

Universidad Internacional de Valencia

Exploration via Elliptical Episodic Bonuses en el entorno MiniGrid

Titulación:
Máster en Inteligencia Artificial
Curso académico 2022-2023

Alumno/a: Rúa Martínez, Alfonso
D.N.I: 34629527z

Director/a de TFM: González León, Borja

Convocatoria:

Primera

Abril 2023

De:
Planeta Formación y Universidades

Sic parvis magna.

Francis Drake

Agradecimientos

Quisiera agradecer a **Borja** por su disposición y empeño, sin su paciencia esto no hubiera sido posible. Ha sido corto pero intenso.

A **Iria**, por nuestra vida juntos.

Índice general

Índice de figuras III

Índice de tablas IV

Índice de algoritmos V

Resumen 1

1. Introducción 3
   1.1. Motivación 3
   1.2. Objetivos 4
   1.3. Estructura 5

2. Marco teórico 7
   2.0.1. Ventajas 11
   2.0.2. Desventajas 11

3. Estado del arte 17
   3.1. Exploración 17
   3.2. Entornos, Minigrid 19
   3.3. Algoritmos 22
   3.3.1. A2C 22
   3.3.2. PPO 25
   3.3.3. E3B 28

4. Metodología 33
   4.1. Propuesta 33
   4.2. Entorno experimental 34
   4.3. Adaptación propuesta 36

5. Resultados y Discusión 39

6. Conclusiones 42

7. Limitaciones y
Perspectivas de Futuro 45
7.1. Limitaciones 45
7.2. Lineas futuras 46

Lista de Acrónimos 49

Bibliografía 53

Índice de figuras

2.1. Space Invaders 7
2.2. Esquema básico de RL 12
2.3. Clasificación RL 12

3.1. Ejemplo de mapa en MiniHack 20
3.2. Mapa MultiRoom en MiniGrid 21
3.3. Mapa Obstructed Maze 2Dlh en MiniGrid 21
3.4. A2C vs A3C 23
3.5. PPO-Clip segun acción 26
3.6. Esquema de E3B 28
3.7. Recompensas elípticas en 2D 30

4.1. Mapa MiniGrid-MultiRoom-N2-S4-v0 34
4.2. Mapa MiniGrid-DoorKey-5x5-v0 34
4.3. Mapa MiniGrid-MultiRoom-N6-v0 35
4.4. Mapa MiniGrid-ObstructedMaze-Full-v0 35
4.5. Esquema Autoencoder 36

5.1. Rendimiento en entorno MiniGrid-MultiRoom-N2-S4-v0 39
5.2. Rendimiento en entorno MiniGrid-DoorKey-5x5-v0 39
5.3. Rendimiento en entorno MiniGrid-MultiRoom-N4-S5-v0 40
5.4. Rendimiento en entorno MiniGrid-MultiRoom-N6-v0 40
5.5. Rendimiento en MiniGrid-ObstructedMaze-2Dlh-v0 40

Índice de tablas

5.1. Tabla de rendimiento de los algoritmos 41

Índice de algoritmos

1. Algoritmo *Parallel Advantage Actor-Critic* (A2C) 24
2. Algoritmo *Proximal Policy Optimization* (PPO-Clip) 27
3. Algoritmo *Exploration via Episodic Elliptical Bonuses* (E3B) 31

Resumen

El presente trabajo de fin de máster aborda el estudio y aplicación del aprendizaje por refuerzo (RL) en algoritmos de inteligencia artificial, centrándose específicamente en la **E**xploración mediante **E**lliptical **E**pisodic **B**onuses (**E3B**) y su impacto en algoritmos *vanilla* de RL como PPO (Proximal Policy Optimization) y A2C (Advantage Actor-Critic). El aprendizaje por refuerzo permite a los agentes aprender de forma autónoma, interactuando con su entorno y ajustando sus acciones para maximizar recompensas acumuladas a lo largo del tiempo. La exploración es un aspecto clave para permitir a los agentes encontrar estrategias óptimas en entornos complejos.

Elliptical Episodic Bonuses (E3B) es una novedosa técnica de exploración que aborda el problema de equilibrar la explotación de conocimientos adquiridos y la exploración de nuevas acciones en entornos parcialmente observables. La propuesta de E3B es utilizar una distribución elíptica en lugar de una distribución uniforme para asignar bonificaciones episódicas a las acciones del agente. Esto fomenta una exploración más eficiente, enfocándose en áreas con mayor incertidumbre y evitando repetir acciones ya conocidas.

Para evaluar el desempeño de E3B, se ha empleado el juego MiniGrid como entorno de prueba. MiniGrid es un entorno de aprendizaje por refuerzo en el que los agentes deben aprender a realizar tareas simples en un mundo 2D. Este entorno presenta una gama de desafíos que requieren habilidades cognitivas básicas, como la percepción del entorno, el razonamiento y la planificación. Además, su simplicidad y flexibilidad permiten una implementación y comparación más sencilla de los métodos de este estudio.

El algoritmo E3B, ha sido propuesto en Enero del presente año, aplicado al entorno de purebas MiniHack así como en Habitat. En ambos entornos, el agente ha de moverse por un entorno dividido en salas, con el objetivo de explorarlo y obtener los objetivos especificados. El objetivo principal de este TFM es comprender la nueva técnica E3B mediante su estudio y análisis de cara a aplicarla en el entorno MiniGrid.

Los resultados obtenidos tratarán de probar que, al integrar la técnica de Elliptical Episodic Bonuses en los algoritmos base de RL, se logra un aumento en la eficiencia de la exploración y una mejora en la convergencia del aprendizaje. Este trabajo respaldará el uso de E3B como una valiosa herramienta para mejorar la capacidad de los agentes de aprendizaje por refuerzo en entornos parcialmente observables y complejos. La implementación exitosa de E3B abrirá nuevas posibilidades en la optimización de algoritmos de aprendizaje por refuerzo.

Introducción 1

Para contextualizar este trabajo, comenzaremos detallando las motivaciones del mismo, por qué motivo se ha escogido este área de la inteligencia artificial y que objetivos se buscan con este estudio. Tras ello, veremos una breve descripción de la estructura de esta memoria.

1.1. Motivación

El aprendizaje por refuerzo (Reinforcement Learning, RL) es un enfoque de la inteligencia artificial (IA) que permite a los agentes aprender a tomar decisiones de manera autónoma al interactuar con su entorno y ajustar sus acciones para maximizar las recompensas acumuladas a lo largo del tiempo.

Inspirado en la forma en que los seres vivos aprenden a partir de la experiencia y las consecuencias de sus acciones, el RL se basa en la idea de que un agente inteligente puede mejorar su rendimiento en una tarea a través de un proceso de ensayo y error.

El aprendizaje por refuerzo ha demostrado ser exitoso en una amplia gama de aplicaciones, desde el control de robots OpenAI et al. (2019b) y la optimización de procesos industriales hasta la creación de agentes inteligentes en videojuegos y sistemas de recomendación. La investigación en RL sigue avanzando rápidamente, con nuevas técnicas y algoritmos desarrollados continuamente para abordar desafíos en la eficiencia, la generalización y la escalabilidad del aprendizaje.

No obstante, el aprendizaje por refuerzo no es perfecto, a menudo requiere de demasiadas interacciones con el entorno y su comportamiento dependendiente de las recompensas no es útil en entornos con escasos o nulos premios. Es por ello que aparecen los estudios en el ámbito de la exploracion y las recompensas intrínsecas, los dos pilares en los que se fundamenta el algoritmo que trataremos aquí.

Se ha elegido esta rama de la inteligencia artificial para la elaboración de este TFM por sus posibilidades a la hora de enfrentarse a problemas complejos en entornos altamente dimensionales; siendo el enfoque detrás de grandes hitos de la IA como lo son AlphaGo Silver et al. (2016) y OpenAI Five OpenAI et al. (2019a), agentes capaces de superar las capacidades del mejor profesional humano en los juegos Go y Dota2 respectivamente.

1.2. Objetivos

El objetivo principal de este trabajo es investigar y evaluar el impacto de la técnica de exploración Elliptical Episodic Bonuses (E3B) en algoritmos de aprendizaje por refuerzo, específicamente Proximal Policy Optimization (PPO) y Advantage Actor-Critic (A2C). Para ello, se establecen los siguientes objetivos parciales:

1. **Objetivo parcial 1.**Revisar y analizar la literatura existente en aprendizaje por refuerzo, exploración y explotación, así como los algoritmos PPO y A2C. Este análisis proporcionará el marco teórico necesario para comprender y aplicar E3B en algoritmos de aprendizaje por refuerzo.

2. **Objetivo parcial 2.**Describir y estudiar en detalle la técnica de Elliptical Episodic Bonuses, su fundamentación teórica y su aplicación en algoritmos de aprendizaje por refuerzo. Esto permitirá una mejor comprensión de cómo E3B puede mejorar la exploración en entornos parcialmente observables.

3. **Objetivo parcial 3.**Implementar y adaptar los algoritmos PPO y A2C, integrando la técnica de E3B en su estructura. Ello facilitará la comparación entre los algoritmos originales y las versiones mejoradas con E3B.

4. **Objetivo parcial 4.**Diseñar y realizar experimentos utilizando el entorno de MiniGrid como plataforma de prueba. Los experimentos permitirán evaluar el desempeño de los algoritmos PPO y A2C con y sin E3B, analizando su eficacia y la velocidad de convergencia del aprendizaje.

5. **Objetivo parcial 5.**Analizar y comparar los resultados experimentales, identificando las ventajas y posibles limitaciones de la técnica E3B en los algoritmos PPO y A2C. Este análisis proporcionará una visión completa del impacto de E3B en el aprendizaje por refuerzo.

6. **Objetivo parcial 6.**Proponer posibles extensiones y lineas de investigación futuras de Elliptical Episodic Bonuses y de los algoritmos de aprendizaje por refuerzo en general. Esto contribuirá a expandir el campo de investigación y abrirá nuevas vías para mejorar la eficiencia en la exploración y aprendizaje de agentes autónomos.

1.3. Estructura

Tras haber visto la motivación del presente TFM así como los objetivos que se buscan, vamos a comentar la forma en la que estructuraremos el grueso de esta memoria.

- **Introducción**, la sección en la que nos encontramos y donde hemos visto el contexto y la motivación detrás del estudio, seguido de los objetivos que se buscan en este TFM, y finalizando con esta descripción general de la estructura.
- **Marco teórico**, aquí revisaremos los conceptos fundamentales del aprendizaje por refuerzo, se utilizará el ejemplo de entorno de un videojuego para facilitar su comprensión. También se discutirán sus ventajas y desventajas, así como la clasificación de los distintos modelos de RL.
- **Estado del arte**, esta sección estudiará el estado actual de la investigación en exploración de RL, se describiran los entornos de aprendizaje más relevantes y los algoritmos más comunes (A2C y PPO) junto con el algoritmo a estudio E3B.
- **Metodología**, en esta parte, presentaremos nuestra propuesta para investigar el impacto del algoritmo E3B en el aprendizaje por refuerzo. Detallaremos el entorno experimental seleccionado dentro de Minigrid, y cómo se adaptará el enfoque E3B en nuestro caso.
- **Resultados y discusión**, aquí, analizaremos y discutiremos los resultados obtenidos tras la experimentación, presentando gráficas y tablas para ilustrar cómo la aplicación de E3B afecta al rendimiento de los algoritmos en los entornos propuestos.
- **Conclusiones**, esta sección resumirá las deducciones obtenidas del estudio, así como la valoración y adecuación del proceso a los objetivos propuestos.
- **Limitaciones y Perspectivas de Futuro**, para finalizar, por una parte abordaremos las limitaciones del estudio, incluyendo limitaciones a nivel técnico, de los algoritmos y sus adaptaciones propuestas y posibles restricciones en la generalización de los resultados. Y por último, identificaremos oportunidades para investigaciones futuras en el área de la exploracion en RL, proponiendo extensiones y mejoras del enfoque actual.

Esta estructura asegura un flujo lógico en la presentación de la investigación, comenzando con los conceptos básicos y estado del arte, seguido de la experimentación y análisis de resultados y finalizando con las conclusiones y direcciones futuras.

Marco teórico

2

Como comentamos en el apartado anterior, este TFM se fundamenta en el aprendizaje por refuerzo (RL), el cual es un enfoque más dentro de la inteligencia artificial. En esta sección se hará un repaso de los conceptos más importantes de este tipo de aprendizaje para contextualizar y ayudar a entender las técnicas que se describirán en las subsiguientes secciones.

El objetivo final del RL es entrenar a un agente inteligente para que sea capaz de interactuar con un entorno de manera inteligente, pero... ¿qué es un Agente, qué es el Entorno?

Para definir y visualizar estos conceptos haremos uso de un ejemplo que nos ayudará a comprender cada parte del RL, para ello, el ejemplo elegido es un videojuego, en concreto uno sencillo como el clásico *Space Invaders* 2.1 de la consola Atari.

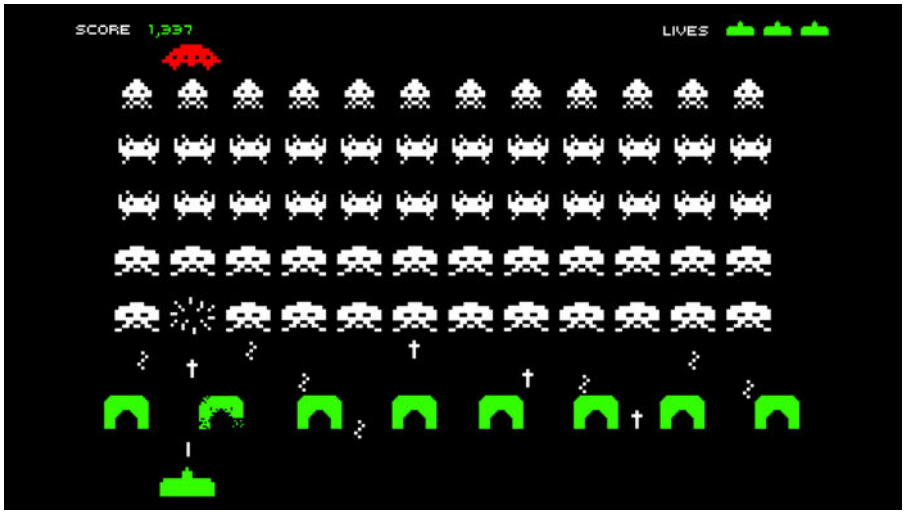


Figura 2.1: **Space Invaders**

- **Entorno**, es el mundo en el que vive el agente y con el que puede interactuar, es la simulación en la cual se mueve nuestro agente e incluye tanto los elementos visibles como la lógica y leyes que gobiernan dicha simulación (puntuaciones, vida, tiempos...). En nuestro ejemplo el entorno corresponde al juego en si: los pixeles que vemos en pantalla, la puntuación, la velocidad a la que nos movemos, los disparos...

- **Agente**, es la entidad que interactúa con el entorno con el fin de aprender, mejorar y lograr un objetivo. Por norma general los objetivos se basan en maximizar una recompensa. En nuestro caso, el agente sería el *"jugador invisible"* que toma las decisiones de como se

moverá la nave, cuando disparar, cuando esconderse, etc. Con la finalidad de obtener la mayor puntuación posible en el videojuego.

Lógicamente, el Agente no sabe como lograr su objetivo desde el principio, y logra este aprendizaje por medio de la interaccion con el Entorno, pero... ¿como interacciona? Veamos algunos conceptos básicos sobre esto:

- **Observación**, corresponde a cada *"fotografía"* del entorno, es decir, toda la información que el entorno puede ofrecernos en un momento dado. Tras cada interacción del agente esta observación cambiará (o no) y generará nueva información para el proceso de entrenamiento. En nuestro caso cada observación se correspondería con cada uno de los fotogramas del videojuego.
- **Estado**, es una adaptación del concepto anterior; si bien una observación corresponde con toda la información del entorno, a menudo esto no es eficiente a la hora de entrenar al agente y es por ello que se procesa para lograr una información que si está lista para consumir por el agente y tomar sus decisiones. En nuestro caso podríamos pasar el fotograma correspondiente por filtros, redimensiones, redes neuronales que extrajeran características de la imágen... la información final resultante sería el estado en ese momento.
- **Acción**, cada una de las decisiones que el agente puede tomar en el entorno. Para nuestro juego serían: mover la nave verde a un lado o al otro y disparar.
- **Espacio**, es el conjunto de todos los estados y acciones disponibles. En este caso dependería de como definieramos los estados.

Ahora bien, sabemos como el agente interacciona con el entorno, pero... ¿cómo sabe si está eligiendo las acciones correctas? Introduzcamos un concepto más:

- **Recompensa**, es el *feedback* que nos devuelve el entorno para evaluar cuán bien lo está haciendo el agente. En nuestro videojuego, la recompensa más evidente es la puntuación, pero también lo serían el numero de vidas o la velocidad (nivel) al que se mueve el juego. Debido a la importancia de este concepto en el presente TFM, vamos a ver los dos tipos de recompensas:
  - **Extrínsecas**, son las recompensas tal como las acabamos de definir, provienen directamente del entorno y a menudo son el objetivo principal que el agente ha de perseguir.
  - **Intrínsecas**, son recompensas generadas por el agente en lugar de por el entorno, están diseñadas para que el agente explore el entorno y aprenda formas más eficientes de interactuar con el. Generalmente están basadas en la novedad o el impacto de las acciones del agente. Hablaremos más sobre ellas en la sección 3.1. Un ejemplo en nuestro caso podría ser una recompensa negativa que incentive al agente a no destruir los bloques de protección.

- **Densas**, son recompensas que se entregan frecuentemente para proporcionar un *feedback* constante al agente y con ello lograr que el aprendizaje converja hacia un comportamiento óptimo. No obstante, pueden llevar al agente a realizar un aprendizaje superficial debido a que se enfoca en acciones a corto plazo en lugar de estrategias a largo plazo. Un ejemplo para nuestro juego podría ser recompensar a nuestro agente cada vez que se mueve a un lado o al otro, para incentivarlo a moverse.
- **Dispersas**, se refiere a las recompensas que se entregan raramente o sólo cuando se cumple un objetivo. Pueden hacer del aprendizaje un proceso más lento y complejo pero al mismo tiempo motivar al agente a descubrir soluciones a largo plazo para conseguir estas recompensas. En nuestro caso podría ser dar al agente una gran recompensa cuando logra acabar con todos los enemigos.

El diseño de recompensas es un aspecto crucial en RL ya que determina cómo se entrenará el agente y que soluciones logrará aprender, esto se explica de forma minuciosa en Sutton y Barto (1998).

Queda por ver cómo medimos los tiempos, para ello definimos:

- **Iteración o *Step***, cada uno de los instantes de tiempo en los que se obtiene una observación, es decir, es la medida de tiempo mínima en nuestro entorno. Resulta fundamental para medir los tiempos y facilidad de aprendizaje de cada algoritmo. En nuestro caso cada *step* es el conjunto de elementos de nuestro algoritmo en cada fotograma.
- **Episodio**, es el conjunto de *steps*, generalmente desde que comienza una nueva ejecución del entorno hasta que finaliza, aunque dependiendo del problema podría haber varios episodios correspondientes a varias fases del entorno. En nuestro caso el episodio iría desde el inicio hasta la pantalla de *game over*.

Con todos estos conceptos, podemos adentrarnos en cómo realmente aprende nuestro agente, pero... ¿qué significa aprender? Podemos deducir, de lo descrito antes, que con aprender en realidad nos referimos a que el agente encuentre una estrategia que maximice un objetivo. El RL se basa en la búsqueda de esta estrategia óptima.

En literatura angloparlante se usa el término ***Policy*** para referirse a la estrategia. En nuestro videojuego, una buena estrategia podría ser disparar a través de uno de los bloques defensivos.

Para ver el proceso de aprendizaje como algo iterativo definimos:

- **Transición**, es una tupla:

[Estado Actual, Siguiente Estado, Recompensa, Acción]

Con estos cuatro términos, podemos reflejar la información suficiente del cambio de estado para que el agente pueda ser capaz de aprender. Este es el concepto "atómico"del

aprendizaje, de forma que si quitamos alguno de los elementos no sería posible la optimización de la *policy*; no obstante, en algoritmos avanzados de RL las transiciones pueden contenter otros términos que ayuden al agente con información útil para su aprendizaje. Un ejemplo en nuestro juego podría ser: ["E1","E2",+100,"disparar"]

Para conseguir recopilar que acciones son mejores con el objetivo de encontrar la *policy* adecuada, tenemos:

- **Experiencia**, es la información que el agente tiene para aprender, la forma más básica de experiencia es un conjutno de transiciones, pero puede variar dependiendo del algoritmo y almacenar otra información útil para el agente, como en Andrychowicz et al. (2018) donde proponen almacenar también metas a corto y largo plazo.

Una vez tenemos una cierta experiencia almacenada surje uno de los mayores desafíos en RL, y el cual incentiva este TFM, ¿cuándo sabemos lo suficiente del entorno, cuándo deberíamos seguir probando? Esto es, el equilibrio entre lo siguiente:

- **Exploración**, es el proceso por el que el agente va recopilando información a base de ensayo-error con el entorno. Es un proceso que habitualmente predomina en el inicio del entrenamiento, puesto que en ese punto la experiencia es excasa y aporta poca información para poder elaborar una *policy* competente. Una forma básica de controlar esto es empezar a probar aleatoriamente y con el tiempo reducir dicha aleatoriedad cada vez más. Se analizará en detalle en la sección 3.1. En nuestro caso de *Space Invaders*, podríamos hacer que juegue aleatoriamente durante 10min y luego que cada vez se base menos en la aleatoriedad y confíe más en su *policy* según avanza el tiempo.

- **Explotación**, este es el proceso por el cual el agente utiliza su conocimiento actual para tomar decisiones que maximizan las recompensas a corto plazo. Esto es fundamental para que el agente tenga un buen rendimiento y maximice la recompensa acumulada. No obstante, si para de explorar y solo explota su experiencia actual el agente puede quedar atrapado en óptimos locales y no lograr aprender la mejor estrategia a largo plazo. Cuando desplegamos nuestra solución en el entorno real, considerando que el agente ya ha aprendido lo suficiente y que por tanto su *policy* es buena, haremos que el agente use única (o principalmente) la explotación.

Un ejemplo aplicado a nuestro videojuego sobre este equilibrio sería:

Tras una primera exploración el agente aprende que si dispara y acierta suma puntos, y si esquiva los disparos no pierde vidas, si ahora dejamos que el agente únicamente explote esta experiencia llegará un momento en que no sea capaz de moverse lo suficientemente rápido para esquivar todos los disparos y no será capaz de aprender la técnica óptima (disparar a traves de los muros defensivos por ejemplo), la cual podría haber sido aprendida si tras más exploración descubre que detras de los muros es un sitio seguro y que es capaz de destruirlos por un único punto por el que luego poder disparar.

A continuación veamos los principales *pros* y *contras* de este enfoque:

2.0.1. Ventajas

1. **Aprendizaje autónomo**: Los agentes de RL aprenden de manera autónoma a través de la interacción con su entorno, sin necesidad de supervisión humana directa ni ejemplos etiquetados. Esto permite que los agentes se adapten a situaciones nuevas y cambiantes.

2. **Toma de decisiones secuenciales**: El RL es especialmente adecuado para problemas de toma de decisiones secuenciales, en los que las acciones del agente tienen consecuencias a largo plazo y afectan a las futuras recompensas. Se puede ver especialmente en Bello et al. (2017).

3. **Generalización**: Los algoritmos de RL pueden generalizar sus conocimientos a través de diferentes estados y situaciones, permitiendo a los agentes adaptarse a cambios en el entorno o a tareas ligeramente diferentes.

4. **Mejora continua**: Los agentes de RL pueden continuar aprendiendo y mejorando su rendimiento a medida que interactúan con el entorno, lo que les permite refinar sus políticas y sobre todo adaptarse a cambios en las condiciones del entorno.

2.0.2. Desventajas

1. **Muestra ineficiente**: El RL a menudo requiere un gran número de interacciones con el entorno para aprender políticas efectivas. Esto puede ser computacionalmente costoso y lento, especialmente en entornos complejos y de alta dimensión.

2. **Equilibrio exploración-explotación**: Los agentes de RL deben equilibrar la exploración de nuevas acciones y la explotación del conocimiento adquirido, lo que puede ser un desafío en problemas con un gran espacio de acciones y estados. Es en este punto donde se centra el presente trabajo.

3. **Dependencia de la función de recompensa**: La calidad del aprendizaje y el rendimiento del agente dependen en gran medida de la función de recompensa, que puede ser difícil de diseñar en algunos casos. Una función de recompensa mal diseñada puede llevar a un comportamiento subóptimo o incluso erroneo, podemos encontrar ejemplos de ello en Lehman et al. (2019).

4. **Dificultad en la transferencia de conocimiento**: Aunque los agentes de RL pueden generalizar dentro de un entorno o tarea específica, a menudo enfrentan dificultades para transferir su conocimiento a entornos o tareas significativamente diferentes, lo que limita su aplicabilidad en una amplia gama de problemas y el aprovechamiento de modelos ya entrenados para su uso en otras taras.

Se han descrito los principales conceptos del aprendizaje por refuerzo, y con ello la idea básica de este tipo de *machine learning* que podemos ver en la figura 2.2, en la cual vemos como el agente interactúa con el entorno por medio de acciones, y recibe de este un nuevo estado y una recompensa, la cual ayuda a ajustar la nueva *policy* del agente para así mejorar su toma de decisiones.

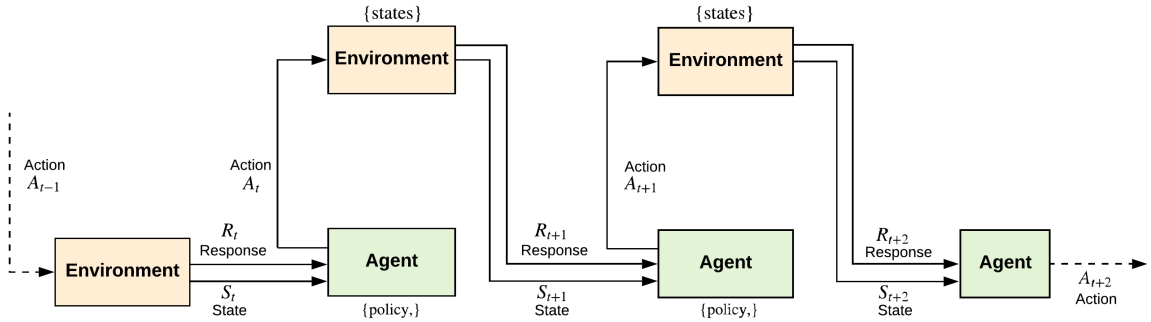


Figura 2.2: **Esquema básico de RL**

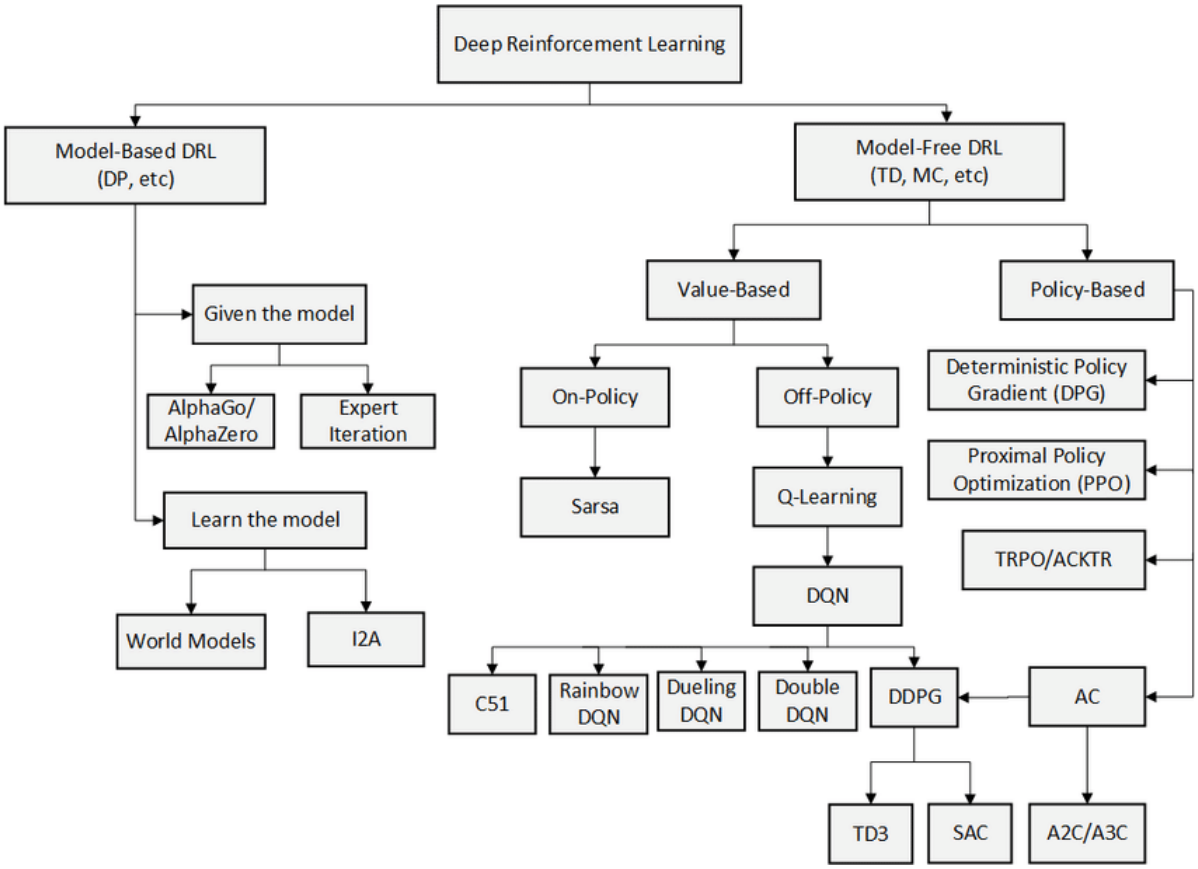


Figura 2.3: **Clasificación RL**

Para finalizar el marco teórico veamos que tipo de problemas y estrategias existen dentro del RL y en que se basan cada una de ellas. La figura 2.3 muestra una clasificación esquemática de estas estrategias, una clasificación pormenorizada puede encontrarse en AlMahamid y Grolinger (2021).

- **Basados en el Modelo**

  En estos problemas, se conoce un modelo de como se comporta el entorno en función de las acciones que el agente tome durante la ejecución. Debido a esto, se pueden combinar algoritmos de RL con otras técnicas de IA como la planificación, la ramificación y poda... Es decir, el agente no sólo trata de aprender la estrategia óptima sino que también puede predecir los resultados de sus acciones y con ello tomar decisiones sin interactuar directamente con el entorno. Se tratan de usar algoritmos de optimización para evitar usar la *fuerza bruta* a la hora de explorar y explotar el entorno.

  Dentro de este grupo tenemos a la vez dos subgrupos:

  - **Dado el modelo**

    En este enfoque, se asume que se tiene acceso a un modelo preciso del entorno. En lugar de aprender el modelo a partir de las interacciones con el entorno, en este caso se utiliza el modelo proporcionado directamente para planificar y tomar decisiones óptimas. Esta estrategia es muy efectiva en términos de muestras, ya que el agente puede aprender a tomar decisiones antes incluso de interactuar con el entorno; no obstante, obtener modelos precisos del entorno no es fácil, sobre todo en el mundo real o en entornos de alta dimensión o no estacionarios (cambios en las transiciones de estados o en las recompensas).

    El más claro ejemplo de este tipo es **AlphaGo/AlphaZero** descritos en Silver et al. (2016) y Silver et al. (2017), son agentes especializados en los juegos go y ajedrez respectivamente, que utilizan MCTS para la estimación de que acción escoger en cada estado. Estos algoritmos a menudo tienen una alta eficiencia a cambio de un coste computacional elevado.

  - **Modelo aprendido**

    En este caso, si se trata de aprender un modelo que se ajuste al entorno a partir de las interacciones del agente. Aquí no se asume un modelo preciso, si no que se busca ir adquiriendo y mejorando el modelo del entorno a través de la experiencia. A menudo involucra la estimación de las transiciones de estado, las recompensas o la probabilidad de finalización del entorno. Una vez aprendido el modelo, este es utilizado para la planificación y toma de decisiones como en el caso enterior.

    Un buen ejemplo de esto es **World Models** Ha y Schmidhuber (2018), en el cual se utilizan redes neuronales para aprender un modelo del entorno y generar secuencias de estados futuros y recompensas.

- **Independientes del modelo (*model-free*)**

  En este tipo de problemas, la simulación no conoce un modelo que pueda predecir los siguientes estados del entorno. Es el caso más típico hoy en día, ya que la mayoría de entornos reales son no deterministas o son demasiado complejos computacionalmente como para poder tratarlos como deterministas. Como no se pueden anticipar los siguientes estados, la estrategia que buscará el agente dependerá sólo del estado actual en el que se encuentre.

  Ejemplos de ello serían los videojuegos, la robótica, la conducción autónoma, etc. Dentro de este grupo existen dos clases más:

  - **Basados en Valor**

    Estos enfoques se centran en aprender una funcion de valor $Q(s, a)$ que estime el valor acumulado futuro esperado de los estados o pares estado-acción. Este tipo de algoritmos no tratan de aprender una política explícita, derivan esta *policy* implícitamente de la funcion de valor eligiendo acciones que maximicen dicha función valor en cada estado.

    Dentro de este tipo, los algoritmos pueden ser:

    - **on-policy**, es decir, las actualizaciones de la función valor de la accion $Q(s, a)$ se basan en la política que el agente está siguiendo en ese momento, en lugar de buscar directamente la política más óptima. El ejemplo de este tipo de algoritmos es **SARSA**.

    - **off-policy**, al contrario que el caso anterior, aquí los algoritmos pueden aprender una política óptima mientras siguen otra política, es decir, pueden usar datos generados por otras *policy* que se se siguieron anteriormente. En este tipo tenemos varios algoritmos que cabe mencionar:

      - **Q-learning**, aprende una función valor $Q(s, a)$, y selecciona en cada estado la acción que mayor recompensa futura aporte. Este algoritmo utiliza la ecuacion de Bellman para actualizar iterativamente la fucnion Q a través de la experiencia.

        $$Q^{\pi^*}(s, a) = r + \gamma \max_{a'} Q\left(s', a'\right)$$

      - **DQN**, sigue la idea del método anterior pero es aplicable en problemas de alta dimensionalidad, puesto que usa una red neuronal para aproximar la función Q. Se describe en Mnih et al. (2013) y es habitual en problemas con entornos visuales, por ello, a menudo las redes neuronales más utilizadas serán redes convolucionales. En problemas de este tipo el metodo de Q-learning se hace inpracticable.

- ◇ **Double DQN**, propuesto en van Hasselt et al. (2015) es una variante de DQN en la que se trata de reducir el problema del sobreajuste optimista en las estimaciones de Q por parte de la red neuronal. Para ello se utilizan dos redes diferentes para calcular y actualizar las estimaciones de Q, una de ellas es la red en uso (*eval*) y otra la red objetivo (*target*).

  Se utiliza la red en uso para seleccionar la accion de mayor valor, y la red objetivo para estimar el valor de dicha acción seleccionada, tras esto se actualiza unicamente la red en uso mediante la minimización de la función de pérdidas. La red objetivo se actualiza cada cierto tiempo a partir de los pesos de la red en uso.

- **Basados en políticas**

  Como su propio nombre indica, estos enfoques se centran en aprender y optimizar directamente una *policy* sin depender de una función valor, haciéndolo directamente mediante la maximización del retorno esperado. Este tipo de enfoques son especialmente útiles cuando las funciones de valor son difíciles de aproximar con precisión, o en entornos de acciones continuas o de alta dimensión. Dentro de este grupo destacan dos algoritmos base, a partir de los cuales se elaboran otros más avanzados que veremos en el apartado 3.3, estos son:

  - ○ **Policy Gradient**
    Cuando el agente está en un estado, tiene a su disposición un conjunto de acciones, de estas potenciaremos la elección de las que devuelvan una recompensa positiva mientras que evitaremos las que devuelvan una recompensa negativa.
    Siendo el objetivo modelar la selección de acciones, para ello, dado un estado, asignaremos a cada una una probabilidad de ser elegida, aumentando o disminuyendo según las recompensas esperadas sean mayores o menores. Este algoritmo entraría dentro de la clase "*on-policy*" en cuanto al tratamiento de la información de cada política.
    El mayor inconveniente de este algoritmo es la alta varianza en sus estimaciones del gradiente y requerirá un gran número de episodios para converger.

  - ○ **Actor-Critic**
    Descrito en 3.3.1, surje para solventar el problema de la alta varianza y convergencia lenta del método anterior. Para ello, introduce el concepto de *Value*, el cual representa el valor medio de la recompensa esperada dado el estado actual, es decir, cuán bueno es estar en el estado actual. No debemos confundir este valor $V(s)$ con la función valor del Q-learning $Q(s,a)$, puesto que el *Value* se calcula únicamente a partir del estado actual y el *Q-value* a partir del estado y una acción.

Esta función Valor se utiliza para actualizar la política de forma más eficiente, en lugar de utilizar los retornos observados (a menudo ruidosos o no representativos). En subsiguientes mejoras de este algoritmo también se utilizará para el cálculo de la Ventaja.
Este método también introduce por primera vez la aparición de dos componentes diferenciados dentro del agente:

- El *Actor*, responsable de devolver la distribución de probabilidades asociada a las acciones, de manera similar al algoritmo anterior.
- El *Critic*, que se encarga de estimar el *Value* en el estado en el que se encuentra el agente

Estado del arte

3

Esta sección representa un componente esencial, ya que permite contextualizar el proyecto en el ámbito del conocimiento actual del aprendizaje por refuerzo y ver la contribución que aporta a este campo. Para abordar de manera más efectiva esta revisión, la desglosaremos en dos puntos: el estado del arte de la exploración en el aprendizaje por refuerzo y el estado del arte de los entornos, en concreto Minigrid.

3.1. Exploración

La exploración es una parte crucial del aprendizaje por refuerzo, ya que ayuda a los agentes a descubrir nuevas estrategias y comportamientos al interactuar con su entorno Berger-Tal et al. (2014). Los algoritmos de RL emplean diversos enfoques de exploración para equilibrar la explotación de las estrategias conocidas con la exploración de acciones desconocidas que podrían generar recompensas mayores.

Actualmente disponemos de las siguientes estrategias:

- **Epsilon-Greedy**, es una estrategia simple pero muy utilizada, es un enfoque clásico que se explica y discute en el libro Sutton y Barto (1998). Consistente en seleccionar la mejor accion conocida con una probabilidad de $1 - \epsilon$ o una acción aleatoria con una probabilidad de ϵ. Si hacemos que el valor de ϵ disminuya con el tiempo, permitimos al agente explorar en las etapas iniciales y luego centrarse en la explotación cuando ya tiene más conocimientos.

- **Softmax Boltzmann Exploration**, se diferencia de la anterior en que en lugar de elegir la mejor accion con una probabilidad dada, aquí las acciones se distribuyen en una distribucion de probabilidad basada en la estimacion de valor de las acciones, por lo que las que tienen mayor estimación de valor tendrán más probabilidades de ser seleccionadas, pero todas ellas pueden ser elegidas con una probabilidad no nula. Esta lógica aparece descrita en Ackley et al. (1985)

- **Upper Confidence Bound**, UCB es una estrategia basada en la confianza en el cual las acciones se seleccionan de acuerdo con su estimación de valor ponderada por un término de confianza, que representa la incertidumbre en la estimación. Aparece por primera vez en el contexto de RL en Auer et al. (2002)

- **Thompson Sampling**, es un enfoque bayesiano que utiliza distribuciones de probabilidad sobre las estimaciones de valor de las acciones para guiar la exploración, a medida que el agente adquiere experiencia las distribuciones se actualizan y se vuelve más preciso. Su aplicación en RL se detalla en Chapelle y Li (2011).

- **Noisy Networks**, esta técnica descrita en Fortunato et al. (2019) consiste en añadir ruido a los pesos de la red neuronal, lo que permite que aumente la exploración al hacer que la política varíe de forma aleatoria.

- **Random Network Distillation**, RND se basa en la idea de proporcionar a los agentes incentivos basados en la novedad de las observaciones del entorno. Para ello se utilizan dos redes neuronales, una objetivo y otra predictora. La objetivo se inicializa al azar y se mantiene fija, mientras que la predictora se entrena para imitar las salidas de la red objetivo, tomando como entrada el estado del entorno. En áreas desconocidas del entorno, la red predictora tendrá dificultades para predecir las salidas de la red objetivo; este error es utilizado como una señal de novedad y usado como recompensa intrínseca. Esta técnica se detalla en Burda et al. (2018) y es especialmente útil en entornos con recompensas escasas o difíciles de encontrar.

- **Intrinsic Curiosity Module**, ICM es una técnica basada en la idea de recompensar a los agentes por explorar acciones que reduzcan la incertidumbre en sus predicciones sobre el entorno. Para ello se entrena un modelo predictivo (red neuronal generalmente) para estimar las consecuencias de una accion dada en un estado dado. Como en el caso anterior, si el agente realiza una accion cuya consecuencia es dificil de predecir recibirá una recompensa intrínseca mayor, consiguiendo incentivarlo para explorar áreas con mayor incertidumbre y descubriendo que consecuencias tienen sus acciones. Se describe en Pathak et al. (2017).

- **Rewarding Impact-Driven Exploration**, RIDE, descrita en Raileanu y Rocktäschel (2020), la idea clave detrás de esta técnica es proporcionar incentivos intrínsecos basados en el impacto de las acciones de un agente sobre su entorno. Al premiar a los agentes por realizar acciones que causen cambios notables en el entorno, RIDE fomenta la exploración de acciones útiles y significativas (tales como apertura de puertas, manipulación de objetos...), en lugar de simplemente recompensar a los agentes por visitar estados desconocidos. Para ello se entrena un modelo predictivo para estimar el impacto de una acción dada en un estado dado.

- **Adversarially Guided Actor-Critic**, AGAC introduce un tercer protagonista al algoritmo Actor-Critic: el Adversario. Su papel es predecir las acciones del Actor, minimizando la diferencia entre su distribución de acciones y la inducida por la política. Por su parte ahora el Actor, además de encontrar las acciones con mayor recompensa, ha de tratar de ser impredecible por el Adversario (maximizando la discrepancia de sus acciones), lo cual fomenta la innovación y exploración. Se detalla en Flet-Berliac et al. (2021).

- **NovelD**, (Novelty Difference) se presenta recientemente, en el artículo Zhang et al. (2021), esta técnica simple consiste en ver cuan parecidos son dos estados para un agente; se dan recompensas intrínsecas si dada una trayectoria, el estado anterior ha sido suficientemente explorado y el siguiente no. Para determinar esta similitud entre estados, los autores usan una RND como la vista anteriormente. Esta técnica es la más cercana a la que se analizará en este TFM (E3B) en cuanto a la lógica de recompensar al agente cuando encuentre estados que difieran entre ellos.

Por lo general, las estrategias anteriores funcionan de forma más efectiva cuanto más modernas son, puesto que van mejorando el estado del arte y unas surjen para solucionar los problemas de las anteriores, no obstante, dependiendo del entorno y objetivo pueden ser más eficaces y rápidas estrategias más antiguas y simples Ladosz et al. (2022).

3.2. Entornos, Minigrid

Los entornos en los que aprenden y se prueban los agentes de aprendizaje por refuerzo van a menudo de la mano de las innovaciones en este campo Zheng (2022). Por ejemplo, el estado del arte en un punto del tiempo es tal que es capaz de ejecutar todas las pruebas de un entorno con buen resutado, esto motiva a desarrolladores a crear entornos más complejos en los que probar estos modelos. Cuando se prueba que en estos entornos más complicados las técnicas actuales no dan el rendimiento esperado, los investigadores de RL tratan de ver como mejorar los métodos o crear nuevos algoritmos que si consigan buenos resultados en el nuevo entorno, creando la *pescadilla que se muerde la cola*.

Los entornos actuales en los que se suelen probar los modelos son los siguientes:

- **Gym (OpenAI)**, Brockman et al. (2016), proporciona una amplia variedad de entornos de prueba, algunos clásicos de control y robótica y otros como los juegos de Atari. Hay que destacar el videojuegojuego *Montezuma's Revenge* por suponer un hito dificil de alcanzar frente al resto de juegos de la Atari, finalmente la propia OpenAI consiguió un buen resultado adaptando el algoritmo PPO.

- **DeepMind Lab**, Beattie et al. (2016) es un entorno 3D diseñado para imitar tareas complejas en entornos realistas.

- **Project Malmo**, Johnson et al. (2016), un entorno de Microsoft basado en el videojuego Minecraft en el que se ofrecen distintas tareas y desafíos para los agentes.

- **Pygame Learning Environment (PLE)**, una librería que permite adaptar juegos creados con Pygame a entornos para el RL, incluye ejemplos clásicos como Pong o Tetris.

- **VizDoom**, entorno 3D basado en el videojuego Doom, en el que el agente ha de moverse y disparar para conseguir objetivos, está detallado en Kempka et al. (2016)

- **Unity ML-Agents**, Juliani et al. (2020), un proyecto que permite usar el motor Unity como plataforma para desarollar entornos 2D o 3D personalizados.

Además, y con especial relevancia para este trabajo, están:

- **MiniHack**, Samvelyan et al. (2021), es una versión simplificada del videojuego de rol estilo *roge-like* NetHack. Es un juego de mazmorras en el que los jugadores deben explorar niveles generados proceduralmente, llenos de monstruos, objetos y misiones 3.1 . Dada su naturaleza altamente compleja y el enorme espacio de acciones disponibles, NetHack es un desafío interesante para el RL. MiniHack conserva la base de NetHack pero se ha rediseñado para ofrecer entornos más simples y tareas más específicas que los agentes consigan resolver, ayudando a los investigadoras a evaluar los algoritmos en aspectos específicos del aprendizaje.

  En Henaff et al. (2023), MiniHack es usado como entorno principal para probar E3B, consiguiendo mejores resultados que únicamente usando el algoritmo base de RL (IMPALA en su caso), tanto en tareas basadas en el movimiento como en tareas basadas en acciones.

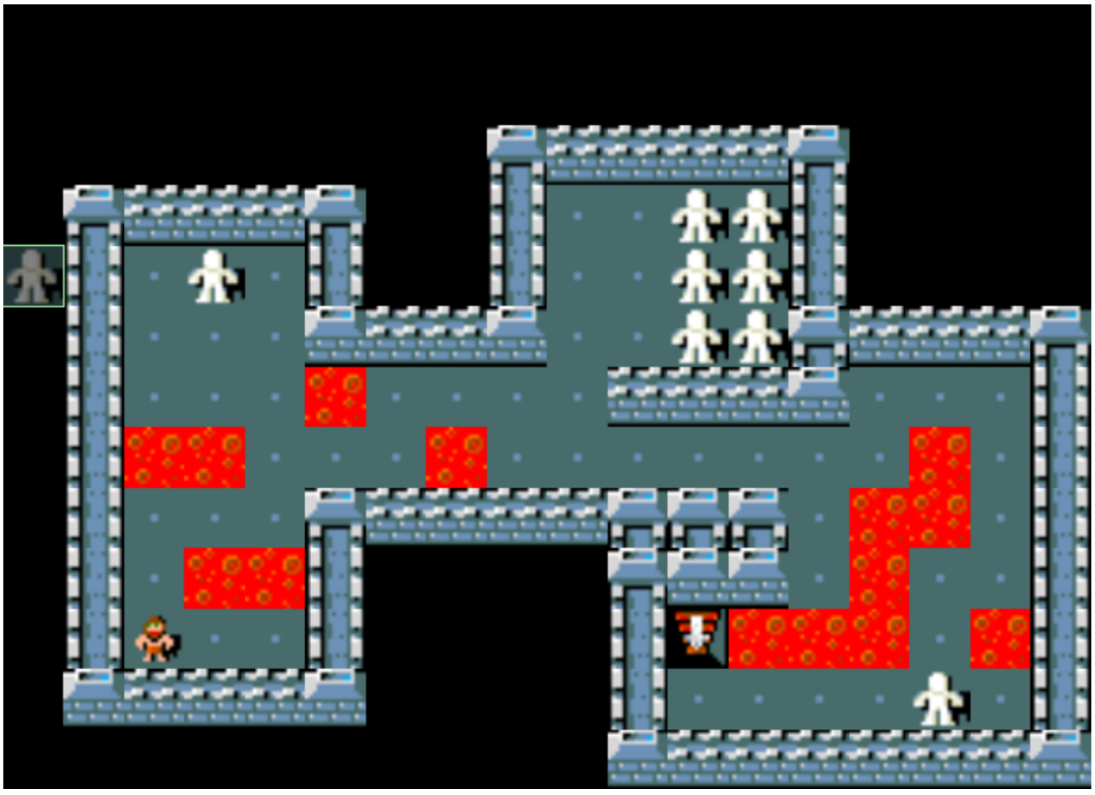


Figura 3.1: **Ejemplo de mapa en MiniHack**

- **MiniGrid**, Chevalier-Boisvert et al. (2018), es un entorno basado en una cuadícula 2D simplificada y ligera, diseñado específicamente para la investigación en RL. Proporciona al investigador un entorno de experimentación capaz de evaluar algoritmos de apren-

dizaje por refuerzo sin la carga computacional asociada a entornos más complejos y realistas.

En este entorno, el agente ha de moverse por una habitación o conjunto de ellas separadas por puertas y que contienen objetos como bloques de colores y llaves con las que el agente puede interactuar. Los niveles o desafíos están diseñados para la recogida de objetos, la navegación en el entorno, la apartura de puertas o la realización de tareas secuenciales.

Cabe mencionar, que el actual estado del arte para MiniGrid se detalla en Andres et al. (2022), en el cual usan un método para clasificar los episodios llamado RAPID (propuesto en Zha et al. (2021)) junto con las recompensas intrinsecas. En este punto, este TFM buscará comprobar si la aplicación de E3B a este entorno supone un nuevo estado del arte en el mismo o por el contrario ofrece peores resultados que los actuales.

Los dos entornos/desafíos clásicos para probar los algoritmos de RL en MiniGrid, y que serán descritos en 4.2 y usados en este trabajo, son los siguientes:

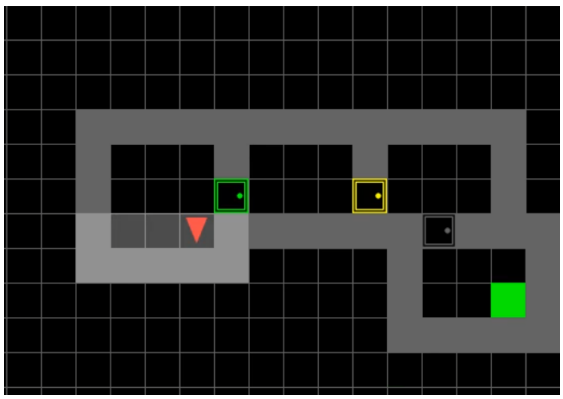


Figura 3.2: **Mapa MultiRoom en MiniGrid**

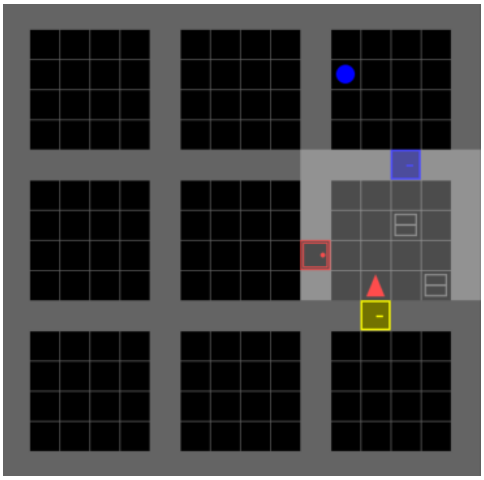


Figura 3.3: **Mapa Obstructed Maze 2Dlh en MiniGrid**

3.3. Algoritmos

Como comentamos en la introducción, los principales algoritmos que vamos a ver y analizar en este trabajo son tres: A2C, PPO y E3B. Los dos primeros son algoritmos de RL basados en Policy Optimization, mientras que el tercero es la nueva técnica de exploración cuya implementación, adaptación y análisis es la base de este TFM.

3.3.1. A2C

La familia de algoritmos Actor Critic es un enfoque de aprendizaje por refuerzo que aparece por primera vez en Witten (1977) en el que el modelo de agente se divide en dos componentes, el Actor y el Critic. El Actor se encarga de decidir qué acciones tomar en función de los estados, mientras que el Critic evalúa la calidad de esas acciones proporcionando una estimación del valor (*Value*) esperado de los estados. La finalidad es utilizar la información que porporciona el Critic para mejorar la política del Actor. Por lo demás, el algoritmo sigue una lógica similar a la mencionada en Policy Gradient.

Respecto a la función de coste, ésta estará repartida también en la parte del Actor y del Critic:

$$\nabla_\theta \log \pi_\theta(s,a)\left(R - V_\varphi(s)\right), \quad \left(R - V_\varphi(s)\right)^2$$

La fórmula para el Actor es similar a la que se usa en Policy Gradient, con la diferencia de que en vez de la recompensa usamos el *Value* estimado. Para el Critic, estimamos el error comentido entre Valor recolectado y estimado.

Por su parte, el Advantage Learning es una técnica que busca estimar la vantaja de una acción en un estado dado. La ventaja es una medida que cuantifica cuánto mejor es una acción en comparación con el valor promedio de todas las acciones posibles en ese estado.

$$A^\pi\left(s_t, a_t\right) := Q^\pi\left(s_t, a_t\right) - V^\pi\left(s_t\right)$$

Esta, es una medida más precisa para decidir si el agente va por buen camino, o no, dada la estrategia actual. Las funciones de relevancia que mas se utilizan en RL son en realidad funciones derivadas de la Ventaja anterior. Una de las más conocidas es GAE (Generalized Advantage Estimation) presentada en Schulman et al. (2018):

$$\begin{aligned}
\mathrm{GAE}(\gamma,0): \quad & \hat{A}_t := \delta_t \quad = r_t + \gamma V\left(s_{t+1}\right) - V\left(s_t\right) \\
\mathrm{GAE}(\gamma,1): \quad & \hat{A}_t := \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} = \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V\left(s_t\right)
\end{aligned}$$

El algoritmo Advanced Actor Critic (A2C) es una nueva version del Actor Critic que utiliza el enfoque de Advantage Learning y está adaptado al multiproceso, en el que una batería de agentes se ejecutan de forma paralela durante el entrenamiento.

En el caso de A2C (a diferencia de A3C) la comunicación entre estos agentes se produce de

forma síncrona, de modo que el modelo general es actualizado a la vez en función de todos los modelos perticulares y es cargado al mismo tiempo en cada agente. Sin embargo en A3C cada instancia puede modificar o cargar el modelo general en cualquier momento como podemos ver en 3.4.

A2C aparece en la literatura como una adaptación de A3C Mnih et al. (2016) para hacer las actualizaciones de form síncrona, siguiendo el pseudocódigo 1

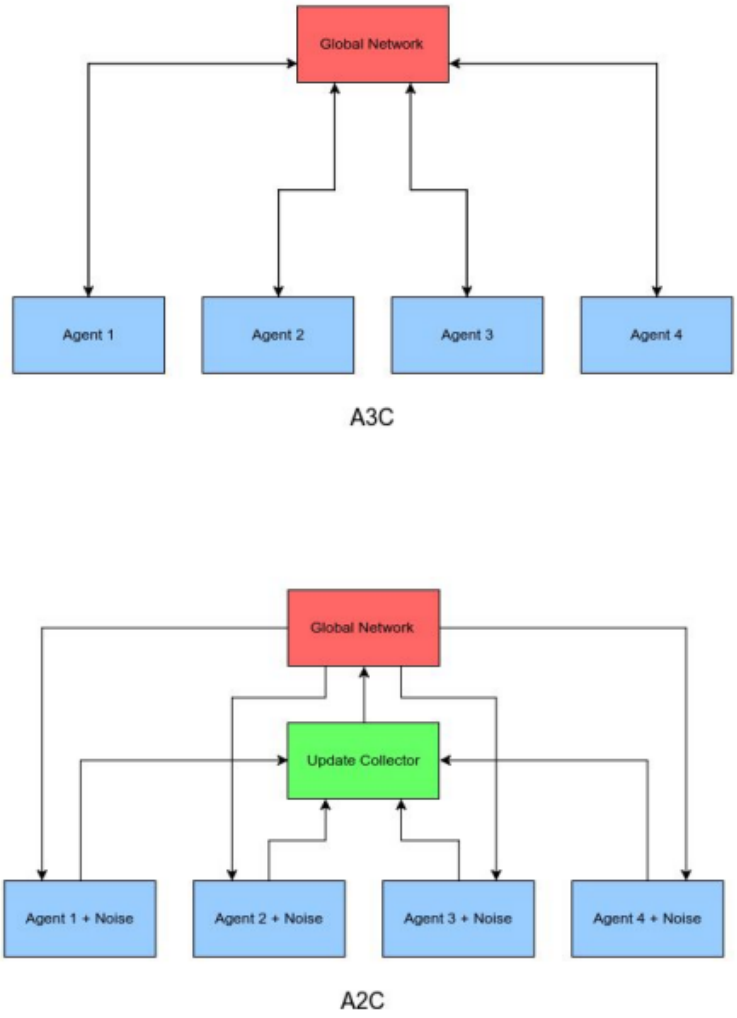


Figura 3.4: **A2C vs A3C**

Este algoritmo tiene la posibilidad de modelar espacios de acciones continuos y entornos estocásticos. También se caracteríza por una mayor diversidad y volumen de datos disponibles para el proceso de entrenamiento. Dado que es un entenemiento multi-proceso, el tiempo del mismo se reduce significativamente.

Algoritmo 1: Algoritmo *Parallel Advantage Actor-Critic* (A2C)

1. Iniciamos el contador tiempo $N = 0$ y los pesos de la red θ, θ_v
2. Instanciamos un conjunto e de n_e entornos
3. **repeat**
   - *a*) **for** $t = 1$ to $t_{\text{máx}}$ **do**
     1) Muestreamos $\boldsymbol{a}_t$ de $\pi\left(\boldsymbol{a}_t \mid \boldsymbol{s}_t; \theta\right)$
     2) Calculamos $\boldsymbol{v}_t$ de $V\left(\boldsymbol{s}_t; \theta_v\right)$
     3) **parallel for** $i = 1$ to n_e**do**
        - *a′* Ejecutar acción $a_{t,i}$ en el entorno e_i
        - *b′* Observar el nuevo estado $s_{t+1,i}$ y recompensa $r_{t+1,i}$
     4) **end parallel for**
   - *b*) **end for**
   - *c*) $\boldsymbol{R}_{t_{\text{máx}}+1} = \begin{cases} 0 & \text{para } \boldsymbol{s}_t \text{ terminales} \\ V\left(s_{t_{\text{máx}}+1}; \theta\right) & \text{para } \boldsymbol{s}_t \text{no terminales} \end{cases}$
   - *d*) **for** $t = t_{\text{máx}}$ down to 1 **do**
     1) $\boldsymbol{R}_t = \boldsymbol{r}_t + \gamma \boldsymbol{R}_{t+1}$
   - *e*) **end for**
   - *f*) $d\theta = \frac{1}{n_e \cdot t_{\text{máx}}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{\text{máx}}} \left(R_{t,i} - v_{t,i}\right) \nabla_\theta \log \pi\left(a_{t,i} \mid s_{t,i}; \theta\right) + \beta \nabla_\theta H\left(\pi\left(s_{e,t}; \theta\right)\right)$
   - *g*) $d\theta_v = \frac{1}{n_e \cdot t_{\text{máx}}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{\text{máx}}} \nabla_{\theta_v} \left(R_{t,i} - V\left(s_{t,i}; \theta_v\right)\right)^2$
   - *h*) Actualizar θ con $d\theta$ y θ_v con $d\theta_v$
   - *i*) $N \leftarrow N + n_e \cdot t_{\text{máx}}$
4. **until** $N \geq N_{\text{máx}}$

3.3.2. PPO

El Proximal Policy Optimization (PPO) es un algoritmo de aprendizaje por refuerzo *on-policy* que busca mejorar la política de un agente mediante actualizaciones de política más estables y eficientes en comparación con otros métodos de optimización de políticas. PPO es de tipo Actor-Crítico y utiliza funciones de ventaja para guiar las actualizaciones de la política. Puede ser usado en entornos de espacios de acciones continuos o discretos.

PPO fué propuesto en Schulman et al. (2017) como un avance sobre TRPO Schulman et al. (2015), con la finalidad de encontrar la mayor mejora posible en la política sin llegar a causar un deterioro del rendimiento. En el caso de TRPO, este problema se trataba de solventar mediante un método complejo de segundo orden, mientras que PPO consigue al menos igualar el rendimiento de TRPO utilizando métodos de primer orden junto con otras técnicas para mantener las nueva políticas cercanas a las anteriores.

Exiten dos variantes principales de PPO:

- **PPO-Penalty** resuelve de forma aproximada una actualización con restricciones KL como TRPO, pero penaliza la divergencia KL en la función objetivo en lugar de convertirla en una restricción, y ajusta automáticamente el coeficiente de penalización a lo largo del entrenamiento para que se escale adecuadamente.

- **PPO-Clip** no tiene un término de divergencia KL en el objetivo y no tiene ninguna restricción. En su lugar, se basa en un recorte especializado en la función objetivo para eliminar los incentivos para que la nueva política se aleje de la antigua. Este trabajo se centrará en esta variante de PPO.

PPO-clip, tal como aparece descrito en Schulman et al. (2017) actualiza su política mediante

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L\left(s, a, \theta_k, \theta\right) \right],$$

donde L se obtiene como

$$L\left(s, a, \theta_k, \theta\right) = \min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} A^{\pi_{\theta_k}}(s,a), \operatorname{clip}\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)}, 1-\epsilon, 1+\epsilon \right) A^{\pi_{\theta_k}}(s,a) \right)$$

en esta ecuación ϵ es el hiperparámetro que mide cuan distinta puede ser la nueva política respecto a la política anterior. No obstante, utilizaremos una simplificación de la misma en nuestra implementación 3, en ella:

$$L\left(s, a, \theta_k, \theta\right) = \min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} A^{\pi_{\theta_k}}(s,a), g\left(\epsilon, A^{\pi_{\theta_k}}(s,a)\right) \right),$$

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

Visualicemos con 3.5 como actua esta fórmula dados un par estado-acción (s, a).

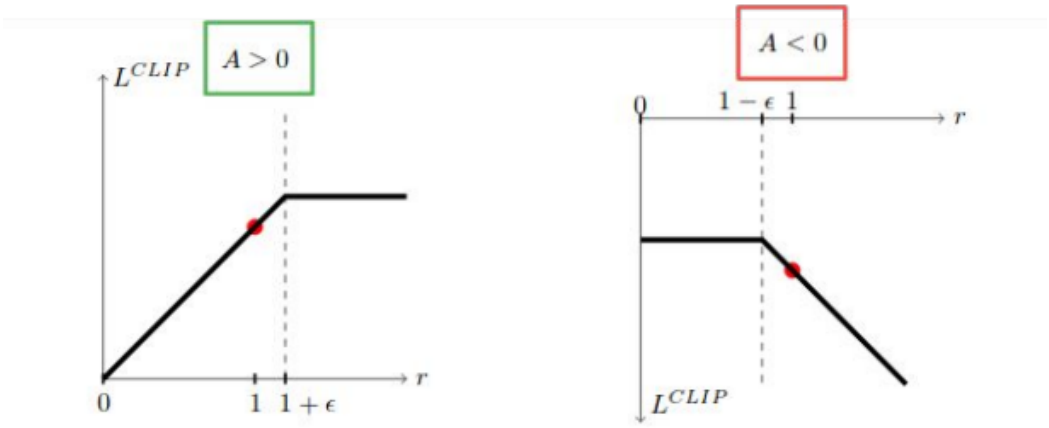


Figura 3.5: **PPO-Clip segun acción**

- **Acción positiva** $(A > 0)$, entonces se incrementará el objetivo en caso de que la acción sea más probable, hasta alcanzar el tope de $1 + \epsilon$, con el objetivo de no incrementar demasiado hasta el punto de empeorar la política. En caso de que la acción sea menos probable nos permite "deshacer" ese paso en la dirección equivocada todo lo que queramos.

- **Acción negativa** $(A < 0)$, en este caso si la accion es menos probable se alcanzará el límite de $1 - \epsilon$ y no nos permitirá actualizar la política más allá de esto para mantenerla controlada. En caso contrario en el que la acción es más probable, podremos corregir el error todo lo que queramos.

Llegamos a lo conclusión que mediante la técnica de *Clip* eliminamos los incentivos que tenía la política de cambiar bruscamente, por lo que nos sirve para estabilizarla.

El pseudocódigo de este algoritmo se muestra en 2

Algoritmo 2: Algoritmo *Proximal Policy Optimization* (PPO-Clip)

1. **Input:** parámetros iniciales de la política θ_0, parámetros iniciales de la funcion-valor ϕ_0
2. **for** $k = 0, 1, 2, \ldots$ **do**

   a) Tomar conjunto de trayectorias $\mathcal{D}_k = \{\tau_i\}$ ejecutando la política $\pi_k = \pi(\theta_k)$ en el entorno.

   b) Calcular recompensas $\hat{R}_t$.

   c) Calcular una estimación de la ventaja, $\hat{A}_t$ (usando cualquier metodo de estimación de ventaja como GAE) basado en la funcion-valor actual V_{ϕ_k}.

   d) Actualizar la política maximizando el objetivo PPO-Clip:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_k}(a_t \mid s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g\left(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)\right) \right),$$

   normalmente usando el ascenso del gradiante estocástico con ADAM.

   e) Ajustar la funcion-valor mediante la regresión del error cuadrático medio:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

   normalmente usando algún algoritmo de descenso del gradiente.

3. **end for**

3.3.3. E3B

Este algoritmo pruepuesto en Henaff et al. (2023), busca una mejor formar de explorar las CMDP(Procesos de Decision de Markov Contextuales) Feinberg (2000) y con ello mejorar la problemática de la exploración-explotación en el RL permitiendo que los agentes generalicen mejor su comportamiento entre entornos similares pero con cambios. Consiste en una generalización de las recompensas episódicas *count-based* a entornos de espacios continuos.

El *count-based episodic bonus* Savinov et al. (2019), es un enfoque de exploración que utiliza información de conteo de visitas a estados o pares estado-acción durante un episodio para incentivar al agente a explorar áreas poco visitadas del espacio de estados-acciones. La idea consiste en agregar una recompensa basada en la rareza del estado o estado-acción. A menudo se calcula mediante funciones cuyo valor disminuye a medida que aumentan las visitas a ese estado o par. El principal problema de este enfoque es su aplicación en entornos con espacios de estado continuos o de alta dimensión, ya que contar de forma precisa las visitas a cada estado-acción puede ser complicado y costoso computacionalmente.

Es en este punto donde los autores de E3B proponen la solución mediante la reducción de estados a espacios latentes y su similitud mediante curvas elípticas.

En 3.6 tenemos un resumen de como funciona el método E3B. Se define una recompensa intrínseca basada en la posición de la respresentación del estado actual respecto a la elipse que forman las representaciones de otros estados del mismo episodio. Esta recompensa se combina con las recompensas del entorno para actualizar la política del agente.

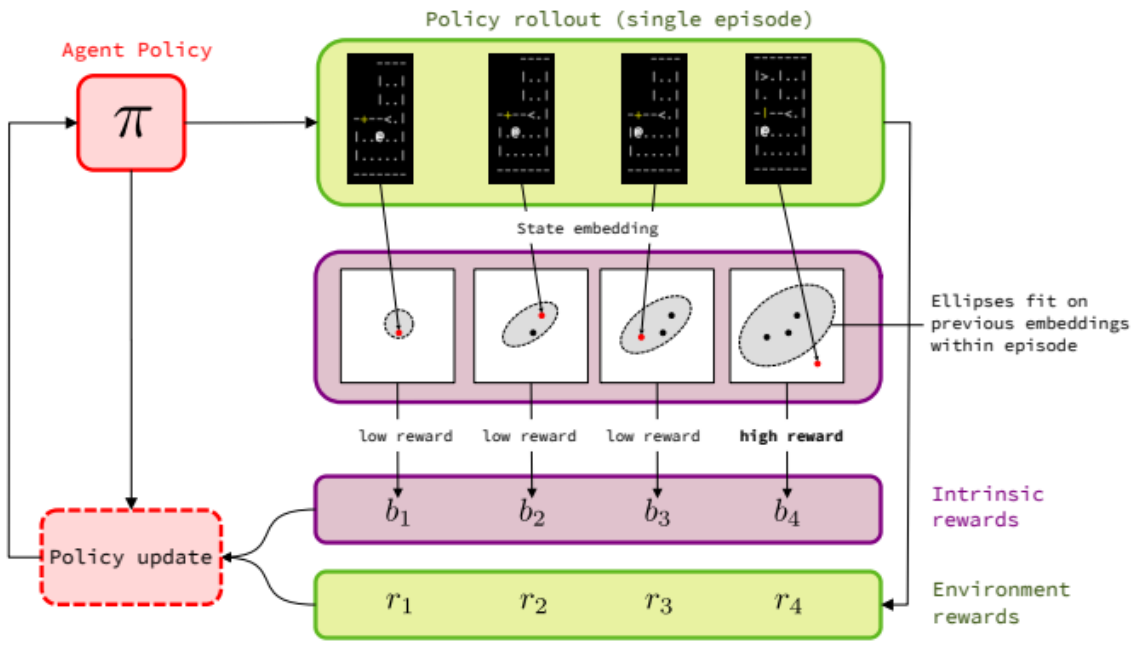


Figura 3.6: **Esquema de E3B**

Veamos ahora los procesos de representación de los estados y de recompensas basados en elipses:

- **El proceso de representacion (*Learned Feature Encoder*)**, consiste en reducir la dimensionalidad del estado actual para poder descartar información del entorno que no está siendo util para predecir las acciones adecuadas para el agente. En Petrenko et al. (2021) se muestra como esta técnica hace al RL más robusto al ruido y a características del entorno irrelevantes par el agente.

  Para ello podría usarse cualquier método de extracción de características, como por ejemplo, tal como se propuso en el paper original, el modelo dinámico inverso, el cual entrena a un modelo g junto con las representaciones ϕ para mapear cada par de representaciones $\phi(s_t), \phi(s_{t+1})$ en una distribución sobre las acciones a_t. En este caso, ϕ está separada de la red de la política. El modelo g será entrenado junto con ϕ usando la siguiente función de perdida:

$$\ell\left(s_t, a_t, s_{t+1}; \phi, g\right) = -\log p\left(a_t \mid g\left(\phi\left(s_t\right), \phi\left(s_{t+1}\right)\right)\right)$$

- **Recompensas episódicas elípticas (*Elliptical Episodic Bonus*)**. Dadas las representaciones anteriores ϕ para cada momento del tiempo t, definimos la recompensa b *bonus* como:

$$b\left(s_t\right) = \phi\left(s_t\right)^\top C_{t-1}^{-1} \phi\left(s_t\right), \quad C_{t-1} = \sum_{i=1}^{t-1} \phi\left(s_i\right) \phi\left(s_i\right)^\top + \lambda I$$

  En donde C es la matriz de covarianza, λ es un escalar y I la matriz identidad, para conseguir que C_{t-1} sea no-singular. La lógica detrás de esto es que la recompensa intrínseca (episódica elíptica) sea mayor cuanto más lejos de le elipse formada por las representaciones de estados anteriores esté la representación del estado actual; de esto se puede deducir que estas recompensas se irán haciendo más pequeñas segun aumente el numero de observaciones.

  Tras esto, la recompensa final se obtiene de combinar la recompensa intrínseca y extrínseca (del entorno) siguiendo:

$$\bar{r}\left(s_t, a_t\right) = r\left(s_t, a_t\right) + \beta \cdot b\left(s_t\right)$$

  en donde β es un hiperparámetro que nos sirve para balancear el compromiso entre exploración y explotación.

En el cálculo de la recompensa intrínseca, se necesita obtener la inversa de la matriz C_{t-1} en cada instante t, lo cual conlleva un coste computacional muy alto para espacios latentes de grandes dimensiones. Una mejora que se utiliza para este cálculo es la siguiente matriz identidad detallada en Sherman y Morrison (1950):

$$C_t^{-1} = \begin{cases} \frac{1}{\lambda} I & \text{if } t = 0 \\ C_{t-1}^{-1} - \frac{C_{t-1}^{-1}\phi(s_t)\phi(s_t)^\top C_{t-1}^{-1\top}}{1+\phi(s_t)^\top C_{t-1}^{-1}\phi(s_t)} & \text{if } t \geq 1 \end{cases}$$

Lo cual nos da un rendimiento unas 3 veces más rápido que invirtiendo la matriz directamente.

En 3.7 se muestra un ejemplo simple que ilustra el cálculo de la recompensa elíptica para un espacio de representación bidimensional.

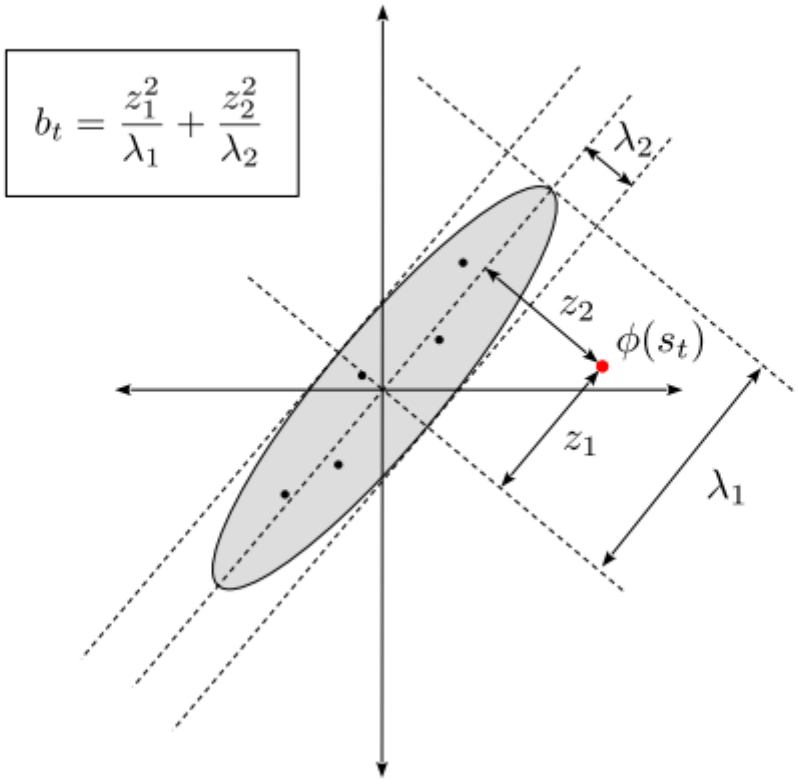


Figura 3.7: **Recompensas elípticas en 2D**

En el algoritmo 3 podemos ver el pseudocódigo de este método.

Algoritmo 3: Algoritmo *Exploration via Episodic Elliptical Bonuses* (E3B)

1. Inicializar politica π, codificador de características ϕ y modelo dinámico inverso f.
2. **while** no converge **do**
   a) Muestrear el contexto $c \sim \mu_C$ y el estado inicial $s_0 \sim \mu_S(\cdot \mid c)$
   b) Inicializar la matriz de covarianza inversa: $C_0^{-1} = \frac{1}{\lambda} I$
   c) **for** $t = 0, \ldots, T$ **do**
      1) $a_t \sim \pi(\cdot \mid s_t)$ // Muestrear una acción
      2) $s_{t+1}, r_{t+1} \sim P(\cdot \mid s_t, a_t)$ // Paso por el entorno
      3) $b_{t+1} = \phi(s_{t+1})^\top C_t^{-1} \phi(s_{t+1})$ // Calcular bonus
      4) $u = C_t^{-1} \phi(s_{t+1})$
      5) $C_{t+1}^{-1} = C_t^{-1} - \frac{1}{1+b_{t+1}} u u^\top$ // Actualizar la matriz de covarianza inversa
      6) $\bar{r}_{t+1} = r_{t+1} + \beta b_{t+1}$
   d) **end for**
   e) Actualizar el gradiente de la politica π usando las recompensas $\bar{r}_1, \ldots, \bar{r}_T$.
   f) Actualizar ϕ y g usando $\{(s_t, a_t, s_{t+1})\}_{t=0}^{T-1}$ para minimizar la pérdida:

$$\ell = -\log\left(p\left(a_t \mid f\left(\phi\left(s_t\right), \phi\left(s_{t+1}\right)\right)\right)\right)$$

end while

Metodología

4

4.1. Propuesta

Para abordar los objetivos previamente descritos, ejecutaremos las diferentes soluciones propuestas en una serie de entornos idénticos, con el fin de comparar la eficacia de cada uno a la hora de resolver las distintas tareas. Para ello será necesario:

- **Programar Agentes**, tendremos que conseguir agentes de cada una de las soluciones que vamos a probar, en concreto, se utilizarán:
  - **A2C y PPO**, en sus versiones *vanilla*, las obtenemos del repositorio rl-starter-files y sobre los cuales haremos cambios mínimos para adaptarlo a nuestro sistema y a librerías actualizadas.
  - **E3B**, su implementación, de forma adaptada tal como se explicará en 4.3, será una de las tareas principales de la parte prácitca de este TFM.
  - **RIDE**, obtendremos este algoritmo del repositorio NovelD, y simplemente lo ejecutaremos en los entornos seleccionados para ver su rendimiento.
  - **NovelD**, como en el caso anterior, será usado únicamente como otro punto que referencia el estado del arte para tener una mejor comparativa, se ejecutará tal como está publicado en impact-diven-exploration, realizando únicamente adaptaciones para nuestro sistema.
- **Seleccionar los entornos**, se detallará en la sección 4.2, se buscarán tanto entornos sencillos como complejos con el fin de ver las capacidades y limitaciones de los algoritmos a analizar, todos ellos del entorno Minigrid disponibles en Minigrid.
- **Entrenar a los Agentes**, se utilizará un ordenador doméstico, sin procesador gráfico ni grandes capacidades computacionales, es por ello que se ha elegido una adaptación simplificada de E3B. El entorno Minigrid es lo suficientemente liviano como para poder ser ejecutado en estas condiciones.
- **Visualizar los resultados**, para ello se utilizarán las métricas obtenidas del entrenamiento de los agentes, las cuales serán procesadas y graficadas.

4.2. Entorno experimental

El entorno serán, como comentábamos, distintos mapas de Minigrid. Por una parte se han seleccionado mapas simples, para ver que los algoritmos base A2C y PPO son capaces de aprender una política sólida en ellos; y por otra parte mapas más complejos, donde se verá que los algoritmos *vanilla* no consiguen buenos resultados. La dificultad de los mapas viene detallada en Chevalier-Boisvert et al. (2019), donde se presenta el entorno y se describen las tareas principales del agente.

En concreto los mapas seleccionados han sido:

- **MiniGrid-MultiRoom-N2-S4-v0** 4.2, está formado por 2 habitaciones conectadas por una puerta, el objetivo es llegar a la casilla verde en la otra habitación.

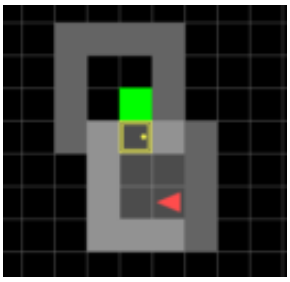


Figura 4.1: **Mapa MiniGrid-MultiRoom-N2-S4-v0**

- **MiniGrid-DoorKey-5x5-v0** 4.2, este mapa está conformado por dos habitaciones pequeñas, en la primera (en la que aparece el agente) hay una llave, la cual debe coger y usar en la puerta para abirla y llegar a la otra habitación donde se encuentra una casilla verde que es el objetivo que le generará una recompensa.

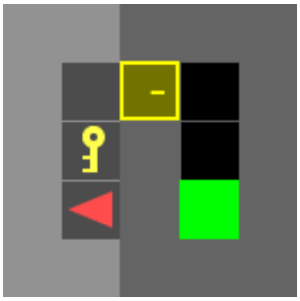


Figura 4.2: **Mapa MiniGrid-DoorKey-5x5-v0**

- **MiniGrid-MultiRoom-N4-S5-v0** 3.2, es un mapa de 4 habitaciones conectadas por puertas, el objetivo es alcanzar la casilla verde situada en la última habitación, aunque similar, es mucho más complejo que su verión con 2 habitaciones.

- **MiniGrid-MultiRoom-N6-v0** 4.3, como el caso anterior pero con 6 habitaciones, es uno de lo mapas más complejos de minigrid para algoritmos de RL *vanilla*, puesto que es muy complicado llegar a la única recompensa.

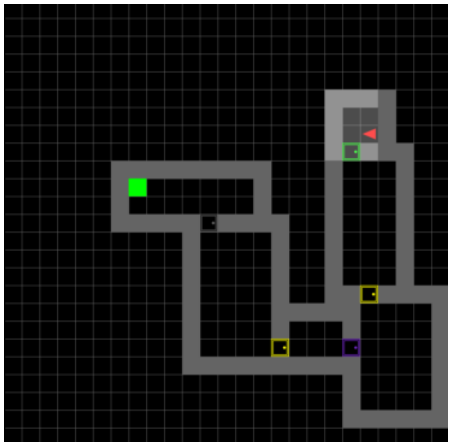


Figura 4.3: **Mapa MiniGrid-MultiRoom-N6-v0**

- **MiniGrid-ObstructedMaze-2Dlh-v0** 3.3, un mapa cuyo objetivo es que el agente consiga la pelota azul, la cual estará situada en una esquina del grupo 3x3 de habitaciones conectadas, las puertas están cerradas y las llaves están escondidas en cajas, pero el problema puede reducirse a 4 habitaciones, puesto que las otras 5 son inaccesibles.

  Es una simplificación de *MiniGrid-ObstructedMaze-Full-v0* 4.4, en el cual entran en juego las 9 habitaciones y además las puertas pueden estar bloqueadas por pelotas que es necesario mover.

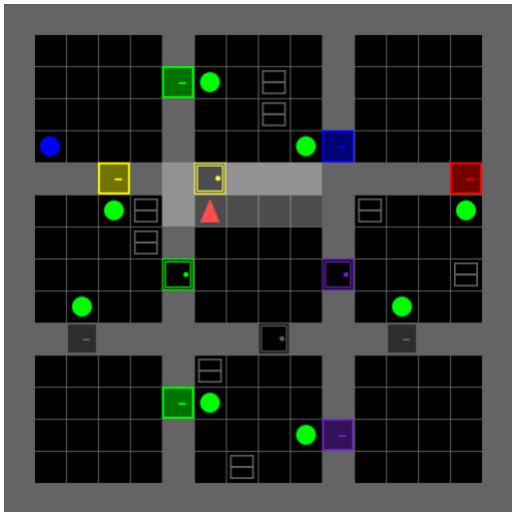


Figura 4.4: **Mapa MiniGrid-ObstructedMaze-Full-v0**

4.3. Adaptación propuesta

Una vez analizado el funcionamiento y la lógica detrás del algoritmo E3B 3.3.3, y con el fin de poder implementarlo de una forma sencilla sobre el algoritmo de RL base PPO, se han propuesto cambios y simplificaciones en el mismo. Otra de las razones que motivó el planteamiento de esta simplificación surjió durante la etapa de testeo del algoritmo E3B original sobre NetHack, puesto que en el sistema del que se disponía para la realización de este TFM su rendimiento era extremadamente lento.

A continuación se detallan las principales características de la simplificación:

- **Espacio latente bidimensional**. En el algoritmo original, tal como detallamos en 3.3.3, el proceso de representación es llevado a cabo por un modelo dinámico inverso. Nosotros obtaremos por el uso de un encoder sobre la *"imagen"* que devuelve el entorno, para pasar los estados a su representación bidimensional en el plano y así facilitar el procesamiento del espacio latente y el cálculo de las recompensas.

  Para ello, en un proceso previo al propio entrenamiento del agente con RL, entrenamos a un encoder mediante un enfoque de aprendizaje no supervisado con un modelo autoencoder Hinton y Sejnowski (1986), este entrenamiento tiene un enfoque *"offline"*, a diferencia de tal como fué propuesto E3B que el modelo dinámico inverso se entrena en cada paso del entorno junto con el agente.

  En primer lugar recopilamos 100 estados de 10 variaciones distintas de cada mapa, obteniendo 5000 estados aleatorios de entornos Minigrid. Tras ello, alimentamos a nuestro autoencoder con estas imágenes y lo entrenamos para que sea capaz de regenerarlas de la forma más fiel posible, siguiendo una función de pérdida de reconstrución (en este caso, el error cuadrático medio).

  El autoencoder tiene la siguiente estructura 4.5:

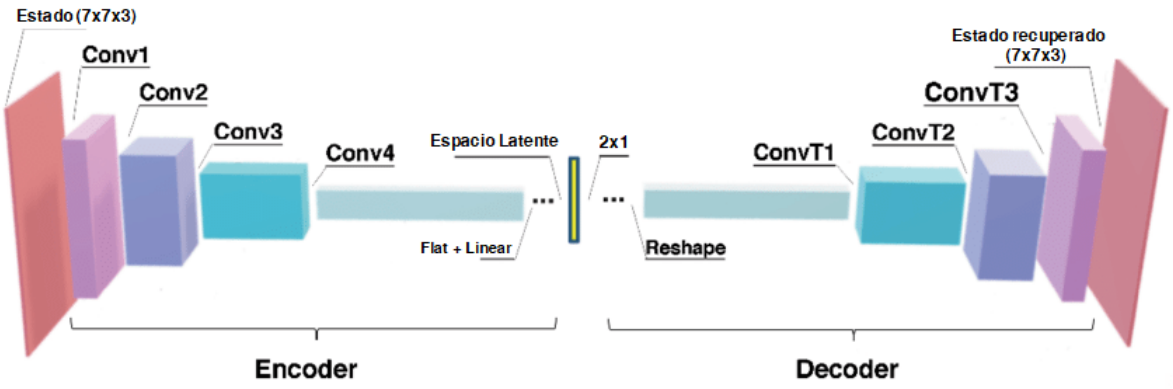


Figura 4.5: **Esquema Autoencoder**

Donde podemos ver que toma imágenes de dimensiones 7x7x3, que son las dimensiones de salida de las imágenes en las observaciones en Minigrid, y va reduciendo su dimensionalidad hasta obtener un vector 2x1, el cual codifica las coordenadas en el espacio

bidimensional. En la parte del decoder hace el paso inverso, llevando estos vectores a las dimensiones de la imagen original.

Se han elegido redes convolucionales Lecun et al. (1998) para crear el autoencoder Masci et al. (2011) puesto que estamos tratando con imágenes, aunque el atributo *"imagen"* que nos devuelve Minigrid en cada observación no es directamente lo que veríamos por pantalla, si no que es una representación de ello, donde cada valor del 7x7x3 es un entero comprendido entre 0 y 5. No obstante, esta *"imagen"* representativa del estado, sigue guardando la lógica de *"localidad"* y de diferentes niveles de abstracción (puntos, paredes, habitaciones), tal como se explica en MiniGrid Documentation.

La implementación del autoencoder es simple, se han propuesto varios modelos con varias configuraciones y se ha seleccionado mediante *ensayo y error* la configuración más favorable. Siendo esta las 4 capas convolucionales que se ven en la imagen 4.5, de tamaños 32, 64, 128 y 256 respectivamente y kernel=3. Cabe mencionar que se personalizó la última capa de activación del decoder para que la *"imagen"* de salida volviese a mostrar valores enteros en el rango [0,5].

Una vez entrenado el autoencoder, nos quedamos con la parte que nos interesa para este caso, el encoder, guardamos sus pesos para su uso recorrente en el cálculo del espacio latente de cada estado.

- **Cálculo de la elipse**, una vez tenemos un conjunto de valores en el plano bidimensional procedemos a calcular las características de la elipse asociada a la distancia de Mahalanobis que representa a estos puntos, en esta adaptación los cálculos se simplifican enormemente puesto que cada punto es un vector en 2D. El proceso es el siguiente:

  - Calculamos la media de los puntos, el resultado $(\mu x, \mu y)$ será el centro de la elipse.
  - Calculamos la matriz de covarianza de los puntos, en este caso se hará mediante el uso de la funión *cov()* de la librería *numpy*.
  - Obtenemos los vectores y valores propios de la matriz de covarianza, lo haremos mediante la función *linalg.eigh()* de *numpy*.
  - Los valores poropios representan la longitud al cuadrado de los ejes mayor y menor de la elipse respectivamente.
  - Los vectores propios indican la dirección de los ejes de la elipse.

- **Cálculo de la recompensa**, una vez tenemos la elipse calculada en el paso anterior, la recompensa se puede calcular de forma simple tal como se ve en la figura 3.7, siguiendo la fórmula:

$$\text{recompensa}_{\mathbf{t}} = \frac{{z_1}^2}{\lambda_1} + \frac{{z_2}^2}{\lambda_2}$$

En donde λ representa los semiejes y z la distancia del punto t a cada uno de los ejes.

- **Algoritmo base PPO**, en nuestro caso se utiliza el algoritmo base de RL PPO, puesto que uno de nuestros objetivos es ver la mejora que aporta este método al algoritmo PPO, en el caso del paper original, los investigadores utilizaron IMPALA como algoritmo base. Se añadirá la lógica aquí descrita al código obtenido de torch-ac.

  Para ello será necesario crear una nueva clase a partir de *PPOAlgo* a la cual se le enviará una copia del encoder entrenado anteriormente y se añadirán funciones responsables del cálculo de las recompensas intrínsecas.

- **Selección de** β , de forma idéntica al método propuesto en Henaff et al. (2023), se utilizará un hiperparámetro (β) para equilibrar la recompensa intrínseca que acabamos de calcular y la extrínseca (proveniente del entorno), de la forma:

$$r_{\text{total}} = r_{\text{extrínseca}} + \beta \cdot r_{\text{intrínseca}}$$

  Este hiperparámetro será modificado para cada entorno concreto, siendo mayor cuanto menor o más dispersa sea la recompensa. Se encuentra en el rango $[0{,}1 \text{ - } 1{,}0]$ y será seleccionado en funcion del rendimiento mediante *ensayo-error*.

Con estas adaptaciones conseguimos implementar E3B en esencia y podemos ejecutarlo en nuestro sistema de baja capacidad computacional. Es de esperar que la reducción de dimensionalidad y complejidad conlleve una reducción en el rendimiento y en la velocidad de aprendizaje del agente Hinton y Salakhutdinov (2006), no obstante en la sección 5 veremos como sigue mejorando al algoritmo *vanilla* y puede compararse con los algoritmos que marcan el estado del arte.

Resultados y Discusión

5

Veamos ahora los resultados de nuestras pruebas mediante una serie de gráficas. Se presenta una gráfica para cada uno de los 5 entornos (mapas) 4.2 , sobre los que se han probado los algoritmos. Cada gráfica tendrá una representación para cada uno de los 5 agentes 4.1 que se han probado en ellos. Estas gráficas muestran en el eje vertical la recompensa obtenida (normalizada entre 0 y 1) y en el eje horizontal el numero de *steps* o itereaciones. Con esto podemos visualizar de forma simple cuán rapido un agente aprende políticas útiles que le reporten buenas recompensas.

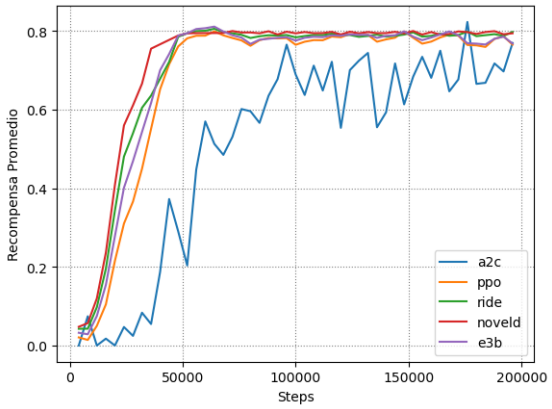


Figura 5.1: **Rendimiento en entorno MiniGrid-MultiRoom-N2-S4-v0**

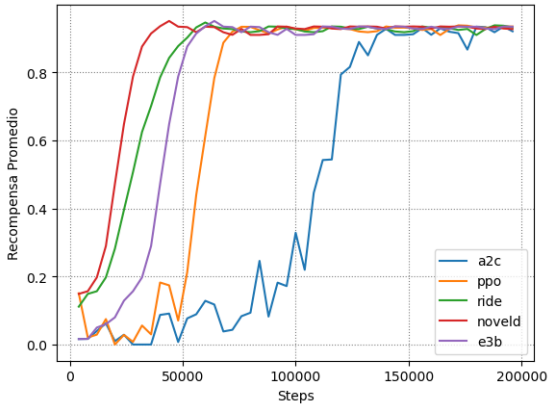


Figura 5.2: **Rendimiento en entorno MiniGrid-DoorKey-5x5-v0**

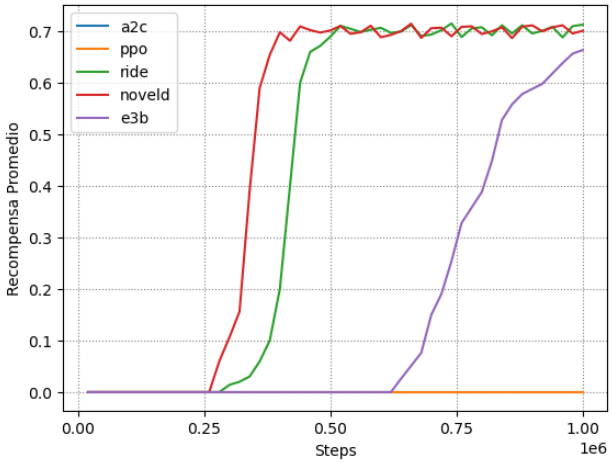


Figura 5.3: **Rendimiento en entorno MiniGrid-MultiRoom-N4-S5-v0**

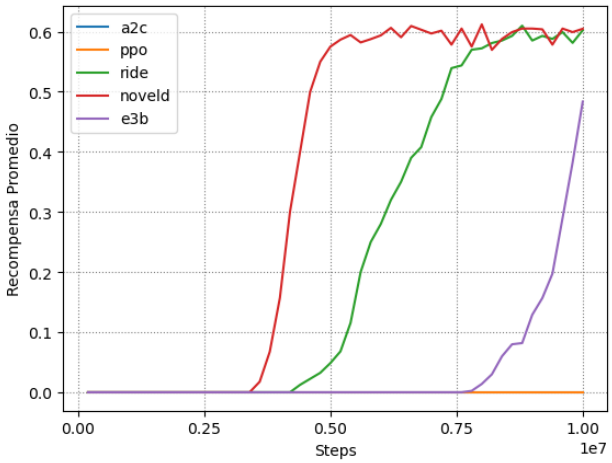


Figura 5.4: **Rendimiento en entorno MiniGrid-MultiRoom-N6-v0**

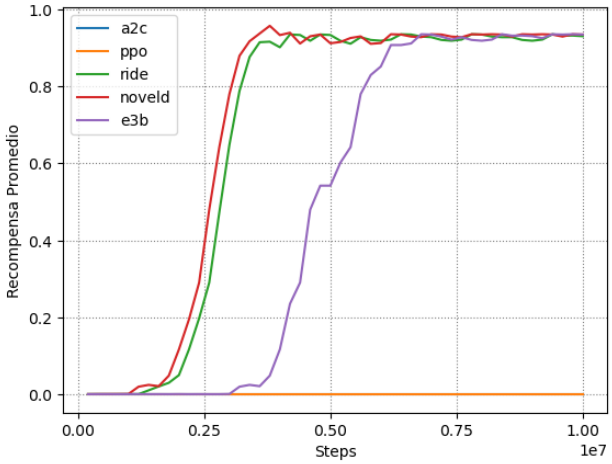


Figura 5.5: **Rendimiento en entorno MiniGrid-ObstructedMaze-2Dlh-v0**

Analizando los gráficos de la recompensa obtenida vemos, en primer lugar, que los algoritmos A2C y PPO consiguen políticas favorables para los entornos más sencillos. También podemos ver una diferencia entre estos dos, siendo PPO más rápido en converger y sobre todo más estable (motivo por el cual fué ideado).

Los algoritmos más avanzados RIDE y NovelD tampoco tienen ningún problema con estos dos primeros entornos, consiguiendo una política eficiente de forma más rápida. E3B por su parte también obtiene buenos resultados, con una convergencia similar a estos dos últimos.

En entornos en los que la recompensa es muy tardía (3, 4 o 6 habitaciones), los algoritmos base no consiguen aprender ninguna política viable y por tanto su recompensa es 0 tras millones de *steps*.

En estos entornos complejos, vemos como el algoritmo E3B consigue encontrar políticas viables a diferencia de los algoritmos base, no obstante sigue por detrás de los algoritmos RIDE y NovelD en cuanto al tiempo que necesita para el aprendizaje. Esto es muy probable que se deba a la simplificación aplicada al método E3B, puesto que en Henaff et al. (2023) este algoritmo se demostró superior a RIDE y NovelD en los entornos MiniHack y Vizdoom, los cuales funcionan de forma similar a Minigrid.

Nuestra implementación de E3B tarda en converger aproximadamente $2,5$ veces la cantidad de *steps* que tarda NovelD, en entornos de pura exploración (MultiRoom-N4 y MultiRoom-N6) y lo hace de manera un poco más progresiva. En el caso de Obstructed maze las recompensas crecen de forma menos estable en el caso de E3B, pero finalmente se estabilizan.

Podemos ver en la tabla 5.1 la cantidad aproximada de steps que necesita cada algoritmo para converger en cada entorno.

Entorno	A2C	PPO	RIDE	NovelD	E3B
MultiRoom-N2-S4	100.000	50.000	50.000	40.000	50.000
DoorKey-5x5	140.000	70.000	50.000	30.000	50.000
MultiRoom-N4-S5	-	-	500.000	300.000	1.000.000
MultiRoom-N6	-	-	8.000.000	5.000.000	>10.000.000
ObstructedMaze-2Dlh	-	-	3.000.000	3.000.000	7.000.000

Tabla 5.1: **Promedio de *steps* necesarios para convergencia**

Conclusiones

6

1. Se han logrado los objetivos propuestos en 1.2. En esta memoria queda probada la consecución de cada objetivo parcial, especialmente a nivel de revisión y análisis de la literatura actual sobre la exploración en RL, siendo esta una base sólida para el aprendizaje de esta rama de la inteligencia artificial.

2. El algoritmo mejora a los algoritmos base. Se ha comprobado como la aplicación de la técnica de E3B mejora a los algoritmos base (A2C y PPO), tanto en la velocidad de convergencia como en la recompensa media obtenida en entornos complejos.

3. La simplificación propuesta es buena. Para poder abordar la implementación de E3B en nuestro sistema hemos detallado las simplificaciones y concesiones que hemos tenido que asumir. No obstante, tras lo visto en la sección de resultados 5, podemos afirmar que nuestra versión aporta valor, siendo mejor que los algoritmos base y próxima a los algoritmos más nuevos del estado del arte.

4. Los entornos complejos, altamente dimensionales o que requieren una sucesión compleja de acciones para obtener una recompensa, son impracticables por algoritmos base (A2C o PPO), e incluso difíciles para algoritmos más avanzados, tardando demasiado tiempo en converger.

5. El costo de investigar, entrenando agentes en entornos complejos, es elevado. Se requieren sistemas con gran capacidad de computación para poder ejecutar en tiempos razonables (días) el entrenamiento de un agente con algoritmos novedosos y poder hacerlo en entornos altamente dimensionales. Sin el acceso a estos recursos solo se pueden probar simplificaciones en entornos sencillos, comprobando si ideas detrás de estos métodos funcionan, pero sin poder ver que resultados conseguirían en entornos complejos.

6. Estamos lejos de la aplicabilidad de agentes de rl a entornos reales complejos. Basta con ver lo dificil que es para un agente aprender una política razonablemente buena en un entorno tan sencillo como un videojuego para darnos cuenta que su aplicabilidad al mundo real en tareas generales requerirá mucho tiempo de investigación y mejora. No obstante, el aprendizaje por refuerzo sigue siendo la rama más prometedora para abordar estas tareas.

7. A nivel personal, el desarrollo de este proyecto me ha dado la oportunidad de utilizar los conocimientos adquiridos durante el Master, no solo los relacionados con la asignatura de aprendizaje por refuerzo, sino que se he tocado temas de redes neuronales, de desarollo en python, de optimización, de aprendizaje no supervisado... Lo cual me motiva a seguir formándome e investigando en inteligencia artificial. En resumen, ha sido un proyecto desafiante pero a la vez satisfactorio.

Limitaciones y Perspectivas de Futuro

7

7.1. Limitaciones

En esta sección veremos con qué limitaciones nos hemos topado en este trabajo, las cuales pueden ser agrupadas en:

- **Computacionales**, el grupo de investigación de Henaff et al. (2023), ha tenido a su disposición equipos con Intel(R) Xeon(R) de 80 núcleos y targetas gráficas NVIDIA GP100 valoradas en 20.000$. En nuestro caso, las pruebas fueron hechas en un equipo mucho más modesto, con un procesador Intel i5 sin targeta gráfica dedicada, es por ello que el planteamiento desde un inicio no podía ser la aplicación del algoritmo E3B tal como fué publicado por Meta en facebookresearch, si no una adaptación del mismo. La mayor simplificación consiste en el cálculo del espacio latente, en nuestro caso bidimensional frente a las 1024 dimensiones que popusieron los autores, lo cual facilita enormemente el cálculo de las elipses y recompensas.

- **De la simplificación**, motivada por la limitación anterior, esta simplificación del método causa una pérdida de eficacia en la exploración y el aprendizaje de los agentes, puesto que con la reducción de dimensionalidad tan drástica del espacio latente, las características que éste puede extraer son menos útiles para diferenciar estados, como se explica en Hinton y Salakhutdinov (2006). Por tanto, las recompensas episódicas no serán tan precisas. Otra limitación relacionada con la simplificación propuesta es el entrenamiento del extractor de características, en nuestro caso el *encoder* se entrena antes del entrenamiento del agente en lugar de hacerlo a la par que este, lo cual causa una inposibilidad de mejora del codificador tras su entrenamiento inicial.

- **Del algoritmo de exploración**, aún ignorando las dos limitaciones anteriores, y suponiendo que fuesemos capaces de ejecutar el algoritmo E3B original en nuestros entornos MiniGrid, este algoritmo tampoco es perfecto y tiene sus limitaciones. La principal es la capacidad de realmente ver que estados son muy similares y cuales son distintos, si bien a priori la técnica de elipses nos muestra cuando un estado es muy diferente de los anteriores, no aporta la idea de que estados son realmente interesantes y cuales son irrelevantes (aún siendo distintos de los demás). Esta exploración también está basada en

recompensas intrínsecas dentro de un mismo episodio, lo cual provoca que al iniciarse cada episodio no tengamos un buen cálculo de dichas recompensas.

- **Del aprendizaje por refuerzo**, el aprendizaje por refuerzo también tiene sus propias limitaciones, relacionadas con lo visto en 2.0.2, como lo son la capacidad de generalizar a nuevos entornos, teniendo que entrenar al agente especificamente en cada mapa de Minigrid; o la capacidad de determinar el mérito de cada accion en la consecución de una recompensa tardía.

7.2. Lineas futuras

Para finalizar, se proponen las siguientes lineas de investigación relacionadas con el trabajo presentado en este TFM.

- **Nuevas métricas de diferenciación**, la métrica usada en este caso son las elipses, relacionada con la distancia de Mahalanobis Mahalanobis (1936) , pero podría haber métricas más precisas para resolver el problema de ver cuando un estado es realmente novedoso.
- **Nuevos algoritmos y arquitecturas**, la búsqueda de nuevos enfoques que mejoren la eficiencia de los algoritmos actuales siempre es un buen punto de partida, en concreto se podría trabajar en el desarollo de nuevas arquitecturas de redes neuronales específicas para la tarea de exploración.
- **Nuevas formas de exploración**, la idea en la que se basan todos los algoritmos de exploración actuales es la curiosidad, premiar al agente cuando encuentra nuevos estados para incentivar su exploración, no obstante, se podría investigar en como dotar a los agentes de otras emociones útiles para los seres humanos en el aprendizaje, tales como el aburrimiento, o el miedo, que conllevaría recompensas negativas ante posibles peligros.
- **Aprendizaje multimapa-multitarea**, sería muy interesante explorar cómo los agentes pueden aprender de manera más efectiva a partir de múltiples tareas relacionadas y aplicar lo aprendido en una tarea nueva. Por ejemplo, en el caso de Minigrid, sería interesante ver como podemos diseñar un modelo que permita llevar el conocimiento que el agente aprendió en mapas simples a mapas más complejos.
- **Adaptacion a nuevos entornos**, relacionado con el punto anterior, sería todavía más interesante idear la manera que los *"conceptos"* aprendidos durante el entrenamiento (tales como moverse, abrir puertas con llaves, etc.) pudieran ser llevados a otros entornos de manera eficiente. Un ejemplo sería poder hacer uso del conocimiento aprendido por un agente en entornos como Minigrid o Habitat para incorporarlo en robots-aspiradora y hacerlos más eficientes. El trabajo de idear un modelo que luego pueda ser generalizable y adaptable a entornos distintos es un gran ámbito de investigación, que facilitaría el

entrenamiento de agentes para ámbitos complejos o con procesos de entrenamiento costosos.

- **Investigación de la atención**, otro punto que puede mejorar estos algoritmos es la incorporación de módulos de atención, que tan buenos resultados han dado en el entorno del lenguaje Vaswani et al. (2017). Habría que estudiar como este enfoque puede mejorar el rendimiento de los agentes enfocándose en la información y acciones relevantes para conseguir la recompensa.

- **Recompensas intrínsecas a largo plazo**, en nuestro caso, las recompensas intrínsecas son calculadas en cada observación, pero no tenemos una forma de recompensar al agente (intrinsecamente) a largo plazo, por ejemplo cuando ha conseguido explorar todo el entorno o cuando ha descubierto todos los objetos. Se podría estudiar la forma de incorporar estas recompensas a largo plazo de forma intrínseca y con ello premiar hitos en la exploración. Incluso se podría proponer el estudio de recompensas intrínsecas entre distintos episodios.

- **Creación de modelos del entorno**, también es interesante el estudio de como los agentes pueden aprender a construir modelos del mundo para su toma de decisiones y planificación. En cierto modo, podemos ver el espacio latente del método E3B como un modelo simple e inexacto a nivel episodio. Esta linea de investigación trataría de enseñar a los agentes una forma de crear un modelo del entorno de manara que reduzcan el ruido en las observaciones y realmente basen sus decisiones en los aspectos importantes del entorno.

Lista de Acrónimos

A2C *Advantage Actor-Critic*.

ADAM *Adaptive Moment Estimation*.

AGAC *Adversarially Guided Actor-Critic*.

CMDP Procesos de decisión de Markov contextuales o *Contextual Markov Decision Process*.

DQN *Deep Q-Networks*.

E3B *Exploration via Elliptical Episodic Bonuses*.

GAE Estimación de Ventaja Generalizada o *Generalized Advantage Estimation*.

IA Inteligencia Artificial.

ICM *Intrinsic Curiosity Module*.

IMPALA *Importance Weighted Actor-Learner Architecture*.

KL *Kullback-Leibler*.

MCTS Busqueda en arbol Monte Carlo o *Monte Carlo Tree Search*.

PLE *Pygame Learning Environment*.

PPO *Proximal Policy Optimization*.

RAPID *Ranking the Episodes*.

RIDE *Rewarding Impact-Driven Exploration*.

RL Aprendizaje por refuerzo o *Reinforcement Learning*.

RND *Random Network Distillation*.

SARSA *State-Action-Reward-State-Action*.

TFM Trabajo de fin de Máster.

TRPO *Trust Region Policy Optimization*.

UCB *Upper Confidence Bound*.

Bibliografía

Ackley, D. H., Hinton, G. E., y Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169.

AlMahamid, F. y Grolinger, K. (2021). Reinforcement learning algorithms: An overview and classification. In *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE.

Andres, A., Villar-Rodriguez, E., y Ser, J. D. (2022). Towards improving exploration in self-imitation learning using intrinsic motivation.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., y Zaremba, W. (2018). Hindsight experience replay.

Auer, P., Cesa-Bianchi, N., y Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2):235–256.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., y Petersen, S. (2016). Deepmind lab.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., y Bengio, S. (2017). Neural combinatorial optimization with reinforcement learning.

Berger-Tal, O., Nathan, J., Meron, E., y Saltz, D. (2014). The exploration-exploitation dilemma: a multidisciplinary framework. *PloS one*, 9(4):e95693.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., y Zaremba, W. (2016). Openai gym.

Burda, Y., Edwards, H., Storkey, A., y Klimov, O. (2018). Exploration by random network distillation.

Chapelle, O. y Li, L. (2011). An empirical evaluation of thompson sampling. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., y Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., y Bengio, Y. (2019). Babyai: A platform to study the sample efficiency of grounded language learning.

Chevalier-Boisvert, M., Willems, L., y Pal, S. (2018). Minimalistic gridworld environment for gymnasium.

Feinberg, E. A. (2000). Constrained discounted markov decision processes and hamiltonian cycles. *Mathematics of Operations Research*, 25(1):130–140.

Flet-Berliac, Y., Ferret, J., Pietquin, O., Preux, P., y Geist, M. (2021). Adversarially guided actor-critic.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., y Legg, S. (2019). Noisy networks for exploration.

Ha, D. y Schmidhuber, J. (2018). World models.

Henaff, M., Raileanu, R., Jiang, M., y Rocktäschel, T. (2023). Exploration via elliptical episodic bonuses.

Hinton, G. E. y Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hinton, G. E. y Sejnowski, T. J. (1986). *Learning and Relearning in Boltzmann Machines*, page 282–317. MIT Press, Cambridge, MA, USA.

Johnson, M., Hofmann, K., Hutton, T., Bignell, D., y Hofmann, K. (2016). The malmo platform for artificial intelligence experimentation. In *25th International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI - Association for the Advancement of Artificial Intelligence.

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., y Lange, D. (2020). Unity: A general platform for intelligent agents.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., y Jaśkowski, W. (2016). Vizdoom: A doom-based ai research platform for visual reinforcement learning.

Ladosz, P., Weng, L., Kim, M., y Oh, H. (2022). Exploration in deep reinforcement learning: A survey. *Information Fusion*, 85:1–22.

Lecun, Y., Bottou, L., Bengio, Y., y Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lehman, J., Clune, J., Misevic, D., Adami, C., Altenberg, L., Beaulieu, J., Bentley, P. J., Bernard, S., Beslon, G., Bryson, D. M., Chrabaszcz, P., Cheney, N., Cully, A., Doncieux, S., Dyer, F. C., Ellefsen, K. O., Feldt, R., Fischer, S., Forrest, S., Frénoy, A., Gagné, C., Goff, L. L., Grabowski, L. M., Hodjat, B., Hutter, F., Keller, L., Knibbe, C., Krcah, P., Lenski, R. E., Lipson, H., MacCurdy, R., Maestre, C., Miikkulainen, R., Mitri, S., Moriarty, D. E., Mouret, J.-B., Nguyen, A., Ofria, C., Parizeau, M., Parsons, D., Pennock, R. T., Punch, W. F., Ray, T. S., Schoenauer, M., Shulte, E., Sims, K., Stanley, K. O., Taddei, F., Tarapore, D., Thibault, S., Weimer, W., Watson, R., y Yosinski, J. (2019). The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities.

Mahalanobis, P. C. (1936). On the generalized distance in statistics. National Institute of Science of India.

Masci, J., Meier, U., Cireşan, D., y Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In Honkela, T., Duch, W., Girolami, M., y Kaski, S., editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, pages 52–59, Berlin, Heidelberg. Springer Berlin Heidelberg.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., y Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Balcan, M. F. y Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., y Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., d. O. Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., y Zhang, S. (2019a). Dota 2 with large scale deep reinforcement learning.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., y Zhang, L. (2019b). Solving rubik's cube with a robot hand.

Pathak, D., Agrawal, P., Efros, A. A., y Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In Precup, D. y Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2778–2787. PMLR.

Petrenko, A., Wijmans, E., Shacklett, B., y Koltun, V. (2021). Megaverse: Simulating embodied agents at one million experiences per second. In Meila, M. y Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8556–8566. PMLR.

Raileanu, R. y Rocktäschel, T. (2020). Ride: Rewarding impact-driven exploration for procedurally-generated environments.

Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Küttler, H., Grefenstette, E., y Rocktäschel, T. (2021). Minihack the planet: A sandbox for open-ended reinforcement learning research.

Savinov, N., Raichuk, A., Marinier, R., Vincent, D., Pollefeys, M., Lillicrap, T., y Gelly, S. (2019). Episodic curiosity through reachability. In *7th International Conference on Learning Representations, ICLR 2019*.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., y Abbeel, P. (2015). Trust region policy optimization. *CoRR*, abs/1502.05477.

Schulman, J., Moritz, P., Levine, S., Jordan, M., y Abbeel, P. (2018). High-dimensional continuous control using generalized advantage estimation.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., y Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Sherman, J. y Morrison, W. J. (1950). Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., y Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., y Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

Sutton, R. S. y Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

van Hasselt, H., Guez, A., y Silver, D. (2015). Deep reinforcement learning with double q-learning.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., y Polosukhin, I. (2017). Attention is all you need.

Witten, I. H. (1977). An adaptive optimal controller for discrete-time markov environments. *Inform. Control*, 34:286–295.

Zha, D., Ma, W., Yuan, L., Hu, X., y Liu, J. (2021). Rank the episodes: A simple approach for exploration in procedurally-generated environments.

Zhang, T., Xu, H., Wang, X., Wu, Y., Keutzer, K., Gonzalez, J. E., y Tian, Y. (2021). Noveld: A simple yet effective exploration criterion. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., y Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 25217–25230. Curran Associates, Inc.

Zheng, S. (2022). The influence of different environments on reinforcement learning. In *2022 3rd International Conference on Computer Vision, Image and Deep Learning International Conference on Computer Engineering and Applications (CVIDL ICCEA)*, pages 401–406.