UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Aprendizaje de agentes en entornos de Minecraft mediante modelos de Reinforcement Learning e Imitation Learning

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Sanfélix Enguídanos, David

Tutor/a: Palanca Cámara, Javier

Cotutor/a: Julian Inglada, Vicente Javier

CURSO ACADÉMICO: 2022/2023

Resum

Aquest treball aprofundeix en els intricats camps de l'Aprenentatge Automàtic, amb un enfocament particular en l'Aprenentatge per Reforç i l'Imitation Learning. Utilitzant Minecraft, un joc tipus "sandbox", com a escenari principal, es trau partit del conjunt de dades de MineRL per a dur a terme l'entrenament d'agents intel·ligents. Aquest treball té com a objectiu fusionar els conceptes teòrics de l'aprenentatge automàtic amb els reptes pràctics que sorgeixen en entorns de jocs com Minecraft. El propòsit és no sols comprendre les tècniques essencials d'aprenentatge, sinó també implementar-les en un entorn que permet destacar el seu potencial i els seus desafiaments inherents.

Paraules clau: Machine Learning, Aprenentatge per reforç, Aprenentatge per imitació, Minecraft, MineRL

Resumen

Este trabajo profundiza en los intrincados campos del Aprendizaje Automático, con un enfoque particular en el Aprendizaje por Refuerzo y el Imitation Learning. Utilizando Minecraft, un juego "sandbox", como escenario principal, se saca partido del conjunto de datos de MineRL para llevar a cabo el entrenamiento de agentes inteligentes. Este trabajo tiene como objetivo fusionar los conceptos teóricos del aprendizaje automático con los retos prácticos que surgen en entornos de juegos como Minecraft. El propósito es no solo comprender las técnicas esenciales de aprendizaje, sino también implementarlas en un entorno que permite destacar su potencial y sus desafíos inherentes.

Palabras clave: Machine Learning, Aprendizaje por refuerzo, Aprendizaje por imitación, Minecraft, MineRL

Abstract

This work delves into the intricate fields of Machine Learning, with a particular focus on Reinforcement Learning and Imitation Learning. Using Minecraft, a sandbox game, as the primary application domain, it takes advantage of the MineRL dataset to perform the training of intelligent agents. This work aims to merge the theoretical concepts of machine learning with the practical challenges that arise in game environments such as Minecraft. The purpose is not only to understand the essential learning techniques, but also to implement them in an environment that allows highlighting their potential and inherent challenges.

Key words: Machine Learning, Reinforcement Learning, Imitation Learning, Minecraft, MineRL

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

1.1 Motivación

La motivación que impulsa este trabajo radica en varios factores. El primero de ellos es el hecho de que, este último año, tuve la oportunidad de formar parte de un diploma de extensión universitaria en Inteligencia Artificial, un campo de la informática que ha cautivado mi imaginación y mi interés profesional desde entonces. La posibilidad de desarrollar sistemas de IA que puedan enfrentarse a problemas complejos, aprender de ellos y evolucionar ha despertado en mí una profunda curiosidad.

Por otra parte, Minecraft siempre ha sido un juego al que le he tenido un cariño especial y que aún disfruto a día de hoy. Más allá de ser una fuente de entretenimiento, ofrece un entorno excepcionalmente versátil y dinámico, lo que lo convierte en un campo de pruebas ideal para nuestros agentes de IA. Al tener que enfrentarse a retos cambiantes y a menudo imprevistos, los agentes deben aprender a adaptarse y tomar decisiones eficaces, de una forma similar a como lo harían en el mundo real.

En este contexto, surgió la idea de fusionar mis dos grandes intereses: los videojuegos y la inteligencia artificial. La combinación de estos dos mundos en mi trabajo de fin de grado me ofreció la oportunidad única de explorar cómo los modelos de aprendizaje por refuerzo e imitación pueden aplicarse en el entorno dinámico y complejo de Minecraft. A través de este proyecto, he buscado no solo expandir mis conocimientos en IA, sino también contribuir a la creciente intersección de estas dos disciplinas, además de explorar y comprobar el estado actual de las mismas.

1.2 Objetivos

El propósito principal de este trabajo es explorar cómo los agentes de inteligencia artificial pueden aprender y adaptarse a un entorno tan dinámico y desafiante como lo es el universo de Minecraft, realizando un estudio de la situación actual de los modelos de Aprendizaje por refuerzo e Imitation Learning, además de realizar pruebas dentro de Minecraft para comprobar y analizar los resultados.

El primer objetivo es adquirir una comprensión profunda de estas dos técnicas de *Machine Learning*. Se estudiarán sus principios fundamentales, sus ventajas y desventajas, y cómo se han aplicado hasta la fecha en el campo del aprendizaje automático, y más específicamente dentro de Minecraft. La comprensión de estos aspectos es esencial para poder aplicar estas técnicas de manera eficiente y poder comprender los resultados.

Como segundo objetivo, nos proponemos desarrollar agentes de inteligencia artificial capaces de operar en el entorno de Minecraft. Para lograr este objetivo haremos uso principalmente de dos librerías de Python, *stable baselines3* [65] y *MineRL* [1], las cuales explicaremos en profundidad en capítulos posteriores.

Por último, nos gustaría llevar acabo una serie de pruebas y experimentos que implementen estas tecnologías, para posteriormente poder evaluar y contrastar los resultados. Esto permitirá identificar las limitaciones y desafíos encontrados durante el estudio, y a partir de esto, se podrán proponer posibles soluciones y mejoras para futuras investigaciones.

A través de estos objetivos, aspiramos a contribuir al campo del aprendizaje automático y la inteligencia artificial asentando las bases de las capacidades actuales de estas tecnologías, a avanzar en nuestro entendimiento de cómo los agentes pueden aprender en entornos dinámicos y complejos, y a abrir nuevas vías de investigación y aplicación para estas tecnologías dentro de nuestra universidad.

1.3 Estructura de la memoria

Esta memoria se divide en los siguientes capítulos:

- **Introducción:** Este capítulo establece el contexto del trabajo, proporcionando una motivación para la investigación y estableciendo los objetivos específicos que se buscan alcanzar, además de presentar una breve descripción de cómo está organizada la memoria.
- **Conceptos clave:** Aquí se introducirán los fundamentos teóricos y prácticos que son esenciales para comprender el resto del trabajo. Se abordarán temas como el Aprendizaje por Refuerzo, el Aprendizaje por Imitación y la relevancia de Minecraft en el contexto de estas áreas de aprendizaje automático.
- **Estado del arte:** Este capítulo revisa la literatura y las investigaciones actuales relacionadas con el tema de estudio. Se destacarán los avances en aprendizaje por refuerzo y por imitación en videojuegos, especialmente en Minecraft, destacando las fortalezas y debilidades de cada aproximación. También se presentarán herramientas y plataformas relevantes en cada campo.
- **Pruebas:** En este capítulo se pondrán a prueba los algoritmos de Reinforcement Learning e Imitation Learning para comprobar su funcionamiento en Minecraft, analizando los resultados y comparándolos con los esperados.
- **Conclusiones:** Se recapitularán los hallazgos y resultados del trabajo, destacando las contribuciones principales y ofreciendo una perspectiva sobre posibles futuros trabajos en el área.

CAPÍTULO 2
Conceptos clave

Para poder comprender este proyecto es necesario introducir los tres conceptos principales: el Aprendizaje por Refuerzo, el Aprendizaje por Imitación y, por último, Minecraft. A continuación proporcionaremos una explicación más en profundidad acerca de cada uno, dejando de esta manera claro el núcleo del proyecto.

2.1 Aprendizaje por refuerzo (Reinforcement Learning, RL)

El aprendizaje por refuerzo es una rama del aprendizaje automático que estudia cómo un agente puede aprender a tomar decisiones, optimizando una función de recompensa acumulativa a lo largo del tiempo. La idea esencial es que el agente tome acciones en un entorno para alcanzar una meta, recibiendo retroalimentación en forma de recompensas.

2.1.1. Componentes

Los componentes principales en RL son:

- **Agente**: La entidad que toma decisiones.
- **Entorno**: Todo lo que el agente no controla y con lo que interactúa.
- **Acciones**: Las decisiones que puede tomar el agente.
- **Estados**: Representan la situación actual del agente.
- **Recompensa**: Retroalimentación que recibe el agente tras tomar una acción en un entorno.

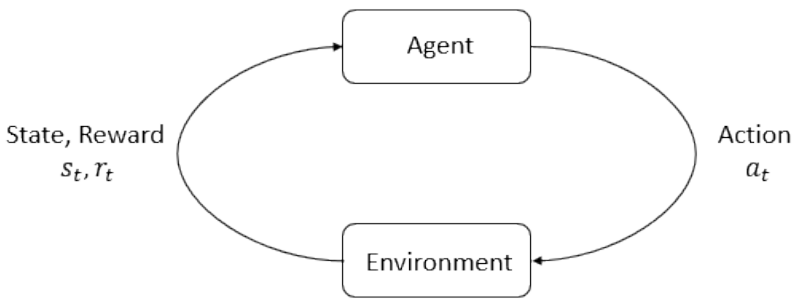

Figura 2.1: Bucle de interacción agente-entorno. Fuente: [2]

Para una introducción detallada de estos componentes, se puede consultar el capítulo 1 de este libro [3].

2.1.2. Proceso de decisión de Markov y función de Bellman

Un Proceso de Decisión de Markov (MDP) ofrece una formulación matemática para describir cómo un agente interactúa con su entorno, toma decisiones y recibe recompensas. Un MDP se define como una tupla (S, A, P, R), donde:

- S: Conjunto de estados (**S**tates).
- A: Conjunto de acciones (**A**ctions).
- P: Probabilidad de transición entre estados (**P**robability).
- R: Función de recompensa (**R**eward).

Para una introducción más en detalle acerca de los MDP, recomiendo consultar el mismo capítulo 1 [3].

La relación fundamental que define la planificación y toma de decisiones óptimas en un MDP es la **ecuación de Bellman**. Esta ecuación establece la relación entre el valor de un estado y el valor esperado acumulado en el futuro al tomar decisiones óptimas. Para un estado s y una acción a, la ecuación de Bellman se expresa como:

$$V(s) = \max_a \sum_{s',r} P(s', r|s, a)[r + \gamma V(s')]$$

Donde $V(s)$ es el valor del estado s, a es una acción tomada en el estado s, s' es el próximo estado posible, r es la recompensa asociada con la transición, $P(s', r|s, a)$ es la probabilidad de transición y γ es el factor de descuento.

En el marco de este proyecto, entender la función de Bellman es esencial, ya que proporciona una base matemática sólida para la optimización y la toma de decisiones en entornos de aprendizaje por refuerzo. Los algoritmos que se aplicarán para entrenar al agente se basan en esta ecuación para maximizar las recompensas acumuladas a lo largo del tiempo. Comprender cómo la función de Bellman se aplica a los MDP permitirá una implementación más eficaz de los métodos de aprendizaje por refuerzo y una mejor adaptación a los desafíos que presenta el entorno de Minecraft. Para una comprensión más profunda de la ecuación de Bellman, se puede consultar el capítulo 5 del mismo libro [3].

2.1.3. Algoritmos y métodos

Dentro del aprendizaje por refuerzo, existen dos categorías principales de métodos: Model-free y Model-based (ver Figura 2.2).

- **Model-Free**: Estos métodos no utilizan un modelo del entorno y aprenden directamente de la experiencia. Son típicamente más simples y requieren menos supuestos sobre la naturaleza del entorno.
- **Model-Based**: A diferencia de los métodos Model-free, los Model-based emplean un modelo del entorno. Esto permite una planificación más eficiente pero puede ser más complejo y sensible a la precisión del modelo.

En nuestro caso nos centraremos en la rama de los métodos Model-Free, dentro de la cual encontramos dos tipos de algoritmos diferentes: los *Value-Based Methods* y los *Policy-Based Methods*.

Value-Based Methods

Los métodos basados en el valor en el aprendizaje por refuerzo se enfocan en aprender una **función de valor**, que cuantifica qué tan buena es una acción o un estado en términos de la recompensa acumulada esperada. A continuación, se describen algunos de los métodos más relevantes:

- **Q-Learning**: Es uno de los métodos basados en el valor más conocidos [4], donde la idea es aprender una función de valor-acción $Q(s, a)$ que representa la recompensa esperada al tomar una acción a en un estado s y luego seguir una política óptima. La actualización de la función Q se realiza mediante la ecuación:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{2.1}$$

  donde α es la tasa de aprendizaje, γ es el factor de descuento, r es la recompensa, s' es el estado siguiente, y a' son las acciones posibles en s'.

- **Deep Q-Networks (DQN)**: Es una extensión de Q-learning que utiliza redes neuronales profundas para aproximar la función Q [5]. Para estabilizar el aprendizaje, DQN utiliza una técnica llamada *experience replay*, donde las transiciones pasadas se almacenan en un búfer y se muestrean aleatoriamente durante el entrenamiento. También se emplea una red objetivo para calcular el valor objetivo de la actualización, reduciendo así la oscilación en el entrenamiento.

  - **Variantes de DQN**: Existen varias mejoras y variantes de DQN, incluyendo:
    - **Double DQN**: Reduce el sesgo de sobreestimación en la selección de acciones al desacoplar la selección y evaluación de acciones, utilizando una red separada para cada uno [6].
    - **Prioritized Experience Replay**: Da prioridad a las transiciones más informativas en el proceso de muestreo [7], asegurando que los ejemplos más relevantes sean utilizados con mayor frecuencia en el entrenamiento.
    - **Dueling DQN**: Divide la función Q en una función de valor y una función de ventaja [8], permitiendo una representación más flexible y facilitando el aprendizaje de la función de valor en estados donde las acciones no difieren mucho en su valor.

  Estas variantes de DQN son fundamentales para mejorar la eficiencia y robustez del algoritmo original, y han sido clave en la aplicación exitosa de DQN en diversos problemas complejos, como el control de agentes en juegos y simulaciones.

Policy-Based Methods

Los métodos basados en políticas en aprendizaje por refuerzo se enfocan en aprender directamente una **política**, que es una función que asigna una acción a cada estado en un entorno. A diferencia de los métodos basados en el valor que tratan de aprender una función de valor y derivar una política de ella, los métodos basados en políticas optimizan directamente la política. A continuación, se describen algunos de los métodos más importantes:

- **Policy Gradient Methods**: Los métodos de gradiente de políticas se enfocan en optimizar una política calculando el gradiente de la recompensa esperada respecto a los parámetros de la política, utilizando técnicas de optimización para ajustar estos parámetros en la dirección que mejora el rendimiento. Algunos de los algoritmos más populares son los siguientes:

  - **REINFORCE**: Es un método de gradiente de política simple y potente. Utiliza la retroalimentación en forma de recompensa para ajustar los parámetros de la política en la dirección que incrementa la recompensa esperada. La actualización se realiza mediante la fórmula:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t|s_t, \theta) \cdot (R_t - b) \tag{2.2}$$

    donde θ son los parámetros de la política, $\pi(a_t|s_t, \theta)$ es la probabilidad de tomar la acción a_t en el estado s_t, y R_t es la recompensa acumulada desde el tiempo t [9].

  - **Actor-Critic Methods**: Esta familia de métodos combina la idea de los métodos de gradiente de política (Actor) con la estimación del valor (Critic). Esto permite una reducción de la varianza y mejora la eficiencia del aprendizaje. Un ejemplo es el algoritmo Advantage Actor-Critic (A2C) [10].

    - **Deep Deterministic Policy Gradients (DDPG)**: Es una variante del algoritmo de Actor-Critic diseñada para entornos continuos. Utiliza redes neuronales profundas para representar tanto al actor como al crítico [11].

  - **Proximal Policy Optimization (PPO)**: PPO es un algoritmo que busca equilibrar la eficiencia de la muestra y la facilidad de implementación. Utiliza técnicas para garantizar que las actualizaciones de la política no sean demasiado grandes, lo que ayuda en la estabilidad del entrenamiento [12].

- **Trust Region Methods**: Estos métodos establecen una región de confianza dentro de la cual se optimiza la política. La idea es evitar actualizaciones excesivamente grandes que puedan dañar el rendimiento. Un ejemplo famoso es Trust Region Policy Optimization (TRPO) [13].

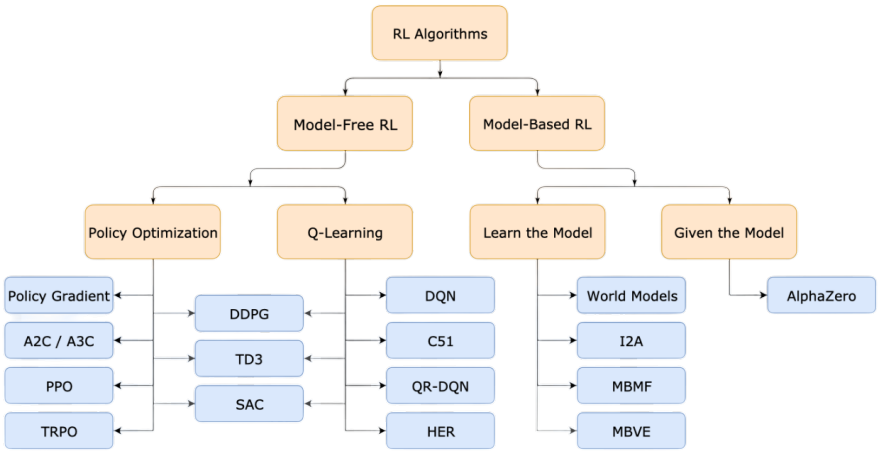

Figura 2.2: Esquema de algoritmos de Reinforcement Learning más relevantes. Fuente: [2]

Aprendizaje por refuerzo multiagente: Por último cabe destacar el *aprendizaje por refuerzo multiagente* (RLM, por sus siglas en inglés). El RLM es un subcampo del aprendizaje por refuerzo (RL) en el cual múltiples agentes aprenden a través de interacciones con su entorno. En contraste con el RL tradicional, donde un único agente toma decisiones, el RLM se centra en escenarios donde múltiples agentes cooperan, compiten o coexisten en un entorno compartido. Esta dinámica puede llevar a comportamientos emergentes, como el uso de herramientas que no fueron enseñadas explícitamente, gracias a la interacción y competencia entre agentes [14].

Uno de los retos más fascinantes en RLM es la coordinación entre agentes en dilemas sociales secuenciales. En estos contextos, el objetivo individual de un agente puede estar en conflicto con el bienestar colectivo. Sin embargo, a través del aprendizaje reforzado, es posible que los agentes desarrollen estrategias que equilibren sus intereses individuales con los del grupo [15].

2.1.4. Aplicaciones y casos de uso

El aprendizaje por refuerzo (RL) se ha aplicado en una amplia gama de dominios y ha demostrado ser una técnica poderosa en áreas como: **robótica**, donde los robots pueden aprender a realizar tareas complejas, desde manipular objetos delicados hasta caminar en terrenos irregulares [16]; **juegos**, donde en videojuegos y juegos de mesa, los agentes de RL han alcanzado y superado el rendimiento humano en juegos como Go, ajedrez y póker [17]; **finanzas**, donde RL se ha utilizado para el comercio algorítmico y la gestión de carteras, optimizando decisiones financieras basadas en la evolución de los mercados [18]; **salud**, donde en el sector médico, RL puede ayudar en la personalización de tratamientos, ajustando las intervenciones médicas según la evolución del paciente [19]; y **control de sistemas**, donde desde la gestión de recursos energéticos hasta el control de tráfico, RL ofrece soluciones para optimizar sistemas complejos y dinámicos [20].

2.1.5. Desafíos y limitaciones

A pesar de su potencial y éxito en diversas aplicaciones, el aprendizaje por refuerzo enfrenta varios desafíos y limitaciones [21, 22]:

- **Necesidad de datos:** El RL puede requerir una gran cantidad de datos para aprender una política eficaz, lo que puede ser costoso o impracticable en algunos entornos.
- **Exploración vs. Explotación:** Equilibrar la exploración de nuevas acciones con la explotación de acciones conocidas es un desafío clave en RL y puede afectar la eficacia del aprendizaje [23].
- **Transferencia de conocimiento:** La aplicación de políticas aprendidas en un entorno a otro ligeramente diferente puede ser compleja, lo que limita la capacidad del RL para generalizar [24].
- **Estabilidad y convergencia:** Los algoritmos de RL pueden ser sensibles a la elección de hiperparámetros y no siempre convergen a una solución óptima.
- **Seguridad y robustez:** En aplicaciones críticas, como puede ser el caso de la conducción autónoma, los errores de RL pueden tener consecuencias graves, lo que plantea desafíos en la garantía de seguridad y robustez.

2.2 Aprendizaje por imitación (Imitation Learning, IL)

Por otro lado, el *Aprendizaje por Imitación*, es una técnica de aprendizaje automático donde el agente aprende a **replicar las acciones de un experto**, en lugar de aprender mediante recompensas. En el aprendizaje por imitación, se proporciona al agente un conjunto de "demostraciones", que son ejemplos de comportamiento experto, y el objetivo del agente es aprender a imitar ese comportamiento lo más cerca posible. El aprendizaje por imitación es particularmente útil en situaciones donde es difícil definir una función de recompensa adecuada o donde la interacción directa con el entorno puede ser costosa o arriesgada.

2.2.1. Métodos principales

Behavior Cloning

El *Behavior Cloning* es un enfoque supervisado en el cual se entrena al agente para mapear directamente las observaciones del entorno a las acciones del experto [25]. La esencia detrás de este método es sencilla, dadas las demostraciones del experto, se dividen en pares de estado-acción y se tratan como ejemplos i.i.d[1]. Posteriormente, se aplica aprendizaje supervisado. La función de pérdida, que puede variar según la aplicación, se denota como L. Así, el algoritmo se resume de la siguiente manera:

1. Recoger las demostraciones (τ^* trayectorias) del experto.
2. Tratar las demostraciones como pares estado-acción i.i.d.: $(s_0)^*, (a_0)^*$, $(s_1)^*, (a_1)^*$, etc.
3. Aprender la política mediante aprendizaje supervisado minimizando la función de pérdida:
$$L\left(a^*, \pi_\theta(s)\right).$$

En algunas aplicaciones, el *behavioral cloning* puede funcionar de manera excelente. Sin embargo, para la mayoría de los casos, este método puede ser problemático. La principal razón de esto es la suposición de i.i.d.: mientras que el aprendizaje supervisado asume que los pares de estado-acción están distribuidos de manera i.i.d., en un MDP una acción en un estado dado induce el siguiente estado, rompiendo la suposición anterior.

Esto también significa que los errores cometidos en diferentes estados se acumulan. Por lo tanto, un error cometido por el agente puede llevarlo fácilmente a un estado que el experto nunca ha visitado y sobre el cual el agente nunca se ha entrenado. En tales estados, el comportamiento es indefinido, lo que puede llevar a fallos catastróficos.

DAgger (Dataset Aggregation): Para abordar algunos de los problemas inherentes al Behavior Cloning, se introdujo DAgger. Esta es una técnica que combina las trayectorias del experto con las trayectorias del agente para entrenar al modelo. La idea principal detrás de Dagger es que, en lugar de entrenar solo en las demostraciones del experto, el agente también se entrena en las áreas donde es probable que actúe, reduciendo así la discrepancia entre la política del experto y la política aprendida por el agente.

El proceso de DAgger es iterativo:

[1] i.i.d. significa ïndependientes e idénticamente distribuidos", lo que implica que cada muestra proviene de la misma distribución probabilística y es independiente de las demás muestras.

1. El agente sigue la política actual para recoger nuevas trayectorias.
2. Las acciones para estas nuevas trayectorias son etiquetadas por el experto.
3. Estas trayectorias etiquetadas se agregan al conjunto de datos original.
4. El agente se reentrena en el conjunto de datos combinado.
5. Se repiten los pasos anteriores hasta la convergencia.

Con DAgger, el agente no solo aprende a imitar al experto, sino que también recibe retroalimentación sobre cómo corregir sus propias acciones cuando se desvía de las trayectorias del experto, lo que lo hace más robusto en situaciones no observadas directamente en las demostraciones del experto [27].

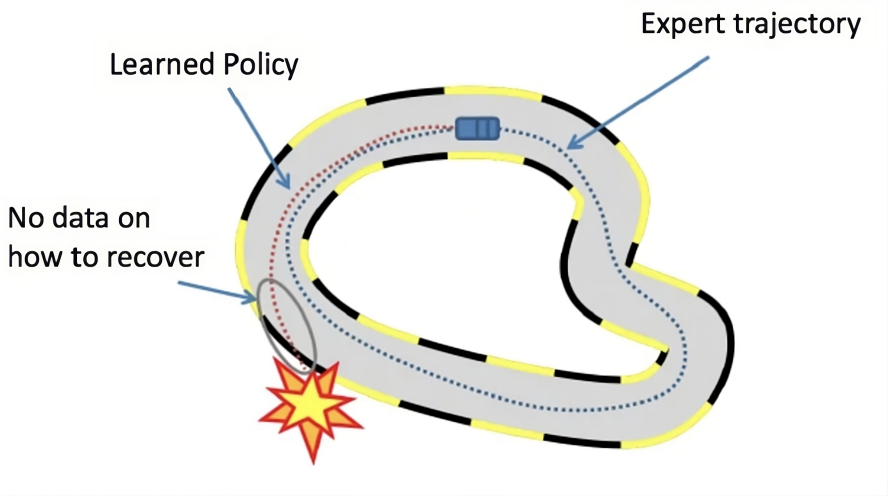

Figura 2.3: Behavioural cloning puede fallar si el agente comete un error. Fuente: [26]

Inverse Reinforcement Learning

El Aprendizaje por Refuerzo Inverso (IRL) es una técnica avanzada dentro del ámbito del Aprendizaje por Imitación que busca entender las decisiones tomadas por un experto. En lugar de simplemente replicar las acciones, IRL infiere las recompensas subyacentes que podrían haber llevado al experto a tomar esas decisiones. Mientras que técnicas como el "Behavior Cloning"se centran en imitar directamente las acciones observadas de un experto, IRL va un paso más allá: intenta **descubrir la función de recompensa que el experto parece estar optimizando** y, posteriormente, utiliza esa función de recompensa para guiar el comportamiento del agente. Esta aproximación es particularmente útil en situaciones donde las recompensas reales que guían a un experto son desconocidas o son difíciles de especificar.

Dentro de la familia de algoritmos IRL, GAIL y AIRL han ganado relevancia en años recientes:

GAIL (Generative Adversarial Imitation Learning): GAIL es un enfoque que combina los principios del IRL con técnicas de aprendizaje generativo adversario [28]. En lugar de intentar inferir directamente una función de recompensa, GAIL establece un juego adversario entre dos redes: un imitador y un discriminador. El imitador busca generar acciones que se asemejen a las del experto, mientras que el discriminador intenta distinguir entre

las acciones del experto y las del imitador. A través de iteraciones sucesivas, el imitador mejora su capacidad para mimetizar el comportamiento del experto sin necesitar una función de recompensa explícita.

AIRL (Adversarial Inverse Reinforcement Learning): Mientras que GAIL y AIRL ambos utilizan técnicas adversarias, su enfoque y objetivos son distintos. Mientras GAIL busca aprender directamente una política que imite al experto sin inferir una función de recompensa explícita, AIRL intenta recuperar esta función de recompensa subyacente que motiva las acciones del experto [29].

En AIRL, el proceso adversario se centra en la función de recompensa. Aquí, hay un discriminador que, al igual que en GAIL, intenta diferenciar entre las trayectorias del experto y las del agente. Sin embargo, la diferencia clave es que en AIRL, el discriminador se utiliza para inferir una función de recompensa. Esta función de recompensa inferida se usa luego para entrenar al agente. Esencialmente, AIRL se esfuerza por responder a la pregunta: "¿Qué recompensas podría estar buscando el experto para actuar de esta manera?". Una vez que esta función de recompensa se ha inferido, se utiliza para guiar el aprendizaje del agente, permitiendo que el agente actúe de manera similar al experto en una variedad de situaciones, incluso en aquellas que no se observaron directamente en los datos del experto.

Al intentar desentrañar la función de recompensa subyacente, AIRL proporciona una comprensión más profunda del comportamiento del experto, permitiendo interpretaciones más ricas y transferencia de conocimiento a situaciones no observadas.

2.2.2. Aplicaciones y casos de uso

El *Imitation Learning* ha emergido como una técnica esencial en diversas áreas. En **robótica**, IL ha sido empleado para enseñar a robots a realizar tareas difíciles de programar manualmente, como el plegado de ropa o la manipulación de objetos delicados, aprendiendo movimientos y estrategias específicas de demostraciones humanas [33]. En la **conducción autónoma**, es fundamental para entrenar vehículos autónomos a aprender comportamientos de conducción seguros y eficientes [34]. En la industria de los **videojuegos**, se utiliza para entrenar personajes no jugadores (NPCs) a imitar tácticas de jugadores expertos, creando adversarios más realistas [35]. Por último en **medicina**, IL ayuda a entrenar sistemas de asistencia en procedimientos médicos, como la realización de cirugías o diagnósticos, observando a médicos expertos en acción [36].

2.2.3. Desafíos y limitaciones

El Aprendizaje por Imitación, aunque prometedor y efectivo en muchos casos, también enfrenta ciertos desafíos y limitaciones que deben tenerse en cuenta [37]:

- **Dependencia de datos de expertos:** La recolección de datos de demostración de calidad de expertos humanos puede ser un proceso laborioso y costoso. La calidad y diversidad de estos datos pueden afectar significativamente el rendimiento del agente.

- **Sensibilidad a errores:** Los métodos como el *Behavior Cloning* pueden ser especialmente sensibles a los errores en las demostraciones de los expertos. Un pequeño error puede propagarse y llevar a un mal rendimiento en tareas posteriores.

- **Generalización:** La capacidad del agente para generalizar a partir de las demostraciones a situaciones no vistas puede ser limitada. Esto puede requerir más ejemplos de entrenamiento o técnicas de regularización.

- **Comprensión de la intención del experto:** Los métodos de IL pueden enfrentar dificultades para comprender y replicar las intenciones subyacentes del experto, especialmente en tareas complejas y polifacéticas.

2.3 Minecraft: Más que un simple juego

Minecraft, que fue lanzado por *Mojang* el *17 de mayo de 2009* y adquirido más tarde por *Microsoft* [38], es un juego de tipo mundo abierto[2] que se basa en la construcción y exploración de un mundo compuesto por bloques. Cada bloque representa un tipo de material o elemento, como tierra, piedra, agua, o diferentes tipos de minerales, que pueden ser usados, alterados o manipulados por los jugadores. La naturaleza procedimental del mundo significa que cada partida ofrece un entorno único y diverso para explorar.

El mundo de Minecraft está habitado por una variedad de animales y criaturas, algunos de los cuales pueden ser domesticados, cazados, o incluso hostiles. Desde vacas y ovejas hasta zombis y esqueletos, estos seres añaden dinamismo y desafíos al juego. La diversidad se extiende a los paisajes, que varían desde densos bosques y altas montañas hasta vastos océanos y áridos desiertos. La diversidad de biomas[3] añade complejidad y oportunidades para distintas estrategias de supervivencia y construcción (ver Figura 2.4).

Aunque el juego ofrece una gran libertad, también existen objetivos como derrotar ciertos enemigos, explorar estructuras generadas automáticamente, y llegar a dimensiones alternativas. Estos objetivos guían la progresión del jugador y ofrecen metas desafiantes.

Uno de los aspectos más destacados de Minecraft es su enfoque en la creatividad [39, 40]. Los jugadores tienen la libertad de construir y diseñar estructuras y mecanismos complejos utilizando una variedad de materiales y herramientas. Desde casas sencillas hasta ciudades enteras y máquinas complejas, las posibilidades son casi infinitas. Además, Minecraft también ofrece diferentes modos de juego, como el Modo Supervivencia, donde los jugadores deben mantenerse vivos y gestionar recursos, y el Modo Creativo, donde pueden construir libremente sin restricciones. Esto permite una variedad de enfoques y estilos de juego, ofreciendo un entorno complejo y adaptable que puede ser un espacio de experimentación, aprendizaje, y expresión tanto para jugadores casuales como investigadores, además de servir como herramienta de aprendizaje [41].

2.3.1. Características clave

- **Mundo procedural:** Minecraft genera mundos de manera procedimental[4], lo que significa que no hay dos mundos exactamente iguales. Esta variabilidad permite que los algoritmos de aprendizaje automático sean entrenados y evaluados en una amplia variedad de entornos.

[2] En los videojuegos, un "mundo abierto" se refiere a un entorno virtual en el que los jugadores pueden explorar y abordar los objetivos libremente, a diferencia de un entorno estructurado con caminos fijos.

[3] Comunidad de plantas y animales que tienen características comunes para el medio ambiente en el que existen.

[4] La generación procedimental se refiere a la creación de contenido automáticamente mediante algoritmos, en lugar de manualmente, asegurando que cada mundo sea único.

Figura 2.4: Observación en primera persona de Minecraft.

- **Interacción compleja:** Los jugadores pueden recolectar recursos, construir estructuras, combatir enemigos y más. Estas interacciones multifacéticas ofrecen una amplia gama de tareas para el entrenamiento de agentes inteligentes.

  La capacidad de craftear[5] objetos en Minecraft representa un desafío jerárquico con objetivos subyacentes. Por ejemplo, para construir un pico de diamante, un jugador primero necesita recolectar diamantes y palos. Pero para poder recolectar diamantes primeramente necesita herramientas de hierro, que a su vez establece su propia jerarquía de tareas necesarias como recolectar hierro, palos, etc.

  Esta naturaleza jerárquica implica que el agente debe aprender no sólo las acciones individuales, sino también cómo coordinar estas acciones en secuencias lógicas y estructuradas para lograr objetivos más complejos.

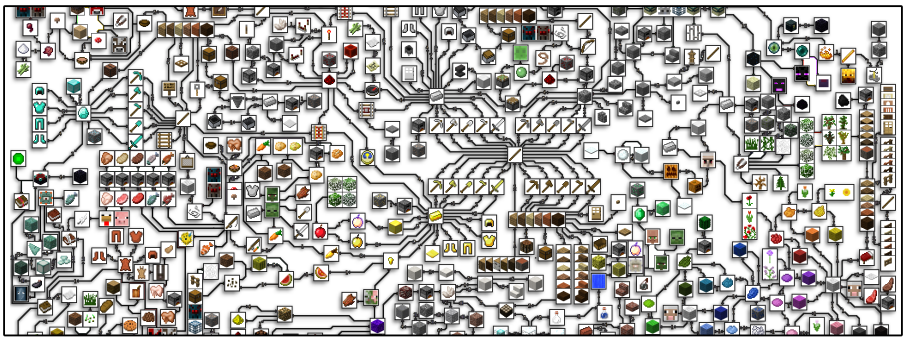

Figura 2.5: Jerarquía de algunos de los objetos de Minecraft. Fuente: [42]

- **Flexibilidad:** Minecraft puede ser modificado para adaptarse a las necesidades específicas de un experimento. Esta alterabilidad es esencial para la creación de entornos de entrenamiento personalizados.

2.3.2. Relevancia para RL y IL

El entorno virtual y diverso de Minecraft lo convierte en un escenario ideal para el aprendizaje por refuerzo y el aprendizaje por imitación:

[5] El término *craftear* proviene del inglés *craft*, que significa *fabricar* o *hacer*. En el contexto de videojuegos como Minecraft, se refiere al proceso de combinar materiales o ingredientes en un orden específico para crear un nuevo objeto o herramienta.

- **Entornos simulados:** La naturaleza virtual de Minecraft permite simular una amplia gama de situaciones y tareas, lo que es esencial para el entrenamiento y evaluación de algoritmos de RL e IL.
- **Seguridad y escalabilidad:** En comparación con el entrenamiento en el mundo real, el entrenamiento en Minecraft elimina los riesgos físicos y permite escalar experimentos sin incurrir en costos adicionales significativos.
- **Recolección de datos:** Para el Aprendizaje por Imitación, los expertos humanos pueden jugar dentro de Minecraft, y sus acciones pueden ser grabadas y utilizadas como datos de entrenamiento para agentes inteligentes.
- **Experimentación:** Los investigadores pueden introducir desafíos específicos o modificar las reglas del juego para estudiar cómo los agentes se adaptan y aprenden en circunstancias cambiantes.

2.3.3. Conclusión

Aunque Minecraft comenzó como un simple juego de construcción, su versatilidad y complejidad lo han transformado en una herramienta con un potencial extraordinario para la comunidad de investigación en aprendizaje automático. Su capacidad para modelar y simular una variedad de tareas y entornos lo posiciona como un recurso clave para el avance y evaluación de algoritmos de RL e IL.

CAPÍTULO 3
Estado del arte

En este capítulo se abordarán temas relacionados con la aplicación de técnicas de Aprendizaje por Refuerzo (RL) e Imitación (IL) en entornos de Minecraft. Para ello, es fundamental considerar el estado actual de la investigación en el dominio de los videojuegos y, específicamente, en Minecraft, lo que permitirá comprender los enfoques más prometedores para la enseñanza de los agentes en este entorno. Para ello, se tendrán en cuenta aspectos como la interacción de los agentes con el entorno, la gestión de las recompensas, así como las herramientas de modelado y simulación. De esta manera, se buscará brindar una visión amplia y detallada sobre cómo se puede lograr un aprendizaje eficiente y exitoso de agentes en entornos de Minecraft mediante modelos de RL e IL.

3.1 Aprendizaje por refuerzo en videojuegos

El aprendizaje por refuerzo (RL) se ha consolidado como uno de los enfoques dominantes en la inteligencia artificial, siendo los videojuegos un medio ideal para probar y desarrollar algoritmos de RL. En los últimos años, hemos visto avances notables en esta área:

- **DeepMind y Atari (2013-2015):** DeepMind, con su algoritmo DQN, demostró que era posible entrenar agentes de RL directamente desde imágenes en bruto para superar el rendimiento humano en una variedad de juegos de Atari 2600 [43].
- **AlphaZero (2018):** AlphaZero, desarrollado por DeepMind, no solo dominó el juego de Go, superando a su predecesor AlphaGo, sino que también aprendió y dominó otros juegos como el ajedrez y el shogi únicamente a través del autoaprendizaje, sin acceso a conocimientos humanos previos más allá de las reglas básicas [44].
- **OpenAI y Dota 2 (2018-2019):** OpenAI presentó a OpenAI Five, un agente que logró competir a nivel profesional en el juego Dota 2, un juego mucho más complejo en términos de estrategia y dinámica que Atari [45].
- **DeepMind y AlphaStar (2019):** DeepMind desarrolló AlphaStar, que logró dominar el complejo videojuego de estrategia en tiempo real StarCraft II, llegando a vencer a jugadores profesionales humanos [46].

A pesar de estos avances, hay varios desafíos a los que se enfrenta el RL en el contexto de los videojuegos:

- **Complejidad creciente:** A medida que la industria de los videojuegos avanza, los juegos se vuelven más complejos en términos de gráficos, mecánicas y estrategias. Esto implica que el espacio de acciones posibles y el espacio de estados crece exponencialmente. En términos prácticos, esto significa que hay una cantidad astronómica de posibles decisiones y escenarios que un agente de RL debe considerar y aprender. Tal crecimiento puede hacer que el entrenamiento sea más lento y que se necesite más memoria para almacenar y procesar la información, lo que limita la eficiencia y eficacia de los algoritmos tradicionales de RL [22].

- **Generalización:** Como introdujimos brevemente en el apartado 2.1.5, mientras que es posible entrenar un agente para que domine un juego específico, no hay garantía de que el mismo agente, o incluso el mismo algoritmo, se desempeñe bien en otro juego. Cada juego tiene sus propias reglas y dinámicas. La generalización, o la habilidad de transferir lo aprendido de un contexto a otro, sigue siendo una cuestión abierta en RL. Aunque existen esfuerzos en dirección a los algoritmos de *aprendizaje de transferencia*, el desafío radica en cómo adaptar y aprovechar efectivamente las habilidades aprendidas en un juego para un nuevo escenario o juego [24].

- **Entrenamiento en tiempo real:** Los videojuegos en tiempo real, a diferencia de los juegos por turnos, requieren que los agentes tomen decisiones rápidas y constantes, a menudo en fracciones de segundo. No sólo el agente debe decidir la mejor acción a seguir, sino que también debe hacerlo muy rápidamente para mantenerse al día con el ritmo del juego. Esto puede requerir optimizaciones específicas en el código y hardware especializado para garantizar que los tiempos de cálculo no interfieran con el rendimiento del juego o del agente [43].

3.1.1. Aprendizaje por refuerzo en Minecraft

Debido a la naturaleza de Minecraft (2.3.1), problemas como la exploración en mundos vastos, el aprendizaje jerárquico, y la generalización entre diferentes mundos se amplifican. La diversidad y escala de los mundos generados aleatoriamente en Minecraft pueden complicar la convergencia de los algoritmos tradicionales de RL. A continuación, se detallan las principales desafíos que enfrenta el RL en entornos de Minecraft:

1. **Espacio de estados enorme:** Minecraft es un juego de mundo abierto con un gran número de posibles estados debido a sus vastos entornos generados de forma aleatoria. Un agente de RL necesita explorar y comprender este vasto espacio, lo que es computacionalmente costoso y consume mucho tiempo.

2. **Recompensas dispersas y retrasadas:** A menudo, las acciones que un agente toma en Minecraft no resultan inmediatamente en una recompensa. Por ejemplo, cavar en busca de un recurso escaso puede requerir mucho tiempo y esfuerzo antes de obtener una recompensa. Esta dispersión y retraso en las recompensas hace que sea difícil para los algoritmos de RL correlacionar acciones con sus consecuencias.

3. **Tareas jerárquicas:** Como comentábamos en el apartado 2.3.1, algunas acciones en Minecraft, como construir una herramienta o una estructura, requieren una serie de pasos intermedios. Esta estructura jerárquica puede ser desafiante para los algoritmos de RL tradicionales que suelen funcionar mejor en tareas más atómicas. Se ha tratado de abordar este tipo de problemas haciendo uso de Hierarchical Deep Reinforcement Learning Network (H-DRLN) [47].

4. **Dinámica del entorno:** El mundo de Minecraft es dinámico, con ciclo día-noche, clima variable, y criaturas que se mueven y reaccionan de manera autónoma. Esto introduce una dimensión adicional de variabilidad que el RL debe manejar.

Por todos estos desafíos, se ha visto una tendencia en la combinación de RL con otras técnicas, como el Imitation Learning, para aprovechar la experiencia humana y guiar a los agentes durante el proceso de aprendizaje en los complejos entornos de Minecraft.

Plataformas de investigación

Dadas estas dificultades, la comunidad científica ha desarrollado plataformas específicas para investigar y abordar estos retos:

- **Project Malmo (2016):** Microsoft introdujo Project Malmo, una plataforma que permite experimentar con RL y otras técnicas de inteligencia artificial dentro de Minecraft [48].
- **MineRL (2019):** Un proyecto centrado en destacar la ineficiencia de muestra de los métodos estándar de Deep Reinforcement Learning. Promueve el uso de demostraciones humanas para un aprendizaje más eficiente, proporcionando un conjunto de datos a gran escala con demostraciones de expertos, así como entornos personalizados de Minecraft [1].

Conclusión

El uso del Aprendizaje por Refuerzo (RL) en Minecraft se enfrenta a desafíos que se magnifican debido a las particularidades inherentes de este juego. Estos desafíos han llevado a la comunidad investigadora a concluir que los métodos tradicionales de RL, por sí solos, pueden no ser suficientemente efectivos en el ámbito de Minecraft. La colaboración entre RL y técnicas como el Imitation Learning (IL) emerge como una solución prometedora, aprovechando las experiencias y demostraciones humanas para guiar y mejorar el proceso de aprendizaje de los agentes.

En vista de la aparente insuficiencia de RL por sí solo y la necesidad de combinarlo con IL, el siguiente apartado se centrará en estudiar el estado del arte del Imitation Learning y explorar sus aplicaciones específicas en el contexto de Minecraft, buscando entender cómo esta combinación puede superar los retos mencionados y mejorar la eficiencia del aprendizaje en estos entornos.

3.2 Aprendizaje por imitación en videojuegos

El aprendizaje por imitación (IL) ha surgido como una estrategia efectiva en el campo de la inteligencia artificial para resolver problemas en los que la función de recompensa es difícil de especificar o donde la exploración tradicional resulta ineficiente. Los videojuegos proporcionan un entorno óptimo para la implementación y prueba de técnicas de IL. Prueba de ello son los diversos avances en la aplicación de IL en el contexto de los videojuegos:

- **Atari y Dagger (2015):** A través del algoritmo Dagger, se demostró que es posible aprender políticas efectivas en juegos de Atari utilizando IL, donde los agentes imitaban el comportamiento del experto en lugar de aprender a través de la interacción directa [49].

- **StarCraft II e Imitation Learning (2018):** Antes del dominio de AlphaStar usando RL, se exploraron enfoques basados en IL, en los que los agentes aprendían inicialmente imitando las estrategias de jugadores humanos para luego perfeccionar sus habilidades mediante la auto-competencia [50].
- **Forza Motorsport (2018):** En el popular juego de carreras, se ha utilizado IL para enseñar a los agentes a conducir imitando los estilos de conducción de jugadores humanos, logrando tiempos de vuelta competitivos y estilos de conducción realistas [51].

Aunque IL ha demostrado ser eficaz en muchos contextos, también presenta desafíos propios en el ámbito de los videojuegos:

- **Error de acumulación:** Al imitar secuencialmente las acciones de un experto, los errores cometidos por el agente pueden acumularse, resultando en un comportamiento muy alejado del experto original.
- **Ausencia de exploración:** A diferencia del RL, donde los agentes exploran activamente el espacio de acciones, en IL, los agentes se limitan a las demostraciones proporcionadas. Esto puede ser limitante si las demostraciones no cubren todas las situaciones posibles en el juego.
- **Costo de demostraciones de expertos:** Obtener demostraciones de expertos puede ser costoso y limitado, especialmente en videojuegos de alto nivel, donde las estrategias óptimas son complejas y cambian con el tiempo.

3.2.1. Aprendizaje por imitación en Minecraft

Minecraft es un videojuego de construcción y aventura con un mundo compuesto por bloques, en el que los jugadores pueden crear estructuras, minar recursos y enfrentarse a criaturas. Gracias a la complejidad y diversidad de tareas que se pueden realizar, este juego ofrece un entorno ideal para la aplicación de técnicas avanzadas de inteligencia artificial. El IL, en particular, puede abordar varios desafíos dentro de este entorno:

1. **Navegación y construcción:** Uno de los principales desafíos en Minecraft es aprender a moverse de manera eficiente a través de terrenos accidentados, laberintos subterráneos, cuevas y estructuras creadas por otros jugadores. Mediante IL, los agentes pueden aprender a navegar y construir observando y replicando las acciones de jugadores expertos. Esta imitación puede ser particularmente útil para enseñar a los agentes patrones de construcción avanzados o estrategias de navegación en terrenos complicados.
2. **Interacción con entidades:** El mundo de Minecraft está poblado por una variedad de entidades, desde pacíficos animales de granja hasta hostiles monstruos nocturnos. Cada entidad tiene su propio conjunto de comportamientos y reacciones frente a las acciones del jugador. A través del IL, un agente puede aprender cómo y cuándo interactuar con estas entidades, ya sea para domesticar animales, comerciar con aldeanos o defenderse de amenazas. Imitando las estrategias de los jugadores, el agente puede aprender, por ejemplo, cuándo es el momento adecuado para atacar, huir o negociar.
3. **Adaptabilidad:** Una característica única de Minecraft es que cada nuevo juego comienza en un mundo generado aleatoriamente. Esto significa que, aunque existen

ciertas constantes (como la aparición de ciertos tipos de bloques o criaturas), la disposición específica del terreno y la ubicación de los recursos varían ampliamente. El IL puede ser esencial aquí, permitiendo a los agentes adaptarse rápidamente a estos nuevos entornos al aprender de demostraciones realizadas en múltiples mundos. Esto los capacita para enfrentar de manera eficiente situaciones nunca antes vistas.

4. **Toma de decisiones bajo incertidumbre:** Minecraft, con su ciclo día-noche y sus cambiantes condiciones, a menudo presenta situaciones donde los jugadores deben tomar decisiones con información limitada. Por ejemplo, un jugador podría encontrarse atrapado en una cueva sin antorchas mientras se aproxima la noche. Mediante IL, los agentes pueden aprender a tomar decisiones estratégicas en tales escenarios, imitando cómo los jugadores expertos anticipan y reaccionan a las amenazas o cómo priorizan recursos bajo presión.

El uso del IL en Minecraft puede abrir puertas a nuevos métodos y estrategias para entrenar agentes inteligentes en entornos complejos y dinámicos, y sirve como un excelente ejemplo de cómo la imitación puede solventar problemas que otros métodos como el RL son incapaces de hacer.

Estudios relevantes

Uno de los principales contribuyentes a esta área de estudio, y sobre el que se sustentan la gran parte de las investigaciones de esta técnica, es MineRL [1]. Ya introducido con anterioridad en el apartado 3.1.1, este proyecto proporciona un extenso conjunto de datos, recogido de jugadores humanos, que abarca más de 60 millones de estados de Minecraft etiquetados con acciones y recompensas. Esta gran base de datos permite a los agentes aprender a partir de interacciones reales, lo cual resulta particularmente útil en un entorno como Minecraft, donde la escasez de recompensas y la complejidad para especificar funciones de recompensa adecuadas presentan desafíos significativos.

Sin embargo, la contribución de MineRL al IL no se limita a la provisión de un conjunto de datos voluminoso. Además, ha proporcionado un entorno de simulación interactivo de alto rendimiento que facilita a los investigadores el desarrollo y prueba de algoritmos de IL en un entorno complejo y realista, proporcionando entornos compatibles sobre los que ejecutar los algoritmos, así como varías competiciones:

- **2019**: *MineRL Competition on Sample Efficient Reinforcement Learning Using Human Priors* [52].
- **2020**: *The MineRL 2020 Competition on Sample Efficient Reinforcement Learning Using Human Priors* [53, 54].
- **2022**: *Benchmark for Agents that Solve Almost-Lifelike Task (BASALT) Competition* [55].

Hay muchos trabajos relevantes al respecto (los cuales se pueden consultar en `https://minerl.readthedocs.io/en/latest/notes/useful-links.html#minerl-papers`), pero sin duda, el más impactante y con mejores resultados de todos, que incluso ha servido de base para la competición BASALT, es el reciente artículo de OpenAI: *Learning to play Minecraft with Video PreTraining* [56].

En este artículo, los investigadores proponen un enfoque innovador para el aprendizaje de agentes en dominios de decisiones secuenciales. En lugar de depender de grandes conjuntos de datos etiquetados, como es el proporcionado por MineRL, utilizan un

enfoque de aprendizaje por imitación semi-supervisado, donde los agentes se entrenan observando vídeos en línea no etiquetados de personas jugando Minecraft (ver Figura 3.1). Este *prior de comportamiento* entrenado demuestra capacidades impresionantes sin entrenamiento adicional y puede ser refinado para tareas complejas. Es una contribución notable en el campo, mostrando que los agentes pueden alcanzar niveles de rendimiento comparables a los humanos en tareas complejas dentro de Minecraft.

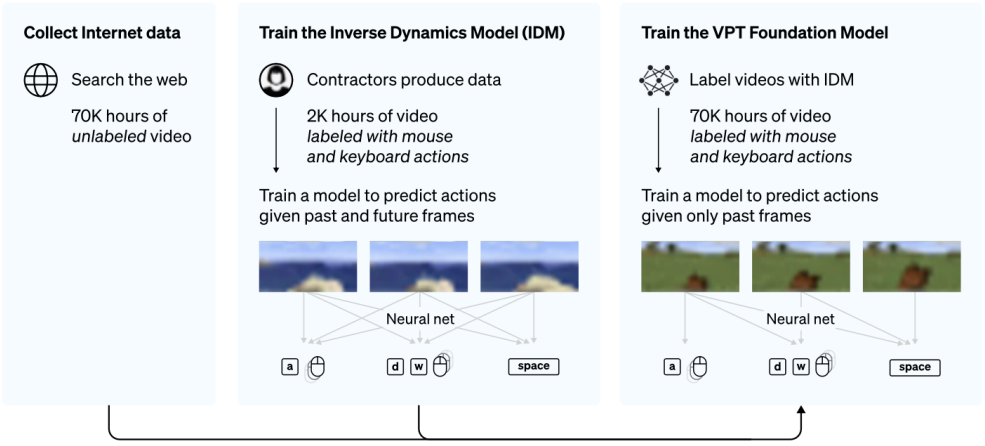

Figura 3.1: Visión general del método VPT. Fuente: [57]

Dados los increíbles resultados (ver Figura 3.2), los investigadores de OpenAI afirman que es la primera vez que alguien muestra un agente informático capaz de fabricar herramientas de diamante en Minecraft, lo que a los humanos les lleva más de 20 minutos (24.000 acciones) de media [57].

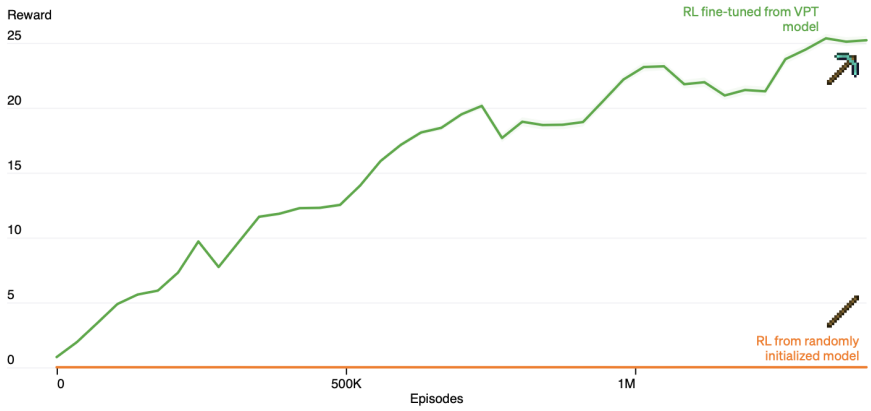

Figura 3.2: Comparativa entre un método de RL estándar y el algoritmo de RL mejorado con el modelo VPT. Fuente: [57]

3.3 Paradigmas emergentes en Minecraft

El pasado año 2022 presenciamos una revolución en los modelos del lenguaje, más concretamente con la presentación del Large Languaje Model (LLM) GPT-3 por parte del equipo de OpenAI. A esto se le ha sumado la incorporación, este mismo año, de otros dos LLM: GPT-3.5 y GPT-4.

Gracias a estos avances se han desarrollado proyectos innovadores que buscan potenciar y diversificar la aplicación de estos modelos en entornos virtuales y simulados, como es el caso de *MineDojo*.

3.3.1. MineDojo

MineDojo es un framework[1] basado en el popular juego Minecraft para la investigación con *embodied agents*[2] [58]. Este nuevo framework aporta tres herramientas clave:

- Un entorno abierto que permite una multitud de tareas y metas diferentes.
- Una base de datos a gran escala de conocimientos multimodales, como son vídeos, tutoriales, páginas wiki y foros de debate sobre Minecraft.
- La arquitectura MineCLIP, con la que, dado una orden de texto, el modelo ejecuta la acción dentro de Minecraft y analiza el vídeo (*clip*) generado por la acción para determinar si el agente está tomando la acción correcta.

Todo esto abierto de manera pública [59], con el fin de fomentar la investigación acerca de la creación de *embodied agents* con capacidad general. Dos de las investigaciones más recientes son Voyager y Ghost, ambos casualmente presentados el mismo día.

Voyager

Haciendo uso de MineDojo, *Voyager* se presenta como un revolucionario agente de aprendizaje continuo en Minecraft [62], impulsado por el avanzado Large Language Model (LLM), GPT-4. Este agente no solo explora el mundo de Minecraft de manera autónoma, sino que también adquiere habilidades y realiza descubrimientos sin necesidad de intervención humana. El diseño de Voyager combina tres componentes: un *automatic curriculum* [60] que prioriza la exploración, una biblioteca de habilidades para almacenar y recuperar comportamientos complejos, y un mecanismo iterativo que genera código ejecutable para el control del *embodied agent* (ver Figura 3.3).

La verdadera innovación radica en su mecanismo iterativo que, basado en retroalimentación en tiempo real, perfecciona sus acciones en el juego. En un entorno tan dinámico como Minecraft, donde los objetivos no están predefinidos, la habilidad de Voyager para adaptarse y aprender de su entorno le permite enfrentar desafíos de largo alcance, mostrando una capacidad de acción única y avanzada.

[1] En informática, un *framework* es un conjunto estandarizado de herramientas y bibliotecas que ofrece una estructura base para desarrollar y desplegar aplicaciones de manera más eficiente.

[2] Así es como se denomina a los agentes virtuales capaces de percibir y actuar con su entorno.

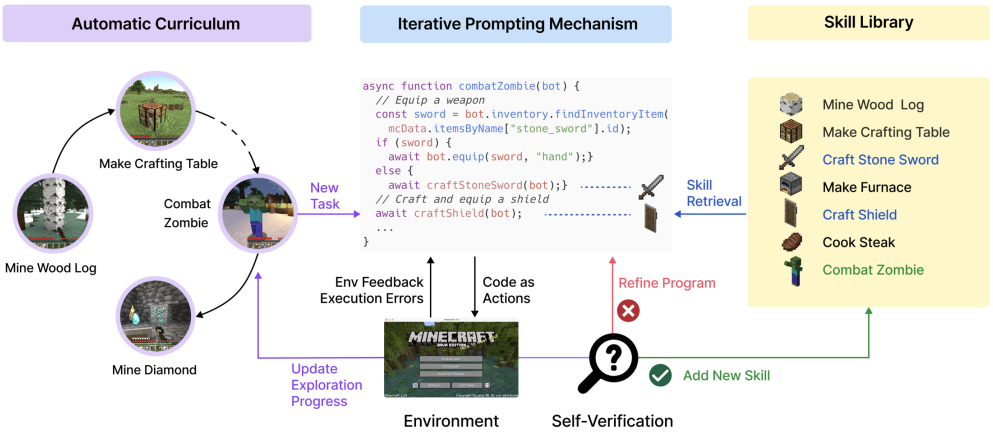

Figura 3.3: Estructura de Voyager. Fuente: [61]

Es relevante señalar la superioridad de Voyager en comparación con otras soluciones. Voyager destaca al descubrir 63 objetos únicos dentro de 160 iteraciones de instrucciones, lo que supone 3.3 veces más ítems que sus competidores y desbloqueando niveles del árbol tecnológico de Minecraft hasta 15.3 veces más rápido que otras soluciones, y siendo el único en alcanzar el codiciado nivel de diamante. Hasta que ese mismo día se presentó el modelo Ghost.

Ghost In The Minecraft (GITM)

Aprovechando la integración de modelos de lenguaje en Minecraft, *Ghost In The Minecraft (GITM)* surge como un novedoso modelo que integra Large Language Models (LLMs) con conocimiento y memoria basados en texto, con el objetivo de crear Agentes Generalmente Capaces (GCAs) en el mundo de Minecraft [63]. A diferencia de otros agentes que dependen principalmente del aprendizaje por refuerzo para sus operaciones, GITM propone una nueva solución que hace uso de agentes basados en LLM.

El diseño de GITM comprende diversos componentes (ver Figura 3.4), como el *LLM Decomposer*, que descompone objetivos complejos en sub-objetivos más alcanzables, y el *LLM Planner*, que integra acciones estructuradas y mecanismos de retroalimentación para permitir una mejor interacción en el entorno de Minecraft. Estos componentes trabajan juntos, permitiendo que GITM desbloquee el árbol tecnológico completo de Minecraft, demostrando una capacidad de recolección formidable.

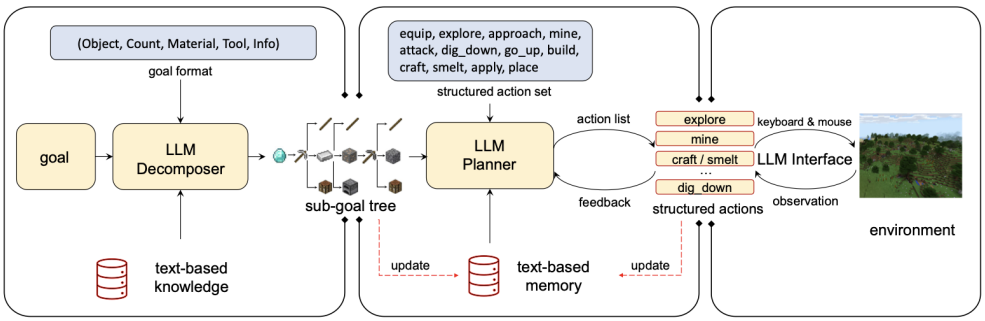

Figura 3.4: Estructura de Ghost In The Minecraft (GITM). Fuente: [63]

Es fundamental destacar la eficacia de GITM en comparación con otros agentes. Dada la figura 3.5, GITM no solo ha demostrado ser capaz de recolectar todos los 262 ítems en

Minecraft, sino que también logra tasas de éxito del 100 % en tareas sencillas, superando a otros agentes en su capacidad de recolección y adaptabilidad. Esta proeza demuestra la potencia y versatilidad del uso de modelos de lenguaje grande en entornos complejos y desafiantes como Minecraft.

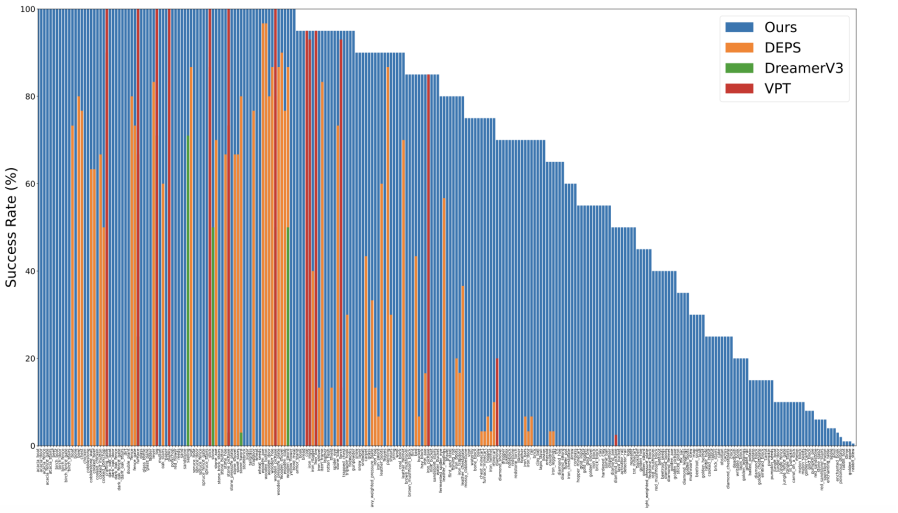

Figura 3.5: Porcentaje de éxito de GITM en la recogida de objetos en comparación con otros agentes. Fuente: [63]

3.4 Conclusión final

En conclusión, el avance de las técnicas de Aprendizaje por Refuerzo (RL) y Aprendizaje por Imitación (IL) en entornos de videojuegos como Minecraft se presenta como un campo prometedor y lleno de oportunidades. A pesar de los retos existentes, tales como la mejora de la eficiencia de aprendizaje, la robustez y la generalización de los algoritmos en nuevos entornos y tareas, se ha logrado un progreso significativo que sienta las bases para futuros avances. A estos avances se le suman las recientes investigaciones haciendo uso de Large Languaje Models (LLM), que están tomando la delantera de una forma vertiginosa en cuanto a capacidades generales se refiere.

Mientras los esfuerzos de investigación se mantengan y se expandan, se espera presenciar avances notables que no solo tendrán un impacto en el campo de la inteligencia artificial en los videojuegos, sino que también encontrarán aplicaciones más amplias en otros dominios. Estas áreas pueden beneficiarse en gran medida de las lecciones aprendidas en estos entornos de videojuegos, que son reconocidos por su complejidad y dinamismo, ofreciendo escenarios ideales para el desarrollo y prueba de algoritmos de aprendizaje autónomo.

CAPÍTULO 4
Pruebas

4.1 Herramientas y plataformas utilizadas

A lo largo de este proyecto, se han utilizado varias herramientas y plataformas para facilitar el desarrollo, ejecución y monitorización del trabajo. A continuación, se detallan estas herramientas y plataformas:

4.1.1. MineRL

MineRL es una librería diseñada específicamente para llevar a cabo investigaciones sobre Reinforcement Learning (RL) e Imitation Learning (IL) dentro del entorno de Minecraft. Esta librería ha sido fundamental para el desarrollo de este trabajo, ya que proporciona una interfaz de alto nivel al estilo OpenAI gym[1] para interactuar con el juego, facilitando la implementación de algoritmos y la recopilación de datos.

También nos ha dado muchos quebraderos de cabeza, que hemos tratado de ir solventando hablando con los desarrolladores de la librería a través de GitHub, en un proceso de retroalimentación para ambos, donde ellos han podido solventar errores que hemos ido encontrando al realizar este trabajo [64].

4.1.2. Stable Baselines3

Stable Baselines3 (SB3) es un conjunto de implementaciones de algoritmos de aprendizaje por refuerzo, escrito en PyTorch [65]. En este trabajo, Stable Baselines3 ha sido utilizado para implementar los algoritmos de RL que se han probado en el entorno de Minecraft proporcionado por MineRL. La gran popularidad de está librería, y lo que decantó nuestra elección por la misma, es debido a la simpleza y facilidad de implementar dichos algoritmos dentro de el código.

```
from stable_baselines3 import PPO

# Cargar el entorno de MineRL
env = gym.make("MineRLTreechop-v0")

# Inicializacion del modelo PPO
model = PPO("CnnPolicy", env, verbose=1)

# Entrenamiento del modelo
model.learn(total_timesteps=5000000)
```

[1]Gym es una librería de Python que proporciona una API estándar para el aprendizaje por refuerzo.

Listing 4.1: Código de ejemplo utilizando Stable Baselines3.

Pese a la gran abstracción que supone SB3 de cara a la implementación del algoritmo, las posibilidades de configuración son excelentes, y están reflejadas en la amplia documentación de la que dispone cada algoritmo. En su configuración puedes especificar parámetros generales como el tipo de política, el ratio de aprendizaje o el batch size, así como parámetros específicos del algoritmo en cuestión, como pueden ser el gae lambda, clip range, entropy coefficient, argumentos para el reply buffer, etc. Además de toda esta configuración, puedes añadir al algoritmo Callbacks personalizados, esto son, funciones que son llamadas cuando ocurre un evento especifico, ya sea una cantidad fija de timesteps, el fin de un rollout[2] o para cada step del algoritmo, entre muchas otras.

Estas características hacen de Stable Baselines3 una librería perfecta para nuestro trabajo, ya que nos permite tener implementaciones de los algoritmos más populares teniendo una seguridad de fiabilidad y capacidades de modificación excepcionales.

4.1.3. Imitation

La librería Imitation es una extensión natural para aquellos que ya están familiarizados con Stable Baselines3, ya que proporciona herramientas y algoritmos para el aprendizaje por imitación en Python [67]. En este trabajo, se ha utilizado la librería Imitation para implementar y evaluar varios algoritmos de IL en el entorno de Minecraft proporcionado por MineRL.

Uno de los principales atractivos de Imitation es su enfoque modular y su estrecha integración con SB3, lo que permite una integración sin problemas entre algoritmos de RL y IL. Además, ofrece una amplia gama de algoritmos de aprendizaje por imitación, desde los más simples, como Behavioral Cloning, hasta enfoques más avanzados que combinan IL con RL.

Al igual que SB3, Imitation también ofrece una amplia flexibilidad en términos de configuración. Permite ajustar numerosos hiperparámetros, dependiendo del algoritmo específico de IL que se esté utilizando. También dispone de la capacidad de utilizar callbacks, como en SB3, asegurando que los investigadores puedan tener un control detallado del proceso de entrenamiento, permitiendo intervenciones y monitoreo en tiempo real.

El diseño intuitivo, la variedad de algoritmos ofrecidos y la compatibilidad con SB3 hacen de Imitation una elección ideal para este proyecto, permitiendo experimentar y comparar diferentes enfoques de aprendizaje por imitación de manera eficiente y efectiva.

4.1.4. Wandb

Wandb es una herramienta para la monitorización de ejecuciones. Ha sido empleada en este proyecto para seguir el progreso de las ejecuciones, identificar posibles problemas y optimizar el proceso de entrenamiento. Para ello, siguiendo lo comentado en el apartado anterior, diseñamos un Callback específico para la monitorización de la ejecución:

```
1    class WandbCallback(BaseCallback):
2        def __init__(self, verbose=0):
```

[2] Un *rollout* se refiere a la simulación de una política en el entorno durante un número específico de pasos o hasta que finalice un episodio. Es esencialmente una muestra sobre cómo se desempeñaría el agente bajo esa política específica.

```
            super(WandbCallback, self).__init__(verbose)
            self.last_logged_episode = -1

        def _on_rollout_end(self):
            # Obtenemos el entorno de entrenamiento
            env = self.training_env.envs[0].env

            # Recuperamos las recompensas y la longitud de los episodios
            rewards = env.get_episode_rewards()
            lengths = env.get_episode_lengths()

            # Comprobacion para evitar solapamiento de datos
            if len(rewards) > self.last_logged_episode + 1:
                mean_reward = sum(rewards[self.last_logged_episode+1:]) / len(
                    rewards[self.last_logged_episode+1:])

                mean_length = sum(lengths[self.last_logged_episode+1:]) / len(
                    lengths[self.last_logged_episode+1:])

                total_timesteps = env.get_total_steps()

                # Loggear los resultados en Wandb
                wandb.log({
                    'mean_reward': mean_reward,
                    'mean_episode_length': mean_length,
                    'total_timesteps': total_timesteps
                })

                self.last_logged_episode = len(rewards) - 1

        def _on_step(self):
            return True
```

4.1.5. VRAIN HPC Cluster y Singularity

Para las ejecuciones de este trabajo hicimos uso del cluster de alto rendimiento (HPC) de VRAIN, que proporciona una plataforma de cálculo potente y escalable, esencial para ejecutar algoritmos intensivos en recursos como los utilizados en este proyecto. Sin embargo, dado que el HPC opera en un entorno sin interfaz gráfica (headless), ha sido necesario abordar desafíos adicionales para que MineRL funcionara correctamente.

Para superar las limitaciones del entorno headless del cluster HPC, creamos un contenedor Singularity personalizado. Singularity permite la creación y ejecución de contenedores ligeros que pueden albergar aplicaciones y sus dependencias, garantizando la reproducibilidad y el aislamiento del entorno.

El contenedor ha sido diseñado para permitir la ejecución de MineRL en el cluster HPC y habilitar el renderizado utilizando GPU, para ello se preparó el contenedor para poder utilizar VirtualGL, CUDA/cuDNN y las librerías mencionadas en los apartados anteriores, instaladas en un entorno virtual de Anaconda. Los archivos necesarios para crear el contenedor y poder ejecutar los programas de MineRL los hemos liberado de manera abierta para que cualquier persona que se encontrara en la misma situación que nosotros pudiera utilizarlo. Se pueden encontrar en el repositorio de GitHub singularity-minerl.

Finalmente, nuestro repositorio ha sido añadido a la documentación de MineRL dentro del apartado *'Performance tips'* por haber contribuido con la creación del contenedor Singularity.

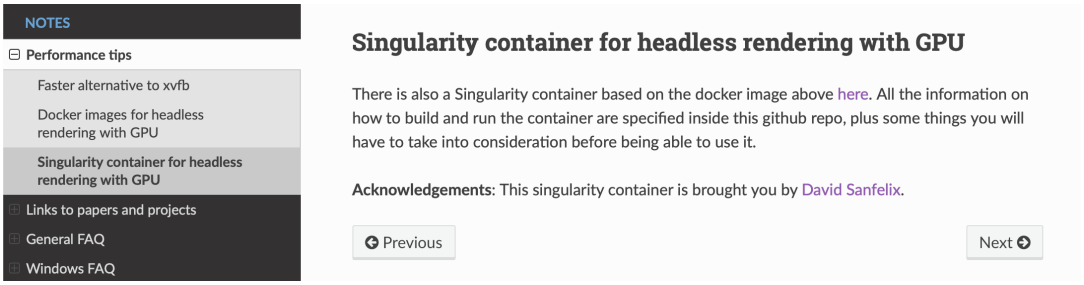

Figura 4.1: Mención y reconocimiento en la documentación de MineRL. Fuente: [66]

4.2 Experimentos

De toda la suite de entornos proporcionados por MineRL nos decantamos por utilizar *MineRLTreechop-v0*, un entorno donde el objetivo del agente es recolectar madera, un recurso primordial en Minecraft. Nos pareció un entorno sencillo en cuanto a las tareas a realizar, pero que seguía presentando las dificultades propias de Minecraft y su entorno dinámico.

- **Espacio de observaciones**: Las observaciones del entorno *MineRLTreechop-v0* están compuestas una imagen RGB con dimensiones: 64 x 64. La observación devuelta por el entorno es un diccionario con una única clave *pov*[3], cuyo valor incluye la matriz con los valores de la imagen.
- **Espacio de acciones**: El espacio de acciones de este entorno está compuesto por un diccionario con todas las acciones de movimiento posibles como son caminar hacia adelante, hacia atrás, a la izquierda, a la derecha, saltar y agacharse, el movimiento de la cámara y otras acciones para interactuar con el inventario. Cada una de ellas con un valor discreto entre 0 y 1, que representa si se realiza o no dicha acción.

4.2.1. Incompatibilidad del espacio de acciones con Stable Baselines3

Una vez dispuestos a iniciar las ejecuciones, nos encontramos con un primer problema, ya que los algoritmos de RL implementados en Stable Baselines3 únicamente permiten espacios de acciones discretos, por lo que no admiten diccionarios como el devuelto por nuestro entorno. Es por ello que tuvimos que crear una solución que permitiera la comunicación de manera correcta entre el algoritmo y el entorno.

Solución

Para solucionar el problema, creamos una envoltura (*wrapper*) llamado `BitMaskWrapper`, que modifica el espacio de acciones del entorno para que sea discreto, y mapea las acciones discretas de vuelta al esquema original de diccionario cuando se envían al entorno. Además, decidimos reducir el espacio de acciones a un total de 9 acciones posibles, ya que varias de las acciones originales no son necesarias para superar el entorno y de esta manera podemos reducir el tamaño del espacio de acciones total.

Cómo funciona el `BitMaskWrapper`

[3] *POV* es el acrónimo de "Point of View"(Punto de Vista). En videojuegos, el POV se refiere a la perspectiva visual desde la cual un jugador experimenta el juego. Minecraft ofrece múltiples POVs, incluyendo vistas en primera y tercera persona.

1. **Inicialización**: Durante la inicialización, el `BitMaskWrapper` guarda el espacio de acción original y crea un nuevo espacio de acción discreto con 2048 valores posibles (2^{11}). También se precalcula una acción "no-op" (sin operación) para utilizarla posteriormente como base en el mapeo de acciones.

2. **Función step**: Cuando se recibe una acción discreta, esta se pasa por el método `_apply_bit_mask` para convertirla en una acción con el formato de diccionario. Después de convertir la acción, se envía al entorno original y se obtiene la respuesta del entorno.

3. **Conversión de la acción (_apply_bit_mask)**: Cada acción discreta se puede representar como un número de 11 bits, 8 bits para las acciones individuales y 3 bits destinados a representar las direcciones de la cámara. De las 8 acciones de cámara posibles, únicamente se han representado 5, por lo que hay espacio para una posible ampliación.

   Se descompone el valor *action* en bits y se comprueba que acciones están o no activadas, y se añade su valor al diccionario de acciones final que será enviado al entorno.

4. **Función reset**: Se reinicia el entorno y procesa la observación para que esté en el formato esperado, extrayendo el POV de la observación.

```
class BitMaskWrapper(gym.Wrapper):
    def __init__(self, env):
        super(BitMaskWrapper, self).__init__(env)
        self.orig_action_space = self.action_space
        # Modificamos el espacio de acciones
        self.action_space = gym.spaces.Discrete(2**11)
        # Modificamos el espacio de observaciones
        self.observation_space = self.observation_space['pov']
        # Guardamos la plantilla de accion vacia
        self.noop_action = self.orig_action_space.noop()


    def step(self, action):
        assert 0 <= action < 2**11, "Invalid action"
        masked_action = self._apply_bit_mask(action)
        obs, reward, done, info = self.env.step(masked_action)
        obs = obs["pov"]
        obs = obs / 255.0
        return obs, reward, done, info

    def reset(self, **kwargs):
        obs = self.env.reset(**kwargs)
        obs = obs["pov"]
        obs = obs / 255.0
        return obs

    def _apply_bit_mask(self, action):
        # Crear un diccionario con la accion vacia
        dic_action = self.noop_action.copy()

        dic_action["attack"] = action & 1
        dic_action["back"] = (action >> 1) & 1
        dic_action["forward"] = (action >> 2) & 1
        dic_action["left"] = (action >> 3) & 1
        dic_action["right"] = (action >> 4) & 1
        dic_action["jump"] = (action >> 5) & 1
        dic_action["sneak"] = (action >> 6) & 1
        dic_action["sprint"] = (action >> 7) & 1
```

```
39
40              # Para la accion de camara, consideraremos 4 direcciones
                    principales
41              camera_dir = (action >> 8) & 7  # 3 bits
42              if camera_dir == 0:
43                  dic_action["camera"] = [0, -20]  # Izquierda
44              elif camera_dir == 1:
45                  dic_action["camera"] = [0, 20]  # Derecha
46              elif camera_dir == 2:
47                  dic_action["camera"] = [20, 0]  # Abajo
48              elif camera_dir == 3:
49                  dic_action["camera"] = [-20, 0]  # Arriba
50              elif camera_dir == 4:
51                  dic_action["camera"] = [0, 0]  # Sin movimiento
52
53              return dic_action
54
55          def get_action_meanings(self):
56              return [str(i) for i in range(self.action_space.n)]
57
58          def render(self, mode='human', **kwargs):
59              return self.env.render(mode, **kwargs)
60
61          def seed(self, seed=None):
62              return self.env.seed(seed)
```

Listing 4.2: Implementación del BitMaskWrapper.

El `BitMaskWrapper` proporciona una solución elegante y efectiva para adaptar entornos complejos a bibliotecas que esperan un espacio de acción discreto como es SB3. Es escalable y fácil de adaptar si se necesitan agregar más acciones en el futuro.

4.2.2. Problemas en la ejecución y comunicación con el proceso de Minecraft

Tras superar el primer obstáculo, nos enfrentamos a un segundo fallo que tardó más de un mes en encontrar solución. El escenario problemático emergía en la fase de ejecución del script de Python destinado a la creación y entrenamiento del modelo. Específicamente, al alcanzar aproximadamente los 120 mil timesteps[4] (2 horas de ejecución), el programa arrojaba un error de tipo *time out*. Este error evidenciaba una falla en la comunicación con el proceso de Minecraft, resultando en una terminación abrupta de la ejecución.

A pesar de nuestros persistentes esfuerzos y de un mes de comunicación constante con el equipo de desarrolladores de MineRL a través de la plataforma GitHub (referencia al problema en el issue: https://github.com/minerllabs/minerl/issues/725), la situación no llegó a una solución concluyente. El error parecía suponer un final para unas ejecuciones todavía sin resultados.

Solución

Ante la persistencia del problema y después de descartar nuestra máquina y código como factores causantes, centrarnos en la librería parecía el paso lógico. Optamos por revertir la versión de la librería desde la `1.0.0` a la `0.4.4`. Este cambio resultó en la solución del problema.

Nuestro análisis adicional reveló hallazgos interesantes para los desarrolladores de la librería. Observamos que la versión `0.4.4` ofrecía una velocidad de entrenamiento sus-

[4] Un *timestep* hace referencia a una interacción individual del agente con el entorno (step).

tancialmente superior a la versión `1.0.0`. Concretamente, mientras que la versión `1.0.0` requería cerca de 2 horas para procesar 120,000 timesteps, la versión `0.4.4` lo lograba en 1 hora y 20 minutos, reflejando una **mejora del 35 %** en eficiencia. Estos resultados no solo evidencian la solución al problema original, sino también una optimización considerable en el rendimiento del proceso de entrenamiento.

Versión de la librería	Tiempo para 120,000 timesteps	Mejora en velocidad
`1.0.0`	2 horas	-
`0.4.4`	1 hora y 20 minutos	35 %

Tabla 4.1: Comparación de tiempos de ejecución entre las versiones `1.0.0` y `0.4.4` de MineRL.

4.2.3. Evaluación de algoritmos de aprendizaje por refuerzo

Como se destacó en secciones anteriores de este trabajo (apartado 2.1.5), el aprendizaje por refuerzo en entornos complejos y dinámicos representa un desafío significativo. Esta complejidad intrínseca nos llevó a anticipar que la obtención de recompensas sería un proceso arduo, y los resultados de nuestras evaluaciones confirman esta expectativa.

Para esta evaluación, se optó por experimentar con dos algoritmos representativos de RL, Proximal Policy Optimization (PPO) y Deep Q-Network (DQN). Si bien ambos pertenecen al dominio del aprendizaje por refuerzo, representan enfoques distintos: PPO es un algoritmo orientado a la optimización de políticas, mientras que DQN se basa en la aproximación de la función de valor utilizando redes neuronales profundas. Esta dualidad en su naturaleza fue la razón principal detrás de su elección, ya que brindaba una oportunidad para contrastar y comparar dos metodologías diferentes en un mismo entorno.

Implementaciones con PPO y DQN

A continuación se detalla la implementaciones de los algoritmos PPO y DQN, y los detalles acerca de las decisiones tomadas en ambas:

1. **Uso de Stable Baselines3:** Durante el proceso de implementación, se utilizó la biblioteca Stable Baselines3 (SB3). Gracias a la naturaleza simplificada y eficiente de SB3, las implementaciones resultaron ser directas y sin complicaciones. Esta biblioteca facilita enormemente la tarea de trabajar con algoritmos de RL, proporcionando implementaciones estándar de alta calidad y permitiendo centrarse en la adaptación al entorno específico en lugar de en detalles técnicos del algoritmo.

2. **Selección de política y red neuronal:** Para nuestras implementaciones, optamos por utilizar la política *CnnPolicy* debido a la naturaleza de nuestras observaciones, que son imágenes. Esta política es óptima cuando las observaciones provienen de entornos visuales, ya que están diseñadas para procesar eficientemente información en formato de imagen. Al seleccionar *CnnPolicy* en Stable Baselines3 (SB3), la biblioteca automáticamente utiliza la red neuronal *NatureCNN*, que fue presentada en el artículo donde se introdujo DQN [43].

3. **Creación y entrenamiento de modelos PPO y DQN:** A continuación se muestra el código donde se define una función para crear el entorno `MineRLBasaltFindCave-v0` y se aplican varios wrappers, el ya mencionado `BitMaskWrapper` (4.2.1) y un wrapper `Monitor`, propio de la librería *gym*, que permite extraer información relevante del entrenamiento. Posteriormente, se muestra como inicializar y entrenar modelos

PPO o DQN utilizando *CnnPolicy*. Para monitorizar el proceso de entrenamiento, se utiliza la herramienta Wandb. Una vez completado el entrenamiento, el modelo se guarda para su uso futuro.

```
        # Funcion para crear el entorno y aplicar los wrappers
        def make_env():
            env = gym.make("MineRLBasaltFindCave-v0")
            env = BitMaskWrapper(env)
            env = Monitor(env, directory="monitor_results", force=True)
            return env

        env = make_env()
        print("Entorno creado!")

        # Elija el modelo que desee, simplemente descomente una de las siguientes lineas
        model = PPO(policy="CnnPolicy", env=env, device="cuda")  # Para PPO
        #model = DQN(policy="CnnPolicy", env=env, device="cuda")  # Para DQN
        print("Modelo creado!")

        # Entrenar el modelo y monitorizar usando Wandb
        model.learn(total_timesteps=5000000, callback=WandbCallback())

        # Guardar el modelo entrenado
        model.save("trained_model")
```

Listing 4.3: Implementación de PPO y DQN utilizando SB3.

4. **Experimentación con hiperparámetros:** En nuestro caso, realizamos diferentes experimentos modificando hiperparámetros como la tasa de aprendizaje, el tamaño del mini-batch, el tamaño del replay buffer, el factor de descuento γ, y el coeficiente de entropía, todo ello con el objetivo de lograr una convergencia más efectiva del algoritmo en el entorno de MineRLTreechop-v0.

   Para una comprender los hiperparámetros permitidos por los algoritmos y sus valores por defecto, se puede consultar la API de Stable Baselines3: PPO y DQN.

5. **Elección de timesteps:** La elección de ejecutar un total de 5 millones de timesteps fue basada en consideraciones prácticas y teóricas. Dado el alto grado de variabilidad y complejidad del entorno, un número elevado de timesteps es esencial para permitir que el agente explore y aprenda de una amplia gama de situaciones. Además, en investigaciones relacionadas y en la literatura de aprendizaje por refuerzo, es común que se realicen millones de timesteps para garantizar una adecuada exploración y convergencia del algoritmo, especialmente en entornos complejos.

   Por último, teniendo en cuenta nuestras capacidades computacionales y el tiempo de entrenamiento estimado, 5 millones de timesteps resultaron ser una cantidad viable y representativa para nuestros experimentos.

6. **Importancia del parámetro *device*:** El hiperparámetro `device` en la inicialización del algoritmo determina si se utilizará la GPU (“cuda”) o la CPU (“cpu”). Se destaca su relevancia al comparar el aumento de velocidad entre la GPU y la CPU:

$$\text{Aumento de velocidad} = \frac{\text{Tiempo con CPU}}{\text{Tiempo con GPU}} = \frac{625 \text{ horas}}{50 \text{ horas}} = 12{,}5 \text{ veces más rápido}$$

Análisis de resultados

A pesar de las variaciones y ajustes realizados, los resultados obtenidos con PPO y DQN en el entorno MineRLTreechop-v0 fueron **consistentemente insatisfactorios**. En cada iteración y ejecución de los algoritmos, la recompensa reportada era de 0, indicando que el agente no estaba aprendiendo una política efectiva para interactuar con el entorno.

Figura 4.2: Resultados de la ejecución de PPO y DQN en MineRLTreechop-v0.

Esta falta de recompensa era previsible y se puede atribuir a varias características intrínsecas de Minecraft:

1. **Complejidad del entorno:** MineRLTreechop-v0 presenta un mundo con una gran cantidad de variables, como diferentes tipos de árboles, terrenos y posibles interacciones. Esta riqueza de elementos crea una amplia gama de estados que el agente debe aprender a navegar, lo que representa un gran desafío para algoritmos como PPO y DQN.

2. **Escasez de recompensas:** En MineRLTreechop-v0, el objetivo es cortar árboles para obtener madera, lo cual implica una serie de acciones precisas como moverse hacia el árbol, alinear el cursor y talar la madera. Si el agente no realiza esta secuencia de acciones de manera efectiva, no recibe ninguna recompensa, haciendo que el algoritmo tenga dificultades para encontrar un camino de aprendizaje efectivo.

3. **Dilema de exploración vs. explotación:** El desafío de equilibrar la exploración de nuevas acciones con la explotación de las ya conocidas es agudizado en MineRLTreechop-v0 debido a su riqueza en acciones y estados y a la escasez de recompensas. Para abordar este desafío, se realizaron ajustes en el hiperparámetro γ. Sin embargo, determinar un equilibrio ideal en este entorno fue una tarea intrincada.

4. **Sensibilidad a hiperparámetros:** A pesar de numerosos ajustes de hiperparámetros, incluidos los relacionados con la tasa de aprendizaje y el tamaño del mini-batch, no se observó una mejora significativa en el rendimiento del algoritmo en MineRLTreechop-v0. Esto sugiere que los hiperparámetros óptimos para este entorno son difíciles de determinar y pueden diferir considerablemente de los valores comúnmente utilizados en entornos más simples.

Estas consideraciones, previamente discutidas (ver apartado 3.1.1), ilustran los desafíos inherentes al entrenamiento de algoritmos de aprendizaje por refuerzo en entornos tan complejos y heterogéneos como lo es MineRLTreechop-v0. Es evidente que, aunque

PPO y DQN son técnicas poderosas en muchos contextos, la naturaleza única y polifacética de este entorno requiere un enfoque más adaptado o, posiblemente, una combinación de técnicas.

El Imitation Learning (IL) emergió como una alternativa prometedora, no solo por su capacidad de aprovechar el conocimiento previo y adaptarse rápidamente a entornos donde las recompensas son escasas o difíciles de obtener, sino también porque la librería MineRL proporciona un dataset valioso que puede ser utilizado para facilitar el aprendizaje supervisado en estrategias de IL.

Con la decisión de explorar IL, buscamos determinar si esta técnica, ya sea sola o en combinación con otras, podría superar las limitaciones observadas con PPO y DQN en MineRLTreechop-v0. En las secciones siguientes, profundizaremos en los algoritmos de IL seleccionados, su implementación y los resultados obtenidos, contrastándolos con las experiencias previas.

4.2.4. Evaluación de algoritmos de aprendizaje por imitación

Después de observar las limitaciones y desafíos presentados por los algoritmos de RL en el entorno MineRLTreechop-v0, nos dirigimos hacia técnicas de Imitation Learning como una posible solución. El aprendizaje por imitación, en lugar de depender únicamente de la recompensa y la exploración, se basa en aprender a imitar comportamientos expertos. Con la disponibilidad del dataset proporcionado por MineRL [1], esta dirección parecía prometedora.

Con el propósito de llevar a cabo una evaluación metódica, optamos por comenzar nuestra investigación con algoritmos de IL puros. Queríamos establecer una línea base utilizando técnicas que aprendieran directamente de las demostraciones sin la necesidad de interacción adicional con el entorno o de una función de recompensa. Esto nos permitiría evaluar cuánto se podía lograr simplemente imitando el comportamiento experto presente en el dataset.

Sin embargo, el mundo del aprendizaje por imitación no se limita únicamente a la imitación pura. Reconociendo que el aprendizaje a partir de demostraciones tiene sus propias limitaciones (ver apartado 2.2.3), decidimos expandir nuestro análisis a algoritmos de IL con RL. Estos algoritmos, como GAIL y AIRL, buscan combinar las ventajas del aprendizaje supervisado con el aprendizaje por refuerzo, permitiendo al agente no solo aprender de las demostraciones, sino también adaptarse y explorar el entorno por sí mismo. Esta combinación prometía una mayor adaptabilidad y la posibilidad de superar el rendimiento del experto.

Con este enfoque bifásico, buscamos obtener una comprensión completa del potencial y las limitaciones de los algoritmos de Imitation Learning en el contexto de Minecraft.

Implementación de Behavioral Cloning

En este apartado, describiremos la implementación del algoritmo Behavioral Cloning (BC) utilizando la librería *Imitation*. Abordaremos los problemas que surgieron durante la implementación y cómo se resolvieron.

1. **Inicialización y configuración del entorno:**

   El entorno elegido es el mismo que el utilizado en el apartado anterior, `MineRLTreechop-v0`, cuyo objetivo recordemos que es obtener madera dentro de el mundo de Minecraft. Para la creación del entorno utilizamos la función `make_vec_env` de la librería *Imi-*

tation, especificando como entornos concurrentes únicamente uno y aplicando el `BitMaskWrapper` (ver apartado 4.2.1).

```
        # Crear un generador de numeros aleatorios
        rng = np.random.default_rng(0)

        # Crear el entorno para el modelo
        env = make_vec_env(
            "MineRLTreechop-v0",
            rng=rng,
            n_envs=1,
            post_wrappers=[
                lambda env, _: BitMaskWrapper(env),
            ],
        )
```

Listing 4.4: Creación del entorno MineRLTreechop-v0.

2. **Carga del dataset:**

   Para entrenar el modelo BC, es necesario un conjunto de datos de demostraciones de expertos. Para ello haremos uso del dataset proporcionado por MineRL para este entorno concreto, el cual está compuesto por trayectorias expertas descritas a través de una observación y las acciones tomadas en dicho estado. Usamos el método `minerl.data.make` para cargar el dataset `MineRLTreechop-v0`:

```
        # Cargar el dataset
        data = minerl.data.make(
            "MineRLTreechop-v0", data_dir="/data/dsaneng@alumno.upv.es/
                sb3"
        )
```

Listing 4.5: Cargar el dataset de MineRLTreechop-v0.

3. **Preprocesamiento del dataset:**

   Tras cargar el dataset, lo procesamos utilizando `buffered_batch_iter` para extraer transiciones, que incluyen estados actuales, acciones, estados siguientes y señales de finalización. En este punto, nos encontramos con un desafío similar al que discutimos en el apartado 4.2.1, pero en un contexto inverso. Mientras que el dataset representa las acciones en formato diccionario, los algoritmos de IL requieren acciones discretas. Para abordar este inconveniente, desarrollamos la función `batched_action_dict_to_discrete`, que convierte las acciones del formato diccionario al formato discreto, actuando de manera contraria al BitMaskWrapper.

   Una vez convertidas las acciones a números discretos, transponemos las observaciones para cuadrar el formato esperado por el algoritmo de canales, altura y anchura de las imágenes, y por último se agrupan las listas para formar un objeto de tipo *Transitions*, proporcionado por la librería Imitation para almacenar las trayectorias del experto.

```
        # Listas para almacenar las transiciones
        all_obs = []
        all_acts = []
        all_infos = []
        all_next_obs = []
        all_dones = []

        # Iterar sobre el dataset usando buffered_batch_iter
        iterator = BufferedBatchIter(data)
        for current_state, action_dict, _, next_state, done in iterator.
            buffered_batch_iter(
```

```
            num_epochs=10, batch_size=128
        ):
            # Convertir el diccionario de acciones a un discreto
            actions = batched_action_dict_to_discrete(action_dict, 128)

            all_obs.append(current_state["pov"])
            all_acts.append(actions)
            all_infos.append(np.zeros(128))
            all_next_obs.append(next_state["pov"])
            all_dones.append(done)

        # Concatenar las listas y crear un objeto con todas las
            transiciones
        all_obs = np.concatenate(all_obs, axis=0)
        all_acts = np.concatenate(all_acts, axis=0)
        all_infos = np.concatenate(all_infos, axis=0)
        all_next_obs = np.concatenate(all_next_obs, axis=0)
        all_dones = np.concatenate(all_dones, axis=0)

        # Transponer las observaciones
        all_obs = all_obs.transpose(0, 3, 1, 2)
        all_next_obs = all_next_obs.transpose(0, 3, 1, 2)

        transitions = types.Transitions(
            obs=all_obs,
            acts=all_acts,
            infos=all_infos,
            next_obs=all_next_obs,
            dones=all_dones,
        )
```

Listing 4.6: Recolección de demostraciones de expertos.

La función `batched_action_dict_to_discrete` convierte un conjunto de acciones representadas en un diccionario en una representación discreta codificada en binario. Específicamente, el valor de **threshold** determina cuándo un movimiento de la cámara es considerado significativo.

Si el threshold es alto, movimientos sutiles pueden ser ignorados, perdiendo detalles potencialmente importantes. Sin embargo, si es demasiado bajo, incluso pequeñas fluctuaciones podrían ser interpretadas como movimientos intencionados, introduciendo ruido. Este valor, por lo tanto, requiere una calibración cuidadosa para equilibrar precisión y robustez, representando un compromiso entre capturar movimientos genuinos y evitar el ruido accidental.

```
        def batched_action_dict_to_discrete(action_dict, batch_size):
        encoded_actions = []

        for i in range(batch_size):
            action_int = 0
            action_int |= action_dict["attack"][i]
            action_int |= action_dict["back"][i] << 1
            action_int |= action_dict["forward"][i] << 2
            action_int |= action_dict["left"][i] << 3
            action_int |= action_dict["right"][i] << 4
            action_int |= action_dict["jump"][i] << 5
            action_int |= action_dict["sneak"][i] << 6
            action_int |= action_dict["sprint"][i] << 7

            # Extraer los valores de la camara
            x, y = action_dict["camera"][i]

            # Especificar el umbral de movimiento
```

```
            threshold = 10

            camera_set = False

            if abs(x) > abs(y):
                # Movimiento horizontal
                if x < -threshold:
                    camera_dir = 0  # Izquierda
                    camera_set = True
                elif x > threshold:
                    camera_dir = 1  # Derecha
                    camera_set = True
            else:
                # Movimiento vertical
                if y < -threshold:
                    camera_dir = 2  # Abajo
                    camera_set = True
                elif y > threshold:
                    camera_dir = 3  # Arriba
                    camera_set = True

            if not camera_set:
                camera_dir = 4  # Sin movimiento

            action_int |= camera_dir << 8

            encoded_actions.append(action_int)

        return np.array(encoded_actions)
```

Listing 4.7: Función para traducir las acciones del experto a números discretos.

4. **Entrenamiento con Behavioral Cloning:**

   Con las transiciones preprocesadas, inicializamos el algoritmo BC proporcionando el espacio de observación, espacio de acción, política, demostraciones y otros parámetros relevantes. La política escogida es de la librería Stable Baselines3 del tipo ActorCriticCNNPolicy, la misma que utilizamos en el algoritmo PPO, ya que el algoritmo de BC únicamente permite políticas de tipo ActorCritic y las redes neuronales CNN son las más adecuadas para este tipo de escenarios. Por ultimo , entrenamos el modelo BC durante 15 epochs[5] utilizando el método `train` y finalmente guardamos la política aprendida.

```
        # Especificar la politica
        policy = ActorCriticCnnPolicy(
            env.observation_space, env.action_space, lr_schedule=lambda _
                : 1e-3
        )

        # Inicializacion de BC
        bc_trainer = BC(
            observation_space=env.observation_space,
            action_space=env.action_space,
            policy=policy,
            demonstrations=transitions,
            batch_size=128,
            device="cuda",
            rng=rng,
        )

```

[5]Un *epoch* en Machine Learning se refiere a una pasada completa de los datos de entrenamiento por el algoritmo

```
        bc_trainer.train(n_epochs=15)

        bc_trainer.save_policy("BC_model")
```

Listing 4.8: Implementación de Behavioral Cloning usando la librería Imitation.

5. **Evaluación del modelo:**

   Una vez finalizado el entrenamiento, es esencial **evaluar el rendimiento** del modelo. Stable Baselines3 facilita esta tarea mediante la función *evaluate_policy*. Esta función permite evaluar el agente en un número específico de episodios y, dependiendo de los parámetros establecidos, devolver la recompensa media y la desviación estándar por episodio. Además, se puede utilizar el callback *EvalCallback* para evaluar periódicamente el desempeño de un agente en un entorno de prueba separado, permitiendo guardar el mejor modelo y registrar los resultados de la evaluación. Estas herramientas son esenciales para entender cómo se desempeña un agente entrenado y ajustar su entrenamiento en consecuencia.

   En nuestro caso, evaluamos la política durante 50 episodios para comprobar como se comporta y los resultados finales son enviados a Wandb, para la posterior creación de gráficas y análisis de los resultados.

```
        # Evaluar la politica aprendida
        rewards, lengths = evaluate_policy(
            bc_trainer.policy,
            env,
            n_eval_episodes=50,
            render=False,
            return_episode_rewards=True,
        )

        # Enviar los resultados a Wandb
        for episode, (reward, length) in enumerate(zip(rewards, lengths))
            :
            wandb.log(
                {"episode_reward": reward, "episode_length": length, "
                    episode": episode}
            )
```

Listing 4.9: Evaluación de la política aprendida por BC.

Análisis de resultados de Behavioural Cloning

En este apartado, se presentan y discuten los resultados obtenidos tras la implementación y evaluación del modelo de BC.

Primera aproximación: Haciendo uso de la implementación descrita en el apartado anterior, pudimos observar resultados prometedores en comparación con los obtenidos con RL (ver Figura 4.2). Durante la evaluación de la política a lo largo de 50 episodios, nuestro agente logró acumular un total de **4 recompensas**. El resultado, si bien no es el ideal, sugiere que el enfoque BC que hemos adoptado está en el camino correcto, asimilando algunas de las estrategias cruciales para el entorno, con potencial de mejora.

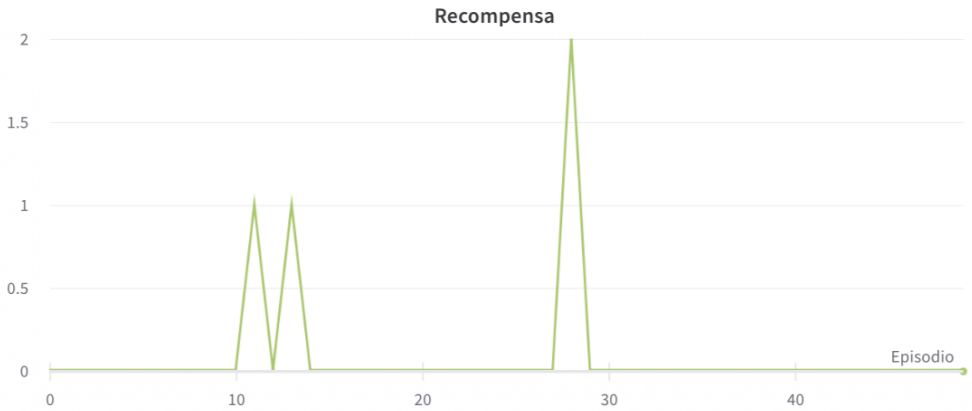

Figura 4.3: Resultados de la primera aproximación de Behavioural Cloning.

Vistos los resultados, nos planteamos cómo podrían ser mejorados. A pesar de la tentación inicial de intentar mejorar el rendimiento simplemente añadiendo más datos, pronto nos dimos cuenta de que esta estrategia tenía limitaciones. Una ampliación en la cantidad de datos no era viable; nuestro dataset ya excedía los 200 GB, lo que suponía un reto en términos de memoria y capacidad de procesamiento. Además, los tiempos de entrenamiento con este volumen de datos eran considerablemente extensos. Estas restricciones nos llevaron a concluir que, en lugar de tratar de optimizar el modelo añadiendo más datos o extendiendo los tiempos de entrenamiento, deberíamos centrarnos en la eficiencia de nuestra implementación.

Identificamos que una posible área de mejora era la función encargada de traducir las acciones del dataset. Dado que este proceso de traducción es fundamental para que el modelo comprenda y aprenda correctamente de los datos, nos propusimos refinar esta función para asegurar que las acciones se tradujeran de la manera más precisa y coherente posible. Esta decisión se basó en la idea de que una mejora en la calidad de la traducción podría tener un impacto significativo en la eficacia del entrenamiento y, en última instancia, en la capacidad del modelo para acumular recompensas en el entorno.

Segunda aproximación: Dentro de la implementación inicial de la función *batched_action_dict_to_discrete*, identificamos dos áreas críticas que requerían optimización para mejorar el rendimiento de nuestro agente en el entorno `MineRLTreechop-v0`.

La **primera área de mejora** estaba relacionada con la acción de ataque. Dado que el objetivo principal en `MineRLTreechop-v0` es cortar bloques de madera, surgió la idea de que mantener la acción de ataque activa de manera continua podría ser beneficioso. Esta hipótesis se basa en la suposición de que, al estar constantemente en modo ataque, el agente estaría en una posición óptima para interactuar con los bloques de madera tan pronto como se encuentren en su campo de visión. Para implementar este cambio en nuestro código, tuvimos que modificar la función *_apply_bit_mask* dentro del `BitMaskWrapper`, activando constantemente la acción de ataque. Dado que esta acción ya no sería controlada por un bit específico, se ajustaron los desplazamientos de bits para las demás acciones:

```
    def _apply_bit_mask(self, action):
        # Crear un diccionario con la accion vacia
        dic_action = self.noop_action.copy()

        # Accion de ataque siempre activada
        dic_action["attack"] = 1

        # Ajustar la decodificacion sin el bit de ataque
        dic_action["back"] = action & 1
        dic_action["forward"] = (action >> 1) & 1
        dic_action["left"] = (action >> 2) & 1
        dic_action["right"] = (action >> 3) & 1
        dic_action["jump"] = (action >> 4) & 1
        dic_action["sneak"] = (action >> 5) & 1
        dic_action["sprint"] = (action >> 6) & 1

        # Decodificar la accion de camara (4 bits)
        camera_dir = (action >> 7) & 0b1111
        cam_x, cam_y = 0, 0

        # Decodificar movimiento vertical
        if camera_dir & 0b0100:  # Abajo
            cam_x = 20
        elif camera_dir & 0b1000:  # Arriba
            cam_x = -20

        # Decodificar movimiento horizontal
        if camera_dir & 0b0001:  # Izquierda
            cam_y = -20
        elif camera_dir & 0b0010:  # Derecha
            cam_y = 20

        dic_action["camera"] = [cam_x, cam_y]

        return dic_action
```

Listing 4.10: Modificación del BitMaskWrapper.

El **segundo aspecto crítico** fue la codificación del movimiento de la cámara. En Minecraft, es esencial dirigir el cursor central hacia el bloque que se desea romper para poder interactuar con el. Por lo tanto, tener una representación precisa del movimiento de la cámara es crucial para el éxito en este entorno. La implementación inicial de la función dividía el movimiento de la cámara en dos categorías distintas: horizontal y vertical. Esta división resultaba en una representación simplificada que no capturaba la riqueza del movimiento real del jugador. Decidimos, entonces, tratar ambos movimientos de manera simultánea, permitiendo una codificación más rica y precisa del movimiento de la cámara.

Con estas optimizaciones, en la función resultante se omite el bit de ataque, ya que este estará siempre activo, y se centra en una representación más detallada del movimiento de la cámara, esperando que estas modificaciones permitan al agente actuar de manera más efectiva en el entorno y, por ende, obtener mejores recompensas.

```
    def batched_action_dict_to_discrete(action_dict, batch_size):
        encoded_actions = []

        for i in range(batch_size):
            action_int = 0

            # Adjusted encoding without the attack bit
            action_int |= action_dict["back"][i]
            action_int |= action_dict["forward"][i] << 1
            action_int |= action_dict["left"][i] << 2
            action_int |= action_dict["right"][i] << 3
            action_int |= action_dict["jump"][i] << 4
            action_int |= action_dict["sneak"][i] << 5
            action_int |= action_dict["sprint"][i] << 6

            # Encoding camera actions
            x, y = action_dict["camera"][i]
            threshold = 20
            camera_dir = 0b0000  # Sin movimiento

            # Encode vertical movement
            if x > threshold:
                camera_dir |= 0b0100  # Abajo
            elif x < -threshold:
                camera_dir |= 0b1000  # Arriba

            # Encode horizontal movement
            if y > threshold:
                camera_dir |= 0b0010  # Derecha
            elif y < -threshold:
                camera_dir |= 0b0001  # Izquierda

            action_int |= camera_dir << 7

            encoded_actions.append(action_int)

        return np.array(encoded_actions)
```

Listing 4.11: Modificación de la función de traducción.

Una vez aplicadas ambas modificaciones a las funciones que traducen las acciones para el algoritmo y el entorno, procedimos a reentrenar nuestro modelo BC y evaluar su desempeño con las nuevas optimizaciones. Estas mejoras, al estar directamente relacionadas con la forma en que el agente interpreta y actúa en el entorno, tenían el potencial de tener un impacto significativo en el rendimiento general del agente.

Como podemos observar en la imagen 4.4, tras implementar las optimizaciones en nuestra función de traducción de acciones, el rendimiento del modelo BC **mejoró sustancialmente**. Durante la evaluación del modelo optimizado a lo largo de los mismos 50 episodios, el agente logró acumular un total de **11 recompensas**. Esta cifra, en comparación con las 4 recompensas obtenidas por el modelo anterior en el mismo número de episodios, representa una mejora considerable y evidencia el potencial de nuestras optimizaciones.

Estos resultados no solo validan la importancia de una correcta traducción y representación de las acciones en entornos complejos como `MineRLTreechop-v0`, sino que también destacan la relevancia de entender las dinámicas específicas del juego. En este caso, el acto constante de atacar y una representación más precisa del movimiento de la cámara resultaron ser elementos clave para mejorar el desempeño del agente.

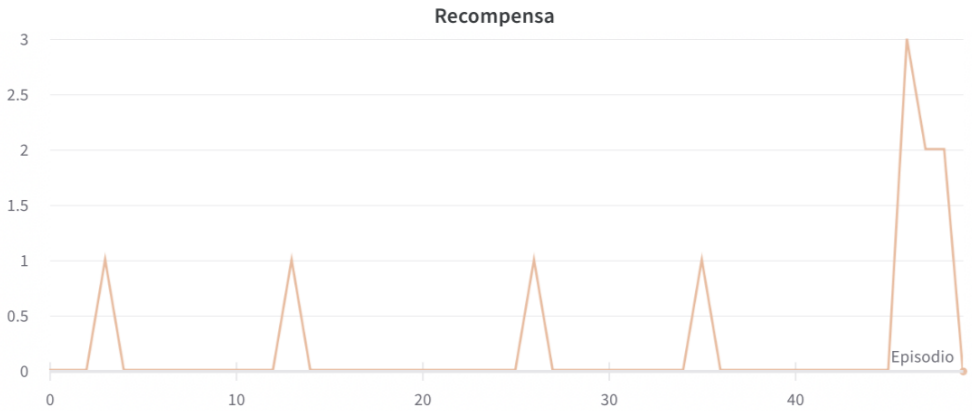

Figura 4.4: Resultados de la segunda aproximación de Behavioural Cloning.

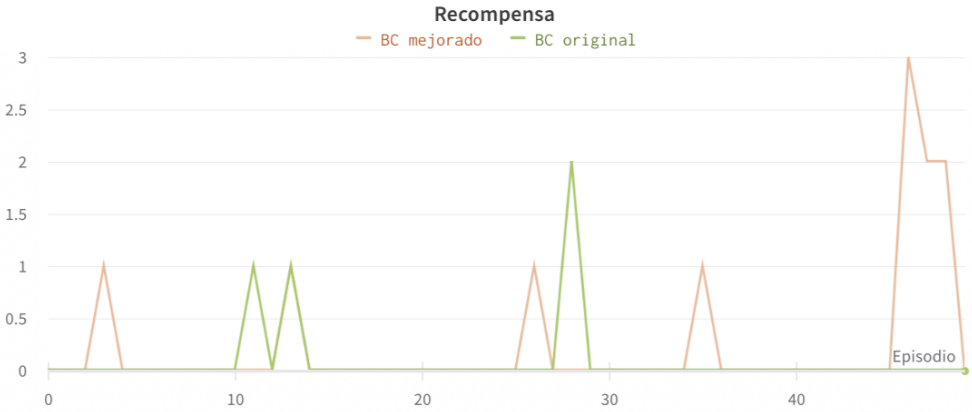

Figura 4.5: Comparativa de resultados entre ambos modelos.

Conclusión: Con estos hallazgos, se pone de manifiesto que, aunque el proceso de aprendizaje por imitación tiene sus propios desafíos, con las adecuadas optimizaciones y un entendimiento profundo del entorno, es posible obtener resultados significativamente mejores. Es alentador observar cómo pequeños ajustes en la traducción de acciones pueden tener un impacto tan grande en el desempeño global del agente. Además, estos resultados evidencian el amplio margen de mejora que aún existe en la traducción de acciones.

Posibles optimizaciones, como ajustar el *threshold* o añadir más bits para una codificación de acciones más granular, podrían permitir una representación aún más precisa y fiel de las acciones del jugador experto, resultando en una posible mejora en la recompensa obtenida. Lamentablemente, debido a limitaciones de tiempo, no pudimos explorar estas optimizaciones adicionales o la experimentación con algoritmos como DAgger, pero serían consideraciones valiosas para futuros trabajos en este ámbito.

Implementación de GAIL y AIRL

Para finalizar este trabajo y proporcionar una visión más completa de la interacción entre Aprendizaje por Refuerzo (RL) y Aprendizaje por Imitación (IL), nos embarcamos en la implementación de dos algoritmos que amalgaman estas disciplinas: Generative Adversarial Imitation Learning (GAIL) y Adversarial Inverse Reinforcement Learning (AIRL).

A medida que avanzamos en este último capítulo de nuestra investigación, es esencial entender que la combinación de RL e IL no es simplemente la suma de sus partes. Más bien, es una interacción compleja que, cuando se entrena de manera correcta, tiene el potencial de superar las limitaciones de cada enfoque individual.

- Nuestro punto de partida fue la estructura ya establecida durante las pruebas de Behavioural Cloning (BC). Mantuvimos intactas las funciones relacionadas con el `BitMaskWrapper` y las relativas al dataset de MineRL. Esto nos ofreció una base sólida y probada desde la que desarrollar y adaptar los nuevos algoritmos.

- Para el componente generativo en GAIL y AIRL, optamos por utilizar el algoritmo *Proximal Policy Optimization* (PPO). Esta elección no fue arbitraria; una de las razones clave detrás de esta decisión fue la comparabilidad, recordando las secciones anteriores donde ya habíamos experimentado con PPO en el contexto de RL puro. Además de la comparabilidad mencionada, optamos por PPO debido a su estabilidad y eficiencia en una amplia variedad de tareas y entornos. A diferencia de otros algoritmos de optimización de políticas, PPO ha sido diseñado específicamente para abordar los problemas de entrenamiento que surgen debido a las grandes actualizaciones de políticas, las cuales pueden hacer que el aprendizaje se vuelva inestable o incluso divergente. Esta característica hace de PPO una opción particularmente robusta para escenarios en los que la estabilidad del entrenamiento es esencial, como en nuestro caso, donde la integración de IL y RL ya introduce complejidades adicionales.

- GAIL y AIRL, al ser adversariales, requieren de una *reward net* o *red de recompensa*. Esta red tiene la tarea de distinguir entre las trayectorias generadas por la política actual del agente y las trayectorias del experto, y guía al agente para que sus acciones se asemejen lo más posible a las del experto. Es una pieza esencial para que estos algoritmos funcionen eficazmente.

- La implementación se vio enormemente facilitada gracias a las bibliotecas Imitation y Stable Baselines3. Estas herramientas, además de ser robustas y eficientes, proporcionan una estructura modular y fácilmente adaptable. Específicamente, el diseño de estas bibliotecas nos permitió reutilizar gran parte del código escrito para GAIL cuando implementamos AIRL. Esta reutilización no solo ahorra tiempo, sino que también garantiza consistencia y reduce la posibilidad de errores.

El código proporcionado a continuación demuestra el proceso de implementación de GAIL. Como se puede observar, el código sigue la estructura familiar utilizada en la implementación de BC, resaltando el potencial de simpleza y comprensión que estas bibliotecas ofrecen.

```
1    # Crear el entorno para el modelo
2    env = make_vec_env(
3        "MineRLTreechop-v0",
4        rng=rng,
5        n_envs=1,
```

```
        post_wrappers=[
            lambda env, _: BitMaskWrapper(env),
        ],
    )

    # Inicializar el algoritmo generador
    gen_algo = PPO("CnnPolicy", env, verbose=1)

    # Inicializar la reward net de tipo CNN (imagenes)
    # Establecer hwc_format = False para respetar el formato
    # establecido en las observaciones
    reward_net_gail = CnnRewardNet(
        observation_space=env.observation_space,
        action_space=env.action_space,
        use_state=True,
        use_action=True,
        hwc_format=False,
    )

    # Inicializar GAIL
    gail_trainer = GAIL(
        demonstrations=transitions,
        demo_batch_size=128,
        venv=env,
        gen_algo=gen_algo,
        reward_net=reward_net_gail,
        allow_variable_horizon=True,
    )

    # Entrenar y evaluar la politica aprendida
    gail_trainer.train(200000)

    rewards, lengths = evaluate_policy(
        gail_trainer.policy,
        env,
        n_eval_episodes=50,
        render=False,
        return_episode_rewards=True,
    )
```

Listing 4.12: Implementación de GAIL.

La adaptación del código para utilizar el algoritmo AIRL en lugar de GAIL es sorprendentemente sencilla, aprovechando la modularidad y flexibilidad de las bibliotecas que estamos utilizando. Concretamente, para implementar AIRL, solo es necesario instanciar el entrenador de AIRL con la configuración adecuada, manteniendo la mayoría de los parámetros de GAIL intactos. La sección de código relevante es la siguiente:

```
    # Inicializar AIRL
    airl_trainer = AIRL(
        demonstrations=transitions,
        demo_batch_size=128,
        venv=env,
        gen_algo=gen_algo,
        reward_net=reward_net_airl,
        allow_variable_horizon=True,
    )
```

Aquí, seguimos utilizando PPO como algoritmo generativo, lo que nos permite no solo mantener la coherencia con las pruebas anteriores, sino también beneficiarnos de las ventajas que PPO ofrece como algoritmo de aprendizaje por refuerzo. La CNNRewardNet continúa siendo la elección para el *reward_net*, dada su capacidad para procesar las

observaciones visuales del entorno de MineRL. Por último, pero no menos importante, el parámetro *allow_variable_horizon* se establece en *True* debido a la variabilidad en la longitud de las trayectorias expertas, garantizando que el algoritmo pueda manejar trayectorias de diferentes longitudes sin problemas. Esta adaptabilidad subraya la eficiencia y facilidad con la que se pueden probar y comparar diferentes algoritmos de imitación utilizando las bibliotecas actuales.

Análisis de resultados de GAIL y AIRL

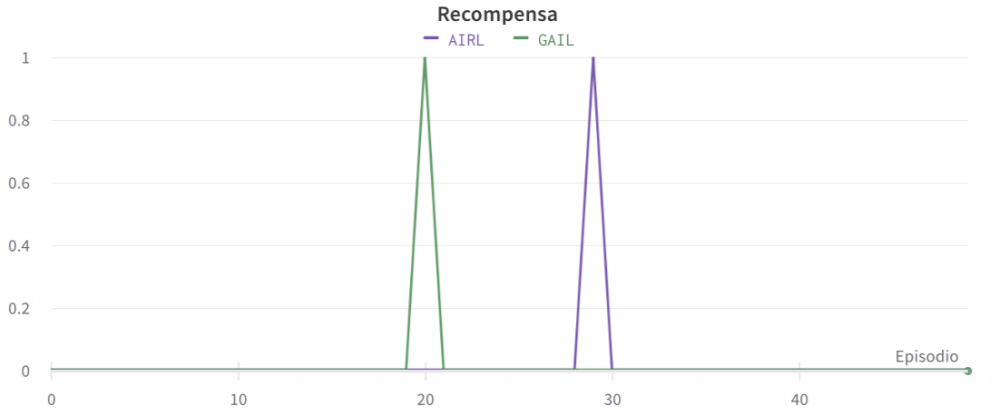

Figura 4.6: Resultados de los modelos GAIL y AIRL.

Al analizar la Figura 4.6, que muestra los resultados obtenidos con GAIL y AIRL, reconocemos oportunidades valiosas para el análisis y la mejora. A pesar de que los resultados reflejan **una recompensa** en 50 episodios, esto subraya la importancia de entender más profundamente las dinámicas del algoritmo, sentando las bases para futuras optimizaciones. La información proporcionada por la implementación nos permite identificar áreas clave de mejora y aprendizaje:

1. **Desempeño de PPO**: El hecho de que PPO no haya conseguido ninguna recompensa por sí solo (ver apartado 4.2.3) es un indicador fundamental de los desafíos que enfrenta el agente en el entorno MineRLTreechop-v0. Como GAIL y AIRL utilizan PPO como algoritmo generativo, su desempeño está intrínsecamente ligado al de PPO. Si PPO, en su forma básica, no puede adaptarse y aprender en el entorno, es probable que las versiones de imitación del aprendizaje, que dependen en gran medida de la capacidad del algoritmo generativo para aprender y mejorar, también tengan un rendimiento insatisfactorio. En esencia, si el fundamento (PPO) es inestable, las estructuras construidas sobre él (GAIL y AIRL) también lo serán.

2. **Duración del entrenamiento**: 100,000 timesteps, aunque pueda parecer una cantidad significativa, puede no ser suficiente para entrenar adecuadamente un modelo en un entorno tan complejo como MineRLTreechop-v0. Es posible que el modelo simplemente no haya tenido tiempo suficiente para aprender y adaptarse al entorno. Esta cantidad de timesteps se traduce en casi un día completo de espera hasta obtener resultados, lo cual limitó nuestra capacidad para realizar iteraciones y mejoras en el modelo en un plazo razonable. Las restricciones temporales y la naturaleza compleja del entorno, con su amplio espacio de acción y observación, sugieren que para ver mejoras significativas, podrían ser necesarios entrenamientos de varios millones de timesteps.

3. **Complicaciones inherentes de GAIL y AIRL**: Ambos algoritmos dependen de una combinación de políticas aprendidas de demostraciones expertas y recompensas adversariales. Si el algoritmo generativo subyacente (PPO) no puede producir recompensas significativas por sí mismo, es posible que las señales de recompensa adversariales no sean lo suficientemente fuertes o claras para guiar al agente hacia comportamientos deseables. En resumen, si PPO no puede aprender el entorno, GAIL y AIRL pueden no recibir la guía adecuada para aprender de las demostraciones expertas.

4. **Futuras líneas de trabajo**: Es evidente que uno de los próximos pasos sería prolongar el entrenamiento y observar si hay una mejora en el rendimiento. Además, sería valioso investigar otras configuraciones o variantes de PPO, o incluso considerar diferentes algoritmos generativos que puedan ser más adecuados para este entorno.

Conclusión: Los experimentos han revelado que, para la complejidad inherente de este problema, los algoritmos de Imitation Learning son notablemente más efectivos. No obstante, las demandas en términos de tiempo de entrenamiento y capacidades de hardware exceden los recursos con los que contábamos para este trabajo, lo que deja una clara dirección para futuras investigaciones.

CAPÍTULO 5
Conclusiones

A lo largo de este Trabajo Fin de Grado, se han explorado diferentes enfoques dentro de los ámbitos del Aprendizaje por Refuerzo (RL) y el Aprendizaje por Imitación (IL) aplicados específicamente en el entorno de Minecraft. A continuación, se presentan las principales conclusiones derivadas del estudio y las pruebas realizadas:

- **Comparativa entre RL e IL**: Las pruebas con algoritmos de RL no arrojaron resultados positivos en términos de recompensas, contrastando con las pruebas de IL, que mostraron resultados más prometedores. Estos hallazgos reafirman la complejidad y desafíos inherentes a la aplicación de RL en entornos dinámicos y complejos, mientras que resaltan las ventajas y la eficacia de las técnicas de IL en escenarios donde se dispone de datos previos o experiencias de expertos.
- **Desafíos y limitaciones**: Una de las principales barreras encontradas en este trabajo ha sido el tiempo. Los algoritmos, especialmente en entornos complejos como Minecraft, requieren un tiempo considerable de ejecución para obtener resultados significativos. En promedio, cada ejecución tardaba más de 16 horas, con algunos algoritmos alcanzando tiempos máximos de ejecución cercanos a los 2 días y medio. Esta duración extensa limitaba considerablemente la capacidad de iterar rápidamente y realizar mejoras consecuentes, suponiendo un limitante en términos de la exploración y optimización de las soluciones propuestas.
- **Trabajos futuros**: Existen varias direcciones prometedoras para expandir este trabajo en investigaciones futuras:
  - *Implementación de DAgger*: Una técnica de IL que no se pudo implementar en este trabajo, pero que tiene el potencial de mejorar aún más los resultados obtenidos con Behavioural Cloning (BC).
  - *Ampliación y optimización en la aplicación de GAIL y AIRL*: Aunque se han realizado pruebas iniciales con estos algoritmos, hay un amplio margen para refinamientos. Con acceso a más recursos de hardware y tiempo se podrían realizar ejecuciones más extensas, experimentar con diferentes hiperparámetros y explorar técnicas complementarias para mejorar el rendimiento.
  - *Exploración de otros algoritmos combinados RL e IL*: Si bien se ha tenido una primera toma de contacto con GAIL y AIRL, existen otros enfoques y técnicas que integran RL e IL. Sería de interés probar y comparar una gama más amplia de estos algoritmos en el entorno de Minecraft, evaluando sus fortalezas, debilidades y posibles aplicaciones.
  - *Aprendizaje en entornos multiagente*: Minecraft, al ser un entorno dinámico y versátil, ofrece la oportunidad de explorar situaciones en las que múltiples agen-

tes interactúan y aprenden juntos. Estudiar cómo los agentes pueden cooperar, competir o coexistir en este entorno puede abrir una nueva dimensión en la investigación de aprendizaje automático.

- *Exploración de paradigmas emergentes*: El campo del aprendizaje automático es dinámico y en constante evolución. Sería interesante adentrarse en paradigmas emergentes, como Minedojo, para descubrir nuevas técnicas y enfoques que puedan ser aplicados en entornos como Minecraft y evaluar su potencial en comparación con las técnicas tradicionales.

- **Reflexiones finales**: Este trabajo ha sido una oportunidad valiosa para adentrarse en el fascinante mundo del aprendizaje automático y sus aplicaciones en videojuegos. A pesar de los desafíos afrontados, el conocimiento adquirido y la capacidad de sobreponerse ellos ha sido gratificante. A pesar de los desafíos y el tiempo invertido, el hecho de ver a los agentes adaptarse y aprender en el entorno de Minecraft ha sido sumamente satisfactorio. En resumen, estoy muy complacido con los resultados alcanzados y el esfuerzo dedicado, y considero que este trabajo sienta unas bases para futuras investigaciones y aplicaciones en el campo del aprendizaje automático.

Con ello, este Trabajo Fin de Grado concluye, no como un final, sino como un punto de partida para futuras investigaciones y exploraciones en el dominio del aprendizaje automático y la inteligencia artificial aplicados en el área de los videojuegos.

Bibliografía

[1] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, Ruslan Salakhutdinov. *MineRL: A Large-Scale Dataset of Minecraft Demonstrations*. *arXiv:1907.13440*, 2019.

[2] OpenAI. OpenAI Spinning Up. https://spinningup.openai.com/en/latest/index.html

[3] Maxim Lapan. *Deep Reinforcement Learning Hands-On*. Packt, Moscow, Segunda edición, Enero 2020

[4] Watkins, C.J.C.H. Learning from delayed rewards. Ph.D. thesis, *University of Cambridge*, England, 1989.

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. Playing Atari with deep reinforcement learning. *arXiv:1312.5602*, 2013.

[6] Van Hasselt, H., Guez, A., Silver, D. Deep Reinforcement Learning with Double Q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[7] Schaul, T., Quan, J., Antonoglou, I., Silver, D. Prioritized experience replay. *arXiv:1511.05952*, 2016.

[8] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., Freitas, N. Dueling network architectures for deep reinforcement learning. *arXiv:1511.06581*, 2016.

[9] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4), 229-256, 1992.

[10] Konda, V. R., Tsitsiklis, J. N. Actor-critic algorithms. In *Advances in neural information processing systems* (pp. 1008-1014), 2000.

[11] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2016.

[12] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[13] Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning* (pp. 1889-1897), 2015.

[14] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, Igor Krawczuk. *Emergent Tool Use from Multi-Agent Autocurricula*. *arXiv:1909.07528*, 2019.

[15] Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, Thore Graepel. *Multi-agent Reinforcement Learning in Sequential Social Dilemmas*. *arXiv:1702.03037*, 2017.

[16] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., Quillen, D. *Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection*. The International Journal of Robotics Research, 37(4-5), 2018.

[17] Silver, D., Schrittwieser, J., Simonyan, K. *Mastering the game of Go without human knowledge*. Nature 550, 354–359, 2017.

[18] Jiang, Z., Xu, D. *Deep reinforcement learning for trading*. *arXiv:1706.10059*, 2017.

[19] Komorowski, M., Celi, L. A., Badawi, O., Gordon, A. C., Faisal, A. A. *The Artificial Intelligence Clinician learns optimal treatment strategies for sepsis in intensive care*. Nature Medicine, 24(11), 2018.

[20] Chakrabarty, A., Agrawal, P. *Deep reinforcement learning based approach for smart grid demand-side management*. Applied Energy, 238, 2019.

[21] Gabriel Dulac-Arnold, Daniel Mankowitz, Todd Hester. *Challenges of Real-World Reinforcement Learning*. *arXiv:1904.12901*, 2019.

[22] Richard S. Sutton, Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[23] Vincent A. W. J. Marchant and Philippe R. Schucht. *The Exploration-Exploitation Dilemma: A Multidisciplinary Framework*. PLoS ONE, 8(3):e58064, 2013.

[24] Matthew E. Taylor and Peter Stone. *Transfer Learning for Reinforcement Learning Domains: A Survey*. Journal of Machine Learning Research, 10:1633-1685, 2009.

[25] Dean A. Pomerleau. *Efficient Training of Artificial Neural Networks for Autonomous Navigation*. Neural Computation, 3(1):88–97, 1991.

[26] A brief overview of Imitation Learning. *Medium*. Consultado en https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c.

[27] Stephane Ross, Geoffrey J. Gordon, Drew Bagnell. *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*. Proceedings of the fourteenth international conference on artificial intelligence and statistics, PMLR 15:627-635, 2011.

[28] J. Ho, S. Ermon. *Generative Adversarial Imitation Learning*. Advances in Neural Information Processing Systems, 29:4565-4573, 2016.

[29] J. Fu, K. Luo, S. Levine. *Learning Robust Rewards with Adversarial Inverse Reinforcement Learning*. Proceedings of the International Conference on Learning Representations (ICLR), 2017.

[30] Andrew Y. Ng and Stuart J. Russell. *Algorithms for Inverse Reinforcement Learning*. Proceedings of the Seventeenth International Conference on Machine Learning (ICML), pages 663–670, 2000.

[31] Jonathan Ho, Pieter Abbeel. *Generative Adversarial Imitation Learning*. *arXiv:1606.03476*, 2016.

[32] Justin Fu, Sergey Levine, Pieter Abbeel. *Learning Robust Rewards with Adversarial Inverse Reinforcement Learning*. *arXiv:1710.11248*, 2017.

[33] Anna L. Osherovich, Vladimir V. Rabchevsky, Yael Edan, Ben-Gurion. *Robotic Ironing with 3D Perception and Force/Torque Feedback in Household Environments*. Journal of Mechanical Design, 2018.

[34] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, et al. *End to End Learning for Self-Driving Cars*. *arXiv:1604.07316*, 2016.

[35] Emilio Parisotto, Jimmy Lei Ba. *Value-Prediction Networks*. Proceedings of the International Conference on Learning Representations (ICLR), 2018.

[36] Lena Maier-Hein, Swaroop S. Vedula, Stefanie Speidel, et al. *Surgical Data Science for Next-generation Interventions*. Nature Biomedical Engineering, 1(9):691–696, 2017.

[37] Stéphane Ross, Geoffrey J. Gordon, Drew Bagnell. *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pages 627–635, 2011.

[38] Mojang. *Minecraft Official Site*. Disponible en: https://www.minecraft.net/.

[39] M. U. Neto, F. V. Brasileiro, W. D. S. Rosa. *A Review on the Use of Minecraft in Education*. Creative Education, 9(6):817–828, 2018.

[40] M. Nebel, S. Schneider, S. Rey. *Educational Benefits of Minecraft: A Literature Review*. Proceedings of the 10th European Conference on Games Based Learning: ECGBL2016, 8:421, 2016.

[41] Singh, S. *Minecraft as a Platform for Project-Based Learning in AI*. Proceedings of the AAAI Conference on Artificial Intelligence, 34(09), 13504-13505, Apr. 2020. https://doi.org/10.1609/aaai.v34i09.7070.

[42] William H. Guss, Cayden Codel, Katja Hofmann, et al. *The MineRL Competition on Sample Efficient Reinforcement Learning using Human Priors*. *arXiv:1904.10079*, NeurIPS Competition Track, 2019.

[43] V. Mnih, K. Kavukcuoglu, D. Silver, et al. *Human-level control through deep reinforcement learning*. Nature, 518(7540):529–533, 2015.

[44] David Silver, T. Hubert, Julian Schrittwieser, et al. *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*. Science, 362(6419):1140-1144, 2018. Consultado en https://www.science.org/doi/10.1126/science.aar6404.

[45] OpenAI. *OpenAI Five*. OpenAI Blog, 2019. Consultado en https://openai.com/blog/openai-five/.

[46] DeepMind. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. DeepMind Blog, 2019. Consultado en https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii.

[47] Tessler, C., Givony, S., Zahavy, T., Mankowitz, D., Mannor, S. *A Deep Hierarchical Approach to Lifelong Learning in Minecraft*. Proceedings of the AAAI Conference on Artificial Intelligence, 31(1), 2017. Consultado en https://doi.org/10.1609/aaai.v31i1.10744.

[48] Johnson M., Hofmann K., Hutton T., Bignell D. *The Malmo Platform for Artificial Intelligence Experimentation*. Proc. 25th International Joint Conference on Artificial Intelligence, Ed. Kambhampati S., p. 4246. AAAI Press, Palo Alto, California USA, 2016. Consultado en https://github.com/Microsoft/malmo.

[49] Ross, S., Gordon, G., Bagnell, D. (2011). *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR 15:627-635.

[50] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Sunehag, P. (2019). *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. DeepMind Blog.

[51] Rajeswaran, A., Kumar, V., Gupta, A., Schulman, J., Todorov, E., Levine, S. *Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations*. Proceedings of Robotics: Science and Systems (RSS), 2018.

[52] Stephanie Milani, Nicholay Topin, Brandon Houghton, William H. Guss, Sharada P. Mohanty, Keisuke Nakata, Oriol Vinyals, and Noboru Sean Kuno, *Retrospective Analysis of the 2019 MineRL Competition on Sample Efficient Reinforcement Learning*. Proceedings of Machine Learning Research: NeurIPS 2019 Competition & Demonstration Track Postproceedings, 2019. Consultado en https://doi.org/10.48550/arXiv.2003.05012.

[53] William H. Guss, Mario Ynocente Castro, Sam Devlin, Brandon Houghton, Noboru Sean Kuno, Crissman Loomis, Stephanie Milani, Sharada Mohanty, Keisuke Nakata, Ruslan Salakhutdinov, John Schulman, Shinya Shiroshita, Nicholay Topin, Avinash Ummadisingu, Oriol Vinyals. *The MineRL 2020 Competition on Sample Efficient Reinforcement Learning using Human Priors*. NeurIPS, 2020. Consultado en https://doi.org/10.48550/arXiv.2101.11071.

[54] William H. Guss, Mario Ynocente Castro, Sam Devlin, Brandon Houghton, Noboru Sean Kuno, Crissman Loomis, Stephanie Milani, Sharada Mohanty, Keisuke Nakata, Ruslan Salakhutdinov, John Schulman, Shinya Shiroshita, Nicholay Topin, Avinash Ummadisingu, Oriol Vinyals. *The MineRL 2020 Competition on Sample Efficient Reinforcement Learning using Human Priors*. NeurIPS, 2020. Consultado en https://doi.org/10.48550/arXiv.2101.11071.

[55] Stephanie Milani, Anssi Kanervisto, Karolis Ramanauskas, Sander Schulhoff, Brandon Houghton, Sharada Mohanty, Byron Galbraith, Ke Chen, Yan Song, Tianze Zhou, Bingquan Yu, He Liu, Kai Guan, Yujing Hu, Tangjie Lv, Federico Malato, Florian Leopold, Amogh Raut, Ville Hautamäki, Andrew Melnik, Shu Ishida, João F. Henriques, Robert Klassert, Walter Laurito, Ellen Novoseller, Vinicius G. Goecks, Nicholas Waytowich, David Watkins, Josh Miller, Rohin Shah. *Towards Solving Fuzzy Tasks with Human Feedback: A Retrospective of the MineRL BASALT 2022 Competition*. NeurIPS, 2022. Consultado en https://doi.org/10.48550/arXiv.2303.13512.

[56] OpenAI. Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos. *arXiv:2206.11795*, 2022.

[57] OpenAI. *Learning to play Minecraft with Video PreTraining*. Consultado en https://openai.com/research/vpt.

[58] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, Anima Anandkumar. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. *ar-*

Xiv:2206.08853, 2022. Comentarios: Outstanding Paper Award at NeurIPS 2022. Página web del proyecto: https://minedojo.org

[59] MineDojo Team. *MineDojo: Open-Ended Embodied Agents with Internet-Scale Knowledge GitHub Repository*. Open-Source Software Repository, 2023. Consultado en https://github.com/MineDojo/MineDojo.

[60] Portelas R., Colas C., Weng L., Hofmann K., Oudeyer P.-Y. *Automatic Curriculum Learning For Deep RL: A Short Survey*. *arXiv.2003.04664*, 2020.

[61] Voyager Team. *Voyager: An Open-Ended Embodied Agent with Large Language Models*. Consultado en https://voyager.minedojo.org.

[62] Wang G., Xie Y., Jiang Y., Mandlekar A., Xiao C., Zhu Y., Fan L., Anandkumar A. *Voyager: An Open-Ended Embodied Agent with Large Language Models*. *arXiv:2305.16291*, 2023.

[63] Zhu X., Chen Y., Tian H., Tao C., Su W., Yang C., Huang G., Li B., Lu L., Wang X., Qiao Y., Zhang Z., Dai J. *Ghost in the Minecraft: Generally Capable Agents for Open-World Environments via Large Language Models with Text-based Knowledge and Memory*. *arXiv:2305.17144*, 2023.

[64] MineRL Team. *MineRL Competition for Sample Efficient Reinforcement Learning - Python Package*. Open-Source Software Repository, 2019. Consultado en https://github.com/minerllabs/minerl.

[65] Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations. Consultado en https://stable-baselines3.readthedocs.io/.

[66] MineRL Docs - Singularity container for headless rendering with GPU. Consultado en https://minerl.readthedocs.io/en/latest/notes/performance-tips.html#singularity-container-for-headless-rendering-with-gpu.

[67] Adam Gleave, Mohammad Taufeeque, Juan Rocamonde, Erik Jenner, Steven H. Wang, Sam Toyer, Maximilian Ernestus, Nora Belrose, Scott Emmons, y Stuart Russell. *imitation: Clean Imitation Learning Implementations*. *arXiv:2211.11972v1 [cs.LG]*, 2022.

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

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

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

ODS 9 - Industria, innovación e infraestructuras:

Mi trabajo tiene una fuerte vinculación con el ODS 9 ya que se centra en el uso y promoción de tecnologías punteras en el ámbito de la inteligencia artificial, concretamente algoritmos de RL (Aprendizaje por Refuerzo) e IL (Aprendizaje por Imitación). Estos algoritmos no solo representan la vanguardia en el campo del aprendizaje automático, sino que también tienen el potencial de transformar la manera en que las industrias operan, mejorando la eficiencia y la innovación.

Además, a lo largo de mi trabajo, no sólo he implementado estas tecnologías, sino que también he procurado explicar y mostrar su funcionamiento y aplicabilidad, facilitando así su comprensión y eventual adopción por parte de otros profesionales e investigadores. Este enfoque educativo y divulgativo puede contribuir a acelerar la innovación en el sector y promover una industria más dinámica y sostenible. **(ODS 4)**

ODS 17 - Alianzas para lograr objetivos:

El desarrollo y éxito de mi trabajo no hubieran sido posibles sin la cooperación y colaboración internacional. A lo largo del proyecto, he colaborado estrechamente con el equipo de la librería MineRL, compuesto por ciudadanos de América. Esta cooperación ha sido esencial para resolver desafíos técnicos, optimizar la implementación de los algoritmos y mejorar en general la calidad del trabajo presentado.

Esta experiencia es un claro testimonio de cómo las alianzas internacionales y la cooperación pueden superar barreras, acelerar la investigación y producir resultados de mayor calidad. En un mundo cada vez más interconectado, la colaboración y la formación de alianzas estratégicas son esenciales para lograr objetivos más ambiciosos y enfrentar desafíos globales.

ETS Enginyeria Informàtica
Camí de Vera, s/n, 46022, València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es