Aprendizaje de Comportamientos en el Juego de Pac-Man
Behaviour Learning in Pac-Man games

Trabajo de Fin de Grado
Curso 2024–2025

Autores

Diego González López, Nota:10
Alejandro Bejarano del Castillo, Nota: 10
Daniel Jiménez Rojo, Nota: 10
Daniel de la Fuente Díez, Nota: 10

Director

Belén Díaz Agudo
Juan Antonio Recio García

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Aprendizaje de Comportamientos en el Juego de Pac-Man
Behaviour Learning in Pac-Man games

Trabajo de Fin de Grado en Ingeniería Informática

Autores

Diego González López, Nota:10
Alejandro Bejarano del Castillo, Nota: 10
Daniel Jiménez Rojo, Nota: 10
Daniel de la Fuente Díez, Nota: 10

Director

Belén Díaz Agudo
Juan Antonio Recio García

Convocatoria: *Junio 2025*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

18 de junio de 2025

Dedicatoria

A nuestras familias, por ser la base de todo. A nuestros compañeros y profes, por los aprendizajes y desafíos compartidos. A los que nos vieron crecer entre apuntes, madrugones, dudas y logros. Este trabajo marca un final, pero también un principio. Gracias por hacerlo con nosotros.

Agradecimientos

A Belén y Juan Antonio por habernos guiado durante todo este camino.

A los alumnos de ICI por habernos permitido utilizar sus trabajos.

A Jero, por guiarnos con paciencia y saber ayudarnos siempre que lo hemos necesitado.

Resumen

Aprendizaje de Comportamientos en el Juego de Pac-Man

El presente trabajo explora el uso de técnicas de aprendizaje supervisado para predecir el comportamiento del agente Pac-Man frente a distintos escenarios del juego. Para ello, se ha utilizado como entorno el juego Ms. Pac-Man Vs Ghosts, sobre el cual se ha generado un conjunto de datos estructurado en función de intersecciones y movimientos.

Se han entrenado y comparado modelos como MLP, TabNet y algoritmos clásicos de Scikit-learn, donde se han aplicado mejoras mediante ingeniería de características y análisis de importancia de variables.

El objetivo final ha sido no solo obtener un modelo preciso, sino también explicable, capaz de justificar sus decisiones en cada situación de juego.

El código de este proyecto es accesible a través de este enlace:

https://github.com/beja28/Ms.Pacman-Machine-Learning

Palabras clave

1. Red neuronal
2. Pac-Man
3. Fantasma
4. Explicabilidad
5. PyTorch
6. Scikit-learn
7. TabNet
8. Inteligencia Artificial (IA)
9. Intersecciones
10. Ingeniería de características

Abstract

Behaviour Learning in Pac-Man games

This paper explores the use of supervised learning techniques to predict the behaviour of the Pac-Man agent in different scenarios of the game. For this purpose, the game Ms. Pac-Man Vs Ghosts has been used as an environment, on which a dataset structured according to intersections and movements has been generated.

Models such as MLP, TabNet and classic Scikit-learn algorithms have been trained and compared, where improvements have been applied by means of feature engineering and variable importance analysis.

The final objective has been not only to obtain an accurate model, but also explainable, capable of justifying its decisions in each game situation.

The code of this project can be found through this link:

https://github.com/beja28/Ms.Pacman-Machine-Learning

Keywords

1. Neural network
2. Pac-Man
3. Ghost
4. Explainability
5. PyTorch
6. Scikit-learn
7. TabNet
8. Artificial Intelligence (AI)
9. Intersections
10. Feature engineering

Índice

1. Introducción **1**

1.1. Motivación 1

1.2. Objetivos 2

1.3. Plan de trabajo 2

1.3.1. Objetivo 1: Estudiar el juego Ms. Pac-Man Vs Ghosts. 2

1.3.2. Objetivo 2: Obtener y preprocesar los datos. 3

1.3.3. Objetivo 3: Comunicación entre sistemas. 3

1.3.4. Objetivo 4: Diseñar, evaluar y entrenar un primer modelo. . . 3

1.3.5. Objetivo 5: Mejora de modelos mediante un análisis de estadísticas y explicabilidad en distintos experimentos. 4

2. Introduction **5**

2.1. Motivation 5

2.2. Objectives 6

2.3. Work plan 6

2.3.1. Objective 1: Study the game Ms. Pac-Man Vs Ghosts. 6

2.3.2. Objective 2: Obtain and pre-process data. 7

2.3.3. Objective 3: Communication between systems. 7

2.3.4. Objective 4: Design, evaluate and train a first model. 7

2.3.5. Objective 5: Model improvement through statistical and explainability analysis in different experiments. 7

3. Estado del Arte **9**

3.1. Historia y evolución de la Inteligencia Artificial en los videojuegos . . 10

3.1.1. Orígenes del uso de IA en videojuegos clásicos 10

3.1.2. Evolución de las técnicas de IA en juegos a lo largo del tiempo 10

3.1.3. Desafíos en la IA de los videojuegos 13
3.1.4. Conclusión 15
3.2. Explicabilidad en Inteligencia Artificial (IA) 15
3.2.1. Técnicas de Explicabilidad Utilizadas 16
3.2.2. Uso de Técnicas de Explicabilidad en el ámbito de Videojuegos 17
3.2.3. Beneficios y Limitaciones de las Técnicas Analizadas 18
3.2.4. Conclusiones 18
3.3. Redes Neuronales Artificiales: Historia, Fundamentos y Aplicaciones . 18
3.3.1. Historia de las Redes Neuronales Artificiales 18
3.3.2. Fundamentos de las Redes Neuronales Artificiales 19
3.3.3. Arquitecturas de Redes Neuronales 19
3.4. Ingeniería de Comportamientos Inteligentes (ICI) 23

4. Arquitectura y descripción funcional del Trabajo 25
4.1. Entorno Java para la gestión de datos y evaluación 25
4.1.1. Adaptación del workspace 26
4.1.2. Estudio de las características del estado del juego 26
4.1.3. Gestión y validación de estados del juego 28
4.1.4. Socket cliente en Java 29
4.1.5. Distintos modos de ejecución 29
4.2. Redes Neuronales en Python 31
4.2.1. Preprocesamiento de los datos 31
4.2.2. Creación de modelos 31
4.2.3. Entrenamiento 32
4.2.4. Socket Servidor en Python 33
4.3. Explicabilidad 33
4.3.1. Técnica 1: SHAP (SHapley Additive exPlanations) 33
4.3.2. Técnica 2: Feature Importance 34
4.3.3. Técnica 3: LIME (Local Interpretable Model-Independent Explanations) 35
4.3.4. Mapas de calor 35
4.4. Análisis estadístico descriptivo de las puntuaciones obtenidas en las partidas 36

5. Evaluación Experimental 39
5.1. Evaluación nº 1: Una única red neuronal 39

5.1.1. Implementación PyTorch 40
5.1.2. Implementación Scikit-Learn 42
5.1.3. Conclusiones 42
5.2. Evaluación nº 2: Una red neuronal por intersección 43
5.2.1. Rendimiento 44
5.2.2. Explicabilidad 46
5.2.3. Conclusiones 59
5.3. Evaluación nº 3: Evaluación con la red neuronal TabNet 60
5.3.1. Rendimiento 61
5.3.2. Explicabilidad con Feature Importance 63
5.3.3. Conclusiones 66
5.4. Evaluación nº 4: Enriquecimiento del dataset y optimización de variables para TabNet 68
5.4.1. Rendimiento 70
5.4.2. Explicabilidad con Feature Importance 71
5.4.3. Conclusiones 75

6. Conclusiones y Trabajo Futuro 77

Conclusions and Future Work 81

Contribuciones Personales 85
6.1. Alejandro Bejarano del Castillo 85
6.1.1. Antecendentes 85
6.1.2. Aportación 85
6.2. Daniel de la Fuente Díez 86
6.2.1. Antecendentes 86
6.2.2. Aportación 87
6.3. Diego González López 88
6.3.1. Antecendentes 88
6.3.2. Aportación 88
6.4. Daniel Jiménez Rojo 90
6.4.1. Antecendentes 90
6.4.2. Aportación 90

Bibliografía 93

Índice de figuras

3.1. Dijkstra's Shortest Path Algorithim Fuente: Elaboración propia. . . . 11

3.2. Ejemplo de A* en Unity. Fuente: 11

3.3. FSM del ciclo de combate de Halo 2. Fuente: Elaboración propia. . . 12

3.4. Linea del tiempo con los periodos AI Winters. Fuente: Elaboración propia. 14

3.5. Línea del tiempo de la evolución de la IA en Videojuegos. Fuente: Elaboración propia. 15

3.6. Representación de la estructura de una red neuronal. Fuente: 20

3.7. Red Neuronal Artificial de tipo perceptrón simple con n neuronas de entrada, m neuronas en su capa oculta y una neurona de salida. Fuente: 21

3.8. Arquitectura de las CNN aplicada al reconocimiento de dígitos. Fuente: 22

3.9. Subajuste frente a sobreajuste. Fuente: Elaboración propia. 23

3.10. Simulador de Ms. Pac-Man utilizado en la asignatura ICI, adaptado por el Prof. Recio-García a partir del entorno competitivo de CIG 2016. 24

4.1. Diagrama de clases del workspace en el que se muestran los componentes y clases más significativos, facilitando entender sus funcionalidades y relaciones. 26

5.1. Histograma Dataset Exp.1 40

5.2. Boxplot Dataset Exp.1 40

5.3. Histograma de PyTorch (Exp.1) 41

5.4. Histograma de Scikit-Learn (Exp.1) 41

5.5. Boxplot de PyTorch (Exp.1) 41

5.6. Boxplot de Scikit-Learn (Exp.1) 41

5.7. Histograma de Pytorch (Exp.2) 44

5.8. Histograma de Scikit-Learn (Exp.2) 44
5.9. Boxplot de Pytorch (Exp.2) 45
5.10. Boxplot de Scikit-Learn (Exp.2) 45
5.11. Importancia de características con SHAP en un modelo de Pytorch. . 47
5.12. Mapa de calor del modelo Pytorch para la característica totalTime 48
5.13. Mapa de calor del modelo Pytorch para la característica score 48
5.14. Mapa de calor del modelo Pytorch para la característica timeOfLastTimeReversal 48
5.15. Mapa de calor del modelo Pytorch para la característica ghost1EdibleTime 49
5.16. Mapa de calor del modelo Pytorch para la característica ghost3EdibleTime 49
5.17. Mapa de calor del modelo Pytorch para la característica ghost4EdibleTime 49
5.18. Mapa de calor del modelo Pytorch para la característica pathDistanceToPp 49
5.19. Mapa de calor del modelo Pytorch para la característica euclideanDistanceToPp 49
5.20. Mapa de calor del modelo Pytorch para la característica remainingPp . . . 49
5.21. Intersección a estudiar en análisis LIME 50
5.22. Importancia de características con LIME en Pytorch para el modelo en la intersección 165 52
5.23. Importancia de características con SHAP en un modelo de Scikit-Learn. 54
5.24. Mapa de calor del modelo Scikit-Learn para la característica timeOfLastGlobalReversal 55
5.25. Mapa de calor del modelo Scikit-Learn para la característica totalTime . . 55
5.26. Mapa de calor del modelo Scikit-Learn para la característica score 55
5.27. Mapa de calor del modelo Scikit-Learn para la característica remainingPp . 55
5.28. Mapa de calor del modelo Scikit-Learn para la característica pathDistanceToPp 55
5.29. Mapa de calor del modelo Scikit-Learn para la característica euclideanDistanceToPp 55
5.30. Mapa de calor del modelo Scikit-Learn para la característica ghost3NodeIndex 56
5.31. Mapa de calor del modelo Scikit-Learn para la característica ghost4NodeIndex 56
5.32. Mapa de calor del modelo Scikit-Learn para la característica ghost1Distance 56
5.33. Importancia de características con Feature Importance en un modelo Sklearn. 57
5.34. Importancia de características con LIME en Sklearn para el modelo en la intersección 165 59
5.35. Histograma de TabNet (Exp.3) 62
5.36. Boxplot de TabNet (Exp.3) 62

5.37. Importancia de características con Feature Importance en un modelo de TabNet. 63
5.38. Mapa de calor del modelo TabNet para la característica pacmanLastMoveMade_UP 64
5.39. Mapa de calor del modelo TabNet para la característica pacmanLastMoveMade_DOWN 64
5.40. Mapa de calor del modelo TabNet para la característica pacmanLastMoveMade_LEFT 64
5.41. Mapa de calor del modelo TabNet para la característica pacmanLastMoveMade_RIGHT 64
5.42. Mapa de calor del modelo TabNet para la característica powerPill_0 . . . 65
5.43. Mapa de calor del modelo TabNet para la característica powerPill_2 . . . 65
5.44. Mapa de calor del modelo TabNet para la característica pillWasEaten . . . 65
5.45. Mapa de calor del modelo TabNet para la característica ghost2Distance . . 66
5.46. Mapa de calor del modelo TabNet para la característica ghost2LastMove_RIGHT 66
5.47. Mapa de calor del modelo TabNet para la característica ghost3EdibleTime 66
5.48. Histograma de TabNet (Exp.4) 71
5.49. Boxplot de TabNet (Exp.4) 71
5.50. Importancia de características con Feature Importance en un modelo de TabNet. 72
5.51. Mapa de calor del modelo TabNet para la característica pacmanlastMoveMade 73
5.52. Mapa generado tras superar el primer laberinto en el Experimento 4. . . . 73
5.53. Mapa de calor del modelo TabNet para la característica pillsPresence . . . 73
5.54. Mapa de calor del modelo TabNet para la característica numEdibleGhosts 73
5.55. Mapa de calor del modelo TabNet para la característica ghostsNearby . . . 73
5.56. Mapa de calor del modelo TabNet para la característica ghostDangerousNearby 73
5.57. Mapa de calor del modelo TabNet para la característica powerPill_0 . . . 74
5.58. Mapa de calor del modelo TabNet para la característica powerPill_1 . . . 74
5.59. Mapa de calor del modelo TabNet para la característica powerPill_2 . . . 74
5.60. Mapa de calor del modelo TabNet para la característica powerPill_3 . . . 74

6.1. Evolución de la puntuación media diferenciada por arquitectura. . . . 78
6.2. Estabilidad frente a Rendimiento según arquitectura. 79
6.3. Evolution of the average score differentiated by architecture. 82
6.4. Stability vs performance according to architecture. 83

Índice de tablas

4.1. Características del estado de juego relacionadas con Pac-Man 27
4.2. Características del estado de juego relacionadas con los fantasmas . . 27
4.3. Características relacionadas con el estado global del juego 27

5.1. Estadísticas del modelo **PyTorch** (Experimento 1). 40
5.2. Estadísticas del modelo **Scikit-Learn** (Experimento 1). 40
5.3. Estadísticas del modelo **PyTorch** (Experimento 1). 44
5.4. Estadísticas del modelo **Scikit-Learn** (Experimento 1). 44
5.5. Estadísticas descriptivas del modelo de TabNet para el experimento 3. 61
5.6. Descripción de las características añadidas para este experimento . . 69
5.7. Descripción de las recompensas asignadas a cada evento. 69
5.8. Estadísticas descriptivas del modelo de TabNet para el experimento 4. 70

6.1. Comparativa de los resultados de los experimentos. 77
6.2. Comparison of the results of the experiments. 81

Capítulo 1

Introducción

1.1. Motivación

La motivación por parte de todo el equipo para elegir este tema como nuestro Trabajo de Fin de Grado nace del deseo común de continuar formándonos e investigando nuevas formas de aplicar la inteligencia artificial, tras haber estudiado diferentes técnicas en las asignaturas de *Inteligencia Artificial I e Inteligencia Artificial II*. Durante estas asignaturas, no solo adquirimos conocimientos teóricos, sino que también desarrollamos pequeñas prácticas que despertaron nuestro interés por aplicaciones más complejas y realistas.

Además, algunos integrantes del grupo ya tenían experiencia en el manejo de grandes volúmenes de datos gracias a la asignatura de *Análisis de Redes Sociales*, mientras que otros habían trabajado con redes neuronales aplicadas al control de un brazo robot en la asignatura de *Inteligencia Artificial Aplicada al Control*. Esto sentó unas bases sólidas en la comprensión del manejo de datos y la importancia de la calidad de los mismos y cómo esto puede afectar al comportamiento de los modelos entrenados.

Una motivación clave fue también la propuesta de los tutores, quienes nos ofrecieron la posibilidad de aprovechar el material de la asignatura *Ingeniería de Comportamientos Inteligentes*, donde se estudian técnicas como el algoritmo lineal, la máquina de estados, reglas lógicas, lógica borrosa y el razonamiento basado en casos (CBR). Todas estas técnicas se aplican al juego de *Ms. Pac-Man vs Ghosts*, proporcionando un marco de referencia interesante para explorar otros enfoques de inteligencia artificial dentro de ese mismo entorno. Todo ese material nos ayudaría a estudiar otro tipo de estrategias más clásicas y serviría como una buena base para compararlos frente a nuestros modelos.

Sin embargo, hasta el momento no se había profundizado en el uso de redes neuronales dentro del entorno del juego *Ms. Pac-Man vs Ghosts*. Esta falta de exploración nos impulsó a tomar la decisión de enfocar el proyecto en investigar los distintos usos y potencial de las redes neuronales en este contexto, considerando que podría ser una gran oportunidad para aprender, experimentar y aportar algo útil al

trabajo que se había realizado hasta ahora.

1.2. Objetivos

Este trabajo de fin de grado se ha centrado en el desarrollo e implementación de redes neuronales con el fin de crear un comportamiento específico para Ms. Pac-Man en el juego de Ms. Pac-Man VS Ghost. Aplicamos, valoramos y evaluamos técnicas de inteligencia artificial en un entorno dinámico, estudiando cómo procesar los datos del estado del juego, para generar respuestas en tiempo real. Para ello, comparamos distintas herramientas como PyTorch, Scikit-Learn y TabNet.

1. Estudiar el juego de Ms. Pac-Man Vs Ghosts: identificar las variables clave, de las cuales depende el estado del juego.
2. Obtención y Preprocesamiento de Datos: Obtener estados específicos del juego con las variables que hemos analizado previamente.
3. Diseño, evaluación y entrenamiento de un modelo de red neuronal de referencia (baseline), junto con la preparación y tratamiento del conjunto de datos correspondiente.
4. Integración de sistemas, crear un socket para la conexión entre la lógica del juego (Java) y las redes neuronales (Python)
5. Mejora de estos modelos mediante un análisis de estadísticas y explicabilidad en diferentes experimentos.

1.3. Plan de trabajo

En este apartado describimos el plan de trabajo a seguir para alcanzar los objetivos definidos en la sección anterior. Para la coordinación del equipo, empleamos una metodología ágil basada en **Scrum**, lo que nos permitió organizar el trabajo en diferentes hitos consiguiendo así una evolución constante del proyecto. Utilizamos herramientas como **Discord** para la comunicación diaria y **GitHub** para el control de versiones, complementadas con reuniones semanales en las que organizábamos y planificábamos las tareas del proyecto.

A continuación, indicamos las actividades y pasos previstos para cada uno de los objetivos:

1.3.1. Objetivo 1: Estudiar el juego Ms. Pac-Man Vs Ghosts.

1. Adaptar el workspace de la asignatura de ICI para que sirva como base del proyecto.

2. Estudiar las variables que determinan el estado del juego, identificando cuáles son útiles y cuáles no.
3. Detectar qué nuevas variables deberemos calcular al no formar parte del estado primitivo del juego.

1.3.2. Objetivo 2: Obtener y preprocesar los datos.

1. Desarrollar el código necesario para extraer y calcular las variables estudiadas a partir de un estado concreto del juego (por cada tick de ejecución).
2. Implementar las clases necesarias para guardar todos los estados del juego durante una partida, centrándose en aquellos momentos en los que Pac-Man se encuentra en una intersección.
3. El sistema debe permitir almacenar los datos en archivos `.csv`.
4. Editar el motor del juego de forma que pueda ejecutar partidas automáticamente sin interfaz gráfica, determinando parámetros como por ejemplo número de partidas, el fichero de almacenamiento , u otros parámetros adicionales.

1.3.3. Objetivo 3: Comunicación entre sistemas.

1. Desarrollar un socket que conecte el juego en **Java** con las redes neuronales en **Python**, permitiendo el envío y recepción de información en tiempo real.
2. Modificar el socket en Java para enviar el estado del juego en tiempo real.
3. Adaptar el socket en Python de tal forma que reciba la información, la procese en la red neuronal correspondiente y devuelva la acción a realizar en Java.
4. Implementar un sistema que asegura que el estado del juego se envíe siempre a la red neuronal correcta según el contexto.

1.3.4. Objetivo 4: Diseñar, evaluar y entrenar un primer modelo.

1. Preprocesar los datos del estado del juego, como el mapeo de los movimientos del Pac-Man a valores numéricos, la transformación de las variables booleanas a valores numéricos y la codificación de las variables categóricas mediante OneHotEncoding.
2. Investigar las diferentes librerías y frameworks para redes neuronales.
3. Diseñar y entrenar un modelo con **PyTorch**, aplicando una arquitectura sencilla y entrenamiento por *batches*.
4. Implementar y entrenar otro modelo con **Scikit-Learn**, utilizando validación cruzada.

1.3.5. Objetivo 5: Mejora de modelos mediante un análisis de estadísticas y explicabilidad en distintos experimentos.

1. Obtener gráficas de explicabilidad sobre diferentes técnicas como **SHAP**, **Feature Importance** y **LIME**.
2. Generar un conjunto de estadísticas en función de las distintas versiones de fantasmas existentes, recolectando los resultados a modo de estadísticas.
3. Generar gráficas sobre los resultados previos.
4. Tomar conclusiones evaluando tanto la explicabilidad como los resultados obtenidos.
5. Apuntar qué cambios y mejoras se han observado de cara al siguiente experimento.
6. Realizar diferentes experimentos teniendo en cuenta las conclusiones obtenidas en experimentos anteriores. Aplicando tanto los cambios y mejoras apuntadas como la incorporación de nuevas ideas, con el fin de perfeccionar progresivamente el rendimiento de los modelos base.
7. Probar diferentes arquitecturas como **Tabnet** y compararlas con los modelos de Scikit-Learn y Pytorch.

Chapter 2

Introduction

2.1. Motivation

The motivation for the whole team to choose this topic as our Final Degree Project stems from the common desire to continue training and researching new ways of applying artificial intelligence, after having studied different techniques in the subjects of *Artificial Intelligence I and Artificial Intelligence II*. During these subjects, we not only acquired theoretical knowledge, but also developed small practices that awakened our interest in more complex and realistic applications.

In addition, some members of the group already had experience in handling large volumes of data thanks to the *Social Network Analysis* course, while others had worked with neural networks applied to the control of a robot arm in the *Artificial Intelligence Applied to Control* course. This laid a solid foundation in understanding data handling and the importance of data quality and how this can affect the behaviour of trained models.

A key motivation was also the proposal of the tutors, who offered us the possibility to use the material of the subject *Intelligent Behavioural Engineering*, where techniques such as linear algorithms, state machines, logic rules, fuzzy logic and case-based reasoning (CBR) are studied. All these techniques are applied to the game of *Ms. Pac-Man vs Ghosts* game, providing an interesting framework for exploring other artificial intelligence approaches within the same environment. All this material would help us to study other types of more classical strategies and would serve as a good basis for comparing them against our models.

However, the use of neural networks within the game environment of *Ms. Pac-Man vs Ghosts* has so far not been explored in any depth. This lack of exploration prompted us to take the decision to focus the project on investigating the different uses and potential of neural networks in this context, considering that it could be a great opportunity to learn, experiment and contribute something useful to the work that had been done so far.

2.2. Objectives

This thesis has focused on the development and implementation of neural networks in order to create a specific behaviour for Ms. Pac-Man in the Ms, Pac-Man VS Ghost game. We applied, assessed and evaluated artificial intelligence techniques in a dynamic environment, studying how to process game state data to generate real-time responses. To do so, we compared different tools such as PyTorch, Scikit-Learn and TabNet.

1. Study the Ms. Pac-Man Vs Ghosts game: identify the key variables, on which the state of the game depends.
2. Data Collection and Preprocessing: Obtain specific states of the game with the variables we have previously analysed.
3. Design, Evaluation and Training of a first version of a neural network to serve as a baseline model and its corresponding dataset.
4. Systems integration, create a socket for connection between game logic (Java) and neural networks (Python)
5. Improvement of these models through statistical and explainability analysis in different experiments.

2.3. Work plan

In this section we describe the work plan to be followed to achieve the objectives defined in the previous section. For the coordination of the team, we adopted an agile methodology based on **Scrum**, which allowed us to organize the work into different milestones, ensuring a constant progress throughout the proyect. We used tools such as **Discord** for daily communication and **GitHub** for version control, complemented with weekly meetings where we organized and planned the project's tasks.

The activities and steps foreseen for each of the objectives are listed below:

2.3.1. Objective 1: Study the game Ms. Pac-Man Vs Ghosts.

1. Adapt the workspace of the ICI course to serve as a basis for the project.
2. Study the variables that determine the state of the game, identifying which are useful and which are not.
3. Detect which new variables we will have to calculate as they are not part of the primitive state of the game.

2.3.2. Objective 2: Obtain and pre-process data.

1. Develop the necessary code to extract and calculate the variables studied from a specific state of the game (for each tick of execution).

2. Implement the necessary classes to save all game states during a game, focusing on those moments when Pac-Man is at an intersection.

3. The system must allow data to be stored in files `.csv`.

4. Edit the game engine so that it can run games automatically without a graphical interface, by setting parameters such as number of games, the save file, or other additional parameters.

2.3.3. Objective 3: Communication between systems.

1. Develop a socket that connects the game in **Java** with the neural networks in **Python**, allowing the sending and receiving of information in real time.

2. Modify the socket in Java to send the game status in real time.

3. Adapt the Python socket in such a way that it receives the information, processes it in the corresponding neural network and returns the action to be performed in Java.

4. Implement a system that ensures that the game state is always sent to the correct neural network according to the context.

2.3.4. Objective 4: Design, evaluate and train a first model.

1. Preprocessing game state data, such as mapping Pac-Man moves to numeric values, transforming Boolean variables to numeric values, and encoding categorical variables using OneHotEncoding.

2. Investigate the different libraries and frameworks for neural networks.

3. Design and train a model with **PyTorch**, applying a simple architecture and training by *batches*.

4. Implement and train another model with **Scikit-Learn**, using cross-validation.

2.3.5. Objective 5: Model improvement through statistical and explainability analysis in different experiments.

1. Obtain explainability plots on different techniques such as **SHAP**, **Feature Importance** and **LIME**.

2. Generate a set of statistics based on the different versions of existing ghosts, collecting the results as statistics.

3. Generate graphs on previous results.

4. Draw conclusions by assessing both the explainability and the results obtained.

5. Note down what changes and improvements have been observed for the next experiment.

6. Carry out different experiments taking into account the conclusions obtained on the previous ones. Applying both the changes and improvements noted and the incorporation of new ideas, in order to progressively improve the agent's performance.

7. Test different architectures such us **TabNet** and compare them with the models from Scikit-Learn y Pytorch.

Capítulo 3

Estado del Arte

En este capítulo se exponen los fundamentos teóricos y tecnológicos que sustentan el desarrollo de este trabajo, ofreciendo una información detallada sobre los enfoques más importantes que se han utilizado dentro de la Inteligencia Artificial aplicados a los videojuegos, las técnicas de explicabilidad y, por último, el papel que han tenido las redes neuronales en este campo.

Iniciaremos realizando un estudio de la evolución de la **Inteligencia Artificial en los videojuegos**, desde sus aplicaciones iniciales en videojuegos clásicos hasta las aplicaciones más recientes, contando con sistemas avanzados de toma de decisiones. Se realizará la exploración de diferentes enfoques, algunos tradicionales, tales como máquinas de estados finitos, búsqueda de caminos y algoritmos de planificación, para posteriormente continuar con su evolución hacia técnicas que emplean métodos de aprendizaje automático y redes neuronales.

Después, estudiaremos la relevancia de la **explicabilidad** en los modelos de IA, un aspecto esencial para el estudio y la mejora de los algoritmos que se utilizan. En este apartado se presentará información de técnicas como SHAP (*SHapley Additive Explanations*), Feature Importance y LIME (*Local Interpretable Model-agnostic Explanations*), explicando su funcionamiento y su importancia en el estudio del comportamiento de los modelos.

Finalmente, se examinará el papel de las **redes neuronales** dentro de la toma de decisiones en contextos de videojuegos. Para ello, se expondrán sus fundamentos, las arquitecturas más comunes, así como su efecto en el ámbito de la mejora del comportamiento de los agentes virtuales. Se presentarán también aplicaciones recientes en la industria del videojuego, donde la utilización de redes neuronales ha permitido desarrollar sistemas de juego más realistas y adaptativos.

En el transcurso de este capítulo revisaremos los avances y los retos asociados a cada uno de estos aspectos, situando este estudio dentro del actual contexto de la Inteligencia Artificial aplicada a videojuegos y dándole sentido a la justificación del desarrollo del mismo.

3.1. Historia y evolución de la Inteligencia Artificial en los videojuegos

La inteligencia artificial (IA) ha representado un aspecto fundamental en el contexto del diseño de videojuegos. En efecto, por medio de la IA se han podido obtener videojuegos con experiencias cada vez más dinámicas, complejas e interactivas para el usuario. Desde sus orígenes, la IA ha ido evolucionando para adaptarse a las necesidades del diseño de videojuegos, muy en particular para poder incluir las exigencias de realismo y complejidad en los videojuegos que han influido en la jugabilidad, la narración y la inmersión del jugador en los videojuegos.

3.1.1. Orígenes del uso de IA en videojuegos clásicos

Los primeros indicios de uso de IA en videojuegos se remontan a los años 50 y juegos primitivos como *Nim*, que usaban principios básicos de IA, utilizando lógica combinatoria para calcular los movimientos óptimos. En los años 70, *Pong* (1972) usó algoritmos sencillos para poder seguir el trayecto de la pelota, potenciando mecánicas básicas de IA en los videojuegos.

Uno de los hitos más importantes fue *Pac-Man* (1980) que asignó patrones de comportamiento a cada uno de sus fantasmas, generando estrategias diferenciadas que aumentaban la dificultad para el jugador. Cada uno de los fantasmas seguía reglas de conducta que hacían variar su comportamiento, consolidando Pac-Man como uno de los videojuegos que más utiliza la IA con patrones.

El primero de ellos, **Blinky (rojo)**, perseguía constantemente a Pac-Man, acortando la distancia con el jugador. **Pinky (rosa)**, intentaba adelantarse al movimiento de Pac-Man, anticipando su posición. **Inky (azul)**, combinaba la posición de Blinky y Pac-Man para generar trayectorias impredecibles. Y por último, **Clyde (naranja)**, alternaba entre moverse hacia Pac-Man o vagar aleatoriamente.

Estos patrones predefinidos contribuyeron a que *Pac-Man* se convirtiera en un referente en el diseño de IA para videojuegos, estableciendo un modelo que influyó en juegos posteriores [Gildardo Sanchez-Ante (2014)].

3.1.2. Evolución de las técnicas de IA en juegos a lo largo del tiempo

Algoritmos de Búsqueda de Caminos (Pathfinding)

La búsqueda de caminos es uno de los procesos básicos en la búsqueda de personajes en entornos complejos. *Tanktics* (1981) fue uno de los primeros en utilizar algoritmos de búsqueda como el de Dijkstra (1956), permitiendo así a los tanques encontrar rutas óptimas para los mapas en hexágonos. Su autor destacó por ser pionero en el uso de técnicas avanzadas en una época en la que este tipo de enfoques apenas se aplicaban en los videojuegos, marcando así un punto de inflexión en el

desarrollo de comportamientos más complejos para personajes en entornos virtuales [Yusuf (2022)].

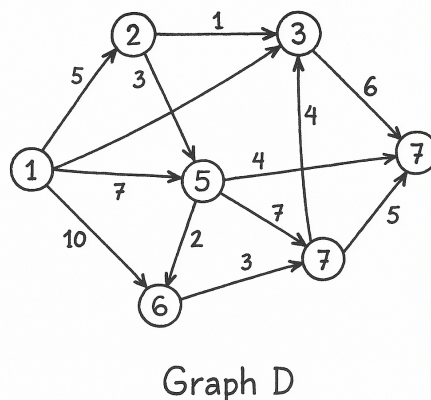

Figura 3.1: Dijkstra's Shortest Path Algorithim Fuente: Elaboración propia.

Posteriormente, el algoritmo A* se volvió el estándar de la industria debido a su habilidad de utilizar heurísticas para poder encontrar rutas y economizar en costes computacionales; juegos como Diablo y The Elder Scrolls II: Daggerfall (1996) usaron A* para guiar a los NPCs a través de entornos complejos, aumentando de esta manera la fluidez y el realismo en los desplazamientos, mejorando así la interacción del usuario con la CPU.

Hoy en día, motores de videojuegos como Unity y Unreal Engine disponen de herramientas avanzadas que permiten integrar A* para aumentar la navegación en diversas dimensiones, y permiten a los desarrolladores crear sistemas de navegación complejos que hacen que la experiencia del jugador sea muy agradable [More (2024)].

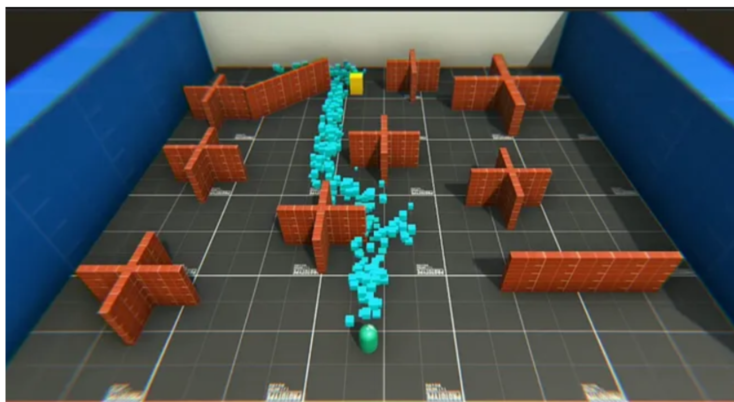

Figura 3.2: Ejemplo de A* en Unity. Fuente: [Juan Martinez (2023)]

Máquinas de Estados Finitos (FSM)

Las máquinas de estados finitos (FSM) marcaron un cambio radical en el comportamiento de los NPCs en los años 90. Estas máquinas permiten modelar com-

portamientos complejos mediante estados y transiciones, lo cual simplifica la lógica de programación de enemigos y aliados. *Herzog Zwei* (1989), considerado uno de los primeros RTS (juegos de estrategia en tiempo real), también utilizó las FSM para controlar las unidades [Jagdale (2021)].

Existen muchas formas de representar las máquinas de estados finitos (FSM). Una de las técnicas más extendidas consiste en representar la máquina de estados mediante un grafo dirigido, generalmente denominado también 'diagrama de estados'[Jagdale (2021)].

Así, en el grafo dirigido, cada uno de los nodos del grafo representa un estado concreto que corresponde a una acción o comportamiento del personaje, por ejemplo, estar parado, saltar, atacar o correr. El paso de estados se halla condicionado por una serie de entradas como la pulsación de teclas, el análisis del entorno o la distancia desde el jugador, entre otros. De esta manera, podemos hacer que el personaje pase de un nodo a otro. Al cambiar el estado se lleva a cabo el comportamiento representado por el estado al que se ha llegado [Jagdale (2021)].

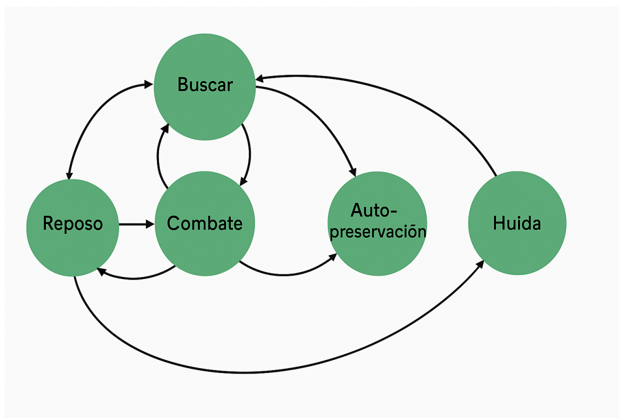

Figura 3.3: FSM del ciclo de combate de Halo 2. Fuente: Elaboración propia.

Dune II (1992) amplió el potencial de las FSMs aplicándolas al control de diferentes tipos de unidades y de edificios, colocándolas en los cimientos de los modernos juegos RTS como *Warcraft, StarCraft o Age of Empires*. Las FSMs facilitaron la modelización de comportamientos complejos como la cosecha de recursos, la construcción de bases y las tácticas de combate.

En los videojuegos, las máquinas de estados finitos (FSM) eran almacenadas internamente en forma de tablas de transición de estados o matrices de adyacencia. Dichas tablas de transición de estados o matrices de adyacencia facilitaban lo que son los estados, junto con la representación de las transiciones y relaciones entre los distintos estados incluidos en el sistema, que permitían simplificar la gestión de las transiciones y cambios de estado y, por tanto, facilitar el correcto funcionamiento de las máquinas de estados finitos [Toptal (2021)].

Las FSM sí poseen mucha importancia hoy en día, aplicándose no sólo al comportamiento de los personajes, sino también a las animaciones, efectos visuales y sistemas de IA de los videojuegos más complejos. Herramientas como Unreal Engine o Unity permiten crear FSMs para adaptarlas al funcionamiento esperado del

videojuego [Toptal (2021)].

Redes Neuronales Artificiales (ANN)

El uso de redes neuronales en videojuegos se remonta a los años noventa, aunque, debido a las limitaciones de hardware y a la enorme complejidad de los modelos en aquellos años su éxito fue escaso. No obstante, el concepto de utilizar redes neuronales para dotar a las criaturas de comportamientos más naturales y adaptativos se fue poco a poco implementando en distintos títulos hasta que se produjo un cambio en la propia evolución de la IA en videojuegos.

Uno de los primeros ejemplos destacados fue **Creatures (1996)**. Este juego fue pionero en la implementación eficaz de redes neuronales para modelar comportamientos complejos. Los jugadores criaban criaturas llamadas *Norns* que poseían cerebros virtuales estructurados por redes neuronales artificiales. Estas criaturas eran capaces de aprender de su entorno y de la interacción con el jugador utilizando mecanismos de aprendizaje por refuerzo. Cada Norn era capaz de aprender a desarrollar comportamientos únicos dependiendo del entorno donde se encontrara y de sus experiencias pasadas [Grand et al. (1996)].

Con la evolución del hardware y la optimización de los algoritmos se implementaron redes neuronales más avanzadas que fueron capaces de llevar a cabo tareas complejas en videojuegos. Técnicas como las CNN y las RNN ampliaban las posibilidades de la IA en los videojuegos. Gracias a la adopción del aprendizaje profundo (*Deep Learning*), las redes neuronales se han establecido como una de las técnicas más relevantes del desarrollo de IA en videojuegos, habilitando a los agentes para tener comportamientos más inteligentes y adaptativos. Surgen proyectos como *AlphaStar* (DeepMind, 2019), donde aplicaron técnicas de **aprendizaje por refuerzo profundo (Deep Reinforcement Learning)** consiguiendo jugar en *StarCraft II* a niveles sobrehumanos, superando a jugadores profesionales [DeepMind (2019)].

Las redes neuronales seguirán siendo una de las claves para el desarrollo de IA en videojuegos; en este sentido, se espera que gracias a futuros avances se puedan crear NPCs que sean capaces de aprender en tiempo real durante la partida. Además de una IA capaz de generar contenido automáticamente (procedural content generation) e interactuar con narrativas capaces de permitir que los personajes adapten sus diálogos y acciones en función de las decisiones del jugador [Games (2025)].

Un caso clásico de redes neuronales con aprendizaje por refuerzo es *Deep Q-Networks* (DQN) aplicado a juegos clásicos de *Atari*. Agentes entrenados en DQN incluso han superado el rendimiento humano en juegos como *Breakout* y *Space Invaders* al aprender sólo de los píxeles de la pantalla y de la puntuación.

3.1.3. Desafíos en la IA de los videojuegos

- **Artificial Stupidity:** Donde el comportamiento complejo puede llevar a fallos no esperados. En algunos casos, esto es intencionado para controlar la dificultad (por ej. hacer que la IA cometa errores intencionales con niveles fá-

ciles), pero también se puede expresar como fallos no intencionados que pueden afectar negativamente la experiencia del jugador.

Ejemplos clásicos de Artificial Stupidity son NPCs que causan comportamientos repetidos por los patrones esperados o que no cumplen con los comportamientos esperados en situaciones adversas debido a errores en los algoritmos de los decisores. Estos problemas pueden romper rápidamente la experiencia de inmersión del jugador y la jugabilidad [Juan Martinez (2023)].

- **Cheating AI:** En otro orden de cosas, existen también ciertos videojuegos que, con el propósito de mitigar las limitaciones técnicas en sus programaciones, usan "Cheating AI". Por ejemplo, *Civilization* (1991) y *Civilization II* (1996) incorporaban la idea de que la IA obtuviera una serie de ventajas ocultas (como generar unidades en lugares intempestivos) con el objetivo de mantener la dificultad de sus respectivos videojuegos, habida cuenta de las limitaciones asociadas a planificar estrategias complejas en un sistema de tiempo real.

  Esta idea, aunque criticada por algunos usuarios de estos videojuegos, permitía mantener la competitividad de la IA en juegos con limitaciones de hardware. En contraposición, los juegos modernos suelen evitar este tipo de prácticas en favor de otras más limpias y equilibradas [Juan Martinez (2023)].

- **AI Winters:** Por último, cabe señalar que el desarrollo de la IA ha tenido sus propios espacios de estancamiento conocidos como "AI Winters", marcados por la reducción de fondos e interés, así como por los resultados por debajo de las expectativas de lo que se puede conseguir con la IA. En la industria del videojuego, estos AI Winters limitaron la innovación relacionada con la IA, de manera que los videojuegos dieron más importancia a los gráficos o a la narrativa. Así, entre 1987 y 2000, la IA en videojuegos progresaba despacio aunque, sí hay que reconocer, salieron a la luz videojuegos notables que incorporaban IA como son *Creatures*. Las producciones de videojuegos sufrieron también del problema de las limitaciones en el hardware, del alto coste del desarrollo de los videojuegos y de las expectativas no cumplidas [Juan Martinez (2023)].

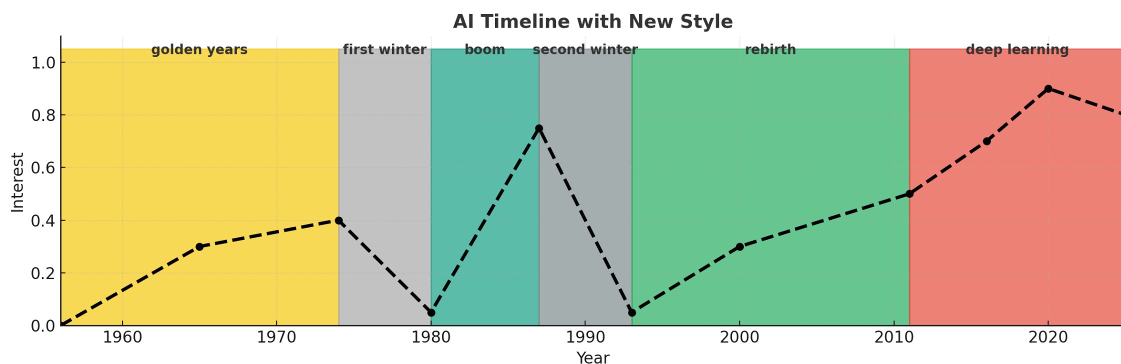

Figura 3.4: Linea del tiempo con los periodos AI Winters. Fuente: Elaboración propia.

3.1.4. Conclusión

La evolución de la IA en los videojuegos ha ido del uso de patrones simples y predecibles a la creación de sistemas complejos que aprenden y se reinventan de forma autónoma en tiempo real. Una IA que ha pasado de algoritmos simples y básicos para la búsqueda de soluciones de juegos hasta el uso de redes neuronales y el Machine Learning: la IA ha enriquecido las experiencias de juego mediante la obtención de mundos más dinámicos y más inmersivos que los tradicionales.

Con el avance de la tecnología, el papel de la IA en los videojuegos también continuará ampliándose, explorando los límites de la creación de experiencias totalmente personalizadas y con una gran carga de realismo; en este sentido, la inclusión de técnicas modernas como el uso de modelos de lenguaje (LLMs) y la inteligencia artificial generativa, preludian un cambio más radical para la industria.

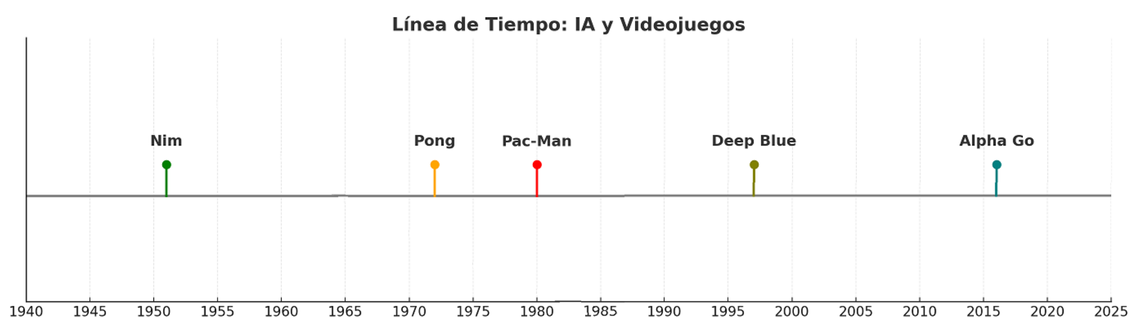

Figura 3.5: Línea del tiempo de la evolución de la IA en Videojuegos. Fuente: Elaboración propia.

3.2. Explicabilidad en Inteligencia Artificial (IA)

Por explicabilidad en IA se entiende la capacidad de entender, interpretar y justificar las decisiones que toman los modelos de Machine Learning. Con el aumento de la complejidad de los modelos de IA también aumenta la dificultad de entender cómo estos llegan a una serie de conclusiones. Esto plantea grandes retos relacionados tanto con la transparencia como con la confianza sobre los sistemas automáticos [IBM (2024)].

La explicabilidad es crucial por diversas razones. En primer lugar, aporta **transparencia**, ya que ofrece una determinada respuesta a las preguntas (¿cómo y por qué lo hace?); en consecuencia, permite la detección de errores o sesgos (si los hay). Además, ofrece una mayor **confianza** de los usuarios y los desarrolladores (valida que las decisiones que toma la IA son coherentes y están justificadas). También permite **detectar sesgos** o errores en el modelo (especialmente en aplicaciones que tratan de manera crítica (ej. temas de salud, temas de justicia, etc.)). Al comprender las decisiones del modelo, se pueden realizar ajustes que optimicen y mejoren su rendimiento. Por último, en sectores regulados, como las finanzas o la medicina, la explicabilidad es necesaria para el **cumplimiento de la normativa**.

Al mismo tiempo, la explicabilidad se ha convertido en uno de los factores más relevantes en el correcto desarrollo ético de la IA, como forma de evitar y de mitigar riesgos asociados a decisiones automatizadas y a impactos en la sociedad.

- **Tipos de explicabilidad:**

  La explicabilidad podemos dividirla en dos grupos muy amplios: la **explicabilidad global** y la **explicabilidad local** [GoogleCloud (2025)].

  La primera de ellas suministra una descripción general de cómo se comporta el modelo en su conjunto. Permite tener una idea de qué características tienen un mayor peso en las predicciones y cómo interactúan entre ellas. Para un recomendador de películas, por ejemplo, la explicabilidad global nos podría ayudar a comprender qué características de las películas hay que tener en cuenta para realizar las recomendaciones. Mientras que la explicabilidad local suministra la capacidad de explicar predicciones particulares realizadas por el modelo (en este caso, los usuarios) analizando qué características han influido en la predicción particular. Por ejemplo, a partir de la explicabilidad local podría averiguarse por qué un sistema de crédito ha rechazado la solicitud de un determinado usuario.

- **Desafíos de la Explicabilidad:**

  La implementación de técnicas de explicabilidad en IA enfrenta varios desafíos importantes. Uno de los principales es la **complejidad de los modelos**, ya que las redes neuronales profundas y otros modelos complejos poseen estructuras difíciles de interpretar debido a la gran cantidad de parámetros involucrados. A menudo, los modelos más precisos son también los más difíciles de explicar. Los modelos simples son interpretables pero menos precisos [ManagementSolutions (2023)]. Otro reto importante son los **sesgos en las explicaciones**, ya que las técnicas de explicabilidad, al simplificar modelos complejos, pueden introducir distorsiones que llevan a interpretaciones erróneas del funcionamiento real del modelo [ReasonWhy (2024)]. Por último, algunas técnicas de explicabilidad, como SHAP —mencionadas en la sección 3.2.1- requieren altos recursos computacionales para generar explicaciones detalladas [ManagementSolutions (2023)].

3.2.1. Técnicas de Explicabilidad Utilizadas

En este trabajo se utilizan tres técnicas ampliamente reconocidas en la explicabilidad de modelos de IA: SHAP, Feature Importance y LIME.

- **SHAP SHapley Additive Explanations**

  SHAP es una técnica basada en la teoría de juegos de Shapley que asigna un valor a cada característica según su contribución a la predicción del modelo. Este enfoque proporciona explicaciones consistentes y equitativas sobre la importancia de las variables. Su fundamento se basa en la teoría de juegos

cooperativos, distribuye la *recompensa* de una predicción entre las características utilizadas. Una de sus principales ventajas es que proporciona explicaciones consistentes y es aplicable a una amplia variedad de modelos. Sin embargo, resulta computacionalmente costoso, especialmente en modelos con muchas variables. En el contexto de los videojuegos permite determinar qué factores (posición de los enemigos, puntuación actual, etc.) influyeron en la decisión de un agente de moverse en cierta dirección [de Delatorre AI (2024)].

- **Feature Importance**

  La técnica de importancia de características evalúa el peso relativo de cada variable en la predicción final. Es comúnmente utilizada en modelos basados en árboles de decisión como Random Forest y XGBoost. Su fundamento se basa en medir cómo la inclusión o exclusión de una característica afecta la precisión del modelo. Es sencilla de interpretar y computacionalmente eficiente a pesar de no capturar interacciones complejas entre variables. En el ámbito de los videojuegos es utilizado para identificar qué variables determinan el comportamiento de personajes no jugadores (NPCs) [APORIA (2022)].

- **LIME (Local Interpretable Model-agnostic Explanations)**

  LIME genera modelos locales simplificados que aproximan el comportamiento del modelo complejo alrededor de una predicción específica. Esto permite obtener explicaciones comprensibles para predicciones individuales. Su lógica se basa en crear un modelo interpretable que aproxima el comportamiento del modelo original. Entre sus puntos fuertes destaca la capacidad de ofrecer explicaciones detalladas y su compatibilidad con cualquier tipo de modelo. Sin embargo, puede generar explicaciones inconsistentes si los datos perturbados no representan adecuadamente la distribución original. En videojuegos resulta útil para analizar por qué un agente decidió atacar o huir en una situación específica [C3.ai (2024)].

3.2.2. Uso de Técnicas de Explicabilidad en el ámbito de Videojuegos

La explicabilidad puede tener aplicaciones muy relevantes para el desarrollo de los videojuegos, dotando de mayor entendimiento al comportamiento de los agentes controlados por IA y mejorando la experiencia de usuario. Entre ellas permite **analizar el comportamiento de los NPCs**, evaluando las decisiones tomadas por estos y corregir su comportamiento para dotarlos de realismo. También facilita el **ajuste de dificultad adaptativa**, permitiendo realizar un ajuste dinámico de la dificultad a partir del rendimiento del jugador. Además, ayuda a **detectar errores** o comportamientos no deseados en la lógica de los agentes y permite **crear mecánicas** más interesantes y balanceadas.

3.2.3. Beneficios y Limitaciones de las Técnicas Analizadas

Entre los principales beneficios de estas técnicas se destacan la mejora en la transparencia y comprensión de los modelos de IA, el aumento de la confianza del usuario al interpretar las predicciones, la facilidad para depurar y optimizar modelos, y la promoción de un desarrollo ético y responsable [ESAN (2024); ManagementSolutions (2023)].

No obstante, también presentan limitaciones importantes: algunas técnicas requieren un alto coste computacional [ManagementSolutions (2023)], no todas son compatibles con cualquier tipo de modelo, y en ciertos casos, las explicaciones pueden simplificar en exceso el comportamiento del sistema, generando interpretaciones erróneas.

3.2.4. Conclusiones

La necesidad de la explicabilidad en IA se halla en la necesidad de mantener la transparencia, la confianza y la ética en la utilización de modelos complejos para la resolución de problemas. Las técnicas de explicabilidad descritas en este trabajo, como SHAP, Feature Importance, LIME, etc., son mecanismos útiles para comprender y optimizar sistemas de IA, especialmente para aplicaciones dinámicas como los videojuegos, pero hay que tener en cuenta que, aunque son útiles, no son la única solución a los problemas que se puedan definir, por lo que hay que tener en cuenta sus limitaciones y elegir la técnica más adecuada según el contexto y los objetivos que se persigan en el análisis.

3.3. Redes Neuronales Artificiales: Historia, Fundamentos y Aplicaciones

Las redes neuronales artificiales (ANN) son modelos computacionales inspirados en la estructura y funcionamiento del cerebro humano. Estas redes están compuestas por unidades interconectadas, denominadas neuronas artificiales, que procesan información en paralelo. A lo largo de las últimas décadas, las ANN han evolucionado significativamente, encontrando aplicaciones en diversos campos como la medicina, la economía, la robótica y, de manera destacada, en la industria de los videojuegos [Schmidhuber (2015)].

3.3.1. Historia de las Redes Neuronales Artificiales

La idea de neuronas artificiales se remonta a 1943 con el trabajo de Warren McCulloch y Walter Pitts, quienes formularon un modelo matemático que describía de forma simplificada el funcionamiento lógico de las neuronas. Este trabajo fue importante porque demostró que, en teoría, las redes neuronales podían computar cualquier tipo de función lógica, lo que representó un paso fundamental para el

desarrollo de las redes neuronales artificiales (ANN) [McCulloch y Pitts (1943)]. El perceptrón fue presentado en 1958 por Frank Rosenblatt como la primera red neuronal de una sola capa capaz de aprender a resolver problemas de clasificación lineales. A pesar de sus limitaciones, como la incapacidad para resolver problemas no lineales, el perceptrón constituyó un avance significativo hacia la investigación en ANN [Rosenblatt (1958)].

Durante las décadas de 1960 y 1970, la investigación en redes neuronales enfrentó desafíos considerables. La publicación del libro *Perceptrons* de Marvin Minsky y Seymour Papert en 1969 destacó las limitaciones del perceptrón, lo que llevó a una disminución del interés y la financiación en este campo, periodo conocido como el "invierno de la IA" [Minsky y Papert (1969)].

No obstante, en 1982, John Hopfield revitalizó el interés en las ANN al introducir las redes de Hopfield, que podían servir como sistemas de memoria asociativa [Hopfield (1982)]. Posteriormente, en 1986, David Rumelhart, Geoffrey Hinton y Ronald Williams popularizaron el algoritmo de retropropagación, permitiendo el entrenamiento efectivo de redes neuronales multicapa y superando las limitaciones anteriores [Rumelhart et al. (1986)].

Con el aumento de la capacidad computacional y la disponibilidad de grandes conjuntos de datos, las ANN experimentaron un resurgimiento en la década de 2000. En 2024, John Hopfield y Geoffrey Hinton fueron galardonados con el Premio Nobel de Física por sus contribuciones fundamentales al aprendizaje automático mediante redes neuronales artificiales [The Royal Swedish Academy of Sciences (2024)].

3.3.2. Fundamentos de las Redes Neuronales Artificiales

Una **neurona artificial** es una unidad básica que recibe una o más entradas, las procesa y genera una salida. Cada entrada tiene un peso asociado que amplifica o atenúa su influencia. La neurona suma las entradas ponderadas y aplica una función de activación al resultado para producir la salida.

La **arquitectura de las Redes Neuronales** se organiza en capas. Una primera *capa de entrada*, donde se reciben los datos iniciales. Pasando por una serie de *capas ocultas* donde se procesan las entradas mediante neuronas intermedias y, por último, una *capa de salida* que genera el resultado final de la red.

La interconexión de estas capas permite a la red aprender representaciones complejas de los datos IBM (2021).

3.3.3. Arquitecturas de Redes Neuronales

Perceptrón Multicapa (MLP)

El Perceptrón Multicapa (MLP) es una de las arquitecturas más básicas y ampliamente utilizadas en redes neuronales artificiales. Es una red de tipo *feedforward*, donde la información fluye en una sola dirección desde la capa de entrada hasta la

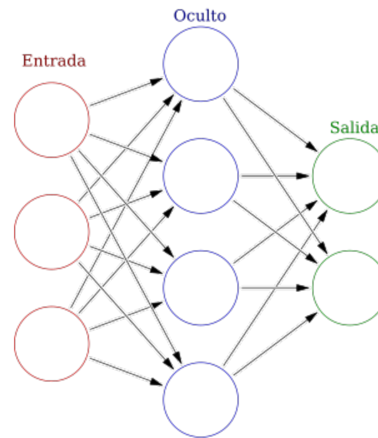

Figura 3.6: Representación de la estructura de una red neuronal. Fuente: [Wikipedia (2025)]

capa de salida, sin ciclos ni bucles.

El MLP está compuesto por una *capa de entrada* que recibe los datos iniciales y los transmite a la primera capa oculta. Una serie de *capas ocultas* donde se lleva a cabo la mayor parte del procesamiento. Cada neurona aplica una función de activación no lineal para permitir a la red aprender patrones complejos. Y por último, una *capa de salida* que genera la predicción o clasificación final basada en la información procesada por las capas ocultas.

El algoritmo de entrenamiento más común en los MLP es la **retropropagación del error** (*backpropagation*), combinado con técnicas de optimización como el *descenso de gradiente*. Este proceso ajusta los pesos sinápticos minimizando la función de pérdida mediante el cálculo del gradiente [Rumelhart et al. (1986)].

Los MLP son especialmente efectivos en tareas como la clasificación supervisada, la regresión y el reconocimiento de patrones, aunque su desempeño disminuye con datos de alta dimensionalidad o secuenciales [Schmidhuber (2015)].

Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales (CNN) son algoritmos especializados en aprendizaje profundo diseñados para análisis de datos con estructura de cuadrícula (e.g., imágenes). Introducidas en la década de 1980, su avance definitivo llegó con LeNet-5 de Yann LeCun et al. (1998), diseñado para reconocer dígitos manuscritos [LeCun et al. (1998)].

Las CNN utilizan capas convolucionales que aplican filtros para detectar características locales (como bordes o texturas) y capas de pooling que reducen la dimensionalidad. Esta arquitectura aprovecha la invariancia espacial, permitiendo reconocer patrones independientemente de su posición [Krizhevsky et al. (2012)].

En 2012, AlexNet demostró su potencial al reducir el error en ImageNet en un 41 % respecto a métodos clásicos, gracias a técnicas como ReLU y dropout

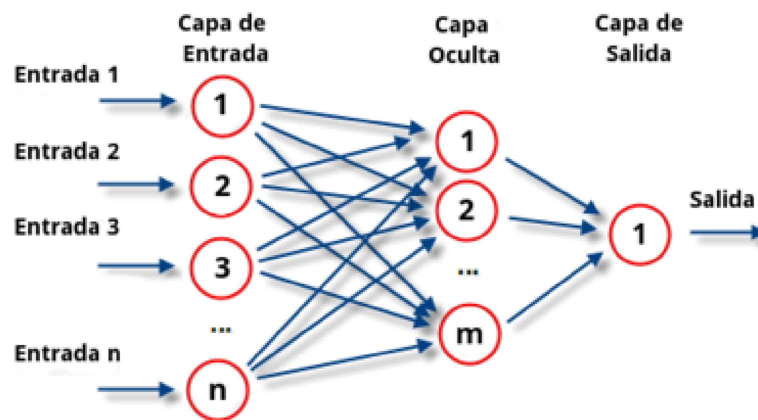

Figura 3.7: Red Neuronal Artificial de tipo perceptrón simple con n neuronas de entrada, m neuronas en su capa oculta y una neurona de salida. Fuente: [ResearchGate (2023)]

[Krizhevsky et al. (2012)]. Las CNN han revolucionado aplicaciones como diagnóstico médico, vehículos autónomos y procesamiento de lenguaje natural [O'Shea y Nash (2015)].

Inspiración biológica y paralelismos con el sistema visual humano

Las CNN están inspiradas en la estructura jerárquica de la corteza visual humana. En el cerebro, las neuronas responden a estímulos específicos dentro de regiones limitadas del campo visual. Del mismo modo, en las CNN, los filtros (o núcleos) extraen características locales, como bordes o texturas, y las capas más profundas combinan estas características para reconocer patrones más complejos.

Los paralelismos clave incluyen una **arquitectura jerárquica**, ya que las características sencillas se extraen en las primeras capas y las complejas en las capas posteriores. La **conectividad local** se refleja en que cada neurona está conectada a una región local de la entrada, similar a cómo funciona la corteza visual. Además, se logra una **invariabilidad traslacional** gracias al uso de capas de agrupamiento (pooling), lo que permite que las CNN sean capaces de reconocer patrones sin importar su posición en la imagen. Por último, se introduce la **no linealidad** a través de funciones de activación como ReLU, permitiendo la modelación de relaciones complejas.

Una red neuronal convolucional típica está compuesta por varias capas especializadas que transforman la entrada inicial en una predicción final:

1. **Capas Convolucionales:** Son el corazón de las CNN. Aplican filtros (o núcleos) sobre la imagen de entrada para extraer características locales. Cada filtro se desliza sobre la imagen realizando una operación de convolución, detectando patrones específicos como bordes o texturas. Los pesos del filtro se ajustan durante el proceso de entrenamiento.

2. **Función de Activación (ReLU):** Tras la convolución, se aplica una función de activación como ReLU (Rectified Linear Unit), que introduce no linealidad

al modelo y mitiga el problema del desvanecimiento del gradiente.

3. **Capas de Agrupamiento (Pooling):** Estas capas reducen la dimensionalidad del mapa de características, disminuyendo la complejidad computacional y previniendo el sobreajuste. Las técnicas más comunes son **Max-pooling**, la cual selecciona el valor máximo en una región específica y **Average-pooling** que calcula el valor promedio en la región.

4. **Capas Completamente Conectadas (Fully Connected):** Después de varias capas convolucionales y de agrupamiento, los mapas de características se aplanan y se introducen en una o más capas densas. Estas capas actúan como un clasificador que utiliza las características extraídas para realizar la predicción final.

5. **Capa de Salida:** En tareas de clasificación, se utiliza una función *softmax* para generar probabilidades asociadas a cada clase.

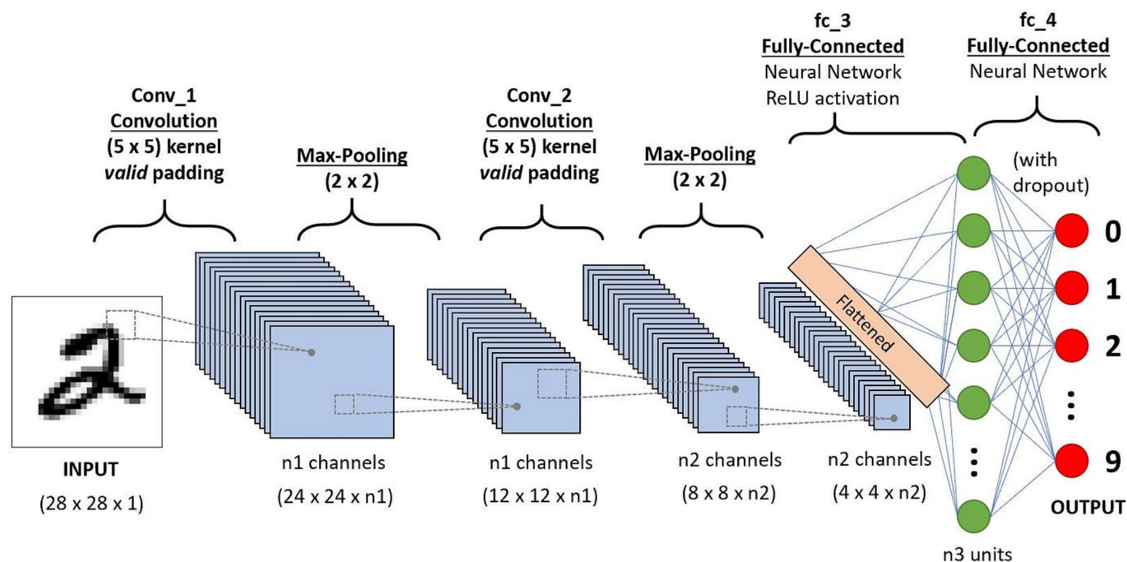

Figura 3.8: Arquitectura de las CNN aplicada al reconocimiento de dígitos. Fuente: [Keita (2025)]

El **sobreajuste** es un problema común en redes neuronales profundas debido a su alta capacidad de modelado. Las CNN pueden memorizar patrones específicos del conjunto de entrenamiento y fallar al generalizar en nuevos datos. Sin embargo, existen varias técnicas para mitigar el sobreajuste. El **abandono (Dropout)** apaga aleatoriamente neuronas durante el entrenamiento para prevenir la dependencia excesiva en ciertas conexiones. La **normalización por lotes (Batch Normalization)** ajusta las activaciones de las capas para acelerar y estabilizar el entrenamiento. La **parada anticipada (Early Stopping)** detiene el entrenamiento cuando la precisión en el conjunto de validación deja de mejorar. El **aumento de datos (Data Augmentation)** genera variaciones artificiales de los datos de entrada mediante rotaciones, escalados o recortes. Finalmente, las técnicas de **regularización L1 y L2**

añaden penalizaciones a los pesos para evitar valores extremos que podrían causar sobreajuste.

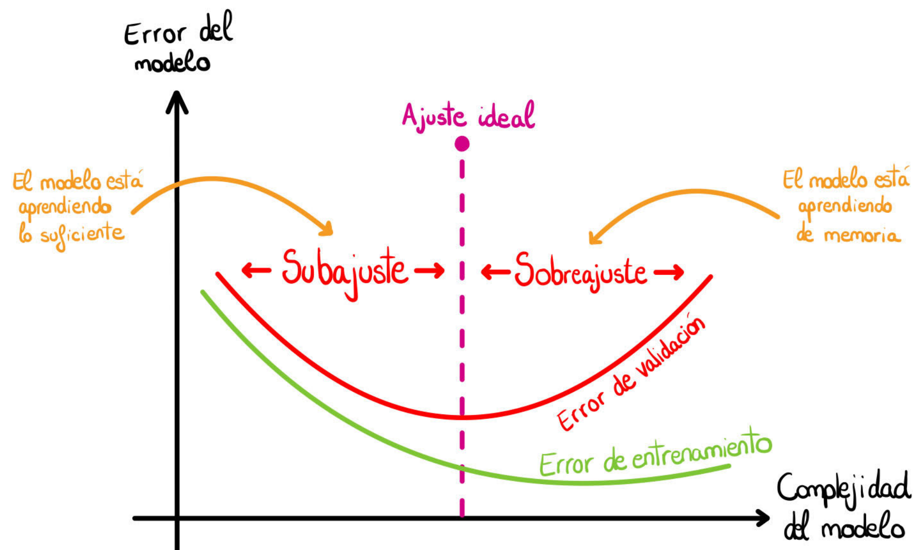

Figura 3.9: Subajuste frente a sobreajuste. Fuente: Elaboración propia.

Las CNN se utilizan ampliamente en múltiples campos debido a su capacidad para procesar datos visuales y secuenciales. En la **clasificación de imágenes** permiten reconocer objetos en diversas aplicaciones, desde aplicaciones médicas hasta redes sociales. Son utilizadas también en **sistemas** que **localizan múltiples objetos** dentro de una imagen, como vehículos en imágenes satelitales. Estas redes son clave en aplicaciones de seguridad y redes sociales gracias al **reconocimiento facial**. Aunque su uso es menos frecuente en **Procesamiento de lenguaje natural (NLP)**, las CNN se han aplicado con éxito en tareas como la clasificación de textos. Además, las CNN también se emplean para extraer características en datos secuenciales gracias al **reconocimiento de voz y análisis de series temporales**

Existen varios marcos populares que facilitan la implementación de CNN. El primero de ellos, **TensorFlow**, desarrollado por Google, es ampliamente utilizado por su escalabilidad y soporte para redes complejas. **Keras**, el cual es construido sobre TensorFlow, ofrece una API de alto nivel fácil de usar para prototipos rápidos. Y por último, **PyTorch**, desarrollado por Facebook, es popular en la investigación gracias a su flexibilidad y gráficos dinámicos [DataCamp (2024)].

3.4. Ingeniería de Comportamientos Inteligentes (ICI)

La asignatura optativa **Ingeniería de Comportamientos Inteligentes (ICI)**, impartida en el Grado de Ingeniería Informática de la UCM, se centra en el estudio y desarrollo de técnicas de Inteligencia Artificial aplicadas al diseño de agentes autónomos. Para facilitar la experimentación, el equipo docente mantiene el *workspace*

Ms. Pac-Man Vs Ghosts, disponible públicamente con fines docentes e investigativos [GAIA Research Group (2021)].

Este entorno parte de la versión competitiva presentada en la *IEEE Conference on Computational Intelligence and Games 2016* (CIG 2016) [Rohlfshagen et al. (2016)]. Dicha versión fue adaptada y ampliada en la UCM por el Prof. Juan A. Recio-García, incorporando utilidades específicas para la asignatura, tales como plantillas de Máquinas de Estados Finitos y módulos de *Case-Based Reasoning*

En el contexto de este Trabajo de Fin de Grado, el *workspace* de ICI constituye la **plataforma de simulación** sobre la que se recopilan los datos del juego para entrenar los modelos de redes neuronales descritas en los capítulos posteriores y se prueban (véase Figura 3.10). Su API expone el estado del juego (`Game`), utilidades de distancia, cálculo de caminos óptimos y un protocolo de control basado en llamadas periódicas a `getMove()`, lo que permite integrar algoritmos de IA en tiempo real sin modificar la lógica interna del motor del juego.

En particular, el entorno garantiza la reproducibilidad, puesto que todo el código y los *assets* son de libre descarga; facilita la escalabilidad docente gracias a que incluye controladores de referencia —aleatorios, heurísticos, FSM, etc.— que sirven de *benchmark* inicial; y ofrece una gran flexibilidad, ya que su arquitectura modular acelera el prototipado de nuevas técnicas de aprendizaje, incluidas redes neuronales y métodos de aprendizaje por refuerzo.

La disponibilidad de esta base consolidada ha permitido centrar los esfuerzos del TFG en el *aprendizaje de comportamientos* y no en la programación del motor del juego, alineando la contribución del proyecto con los objetivos académicos de la asignatura ICI y con las tendencias actuales de investigación en IA para videojuegos.

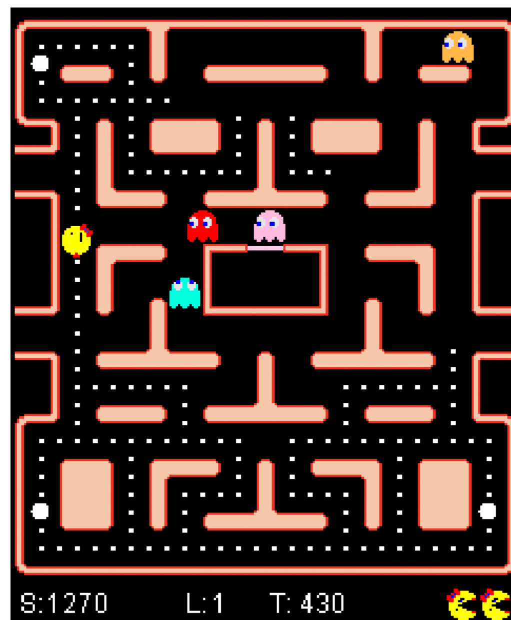

Figura 3.10: Simulador de Ms. Pac-Man utilizado en la asignatura ICI, adaptado por el Prof. Recio-García a partir del entorno competitivo de CIG 2016.

Capítulo 4

Arquitectura y descripción funcional del Trabajo

4.1. Entorno Java para la gestión de datos y evaluación

Para desarrollar esta sección, ha sido imprescindible el material proporcionado por la asignatura de Ingeniería de Comportamientos Inteligentes (ICI), un workspace en Java con la implementación del juego del Pac-Man. Este workspace incluye la lógica del juego y un entorno visual que nos permite observar el comportamiento de los distintos modelos de redes neuronales.

Antes de comenzar con la gestión de datos, se ha realizado un estudio sobre las características que componen el estado del juego con el fin de identificar las variables clave que aportan información relevante y descartar las que son innecesarias para el entrenamiento de los distintos modelos de redes neuronales.

Por otra parte, se ha llevado a cabo una serie de modificaciones con el fin de permitir la recopilación y procesamiento de datos del estado del juego para generar datasets con distintas características. Estos se usarán para el entrenamiento de múltiples modelos de redes neuronales.

Otra de las modificaciones principales ha sido el desarrollo de sockets para establecer una conexión a tiempo real entre Python y Java, lo que permite la interacción de redes neuronales ya entrenadas con el propio juego.

Se ha considerado relevante incluir un diagrama de clases que recoge las clases más representativas del proyecto. Este diagrama tiene como objetivo proporcionar una visión general de la estructura del sistema, permitiendo entender las relaciones, dependencias y herencias entre los principales componentes. Esta representación visual, permite comprender mejor las funcionalidades que se explican en los siguientes apartados, ayudando a interpretar el uso y la interacción entre las distintas clases.

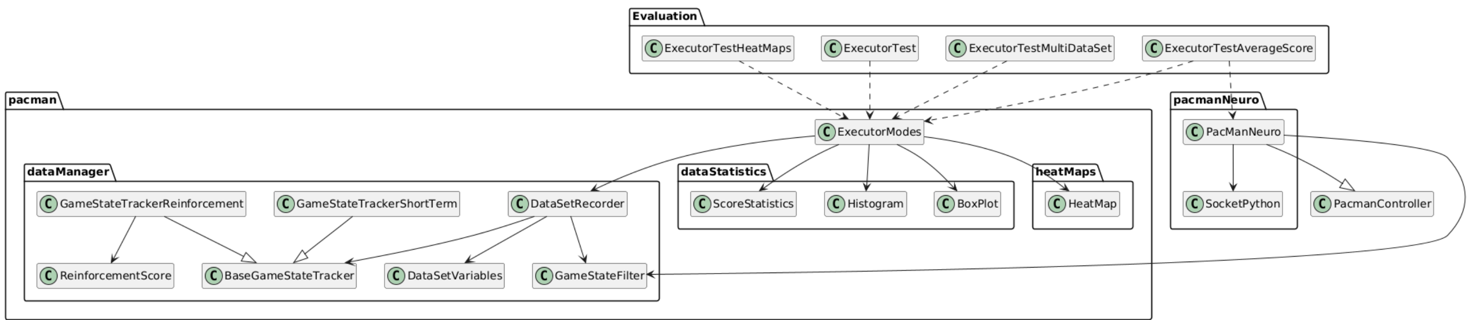

Figura 4.1: Diagrama de clases del workspace en el que se muestran los componentes y clases más significativos, facilitando entender sus funcionalidades y relaciones.

4.1.1. Adaptación del workspace

Con el fin de adaptar el entorno a las necesidades del proyecto, se han llevado a cabo una serie de modificaciones sobre el workspace original de la asignatura de Ingeniería de Comportamientos Inteligentes (ICI).

En primer lugar, se eliminaron los proyectos que no eran relevantes, principalmente aquellos enfocados a la realización de prácticas en la asignatura. Se mantuvieron únicamente aquellos proyectos indispensables para la ejecución del juego, concretamente **PacManEngine** y **Evaluation**, los cuales contienen la lógica principal del motor del juego y la evaluación de implementaciones.

Finalmente, se creó un nuevo proyecto denominado **PacmanNeuro**, encargado de implementar la lógica de Pac-Man a través de los modelos de redes neuronales entrenados previamente en Python. Lo que facilita la comunicación a tiempo real entre el entorno del juego en Java y la toma de decisiones de qué movimiento debe realizar Pac-Man. Esto se lleva a cabo a través de un socket, explicado más adelante en la Sección 4.1.4.

4.1.2. Estudio de las características del estado del juego

Para entrenar los modelos de redes neuronales, el conjunto de características utilizado ha sido definido tras un estudio previo sobre el estado del juego, con el objetivo de identificar las variables más relevantes para la toma de decisiones por parte de Pac-Man.

Estas características permiten capturar tanto información local como contextual, y han sido utilizadas en el entrenamiento de los modelos empleados en los dos primeros experimentos (Secciones 5.1 - Experimento 1 y 5.2 - Experimento 2). Para facilitar su explicación, se han agrupado en tres bloques: características relacionadas con Pac-Man, con los fantasmas y con el estado global del juego.

- **Pac-Man:** Las variables relacionadas con Pac-Man permiten describir su posición actual, su dirección de movimiento y su proximidad respecto a las Power Pills más cercanas (Tabla 4.1).

Variable	Descripción
PacmanMove	Movimiento que debe predecir el modelo.
pacmanCurrentNodeIndex	Nodo actual en el que se encuentra Pac-Man.
pacmanLastMoveMade	Último movimiento realizado por Pac-Man.
euclideanDistanceToPp	Distancia euclídea a la Power Pill más cercana.
pathDistanceToPp	Distancia del camino más corto a la Power Pill más cercana.
pacmanWasEaten	Indica si Pac-Man ha sido comido.

Tabla 4.1: Características del estado de juego relacionadas con Pac-Man

- **Fantasmas:** Cada uno de los cuatro fantasmas de cada implementación cuenta con un conjunto de variables que permiten determinar si en cada momento representan una amenaza para Pac-Man o una oportunidad para ser comidos (la letra `X` indica el número del fantasma, del 1 al 4) (Tabla 4.2).

Variable	Descripción
ghostXNodeIndex	Nodo en el que se encuentra el fantasma X.
ghostXEdibleTime	Tiempo restante durante el cual el fantasma X es comestible.
ghostXLairTime	Tiempo restante que el fantasma X debe estar en la cárcel.
ghostXLastMove	Último movimiento realizado por el fantasma X.
ghostXDistance	Distancia desde Pac-Man al fantasma X.
ghostXEaten	Indica si el fantasma X ha sido comido.

Tabla 4.2: Características del estado de juego relacionadas con los fantasmas

- **Estado del juego:** Estas variables proporcionan información contextual sobre el desarrollo de la partida y permiten adaptar la estrategia de Pac-Man en función del momento (Tabla 4.3).

Variable	Descripción
totalTime	Tiempo total transcurrido en la partida.
score	Puntuación acumulada en la partida.
timeOfLastGlobalReversal	Tick en el que los fantasmas invirtieron globalmente su dirección por última vez.
remainingPp	Número de Power Pills activas restantes.

Tabla 4.3: Características relacionadas con el estado global del juego

Este conjunto de variables no ha permanecido estático a lo largo del desarrollo del proyecto. A medida que se han aplicado técnicas de explicabilidad (Sección 4.3), se han realizado ajustes sobre el conjunto de datos original.

Por un lado, algunas variables han sido eliminadas tras comprobar que no aportaban información significativa para la predicción del movimiento de Pac-Man, además de incrementar el tamaño del espacio de características, introduciendo posibles fuentes de ruido que podían dificultar el aprendizaje del modelo.

Por otro lado, se han incorporado nuevas características con el objetivo de mejorar la representación del estado del juego y proporcionar al modelo información adicional que anteriormente no se estaba teniendo en cuenta. Esto permite mejorar la capacidad de generalización de las redes neuronales y aumentar la precisión en la toma de decisiones, al disponer de descripciones más completas del estado del juego.

4.1.3. Gestión y validación de estados del juego

Para llevar a cabo la recolección y almacenamiento de los estados del juego necesarios para entrenar los modelos de redes neuronales, se ha desarrollado un módulo específico ubicado en la carpeta **DataManager**. Este módulo contiene varias clases encargadas de filtrar, almacenar y validar los datos del juego antes de ser incluidos en el dataset final.

La clase **DataSetRecorder** se ejecuta en cada tick del juego y tiene dos funciones principales. Por un lado, comprueba si Pac-Man se encuentra en una intersección del mapa. En ese caso, el estado del juego se procesa mediante la clase **GameStateFilter**, que se encarga de eliminar características irrelevantes y calcular nuevas variables útiles. A continuación, el estado filtrado se envía a la clase **GameStateTracker**, que lo almacena temporalmente en un buffer, implementado como una cola.

Por otro lado, **DataSetRecorder** también verifica en cada tick si alguno de los estados almacenados en la cola ha alcanzado el tiempo de espera necesario para ser evaluado. Esta evaluación también se realiza mediante la clase **GameStateTracker**, que analiza el comportamiento de Pac-Man durante los ticks posteriores a ese estado: si ha mejorado su puntuación, ha comido fantasmas, o por el contrario, ha sido comido. En función de este análisis, se determina si el estado es válido o no. Solo los estados considerados válidos se almacenan finalmente en un archivo en formato **.csv**. Cabe recalcar que esta variación no se ha utilizado en todos los experimentos, en concreto en el último, ya que con esta implementación se perdía una gran cantidad de información relevante que posteriormente permitiría al agente aprender de estos estados a los que llamábamos no válidos.

Finalmente, la clase **DataSetVariables** permite gestionar de forma estructurada las listas de características implicadas en el proceso. Define qué variables del juego original se deben conservar, cuáles se eliminan y cuáles se han añadido como nuevas. Esto permite adaptar de forma flexible el conjunto de datos a los distintos experimentos realizados durante el desarrollo del proyecto.

4.1.4. Socket cliente en Java

Para establecer la conexión entre el juego y los distintos modelos de redes neuronales entrenados en Python, se ha implementado un sistema de comunicación basado en sockets. Este sistema permite el envío y recepción de información en tiempo real entre ambos entornos, garantizando una interacción continua durante la ejecución del juego.

La comunicación se inicia desde la parte de Python, donde el usuario debe lanzar manualmente el servidor socket, seleccionar el modelo de red neuronal con el que se desea trabajar y dejar el socket en escucha. Este servidor se mantiene abierto permanentemente, esperando conexiones entrantes desde Java. No se cierra automáticamente entre partidas, sino que permanece activo hasta que el usuario decide detenerlo explícitamente.

En el lado de Java, cada vez que se lanza una partida, se establece una nueva conexión con el servidor Python. La clase **SocketPython** es la encargada de gestionar esta conexión con el puerto correspondiente, permitiendo la transmisión de datos del estado del juego y la recepción del movimiento predicho por el modelo.

Su funcionamiento se basa en dos procesos principales:

- **Envío del estado del juego:** Cada vez que Pac-Man pasa por una intersección del mapa, Java envía el estado del juego ya preprocesado al servidor en Python. Esta funcionalidad está encapsulada dentro del proyecto **Pacman-Neuro**, que se encarga de detectar estas situaciones y gestionar el envío del estado a través del socket. De esta forma, se centraliza la lógica de comunicación, permitiendo la interacción entre el entorno de juego y los modelos de redes neuronales.

- **Recepción del movimiento predicho:** Python procesa el estado recibido utilizando el modelo de red neuronal seleccionado previamente, y devuelve el movimiento que debe realizar Pac-Man. Este movimiento puede ser: `RIGHT`, `LEFT`, `UP`, `DOWN` o `NEUTRAL`.

Una vez recibido el movimiento correspondiente, el juego lo ejecuta y continúa la partida. Este proceso se repite de forma continua hasta el final de la misma, momento en el que se cierra la conexión desde Java, pero el servidor en Python permanece en ejecución a la espera de nuevas partidas.

4.1.5. Distintos modos de ejecución

Para gestionar las principales funcionalidades de la implementación, se han creado distintos modos de ejecución dentro de la clase **ExecutorModes**. Esto permite ejecutar el juego en función del propósito requerido.

- **Modo de creación de datasets (runGameGenerateMultiDataSet):** Este modo permite ejecutar múltiples partidas del juego utilizando distintas

combinaciones de controladores tanto para Pac-Man como para los fantasmas, la finalidad es obtener datos sobre los estados del juego y almacenarlos en formato **.csv**.

El usuario puede configurar distintos parámetros, como el número de partidas a ejecutar por combinación, un umbral mínimo de puntuación para decidir qué estados guardar, el nombre del archivo de salida y la activación de un modo de depuración (DEBUG) que proporciona información detallada sobre cada simulación.

Una vez finalizado el proceso, se muestra un resumen con la cantidad total de datos generados y el tiempo empleado. Además, se calculan estadísticas descriptivas de las puntuaciones obtenidas en las partidas, que se acompañan de un Histograma y un Boxplot para facilitar su interpretación visual. Este análisis se describe con mayor profundidad en el Apartado 4.4.

- **Modo de evaluación del rendimiento (runGameCalculateAverageScore):**
  Este modo ejecuta partidas del juego enfrentando una implementación específica de Pac-Man (en nuestro caso, un controlador basado en un modelo de red neuronal propia) contra diferentes implementaciones de fantasmas.

  El objetivo es calcular la puntuación promedio obtenida por Pac-Man después de realizar un número determinado de partidas contra cada grupo de fantasmas. De esta forma, podemos evaluar fácilmente el rendimiento y la eficacia de los modelos entrenados. El usuario puede definir cuántas partidas jugar contra cada implementación y ajustar la velocidad de simulación.

  Al igual que en el modo anterior, al finalizar la ejecución se genera un análisis estadístico de las puntuaciones obtenidas, incluyendo sus visualizaciones correspondientes. Para más detalles, véase el Apartado 4.4.

- **Modo generación de mapas de calor (runGameHeatMaps):**

  Este modo permite visualizar mapas de calor basados en el peso de las características obtenidas mediante técnicas de explicabilidad como SHAP o Feature Importance, dependiendo del modelo.

  Durante la ejecución, se lanza una simulación del juego entre cualquier combinación de controladores para Pac-Man y fantasmas con el objetivo de poder pintar el mapa. A partir del peso de las características, se transforman esos valores en colores mediante un proceso de normalización, generando automáticamente mapas de calor que representan la relevancia, positiva o negativa, de cada característica en cada intersección del tablero. Todos los mapas generados se almacenan automáticamente en memoria tras su creación.

  El funcionamiento detallado de la transformación de datos y la lógica de asignación de colores puede consultarse en el Apartado 4.3.4.

4.2. Redes Neuronales en Python

Se han utilizado diferentes frameworks y arquitecturas de redes neuronales con el fin de investigar y analizar cuál se adapta mejor a las necesidades del proyecto. Para desarrollar los diferentes modelos hemos seguido unas pautas comunes:

4.2.1. Preprocesamiento de los datos

El preprocesamiento de los datos es necesario para el desarrollo de redes neuronales, ya que transforma los datos en un formato adecuado para poder ser entrenados posteriormente.

El estado del juego contiene información sobre los movimientos efectuados por el Pac-Man y los fantasmas, representados mediante variables categóricas (UP, DOWN, LEFT, RIGHT y NEUTRAL). Para poder ser utilizados por la red neuronal, tienen que ser transformados en variables numéricas. Para ello, utilizamos One-Hot Encoding sobre las columnas **pacmanLastMoveMade** y **ghostXLastMove**. Esto implica que cada columna se transforma en un vector binario cuya longitud es el número de posibles movimientos a realizar, donde solo una de las posiciones toma el valor 1, mientras que las restantes tendrán un 0.

La columna **PacmanMove** (movimiento tomado por el Pac-Man), se ha convertido en una variable numérica mediante la asignación de un valor numérico por cada acción posible. Se ha realizado de esta manera y no con One-Hot Encoding para facilitar su posterior decodificación al enviarlo por el socket. Existen algunas características que devuelven un valor booleano, como por ejemplo **PacmanWasEaten**. Estas se han transformado en valores 0 o 1 para su posterior tratamiento por parte del modelo.

A lo largo de los diferentes experimentos realizados nos dimos cuenta de que era necesario hacer un mejor escalado de los datos, es por ello que, por cada intersección guardamos una configuración individual de StandardScaler con el fin de normalizar las características con valores numéricos, reduciendo así el impacto de las que tienen valores más altos de forma natural. Esto permitirá que todas las características influyan de manera equilibrada en la decisión del modelo. Tras todas las transformaciones citadas se ha construido un nuevo conjunto de datos donde todas las columnas están codificadas y estructuradas de la manera correcta para su uso en el modelo.

4.2.2. Creación de modelos

Se han implementado dos modelos de clasificación, uno basado en la librería **Scikit-Learn**, y otro desarrollado con **PyTorch**. Ambos modelos representan una primera versión básica con arquitecturas simples que sirven como punto de partida para los diferentes experimentos.

El modelo con Scikit-Learn utiliza la clase MLPClassifier, la cual implementa un

perceptrón multicapa (MLP). Está compuesta por una única capa oculta de 100 neuronas, una función de activación ReLU y el optimizador Adam. De manera similar, el modelo desarrollado en PyTorch utiliza una red neuronal básica y completamente conectada. Esta red tiene una capa de entrada con un número de características, una capa oculta de 100 neuronas y una función de activación ReLU. Además, cuenta con una capa de salida con el número de clases de salida correspondientes a los posibles movimientos del Pac-Man.

Durante el desarrollo del trabajo dimos con una arquitectura llamada TabNet y decidimos probarla debido a su gran potencial. Esta red se diferencia del enfoque tradicional de las MLP al incorporar un mecanismo de atención secuencial que le permite seleccionar de forma dinámica qué características del input deben ser tenidas en cuenta en cada etapa del procesamiento. En lugar de tratar todas las variables por igual, TabNet realiza una especie de cribado inteligente, donde el modelo se "fija" en las columnas que más aportan según el contexto del ejemplo que está evaluando. Esto no solo mejora la eficiencia, sino que además aporta una mayor capacidad explicativa, ya que genera máscaras de atención que permiten visualizar qué atributos fueron relevantes en cada decisión. Su integración permitió explorar una tercera vía más avanzada y centrada en la interpretabilidad de las predicciones.

4.2.3. Entrenamiento

Para el modelo MLP con Scikit-Learn se utiliza la función que aplica validación cruzada sobre los datos recibidos para evaluar el rendimiento del modelo con diferentes divisiones de estos. Una vez realizada, el modelo es entrenado con el conjunto de datos completo separado por características y etiquetas. Este método permite evaluar el rendimiento del modelo, midiendo su precisión tanto en los datos de entrenamiento como en los de prueba.

El modelo en Pytorch se entrena repitiendo el proceso 500 veces (épocas), utilizando el optimizador Adam y la función de pérdida CrossEntropyLoss. Durante cada época se calcula la precisión y la pérdida del modelo. Una vez terminado el entrenamiento, se evalúa el modelo en los datos de entrenamiento y en los datos de validación cruzada con el objetivo de observar si el modelo ha entendido de manera correcta cómo hacer buenas predicciones. El código de entrenamiento fue adaptado del material de la asignatura *Aprendizaje Automático y Big Data* y ajustado a las necesidades del proyecto.

Para el modelo de aprendizaje basado en la arquitectura TabNet, se lleva a cabo el desarrollo de una función propia de entrenamiento, *train_tabnet_nn*, que permite ajustar los hiperparámetros más importantes de la red, tales como el número de pasos (*n_steps*), la dimensionalidad de las capas de decisión y atención (*n_d*, *n_a*) y el grado de regularización mediante *lambda_sparse*. El entrenamiento se realiza utilizando el optimizador Adam y un planificador de tasa de aprendizaje basado en la estrategia de *step decay*. Durante este procedimiento se aplica la técnica de Early Stopping, explicada en el Apartado 3.3.3, para evitar el sobreajuste monitorizando el rendimiento del modelo sobre un conjunto de validación. Al igual que en los otros modelos, una vez finalizado el proceso, se evalúa la precisión sobre un conjunto de

prueba independiente y se generan tanto el informe de clasificación como la matriz de confusión. Además, se extrae y almacena la importancia de las características aprendidas automáticamente por el modelo gracias al mecanismo de atención de TabNet, lo cual contribuye significativamente a la capacidad explicativa del sistema.

4.2.4. Socket Servidor en Python

Se ha implementado un socket para poder recibir un estado del juego en tiempo real, realizar la predicción del siguiente movimiento del Pac-Man y enviar esta predicción de vuelta al cliente. Con el fin de optimizar el proceso de intercambio de datos entre Java y Python se cargan previamente todos los modelos de redes entrenadas, uno por intersección, guardándolos en un diccionario organizado por el número de cada intersección.

La función **get_prediction** procesa el estado del juego y realiza la predicción del movimiento a realizar escogiendo del diccionario mencionado anteriormente el modelo entrenado para dicha intersección. La predicción resultante se mapea a uno de los posibles movimientos: RIGHT, LEFT, UP, DOWN o NEUTRAL. El servidor, mediante la función **start_socket**, seguirá escuchando por medio del socket nuevos estados para continuar prediciendo el movimiento en tiempo real.

4.3. Explicabilidad

Para abordar la explicabilidad de los modelos utilizados en nuestro trabajo, hemos empleado diversas técnicas que nos permiten descomponer y entender cómo las características de los datos influyen en las predicciones. En esta sección, se describen las metodologías aplicadas, que incluyen **SHAP**, **Feature Importance** y **LIME**. Para un mejor análisis de las características con mayor impacto, se ha generado una serie de mapas de calor. Con ellos podemos ver realmente cuales son las zonas del tablero donde adquieren más relevancia estas variables.

Cada una de estas técnicas fue seleccionada y adaptada para proporcionar una visión clara y detallada sobre el comportamiento de los modelos de redes neuronales en el contexto del juego *Ms. Pac-Man vs Ghosts*. A continuación, se detallan los procedimientos seguidos y las herramientas utilizadas en cada técnica para realizar un análisis exhaustivo de las contribuciones de las variables a las decisiones del modelo, facilitando la interpretación y validación de los resultados obtenidos.

4.3.1. Técnica 1: SHAP (SHapley Additive exPlanations)

Para **SHAP** [*de Delatorre AI (2024)*] con **Scikit-Learn** se seleccionan subconjuntos de los datos para los cálculos. Los datos se pueden dividir en **datos de fondo**, los cuales sirven como punto de referencia para calcular las contribuciones en las predicciones y **datos a explicar**, en los cuales se desea analizar las predicciones del

modelo. Hemos decidido escoger una muestra de 500 datos de fondo y una muestra de 50 datos a explicar.

Durante el cálculo de valores SHAP se emplea el método **KernelExplainer** de SHAP. Este método utiliza un enfoque basado en una aproximación lineal y en un concepto de "juegos cooperativos", donde las características son tratadas como jugadores que contribuyen a la predicción del modelo. El objetivo del método es medir la contribución marginal e individual de cada característica, considerando todas las combinaciones posibles de las características, tanto incluidas como excluidas.

El muestreo de los datos para **SHAP** con **Pytorch** se hace exactamente igual que para Scikit-Learn, eligiendo la misma cantidad en cada muestra. Para optimizar el rendimiento, el modelo y los datos se cargan en un dispositivo disponible (`GPU` o `CPU`), utilizando `torch.device(’’cuda’’)` si hay GPU disponible, o `torch.device(’’cpu’’)` en caso contrario.

A diferencia de Scikit-Learn, se utiliza el método **DeepExplainer**, diseñado específicamente para redes neuronales profundas (como los modelos en PyTorch). Este método utiliza gradientes del modelo para calcular cómo cada característica contribuye a la predicción. Los valores SHAP se calculan utilizando las muestras seleccionadas como datos a explicar. Esto mide la contribución individual de cada característica para las predicciones realizadas por el modelo.

Para una mejor visualización de los datos se genera un gráfico de barras para ambos modelos, el cual muestra la media de los valores SHAP absolutos ordenados, mostrando así la importancia de cada característica.

4.3.2. Técnica 2: Feature Importance

La siguiente técnica no puede ser usada para el modelo de Pytorch, al ser un framework de bajo nivel. Por lo que esta técnica será empleada para nuestros modelos de Scikit-Learn y TabNet.

A diferencia de **Scikit-Learn** donde se selecciona un subconjunto aleatorio de 1500 instancias del conjunto de datos para aplicar la técnica de **Permutation Importance**, **TabNet** emplea un muestreo de datos interno y dinámico. Es decir, el modelo decide qué características consultar y cuales ignorar, asignando un peso a cada una en función de su relevancia en cada paso de decisión.

Tras el entrenamiento del modelo de Scikit-Learn se calcula la importancia media de cada característica en función de cómo su permutación afecta el rendimiento del modelo. Estas se guardan en un DataFrame ordenadas de mayor a menor. Sin embargo en TabNet la importancia de cada característica se guarda ordenada igual que en Scikit-Learn al finalizar el entrenamiento de forma automática.

Para ambos modelos se utiliza un gráfico de barras horizontales para representar las importancias promedio de las características.

4.3.3. Técnica 3: LIME (Local Interpretable Model-Independent Explanations)

Debido a la arquitectura específica del agente implementado, en la que se entrena una red neuronal independiente para cada intersección del mapa, ha sido necesario adaptar la aplicación tradicional de la técnica LIME para que su uso tenga sentido en este contexto.

En lugar de seleccionar una instancia aleatoria del conjunto de datos completo —lo cual podría corresponder a cualquier intersección del mapa y, por tanto, a una red neuronal distinta—, se ha modificado el procedimiento de forma que el usuario seleccione manualmente la intersección que desea analizar. A partir de dicha selección, se identifica automáticamente el modelo correspondiente (extraído del nombre del archivo del modelo) y se filtran los estados del conjunto de datos que hayan sido generados en esa misma intersección, utilizando para ello la característica **pacmanCurrentNodeIndex**.

Una vez filtrados los estados, se selecciona aleatoriamente una instancia válida. Para asegurar una correcta interpretación de las características, se convierte el conjunto de datos a formato **DataFrame** en caso de no estarlo, y se asignan los nombres de las columnas utilizando el atributo **feature_names_in_** del modelo. Posteriormente, se configura un objeto **LimeTabularExplainer**, diseñado para datos tabulares. Este se inicializa con los valores del conjunto de datos, el modo "classification", la lista de nombres de características y la opción de discretización automática de variables continuas, lo que facilita la comprensión de los resultados.

La explicación se genera mediante el método **explain_instance**, que analiza la influencia de cada característica en la predicción del modelo para la instancia seleccionada. Para ello, el modelo debe implementar el método **predict_proba**, ya que LIME se basa en las probabilidades devueltas para calcular el peso de cada característica. En el caso de modelos desarrollados con **PyTorch**, se implementa manualmente una función **predict_proba** que aplica **softmax** a la salida del modelo para obtener la distribución de probabilidades.

Los resultados obtenidos se almacenan en memoria junto con el nombre del modelo, el movimiento predicho y el estado analizado. Además, se imprimen en consola, se guardan en un archivo **.txt** con formato legible y se genera una visualización utilizando **matplotlib**. En dicho gráfico, el eje Y muestra las características más influyentes en ese estado particular y su valor asociado, mientras que el eje X representa el impacto de cada una sobre la predicción del modelo.

Gracias a esta adaptación, es posible aplicar LIME de forma localizada y coherente, obteniendo explicaciones detalladas sobre el comportamiento de una red neuronal concreta en una intersección específica del mapa.

4.3.4. Mapas de calor

Con la finalidad de profundizar en la interpretabilidad de las decisiones que tomaron ambos modelos, se ha generado un conjunto de mapas de calor sobre las

características con más impacto según la técnica SHAP para los modelos Pytorch y Scikit-Learn, y Feature Importance para TabNet. Debido a las diferencias en sus técnicas de computación, se normalizaron estos valores de impacto durante el cálculo del color asignado a cada intersección.

Se generaron documentos de texto independientes por cada característica con mayor impacto, donde por cada intersección del tablero del juego se guarda el valor asignado a cada una de ellas. Dichos documentos fueron posteriormente utilizados en un método en Java creado, especificamente, para convertir estos valores numéricos en colores, generando así mapas visuales claros y comprensibles. Este método se encarga específicamente de normalizar el impacto según el modelo evaluado. Posteriormente se divide entre un valor máximo predefinido obteniendo un valor llamado ratio.

$$ratio = \frac{|impacto|}{impacto_{max}}$$

Este valor obtenido permite definir el color a colocar dentro de cada intersección siguiendo la siguiente estructura. Aparecerá un color de rojo a naranja si el ratio está entre 0 - 0.33, de naranja a amarillo si está entre 0.33 - 0.66 y de amarillo a verde para valores entre 0.66 - 1.

Estos mapas de calor hacen que sea posible identificar de forma más rápida y visual las partes del tablero donde las características tienen mayor o menor peso en las decisiones del modelo. Esto nos permite detectar donde puede estar tomando decisiones inesperadas o poco eficientes, para así poder mejorar y optimizar el rendimiento del modelo.

4.4. Análisis estadístico descriptivo de las puntuaciones obtenidas en las partidas

Este análisis estadístico, enfocado en resumir y caracterizar las puntuaciones obtenidas en las partidas jugadas, será empleado principalmente en dos contextos diferentes dentro del proyecto. El primero de ellos al **generar datasets** a partir de partidas jugadas con diferentes implementaciones de Pac-Man y fantasmas. Estas estadísticas permiten analizar el comportamiento general del conjunto de datos antes de utilizar estos para el entrenamiento de los distintos modelos de redes neuronales.

Por otro lado, nos permitirá evaluar el desempeño de los **modelos entrenados** ya que se enfrentarán contra distintas implementaciones de fantasmas. La comparación de las estadísticas obtenidas por el modelo, con aquellas del dataset original permite determinar hasta qué punto el modelo ha sido capaz de capturar y reproducir el comportamiento aprendido durante el entrenamiento.

Para complementar los resultados de las estadísticas se han generado tanto un **Histograma** como un diagrama de caja (**Boxplot**). Ambas visualizaciones permiten interpretar de forma más sencilla y visual la distribución de los datos.

La base teórica utilizada para describir las estadísticas y representaciones gráficas

en los siguientes apartados ha sido obtenida principalmente del libro *OpenIntro Statistics* Diez et al. (2024).

Capítulo 5

Evaluación Experimental

5.1. Evaluación nº 1: Una única red neuronal

En este experimento, se ha entrenado una red neuronal que representa una primera versión del modelo de clasificación mencionado previamente. El dataset empleado en este experimento se ha generado a partir de partidas jugadas con diversas implementaciones de Pac-Man y fantasmas de años anteriores, seleccionando aquellas con un rendimiento destacado en competiciones previas. Esto asegura que los datos recopilados reflejan estrategias de juego bien desarrolladas, proporcionando un conjunto de entrenamiento representativo y variado. Para una descripción completa de las características incluidas en el dataset, se puede consultar la Sección 4.1.2.

Las partidas jugadas para la creación del dataset muestran una amplia variabilidad en las puntuaciones obtenidas, lo que permite capturar distintos niveles de desempeño y escenarios de juego. En términos estadísticos, y según se muestra en las Figuras 5.1 y 5.2, la puntuación media registrada es de 3745, mientras que la mediana se sitúa en 2900, indicando una distribución con algunos valores extremos. La desviación típica es de 2986, reflejando una alta dispersión en los resultados, con un rango total de 18080 puntos entre la mejor y la peor partida registrada. Además, los percentiles 25 y 75 se encuentran en 1220 y 5540 respectivamente, lo que indica que la mayoría de las partidas oscilan en dicho intervalo. Finalmente, la asimetría (0,95) y la curtosis (0,13) revelan que la distribución de las puntuaciones está sesgada ligeramente hacia valores más altos, sin una concentración extrema de outliers.

Estos datos serán fundamentales para comparar posteriormente el desempeño de los modelos entrenados y evaluar hasta qué punto logran reproducir las tendencias observadas en el conjunto de datos original.

Para llevar a cabo este experimento, se han implementado dos versiones del modelo de clasificación: una basada en **Scikit-Learn** y otra en **PyTorch**. El conjunto de características utilizado para el entrenamiento de estos modelos es el explicado en la Sección 4.1.2. La comparación entre ambas implementaciones se realiza mediante el estudio de las estadísticas del modelo jugando partidas reales.

El propósito principal de este experimento es establecer una base de comparación con el modelo basado en **una red por intersección**, que será desarrollado en el próximo experimento y descrito en la Sección 5.2. Este análisis permitirá evaluar la mejora en el rendimiento que se obtiene al utilizar una arquitectura que segmenta la toma de decisiones por intersecciones, en lugar de una única red neuronal entrenada con el conjunto completo de datos.

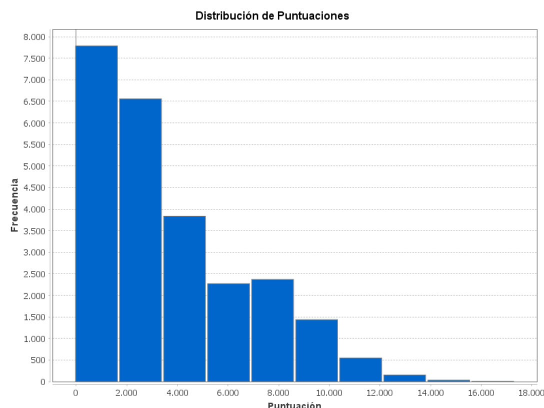

Figura 5.1: Histograma Dataset Exp.1

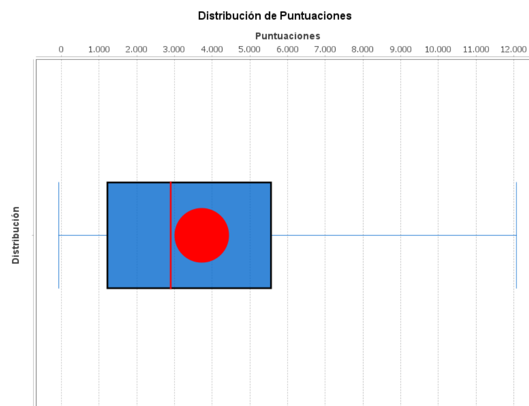

Figura 5.2: Boxplot Dataset Exp.1

5.1.1. Implementación PyTorch

Con el objetivo de poder observar una mejora entre experimentos se ha generado un conjunto de estadísticas poniendo a jugar al modelo un total de 600 partidas contra distintas implementaciones de fantasmas.

Estadística	**Valor**
Media	1514,47
Mediana	1470,00
Desviación típica	652,58
Máximo	4380,00
Mínimo	20,00
Rango	4360,00
Percentil 25 (Q1)	1050,00
Percentil 75 (Q3)	1887,50
Percentil 90	2300,00
Asimetría (Skewness)	0,74
Curtosis (Kurtosis)	1,26

Tabla 5.1: Estadísticas del modelo **PyTorch** (Experimento 1).

Estadística	**Valor**
Media	1685,22
Mediana	1680,00
Desviación típica	685,46
Máximo	4440,00
Mínimo	270,00
Rango	4170,00
Percentil 25 (Q1)	1220,00
Percentil 75 (Q3)	2050,00
Percentil 90	2469,00
Asimetría (Skewness)	0,59
Curtosis (Kurtosis)	0,98

Tabla 5.2: Estadísticas del modelo **Scikit-Learn** (Experimento 1).

La Tabla 5.3 muestra la diferencia entre el máximo (4380) y el mínimo de puntuación (20), lo que indica que existen fallos críticos en ciertos momentos. Probablemente, esas partidas con una alta puntuación están asociadas con momentos donde el Pac-Man está huyendo de los fantasmas y posteriormente recoge una PowerPill, pudiendo

devorar varios fantasmas seguidos, consiguiendo así maximizar la puntuación obtenida. Esto sostiene el valor de la desviación típica de 652,58, sugieriendo que el agente puede que no reaccione igual ante diferentes implementaciones de fantasmas.

El valor de la curtosis inferior a 3 indica una distribución más aplanada que una normal. En el contexto del Pac-man indica que hay más partidas en los extremos (bajas y altas) que en una distribución normal.

Tanto el Histograma 5.3 como el Boxplot 5.5 permiten observar una asimetría en los datos. El rango intercuartílico se encuentra entre los percentiles 25 y 75, es decir, el 50 % de las partidas están en una puntuación de entre 1050 y 1887,50. Esa diferencia de 837,50 indica que hay una diversidad considerable en el rendimiento del agente. El hecho de que el círculo rojo esté a la derecha de la línea roja, es decir, que la media esté por encima de la mediana confirma de manera visual que hay más partidas por debajo de la media, pero como vimos anteriormente, existen partidas con puntuaciones elevadas que empujan la media hacia arriba.

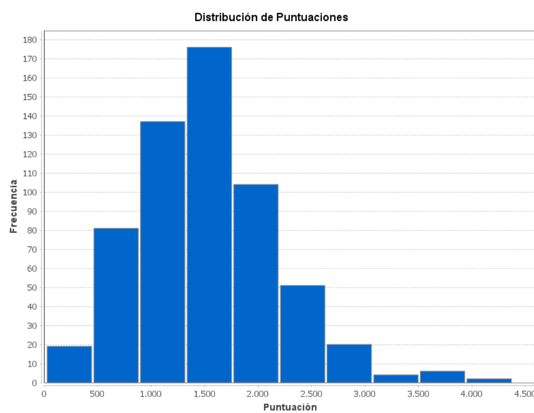

Figura 5.3: Histograma de PyTorch (Exp.1)

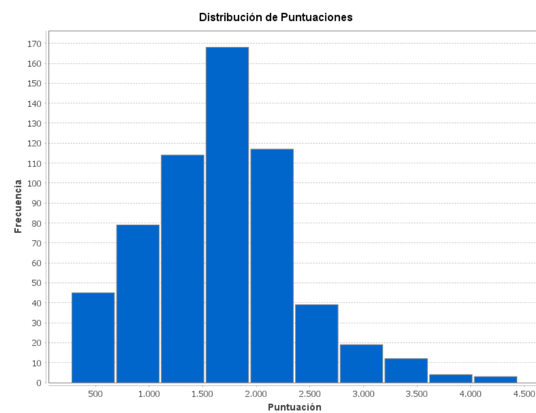

Figura 5.4: Histograma de Scikit-Learn (Exp.1)

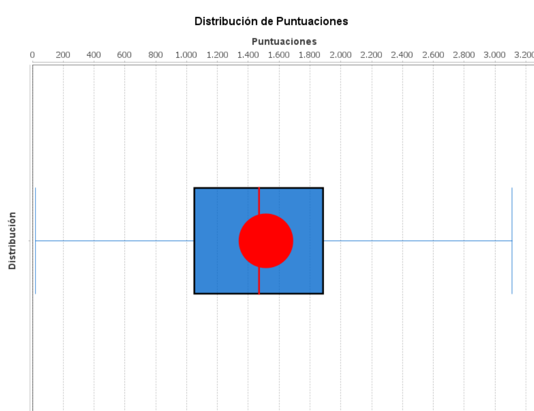

Figura 5.5: Boxplot de PyTorch (Exp.1)

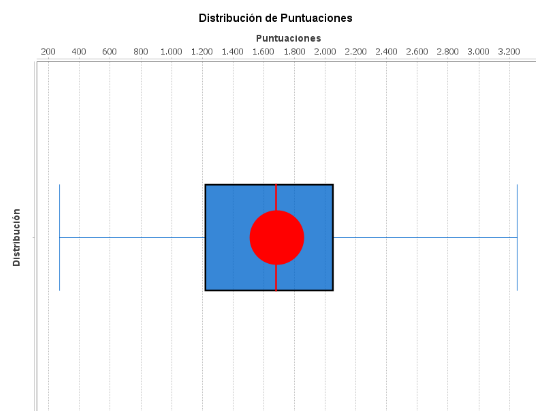

Figura 5.6: Boxplot de Scikit-Learn (Exp.1)

Se concluye que el modelo tiene un amplio rango de mejora, ya que a pesar de mostrar un rendimiento aceptable con partidas destacadas, muestra una alta variabilidad e inconsistencias claras.

5.1.2. Implementación Scikit-Learn

Al igual que con la versión en PyTorch, se realizaron un total de 600 partidas contra diversas implementaciones de fantasmas, a fin de evaluar el desempeño del modelo basado en **Scikit-Learn** para una sola intersección. Los resultados se muestran en la Tabla 5.4.

A simple vista, en el Boxplot 5.6 se observa que la mediana y la media están prácticamente alineadas, lo que sugiere una distribución menos sesgada en comparación con la versión en PyTorch, donde la media se alejaba más de la mediana. El rango intercuartílico se extiende aproximadamente entre 1000 y 1900 puntos, abarcando la mitad central de las partidas y evidenciando, aun así, una diferencia considerable en los resultados.

Al examinar el Histograma 5.4, se ve que la mayor densidad de puntuaciones se concentra a partir de los 500 puntos y llega hasta cifras próximas a los 4500. Esto indica una variabilidad notable: aunque existe un gran bloque de partidas en el tramo intermedio, algunos resultados se sitúan claramente por encima de la media, lo que contribuye a que la distribución se extienda hacia la derecha. Aun así, la ausencia de puntuaciones muy bajas (como las observadas con PyTorch) podría apuntar a que el agente comete menos errores catastróficos en esta implementación.

La dispersión queda reflejada en una desviación típica que, aunque no eliminan por completo la amplitud de los valores, señala una tendencia algo más consistente que la vista previamente con PyTorch. La curtosis inferior a 3, al igual que sucedía antes, indica que la distribución sigue siendo relativamente aplanada, con colas pobladas y partidas destacadas (tanto en el rango medio-alto como en el extremo superior).

En conjunto, estos resultados señalan que la versión con Scikit-Learn mantiene un rendimiento irregular, pero con una menor incidencia de puntuaciones muy bajas y un sesgo más reducido. No obstante, la presencia de partidas con puntuaciones muy altos demuestra que el agente puede lograr resultados sobresalientes en ciertos escenarios. Ajustes adicionales podrían mejorar todavía más su consistencia, reduciendo la dispersión y sacando mayor partido a las rondas en las que el Pac-Man se ve favorecido por su estrategia.

5.1.3. Conclusiones

En esta subsección, se proponen unas conclusiones generales tras el análisis del Experimento 1 para los modelos de PyTorch y Scikit-Learn entrenados sobre un *dataset unificado* que incluye todos los estados del juego:

El entrenamiento de una sola red neuronal para clasificar los movimientos de todas las intersecciones a partir de un dataset conjunto demostró ser viable para obtener un primer agente con capacidades de juego. Sin embargo, la gran diversidad de escenarios que contiene el dataset dificulta la correcta adaptación del modelo a los distintos contextos del mapa.

Los resultados han demostrado un *rendimiento promedio aceptable* para ambos frameworks (PyTorch y Scikit-Learn), aunque se sitúan por debajo de las puntuaciones

de los controladores empleados para generar el dataset.

Las estadísticas reflejan que el agente puede alcanzar partidas con puntuaciones destacadas, pero también cometer fallos críticos. Esto se ve en la amplitud del rango de puntuaciones en ambos modelos.

Ambas implementaciones consiguieron un comportamiento global similar, pero se aprecian ligeras diferencias. El modelo de PyTorch muestra más flexibilidad, aunque con mayor dispersión en las puntuaciones, mientras que la versión de Scikit-Learn presenta menor sensibilidad a las situaciones extremas, logrando una distribución algo más estable.

La principal limitación de un enfoque unificado es no aprovechar las diferencias entre intersecciones: el modelo debe aprender diversas estrategias para cada región del mapa en una sola red.

En general, este experimento es útil como punto de partida, deja margen de mejora para facilitar la comparación con los siguientes experimentos donde se implementará una red por cada intersección, aumentando la complejidad de estas.

5.2. Evaluación nº 2: Una red neuronal por intersección

En este segundo experimento se propone un enfoque diferente al del Experimento 1. En lugar de entrenar una única red neuronal para predecir los movimientos de Pac-Man en cualquier situación, se implementará una arquitectura en la que cada intersección del mapa tiene su propia red neuronal. El objetivo principal es evaluar si esta estrategia permite al modelo adaptarse mejor a las situaciones que ocurren en ese lugar específico del mapa, permitiendo una toma de decisiones más precisa y efectiva en función de la intersección que se encuentra Pac-Man.

Para garantizar una comparación justa con el experimento anterior, se utiliza exactamente el mismo conjunto de características (Sección 4.1.2) y el mismo dataset descrito en la Sección 5.1. La única diferencia en el procesamiento de los datos se encuentra en la separación del conjunto de datos por intersecciones, utilizando la característica *pacmanCurrentNodeIndex* como criterio para asignar cada estado de juego a la red neuronal correspondiente. De este modo, cada modelo entrena exclusivamente con los datos de una intersección específica, en contraste con el enfoque global del Experimento 1.

La arquitectura de cada red neuronal es idéntica a la utilizada en el experimento anterior, lo que permite analizar de forma concreta el impacto del cambio de una única red neuronal a una por cada intersección.

Un aspecto fundamental de este experimento es el análisis comparativo de los resultados. Las conclusiones obtenidas se basan en la comparación de las estadísticas de los modelos entrenados con las estadísticas del dataset original, presentadas en el Experimento 1. De este modo, se podrá determinar hasta qué punto el modelo basado en redes por intersección logra capturar con mayor precisión los patrones de

juego presentes en los datos originales, y si esto supone una mejora respecto a la estrategia de una única red neuronal para todo el mapa.

5.2.1. Rendimiento

El análisis comparativo entre las estadísticas del dataset original y los resultados obtenidos por los modelos entrenados con PyTorch y Scikit-Learn permite identificar diferencias significativas en cuanto a rendimiento, estabilidad y comportamiento de cada modelo. A continuación se detallan los principales hallazgos.

Estadística	Valor
Media	2368,30
Mediana	2250,00
Desviación típica	1119,38
Máximo	7470,00
Mínimo	330,00
Rango	7140,00
Percentil 25 (Q1)	1560,00
Percentil 75 (Q3)	2940,00
Percentil 90	3839,00
Asimetría (Skewness)	0,92
Curtosis (Kurtosis)	1,27

Tabla 5.3: Estadísticas del modelo **PyTorch** (Experimento 1).

Estadística	Valor
Media	2101,43
Mediana	2055,00
Desviación típica	878,35
Máximo	7490,00
Mínimo	350,00
Rango	7140,00
Percentil 25 (Q1)	1452,50
Percentil 75 (Q3)	2570,00
Percentil 90	3089,00
Asimetría (Skewness)	1,00
Curtosis (Kurtosis)	2,75

Tabla 5.4: Estadísticas del modelo **Scikit-Learn** (Experimento 1).

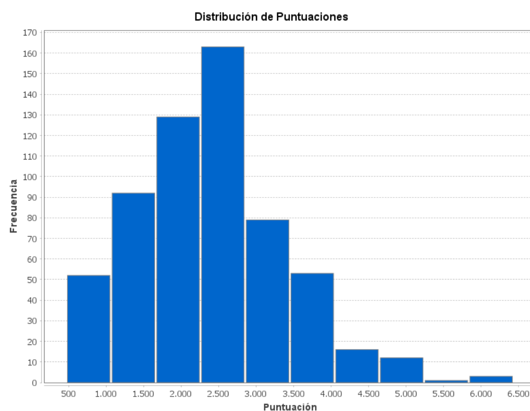

Figura 5.7: Histograma de Pytorch (Exp.2)

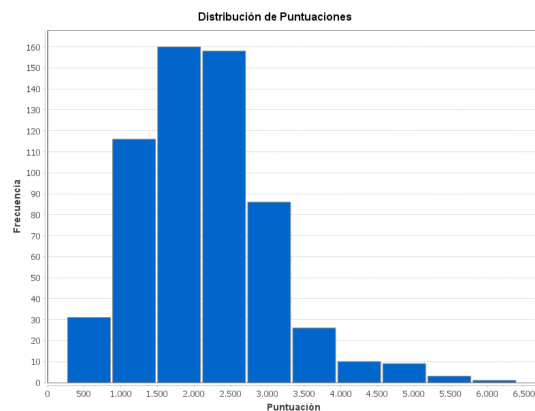

Figura 5.8: Histograma de Scikit-Learn (Exp.2)

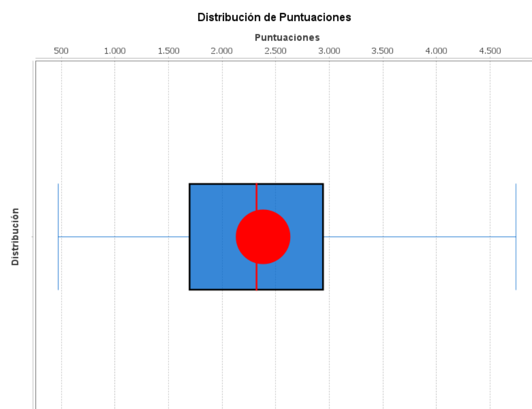

Figura 5.9: Boxplot de Pytorch (Exp.2)

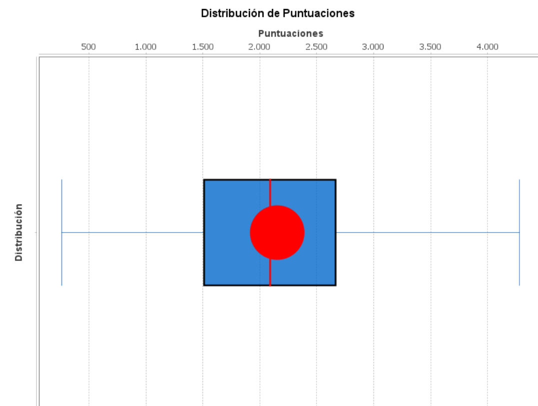

Figura 5.10: Boxplot de Scikit-Learn (Exp.2)

5.2.1.1. Implementación Pytorch

Las estadísticas revelan que el modelo PyTorch logra una media y mediana relativamente altas, lo que sugiere un buen desempeño promedio en la mayoría de partidas.

Sin embargo, la desviación típica indica un rango amplio de puntuaciones, mostrando partidas muy exitosas frente a otras no tan buenas. El máximo alcanzado (7470) confirma que el modelo puede obtener puntuaciones elevadas en circunstancias favorables, mientras que un mínimo de 330 deja entrever la presencia de algunos encuentros desventajosos. De hecho, los percentiles (Q1, Q3 y P90) refuerzan la idea de que la mayoría de las puntuaciones se sitúa por encima de 1500 puntos, y que el 10 % superior llega a rozar o superar la barrera de los 3800 puntos.

Por último, la asimetría (0,92) sugiere la existencia de una cola derecha ligeramente alargada (es decir, aparecen valores muy altos), al tiempo que la curtosis (1,27) denota una distribución algo más aplanada que la normal, sin colas excesivamente pesadas. Estos dos parámetros, en conjunto, describen un modelo que, si bien produce resultados destacados en algunas partidas, tiende a mantener la mayoría de las puntuaciones en un rango medio-alto

5.2.1.2. Implementación Scikit-Learn

Al igual que con la versión en PyTorch, se realizaron un total de 600 partidas contra diversas implementaciones de fantasmas. Las estadísticas muestran que el modelo de Scikit-Learn obtiene tanto media como mediana algo más bajas que el modelo PyTorch, lo cual apunta a un rendimiento medio más bajo en términos de puntuación.

Por otro lado, el modelo de **Scikit-Learn presenta una mayor estabilidad**, lo que puede observarse en la comparativa de los Histogramas 5.7 y 5.8. Si bien esto resulta ventajoso en ciertos casos, otro punto a tener en cuenta puede ser una menor sensibilidad del modelo que lo conduzca a no captar o no adaptarse convenientemente a las diferencias que existen realmente entre los casos gracias a la variabilidad en los

datos. El máximo (7490) es prácticamente equiparable al de PyTorch, la distribución completa de resultados se decanta ligeramente hacia valores algo más bajos, como lo evidencian los percentiles (Q1, Q3 y P90). Dichos percentiles apuntan a que la mayoría de las puntuaciones suele rondar la zona de 1500 a 2500 puntos, sin elevarse tanto como en el caso de PyTorch.

La forma de la **distribución de los valores predichos** también varía en función de los distintos modelos (Boxplot 5.9 y 5.10). En el caso de PyTorch, muestra una distribución más amplia, lo que puede interpretarse como una mayor exploración del espacio de predicción. Sin embargo, **Scikit-Learn tiene una distribución más concentrada**, ya que sitúa la mayoría de sus predicciones alrededor de determinados valores; esta es, además, la que muestra una menor dispersión y, por tanto, la más estable.

Finalmente, la asimetría (1,00) indica la presencia de colas a la derecha un poco más acentuadas, mientras que la curtosis de 2,75 sugiere colas relativamente pesadas, más que en el modelo PyTorch. En conjunto, estas últimas medidas apuntan a que, si bien el modelo puede llegar a producir puntuaciones altas en escenarios puntuales, su rendimiento se concentra mayormente en valores medios, con una tendencia a generar outliers (partidas excepcionalmente buenas o malas) en menor proporción que PyTorch, pero con mayor peso en la cola cuando suceden.

5.2.2. Explicabilidad

En esta sección se presentan los resultados obtenidos con la implementación del modelo de **PyTorch** y **Scikit-Learn**. Se analiza su desempeño en comparación con otras versiones del modelo, evaluando su capacidad para generalizar el comportamiento de Pac-Man. Además, se realiza un estudio detallado mediante modelos de explicabilidad y mapas de calor, lo que permite interpretar las decisiones de la red neuronal y comprender los factores que influyen en su toma de decisiones.

5.2.2.1. Explicabilidad modelo Pytorch

1. SHAP

En la Gráfica 5.11, se puede observar que las características **totalTime** y **score** tienen un gran impacto en la predicción. Esto se debe a que el modelo busca en todo momento una mejoría de la puntuación, recogiendo únicamente los estados del juego en los que haya una diferencia de puntuación, con respecto a estados anteriores, superior a un cierto umbral. El tiempo total que lleva el Pac-Man vivo junto con su puntuación reflejan qué tan bien está jugando el modelo, es por ello que tiene ese impacto tan elevado.

La característica **timeOfLastGlobalReversal** refleja el estado temporal del juego y puede sugerir posibles cambios en los comportamientos de los fantasmas, lo cual influye directamente en las decisiones estratégicas que debe tomar Pac-Man para maximizar su puntuación.

Estas 3 primeras características destacan muy por encima del resto, por lo que es muy probable que el modelo les asigne tanto peso por los valores tan altos que toman en el conjunto de datos. Para igualar, será necesario normalizar los datos.

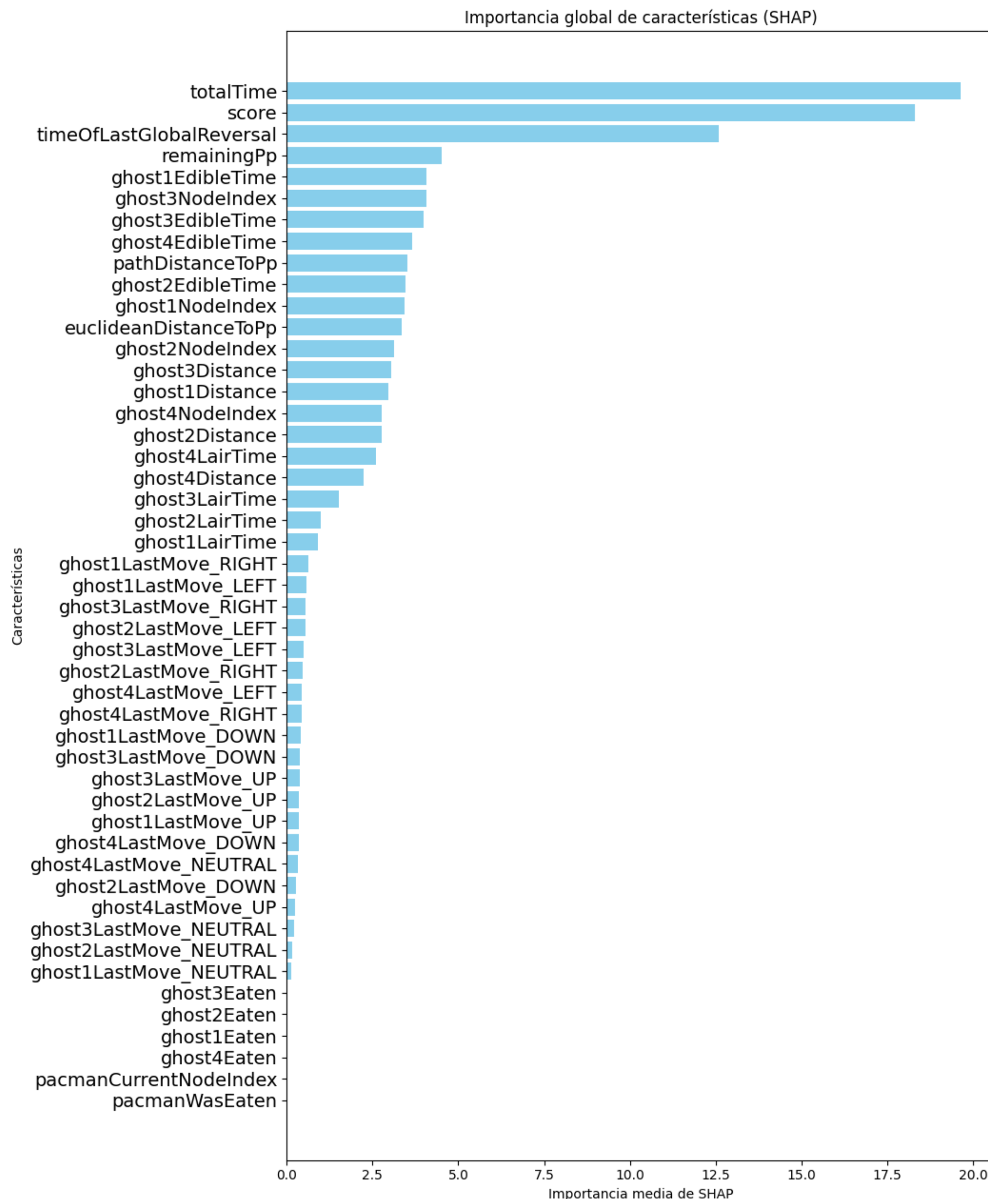

Figura 5.11: Importancia de características con SHAP en un modelo de Pytorch.

En la Figura 5.11 aparecen un conjunto de características con un impacto similar en las decisiones del modelo, ya que todas ellas están relacionadas con cómo Pac-Man percibe su entorno en una situación real de juego. Estas son **remainingPp**, **ghostXNodeIndex**, **ghostXDistance**, **ghostXLairTime**, **distanceToPp**.

El hecho de que estas características tengan una relevancia equilibrada indica que el modelo considera no uno, sino varios factores al tomar decisiones. La distancia a los fantasmas y su vulnerabilidad influyen en si Pac-Man adopta una estrategia ofensiva

o defensiva. Por otro lado, la distancia a la PowerPill más cercana puede hacer que priorice recogerla en ciertos momentos. Además, el número de pills restantes da una aproximación de cuál es el progreso de la partida y afectará a la estrategia general.

Las últimas características no tienen impacto en las decisiones del modelo, ya que solo se usan para separar por intersecciones o no aportan información útil a PacMan en una situación de juego, como es el caso de **PacmanWasEaten** y **GhostXEaten**.

Mapas de calor de las características más significativas

Tal como indica la explicabilidad, hay tres características cuyo impacto domina claramente sobre las demás. Esto queda reflejado en la mayoría de las intersecciones, donde su influencia es considerablemente superior en comparación con el resto. Este patrón confirma que la estrategia de Pac-Man se basa principalmente en el tiempo total transcurrido en la partida, la puntuación acumulada y los cambios globales del estado de los fantasmas.

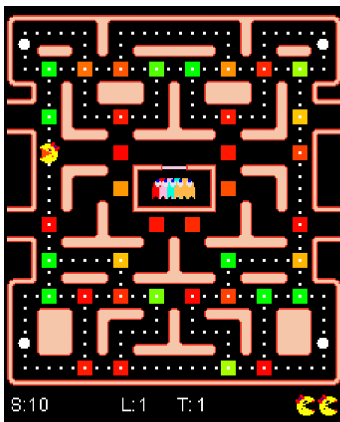

Figura 5.12: Mapa de calor del modelo Pytorch para la característica totalTime

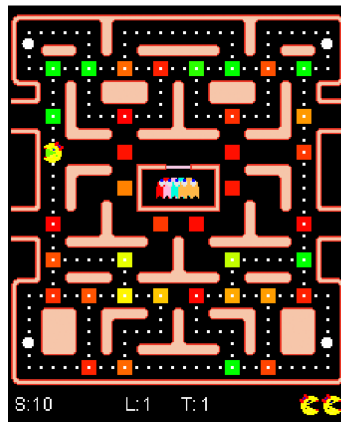

Figura 5.13: Mapa de calor del modelo Pytorch para la característica score

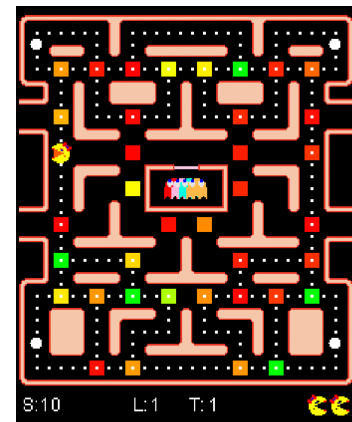

Figura 5.14: Mapa de calor del modelo Pytorch para la característica timeOfLastTimeReversal

Por otro lado, las características que ocupan posiciones más bajas en el ranking de importancia tienen un impacto global menor, aunque en algunas intersecciones específicas su relevancia aumenta. Esto ocurre en momentos clave del juego, donde la información que aportan es crucial para la toma de decisiones. Un ejemplo claro es **ghostXEdibleTime**, la cual tiene un mayor impacto cerca de las Power Pills. Esto indica que el modelo reconoce estos puntos como estratégicos, ya que cuando un fantasma es vulnerable, Pac-Man puede aprovechar la oportunidad para comérselo.

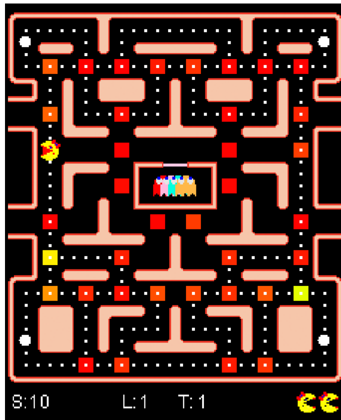

Figura 5.15: Mapa de calor del modelo Pytorch para la característica ghost1EdibleTime

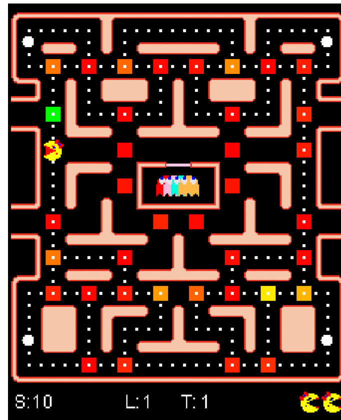

Figura 5.16: Mapa de calor del modelo Pytorch para la característica ghost3EdibleTime

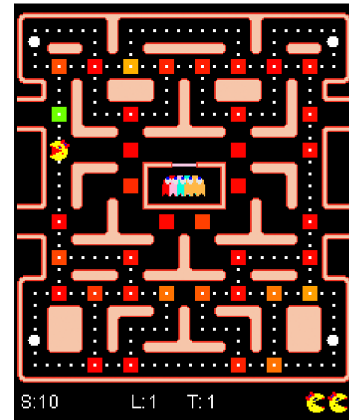

Figura 5.17: Mapa de calor del modelo Pytorch para la característica ghost4EdibleTime

Se puede apreciar que las características **pathDistanceToPp** (Figura 5.18) y **euclideanDistanceToPp** (Figura 5.19) tienen practicamente un 100 % de correlación, ya que en ambos mapas de calor muestran un impacto idéntico en las mismas posiciones del tablero. En futuros experimentos una de estas características será eliminada.

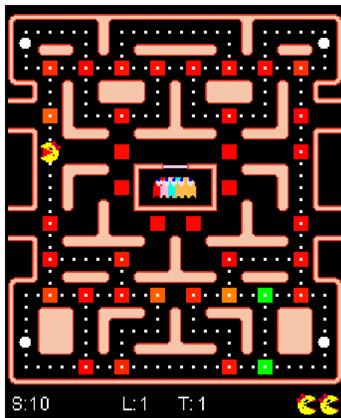

Figura 5.18: Mapa de calor del modelo Pytorch para la característica pathDistanceToPp

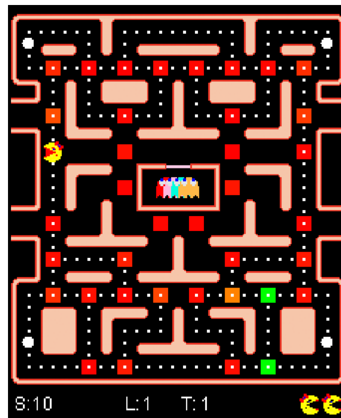

Figura 5.19: Mapa de calor del modelo Pytorch para la característica euclideanDistanceToPp

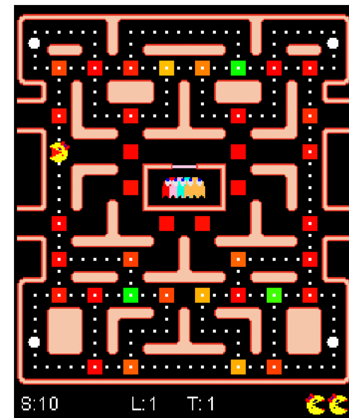

Figura 5.20: Mapa de calor del modelo Pytorch para la característica remainingPp

2. LIME

Para el análisis con esta técnica de explicabilidad, se ha seleccionado una intersección específica del mapa del juego. Dado que LIME se enfoca en estudiar estados concretos del conjunto de datos, el objetivo principal es evaluar la explicabilidad asociada a una única red neuronal específica del modelo implementado en PyTorch.

La intersección seleccionada se encuentra en la zona superior izquierda del mapa, considerada un punto estratégico por diversas razones. En primer lugar, su proximidad a una Power Pill implica que Pac-Man puede enfrentarse frecuentemente a situaciones clave, como la posibilidad de comer fantasmas o, por el contrario, huir de ellos mientras estos intentan acorralarla en la esquina. En segundo lugar, dicha intersección permite un total de cuatro movimientos posibles, lo que añade complejidad al proceso de toma de decisiones. Estos factores la convierten en una zona de interés clave para analizar cómo actúa el agente en situaciones relevantes y dinámicas.

Otro motivo relevante para seleccionar esta intersección radica en los resultados observados en el experimento previo (Sección 5.1), donde se detectó que Pac-Man frecuentemente realizaba bucles en las esquinas del mapa. Este análisis busca verificar que dicho comportamiento haya sido corregido y que las acciones actuales de Pac-Man muestren coherencia con respecto a cada estado específico del juego.

La intersección concreta seleccionada para el análisis está representada en la Imagen 5.21. A partir de ella se han seleccionado dos explicaciones generadas por LIME en dicho punto, permitiendo evaluar si su comportamiento resulta coherente en ese contexto del juego.

Figura 5.21: Intersección a estudiar en análisis LIME

En la Gráfica 5.22 se pueden estudiar dos casos distintos, uno en el que Pac-Man busca la primera Power Pill en un momento temprano de la partida y otro en el que ha consumido una o varias Power Pills y aprovecha el vulnerabilidad de los fantasmas. A continuación, se describen los aspectos más relevantes, analizando las cinco características con mayor peso en las predicciones proporcionadas por LIME.

- **Gráfica correspondiente al movimiento LEFT**
  En el estado del juego analizado por LIME, la predicción realizada por el modelo es que Pac-Man debe moverse hacia la izquierda. Las características con mayor peso positivo son **totalTime <= 358** y **score <= 540**, esta gran influencia en la toma de la decisión se ha observado en otros análisis, y se atribuye a que ambas variables no fueron normalizadas, lo que provoca que su impacto sea mayor respecto al resto.

  La primera indica que la partida ha comenzado recientemente, por lo que es probable que Pac-Man aún no haya consumido ninguna Power Pill y se esté moviendo para encontrar una. La segunda característica, que refleja una puntuación baja, respalda esta teoría, sugiriendo que Pac-Man busca aumentar rápidamente su puntuación al consumir la primera Power Pill disponible moviéndose hacia la izquierda.

  Por otro lado, las características con mayor peso negativo son **ghost4EdibleTime <= 0**, **ghost2EdibleTime <= 0** y **ghost3EdibleTime <= 0**, las cuales indican que esos fantasmas no están comestibles en este momento. Resulta extraño que estas características tengan un peso tan negativo en contra de moverse a la izquierda, ya que precisamente sería lógico que Pac-Man buscase una Power Pill para aprovechar que los fantasmas actualmente no son comestibles.

  Esta aparente contradicción podría deberse a dos causas: una posible asociación de estas características con la posición cercana de los fantasmas respecto a Pac-Man, indicando un peligro inmediato en la dirección izquierda, o a la falta de normalización de las variables principales, que desplaza el peso de otras características. Sin embargo, dado que solo contamos con las cinco características de mayor peso, no es posible comprobar explícitamente esta relación con las posiciones de los fantasmas.

  En resumen, las características positivas presentan claramente sentido en relación con la decisión tomada, pero las negativas requerirían información adicional sobre la posición de los fantasmas y otras características para entender plenamente su impacto negativo tan marcado.

- **Gráfica correspondiente al movimiento RIGHT**
  El modelo realizó la predicción de que Pac-Man debe moverse hacia la derecha. En este análisis, todas las características tienen peso positivo. La característica con mayor peso es **540 <score <= 1370**, lo cual refleja una puntuación baja-media. Esto podría implicar que Pac-Man se encuentra en una etapa en la que ya ha obtenido algunos puntos y probablemente esté intentando moverse hacia la posición de otra Power Pill situada en la parte derecha para incrementar aún más su puntuación.

La tercera característica más importante es **358 <totalTime <= 759** lo que refuerza que Pac-Man aún se encuentra en una fase relativamente temprana de la partida. Las otras características relevantes indican claramente que los fantasmas 4, 3 y 2 se encuentran en estado comestible, por lo que se puede concluir que Pac-Man ha consumido recientemente una Power Pill y actualmente está persiguiendo fantasmas hacia la derecha para comerlos.

Una observación adicional importante es la diferencia en el tiempo de ser comestibles de los fantasmas: **ghost4EdibleTime >43**, **ghost2EdibleTime >79** y **ghost3EdibleTime >79**. El fantasma 4 tiene un tiempo restante de "43", mientras que los fantasmas 3 y 2 tienen tiempo restante de "79". Esto sugiere que Pac-Man podría haber consumido dos Power Pills en un intervalo de tiempo muy corto, lo cual es una decisión muy contraproducente ya que esto podría reducir la efectividad del uso de las Power Pills, ya que lo ideal sería consumir una, aprovechar plenamente su efecto, y posteriormente consumir la siguiente cuando el primer efecto haya terminado, maximizando así su utilidad.

Aun así, esta hipótesis también refuerza que el modelo interpreta correctamente la vulnerabilidad de los fantasmas como una oportunidad de ganar puntuación, priorizando movimientos ofensivos.

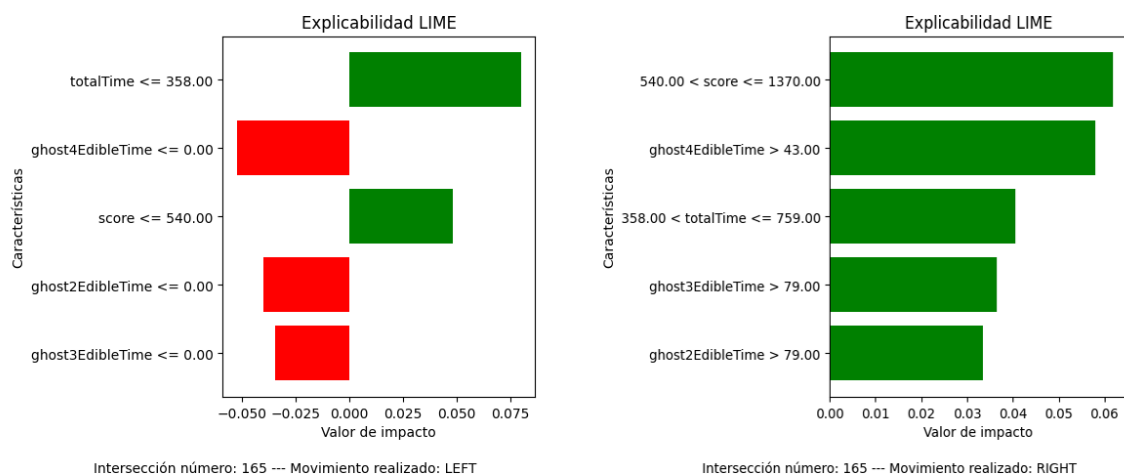

Figura 5.22: Importancia de características con LIME en Pytorch para el modelo en la intersección 165

5.2.2.2. Explicabilidad modelo Scikit-Learn

1. SHAP

- **Características más importantes**

  En el análisis se aprecia una influencia clara de cuatro características.

  La característica que tiene un mayor valor SHAP (0,35) es **Score**, esto indica que el modelo prioriza movimientos que maximizan la puntuación inmediata. Con algo menos de peso (0,30) encontramos **totalTime**, el tiempo transcurrido influye notablemente en la decisión del modelo, podríamos esperar un buen peso en este, sin embargo, sus altos valores podrían indicar que el modelo está teniendo en cuenta que tiene más peso del que tiene. Por otra parte tenemos a **timeOfLastGlobalReversal** (0,25) condicionando significativamente en la predicción, mostrando el mismo problema que totalTime, al presentar las mismas medidas de tiempo.

  Por último, encontramos a remainingPp (0,15), mostrando una clara importancia por las Power Pills restantes en el mapa.

- **Características de Posicionamiento**

  Los índices de nodos de los fantasmas (**ghostXNodeIndex**) presentan valores SHAP entre 0.08 y 0.12 Las distancias euclídeas y de camino a power pills (**euclideanDistanceToPp**, **pathDistanceToPp**) tienen un peso similar. Sería deseable obtener un peso mayor de estas características, compensando así el impacto entre características con el fin de asegurar una jugabilidad correcta.

- **Características de Estado de Fantasmas**

  En este apartado se encuentran con un peso inusualmente bajo características referidas al estado de los fantasmas como los tiempos comestibles de estos (**ghostXEdibleTime**), los tiempos de reaparición (**ghostXLairTime**) y el último movimiento de los fantasmas (**ghostXLastMove**).

- **Características de Bajo Impacto**

  Es destacable que ciertos eventos como fantasmas comidos (*ghostXEaten*) o Ms. Pac-Man comido (*pacmanWasEaten*) tienen una influencia mínima, sugiriendo que el modelo se centra más en el estado actual del juego que en eventos pasados inmediatos.

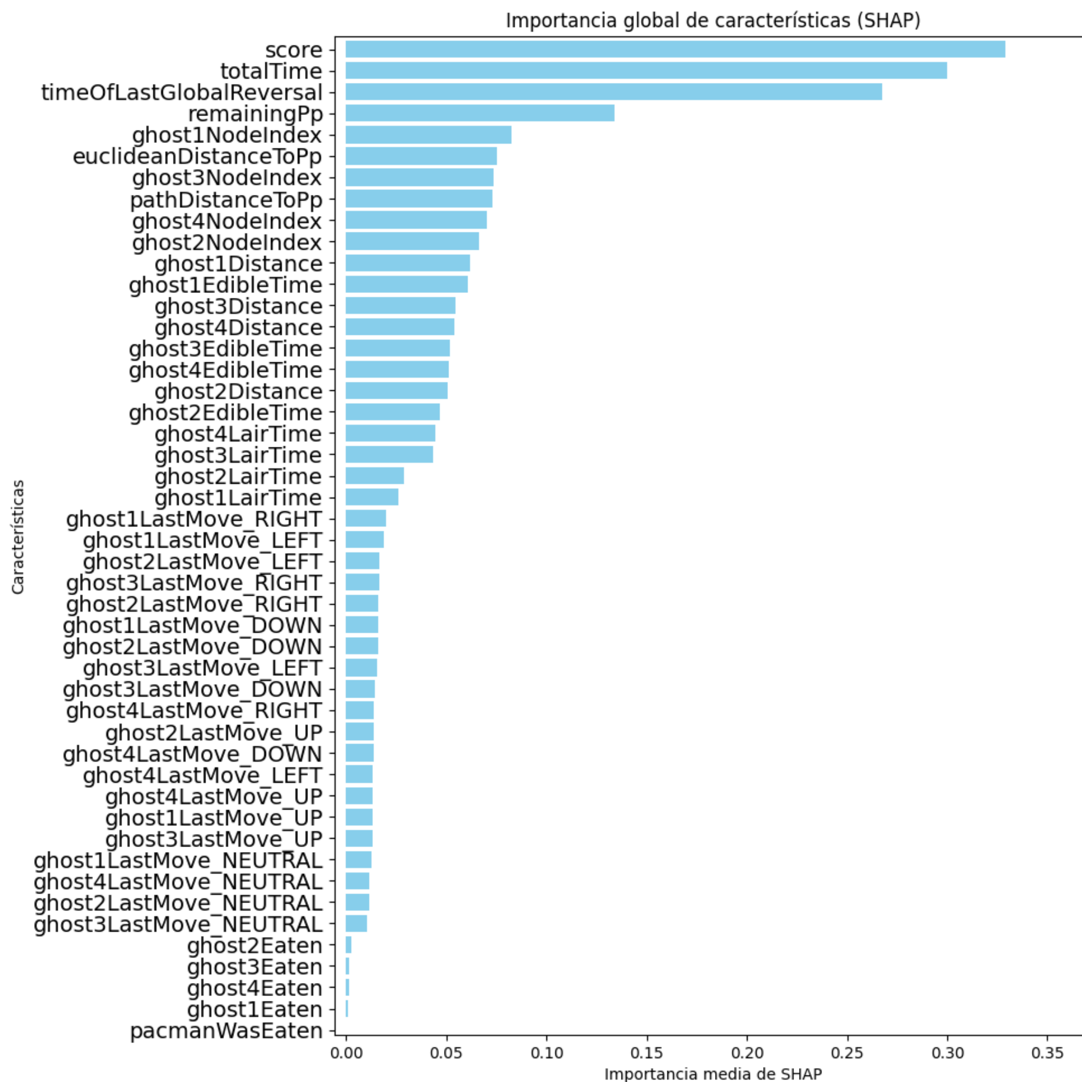

Figura 5.23: Importancia de características con SHAP en un modelo de Scikit-Learn.

- **Mapas de calor de las características más significativas**

  El mapa de calor de *totalTime* muestra una distribución relativamente estable en las intersecciones, con un patrón interesante: las intersecciones centrales presentan menor significancia (tonos más rojizos). Esto podría explicarse porque el agente transita menos por estas áreas centrales durante el juego, o posiblemente porque en esta zona central existen otras características que predominan más en la toma de decisiones.

  Para *timeOfLastGlobalReversal*, se observa una importancia ligeramente mayor en la parte inferior del mapa (tonos más amarillos). Este patrón es intrigante y podría interpretarse de dos maneras: o bien el modelo ha aprendido a valorar específicamente los eventos de reversión global cuando se encuentra en esta zona del laberinto (quizás por ser un área de mayor riesgo), o podría estar relacionado con una falta de normalización.

  El factor *score* presenta una distribución notablemente estable a través del

laberinto, con una coloración bastante uniforme. Este patrón confirma que el modelo efectivamente prioriza movimientos que maximizan la puntuación de manera consistente, independientemente de la ubicación específica en el mapa.

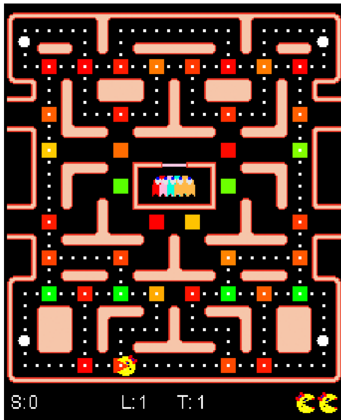

Figura 5.24: Mapa de calor del modelo Scikit-Learn para la característica timeOfLastGlobalReversal

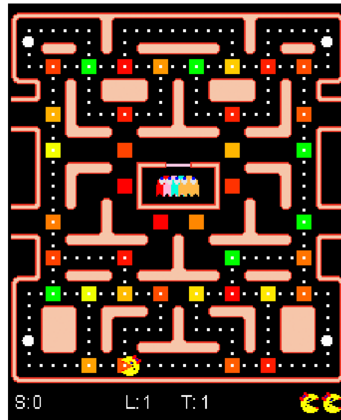

Figura 5.25: Mapa de calor del modelo Scikit-Learn para la característica totalTime

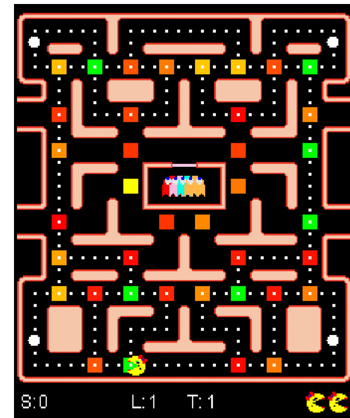

Figura 5.26: Mapa de calor del modelo Scikit-Learn para la característica score

Las tres características relacionadas con power pills (*RemainingPp*, *pathDistanceToPp* y *euclideanDistanceToPp*) muestran generalmente baja importancia global, aunque con un patrón común: las intersecciones más próximas a las ubicaciones de power pills son las que tienen tonalidades más amarillentas o anaranjadas. Esto sugiere que el modelo considera estas variables prioritariamente cuando se encuentra en posiciones donde puede evaluar rutas efectivas hacia estos elementos críticos del juego.

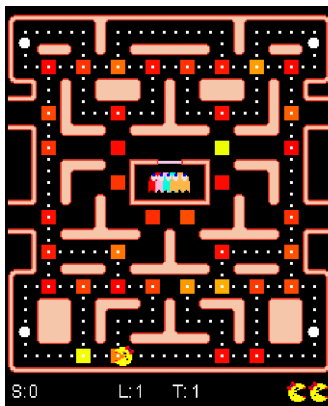

Figura 5.27: Mapa de calor del modelo Scikit-Learn para la característica remainingPp

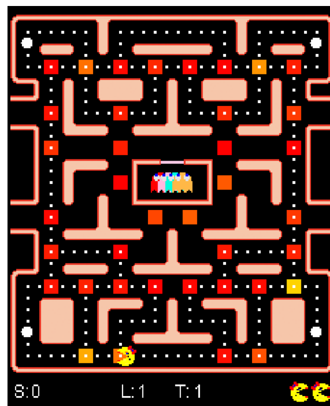

Figura 5.28: Mapa de calor del modelo Scikit-Learn para la característica pathDistanceToPp

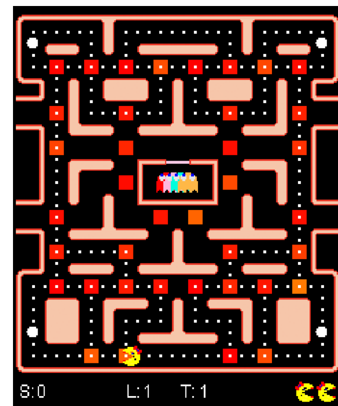

Figura 5.29: Mapa de calor del modelo Scikit-Learn para la característica euclideanDistanceToPp

Los mapas correspondientes a *ghost4NodeIndex*, *ghost3NodeIndex* y *ghost1Distance* presentan, en general, una importancia relativamente baja (predominio de rojo), pero con algunas intersecciones que muestran tonalidades menos intensas (más anaranjadas o amarillentas). Esta variación podría indicar que existen estados específicos donde la proximidad de los fantasmas adquiere mayor relevancia decisoria, posiblemente en situaciones de peligro inminente o en configuraciones espaciales particulares que requieren una evaluación más cuidadosa de la posición de los adversarios.

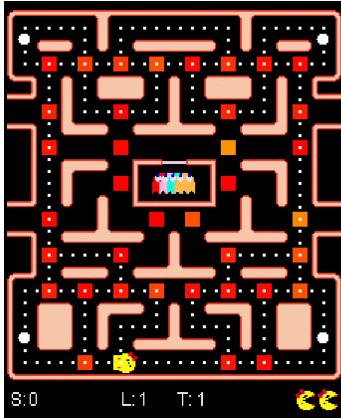

Figura 5.30: Mapa de calor del modelo Scikit-Learn para la característica ghost3NodeIndex

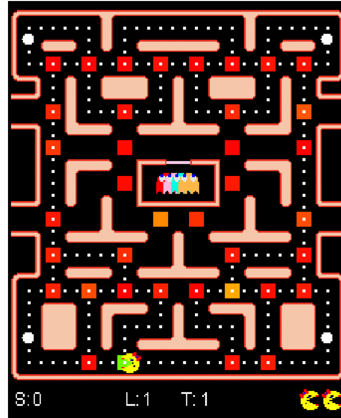

Figura 5.31: Mapa de calor del modelo Scikit-Learn para la característica ghost4NodeIndex

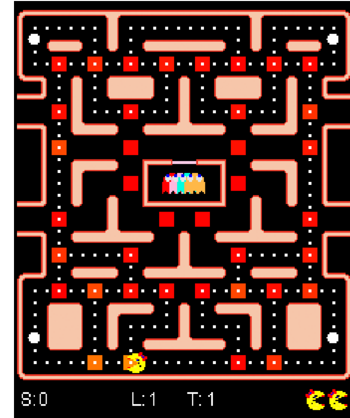

Figura 5.32: Mapa de calor del modelo Scikit-Learn para la característica ghost1Distance

2. Feature Importance

Al igual que en el gráfico de SHAP, **Score**, **totalTime** y **timeOfLastGlobalReversal** aparecen en las posiciones más altas del gráfico de barras, lo que significa que el modelo da mucho peso al tiempo que ha pasado desde el comienzo de la partida. También otorga mucho peso al score y al tiempo transcurrido desde la última inversión global, lo que refleja que intenta optimizar sus decisiones con el objetivo de maximizar la puntuación obtenida. Es posible que, tal y como se mencionó anteriormente, la desproporción que hay con el resto de características venga dada por los valores tan altos que toman estas variables en el dataset.

Las siguientes características indican que el Pac-Man está aprendiendo a percibir su entorno en una situación real de juego, pero mediante características con un menor impacto que las anteriores. Las distancias pathDistanceToPp y euclideanDistanceToPp son importantes en ambas métricas, lo que deja claro que el agente comienza a tomar decisiones en función de la proximidad de las 'pills'.

Finalmente se aprecia que en la zona baja de la gráfica aparecen características como **ghostEaten** o **pacmanWasEaten**, lo que deja ver que el modelo no da mucho peso a los eventos pasados, sino que se centra en el estado del juego actual.

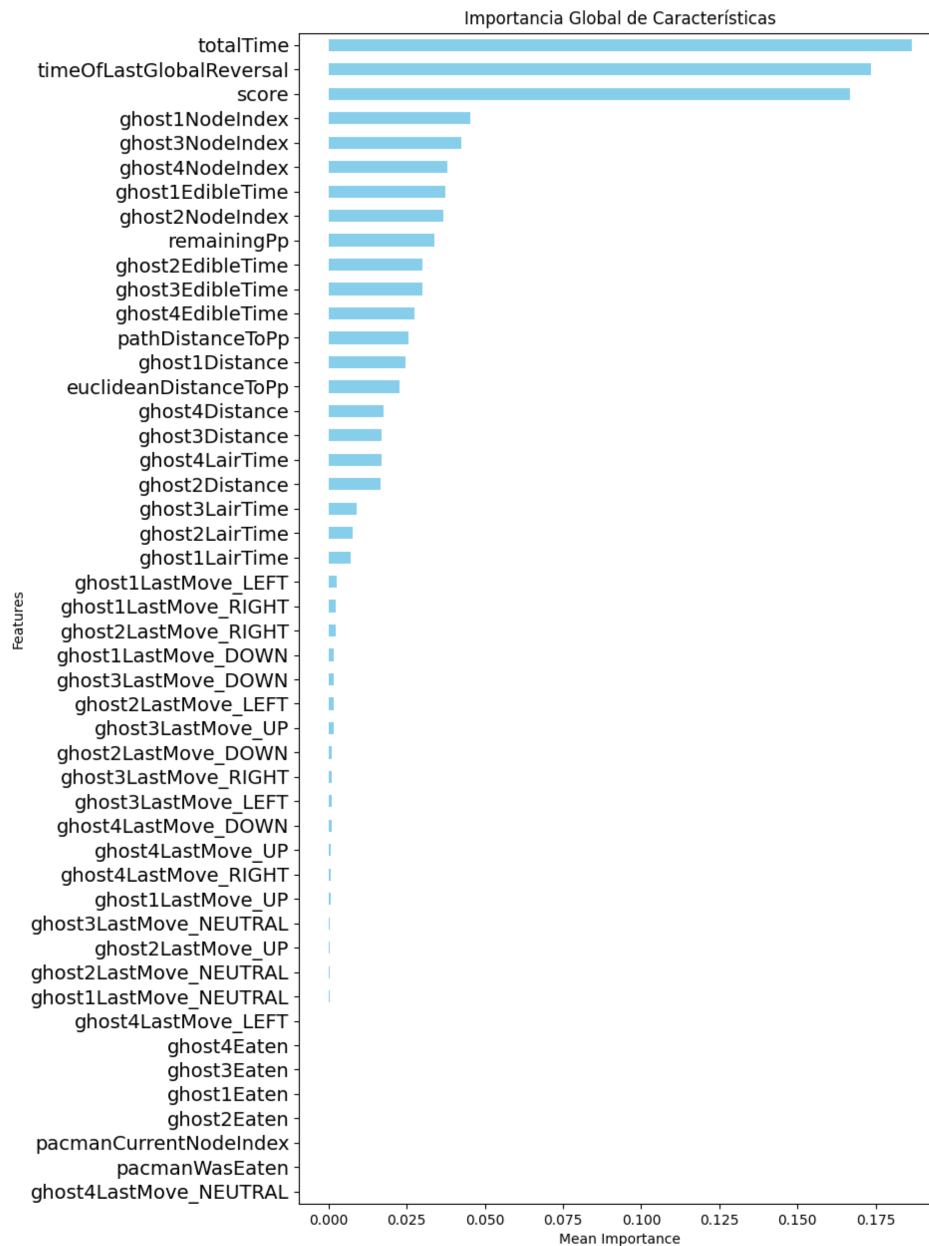

Figura 5.33: Importancia de características con Feature Importance en un modelo Sklearn.

3. LIME

Por coherencia respecto al análisis realizado previamente para el modelo implementado en PyTorch, se ha llevado a cabo un estudio similar para la parte de Scikit-Learn, utilizando la misma técnica de explicabilidad, y manteniendo fija la intersección del mapa previamente analizada (Imagen 5.21). Con ello, se pretende comparar cómo varía el comportamiento en función del framework utilizado para su entrenamiento. Esta comparación resulta de gran utilidad para identificar si las decisiones del modelo están condicionadas por el tipo de red utilizado, o si, por el contrario, existe

coherencia en la importancia de las características observadas en ese punto específico del mapa. Al igual que en el caso anterior, se analizarán las predicciones más representativas a partir de las explicaciones locales generadas por LIME, centrándonos en las cinco características con mayor peso en cada caso.

- **Gráfica correspondiente al movimiento RIGHT**
  En la gráfica de la izquierda, el modelo decide que Pac-Man debe moverse hacia la derecha. La característica con mayor peso positivo en esta decisión es **ghost4NodeIndex <1034**, lo que puede deberse a que el nodo 1034 está próximo a la intersección 165, desde la cual se hace la predicción, ya que la numeración de los nodos sigue un orden vertical descendente. Otra posible explicación es la falta de normalización de los datos, y por tanto valores altos como este tengan un impacto desproporcionado en la predicción.

  Por otro lado, la característica con mayor peso negativo es **ghost4EdibleTime <= 0**, lo que indica que el fantasma número 4 no es comestible, y por tanto se encuentra persiguiendo a Pac-Man. Esta característica negativa parece estar relacionada con la primera: aunque la posición del fantasma 4 contribuye a moverse hacia la derecha, el hecho de que esté activo y represente una amenaza directa reduce la confianza del modelo en ese movimiento a la derecha. Siguiendo ese mismo razanamiento, la penúltima característica más relevante es la **distancia al fantasma 4** (**ghost4Distance <= 19**), la cual también tiene peso positivo. Esto refuerza la idea de que el modelo asocia de forma conjunta la posición, distancia y estado del fantasma como factores influyentes. El valor 19 es bajo, lo cual indica que el fantasma está muy cerca de Pac-Man en ese momento. Finalmente, las otras dos características importantes con peso positivo son el tiempo de ser comestible de los fantasmas 1 y 2: **ghost1EdibleTime >82** y **ghost2EdibleTime >79**, lo que sugiere que el modelo predice que Pac-Man debe moverse hacia la derecha para aprovechar la oportunidad de comerse a estos dos fantasmas.

- **Gráfica correspondiente al movimiento UP**
  En la gráfica de la derecha, el modelo ha realizado la predicción de que Pac-Man debe moverse hacia arriba. De forma similar a la gráfica anterior, la característica con más peso positivo es **ghost4NodeIndex <1034**, lo que indica nuevamente la posición del fantasma número 4 como un factor relevante. Esta característica se ve acompañada de **ghost4EdibleTime <= 0**, lo cual indica que el fantasma no es comestible y, por tanto, se encuentra persiguiendo activamente a Pac-Man.

  La segunda característica con mayor peso positivo es **timeOfLastGlobalReversal <= 154**, la cual indica que ha pasado poco tiempo desde que Pac-Man consumió una Power Pill y finalizó su efecto. Esto tiene sentido dado que no se detecta ninguna característica con peso significativo que indique que los fantasmas estén en estado comestible, a diferencia de la gráfica anterior. Por último, otra característica relevante con peso positivo es la **posición del fantasma 1** (**ghost1NodeIndex >639**), junto con su estado no comestible, lo cual refuerza la idea de que el modelo si que tiene en cuenta la proximidad

y el estado de los fantasmas al tomar decisiones, especialmente en situaciones en las que hay una amenaza directa de ser comido por los fantasmas.

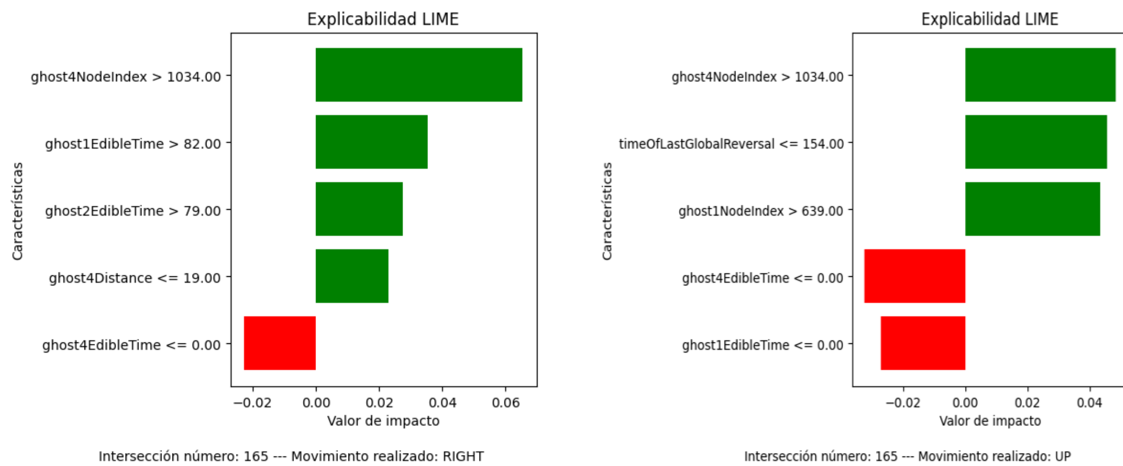

Figura 5.34: Importancia de características con LIME en Sklearn para el modelo en la intersección 165

5.2.3. Conclusiones

En líneas generales, **PyTorch muestra más capacidad de aprendizaje**, adaptándose mejor al patrón general de los datos, aunque esto implica una mayor variabilidad en sus predicciones. Este comportamiento puede ser el adecuado en aquellas situaciones donde se prefieran las predicciones que sean muy precisas en promedio, incluso si eso implica una mayor variabilidad.

Por otro lado, **Scikit-Learn da como resultado una mayor estabilidad y consistencia**, ya que da como resultado predicciones más centradas y menos variadas, lo cual puede resultar preferible cuando se hacen predicciones en situaciones donde se prefiere buscar la seguridad o tratar de ajustar al máximo la posibilidad de errores extremos, pues no hay que olvidar el riesgo en aquellas situaciones donde lo que se busca es realizar predicciones.

No obstante, **ninguno de los dos modelos es capaz de reproducir las puntuaciones del dataset original**, por lo que ambos tienen margen para mejoras. Ya sea mediante ajustes en la arquitectura, selección de hiperparámetros añadidos o buscando enriquecimiento del conjunto de entrenamiento o del conjunto test, sería necesario refinar ambos modelos para dar como resultado una mejora tanto en la exactitud como en la generalización.

Próximos pasos

Tras analizar la explicabilidad de los modelos, se observa que las variables con mayor impacto suelen ser aquellas que presentan valores naturalmente más altos, lo que sugiere un posible sesgo del modelo hacia la magnitud de ciertas características. Esto

apunta a la necesidad de aplicar una normalización o escalado adecuado del conjunto de datos, mediante técnicas como `StandardScaler` o `RobustScaler`, con el fin de equilibrar la influencia de las variables y facilitar una interpretación más justa por parte del modelo. Además, se recomienda introducir mecanismos de regularización más fuertes, especialmente en el caso de PyTorch, para evitar el sobreajuste que parece producirse al adaptarse demasiado al conjunto de entrenamiento. Ajustes como el uso de `dropout` o la implementación de `early stopping` podrían contribuir a una mejor generalización.

Como evolución del trabajo, se propone experimentar con un modelo TabNet, cuya arquitectura basada en atención secuencial sobre las características permite identificar de forma más precisa qué variables son relevantes en cada decisión sin perder la estructura tabular original. Su capacidad de aprendizaje profundo con interpretabilidad resulta especialmente valiosa ante las limitaciones detectadas en los modelos actuales. Finalmente, también se plantea la posibilidad de refinar el conjunto de datos, ya sea mediante la ingeniería de nuevas variables, la creación de combinaciones relevantes o incluso la incorporación de datos externos que enriquezcan el contexto del modelo.

5.3. Evaluación nº 3: Evaluación con la red neuronal TabNet

Para este tercer experimento se ha decidido emplear una nueva arquitectura de red neuronal llamada **TabNet**, diseñada para trabajar con datos tabulares y resolver problemas de sobreajuste y conjuntos de datos muy sesgados. El objetivo de este experimento es evaluar el correcto funcionamiento de nuestro conjunto de datos y observar el impacto de una arquitectura nueva en el comportamiento del Pac-Man.

Además de la incorporación de esta nueva arquitectura se han realizado algunos cambios apoyados por el análisis de la explicabilidad del Experimento 2. Se ha generado un nuevo conjunto de datos en el que se añaden solamente los estados del juego donde el movimiento realizado por el Pac-Man es válido para la intersección donde se encuentra. Con esto pretendemos conseguir que el modelo aprenda en base a situaciones de juego estrictamente posibles, consiguiendo así reducir el número de clases por intersección, favoreciendo a un aprendizaje más robusto y realista. Estos cambios en el dataset se apoyan de la incorporación de un archivo JSON por intersección con el índice de los movimientos posibles a realizar por intersección. A la hora de enviar el movimiento predicho por el socket, si se predice uno no válido, se corrige seleccionando el que mayor probabilidad tenga dentro de las clases permitidas.

Se ha realizado una mejora significativa en el escalado de los datos. Gracias al análisis de la explicabilidad del experimento anterior, se ha detectado que algunas características dominaban por encima de otras debido a sus altos valores numéricos. Para solucionar este problema, se ha añadido un *StardardScaler* por intersección, entrenado únicamente con los datos concretos de ese punto del mapa. Además, se

han incorporado características existentes en el propio entorno del juego, como otras creadas mediante funciones en Python. Un ejemplo es la variable booleana *isDanger*, que permite indicar si el Pac-Man está en peligro calculando la distancia mínima a los fantasmas.

Por último, se ha eliminado la posibilidad de predecir un movimiento NEUTRAL, el cual era devuelto en experimentos anteriores cuando se predecía un movimiento inválido en esa intersección. Gracias al análisis realizado, se ha confirmado que este movimiento no aportaba valor al comportamiento del Pac-Man, sino que introducía ruido en el proceso de aprendizaje.

Mientras modelos anteriores como el de Pytorch o Scikit-Learn con una o dos capas ocultas eran más propensos al sobreajuste (Pytorch) o a generar predicciones más conservadoras y estables (Scikit-Learn), TabNet busca un equilibrio más avanzado, dándole más importancia a unas características que a otras según la situación de juego en la que esté. Esta arquitectura posee una explicabilidad nativa, permitiendo entender mejor la importancia de cada característica en futuras fases del experimento.

5.3.1. Rendimiento

Para evaluar el rendimiento del modelo entrenado con **TabNet**, se realizaron un total de 600 partidas frente a las mismas implementaciones de fantasmas que en experimentos anteriores. Este enfoque permite comparar su desempeño con el de modelos previamente utilizados como PyTorch y Scikit-Learn, y analizar la mejora obtenida.

A continuación, se presentan las estadísticas extraídas de las puntuaciones obtenidas:

Estadística	**Valor**
Media	2479,68
Mediana	2390,00
Desviación típica	814,57
Máximo	6550,00
Mínimo	320,00
Rango	6230,00
Percentil 25 (Q1)	1990,00
Percentil 75 (Q3)	2890,00
Percentil 90	3509,00
Asimetría (Skewness)	0,95
Curtosis (Kurtosis)	2,70

Tabla 5.5: Estadísticas descriptivas del modelo de TabNet para el experimento 3.

Las estadísticas del modelo **TabNet** muestran un **rendimiento medio superior** al de los modelos anteriores. Su *media* (2479,68) y *mediana* (2390,00) superan a las obtenidas por PyTorch (2368,30 y 2250,00) y, con mayor diferencia, a las de Scikit-Learn (2101,43 y 2055,00). Esto sugiere que TabNet logra, en promedio, un mejor

desempeño a lo largo de las partidas.

Además, el modelo mantiene una **desviación típica considerablemente más baja** (814,57), lo que indica una mayor *consistencia* en los resultados: obtiene puntuaciones altas de forma más regular y con menor dispersión que PyTorch (1119,38), y similar a Scikit-Learn (878,35), pero con valores centrales más favorables. Esta regularidad queda reflejada también en la distribución representada en el Histograma (Figura 5.35), donde se observa una alta concentración de partidas con puntuaciones entre 2000 y 3000 puntos.

En cuanto a los *percentiles*, el modelo TabNet se posiciona en un nivel intermedio-alto:

- El **percentil 25** (1990) y el **percentil 75** (2890) indican que el 50 % de las partidas se sitúan dentro de un rango de puntuaciones bastante elevado y estable, con un rendimiento claramente competitivo.
- El **percentil 90** (3509) muestra que su *top 10 %* alcanza un rendimiento notable, aunque algo inferior al del modelo PyTorch (3839), pero mejor que Scikit-Learn (3089).

Respecto a la *asimetría* (0,95), TabNet presenta una distribución moderadamente sesgada hacia la derecha, con presencia de puntuaciones altas similares a las observadas en PyTorch (0,92). Este comportamiento se corrobora visualmente en el Histograma mencionado y en el **Boxplot** correspondiente (Figura 5.36), donde se aprecia una leve extensión de la cola derecha. La *curtosis* (2,70), por su parte, sugiere colas ligeramente pesadas, muy similares a las del modelo Scikit-Learn (2,75), lo que refleja cierta presencia de valores extremos, aunque en menor medida que PyTorch (1,27), cuya distribución es más plana.

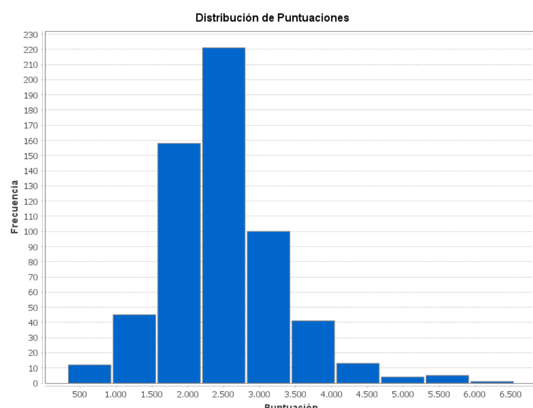

Figura 5.35: Histograma de TabNet (Exp.3)

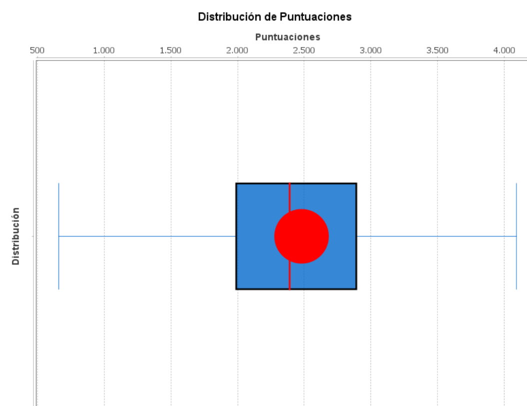

Figura 5.36: Boxplot de TabNet (Exp.3)

5.3.2. Explicabilidad con Feature Importance

Para obtener un mejor entendimiento de este nuevo modelo, se generan, al igual que en experimentos anteriores, una gráfica con la importancia media de cada característica y un conjunto de mapas de calor por característica. Esto permitirá observar qué características influyen más en las decisiones de nuestro nuevo modelo.

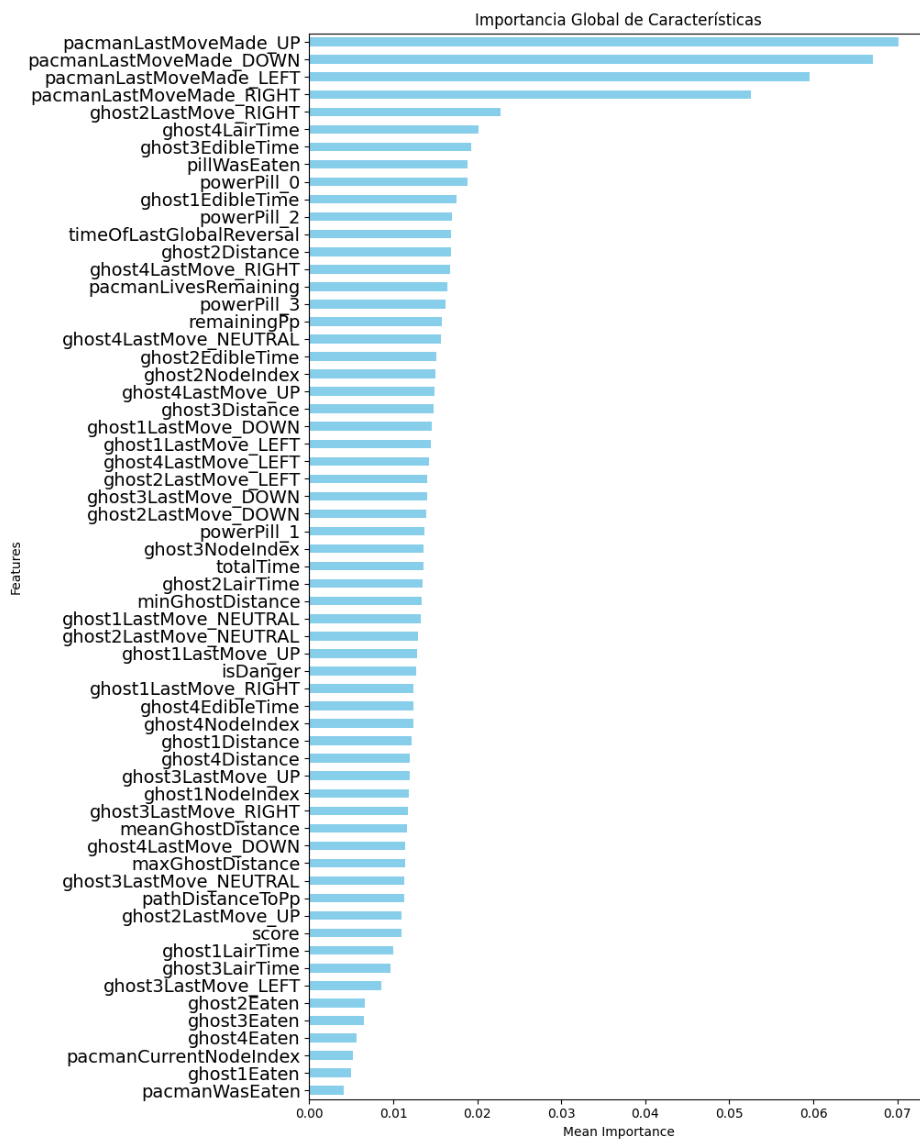

Figura 5.37: Importancia de características con Feature Importance en un modelo de TabNet.

El Gráfico de barras 5.37 muestra que las características relacionadas con el último movimiento del Pac-Man tienen un gran impacto en la predicción del movimiento. Esto se puede observar en los mapas de calor, donde estas variables se tienen en cuenta en prácticamente todas las intersecciones del mapa. A diferencia de experimentos anteriores, donde se priorizaba el tiempo total transcurrido o la puntuación

obtenida, tiene mucho más sentido que características relacionadas con el último movimiento del Pac-Man estén en la parte superior de la gráfica. Ya que estas ayudan al modelo a mantener una coherencia en sus movimientos y evitar acciones imposibles.

Un ejemplo visual de esto se puede observar en el Mapa de calor de pacmanLastMoveMade_LEFT 5.40. En intersecciones donde el color es totalmente rojo se deduce que el modelo ha aprendido que para llegar a esa intersección no es posible realizar un movimiento a la izquierda. De este modo, se concluye que el modelo ha aprendido qué movimientos va a poder realizar en cada intersección. En zonas centrales del mapa se observan colores más anaranjados o rojizos para estas características. Esto es debido a que para esas zonas hay una menor cantidad de datos durante el entrenamiento ya que Pac-Man tiende a moverse más por fuera que por dentro del mapa. Es decir, por lugares donde hay pills o powerPills y una menor cantidad de fantasmas.

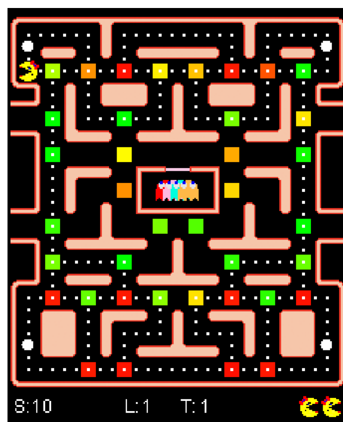

Figura 5.38: Mapa de calor del modelo TabNet para la característica pacmanLastMoveMade_UP

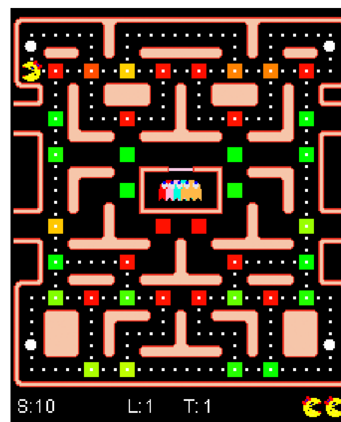

Figura 5.39: Mapa de calor del modelo TabNet para la característica pacmanLastMoveMade_DOWN

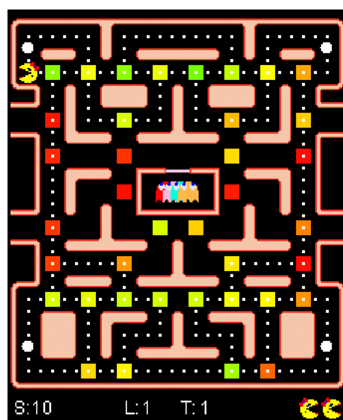

Figura 5.40: Mapa de calor del modelo TabNet para la característica pacmanLastMoveMade_LEFT

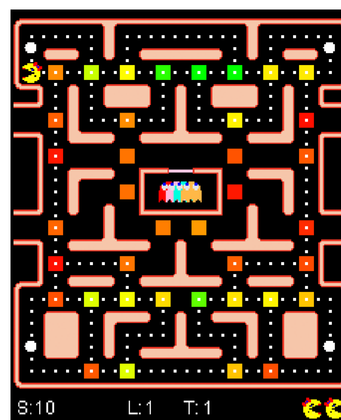

Figura 5.41: Mapa de calor del modelo TabNet para la característica pacmanLastMoveMade_RIGHT

Variables como *powerPill_0*, *powerPill_2* o *pillWasEaten* muestran una importancia más localizada. Es decir, son útiles en lugares concretos del tablero, ya que informan sobre la presencia o no de una PowerPill. Tal y como se refleja en los Mapas de calor 5.42 y 5.43, estas características tienen impacto en las intersecciones cercanas a la ubicación de las PowerPills, mientras que en el resto del tablero su importancia es mínima. Esto justifica que estén en una posición más baja de la gráfica a pesar de su importancia en esas zonas del tablero. Además, en el Mapa de calor de *pillWasEaten* 5.44 se observa que el modelo ha comprendido en qué zonas del tablero es más sencillo comer pills, lo que encaja perfectamente con lo comentado anteriormente sobre las zonas centrales del mapa.

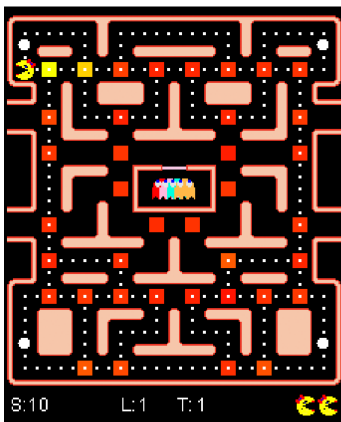

Figura 5.42: Mapa de calor del modelo TabNet para la característica powerPill_0

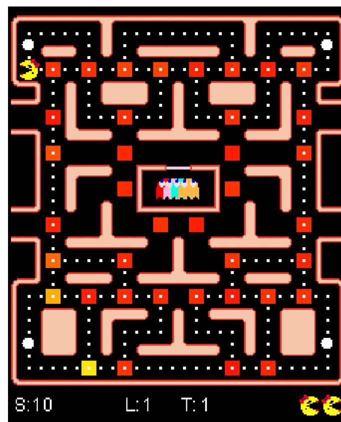

Figura 5.43: Mapa de calor del modelo TabNet para la característica powerPill_2

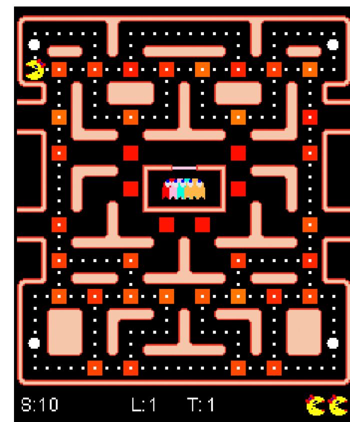

Figura 5.44: Mapa de calor del modelo TabNet para la característica pillWasEaten

Por otro lado, se observa que el modelo tiende a darle una menor importancia a las características relacionadas con los fantasmas. Esto da a entender que su objetivo es conseguir sobrevivir maximizando el número de Pills y PowerPills comidas. Todo esto se aprecia visualmente en la zona inferior de la gráfica, donde aparecen características como ghostXEaten. A pesar de ello, aparece un conjunto equilibrado de características relacionadas con los fantasmas en la mitad de la gráfica, como las de la Figura 5.47. En estos mapas de calor se aprecia que el modelo les da más importancia en pasillos cercanos al centro e intersecciones críticas, donde pueden resultar muy útiles para decidir si escapar o tratar de comer algún fantasma.

Finalmente, observando la diferencia significativa con respecto al resto de características, parece que el modelo ha sobreaprendido de su último movimiento realizado. Por lo que en un futuro experimento se tratará de reducir esta diferencia, con el fin de conseguir que se apoye más en variables relacionadas con el comportamiento de los fantasmas. De manera que el modelo no solo intente sobrevivir obteniendo el mayor número de Pills y PowerPills, sino que también trate de perseguir a los fantasmas cuando la situación lo permita, maximizando así la puntuación obtenida.

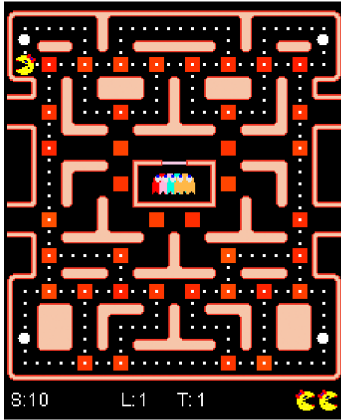

Figura 5.45: Mapa de calor del modelo TabNet para la característica ghost2Distance

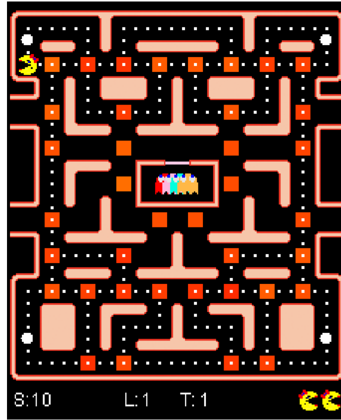

Figura 5.46: Mapa de calor del modelo TabNet para la característica ghost2LastMove_RIGHT

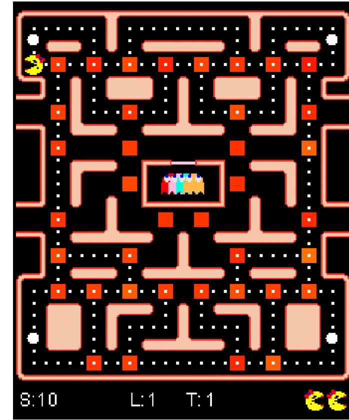

Figura 5.47: Mapa de calor del modelo TabNet para la característica ghost3EdibleTime

5.3.3. Conclusiones

En este tercer experimento se han mejorado de forma clara los fallos que veníamos arrastrando en los anteriores. Gracias a los cambios en el dataset y a un escalado más fino de los datos, hecho por intersección y con más variables significativas, el modelo ha ganado en estabilidad y precisión. TabNet, como nueva arquitectura, ha encajado muy bien en este escenario: es capaz de aprender de forma más inteligente, seleccionando qué variables importan según la situación, y evitando caer en los extremos del sobreajuste o en decisiones demasiado conservadoras.

Los resultados lo reflejan: tanto la media como la mediana de puntuaciones han subido, y además con una dispersión más baja. En otras palabras, el modelo no solo consigue mejores resultados, sino que lo hace de forma más regular, sin tantas partidas malas que arrastren la media hacia abajo, presentando un comportamiento más sólido y predecible, algo que hasta ahora costaba alcanzar.

Sin embargo, analizando de cerca lo que hace el modelo gracias a las herramientas de explicabilidad, se evidencia que el *Pac-Man* se comporta de forma bastante cauta. Suele moverse por los bordes y las esquinas del mapa, evitando las zonas centrales donde es más probable cruzarse con fantasmas. Esto tiene sentido desde el punto de vista de la supervivencia, pero también explica por qué la puntuación no se dispara: el modelo prefiere jugar sobre seguro antes que arriesgarse a cazar fantasmas cuando están comestibles. Es decir, prefiere no morir antes que sumar más puntos. Tiene lógica, pero también proporciona indicios para seguir mejorando su comportamiento. Además la introducción de la variable *powerpill_X* ha permitido al agente tener un conocimiento más concreto de las esquinas, evitando así los bucles que observabamos en experimentos anteriores.

También se han hecho ajustes que han ayudado a limpiar el entrenamiento: por ejemplo, se ha eliminado la opción de predecir el movimiento *NEUTRAL*, que solo generaba ruido y confundía al modelo. Ahora, si el modelo predice un movimiento

que no es válido, automáticamente se corrige en tiempo real y se elige el más probable entre los movimientos permitidos. Así se asegura que todas las decisiones que toma tengan sentido en el contexto del juego.

En resumen, TabNet ha traído mejoras claras: mejores resultados, más coherencia en las decisiones, y una base sólida para seguir trabajando. Aun así, el modelo sigue siendo demasiado "prudente", y eso limita su capacidad de alcanzar puntuaciones más altas.

Próximos pasos

Después de los avances logrados en este tercer experimento, el siguiente objetivo es seguir afinando el comportamiento del modelo, no solo para que sobreviva más tiempo, sino también para que sepa cuándo arriesgar para maximizar la puntuación. Para ello, se plantea reforzar la calidad de las variables utilizadas, sin necesidad de eliminar ninguna de las ya introducidas, sino mejorando y expandiendo su capacidad informativa.

Un primer cambio será tratar ciertos atributos como variables categóricas en lugar de separarlos por one-hot encoding. En concreto, se aplicará a los movimientos anteriores tanto de Pac-Man como de los fantasmas. Esta decisión tiene dos ventajas clave: por un lado, TabNet maneja de forma natural las variables categóricas, permitiendo al modelo aprender relaciones sin necesidad de aumentar la dimensionalidad; por otro, al liberar espacio en el dataset, lo que abre la puerta a introducir nuevas características sin saturar el modelo.

Entre las nuevas variables que se están diseñando, una de ellas define un "modo de comportamiento" dinámico para Pac-Man, basado en la proximidad y el estado (comestible o no) de los fantasmas. Según el equilibrio entre amenazas y oportunidades cercanas, el modelo podría entrar en un modo huida, modo ataque, o mantener un modo normal. Esta capa de interpretación táctica permitirá al agente tomar decisiones más alineadas con el contexto real de la partida, y evitar conductas excesivamente conservadoras como las observadas en el experimento actual.

Además, con estas mejoras se espera que el modelo empiece a apoyarse más en señales de alto nivel (como el modo de juego) y no dependa tanto de la memoria inmediata del último movimiento. La idea es que desarrolle una especie de "intuición" situacional que le permita decidir cuándo vale la pena arriesgarse y cuándo no, elevando así su rendimiento no solo en supervivencia, sino también en puntuación total.

5.4. Evaluación nº 4: Enriquecimiento del dataset y optimización de variables para TabNet

En este cuarto experimento se continúa el trabajo iniciado en el Experimento 3, con el propósito de perfeccionar tanto la estructura del conjunto de datos como la forma en la que este se comunica con el modelo TabNet. Se han aplicado varias transformaciones significativas con el fin de maximizar las fortalezas de esta arquitectura, especialmente en escenarios con entradas densas y contexto cambiante difíciles de interpretar.

Una de las principales modificaciones ha sido el tratamiento de ciertos atributos como variables categóricas, abandonando el clásico one-hot encoding. Esta decisión se ha aplicado a los movimientos previos realizados tanto por Pac-Man como por los fantasmas. La ventaja es doble: por un lado, TabNet gestiona estas variables de forma nativa, permitiendo que el modelo capte relaciones internas sin necesidad de duplicar columnas; por otro lado, al reducir la dimensionalidad del dataset, se libera espacio que puede aprovecharse para introducir nuevas características sin poner en riesgo la estabilidad del aprendizaje.

En esa misma línea, se ha enriquecido el dataset con información más contextual (Figura 5.6), diseñada para reflejar el entorno de juego con mayor fidelidad. Se han añadido características referidas a los fantasmas permitiendo al modelo comprenda mejor los riesgos y ventajas situacionales. En experimentos anteriores se detectó que el Pac-Man no mostraba interés por estas píldoras, lo que le impedía avanzar al siguiente nivel. Es por ello que se ha añadido un bloque de columnas que reflejan el estado de las **píldoras normales del mapa** (hasta 220 solo para el primer laberinto). TabNet, gracias a su mecanismo de atención, puede manejar sin problema esta gran cantidad de variables, identificando cuáles son útiles en cada decisión sin sufrir penalizaciones por la dimensionalidad.

Además, se ha realizado una limpieza importante del dataset. Se han eliminado aquellas variables que actuaban como consecuencias del movimiento (por ejemplo, *pacManWasEaten*, *ghostEaten*, *pillEaten*, *PacmanCurrentNodeIndex*, *timeOfLastGlobalReversal*, etc.), las cuales generaban sesgos al dar al modelo información que realmente solo está disponible después de tomar una decisión. En su lugar, se ha implementado un enfoque de aprendizaje más cercano al refuerzo: se ha creado una nueva columna llamada ***reward*** (Figura 5.7), que no se usa como característica de entrada, sino como una variable externa que otorga peso a cada muestra durante el entrenamiento. Esta *reward* se calcula entre intersecciones consecutivas, midiendo el impacto de cada acción, y permite que el modelo aprenda a priorizar situaciones relevantes, ya sea por un resultado positivo o por una consecuencia negativa importante. Así, el modelo da más importancia a momentos críticos del juego sin necesidad de depender de variables que inducen fuga de información.

Característica	Descripción
numEdibleGhosts	Número de fantasmas comestibles.
ghostNearby	Número de fantasmas a una distancia específica.
ghostDangerousNearby	Número de fantasmas peligrosos a una distancia específica.
ghostEdibleNearby	Número de fantasmas comestibles a una distancia específica.
stdGhostDistance	Indica cuán dispersas están las posiciones de los fantasmas con respecto a Pac-Man.
avgGhostDistance	Distancia media entre Pac-Man y los fantasmas.
minGhostDistance	Distancia mínima entre Pac-Man y un fantasma.
maxGhostDistance	Distancia máxima entre Pac-Man y un fantasma.
edibleClosestGhostDistance	Distancia al fantasma comestible más cercano.
closestGhostDistance	Distancia al fantasma más cercano, sin importar su estado.
numActiveGhosts	Número de fantasmas fuera de la cárcel.
pacmanMode	Estado de Pac-Man: atacante, defensivo o normal.
pill_X	Presencia de la pill X.

Tabla 5.6: Descripción de las características añadidas para este experimento

Evento	Recompensa	Descripción
Pill	12	Cuando Pac-Man se come una pill.
Power Pill	0	Cuando Pac-Man consume una power pill.
Fantasma	40	Cuando Pac-Man se come un fantasma en estado comestible.
Paso	-5	Penalización por cada tick de juego ejecutado. Se utiliza para conseguir que el Pac-Man tome decisiones eficientes, evitando así la vagueza.
Victoria	50	Cuando Pac-Man ha consumido todas las píldoras del laberinto.
Derrota	-375	Cuando Pac-Man ha sido comido por un fantasma no vulnerable.

Tabla 5.7: Descripción de las recompensas asignadas a cada evento.

Por último, cabe señalar que el dataset se ha generado mediante partidas jugadas con la mejor versión de Pac-Man contra los fantasmas con mayor puntuación, siguiendo las directrices marcadas por la asignatura de ICI. Esto garantiza que los datos capturan situaciones complejas, desafiantes y realistas, ofreciendo una base sólida para entrenar un modelo que pueda tomar decisiones competitivas.

5.4.1. Rendimiento

Para evaluar el rendimiento del modelo entrenado con **TabNet**, se realizaron un total de 400 partidas frente a las mismas implementaciones de fantasmas que se emplearon para generar el conjunto de datos. Esta configuración garantiza coherencia entre el entrenamiento y la evaluación.
Cabe destacar que en este experimento se introdujo una **limitación artificial en las puntuaciones**: dado que el modelo es ahora capaz de completar el primer mapa y pasar al segundo, se decidió finalizar cada partida en caso de que Pac-Man consiguiera pasar al segundo mapa. Esto implica que las puntuaciones podrían haber sido más altas, pero se decidió finalizar las partidas en ese punto para mantener la evaluación centrada en el primer mapa, como en los experimentos previos.

Estadística	**Valor**
Media	2630,08
Mediana	2510,00
Desviación típica	877,15
Máximo	5880,00
Mínimo	590,00
Rango	5290,00
Percentil 25 (Q1)	2110,00
Percentil 75 (Q3)	3117,50
Percentil 90	3669,00
Asimetría (Skewness)	0,74
Curtosis (Kurtosis)	1,56

Tabla 5.8: Estadísticas descriptivas del modelo de TabNet para el experimento 4.

Las estadísticas del modelo **TabNet** en este experimento muestran un **ligero aumento en la media y mediana** respecto al experimento anterior. La *media* (2630,08) y la *mediana* (2510,00) superan a las obtenidas en el experimento 3 (2479,68 y 2390,00 respectivamente), lo que sugiere una leve mejora general en el rendimiento, a pesar de la limitación en la puntuación al pasar de mapa.

La **desviación típica** (877,15) es algo mayor que la del experimento 3 (814,57), lo cual indica una dispersión algo más amplia, pero dentro de un margen razonable. El Histograma (Figura 5.48) refleja una alta concentración de partidas en el rango de 2000 a 3000 puntos, lo que respalda esta observación.

En cuanto a los *percentiles*, el modelo mantiene un rendimiento competitivo:

- El **percentil 25** (2110) y el **percentil 75** (3117,50) muestran que la mitad de las partidas se sitúan en un rango sólido, incluso algo más alto que el del experimento anterior.
- El **percentil 90** (3669) refleja un *top 10 %* con puntuaciones destacadas, ligeramente superiores también a las del experimento 3 (3509).

Respecto a la *asimetría* (0,74), la distribución sigue sesgada hacia la derecha, aunque de forma menos pronunciada que en el experimento anterior (0,95), lo que puede

interpretarse como una mejora en la regularidad de las partidas altas. En el **Boxplot** (Figura 5.49) se observa una forma más centrada, con una cola derecha menos alargada. La *curtosis* (1,56), por su parte, es también menor que en el experimento 3 (2,70), lo que sugiere colas menos pesadas y menor presencia de valores extremos.

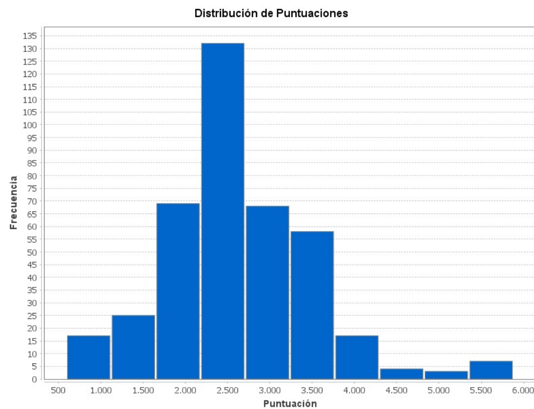

Figura 5.48: Histograma de TabNet (Exp.4)

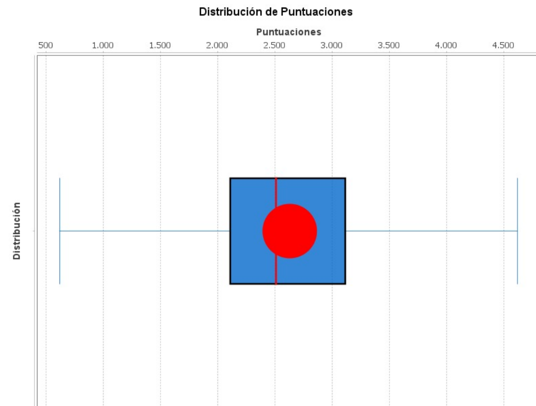

Figura 5.49: Boxplot de TabNet (Exp.4)

5.4.2. Explicabilidad con Feature Importance

La explicabilidad de este experimento evidencia que se ha conseguido repartir el impacto entre características de forma que no haya una gran diferencia entre unas y otras, como sucedía en experimentos pasados. Al igual que en el anterior experimento, se puede observar que el modelo sigue apoyándose del **último movimiento realizado** ya que permite conocer cuáles son los movimientos posibles a realizar. Su mapa de calor permite observar cuáles son los lugares del tablero donde más tiempo tiende a pasar Pac-Man. Como se apreció en el experimento anterior, trata de buscar las esquinas, donde tiende a haber un mayor número de pills y power-Pills, consiguiendo así más oportunidades para comer fantasmas. La transformación de **pacmanLastMoveMade** y **ghostXLastMove** a una sola columna ha permitido al modelo reducir la dimensionalidad y repartir de manera más equilibrada el impacto de estas características.

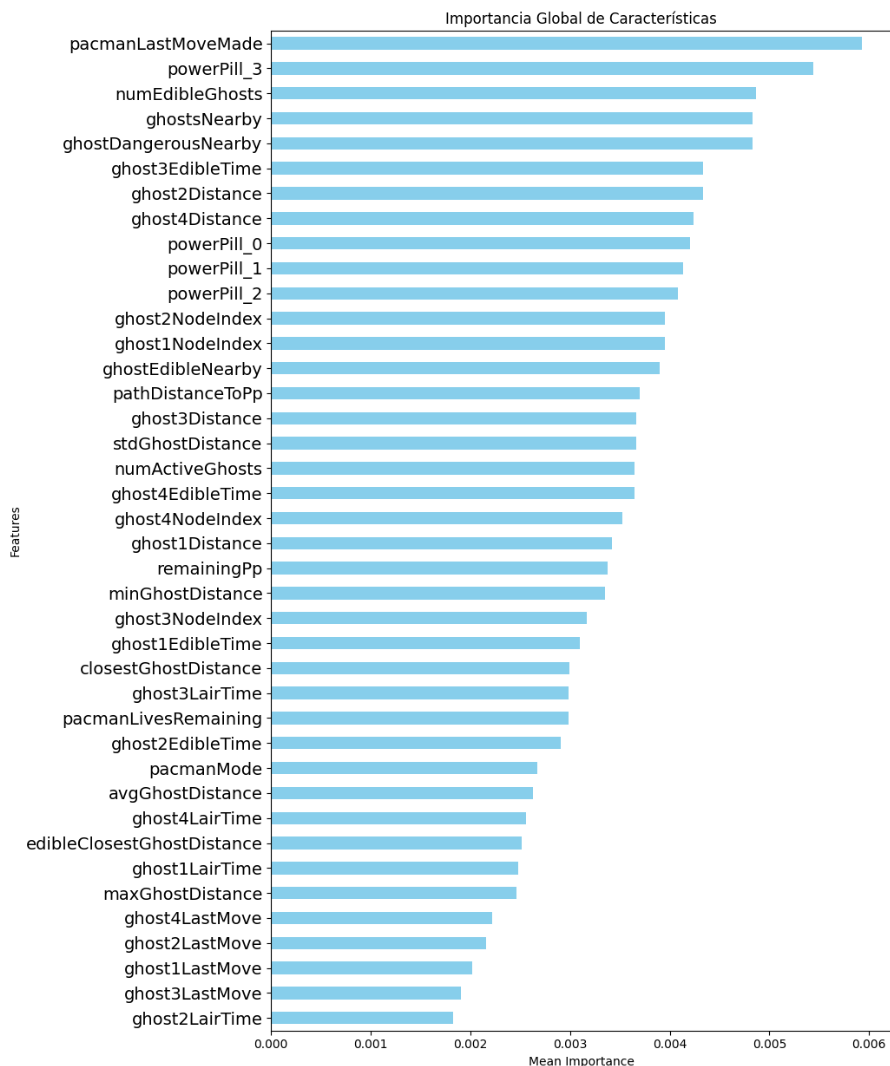

Figura 5.50: Importancia de características con Feature Importance en un modelo de TabNet.

La incorporación de 220 características (**pill_X**) con información de qué pills están activas en cada estado del juego ha tenido un gran impacto significativo, permitiendo al modelo crear caminos más estratégicos a lo largo del mapa, evitando rutas sin pills. Esto ha producido como resultado un comportamiento más inteligente, consiguiendo así en algunas ocasiones avanzar de mapa (5.52). Debido a su gran cantidad y su baja importancia individual, se eliminaron de la gráfica de importancia global. Sin embargo, se ha decidido sumar todos los valores de impacto en una sola característica, demostrando así su importancia general en todas las intersecciones del mapa.

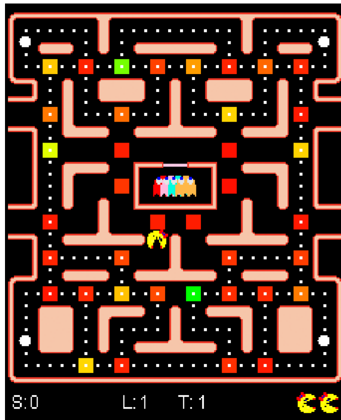

Figura 5.51: Mapa de calor del modelo TabNet para la característica pacmanlastMoveMade

Figura 5.52: Mapa generado tras superar el primer laberinto en el Experimento 4.

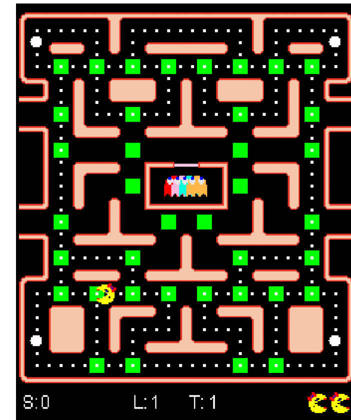

Figura 5.53: Mapa de calor del modelo TabNet para la característica pillsPresence

La poca atención en el estado de los fantasmas como no huir o atacarlos cuando corresponde era otro problema que se quería solucionar en este experimento. En la explicabilidad queda reflejado cómo el modelo ha conseguido relacionar mejor las características referidas a los fantasmas (**ghostXDistance** y **ghostXNodeIndex**), en parte gracias a características que refuerzan su comportamiento como **pacmanMode, numEdibleGhosts, ghostsNearby y ghostsDangerousNearby,** estos tres últimos encabezando la gráfica. De esta manera, como indican sus mapas de calor, se consigue un comportamiento estable por parte del modelo a la hora de tomar decisiones situacionales, las cuales no se encontraban de igual manera en experimentos anteriores, provocando un comportamiento poco adecuado a la información del estado de los fantasmas.

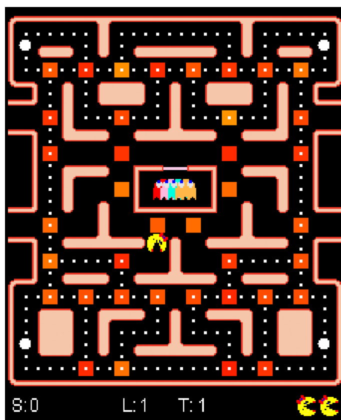

Figura 5.54: Mapa de calor del modelo TabNet para la característica numEdibleGhosts

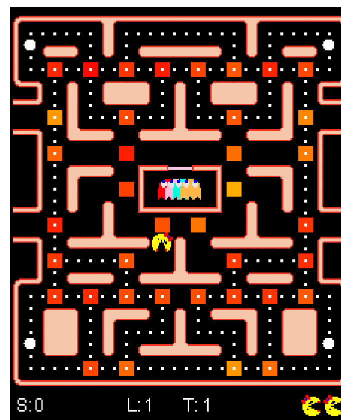

Figura 5.55: Mapa de calor del modelo TabNet para la característica ghostsNearby

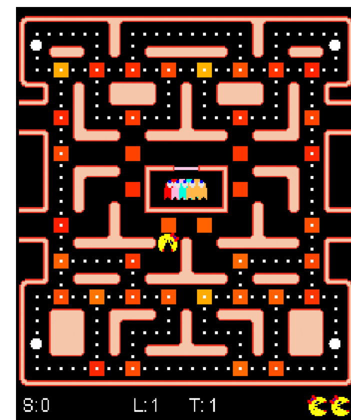

Figura 5.56: Mapa de calor del modelo TabNet para la característica ghostDangerousNearby

Los Mapas de calor como 5.57 muestran una diferencia con respecto al experimento anterior. Se puede observar como no solo se le da importancia a una powerPill cuando está cerca de ella, mostrando así que el modelo comprende de mejor manera el contexto general del mapa.

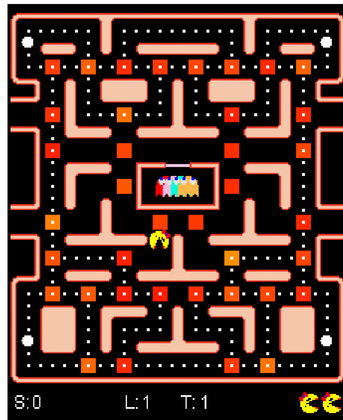

Figura 5.57: Mapa de calor del modelo TabNet para la característica powerPill_0

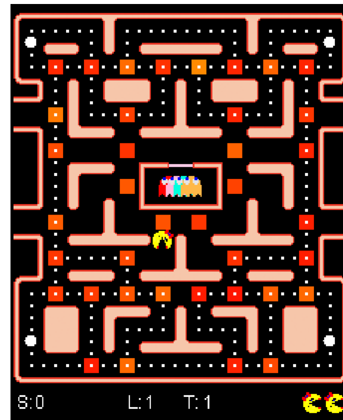

Figura 5.58: Mapa de calor del modelo TabNet para la característica powerPill_1

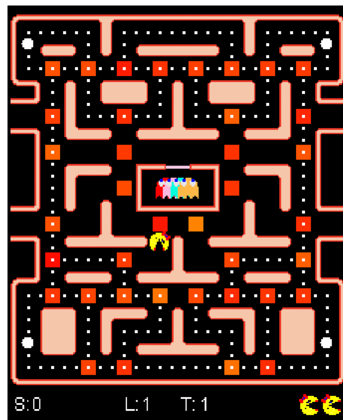

Figura 5.59: Mapa de calor del modelo TabNet para la característica powerPill_2

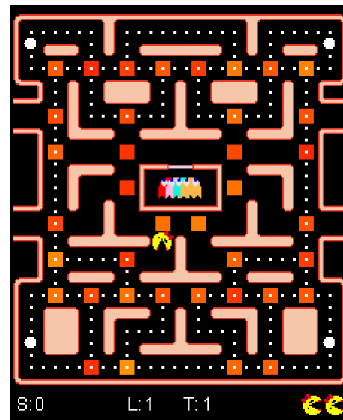

Figura 5.60: Mapa de calor del modelo TabNet para la característica powerPill_3

De esta manera, los resultados de la explicabilidad demuestran que estos cambios han permitido que las decisiones que toma el Pac-Man estén basadas en un equilibrio entre supervivencia, recolección de pills y maximización de puntos mediante la persecución de fantasmas comestibles.

5.4.3. Conclusiones

En este cuarto experimento se ha evidenciado una evolución clara en el comportamiento del modelo. Los cambios introducidos, como la reorganización de variables y la incorporación de nuevas características sobre el estado del juego, han permitido que Pac-Man tome decisiones más equilibradas, combinando mejor la recolección de pills, la gestión del peligro y la obtención de puntos a través de los fantasmas.

A diferencia de los experimentos anteriores, ya no se observa una dependencia tan marcada en una sola característica, como ocurría con el último movimiento realizado en experimentos anteriores. La importancia de las variables se ha repartido de forma más uniforme, lo que indica que el modelo ha adquirido una visión más completa del entorno. Además, ha aprovechado de forma más eficaz la información sobre las pills activas, lo que le ha permitido seguir rutas más efectivas y evitar zonas sin recompensa.

También se ha mejorado la capacidad del modelo para reaccionar ante los fantasmas. En experimentos anteriores apenas les prestaba atención; sin embargo, en esta ocasión, gracias a las nuevas variables que reflejan su cercanía y estado, ha sido capaz de decidir mejor cuándo huir o cuándo arriesgarse para obtener más puntos. Esto ha dado lugar a un comportamiento más inteligente y adaptado a cada situación del juego.

En cuanto al rendimiento, las estadísticas han mostrado una ligera mejora respecto al experimento anterior, con una media y una mediana más elevadas. Aunque las partidas se han interrumpido de forma manual al alcanzar el segundo mapa, limitando así la puntuación máxima, los resultados han sido positivos. El modelo ha mantenido una buena estabilidad, con menos partidas extremas y una distribución más regular.

En definitiva, este experimento ha supuesto un paso importante en la evolución del modelo. Ha conseguido combinar de forma más equilibrada los tres objetivos fundamentales del juego: sobrevivir, recoger pills y aprovechar las oportunidades para cazar fantasmas comestibles. Su comportamiento se ha vuelto más completo, menos predecible y con mayor capacidad de adaptación a lo que ocurre durante la partida.

Capítulo 6

Conclusiones y Trabajo Futuro

El objetivo principal de este trabajo ha sido diseñar, entrenar y evaluar distintos modelos de inteligencia artificial aplicados al juego de Ms. Pac-Man vs Ghosts. Se ha investigado sobre diferentes arquitecturas y estrategias con el propósito de encontrar un modelo capaz de decidir de manera eficaz en situaciones dinámicas e inciertas.

Con el fin de conseguir el objetivo principal hemos tenido que completar los diferentes subobjetivos que planteamos al inicio, en primer lugar, estudiamos el entorno del juego, adaptando el workspace de la asignatura ICI, identificando variables relevantes y diseñando nuevas características que no formaban parte del estado original. A continuación, implementamos el sistema de recogida de datos importantes en CSV y adaptamos el motor del juego para llevar a cabo partidas automáticas. La tercera meta fue conectar en tiempo real el juego y las redes neuronales mediante sockets, después procesamos los datos y entrenamos modelos con PyTorch, Scikit-Learn y posteriormente TabNet. Por último, llevamos a cabo el análisis de la explicabilidad de cada modelo con SHAP, LIME y Feature Importance, logrando así completar los pasos estipulados para llegar a nuestro objetivo.

En la siguiente tabla se puede observar la evolución del modelo en los distintos experimentos, destacando la diferencia de puntuación media de más de 1000 puntos entre el primer y último experimento. Un detalle a recordar es que este cuarto experimento hemos tenido que finalizar la partida en los momentos en los que se avanzaba de mapa, limitando así el número de partidas con puntuaciones elevadas.

Tabla 6.1: Comparativa de los resultados de los experimentos.

Experimento	Media	Desviacion Tipica	Q25	Q75
Pytorch Exp 1	1514.47	652.58	20	4380
Scikit-Learn Exp 1	1685.22	685.46	270	4440
Pytorch Exp 2	2368.30	1119.38	330	7470
Scikit-Learn Exp 2	2101.43	878.35	350	7490
TabNet Exp 3	2479.68	814.57	320	6550
TabNet Exp 4	2630.08	877.15	590	5880

Esta mejora progresiva en los resultados muestra con claridad el impacto positivo que han tenido los cambios realizados en cada fase. En especial, se puede observar la diferencia entre arquitecturas clásicas como Scikit-Learn y las más modernas como Tabnet. Su uso ha permitido al agente entender mejor el entorno, ya que permite distinguir cuáles son las características más importantes en cada momento de la partida y darles más peso a la hora de tomar la decisión. Esto se ha traducido en un comportamiento más inteligente donde el modelo empezó a tomar decisiones en función de situaciones específicas del momento, evitando la repetición de patrones y consiguiendo una percepción más general del entorno.

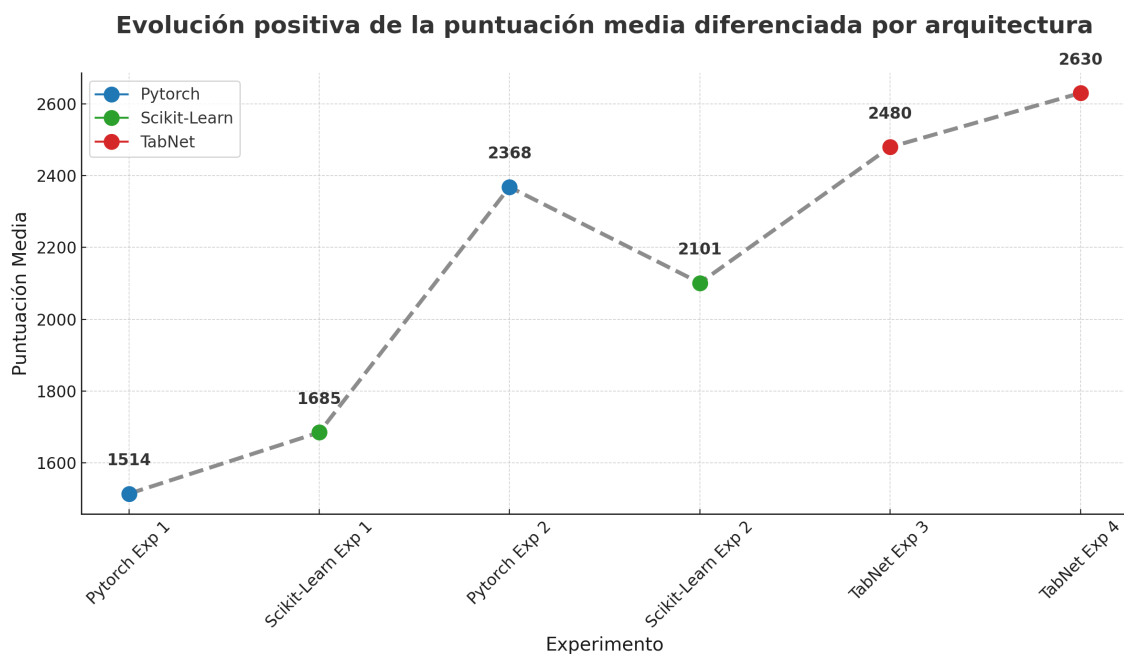

Figura 6.1: Evolución de la puntuación media diferenciada por arquitectura.

A lo largo del proyecto, hemos comprobado de forma muy clara que la arquitectura elegida tiene un peso enorme en el rendimiento del agente, sobre todo cuando se enfrenta a entornos dinámicos y cambiantes. No es suficiente tener un modelo que funcione en términos generales; es fundamental que sepa adaptarse a cambios constantes, identificar señales clave en cada momento y tomar decisiones que no sigan patrones fijos. Cada experimento nos ha ido enseñando algo nuevo: desde las limitaciones de los modelos más simples hasta el potencial de aquellos que integran mecanismos más avanzados de atención y procesamiento contextual, como es TabNet.

Además, algo que ha marcado la diferencia entre experimentos ha sido el trabajo sobre el dataset. Al principio partíamos de variables básicas, como la dirección anterior de Pac-Man o la posición de los fantasmas, pero con el tiempo vimos que esto no era suficiente para capturar la complejidad real de la partida. Empezamos a diseñar nuevas features derivadas, combinando variables, calculando distancias, evaluando estados de peligro o ventaja, y midiendo patrones de comportamiento. Esta fase de feature engineering ha sido clave: muchas de estas variables nuevas han aportado mucho más valor que las originales, permitiendo a los modelos detectar contextos

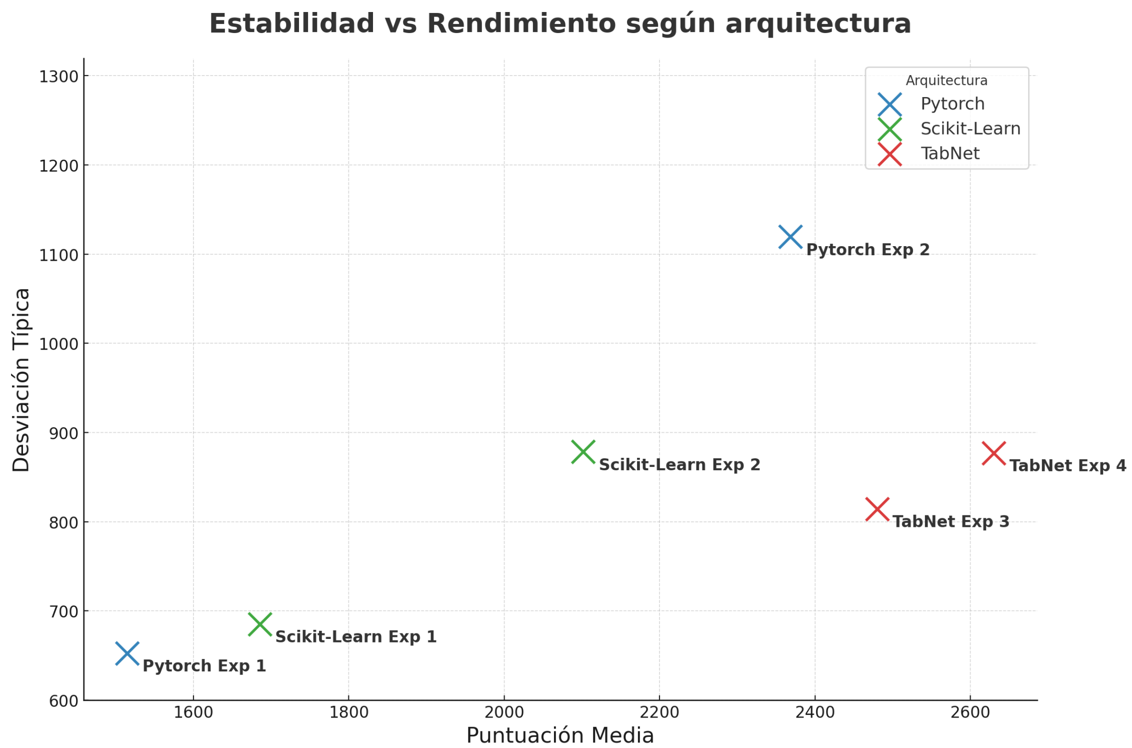

Figura 6.2: Estabilidad frente a Rendimiento según arquitectura.

específicos (por ejemplo, cuándo conviene atacar, huir o centrarse en comer pills). Hemos aprendido que la calidad y expresividad del dataset son tan importantes como la arquitectura del modelo.

Una de las principales limitaciones de este trabajo ha sido la forma en la que se generó el dataset. Para poder entrenar nuestros modelos, utilizamos partidas jugadas por controladores ya existentes, tanto para Pac-Man como para los fantasmas. Estos controladores venían preentrenados, y aunque fueron útiles para obtener una gran cantidad de datos rápidamente, no conocíamos en detalle cómo tomaban decisiones. Esto significa que el comportamiento que aprendió nuestro modelo estaba influenciado por unos patrones que no controlábamos ni podíamos ajustar, lo que posiblemente limitó la diversidad de situaciones recogidas en los datos.

Además, todo el enfoque que hemos seguido se ha basado en datos tabulares, es decir, en representar el estado del juego como una tabla con números y variables concretas. Aunque esto facilita el entrenamiento con ciertos algoritmos, también supone una simplificación del entorno. Por ejemplo, al no representar el mapa como una imagen o matriz, no hemos podido aprovechar modelos como las redes neuronales convolucionales, que son muy buenos analizando relaciones espaciales. En el caso de un juego como Ms. Pac-Man, donde el entorno visual y la posición relativa de los objetos es clave, esto podría ser una pérdida de información importante. Tampoco se utilizaron enfoques secuenciales, como modelos que tengan en cuenta lo que pasó en los últimos turnos, lo que habría permitido captar patrones a lo largo del tiempo y posiblemente mejorar la toma de decisiones.

Por último, conforme el proyecto fue avanzando y el tamaño del dataset aumentaba, también crecieron las necesidades de computación. En el último experimento, en el que entrenamos TabNet con una gran cantidad de datos, nos vimos obligados a usar máquinas virtuales como Google Colab, ya que entrenarlo en un equipo local resultaba inviable por el alto consumo de RAM, almacenamiento y uso de GPU. Este tipo de limitaciones técnicas también marcó el ritmo del desarrollo y nos obligó a optimizar recursos constantemente.

Como trabajo futuro, durante el desarrollo de este TFG, hemos descubierto la existencia de Sonnet, una librería desarrollada por DeepMind (Google), orientada al diseño de redes neuronales modulares y flexibles, especialmente en entornos donde los datos tienen una estructura interna compleja.

Este enfoque resulta especialmente interesante en nuestro contexto, ya que el estado del juego de Pac-Man no es un simple vector plano, sino una combinación de características distintas como Pac-Man, los fantasmas y el entorno (power pills, pills, peligros, etc.). Sonnet permite definir subredes independientes para cada bloque de datos, manteniendo dicha estructura y permitiendo que el modelo aprenda de forma más específica y contextualizada.

Aunque no ha sido posible integrar esta librería en el trabajo actual, creemos que utilizar modelos de este tipo podría mejorar la forma en la que juega Pac-Man, consiguiendo más puntuación. Especialmente en tareas de clasificación de acciones en intersecciones del mapa. Además, su compatibilidad con JAX (una librería de computación numérica acelerada y diferenciación automática) y Optax (una librería de optimización modular para modelos entrenados con JAX) facilitaría una futura migración a un entorno más optimizado.

Proponemos como línea de trabajo futura el rediseño del modelo actual utilizando Sonnet, evaluando si esta forma estructurada de representar y procesar los datos mejora la capacidad de generalización y la toma de decisiones del agente. Este enfoque también abre la puerta a combinar diferentes tipos de modelos dentro de una misma arquitectura (como convolucionales, secuenciales o de atención), algo difícil de implementar actualmente en este TFG. Además, se podría aprovechar este rediseño para incorporar nuevas fuentes de información, como secuencias de movimientos pasados o representación visual del entorno, lo que enriquecería aún más el aprendizaje del agente.

Con todo ello, este proyecto ha servido como punto de partida para explorar no solo la efectividad de distintos modelos de IA aplicados a un entorno dinámico como Ms. Pac-Man, sino también para entender en profundidad cómo la representación de los datos, la arquitectura de los modelos y las decisiones de diseño pueden marcar la diferencia entre un comportamiento limitado y uno realmente adaptativo e inteligente. El camino recorrido ha dejado muchas puertas abiertas, y sienta las bases para futuros desarrollos más completos.

Conclusions and Future Work

The main objective of this work has been to design, train and evaluate different artificial intelligence models applied to the game Ms. Pac-Man vs Ghosts. Different architectures and strategies have been investigated in order to find a model capable of deciding efficiently in dynamic and uncertain situations.

In order to achieve the main objective, we had to complete the various sub-goals we set at the beginning. First, we studied the game environment, adapting the workspace from the ICI course, identifying relevant variables, and designing new features that were not part of the original state. Next, we implemented a system to collect key data in CSV format and adapted the game engine to support automatic gameplay. The third milestone was to connect the game and the neural networks in real time via sockets. After that, we processed the data and trained models using PyTorch, Scikit-Learn, and later TabNet. Finally, we performed an explainability analysis for each model using SHAP, LIME, and Feature Importance, thereby completing all the necessary steps to achieve our objective.

The following table shows the evolution of the model in the different experiments, highlighting the average score difference of more than 1000 points between the first and the last experiment. A detail to remember is that in this fourth experiment, we had to end the game at the moments when the map advanced, thus limiting the number of games with high scores.

Table 6.2: Comparison of the results of the experiments.

Experiment	Average	Standard Deviation	Q25	Q75
Pytorch Exp 1	1514.47	652.58	20	4380
Scikit-Learn Exp 1	1685.22	685.46	270	4440
Pytorch Exp 2	2368.30	1119.38	330	7470
Scikit-Learn Exp 2	2101.43	878.35	350	7490
TabNet Exp 3	2479.68	814.57	320	6550
TabNet Exp 4	2630.08	877.15	590	5880

This progressive improvement in the results clearly shows the positive impact that the changes made in each phase have had. In particular, the difference between classic architectures such as Scikit-Learn and more modern ones such as Tabnet can

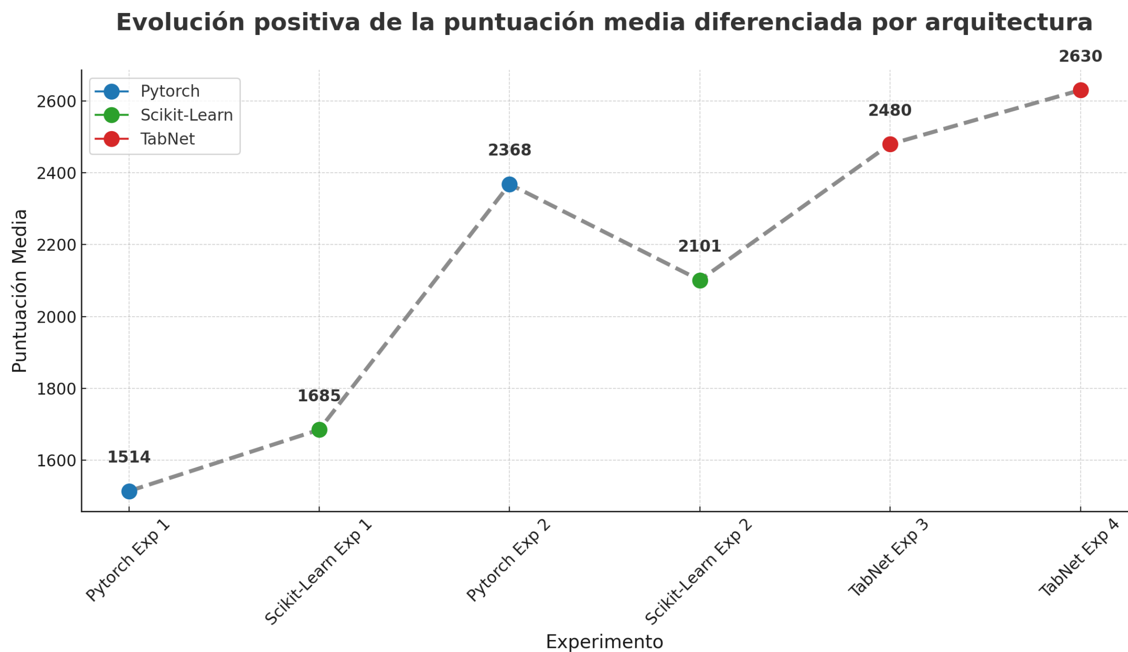

Figure 6.3: Evolution of the average score differentiated by architecture.

be observed. Its use has allowed the agent to better understand the environment, as it allows it to distinguish which are the most important characteristics at each moment of the game and give them more weight when making the decision. This has resulted in a more intelligent behaviour where the model began to make decisions based on specific situations of the moment, avoiding the repetition of patterns and achieving a more general perception of the environment.

Throughout the project, we have seen very clearly that the chosen architecture has a huge impact on the performance of the agent, especially when faced with dynamic and changing environments. It is not enough to have a model that works in general terms; it is essential that it can adapt to constant changes, identify key signals at all times, and make decisions that do not follow fixed patterns. Each experiment has taught us something new: from the limitations of the simplest models to the potential of those that integrate more advanced mechanisms of attention and contextual processing, such as TabNet.

In addition, something that has made the difference between experiments has been the work on the dataset. At the beginning, we started with basic variables, such as Pac-Man's previous direction or the position of the ghosts, but over time we realised that this was not enough to capture the real complexity of the game. We started to design new derived features, combining variables, calculating distances, evaluating states of danger or advantage, and measuring behavioural patterns. This feature engineering phase has been key: many of these new variables have provided much more value than the original ones, allowing the models to detect specific contexts (e.g. when to attack, flee or focus on eating pills). We have learned that the quality and expressiveness of the dataset are as important as the architecture of the model.

One of the main limitations of this work has been the way in which the dataset was

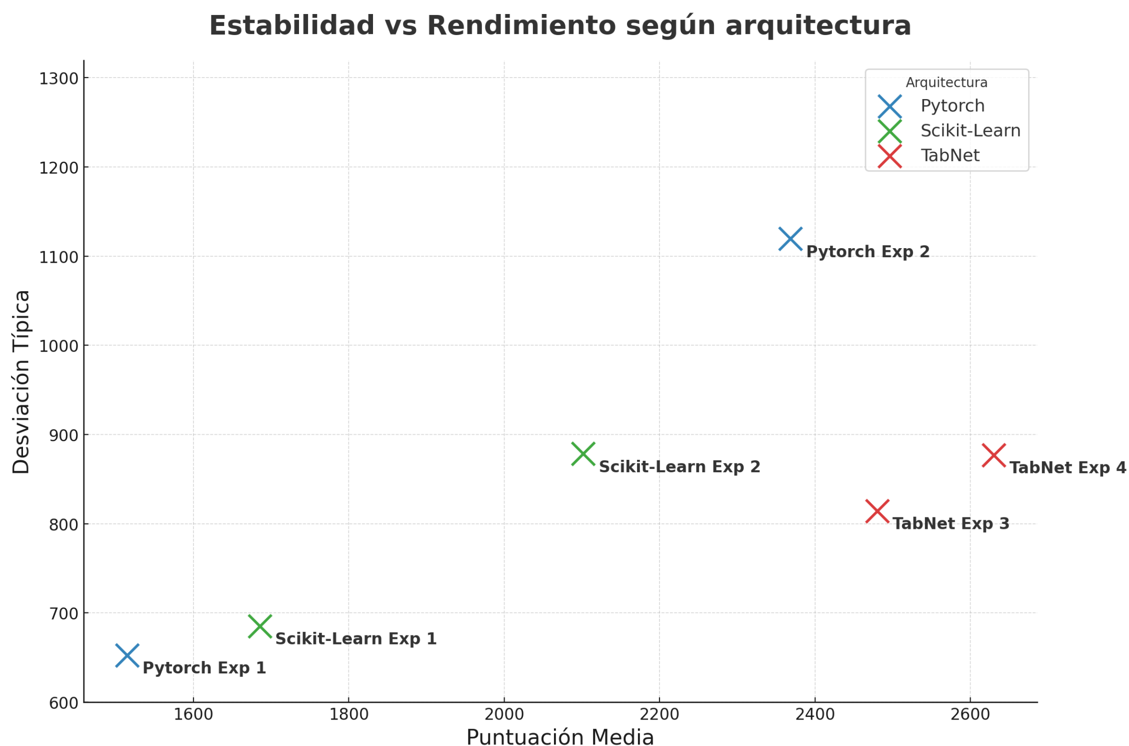

Figure 6.4: Stability vs performance according to architecture.

generated. In order to train our models, we used games played by existing controllers for both Pac-Man and ghosts. These controllers came pre-trained, and while they were useful for getting a lot of data quickly, we did not know in detail how they made decisions. This meant that the behaviour our model learned was influenced by patterns that we did not control and could not adjust, possibly limiting the diversity of situations captured in the data.

Furthermore, the whole approach we have followed has been based on tabular data, i.e. representing the state of the game as a table with concrete numbers and variables. While this facilitates training with certain algorithms, it also simplifies the environment. For example, by not representing the map as an image or matrix, we have not been able to take advantage of models such as convolutional neural networks, which are very good at analysing spatial relationships. In the case of a game like Ms. Pac-Man, where the visual environment and the relative position of objects is key, this could be a significant loss of information. Sequential approaches, such as models that take into account what happened in the last few turns, were also not used, which would have allowed us to capture patterns over time and possibly improve decision-making.

Finally, as the project progressed and the size of the dataset increased, so did the computational needs. In the last experiment, in which we trained TabNet with a large amount of data, we were forced to use virtual machines such as Google Colab, as training it on a local machine was unfeasible due to the high consumption of RAM, storage and GPU usage. These kinds of technical limitations also set the pace of development and forced us to constantly optimise resources.

As future work, during the development of this TFG, we have discovered the existence of Sonnet, a library developed by DeepMind (Google), oriented to the design of modular and flexible neural networks, especially in environments where the data has a complex internal structure.

This approach is particularly interesting in our context, as the Pac-Man game state is not a simple flat vector, but a combination of different features such as Pac-Man, ghosts and the environment (power pills, pills, hazards, etc.). Sonnet allows separate subnetworks to be defined for each block of data, maintaining that structure and allowing the model to learn in a more specific and contextualised way.

Although it has not been possible to integrate this library into the current work, we believe that using models of this type could improve the way Pac-Man plays, leading to higher scores. Especially in tasks of ranking actions at intersections on the map. Furthermore, its compatibility with JAX (a library for accelerated numerical computation and automatic differentiation) and Optax (a modular optimisation library for models trained with JAX) would facilitate a future migration to a more optimised environment.

We propose as a future line of work the redesign of the current model using Sonnet, evaluating whether this structured way of representing and processing data improves the generalisation and decision-making capacity of the agent. This approach also opens the door to combine different types of models within the same architecture (such as convolutional, sequential or attention models), which is currently difficult to implement in this TFG. Furthermore, this redesign could be used to incorporate new sources of information, such as sequences of past movements or visual representations of the environment, which would further enrich the agent's learning.

All in all, this project has served as a starting point to explore not only the effectiveness of different AI models applied to a dynamic environment such as Ms. Pac-Man, but also to understand in depth how data representation, model architecture and design decisions can make the difference between limited and truly adaptive and intelligent behaviour. The road travelled has left many doors open, and lays the groundwork for more comprehensive future developments.

Contribuciones Personales

6.1. Alejandro Bejarano del Castillo

6.1.1. Antecendentes

Mi interés por la inteligencia artificial surgió durante el segundo curso del grado. En ese momento, comencé a informarme por mi cuenta y a explorar algunos conocimientos básicos de forma autodidacta. Esta primera aproximación despertó en mí una fuerte curiosidad por el área, motivándome a seguir profundizando en el campo.

Más adelante, tuve la oportunidad de estudiarlo en mayor profundidad en las asignaturas de Inteligencia Artificial I e Inteligencia Artificial II. En ellas, aprendí diversas técnicas de aprendizaje automático, como los árboles de decisión, los métodos de clasificación supervisada y no supervisada, o los algoritmos genéticos. Estas asignaturas consolidaron mi interés por la IA, llevándome a cursar también la asignatura optativa de Inteligencia Artificial aplicada al Control.

En esta última, exploramos metodologías orientadas al diseño de sistemas inteligentes en entornos dinámicos, utilizando técnicas como la lógica difusa o los sistemas basados en reglas. Además, fue en este contexto donde tuve mi primer contacto directo con una red neuronal, aplicada al control de un brazo robótico, lo que me permitió entender de forma práctica su funcionamiento y potencial.

6.1.2. Aportación

A lo largo del desarrollo de este proyecto, he estado implicado de forma activa en distintas fases clave, combinando tareas técnicas, organizativas y de documentación.

Una de mis principales aportaciones ha sido la participación directa en el diseño y desarrollo de los modelos de redes neuronales, tanto utilizando la biblioteca Scikit-Learn como mediante PyTorch. Esta contribución incluyó la toma de decisiones relativas a la construcción de los modelos, así como la estrategia de entrenamiento. Además, llevé a cabo la implementación del código encargado de dividir el dataset original en distintos subconjuntos de datos, agrupados por intersecciones del mapa del juego Ms. Pac-Man vs Ghosts. Este paso resultó esencial para permitir el entrena-

miento individualizado de una red por intersección, optimizando el comportamiento del agente en función del contexto local.

También colaboré en el desarrollo de la lógica para el filtrado de estados válidos, descartando aquellos que no representaban situaciones óptimas o relevantes para el entrenamiento. Esto permitió refinar el conjunto de datos y asegurar que las redes aprendieran a partir de ejemplos representativos.

En el plano organizativo, me encargué de la planificación del trabajo mediante milestones en GitHub, permitiendo dividir las tareas por categorías (implementación, análisis, documentación, etc.) y visualizar de forma clara el progreso del proyecto. Asimismo, participé activamente en la toma de apuntes durante las reuniones periódicas con los tutores, cuya información fue utilizada posteriormente para ajustar y enriquecer la planificación y gestión del proyecto en GitHub.

Contribuí en el estudio del estado del arte, documentándome sobre herramientas y metodologías aplicadas en proyectos similares, especialmente en lo relativo a la inteligencia artificial en videojuegos. En las primeras etapas del proyecto, redacté parte de las subsecciones correspondientes a este análisis.

Otro aspecto relevante de mi aportación fue la extracción y análisis de gráficos de explicabilidad, utilizando herramientas como SHAP y Feature Importance, y colaborando en la interpretación de los resultados para cada uno de los experimentos realizados. Junto a mi compañero Diego González, llevamos a cabo el análisis de mapas de calor generados durante las fases de evaluación, lo que permitió identificar patrones en el comportamiento de los modelos y extraer conclusiones sobre la relevancia de las distintas variables utilizadas. Toda esta labor fue documentada en detalle en los informes de cada experimento y sirvió como base para tomar decisiones sobre la introducción de nuevas variables o ajustes en los modelos.

En conclusión, mi aportación ha estado equilibrada entre el desarrollo técnico, el análisis de resultados, la documentación y la organización del trabajo. Considero que he contribuido de forma significativa al avance del proyecto, trabajando de manera coordinada con el resto del equipo y logrando buenos resultados gracias a una comunicación constante y una planificación eficaz junto a mis compañeros.

6.2. Daniel de la Fuente Díez

6.2.1. Antecendentes

Mis primeros pasos en el ámbito de la Inteligencia Artificial comenzaron durante el Grado en Ingeniería Informática, concretamente en las asignaturas de Inteligencia Artificial I y II. A través de ellas, adquirí las competencias básicas sobre representación del conocimiento, búsqueda heurística, así como una primera aproximación al aprendizaje automático y los sistemas inteligentes. Estas asignaturas sentaron las bases teóricas que más adelante he ido reforzando tanto a nivel personal como profesional.

De forma autodidacta, he desarrollado distintos proyectos personales orientados

al aprendizaje automático. En especial, he trabajado con modelos de predicción aplicados a contextos como la estimación de precios de hoteles, combinando variables internas con fuentes externas como clima o temporada. Además, he explorado el reconocimiento de objetos en imágenes utilizando redes neuronales convolucionales (CNN), lo cual me ha permitido familiarizarme con tareas supervisadas y técnicas de procesamiento de imágenes. Este enfoque práctico me ha ayudado a consolidar lo aprendido y a ganar experiencia en el diseño, entrenamiento y validación de modelos.

A nivel profesional, formo parte de un equipo en el que he tenido la oportunidad de aplicar IA en entornos reales. Una de las tareas más relevantes ha sido la implementación de un sistema RAG (Retrieval-Augmented Generation), que combina recuperación de información con generación de lenguaje natural. Esta solución permite enriquecer las respuestas generadas por un modelo de lenguaje con información específica y actualizada, mejorando así la precisión en escenarios de consultas específicas acerca de un tema.

6.2.2. Aportación

Durante el desarrollo del proyecto, he participado activamente en distintas fases tanto teóricas como prácticas. Una de las primeras aportaciones fue la redacción y búsqueda de información para gran parte del Estado del Arte.

En los primeros experimentos, el trabajo se organizó de forma colaborativa. Personalmente, contribuí proponiendo ideas para identificar problemas de comportamiento en el modelo, como la repetición de bucles en intersecciones. Participé en la interpretación de los resultados obtenidos mediante SHAP y Feature Importance.

Mi implicación fue más directa en la parte relacionada con el modelo **TabNet**, donde me encargué de estudiar si esta arquitectura era adecuada para nuestro contexto y de diseñar los experimentos correspondientes. Empecé por investigar si esta arquitectura se adaptaba adecuadamente a nuestro contexto: datos tabulares, características con distinta escala y estructura de entrada heterogénea. Tras confirmar su idoneidad, trabajé en la implementación y ajuste de los parámetros clave del modelo, me ocupé de ajustar parámetros clave como `n_d`, `n_a`, `n_steps` y `lambda_sparse`, además de configurar el uso de `early stopping` y planificadores de tasa de aprendizaje. Esta fase también incluyó la evaluación detallada de resultados y la validación cruzada.

Más allá de los aspectos técnicos del entrenamiento, dediqué tiempo a evaluar el comportamiento del modelo con métricas e indicadores, comparando su rendimiento frente a las arquitecturas anteriores (PyTorch y Scikit-Learn) y observando su estabilidad partida a partida. Una vez analizados los primeros resultados, propuse modificaciones en la estructura del dataset y en la ingeniería de variables para mejorar la capacidad del modelo de tomar decisiones más contextualizadas.

Entre estas mejoras, propuse y ayudé a implementar una nueva variable externa de tipo *reward*, pensada como una medida del impacto real de cada acción dentro del juego. Esta variable no se incluyó como input directo, sino que se utilizó como peso en el entrenamiento, permitiendo dar más relevancia a los ejemplos en los que Pac-Man

obtenía buenos resultados (como comer un fantasma o completar un laberinto) o penalizar aquellos con consecuencias negativas (como ser atrapado). Esta estrategia se inspiró en enfoques de aprendizaje por refuerzo y permitió guiar mejor el proceso de optimización.

Además, colaboré en la propuesta y diseño de nuevas características contextuales, como el número de fantasmas activos o un modo de comportamiento, y en la revisión del tratamiento de variables categóricas, sustituyendo el one-hot encoding por codificación nativa cuando era posible, con el objetivo de reducir la dimensionalidad del dataset y aprovechar mejor las capacidades internas de TabNet para seleccionar qué variables atender en cada paso de decisión.

En conjunto, considero que mi aportación ha sido equilibrada entre las fases de análisis, diseño experimental y desarrollo, con una especial implicación en la integración y evaluación del modelo TabNet. No obstante, muchas de estas mejoras fueron fruto de un trabajo conjunto continuo, donde la coordinación y discusión entre todos los miembros del equipo fueron claves para la evolución del proyecto.

6.3. Diego González López

6.3.1. Antecendentes

Las asignaturas Inteligencia Artificial I y II fueron mi único contacto con la Inteligencia Artificial hasta el inicio de este trabajo. Es en estas donde aprendí la base teórica y práctica de las técnicas que posteriormente hemos empleado en el trabajo, como el aprendizaje por refuerzo y el aprendizaje supervisado.

Además, en la asignatura Análisis de Redes Sociales aprendí a trabajar y analizar grandes volúmenes de datos. Este conocimiento adquirido me resultó muy útil para explotar al máximo una de las técnicas más importantes del trabajo como es el análisis de la Explicabilidad de los modelos. Asimismo, las asignaturas de Redes y Programación Concurrente me resultaron muy útiles para entender la lógica cliente-servidor, lo que facilitó la comunicación entre el motor del juego en Java y las redes neuronales implementadas en Python.

Debido a que mi conocimiento sobre la Inteligencia Artificial se limitaba a conceptos básicos, tuve que aprender de manera autodidacta mediante cursos online. Es en estos cursos donde diseñé mi primer modelo de red neuronal multicapa, por lo que, al implementarlo manualmente pude comprender en detalle el funcionamiento de estas.

6.3.2. Aportación

Durante las primeras fases del proyecto me encargué de diseñar gran parte del preprocesado de datos, definiendo como normalizar y preparar cada estado del juego para su posterior entrenamiento. Tras esto participé de manera directa en la creación

de los primeros modelos en Scikit-Learn y Pytorch e implementé la lógica necesaria para su entrenamiento continuo.

Para poder automatizar las diferentes funciones del codigo diseñé un flujo de ejecución que permitía alternar entre entrenar nuevos modelos, evaluar el modelo en tiempo real mediante la conexión con el motor del juego y extraer la explicabilidad de cada modelo. Todo esto permitió asentar una base sobre la que pudimos trabajar de manera continua entre experimentos.

Con el objetivo de poder extraer la mayor información del comportamiento de nuestros modelos, desarrollé el sistema de generación de mapas de calor, el cual muestra la importancia que le asigna el agente a cada característica en las diferentes zonas del tablero. Mediante los valores de impacto para cada característica por intersección obtenidos en la explicabilidad creé un mecanismo que traduce esos impactos en colores más rojos o verdes en función de la importancia asignada. Estos mapas nos han permitido hacer un mejor seguimiento de cómo actua el modelo, enriqueciendo así las discusiones sobre los ajustes y refuerzos necesarios.

Durante la elaboración de la memoria participé de manera directa en gran parte del análisis de la explicabilidad de los experimentos, así como en la generación de la mayoría de las gráficas de importancia y mapas de calor. Al haber diseñado gran parte del código en Python, me encargué de redactar lo realizado en la Descripción del Trabajo.

Gracias al análisis de la explicabilidad y de los mapas de calor pudimos entender que el agente necesitaba compensar las importancias entre características. Por lo que en esta fase final trabajé junto con mis compañeros en el desarrollo de un nuevo modelo. Propuse la inclusión de nuevas variables, como el número de fantasmas peligrosos y comestibles cercanos o el modo de juego de Pac-Man, con el objetivo de que el modelo comenzara a apoyarse más en la información que tenía sobre los fantasmas. Junto con mi compañero Daniel de la Fuente nos encargamos de diseñar estas características e integrarlas en el conjunto de entrenamiento para su posterior análisis.

Los últimos días antes de la entrega final dediqué gran parte del tiempo en preparar el proyecto para que pudieran ser probados todos los modelos creados en los diferentes experimentos. Redacté un README donde se explica qué pasos hay que hacer para poder ejecutar dichos modelos.

En general, considero que he participado de manera equilibrada tanto en el diseño de los modelos, análisis de estos e integración del proyecto en la memoria. Sin embargo, el éxito de este trabajo se debe al constante intercambio de ideas en reuniones semanales que hemos tenido durante todos estos meses.

6.4. Daniel Jiménez Rojo

6.4.1. Antecendentes

Mi interés por la Inteligencia Artificial nació en segundo curso del grado, cuando empecé a investigar por mi cuenta y a realizar pequeños proyectos autodidactas para entender mejor qué había detrás de los algoritmos de inteligencia artificial. Aquella curiosidad fue tan fuerte que, al elegir itinerario al finalizar ese año, opté por el de Computación con el objetivo principal de profundizar en IA.

Ya en tercer curso cursé Inteligencia Artificial I y II, que me proporcionaron los fundamentos formales de la disciplina: representación del conocimiento, búsqueda heurística, algoritmos adversarios y las primeras técnicas de aprendizaje supervisado. Estos contenidos asentaron la base necesaria para abordar materias más avanzadas y me confirmaron que la IA era el campo en el que quería especializarme.

A partir de ahí me matriculé en todas las optativas disponibles relacionadas con el tema. En Aprendizaje Automático y Big Data, por ejemplo, trabajé con regresión, clasificación, árboles de decisión, clustering, sistemas de recomendación y redes neuronales, aplicando lo aprendido en un proyecto de detección de correos spam donde comparé clasificadores tradicionales y modelos neuronales en Scikit-Learn y PyTorch.

La asignatura de Inteligencia Artificial Aplicada al Control me mostró el potencial de la IA en entornos dinámicos; implementé controladores PID y sistemas basados en lógica difusa y construí mi primera red neuronal para controlar un brazo robótico, experiencia que conectó teoría y aplicación real de forma muy práctica.

Finalmente, cursé la asignatura de Ingeniería de Comportamientos Inteligentes al inicio del mismo curso en el que comenzamos este Trabajo de Fin de Grado, lo que resultó de gran ayuda para entender a fondo el funcionamiento interno del juego de Pac-Man y qué elementos influían más en la toma de decisiones del agente. Las prácticas realizadas en esta asignatura me sirvieron como una base sólida para enfocar el proyecto desde una perspectiva más técnica y fundamentada. Aunque en clase trabajamos con controladores basados en reglas (RBS), lógica difusa (Fuzzy) y razonamiento por casos (CBR), este conocimiento me permitió comprender mejor el entorno y diseñar una solución alternativa con redes neuronales. Me alegra haber podido contribuir con un enfoque novedoso a la línea docente de la asignatura, que considero especialmente útil y formativa dentro del itinerario de Computación.

6.4.2. Aportación

En las fases iniciales del proyecto, depuré el workspace Java heredado de la asignatura ICI, y sobre los módulos esenciales, levanté el proyecto *PacmanNeuro*, la base de la lógica que permite a Pac-Man tomar decisiones mediante redes neuronales entrenadas previamente en Python. En la fase inicial desarrollé, junto a mi compañero Daniel de la Fuente, un primer socket bidireccional Java-Python que transmitía el

estado del juego y devolvía la acción en tiempo real. Más adelante, optimicé ambos extremos del socket, asegurando una conexión persistente entre partidas, incorporando un manejo robusto de errores y sincronizando correctamente los intercambios de mensajes.

Sobre esa base desarrollé *DataManager*, el engranaje que transforma el flujo continuo de estados del juego en un dataset limpio y útil. En cada tick captura el estado del juego, elimina información innecesaria, calcula nuevas variables y lo guarda provisionalmente en un búfer en forma de cola. Un validador decide si el estado ha sido prometedor y lo almacena en memoria, evitando estados en los que Pac-Man no ha jugado bien. Inicialmente diseñé todo el pipeline y, más adelante, lo revisé con el equipo para introducir criterios inspirados en aprendizaje por refuerzo que afinan aún más la selección de estados.

En paralelo diseñé dos modos de ejecución. El primero automatiza la generación de datasets: lanza partidas con diversas combinaciones de controladores, valida cada estado mediante DataManager y almacena solo las muestras útiles para el entrenamiento. El segundo evalúa agentes enfrentando cualquier versión de Pac-Man a distintos equipos de fantasmas para medir su rendimiento. Al finalizar cualquiera de los dos modos, el sistema compila un informe estadístico con medias, diagramas de caja e histogramas elaborados con JFreeChart, de modo que podemos contrastar, con métricas objetivas, tanto la calidad de los datos como la eficacia de los modelos entrenados.

En las fases iniciales aporté modelos preliminares de PyTorch y Scikit-Learn, desarrollados previamente en asignaturas de la carrera como Aprendizaje Automático y Big Data, que sirvieron de banco de pruebas para explorar arquitecturas y métodos de entrenamiento, sentando así la base sobre la que se iría puliendo el rendimiento de los modelos. Asumí la recopilación y el análisis de las métricas en los experimentos, tanto de datasets, como de modelos implementados: combiné técnicas de explicabilidad con los estudios estadísticos para descubrir patrones de fallo y generar propuestas de ajuste. Cada mejora se debatía siempre en sesiones conjuntas, donde los cuatro en equipo, contrastábamos enfoques, unos más centrado en datos, en infraestructura, visualización, y solo se aplicaba lo que alcanzaba consenso. Este ciclo aplicado en todos los experimentos, facilitó ajustes clave en variables y datasets, mejorando cada vez más el aprendizaje de los modelos.

Adapté la explicabilidad de LIME al contexto de nuestros modelos, que contienen una red distinta por intersección: la herramienta filtra solo los estados de la intersección correspondiente, ejecuta la predicción de esa red concreta y genera la gráfica de los pesos con mayor impacto. El análisis resultante mostró decisiones mayormente coherentes con la situación del juego y ayudó a indentificar algunas variables que influían de forma inesperada o desproporcionada en las predicciones, lo que motivó ajustes en los conjuntos de datos y en la arquitectura de las redes.

A lo largo del proyecto considero que he aportado de forma equivalente a mis compañeros, participando activamente en tareas clave como la generación de datasets, el estudio estadístico del rendimiento, el análisis con técnicas de explicabilidad como LIME y la interpretación de los resultados obtenidos para proponer mejoras

tanto en los modelos como en los conjuntos de datos. Gracias al trabajo conjunto y al apoyo constante de mis compañeros, hemos conseguido llevar a cabo un proyecto sólido y bien estructurado. Ha sido un placer formar parte de este equipo, en el que cada uno ha contribuido desde sus fortalezas y del que me llevo no solo aprendizajes técnicos, sino también una experiencia de colaboración muy enriquecedora.

Bibliografía

APORIA. Feature importance: 7 methods and a quick tutorial. 2022. Último acceso: enero 2025.

C3.ai. What is local interpretable model-agnostic explanations (lime)? 2024. Último acceso: enero 2025.

DataCamp. Introducción a las redes neuronales convolucionales (cnn). 2024. Último acceso: enero 2025.

DeepMind. Alphastar: Mastering the real-time strategy game starcraft ii. 2019.

de Delatorre AI, E. ¿qué es shap? explicación clara para modelos de ia. 2024. Último acceso: diciembre 2024.

Diez, D. M., Barr, C. D. y Çetinkaya Rundel, M. *OpenIntro Statistics*. OpenIntro, 2024.

ESAN. Explicabilidad: Un concepto clave para el uso eficaz de modelos de inteligencia artificial. 2024. Último acceso: febrero 2025.

GAIA Research Group. Entorno virtual para el uso de videojuegos como estrategia de enseñanza activa y método de evaluación. 2021. Último acceso: mayo 2025.

Games, W. Mastering ai-powered procedural content generation for games. 2025.

Gildardo Sanchez-Ante, E. Sistemas inteligentes: Reportes finales ene-may 2014. *Reporte Tecnico RT-0001-2014*, 2014.

GoogleCloud. Descripción general de explainable ai de bigquery. 2025. Último acceso: febrero 2025.

Grand, S., Cliff, D. y Malhotra, A. Creatures: Artificial life autonomous software agents for home entertainment. Informe técnico, University of Sussex, Cognitive Science Research Paper 434, 1996.

Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 1982.

IBM. El modelo de redes neuronales. 2021. Último acceso: enero 2025.

IBM. ¿qué es la ia explicable? 2024. Último acceso: febrero 2025.

Jagdale, D. Finite state machine in game development. *International Journal of Advanced Research in Science, Communication and Technology*, 2021.

Juan Martinez, M. Ai in video games: a historical evolution, from search trees to llms. chapter 2: 1980–2000. 2023. Último acceso: enero 2025.

Keita, Z. Arquitectura de las cnn aplicada al reconocimiento de dígitos. 2025. Consultado en 2025.

Krizhevsky, A., Sutskever, I. y Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, vol. 25, páginas 1097–1105, 2012.

LeCun, Y., Bottou, L., Bengio, Y. y Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, vol. 86(11), páginas 2278–2324, 1998.

ManagementSolutions. Explainable artificial intelligence (xai) desafíos en la interpretabilidad de los modelos. 2023. Último acceso: febrero 2025.

McCulloch, W. S. y Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 1943.

Minsky, M. y Papert, S. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

More, S. Game navigation: A* pathfinding in unity. https://swapnilmore03.medium.com/game-navigation-a-pathfinding-in-unity-4e8203e3d40b, 2024. Consultado el 25 de mayo de 2025.

O'Shea, K. y Nash, R. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

ReasonWhy. De los sesgos a la manipulación, la cuestión ética es ineludible en el desarrollo de la inteligencia artificial. 2024. Último acceso: febrero 2025.

ResearchGate. Esquema de red neuronal de tres capas. 2023. Consultado en 2025.

Rohlfshagen, P., Pérez-Liébana, D. y Lucas, S. Ms. pac-man versus ghost team cig competition. En *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. 2016. PDF en línea. Último acceso: mayo 2025.

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 1958.

Rumelhart, D. E., Hinton, G. E. y Williams, R. J. Learning representations by back-propagating errors. *Nature*, 1986.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural Networks*, 2015.

THE ROYAL SWEDISH ACADEMY OF SCIENCES. The nobel prize in physics 2024. https://www.nobelprize.org/prizes/physics/2024/press-release/, 2024. Awarded to John J. Hopfield and Geoffrey Hinton "for foundational discoveries and inventions that enable machine learning with artificial neural networks".

TOPTAL. Unity ai development: A finite-state machine tutorial. https://www.toptal.com/unity/unity-ai-development-finite-state-machine-tutorial, 2021.

WIKIPEDIA. Representación de la estructura de una red neuronal. 2025. Consultado en 2025.

YUSUF, A. M. Implementation of dijkstra's algorithm in pathfinding for artificial intelligence in video games. *Informatika STEI ITB*, 2022.