Universidad Europea VALENCIA

MÁSTER DE FORMACIÓN PERMANENTE EN

INTELIGENCIA ARTIFICIAL

Aplicación de aprendizaje por refuerzo para el desarrollo de estrategias de persecución y evasión en videojuegos

Presentado por:

CESAR AUGUSTO AROCHA CASTILLO

Dirigido por:

VICENTE CASTILLO FAULÍ

CURSO ACADÉMICO 2024-2025

Resumen

El desarrollo de videojuegos, es un campo que requiere equipos multidisciplinarios y recursos considerables, debido al actual auge de herramientas Open source de Inteligencia Artificial (IA), abren camino a desarrolladores independientes y proyectos con presupuestos limitados puedan hacerse realidad. Este Trabajo Final de Máster (TFM) investiga la viabilidad de un flujo de trabajo que integra estas herramientas de IA en el desarrollo de un prototipo de videojuego.

El proyecto se inspira en el juego infantil "Pilla-pilla" y su adaptación profesional, el "Chase Tag". El prototipo, construido en Python, simula un entorno de persecución donde dos agentes autónomos, un cazador (Chaser) y un fugitivo (Evader), interactúan. Su comportamiento se rige por algoritmos de aprendizaje por refuerzo (Reinforcement Learning - RL), que les permiten aprender estrategias de persecución-evasión de manera autónoma.

A lo largo del desarrollo, se utilizaron herramientas de IA para crear fragmentos de código, con el objetivo de ensamblar un prototipo funcional. Permitiéndonos analizar la calidad y adaptabilidad del contenido generado, la complejidad de las funcionalidades que pueden desarrollarse y los desafíos de integrar agentes inteligentes.

La finalidad del TFM es determinar si un prototipo integral, con agentes funcionales y un entorno coherente, puede ser creado con este tipo de herramientas, describiendo los desafíos técnicos y las limitaciones encontradas.

El objetivo general es implementar agentes inteligentes para emular estrategias de persecución y evasión en un videojuego interactivo. Para ello, los objetivos específicos incluyen: analizar herramientas de IA de código abierto, desarrollar el entorno del juego en Python, diseñar un protocolo de entrenamiento, crear un sistema de recompensas para el aprendizaje de los agentes y, finalmente, implementar los agentes de RL.

En conclusión, este estudio busca ofrecer una perspectiva sobre el potencial de la IA en el desarrollo de videojuegos, sirviendo de guía para futuros proyectos. Al demostrar que es posible construir sistemas complejos y funcionales sin grandes inversiones, el TFM aspira a sentar bases para la democratización de la tecnología y la innovación en el campo del diseño de videojuegos y otros proyectos tecnológicos en sus etapas iniciales.

Palabras clave: Inteligencia Artificial, Aprendizaje por Refuerzo, Videojuego, Pygame, Prototipo, Herramientas Open Source, Agentes Autónomos, Chase Tag.

Enlace a github: https://github.com/ces-arocha/TFM

Commit Hash: c5a61e5b0a4ebde0155ae591b45c3d1e4da8a867.

Resumen realizado en Gemini con el prompt: “Para ser utilizado como Resumen de un TFM. Crea el resumen de 500 palabras, del siguiente TFM”.

Abstract

The field of video game development, which traditionally requires multidisciplinary teams and significant resources, is becoming more accessible to independent developers and projects with limited budgets thanks to the rise of open-source Artificial Intelligence (AI) tools. This Master's Thesis investigates the feasibility of a workflow that integrates these AI tools into the development of a video game prototype.

The project is inspired by the children's game "Tag" and its professional adaptation, "Chase Tag." The prototype, built in Python, simulates a pursuit environment where two autonomous agents, a hunter (Chaser) and a fugitive (Evader), interact. Their behavior is governed by **reinforcement learning** algorithms, which allow them to autonomously learn capture and evasion strategies. Throughout the development, AI tools were used to create code fragments, with the goal of assembling a functional prototype. This allowed for an analysis of the quality and adaptability of the generated content, the complexity of the functionalities that can be developed, and the challenges of integrating intelligent agents.

The main purpose of the thesis is to determine if a complete prototype, with functional agents and a coherent environment, can be created with this type of tool, by describing the technical challenges and limitations encountered. The general objective is to implement intelligent agents to emulate pursuit and evasion strategies in an interactive video game. The specific objectives include: analyzing open-source AI tools, developing the game environment in Python, designing a training protocol, creating a reward system for the agents' learning, and finally, implementing the reinforcement learning agents.

In conclusion, this study seeks to offer a perspective on the potential of AI in video game development, serving as a guide for future projects. By demonstrating that it is possible to build complex and functional systems without large investments, the thesis aims to lay the groundwork for the democratization of technology and innovation in the field of video game design and other early-stage tech projects.

Keywords: Artificial Intelligence, Reinforcement Learning, Video Game, Pygame, Prototype, Open-Source Tools, Autonomous Agents, Chase Tag.

GitHub Link: https://github.com/ces-arocha/TFM

Commit Hash: c5a61e5b0a4ebde0155ae591b45c3d1e4da8a867.

Abstract created with Gemini using the prompt: "Translate to English: Resumen”

Índice de contenidos

I. Introducción

La interacción con los videojuegos ha sido tradicionalmente asociada con el entretenimiento y la recreación infantil. Sin embargo, múltiples estudios han evidenciado el potencial de los videojuegos para fomentar habilidades cognitivas, sociales y de resolución de problemas, especialmente cuando se emplean en entornos controlados y con fines didácticos. Ejemplos clásicos son los simuladores utilizados en la Fórmula 1 o en la formación de pilotos de aviación.

Otro ejemplo de la aplicación de videojuegos como herramienta de aprendizaje, son los juegos “Buscaminas” y “Solitario”. Estos juegos inicialmente fueron creados con la intención de que nuevos usuarios de computadoras, pudiesen adaptarse a interactuar con el “mouse”. Demostrando de esta manera, la utilidad de los videojuegos en la adquisición de destrezas específicas en escenarios seguros y repetibles.

No obstante, debido a la reciente proliferación de herramientas de Inteligencia Artificial (IA) y la facilidad en cuanto a disposición y acceso, han comenzado a introducirse rápidamente en muchas disciplinas. Estas tecnologías no solo facilitan la generación de activos visuales y sonoros, sino que también permiten la implementación con bajo coste de agentes inteligentes mediante algoritmos de acceso libre de aprendizaje por refuerzo (Reinforcement Learning - RL), democratizando el acceso al desarrollo y prototipado, especialmente para equipos independientes con recursos limitados.

Este Trabajo Final de Máster (TFM) se enmarca en la creación de un prototipo de videojuego basado en la dinámica de la competición “Chase Tag”, que en la literatura científica es conocida como "Pursuit-Evasion").

Esta dinámica consiste en que dos agentes, un Perseguidor (Chaser) y un Evasor (Evader), interactúan dentro de un entorno de dos dimensiones tipo cuadrícula (grid). Ambos agentes se enfrentarán de forma autónoma, y sus comportamientos se regirán por algoritmos de RL.

El alcance de este trabajo se delimita a la implementación de dos agentes inteligentes basados en RL dentro de un entorno de videojuegos, empleando únicamente herramientas y librerías de acceso libre, y a la documentación de un pipeline completo que abarque desde la generación de recursos hasta la evaluación de estrategias de los agentes. Las preguntas de investigación que guían este estudio son:

1. ¿Es posible la creación del prototipo completamente sin conocimientos en programación de videojuegos?
2. ¿En qué medida puede la IA facilitar la creación integral de un videojuego funcional y reproducible en un entorno limitado?

3. ¿Es posible derivar la interacción entre agentes IA (IA vs IA o usuario vs IA), concretamente a algoritmos de machine learning por refuerzo, para el desarrollo de estrategías de persecución-evasión?
4. ¿Cuáles son las limitaciones y oportunidades del uso exclusivo de herramientas libres en el proceso?

A lo largo de este trabajo se responde a estas preguntas que guían el desarrollo realizado.

Delimitación del TFM

El estudio se restringe al desarrollo de un prototipo de videojuego en un entorno bidimensional (2D) con una estructura de cuadrícula (grid), utilizando exclusivamente en lenguaje de programación Python y librerías de acceso libre, tales como Pygame. El alcance se limita a la implementación y entrenamiento de dos agentes autónomos que interactúan en dicho prototipo, bajo políticas de RL, sin incorporar elementos tridimensionales, motores gráficos avanzados ni componentes de audio.

Contribuciones del TFM

- Un prototipo de videojuego reproducible.
- Un esquema de recompensas y políticas.
- Un conjunto de métricas para evaluar el desempeño de los agentes.
- Un pipeline de desarrollo íntegramente basado en herramientas libres.

Los objetivos específicos son: seleccionar Herramientas para recursos gráficos y lógicas de juegos, Elaboración de entorno interactivo para agentes autónomos en Python, diseñar y configurar entrenamientos, crear un sistema de recompensas y por último implementar y evaluar la interacción entre agentes**.**

Finalmente, dentro de los apartados de marco teórico y la metodología de trabajo, se detallarán conceptos fundamentales sobre RL, las herramientas empleadas durante la elaboración del proyecto, así como también, el diagramas de diseño experimentales y las métricas de evaluación utilizadas en el estudio.

I.1. Justificación

La creación de videojuegos tradicionalmente ha estado reservada para equipos multidisciplinarios con altos niveles de especialización y acceso a recursos considerables, lo que limita el acceso a este campo para desarrolladores independientes o proyectos con recursos limitados.

En este contexto, el auge de herramientas de IA de libre acceso representa un cambio de paradigma al democratizar la creación de contenido y la implementación de funcionalidades complejas.

En primer lugar, la generación de contenido, entornos y funcionalidades relacionadas al desarrollo de videojuegos requiere una inversión muy significativa, en cuanto al tiempo de ejecución, recursos creativos y por la variedad de habilidades requeridas. Debido a esto, se dificulta a pequeños equipos la posibilidad de exploración de ideas innovadoras sin la dependencia de grandes inversiones iniciales.

Recientemente, gracias al creciente interés en la aplicación de IA en diferentes ámbitos, se han abierto nuevas posibilidades para la creación automática de activos gráficos, sonoros y de algoritmos. Lo que permite a equipos de desarrollo poder optimizar sus procesos, reducir costos, complementar experiencias de usuario, ayudando de esta manera a transformar más ideas simples en prototipos implementables.

En segundo lugar, esta investigación tiene una dimensión social y educativa, ya que promueve el acceso a nuevas tecnologías y estimula la creatividad en personas autodidactas o con recursos propios que no cuentan con la experiencia necesaria. Al demostrar la posibilidad y límites de la IA, se abren nuevas puertas para la innovación y la enseñanza en áreas relacionadas con el diseño de videojuegos, la inteligencia artificial y el desarrollo de software en general.

Este TFM se justifica por la necesidad de explorar cómo las herramientas de IA pueden integrarse en el flujo de trabajo, ya sea para el desarrollo de un videojuego o para el desarrollo de iniciativas. Esto permitirá demostrar el potencial de la IA accesible para la creación de interacciones complejas sin depender de costosas licencias o software propietario, abriendo el camino a desarrolladores independientes y proyectos con recursos limitados.

I.2. Problemática

Si bien las herramientas de inteligencia artificial cada día se están actualizando y las versiones gratuitas para la generación de contenido son cada vez

más sofisticadas, su aplicación en el contexto del desarrollo de videojuegos plantea muchas interrogantes sobre la calidad de recursos, la posibilidad de realizar funcionalidades complejas o la adaptabilidad del contenido generado a las necesidades específicas de un proyecto.

Por otra parte, la integración de este contenido generado con un agente inteligente también presenta desafíos adicionales. No solo garantizando que el agente aprenda comportamientos útiles, sino también asegurar que dicho comportamiento sea coherente con el entorno virtual al que pertenece, nos da la posibilidad de detectar bucles infinitos en los entrenamientos o aparición de comportamientos no deseados.

Además de esto, la integración de agentes generados automáticamente por IA genera problemas de compatibilidad y dificultades técnicas al momento de entrenar o evaluar, dificultando la creación de un agente con un rendimiento óptimo y una capacidad de interacción sofisticada.

A esto se suman limitaciones prácticas como la necesidad de hardware adecuado para el entrenamiento de los agentes, la gestión óptima de los recursos computacionales y la dependencia de la supervisión humana para ajustar parámetros, corregir errores y validar la calidad del resultado.

Con todo lo mencionado, nos genera incertidumbre respecto a la viabilidad de utilizar únicamente herramientas de IA gratuitas para un prototipado integral de videojuegos, especialmente si se pretende crear agentes funcionales dentro de entornos coherentes de interacción.

I.3. Finalidad

La finalidad principal de este trabajo final de máster es analizar y determinar la viabilidad o posibles limitaciones de utilizar un flujo de trabajo que combine la generación de contenido con IA y el uso de algoritmos de aprendizaje por refuerzo para la creación de agentes autónomos en el desarrollo de videojuegos.

Los resultados de esta investigación podrían ofrecer nuevas perspectivas sobre el uso de la inteligencia artificial accesible para la creación integral de videojuegos. Se busca analizar la calidad de las interacciones de los agentes, evaluando los procesos desde la creación de recursos visuales, algoritmos de configuración de movimientos o acciones, hasta su integración dentro de los entrenamientos, describiendo los desafíos encontrados durante su implementación.

En resumen, el propósito es ofrecer conclusiones fundamentadas que sirvan de guía para futuros desarrolladores y académicos interesados en aprovechar la IA accesible para innovar en el campo del desarrollo de videojuegos y, potencialmente, en otros proyectos tecnológicos en fases iniciales.

I.4. Objetivos

I.4.1. Objetivo General

Implementar agentes inteligentes basados en aprendizaje por refuerzo con objetivo de emular estrategías de persecución-evasión (pursuit-evasion o Chase Tag), dentro de un entorno de videojuego interactivo creado totalmente en Python.

I.4.2. Objetivos Específicos

I.4.3. Herramientas para recursos gráficos y lógicas de juegos: Investigar y seleccionar herramientas de inteligencia artificial Open Source para la generación de recursos gráficos y la programación de lógicas de juego, con el fin de identificar las más adecuadas para el proyecto.

I.4.4. Elaboración de entorno en Python: Desarrollar un prototipo de videojuego 2D en Python que sirva como entorno de simulación para agentes de IA, implementando las reglas del juego de persecución-evasión y permitiendo la interacción y recolección de datos de manera controlada.

I.4.5. Diseño y configuración de entrenamientos: Diseñar y protocolizar los entrenamientos para los agentes de IA, estableciendo las fases, métricas y criterios de evaluación que permitan la interacción efectiva entre ellos y la recolección de datos para su optimización.

I.4.6. Sistema de recompensas para optimización de aprendizaje: Diseñar e implementar un sistema de recompensas efectivo, que incentive el comportamiento deseado de los agentes y que sirva como la señal de retroalimentación clave para el aprendizaje por refuerzo.

I.4.7. Implementación de agentes con machine learning por refuerzo: Desarrollo y configuración de agentes inteligentes basados en algoritmos de aprendizaje por refuerzo (Q-Learning) para la realización de un objetivo de persecución-evasión.

A lo largo de este estudio, estos objetivos específicos se implementarán, a través de la resolución de tareas descritas en el apartado de diseño y metodología. Además de esto, discutiremos los resultados obtenidos tras la realización de todas las pruebas planificadas, dando así conclusiones respecto a su ejecución.

II. Marco teórico

II.1. Juegos tradicionales como pilla-pilla y su adaptación a tiempos modernos

El pilla-pilla es considerado un juego de persecución, y en la actualidad existen múltiples variantes según la región. Si bien la mecánica central del juego presenta poca o nula variación en las diferentes culturas donde se practica, es común que su denominación cambie en función del país o incluso de la localidad.

Por ejemplo, en Venezuela se conoce como “La ere”, en España se puede llamar “las atrapadas” o “pilla-pilla”, en México y Centroamérica predomina el nombre “la traes”, mientras que en Argentina, aunque existen algunas variantes en las reglas o dinámicas de juego, se le conoce principalmente como “la mancha”.

A esto se suman otras denominaciones como “la lleva”, “la queda” o “la pinta” en distintas regiones hispanohablantes y más allá. Esta riqueza de nombres y pequeñas adaptaciones demuestra no solo la difusión global del juego, sino también su capacidad de arraigarse en la cultura popular de cada sociedad.

A pesar de las diferencias nominales y de las ligeras adaptaciones en las reglas que pueden encontrarse en cada región, la esencia del pilla-pilla radica en la simplicidad de correr y atrapar a los demás jugadores. Esta característica ha permitido que este juego trascienda durante generaciones y continentes, convirtiéndose en una actividad lúdica presente en la infancia de millones de personas en todo el mundo.

Fig. 1, Imagen pilla-pilla

Reglas básicas del pilla-pilla:

- Participantes: El juego requiere al menos de dos jugadores, aunque puede participar un grupo grande.
- Designación del perseguidor: Al inicio, se elige mediante sorteo o consenso a una persona que será “el que la queda” (el perseguidor o “it” en inglés).

- Objetivo: El perseguidor debe atrapar a cualquiera de los otros jugadores tocándolo físicamente.
- Cambio de rol: Cuando un jugador es tocado por el perseguidor, este pasa a ser el nuevo perseguidor (“la queda”, “la trae”, etc.), y el anterior perseguidor se suma a los que huyen.
- Zona de juego: Generalmente se delimita un espacio físico (patio, parque, salón, etc.) donde se desarrolla el juego. Salir de la zona puede implicar penalizaciones o expulsión temporal.
- Reglas adicionales: En algunas variantes, existen “zonas seguras” donde los jugadores no pueden ser atrapados, o reglas para evitar que la misma persona sea perseguida varias veces seguidas. Otras versiones pueden incluir varios perseguidores o condiciones especiales para cambiar de rol.

La universalidad de este juego, junto con la simplicidad y accesibilidad de sus reglas, han hecho posible que permanezca vigente a lo largo de los siglos hasta la actualidad. Un claro ejemplo de la evolución y modernización del pilla-pilla es la competición internacional conocida como “Chase Tag”.

Esta modalidad toma como base la dinámica original del juego, pero la lleva a un nivel profesional, donde los participantes -expertos en parkour y otras disciplinas acrobáticas- deben demostrar gran habilidad y estrategia para esquivar y atrapar a sus oponentes en un entorno especialmente diseñado.

La combinación de sencillez en el objetivo y la espectacularidad de las habilidades físicas, ha generado una competencia que se ha vuelto cada vez más popular, impulsada por eventos televisados y millones de visualizaciones en plataformas digitales. Así, el pilla-pilla no solo sobrevive, sino que sigue reinventándose y adaptándose a los nuevos tiempos y formas de entretenimiento.

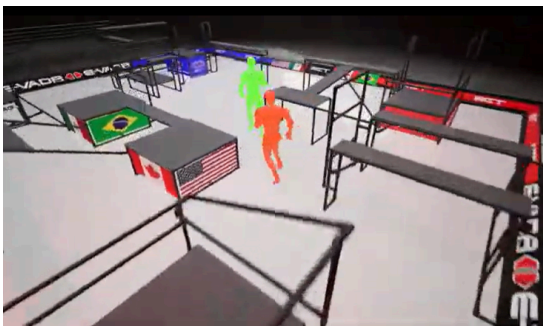

Fig. 2, Imagen World Chase Tag

Reglas básicas del Chase Tag (*World Chase Tag*®, 2025):

- Participantes: Se divide en dos equipos de hasta 5 jugadores, pero solo participan 2 a la vez (uno por equipo).

- Designación del perseguidor: Al inicio, se elige mediante sorteo que equipo arranca como perseguidor (“Chaser”) el otro será evasor o (“Evader”).
- Objetivo: En un periodo de 20 segundos, el “Chaser” debe atrapar al “Evader” tocándolo físicamente con la mano.
- Puntuación: Solo puede ganar puntos el equipo del jugador que está como “Evader”. Si en el tiempo dado el “Evader” no es tocado, su equipo suma un punto y el equipo del “Chaser” debe cambiar de representante.
- Cambio de rol: Si el “Evader” es tocado, este sale del campo y el jugador “Chaser” pasa a ser el nuevo “Evader”. El equipo del jugador eliminado debe incorporar a un nuevo miembro como “Chaser”.
- Zona de juego: El área se denomina “Quad” y está delimitada físicamente, contando con obstáculos variables según la competición o el nivel de dificultad. Salir de la zona señalizada implica la pérdida automática de la persecución.

II.2. Videojuegos como herramienta de aprendizaje y pygame

Los videojuegos, tradicionalmente concebidos como una forma de entretenimiento, han demostrado en los últimos años un enorme potencial como herramienta de aprendizaje. Ejemplos de esto los podemos ver en salones de entrenamiento dentro de diferentes industrias como la Conducción, Transporte y Logística. Como por ejemplo, los simuladores para pilotos de Fórmula 1, donde ayuda a los conductores a ganar incrementan sus niveles de seguridad al realizar una maniobra, permitiendo familiarizarse con procedimientos sin riesgos reales.

Otro ejemplo son las escuelas de aviación para vuelos comerciales, que permiten practicar maniobras, procedimientos de emergencia, navegación y comunicación sin riesgos.

Un ejemplo clásico en la informática, es el del popular juego “Minesweeper” o “Buscaminas” como se conoce en los países de habla hispana. Este juego fue creado con la intención de ser un juego de estrategía, pero muchos profesores en escuelas lo utilizaban para enseñar a nuevos usuarios a interactuar con el mouse de la computadora, permitiendo a los usuarios ganar habilidad y precisión al clickear.

Todo esto es posible gracias al carácter interactivo que aportan los videojuegos y que se ve aumentada por la creatividad de los desarrolladores al crear entornos dinámicos donde los usuarios pueden experimentar, tomar decisiones y recibir retroalimentación inmediata para aprender de los errores, todo esto dentro de un ambiente seguro y controlado.

Diversos estudios han señalado que los videojuegos creados con fines educativos pueden mejorar la retención de conocimientos, el desarrollo de habilidades cognitivas (como la resolución de problemas, la memoria y la atención) e incluso fomentar competencias sociales, como el trabajo en equipo y la comunicación. Además, permiten la personalización del ritmo de aprendizaje y la adaptación de los contenidos a las necesidades individuales de cada usuario, incrementando así la motivación y el compromiso con el proceso formativo.

Por otro lado, los videojuegos facilitan la simulación de escenarios complejos y abstractos que serían difíciles de replicar en un entorno tradicional, como experimentos científicos, situaciones históricas o contextos laborales. De esta manera, los estudiantes pueden aprender de manera activa y significativa, construyendo su propio conocimiento a través de la experiencia directa.

En resumen, el uso de videojuegos en el ámbito educativo no solo transforma la manera en que se transmiten los contenidos, sino que también abre nuevas posibilidades para la enseñanza y el aprendizaje, haciendo el proceso más atractivo, efectivo y accesible para un público diverso.

II.3. Inteligencia Artificial y Machine Learning

La inteligencia artificial (IA) hace referencia a la creación o usos de algoritmos computarizados que sean capaces de emular la mente humana en la realización de actividades. Sin embargo, los psicólogos, biólogos y neurocientíficos, siguen teniendo una noción difusa de la inteligencia, tanto en humanos como en máquinas.

Por esta razón, quienes investigan en el ámbito de la IA suelen emplear preferentemente el término “racionalidad”. La racionalidad como significado, según lo indica la Real Academia Española, es la capacidad de actuar, pensar y juzgar de acuerdo con la razón y la lógica. Que podemos interpretar como la capacidad para poder seleccionar la mejor acción posible con la intención de alcanzar un objetivo específico, considerando criterios de optimización y los recursos disponibles. Aunque la racionalidad no agota el significado de inteligencia, constituye un elemento fundamental.

En este contexto, se utiliza la expresión “sistema de IA” para referirse a cualquier componente, ya sea de software o hardware, que integre IA. Habitualmente, estos sistemas forman parte de plataformas más amplias y no suelen operar de forma completamente autónoma. Así, de acuerdo con uno de los manuales más conocidos de la disciplina, un sistema de IA se caracteriza principalmente por su racionalidad (Dúo Terrón et al., 2023).

Para lograrla, el sistema percibe su entorno mediante sensores, recopila e interpreta datos, razona sobre la información obtenida, decide la mejor acción posible y actúa en consecuencia a través de actuadores, modificando así su

entorno. Los sistemas de IA pueden recurrir tanto a reglas simbólicas como a modelos numéricos basados en aprendizaje, y son capaces de adaptar su comportamiento analizando el impacto de sus acciones previas en el entorno.

El aprendizaje automático (Machine Learning, ML) es una rama de la inteligencia artificial que se centra en el desarrollo de algoritmos y técnicas que permiten a los sistemas aprender patrones y tomar decisiones a partir de datos, sin estar explícitamente programados para realizar tareas específicas (Mitchell, 1997).

Dentro de los distintos tipos de machine learning podemos encontrar:

El aprendizaje supervisado, el cual se utiliza para tareas de clasificación y regresión. Este modelo aprende a partir de una base de datos previamente etiquetados, es decir, cada entrada tiene una respuesta correcta y conocida. Esto con el objetivo de que el algoritmo encuentre una función patrones que permitan predecir la etiqueta de datos no vistos.

El aprendizaje no supervisado, se utiliza para tareas de segmentación de clientes, detección de anomalías y reducción de dimensionalidad, particularmente, donde los datos no están etiquetados. El algoritmo debe encontrar por sí mismo los patrones o la estructura de los datos, como agrupación o asociación entre ellos.

El aprendizaje por refuerzo (Reinforcement Learning - RL) consiste en aprender qué hacer, cómo relacionar situaciones con acciones, para maximizar una señal de recompensa numérica. Al agente de aprendizaje no se le dice qué acciones tomar, sino que debe descubrir cuáles acciones producen la mayor recompensa al probarlas.

En los casos más interesantes y desafiantes, las acciones pueden afectar no solo la recompensa inmediata, sino también la siguiente situación y, a través de ella, todas las recompensas subsiguientes. Estas dos características, la búsqueda por prueba y error y la recompensa retardada, son los dos rasgos distintivos más importantes del RL (Dúo Terrón et al., 2023).

II.3.1. Exploración - Explotación

Uno de los desafíos que surgen en el RL, y no en otros tipos de aprendizaje, es la disyuntiva entre exploración y explotación. Para obtener una gran cantidad de recompensa, un agente de RL debe preferir acciones que ha probado en el pasado y que ha encontrado efectivas para producir recompensa. Pero para descubrir tales acciones, tiene que probar acciones que no ha seleccionado antes.

El agente tiene que explotar lo que ya ha experimentado para obtener recompensa, pero también tiene que explorar para tomar mejores decisiones de acción en el futuro. El dilema es que ni la exploración ni la explotación pueden

buscarse de forma exclusiva ya que dará a lugar al fracaso en la tarea conjunta. El agente debe probar (explorar) una variedad de acciones y favorecer progresivamente (explotar) aquellas que parecen ser mejores.

En una tarea estocástica, cada acción debe probarse muchas veces para obtener una estimación fiable de su recompensa esperada. El dilema exploración-explotación ha sido estudiado intensamente por matemáticos durante muchas décadas, y sin embargo, sigue sin resolverse (Sutton & Barto, 2014).

Por ahora, simplemente señalamos que todo el problema de equilibrar la exploración y la explotación ni siquiera surge en el aprendizaje supervisado y no supervisado, al menos en las formas más puras de estos paradigmas.

II.3.2. Algoritmos de Machine Learning por refuerzo

Más allá del agente y el entorno, se pueden identificar cuatro subelementos principales en un sistema de aprendizaje por refuerzo (Reinforcement Learning - RL): una política, una señal de recompensa, una función de valor y, opcionalmente, un modelo del entorno (Sutton & Barto, 2014).

Una política es la que define las reglas según las cuales los agentes se comportan en un momento dado. A grandes rasgos, una política es un mapeo de los estados percibidos del entorno a las acciones que deben tomarse cuando se está en esos estados. Corresponde a lo que en psicología se llamaría un conjunto de reglas o asociaciones de estímulo-respuesta (Silver et al., 2016). En algunos casos, la política puede ser una función simple o una tabla de consulta, mientras que en otros puede implicar un cálculo extenso, como un proceso de persecución.

La política es el núcleo de un agente de RL en el sentido de que por sí sola es suficiente para determinar el comportamiento. En general, las políticas pueden ser estocásticas, especificando probabilidades para cada acción.

Una señal de recompensa define el objetivo de un problema de RL. En cada paso de tiempo, el entorno envía al agente de RL un único número llamado recompensa. El único objetivo del agente es maximizar la recompensa total que recibe a largo plazo. La señal de recompensa, por lo tanto, define cuáles son los eventos buenos y malos para el agente. En un sistema biológico, podríamos pensar en las recompensas como análogas a las experiencias de placer o dolor. Son las características inmediatas y definitorias del problema al que se enfrenta el agente.

La señal de recompensa es la base principal para alterar la política; si una acción seleccionada por la política es seguida por una recompensa baja, entonces la política puede cambiarse para seleccionar alguna otra acción en esa situación en el futuro.

En general, las señales de recompensa pueden ser funciones estocásticas del estado del entorno y de las acciones tomadas.

Dentro de los algoritmos de RL podemos encontrar:

El **Q-Learning,** este es un algoritmo popular que utiliza la Ecuación de Optimalidad de Bellman para aprender la función de valor de acción óptima (Q-values) sin necesidad de un modelo del entorno. El Q-learning actualiza los valores Q en función de la recompensa inmediata y el valor Q máximo esperado del siguiente estado.

SARSA, este es similar al Q-learning, pero el valor del siguiente estado se basa en la acción que realmente se toma, en lugar de la acción que maximiza el valor.

II.3.2.1. Entorno de ejecución

En el Aprendizaje por Refuerzo (Reinforcement Learning - RL), el entorno es el mundo con el que el agente interactúa. Para que este se considere que esté bien definido, debe proporcionar una base sobre la cual el agente pueda aprender a través de la interacción, buscando maximizar las recompensas recibidas a lo largo del tiempo. Es el "campo de juego" y las "reglas" que el agente debe dominar.

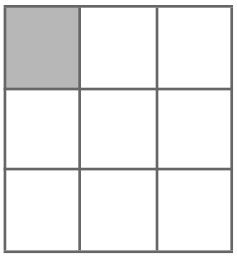

Fig. 3, Cuadrícula Referencia para Entorno

Tabla de Valores de recompensa (Q-Table): representa a la tabla de consulta utilizada en algoritmos de RL para guardar los valores de recompensa obtenidos al cambiar de estado a través de una acción particular. La tabla tiene una fila por cada estado existente y una columna por acción posible.

Estados	**Acciones**				
	Quieto	Arriba	Abajo	Izquierda	Derecha
Estado 1					
Estado 2					
Estado 3					
Estado N					

Fig. 4, Ejemplo de registro de Q Table

Estados (States - S): Los estados representan todas las diferentes situaciones o configuraciones en las que el agente puede encontrarse. Un estado debe proporcionar suficiente información para que el agente tome una decisión informada. Ejemplos de esto lo podemos encontrar en el Ajedrez, que hace referencia a la posición de todas las piezas en el tablero. Otro ejemplo puede ser un Robot móvil, la ubicación actual del robot (coordenadas X, Y), su orientación, la presencia de obstáculos cercanos.

Acciones (Actions - A): Las acciones son los movimientos o decisiones que el agente puede tomar cuando se encuentra en un estado particular. Estas acciones tomadas afectan directamente los cambios de estado. Siguiendo los ejemplos antes mencionados, una acción en el Ajedrez, es mover una pieza de una casilla a otra. Para el caso del Robot móvil las acciones son girar a la izquierda, avanzar y detenerse.

II.3.2.2. Bellman's Equation

La Ecuación de Bellman es un concepto fundamental en el RL. Su principal uso es permitir a los agentes tomar decisiones óptimas en entornos dinámicos e inciertos, descomponiendo un problema complejo de toma de decisiones en pasos más pequeños y manejables. En esencia, la Ecuación de Bellman establece una relación recursiva entre el valor de un estado (o un par estado-acción) en un momento dado y el valor de los estados futuros. Esto permite calcular el valor esperado a largo plazo de estar en un estado particular y seguir una política determinada.

$$Q(S_1, A_1) = reward(S_1) + \gamma \left[max_{A'} Q(S', A') \right]$$

	-1	-1
-1	-1	-1
-1	-1	**10**

Fig. 5, Ejemplo de registro de Recompensas

La versión de la Ecuación de Bellman conocida como Ecuación de Optimalidad de Bellman es fundamental para encontrar la política óptima (Dúo Terrón et al., 2023). Esta ecuación busca la acción que maximiza la recompensa esperada a

largo plazo en cada estado. Al resolver la Ecuación de Optimalidad de Bellman, se puede determinar la mejor acción a tomar en cada situación para maximizar las recompensas acumuladas.

II.3.2.3. Value-based

Mientras que la señal de recompensa indica lo que es bueno en un sentido inmediato, la función de valor obtenida de la ecuación de Bellman indica lo que es bueno a largo plazo. A grandes rasgos, el valor de un estado es la cantidad total de recompensa que un agente puede esperar acumular en el futuro, comenzando desde ese estado. Mientras que las recompensas determinan la utilidad intrínseca e inmediata de los estados del entorno, los valores indican la utilidad a largo plazo de los estados, teniendo en cuenta los estados que probablemente seguirán y las recompensas disponibles en esos estados.

II.3.2.4. Aprendizaje con Temporal Difference Error (TD)

Dentro del RL se aplican técnicas adicionales como la diferencia temporal (TD) para mejorar la capacidad de aprendizaje de los agentes de IA a partir de experiencias parciales. Esto ayuda a que la toma de decisiones sea más óptima en entornos dinámicos y desconocidos, sin la necesidad de contar con un modelo explícito del entorno.

Esta forma de aprendizaje es fundamental para estimar las funciones de valor de estado (V(s)) y las funciones de valor de acción (Q(s,a)). Estas funciones representan la recompensa esperada a largo plazo para cada estado o al tomar una acción específica desde un estado dado. A diferencia de los métodos Monte Carlo, que esperan hasta el final de un episodio para actualizar sus estimaciones, los métodos TD actualizan las predicciones incrementalmente en cada paso, lo que permite un aprendizaje más eficiente en entornos con recompensas diferidas o largos episodios.

$$TD\ Error = Q(S_1, A_1)_{Observado} - Q(S_1, A)_{Esperado}$$

Tanto SARSA y Q-learning utilizan este TD error, para la actualización de valores de recompensa.

II.3.2.5. Esperado vs. Actualización de Muestra

La clave del Q-learning es cómo la Tabla de Valores (Q-table) se actualiza iterativamente a medida que el agente interactúa con el entorno. Cada vez que el

agente realiza una acción, observa una recompensa, y transita a un nuevo estado, utilizando esta información para refinar sus estimaciones de valor Q.

$$Q(S_1,\ A_1)' \leftarrow Q(S_1,\ A_1)\ +\ \alpha\,(TD\ Error)$$

La actualización de la Q-table ocurre de forma iterativa durante el entrenamiento:

Inicialización: la Q-table se inicializa con valores arbitrarios (comúnmente ceros o números aleatorios pequeños).

Exploración y Explotación: en cada paso de tiempo, el agente se encuentra en un estado. Utiliza una política de selección de acciones (comúnmente ϵ-greedy) para elegir una acción:

- Con probabilidad ϵ (epsilon), el agente explora y elige una acción aleatoria para descubrir nuevas posibilidades.

- Con probabilidad 1−ϵ, el agente explota y elige la acción que tiene el valor Q(S,A) más alto en el estado actual, basándose en su conocimiento actual.

Ejecución de la Acción: es el movimiento definido que puede ejecutar el agente.

Observación: es el valor de recompensa que recibe el agente cuando transita los estados.

Actualización del Q-value: usando la fórmula de Bellman anterior, el agente actualiza el valor Q(s,a) en la Q-table.

Nuevo Estado: el estado actual se convierte en el nuevo estado.

Este proceso se repite durante muchas iteraciones a las que llamamos “episodios”, hasta que la Q-table converge, es decir, los valores Q ya no cambian significativamente (Sutton & Barto, 2014).

II.4. Estado del arte

Dentro de este estudio se seleccionaron para consulta artículos y recursos académicos con temáticas de aplicación de Reinforcement learning dentro de

estrategias de persecución-evasión. Se incluye una descripción que se usó como base para su evaluación, inspiración o descarte en el presente trabajo.

II.4.1. Listado de documentación científica (ordenados por enfoque)

II.4.1.1. Visión y aprendizaje distribuido en entornos complejos

"Viper: Visibility-based pursuit-evasion via reinforcement learning." Utiliza un modelo de atención gráfica para coordinar agentes que detectan evasores. (Wang, Y., Cao, Y., Chiun, J., Koley, S., Pham, M., & Sartoretti, G. A., 2024).

II.4.1.2. MADDPG en escenarios dinámicos y parcialmente observables

“Pursuit-Evasion for Car-like Robots with Sensor Constraints” Modela un juego con agentes con restricciones cinemáticas. Utiliza Multi-Agent Deep Deterministic Policy Gradient (MADDPG) transfiriendo a robots reales (Gonultas & Isler, 2025).

II.4.1.3. MAPPO en entornos complejos

"Distributed Pursuit–Evasion Game Decision-Making Based on Multi-Agent Deep Reinforcement Learning” Este combina aprendizaje por currículos automáticos con Multi-Agent Proximal Policy Optimization (MAPPO) (Lin et al., 2025).

II.4.1.4. Implementación real con UAVs

"Pursuit-evasion game with online planning using deep reinforcement learning” Desarrolla un sistema distribuidos para con MADDPG que predice la trayectoria (Chen et al., 2025).

II.4.1.5. Comportamientos emergentes en entornos tabulares

"Emergent behaviors in multiagent pursuit evasion games within a bounded 2D grid world" Explora comportamientos emergentes detectados sobre trayectorias en escenarios de cuadrícula (Xu & Dang, 2025).

II.4.1.6. Método clásico: aprendizaje basado en modelos en cuadrícula pequeñas

“Multi-Agent Model-Based Reinforcement Learning Experiments in the Pursuit Evasion Game.” R-max en cuadrículas comparando enfoques centralizados vs distribuidos (Bouzy & Métivier, 2007).

II.4.1.7. Swarm descentralizado con MADDPG en espacio continuo

"Pursuit-evasion with Decentralized Robotic Swarm in Continuous State Space and Action Space via Deep Reinforcement Learning" Multi-Agent Proximal Policy Optimization (MAPPO) para robots en espacios continuos (Singh et al., 2020).

II.4.1.8. Aplicaciones especializadas: microagentes o entornos físicos únicos

"Reinforcement learning for pursuit and evasion of microswimmers at low Reynolds number" RL en micro agentes para persecución–evasión (Borra et al., 2022).

"Intelligent Pursuit–Evasion Game Based on Deep Reinforcement Learning for Hypersonic Vehicles" entrenamiento reforzado basado en juegos (Gao et al., 2023).

II.4.1.9. Adversario consciente y modelado de oponentes

"An Opponent-Aware Reinforcement Learning Method for Team-to-Team Multi-Vehicle Pursuit via Maximizing Mutual Information Indicator" Modelado de estrategia del oponente mediante DQN (Wang et al., 2022).

"Adversary agent reinforcement learning for pursuit-evasion" Entrenamiento en entornos de visibilidad limitada, mediante agentes adversarios (Huang, 2021).

"Decentralized Multi-Agent Pursuit using Deep Reinforcement Learning" Modelado con recompensas individuales/colectivas con aplicación real reales (Souza et al., 2020).

"Diffusion-Reinforcement Learning Hierarchical Motion Planning in Multi-agent Adversarial Games" Modelo difuso de planificación global con RL en escenarios parcialmente observables (Wu et al., 2024).

II.4.2. Comparación de trabajos con RL relevantes a *pursuit–evasion*

Referencia	Entorno	Algoritmo principal	Aplicación práctica
(Bouzy & Métivier, 2007)	Cuadrícula 2D acotada	Q-learning tabular	Simulación de comportamientos

			emergentes (flanco, emboscada)
(Singh et al., 2020)	Espacio continuo	MADDPG (deep RL multiagente)	Simulación de robots en enjambre, control descentralizado
(Wang et al., 2022)	Urbano simulado, parcialmente observable	DQN + opponent modeling	Persecución en vehículos urbanos (simulación avanzada)
(Souza et al., 2020)	Espacio continuo	Curriculum learning + DDPG/MADDPG	Transferencia de políticas a drones reales
(Wang, Y., Cao, Y., Chiun, J., Koley, S., Pham, M., & Sartoretti, G. A., 2024)	Entorno complejo con visibilidad limitada	GAT (Graph Attention) + MARL	Simulación con percepción visual realista
(Gonultas & Isler, 2025)	Entorno con restricciones de visión y sensores	MADDPG + Curriculum	Transferencia parcial a robots reales
(Lin et al., 2025)	Espacio continuo, multi-UAV	MAPPO + self-play	Estrategias cooperativas de UAVs
(Chen et al., 2025)	Espacio continuo	MADDPG + predicción de trayectoria	Implementación en UAVs físicos (quadcopters)
(Borra et al., 2022)	Medio físico fluido (simulación continua)	RL tabular / deep RL básico	Microswimmers en entornos de dinámica de fluidos
(Gao et al., 2023)	Espacio continuo, dinámica extrema	TD3 (Twin Delayed DDPG)	Aplicación militar (vehículos hipersónicos)

II.4.3. Breve contexto y observaciones comparativas

II.4.3.1. Entorno (cuadrícula vs continuo)

Cuadrícula 2D: trabajos iniciales y académicos (Bouzy & Métivier, 2007); (Xu & Dang, 2025). Se relacionan con algoritmo Q-learning en cuadrícula, similar al utilizado en este TFM.

Espacio continuo: la mayoría de papers encontrados (UAVs, microswimmers, entornos urbanos) usan espacios continuos. Permite mayor capacidad de movimiento, pero es necesario aplicar algoritmos profundos (DQN, MADDPG, MAPPO, TD3).

II.4.3.2. Algoritmos:

Q-learning tabular: aplicación para accionar en cuadrículas como en este TFM.

Opponent-aware & graph-based: necesario para mejorar la coordinación de movimientos y la percepción del ambiente.

II.4.3.3. Aplicaciones prácticas:

Simulación tabular: utilizada para la prueba de hipótesis (ej. comportamiento emergente).

Robótica: tendencia en aumento en los últimos años (MADDPG, MAPPO).

Escenarios especializados: fluidos (micro swimmers) o militares (misiles hipersónicos) muestran aplicaciones en sistemas persecución–evasión.

En resumen, de la revisión podemos confirmar que el problema de persecución–evasión ha sido ampliamente estudiado en el ámbito del *reinforcement learning*, tanto en entornos de cuadrículas como en aplicaciones espacios continuos y sistemas multiagente.

Estos trabajos demuestran que la persecución–evasión es un dominio válido para ser analizado a fondo y evaluar la coordinación, métodos de exploración–explotación y los comportamientos emergentes de las interacciones.

Por tanto, el presente TFM aporta una contribución adicional diferenciada, ya que sitúa el problema en el ámbito de los videojuegos, utilizando un enfoque de Q-learning tabular dentro de entornos de cuadrícula, permitiendo generar un prototipo didáctico, reproducible e interpretable.

Si bien esta aplicación no alcanzó a implementar modelos basados en *Deep Reinforcement Learning*, esta aproximación ofrece valor académico al mostrar de forma clara y experimental cómo emergen estrategias de persecución y evasión, sentando una base pedagógica y un desarrollo con validación preliminar que puede ser adaptada en futuros trabajos hacia escenarios más realistas y algoritmos de Deep Learning.

III. Metodología

En este capítulo se expondrán las metodologías escogidas y utilizadas para la realización, implementación y cumplimiento de los objetivos planteados para el presente proyecto.

La presente investigación propone la creación de algoritmos orientados a la ayuda en el desarrollo y evaluación para un entorno de videojuego que simula la dinámica del pilla-pilla, integrando agentes inteligentes mediante técnicas de aprendizaje por refuerzo (Reinforcement Learning - RL).

III.1. Diseño

La investigación se desarrollará bajo un enfoque exploratorio de tipo aplicado, y finalizará con la comprobación de si la aplicación consciente de herramientas de IA son funcionales o no. Se optará por un diseño cuasiexperimental, ya que se manipulan variables independientes (implementación y configuración de los agentes, tipos de recompensas, etc.) para observar su impacto en variables dependientes como el desempeño, la adaptabilidad y la efectividad de los agentes durante las partidas.

Para implementar los agentes de IA, necesitamos que el diseño se desarrolle por etapas, partiendo de los elementos más básicos y específicos del proyecto. De esta forma, comenzamos identificando y desarrollando los componentes o módulos individuales que serán necesarios, asegurándonos de entender y optimizar cada parte de manera independiente antes de integrarlas en sistemas más complejos.

La metodología de ML utilizada es CRISP-DM (Cross Industry Standard Process for Data Mining) la cual describe como el ciclo de vida para proyectos de datos no es un método lineal, sino un modelo cíclico y flexible. Esto permite avanzar por bloques y regresar a fases anteriores, si la información no está en condiciones o no es suficiente para avanzar (Kotsiantis et al., 2006).

Este proceso está compuesto por seis fases principales.

1. **Comprensión del negocio (Business Understanding):** Es la etapa inicial que tiene como misión, comprender los objetivos del proyecto desde el punto de vista del negocio.

2. **Comprensión de los datos (Data Understanding):** Es la etapa de recolección y familiarización de datos y requerimientos iniciales. Mediante la exploración de sus propiedades, se identifican problemas de calidad para la formulación de hipótesis preliminares.

3. **Preparación de los datos (Data Preparation):** Es la etapa donde se limpian, transforman y seleccionan los requerimientos para ser integrados al proyecto. Se evalúan faltantes, además de corregir errores y construir hipótesis para modelar.

4. **Modelado (Modeling):** En esta fase se elige el algoritmo de aprendizaje más adecuado según el problema a resolver según la clasificación seleccionada. Se entrena utilizando el modelo ajustando sus parámetros internos para minimizar el error y optimizar el rendimiento.

5. **Evaluación y validación (Evaluation):** Una vez entrenado el modelo, se evalúa su desempeño utilizando el conjunto de pruebas y métricas específicas, para verificar su capacidad de generalización. Si el desempeño no es satisfactorio, se pueden ajustar los parámetros del modelo (hiper parámetros), seleccionar nuevas características o incluso probar con otros algoritmos.

6. **Implementación y monitoreo (Deployment):** Finalmente, el modelo aprendido se integra en un entorno de producción, donde debe ser monitoreado y actualizado periódicamente con nuevos datos para mantener su eficacia.

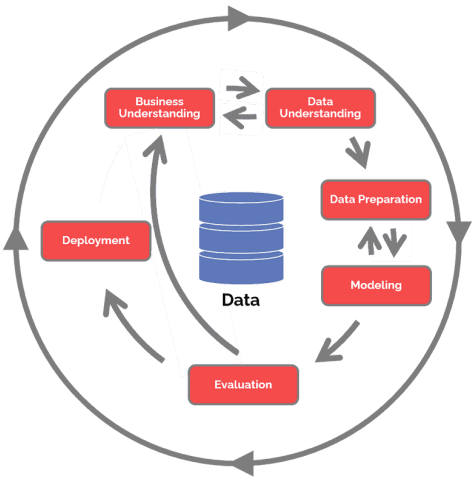

Fig. 6, Esquema CRISP-DM

CRISP-DM es una metodología iterativa que permite construir sistemas capaces de adaptarse y mejorar a medida que se dispone de más datos, siendo fundamental en aplicaciones como la visión por computadora, el procesamiento de lenguaje natural y la predicción de series temporales.

Aunque el proyecto se desarrolló con una metodología CRISP-DM, nos basamos en un esquema "Bottom-Up" para la realización de avances. Partiendo de la construcción del proyecto desde los niveles más bajos, es decir, a partir de los detalles y funcionalidades particulares de cada módulo.

Como por ejemplo la comprensión del problema para diseño de entorno y primeros accionares, definición de estados y recompensas (preparación de los datos), implementación del algoritmo de RL (modelado)

Cada componente se diseña y evalúa por separado, mediante métricas o funcionalidad (evaluación), para ser integrados posteriormente según se coordinen los entregables para formar el sistema completo (despliegue), en nuestro caso, un prototipo de videojuego de persecución-evasión.

Por medio de este proceso de integración gradual, se logra una solución global a partir de la suma de componentes bien definidos, permitiendo identificar y resolver posibles problemas desde las etapas iniciales de desarrollo..

Esta metodología es una estrategia de procesamiento de información utilizada especialmente en la ingeniería y el desarrollo de sistemas, ya que favorece la robustez y flexibilidad en el diseño, permitiendo identificar y resolver posibles problemas desde las etapas iniciales de desarrollo.

III.2. Participantes

Dado que el sistema simulado se centra en la interacción entre agentes virtuales, los "participantes" principales serán los agentes inteligentes codificados para desempeñar los roles de perseguidor y evasor dentro del juego.

Únicamente el autor del presente trabajo ha participado en la realización de evaluaciones de funcionalidad y apreciativas sobre el desenvolvimiento de los agentes en el entorno principal de Juego, de esta manera se pudo validar el desempeño y la interacción humano-agente.

III.3. Instrumentos

III.3.1. Recursos de Hardware

- **Procesador:** 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz (1.69 GHz).
- **RAM:** 16,0 GB (15,4 GB usable).

III.3.2. Recursos de Software y Lenguaje de Programación

- **Windows 11 Pro (versión 24H2):** Sistema Operativo.
- **Visual Studio Code (versión 1.103.2):** Editor de Código.
- **Lenguaje de Programación Python (versión 3.12.10):** Lenguaje principal sobre el que se ha desarrollado toda la lógica del proyecto, debido a su versatilidad y el amplio ecosistema de bibliotecas disponibles (Wouters, 2025).
- **Piskel:** Aplicación de edición de imagen y creación de pixel art utilizada para el diseño y la elaboración de todos los recursos gráficos y sprites del videojuego (Descottes, 2017).

III.3.2.1. Bibliotecas de Python

- **Pygame (versión 2.6.1):** Biblioteca principal para el desarrollo del entorno gráfico del videojuego, la gestión de eventos (teclado, ratón) y la simulación interactiva de las partidas (*Pygame*, s. f.).
- **NumPy (versión 2.1.3):** Utilizada para operaciones de cálculo numérico y la gestión eficiente de matrices, fundamental para la lógica subyacente de la simulación.
- **Pandas (versión 2.3.1):** Empleada para la manipulación y el análisis de los datos generados durante las simulaciones.
- **Matplotlib (versión 3.10.3):** Usada para la creación de gráficos y la visualización de los resultados obtenidos.
- **Pickle (versión 4):** Biblioteca estándar de Python, utilizada para serializar y guardar el estado de los objetos y partidas, permitiendo la persistencia de datos.
- **Random (Seed 42):** Biblioteca estándar de Python, empleada para la generación de números pseudoaleatorios necesarios en la simulación de eventos estocásticos.

III.3.2.2. Gestión del código

- **Git:** Se utilizó el sistema de control de versiones Git para el seguimiento y la gestión de los cambios en el código.
- **Política de Versionado:** Se aplicó una política de versionado con Git, identificando cada nueva versión con un commit hash de confirmación único. Esto garantiza la trazabilidad y la reproducibilidad de las versiones.
- **Versión actual:** Git con el hash c5a61e5b0a4ebde0155ae591b45c3d1e4da8a867.

III.3.3. Herramientas de apoyo con IA

- **Google Gemini (2.5 Fast all-around help):** Modelo de lenguaje avanzado utilizado como asistente para la resolución de problemas lógicos, la optimización de algoritmos y el apoyo en el desarrollo de subprogramas.
- **GitHub Copilot (GPT-4.1):** Herramienta de autocompletado de código integrada en el editor, usada para agilizar el proceso de programación y la implementación de funcionalidades.

III.4. Procedimiento

III.4.1. Esquema de trabajo

Se parte de la realización de un esquema que sirva como guía para la realización de los módulos a utilizar. Este esquema consta de 5 módulos:

- El primero es el **módulo de Juego**, este es el principal y está dedicado a la configuración del juego, comunicándose con el resto de módulos mediante la realización de consultas para la obtención de información requerida.
- El segundo es el **módulo de Entorno,** este se encargará del diseño del campo de juego donde los agentes interactúan, además de las posibles acciones que estos agentes pueden tomar.
- El tercero es el **módulo de Entrenamiento**, este será el responsable de la gestión y organización de los entrenamientos de los agentes.
- El cuarto y quinto son los **módulos de Agentes**, son dos módulos idénticos, cada uno dedicado a la gestión y accionar de cada agente.

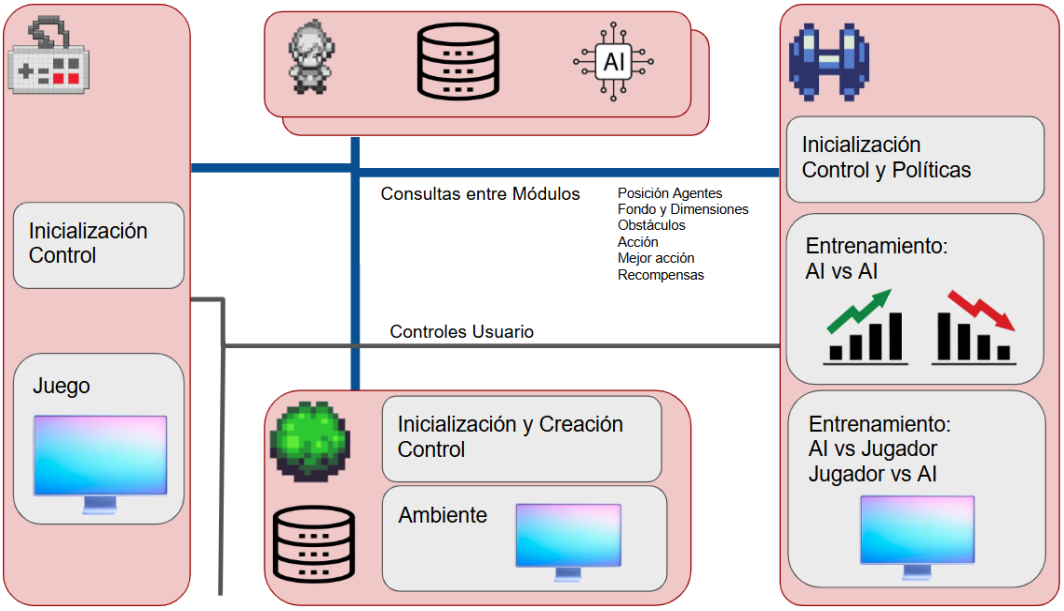

Fig. 7, Diagrama de Módulos

Considerando las restricciones de tiempo inherentes a la ejecución del proyecto, basamos la planificación inicial en un esquema de trabajo por entregable probados antes de integrarse, para generar avances específicos a producto terminado que nos permitan ir completando el proyecto.

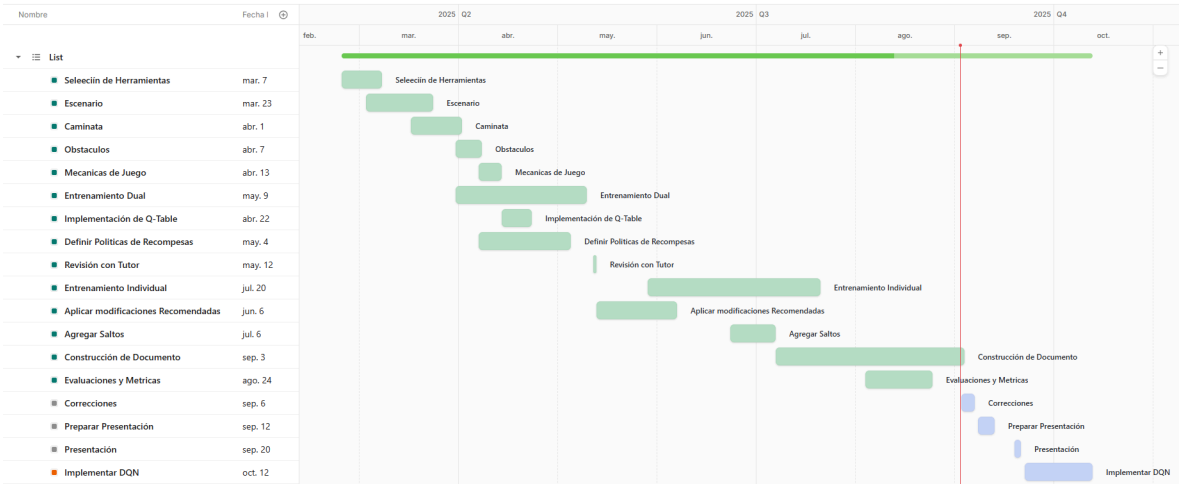

Fig. 8, Planificación de Trabajo

III.4.2. Herramientas para recursos gráficos y lógicas de juegos

Para la generación de recursos, como imágenes, y el desarrollo de subprogramas auxiliares, se han utilizado los módulos para edición de imágenes y los modelos de lenguaje de las herramientas de IA de código abierto Gemini (2.5) y Copilot (GPT-4.1).

III.4.2.1. Recursos Gráficos

Para la generación de recursos visuales, se llevó a cabo una serie de pruebas utilizando modelos de inteligencia artificial generativa. El proceso consistió en el uso de los módulos de gestión de imágenes de Gemini y Copilot, a los que se alimentó con imágenes de muestra, prompts y descripciones textuales detalladas.

Se emplearon diferentes *prompts* o instrucciones de texto, variando los parámetros de estilo, color y composición, con el objetivo de obtener imágenes de alta calidad que fueran coherentes con los requisitos del proyecto.

Fig. 9, Ejemplo de imagen de movimiento de personaje

La evaluación de los resultados se llevó a cabo de manera sistemática, comparando la fluidez vista de las imágenes generadas dentro de los módulos del juego.

III.4.2.2. Lógicas de Juegos

Para la generación de recursos de lógicas de juego, al igual que con las imágenes se llevó a cabo una serie de pruebas utilizando modelos de inteligencia artificial generativa. En esta oportunidad el proceso consistió en el uso de los módulos LLM de Gemini y Copilot, a los que se alimentó con prompts y descripciones textuales detalladas.

Se utilizaron variaciones de *prompts* e instrucciones de texto, variando las redacciones y alcance de los pedidos, con el objetivo de ver hasta donde se podían generar códigos funcionales.

La evaluación de resultados se llevó a cabo de manera sistemática, comparando funcionalidad, usabilidad e integración con cada uno de los módulos del proyecto.

III.4.3. Elaboración de entorno en Python

III.4.3.1. Selección de Aplicativo o librería para el Entorno

En este apartado se identifican las herramientas de inteligencia artificial de código abierto evaluadas para la generación de recursos de imágenes y el desarrollo de subprogramas auxiliares, luego realizamos la selección de las más adecuada para la realización del TFM.

Unity: Uno de los motores más populares y robustos, utilizado tanto por estudios independientes como por grandes empresas. Su versión Open Source (Personal) es muy completa y permite exportar a múltiples plataformas. Al igual que Unreal, tiene un umbral de costos antes de que se requiera una licencia de pago.

Se consideró como primera opción, ya que es una de las aplicaciones de desarrollo más populares del mercado y muchos de los juegos actuales en muchas plataformas son desarrollados con esta (Ej. Hollow Knight, Cuphead, Fall Guys y Among Us).

Utiliza el lenguaje de programación C# y actualmente está promocionando el uso de Agentes de IA dentro de su entorno. Dentro de sus ventajas encontramos que debido a su alta popularidad, dispone de una enorme cantidad de recursos disponibles, tutoriales, cursos y mucha documentación para aprender.

Si bien era una opción gratuita viable, se descartó. Esto se debe a que la implementación de agentes es en formato caja negra, y no es posible aplicar ninguna de las estrategias de Machine Learning o IA para configurar o modificar casos de estudios, diferentes a las presentadas por la herramienta.

Godot Engine: Una de las mejores opciones, es de código abierto, completamente gratuito y muy versátil para crear juegos 2D y 3D. Es muy popular entre desarrolladores independientes por su facilidad de uso y la activa comunidad.

Se estudió como alternativa durante el desarrollo, también cuenta con gran popularidad dentro de la comunidad de desarrollo de videojuegos. Esta herramienta, además de desarrollos 3D, nos presenta la posibilidad de realizar desarrollos en 2D, los cuales son más sencillos en cuanto a la cantidad de variables a manejar, así poder realizar un entregable más completo inicialmente.

Respecto a recursos de aprendizaje, pasa algo similar a Unity. Debido a su popularidad, es muy accesible a la hora de conseguir material de apoyo y tutoriales para implementar pequeños arreglos. Siguiendo en la línea de la implementación de IA, incluye una opción para desarrollo de agente de IA.

El uso de IA es más complejo de implementar que en el caso de Unity, aunque también da la posibilidad de poder acceder a interactuar con el desarrollo del agente. El problema en este caso, es que Godot utiliza principalmente un lenguaje de programación propio llamado GDScript.

Pygame (Pygame Software Foundation, 2024)**:** Especialmente enfocado en el desarrollo de juegos 2D. Una de las herramientas más prácticas, ya que es una librería de Python completamente Open Source. Muy accesible para desarrolladores sin experiencia que solo quieren realizar pruebas de funcionamiento o iniciar en el mundo del desarrollo de videojuegos.

Es la opción más genérica y sencilla de aplicar, ya que es una librería de Python. Al utilizar esta podemos evitar inconvenientes de incompatibilidad entre lenguajes y permite acceder a todas las variables y a todas las instancias del código, además de poder realizar fragmentos de código combinando módulos si es requerido.

Es la opción de menor curva de aprendizaje y para este punto proporciona la potencia de desarrollo necesaria para la realización de las pruebas que componen este trabajo.

III.4.3.2. Diseño de entorno

Para el desarrollo del entorno del videojuego en Python, se utilizó la librería Pygame, permitiendo emular la dinámica persecución-evasión, mediante la representación gráfica de los agentes y el área de juego. Además de esto, es un bloque de programación referencial para el funcionamiento de otros módulos y tiene dos funciones:

Creación de Entorno: Define las posibles acciones que pueden ser tomadas por los agentes, además de sus posiciones de partida. También, es el módulo encargado de crear y dar permitir a los usuarios poder visualizar el juego.

Calcular movimiento: Otra de las características de este módulo, es la evaluación y aplicación de movimientos según las acciones tomadas por los agentes.

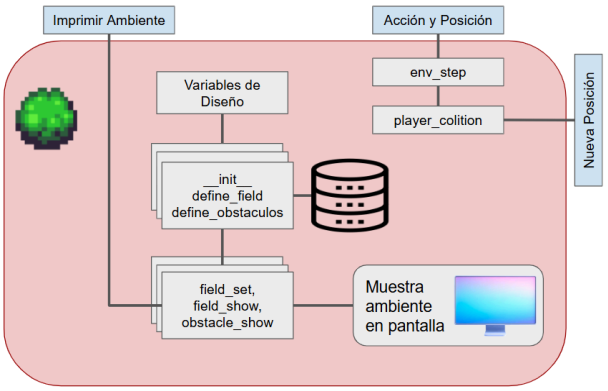

Fig. 10, Diagrama del Entorno de juego

Este entorno consta de tres etapas: en primer lugar tenemos la capa de fondo, luego contamos con la capa de obstáculos y por último las mecánicas de juego.

Capa de Fondo: Este delimita el área de movimiento, que se considera como el espacio donde se diseñarán rutas, colocarán los obstáculos y los jugadores para que estos puedan interactuar.

Dentro de este entorno se delimita la vista de la pantalla y define la cantidad de estados con los que vamos a estar interactuando dentro de las Q-Tables. El máximo de estados que tenemos por jugador, viene de la multiplicación de las coordenadas X, Y, Z, donde X, se refiere a la máxima cantidad posible de movimientos horizontales, o la distancia máxima en movimientos de izquierda a derecha.

Continuamos con Y, que se refiere a la máxima cantidad posible de movimientos verticales, o la máxima distancia en movimientos de arriba a abajo. Y por último Z, que se refiere a los posibles movimientos en el eje Z, como por ejemplo superficies en dos planos diferentes superpuestos.

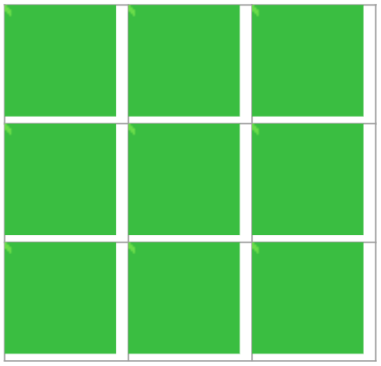

Fig. 11, Ejemplo de la capa de fondo del Entorno

Cálculo de ejemplo: para dos agentes dentro de un entorno de ejemplo con dimensiones 3x3.

Ag1: Agente 1
Ag2: Agente 2

X = 3
Y = 3
Z = 1 (un solo plano)

$$Estados = Estados(Ag\ 1) \times Estados(Ag\ 1)$$
$$Estados = (X \times Y \times Z) \times (X \times Y \times Z)$$
$$Estados = (3 \times 3 \times 1) \times (3 \times 3 \times 1)$$
$$Estados = 9 \times 9$$
$$Estados = 81$$

Esta capa de fondo, afecta directamente el aprendizaje de los agentes, ya que todos los datos y tablas van sincronizados con este diseño. Si por alguna razón este diseño cambia, va a afectar el resto de mecánicas y posibles acciones que los

agentes tomen. Ante cada modificación, se debe confirmar que los obstáculos y puntos de partida sean funcionales con el nuevo tamaño.

Obstáculos: Los obstáculos son imágenes agregadas sobre el fondo. Agregar obstáculos permite tener mayor diversidad de estrategias, crear caminos para estrategias, superficies donde los agentes se pueden subir para acceder a un segundo nivel o trampas que los agentes deberán esquivar.

Fig. 12, Ejemplo de Obstáculos

Es importante notar que cada obstáculo a agregar debe tener su mecánica de acción, ya sea un bloqueo de cuadrícula, trampa con alguna acción definida o una superficie para que los agentes accedan a un nivel superior. Esta mecánica debe estar correctamente definida antes de agregarla y se debe verificar el impacto que tiene sobre las políticas de recompensa actuales.

Acciones: Las acciones demarcan las direcciones y movimientos que los agentes pueden tomar. Estas variables se deben definir antes de iniciar los entrenamientos y afectan directamente al tamaño de las tablas usadas para tomar decisiones.

Además de esto, las variables deben estar diseñadas para poder funcionar con los obstáculos implantados. También, cada acción debe poder ser interpretada por las políticas de recompensa definidas.

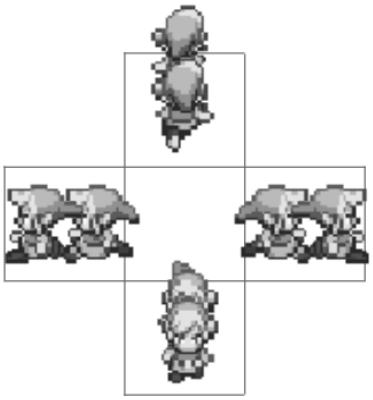

Fig. 13, Ejemplo de Direcciones de Movimiento

Mecánicas: son las interacciones y efectos entre agentes, obstáculos o usuarios.

- **Bloqueo**: es un objeto que se coloca en el fondo, impidiendo que el jugador avance en esa dirección.

Fig. 14, Ejemplo de Obstáculo

- **Agua:** se inserta la imagen en una región del fondo. El jugador que contacte con esto se ahogará y perderá instantáneamente.

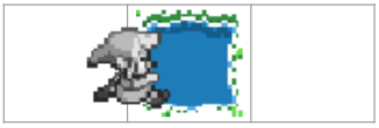

Fig. 15, Ejemplo de Ahogado

- **Salto:** son ramas u obstáculos particulares que bloquean el paso de los jugadores. Los agentes pueden evitarlos saltando.

Fig. 16, Ejemplo de Salto

- **Captura:** ocurre cuando los agentes tienen algún tipo de contacto entre ellos.

Fig. 17, Ejemplo de Captura

III.4.4. Diseño y configuración de entrenamientos

Este es el bloque encargado del control de los entrenamientos de los agentes. Si bien, desde el módulo de juego se puede consultar la Q-table, sólo durante la ejecución de este módulo se pueden modificar dichas tablas. Se implementaron dos flujos de entrenamiento con el mismo entorno, uno automático y otro donde usuarios humanos pueden controlar a uno de los agentes.

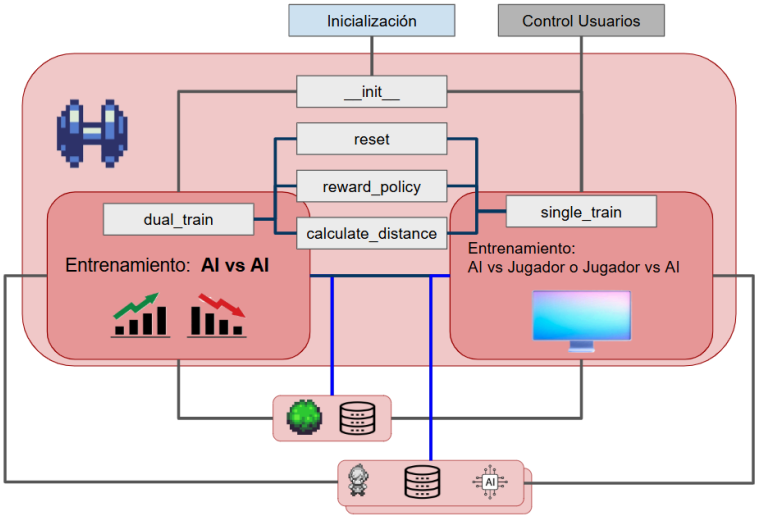

Fig. 18, Bloques de módulo de entrenamiento

Entrenamiento Automático: es una función del módulo de entrenamiento, donde los agentes entrenan solos, IA vs IA. Esta función está diseñada para entrenamientos de gran cantidad de sesiones y no para ser visualizado por usuarios, con la intención de que se puedan realizar la mayor cantidad de interacciones entre agentes en el menor tiempo posible. Durante este entrenamiento, los agentes interactúan en un entorno no visible, siguiendo las mismas configuraciones dentro de un entorno real y aprendiendo de todos los movimientos realizados.

Entrenamiento Individual: es una función del módulo de entrenamiento que cuenta con dos formas de uso. En el primer formato, el usuario puede seleccionar el agente que manejara, ya sea eligiendo ser “Chaser” o “Evader”. Esta opción se puede utilizar para corregir patrones repetitivos identificados durante el entrenamiento automático, como por ejemplo, uno de los agentes no posee datos en una condición en particular. El segundo formato de uso, es donde no interactúan los usuarios y dejan que los agentes interactúen entre ellos.

La diferencia de este formato con respecto al Automático, es que esta versión está diseñada para que se corrijan o mejoren condiciones vistas durante la ejecución del juego principal o para realizar entrenamientos bajo la supervisión visual, de interacciones entre agentes, de un usuario. De esta manera, poder validar si las interacciones están siendo efectivas o se pueden identificar puntos de mejora.

El funcionamiento de la dinámica de los dos entrenamientos es similar. En un principio los agentes toman movimientos al azar de las acciones previamente definidas en el entorno. Posteriormente, a medida que van avanzando las iteraciones, aplicamos la política de E-greedy, haciendo que disminuya el nivel de aleatoriedad, valor de epsilon, de las acciones por cada jugada realizada. De este modo, nos permite recaer más en los valores aprendidos por todas las iteraciones pasadas.

III.4.5. Sistema de recompensas para optimización de aprendizaje

Según lo indicado en nuestro objetivo de elaboración de sistema de recompensas, debemos garantizar que dicho sistema incentive las conductas deseadas (captura exitosa, evasión prolongada, etc.), y que estas sean estén acordes al entorno de entrenamiento diseñado, permitiendo a los agentes utilizar estas recompensas para aprender a través de la experiencia.

III.4.5.1. Recompensa por movimiento

Con el objetivo de crear el incentivo a mantenerse en constante movimiento y buscando la mayor recompensa posible, se implementó un sistema de penalización por movimiento. Este sistema aplica un valor de recompensa negativa a cada acción realizada por los agentes, para fomentar la toma de decisiones eficiente.

También, con la intención de fomentar el objetivo persecución-evasión, incorporamos recompensas dinámicas basadas en la distancia entre agentes.

Al perseguidor (Chaser) se le asigna una recompensa inversamente proporcional a su distancia con el evasor (Evader), es decir, la recompensa aumenta a medida que se acerca a él. Por otra parte, el Evader recibe una recompensa más alta cuanto más lejos se encuentre del Chaser.

Para poder obtener estos valores necesitamos realizar el cálculos de distancia, el cual se realiza de la siguiente manera:

$$distancia = \sqrt{\left(X_2 - X_1\right)^2 + \left(Y_2 - Y_1\right)^2}$$

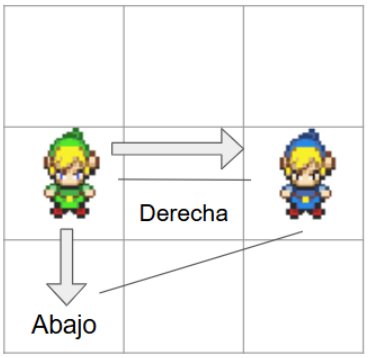

Fig. 19, Ejemplo de cálculo de recompensa

III.4.5.1.1. **Movimiento a la derecha del Chaser**

Se calcula la distancia entre puntos desde la posición final del Chaser hasta la posición del Evader.

Posición Chaser: $X_1 = 1, Y_1 = 1$

Posición Evader: $X_2 = 2, Y_2 = 1$

$$d = \sqrt{\left(X_2 - X_1\right)^2 + \left(Y_2 - Y_1\right)^2}$$

$$\sqrt{(2-1)^2 + (1-1)^2}$$

$$\sqrt{(1)^2 + (0)^2}$$

$$\sqrt{1+0} = 1$$

La recompensa del “Chaser” por realizar un movimiento a la derecha acercándose al “Evader” será de -1. Siempre obteniendo una recompensa negativa, siendo esta menor a medida de que esté más cerca al objetivo.

III.4.5.1.2. Movimiento hacia abajo del Chaser

Se calcula la distancia entre puntos desde la posición final del Chaser hasta la posición del Evader.

Posición Chaser: $X_1 = 0, Y_1 = 2$

Posición Evader: $X_2 = 2, Y_2 = 1$

$$d = \sqrt{\left(X_2 - X_1\right)^2 + \left(Y_2 - Y_1\right)^2}$$

$$\sqrt{(2-0)^2 + (1-2)^2}$$

$$\sqrt{(2)^2 + (-1)^2}$$

$$\sqrt{4+1} = 2,23$$

La recompensa del “Chaser” por realizar un movimiento hacia abajo alejándose del “Evader” será de -2,23. Siempre obteniendo una recompensa negativa, siendo esta mayor a medida de que esté más lejos del objetivo.

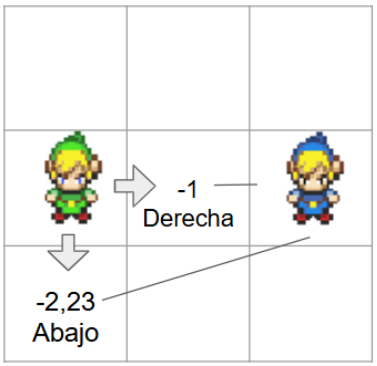

Fig. 20, Ejemplo de cálculo de recompensa Chaser

III.4.5.1.3. Movimiento hacia abajo del Evader

Se calcula la distancia entre puntos desde la posición final del Chaser hasta la posición del Evader.

Posición Chaser:
X_1 = 0, Y_1 = 1

Posición Evader:
X_2 = 2, Y_2 = 2

$$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

$$d = \sqrt{(2 - 0)^2 + (2 - 1)^2}$$

$$d = \sqrt{(2)^2 + (1)^2}$$

$$d = \sqrt{4 + 1} = 2,23$$

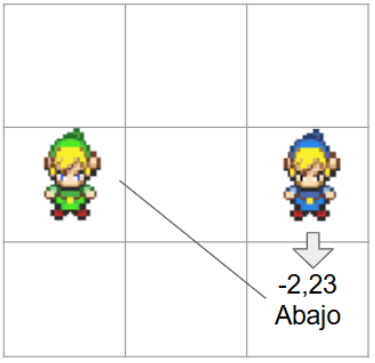

Fig. 21, Ejemplo de cálculo de recompensa Evader

Cálculo de la distancia máxima posible, para realizar la operación que garantice que el evader se aleje del Chaser.

Tamaño del Fondo:

X_0 = 0, Y_0 = 0

X_{max} = 3, Y_{max} = 3

$$d_{max} = \sqrt{\left(X_{max} - X_0\right)^2 + \left(Y_{max} - Y_0\right)^2}$$

$$d_{max} = \sqrt{(3 - 0)^2 + (3 - 0)^2}$$

$$d_{max} = 4{,}24164$$

recompensa = $-\left(d_{max} - d\right)$ = - (4,24164 - 2,23) = - 2,01164

La recompensa del “Evader” por realizar un movimiento hacia abajo alejándose del “Chaser” será la resta de la distancia máxima posible menos (4,24164) menos la distancia a la que quedará (2,23), quedando una recompensa total de -2,01164. Siempre obteniendo una recompensa negativa, siendo esta mayor a medida de que esté más lejos del objetivo.

III.4.5.2. Recompensa por Saltos

Esta recompensa se aplica siempre que se haga la acción de saltar en cualquiera de las direcciones, de esta manera podemos garantizar que el jugador utilice el salto solo en la condición necesaria

Partiendo del cálculo del ejemplo anterior, podemos ver que en el movimiento a la derecha representado en la imagen tendría una recompensa de -1.

Fig. 22, Imagen recompensa movimiento

Siguiendo la misma línea, se emuló el salto en la misma dirección, en esta oportunidad la recompensa será de -1 por el movimiento a la derecha, más -0.1 por realizar el salto. De esta manera la recompensa final para esa acción será de -1.1.

Fig. 23, Imagen recompensa salto

Con esto se garantiza que siempre los valores de recompensa en la Q-table son mayores para la caminata que para el salto, de esta manera evitamos que el personaje salte, y lo haga solo cuando sea necesario.

III.4.5.3. Recomple por contacto con obstáculos

Esta recompensa se aplica cada vez que un jugador hace un movimiento en sentido de un obstáculo y, debido al obstáculo, no se puede generar el movimiento en esa dirección.

Fig. 24, Imagen recompensa obstáculo

Partiendo del mismo ejemplo de movimiento a la derecha trabajado. Al realizar un movimiento a la derecha, la recompensa será de -1 por el movimiento a la derecha, más -0.5 por chocar contra el obstáculo. De esta manera la recompensa final para esa acción será de -1.5, contra los -1.1 que sería al saltar.

III.4.5.4. Recompensa por tocar el Agua

Esta recompensa se aplica cada vez que un jugador hace un movimiento en sentido de los pozos de agua y, debido a esto, el jugador pierde directamente.

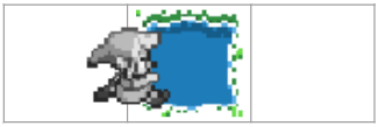

Fig. 25, Imagen recompensa tocar agua

Con esto garantizamos que siempre los valores de recompensa en la Q-table cuyo movimiento haga que el personaje entre en el agua, sean menores. De esta manera evitamos que los personajes se muevan en dirección al agua.

III.4.5.5. Recompensa de victoria

Esta recompensa se aplica cada vez que un jugador hace un movimiento en sentido del otro jugador y, debido a esto, el Chaser gana y el Evader pierde.

Fig. 26, Imagen recompensa victoria

Con esto garantizamos que siempre los valores de recompensa en la Q-table cuyo movimiento haga que los personajes entren en contacto, sean mayores para el Chaser y menores para el Evader. De esta manera garantizamos que los dos jugadores tengan una motivación para hacer los movimientos de persecución-evasión.

III.4.6. Implementación de agentes con machine learning por refuerzo

Según nos pide uno nuestro objetivo, debemos Implementar dos agentes inteligentes (perseguidor y evasor) utilizando algoritmos de RL, como Q-learning o Deep Q-Network (DQN), configurando adecuadamente los estados, acciones y recompensas. Este módulo cuenta con tres responsabilidades principales, además de esto maneja tres entradas y dos salidas.

Inicialización: es la primera etapa que actúa al momento de configurar el agente. Esta es activada a partir de los módulos de entrenamiento o juego para inicializar a los agentes, mediante la preparación de todas las variables.

Acción y Posición: esta es una etapa de uso recurrente y es la encargada de recibir la posición actual del agente para luego buscar la mejor acción dentro de la memoria del agente.

Guardar: esta etapa puede ser de uso recurrente y es utilizada por el módulo de entrenamiento para hacer una copia de la tabla de datos vigente dentro de la memoria a largo plazo del Agente.

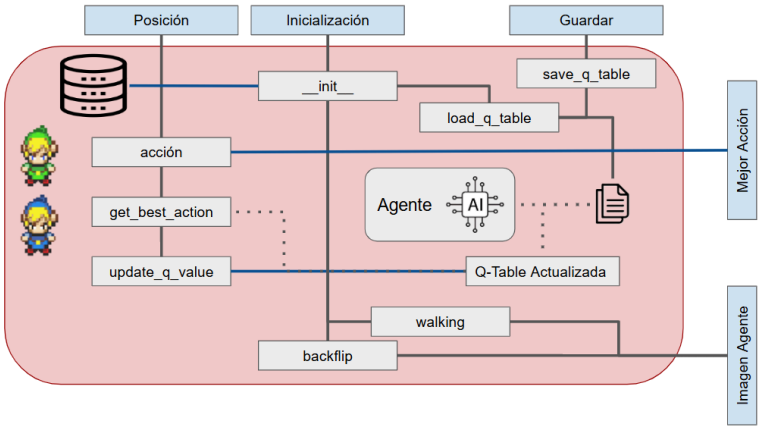

Fig. 27, Modulos de Agentes

Mejor Acción: esta variable de salida contiene la mejor acción que puede tomar el agente en función a la entrada recibida.

Imagen Agente: esta variable de salida contiene la imagen que debe mostrar el agente en función a la acción y posición.

III.4.7. Entrenamiento y prueba de los agentes

El desarrollo y las pruebas se realizaron en un entorno controlado, con el objetivo de validar la funcionalidad del sistema y analizar el comportamiento de los agentes en diferentes escenarios, ajustando parámetros para comparar resultados bajo distintas condiciones experimentales.

Ejecutar múltiples episodios de entrenamiento, ajustando parámetros como tasa de exploración (ε), tasa de aprendizaje (α) y gamma (γ) para optimizar el aprendizaje de los agentes.

Estados	Acciones								
	(0,0,0) Quieto	(0,-1,0) Arriba	(0,1,0) Abajo	(-1,0,0) Izquierda	(1,0,0) Derecha	(0,-1,1) Salto Arriba	(0,1,1) Salto Abajo	(-1,0,1) Salto Izquierda	(1,0,0) Salto Derecha
[(0, 0), (2, 2)]									
[(0, 1), (2, 2)]									
[(0, 2), (2, 2)]									
[(1, 0), (2, 2)]									
[(1, 1), (2, 2)]									
[(1, 2), (2, 2)]									

Fig. 28, Imagen Muestra de Q-table

III.5. Análisis de datos

III.5.1. Validación de resultados en Aplicación

Durante la realización de pruebas con el usuario humano, comparamos el desempeño de los agentes con el del participante evaluador, recogiendo información cuantitativa y cualitativa sobre la experiencia.

Estas pruebas fueron realizadas dentro del entorno de juego principal, debido a que este permite que los jugadores interactúen sin afectar los valores de Q-Tables y mientras que los agentes basan sus acciones completamente de los valores almacenados en dichas tablas.

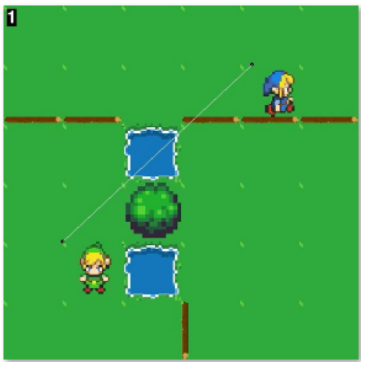

Fig. 29, Imagen Muestra plataforma de juego

Dentro de este, se validó la interacción de los agentes, ya sea en su rol de Chaser o de Evader, pudiendo sacar apreciaciones sobre los accionares y dificultades que les propusieron los agentes con sus acciones.

III.5.2. Validación de Resultados de Entrenamiento

Durante la realización de entrenamientos, además de los registros de las Q-Tables, se almacenaron resultados de cada uno de los bloques repetitivos. Dentro de estos bloques, que sirven como punto de control, se fueron almacenando los resultados de cada episodio, incluyendo métricas de desempeño, logs de decisiones y evolución de las políticas de los agentes.

Al finalizar cada bloque de entrenamiento, se obtuvieron las siguientes gráficas resultado:

III.5.2.1. Gráfica de Bigote

Esta es la primera gráfica obtenida, muestra los avances en cada uno de los bloques de entrenamiento. Esta permite visualizar el avance del entrenamiento dándonos indicios de la evolución del entrenamiento para decidir si se debe o no intervenir.

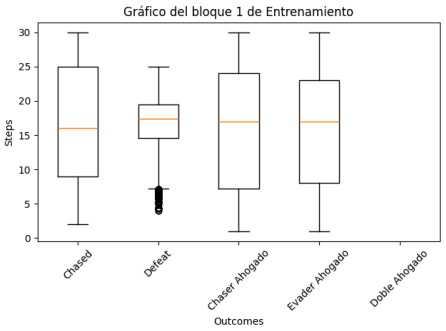

Fig. 30, Ejemplo gráfico de bigote intermedio

En esta gráfica veremos 5 columnas, cada una reflejando una condición en especial.

Chased: refleja cuando el Chaser logra alcanzar el objetivo. Se grafica un punto con el valor del paso (step) en el que se logró alcanzar el objetivo. Con esta representación podemos apreciar el punto en el cual se está logrando el objetivo.

De esta forma se puede inferir, por ejemplo, si el promedio está en la parte baja de la gráfica, quiere decir que el agente está cumpliendo su objetivo rápidamente.

Defeated: refleja cuando el Evader logra escapar. Se grafica un punto con el valor de la distancia promedio entre Chaser y Evader durante la ronda. Con esta representación se puede validar si el Evader está logrando mantenerse distanciado del Chaser.

De esta forma se puede inferir, por ejemplo, si el promedio de estos valores está en la parte baja de la gráfica, quiere decir que el agente no está logrando mantenerse alejado.

Chaser Ahogado y Evader Ahogado: refleja cuando el Chaser o Evader hacen un movimiento en sentido del agua teniendo contacto con ella. Se grafica un punto con el valor del paso (step) en el que el jugador toca el agua. Con esta representación se puede apreciar el punto en el cual alguno de los dos jugadores pierde.

De esta manera se confirma, en qué momento alguno de los jugadores pierde por ahogamiento.

Doble Ahogado: funciona muy similar a los valores obtenidos en las columnas “Ahogado”, con la diferencia de que acá se toma en cuenta cuando los dos jugadores tocan el agua en simultáneo.

III.5.2.2. Grafica total de resultados

Es la gráfica de cierre del entrenamiento automático, es un gráfico de líneas que nos permitirá ver los resultados de avances que vamos a ir teniendo después de cada bloque de entrenamiento.

La gráfica muestra el comportamiento global de los agentes, el dominio de un agente sobre el otro o la afectación producida por el entorno.

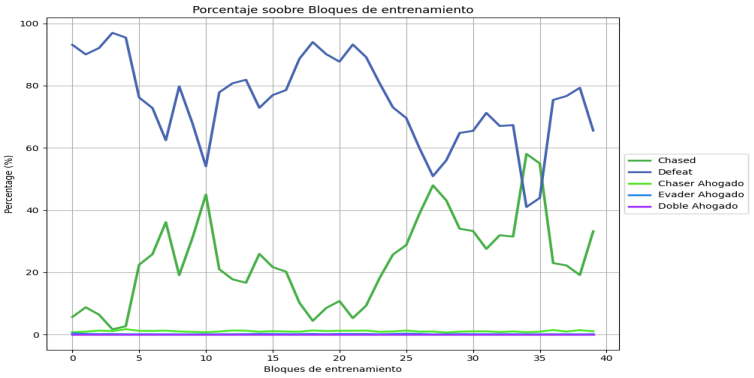

Fig. 31, Ejemplo gráfico de Línea evaluativo

Debido a que los valores obtenidos en cada bloque provienen de un conjunto de jugadas realizadas por los agentes, se decidió avanzar con esta gráfica como la definitiva para la toma de decisiones.

En otras palabras, esta gráfica decide si los agentes son funcionales, en cuanto a su desenvolvimiento dentro de las jugadas.

III.5.2.3. Análisis de Tabla de Valores (Q-Tables)

Dentro de las herramientas con las que contamos, se cuenta con la posibilidad de acceder a los valores de las Q-Tables para visualización. Esto permite analizar, en posiciones particulares, la existencia de valores faltantes o fuera de lo común. De esta manera se usó para el análisis de causa raíz, como por ejemplo, en un bucle de movimiento dentro del accionar de los agentes.

Estados	Acciones								
	(0,0,0) Quieto	(0,-1,0) Arriba	(0,1,0) Abajo	(-1,0,0) Izquierda	(1,0,0) Derecha	(0,-1,1) Salto Arriba	(0,1,1) Salto Abajo	(-1,0,1) Salto Izquierda	(1,0,0) Salto Derecha
((5, 4, 0), (5, 2, 0))	-6,72313338	-140,76897677	-5,52848633	-6,68371793	2,82560557	-40,91517323	-6,59652289	-6,70567397	-6,73727889
((5, 3, 0), (5, 1, 0))	-6,72379181	-55,81512469	-3,51257189	-6,75439693	-6,73971575	-77,27028556	-6,66509036	-6,71287023	-6,70447592
((5, 1, 0), (5, 1, 0))	0,00000000	0,00000000	0,00000000	0,00000000	0,00000000	0,00000000	0,00000000	0,00000000	0,00000000
((5, 2, 0), (4, 4, 0))	-5,04952410	-5,44703569	-6,94799360	-6,43503303	-6,47535502	3,30854107	-7,04638421	-6,50726011	-5,45726502
((2, 2, 0), (1, 3, 0))	0,00000000	0,00000000	0,00000000	0,00000000	0,00000000	0,00000000	0,00000000	0,00000000	0,00000000

Fig. 32, Ejemplo de Q-table

Como ejemplos de análisis, en el primer estado dentro de esta tabla. Se observa que el valor de movimiento a la derecha está totalmente incorrecto, ya que por la posición, no se permite mover a la derecha, por tanto no permite al agente tomar las acciones de movimiento hacia abajo, siendo estas las correctas, dando prioridad la del paso abajo en lugar del salto.

IV. Resultados

IV.1. Generación de Imágenes

Esta etapa del proceso consistió en la aplicación de distintos prompts descriptivos que detallan las características para las imágenes deseadas de los personajes. Continuamente se fueron ajustando los prompts para poder guiar a los modelos a generar mejores resultados permitiendo que se adapten a nuestro proyecto.

Prompt usado: “Reemplaza al personaje presente en esta imagen por uno similar a Viví el de Final Fantasy 9. Ten en cuenta que las imágenes tienen un sentido y dirección de movimiento”

Las imágenes generadas por Gemini, fueron mejorando con cada actualización de prompt, logrando mostrar resultados con buen nivel de detalle. Estas imágenes todavía presentan problemas de coherencia entre los movimientos hacia arriba, abajo y de los saltos.

Fig. 33, Mejor imagen obtenida IA Gemini

Al igual que Gemini, la herramienta Copilot mejoró los detalles de las imágenes con cada mejora del prompt, pero aún no logra generar completamente coherencia entre los movimientos hacia arriba y en los saltos.

Fig. 34, Mejor imagen obtenida IA Copilot

En ambos casos, las imágenes deben ser recortadas individualmente para armar un nuevo conjunto de imágenes funcional.

IV.2. Entrenamientos

En esta etapa del proceso, se establecen las condiciones iniciales y con las que se realizaron las pruebas. De esta manera se evaluará la efectividad de los agentes dentro del entorno y su capacidad de aprendizaje.

Los parámetros iniciales y constantes durante todas las pruebas a realizar serán:

- **Entorno no variable**, permanecerá con las mismas dimensiones, 5x5 y la misma disposición de obstáculos.
- **Accionar fijo,** para los dos agentes, las acciones permitidas serán: estático, arriba, abajo, izquierda, derecha, saltar arriba, saltar abajo, salto izquierda y salto derecha.
- **Orden de accionar de los agentes**, primero Evader y luego Chaser.
- **Posición inicial de los agentes,** se dispondrán aleatoriamente dentro de estas cuatro coordenadas, (1, 1, 0), (4, 4, 0), (1, 4, 0) y (4, 1, 0), excluyendo salidas en la misma posición.
- **Máxima cantidad de pasos** será de 30
- **Valor mínimo de Epsilon** en 0.01 (1%)

- **Valor de ratio de aprendizaje** (learning rate) en 0.01 (1%)
- **Factor de descuento** (discount factor) en 0.1 (1%)
- Cada entrenamiento consta de 50 bloques de 147.456 jugadas por bloque.

IV.2.1. Primera ronda de entrenamiento.

En la primera ronda de entrenamiento se inicia con todos los valores de Q-Tables en cero. Las variables particulares de este entrenamiento, además de las condiciones generales antes mencionadas, serán:

- Epsilon inicial de 1
- Disminución de Epsilon (Epsilon decay) de 0.98 por jugada.

IV.2.1.1. Resultados Gráficos

A continuación se mostrarán los resultados de 3 de los 50 bloques de entrenamiento realizados.

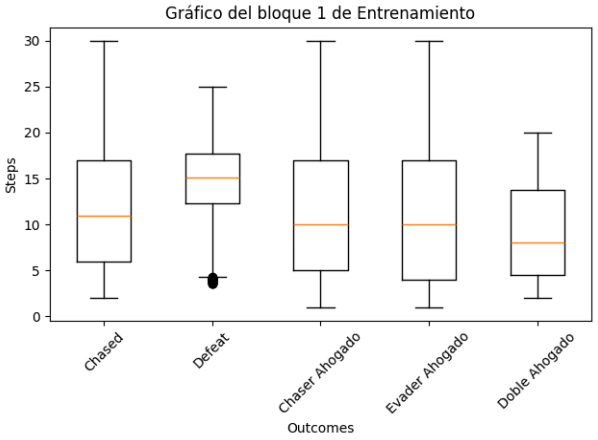

Fig. 35, Gráfico de Bigote bloque 1, 1er Entrenamiento

Durante este bloque de entrenamiento el Epsilon Inicial fue del 100%, al final del bloque y por el resto del entrenamiento quedó en 0%.

Inicia con victoria clara para el Chaser, logrando capturar a su objetivo en el 75,85% de las 147.412 jugadas, y en promedio logró realizar la captura en el paso número 11. Contra el 22% de evasiones realizadas por parte del Evader.

Revisando el desenvolvimiento de Evader, se aprecia que en promedio permanece al 50% de la distancia máxima posible con respecto al Chaser.

Con respecto al ahogamiento de los agentes, a pesar de ser valores bajos, el Chaser se ahogó en el 1,53% de las veces, esto es casi el triple de veces que el

Evader con 0,58% de las veces. También en 10 oportunidades los dos agentes se ahogan a la vez.

Durante los próximos 24 bloques de entrenamiento se invierte el ganador y es el Evader que supera al Chaser, con más del 60% de victorias por cada bloque. Mientras que el Chaser no logra superar el 40% en dos bloques seguidos. A partir del bloque 4, dejan de ahogarse los dos agentes a la vez.

Estos valores se pueden consultar en la gráfica total mostrada más adelante o en el archivo de entrenamiento dentro del enlace a git.

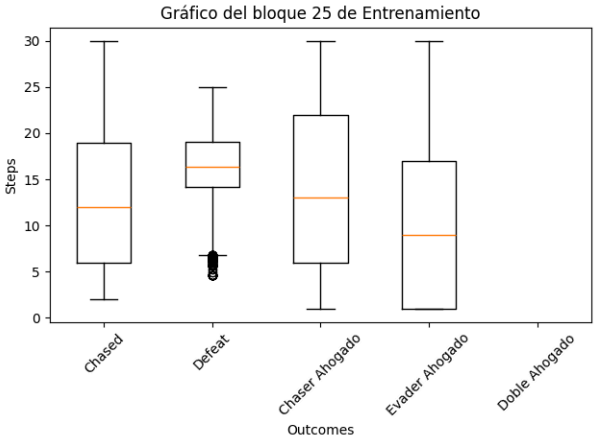

Fig. 36, Gráfico de Bigote bloque 25, 1er Entrenamiento

Resultado a mitad de entrenamiento con un Epsilon de 0%.

Evader obtiene la victoria, logrando escapar de su rival el 67,01% de las 147.455 jugadas, logrando también aumentar la distancia máxima con respecto al Chaser de 50% a 53,3%.

Contra el 31,7% de capturas realizadas por parte del Chaser, que logró mantener el promedio de capturas en el paso 11.

Con respecto al ahogamiento de los agentes, los dos lograron reducir las jugadas que terminan ahogados, el Chaser se ahogó en el 1,1% de las veces, mientras que el Evader con 0,19% de las veces.

Desde acá hasta el último bloque, los resultados de victorias van oscilando entre Chaser - Evader. Se puede consultar en la gráfica de línea mostrada más adelante o en el archivo de entrenamiento.

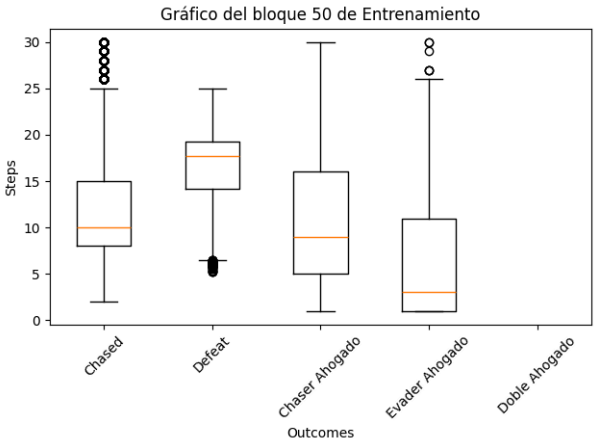

Fig. 37, Gráfico de Bigote bloque 50, 1er Entrenamiento

Para cierre del entrenamiento la victoria fue para el Chaser, logrando capturar de su rival el 67,91% de las 147.455 jugadas, bajando el promedio de captura al paso 10.

Contra el 30,98% de evasiones realizadas por parte del Evader, volvió a aumentar su distancia máxima promedio a un 60%.

Con respecto al ahogamiento de los agentes, volvieron a reducir las jugadas que terminan ahogados, el Chaser se ahogó en el 0,95% de las veces, mientras que el Evader con 0,16% de las veces.

Al resumir y graficar todos los porcentajes de acierto obtenidos en cada una de los 50 bloques del entrenamiento, se obtiene el avance y rendimiento de los agentes durante todo el entrenamiento.

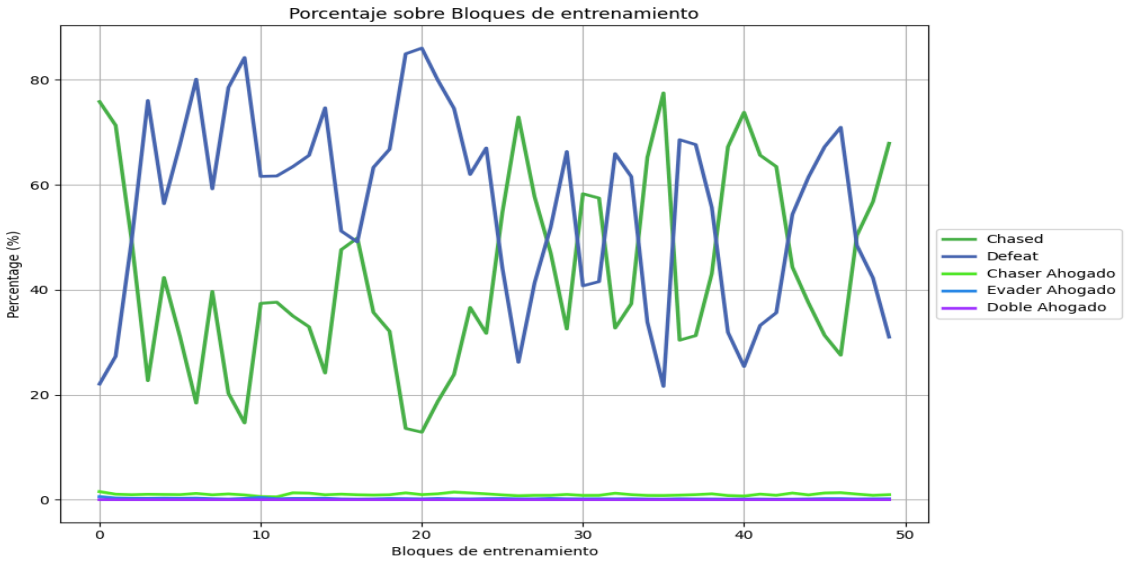

Fig. 38, Gráfico de Línea 1er Entrenamiento

En la gráfica se observó claramente, cómo los agentes aprendieron rápidamente a evadir el agua, siendo muy bajos los valores de ahogamiento durante todo el entrenamiento.

La asertividad de los agentes dentro de sus respectivos objetivos fue mixta, pero variando dentro del 50%, con la particularidad de que en las primeras etapas del entrenamiento le cuesta un poco más al Chaser alcanzar el objetivo.

Estos resultados los analisaremos y compararemos con el resto de rondas de entrenamiento en el apartado de discuciones de este TFM.

IV.2.1.2. Resultado en Entorno principal de Juego

En esta evaluación se puso a prueba funcional el entorno principal de juego con el entrenamiento realizado, y se observaron los comportamientos de los agentes. En esta prueba se observó cómo realizan las acciones y que tipo de acciones están tomando los agentes.

Dentro de las jugadas ejecutadas destacan:

- Cumpliendo el objetivo de persecución-evasión.
- Interacción correcta con el entorno, moviéndose correctamente y respetando el funcionamiento de los obstáculos.
- Usan el accionar de salto como movimiento en lugares que no corresponde.
- Las jugadas vistas son repetitivas y limitadas.
- Se detectan bucles infinitos.

Fig. 39, Muestra de Juego 1er Entrenamiento

En la posición mostrada en la imagen, los agentes están haciendo los movimientos correctos según lo indicado en su Q-Table, pero al no haber restricciones de contacto con paredes, ciclos activos para evitar bucles o aleatoriedad en los movimientos, estos se quedan en este estado hasta reiniciar la Jugada.

IV.2.2. Segunda ronda de entrenamiento

En la segunda ronda de entrenamiento se partió con todos los valores de Q-Tables en cero. Para diferenciarlo del entrenamiento del anterior, fue planteado un Epsilon que variará en función del cambio de bloque de entrenamiento y a su vez, variaciones muy pequeñas dentro de cada jugada del bloque.

Las variables particulares de este entrenamiento, además de las condiciones generales antes mencionadas, serán:

- Epsilon inicial de 1
- Disminución de Epsilon (Epsilon decay) de 0,02 por bloque
- Disminución de Epsilon interna al bloque de 0,999999 por jugada (solo afecta al bloque).

IV.2.2.1. Resultados Gráficos

A continuación se muestran los resultados de 3 de los 50 bloques de entrenamiento realizados.

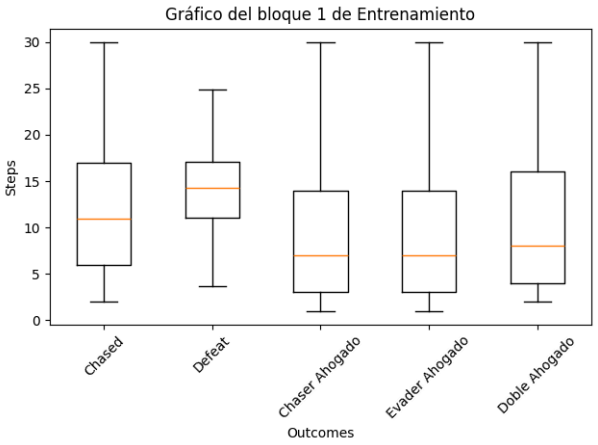

Fig. 40, Gráfico de Bigote bloque 1, 2do Entrenamiento

Durante este primer bloque, el Epsilon osciló entre 100% como valor máximo y 86,29% como valor mínimo.

Inició con una victoria clara para el Chaser, logrando capturar a su objetivo en el 15,42% de las 144.221 jugadas, que en promedio logró realizar la captura en el paso número 11.

Contra el 11,08% de evasiones realizadas por parte del Evader. Revisando el desenvolvimiento de Evader, se aprecia que en promedio permanece al 46,6% de la distancia máxima posible con respecto al Chaser.

En cuanto al ahogamiento de los agentes, en esta oportunidad fueron valores altos, el Chaser se ahogó en el 38,36% de las veces, muy cercano está el Evader con 34,77% de las veces. También en 528 oportunidades los dos agentes se ahogan a la vez, que equivale al 0.37% de las jugadas.

Para el cálculo de valor inicial de Epsilon del siguiente bloque, se le aplicó una reducción de 2% al Epsilon Inicial del bloque anterior (en este caso, 100% en el bloque 1), quedando en 98% como valor inicial del siguiente bloque.

Durante los siguientes 24 bloques de entrenamiento el Chaser siempre predomina en victorias, aumentando con cada bloque la diferencia porcentual respecto a las victorias del Evader. También se aprecia una reducción gradual en la cantidad de jugadas que los dos agentes terminan ahogándose, siendo el Evader que aprende con una pendiente más elevada.

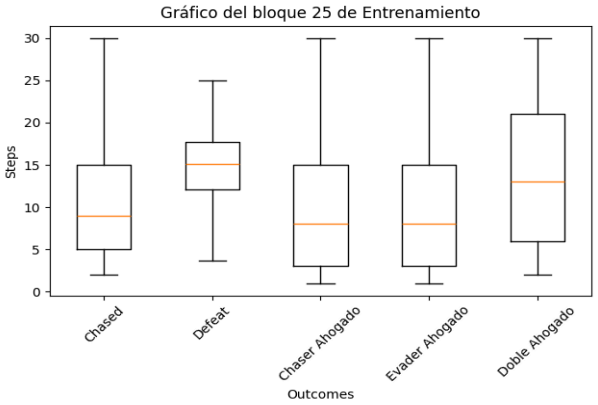

Fig. 41, Gráfico de Bigote bloque 25, 2do Entrenamiento

Durante el bloque 25, el Epsilon osciló entre 50% como valor máximo y 11,44% como valor mínimo.

Al igual que en el análisis anterior, inicia con victoria clara para el Chaser, logrando capturar a su objetivo en el 43,28% de las 145.735 jugadas, además logró bajar el promedio de captura a 9 pasos.

Mientras el Evader logró evadir sólo el 13,75% de las jugadas. Pero en esta oportunidad logró aumentar la distancia máxima posible promedio a un 50% con respecto al Chaser.

En cuanto al ahogamiento de los agentes, en esta oportunidad siguen siendo valores altos pero hay disminución de ellos, el Chaser se ahogó en el 27,73% de las veces, y con una mayor corrección está el Evader con 15,12% de las veces. También en 161 oportunidades los dos agentes se ahogan a la vez, que equivale al 0,11% de las jugadas.

Para el cálculo de valor inicial de Epsilon del siguiente bloque, se le aplicó una reducción de 2% al Epsilon Inicial del bloque anterior (en este caso, 50% en el bloque 25), quedando en 48% como valor inicial para el bloque 26.

Durante los siguientes 24 bloques de entrenamiento el Chaser siempre predomina en victorias, aumentando con cada bloque la diferencia porcentual respecto a las victorias del Evader hasta el bloque 40.

En este punto el Chaser muestra indicios de estabilidad en el valor porcentual de victorias, hasta los últimos dos bloques, donde se desploma al 50%. Continúa la reducción gradual en la cantidad de jugadas que los dos agentes terminan ahogándose, siendo el Evader que aprende con una pendiente más elevada.

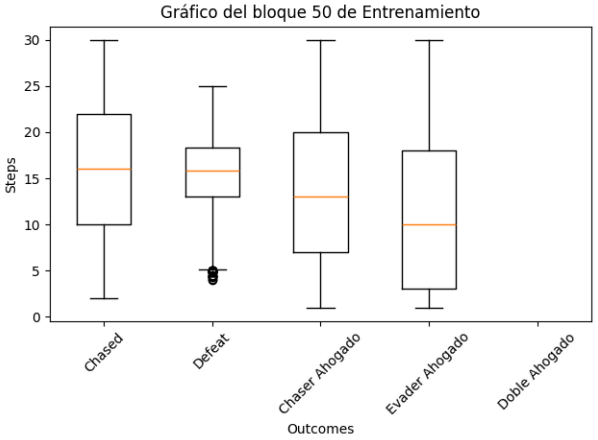

Fig. 42, Gráfico de Bigote bloque 50, 2do Entrenamiento

Durante el bloque 25, el Epsilon osciló entre 2% como valor máximo y 1,73% como valor mínimo.

Al igual que en el análisis anterior, continúa con victoria para el Chaser, logrando capturar a su objetivo en el 48,99% de las 147.454 jugadas, en esta oportunidad aumentó el promedio de captura a 16 pasos.

Mientras el Evader redujo mucho la ventaja logrando evadir el 48,60% de las jugadas. Además de aumentar la distancia máxima posible promedio a un 53,3% con respecto al Chaser.

En cuanto al ahogamiento de los agentes, en esta oportunidad hubo una mejora substancial, el Chaser se ahogó en el 1,93% de las veces, y con una mayor corrección está el Evader con 0,48% de las veces. Solo en este último bloque, no hubo ahogamiento simultáneo de los dos agentes.

Al graficar los porcentajes de acierto obtenidos en cada una de los 50 bloques del entrenamiento, con la intención de poder apreciar el avance y

rendimiento de los agentes durante todo el entrenamiento, se obtuvo la siguiente gráfica.

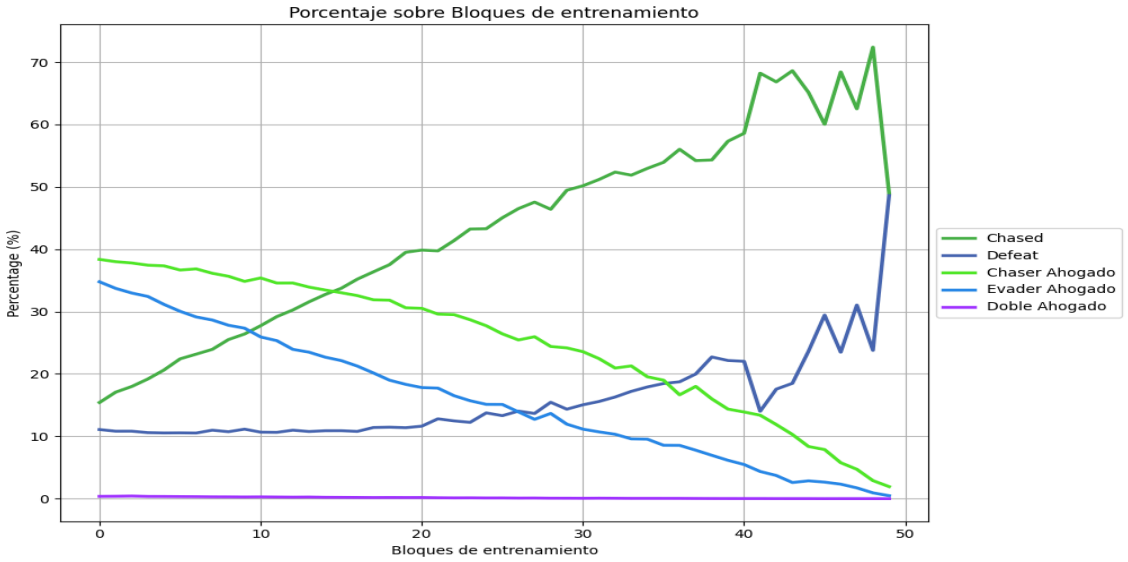

Fig. 43, Gráfico de Línea 2do Entrenamiento

Como observaciones dentro de este análisis, se puede confirmar que a medida que avanzan los bloques de entrenamiento los dos agentes se hacen más asertivos. Dejan de moverse en dirección al agua.

Como acotación, durante el desarrollo de los bloques de entrenamiento la pendiente de mejora del Evasor es baja, esto se mantiene durante todo el entrenamiento.

Además de esto, se puede observar cómo los agentes, a medida que avanzan los bloques, se van ahogando menos, confirmando las mismas apreciaciones vistas en el diagrama de bigote. También se confirma la tendencia alcista en la asertividad de los agentes para cumplir sus objetivos.

IV.2.2.2. Resultado en Entorno de Juego

Al igual que en la primera ronda de entrenamiento, se probó la funcionalidad del modelo en el entorno principal de juego.

Dentro de las jugadas ejecutadas, podemos destacar:

- No cumple el objetivo de Persecución-Evasión.
- Interactúan correctamente dentro del entorno, moviéndose correctamente y respetando el funcionamiento de los obstáculos.
- Usa en menor medida los saltos al moverse.

- Se aprecian bucles infinitos en jugadas, ya sea a un movimiento, dos o dentro de una secuencia. En algunos casos pueden ser atribuidas al Chaser por no intentar acercarse al Evader

IV.2.3. Ronda de complemento de entrenamiento

En la última ronda de entrenamiento se partió de la realización del segundo entrenamiento dos veces, con la intención de incrementar la experiencia de los agentes. Luego, con las Q-Tables obtenidas, replicamos el entrenamiento de la primera ronda.

Las variables particulares de este entrenamiento, además de las condiciones generales antes mencionadas, serán:

- Epsilon inicial de 1
- Disminución de Epsilon (Epsilon decay) de 0.98 por jugada.

IV.2.3.1. Resultados Gráficos

A continuación se muestran los resultados de 3 de los 50 bloques de entrenamiento realizados.

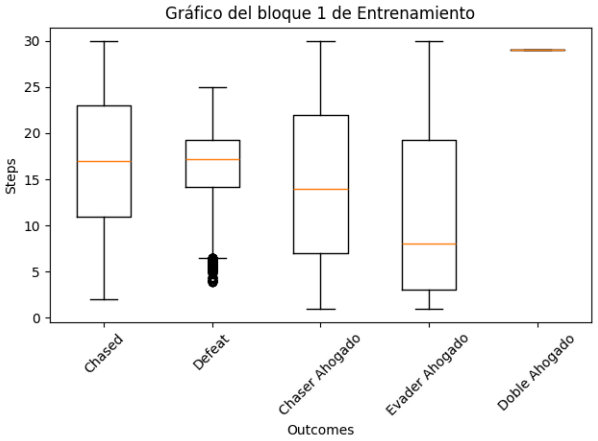

Fig. 44, Gráfico de Bigote bloque 1, 3er Entrenamiento

Durante este bloque de entrenamiento el Epsilon Inicial fue del 100%, al final del bloque y por el resto del entrenamiento quedó en 0%.

Parte con victoria clara para el Evader, logrando evadir su objetivo en el 78,78% de las 147.455 jugadas. También se puede apreciar, que en promedio permanece al 56,6% de la distancia máxima posible con respecto al Chaser.

Contra el 19,62% de evasiones realizadas por parte del Evader., que en promedio logró realizar la captura en el paso número 17.

En cuanto al ahogamiento de los agentes, en esta oportunidad fueron valores bajos, el Chaser se ahogó en el 1,35% de las veces, bastante despegado está el Evader con 0,25% de las veces. También solo en 1 oportunidad los dos agentes se ahogan a la vez.

Durante los siguientes 24 bloques de entrenamiento existe dualidad en las victorias reduciendo ventajas y luego comenzando a ganar. También se aprecia una reducción en la cantidad de jugadas que los dos agentes terminan ahogándose, para ya partiendo de valores cercanos o menores al 1%.

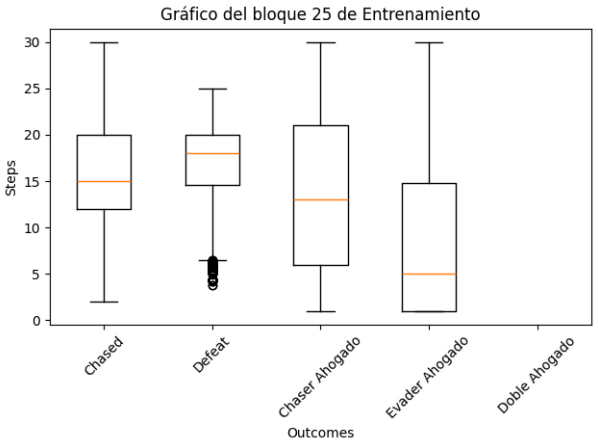

Fig. 45, Gráfico de Bigote bloque 25, 3er Entrenamiento

Resultado a mitad de entrenamiento con valores de Epsilon 0%.

En este bloque refleja una victoria para el Evader, logrando escapar de su rival el 80,24% de las 147.456 jugadas, logrando también aumentar la distancia máxima con respecto al Chaser de 56,6% a 60%.

Contra el 18,66% de capturas realizadas por parte del Chaser, que logró reducir el promedio de capturas en el paso 15.

Con respecto al ahogamiento de los agentes, los dos lograron reducir las jugadas que terminan ahogados, el Chaser se ahogó en el 0,95% de las veces, mientras que el Evader con 0,15% de las veces.

Desde acá hasta el último bloque, los resultados de victorias van oscilando entre Chaser - Evader. Puedes consultar en la gráfica total mostrada más adelante o en el archivo de entrenamiento.

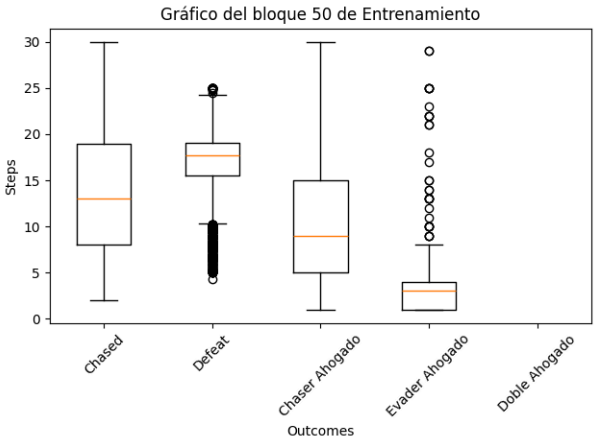

Fig. 46, Gráfico de Bigote bloque 50, 3er Entrenamiento

Para cierre del entrenamiento la victoria fue para el Chaser, logrando capturar de su rival el 82,47% de las 147.456 jugadas, bajando el promedio de captura al paso 14.

Contra el 16,55% de evasiones realizadas por parte del Evader, que bajó su distancia máxima promedio a un 56,6%.

Con respecto al ahogamiento de los agentes, volvieron a reducir las jugadas que terminan ahogados manteniéndose por debajo del 1%, el Chaser se ahogó en el 0,88% de las veces, mientras que el Evader con 0,11% de las veces.

Al resumir y graficar todos los porcentajes de acierto obtenidos en cada una de los 50 bloques del entrenamiento, se obtuvo la siguiente gráfica.

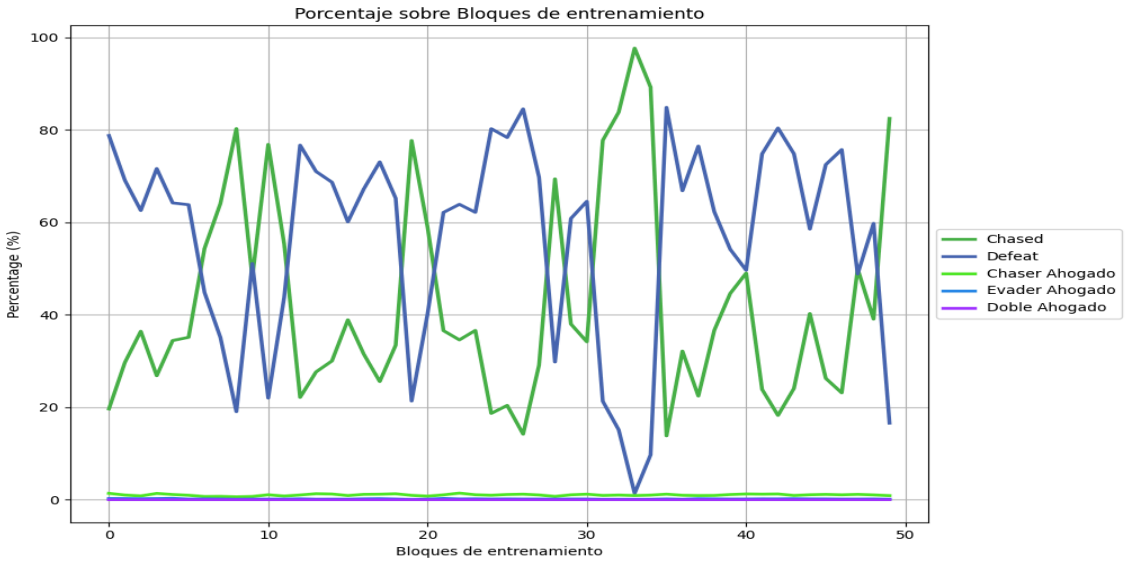

Fig. 47, Gráfico de Línea 3er Entrenamiento

En esta gráfica se observan, valores de ahogamiento muy bajos durante todo el entrenamiento y el comportamiento de los agentes es variable. A pesar de partir con valores altos de captura-evasión, se observa una relación inversa entre los resultados de los agentes, quedando pendiente una validación de razones de este comportamiento.

IV.2.3.2. Resultado en Entorno de Juego

Al igual que en las rondas anteriores, se puso a prueba funcional el modelo en el entorno principal de juego.

Dentro de las jugadas ejecutadas, podemos destacar:

- Se cumple el objetivo de Persecución-Evasión.
- Interacción correcta dentro del entorno, moviéndose correctamente y respetando el funcionamiento de los obstáculos.
- Se usa en menor medida los saltos como movimiento.
- Se aprecian bucles infinitos en jugadas, ya sea a un movimiento, dos o dentro de una secuencia. En algunos casos pueden ser atribuidas al Chaser por no intentar acercarse al Evader

V. Discusión

A continuación se avanzó con la discusión de relacionada a la implementación y evaluación del proyecto

V.1. Análisis por Rondas de Entrenamiento

V.1.1. Primera ronda de entrenamiento

Dentro de esta prueba se observó el impacto que tiene la variación de Epsilon durante el entrenamiento. Como se minimizó la aleatoriedad de movimientos de los agentes muy pronto en el entrenamiento, se generó la intención en los agentes a aprendan a moverse dentro de las reglas dadas sin explorar, basándose solo en experiencias conocidas.

Si bien, el agente va enfocado y comienza a avanzar en función de su objetivo particular, deja de lado la posibilidad de aprender del entorno cercano.

Esto se ve empeorado, debido al impacto que tiene la relación del estado con el movimiento que realiza el oponente.

También pudimos confirmar la poca eficiencia de este entrenamiento en la obtención de recompensas, ya que, a pesar de que los saltos eran más costosos. En cuanto a la recompensa, como parte de cero la Q-table, el agente va probando todos las acciones desde el estado inicial, llevándonos a tener muchos movimientos con saltos no por eficientes, si no por no haber sido usados todavía.

Como cierre del análisis se confirmó que el resultado general, en cuestión de cumplimiento de objetivos, es el esperable. Siendo que, durante el entrenamiento el agente más victorioso al inicio y por los primeros bloques fue el Evader.

Esto debido a que, mientras los agentes no conozcan el entorno ni tengan referencias en Q-Tables de lo que tienen que hacer, los movimientos serán proactivamente al azar, haciendo que el Chaser no cumpla su objetivo hasta tanto no haya desarrollado una estrategia de persecución. Una vez el Chaser empiece a obtener experiencia de victoria lo va a ir acoplando más a su estrategia y empiezan a ir rotando las victorias.

V.1.2. Segunda ronda de entrenamiento

El resultado de esta prueba fue el esperado, debido a la aleatoriedad de movimientos generada, obtuvimos agentes que se desenvuelven mucho mejor.

Continuando con la revisión en el entorno principal de juego, pudimos observar que los agentes generan menos saltos innecesarios y caminan mucho más en los lugares correspondientes. No se eliminó el salto totalmente, pero mejoró en cuanto a la ronda anterior.

Como punto negativo, luego de esta ronda, no se observó que estuvieran siguiendo el objetivo principal y en el entorno principal de juego no hay intención de persecución-evasión. Además de esto, y como resultado sorpresa, observamos que el Chaser estuvo dominando, en cuanto a victorias, todo el entrenamiento, cuando lo esperado era lo contrario.

V.1.3. Ronda de complemento de entrenamiento

En esta ronda se sumaron horas de entrenamiento y se replicó el mismo esquema del primer entrenamiento, pero sin partir con las Q-tables en cero.

Como era esperado, obtuvimos mejoría en el desenvolvimiento de los agentes, logrando reducir el uso de saltos, solo a cuando es necesario. Además de esto, la asertividad de los agentes estuvo oscilando en torno al 50% esperado, y produciendo dualidad entre los ganadores. Dentro del entorno principal de juego, existe la intención de persecución-evasión y genera un entorno de juego funcional.

V.2. Resumen de los entrenamientos

Como cierre de los entrenamientos, se da como satisfactorio los resultados obtenidos, teniendo en cuenta que se deben aplicar correcciones para poder evolucionar el proyecto.

Debemos considerar el impacto de Epsilon para los entrenamientos, ya que esto puede ser muy útil al momento de focalizar alguna estrategía.

Identificamos que, el análisis de las jugadas registradas es vital para identificar errores que pueden llevar a conclusiones imprecisas, además de mostrarnos las interacciones permitiéndonos crear soluciones.

VI. Conclusiones y Limitaciones

VI.1. Conclusiones

Con la finalización de este proyecto se ha logrado implementar agentes inteligentes basados en aprendizaje por refuerzo (Reinforcement Learning - RL). Dentro de los logros, se pudo integrar dentro de un videojuego creado totalmente en el lenguaje de programación Python, permitiéndonos a su vez interactuar como usuarios con dichos agentes.

La obtención de recursos gráficos a pesar de no ser totalmente satisfactoria, generan una buena base para ser editadas manualmente. Si bien obtuvimos buenos avances con este objetivo, no se implementó en el prototipo final. Esto debido a que no fue posible la obtención de todas las imágenes coherentes, en cuanto a tamaños, secuencias de movimientos funcionales y obstáculos funcionales, dentro de los tiempos establecidos para este proyecto.

Dentro de este mismo apartado, pudimos confirmar que sí existen opciones de pago en el mercado, como por ejemplo Scenario o Upscale Media, para esta obtención de estos recursos. En estas se pueden enviar videos con la secuencia de movimiento y personaje, para la creación de la secuencia de movimiento.

Por otro lado, la IA no solo nos ayudó a entender procesos básicos para la creación de un videojuego, si no que fue clave. Aunque, la IA no pudo generar el código completo, colaboró en la obtención de bloques funcionales de código adaptables al desarrollo de la aplicación, siendo soporte fundamental para poder completar el trabajo. Logrando generar bloques y funciones limitadas a operaciones, que luego pudieron corregidas y fácilmente integradas dentro del proyecto.

Se logró implementar y configurar agentes inteligentes basados en algoritmos de RL con todas sus funciones, sistema de recompensas y un protocolo de entrenamiento que demuestra el aprendizaje de los agentes.

Dentro de los entregables realizados destacan, un entorno prototipo de videojuego con tres posibilidades de usos. La primera, entorno de juego totalmente funcional.

Segundo, un entorno de pruebas solo para interacción entre usuarios y obstáculos. Tercero, un entorno de entrenamiento guiado para corrección de errores o refuerzos de entrenamiento.

Finalmente, un entorno 100% dedicado a entrenamientos aislados entre agentes.

VI.2. Limitaciones y Problemáticas

Aunque el desarrollo del TFM fue en general fluido, la investigación logró ofrecer un producto funcional. De esta manera consideramos que es crucial reconocer sus limitaciones y dar a conocer los desafíos enfrentados.

VI.2.1. Limitaciones

La principal limitación es que el juego no es un entregable completo, falta la realización de interfaz que sirva de amalgama general de juego y evite tener que acceder al código para poder realizar ejecuciones, revisiones o modificaciones. Además, queda pendiente la corrección de los bucles infinitos presentes con la ausencia de interacción humana.

VI.2.2. Problemáticas Generales

La principal problemática, fue el tiempo que toma desarrollar el entorno, si bien estaba dentro de las consideraciones de complicidad iniciales, fue bastante problemático el poder conseguir los recursos solo usando IA gratuita, si bien podemos obtener fácilmente recursos de imágenes, estos no eran los suficientemente precisos para no generar incongruencias en los movimientos de los personajes.

Otra de las dificultades presente son atribuibles a la programación y aplicación de la IA. Al implementar IA, no abordamos el cómo detectar y responder antes los bucles infinitos en los que entran los agentes.

Si bien, son interrumpidos por la finalización de la partida, el hecho de que al eliminar aleatoriedad en los movimientos ocasiona que existan pocas estrategias, y al existir bucles en algunas jugadas, tenemos jugadas en entrenamientos perdidas.

Además de esto, otra de las problemáticas a mejorar es la priorización y la evaluación continua de las políticas de recompensa. Estas tienen un gran impacto en cuanto a la interacción con el entorno y objetivo principal, como por ejemplo, los bordes no tienen ningún tipo de valoración de recompensa, lo que ocasiona que en esos momentos los agentes reaccionen sin alguna lógica en particular.

Finalmente, los recursos computacionales tanto para almacenamiento como para procesamiento de datos fue el desafío más significativo. La creación del entorno, grabado de vídeos de muestras, almacenamiento de datos en tablas análisis de movimientos y los códigos para de prueba utilizados, colapsaron en varias oportunidades el ordenador. Se tuvo que recurrir a herramientas externas,

como Google Colab, que a pesar de ser excelente ayuda durante el proyecto, debido a cortes realizados por el proveedor, de igual forma ocasionó horas de entrenamiento pérdidas.

VI.3. Futuras líneas de investigación

Como punto de partida para continuar y como posibles proyectos de programación. Lograr implementar las correcciones de todas las recomendaciones dadas en este trabajo, de entre las cuales podemos destacar. En primer lugar, la implementación de algoritmos de visión por computadora para detección de acciones y estados. Segundo lugar, el desarrollo de una interfaz de control para no que permita tener un producto de juego terminado. Y por último, la estandarización de movimientos para lograr que los agentes detecten el seguimiento de acciones futuras dentro de las recompensas. Esto con la intención de garantizar la evolución regular del software generado, permitiéndonos tener un software más completo y de ser posible funcional en alguna plataforma de juegos.

Dentro de la rama de IA, la principal línea de investigación a realizar, es la de implementar algoritmos de Deep Q-Network (DQN) o RL profundo. Como evolución directa al algoritmo de este proyecto tenemos la implementación de agentes basados en modelos de DQN, con el fin de mejorar la capacidad de aprendizaje y la complejidad estratégica de los agentes. Esto nos permitirá aumentar la cantidad de estados en el juego o agregar nuevas funcionalidades que lo complejizan aún más.

Referencias bibliográficas

Borra, F., Biferale, L., Cencini, M., & Celani, A. (2022). Reinforcement learning for pursuit and evasion of microswimmers at low Reynolds number. *Physical Review Fluids*, *7*(2), 023103. https://doi.org/10.1103/PhysRevFluids.7.023103

Bouzy, B., & Métivier, M. (2007). Multi-agent model-based reinforcement learning experiments in the pursuit evasion game. *Artificial Intelligence*, *171*, 365-377.

Chen, Y., Shi, Y., Dai, X., Meng, Q., & Yu, T. (2025). Pursuit-evasion game with online planning using deep reinforcement learning. *Applied Intelligence*, *55*(7), 512. https://doi.org/10.1007/s10489-025-06396-3

Descottes, J. (2017). *Piskel* [Software]. https://www.piskelapp.com/

Dúo Terrón, P., Moreno Guerrero, A. J., López Belmonte, J., & Marín Marín, J. A. (2023). Inteligencia Artificial y Machine Learning como recurso educativo desde la perspectiva de docentes en distintas etapas educativas no universitarias. *Revista Interuniversitaria de Investigación en Tecnología Educativa*, 58-78. https://doi.org/10.6018/riite.579611

Gao, M., Yan, T., Li, Q., Fu, W., & Zhang, J. (2023). Intelligent Pursuit–Evasion Game Based on Deep Reinforcement Learning for Hypersonic Vehicles. *Aerospace*, *10*(1), 86. https://doi.org/10.3390/aerospace10010086

Gonultas, B. M., & Isler, V. (2025). *Pursuit-Evasion for Car-like Robots with Sensor Constraints* (No. arXiv:2405.05372). arXiv. https://doi.org/10.48550/arXiv.2405.05372

Huang, X. (2021). *Adversary agent reinforcement learning for pursuit-evasion* (Versión 1). arXiv. https://doi.org/10.48550/ARXIV.2108.11010

Kotsiantis, S. B., Zaharakis, I. D., & Pintelas, P. E. (2006). Machine learning: A review of classification and combining techniques. *Artificial Intelligence Review*, *26*(3), 159-190. https://doi.org/10.1007/s10462-007-9052-3

Lin, Y., Gao, H., & Xia, Y. (2025). Distributed Pursuit–Evasion Game Decision-Making Based on Multi-Agent Deep Reinforcement Learning. *Electronics*, *14*(11), 2141.

https://doi.org/10.3390/electronics14112141

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Pygame. (s. f.). [Software]. https://www.pygame.org

Pygame Software Foundation. (2024). *PyGame* (Versión 2.6.1) [Software]. https://www.pygame.org/

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484-489. https://doi.org/10.1038/nature16961

Singh, G., Lofaro, D., & Sofge, D. (2020). Pursuit-evasion with Decentralized Robotic Swarm in Continuous State Space and Action Space via Deep Reinforcement Learning: *Proceedings of the 12th International Conference on Agents and Artificial Intelligence*, 226-233. https://doi.org/10.5220/0008971502260233

Souza, C. de, Newbury, R., Cosgun, A., Castillo, P., Vidolov, B., & Kulic, D. (2020). *Decentralized Multi-Agent Pursuit using Deep Reinforcement Learning*. https://doi.org/10.48550/ARXIV.2010.08193

Sutton, R. S., & Barto, A. (2014). *Reinforcement learning: An introduction* (Nachdruck). The MIT Press.

Wang, Q., Li, X., Yuan, Z., Yang, Y., Xu, C., & Zhang, L. (2022). *An Opponent-Aware Reinforcement Learning Method for Team-to-Team Multi-Vehicle Pursuit via Maximizing Mutual Information Indicator* (Versión 1). arXiv. https://doi.org/10.48550/ARXIV.2210.13015

Wang, Y., Cao, Y., Chiun, J., Koley, S., Pham, M., & Sartoretti, G. A. (2024). Viper: Visibility-based pursuit-evasion via reinforcement learning. *In 8th Annual Conference on Robot Learning.* https://openreview.net/forum?id=EPujQZWemk

World Chase Tag®. (2025). https://wct.webflow.io/

Wouters, T. (2025). *Python* (Versión 3.12.10) [Software]. https://www.python.org/

Wu, Z., Ye, S., Natarajan, M., & Gombolay, M. C. (2024). *Diffusion-Reinforcement Learning Hierarchical Motion Planning in Multi-agent Adversarial Games* (Versión 2). arXiv. https://doi.org/10.48550/ARXIV.2403.10794

Xu, S., & Dang, Z. (2025). Emergent behaviors in multiagent pursuit evasion games within a bounded 2D grid world. *Scientific Reports*, *15*(1), 29376. https://doi.org/10.1038/s41598-025-15057-x

Índice de figuras