstacles edge length", in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1-8.

[14] S.-H Jang, J. Yoon, and S.-B. Cho. "Optimal strategy selection of non-player character on real time strategy game using a speciated evolutionary algorithm", in *IEEE Symposium on Computational Intelligence and Games*, P. L. Lanzi, Ed. IEEE, 2009, pp. 75-79.

[15] A. Kovarsky and M. Buro, "A First Look at Build-Order Optimization in Real-Time Strategy Games", in *GameOn Conference*, L. Wolf and M. Magnor, Eds. EUROSIS, 2006, pp.18-22.

[16] M. Grimani, "Wall Building for RTS Games", in *AI Game Programming Wisdom 2*. Hingham, Massachusetts: Charles River Media, Inc., 2004, pp. 425-437.

[17] V. Alcázar, D. Borrajo, and C. Linares López, "Modelling a RTS planning domain with cost

conversion and rewards", in *Artificial Intelligence in Games. Workshop of the Eighteenth European Conference on Artificial Intelligence,* Patras, Greece, 2008, pp. 50-54.

[18] H. Danielsiek, R. Stür, A. Thom, N. Beume, B. Naujoks, and M. Preuss, "Intelligent moving of groups in real-time strategy games", in *IEEE Symposium on Computational Intelligence and Games*, P. Hingston and L. Barone, Eds. IEEE, 2008, pp. 71-78.

[19] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game", *IEEE Transactions on Evolutionary Computation,* vol. 9, no. 6, pp. 653-668, 2005.

[20] J. M. Traish and J. R. Tulip, "Towards adaptive online RTS AI with NEAT", in *Computational Intelligence and Games*. IEEE, 2012, pp. 430-437.

[21] J. K. Olesen, G. N. Yannakakis, and J. Hallam, "Real-time challenge balance in an RTS game using rtNEAT", in *IEEE Symposium on Computational Intelligence and Games,* P. Hingston and L. Barone, Eds. IEEE, 2008, pp. 87-94.

[22] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction", in *IEEE Symposium on Computational Intelligence and Games*, P. L. Lanzi, Ed. IEEE, 2009, pp. 140-147.

[23] "Using multi-agent potential fields in real-time strategy games", in *Autonomous Agents and Multiagent Systems*, L. Padgham *et* al., Eds, IFAAMAS, 2008, pp. 631-638.

[24] "A multi-agent potential field-based bot for a full RTS game scenario", in *Artificial Intelligence and Interactive Digital Entertainment Conference*, C. Darken and G. M. Youngblood, Eds. The AAAI Press, 2009.

[25] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529-533.

[26] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., ... & Legg, S. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*.

[27] Zhang, K., Yang, Z., & Başar, T. (2019). Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. *arXiv preprint arXiv:1911.10635*.

[28] Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning (Vol. 135). Cambridge: MIT press.