UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Aprendizaje por refuerzo en logística corporativa: Next Best Action

Trabajo Fin de Grado

Grado en Ciencia de Datos

AUTOR/A: Obrador Reina, Miquel

Tutor/a: Monserrat Aranda, Carlos

Cotutor/a externo: REY MASID, ANXO

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Automatización y Sistemas (Logística Corporativa: Next Best Action)

TRABAJO FIN DE GRADO

Grado en Ciencia de Datos

Autor: Miquel Obrador Reina

Tutor: Carlos Monserrat Aranda
Anxo Rey Masid

Curso 2022-2023

Resum

Al camp de l'aprenentatge per reforç es busca entrenar agents intel·ligents perquè aprenguin a prendre decisions òptimes en situacions complexes a través de la interacció amb un ambient. En aquest treball realitzat juntament amb Inditex, l'agent será un robot que s'encarregui de la logística en un magatzem, específicament la retirada i l'entrada de caixes de forma automàtica en prestatgeries. A mesura que es fan diferents iteracions del projecte, augmenta la complexitat de l'entorn i del problema a resoldre per a l'agent. S'estudiaran el possible ús de diferents arquitectures de xarxa neuronal (Xarxes Neuronals Artificials i Xarxes Neuronals Recurrents) i tècniques d'entrenament (per exemple, Deep Q-Learning, Actor-Critic i Policy Gradient) per seleccionar la millor opció per a cada escenari. A més, heu de seleccionar correctament els hiperparàmetres, com el factor de descompte i la taxa d'aprenentatge, en l'exercici del model. S'espera que l'agent aconsegueixi aprendre a fer la logística del magatzem de forma eficient i efectiva.

Paraules clau: aprenentatge per reforç, xarxes neuronals, magatzems, logística, optimització

Resumen

En el campo del aprendizaje por refuerzo se busca entrenar agentes inteligentes para que aprendan a tomar decisiones óptimas en situaciones complejas a través de la interacción con un ambiente. En este trabajo realizado juntamente con Inditex, el agente será un robot que se encargue de la logística en un almacén, específicamente la retirada y entrada de cajas de forma automática en estanterías. A medida que se realizan distintas iteraciones del proyecto, se aumenta la complejidad del entorno y del problema a resolver para el agente. Se estudiarán el posible uso de diferentes arquitecturas de redes neuronales (Redes Neuronales Artificiales y Redes Neuronales Recurrentes) y técnicas de entrenamiento (por ejemplo, Deep Q-Learning, Actor-Critic y Policy Gradient) para seleccionar la mejor opción para cada escenario. Además de seleccionar correctamente los hiperparámetros, como el factor de descuento y la tasa de aprendizaje, para el mejor desempeño del modelo. Se espera que el agente logre aprender a realizar la logística del almacén de manera eficiente y efectiva.

Palabras clave: aprendizaje por refuerzo, redes neuronales, almacenes, logística, optimización

Abstract

In the field of reinforcement learning, the aim is to train intelligent agents to learn to make optimal decisions in complex situations through interaction with an environment. In this work carried out jointly with Inditex, the agent will be a robot in charge of logistics in a warehouse, specifically the automatic removal and entry of boxes on shelves. As different iterations of the project are carried out, the complexity of the environment and the problem to be solved by the agent increases. The possible use of different neural network architectures (Artificial Neural Networks and Recurrent Neural Networks) and training techniques (e.g. Deep Q-Learning, Actor-Critic and Policy Gradient) will be studied to select the best option for each scenario. In addition to correctly selecting the hyperparameters, such as discount factor and learning rate, for the best model performance. The agent is expected to learn how to perform warehouse logistics efficiently and effectively.

Key words: reinforcement learning, neural networks, warehouse, logistics, optimization

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Preámbulo

El uso de robots para la automatización de procesos logísticos es una tendencia cada vez más presente en la industria y la tecnología. Estos robots son capaces de realizar tareas de carga, descarga y traslado de objetos, lo que permite una mayor eficiencia en los procesos de almacenamiento y distribución. Sin embargo, la programación de estos robots para realizar tareas específicas puede ser un desafío, especialmente en entornos cambiantes y con múltiples restricciones. El aprendizaje por refuerzo (RF, por sus siglas en inglés reinforcement learning) se presenta como una herramienta prometedora para abordar este desafío, permitiendo que los robots aprendan a través de la interacción con el entorno y la retroalimentación que reciben de él. En particular, el uso de redes neuronales en el aprendizaje por refuerzo ha mostrado resultados significativos en la resolución de problemas complejos, como la planificación y ejecución de tareas logísticas en almacenes.

En este contexto, este trabajo se enfoca en el entrenamiento de un agente robótico para la logística de un almacén, pero el aprendizaje por refuerzo se utiliza en numerosas áreas y con distintos fines. Tal vez las empresas más reconocidas por usarlo sean OpenAI y DeepMind, algunos ejemplos de modelos usados por estas empresas para demostrar el potencial del RF son entrenar agentes para que jueguen a distintos juegos de mesa o videojuegos, lo cual parecerá una tontería, pero hay que pensar que este tipo de entornos cambian constantemente y pueden llegar a ser muy complicados incluso para un humano.

Por parte de DeepMind, ya en 2013 lanzaron un artículo "Playing Atari with Deep Reinforcement Learning" [8] donde usaban redes neuronales convolucionales y Deep Q-Learning para jugar a muchos juegos de Atari, algunos a nivel sobrehumano. En 2017 revolucionaron el mundo de la IA con AlphaZero que en tan solo 24 horas de entrenamiento logró alcanzar un nivel de juegos sobrehumano en ajedrez, shogi y Go al derrotar a Stockfish fácilmente (uno de los mejores motores de juego, pero que funciona básicamente por "fuerza bruta") e incluso a campeones del mundo.

En 2019 otro increíble modelo fue presentado, AlphaStar [21], este era capaz de jugar al juego "StarCraft II" de forma sumamente superior a cualquier humano. Este es un videojuego muy complejo de un jugador contra uno o más jugadores en tiempo real. Donde el objetivo es construir un base militar gestionando recursos, investigación, construcciones, etc. Para protegerse de ataques enemigos a la vez que intentas destruir la base enemiga. Tanto AlphaZero como AlphaStar fueron entrenados jugando contra sí mismos, por lo tanto, aprendieron a jugar sin ningún feedback humano.

Trabajos más recientes de DeepMind son AlphaTensor publicado en 2022, este modelo "juega" a encontrar algoritmos más eficientes para multiplicar tensores, es un ejemplo interesante de cómo podemos aplicar aprendizaje por refuerzo para resolver problemas complejos.

Por parte de OpenAI en 2019 y 2020 sacaron dos sistemas multi-agente de aprendizaje por refuerzo, donde los agentes interactuaban y se ayudaban entre ellos para conseguir un objetivo en común. En el primer paper "Emergent tool use from multi-agent autocurricula" [2] el objetivo de los agentes era jugar al "Hide-and-Seek" o al "Escondite". Los agentes se dividían en dos roles, escondidos y buscadores, como podemos imaginar el objetivo de los buscadores era encontrar a los escondidos y viceversa. Los agentes tenían a su disposición elementos en el entorno que podían agarrar, como rampas, paredes y bloques para bloquear a los buscadores, etc.

El otro sistema es OpenAI Five [3] un modelo que era capaz de jugar al juego "Dota 2" de forma profesional, "Dota 2" es un juego de estrategia de dificultad similar al ajedrez o al antes mencionado "StarCraft II" donde el objetivo es destruir la base rival. En este caso es de 5 jugadores contra 5 jugadores, los cuales deben cooperar y seguir una estrategia sólida para conseguir la victoria. Existe también la dificultad añadida de los distintos personajes que se pueden usar por los jugadores, siendo unos personajes mejores contra otros.

Finalmente, destacar el uso de aprendizaje por refuerzo mediante human-feedback usado para entrenar al famoso ChatGPT, OpenAI uso su enorme modelo del lenguaje preentrenado de forma no supervisada GTP3.5 y le hizo un fine-tuning con RF de human feedback [15] para transformarlo en este potente chatbot del que todo el mundo habla.

1.2 Introducción

Una vez completado el preámbulo y después de examinar algunos de los ejemplos más relevantes de la tecnología que se estudiará en este trabajo, procederemos a explicar el propósito de esta investigación.

En los amplios almacenes de las grandes empresas, diariamente ingresan y salen numerosos camiones cargados de mercancías. Es fundamental gestionar de manera eficiente el tiempo de carga y descarga de estos camiones, ya que cualquier retraso en esta tarea puede suponer pérdidas significativas para la empresa. Obviamente, resulta imposible controlar todo el almacén (incluyendo las entradas y salidas) únicamente con trabajadores humanos. Por eso, estas instalaciones tienen sistemas automatizados de almacenamiento y recuperación (ASRS por sus siglas en inglés) para gestionar eficientemente los flujos de entradas y salidas. Estos sistemas se basan en cintas transportadoras, clasificadores y robots que se encargan de la logística. No obstante, todos los robots deben ser controlados mediante software, frecuentemente algoritmos clásicos, los cuales procesan las cajas previamente programadas por un operario, o se recogen las cajas siguiendo una dinámica similar.

Por lo tanto, el objetivo de este trabajo es abordar el desafío de gestionar la demanda de entradas y salidas de cajas mediante el uso de Deep Learning y aprendizaje por refuerzo. Si bien este trabajo representa una versión simplificada de la realidad, se espera que pueda servir como punto de partida para aplicaciones en la industria real.

1.3 Objetivo

Ahora centrándonos más en el objetivo de este trabajo. El problema principal para abordar este problema consiste en un agente entrenado con aprendizaje por refuerzo. El cual deberá cumplir las demandas que se le impongan de forma completamente autónoma y eficiente en un entorno virtual similar a la realidad. Empezaremos con un entorno simple y con una demanda sencilla y a medida que vayamos avanzando el trabajo el agente se tendrá que enfrentar a demandas más exigentes en entornos más complejos.

1.4 Motivación

Las ganas de aprender cosas nuevas, sobre todo de redes neuronales por todo lo que son capaces de hacer y la gran revolución que han supuesto para el machine learning. Aprender sobre reinforcement learning, un campo que había oído hablar de él, pero del cual desconocía completamente su modus operandi. Todo esto unido a la beca y propuesta de trabajo por parte de Inditex me ha dado la posibilidad de aprender y aplicar deep learning y aprendizaje por refuerzo en la tarea de la gestión logística de un almacén y aprender distintos algoritmos que se usan en las industrias para optimizar las funciones de perdida de estos modelos y formas de codificación del espacio observable por el agente.

1.5 Estructura de la memoria

La memoria se estructura del siguiente modo:

- **Capítulo 1, Introducción**: En este capítulo se habla de la motivación, los objetivos y la estructura de la propia memoria.
- **Capítulo 2, Estado del arte**: Aquí revisamos las principales técnicas usadas en aprendizaje por refuerzo y como se llevan a cabo.
- **Capítulo 3, Análisis del problema**: En este apartado revisaremos el problema y las razones por las cuales desarrollar el trabajo.
- **Capítulo 4, Creación del entorno**: Se explicará la creación y el porqué del entorno (entorno) donde el agente trabajará
- **Capítulo 5, Espacio de acción y espacio de observación**: Aquí se explicará que información recibirá el agente de su entorno y que acciones es capaz de tomar.
- **Capítulo 6, Interacción con el entorno**: Este capítulo se centrará en como el agente interactúa con el entorno y qué feedback/recompensa recibe de este.
- **Capítulo 7, Modelo y entrenamiento**: Se explicará el modelo y usado en este trabajo, su arquitectura y como se entrenará.
- **Capítulo 8, Resultados**: Se explicarán los resultados obtenidos en cada uno de los objetivos propuestos, esto incluirá resultados de entrenamiento y resultados de testeo.
- **Capítulo 9, Conclusiones**: En este último capítulo, detallamos los objetivos alcanzados. Además, hablamos de la relación de este proyecto con el Grado en Ciencia

de Datos. Por último, discutimos cómo se podría mejorar este proyecto con investigación.

CAPÍTULO 2
Estado del arte

En este capítulo, se realizará una revisión de unos conceptos básicos que serán útiles para el lector. A continuación de esto se explicará que es y como funciona el aprendizaje por refuerzo (RF de ahora en adelante) seguido de una evolución temporal del estado del arte en algoritmos de RF desde 2013 hasta 2017. Seguido de esto se explicará un poco la particularidad del problema que se va a resolver. Y finalmente se abordarán distintas técnicas que se utilizan en los distintos experimentos que se realizan en el trabajo.

2.1 Aprendizaje por refuerzo

Es una técnica de aprendizaje automático (machine learning) que permite que un agente, en nuestro caso una inteligencia artificial, aprenda a tomar decisiones a través de la interacción y recopilación de datos con su entorno. El objetivo principal es que el agente aprenda a maximizar una recompensa o minimizar una penalización a lo largo del tiempo (véase la Ilustración 2.1)

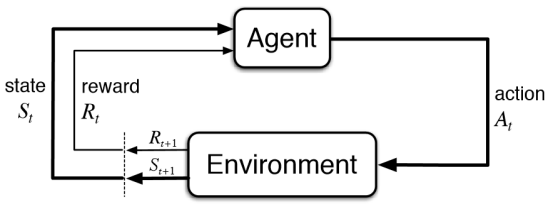

Figura 2.1: Funcionamiento del Reinforcement Learning

En el reinforcement learning hay algunos conceptos claves o nomenclaturas utilizadas en este campo que debemos conocer, estas son:

1. **Agente**: mencionado anteriormente, es el aprendiz o la entidad debe aprender a realizar la tarea propuesta
2. **Entorno**: mundo físico o simulado donde el agente opera y toma decisiones
3. **Estado**: situación actual del entorno y el agente
4. **Recompensa**: es la señal numérica de feedback que el agente recibe del entorno, esto le indica al agente como de bien o mal lo está haciendo.
5. **Política**: es la estrategia que el agente utilizará para tomar las decisiones

6. **Valor o función valor**: es una función que estima la recompensa acumulada que el agente puede obtener a largo plazo a partir de un estado dado y una política dada.

Existen numerosas políticas, *Random Policy*, donde el agente simplemente toma decisiones aleatorias, hasta unas más refinadas y complejas como redes neuronales que lleven a cabo las decisiones. En este trabajo se van a utilizar redes neuronales como política de nuestro agente gracias a su versatilidad y potencia.

2.2 Redes neuronales y optimización de parámetros

Primero debemos entender que es una red neuronal, la cual se puede ver como básicamente un aproximador universal de funciones [4]. Esta se compone de muchos regresores lineales unidos entre sí por distintas capas, y aplicando en cada uno de ellos distintas funciones de activación no lineales, nos permite la aproximación de funciones complejas. Esto les permite identificar patrones e información compleja y abstracta, siendo usual que en las primeras capas se detecten rasgos generales de los datos y en las capas más profundas detalles más específicos. Por ejemplo, en el caso de la imagen de un perro, en las primeras capas la red aprenderían que son líneas rectas y horizontales y en las últimas capas características como que es un ojo, la nariz, etc. [1]

Existen multitud de tipos o arquitecturas de redes neuronales donde las más destacables serían:

1. Redes neuronales multicapa (MLP del inglés *Multi Layer Perceptron*) [13] Son las redes neuronales más simples después del perceptrón (una única neurona) donde todas las neuronas de una capa se conectan con todas las neuronas de la capa anterior y de la capa posterior, este tipo de redes no pueden ser muy profundas ni muy grandes, ya que pierden el gradiente muy rápido. Hoy en día estas redes se suelen utilizar como las últimas capas de la red para llevar a cabo de clasificación o regresión, y por encima se utilizan otro tipo de redes.

2. Redes Convolucionales (CNN del inglés *Convolutional Neural Network*) [6] Estas redes se suelen utilizar para procesar imágenes, su funcionamiento es aplicar convoluciones discretas con filtros cuyos valores son automáticamente aprendidos, para que cada filtro extraiga y codifique cierta información espacial de la imagen.

3. Redes Recurrentes (RNN del inglés *Recurrent Neural Network*)[18] Estas redes se utilizan en un principio para procesar secuencias de datos (series temporales) o lenguaje natural, esto es porque estas redes tiene capacidad de retener información de entradas anteriores, por ejemplo, en el lenguaje natural son útiles para obtener información sobre el orden de las palabras y el contexto entre ellas (estas fueron más tarde sustituidas por los transformers[20]) pero hoy en día son útiles en series temporales o en reinforcement learning al tener codificada información de estados y acciones anteriores realizadas por el agente.

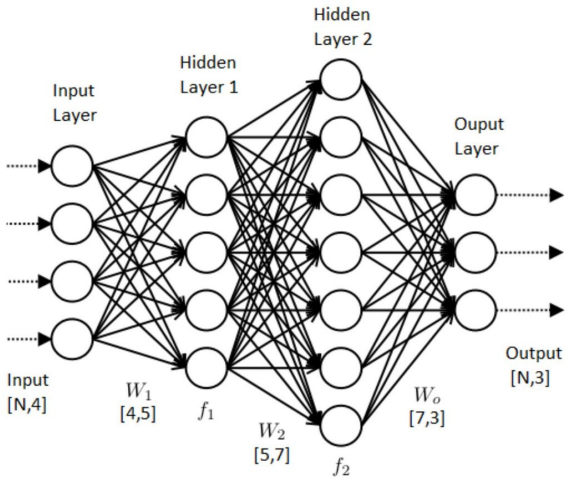

Figura 2.2: Red neuronal multicapa

En la figura 2.2 se muestra un ejemplo de una red neuronal de múltiples capas. La capa de entrada recibe los valores, en este caso en concreto tenemos 4 neuronas de entrada, por lo tanto, 4 valores. El resultado de cada neurona se envía a las neuronas de la capa siguiente, donde cada una actúa como un regresor lineal al que se le aplica una función de activación no lineal. Una de las funciones de activación más usada es la ReLu (transforma los valores negativos en 0 y los demás los deja como están) por su simplicidad, mayor capacidad de representación de valores positivos y porque mitiga el problema de desvanecimiento de gradiente. Pero otras funciones como la tangente hiperbólica (que acota los valores entre -1 y 1) o la sigmoide (entre 0 y 1) pueden ser usadas.

Una vez se ha elegido el modelo que utilizaremos tenemos que optimizar los parámetros de la red, para esto se debe seleccionar una función de pérdida la cual será maximizada o minimizada (lo veremos en la siguiente sección, ya que en nuestro caso las funciones de pérdida serán distintas a las típicas usadas en aprendizaje supervisado como pueda ser *Binary Cross-entropy* o *Mean Squared Error*). Una variable muy importante en el optimizador será el learning rate, donde valores muy altos provocará, saltos muy grandes en los parámetros de la red y falta de convergencia y valores bajos pueden o ralentizar mucho la convergencia o acabar en un mínimo local. Para optimizar esta función existen distintos optimizadores como ADAM, SGD (Descenso por gradiente estocástico) con momentum o sin o RSMProp, cada uno tiene sus ventajas y desventajas.

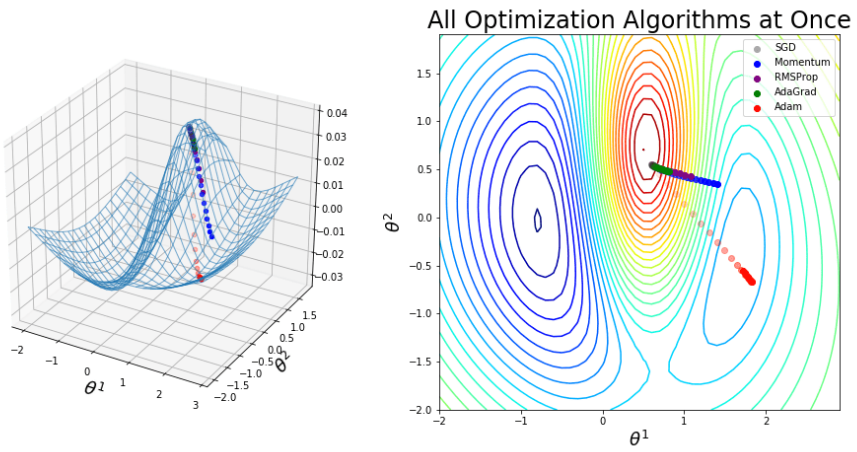

Figura 2.3: Optimización función pérdida

En la figura 2.3 podemos ver como los distintos algoritmos buscan el mínimo en una función a optimizar, las diferencias suelen ser en la velocidad que tardan en converger y la tendencia a encontrar mínimos mejores que otros. Por ejemplo, normalmente SGD tarda considerables iteraciones a encontrar un mínimo y este suele ser local. Nosotros usaremos ADAM, como podemos ver la gráfica es el que encuentra el mejor mínimo, pero recordemos que esto no significa que sea el mejor. ADAM básicamente es una combinación de SGD con momento y RMSProp, combina la idea de ir conservando la dirección (momento) del gradiente a través de las iteraciones y de RMSProp de usar los gradientes más recientes aplicando una media exponencial sobre estos para suavizar los cambios a los parámetros. Todo esto hace que ADAM tenga unos muy buenos resultados en general, por su velocidad en convergencia y la calidad de sus mínimos encontrados.

2.3 Métodos para entrenar Políticas

Para entrenar las redes neuronales (política de ahora en adelante), en el campo del aprendizaje por refuerzo existen diversos métodos, lo más típico es combinar los métodos basados en valor (*Q-Leaning*) o métodos basados la política (*Policy Gradient*).

2.3.1. Métodos basados en valor

Los métodos basados únicamente en el valor son un enfoque en el aprendizaje por refuerzo que se centra en estimar y mejorar directamente la función de valor del estado o estado-acción, sin tener en cuenta una política explícita (*off-policy*). Estos no te dicen directamente qué acción tomar, sino que te indican la recompensa esperada desde cada estado o estado y acción seleccionados.

Uno de los más famosos es el método Q-Learning, este es un algoritmo basado en tablas (tabular) que estima la función Q-value (valor de estado-acción) para cada par estado-acción. El objetivo del Q-Learning es aprender una política óptima al actualizar iterativamente los valores Q utilizando la ecuación de Bellman (Véase en la Figura 2.4), que combina la recompensa inmediata y el mejor valor Q del estado siguiente. [16]

$$Q(s,a) = Q(s,a) + \alpha * (r + \gamma * max(Q(s',a')) - Q(s,a))$$

Figura 2.4: Equación de Bellman[16]

Donde α es la tasa de aprendizaje y γ es el factor de descuento que determina la importancia relativa de las recompensas futuras. A través de la exploración y explotación, el algoritmo Q-Learning encuentra una política que maximiza los valores Q y, por lo tanto, maximiza la recompensa acumulada esperada.

El uso de redes neuronales en el Q-Learning permite manejar problemas más complejos con espacios de estados y acciones grandes o continuos. Sin embargo, también introduce desafíos adicionales, como la estabilidad y la convergencia del entrenamiento, que se abordan mediante técnicas como el uso de objetivos fijos y la experiencia de repetición (replay buffer).

Algunos aspectos clave del uso de redes neuronales en el Q-Learning:

1. **Experiencia de repetición (Replay Buffer)**:

   Durante las interacciones del agente con el entorno, se almacenan las transiciones de experiencia en un búfer de repetición (replay buffer). En cada iteración de entrenamiento, se muestrean lotes de transiciones de experiencia del replay buffer para

entrenar la red neuronal. El uso de experiencia de repetición ayuda a mitigar la correlación temporal y mejorar la estabilidad del entrenamiento.

2. **Red neuronal como aproximador de la función Q**: La red neuronal toma el estado como entrada y produce una salida para cada posible acción. Durante el entrenamiento, se ajustan los pesos y los sesgos de la red utilizando técnicas de optimización, como el descenso de gradiente, para minimizar la diferencia entre los valores Q estimados y los valores Q reales. El objetivo es que la red neuronal aprenda una representación de la función Q que generalice bien y pueda estimar los valores Q para nuevos estados y acciones.

3. **Exploración y explotación**: Durante el entrenamiento, se utiliza una política de exploración y explotación para equilibrar la exploración del entorno y la elección de acciones basada en la estimación actual de los valores Q. Una política ϵ-greedy es comúnmente utilizada, donde se selecciona una acción aleatoria con probabilidad ϵ y la mejor acción estimada con probabilidad (1-ϵ).

2.3.2. Métodos basados en política

Los métodos basados en la política se centran solamente en el agente y su política (*on-policy*) es decir, no estamos estimando ninguna función valor $V(s)$ o $Q(s,a)$ sino, una función política $\pi(a|s)$ que estima una probabilidad de tomar cada acción posible desde un estado dado. La forma en que se entrenan estas redes es distinta a como se hace con las de valor. [10] [16].

En primer lugar, el tensor output de la red neuronal pasará a ser una función de distribución de probabilidades para cada una de las distintas opciones, aplicando la función de activación softmax a la última capa de salida obtendremos un vector donde la suma total de todos sus elementos sea 1. La acción escogida por el agente será un muestreo aleatorio de esa distribución de acciones usando las probabilidades de la salida de la red como pesos de la distribución. Al principio la red nos dará valores completamente aleatorios para cada acción, teniendo así un comportamiento "aleatorio" o en estado de exploración, a medida que la red sea entrenada el peso de distintas acciones será menor que otras, propiciando la salida de estas con mayor peso, es decir, la política se centra más en explotación que en exploración, todo esto dependerá del bucle de entrenamiento y de las condiciones de partida.

Para entrenar la red debemos computar nuestra función de pérdida. Hay distintas formas de hacerlo, pero la más simple es, primeramente, aplicar la función log a las probabilidades de salida de la red, una vez las tenemos creamos una máscara para quedarnos solo con el log de la probabilidad de la acción usada en ese paso, por último nuestra pérdida final será la multiplicación de este vector de máscara por el retorno, es decir las recompensas que llevamos acumulados, todo esto obviamente se hace en lotes, normalmente las librerías tensoriales se encargan de gestionar eso fácilmente. Un ejemplo explicado con código de lo anteriormente explicado se puede encontrar aquí [10]

Aunque utilizamos el término "función de pérdida" para describirla, esta función difiere significativamente de las funciones de pérdida típicas utilizadas en el aprendizaje supervisado. Hay dos distinciones principales con respecto a las funciones de pérdida estándar:

1. La distribución de los datos depende de los parámetros. En los entornos tradicionales, una función de pérdida se define basándose en una distribución de datos fija que es independiente de los parámetros que se están optimizando. Sin embargo,

en este caso, los datos deben muestrearse utilizando los parámetros de política más actualizados.

2. No mide directamente el rendimiento. Una función de pérdida convencional suele evaluar la métrica de rendimiento que interesa. Sin embargo, en este contexto, nuestra función de "pérdida" no se aproxima al rendimiento esperado, $J(\pi_\theta)$, ni siquiera en promedio. Su única utilidad reside en el hecho de que, cuando se evalúa con los datos generados por los parámetros actuales, proporciona el gradiente negativo de rendimiento.

Una vez completado el paso inicial del descenso de gradiente, ya no existe ninguna conexión con el rendimiento. En consecuencia, minimizar esta función de "pérdida", dado un lote de datos, no garantiza ninguna mejora del rendimiento esperado. Es posible llevar esta pérdida a infinito negativo, pero el rendimiento de la política podría disminuir gravemente; de hecho, esto es lo que ocurre a menudo. A veces, un investigador de RL profunda puede referirse a este resultado como el "sobre ajuste" de la política a un lote de datos. Aunque este término es descriptivo, no debe interpretarse literalmente, ya que no se refiere al error de generalización.

2.3.3. Métodos Actor-Crític

Finalmente, los Actor-Crític methods, mencionados anteriormente, como podemos pensar, combinan elementos del enfoque basado en valor (value-based) y del enfoque basado en política (policy-based) para aprender y mejorar un agente que interactúa con un entorno.

En los métodos de Actor-Critic, hay dos componentes principales:

1. **El Actor**: El actor es responsable de seleccionar acciones basándose en el estado actual del entorno. Puede verse como una política que toma decisiones. El objetivo del actor es encontrar la política óptima que maximice la recompensa esperada a largo plazo.

2. **El Crítico**: El crítico es responsable de evaluar la calidad de las acciones tomadas por el actor. Su objetivo es aprender una función de valor que estime la recompensa esperada a largo plazo de estar en un estado determinado y seguir una política dada.

El actor y el crítico en nuestro caso serán dos redes neuronales, las cuales no tienen por qué ser iguales, pero normalmente suelen serlo, la única diferencia es que el actor, su capa de salida, serán tantas neuronas como acciones posibles (con una función de activación softmax) y el crítico siempre una única neurona que será la recompensa esperada (función activación linear). Actor y crítico interactúan entre sí en un proceso iterativo de esta manera.

1. El agente interactúa con el entorno al seleccionar una acción en función del estado actual utilizando el actor.

2. La acción se ejecuta en el entorno y se observa la recompensa y el próximo estado.

3. El crítico utiliza esta información para evaluar la calidad de la acción tomada. Calcula la recompensa esperada a largo plazo utilizando una función de valor, como la función de valor de estado-acción (Q-value), y actualiza sus parámetros para mejorar la precisión de sus estimaciones.

4. El actor utiliza la evaluación proporcionada por el crítico para mejorar su política. Esto se puede hacer mediante métodos como el gradiente de política (policy gradient), donde se calcula el gradiente de la política con respecto a los parámetros del actor y se actualizan los parámetros para maximizar la recompensa esperada.

5. Se repiten los pasos anteriores para recopilar más datos de interacción, mejorar la evaluación del crítico y ajustar la política del actor.

Normalmente, para maximizar aún más el entrenamiento de estas redes, la función de perdida usada es la combinación de las funciones de perdida del actor y el crítico $L = L_{actor} + L_{critic}$ [17] L_{actor} se define como:

$$L_{actor} = -\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t)[G(s_t, a_t) - V_\theta^\pi(s_t)]$$

Figura 2.5: Función de pérdida del actor [17]

donde:

- T: el número de pasos de tiempo por episodio, que puede variar según el episodio
- s_t: el estado en el paso de tiempo t
- a_t: acción elegida en el paso de tiempo t
- π_θ: es la política (Actor) parametrizado por θ
- V_θ^π: es la función valor (Crítico) también parametrizado por θ
- $G = G_t$: el rendimiento esperado para un par de estado y acción en un momento (t) dado

Se añade un término negativo a la suma, ya que la idea es maximizar las probabilidades de que las acciones produzcan mayores recompensas, minimizando la pérdida combinada.

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

Figura 2.6: Retorno esperado [17]

El retorno esperado o G_t (Figura 2.6) se encarga de transformar la secuencia de recompensas de cada paso de tiempo t, $\{r_t\}_{t=1}^{T}$ obtenidas en un episodio, en una secuencia de retorno esperado, $\{G_t\}_{t=1}^{T}$ donde se suman todos las recompensas obtenidas desde t hasta T cada recompensa siendo multiplicado por un factor de descuento exponencialmente decreciente γ

$\gamma \in (0, 1)$, por lo tanto, las recompensas que están más alejados del paso de tiempo actual tienen menos peso. Intuitivamente, el rendimiento esperado simplemente implica que las recompensas ahora son mejores que las recompensas posteriores. En un sentido matemático, es para asegurar que la suma de las recompensas converja. Para estabilizar el entrenamiento, la secuencia resultante de retornos también está estandarizada (es decir, para tener media cero y desviación estándar unitaria).

El término $G - V$ en L_{actor} se llama **ventaja**[17] e indica cuánto mejor es una acción dado un estado particular sobre una acción aleatoria seleccionada según la política π para ese estado. Este término nos ayuda a reducir la varianza en el entrenamiento, ya que si no

está sería muy alta. Además, al elegir el crítico V tiene un beneficio extra, ya que este está siendo entrenado continuamente para parecerse a G disminuyendo aún más la varianza.

Además, sin el Crítico, el algoritmo intentaría aumentar las probabilidades de las acciones tomadas en un estado concreto basándose en el rendimiento esperado, lo que puede no suponer una gran diferencia si las probabilidades relativas entre las acciones siguen siendo las mismas.

Vamos a decir, por ejemplo, que dos acciones para un estado determinado produjeran el mismo rendimiento esperado. Sin el Crítico, el algoritmo intentaría aumentar la probabilidad de estas acciones basándose en el objetivo J. Con el Crítico, puede resultar que no haya Ventaja ($G - V = 0$) y, por tanto, no se obtendría ningún beneficio aumentando las probabilidades de las acciones, por lo que el algoritmo fijaría los gradientes en cero.

Pasamos ahora a la función de pérdida del crítico L_{critic}:

$$L_{critic} = L_{\delta}(G, V_{\theta}^{\pi})$$

Figura 2.7: Función de pérdida del crítico [17]

donde L_{δ} es la Huber Loss, la cual es menos sensible a los valores atípicos como pueda ser mean squared-error, aunque se pueden usar las dos y dan resultados similares. Un ejemplo con código puede ser encontrado aquí [17]

Esta sería la forma más básica de entrenamiento de agentes con aprendizaje por refuerzo usando métodos de actor y crítico, luego dentro de esta rama hay muchos algoritmos que intentan arreglar los problemas que tienen estos métodos usando distintas herramientas.

Un algoritmo muy famoso podría ser Proximal Policy Optimization (PPO) [14] este algoritmo intenta solucionar el problema de actualización de las políticas, cuando una política es actualizada puede ser que cambie mucho, aún se use un learning rate bajo, esto puede llevar a saltos muy grandes en el gradiente en la función da optimizar y que el algoritmo nunca converja. La idea principal detrás de PPO es actualizar la política de manera incremental y limitar el cambio en cada actualización. Esto se hace mediante la introducción de una "clipping objective" que controla la distancia entre la política actual y la política actualizada. Esto ayuda a evitar actualizaciones drásticas que puedan llevar a la inestabilidad o a un peor rendimiento.

Después de recopilar datos de interacción con el entorno, se utilizan estos datos para calcular la función de pérdida que se utilizará para optimizar la política. Se calcula el valor de probabilidad (surrogate objective) que mide la diferencia entre la probabilidad de acción de la política actual y la probabilidad de acción de la política actualizada. Esto se hace multiplicando las probabilidades de acción de la política actual por las probabilidades de acción de la política actualizada. Se introduce un rango de confianza (clip range) que especifica cuánto se permite que la política actualizada se aleje de la política actual. El rango de confianza es un hiperparámetro que se ajusta según el problema y los requisitos de estabilidad.

Finalmente, se utiliza el clipping objective para limitar la actualización de la política. La actualización de la política se realiza eligiendo el valor mínimo entre el valor de probabilidad y el valor del clipping objective. Si el valor de probabilidad está dentro del rango de confianza, se utiliza el valor de probabilidad sin cambios. Si el valor de probabilidad está fuera del rango de confianza, se utiliza el valor de probabilidad ajustado dentro del rango de confianza (Figura 2.8)

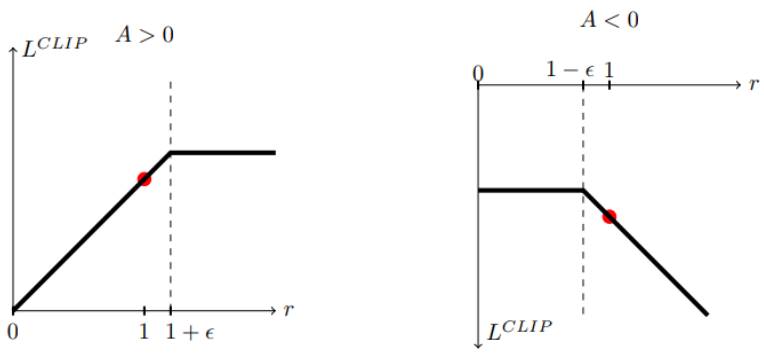

Figura 2.8: L^{CLIP} [14]

2.4 Trabajos relacionados

En esta sección vamos a estudiar los trabajos relacionados con este caso, cualquier trabajo relacionado con aprendizaje por refuerzo nos serviría, ya que normalmente para este tipo de problema no se suelen aplicar algoritmos de machine learning, se suele llevar a cabo con algoritmos tradicionales y normalmente se tienen que adaptar a cada configuración de almacén, lo cual supone un coste enorme en I+D e ingeniera a las empresas. Por otro lado, las que usan estos nuevos algoritmos de Deep Learning en sus almacenes como pueda ser "Amazon" obviamente no revelan como lo hacen, ya que pertenece a su software privado.

Pero a finales de 2022 se llevó publicó un artículo de un estudio sobre esto en la Universidad de Edinburgo el 22 de diciembre de 2022, **Scalable Multi-Agent Reinforcement Learning for Warehouse Logistics with Robotic and Human Co-Workers** [5]. En este trabajo plantean un proyecto que utiliza el aprendizaje por refuerzo con múltiples agentes (MARL) para mejorar la eficiencia y flexibilidad de los sistemas de preparación de pedidos en almacenes comerciales. Este enfoque busca optimizar el rendimiento de la preparación de pedidos al coordinar el movimiento y las acciones de los robots móviles y los operarios humanos en el almacén. El problema central abordado en el proyecto, conocido como el problema de la preparación de pedidos, consiste en cómo coordinar el movimiento y las acciones de los agentes trabajadores para maximizar el rendimiento (por ejemplo, el flujo de pedidos) bajo restricciones de recursos. Los métodos tradicionales utilizados en la industria requieren grandes esfuerzos de ingeniería para optimizar las configuraciones variables de los almacenes, como hemos mencionado anteriormente. En cambio, el enfoque de MARL se puede aplicar de manera flexible a cualquier configuración de almacén (como tamaño, distribución, número/tipo de trabajadores, frecuencia de reposición de artículos), y los agentes aprenden a cooperar óptimamente entre sí a través de un proceso de prueba y error. El artículo detalla el estado actual del esfuerzo de I+D iniciado por Dematic y la Universidad de Edimburgo para desarrollar una solución de MARL escalable y de propósito general para el problema de la preparación de pedidos en almacenes realistas.

En este trabajo se plantea una jerarquía de 3 modelos, los cuales tendrán distintas tareas. El administrador, así como los agentes AGV y los operarios, se modelan utilizando redes neuronales con múltiples cabezas. La red del administrador consta de tres capas completamente conectadas con 128 neuronas en cada capa, tanto la value network como la policy network, de similar forma los agentes AGV y los operarios tienen redes con 2 capas y 64 neuronas. El modelo jerárquico se entrena utilizando el algoritmo Shared Network Actor-Critic (SNAC), en el cual se comparten las redes entre los operarios y

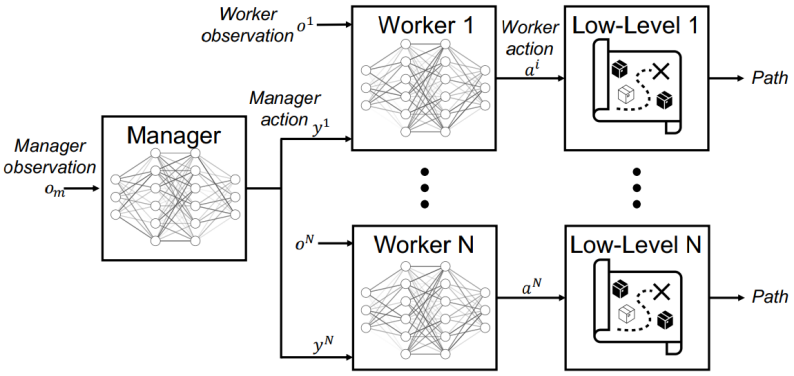

Figura 2.9: Diagram of 3-layer Feudal Hierarchy [5]

los AGVs para mejorar la eficiencia del proceso de entrenamiento. Además, se utilizan técnicas como el descuento temporal (factor γ), la estimación de la ventaja generalizada (GAE, Generalized Advantage Estimate) y la estandarización de estas para mejorar el rendimiento del aprendizaje (Véase en la figura 2.9).

La idea es dividir las ubicaciones dentro del almacén en sectores disjuntos. El administrador observa las ubicaciones actuales y los objetivos de todos los agentes, así como los pedidos actuales asignados a cada AGV (Automated Guided Vehicle, por sus siglas en inglés), y determina un sector al que cada agente debe moverse. Una vez que se ha determinado la ubicación objetivo de cada agente, un controlador de nivel inferior calculará una ruta desde su ubicación actual y ejecutará la secuencia necesaria de acciones primitivas.

Además, se reduce aún más el tamaño del espacio de acciones efectivas de los agentes mediante el enmascaramiento de acciones. Se observa que cada AGV solamente necesita recoger los elementos de su pedido actual, por lo que solo deben moverse a ubicaciones dentro del almacén que contengan estos elementos. Esto se logra enmascarando las acciones que hacen referencia a ubicaciones sin elementos en el pedido actual de un AGV. De manera similar, se aplica un enfoque de enmascaramiento de acciones para los operarios encargados de la preparación de pedidos.

En nuestro trabajo eliminamos esta jerarquía y utilizamos solamente un agente, esto le complica más la tarea al agente, pero hace que se necesite menos preparación previa y este puede llegar a ser más versátil en según que situaciones. Es muy probable que el sistema estudiado en este artículo sea bastante más eficiente que el nuestro por el hecho de la división de tareas.

CAPÍTULO 3
Análisis del problema

El manejo eficiente de las operaciones logísticas en los almacenes es fundamental para optimizar los procesos y garantizar un flujo de trabajo suave. La tarea de gestionar las entradas y salidas de cajas implica rastrear y coordinar el movimiento de los productos almacenados, lo cual puede ser un desafío en entornos con un alto volumen de mercancías.

Algunas razones por las cuales desarrollar un agente de IA para esta tarea pueden ser:

- **Automatización de procesos**: Un agente de IA puede realizar tareas repetitivas y monótonas de manera eficiente y precisa. Puede encargarse de verificar y registrar las entradas y salidas de cajas, evitando la necesidad de intervención humana constante. Esto ayuda a reducir errores y liberar tiempo para que los empleados se enfoquen en actividades más estratégicas. [11][5]
- **Optimización del espacio de almacenamiento**: El agente de IA puede utilizar algoritmos de optimización para determinar la ubicación óptima de las cajas en el almacén. Esto asegura que el espacio se utilice de manera eficiente, maximizando la capacidad de almacenamiento y facilitando la localización y recuperación de productos cuando sea necesario.[12]
- **Gestión de inventario en tiempo real**: Un agente de IA puede llevar un registro preciso de las cajas que ingresan y salen del almacén. Esto proporciona una visión en tiempo real del inventario, lo que facilita la planificación de pedidos, el seguimiento de productos y la identificación de posibles problemas, como faltantes o excesos de inventario.[11][7]
- **Optimización de rutas y procesos de despacho**: Al utilizar técnicas de IA, el agente puede analizar datos históricos y en tiempo real para identificar patrones y tendencias en la demanda y el flujo de productos. Esto permite optimizar las rutas de entrega y los procesos de despacho, reduciendo los tiempos de espera y los costos de envío.[5]
- **Detección de anomalías y problemas de calidad**: El agente de IA puede estar equipado con algoritmos de detección de anomalías que alerten sobre posibles problemas, como cajas dañadas, productos faltantes o errores en el etiquetado. Esto ayuda a garantizar la calidad de los productos y facilita la resolución temprana de problemas.[9]

En resumen, desarrollar un agente de IA para gestionar las entradas y salidas de cajas en grandes almacenes puede proporcionar una serie de beneficios, como la automatiza-

ción de tareas, la optimización del espacio de almacenamiento, una gestión de inventario más precisa, la optimización de rutas y procesos de despacho, y la detección de anomalías. Estas ventajas pueden mejorar la eficiencia operativa, reducir los errores y costos, y ofrecer un mejor servicio a los clientes.

En nuestro trabajo nos vamos a centrar principalmente en el punto de **Automatización de procesos** e intentaremos optimizar el espacio usado en el almacén y rutas seguidas por el agente como objetivos secundarios.

Durante este proyecto nos hemos enfrentado a distintos retos, como crear un espacio simulado controlado similar a la realidad para entrenar a nuestro agente, ya que no podemos hacerlo en la vida real. Representar la información del entorno de forma que el agente puede procesarla y entenderla correctamente. La división del trabajo en distintos subproyectos cada uno más complicado para llegar a la solución final de forma iterativa, en la primera iteración el objetivo del agente será recoger cajas de su entorno y depositarlas en el punto de entrega, en la segunda iteración deberá realizar la colocación de cajas en estanterías (recoger las cajas del punto de recogida y colocarlas), en la tercera iteración su objetivo será las dos anteriores combinadas, se le asignarán unas cajas que deben ser entregadas y otras que deben ser ordenadas, pero si ninguna restricción de en que momento entregar o recoger, hasta ahora las cajas serán todas iguales, en la última iteración deberá trabajar con distintos tipos de cajas.

CAPÍTULO 4
Creación del entorno

En esta sección explicaremos como se crea el entorno en el cual el agente va a interactuar. Así como sus características principales, las cuales cambiarán en cada iteración del trabajo. Veamos primero como se ve el entorno en la vida real (Figura 4.1).

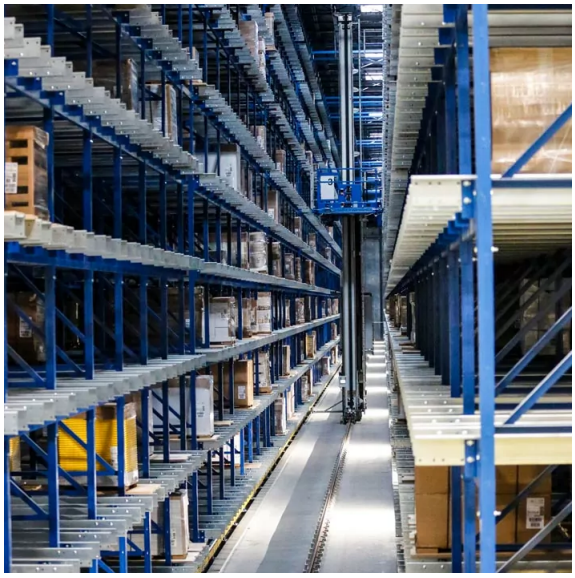

Figura 4.1: Sistema ASRS en la vida real

El entorno tiene que simular el escenario de la figura 4.1, como podemos ver el robot se puede mover en dos ejes y siempre sobre las estanterías (significa que no tiene que evitar obstáculos más allá de salirse de su raíl). Cada robot tiene acceso a dos estanterías y puede llegar a una profundidad de dos casillas en cada una de ellas, además el robot posee espacio para 3 cajas en su inventario. Transformando todo esto en un lenguaje más matemático, cada posible posición en una estantería e dada la podemos definir como (x, y, z), donde x, y es la posición de la casilla en la profundidad z de una estantería e dada, esta puede contener una caja o no.

Para simplificar el problema, por ahora el agente únicamente tendrá acceso a una estantería y la profundidad será de 1 (solo una caja en cada posición x, y), gracias a esta simplificación el entorno se puede representar con una sola matriz de nxn o grid de ahora en adelante (Véase en la figura 4.2) y el robot como una posición aparte x, y ya que este puede moverse libremente encima de esa matriz. Por ejemplo, con cada acción de movimiento, el agente puede moverse una casilla en cualquier dirección.

```
array([[ 1,  2,  1,  1,  3,  2,  0,  2,  1,  1,  3,  3,  0,  3,  2],
       [ 0,  0,  2,  1,  1,  0,  2,  2,  2,  0,  2,  3,  2,  3,  0],
       [ 2,  2,  3,  0,  3,  3,  0,  0,  3,  1,  3,  0,  0,  2,  0],
       [ 0,  1,  0,  3,  3,  2,  2,  3,  2,  0,  2,  2,  0,  0,  2],
       [ 3,  3,  0,  2,  3,  3,  0,  1,  1,  2,  0,  1,  2,  1,  0],
       [ 2,  2,  1,  3,  2,  1,  3,  2,  3,  3,  3,  2,  1,  0,  2],
       [ 2,  1,  3,  1,  1,  1,  3,  3,  2,  3,  0,  3,  3,  1,  2],
       [ 3,  3,  3,  3,  0,  1,  2, -1,  2,  2,  0,  1,  2,  2,  0],
       [ 2,  0,  1,  2,  1,  1,  0,  2,  0,  0,  0,  1,  0,  0,  0],
       [ 0,  0,  2,  2,  0,  3,  0,  3,  0,  3,  0,  3,  3,  2,  3],
       [ 3,  3,  3,  0,  0,  2,  1,  2,  1,  3,  1,  3,  1,  2,  0],
       [ 3,  0,  1,  2,  0,  0,  2,  1,  2,  3,  0,  2,  2,  3,  2],
       [ 1,  2,  0,  2,  0,  1,  0,  0,  1,  3,  2,  3,  1,  3,  1],
       [ 0,  2,  0,  0,  0,  0,  3,  0,  0,  2,  0,  1,  2,  3,  1],
       [ 2,  2,  1,  2,  1,  2,  0,  1,  0,  3,  2,  1,  0,  1,  0]],
      dtype=int8)
```

Figura 4.2: Matriz entorno

Como podemos ver en la figura 4.2 sería un ejemplo de un entorno de un tamaño de 15x15, donde existen distintos posibles valores

- **-1**: simboliza el punto de recogida o entrega de cajas (es el mismo punto).
- **0**: simboliza que el espacio está vacío.
- **1 y superior**: simboliza que él en este espacio hay una caja, y que tipo de caja es (1 es un tipo de caja, 2 es otro tipo de caja, 3 otro ...).

Cuando el entorno es creado se definen dos constantes *pasos para agregar demanda*, esta constante nos dice cada cuantos pasos una nueva orden de demanda va a llegar al agente la cual deberá cumplir y también *max pasos*, significa el número máximo de pasos que el agente puede realizar en el entorno, si el agente cumple con toda la demanda sin llegar a los pasos necesarios para ninguna de estas dos constantes el episodio acaba y es recompensado positivamente.

En el momento que se crea el entorno el grid se inicializa con todo 0 del tamaño deseado, seguidamente se rellena aleatoriamente con cajas (los distintos tipos de cajas que habrá se pueden elegir libremente) dejando espacios vacíos, seguidamente se inicializa la posición del agente de forma aleatoria en el grid cerca del centro y finalmente se coloca el punto de entrega/recogida en el centro.

CAPÍTULO 5
Espacio de acción y espacio de observación

5.1 Espacio de acción

El espacio de acción se define como las posibles acciones que puede tomar el agente, estas pueden ser tanto discretas como continúas, un ejemplo de acciones discretas sería por ejemplo controlar un mando, mientras que un ejemplo de acciones continuas sería flexionar un brazo robótico una cierta cantidad de grados. En nuestro caso nos decantamos por la acción discreta, por las características del entorno y por la facilidad de entrenamiento de estas, ya que si la acción es discreta, nuestra salida de la red neuronal será casi como si fuera un problema de clasificación en el cual tendremos una distribución de probabilidad que sumará 1 (función de activación softmax) para cada posible acción.

El tamaño y la complejidad del espacio de acción pueden influir en el desafío del problema. Si el espacio de acción es grande o continuo, el agente puede necesitar explorar y experimentar más para encontrar las acciones óptimas. Además, el diseño del espacio de acción puede influir en la eficiencia y efectividad del aprendizaje, ya que un espacio de acción bien definido puede facilitar el descubrimiento de políticas de comportamiento eficientes.

En este trabajo usaremos 6 posibles acciones, 4 acciones de movimiento en las cuales puede moverse una casilla en cualquiera de las 4 direcciones (arriba, abajo, derecha o izquierda) y dos acciones adicionales que serían las acciones de "Coger", donde el agente recoge una caja del grid y la coloca en su inventario o "Dejar", donde el agente deja la primera caja de su inventario (su inventario funciona como una cola).

Las acciones de movimiento simplemente serán sumar a la posición del agente un vector unitario en la dirección que deseemos, mientras que las acciones "Dejar" y "Coger" serán gestionadas con condiciones, lo veremos en el siguiente capítulo.

Tabla 5.1: Codificación de acciones

Acción (Índice)	Acción (Codificación)	Descripción
0	np.array([0, 1])	El agente realiza un paso a la derecha
1	np.array([0, -1])	El agente realiza un paso a la izquierda
2	np.array([1, 0])	El agente realiza un paso hacia abajo
3	np.array([-1, 0])	El agente realiza un paso hacia arriba
4	"Coger"	El agente intenta coger una caja de la casilla actual
5	"Dejar"	El agente intenta dejar una caja en la casilla actual

La tabla 5.1 nos enseña como se codifica él espació de acción en el código

5.2 Espacio de observación

El espacio de observación, también conocido como espacio de estado, se refiere al conjunto de todas las posibles observaciones que un agente de aprendizaje puede recibir del entorno en el que se encuentra. Estas observaciones proporcionan información sobre el estado actual del entorno y permiten al agente tomar decisiones sobre las acciones que debe tomar.

Este puede variar en complejidad y tipo de datos que contiene. Puede ser discreto o continuo, dependiendo de si las observaciones se pueden describir mediante un conjunto finito o infinito de valores. Por ejemplo, en un juego de cartas como el póker, el espacio de observación podría ser el conjunto de cartas que cada jugador tiene en la mano y las cartas comunitarias en la mesa.

Es importante entender el espacio de observación porque determina qué información tiene disponible el agente para tomar decisiones. La calidad y la relevancia de las observaciones influirán en la capacidad del agente para aprender y realizar acciones efectivas.

En resumen, es crucial tener una buena representación del entorno/problema en él espació de observación para obtener unos buenos resultados. En nuestro problema nos basaremos en la solución aportada en [19] en el cual plantean una forma de codificación para *GridWorld* o espacios matriciales como es nuestro caso (Véase en la figura 5.1).

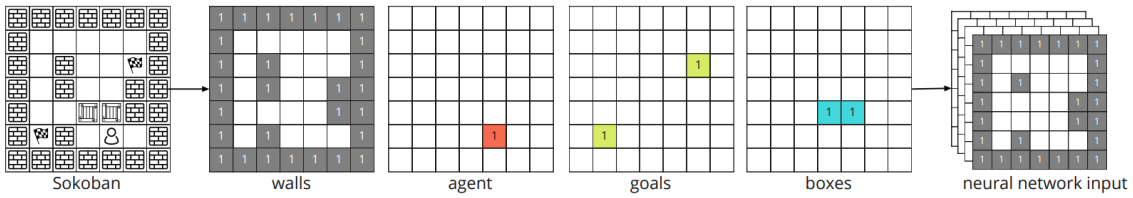

Figura 5.1: Grid Representation in NN [19]

Recordemos que este trabajo tenía bastantes iteraciones, en las cuales en primer lugar el robot debería aprender únicamente a recoger cajas y llevarlas al punto de entrega, en la siguiente iteración el agente tendría que ir a buscarlas al punto de recogida y colocarlas en las estanterías, etc. En cada iteración el espacio de observación cambia ligeramente, pero la codificación general funciona de la siguiente forma.

- **observación del Grid**: El robot debe observar el Grid para saber donde está, lo que sería equivalente a "ver" para los humanos, para eso el agente puede saber información de 7x7 casillas a su alrededor para un total de 49 casillas, saber su información significa, saber si hay una caja, si está vacío, si es una pared o si es el punto recogida. Para saber eso utilizamos 2 sub-grids como indican en [19], los grids contienen solamente 0, 1 y -1, el -1 simboliza que el agente está "'mirando" fuera del entorno" si en los 2 sub-grids encontramos un 0, significa que la casilla está vacía. Ahora bien, si encontramos un 1 depende de en que sub-grid se encuentra este, si está en el 1 significa que esa caja es reclamada en el punto de entrega, si el 1 está en el sub-grid 2 significa que esa casilla está ocupada, pero con un tipo de caja que no nos interesa. Podemos ver una representación visual en la figura 5.2

- **observación del Inventario**: Para el inventario el agente puede ver qué estado tiene cada espacio de su inventario con 2 valores, su capacidad máxima de cajas cargadas

simultáneamente es de 3 y las cajas pueden ser de dos tipos, cajas que necesitan ser entregadas o cajas que necesitan ser depositadas, por lo tanto, cada espacio de inventario se codifica con un one-hot-encoding de estas dos cajas, si los dos valores son 0, ese espacio del inventario está vacío.

- **observación de la demanda**: Para saber cuanto le falta al robot tiene información de cuantas cajas faltan por ser recogidas y cuantas cajas faltan por ser entregadas. Eso es el valor exacto de cajas que faltan para cada tipo de demanda.
- **observación del punto de recogida/descarga**: Para hacer la vida más fácil al agente, este sabe en todo momento a la distancia que se encuentra del punto de recogida/-descarga.

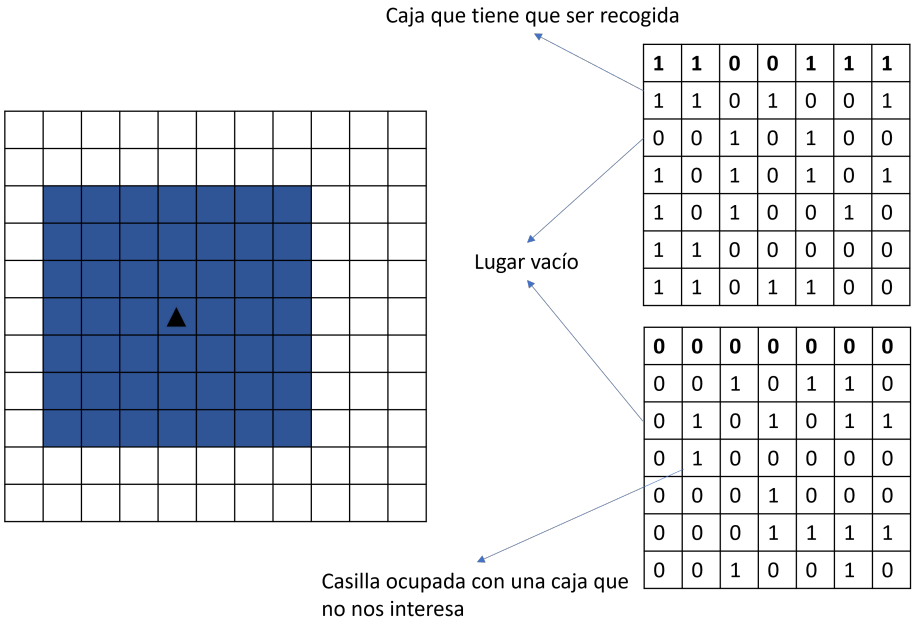

Figura 5.2: Visión del agente

CAPÍTULO 6
Interacción con el entorno

Cada vez que se realiza un reseteo (empieza un nuevo episodio) en el entorno el inventario del robot se vacía, el agente se coloca de nuevo en una posición inicial cerca del centro y se borra la demanda acumulada de antiguos episodio y se crea una nueva, el estado de las estanterías permanece constante.

Para añadir demanda en cajas que deben ser colocadas en las estanterías se inicializa una cola con longitud variable y tipos de caja aleatoria, cada vez que el robot ejecute la acción "Coger" en la casilla de recogida, la siguiente caja en la cola será cargada en el robot y eliminada de la cola.

Para añadir demanda en las cajas que se deben entregar, simplemente asignamos un número aleatorio de demanda a cada tipo de caja que tenemos en el entorno. Cada vez que se ejecute una acción "Dejar" en el sitio de descarga, la demanda de un tipo de caja i bajará una unidad.

En cada paso del entorno, una acción deberá ser elegida por el agente, así es como gestionamos las acciones en el entorno:

1. Si la acción es de dirección, se comprueba que este no se esté saliendo de los límites del entorno y se lleva a cabo la acción.
2. Si la acción es de "Coger" existen dos situaciones donde se puede llevar a cabo esta acción:
   - Si el agente se encuentra en el punto de recogida y tiene espacio en su inventario, este coge la siguiente caja en la cola de demanda (primero se comprueba que haya cajas en la cola y de ser así se elimina de esta) y la coloca en la primera posición libre de su inventario.
   - La otra posible situación es que recoja una caja que se encuentra en una estantería, en ese caso comprobamos que efectivamente hay una caja en ese lugar y que el agente tiene espacio en su inventario, si eso se cumple se carga la caja.
3. Si la acción es de "Dejar" existen también dos situaciones donde se puede llevar a cabo:
   - Si el agente se encuentra en el punto de descarga y tiene alguna caja en su inventario, se comprueba que exista demanda de esta y se deja la primera de su inventario en este punto (recordemos que funciona como una cola).
   - Si el agente se encuentra en algún lugar de la estantería que esté vacío y tiene alguna caja en su inventarió, se deposita igual que en el anterior caso.

Finalmente, se hacen comprobaciones para estimar las recompensas de este paso:

- En cada paso por defecto la recompensa será de -0.1
- Si el robot ha hecho un “Coger“ de una caja de la cola de demanda del punto de recogida es recompensado con +10 de recompensa, si el “Coger“ es sobre una caja de las estanterías la recompensa es el de por defecto.
- Si este ha hecho un “Dejar“ de una caja con demanda en el punto de entrega, su recompensa será también de +10, al igual que antes, si el “Dejar“ es sobre alguna caja de las estanterías, la recompensa el de por defecto.
- Si el agente ha cumplido con toda la demanda del episodio, es decir, no queda ninguna caja por recoger/ordenar, el episodio acaba y este es recompensado con +1000 de recompensa.
- Sin embargo, si el agente ha agotado el número de pasos permitidos en este episodio, este también acaba, pero es penalizado con -1000 de recompensa.
- Finalmente, se hace una comprobación de si se debe añadir más demanda, eso es sí, pasos >= pasos para agregar demanda.

CAPÍTULO 7
Modelo y entrenamiento

7.1 Modelo: Arquitectura de red neuronal

La arquitectura de la red neuronal es crucial para el correcto funcionamiento del agente, si usamos una red neuronal muy simple puede existir *underfitting* haciendo que el modelo no aprenda correctamente. Por otra parte, arquitecturas demasiado complejas pueden producir demasiado ruido y complicar la convergencia.

En este trabajo se han usado arquitecturas muy similares a las mencionadas en los artículos [5] y [14], al tener observaciones en forma de grid podríamos pensar que en este caso utilizar una Red Neuronal Convolucional (CNN) para la observación cercana del agente y luego concatenar el resto de tensores podría ser una buena idea. Se ha intentado, pero al tener un grid tan pequeño (recordemos que la observación del agente es un 7x7 a su alrededor) no es necesario, lo único que provoca es aumentar el tiempo de entrenamiento necesario, ya que en un entrenamiento por aprendizaje por refuerzo usando CNN, primero las capas convolucionales tienen que aprender a "ver" y posteriormente el modelo aprende a hacer la tarea correcta.

Por lo tanto, se decidió aplanar todas las observaciones, concatenarlas y pasarlas a la red neuronal como capa de entrada, por ejemplo en la última iteración del trabajo (Agente de operativa de entrega y recogida a la vez con distintos tipos de cajas) tenemos 105 neuronas en la capa de entrada.

Como núcleo de nuestra red se estudió el posible uso de Redes Neuronales Recurrentes (RNN), más en concreto usando capas LSTM, esto podría haber sido útil, ya que gracias a su capacidad de almacenar información sobre entradas anteriores, podría ser capaz de optimizar sus rutas de una mejor forma, recordando lugares los cuales ya ha visitado anteriormente. Lamentablemente, el hecho de usar capas recurrentes de forma efectiva en el algoritmo PPO no es tan fácil como simplemente añadir estas capas, sino que se tiene que tener en cuenta a la hora de programar el algoritmo. Si es verdad que estas capas se usan en artículos como [21] o [3] no está demostrado que supongan una enorme mejora en el desempeño de los agentes.

Por lo tanto, usaremos un multilayer perceptron clásico, este tendrá 2 capas ocultas de 64 neuronas cada una, con una función de activación de tangente hiperbólica (Tanh), la cual nos limita los valores entre -1 y 1, se podría usar la ReLU la cual tiene una computación más sencilla, pero al no tener los valores de la capa de entrada entre 0 y 1 puede dar resultados y gradientes no deseados. Esta arquitectura se compartirá tanto para el actor como para el crítico (la función valor), lo único que cambia es la capa de salida, la del actor será una capa de salida con 6 neuronas (una para cada posible acción discre-

ta) con una función de activación softmax y la del crítico será una sola neurona con una activación lineal (o lo que es lo mismo, sin función de activación)

A continuación veremos los hiperparámetros a tener en cuenta, los valores elegidos para estos se han basado en los artículos [14] donde proponen el algoritmo PPO y posibles valores de sus hiperparámetros y [5] que es un problema muy similar al nuestro.

- **Coeficiente de factor de descuento (discount factor)**: Al determinar la importancia relativa de las recompensas futuras en comparación a las inmediatas suelen ser valores muy cercanos a 1, como 0.99, 0.999 o 0.9999, en nuestro caso y siguiendo el artículo [5] usaremos 0.99
- **Tasa de aprendizaje (learning rate)**: La velocidad a la que se actualizan los parámetros en la política suele ser muy importante para evitar grandes saltos, por lo tanto, queremos valores bajos para evitar eso mismo, en nuestro caso será 0.0003
- **Coeficiente de ratio de recorte (clipping ratio)**: Es utilizado en la función de pérdida para limitar las actualizaciones de la política. En nuestro caso usaremos 0.2 como mencionan en [14]
- **Coeficiente de ratio de valor (value coefficient)**: Controla la importancia relativa del término de valor en la función de pérdida. Hemos decidido usar 0.5, ya que parece dar mejores resultados según [14]

7.2 bucle de entrenamiento

En este capítulo estudiaremos como hemos llevado a cabo el bucle de entrenamiento, como recopilamos datos durante el entrenamiento y como guardaremos los modelos en cada iteración.

Como hemos mencionado anteriormente, este trabajo ha tenido distintas iteraciones a lo largo del proyecto y no en todas el bucle de entrenamiento ha funcionado de la misma forma, ya que hay tareas que necesitan de un aprendizaje más extremo para realizarse correctamente que otras. Pero en general podemos encontrar una tendencia similar, en los primeros pasos el modelo aprende a hacer la tarea base, ya sea recoger cajas, depositarlas o las dos, normalmente se le da un entorno sencillo para resolver y pocas cajas de demanda, por ejemplo un entorno de 11x11 y 5 cajas como objetivo. Luego de determinados pasos de tiempo o luego de que se llegue a un umbral, el entorno cambiará de tamaño o el número de cajas de demanda o ambos al mismo tiempo, por lo que podemos esperar una bajada de rendimiento del agente, pero que deberá recuperarse lentamente. Esto se debe a que, si lo colocamos directamente en un entorno muy grande con muchas cajas, es difícil que el modelo llegue a converger en algún punto, y si lo hace necesitaríamos muchos pasos de tiempo.

El algoritmo que usaremos para entrenar la red neuronal será PPO[14] el cual hemos explicado anteriormente, ya que nos permite trabajar con espacios de acción discretos. Adicionalmente, cada 1000 pasos de tiempo se lleva a cabo una llamada a la función callback, la cual va a calcular la recompensa media de los últimos 100 episodios y guardarla en un archivo de registros para su posterior visualización, además de guardar el modelo solamente si la recompensa actual supera el mejor recompensa anterior, así nos quedaremos siempre con el mejor modelo durante el entrenamiento, no con el último. Es normal ver oscilaciones en las recompensas medios durante el entrenamiento en todos los algoritmos de reinforcement learning, más aún cuando usamos magnitudes de recompensas altas como son las nuestras. En el siguiente capítulo veremos los resultados en cada iteración del trabajo con cada algoritmo.

CAPÍTULO 8
Resultados

Este capítulo tendrá 4 secciones, una para cada objetivo del robot, recordemos que empezamos solamente entrenando al agente parar recoger cajas, y desde ahí vamos escalando la complejidad, en cada sección haremos un pequeño resumen de lo que intentamos y como lo hacemos y los resultados comentados brevemente.

8.1 Objetivo 1: Entregar cajas genéricas

En este objetivo el agente tenía que ser capaz de recoger cajas genéricas del entorno y depositarlas en el lugar de entrega que está en el centro, la información del agente sería su entorno (si hay caja, si está vacío, si es el lugar de entrega o es una pared), el número de espacios en su inventario, el número de cajas que le faltan por entregar y la distancia al punto de entrega. En este caso el robot tiene completa libertad de hacer lo que quiera, puede mover las cajas que él quiera, pero solamente depositar cajas en el punto de entrega le supondrá una recompensa positiva. El número de cajas que tendrá que entregar es aleatorio en cada episodio, el mínimo es 4 y el máximo depende de la complejidad que queramos darle al entorno, irá variando durante el entrenamiento.

Para considerar que el objetivo se ha resuelto correctamente la media de las recompensas debe estar en torno a 1000, por el sistema de recompensas utilizado, puede ser que esté un poco por debajo o incluso por encima, depende de cuán eficientemente lo consiga resolver, ya que recordemos que por cada paso la recompensa se ve disminuido -0.1, pero con cada caja +10.

En este caso el bucle de entrenamiento será de unos 300000 pasos de tiempo totales, cada 100000 pasos se cambia el entorno por uno más complejo, en la 1.ª etapa es entrenado en un entorno de 11x11 con un mínimo de 4 y un máximo de 5 cajas que entregar, es una tarea bastante sencilla y se espera que la consiga resolver relativamente rápido, en la 2.ª etapa el tamaño sigue siendo de 11x11, pero ahora el máximo de cajas que puede que tenga que entregar pasa a ser 20, aquí nos aseguramos que entiende su tarea según la información que él posee de las cajas que tiene que entregar y no está llevando a cabo una serie de acciones predeterminadas todo el rato y finalmente en la 3.ª se agranda el tamaño del entorno a 21x21 para confirmar que es capaz de trabajar en distintos tamaños de entorno.

- **Primera etapa**: Si observamos la figura 8.1 en la primera etapa el agente parte desde 0 sin saber nada, pero podemos observar que rápidamente descubre como aumentar su recompensa de forma casi exponencial, situándose alrededor de 200 de recompensa media a los 5000 pasos de tiempo y a los 20000 pasos de tiempo ya es capaz de obtener una recompensa media de unos 900, lo cual se podría considerar

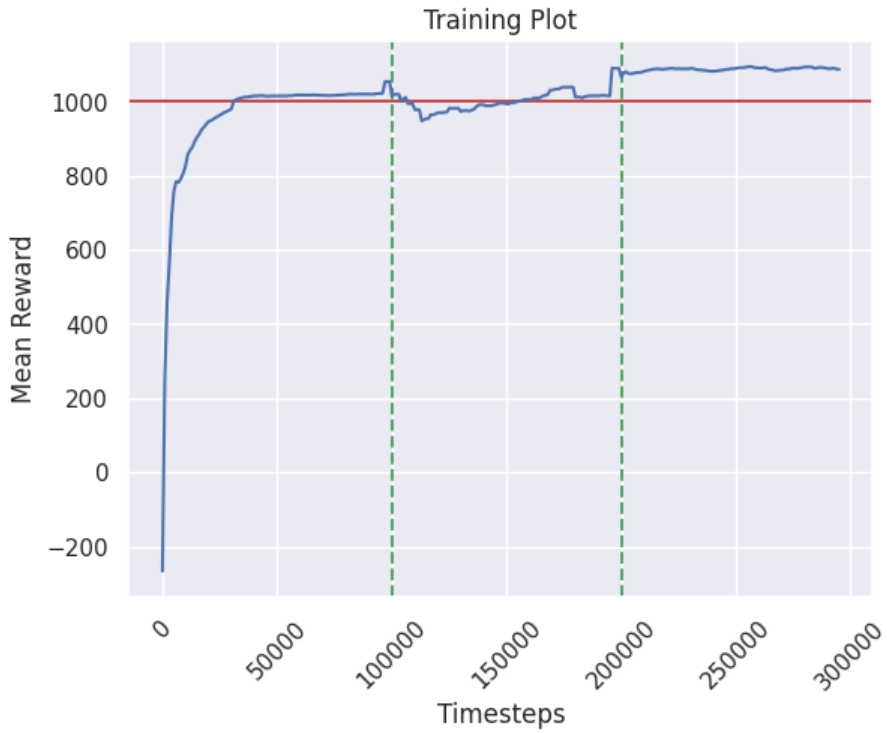

Figura 8.1: Curva de entrenamiento para entrega de cajas

ya resuelto, luego hasta llegar a los 100000 pasos de tiempo el modelo sigue mejorando lentamente aumentando su eficiencia en la tarea (reduciendo su número de pasos necesários)

- **Segunda etapa**: En este momento que aumentamos el número de cajas que el agente debe entregar vemos una pérdida de rendimiento, lo cual es esperable al estar aumentando la complejidad, pero este se recupera a lo largo de los pasos de tiempo de una manera constante, podemos ver pequeñas bajadas, pero es normal al estar frente un problema de aprendizaje por refuerzo. Pero llega a mejorar por bastante por encima del umbral del que se considera resuelto, aumentando muchísimo la eficiencia.

- **Tercera etapa**: En la etapa final se ha aumentado el tamaño del entorno, pero como podemos ver la pérdida de rendimiento es mínima y se recupera casi instantáneamente, en este punto la eficiencia del agente parece haber llegado a su máximo y el entrenamiento se estanca en esa asíntota.

En la tabla 8.1 podemos observar las recompensas medios máximos obtenidos en cada etapa del entrenamiento, junto a su entorno de entrenamiento, recordemos que nos hemos quedado con el mejor modelo, no con el último gracias al callback, por lo tanto, nuestra recompensa media final es de 1095.02.

Tabla 8.1: Recompensas máximas por etapa de entrenamiento en entrega

Etapa	Entorno	Mejor recompensa media
1	tamaño = 11x11 mínimo de cajas 4, máximo 5	1022.25
2	tamaño = 11x11 mínimo de cajas 4, máximo 20	1054.33
3	tamaño = 21x21 mínimo de cajas 4, máximo 20	1095.02

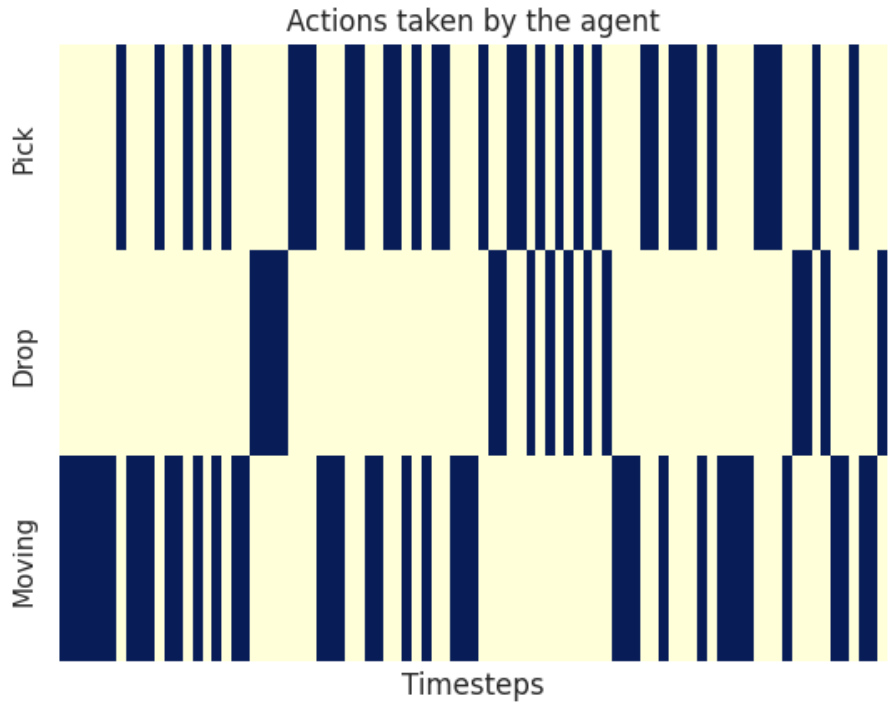

Figura 8.2: Mapa de acciones en entrega de cajas

Adicionalmente, en cada objetivo nos pareció interesante saber como se distribuían las acciones del robot a lo largo, (Véase en la figura 8.2 para ello se creó una gráfica de mapa de calor que nos enseña como se distribuyen las acciones tomadas por el agente a lo largo de los pasos de tiempo, para crearla ejecutamos un episodio de forma aleatoria y lo resolvemos recopilando los datos de este, en este caso intentaremos que el episodio relativamente sencillo y se puede resolver con pocos pasos, así no nos queda una gráfica demasiado difícil de leer.

En la figura 8.2 podemos observar como la mayoría de acciones que realiza el agente son de movimiento como podíamos esperar; sin embargo, parece que intenta hacer muchas acciones de "Coger" seguidas, recordemos que el inventario del agente es de 3 espacios, por lo tanto, más de 3 acciones de "Coger" o de "Dejar" seguidas no tienen mucho sentido, una forma de evitar esto tal vez sería darle una recompensa negativa para que no haga estas decisiones "malas", pero al tratarse de un algoritmo de machine learning relativamente sencillo, creemos que podría llegar a ser incluso contraproducente y empeorar el resultado, otro modo de solucionarlo podría ser aumentar el tiempo de entrenamiento. En último lugar, las acciones de "Dejar", las cuales consideramos las importantes al tener que entregar cajas, parecen ser bastante acertadas y no las hace aleatoriamente.

Finalmente, testearemos el funcionamiento del modelo, para ello simplemente le dejaremos actuar durante 1000 episodios seguidos para ver si es capaz de resolver todos y cada uno de estos episodios, también estudiaremos cuanto tiempo o pasos necesita para resolver-los y cuantas órdenes de demanda consume en cada episodio, lo ideal sería que resolviese todas las órdenes de demanda a la primera, pero si no lo hace y consigue recuperarse también es buena señal. El entorno que se ve a testear en este caso va a ser de 101x101 con un máximo de 30 cajas a entregar. En la figura 8.3 podemos ver 20 episodios aleatorios de estos 1000 episodios del testing. Los resultados finales nos indican que puede completar todo a la perfección con un 100 % de episodios completados y con una media de 478.08 pasos por episodio.

Cajas Entregadas	Cajas en el inventario del robot	Completado	Steps	Demandas
20	0	True	601	1
23	0	True	649	1
17	1	True	390	1
6	0	True	76	1
10	0	True	250	1
8	1	True	178	1
6	2	True	143	1
26	0	True	843	1
4	1	True	90	1
27	0	True	707	1
4	0	True	67	1
29	0	True	667	1
18	0	True	290	1
4	0	True	84	1
16	3	True	354	1
6	1	True	119	1
14	1	True	330	1
25	0	True	474	1
24	1	True	510	1
6	1	True	110	1

Figura 8.3: Testing en entrega de cajas

Como podemos ver en la figura 8.3 todos los episodios se han completado de forma correcta en relativamente pocos pasos, es normal que cuantas más cajas se tengan que entregar, más pasos se requieran, cabe destacar que a veces el agente deja su inventario vacío (Cajas en el inventario del robot) esto va a beneficiar a las futuras cargas de demanda, así el agente no deberá primero vaciar el inventario y volverlo a cargar de nuevo. También podemos ver que todos los episodios los resuelve con una sola orden de demanda, es decir, no se le acumulan pedidos.

8.2 Objetivo 2: Colocar cajas genéricas

En este objetivo el agente tenía que ser capaz de colocar cajas genéricas del punto de recogida, el cual está en el centro y depositarlas en las estanterías, como el objetivo anterior la información del agente sería su entorno (si hay caja, si está vacío, es el lugar de recogida o es una pared), el número de espacios en su inventario, el número de cajas que le faltan por colocar y la distancia al punto de recogida. Aquí, el robot también tiene completa libertad de hacer lo que quiera, puede mover las cajas que él quiera, pero solamente coger cajas del punto de recogida le supondrá una recompensa positiva. El número de cajas que tendrá que colocar también es aleatorio en cada episodio, el mínimo es 4 y el máximo vuelve a depender de la etapa. Al igual que antes, para considerar que el objetivo se ha resuelto correctamente, la media de las recompensas debe estar en torno a 1000.

El bucle de entrenamiento es exactamente igual que el anterior objetivo, unos, 300000 pasos de tiempo totales, volvemos a tener 3 etapas, En la primera etapa, el modelo se en-

trena en un entorno de tamaño 11x11, donde debe recoger un mínimo de 4 y un máximo de 5 cajas. Esta tarea es relativamente fácil y se espera que el modelo la resuelva rápidamente. En la segunda etapa, el tamaño del entorno sigue siendo de 11x11, pero ahora el número máximo de cajas que el modelo puede tener que recoger aumenta a 20. Aquí, el objetivo es asegurarse de que el modelo entienda la tarea basándose en la información que tiene sobre las cajas que debe recoger, en lugar de simplemente seguir una serie de acciones predefinidas todo el tiempo. Finalmente, en la tercera etapa, el tamaño del entorno se amplía a 21x21 para confirmar que el modelo es capaz de trabajar en diferentes tamaños de entorno.

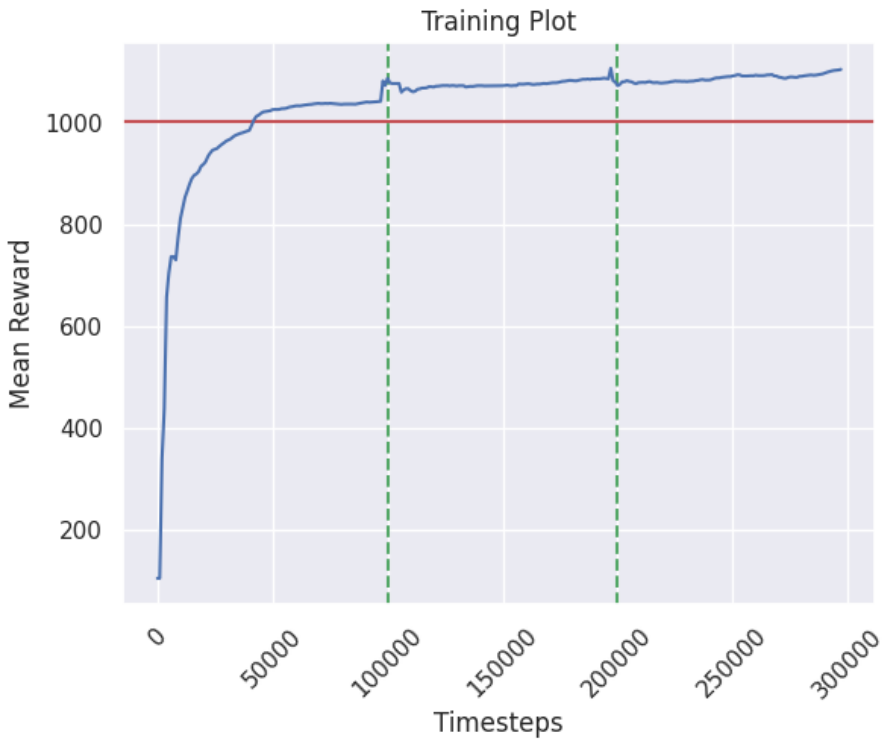

Figura 8.4: Curva de entrenamiento para la colocación de cajas

- **Primera etapa**: Si observamos la figura 8.4 como antes el agente parte desde 0 sin saber nada sobre la tarea que tiene que realizar, es muy parecido a la primera etapa del objetivo anterior, parece que rápidamente encuentra la forma de conseguir un buen recompensa rápidamente y a poco a poco va a mejorando su eficiencia hasta los 10000 pasos de tiempo que alcanza su máximo.

- **Segunda etapa**: En este momento que aumentamos el número máximo de cajas que el agente puede tener que colocar en las estanterías, como podemos ver, a diferencia del caso anterior, esto no parece afectar en lo más mínimo al rendimiento, ya que parece haber una pequeña bajada, pero esta puede deberse perfectamente a las oscilaciones propias del algoritmo de aprendizaje por refuerzo o de la optimización mediante gradiente.

- **Tercera etapa**: En la etapa final se ha aumentado el tamaño del entorno, puede ser que haya una pequeñísima pérdida de rendimiento, pero se recupera muy rápido y parece quedarse estancando en un máximo, creado una asíntota horizontal.

las recompensas medios máximos en cada etapa se pueden encontrar en la tabla 8.2

Tabla 8.2: Recompensas máximas por etapa de entrenamiento en colocación

Etapa	Entorno	Mejor recompensa media
1	tamaño = 11x11 mínimo de cajas 4, máximo 5	1080.90
2	tamaño = 11x11 mínimo de cajas 4, máximo 20	1105.80
3	tamaño = 21x21 mínimo de cajas 4, máximo 20	1103.56

Como podemos ver, nuestra recompensa media final de la etapa 3 es peor que el de la etapa 2, pero al ser tan mínima la diferencia, hemos preferido usar el modelo de la etapa 3 al haber sido entrenado en más entornos.

Figura 8.5: Mapa de acciones en recogida de cajas

En la figura 8.5, podemos observar que al igual que antes existe una acción predominante sobre el resto, la de movimiento, lo cual tiene bastante sentido, a diferencia del objetivo anterior, aquí parece ser mucho más consecuente y eficaz con las acciones de "Dejar" y "Coger" llevándolas a cabo un menor número de veces y parece que gestionara mejor el espacio en su inventario. Sería complicado decir a simple vista, si realiza más acciones de "Dejar" que de "Coger", podríamos incluso pensar que las dos se hacen el mismo número de veces, lo cual, sería lo más óptimo.

Como antes, testearemos el funcionamiento del agente en 1000 episodios seguidos de un tamaño de 101x101 con un máximo de 30 cajas por entregar. Los resultados nos indican que se resuelven el 100 % de los episodios correctamente con una media de 424.18 pasos similar al objetivo anterior. Todos los episodios se resolvieron con una sola entrada de demanda, por lo tanto, en ningún momento se acumularon pedidos. En la figura 8.6 podemos observar información detallada sobre 20 de estos 1000 episodios.

Cajas Ordenadas	Cajas en el inventario del robot	Completado	Steps	Demandas
27	0	True	1207	1
17	1	True	260	1
21	1	True	475	1
23	2	True	347	1
12	0	True	182	1
17	0	True	244	1
16	0	True	297	1
11	2	True	197	1
8	2	True	113	1
10	1	True	147	1
10	2	True	133	1
19	2	True	311	1
16	0	True	236	1
16	2	True	269	1
16	0	True	231	1
22	1	True	402	1
4	2	True	104	1
19	2	True	275	1
21	1	True	407	1
4	0	True	98	1

Figura 8.6: Testing en colocación de cajas

8.3 Objetivo 3: Entregar y Colocar cajas genéricas (Operativa mixta)

Ahora el agente tendrá que ser capaz de gestionar la logística completa del almacén, tanto las entradas como las salidas sin ningún tipo de limitación o que le digamos que tenga que hacer primero, el mismo deberá priorizar las cajas que deben salir antes, o las que tienen que ser ordenadas. En un futuro se podría crear un tag para la prioridad de las cajas de forma manual, pero esto lo hemos evitado por ahora, ya que se escapa del propósito del trabajo, que es que el agente pueda gestionar únicamente la logística. La información que el agente recibe es exactamente igual que en anteriores objetivos. Como siempre, el agente tiene completa libertad para hacer cualquier cosa, coger y dejar las cajas que quiera. Aquí el sistema de demanda cambia un poco, la cola de demanda de entrega de cajas será un número aleatorio entre 20 y 50, elegido de forma empírica, ya que es lo que normalmente se gestiona en los almacenes de Inditex, la demanda de colocación de cajas también es una cola de entre 20 y 50 cajas. El sistema de recompensas permanece constante, por lo tanto, una recompensa media de 1000 se considera que el problema está resuelto

El bucle de entrenamiento en este caso solamente serán 150000 pasos de tiempo en total, como antes tendremos 3 etapas, anteriormente aumentábamos tanto el número de cajas que se tenían que entre entregar/recoger y el tamaño del entorno en cada etapa, en este objetivo al ser el número de cajas al tener un máximo y un mínimo de cajas preestablecido de entre 20 y 50 no hay opción a aumentar estos límites, por lo tanto, se quedarán

constantes a lo largo del entrenamiento y solo sé varía el tamaño del entorno. Por esta razón no necesitamos tantos pasos de tiempo como en los anteriores objetivos, ya que el número de cajas a entregar/recoger, que solía ser el parámetro que cambiaba el resultado de la recompensa media, ya no existe. En la primera etapa el entorno será de 11x11, en la segunda etapa de 21x21 y en la tercera etapa de 51x51.

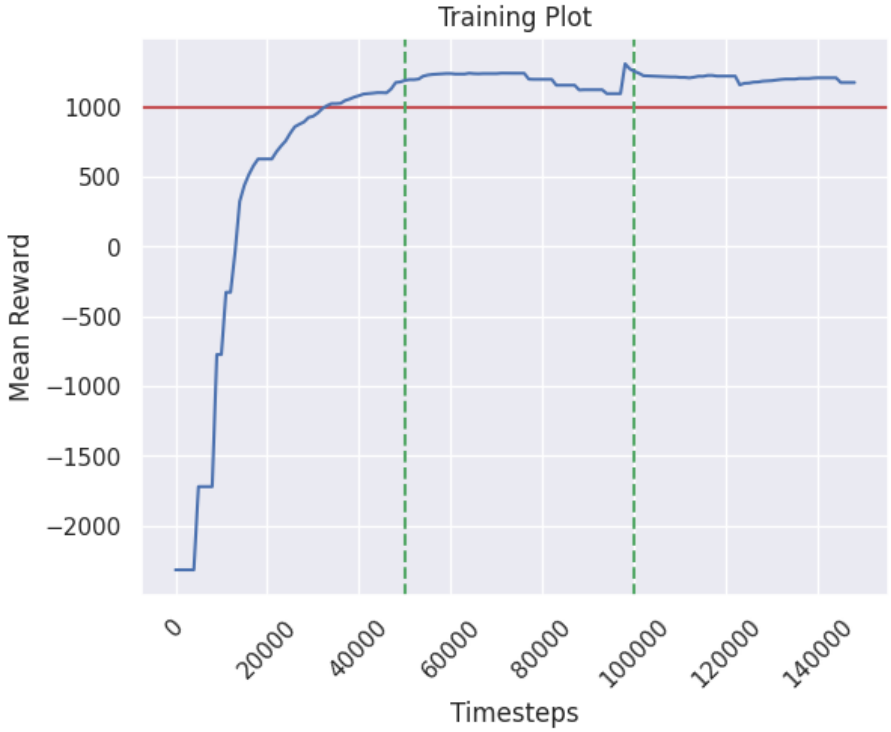

Figura 8.7: Curva de entrenamiento para la operativa mixta

- **Primera etapa**: En la figura 8.7 a comparación de la de las otras curvas de entrenamiento, parece que le cuesta un poco más llegar a "resolver" el objetivo, pero recordemos que antes le dábamos 100000 pasos de tiempo por etapa y ahora solamente 50000, entonces la curva de aprendizaje es muy similar o igual a los otros objetivos para la primera etapa. Alrededor de los 30000 pasos de tiempo parece haber resuelto por completo superando la recompensa media de 1000.
- **Segunda etapa**: En este momento aumentamos el tamaño del entorno y parece no verse afectado el modelo en absoluto, a mitad de esta etapa el modelo empieza a bajar su recompensa media, pero asumimos que es por la inestabilidad propia del entrenamiento de los modelos de aprendizaje por refuerzo, ya que al final se consigue recuperar, de todas maneras nosotros siempre nos quedamos con el mejor modelo en cada etapa.
- **Tercera etapa**: En esta etapa pasa algo muy similar que en la etapa dos, parece que el modelo baja su rendimiento, pero seguramente sea por la propia inestabilidad del entrenamiento y simplemente va fluctuando entre esos valores, recordemos que esos valores son muy buenos, ya que están por encima de valores de 1000, por lo tanto, no nos preocupan mucho.

las recompensas medios máximos en cada etapa se pueden encontrar en la tabla 8.3 como podemos ver en cada etapa, incluso mejoramos la recompensa media de la etapa anterior, así que nos quedaremos con el mejor modelo de la etapa 3, que ha conseguido una recompensa de 1309.80.

Figura 8.8: Mapa de acciones en operativa mixta

Tabla 8.3: Recompensas máximas por etapa de entrenamiento en operativa mixta

Etapa	Entorno	Mejor recompensa media
1	tamaño = 11x11 mínimo de cajas 20, máximo 50	1103.09
2	tamaño = 21x21 mínimo de cajas 20, máximo 50	1242.68
3	tamaño = 51x51 mínimo de cajas 20, máximo 50	1309.80

En el mapa de calor de las acciones elegidas por el agente (Figura 8.8) podemos observar que pasa algo similar al objetivo 1 de entrega de cajas, las acciones predominantes son "Coger" y acciones de movimiento. Aquí "Coger" predomina encima del movimiento, eso nos hace pensar que el agente intentar hacer muchos "Coger" que en realidad son innecesarios empeorando la eficiencia, una forma de solucionarlo sería imponer un impacto negativo en la recompensa para que dejara de hacerlo. Lo probamos, pero empeoraba tanto el modelo que este dejaba de funcionar por completo, por lo tanto, preferimos sacrificar esta eficiencia y obtener al menos un funcionamiento correcto. Como es obvio, la siguiente acción más usada es la de movimiento para recorrer el grid. Finalmente, los "Dejar" que realiza el agente parecen ser muy acertados, haciendo solamente los necesarios, aun así parece que no son del todo eficientes, ya que al tener 3 espacios en el inventario del agente no parece aprovecharlos del todo, pero es muy complicado de saber. Al final el agente ha aprendido mediante reinfocement learning y es casi imposible que lleve a cabo todos los movimientos de forma perfecta y eficiente.

Para el testing de este objetivo seguimos con los 1000 episodios como en anteriores casos, esta vez se reduce el tamaño del entorno a 51x51 porque ahora el objetivo es mucho más complejo, ya que tiene que entregar y recoger un gran número de cajas, entre 20 y 50.

Cajas por ordenar	Cajas por entregar	Cajas en el inventario del robot	Completado	Steps	Demandas
0	0	[1 1 1]	True	629	1
0	0	[1 1 0]	True	493	1
0	0	[0 0 0]	True	920	1
0	0	[0 0 0]	True	1092	1
0	0	[1 0 0]	True	327	1
0	0	[0 0 0]	True	150	1
0	0	[1 1 0]	True	420	1
0	0	[1 1 1]	True	337	1
0	0	[1 0 0]	True	2774	2
0	0	[0 0 0]	True	366	1
0	0	[0 0 0]	True	2914	2
0	0	[0 0 0]	True	2641	2
0	0	[1 1 1]	True	317	1
0	0	[0 0 0]	True	2690	2
0	0	[1 1 0]	True	2629	2
0	0	[0 0 0]	True	196	1
0	0	[1 1 1]	True	249	1
0	0	[1 1 0]	True	261	1
0	0	[1 1 0]	True	1024	1
0	0	[1 1 1]	True	1171	1

Figura 8.9: Testing en colocación y entrega de cajas

En la figura 8.9 podemos ver 20 de estos 1000 episodios, el formato es un poco distinto a las anteriores, ya que nos enseña las cajas que quedan por entregar por como se ha hecho el código de log del agente, pero como podemos ver sigue sin tener ningún problema, el modelo es capaz de resolver todos los episodios de forma correcta. Podemos ver bastantes fluctuaciones en los pasos necesarios para resolver cada episodio, eso explica la media de 800 pasos por episodio que nos da al calcularla de los 1000 episodios testeados, parece ser que algunas veces le cuesta de 2000 a 3000 pasos resolver el objetivo. La lista del inventario del robot simboliza que posiciones de su inventario tiene cubiertas, si hay un 0 significa que esa posición está libre, si hay un 1 significa que hay una caja. En cuanto a las entradas de demanda dejamos de ver que todos los episodios se consiguen en 1 entrada, podemos ver que hay algunos episodios que tiene 2 entradas de demanda, esto no es algo necesariamente malo, ya que parece que el agente ha sabido recuperarse de esa acumulación de demanda y resolver el episodio antes del límite de tiempo, aun así no es algo que nos alegre, ya que el agente debería tener tiempo de sobra para cumplir cada demanda impuesta no se debería tener que recuperar.

8.4 Objetivo 4: Entregar y Colocar con distintos tipos de cajas

Finalmente, llegamos a nuestro objetivo verdadero, el agente deberá ser capaz de resolver el entorno con distintos tipos de cajas en él. Recordemos que el agente no necesita distinguir estos tipos de cajas, para que le sea más fácil estos han sido enmascarados y solamente se tienen en cuenta en el backend, para el agente únicamente existen 2 tipos de cajas, las que necesitan ser procesadas y las que ya han sido procesadas o no son necesarias. Como antes, el agente no tiene ningún tipo de restricción a la hora de mover cajas por el entorno, puede mover cajas que ya han sido procesadas perfectamente, por si por alguna razón quisiera optimizar el espacio. Como el anterior objetivo por concenso con Inditex y similitud con su almacén, las órdenes de demanda se situarán entre 20 y 50 cajas en total, es decir, contando con todos los tipos de cajas, no para cada tipo de caja, ya

que sería imposible. El sistema de recompensas permanece constante, por lo tanto, una recompensa media de 1000 se considera que el problema está resuelto.

En este objetivo no sabemos con cuantas cajas será capaz de trabajar el agente, ya que es lo más complicado del trabajo que el agente sepa trabajar con distintos tipos de cajas, pero si saber exactamente qué tipo de caja es, por lo tanto, llevaremos a cabo distintas pruebas a ver como reacciona el agente. El primer bucle de entrenamiento y el más sencillo que usaremos será de 300000 pasos de tiempo y 3 etapas, cada una de 100000 pasos de tiempo. Recordemos que en todas las etapas el agente tiene que entregar y recoger entre 20 y 50 cajas. En la primera etapa el agente tendrá 3 tipos de cajas distintos y un entorno de 11x11, en la siguiente etapa el entorno será de 21x21 y finalmente en la tercera etapa de 31x31.

Figura 8.10: Curva de entrenamiento con un size de 31 y 3 tipos de cajas

- **Primera etapa**: Como podemos ver en la gráfica de entrenamiento (Figura 8.10 el agente parece costarle bastante en comparación a objetivos anteriores, aprender y optimizar este proceso. En la primera etapa relativamente rápido, con unos, 25000 pasos de tiempo, consigue obtener una recompensa media de alrededor de 500 y parece ser que se estanca ahí bajando un poco hasta que da un pico que supera los 1000. Este pico es un tanto sospechoso porque aparece de la nada, pero por como es el funcionamiento de estos algoritmos de aprendizaje por refuerzo podría ser que pasara algo así.

- **Segunda etapa**: En la segunda etapa, cuando aumentamos el tamaño del entorno, podemos observar una bajada drástica en la recompensa del agente como es de esperar, pero rápidamente se recupera y se vuelve a quedar estancado alrededor de 500 hasta que vuelve a darnos un pico por encima de 1000, exactamente igual que lo que pasaba en la etapa 2.

- **Tercera etapa**: En la tercera etapa repetimos el patrón de inicio de las otras etapas, podemos observar una bajada no tan alta como antes de recompensa media y nos

estancamos alrededor de 500, parece no ser capaz de encontrar un mejor mínimo para la función de perdida y va oscilando entre esos valores indefinidamente.

Parece que nuestro límite es de 21x21 en tamaño con 3 tipos de cajas, antes de probar a subir el número de tipos de cajas vamos a ver como se comporta este agente con un entorno de 21x21 y 3 tipos de cajas y en un entorno de 31x31 y también 3 tipos de cajas. Vamos a testear estos dos tipos de entornos en 1000 episodios para saber si es capaz de resolverlos de forma correcta.

Cajas por ordenar	Cajas por entregar	Cajas en el inventario del robot	Completado	Steps	Demandas
[0 0 0]	[0 0 0]	[1 3 0]	True	324	1
[0 0 0]	[0 0 0]	[0 0 0]	True	438	1
[0 0 0]	[0 0 0]	[0 0 0]	True	935	1
[0 0 0]	[0 0 0]	[3 3 2]	True	786	1
[0 0 0]	[0 0 0]	[3 0 0]	True	922	1
[0 0 0]	[0 0 0]	[2 1 2]	True	1973	1
[0 0 0]	[0 0 0]	[1 0 0]	True	799	1
[0 0 0]	[0 0 0]	[1 2 3]	True	949	1
[0 0 0]	[0 0 0]	[3 3 2]	True	1117	1
[0 0 0]	[0 0 0]	[2 0 0]	True	1169	1
[0 0 0]	[0 0 0]	[2 1 0]	True	446	1
[0 0 0]	[0 0 0]	[3 1 0]	True	1678	1
[0 0 0]	[0 0 0]	[0 0 0]	True	1923	1
[0 0 0]	[0 0 0]	[0 0 0]	True	446	1
[0 0 0]	[0 0 0]	[0 0 0]	True	365	1
[0 0 0]	[0 0 0]	[0 0 0]	True	1298	1
[0 0 0]	[0 0 0]	[1 2 1]	True	1462	1
[0 0 0]	[0 0 0]	[0 0 0]	True	207	1
[0 0 0]	[0 0 0]	[1 3 0]	True	350	1
[0 0 0]	[0 0 0]	[0 0 0]	True	645	1

Figura 8.11: Testing en colocación y entrega con tamaño de 21x21 y 3 tipos de cajas

En la figura 8.11 podemos ver 20 de estos 1000 episodios del entorno de 21x21 y 3 tipos de cajas, en esta muestra todos los episodios han sido completados con éxito y con una sola orden de demanda, como podemos ver en las 3 primeras columnas, las cajas restantes por ordenar y entregar de cada tipo son 0. En cuanto a los pasos necesarios para resolver estos entornos pueden variar muchísimo, ya que podemos encontrarnos con una resolución con 200 pasos y otras que han necesitado casi 2000 pasos para ser resueltas, esto es normal debido a tener que gestionar 3 tipos de cajas con las limitaciones que tiene el agente. Pero comprobando los 1000 episodios nos encontramos con un 97 % de completado, lo que significa que el 3 % de los episodios, unos 30 los ha fallado y no ha conseguido el objetivo de entregar o recoger todas las cajas. Los pasos medios de los 1000 episodios se sitúan en 1239, no es algo muy eficiente. Aun así comprobamos que pasa con el entorno de 31x31 (Véase la figura 8.12) aunque no esperemos muy buenos resultados.

En 20 de estos 1000 episodios ya podemos encontrar 6 episodios fallidos, el cual en los 1000 episodios aumenta a 87 % de completado correctamente, esto significa que hemos fallado unos 130 episodios, esto no parece mucho si lo comparamos con los 870 completados, pero hay que pensar que esto se debería poder implementar en un entorno real y que si falla episodios eso significa que el agente no está entendiendo bien algo del entorno o le falta información de algún tipo, es complicado saber donde fallan estos algoritmos al ser de caja negra y no tener acceso a más información. Además, si observamos algunos de estos episodios fallidos, normalmente en la primera orden de demanda el agente tal vez le quedan 2-3 cajas para entregar, pero al llegar la siguiente orden, en vez de inten-

Cajas por ordenar	Cajas por entregar	Cajas en el inventario del robot	Completado	Steps	Demandas
[0 0 0]	[0 0 0]	[3 1 2]	True	411	1
[0 0 0]	[0 0 0]	[0 0 0]	True	555	1
[0 0 0]	[0 0 0]	[0 0 0]	True	910	1
[0 0 0]	[0 0 0]	[1 2 1]	True	997	1
[5 3 4]	[4 4 9]	[0 0 0]	False	5000	3
[0 0 0]	[0 0 0]	[0 0 0]	True	938	1
[0 0 0]	[0 0 0]	[2 0 0]	True	903	1
[6 3 3]	[4 8 4]	[0 0 0]	False	5000	3
[4 4 5]	[5 8 8]	[1 2 1]	False	5000	3
[0 0 0]	[0 0 0]	[3 1 2]	True	598	1
[0 0 0]	[0 0 0]	[0 0 0]	True	1685	1
[0 0 0]	[0 0 0]	[1 2 1]	True	669	1
[0 0 0]	[0 0 0]	[1 0 0]	True	645	1
[8 2 5]	[13 3 8]	[3 3 1]	False	5000	3
[0 0 0]	[0 0 0]	[3 3 0]	True	2997	2
[0 0 0]	[0 0 0]	[0 0 0]	True	506	1
[0 0 0]	[0 0 0]	[0 0 0]	True	1691	1
[5 2 3]	[7 8 2]	[0 0 0]	False	5000	3
[3 3 7]	[6 12 5]	[0 0 0]	False	5000	3
[0 0 0]	[0 0 0]	[0 0 0]	True	1832	1

Figura 8.12: Testing en colocación y entrega con tamaño de 31x31 y 3 tipos de cajas

tar recuperarse parece quedarse estancado en intentar recuperar esas cajas no entregadas anteriormente, esto parece indicar que el agente no está entendiendo bien su objetivo de entregar un cierto número de cajas, se ha intentado cambiar la representación de esto, pero nada ha dado un resultado mejor, siempre empeoraba.

Si miramos la distribución de acciones del agente durante algunos episodios por si pudiéramos detectar algún patrón extraño (En la figura 8.13 podemos ver uno de estos) y todos tenía un formato similar y sin nada que levantara sospechas, ya que como podemos ver casi todas las acciones llevadas a cabo por el agente son de movimiento seguido de "Coger" y "Dejar", algo que esperaríamos. Algo que si nos concierne y nos levantó dudas fue que en episodio que mostramos en la gráfica, el cual falló, al final casi solamente intenta hacer movimientos como si no supiera muy bien que está haciendo y no intenta ni coger y dejar cajas, esta podría ser la razón del fallo, pero es difícil saber porque pasa esto.

Aun así decidimos probar a hacer un entrenamiento con 5 tipos de cajas para ver lo que pasaba, partiríamos desde el mejor modelo pasado que era con 3 tipos de cajas y un entorno de 21x21 y le entrenaríamos para ese mismo entorno pero con 5 cajas.

Como era de esperar en la curva de entrenamiento (Figura 8.14) el modelo no es capaz de aprender, se queda estancado en un mínimo local que está bastante alejado de ser el óptimo, ya que obtenemos una recompensa media de unos 350. El algoritmo fue entrenado por 100000 pasos de tiempo sin cambio alguno, eso prácticamente nos garantiza al 100 % que por mucho más que lo entrenemos no conseguiremos hacerlo converger.

En la figura 8.15 hay 20 episodios ejecutados con 5 tipos de cajas, de estos, 13 no han sido completados lo cual es más de la mitad, era de esperar observando la recompensa media de la curva de entrenamiento, y los que si ha resuelto, lo hace con más de 1000 pasos, rozando casi los 5000 en algunos, esto nos indica que el agente no es nada eficiente haciendo esta tarea.

Figura 8.13: Mapa de acciones en operativa mixta con distintos tipos de caja

Figura 8.14: Curva de entrenamiento con entorno 21x21 y 5 tipos de caja

Cajas por ordenar	Cajas por entregar	Cajas en el inventario del robot	Completado	Steps	Demandas
[0 0 0 0 0]	[0 0 0 0 0]	[0 0 0]	True	1966	1
[0 0 0 0 0]	[0 0 0 0 0]	[0 0 0]	True	1278	1
[0 3 1 2 4]	[4 5 4 5 5]	[5 0 0]	False	5000	3
[3 6 2 5 3]	[6 2 2 12 3]	[3 4 4]	False	5000	3
[6 5 1 6 1]	[6 4 2 0 3]	[0 0 0]	False	5000	3
[3 0 4 4 2]	[3 5 3 4 2]	[0 0 0]	False	5000	3
[0 0 0 0 0]	[0 0 0 0 0]	[3 1 3]	True	1461	1
[0 0 0 0 0]	[0 0 0 0 0]	[5 5 0]	True	1190	1
[5 7 0 3 5]	[5 3 3 4 5]	[3 4 2]	False	5000	3
[0 0 0 0 0]	[0 0 0 0 0]	[0 0 0]	True	3400	2
[1 3 1 3 3]	[3 1 6 11 3]	[4 5 0]	False	5000	3
[0 0 0 0 0]	[0 0 0 0 0]	[0 0 0]	True	4282	2
[0 0 0 0 0]	[0 0 0 0 0]	[0 0 0]	True	4502	2
[7 3 2 2 5]	[4 4 2 5 7]	[4 5 0]	False	5000	3
[0 0 2 5 5]	[2 2 7 2 1]	[0 0 0]	False	5000	3
[2 2 6 1 1]	[4 3 3 2 3]	[2 4 2]	False	5000	3
[4 1 2 7 8]	[4 8 6 2 3]	[4 2 3]	False	5000	3
[4 5 2 4 1]	[1 2 4 1 2]	[5 2 2]	False	5000	3
[4 1 2 7 1]	[5 6 0 3 0]	[0 0 0]	False	5000	3
[2 4 4 6 2]	[5 3 5 3 3]	[1 4 0]	False	5000	3

Figura 8.15: 20 episodios con 5 tipos de cajas

En resumen, los resultados son buenos hasta que le introducimos distintos tipos de cajas al agente, se ha intentado aumentar la complejidad del modelo, cambiar el sistema de recompensas y codificar de alguna forma distinta los tipos de cajas del entorno para el agente, pero ninguna daba buenos resultados, de hecho los empeoraba. Así que nos quedamos con el modelo que es capaz de realizar el objetivo con un tipo de caja genérico, esto es practicable en la vida real, el agente solamente se tiene que centrar en un tipo de caja a la vez, pero obviamente no conseguiremos el máximo absoluto en eficiencia si no, que será una mera aproximación, lo cual puede ser suficiente en algunos casos. Una forma de usar distintos tipos de cajas la cual se discute en trabajos futuros sería la división de tareas, es decir, una red neuronal que sí tenga información sobre todos los tipos de cajas y esta solamente se encargue de asignar tareas a los otros agentes, como si fuera una especie de administrador, así es como lo solucionan en el paper [5], por falta de tiempo y recursos no ha sido posible implementar esta técnica, ya que las 2 redes se tiene que entrenar en el mismo paso de backpropagation y era complicado de implementar.

En la tabla 8.4 podemos encontrar una tabla resumen de los 4 objetivos o aproximaciones estudiadas.

Tabla 8.4: Tabla resumen aproximaciones

ID Objetivo	Tamaño entorno	Cajas a entregar	Cajas a recoger	Tipos de cajas	Recompensa media	Tasa de éxito
1	101x101	4 a 30	0	1	1075.34	100 %
2	101x101	0	4 a 30	1	1068.54	100 %
3	51x51	20 a 50	20 a 50	1	1309.80	100 %
4	21x21	20 a 50	20 a 50	3	967.43	97 %
4	31x31	20 a 50	20 a 50	3	638.48	87 %
4	21x21	20 a 50	20 a 50	5	354.97	43 %

CAPÍTULO 9
Conclusiones

En conclusión, este trabajo enfatiza la importancia de una gestión eficaz de la logística de un almacén, de sus entradas y salidas y su distribución de los bienes por todo el almacén para proporcionar una mejora en la eficiencia de la logística de una empresa. Un factor clave para lograr este objetivo es que el agente involucrado sepa gestionar de forma eficiente todas las entradas y salidas necesarias del almacén y cumplir la demanda propuesta en cada momento, lo que puede ayudar a optimizar el funcionamiento en general del almacén y la empresa, aumentar la velocidad de procesamiento de los pedidos y disminuir los tiempos de espera de los camiones que se encargan de cargar y descargar los bienes, beneficiando también a los camioneros. Sin embargo, en este ámbito no se suelen abordar estos problemas con machine learning o de la forma como nosotros los hemos hecho, sino que se utilizan algoritmos más clásicos y costosos o el propio ingenio humano, y cada empresa suele tener su propia forma de hacerlo y no la comparten por miedo a que sus competidores puedan mejorar. Por lo tanto, este estudio está destinado a proponer una forma "innovadora" y distinta de abordar este problema o problemas similares de entregar y recogida en entornos grid usando técnica de machine learning y aprendizaje por refuerzo tanto para la gestión del almacén y control del agente robótico, principalmente una de la idea principal es eliminar el factor humano en esta tarea. La principal contribución del estudio es proponer un modelo, arquitectura, espacio de observación y sistema de recompensas para que un agente robótico totalmente autónomo puede gestionar un almacén por completo y sus simular la demanda que pueda haber en este entorno para ver como se desenvuelve el agente.

Los objetivos alcanzados durante este proyecto son los siguientes:

- En primer lugar, se creó un entorno donde el agente pudiera interactuar y llevar a cabo su tarea. Este entorno debe ser lo más parecido a la realidad, por lo tanto, se llevó a cabo un estudio de como funcionan estos almacenes en profundidad, extrayendo información de Inditex que nos enseñó exactamente como funciona en concreto su almacén logístico, se estudiaron otras empresas, pero todas tenían un funcionamiento similar y al estar nuestro trabajo basado en Inditex este fue el entorno elegido para basar nuestra simulación. La simulación debe ser próxima a la realidad, pero también fácilmente computable y las situaciones reproducibles
- Una vez creado el entorno se debe investigar que "datos" necesita el agente para poder realizar la tarea, estos "datos" en aprendizaje por refuerzo se llaman espacio de observación, como el propio nombre indica lo que observa el agente, por ejemplo, los humanos usamos nuestros ojos, oídos, tacto, etc. Para obtener información de nuestro entorno y hacer decisiones. Lo mismo pasa para nuestro robot, se llevó a cabo una búsqueda intensiva de qué información se entregaba a estos agentes y como sé pre-procesaba. En la mayoría de casos el agente recibe información muy

detalla de su entorno inmediato, esto en nuestro caso fue traducido en como se encuentra su entorno en un 7x7 (si está ocupado, qué tipo de cajas, etc.) y aparte reciben información más general, como cuanto falta para conseguir el objetivo. Cuanta más información podamos aportar mejor.

- Esta información debe estar pre-procesada para que pueda ser manejada correctamente por una red neuronal, ya que sabemos que estas trabajan mejor con datos entre 0 y 1 o -1 y 1 y datos continuos, no discretos, por eso se llevan a cabo transformaciones como one-hot-encodings para adaptarlo y funciones de activación que nos acoten estos datos entre -1 y 1 como es la función de tangente hiperbólica.

- Adicionalmente, queremos que este agente pueda funcionar en situaciones más parecidas a la realidad posibles, entonces para ello la simulación funciona creado un entorno inicial y llenándolo de cajas proceduralmente. A partir de este punto el entorno no es modificado por otra cosa que no sea el robot y cada nueva orden de demanda parte desde el anterior estado del robot, así simulamos un trabajo continuo del agente, como se haría en un almacén real. Él se encuentra en un lugar del almacén, con unas cajas en el inventario, y se debe adaptar a esta situación para cumplir la demanda actual.

- Para hacer la vida un poco más fácil al agente, algunas observaciones se enmascaran, como por ejemplo las distintas cajas que ve, para el robot solamente existen 3 tipos de estados para las estanterías, que el lugar está vacío, que contiene una caja que debe ser procesada o contiene una caja que ya ha sido procesada y, por lo tanto, no nos interesa. Así solucionamos el problema con los cientos de tipos de cajas distintas que pueda haber en un almacén como es el de Inditex y el agente solamente debe centrarse en procesar las cajas que ve.

- Finalmente, los resultados indican que el agente puede resolver eficazmente entornos grandes con muchas órdenes de demanda, siempre y cuando usemos cajas genéricas, cuando usamos distintos tipos de cajas el modelo tiene problemas a la hora de resolver entornos grandes con tal cantidad de cajas de demanda. Por lo tanto, el sistema es capaz de funcionar perfectamente solamente si un tipo de caja a la vez es usado en el sistema, esto se puede hacer perfectamente, pero disminuye en gran medida la eficiencia del modelo.

En conclusión, este proyecto ha cumplido con éxito los objetivos que sirven como primer paso a la implementación de agentes completamente autónomos en los grandes almacenes de Inditex para que su compleja logística puede ser completamente automatizada. Si es verdad, el objetivo con distintos tipos de cajas no ha sido resuelto al nivel que nos gustaría, sino rebajado a un nivel más simple. Por el hecho de que únicamente funciona correctamente con un tipo de caja a la vez, la eficiencia de este sistema puede ser mejorada en gran medida, pero en este punto sabemos que es capaz crear un sistema completamente autónomo y que cumple con las demandas establecidas si le encargamos solamente con un tipo de caja a la vez.

9.1 Trabajos futuros

Aunque este proyecto ha logrado importantes avances en la gestión de la logística usando aprendizaje por refuerzo, existen varias vías para la futura exploración y mejora. A continuación se enumeran posibles áreas de investigación y desarrollo.

1. **entorno más realista** Como hemos mencionado anteriormente, el entorno era una versión muy simplificada de la realidad, discreta, en 2 dimensiones y con solamente una estantería. Sería una mejora bastante buena, tener un entorno en 3 dimensiones y continuo, incluso con la posibilidad de ser renderizado y visualizado. También la posibilidad de tener múltiples agentes en el mismo entorno para que interactúen entre ellos.

2. **Sistema multi-agente** Para tener una versión más eficiente capaz de gestionar todo el almacén completo, sería necesario tener un sistema multi-agente donde los distintos agentes tuvieran información sobre ellos y pudieran interactuar. Ya que tener un modelo para cada agente individualmente (aislado de los demás) no garantiza un mínimo absoluto en nuestra función de optimización, para obtener la máxima eficiencia todos los agentes debería estar interconectado y cooperar.

3. **División de tareas** Hasta ahora el agente se encarga de gestionar la demanda y su movimiento y acciones en el entorno a la vez, pero sería buena idea dividir estas redes. Así, una se encargaría por ejemplo de distribuir la demanda a través de los distintos agentes y asignar tareas, luego cada agente individualmente tendría una red que únicamente tendría que cumplir la tarea asignada por la red administradora.

4. **Escalabilidad** Este sistema está pensado para ser escalable, y nada nos dice que no lo es, en teoría podría trabajar en entornos muy grandes con muchísima cantidad de cajas y demandas muy altas y adaptarse a estos entornos. En la práctica, esto parece no ser así por los experimentos que se han realizado. Parece ser que el agente no trabaja bien en entornos muy complejos y si añadiésemos otra estantería nada nos garantiza que funcionaría el agente actual. Por limitaciones técnicas esto no se ha podido llevar a cabo en este trabajo, pero se debería de investigar para futuros proyectos.

5. **Embedding de las observaciones** En trabajos similares, lo que se hace para codificar las observaciones de una forma que las redes lo entiendan, es construir embeddings los cuales se encargarán de transformar las observaciones crudas en vectores más entendibles para una red neuronal, así no nos tenemos de preocupar que las observaciones estén correctamente representadas. Esto podría ayudar a hacer una representación más simple de los distintos tipos de cajas y que el agente pudiera funcionar con muchos tipos de cajas distintos.

9.2 Relación del trabajo desarrollado con los estudios cursados

Este proyecto debe su éxito a la carrera de ciencia de datos cursada en esta universidad. La formación integral recibida y las habilidades perfeccionadas a lo largo de cuatro académicos me han capacitado para aplicar y obtener valor de estos conocimientos en la práctica con eficacia. A continuación se destacarán las asignaturas que desempeñaron un papel clave en el desarrollo de este proyecto.

- Todos los modelos desarrollados han sido fruto de un amplio abanico de métodos y habilidades aprendidas en asignaturas como Modelos Estadísticos para la Toma de Decisiones I y II, Modelos Descriptivos y Predictivos I y II, Proyectos I a III, y Evaluación, Despliegue y Monitorización de Modelos.

- Destacar las asignaturas de Técnicas escalables en aprendizaje automático y Análisis de imágenes y vídeos en las cuales aprendí a como utilizar redes neuronales,

su funcionamiento, su entrenamiento y como deben ser tratados los datos para los distintos tipos de capas que existen. Como es el modelo principal usado en este trabajo, fueron muy útiles.

- Las asignaturas de programación y sus vertientes como son algorítmica y estructuras de datos me enseñaron a programar de forma correcta, limpia y eficiente en python tanto para crear en entorno como para programar las redes neuronales usadas.
- La asignatura de optimización por explicar como se aplican algoritmos de descenso por gradiente para optimizar parámetros de modelos como pueda ser una red neuronal.

Bibliografía

[1] Carlos Santana (DotCSV). *¡Redes Neuronales CONVOLUCIONALES! ¿Cómo funcionan?* URL: https://www.youtube.com/watch?v=V8j1oENVz00.

[2] Bowen Baker et al. *Emergent Tool Use From Multi-Agent Autocurricula*. 2020. arXiv: 1909.07528 [cs.LG].

[3] Christopher Berner et al. «Dota 2 with large scale deep reinforcement learning». En: *arXiv preprint arXiv:1912.06680* (2019).

[4] Kurt Hornik, Maxwell Stinchcombe y Halbert White. «Multilayer feedforward networks are universal approximators». En: *Neural networks* 2.5 (1989), págs. 359-366.

[5] Aleksandar Krnjaic et al. *Scalable Multi-Agent Reinforcement Learning for Warehouse Logistics with Robotic and Human Co-Workers*. 2022. arXiv: 2212.11498 [cs.LG].

[6] Y. LeCun et al. «Backpropagation Applied to Handwritten Zip Code Recognition». En: *Neural Computation* 1.4 (1989), págs. 541-551. DOI: 10.1162/neco.1989.1.4.541.

[7] Hokey Min. «Artificial intelligence in supply chain management: theory and applications». En: *International Journal of Logistics: Research and Applications* 13.1 (2010), págs. 13-39.

[8] Volodymyr Mnih et al. «Playing atari with deep reinforcement learning». En: *arXiv preprint arXiv:1312.5602* (2013).

[9] Hamid R Nemati et al. «Knowledge warehouse: an architectural integration of knowledge management, decision support, artificial intelligence and data warehousing». En: *Decision Support Systems* 33.2 (2002), págs. 143-161.

[10] SpinUp OpenAI. *Intro to Policy Optimization*. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html.

[11] A Pasumpon Pandian. «Artificial intelligence application in smart warehousing environment for automated logistics». En: *Journal of Artificial Intelligence* 1.02 (2019), págs. 63-72.

[12] Dinesha Perera, Uditha Mirando y Anushika Fernando. «WAREHOUSE SPACE OPTIMIZATION USING LINEAR PROGRAMMING MODEL AND GOAL PROGRAMMING MODEL». En: 1 (jun. de 2022), págs. 103-124.

[13] Marius-Constantin Popescu et al. «Multilayer perceptron and neural networks». En: *WSEAS Transactions on Circuits and Systems* 8 (jul. de 2009).

[14] John Schulman et al. «Proximal policy optimization algorithms». En: *arXiv preprint arXiv:1707.06347* (2017).

[15] Nisan Stiennon et al. «Learning to summarize with human feedback». En: *Advances in Neural Information Processing Systems* 33 (2020), págs. 3008-3021.

[16] Richard S Sutton y Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[17] TensorFlow. *Playing CartPole with the Actor-Critic method*. URL: https://www.tensorflow.org/tutorials/reinforcement_learning/actor_critic#:~:text=Actor-Critic%20methods%20are%20temporal,based%20on%20the%20given%20state..

[18] homas Tracey. *Language Translation with RNNs*. URL: https://towardsdatascience.com/language-translation-with-rnns-d84d43b40571.

[19] Michaela Urbanovská y Antonın Komenda. «Grid Representation in Neural Networks for Automated Planning.» En: *ICAART (3)*. 2022, págs. 871-880.

[20] Ashish Vaswani et al. «Attention is all you need». En: *Advances in neural information processing systems* 30 (2017).

[21] Oriol Vinyals et al. «Grandmaster level in StarCraft II using multi-agent reinforcement learning». En: *Nature* 575.7782 (2019), págs. 350-354.

APÉNDICE A

Relación del proyecto con los Objetivos de Desarrollo Sostenible

El trabajo descrito, que involucra el entrenamiento de un agente con aprendizaje por refuerzo para cumplir demandas autónomamente y de manera eficiente en un entorno virtual similar a la realidad, puede relacionarse con varios Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas. A continuación, se presentan algunos ODS que se pueden relacionar con este tipo de trabajo:

- ODS 9 - Industria, Innovación e Infraestructura: Este objetivo busca promover la construcción de infraestructuras resilientes, el fomento de la innovación y el fomento de la adopción de tecnologías limpias y sostenibles. El desarrollo y entrenamiento de agentes de inteligencia artificial, como el mencionado en el problema, puede contribuir a la innovación tecnológica y a la mejora de las infraestructuras.

- ODS 11 - Ciudades y comunidades sostenibles: Este objetivo tiene como objetivo lograr que las ciudades y los asentamientos humanos sean inclusivos, seguros, resilientes y sostenibles. Los avances en la tecnología de inteligencia artificial pueden ayudar a mejorar la planificación urbana, la gestión de la movilidad y la eficiencia energética, lo que contribuye a la creación de ciudades más sostenibles.

- ODS 12 - Producción y consumo responsables: Este objetivo busca promover patrones de producción y consumo sostenibles. Mediante el uso de algoritmos de aprendizaje por refuerzo, los agentes pueden aprender a optimizar recursos y minimizar el desperdicio, lo que contribuye a una producción y consumo más eficientes y responsables.

- ODS 13 - Acción por el clima: Este objetivo se centra en tomar medidas urgentes para combatir el cambio climático y sus impactos. El desarrollo de algoritmos y agentes de inteligencia artificial que puedan mejorar la eficiencia energética, la gestión de recursos y la mitigación de emisiones puede ayudar en la lucha contra el cambio climático.

Es importante tener en cuenta que la aplicación de la inteligencia artificial y el aprendizaje por refuerzo debe llevarse a cabo de manera ética y responsable, teniendo en cuenta los impactos sociales, económicos y ambientales. El desarrollo de políticas y marcos regulatorios sólidos también es crucial para garantizar que estas tecnologías se utilicen de manera sostenible y beneficiosa para la sociedad en su conjunto.

Tabla A.1: Impacto del trabajo en los Objetivos de Desarrollo Sostenible (ODS)

ODS	**Nivel de Impacto**
ODS 1 - Fin de la pobreza	Nulo
ODS 2 - Hambre cero	Nulo
ODS 3 - Salud y bienestar	Nulo
ODS 4 - Educación de calidad	Nulo
ODS 5 - Igualdad de género	Nulo
ODS 6 - Agua limpia y saneamiento	Nulo
ODS 7 - Energía asequible y no contaminante	Medio
ODS 8 - Trabajo decente y crecimiento económico	Medio a Alto
ODS 9 - Industria, Innovación e Infraestructura	Alto
ODS 10 - Reducción de las desigualdades	Bajo a Medio
ODS 11 - Ciudades y comunidades sostenibles	Medio a Alto
ODS 12 - Producción y consumo responsables	Medio a Alto
ODS 13 - Acción por el clima	Medio a Alto
ODS 14 - Vida submarina	Nulo
ODS 15 - Vida de ecosistemas terrestres	Nulo
ODS 16 - Paz, justicia e instituciones sólidas	Nulo
ODS 17 - Alianzas para lograr los objetivos	Nulo