I

Universidad de las Ciencias Informáticas

Facultad 4

“Aprendizaje reforzado para el desarrollo de jugadores virtuales”

Trabajo Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor

Daniel Enrique Sánchez Zerquera

Tutores

Ms.C. A Rubén Alcolea Nuñez

Ing. Rafael Iglesias Miranda

La Habana, noviembre de 2022

DEDICATORIA

Dedico este trabajo a mi familia, en especial a mi madre y padre que han sido mi apoyo durante estos años de estudio, gracias a ustedes estoy hoy aquí.

AGRADECIMIENTOS

A mi madre, a mi padre y a mis hermanos por su amor, comprensión y apoyo incondicional.

A moru, estos años no hubieran sido tan buenos sin ti, gracias por todos los juegos juntos, las risas y hasta las peleas.

A Clau, por su enorme cariño y apoyo.

A mi familia por su preocupación y apoyo.

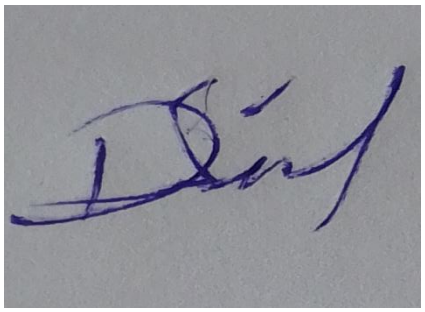
A mis tutores, amigos y todas aquellas personas que contribuyeron de alguna manera en mi vida a lo largo de la universidad.

A todos, muchas gracias.

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis que tiene por título: y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los 30 días del mes de noviembre del año 2022

<Daniel Enrique Sanchez Zerquera>



Firma del Autor

< Rubén Alcolea Nuñez>

Firma del Tutor

< Rafael Iglesias Miranda>

Firma del Tutor

\

Resumen

El aprendizaje reforzado es una vertiente del *machine learning* o aprendizaje automático que en la última década ha ganado gran protagonismo por su elevado desempeño en todo tipo de tareas en las que programar manualmente una inteligencia artificial que las realice sea demasiado complicado, impulsado por constantes avances y nuevos algoritmos que surgen año tras año. Expertos han logrado además, mediante el uso de técnicas de aprendizaje reforzado, un hito antes imposible de alcanzar: desempeño sobrehumano de agentes en juegos mucho más complejos que un simple Pong, lo que, junto a la ausencia en el Centro de Tecnologías Interactivas de la Universidad de las Ciencias Informáticas de productos que usen este tipo de tecnología, impulsó la investigación para desarrollar jugadores virtuales empleando aprendizaje reforzado. Un profundo análisis de diversos elementos teóricos relacionados con el tema permite la selección del algoritmo PPO con una CNN por encima de otros para emplear como base para el agente, implementado con ayuda del paquete ML-Agents para Unity y Python. La metodología XP fue la guía para el diseño e implementación de la solución, teniendo como apoyo una serie de pruebas de aceptación para el demo y de desempeño del agente para validar la solución como adecuada.

Palabras clave: agente, aprendizaje automático, aprendizaje reforzado, inteligencia artificial

\

Abstract

Reinforcement learning is a branch of machine learning that in the last decade has gained great prominence due to its high performance in all kinds of tasks in which manually programming an artificial intelligence to perform them is too complicated, driven by constant advances and new algorithms that emerge year after year. Experts have also achieved, through the use of reinforcement learning techniques, a milestone previously impossible to achieve: superhuman performance of agents in games much more complex than simple Pong, which, together with the absence in the Interactive Technologies Center of the University of Informatics Sciences of products that use this type of technology, promoted research to develop virtual players using reinforcement learning. A deep analysis of various theoretical elements related to the subject allows the selection of the PPO algorithm with a CNN over others to use as a base for the agent, implemented with the help of the ML-Agents package for Unity and Python. The XP methodology was the guide for the design and implementation of the solution, supported by a series of acceptance tests for the demo and agent performance to validate the solution as adequate.

Keywords: agent, machine learning, reinforcement learning, artificial intelligence

Índice

Abstract VI

INTRODUCCIÓN 1

Capítulo 1: Fundamentación Teórica 6

1.1 Aprendizaje Automático 6

1.1.1 Tipos de Aprendizaje Automático 6

1.1.2 Aprendizaje Reforzado 7

1.1.3 Aprendizaje Profundo 11

1.1.4 Deep Reinforcement Learning 12

1.1.5 Redes Neuronales 13

1.1.6 Policy Gradient Methods 16

1.2 Videojuegos 18

1.2.1 IA en Videojuegos 18

1.2.2 Aprendizaje Profundo en videojuegos 20

1.3 Aplicación de IA y RF en videojuegos desarrollados en VERTEX 21

1.4 Análisis de homólogos 22

1.5 Algoritmos de aprendizaje reforzado profundo 23

1.5.1 Deep Q-Network 23

1.5.2 Trust Region Policy Optimization 25

1.5.3 Actor Critic with Experience Replay 27

1.5.4 Proximal Policy Optimization 28

1.5.5 Soft Actor-Critic 28

1.5.6 Selección del algoritmo a utilizar 29

1.6 Metodologías para el desarrollo de software 30

1.5.1 Metodología de Software 30

1.7 Herramientas y tecnologías 32

1.7.1 Lenguaje de Programación y Motor de videojuego 32

1.7.2 Herramientas para el modelado 35

1.8 Conclusiones del capítulo 36

Capítulo 2: Planificación y diseño de la propuesta de solución 37

2.1 Propuesta de solución 37
2.2 Planificación 48
2.3 Requisitos 48
2.3.1 Historias de Usuario 49
2.3.2 Plan de Entrega 50
2.3.3 Iteraciones 51
2.4 Diseño 51
2.4.1 Tarjetas CRC 52
2.5 Conclusiones del capítulo 53
Capítulo 3: Implementación del agente y pruebas realizadas 54
3.1 Estándar de codificación 54
3.2 Tareas de Ingeniería 55
Primera Iteración: 56
Segunda Iteración: 59
3.3 Descripción de la solución 60
3.4 Pruebas 61
3.4.1 Pruebas de Aceptación 62
3.4.2 Pruebas de desempeño del agente 64
3.5 Conclusiones del capítulo 69
Conclusiones generales 70
Recomendaciones 71
Referencias Bibliográficas 72
Anexos 77
1. Historias de Usuario: 77
Casos de Prueba: 78

Indice de tablas

Tabla 1 .*Comparación de Metodologías* 30
Tabla 2 . *Hiperparámetros* 38
Tabla 3 . *Requisitos* 49
Tabla 4 . *HU 1: Modo de juego individual* 50
Tabla 5 . *HU 2: Menú principal* 50
Tabla 6 . *HU 5: Acciones del agente* 50
Tabla 7 . *Plan de Entrega* 50
Tabla 8 . *Iteraciones* 51
Tabla 9 . *Tarjeta CRC 1: Disparos* 52
Tabla 10 . *Tarjeta CRC 2: PlayerScript* 52
Tabla 11 . *Tarjeta CRC 3: SpaceShipAgent* 53
Tabla 12 . *Iteración 1. Tarea 1* 56
Tabla 13 . *Iteración 1. Tarea 2* 56
Tabla 14 . *Iteración 1. Tarea 3* 56
Tabla 15 . *Iteración 1. Tarea 4* 57
Tabla 16 . *Iteración 1. Tarea 5* 57
Tabla 17 . *Iteración 1. Tarea 6* 57
Tabla 18 . *Iteración 1. Tarea 7* 57
Tabla 19 . *Iteración 1. Tarea 8* 58
Tabla 20 . *Iteración 1. Tarea 9* 58
Tabla 21 . *Iteración 1. Tarea 10* 58
Tabla 22 . *Iteración 2. Tarea 11* 59
Tabla 23 . *Iteración 2. Tarea 12* 59
Tabla 24 . *Iteración 2. Tarea 13* 59
Tabla 25 . *Iteración 2. Tarea 14* 60
Tabla 26 . *Iteración 2. Tarea 15* 60
Tabla 27 . *CP iniciar partida individual* 62
Tabla 28 . *CP pausar una partida* 62
Tabla 29 . *CP reanudar una partida* 63
Tabla 30 . *Recompensas por prueba* 64

Indice de Figuras

Figura 1 . *Red Neuronal Profunda* 12
Figura 2 . *Una arquitectura CNN simple, compuesta de solo cinco capas.* 14
Figura 3 . *Comparación entre Q-Learning y DQN. Elaborado por el autor.* 23
Figura 4 . *La Ecuación de Bellman indica como actualizar la Q-Table* 24
Figura 5 . *Mejoras de ACER en la complejidad de muestra (IZQUIERDA) y computación (DERECHA) en Atari* 27
Figura 6 . Componentes del entono de aprendizaje 35
Figura 7 . *Parámetros de comportamiento del agente* 45
Figura 8 . *RayPerceptionSensor 2D, componente que otorgar visión del entorno al agente* 46
Figura 9 . *Definición de clases* 54
Figura 10 . *Declaración de funciones y atributos* 54
Figura 11 . *Definición de parámetros dentro de las funciones y constructores de las clases* 54
Figura 12 . *Definición de estructuras de control* 55
Figura 13 . *Definición de estructuras de bucles* 55
Figura 14 . *Resultados de las pruebas de aceptación* 63
Figura 15 . *Resultados de pruebas 3, 4, 5 y 6* 66
Figura 16 . *Resultados de pruebas 1, 2, 7 y 8* 66
Figura 17 . Resultados de pruebas de inferencia 68

INTRODUCCIÓN

La inteligencia artificial lleva varias décadas desde su primera concepción, y lo cierto es que ha dado grandes avances en los últimos tiempos, permitiendo que se dejen de lado los paradigmas clásicos de la programación en pos de emular un comportamiento inteligente o lógico, lo cual constituye un sueño para muchos investigadores. Primeramente, es necesario abordar el tema de la Inteligencia Artificial como ciencia. Algunos estudiosos se refieren al campo de estudio de la inteligencia artificial como Inteligencia Computacional, la cual responde al estudio del diseño de agentes inteligentes [1]. Un agente es un ente que actúa en un entorno, realizando acciones. Los agentes incluyen todo tipo de entes, ya sean gusanos, perros, termostatos, aviones, humanos, organizaciones y la sociedad [1].

Un agente inteligente es un sistema que actúa inteligentemente: lo que hace es apropiado para sus circunstancias y su objetivo, es flexible para cambiar los entornos y los objetivos, aprende de la experiencia y toma las decisiones adecuadas dadas las limitaciones de percepción y la computación finita [1]. El objetivo científico central de la inteligencia computacional es comprender los principios que hacen posible el comportamiento inteligente, en sistemas naturales o artificiales; su hipótesis principal es que el razonamiento es computación; y su objetivo ingenieril es especificar métodos para el diseño de artefactos útiles e inteligentes [1].

Estrictamente, la inteligencia artificial (IA) es la inteligencia exhibida por las máquinas. En informática, una máquina "inteligente" ideal es un agente racional flexible que percibe su entorno y realiza acciones que maximizan sus posibilidades de éxito en algún objetivo. Coloquialmente, el término "inteligencia artificial" se aplica cuando una máquina imita funciones "cognitivas" que los humanos asocian con otras mentes humanas, como "aprender" y "resolver problemas". El aprendizaje automático o machine learning es una disciplina enfocada en dos cuestiones interrelacionadas: ¿Cómo se pueden construir sistemas informáticos que mejoran automáticamente a través de la experiencia? y ¿Cuáles son las leyes fundamentales de la teoría de la información computacional estadística que gobiernan todos los sistemas de aprendizaje, incluidas las computadoras, humanos y organizaciones? El estudio del machine learning es importante para abordar estas preguntas científicas y de ingeniería fundamentales y por el software altamente práctico que ha producido y desplegado a través de muchas aplicaciones [2].

El aprendizaje automático ha progresado dramáticamente en las últimas dos décadas, yendo desde ser una curiosidad de laboratorio a ser una tecnología práctica en general y de uso comercial. Dentro de la IA, el aprendizaje automático se ha convertido en el método de elección para el desarrollo de software práctico para visión por computadora, reconocimiento de voz, procesamiento de lenguaje natural y control de robots [2], llegando hasta la ingeniería de naves espaciales, las finanzas, el entretenimiento y la biología computacional, aplicaciones biomédicas y médicas y otras aplicaciones [3].

Una de las vertientes principales del machine learning es el Aprendizaje Reforzado o *Reinforcement Learning*, un área del aprendizaje automático que se ocupa de cómo los agentes inteligentes deben realizar acciones en un entorno para maximizar la noción de recompensa acumulativa [4], siendo uno de los tres paradigmas básicos de aprendizaje automático, junto con el aprendizaje supervisado y el aprendizaje no supervisado. Al agente no se le dice qué acciones realizar, sino que debe descubrir qué acciones producen la mayor recompensa probándolas [5].

En Cuba, se pueden apreciar varios logros y realizaciones como los sistemas de video-vigilancia que hoy funcionan en varios sitios del país, programas de salud y de procesamiento de textos, además de software muy sofisticados para otros fines como la agricultura, entre otros sectores [6]. Sin embargo aunque desde los años 90 del pasado siglo se trabaja en ese tema por varias universidades como la CUJAE, la Universidad de La Habana, la Universidad de Camagüey, la Universidad de Oriente, la “Marta Abreu” de Las Villas, y en empresas e institutos de investigación, aún resta un largo camino por andar para que la inteligencia artificial desempeñe el rol que debe alcanzar en el progreso de la nación.

Se destacan la utilización de técnicas de inteligencia artificial en la lucha contra la covid-19, en este esfuerzo abordan el desarrollo de modelos para el pronóstico y evaluación de la pandemia, incluyendo los análisis desde la perspectiva de la geolocalización y de la influencia de los factores climáticos [6]. Además, se aprecian estudios de métodos para la clasificación de la gravedad de los pacientes y pronósticos de su evolución, la evaluación de resultados en aplicaciones informáticas y el estudio de movilidad basados en análisis de grandes volúmenes de datos (*Big Data*) [6].

Cuba lleva incursionando desde hace unos años en el desarrollo de videojuegos, los cuales son escenarios ideales en el estudio y desarrollo de la IA y especialmente de algoritmos de *machine learning*, debido a la naturaleza incierta inherente de los videojuegos, así como sus enormes cantidades de entradas y salidas posibles. La Universidad de las Ciencias Informáticas (UCI) es una institución con la misión de formar profesionales calificados en la rama de la informática y la producción de aplicaciones y servicios mediante el vínculo docencia-investigación-producción. La UCI cuenta con varios centros de desarrollo en sus facultades, entre ellos el Centro de Tecnologías Interactivas (VERTEX), el cual cuenta con una línea de desarrollo de videojuegos que exhibe entre sus títulos más importantes videojuegos como las Neurona 1 y 2, y el videojuego MOBA-RPG de acción Coliseum [7]. Otro resultado importante es el desarrollo de la plataforma Cosmox para videojuegos en línea, la cual en su primera etapa se concibió para posibilitar la realización de partidas en línea para videojuegos por turnos, tales como Batalla naval, Tanks 2D, Dominó, entre otros.

Una de las deficiencias del proceso de desarrollo actual de videojuegos en el centro VERTEX, es el desarrollo de jugadores virtuales con un comportamiento lo suficientemente inteligente para que el usuario disfrute de una experiencia de juego atractiva y retadora, que lo motive a seguir jugando. Los jugadores virtuales se pueden emplear entre otros roles como personajes no jugables (NPC

por sus siglas en inglés), como pueden ser unidades enemigas que atacan al protagonista del videojuego o como jugadores virtuales con un nivel de inteligencia artificial que puede compararse con el de otro jugador en videojuegos por turnos como pueden ser el dominó, el ajedrez u otros similares. En ambos casos, el nivel de aprendizaje que se logre incorporar al jugador virtual, influirá en la experiencia final del usuario y su motivación por continuar jugando el videojuego.

La forma actual de abordar el problema de la necesaria incorporación de inteligencias artificiales en los videojuegos desarrollados por VERTEX, ha sido la implementación heurística de comportamiento basadas en su mayoría en reglas de comportamiento predefinidas (Coliseum) o en modelos de optimización (Kuba Kart) que logran el cumplimiento del objetivo. Algunos ejemplos de estas heurísticas son apreciables en los personajes enemigos del videojuego Coliseum, los cuales tienden a seguir el patrón de atacar y buscar las runas de vida o de algún poder siempre que estas aparecen en el terreno. Sin embargo, estas estrategias no logran que el jugador virtual sea capaz de aprender o mejorar su comportamiento, lo que provoca que ciertas situaciones dentro del videojuego sean predecibles después de cierto tiempo de juego.

Como resultado de la problemática anterior se define el siguiente **problema a resolver**: ¿Cómo incorporara los videojuegos jugadores virtuales capaces de aprender? Se define como **objeto de estudio** los jugadores virtuales y como **objetivo general** de la investigación desarrollar un demo que permita utilizar un jugador virtual capaz de aprender para mejorar su comportamiento en el videojuego. A partir del objetivo, se define como **campo de acción** los jugadores virtuales basados en aprendizaje reforzado.

En pos de cumplir el objetivo propuesto y guiar el desarrollo del presente trabajo, fueron definidas las siguientes **tareas a cumplir por el estudiante:**

1. Elaboración del marco teórico a partir del estudio del estado del arte de los jugadores virtuales basados en aprendizaje.
2. Diseño de la propuesta de solución para incorporar a un videojuego jugadores virtuales capaces de aprender y mejorar su comportamiento.
3. Definición de requisitos funcionales y no funcionales para la solución.
4. Diseño de la solución propuesta.
5. Implementación de un demo que utilice un jugador virtual capaz de aprender para mejorar su comportamiento en el videojuego.
6. Validación de la solución propuesta.
7. Pruebas a la solución implementada.

Como **resultado** de la presente investigación se espera obtener un demo que demuestre la utilización de un jugador virtual capaz de aprender para mejorar su comportamiento en el videojuego.

Para la realización de las tareas de investigación se emplearon los métodos de investigación científica que se describen a continuación:

Métodos teóricos:

- **Histórico-Lógico**: método que se utiliza para el estudio de los aspectos de interés acerca del desarrollo de agentes que utilicen aprendizaje reforzado en videojuegos. Se estudió la definición de aprendizaje automático y aprendizaje reforzado, varios algoritmos de aprendizaje reforzado y aprendizaje reforzado profundo utilizados en videojuegos.

- **Analítico-Sintético**: empleado con el propósito de estudiar la bibliografía necesaria para este tema, de la cual se extrajeron y sintetizaron elementos relacionados con el aprendizaje automático, aprendizaje reforzado y su uso en los videojuegos, así como documentación referente al despliegue y desarrollo de componentes creados bajo el Motor de videojuegos Unity, el uso de paquetes en el motor gráfico y en Python.

-Inductivo - Deductivo: se realizó un estudio general sobre videojuegos con IA incorporada e IA capaces de jugar videojuegos, especialmente aquellos en los que se utilizaron técnicas de aprendizaje reforzado. Con esto se logró una comprensión más precisa sobre el tema que sustenta la investigación; basando el camino a la solución en argumentos debidamente documentados a los que se podrá recurrir con facilidad.

Métodos empíricos:

- **Consulta bibliográfica**: empleado en la elaboración del marco teórico de la investigación, mediante la revisión de artículos y libros que giran en torno al aprendizaje reforzado, en especial algoritmos prominentes de aprendizaje reforzado profundo.

- **Pruebas**: utilizado para comprobar que el agente desarrollado para incluirse en el demo propuesto como solución tenga un desempeño satisfactorio, para así poder cumplir el objetivo de la investigación.

Para abarcar el contenido a abordar el documento contiene los siguientes capítulos:

Capítulo 1: Fundamentación teórica: capítulo que contiene los elementos relacionados con el objeto de estudio y el campo de acción. Incluye un estudio del marco teórico fundamental alrededor del aprendizaje reforzado y sus algoritmos prominentes, en pos de seleccionar uno para utilizar en la solución. Se define la metodología XP como la adecuada para el diseño e implementación de la solución, con el motor gráfico Unity, incluyendo el paquete ML-Agents, y el lenguaje de programación Python como las herramientas y tecnologías a utilizar.

Capítulo 2: Planificación y diseño de la propuesta de solución: en el desarrollo de este capítulo se reflejan las características del sistema a desarrollar. Se diseña y describe la propuesta de solución, especificando las características del agente, su forma de recolectar información sobre el entorno y las acciones que puede realizar, incluyendo los hiperparámetros a utilizar en el entrenamiento. Se crean los artefactos relativos al diseño del desarrollo de software basado en la metodología XP.

Capítulo 3: Implementación del agente y pruebas realizadas: en el capítulo se plasma el proceso de implementación de la solución, definiéndose las diferentes tareas por iteraciones según es definido por la metodología XP. Se realizaron pruebas de desempeño sobre varias redes neuronales con variaciones en las recompensas empleadas, entre las cuales fue seleccionada la de mejores resultados para ser empleada por el agente en el demo, y de aceptación para verificar si la solución cumple con las especificaciones del cliente.

Capítulo 1: Fundamentación Teórica

En el presente capítulo se realiza un análisis sobre los fundamentos teóricos de la investigación, abordando conceptos relacionados al aprendizaje reforzado, con el objetivo final de llegar seleccionar un algoritmo y un tipo de red neuronal a ser empleados en la solución. Se hace un estudio de sistemas homólogos existentes en el mundo y sus posibles aportes a la investigación. Se caracteriza además la metodología, las tecnologías, lenguajes y herramientas a utilizar para el desarrollo de la solución.

1.1 Aprendizaje Automático

El aprendizaje automático o *machine learning* es una rama en constante evolución de los algoritmos computacionales que están diseñados para emular la inteligencia humana aprendiendo del entorno circundante [3]. Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas T y de medida de rendimiento P, si su rendimiento en tareas en T, medido por P, mejora con la experiencia E [8].

Muchos desarrolladores de sistemas de inteligencia artificial reconocen que actualmente, para muchas aplicaciones, puede ser mucho más fácil entrenar un sistema mostrándole ejemplos del comportamiento de entrada-salida deseado que programarlo manualmente anticipando la respuesta deseada para todas las entradas posibles [2]. Y de hecho, gran parte de la investigación de aprendizaje automático está inspirada por problemas de peso de biología, medicina, finanzas, astronomía, etc [9], lo que convierte al *machine learning* en una disciplina inherentemente práctica, con un enorme potencial que crece con el paso del tiempo y los avances teóricos y de hardware.

1.1.1 Tipos de Aprendizaje Automático

El aprendizaje automático consta de tres tipos o enfoques principales del aprendizaje automático, que se resumen a continuación:

Aprendizaje supervisado: La característica definitoria del aprendizaje supervisado es la disponibilidad de datos de entrenamiento etiquetados. El nombre invoca la idea de un "supervisor" que instruye al sistema de aprendizaje sobre las etiquetas para asociarlas con ejemplos de capacitación. Por lo general, estas etiquetas son las que asocian una clase a un conjunto de datos en problemas de clasificación. Los algoritmos de aprendizaje supervisado inducen modelos a partir de estos datos de entrenamiento y estos modelos se pueden usar para clasificar otros datos no etiquetados [10].

Aprendizaje no supervisado: El objetivo del aprendizaje no supervisado es que, al recibir una entrada sin etiquetas, la máquina pueda construir representaciones de la entrada que se puedan usar para la toma de decisiones, la predicción de entradas futuras, la comunicación eficiente de las

entradas a otra máquina, etc. En cierto sentido, el aprendizaje no supervisado puede ser pensado como encontrar patrones en los datos por encima y más allá de lo que sería considerado puro ruido no estructurado [11].

Aprendizaje reforzado: El aprendizaje reforzado o reinforcement learning consiste en aprender qué hacer (cómo mapear situaciones en acciones) de modo que se pueda maximizar una señal de recompensa numérica. Al agente no se le dice qué acciones realizar, sino que debe descubrir qué acciones producen la mayor recompensa probándolas [5].

Entre estos tres enfoques, el aprendizaje reforzado es el que más se ajusta al contexto de los videojuegos: el concepto de juego, como un entorno con un conjunto de reglas definidas en el que el objetivo principal es lograr una meta determinada concuerda con el concepto de aprendizaje reforzado. El aprendizaje no supervisado por su parte, no encuentra una forma de ser utilizado en los videojuegos directamente; en cuanto al supervisado, si bien se puede emplear, no es suficiente en entornos complejos, pues el conjunto de partidas de muestra a analizar por el agente seguramente no podría cubrir la mayoría de estados posibles en un videojuego. La exploración inherente del entrenamiento utilizado en el aprendizaje reforzado ayuda a explorar mayor cantidad de posibles estados, y gracias a su propio concepto de como realizar el aprendizaje los juegos resultan ser un escenario ideal para los algoritmos de aprendizaje reforzado, siendo estos argumentos más que suficientes para justificar su uso en videojuegos por encima de otros enfoques.

1.1.2 Aprendizaje Reforzado

El aprendizaje reforzado (*reinforcement learning* en inglés) es un área de aprendizaje automático que se ocupa de cómo los agentes inteligentes deben tomar medidas en un entorno para maximizar la noción de recompensa acumulativa. El aprendizaje por refuerzo es uno de los tres paradigmas básicos de aprendizaje automático, junto con el aprendizaje supervisado y el aprendizaje no supervisado, que difiere del aprendizaje supervisado en no necesitar que se presenten pares de entrada/salida etiquetados, y en no necesitar acciones subóptimas para ser corregidas explícitamente. En cambio, la atención se centra en encontrar un equilibrio entre la exploración (de territorio inexplorado) y la explotación (del conocimiento actual) [4].

El aprendizaje reforzado consta básicamente de los siguientes componentes:

Agente: constituye el modelo que se desea entrenar y que aprenda a tomar decisiones.

Ambiente: es el entorno en donde interactúa y “se mueve” el agente, representado por un conjunto de estados en cada momento (S). El ambiente contiene las limitaciones y reglas posibles a cada momento.

Entre ellos hay una relación que de constante retroalimentación y cuenta con los siguientes nexos:

Acciones: las posibles acciones que puede tomar en un momento determinado el Agente(A).

Estado (del ambiente): son los indicadores del ambiente de cómo están los diversos elementos que lo componen en ese momento.

Recompensas: a raíz de cada acción tomada por el Agente, podrá obtener un premio o una penalización que orientarán al Agente en si lo está haciendo bien o mal.

Se cuentan además con un conjunto de reglas que definen la transición entre los estados, la recompensa inmediata escalar de una transición y lo que observa el agente, o sea la información que extrae del entorno. En un primer momento, el agente recibe un estado inicial y toma una acción con lo cual influye e interviene en el ambiente, y esa decisión tendrá sus consecuencias: en la siguiente iteración el ambiente devolverá al agente el nuevo estado y la recompensa obtenida. Si la recompensa es positiva se reforzará ese comportamiento para el futuro. En cambio, si la recompensa es negativa dicho valor servirá de penalización, para que ante la misma situación el agente actúe de manera distinta, cada uno de estos datos es almacenado en forma de políticas, para ser utilizada en el futuro, dando lugar al aprendizaje. El esquema en el que se apoya el *Reinforcement Learning* es en el de Proceso de Decisión de Markov(MDP por sus siglas en ingles)[1] [4].

1.1.2.1 Conceptos adicionales relacionados al aprendizaje reforzado

Políticas

Una política define la forma de comportarse del agente de aprendizaje en un momento dado. Se puede decir que una política es un mapeo de los estados percibidos del medio ambiente a las acciones a tomar cuando se encuentra en esos estados [12].

La forma que tiene un algoritmo de aprender sobre una política se divide en 2 vertientes, aprendizaje en política (*on-policy*) o fuera de política (*off-policy*). El aprendizaje *on-policy* aprende valores de acción no para la política óptima, sino para una política casi óptima que todavía explora: una misma política que usa para elegir el comportamiento del agente y actualiza constantemente. Un enfoque más directo es usar dos políticas, una que se aprende y que se convierte en la política óptima, y uno que es más exploratoria y se utiliza para generar comportamiento. La política sobre la que se aprende se denomina política objetivo o destino (*Target Policy*), y la política utilizada para generar el comportamiento se denomina política de conducta o de comportamiento (Behavior Policy). En este caso decimos que el aprendizaje es a partir de los datos “fuera” de la política objetivo, y el proceso general se denomina aprendizaje *off-policy* [5].

---

[1] En matemáticas, un proceso de decisión de Markov (MDP) es un proceso de control estocástico de tiempo discreto. Proporciona un marco matemático para modelar la toma de decisiones en situaciones en las que los resultados son en parte aleatorios y en parte bajo el control de quien toma las decisiones.

Política objetivo $\pi(a|s)$: Es la política que un agente está tratando de aprender, es decir, el agente está aprendiendo la función de valor para esta política.

Política de comportamiento $\beta(a|s)$: Es la política que utiliza un agente para seleccionar una acción, es decir, el agente sigue esta política para interactuar con el entorno [12].

La notación (a|s) se refiere a *action*|*state*, la relación estado-acción, donde para un estado s se toma una acción a.

Los métodos *on-policy* son generalmente más simples y se consideran primero a la hora de resolver problemas. Los métodos *off-policy* requieren conceptos y notación adicionales, y debido a que los datos se deben a una política diferente, son a menudo de mayor variación y convergen más lento. Por otro lado, los métodos *off-policy* son más poderosos y generales [5].

Modelo

El modelo es un elemento que imita el comportamiento del entorno, o más generalmente, que permite hacer inferencias sobre cómo se comportará el medio ambiente. Por ejemplo, dado un estado y una acción, el modelo podría predecir el siguiente estado resultante y próxima recompensa. Los modelos se utilizan para la planificación, por lo que nos referimos a cualquier forma de decidir en un curso de acción al considerar posibles situaciones futuras antes de que realmente sean experimentado. Métodos para resolver problemas de aprendizaje por refuerzo que utilizan modelos y planificación se denominan métodos basados en modelos (*model-based*), a diferencia de los métodos más simples sin modelo *(model-free*) que son explícitamente aprendices de prueba y error, vistos como casi lo opuesto a la planificación [5].

Recompensas

Una señal de recompensa define el objetivo de un problema de aprendizaje por refuerzo. en cada momento paso, el entorno envía al agente de aprendizaje por refuerzo un único número llamado la recompensa. El único objetivo del agente es maximizar la recompensa total que recibe sobre el largo plazo La señal de recompensa define así cuáles son los eventos buenos y malos para el agente [5].

En el aprendizaje por refuerzo, el objetivo final del agente es descubrir un comportamiento (una política) que maximice una recompensa y para ello se deberá proporcionar al agente una o más señales de recompensa para usar durante el entrenamiento. Por lo general, una recompensa está definida por su entorno y corresponde a alcanzar algún objetivo. A esto nos referimos como recompensas extrínsecas, ya que se definen fuera del algoritmo de aprendizaje.

Las recompensas, sin embargo, también se pueden definir fuera del entorno, para alentar al agente a comportarse de cierta manera o para ayudar al aprendizaje de la verdadera recompensa

extrínseca. Nos referimos a estas recompensas como señales de recompensa intrínsecas, cuando un agente es recompensado por comportamientos distintos a los estrictamente relacionados con la tarea que se está realizando, por ejemplo, al explorar o “jugar” con elementos de su entorno, que lo llevan a lograr el objetivo. La recompensa total que el agente aprenderá a maximizar puede ser una combinación de señales de recompensa extrínsecas e intrínsecas [13].

Experience Replay

Experience Replay es una técnica de memoria de reproducción utilizada en el aprendizaje reforzado en el que se almacenan las experiencias del agente en cada paso de tiempo en un conjunto de datos agrupados durante muchos episodios en una memoria de reproducción (*replay buffer*)[2]. Luego, generalmente se prueba la memoria aleatoriamente para obtener un minilote de experiencia y lo usamos para aprender *off-policy*. Esto aborda el problema de la autocorrelación que conduce a un entrenamiento inestable, haciendo que el problema se asemeje más a un problema de aprendizaje supervisado [15].

Curiosity

Los algoritmos de aprendizaje por refuerzo apuntan a políticas de aprendizaje para lograr tareas objetivo al maximizar las recompensas proporcionadas por el entorno. En algunos escenarios, estas recompensas se entregan al agente de forma continua, por ejemplo, la puntuación de ejecución en un juego de Atari, o la distancia entre un brazo robótico y un objeto en una tarea de alcance. Sin embargo, en muchos escenarios del mundo real, las recompensas extrínsecas al agente son extremadamente escasas o faltan por completo, y no es posible construir una función de recompensa adecuada. Esto es un problema ya que el agente recibe refuerzo por actualizar su política solo si logra alcanzar un estado objetivo pre especificado. Es probable que la esperanza de tropezar con un estado objetivo por casualidad (es decir, exploración aleatoria) sea inútil para todos los entornos, excepto para los más simples. Por lo tanto, la motivación/recompensas intrínsecas obtienen una importancia crítica cuando las recompensas extrínsecas son escasas [16].

La mayoría de las formulaciones de recompensa intrínseca se pueden agrupar en dos amplias clases: 1) alentar al agente a explorar estados "nuevos" [17] o, 2) alentar al agente a realizar

---

[2] Los algoritmos de aprendizaje por refuerzo utilizan búferes de reproducción(replay buffer) para almacenar trayectorias de experiencia al ejecutar una política en un entorno. Durante el entrenamiento, se consultan los búferes de reproducción para un subconjunto de las trayectorias (ya sea un subconjunto secuencial o una muestra) para "reproducir" la experiencia del agente [14]

acciones que reduzcan el error/incertidumbre en la capacidad del agente para predecir las consecuencias de sus propias acciones (es decir, su conocimiento sobre el medio ambiente) [18].

El papel de la curiosidad ha sido ampliamente estudiado en el contexto de la resolución de tareas con escasas recompensas. La curiosidad tiene otros dos usos fundamentales: ayuda a un agente a explorar su entorno en la búsqueda de nuevos conocimientos (una característica deseable del comportamiento exploratorio es que debe mejorar a medida que el agente adquiere más conocimiento). Además, la curiosidad es un mecanismo para que un agente aprenda habilidades que podrían ser útiles en escenarios futuros [16].

Self-Play

Con el Self-play, un agente aprende en juegos de confrontación compitiendo contra versiones fijas pasadas de su oponente (que podría ser él mismo como en los juegos simétricos) para proporcionar un entorno de aprendizaje estacionario más estable. Para ello se tiene un *buffer* que almacena copias de versiones pasadas del propio agente, con políticas aprendidas en diversos momentos. La cantidad de versiones almacenadas por el buffer tiene un límite determinado y cada cierta cantidad de pasos de tiempo se eliminan las copias más antiguas, siendo reemplazadas por las nuevas versiones, constituyendo estas los nuevos oponentes durante un tiempo determinado; esto se compara con competir contra el mejor oponente actual en cada episodio, que cambia constantemente (porque está aprendiendo). Sin embargo, si se usaran copias exactas del agente, con sus políticas actualizándose constantemente, el entrenamiento se volvería inestable debido a la naturaleza cambiante de los oponentes; de ahí el uso de versiones pasadas fijas, logrando, como se menciona anteriormente, un aprendizaje más estable [13].

1.1.3 Aprendizaje Profundo

El aprendizaje profundo (*deep learning*) es una clase de algoritmos de aprendizaje automático que utiliza múltiples capas para extraer progresivamente características de nivel superior de la entrada sin procesar [24]. En esta categoría suelen agruparse algoritmos que utilizan redes neuronales como ANN de una gran cantidad de capas, u otras arquitecturas de red neuronal como las anteriormente discutidas CNN y RNN. El *deep learning* y las redes neuronales tienden a utilizarse indistintamente en la conversación, lo que puede ser confuso. Vale la pena señalar que "*deep*" en *deep learning* solo hace referencia a la profundidad de las capas en una red neuronal. Una red neuronal que consta de más de tres capas (incluidas las entradas y la salida) puede considerarse un algoritmo de *deep learning*. Una red neuronal que solo tiene dos o tres capas es una red neuronal básica [21].

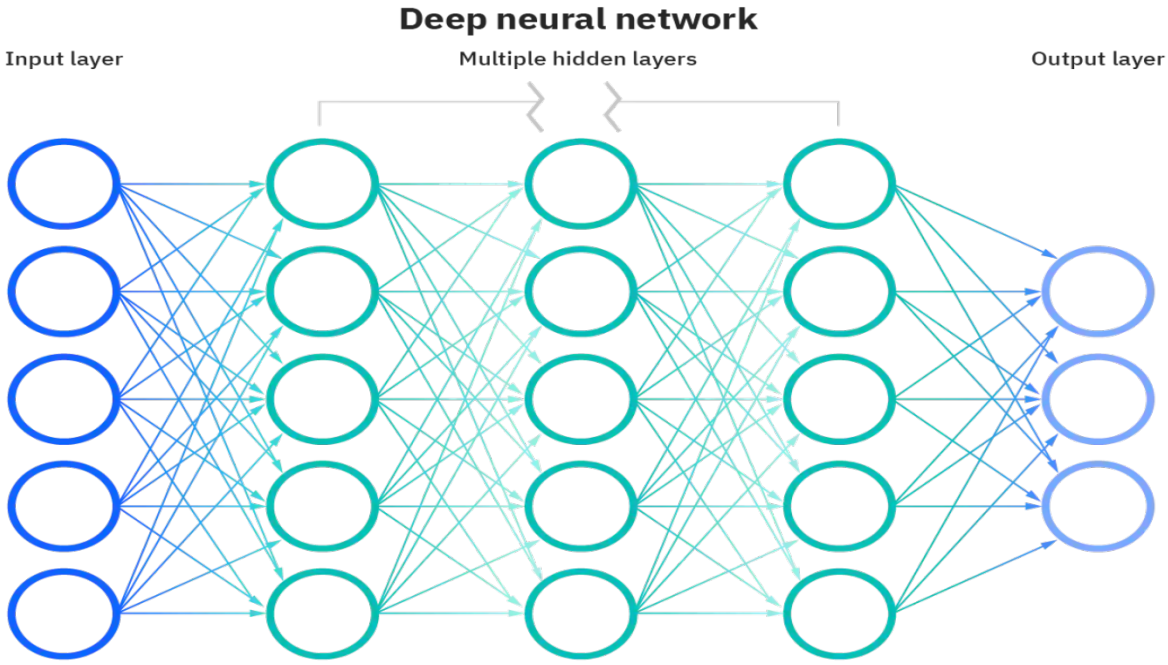


Figura 1. Red Neuronal Profunda

1.1.4 Deep Reinforcement Learning

El Aprendizaje por Refuerzo Profundo *(Deep Reinforcement Learning*) es la combinación del Aprendizaje por Refuerzo con técnicas de Aprendizaje Profundo para resolver problemas desafiantes de toma de decisiones secuenciales. El uso del aprendizaje profundo es más útil en problemas con espacio de estado de alta dimensión. Esto significa que, con el aprendizaje profundo, el aprendizaje por refuerzo puede resolver tareas más complicadas con un conocimiento previo más bajo debido a su capacidad para aprender diferentes niveles de abstracción de los datos.

Sin embargo, para usar el aprendizaje por refuerzo con éxito en situaciones que se acerquen a la complejidad del mundo real, los agentes se enfrentan a una tarea difícil: deben derivar representaciones eficientes del entorno a partir de entradas sensoriales de alta dimensión y usarlas para generalizar experiencias pasadas a nuevas situaciones. Esto hace posible que las máquinas imiten algunas capacidades humanas de resolución de problemas, incluso en un espacio de alta dimensión, que hace solo unos años era difícil de concebir [25]. Cabe destacar que, aunque los algoritmos de *deep reinforcement learning* se basan en *deep learning* para su concepción, estos no necesitan que la red neuronal tenga docenas de *hidden layers*; incluso teniendo pocos de estos, por la propia arquitectura de las redes usualmente utilizadas, tanto CNN como RNN u otras, y los algoritmos usados, son considerados algoritmos de *deep reinforcement learning*.

Se decide que la utilización de algoritmos de aprendizaje reforzado con redes neuronales es adecuado como solución, estos tienen un mejor escalado que los clásicos como *Q-Learning*, la red

neuronal resultante necesita poco espacio de almacenamiento, y existen varios paquetes y bibliotecas que permiten el uso con relativa facilidad de redes neuronales y de algoritmos de aprendizaje reforzado que las utilicen, así como el entrenamiento en paralelo de los agentes.

1.1.5 Redes Neuronales

Las redes neuronales, también conocidas como redes neuronales artificiales (artificial neural network o ANN) son llamadas así porque son una red de elementos interconectados. Estos elementos se inspiraron en estudios de sistemas nerviosos biológicos. En otras palabras, las redes neuronales son un intento de crear máquinas que funcionen en una manera similar al cerebro humano al construir estas máquinas usando componentes que se comportan como neuronas biológicas. La función de una red neuronal es producir un patrón de salida cuando se presenta con un patrón de entrada [19]. Una red neuronal artificial consta de una capa de neuronas (o nodos, unidades) de entrada, una o dos (o incluso tres) capas ocultas de neuronas, y una capa final de neuronas de salida [20].

Las redes neuronales se basan en entrenar datos para aprender y mejorar su precisión con el tiempo. No obstante, una vez que estos algoritmos de aprendizaje se ajustan de manera precisa, son potentes herramientas de informática e inteligencia artificial, lo que nos permite clasificar y agrupar los datos a una alta velocidad. Las tareas de reconocimiento de voz o reconocimiento de imagen pueden tardar minutos frente a las horas que requiere la identificación manual de expertos humanos. Una de las redes neuronales más conocidas es el algoritmo de búsqueda de Google [21].

1.1.5.1 Convolutional Neural Networks

Las redes neuronales convolucionales, en inglés *convolutional neural networks* (CNN), son análogas a las ANN tradicionales en que están compuestos por neuronas que se auto-optimizan a través del aprendizaje. Cada neurona aún recibirá una entrada y realizará una operación (como un producto escalar seguido de una función no lineal) - la base de innumerables ANN. Desde los vectores de imagen sin procesar de entrada a la salida final de la puntuación de la clase, la totalidad de la red seguirá expresando una única función de puntuación perceptiva (el peso). La última capa contendrá las funciones asociadas a las clases, y todas los consejos y trucos regulares desarrollados para ANN tradicionales aún se aplican [22].

Las ANN sin embargo vienen con problemas inherentes a su diseño, se puede pensar que para resolver cualquier problema bastaría con incrementar el número de capas ocultas o quizás el número de neuronas en cada una de ellas; sin embargo, existe el problema de no tener poder computacional y tiempo ilimitados para entrenar estas enormes ANN. La segunda razón es detener o reducir los efectos del *overfitting*. El *overfitting* es básicamente cuando una red no puede

aprender de manera efectiva debido a una serie de razones. Es un concepto importante de la mayoría, sino de todos, los algoritmos de aprendizaje automático y es importante que se tomen todas las precauciones para reducir sus efectos. Si los modelos mostraran signos de overfitting, es posible que se vea una reducción en la capacidad de identificar características generalizadas no solo para el conjunto de datos de entrenamiento, sino también los conjuntos de prueba y predicción. Esta es la razón principal detrás de la reducción de la complejidad de nuestras ANN. Los menos parámetros requeridos para entrenar, es menos probable que la red sufra de overfitting - y por supuesto supuesto, mejorar el rendimiento predictivo del modelo [22].

Las CNN se centran principalmente en la base de que la entrada estará compuesta por imágenes. Esto enfoca la arquitectura en que se configurará de manera que mejor se adapte a la necesidad de tratar con el tipo específico de datos. Una de las diferencias clave es que las neuronas que forman las capas dentro de la CNN están compuestas por neuronas organizadas en tres dimensiones, la dimensionalidad espacial de la entrada (alto y ancho) y la profundidad. La profundidad no se refiere al número total de capas dentro de la ANN, sino a la tercera dimensión de un volumen de activación. A diferencia de una ANN estándar, las neuronas dentro de cualquier capa dada solo se conectarán a una pequeña región de la capa que la precede [22].

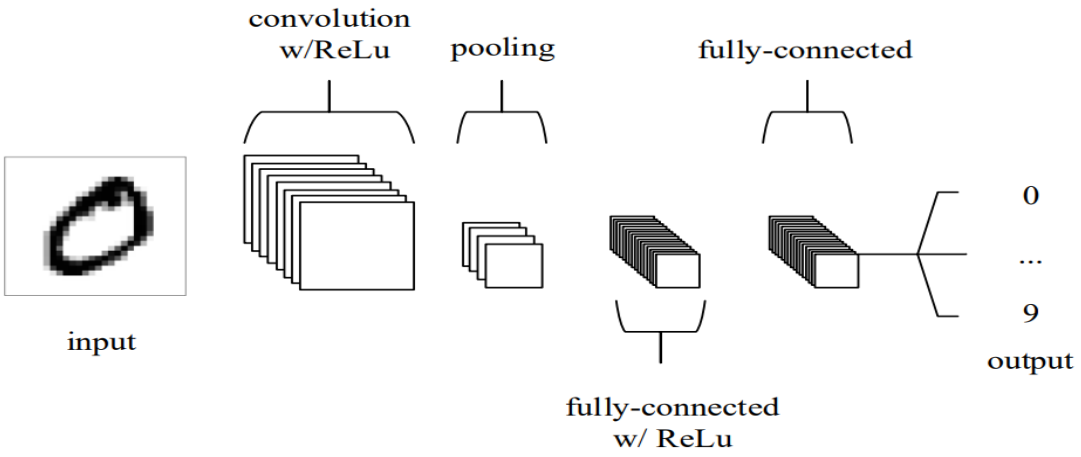


Figura 2. Una arquitectura CNN simple, compuesta de solo cinco capas, tomado de [22].

La funcionalidad básica del ejemplo CNN anterior se puede dividir en cuatro áreas clave.

1. Como se encuentra en otras formas de ANN, la capa de entrada contendrá los valores de los píxeles de la imagen.
2. La capa convolucional (***convolutional layer***) determinará la salida de las neuronas, las cuales son conectadas a regiones locales de la entrada, a través del cálculo del producto escalar entre sus pesos y la región conectada al volumen de entrada. La unidad lineal rectificada (comúnmente abreviada como ReLu) tiene como objetivo aplicar una función de activación 'orientada a elementos' como sigmoide a la salida de la activación producida por la capa anterior.

3. La capa de agrupación (***pooling layer***) simplemente realizará una reducción de muestreo a lo largo de la dimensionalidad espacial de la entrada dada, reduciendo aún más la cantidad de parámetros dentro de esa activación.
4. Las capas completamente conectadas (***fully-connected layers***) realizarán las mismas tareas que se encuentran en ANN estándar e intentar producir puntajes de clase a partir de las activaciones, que se utilizará para la clasificación. También se sugiere que ReLu puede usarse entre estas capas, para mejorar el rendimiento.

Las redes neuronales convolucionales se diferencian de otras formas de redes neuronales artificiales en que, en lugar de centrarse en la totalidad del dominio del problema, se explota el conocimiento sobre el tipo específico de entrada. Esto, a su vez, permite configurar una arquitectura de red mucho más sencilla [22].

Las CNN operan sobre imágenes, que podemos definir vagamente como una colección de objetos o entidades en relación espacial entre sí. Una CNN asume que se puede decir que una entidad está en una dirección de otra. Además, es posible usar CNN en cualquier tipo de datos, pero se recomienda usar CNN solo en datos que tienen características espaciales. En la solución propuesta, el ser un videojuego, la entrada de datos a la red pueden ser imágenes o bien datos de los objetos en pantalla, como su posición en los ejes X, Y y Z o su velocidad, siendo estos datos con características espaciales aptas para el uso de una CNN.

1.1.5.2 Recurrent Neural Network

Una red neuronal recurrente, en inglés *recurrent neural network* (RNN), es un tipo de red neuronal artificial que utiliza datos secuenciales o datos de series temporales. Estos algoritmos de aprendizaje profundo se usan comúnmente para problemas ordinales o temporales, como traducción de idiomas, procesamiento de lenguaje natural (NLP), reconocimiento de voz y subtítulos de imágenes; se incorporan a aplicaciones populares como Siri, búsqueda por voz y Google Translate. Al igual que las redes neuronales convolucionales (CNN) y de feed-forward, las redes neuronales recurrentes utilizan datos de entrenamiento para aprender. Se distinguen por su "memoria", ya que toman información de entradas anteriores para influir en la entrada y salida actual: mientras que las redes neuronales profundas tradicionales asumen que las entradas y salidas son independientes entre sí, la salida de las redes neuronales recurrentes depende de los elementos previos dentro de la secuencia. Si bien los eventos futuros también serían útiles para determinar el resultado de una secuencia determinada, las redes neuronales recurrentes unidireccionales no pueden tener en cuenta estos eventos en sus predicciones [23].

Una característica distintiva de las redes recurrentes es que comparten parámetros en cada capa de la red. Mientras que las redes feedforward tienen diferentes pesos en cada nodo, las redes neuronales recurrentes comparten el mismo parámetro de peso dentro de cada capa de la red.

Dicho esto, estos pesos aún se ajustan a través de los procesos de retropropagación y descenso de gradiente para facilitar el aprendizaje por refuerzo [23].

Las redes neuronales recurrentes aprovechan el algoritmo de retropropagación a través del tiempo (BPTT) para determinar los gradientes, que es ligeramente diferente de la retropropagación (*backpropagation*) tradicional, ya que es específico de los datos de secuencia. Los principios de BPTT son los mismos que los de la retropropagación tradicional, donde el modelo se entrena a sí mismo calculando errores desde su capa de salida hasta su capa de entrada. Estos cálculos permiten ajustar los parámetros del modelo adecuadamente. BPTT difiere del enfoque tradicional en que BPTT suma los errores en cada paso de tiempo, mientras que las redes *feedforward* no necesitan sumar errores ya que no comparten parámetros en cada capa [23].

A través de este proceso, las RNN tienden a encontrarse con dos problemas, conocidos como explosión de gradientes y desaparición de gradientes. Estos problemas se definen por el tamaño del gradiente, que es la pendiente de la función de pérdida a lo largo de la curva de error. Cuando el gradiente es demasiado pequeño, continúa haciéndose más pequeño, actualizando los parámetros de peso hasta que se vuelven insignificantes, es decir, 0. Cuando eso ocurre, el algoritmo ya no está aprendiendo. La explosión de gradientes se produce cuando el gradiente es demasiado grande, lo que crea un modelo inestable. En este caso, los pesos del modelo crecerán demasiado y finalmente se representarán como NaN [23].

Si bien la capacidad que otorgan las RNN a los algoritmos de tener memoria es una herramienta poderosa en algunos problemas, en este caso no se hace necesario su uso. Se pudo constatar además en el subepígrafe anterior que las CNN se ajustan a las características que se buscan para el agente a desarrollar, por tanto, una CNN será la red neuronal utilizada en la solución.

1.1.6 Policy Gradient Methods

Los métodos de gradiente de políticas son un tipo de técnicas de aprendizaje por refuerzo que se basan en la optimización de políticas parametrizadas con respecto al rendimiento esperado (recompensa acumulada a largo plazo) por descenso de gradiente[3] (*gradient descent*). Para lograr esto, se asume que, en cada paso de tiempo K, las acciones están generadas por una política modelada como una distribución probabilística con el objetivo de incorporar acciones de exploración. De dicha política se asume que está parametrizada por K, con parámetros de política θ. La meta general de la optimización de política es optimizar los parámetros de política de manera que la recompensa esperada esté optimizada para cada factor de peso dependiente de pasos de tiempo.

[3] El descenso de gradiente (GD) es un algoritmo iterativo de optimización de primer orden que se utiliza para encontrar un mínimo/máximo local de una función dada. Este método se usa comúnmente en aprendizaje automático (ML) y aprendizaje profundo (DL) para minimizar una función de costo/pérdida (por ejemplo, en una regresión lineal) [26]

Para las aplicaciones del mundo real y algunas otras tareas complejas, se requiere que cualquier cambio en la parametrización de la política sea suave, ya que los cambios drásticos pueden ser peligrosos para el actor, y las inicializaciones útiles de la política basadas en el conocimiento del dominio desaparecerían después de un único paso de actualización. Por estas razones, los métodos de gradiente de política que siguen el descenso más pronunciado en el rendimiento esperado son el método de elección. Estos métodos actualizan la parametrización de la política de acuerdo con la regla de actualización de gradiente:

$$\theta_{h+1}=\theta_h+\alpha_h\nabla_\theta J\Big|_{\theta=\theta_h}$$

donde $\alpha_h\in R_+$ denota la tasa de aprendizaje y $h\in\{0,1,2,...\}$ el número de actualización actual. El problema principal en los métodos de gradiente de política es obtener un buen estimador para el gradiente de política $\nabla_\theta J\Big|_{\theta=\theta_h}$.

En la mayoría de problemas a los que están enfocados los *policy gradient methods*, es imposible esperar que se pueda modelar todos y cada uno de los detalles del sistema, por tanto se necesita estimar el gradiente de política simplemente a partir de los datos generados durante la ejecución de una tarea, es decir, sin necesidad de un modelo [27].

La mayoría de los métodos clásicos de aprendizaje reforzado no son aplicables a muchos problemas como la robótica, el control motor, etc. Esta inaplicabilidad puede deberse a problemas con información de estado incierto. Por lo tanto, esos sistemas deben modelarse como problemas de decisión de Markov parcialmente observables, lo que a menudo resulta en demandas computacionales excesivas. La mayoría de los métodos tradicionales de aprendizaje por refuerzo no tienen garantías de convergencia e incluso existen ejemplos de divergencia. Los estados y acciones continuos en espacios de alta dimensión no pueden ser tratados por la mayoría de los enfoques clásicos de aprendizaje por refuerzo [27].

Los métodos de gradiente de políticas difieren significativamente ya que no sufren estos problemas de la misma manera. Por ejemplo, la incertidumbre en el estado podría degradar el rendimiento de la política (si no se utiliza un estimador de estado adicional), pero no es necesario cambiar las técnicas de optimización para la política. Los estados y acciones continuos pueden tratarse exactamente de la misma manera que los discretos mientras que, además, el rendimiento del aprendizaje suele aumentar. Se garantiza la convergencia al menos a un óptimo local [27].

Las ventajas de los métodos de gradiente de políticas para aplicaciones del mundo real son numerosas. Entre los más importantes se encuentran que las representaciones de políticas se pueden elegir de modo que sean significativas para la tarea y puedan incorporar conocimiento del dominio, que a menudo se necesitan menos parámetros en el proceso de aprendizaje que en los enfoques basados en funciones de valor y que hay una variedad de diferentes algoritmos para la

estimación del gradiente de políticas en la literatura que tienen una base teórica bastante sólida. Además, los métodos de gradiente de políticas se pueden utilizar sin modelo o basados en modelo, ya que son una formulación genérica [27].

A pesar de sus ventajas, obtener buenos resultados a través de métodos de gradiente de políticas es un desafío porque son sensibles a la elección del tamaño del paso: demasiado pequeño y el progreso es irremediablemente lento; demasiado grande y la señal se ve abrumada por el ruido, o se pueden ver caídas importantes en el rendimiento; es muy difícil encontrar una tasa de aprendizaje adecuada para todo el proceso de optimización. También suelen tener una eficiencia de muestra muy baja, y requieren millones (o miles de millones) de pasos de tiempo para aprender tareas simples [28].

1.2 Videojuegos

Es imposible hoy por hoy hablar de IA sin mencionar a los videojuegos, una industria que ha crecido inmensamente desde sus inicios, llegando al punto en que, a partir de 2020, el mercado global de videojuegos tiene ingresos anuales estimados de 159 mil millones de dólares americanos en hardware, software y servicios. Esto es tres veces el tamaño de la industria musical global de 2019 y cuatro veces el de la industria cinematográfica de 2019 [29].

Un videojuego puede definirse como un tipo de juego que existe como software y está controlado por este, generalmente ejecutado por una consola de videojuegos o una computadora, y se juega en una terminal de video o pantalla de televisión. Controlado por una paleta, joystick[4], joypad[5], mouse, teclado o una combinación de cualquiera de estos dispositivos de entrada [30].

Por otro lado, los videojuegos están determinados por reglas, a pesar de que el jugador pueda creer que tiene total libertad en realidad se está rigiendo por las reglas del juego. Se puede decir entonces que un videojuego es un juego electrónico que recrea una situación ficticia en la cual el jugador puede interactuar, sometido a determinadas reglas, para alcanzar un objetivo específico [31].

1.2.1 IA en Videojuegos

Como fue mencionado anteriormente, es imposible abordar el tema de los videojuegos sin mencionar la inteligencia artificial, y es que, en pos de crear escenarios interactivos, retos, contar una historia y otros aspectos importantes dentro de un juego, es necesario la presencia de

---

[4] Un **joystick** es un instrumento que se emplea para controlar un sistema o un dispositivo. Por lo general un **joystick** es una palanca que cuenta con una base y que puede realizar una cierta gama de movimientos. (Tomado de: https://definicion.de)

[5] dispositivo de entrada portátil utilizado en los videojuegos para controlar el movimiento de los elementos gráficos en la pantalla, generalmente con botones y un control direccional. (Tomado de https://www.dictionary.com)

personajes no jugador (PNJ), los cuales dan la ilusión de inteligencia mediante el uso de IA, Pero el aplicativo de la IA en los videojuegos va más allá del concepto humano vs máquina, también se aplica a la hora de “crear” mapas en los llamados juegos procedurales[6].

Muchos videojuegos contemporáneos entran en la categoría de acción, juegos de disparos en primera persona (*first person shooter*), o aventura. En la mayoría de estos tipos de juegos hay un cierto nivel de combate que se lleva a cabo, siendo la capacidad de la IA para ser eficiente en el combate importante en estos géneros. Un objetivo común hoy en día es hacer que la IA sea más humana, o al menos parecerlo.

Una de las características más positivas y eficaces que se encuentran en las IA modernas es la habilidad para cazar. La IA originalmente reaccionaba de una manera muy simple: si el jugador se encontraba en un área específica, a continuación, la IA reaccionaba, ya sea en una manera totalmente ofensiva o totalmente defensiva. En los últimos años, la idea de "caza" se ha introducido. En esta la IA buscará marcadores realistas, tales como los sonidos hechos por el personaje o huellas que pudieron haber dejado atrás.

Estos desarrollos en la IA permiten que se desarrollen formas más complejas de juego, pues con esta característica, el jugador puede realmente considerar la manera de acercarse o evitar a un enemigo. Esta es una función que es particularmente frecuente en el género del sigilo.

Otra novedad en la IA de los juegos recientes ha sido el desarrollo de "instinto de supervivencia". El ordenador puede reconocer diferentes objetos en un ambiente y determinar si es beneficioso o perjudicial para su supervivencia. Al igual que un usuario, la IA puede "mirar" para cubrirse en un tiroteo antes de tomar acciones que podrían dejarla vulnerable, tales como la recarga de un arma o lanzar una granada. Se puede establecer marcadores que le indican cuándo reaccionar de una manera determinada. Por ejemplo, si la IA se utiliza un comando para comprobar su salud a través de un juego, se puede configurar de manera que reaccione de forma específica a un cierto porcentaje de la salud: si la salud está por debajo de cierto umbral entonces la IA puede ser configurada para ir lejos del jugador y evitarlo hasta que otra función se active. Otro ejemplo podría ser si la IA se da cuenta de que se ha quedado sin balas, encuentra un objeto que le dé cobertura y se esconde detrás de ella hasta que se recargue. Acciones como estas hacen que la IA parezca más humana, sin embargo, todavía hay una necesidad de mejora en este ámbito. A diferencia de un jugador humano, la IA normalmente debe ser programada para todos los escenarios posibles. Esto limita seriamente su capacidad para sorprender al jugador.

Algunas de las estrategias y técnicas más usadas son:

- Comportamientos de locomoción: persiguen lograr tareas básicas de movimiento en el entorno. Uno de los comportamientos más utilizados es el de capturar y huir. Es muy sencillo pues persigue

[6] Los juegos procedurales o de generación procedural son aquellos donde el motor del videojuego crea contenido dentro del mismo juego sin la necesidad de interactuar con un equipo de creadores o personas en todo momento, mediante algoritmos

solo el objetivo de capturar al jugador y huir de él. Primeramente, la IA debe decidir si está en condiciones de perseguir al jugador y luego lleva a cabo la acción mediante una estrategia que se trace.

- Búsqueda de caminos: es una técnica clásica de IA. se utiliza para determinar el camino a seguir de un punto a otro teniendo en cuenta las características del escenario.

- Redes neuronales: sigue el modelo de las redes neuronales biológicas pues se compone de entradas a la red, salidas correspondientes y neuronas que conectan las entradas y salidas.

- Máquinas de estados finitos: son sistemas formados por un conjunto de diferentes estados y las transiciones entre cada uno de ellos producidas por eventos.

- Agentes inteligentes: sistema capaz de percibir su entorno y actuar en consecuencia. Su comportamiento se determina por la percepción, planificación y actuación.

-Toma de decisiones: a partir de determinadas condiciones el sistema debe tomar la mejor decisión para lograr el objetivo del juego. Para esto se debe introducir determinadas sentencias en cada caso que podría ocurrir dentro del mundo del videojuego [32].

1.2.2 Aprendizaje Profundo en videojuegos

La mayoría de las personas probablemente imagina que la mayoría de los juegos lanzados en los últimos años tienen una Inteligencia Artificial muy sofisticada para cualquier personaje, criatura o animal no controlado por el jugador. Sin embargo, muchos desarrolladores de videojuegos dudan en incorporar una Inteligencia Artificial avanzada en sus juegos por temor a perder el control de la experiencia general del jugador. De hecho, el objetivo de la Inteligencia Artificial en los videojuegos no es crear una entidad imbatible contra la que los jugadores pueden luchar, sino que se trata de maximizar la participación y el disfrute de los jugadores durante largos periodos de tiempo. Más aún, lo que es notable es que la Inteligencia Artificial que en los juegos de hoy en día ha permanecido inalterada a lo largo del tiempo. Dos de los componentes principales de estas inteligencias son las máquinas de búsqueda y las maquinas de estados finitos, a pesar de que algunos incursionan en otro tipo de técnicas [33].

Sin embargo, los videojuegos constituyen un escenario perfecto para la utilización de técnicas de aprendizaje reforzado: poseen una gran cantidad de estados y variables, además son entornos ya programados y controlables, donde se puede investigar fácilmente los resultados luego de aplicar algún algoritmo específico. Esto los ha convertido en un perfecto escenario para los investigadores que desean incursionar y profundizar en el campo del aprendizaje reforzado. Si bien hay varios algoritmos usados para jugar juegos utilizando aprendizaje reforzado para lograr resultados comparables o superiores a los de un humano, no hay muchos juegos que en su implementación utilicen aprendizaje reforzado en la actualidad.

1.3 Aplicación de IA y RF en videojuegos desarrollados en VERTEX

El Centro de Tecnologías Interactivas VERTEX presenta entre sus líneas de desarrollo la realización de videojuegos. Según datos consultados por el centro, tres de los videojuegos realizados han implementado comportamientos inteligentes, estos son:

- Kuba Kart: videojuego de carrera donde el jugador deberá competir en distintas pistas contra varios oponentes para obtener el primer lugar. Los oponentes llegarán a la meta guiados por una serie de *way points* que le indicarán el camino a seguir. Para evitar salirse de la pista cuentan con varios Raycast que les indican cuando se desvían. El juego presenta además un mecanismo para los poderes que se encuentran a lo largo de la carrera que tanto el jugador como los no jugadores pueden obtener. Si el jugador se halla entre los últimos lugares el poder que obtendrá será bastante potente y lo contrario sucede con los que se encuentran ganando la carrera.
- Especies invasoras: es un videojuego de estrategia que tiene como objetivo recuperar todo el terreno de islas Casabe y desplazar las especies invasoras del territorio que han ocupado. En el juego se encuentran las casas que ocupa el jugador, terrenos neutrales y casas ocupadas por el enemigo. El enemigo tendrá el mismo objetivo del jugador, expandir su territorio, por lo que enviará unidades a los terrenos neutrales y a las casas del jugador para luchar.
- Coliseum: videojuego de lucha donde el jugador deberá enfrentarse a varios enemigos con el fin de proteger a la tierra de una futura invasión inter-dimensional. Para esto se implementaron dos niveles de IA, el primero consiste en que el oponente perseguirá y atacará al jugador. El segundo nivel tiene el mismo objetivo que el primero, pero tendrá en cuenta el nivel de vida. Si el jugador tiene mayor nivel de vida el oponente deberá buscar primero una gema de vida y luego luchar. Se está trabajando actualmente con el desarrollo de un tercer nivel de IA en el que varios enemigos realizarán ataque en conjunto con un plan determinado para acabar de forma más rápida con el jugador. También los no jugadores cuentan con una máquina de estado para perseguir, atacar o buscar gema de vida [34].

El centro VERTEX ha desarrollado gran cantidad de videojuegos, sin embargo, pocos cuentan con mecanismos que presentan IA. Se evidencian comportamientos inteligentes, pero no una realización sólida de las técnicas de IA para videojuegos. El uso de técnicas de IA brindará mayor calidad, realismo y garantizará que el juego exija un mayor esfuerzo del jugador para ganarlo. Ejemplo de esto es el éxito que ha tenido el videojuego Coliseum el cual hasta el momento es en el que se ha implementado de una mejor forma la IA [34].

Sin embargo, no ha implementado aún ningún videojuego que utilice técnicas de aprendizaje reforzado.

1.4 Análisis de homólogos

Si bien es un campo relativamente nuevo y aun en desarrollo, la comunidad internacional ha demostrado gran interés en la creación de jugadores virtuales capaces de lograr un desempeño igual o mayor que los jugadores humanos más prominentes, a continuación, se describen agentes que utilizan técnicas de aprendizaje reforzado profundo para estos fines:

Agent57: El conjunto de juegos Atari57 es un punto de referencia de large-data para medir el desempeño de los agentes en una amplia gama de tareas. DeepMind desarrolló Agent57, el primer agente de aprendizaje de refuerzo profundo que obtiene una puntuación superior a la línea de base humana en los 57 juegos de Atari 2600. Agent57 combina un algoritmo para una exploración eficiente con un metacontrolador que adapta la exploración y el comportamiento a corto y largo plazo del agente [35].

OpenAi Five: esta IA utilizó el videojuego multijugador Dota 2 como plataforma de investigación para sistemas de IA de propósito general. La IA desarrollada por OpenAI de Dota 2, llamada OpenAI Five, aprendió jugando más de 10 000 años contra sí misma. Demostró la capacidad de lograr un desempeño de nivel experto, aprender la cooperación humano-IA y operar a escala de Internet, siendo la primera IA en poder vencer a un equipo de campeones mundiales [36].

AlphaStar: AlphaStar es la primera IA en llegar a la liga superior de un esport[7] muy popular sin restricciones de juego. En enero de 2019, una versión preliminar de AlphaStar desafió a dos de los mejores jugadores del mundo en StarCraft II, uno de los videojuegos de estrategia en tiempo real más duraderos y populares de todos los tiempos. Desde entonces, DeepMind, sus creadores, aceptaron un desafío mucho mayor: jugar el juego completo a nivel de Gran Maestro en condiciones aprobadas profesionalmente. Para noviembre de 2019, AlphaStar se clasificó por encima del 99,8 % de los jugadores activos en Battle.net y alcanzó el nivel de *Grandmaster* (Gran Maestro) en las tres razas de StarCraft II: protoss, terran y zerg [37].

Luego del análisis de las aplicaciones descritas anteriormente se pueden arribar a varias conclusiones: primeramente, son IAs muy enfocadas en un tipo específico de juegos, desarrolladas por grandes empresas con el fin de romper las barreras sobre el aprendizaje reforzado. El caso de Agent57 es un ejemplo potente, pero solo disponible para los juegos de Atari 2600, al igual que OpenAI Five y AlphaStar se ven limitadas a Dota 2 y StarCraft II respectivamente. Teniendo en cuenta que el centro VERTEX produce mayormente utilizando el motor gráfico Unity, y que las prestaciones de las PC disponibles no pueden alcanzar el rendimiento necesario para IAs dedicadas como las anteriores, se constata que, si bien el aprendizaje reforzado profundo parece prometedor, las soluciones actuales no son apropiadas como una solución a la problemática planteada. Por tanto, se determina implementar un agente que utilice aprendizaje reforzado en

[7] El término eSports viene de "*Electronic Sports*", que en español significa "Deportes Electrónicos". El concepto se usa para denominar a todas las competiciones organizadas en los videojuegos, especialmente entre profesionales.

Unity, que no sea costoso en requerimientos del sistema a la hora de entrenar en un entorno en forma del demo a desarrollar.

1.5 Algoritmos de aprendizaje reforzado profundo

En la siguiente sección se analizarán diversos algoritmos, de los cuales se tomará uno que será la base de la implementación de la propuesta de solución. Se tocarán algunos con conocido desempeño y gran cantidad de pruebas en entornos de videojuegos, y con implementaciones conocidas y robustas, tomándose en cuenta estos aspectos y una relativa baja complejidad de implementación y requerimientos para la selección final del algoritmo a utilizar.

1.5.1 Deep Q-Network

Deep Q-Network (DQN) puede interpretarse como una extensión del algoritmo clásico de Q-learning, que utiliza redes neuronales profundas para aproximar la función acción-valor(Q-values) en lugar de una tabla de políticas (Q-table) [43].

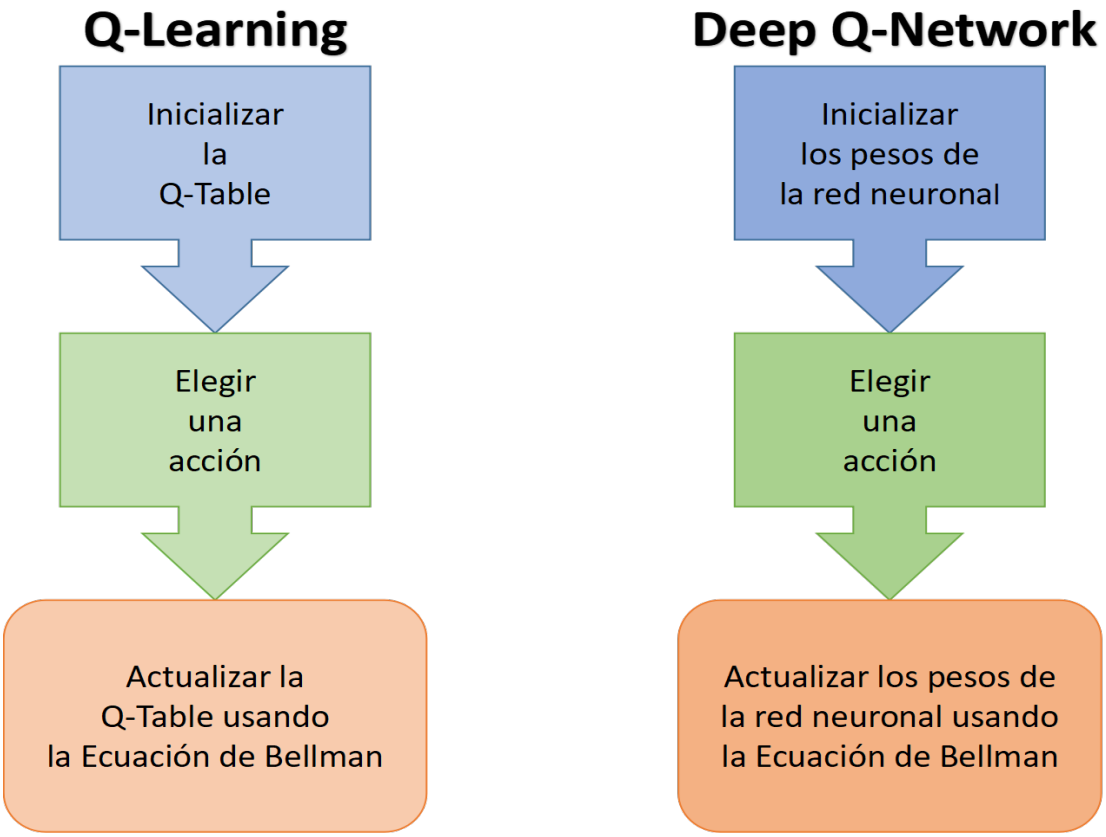


Figura 3. Comparación entre Q-Learning y DQN. Elaborado por el autor.

Una diferencia fundamental entre Deep Q-Learning y Q-Learning es la implementación de la Q-table. Fundamentalmente, Deep Q-Learning reemplaza la *Q-table* regular con una red neuronal.

En lugar de asignar un par estado-acción a un *Q-value*, una red neuronal asigna estados de entrada a pares (acción, *Q-value*).
Una de las cosas interesantes de Deep Q-Learning es que el proceso de aprendizaje usa 2 redes neuronales. DQN usa otra red neuronal llamada red objetivo para obtener un estimador imparcial del error cuadrático medio de Bellman usado en el entrenamiento de la red Q. La red de destino se sincroniza con la red Q después de cada período de iteraciones, lo que conduce a un acoplamiento entre las dos redes[43]. El uso de ambas redes conduce a una mayor estabilidad en el proceso de aprendizaje y ayuda al algoritmo a aprender de manera más efectiva.

Una estrategia común para abordar el equilibrio entre exploración y explotación al elegir los próximos pasos es la estrategia de exploración codiciosa de Epsilon, en la que el agente elige una acción aleatoria con probabilidad épsilon y explota la acción más conocida con probabilidad 1-épsilon. Para conocer la mejor acción de la red, tanto el modelo principal como el modelo objetivo asignan estados de entrada a acciones de salida. Estas acciones de salida en realidad representan el valor Q previsto del modelo. En este caso, la acción que tiene el mayor valor Q predicho es la acción mejor conocida en ese estado [44].

En Q-Learning clásico, la Ecuación de Bellman dicta cómo actualizar la Q-table después de cada paso que da el agente. Para resumir esta ecuación, el agente actualiza el valor percibido actual con la recompensa futura óptima estimada, lo que supone que realice la mejor acción conocida actual. En una implementación, el agente buscará en todas las acciones un estado en particular y elegirá el par estado-acción con el Q-value correspondiente más alto.

$$Q(S_t, A_t) = (1 - \alpha)\, Q(S_t, A_t) + \alpha * (R_t + \lambda * max_a\, Q(S_{t+1}, a))$$

Figura 4. La Ecuación de Bellman indica como actualizar la Q-Table

S = el Estado o la Observación

A = la Acción que toma el agente

R = la Recompensa por tomar una Acción

t = el paso de tiempo

α = el ratio de aprendizaje

λ = el factor de descuento que causa que las recompensar vayan perdiendo valor a lo largo del tiempo, de modo que recompensas inmediatas sean más valiosas

En DQN se utiliza la Ecuación de Bellman para actualizar los pesos de la red neuronal. A partir de la ecuación original de Bellman, se desea replicar la operación de destino de diferencia temporal usando la red neuronal en lugar de usar una tabla Q.

$$(R_t + \lambda * max_a Q(S_{t+1}, a))$$

Figure 1 Diferencia temporal

Se debe tener en cuenta que la red de destino y no la red principal se utiliza para calcular el objetivo de diferencia temporal. Posteriormente el valor obtenido de diferencia temporal se usa en la red principal para reemplazar el q-value obtenido para actualizar los pesos [44].

Un elemento clave en el desempeño del algoritmo DQN es el experience replay. En cada iteración de DQN, un minilote de estados, las acciones, las recompensas y los siguientes estados se muestrean de la memoria de repetición como observaciones para entrenar al Q-network, que aproxima la función acción-valor. La intuición detrás de la repetición de la experiencia es lograr la estabilidad rompiendo la dependencia temporal entre las observaciones utilizadas en el entrenamiento la red neuronal profunda [43].

DQN es una mejora significativa a los algoritmos de RL clásicos, sin embargo es bien sabido que posee problemas de eficiencia en las muestras, además en comparación con otro algoritmos es inestable y no logra tan buenos resultados tanto en desempeño como en tiempo de entrenamiento, como se mostrará más adelante. Una de las razones por las que DQN sufre de una eficiencia de muestra tan baja es el método de muestreo de la memoria de repetición [45].

A esto se añade la naturaleza determinista de la política óptima, que limita su uso en dominios de adversarios y que encontrar la acción *greedy* con respecto a la función Q es costoso para espacios de acción grandes [46].

Para resolver dichos problemas, alrededor de DQN se han implementado diversas mejoras y aplicado diversas técnicas, en un intento por solventar sus problemas inherentes, sin embargo a costa de aumentar la complejidad de implementación. Por tanto, se analizarán otros que sean más simples de implementar y posean un mejor desempeño.

1.5.2 Trust Region Policy Optimization

Trust Region Policy Optimization, o TRPO, es un método de gradiente de políticas que evita las actualizaciones de parámetros que cambian demasiado la política con una restricción de

divergencia KL[8] sobre el tamaño de la actualización de la política en cada iteración[48], esto define una región que delimita los cambios en la política, la llamada Trust Region; entonces, esencialmente, se vuelve un problema de optimización con restricciones. El uso de la Trust Region ayuda a resolver el problema relativo a la tasa de aprendizaje en los métodos de gradiente de políticas, sin embargo, requiere el uso de gradientes conjugados[9] para la solución de la función con restricciones.

En las Trust Region, se determina el tamaño de paso máximo y luego se encuentra el máximo local de la política dentro de la región. Al continuar el mismo proceso iterativamente, se encontrará el máximo global. También es posible ampliar o reducir la región en función de lo buena que sea la nueva aproximación. De esa manera, se asegura que las nuevas políticas pueden ser confiables y no conducirán a una degradación de políticas dramáticamente mala [50].

TRPO utiliza dos posibles esquemas en su funcionamiento: El primer esquema de muestreo, llamado ruta única, es el que normalmente se usa para la estimación del gradiente de políticas, y se basa en el muestreo trayectorias individuales. El segundo esquema, llamado vine, implica construir un conjunto de implementación y luego realizar múltiples acciones desde cada estado en el conjunto de implementación. Este método se ha explorado principalmente en el contexto de los métodos de iteración de políticas [51].

TRPO es un algoritmo poderoso pero, aunque es útil para tareas de control continuo, no es fácilmente compatible con algoritmos que comparten parámetros entre una política y una función de valor o pérdidas auxiliares, como las que se usan para resolver problemas en Atari y otros dominios donde la entrada visual es significativa [28].

Además, el uso de los gradientes conjugados hace que la implementación sea más complicada y menos flexible que usando Stochastic Gradient Descent[10] [53].

---

[8] La divergencia de Kullback-Leibler(KL divergence, también llamada entropía relativa y divergencia I), denotada $D_{KL}(P||Q)$, es un tipo de distancia estadística: una medida de cómo una distribución de probabilidad P es diferente de una segunda distribución de probabilidad de referencia Q [47].

[9] El método del gradiente conjugado es una técnica matemática que puede ser útil para la optimización de sistemas tanto lineales como no lineales. Esta técnica generalmente se usa como un algoritmo iterativo, sin embargo, se puede usar como un método directo y producirá una solución numérica [49].

[10] El descenso de gradiente estocástico (SGD) es un algoritmo de optimización que a menudo se usa en aplicaciones de aprendizaje automático para encontrar los parámetros del modelo que se corresponden con el mejor ajuste entre los resultados previstos y los reales. La ruta de convergencia de SGD es más ruidosa que la del descenso de gradiente original, pero mucho más rápida, y es usualmente considerada una opción mucho mejor [52].

1.5.3 Actor Critic with Experience Replay

Actor Critic with Experience Replay (ACER) surgió como alternativa a las variantes existentes de Deep Q-Networks con prioritized replay (repetición de experiencia priorizada (prioritized experience replay) es un tipo de experience replay en el aprendizaje por refuerzo en el que, con mayor frecuencia, reproducimos transiciones con un alto progreso de aprendizaje esperado, medido por la magnitud de su error de diferencia temporal (TD) [54], y a las variantes existentes de métodos de gradiente de políticas aplicables a dominios tanto continuos como a discretos , como el actor crítico de ventaja asincrónica en política (A3C), debido a las limitaciones de los primeros (explicadas anteriormente) y a la alta ineficiencia de muestras de los segundos. ACER coincide con el rendimiento de las mejores deep Q-networks con reproducción priorizada contemporáneos a él en Atari, y supera sustancialmente a A3C en términos de muestra eficiencia tanto en Atari como en dominios de control continuo [46].

ACER puede ser entendido como la contraparte off-policy del método A3C [55]. Como tal, ACER se basa en todas las innovaciones de ingeniería de A3C, incluida la computación de CPU paralela eficiente. Sin embargo para poder lograr el alto desempeño que posee ACER aprovecha varias técnicas avanzadas en redes neuronales profundas, técnicas de reducción de varianza, el algoritmo *off-policy* Retrace [56] y entrenamiento paralelo de agentes RL [55]. Sin embargo, de manera crucial, su éxito depende de varias innovaciones: muestreo de importancia truncado con corrección de sesgo, arquitecturas de red de duelo estocástico y una versión más eficiente de TRPO [46].

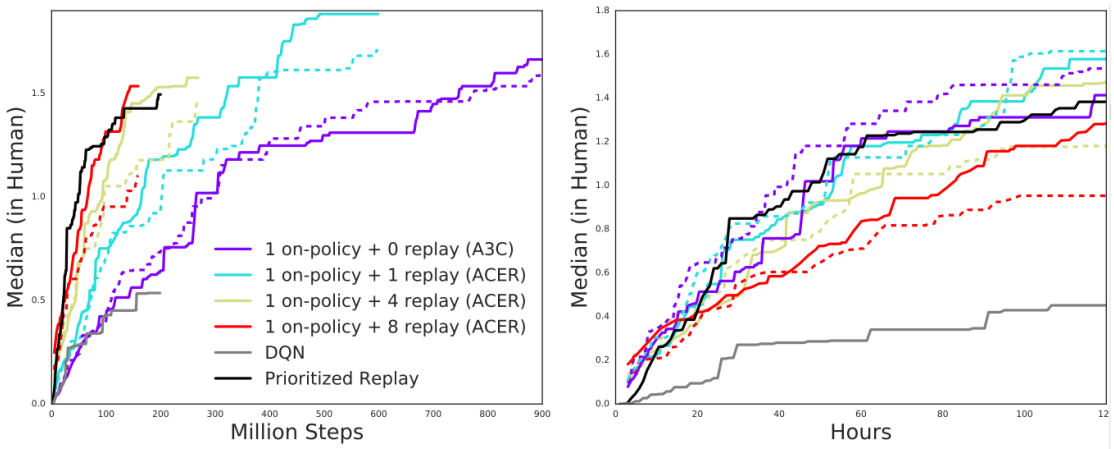


Figura 5. Mejoras de ACER en la complejidad de muestra (IZQUIERDA) y computación (DERECHA) en Atari (tomado de [46]*)*

En la imagen se muestra la comparación de ACER con otros algoritmos. En cada gráfico, la mediana de la puntuación normalizada por humanos en los 57 juegos de Atari se presenta para 4 proporciones de *experiece replay* con 0 replays correspondiente a A3C on-policy. Las líneas sólidas y discontinuas de colores representan ACER con y sin actualización de la región de

confianza respectivamente. Los pasos del entorno son contados para todos los hilos. La curva gris es el agente DQN original y la curva negra es uno de los agentes usando Double DQN con prioritized replay de [54].

El desempeño de ACER es notablemente elevado, probándose como un algoritmo robusto, pero a la vez de una alta complejidad al juntar una gran cantidad de técnicas para solventar los problemas en sus predecesores.

1.5.4 Proximal Policy Optimization

Como se ha visto anteriormente, los algoritmos de aprendizaje reforzado vienen con algunos problemas generalmente: tienen muchas partes móviles que son difíciles de depurar y requieren un esfuerzo considerable de ajuste para obtener buenos resultados. Proximal Policy Optimization (PPO) logra un equilibrio entre la facilidad de implementación, la complejidad de la muestra y la facilidad de ajuste, tratando de calcular una actualización en cada paso que minimice la función de costo mientras asegura que la desviación de la política anterior sea relativamente pequeña.

Su diseño original era similar al de TRPO que utiliza una penalización KL adaptativa para controlar el cambio de la política en cada iteración; sin embargo, esta idea inicial fue reemplazada por una nueva variante que utiliza una nueva función objetivo que normalmente no se encuentra en otros algoritmos:

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

- θ *es el parámetro de la política*
- $\hat{E}_t$ denota la expectativa empírica por cada lapso de tiempo
- r_t es la razón de probabilidad bajo las políticas nueva y vieja respectivamente
- $\hat{A}_t$ es la ventaja estimada en el tiempo t
- ε es un hiperparámetro, usualmente 0.1 o 0.2

Este objetivo implementa una forma de realizar una actualización de la Trust Region que es compatible con SGD y simplifica el algoritmo al eliminar la penalización de KL y la necesidad de realizar actualizaciones adaptativas. En las pruebas, este algoritmo ha mostrado el mejor desempeño en tareas de control continuo y casi iguala el desempeño de ACER en Atari, a pesar de ser mucho más simple de implementar [28].

1.5.5 Soft Actor-Critic

Soft actor-critic (SAC) es un algoritmo de RL profundo *model-free* y off-policy que está bien alineado con los requisitos del uso de RL en la vida real: eficiencia de la muestra, sin hiperparámetros sensibles y aprendizaje off-policy [57].

SAC se define para tareas de RL que involucran acciones continuas, siendo su característica principal que utiliza una función objetivo RL modificada: en lugar de solo buscar maximizar las

recompensas de por vida, SAC también busca maximizar la entropía de la política. Se puede pensar en la entropía como lo impredecible que es una variable aleatoria: si una variable aleatoria siempre toma un solo valor, entonces tiene entropía cero porque no es impredecible en absoluto; al contrario, si una variable aleatoria puede ser cualquier número real con la misma probabilidad, entonces tiene una entropía muy alta porque es muy impredecible [58].

Se quiere una alta entropía en la política para alentar explícitamente la exploración, para alentar a la política a asignar probabilidades iguales a acciones que tienen valores Q iguales o casi iguales, y también para garantizar que no colapse seleccionando repetidamente una acción particular que podría explotar alguna inconsistencia en la función Q aproximada. Por lo tanto, SAC supera el problema de la fragilidad al alentar a la red de políticas a explorar y no asignar una probabilidad muy alta a ninguna parte del rango de acciones [58].

La formulación de máxima entropía proporciona una sustancial mejora en exploración y robustez: las políticas de máxima entropía son robustas ante errores de modelo y estimación, y mejoran la exploración adquiriendo conductas diversas [59].

A diferencia de PPO, SAC es off-policy, lo que significa que puede aprender de las experiencias recopiladas en cualquier momento del pasado. A medida que se recopilan las experiencias, se colocan en un búfer de reproducción de experiencias (experience replay buffer) y se extraen aleatoriamente durante el entrenamiento. Esto hace que SAC sea significativamente más eficiente en cuanto a muestras, ya que a menudo requiere de 5 a 10 veces menos muestras para aprender la misma tarea que PPO. Sin embargo, SAC tiende a requerir más actualizaciones de modelos. SAC es una buena opción para entornos más pesados o más lentos (alrededor de 0,1 segundos por paso o más). SAC es también un algoritmo de máxima entropía y permite la exploración de forma intrínseca [13].

Esto prueba que SAC sería una mejor opción en general que PPO, sin embargo la idea tras el trabajo de investigación actual es el desarrollo de un agente capaz de desempeñarse en un entorno de adversarios, dígase contra un oponente humano, siendo le técnica de Self-play lo ideal en este caso. Self-play se puede usar con ambas implementaciones de PPO y SAC. Sin embargo, desde la perspectiva de un agente individual, estos escenarios parecen tener una dinámica no estacionaria porque el oponente a menudo cambia. Esto puede causar problemas significativos en el mecanismo de experience replay que utiliza SAC. Por lo tanto, se recomienda que los usuarios utilicen PPO [13].

1.5.6 Selección del algoritmo a utilizar

Luego del análisis de los algoritmos anteriores, se descarta el uso de DQN, TRPO y ACER de inmediato. DQN posee un desempeño pobre comparado a todos los demás: ineficiencia de muestra, inestabilidad y menores recompensas en general, y para solventarlos requiere adicionar varias técnicas adicionales que suman complejidad al algoritmo base. TRPO constituye una mejora

con respecto a DQN, sin embargo, sigue teniendo limitaciones y una alta complejidad en su implementación. ACER presenta un desempeño muy elevado y no presenta los problemas de los anteriores, pero su diseño lo hace demasiado complicado de entender e implementar. PPO logra un desempeño comparable al de ACER, siendo mucho más simple de implementar tanto que ACER como de TRPO, logrando estabilidad y una eficiencia de muestra aceptable. Si bien SAC supera a PPO en la mayoría de aspectos, con una alta eficiencia de muestra y estabilidad, sus problemas intrínsecos al enfrentarse a dominios adversarios hacen que no sea viable para usarse en la solución, por tanto, PPO es el elegido como algoritmo a utilizar.

1.6 Metodologías para el desarrollo de software

A continuación, se describe la metodología empleada durante la implementación de la solución. Se realizó un análisis de las metodologías de software más populares, fundamentando así la selección de la más adecuada para guiar la etapa de desarrollo del componente.

1.5.1 Metodología de Software

La metodología de desarrollo de software es el conjunto de técnicas y métodos que se utilizan para diseñar una solución de software informático. Es importante señalar que existen varias, de manera que es una decisión de cada equipo. Trabajar con una metodología es imprescindible por una cuestión de organización. No en vano, los factores tienen que estar ordenados y saber cómo se van a utilizar. Por otra parte, las metodologías también sirven para controlar el desarrollo del trabajo. Esto sirve para minimizar los márgenes de errores y anticiparse a esa situación [38].
Existen dos tipos de enfoques para metodologías de software: tradicionales y ágiles, con sus características propias que difieren bastante. A continuación, se muestra una tabla comparativa de dichas características:

Tabla 1.Comparación de Metodologías [39]

Metodologías tradicionales	Metodologías ágiles
Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo	Basadas en heurísticas provenientes de prácticas de producción de código
Cierta, aunque leve, resistencia a cambios	Especialmente preparadas para cambios durante el proyecto
Impuestas externamente	Impuestas internamente (por el equipo)
Proceso mucho más controlado, con numerosas políticas y normas	Proceso menos controlado, con pocos principios

El cliente interactúa con el equipo de desarrollo mediante reuniones	El cliente es visto como parte del equipo de desarrollo, participa activamente
Grupos grandes y posiblemente distribuidos	Grupos pequeños (menos de 10 integrantes) y trabajando en el mismo sitio
La arquitectura de software es esencial y se expresa mediante modelos	Menos énfasis en la arquitectura de software

Partiendo de esta comparación, las características, etapas y ventajas de cada una, y tomando en cuenta el tamaño del equipo de desarrollo (solo una persona en este caso), así como la magnitud del proyecto y sus particularidades, se determina utilizar una metodología ágil. Dentro de las metodologías ágiles hay bastante diversidad, así que se procede a analizar algunas de las más populares:

Scrum: Un marco de trabajo por el cual las personas pueden acometer problemas complejos adaptativos, a la vez que entregar productos del máximo valor posible productiva y creativamente. Según describen sus creadores, Scrum es: ligero, fácil de entender y extremadamente difícil de llegar a dominar. Scrum es un marco de trabajo de procesos que ha sido usado para gestionar el desarrollo de productos complejos desde principios de los años 90, no es un proceso o una técnica para construir productos; en lugar de eso, es un marco de trabajo dentro del cual se pueden emplear varias técnicas y procesos. Scrum muestra la eficacia relativa de las prácticas de gestión de producto y las prácticas de desarrollo, de modo que el equipo pueda mejorar constantemente [40].

Kanban: Kanban consiste en un sistema de señales visuales de control de producción que mantiene activo el proceso de reabastecimiento. Para mandar la señal de reabastecimiento existen una amplia variedad de métodos, desde tarjetas o tableros, señales visuales o electrónicas. La elección de un método de aviso u otro dependerá de las condiciones de la empresa, así como de las características del producto. Esto tiene como objetivo el control de flujo, evitar la sobreproducción, incrementar y mejorar la comunicación entre procesos y centros de trabajo [41].

XP: La Programación Extrema es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. XP surge como respuesta y posible solución a los problemas derivados del cambio en los requerimientos, por tanto se plantea como una metodología a emplear en proyectos de riesgo y su objetivo primordial es aumentar la productividad [42].

Luego del análisis de las variantes más populares de entre las metodologías ágiles, se decide optar por XP como metodología guía para el desarrollo del software, ya que el flujo de trabajo, además de la simplicidad propuestas por ella, se adaptan a las necesidades del proyecto características del equipo de desarrollo.

1.7 Herramientas y tecnologías

A continuación, se describen las herramientas y tecnologías a utilizar durante el desarrollo de la solución.

1.7.1 Lenguaje de Programación y Motor de videojuego

Python: es un lenguaje de programación de propósito general de alto nivel. Su filosofía de diseño enfatiza la legibilidad del código con el uso de una sangría significativa. Sus construcciones de lenguaje y su enfoque orientado a objetos tienen como objetivo ayudar a los programadores a escribir código claro y lógico para proyectos de pequeña y gran escala. Python se escribe dinámicamente y admite múltiples paradigmas de programación, incluida la programación estructurada (particularmente, procedural), orientada a objetos y funcional. A menudo se describe como un idioma de "baterías incluidas" debido a su completa biblioteca estándar. ¿Pero, por qué Python? [60]

Tiene una gran cantidad de bibliotecas y marcos: el lenguaje Python viene con muchas bibliotecas y marcos que facilitan la codificación. Esto también ahorra una cantidad significativa de tiempo. Las bibliotecas más populares son NumPy, que se utiliza para cálculos científicos; SciPy para cálculos más avanzados; y scikit, para aprender minería y análisis de datos. Estas bibliotecas funcionan junto con potentes marcos como TensorFlow, CNTK y Apache Spark. Estas bibliotecas y marcos son esenciales cuando se trata de proyectos de machine learning y deep learning [60].

Simplicidad: el código de Python es conciso y legible incluso para los nuevos desarrolladores, lo que es beneficioso para los proyectos de aprendizaje automático y profundo. Debido a su sintaxis simple, el desarrollo de aplicaciones con Python es rápido en comparación con muchos lenguajes de programación. Además, permite al desarrollador probar algoritmos sin implementarlos. El código legible también es vital para la codificación colaborativa. Muchos individuos pueden trabajar juntos en un proyecto complejo.

El soporte masivo en línea: Python es un lenguaje de programación de código abierto y disfruta de un excelente soporte de muchos recursos y documentación de calidad en todo el mundo. También cuenta con una comunidad grande y activa de desarrolladores que brindan su asistencia en cualquier etapa del desarrollo. La mayoría de los científicos han adoptado Python para proyectos de aprendizaje automático y aprendizaje profundo, lo que significa que la mayoría de las mentes más brillantes del mundo se pueden encontrar en las comunidades de Python.

Desarrollo rápido: Python tiene una sintaxis que es fácil de entender y amigable. Además, los numerosos marcos y bibliotecas impulsan el desarrollo de software. Mediante el uso de soluciones listas para usar, se puede hacer mucho con unas pocas líneas de código. Python es bueno para desarrollar prototipos, lo que aumenta la productividad.

Integraciones flexibles: los proyectos de Python se pueden integrar con otros sistemas codificados en diferentes lenguajes de programación. Esto significa que es mucho más fácil combinarlo con otros proyectos de IA escritos en otros idiomas. Además, dado que es extensible y portátil, Python se puede usar para realizar tareas entre idiomas. La adaptabilidad de Python facilita a los científicos y desarrolladores de datos entrenar modelos de machine learning [60].

Pruebas de código rápidas: Python proporciona muchas herramientas de prueba y revisión de código. Los desarrolladores pueden verificar rápidamente la corrección y la calidad del código. Los proyectos de IA tienden a consumir mucho tiempo, por lo que se necesita un entorno bien estructurado para probar y verificar errores. Python es el lenguaje ideal ya que soporta estas características [60].

Herramientas de visualización: Python viene con una amplia variedad de bibliotecas. Algunos de estos marcos ofrecen buenas herramientas de visualización. En IA, machine learning y deep learning, es importante presentar los datos en un formato legible por humanos. Por lo tanto, Python es una opción perfecta para implementar esta función [60].

Unity: es un motor de juegos multiplataforma desarrollado por Unity Technologies, anunciado y lanzado por primera vez en junio de 2005 en la Conferencia Mundial de Desarrolladores de Apple Inc. como un motor de juegos exclusivo de Mac OS X. Desde entonces, el motor se ha ampliado gradualmente para admitir una variedad de plataformas de escritorio, móviles, consolas y realidad virtual. Es particularmente popular para el desarrollo de juegos móviles iOS y Android y se utiliza para juegos como Pokémon Go, Monument Valley, Call of Duty: Mobile, Beat Saber y Cuphead.[4] Se considera fácil de usar para desarrolladores principiantes y es popular para el desarrollo de juegos independientes. El motor se puede utilizar para crear juegos tridimensionales (3D) y bidimensionales (2D), así como simulaciones interactivas y otras experiencias. Siendo este el motor gráfico principal utilizado por el centro VERTEX, se consideró pertinente seleccionarlo para el desarrollo de un entorno donde pueda desenvolverse el agente desarrollado [61].

Paquete ML-Agents: El kit de herramientas de agentes de aprendizaje automático de Unity (ML-Agents) es un proyecto de código abierto que permite que los juegos y las simulaciones sirvan como entornos para entrenar agentes inteligentes. Proporciona implementaciones (basadas en PyTorch) de algoritmos de última generación para permitir a los desarrolladores de juegos y aficionados entrenar fácilmente a agentes inteligentes para juegos 2D, 3D y VR/AR. Los investigadores también pueden usar la API de Python fácil de usar proporcionada para capacitar a los agentes mediante el aprendizaje reforzado, el aprendizaje por imitación, la neuroevolución o cualquier otro método. Estos agentes capacitados se pueden usar para múltiples propósitos, incluido el control del comportamiento de los NPC (en una variedad de entornos, como multiagente y adversario), pruebas automatizadas de compilaciones de juegos y evaluación de diferentes decisiones de diseño de juegos antes del lanzamiento. El ML-Agents Toolkit es mutuamente beneficioso tanto para los desarrolladores de juegos como para los investigadores de IA, ya que

proporciona una plataforma central donde los avances en IA pueden evaluarse en los entornos ricos de Unity y luego ponerse a disposición de las comunidades más amplias de investigación y desarrolladores de juegos [13].

Funcionamiento del paquete ML-Agents:

Con ML-Agents es posible entrenar el comportamiento de NPCs (*non-player characters*) llamados agentes, usando diversos métodos, los cuales parten de tres entidades básicas para cada momento del juego, que sería el entorno en este caso [13]:

- **Observaciones**: lo que el agente percibe sobre el medio ambiente. Las observaciones pueden ser numéricas y/o visuales, las primeras numéricas miden atributos del entorno desde el punto de vista del agente, estos serían atributos del campo de batalla que son visibles para él. Para la mayoría de los entornos interesantes, un agente requerirá varias observaciones numéricas continuas. Las observaciones visuales, por otro lado, son imágenes generadas por las cámaras adjuntas al agente y representan lo que el agente está viendo en ese momento. Es común confundir la observación de un agente con el estado del entorno (o del juego). El estado del entorno representa información sobre toda la escena que contiene todos los personajes del juego. Sin embargo, la observación de los agentes solo contiene información de la que el agente es consciente y, por lo general, es un subconjunto del estado del entorno.
- **Acciones**: qué acciones puede tomar el agente. Al igual que las observaciones, las acciones pueden ser continuas o discretas según la complejidad del entorno y el agente: si el entorno es un mundo de cuadrícula simple donde solo importa su ubicación, entonces basta con una acción discreta que se tome en uno de los cuatro valores (norte, sur, este, oeste). Sin embargo, si el entorno es más complejo y el agente puede moverse libremente, entonces es más apropiado usar dos acciones continuas (una para la dirección y otra para la velocidad).
- **Señales de recompensa**: un valor escalar que indica qué tan bien lo está haciendo el agente. Hay que tener en cuenta que la señal de recompensa no necesita proporcionarse en todo momento, sino solo cuando el agente realiza una acción buena o mala. Por ejemplo, puede recibir una gran recompensa negativa si muere, una recompensa positiva modesta cuando revive a un miembro del equipo herido y una recompensa negativa modesta cuando un miembro del equipo herido muere debido a la falta de asistencia. La señal de recompensa es cómo se comunican los objetivos de la tarea al agente, por lo que deben configurarse de manera que maximizar la recompensa genere el comportamiento óptimo deseado.

Las entidades mencionadas anteriormente están implementadas como componentes de alto nivel como parte del *ML-Agents Toolkit* [13]:

Entorno de aprendizaje: que contiene la escena de Unity y todos los personajes del juego. La escena de Unity proporciona el entorno en el que los agentes observan, actúan y aprenden.

Python API: que contiene una interfaz Python de bajo nivel para interactuar y manipular un entorno de aprendizaje. A diferencia del entorno de aprendizaje, la API de Python no forma parte de Unity, sino que se encuentra fuera y se comunica con Unity a través de un Comunicador.

Comunicador Externo: que conecta el entorno de aprendizaje con la API de bajo nivel de Python. Vive dentro del entorno de aprendizaje.

Entrenadores en Python: que contiene todos los algoritmos de aprendizaje automático que permiten entrenar a los agentes. Los algoritmos se implementan en Python y forman parte de su propio paquete *mlagents*. El paquete expone una utilidad de línea de comandos: *mlagents-learn* que admite todos los métodos y opciones de capacitación del paquete.

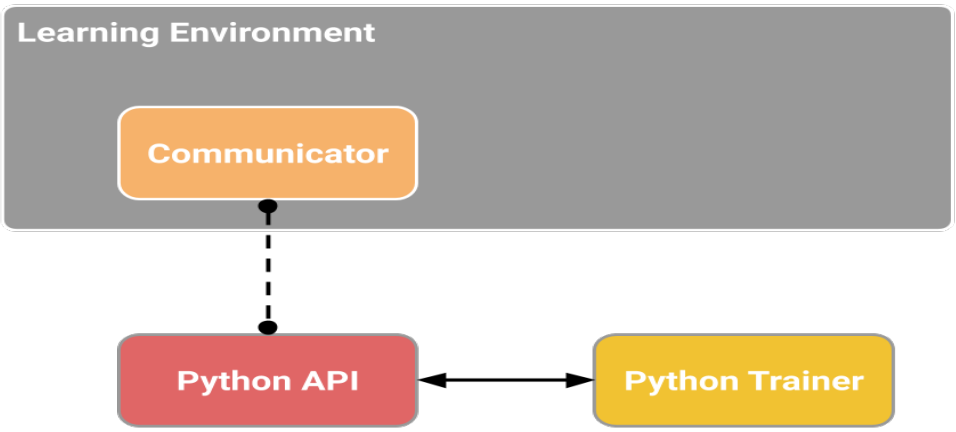


Figura 6. Componentes del entono de aprendizaje. Tomado de [13]

Aprendizaje reforzado profundo:

ML-Agents presenta implementaciones de dos algoritmos de aprendizaje reforzado: *Proximal Policy Optimization* (PPO) y *Soft Actor-Critic* (SAC), siendo PPO el algoritmo por defecto y utilizado en la solución [13].

1.7.2 Herramientas para el modelado

Para el modelado de la propuesta de solución se ha elegido Visual Paradigm for UML. Visual Paradigm (VP-UML) es una herramienta UML CASE compatible con UML 2, SysML y notación de modelado de procesos comerciales (BPMN) del Grupo de gestión de objetos (OMG). Además del soporte de modelado, proporciona capacidades de generación de informes e ingeniería de código, incluida la generación de código. Puede realizar ingeniería inversa de diagramas a partir de código y proporcionar ingeniería de ida y vuelta para varios lenguajes de programación [62].

1.8 Conclusiones del capítulo

En el presente capítulo, luego del acercamiento teórico pertinente, se concluye que un agente inteligente que emplee aprendizaje reforzado es ideal para su utilización en los videojuegos y que., utilizando el algoritmo PPO y una CNN para el aprendizaje, constituye la solución adecuada en pos de lograr el objetivo de la investigación. La comparación entre ambos enfoques, tradicionales y ágiles, de las metodologías de desarrollo de software llevó a la conclusión de que una metodología ágil era más adecuada para guiar el desarrollo e implementación de la propuesta de solución; eligiéndose de entre las variantes ágiles XP como metodología de desarrollo de software a utilizar.

Capítulo 2: Planificación y diseño de la propuesta de solución

El siguiente capítulo recoge las fases de planificación y diseño propuestas por la metodología XP, seleccionada anteriormente, incluyendo los artefactos generados en cada caso. Se describió en profundidad el funcionamiento tras el paquete ML-Agents, así como la implementación del algoritmo de aprendizaje reforzado PPO, usado en el agente. Sobre el propio agente se plasmó su comportamiento y las entradas y salidas necesarias para desenvolverse en el entorno proporcionado por el demo, también descrito en este capítulo. Se estableció un plan de entrega, y las iteraciones en que se cumplirá lo establecido en el mismo.

2.1 Propuesta de solución

Para dar solución al objetivo central de la investigación se propone elaborar un agente que utilice un algoritmo de aprendizaje reforzado profundo (*deep reinforcement learning*) para aprender por sí mismo a jugar un demo de un juego de naves estilo Space Invaders, donde podrá enfrentarse tanto a otras naves preprogramadas como a un jugador humano, pudiendo moverse en los ejes horizontal y vertical y disparar hacia las naves enemigas. El demo será implementado en Unity, apoyándose del paquete ML-Agents, para convertir la escena principal en un entorno de entrenamiento, que pueda ser empleado por la contraparte de dicho paquete en Python, el cual contiene implementaciones basadas en Pytorch y Tensorflow de algoritmos de *deep reinforcemnt learning.* Como algoritmo a ser empleado por el agente se utilizará la implementación de PPO que viene incluida en el paquete ML-Agents y una CNN como red neuronal, también parte de la configuración por defecto de dicho paquete.

El kit de herramientas de Unity ML-Agents proporciona una amplia variedad de escenarios, métodos y opciones de entrenamiento. Como tal, las ejecuciones de entrenamiento específicas pueden requerir diferentes configuraciones de entrenamiento y pueden generar diferentes artefactos y estadísticas de TensorBoard [54].

La configuración se determina mediante hiperparámetros que son definidos en un archivo de configuración, cuya ruta ese especificada al comenzar un entrenamiento. Es importante resaltar que entrenar con éxito un comportamiento en el kit de herramientas de ML-Agents implica ajustar la configuración y los hiperparámetros de entrenamiento [63].

A continuación, se muestran los hiperparámetros a utilizar en la solución, que permiten implementar las diferentes técnicas seleccionadas para el entrenamiento del agente, el algoritmo elegido y configurar la red neuronal, todo detallado en el anterior capítulo. Se prepara inicialmente la configuración del comportamiento, en un archivo nombrado VSBehavior.yaml, cuyo tipo de entrenamiento, determinado por trainer_type, es la implementación de PPO incluida en el paquete ML-Agents. Cabe destacar que Ml-Agents utiliza una CNN por defecto a no ser que se especifique

lo contrario coincidiendo con el tipo de red seleccionada para la solución. La siguiente tabla contiene las configuraciones usadas con su descripción correspondiente.

Tabla 2. Hiperparámetros

Configuración	Descripción
trainer_type	El tipo de entrenador a utilizar: PPO o SAC
Hiperparámetros	
batch_size	Número de experiencias en cada iteración de descenso de gradiente. Esto siempre debe ser varias veces más pequeño que buffer_size. Si está utilizando un espacio de acción continuo, este valor debe ser grande (del orden de 1000). Si está utilizando un espacio de acción discreto, este valor debe ser más pequeño (en el orden de 10). Para la solución se utiliza un espacio de acción continuo. Rango típico: (Continuo - PPO): 512 - 5120
buffer_size	Número de experiencias a recopilar antes de actualizar el modelo de política. Corresponde a cuántas experiencias se deben recopilar antes de realizar cualquier aprendizaje o actualización del modelo. Esto debería ser varias veces más grande que batch_size. Normalmente, un buffer_size más grande corresponde a actualizaciones de entrenamiento más estables. Rango típico: PPO: 2048 - 409600
learning_rate	Tasa de aprendizaje inicial para descenso de gradiente. Corresponde a la fuerza de cada paso de actualización de descenso de gradiente. Por lo general, esto debe reducirse si el entrenamiento es inestable y la recompensa no aumenta constantemente. Rango típico: 1e-5 - 1e-3
Hiperparámetros específicos de PPO	
beta	Fuerza de la regularización de la entropía, que hace que la política sea "más aleatoria". Esto asegura que los agentes exploren adecuadamente el espacio de acción durante el

	entrenamiento. Aumentar esto asegurará que se tomen más acciones aleatorias. Esto debe ajustarse de manera que la entropía (medible desde TensorBoard) disminuya lentamente junto con los aumentos en la recompensa. Si la entropía cae demasiado rápido, se debe aumentar beta. Si la entropía cae muy lentamente, se debe disminuir beta. Rango típico: 1e-4 - 1e-2
epsilon	Influye en la rapidez con la que la política puede evolucionar durante el entrenamiento. Corresponde al umbral aceptable de divergencia entre la política antigua y la nueva durante la actualización de descenso de gradiente. Establecer este valor en un valor pequeño dará como resultado actualizaciones más estables, pero también ralentizará el proceso de capacitación. Rango típico: 0.1 - 0.3
lambd	Parámetro de regularización (lambda) utilizado al calcular la Estimación de Ventaja Generalizada (GAE). Esto se puede considerar como cuánto confía el agente en su estimación de valor actual al calcular una estimación de valor actualizada. Los valores bajos corresponden a confiar más en la estimación del valor actual (que puede tener un alto sesgo), y los valores altos corresponden a confiar más en las recompensas reales recibidas en el entorno (que pueden tener una variación alta). El parámetro proporciona una compensación entre los dos, y el valor correcto puede conducir a un proceso de entrenamiento más estable. Rango típico: 0,9 - 0,95
num_epoch	Número de pasadas para realizar a través del búfer de experiencia al realizar la optimización de descenso de gradiente. Cuanto mayor sea el batch_size, mayor será aceptable para hacer esto. Disminuir esto asegurará actualizaciones más estables, a costa de un aprendizaje más lento.

	Rango típico: 3 - 10
learning_rate_schedule	Determina cómo cambia la tasa de aprendizaje con el tiempo: linear decae el learning_rate linealmente, llegando a 0 en max_steps, mientras que constant mantiene constante el ritmo de aprendizaje durante toda la ejecución de entrenamiento. Para PPO, se recomienda disminuir la tasa de aprendizaje hasta max_steps para que el aprendizaje converja de manera más estable(linear).
Configuración de la Red	
normalize	Determina si la normalización se aplica o no a las entradas del vector de observaciones. Esta normalización se basa en el promedio móvil y la varianza del vector de observaciones. La normalización puede ser útil en casos con problemas de control continuo complejos, pero puede ser perjudicial con problemas de control discreto más simples.
hidden_units	Número de unidades en las capas ocultas de la red neuronal. Corresponde a cuántas unidades hay en cada capa completamente conectada de la red neuronal. Para problemas simples donde la acción correcta es una combinación directa de las entradas de observación, esto debería ser pequeño. Para problemas donde la acción es una interacción muy compleja entre las variables de observación, esto debería ser mayor. Rango típico: 32 - 512
num_layers	El número de capas ocultas en la red neuronal. Corresponde a cuántas capas ocultas están presentes después de la entrada de observación, o después de la codificación CNN de la observación visual. Para problemas simples, es probable que menos capas entrenen más rápido y de manera más eficiente. Pueden ser necesarias más capas para problemas de control más complejos. Rango típico: 1 - 3
Señales de recompensa: extrínsecas	

strenght	Factor por el que multiplicar la recompensa otorgada por el entorno. Los rangos típicos variarán según la señal de recompensa. Rango típico: 1.00
gamma	Factor de descuento para futuras recompensas provenientes del medio ambiente. Esto se puede considerar como qué tan lejos en el futuro el agente debería preocuparse por las posibles recompensas. En situaciones en las que el agente debería estar actuando en el presente para prepararse para las recompensas en un futuro lejano, este valor debería ser grande. En los casos en que las recompensas son más inmediatas, puede ser menor. Debe ser estrictamente menor que 1. Rango típico: 0,8 - 0,995
Señales de recompensa: Curiosity	
strenght	Magnitud de la recompensa de curiosidad generada por el módulo de curiosidad intrínseca(ICM). Esto debe escalarse para garantizar que sea lo suficientemente grande como para no verse abrumado por señales de recompensa extrínsecas en el entorno. Asimismo, no debe ser demasiado grande para abrumar la señal de recompensa extrínseca. Rango típico: 0.001 - 0.1
gamma	Factor de descuento para recompensas futuras. Rango típico: 0,8 - 0,995
encoding_size	Tamaño de la codificación utilizada por el modelo de curiosidad intrínseca. Este valor debe ser lo suficientemente pequeño para alentar al ICM a comprimir la observación original, pero tampoco demasiado pequeño para evitar que aprenda a diferenciar entre las observaciones esperadas y las reales. Rango típico: 64 - 256
learning_rate	Tasa de aprendizaje utilizada para actualizar el módulo de

	curiosidad intrínseca. Por lo general, esto debería reducirse si el entrenamiento es inestable y la pérdida de curiosidad es inestable. Rango típico: 1e-5 - 1e-3
Configuraciones generales	
max_steps	Número total de pasos (es decir, observación recopilada y acción tomada) que se deben realizar en el entorno (o en todos los entornos si se usan varios en paralelo) antes de finalizar el proceso de capacitación. Si tiene varios agentes con el mismo nombre de comportamiento dentro de su entorno, todos los pasos realizados por esos agentes contribuirán al mismo recuento de max_steps. Rango típico: 5e5 - 1e7
time_horizon	Cuántos pasos de experiencia recopilar por agente antes de agregarlo al búfer de experiencia. Cuando se alcanza este límite antes del final de un episodio, se usa una estimación de valor para predecir la recompensa general esperada del estado actual del agente. Como tal, este parámetro compensa entre una estimación menos sesgada, pero con mayor varianza (horizonte de tiempo largo) y una estimación más sesgada, pero menos variada (horizonte de tiempo corto). En los casos en los que hay recompensas frecuentes dentro de un episodio, o los episodios son prohibitivamente grandes, un número más pequeño puede ser más ideal. Este número debe ser lo suficientemente grande para capturar todo el comportamiento importante dentro de una secuencia de acciones de un agente. Rango típico: 32 - 2048
summary_freq	Número de experiencias que deben recopilarse antes de generar y mostrar estadísticas de capacitación. Esto determina la granularidad de los gráficos en Tensorboard.
Self-Play	
save_steps	Número de pasos del entrenador entre “instantáneas”. Por ejemplo, si save_steps = 10000, se guardará una

	“instantánea” de la política actual cada 10000 pasos del entrenador. Tenga en cuenta que los pasos del entrenador se cuentan por agente. Un valor mayor de save_steps generará un conjunto de oponentes que cubren una gama más amplia de niveles de habilidad y posiblemente estilos de juego, ya que la política recibe más entrenamiento. Como resultado, el agente entrena contra una variedad más amplia de oponentes. Aprender una política para derrotar a oponentes más diversos es un problema más difícil y, por lo tanto, puede requerir más pasos generales de entrenamiento, pero también puede conducir a una política más general y sólida al final de la capacitación. Este valor también depende de cuán intrínsecamente difícil sea el entorno para el agente. Rango típico: 10000 - 100000
team_change	Número de pasos de entrenamiento requeridos para cambiar el equipo de aprendizaje. Esta es la cantidad de pasos de entrenador que entrenarán los equipos asociados con un entrenador fantasma específico antes de que un equipo diferente se convierta en el nuevo equipo de aprendizaje. Es posible que, en los juegos asimétricos, los equipos contrarios requieran menos pasos del entrenador para obtener ganancias de rendimiento similares. Esto permite a los usuarios capacitar a un equipo de agentes más complicado para más pasos de entrenador que un equipo de agentes más simple por cambio de equipo. Un mayor valor de team_change permitirá que el agente entrene más tiempo contra sus oponentes. Cuanto más tiempo entrene un agente contra el mismo grupo de oponentes, más capaz será de derrotarlos. Sin embargo, entrenar contra ellos durante demasiado tiempo puede resultar en un ajuste excesivo a las estrategias particulares del oponente(overfitting) y, por lo tanto, el agente puede fallar contra el siguiente grupo de oponentes. El valor del cambio de equipo determinará cuántas

	“instantáneas” de la política del agente se guardan para usarlas como oponentes para el otro equipo. Por lo tanto, se recomienda configurar este valor como una función del parámetro save_steps discutido anteriormente. Rango típico: 4x-10x donde x= save_steps
swap_steps	Número de pasos fantasma (no pasos de entrenador) entre el intercambio de la política de los oponentes con una instantánea diferente. Un “paso fantasma” se refiere a un paso dado por un agente que sigue una política fija y no aprende, es decir, el oponente de nuestro agente de aprendizaje actual. Rango típico: 10000 - 100000
windows	Tamaño del *buffer* de instantáneas pasadas de las que se muestrean los oponentes del agente. Por ejemplo, un tamaño de windows de 5 guardará las últimas 5 instantáneas tomadas. Cada vez que se toma una nueva instantánea, se descarta la más antigua. Un valor mayor de windows significa que el grupo de oponentes de un agente contendrá una mayor diversidad de comportamientos, ya que contendrá políticas anteriores en la ejecución de entrenamiento. Al igual que en el hiperparámetro save_steps, el agente entrena contra una variedad más amplia de oponentes. Aprender una política para derrotar a oponentes más diversos es un problema más difícil y, por lo tanto, puede requerir más pasos generales de entrenamiento, pero también puede conducir a una política más general y sólida al final del entrenamiento. Rango típico: 5 - 30
play_against_latest_model_ratio	Probabilidad de que un agente juegue en contra de la última política del oponente. Con probabilidad 1 - play_against_latest_model_ratio, el agente jugará contra una instantánea de su oponente de una iteración anterior. Un valor mayor de play_against_latest_model_ratio indica que un agente jugará contra el oponente actual con más

	frecuencia. Dado que el agente está actualizando su política, el oponente será diferente de una iteración a otra. Esto puede conducir a un entorno de aprendizaje inestable, pero plantea al agente un currículo automático de situaciones cada vez más desafiantes que pueden conducir a una política final más sólida. Rango típico: 0.0 - 1.0

(info de la tabla tomada de [64])

La clase Agente representa a un actor en la escena que recopila observaciones y realiza acciones. La clase Agente generalmente se adjunta al GameObject en la escena que representa al actor, en este caso a las naves espaciales en escena. Cada Agente debe tener Parámetros de Comportamiento (Behavior Parameters) apropiados.

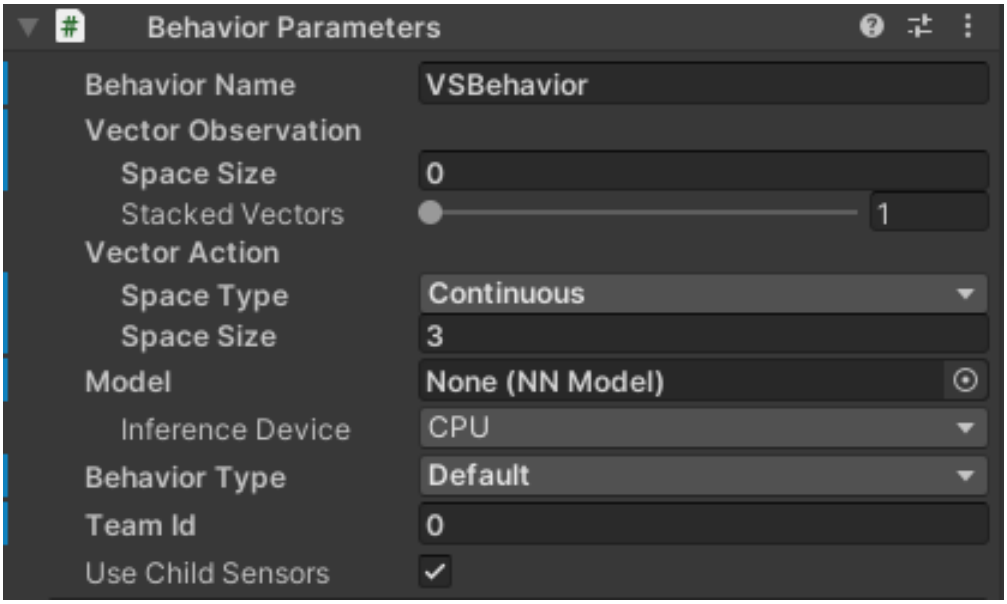


Figura 7. Parámetros de comportamiento del agente

De especial importancia entre estos son Vector Observation y Vactor Action. Una acción es una instrucción de la política que realiza el agente, cuando se especifica que el espacio de acción del vector es Continuo, el parámetro de acción que se pasa al Agente es un arreglo de números de punto flotante con una longitud igual al Vector Action Space Size, en este caso de valor 3. Los valores individuales en el arreglo tienen cualquier significado que se les atribuya, dependiendo del problema a resolver: si se asigna un elemento en el arreglo como la velocidad de un Agente, por ejemplo, el proceso de entrenamiento aprende a controlar la velocidad del Agente a través de este

parámetro. En la solución a implementar, el agente utiliza 3 valores del vector de acciones, con uno representando el movimiento en el eje X, el segundo el respectivo movimiento en el eje Y y el tercero un valor que determina si dispara o no.

Para que un agente aprenda, las observaciones deben incluir toda la información que un agente necesita para realizar su tarea. Sin información suficiente y relevante, un agente puede aprender mal o no aprender nada. Un enfoque razonable para determinar qué información debe incluirse es considerar lo que necesitaría para calcular una solución analítica al problema, o lo que esperaría que un ser humano pueda usar para resolver el problema [64].

En este caso Vector Observation Space Size posee un valor de 0 ya que las observaciones no se codifican manualmente, sino que se obtienen mediante Raycasts, ya que el agente puede usarlos para detectar visualmente al enemigo y los disparos en pantalla.

Esto se puede implementar fácilmente agregando un RayPerceptionSensorComponent2D al agente. Durante las observaciones, varios rayos (o esferas, según la configuración) se proyectan en el mundo de la física, y los objetos que golpean determinan el vector de observación que se produce.

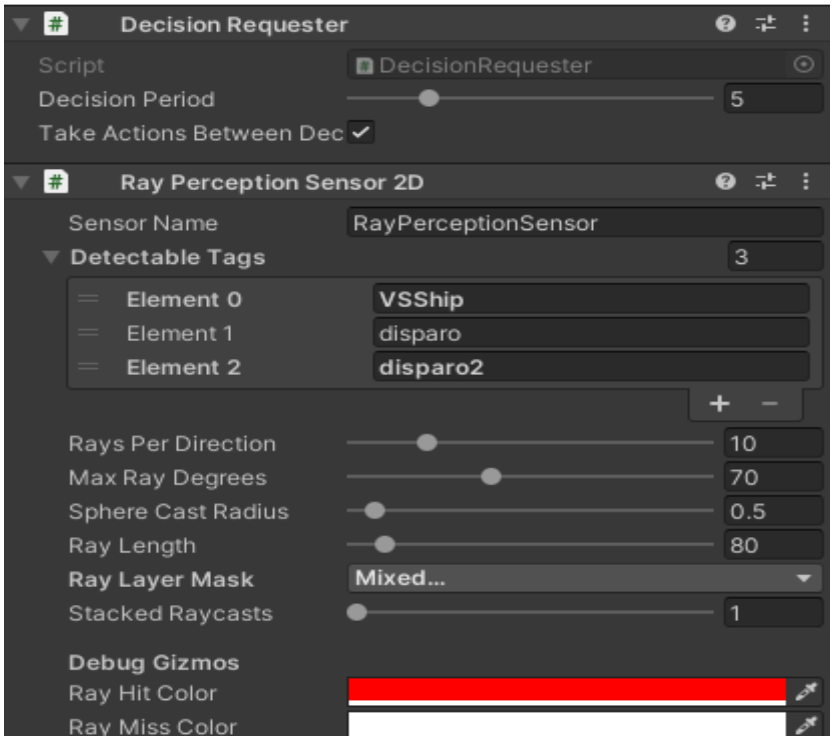


Figura 8. RayPerceptionSensor 2D, componente que otorgar visión del entorno al agente

Dicho componente cuenta con los siguientes elementos:

- Detectable Tags: Una lista de cadenas correspondientes a los tipos de objetos que el agente debería poder distinguir. Para el agente a implementar se utilizarán las etiquetas “VSShip”, “disparo” y “disparo2” correspondientes a la nave enemiga, y los disparos propios y enemigos respectivamente.

- Rays Per Direction: Determina el número de rayos que se emiten. Un rayo siempre se proyecta hacia adelante, y esta cantidad de rayos se proyecta hacia la izquierda y la derecha, 10 en el caso del agente.
- Max Ray Degrees: El ángulo (en grados) para los rayos más externos. Colocado en 70 grados hacia el frente del agente.
- Sphere Cast Radius: El tamaño de la esfera utilizada para la proyección de esferas. Se establece en 0, por tanto, se utilizarán rayos en lugar de esferas. Los rayos pueden ser más eficientes, especialmente en escenas complejas.
- Ray Lenght: La longitud de las proyecciones, 80 para abarcar toda la escena desde cualquier lugar donde se encuentre el agente.
- Ray Layer Mask: La layermask que se pasa al raycast. En este caso se utilizó para que los rayos de los agentes no sobrepasaran los límites del entorno donde se encuentran, sin tener que añadir las paredes como tags observables.
- Stacked Raycasts: El número de resultados anteriores para "apilar" con los resultados de la observación actual, con el objetivo de añadir memoria al agente, colocado a 1 ya que no interesa en este caso. Esto puede ser independiente de la configuración de Stacked Vectors en los Behavior Parameters.

Cabe destacar que se recomienda utilizar la menor cantidad de rayos y etiquetas necesarios para resolver el problema a fin de mejorar la estabilidad del aprendizaje y el rendimiento del agente[64].

En cuanto a las señales de recompensa, las cuales determinarán buena parte del comportamiento del agente, estarán divididas en recompensas extrínsecas e intrínsecas. Las primeras son las recompensas directas, en este caso el agente obtendrá una recompensa de +1 al golpear una nave enemiga, una de -1 al ser golpeado por un contrario. Las intrínsecas están determinadas por la *Curiosity*, explicada en el capítulo anterior, pues el entorno posee recompensas muy esporádicas.

El agente experimentará primero una fase de entrenamiento, en la cual usará las observaciones obtenidas del entorno para tomar las acciones posibles, derivándose de estas acciones una señal de recompensa que será usada para ajustar los pesos de la red lentamente. Como se menciona en el capítulo anterior, la exploración es parte importante, por tanto en el entrenamiento siempre estará presente la posibilidad de que el agente tome una acción aleatoria y así permitir que explore estados que de otra forma serían imposibles de visitar. El entrenamiento contará con ochocientos mil pasos en total, en los cuales el agente se enfrentará a sí mismo mediante el uso de *Self-Play* en partidas de dos mil pasos de duración, o hasta que uno elimine las tres vidas del otro, tras lo cual la partida volverá a iniciarse.

Al concluirse el entrenamiento se obtendrá una red ya entrenada, que puede ser incorporada al agente para que este juegue partidas en contra de jugadores humanos. Esta fase, conocida como fase de inferencia, evita toda exploración, centrándose en tomar las mejores decisiones posibles que permita la política aprendida durante el entrenamiento.

Comportamiento del Agente en una partida

Al iniciar cada partida el agente empezará en una posición aleatoria dentro de una porción de la pantalla de juego, cara a cara contra su adversario ya sea un jugador humano o NPCs. De acuerdo a lo definido por el paquete ML-Agents, contará con unas observaciones en forma de vector, que almacenará la información requerida por el agente para determinar el estado actual del entorno:

- Su posición como un vector de 3 elementos para las posiciones x, y, z
- La posición del/los enemigos cada una como un vector de 3 elementos similar al anterior
- La posición y velocidad de las balas disparadas por el/los enemigos y por sí mismo
- La posición de obstáculos que aparezcan en el entorno

Para determinar cada posible acción contará con un arreglo de 3 elementos, con cada posición del mismo determinando una acción posible:

- Posición 0: determina el movimiento en el eje X
- Posición 1: determina el movimiento en el eje Y
- Posición 2: determina cuando disparar

2.2 Planificación

Es la fase inicial de la metodología XP, caracterizada por una comunicación continua con el cliente en pos de definir los requisitos y las historias de usuarios, así como el alcance del proyecto y las fechas de entrega. Según la identificación de las historias de usuario, se priorizan y se descomponen en mini-versiones. La planificación se va a ir revisando y luego de la duración definida para cada iteración, se debe obtener un software útil, funcional, listo para probar y lanzar [65].

2.3 Requisitos

La especificación de requisitos de software (ERS) es una descripción completa del comportamiento del sistema que se va a desarrollar. Incluye un conjunto de casos de uso que describe todas las interacciones que tendrán los usuarios con el software. Los casos de uso también son conocidos como requisitos funcionales. Además de los casos de uso, la ERS también contiene requisitos no funcionales (complementarios). Los requisitos no funcionales son requisitos que imponen restricciones en el diseño o la implementación, como, por ejemplo, restricciones en el diseño o estándares de calidad. Está dirigida tanto al cliente como al equipo de desarrollo. El lenguaje utilizado para su redacción debe ser informal, de forma que sea fácilmente comprensible para todas las partes involucradas en el desarrollo [66].

Si bien la metodología XP no define los requisitos como un artefacto en si mismos, se decidió incluirlos para facilitar y apoyar la elaboración de las Historias de Usuario.

Tabla 3. Requisitos

Código	Descripción
Requisitos funcionales	
RF-01	Iniciar partida individual.
RF-02	Reiniciar partidas una vez acabadas
RF-03	Limitar el movimiento de las naves en el entorno
RF-04	Permitir el movimiento de las naves en los ejes horizontal y vertical
RF-05	Permitir ejecutar disparos
RF-06	Pausar la partida
RF-07	Seleccionar dificultad de la IA
Requisitos no funcionales	
RnF-01	No debe apreciarse un retraso visible en las acciones del agente
RnF-02	Bajos requerimientos para realizar el entrenamiento. Debe poder realizarse en computadoras i5 de cuarta generación o superior, que utilicen Windows 8 o superior.

2.3.1 Historias de Usuario

El primer paso de cualquier proyecto que siga la metodología XP es definir las historias de usuario con el cliente. Las historias de usuario tienen la misma finalidad que los casos de uso, pero con algunas diferencias: Constan de 3 o 4 líneas escritas por el cliente en un lenguaje no técnico sin hacer mucho hincapié en los detalles; no se debe hablar ni de posibles algoritmos para su implementación ni de diseños de base de datos adecuados, etc. Son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cuando llega la hora de implementar una historia de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia. El tiempo de desarrollo ideal para una historia de usuario es entre 1 y 3 semanas [65].

Existen varios tipos de plantillas para crear Historias de Usuario, así que se seleccionó la información más relevante a mostrar para las HU relativas al proyecto:

Número: índice que identifica la historia de usuario

Nombre: Nombre de la historia de usuario

Usuario: Usuario esperado de la funcionalidad que describe la historia

Iteración asignada: iteración planificada para el desarrollo de la historia de usuario

Prioridad: nivel de inmediatez e la historia de usuario

Puntos estimados: estimación de la duración del proceso de implementación de la historia usuario, típicamente en un rango de aproximadamente 40 horas de trabajo, siendo una semana y 5 días el tiempo ideal según expertos

Riesgo en desarrollo: impacto que traería no realizar la historia de usuario

Descripción: breve descripción de requerimientos

Tabla 4. HU 1: Modo de juego individual

HISTORIA DE USUARIO	
Número: 1	**Usuario**: jugador
Nombre: Modo de juego individual	
Prioridad: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 1.5	**Iteración**: 2
Descripción: el usuario debe tener la opción de empezar una partida individual, donde la IA es capaz de realizar acciones para derrotar al jugador.	

Tabla 5. HU 2: Menú principal

HISTORIA DE USUARIO	
Número: 2	**Usuario**: jugador
Nombre: Menú principal	
Prioridad: Media	**Riesgo en Desarrollo**: Media
Puntos Estimados: 1	**Iteración**: 2
Descripción: el demo debe contar con un menú principal que contenga las opciones pertinentes al inicio de una partida.	

Tabla 6. HU 5: Acciones del agente

HISTORIA DE USUARIO	
Número: 5	**Usuario**: jugador
Nombre: Acciones del agente	
Prioridad: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 2	**Iteración**: 1
Descripción: el agente debe aprender por si mismo a realizar las mismas acciones que el jugador de manera eficiente	

El resto de historias de usuario se encuentran en los anexos del documento.

2.3.2 Plan de Entrega

Luego de haber establecido las Historias de Usuario se realiza la planificación de lanzamientos o entregas, como resultado de esto se establece el Plan de entregas. Luego de haber negociado con el cliente los alcances de cada Historia de Usuario, es decir haber evaluado los riesgos, las prioridades y realizado estimaciones [67].

Tabla 7. Plan de Entrega

Historia de Usuario	**Puntos estimados**
Modo de juego individual.	1.5
Acciones del agente	2.0
Menú principal	1.0
Implementación del Entorno	0.5
Movimiento y disparos de la nave	0.5

Menú de pausa	0.5
Seleccionar dificultad de la IA	0.5
Total	6.5

2.3.3 Iteraciones

Esta fase incluye las iteraciones por las que pasara el desarrollo del sistema antes de ser entregado, donde se implementarán las Historias de Usuario en cuanto a la prioridad de estas definida en el Plan de Entrega, siendo recomendable no excederse de tres semanas por cada iteración. Se definieron dos iteraciones en este caso:

- 1ra Iteración: se abordan las historias de usuario que representan las funcionalidades mínimas para el funcionamiento del entorno y el agente(Modo de juego individual y Acciones del agente).
- 2da Iteración: abarca las historias de usuario restantes, con funcionalidades que incluyen componentes de importancia para el demo(Movimientos y Disparos de la nave) y otros simplemente complementarios(Obstáculos del entorno, dificultad de la IA, menús principal y de pausa).

Tabla 8. Iteraciones

Iteración	**Historia de usuario**	**Duración**
1	Movimiento y disparos de la nave	3.0
	Acciones del agente	
	Implementación del Entono	
2	Menú principal	3.0
	Modo de juego individual	
	Menú de pausa	
	Seleccionar dificultad de la IA	
Total		6.5

2.4 Diseño

La Metodología XP hace especial énfasis en los diseños simples y claros. Los conceptos más importantes de diseño en esta metodología son los siguientes:

- Simplicidad: Un diseño simple se implementa más rápidamente que uno complejo. Por ello XP propone implementar el diseño más simple posible que funcione.
- Soluciones "*Spike*": Cuando aparecen problemas técnicos, o cuando es difícil de estimar el tiempo para implementar una historia de usuario, pueden utilizarse pequeños programas de prueba (llamados "*Spike*"), para explorar diferentes soluciones.

- Recodificación ("*Refactoring*"): Consiste en escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de crearlo más simple, conciso y entendible. Las metodologías de XP sugieren re codificar cada vez que sea necesario [68].

2.4.1 Tarjetas CRC

Las tarjetas CRC (Clase-Responsabilidad-Colaboración) son parte de la metodología XP para el diseño de software orientado a objetos, la forma de diseño y organización es diseñar una tarjeta CRC por cada historia de usuario, ya que brinda una funcionalidad directa al negocio, una clase es una persona, cosa, evento, concepto, pantalla o reporte, las responsabilidades de una clase y los colaboradores de la misma [67].

Para ello la tarjeta CRC se divide en tres zonas:

- En la parte superior, el nombre de la clase.
- Debajo, en la parte izquierda, las responsabilidades de dicha clase, sus objetivos, a alto nivel.
- A la derecha de las responsabilidades, los colaboradores, las otras clases que ayudan a conseguir cumplir a esta con sus responsabilidades.

[69]

Tabla 9. Tarjeta CRC 1: Disparos

Tarjeta CRC	
Clase: Disparos	
Responsabilidad	**Colaboración**
• Contiene las funcionalidades necesarias para que los disparos se muevan por la pantalla • Registra los impactos de los disparos con los objetos, activando los métodos pertinentes en caso de impactar a un jugador	• SpaceShipAgent • PlayerScript

Tabla 10. Tarjeta CRC 2: PlayerScript

Tarjeta CRC	
Clase: PlayerScript	
Responsabilidad	**Colaboración**
• Contiene los algoritmos necesarios para mover la nave • Contiene el algoritmo para disparar • Contiene los métodos que permiten recibir castigos al recibir disparos y acabar la partida al perder todas las vidas	• SpaceShipAgent • Disparos

Tabla 11. Tarjeta CRC 3: SpaceShipAgent

Tarjeta CRC	
Clase: SpaceShipAgent	
Responsabilidad	**Colaboración**
• Contiene los algoritmos necesarios para el aprendizaje del agente • Contiene lo relativo a las observaciones y las acciones a tomar	• PlayerScript

2.5 Conclusiones del capítulo

Durante el capítulo se describió la propuesta de solución que será implementada en la siguiente fase del proyecto, definiéndose dos iteraciones de tres semanas aproximadamente, según un plan de entrega elaborado con las prioridades para el proyecto, basándose en las necesidades del cliente. Fueron escritas las historias de usuario, apoyadas en requisitos funcionales y no funcionales, para guiar la implementación; además, se establecieron las relaciones entre las clases que intervendrán en el funcionamiento del agente, en forma de Tarjetas CRC.

Capítulo 3: Implementación del agente y pruebas realizadas

3.1 Estándar de codificación

Los estándares de codificación incorporan principios de ingeniería sólidos para la programación en sus respectivos lenguajes y forman la base de cualquier enfoque preventivo [70].
Estos se utilizan durante la fase de implementación, para facilitar la comprensión y futuro mantenimiento del código. A continuación, se muestra el estándar de codificación a utilizar en la implementación:
Definición de clases: Todos los nombres de las clases comenzarán con letra inicial mayúscula, utilizando la normativa CamelCase-UpperCamelCase, que determina que la primera letra de cada palabra debe ser mayúscula.

```
26  class Agent {
27      //cuerpo de la clase
28  }
```

Figura 9. Definición de clases

Declaración de funciones y atributos: Los nombres de los métodos o funciones y atributos se escribirán con minúsculas; en caso de nombre compuesto, se regirá por la normativa CamelCase-lowerCamelCase.

```
30  <Tipo dato> atributo;
31  <Tipo dato> atributoNombreCompuesto;
```

Figura 10. Declaración de funciones y atributos

Definición de parámetros dentro de las funciones y constructores de las clases: Los nombres de los identificadores de los parámetros en las funciones y en los constructores de las clases deben estar escritos con minúsculas, separados a un espacio cada uno y en caso de ser compuesto utilizar la normativa CamelCase-lowerCamelCase.

```
33  <Tipo dato retorno> funcion(<tipo> <id1>,<tipo> <id2>,<tipo> <idN>)
34  Clase(tipo> <id1>,<tipo> <id2>,<tipo> <idN>)
```

Figura 11. Definición de parámetros dentro de las funciones y constructores de las clases

Definición de estructuras de control y bucles: En caso de las estructuras de control se separarán con un espacio las condiciones de la estructura de control, siempre abriendo llaves en la misma linea, y cerrándolas luego de las acciones, con un salto de línea como separador. En caso de las estructuras de control, else y else if, se escribirán en la misma línea que el cierre de llaves del if anterior.

```
if (condicion) {
    //acciones
}

if (condicion) {
    //acciones
} else {
    //otras acciones
}

if (condicion) {
    //acciones
} else if (condicion) {
    //otras acciones
} else {
    //otras acciones
}
```

Figura 12. Definición de estructuras de control

En el caso de los bucles, se escribirán con la misma estructura que un if:

```
<bucle> (condicion) {
    //acciones
}
```

Figura 13. Definición de estructuras de bucles

Comentarios: Los comentarios serán escritos según el estándar del lenguaje C#. El tipo de comentario más básico en C# es el comentario de una línea. Como su nombre indica, convierte una línea en un comentario:

```
// Comentario
```

En caso de los comentarios con múltiples líneas se utilizará el formato siguiente:

```
/*
Comentarios
*/
```

Se utilizará un salto de línea como divisor tras el inicio del comentario, marcado por "/*" y justo antes del final, marcado por "*/".

3.2 Tareas de Ingeniería

Las tareas de ingeniería pueden estar descritas por un lenguaje técnico y no ser necesariamente entendibles por el cliente. Tienen como objetivo definir cada una de las actividades que dan

cumplimiento a las HU, de forma tal que se entienda lo que el sistema tiene que hacer y facilite su construcción [71]. Las tareas de ingeniería serán separadas por las iteraciones definidas en el capítulo anterior, por tanto serán relativas a las HU dentro de cada una de ellas.

Primera Iteración:

Durante esta primera iteración, se pretende trabajar en los requisitos funcionales más importantes para el funcionamiento de la solución: la creación del entorno, las naves y el agente de aprendizaje profundo, con las funcionalidades requeridas para jugar y realizar el entrenamiento del agente, el cual también se desarrollará en esta iteración.

Tabla 12. Iteración 1. Tarea 1

Tarea de Ingeniería	
Número: 1	**Número de la HU**: 3
Nombre: Crear aracterísticas visuales del Entorno	
Estado: Completado	**Puntos estimados**: 0.2
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Crear la forma básica del entorno, su fondo, partículas y todo lo que permita simular el espacio exterior donde se moverán las naves.	

Tabla 13. Iteración 1. Tarea 2

Tarea de Ingeniería	
Número: 2	**Número de la HU**: 3
Nombre: Implementar límites del Entorno	
Estado: Completado	**Puntos estimados**: 0.3
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Se dota al entorno de bordes invisibles para limitar el movimiento de las naves, de manera que no salgan del mismo.	

Tabla 14. Iteración 1. Tarea 3

Tarea de Ingeniería	
Número: 3	**Número de la HU**: 4
Nombre: Crear modelos de las naves	
Estado: Completado	**Puntos estimados**: 0.1
Fecha de inicio:	**Fecha fin**:

Programador responsable: Daniel E. Sánchez Zerquera
Descripción: Crear los modelos de naves propias y enemigas.

Tabla 15. Iteración 1. Tarea 4

Tarea de Ingeniería	
Número: 4	**Número de la HU**: 4
Nombre: Implementar movimiento de la nave	
Estado: Completado	**Puntos estimados**: 0.1
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Otorgar a la nave del jugador movimiento al tocar las teclas de dirección en el teclado.	

Tabla 16. Iteración 1. Tarea 5

Tarea de Ingeniería	
Número: 5	**Número de la HU**: 4
Nombre: Implementar disparos de la nave	
Estado: Completado	**Puntos estimados**: 0.3
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Permitir a la nave realizar disparos en línea recta al pulsar la tecla “Espacio”, y que los disparos se destruyan al chocar contra los enemigos o las paredes.	

Tabla 17. Iteración 1. Tarea 6

Tarea de Ingeniería	
Número: 6	**Número de la HU**: 5
Nombre: Implementar principales acciones del agente RL	
Estado: Completado	**Puntos estimados**: 0.8
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Implementar la clase Agente, con los métodos básicos que le permitan realizar las acciones de moverse y disparar en base de los resultados devueltos por la red neuronal incorporada al mismo.	

Tabla 18. Iteración 1. Tarea 7

Tarea de Ingeniería

Número: 7	**Número de la HU**: 5
Nombre: Declarar red Neuronal asociada	
Estado: Completado	**Puntos estimados**: 0.3
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Declarar la red neuronal a utilizar, con los hiperparámetros definidos en el capítulo anterior y sus valores, que serán ajustados en base a los resultados de entrenamiento.	

Tabla 19. Iteración 1. Tarea 8

Tarea de Ingeniería	
Número: 8	**Número de la HU**: 5
Nombre: Implementar observaciones del agente RL	
Estado: Completado	**Puntos estimados**: 0.2
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Añadir los vectores de observación al agente, como se especifica en el capítulo anterior, los cuales dotaran al mismo de “visión” de los objetos relevantes en el entorno.	

Tabla 20. Iteración 1. Tarea 9

Tarea de Ingeniería	
Número: 9	**Número de la HU**: 5
Nombre: Implementar recompensas	
Estado: Completado	**Puntos estimados**: 0.3
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Implementar una serie de recompensas, que determinarán el aprendizaje del agente	

Tabla 21. Iteración 1. Tarea 10

Tarea de Ingeniería	
Número: 10	**Número de la HU**: 5
Nombre: Realizar el entrenamiento del agente	
Estado: Completado	**Puntos estimados**: 0.4
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Llevar a cabo el entrenamiento del agente, con diversos cambios en los valores de los hiperparámetros para ajustar el desempeño, enfocado en el entorno adversario con Self-Play,	

pero también contra enemigos fijos.

Segunda Iteración:

Durante esta segunda iteración, se planificó la implementación de las funcionalidades relativas a la interacción del usuario con el software, permitiendo iniciar una partida contra el agente y la selección de dificultad, así como funcionalidades más secundarias como los obstáculos del entorno.

Tabla 22. Iteración 2. Tarea 11

Tarea de Ingeniería	
Número: 11	**Número de la HU**: 2
Nombre: Implementar menú principal	
Estado: Completado	**Puntos estimados**: 1.0
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Implementar un menú principal, dentro del cual el usuario podrá elegir la dificultad, inciar partida o cerrar el demo.	

Tabla 23. Iteración 2. Tarea 12

Tarea de Ingeniería	
Número: 12	**Número de la HU**: 1
Nombre: Implementar modo de juego individual	
Estado: Completado	**Puntos estimados**: 1.0
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Implementar una sección que permita al usuario jugar un uno contra uno contra el agente.	

Tabla 24. Iteración 2. Tarea 13

Tarea de Ingeniería	
Número: 13	**Número de la HU**: 1
Nombre: Implementar vidas	
Estado: Completado	**Puntos estimados**: 0.5
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	

Descripción: Implementar la representación visual de las vidas del enemigos y del jugador para mejorar la experiencia.

Tabla 25. Iteración 2. Tarea 14

Tarea de Ingeniería	
Número: 14	**Número de la HU**: 7
Nombre: Implementar selección de la dificultad	
Estado: Completado	**Puntos estimados**: 0.5
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Programar la selección de la dificultad, que utilizara diferentes redes con diversos entrenamientos en cada dificultad, creciendo el reto a medida que la dificultad aumenta.	

Tabla 26. Iteración 2. Tarea 15

Tarea de Ingeniería	
Número: 15	**Número de la HU**: 6
Nombre: Implementar menú de pausa	
Estado: Completado	**Puntos estimados**: 0.5
Fecha de inicio:	**Fecha fin**:
Programador responsable: Daniel E. Sánchez Zerquera	
Descripción: Implementar un menú de pausa, que permita el jugador pausar el partida, reanudarla en caso de ya estar pausada o volver al menú principal.	

3.3 Descripción de la solución

La solución es básicamente un demo, que incluye un jugador virtual como oponente para un jugador real, en un entorno modelado en forma de un juego donde dos naves en el espacio se enfrentan. Cuenta con una parte visual, como cualquier videojuego diseñado en Unity 3D, aunque el demo en sí es un entorno enteramente 2D. Posee un grupo de clases, con métodos encargados de su funcionamiento como videojuego, definidas en scripts presentes en las naves y algunos otros scripts auxiliares para el movimiento y funcionamiento de los proyectiles disparados por las naves. Cuenta también con una clase Agente, que es la encargada del funcionamiento del agente como jugador virtual.

Al iniciar el demo, se aprecia un menú principal, con tres botones:

Un Jugador: Inicia una partida individual, en la cual el jugador podrá enfrentarse al agente, la dificultad de la partida será determinada en dependencia de la opción elegida en la sección “Seleccionar dificultad”, siendo “Medio” la dificultad por defecto.

Seleccionar dificultad: Pasa a una pantalla de selección simple, donde el jugador puede elegir a dificultad con la que quiere enfrentar al agente, de entre las posibles “Fácil”, ”Medio” y ”Difícil”.

Salir: Cierra la aplicación.

Una vez iniciada la partida, la mayor parte de las funcionalidades generales de la misma se encuentran en el script PlayerScript, que contiene los siguientes métodos:

movement(): Este método es el encargado del movimiento de la nave a control del jugador, en los ejes X y Y.

disparar(): Se encarga de realizar los disparos de las naves, además de regular el cooldown del mismo, provocando que solo se pueda disparar un proyectil cada medio segundo.

onHit(): Este método es llamado cuando el agente o el jugador reciben un disparo, reduce la cantidad de vida respectivamente, chequea cuando una de ambas haya llegado a cero para acabar la partida y, en caso del entrenamiento del agente, es la encargada de disminuir la recompensa del mismo al ser golpeado por un enemigo.

El script Proyectil posee solo un par de métodos, pero tiene gran importancia ya que es quien se encarga del movimiento de las balas y de otorgar recompensas al agente cuando sus disparos golpean a un enemigo.

El propio agente posee su script, llamado SpaceShipAgent, el cual cuenta con los siguientes métodos:

OnEpisodeBegin(): Uno de los métodos definidos por el paquete ML-Agents, define las acciones que se realizarán con cada inicio de un nuevo episodio de entrenamiento, en este caso mover tanto al agente como a su adversario a una posición aleatoria, y reiniciar sus vidas.

OnActionReceived(): Otro método definido por el paquete ML-Agents, recibe un arreglo con los valores devueltos por la red neuronal para asignarles una acción, como fue planteado en el capítulo anterior, el arreglo cuenta con tres valores, uno para el movimiento en el eje X, otro para el movimiento en el eje Y y otro para determinar cuando disparar o no, para darle dicho sentido a los valores, el método llama al método auxiliar agentStuff().

agentStuff(): Usa los valores obtenidos en el método OnActionReceived() para permitirse al agente moverse y disparar.

recompensa(): Es un método simple que recibe un número y lo adiciona como recompensa al agente, usando el método AddReward(), definido en el paquete ML-Agents para ese propósito.

3.4 Pruebas

La prueba de software es el proceso de evaluación y verificación de un producto o aplicación de software para saber si hace lo que se supone que debe hacer. Los beneficios de las pruebas

incluyen la prevención de errores, la reducción de los costos de desarrollo y la mejora del rendimiento [72]. Se definió que se realizarían pruebas de aceptación en las últimas etapas de desarrollo de la solución, utilizando varios casos de prueba para medir que tanto se ajusta la aplicación real a los resultados esperados. Además, se realizarían pruebas de desempeño al agente, dichas pruebas con el objetivo de mesurar el desempeño tanto en el entrenamiento como en la fase de inferencia para seleccionar de entre varias combinaciones de señales de recompensa cual produce mejores resultados.

3.4.1 Pruebas de Aceptación

Las pruebas de aceptación pertenecen a las últimas etapas previas a la liberación en firme de versiones nuevas a fin de determinar si cumplen con las necesidades y/o requerimientos de las empresas y sus usuarios [73]. El enfoque es verificar que todo opere correctamente, en el concepto de que si hay un daño este sea limitado y controlado. A continuación, se exponen las pruebas de aceptación realizadas sobre el software.

Tabla 27. CP iniciar partida individual

Caso de prueba de aceptación	
Número de la HU: 1	**Id de prueba**: 1
Nombre del caso de prueba: CP iniciar partida individual	
Descripción del caso de prueba: Permite al jugador iniciar una partida individual contra el agente	
Condiciones de ejecución: Debe haber iniciado la aplicación y encontrarse en el menú principal	
Entrada/Pasos de ejecución: • Iniciar la aplicación • Presionar el botón “Iniciar Partida”	
Resultado esperado: Se inicia la partida en contra del agente	
Evaluación de la prueba:	

Tabla 28. CP pausar una partida

Caso de prueba de aceptación	
Número de la HU: 2	**Id de prueba**: 2
Nombre del caso de prueba: CP pausar una partida	
Descripción del caso de prueba: Permite al jugador pausar una partida previamente iniciada	
Condiciones de ejecución: Debe haber iniciado la partida	
Entrada/Pasos de ejecución: • Iniciar una partida • Presionar la tecla “Esc”	
Resultado esperado: Se pausa la partida y se muestra el menu de pausa	

Evaluación de la prueba:

Tabla 29. CP reanudar una partida

Caso de prueba de aceptación	
Número de la HU: 2	**Id de prueba**: 3
Nombre del caso de prueba: CP reanudar una partida	
Descripción del caso de prueba: Permite al jugador reanudar una partida previamente pausada	
Condiciones de ejecución: Debe haber pausado la partida	
Entrada/Pasos de ejecución: • Iniciar una partida • Presionar la tecla “Esc” • Presionar el botón “Continuar”	
Resultado esperado: Se reanuda la partida	
Evaluación de la prueba:	

Los restantes casos de prueba se encuentran en los anexos del documento.

Para llevar a cabo las pruebas de aceptación se realizaron tres etapas de pruebas, realizando siete de los casos de pruebas en la primera, los cinco restantes en la segunda, y luego una prueba general donde se verificaron todos los casos de prueba. A continuación, se muestra una gráfica con los resultados de las pruebas.

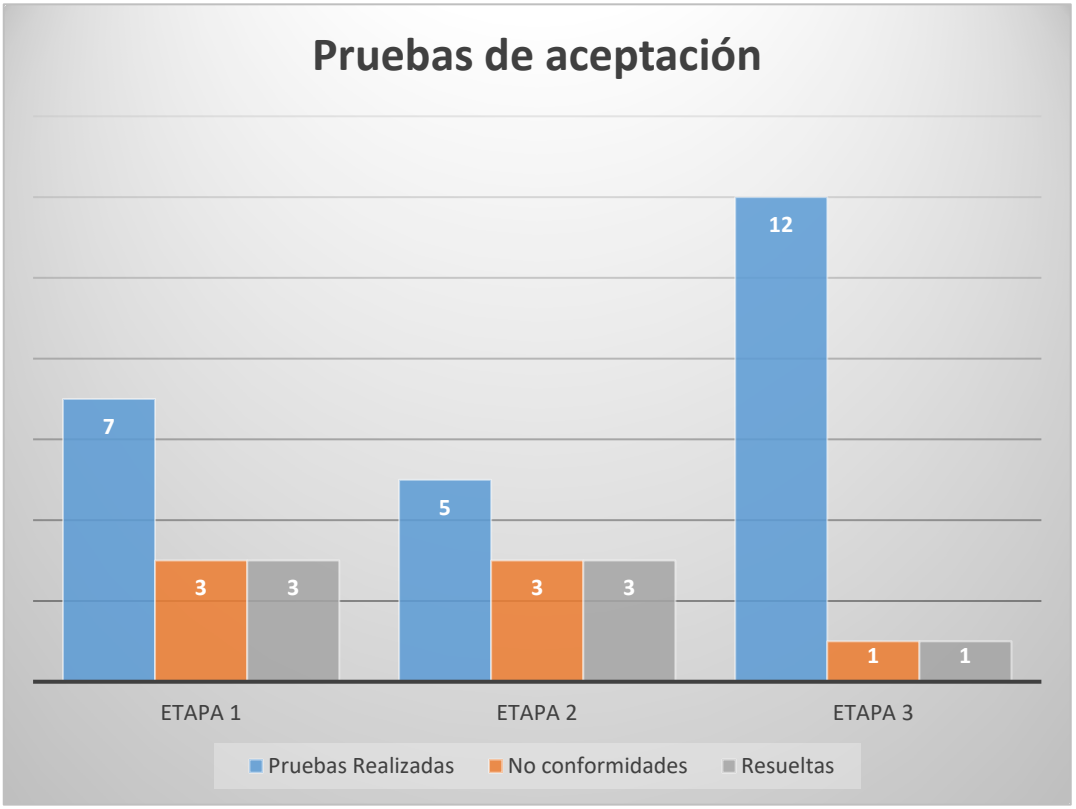


Figura 14. Resultados de las pruebas de aceptación

Se puede apreciar que durante la primera etapa se encontraron 3 no conformidades entre las 7 pruebas realizadas, de las cuales todas fueron resueltas. Al realizarse la segunda etapa, entre las 5 pruebas salieron a la luz 3 no conformidades, que una vez más fueron resueltas. En la tercera y última etapa, se realizaron todos los 12 casos de prueba definidos, y se encontró una no conformidad, surgida a raíz de una de las correcciones de no conformidades de la segunda etapa, la cual fue resuelta. Al resolver todas las no conformidades encontradas en cada etapa se logró hacer entrega de un producto libre de errores.

3.4.2 Pruebas de desempeño del agente

Además de las pruebas de aceptación sobre el demo, se hace necesario de alguna forma medir el desempeño del algoritmo en sí, para ello se realizaron pruebas de desempeño al agente, las cuales fueron divididas en dos fases, coincidiendo con las fases de entrenamiento y de inferencia. Así, de entre las redes entrenada se seleccionarían aquellas de mejores resultados en el entrenamiento y se evaluaría su desempeño real. Gracias a esta combinación de pruebas, el resultado esperado es obtener el agente con el mejor aprendizaje para usar en el demo.

Pruebas sobre el entrenamiento

Medir el desempeño en la fase de entrenamiento es básico para saber que tan bien o mal realiza el agente su trabajo. Una forma de mostrar el rendimiento de un algoritmo de aprendizaje reforzado es graficar la recompensa acumulada (la suma de todas las recompensas recibidas hasta el momento) en función del número de pasos. Un algoritmo domina a otro si su gráfico está consistentemente por encima del otro [74].

Fueron realizadas 8 pruebas, utilizando los mismos hiperparámetros generales, con valores medios según lo recomendado en la documentación del paquete ML-Agents, tratados en el capítulo anterior. La diferencia entre cada prueba radica en las señales de recompensa, pues esto influye directamente en como el agente interpreta el entorno y sus reglas para llegar al resultado deseado. La siguiente tabla resume las señales de recompensa en cada prueba.

Tabla 30. Recompensas por prueba

Número de la Prueba	Recompensas
Prueba 1	+3 al golpear un enemigo -1 al ser golpeado -0,5 al tocar una pared
Prueba 2	+5 al golpear un enemigo

	-1 al ser golpeado -0,5 al tocar una pared -0,05 por cada disparo
Prueba 3	+5 al golpear un enemigo -1 al ser golpeado -0,5 al tocar una pared -0,01 por cada acción
Prueba 4	+5 al golpear un enemigo -1 al ser golpeado
Prueba 5	fijo 1 al ganar fijo -1 al perder -0.1 al golpear una pared
Prueba 6	fijo 1 al ganar fijo -3 al perder -0.1 al golpear una pared
Prueba 7	fijo 10 al ganar fijo -10 al perder -0.5 al golpear una pared +1 al golpear un enemigo -1 al recibir un golpe del enemigo
Prueba 8	fijo 5 al ganar fijo -3 al perder -0.1 al golpear una pared +2 al golpear un enemigo -1 al recibir un golpe del enemigo

Como se puede apreciar, se utilizaron diferentes recompensas, pero siempre manteniendo las recompensas que dan la condición de ganar o perder: infligir o recibir daño, o perder o ganar la partida en sí. Algunas recompensas adicionales fueron utilizadas como “incentivos” para comportamientos determinados, por ejemplo, recompensas negativas al tocar una pared deben inducir el comportamiento de evitar esquinarse al moverse o recompensas negativas por cada acción podrían lograr que el agente intentara obtener recompensa más rápidamente.

Una vez realizados los entrenamientos, gracias al propio paquete ML-Agents se pueden visualizar los resultados, incluso de varios entrenamientos a la vez, en una gráfica, como se muestra a continuación:

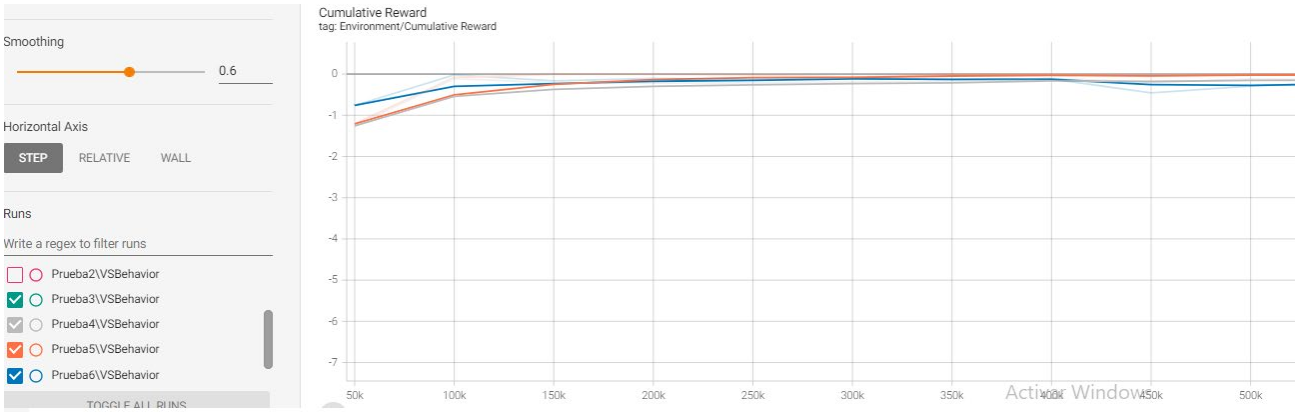


Figura 15. Resultados de pruebas 3, 4, 5 y 6

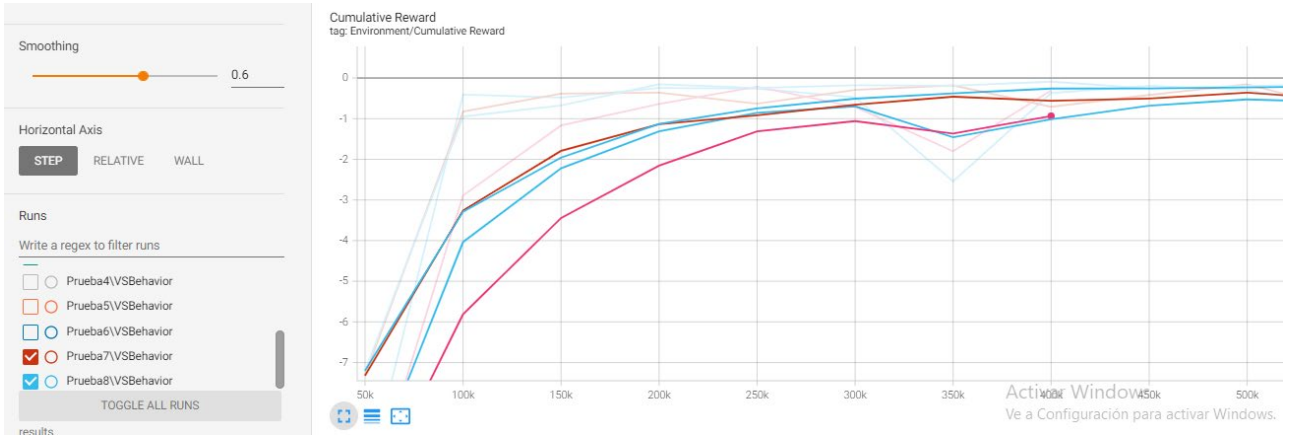


Figura 16. Resultados de pruebas 1, 2, 7 y 8

Se puede apreciar que varios entrenamientos logran un puntaje relativamente alto, cercano a cero ya que al tratarse de un dominio de adversarios y que ambos, el agente y su adversario comparten la misma red y aportan ambos a su aprendizaje, cada uno gana o pierde recompensa como si se tratara de un juego de suma cero. Debido a esto, se toma una puntuación cercana al cero como una buena política.

Según los resultados se descarta enseguida la prueba 6 pues posee recompensas pobres con respecto a los demás. El caso contrario son las pruebas 3, 4 y 5, si bien estas poseen puntajes altos, también comienzan con un puntaje alto por lo que no hay un crecimiento apreciable durante el aprendizaje. Aún si cero es considerado un puntaje alto, estos resultados no concuerdan con un entrenamiento adecuado, más bien indican que solo sin altos debido a la configuración de las propias recompensas. Las pruebas 1, 2, 7 y 8 logran llegar a las recompensas altas relativamente rápidos y tienen una curva que muestra un aprendizaje desde recompensas relativamente bajas hasta las más significativas, insinuando un aprendizaje exitoso, sin embargo, la prueba 2 muestra

un desempeño relativamente menor a las demás, así que fue descartada. Se decide entonces realizar pruebas de inferencia sobre las redes entrenadas durante las pruebas 1, 7 y 8.
Cabe destacar, que si bien el número de pasos puede parecer grande, con ciento de miles de estos, los entrenamientos tuvieron una duración que no superaba los 50 minutos, lo cual es un tiempo relativamente corto. En cuanto a los requerimientos, mencionados varias veces antes, no son para nada altos. El demo y sus consiguientes pruebas fueron realizadas en un i5 de cuarta generación con 8GB de RAM sin una GPU dedicada, máquina que podía manejar el entrenamiento bastante bien. Teniendo en cuenta que esas especificaciones son más bien bajo rendimiento o *low-end* para los estándares actuales, y probablemente para los del centro VERTEX, se afirma que se cumple el objetivo secundario o requisitos no funcionales que implican bajos requerimientos de implementación.

Pruebas de inferencia

En la práctica, muchos problemas interesantes no tienen un límite superior conocido en los totales o promedios de las recompensas. Para esos problemas, normalmente lo mejor que puede hacer es comparar entre agentes. Se puede comparar con:

- Un agente que actúa al azar. Esto normalmente sería solo como una línea de base, para mostrar que el agente había aprendido algo.
- Un agente automatizado que usa una heurística de elección de acción simple, que podría ser algo natural u obvio en el problema dado.
- Uno o más humanos en la misma tarea.
- Otros agentes capacitados en ML, incluidas instancias anteriores del mismo agente

Usualmente se desea realizar varias pruebas y promediar los resultados, si la política o el entorno son estocásticos, para evaluar un agente con los valores esperados tanto como sea posible. Como la idea del demo es enfrentar al agente contra humanos reales, este método fue el elegido. Se utilizaron diversos jugadores humanos, los cuales realizaron varias partidas cada uno contra cada una de los agentes entrenado, registrándose las victorias y derrotas del mismo, así como la efectividad al esquivar (golpes recibidos/golpes efectuados por el jugador). Como se especificó anteriormente, el objetivo es intentar mesurar que tan bien se desempeña el agente en el entorno para el que fue diseñado. El término “precisión” en aprendizaje automático es subjetivo, sin embargo, se puede decir que cualquier modelo con más de 70% de precisión muestra un nivel de desempeño importante. De hecho, una medida de precisión entre 70% y 90% no es solo ideal, sino realista[75]. Se tomaron datos de 20 partidas jugadas contra cada agente usando las redes elegidas. Los resultados de las victorias y derrotas se muestran a continuación:

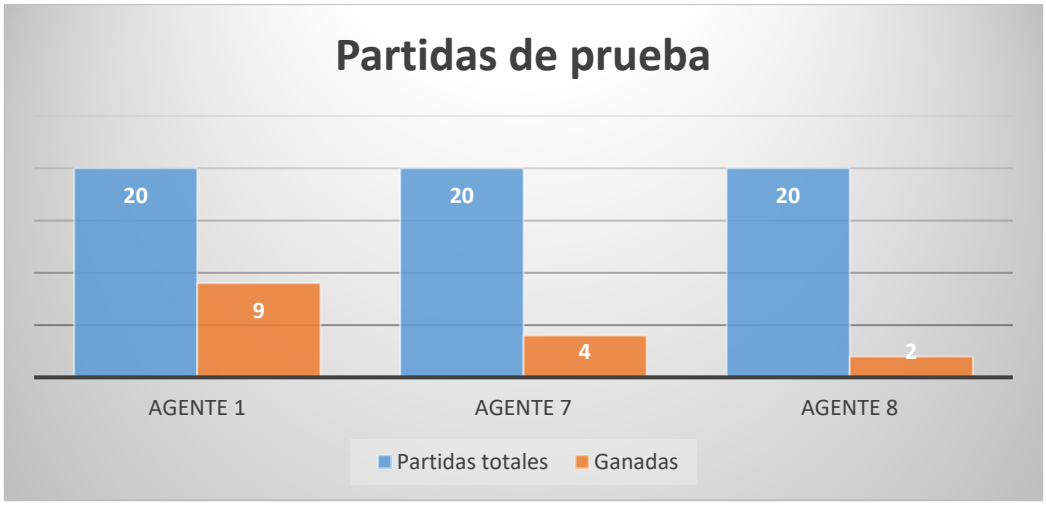


Figura 17. Resultados de pruebas de inferencia

Se puede apreciar que el Agente 1 mostró una clara superioridad en cuanto a partidas ganadas contra jugadores humanos con respecto a los otros dos. En cuanto a la precisión, los datos arrojaron que, en estas partidas, el jugador necesitó una media de aproximadamente 7 disparos para acabar con una vida del agente 1, por tanto, es capaz de esquivar cerca del 85% de los disparos. Por su parte el agente 7 mostró una esquiva de aproximadamente un 75% y el agente 8 de 78%. Los tres agentes muestran una esquiva alta, siendo una vez más el agente 1 quien se destaca, teniendo, al menos defensivamente, una alta efectividad. Si tomamos en cuenta que el agente 1 ganó cerca de la mitad de las partidas, y que aun perdiendo tenía un desempeño considerable, se puede decir que cumple el objetivo de lograr que el agente pueda desempeñarse bien en contra de jugadores humanos.

Durante cada una de las pruebas los agentes mostraron buena fluidez en sus movimientos en todo momento, manteniendo una experiencia de usuario de calidad, por tanto el requisito no funcional relativo a los tiempos de espera fue cumplido con creces.

Como nota adicional, el agente 7 mostraba un comportamiento enfocado en moverse a la parte izquierda del entorno. El agente 8 poseía uno similar, pero en ocasiones se movía al lado derecho. El agente 1 era más heterogéneo en sus movimientos. Para los tres agentes, no presentaban conductas especialmente agresivas, más bien defensivas, priorizando esquivar ataques a ir a atacar deliberadamente.

3.5 Conclusiones del capítulo

En este capítulo fueron abordados los temas referentes a la implementación y validación y prueba de la solución. Las pruebas efectuadas arrojaron diversos resultados, que permitieron concluir que diferentes recompensas pueden lograr comportamientos muy diferentes en los agentes. No siempre los resultados de entrenamiento logran plasmar del todo que tan bien o no se desempeña un agente, como se pudo apreciar con los agentes 1, 7 y 8 que, aun obteniendo gráficas de recompensas similares, poseían bastantes diferencias en sus desempeños. El agente 8 presentó los mejores resultados en general, con un buen ratio de partidas ganadas contra jugadores humanos y una esquiva sobresaliente con respecto a los demás, logrando su máximo desempeño a partir de los quinientos mil pasos, lo cual si bien parece mucho, nunca tomó más de 40 minutos al entrenar, un tiempo relativamente corto. Además, tanto el entrenamiento como el comportamiento del agente en el juego cumplen con los requisitos no funcionales planteados en el capítulo anterior.

Conclusiones generales

Durante el transcurso de la investigación y el proceso de desarrollo de la solución propuesta han sido logrados los objetivos propuestos, arribándose a las siguientes conclusiones:

- El desarrollo del demo demuestra que los algoritmos de aprendizaje reforzado pueden ser utilizados en los videojuegos, con un desempeño considerable.
- El algoritmo PPO destaca en su empleo en el aprendizaje reforzado en videojuegos, por encima de otros, debido a su gran desempeño en casi cualquier entorno y su facilidad de implementación.
- Las CNN son ideales para su utilización en videojuegos, más allá de su habitual uso en el reconocimiento de imágenes.
- Es posible ajustar el comportamiento de los agentes hasta cierto punto, modificando las recompensas, sin embargo, puede llegar a ser complicado lograr que tengan el comportamiento deseado más allá de lograr los resultados deseados.
- Por su difícil manejo, el inherente tiempo de entrenamiento que conllevan y lo complicado de su trazabilidad, puede ser todo un reto incluirlas en el proceso de desarrollo de videojuegos, en dependencia de las características de cada uno.

Recomendaciones

Con base en la investigación realizada, la implementación y prueba de la solución y las conclusiones obtenidas, se plantea la siguiente recomendación:

- Lograr un entrenamiento que resulte en una postura más ofensiva del agente, ya que usualmente adoptaba una defensiva. Podría lograrse con cambios más radicales en las recompensas, o incluso en los hiperparámetros, la red neuronal utilizada o el algoritmo empleado.

Referencias Bibliográficas

[1] D. L. Poole, A. K. Mackworth, y R. Goebel, *Computational intelligence: a logical approach*. New York: Oxford University Press, 1998. [En línea]. Disponible en: https://www.cs.ubc.ca/~poole/ci/front.pdf

[2] M. I. Jordan y T. M. Mitchell, «Machine learning: Trends, perspectives, and prospects», *Science*, vol. 349, n.º 6245, pp. 255-260, jul. 2015, doi: 10.1126/science.aaa8415.

[3] I. El Naqa y M. J. Murphy, «What Is Machine Learning?», en *Machine Learning in Radiation Oncology: Theory and Applications*, I. El Naqa, R. Li, y M. J. Murphy, Eds. Cham: Springer International Publishing, 2015, pp. 3-11. doi: 10.1007/978-3-319-18305-3_1.

[4] L. P. Kaelbling, M. L. Littman, y A. W. Moore, «Reinforcement Learning: A Survey». arXiv, 30 de abril de 1996. Accedido: 21 de septiembre de 2022. [En línea]. Disponible en: http://arxiv.org/abs/cs/9605103

[5] R. S. Sutton y A. Barto, *Reinforcement learning: an introduction*, Second edition. Cambridge, Massachusetts London, England: The MIT Press, 2018.

[6] «La inteligencia de Cuba en la Inteligencia Artificial», *Universidad Central «Marta Abreu» de Las Villas*, 5 de septiembre de 2021. https://www.uclv.edu.cu/la-inteligencia-de-cuba-en-la-inteligencia-artificial/ (accedido 7 de noviembre de 2022).

[7] «COSMOX | Videojuego Coliseum», 6 de noviembre de 2018. https://cosmox.uci.cu/frontend/web/videogame/40 (accedido 7 de noviembre de 2022).

[8] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[9] K. L. Wagstaff, «Machine Learning that Matters», p. 6.

[10] P. Cunningham, M. Cord, y S. J. Delany, «Supervised Learning», en *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*, M. Cord y P. Cunningham, Eds. Berlin, Heidelberg: Springer, 2008, pp. 21-49. doi: 10.1007/978-3-540-75171-7_2.

[11] Z. Ghahramani, «Unsupervised Learning», en *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*, O. Bousquet, U. von Luxburg, y G. Rätsch, Eds. Berlin, Heidelberg: Springer, 2004, pp. 72-112. doi: 10.1007/978-3-540-28650-9_5.

[12] A. Suran, «On-Policy v/s Off-Policy Learning», *Medium*, 14 de julio de 2020. https://towardsdatascience.com/on-policy-v-s-off-policy-learning-75089916bc2f (accedido 21 de septiembre de 2022).

[13] «Unity ML-Agents Toolkit Overview». Unity Technologies, 21 de septiembre de 2022. Accedido: 21 de septiembre de 2022. [En línea]. Disponible en: https://github.com/Unity-Technologies/ml-agents/blob/da1de79ff205fbc5f17e332f3ed030faacd9a565/docs/ML-Agents-Overview.md

[14] TensorFlow Staff, «Búferes de reproducción | TensorFlow Agents», *TensorFlow*. https://www.tensorflow.org/agents/tutorials/5_replay_buffers_tutorial?hl=es-419 (accedido 21 de septiembre de 2022).

[15] «Papers with Code - Experience Replay Explained». https://paperswithcode.com/method/experience-replay (accedido 21 de septiembre de 2022).

[16] D. Pathak, P. Agrawal, A. A. Efros, y T. Darrell, «Curiosity-Driven Exploration by Self-Supervised Prediction», en *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Honolulu, HI, USA, jul. 2017, pp. 488-489. doi: 10.1109/CVPRW.2017.70.

[17] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, y R. Munos, «Unifying Count-Based Exploration and Intrinsic Motivation». arXiv, 7 de noviembre de 2016. Accedido: 21 de septiembre de 2022. [En línea]. Disponible en: http://arxiv.org/abs/1606.01868

[18] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, y P. Abbeel, «VIME: Variational Information Maximizing Exploration». arXiv, 27 de enero de 2017. Accedido: 21 de septiembre de 2022. [En línea]. Disponible en: http://arxiv.org/abs/1605.09674

[19] P. Picton, «What is a Neural Network?», en *Introduction to Neural Networks*, P. Picton, Ed. London: Macmillan Education UK, 1994, pp. 1-12. doi: 10.1007/978-1-349-13530-1_1.

[20] S.-C. Wang, «Artificial Neural Network», en *Interdisciplinary Computing in Java Programming*, S.-C. Wang, Ed. Boston, MA: Springer US, 2003, pp. 81-100. doi: 10.1007/978-1-4615-0377-4_5.

[21] IBM Cloud Education, «¿Qué son las redes neuronales?», 26 de agosto de 2021. https://www.ibm.com/es-es/cloud/learn/neural-networks (accedido 21 de septiembre de 2022).

[22] K. O'Shea y R. Nash, «An Introduction to Convolutional Neural Networks». arXiv, 2 de diciembre de 2015. Accedido: 21 de septiembre de 2022. [En línea]. Disponible en: http://arxiv.org/abs/1511.08458

[23] IBM Cloud Education, «What are Recurrent Neural Networks?», 7 de abril de 2021. https://www.ibm.com/cloud/learn/recurrent-neural-networks (accedido 21 de septiembre de 2022).

[24] L. Deng, «Deep Learning: Methods and Applications», *Found. Trends® Signal Process.*, vol. 7, n.º 3-4, pp. 197-387, 2014, doi: 10.1561/2000000039.

[25] G. Boesch, «Deep Reinforcement Learning: How It Works and Real World Examples», *viso.ai*, 23 de mayo de 2022. https://viso.ai/deep-learning/deep-reinforcement-learning/ (accedido 21 de septiembre de 2022).

[26] R. Kwiatkowski, «Gradient Descent Algorithm — a deep dive | Towards Data Science». https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21 (accedido 21 de septiembre de 2022).

[27] J. Peters, «Policy gradient methods - Scholarpedia». http://www.scholarpedia.org/article/Policy_gradient_methods?source=post_page-----acc0344f5d72---------) (accedido 21 de septiembre de 2022).

[28] J. Schulman, O. Klimov, F. Wolski, y P. Dhariwal, «Proximal Policy Optimization», *OpenAI*, 20 de julio de 2017. https://openai.com/blog/openai-baselines-ppo/ (accedido 21 de septiembre de 2022).

[29] S. B. Hall, «COVID-19 is taking gaming and esports to the next level», *World Economic Forum*. https://www.weforum.org/agenda/2020/05/covid-19-taking-gaming-and-esports-next-level/ (accedido 21 de septiembre de 2022).

[30] H. HarperCollins, «The American Heritage Dictionary entry: fifth». https://ahdictionary.com/word/search.html?q=fifth (accedido 21 de septiembre de 2022).

[31] J. Stenros, «The Game Definition Game: A Review», *Games Cult.*, 2016, doi: 10.1177/1555412016655679.

[32] Á. Parra de Miguel, «Búsqueda de caminos en mapas de videojuegos: desarrollo de técnicas de búsqueda de caminos con preprocesamiento para mapas de videojuegos». https://e-archivo.uc3m.es/handle/10016/23954 (accedido 22 de septiembre de 2022).

[33] L. González, «Inteligencia Artificial en los Videojuegos», *Aprende IA*, 15 de diciembre de 2020. https://aprendeia.com/inteligencia-artificial-en-los-videojuegos/ (accedido 22 de septiembre de 2022).

[34] C. Cuza Soca, «Videojuego El mundo de Wumpus basado en técnicas de Inteligencia Artificial.», 2019, Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: https://repositorio.uci.cu/jspui/handle/123456789/10120

[35] A. Puigdomènech *et al.*, «Agent57: Outperforming the human Atari benchmark». https://www.deepmind.com/blog/agent57-outperforming-the-human-atari-benchmark (accedido 22 de septiembre de 2022).

[36] «OpenAI Five», *OpenAI*, 13 de diciembre de 2019. https://openai.com/five/ (accedido 22 de septiembre de 2022).

[37] The Alpha Star Team, O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, y A. Dudzik, «AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning». https://www.deepmind.com/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning (accedido 22 de septiembre de 2022).

[38] «Metodologías de desarrollo de software | Universitat Carlemany». https://www.universitatcarlemany.com/actualidad/metodologias-de-desarrollo-de-software (accedido 22 de septiembre de 2022).

[39] R. Figueroa-Diaz, C. Sólis, y A. Cabrera, *METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES*. 2007. doi: 10.13140/RG.2.1.2897.3206.

[40] K. Schwaber y J. Sutherland, «The Scrum Guide», *Scrum.org*. https://www.scrum.org/resources/scrum-guide (accedido 22 de septiembre de 2022).

[41] L. Castellano Lendínez, «Kanban. Metodología para aumentar la eficiencia de los procesos», *3C Tecnol. Innov. Apl. Pyme*, vol. 29, n.º 1, pp. 30-41, mar. 2019, doi: 10.17993/3ctecno/2019.v8n1e29/30-41.

[42] G. F. Escribano, «Introducción a Extreme Programming», p. 14.

[43] J. Fan, Z. Wang, Y. Xie, y Z. Yang, «A Theoretical Analysis of Deep Q-Learning», en *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, jul. 2020, pp. 486-489. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: https://proceedings.mlr.press/v120/yang20a.html

[44] M. Wang, «Deep Q-Learning Tutorial: minDQN. A Practical Guide to Deep Q-Networks | Towards Data Science». https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc (accedido 22 de septiembre de 2022).

[45] S. Y. Lee, C. Sungik, y S.-Y. Chung, «Sample-Efficient Deep Reinforcement Learning via Episodic Backward Update», en *Advances in Neural Information Processing Systems*, 2019, vol. 32. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: https://proceedings.neurips.cc/paper/2019/hash/e6d8545daa42d5ced125a4bf747b3688-Abstract.html

[46] Z. Wang *et al.*, «Sample Efficient Actor-Critic with Experience Replay». arXiv, 10 de julio de 2017. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: http://arxiv.org/abs/1611.01224

[47] S. Kullback y R. A. Leibler, «On Information and Sufficiency», *Ann. Math. Stat.*, vol. 22, n.º 1, pp. 79-86, mar. 1951, doi: 10.1214/aoms/1177729694.

[48] «Papers with Code - TRPO Explained». https://paperswithcode.com/method/trpo (accedido 22 de septiembre de 2022).

[49] E. Zuehlke, «Conjugate gradient methods - optimization». https://optimization.mccormick.northwestern.edu/index.php/Conjugate_gradient_methods (accedido 22 de septiembre de 2022).

[50] S. Karagiannakos, «Trust Region and Proximal policy optimization (TRPO and PPO)», *AI Summer*, 11 de enero de 2019. https://theaisummer.com/TRPO_PPO/ (accedido 22 de septiembre de 2022).

[51] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, y P. Abbeel, «Trust Region Policy Optimization». arXiv, 20 de abril de 2017. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: http://arxiv.org/abs/1502.05477

[52] M. Stojiljkovic, «Stochastic Gradient Descent Algorithm With Python and NumPy – Real Python». https://realpython.com/gradient-descent-algorithm-python/ (accedido 22 de septiembre de 2022).

[53] S. Gujar, «Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO)| Medium». https://medium.com/@sanketgujar95/trust-region-policy-optimization-trpo-and-proximal-policy-optimization-ppo-e6e7075f39ed (accedido 22 de septiembre de 2022).

[54] T. Schaul, J. Quan, I. Antonoglou, y D. Silver, «Prioritized Experience Replay». arXiv, 25 de febrero de 2016. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: http://arxiv.org/abs/1511.05952

[55] V. Mnih *et al.*, «Asynchronous Methods for Deep Reinforcement Learning». arXiv, 16 de junio de 2016. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: http://arxiv.org/abs/1602.01783

[56] R. Munos, T. Stepleton, A. Harutyunyan, y M. G. Bellemare, «Safe and Efficient Off-Policy Reinforcement Learning». arXiv, 7 de noviembre de 2016. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: http://arxiv.org/abs/1606.02647

[57] T. Haarnoja, V. Pong, K. Hartikainen, A. Zhou, M. Dalal, y S. Levine, «Soft Actor Critic—Deep Reinforcement Learning with Real-World Robots», *The Berkeley Artificial Intelligence Research Blog*. http://bair.berkeley.edu/blog/2018/12/14/sac/ (accedido 22 de septiembre de 2022).

[58] V. V. Kumar, «Soft Actor-Critic Demystified. An intuitive explanation of the theory…Towards Data Science». https://towardsdatascience.com/soft-actor-critic-demystified-b8427df61665 (accedido 22 de septiembre de 2022).

[59] T. Haarnoja, A. Zhou, P. Abbeel, y S. Levine, «Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor». arXiv, 8 de agosto de 2018. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: http://arxiv.org/abs/1801.01290

[60] L. Ogoti, «Why Python is Good for Machine Learning», *Engineering Education (EngEd) Program | Section*. https://www.section.io/engineering-education/why-python-is-good-for-machine-learning/ (accedido 22 de septiembre de 2022).

[61] T. Bradshaw y M. Kruppa, «Epic and Unity rev their engines for the next era of entertainment», *Financial Times*, 12 de agosto de 2020.

[62] «About Visual Paradigm». https://www.visual-paradigm.com/aboutus/ (accedido 22 de septiembre de 2022).

[63] V.-P. Berges, C. Elion, A. Juliani, y A. Bhatnagar, «Unity ML-Agents Toolkit Training ML-Agents». Unity Technologies, 22 de septiembre de 2022. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en:

https://github.com/Unity-Technologies/ml-agents/blob/da1de79ff205fbc5f17e332f3ed030faacd9a565/docs/Training-ML-Agents.md

[64] V.-P. Berges, C. Elion, A. Juliani, y A. Bhatnagar, «Unity ML-Agents Toolkit Agents». Unity Technologies, 22 de septiembre de 2022. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: https://github.com/Unity-Technologies/ml-agents/blob/da1de79ff205fbc5f17e332f3ed030faacd9a565/docs/Learning-Environment-Design-Agents.md

[65] D. Bustamante y J. C. Rodríguez, «Metodologia XP». Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: https://luismejias21.files.wordpress.com/2018/03/metodologia-xp.pdf

[66] R. Turner, *The Foundations of Specification*, vol. 15. Journal of Logic and Computation, 2005. Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: https://www.researchgate.net/publication/31374075_The_Foundations_of_Specification#read

[67] P. Quile y D. Isaías, «Sistema informático para emprendimientos en la Facultad de Ciencia e Ingeniería en Alimentos de la Universidad Técnica de Ambato y comunidad», abr. 2019, Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: https://repositorio.uta.edu.ec:8443/jspui/handle/123456789/29661

[68] S. M. MELÉNDEZ VALLADAREZ, M. E. GAITAN, y N. N. PÉREZ REYES, «Metodología Ágil Programación Extrema XP», p. 146.

[69] K. Beck y W. Cunningham, «A Laboratory For Teaching Object-Oriented Thinking», p. 7.

[70] A. Kiken, «Estándares de codificación de software y pautas de programación», 24 de abril de 2020. https://es.parasoft.com/blog/an-ounce-of-prevention-software-safety-security-through-coding-standards/ (accedido 27 de octubre de 2022).

[71] E. M. G. Giraldo, «Sistema multi-agente deliberativo para la obtención y análisis de datos en Honeynets», 2016.

[72] «¿Qué son las pruebas de software y cómo funcionan? | IBM». https://www.ibm.com/cl-es/topics/software-testing (accedido 27 de octubre de 2022).

[73] J. Ellingwood, «An Introduction to Continuous Integration, Delivery, and Deployment | DigitalOcean», 10 de mayo de 2017. https://web.archive.org/web/20170924041107/https://www.digitalocean.com/community/tutorials/an-introduction-to-continuous-integration-delivery-and-deployment (accedido 27 de octubre de 2022).

[74] «Evaluating Reinforcement Learning Algorithms▸ Chapter 12 Learning to Act ▸ Artificial Intelligence: Foundations of Computational Agents, 2nd Edition», 3 de noviembre de 2018. https://artint.info/2e/html/ArtInt2e.Ch12.S6.html (accedido 27 de octubre de 2022).

[75] K. Barkved, «How To Know if Your Machine Learning Model Has Good Performance | Obviously AI», 9 de marzo de 2022. https://www.obviously.ai/post/machine-learning-model-performance (accedido 27 de octubre de 2022).

Anexos

1. Historias de Usuario:

HU 1: Modo de juego individual

HISTORIA DE USUARIO	
Número: 1	**Usuario**: jugador
Nombre: Modo de juego individual	
Prioridad: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 1.5	**Iteración**: 2
Descripción: el usuario debe tener la opción de empezar una partida individual, donde la IA es capaz de realizar acciones para derrotar al jugador.	

HU 2: Menú principal

HISTORIA DE USUARIO	
Número: 2	**Usuario**: jugador
Nombre: Menú principal	
Prioridad: Media	**Riesgo en Desarrollo**: Media
Puntos Estimados: 1	**Iteración**: 2
Descripción: el demo debe contar con un menú principal que contenga las opciones pertinentes al inicio de una partida.	

HU 3: Implementar el Entorno

HISTORIA DE USUARIO	
Número: 3	**Usuario**: jugador
Nombre: Implementar el Entono	
Prioridad: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 0.5	**Iteración**: 1
Descripción: el entorno debe contener a las naves tanto la del jugador como las enemigas, así como delimitar su movimiento.	

HU 4: Movimiento y disparos de la nave

HISTORIA DE USUARIO	
Número: 3	**Usuario**: jugador
Nombre: Movimiento y disparos de la nave	
Prioridad: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 0.5	**Iteración**: 1
Descripción: el usuario debe poder mover su nave asignada en los ejes horizontal y vertical, además de poder disparar al pulsar la tecla “Espacio”	

HU 5: Acciones del agente

HISTORIA DE USUARIO

Número: 4	**Usuario**: jugador
Nombre: Acciones del agente	
Prioridad: Alta	**Riesgo en Desarrollo**: Alta
Puntos Estimados: 2	**Iteración**: 1
Descripción: el agente debe aprender por si mismo a realizar las mismas acciones que el jugador de manera eficiente	

HU 6: Menú de pausa

HISTORIA DE USUARIO	
Número: 5	**Usuario**: jugador
Nombre: *Menú de pausa*	
Prioridad: Baja	**Riesgo en Desarrollo**: Baja
Puntos Estimados: 0.5	**Iteración:** 2
Descripción: el demo debe permitir al jugador pausar la partida, y reanudarla o acceder al menú principal	

HU 7: Seleccionar dificultad de la IA

HISTORIA DE USUARIO	
Número: 6	**Usuario**: jugador
Nombre: Seleccionar dificultad de la IA	
Prioridad: Baja	**Riesgo en Desarrollo**: Baja
Puntos Estimados: 0.5	**Iteración:** 2
Descripción: permitir al usuario seleccionar el nivel de dificultad que aportará la IA a la partida.	

Casos de Prueba:

Caso de Prueba 1. CP iniciar partida individual

Caso de prueba de aceptación	
Número de la HU: 1	**Id de prueba**: 1
Nombre del caso de prueba: CP iniciar partida individual	
Descripción del caso de prueba: Permite al jugador iniciar una partida individual contra el agente	
Condiciones de ejecución: Debe haber iniciado la aplicación y encontrarse en el menú principal	
Entrada/Pasos de ejecución: • Iniciar la aplicación • Presionar el botón “Iniciar Partida”	
Resultado esperado: Se inicia la partida en contra del agente	
Evaluación de la prueba:	

Caso de Prueba 2. CP pausar una partida

Caso de prueba de aceptación	
Número de la HU: 2	**Id de prueba**: 2
Nombre del caso de prueba: CP pausar una partida	
Descripción del caso de prueba: Permite al jugador pausar una partida previamente iniciada	
Condiciones de ejecución: Debe haber iniciado la partida	
Entrada/Pasos de ejecución: • Iniciar una partida • Presionar la tecla “Esc”	
Resultado esperado: Se pausa la partida y se muestra el menu de pausa	
Evaluación de la prueba:	

Caso de Prueba 3. CP reanudar una partida

Caso de prueba de aceptación	
Número de la HU: 2	**Id de prueba**: 3
Nombre del caso de prueba: CP reanudar una partida	
Descripción del caso de prueba: Permite al jugador reanudar una partida previamente pausada	
Condiciones de ejecución: Debe haber pausado la partida	
Entrada/Pasos de ejecución: • Iniciar una partida • Presionar la tecla “Esc” • Presionar el botón “Continuar”	
Resultado esperado: Se reanuda la partida	
Evaluación de la prueba:	

Caso de Prueba 4. CP ir al menú principal

Caso de prueba de aceptación	
Número de la HU: 2	**Id de prueba**: 4
Nombre del caso de prueba: CP ir al menú principal	
Descripción del caso de prueba: Permite al jugador ir al menú principal desde el menú de pausa en una partida	
Condiciones de ejecución: Debe haber pausado la partida	
Entrada/Pasos de ejecución: • Iniciar una partida • Presionar la tecla “Esc”	

• Presionar el botón “Menú principal”
Resultado esperado: Se cierra la partida y se regresa al menú principal
Evaluación de la prueba:

Caso de Prueba 5. CP seleccionar dificultad media

Caso de prueba de aceptación	
Número de la HU: 7	**Id de prueba**: 5
Nombre del caso de prueba: CP seleccionar dificultad media	
Descripción del caso de prueba: Permite al jugador seleccionar la dificultad “Medio” al enfrentarse al agente	
Condiciones de ejecución: Debe haber iniciado la aplicación y encontrarse en el menú principal	
Entrada/Pasos de ejecución: • Iniciar la aplicación • Seleccionar la opción “Medio” en la sección de “Selección de dificultad” • Presionar el botón aceptar • Iniciar una partida individual	
Resultado esperado: Se inicia la partida en contra del agente, con un agente presentando una dificultad moderada	
Evaluación de la prueba:	

Caso de Prueba 6. CP seleccionar dificultad fácil

Caso de prueba de aceptación	
Número de la HU: 7	**Id de prueba**: 6
Nombre del caso de prueba: CP seleccionar dificultad fácil	
Descripción del caso de prueba: Permite al jugador seleccionar la dificultad “Fácil” al enfrentarse al agente	
Condiciones de ejecución: Debe haber iniciado la aplicación y encontrarse en el menú principal	
Entrada/Pasos de ejecución: • Iniciar la aplicación • Seleccionar la opción “Fácil” en la sección de “Selección de dificultad” • Presionar el botón aceptar • Iniciar una partida individual	
Resultado esperado: Se inicia la partida en contra del agente, con un agente presentando una dificultad disminuida	
Evaluación de la prueba:	

Caso de Prueba 7. CP seleccionar dificultad difícil

Caso de prueba de aceptación	
Número de la HU: 7	**Id de prueba**: 7
Nombre del caso de prueba: CP seleccionar dificultad difícil	
Descripción del caso de prueba: Permite al jugador seleccionar la dificultad “Difícil” al enfrentarse al agente	
Condiciones de ejecución: Debe haber iniciado la aplicación y encontrarse en el menú principal	
Entrada/Pasos de ejecución: • Iniciar la aplicación • Seleccionar la opción “Difícil” en la sección de “Selección de dificultad” • Presionar el botón aceptar • Iniciar una partida individual	
Resultado esperado: Se inicia la partida en contra del agente, con un agente presentando una dificultad incrementada	
Evaluación de la prueba:	

Caso de Prueba 8. CP ganar una partida

Caso de prueba de aceptación	
Número de la HU: 1	**Id de prueba**: 8
Nombre del caso de prueba: CP ganar una partida	
Descripción del caso de prueba: Debe mostrar un mensaje de “Ha Ganado” luego de bajar la vida del enemigo	
Condiciones de ejecución: Debe haber iniciado una partida y haber eliminado tres vidas del enemigo	
Entrada/Pasos de ejecución: • Iniciar una partida • Golpear al enemigo tres veces, eliminando sus vidas	
Resultado esperado: Muestra el mensaje “Ha Ganado” y muestra un menú	
Evaluación de la prueba:	

Caso de Prueba 9. CP perder una partida

Caso de prueba de aceptación	
Número de la HU: 1	**Id de prueba**: 9
Nombre del caso de prueba: CP perder una partida	
Descripción del caso de prueba: Debe mostrar un mensaje de “Ha Perdido” luego de que el enemigo baje la vida del jugador	

Condiciones de ejecución: Debe haber iniciado una partida y haber perdido tres vidas
Entrada/Pasos de ejecución: • Iniciar una partida • Ser golpeado tres veces, perdiendo las vidas
Resultado esperado: Muestra el mensaje “Ha Perdido” y muestra un menú
Evaluación de la prueba:

Caso de Prueba 10. CP reiniciar una partida acabada

Caso de prueba de aceptación	
Número de la HU: 1	**Id de prueba**: 10
Nombre del caso de prueba: CP reiniciar una partida acabada	
Descripción del caso de prueba: Permite al jugador reiniciar una partida que acabó	
Condiciones de ejecución: Debe haber acabado una partida individual	
Entrada/Pasos de ejecución: • Derrotar al agente o ser derrotado por este • Presionar el botón “Reinicar”	
Resultado esperado: Se reiniciar la partida	
Evaluación de la prueba:	

Caso de Prueba 11. CP volver al menú principal

Caso de prueba de aceptación	
Número de la HU: 1	**Id de prueba**: 11
Nombre del caso de prueba: CP volver al menú principal	
Descripción del caso de prueba: Permite al jugador volver al menú principal luego de acabar una partida	
Condiciones de ejecución: Debe haber acabado una partida individual	
Entrada/Pasos de ejecución: • Derrotar al agente o ser derrotado por este • Presionar el botón “Menú principal”	
Resultado esperado: Se regresa al menú principal	
Evaluación de la prueba:	

Caso de Prueba 12. CP salir del juego

Caso de prueba de aceptación	
Número de la HU: 1	**Id de prueba**: 12
Nombre del caso de prueba: CP salir del juego	

Descripción del caso de prueba: Permite al jugador cerrar la aplicación desde el menú principal
Condiciones de ejecución: Debe encontrarse en el menú principal
Entrada/Pasos de ejecución: • Presionar el botón “Salir”
Resultado esperado: Se cierra la aplicación
Evaluación de la prueba: