# Aprendizaje por refuerzo aplicado al juego de mesa Sushi Go!:
## Desarrollo y evaluación de agentes inteligentes

Jaime Espel Manzano

17 de octubre de 2024

**Resumen**

El objetivo de este trabajo es desarrollar agentes capaces de jugar Sushi Go! utilizando diversas técnicas de aprendizaje por refuerzo. Dado que el juego involucra decisiones estratégicas complejas, se enseña a los agentes a tomar decisiones ajustándose a las acciones de otros jugadores. Para alcanzar este objetivo, se procede a través de una serie de pasos, que incluyen el desarrollo de un entorno Gymnasium personalizado, el empleo de algoritmos de Reinforcement Learning proporcionados por la biblioteca Stable-Baselines3, y el desarrollo de una interfaz gráfica mediante Pygame, que nos permite jugar contra varios agentes. Finalmente, se evalúa y compara el desempeño de los agentes, con un enfoque en la transferencia de conocimiento entre distintas versiones del juego.

# 1. Introducción

## 1.1. Introducción

El Aprendizaje por Refuerzo, o Reinforcement Learning (RL), representa un enfoque diferente a los principales paradigmas del aprendizaje automático. En el RL, un agente aprende a tomar decisiones a través de la interacción con su entorno, recibiendo recompensas o castigos en función de las acciones realizadas.

El RL, por su naturaleza, está bien adaptado para su aplicación en juegos de mesa, como el *Sushi Go!*

- Esto se debe a que el *Sushi Go!* presenta un entorno claramente definido y reglas fijas.
- Tiene una estructura episódica, es decir, los inicios, finales y momentos de toma de decisiones están bien marcados.
- Y, tenemos la posibilidad de simular partidas, permitiendo que los agentes aprendan sin necesidad de intervención humana constante.

Respecto, a las extensiones del Sushi Go!, aunque existen distintos tipos de extensiones, como el Sushi Go Party!, con nuevos tipos de cartas y reglas adicionales, he optado por utilizar la versión clásica del Sushi Go!

### 1.2. Objetivos

Los objetivos principales de este proyecto han sido:

- desarrollar agentes que puedan jugar al Sushi Go! utilizando técnicas de RL,
- identificar los algoritmos que ofrecen mejor desempeño en términos de competitividad y velocidad de entrenamiento,
- y analizar la efectividad del Transfer Learning en este contexto.

## 2. Marco teórico: Reinforcement Learning

### 2.1. Reinforcement Learning

El RL se basa en el principio del aprendizaje mediante prueba y error, donde los agentes aprenden a tomar decisiones interactuando con su entorno. Para esto, se necesita programar un simulador del entorno, ya que aprender directamente del mundo real, a priori, es inviable.

A través de este proceso de prueba y error, el agente recibe feedback en forma de recompensas por las acciones que realiza, siendo el objetivo maximizar la recompensa acumulada a lo largo del tiempo. Este enfoque se formaliza mediante los Procesos de Decisión de Markov (MDP). La Figura 1 de *Reinforcement Learning: An Introduction* (Sutton, Barto, 2018) [1] ilustra la interacción entre el agente y el entorno en un MDP.

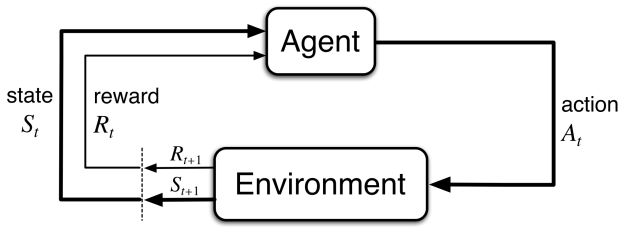



Figura 1: La interacción agente-entorno en un proceso de decisión de Markov

En cada paso de tiempo $t = 0, 1, 2, 3, \ldots$, el agente recibe alguna representación del estado del entorno, $S_t \in \mathcal{S}$, y en base a ello selecciona una acción, $A_t \in \mathcal{A}(s)$. Un paso de tiempo después, en parte como consecuencia de su acción, el agente recibe una recompensa numérica, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, y se encuentra en un nuevo estado, $S_{t+1}$. El MDP y el agente juntos dan lugar a una secuencia o trayectoria que comienza así:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

### 2.2. Algoritmos de RL

En cuanto a los algoritmos de RL, los algoritmos clásicos, como Q-Learning o Sarsa, requieren almacenar una tabla con el valor $Q(s, a)$ para cada combinación de estado y acción.

Guardar la información más relevante para la toma de decisiones, puede suponer una cantidad enorme de estados posibles. Haciendo que el enfoque basado en tablas de estos algoritmos clásicos se vuelva impracticable.

Luego, para superar estas limitaciones, se utilizan métodos de Deep Reinforcement Learning, que combinan técnicas de Aprendizaje por Refuerzo con Redes Neuronales, que permiten al agente generalizar entre estados similares.

Además, se puede hacer uso del Transfer Learning, un enfoque que permite transferir conocimientos de una tarea previamente aprendida a una nueva tarea relacionada, para mejorar la eficiencia y efectividad del entrenamiento de los agentes.

# 3. SushiGo!

Sushi Go! es un juego muy dinámico de cartas en el que los jugadores van eligiendo cartas con distintos tipos de sushi a medida que van pasando por sus manos, y cada carta o combinación de cartas proporciona una puntuación.

## 3.1. Objetivos del SushiGo!

El objetivo es, durante tres rondas, seleccionar la combinación de platos sushi que maximice los puntos acumulados, mientras se intenta minimizar las puntuaciones de los oponentes, de manera que al finalizar las tres rondas, el jugador con el mayor número de puntos acumulados es el ganador de la partida.

## 3.2. Distribución de las Cartas

El juego consta de 108 cartas distribuidas de la siguiente manera [2]:

- 14 cartas de Tempura
- 14 cartas de Sashimi
- 14 cartas de Gyoza
- 12 cartas de Maki (2 rollos)
- 8 cartas de Maki (3 rollos)
- 6 cartas de Maki (1 rollo)
- 10 cartas de Nigiri de salmón
- 5 cartas de Nigiri de calamar
- 5 cartas de Nigiri de tortilla
- 10 cartas de Pudin
- 6 cartas de Wasabi
- 4 cartas de Palillos

## 3.3. Preparación Previa al Juego

La preparación del juego comienza barajando bien las cartas del mazo, y repartiendo cierto número de cartas dependiendo del número de jugadores [2]:

- **Para 2 jugadores:** Cada uno recibe 10 cartas.
- **Para 3 jugadores:** Cada uno recibe 9 cartas.
- **Para 4 jugadores:** Cada uno recibe 8 cartas.
- **Para 5 jugadores:** Cada uno recibe 7 cartas.

Esta distribución de cartas se aplica al inicio de cada nueva ronda.

Cada jugador deberá mantener estas cartas en su mano, de tal forma que los oponentes sean incapaces de verlas. El resto de las cartas deberán ser colocadas boca abajo, formando una pila, en el centro de la mesa, para ser jugadas en rondas posteriores.

## 3.4. Funcionamiento de una Ronda

La partida se juega en tres rondas. En cada ronda, los jugadores comienzan con una mano de cartas cuyo número varía según la cantidad de participantes, tal como se ha detallado en la Subsección de Preparación Previa al Juego (3.3).

Cada turno, los jugadores seleccionan una carta de su mano, para jugar, y la colocan boca abajo delante suyo. A continuación, todos los jugadores revelan las cartas elegidas, simultáneamente. Luego, pasan el resto de cartas de la mano al jugador de su izquierda. Este proceso se repite hasta que todas las cartas se han jugado, marcando el fin de la ronda. En ese momento, hay que calcular los puntos de las cartas jugadas por cada jugador.

## 3.5. Puntuación de las Cartas

- **Gyoza**: Cuantas más cartas de Gyoza tenga, más puntos obtendrá el jugador, con un máximo de 15 puntos si tiene cinco cartas Gyoza o más [2]. Siendo $n$ el número de Gyozas recolectadas, la fórmula para calcular la puntuación de las Gyoza se puede expresar como:

$$\text{Puntuación} = \text{mín}\left(\frac{n \cdot (n+1)}{2}, 15\right)$$

  Es decir, la distribución de puntos quedaría de la siguiente forma, según el número de cartas Gyoza:

| Gyoza: | 1 | 2 | 3 | 4 | 5 o más |
|---|---|---|---|---|---|
| Puntos: | **1** | **3** | **6** | **10** | **15** |

Tabla 1: Tabla de puntuaciones para Gyoza


Figura 2: Gyoza

- **Tempura y Sashimi**: Una pareja (2 cartas) de Tempura otorga 5 puntos. Un trío (3 cartas) de Sashimi otorga 10 puntos. Un conjunto incompleto, ya sea una única carta de Tempura o Sashimi, o una pareja de Sashimi, no aporta puntos. Se pueden puntuar varias parejas de Tempura, y varios tríos de Sashimi, en una misma ronda [2].


Figura 3: Tempura y Sashimi

- **Nigiris y Wasabi**: Un Nigiri de tortilla aporta 1 punto, un Nigiri de salmón 2 puntos, y un Nigiri de calamar aporta 3 puntos. Si se juegan sobre una carta de Wasabi, su valor se triplica [2].

  Si un jugador elige jugar una carta Nigiri teniendo en juego una carta de Wasabi libre, este se colocará sobre el Wasabi y pasará a valer el triple de puntos. No se puede optar por guardar el Wasabi para futuras cartas Nigiri. Además, si el Wasabi se juega demasiado tarde y nunca se coloca un Nigiri sobre él, entonces será completamente inútil, ya que este no aporta puntos por sí solo [2].

  Cabe destacar que solamente se puede colocar una carta de Nigiri encima de cada carta Wasabi [2].


Figura 4: Nigiri de Tortilla, Nigiri de Salmón, Nigiri de Calamar y Wasabi

- **Maki rolls**: Cada carta de Maki roll puede tener 1, 2 o 3 rollos, y la puntuación que se otorgue dependerá de cómo se compita con los otros jugadores por acumular la mayor cantidad de rollos. Al final de cada ronda, cada jugador debe sumar el número total de rollos de Maki indicados en la parte superior de las cartas de Maki que ha jugado.

  El jugador que acumule la mayor cantidad de rollos de Maki recibe 6 puntos, mientras que el segundo lugar obtiene 3 puntos. Si hay un empate, en cualquiera de las posiciones, los puntos se dividen equitativamente entre los jugadores empatados, obviando el resto resultante de la división. En el caso de que se dé un empate en la primera posición, el segundo lugar no recibe puntos.


Figura 5: Maki 1, Maki 2 y Maki 3

- **Palillos**: Las cartas de Palillos no otorgan puntos por sí mismas; sin embargo, cuando un jugador tiene una de estas cartas en juego, en la mesa, le permite seleccionar dos cartas en un mismo turno.

  El procedimiento de su uso es el siguiente: al inicio de un turno, el jugador selecciona una carta de su mano de la manera habitual. Si desea seleccionar una segunda carta, deberá decir "Sushi Go!" en voz alta, antes de que los demás jugadores revelen sus cartas seleccionadas. A continuación, escoge una segunda carta de su mano y la coloca boca abajo en la mesa junto a la primera. Una vez todos los jugadores han colocado sus cartas, estas se revelan simultáneamente.

  Además, antes de pasar las cartas restantes al jugador de la izquierda, el jugador que ha utilizado los Palillos, debe devolver la carta de Palillos a su mano, de modo que otros jugadores puedan utilizarla en turnos futuros.

  Cabe destacar que un jugador puede acumular varias cartas de Palillos frente a sí, pero solo le está permitido utilizar una carta de Palillos por turno.


Figura 6: Palillos

## 3.6. Inicio de una nueva ronda

Una vez concluida una ronda, es fundamental seguir el siguiente procedimiento:

I. Registrar las puntuaciones obtenidas por cada jugador durante la ronda recién finalizada.

II. Dejar las cartas jugadas en la ronda anterior en una pila de descartes, exceptuando las cartas de Pudin, que se deben conservar frente a cada jugador hasta el final de la partida.

III. Repartir una nueva mano a cada jugador, de entre las cartas disponibles, siguiendo el número de cartas especificado según la cantidad de jugadores, tal como se especifica en la Subsección 3.3.

## 3.7. Fin de la partida

Tras la puntuación de la tercera ronda, las cartas que queden en la pila sin jugar deberán ser ignoradas. A continuación, se procede al cálculo de puntos correspondiente a las cartas de Pudin.

- **Pudin**: El jugador con más cartas de Pudin recibe 6 puntos y el jugador con menos pierde 6 puntos. En partidas de dos jugadores, no se aplica la penalización al jugador con menos púdines. Si varios jugadores empatan, se reparten los puntos en ambos casos (obviando el resto).

  En el caso poco frecuente de que todos los jugadores tengan la misma cantidad de púdines, ningún jugador obtiene recompensa o penalización alguna.


Figura 7: Pudin

## 3.8. Determinación del Ganador

El ganador es aquel jugador que haya acumulado más puntos después de las tres rondas. En caso de empate, gana el que más Pudines tenga. Si persiste el empate en puntos y Pudines, se considera empate, aunque esta última regla no está especificada en el reglamento.

### 3.9. Resumen de Reglas

| | |
|---|---|
| 3 RONDAS | para 2 jugadores, 10 cartas para cada uno;<br>para 3 jugadores, 9 cartas para cada uno;<br>para 4 jugadores, 8 cartas para cada uno;<br>para 5 jugadores, 7 cartas para cada uno. |
| Escoged 1 carta. Reveladlas a la vez. Pasad cartas a la izquierda. | |

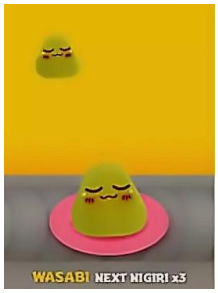

**MAKI**
Más: 6
Segundo: 3
Empates: a partes iguales

**NIGIRI**
De calamar: 3
De salmón: 2
De tortilla: 1



**TEMPURA**
Pareja: 5
Si no: 0

**WASABI**
Triplica el valor del siguiente nigiri





**SASHIMI**
Trío: 10
Si no: 0

**PALILLOS**
Úsalos en un turno posterior para elegir 2 cartas



**GYOZA**
x1 2 3 4 5+
1 3 6 10 15

**PUDIN**
Se puntúan al final de la partida
Más: +6 Menos: -6
Empates: a partes iguales


[2]

# 4. Desarrollo del Proyecto

Este proyecto se ha desarrollado en tres fases:

1. Desarrollo del simulador Sushi Go!, que es compatible con Gymnasium.
2. Integración de los algoritmos de Stable-Baselines3 en el entorno y Entrenamiento.
3. Implementación de Interfaz Gráfica utilizando Pygame

## 4.1. Desarrollo del simulador Sushi Go! compatible con Gymnasium

El primer paso fue el desarrollo del simulador Sushi Go! que es compatible con Gymnasium, lo que ha permitido utilizar la librería Stable-Baselines3 para el entrenamiento de los agentes.

El desarrollo de este simulador incluyó la inicialización del entorno, donde se definieron:

- El *espacio de observación* ($\mathcal{S}$): donde se define la información disponible para el agente en cada paso. En nuestro caso, tal como se detalla en la Subsección 5.1, será un vector de enteros, que contendrá la información de las manos y cartas jugadas para todos los jugadores.
- El *espacio de acciones*($\mathcal{A}(s)$): donde se define el conjunto de acciones posibles que el agente puede tomar en cada turno. En este caso, se define como un espacio discreto, que abarca los números enteros del 0 al 11 (hay 12 tipos de cartas en el Sushi Go!), donde cada número entero representa una acción única, que se corresponde con un tipo de carta específico, conforme a la Tabla 2.

Además, se implementaron los métodos principales del entorno Gymansium, el `reset()`, `step()` y `render()`.

- `reset()`: Se utiliza para iniciar una nueva partida o episodio en el entrenamiento.
- `step(action)`: Permite avanzar en la partida ejecutando las acciones seleccionadas por el jugador.
- `render()`: Proporciona una representación visual, o legible por humanos, del estado actual del entorno.

### 4.1.1. Método `step(action)`

El método `step()` desempeña un papel fundamental en los entornos Gymnasium, ya que gestiona la transición del estado del entorno en respuesta a la acción seleccionada por el agente. Este método actualiza el entorno, calcula las recompensas obtenidas y determina el nuevo estado del juego, permitiendo al agente aprender a través de la retroalimentación obtenida de sus acciones.

En este entorno, el jugador 0 es el jugador que está siendo entrenado. Tal como se puede observar en el diagrama de flujo presentado en la Figura 8, el flujo del método comienza con la verificación del *flag* utilizado, `state_chopsticks`, este nos permite distinguir entre una elección de carta realizada con palillos y sin palillos (cuando esta en `False` indica que es una elección de carta normal, es decir, sin el uso de palillos).

Cuando el flag está desactivado, procedemos con la verificación de la acción seleccionada por el jugador 0. Si la acción es válida, es decir, si el jugador tiene la carta que desea jugar en la mano, se permite que la acción continúe. Si la acción no es válida se aplica una penalización a la recompensa del agente, tal como se detalla en la Subsección 5.4, y además, se selecciona una acción aleatoria de entre las válidas para el jugador 0.

Tras realizar la acción, se comprueba si el jugador 0 puede utilizar la carta especial de palillos (*chopsticks*). Si los palillos están disponibles, el estado del entorno cambia activando la variable `state_chopsticks = True`. Una vez en este estado, se devuelve la observación actualizada y se finaliza el turno, permitiendo que el jugador juegue una carta adicional en el siguiente paso, o *step*.

Cuando el método se vuelve a ejecutar con `state_chopsticks` activado, el jugador tiene la opción de seleccionar otra acción. Si el jugador decide no utilizar los palillos seleccionando una acción no válida, el entorno continúa sin aplicar una penalización.

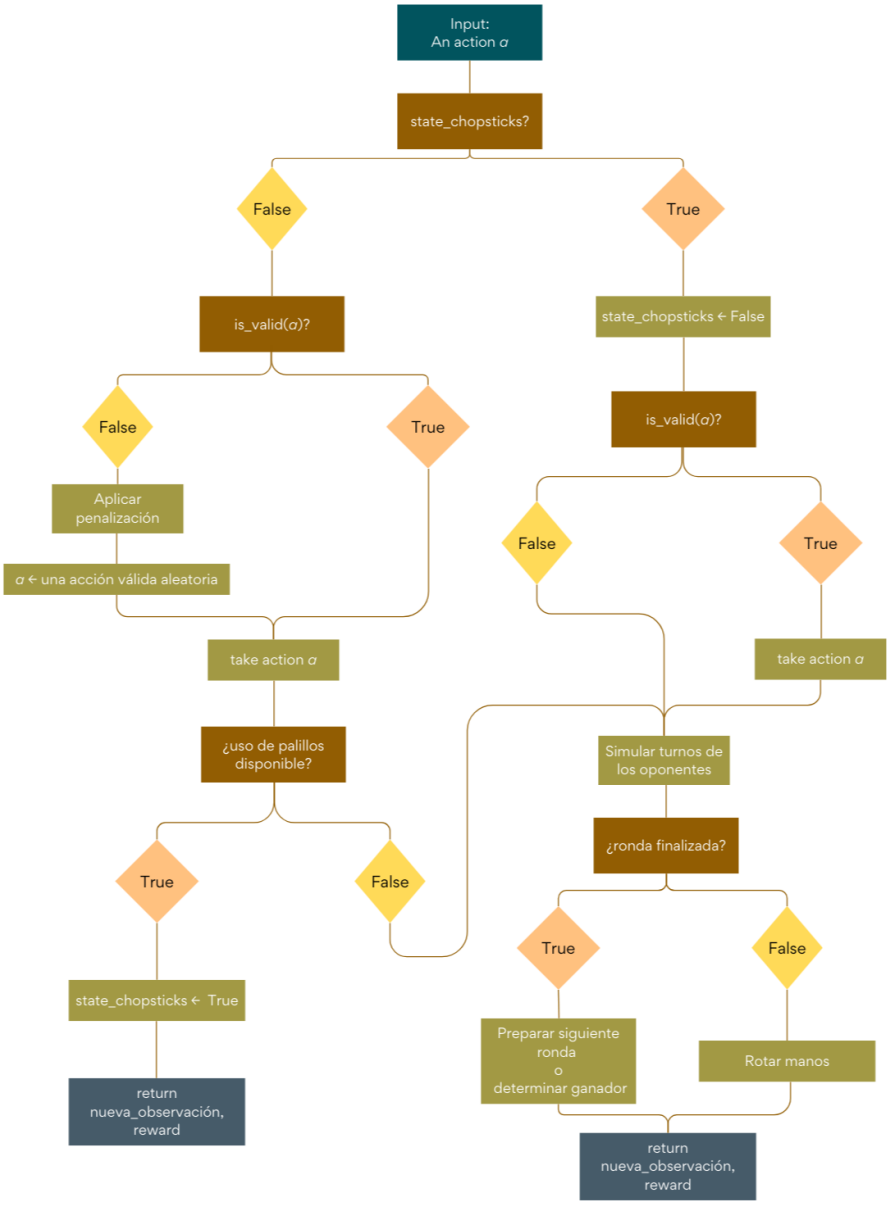
Posteriormente, se simulan los turnos de los oponentes. Cada oponente elige y juega una carta según su estrategia predefinida, lo que afecta el estado general del juego.

El método también verifica si la ronda ha finalizado. Una ronda se considera finalizada cuando todos los jugadores se quedan sin cartas en la mano. Si es el final de la ronda, se calculan las puntuaciones de los jugadores y se reparten nuevas manos para la siguiente ronda, si todavía quedan rondas por jugar. Si todas las rondas del juego ya se han completado, se calcula una puntuación adicional basada en las cartas de pudín jugadas y se determina el ganador del juego.

Si el jugador 0 resulta ser el ganador, recibe una recompensa adicional como incentivo por su desempeño. Si el jugador pierde, se le penaliza en la recompensa final. Esto se describe en detalle en la Subsección 5.4.

Finalmente, si el juego no ha terminado, las manos de cartas de los jugadores se rotan a la izquierda, siguiendo la mecánica de Sushi Go! Se actualiza la observación del entorno para reflejar el nuevo estado del jugador 0 y se prepara el entorno para el próximo turno.

Figura 8: Diagrama de Flujo del método `step(action)`



## 4.2. Integración de los algoritmos de Stable-Baselines3 en el entorno y Entrenamiento

Una vez desarrollado y validado el entorno de Sushi Go! en Gymnasium, el siguiente paso consiste en la implementación de agentes de aprendizaje por refuerzo que puedan interactuar y aprender dentro de este entorno. Posteriormente, se realiza el entrenamiento. Para ello, se ha utilizado Stable-Baselines3, una biblioteca ampliamente utilizada en la comunidad de RL que ofrece implementaciones de algoritmos de aprendizaje por refuerzo.

- De Stable-Baselines3, se emplean los algoritmos:
  - *Proximal Policy Optimization (PPO)*
  - *Deep Q-Network (DQN)*
  - *Advantage Actor-Critic (A2C)*

- Por otro lado, de Stable-Baselines3 Contrib, se emplean los siguientes algoritmos experimentales:
  - *Quantile Regression Deep Q-Network (QR-DQN)*
  - *Trust Region Policy Optimization (TRPO)*

La elección de estos algoritmos se debe a que son algunos de los principales algoritmos de RL que pueden manejar eficientemente entornos con espacios de acción discretos, como es en nuestro caso.

No se ha llevado a cabo un ajuste exhaustivo de los hiperparámetros (*hyperparameter tuning*). En su lugar, se han utilizado los hiperparámetros predeterminados proporcionados por Stable-Baselines3 (SB3) para todos los algoritmos, ya que estos valores por defecto han demostrado ser robustos para la mayoría de los agentes entrenados.

El entrenamiento se realiza mediante el método `learn()`, que permite que el agente interactúe con el entorno a través de múltiples episodios/partidas y pasos de tiempo (tomas de decisiones). Durante este proceso, el agente observa el estado actual del entorno, donde recibe información relevante para la toma de decisiones. A partir de esta información, el agente selecciona una acción dentro del espacio de acciones disponible, basándose en su política de decisiones.

El método `step()` gestiona tanto la ejecución de la acción como la respuesta del entorno, proporcionando una recompensa. Además de procesar la acción del agente, el método `step()` simula los turnos de los oponentes. Para esto, se utiliza el mismo agente de algunos episodios pasados. De esta manera, los oponentes simulan estrategias que ya han sido aprendidas en iteraciones previas del proceso de aprendizaje. Posteriormente, el agente utiliza la recompensa para ajustar su política, mejorando su capacidad de tomar mejores decisiones en el futuro.

Este ciclo de aprendizaje se repite durante numerosos episodios, permitiendo que el agente refine su comportamiento y optimice sus decisiones a medida que interactúa repetidamente con el entorno.

Los *callbacks* estándar en SB3 permiten el guardado de modelos basados en el número de pasos de tiempo (*timesteps*). Sin embargo, en entornos episódicos como Sushi Go!, resulta más relevante y significativo guardar el modelo al finalizar un episodio completo, lo que asegura que el agente ha finalizado una secuencia coherente de acciones antes de almacenar su progreso.

Además, el *callback* tiene que tener la capacidad de cargar el último modelo guardado para utilizarlo como oponente en entrenamientos futuros. En un entorno como Sushi Go!, donde el agente debe aprender a competir contra otros jugadores, la capacidad de entrenar contra versiones anteriores de sí mismo es esencial para mejorar la calidad del aprendizaje.

Es decir, se necesita un control más granular sobre cómo y cuándo se guardan los modelos durante el entrenamiento, así como la posibilidad de cargar versiones anteriores de estos modelos como oponentes. Para abordar estos necesidades, se desarrolló el *callback* personalizado `LastCheckpointCallback`, que gestiona el almacenamiento periódico de los modelos y permite cargar modelos guardados previamente como oponentes.

## 4.3. Implementación de Interfaz Gráfica utilizando Pygame

Para crear una experiencia interactiva y visualmente atractiva en el entorno del juego Sushi Go!, se ha implementado una interfaz gráfica utilizando Pygame. Esta biblioteca permite manejar eventos y renderizar componentes de manera eficiente, facilitando la interacción entre los jugadores y el entorno del juego.

Además de la visualización estándar, donde se puede ver como juegan los agentes entrenados, se ha implementado un método específico llamado `render_human()`, diseñado para manejar las interacciones durante el turno de un jugador humano. Los jugadores pueden seleccionar cartas directamente haciendo clic en ellas con el ratón, y utilizar un botón "Listo" para confirmar sus decisiones. El botón cambia de color según el estado de selección, proporcionando retroalimentación visual clara.

Para mejorar la experiencia del jugador, se han agregado animaciones visuales, como mensajes de "Your turn!" y "Sushi Go!", que enriquecen la interacción con el juego. También se ofrece soporte para múltiples jugadores humanos, gestionando adecuadamente sus turnos.


Figura 9: Interfaz Gráfica para 2 jugadores

# 5. Experimentación y Resultados

Comenzamos con una descripción detallada del Proceso de Decisión de Markov utilizado para modelar el entorno del juego, lo que incluye la definición de los estados, las acciones y la función de recompensa. Posteriormente, se discutirán los experimentos realizados para entrenar y evaluar el rendimiento de los distintos agentes.

## 5.1. Definición de los estados

El estado proporcionará la información necesaria para que el agente pueda tomar una decisión adecuada. En nuestro caso, cada carta del Sushi Go! se representa mediante un índice único, lo que nos permite codificar tanto las cartas individuales como las combinaciones especiales (Wasabi con Nigiri) de manera compacta y eficiente. La Tabla 2 muestra la correspondencia entre las cartas, o combinaciones, y sus índices.

| **Num** | **Carta** |
|---|---|
| 0 | PUDIN |
| 1 | MAKI_1 |
| 2 | MAKI_2 |
| 3 | MAKI_3 |
| 4 | WASABI |
| 5 | NIGIRI_TORTILLA |
| 6 | NIGIRI_SALMON |
| 7 | NIGIRI_CALAMAR |
| 8 | TEMPURA |
| 9 | SASHIMI |
| 10 | GYOZA |
| 11 | PALILLOS |
| 12 | NIGIRI_TORTILLA_WASABI |
| 13 | NIGIRI_SALMON_WASABI |
| 14 | NIGIRI_CALAMAR_WASABI |

Tabla 2: Correspondencia entre índices y cartas, o combinaciones, en Sushi Go!

El estado del juego para los agentes incluye la información de las manos (12 posiciones por jugador) y las cartas jugadas (15 posiciones por jugador) para todos los jugadores, que se representa en un vector de enteros, concatenando esta información jugador por jugador, de tamaño:

$$\text{Tamaño Total} = (\text{diff\_num\_cards} + \text{played\_num\_cards}) \cdot \text{num\_players}, \tag{1}$$

tal como se muestra en la Tabla 3.

| | |
|---|---|
| P0 | Mano [12] |
| | Cartas Jugadas [15] |
| P1 | Mano [12] |
| | Cartas Jugadas [15] |
| P2 | Mano [12] |
| | Cartas Jugadas [15] |
| P3 | Mano [12] |
| | Cartas Jugadas [15] |
| P4 | Mano [12] |
| | Cartas Jugadas [15] |

Tabla 3: Estado para el jugador P0 en el juego Sushi Go! (5 jugadores)

## 5.2. Representación de Manos y Cartas Jugadas

Para la representación de las manos se han barajado varias alternativas. Pero la representación de manos elegida, asigna a cada tipo de carta una posición fija en un vector según la Tabla 2, cada valor indicando cuántas cartas de ese tipo posee el jugador. Por ejemplo, si un jugador tiene dos cartas de *MAKI_3* y una carta de *TEMPURA*, el vector de la mano se vería algo así como:

$$\text{Mano del Jugador} = [0, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0]$$

Para mayor claridad, a partir de ahora se utilizarán los nombres de las cartas como índices, como en la Tabla 4.

| PUDIN | MAKI_1 | MAKI_2 | MAKI_3 | WASAB | NI_TOR | NI_SAL | NI_CAL | TEMP | SASHI | GYOZA | PALI |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

Tabla 4: Representación de la Mano de un Jugador

Los jugadores solo tienen información completa sobre las manos que ya hayan pasado por ellos. Luego, cuando se desconoce la mano de un jugador, las posiciones correspondientes en el vector se llenan con el valor $-1$, reflejando la información incompleta, tal como se puede ver en la Tabla 5. Esta configuración permite que el agente tome decisiones basadas en la información disponible, ajustándose dinámicamente a la incertidumbre.

| PUDIN | MAKI_1 | MAKI_2 | MAKI_3 | WASAB | NI_TOR | NI_SAL | NI_CAL | TEMP | SASHI | GYOZA | PALI |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Tabla 5: Representación de la Mano Desconocida de un Jugador

Las cartas jugadas por cada jugador se representan mediante otro vector de enteros, esta vez de longitud 15. Este vector cubre las 12 cartas individuales más las 3 combinaciones de Nigiri con Wasabi. Cada posición en este vector indica la cantidad de veces que una carta específica o combinación ha sido jugada por un jugador en particular. Por ejemplo, si un jugador ha jugado un NIGIRI_SALMON, dos TEMPURA y una combinación de NIGIRI_TORTILLA con WASABI, el vector de cartas jugadas podría verse como:

| PUDIN | MAKI_1 | MAKI_2 | MAKI_3 | WASAB | NI_TOR | NI_SAL | NI_CAL | TEMP | SASHI | GYOZA | PALI | TOR+W | SAL+W | CAL+W |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 0 |

Tabla 6: Representación de las Cartas Jugadas

#### Ventajas e inconvenientes de las representaciones de los estados

En la representación elegida, la información está organizada de manera explícita y predecible, facilitando al agente la identificación de patrones y correlaciones. Sin embargo, uno de los principales inconvenientes, de esta representación, es la dificultad para extender el juego con nuevos tipos de cartas, ya que esto requiere expandir el tamaño del vector.

#### Análisis del número de estados

Ahora, analizaremos la complejidad aproximada del espacio de estados para la representación escogida de Sushi Go!, para dos jugadores, utilizando combinaciones con repetición. La intención es calcular el número total aproximado de configuraciones posibles para diferentes combinaciones de cartas en la mano y jugadas realizadas.

El coeficiente binomial, comúnmente representado como $\binom{n}{k}$, calcula el número de formas en que se pueden escoger $k$ elementos de un conjunto de $n$ elementos sin considerar el orden. La fórmula matemática para el coeficiente binomial es:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

En el contexto de Sushi Go!, usamos la fórmula del coeficiente binomial para calcular combinaciones con repetición, necesarias para evaluar todas las distribuciones posibles de cartas:

$$\binom{n+r-1}{r} = \frac{(n+r-1)!}{r!(n-1)!}$$

Donde $n$ es el número de tipos de cartas (12 en Sushi Go!) y $r$ es el número de cartas que están siendo consideradas para estar en mano o jugadas. Para un jugador de Sushi Go!, calculamos el número de combinaciones para cada posible distribución de sus cartas en mano y sus cartas jugadas.

Cuando $k$ cartas están en mano:

$$\binom{12+k-1}{k} = \frac{(12+k-1)!}{k! \cdot 11!}$$

Cuando hay $10-k$ cartas jugadas:

$$\binom{12+(10-k)-1}{10-k} = \frac{(21-k)!}{(10-k)! \cdot 11!}$$

Aquí están los resultados para cada configuración de cartas en mano y cartas jugadas, junto con el número total de configuraciones posibles:

- 10 cartas en mano, 0 jugadas: 352.716 configuraciones
- 9 cartas en mano, 1 jugada: 2.015.520 configuraciones
- 8 cartas en mano, 2 jugadas: 5.895.396 configuraciones
- 7 cartas en mano, 3 jugadas: 11.583.936 configuraciones
- 6 cartas en mano, 4 jugadas: 16.893.240 configuraciones
- 5 cartas en mano, 5 jugadas: 19.079.424 configuraciones
- 4 cartas en mano, 6 jugadas: 16.893.240 configuraciones
- 3 cartas en mano, 7 jugadas: 11.583.936 configuraciones
- 2 cartas en mano, 8 jugadas: 5.895.396 configuraciones
- 1 carta en mano, 9 jugadas: 2.015.520 configuraciones
- 0 cartas en mano, 10 jugadas: 352.716 configuraciones

Para encontrar el número total de configuraciones para un jugador, sumamos los productos de combinaciones de cartas en mano y jugadas para cada valor posible de $k$:

$$N = \sum_{k=0}^{10} \left( \binom{12+k-1}{k} \cdot \binom{12+(10-k)-1}{10-k} \right) = 92.561.040$$

Esta suma, $N$, representa el número total de estados diferentes en los que puede encontrarse un solo jugador en cualquier momento del juego.

Cuando se consideran dos jugadores, cada uno con sus propios estados independientes, el número total de estados del juego se convierte en el producto cuadrado de los estados de un solo jugador:

$$\text{Total de estados para dos jugadores} = N^2 = 8.567.546.125.881.600$$

Es importante destacar que este cálculo es una aproximación, ya que no se ha tenido en cuenta la dependencia entre los conjuntos ni se ha controlado el límite de cartas en la distribución del mazo.

El análisis de los estados posibles en Sushi Go! utilizando combinaciones con repetición revela la complejidad y diversidad inherente al juego. Este cálculo detallado muestra cómo incluso juegos aparentemente sencillos pueden generar una cantidad sorprendente de estados posibles, reflejando la profundidad estratégica que el juego puede ofrecer.

Debido al número total de estados posibles, algoritmos como Q-Learning o Sarsa quedan descartados.

## 5.3. Representación de las acciones

El espacio de acciones está definido por los enteros del 0 al 11, que se corresponden directamente con los índices de las cartas disponibles en el juego, siguiendo la estructura de representación de las cartas definida en la Tabla 2. Cada acción representa la selección de una carta específica de la mano del jugador para jugar en su turno.

Este espacio de acciones es estático, lo que significa que el agente siempre puede elegir entre los enteros del 0 al 11, independientemente de las cartas disponibles en su mano en cada momento. Sin embargo, si el agente selecciona una acción inválida —es decir, una carta que no está en su mano actual— recibirá una penalización conforme a la función de recompensa definida en la Subsección 5.4.

## 5.4. Función de recompensa

La función de recompensa en el modelo de Aprendizaje por Refuerzo para *Sushi Go!* sigue una estructura basada en los resultados del juego y las acciones tomadas por el agente. El objetivo es guiar al agente hacia decisiones óptimas, utilizando recompensas y penalizaciones que reflejan el resultado de sus elecciones.

$$\text{Recompensa Final} = \begin{cases} +1, & \text{Si el agente gana la partida} \\ -1, & \text{Si el agente pierde la partida} \\ 0, & \text{En caso de empate} \end{cases} \tag{2}$$

Estos valores proporcionan un objetivo claro al agente: ganar la partida. Sin embargo, durante el transcurso de la partida, el agente también recibe retroalimentación a través de penalizaciones intermedias cuando comete acciones inválidas, lo que lo orienta hacia un comportamiento correcto.

$$\text{Recompensa Intermedia} = \begin{cases} -\frac{2}{(12-\text{num\_players})\times 3}, & \text{Si se selecciona una carta inválida} \\ 0, & \text{En caso contrario} \end{cases} \tag{3}$$

El valor de penalización por acción inválida ha sido ajustado cuidadosamente. Hay que equilibrar entre un castigo lo suficientemente grande para que el agente aprenda a evitar acciones incorrectas, pero lo suficientemente pequeño para que no se concentre únicamente en evitar estas acciones y que aprenda a jugar estratégicamente.

Esta fórmula está diseñada para encontrar un equilibrio en la penalización por acciones incorrectas. La expresión $(12 - \text{num\_players}) \times 3$ representa el número total de decisiones que el agente debe tomar. Esto asegura que la penalización se ajuste correctamente al tamaño de la partida.

Imaginemos que el agente selecciona la acción correspondiente a la carta *TEMPURA* (índice 8), pero esa carta no está presente en su mano actual. Al ejecutar esta acción, el agente recibe inmediatamente la penalización mencionada y se selecciona una acción aleatoria, de entre las válidas, tal como se puede observar en el diagrama de flujo del método `step()` presentado en la Figura 8.

## 5.5. Algoritmos utilizados

Para evaluar el rendimiento de los agentes de RL en Sushi Go!, se ha implementado una variedad de agentes utilizando diferentes algoritmos de RL. Además, para contrastar su desempeño y evitar sesgos, se han implementado agentes de referencia que siguen estrategias simples pero consistentes.

### Algoritmos entrenados

Para entrenar a los agentes en el juego Sushi Go!, se han utilizado distintos de algoritmos de RL, incluyendo:

- A2C (*Actor-Critic Asíncrono*): Un enfoque popular que entrena simultáneamente una política (actor) y un estimador del valor de la política (crítico), lo que permite una actualización más estable de los parámetros del agente.
- DQN (*Deep Q-Network*): Un algoritmo que utiliza una red neuronal para aproximar la función de valor Q y, por lo tanto, tomar decisiones basadas en la maximización de las recompensas esperadas. Este enfoque es eficiente en entornos con acciones discretas, como en Sushi Go!
- PPO (*Proximal Policy Optimization*): Un algoritmo que equilibra la exploración y la explotación al restringir los cambios de política a través de una función de probabilidad, lo que mejora la estabilidad del aprendizaje.
- QR-DQN (*Quantile Regression DQN*): Una variación de DQN que estima la distribución completa de las recompensas futuras mediante la regresión cuantil, proporcionando una mejor estimación del riesgo en la toma de decisiones.
- TRPO (*Trust Region Policy Optimization*): Este algoritmo optimiza la política del agente limitando los cambios grandes en los parámetros, asegurando una mejora más confiable y estable en el rendimiento.

### Agentes implementados

Con el fin de establecer puntos de referencia claros para comparar el rendimiento de los agentes de RL, se han desarrollado agentes de referencia que siguen enfoques más simples, como un agente *Random* y un agente *Rule-based*, o basado en reglas.

Estos algoritmos de referencia permiten verificar que los agentes de RL han desarrollado estrategias efectivas, demostrando que su rendimiento no es resultado del azar, sino de la adquisición de habilidades concretas. El agente Rule-based actúa como un segundo punto de referencia más sofisticado que el agente Random, ya que sigue una lógica estratégica que puede ofrecer una mayor resistencia a los agentes de RL.

Si únicamente se compararan los algoritmos de RL entre sí, no habría forma de determinar si el mejor de ellos es verdaderamente un buen jugador o simplemente el menos deficiente. El dicho "*en el país de los ciegos, el tuerto es rey*" describe de manera adecuada esta situación. Es posible que el "mejor" algoritmo de RL sea, en realidad, un jugador mediocre, similar al tuerto que sobresale solo porque los demás carecen completamente de visión. De ahí la necesidad de los algoritmos de referencia en los experimentos. Si únicamente se comparan los algoritmos de RL entre sí, no hay forma de determinar si el mejor de ellos es verdaderamente un buen jugador o simplemente el menos deficiente.

## 5.6. Descripción de los experimentos

Los experimentos se dividen en dos categorías principales: *Experimentos de Aprendizaje* y *Experimentos de Transfer Learning*.

Los Experimentos de Aprendizaje se centran en evaluar el rendimiento de los agentes en entornos de dos y cinco jugadores. Además, se analiza la velocidad de aprendizaje de los agentes. Por otro lado, los Experimentos de Transfer Learning investigan la capacidad de los agentes para transferir el conocimiento adquirido en un entorno de juego a otro.

| Tipo de Experimento | Descripción de los Experimentos |
|---|---|
| **Tipo 1: Experimentos de Aprendizaje** | ▪ **Experimento 1.1: El Mejor en P2**<br>Evaluar el rendimiento de los agentes en partidas de uno contra uno para determinar cuál es el agente que muestra mejores habilidades estratégicas.<br>▪ **Experimento 1.2: El Mejor en P5**<br>Ampliar la prueba a partidas con cinco jugadores, analizando la capacidad de los agentes para competir en un entorno más complejo.<br>▪ **Experimento 1.3: El que Más Rápido Aprende**<br>Identificar qué agente alcanza un nivel competitivo en el menor tiempo posible, enfrentando los distintos algoritmos a un agente Rule-based. |
| **Tipo 2: Experimentos de Transfer Learning** | ▪ **Experimento 2.1: Transferencia de Aprendizaje de P2 a P5**<br>Entrenar un modelo, utilizando el mejor algoritmo, en un entorno de dos jugadores (P2) y evaluar cuánto tiempo tarda en adaptarse y superar a un modelo entrenado desde cero en un entorno de cinco jugadores (P5). |

Tabla 7: Descripción de los Experimentos

## 5.7. Experimento 1.1: El Mejor en P2

En el Experimento 1.1: El Mejor en P2, el objetivo es determinar qué algoritmo de RL obtiene más victorias, en partidas de dos jugadores en el entorno de SushiGo!

Para ello, los agentes se entrenan durante 1.2 millones de episodios/partidas, compitiendo contra una versión anterior de sí mismos, de hasta 20 episodios anteriores, comenzando desde una versión inicial Random.

El formato es una liga, en la que cada combinación de pares de algoritmos se enfrenta en 100 partidas, 50 como local y 50 como visitante. Cada algoritmo jugando un total de 600 partidas. En las partidas de ida, ambos agentes juegan con manos generadas aleatoriamente, mientras que en las partidas de vuelta se utilizan las manos que le han tocado al adversario en la partida de ida.

Para poder reutilizar estos agentes entrenados en el entorno de dos jugadores, para la experimentación de *Transfer Learning*, se utiliza una representación de tamaño uniforme del estado del juego, independientemente del número real de participantes. Donde las posiciones correspondientes a los jugadores ausentes se rellenan con el valor -1. Tal como se puede observar en el la Tabla 8.

| | |
|---|---|
| P0 | Mano [12] |
| | Cartas Jugadas [15] |
| P1 | Mano [12] |
| | Cartas Jugadas [15] |
| P2 | -1 [12] |
| | -1 [15] |
| P3 | -1 [12] |
| | -1 [15] |
| P4 | -1 [12] |
| | -1 [15] |

Tabla 8: Estado para el jugador P0 en el juego Sushi Go! (2 jugadores)

Aunque este relleno introduce un pequeño ruido en la representación del estado, su impacto es mínimo, ya que el valor -1 no tiene relevancia semántica en el contexto del juego. Además, el agente puede aprender a ignorar las posiciones vacías durante el proceso de entrenamiento. Adicionalmente, el número total de estados posibles en el juego permanece intacto, ya que el tamaño del espacio de observación sigue siendo el mismo, añadiendo, cuando es necesario, los -1 correspondientes por detrás.

### Resultados

La siguiente tabla muestra los resultados absolutos de los enfrentamientos entre los distintos algoritmos. Para cada par de agentes, se registran las victorias, empates y derrotas. Cada columna corresponde a un algoritmo específico, Para cada celda, los valores *W/D/L* (*Wins/Draws/Losses*) representan cuántas veces el algoritmo de la columna ganó ($W$), empató ($D$) o perdió ($L$) contra el algoritmo de la fila. Por ejemplo, observemos el enfrentamiento entre DQN (columna) y A2C (fila): En la celda correspondiente, vemos $W = 99, D = 0, L = 1$. Esto significa que DQN ganó 99 veces, empató 0 veces y perdió 1 vez contra A2C.

| **DQN** | | | **PPO** | | | **QR-DQN** | | | **TRPO** | | | **Random** | | | **Rules** | | | Player |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | D | L | W | D | L | W | D | L | W | D | L | W | D | L | W | D | L | |
| 99 | 0 | 1 | 89 | 1 | 10 | 97 | 1 | 2 | 99 | 0 | 1 | 31 | 1 | 68 | 95 | 1 | 4 | **A2C** |
| - | - | - | 55 | 2 | 43 | 64 | 1 | 35 | 85 | 1 | 14 | 7 | 0 | 93 | 35 | 3 | 62 | **DQN** |
| - | - | - | - | - | - | 71 | 2 | 27 | 92 | 1 | 7 | 2 | 0 | 98 | 27 | 5 | 68 | **PPO** |
| - | - | - | - | - | - | - | - | - | 80 | 0 | 20 | 0 | 0 | 100 | 36 | 4 | 60 | **QR-DQN** |
| - | - | - | - | - | - | - | - | - | - | - | - | 0 | 0 | 100 | 8 | 2 | 90 | **TRPO** |
| - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 88 | 1 | 11 | **Random** |

Tabla 9: Resultados de la Liga P2

A continuación, se presenta una tabla con varias estadísticas básicas que resumen los resultados totales de cada algoritmo. Se incluyen el total de victorias, empates, derrotas, así como las tasas de victoria (*Win Rate*), derrota (*Loss Rate*) y empate (*Draw Rate*).

| Player | Victorias | Empates | Derrotas | Win Rate (%) | Loss Rate (%) | Draw Rate (%) |
|---|---|---|---|---|---|---|
| **A2C** | 86 | 4 | 510 | 14.33 | 85.00 | 0.67 |
| **DQN** | 346 | 7 | 247 | 57.67 | 41.17 | 1.17 |
| **PPO** | 344 | 11 | 245 | 57.33 | 40.83 | 1.83 |
| **QR-DQN** | 412 | 8 | 180 | 68.67 | 30.00 | 1.33 |
| **TRPO** | **546** | 4 | 50 | **91.00** | 8.33 | 0.67 |
| **Random** | 51 | 2 | 547 | 8.50 | 91.17 | 0.33 |
| **Rules** | 289 | 16 | 295 | 48.17 | 49.17 | 2.67 |

Tabla 10: Resultados P2: Comparación de Rendimiento

El rendimiento general coloca a TRPO como el mejor algoritmo con un *win rate* del 91 %, mientras que A2C tiene el peor rendimiento entre los algoritmos de RL, con tan solo un 14.33 % de victorias.

QR-DQN ocupa el segundo lugar en términos de rendimiento, con un *win rate* del 68.67 %. Aunque no alcanza los niveles de TRPO, se posiciona claramente por encima de los demás algoritmos, como DQN y PPO, que muestran un rendimiento muy similar.

El algoritmo A2C, con un *win rate* del 14.33 %, solo supera al algoritmo Random, que obtuvo el peor rendimiento.

En resumen, este análisis muestra que los algoritmos más avanzados de RL, especialmente TRPO y QR-DQN, son claramente superiores a las estrategias heurísticas o aleatorias, y hay una jerarquía de rendimiento clara entre los diferentes algoritmos de RL.

## 5.8. Experimento 1.2: El Mejor en P5

En el Experimento 1.2: El Mejor en P5, el objetivo es determinar qué algoritmo de RL tiene el mejor desempeño, en término de victorias, en partidas de cinco jugadores en el entorno de SushiGo!.

Para ello, los agentes se entrenan durante 1.2 millones de episodios, compitiendo contra contra 4 modelos idénticos de hasta 20 episodios anteriores, comenzando desde versiones iniciales con una estrategia Random.

### 5.8.1. Liga P5

El formato es una liga, en la que participan los cinco agentes de RL simultáneamente. Jugando un total de 100 partidas, cinco ciclos de 20 partidas. Donde cada ciclo, corresponde a una rotación completa de las cartas iniciales entre los jugadores, asegurando que ningún agente se vea favorecido por una disposición particular de las cartas.

#### Resultados

La Tabla 11 resume los resultados generales de la liga de cinco jugadores, en términos de partidas ganadas (W) y empatadas (D). Asimismo, se muestra el promedio de clasificación de cada agente (*Avg. Rank*), así como el número de veces que cada uno obtuvo la 1ª, 2ª, 3ª, 4ª o 5ª posición. Esta distribución de posiciones ocupadas por cada agente ofrece una visión más matizada de su rendimiento general.

| Player | W | D | Avg. Rank | 1st | 2nd | 3rd | 4th | 5th |
|---|---|---|---|---|---|---|---|---|
| **A2C** | 5 | 0 | 3.91 | 5 | 9 | 17 | 28 | 41 |
| **DQN** | 7 | 0 | 3.48 | 7 | 17 | 20 | 33 | 23 |
| **PPO** | 12 | 0 | 3.22 | 12 | 19 | 26 | 21 | 22 |
| **QR-DQN** | 22 | 0 | 2.53 | 22 | 33 | 24 | 12 | 9 |
| **TRPO** | **54** | 0 | **1.85** | **54** | 22 | 13 | 7 | 4 |

Tabla 11: Resultados de la Liga P5

El agente TRPO es el claro vencedor, obteniendo 54 victorias, lo que representa más de la mitad de las 100 partidas jugadas. Este resultado muestra una clara ventaja sobre el resto de los agentes, siendo más del doble del *win rate* del segundo mejor agente, QR-DQN (con 22 victorias). Por otro lado, el agente A2C es el que más veces ocupa el último lugar, lo que nos indica un rendimiento deficiente en comparación con los demás, algo muy similar a lo que ocurría en la liga P2.

### 5.8.2. RL vs 4 Agentes de Referencia

Los algoritmos de RL, también se han enfrentado a los algoritmos de referencia, Random y Rule-based.

El formato consiste en enfrentamientos de un agente de RL contra 4 agentes de referencia, que puden ser todos Random o todos Rule-based. Cada algoritmo de RL jugando un total de 200 partidas. 100 partidas con cada algoritmo de referencia, distribuidas en cinco ciclos de 20 partidas cada uno. Cada ciclo corresponde a una rotación completa de las cartas iniciales entre los jugadores, tal como se ha hecho en la liga P5, asegurando una evaluación equilibrada de las distintas configuraciones iniciales.

**Resultados**

La siguiente tabla presenta el desempeño de los agentes de RL al enfrentarse a oponentes Random y Rule-based. Se muestra el número de partidas ganadas (W), empatadas (D) y perdidas (L) para cada configuración.

| A2C | | | DQN | | | PPO | | | QR-DQN | | | TRPO | | | Player |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| W | D | L | W | D | L | W | D | L | W | D | L | W | D | L | |
| 38 | 0 | 62 | 44 | 1 | 55 | 73 | 0 | 27 | 66 | 1 | 33 | 78 | 1 | 21 | **Random** |
| 20 | 0 | 80 | 29 | 1 | 70 | 38 | 0 | 62 | 39 | 0 | 61 | 57 | 2 | 41 | **Rules** |

Tabla 12: Resultados P5: RL vs Algoritmos de Referencia

Es importante destacar que el número de victorias disminuye debido a que en partidas con 5 jugadores, en comparación con partidas de 2 jugadores, la estrategia individual tiene menos peso. En partidas de 5 jugadores, para cuando un jugador ha logrado deducir las manos de los demás, solo quedan dos cartas, que suelen ser de menor utilidad porque las mejores opciones ya han sido elegidas en los turnos anteriores. Esto contrasta con las partidas de 2 jugadores, donde es más fácil y rápido identificar qué cartas tiene el oponente, lo que permite tomar decisiones más informadas y ajustar la estrategia con mayor eficacia a lo largo de la partida.

La Tabla 13 incluye una comparación detallada entre los distintos agentes, mostrando su rendimiento en términos de promedio de clasificación y las posiciones que ocuparon (del 1º al 5º lugar) al enfrentarse los agentes Rule-based.

| Player | Avg. Rank | 1st | 2nd | 3rd | 4th | 5th |
|---|---|---|---|---|---|---|
| **A2C** | 3.57 | 20 | 9 | 11 | 14 | 46 |
| **DQN** | 2.77 | 30 | 21 | 15 | 10 | 24 |
| **PPO** | 2.49 | 38 | 21 | 12 | 12 | 17 |
| **QR-DQN** | 2.41 | 39 | 21 | 14 | 12 | 14 |
| **TRPO** | 1.86 | 59 | 15 | 12 | 9 | 5 |
| **Random** | 4.5 | 3 | 4 | 9 | 8 | 76 |

Tabla 13: Resultados P5: Promedio de Clasificación vs Agentes Rule-based

TRPO destaca como el mejor algoritmo con un promedio de 1,86. Le siguen QR-DQN y PPO. A2C y DQN, por lo general, acaban en posiciones más bajas. Aunque el rendimiento del A2C es deficiente en comparación con otros algoritmos de RL, es notablemente superior al agente Random.

El agente Random tiene un promedio de 4.5, el más alto entre todos los agentes. Esto refleja que los agentes Rule-based tienen la capacidad de explotar de manera eficiente la falta de estrategia de un oponente Random. Al superar los agentes RL a los agentes Rule-based, validan la idea de que los agentes RL están aprendiendo estrategias sofisticadas.

## 5.9. Experimento 1.3: El que Más Rápido Aprende

En el Experimento 1.3: El que Más Rápido Aprende, el objetivo es determinar qué algoritmo de RL aprende más rápido.

Para ello, los agentes se entrenan durante 1.2 millones de episodios, evaluados en intervalos de 20,000 episodios.

Cada 20.000 episodios de entrenamiento se juegan 1000 partidas contra agentes Rule-based (un agente para P2, y cuatro para P5).

### El que Más Rápido Aprende para P2

A continuación, en la Figura 10, se presenta un gráfico que muestra la evolución del *win rate* de cada agente a lo largo de los episodios en su entrenamiento, comparadas con el agente Random que tiene un *win rate* constante de 0.1.

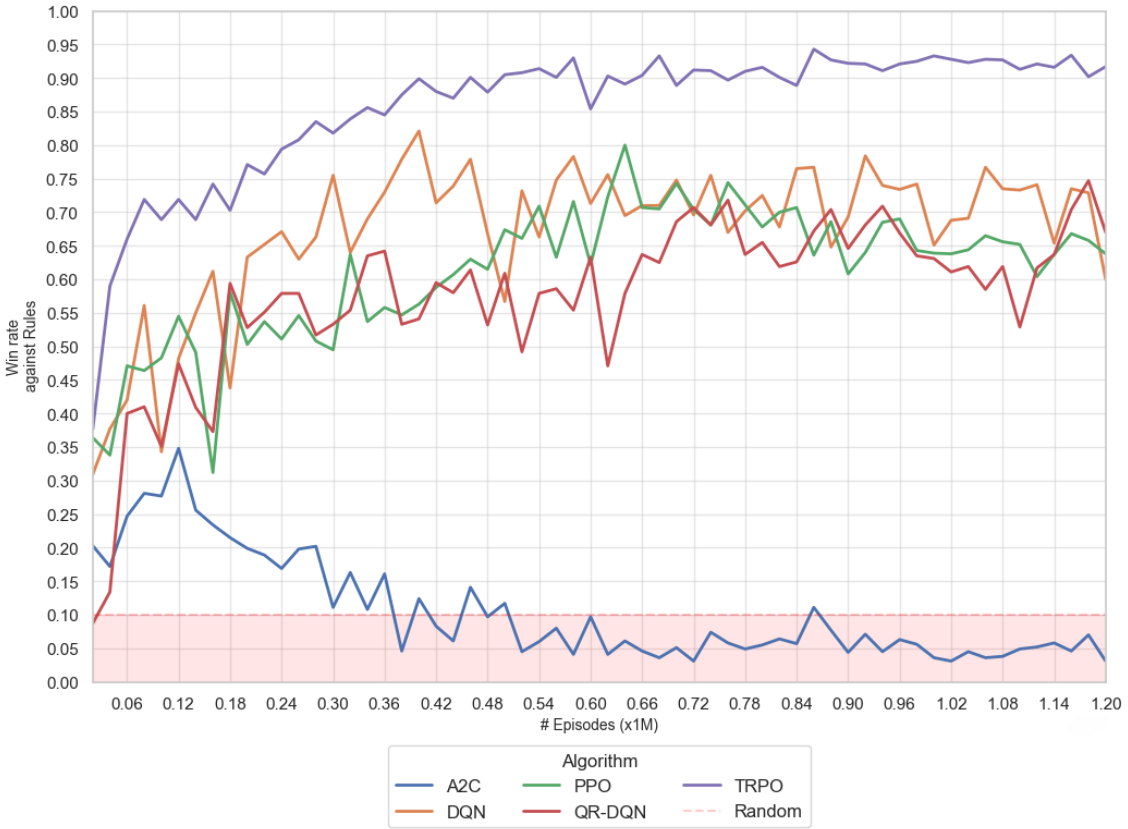



Figura 10: Win Rate de los Agentes RL durante el Entrenamiento contra un oponente Rule-based (2 Jugadores)

Para complementar la visualización del gráfico, se incluye la Tabla 14, que presenta los datos con una granularidad de 100.000 episodios, exceptuando la primera evaluación realizada a los 20.000 episodios. Permitiéndonos observar el progreso inicial y ubicar mejor los hitos clave del entrenamiento en intervalos regulares de 100.000 episodios.

| # episode | 20k | 100k | 200k | 300k | 400k | 500k | 600k | 700k | 800k | 900k | 1M | 1.1M | 1.2M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A2C | 0.2 | 0.28 | 0.2 | 0.11 | 0.12 | 0.12 | 0.1 | 0.05 | 0.06 | 0.04 | 0.04 | 0.05 | 0.03 |
| DQN | 0.31 | 0.34 | 0.63 | 0.76 | 0.82 | 0.57 | 0.71 | 0.75 | 0.72 | 0.69 | 0.65 | 0.73 | 0.6 |
| PPO | 0.36 | 0.48 | 0.5 | 0.5 | 0.56 | 0.67 | 0.62 | 0.74 | 0.68 | 0.61 | 0.64 | 0.65 | 0.64 |
| QR-DQN | 0.09 | 0.35 | 0.53 | 0.53 | 0.54 | 0.61 | 0.63 | 0.69 | 0.66 | 0.65 | 0.63 | 0.53 | 0.67 |
| TRPO | 0.38 | 0.69 | 0.77 | 0.82 | 0.9 | 0.9 | 0.85 | 0.89 | 0.92 | 0.92 | 0.93 | 0.91 | 0.92 |

Tabla 14: Win Rate de los Agentes de RL durante el Entrenamiento contra un oponente Rule-based (2 Jugadores)

Como se puede observar, TRPO es el modelo con mejor rendimiento en todas las etapas del entrenamiento.

Por otro lado, DQN, PPO y QR-DQN muestran unos *win rate* similares. DQN tiene una buena etapa inicial, pero luego empeora. La mejora de PPO es gradual. QR-DQN comienza con un *win rate* bajo, incluso por debajo del agente Random. Sin embargo, a medida que avanza el entrenamiento, el modelo se recupera.

A2C, por su parte, Aunque alcanza un pico temprano, se queda atrás, siendo el que muestra un crecimiento más limitado y lento.

### El que Más Rápido Aprende para P5

A continuación, en la Figura 11, se presenta un gráfico que muestra la evolución del *win rate* de cada agente a lo largo de los episodios en su entrenamiento, comparadas con el agente Random que tiene un *win rate* constante de 0.04.

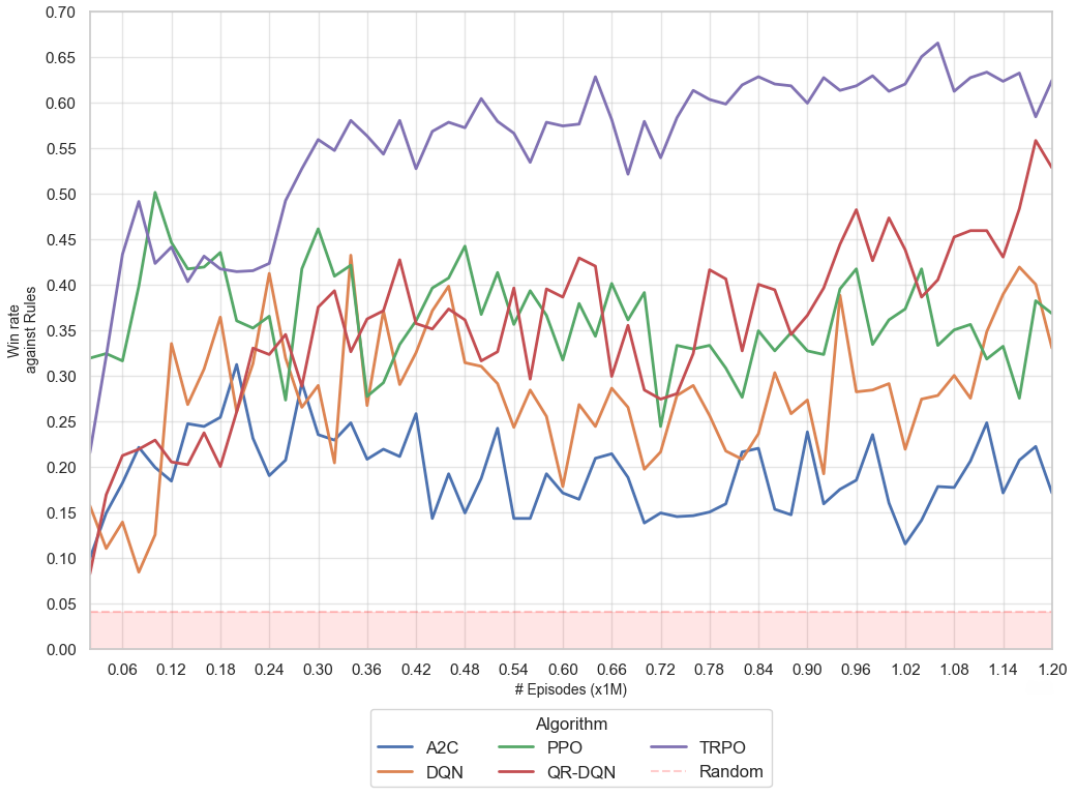



Figura 11: Win Rate de los Agentes de RL durante el Entrenamiento contra 4 oponentes Rule-based (5 Jugadores)

Para complementar la visualización del gráfico, se incluye la Tabla 15, que presenta los datos con una granularidad de 100.000 episodios, exceptuando la primera evaluación realizada a los 20.000 episodios. Permitiéndonos observar el progreso inicial y ubicar mejor los hitos clave del entrenamiento en intervalos regulares de 100.000 episodios.

| # episode | 20k | 100k | 200k | 300k | 400k | 500k | 600k | 700k | 800k | 900k | 1M | 1.1M | 1.2M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A2C | 0.1 | 0.2 | 0.31 | 0.24 | 0.21 | 0.19 | 0.17 | 0.14 | 0.16 | 0.24 | 0.16 | 0.21 | 0.17 |
| DQN | 0.16 | 0.12 | 0.26 | 0.29 | 0.29 | 0.31 | 0.18 | 0.2 | 0.22 | 0.27 | 0.29 | 0.28 | 0.33 |
| PPO | 0.32 | 0.5 | 0.36 | 0.46 | 0.33 | 0.37 | 0.32 | 0.39 | 0.31 | 0.33 | 0.36 | 0.36 | 0.37 |
| QR-DQN | 0.08 | 0.23 | 0.26 | 0.38 | 0.43 | 0.32 | 0.39 | 0.28 | 0.41 | 0.37 | 0.47 | 0.46 | 0.53 |
| TRPO | 0.22 | 0.42 | 0.41 | 0.56 | 0.58 | 0.6 | 0.57 | 0.58 | 0.6 | 0.6 | 0.61 | 0.63 | 0.62 |

Tabla 15: Win Rate de los Agentes de RL durante el Entrenamiento (5 Jugadores)

TRPO demuestra ser, de nuevo, el algoritmo más estable y consistente, alcanzando el *win rate* promedio más alto y mostrando una mejora continua a lo largo de los episodios. Tanto TRPO como PPO destacan por su capacidad de aprendizaje rápido, aunque PPO tiende a estabilizarse tras alcanzar un pico temprano. QR-DQN, aunque con un considerable potencial, muestra altos niveles de fluctuación, obteniendo un aprendizaje tardío. En cambio, DQN presenta un rendimiento mediocre.

Finalmente, A2C es el algoritmo de peor rendimiento, mostrando una ligera mejora, pero sin llegar a ser competitivo. Aún así, todos los algoritmos de RL quedan por encima del agente Random.

## 5.10. Experimento 2.1: Transferencia de Aprendizaje de P2 a P5

En el Experimento 2.1: Transferencia de Aprendizaje de P2 a P5, el objetivo es explorar el impacto del Transfer Learning en agentes de RL.

Para ello, se han entrenado dos agentes:

- Standard-TRPO: se ha entrenado desde cero en un entorno de 5 jugadores durante 1.2 millones de episodios.
- Transfer-TRPO: que es el modelo entrenado en el experimento P2 durante 1.2 millones de episodios. Transferido al entorno de 5 jugadores para continuar su entrenamiento por otros 1.2 millones de episodios.

Los agentes se entrenan durante 1.2 millones de episodios en el entorno P5, evaluados en intervalos de 20,000 episodios, donde se juegan 1000 partidas contra cuatro agentes Rule-based.

### Resultados

A continuación, en la Figura 12, se presenta un gráfico que ilustra la evolución del *win rate* de cada agente a lo largo de los episodios de su entrenamiento en un entorno con cinco jugadores, P5.

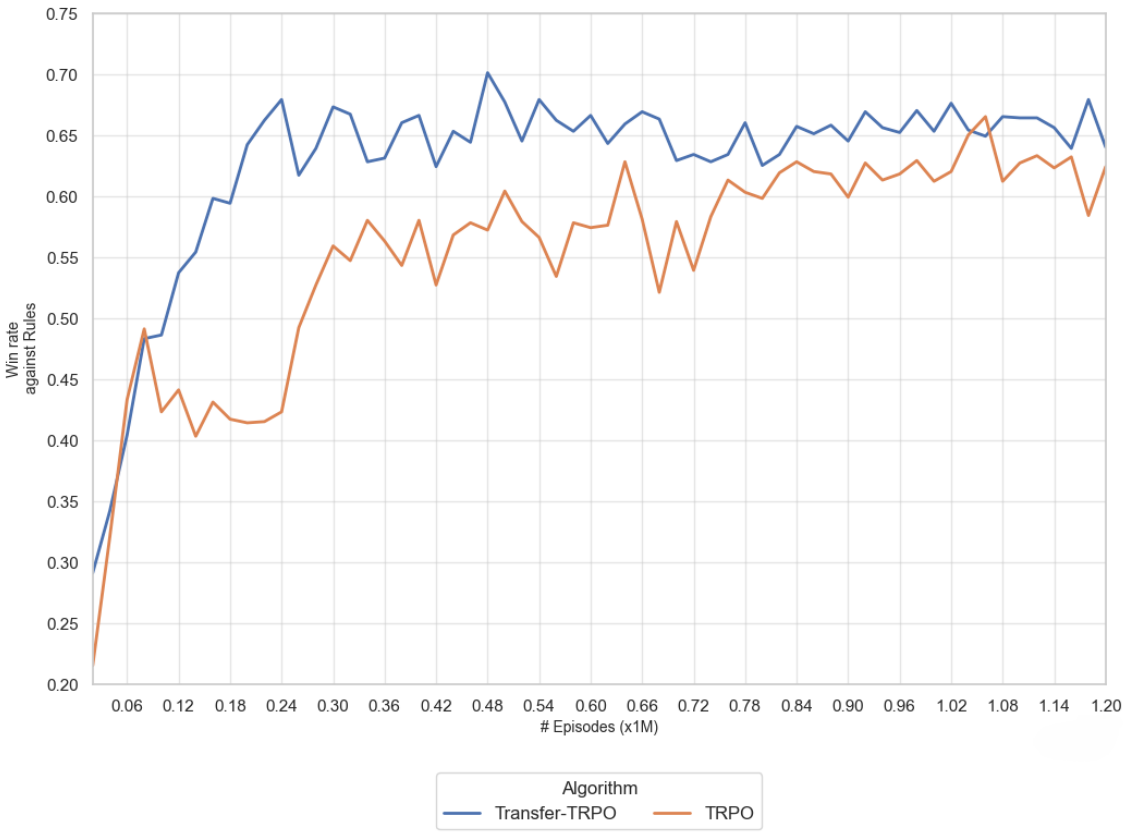



Figura 12: Win Rate del Standard-TRPO y Transfer-TRPO durante el Entrenamiento contra cuatro oponentes Rule-based (*Transfer Learning*)

Para complementar la visualización del gráfico, se incluye la Tabla 16, que presenta los datos con una granularidad de 100.000 episodios, exceptuando la primera evaluación realizada a los 20.000 episodios. Permitiéndonos observar el progreso inicial y ubicar mejor los hitos clave del entrenamiento en intervalos regulares de 100.000 episodios.

| # episode | 20k | 100k | 200k | 300k | 400k | 500k | 600k | 700k | 800k | 900k | 1M | 1.1M | 1.2M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TRPO | 0.22 | 0.42 | 0.41 | 0.56 | 0.58 | 0.6 | 0.57 | 0.58 | 0.6 | 0.6 | 0.61 | 0.63 | 0.62 |
| Transfer-TRPO | 0.29 | 0.49 | 0.64 | 0.67 | 0.67 | 0.68 | 0.67 | 0.63 | 0.62 | 0.64 | 0.65 | 0.66 | 0.64 |

Tabla 16: Win Rate del Standard-TRPO y Transfer-TRPO durante el Entrenamiento contra cuatro oponentes Rule-based (*Transfer Learning*)

Uno de los primeros efectos observables del *Transfer Learning* es que el modelo Transfer-TRPO tiene un mejor punto de partida al inicio del entrenamiento.

Curiosamente, en el episodio 80.000 ambos modelos obtienen casi el mismo *win rate*, pero a partir de ahí, el Transfer-TRPO empieza a tomar ventaja, mostrando un rendimiento superior en el resto del entrenamiento.

Finalmente, uno de los beneficios más importantes del *Transfer Learning* es que el modelo tiende a alcanzar un mayor rendimiento final, es decir, el rendimiento tiene una asintóta más alta.

Esto significa que el modelo entrenado con *Transfer Learning* no solo aprende más rápido y empieza en un punto más alto, sino que también es capaz de llegar a un nivel de desempeño superior en comparación con un modelo entrenado desde cero.

# 6. Conclusiones

En este trabajo, se ha desarrollado un entorno personalizado para el juego Sushi Go!, diseñado para entrenar agentes inteligentes mediante técnicas de aprendizaje por refuerzo. El entorno es compatible con la interfaz de Gymnasium, lo que ha permitido utilizar la librería Stable-Baselines3 para el entrenamiento de los agentes. Se han evaluado diversos algoritmos de aprendizaje por refuerzo, entre ellos A2C, DQN, PPO, QR-DQN y TRPO.

Los experimentos se han dividido en dos categorías principales: *Experimentos de Aprendizaje* y *Experimentos de Transfer Learning*. Donde los Experimentos de Aprendizaje se centran en evaluar el rendimiento y la velocidad de aprendizaje de los agentes. Y los Experimentos de Transfer Learning investigan la capacidad de los agentes para transferir el conocimiento adquirido en un entorno de juego a otro.

Los experimentos realizados han revelado que TRPO es consistentemente el mejor algoritmo, en términos de rendimiento y velocidad de aprendizaje, en los entornos de dos y cinco jugadores. Además, los resultados obtenidos demuestran que el uso del *Transfer Learning* puede acelerar significativamente el proceso de entrenamiento.

A diferencia de los trabajos del estado del arte, donde se hace uso exclusivo del PPO, hay una simplificación excesiva del juego, o se reduce en demasía la información de los estados, para el empleo de técnicas clásicas. A través de este trabajo, he mostrado que para proporcionar al agente estados con suficiente información para tomar decisiones óptimas, la gran cantidad de posibles configuraciones en el estado exige del uso de técnicas avanzadas como *Deep Reinforcement Learning*, capaces de gestionar y generalizar eficazmente en entornos de alta dimensionalidad.

En cuanto a trabajo futuro, una línea prometedora de investigación sería la extensión del entorno con nuevas versiones del juego Sushi Go!, que incluyan tipos adicionales de cartas, lo que permitiría explorar más a fondo el potencial del *Transfer Learning*. Además, se podrían entrenar agentes de distintos niveles que imiten el estilo de juego de jugadores reales utilizando técnicas de *Imitation Learning*.

# Referencias

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[2] Phil Walker-Harding. *Sushi Go! Reglamento*. Gamewright, Newton, MA, 2014. Traducido al español por Marià Pitarque y Marc Figueras, Devir Iberia, S.L.