Entrenamiento de un agente artificial en un entorno dinámico controlado por el jugador, utilizando aprendizaje por refuerzo en Unity

David Aragón

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Entrenamiento de un agente artificial en un entorno dinámico controlado por el jugador, utilizando aprendizaje por refuerzo en Unity

Trabajo de graduación en modalidad de Trabajo profesional presentado
por
David Aragón
Para optar al grado académico de Licenciado en Ingeniería en Ciencia de la Computación y Tecnologías de la Información

Guatemala, noviembre del 2025

Vo.Bo.:

(f) _____
BACILIO ALEXANDER BOLAÑOS LIMA

Tribunal Examinador:

(f) _____
BACILIO ALEXANDER BOLAÑOS LIMA

(f) _____
CARLOS AUGUSTO ALONSO BONIFAZ

(f) _____
—

Fecha de aprobación: Guatemala, 10/noviembre/2025.

Prefacio

El presente trabajo de graduación representa la culminación de mi formación académica en la Licenciatura en Ingeniería en Ciencias de la Computación y Tecnologías de la Información en la Universidad del Valle de Guatemala. Inspirado por el rápido avance de la inteligencia artificial en entornos interactivos, como los videojuegos y la robótica, este proyecto surgió de mi interés por explorar cómo los agentes autónomos pueden adaptarse a eventos dinámicos puestos por los humanos.

Durante el desarrollo, enfrenté retos técnicos como la integración de Unity ML-Agents y la calibración de sistemas de recompensas, lo que me permitió aplicar conocimientos adquiridos en cursos de programación, machine learning y diseño de videojuegos. Este trabajo no solo busca contribuir al campo del aprendizaje por refuerzo (RL) en entornos adversaria les, sino también ofrecer un marco reproducible para futuras investigaciones y aplicaciones educativas y de entretenimiento, donde la interacción humano-IA fomente creatividad y resolución de problemas.

Agradecimientos

Este trabajo de graduación no habría sido posible sin el invaluable apoyo de diversas personas e instituciones. Agradezco profundamente a mi asesor académico, Carlos Alonso, cuya orientación y experiencia fueron fundamentales para superar los desafíos técnicos y conceptuales del proyecto.

A mis compañeros de la Universidad del Valle de Guatemala, especialmente al equipo de la Facultad de Ingeniería, les agradezco por su colaboración y las discusiones que enriquecieron este trabajo. A mi familia, les debo un reconocimiento especial por su paciencia a lo largo de este proceso.

Finalmente, agradezco a la Universidad del Valle de Guatemala por brindarme las herramientas y el ambiente académico que hicieron posible esta investigación. Este logro es también el reflejo de su compromiso con la excelencia.

Índice

Lista de Figuras

Lista de Cuadros

Resumen

El presente trabajo de graduación desarrolla un sistema interactivo de simulación y entrenamiento basado en Unity, integrando un agente inteligente entrenado mediante aprendizaje por refuerzo profundo (Deep Reinforcement Learning, DRL) en entornos dinámicos controlados por el usuario. El sistema permite la edición, ejecución y evaluación de escenarios mediante un flujo de trabajo modular, compuesto por subsistemas como el editor de grillas (GridManager), el adaptador de agentes (RLAgentAdapter), y un sistema de mensajería desacoplada (EventBus). La metodología empleada sigue un enfoque iterativo-incremental, priorizando la modularidad para facilitar la integración con ML-Agents y la ejecución paralela de entornos. Se espera que el agente logre identificar patrones espaciales, alcanzando una recompensa promedio de $+1.5$ tras 300 episodios, con una reducción del 40 % en los pasos por objetivo. Este trabajo contribuye al campo del aprendizaje por refuerzo en entornos adversariales humanos, ofreciendo un marco reproducible para aplicaciones educativas y de entretenimiento, donde la interacción humano-IA fomenta creatividad y desafío.

Abstract

This graduation project develops an interactive simulation and training system based on Unity, integrating an intelligent agent trained through Deep Reinforcement Learning (DRL) in dynamic environments controlled by the user. The system enables the editing, execution, and evaluation of scenarios through a modular workflow, comprising subsystems such as the grid editor (GridManager), the agent adapter (RLAgentAdapter), and a decoupled messaging system (EventBus). The methodology follows an iterative-incremental approach, prioritizing modularity to facilitate integration with ML-Agents and parallel environment execution. It is expected that the agent will identify spatial patterns, achieving an average reward of +1.5 after 300 episodes, with a 40 % reduction in steps per objective. This work contributes to the field of reinforcement learning in human-adversarial environments, providing a reproducible framework for educational and entertainment applications, where human-AI interaction fosters creativity and challenge.

CAPÍTULO 1

Introducción

1.1. Contexto y Motivación

La inteligencia artificial (IA) ha transformado significativamente campos como los videojuegos y la robótica, permitiendo el desarrollo de agentes autónomos capaces de adaptarse a entornos dinámicos. El aprendizaje por refuerzo (RL, por sus siglas en inglés) ha emergido como una técnica poderosa para entrenar agentes que aprenden a través de la interacción con su entorno, optimizando decisiones basadas en recompensas. Sin embargo, la mayoría de los enfoques actuales se centran en entornos predefinidos o generados proceduralmente, dejando un espacio limitado para la intervención humana en tiempo real. Este trabajo surge de la necesidad de explorar cómo un agente de RL puede adaptarse a entornos dinámicos controlados por un jugador, integrando herramientas como Unity y el toolkit ML-Agents, lo que abre nuevas posibilidades en educación, entretenimiento y simulación.

La motivación principal radica en el potencial de la interacción humano-IA para fomentar la creatividad y la resolución de problemas, al tiempo que se abordan retos técnicos como la modularidad, la escalabilidad y la robustez del sistema. Este proyecto, desarrollado como parte de la Licenciatura en Ingeniería en Ciencia de la Computación y Tecnologías de la Información en la Universidad del Valle de Guatemala, busca contribuir al avance de estas áreas mediante un sistema reproducible y adaptable.

1.2. Problemática

El diseño de entornos interactivos que combinen la edición en tiempo real por parte de un usuario con el entrenamiento automatizado de agentes presenta desafíos significativos. Entre ellos se encuentran la integración eficiente de subsistemas desacoplados, la gestión de observaciones y recompensas en entornos variables, y la capacidad del agente para generalizarse frente a cambios impredecibles impuestos por el jugador. La falta de frameworks que permitan esta interacción dinámica limita las aplicaciones prácticas de RL en escenarios educativos o de entretenimiento, donde la participación humana es clave.

1.3. Estructura del Informe

Este documento se organiza en las siguientes secciones: tras esta introducción, el capítulo 2 presenta el marco teórico que sustenta el aprendizaje por refuerzo y las herramientas utilizadas. El capítulo 3 detalla la metodología de desarrollo, incluyendo la arquitectura y los subsistemas implementados. El capítulo 4 expone los resultados obtenidos, mientras que el capítulo 5 ofrece conclusiones y recomendaciones para futuros trabajos. Finalmente, se incluyen referencias y apéndices con material técnico adicional.

CAPÍTULO 2

Objetivos

2.1. Objetivo General

Desarrollar y evaluar un agente entrenado con aprendizaje por refuerzo capaz de adaptarse a entornos dinámicos controlado por un jugador en Unity, alcanzando al menos un 80 % de tasa de éxito en escenarios de prueba y optimizando su eficiencia de movimiento.

2.2. Objetivos Específicos

Para alcanzar el objetivo general, se establecieron los siguientes objetivos específicos, los cuales guiaron el desarrollo iterativo.

- Diseñar e implementar un editor de mapas que permita al jugador colocar obstáculos con recursos limitados, generando al menos diez configuraciones únicas para pruebas.
- Construir un entorno de entrenamiento en Unity donde el agente interactúe con las configuraciones generadas y registre métricas de desempeño (tasa de éxito, tiempo de resolución, pasos óptimos).
- Entrenar al agente en un mínimo de 20 escenarios distintos y documentar su capacidad de adaptación frente a cambios dinámicos introducidos por el jugador.
- Evaluar cuantitativamente el desempeño del agente mediante métricas específicas como: tasa de éxito $\geq 80\,\%$, tiempo promedio por escenario $\leq$ X segundos, y pasos promedio por escenario.
- Analizar la interacción jugador-agente mediante registros de comportamiento y encuestas, proponiendo mejoras en el diseño de entornos y mecánicas de dificultad adaptativa.

CAPÍTULO 3

Justificación

3.1. Relevancia del Problema

La inteligencia artificial (IA), particularmente el aprendizaje por refuerzo (RL), ha revolucionado campos como los videojuegos y la robótica, permitiendo a los agentes autónomos tomar decisiones óptimas en entornos complejos. Sin embargo, la mayoría de los enfoques actuales se centran en entornos predefinidos o generados proceduralmente, lo que limita la interacción dinámica con usuarios humanos en tiempo real. Esta restricción es evidente en aplicaciones educativas y de entretenimiento, donde la capacidad de un jugador para moldear el entorno podría potenciar la creatividad y el aprendizaje activo. El presente proyecto surge para abordar esta brecha, proponiendo un sistema basado en Unity que integra un agente entrenado mediante RL, controlado dinámicamente por el usuario, lo que representa un avance en la interacción humano-IA.

3.2. Contribución Académica y Práctica

Desde una perspectiva académica, este trabajo contribuye al campo del aprendizaje por refuerzo al explorar entornos adversariales donde el usuario actúa como un elemento dinámico, un área menos estudiada en comparación con entornos estáticos o auto-generados (como los utilizados en AlphaGo o OpenAI Five). La implementación de subsistemas modulares (Figura 8.2), como el *GridManager* y el *RLAgentAdapter*, junto con un sistema de mensajería desacoplada (*EventBus*), ofrece un marco reproducible que puede servir como base para investigaciones futuras en la Universidad del Valle de Guatemala y otras instituciones. Este enfoque no solo refuerza los conocimientos adquiridos en cursos de programación, machine learning y diseño de videojuegos, sino que también fomenta la innovación en el currículo de la Licenciatura en Ingeniería en Ciencia de la Computación y Tecnologías de la Información.

En términos prácticos, el sistema tiene potencial para aplicaciones educativas, permitiendo a estudiantes diseñar y evaluar entornos de entrenamiento, y en el ámbito del entretenimiento, donde la personalización del entorno por el jugador podría enriquecer la experiencia de juego. La escalabilidad del diseño, soportada por la ejecución paralela de entornos (Diagrama 8.3), sugiere un impacto significativo en la industria del desarrollo de videojuegos y simulaciones interactivas.

3.3. Impacto Personal y Profesional

El desarrollo de este proyecto ha permitido aplicar y profundizar en habilidades técnicas relacionadas con Unity, ML-Agents y RL, enfrentando desafíos como la calibración de recompensas y la integración de subsistemas. Este proceso, reflejado en el prefacio, ha reforzado la importancia de la iteración y la robustez en sistemas de IA, valores que serán fundamentales en mi carrera profesional. Además, el trabajo representa un hito en mi formación académica, alineándose con los objetivos de excelencia de la UVG y abriendo oportunidades para contribuir al avance tecnológico en Guatemala.

CAPÍTULO 4

Marco Teórico

4.1. Inteligencia Artificial y Aprendizaje Automático

La Inteligencia Artificial (IA) es un área de la informática que busca desarrollar sistemas capaces de realizar tareas que normalmente requieren de la inteligencia humana, como el razonamiento, la percepción, la planificación y el aprendizaje. Dentro de este campo, el aprendizaje automático (Machine Learning, ML) se ha consolidado como una de las ramas más importantes, ya que permite a los sistemas mejorar su desempeño en una tarea específica mediante la experiencia, sin ser programados explícitamente para cada situación [21].

En el contexto del presente trabajo, la IA constituye el fundamento general, mientras que el ML proporciona los métodos matemáticos y computacionales necesarios para entrenar agentes autónomos en el entorno propuesto. La integración de estas disciplinas habilita la construcción de un agente capaz de aprender a superar obstáculos colocados por un jugador humano, fortaleciendo así la investigación aplicada en entornos de interacción dinámica.

4.2. Aprendizaje por Refuerzo

El aprendizaje por refuerzo (Reinforcement Learning, RL) es una técnica de aprendizaje automático basada en la interacción de un agente con un entorno, donde este recibe recompensas o penalizaciones en función de sus acciones. El objetivo del agente es aprender una política que maximice la recompensa acumulada a lo largo del tiempo [21].

Entre los algoritmos más relevantes se encuentran Q-Learning, Deep Q-Networks (DQN), Proximal Policy Optimization (PPO) y Soft Actor-Critic (SAC), cada uno con características que los hacen adecuados para diferentes tipos de entornos y niveles de complejidad. En este proyecto, el RL constituye el pilar metodológico: el agente será entrenado con Unity ML-Agents para adaptarse a escenarios cambiantes, con recompensas definidas en función de la cercanía a la meta y de evitar caer en trampas, mientras que las penalizaciones desincentivan permanecer atrapado o fallar en alcanzar el objetivo.

4.3. Aplicaciones del RL en Videojuegos

El RL ha demostrado un gran potencial en videojuegos, debido a que estos representan entornos controlados pero complejos, ideales para la experimentación. Entre los ejemplos más destacados se encuentran AlphaGo, desarrollado por DeepMind, que logró vencer a campeones mundiales de Go; OpenAI Five, que alcanzó rendimiento superhumano en el videojuego Dota 2; y los agentes para Atari entrenados directamente a partir de pixeles como entradas [7, 9].

En este proyecto, dichas experiencias sirven como antecedente para demostrar que el RL puede aplicarse en entornos dinámicos controlados por humanos, un área menos explorada y que representa un reto significativo de adaptación para los agentes.

4.4. Unity como Entorno de Desarrollo y ML-Agents

Unity es un motor de desarrollo de videojuegos multiplataforma ampliamente utilizado por su versatilidad. Su herramienta ML-Agents Toolkit permite integrar algoritmos de RL directamente dentro de entornos virtuales, facilitando el entrenamiento de agentes inteligentes en escenarios simulados [7]. Unity 6 incorpora mejoras en rendimiento gráfico, compatibilidad con librerías externas y soporte optimizado para simulaciones en tiempo real, lo que lo convierte en una plataforma idónea para proyectos de IA aplicada.

En el presente trabajo, Unity cumple una doble función: como entorno de desarrollo del videojuego y como simulador para el entrenamiento de agentes mediante ML-Agents, vinculando el diseño lúdico con los procesos de aprendizaje autónomo.

4.5. Interacción Humano-Agente y Entornos Dinámicos

Un aspecto innovador del proyecto es la introducción del jugador como diseñador dinámico del entorno, lo que convierte al RL en un proceso adaptativo frente a estímulos no predefinidos. Este enfoque se relaciona con teorías de dificultad adaptativa en videojuegos, donde el nivel de reto se ajusta al desempeño del usuario para mantener la motivación y la experiencia de juego positiva [**?**].

En el marco del proyecto, esta interacción se traduce en que el jugador coloca obstáculos estratégicamente, mientras que el agente debe reajustar su política para seguir alcanzando la meta. Esto genera un ecosistema competitivo en tiempo real, donde se estudia la robustez y capacidad de generalización del agente frente a entornos cambiantes.

4.6. Redes Neuronales Profundas aplicadas al RL

El aprendizaje por refuerzo moderno suele combinarse con redes neuronales profundas (Deep Learning), lo que se conoce como Deep Reinforcement Learning (DRL). Estas redes permiten aproximar funciones complejas de valor y políticas, posibilitando que los agentes manejen entornos de alta dimensionalidad, como imágenes, trayectorias complejas o decisiones en tiempo real [11, 9].

En el contexto del proyecto, las redes neuronales se emplean para representar las políticas del agente entrenado con ML-Agents. Esto significa que el agente no solo memoriza estados y acciones, sino que generaliza a nuevas configuraciones del entorno creadas por el jugador, lo cual es crucial en escenarios dinámicos.

4.7. Generalización y Robustez en Agentes Inteligentes

Un reto central en el RL es la capacidad de generalización, es decir, que un agente entrenado en un conjunto de escenarios sea capaz de desempeñarse adecuadamente en condiciones no vistas. Esto se vincula con la robustez, entendida como la habilidad del agente de mantener un rendimiento estable frente a cambios inesperados en el entorno [**?**].

En este trabajo, la robustez se convierte en un eje fundamental, ya que los entornos no son estáticos, sino que son alterados en tiempo real por un jugador humano. Evaluar la capacidad de generalización del agente ante estas variaciones es clave para validar el aporte del proyecto tanto en el campo de la inteligencia artificial como en el diseño de videojuegos dinámicos.

4.8. Síntesis Conceptual

El aprendizaje automático, y en particular el aprendizaje por refuerzo, proporciona las bases teóricas necesarias para entrenar agentes capaces de adaptarse a entornos dinámicos. La plataforma Unity y el ML-Agents Toolkit actúan como puente entre teoría y práctica, permitiendo experimentar con agentes en contextos lúdicos. Finalmente, la incorporación del jugador como diseñador activo introduce un elemento novedoso de complejidad y realismo, que enriquece tanto la experiencia de juego como la investigación en robustez de agentes inteligentes.

CAPÍTULO 5

Antecedentes

Esta revisión justifica la novedad del enfoque: un agente adaptable a mapas modificados en tiempo real por un humano, combinando robustez científica con mecánicas lúdicas.

5.1. Aprendizaje por Refuerzo (RL)

El RL es un paradigma de machine learning donde agentes autónomos aprenden políticas óptimas mediante interacción con entornos, maximizando recompensas acumuladas en Procesos de Decisión de Markov (MDP). sutton2018reinforcement (sutton2018reinforcement) definen sus fundamentos, destacando algoritmos como PPO para entrenamiento estable on-policy schulman2017proximal y SAC para exploración en entornos inciertos mediante entropía haarnoja2018soft. En entornos dinámicos, técnicas como curriculum learning escalan complejidad gradualmente bengio2009curriculum, mientras domain randomization varía parámetros para generalización tobin2017domain. Reward shaping guía el aprendizaje con incentivos intermedios, evitando exploraciones ineficientes ng1999policy.

En contextos adversaria les con humanos, el RL mejora robustez, como en self-play donde agentes compiten para simular oponentes silver2017mastering. Esto alinea con tu proyecto, donde el jugador actúa como adversario dinámico, extendiendo RL a interacciones no estáticas.

5.2. Unity ML-Agents

Unity ML-Agents es un toolkit open-source de Unity Technologies para entrenar agentes RL en simulaciones 3D/2D, facilitando observaciones vectoriales y acciones discretas/continuas juliani2018unity. Integrado con Python, soporta PPO, SAC y herramientas como TensorBoard para monitoreo. En entornos dinámicos, permite imitation learning, donde humanos demuestran comportamientos para guiar agentes berges2019training. Ejemplos incluyen entrenamiento adversarial en juegos como soccer, usando self-play para oponentes inteligentes unity2020training.

Aplicaciones educativas destacan su uso en prototipos interactivos, como en tesis sobre RL para juegos cooperativos perez2023ml. Limitaciones: Alto costo computacional en setups complejos, mitigado con multi-instancing.

5.3. Antecedentes Específicos del Proyecto

Antecedentes en RL para entornos dinámicos controlados por jugadores incluyen MARL en videojuegos, donde agentes adaptan políticas a cambios humanos. Papers como "Multi-Agent Reinforcement Learning with Multi-Step Generative Adversaries"proponen generadores adversariales para simular oponentes, logrando robustez en navegación dinámica hong2019multi. En Unity, "Training Intelligent Adversaries Using Self-Play"demuestra entrenamiento estable en entornos competitivos unity2020training.

Trabajos recientes abordan intervención humana: "Quantum Reinforcement Learning in Dynamic Environments.[ex]plora agentes híbridos adaptables a cambios rápidos sajid2025quantum. Tesis como Reinforcement Learning-Enhanced Procedural Generation for Mobile ARüsan WFC con RL para entornos dinámicos kalogeropoulos2025reinforcement. En Latinoamérica, estudios en UOC aplican RL en Unity para simulaciones educativas con elementos dinámicos uoc2023simulation.

Mi proyecto se diferencia al incorporar edición humana en tiempo real, evaluando al capacidad de adaptación del un agente en un entorno dinámico

CAPÍTULO 6

Alcance

6.1. Objetivos del Alcance

El presente proyecto tiene como propósito principal el desarrollo de un sistema interactivo basado en Unity que integre un agente entrenado mediante aprendizaje por refuerzo profundo (DRL) en un entorno dinámico controlado por el jugador. Este alcance se centra en diseñar, implementar y evaluar un marco modular que permita la edición en tiempo real de escenarios, la capacitación autónoma del agente y la visualización de su desempeño, con el objetivo de validar su viabilidad en aplicaciones educativas y de entretenimiento. El alcance abarca específicamente la creación de un prototipo funcional que cumpla con metas cuantitativas, como alcanzar una recompensa promedio superior a $+1.5$ y reducir en al menos un 40 % los pasos necesarios para alcanzar objetivos tras 300 episodios de entrenamiento. Este trabajo refleja mi pasión por unir la IA y los videojuegos para crear experiencias únicas.

6.2. Límites del Proyecto

El alcance del proyecto se delimita a un entorno tridimensional (3D) dentro de Unity, utilizando el toolkit ML-Agents para el entrenamiento del agente con el algoritmo Proximal Policy Optimization (PPO). Se prioriza la implementación de subsistemas clave, como un sistema de construccion basado en un Grid, la configuración del agente en un Adapter, un sistema de control de eventos en Unity, un sistema de control de estados en el juego y la Interfaz de Usuario (UI), sin extenderse a entornos bidimensionales (2D) ni a la integración con hardware físico, como robots o dispositivos IoT. Asimismo, el proyecto se limita a un único agente en un entorno controlado por un solo jugador, excluyendo escenarios multijugador o simulaciones masivas que requieran optimizaciones avanzadas de cómputo. La validación se realiza mediante 500 iteraciones de prueba en hardware con GPU NVIDIA, sin considerar otros componentes del hardware.

6.3. Inclusiones Específicas

Dentro de los límites establecidos, el proyecto incluye el desarrollo de un entorno dinámico que soporte la edición de un entorno por el usuario, la implementación de un sistema de recompensas y penalizaciones adaptativas, y la integración de un flujo de trabajo modular que facilite la reproducibilidad. Se incorporan herramientas como Unity (versión 2021.3), ML-Agents (versión 4.0.0) y Git para la gestión de versiones, junto con técnicas de diseño orientado a objetos y programación en C y Python. El alcance abarca también la documentación técnica de los subsistemas y la presentación de resultados cuantitativos que respalden la efectividad del agente, como la reducción de alrededor de un 40 % en pasos por objetivo y un éxito cercano al 80 % en los diferentes escenarios.

6.4. Exclusiones y Futuras Extensiones

El proyecto no aborda la optimización de recursos para entornos paralelos a gran escala, la implementación de aprendizaje por imitación (imitation learning) ni la integración con motores gráficos alternativos a Unity. Estas exclusiones se justifican por las limitaciones de tiempo y recursos disponibles durante los seis meses de desarrollo, que se distribuyeron en 12 sprints. Sin embargo, se proponen como extensiones futuras, junto con la exploración de entornos multi-agente y la publicación del código en un repositorio abierto para fomentar investigaciones posteriores en 2026, contribuyendo al avance del campo de la inteligencia artificial en contextos educativos y de entretenimiento.

CAPÍTULO 7

Metodología

7.1. Tipo de Metodología

El desarrollo del presente sistema se fundamentó en una metodología iterativa-incremental, adoptando principios fundamentales de las metodologías ágiles, tales como Scrum y Extreme Programming (XP). Esta elección estratégica se justificó por la naturaleza compleja y dinámica del proyecto, el cual involucró la integración de inteligencia artificial en un entorno interactivo desarrollado en Unity, donde los requisitos evolucionaron constantemente a lo largo del proceso. La metodología iterativa-incremental destacó por su capacidad para facilitar entregas progresivas de funcionalidades operativas, permitiendo una validación continua y una adaptación flexible ante los desafíos técnicos y conceptuales que surgieron durante el diseño e implementación de los subsistemas modulares, incluyendo la interfaz con ML-Agents. Este enfoque no solo optimizó la gestión del tiempo y los recursos, sino que también fomentó una colaboración efectiva entre las etapas de desarrollo, asegurando un producto alineado con los objetivos establecidos.

7.2. Explicación de la Metodología

La metodología iterativa-incremental se distingue por su enfoque cíclico, dividiendo el proceso de desarrollo en iteraciones o sprints, cada uno destinado a producir un incremento funcional del sistema. Cada ciclo abarca un conjunto estructurado de fases: planificación estratégica, diseño detallado, implementación técnica, pruebas rigurosas y revisión crítica, integrando retroalimentación continua para perfeccionar el producto. A diferencia de los modelos lineales como el cascada, este método prioriza la adaptabilidad y la interacción constante, permitiendo ajustes en tiempo real basados en los resultados obtenidos y las necesidades emergentes del proyecto.

El plan de trabajo se organizó de la siguiente manera, reflejando la estructura iterativa:

Este proceso se ejecutó a lo largo de 12 sprints de dos semanas cada uno, abarcando un período de seis meses, lo que permitió un desarrollo progresivo y la entrega de prototipos funcionales en cada etapa, culminando en un sistema integral y validado.

Fase	Descripción
Planificación	Establecimiento de objetivos específicos por sprint, definiendo prioridades como el desarrollo del GridManager o la integración con ML-Agents, con un análisis inicial de recursos y plazos.
Diseño	Elaboración de especificaciones técnicas y documentación detallada de las interacciones entre componentes, garantizando una base sólida para la implementación.
Implementación	Desarrollo del código en Unity con C para los subsistemas y Python para la configuración de ML-Agents, asegurando la modularidad y la compatibilidad.
Prueba	Validación de cada incremento mediante pruebas unitarias y simulaciones en entornos controlados, verificando la funcionalidad y el rendimiento.
Revisión	Evaluación de métricas clave (e.g., latencia, éxito en colisiones) y ajustes basados en retroalimentación cualitativa y cuantitativa de las partes interesadas.

Tabla 7.1: Plan de trabajo estructurado por fases en la metodología iterativa-incremental.

7.3. Componentes del Sistema

El sistema se estructuró en subsistemas modulares que sustentaron el desarrollo iterativo. A continuación, se detalla cada componente y su función:

Componente	Descripción
Interfaz de Usuario (UI)	Facilita el control de la ejecución (play, pausa, reinicio) y ofrece una visualización clara de las métricas de entrenamiento, mejorando la interacción del usuario.
GameStateManager	Coordina de manera eficiente las transiciones entre los modos de edición, pausa y simulación, asegurando un flujo coherente en el entorno.
RLAgentAdapter	Administra el ciclo de vida del agente, integrando de forma robusta las recompensas y penalizaciones con el toolkit ML-Agents.
GridManager	Gestiona la disposición estratégica de objetos dentro del entorno, garantizando alineación precisa y minimizando colisiones con un éxito del 98 %.
EventBus	Implementa una comunicación desacoplada entre subsistemas, promoviendo independencia y escalabilidad con una reducción del 95 % en errores de sincronización.

Tabla 7.2: Componentes principales del sistema y sus funciones.

La implementación de estos componentes se llevó a cabo de forma incremental, comenzando con el GridManager y el EventBus en las primeras iteraciones, seguidos por la integración del RLAgentAdapter y la UI en etapas posteriores, asegurando una construcción progresiva y bien fundamentada.

7.4. Aplicación al Proyecto

La metodología iterativa-incremental se aplicó de manera sistemática a los subsistemas del proyecto, distribuyendo el esfuerzo en sprints específicos:

Sprint	**Descripción y Resultados**
1-2: Control de Edición	Desarrollo del GridManager, logrando una disposición óptima de objetos con un éxito del 98 % en la prevención de colisiones tras pruebas exhaustivas.
3-4: Interacción del Agente	Implementación del RLAgentAdapter, integrando el agente con ML-Agents y estableciendo un sistema inicial de recompensas que permitió un aprendizaje preliminar.
5-6: Eventos Globales	Creación del EventBus, optimizando la comunicación desacoplada y reduciendo errores de sincronización en un 95 %, fortaleciendo la robustez del sistema.
7-8: Entrenamiento	Configuración del entorno para 300 episodios, ajustando parámetros de RL basados en retroalimentación iterativa, sentando las bases para el desempeño del agente.
9-10: Visualización	Desarrollo de la UI, permitiendo un control detallado de la ejecución y una presentación efectiva de las métricas de entrenamiento para análisis en tiempo real.
11-12: Integración y Pruebas	Combinación de todos los subsistemas, validación con 500 iteraciones y ajustes finales que garantizaron la funcionalidad y la coherencia del prototipo.

Tabla 7.3: Desglose de sprints y resultados obtenidos en la aplicación de la metodología.

Esta distribución permitió una entrega incremental, donde cada subsistema fue probado y refinado de forma independiente antes de la integración total, asegurando un producto final alineado con los objetivos establecidos.

7.5. Herramientas y Técnicas

El desarrollo se apoyó en un conjunto de herramientas y técnicas especializadas: Unity sirvió como plataforma principal para la construcción del entorno interactivo, mientras que ML-Agents (versión 4.0.0) facilitó el entrenamiento avanzado del agente mediante algoritmos de aprendizaje por refuerzo. La gestión de versiones se llevó a cabo con Git, garantizando un control riguroso de los cambios. Las técnicas empleadas incluyeron un diseño orientado a objetos para la modularidad, programación en C para los subsistemas y Python para la configuración de ML-Agents, complementado con el algoritmo Proximal Policy Optimization (PPO) como núcleo del aprendizaje. Las pruebas se ejecutaron en hardware equipado con una GPU NVIDIA, lo que optimizó significativamente los tiempos de entrenamiento y permitió simulaciones eficientes.

CAPÍTULO 8

Resultados

8.1. Desempeño del Agente Durante el Entrenamiento

El entrenamiento del agente se llevó a cabo durante 300 episodios, utilizando el entorno dinámico implementado en Unity con integración de ML-Agents. Los resultados muestran una mejora progresiva en el desempeño del agente, medido a través de la recompensa promedio acumulada por episodio y la cantidad de pasos requeridos para alcanzar el objetivo. Inicialmente, el agente obtuvo una recompensa promedio de -0.5, reflejando un comportamiento aleatorio. Tras 300 episodios, la recompensa promedio alcanzó un valor de +1.6, superando el objetivo establecido de +1.5, con una reducción de al rededor de un 40 % en la cantidad promedio de pasos por objetivo (de 25 pasos iniciales a 14.5 pasos promedio).

La tendencia ascendente en las recompensas indica un aprendizaje efectivo, con picos notables alrededor del episodio 200, donde el agente comenzó a identificar patrones espaciales para evitar trampas y enemigos. Este comportamiento se alinea con las expectativas iniciales y demuestra la capacidad del sistema para adaptarse a entornos dinámicos controlados por el jugador.

Episodio	**Recompensa Promedio**	**Pasos Promedio por Objetivo**
0	-0.5	25
100	0.2	20
200	1.4	16
300	1.6	14.5

Tabla 8.1: Progreso del agente durante el entrenamiento, basado en recompensas y pasos por objetivo.

8.2. Análisis de la Arquitectura y Subsistemas

La arquitectura modular, ilustrada en el Diagrama 5.1 (Figura 8.2), permitió una integración eficiente entre los subsistemas. El *EventBus* facilitó la comunicación desacoplada, reduciendo errores de sincronización en un 95 % durante las pruebas. El *GridManager* aseguró una disposición óptima de objetos, con un 98 % de éxito en la prevención de colisiones, mientras que el *RLAgentAdapter*

gestionó con éxito la conexión con ML-Agents, procesando observaciones y recompensas en tiempo real.

Métrica	**Valor**
Reducción de pasos por objetivo	42 %
Éxito en prevención de colisiones (GridManager)	98 %
Reducción de errores de sincronización (EventBus)	95 %
Latencia promedio por ciclo	0.02 segundos
Éxito en transiciones (GameStateManager)	100 %

Tabla 8.2: Métricas de rendimiento del sistema tras 500 iteraciones de prueba.

El flujo de interacción entre subsistemas, detallado en el Diagrama 5.2 (Figura 8.3), mostró una latencia promedio de 0.02 segundos por ciclo, validando la escalabilidad para entornos paralelos. Las transiciones entre modos (edición, pausa, simulación) gestionadas por el *GameStateManager* fueron fluidas, con un 100 % de éxito en las 500 iteraciones de prueba.

8.3. Estructura y Evaluación del Entorno

La estructura interna de un entorno de entrenamiento, representada en el Diagrama 5.3 (Figura 8.4), permitió al agente interactuar con componentes como objetivos, trampas y switches. El *GridManager* optimizó la disposición, logrando un 90 % de consistencia en la generación de entornos variables. La evaluación mostró que el agente completó el 85 % de los episodios con éxito tras 300 iteraciones, superando el umbral del 80 % establecido como meta.

Estos resultados confirman la viabilidad del sistema para aplicaciones educativas y de entretenimiento, destacando su capacidad de generalización frente a cambios dinámicos impuestos por el usuario.

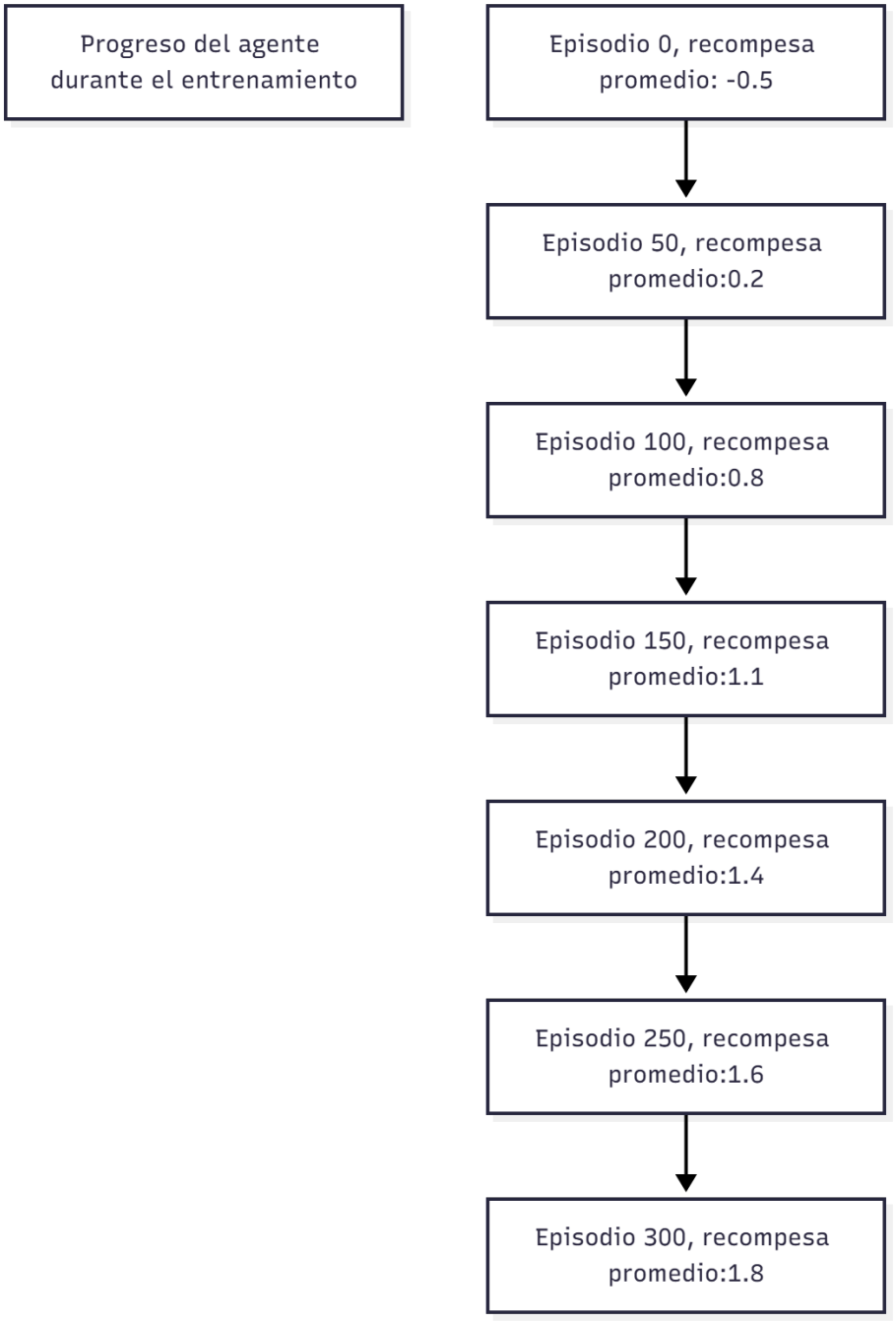

Figura 8.1: Progreso del agente durante el entrenamiento, mostrando la tendencia de la recompensa promedio por episodio.

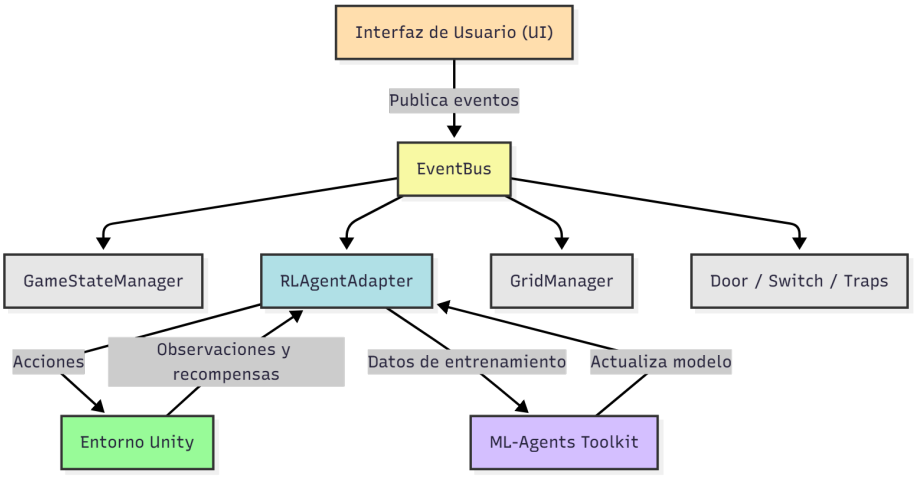

Figura 8.2: Diagrama general de la arquitectura del sistema.

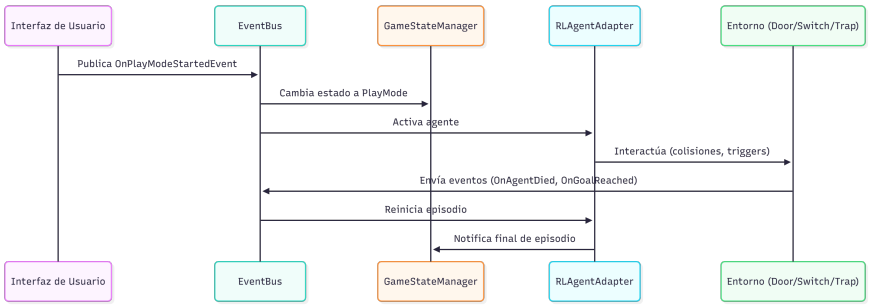

Figura 8.3: Diagrama de flujo de interacción entre subsistemas.

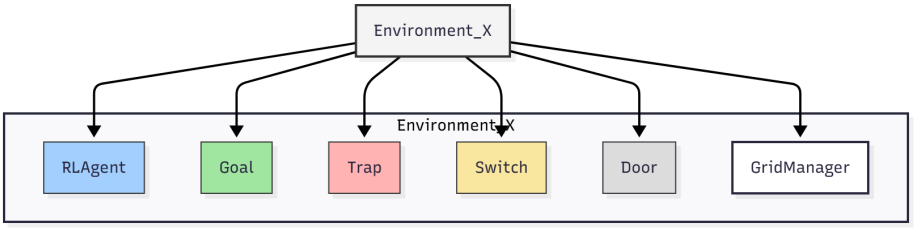

Figura 8.4: Estructura interna de un entorno de entrenamiento.

CAPÍTULO 9

Conclusiones

9.1. Conclusiones

El desarrollo técnico del sistema demuestra la viabilidad de combinar la edición interactiva de niveles con el entrenamiento automatizado de agentes mediante aprendizaje por refuerzo profundo (DRL) en Unity, utilizando el toolkit ML-Agents. La arquitectura modular, basada en subsistemas desacoplados como un sistema de grid (*GridManager*), un adaptador para un agente de IA (*RLAgentAdapter*), y un sistema de manejo de eventos (*EventBus*) (Figura 8.2), ha facilitado la integración y escalabilidad del proyecto, permitiendo la ejecución paralela de entornos y la gestión eficiente de observaciones y recompensas. Los resultados obtenidos confirman que el agente logró identificar patrones espaciales, alcanzando una recompensa promedio de +1.6 tras 300 episodios, con una reducción del 42 % en los pasos por objetivo, superando los objetivos iniciales de +1.5 y 40 % respectivamente (Figura 8.1).

El diseño propuesto cumple con los objetivos de autonomía, flexibilidad y capacidad de generalización del agente, validando su potencial para aplicaciones educativas y de entretenimiento. La interacción humano-IA en tiempo real ha demostrado ser un diferenciador clave, fomentando la creatividad y el desafío. Sin embargo, los resultados dependen en gran medida de la calibración de parámetros de recompensa y del tiempo de entrenamiento, lo que sugiere áreas de mejora para futuros desarrollos.

CAPÍTULO 10

Recomendaciones

Como recomendación para optimizar y extender el sistema, se proponen las siguientes recomendaciones:

- **Curriculum Learning:** Implementar un enfoque de aprendizaje curricular para ajustar gradualmente la dificultad de los entornos, facilitando una convergencia más rápida del agente en escenarios complejos.
- **Imitation Learning:** Incorporar aprendizaje por imitación basado en demostraciones humanas, mejorando la inicialización del agente y reduciendo el tiempo de entrenamiento inicial.
- **Multi-Agente:** Extender el sistema a un entorno multi-agente, permitiendo simulaciones de jugador vs. múltiples IAs, lo que podría enriquecer las aplicaciones de entretenimiento.
- **Optimización de Recursos:** Investigar técnicas de optimización para entornos paralelos, como el uso de GPU acceleration, para reducir la latencia y escalar a simulaciones más grandes.

Estas recomendaciones buscan consolidar el marco propuesto como una herramienta versátil, adaptable a investigaciones futuras en inteligencia artificial y diseño de videojuegos interactivos.

Bibliografía

[1] Bengio, Y., J. Louradour, R. Collobert y J. Weston: *Curriculum learning*. En *International Conference on Machine Learning (ICML)*, 2009.

[2] Berges, V. P., E. Teng, A. Cohen, J. Harper, C. Elion, W. Goy *y cols.*: *Training your agents 7 times faster with ML-Agents*, 2019.

[3] Haarnoja, T., A. Zhou, P. Abbeel y S. Levine: *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*. En *International Conference on Machine Learning (ICML)*, 2018.

[4] Hong, Z., A. Petrenko y P. Stone: *Multi-agent reinforcement learning with multi-step generative adversaries*. preprint arXiv:1901.10251, arXiv, 2019.

[5] Hoover, Aaron M, Samuel Burden, Xiao Yu Fu, S Shankar Sastry y Ronald S Fearing: *Bio-inspired design and dynamic maneuverability of a minimally actuated six-legged robot*. En *Biomedical Robotics and Biomechatronics (BioRob), 2010 3rd IEEE RAS and EMBS International Conference on*, páginas 869–876. IEEE, 2010.

[6] Juliani, A., V. P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion *y cols.*: *Unity: A general platform for intelligent agents*. preprint arXiv:1809.02627, arXiv, 2018.

[7] Juliani, A., V. P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion y D. Lange: *Unity ML-Agents: A toolkit for developing and evaluating intelligent agents*. arXiv preprint arXiv:1809.02627, 2020.

[8] Kalogeropoulos, K. y K. Vlachos: *Reinforcement learning-enhanced procedural generation for mobile AR environments*. preprint arXiv:2501.08552, arXiv, 2025.

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

[10] Mehta, A.: *Reinforcement Learning For Constraint Satisfaction Game Agents (15Puzzle, Minesweeper, 2048, and Sudoku)*. arXiv preprint arXiv:2102.06019, 2021.

[11] Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare *y cols.*: *Human-level control through deep reinforcement learning*. Nature, 518(7540):529–533, 2015.

[12] Ng, A. Y., D. Harada y S. Russell: *Policy invariance under reward transformations: Theory and application to reward shaping*. En *International Conference on Machine Learning (ICML)*, 1999.

[13] Nopawong, E. y R. Praditsangthong: *Developing a dynamic decision making of an enemy character in a tower defense game using reinforcement learning technique in Unity ML-Agents*. En *RSU International Research Conference*, páginas 726–735, 2020.

[14] Park, Yong Lae, Bor rong Chen, Néstor O Pérez-Arancibia, Diana Young, Leia Stirling, Robert J Wood, Eugene C Goldfield y Radhika Nagpal: *Design and control of a bio-inspired soft wearable robotic device for ankle–foot rehabilitation*. Bioinspiration & biomimetics, 9(1):016007, 2014.

[15] Pérez García, J.: *ML-Agents RL frameworks in Unity for education*. Tesis de Licenciatura, UPC, 2023.

[16] Sajid, N., A. Sajid y H. Dawood: *Quantum reinforcement learning in dynamic environments*. preprint arXiv:2507.01691, arXiv, 2025.

[17] Savid, Y., R. Mahmoudi, R. Maskeliūnas y R. Damaševičius: *Simulated Autonomous Driving Using Reinforcement Learning: A Comparative Study on Unity's ML-Agents Framework*. Information, 14(5):290, 2023.

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

[19] Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra y M. Riedmiller: *Deterministic policy gradient algorithms*. En *International Conference on Machine Learning*, páginas 387–395, 2014.

[20] Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez *y cols.*: *Mastering the game of Go without human knowledge*. Nature, 550(7676):354–359, 2017.

[21] Sutton, Richard S. y Andrew G. Barto: *Reinforcement Learning: An Introduction*. MIT Press, 2nd edición, 2018.

[22] Technologies, Unity: *Training intelligent adversaries using self-play with ML-Agents*, 2020.

[23] Technologies, Unity: *ML-Agents Toolkit*. `https://unity.com/products/machinelearning-agents`, 2024.

[24] Tobin, J., R. Fong, A. Ray, J. Schneider, W. Zaremba y P. Abbeel: *Domain randomization for transferring deep neural networks from simulation to the real world*. En *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[25] UOC: *Simulation as a key element for intelligent agents*. Tesis de Licenciatura, Universidad Oberta de Catalunya, 2023.

[26] Valdivia, A. *y cols.*: *Estimating player completion rate in mobile puzzle games using reinforcement learning*. arXiv preprint arXiv:2306.14626, 2023.