UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Evaluación y mejora de la Generalización en algoritmos de Aprendizaje por Refuerzo

POR

Gustavo Alonso González Gutiérrez

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Informático

Profesor Guía

Julio Erasmo Godoy del Campo

2025

Concepción (Chile)

Índice

Resumen

La generalización en el aprendizaje por refuerzo (RL) continúa siendo una de las principales barreras para su aplicación en entornos reales, donde la variabilidad y las condiciones no previstas pueden deteriorar notoriamente el rendimiento de los agentes entrenados. Esta memoria de título aborda dicho problema evaluando y mejorando la capacidad de generalización de algoritmos con enfoques distintos, tanto basados en valores (DQN y Rainbow), basados en políticas (PPO y PPO con LSTM) y neuroevolutivos (NEAT e HyperNEAT), utilizando como entorno experimental diversos niveles del videojuego Super Mario Bros.

Se implementan y combinan tres métodos complementarios orientados a mejorar la generalización, Domain Randomization para aumentar la diversidad de dinámicas del entorno, la arquitectura de redes neuronales IMPALA como mejora algorítmica en la extracción de características, y Explore Go como estrategia de exploración que amplía la cobertura del espacio de estados sin afectar la estabilidad del aprendizaje. Los aportes de este trabajo incluyen la validación empírica de que métodos de generalización con enfoques distintos pueden combinarse efectivamente, y la identificación de trade-offs importantes entre la eficiencia de convergencia y capacidad de generalización.

Los resultados demuestran que Rainbow con los tres métodos aplicados logra mejoras significativas en niveles no vistos, mientras que PPO muestra una mejor generalización sin métodos adicionales. IMPALA resulta ser el método de mayor impacto, y Explore Go ayudó a obtener mejoras significativas en niveles donde previamente no habían avances significativos. Los algoritmos neuroevolutivos muestran limitaciones significativas sin redes profundas. Este trabajo proporciona una metodología replicable y evidencia empírica sobre combinaciones efectivas de métodos según el tipo de algoritmo.

1. Introducción

En la actualidad, el aprendizaje por refuerzo (RL) [14] ha surgido como una vía prometedora para resolver problemas del mundo real. Problemas como el control de robots en entornos dinámicos o la conducción con piloto automático, son casos donde el agente debe aprender comportamientos eficientes en escenarios muy cambiantes, es entonces cuando la habilidad para adaptarse se convierte en un desafío, ya que un agente que solo funciona bien en un entorno de entrenamiento fijo, no servirá de mucho al enfrentarse a la variabilidad del mundo real.

El aprendizaje por refuerzo es un área del aprendizaje automático (Machine Learning), el cual se basa en dejar a un agente aprender a tomar decisiones mediante la interacción con un entorno. A diferencia del aprendizaje supervisado, donde el agente se entrena con datos etiquetados, en el aprendizaje por refuerzo el agente explora un entorno por su cuenta y descubre qué acciones tomar para maximizar la señal de recompensa obtenida.

En este contexto, el agente toma acciones en un entorno dado, pasando entre diferentes estados y recibiendo recompensas o castigos dependiendo de la calidad de sus decisiones con respecto al objetivo del entorno, el objetivo del agente es maximizar la recompensa total que puede obtener durante un "episodio" dentro del entorno. Para conseguirlo, el agente debe aprender una política óptima, que es un mapeo de estados a acciones que se consideran óptimas para maximizar la recompensa obtenida.

Uno de los desafíos presentes a día de hoy en el aprendizaje por refuerzo es la capacidad de generalización [23], es decir, la habilidad de un agente para aplicar el conocimiento adquirido a diferentes situaciones o distintos entornos similares. Es muy común encontrar que el desempeño de un agente empeora considerablemente cuando se enfrenta a nuevas condiciones no presentes durante el entrenamiento, siendo esto una gran limitante que impide su aplicación en entornos reales de manera segura, donde la variabilidad y situaciones no previstas son bastante comunes. Por ejemplo, un agente que haya aprendido a jugar un nivel de un videojuego puede fallar rotundamente al enfrentarse a otro nivel con obstáculos diferentes, incluso si estos comparten la misma lógica para superarlos, evidenciando un "sobreajuste" y memorización de los datos de entrenamiento, y por consiguiente, una mala generalización.

Para abordar este tema, se han elegido dos algoritmos de aprendizaje por refuerzo, PPO (Proximal Policy Optimization) [2] y DQN (Deep Q-Network) [1], los cuales presentan distintos enfoques, por un lado PPO es un algoritmo basado en políticas y DQN basado en valores, ambos son ampliamente utilizados para el desarrollo de agentes de aprendizaje por refuerzo. Además se utilizará un algoritmo neuroevolutivo, NEAT (NeuroEvolution of Augmenting Topologies) [3] el cual ajusta su estructura de red neuronal y los pesos de las conexiones, lo que le permite la adaptación a entornos complejos.

En la presente memoria de título, se busca evaluar, mejorar y analizar el desempeño y la capacidad de generalización de los algoritmos mencionados utilizando entornos de videojuegos.

1.2 Objetivos

La presente memoria de título tiene como objetivo principal implementar y evaluar métodos que buscan mejoras en la capacidad de generalización de algoritmos tales como DQN, PPO y NEAT, utilizando diferentes entornos del videojuego Super Mario Bros.

Para abordar este problema, los objetivos específicos son:

· Implementación de métodos que mejoran la capacidad de generalización de los algoritmos, validando y evaluando el desempeño obtenido.

· Comparación de algoritmos en términos de desempeño y de generalización, usando como métricas la recompensa promedio obtenida y la tasa de éxito en entornos nunca vistos.

· Determinar qué algoritmo obtuvo el mejor rendimiento en cuanto a su capacidad de generalización en las pruebas realizadas, identificando las razones.

2. Marco teórico

A continuación, se explican los conceptos teóricos fundamentales que cimentan las bases para el desarrollo de la presente memoria de título.

2.1 Machine learning

Machine learning o aprendizaje automático [20], es una rama de la inteligencia artificial en la cual, a través de algoritmos, los computadores son capaces de aprender relaciones o patrones a partir de un conjunto de datos y mejorar su desempeño en tareas específicas sin ser programadas específicamente para eso, por ejemplo clasificación de imágenes, realizar predicciones, buscar patrones o el control autónomo de robots.

Dependiendo del tipo de problema, el aprendizaje del computador se puede clasificar como (Figura 1):

· **Aprendizaje supervisado:** Usado para tareas de clasificación o regresión, aprendiendo a partir de un conjunto de datos etiquetados, donde cada dato ingresado tiene una salida conocida. El objetivo es encontrar una relación entre los datos y sus salidas para tener una predicción acertada al momento de ingresar nuevos datos.

· **Aprendizaje no supervisado:** Usado en la búsqueda de grupos en los datos, de patrones o relaciones entre los mismos, donde se aprende a partir de un conjunto de datos no etiquetados, en busca de patrones de comportamiento o estructuras ocultas.

· **Aprendizaje por refuerzo:** Comúnmente usado para el entrenamiento de robots y optimización, donde el aprendizaje ocurre mediante la interacción de un agente con un entorno, el cual toma decisiones y recibe señales de recompensa dependiendo del resultado de sus acciones. El objetivo es aprender una política que maximice su recompensa acumulada a largo plazo.

Dado que no existen etiquetas que indiquen la calidad de las acciones tomadas en entornos de videojuegos, el enfoque de esta memoria de título es el aprendizaje por refuerzo, que permite al agente aprender de la interacción con el entorno y las recompensas obtenidas.

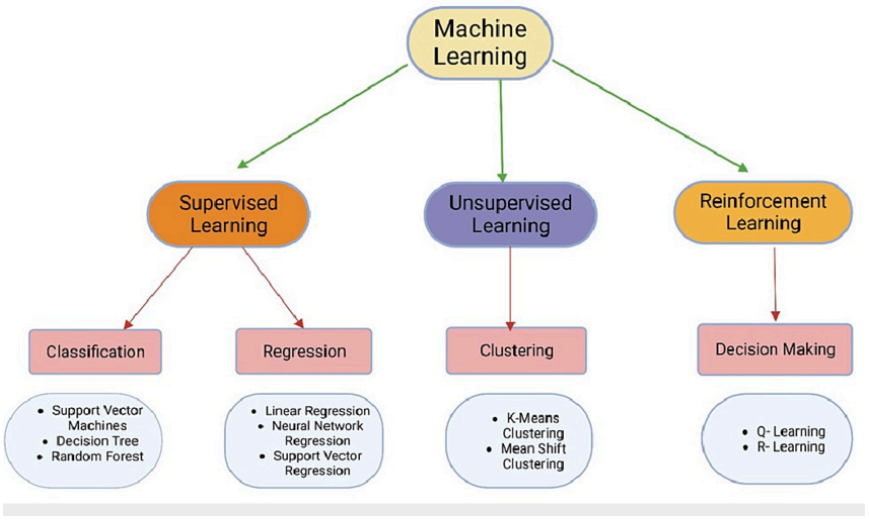

Figura 1. Diagrama de tipos de aprendizaje en Machine Learning con algoritmos utilizados, obtenido de [22].

2.1.1 Aprendizaje por refuerzo

Es un tipo de aprendizaje automático en el cual un agente aprende a maximizar una recompensa acumulada tomando decisiones dentro de un entorno [14]. El aprendizaje del agente consiste en interactuar con un entorno de manera iterativa hasta converger en un comportamiento establecido. En el contexto de videojuegos, un episodio puede ser todas las acciones realizadas desde empezar un nivel hasta terminarlo, o hasta perder una o más vidas. Durante estos episodios de entrenamiento, el agente recibe retroalimentación tras cada interacción con el entorno en forma de recompensas positivas o negativas (Figura 2), las cuales utiliza para ajustar su comportamiento y aprender a interactuar con el entorno efectivamente.

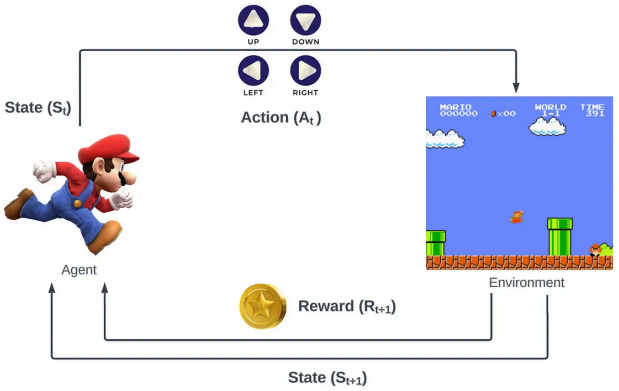

Figura 2. Diagrama de funcionamiento básico de aprendizaje por refuerzo, obtenido de [21].

El aprendizaje por refuerzo aborda un problema formulado a partir del proceso de decisión de Markov (MDP, por sus siglas en inglés) [14], el cual describe formalmente cómo un agente toma decisiones en un entorno donde los resultados son en parte aleatorios y en otra parte controlados por el propio agente.

Un MDP se compone de los siguientes elementos:

· **Agente:** Es el que toma las decisiones, en un videojuego, el agente corresponde al jugador, en el contexto actual, el jugador corresponderá a un modelo de inteligencia artificial.

· **Entorno (ambiente):** Conjunto de todos los estados posibles S. Cada estado s describe la situación del entorno en un momento dado, por ejemplo, en Super Mario Bros, cada nivel constituye un entorno distinto, y un estado puede describir la posición de Mario y los enemigos que hay en pantalla.

· **Acciones (A):** Conjunto de decisiones que el agente puede tomar en un estado s determinado, cada acción, como moverse o saltar, se denota como a.

· **Función de transición ($P_a(s,\ s')$):** Es la probabilidad de que una acción a, en un estado s y en un momento t, lleve al estado s' en el momento $t\ +\ 1$

· **Función de recompensa ($R_a(s,\ s')$):** Retroalimentación que recibe el agente tras ejecutar la acción a para pasar del estado s a s'. Si esta acción contribuye a cumplir el objetivo del entorno, por ejemplo, en Super Mario Bros, avanzar hacia la meta, esta recompensa será positiva, en caso contrario, negativa.

· **Política (π):** Es la estrategia aprendida por el agente, define qué acción tomar según un estado dado.

El objetivo del MDP es elegir una política que logre maximizar una función acumulativa de recompensa, normalmente la suma descontada esperada en un horizonte potencialmente infinito:

$$E\left[\sum_{t=0}^{\infty}\gamma^t R_{a_t}(s_t, s_{t+1})\right]$$

Donde la esperanza se toma sobre $t + 1$ obtenido desde $P_a(s_t, s_{t+1})$, y γ es el factor de descuento, la cual toma valores entre 0 y 1, a más bajo el valor de γ, mayor motivación para el agente de tomar acciones que lo favorezcan a corto plazo, en caso contrario, el agente tendrá en consideración acciones que lo puedan favorecer a largo plazo.

2.2 Contextual Markov Decision Process

Cuando se trata de la generalización en el aprendizaje por refuerzo, el MDP convencional puede resultar algo limitado, ya que parte de la idea de unas dinámicas y recompensas fijas. Sin embargo, en muchos casos de la vida real, las tareas pueden compartir una estructura común, pero variar en cosas puntuales. Por ejemplo al recoger una taza, si bien hay una acción general para recoger una taza, esta puede variar según la posición inicial, la orientación de la taza, la forma de esta, etc. Toda esta variabilidad mencionada puede hacer que un agente entrenado en un conjunto de tareas fijas tenga dificultades para generalizar a una nueva tarea similar pero nunca vista.

Dado lo anterior se propuso el Proceso de decisión de Markov Contextual (CMDP, por sus siglas en inglés) [27], el cual extiende los MDPs convencionales incorporando un contexto $c \in C$, pudiendo cambiar la dinámica del entorno o su función de recompensa, cada contexto c define un MDP particular, como se puede ver en la Figura 3. Un CMDP se define como una 6-tupla $(C, S, A, T_c, R_c, \gamma)$ donde:

· C es el espacio de contextos.

· S es el espacio de estados.

· A es el espacio de acciones.

· $T_c(s'|s, a)$ es la dinámica de transición condicionada por el contexto.

· $R_c(s, a)$ es la función de recompensa condicionada por el contexto.

· γ es el factor de descuento.

El objetivo de un agente en un CMDP es aprender una política π que funcione bien en una amplia variedad de contextos, incluyendo aquellos no vistos durante el entrenamiento.

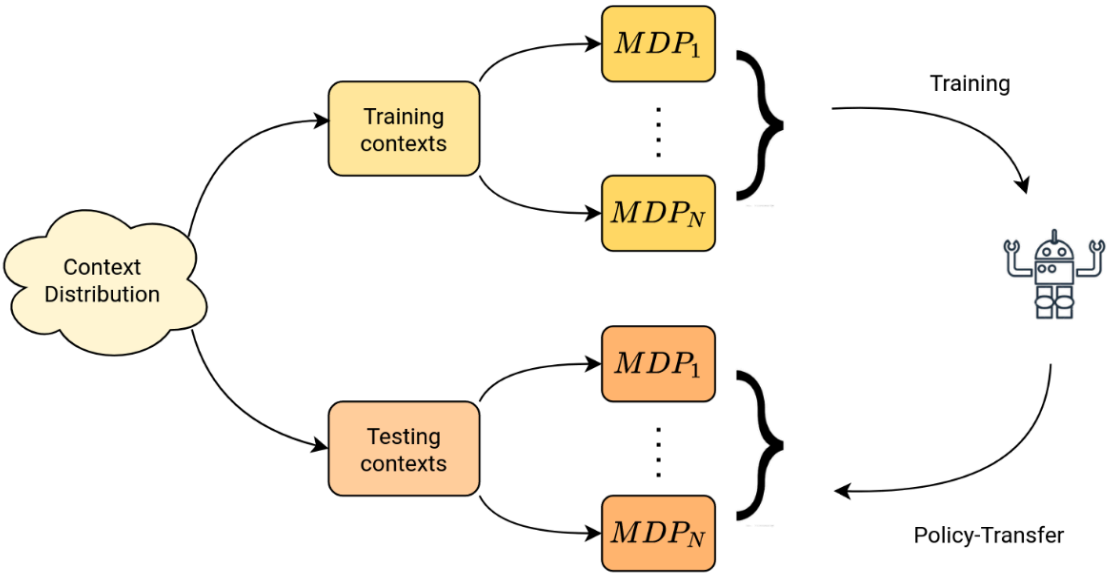

Figura 3. Diagrama de CMDP, obtenido de [28].

2.3 Q-learning

Es un algoritmo fundamental en el aprendizaje por refuerzo. Basado en valores, su objetivo principal es ayudar al agente a aprender qué acción tomar para maximizar su recompensa. Esto se logra estimando una función de valor Q (Figura 4) que representa la calidad de tomar la acción a en un estado s. Se utiliza una Q-table para almacenar valores de la función para cada combinación posible de pares (s, a) [19].

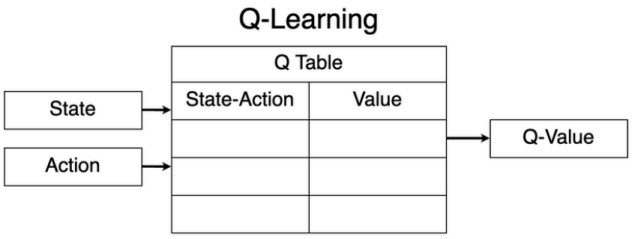

Figura 4. Diagrama de funcionamiento de Q-learning, obtenido de [18].

La actualización de los valores utiliza la siguiente fórmula (Ecuación de Bellman):

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma_{max\,\alpha'} Q(s',a') - Q(s,a)]$$

donde:

· r es la recompensa recibida tras realizar la acción a.

· s' es el nuevo estado al que el agente llega después de realizar la acción a.

· α es la tasa de aprendizaje, determina la importancia que se le da a la nueva información obtenida.

· γ es el factor de descuento, el cual determina la importancia de las recompensas futuras.

Q-learning puede ser una buena alternativa para problemas pequeños y en entornos discretos dada su fácil implementación, dado que es un algoritmo off-policy. Pero al estar en entornos más grandes y complejos, la Q-table podría resultar muy grande y difícil de manejar, lo que resulta en problemas de memoria y en tiempo de cómputo. Por esto más adelante se ocuparía Q-learning como base para crear nuevos algoritmos más potentes que logren superar sus limitaciones.

2.4 REINFORCE y algoritmos de Gradiente de política

Al igual que Q-learning, REINFORCE es un algoritmo fundamental para el aprendizaje por refuerzo, siendo este uno de los primeros algoritmos de gradiente de política [14], se basa en optimizar los parámetros θ de una política estocástica π_θ.

A diferencia de Q-learning, donde a partir de Q-values se elige la siguiente acción, REINFORCE (Figura 5) le da probabilidades a cada acción posible en una trayectoria, donde la probabilidad de cada acción se denota como $\pi_\theta(s|a)$, la probabilidad de elegir la acción a dado el estado s.

El objetivo es aumentar las probabilidades de acciones favorables durante el entrenamiento, lo cual se logra mediante la actualización de los parámetros de la política, usando ascenso de gradiente:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi_\theta)$$

donde α corresponde al learning rate y $J(\pi_\theta)$ equivale a una función de pérdida.

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Figura 5. Pseudocódigo de REINFORCE. Obtenido de [14].

De este algoritmo salen muchos otros algoritmos basados en esta idea, que mejoran su eficiencia y estabilidad, añadiendo mejoras para estimar mejor las ventajas o acelerar la convergencia a una política óptima, combinando ideas de los métodos basados en valores y basados en políticas, modificando la forma de seleccionar acciones o evaluar su calidad.

2.5 Redes neuronales

Las redes neuronales (Figura 7) son un mecanismo utilizado en el aprendizaje automático profundo, el cual simula el mecanismo de aprendizaje en organismos biológicos [24]. Estas redes contienen unidades computacionales llamadas neuronas o perceptrones, conectadas entre sí mediante pesos que representan conexiones sinápticas en los organismos biológicos.

El perceptrón (Figura 6) es la unidad básica de una red neuronal, el cual recibe una o varias entradas, las multiplica por un conjunto de pesos, suma los resultados, y aplicando una función de activación produce una salida. Esto permite al perceptrón tomar decisiones simples, y al unir múltiples perceptrones, se logran construir redes más complejas capaces de cumplir tareas más difíciles.

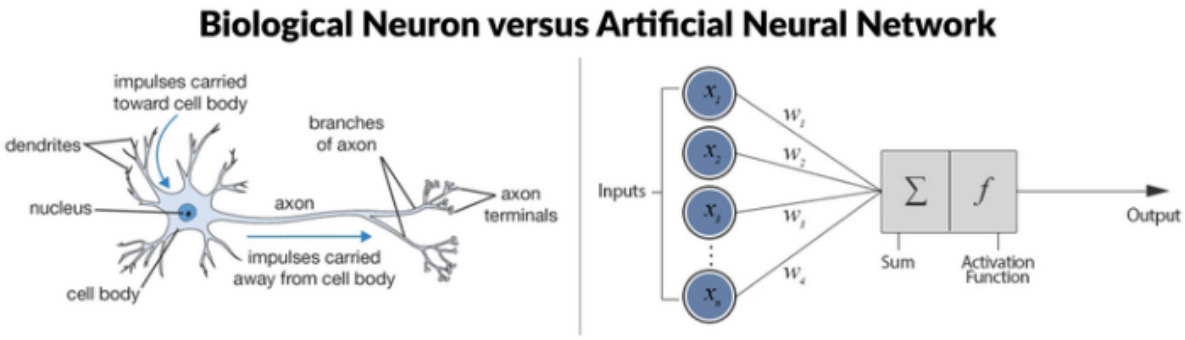

Figura 6. Comparación de una neurona con un perceptrón. Obtenido de [36].

Estas arquitecturas permiten calcular funciones complejas a través de un proceso de propagación hacia adelante de la información en la red, mientras que el aprendizaje ocurre ajustando los pesos mediante la propagación hacia atrás en base a los errores de predicción realizados durante el entrenamiento.

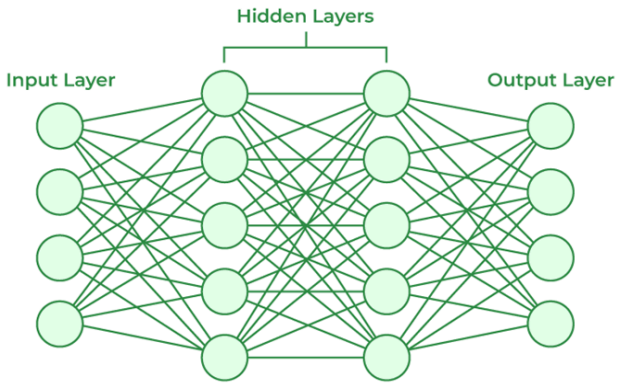

Figura 7. Red neuronal con capa de entrada, capas ocultas y capa de salida, obtenido de [16].

Existen diversos tipos de redes neuronales con distintos propósitos, algunas de ellas son:

· **Redes neuronales recurrentes (RNN):** Están diseñadas para datos secuenciales como texto, series de tiempo y secuencias biológicas, útiles para tareas como predicción de series temporales.

· **Redes neuronales convolucionales (CNN):** Utilizadas para procesar imágenes. Estas redes emplean capas convolucionales (Figura 8) que aprenden características relacionadas con la estructura de los datos, usado para reconocimiento de imágenes y visión computacional.

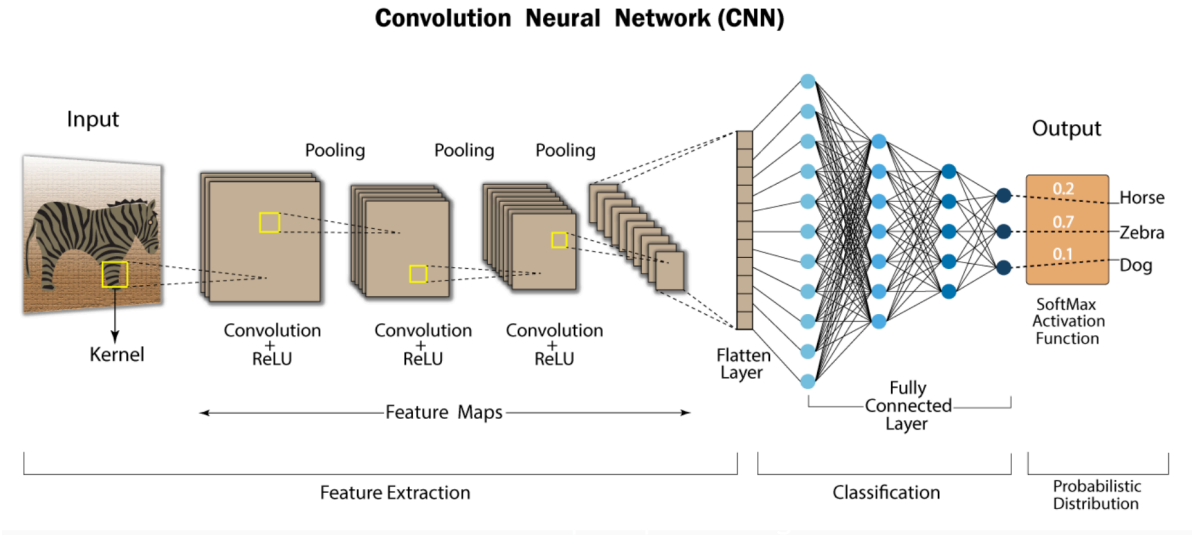

Figura 8. Funcionamiento de redes neuronales convolucionales en clasificación de imágenes, obtenido de [26].

En los algoritmos que se presentan en las siguientes secciones, las redes convolucionales cumplen un rol fundamental, ya que los entornos son percibidos como fotogramas que son procesados por las redes convolucionales, entregando las mejores acciones posibles.

Los datos son procesados (Figura 9) por las redes convolucionales de la siguiente manera:

1. Los fotogramas del entorno son representados como un tensor de dimensiones (altura x ancho x canales), generalmente las imágenes procesadas son RGB por lo que se utilizan 3 canales (rojo, verde y azul).
2. Luego, pasan a las capas convolucionales, donde se extraen sus características, mediante operaciones convolucionales, donde filtros o "kernels" (tensores de menor dimensión) recorren la entrada recibida realizando productos punto con pequeñas regiones de la imagen, generando un mapa de características.

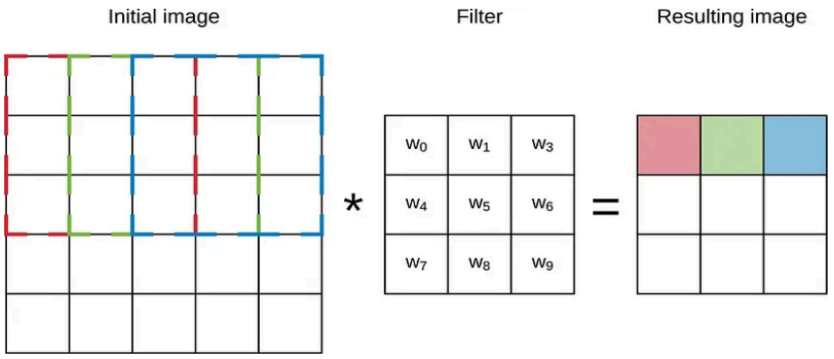

Figura 9. Funcionamiento de operación convolucional, obtenido de [25].

3. Después de la convolución, generalmente se aplica una función de activación como ReLU o Sigmoid (Anexo A) con la finalidad de introducir no linealidad, permitiendo a la red aprender relaciones más complejas, consiguiendo que los resultados sean más precisos.
4. Luego de la función de activación, se suele usar una capa de pooling, con la finalidad de reducir la resolución de los mapas de características, manteniendo la información relevante a la vez que reduce el tamaño de los datos, ayudando a reducir el sobreajuste.
5. Luego, estos mapas de características se aplanan en un vector unidimensional en las capas completamente conectadas.

6. Finalmente la red convolucional genera un vector de probabilidades o valores asociados.

Estas redes permiten, aproximar la función Q y mapearlos a estados específicos, permitiendo usar este enfoque en entornos más complejos, resolviendo el problema de escalabilidad de algoritmos tabulares como Q-learning visto previamente u otros.

2.6 Deep Q-Network

Deep Q-Network (DQN) es un algoritmo que combina Q-learning con redes neuronales [1], el cual logró superar el rendimiento de personas expertas en diversos videojuegos de Atari, aprendiendo solo a base de los píxeles en pantalla.

DQN, en lugar de usar una tabla para almacenar los Q-values, utiliza una red neuronal para aproximar la función Q (Figura 10). Esta red neuronal toma como entrada el estado s y predice los valores $Q(s, a)$ para todas las acciones posibles a, así el agente puede estimar la mejor acción sin necesidad de almacenar todos los Q-values posibles.

A continuación, se explican los componentes esenciales de DQN:

· **Experience replay:** El agente almacena durante el entrenamiento un historial de transiciones, guardando una tupla que incluye el estado, acción, recompensa y nuevo estado, para que, en lugar de actualizar la red neuronal tras cada transición, se toma una transición aleatoria de almacenada para entrenar la red, lo que permite reducir la correlación entre transiciones consecutivas, mejorando la estabilidad del entrenamiento.

· **Target network:** Se utilizan dos redes neuronales, la principal, la cual se entrena y se usa para predecir los Q-values, y la red objetivo o target, la cual se utiliza para calcular los Q-values objetivo (el valor $r + \gamma_{max\, a'} Q(s', a')$) y se mantiene congelada por un número fijo de pasos de entrenamiento, para luego actualizarla usando los valores de la red principal, evitando que la red se ajuste continuamente a una estimación inestable.

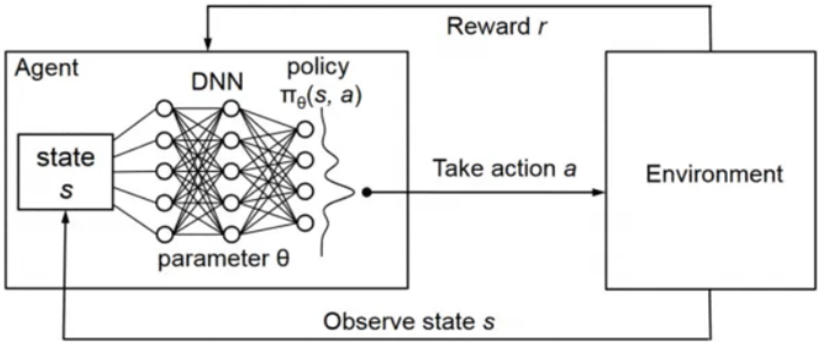

Figura 10. Diagrama de funcionamiento de DQN, obtenido de [17].

A continuación, se describe el funcionamiento de DQN a alto nivel:

1. Se inicializa la red neuronal con pesos aleatorios para predecir los Q-values, y se crea la experience replay para almacenar las transiciones.
2. Para entrenar al agente, en cada paso se observa el estado actual, basado en eso, el agente selecciona una acción (Algoritmo 1) usando una política de exploración/explotación, la más usada suele ser ε-greedy, eligiendo la acción óptima la mayoría de las veces, con alguna probabilidad de seleccionar una acción aleatoria.
3. El agente realiza la acción α, recibe la recompensa r y observa el nuevo estado s', luego se almacena la transición (s, a, r, s') en el experience replay, para luego tomar una transición aleatoria para que la Q-network se entrene minimizando la diferencia entre los Q-values actuales y los Q-values objetivo calculados usando la target-network, pasado una cantidad fija de pasos, los parámetros de la Q-network principal se copian a la target-network.

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Algoritmo 1. Pseudocódigo de DQN, obtenido de [1].

2.6.1 Rainbow

Rainbow es una variante de DQN presentada por investigadores de DeepMind [5], que combina diversas mejoras descubiertas tras la publicación inicial de DQN, tales como:

· **Double Q-learning:** DQN existe una sobreestimación de los Q-values de las acciones, es decir, el agente elige acciones subóptimas ya que tiene un Q-value mayor. Esto pasa ya que la selección de acciones y la evaluación de estas se realizan en la misma red, lo que puede generar que el agente elija acciones subóptimas. Para solucionar esto, van Hasselt propuso el algoritmo Double Q-learning [6], el cual consiste en separar la selección de la acción con su evaluación usando dos redes diferentes, la red principal para elegir la mejor acción posible, y la red target para evaluar la acción, reduciendo el sesgo y mejorando la estabilidad del entrenamiento.

· **Prioritized experience replay (PER):** En DQN, las transiciones guardadas en el experience replay que se utilizan para enseñar al agente se seleccionan al azar, esto puede perjudicar al aprendizaje del agente ya que se pueden estar obviando algunas transiciones con un mayor valor de aprendizaje que puedan resultar más útiles.
Introducido por DeepMind [7], prioritized experience replay busca que el agente de más prioridad a las transiciones con mayor error temporal (TD error), dicho de otra forma, las transiciones en las que la predicción del agente se aleja más del resultado obtenido, siendo estas las transiciones donde el agente más puede aprender.

· **Dueling network:** Es una modificación de la arquitectura de red presentada por Wang [8], la cual separa el cálculo del valor del estado (lo bueno que es estar en un estado s), del valor de cada acción (que tan buena es una acción a en un estado s). Esta separación (Figura 11) permite al agente aprender mejor en situaciones donde la acción no es tan importante, o en donde todas las acciones dan un resultado muy similar.

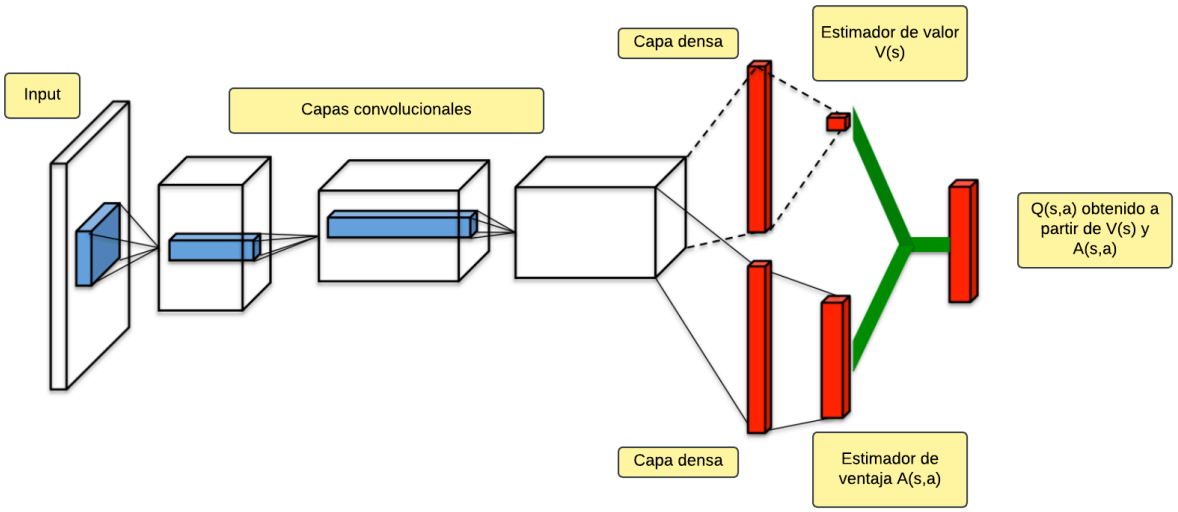

Figura 11. Diagrama de funcionamiento de Dueling Networks, obtenido de [8].

· **Multi-step Learning:** DQN solo toma en cuenta la recompensa inmediata, pudiendo demorar el aprendizaje de secuencias más largas, multi-step Learning [9] toma esto en consideración y mejora tomando la suma de recompensas acumuladas en varios pasos antes de actualizar los Q-values, permitiendo propagar los valores hacia atrás en la red, ayudando al agente a aprender mejor y de forma más rápida.

· **Distributional RL:** DQN calcula un único Q-value, ignorando la variabilidad de las recompensas, Distributional RL [10], modela la distribución completa de las recompensas futuras para cada acción en lugar de solo un valor promedio, proporcionando al agente una visión más completa del rango de valores posible para cada acción, mejorando la calidad de la toma de decisiones.

· **Noisy nets:** La estrategia ε-greedy usada en DQN puede no ser del todo óptima en entornos complejos donde la recompensa puede ser escasa o se necesita descubrir alguna acción importante, dado el factor de aleatoriedad que posee, Noisy nets [11] introduce ruidos en los pesos de la red, permitiendo que la exploración se ajuste a la situación actual.

Rainbow DQN aplica todas estas mejoras en un solo algoritmo, permitiendo una mejora sustancial en rendimiento, como se puede ver en la Figura 12, en comparación a otros

algoritmos de aprendizaje por refuerzo, logrando aprender políticas más efectivas y robustas.

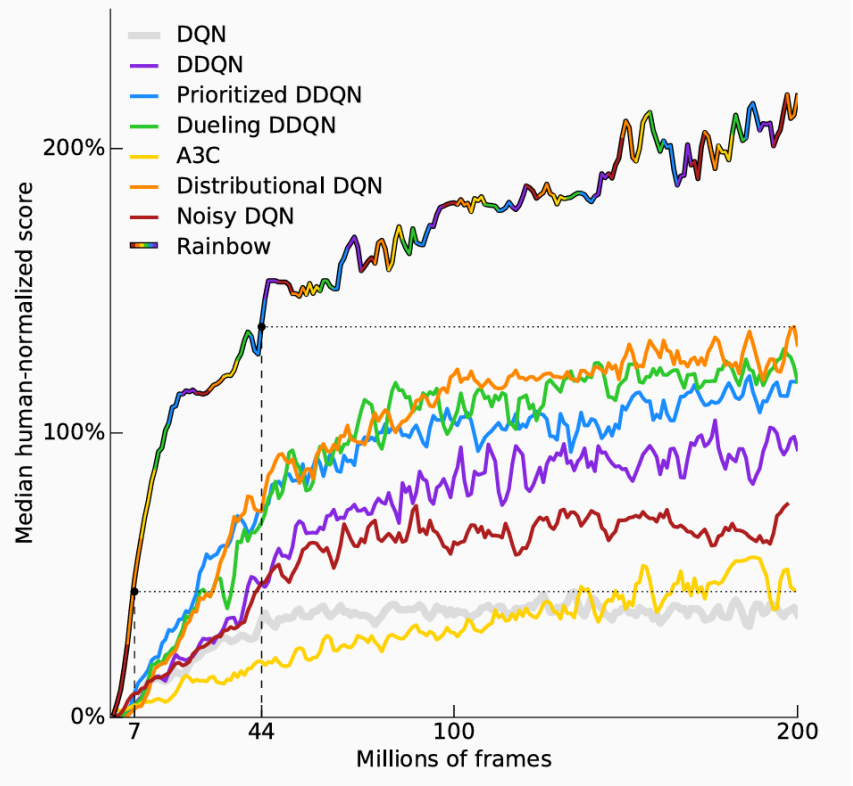

Figura 12. Gráfico comparativo de rendimiento de DQN y sus variantes usando distintos juegos de Atari, obtenido de [5].

2.7 Proximal Policy Optimization

Proximal Policy Optimization (PPO) es un algoritmo basado en políticas, desarrollado por OpenAI [2]. Este algoritmo nace como una mejora a TRPO (Trust Region Policy Optimization), ya que evita el uso de optimizaciones costosas y restricciones complejas en la actualización de la política. En vez de esto, PPO usa una función objetivo con "clipping" que mantiene los cambios de política dentro de un rango seguro, facilitando el entrenamiento y la estabilidad. Este algoritmo se considera actualmente estado del arte en el campo de aprendizaje por refuerzo dada su simplicidad y rendimiento.

Dado que PPO es un algoritmo basado en políticas, este solo optimiza la política que el agente sigue, en vez de aprender Q-values como Q-learning o DQN. La política $\pi_\theta(a|s)$ es la probabilidad de realizar la acción a dado un estado s, esta política se optimiza de manera regresiva gracias a una función de pérdida para evitar grandes cambios de una política a otra, para evitar desestabilizar el entrenamiento, la función de pérdida "clip", se describe como:

$$L^{clip}(\theta) = E_t[min(r_t(\theta)\hat{A}_t,\ clip(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon)\hat{A}_t]$$

Donde:

· $r_t(\theta)$ es la proporción de probabilidades de la nueva política respecto a la anterior, $r_t(\theta) = \frac{\pi_\theta(a_s|s_t)}{\pi_{\theta_{old}}(a_s|s_t)}$.

· $\hat{A}_t$ es la ventaja estimada en tiempo t, que mide que tan buena fue la acción comparada con lo esperado.

· La función clip restringe como se cambia la política. Si el nuevo valor $r_t(\theta)$ se aleja mucho de 1, se recorta a un valor entre $1 - \varepsilon$ y $1 + \varepsilon$, no permitiendo cambios bruscos en la política.

El uso de la función "clip" garantiza que las actualizaciones en la política sean pequeñas, lo que evita cambios repentinos que puedan desestabilizar el entrenamiento del agente,

ayudando a mantener el equilibrio entre exploración y explotación, evitando darles demasiada importancia a acciones recientes y olvide acciones anteriores.

El funcionamiento de PPO (Algoritmo 2) a alto nivel es el siguiente:

1. El agente actúa en el entorno siguiendo la política actual $\pi_\theta(a|s)$, recolectando estados, acciones, recompensas y nuevos estados.
2. Luego se calcula la ventaja $\hat{A}_t$ para cada acción realizada, comparando la recompensa obtenida con el valor esperado del estado.
3. Utilizando la función "clip" se actualiza la política π_θ para maximizar recompensas a largo plazo, asegurando que los cambios no sean drásticos.
4. Se repite el proceso hasta que el agente logre aprender una política que maximice las recompensas en el entorno.

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
  Update the policy by maximizing the PPO-Clip objective:

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

  typically via stochastic gradient ascent with Adam.
  Fit value function by regression on mean-squared error:

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

  typically via some gradient descent algorithm.
end for

Algoritmo 2. Pseudocódigo PPO con función clipping, obtenido de [15].

2.7.1 PPO con Redes neuronales recurrentes

Este algoritmo es una variante de Proximal Policy Optimization, la cual incluye memoria temporal a través de redes recurrentes [33], ya sea Long short-term memory (LSTM) o Gated Recurrent Unit (GRU). Para efectos de esta memoria de título se usarán capas LSTM junto a PPO, dado que es ampliamente utilizada por las herramientas de aprendizaje por refuerzo disponibles.

Las redes recurrentes Long short-term memory (LSTM) [32], están diseñadas para aprender dependencias a largo plazo en secuencias de datos, lo que puede ser útil para entornos parcialmente observables, es decir, entornos donde el agente no puede acceder a toda la información del estado en cada instante del tiempo.

Estas redes LSTM introducen un mecanismo de memoria llamado "celda", que puede mantener información durante largos periodos de tiempo. También utilizan puertas para controlar el flujo de información. Existen tres puertas para controlar el flujo de información en una unidad LSTM (Figura 13), las cuales son:

· Puerta de olvido, la cual decide qué información obtenida previamente se descarta para dar paso a nueva información.

· Puerta de entrada, determina la información nueva que se almacenará en la unidad.

· Puerta de salida, selecciona qué parte de la memoria se enviará a la salida de la unidad.

Cada unidad de LSTM mantiene un estado oculto (Figura 13), el cual es una representación interna de corto plazo que resume la información procesada hasta el momento. Este estado oculto se actualiza a cada paso de tiempo y se usa como salida inmediata de la celda, además de entrada para los siguientes pasos temporales, permitiendo que la red mantenga y actualice información relevante en cada momento, adaptándose a nuevos cambios.

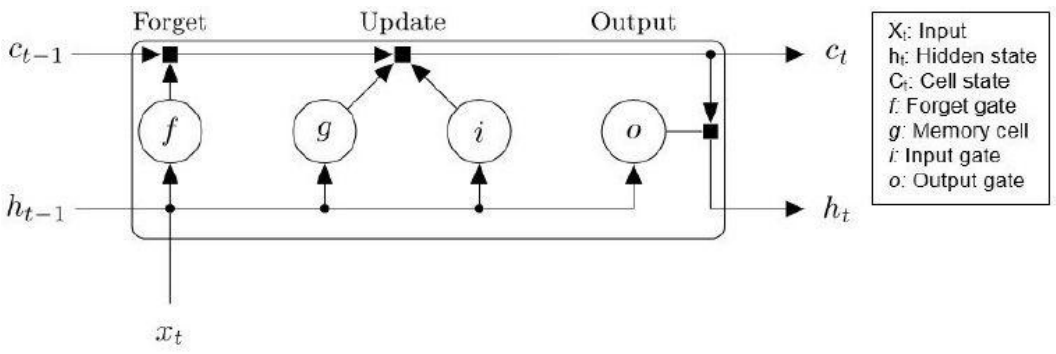

Figura 13. Unidad básica de LSTM, incluyendo las puertas de olvido, entrada o de actualización, y la puerta de salida, así como el estado oculto. Obtenido de [34].

Ahora, PPO usa LSTM de la siguiente forma:

1. Se utiliza una capa LSTM como núcleo central de la política y de la red de valor, esta capa se utiliza entre el extractor de características y las salidas de la red, tanto la de política como la de valor, permitiendo mantener un estado oculto, que se actualiza a cada paso de tiempo.
2. Se utiliza un buffer distinto a PPO convencional, el cual almacena secuencias completas respetando el orden temporal, además, al iniciar una nueva secuencia, se usan máscaras para reiniciar el estado oculto (esto ocurre por ejemplo, al reiniciar un episodio).
3. Durante el entrenamiento, se procesan los mini-batches de subsecuencias usando retropropagación a través del tiempo (BPTT).

2.8 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) es un algoritmo evolutivo, que combina el aprendizaje de redes neuronales con la evolución genética, este algoritmo fue propuesto por Kenneth Stanley [3], y su principal innovación fue que no solo entrena los pesos de las conexiones neuronales, sino que también evoluciona la estructura de las redes a lo largo del tiempo, permitiendo explorar soluciones más complejas durante el proceso evolutivo.

El funcionamiento general de este algoritmo consiste de tres elementos fundamentales, tales como:

· **Codificación genética:** Cada red neuronal es representada por un genoma, el cual contiene información sobre los nodos y las conexiones entre sí. Estas redes comienzan con estructuras simples, y a medida que avanzan las generaciones se van complejizando, añadiendo nuevos nodos y conexiones mediante mutaciones genéticas.

· **Evolución de topologías:** La estructura de la red neuronal puede cambiar según el avance del proceso evolutivo, permitiendo explorar distintas soluciones y descubrir tanto los pesos óptimos para las conexiones de nodos, como una estructura de red óptima.

· **Especiación:** Las redes neuronales se dividen en especies las cuales compiten entre sí, lo que permite que redes con estructuras diferentes puedan evolucionar sin ser demasiado penalizadas por ser demasiado distintas.

NEAT (Figura 14) se inicializa como con una generación de redes neuronales muy simple, con una capa de entrada y otra de salida, estas se evalúan en un entorno dado y se les asigna una puntuación según su rendimiento, las con mayor puntaje se seleccionan para reproducirse y generar la siguiente generación, mientras que las con menor puntaje son eliminadas. Durante la reproducción se realizan mutaciones, las cuales pueden alterar tanto los pesos de las conexiones, como la estructura de estas redes añadiendo nuevas conexiones o nodos. También, durante este proceso de reproducción existe la posibilidad de que se junten dos redes de la misma especie para hacer “crossover” y crear una nueva red combinando características de las redes progenitoras.Finalmente se realiza la especiación, separando a las redes en especies para que exploren diferentes configuraciones sin ser muy penalizadas en su puntaje.

El ciclo de evaluación, selección, reproducción y mutación se repite por cada nueva generación, logrando así que la evolución de redes sean más complejas y eficientes.

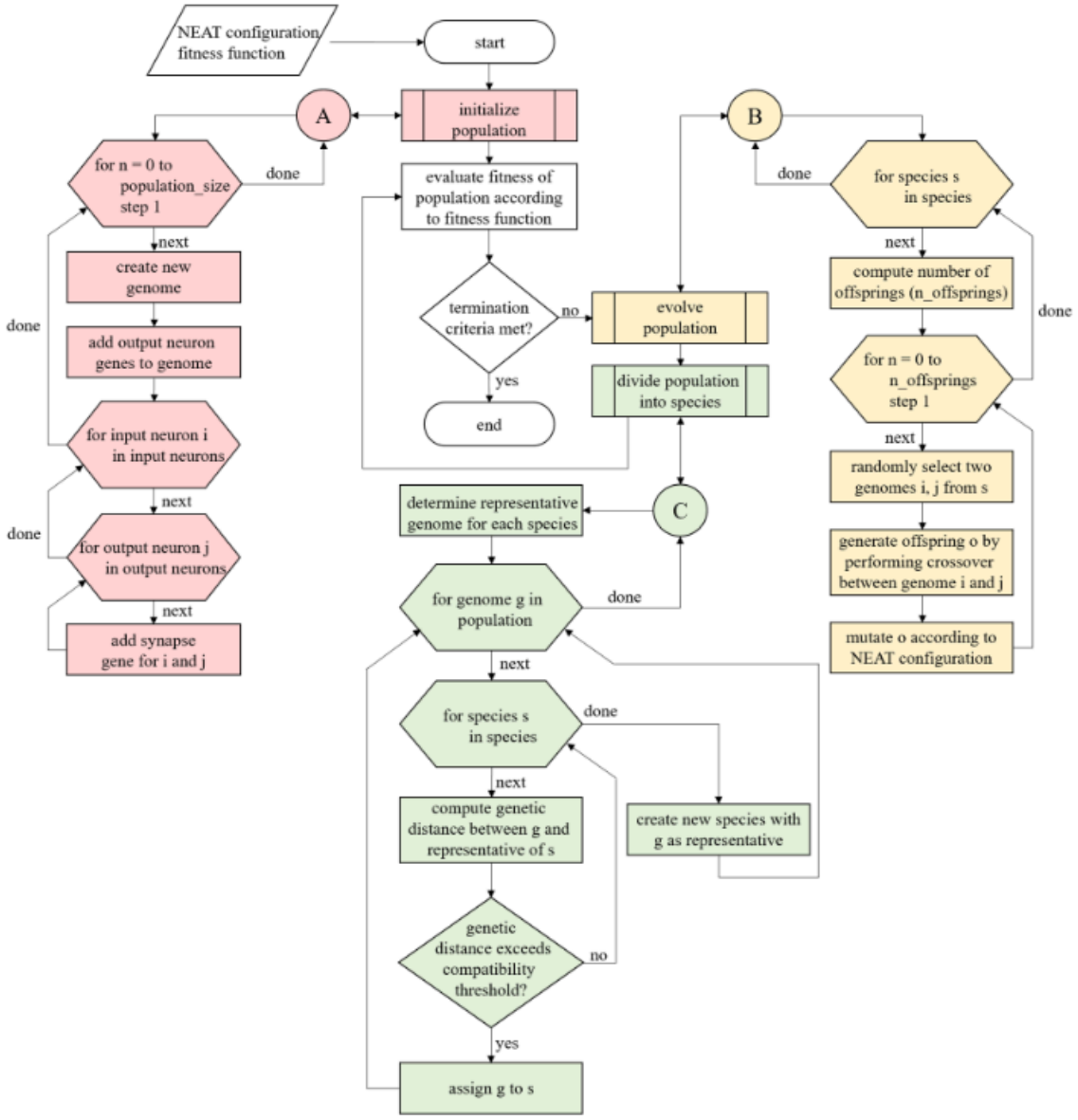

Figura 14. Diagrama de funcionamiento del algoritmo NEAT, obtenido de [4].

2.8.1 HyperNEAT

Hypercube-based NeuroEvolution of Augmenting Topologies [13] es una variante de NEAT, la cual introduce una nueva codificación de forma indirecta utilizando Compositional Pattern Producing Networks (CPPNs), a diferencia del NEAT convencional donde se usaba una codificación directa mediante la topología de la red y sus pesos correspondientes.

Un CPPN es una red neuronal que genera patrones espaciales regulares, tomando como entrada las coordenadas de dos puntos en un espacio bidimensional, luego utilizando una función define los pesos entre los nodos, generando un patrón especial que muestra patrones o simetrías presentes en el entorno donde se está ejecutando.

El proceso de generación de pesos (Figura 15) es el siguiente:

1. Se define un sustrato, que es una cuadrícula que representa los nodos de la red neuronal en un espacio bidimensional.
2. Para cada conexión potencial entre nodos en el sustrato, se evalúan las coordenadas espaciales de los puntos de origen y destino.
3. Estas coordenadas se introducen en el CPPN, que calcula y genera el peso correspondiente basado en las relaciones espaciales definidas por sus funciones internas.

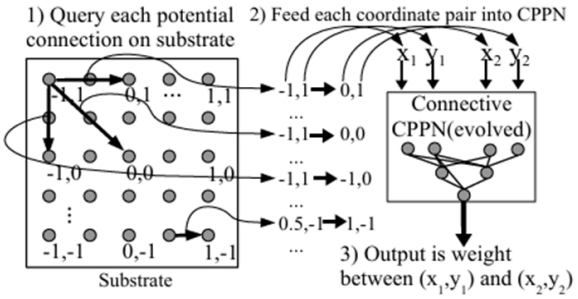

Figura 15. Proceso de generación de pesos, obtenido de [13].

Los CPPNs utilizan distintas funciones para calcular los patrones, ya sea funciones lineales, gaussianas o senoides con la finalidad de darle más complejidad y diversidad a los patrones generados.

2.9 Resumen del capítulo

En este capítulo se han visto los fundamentos teóricos necesarios para el desarrollo de la presente memoria de título, explicando las bases del Machine Learning y el aprendizaje por refuerzo, así como la manera de describir formalmente problemas donde la generalización es clave, usando el CMDP. Además se explican algoritmos clave del aprendizaje por refuerzo como el Q-learning, algoritmo basado en valores, y REINFORCE, algoritmo basado en políticas, los cuales son las bases para los algoritmos con los cuales se trabaja durante el desarrollo de la memoria de título. Además, se explica el funcionamiento de las redes neuronales profundas, ampliamente utilizadas en el Machine Learning, las cuales ayudan a los agentes de aprendizaje por refuerzo a aprender cómo interactuar con el entorno en el que se desenvuelve.

Se enseña el funcionamiento de los algoritmos con los que se trabaja en esta memoria de título, como DQN, algoritmo off-policy basado en valores, el cual se cimenta a partir de Q-learning, y su variante Rainbow, la cual implementar mejoras en sus componentes para mejorar su desempeño considerablemente. Además, se revisa el algoritmo on-policy PPO, basado en políticas el cual usa un “clipping” para evitar cambios bruscos en su política, y su variante PPO con redes recurrentes, la cual memoriza patrones y secuencias temporales, útil para entornos parcialmente observables. Además se revisa un algoritmo con enfoque neuroevolutivo, NEAT y su variante HyperNEAT, los cuales adaptan los pesos y la topología de red de sus individuos, ayudando a la exploración y a la diversidad de soluciones.

Estos algoritmos fueron seleccionados ya que cubren distintos enfoques, basado en valores con DQN y Rainbow, basado en política con PPO y PPO con redes recurrentes, y neuroevolutivo con NEAT e HyperNEAT, lo que permite compararlos de forma complementaria, facilitando la identificación de fortalezas y debilidades de cada enfoque.

3. Generalización en aprendizaje por refuerzo

En esta sección se presentan los métodos seleccionados con el propósito de mejorar la capacidad de generalización de los algoritmos vistos previamente, estos métodos fueron seleccionados de manera que se puedan integrar de forma conjunta sin conflictos, con la finalidad de poder medir el impacto en la generalización de cada método.

Los métodos seleccionados son “Domain Randomization”, la arquitectura de redes neuronales “IMPALA” y el método de exploración “Explore Go”. Antes de ver los métodos para la generalización, estos se categorizan [53] según lo siguiente:

· **Generalización algorítmica:** Modifica directamente el algoritmo de entrenamiento, ya sea con cambios en las reglas de actualización del agente, usando regularizadores, etc. Un ejemplo de esto sería el uso de batch normalization o dropout, comúnmente usado en el aprendizaje supervisado.

· **Generalización a través de la recompensa:** Modifica las recompensas obtenidas en el entrenamiento para fomentar distintos comportamientos. Ejemplos de esto son las recompensas intrínsecas o escalamiento de recompensas.

· **Generalización a través de las observaciones:** Transforma los estados observados con modificaciones aleatorias, ya se cortes de las observaciones, cambios en los colores, rotaciones u otros.

· **Generalización a través de dinámicas del entorno:** Se altera la dinámica del entorno, modificando las probabilidades de transición para exponer al agente a variaciones y hacerlo más robusto. Por ejemplo, variar la gravedad en los saltos en juegos como Super Mario o Sonic.

· **Generalización a través de la política:** Modifica la política usada para elegir acciones, por lo general durante la exploración. Ejemplos de esto son ε-greedy o Noisy Nets.

Esta memoria de título se enfocará en la generalización a través de las dinámicas del entorno, generalización algorítmica y a través de la política.

3.1 Data augmentation

Cuando se trabaja con un conjunto de datos de entrenamiento relativamente pequeños, los agentes entrenados tienen bastantes dificultades para generalizar a nuevos datos no vistos previamente, esto es debido a que el agente se sobre ajusta a lo aprendido durante el entrenamiento, sin aprender patrones generales. Para aliviar este problema se plantea el Data augmentation [29], el cual es un método para la generalización a través de observaciones, consiste en utilizar diversos métodos para aumentar artificialmente el volumen de datos de entrenamiento, como por ejemplo, en un entorno donde se trabaje con imágenes, recortarlas, añadir ruido, rotarlas, etc (Figura 16). También existen otras formas, de manera más implícita, tales como introducir factores aleatorios en el entorno de prueba, o forzar al agente a explorar más el entorno.

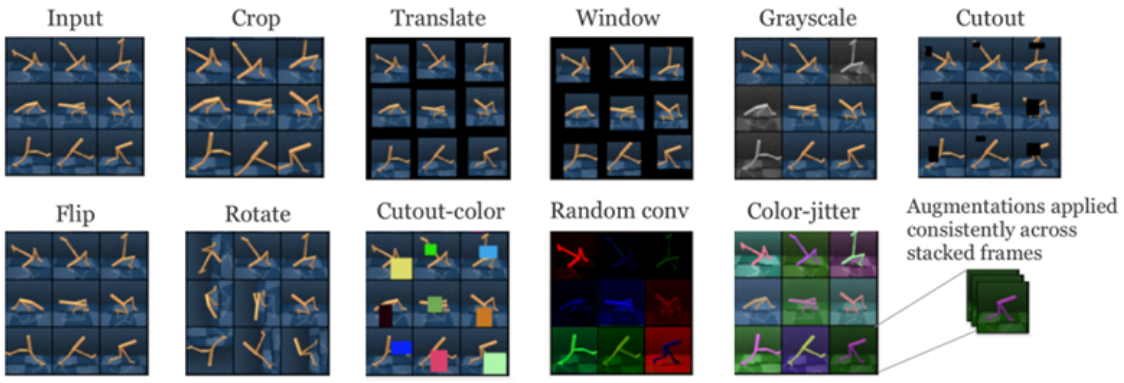

Figura 16. Tipos de data augmentation para imágenes, obtenido de [29].

3.1.1 Domain Randomization

Es un método para generalización a través de las dinámicas del entorno, utilizado para generar "datos sintéticos" sobre el entorno y así entrenar en una mayor cantidad de datos, para así mejorar la capacidad de generalización en agentes de aprendizaje por refuerzo [49]. La idea principal es aleatorizar ciertos aspectos del entorno de entrenamiento, ya sea la apariencia visual, físicas del entorno u otros, de forma que el agente aprenda una política robusta que no dependa de una configuración específica del entorno, para que así el entorno de prueba parezca simplemente una variación más de los entornos de prueba (Figura 17).

Este método busca atacar el problema del sobreajuste al entorno del entrenamiento, donde el agente aprende un comportamiento muy definido para un entorno en específico, pero baja notoriamente su rendimiento ante ligeras variaciones. Para evitar esto, se expone al agente a múltiples entornos con variaciones aleatorias, obligándolo a identificar patrones que sean invariantes a las variaciones del entorno.

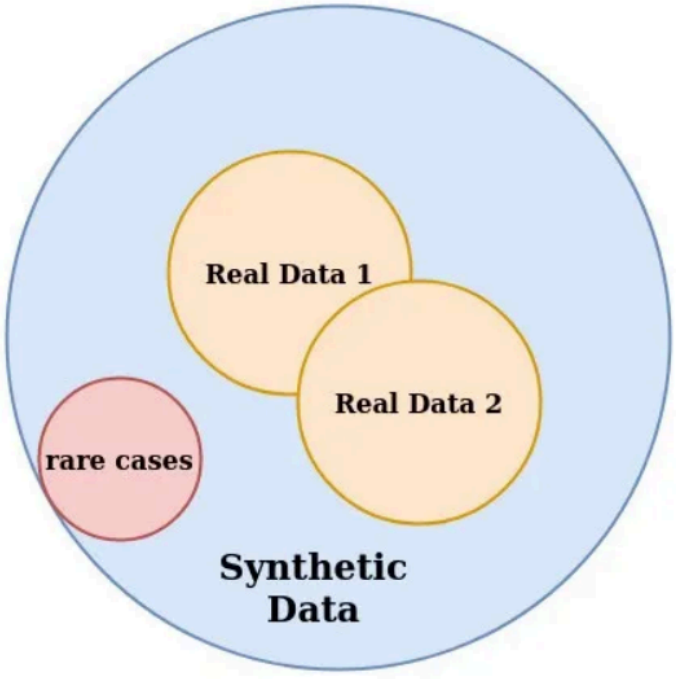

Figura 17. Imagen representativa del objetivo de Domain randomization, generar datos sintéticos que cubren un gran número de casos, con la probabilidad de encontrar variaciones muy similares o iguales a casos reales. Obtenido de [35].

3.2 Arquitectura de red neuronal IMPALA

Como se ha mencionado anteriormente, las redes neuronales son fundamentales para el funcionamiento de los algoritmos de aprendizaje por refuerzo, ya que permiten representar políticas óptimas, funciones de valor e identificar patrones clave dentro de los entornos donde se trabaja. En trabajos previos [37] se ha demostrado que arquitecturas de redes neuronales pequeñas (por ejemplo, la arquitectura usada en el paper Nature [1]), pueden no ser suficientes para tareas con una mayor complejidad o diversidad (Figura 18), teniendo un rendimiento y una capacidad de generalización menor. Por esto, para problemas más complejos se opta por usar una arquitectura de red neuronal basada en el trabajo hecho por DeepMind con IMPALA (Importance Weighted Actor-Learner Architecture), como método de generalización algorítmica [38].

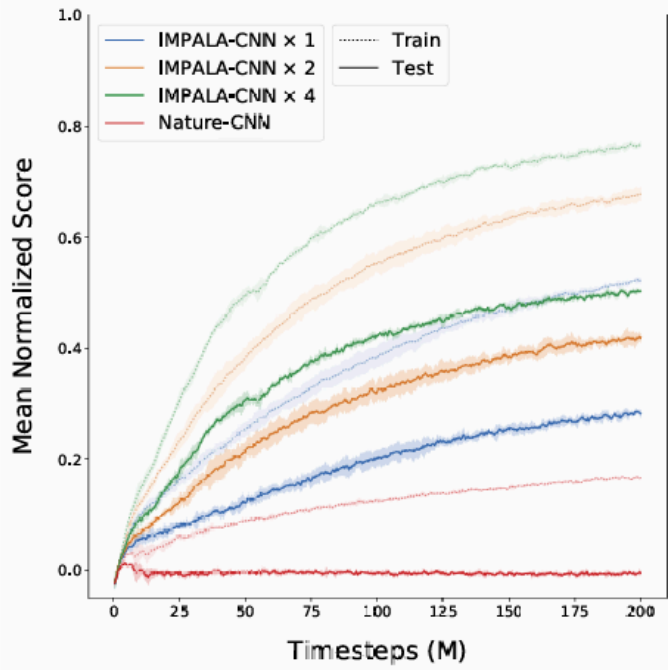

Figura 18. Gráfica comparativa entre la red propuesta en Nature y la red convolucional IMPALA, se prueban múltiples versiones de IMPALA escalando el número de canales por 1, 2 o 4, aumentando el desempeño a la vez que el costo computacional. Obtenido de [37].

IMPALA usa una arquitectura que combina redes convolucionales y bloques residuales [39], y si bien en la arquitectura presentada en IMPALA se ocupa una red LSTM, en esta implementación se optó por no usarla, ya que la información temporal se maneja con un stack de frames.

La secuencia de redes IMPALA es la siguiente (Figura 19):

1. Primero entra a una red convolucional con tamaño de kernel de 3x3, y stride de 1.
2. Luego se hace Max Pooling con tamaño de kernel de 3x3, stride de 2.
3. Luego pasa por dos bloques residuales, consistentes de dos capas convolucionales con tamaño de kernel de 3x3, con stride de 1, y luego de cada convolución se usa una función de activación ReLU, y luego se suman los tensores previos a las operaciones convolucionales al tensor resultante de las operaciones.
4. Finalmente se pasa por una capa completamente conectada, pero antes y después de esto se aplica función de activación ReLU.

A diferencia de la red neuronal Nature, la cual usa únicamente tres operaciones convolucionales seguida de una capa completamente conectada, la arquitectura de IMPALA emplea una red más profunda, compuesta por múltiples capas convolucionales y bloques residuales. Esta diferencia permite a IMPALA extraer características de mayor nivel y capturar estructuras espaciales más complejas en las observaciones visuales, lo que se traduce en un mejor desempeño en tareas de mayor complejidad.

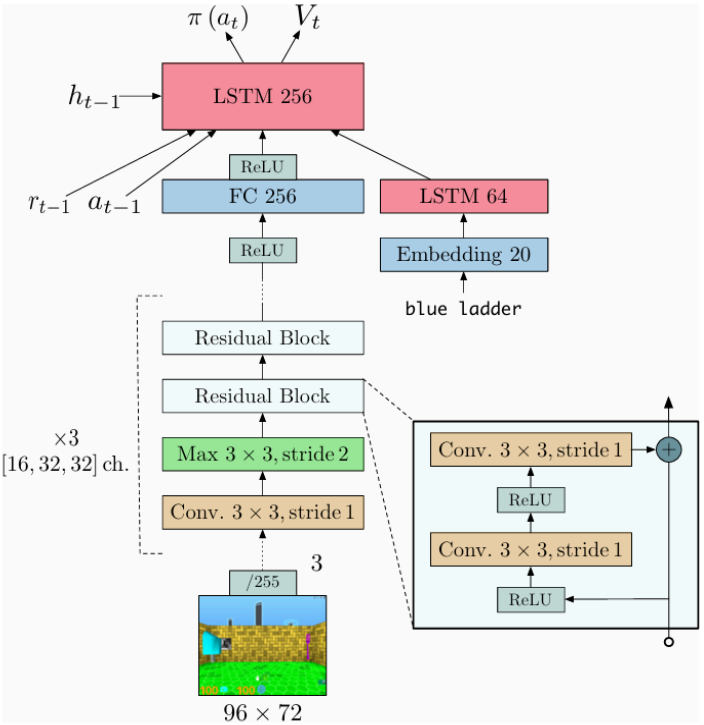

Figura 19. Modelo de arquitectura de red IMPALA, obtenido de [38].

3.3 Explore Go

La exploración ha demostrado ser muy valiosa para mejorar la generalización a entornos nunca vistos durante el entrenamiento, llegando a proponer métodos de exploración continua [31], pero si esto se realiza de sobremanera, puede perjudicar los valores de estimación del agente, deteriorando el rendimiento. Para esto, se propone como solución el método Explore Go [30], el cual ayuda a la generalización a través de la política (Algoritmo 3), donde se introduce una fase de exploración inicial durante un número aleatorio entre 0 y *k* pasos, en la cual se ignora por completo la señal de recompensa recibida del entorno, para luego dejar actuar al agente principal o de "explotación". El objetivo de Explore Go es aumentar artificialmente la diversidad de estados iniciales disponibles para el agente principal, incrementando así la cobertura del espacio de estados alcanzables durante el entrenamiento.

A diferencia de los métodos de exploración continua, este método introduce una breve fase de exploración en cada episodio, entrenando solo con los datos posteriores a la fase de exploración, evitando errores inducidos de transiciones de exploración, permitiendo aumentar la diversidad de estados iniciales sin comprometer la precisión del aprendizaje, actuando de cierta forma como un "data augmentation" implícito, además de ser fácilmente aplicable tanto a algoritmos on-policy como off-policy.

Algorithm 1: Generic CollectRollouts + Explore-Go

Input: number of steps to collect N, pure exploration policy π_{PE}, max number of pure exploration steps K

$k \leftarrow Uniform(0, K)$;
$\mathcal{D}_{rollout} \leftarrow \{\}$;
$num\_steps\_collected \leftarrow 0$;
while $num\_steps\_collected < N$ **do**
  if $episode\_step < k$ **then**
    Sample transition t using π_{PE};
  else
    Sample transition t with base RL algorithm;
    Add t to $\mathcal{D}_{rollout}$;
    $num\_steps\_collected$ += 1;
  end if
  $episode\_step$ += 1;
  if *end of episode* **then**
    $k \leftarrow Uniform(0, K)$;
    $episode\_step \leftarrow 0$;
    Reset environment;
  end if
end
Return $\mathcal{D}_{rollout}$;

Algoritmo 3. Pseudocódigo oficial de Explore Go, obtenido de [30].

3.4 Resumen del capítulo

En este capítulo se ha visto la generalización en el aprendizaje por refuerzo, las formas de obtener una mejor generalización, ya sea a través de la recompensa, de las observaciones, a través de las dinámicas del entorno, a través de la política o generalización algorítmica.

Se ha visto el Data augmentation, el cual es un método de la generalización a través de observaciones, el cual también es usado como un concepto importante dentro de la generalización, sobre el cual se cimentan muchos métodos de generalización. Data augmentation se refiere al aumento de forma artificial del volumen de datos disponibles, lo cual expone al agente a diversas situaciones y estados distintos, favoreciendo la generalización.

También se explican los métodos a desarrollar durante la memoria de título, siendo estos Domain Randomization, el cual consiste en aleatorizar ciertos elementos dentro del entorno con la finalidad de aumentar el volumen de datos disponible para entrenar. También se enseña la arquitectura de redes neuronales IMPALA, la cual mejora la representación de características usando bloques residuales. Y el método Explore Go, el cual añade una fase de exploración inicial al inicio de cada episodio con la finalidad de aumentar la cantidad de estados visitados por el agente.

Estos métodos son seleccionados porque resultan complementarios y compatibles con los algoritmos de aprendizaje por refuerzo seleccionados. Cada uno presenta un enfoque diferente y afecta una parte distinta del entrenamiento de los agentes, como la interacción con el entorno, la exploración o la actualización de la política, sin generar conflictos entre los métodos. En conjunto, estos métodos permiten abordar el problema de la generalización con una visión general, cubriendo 3 de los 5 enfoques con los que lograr la generalización.

4. Metodología

En esta sección se presenta la metodología utilizada para el desarrollo de la memoria de título. El enfoque de este trabajo es experimental, centrado en la implementación y análisis comparativo de métodos que mejoran la generalización de los algoritmos previamente descritos, además del proceso de entrenamiento y evaluación de los agentes obtenidos.

Se detallan los entornos a utilizar durante las fases de entrenamiento y evaluación, las herramientas utilizadas para la implementación de los algoritmos y sus variantes seleccionadas. Además se muestran detalles de implementación sobre los métodos de mejora de generalización descritos previamente, así como los criterios de evaluación de los agentes obtenidos en entornos no vistos durante el entrenamiento.

4.1 Herramientas de trabajo

Para el desarrollo de la presente memoria de título se han utilizado las siguientes herramientas tanto de software como de hardware:

· **Gym:** Paquete de Python ampliamente usado en el campo del aprendizaje por refuerzo [44], propuesto inicialmente por OpenAI, pero mantenido actualmente por Farama Foundation. Este paquete brinda una interfaz en la cual crear, entrenar y evaluar algoritmos de RL en entornos controlados. En esta ocasión por temas de compatibilidad con otros paquetes, se utiliza la versión 0.21.0.

· **Gym-super-mario-bros:** Paquete de Python el cual provee una ROM del videojuego Super Mario Bros original para la NES [45], contiene los distintos niveles del juego y se usa como entorno de prueba para los algoritmos seleccionados. Se utiliza la última versión de este paquete 7.4.0 la cual incluye una función donde seleccionar niveles aleatorios en cada episodio.

· **Stable-baselines3 y sb3-contrib:** Biblioteca la cual contiene implementaciones eficientes y fáciles de usar de PPO, DQN entre otros [46]. Por otro lado, sb3-contrib posee implementaciones de PPO con redes recurrentes y QR-DQN, una variante de DQN la cual incluye regresión de cuantiles, el cual es usado como parte de la implementación del agente Rainbow detallado posteriormente. Se utiliza la versión 1.7.0 de esta librería ya que es de las

últimas versiones compatibles con Gym, versiones posteriores no son compatibles con gym-super-mario-bros.

· **Neat-python y PyTorch-NEAT:** Biblioteca que implementa el algoritmo NEAT [47] y otorga una forma sencilla de configurar pruebas mediante un archivo config determinando parámetros como los criterios del fitness, funciones de activación o biases. Además, se utiliza PyTorch-NEAT [48] para el agente HyperNEAT, el cual funciona en base a Neat-python, por lo que solo hay que definir un archivo config similar en Neat-python y un sustrato para generar las conexiones de las CPPNs.

Se opta por usar implementaciones hechas de bibliotecas externas, ya que resultan confiables al ser usadas por una gran parte de la comunidad, evitando posibles errores sutiles derivados de realizar implementaciones propias. Además estas implementaciones están optimizadas para tener un alto rendimiento computacional, permitiendo entrenar modelos de forma más rápida y sencilla. Esto permite enfocar la investigación al ámbito teórico y experimental, sin desviar tiempo ni recursos en detalles de implementación de algoritmos.

En cuanto al hardware utilizado para el entrenamiento de los agentes, se utiliza:

· **CPU:** Intel Core i5 13600k.

· **RAM:** 64 GB RAM DDR5.

· **GPU:** Geforce RTX 4080.

· **SO:** Ubuntu 22.04.

Cabe añadir el uso de LLMs como ChatGPT para el desarrollo de la memoria de título, principalmente como ayuda para la redacción del presente informe y para la búsqueda de artículos académicos. Además de parcialmente para generar código, en este caso, se usó para empezar el desarrollo de algún método, nunca para hacerlo en su completitud con ChatGPT. Toda la información fue contrastada con fuentes confiables presentes en su gran mayoría en las referencias bibliográficas.

4.2 Descripción del entorno

El entorno seleccionado para entrenar al agente es el videojuego Super Mario Bros desarrollado por Nintendo, utilizando el paquete gym-super-mario-bros [45] el cual otorga una interfaz compatible con Gym conteniendo los 32 niveles del juego original.

Se opta por usar este entorno debido a su amplio uso en el campo de investigación en el aprendizaje por refuerzo, además de su alta complejidad proveniente de la exploración y el uso preciso de los controles del juego.

· **4.2.1 Función de recompensa:** La función de recompensa por defecto de entorno asume que el objetivo del juego es avanzar a la derecha lo más rápido posible sin morir. Considerando esto, la función de recompensa se compone de tres variables principales:

$$r = v + c + d$$

Donde:

· v es la diferencia en la posición horizontal del agente entre dos estados, en caso de moverse a la derecha, $v > 0$, en caso contrario $v < 0$, y en caso de no moverse $v = 0$.

· c es una penalización por tiempo, en caso de haber un tick en el reloj del entorno $c < 0$, en caso contrario $c = 0$.

· d es una penalización en caso de que el agente muera dentro de un episodio, en caso de morir $d = 15$, en caso contrario $d = 0$.

· r es la suma de todas las variables y representa la recompensa dada por una acción dentro del entorno.

A esta función de recompensa se le añaden dos nuevas variables s y g, para prevenir el estancamiento del agente y para fomentar el completar niveles. Obteniendo lo siguiente:

$$r = v + c + d + s + g$$

Donde:

· s representa el estancamiento del agente, en caso de no estar estancado $s = 0$, en caso de estancarse $s = -0.2$, y en caso de estar más de 20 acciones seguidas estancado en la posición x, y luego consigue avanzar, se le da una recompensa al agente $s = 1$.

· g es una recompensa en caso de que el agente logre completar un nivel, en caso de completar un nivel $g = 50$, en caso contrario $g = 0$.

· **4.2.2 Preprocesamiento de imágenes:** Las observaciones recibidas desde el entorno se usan para alimentar las redes neuronales usadas en los algoritmos PPO y DQN, estas suelen tener demasiada información innecesaria para entrenar la red.

Con la finalidad de alivianar la complejidad de cómputo y facilitar el entrenamiento de la red se aplican los siguientes pre-procesamiento (Figura 20) a las observaciones:

· **Frame-skipping:** Se usa para realizar la misma acción durante una cantidad de frames n consecutivos (en este caso n=4), con la finalidad de eliminar información redundante y reducir la cantidad de toma de decisiones del agente.

· **Reducción de imágenes y escala de grises:** Por defecto este entorno devuelve observaciones de 240 px de alto, 256 px de ancho y 3 canales rgb, lo cual puede ser demandante en cómputo, por esto, se reducen las dimensiones de las observaciones a 84px de alto y 84 de ancho para reducir costes de cómputo sin perder información útil para el agente. Además, las imágenes se convierten a un solo canal para eliminar información de color irrelevante, ya que la estructura visual es más importante que los colores en este entorno, eliminando posibles sobreajustes a colores específicos.

· **Framestacking:** Se apilan frames en grupos para capturar la información temporal del entorno, como la velocidad y dirección del agente o los enemigos.

Todas estas modificaciones en las observaciones son utilizadas en todos los experimentos a no ser que se especifique lo contrario.

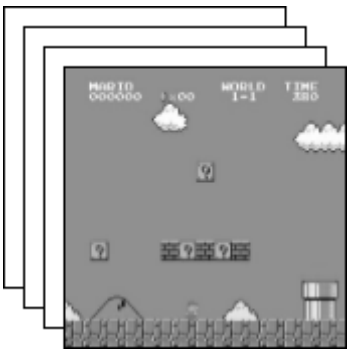

Figura 20. Observaciones del entorno post pre-procesamiento, obtenido de docs.pytorch.org.

Este entorno contiene 4 tipos de ROM a cargar para el entorno (Figura 21), donde cada una modifica la información visual, disminuyendo el detalle de los elementos del fondo, enemigos, obstáculos o el propio Mario en pantalla (Figura 21). Se opta por usar la versión v1, "downsample", la cual reduce información innecesaria del fondo de cada nivel.

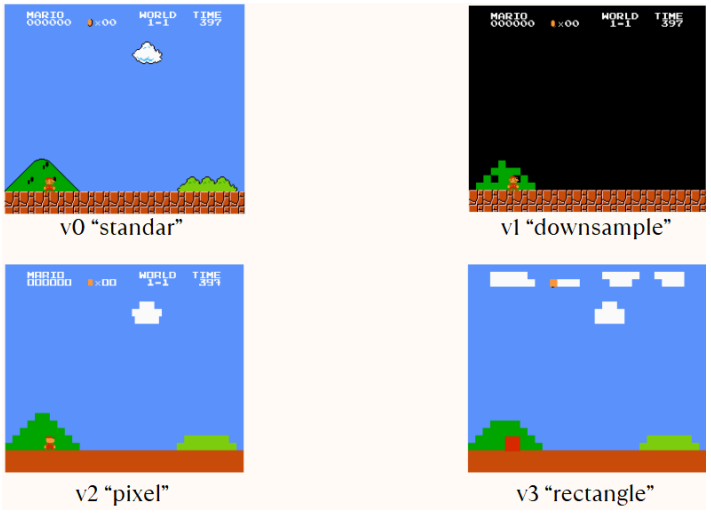

Figura 21. Todas las versiones de ROM disponibles en gym-super-mario-bros, obtenido de [45].

· **Acciones discretas:** El entorno de Super Mario por defecto tiene una acción posible para cada botón del control de juego, como la cruceta de direcciones o el botón A o B, esto puede resultar en más de 256 combinaciones posibles, y entrenar sobre tantas combinaciones resulta ineficiente, por lo que se opta por transformar el espacio de acciones a acciones discretas, restringiendo las acciones a una lista de combinaciones de botones útiles para el agente. En este caso se ocupa la lista de acciones de acciones discretas "SIMPLE MOVEMENT", la cual permite las siguientes combinaciones:

[['NOOP'], ['right'], ['right', 'A'], ['right', 'B'], ['right', 'A', 'B'], ['A'], ['left']]

4.3 Implementación de algoritmos

Se ocupan implementaciones hechas de las bibliotecas mencionadas en las Herramientas de trabajo tanto para DQN, PPO, PPO recurrente con LSTM, NEAT e HyperNEAT.

· **Rainbow:** Se ha implementado en base a la implementación de DQN con regresión de cuantiles de la biblioteca sb3-contrib, es decir, se han implementado las mejoras propias de Rainbow siguiendo la estructura de los demás algoritmos de la biblioteca de Stable-baselines3 (Figura 22), usando como referencia el repositorio "Rainbow is all you need" [50].

Para crear la estructura de redes usada por Rainbow, es decir, las redes Noisy combinadas con redes Dueling, se crea una clase llamada "RainbowNet", la cual se encarga de definir las redes de ventaja y valor, cuyas capas ocultas son Noisy nets con ruido gaussiano factorizado. Estas redes se ocupan si al momento de inicializar el agente se pasan los parámetros booleanos dueling y noisy como True, en caso contrario, usa las redes de quantiles por defecto. La clase "RainbowPolicy" es la encargada de construir las instancias de "RainbowNet", creando la red objetivo y la principal.

Para la implementación de PER se crea la clase "PER", la cual hereda de la clase "ReplayBuffer" de stable-baselines3. Se utilizan árboles binarios para almacenar y acceder de forma eficiente tanto a las sumas de las prioridades como los valores mínimos de prioridad en el buffer. Se añade n-steps return utilizando una deque almacenando los últimos n pasos realizados, una vez se acumulan los pasos suficientes, se calcula una recompensa acumulada y se guarda la transición en el buffer.

Finalmente, la clase "Rainbow" es la que implementa todo el algoritmo Rainbow completo, en el método "train" de esta clase es donde el agente se entrena, hace un reinicio del ruido de las Noisy Nets, y se implementa el Double Q-learning, utilizando la red principal para seleccionar la acción y evaluarla usando la red objetivo, además se calculan las diferencias temporales de las transiciones, para actualizar las prioridades de las transiciones en el PER. El código asociado a Rainbow está disponible en repositorio de GitHub.

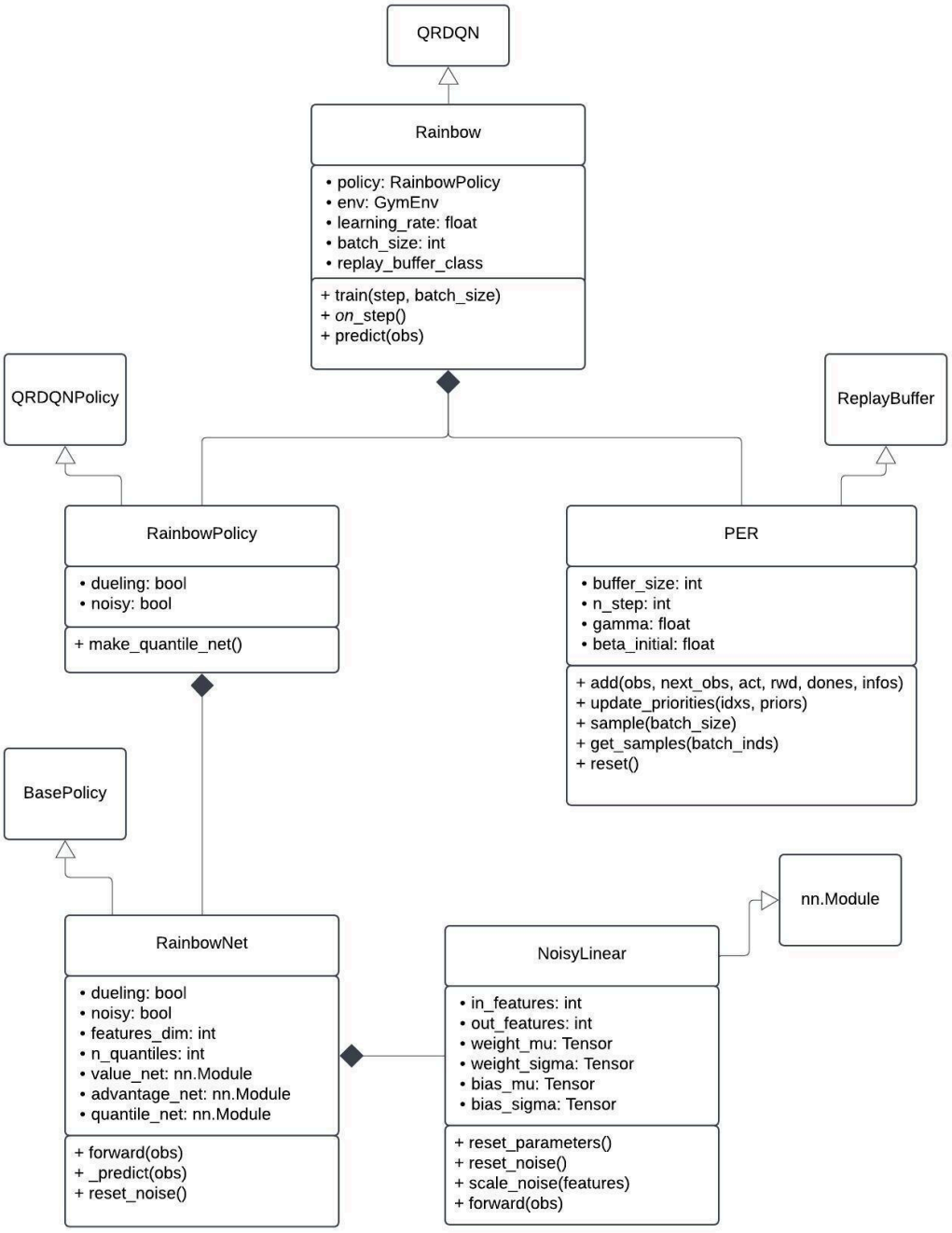

Figura 22. Diagrama UML de agente Rainbow.

· **Sobre NEAT e HyperNEAT:** Para los experimentos con NEAT e HyperNEAT, el único preprocesamiento de imagen que se utiliza es el de reducción de imágenes y conversión a escala de grises, para ambos algoritmos se usan observaciones de 32 x 32 px, tratando de mantener un equilibrio entre mantener la información relevante de cada observación y no complejizar la entrada. Además se opta por usar como la función fitness la posición en X del agente sobre el nivel actual, en lugar de la función de recompensa definida anteriormente, dado que esto resulta más general y estable para el aprendizaje de los algoritmos evolutivos. La evaluación de los genomas se realiza seleccionando un nivel aleatorio del set de niveles de entrenamiento y retornando la mayor posición en X alcanzada. Los niveles de entrenamiento y evaluación son detallados en la sección 4.5 Configuración experimental.

En cuanto al sustrato definido para HyperNEAT (Figura 23), se definen las coordenadas de entrada como una grilla de nodos equidistantes de 32 de alto y 32 de ancho, mientras que las coordenadas de salida se define como un vector con tantos nodos equidistantes como acciones posibles tenga el entorno, en este caso dado que se usa "SIMPLE MOVEMENT" mencionado anteriormente, el vector es de largo 7, donde cada nodo de salida tiene asociada una acción posible.

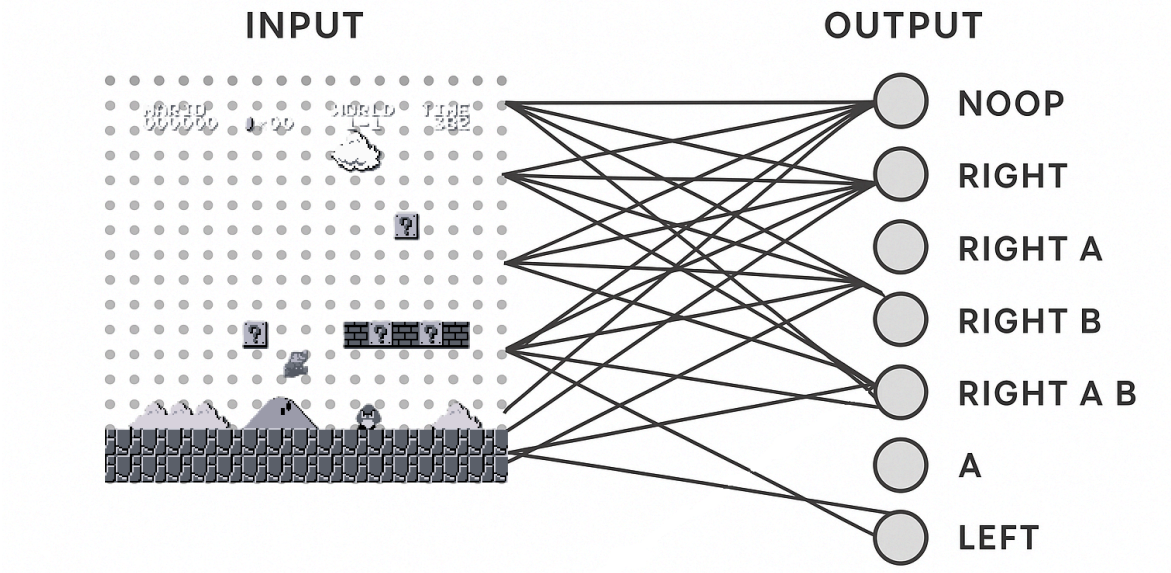

Figura 23. Sustrato de HyperNEAT con entrada visual, imagen generada con LLMs.

4.4 Implementación de métodos de generalización

A continuación se presentan los detalles de implementación de cada método propuesto para mejorar la generalización de los algoritmos.

· **4.4.1 Domain Randomization y estrategias para evitar sobreajuste:** El entorno seleccionado es completamente determinista, por lo que es muy probable que el agente no aprenda políticas generales si no se encuentra con distintas distribuciones de obstáculos y enemigos. Por lo anterior, se opta por usar la función "Random Stages" del entorno, la cual permite al agente jugar en distintos niveles durante el entrenamiento, exponiéndose a una mayor diversidad de estados. Se detallan los niveles seleccionados para entrenamiento y evaluación en la sección de Configuración Experimental.

El entorno brindado por Gym-super-mario-bros expone la memoria RAM del entorno, por lo que utilizando una tabla detallando las direcciones de memoria relevantes, [40] se logra añadir estocasticidad en los enemigos mediante leves modificaciones en su velocidad de desplazamiento (Algoritmo 4), además de la posibilidad de cambiar ciertos enemigos por otros.

Para modificar el entorno sin crear cambios no deseados, romper la lógica del juego, ni volverlo muy difícil (o fácil), se crea una lista de los enemigos los cuales se permiten modificar, y también permitir que estas modificaciones se hagan luego de una cantidad elevada de pasos dentro del nivel.

Se crean dos listas con las direcciones de memoria de enemigos permisibles, una lista con enemigos considerados fáciles y otra lista con enemigos más difíciles, esto con la finalidad de no alterar demasiado la dificultad del nivel actual. Luego de una cantidad fija de pasos dentro del entorno, se determina si hay enemigos en pantalla, en caso de encontrar enemigos permisibles, existe un 50% de probabilidad de cambiar para cada enemigo en pantalla. Si el enemigo permisible está en la lista de enemigos fáciles, se cambia por otro de la misma lista, con una chance de cambiarlo por otro de la lista de enemigos más difíciles. Para enemigos permisibles más difíciles se hace lo mismo, se cambia por uno de la misma lista, con una chance de cambiarlo por un enemigo más fácil.

Algorithm 1 Randomizar Enemigos

```
function RandomizarEnemigo(id_enemigo)
    if id_enemigo ∈ enemigos_faciles then
        return elección aleatoria de enemigos_faciles ∪ primeros 2 de enemigos_medios
    else if id_enemigo ∈ enemigos_medios then
        return elección aleatoria de enemigos_medios ∪ primeros 2 de enemigos_faciles
    end if
end function
function RandomizarEnemigos(env)
    ram ← env.ram
    for i = 0 to 4 do
        activo ← ram["active" + i]
        id_enemigo ← ram["enemy_id" + i]
        if (id_enemigo ∈ enemigos_faciles or id_enemigo ∈ enemigos_medios) and activo = 1 then
            if random() < 0.5 then
                ram["enemy_id" + i] ← RandomizarEnemigo(id_enemigo)
            end if
        end if
    end for
end function
```

Algoritmo 4. Pseudocódigo de función para randomizar enemigos de Super Mario Bros.

Para las modificaciones de velocidad de los enemigos activos en pantalla, se sigue una lógica muy similar. Específicamente, se almacenan los valores de las direcciones de memoria donde se indica si el enemigo está activo en pantalla, su ID específico, y su velocidad, si es un enemigo permisible entonces existe un 50% de posibilidades de sumar un pequeño valor a su velocidad actual. Código asociado a Domain Randomization disponible en un repositorio de GitHub.

· **4.4.2 Arquitectura de redes neuronales IMPALA:** La implementación de la red neuronal IMPALA fue realizada en Pytorch [41], siguiendo lo mencionado en el paper donde se muestra por primera vez [38], a las que se añaden capas de Group Normalization a los bloques residuales y una capa de Dropout antes de pasar a la capa lineal (Anexo B), según lo implementado en una investigación de OpenAI para cuantificar la generalización [42]. Además, se cambia la capa de "Flatten" usada en la implementación tradicional, por una capa de pooling adaptativa (Figura 24) que promedia las características obtenidas por cada canal antes del Dropout según un paper reciente [43], el cual demuestra obtener una mejor generalización en entornos procedurales además de reducir significativamente la cantidad de parámetros utilizados. Código asociado a IMPALA disponible en repositorio de GitHub.

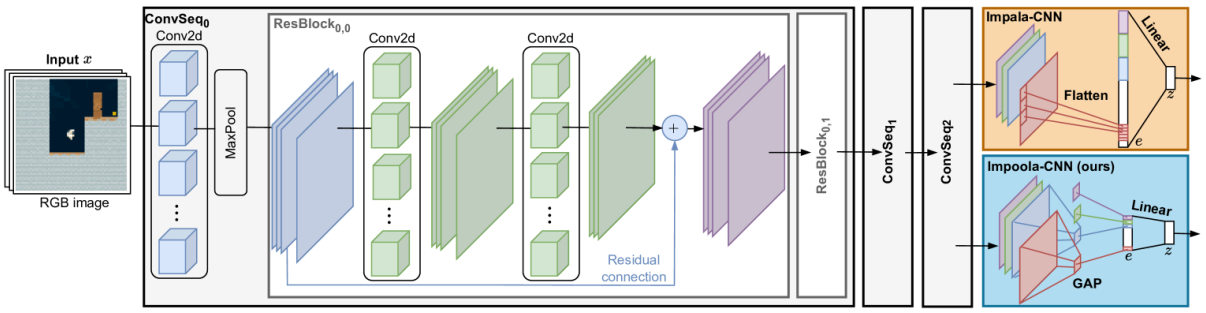

Figura 24. Diferencia entre la red neuronal IMPALA tradicional y la red Impoola, obtenida de [43].

· **4.4.3 Explore Go:** En la implementación oficial de Explore Go se modifica el método de recolección de experiencias para implementar la fase de exploración, como se muestra en el Algoritmo 3. Para implementar este método en el entorno seleccionado surge un problema, dado que la llamada al método de recolección de experiencias no coincide necesariamente con el inicio de cada episodio del agente, lo que resulta en fases exploratorias en medio de un episodio. Por lo que se opta por aplicar la lógica de Explore Go como un modificador de Gym (Algoritmo 5), precisamente modificando el reset del entorno, devolviendo la observación del último paso de exploración para el agente de explotación.

Se implementa un agente de exploración ICM [12] para realizar la fase de exploración (Algoritmo 6), tratando de obtener la mayor cantidad de estados iniciales distintos para el agente de explotación. Código asociado a IMPALA disponible en repositorio de GitHub.

Algorithm 1 `reset` modificado con pasos de exploración

Entrada: Entorno `env`, número máximo de pasos de exploración N, módulo de exploración `explorer` (opcional)
Salida: Observación inicial `obs` luego de la exploración

```
obs ← env.reset()
K ← número aleatorio entre 0 y N
for i = 1 to K do
    if explorer ≠ None then
        action ← explorer.select_action(obs)
    else
        action ← acción aleatoria
    end if
    (next_obs, reward, done, info) ← env.step(action)
    if explorer ≠ None then
        (intrinsic_reward, loss) ← explorer.intrinsic_reward(obs, next_obs, action)
        explorer.update(loss)
    end if
    if done then
        obs ← env.reset()
    else
        obs ← next_obs
    end if
end for
return obs
```

Algoritmo 5. Pseudocódigo de implementación de Explore Go.

El agente de exploración ICM (Intrinsic Curiosity Module) se compone de un extractor de características, el cual consiste de una capa inicial de Max Pooling para reducir las dimensiones de las observaciones recibidas de 84x84 px a 42x42 px, 4 capas convolucionales con 32 canales de entrada y salida, y tamaño de kernel de 3x3, usando función de activación ELU (Exponential Linear Unit) entre cada salida de la capa convolucional.

El modelo inverso, aprende a predecir la acción $\hat{a}_t$ que tomó el agente dado dos estados:

$$\hat{a}_t = f_{\text{inv}}(\phi(s_t), \phi(s_{t+1}))$$

Se compone de una capa lineal que recibe como entrada una concatenación de los estados ya procesados s_t y s_{t+1}. Entre las capas lineales, pasa por una función de activación ReLU, para luego pasar por la última capa lineal que recibe la salida de la capa anterior, y retorna un vector con los valores asociados a cada acción.

El modelo directo, aprende a predecir el siguiente estado $\hat{\phi}(s_{t+1})$ a partir del estado s_t y la acción a_t:

$$\hat{\phi}(s_{t+1}) = f_{\text{fwd}}(\phi(s_t), a_t)$$

Se compone de una capa lineal que recibe de entrada una representación del estado actual concatenada con una codificación de la acción realizada, pasa por una función de activación ReLU, finalmente por otra capa lineal que recibe la salida de la capa anterior y produce un vector de salida con la predicción de estado siguiente.

Para hacer que sea un agente de pura exploración, es decir, que ignora toda recompensa proveniente del entorno, se recibe la observación del entorno y pasa por el extractor de características, luego se crea una copia de la salida por cada acción posible. Además se crea un vector de acciones one-hot, donde cada fila representa una acción distinta. Luego se junta cada vector de características con cada acción posible, y se ocupa como entrada para el modelo directo.

Obtenido una vez la salida del modelo directo, se calcula el error usando una función de pérdida de error cuadrático medio (MSE) entre entre el estado predicho y el real:

$$r_t^{\text{int}} = \frac{\eta}{2} \left\| \hat{\phi}(s_{t+1}) - \phi(s_{t+1}) \right\|^2$$

y se elige la acción con mayor error, un mayor error significa que esta acción será más novedosa para el agente, fomentando la exploración. Código asociado al agente de exploración ICM disponible en repositorio de GitHub.

Algorithm 1 Selección de acción basada en curiosidad (ICM)

function SELECTACTION(obs)
    $\phi \leftarrow$ Extractor(obs)
    $\phi_{\text{rep}} \leftarrow$ Repetir ϕ por cada acción
    $A_{\text{onehot}} \leftarrow$ Identidad de acciones de tamaño A
    $x_{\text{input}} \leftarrow$ Concatenar$(\phi_{\text{rep}}, A_{\text{onehot}})$
    $\hat{\phi}_{\text{next}} \leftarrow$ ModeloDirecto(x_{input})
    curiosidad $\leftarrow$ MSE$(\hat{\phi}_{\text{next}}, \phi_{\text{rep}})$
    curiosidad $\leftarrow$ Promediar sobre dimensión de características
    acción $\leftarrow \arg\max$(curiosidad)
    return acción
end function

Algoritmo 6. Pseudocódigo de selección de acciones de exploración.

4.5 Configuración experimental

A continuación se mencionan los detalles relacionados con los experimentos realizados, la cantidad de pasos de entrenamiento, hiperparametros, sets de entrenamiento y evaluación, y métricas de comparación a usar.

· **4.5.1 Tiempo de entrenamiento:** Considerando que en trabajos previos donde se busca evaluar la generalización, como “Gotta Learn Fast” [51] o “Leveraging Procedural Generation to Benchmark Reinforcement Learning” [37] , donde se entrenan a los agentes por entre 200 y 400 millones de pasos en cuestión de horas ya que los entornos son menos complejos y permiten muchos más entornos en simultáneo, acelerando el entrenamiento. En cambio, siendo cada nivel de Super Mario Bros un entorno complejo, es inviable entrenar durante una cantidad similar de pasos dado el costo de tiempo que supone. De todas formas se intenta reducir el tiempo de entrenamiento usando la función de Stable baselines3 “SubprocVecEnv”, la cual permite vectorizar cualquier entorno mediante subprocesos, además de diversificar la recolección de datos. Se realizaron pruebas con 100 millones de pasos para DQN, donde se obtuvo que la recompensa promedio dejaba de aumentar cerca de los 70 millones de pasos, por lo que finalmente se opta por dejar para todas las pruebas de DQN en 75 millones de pasos, demorando aproximadamente 50 horas de entrenamiento por cada agente. Para PPO se hicieron pruebas con hasta 150 millones de pasos, donde se obtuvo que la convergencia ocurre alrededor de los 120 millones de pasos, optando por dejar los experimentos de PPO en 150 millones de pasos.

En cuanto a NEAT e HyperNEAT, se entrena con una población de 100 individuos y 200 generaciones, evaluando en total 20.000 episodios, y asumiendo que cada episodio dura alrededor de 2500 pasos, se tendría aproximadamente 50 millones de pasos de entrenamiento.

· **4.5.2 Set de entrenamiento y evaluación:** Para evaluar la generalización, es necesario separar los entornos de entrenamiento y evaluación, por lo que se toman los 32 niveles del juego original, se eliminan los niveles donde se altere alguna mecánica base del juego (Figura 25), tales como los niveles de agua que modifican la mecánica de salto o niveles que se deben recorrer de una forma en concreto para llegar al final de este.

Luego, el resto de niveles se distribuyen de forma que el set de entrenamiento tenga una alta variedad tanto en distribución de enemigos y obstáculos como en dificultad, y el set de evaluación de forma que contenga una distribución distinta con algunas similitudes a los niveles del set de entrenamiento.

Los niveles excluidos son:

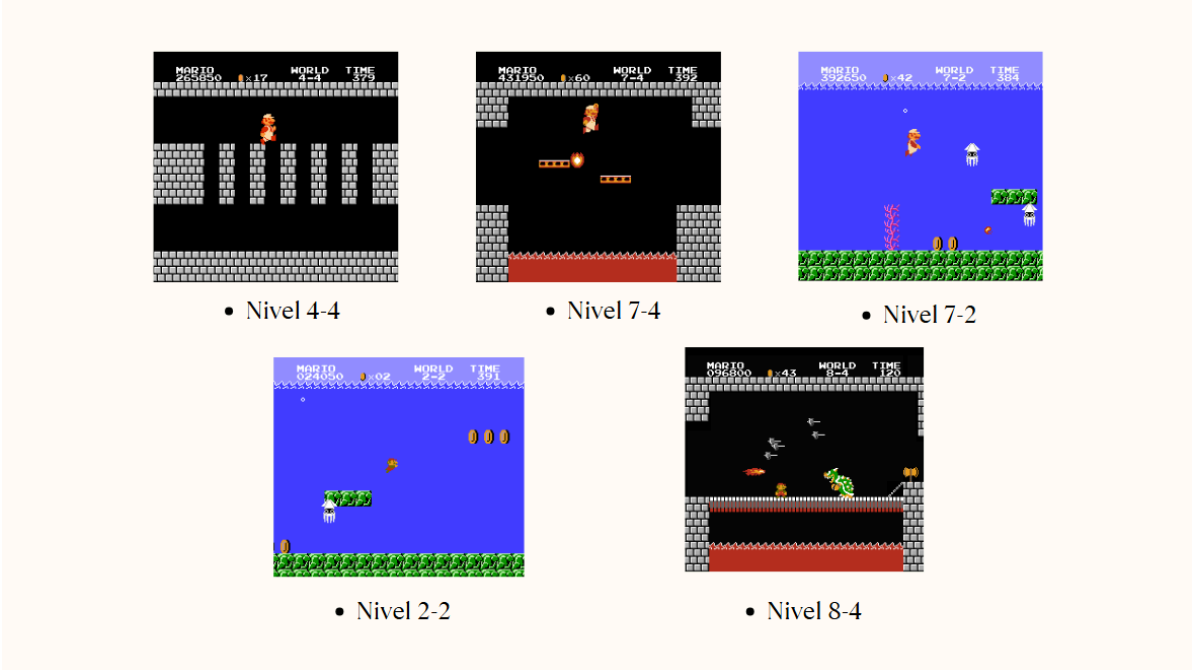

Figura 25. Niveles excluidos de la experimentación, todas las imágenes obtenidas de mariowiki.com.

Con esto solo quedan niveles “clásicos”, los cuales consisten en avanzar a la derecha, superando obstáculos y pisando enemigos, evitando ser golpeado por los mismos y alcanzar la bandera para superar el nivel.

El set de entrenamiento consiste en 14 niveles (Figura 26), teniendo al menos un nivel de cada mundo del juego, con la finalidad de mantener la variabilidad y dificultad que presenta el juego, específicamente los niveles son:

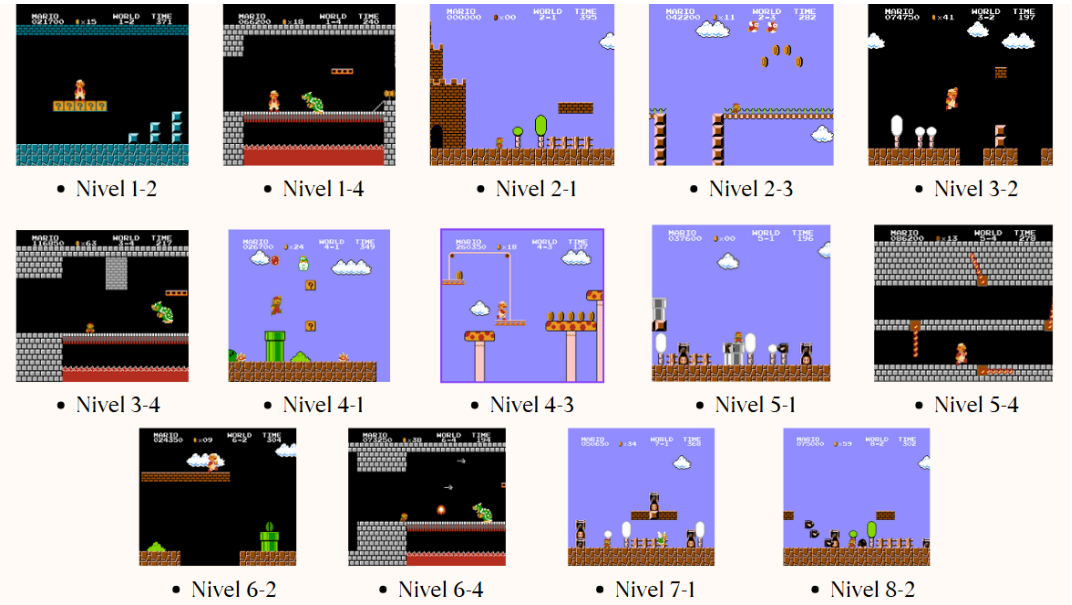

Figura 26. Niveles presentes en el set de entrenamiento, todas las imágenes obtenidas de mariowiki.com.

Finalmente el set de evaluación (Figura 27) consiste en los 13 niveles restantes del juego:

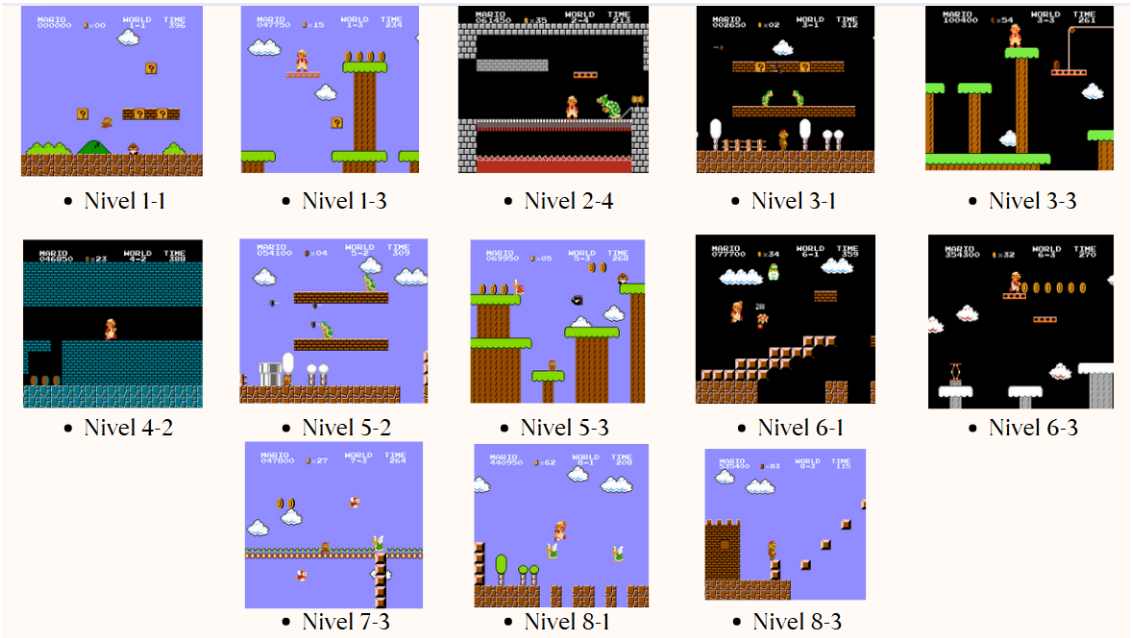

Figura 27. Niveles presentes en el set de evaluación, todas las imágenes obtenidas de mariowiki.com.

· **4.5.3 Hiperparámetros:** A continuación se presentan los hiperparámetros utilizados para el entrenamiento de los agentes.

Los valores utilizados en los hiperparámetros de PPO (Tabla 1) son obtenidos del trabajo inicial donde se presenta PPO [2], tanto el factor de descuento, el estimador generalizado de ventaja y el número de épocas se obtiene del experimento con juegos de Atari.

En cuanto al learning rate se ocupan valores utilizados en otros trabajos [57], utilizando una función que va reduciendo el valor del learning rate de forma lineal hasta el final del entrenamiento, el learning rate utilizado en caso de entrenar con la red neuronal IMPALA es obtenido del trabajo de "Impoola" [43].

El tamaño del batch se obtiene como la multiplicación del número de entornos × número de pasos por actualización, dando un tamaño total de 5632. Ahora, dado que tenemos 4 actualizaciones de política por iteración, podemos obtener la cantidad de pasos de optimización como:

$$Núm.\ de\ actualizaciones\ = \frac{tamaño\ del\ batch}{tamaño\ del\ minibatch} \times núm.\ de\ épocas$$

Obteniendo un total de 66 pasos de optimización por cada actualización del agente.

Hiperparámetro	**Valor**
Entornos	11
Learning rate	1×10^{-4}, para IMPALA 1.75×10^{-4}
Factor de descuento	0.99
Estimador generalizado de ventaja (GAE)	0.95
Número de pasos por actualización	512
Tamaño del minibatch	256
Entropía	0.03
Número de épocas	3

Tabla 1. Hiperparametros utilizados para PPO y PPO con LSTM.

Los valores utilizados para DQN y Rainbow (Tabla 2) son obtenidos en gran parte del paper Nature [1], tanto el factor de descuento, la frecuencia de entrenamiento y la actualización de la red objetivo.

En cuanto al tamaño del experience replay, se optó por uno más pequeño en comparación al usado comúnmente (usualmente se usa de 1 millón) para ahorrar memoria, pero se aumenta el tamaño del minibatch de 32 a 64 para obtener un entrenamiento más estable.

En cuanto al learning rate, se usa un valor estándar obtenido de otro trabajo [57], y en caso de utilizar la implementación de redes IMPALA, se usa un learning rate obtenido igualmente del trabajo de "Impoola" [43].

Como parámetros adicionales de Rainbow, cabe mencionar que se utiliza un valor $n = 3$ para el Multi-step learning, para las Noisy nets el valor $\sigma = 0.2$ para controlar la magnitud del ruido inicial en los pesos de la red, y en cuanto al PER, se utiliza un valor de $\beta = 0.45$, el cual ayuda a corregir el sesgo introducido al muestreo de PER.

Hiperparámetro	**Valor**
Entornos	11
Learning rate	1×10^{-4}, para IMPALA 2.5×10^{-4}
Tamaño del experience replay	100.000
Factor de descuento	0.99
Tamaño del minibatch	64
Frecuencia de entrenamiento	Cada 4 pasos
Actualización de la red objetivo	Cada 10.000 pasos en Rainbow, en DQN cada 1000 pasos.

Tabla 2. Hiperparámetros para DQN y Rainbow.

En cuanto a los parámetros de NEAT e HyperNEAT (Tabla 3), se ocupan archivos config para determinar los parámetros de entrenamiento, el archivo config utilizado fue obtenido del repositorio “Mario_NEAT_AI” [52].

Parámetro	**Valor**
Tamaño de población	300
Generaciones	200
Elitismo	10
Función de activación	Sigmoid
Mutación de peso de conexión	80%
Mutación de habilitación de conexión	60%
Mutación de deshabilitación de conexión	80%
Mutación de adición de conexión	90%
Mutación de adición de nodo	50%

Tabla 3. Parámetros más importantes para NEAT e HyperNEAT.

Se definen más parámetros para NEAT e HyperNEAT, como la inicialización de pesos o bias. Archivo asociado a los parámetros de NEAT disponible en repositorio de GitHub.

· **4.5.4 Métricas:** Para evaluar y cuantificar la capacidad de generalización y el desempeño de los agentes entrenados, se utilizan las siguientes métricas que dan una visión clara tanto de su rendimiento como de su capacidad de generalización:

· **Recompensa promedio móvil:** Esta métrica se calcula como el promedio de las recompensas obtenidas en los últimos n episodios durante el entrenamiento, esto se hace dado la alta variabilidad de las recompensas en cada episodio dada la exploración o la estocasticidad del entorno. Esto facilita la visualización del aprendizaje del agente durante el entrenamiento, su estancamiento o posible deterioro durante el mismo.

La recompensa promedio móvil se define dado un conjunto de recompensas R_i por episodio, la recompensa promedio en el paso t, en un espacio de tiempo k, se define como:

$$\bar{R}_t = \frac{1}{k} \sum_{i=t-k+1}^{t} R_i \quad \text{para t} \geq k$$

· **Evolución de la pérdida:** Esto se utiliza en los algoritmos de aprendizaje por refuerzo para reflejar cómo los agentes de aprendizaje por refuerzo están ajustando sus estimaciones, además puede servir como indicador de problemas en el aprendizaje como aprendizaje inestable o exploración insuficiente.

· **Recompensa promedio por nivel:** Una vez terminado el entrenamiento del agente, este se evalúa en el set de evaluación definido previamente, dando 10 oportunidades por cada nivel, calculando la recompensa promedio obtenida y mostrando el puntaje máximo de recompensa obtenido. Además se obtiene la desviación estándar en cada nivel, para mostrar la variabilidad entre las recompensas obtenidas por episodio y mostrar la consistencia del agente.

La desviación estándar σ se define como dado un conjunto de recompensas R_i por episodio:

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (R_i - \bar{R})^2}$$

5. Resultados

En esta sección se presentan los resultados obtenidos tras entrenar y evaluar los distintos algoritmos con los métodos de generalización propuestos. El objetivo es analizar el desempeño de cada agente en cuanto a la recompensa promedio obtenida durante el entrenamiento. Además se busca evaluar y analizar el desempeño en entornos no vistos, obteniendo su recompensa promedio por nivel, desviación estándar, porcentaje de completitud por nivel promedio y el máximo alcanzado.

Para simplificar la cantidad de modelos a evaluar, inicialmente se realiza una comparación entre el algoritmo en su versión base, es decir, sin mejoras ni métodos, y una de sus extensiones seleccionadas. En el caso de PPO se compara con su extensión PPO con redes recurrentes, la cual incluye capas de LSTM para incluir memoria en el aprendizaje. Y en el caso de DQN se compara con su extensión Rainbow, la cual incluye muchas mejoras como Double Q-learning, PER, Dueling Networks, entre otras. Dependiendo de qué algoritmo obtenga los mejores resultados, se aplicarán los métodos para mejorar la capacidad de generalización del agente.

El detalle de los resultados de cada experimento se encuentra en el Anexo C.

5.1 NEAT e HyperNEAT

· **Rendimiento en entrenamiento:** Los resultados obtenidos en los niveles de entrenamiento (Figura 26) del agente NEAT en cuanto a fitness promedio, fitness máximo y desviación estándar son los siguientes:

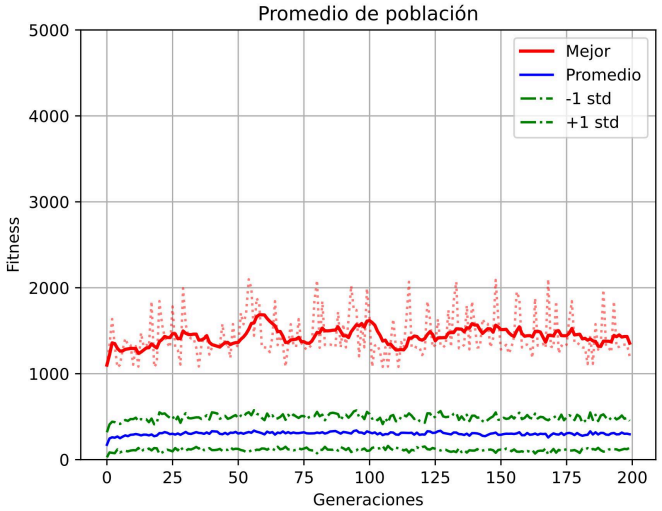

Figura 28. Gráfico de entrenamiento de NEAT.

En entrenamiento, como indica la Figura 28, NEAT obtiene valores de fitness muy bajos, alcanzando en promedio durante casi todo el entrenamiento valores de entre 250 y 400 en promedio por cada generación, convergiendo muy tempranamente en valores subóptimos, en cuanto al mejor genoma por cada generación, los valores son muy similares, variando entre 1000 y 2000 de fitness, indicando que ni el mejor genoma de cada generación fue capaz de terminar un nivel durante el entrenamiento.

Evaluando el agente NEAT en los niveles de entrenamiento (Figura 26) tras 200 generaciones, en cuanto a su porcentaje de completitud promedio y máximo por nivel, se obtuvieron los siguientes resultados:

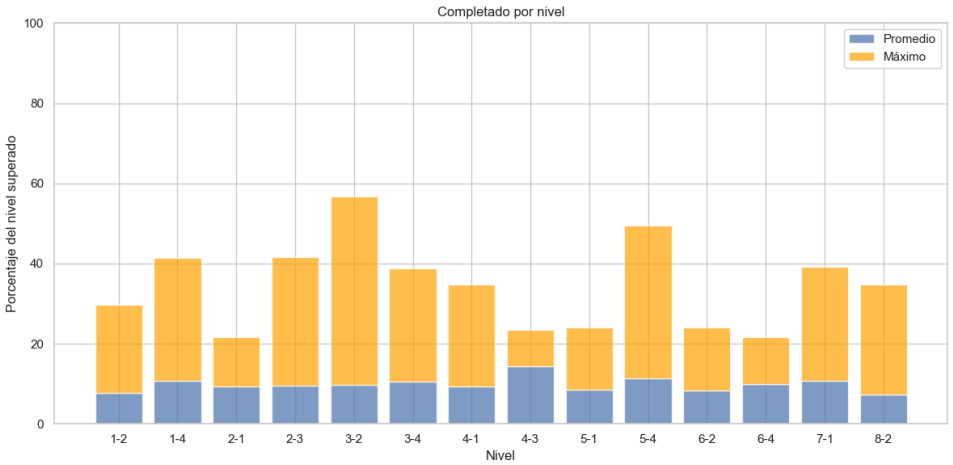

Figura 29. Rendimiento en el set de entrenamiento de NEAT.

Como se puede ver en la Figura 29, en promedio los individuos no son capaces de superar un 20% de cada nivel, siendo el más cercano el nivel 4-3. Con el caso específico de algunos individuos que logran un notorio mejor desempeño, logrando superar el 40% del nivel, como en los niveles 1-4, 2-3, 3-2 y 5-4, fuera de estos individuos, el desempeño de la generación es bastante deficiente.

Los resultados obtenidos en los niveles de entrenamiento (Figura 26) del agente HyperNEAT en cuanto a fitness promedio, fitness máximo y desviación estándar son los siguientes:

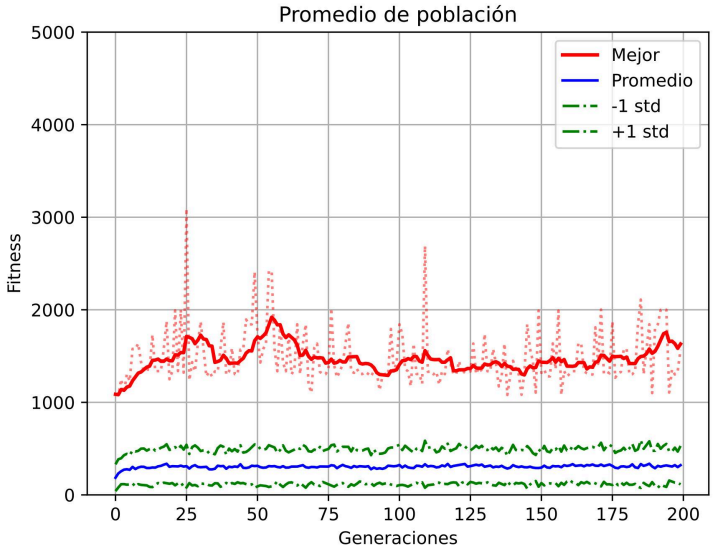

Figura 30. Gráfico de entrenamiento de HyperNEAT.

El rendimiento en entrenamiento de HyperNEAT (Figura 30) resulta en valores muy similares a los obtenidos por NEAT, con la diferencia de que en algunas generaciones el mejor genoma obtuvo valores de fitness que superan los 2000 de fitness, pero sigue sin ser suficiente para superar un nivel en entrenamiento.

Evaluando el agente HyperNEAT en los niveles de entrenamiento (Figura 26) en cuanto a su porcentaje de completitud promedio y máximo por nivel, tras 200 generaciones, se obtuvieron los siguientes resultados:

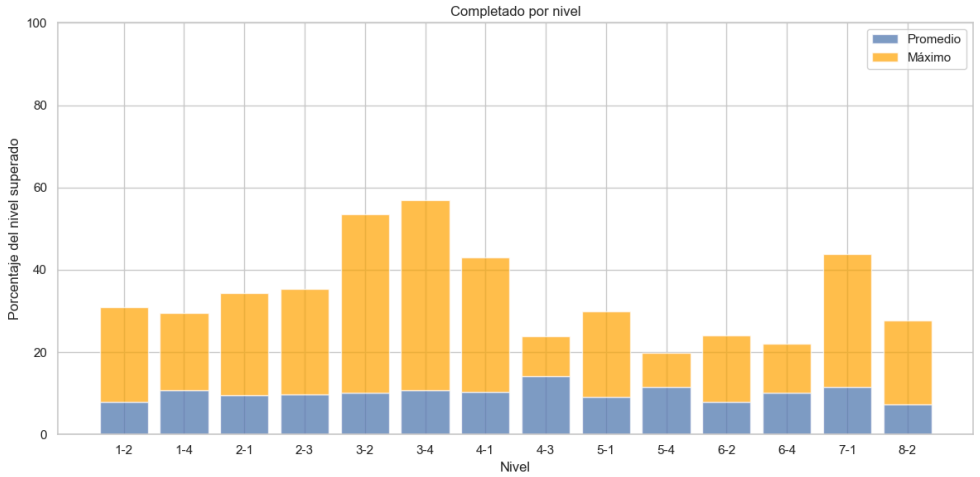

Figura 31. Rendimiento en el set de entrenamiento de HyperNEAT.

Como muestra la Figura 31, el agente HyperNEAT, obtiene mejoras en algunos niveles, como el 2-1 o 3-4, pero en líneas generales el rendimiento es bastante similar a NEAT. No obstante, el comportamiento de HyperNEAT es más consistente entre niveles, mostrando menos variabilidad en comparación a NEAT, el cual presenta aumentos de rendimiento en algunos niveles específicos, lo cual podría indicar que el uso de las CPPNs de HyperNEAT podría tener un pequeño impacto en la estabilidad y la generalización. sin diferencias marcadas.

· **Rendimiento en evaluación:** Los resultados del agente NEAT tras 200 generaciones en los niveles de evaluación (Figura 27), en cuanto a porcentaje de completitud promedio y máximo por nivel, son los siguientes:

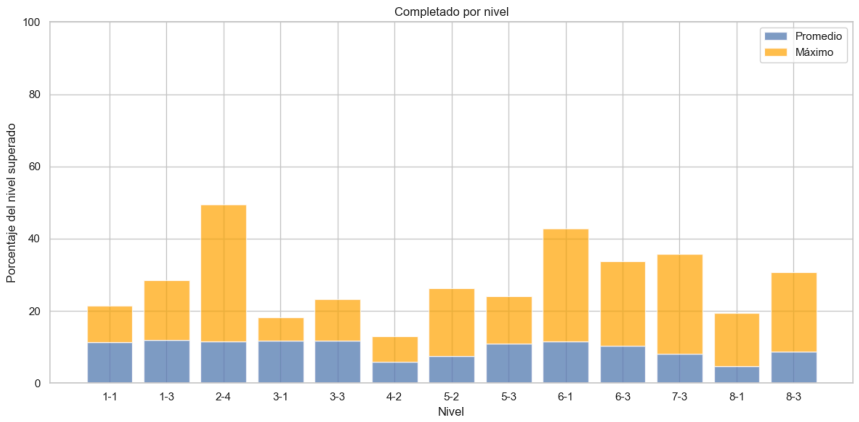

Figura 32. Rendimiento en el set de evaluación de NEAT.

Evaluando los 300 individuos de la población en cada nivel de evaluación (Figura 32), nuevamente se obtuvieron resultados en promedio bastante bajos, ninguno superando el 12% del nivel superado en promedio, teniendo individuos específicos con un rendimiento notoriamente superior al promedio. Los niveles con mejor rendimiento fueron los niveles 2-4, 6-1, 6-3 y 7-3, donde en los mejores casos superan el 40% del nivel superado o se acercan bastante.

Los resultados del individuo ganador de NEAT en los niveles de evaluación (Figura 27), en cuanto a porcentaje de completitud por nivel son los siguientes:

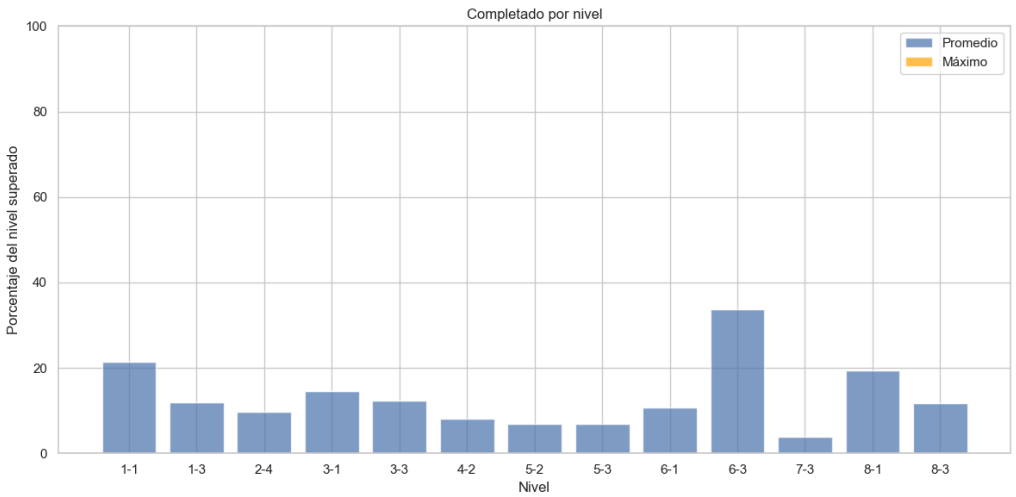

Figura 33. Rendimiento del individuo ganador de NEAT en los niveles de evaluación.

El individuo ganador se ve que obtiene resultados (Figura 33) en los niveles 1-1, 6-3 y 8-1 (Figura 34), mientras que en el resto de niveles se observa un agente que se estanca con facilidad, o que cae directamente en el primer precipicio, mostrando un mal rendimiento en general a nuevos niveles no vistos.

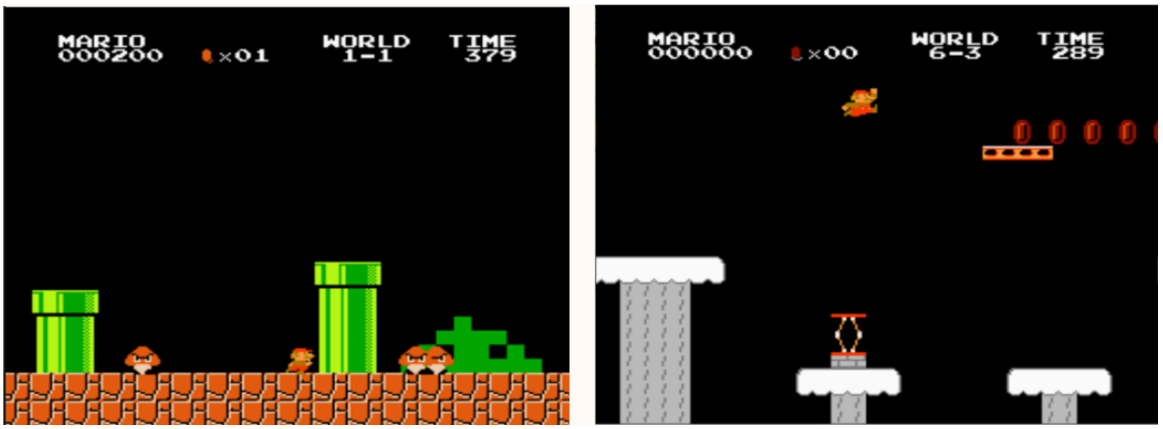

Figura 34. Niveles donde el individuo ganador de NEAT tiene un buen rendimiento, nivel 1-1 (izquierda) y 6-3 (derecha).

Evaluando el agente HyperNEAT en los niveles de evaluación (Figura 27) tras 200 generaciones, en cuanto a porcentaje de completitud promedio y máximo por nivel, se obtuvieron los siguientes resultados:

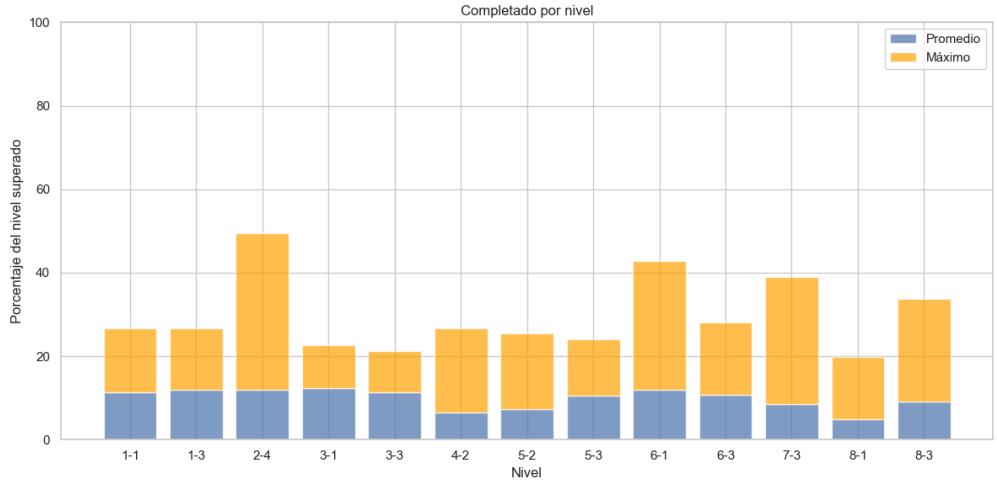

Figura 35. Rendimiento en el set de evaluación de HyperNEAT.

El agente HyperNEAT nuevamente obtiene resultados (Figura 35) muy similares en evaluación al agente NEAT, obteniendo únicamente una mejora estadísticamente significativa en el nivel 4-2 ($p < 0.01$), y para el resto de niveles donde parece haber una leve mejora o disminución en el rendimiento, ninguna de estas llega a ser estadísticamente significativa ($p > 0.1$).

Los resultados del individuo ganador de HyperNEAT en los niveles de evaluación (Figura 27), en cuanto a porcentaje de completitud por nivel son los siguientes:

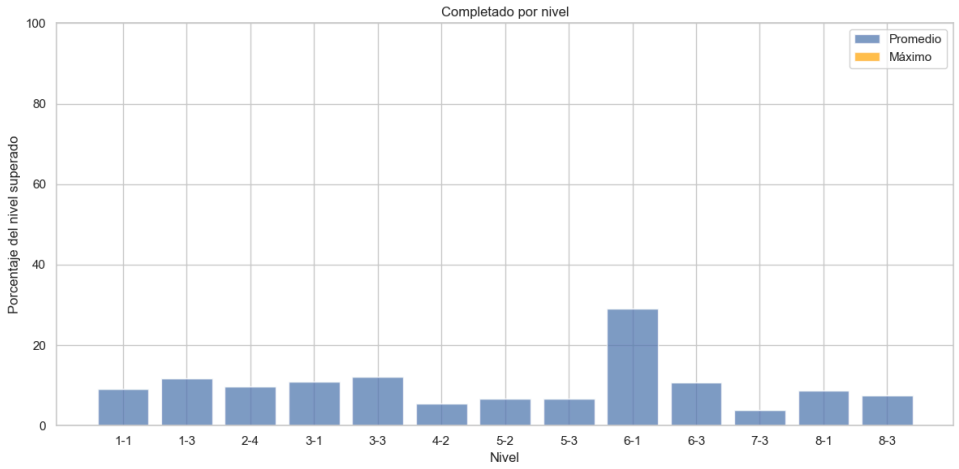

Figura 36. Rendimiento del individuo ganador de HyperNEAT en los niveles de evaluación.

El único nivel donde destaca el individuo ganador, como se muestra en la Figura 36, es en el nivel 6-1 (Figura 37), donde alcanza casi un 30% del nivel, en el resto de niveles obtiene un desempeño muy bajo, completando entre un 3% y un 11% en el resto de niveles.

Figura 37. Nivel 6-1 donde el individuo ganador de HyperNEAT obtiene avances considerables.

El impacto de las CPPNs en los experimentos realizados resulta mínimo o casi nulo. Si bien HyperNEAT muestra ligeras mejoras en el entrenamiento y también en la evaluación, no supuso ninguna mejora significativa en ningún apartado, tanto en la generalización como en el entrenamiento. Las CPPNs no fueron capaces de capturar patrones espaciales quizás por la alta variabilidad de los entornos a los que se somete a la población durante el entrenamiento.

Para el caso de NEAT e HyperNEAT no se realizan más pruebas con los métodos para la generalización seleccionados, dado que estos están pensados para usarse con algoritmos de aprendizaje por refuerzo, por lo que resultarían poco efectivos, optando por utilizar los resultados obtenidos como línea base para los demás algoritmos.

5.2 Deep Q Network y Rainbow

· **Rendimiento en entrenamiento:** Los resultados obtenidos en cuanto a la recompensa promedio móvil en el entrenamiento (Figura 26) de los agentes Rainbow y DQN son los siguientes:

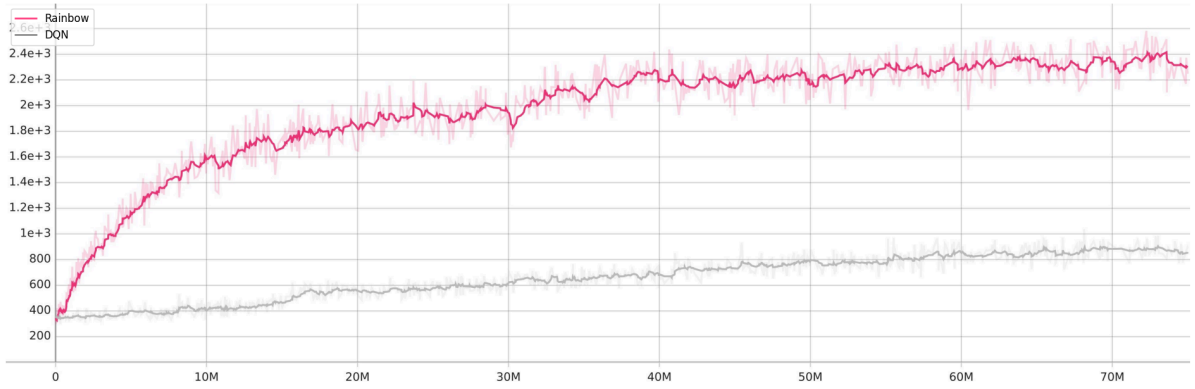

Figura 38. Gráfico de recompensa promedio móvil entre Rainbow y DQN.

Se puede observar la notoria diferencia en cuanto a la recompensa obtenida en la Figura 38, Rainbow triplica casi en todo momento la recompensa obtenida por DQN, evidenciando la eficacia de las mejoras propuestas y el rendimiento mostrado en la Figura 12.

Mientras en Rainbow se puede ver una curva de aprendizaje pronunciada hasta los 10 millones, no se puede decir lo mismo de DQN, donde se puede ver una lenta curva durante todo el entrenamiento, junto con algunos periodos de estancamiento.

Evaluando el agente DQN en los niveles de entrenamiento (Figura 26) en cuanto a la recompensa promedio obtenida, desviación estándar y porcentaje promedio y máximo de completitud por nivel, se obtuvieron los siguientes resultados:

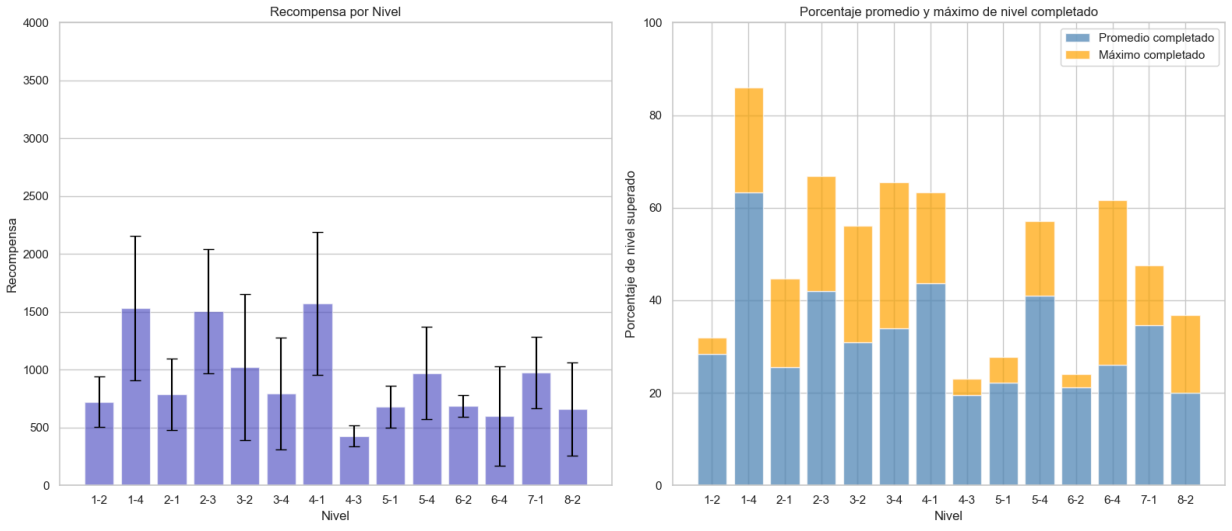

Figura 39. Rendimiento en el set de niveles de entrenamiento de DQN.

Podemos ver que la política aprendida por DQN durante el entrenamiento (Figura 39) no alcanza para completar ningún nivel de entrenamiento, obteniendo mal desempeño en los niveles 4-3, 5-1 y 8-2, donde apenas logra superar el 20% del nivel, mostrando que la configuración de entrenamiento puede ser muy compleja y variable para que DQN logre aprender una política que logre completar algún nivel de entrenamiento y lograr más avances en estos niveles.

Evaluando el agente Rainbow en los niveles de entrenamiento (Figura 26) en cuanto a la recompensa promedio obtenida, desviación estándar y porcentaje promedio y máximo de completitud por nivel se obtuvieron los siguientes resultados:

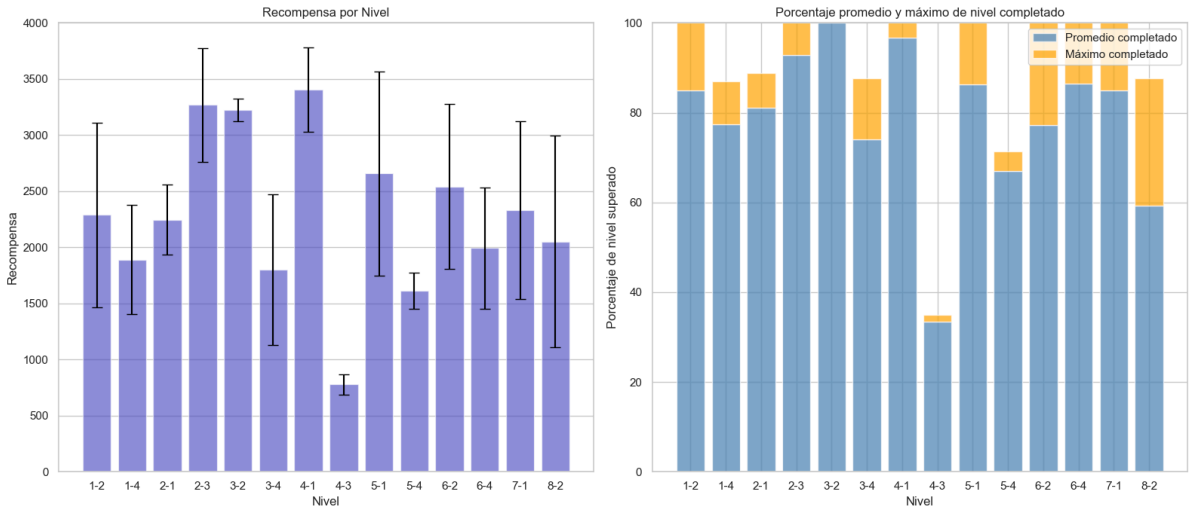

Figura 40. Rendimiento en el set de niveles de entrenamiento de Rainbow.

La política aprendida por Rainbow, (Figura 40) por otra parte logra superar la gran mayoría de niveles de entrenamiento y logra avances entre el 60% y 80% en los niveles que no logra superar. A excepción del nivel 4-3, el cual es uno de los niveles más difíciles del juego, ya que tiene zonas amplias donde es posible caer al vacío y perder en el nivel. Pese a esto, Rainbow presenta una más que obvia mejora con respecto a DQN en todos los niveles.

· **Evolución de la pérdida:** Los resultados obtenidos en cuanto a la pérdida en el entrenamiento (Figura 26) de los agentes Rainbow y DQN son los siguientes:

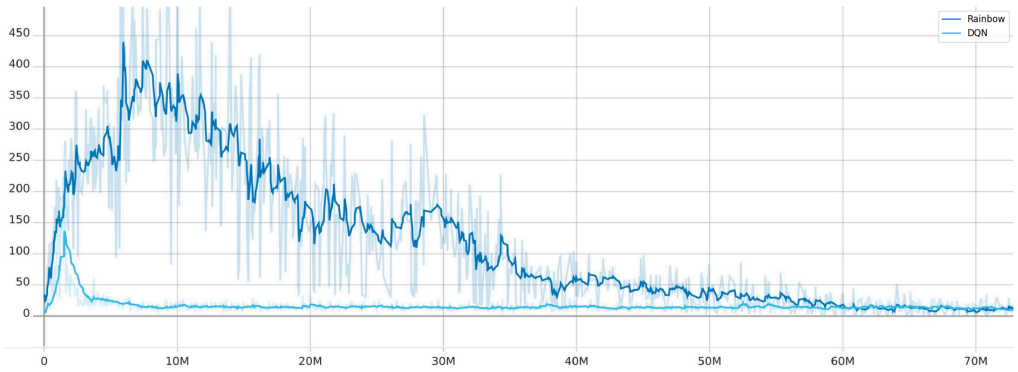

Figura 41. Gráfico de pérdida entre Rainbow y DQN.

Pese al gran rendimiento obtenido por Rainbow, tiene valores de pérdida mucho mayores que DQN (Figura 41), el cual parece crecer y disminuir rápidamente, pareciendo más estable, pero dado las recompensas obtenidas de la Figura 38, se podría decir que está aprendiendo una política subóptima. Rainbow parece ser más inestable, pero el estar durante gran parte del entrenamiento ajustando su política, le permite aprender una política más robusta para avanzar en los niveles de entrenamiento.

· **Rendimiento en evaluación:** Los resultados del agente DQN en los niveles de evaluación (Figura 27), en cuanto a recompensa promedio obtenida, desviación estándar y porcentaje promedio y máximo de completitud por nivel, son los siguientes:

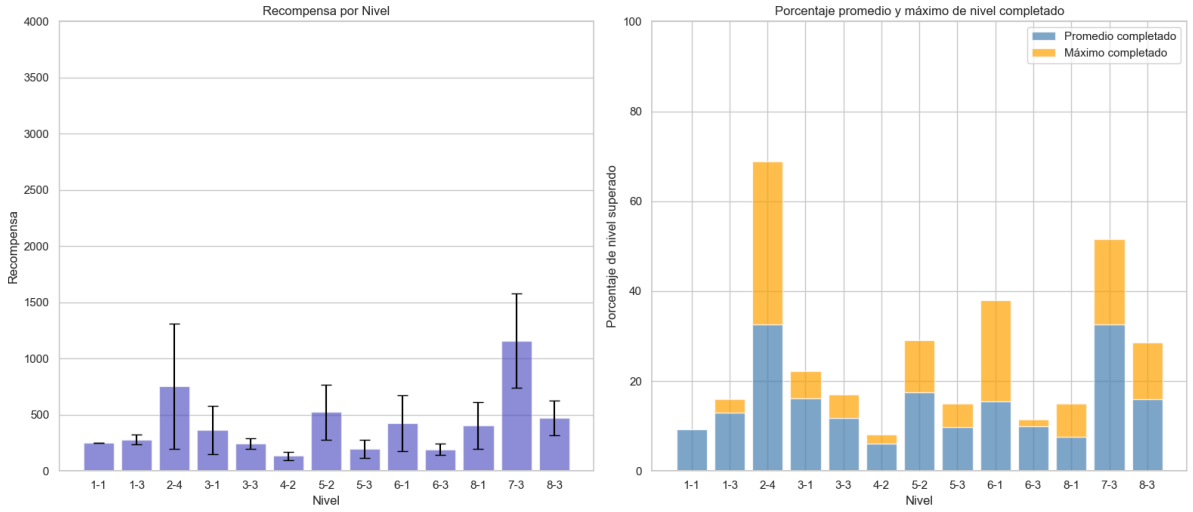

Figura 42. Rendimiento en el set de niveles de evaluación de DQN.

Los resultados (Figura 42) demuestran una mala generalización en la mayor cantidad de niveles, en cuanto a las recompensas obtenidas, esta varía mucho, logrando las recompensas

más altas en los niveles 2-4 con 754 y 7-3 con 1159 de recompensa promedio, mientras que en el resto de niveles la recompensa obtenida oscilan entre valores similares de entre 150 y 500.

En cuanto a el porcentaje de cada nivel avanzando, la tendencia es similar, donde en los niveles 2-4, 5-2, 6-1, 7-3 y 8-3 el agente logra superar más del 20% del nivel, incluso en los niveles 2-4 y 7-3 logra superar más de la mitad del nivel en el mejor caso, con un 68% y un 52% respectivamente. Pero no se puede decir lo mismo del resto de niveles, donde ninguno más logra superar el 20% del nivel.

El bajo rendimiento de DQN se evidencia desde la curva de recompensa obtenida en el entrenamiento y es confirmado en la evaluación, evidenciando el sobreajuste a un subconjunto pequeño de niveles, fallando en encontrar una política que logre generalizar o superar niveles de entrenamiento de forma consistente.

Los resultados del agente Rainbow en los niveles de evaluación (Figura 27), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, son los siguientes:

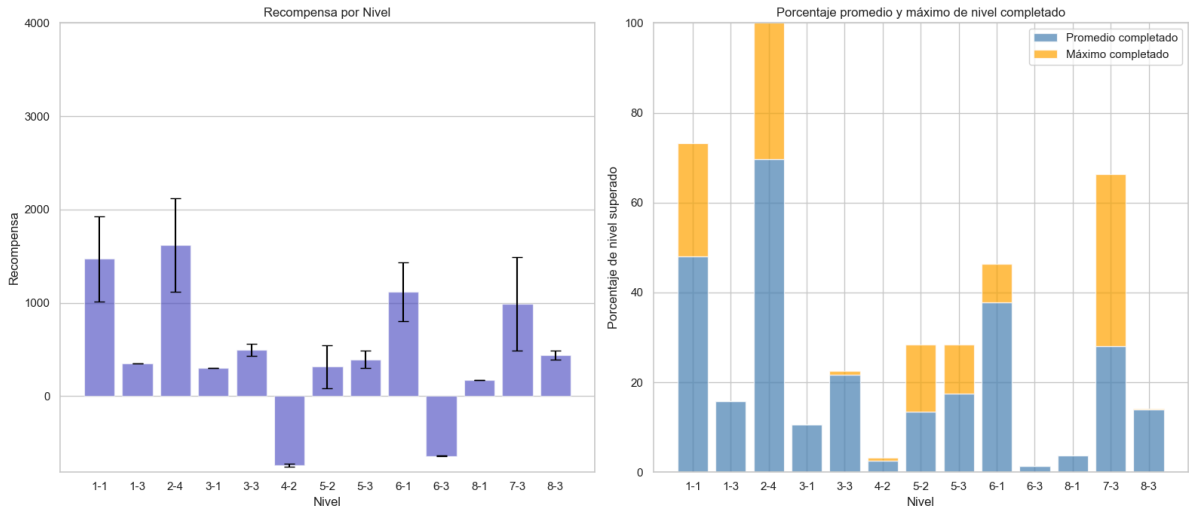

Figura 43. Rendimiento en el set de niveles de evaluación de Rainbow.

Los resultados obtenidos por el agente Rainbow (Figura 43) demuestran una mejora en la generalización, obteniendo un mejor rendimiento en una mayor cantidad de niveles, por ejemplo el nivel 1-1, donde en promedio DQN obtiene una recompensa promedio de 250 y

en promedio logra completar menos del 10% del nivel, Rainbow logra una diferencia estadísticamente significativa ($p < 0.0001$), además de una recompensa de 1470 en promedio, completando cerca de un 50% del nivel en promedio. En el nivel 2-4, el cual es uno de los niveles donde DQN tuvo un mejor rendimiento, Rainbow logra una diferencia estadísticamente significativa ($p < 0.001$), completando el nivel obteniendo una mayor recompensa promedio. Además en el nivel 7-3, donde DQN obtiene en promedio 1159 de recompensa, Rainbow también logra superar esto con una diferencia estadísticamente significativa ($p < 0.0001$) y un promedio de recompensa de 1987. Hay niveles como el 3-1, 8-1 y 8-3 donde DQN obtiene resultados ligeramente mejores que Rainbow, pero estos no resultan en una diferencia estadísticamente significativa ($p > 0.05$).

Sin embargo, hay niveles como el 4-2 y el 6-3 donde el agente Rainbow obtiene recompensas negativas y no avanza en el nivel (Figura 44). En estos niveles se ve que el agente se queda quieto durante todo el episodio o moviéndose únicamente hacia la izquierda, sin lograr ningún avance, este comportamiento podría deberse a una falta de representación en el set de niveles de entrenamiento, o a que el agente Rainbow haya aprendido correlaciones equivocadas.

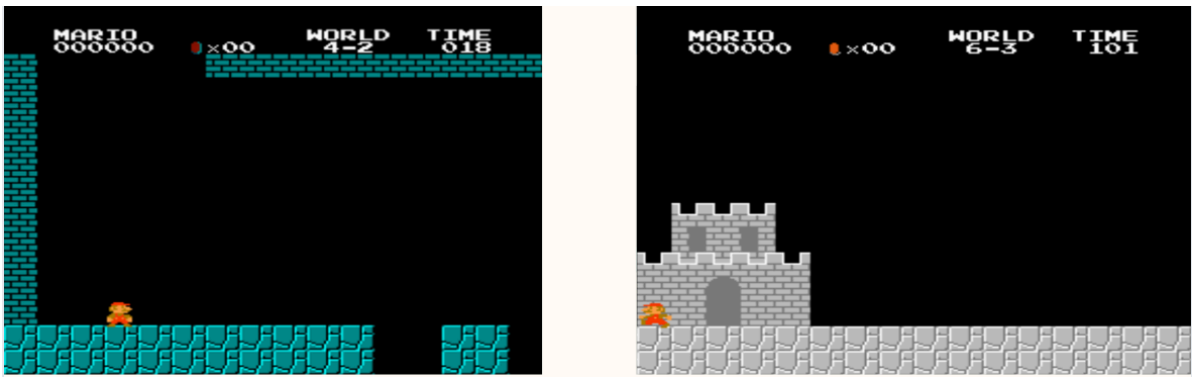

Figura 44. Niveles donde el agente Rainbow tiene problemas, el nivel 4-2 (izquierda) se puede ver al agente quieto esperando que pase el tiempo, mientras que en el nivel 6-3 (derecha), se puede ver al agente corriendo constantemente hacia la izquierda, sin avanzar durante el nivel.

Pese a los malos resultados obtenidos en los niveles 4-2 y 6-3, Rainbow demuestra mejores resultados en entrenamiento y en el set de evaluación (aunque aún mejorable) gracias a las mejoras en sus componentes, por esto, Rainbow es el algoritmo elegido para aplicar los métodos de mejora de generalización.

5.2.1 Rainbow con Domain Randomization

· **Rendimiento en entrenamiento:** Los resultados obtenidos en cuanto a la recompensa promedio móvil en el entrenamiento (Figura 26) de Rainbow con randomización, son los siguientes:

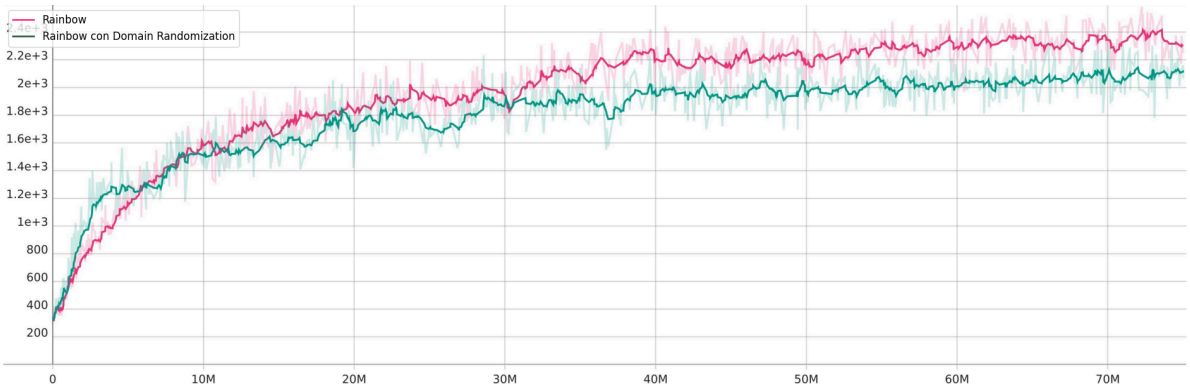

Figura 45. Gráfico de recompensa promedio entre Rainbow estándar y con randomización.

Se puede observar una curva similar entre Rainbow y el mismo con randomización de dominio (Figura 45), si bien la curva de aprendizaje inicial entre los 10 millones de pasos es más pronunciada para el agente con randomización, luego este obtiene menos recompensa durante el resto del entrenamiento.

Evaluando el agente Rainbow con randomización en los niveles de entrenamiento (Figura 26), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, se obtuvieron los siguientes resultados:

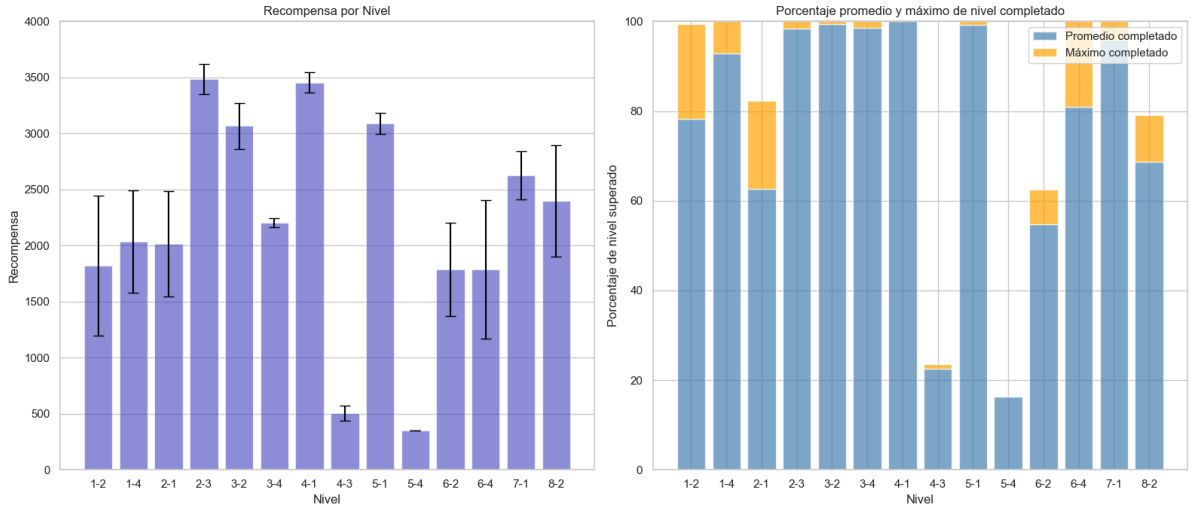

Figura 46. Rendimiento en el set de niveles de entrenamiento de Rainbow con randomización.

La política aprendida por Rainbow con randomización (Figura 46) es bastante similar a la de Rainbow estándar con un par de excepciones, ya que el agente con randomización logra superar el nivel 1-4 y 3-4 cuando el agente estándar no lo consigue, pero a la misma vez el agente con randomización obtiene un menor rendimiento en los niveles 5-4 y 6-2, además del nivel 4-3, donde si bien ninguno de los agentes Rainbow logra superar el nivel, el agente con randomización obtiene un peor rendimiento, apenas superando el 20% del nivel.

· **Evolución de la pérdida:** Los resultados obtenidos en cuanto a la pérdida en el entrenamiento (Figura 26) del agente Rainbow con randomización son los siguientes:

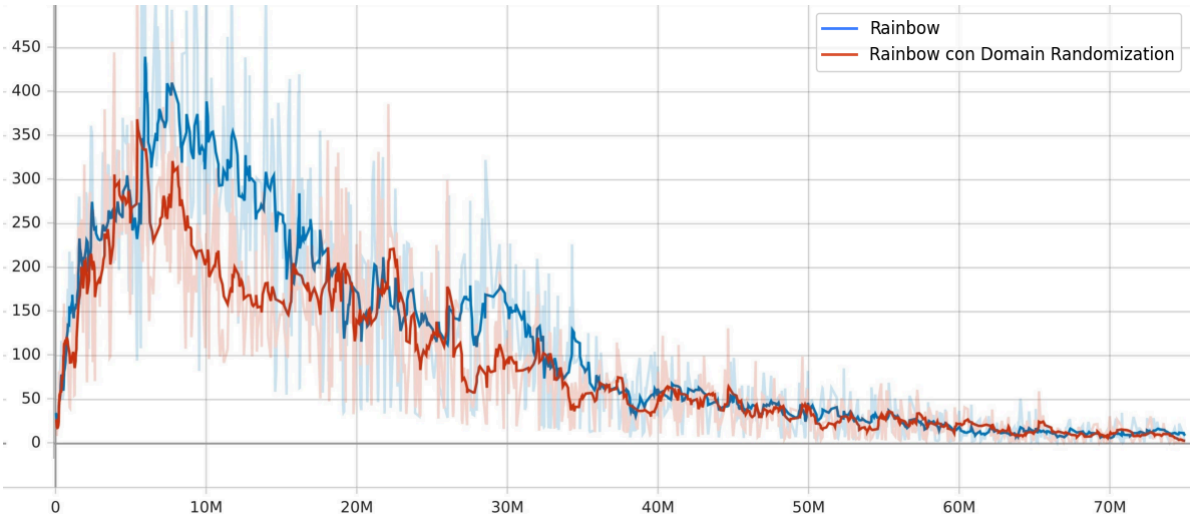

Figura 47. Gráfico de pérdida entre Rainbow estándar y con randomización.

La curva de pérdida de Rainbow con randomización (Figura 47) parece converger antes y ser más estable que la curva de pérdida de Rainbow estándar, aunque esto pueda parecer contraintuitivo, esto parece ser señal de que la randomización actúa como un regularizador, haciendo que el agente no aprenda políticas específicas que luego deba desaprender, logrando que los valores de pérdida de este agente sean menores, y ayudando al agente a converger más tempranamente.

· **Rendimiento en evaluación:** Los resultados obtenidos por el agente Rainbow con randomización en los niveles de evaluación (Figura 27), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, son los siguientes:

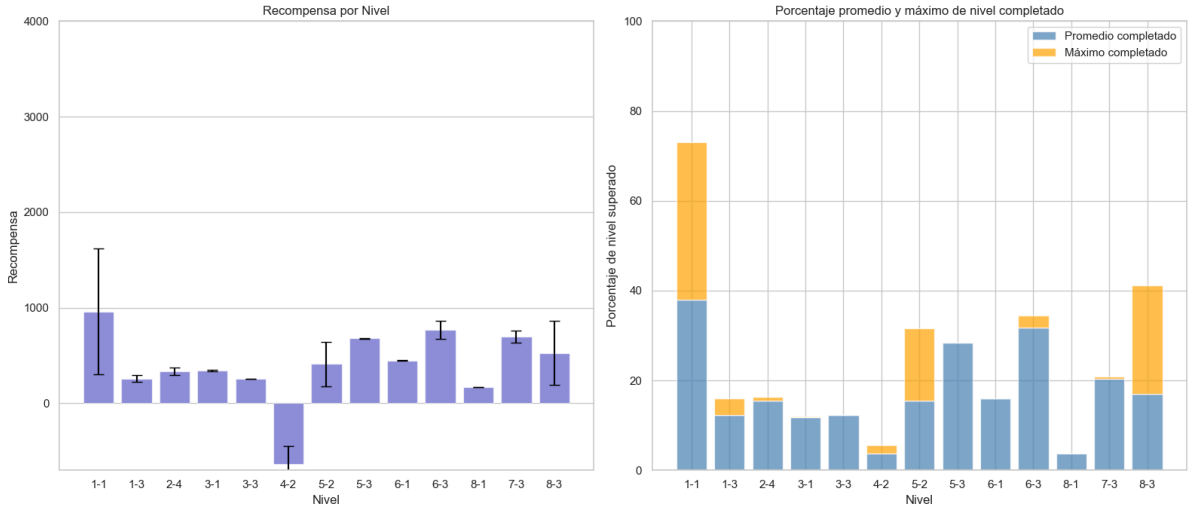

Figura 48. Rendimiento en el set de niveles de evaluación de Rainbow con randomización.

Del gráfico de evaluación (Figura 48), podemos observar que obtiene menos recompensas en casi todos los niveles, a pesar de que parece avanzar un porcentaje similar a Rainbow estándar, esto se debe a que durante la evaluación se puede observar a un agente que parece "dudar" de sus acciones, demorando un poco entre cada acción realizada, obteniendo penalizaciones por estancamiento y paso del tiempo.

En cuanto a lo que el agente logra avanzar por nivel, tenemos el caso del nivel 2-4, donde el agente Rainbow estándar consigue completar el nivel, el agente con randomización falla constantemente el primer salto y no logra seguir avanzando. Por otro lado, en el nivel 5-3 el agente con randomización obtiene una mejora estadísticamente significativa con respecto a

Rainbow ($p < 0.0001$), y en el nivel 8-3 pese a lograr avanzar más en el nivel, no logra una diferencia estadísticamente significativa ($p > 0.05$).

El caso más llamativo es el del nivel 6-3, donde el agente Rainbow estándar no conseguía avanzar en el nivel, moviéndose constantemente hacia la izquierda sin avanzar, esto no ocurre en el agente con randomización, donde el agente logra avances en el nivel. Una posible razón del comportamiento dado en este nivel es que el agente Rainbow estándar haya aprendido una correlación falsa con el dibujo del castillo del fondo, presente en gran parte de los niveles del juego tras la bandera que marca el final del nivel. En el caso del nivel 6-3, un castillo muy similar está presente al inicio del nivel (Figura 49), el comportamiento aprendido por el agente estándar puede indicar que haya aprendido una relación donde el agente diga "hay que llegar al castillo", en lugar de "debo moverme hacia la derecha", este problema se resuelve con la randomización del dominio, ayudando al agente a reducir la dependencia de características visuales y enfocarse en patrones más importantes, como el movimiento del agente.

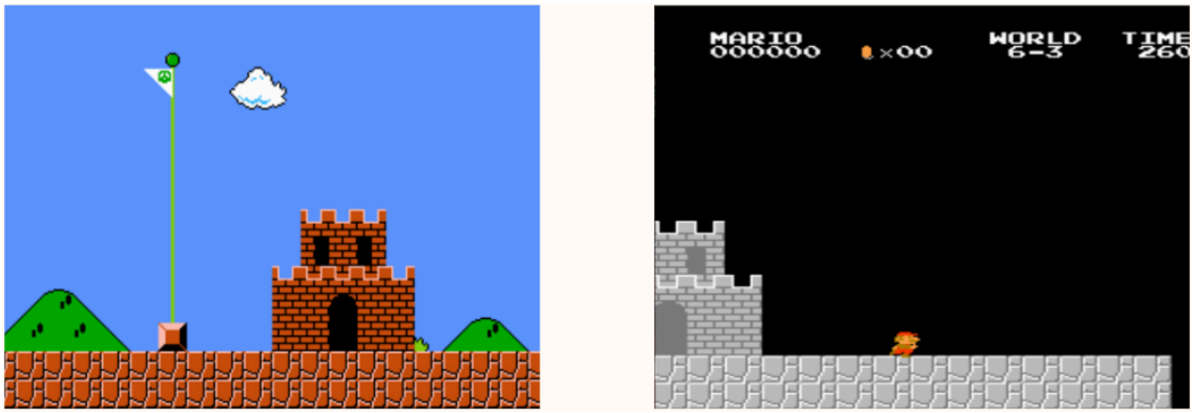

Figura 49. Castillo al final del nivel 1-1 (izquierda), un ejemplo del castillo presente al final de cada nivel tras la bandera, se puede ver la similitud del castillo del nivel 6-3 (derecha), y como el agente Rainbow con randomización avanza en el nivel.

Ahora bien, este método por sí solo no parece mostrar una gran mejora en la generalización del agente, ya que si bien se logra corregir comportamientos "catastróficos", como en el nivel 6-3, y comportamientos más consistentes en niveles específicos como el nivel 5-3 o el 7-3, el rendimiento en el resto de niveles fue menor que el agente Rainbow estándar y aún no logra avanzar en el nivel 4-2, indicando un "sacrificio" del rendimiento por un agente algo más robusto. Este método resultaría más eficaz si lograra aleatorizar más elementos del

juego, como su estructura y obstáculos, como los dibujos del fondo de cada nivel, algo que resultaría complejo de realizar, considerando que se debe garantizar solubilidad y coherencia en los niveles.

5.2.2 Rainbow con Domain Randomization e IMPALA

· **Rendimiento en entrenamiento:** Los resultados obtenidos en cuanto a la recompensa promedio móvil en el entrenamiento (Figura 26) de Rainbow con IMPALA, es el siguiente:

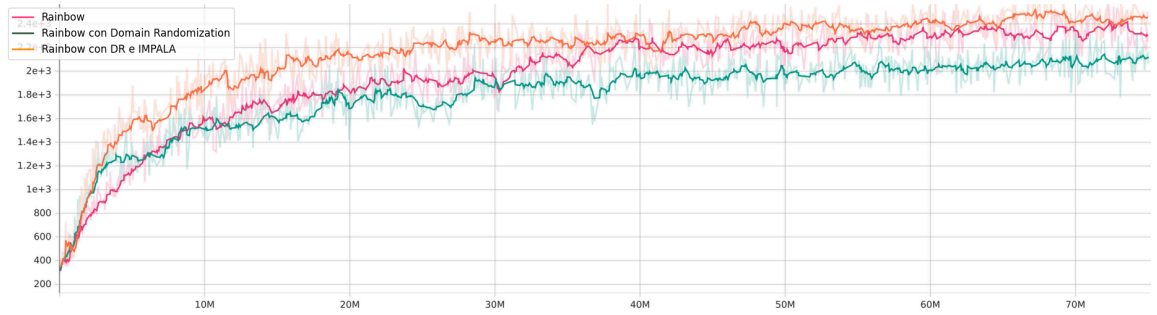

Figura 50. Gráfico de recompensa promedio entre Rainbow estándar, con randomización y con IMPALA.

Del gráfico de la Figura 50 se puede observar una curva de aprendizaje aún más pronunciada que los agentes Rainbow anteriores, y una recompensa promedio superior en todo el entrenamiento, haciendo que el agente Rainbow con IMPALA converja antes.

Evaluando el agente Rainbow con IMPALA en los niveles de entrenamiento (Figura 26), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, se obtuvieron los siguientes resultados:

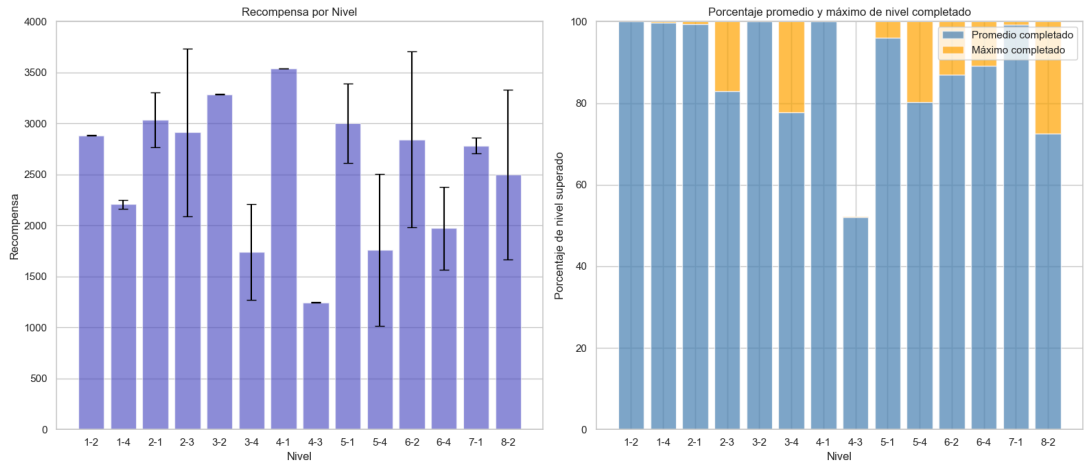

Figura 51. Rendimiento en el set de niveles de entrenamiento de Rainbow con randomización e IMPALA.

La política aprendida por Rainbow con IMPALA (Figura 51) muestra que es capaz de superar todos los niveles de entrenamiento al menos una vez a excepción del nivel 4-3, mostrando una gran mejora en comparación a los agentes Rainbow previos.

· **Evolución de la pérdida:** Los resultados obtenidos en cuanto a la pérdida en el entrenamiento (Figura 26) del agente Rainbow con IMPALA son los siguientes:

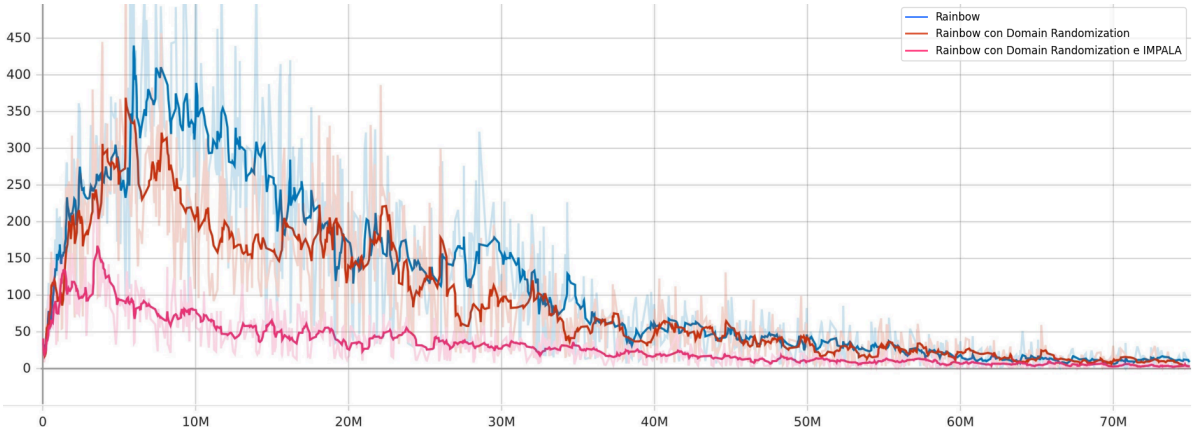

Figura 52. Gráfico de pérdida entre Rainbow estándar, con randomización y con IMPALA.

En cuanto a la evolución de la pérdida (Figura 52), se nota el efecto de los regularizadores usados con la red IMPALA, obteniendo valores de pérdida mucho más pequeños y estables que los agentes anteriores.

· **Rendimiento en evaluación:** Los resultados del agente Rainbow con IMPALA en los niveles de evaluación (Figura 27), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, son los siguientes:

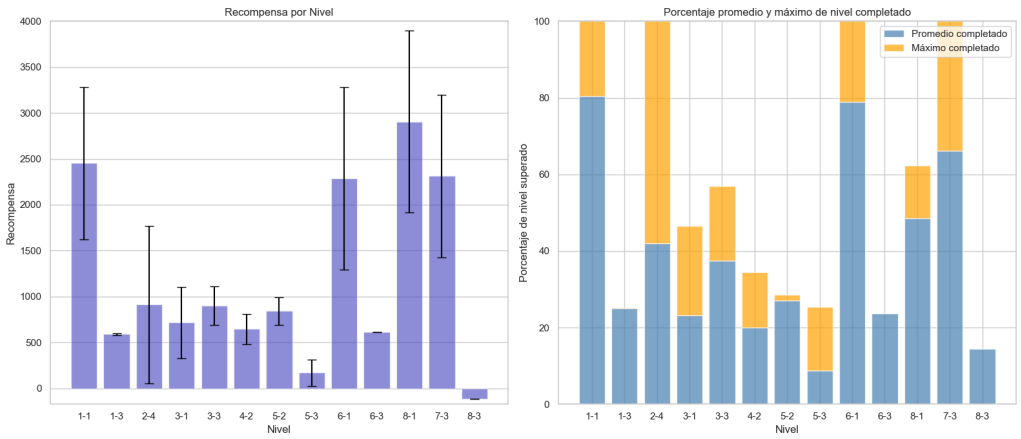

Figura 53. Rendimiento en el set de niveles de evaluación de Rainbow con randomización e IMPALA.

Se puede observar una mejora sustancial en cuanto a la generalización a niveles no vistos (Figura 53), logrando completar los niveles 1-1, 2-4, 6-1 y 7-3 de forma consistente, donde de los 10 intentos por nivel, estos niveles son completados aproximadamente 8 veces.

También se puede notar una mejora en niveles más difíciles donde el agente no ha obtenido buenos resultados, como el nivel 1-3, donde el agente Rainbow con IMPALA obtiene una mejora estadísticamente significativa ($p < 0.0001$) con respecto a Rainbow estándar, logrando superar el 20% del nivel completado. También es el caso del nivel 3-1, donde nuevamente el agente Rainbow con IMPALA logra una mejora estadísticamente significativa ($p < 0.005$), logrando superar el 20% del nivel completado en promedio y superando el 40% en el máximo alcanzado, donde Rainbow estándar y con randomización no logran superar el 20%. Otro caso digno de mención es el nivel 3-3, donde nuevamente Rainbow con IMPALA logra una mejora estadísticamente significativa ($p < 0.0001$), logrando llegar a casi un 40% del nivel completado en promedio y cerca de un 60% como máximo mientras que ambos agentes Rainbow previos no superan el 20%.

De todas formas, aún hay niveles donde el agente no parece tener avances significativos, como el nivel 5-3 y 8-3 (Figura 54), donde avanzan hacia adelante hasta chocar con el primer enemigo, o correr hacia adelante constantemente contra una pared.

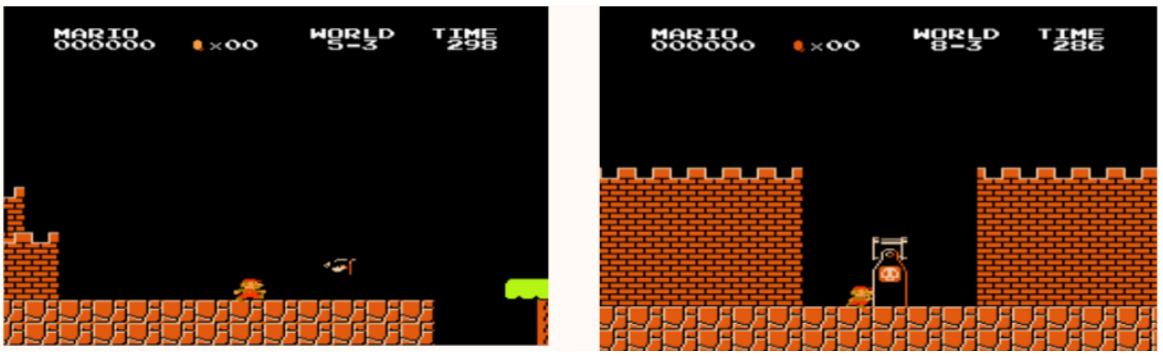

Figura 54. Niveles donde el agente Rainbow con IMPALA falla, en el nivel 5-3 (izquierda), avanza constantemente contra el primer enemigo. mientras que en el nivel 8-3 (derecha), se queda atascado en el segundo obstáculo del nivel.

Los resultados obtenidos demuestran lo dicho en trabajos previos de generalización donde usan la red neuronal IMPALA [37][42], redes más pequeñas pueden ser efectivas en entrenamiento y cuando la evaluación del agente se hace con los mismos datos de entrenamiento, pero para pruebas más complejas y con datos más variados, se notan más sus

defectos. La implementación de IMPALA hecha junto con los métodos de regularización no solo mejoran el desempeño en entrenamiento, sino también en la evaluación, obteniendo un agente mucho más robusto.

5.2.3 Rainbow con Domain Randomization, IMPALA y Explore Go

Para entrenar al agente Rainbow con randomización, IMPALA y Explore Go, se debe establecer un valor de k, correspondiente a la cantidad máxima de pasos de exploración por episodio. Se hicieron pruebas utilizando k con los valores 200, 400 y 800, intentando encontrar un valor acorde entre la variedad de estados iniciales y no afectar la cantidad de datos disponibles para entrenar.

Se obtiene que con un valor de k de 200 no logra aportar una variedad de estados iniciales significativa que aporte una mejora a la generalización del agente. Con un k igual a 800 se obtuvo un agente entrenado en menos datos, teniendo un rendimiento menor al agente Rainbow con randomización e IMPALA, esto sumado a un largo tiempo de entrenamiento dado los extensos períodos de exploración iniciales, se opta por usar un valor final de k en 400, además de reducir la cantidad de pasos totales para el agente Rainbow, de 75 millones de pasos a 50 millones, reduciendo el tiempo necesario para el entrenamiento.

· **Rendimiento en entrenamiento:** Los resultados obtenidos en cuanto a la recompensa promedio móvil en el entrenamiento (Figura 26) de Rainbow con Explore Go, son los siguientes:

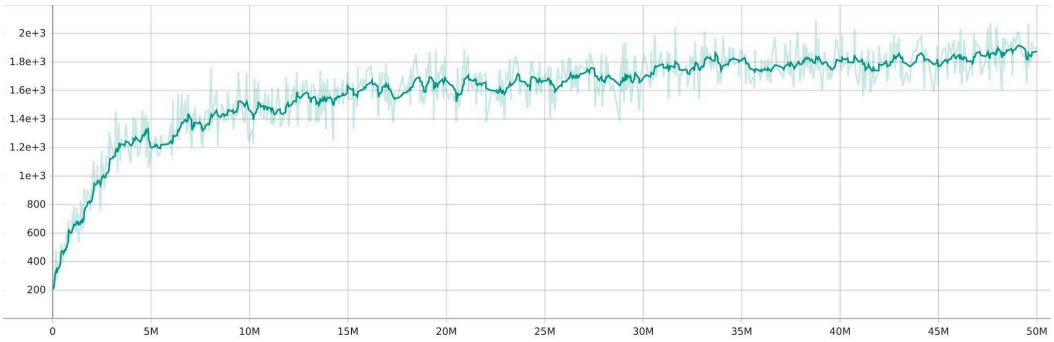

Figura 55. Gráfica de recompensa de Rainbow con randomización, IMPALA y Explore Go.

Dado que el método Explore Go incluye una fase de exploración inicial, el valor máximo posible de recompensa acumulada disminuye. Esto ocurre porque la recompensa comienza a acumularse solo una vez terminada esta fase. Por esto, se opta por no comparar su gráfica de

recompensa promedio durante el entrenamiento con la del resto de los agentes, ya que no sería justo hacerlo con agentes que siempre obtendrán una recompensa promedio mayor.

Dicho esto, la gráfica de recompensa de Rainbow con Explore Go (Figura 55), denota una curva de aprendizaje pronunciada hasta los 10 millones de pasos, luego subiendo progresivamente hasta finalizar el entrenamiento.

Evaluando el agente Rainbow con Explore Go en los niveles de entrenamiento (Figura 26), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, se obtuvieron los siguientes resultados:

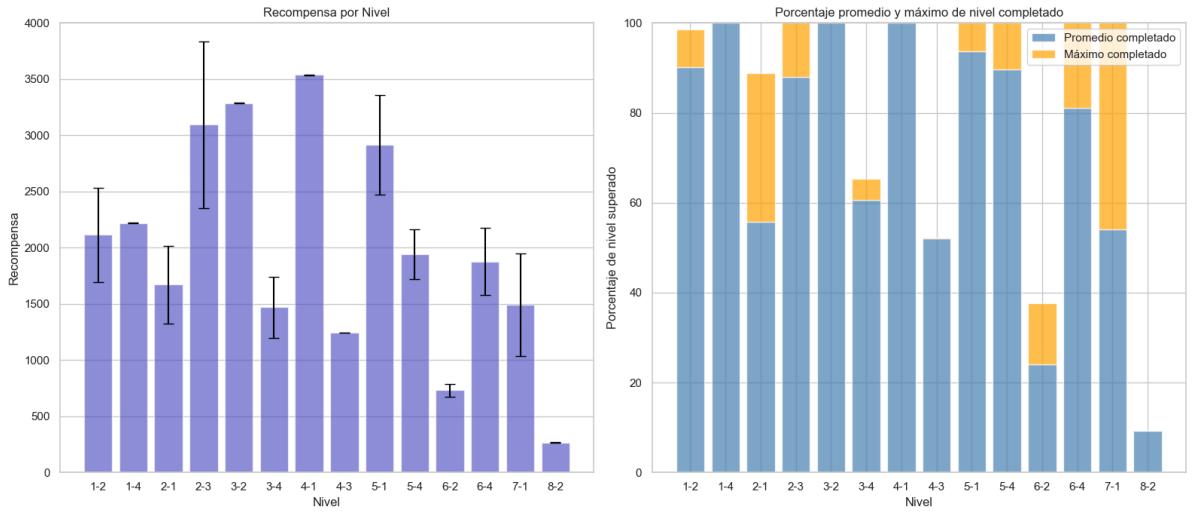

Figura 56. Rendimiento en set de niveles de entrenamiento de Rainbow con randomización, IMPALA y Explore Go.

La política aprendida por el agente Rainbow con randomización, IMPALA y Explore Go logra completar varios niveles del set de entrenamiento (Figura 56), pero se ve con una alta varianza comparado a la política aprendida por el agente Rainbow con randomización e IMPALA, y con dificultades en niveles donde el agente previo sin Explore Go logra superar, como los niveles 2-1, 6-2 y 8-2.

· **Evolución de la pérdida:** Los resultados obtenidos en cuanto a la pérdida en el entrenamiento (Figura 26) del agente Rainbow con Explore Go son los siguientes:

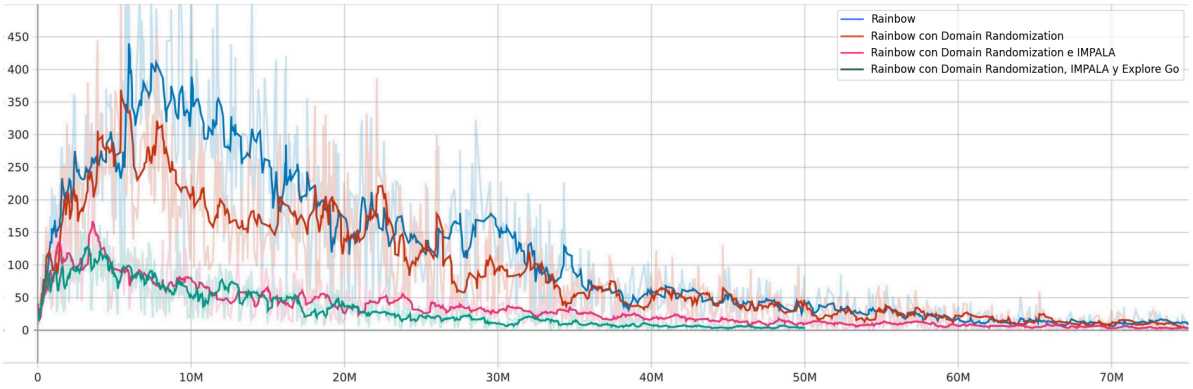

Figura 57. Gráfico de pérdida de Rainbow con randomización, IMPALA y Explore Go.

Los valores de pérdida resultan con una curva muy similar a las de los anteriores agentes (Figura 57), con valores incluso menores que el agente Rainbow con randomización e IMPALA, seguramente debido a que en esta ocasión el agente con Explore Go se entrena con menos datos de cada nivel.

· **Rendimiento en evaluación:** Los resultados del agente Rainbow con IMPALA en los niveles de evaluación (Figura 27), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, son los siguientes:

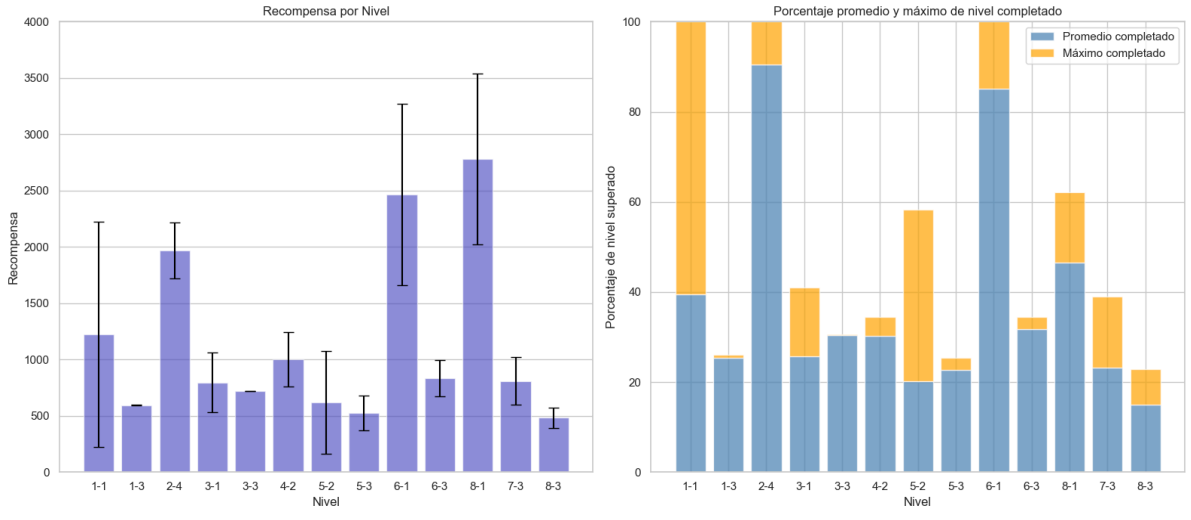

Figura 58. Rendimiento en set de evaluación de Rainbow con randomización, IMPALA y Explore Go.

Los resultados del agente (Figura 58) se muestran similares a los del agente Rainbow con randomización e IMPALA, logrando superar los niveles 1-1, 2-4 y 6-1, pero a diferencia del agente anterior, éste no logra superar el nivel 7-3. Aunque ambos agentes logran superar el nivel 1-1 la varianza en el nivel es bastante alta para el agente Rainbow con Explore Go,

siendo una diferencia estadísticamente significativa ($p < 0.008$) en el nivel, un caso similar pero al contrario pasa en el nivel 2-4, donde el agente con Explore Go logra superar el nivel de forma consistente, avanzando en promedio un 80% del nivel, el agente Rainbow con IMPALA no logra superar el nivel de forma consistente, siendo una mejora estadísticamente significativa ($p < 0.01$), además, el agente con Explore Go logra superar el nivel 2-4 de una forma un tanto distinta en comparación al resto de agentes Rainbow (Figura 59), saliendo por una parte del mapa para saltarse una parte del nivel.

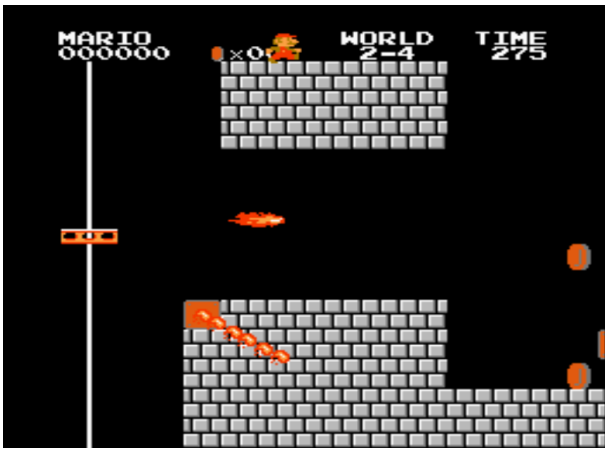

Figura 59. Atajo que toma el agente Rainbow con randomización, IMPALA y Explore Go para superar el nivel 2-4.

El agente Rainbow con Explore Go también obtiene mejoras en niveles difíciles donde ningún agente ha conseguido un gran avance, como en el nivel 4-2, obteniendo un 10% de avance más que el agente con IMPALA, obteniendo una mejora estadísticamente significativa ($p < 0.001$), aun así, sigue lejos de poder superar el nivel.

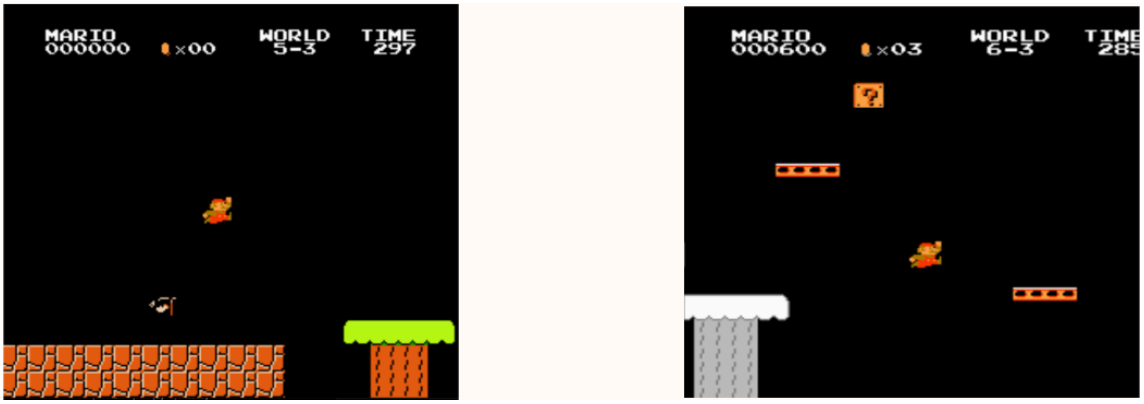

Figura 60. Agente Rainbow con Explore Go superando el primer enemigo en el nivel 5-3 (izquierda), y el máximo que logra en el nivel 6-3 antes de no llegar a una plataforma flotante y caer al vacío.

El agente Rainbow con Explore Go logra mejorar el rendimiento en uno de los niveles con peor rendimiento del agente Rainbow con IMPALA (Figura 60), el nivel 5-3, donde antes el agente chocaría siempre con el primer enemigo del nivel, el agente Rainbow con Explore Go logra superar este enemigo y consigue avanzar un 20% más en el nivel, obteniendo una mejora estadísticamente significativa ($p < 0.0001$). También se consigue una mejora en el nivel 6-3, mejorando el promedio de completado del nivel en un 10%, obteniendo una mejora estadísticamente significativa ($p < 0.003$).

Aún hay niveles donde el rendimiento fue idéntico o muy similar, como el nivel 1-3, 3-1 o el caso más llamativo del nivel 8-3 (Figura 61), donde previamente el agente Rainbow con IMPALA se quedaba estancado en el segundo obstáculo, llevándolo a una gran penalización por estancamiento y paso del tiempo, ahora el agente Rainbow con Explore Go muere en el obstáculo en lugar de quedarse estancado, teniendo un mejor rendimiento en cuanto recompensa que el agente anterior ($p < 0.0001$), pero obteniendo el mismo 15% de nivel superado.

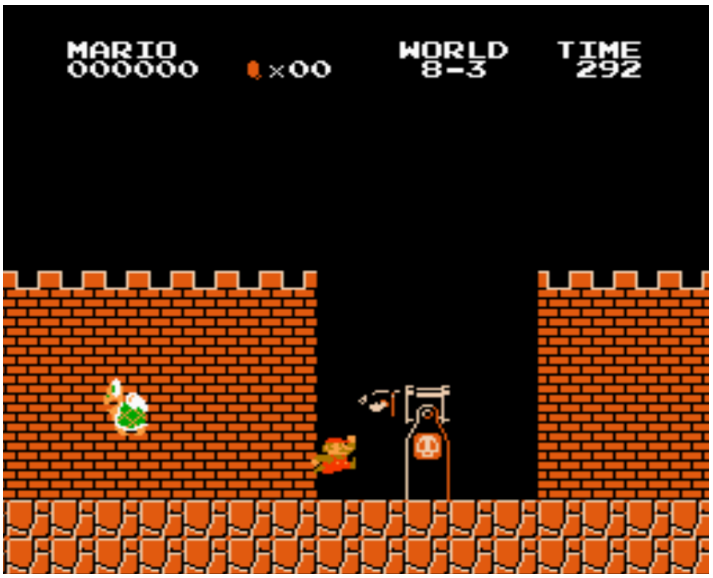

Figura 61. Agente Rainbow con Explore Go antes de morir contra el enemigo sobre su cabeza en el nivel 8-3.

Pese al mal rendimiento que tuvo el agente Rainbow con Explore Go en los niveles de entrenamiento, se obtuvieron mejoras en la generalización con este método, sobretodo en niveles donde los agentes anteriores no obtienen buenos resultados, indicando que la fase exploratoria inicial en cada episodio para aumentar la cantidad de estados iniciales resulta beneficiosa para la generalización en entornos como Super Mario Bros, pese a esto, este

método presenta desventajas como la necesidad de usar un valor de k adecuado, y un aumento considerable en el tiempo de entrenamiento del agente, dado que la fase exploratoria no se considera como pasos realizados por el agente principal.

5.3 PPO y PPO con redes recurrentes

· **Rendimiento en entrenamiento:** Los resultados obtenidos en cuanto a la recompensa promedio móvil en el entrenamiento (Figura 26) de PPO y PPO con redes recurrentes, son los siguientes:

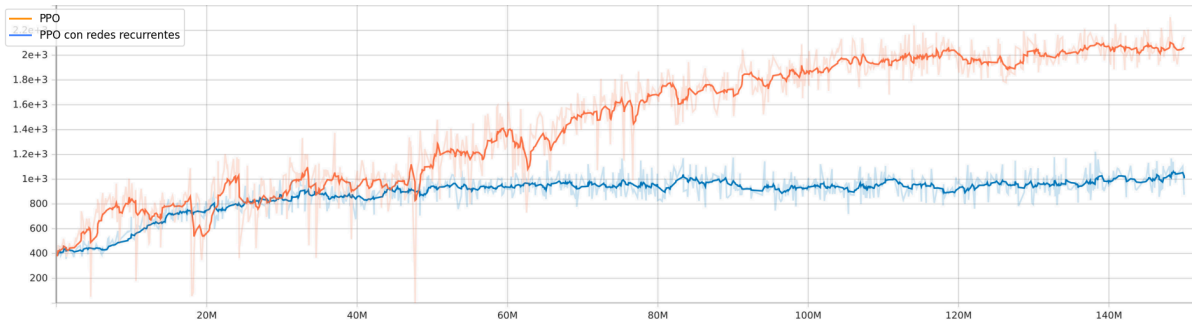

Figura 62. Gráfico de recompensa promedio entre PPO y PPO con redes recurrentes.

La gráfica de recompensa promedio de PPO (Figura 62) muestra una inestabilidad en el entrenamiento, con una bajada significativa cercana a los 20 millones de pasos, estas bajadas siguen ocurriendo durante el entrenamiento pero cada vez en menor medida, llegando a converger cerca de los 120 millones de pasos.

En cuanto a la recompensa promedio obtenida por PPO con redes recurrentes, se observa una curva de aprendizaje mucho más estable, pero con una peor recompensa promedio en todo momento en comparación a PPO, esto puede deberse a la alta variabilidad en las secuencias temporales dado que se entrena en diversos niveles del juego. Cabe destacar que el entrenamiento de PPO con redes recurrentes se hizo sin Framestack, esto debido a que las redes recurrentes son capaces de captar movimientos y relaciones temporales.

Evaluando el agente PPO en los niveles de entrenamiento (Figura 26), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, se obtuvieron los siguientes resultados:

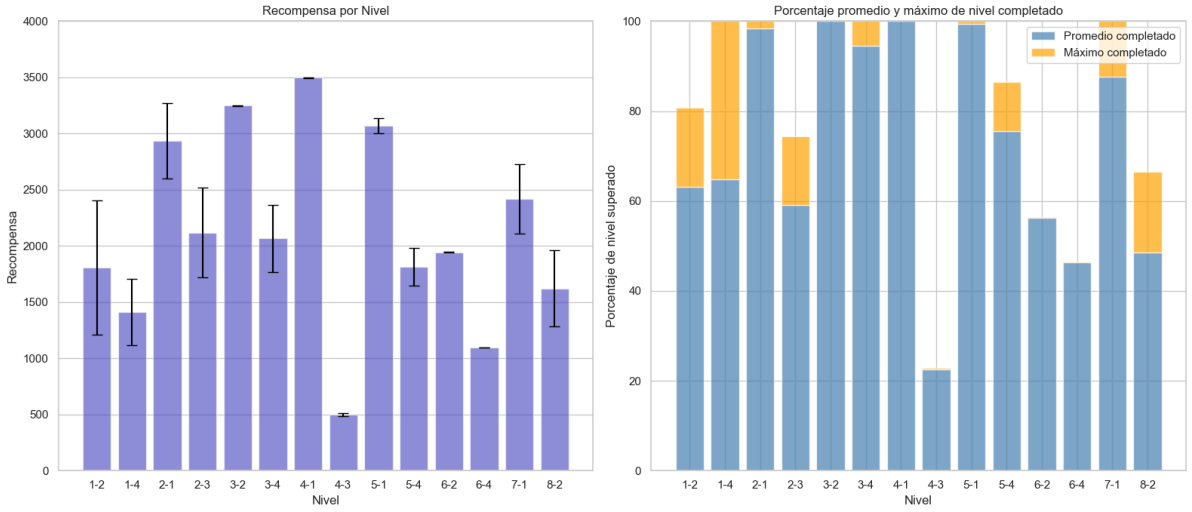

Figura 63. Rendimiento en el set de niveles de entrenamiento de PPO.

Podemos ver que la política aprendida por PPO (Figura 63) logra superar varios niveles, pero pese a entrenarse por más tiempo que Rainbow o DQN, no logra aprender una política más robusta que el agente Rainbow estándar, presentando problemas en los niveles 6-2 y 6-4, donde Rainbow si logra completar dichos niveles.

Evaluando el agente PPO con redes recurrentes en los niveles de entrenamiento (Figura 26), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, se obtuvieron los siguientes resultados:

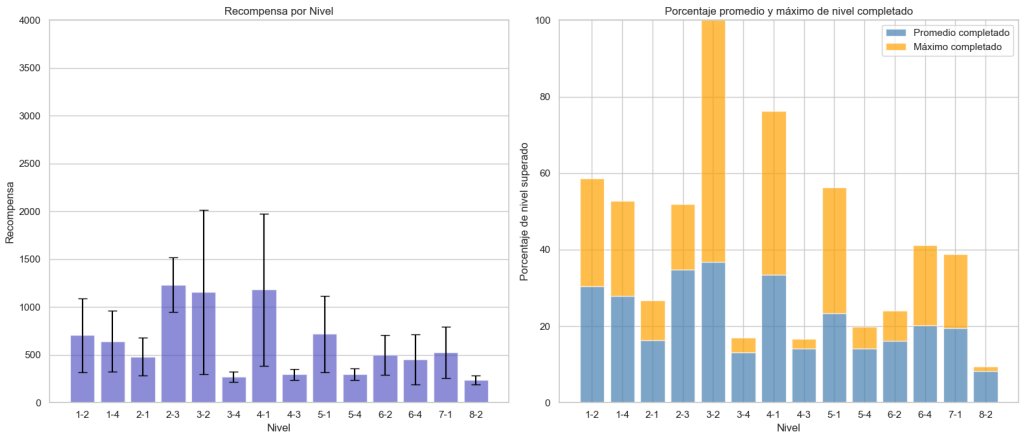

Figura 64. Rendimiento en set de niveles de entrenamiento de PPO con redes recurrentes.

La política aprendida por PPO con redes recurrentes (Figura 64) demuestra que el agente no fue capaz de aprender una política estable, donde incluso en los niveles donde tiene mejor desempeño, aún tiene una gran varianza, obteniendo resultados incluso peores que DQN en varios niveles.

· **Evolución de la pérdida:** Los resultados obtenidos en cuanto a la pérdida en el entrenamiento (Figura 26) de los agentes PPO y PPO con redes recurrentes son los siguientes:

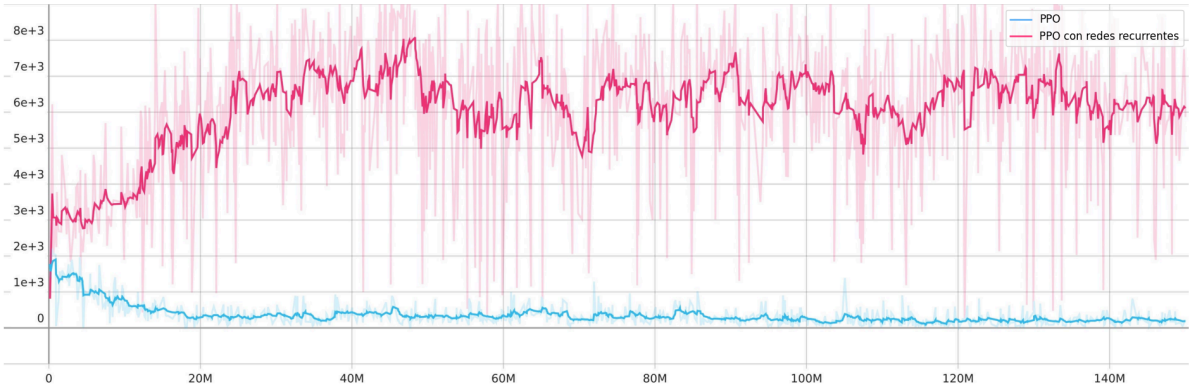

Figura 65. Gráfico de pérdida entre PPO y PPO con redes recurrentes.

Se observa una curva de pérdida normal para PPO (Figura 65), con una tendencia a disminuir durante el tiempo, lo que evidencia su aprendizaje. Distinto es el caso de PPO con redes recurrentes, el cual obtuvo valores muy altos y variables durante todo el entrenamiento. Una posible causa de esto es que se entrena en distintos niveles del juego, lo cual introduce mucha variabilidad a las secuencias temporales, lo cual puede afectar a los estados ocultos de las redes recurrentes, volviendolas inconsistentes, de esta forma los gradientes resultantes son inconsistentes, lo que hace que la retropropagación a través del tiempo sea ineficiente.

· **Rendimiento en evaluación:** Los resultados del agente PPO en los niveles de evaluación (Figura 27), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, son los siguientes:

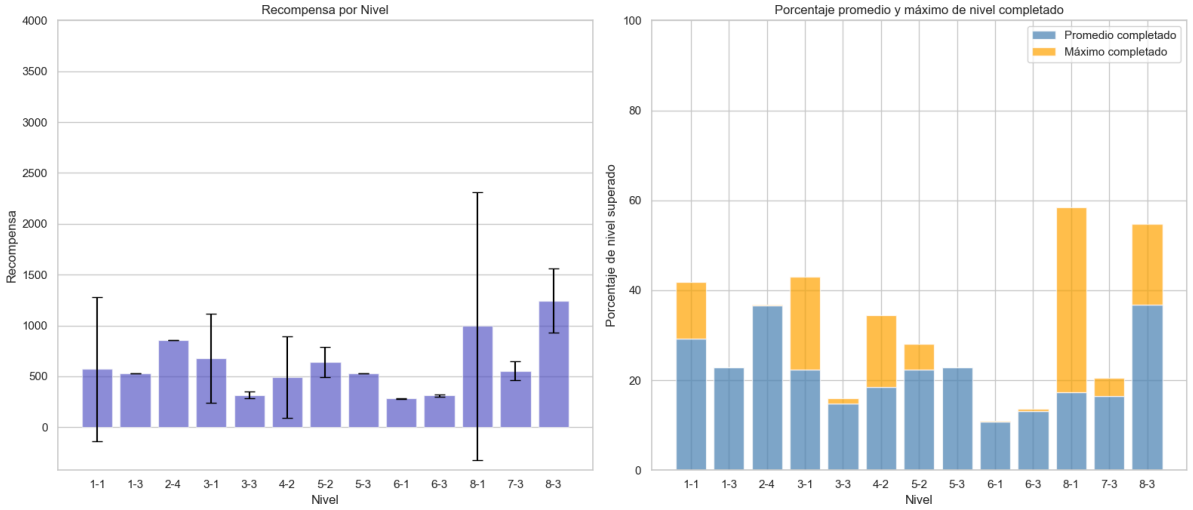

Figura 66. Rendimiento en el set de niveles de evaluación de PPO.

Los resultados de PPO (Figura 66) muestran una gran varianza en los niveles 1-1 y 8-1, donde se observa que a veces el tomar un camino distinto, o una pequeña acción distinta puede llevar al agente a quedarse estancado durante el resto del episodio (Figura 67), una posible causa a esto es las limitaciones de memoria características de PPO, donde a diferencia de Rainbow, el cual puede almacenar experiencias pasadas para luego aprender de ellas con el PER, PPO solo puede aprender de experiencias recientes con un buffer de experiencias mucho más reducido.

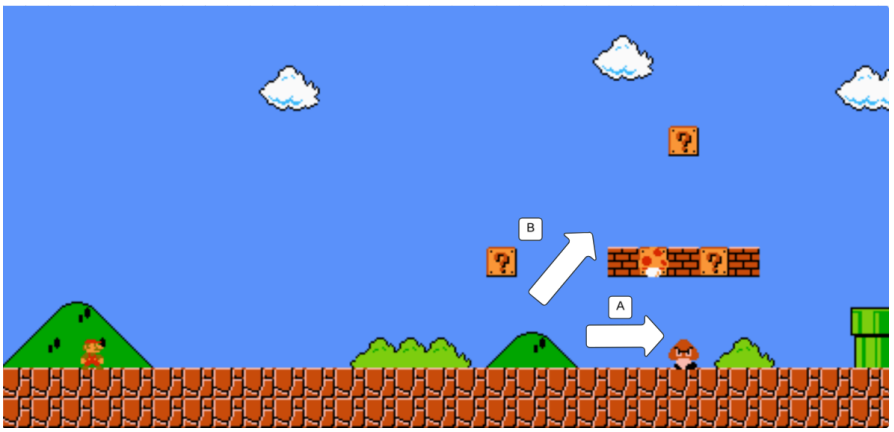

Figura 67. Inicio del nivel 1-1, donde si el agente PPO avanza por el camino A, es decir por debajo de los bloques flotantes y superando al enemigo, es capaz de realizar avances significativos dentro del nivel, pero si

avanza por el camino B, caminando por los bloques flotantes, el agente se queda parado frente a la tubería sin saber cómo continuar en el nivel.

Pese a este problema y a que no logra completar ningún nivel de evaluación, el agente PPO logra un rendimiento más generalizado que Rainbow, sin tener niveles con comportamientos catastróficos ni con recompensas promedio negativas.

PPO logra una mejora estadísticamente significativa con respecto a Rainbow en el nivel 1-3 ($p < 0.0001$), logrando llegar a un 20% de nivel completado en promedio, obteniendo un desempeño similar a Rainbow con IMPALA. Otro caso donde PPO mejora a Rainbow es en el nivel 3-1, con una diferencia estadísticamente significativa ($p < 0.02$), logrando llegar a un 40% de nivel completado como máximo. Otros niveles donde PPO logra una mejora estadísticamente significativa sobre Rainbow son los niveles 5-2 y 8-3 ($p < 0.0001$), en el caso del nivel 8-3 llega a rendir incluso mejor que Rainbow con IMPALA, llegando a casi un 40% de nivel completado en promedio, mientras que Rainbow con IMPALA no logra llegar a un 20% de nivel completado.

Los resultados del agente PPO con redes recurrentes en los niveles de evaluación (Figura 27), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, son los siguientes:

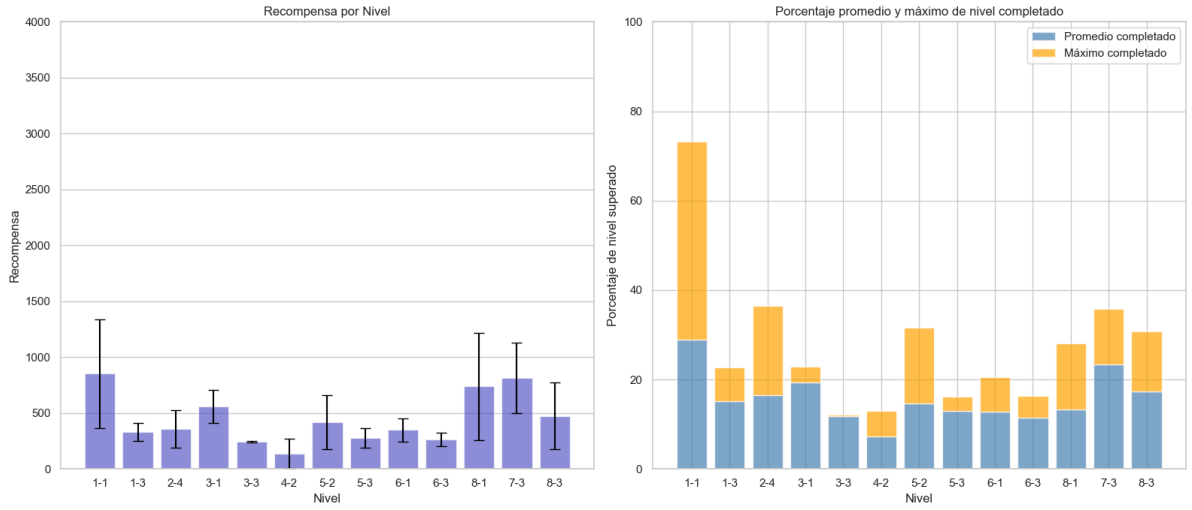

Figura 68. Rendimiento en el set de niveles de evaluación de PPO con redes recurrentes.

Se puede ver a simple vista que los resultados obtenidos por PPO con redes recurrentes (Figura 68) son peores en todos los niveles, a excepción del nivel 1-1, donde se puede ver que el agente con recurrencia logra avanzar más en el nivel, pero sin lograr una diferencia estadísticamente significativa ($p > 0.05$).

Dados los resultados obtenidos tanto en entrenamiento como en evaluación, además de su estabilidad en entrenamiento en comparación a su variante, se elige a PPO como el algoritmo al cual se le aplican las mejoras de generalización.

5.3.1 PPO con Domain Randomization

· **Rendimiento en entrenamiento:** Los resultados obtenidos en cuanto a la recompensa promedio móvil en el entrenamiento (Figura 26) de PPO con randomización, son los siguientes:

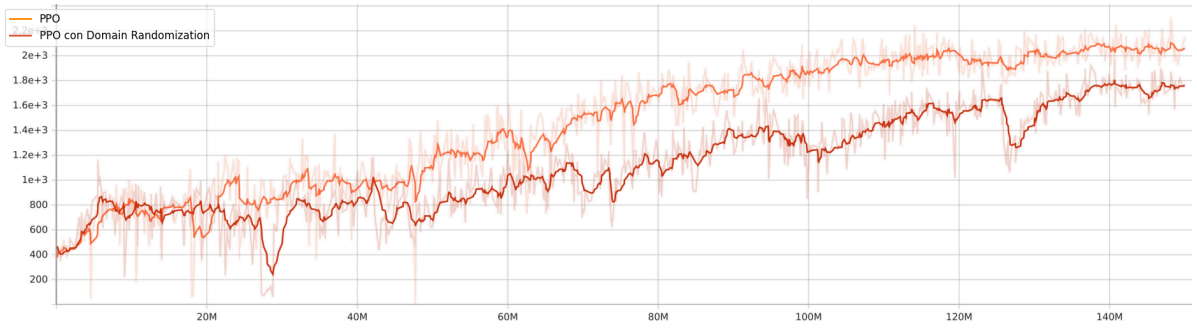

Figura 69. Gráfico de recompensa promedio entre PPO y PPO con randomización de dominio.

Siguiendo una tónica similar a Rainbow con randomización de dominio, la gráfica de recompensa promedio muestra que en casi todo momento el agente PPO con randomización de dominio (Figura 69) obtuvo una peor recompensa promedio durante todo el entrenamiento que el agente PPO estándar, con una caída del rendimiento muy significativa cercana a los 30 millones de pasos debido a cambios muy bruscos en las actualizaciones realizadas a la política, pero de la cual se recupera rápidamente.

Evaluando el agente PPO con randomización en los niveles de entrenamiento (Figura 26), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, se obtuvieron los siguientes resultados:

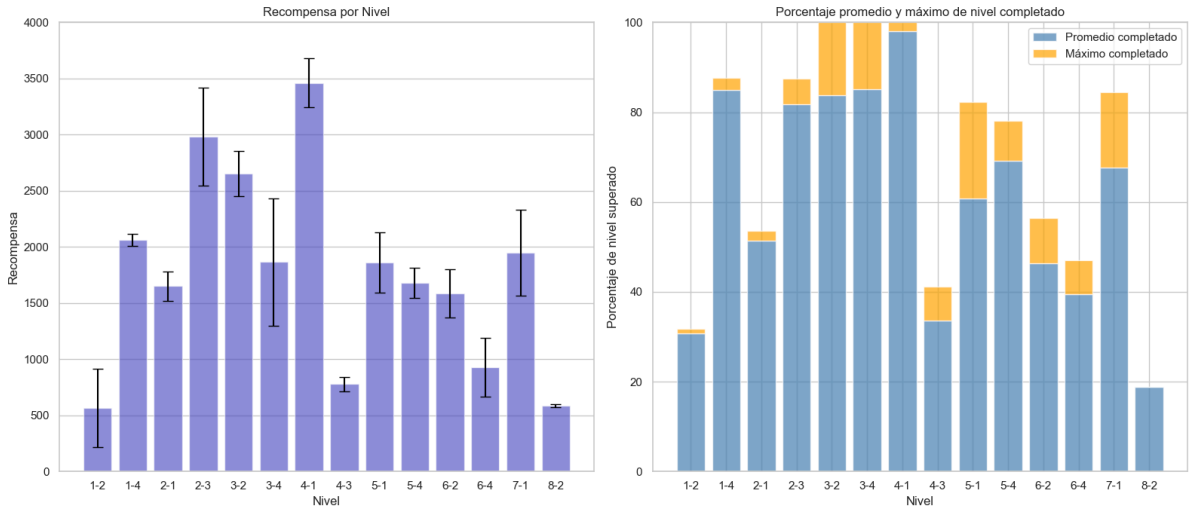

Figura 70. Rendimiento en el set de entrenamiento de PPO con randomización de dominio.

La política aprendida por el agente PPO con randomización (Figura 70) solo logra completar los niveles 3-2, 3-4 y 4-1 en el mejor caso, teniendo dificultades en el resto de niveles como el 8-2 donde no logra superar el 20% del nivel, o el nivel 1-2 donde llega solo al 30% del nivel.

· **Evolución de la pérdida:** Los resultados obtenidos en cuanto a la pérdida en el entrenamiento (Figura 26) del agente PPO con randomización son los siguientes:

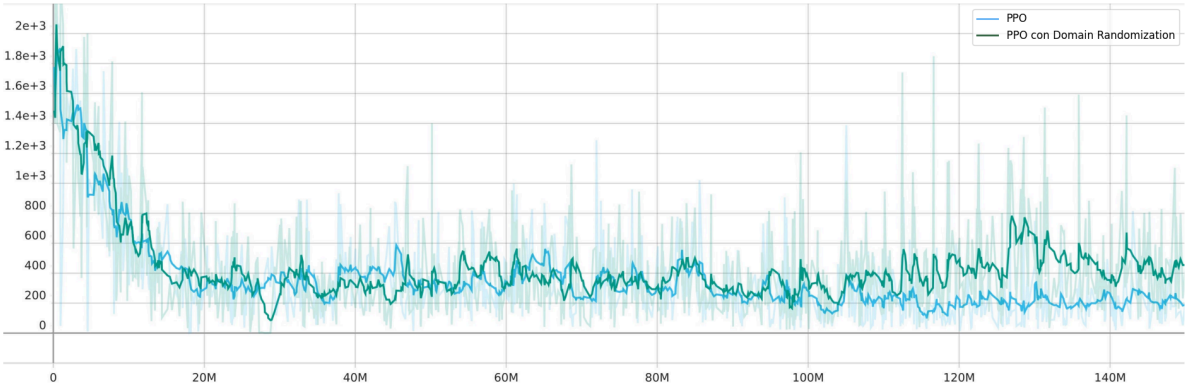

Figura 71. Gráfico de pérdida entre PPO y PPO con randomización.

La curva de pérdida para PPO con randomización (Figura 71) resulta en una tendencia similar a la curva de pérdida de PPO estándar, con una leve subida en los valores de pérdida hacia el final del entrenamiento, cercano a los 130 millones de pasos.

· **Rendimiento en evaluación:** Los resultados del agente PPO con randomización en los niveles de evaluación (Figura 27), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, son los siguientes:

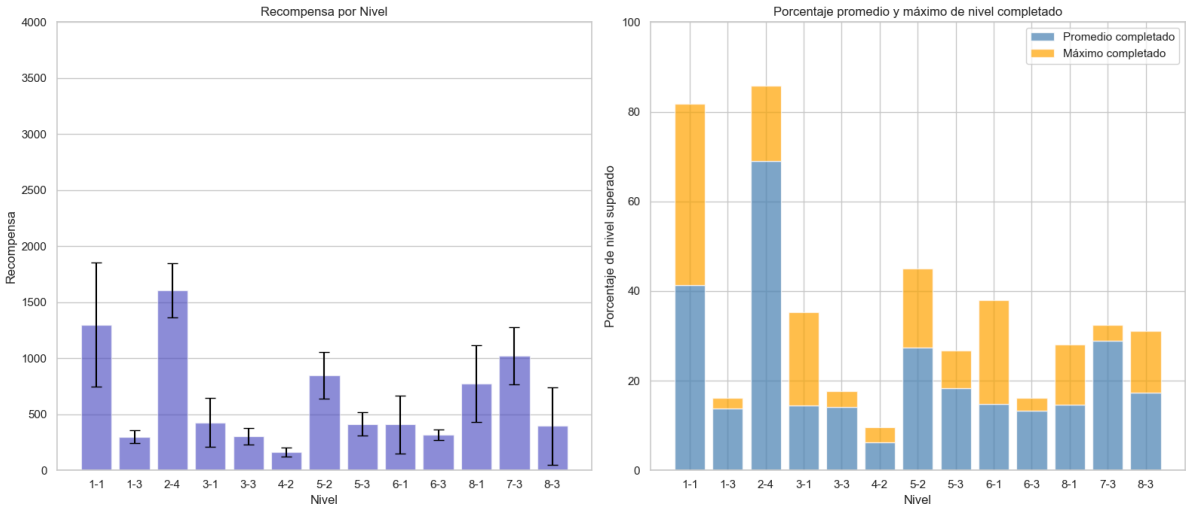

Figura 72. Rendimiento en el set de evaluación de PPO con randomización de dominio.

Pese al peor rendimiento obtenido por PPO con randomización con respecto al agente PPO estándar en los niveles de entrenamiento, en los niveles de evaluación (Figura 72) hay una menor varianza en los niveles 1-1 y 8-1, aunque se puede ver que hay una baja en rendimiento en muchos niveles, como el 8-3, donde el agente PPO estándar logra completar cerca del 40% del nivel, el agente con randomización no llega a completar un 20% del nivel en promedio. Otro nivel donde el agente con randomización empeora en rendimiento es el 8-1, pero dada la alta varianza del agente PPO estándar, esta no logra ser una diferencia estadísticamente significativa ($p > 0.5$).

Aún así, hay niveles donde hay notables mejoras, como el nivel 1-1, donde el agente PPO estándar consigue alcanzar como máximo un 40% del nivel, el agente con randomización logra llegar a un 40% en promedio, y un 80% como máximo, logrando una mejora estadísticamente significativa ($p < 0.02$). También es el caso del nivel 2-4, donde el agente PPO estándar logra un poco menos de 40% del nivel completado, el agente con randomización logra superar el 60% completado en promedio, llegando a un máximo de

80% del nivel completado, con una mejora estadísticamente significativa ($p < 0.0001$). En el nivel 6-1 el agente con randomización parece lograr una mejora con respecto al agente PPO estándar, donde el agente estándar consigue en promedio un 10% del nivel completado, el agente con randomización logra un 15%, pero dada la varianza del agente con randomización, no supone una mejora estadísticamente significativa ($p > 0.1$).

Comparando el agente PPO con randomización y el agente Rainbow con randomización, los resultados son muy similares, siendo PPO ligeramente superior en la mayoría de los casos, o el caso del nivel 2-4, donde el agente Rainbow no logra superar el 20% del nivel, el agente PPO logra superar el 60% en promedio.

Los resultados de ambos agentes con randomización, tanto Rainbow y PPO no obtuvieron una mejora general sustancial con respecto a sus agentes estándar, en algunos casos perjudicando su generalización en algunos niveles, aún así, se puede ver que PPO parece ser menos sensible a la randomización de dominio, obteniendo resultados similares al agente PPO sin randomización.

5.3.2 PPO con Domain Randomization e IMPALA

Para aplicar IMPALA con PPO se hicieron algunas modificaciones según un trabajo de RL realizado en Super Mario [54], separando el extractor de características, dándole un extractor de características propio al Actor y al Crítico (Figura 73), mejorando la estabilidad del entrenamiento a cambio de más costo computacional. Además, se obtuvo que PPO se beneficia mucho de tener una red Crítico más profunda, lo cual ayuda a acelerar la convergencia del agente. Por lo cual se opta por usar una estructura donde tanto el Actor como el Crítico tengan redes separadas, el extractor de características para el Actor se compone de una red Nature [1], es decir, 3 capas convolucionales con activaciones ReLU y una capa lineal, mientras que la red para el Crítico se compone de la red IMPALA descrita en la sección 4.4.2.

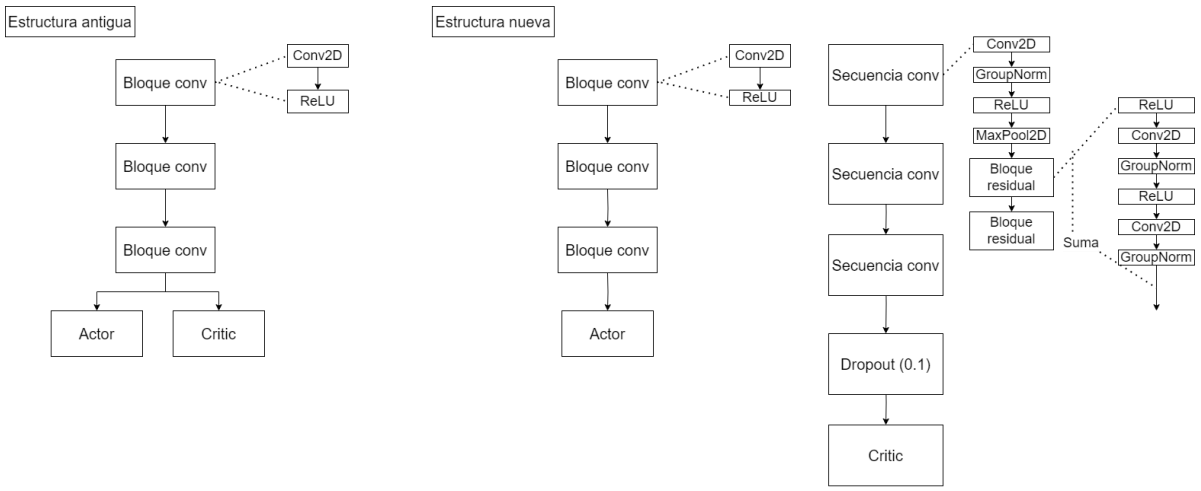

Figura 73. Diagrama comparativo entre las versiones de PPO.

Además se ocupa un "wrapper" de Stable-baselines3, [46] "VecNormalize" el cual normaliza las recompensas, obteniendo un resultado muy positivo al estabilizar los valores de pérdida.

· **Rendimiento en entrenamiento:** Los resultados obtenidos en cuanto a la recompensa promedio móvil en el entrenamiento (Figura 26) de PPO con IMPALA, son los siguientes:

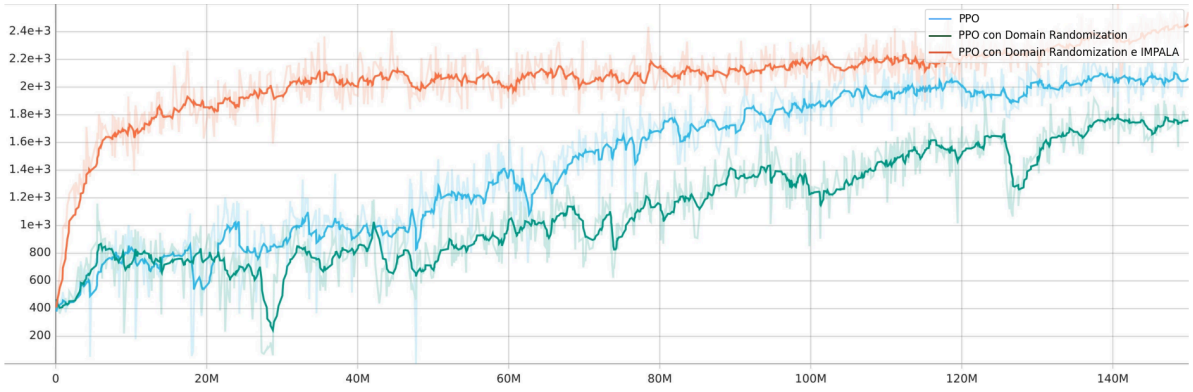

Figura 74. Gráfico de recompensa promedio entre PPO, PPO con randomización de dominio, y PPO con randomización e IMPALA.

Se puede apreciar a simple vista la gran mejora en entrenamiento que supuso los cambios realizados en la estructura de PPO (Figura 74), obteniendo una gran curva muy pronunciada al inicio del entrenamiento, para luego de pasados los 20 millones de pasos se estabilice y empiece a crecer de forma más lenta y progresiva, además se observa que se solucionan los cambios bruscos de política que llevaban a la inestabilidad en varios puntos del entrenamiento de agentes PPO previos.

Evaluando el agente PPO con IMPALA en los niveles de entrenamiento (Figura 26), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, se obtuvieron los siguientes resultados:

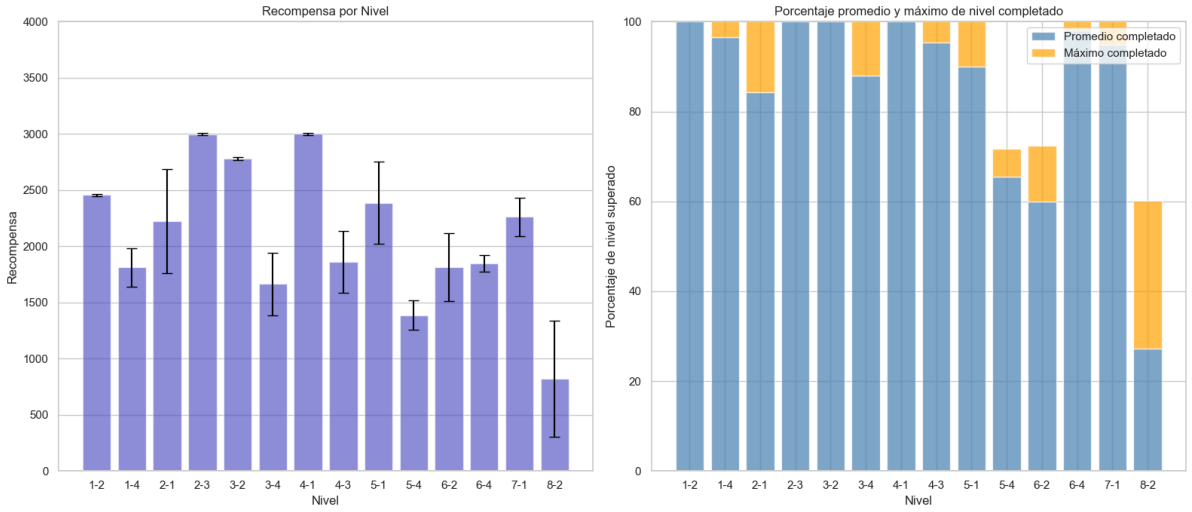

Figura 75. Rendimiento en set de entrenamiento de PPO con randomización e IMPALA.

Se observa también un aumento considerable en el rendimiento en el set de niveles de entrenamiento con respecto a los demás agentes PPO (Figura 75), donde solo no consigue completar los niveles 5-4, 6-2 y 8-2, obteniendo un rendimiento similar a los de agentes Rainbow.

· **Evolución de la pérdida:** Los resultados obtenidos en cuanto a la pérdida en el entrenamiento (Figura 26) del agente PPO con IMPALA son los siguientes:

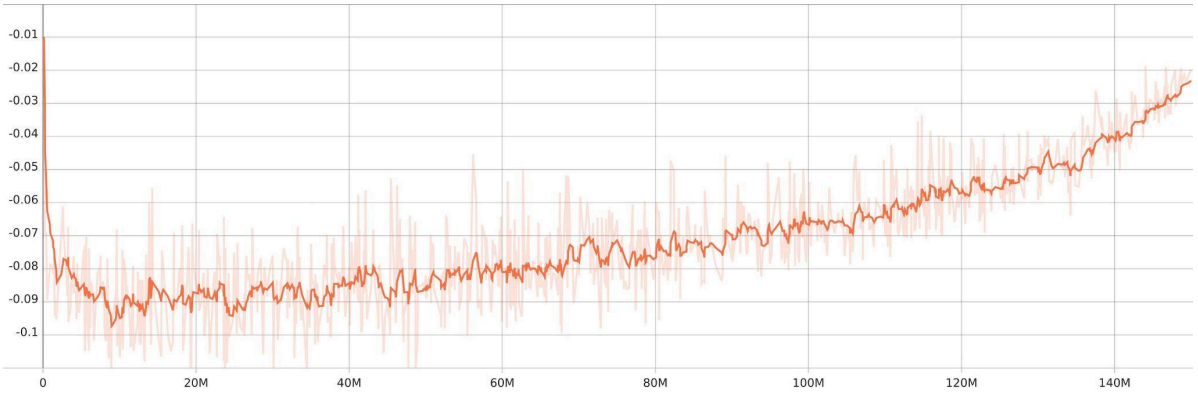

Figura 76. Gráfico de pérdida de PPO con randomización e IMPALA.

Dado que el uso de VecNormalize modifica de forma muy significativa los valores de pérdida del agente PPO con IMPALA, se opta por no compararlo con ningún agente PPO previo, debido a que las escalas de pérdida difieren mucho entre sí.

Los valores de pérdida de PPO con IMPALA (Figura 76) se mantienen en valores negativos durante todo el entrenamiento, indicando que la pérdida de la política predomina sobre la pérdida de la entropía y de valor, y su tendencia hacia converger a 0 indica que el agente aprende correctamente.

· **Rendimiento en evaluación:** Los resultados del agente PPO con IMPALA en los niveles de evaluación (Figura 27), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, son los siguientes:

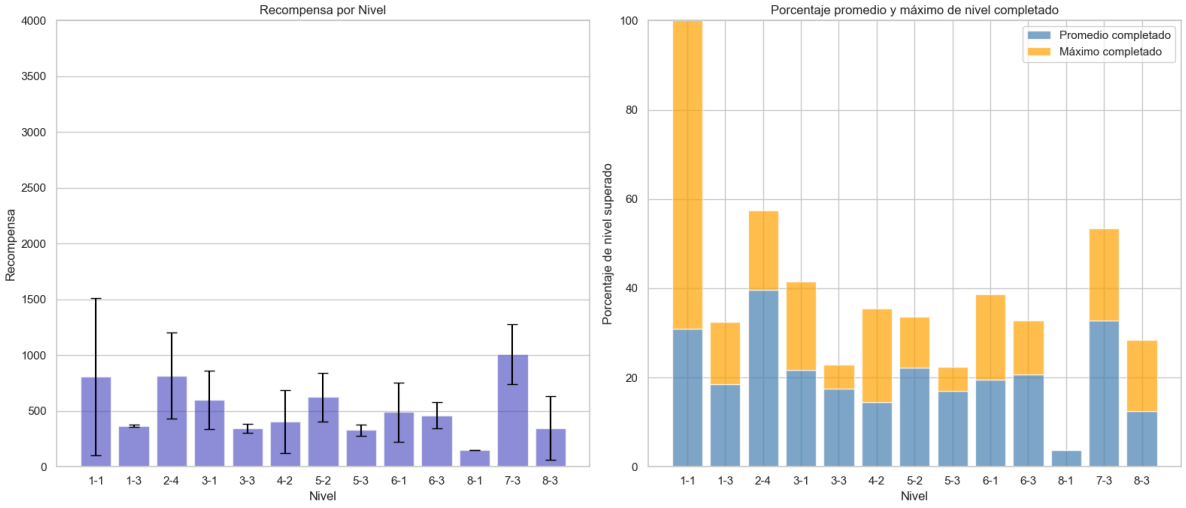

Figura 77. Rendimiento en el set de evaluación de PPO con randomización e IMPALA.

Los resultados obtenidos por el agente PPO con IMPALA en los niveles de evaluación (Figura 77) indican que pese a la gran mejora obtenida en la estabilidad en el entrenamiento, la mejora en la generalización no fue tan notoria en comparación al agente Rainbow con IMPALA, pero este agente ha logrado superar un nivel, específicamente el nivel 1-1, cosa que los demás agentes PPO no han logrado. Aun así, podemos ver el nivel 8-1 con un rendimiento catastrófico (Figura 78), donde el agente no es capaz de superar el primer enemigo del nivel, muriendo constantemente contra él.

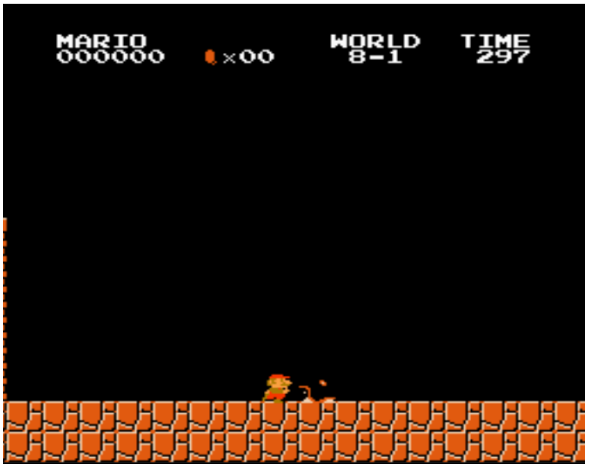

Figura 78. Agente PPO con randomización e IMPALA, justo antes de morir en el nivel 8-1.

Pese al bajo rendimiento general en comparación al agente Rainbow con IMPALA, hay leves mejoras en comparación al agente PPO con randomización, como en los niveles 1-3, 3-1, 3-3, 6-1 y 7-3, las cuales no alcanzan a considerarse mejoras estadísticamente significativas ($p > 0.1$), pocos casos de mejora hay en este agente PPO en comparación al con randomización, siendo estos los niveles 4-2 y 6-3 ($p < 0.02$), como se observa en la Figura 79.

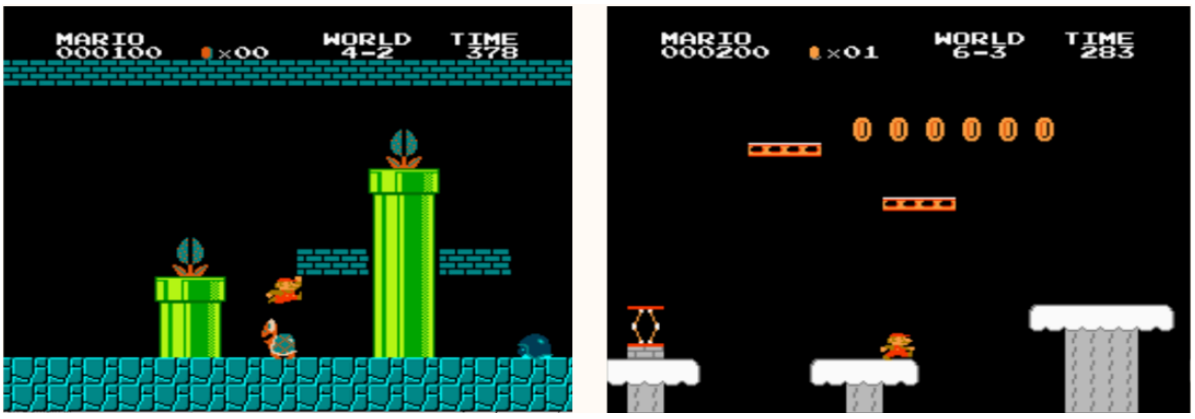

Figura 79. Máximo alcanzado por el agente PPO con IMPALA en los niveles 4-2 (izquierda) y 6-3 (derecha).

El cambio en la estructura de PPO para este agente tuvo tanto ventajas como desventajas, acelerando considerablemente la convergencia en entrenamiento, pero obteniendo pocas mejoras en la generalización. La principal causa de la poca mejora en la generalización es que la estructura IMPALA fue usada en el crítico, cuya función es guiar el entrenamiento del actor, el cual es el que efectivamente aprende la política, y como la red actor es más pequeña en comparación, no logra aprender una política más robusta con mejores

capacidades de generalización. Otra razón posible del bajo rendimiento de este agente es un sobreajuste dado que tanto actor como crítico tienen extractor de características separadas, por lo que el crítico obtiene mejores observaciones de cada estado en comparación al actor, lo cual puede generar que el crítico produzca ventajas sesgadas y que hagan que el actor explote esas ventajas aprendiendo una política sobreajustada a los niveles de entrenamiento.

5.3.3 PPO con Domain Randomization, IMPALA y Explore Go

Para reducir lo máximo posible el tiempo de entrenamiento del agente PPO con Explore Go, se hicieron cambios en la estructura del actor y critico (Figura 80), con el objetivo de obtener una convergencia más rápida y poder entrenar por una menor cantidad de pasos, reduciendo el tiempo necesario de entrenamiento, entrenando en un total de 50 millones de pasos con un valor de k de 400, igual que el agente Rainbow con Explore Go.

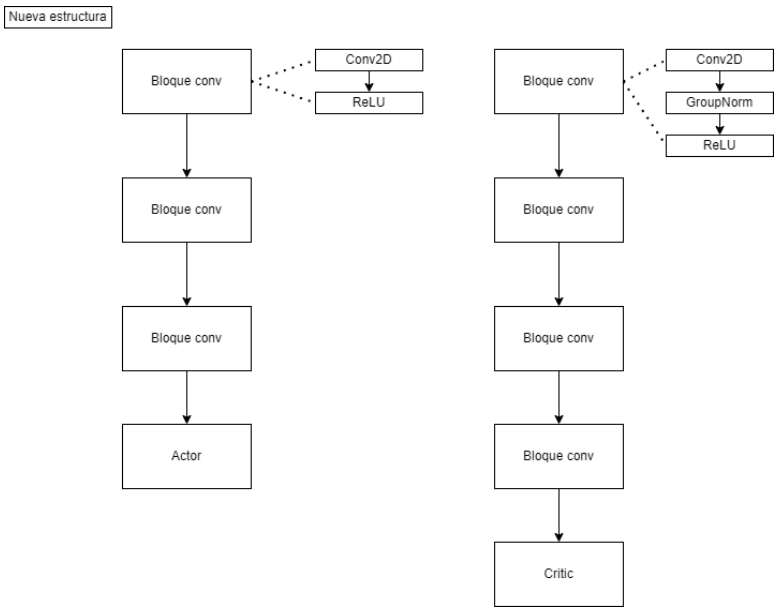

Figura 80. Nueva estructura utilizada para entrenar agente PPO con Explore Go.

El extractor de características del actor sigue siendo una red Nature convencional con cada bloque con 32, 64 y 64 filtros respectivamente, mientras que el extractor de características del crítico incluye un bloque convolucional extra, además de un regularizador Group Norm entre la capa convolucional y la función de activación ReLU de cada bloque. Cada bloque convolucional tiene filtros de 64, 128, 256 y 256, respectivamente.

· **Rendimiento en entrenamiento:** Los resultados obtenidos en cuanto a la recompensa promedio móvil en el entrenamiento (Figura 26) de PPO con Explore Go, son los siguientes:

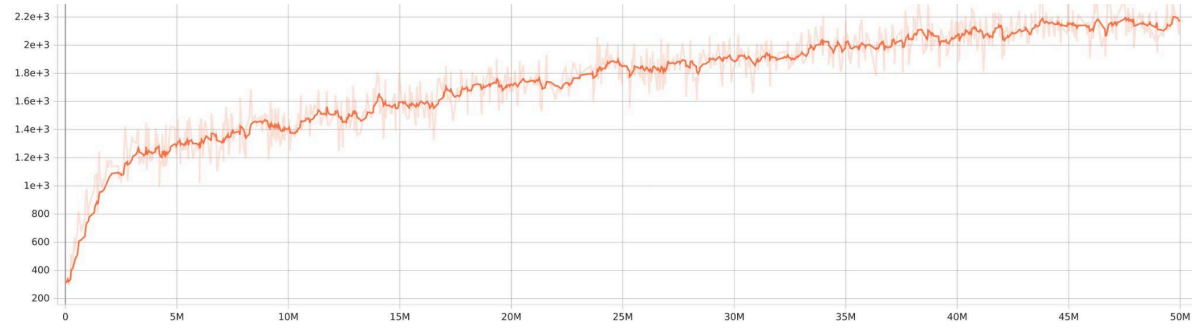

Figura 81. Gráfico de recompensa promedio de PPO con randomización, IMPALA y Explore Go.

Por las mismas razones que el agente Rainbow con Explore Go, no se compara el gráfico de recompensa promedio con los demás agentes PPO. El gráfico muestra una curva muy pronunciada durante los primeros 5 millones de pasos de entrenamiento (Figura 81), para que durante el resto del entrenamiento una curva con aumentos lentos pero constantes de recompensa, indicando un entrenamiento exitoso y estable.

Evaluando el agente PPO con Explore Go en los niveles de entrenamiento (Figura 26), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, se obtuvieron los siguientes resultados:

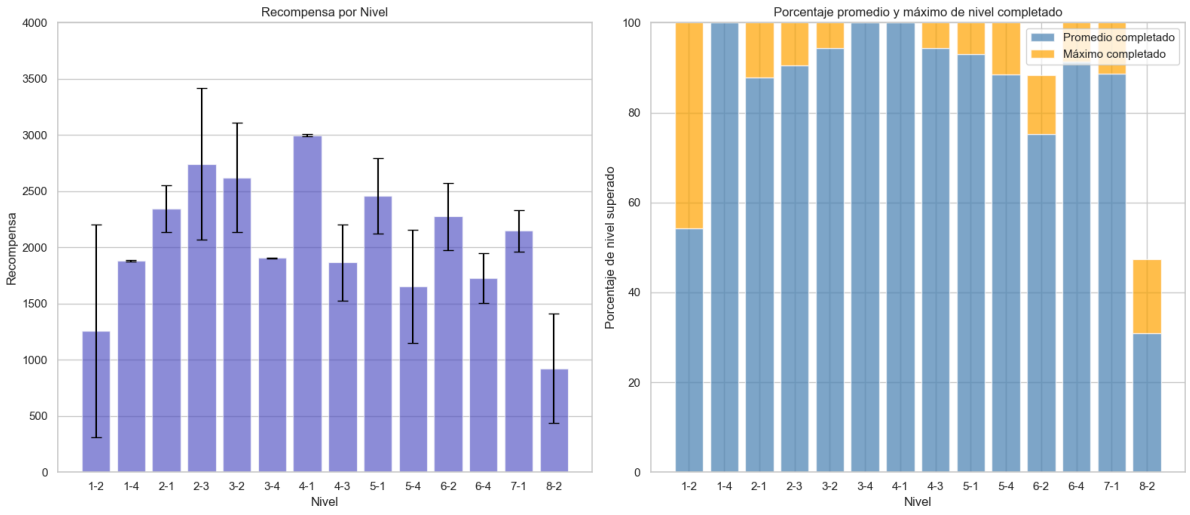

Figura 82. Rendimiento en set de entrenamiento de PPO con randomización, IMPALA y Explore Go.

Se puede ver en la Figura 82 que la política aprendida logra superar gran parte de los niveles de entrenamiento, fallando solo en los niveles 6-2 y 8-2, aún así realizando avances significativos en dichos niveles, además, aunque logra superar el nivel 1-2, el agente obtuvo una alta varianza. Incluso este agente logra superar el nivel 4-3 el cual resulta en un nivel difícil que ningún agente Rainbow pudo superar.

· **Evolución de la pérdida:** Los resultados obtenidos en cuanto a la pérdida en el entrenamiento (Figura 26) del agente PPO con Explore Go son los siguientes:

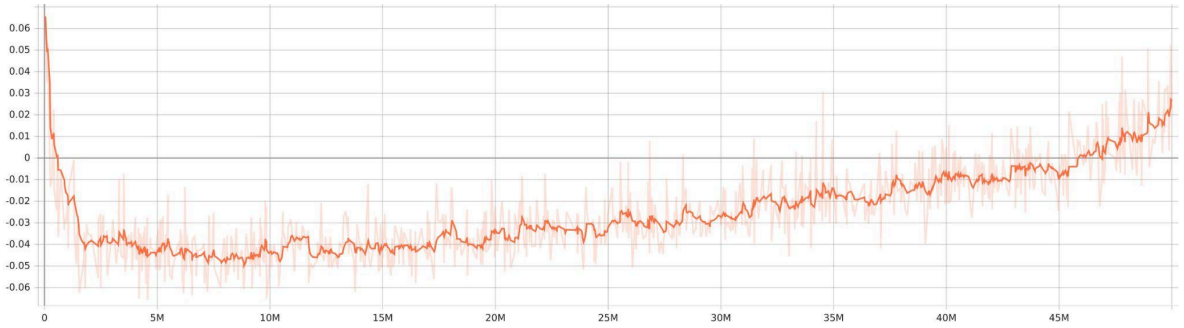

Figura 83. Gráfico de pérdida de PPO con randomización, IMPALA y Explore Go.

Los valores de pérdida de PPO con Explore Go (Figura 83) muestran una curva bastante convencional, con una pendiente pronunciada hasta los 5 millones de pasos, la cual coincide con el gran aumento de recompensa del agente, periodo donde el agente aprende a jugar en los diferentes entornos. La tendencia durante gran parte del entrenamiento fue obtener valores de pérdida negativos, indicando la predominancia de la pérdida de la política por sobre los valores de pérdida de entropía y de la función de valor. Durante el tramo final del entrenamiento, desde los 45 millones de pasos, la pérdida parece ir en un constante aumento, lo cual puede ser un indicio de un sobreajuste a los niveles de entrenamiento, dado que las recompensas obtenidas durante el entrenamiento se mantuvieron de forma estable durante esta fase del entrenamiento.

· **Rendimiento en evaluación:** Los resultados del agente PPO con IMPALA en los niveles de evaluación (Figura 27), en cuanto a recompensa promedio por nivel, desviación estándar y porcentaje de completitud promedio y máximo por nivel, son los siguientes:

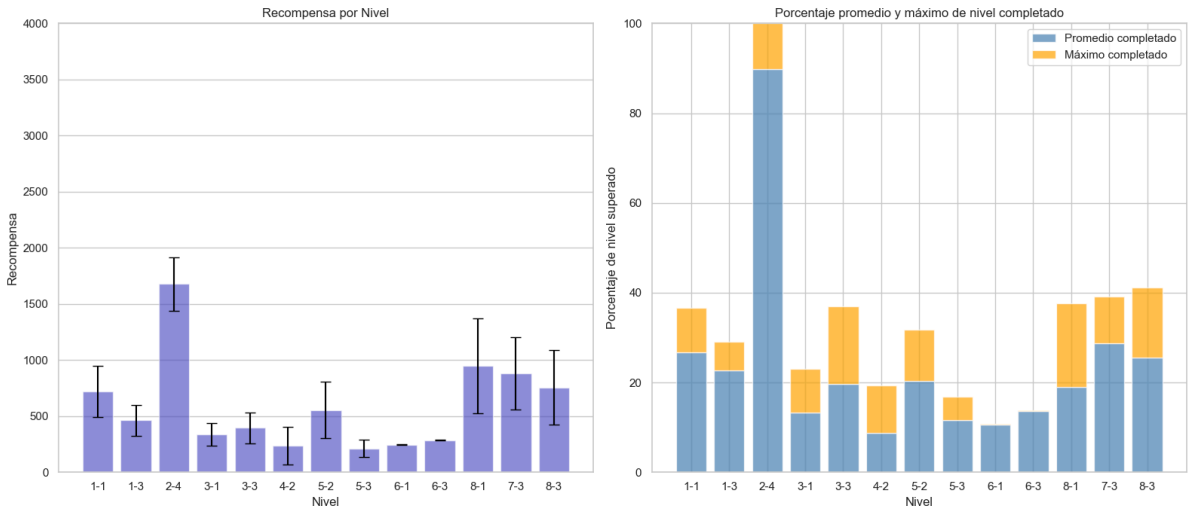

Figura 84. Rendimiento en el set de evaluación de PPO con randomización, IMPALA y Explore Go.

Los resultados obtenidos por el agente PPO con Explore Go (Figura 84) en los niveles de evaluación parecen evidenciar un sobreajuste dado los resultados obtenidos en los niveles de entrenamiento, además del aumento en los valores de pérdida en los tramos finales del entrenamiento.

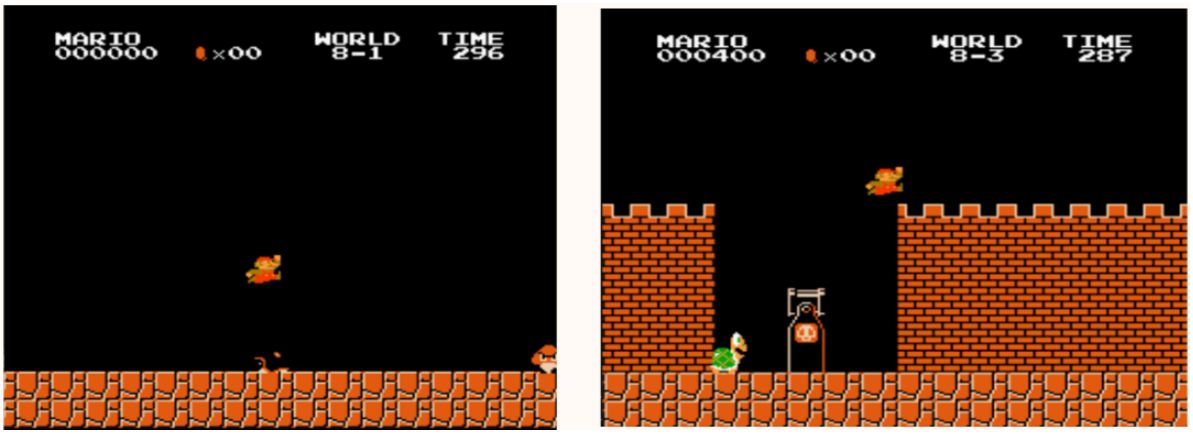

Figura 85. Agente PPO con Explore Go superando puntos de estancamiento de otros agentes en los niveles 8-1 (izquierda) y 8-3 (derecha).

Aún así, hay niveles donde el agente PPO con Explore Go obtiene mejoras con respecto al agente PPO con IMPALA, el primero y más evidente es en el nivel 2-4, siendo capaz de superar el nivel de forma consistente ($p < 0.0001$), además, en el nivel 8-1 (Figura 85), donde el agente PPO con IMPALA no logra superar al primer enemigo del nivel, el agente PPO con Explore Go logra superarlo ($p < 0.0001$) y avanzar un 15% en promedio y un 37% como máximo en el nivel. El último nivel con mejora significativa es el nivel 8-3 ($p < 0.01$), donde el agente PPO con Explore Go logra una mejora del 12% en cuanto al avance promedio, logrando superar los obstáculos iniciales (Figura 85) que han resultado en una barrera importante que no permitía el avance a agentes previos.

Los resultados de los agentes PPO con IMPALA y PPO con Explore Go con los cambios estructurales realizados muestran una gran mejora en la estabilidad del entrenamiento, obteniendo cambios estables en la política luego de cada actualización, a diferencia de los agentes PPO estándar o PPO con randomización. Además, estos resultados sugieren que para obtener una mejor generalización algorítmica en PPO se debe hacer un "sacrificio", si se quiere buscar una mejor convergencia del agente, se sacrifica en la capacidad de generalización, arriesgando un posible sobreajuste. Mientras que al parecer aprendizajes más lentos como los de los agentes PPO estándar o PPO con randomización parecen obtener una mejor generalización a costa de una menor eficiencia de muestreo, necesitando de más pasos de entrenamiento para converger a una solución.

5.4 Análisis de resultados

En cuanto a los agentes entrenados, los métodos de generalización y en cuanto a los resultados obtenidos, se puede decir:

· **DQN y Rainbow:** Rainbow obtuvo un evidente mejor desempeño que DQN en entrenamiento, mientras que en evaluación se puede ver un sobreajuste en algunos niveles por parte de Rainbow obteniendo resultados negativos en varios niveles, pero resultados muy positivos en otros, como el nivel 1-1 o 7-3, o incluso superando el nivel 2-4. La diferencia de rendimiento y generalización es gracias a los componentes mejorados de Rainbow, como PER y Double Q-learning, haciendo un algoritmo más potente pero bastante más pesado en cómputo.

· **PPO y PPO recurrente:** Las redes recurrentes no parecen útiles para esta configuración de entrenamiento, incluso degradando bastante el rendimiento, el principal motivo siendo la alta varianza de las secuencias temporales entre un nivel y otro, haciendo que las redes recurrentes no puedan captar patrones temporales, perjudicando el entrenamiento y el rendimiento en evaluación. En cuanto a PPO, se ve un aprendizaje inestable, marcado por cambios bruscos en la política, demorando su convergencia, además de una baja eficiencia de muestreo en comparación a Rainbow, lo cual hizo necesario aumentar la cantidad de pasos para obtener mejores resultados y lograr que converja.

· **NEAT e HyperNEAT:** En general no se obtuvieron buenos resultados tanto con NEAT como HyperNEAT, obteniendo resultados muy similares entre ellos, pese a esto, los agentes logran avances, sin conseguir completar ningún nivel. Es evidente que al no ocupar redes neuronales profundas, sin gradientes ni retropropagación, estos agentes están en desventaja frente a algoritmos de aprendizaje por refuerzo profundo.

· **Domain Randomization:** La randomización se demostró que reduce el sobreajuste, y en el caso de Rainbow incluso sirve para reducir los valores de pérdida durante el entrenamiento, pero los resultados obtenidos por este método por sí solo no fueron favorables, sobretodo para Rainbow, donde se obtuvo una evidente baja en el rendimiento en el set de niveles de evaluación. Distinto es el caso de la randomización en PPO, donde no se obtuvo una baja muy notable en el rendimiento en evaluación, en muchos niveles siendo incluso mejor que su agente estándar, PPO resulta menos sensible a la randomización de dominio, esto debido a que es un algoritmo on-policy, al aprender solo de experiencias recientes, se puede adaptar más fácilmente a las variaciones de Domain Randomization.

· **IMPALA:** Este método fue el que más impacto obtuvo en general, sobre todo en Rainbow, donde se obtuvieron mejoras en el rendimiento de niveles de entrenamiento, se redujeron los valores de pérdida, y una mejora sustancial a la generalización en los niveles de evaluación. En PPO, junto con las modificaciones hechas en la estructura de Actor y Crítico, se obtuvieron mejoras en la eficiencia de entrenamiento junto con una mejor estabilidad en entrenamiento, pero la mejora en la generalización no fue tan notable a diferencia de Rainbow.

· **Explore Go:** Pese al considerable aumento en el tiempo de entrenamiento proveniente de la fase exploratoria, este método demostró ser efectivo para mejorar la generalización sobre

todo a los niveles más complejos del set de evaluación, reduciendo el sobreajuste a los niveles de entrenamiento. El éxito de este método indica que la exploración y la variedad de estados iniciales es un factor importante a considerar en el aprendizaje por refuerzo y la generalización.

· **Rainbow y PPO:** Comparando ambos algoritmos, PPO mostró un comportamiento más estable y generalizado que Rainbow en evaluación, sin tener niveles con comportamientos catastróficos ni recompensas negativas, mientras que Rainbow obtuvo mejores máximos de rendimiento pero mostrando sobreajuste en algunos niveles.

Por otra parte, Rainbow logra un aprendizaje más rápido y estable que PPO, pero con una política sobreajustada a los niveles de entrenamientos, la cual gracias a los métodos de generalización probados se pudo evidenciar un menor sobreajuste y una mejora notable en la generalización, logrando corregir las recompensas acumuladas negativas y los comportamientos catastróficos. Mientras que PPO tuvo un aprendizaje más lento e inestable que Rainbow, pero obteniendo una política más robusta, con los cambios estructurales realizados a PPO, con la finalidad de obtener un rendimiento en entrenamiento similar a Rainbow, se vieron perjudicados los métodos IMPALA y Explore Go, obteniendo políticas con más sobreajuste que los agentes PPO previos.

Los métodos de generalización probados demostraron que métodos para la generalización con distintos enfoques pueden ser combinados para obtener mejores resultados, y que pueden ser efectivos tanto para algoritmos on-policy como off-policy.

6. Conclusiones

La generalización en el aprendizaje por refuerzo es a día de hoy un campo de investigación activo muy importante para posibilitar una aplicación de agentes de aprendizaje por refuerzo más robustos en situaciones más cotidianas, donde la capacidad de transferir el conocimiento adquirido en un entorno hacia nuevos entornos no vistos previamente es clave para obtener un comportamiento adaptable y seguro, permitiendo que los agentes no solo memoricen patrones específicos del entorno de entrenamiento, sino que aprendan representaciones más abstractas que les permitan responder acorde a cambios no previstos.

A lo largo de esta memoria de título se entrenaron y evaluaron agentes de aprendizaje por refuerzo y neuroevolutivos con el objetivo de analizar y mejorar su capacidad de generalización en el entorno de Super Mario Bros. Para ello se implementaron tres métodos para la generalización con distintos enfoques, Domain Randomization, arquitectura de red neuronal IMPALA y Explore Go, aplicados sobre DQN y PPO. Estos métodos permitieron estudiar cómo la variación en los estados del entrenamiento, la estructura de las redes neuronales y la exploración influyen en la capacidad de los agentes para adaptarse a nuevos entornos no vistos. La comparación entre algoritmos mostró que PPO logró la generalización más robusta, mientras que Rainbow mejoró notablemente al aplicar los métodos propuestos, mientras que NEAT no obtuvo un rendimiento comparable a DQN o PPO.

En general, los métodos para la generalización demostraron ser efectivos y combinables, favoreciendo la generalización, por lo general a costa de un mayor costo computacional o tiempo de entrenamiento.

Considerando estos resultados, el objetivo general de la memoria fue cumplido, así como los objetivos específicos, se implementaron y validaron métodos de generalización, se compararon algoritmos en cuanto a desempeño y generalización, y se identifica como el mejor algoritmo a PPO en cuanto a generalización, y Rainbow en cuanto a los mejores resultados.

6.1 Trabajo futuro

Existen muchas formas de conseguir una mejor generalización, en este trabajo solo se han visto unas cuantas, por lo que aún queda espacio para seguir experimentando con métodos para obtener mejoras de la generalización mediante observaciones y recompensas.

El entorno seleccionado si bien es bueno para analizar la transferencia entre niveles, limita un poco dado la poca cantidad de niveles del juego, lo que limita la diversidad de experiencias posibles, de ahí la necesidad de implementar métodos como Domain Randomization. Por ello sería una buena idea probar estos métodos en otros entornos procedurales, los cuales ofrecen una mayor cantidad de variedad de escenarios, permitiendo medir mejor la generalización. Asimismo, se podría seguir experimentando con diferentes combinaciones de niveles de entrenamiento y de evaluación dentro del mismo juego, para evaluar cómo la selección del conjunto de niveles afecta el rendimiento en la evaluación.

En cuanto a los algoritmos, sería interesante probar una variante de PPO de estudios recientes [55], la cual incluye un experience replay y clipping adaptativo, fue probado en entornos de Super Mario Bros y tuvo muy buenos resultados, no se pudo implementar por falta de tiempo además de falta de documentación sobre los cambios distintivos de la implementación, por lo que se optó por seguir con PPO recurrente.

Finalmente, para NEAT e HyperNEAT, algo que vale la pena probar sería usar Novelty Search [56], para fomentar más la diversidad de comportamientos, permitiendo explorar más en cada nivel, haciendo que el agente no converja antes y encuentre mejores soluciones.

Referencias

1. V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.
2. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.
3. K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," Evolutionary Computation, vol. 10, no. 2, pp. 99-127, 2002.
4. S. Lang, T. Reggelin, J. Schmidt, M. Müller, and A. Nahhas, "NeuroEvolution of augmenting topologies for solving a two-stage hybrid flow shop scheduling problem: A comparison of different solution strategies," Expert Systems with Applications, vol. 172, p. 114666, Feb. 2021.
5. M. Hessel et al., “Rainbow: Combining Improvements in Deep Reinforcement Learning,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, Apr. 2018, doi: https://doi.org/10.1609/aaai.v32i1.11796.
6. H. van Hasselt, "Double Q-learning," in Advances in Neural Information Processing Systems (NIPS), 2010, pp. 2613-2621.
7. T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," presented at the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2016. [Online]. Available: https://arxiv.org/abs/1511.05952
8. Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning" in Proceedings of the International Conference on Machine Learning (ICML), New York, USA, 2016.
9. R. S. Sutton, "Learning to predict by the methods of temporal differences," Machine Learning, vol. 3, no. 1, pp. 9–44, Aug. 1988.
10. M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in Proceedings of the 34th International Conference on Machine Learning (ICML), Sydney, Australia, 2017, pp. 449–458.
11. M. Fortunato, M. G. Azar, B. Piot, J. Menick, M. Hessel, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy networks for exploration," in Proceedings of the International Conference on Learning Representations (ICLR), Vancouver, Canada, 2018.

12. D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in Proceedings of the 34th International Conference on Machine Learning (ICML), Sydney, Australia, 2017.
13. K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks," Artificial Life, vol. 15, no. 2, pp. 185–212, Apr. 2009, doi: https://doi.org/10.1162/artl.2009.15.2.15202.
14. R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed., Cambridge, MA, USA: MIT Press, 2018.
15. Proximal Policy Optimization (PPO)," OpenAI Spinning Up. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html. [Accessed: Nov. 7, 2024].
16. Artificial Neural Networks and its Applications," GeeksforGeeks. Available: https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/ [Accessed: Nov. 7, 2024].
17. Deep Q-Networks (DQN), Dilith Jay. Available: https://dilithjay.com/blog/dqn. [Accessed: Nov. 7, 2024].
18. Q-Learning vs. Deep Q-Learning, ResearchGate. Available: https://www.researchgate.net/figure/Q-Learning-vs-Deep-Q-Learning_fig1_351884746 [Accessed: Nov. 7, 2024].
19. C. J. C. H. Watkins and P. Dayan, "Q-learning," Machine Learning, vol. 8, no. 3-4, pp. 279–292, 1992.
20. M. Kubat, An Introduction to Machine Learning, 2nd ed., Cham, Switzerland: Springer, 2017.
21. A. Hastak, "Reinforcement Learning," Medium. [Online]. Available: https://aishwaryahastak.medium.com/reinforcement-learning-bb34d8a369a2. [Accessed: Jan. 8, 2025].
22. "Types of Machine Learning," ResearchGate. [Online]. Available: https://www.researchgate.net/figure/Types-of-machine-learning-Machine-learning-encompasses-three-main-types-supervised_fig1_373535780. [Accessed: Jan. 8, 2025].
23. R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, "A Survey of Zero-shot Generalisation in Deep Reinforcement Learning," Journal of Artificial Intelligence Research, vol. 76, pp. 201–264, Jan. 2023

24. C. C. Aggarwal, Neural Networks and Deep Learning: A Textbook, 2nd ed., Cham, Switzerland: Springer, 2023. [Online]. Available: https://doi.org/10.1007/978-3-031-29642-0.
25. Codex, "Kernels & filters in convolutional neural network (CNN): Let's talk about them," Medium, [Online]. Available: https://medium.com/codex/kernels-filters-in-convolutional-neural-network-cnn-lets-talk-about-them-ee4e94f3319. [Accessed: Jan 8, 2025].
26. "Convolution Neural Network Deep Learning," Developers Breach. [Online]. Available: https://developersbreach.com/convolution-neural-network-deep-learning/. [Accessed: Jan. 8, 2025].
27. A. Hallak, D. Di Castro, and S. Mannor, "Contextual Markov Decision Processes," arXiv preprint arXiv:1502.02259, 2015.
28. AutoML.org. Contextual Reinforcement Learning. [Online]. Available: https://www.automl.org/contextual-rl/. Accessed: [Accessed: May 30, 2025].
29. M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel y A. Srinivas, "Reinforcement Learning with Augmented Data," arXiv preprint arXiv:2004.14990, 2020.
30. M. Weltevrede, C. Horsch, M. T. J. Spaan, and W. Böhmer, “Exploration Implies Data Augmentation: Reachability and Generalisation in Contextual MDPs,” arXiv preprint arXiv:2410.03565.
31. Y. Jiang, J. Z. Kolter, and R. Raileanu, “On the Importance of Exploration for Generalization in Reinforcement Learning,” in Proc. 37th Conf. Neural Information Processing Systems (NeurIPS), New Orleans, LA, USA, 2023.
32. S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.
33. M. Pleines, M. Pallasch, F. Zimmer, y M. Preuss, “Generalization, Mayhems and Limits in Recurrent Proximal Policy Optimization,” arXiv preprint arXiv:2205.11104, 2022. [Online]. Available: https://arxiv.org/abs/2205.11104
34. Long Short-Term Memory Networks, MathWorks. [Online]. Available: https://la.mathworks.com/discovery/lstm.html. [Accessed: Jun 10, 2025].
35. U. Muaz, "Domain Randomization: future of robust modeling," Medium, May 12, 2021. [Online]. Available: https://medium.com/data-science/domain-randomization-c7942ed66583

36. S. Rout, V. Dwivedi y B. Srinivasan, “Numerical Approximation in CFD Problems Using Physics Informed Machine Learning,” arXiv, Nov., 2021. arXiv:2111.02987. https://doi.org/10.48550/arXiv.2111.02987.
37. K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, “Leveraging Procedural Generation to Benchmark Reinforcement Learning,” arXiv preprint, arXiv:1912.01588, Dec. 2019.
38. L. Espeholt, H. Soyer, R. Munos et al., “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures,” arXiv preprint, arXiv:1802.01561, Feb. 2018.
39. K. He et al., “Deep Residual Learning for Image Recognition,” arXiv preprint, arXiv:1512.03385, Dec. 2015.
40. "Super Mario Bros./RAM map," Data Crystal, The Cutting Room Floor. [Online]. Available:https://datacrystal.tcrf.net/wiki/Super_Mario_Bros./RAM_map. [Accessed: Jul. 11, 2025].
41. A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in Proc. 33rd Conf. Neural Information Processing Systems (NeurIPS), Vancouver, Canada, 2019. Available: https://arxiv.org/abs/1912.01703
42. K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, “Quantifying generalization in reinforcement learning,” in Proc. 36th Int. Conf. Machine Learning (ICML), Long Beach, CA, USA, 2019. Available: https://arxiv.org/abs/1812.02341
43. R. Trumpp, A. Schäfftlein, M. Theile, and M. Caccamo, “Impoola: The power of average pooling for image‑based deep reinforcement learning,” arXiv:2503.05546, Mar. 7, 2025. Available: https://arxiv.org/abs/2503.05546
44. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, y W. Zaremba, “OpenAI Gym,” arXiv preprint arXiv:1606.01540, 5 de junio de 2016. Available: https://arxiv.org/abs/1606.01540
45. C. Kauten, “Super Mario Bros for OpenAI Gym", GitHub repository, 2018. [Online]. Available: https://github.com/Kautenja/gym-super-mario-bros [Accessed: Jul. 10, 2025].
46. A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, y N. Dormann, “Stable‑Baselines3: Reliable Reinforcement Learning Implementations,” Journal of Machine Learning Research, vol. 22, no. 268, pp. 1–8, 2021. Available: http://jmlr.org/papers/v22/20-1364.html

47. A. McIntyre, M. Kallada, C. G. Miguel, C. Feher de Silva, y M. L. Netto, “neat-python”, GitHub repository, 2017. [Online]. Available: https://github.com/CodeReclaimers/neat-python
48. Uber Research, “PyTorch-NEAT,” GitHub repository, 2018–2019. [Online]. Available: https://github.com/uber-research/PyTorch-NEAT. [Accessed: Jul. 10, 2025].
49. J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World,” in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, Sep. 2017, pp. 23–30. doi: 10.1109/IROS.2017.8202133.
50. C. Park, rainbow-is-all-you-need, GitHub repository, https://github.com/Curt-Park/rainbow-is-all-you-need. [Accessed: July 13, 2025].
51. A. Nichol, V. Pfau, C. Hesse, O. Klimov y J. Schulman, “Gotta Learn Fast: A New Benchmark for Generalization in RL,” arXiv preprint arXiv:1804.03720, Apr. 2018.
52. G. Mailland, Mario_NEAT_AI, GitHub repository. [Online]. Available: https://github.com/GwenMailland/Mario_NEAT_AI. [Accessed: Jul. 10, 2025].
53. E. Korkmaz, A Survey Analyzing Generalization in Deep Reinforcement Learning, arXiv:2401.02349, 2024.
54. N. Zahedi, N. Kuhn, and E. Yee, "MARIO: Reinforcement Learning on Image Observations," Stanford University, CS224R Course Report, 2025.
55. Wang, B. Li, S. Wang, and T. Wang, “Enhanced Proximal Policy Optimization for Complex Game AI: Applying Reinforcement Learning to Super Mario,” Academic Journal of Computing & Information Science, vol. 7, no. 11, pp. 150–154, 2024, doi: 10.25236/AJCIS.2024.071120.
56. S. Doncieux, G. Paolo, A. Laflaquière, and A. Coninx, “Novelty Search makes Evolvability Inevitable,” in Proc. Genetic and Evolutionary Computation Conf. (GECCO ’20), Cancún, México, Jul. 2020, pp. 9. doi: 10.1145/3377930.3389840
57. N. H. Viet, Super Mario Bros PPO PyTorch, GitHub repository, 2021. [Online]. Available: https://github.com/vietnh1009/Super-mario-bros-PPO-pytorch. [Accessed: Nov. 4, 2025].

Anexos

Anexo A: Funciones de activación

En esta memoria de título las funciones de activación usadas son Sigmoid, tanh, ReLU y ELU, las cuales se definen como:

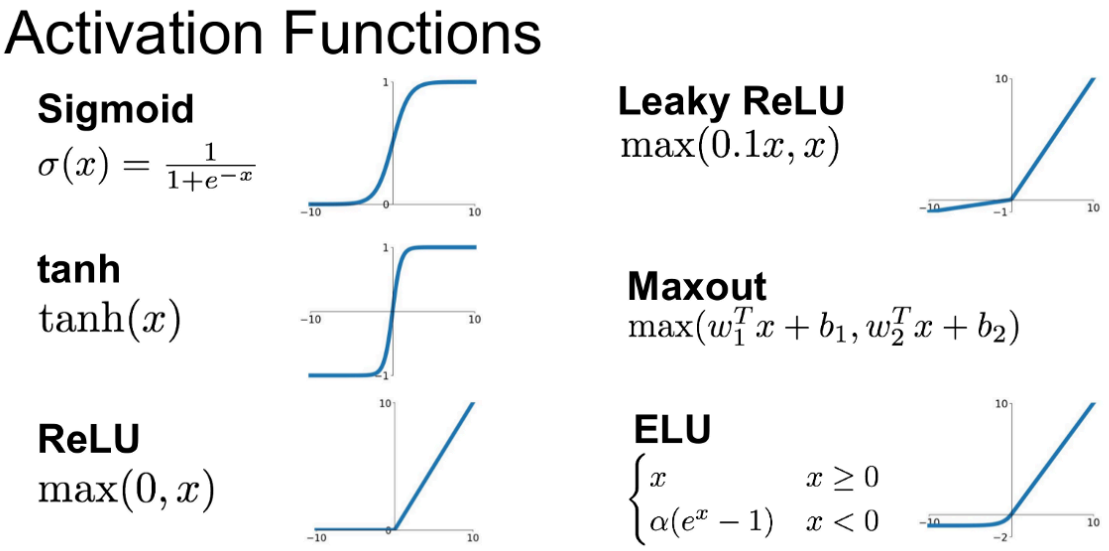

Figura A1. Funciones de activación más conocidas, imagen obtenida de electronics.stackexchange.com.

Anexo B: Bloques residuales, Group normalization y dropout

Los bloques residuales [39] son un método muy importante a la hora de implementar grandes arquitecturas de redes neuronales, ya que permiten no perder información importante tras el paso de cada capa.

El funcionamiento es el siguiente (Figura y):

1. Recibe los datos de entrada y los guarda en una variable x.
2. Los datos pasan por las capas correspondientes (convoluciones, capas lineales, activaciones, etc), obteniendo F(x).
3. Antes de retornar el resultado, a F(x) se le suma x, permitiendo mantener los gradientes y preservar la información útil.

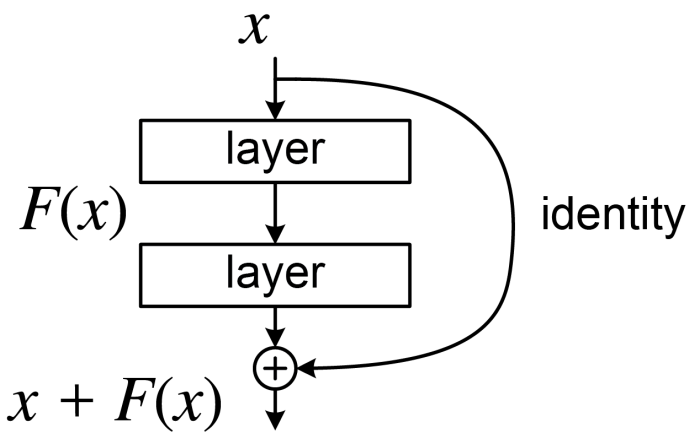

Figura B1. Diagrama de funcionamiento de bloque residuales, obtenido de wikipedia.org.

· **Group normalization:** Método de normalización que funciona dividiendo los canales en grupos y normalizando dentro de cada grupo, a diferencia de Batch Normalization, el cual usa los valores del batch completo para normalizar. Group normalization es más efectivo cuando los batches son muy variables y pequeños, como en este caso, así que se opta por usarlo antes que Batch normalization.

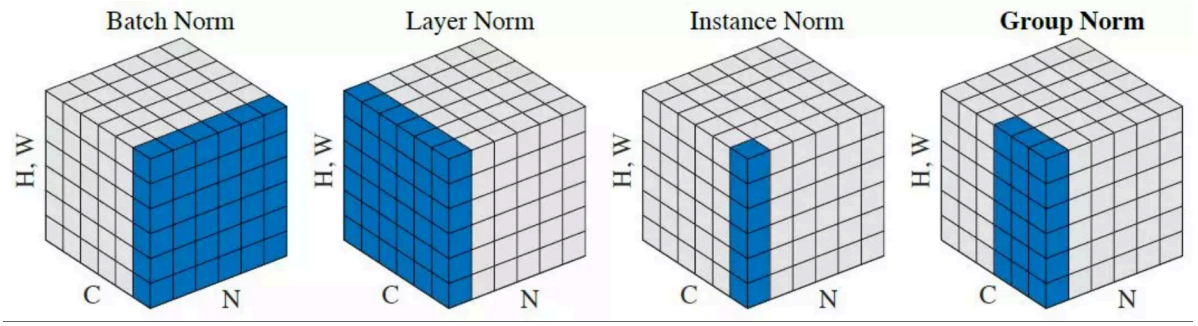

Figura B2. Tipos de normalización existentes y cómo se agrupan.

· **Dropout:** Es un método de regularización estocástico el cual consiste en dar una probabilidad p a cada neurona de la red para desactivarse durante el entrenamiento (Figura z1), esto se usa para que el modelo no dependa de neuronas específicas, generando un modelo más robusto.

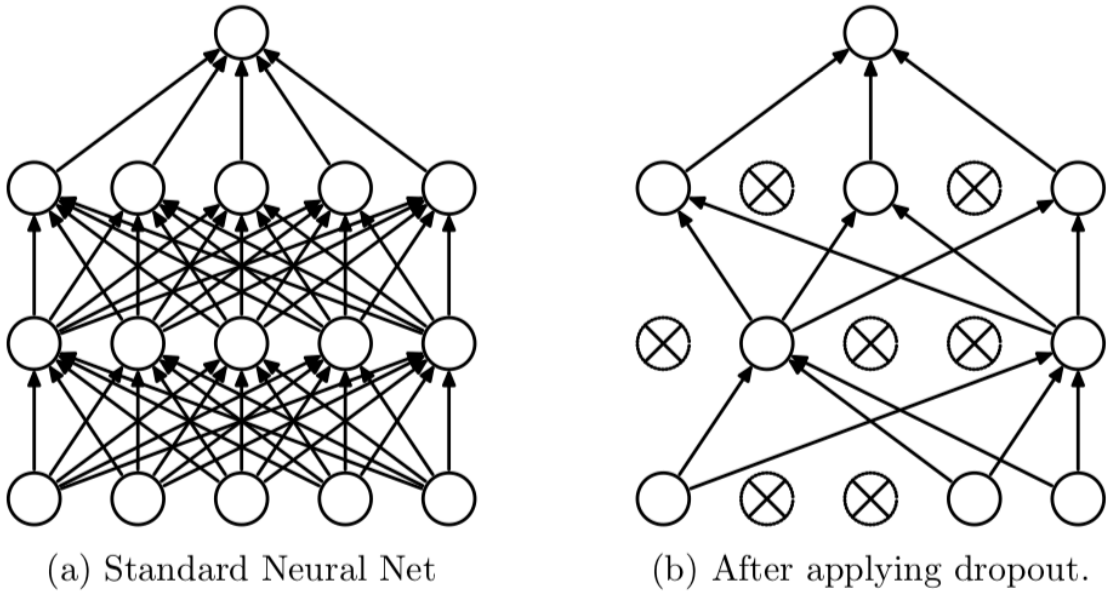

Figura B3. Diagrama de ejemplo de Dropout.

Anexo C: Resultados de evaluación en detalle

Nivel	Fitness promedio	Desviación estándar	Porcentaje avanzado promedio	Porcentaje máximo avanzado
1-1	432.5	±200.9	11.3%	21.4%
1-2	254.4	±170.3	7.7%	29.8%
1-3	366	±139.4	11.9%	28.5%
1-4	320.5	±197.9	10.7%	41.3%
2-1	346.1	±161.9	9.3%	21.6%
2-3	393.8	±269.2	9.4%	41.5%
2-4	348	±167.2	11.6%	49.5%
3-1	458.9	±180.7	11.8%	18.3%
3-2	378	±349.5	9.7%	56.8%
3-3	356.4	±116.7	11.7%	23.3%
3-4	315.1	±155.7	10.6%	38.9%
4-1	389.8	±252.7	9.3%	34.8%
4-2	207.4	±98.3	5.9%	13%
4-3	442.4	±141.5	14.3%	23.5%
5-1	307.3	±228.4	8.4%	24.1%
5-2	272.6	±229.7	7.5%	26.2%
5-3	331.2	±142.1	10.9%	23.9%
5-4	342.5	±154.3	11.4%	49.5%
6-1	417.9	±236.1	11.4%	42.9%
6-2	327.4	±133.8	8.2%	24%
6-3	337.4	±144.8	10.4%	33.7%
6-4	290.3	±149.9	9.9%	21.7%
7-1	374.6	±205.9	10.7%	39.2%
7-3	329	±287.3	8.1%	35.8%
8-1	270.1	±200.4	4.6%	19.3%
8-2	284.7	±156.9	7.4%	34.8%
8-3	351.6	±194.5	8.7%	30.8%

Tabla 4. Resultados de evaluación de NEAT.

Nivel	Fitness promedio	Desviación estándar	Porcentaje avanzado promedio	Porcentaje máximo avanzado
1-1	433	±213.8	11.3%	26.6%
1-2	263.4	±166.7	7.9%	30.9%
1-3	367	±140.2	11.85%	26.7%
1-4	319.3	±171.8	10.7%	29.6%
2-1	354.9	±154.2	9.5%	34.4%
2-3	408.3	±387	9.7%	35.3%
2-4	359.5	±139.9	11.8%	49.5%
3-1	479.1	±161.5	12.3%	22.6%
3-2	395.3	±339.6	10.1%	54.5%
3-3	348.9	±118.6	11.4%	21.2%
3-4	322.1	±161.4	10.8%	56.9%
4-1	440.3	±327.9	10.3%	42.9%
4-2	237.2	±138.4	6.5%	26.7%
4-3	437.9	±152.3	14.2%	23.8%
5-1	335.9	±258.9	9%	29.9%
5-2	265.5	±203.2	7.3%	25.5%
5-3	319.3	±147.8	10.5%	24%
5-4	349.5	±113.3	11.6%	19.8%
6-1	443.6	±243.5	12%	43.8%
6-2	316.2	±139.2	7.9%	24.1%
6-3	351.9	±142	10.7%	28%
6-4	299.3	±138.9	10.1%	21.9%
7-1	409.7	±226.9	11.5%	43.8%
7-3	354.3	±297.2	8.5%	38.9%
8-1	291	±201	4.8%	19.7%
8-2	284.1	±129.4	7.4%	27.8%
8-3	371.6	±221.8	9.2%	33.8%

Tabla 5. Resultados de evaluación de HyperNEAT.

Nivel	Recompensa promedio	Desviación estándar	Porcentaje avanzado promedio	Completado?
1-1	250.4	±0.8	9.3%	No
1-2	721.2	±217.9	28.3%	No
1-3	277.2	±43.4	12.9%	No
1-4	1532.6	±623	63.3%	No
2-1	787	±306.2	25.4%	No
2-3	1504	±539.5	41.9%	No
2-4	754.1	±555	32.5%	No
3-1	362.2	±214	16%	No
3-2	1023	±632	30.9%	No
3-3	244.1	±44.7	11%	No
3-4	796.3	±483	33.9%	No
4-1	1574	±617.3	43.6%	No
4-2	133.7	±34.7	6%	No
4-3	427.3	±91.7	19.4%	No
5-1	681.2	±182.1	22.2%	No
5-2	522.1	±246.6	17.5%	No
5-3	195.3	±81.3	9.8%	No
5-4	971	±401	40.9%	No
6-1	422.9	±247.6	15.4%	No
6-2	686.2	±95.9	21.3%	No
6-3	190.7	±51.2	10%	No
6-4	598.8	±429.4	26.1%	No
7-1	976.3	±309.7	34.5%	No
7-3	1159.1	±420.9	32.6%	No
8-1	405.3	±210.2	7.6%	No
8-2	659.2	±402	19.9%	No
8-3	472.3	±151.9	16%	No

Tabla 6. Resultados de evaluación de DQN.

Nivel	Recompensa promedio	Desviación estándar	Porcentaje avanzado promedio	Completado?
1-1	1470	±454.5	47.9%	No
1-2	2289.5	±821	85%	Si
1-3	351.8	±0.0	15.8%	No
1-4	1890.9	±488.8	77%	No
2-1	2244.7	±311.6	81.2%	No
2-3	3267.9	±505.7	92.4%	Si
2-4	1617.3	±500	69.6%	Si
3-1	299.6	±0.6	10.67%	No
3-2	3225.2	±102.1	100%	Si
3-3	494.1	±65.5	21.6%	No
3-4	1799.1	±671.2	74.1%	No
4-1	3403.8	±377	96.6%	Si
4-2	-742.3	±16.7	2.5%	No
4-3	778.2	±90.5	33.4%	No
5-1	2658.8	±909.5	86.3%	Si
5-2	315	±231.6	13.5%	No
5-3	393.6	±94.4	17.4%	No
5-4	1612.8	±162.5	66.9%	No
6-1	1118.9	±317.8	37.7%	No
6-2	2540.8	±736.5	77.2%	Si
6-3	-643	±4.5	1.3%	No
6-4	1994.2	±540	86.5%	Si
7-1	2331.2	±794.9	84.9%	Si
7-3	987.5	±501.4	28%	No
8-1	169.3	±1.0	3.7%	No
8-2	2052.4	±940.6	59.3%	No
8-3	437.8	±45.5	13.9%	No

Tabla 7. Resultados de evaluación de Rainbow.

Nivel	Recompensa promedio	Desviación estándar	Porcentaje avanzado promedio	Completado?
1-1	959.7	±659.5	37.9%	No
1-2	1823.1	±625.5	78.3%	Si
1-3	256.9	±33.3	12.3%	No
1-4	2034.3	±455.8	92.8%	Si
2-1	2018.3	±469.9	62.6%	No
2-3	3485.7	±134.8	98%	Si
2-4	332.9	±39.6	15.5%	No
3-1	340.3	±10.5	11.8%	No
3-2	3068.2	±204.4	99.4%	Si
3-3	256.0	±1.8	12.2%	No
3-4	2202.4	±40.2	98.5%	Si
4-1	3454.7	±134.7	98.4%	Si
4-2	-638.8	±192.6	3.8%	No
4-3	505.1	±69.9	22.5%	No
5-1	3087.2	±94	99.2%	Si
5-2	408.3	±235.7	15.5%	No
5-3	676.6	±0.6	28.5%	No
5-4	352.8	±0.12	16.3%	No
6-1	447.0	±5.4	16.0%	No
6-2	1787.4	±417.7	54.7%	No
6-3	765.2	±93.6	31.8%	No
6-4	1785.4	±616.6	80.9%	Si
7-1	2626.9	±216	95%	Si
7-3	699.4	±63.7	20.3%	No
8-1	169.8	±0.0	3.7%	No
8-2	2397.7	±496.1	68.6%	No
8-3	525.9	±333.9	16.9%	No

Tabla 8. Resultados de evaluación de Rainbow con randomización de dominio.

Nivel	Recompensa promedio	Desviación estándar	Porcentaje avanzado promedio	Completado?
1-1	2453.2	±831.4	80.43%	Si
1-2	2884.9	±1.0	100%	Si
1-3	589.7	±12.3	25.11%	No
1-4	2208.7	±43.2	99.6%	Si
2-1	3036.8	±270.4	99.4%	Si
2-3	2912.9	±823.2	83%	Si
2-4	912.3	±857.8	42.4%	Si
3-1	714.7	±385.8	23.2%	No
3-2	3286.1	±2.7	100%	Si
3-3	900.3	±209.4	37.4%	No
3-4	1739.2	±469.3	77.7%	Si
4-1	3539.3	±3.0	100%	Si
4-2	644.9	±162.4	20%	No
4-3	1244.7	±2.3	52%	No
5-1	3001.7	±388.6	96%	Si
5-2	842.9	±149.4	27%	No
5-3	169.4	±142.8	8.82%	No
5-4	1759.6	±744.6	80.2%	Si
6-1	2287.1	±997.1	78.8%	Si
6-2	2842.5	±861.7	86.9%	Si
6-3	612.5	±0.44	23.7%	No
6-4	1972.4	±405.9	89%	Si
7-1	2784	±78.3	99.2%	Si
7-3	2312.6	±887.7	66.2%	Si
8-1	2906.5	±991.9	48.5%	No
8-2	2496.9	±831.2	72.5%	Si
8-3	-115.3	±1.41	14.4%	No

Tabla 9. Resultados de evaluación de Rainbow con randomización e IMPALA.

Nivel	Recompensa promedio	Desviación estándar	Porcentaje avanzado promedio	Completado?
1-1	1221.7	±1000.7	39%	Si
1-2	2113.6	±420.3	90%	Si
1-3	595.7	±5.7	25.3%	No
1-4	2219.4	±1.8	100%	Si
2-1	1670.9	±346.4	55.6%	No
2-3	3094.8	±741.7	87.9%	Si
2-4	1970.8	±249.2	90.4%	Si
3-1	795.7	±265.9	25.7%	No
3-2	3287	±4.2	100%	Si
3-3	720.2	±1.6	30.4%	No
3-4	1469	±269.9	60.6%	No
4-1	3538.9	3.9	100%	Si
4-2	1003.6	±241.9	30.3%	No
4-3	1243.9	±1.1	52%	No
5-1	2914	±442.8	93.6%	Si
5-2	618.8	±453.9	20.3%	No
5-3	525.6	±153.3	22.6%	No
5-4	1942.6	±223.5	89.6%	Si
6-1	2466	±804.2	85%	Si
6-2	732.3	±58	24%	No
6-3	834.7	±158.3	31.7%	No
6-4	1876.8	±298.9	81%	Si
7-1	1494.4	±455.9	54%	Si
7-3	807.6	±211.2	23.2%	No
8-1	2780.4	±757.9	46.5%	No
8-2	266.3	±2	9.2%	No
8-3	482.9	±90.3	15%	No

Tabla 10. Resultados de evaluación de Rainbow con randomización, IMPALA y Explore Go.

Nivel	Recompensa promedio	Desviación estándar	Porcentaje avanzado promedio	Completado?
1-1	572	±709.2	29.3%	No
1-2	1808.6	±596.6	63.1%	No
1-3	526.3	±0.6	22.8%	No
1-4	2935.3	±338.2	98%	Si
2-1	2935.3	±295.1	64.9%	Si
2-3	2117.2	±400.1	59.1%	No
2-4	854	±1. 6	36.6%	No
3-1	677.9	±435.3	22.3%	No
3-2	3248.3	±1.9	100%	Si
3-3	316.5	±34.4	14.8%	No
3-4	2066.1	±296.9	94.6%	Si
4-1	3496.8	±1.2	100%	Si
4-2	492.5	±400.8	18.5%	No
4-3	498.9	±11.5	22.4%	No
5-1	3069.9	±67.8	99.5%	Si
5-2	640.9	±149.7	22.3%	No
5-3	525.9	±0.6	22.8%	No
5-4	1814.2	±166.6	75.5%	No
6-1	281.5	±4.12	10.8%	No
6-2	1943.3	±3.8	56.3%	No
6-3	310.8	±10.6	13.2%	No
6-4	1095.6	±1.7	46.3%	No
7-1	2419.9	±310.3	87.6%	Si
7-3	553.4	±92.4	16.5%	No
8-1	994.8	±1318.3	17.4%	No
8-2	1622.3	±339.1	48.5%	No
8-3	1243.9	±313.6	36.8%	No

Tabla 11. Resultados de evaluación de PPO.

Nivel	Recompensa promedio	Desviación estándar	Porcentaje avanzado promedio	Completado?
1-1	851.3	±484.2	28.8%	No
1-2	704.7	±387	30%	No
1-3	331.5	±80.9	15.1%	No
1-4	641	±318.8	27.9%	No
2-1	480.1	±198.8	16.3%	No
2-3	1233	±284.4	34.7%	No
2-4	355.5	±168.2	16.5%	No
3-1	559	±148.2	19.3%	No
3-2	1154.5	±858.8	36.8%	Si
3-3	242.8	±8.7	11.7%	No
3-4	271.4	±54.3	13.1%	No
4-1	1180	±795.6	33.4%	No
4-2	134.1	±134.9	7.3%	No
4-3	295.3	±57.3	14.1%	No
5-1	717.4	±399.3	23.4%	No
5-2	419	±241.0	14.7%	No
5-3	277.7	±89.8	13%	No
5-4	296.8	±61.3	14.1%	No
6-1	348.3	±105.5	12.8%	No
6-2	500	±209.8	16.1%	No
6-3	263.8	±63.3	11.4%	No
6-4	449.2	±262.2	20.2%	No
7-1	524.7	±266.9	19.4%	No
7-3	814.2	±318.5	23.4%	No
8-1	739.3	±479.9	13.3%	No
8-2	236.5	±44.6	8.3%	No
8-3	472.3	±298	17.3%	No

Tabla 12. Resultados de evaluación de PPO con redes recurrentes.

Nivel	Recompensa promedio	Desviación estándar	Porcentaje avanzado promedio	Completado?
1-1	1299.3	±553.3	41.3%	No
1-2	566.4	±347.4	30%	No
1-3	299.4	±56.6	13.9%	No
1-4	2065.1	±54.3	84.9%	No
2-1	1650.7	±131.9	51.4%	No
2-3	2985	±436.3	81.7%	No
2-4	1603.4	±241.4	68.9%	No
3-1	425.5	±218	14.5%	No
3-2	2655.9	±201.8	83.8%	Si
3-3	302.8	±73.4	14%	No
3-4	1866.1	±565.7	85.1%	Si
4-1	3461.5	±220.6	98%	Si
4-2	162.7	±38.7	6.3%	No
4-3	779.6	±64.4	33.5%	No
5-1	1858.6	±267.9	60.8%	No
5-2	850.2	±208.4	27%	No
5-3	413.4	±102.3	18.3%	No
5-4	1678.4	±133.9	69.2%	No
6-1	410.6	±258.7	14.8%	No
6-2	1585.6	±213	46.3%	No
6-3	317	±45.3	13.3%	No
6-4	926.4	±261.3	39.4%	No
7-1	1947.6	±383	67.6%	No
7-3	1022.6	±252.4	28.9%	No
8-1	774.9	±340.6	14.5%	No
8-2	588.8	±13.5	18.8%	No
8-3	394.9	±345.4	17.3%	No

Tabla 13. Resultados de evaluación de PPO con randomización.

Nivel	Recompensa promedio	Desviación estándar	Porcentaje avanzado promedio	Completado?
1-1	804.6	±704.5	31%	Si
1-2	2456.7	±10.5	100%	Si
1-3	366.9	±116.7	18.5%	No
1-4	1811.9	±170.4	96.5%	Si
2-1	2224.9	±465.2	84.2%	Si
2-3	2998.9	±11.2	100%	Si
2-4	816.5	±384.9	39.6%	No
3-1	596.9	±261.4	21.6%	No
3-2	2782.3	±12.3	100%	Si
3-3	343.7	±37.2	17.5%	No
3-4	1663.2	±276.5	87.9%	Si
4-1	2999.9	±9.7	100%	Si
4-2	404.7	±280.2	14.5%	No
4-3	1859.7	±276.4	95.3%	Si
5-1	2387.3	±366.8	90.1%	Si
5-2	622.7	±216.2	22.1%	No
5-3	328.7	±49	16.9%	No
5-4	1387.2	±130.2	65%	No
6-1	488.7	±264.7	19.4%	No
6-2	1812.9	±302	59.9%	No
6-3	461.2	±119.6	20.8%	No
6-4	1847.3	±71.6	98%	Si
7-1	2260.9	±174.1	94.8%	Si
7-3	1010.5	±268.1	32.7%	No
8-1	148.8	±0	3.7%	No
8-2	820.2	±515.3	27.3%	No
8-3	346.4	±283.2	12.5%	No

Tabla 14. Resultados de evaluación de PPO con randomización e IMPALA.

Nivel	Recompensa promedio	Desviación estándar	Porcentaje avanzado promedio	Completado?
1-1	717.2	±227.7	26.7%	No
1-2	1257.5	±947.2	54.2%	Si
1-3	462.4	±137.3	22.7%	No
1-4	1882.8	±8.38	100%	Si
2-1	2344.9	±206.3	87.8%	Si
2-3	2741.6	±674.7	90.4%	Si
2-4	1678.1	±237.4	89.8%	Si
3-1	338.8	±100.1	13.4%	No
3-2	2621.5	±485.9	94.3%	Si
3-3	396.4	±137.5	19.7%	No
3-4	1905.2	±5.2	100%	Si
4-1	2998	±10.6	100%	Si
4-2	233.5	±167.8	13.8%	No
4-3	1866.5	±338.3	94.4%	Si
5-1	2459.2	±334.2	93%	Si
5-2	554.3	±253.2	20.3%	No
5-3	212.5	±76.6	11.6%	No
5-4	1653.3	±505.2	88.4%	Si
6-1	245.3	±2.7	10.6%	No
6-2	2274.4	±299	75.2%	No
6-3	285.3	±1.9	13.6%	No
6-4	1728.4	±219.8	91.4%	Si
7-1	2147	±185.3	88.7%	Si
7-3	880.3	±319.9	28.7%	No
8-1	948	±424.8	18.9%	No
8-2	923.8	±486.4	30.8%	No
8-3	755.2	±333.8	25.5%	No

Tabla 15. Resultados de evaluación de PPO con randomización, IMPALA y Explore Go.