UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

INGENIERÍA DEL SOFTWARE

Aprendizaje por refuerzo aplicado a videojuegos de gestión por recursos

Deep learning applied to resource management videogames

Realizado por
Jesús Ariza Pomares

Tutorizado por
Eduardo Guzmán de los Riscos

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, junio de 2025

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

Aprendizaje por refuerzo aplicado a videojuegos de gestión por recursos

Deep learning applied to resource management videogames

Realizado por
Jesús Ariza Pomares

Tutorizado por
Eduardo Guzmán de los Riscos

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2025

Fecha defensa: julio de 2025

Resumen

El objetivo de este proyecto es analizar, estudiar y aprender sobre el aprendizaje por refuerzo, aplicándolo a un videojuego de gestión por recursos. El videojuego simula una granja donde se deben gestionar cultivos, árboles y un sistema de energía que servirá como núcleo mecánico para todas las interacciones del videojuego. Estos elementos se ven afectados por ciclos diarios y estacionales, añadiendo capas de complejidad estratégica. El modelo de aprendizaje por refuerzo se evaluará según su rendimiento en este entorno dinámico, donde deberá adaptarse a mapas generados aleatoriamente mientras aprende todas estas mecánicas y sus sinergias.

Este proyecto abordará desde el desarrollo principal del videojuego hasta las pruebas realizadas con el mismo para analizar y evaluar el trabajo realizado en materia de aprendizaje por refuerzo, pasando por la descripción del propio juego, sus mecánicas y las elecciones tomadas para su estudio.

Palabras clave: aprendizaje por refuerzo, gestión de recursos, videojuegos, Q-learning

Abstract

The objective of this project is to analyze, study and learn about reinforcement learning, applying it to a resource management videogame. The game simulates a farm where one must manage crops, trees and an energy system that serves as the mechanical core for all interactions in the videogame. These elements are affected by daily and seasonal cycles, adding layers of strategic complexity. The reinforcement learning model will be evaluated based on its performance in this dynamic environment, where it must adapt to randomly generated maps while learning all these mechanics and their synergies.

This project will address from the main development of the videogame to the tests

carried out to analyze and evaluate the work done on reinforcement learning, through the description of the game itself, its mechanics and the choices made for its study.

Keywords: reinforcement learning, resource management, videogames, Q-learning

Índice

1 Introducción

Todos nos hemos enfrentado alguna vez a un reto a superar, una tarea que realizar o un objetivo que alcanzar. Tanto en nuestra vida privada como en la rutina del día a día, nos encontramos con infinidad de situaciones donde debemos tomar una serie de elecciones, conscientes o inconscientes.

Para todas estas acciones, tanto sencillas como complejas, siempre hemos tenido una primera toma de contacto donde no sabíamos muy bien como actuar o proceder pero, mediante repetición, costumbre y aprendizaje, conseguimos optimizar y mejorar la forma en la que realizamos estas acciones.

Esta optimización la conseguimos gracias a poder reconocer los errores que cometemos y mejorar en consecuencia, analizando las acciones tomadas y el resultado obtenido según el escenario en el que nos encontremos.

Este proceso mental, compuesto por el ensayo, error y una adaptación continua son también parte fundamental del aprendizaje por refuerzo. Al igual que nosotros somos capaces de razonar y actuar en consecuencia tras evaluar los resultados de nuestras decisiones, un agente de aprendizaje por refuerzo es capaz de actuar con un patrón similar. Dado un entorno, unas acciones y una balanza de recompensas y penalizaciones, cualquier agente de aprendizaje por refuerzo puede acabar mejorando su toma de decisiones, optimizando las acciones que realiza.

En este proyecto se investigará este proceso de aprendizaje para analizar y estudiar sus capacidades en un entorno cerrado (en este caso un videojuego de gestión por recursos) observando a su vez el progreso del agente y los resultados que este obtenga.

1.1. Motivación

La motivación de este trabajo viene de la curiosidad que nace de proyectos similares donde se documenta el avance y el crecimiento de agentes, en los que se puede observar, casi por arte

de magia, cómo una máquina es capaz de aprender y optimizar su comportamiento.

Como desarrollador de videojuegos, no hay nada comparable a la sensación de ver a nuevos jugadores aprendiendo a jugar, probar mecánicas y abrirse paso entre los retos que ofrece tu propio videojuego. Es también por esta razón que el aprendizaje por refuerzo resulta tan atractivo, ya que al final del día son dos escenarios muy similares, donde en ambos casos tomas el rol de guía (con indicaciones, recompensas o penalizaciones) para ver cómo aprenden y avanzan.

Además, la diferentes formas en las que un agente intenta optimizar y se abre paso por los diferentes entornos resultan tanto fascinantes como interesantes. Desde un punto de análisis de diseño es otro contexto totalmente diferente, poniendo al límite los recursos y el entorno en el que se encuentra. Este contexto podría replicarse por un humano, pero estos agentes no cuentan con el sesgo de evitar acciones que un principio puedan parecer "sin sentido" o "erróneas", que luego acaban siendo la mejor opción o descubriendo secretos de la situación en la que se encuentran.

A lo largo de la carrera aprendemos y descubrimos infinidad de temas relacionados tanto con la ingeniería informática como con la ingeniería del software, pero el campo del aprendizaje por refuerzo no es un tema que se pueda encontrar tan fácilmente en los contenidos ofrecidos.

Por todas estas razones se consideró un TFG puede representar un buen momento para adentrarse en este área de la inteligencia artificial.

1.2. Objetivos del proyecto

Los objetivos del proyecto se alinean con lo que motivó la realización de este proyecto además de la realización de un videojuego fácilmente analizable y manejable para estudiar el aprendizaje por refuerzo aplicado al mismo.

Objetivos:

- Desarrollo de un videojuego de gestión por recursos simple al que se pueda aplicar el aprendizaje por refuerzo.

- Estudiar y aprender sobre el aprendizaje por refuerzo genérico y sobre el aplicado a videojuegos.

- Analizar el comportamiento observado del agente en el videojuego, iterando y mejorando el sistema según sea necesario para lograr un agente lo más inteligente posible.

1.3. Organización del documento

La organización general del documento se puede agrupar en cuatro bloques diferenciados. El primero consta del estado del arte en materia del aprendizaje por refuerzo, donde este se analiza, desde un punto de vista general tanto concreto, mostrando en el camino proyectos similares.

El segundo bloque se centra en documentar el diseño y especificación del videojuego de gestión por recursos. En él se habla del proceso de creación del juego y el porqué de su diseño. Este bloque estará acompañando de comentarios que pondrán en contexto decisiones tomadas en el desarrollo del videojuego para permitir un mayor abanico de posibilidades a estudiar de cara a su interacción con los agentes de aprendizaje por refuerzo.

El tercer bloque incluirá todo el contenido del proyecto relacionado con el aprendizaje por refuerzo, desde los fundamentos teóricos hasta su aplicación sobre el videojuego. A lo largo de este bloque se comentan los avances realizados y los resultados conseguidos según que condiciones.

El último y cuarto bloque comenta las conclusiones del proyecto acompañadas de trabajos futuros a realizar sobre el proyecto según los objetivos cumplidos, las dificultades encontradas a lo largo del desarrollo y posibles ampliaciones donde se debaten diferentes caminos para diferentes rutas de trabajo.

2

Marco teórico

2.1. Aprendizaje por refuerzo en inteligencia artificial

La inteligencia artificial cuenta con una amplia gama de tipos y enfoques disciplinares. Entre todas estas categorías, se encuentra la base de este trabajo, el aprendizaje por refuerzo. El aprendizaje por refuerzo (Reinforcement Learning en inglés, RL) es un paradigma de aprendizaje automático en el que un agente interactúa con un entorno para maximizar una recompensa acumulativa [2].

A su vez, este paradigma se ubica en la categoría de aprendizaje no supervisado, diferenciándose del supervisado al no contar con información ni un conjunto de datos etiquetados para poder operar y funcionar, sino que aprende mediante la exploración y la retroalimentación dada por el entorno en el que se encuentra dicho agente. Este modelo de aprendizaje se asemeja a la psicología conductista, llegando a reproducir comportamientos y formas de actuar similares a las que se pueden observar del aprendizaje humano mediante el ensayo, error y recompensas [21].

Los fundamentos teóricos del RL llegan a remontarse a los trabajos de Richard Bellman entre los años 1950 y 1970 sobre la programación dinámica [8]. Durante estos años y a través de estos trabajos, introdujo el concepto de ecuación de optimalidad (o de Bellman), en él se profundizará más adelante en este documento.

Aún con la introducción de la ecuación de optimalidad, no fue hasta que se combinó con redes neuronales profundas y técnicas como el *experience replay* que el campo superó limitaciones clave en dimensionalidad y escalabilidad. Esta combinación de técnicas queda respaldada en el trabajo de Mnhin [14], donde demuestra como un agente DQN (*Deep Q Network*) aprendió directamente de los píxeles en videojuegos de la consola Atari, alcanzando al nivel humano sin soporte directo del mismo e incluso llegando al nivel de jugadores profesionales.

Actualmente, los algoritmos de RL se pueden clasificar en tres grandes familias:

- **Métodos basados en valor**: como el Q-Learning [26] y sus variantes profundas: el DQN, tal y como se menciona en el párrafo anterior, que superó las limitaciones de dimensionalidad, procesando píxeles directamente, o el Double DQN con el que se logra evitar el sesgo de sobreestimaciones que puede suceder con el DQN simple [24].

- **Métodos basados en políticas**: REINFORCE [27] o *Policy Gradient* [22], que aprenden directamente una política y que brillan en escenarios donde la acción sea continua. Versiones más recientes como PPO (*proximal policy optimization*) [18] introducen mecanismos para poder mejorar la estabilidad y la eficiencia del entrenamiento.

- **Métodos Actor-Crítico**: combinan los métodos del primer punto con un componente crítico y los métodos del segundo con un actor [12]. Con el componente crítico se calcula la función de valor mientras que el actor se encarga de aprender la política. Esta combinación permite reducir la varianza del gradiente y mejorar la estabilidad del entrenamiento. Sistemas avanzados como AlphaGo [19] demostraron su eficacia en entornos altamente complejos.

Un aspecto crucial en RL es el dilema exploración-explotación [11]. Los agentes deben encontrar el balance entre explorar nuevas acciones con la promesa de poder descubrir o encontrar mejores estrategias, o explotar acciones que ya conocen para intentar maximizar la recompensa. Justamente este equilibrio y el dilema que trae consigo está presente en todos los videojuegos donde se pueda obtener alguna puntuación y el orden de las mismas afecte a esta, dando así libertad a la exploración y explotación de mecánicas para optimizarla.

2.2. Fundamentos del aprendizaje por refuerzo

Como se menciona al principio del apartado anterior, el RL es un paradigma de aprendizaje automático donde un agente interactúa con un entorno desconocido, realizando acciones que modifican su estado y recibiendo recompensas o penalizaciones como retroalimentación [2]. Inspirado en la psicología conductista de Skinner [21], el RL se distingue por un enfoque de

aprendizaje mediante la prueba y el error, donde se busca maximizar la recompensa acumulada a lo largo de de las simulaciones y ejecuciones.

2.3. Componentes principales del aprendizaje por refuerzo

- Agente: se encarga de tomar las acciones, aprender y evolucionar al interactuar con el entorno en el que se encuentre.
- Entorno: el contexto en el que el agente participa. Algunos ejemplos de entorno podrían ser el mapa sobre el que se desarrollo este trabajo o incluso un juego de mesa clásico.
- Estado (s): representa un momento o situación en la que se encuentra el entorno. El agente usa este estado para la toma de decisiones.
- Acción (a): cada una de las decisiones que puede tomar el agente.
- Recompensa (r): la retroalimentación numérica (tanto positiva como negativa) que se da cuando el agente realiza una de las acciones de las que dispone al tomar una decisión.
- Política (π): la estrategia que el agente sigue para poder elegir las acciones en función del estado.
- Función de valor (V): la recompensa acumulada que se espera al seguir una política desde un estado concreto.
- Función de valor-acción (Q): la recompensa acumulada que se espera al realizar una acción en un estado y continuar con una política desde ese momento.

2.4. Procesos de Decisión de Markov (MDP)

Los Procesos de Decisión de Markov (MDP) constituyen el marco matemático fundamental para problemas de toma de decisiones [16]. Un MDP cuenta con los siguientes elementos:

- Conjunto de estados (S): representa las configuraciones posibles del entorno. En el caso de este proyecto, cada estado incluye: energía del agente, hora del día, temporada, inventario y posición.

- Conjunto de acciones (A): conjunto de decisiones que puede tomar el agente. En el caso de este proyecto, cuenta con hasta 10 posibles acciones: movimientos, comer, dormir, cultivar y cosechar entre otros. Cada acción que realiza el agente se provoca una transición de estado según las reglas del entorno.

- Función de transición ($P(s'|s,a)$): especifica la probabilidad de alcanzar el estado s' desde s al ejecutar a. El entorno de este proyecto presenta transiciones básicas que se alejan de otros videojuegos que pueden presentar situaciones estocásticas.

- Función de recompensa ($R(s,a,s')$): se encarga de la retroalimentación numérica que se otorga en cada transición. En este proyecto de cuenta con un sistema variado de recompensas o penalizaciones que puede acabar tanto en positivo como en negativo.

- Factor de descuento (γ): pondera la importancia entre recompensas futuras y recompensas inmediatas. Un valor cercano a 1 favorece estrategias a largo plazo [7].

La propiedad de Markov establece que la evolución futura del proceso depende únicamente del estado actual:

$$P[S_{t+1}|S_t, A_t] = P[S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, ..., S_0, A_0] \tag{1}$$

Esta propiedad permite resolver problemas complejos mediante programación dinámica [9], evitando la necesidad de considerar el historial de interacciones.

2.5. Ecuación de Bellman

La ecuación de Bellman [8] es fundamental en el RL al dividir problemas de decisión en pasos más pequeños. Ayuda al agente a calcular el valor de cada estado considerando las recompensas, permitiéndole tomar mejores decisiones, maximizando su recompensa como se ve en la Figura 1.

2.6. Algoritmos clave

- Q-Learning [26, 10]: algoritmo de RL que se basa en la estimación de una función $Q(s,a)$ en la que se representa el valor esperado de realizar una acción a en un estado s. La tabla Q, o Q-table, almacena estos valores para cada par estado-acción.

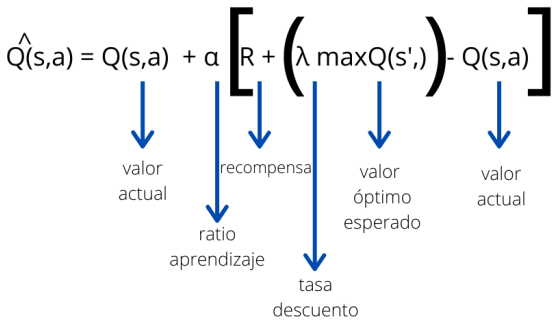

Figura 1: Ecuación de Bellman

La actualización de la esta tabla se realiza con la siguiente fórmula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Esta fórmula ajusta los valores de la tabla Q para reflejar la utilidad de las acciones en cada estado donde:

- α es la tasa de aprendizaje.
- γ es el factor de descuento.
- r es la recompensa que se recibe al tomar la acción a en el estado s.
- s' es el nuevo estado al que se llega al ejecutar la acción a.
- $\max_{a'} Q(s',a')$ es el valor máximo estimado del estado siguiente s'.

- Experience Replay [14, 13]: técnica que almacena experiencias en un búfer de capacidad fija que permite reutilizar datos, mejorar la estabilidad al mezclar experiencias temporales y logra un aprendizaje continuo.
- Exploración ε-greedy [2]: estrategia dedicada a la selección de acciones de forma balanceada donde se vuelve a exponer el equilibrio entre la exploración y la explotación.

2.7. Aprendizaje por refuerzo aplicado a videojuegos y demostraciones

Los videojuegos ofrecen un entorno ideal de pruebas para algoritmos de RL [6], ya que pueden poner a disposición de cualquier trabajando con estos:

- Entornos fácilmente controlables y totalmente reproducibles (según el control que se tenga sobre el juego y su código).
- Reglas claras y bien definidas a través del diseño de mecánicas.
- Métricas objetivas ya definidas gracias a las puntuaciones.
- Amplio abanico de comportamientos que requieren diferentes estrategias y acciones (y su exploración) para alcanzar los objetivos definidos.
- Posibilidad de entrenar agentes con observación visual o características similares a esta, imitando a la humana.
- Dificultad ajustable para poner a prueba al agente en diferentes escenarios.
- Bajo coste computacional en comparación con entornos físicos.
- Simulación rápida y segura para realizar una gran cantidad de iteraciones de entrenamiento.

El hito fundacional fue el desarrollo del modelo de MNih [14]. Lograron superar con su modelo más del 75 % de la puntuación obtenida por humanos en más de la mitad de los 49 videojuegos clásicos de la Atari 2600 (un total de 29). Pudieron lograr estos resultados al tener en cuenta las siguientes innovaciones:

- Reutilizar y aprovechar conocimiento de experiencias pasadas gracias al uso de experience replay.
- Mejora en la estabilidad al implementar una red objetivo (target network) por separado.
- Procesamiento directo de píxeles mediante redes convolucionales.

Además de este avance y en poco más de un año, AlphaGo [19] y su sucesor más generalizado AlphaZero [20] consiguieron revolucionar el campo del RL al demostrar que se podían dominar juegos con una gran complejidad estratégica y combinatoria como son el Go, el ajedrez o incluso el shogi. Entre muchos de estos avances, AlphaZero mostró su capacidad de:

- Aprender desde cero sin un conocimiento previo.

- Innovar y desarrollar estrategias no documentadas hasta el momento.
- Superar a motores tradicionales basados en búsqueda.

En el ámbito de los juegos de estrategia en tiempo real (RTS), conocidos por su profundidad estratégica, el trabajo en StarCraft II de Vinyals et al. [25] estableció nuevos estándares al manejar:

- Espacios de acción de gran dimensionalidad (mayores que en ajedrez o Go), con aproximadamente 10^{26} posibilidades.
- Larga duración de las partidas, pudiendo llegar a sesiones de 1 hora de juego.
- Información imperfecta (gracias a la mecánica de fog of war) y observaciones parciales del estado del videojuego.

En juegos de gestión de recursos como con el que se trabaja en este proyecto, el RL ha llegado a mostrar resultados interesantes como el caso de Agent57 [4]. Demostró una capacidad para adaptarse a diferentes géneros de videojuegos, incluyendo incluso los que requieren de una buena planificación tanto a corto como largo plazo, además de un acceso limitado a recursos.

En juegos que requieren de planificación estratégica y de un manejo de recursos limitados, como sucede en alguno de los videojuegos de la Atari 2600, el RL ha visto avances con agentes como Agent57 [4], que obtiene su nombre debido a superar hasta en 57 videojuegos al rendimiento y nivel humano. De entre estos 57 videojuegos no todos se acercan al género de gestión por recursos, pero es importante destacar el control del agente en escenarios donde los recursos son limitados y su correcto uso es lo que permite al agente lograr esas puntuaciones.

Un caso relevante es el del conocido y popular videojuego sandbox Minecraft [5], donde los agentes aprendieron a:

- Fabricar herramientas en secuencias complejas.
- Gestión de un inventario limitado a través de las mecánicas del videojuego.
- Priorizar objetivos a corto y largo plazo.

2.8. Limitaciones actuales y desafíos

A pesar de todos estos trabajos y el progreso que han logrado, el RL aplicado a videojuegos enfrenta varios retos fundamentales [3]:

- Eficiencia en el muestreo: la mayoría de algoritmos requieren de una cantidad enorme de interacciones que además superan con creces a las del humano en casos donde ambos rinden a niveles similares en el videojuego. Un ejemplo de esto sería DQN, que necesitó 50 millones de frames para aprender a jugar [14].

- Diseño de recompensas: en [1] se documentan casos donde los agentes encuentran soluciones que no se esperaban y donde incluso logran maximizar la puntuación o recompensa obtenida. Estas soluciones no son útiles en realidad ya que no cumplen el objetivo real que se espera. Un ejemplo podría ser recompensar a un robot por mantener por limpiar, ¿y si apaga su visión para no ver esa suciedad? Esto provocaría que el robot siempre este logrando su supuesto objetivo ya que no encuentra esa suciedad que debe limpiar.

- Generalización: como muestra [15], los agentes pueden llegar a ajustarse de más a entornos concretos, esto puede provocar que un agente entrenado en un nivel no pueda superar otro muy similar aunque no idéntico.

- Transferencia entre dominios: según [23], no es común que el conocimiento que un agente adquiera en un juego pueda transferirse a otros, ni siquiera estando ambos juegos en el mismo género.

Para juegos de gestión por recursos, específicamente, los desafíos incluyen:

- Un conjunto de recompensas muy espaciadas en el tiempo de juego.

- Espacios de estado que superan con creces a cualquier videojuego de otro género diferente a este.

- Sistemas de puntuación dinámicos que pueden chocar con un acercamiento estático como el que se puede observar en el entrenamiento de otros videojuegos.

3

Conceptualización del videojuego

3.1. Justificación del diseño

En este capítulo se analizará la conceptualización del videojuego, desde la idea inicial hasta su concepto completo y final, pasando por las decisiones tomadas y el porqué de estas. El capítulo se divide en tres partes: el diseño conceptual del videojuego, donde se comenta desde un punto de vista más genérico; el diseño de mecánicas, donde se profundiza en las acciones e interacciones; y el diseño visual, que aborda el aspecto del proyecto, que será útil tanto para realizar las pruebas necesarias para confirmar el funcionamiento de la implementación y del agente.

3.1.1. Diseño conceptual

El primer concepto del videojuego desarrollado para este proyecto nació de la necesidad de un escenario que pudiera poner a prueba a un agente de RL pero que al mismo tiempo contara con una complejidad sencilla. Partiendo de esta base, la elección del género cayó por su propio peso, gracias a todo lo que pueden ofrecer los videojuegos de gestión por recursos y la profundidad que pueden alcanzar.

Estos videojuegos presentan una simulación muy cercana a lo que puede ser una gestión real y tangible de recursos manejados por humanos donde con unas pocas reglas y mecánicas pueden llegar a miles de combinaciones muy interesantes. Aunque no termina de tener exactamente el mismo género con el que se trabaja en este proyecto. Un claro ejemplo de esto es Cities Skylines, donde más de una vez se ha podido mostrar gracias a este videojuego como ciertos

cambios o condiciones urbanísticas pueden afectar a pueblos y ciudades [17].
También se aprovecha la oportunidad que ofrece el proyecto para trabajar con un género, como es el de gestión de recursos, que no cuenta con tantos estudios o reflexiones como se encuentran para otros géneros como el RTS o arcade.

En el caso de este proyecto, el videojuego tendrá un enfoque relacionado con la plantación, recolección y abastecimiento de cultivos y los alimentos que estos generen. Este género se suele adaptar a infinidad de escenarios siempre y cuando haya recursos que gestionar pero, la temática principal y más popular es, sin lugar a dudas, la de granjas y cultivos.

Como se busca un complejidad sencilla se decide modelar el mundo virtual en una disposición de cuadrícula en dos dimensiones (recordando a un array bidimensional) en el que el agente realiza las acciones que vea convenientes para superar los retos y objetivos a los que se enfrente.

3.1.2. Diseño de mecánicas

A la hora de diseñar las mecánicas del videojuego se busca una coherencia y simpleza entre ellas, haciendo que sea lo más intuitivo y fácil de comprender posible. Por el propio interés del diseño de videojuegos y con vistas a como pueda afrontarlo el agente de RL, hay mecánicas y elementos que interactúan entre sí, donde la capacidad de comprender estas relaciones y sinergias será crucial para poder avanzar en las ejecuciones.

Previo a la enumeración y descripción de forma concreta de cada una de las mecánicas del videojuego, es necesario mencionar el bucle principal del juego.
El bucle de juego tiene como protagonista a un granjero, que servirá al agente para interactuar con los elementos que se encuentre en el mundo virtual. Por no limitar al agente en las simulaciones, el videojuego no cuenta con ningún final u objetivo cerrado, sino que el diseño tiene en cuenta la relevancia que puede ofrecer al análisis del agente una libertad total. Teniendo esto en cuenta, el único requisito que debe cumplir es el de no quedarse sin energías durante el mayor tiempo posible. Para evitar quedarse sin energía a lo largo de las ejecuciones, el agente deberá aprender a interactuar con aquellos elementos que más le favorezcan para ir mejorando

y superando su récord de días.

El agente dispondrá de un inventario, en el que podrá almacenar todas las semillas y frutos que vaya consiguiendo a medida que avanza en las simulaciones. Tendrá que tener en cuenta los cambios de estación y como se relacionan las estaciones con sus cultivos para asegurar una cosecha prolifera y sin problemas.

1. Generación procedimental de mapas

   Para lograr un mejor análisis sobre el comportamiento del agente, se decide implementar un sistema de generación procedimental de mapas para lograr una disposición diferente, en cada iteración, de los elementos que encontrará.

   - Características básicas del mapa:

     Cada mapa cuenta con una disposición de 15x15 casillas donde se pueden generar o no las casillas especiales que se mencionan en el próximo punto (en caso de que una casilla no sea especial, no se generará nada en ella y simplemente contará como suelo). El agente tiene total libertad de movimiento a lo largo del mapa, pudiendo moverse de una casilla en una, siempre y cuando algún elemento o casilla no le impida el paso.

   - Casillas especiales:

     - Agua: se generan clusters de 4–8 casillas conectadas, ofreciendo al agente zonas en las que cultivar semillas.
     - Tierra: se generan de 1–8 casillas a lo largo del mapa que servirán al agente para plantar árboles.
     - Casa: se genera una casilla reservada como la casa del agente en la que podrá descansar y recuperar parte de su medidor de energía. Siempre se asegura el acceso a esta casilla.

   - Propiedades de la generación del mapa:

     - Las simulaciones cuentan con un espacio de estados acotados aunque no deja de ser trivial al poder lograr aproximadamente 10^{135} configuraciones posibles (15×15 casillas $\approx 10^{135}$, donde casa casilla puede ser suelo, agua, tierra o la casa).

        - Se garantiza una jugabilidad básica en todos los mapas donde no hay elementos o casillas que estén diseñadas para un detrimento del rendimiento del agente.

2. Sistema agrícola y su clasificación

   El agente dispondrá de una gama reducida de semillas que podrá plantar para obtener con el paso del tiempo la cosecha del fruto que den sus cultivos. Para el videojuego se han implementado dos tipos de semillas: las de cultivos estacionales y las de árboles frutales.

   - Cultivos estacionales:
     - Requisitos para plantar la semilla: debe plantarse en una casilla colindante a otra casilla de agua para que pueda crecer con el paso de los días.
     - Ciclo de vida de la planta: una vez plantada la semilla, el agente deberá esperar 2 días para que madure y así poder recoger su fruto. Además, estas plantas, en caso de darse un cambio de estaciones, se marchitará y el agente no podrá aprovechar el fruto que genera.
     - Cosecha y recompensa: en caso de que el agente logre cosechar la planta se le añadirá a su inventario el fruto y 3 semillas para poder cultivarlas de nuevo. Estos cultivos estacionales se dividen en 3 tipos: tulipanes, girasoles y hojas de arce. En el apartado de las estaciones se especifica el rol de cada uno.
   - Árboles frutales:
     - Requisitos para plantar la semilla: debe plantarse sobre una casilla de tierra, independientemente de que la tierra esté rodeada o cerca de una casilla de agua.
     - Ciclo de vida de la planta: una vez plantada la semilla, el árbol tardará en crecer 6 días, pero a diferencia de los cultivos estacionales, estos árboles no se marchitarán con el cambio de estaciones.
     - Cosecha y recompensa: producen manzanas cada 4 días en las casillas que tenga el árbol arriba, a la derecha, abajo y a la izquierda en caso de ser posible.

3. Sistema de tiempo y estaciones

- Modelo temporal:
  - Cada acción que ejecute el agente avanza el reloj 10 minutos, creando una relación directa entre el tiempo del que dispone y la energía que le conlleva realizarla.
  - El videojuego cuenta con un ciclo diario típico para ayudar a la comprensión de las simulaciones y hacerlas más intuitivas, es decir, días de 24 horas que además incluyen una fase diurna (de 6:00–19:00) y nocturna (de 19:00–6:00, donde las acciones que realice el agente le costarán el doble de energía).
- Estaciones:
  - Se implementan las cuatro estaciones del año, donde cada una cuenta con una duración de 12 días y una regla concreta para cada una:
    - Primavera: estación exclusiva para cultivar tulipanes.
    - Verano: estación exclusiva para cultivar girasoles.
    - Otoño: estación exclusiva para cultivar hojas de arce.
    - Invierno: estación sin posibilidad de cultivar.

4. Sistema de energía y supervivencia

   Todas y cada una de las acciones del agente le costarán cierto porcentaje de su medidor de energía, lo que permite establecer y crear un balance entre acciones más triviales y acciones importantes. Por otro lado, el alimentarse con lo que logre cosechar o descansando en la casilla especial de la casa, podrá recuperar energía.

   - Mecanismos de recuperación:
     - Dormir: proporciona un aumento del 10 % del medidor de energía por cada hora de descanso. Para poder dormir debe estar dentro de la casa. El agente cuenta con una acción que le permite dormir 8 horas seguidas.
     - Comer:
       - Manzanas: +1 % por unidad.
       - Frutos: +3 % por tres unidades si se consumen en la estación a la que pertenece su planta, 1 % en caso contrario.

    - Castigo energético: si el agente fuerza su medidor por debajo del 20 % de energía, la recuperación de energía a la hora de dormir se verá reducida a la mitad.
- Condición de pérdida de la partida y fin de la simulación: si en cualquier momento de la ejecución el medidor del agente alcance un 5 % o menos, el juego y la simulación terminan.

3.1.3. Diseño visual

El diseño visual del videojuego se ha realizado sobre tres pilares esenciales: claridad informativa del estado en el que se encuentra la ejecución, simpleza en los elementos gráficos para optimizar recursos a la hora de ejecutar simulaciones y un acercamiento sencillo a la representación de los elementos para dar prioridad a la implementación.

Al comienzo de la conceptualización visual se barajó la posibilidad de utilizar motores gráficos dedicados a los videojuegos (como, por ejemplo, Godot) pero eso solo aumentaba la escala del proyecto y sería un esfuerzo no crucial para el objetivo principal, por lo que se terminó descartando la idea para optar por un acercamiento más sencillo: aprovechar librerías de Python que funcionaran sin problemas con las que más tarde se usarían para el modelo de aprendizaje.

Una vez encasillado el diseño visual en la tecnología que fuera a mostrarlo, la siguiente decisión a tomar fue elegir entre gráficos ya diseñados con licencias acordes al proyecto o hacer gráficos propios. Investigando y probando con varias librerías se llegó a la conclusión de que lo mejor para el proyecto era aprovechar gráficos que cualquier pudiera entender, que fueran fáciles de implementar y que incluso evitaban el trabajar directamente con gráficos: los emojis.

Todo el apartado visual del proyecto, como se puede ver en la Figura 2, está basado en colores planos, una misma fuente y otra fuente dedicada a los emojis. Gracias a trabajar con los emojis se permite una integración sin resistencia entre el código y las representaciones visuales de los elementos, provocando un resultado simple pero eficaz.

La elección de emojis como representante visual pueden parecer un acercamiento menos profesional que otros, pero no más lejos de este prejuicio, son una opción más que eficaz, potente y útil. Al tratarlos como una fuente, permiten reescalado sin ningún tipo de problemas, no suponen un consumo ni de capacidad ni de memoria extra, no ponen entre el proyecto y el

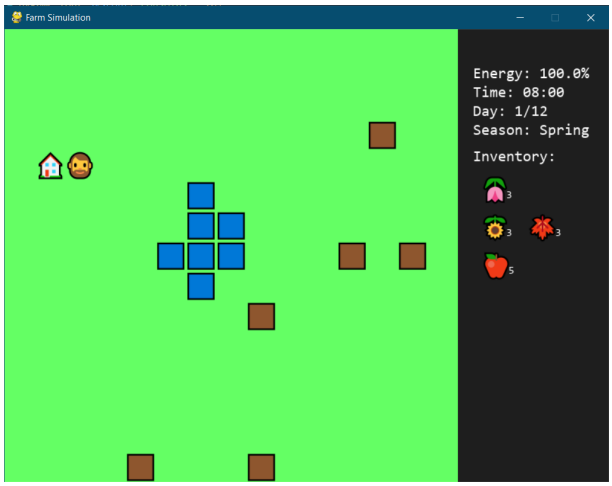

Figura 2: Ejemplo de la simulación al iniciarse

Figura 3: Primer concepto desde la consola

espectador una distancia visual, como se podría dar en casos donde los gráficos sean nuevos u originales y son una gran herramienta para realizar prototipos.

En las primeras implementaciones, el desarrollo fue más fluido de lo normal al no tener que estar pendiente de cómo de bien o mal se verían los gráficos, no tener que buscar imágenes provisionales (placeholders) y poder hacer las primeras pruebas directamente desde la consola del entorno de desarrollo como muestra la Figura 3.

- **Guía visual**:
    - Semillas y plantas: en la interfaz del inventario y en la representación del mapa se pueden observar emojis relacionados con los cultivos que puede plantar el agente. Las semillas se muestran como la flor a la que pertenecen pero girada 180 grados,

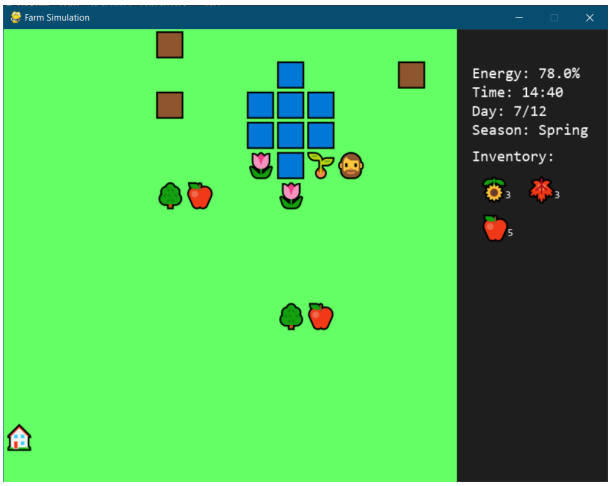

Figura 4: Ejecución con todas los emojis a la vista

para poder diferenciarlas de las plantas una vez terminen de crecer. Una vez se planta una semilla, aparecerá como una semilla germinada. En el caso de los árboles y las manzanas, estos se representan con sus correspondientes emojis una vez crecen o se generan.

- Casillas especiales: tanto el agua, como la tierra y la casa se muestran en el mapa con cuadrados azules, cuadrados marrones y el emoji básico de una casa.
- Estacionalidad y ciclos: los colores del mapa se adaptan tanto a la estación del año: verde claro (primavera), verde (verano), naranja (otoño), azul claro (invierno) como a los ciclos diurnos y nocturnos: por el día muestra el color de la estación y por la noche se presenta el mapa con un tono azul oscuro.

En la Figura 4 se pueden observar todas las referencias que se mencionan en este apartado.

Interfaz de usuario

- Panel derecho:
  - Información del agente y el entorno:
    - Medidor de energía del agente.
    - Hora, día de la simulación y la estación del año en la que se encuentra.

- Inventario:
    - Diseño de doble columna que muestra:
        - Semillas y frutos de los que dispone el agente.
        - Cantidad de cada objeto superpuesta encima de cada uno de los emojis.

Finalmente, para cuando el agente se agote y no disponga de más energía, se muestra un simple texto que menciona el fin de partida para dejar claro que no se puede continuar ni avanzar con la ejecución.

4

Marco tecnológico y metolodológico

Aunque la importancia de este proyecto tiene el foco puesto en el diseño de mecánicas con las que el agente puede interactuar, el porqué de estas mecánicas y el modelo de RL, el desarrollo del videojuego sigue siendo una parte importante del trabajo y que, por lo tanto, se debe comentar.

4.1. Implementación y tecnologías utilizadas

El desarrollo del videojuego ha seguido un enfoque iterativo y modular, priorizando la claridad y reutilización del código. Todo el videojuego se ha podido implementar con tan solo usar Python y la librería Pygame, aprovechando las facilidades que ofrece el entorno de desarrollo Visual Studio Code.

La base del videojuego se ubica en la clase principal llamada FarmGame, que encapsula toda la lógica del juego. Esta clase incluye desde la generación del mundo hasta la gestión del inventario, pasando por las mecánicas relacionadas con la energía o los ciclos de cultivos. También actúa como el núcleo principal del videojuego, logrando mantener toda la información y el estado del entorno y del agente.

4.1.1. Estructura del código

- Se trabaja con variables globales colocadas al principio del archivo de código y de forma ordenada en caso de querer ubicar o modificar ciertos parámetros cómodamente. Entre estas variables globales se ubican los costes de energía de cada acción, la duración de los días o incluso los emojis utilizados en el videojuego.

- Además de las variables globales y, como ya se ha comentado, el archivo cuenta con la clase FarmGame, la cual agrupa:
    - Los datos internos del juego como la posición del agente, su medidor de energía, el día, la hora, la estación, el inventario y todos los valores necesarios para cada tipo de cultivo.
    - Métodos de interacción con el mapa como moverse, dormir o comer a los que tiene acceso el agente y que muestran cambios visuales para el entendimiento del espectador.
    - La generación procedimental del mapa y el resto de métodos internos a los que el agente no tiene acceso al no ser acciones como los que se encargan de actualizar el estado de las plantas o árboles o incluso generar las manzanas.
- Un bucle principal de ejecución que escucha los eventos de teclado y actualiza visualmente el videojuego al igual que lo haría con el agente (con motivos de depuración por parte del desarrollador).

4.1.2. Metodología de desarrollo

La implementación del videojuego se estructuró en un módulo que centraliza la lógica del juego. De esta forma se puede priorizar la conexión y cohesión con la implementación del modelo de RL, optando al mismo tiempo por una arquitectura monolítica que se apoya en:

- Uso de variables globales (o constantes) nombradas en mayúsculas para facilitar cambios o pequeños ajustes rápidos sin tener que buscarlas a lo largo del código.
- Métodos privados a los que el agente nunca podrá acceder para poder controlar sin problemas ni acciones no deseadas el flujo del videojuego.
- Nombres de funciones y variables autoexplicativos, siguiendo un formato consistente como con los métodos privados y los públicos, en los que se introduce una barra baja delante de los métodos en caso de ser privados.
- División estructurada según la función que cumple cada uno de los métodos.

Aunque no se ha seguido un paradigma de orientación a objetos tradicional, el uso de una clase principal es más que suficiente para mantener el código organizado y legible, permitiendo realizar el mantenimiento que necesite sin problemas. Esta decisión se ha tomado ya que, como ya se ha comentado al inicio, el foco del proyecto no reside en el desarrollo del videojuego sino en el diseño de mecánicas, su interactividad y las pruebas realizadas con el agente.

El desarrollo se realizó de manera incremental, acotando según funcionalidades independientes. Se comenzó con una plantilla de mapa que no contaba con ningún tipo de generación para más tarde ir implementando las mecánicas del movimiento, la energía y el paso del tiempo. Sobre este núcleo de mecánicas se pudo entonces implementar e incorporar el resto de ellas: funcionamiento e interacción con las plantas, el paso de las estaciones y su efecto sobre los cultivos y por último el inventario.

4.2. Metodología de verificación y pruebas

El proceso de validación de la implementación se realizó mediante pruebas manuales iterativas adaptadas a la escala del proyecto y sus características:

- Enfoque ágil: cada nueva funcionalidad implementada se ponía a prueba y se verificaba al momento aprovechando la interacción directa con el sistema gracias al bucle principal que espera entradas por teclado, permitiendo:
    - Detección temprana de errores o anomalías.
    - Validación de las funcionalidades y sinergias que se den en momentos concretos con las mecánicas en tiempo real.
    - Retroalimentación visual que puede reflejar problemas tanto internos (ya sean de datos o código) como externos (por fallos en la representación visual).
    - Poder revisar la interacción de una nueva funcionalidad con diferentes elementos del mapa sin tener que realizar pruebas o casos por separado.

4.2.1. Ventajas y limitaciones

Una de las ventajas de este acercamiento es la adaptabilidad a un proyecto con requisitos y funcionalidades menores donde interesa hacer o poner a prueba diferentes valores o ajustes. Sin embargo, alguna de las limitaciones conscientes con esta metodología incluye la falta de una cobertura tradicional.

En el caso de tratarse de un proyecto de mayor envergadura o con una complejidad mecánica y de entrelazado entre ellas superior, sería interesante enfocar su validación mediante un sistema de pruebas automatizadas o un marco formal de pruebas (como podría ser pytest).

Con este flujo de trabajo se ha logrado validar de forma eficaz el comportamiento del videojuego y no se ha encontrado en ningún momento problemas críticos, ni en pruebas acotadas, ni con el agente ejecutando miles de simulaciones.

5

Análisis previo al aprendizaje por refuerzo

5.1. Propósito principal del agente

El propósito principal del agente radica en adquirir una capacidad y conocimiento incremental del entorno en el que se le ha situado. Desde ser capaz de realizar sus primeros movimientos hasta el punto de poder predecir estaciones o situaciones que le beneficien o no incluso en diferentes generaciones del mapa.

5.2. Metas a lograr

5.2.1. Metas generales

Partiendo del propósito general, una lista de posibles metas que podría lograr el agente tras el aprendizaje necesario incluyen:

- Poder caminar por el mapa sin problemas.
- Mostrar capacidad de cultivo en caso de encontrar una casilla especial que se lo permita.
- Regresar a casa y/o alimentarse para recuperar energía.

Gracias a la baja dificultad que se ha diseñado, estas tres metas generales permitirían al agente un bucle de juego sin complicaciones donde podría ir avanzando poco a poco y recuperarse.

5.2.2. Metas específicas

Además de las generales, se pueden comentar metas específicas o más complicadas a las que se podría enfrentar el agente:

- Análisis previo de las casillas especiales de las que dispone para optimizar sus recursos y energías aunque el bucle de juego no sea el natural.
- Evitar conductas predecibles en pos de mejorar el rendimiento y maximizar los días que aguanta con energías.
- Suprimir mecánicas que el agente puede entender como decisiones que le perjudican más que le benefician.

5.3. Posibles problemas

5.3.1. Problemas generales del RL

- Exploración insuficiente: el agente puede que no descubra ciertas estrategias si no visita los estados suficientes como puede darse con las estaciones.
- Asignación de crédito: las recompensas tardías (como los frutos que aparecen días después de plantar) dificultan saber qué acciones fueron realmente útiles para lograr la recompensa de recursos.
- Sobreajuste: riesgo de que el agente aprenda políticas que funcionen en mapas con cierta disposición de elementos pero que en otros mapas no.

5.3.2. Problemas específicos del juego

- Desbalanceo estacional: en invierno desaparecen todas las plantas, lo que puede pillar desprevenido al agente si ha aprendido a mantener un bucle de juego que priorice los cultivos estacionales.
- Complejidad del estado: combinar 9 variables diferentes (energía, hora, estación, etc.) crea un espacio de estados enorme y difícil de explorar completamente.

- Seguridad del hogar: puede que el agente se de cuenta de que al final la mejor opción no participar en ninguna de las mecánicas que ofrece el videojuego y descansar todas horas en la casa.

5.3.3. Soluciones implementadas

- Exploración adaptativa: uso de ε-greedy que empieza explorando mucho y gradualmente explota más.
- Recompensas intermedias: sistema de 18 tipos de recompensas que guían al agente paso a paso.
- Memoria de experiencias: almacenamiento y muestreo aleatorio de transiciones pasadas para mejorar el aprendizaje.

6

Diseño e implementación del modelo

El modelo de RL se basa en un enfoque que utiliza un entorno modelado como un MDP. Este apartado describe los algoritmos utilizados, la interpretación del entorno por parte del agente, el espacio de estados y acciones, y la función de recompensa, conectando estos elementos con los fundamentos teóricos del RL y las mecánicas del videojuego.

6.1. Algoritmos utilizados

El modelo de RL propuesto se basa en el algoritmo Q-Learning, un método basado en valores que aprende una función de valor-acción $Q(s, a)$ para determinar la política óptima. La elección de Q-Learning se debe a la naturaleza determinista de las mecánicas del videojuego (aunque cuente con elementos estocásticos como la generación del mapa) permitiendo al agente aprender sin necesidad de modelar las probabilidades de transición.

Para manejar el equilibrio entre exploración y explotación, se implementa una estrategia ϵ-greedy. A cada paso que da, el agente selecciona la acción con el mayor valor $Q(s, a)$ con una probabilidad $1 - \epsilon$ y una acción con probabilidad ϵ. El valor de ϵ se va reduciendo a lo largo del entrenamiento (por ejemplo, desde 0.9 hasta 0.1) para priorizar la explotación a medida que aprende.

Debido al tamaño del espacio de estados, se puede considerar una posible extensión de la implementación más encaminada al DQN, que utiliza una red neuronal para aproximar la función Q.

6.2. Interpretación del entorno para el agente

El entorno del videojuego donde el agente interactúa con las mecánicas, se interpreta como un MDP y que al mismo tiempo incluye una cuadrícula, recursos, ciclos temporales y un sistema de energía. La interpretación de este entorno se centra en:

- Observaciones: el agente percibe el estado actual a través de variables clave como su posición, el inventario o la energía. Estas observaciones cumplen con la propiedad de Markov al contener toda la información necesario sin depender de un historial previo.
- Acciones válidas: el agente evalúa las acciones disponibles en cada momento para evitar, por ejemplo, el plantar una semilla sin agua adyacente o si no tiene semillas en el inventario.
- Dinámica temporal: el agente debe interpretar el paso del tiempo y sus efectos, como el crecimiento de plantas y la producción de manzanas. También se logra introducir un desafío de planificación a largo plazo por el comportamiento de la mecánica de los árboles.

6.3. Espacio de estados y acciones

El espacio de estados es multidimensional debido a la variedad que ofrece el videojuego:

- Posición del agente: representada por coordenadas $[i, j]$ en una cuadrícula de 15×15, lo que da 225 posibles posiciones.
- Energía: un valor continuo entre 0 y 100.
- Inventario: cantidades de semillas y frutos.
- Cuadrícula: una matriz de 15×15 con estados posibles para cada casilla donde el agente puede considerar solo las casillas adyacentes.
- Tiempo y estación: la hora, el día y la estación, que afectan a la validez de las acciones y el crecimiento de las plantas.

El espacio de acciones es finito y consta de:

- Movimientos: moverse en 4 direcciones: arriba, derecha, abajo e izquierda.
- Interacciones con el entorno: plantar semillas, cosechar plantas, plantar árboles y recoger manzanas.
- Gestión de la energía: dormir por una u ocho horas y comer frutos.

Esto resulta en un total de aproximadamente 10 acciones, dependiendo de las condiciones del entorno.

6.4. Función de recompensa

La función de recompensa guía al agente hacia una política óptima. Aunque no está explícitamente definida en el código, se propone un esquema de recompensa basado en las mecánicas del juego y los objetivos del proyecto.

Las recompensas positivas incluyen: recolectar recursos como los frutos de las plantas o las manzanas, y la recuperación de energía por cualquiera de sus vías, tanto alimentándose como descansando en la casa.
Las recompensas negativas incluyen: costo energético según acciones para evitar malgastar energía, acciones nocturnas y el fin de partida. Con estas penalizaciones se busca encaminar al agente a una toma de decisiones óptimas sin desperdiciar el tiempo o energía.

6.5. Tecnologías y consideraciones de la implementación

6.5.1. Librerías principales

- pygame sirve como motor principal para la simulación visual y gráfica del entorno.
- numpy para realizar los cálculos numéricos del RL. Se utilizad para las operaciones vectorizadas en la tabla Q, discretizar el espacio de estados y calcular las recompensas acumuladas.
- pickle permite la persistencia del modelo al poder guardar y cargar los modelos entrenados.

6.5.2. Estructuras de datos clave

- defaultdict (de collections) para implementar la tabla Q como diccionario anidado que inicializa estados no vistos con arrays de ceros y que optimiza el acceso a valores Q(s,a).
- deque con un buffer circular para el experience replay que mantiene un historial de transiciones y que permite un muestreo aleatorio eficiente.

6.5.3. Visualización y monitorización

- matplotlib genera gráficos del aprendizaje del agente para estudiar sus avances, el cambio de la tasa de exploración y explotación y la cantidad de días sobrevividos.
- time para la medir la duración de los episodios y la velocidad del entrenamiento.

6.6. Conclusión del diseño

El diseño del modelo de RL utiliza Q-Learning con una estrategia ϵ-greedy para aprender una política óptima en un entorno de gestión de recursos. El entorno se interpreta como un MDP con un espacio de estados que combina la información espacial, temporal y de recursos, y un espacio de acciones finito que refleja las interacciones del juego. Las recompensas están diseñadas para equilibrar la gestión de energía y la acumulación de recursos, incentivando decisiones estratégicas a corto y a largo plazo.

6.7. Entrenamiento y puesta en acción del agente

6.7.1. Configuración del entrenamiento

El entrenamiento que se va a comentar en este apartado ha contado con 5000 episodios, un máximo de 100 pasos por episodio, una tasa de aprendizaje de 0.1 y un factor de descuento del 0.9995. De esta forma se permite al agente tener en cuenta tanto las recompensas inmediatas como las recompensas a largo plazo.

Con el objetivo de equilibrar la exploración y la explotación, se ha utilizado una política ϵ-greedy con un valor inicial de exploración del 100 %, que decae progresivamente hasta un 15 % y un tamaño del batch para actualizaciones de la tabla Q de 100.

Además, se ha diseñado un sistema de recompensas que premia acciones como recolectar recursos, alimentarse o descubrir nuevas zonas del mapa mientras penaliza acciones inválidas como quedarse sin energía o perder la partida.

6.7.2. Análisis y resultados

Durante el entrenamiento, el agente ha mostrado una evolución positiva en varios aspectos como el ser capaz de realizar acciones básicas. Estas acciones incluyen la capacidad de moverse por el mapa, alimentarse, entrar en la casa, descansar en ella y plantar cultivos, tanto los estacionales como los árboles.

Sin embargo, más allá de estas habilidades básicas, el agente parece no mostrar indicios de haber desarrollado un flujo de trabajo que se pueda considerar como razonado. En las ejecuciones gráficas ejecutadas gracias a los archivos .pkl generados por el modelo de RL, no se termina de observar que el agente haya obtenido consciencia plena del entorno que le rodea. Esto provoca que el agente no aproveche al máximo las oportunidades que le ofrece el entorno con los elementos que dispone, provocando un rendimiento menor del esperado.

A continuación se muestran cuatro gráficas que reflejan esta situación en la que la mayoría de los episodios se caracterizan por patrones repetitivos, donde el agente realiza acciones muy parecidas sin una gran profundidad lógica, que acaban provocando unos resultados casi lineales.

En la Figura 5 se puede observar el rendimiento del agente cuantificado mediante la recompensa que ha ido obteniendo a lo largo de los episodios. El 100 de la leyenda hace referencia al promedio móvil de las recompensas totales en una ventana de 100 episodios consecutivos. Realizar este cálculo de esta forma permite:

- Filtrar posible ruido de episodios individuales.
- Evitar a lo largo de la gráfica la interferencia de valores atípicos en el muestreo.
- Mantener y mostrar posibles patrones a lo largo de la representación.

Como ya se ha mencionado, en la Figura 5 se observa como el agente muestra un comportamiento donde incrementa la cantidad de penalizaciones según van pasando los episodios en

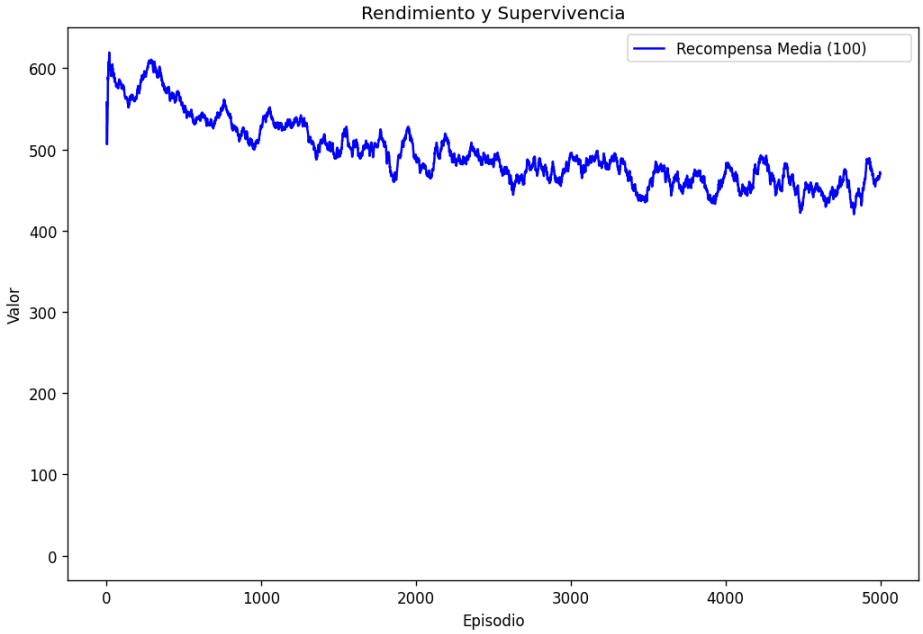

Figura 5: Recompensa media

vez de un incremento de recompensas. Como se puede ver en la Figura 6 este comportamiento parece estar conectado al dilema de la exploración y explotación del que ya se ha hablado en esta memoria.

A medida que avanzan los episodios, se muestra en la Figura 6 como la tasa de exploración va disminuyendo tal y como se espera de la política ϵ-greedy, manteniéndose en un mínimo de 0.15 establecido en los parámetros del modelo.

En esta segunda gráfica se observa como la reducción de la exploración afecta negativamente al rendimiento del agente. Tanto los errores como la media de estos muestran un carácter ascendente a medida que el agente reduce su índice de exploración, pudiendo relacionar con certeza que los resultados de la Figura 5 se deben a esta reducción.

Además de estas gráficas que reflejan el rendimiento y evolución del agente, es importante reflexionar sobre el reto principal al que se enfrentaba: aguantar la máxima capacidad de días sin quedarse sin energías. Similar a las figuras 5 y 6, en esta nueva Figura 7 se observa una consistencia (aunque mínima) de días que ha conseguido aguantar el modelo sin gastar su medidor de energías.

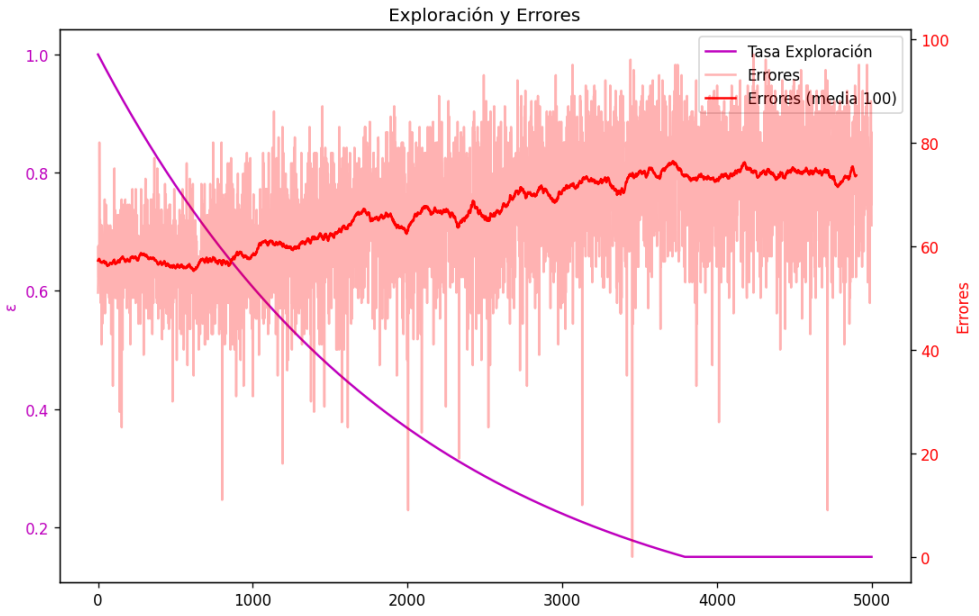

Figura 6: Exploración y errores

El análisis de esta gráfica se puede dividir en dos enfoques: el mecánico y el del modelo. Por una parte habría que estudiar si esta consistencia que se observa es fruto de los valores atribuidos a las mecánicas del videojuego, que provocan un ambiente ideal para que el agente, sin apenas realizar acciones relevantes, obtenga estos resultados. Por otro lado, puede que esta capacidad del agente realmente demuestre una capacidad tangible de su avance y evolución desde que comienza su aprendizaje, debido a una correcta configuración de los parámetros y recompensas. Para poder declarar una respuesta sería necesario exponer el videojuego a personas que intenten obtener resultados similares, pudiendo comprobar si la meta lograda resulta de una reacción en cadena de mecánicas o de un agente que consigue dar sus primeros pasos en la dirección correcta.

También se observan tres valores atípicos que coinciden con la capacidad del agente de aguantar hasta tres días sin quedarse sin energías. Estos valores podrían indicar dos cosas: un golpe de suerte por parte del agente o la capacidad de poder evolucionar en ciertos entornos si las casillas o elementos están dispuestos de formas concretas.

Como última gráfica y volviendo a visitar la cantidad de días sobrevividos, la Figura 8. En esta figura se observan puntos que son capaces de referenciar tres datos a la vez ofreciendo

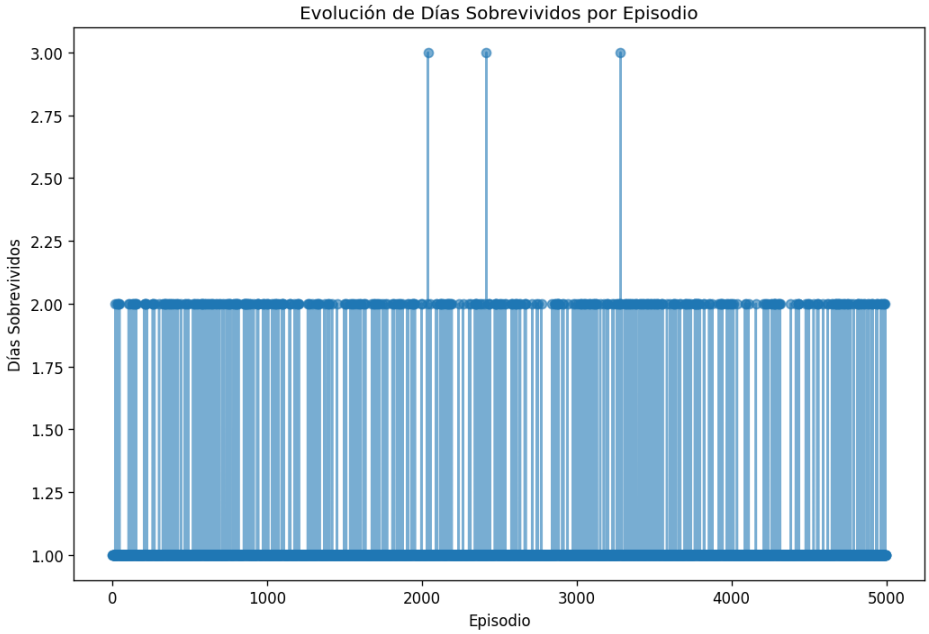

Figura 7: Aguante de energías en días

diferentes puntos de vista sobre la información recabada, reflejando a su vez casos concretos, como por ejemplo:

- Las tres simulaciones en las que el agente llegó a los tres días se consiguieron recompensas que se encuentran entre las más altas. Esto indica que llegó por haberse adaptado hasta cierto grado al entorno y no por suerte o coincidencia.

- Esta misma correlación entre la recompensa obtenida y los días de aguante se puede observar en las columnas de puntos y sus diferencias. En la segunda columna se puede ver como, aunque haya menos cantidad de puntos en general, no se encuentra la misma densidad de puntos que tiene la primera columna entre el 0 y 200 de recompensa, demostrando que hay una relación entre aguantar más días y lograr una mayor recompensa.

- Se vuelve a plasmar como los episodios más avanzados y por ende con menos exploración se acaban estancando en la cantidad de días que aguantan. Esto queda claro al observar que la gran mayoría de puntos amarillos, que representan los episodios tardíos, se concentran en la primera columna, mientras la segunda y tercera apenas disponen de este color.

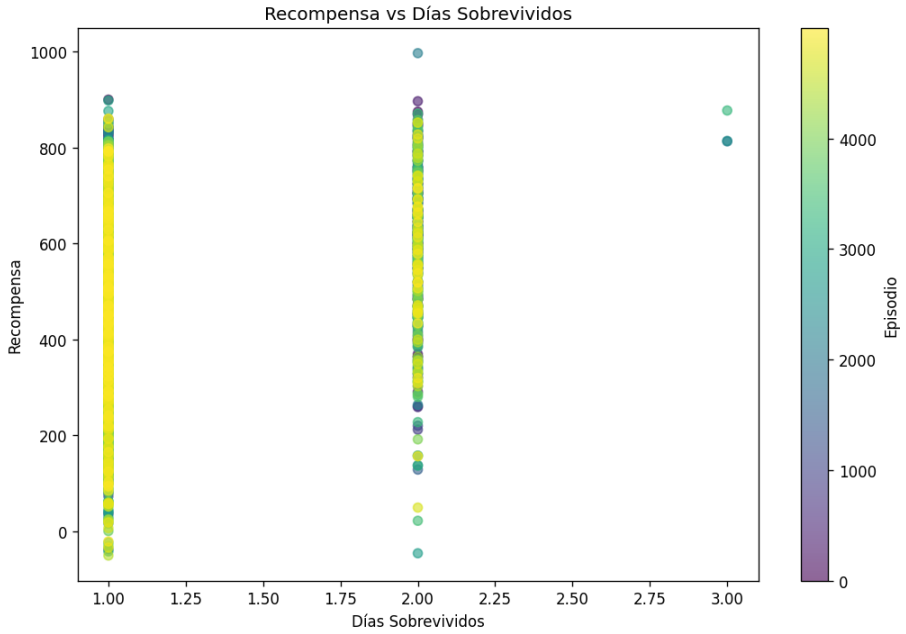

Figura 8: Recompensas, días y episodios

- Aunque se consideren casos atípicos, es interesante mencionar y teorizar tanto con el mejor caso de la segunda columna, como con el peor, y que puede implicar cada uno de ellos. El mejor caso muestra una recompensa aproximada al 1000, lo que podría indicar un agente que ha sido capaz de cultivar, alimentarse, descubrir casillas especiales, recolectar frutos y plantar árboles, mientras que el peor de los casos puede ser un mínimo viable con el que el agente evitó penalizaciones por su comportamiento, pero que provocaría no poder progresar.

Para finalizar con el análisis de los resultados obtenidos se puede hacer una última reflexión al superponer las tres primeras gráficas, obteniendo un nuevo prisma desde el que observar la información recolectada con la Figura 9.

Si se observa con detenimiento, las líneas que corresponden con las ejecuciones que han llegado a los tres días de aguante, muestran ciertos picos. Aunque no estén demasiado pronunciados, estos picos se dan tanto en la media de recompensas como en la media de errores, pudiendo indicar que aquellas ejecuciones fueron el resultado de combinar errores, que aunque negativos, pudieran servir de aprendizaje o de guía a acciones con buena carga de recompensa.

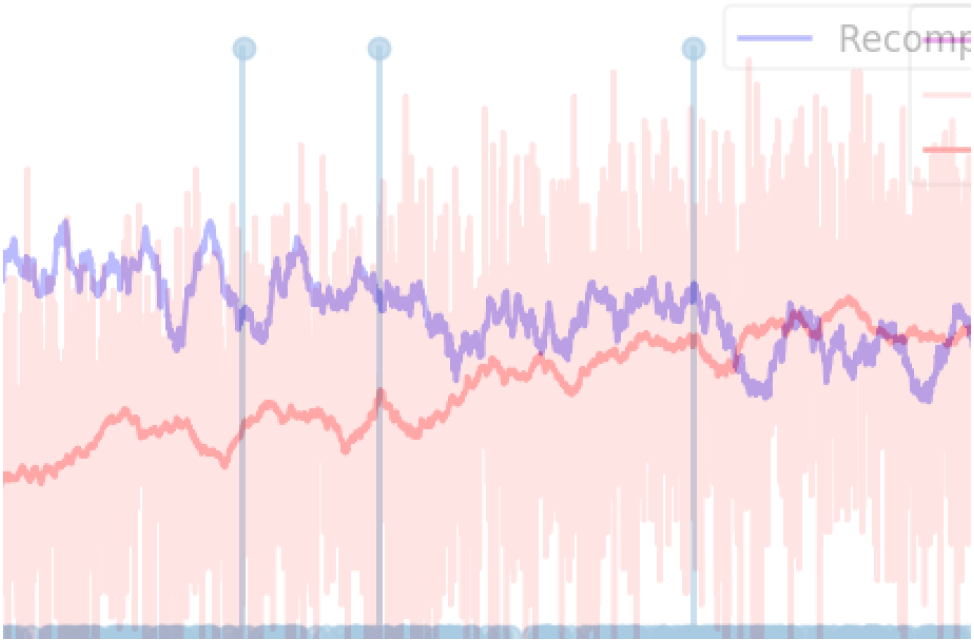

Figura 9: Superposición de gráficas

6.7.3. Mejora del modelo

A lo largo del desarrollo del proyecto se ha experimentado con varias configuraciones tanto de recompensas como de la configuración base con el objetivo de guiar al agente hacia un comportamiento más avanzado. Aún así el agente, aunque haya podido progresar, no lo ha hecho mucho más que los resultados mostrados en el apartado anterior de análisis.

Este comportamiento sugiere que la configuración actual, la base del código o la potencia computacional no logran proporcionar lo necesario para que el agente aprenda correctamente en el entorno. Evaluando los resultados es posible que los siguientes factores estén limitando este objetivo:

- Un entorno demasiado complejo con mecánicas que no terminan de encajar como deben.
- La exploración del espacio de estados puede no estar a la altura, ya sea por la política ϵ-greedy como por el número limitado de episodios, debido a restricciones de recursos computacionales.
- El modelo de entrenamiento con el que se trabaja puede presentar ciertas limitaciones en entornos con una escala tan alta de opciones.

- La configuración que se esta empleando carece de algún matiz que esté provocando este escenario.

Por tanto, aunque el comportamiento final del agente no haya alcanzado todos los objetivos propuestos en un principio, los resultados obtenidos siguen siendo valiosos para al proyecto y la puesta en práctica de un modelo de RL que ahora ofrece la oportunidad de mejorar tanto las mecánicas del videojuego como el modelo de aprendizaje.

7

Conclusiones y trabajos futuros

7.1. Objetivos cumplidos

A pesar de no lograr todos los objetivos establecidos, se han alcanzado varios de ellos, sirviendo así de base para futuras mejoras del proyecto. Entre los objetivos cabe destacar:

- Diseño y construcción de un entorno de simulación con el que poder trabajar e integrar el modelo de aprendizaje sin ningún tipo de problema.
- Presentar un entorno con mecánicas y características del género de gestión por recursos con mecánicas que tienen en cuenta los matices del RL.
- La capacidad del agente de aprender comportamientos, que aunque básicos, siguen formando parte de un avance crucial en el objetivo que tiene este proyecto, que es el de aprender e investigar sobre el RL.
- Validación de un sistema general que consigue funcionar sin problemas y ofrecer información útil sobre los resultados del agente.
- Replicación del entrenamiento de forma gráfica para poder observar como se comporta el agente en tiempo real.

7.1.1. Análisis de posibles complicaciones

Aunque el proyecto presenta un matiz muy interesante del RL aplicado a videojuegos, en este caso uno de gestión por recursos, es posible que el acercamiento se podría haber enfocado con un escenario más sencillo:

- Simplificar las mecánicas de gestión por recursos eliminando la variedad de cultivos y las estaciones.
- Ofrecer al agente un mapa que además de constante y permanente, que estuviera diseñado de forma meticulosa para forzar al agente a aprender ciertas rutinas o flujos de trabajo.
- Reducir el tamaño del mapa para entrenar al agente en un espacio de estados y de casillas menor.

7.1.2. Posibles mejoras y trabajos futuros

Los resultados obtenidos, aunque parciales, ofrecen una base sólida sobre la que plantearse mejoras y cambios que mejoren el rendimiento del agente como por ejemplo:

- Poner en práctica los cambios comentados en el apartado anterior.
- Introducir en el proyecto algoritmos más avanzados como DQN, un RL reforzado o el uso de redes convolucionales para representar el entorno desde otro punto de vista.
- Aumento de la capacidad computacional y de los recursos para lograr una mayor duración del entrenamiento y del número de episodios.
- Ajustes a las recompensas que se otorgan al agente, creando refuerzos positivos que se relacionen con estados intermedios para guiarlo más fácilmente.
- Diseño de una gama de mapas y entornos controlados en los que ir complicando el entrenamiento de forma gradual.
- Rediseñar las mecánicas del videojuego tanto por simplificarlas, como por crear sinergias e interconexiones más robustas que provoquen un curso de acción más exacto para lograr los objetivos.

7.2. Conclusión final

En resumen, el proyecto ha permitido explorar y estudiar de forma práctica la aplicación del RL en un entorno simulado que contaba con factores múltiples y conectados entre sí. Aunque el agente no ha alcanzado el nivel de comportamiento esperado, los resultados obtenidos

son igualmente valiosos: evidencian los retos actuales de este tipo de enfoques y sientan las bases para nuevos proyectos y mejoras.

Este trabajo ha permitido comprender mejor cómo diseñar entornos interactivos para agentes autónomos, ofreciendo una visión realista de los retos técnicos y conceptuales que implica aplicar RL en entornos concretos, además de la creación y conceptualización de estos para su aplicación al RL.

Referencias

[1] Dario Amodei et al. "Concrete problems in AI safety". En: *arXiv preprint arXiv:1606.06565* (2016).

[2] Barto Andrew y Sutton Richard S. "Reinforcement learning: an introduction". En: (2018).

[3] Kai Arulkumaran et al. "A brief survey of deep reinforcement learning". En: *arXiv preprint arXiv:1708.05866* (2017).

[4] Adrià Puigdomènech Badia et al. "Agent57: Outperforming the atari human benchmark". En: *International conference on machine learning*. PMLR. 2020, págs. 507-517.

[5] Bowen Baker et al. "Video pretraining (vpt): Learning to act by watching unlabeled online videos". En: *Advances in Neural Information Processing Systems* 35 (2022), págs. 24639-24654.

[6] Marc G Bellemare et al. "The arcade learning environment: An evaluation platform for general agents". En: *Journal of artificial intelligence research* 47 (2013), págs. 253-279.

[7] Richard Bellman. "A Markovian decision process". En: *Journal of mathematics and mechanics* (1957), págs. 679-684.

[8] Richard Bellman. "Dynamic programming". En: *science* 153.3731 (1966), págs. 34-37.

[9] Ronald A Howard. "Dynamic programming and markov processes." En: (1960).

[10] Tommi Jaakkola, Michael Jordan y Satinder Singh. "Convergence of stochastic iterative dynamic programming algorithms". En: *Advances in neural information processing systems* 6 (1993).

[11] Leslie Pack Kaelbling, Michael L Littman y Andrew W Moore. "Reinforcement learning: A survey". En: *Journal of artificial intelligence research* 4 (1996), págs. 237-285.

[12] Vijay Konda y John Tsitsiklis. "Actor-critic algorithms". En: *Advances in neural information processing systems* 12 (1999).

[13] Biao Luo, Yin Yang y Derong Liu. "Adaptive Q-learning for data-based optimal output regulation with experience replay". En: *IEEE transactions on cybernetics* 48.12 (2018), págs. 3337-3348.

[14] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. En: *nature* 518.7540 (2015), págs. 529-533.

[15] Charles Packer et al. “Assessing generalization in deep reinforcement learning”. En: *arXiv preprint arXiv:1810.12282* (2018).

[16] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[17] Laura Cañete Sanz, Sjors Martens y Teresa de la Hera. “The case of Cities: Skylines versions—Affordances in urban planning education”. En: *Media and Communication* 13 (2025).

[18] John Schulman et al. “Proximal policy optimization algorithms”. En: *arXiv preprint arXiv:1707.06347* (2017).

[19] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. En: *nature* 529.7587 (2016), págs. 484-489.

[20] David Silver et al. “Mastering the game of go without human knowledge”. En: *nature* 550.7676 (2017), págs. 354-359.

[21] Burrhus Frederic Skinner. *Science and human behavior*. 92904. Simon y Schuster, 1965.

[22] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. En: *Advances in neural information processing systems* 12 (1999).

[23] Matthew E Taylor y Peter Stone. “Transfer learning for reinforcement learning domains: A survey.” En: *Journal of Machine Learning Research* 10.7 (2009).

[24] Hado Van Hasselt, Arthur Guez y David Silver. “Deep reinforcement learning with double q-learning”. En: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[25] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. En: *nature* 575.7782 (2019), págs. 350-354.

[26] Christopher JCH Watkins y Peter Dayan. “Q-learning”. En: *Machine learning* 8 (1992), págs. 279-292.

[27] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. En: *Machine learning* 8 (1992), págs. 229-256.

Apéndice A
Manual de Instalación

Para instalar y ejecutar el proyecto correctamente, solo es necesario disponer de la última versión de Python y tener instaladas las siguientes librerías:

- pygame
- numpy
- matplotlib

La instalación de las librerías se puede realizar mediante pip tal que:

```
pip install pygame numpy matplotlib
```

Una vez completada la instalación, el entorno estará listo para ejecutar el proyecto.

Apéndice B
Manual de Usuario

B.1. Entrenamiento del agente

Para entrenar al agente desde cero, se debe ejecutar el archivo `farm_rl_trainer.py`. Este script entrenará al agente dentro del entorno del videojuego y generará un archivo .pkl que almacena el aprendizaje del agente. El comando a ejecutar es el siguiente:

```
python farm_rl_trainer.py
```

B.2. Visualización gráfica del agente y la simulación

Para observar el comportamiento del agente entrenado se debe ejecutar `agent_demo.py`, que cargará el código base del videojuego que se encuentra en el archivo `videogame.py`.

Es necesario asegurar que la ruta del archivo .pkl previamente generado es la correcta. Para revisar la ruta a la que accede el agente para cargar el archivo, basta con mirar la línea número 14 del código. En caso de no estar bien seleccionada la ruta del archivo, es necesario modificarla para la correcta ejecución de la demostración visual.

El comando para ejecutar la demo es:

```
python agent_demo.py
```

UNIVERSIDAD DE MÁLAGA | uma.es

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

E.T.S. DE INGENIERÍA INFORMÁTICA