DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN QUE PERMITA EL DESARROLLO DE PARTIDAS DE NLHE PARA HUMANOS Y BOTS

DESIGN AND IMPLEMENTATION OF AN APPLICATION THAT ALLOWS THE DEVELOPMENT OF NLHE GAMES FOR HUMANS AND BOTS



TRABAJO FIN DE GRADO
CURSO 2025-2026

AUTORES
ÓSCAR PINEDA GERMÁN (GIC)
VALERIA CORINA PULIDO LOZADA(GIS)
CARLA TOAPANTA TAIPE (GII)

DIRECTOR
ALBERTO NÚÑEZ COVARRUBIAS

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN QUE PERMITA EL DESARROLLO DE PARTIDAS DE NLHE PARA HUMANOS Y BOTS

DESIGN AND IMPLEMENTATION OF AN APPLICATION THAT ALLOWS THE DEVELOPMENT OF NLHE GAMES FOR HUMANS AND BOTS

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

AUTORES
ÓSCAR PINEDA GERMÁN (GIC)
VALERIA CORINA PULIDO LOZADA (GIS)
CARLA TOAPANTA TAIPE (GII)

DIRECTOR
ALBERTO NÚÑEZ COVARRUBIAS

CONVOCATORIA: SEPTIEMBRE 2026

ÓSCAR PINEDA GERMÁN – SOBRESALIENTE (9,9)
VALERIA CORINA PULIDO LOZADA – SOBRESALIENTE (9,9)
CARLA TOAPANTA TAIPE – SOBRESALIENTE (9,9)

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

8 DE SEPTIEMBRE DE 2026

Autorización de Difusión

Autores

Óscar Pineda Germán (GIC)

Valeria Corina Pulido (GIS)

Carla Toapanta Taipe (GII)

Fecha

Madrid, 8 de Septiembre de 2026.

Los abajo firmantes, matriculados en: Grado de Ingeniería Informática, Grado de Ingeniería de Computadores y Grado de Ingeniería de Software, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo de Fin de Grado: Diseño e implementación de una aplicación que permita el desarrollo de partidas de NLHE para humanos y bots, realizado durante el curso académico 2025-26 bajo la dirección de Alberto Núñex Covarrubias, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

DEDICATORIA

A todos nuestros familiares por apoyarnos constantemente y acompañarnos durante toda esta etapa académica. Gracias por estar siempre presentes y por impulsarnos a seguir adelante sobre todo en los momentos más difíciles.

AGRADECIMIENTOS

En primer lugar, queremos agradecer a nuestro tutor, por su dedicación y apoyo durante la realización de este trabajo, así como sus consejos y sugerencias a lo largo de este proceso. También queremos agradecer a los participantes que colaboraron en las pruebas para nuestro estudio empírico por invertir su tiempo y por su gran implicación. Por último, agradecemos a nuestros familiares y amigos por su apoyo y su paciencia a lo largo de estos años de carrera, porque sin ellos no estaríamos escribiendo esta memoria

RESUMEN

Diseño e implementación de una aplicación que permita el desarrollo de partidas de NLHE para humanos y bots

El siguiente trabajo que se presenta consta del diseño e implementación de una aplicación que permite el desarrollo de partidas de NLHE para humanos y bots. El objetivo principal del proyecto es realizar una aplicación en la que los bots puedan integrarse en partidas de póker de la forma más fácil y extensible posible sin importar el modelo de inteligencia artificial que se haya elegido para su implementación. De esta manera esta aplicación sirve como herramienta para el desarrollo de nuevas tecnologías de inteligencia artificial en juegos imperfectos como el póker.

Este proyecto incluye una descripción de los fundamentos teóricos y tecnológicos que definen el entorno del proyecto. A continuación, se profundiza en el diseño e implementación de la aplicación desarrollada para alcanzar el objetivo principal. Además, se ha realizado un estudio empírico en el que los diferentes bots implementados juegan en múltiples escenarios, esto tiene como propósito poder utilizar la infraestructura creada para evaluar el comportamiento de los bots, comprobar la capacidad de la aplicación para soportar este tipo de experimentos y disponer de una base objetiva sobre la que analizar sus decisiones y rendimiento.

Palabras clave

Póker NLHE, framework, Inteligencia artificial, interfaz gráfica, distribuida, bot, aprendizaje, integración, LLM

ABSTRACT

Design and implementation of an application that allows the development of NLHE games for humans and bots

This project focuses on the design and implementation of an application that enables No-Limit Texas Hold'em games to be played by both human players and bots. The primary objective of the project is to develop a platform in which bots can be integrated into poker games as easily and extensibly as possible, regardless of the artificial intelligence model chosen for their implementation. In this way, the application serves as a valuable tool for the development and evaluation of new artificial intelligence technologies in imperfect-information games such as poker.

The project includes a description of the theoretical and technological foundations that define the context in which it is developed. It then provides a detailed discussion of the design and implementation of the application created to achieve the project's main objective. Furthermore, an empirical study has been conducted in which the different implemented bots compete across multiple scenarios. The purpose of this study is to make use of the infrastructure developed to evaluate bot behaviour, assess the application's ability to support this type of experimentation, and provide an objective basis for analyzing the bots decision-making processes and overall performance.

Keywords

No-Limit Texas Hold'em Poker, Artificial Intelligence, Graphical User Interface, Distributed Architecture, Learning, Integration, Large Language Model.

ÍNDICE DE CONTENIDOS

Capítulo 1 - Introducción 1
1.1 Motivación y alcance 1
1.2 Objetivos 2
1.3 Plan de trabajo 3
1.3.1 Planificación del proyecto 3
1.3.2 Fase I: Análisis y diseño del framework 5
1.3.3 Fase II: Desarrollo de la lógica del juego y la integración de los bots 6
1.3.4 Fase III: Desarrollo de la interfaz gráfica 7
1.3.5 Fase IV: Desarrollo de la arquitectura distribuida 7
1.3.6 Fase V: Experimentación y evaluación 8
1.3.7 Fase VI: Documentación y elaboración de la memoria 9
1.4 Estructura del documento 9
Capítulo 2 - Contextualización 11
2.1 Póker NLHE 11
2.1.1 Historia y definición del póker 11
2.1.2 Reglas del póker NLHE 12
2.1.3 Glosario del póker 14
2.2 Técnicas de IA para jugar al póker 15
2.2.1 Teoría de juegos 15
2.2.2 Aprendizaje por refuerzo 16
2.2.3 Redes Neuronales 20
2.2.4 LLM (Large Language Model) 24
2.2.5 Counterfactual Regret Minimization (CFR) 25
Capítulo 3 - Diseño 27
3.1 GUI 27

3.2 Componentes de una partida de póker 30
3.3 Integración de bots 32
3.4 Arquitectura distribuida 35
3.5 Monitorización de partidas 35
Capítulo 4 - Implementación 37
4.1 GUI 37
4.1.1 Configuración inicial y gestión de vistas 38
4.1.2 Creación de una partida 39
4.1.3 Agregación y configuración de bots 41
4.1.4 Unión de una partida y sala de espera 43
4.1.5 En partida 45
4.1.6 Visualización de la interfaz 48
4.1.7 Generación dinámica de avatares 51
4.2 Componentes de una partida de póker 53
4.3 Integración de bots 63
4.3.1 Ejemplo de integración LLM Online (OpenAI) 75
4.3.2 Ejemplo de integración de Red Neuronal 79
4.4 Arquitectura distribuida 81
4.4.1 Common 81
4.4.2 Aplicación servidor 83
4.4.3 Aplicación cliente 85
4.5 Monitorización de partidas 87
Capítulo 5 - Estudio empírico 89
5.1 Proceso experimental 89
5.2 Objetivo del estudio 91
5.2.1 Objetivos generales 91

5.2.2 Objetivos específicos por experimento 92
5.3 Diseño y configuración experimental 92
5.3.1 Variables experimentales 92
5.3.2 Experimentación 93
5.4 Métricas de evaluación 95
5.4.1 Métricas de comportamiento 95
5.4.2 Métricas de rendimiento 96
5.4.3 Métricas relacionadas con el equity 96
5.4.4 Métricas de agresividad y All-ins 97
5.4.5 Métricas de decisiones y evolución 97
5.5 Resultados de los experimentos 97
5.5.1 ¿Cómo juega cada bot? 98
5.5.2 ¿Qué bot obtiene mejores resultados? 100
5.5.3 ¿La agresividad mejora los resultados? 101
5.5.4 ¿Qué decisiones tienen mayor impacto en el resultado? 103
5.5.5 ¿Cómo evoluciona el bot durante la partida? 105
5.5.6 ¿Cómo consigue ganar cada bot? 106
5.5.7 ¿Influye el tipo de partida? 107
5.5.8 ¿Cómo cambia el bot frente a distintos niveles de jugador? 108
5.5.9 ¿Cómo se comportan los bots en una mesa con humanos? 110
5.5.10 ¿Los estilos cambian el comportamiento del mismo modelo? 111
5.5.11 ¿Cómo compite el LLM contra NN y FSM? 113
5.6 Análisis y discusión de resultados 114
Capítulo 6 - Conclusiones y trabajo futuro 117
Introduction 121
Conclusions and future work 131
Contribuciones personales 135

Bibliografía **143**

Capítulo 1 - Introducción

En este capítulo presentaremos las diferentes secciones del documento que explican en detalle los objetivos, contexto, plan de trabajo y dificultades durante el desarrollo, entre otros.

1.1 Motivación y alcance

La idea de máquinas o seres inteligentes creados por los humanos ha sido recurrente a lo largo de la historia de la humanidad, desde obras clásicas como Los viajes de Gulliver[1] de Jonathan Swift hasta películas más modernas como The Matrix de las hermanas Wachowski.

La sociedad moderna ha logrado introducir máquinas en prácticamente todas las áreas dominadas por los humanos, incluso en ámbitos en los que se creía que éramos imbatibles como en juegos de estrategia similares a las damas o el ajedrez.

Actualmente, las inteligencias artificiales son superiores en los juegos de estrategia mencionados. Sin embargo, el póker sigue siendo un área especialmente compleja para que puedan superar a los humanos, lo cual presenta un área idónea para una herramienta para evaluar, estudiar, observar e interactuar con diferentes tipos de inteligencias artificiales en el póker, sobre todo con el reciente auge de los modelos de inteligencia artificial más modernos.

En cuanto al alcance de nuestro proyecto debemos destacar primeramente la posibilidad de que otros desarrolladores diseñen, entrenen, ajusten y evalúen sus propios modelos de inteligencia artificial de cualquier tipo, así como incorporar modelos desarrollados por terceros con el mismo fin. Esto puede ocurrir en entornos mixtos, es decir; con una mezcla de humanos y bots, o puramente formado por bots y con diferentes tipos de configuraciones de partida.

En segundo lugar y también centrándonos en otros desarrolladores, la aplicación ofrece la posibilidad de realizar modificaciones para mejorar o añadir nuevas funcionalidades a esta mediante variedad de decisiones de diseño con el fin de obtener un producto software flexible y capaz de adaptarse a nuevas necesidades.

Además del uso esperado por parte de otros desarrolladores, otros perfiles no técnicos pueden utilizar la aplicación para organizar partidas entre humanos y/o bots con un máximo de 9 jugadores de cualquier tipo de forma totalmente casual, recreativa y/o de aprendizaje, bien sea de forma individual o para un grupo de personas.

1.2 Objetivos

El objetivo principal de este proyecto es implementar una aplicación que permita jugar partidas de No-Limit Texas Hold'em (NLHE) a jugadores humanos y agentes artificiales, y que sirva como herramienta para integrar cualquier tipo de bot de forma fácil y extensible.

La principal motivación del proyecto surge de la necesidad de disponer de una plataforma intuitiva que facilite la integración y evaluación de modelos de inteligencia artificial dentro de un entorno de juego de información imperfecta como es el póker. Para conseguirlo, utilizamos en una estructura distribuida que permite agregar nuevos tipos de bots de forma sencilla, independientemente de la tecnología usada para su implementación.

Asimismo, se persigue alcanzar los siguientes objetivos secundarios:

1. Desarrollar una aplicación funcional de póker NLHE multijugador, permitiendo que puedan jugar tanto participantes humanos como bots dentro de una misma partida.
2. Diseñar una arquitectura extensible y distribuida, que facilite la integración de varios tipos de bots sin necesidad de modificar el núcleo de la aplicación.
3. Integrar diferentes agentes a las partidas de póker, sin importar el modelo de inteligencia artificial.
4. Implementar mecanismos de monitorización y análisis, capaces de registrar información relevante de las partidas y las decisiones tomadas por los bots.

1.3 Plan de trabajo

El desarrollo de esta aplicación fue planteado para seguir un enfoque iterativo e incremental, de forma que la aplicación se construyera progresivamente mediante ciclos de análisis, diseño, implementación y prueba. Este enfoque ha permitido partir de una estructura inicial básica y ampliarla a medida que se han incorporado nuevas funcionalidades, revisando y mejorando las decisiones tomadas en etapas anteriores cuando ha sido necesario.

Aunque las fases se describen individualmente para facilitar su explicación, no se ha seguido un desarrollo completamente lineal, sino que existe un solapamiento entre ellas. Por el contrario, las distintas partes del proyecto han evolucionado de forma conjunta, de manera que lo observado durante la implementación y las pruebas permitiera detectar nuevas necesidades y realizar modificaciones sobre el diseño inicial.

1.3.1 Planificación del proyecto

Al tratarse de un proyecto desarrollado por tres integrantes, desde la primera etapa fue necesario establecer una forma de organización que permitiera repartir el trabajo y mantener una visión común sobre el estado del proyecto. La planificación no se estableció como una distribución fija de tareas al comienzo del TFG, sino que se fue adaptando durante el desarrollo en función de las necesidades que iban apareciendo. De esta manera las tareas podrían dividirse, modificarse, priorizarse o reasignarse cuando era necesario.

Para organizar el trabajo adoptamos algunos principios de la metodología Scrum, aunque adaptados a las características de nuestro proyecto y a un equipo formado únicamente por tres personas. No seguimos Scrum de forma estricta, ya que no asignamos ningunos de sus roles ni establecimos *sprints* de duración fija. En su lugar, utilizamos una planificación incremental basada en tareas y revisiones periódicas del estado del proyecto.

Desde el inicio del proyecto, en septiembre de 2025, utilizamos GitHub como plataforma principal para organizar el desarrollo, ya que era la herramienta con la que los integrantes estábamos más familiarizados. Se creó un repositorio común y se utilizó *GitHub Projects* para gestionar las tareas mediante un tablero. En él podíamos

añadir y asignar tareas, así como realizar un seguimiento de su estado, diferenciando aquellas pendientes, preparadas para comenzar, en desarrollo o pendientes de revisión. Esto nos permitía coordinar el trabajo y conocer en todo momento las tareas pendientes y las que estaba realizando cada integrante.

La coordinación se realizó principalmente mediante reuniones periódicas tanto presenciales como online mediante *Teams*, con una frecuencia de tres o cuatro reuniones al mes, aunque esto podía variar dependiendo de las necesidades del proyecto. En ella revisábamos el trabajo realizado, comentábamos las dificultades y establecíamos las siguientes tareas. Los principales temas tratados en cada reunión se recogían en un historial para mantener un registro de las decisiones tomadas.

Para el código utilizamos Git para el repositorio común y se utilizaban diferentes ramas *main* y *develop*, como ramas principales, mientras que cada integrante podía trabajar en ramas individuales para desarrollar las tareas que tenía asignadas. Así los cambios permanecían separados del código común hasta que estaban preparados para ser integrados mediante *Pull requests*. Esto nos permitió trabajar de forma paralela sin modificar ramas comunes y poder hacer revisiones previas.

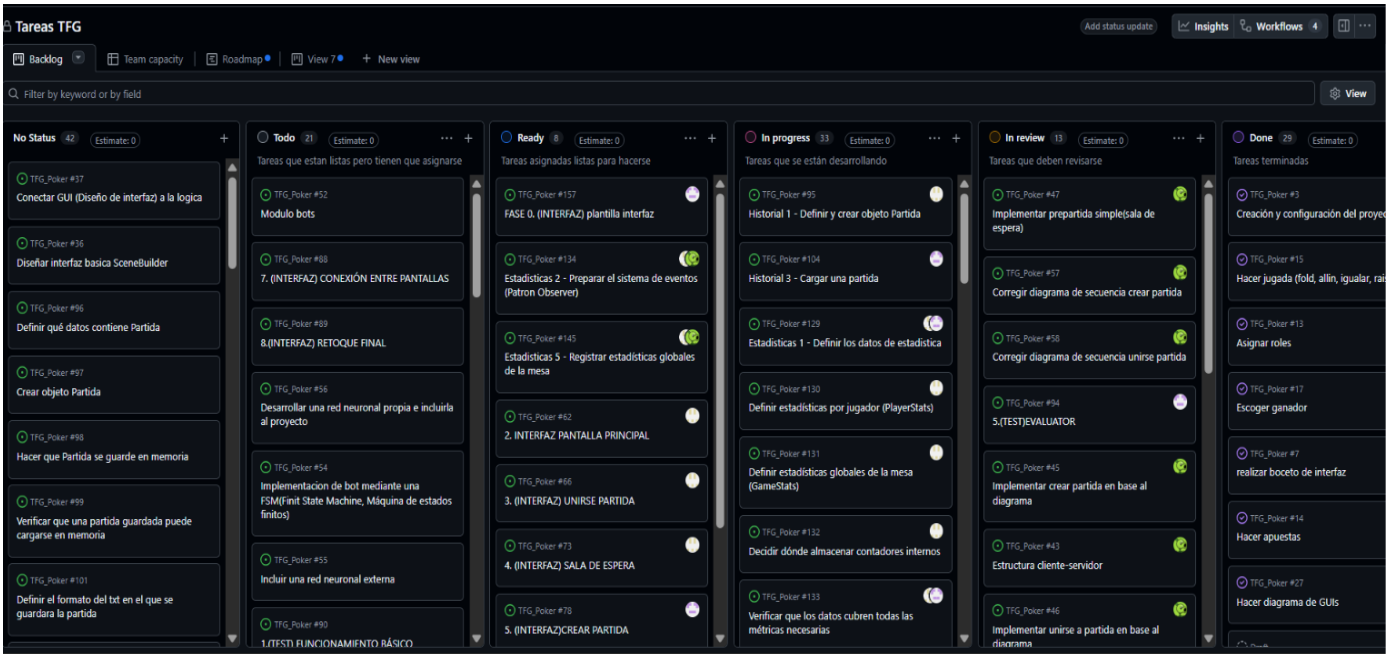


Figura 1-1. Ejemplo del backlog de tareas utilizado durante el desarrollo del proyecto

Durante las primeras etapas fue necesario establecer una forma de trabajo común, definir cómo se organizarían las tareas y decidir las herramientas que

utilizaríamos para coordinarnos, como la mostrada en la Figura 1-1. Posteriormente fue posible trabajar en paralelo manteniendo un código común y controlado.

1.3.2 Fase I: Análisis y diseño del framework

El objetivo de esta primera fase fue establecer una base sobre la que desarrollar el resto de la aplicación. Durante las primeras semanas dedicamos una parte importante de tiempo a analizar las necesidades del proyecto, definir su estructura y concretar las funcionalidades que queríamos incluir. También revisamos proyectos anteriores realizados por los integrantes del equipo y buscamos repositorios y otros recursos que pudieran servirnos como referencia. Entre los recursos encontrados se encontraba una implementación para la evaluación de las jugadas de póker que posteriormente utilizamos como referencia.

Durante las primeras semanas trabajamos principalmente de forma conjunta. El 18 de septiembre de 2025 realizamos un primer diagrama general de clases de la lógica del juego, que posteriormente revisamos y ampliamos hasta obtener un diagrama de clases más completo el 29 de septiembre. A continuación, el 2 de octubre definimos los principales casos de uso de la aplicación, relacionados con la gestión de jugadores y cartas, la configuración de la partida y las diferentes acciones durante una mano de póker. Esto nos permitió concretar las funcionalidades y comenzar a transformarlas en tareas de desarrollo.

Paralelamente, comenzamos a tener reuniones con el tutor para revisar el planteamiento y el alcance del proyecto. Durante esta etapa detectamos que inicialmente habíamos planteado un número demasiado elevado de funcionalidades. Tras revisar junto con el tutor, establecimos una prioridad entre ellos, diferenciando las necesarias para cumplir los objetivos principales.

Una vez establecidas estas prioridades, el 9 de octubre elaboramos una primera lista de tareas para el sprint backlog. Durante las semanas siguientes realizamos las primeras implementaciones y pruebas para comprobar la estructura planteada. El 16 de octubre se comenzó a trabajar sobre la lógica del sistema y el 22 de octubre se incorporaron mecanismos de depuración. Finalmente, el 5 de noviembre se dejó preparada una primera base funcional.

Como resultado conseguimos establecer la estructura inicial de la aplicación, concretar sus funcionalidades y ajustar el alcance a los objetivos del Trabajo de Fin de Grado. Fue un punto de partida para continuar con el desarrollo.

1.3.3 Fase II: Desarrollo de la lógica del juego y la integración de los bots

El objetivo principal de esta segunda fase fue desarrollar la lógica necesaria para poder ejecutar una partida de No-Limit Texas Hold'em e integrar diferentes tipos de bots. Esta fase fue una de las más importantes del proyecto, ya que es la principal aportación del TFG. Por ello, no nos limitamos a desarrollar un conjunto de bots concretos, sino que fuimos construyendo y comprobando progresivamente una estructura que permitiera integrar a cualquier usuario sus propias implementaciones y utilizarlas para jugar al póker.

Durante las primeras etapas terminamos de desarrollar la lógica básica del juego necesaria para poder ejecutar una partida y gestionar las acciones de los jugadores. Para ello, utilizamos como referencia algunos proyectos relacionados con este ámbito, adaptando las soluciones que podían resultar útiles a las necesidades de nuestra aplicación. Una vez tuvimos esta base, empezamos a integrar los bots poco a poco. El primer bot se desarrolló en febrero de 2026 y estaba basado en un modelo de lenguaje local (LLM). En esta fase investigamos diferentes alternativas, consultando repositorios y plataformas como Hugging Face y R, probando distintos modelos hasta encontrar los que mejor se adaptaban al proyecto. En marzo incorporamos un segundo bot basado en un modelo de lenguaje online y posteriormente, durante abril, añadimos los bots basados en una red neuronal (NN) y en una máquina de estados (FSM).

A medida que incorporábamos los bots, también fuimos modificando y mejorando el framework. El objetivo era conseguir que añadir una nueva implementación no requiriese modificar la lógica principal de la partida. De esta forma, cada nuevo bot no sólo ampliaba las capacidades de la aplicación, sino que también nos permitía comprobar y mejorar la flexibilidad de la aplicación.

Como resultado obtuvimos una primera versión funcional del framework capaz de integrar diferentes tipos de bots y permitirles participar en las mismas partidas de póker.

1.3.4 Fase III: Desarrollo de la interfaz gráfica

Hasta este momento, gran parte de las pruebas se habían realizado directamente sobre la lógica del juego, por lo que necesitábamos disponer de una interfaz que permitiera utilizar las funcionalidades desarrolladas. Antes de comenzar con la implementación dedicamos varias semanas al diseño de la interfaz. Alrededor del mes de noviembre, cada integrante realizó sus propuestas y lo unificamos comparando ideas y decidiendo qué elementos utilizar.

Una vez definido el diseño común, comenzamos a desarrollar los contenedores y elementos básicos utilizando *JavaFX*. El desarrollo de interfaz fue en realidad un trabajo progresivo, se iban incorporando las diferentes vistas paralelamente al desarrollo de las implementaciones, realizando diferentes pruebas para comprobar las acciones realizadas y haciendo cambios necesarios. Alrededor del mes de abril teníamos ya la versión final de la interfaz, que fue mostrada al tutor, revisada y puesta en prueba para detectar posibles aspectos que debían modificarse.

1.3.5 Fase IV: Desarrollo de la arquitectura distribuida

El objetivo principal de esta cuarta fase fue organizar la aplicación mediante una arquitectura distribuida que permitiera separar las responsabilidades de sus diferentes componentes. Desde las primeras etapas planteamos esta división, por lo que no esperamos a finalizar el desarrollo para separar la aplicación, sino que fuimos incorporando nuevas funcionalidades con esta arquitectura de base.

La aplicación se organizó en tres partes principales: cliente, servidor y módulo común. Esta separación se fue realizando de forma paralela al resto del desarrollo. Cada vez que añadimos una funcionalidad, determinábamos o qué parte correspondía con el servidor, cual al cliente y qué información debía compartirse entre ambos. Uno de los integrantes tenía más experiencia que el resto en este tipo de arquitecturas, por lo que asumió el rol de colaborador principal. Durante este proceso realizamos pruebas de comunicación para detectar problemas y realizar modificaciones, en lugar de esperar a terminar toda la aplicación.

De esta manera, conseguimos una organización en la que el cliente y del servidor podían trabajar de forma independiente, separando la lógica del juego de

la interfaz. Una vez definido esto completamente en el mes de febrero, el desarrollo fue avanzando con muchísima más velocidad.

Para abril de 2026 teníamos como resultado una versión final completamente funcional y distribuida, capaz de ejecutar partidas de póker e integrar los diferentes tipos de bots desarrollados.

1.3.6 Fase V: Experimentación y evaluación

El objetivo principal de esta quinta fase fue realizar las pruebas necesarias para evaluar la integración y el funcionamiento de los diferentes bots y comprobar la lógica de la aplicación. Antes de comenzar con las pruebas dedicamos un tiempo, junto con el tutor, a definir los escenarios que queríamos analizar y a determinar cómo debíamos organizar las partidas para que los resultados obtenidos fueran útiles.

Una parte importante del trabajo consistió en buscar personas que pudieran participar en las pruebas, con distintos niveles de agilidad en el póker, ya que queríamos disponer de suficientes partidas y situaciones para que las estadísticas obtenidas fueran representativas. Para ello fue necesario coordinar los horarios de los participantes y adaptar las sesiones para que pudieran llevarse a acabo. Esta organización hizo que las pruebas se extendieran durante varios meses.

Las sesiones de experimentación se hicieron principalmente durante junio, julio y agosto de 2026. Durante este periodo fuimos ejecutando partidas y recopilando los datos necesarios para posteriormente analizar el comportamiento de los bots, ajustando los errores que iban surgiendo o los cambios necesarios.

No se trataba únicamente de comprobar que la aplicación funcionara, sino obtener información suficiente para poder estudiar las diferencias entre los bots, valorar el comportamiento de las distintas estrategias y la lógica detrás. Dedicamos especial atención a mantener unas condiciones de prueba suficientemente consistentes, y esto fue la base para estudiar el comportamiento y extraer las conclusiones del proyecto.

1.3.7 Fase VI: Documentación y elaboración de la memoria

El objetivo de esta última fase fue recoger y organizar todo el trabajo realizado durante el desarrollo del proyecto y elaborar la memoria del TFG. Aunque la redacción final se concretó principalmente en los últimos meses, la documentación se fue creando desde el comienzo del proyecto. Durante el desarrollo mantuvimos un historial de las reuniones, las tareas, decisiones tomadas, los problemas encontrados y que posteriormente nos permitió recuperar la evolución del proyecto y utilizarla como base para redactar la memoria. Además, desde las primeras etapas teníamos planteada la estructura que tendría la memoria, por lo que al comenzar su elaboración ya teníamos una idea del contenido que debíamos desarrollar.

A partir de junio de 2026 comenzamos a dedicar una parte del trabajo a la elaboración de la memoria. Durante las semanas siguientes asignamos el contenido, redactamos los diferentes apartados y revisamos la documentación para comprobar que reflejaba correctamente el desarrollo realizado.

Para mediados de agosto la mayor parte de la memoria ya estaba redactada. En ese momento, uno de los elementos que todavía quedaba por completar era la incorporación de los resultados obtenidos durante la experimentación, las estadísticas correspondientes y su posterior análisis. Estas se fueron añadiendo una vez finalizadas las pruebas de forma que la memoria recopilara todos los datos definitivos.

De esta forma, el trabajo realizado durante esta fase se centró en completar y documentar el proyecto, incorporando los datos de la experimentación.

1.4 Estructura del documento

El resto del contenido de este documento está organizado en los siguientes capítulos:

- Capítulo 2: Contextualización. Este capítulo presenta los conceptos teóricos y tecnológicos necesarios para entender el entorno del proyecto.
- Capítulo 3: Diseño. En este capítulo se describen las diferentes partes de la aplicación.

- Capítulo 4: Implementación. En este capítulo se detalla la implementación de dichas partes comentadas en el diseño.
- Capítulo 5: Estudio empírico. Expone la configuración experimental, las pruebas realizadas y el análisis de resultados obtenidos a partir de los diferentes escenarios a los que se han enfrentado los bots.
- Capítulo 6: Conclusiones y trabajo futuro. El último capítulo presenta las principales conclusiones derivadas del trabajo realizado y las posibles líneas de desarrollo a futuro.

Todo el trabajo realizado para este proyecto (código, pruebas, archivo de imágenes, etc.) está almacenado en el siguiente repositorio de GitHub.

Capítulo 2 - Contextualización

En este capítulo se explica qué es y cómo se juega al póker, y se exponen las diferentes técnicas de inteligencia artificial empleadas, así como sus limitaciones, eficacia y ventajas e inconvenientes.

2.1 Póker NLHE

Para entender la historia del póker y qué es, es necesario estudiar sus antepasados y los juegos que lo precedieron. Estos juegos dieron forma al póker tal como lo conocemos hoy. Esto es lo que se detalla en este apartado.

2.1.1 Historia y definición del póker

Varios historiadores creen que las raíces del póker surgen de diferentes juegos de cartas que evolucionaron a lo largo de los siglos en diferentes regiones del mundo. Entre sus posibles antecedentes destaca el "As-Nas", un juego persa que surgió en el siglo XVI compuesto por una baraja de 25 cartas e incluía rondas de apuestas y elementos de farol, similar al póker moderno. Más tarde, en el siglo XVII, los franceses introdujeron el "poque", un juego de 52 cartas que incluía elementos de faroles y apuestas, y que fue llevado a América del Norte por los colonos franceses.[2]

El póker moderno empezó a formarse en los Estados Unidos durante el siglo XIX. La ciudad de Nueva Orleans desempeñó un papel fundamental en su difusión y acontecimientos como la Guerra Civil ayudaron a extender su popularidad, dando lugar a nuevas variantes y a evolucionar hasta diferenciarse claramente de sus predecesores.[17]

El siglo XX fue un periodo decisivo para la consolidación del póker. En esta época se establecieron las reglas de manera uniforme, se crearon las WSOP (World Series of Poker) y el Texas Hold'em se volvió muy popular. Además, la influencia de Las Vegas y la creciente presencia del póker en los medios y la cultura favorecieron su expansión internacional.[2][17]

A principios del siglo XXI, el póker experimentó un crecimiento sin precedentes gracias al llamado "efecto Moneymaker" y al auge del póker online, impulsado por internet y los avances tecnológicos. Más recientemente, la incorporación de la

inteligencia artificial ha transformado tanto el estudio como la práctica del juego mediante técnicas de aprendizaje automático, teoría de juegos y análisis masivo de datos. Esta evolución ofrece nuevas herramientas para el estudio estratégico, aunque también plantea desafíos éticos relacionados con el uso de bots o herramientas de asistencia en tiempo real, lo que ha llevado a las salas de póker a desarrollar también sistemas de inteligencia artificial orientados a la detección de fraude y protección de la integridad competitiva.[4][5][17]

En conclusión, el póker es uno de los juegos de cartas más practicados en todo el mundo, y aunque presenta reglas básicas simples, posee una gran profundidad estratégica y conceptual, basada en la combinación de apuestas, toma de decisiones y gestión del riesgo, como dice David Sklansky en su libro "*La teoría del póker*": « La belleza del póquer es que en la superficie es un juego de absoluta simplicidad, pero debajo de la superficie es profundo, rico y lleno de sutileza »[6]. Se caracteriza principalmente como un juego de habilidad más que de azar y en el que los jugadores utilizan decisiones estratégicas para maximizar sus ganancias y minimizar sus pérdidas.[6]

2.1.2 Reglas del póker NLHE

En el póker No-Limit Texas Hold'em, cada jugador recibe dos cartas privadas, que junto a las otras cinco cartas que se colocarán en la mesa, denominadas cartas *comunitarias*, permitirán al jugador obtener la mejor mano posible con cualquier conjunto de cinco cartas de las siete que tendrá disponible, como máximo. Existe un orden en las manos de cinco cartas que determina qué mano es superior a otras que podemos ver representado en la Figura 2-1, donde la mano con un valor numérico más pequeño es superior a las que tengan un valor numérico mayor.

La metodología del juego se basa en varias rondas en las que los jugadores actúan por turnos y donde al comienzo de cada una se colocan boca arriba cartas de la baraja en el centro de la mesa.



Figura 2-1. Reglas específicas para la clasificación de las manos

Al final de la última ronda, el o los jugadores que con la mejor combinación de cartas que se mantuvo en juego se llevan el total apostado. Dichas rondas de apuestas se denominan:

- Preflop: se reparten dos cartas boca abajo a cada jugador (pocket cards o hole cards). Son las únicas cartas que cada jugador recibe individualmente y sólo son descubiertas al final de la mano o si todos los jugadores apuestan all-in. La mano empieza con el primer jugador situado a la izquierda del big blind, y debe igualar la cantidad del big blind, subir para aumentar la apuesta o retirarse y descartar su mano. Los demás jugadores tienen las mismas tres opciones: igualar, subir o retirarse.[8]
- Flop: comienza con el reparto de las tres primeras cartas descubiertas y comunitarias (después de "quemar" y retirar una carta). Los jugadores tienen las mismas opciones para decidir qué acción realizar, junto con "pasar".
- Turn: se descubre la cuarta carta comunitaria. Una vez más, los jugadores pueden pasar, apostar, igualar, subir o retirarse.
- River: después del turn, se saca otra carta descubierta comunitaria, y es seguido de otra ronda de apuestas (Fifth street).
- Showdown: finalmente, los jugadores que siguen activos deben mostrar o "declarar" sus manos, el jugador o jugadores con la mejor mano o manos ganan

el bote. Los jugadores suelen mostrar sus manos en orden en lugar de hacerlo todos a la vez. Si los jugadores tienen la misma mano, el bote se reparte. El bote se divide de distintas maneras en función de las reglas del juego y de la valoración de la mano de cada jugador en comparación con la de sus rivales.[8]

Si solo queda un jugador antes de la confrontación final, dicho jugador gana el bote automáticamente y no es necesario que descubra sus cartas.

2.1.3 Glosario del póker

En este glosario de póker, según el libro "Professional No-Limit Hold'em Volumen I" de Matt Flynn, Sunny Mehta y Ed Miller[7] se explican los términos más relevantes que se suelen utilizar en el juego de NLHE, así como sus traducciones al español. Algunos términos no tienen traducción por lo que se han dejado tal como son en inglés.

- Bet (apostar): Cuando te llega tu turno y tienes la opción de apostar.
- Small blind (ciega pequeña): Es una apuesta obligatoria que en cada mano que se reparte debe efectuar el jugador que esté a la izquierda del Dealer.
- Big blind (ciega grande): Es una apuesta obligatoria que en cada mano tiene que efectuar el jugador que está dos sitios a la izquierda del dealer.
- Bluff (farol): Apostar sin tener una jugada hecha y por lo tanto sin posibilidades de ganar la mano si otro jugador ve la apuesta.
- Button (botón): Ficha de plástico bastante más grande que las de apostar y con una D en el centro. Esta ficha marca la posición del dealer.
- Community Cards (cartas comunitarias): En Texas Hold'em, Omaha y otras variantes del póker, son las cartas que se muestran boca arriba en el centro de la mesa y que todos los jugadores pueden usar para hacer su jugada.
- Dealer (repartidor): Jugador que hace las veces de dealer virtual y desde cuya posición se reparten las cartas.
- Heads up (mano a mano): Torneo o partida de tan sólo dos jugadores.
- No-limit Póker (póker sin límite): Modalidad de póker en la que se puede apostar cualquier cantidad de fichas o dinero sin ningún tipo de límite.
- Play money (dinero ficticio): Las salas de póker en línea casi siempre ofrecen la posibilidad de jugar con dinero ficticio para que puedas practicar.

- Pocket Cards/Hole Card (cartas de mano): Las cartas que se dan a los jugadores al principio de la mano y que sólo ellos pueden ver.
- Pot (bote): El bote es el total de fichas o dinero que se ha apostado en una mano.

2.2 Técnicas de IA para jugar al póker

Gran parte de los conceptos teóricos presentados en esta sección se basan en la documentación técnica de IBM.

El mundo del póker online ha evolucionado de ser un simple juego de cartas a un complejo campo de batalla estratégico. Hoy, un nuevo jugador se sienta en la mesa virtual: la inteligencia artificial. En este apartado se indican las diferentes técnicas de IA para poder interpretar patrones de apuestas, estimar qué cartas puede tener el rival, decidir cuándo apostar, igualar o retirarse, y equilibrar sus propias decisiones para no ser predecible.

2.2.1 Teoría de juegos

La teoría de juegos es una rama de las matemáticas que estudia situaciones en las que varios participantes toman decisiones estratégicas y el resultado depende de las acciones de todos. Su objetivo es analizar cómo interactúan distintos agentes cuando persiguen objetivos que pueden ser comunes, opuestos o parcialmente coincidentes.[11]

Cada agente dispone de un conjunto de posibles acciones o estrategias y trata de elegir aquellas que le proporcione el mayor beneficio posible. Sin embargo, la complejidad surge porque el éxito de una estrategia depende del comportamiento de los demás agentes involucrados. Por este motivo, la teoría de juegos proporciona un marco formal para estudiar problemas de cooperación, competencia, negociación y coordinación entre múltiples participantes.[9]

En el póker, cada jugador intenta maximizar sus ganancias, pero sus decisiones influyen en las opciones futuras y en el comportamiento de los demás, generando un entorno dinámico donde la planificación y la adaptación resultan fundamentales.[9]

Una de las ideas más importantes de esta técnica aplicada al póker es la estrategia GTO (Game Theory Optimal), su objetivo es que, juegues contra quien juegues, tu estrategia no pueda ser derrotada sistemáticamente, ya que se basa en

un punto de equilibrio donde ningún rival puede obtener ventajas modificando unilateralmente su forma de jugar. Esta lógica deriva directamente del *equilibrio de Nash*, otro concepto fundamental en teoría de juegos para juegos con dos o más jugadores, como el póker, que se produce cuando las estrategias seleccionadas por todos los jugadores constituyen respuestas óptimas entre sí. Esto significa que cada participante maximiza su beneficio esperado dadas las decisiones de los demás, de modo que ningún jugador obtiene ventajas al cambiar individualmente de estrategia. Es importante tener en cuenta que un equilibrio de Nash no implica que se logre el mejor resultado conjunto para los participantes, sino solo el mejor resultado para cada uno de ellos considerados individualmente.[10]

El mejor ejemplo de un equilibrio de Nash es la variación del conocido "*Dilema del prisionero*", en esta versión muestra cómo varios jugadores pueden terminar tomando decisiones que no son las más beneficiosas para ellos, pudiendo obtener mejor resultado si todos cooperaran y no declararan, pero, dado que cada cual persigue su interés, y ninguno puede confiar en que nadie declarará, todos deben adoptar la estrategia de declarar, lo que termina en una situación (equilibrio) en la cual uno minimiza su posible pérdida.[10]

2.2.2 Aprendizaje por refuerzo

En el campo de la IA, el machine learning constituye la columna vertebral de la mayoría de los sistemas modernos, desde modelos de previsión hasta vehículos autónomos, pasando por modelos de lenguaje de gran tamaño (LLM) y otras técnicas de IA generativa. Se define como el subconjunto de la inteligencia artificial centrada en algoritmos que pueden aprender los patrones de los datos de entrenamiento y, posteriormente, hacer inferencias precisas sobre nuevos datos. Esta capacidad de reconocimiento de patrones permite a los modelos de machine learning tomar decisiones o realizar predicciones sin instrucciones explícitas y codificadas de forma rígida.[12]

El aprendizaje por refuerzo (RL) es un tipo de proceso de machine learning en el que los agentes autónomos aprenden a tomar decisiones interactuando con su entorno.[13]

Un agente autónomo es cualquier sistema que puede tomar decisiones y actuar en respuesta a su entorno independientemente de las interacciones directas de un usuario humano. Dos ejemplos de agentes autónomos son los robots y los coches automáticos.

En el aprendizaje por refuerzo los agentes autónomos aprenden a realizar una tarea por ensayo y error en ausencia de cualquier orientación de un usuario humano. Aborda principalmente los problemas de toma de decisiones secuenciales en entornos inciertos, como el póker.

Entonces, el aprendizaje por refuerzo consiste esencialmente en la relación entre un agente, el entorno y el objetivo. Esta dinámica se presenta comúnmente mediante el proceso de decisión de Markov (MDP). [13]

2.2.2.1 Proceso de decisión de Markov

El agente por refuerzo aprende sobre un problema al interactuar con su entorno. El entorno proporciona información sobre su estado actual. A continuación, el agente utiliza esa información para determinar qué acciones tomar. Si esa acción obtiene una señal de recompensa del entorno, se anima al agente a repetirla cuando se encuentre en un estado a futuro similar. Este proceso se repite para cada nuevo estado a partir de entonces. Con el tiempo, el agente aprenderá debido a un refuerzo positivo mediante recompensas y penalizaciones a la hora de tomar decisiones dentro del entorno que cumplan un objetivo específico.[13] Este proceso se puede visualizar en la Figura 2-2.

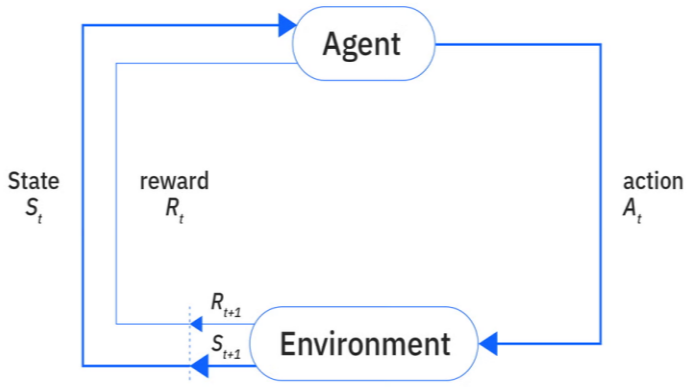


Figura 2-2. Proceso de decisión de Markov. Fuente: IBM [13]

En estos procesos, el *espacio de estados* se refiere a toda la información proporcionada por el estado de un entorno. El *estado de acción* se refiere a todas las acciones posibles que un agente puede tomar dentro de un estado.

Además, de la relación entre agente, entorno y objetivo, los problemas de aprendizaje por refuerzo se definen por cuatro componentes fundamentales adicionales:

- Política: define el comportamiento del agente RL mapeando los estados ambientales percibidos a acciones específicas que el agente debe tomar cuando se encuentran en esos estados. Puede adoptar la forma de una función rudimentaria o de un proceso computacional más complejo.
- Señal de recompensa: designa el objetivo del problema de RL. Cada una de las acciones del agente RL recibe o no una recompensa del entorno.
- Función del valor: especifica un beneficio a largo plazo. Se refiere a la conveniencia de un estado por todos los estados que probablemente le sigan.
- Modelo: este es opcional de los sistemas de aprendizaje RL. Los modelos permiten a los agentes predecir el comportamiento del entorno para posibles acciones. A continuación, utilizan estas predicciones para determinar posibles líneas de actuación en función de los resultados potenciales.

  El aprendizaje por refuerzo es una disciplina en continua expansión que ha propiciado el surgimiento de múltiples enfoques de aprendizaje. No obstante, tres metodologías se consideran pilares fundamentales en esta área de investigación: la programación dinámica y el Monte Carlo.

- Programación dinámica: descompone las tareas más grandes en tareas más pequeñas. De esta manera, los problemas se modelan como procesos de decisiones secuenciales en los que, en cada paso temporal, el agente selecciona una acción considerando los posibles estados futuros. Podemos observar en la ecuación de la Figura 2-3 que la recompensa de un agente (r) por una acción determinada se define en función de esa acción (a), de los estados ambientales actuales y de los posibles siguientes estados s':

$$r_t(s,a) = \sum_{s' \in S} r_t(s,a,s') p_t(s'|s,a)$$

Figura 2-3. Función de recompensa. Fuente: IBM [13]

La función de recompensa proporciona información esencial para determinar cómo debe actuar un agente dentro de su entorno. En consecuencia, gran parte de los métodos de programación dinámica se centran en encontrar la política óptima.

Para ello, se utiliza la ecuación de Bellman, representada en la Figura 2-4

$$v_t(s) = \max_{a \in A(s)} \{ r_t(s,a) + \sum_{s' \in S} p(s'|s,a) v_{t+1}(s') \}$$

Figura 2-4. Ecuacion de Bellman. Fuente: IBM

En términos generales, esta ecuación expresa *vt(s)* como la recompensa acumulada esperada que un agente puede obtener desde un estado *s* en el instante *t* hasta la finalización del proceso de decisión. Para ello, descompone esta recompensa total en dos componentes: la recompensa inmediata *rt(s, a)* y la recompensa total esperada del agente. De esta forma, el agente puede identificar la acción que maximiza el valor esperado en cada estado y, por tanto, optimizar su comportamiento.

Para entender este método aplicado al póker. Cada jugador cuando realiza una acción ya sea *fold*, *call*, *check*, *raise* o *all-in*, debe pensar no solo en la recompensa inmediata que puede obtener, sino también cómo afectará a las situaciones futuras de la partida. Según el principio de Bellman, el valor de tomar una decisión u otra no viene únicamente determinado por la recompensa que proporciona, también depende del valor esperado de los estados futuros que pueden darse como consecuencia de dicha decisión. Por lo que, una decisión que aparentemente no sea favorable puede resultar óptima si aumenta la probabilidad de obtener beneficios en las rondas posteriores.

2.2.2.2 Método Monte Carlo

A diferencia de la programación dinámica que requiere disponer de un modelo del entorno para estimar transiciones entre estados y recompensas futuras los métodos de Monte Carlo operan sin dicho conocimiento previo. Estos métodos consideran el entorno como una caja negra y aprenden exclusivamente a partir de la experiencia obtenida durante la interacción directa con él. [13]

En lugar de predecir los estados futuros mediante un modelo, los algoritmos de Monte Carlo recopilan muestras de secuencias de estados, acciones y recompensas observadas durante la ejecución. Por ello, su aprendizaje se fundamenta en la experiencia y el ensayo y error, más que en el uso de distribuciones de probabilidad conocidas.

Asimismo, ambos enfoques difieren en la forma de estimar las funciones de valor. Mientras que la programación dinámica calcula el valor de los estados utilizando estimaciones recursivas basadas en el modelo de entorno, el método de Monte Carlo evalúa cada para estado-acción a partir del promedio de los retornos realmente obtenidos en episodios anteriores. Como consecuencia, estos métodos necesitan que un episodio finalice antes de poder calcular el retorno acumulado, actualizar la función de valor y justificar la política de actuación del agente.

Usar este método para partidas de póker no es recomendable ya que llevaría demasiado tiempo recopilar muestras de todos los estados, y tampoco se conseguiría una simulación realista teniendo en cuenta la gran cantidad de movimientos permitidos.

2.2.3 Redes Neuronales

La red neuronal es otro tipo de modelo de machine learning basado en el funcionamiento del cerebro humano, formados por capas de neuronas artificiales, que ajustan sus parámetros durante el entrenamiento para identificar patrones a partir de datos para asignar entradas a outputs.[14]

Las redes neuronales se han consolidado como uno de los enfoques más influyentes en la inteligencia artificial moderna. Han impulsado avances significativos en campos como la visión artificial, el procesamiento del lenguaje natural (PLN), el reconocimiento de voz y otros sistemas inteligentes. A pesar de la sofisticación de

arquitecturas actuales como los transformadores o las redes neuronales convolucionales (CNN), los orígenes de las redes neuronales se remontan a modelos simples como la regresión lineal *(Perceptrón, 1958, Frank Rosenblatt)*[14] y cómo el cerebro humano digiere, procesa y decide sobre la información presentada a ello.

El funcionamiento de una red neuronal puede entenderse como un proceso de transformación de datos de entrada en una predicción o decisión. Por ejemplo, para detectar correos no deseados, la red analiza características como determinadas palabras o expresiones presentes en el mensaje. A medida que la información atraviesa las distintas capas de neuronas, se identifican patrones cada vez más complejos hasta estimar la probabilidad de que el correo sea spam. Su capacidad de aprendizaje depende del ajuste de pesos y sesgos, parámetros que determinan la relevancia de cada entrada y que se optimizan durante el entrenamiento para generar predicciones precisas. Una red neuronal se compone de varias capas, reflejadas en la Figura 2-5:

- Capa de entrada: contiene las características sin procesar.
- Capas ocultas: consisten en neuronas artificiales o nodos encargadas de transformar las entradas en nuevas representaciones. En ellas, cada entrada se combina con un conjunto de pesos y sesgos para pasar de una capa a la siguiente, llegando finalmente a la capa de output final. Aquí es donde ocurre la transformación lineal entre entrada y output.
- Capa de output: tras aplicar la transformación lineal, se incorpora una función de activación no lineal (tanh, sigmoide o ReLU) para obtener la salida final, que puede representar tanto una estimación numérica como una distribución de probabilidades para la clasificación.

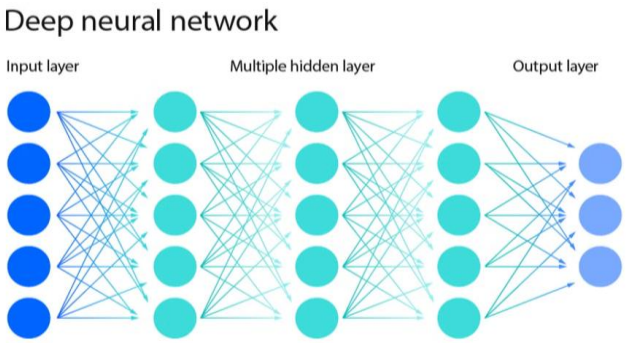


Figura 2-5. Red neuronal feedforward estándar con 3 capas ocultas. Fuente: IBM.[14]

El correcto funcionamiento de una red neuronal depende de un proceso de entrenamiento previo. Para entrenar una red, una sola neurona calcula:

$$\boldsymbol{z} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{w_i x_i + b}$$

$$a = \sigma(z)$$

Donde:

$\boldsymbol{x_i}$: característica de entrada $\boldsymbol{a}$: output

w_i : peso $\boldsymbol{b}$: sesgo

$\boldsymbol{z}$ **:** suma ponderada (transformación lineal)

$\boldsymbol{\sigma}$: función de activación (transformación no lineal)

La función de activación ($\boldsymbol{\sigma}$) en la capa de output transforma la combinación lineal para ajustarla a la decisión de la función. A continuación, las características de entrada X se transforman en un output Y, que sirve como un modelo predictivo de machine learning.[14]

La capacidad de aprendizaje de una red neuronal radica en la optimización de pesos y sesgos. Para ello, las predicciones generadas por el modelo $\hat{Y}$ se compara con los valores reales Y mediante una función de pérdida, que cuantifica el error cometido.

La red neuronal se entrena en cuatro pasos:

- Pasada hacia adelante: los datos de entrada atraviesan la red, se calculan combinaciones lineales, se pasa por la función de activación no lineal y se produce una predicción de output.
- Cálculo de error: la función de pérdida calcula la discrepancia entre los valores estimados y los valores reales.
- Pasada hacía atrás (retropropagación): el error se propaga en sentido inverso a través de la red neuronal. Durante este proceso, se determina el impacto que cada peso y sesgo ha tenido en el error mediante la regla de la cadena de cálculo.

- Actualización de ponderación: los pesos y los sesgos se corrigen en la dirección que produce una disminución del error. Este proceso suele llevarse a cabo mediante técnicas de optimización, como el descenso de gradiente que se encuentra en la Figura 2-6.

Este proceso se repite de manera iterativa sobre los datos de entrenamiento disponibles, permitiendo que la red neuronal ajuste progresivamente sus parámetros internos para mejorar su capacidad de predicción.

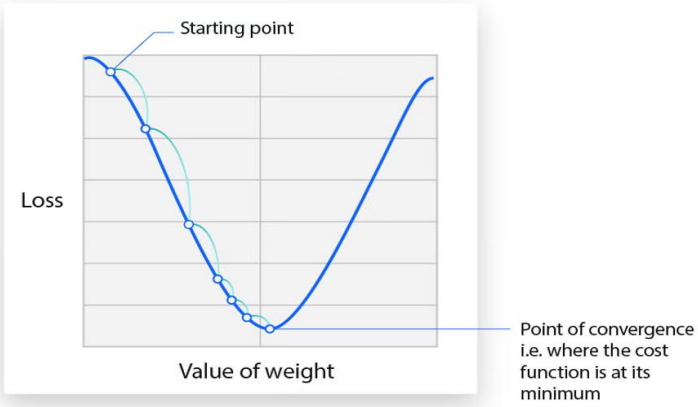


Figura 2-6. Función de coste. Fuente: IBM

No obstante, las redes neuronales también presentan limitaciones, siendo el sobreajuste uno de los problemas más frecuentes. Este fenómeno ocurre cuando el modelo se vuelve demasiado complejo con demasiados parámetros perdiendo la capacidad de predecir bien.

Con el tiempo las redes neuronales han evolucionado hacia arquitecturas especializadas orientadas a resolver problemas concretos:[14]

- Redes neuronales convolucionales (CNN o ConvNets): diseñadas para procesar datos en forma de cuadrícula, como imágenes.
- Redes neuronales concurrentes: incorporan mecanismos de memoria que permiten aprovechar información de etapas anteriores del procesamiento.
- Transformadores: a diferencia de las RNN, utilizan mecanismos de atención para modelar relaciones entre diferentes elementos de una secuencia, lo que ha revolucionado el campo del procesamiento del lenguaje natural y ha dado lugar a modelos de gran capacidad como GPT.

En el póker, procesa el estado actual del juego, aplica los patrones aprendidos en el entrenamiento, construye un modelo específico del oponente en tiempo real y produce una decisión en tiempo récord.

2.2.4 LLM (Large Language Model)

A raíz del machine learning, nace un subconjunto llamado deep learning impulsado por redes neuronales cuyo diseño se basa en la estructura del cerebro humano.[15]

A diferencia de muchos algoritmos tradicionales de aprendizaje automático, las redes neuronales de los modelos de deep learning están compuestas por múltiples capas de neuronas artificiales interconectadas en las que cada una realiza operaciones matemáticas. Es importante recalcar que "deep learning" no es sinónimo de red neuronal, el primero se refiere al entrenamiento de modelos con al menos cuatro capas.

Los modelos de deep learning suelen entrenarse mediante el aprendizaje supervisado, que es un paradigma de machine learning que consiste en entrenar un modelo con datos etiquetados, es decir, datos identificados sin procesar, para que aprenda patrones y pueda predecir correctamente resultados futuros. Sin embargo, como las redes neuronales suelen requerir grandes cantidades de datos de entrenamiento, se desarrolló una técnica que replica tareas de aprendizaje supervisado utilizando datos no etiquetados, denominada *aprendizaje autosupervisado (2010, por Yann LeCun)*.[15]

Los modelos de lenguaje de gran tamaño (LLM) son sistemas de inteligencia artificial basados en técnicas de deep learning y arquitecturas de transformadores. Estos modelos se entrenan con enormes cantidades de texto procedente de libros, páginas web, artículos, código fuente y otras fuentes de información, lo que les permite comprender y generar lenguaje con un alto grado de precisión.[16]

Su funcionamiento se basa en aprender patrones estadísticos presentes en los datos para predecir la palabra o secuencia de palabras más probable en un determinado contexto. Gracias a este proceso, son capaces de realizar tareas como responder preguntas, resumir información, traducir textos, generar contenido o mantener conversaciones en lenguaje natural.

El entrenamiento de un LLM implica procesar grandes volúmenes de información, dividir el texto en unidades denominadas *tokens* y ajustar millones o incluso miles de millones de parámetros mediante técnicas de optimización y retropropagación. Como resultado, el modelo aprende relaciones semánticas, estructuras gramaticales y patrones de razonamiento presentes en los datos de entrenamiento.

Una vez entrenados, estos modelos pueden adaptarse a dominios o tareas específicas mediante procesos de ajuste adicional *(fine-tunning)* o a través de técnicas de ingeniería de *prompts*. Entre los LLM más conocidos se encuentran GPT, Gemini, Llama, Copilot y Claude, que han transformado la forma en la que las personas interactúan con los sistemas de inteligencia artificial.[16]

En el contexto del proyecto, los LLM se emplean como agentes capaces de participar en partidas de póker. Para ello, reciben información sobre el estado actual de la partida, como las cartas disponibles, las apuestas realizadas, la posición de los jugadores o el historial de acciones, y generan una decisión en lenguaje natural. En la implementación desarrollada hemos integrado dos modelos de lenguaje, Llama y Gemini, permitiendo comparar su comportamiento frente a otros tipos de bots en la aplicación.

2.2.5 Counterfactual Regret Minimization (CFR)

Es un algoritmo importante de aprendizaje automático para juegos de "información imperfecta", como el póker, donde parte de la información sobre el estado del juego se oculta a los jugadores, pero las reglas y los objetivos son conocidos. La idea principal es que la IA analiza decisiones pasadas y calcula cuánto habría ganado o perdido si hubiera elegido otra acción. Ese "arrepentimiento" se utiliza para mejorar futuras decisiones. Su funcionamiento se resume en:[18][19]

1. Ejecutar el agente contra sí mismo durante muchas iteraciones (algoritmo de autojuego).
2. En cada conjunto de información, calcular la estrategia utilizando los arrepentimientos acumulados actuales.
3. Calcular recursivamente la utilidad de todas las acciones posibles y actualizar el arrepentimiento acumulado.

4. La estrategia promedio a lo largo de todas las iteraciones en cada nodo converge hacia un equilibrio de Nash.

Se ha usado en sistemas avanzados como: Libratus (CMU,2017)[3], una IA que jugaba Heads-up No-Limit Texas Hold'em que derrotó a cuatro top profesionales en el evento "cerebros contra IA", duró 20 días y generó alrededor de 120000 manos jugadas; y Pluribus (Science, 2019)[3], primera IA en vencer a profesionales humanos en el No-Limit Hold'em de seis jugadores.

Capítulo 3 - Diseño

En este capítulo se presenta el diseño de la aplicación, partiendo de la estructura necesaria para gestionar una partida de No-Limit Texas Hold'em y llegando hasta la integración de los diferentes tipos de bots. También se muestra cómo se han organizado los distintos componentes de la aplicación para que puedan trabajar conjuntamente y mantener una estructura flexible y extensible.

3.1 GUI

La interfaz gráfica de usuario (GUI) ha sido diseñada con el objetivo de ofrecer una experiencia intuitiva y sencilla durante todas las fases de una partida de No-Limit Hold'em, desde su configuración inicial hasta su finalización. Para ello, la aplicación organiza la navegación mediante un conjunto de ventanas conectadas entre sí que guían al usuario a través de los distintos pasos necesarios para crear una partida, unirse a una existente, esperar a jugar o participar como espectador. La distribución de los elementos visuales y la secuencia de navegación han sido concebidas para que cualquier usuario, independientemente de su experiencia previa con la aplicación, pueda comprender rápidamente el funcionamiento general del sistema.

La navegación principal de la aplicación se divide en dos grandes flujos. El primero permite crear una partida nueva, configurando progresivamente sus principales parámetros, mientras que el segundo posibilita unirse a una partida ya existente mediante conexión a la máquina desde donde se ha creado la partida. Esta organización por etapas evita sobrecargar al usuario con demasiadas opciones simultáneamente y facilita la comprensión del proceso de configuración.

Dentro del flujo de creación de partidas, se han diseñado diferentes ventanas especializadas para configurar aspectos concretos del juego, como los parámetros generales de la partida, los jugadores participantes, la mesa seleccionada o los bots que actuarán como otros jugadores. Entre todas ellas, la ventana de agregación de bots, que se muestra en la Figura 3-1, presenta un comportamiento dinámico, ya que adapta automáticamente su contenido a los bots disponibles en la aplicación. Como resultado, cualquier bot incorporado al sistema aparece automáticamente en esta

pantalla sin necesidad de modificar manualmente el diseño de la interfaz, garantizando así la escalabilidad y extensibilidad de la plataforma.



Figura 3-1. Diseño de la venta de agregación de bots a una partida

Cada bot disponible se representa mediante una carta visual que reúne toda la información relevante para el usuario. Entre los datos mostrados se incluye el nombre del bot, una descripción resumida de sus características, una representación gráfica identificativa y los mecanismos necesarios para que permita estilos de jugador o no. En el caso de que el bot, sí deje incorporarlos, la interfaz configura la distribución de personalidades disponibles, como *loose-aggressive*, *loose-passive*, *tight-aggressive*, *tight-passive*, *maniac* o *default*. Esta diferencia permite adaptar el comportamiento de cada bot sin necesidad de modificar su implementación interna.

La pantalla principal de juego, mostrada en la Figura 3-2, constituye el núcleo visual de la aplicación. Su diseño replica la disposición habitual de una mesa de póker real, mostrando las posiciones fijas de los jugadores, las cartas comunitarias, el bote acumulado, las fichas de cada participante y las distintas acciones realizadas durante la partida. Además, determinadas acciones relevantes, como la aparición progresiva de cartas comunitarias o la actualización de información de los jugadores, se representan mediante elementos visuales y animaciones que facilitan el seguimiento de la partida y mejoran la experiencia de usuario.



Figura 3-2. Diseño de la ventana de juego

Durante el desarrollo de una partida, la interfaz permanece sincronizada con el estado del juego, actualizando automáticamente la información mostrada conforme evolucionan las manos. De esta forma, el usuario puede conocer en todo momento la situación actual de la mesa, las acciones realizadas por cada participante y los resultados obtenidos tras cada ronda de apuestas. El diseño prioriza la claridad visual y la accesibilidad de la información para facilitar la toma de decisiones y el seguimiento del desarrollo del juego.

La aplicación incorpora además un "*modo espectador*", una funcionalidad que permite crear una partida y observar posteriormente su desarrollo sin participar activamente en ella. Este modo se ha creado con la intención que sirva de herramienta de análisis y depuración, permitiendo estudiar el comportamiento de los distintos bots, detectar posibles errores o incoherencias y recopilar información sobre sus estrategias. No obstante, también puede emplearse con fines educativos, ya que ofrece una visión completa de las decisiones tomadas por todos los participantes de la sala.

Con el fin de mantener una experiencia visual coherente, todas las ventanas de la aplicación comparten una misma línea estética basadas en componentes reutilizables y criterios de diseño homogéneos. Esta consistencia visual reduce la curva de aprendizaje para el usuario, facilita la navegación entre pantallas y contribuye a

que la aplicación presente una identidad visual unificada. El diseño general fue realizado previamente mediante bocetos y prototipos, permitiendo definir de forma progresiva la apariencia final de la interfaz antes de proceder a su implementación.

Por último, la aplicación genera automáticamente una representación visual única para cada participante de la partida. Tanto los jugadores humanos como los bots disponen de un avatar identificativo que permite distinguirlos rápidamente durante el juego, reforzando la claridad visual de la mesa y facilitando el seguimiento de la acción por parte de todos los usuarios.

3.2 Componentes de una partida de póker

A continuación, en la Figura 3-3 se exponen los diferentes componentes que existen y cooperan en el desarrollo de una partida de póker completa y se procederá a ofrecer una visión general de todos ellos para entender el funcionamiento de este sistema.

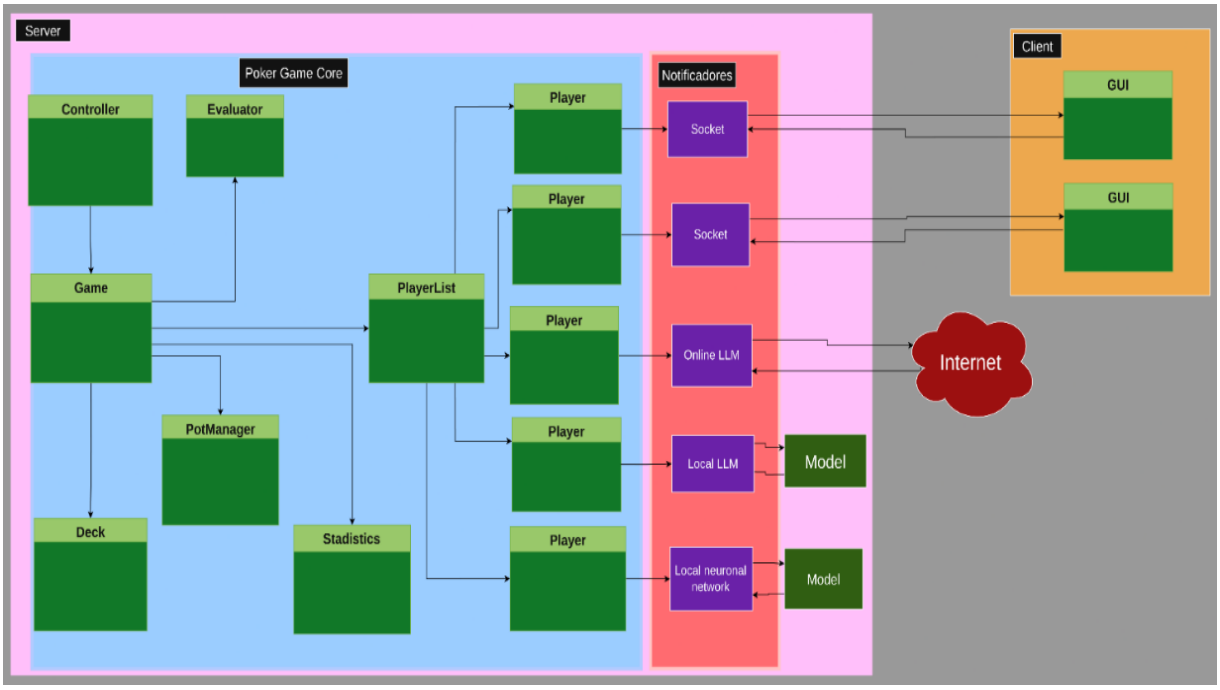


Figura 3-3. Implementación de lógica de póker

Podemos entender toda la lógica del póker como una serie de módulos interconectados para completar de forma correcta una partida de póker completa. Podemos distinguir como los módulos más relevantes los listados a continuación:

- Controlador: módulo que mantiene el bucle principal de toda partida de póker para asegurar que siempre se siguen las mismas secciones bien diferenciadas en cada mano. Sirve como punto de entrada a toda la lógica y como una referencia explícita a las distintas etapas de un juego de póker completo. Su estructura es extremadamente sencilla y se basa en las acciones secuenciales asignar roles, repartir cartas a los jugadores, jugar preflop, jugar flop, jugar turn, jugar river y showdown, todo mientras queden dos jugadores o más con dinero disponible.
- Contenedor de juego: módulo que contiene todos los elementos involucrados en una partida de póker y el estado del juego en cada momento. Esto lo podemos entender como un módulo de gran tamaño encargado de aunar a todos los demás módulos responsables para completar ciertas tareas dentro de una partida. Estas tareas pueden ser asignar roles a cada jugador, colocar o retirar cartas de la mesa, repartir o recoger las cartas de los jugadores, seleccionar cartas del mazo de manera aleatoria o pseudoaleatoria, determinar al ganador de una mano, determinar el bote correspondiente para cada ganador y otros muchos aspectos del juego.
- Lista de jugadores: módulo que mantiene el estado de todos los jugadores y se encarga de gestionar a estos. Esta lista de jugadores también delega ciertas acciones mucho más específicas en otros módulos como el evaluador de manos ganadora y el repartidor de mano ganadora. Además de estas tareas, también mantiene el estado de ciertos aspectos más concretos de la partida, como qué jugadores se encuentran desconectados por cualquier problema de conexión o el bote total de la mano actual.
- Evaluador de mano ganadora: módulo especializado en determinar el ganador según sus cartas en mano y las cartas comunitarias entre un conjunto de jugadores involucrados.
- Repartidor de bote ganador: módulo especializado en determinar la cantidad de bote total para el jugador ganador. Teniendo en cuenta los casos de empate, jugadores que realicen apuestas menores al resto de implicados y otros casos peculiares.
- Notificadores: estos no son un módulo propiamente dicho, sino que son un conjunto de llamados notificadores, los cuales sí que son módulos especializados

en hacer de puente entre cada evento relevante ocurrido en partida y el tipo de cliente que controla a un jugador concreto.

Estos notificadores se encargan de entregar cualquier tipo de información a jugadores humanos, bots de tipo LLM, bots de tipo red neuronal y cualquier otro tipo de implementación de bot que puedan incluirse.

- Jugador: módulo asignado a cada cliente conectado que contiene el estado de cada jugador en la partida. Este módulo contiene toda la información y acciones relevantes para cualquier tipo de jugador, bien sea su nombre, identificador, sus cartas, su dinero apostado o sin apostar y otros muchos datos relevantes que serán proporcionados a su notificador para enviar la información relevante.
- Módulos estadísticos: módulo encargado de registrar todas las acciones relevantes de cada jugador y mantener un historial y cálculos relevantes para obtener diferentes métricas.

3.3 Integración de bots

En este capítulo se detalla el funcionamiento de todos los tipos de bots de la aplicación, así como la jerarquía de clases y las interfaces implementadas, tal y como se muestra en la Figura 3-4, y otros mecanismos auxiliares que cooperan para proveer todo el contexto necesario para realizar jugadas o analizar la partida.

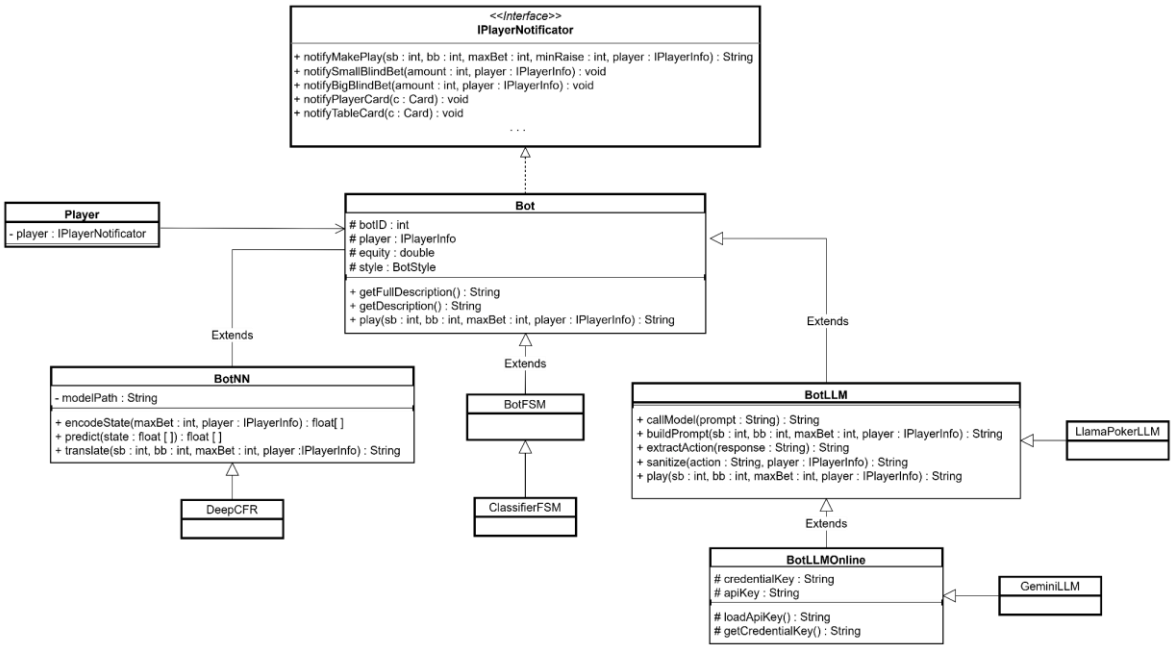


Figura 3-4. Diagrama de clase jerarquía de bots e interfaces

Debido a la necesidad de la aplicación de soportar variedad de estrategias e implementaciones de modelos de inteligencia artificial, se ha diseñado un sistema independiente del tipo de jugador que participe en la partida. De esta manera, la aplicación puede utilizar jugadores humanos, modelos de lenguaje, redes neuronales, máquinas de estados y otras implementaciones sin tener que modificar el funcionamiento general de la partida.

Para conseguir esta independencia, se ha establecido una estructura común para todos los bots. Independientemente de la tecnología utilizada para desarrollarlos, todos deben poder recibir la información necesaria sobre la situación actual de la partida, procesarla según las necesidades del bot y devolver una decisión que pueda ser interpretada por el juego. Esta información de la partida pueden ser las cartas comunitarias, las acciones realizadas por cada jugador, el estado de estos tras estas acciones, desconexiones de jugadores, eliminaciones, bote total en la mesa, valor del *small blind* y *big blind,* valor de la última apuesta y cualquier otro evento o información relevante de la partida o los jugadores.

La comunicación entre la lógica de la partida y cada bot se realiza mediante los notificadores descritos anteriormente. Estos actúan como intermediarios entre el juego y la implementación concreta del jugador. Cuando ocurre un evento relevante durante la partida, como recibir nuevas cartas, comenzar un turno o producirse una acción de otro jugador, el notificador proporciona al bot la información que necesita para tomar su decisión. Una vez obtenida esta decisión, esta se devuelve a la lógica de la partida para que pueda ser procesada de la misma forma que una acción realizada por un jugador humano.

Gracias a esta estructura, el bot puede ser tratado por la aplicación como un jugador más de la partida sin necesidad de diferenciarlo de un jugador humano. Ambos siguen el mismo flujo del juego y sus acciones son procesadas de la misma forma, la diferencia se encuentra únicamente en cómo cada uno obtiene la decisión que va a realizar.

Esta separación permite que cada tipo de bot pueda utilizar una estrategia diferente sin afectar al resto de la aplicación. Por ejemplo, un bot basado en un modelo de lenguaje recibe la información de la partida mediante una descripción de la situación y devuelve una respuesta a partir de la cual se obtiene la acción que

debe realizar. En cambio, un bot basado en una red neuronal recibe los datos de la partida en un formato numérico adaptado a su modelo y devuelve la acción correspondiente. Desde el punto de vista de la partida, ambos se comportan de la misma forma, ya que reciben la información correspondiente y devuelven una acción válida.

En el caso del bot mediante una máquina de estados la lectura de datos es directa, puesto que no necesita ningún tipo de interpretación de los datos. Por otro lado, el bot mediante una red neuronal hará uso de los métodos `encodeState()`, `predict()` y `translate()` para formar un *pipeline* que se encargue de codificar el contexto del juego a un formato adecuado, alimentar a la red para producir una respuesta y transformar esta respuesta en un input comprensible por el servidor. En el caso del LLM, este *pipeline* estará formado por los métodos `buildPrompt()`, `callModel()`, `extractAction()` y `sanitize()` para formar un prompt, enviarlo al modelo, recibir su respuesta, extraer la acción seleccionada y realizar una limpieza de los datos para evitar enviar respuestas malformadas.

Una de las principales ventajas de este diseño es la facilidad para incorporar nuevos bots. Para añadir una nueva implementación no es necesario modificar la lógica principal del juego, sino adaptar el nuevo bot a la estructura común definida por el framework., Esto permite que puedan incorporarse implementaciones desarrolladas con tecnologías diferentes a las utilizadas originalmente en el proyecto.

De esta forma este diseño está planteado para separar la toma de decisiones de la ejecución de la partida. La aplicación se encarga de gestionar las reglas y el estado del juego, mientras que cada bot es responsable únicamente de decidir qué acción realizar. Esta separación proporciona la flexibilidad necesaria para utilizar el framework como una plataforma sobre la que desarrollar. integrar y probar diferentes tipos de bots de póker e incluso ponerlos a competir entre sí, haciendo comparaciones.

3.4 Arquitectura distribuida

Otra de las decisiones importantes sobre esta aplicación es la incorporación de la arquitectura cliente-servidor, transformando esta aplicación local en una aplicación distribuida. Esto es cualquier tipo de aplicación software que, mediante el uso de distintos sistemas independientes, cooperan y se comunican para cumplir con un objetivo común.

Esto supone un incremento en la complejidad del proyecto, pero aporta una mayor facilidad para la creación de partidas con humanos y bots al ser accesibles fácilmente a través de internet y evitar que los usuarios finales requieran una máquina con altas especificaciones para poder interactuar con la multitud de funcionalidades disponibles en la aplicación. Con esta nueva arquitectura, debemos distinguir dos módulos claramente diferenciados que trabajan conjuntamente para la completa funcionalidad de la aplicación.

Por un lado, tenemos la aplicación servidor, encargada de alojar toda la lógica para dar lugar a partidas de póker completas, gestionar posibles errores debidos a desconexiones, gestionar los recursos y claves privadas necesarias para ejecutar distintos tipos de modelos de inteligencia artificial y comunicarse con los clientes involucrados.

Por otro lado, tenemos la aplicación cliente, encargada de enviar y recibir información al servidor mediante una interfaz gráfica intuitiva y atractiva para el usuario final. De esta forma, la aplicación cliente únicamente debe encargarse de esta transmisión y presentación de información para realizar cualquier tipo de acción, reduciendo y simplificando en gran cantidad el trabajo realizado por las máquinas cliente.

3.5 Monitorización de partidas

La aplicación tiene un sistema de monitorización diseñado para mantener un registro del desarrollo de las partidas. Su diseño se planteó para que los acontecimientos que se producen durante una mano puedan ser identificados, asociados al jugador y almacenados de forma que se puedan consultar los resultados y analizarlos.

Esta monitorización se representa a través de un historial de partidas, que actúa como registro de todo lo ocurrido durante cada mano. En él se organiza la información general de la partida y la información particular de cada jugador. Se contemplan las acciones realizadas, las cartas, el resultado de la mano y las características de cada jugador. De esta forma, es posible relacionar cada acontecimiento con el jugador que lo ha producido y conservar información adicional sobre su participación, como el tipo de jugador o el modelo utilizado.

Este diseño se apoya en la separación utilizada para la integración de los bots mediante los notificadores. Estos actúan como punto de comunicación entre la partida y los jugadores, identificando de forma independiente los acontecimientos que se producen durante el juego. Así el sistema puede tratar de la misma forma la información generada por un jugador humano o por cualquiera de los bots, sin depender de cómo se haya implementado cada uno.

Como consecuencia, el diseño de la monitorización complementa al resto de los componentes de la aplicación. Mientras la lógica de póker se encarga de ejecutar la partida y los bots de tomar decisiones, el sistema de monitorización conserva lo ocurrido durante su ejecución. Esto permite utilizar posteriormente las mismas partidas para analizar y comparar los diferentes bots y constituye la base sobre la que se realiza la fase de experimentación.

Capítulo 4 - Implementación

Una vez definido el diseño de la aplicación y establecidos los principales componentes y relaciones entre ellos, en este capítulo se describe cómo se han llevado estas decisiones a la práctica. Para ello, se explica la implementación de los elementos que forman la aplicación y las soluciones planteadas en el diseño, comenzando por la interfaz gráfica y continuando con el resto de la lógica del sistema.

4.1 GUI

En esta sección se profundiza en la implementación de la interfaz gráfica de usuario (GUI). La Figura 4-1, muestra todos los componentes que intervienen en su desarrollo.

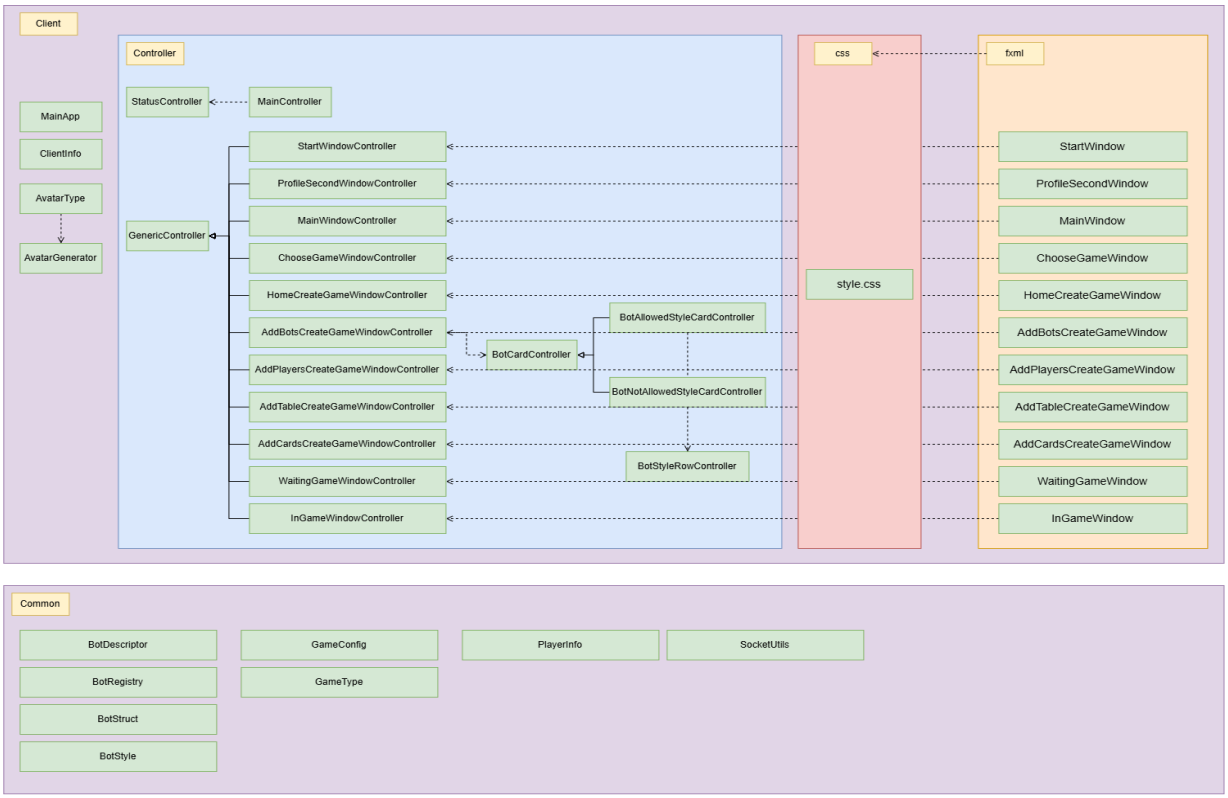


Figura 4-1. Estructura de la Implementación de la GUI

4.1.1 Configuración inicial y gestión de vistas

La gestión de la interfaz gráfica se centraliza en la clase `MainController`, responsable de controlar la navegación entre todas las ventanas de la aplicación. Su principal función consiste en cargar dinámicamente las vistas FXML correspondientes al estado actual de la aplicación y asociarlas a su controlador específico.

Para determinar qué ventana debe mostrarse en cada momento se utiliza la clase *`StatusController`*, que implementa una máquina de estados finita. Cada estado representa una fase concreta de la aplicación, como la creación de una partida, la configuración de bots, la selección de jugadores o el desarrollo de la partida. La navegación se realiza mediante los métodos `next()` y `back()`, los cuales actualizan el estado actual según el flujo previamente definido. La distribución de estas clases se puede observar en la Figura 4-2, que pertenece al bloque de `Controller`.

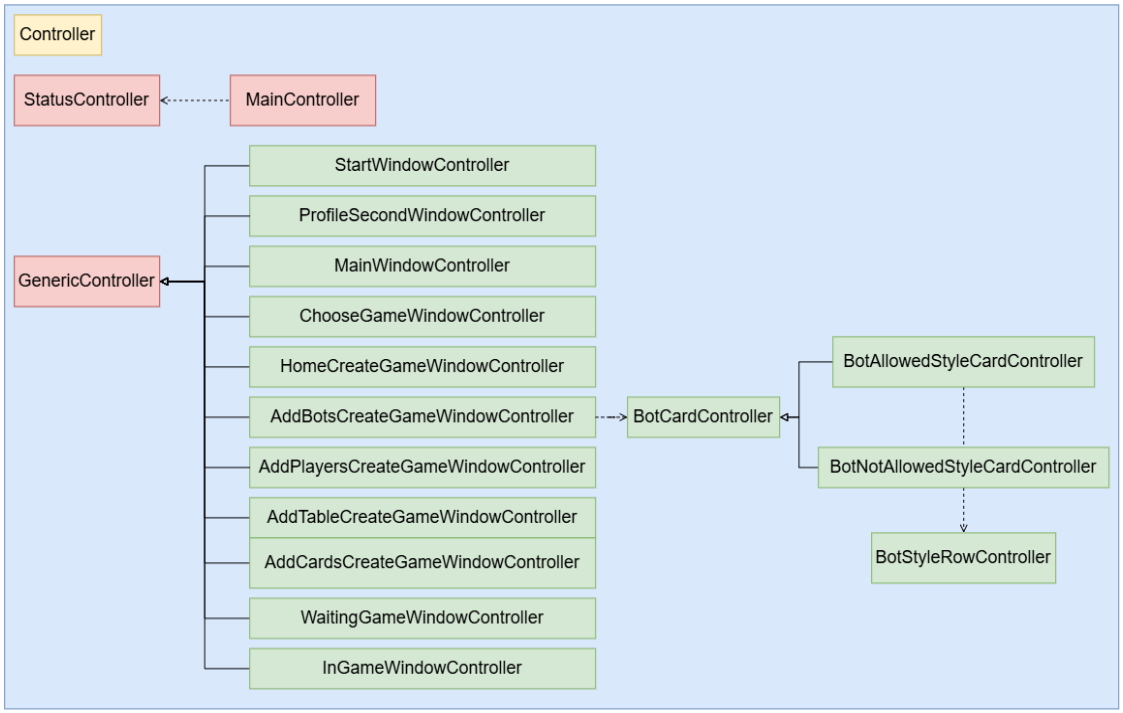


Figura 4-2. Clases que intervienen en la configuración inicial y gestión de vistas

Una vez actualizado el estado, `MainController` ejecuta el método `updateView()`, encargado de cargar el archivo FXML correspondiente mediante un objeto FXMLLoader. Después, se obtiene una referencia al controlador asociado a la

vista y se establece la comunicación con el controlador principal. Gracias a este mecanismo, cada ventana permanece desacoplada del resto de la aplicación y únicamente se preocupa de gestionar los elementos visuales que le corresponden.

Todas las ventanas heredan de la clase abstracta `GenericController`, mostrada en la Figura 4-2, que proporciona una interfaz común para la navegación entre vistas y el acceso a los datos compartidos de la aplicación. Esto permite un comportamiento homogéneo en toda la GUI y simplifica considerablemente la comunicación entre ventanas.

4.1.2 Creación de una partida

La creación de una partida se implementa mediante una secuencia de ventanas que se irán describiendo en este subapartado, donde cada una se encarga de configurar progresivamente todos los parámetros necesarios antes de iniciar una nueva sala de juego. Cada una de estas ventanas, presentes en la Figura 4-3, se encarga de una responsabilidad concreta y almacena la información introducida por el usuario en una única instancia compartida de la clase `GameConfig`, situada en el módulo *`common`*.

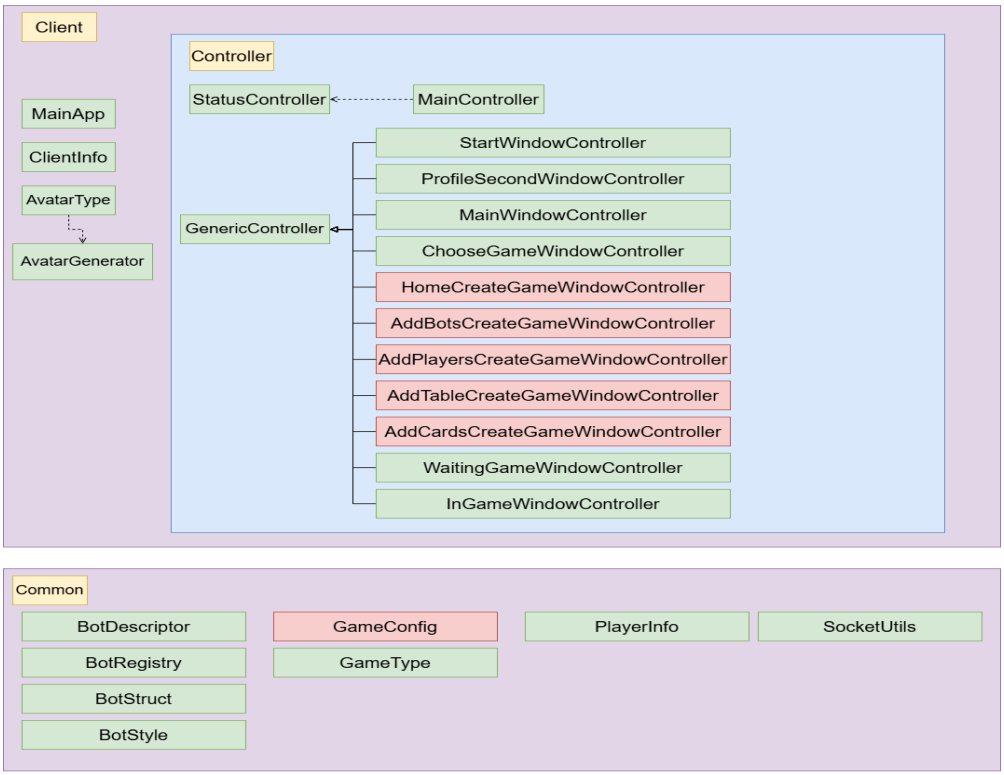


Figura 4-3. Clases que intervienen en la creación de una partida

Esta estructura permite desacoplar la configuración en diferentes etapas, facilitando la interacción del usuario y reduciendo la complejidad visual de cada pantalla. Además, gracias al almacenamiento centralizado proporcionado por `GameConfig`, los valores seleccionados permanecen disponibles durante toda la navegación entre ventanas.

La primera etapa del proceso para la creación de una sala está controlada por la clase `HomeCreateGameWindowController`, también visualizada en la Figura 4-3. Su responsabilidad consiste en configurar los parámetros generales de la sala. Durante la inicialización de la ventana se recuperan automáticamente los valores almacenados en `GameConfig`, permitiendo restaurar configuraciones anteriores si el usuario regresa posteriormente a esta pantalla.

Para garantizar la consistencia de los datos introducidos se aplican diferentes mecanismos de validación. Por ejemplo, el nombre de la sala se restringe a una longitud máxima y únicamente admite caracteres alfanuméricos, mientras que el dinero inicial se limita a un intervalo válido definido por la aplicación.

La información recogida es almacenada mediante el método `SaveHomeConfig()`, que actualiza los atributos correspondientes de `GameConfig`.

La segunda fase de configuración permite definir los bots que participarán en la partida. Esta funcionalidad se encuentra implementada mediante la clase `AddBotsCreateGameWindowController` (Figura 4-3), cuyo funcionamiento se describe detalladamente en el apartado siguiente debido a su complejidad y al elevado número de clases implicadas.

Al finalizar este proceso, la información queda almacenada dentro de las estructuras `_botsByType` y `_botStyleByType` de la clase `GameConfig`.

La configuración de jugadores humanos se implementa mediante la clase `AddPlayersCreateGameWindowController` (Figura 4-3). Además de permitir seleccionar el número de jugadores humanos que participarán en la partida, permite configurar el tiempo máximo disponible para realizar una acción durante cada turno.

Uno de los aspectos más importantes de esta implementación es la relación directa existente entre la cantidad de bots configurados y el número máximo de jugadores humanos que pueden añadirse. Para ello se utiliza el método

`getMaxRemainingPlayers()` que calcula dinámicamente el número de plazas disponibles en función de los bots previamente configurados y el número máximo de participantes que varía según si se ha habilitado la opción de espectador o no.

La personalización visual de la mesa y la baraja funcionan a partir de un controlador cada uno que mantienen una colección de mesas o cartas disponibles y permiten navegar entre ellas mediante controles de desplazamiento. De modo que todas las cartas mostradas durante la partida utilizarán el diseño escogido por el usuario, al igual que con la selección de mesas.

Para finalizar la creación de la sala, desde la clase `AddCardsCreateGameWindowController`, mostrada en la Figura 4-3, se realiza una validación básica para comprobar que existe al menos un participante en la partida, si la validación es satisfactoria, toda la configuración almacenada en `GameConfig` se serializa y envía al servidor. Posteriormente se espera la respuesta del servidor para determinar si la sala se ha creado correctamente. En caso de éxito, se almacena el identificador de la sala y se registra el rol del cliente como host. Finalmente, la aplicación avanza a la ventana de espera donde permanecerán todos los participantes hasta que el host decida iniciar la partida.

4.1.3 Agregación y configuración de bots

La configuración de los bots constituye una de las partes más complejas y extensibles de la interfaz gráfica, todas las clases que intervienen en su desarrollo están destacadas en la Figura 4-4. Esta funcionalidad se implementa mediante la clase `AddBotsCreateGameWindowController`, cuya responsabilidad principal consiste en generar dinámicamente la representación visual de todos los bots disponibles en la aplicación.

Para ello se utiliza la clase `BotRegistry`, que mantiene un registro centralizado de los bots implementados. Cada bot se describe mediante un objeto `BotDescriptor`, el cual almacena toda la información necesaria para construir su representación visual: identificador interno, nombre visible, descripción breve, descripción más detallada, imagen asociada e indicación si admite estilos de jugador.

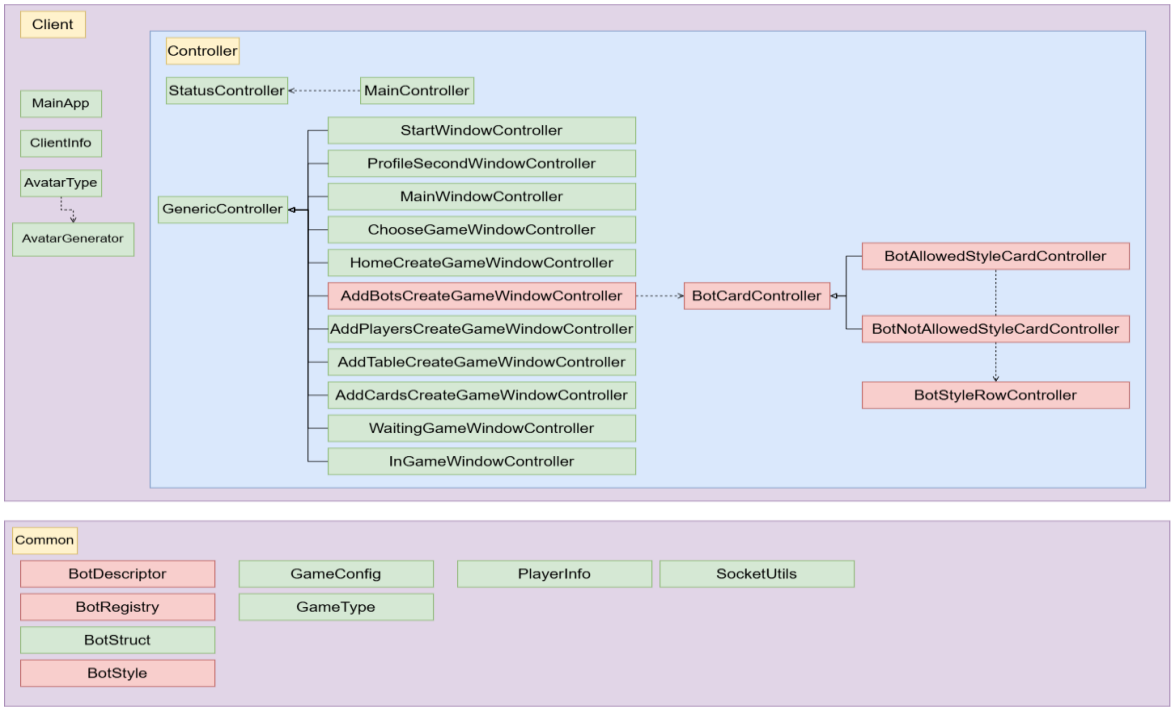


Figura 4-4. Clases que intervienen en la configuración de bots

Durante la inicialización de la ventana se recorre la colección de descriptores devuelta por `BotRegistry.getAvailableBots()`. Para cada descriptor se genera dinámicamente una nueva carta visual mediante el método `loadBotCard()`. Gracias a este enfoque, la interfaz se adapta automáticamente a los bots disponibles y la incorporación de nuevos tipos de bots no requiere modificaciones en el controlador principal de la ventana.

La estructura de herencia utilizada para las cartas de bots se basa en la clase abstracta `BotCardController`. Esta clase contiene toda la funcionalidad común compartida por cualquier tipo de bot, incluyendo la carga de imágenes, la visualización de descripciones, la animación de volteo de la carta, la inicialización de valores previamente almacenados y la gestión general de componentes visuales.

A partir de esta clase se derivan dos implementaciones concretas. La primera, `BotAllowedStyleCardController`, se utiliza para aquellos bots que admiten distintos estilos de juego. La segunda `BotNotAllowedStyleCardController`, se emplea para bots cuyo comportamiento no puede personalizar mediante estilos.

En la clase para los bots que permiten estilos se genera dinámicamente una fila visual para cada estilo definido en la enumeración `BotStyle`. Cada fila está

representada mediante un objeto `BotStyleRowController`, responsable de gestionar el selector numérico asociado a un tipo de bot y estilo. De esta forma, la cantidad de estilos mostrados en pantalla depende exclusivamente de los estilos registrados en `BotStyle`, favoreciendo la extensibilidad del sistema.

Cada `BotStyleRowController`, contiene un componente visual compuesto por una etiqueta descriptiva y un *spinner* que permite seleccionar cuántos bots de ese estilo concreto se incorporarán a la partida. Cuando el usuario incrementa o reduce la cantidad de bots de un determinado estilo, se notifica a su carta correspondiente, la cual actualiza automáticamente el número total de bots de ese tipo.

Por otra parte, los bots que no admiten estilos utilizan el controlador `BotNotAllowedStyleCardController`. Su funcionamiento es similar al de los bots con estilos, la diferencia es que disponen de una única fila con controles.

Uno de los aspectos más importantes de esta implementación es el control dinámico de los límites de selección. Cada vez que se modifica la cantidad de bots de cualquier tipo o estilo, el método `updateAllSpinners()` recalcula el número total de bots configurados y actualiza automáticamente los límites superiores de todos los selectores. Gracias a ello, la interfaz garantiza que nunca sea posible superar la cantidad máxima de participantes en una partida.

4.1.4 Unión de una partida y sala de espera

Una vez que se haya creado una partida, otros jugadores pueden unirse a ella mediante la opción de "*unirse partida*" accesible desde la ventana `ChooseGameWindow` de la carpeta *`Controller`* que se muestra en la Figura 4-5. Una vez procesada la solicitud, se mostrará un mensaje de éxito en caso de unirse correctamente, y de error, en caso contrario.

Después de unirse, el jugador pasa a la ventana `WaitingGameWindow`, destacado en la Figura 4-5, cuyo objetivo es mostrar a todos los jugadores, tanto los humanos conectados como los bots, y mantener sincronizados los posibles cambios que pueden ocurrir hasta que el creador de la partida decida iniciarla.

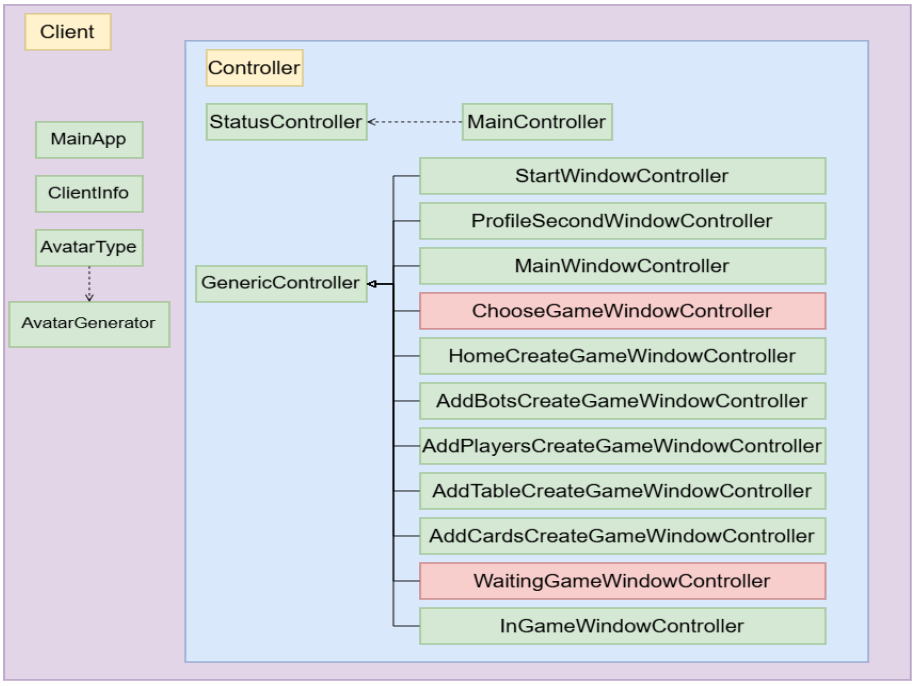


Figura 4-5. Clases que intervienen en la unión y espera de una partida

Dentro de la ventana `WaitingGameWindow` cabe destacar la visualización de las posiciones de los jugadores. Implementadas a partir de la lista ordenada de objetos `PlayerInfo`, la misma lista que se iba actualizando a medida que un nuevo participante se unía a la partida. La complejidad de esta implementación surge de no poder mostrar la lista directamente, pues las posiciones de los usuarios varían dependiendo del orden de conexión. Por lo que optamos por poner en la posición central inferior de la mesa (posición 0) al jugador local independientemente de la posición real que ocupe dentro de la lista. Después obtenemos su índice, a continuación, se posicionan el resto de los jugadores, según si están situados antes o después del usuario en la lista y siguiente dicho índice como referencia. Los jugadores que van antes del jugador actual se colocan en las posiciones 8, 7, 6… mientras que los situados después van en las posiciones 1, 2, 3…

Durante la ventana de espera, el modo espectador requiere un tratamiento especial. Para la recolocación de las posiciones de los jugadores, se asigna al espectador un identificador específico. Cuando la interfaz detecta esta situación, omite el orden de posiciones basado en el jugador local y muestra directamente todos los participantes de la mesa en el orden recibido desde el servidor.

4.1.5 En partida

La pantalla de juego constituye la interfaz más compleja de toda la aplicación, ya que se tiene que representar gráficamente el estado completo de una partida NLHE e ir actualizándose continuamente conforme se van produciendo acciones de juego, cambios de turno o nuevas cartas comunitarias.

La implementación de esta funcionalidad se realiza principalmente mediante la clase `InGameWindowController`, observable en la Figura 4-6, que se encarga de coordinar la lógica gráfica de la mesa y sincronizarla con la información recibida desde el servidor.

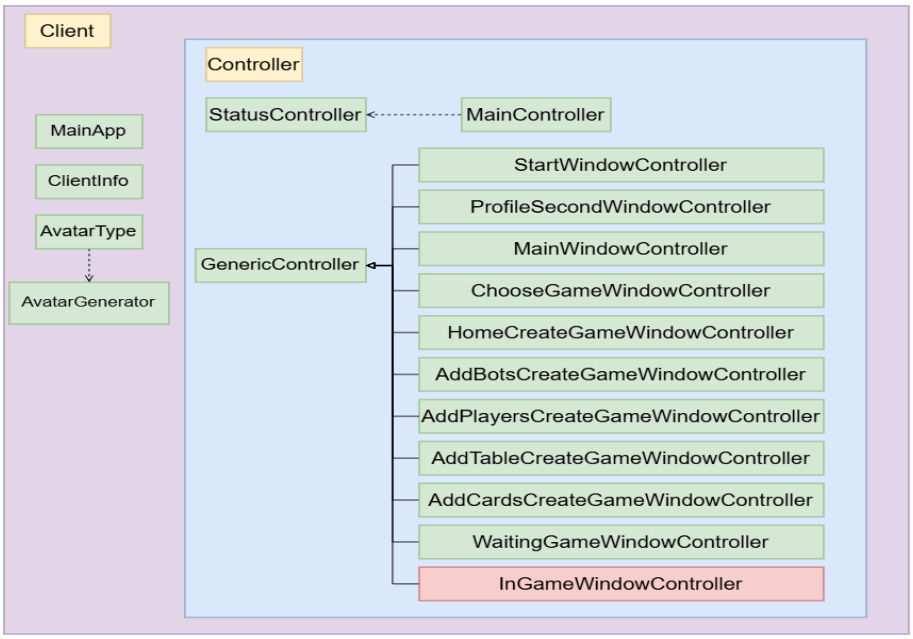


Figura 4-6. Visualización del controlador de la partida en juego dentro de la estructura de la GUI

La estructura general de la mesa de póker está diseñada para una capacidad de nueve participantes a la vez. Cada posición de la mesa consta de varios elementos visuales: El avatar del jugador, su nombre, su dinero disponible, la cantidad que apuesta, sus cartas privadas, el indicador de turno, el indicador de dealer, un indicador para el equity y un temporizador visual para marcar el tiempo de acción de cada jugador.

Para la asignación de posiciones en la mesa, al igual que sucede en la sala de espera, las posiciones de los jugadores en la mesa están orientadas a partir del jugador local, que ocupa la posición 0, mientras que el resto se distribuyen de forma

relativa. Durante la inicialización de las posiciones se crea un mapa de correspondencia entre identificadores de jugador y posiciones gráficas para almacenar su posición visual y poder identificar todos los componentes visuales asociados a un jugador en concreto rápidamente.

La representación de las cartas tanto privadas como comunitarias, se implementan mediante componentes `ImageView`. Para poder mostrar una carta determinada, desde el método `GUI_showCard()` se cogen los atributos internos de una carta *(número y palo)* y se transforman en rutas de imagen específicas que posteriormente son cargadas y mostradas en la interfaz (ver Figura 3-2).

En cuanto a la visualización de apuestas, continuamente la interfaz muestra el estado económico de la partida. Para cada participante se muestra el dinero comprometido actualmente en el bote y el que le queda. Cuando un jugador realiza una acción de apuesta se actualiza inmediatamente dichas cantidades asociadas, y la cantidad del bote común, ya que desde el servidor se envía el nuevo valor para mantener la sincronización económica de la mano.

La interacción del jugador con la interfaz sucede cuando le toca realizar una acción (ver Figura 3-2) que puede ser: fold, check, call o raise. También se puede aumentar la apuesta, o bien con los botones *MIN (apuesta actual + (apuesta actual - última apuesta)), HALF (apostar la mitad del dinero del jugador) y MAX (equivalente a hacer un Allin)*, o bien con los botones de aumentar o disminuir la apuesta siguiendo incrementos adaptados al valor de la ciega grande, o bien seleccionando la cantidad apostada con un *slider* que se configura dinámicamente según la situación actual de la ronda.

Para la gestión de la acción del jugador, una vez que el jugador ha realizado una acción no se envía directamente al servidor, en su lugar se introduce en una cola de comandos, para evitar problemas provocados por múltiples clics consecutivos. Posteriormente, el método `selectCommand()` extrae la acción elegida y la transmite al servidor.

Con el objetivo de mejorar la claridad visual de la interfaz de la partida, el jugador que dispone del turno activo se destaca gráficamente media un efecto visual luminoso (ver Figura 3-2), que va cambiando a medida que avanzan los turnos gracias al servidor que comunica el cambio ejecutando `GUI_putTurnPlayer()`.

Además, cada turno cuenta con un tiempo máximo de respuesta modificable desde la ventana de configuración de jugadores durante la creación de partida. Para representar el tiempo restante visualmente se ha desarrollado un temporizador gráfico de seis bloques rectangulares (ver Figura 3-2). La implementación utiliza un `ScheduledExecutorService` que actualiza los bloques cada pocos milisegundos. A medida que transcurre el tiempo, los bloques desaparecen progresivamente hasta indicar que el turno ha expirado.

La interfaz también representa gráficamente los roles especiales de cada mano. El botón del dealer se muestra mediante imágenes específicas asociadas a cada asiento (ver Figura 3-2). La posición de este marcador se actualiza automáticamente al comienzo de cada nueva mano según la información recibida desde el servidor.

La aplicación incorpora una funcionalidad adicional destinada al análisis de probabilidades. Durante la partida el servidor calcula el equity asociado a la mano actual del jugador y la envía periódicamente al cliente. Dicha información se almacena en una variable que puede mostrarse u ocultarse mediante un botón específico. Además, se utilizan varios colores para representar el nivel de probabilidad de ganar: verde (alta), amarillo(intermedia) y rojo(baja), tal y como podemos ver en la Figura 4-7.



Figura 4-7. Visualización del equity de cada jugador durante una partida de póker

Finalmente, la misma ventana también soporta el modo espectador, configurable a la hora de crear una nueva partida en la ventana `HomeCreateGameWindowController`. En este caso, como el usuario no participa en la partida, la interfaz deshabilita todos los controles del juego. Sin embargo, continúa

recibiendo todos los eventos enviados por el servidor, permitiéndole observar en tiempo real el paso a paso de la partida sin intervenir en su desarrollo.

4.1.6 Visualización de la interfaz

A partir de los bocetos hechos para el diseño de la interfaz, se ha intentado recrearlos utilizando los elementos proporcionados por JavaFX. Concretamente, archivos FXML para definir la estructura visual de cada ventana, *SceneBuilder* para facilitar el diseño de las vistas y una hoja de estilo CSS para englobar la apariencia visual de toda la aplicación. Teniendo esta estructura se consigue una interfaz modular, reutilizable y mantenible.

Para cada ventana, su visualización se define mediante un archivo FXML donde se describen los componentes gráficos que forman la interfaz y su distribución dentro de la ventana. Un ejemplo de ello se puede ver en la Figura 4-8, correspondiente a la ventana `homeCreateGameWindow.fxml`.

En la Figura 4-9 se muestra como cada archivo FXML está asociado a un controlador, de forma que la lógica de comportamiento queda separada de la definición visual de los elementos. Cabe destacar que todos los archivos presentan componentes reutilizables y una estructura similar, facilitando la navegación del usuario entre las distintas pantallas de la aplicación.

Para facilitar la edición manual de archivos FXML, se ha utilizado la herramienta SceneBuilder, de esta forma nos ha permitido diseñar las vistas de manera más sencilla usando una metodología de arrastrar y soltar componentes JavaFX sobre la escena. Además, nos ha permitido: definir posiciones relativas de los elementos; implementar jerarquías de contenedores; establecer márgenes y separaciones; usar identificadores, útiles a la hora de implementar el controlador de la vista y asociar acciones del usuario con métodos de Java.

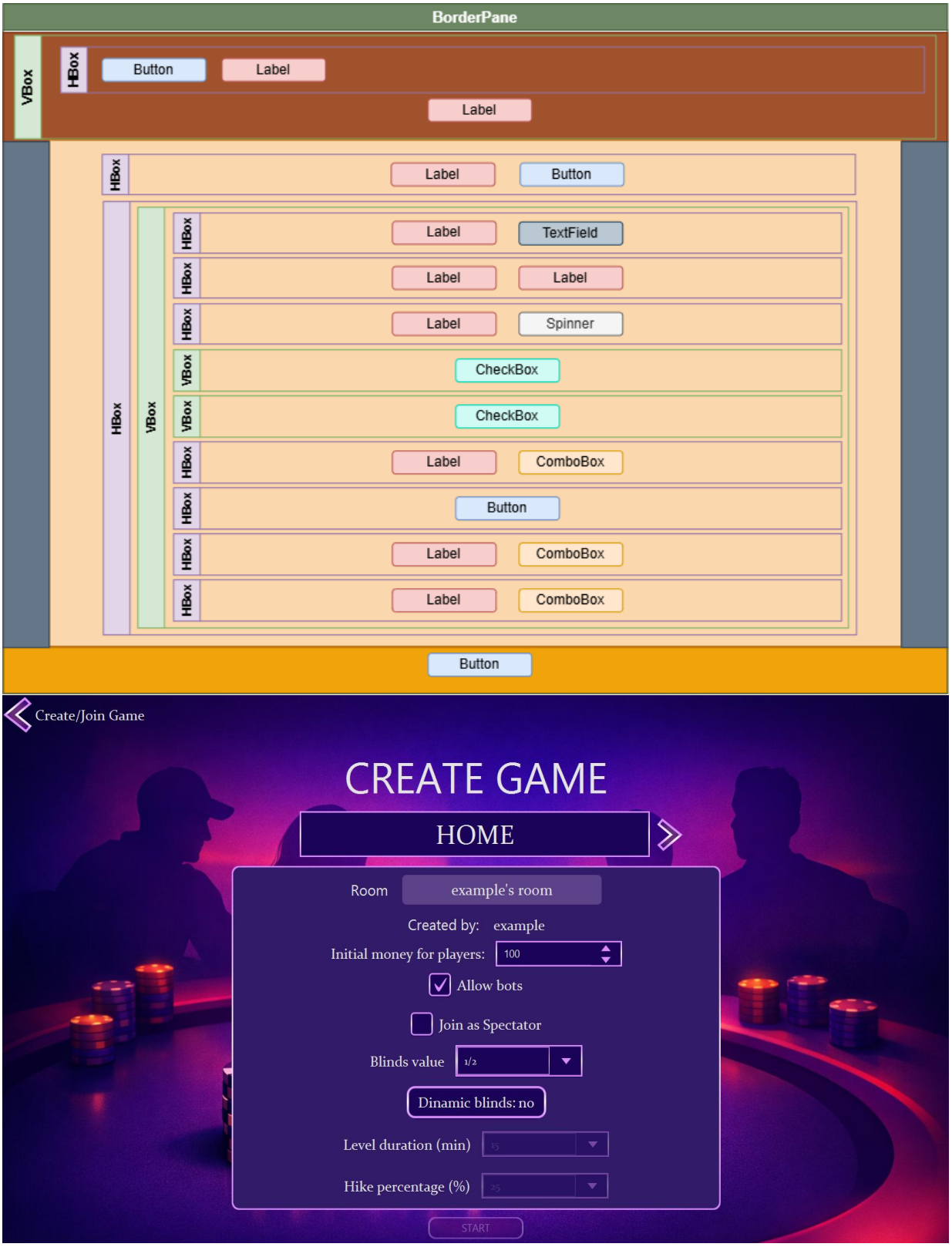


Figura 4-8 Diagrama de distribución de la venta homeCreateGameWindow

Finalmente, para mantener una identidad visual coherente en todas las ventanas, la apariencia de la aplicación se centraliza en una única hoja de estilos css, llamada `style.css` (ver Figura 4-1). De forma que cualquier cambio visual realizado en esta clase, automáticamente se actualiza en toda la aplicación.

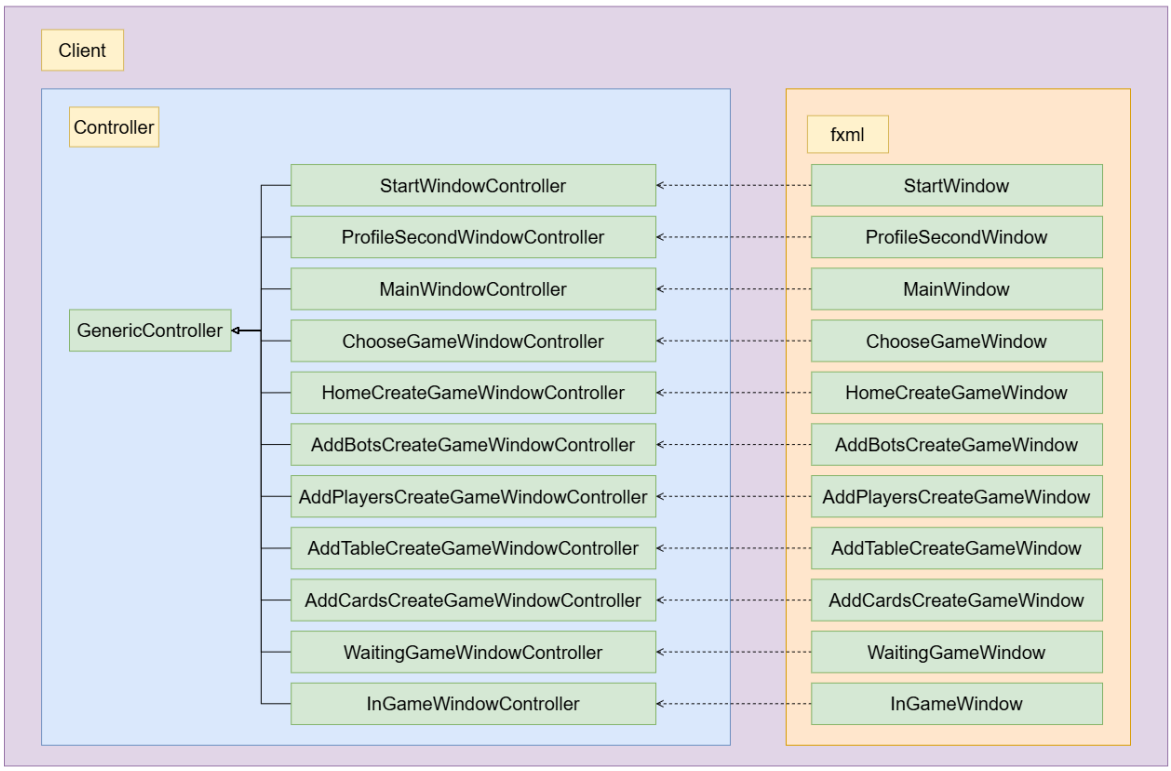


Figura 4-9. Estructura de los controladores y archivos FXML correspondientes

Esta hoja está estructurada según distintas categorías:

- Estilo general de la aplicación: en esta parte se encuentran todas las clases de componentes generales, como la raíz, donde se establece el fondo común para todas las ventanas, el estilo de los botones o la tipografía de la aplicación (*Constantia*) juntos con sus variantes en función si es un título, subtítulo o una descripción.
- Flechas: aquí se define el diseño de todas las flechas de la aplicación.
- Contenedores y Layouts: gran parte de la interfaz se basa en clases CSS reutilizables destinadas a contenedores y layouts ya que se usa en la mayoría de las ventanas.
- otras categorías que aportan información visual al usuario y añaden efectos luminosos a los elementos seleccionados: iconos, alertas y notificaciones.

4.1.7 Generación dinámica de avatares

Con el objetivo de proporcionar una representación visual única para cada participante sin necesidad de almacenar una imagen externa, hemos desarrollado un sistema para generar avatares de forma dinámica, en la Figura 4-10 se puede visualizar la organización de las clases que intervienen en el desarrollo de esta visualización. Gracias a este mecanismo se pueden crear imágenes identificativas para cada jugador, tanto humano como bot, de forma automática a partir de información ya disponible en la aplicación.

Para la generación de avatares diferenciamos entre jugadores humanos y bots, creados a partir de la clase `AvatarGenerator`.

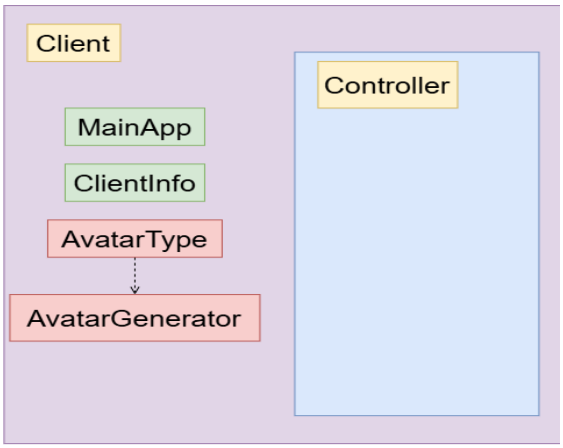


Figura 4-10. Estructura de las clases generadoras de los avatares

En función de para quién sea el avatar, bien se llamará al método:

```
generateHuman(String seed, int size)
generateBot(AvatarType type, int size)
```

donde el parámetro *size* determina las dimensiones de la imagen generada y los parámetros *seed* y *type* permiten construir una representación visual consistente para cada participante.

La generación de avatares para jugadores humanos se realiza a partir del nombre introducido por el usuario en la ventana `ProfileSecondWindow`, mostrada en la Figura 4-11. Para garantizar que un mismo nombre siempre produzca la misma

imagen se utiliza un algoritmo criptográfico *SHA-256,* el hash generado se transforma en una semilla numérica que se utiliza para iniciar un generador pseudoaleatorio.

A continuación, se construye la imagen usando un objeto `Canvas` de *JavaFX* que genera los elementos faciales que se dibujan gracias a la clase `GraphicsContext`. Finalmente, el resultado se convierte en un objeto `WritableImage` que puede usarse en cualquier componente gráfico de JavaFx.

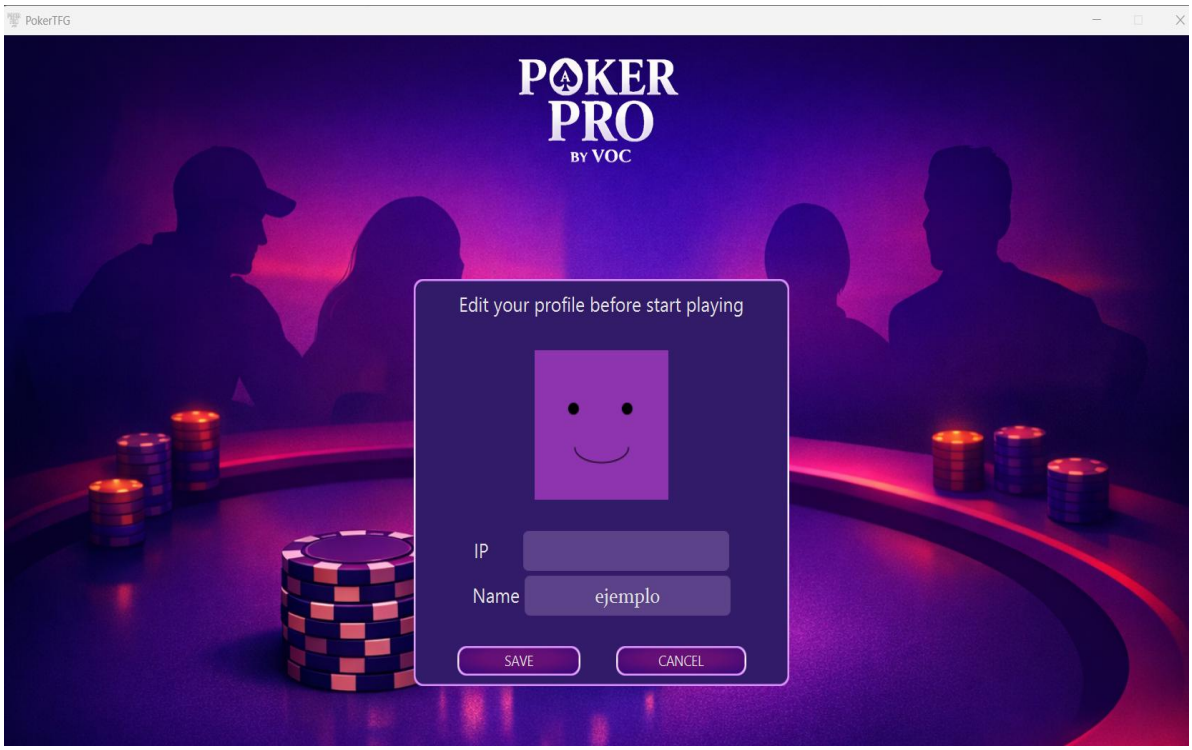


Figura 4-11. Visualización de la ventana ProfileSecondWindow

La forma en la que se generan los avatares de bots es diferente a los avatares humanos, pues no son dinámicos, sino que se basan en una identidad gráfica común que permite diferenciarlos de los jugadores humanos.

En primer lugar, tenemos una clase `AvatarType`, desde donde se definen los tipos de bots para los que se generarán un avatar diferente, por ello, cada uno posee un color principal distinto que lo caracteriza.

Para la generación del avatar se utiliza el método `generateBotAvatar()` que construye una cara con apariencia robótica. Todos los elementos son dibujados con operaciones vectoriales sobre un objeto *Canvas*.

La principal ventaja de este enfoque es que la aplicación puede generar múltiples avatares diferentes sin ocupar espacio de almacenamiento en el proyecto, garantizando que cada jugador conserve siempre la misma apariencia a partir de su nombre y que cada tipo de bot posea una identidad gráfica claramente diferenciada.

4.2 Componentes de una partida de póker

A continuación, se explica el funcionamiento de cada uno de los módulos implicados en el funcionamiento del juego de póker y en el mecanismo de comunicación con otros módulos en detalle:

- Controlador: este módulo se encarga de establecer el comienzo el juego y coordinar las diferentes acciones del juego. Estas acciones son la asignación de roles de los jugadores, el reparto de sus cartas, la colocación de las cartas comunitarias, el reparto del bot a los ganadores de la ronda y el paso de turnos para la siguiente ronda. También debe comprobar si el juego ha llegado a su fin, debido a que solo quede un jugador en la mesa con dinero, y terminarlo si es necesario.

  Este módulo tiene una implementación completamente secuencial y basada en la implementación del módulo contenedor de juego, orquestando el bucle principal del juego y sirviendo como primer punto de entrada a toda la lógica.

- Contenedor de juego: Este módulo de gran tamaño se encarga mayormente de delegar tareas específicas al módulo de lista de jugadores y al módulo de estadística, almacenar la configuración del juego, configurar y gestionar temporizadores si son necesarios e interactuar con el mazo de cartas cuando haya que repartir o recoger cartas de los jugadores o de la mesa.

  Podemos entender este módulo como un proveedor de todas las acciones y dependencias necesarias para que el juego de póker se desarrolle.

- Lista de jugadores: este módulo es el encargado de interactuar de forma directa con multitud de módulos diferentes para desarrollar todas las tareas solicitadas por el contenedor de juego. Este se comunica de forma directa con los jugadores, el espectador de la partida, si hay uno; con el módulo repartidor de bote ganador, notificar a todos los jugadores de cualquier tipo de evento relevante

sucedido en la partida, entre otros. De todas estas acciones, podemos destacar la implementación de la funcionalidad encargada de desarrollar el juego de una mano completa.

Para entender el funcionamiento de esta, debemos entender primero que la lista que alberga a todos los jugadores se trata de una lista doblemente enlazada circular.

La Figura 4-12 muestra la estructura de datos basada en una lista que almacena una serie de *nodos*, los cuales son los que almacenan la información deseada; que en este caso son los jugadores junto a ciertas variables adicionales. Cada nodo almacena además un llamado *puntero* al nodo anterior o siguiente nodo respecto a él, con la peculiaridad de que el primer nodo y el último se apuntan mutuamente, cerrando la lista por ambos extremos y formando un círculo.

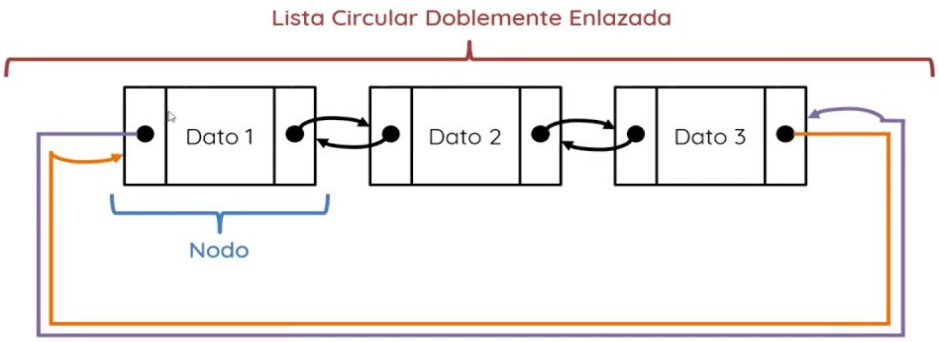


Figura 4-12. Esquema de lista circular doblemente enlazada

Teniendo en cuenta esto, podemos ver la semejanza del funcionamiento de esta estructura de datos con el de una mesa de póker real y pasar a entender cómo logramos replicar este comportamiento.

En primer lugar, una condición que siempre debe cumplirse en esta lista es que el primer nodo siempre deberá *apuntar* al *dealer* de la mesa, para poder acceder al resto de jugadores en orden según sus roles. El único caso que no sigue esta condición es el de una mesa de únicamente dos jugadores activos, lo que causará que el primer nodo apunte al small blind en ese momento, prescindiendo del *dealer* por completo.

Para completar una mano de póker de forma correcta marcaremos a cada jugador tras realizar una acción para saber si ya ha tenido oportunidad de actuar

durante la ronda de apuestas actual. De esta forma, el sistema podrá determinar cuándo todos los jugadores activos han respondido a la apuesta vigente y, por tanto, cuándo puede darse por finalizada la ronda y avanzar a la siguiente fase de la mano

Este marcaje se producirá cuando un jugador hace fold, call o check, debido a que estas jugadas eliminan al jugador de la mano actual o igualan su apuesta con la apuesta máxima de la mano actual. Por otro lado, cuando un jugador haga un raise o un all-in, deberá comprobarse que la nueva cantidad de apuesta es realmente mayor que la actual, produciendo así que el resto de turnos de jugadores se "invaliden" o desmarquen, obligándolos a hablar de nuevo para decidir nuevamente si igualan, suben o se retiran de la mano.

Este bucle se repetirá hasta que todos los jugadores hayan hablado ya o bien todos excepto uno se haya retirado de la mano, tal y como ocurre en el póker NLHE.

- Evaluador de mano ganadora: este módulo tiene la vital tarea de determinar rápidamente y con total precisión cuál es la mano de cartas ganadora de entre una serie de jugadores.

  Antes de nada, debemos tener en cuenta los retos que se nos presentan, los cuales son obtener un algoritmo que sea agnóstico al orden en el que se presentan las cartas y con un tiempo de respuesta prácticamente inmediato.

  Existen multitud de estrategias para realizar este cálculo por ordenador de la forma más óptima, determinista y rápida posible y, en nuestro caso, hemos decidido utilizar un algoritmo público desarrollado por "*Cactus Kev*".

  En primer lugar, debemos saber que en la modalidad de póker NLHE tenemos una baraja de 52 cartas, de las cuales los jugadores tendrán un subconjunto de 7 de ellas para obtener la mejor mano posible de 5 cartas.

  Por ello existen 2.598.960 combinaciones (52 cartas en grupos de 5), las cuales son demasiadas como para intentar evaluarlas mediante fuerza bruta de forma individual con un coste en tiempo razonable.

  Sin embargo, esta cantidad se ve reducida drásticamente si tenemos en cuenta que únicamente hay 7462 combinaciones distintas, sin tener en cuenta los palos

de las cartas y el orden en el que las obtengamos, lo que es un volumen de datos más manejable. Con esto en mente, dada cualquier combinación de 5 cartas podemos traducirla a su mano única correspondiente, pudiendo compararla más fácilmente con otras manos de mayor, igual o menor valor y obtener la mano ganadora entre un conjunto de estas.

Teniendo en cuenta las consideraciones mencionadas anteriormente en la sección de diseño de este mismo apartado podemos entender mejor cómo funciona el algoritmo, el cual está basado en la multiplicación de números primos.

Esto es debido a que la multiplicación de una serie de números primos producirá un resultado único independientemente del orden de estos. Esto es vital ya que, si asignamos a cada número de la baraja un número primo distinto y multiplicamos sus valores, podremos obtener un número que identifica inequívocamente a un subconjunto de cartas, lo cual resulta clave para obtener un algoritmo agnóstico al orden de las cartas.

Por último, debemos saber si este conjunto de 5 cartas son del mismo palo ya que, de ser así, esto aumentaría drásticamente el valor de esta mano.

Debido a que cada jugador tiene a su disposición las 5 cartas comunitarias y sus 2 cartas en mano, podemos obtener su mejor mano de 5 cartas de entre estas 7 haciendo todas las combinaciones posibles. Ya que la valoración de una mano de cartas es realmente rápida y sólo existen 21 combinaciones de manos por cada jugador, con un máximo de 9 jugadores por partida, obtenemos un volumen de cálculos máximo realmente reducido (189 operaciones) para realizar esta operación mediante "fuerza bruta".

Con todo esto, podemos obtener el valor de una mano haciendo uso de unas tablas precalculadas para los *flushes* y otras manos únicas y obtener su valor con un coste en tiempo constante.

Sin embargo, el tamaño de alojar estas tablas completas en memoria supone un gasto de memoria sobre los 100KB, lo cual no es una cifra alarmante teniendo en cuenta que en la aplicación van a existir modelos de inteligencia artificial mucho más grandes que esto.

De igual manera debemos tener en mente este consumo de memoria para evitar desperdiciar los recursos del sistema y, ya que cada partida de póker va a requerir de esta estructura de datos para calcular las manos de cartas ganadoras en multitud de ocasiones, hemos hecho uso del patrón de diseño Singleton.

De esta forma, mantendremos una única instancia de esta estructura de datos accesible de forma global por la aplicación, de forma que el consumo en memoria no se eleve en gran medida por cada partida simultánea existente.

- Repartidor de bote ganador: este módulo también tiene una tarea realmente importante y decidir qué jugadores es el ganador de la mano según sus cartas y, en casos excepcionales, qué jugadores han empatado y cuánto bote reciben.

  Primero, hay que entender dos casos excepcionales que hay que tener en cuenta.

  El primer caso ocurre cuando varios jugadores tienen exactamente la misma mano y, por lo tanto, ambos deberían recibir la totalidad del bote. En esos casos la totalidad del bote se divide entre todos los ganadores de forma igualitaria.

  El otro caso es más complejo, denominado *side-pots*, y ocurre cuando existen jugadores que apuestan en una ronda una cantidad menor a la que colocarían en condiciones normales debido a una menor cantidad de dinero a su disposición.

  Por ejemplo, un small blind apuesta 4 fichas reglamentarias, el big blind apuesta 8 fichas reglamentarias y el siguiente jugador, que supongamos que tiene un total de 6 fichas, no podría apostar las 8 que le correspondería apostar y deberá apostar las 6 que tiene si quiere entrar en la apuesta. El resto de la ronda continuaría con normalidad, pero, si ese jugador que ha realizado una apuesta menor ganase la mano, el bote a recibir no sería el total postado por todos los jugadores, que supongamos que es 24 fichas (8 del small blind, 8 del big blind y 6 del jugador restante) recibiría únicamente las 6 primeras fichas de la apuesta total de cada jugador involucrando. Esto sería, 6 de las 8 fichas apostadas por los otros jugadores y las 6 de él mismo. El resto de las fichas de otras apuestas se resolverán sin tener en cuenta a este jugador con una menor cantidad de fichas ya que este no habría participado en ellas.

Para resolver cualquier reparto en circunstancias normales y extraordinarias el algoritmo empleado se dividirá en dos partes.

La primera que se encargará de recoger y almacenar la apuesta total puesta por cada jugador a lo largo de todas las manos de una ronda, por lo que ocurrirá al final de cada mano de una ronda completa.

La segunda parte es la encargada de resolver el reparto del bote según las cartas de cada uno de ellos en el showdown. Para ello se formará una lista con la apuesta total de cada jugador ordenada de menor a mayor, se añadirán a una lista secundaria de candidatos y se les quitará de su apuesta total inicial la cantidad puesta por el jugador con la menor apuesta, ya que pueden existir *side pots* en los que pueden participar estos otros jugadores. En caso de que haya jugadores con una apuesta total con un valor de cero, estos serán eliminados de la lista principal y se repetirá el proceso hasta que se hayan procesado a todos los jugadores.

Tras crear todos los *side pots* que sean necesarios, se procederá a resolverlos, lo cual simplemente significa seleccionar al jugador o jugadores con la mejor valoración de sus cartas y sumar esa cantidad a su *stack* actual.

Con este algoritmo podemos repartir de una forma sencilla, uniforme y con un coste algorítmico y en tiempo muy bajo la distribución del bote que se ha de hacer según las circunstancias de la partida.

- Notificadores: este módulo o sección está formada por todas las implementaciones de notificadores para los jugadores y se dividen según el tipo de comunicación que se mantenga con cada tipo de jugador. Las implementaciones actuales se dividen en representadas en la Figura 4-13:

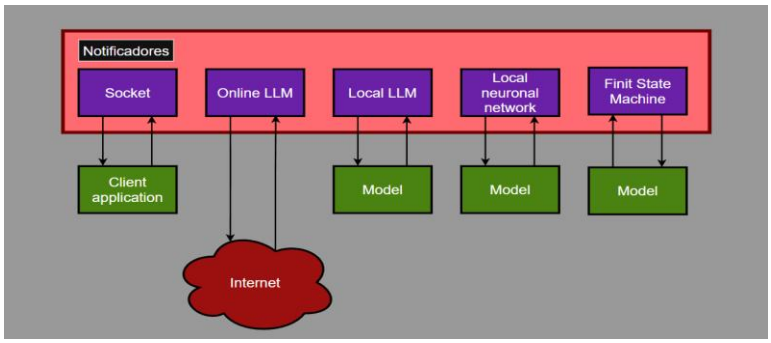


Figura 4-13. Arquitectura de notificadores

- HumanPlayer: este notificador se encarga de enviar y recibir datos a un cliente humano conectado mediante un socket TCP. Cada evento del juego está codificado en paquetes indicando primero el tipo de evento codificado con un identificador único y, a continuación, los datos necesarios para enviar la información del evento.
- Spectator: este notificador es una especialización de la implementación de `HumanPlayer` particular, pensada para adaptarse a ciertas necesidades de este tipo de jugador. Mantienen el mismo tipo de conexión mediante un socket TCP con la diferencia de que no recibe turno de juego y algunos datos los recibe de una manera ligeramente diferente, por lo que debe adaptarse.
- BotLLM**:** este notificador se diseñó para integrar modelos de lenguaje como jugadores de la partida. Su función principal es transformar el contexto del juego, que internamente se encuentra representado mediante diferentes estructuras y objetos, en un contexto textual del juego, que internamente se encuentra representado mediante diferentes estructuras y objetos, en un contexto textual que pueda ser procesado por el modelo. Para ello, se proporciona la información necesaria para que el modelo pueda tomar una decisión, como las acciones realizadas por los jugadores anteriores, sus posiciones, las cartas comunitarias, las cartas del jugador, el equity o el estilo de juego que se desea seguir. Una vez generado este contexto, el modelo proporciona una respuesta que posteriormente debe transformarse en una acción reconocida por la aplicación.

  Para la utilización de estos modelos se contemplaron tanto alternativas locales como servicios externos. En el caso de los modelos locales se utilizó Ollama, que permite ejecutar el modelo en el propio entorno de la aplicación y realizar las peticiones sin depender de un servicio externo. Para seleccionar el modelo que se utilizaría en esta configuración se recurrió a Hugging Face, buscando específicamente modelos que hubieran sido entrenados o adaptados para la toma de decisiones en póker. Entre las alternativas disponibles se seleccionó un modelo basado en LLama 3.1 8b Instruct, que ha sido entrenado específicamente para la toma de decisiones en póker utilizando el conjunto de datos *PokerBench*. Esta especificación resultaba especialmente adecuada para el proyecto, ya que permitía utilizar un modelo previamente adaptado al dominio del póker en lugar de partir de un modelo de propósito general.

Además, el modelo seleccionado proporciona sus decisiones mediante un formato estructurado basado en etiquetas <action> pudiendo generar acciones como fold, call, check, o raise junto con la cantidad correspondiente. Esto facilita su integración con el resto de componentes, ya que la respuesta obtenida puede extraerse y transformarse posteriormente en una acción válida para la partida.

- BotLLMOnline: este notificador es una especialización del notificador `BotLLM` que únicamente añade un mecanismo para aportar la clave de una API para poder acceder al modelo y realizar peticiones, como suele ocurrir con este tipo de servicios.

  En este caso, la aplicación implementa un LLM conectado al modelo de Google llamado *Gemini*, en concreto el modelo *Gemini 2.5 Flash*, un modelo que pretende mantener un mejor equilibro calidad-precio respecto a otros modelos mucho más potentes y, en consecuencia, más costosos.

- BotNN: este notificador se utiliza para la integración de una red neuronal o perceptrón multicapa. Esta implementación base requiere un buen conocimiento de la arquitectura de esta ya que será altamente importante para el correcto funcionamiento de la red según el tipo de entrenamiento recibido. Primero deberemos conocer las limitaciones de este tipo de modelos ya que son muy poco flexibles y, para entender en profundidad el contexto del juego de póker, necesitarán una gran cantidad de datos de entrada y salida y la codificación de estos.

  Una vez entendida su arquitectura y la forma de interpretar sus decisiones, deberemos decodificar la acción a realizar por la red y convertirlos en una serie de datos que la lógica de póker pueda comprender y procesar.

  Para ello, esta implementación requiere que cada red neuronal defina cómo se codifican los datos de entrada necesarios para aportar el contexto necesario a la red y que pueda tomar decisiones. Tras esto, la codificación obtenida debe ser suministrada a la red y recibir las salidas producidas por ésta. Teniendo la acción final, solo deberemos descodificar está de nuevo para producir una salida final coherente para la lógica de póker.

El modelo empleado en la aplicación se basa en la técnica de *Counterfactual Regret Minimization (CFR)*, dando lugar a un tipo de red neuronal llamada *Deep CFR* formada por 226.826 parámetros, ya que está formada por dos redes idénticas de 113.413 parámetros cada una. Cada red tiene una capa de entrada formada por 54 neuronas (52 para cada una de las posibles cartas de la baraja + apuesta actual de la red neuronal + apuesta máxima de la mesa), tres capas ocultas de 256, 256 y 128 neuronas, respectivamente, y una capa de salida de 5 neuronas codificando cada acción disponible (*fold, call, raise half-pot, raise full-pot, all-in*).

- BotFSM: esta implementación está pensada para una máquina de estados, o *Finit State Machine*, un tipo de inteligencia artificial muy sencillo pero limitado. Debido a que este tipo de implementación no posee nada más que de una serie de reglas secuenciales a evaluar para determinar la salida final. Al tratarse de una implementación sencilla y directa, lo único que debe de hacer este comunicador es leer el contexto necesario del juego, hacer una evaluación y responder. Este comunicador es el más útil para hacer implementaciones sencillas como un jugador completamente aleatorio u otras estrategias algo más deterministas como un bot que calcule la probabilidad de aparición de cartas según sus apariciones en partida para realizar cálculos estadísticos.

  La implementación de este modelo en la aplicación se basa en asignar una puntuación al valor de las cartas comunitarias y las cartas privadas y decidir mediante una distribución si debiera hacer *fold*, *call, raise* y *all-in*. Este cálculo no tiene en cuenta más variables de juego y resulta un método de decisión prácticamente pseudoaleatorio que hará que el bot apuesta en manos que contiene cartas altas y se retire en manos débiles.

- Jugador: podemos entender este módulo como una representación virtual de cada jugador en la partida, independientemente de si es un humano mediante una conexión remota, un LLM online, una red neuronal o cualquier otro tipo de inteligencia artificial. Este "jugador virtual" es el responsable de almacenar datos importantes como el nombre, las cartas en mano, el dinero total y apostado y otras muchas variables de estado necesarias para el desarrollo del juego de póker.

La clase *Jugador* está compuesta por varios interfaces, cada una con un propósito específico los módulos externos que necesiten interactuar con un jugador. Estas interfaces son:

  - IPlayerActions: proporciona las acciones que puede realizar un jugador en su turno y deben implementar el efecto que tienen en este. Estas acciones son hacer *call*, *check*, *fold*, *raise* y *all-in* y son las responsables de gestionar el dinero o estado del jugador. Estas acciones son utilizadas por los comandos del juego.
  - IPlayerInfo: proporciona acceso a los diferentes parámetros del jugador, como su dinero o identificador único, y permite únicamente consultar el estado de este. Esto es necesario para los mencionados notificadores y evitar la modificación de datos sensibles.
  - IPlayerNotificator: proporciona las diferentes notificaciones emitidas por el juego y que deben ser recibidas y tratadas de forma distinta según el tipo de jugador que se encuentre conectado. Esta interfaz tiene como objetivo proporcionar distintas formas de enviar estos datos a los clientes y responder a los eventos del juego según convenga. Por ejemplo, para un jugador humano puede ser mediante una conexión mediante un *socket* o *ssh*, para un LLM puede ser mediante una petición HTTP/HTTPS y así para cualquier tipo de jugador o forma de comunicación, permitiendo una gran variedad de métodos para comunicarse con el juego de póker.

- Módulos estadísticos: podemos entender esta sección como un conglomerado de módulos especializados en obtener diferentes métricas relevantes para evaluar la habilidad o el contexto de un jugador. Estas métricas pueden ser tan sencillas como la cantidad de veces que un jugador realiza una acción específica como el *equity* de este. Para ello esta serie de módulos puede obtener estas métricas en momentos específicos del juego, como ocurre al inicio de cada mano con el *equity*, o puede nutrirse del sistema de notificaciones para obtener y calcular estos datos al momento, al igual que enviárselos a otros jugadores si fuera necesario o relevante.

Debido a que este sistema está formado por varios módulos especializados, podemos añadir y eliminar cualquiera de ellos para obtener mayor cantidad de estos, mayor rendimiento o cualquier otro requisito.

Su implementación resulta especialmente sencilla, pues solo deben mantener un recuento global de cualquier tipo de evento. En el caso del cálculo del *equity* empleados el método de Monte Carlo. Este método es altamente utilizado y conocido para este cálculo y se basa en la simulación aleatoria de las cartas restantes, es decir, teniendo en cuenta las cartas del jugador y las cartas comunitarias, para obtener una probabilidad de victoria dado una selección de cartas.

4.3 Integración de bots

La integración de los bots constituye una de las partes principales de la implementación del proyecto, ya que sobre esta estructura se construye el framework que permite incorporar diferentes tipos de jugadores automatizados. Como se ha explicado anteriormente, la comunicación entre la lógica de la partida y los jugadores humanos o bots se realiza mediante la interfaz *IPlayerNotificator* para utilizar el mecanismo de notificaciones de la partida y realizar una acción en base a todo el contexto de juego proporcionado

A partir de esta base común, cada bot adapta el proceso de toma de decisiones a las características de la tecnología utilizada. De esta forma, aunque todos interactúan con la partida mediante el mismo mecanismo, la información puede ser representada y procesada de manera diferente antes de obtener la acción que realizará el jugador.

La clase *Bot* implementa la interfaz `IPlayerNoticator`, que establece los métodos utilizados para comunicar al jugador los diferentes eventos que tienen lugar durante una partida. De esta forma, el bot no necesita conocer directamente cómo se desarrolla la partida, sino que recibe la información que necesita a través de una serie de notificaciones como podemos ver en la Figura 4-14 donde se detalla la relación entre estas dos clases.

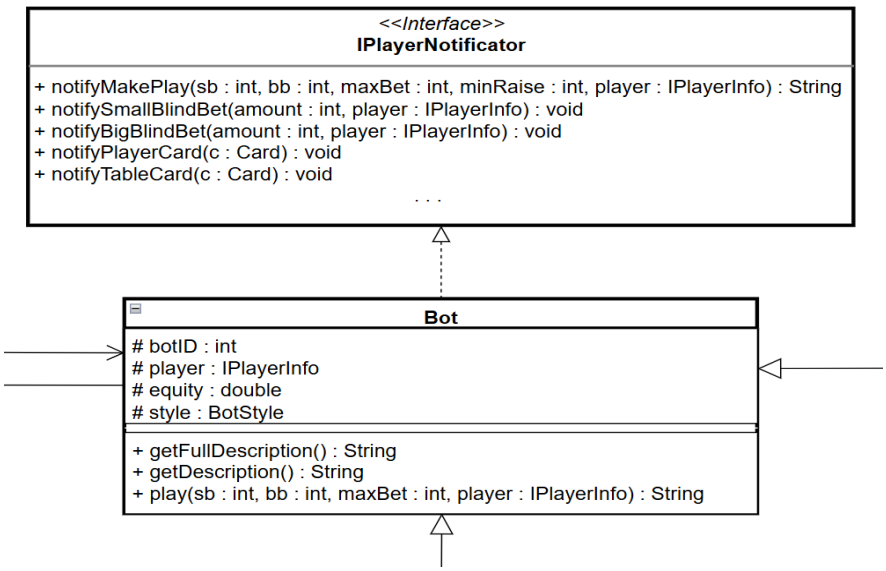


Figura 4-14. Relación entre IPlayerNotificator y Bot

La definición de *Bot*, parte por tanto de una serie de atributos comunes a cualquier jugador automático. Entre ellos se encuentran las cartas de la mesa, el historial de acciones, las ciegas y la apuesta máxima, el bote total y la equity calculada para la situación actual. Estos elementos permiten que las diferentes implementaciones del bot puedan tomar una decisión utilizando la misma información de la partida. En el código de la Figura 4-15 se muestra una parte de la definición de la clase y de los atributos utilizados para almacenar ese estado.

```
public abstract class Bot implements IPlayerNotificator {

	private List<Card> table;
	private List<String> actionHistory;

	private int _smallBlind;
	private int _bigBlind;
	private int _maxBet;
	private int _totalPot;
	private double _equity;

	private IPlayerInfo _player;
}
```

Figura 4-15. Atributos principales de la clase Bot

Una vez creado el bot, la información de la partida se va incorporando mediante los métodos definidos por *IPlayerNotificator*. Por ejemplo, cuando aparece una nueva carta comunitaria, se ejecuta `notifyTableCard()`, que añade dicha carta a la lista que mantiene el bot. De forma similar, `notifyTotalPot()` actualiza el valor del bote percibido por el bot y `notifyEquity()` almacena la equity correspondiente a la situación actual, como podemos ver en la Figura 4-16.

```
@Override
public void notifyTableCard(Card c) {
	table.add(c);
}
@Override
public void notifyTotalPot(int total) {
	_totalPot = total;
}
@Override
public void notifyEquity(double equity) {
	_equity = equity;
}
```

Figura 4-16. Actualización del estado interno del bot mediante notificaciones

Además de la información propia de la partida, el bot necesita conocer las acciones realizadas por los demás jugadores. Para ello, `notifyOtherPayerAction()` obtiene la última acción registrada del jugador y la incorpora al historial. De esta manera, el estado que recibe posteriormente el mecanismo de decisión incluye también la secuencia de acciones producidas durante la mano.

Para gestionar la participación de cada jugador en la partida se utiliza la clase Player, que constituye su representación dentro del sistema. Esta clase mantiene su estado, incluyendo información como las cartas, el dinero, la posición y las acciones realizadas, y además centraliza las operaciones que permiten interactuar con el resto de los componentes de la partida. En el caso de los bots, Player delega la obtención de la decisión en el objeto notificable asociado al jugador, como se muestra en la Figura 4-17.

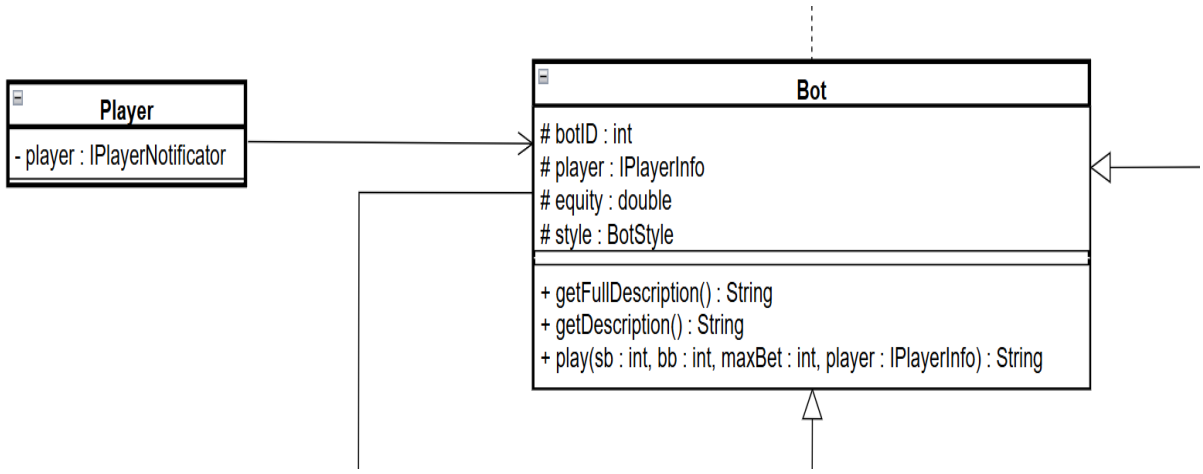


Figura 4-17. Relación clase Bot y Player

Cuando llega el turno del bot, comienza la segunda parte del proceso que es la obtención de una decisión. En este punto intervienen *Player* e *IPlayerNotificator*. La case Player, solicita la acción mediante el método `makePlay()`. Este método no contiene la lógica específica del bot, sino que delega la decisión en `_playerInfo.notifyMakePlay()` como se muestra en el código de la Figura 4-18.

```
public String makePlay(int sb, int bb, int maxBet, int minRaise) {

	final long start = System.nanoTime();
	String action = _playerInfo.notifyMakePlay(
		sb, bb, maxBet, minRaise, this
	);

	final long end = System.nanoTime();
	_experimentData.setDecisionTime(
		(end - start) / 1_000_000
	);
	return action;
}
```

Figura 4-18. Solicitud de una decisión desde Player

A partir de esta llamada, el control llega a `Bot.NotifyMakePlay()`. Este método constituye el punto común de entrada para la decisión de todos los bots. En lugar de implementar directamente aquí la estrategia concreta, Bot ejecuta el método abstracto `play()`, que será definido posteriormente por los distintos tipos de

bots. El flujo completo entonces, queda representado por la Figura 4-19, mientras que la implementación de `notifyMakePlay()` se muestra en el código de la Figura 4-25 explicado más adelante.

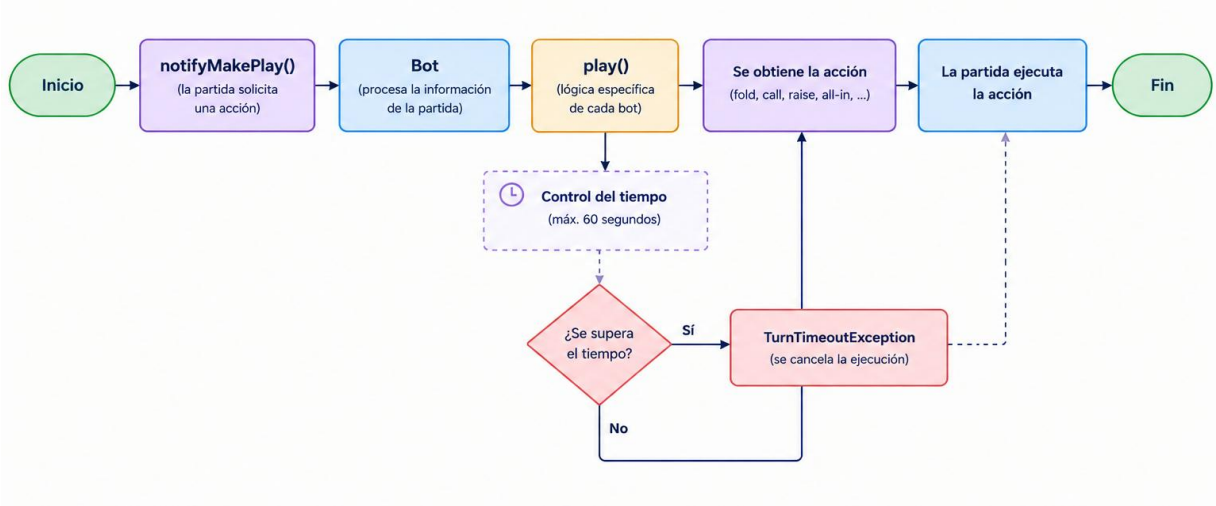


Figura 4-19. Flujo de toma de decisión de un bot

Este diseño permite separar el funcionamiento de la partida de la estrategia utilizada para decidir. Bot se encarga de recibir y mantener la información necesaria y de proporcionar el punto común desde el que se solicita una decisión, mientras que `play()` queda como método abstracto para que cada tipo de bot pueda implementar su propio mecanismo.

Una vez definida la clase Bot, se planteó cómo incorporar los diferentes tipos de jugadores que se querían utilizar en el proyecto. Como cada uno utiliza un mecanismo distinto para tomar sus decisiones, se crearon varias clases abstractas a partir de Bot que sirven como ejemplo de integración de cada tecnología. Entre ellas se encuentran `BotFSM`, `BotNN` y `BotLLM`, además de `BotLLMOnline` para los modelos de lenguaje que utilizan un servicio externo, como se detalla mediante el diagrama de la Figura 4-20.

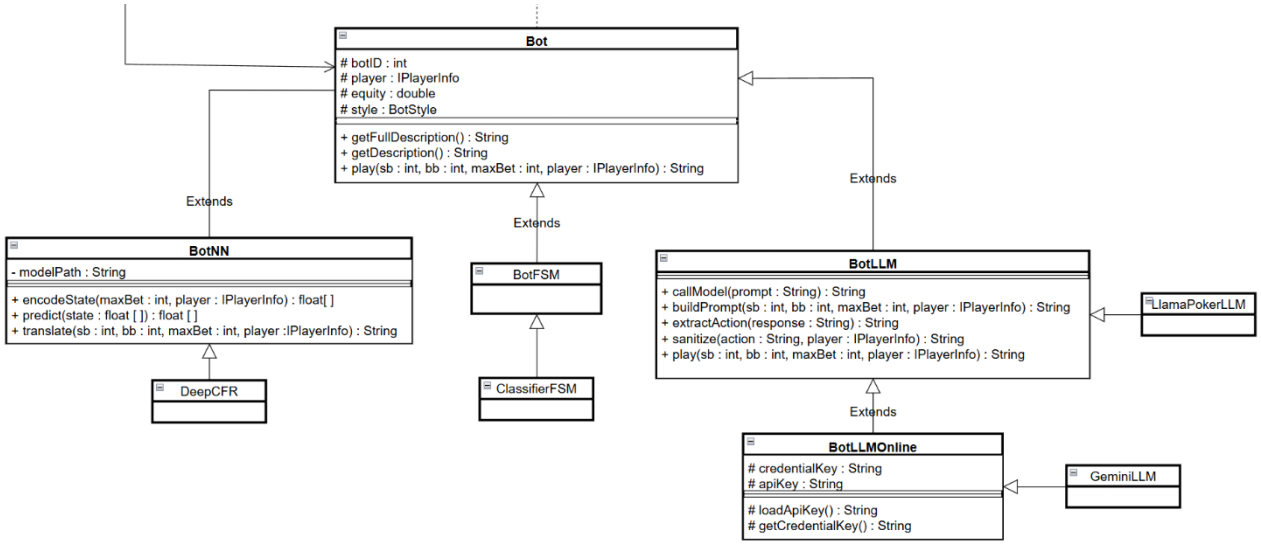


Figura 4-20. Jerarquía de clases para la integración de los diferentes tipos de bots

Esta jerarquía permite que todos los bots mantengan el mismo funcionamiento respecto a la partida, pero que la obtención de la decisión se adapte al mecanismo utilizado. Así, un bot basado en una máquina de estados, una red neuronal o un modelo de lenguaje puede recibir la misma información y ser solicitado de la misma manera, aunque internamente procese dicha información de forma diferente. Todas siguen el mismo funcionamiento general dentro de la partida: reciben la información necesaria, esperan a que llegue su turno y generan una acción.

En el caso de los bots basados en modelos de lenguaje (LLM), la información de la partida se transforma en un contexto textual que incluye los datos necesarios para que el modelo pueda analizar la situación y generar una respuesta. Esta respuesta se procesa posteriormente para obtener la acción de póker correspondiente. Para esta implementación se buscó un modelo de lenguaje específicamente entrenado para jugar partidas de póker No-Limit Texas Hold'em, con el objetivo de integrarlo en el framework y utilizarlo como un jugador más.

Se creó la clase `BotLLM`, que define el proceso que debe seguir un bot de este tipo. Como se muestra en el código de la Figura 4-21, primero se construye el prompt con la información de la partida, después se realiza la llamada al modelo y finalmente se procesa su respuesta hasta obtener una acción válida para el juego. Este proceso se encuentra definido en el método `play()` implementado por la clase.

```java
@Override
public String play(int sb, int bb, int maxBet, IPlayerInfo player)
throws IOException, TurnTimeoutException {

      _smallBlind = sb;
      _bigBlind = bb;
      _maxBet = maxBet;
      _player = player;

      String prompt = buildPrompt(sb, bb, maxBet, player);
      String response = callModel(prompt);
      String action = extractAction(response);
      String sanitized = sanitize(action, player);

      return sanitized;
}
```

Figura 4-21. Proceso de decisión de BotLLM

De esta forma, `BotLLM` se encarga de las partes comunes del proceso, mientras que la comunicación con el modelo se deja en `callModel()`. Este método se define como abstracto porque su implementación depende del modelo que se quiere utilizar. Así, para integrar un nuevo LLM no es necesario modificar el funcionamiento del bot, sino proporcionar la forma de realizar la llamada al modelo correspondiente.

En el caso del modelo de lenguaje online, el proceso es equivalente, pero la decisión se obtiene mediante la comunicación con un proveedor externo. Para ello se creó `BotOnline`, amplía `BotLLM` y que se encarga de gestionar las credenciales necesarias para realizar la peticiones al servicio. Estas credenciales se cargan desde el archivo credentials.json, mientras que cada implementación concreta indica mediante `getCredentialKey()` cuál debe utilizar.

Por ejemplo, para integrar Gemini, se creó la clase *`GeminiLLM`*, que hereda de `BotLLMOnline`, En ella se configura el modelo que se utilizará y se implementa `callModel()` para realizar la petición al servicio. De esta manera, la parte específica

del proveedor queda separada del resto del proceso definido en `BotLLM`, como se observa en el código de la Figura 4-22.

```
@Override
protected String callModel(String prompt) throws
TurnTimeoutException {


	String response = GameType.FOLD_ACTION_FULL;
	try {
		response = gemini.chat(prompt);
	}
	catch(Exception e) {
		log.error(
		"GeminiLLM could not be reached!
		Action made in this case: FOLD");
	}
	return response;
}
```

Figura 4-22. Implementación de la llamada al modelo en GeminiLLM

El bot basado en una red neuronal (NN) utiliza, en cambio, una representación numérica del estado de la partida. La información disponible se transforma en los datos de entrada esperados por el modelo y, una vez realizada la predicción, su salida se interpreta para obtener la acción correspondiente. Para ello se creó `BotNN`, que proporciona el proceso general de preparación del estado, predicción y traducción del resultado, como podemos ver en el código de la Diseño

En este caso, la diferencia respecto al LLM se encuentra principalmente en la forma de representar y procesar la información. Mientras que `BotLLM` construye un prompt y trabaja con una respuesta textual, `BotNN` transforma el estado de la partida en un vector numérico y obtiene una predicción del modelo. Este proceso queda reflejado en el método `play()`, donde se realizan estas operaciones de forma consecutiva. Podemos ver un ejemplo en la Figura 4-23.

```java
@Override
public String play(int sb, int bb, int maxBet, IPlayerInfo player)
throws IOException {
	float[] state = encondeState(maxBet, player);
	float[] prediction = predict(state);
	String response = translate(sb, bb, maxBet, player, prediction);

	return response;
}
```

Figura 4-23. Proceso de decisión de BoNN

Finalmente, el bot basado en una máquina de estados (FSM) fue desarrollado específicamente dentro del proyecto. Para facilitar su integración se creó `BotFSM`, que sirve como base para implementaciones de este tipo. La estrategia se define mediante estados y condiciones que utilizan directamente la información de la partida para determinar en qué situación se encuentra el jugador y qué acción debe realizar.

Como ejemplo de esta integración se desarrolló `ClassifierFSM`, una implementación que determina la acción del jugador a partir del valor de las cartas que tiene en su mano y de las cartas que se encuentran sobre la mesa.

En cada turno, `ClassifierFSM` obtiene los valores de las cartas disponibles y calcula una puntuación que representa la situación actual del jugador. Para ello, primero se suman los valores de las cartas de la mesa y de las dos cartas del jugador. Posteriormente, esta puntuación se utiliza para determinar la acción que debe realizar el bot. La lógica principal se puede observar en el código de la Figura 4-24.

```
@Override
public String play (int sb, int bb, int maxBet, IPlayerInfo player) throws
IOException {
                                        . . .
    else if (totalValue < points_per_action) {
        return GameType.FOLD_ACTION_FULL;
    }
    else if (totalValue < 2 * points_per_action) {
        return GameType.CALL_ACTION_FULL;
    }
    else if (totalValue < 3 * points_per_action) {
        return GameType.RAISE_ACTION_FULL + " " + String.valueOf(2 * maxBet);
    }
    else {
        return GameType.ALL_IN_ACTION_FULL;
    }
}
```

Figura 4-24. Clasificación de la acción en ClassifierFSM

Por tanto, aunque el mecanismo utilizado para obtener la decisión es diferente en cada caso, el punto de entrada y la salida de todos los bots son comunes. Todos reciben la información de la partida mediante la estructura definida en Bot y terminan proporcionando una acción que puede ser interpretada por la lógica de la partida. La creación de estas diferentes implementaciones permitió comprobar que la estructura desarrollada puede utilizarse con mecanismos de toma de decisiones muy diferentes dentro de una misma partida, manteniendo el funcionamiento general de la aplicación.

Una vez explicada la forma en la que cada tipo de bot obtiene su decisión, es necesario considerar también cómo se controla su ejecución dentro de la partida. Los diferentes mecanismos pueden presentar tiempos de respuesta muy distintos, especialmente en el caso de los modelos de inteligencia artificial, donde la generación de una decisión puede requerir varios segundos. Pra evitar que una decisión bloquee indefinidamente el desarrollo de una partida, esta gestión se realiza de forma común en la clase Bot.

Para ello, la clase Bot utiliza un *`ExecutorService`* que permite ejecutar el método `play()` de forma independiente. Cuando llega el turno del bot,

`notifyMakePlay()` envía la tarea al ejecutor y espera su resultado durante un tiempo máximo de 60 segundos, como podemos ver en el código de la Figura 4-25.

```
@Override
public String notifyMakePlay(
int sb, int bb, int maxBet, int minRaise, IPlayerInfo player)
throws IOException, TurnTimeoutException {

	// Time before waiting for the bot response
	long startTime = System.currentTimeMillis();

	// Wait bot response with timeout
	String action = GameType.FOLD_ACTION_FULL;
	Future<String> future = _playExec.submit(
		() -> play(sb, bb, maxBet, player)
	);
	try {
		// El bot tiene como máximo 60 segundos para responder
		action = future.get(SECONDS_TIMEOUT, TimeUnit.SECONDS);
	}
	catch (TimeoutException e) {
		future.cancel(true);
		throw new TurnTimeoutException();
	}

	return action;
}
```

Figura 4-25. Método notifyMakePlay() de la clase Bot

De esta forma, el mecanismo de control es independiente de la tecnología utilizada para tomar la decisión. Un `BotLLM`, por ejemplo, puede estar esperando la respuesta de un modelo, mientras que un `BotNN` puede realizar una predicción local o un `BotFSM` puede resolver su decisión directamente mediante sus reglas. En todos los casos, la partida utiliza el mismo mecanismo para solicitar la acción y controlar el tiempo disponible para obtenerla.

Una vez obtenida la decisión, esta vuelve a la lógica de la partida para ser interpretada y ejecutada. Aunque internamente cada bot puede utilizar un procedimiento diferente, el resultado de `play()` se transforma en un `String` con una acción reconocida por el sistema. En el caso de los bots basados en modelos de lenguaje, por ejemplo, la respuesta obtenida del modelo se procesa mediante *`extractAction()`* y *`sanitize()`* antes de ser devuelta. Así una respuesta como <action>RAISE 200</action> se transforma en raise 200.

A partir de este resultado, la aplicación identifica la acción solicitada y realiza la operación correspondiente sobre el jugador. De esta forma, el bot puede indicar acciones como retirarse (fold), pasar(check), igualar(call) subir la apuesta (raise) o apostar todas sus fichas (all-in), que son posteriormente procesadas por la lógica de la partida. Podemos ver un diagrama sobre esto en Figura 4-26.

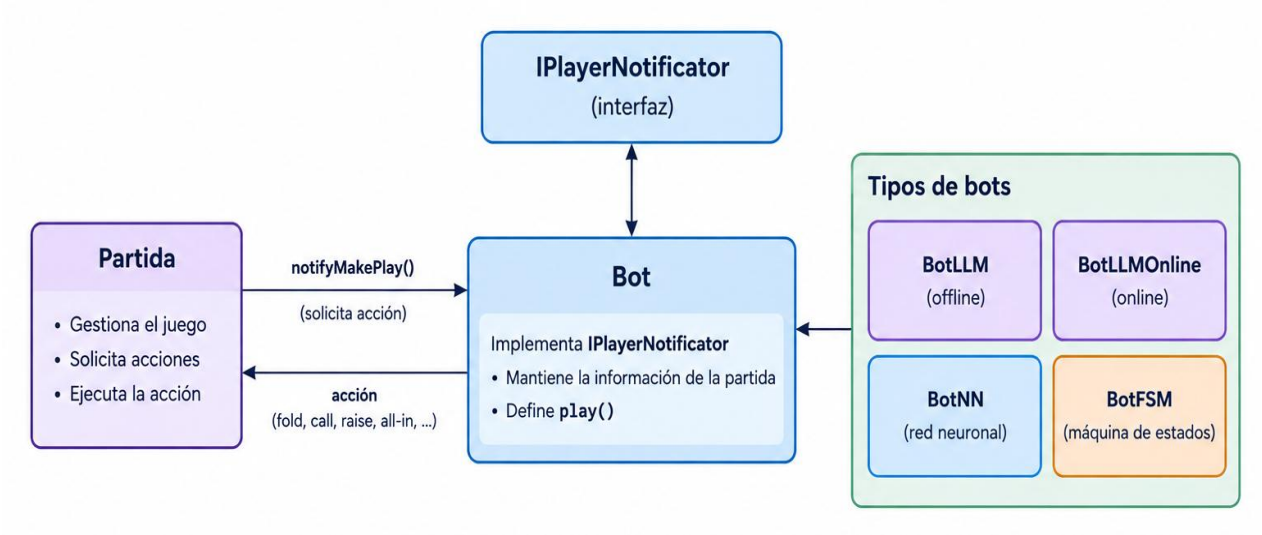


Figura 4-26. Arquitectura de la integración de bots

La utilización de esta estructura permite que los bots sean tratados como un jugador más dentro de la aplicación. La lógica de la partida no necesita conocer el mecanismo concreto utilizado para tomar la decisión, sino únicamente solicitarla y procesar la acción obtenida.

Por tanto, la implementación mantiene una separación clara entre la obtención de la decisión y la ejecución de la partida. Cada bot puede utilizar una tecnología y una representación de información diferente, mientras que la comunicación con la lógica de póker se mantiene común.

Esta estructura no solo permite utilizar los bots incorporados durante el proyecto, sino que proporciona una base sobre la que incorporar nuevas implementaciones. Para ello, el nuevo bot debe adaptarse al mecanismo común de comunicación y proporcionar su propia lógica de decisión, mientras que el resto de la partida permanece independiente de la tecnología utilizada.

Como resultado, los diferentes bots pueden participar en las mismas partidas bajo unas condiciones de juego comunes, permitiendo posteriormente registrar sus decisiones y resultados y tiempos de respuesta. Este es en realidad el objetivo central del proyecto, ya que la misma estructura que permite integrar los bots hace posible enfrentarlos entre sí y obtener la información necesaria para comparar posteriormente su comportamiento y rendimiento

4.3.1 Ejemplo de integración LLM Online (OpenAI)

A continuación, mostramos una guía para integrar un nuevo LLM Online, utilizando el modelo de OpenAI.

1. Seleccionar un LLM adecuado: normalmente los servicios de LLMs requieren obtener una clave de acceso para poder comunicar un programa informático con el servicio para hacer peticiones. Normalmente estos servicios son accesibles mediante una suscripción, aunque existen modelos o plataformas que ofrecen estas claves de forma gratuita con un límite de peticiones bastante reducido, por lo que para mantener partidas de larga duración es más conveniente tener un modelo de pago. En nuestro caso, utilizaremos un modelo de OpenAI.
2. Obtener una clave de acceso: solicitaremos la clave y la copiaremos en algún lugar seguro para poder acceder a ella más tarde.
3. Crear o modificar el archivo credentials.json: para que la aplicación servidor pueda comunicarse con estos LLMs mediante la clave de acceso, deberá crearse o modificarse el archivo en la ruta `tfgpoker/server/src/main/resources` para poder autenticarse en el servicio y obtener respuestas. Este archivo no debe ser compartido ni filtrado bajo ningún concepto, ya que cualquiera podría tener acceso a esta clave y realizar peticiones sin ningún tipo de control. Este archivo de credenciales debe tener el formato nombre-clave en el que el nombre puede tener el formato que nosotros deseemos para identificar cada servicio y la clave será la que hayamos obtenido en nuestro servicio.

Es importante recordar el nombre que le hemos asignado puesto que en pasos posteriores deberemos buscarlo, tal y como podemos ver en la Figura 4-27.

```
{
    "GEMINI_API_KEY" : "a54bdfAGIr32ogjcJgWcB",
    "OPENAI_API_KEY" : "Pjgf557MN8klFsLK43NfP"
}
```

Figura 4-27. Formato del archivo credentials.json

4. Crear una nueva clase que herede de `BotLLMOnline`: crearemos una nueva clave Java mediante un fichero llamado BotOpenAi.java en el paquete `tfgpoker/server/src/main/java/com/ucm/server/players` que herede de `BotLLMOnline`. Tras esto, podremos abrir este archivo en cualquier IDE que nos permita ver qué métodos necesitaremos implementar.
5. Crear el identificador del nuevo bot: En el módulo `common` debemos añadir un identificador de nuestro bot con un valor que no tenga otro bot. Recomendamos seguir el formato que tiene el archivo como se muestra en la Figura 4-28.

```
// Bots identifier
public static final int BOT_GEMINI = 10000;
public static final int BOT_LLAMA = 10001;
public static final int BOT_NN_MODEL_1 = 10002;
public static final int BOT_FSM_1 = 10003;
public static final int BOT_OPENAI = 10004;
```

Figura 4-28. Identificador del bot

6. Sobrescribir o utilizar la implementación de los métodos relevantes: según la implementación de nuestro modelo, podremos utilizar o sobrescribir los métodos que nos hagan falta para nuestro LLM. Los principales métodos que podemos modificar son `buildPrompt()`, para construir el prompt que se enviará al modelo; `callModel()`, para enviar la petición a nuestro modelo; `extractAction()`, para obtener la respuesta final simplificada y `sanitize()`, para realizar algún tipo de limpieza necesaria de la respuesta final.

La implementación puede ser similar a la de la Figura 4-29, aunque puede variar según nuestras intenciones o necesidades de nuestro bot.

```
public class BotOpenAI extends BotLLMOnline {

      @Override
      protected String callModel(String prompt)
      throws IOException, TurnTimeoutException {
            return call_open_ai(prompt);
      }
      @Override
      protected String buildPrompt(
      int sb, int bb, int maxBet, IPlayerInfo player){
            return String.format(
                "You are an expert poker player.\n
                Blinds are %d/%d, last bet was %d\n
                Your cards are %s and %s.",
                sb, bb, maxBet,
                 player.getCards()[0].toString(),
                 player.getCards()[1].toString());
      }
      @Override  // No extract action
      protected String extractAction(String text) { return text; }
      @Override  // No sanitize
      protected String sanitize(String action, IPlayerInfo player)
      { return action; }
                                  ...
}
```

Figura 4-29. Ejemplo de código básico de bot LLM online

7. Implementar el resto de métodos: implementar los métodos de acceso a la clave y otros métodos menor relevantes como la descripción de este, obteniendo algo similar al código mostrado en la Figura 4-30.

```
public class BotOpenAI extends BotLLMOnline {

	private static final int OPENAI_ID = GameType.BOT_OPENAI;
				...
	public BotOpenAI(int botID, BotStyle style){
		super(botID, style);
	}

	@Override public Bot create(BotStyle style) {
		return new BotOpenAI(OPENAI_ID, style);
	}

	@Override public String getFullDescription() {
		return "Detailed description";
	}

	@Override public String getDescription() {
		return "Short description";
	}
}
```

Figura 4-30. Ejemplo de código de acceso a clave y otros métodos

Añadir un nuevo `BotDescriptor` en la clase `BotRegistry` para que el servidor pueda detectar el nuevo tipo de bot, permitiendo o no la posibilidad de incluir estilos en el bot: Tras completar la lógica de nuestro bot, deberemos registrarlo en la clase `BotRegistry` para que el servidor pueda transmitir la información a la aplicación cliente y pueda ser seleccionado. Uno de los aspectos importantes aquí es permitir o no instanciar nuestro bot LLM con estilos de juego tipo *maniac, loose-aggressive* o cualquier otro.

8. Registrar el nuevo bot en la clase `BotManager`: Además de registrarlo en la clase `BotDescriptor` deberemos registrarlo en la clase `BotManager` para que el servidor pueda reconocerlo.
9. Recompilar todos los módulos del proyecto: deberemos recompilar nuestra aplicación servidor y al lanzarla de nuevo, podremos ver desde la aplicación cliente nuestro bot y verlo jugar en nuestras partidas. Para instalar las

dependencias utilizaremos el comando `mvn clean install -DskipTests` y para compilar el proyecto utiliaremos el comando `mvn clean compile`.

10. Lanzar ambas aplicaciones y probar el bot: Tras comprobar la correcta integración de nuestro bot, podremos hacer todas las pruebas que deseemos para evaluar a nuestro bot. Podremos probarlo en diferentes entornos y estudiar el funcionamiento de nuestro bot para realizar los ajustes necesarios.

4.3.2 Ejemplo de integración de Red Neuronal

A continuación, mostramos una guía para integrar un nuevo bot mediante una red neuronal. En esta sección no nos centraremos en la arquitectura de esta red, sino que se pondrá foco en cómo integrarla al framework.

1. Seleccionar una arquitectura claramente definida para nuestra red neuronal: es realmente importante saber el número de entradas y salidas de nuestro modelo, al igual que la codificación adecuada de estos. Estos datos serán altamente relevantes para alimentar a nuestra red e interpretar su respuesta.
2. Exportar nuestro modelo con formato ONNX: Tras evaluar nuestro modelo propio o de terceros, deberemos exportarlo en formato .onnx mediante la librería de Python o alguna otra que nos lo permita. Es muy importante que este archivo contenga toda la información de nuestro modelo, tanto la arquitectura como los pesos.
3. Añadir el archivo .onnx de nuestro modelo en el directorio `/tfgpoker/server/src/main/resources/models_nn`: copiamos este archivo .onnx en el directorio del servidor para poder acceder a él.
4. Creamos la clase NuevaRed: mediante un archivo .java en el directorio `/tfgpoker/server/src/main/java/com/ucm/server/players/NuevaRed.java` que herede de la clase padre BotNN
5. Implementaremos lo métodos requeridos: para completar la implementación básica de nuestra red neuronal, completarmos el código de los siguientes métodos:

    - Constructor: Debemos llamar al constructor de la clase padre y proporcionar el nombre del archivo .onnx de nuestro modelo

- *encodeState*: método que recibe el contexto del juego y su jugador *virtual* para codificar toda la información necesaria para alimentar a nuestra red. Es muy importante que la codificación devuelta por este método coincida a la perfección con la implementación.

- *predict*: método que recoge la codificación del método `encodeState()` para alimentar a la red. Este método debe devolver la respuesta de esta.

- *translate*: este método recibe la predicción hecha por la red y debemos realizar la traducción de la acción *cruda* de la red a la acción que representa en la implementación del juego de póker.

6. Añadir un nuevo `BotDescriptor` en la clase `BotRegistry` para que el servidor pueda detectar el nuevo tipo de bot, sin permitir la posibilidad de incluir estilos en el bot: Tras completar la lógica de nuestro bot, deberemos registrarlo en la clase `BotRegistry` para que el servidor pueda transmitir la información a la aplicación cliente y pueda ser seleccionado. Uno de los aspectos importantes aquí es permitir o no instanciar nuestro bot mediante la red neuronal.
7. Registrar el nuevo bot en la clase `BotManager`: Además de registrarlo en la clase `BotDescriptor` deberemos registrarlo en la clase `BotManager` para que el servidor pueda reconocerlo.
8. Recompilar todos los módulos del proyecto: deberemos recompilar nuestra aplicación servidor y al lanzarla de nuevo, podremos ver desde la aplicación cliente nuestro bot y verlo jugar en nuestras partidas. Para instalar las dependencias utilizaremos el comando `mvn clean install -DskipTests` y para compilar el proyecto utilizaremos el comando `mvn clean compile`.
9. Lanzar ambas aplicaciones y probar el bot: Tras comprobar la correcta integración de nuestro bot, podremos hacer todas las pruebas que deseemos para evaluar a nuestro bot. Podremos probarlo en diferentes entornos y estudiar el funcionamiento de nuestro bot para realizar los ajustes necesarios.

4.4 Arquitectura distribuida

En esta sección desarrollaremos la implementación de nuestra aplicación distribuida, la cual está dividida en los módulos *common*(comunes), *server*(aplicación servidor) y *client*(aplicación cliente). Hablaremos de sus responsabilidades, alcance y otros detalles relevantes.

4.4.1 Common

Antes de entender la necesidad de este módulo que ambas aplicaciones utilizan, debemos entender que para que la aplicación cliente y servidor puedan cooperar es necesario un orden estricto y una serie de definiciones que sean fácilmente modificables e implementables en ambas aplicaciones, ya que estas deben estar completamente sincronizadas en todo momento. Para ello este módulo contiene multitud de definiciones y métodos relevantes en ambas aplicaciones.

Podemos entender que este módulo define, principalmente, una serie de números que identifican un tipo de *paquete* y la información adicional de este.

Estos paquetes, formados como muestra la Figura 4-31, son un conjunto de información codificada que envían un mensaje concreto y que, al recibir esa petición concreta, la aplicación receptora deberá interpretarlos de la misma forma para la correcta comunicación de la aplicación.

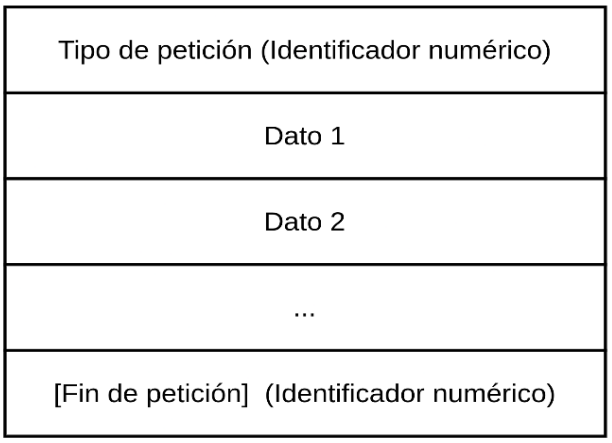


Figura 4-31. Estructura de paquetes. Datos entre [ ] son opcionales

Un ejemplo real de este tipo de paquetes puede ser el de la información enviada por el servidor para indicar las cartas en mano de un jugador cualquiera mediante su identificador, como podemos ver en la Figura 4-32.



Figura 4-32. Paquete PLAYER_CARDS que notifica las cartas de un jugador

Además de todas estas definiciones, este módulo incluye una serie de funciones de envío y recepción de información que sirve también como forma de mantener una estructura clara y ordenada para el mantenimiento de ambas aplicaciones.

También se definen múltiples estructuras de datos relevantes para el juego de póker, como las cartas o el rol de los jugadores, con el objetivo de proveer a ambas aplicaciones con estas estructuras y todas sus funcionalidades y definir en estas cómo su información interna debe ser codificada e interpretada para su envío y reconstrucción a ambos lados de la comunicación.

Por último, es importante a tener en cuenta que este módulo debe servir en todo momento como auxiliar para las demás partes y nunca como un módulo con un papel relevante en las comunicaciones o el desarrollo de la aplicación, además de que cualquier cambio en ese módulo requerirá la posterior compilación de ambas aplicaciones para mantenerse sincronizadas.

4.4.2 Aplicación servidor

La aplicación servidor es el verdadero eje central de toda la arquitectura y funciona como única autoridad frente al juego de póker. Esto quiere decir que es aquí donde ocurre realmente toda la ejecución de juego y el resto de componentes que quieran acceder a la información, como la aplicación cliente, deberán solicitarla al servidor.

El desarrollo del servidor podemos dividirlo en varias fases y, antes de desarrollarlas, debemos entender que esta implementación solo permite la ejecución de una partida de póker simultánea, aunque sí permite la conexión de multitud de clientes. El objetivo de esto es reducir en gran cantidad la complejidad de este sistema y, sobre todo, ajustar la implementación al alcance del proyecto. Aunque este diseño reduce significativamente la complejidad global, esto no elimina la multitud de responsabilidades propias de un servidor autoritativo como puede ser la gestión de múltiples conexiones simultáneas, esperas indefinidas, desconexiones repentinas, bloqueos o errores accidentales o intencionados en el paso de mensajes.

Estas problemáticas han sido resueltas mediante *sockets TCP*, para asegurar la correcta transmisión y recepción de mensajes; el uso de *threads* y *locks* para permitir conexiones simultáneas y evitar *race conditions*, temporizadores para evitar esperas indefinidas en el turno de los jugadores y comprobaciones auxiliares de los datos recibidos por los clientes. Sabiendo esto, podemos continuar con el desarrollo completo normal del servidor:

1. Conexiones remotas de clientes: en esta primera fase del servidor todos los clientes procederán a conectarse mediante la petición PETITION_PLAYER_NAME aportando un nombre elegido por estos clientes para poder ser identificados. Este nombre servirá posteriormente para ser asignado a su jugador virtual, pero no como forma de identificación única.

   Tras la conexión y aceptación de un nuevo cliente, el servidor se mantendrá a la espera para poder recibir nuevas peticiones de este usuario, bien será para unirse a una partida, crear una nueva si no existe una o simplemente desconectarse.

2. Creación de partida: el usuario que desee crear una partida, si ya no existe una, deberá enviar a continuación de la petición PETITION_CREATE_GAME junto a toda

la configuración deseada para esta partida. Estos parámetros son el nombre de la sala, el nombre del creador de esta, si permite bots, si se desea incluir un espectador, la cantidad de fichas iniciales de cada jugador, ciegas estáticas o dinámicas y, en el segundo caso, el intervalo y porcentaje de aumento, los bots y sus configuraciones y el estilo del reverso de las cartas y de la mesa de juego.

Tras recibir esta información y comprobar que no existe una partida ya creada o que la configuración sea incorrecta, se procederá a crear los recursos necesarios para ello y se le enviará al cliente su identificador de partida y se pasará a la fase de *prepartida* o sala de espera. En ese punto, el cliente o ahora llamado *host* tendrá que esperar hasta que haya al menos otro jugador o bot seleccionado en la configuración en la sala para poder comenzar la partida cuando él desee.

3. Unión a una partida: los usuarios conectados podrán unirse a la partida existente mediante el paquete PETITION_JOIN_GAME y se les enviará toda la información necesaria para poder saber qué bots u otros jugadores se encuentran en la sala de espera. Estos usuarios no podrán hacer nada más que esperar a que el host comience o se salga, lo que producirá la cancelación de la partida.

   Si no existe una partida en espera o existe una ya comenzada, se le notificará al usuario y no tendrá lugar ninguna otra acción.

4. Desarrollo de partida: Tras el final de la prepartida comenzará verdaderamente el desarrollo de la partida de póker y los jugadores y bots podrán realizar acciones en sus respectivos turnos.

   A partir de aquí el desarrollo del juego es el esperado y todos los clientes se mantienen a la espera de actualizaciones del servidor. La única información que el servidor espera de los clientes es la acción que desea realizar y viene codificada como texto llano del tipo “call”, “fold”, “check”, “allin” y “raise <amount>”, donde <amount> es la cantidad numérica a la que se desea subir en ese instante. La espera de esta respuesta no debe ser indefinida y el mecanismo del servidor para evitar esto es mediante un temporizador que, al expirar, ignora la respuesta del cliente y realiza un *fold* para pasar al siguiente jugador.

   Por otro lado, en un sistema distribuido siempre pueden ocurrir desconexiones por cualquier motivo y, en estos casos, hemos optado por mantener a estos jugadores

totalmente ignorados durante el resto de la partida y sin un mecanismo de reconexión, además de la cancelación total de partida en caso de quedarse un único jugador remoto conectado a esta. Es decir, si un cliente se desconecta este no recibiría turno ni recibiría información, de la misma forma que ocurre con un jugador eliminado de la partida tras perder todas sus fichas.

Esta decisión sigue en la línea de mantener una implementación sencilla y centrada una aplicación distribuida más centrada en un uso privado o restringido al público masivo.

Por último, en caso de la cancelación o finalización normal de la partida, todos los recursos serán liberados y el servidor se mantendrá listo para la creación de una nueva partida.

4.4.3 Aplicación cliente

La interacción de la aplicación cliente es bastante reducida y en su mayoría del tiempo se dedica a la recepción de información durante la partida de póker, aunque primero podemos destacar ciertas interacciones relevantes que debe iniciar el cliente:

1. Identificación del cliente: este tipo de petición denominada PETITION_PLAYER_NAME se envía al inicio de la aplicación junto con el nombre seleccionado por el cliente y es el que servirá como identificación visual para él mismo y el resto de jugadores durante la partida ya que no tiene por qué ser único. Este nombre debe tener una longitud de entre 3 y 10 caracteres. De lo contrario, el servidor avisará al cliente del problema y no aceptará su conexión, obligándole a reintentarlo.
2. Crear partida: esta petición se envía mediante el paquete PETITION_CREATE_GAME junto a la configuración de la partida y servirá al servidor para saber qué tipo de ajustes y recursos utilizar para satisfacer las restricciones impuestas por el creador de la partida. Tras enviar esta información correctamente, el cliente recibirá el visto bueno del servidor junto a la información adicional de la prepartida y su identificador, además de avisarle de que él será el cliente host de la partida, lo que significa que él deberá comenzar y que, en caso

de que se desconecte de forma abrupta, la partida se cancelará y todos los jugadores que quedan en ella.

Durante la prepartida pueden unirse y salirse otros jugadores por lo que, si alguno de estos eventos ocurriese, esta notificación también se recibiría y se actualizará la información de la sala con la nueva distribución de jugadores.

3. Unirse a partida: esta petición es enviada por un cliente que, idealmente, quiere unirse a una partida existente mediante el paquete PETITION_JOIN_GAME. Esta petición no requiere de información inicial y, tras ella, el usuario podrá comenzar a recibir la información de la prepartida y ver qué otros jugadores se unen.

   Naturalmente, pueden unirse y salir otros jugadores por lo que, si alguno de estos eventos ocurriese, esta notificación se recibiría y se actualizará la información de la sala con la nueva distribución de jugadores.

4. Comenzar partida: esta petición mediante el paquete EVENT_GAME_STARTS solo podrá ser enviada por el host de una partida para tener éxito. Tras este mensaje el usuario host recibirá el paquete CONFIRMATION_PLAYER_STARTS, al igual que el resto de jugadores, la confirmación del inicio de partida y dará comienzo. Después de esto, la partida de póker ocurrirá de forma esperada.
5. Acción de juego: esta petición es enviada durante el desarrollo de póker cuando es el turno del usuario tras recibir la notificación del servidor que nos indica que debemos realizar una acción. Esta respuesta deberá ocurrir dentro de un rango de tiempo configurado por el creador de la partida ya que, si no hacemos nada, el servidor no tendrá en cuenta nuestra respuesta. Esta petición que recibe una jugada no provendrá, naturalmente, de jugadores que hayan hecho fold, all-in, se hayan desconectado o hayan perdido todas sus fichas previamente.

   Además de toda la información enviada por el cliente hay muchos otros datos que recibe durante el transcurso de la prepartida o la partida, estos datos pueden ser cartas colocadas en la mesa, sus propias cartas, las acciones realizadas por otros jugadores y otros muchos eventos ocurridos y que es necesario que el cliente los reciba correctamente para mantenerse sincronizado con el estado del juego en el servidor.

4.5 Monitorización de partidas

La monitorización de las partidas es otro de los elementos importantes del diseño de la aplicación, especialmente debido a que los experimentos realizados posteriormente necesitan disponer de información detallada sobre lo ocurrido durante cada mano. Por ello, se diseñó un sistema de registro que permite almacenar de forma estructurada los acontecimientos relevantes de las partidas, como las acciones realizadas por los jugadores o los cambios producidos en el estado del juego.

El registro está integrado directamente en la lógica de la partida, de forma que la información se almacena en el mismo momento en el que ocurre cada evento. Para ello se utiliza la clase `PokerHistory`, encargada de mantener el historial de la mano y registrar los acontecimientos relevantes. El historial sigue una estructura similar a la utilizada por PokerStars, por lo que se representa de forma clara y ordenada el desarrollo de cada mano.

Los puntos de monitorización se encuentran distribuidos a lo largo de las diferentes fases de la partida. De esta forma, al comenzar una mano se registra la información inicial de los jugadores y de la mesa, y posteriormente se registran las cartas que aparecen y las acciones realizadas durante cada ronda. Las acciones de los jugadores se registran en el momento en el que se ejecutan, permitiendo conocer qué jugador realiza cada acción y en qué momento de la mano.

Para realizar este registro se utiliza el sistema de login proporcionado por Java mediante Log4J, configurado para dirigir los mensajes generados durante la partida a los ficheros correspondientes. El diseño permite identificar cada partida y cada mano mediante su propio contexto, de manera que los acontecimientos quedan organizados en ficheros independientes y pueden ser procesados posteriormente,

Además del historial de acciones, se registra información específica para los experimentos. Durante el desarrollo de la mano se almacenan datos como el equity de los jugadores en las diferentes fases y al finalizar información como el tipo de jugador, modelo, estilo, posición, stock inicial y final, etc.

La separación existente entre la lógica de la partida y la toma de decisiones de los jugadores/bots facilita identificar y registrar las acciones de cada jugador.

Capítulo 5 - Estudio empírico

Una vez desarrollado el framework y comprobado su funcionamiento mediante diferentes pruebas, se realizó un estudio empírico con el objetivo de evaluar su comportamiento en un entorno real de juego y analizar las diferencias entre los distintos tipos de jugadores implementados. Este estudio constituye una parte fundamental del proyecto, ya que permite evaluar el funcionamiento del framework más allá de las pruebas realizadas durante su desarrollo y validar esta integración en un escenario real de juego. El análisis del comportamiento de los jugadores constituye una forma de validar que las diferentes implementaciones pueden participar conjuntamente en una misma partida y que el framework permite mantener su funcionamiento independientemente de la tecnología utilizada.

Los resultados obtenidos durante las partidas fueron recopilados y posteriormente procesados para obtener diferentes métricas y estadísticas que permitieran comparar el comportamiento de los jugadores. A partir de estos datos se realizó el análisis que se presenta en las siguientes secciones.

5.1 Proceso experimental

El estudio experimental se planteó para analizar el comportamiento de los bots desde más de un punto de vista. Por un lado, se realiza un análisis cuantitativo, basado en los datos generados durante las partidas, que permite medir cómo juegan los bots, qué decisiones toman y qué resultados obtienen. Por otro, se incorpora un análisis cualitativo a partir de la experiencia de los jugadores humanos, con el que se busca conocer cómo perciben el comportamiento de los bots y si las diferencias observadas en los datos también se reflejan durante el juego. Ambas perspectivas se complementan para responder no solo a qué bot obtiene mejores resultados, sino también a cómo juega, cómo cambia su comportamiento y qué impresión genera frente a un jugador humano.

Para estudiar estas cuestiones se decidió no limitar la experimentación a un único tipo de partida. La variedad de jugadores disponibles permitía construir escenarios muy diferentes y aprovechar esta diversidad para poner a prueba los bots

en situaciones distintas. Así, se diseñaron experimentos heterogéneos con jugadores humanos de diferentes niveles, mesas mixtas de humanos y bots, partidas exclusivamente entre bots con enfrentamientos directos entre diferentes inteligencias artificiales. A esto se añadió la posibilidad de modificar los estilos de juego de los LLM, lo que permitió estudiar también si una misma tecnología se comporta de forma diferente al cambiar la estrategia que utiliza.

Esta combinación de escenarios permite observar el comportamiento de los bots bajo condiciones que responden a preguntas diferentes. Enfrentarlos a humanos permite introducir la variabilidad propia de un jugador real, enfrentarlos entre ellos facilita realizar comparaciones bajo condiciones más controladas y modificar sus estilos permite comprobar hasta qué punto la estrategia configurada se refleja realmente en sus decisiones. De esta forma, los experimentos no se plantearon como pruebas independientes, sino como distintas perspectivas de un mismo estudio.

Diseñar estas pruebas implicó determinar previamente qué jugadores participaron en cada escenario, qué estilos utilizarían, cuántas partidas se realizaron y qué condiciones debían mantenerse constantes. También fue necesario controlar la distribución de las cartas mediante seeds, evitando que una determinada secuencia de cartas condicionara las comparaciones entre bots. En las pruebas con jugadores humanos, además hubo que organizar las sesiones y contar con participantes que permitieran incorporar diferentes niveles de experiencia al estudio.

Una vez definidas las pruebas, fue necesario preparar el framework para poder ejecutarlas y registrar de forma detallada la información generada durante cada partida. Para ello, desarrollamos y configuramos el sistema de logs, encargado de almacenar la información de las manos y de las acciones realizadas por los jugadores, así como otros datos relacionados con el estado de la partida y la evolución de los stacks.

El procesamiento de toda esta información también formó parte del trabajo experimental. Los logs generados fueron recopilados y tratados mediante scripts de Python desarrollados específicamente para el estudio, encargados de extraer los datos, calcular las métricas seleccionadas y generar las gráficas utilizadas posteriormente. En paralelo, las partidas con jugadores humanos se complementaron

con cuestionarios realizados al finalizar las pruebas, incorporando así información que no puede obtenerse únicamente a partir de los registros de la partida.

El resultado es un proceso que abarca desde decidir qué situaciones merece la pena estudiar hasta convertir cada partida en información que pueda ser comparada e interpretada. Los siguientes apartados presentan las preguntas que se plantearon, la configuración utilizada en cada experimento y las métricas empleadas para analizar los resultados.

5.2 Objetivo del estudio

Una vez implementado los diferentes tipos de bots en la aplicación es importante ver cómo se comportan en situaciones reales de juego.

Con este propósito, se ha llevado a cabo un estudio con varios experimentos en diferentes escenarios de juego, donde los bot se enfrentan tanto a jugadores humanos como a otros bots. La diversidad de escenarios permite realizar un estudio de los resultados por cada tipo de bot, y sirven para observar la influencia que ejercen variables como el nivel de los jugadores humanos, el número de participantes en la mesa, el estilo de juego o el tipo de inteligencia artificial empleada.

De esta forma, el estudio nos da una visión completa de cómo se comportan los bots y nos permite sacar conclusiones sólidas acerca de su competitividad, capacidad de adaptación y estrategias.

5.2.1 Objetivos generales

Una vez recopilados los datos de todos los experimentos realizados se pretende responder a las siguientes cuestiones de investigación que conforman los objetivos generales del estudio empírico:

1. ¿Cómo juega cada tipo de bot?
2. ¿Qué bot obtiene mejores resultados?
3. ¿Qué influencia tiene la agresividad sobre el rendimiento final?
4. ¿Qué decisiones tienen mayor impacto en el resultado?
5. ¿Cómo evoluciona el comportamiento de cada bot durante la partida?
6. ¿Qué estrategias conducen con mayor frecuencia a la victoria?
7. ¿Influye el tipo de partida

Además de evaluar el rendimiento global, el objetivo de este estudio también es identificar patrones de comportamiento característicos de cada modelo.

5.2.2 Objetivos específicos por experimento

Aunque todos los experimentos persiguen el objetivo común de evaluar los bots implementados, cada uno analiza sus comportamientos en un contexto de juego distinto. Es debido a esto que se han planteado las siguientes cuestiones que se centran en un experimento concreto:

8. ¿Cómo cambia el bot frente a distintos niveles de jugador?
9. ¿Cómo se comportan los bots en una mesa con humanos?
10. ¿Los estilos realmente cambian el comportamiento del mismo modelo?
11. ¿Cómo compite el LLM contra NN y FSM?

5.3 Diseño y configuración experimental

Para el estudio empírico se ha diseñado un conjunto de experimentos orientados a analizar el comportamiento de los distintos modelos de IA en distintos escenarios utilizando la infraestructura creada. Antes de empezar con la experimentación fue necesario definir una serie de variables, con el fin de obtener resultados representativos y comparables.

5.3.1 Variables experimentales

Estas variables determinan la configuración de cada prueba, y son importantes ya que influyen en los resultados.

- Tipos de bots: se evaluaron el bot LLM, el bot Red Neuronal y el bot FSM.
- Tipos de humanos: los participantes humanos se clasificaron en varios niveles según su experiencia y conocimiento sobre el póker, estos son: principiante, intermedio y profesional.
- Tipos de entornos: Heads-up, mesas mixtas con humanos y bots, entorno donde únicamente participan bots y mesas donde se enfrentan un bot contra un bot.
- Tipos de estilos: además de elegir los tipos de bots con los que se jugó, los bots LLM Ollama y LLM Gemini presentan varios estilos de jugador (Loose-aggressive, Loose-Passive, Tight-Aggressive, Tight-Passive, Maniac y Default). Por ello, realizamos una

distribución variada entre todas las pruebas para que se jugasen los tipos de bots, junto con sus estilos, de manera equitativa.

- Tipos de seeds: para que las conclusiones del estudio empírico no dependan de una única distribución de cartas, ya sea de las comunitarias como las de la mano, se eligió una seed para cada experimento. A excepción de la primera experimentación donde intervinieron participantes humanos de diferentes niveles, la elección de la seed es para cada estilo de bot ya que, a lo hora de jugar varias partidas con diferentes modelos de inteligencia artificial, los jugadores podían darse cuenta de la repetición de las cartas y jugar con ventaja, interfiriendo negativamente en el estudio.
- Número de pruebas: para cada experimentó se predeterminó el número de partidas para reducir la influencia del azar y obtener resultados más representativos. En el caso del Experimento 1, el número de partidas que jugaron los participantes dependió del tipo de humano.

  La elección de esta configuración es debido a que se persigue evaluar el comportamiento de los bots desde perspectivas complementarias. Mientras que las partidas contra humanos permiten estudiar de qué forma se adaptan los bot a diferentes perfiles de jugador, las pruebas únicamente jugadas por bots permiten observar sus estrategias en entornos controlados y reproducibles.

5.3.2 Experimentación

En total, se realizaron 67 partidas experimentales divididas en cuatros escenarios de experimentación.

5.3.2.1 Experimento 1: Humano vs bot (1v1 – Heads-Up)

Este primer experimento permite estudiar el comportamiento de los distintos bots frente a jugadores con distinto nivel de experiencia. Para ello se definieron tres estilos: Nivel principiante, intermedio y profesional.

El jugador con el nivel más bajo (principiante) realizó 8 partidas, dos por cada tipo: Bot LLM Ollama estilo default, Bot LLM Ollama estilo loose-aggressive, Bot LLM Ollama estilo maniac, Bot red neuronal

El participante de nivel intermedio jugó 10 partidas en total, dos por cada tipo: Bot LLM Ollama estilo tight-passive, Bot LLM Ollama estilo default, Bot LLM Ollama estilo tight-aggressive, Bot LLM Ollama estilo loose-aggressive y Bot red neuronal

Por último, el jugador de mayor nivel jugó 9 partidas: LLM Bot Ollama estilo tight-passive (2 partidas), Bot LLM Ollama estilo default (2 partidas), Bot LLM Ollama estilo loose-aggressive (1 partida), Bot LLM Ollama estilo maniac (1 partida), Bot red neuronal (3 partidas)

En total, este primer experimentó está compuesto por 27 partidas y cuenta con las siguientes seeds para cada estilo del bot rival: 10 (bot LLM Default), 20 (bot LLM Tight-Passive), 30 (bot LLM Tight-Aggressive), 40 (bot LLM Loose-Passive), 50 (bot LLM Loose Aggressive), 60 (bot LLM Maniac).

Los jugadores al finalizar todas las pruebas tuvieron que rellenar un formulario donde proporcionaron información sobre su experiencia de juego, la percepción de los bots o el grado de realismo, entre otras cuestiones. Estas respuestas ayudan a complementar los resultados cuantitativos obtenidos y permiten extraer conclusiones desde el punto de vista del usuario. En este apartado de la memoria se proporciona un enlace a dichas respuestas. Respuesta formularios

5.3.2.2 Experimento 2: 1 humano vs 4 bots

Para este segundo experimento se crea un escenario en el que participan un jugador humano contra varios bots de diferentes tipos, todos con su estilo por defecto: Bot LLM Gemini, Bot LLM Ollama, Bot red neuronal y Bot FSM. En total se jugaron 17 partidas, 14 de ellas fueron jugadas por un humano A y el resto por un humano B. También se usó la misma seed para todas las partidas del experimento: 1777.

5.3.2.3 Experimento 3: 6 bots

En este experimento únicamente se enfrentan bots, de esta forma se puede estudiar su comportamiento en un entorno totalmente controlado. Se realizaron dos tipos de pruebas: en la primera prueba se jugaron dos partidas con la siguiente composición: un bot LLM Gemini estilo Tight-Aggressive, un bot LLM Gemini estilo Loose-Passive, un bot LLM Ollama estilo Tight-Aggressive, un bot LLM Ollama estilo Loose-Passive, un bot red neuronal y un bot FSM; en la segunda prueba se ejecutaron

3 partidas compuestas únicamente por bots LLM Ollama con los seis estilos de juegos existentes en la aplicación. De nuevo se eligió una única seed para todas las partidas: 2732.

5.3.2.4 Experimento 4: bot vs bot

Este último experimento se pensó para hacer comparaciones directas entre los tipos de bot implementados, esto es, sin la intervención de ningún humano. En este caso en particular, se eligieron tres estilos de bot LLM Ollama (*Default*, *Tight-Passive* y *Loose Aggressive*) para enfrentarse cada uno con el bot red neuronal y el bot FSM. En cada competición se realizaron tres pruebas, es decir, en total se realizaron 18 pruebas, y cuentan con una seed específica: 3387 (bot LLM Ollama Default vs bot red neuronal), 3333 (bot LLM Ollama Tight-Passive vs bot red neuronal), 3434 (bot LLM Ollama Loose Aggressive vs bot red neuronal), 3210 (bot LLM Ollama Default vs bot FSM), 3500 (bot LLM Ollama Tight-Passive vs bot FSM) y 3750 (bot LLM Ollama Loose Aggressive vs bot FSM).

5.4 Métricas de evaluación

Para analizar el comportamiento y el rendimiento de los bots se definieron un conjunto de métricas que permiten analizar sus decisiones, resultados y evolución durante las partidas. Estas métricas se agrupan según el aspecto que permiten estudiar.

5.4.1 Métricas de comportamiento

Estas métricas permiten caracterizar la forma en la que juega cada bot y comparar las diferencias entre las distintas tecnologías y estilos de juego:

- VPIP (Voluntary Put Money In Pot): porcentaje de manos en las que el jugador invierte fichas voluntariamente antes del flop. Se obtiene dividiendo las manos en las que el jugador participa voluntariamente entre el total de manos jugadas. Un rango orientativo habitual se encuentra alrededor del 20-30% valores inferiores indican un juego más selectivo y valores superiores un juego más arriesgado.
- PFR (Preflop Raise): porcentaje de manos en las que el jugador realiza un raise durante el preflop. Se obtiene dividiendo las manos en las que realiza un raise

preflop entre el total de manos. Un rango aproximado de 15-25% suele asociarse a un juego equilibrado

- 3Bet: porcentaje de veces sobre el total de manos jugadas en las que un jugador realiza una tercera subida en el preflop. Un valor aproximado de 5-10% suele ser lo habitual.
- WTSD (Went To Showdown): porcentaje de manos en las que un jugador llega al showdown después de haber visto el flop. Se obtiene dividiendo las manos que llegan al showdown entre las manos en las que el jugador vio el flop. Un rango aproximado de 22-30% suele considerarse habitual.
- WSD (Won Showdown): porcentaje de showdowns ganados por un jugador. Se calcula como el número de showdowns ganados entre el total de showdowns jugados. Un valor cercano al 50% constituye una referencia habitual.
- WWSF (Won When Saw Flop): porcentaje de manos ganadas entre aquellas en las que el jugador ha visto el flop. Se obtiene dividiendo las manos ganadas después de ver el flop entre el total de manos en las que lo vio. Un rango aproximado de 45-52% puede utilizarse como referencia.

5.4.2 Métricas de rendimiento

Estas métricas permiten evaluar la capacidad de los diferentes bots para obtener buenos resultados durante las partidas.

- Win rate: porcentaje de partidas ganadas respecto al total de partidas jugadas.
- Chips ganadas: diferencia entre el stack final y el inicial.
- Manos hasta ganar: número de manos jugadas hasta conseguir la primera victoria.
- Supervivencia: número de manos o tiempo que el jugador permanece en la partida antes de ser eliminado.

5.4.3 Métricas relacionadas con el equity

Estas métricas miden la fortaleza estimada de la situación del jugador en cada momento de la mano.

- Equity: probabilidad estimada de que un jugador gane una mano en una situación determinada, teniendo en cuenta sus cartas, las cartas comunitarias y

las posibles cartas restantes de los rivales. Su valor se encuentra entre 0% y 100%, donde un valor mayor indica una mayor probabilidad estimada de ganar la mano.

- Equity en la decisión: equity del jugador en el momento de realizar una acción.
- Equity en un All-in: equity del jugador en el momento de realizar un All-in.

5.4.4 Métricas de agresividad y All-ins

Estas métricas miden la frecuencia y el resultado de las decisiones de mayor riesgo.

- Número de All-ins: cantidad de All-ins realizados por el jugador.
- All-ins ganados/perdidos: proporción de All-ins que terminan en victoria o derrota.
- Calle del All-in: fase de la mano en la que se realiza el All-in: preflop, flop, turn o river.
- Eliminaciones: número de jugadores eliminados por cada bot.

5.4.5 Métricas de decisiones y evolución

Estas métricas miden qué decisiones toma el bot y cómo cambia su comportamiento o situación durante la partida.

- Acción realizada: decisión tomada por el jugador: fold, check, call, raise o All-in.
- Evolución del stack: variación del stack a lo largo de la partida.
- Tiempo como líder: tiempo durante el que el jugador mantiene el mayor stack de la mesa.

Los rangos indicados para las métricas de comportamiento se consideran únicamente valores orientativos, mientras que el resto de métricas se interpretarán principalmente mediante la comparación entre los diferentes bots.

5.5 Resultados de los experimentos

La ejecución de las partidas constituye el punto de partida del análisis experimental, pero no el resultado final. Para poder estudiar de forma objetiva el comportamiento de los bots, fue necesario recopilar, estructurar y procesar la información generada durante cada partida. Para ello se utilizó el sistema de monitorización desarrollado específicamente para este proyecto que permitió

registrar de forma sistemática las decisiones y resultados producidos durante la ejecución de los experimentos. A partir de estos registros se construyó el conjunto de datos empleado en el análisis y se calcularon las métricas definidas previamente. Esto permitió pasar de la observación individual de las partidas a una evaluación cuantitativa del comportamiento de los bots, haciendo posible identificar diferencias entre las distintas arquitecturas y estrategias, analizar sus patrones de juego y estudiar cómo evolucionan sus decisiones a lo largo de las partidas.

El análisis se ha estructurado en diferentes bloques, cada uno orientado a estudiar un aspecto concreto del comportamiento de los bots. Para cada uno de ellos se seleccionaron las métricas más representativas y se realizaron las comparaciones necesarias para extraer conclusiones a partir de los datos obtenidos. De esta manera, las figuras que se presentan a continuación no constituyen únicamente una representación de los resultados, sino que sintetizan el procesamiento de miles de decisiones y situaciones de juego recopiladas durante los experimentos y permiten visualizar los patrones que de otro modo serían difíciles de detectar directamente en las partidas.

Si se quisiera consultar la información utilizada para obtener estos resultados, se dispone de un **repositorio con los logs generados durante las partidas,** así como de los **ficheros de Python** utilizados para procesar los datos y generar las diferentes gráficas. De este modo, los resultados presentados pueden ser contrastados directamente a partir de los datos y del procedimiento empleado para obtenerlos.

5.5.1 ¿Cómo juega cada bot?

Una vez obtenidos los resultados de las partidas, no basta con conocer qué bot consigue un mejor resultado final. Para comparar las diferentes arquitecturas es necesario analizar cómo llegan a esos resultados, observando qué decisiones toman y cómo evoluciona su comportamiento a lo largo de las manos. Por este motivo, se utilizaron conjuntamente las métricas VPIP, PFR, 3Bet, WTSD, WSD y WWSF, que permiten seguir este comportamiento desde la participación inicial hasta las fases posteriores al flop.

La Figura 5-1.A muestra patrones claramente diferenciados. El BOT_NN presenta simultáneamente valores muy elevados de VPIP (82 %) y PFR (81 %), reflejando un juego en el que la mayoría de las manos en las que participa también comienzan con una subida. Además, su 3Bet alcanza aproximadamente el 5%, mostrando también una presencia significativa en situaciones de resubida preflop. El BOT_LLM, en cambio, combina un VPIP del 54% con un PFR de solo 4% y un 3Bet cercano al 1%, mostrando una participación frecuente, pero con mucha menor iniciativa preflop. En las fases posteriores, los tres bots mantienen un WTSD superior al 70 %, pero el BOT_NN alcanza un WSD del 49 % y un WWSF del 41 %, frente al 24 % y 19 % del BOT_FSM.

La Figura 5-1.B permite observar cómo este patrón cambia al modificar el estilo del LLM y donde observamos una de las mayores diferencias en el VPIP. También vemos que el estilo LOOSE_PASSIVE participa en aproximadamente el 30% de las manos, mientras que LOOSE_AGGRESSIVE supera el 65%. Esto supone un cambio de 3 de cada 10 manos a más de 6 de cada 10. Además, el WTSD varía entre aproximadamente el 59% y el 81%, mostrando que los estilos no solo modifican la frecuencia con la que se entra en una mano, sino también la tendencia a mantenerla hasta el showdown.

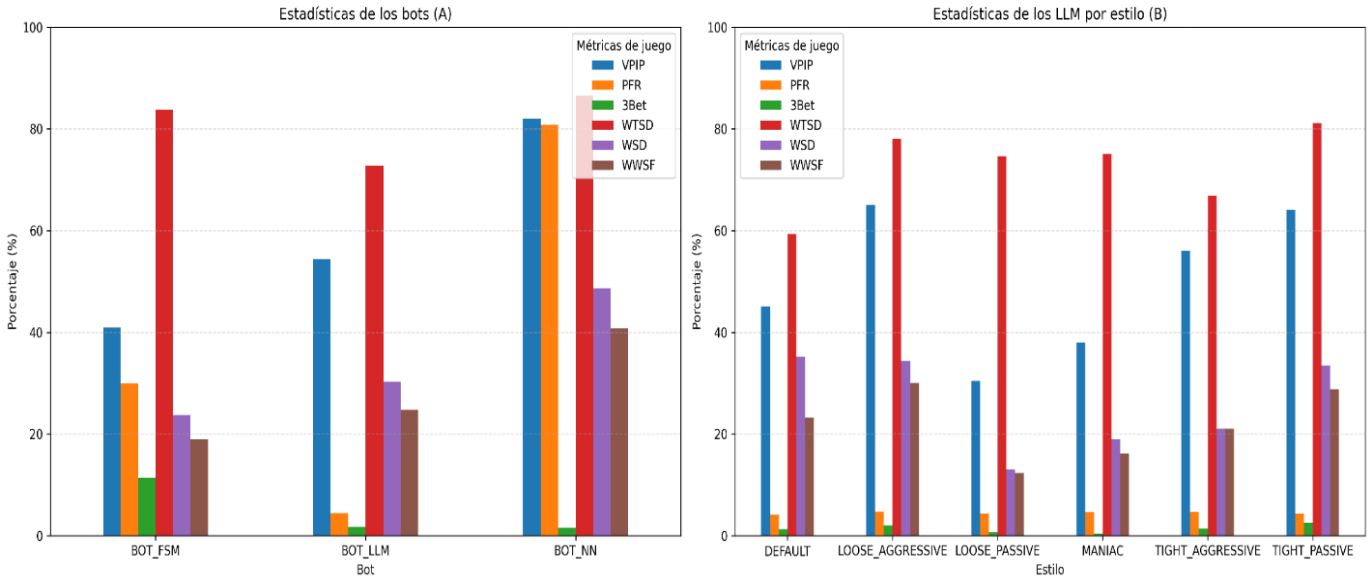


Figura 5-1. Distribución de métricas por tipo de bot y estilo de LLM

5.5.2 ¿Qué bot obtiene mejores resultados?

Una vez analizado cómo juega cada bot, la siguiente cuestión es qué consecuencias tiene ese comportamiento en las partidas. Por ello, no se considera únicamente quién gana más, sino también si esas victorias se traducen en fichas y cuánto consigue mantenerse en juego. Esta combinación permite pasar de analizar cómo juegan los bots a qué resultados produce su forma de jugar.

La Figura 5-2 muestra una clara diferencia entre arquitecturas. El BOT_NN obtiene el mayor *win rate* (48,7%), seguido del BOT_LLM (30,5%) y del BOT_FSM (23,9%). Esta diferencia también aparece en las fichas medias: −1,15, −2,05 y −3,20, respectivamente. Además, NN y LLM sobreviven unas 122 manos, frente a las 56 del FSM. En conjunto, los datos muestran que el BOT_NN no solo gana más partidas, sino que alcanza esas victorias manteniéndose más tiempo en juego y acumulando menores pérdidas medias.

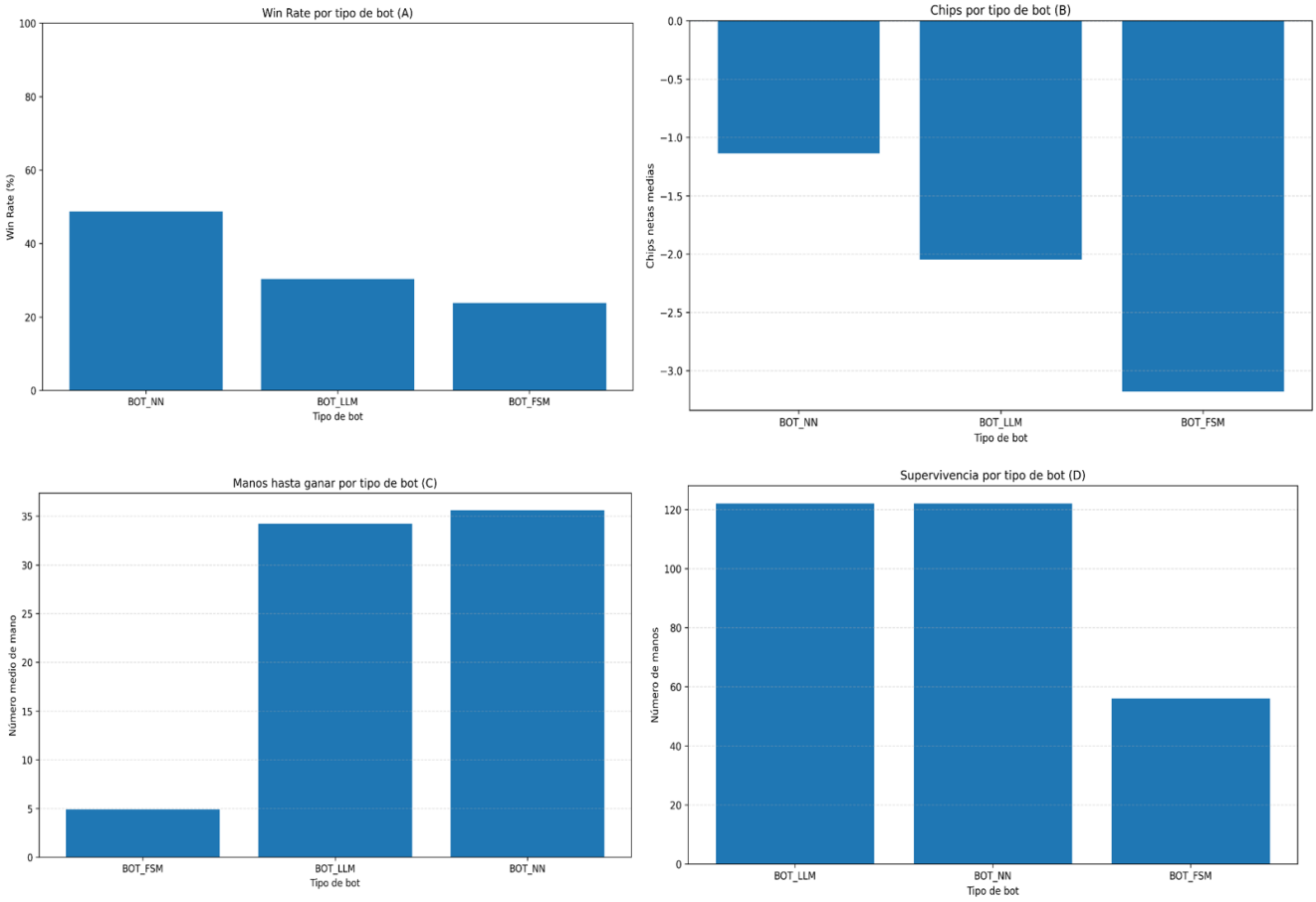


Figura 5-2. Distribución por tipo de bot

Al centrarnos en los estilos del LLM, aparece un comportamiento más interesante, como podemos ver en la Figura 5-3. LOOSE_PASSIVE obtiene solo un 13,0% de victorias, mientras que DEFAULT, LOOSE_AGGRESSIVE y TIGHT_PASSIVE se sitúan entre el 33,5% y el 35,2%. Sin embargo, TIGHT_PASSIVE destaca por ser el único con fichas medias positivas (+0,08) y alcanza además la mayor supervivencia (122 manos). Esto muestra que un porcentaje de victorias similar no implica necesariamente el mismo resultado en fichas ni la misma evolución durante la partida.

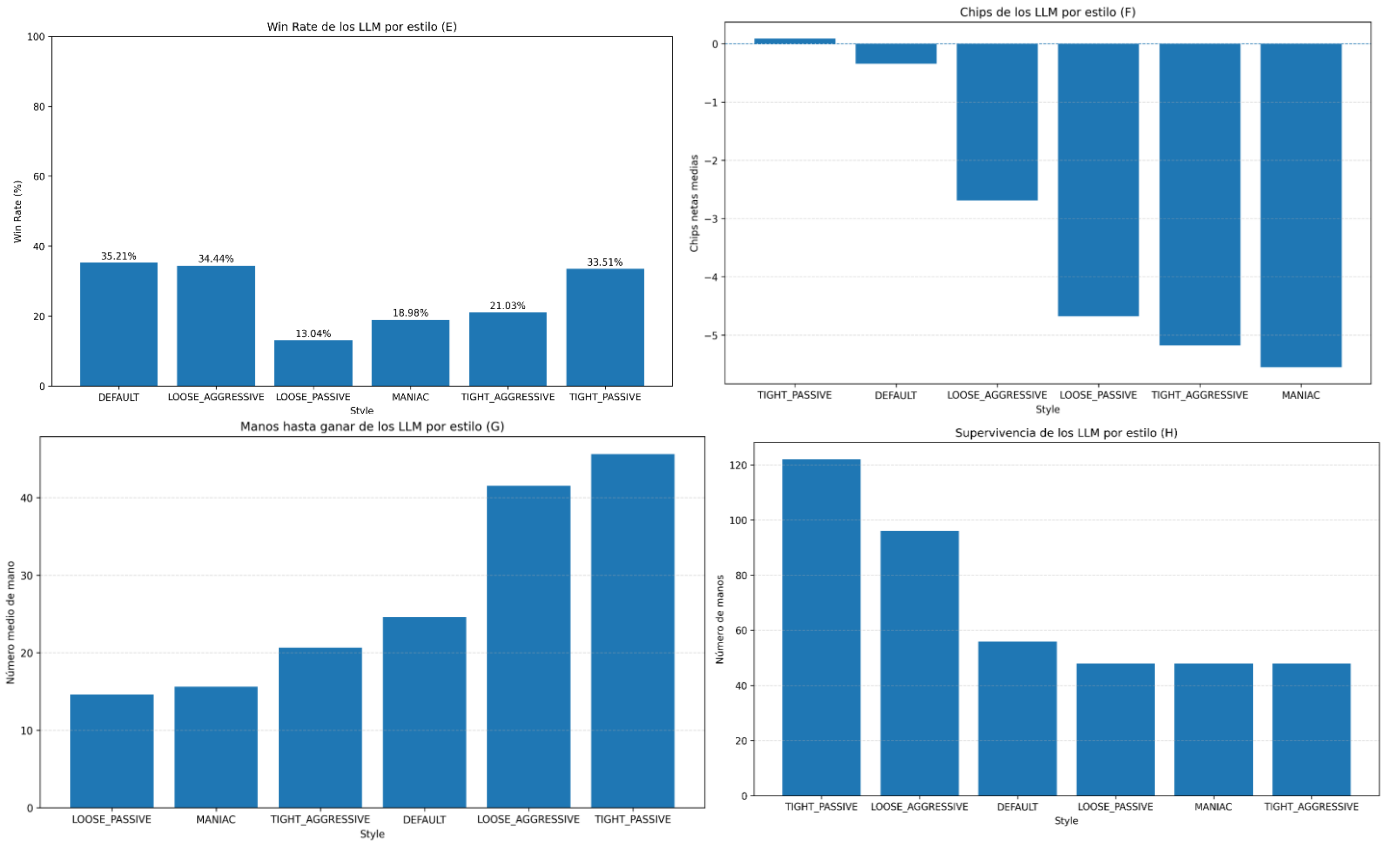


Figura 5-3. Distribución por estilo de LLM

5.5.3 ¿La agresividad mejora los resultados?

Las diferencias observadas entre los bots plantean una cuestión adicional: ¿asumir más situaciones de riesgo se traduce en mejores resultados? Para estudiarlo, se analizaron los *all-in*, relacionando su frecuencia, el momento en que se producen y el equity disponible con su resultado final. De esta forma, se puede observar no solo cuánto arriesga cada bot, sino qué ocurre cuando lo hace.

La Figura 5-4 muestra que el número de *all-in* no se relaciona directamente con un mejor resultado. El BOT_LLM realiza 91, muy por encima de los 57 del FSM y los 34 del NN, pero pierde 61, frente a 30 ganados. Además, su equity medio en el momento del *all-in* es del 36,39 %, inferior al 42,62 % del FSM y al 45,74 % del NN. El momento en que se asume el riesgo también cambia: el LLM concentra los *all-in* especialmente en el turn, mientras que el FSM los realiza principalmente en el flop.

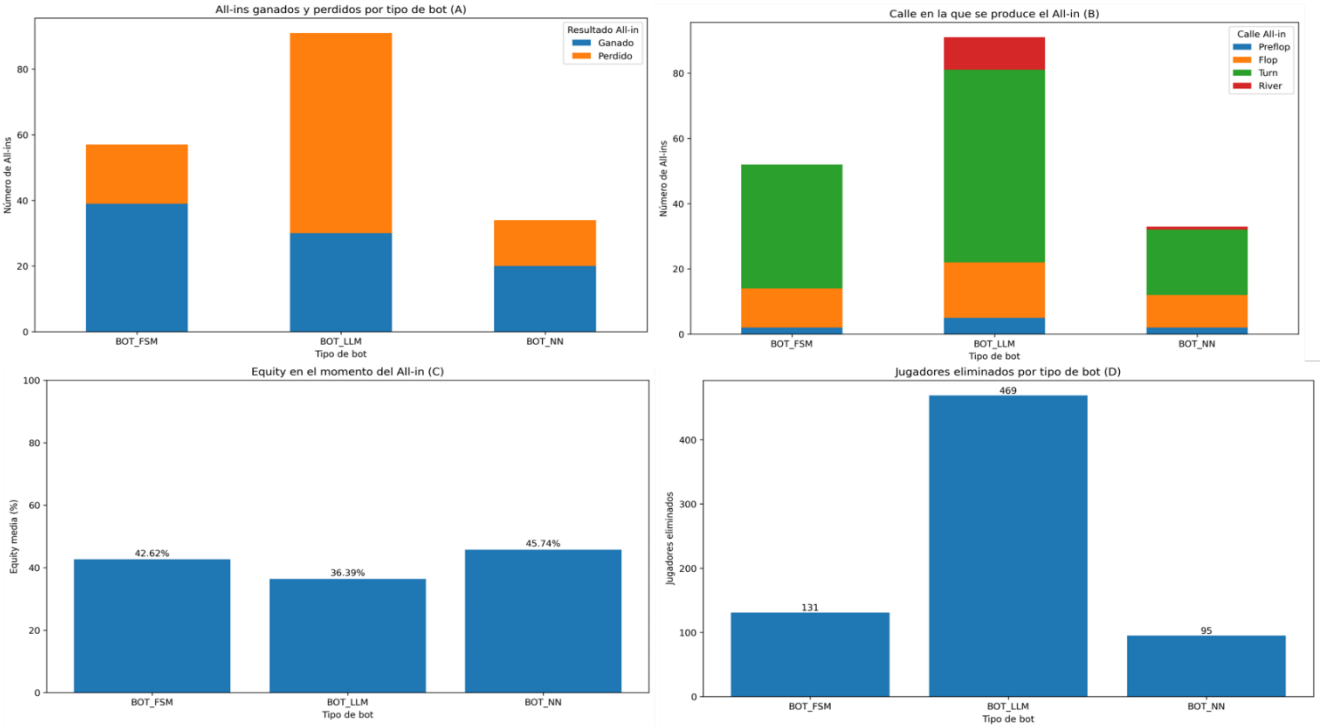


Figura 5-4. Distribución por tipo de bot

En los estilos del LLM, tampoco aparece una relación directa entre más *all-in* y mejores resultados. En la Figura 5-5 podemos observar como LOOSE_AGGRESSIVE y TIGHT_PASSIVE realizan 22 *all-in* cada uno, pero sus equity medios son muy diferentes: 37,62% y 52,31%, respectivamente. En particular, TIGHT_PASSIVE gana 11 de sus 22 *all-in*, mientras que LOOSE_AGGRESSIVE solo gana 7. Además, el número de jugadores eliminados presenta una diferencia muy marcada entre arquitecturas: 469 con LLM,

frente a 131 con FSM y 95 con NN, mostrando que la frecuencia de estas situaciones también condiciona el impacto que tienen durante las partidas.

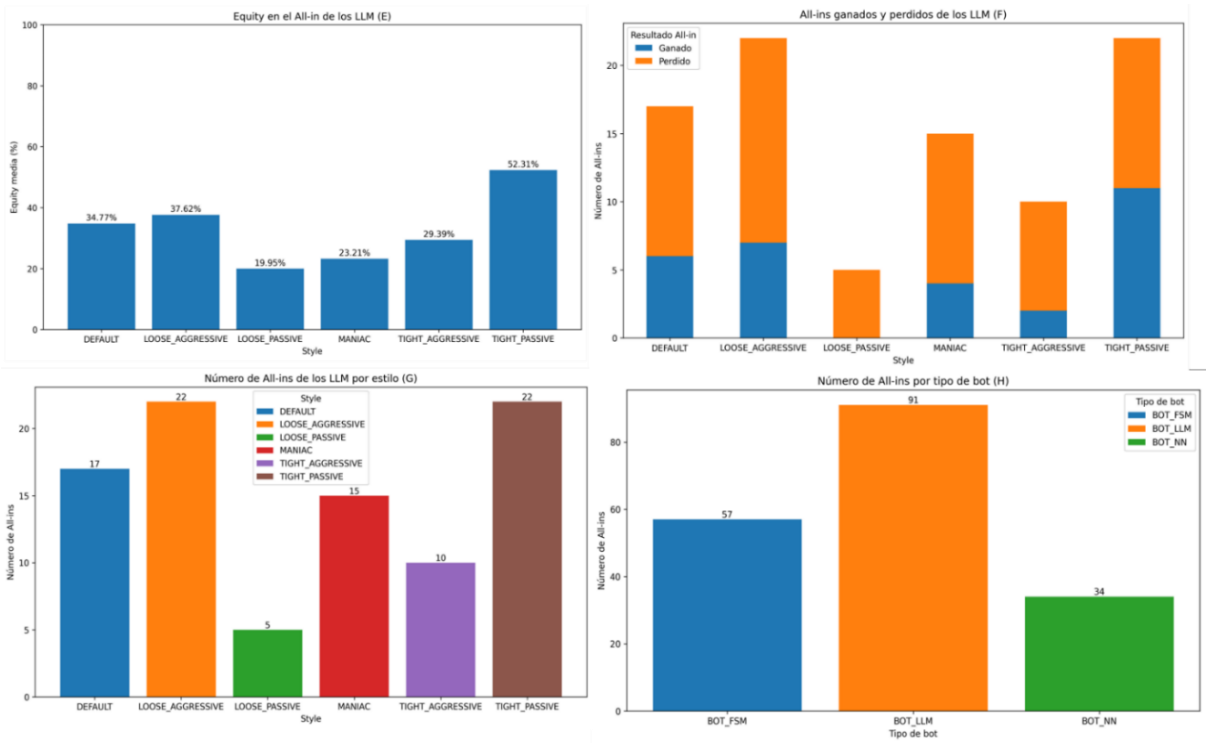


Figura 5-5. Distribución por estilo de LLM y tipo de bot

5.5.4 ¿Qué decisiones tienen mayor impacto en el resultado?

Para determinar qué decisiones tienen mayor impacto en el resultado, se analizó cómo se distribuyen las acciones tomadas por los bots entre fold, call, raise y All-in. Comparar esta distribución entre arquitecturas y estilos permite relacionar los resultados obtenidos con los patrones de decisión que los generan.

Podemos ver que la Figura 5-6.A y la Figura 5-6.B muestran las decisiones recopiladas para cada estilo del LLM, respectivamente. Entre las arquitecturas aparecen patrones claramente diferenciados. El BOT_LLM concentra sus decisiones principalmente en call y fold, mientras que el BOT_NN lo hace en raise, con unas 850 decisiones de este tipo. El BOT_FSM presenta un reparto más equilibrado, aunque destaca la frecuencia de All-in. Entre los estilos del LLM también cambia la acción predominante: LOOSE_AGGRESSIVE, TIGHT_PASSIVE y TIGHT_AGGRESSIVE concentran

sus decisiones en call, mientras que LOOSE_PASSIVE presenta un reparto más equilibrado entre call y fold. MANIAC destaca por una mayor presencia de All-in.

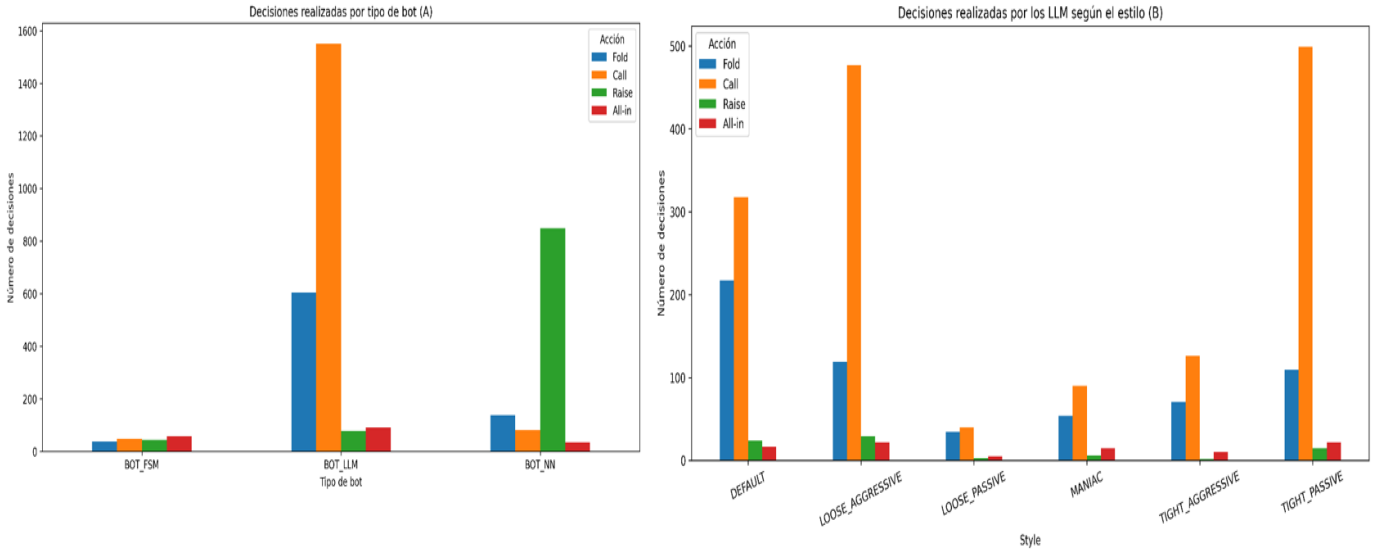


Figura 5-6. Distribución de decisiones por tipo de bot y estilo de LLM

Podemos apreciar que la Figura 5-7.C y la Figura 5-7.D permiten ver estas diferencias mediante gráficos de distribución. En las arquitecturas, el BOT_NN concentra el 77 % de sus decisiones en raise, aproximadamente 3 de cada 4. El BOT_LLM, en cambio, concentra el 66,7% en call y el 26% en fold, mientras que raise y All-in representan solo el 3,4% y 3,9%. El BOT_FSM mantiene una distribución más equilibrada, aunque el *all-in* alcanza el 31%.

En los estilos del LLM, LOOSE_AGGRESSIVE concentra el 73,7 % en call, frente al 48,2% de LOOSE_PASSIVE, que eleva el fold hasta el 42,2%. TIGHT_PASSIVE alcanza el mayor porcentaje de call, con un 77,4%, mientras que MANIAC presenta la mayor proporción de All-in, con un 9,1%. Estas diferencias permiten comparar los patrones de decisión con los resultados obtenidos en las métricas anteriores y estudiar qué acciones aparecen asociadas a cada nivel de rendimiento.

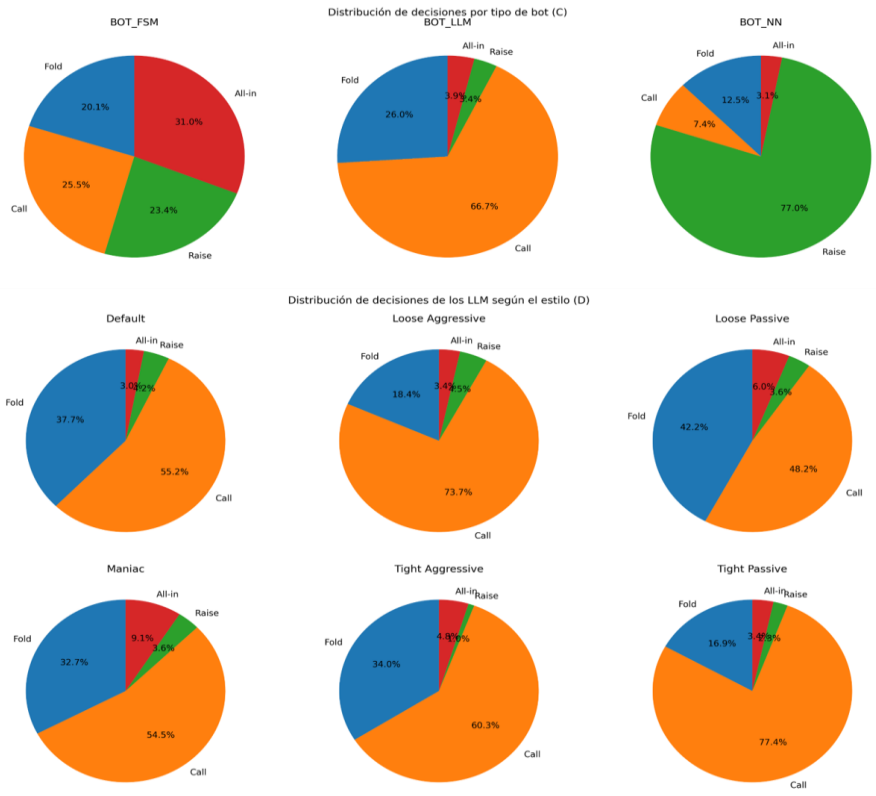


Figura 5-7. Distribución de decisiones por tipo de bot y estilo de LLM

5.5.5 ¿Cómo evoluciona el bot durante la partida?

En este capítulo nos interesa medir la evolución del bot durante una partida mediante su stack de fichas, su VPIP, su *equity* y la cantidad de manos en las que es el líder de la partida, es decir, en cuántas manos es el jugador con más fichas.

Podemos observar en la Figura 5-8.A que el bot FSM pierde con relativa facilidad frente al resto de bots, mientras que el bot NN y el bot LLM no son realmente agresivos debido a la poca variabilidad de su stack, indicando apuestas bajas propias de un perfil más conservador. Por otro lado, la Figura 5-8.B nos muestra que los bots NN y LLM se involucran en una gran cantidad de manos debido a unos valores de VPIP por encima del 70%. Sin embargo, podemos ver en la Figura 5-8.C que estos dos bots se mantienen como líderes una gran cantidad de manos a pesar de tener un *equity* en torno al 50%, como muestra la Figura 5-8.D, indicando más bien que intercambian el liderazgo continuamente de forma aleatoria y no por una dominancia real.

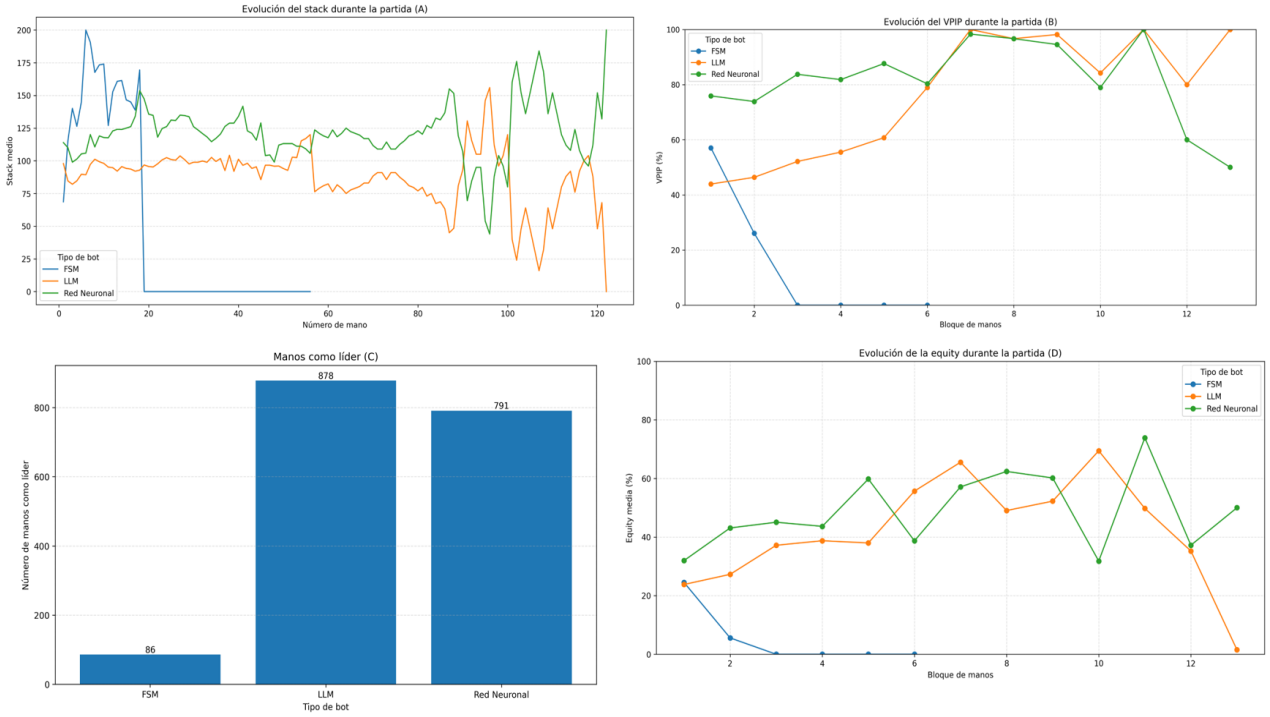


Figura 5-8. Evolución del stack, VPIP, manos como líder, y evolución de la equity por tipo de bot

5.5.6 ¿Cómo consigue ganar cada bot?

Para entender mejor el comportamiento de los bots hemos estudio la forma en la que estos ganan sus manos, la cantidad de manos necesarias para ganar a sus rivales y cuantas veces han eliminado a un jugador.

La Figura 5-9.A nos muestra que el bot FSM ha obtenido victorias muy rápidas y espontáneas debido, en su mayoría, a all-ins, como muestra el 72% de all-ins por victoria de la Figura 5-9.C.

Por otro lado, los bots NN y LLM, tienen un comportamiento muy similar y con una poca cantidad de manos para obtener una victoria, como se aprecia en la Figura 5-9.A, Figura 5-9.C y Figura 5-9.D. Sin embargo, en la Figura 5-9.B se puede observar que el bot LLM tiene la mayor cantidad de eliminaciones por una diferencia bastante considerable respecto al resto de bots.

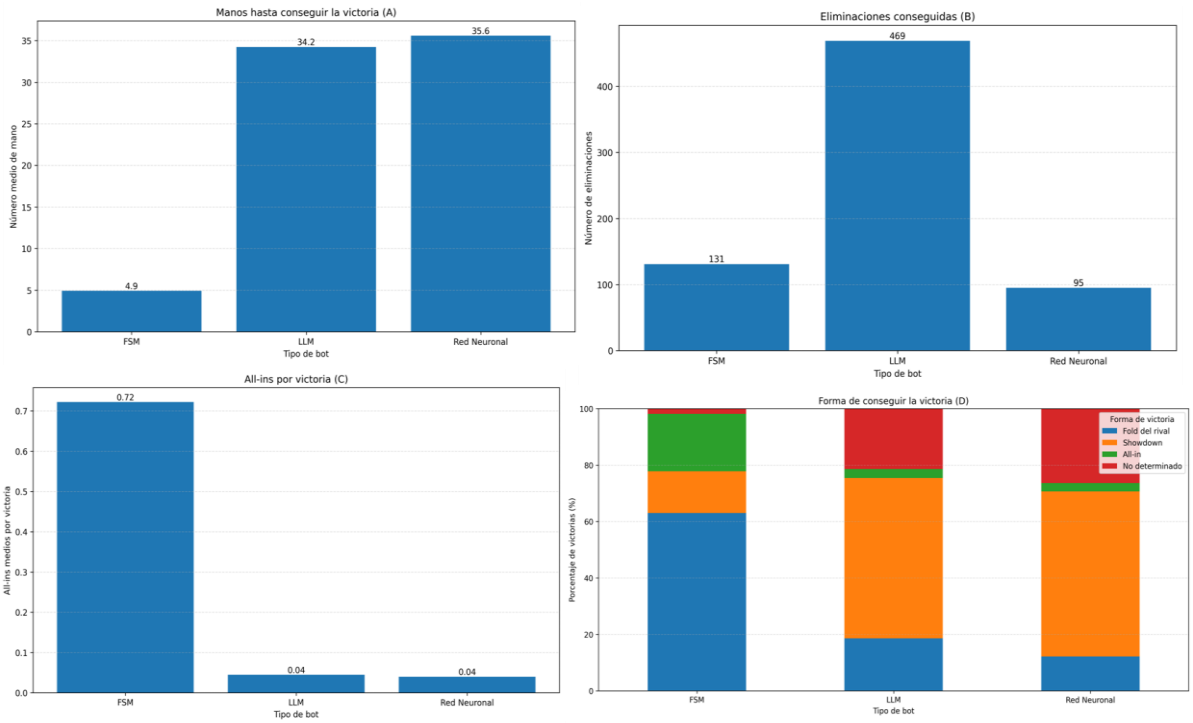


Figura 5-9. Manos hasta conseguir la victoria, eliminaciones, all-ins y formas de conseguir la victoria

5.5.7 ¿Influye el tipo de partida?

Para averiguar si el tipo de partida afecta al comportamiento de cada bot hemos medido sus *win rates*, *VPIP*, cantidad de *all-ins*, *equity* y la media de manos necesarias hasta ganar, obteniendo un mejor entendimiento del desempeño de cada bot en las diferentes pruebas.

Podemos observar que el *win rate*, representado en la Figura 5-10.A, tiene mayores fluctuaciones y destacando que la red neuronal obtiene un mejor desempeño en las partidas *heads-up*, lo que puede ser producido por su entrenamiento específico en este campo.

El *VPIP* representado en la Figura 5-10.B se mantiene un poco más estable, sobre todo en las partidas de bots, aunque con una mayor dispersión a favor de la red neuronal, quizá por el mismo motivo mencionado anteriormente.

El *equity* y el número de *all-ins* se mantiene más estable en la Figura 5-10.C y Figura 5-10.E, aunque en el caso de la máquina de estados se puede apreciar un comportamiento más agresivo mediante *all-ins* con un *equity* medio débil, obteniendo victorias muy esporádicas a costa de multitud de pérdidas.

Por último, la Figura 5-10.D nos muestra que los bots tienen a ganar mediante partidas cortas, ya que necesitan en torno a 40 manos o menos.

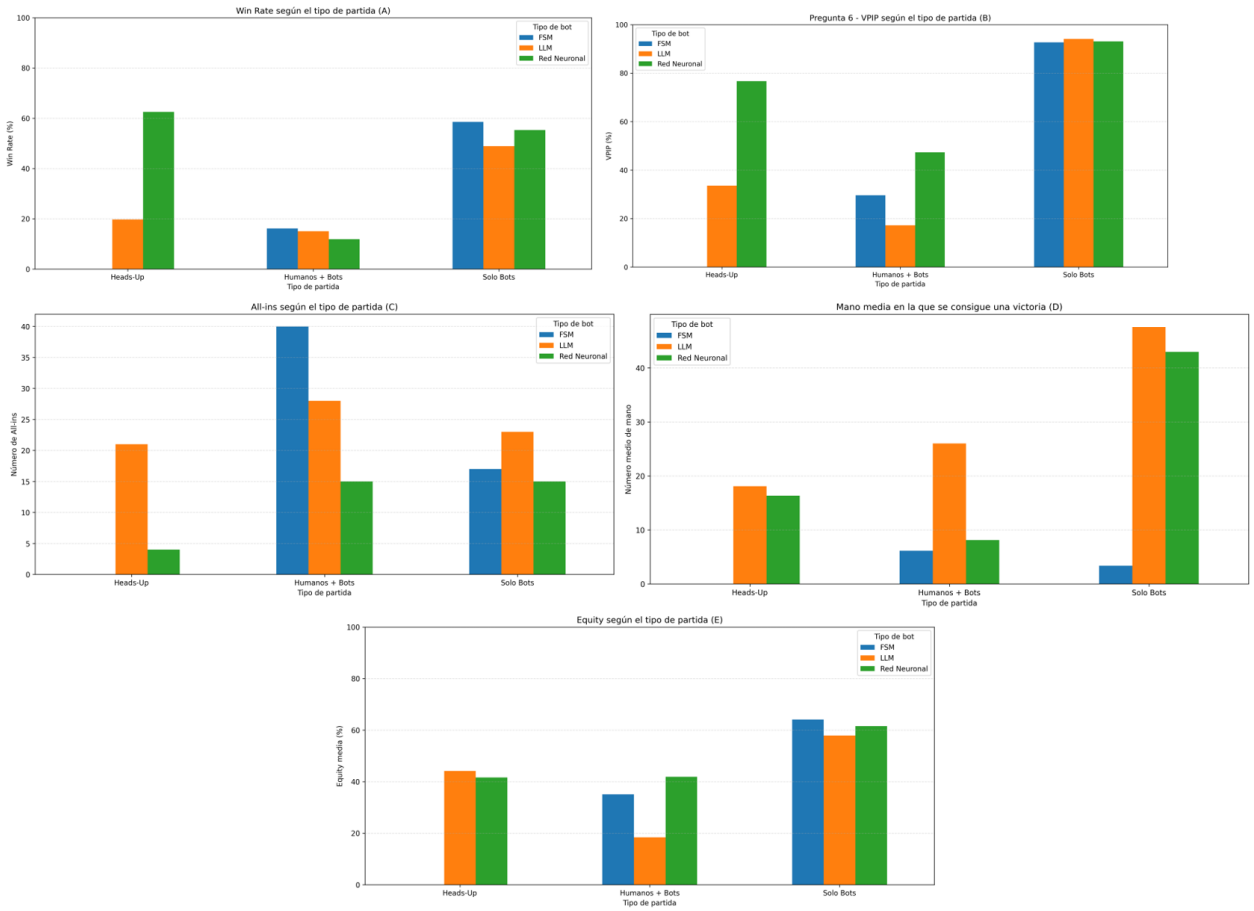


Figura 5-10. Win rate, VPIP, All-ins, Mano media en la que se consigue una victoria, Equity según el tipo de partida

5.5.8 ¿Cómo cambia el bot frente a distintos niveles de jugador?

En este capítulo medimos el desempeño del bot LLM con múltiples estilos de juego y el bot mediante red neuronal al enfrentarse a varios humanos en partidas *heads-up* con nivel de habilidad: principiante, intermedio y profesional. Las métricas empleadas para ello han sido el *win rate*, las fichas obtenidas de media en cada mano, su *equity*, el *VPIP* y el *PFR* de este en cada partida.

La Figura 5-11.*A* muestra que la *red neuronal* mantiene una tasa de victorias constantemente en torno al 60% ante cualquier rival, mientras que el LLM sufren un descenso frente a rivales intermedios antes de repuntar ante profesionales. En la Figura 5-11.*B* se pueden apreciar balances negativos para la mayoría en el nivel intermedio, donde aparece un dato puntual para el bot *tight-aggressive*, y un

rendimiento positivo únicamente para el perfil *default* en el nivel profesional, donde se observa el punto del bot *maniac*. Por otro lado, la Figura 5-11.*C* compara la estrategia del LLM y de la red neuronal, observándose que la primera reduce notablemente su participación y agresividad conforme sube el nivel del rival, mientras la red neuronal mantiene porcentajes elevados de forma regular. En la Figura 5-11.*D* que la cantidad de *all-ins* tiende a reducirse al jugar con humanos más experimentados. Por último, la Figura 5-11.*E* ilustra cómo el valor esperado medio de las manos de casi todos los bots se incrementa al jugar contra rivales profesionales, manteniéndose en torno al 50%.

Todas estas métricas resultan interesantes para decidir si ciertos tipos de bots con o sin configuraciones adicionales de estilo de juego son capaces de replicar el estilo de juego deseado sobre humanos con diferentes niveles de habilidad. Por otro lado, también nos permite ver si el nivel de experiencia de juego de un humano repercute a su vez en el rendimiento del bot en partidas uno contra uno.

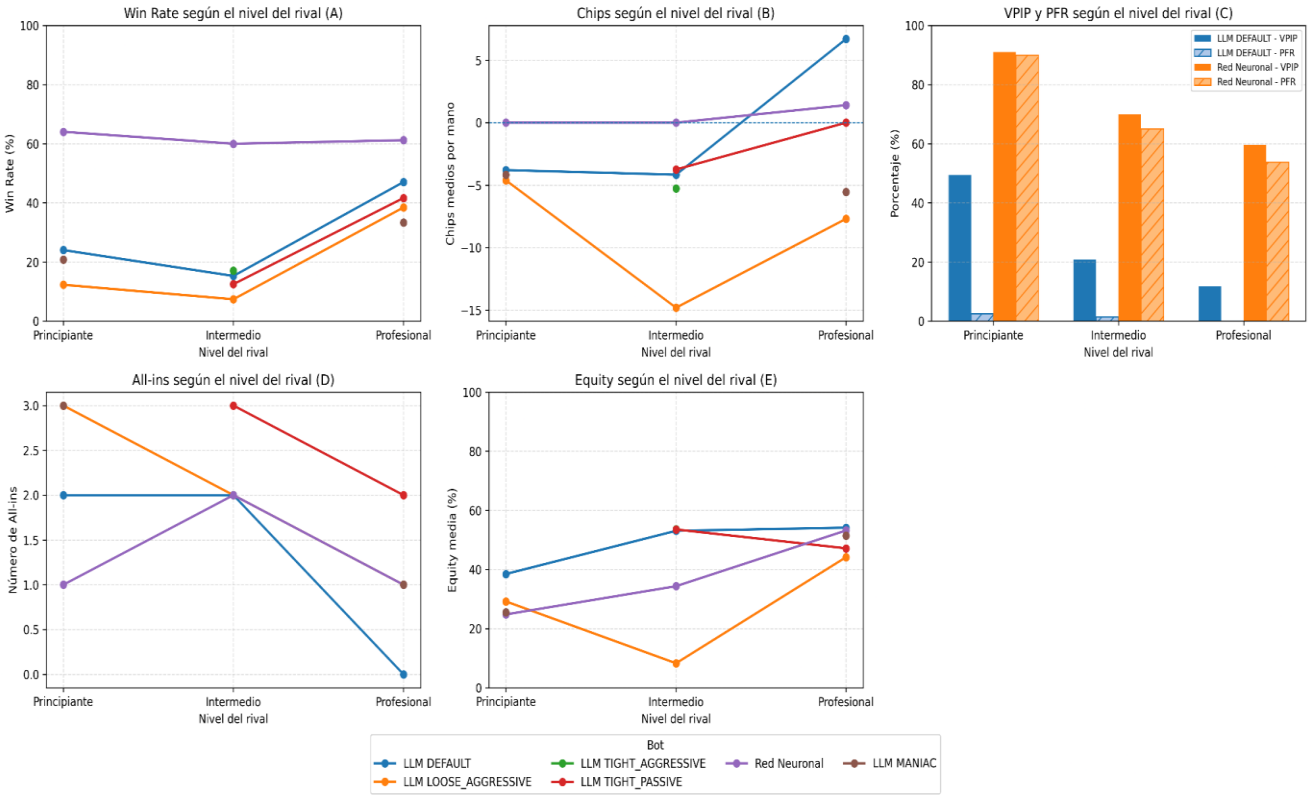


Figura 5-11. Win rate, Chips, VPIP y PFR, All-ins, Equity según el nivel del rival

5.5.9 ¿Cómo se comportan los bots en una mesa con humanos?

En este experimento podemos observar el enfrentamiento entre dos jugadores humanos principiantes contra los bots LLM, la red neuronal y la máquina de estados. La evolución del stack, el *win rate*, VPIP y las eliminaciones nos permiten comparar cómo se comportan los diferentes jugadores dentro de una misma mesa y cómo evoluciona su posición durante la partida.

La Figura 5-12 muestra cuatro gráficas de resultados, comparando el humano A con los tres modelos de bots mencionados. Se puede apreciar que el humano A adopta el estilo de juego más activo de la mesa, reflejado tanto en su *win rate* (ver Figura 5-12.B) como en su elevado VPIP (ver Figura 5-12.C). Esta agresividad controlada dictada por dichas métricas, le permiten acumular fichas durante gran parte de la partida y alcanzar el mayor stack registrado, aunque siempre termine eliminado a mirada de las manos. En cuanto al bot LLM, su VPIP se encuentra por debajo del rango habitual y presenta un *win rate* moderado, sin embargo, es el único que consigue sobrevivir hasta el final de la partida. La Red Neuronal muestra una gran participación en las manos, aunque no se traduce en ganar partidas. El bot FSM destaca por ser el jugador con más *all-ins*, un total de 28, reflejando un comportamiento especialmente agresivo que tampoco se transforma en un mejor rendimiento. En conjunto, los resultados sugieren que en este escenario la frecuencia de participación y la agresividad no garantizan el éxito final.

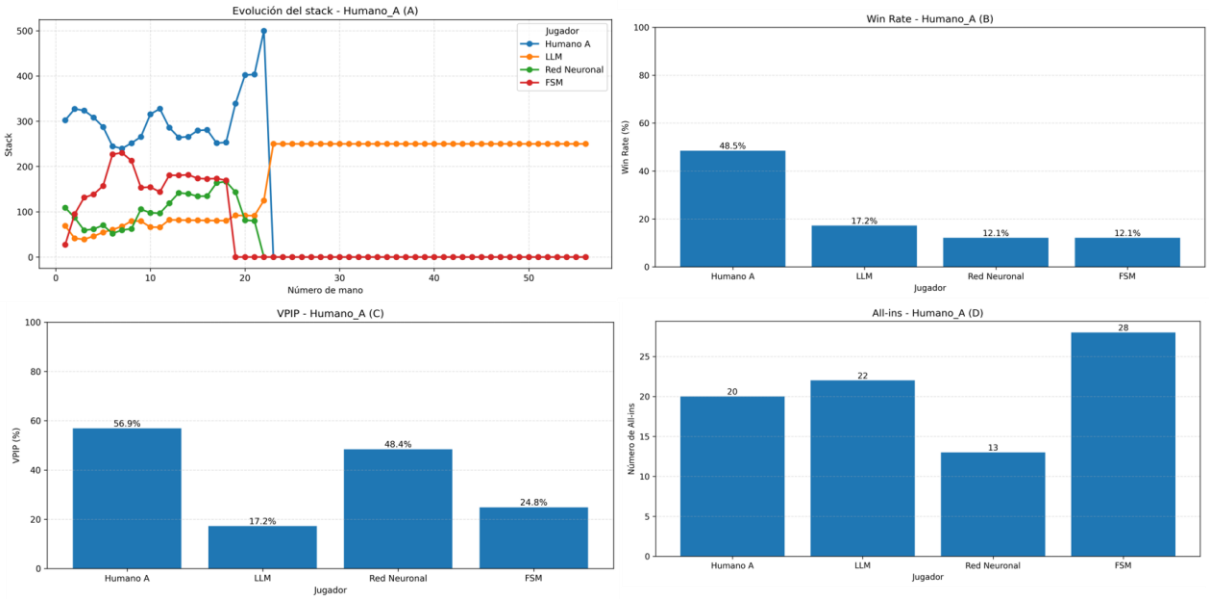


Figura 5-12. Evolución, win Rate, VPIP, All-ins del Humano A

En la Figura 5-13 se comparan el humano B con los bots. El conjunto de los resultados de las cuatro gráficas, muestran un claro dominio del jugador humano que presenta el mayor VPIP (75,9%) y *win rate* (58,6%), además de mantener una evolución del stack ascendente que se traduce en una participación muy activa en la manos, a diferencia de los bots. El bot FSM destaca de nuevo por ser el jugador que realiza más *all-ins*, que a diferencia con el primer humano (ver Figura 5-12.A y Figura 5-13.A) lidera la partida durante las primeras manos, aunque posteriormente pierde gran parte de su stack y termina eliminado, reflejando su fisionomía aleatoria. Tanto el bot Red Neuronal como el bot LLM muestran un rendimiento inferior ya que quedan eliminados al principio de la partida, como se puede observar en la Figura 5-13.A

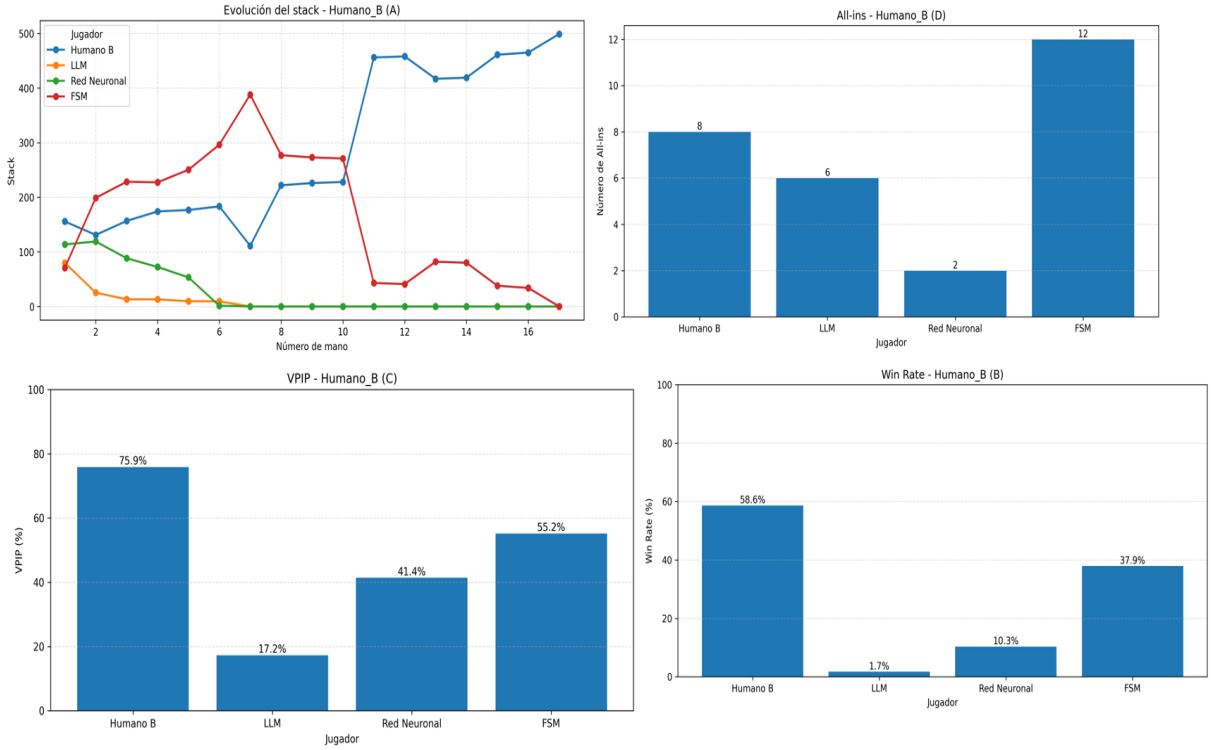


Figura 5-13. Evolución, Win rate, VPIP, All-ins del Humano B

5.5.10 ¿Los estilos cambian el comportamiento del mismo modelo?

Para comprobar el efecto de los estilos de juego se compararon seis instancias de los bots LLMs configuradas con DEFAULT, TP, TAG, LP, LAG Y MANIAC. Las métricas como VPIP, PFR y 3Bet permiten comprobar si los estilos generan diferencias en el comportamiento, mientras que *win rate*, chips y All-ins muestran si estas diferencias también se reflejan en los resultados.

Los resultados de la Figura 5-14 muestran las diferencias significativas entre bots del mismo modelo. En particular podemos observar la métrica VPIP de la Figura 5-14.A, que refleja la existencia de estilos muy pasivos como *Loose Aggressive*, hasta otros extremadamente activos como el *Maniac*, llegando casi al 100%. Estas diferencias también se trasladan al rendimiento, observándose variaciones en el *win rate*, las *Chips* y el número de All-ins realizados. En la Figura 5-14.C destaca el estilo Maniac por ser el que más fichas ha ganado en las partidas en comparación con el resto de los estilos. Sin embargo, el *Loose Aggressive* alcanza el mayor porcentaje de victorias.

Un resultado interesante de mencionar es el porcentaje tan bajo de VPIP esperado para el estilo *Loose Aggressive*, demostrando que el estilo configurado no siempre refleja el comportamiento esperado y que puede influir el contexto a la hora que el bot toma una decisión.

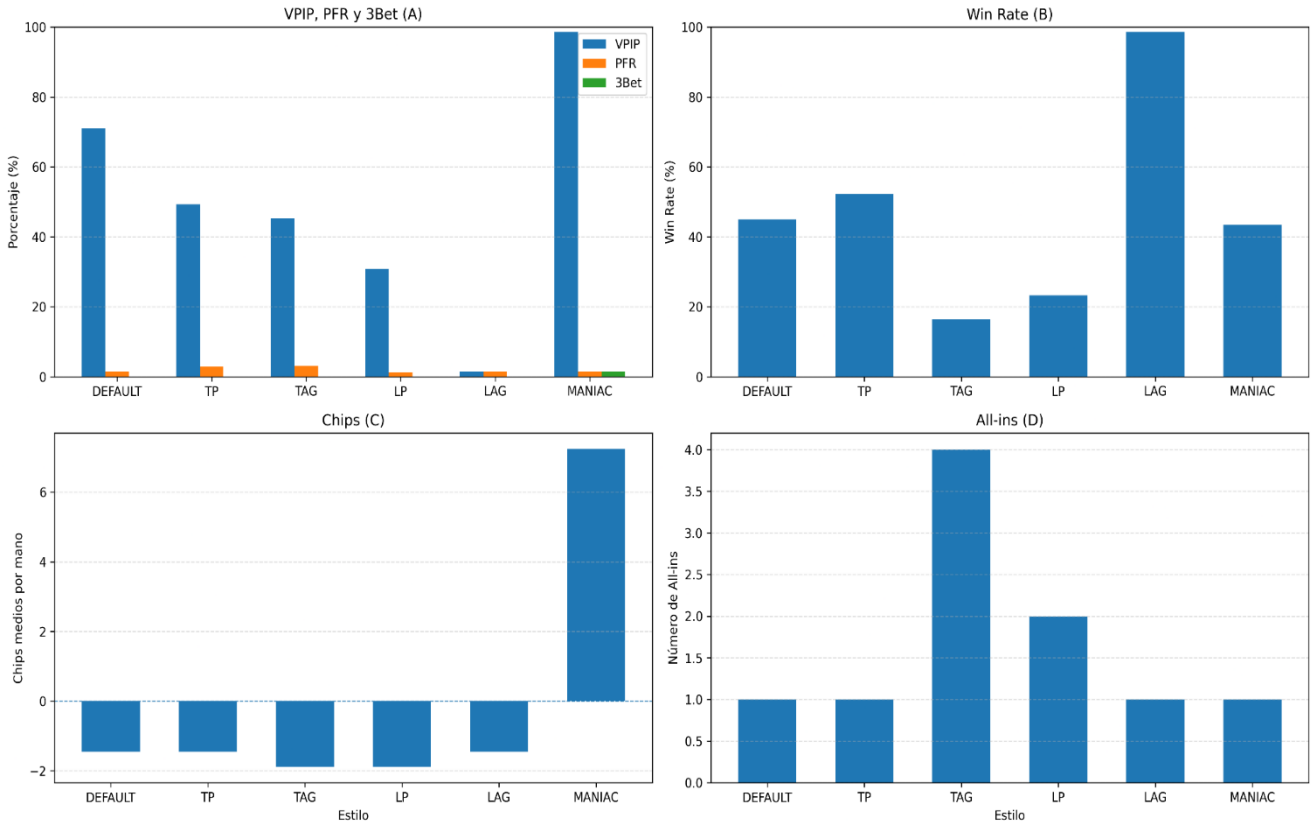


Figura 5-14. VPIP, PFR Y 3Bet, Win rate, Chips y All-ins diferencias entre estilos del mismo modelo

5.5.11 ¿Cómo compite el LLM contra NN y FSM?

Finalmente se enfrentaron los distintos bots LLM con sus diferentes estilos contra la red neuronal y la máquina de estados para comprobar qué diferencias existen entre ellos tanto en sus resultados como en su forma de jugar. Para ello se analizaron el *win rate*, las chips, el equity y los All-ins.

Las gráficas de la Figura 5-15 muestra las diferencias en comportamiento entre los tres tipos de bots. En términos de *win rate* (ver Figura 5-15. Win rate, Chips medios, Equity media y All-ins por tipo de botA), el bot FSM obtiene mejor resultado (58,5%), seguido por la Red Neuronal (55,3%) y el LLM (48,9%). Sin embargo, al analizar las chips medias, tanto el LLM como la Red Neuronal tienen valores positivos, al contrario que el FSM que acumula pérdidas medias significativas, indicando que no hay una relación directa entre ganar partidas y mayor rentabilidad.

Respecto a la media del equity por tipo de bot de la Figura 5-15.C, el FSM se involucra más en situaciones estadísticamente más favorables (64,12%), seguido de la Red Neuronal (61,52%) y el LLM (57,90%). En cuanto a la agresividad, el LLM es el bot que realiza un mayor número de All-in, compensando su menor equity de media y permitiéndole tener un balance de fichas superior al del FSM.

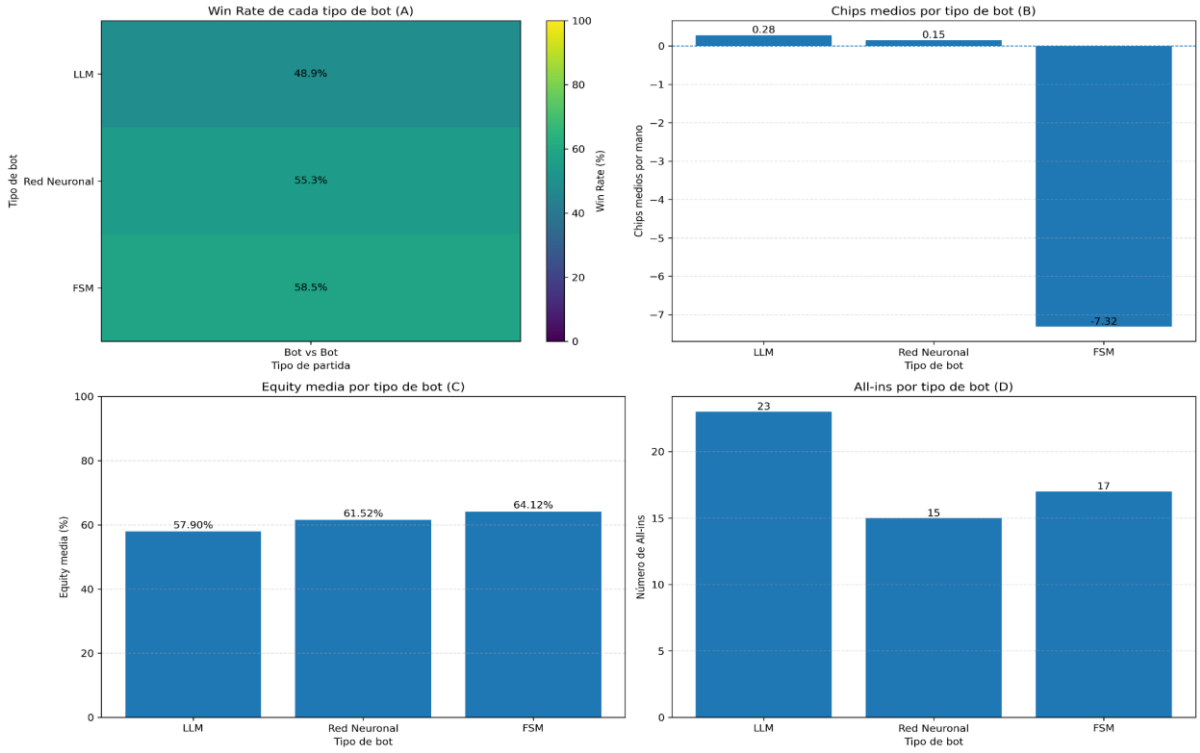


Figura 5-15. Win rate, Chips medios, Equity media y All-ins por tipo de bot

5.6 Análisis y discusión de resultados

A primera vista, los resultados obtenidos podrían parecer suficientes para determinar qué bot es mejor: bastaría con comparar quién gana más partidas, quién acumula más fichas o quién sobrevive durante más tiempo. Sin embargo, cuando se analizan todas las métricas y las decisiones tomadas durante la partida, esta clasificación deja de ser tan evidente.

Esta dificultad tiene su origen en la propia naturaleza de un juego imperfecto como el póker. No se trata de un juego en el que exista una secuencia fija de decisiones que conduzca siempre al mejor resultado, sino de un entorno en el que cada acción modifica el contexto. Las cartas , las apuestas, los rivales y la evolución de la partida exigen al jugador adaptar su estrategia. Los experimentos muestran que nuestros bots, aunque son capaces de jugar y de establecer patrones y comportamientos distintos, tienden a mantenerse dentro de una estrategia específica en lugar de adaptarse a lo que está ocurriendo en la mesa, por lo que resultan siendo poco competitivos.

En términos generales, hemos podido observar que las actuales implementaciones de los bots mediante LLMs y NNs tienen a mantener un perfil bastante conservador, realizando apuestas bajas y asumiendo pequeños riesgos en momentos muy puntuales. Además de eso, los estilos de juego en los bots LLM no parecen influir significativamente en el comportamiento de estos, debido a que estos son más bien un intento de reconducir la toma de decisiones del bot por un camino preestablecido más que una aproximación determinista.

Por otro lado, el bot mediante máquina de estados es el menos consistente de todos, debido a su carácter aleatorio y limitado.

El comportamiento de la red neuronal está caracterizado por una elevada participación y una tendencia a tomar la iniciativa. Además, aparece acompañado del mayor *win rate* entre las arquitecturas, aunque esté lejos de ser un jugador sólido.

Finalmente, los resultados cuantitativos obtenidos durante las pruebas se complementaron con la información cualitativa mediante formularios que realizaron los participantes del *experimento 1* y *2*. El objetivo de esta evaluación fue

complementar los resultados cuantitativos con la visión de los jugadores sobre el comportamiento, el realismo y la dificultad de los distintos bots.

Los participantes de nivel principiante coincidieron en que todos los bots eran capaces de seguir las reglas del juego y completar las partidas sin errores significativos. Sin embargo, la percepción de su nivel de juego varió en función de la experiencia previa de cada jugador humano. Destacaron dificultades a la hora de identificar diferencias claras entre algunos estilos de los bots LLM, por ejemplo, con el Loose Aggressive, percibiendo en muchos casos comportamientos demasiados conservadores y un número elevado de veces la acción *fold* al principio de la partida. A pesar de ello, reconocieron que el estilo *Maniac*, mostraba una iniciativa de juego mayor que el resto.

El participante intermedio también percibió una tendencia conservadora en los bot LLM, indicando que los bots realizaban demasiado *folds* y pocas acciones agresivas. En sus respuestas destacó que los agentes cumplían correctamente las reglas y respondían de forma coherente, pero consideró que su comportamiento todavía estaba lejos de representar un reto significativo para un jugador con un poco más de entendimiento en el póker. Por el contrario, valoró positivamente la prudencia mostrada por algunos bots a la hora de aceptar apuestas y asumir riesgos, como el tight-passive.

La red neuronal obtuvo la mejor valoración. Tanto los jugadores principiantes como el intermedio coincidieron en señalar que este agente mostraba un comportamiento más consistente y cercano al de un jugador humano. Destacando su capacidad para seleccionar adecuadamente las manos que disputar, reaccionar de forma consistente ante las acciones del rival y mantener una estrategia más coherente durante toda la partida.

No obstante, el jugador experto ofreció una valoración más crítica. Aunque reconoció que los distintos bots eran funcionales y mostraban ciertos comportamientos estratégicos coherentes, consideró que todos ellos presentan todavía limitaciones importantes para competir frente a jugadores experimentados.

En cuanto al FSM, las opiniones de los participantes coinciden con los resultados cuantitativos obtenidos, destacando un comportamiento menos consistente, más predecible y con una tendencia a alcanzar *showdowns* con manos débiles.

Estos experimentos permiten validar el funcionamiento del framework, al poner a prueba diferentes modelos dentro de una misma partida, compartiendo las mismas reglas, condiciones y contexto de juego. De esta forma, los diferentes bots pueden enfrentarse directamente entre sí, tomar decisiones sobre una misma partida y generar los datos que posteriormente pueden comprobarse bajo los mismos criterios. La posibilidad de realizar este proceso sin modificar la lógica de la partida para cada arquitectura demuestra que el framework proporciona el entorno común planteado como objetivo del proyecto.

Capítulo 6 - Conclusiones y trabajo futuro

El desarrollo de este trabajo ha permitido crear una aplicación de No-Limit Texas Hold'em integrando en un mismo entorno de juego a humanos y bots. Más allá de ofrecer una implementación funcional del juego, la aplicación se ha concebido como una herramienta sencilla y extensible orientada a la experimentación con distintos modelos de inteligencia artificial en un juego imperfecto como el póker. Durante la implementación nos hemos encontrado varios problemas, principalmente, el de saber cómo configurar adecuadamente los hilos y conexiones del juego sin que quede algún bug pendiente o quede bloqueado el juego, nos ha llevado tiempo entender que construir una red de conexiones en el que se pueda simultanear varias partidas es más complejo de lo que pensábamos y tener en consideración todos los posibles errores más. Una vez comprendido el grandísimo tiempo que nos llevaría hacer una aplicación con múltiples partidas jugándose a la vez, nos enfocamos en implementar una arquitectura, claramente separada por cliente, servidor y los componentes compartidos, para una única partida. Esta estructura ha demostrado poder soportar distintos tipos de jugadores, gestionar la lógica de las partidas y mantener una comunicación consistente entre los diferentes elementos del sistema.

Uno de los aspectos más relevantes del proyecto y el que nos ha supuesto un mayor desafío es la creación de un mecanismo de integración de bots independiente de la tecnología empleada para su implementación. Debido al bajo nivel de experiencia en cuanto a desarrolladores de modelos de inteligencia artificial, la primera dificultad con la que nos enfrentamos fue implementar los diferentes tipos de bots, ya que durante la carrera no hemos podido aprender en profundidad todos los conocimientos necesarios, teniendo que documentarnos por otros medios. El segundo contratiempo fue el hallar una forma de integración de los bots, lo más fácil y automática posible, esto lo conseguimos gracias al diseño e implementación de una arquitectura distribuida que separa las responsabilidades, facilitando la mantenibilidad del sistema y sin afectar el núcleo de la aplicación.

Por otra parte, el estudio empírico nos ha permitido analizar el comportamiento de los distintos modelos de inteligencia artificial en escenarios variados: bot contra humano, mesas mixtas y partidas completamente

automatizadas. La diversidad de pruebas realizadas nos ha facilitado la obtención de una visión global de todos los tipos de bots, sobre, por ejemplo, el funcionamiento de cada uno, sus debilidades y fortalezas, el método de juego según el nivel de juego del rival o la influencia de factores importantes como la agresividad.

En conjunto, este trabajo demuestra la viabilidad de utilizar una forma claramente unificada para integrar, analizar y comparar diferentes tipos de bots en un entorno de póker. Nuestra aplicación, además de cumplir con los requisitos establecidos desde el inicio del proyecto, proporciona una base sólida para futuras investigaciones relacionadas con la integración de inteligencia artificial en juegos como el póker, no solo constituye una solución funcional de partidas de NLHE, sino que también sirve de herramienta de experimentación y enseñanza en el desarrollo de nuevas técnicas de inteligencia artificial.

Comenzando por la parte del cliente, un gran añadido podría ser un sistema de actualizaciones para adaptarse a las nuevas versiones del servidor sin necesidad de recopilaciones, lo que mejoraría la experiencia y comodidad del usuario frente a nuevas funcionalidades.

En el apartado gráfico, podríamos tener un diseño y una serie de funcionalidades similares a otras aplicaciones referentes orientadas al aprendizaje, estudio y práctica del póker como GTO Wizzard, con el fin de ofrecer una plataforma atractiva y extremadamente útil a jugadores con cualquier nivel de experiencia. Otros detalles menos relevantes pero que podrían mejorar la experiencia del usuario y aportar un toque más moderno y llamativo es un sistema de chat mediante mensajes predefinidos y reacciones y visualizar de mejor manera la mano ganadora, resaltando las cartas implicadas en la mano del jugador ganador y el tipo de mano.

Acerca de la aplicación servidor, una de las primeras adiciones podría ser una base de datos para poder almacenar gran cantidad de cuentas de usuarios, sus historiales, perfiles de juego, estadísticas personales a lo largo de todo su historial de partidas y cualquier tipo de información que pueda ser de interés. Este punto debe venir acompañado de una mejora en la gestión de conexiones de usuario y desarrollo de partidas, con el fin de soportar la conexión de un gran volumen de usuarios y partidas simultáneas, como ocurre en otras aplicaciones o servicios software modernos.

Con esta nueva arquitectura se podrían obtener una gran cantidad y variedad de métricas acerca de la calidad y estilo de juego de cada usuario o bot, lo que supondría una mejora significativa a la utilidad y relevancia de estos datos.

Otras incorporaciones realmente interesantes en nuestra aplicación, que debido a falta de tiempo y a tener que priorizar otros aspectos hemos tenido que descartar, son distintos "modos de juego" en los que se puedan jugar partidas de póker completamente naturales o partidas más didácticas en las que un "maestro" pueda controlar por completo el contexto del juego. Esto puede ocurrir seleccionando las cartas a revelar en la mesa o a entregar a los jugadores, deshacer acciones y cualquier otra herramienta que sirva para estudiar y replicar cualquier escenario relevante como forma de análisis, bien sea con otros humanos o con bots.

Por último, otro sistema que podría ser realmente llamativo es un historial de partidas pasadas que permite reproducirlas paso a paso y jugar de forma normal a partir de cierto instante, como si de una nueva partida se tratase, creando una rama que puede deshacerse para seguir el curso original.

Introduction

In this chapter, we will present the different sections of the document, along with their objectives, context, and the challenges encountered throughout the development process, among other aspects.

Motivation and scope

The idea of intelligent machines or beings created by humans has been a recurring theme throughout human history, ranging from classic literary works such as *The Travels of Gulliver* by *Jonathan Swift*[1] to more contemporary films such as *The Matrix* by the *Wachowski* sisters.

Modern society has succeeded in introducing machines into virtually every domain traditionally dominated by humans, including areas once considered beyond their reach, such as strategic games like checkers and chess.

Today, artificial intelligence systems outperform humans in the aforementioned strategy games. However, poker remains a particularly complex domain in which AI systems still face unique challenges when competing against human players.

For this reason, we found the prospect of developing a tool to evaluate, analyze, observe, and interact with different types of artificial intelligence in the context of poker particularly appealing, especially given the recent rise of more advanced AI models.

Regarding the scope of the project, it is important to highlight the two primary purposes of the application.

First, the platform can be used by developers to create, train, test, fine-tune, and evaluate their own artificial intelligence model, regardless of their underlying approaches. It also enables the integration of third-party models for the same purposes. These activities may take place in mixed environments, consisting of both human players and bots, or in fully automated settings composed exclusively of bots under a variety of game configurations.

Second, and again with developers as a key target audience, the application provides the flexibility to extend or enhance its functionality through a range of design

decisions aimed at delivering a software product that is both adaptable and capable of evolving to meet future requirements.

In addition to its intended use by developers, the application may also be used by non-technical users to organize games involving humans, bot, or a combination of both in a completely informal, recreational, or educational setting, either individually or as part of a group.

Objective

The primary objective of this project is to develop an application that enables users to play No-Limit Texas Hold'em (NLHE) games while also serving as a tool for the easy and extensible integration of any type of bot.

The main motivation behind this project stems from the need for an intuitive platform that facilitates the integration and evaluation of artificial intelligence models within an imperfect-information gaming environment such as poker. To achieve this, we designed a distributed architecture that allows new types of bots to be added with minimal effort, regardless of the technology used in their implementation. Furthermore, the project pursues the following secondary objectives:

1. Develop a fully functional multiplayer NLHE poker application, allowing both human participants and bots to play together in the same game.
2. Design an extensible and distributed architecture that facilitates the integration of different types of bots without requiring modifications to the core of the application.
3. Integrate different agents into poker games, regardless of the underlying artificial intelligence model.
4. Implement monitoring and analysis mechanisms capable of recording relevant information about the games and the decisions made by the bots.

Work schedule

The development of this application was conceived following an iterative and incremental approach, allowing the system to be progressively built through successive cycles of analysis, design, implementation, and testing. This approach made it possible to begin with a basic initial structure and gradually extend it as new

functionalities were incorporated, while reviewing and refining decisions made in earlier stages whenever necessary.

Based on this methodology, the work was organized into several phases that illustrate the evolution of the project, from the initial analysis and design stages through to the experimentation and evaluation of the application. Although these phases are described individually for the sake of clarity, the development process was not strictly linear, with each stage being definitively completed before the next one began. On the contrary, the different components of the project evolved in parallel, allowing observations made during implementation and testing to reveal new requirements and drive refinements to the original design.

The following sections describe the planning approach used to organize the project, as well as the activities carried out during each phase of its development.

Project planning

As the project was developed by a team of three members, it was necessary from the very beginning to establish an organizational approach that would allow the workload to be distributed effectively while maintaining a shared understanding of the project's status. The planning process was not defined as a fixed allocation of tasks at the start of the Final Degree Project; instead, it evolved throughout development in response to emerging requirements. As a result, tasks could be divided, modified, prioritized, or reassigned whenever necessary.

To organize the work, we adopted several principles of the Scrum methodology, adapting them to the specific characteristics of our project and to a team consisting of only three members. Scrum was not followed strictly, as no formal Scrum roles were assigned and no fixed-duration sprints were established. Instead, we relied on an incremental planning approach based on task management and regular project reviews.

From the outset of the project in September 2025, GitHub was used as the primary platform for organizing development, as it was the tool with which all the team members were most familiar. A shared repository was created, and GitHub Projects was used to manage tasks through a project board. This allowed us to create and assign tasks, track their progress, and distinguish between tasks that were pending,

ready to start, in development, or awaiting review. As a result, we were able to coordinate our work effectively and maintain continuous visibility of both outstanding tasks and each team member's responsibilities.

Coordination was carried out primarily through regular meetings, held both in person and online via Microsoft Teams, typically three to four times per month, although the frequency varied according to the needs of the project. During these meetings, we reviewed completed work, discussed challenges encountered, and defined the next set of tasks. The main topics addressed in each meeting were recorded in a project log to maintain a history of the decisions that had been made.

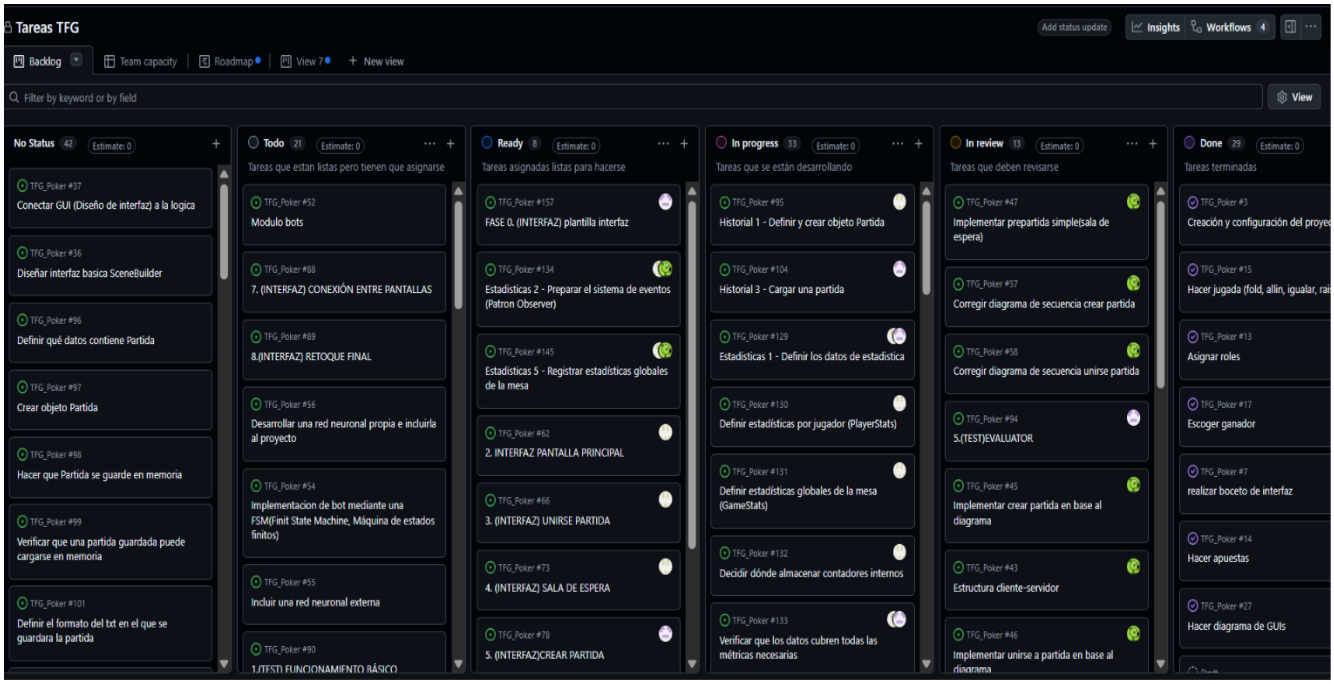


Figure 6-1. Example of the task backlog used during the project development

For source code management, Git was used within the shared repository. The main and develop branches served as the primary branches, while each team member worked on individual feature branches for their assigned tasks. This ensured that changes remained isolated from the shared codebase until they were ready to be integrated throughout Pull Requests. Consequently, team members were able to work in parallel without directly affecting the common branches and while benefiting from code reviews prior to integration.

During the initial stages of the project, it was necessary to establish a common workflow, define how tasks would be organized, and select the tools that would be used for coordination like the one showed in the Figure 6-1. Once these foundations

had been put in place, team members were able to work independently yet simultaneously, while maintaining a shared, controlled, and consistently managed codebase.

Phase I: Framework Analysis and Design

The objective of this initial phase was to establish a solid foundation for the development of the rest of the application. During the first few weeks, a significant amount of time was devoted to analyzing the project's requirements, defining its structure, and identifying previous projects developed by the team members and searching for repositories and other resources that could serve as useful references. Among the resources identified was a poker hand evaluation implementation, which was later used as a reference during development.

During these early weeks, most of the work was carried out collaboratively. On 18 September 2025, we produced an initial high-level class diagram for the game logic, which was subsequently reviewed and expanded until a more comprehensive class diagram was completed on 29 September. Following this, on 2 October, we defined the application's main use cases, covering player and card management, game configuration, and the various actions that can be performed during a poker hand. This process enabled us to refine the system's functionality and begin translating requirements into concrete development tasks.

At the same time, we began holding meetings with the project supervisor to review the project's objectives and scope. During this stage, we realized that we had initially proposed an excessive number of features. After reviewing them together with the supervisor, we established priorities and distinguished those that were essential for achieving the project's primary objectives.

Once these priorities have been identified, we prepared an initial sprint backlog on 9 October. Over the following weeks, the first implementations and tests were carried out to validate the proposed architecture. Development of the system's core logic began on 16 October, and debugging mechanisms were incorporated on 22 October. Finally, by 5 November, a first functional version of the application's foundation had been completed.

As a result, we successfully established the initial structure of the application, defined its key functionalities, and aligned its scope with the objectives of the Final Degree Project. This phase provided a solid starting point for the subsequent stages of development.

Phase II: Bot Logic Development

The main objective of this second phase was to develop the necessary logic to run a game on No-Limit Texas Hold'em and integrate different types of bots. This phase was one of the most important stages of the project, as it represents the main contribution of the thesis. For this reason, rather than limiting ourselves to developing a specific set of bots, we progressively designed and tested a structure that would allow users to integrate their own implementation and use them to play poker.

During the initial stages, we completed the development of the core game logic required to run a game and manage player actions. To achieve this, we used several projects related to this field as references, adapting the solutions that could be useful to the specific requirements of our application. Once this foundation was in place, we began integrating the bots incrementally. The first bot was developed in February 2026 and was based on a local Large Language Model. During this stage, we investigated different alternatives by consulting repositories and platforms such as Hugging Face and R, and testing different models until we found those that best suited the project's requirements. In March, we incorporated a second bot based on an online language model, and later, in April, we added bots based on a Neural Network and a Finite State Machine.

As we incorporated the bots, we also continuously modified and improved the framework. The goal was to ensure that adding a new implementation would not require changes to the core game logic. In this way, each new bot not only expanded the capabilities of the application but also allowed us to test and further improve its flexibility.

As a result, we obtained the first functional version of the framework capable of integrating different types of bots and allowing them to participate in the same poker games.

Phase III: Graphical User Interface

Up to this point, a large part of the testing had been carried out directly on the game logic, so we needed an interface that would allow us to make use of the functionalities we had developed. Before starting the implementation, we spent several weeks designing the interface. Around November, each team member prepared their own proposals, which we then brought together by comparing ideas and deciding which elements should be incorporated.

Once the overall design had been established, we began developing the main containers and basic components using JavaFX. The interface development was, in fact, an iterative process, with the different views being incorporated alongside the development of the various implementations. We carried out several tests to verify the implemented actions and made the necessary adjustments as development progressed. By around April, we had completed the final version of the interface, which was presented to the supervisor, reviewed, and tested in order to identify any aspects that needed to be modified.

Phase IV: Distributed Architecture Development

The main objective of this fourth phase was to organize the application using a distributed architecture that would allow the responsibilities of its different components to be clearly separated. We planned this division from the early stages of the project, so we did not wait until development was complete to separate the application. Instead, we incorporated new functionalities progressively using this architecture as the foundation.

This application was organized into three main components: the client, the server, and a shared module. This separation was carried out in parallel with the rest of the development. Whenever we added a new functionality, we determined which part belonged to the server, which to the client, and what information needed to be shared between them. One of the team members had more experience with this type of architecture than the rest, so they took on the role of the main contributor in this area. Throughout the process, we carried out communication tests to identify potential issues and make the necessary modifications, rather than waiting until the entire application had been completed.

In this way, we achieved an architecture in which the client and server could operate independently, separating the game logic from the user interface. Once this architecture had been fully defined in February, development progressed at a much faster pace.

By April 2026, we had achieved a fully functional and distributed final version capable of running poker games and integrating the different types of bots that had been developed.

Phase V: Experimentation and Evaluation

The main objective of this fifth phase was to carry out the necessary tests to evaluate the integration and performance of the different bots and to verify the application's logic. Before starting the testing process, we spent some time working with our supervisor to define the scenarios we wanted to analyze and determine how the game should be organized so that the results obtained would be meaningful.

An important part of the work involved finding participants who could take part in the test, with different levels of poker proficiency, as we wanted to have enough games and situations for the resulting statistics to be representative. This required coordinating the participants' schedules and adapting the sessions to make them feasible. As a result, the testing process extended over several months.

The experimental sessions were conducted mainly throughout June and August 2026. During this period, we ran games and collected the necessary data to later analyze the bots behaviour, while addressing any error that arose and implementing the changes that were deemed necessary.

The purpose was not to verify that the application worked correctly, but also to obtain enough information to study the differences between the bots and assess the behaviour of their different strategies and the logic behind them. We paid particular attention to maintaining sufficiently consistent testing conditions, as this provided the basis for studying the bots behaviour and drawing the project's conclusions.

Phase VI: Documentation and Dissertation Preparation

The objective of this final phase was to collect and organize all the work carried out throughout the development of the project and to prepare the TFG report. Although the final writing was mainly completed during the last few months, the documentation process began from the very start of the project. Throughout the development process, we kept a record of meetings, tasks, decisions made, and issues encountered. This allowed us to reconstruct the evolution of the project and use it as a basis for writing the report. In addition, we had defined the structure of the report from the early stages, so when we began writing it, we already had a clear idea of the content that needed to be developed.

From June 2026 onward, we began dedicating part of our work to preparing the report. During the following weeks, we assigned the different sections, drafted their content, and reviewed the existing documentation to ensure that it accurately reflected the development process.

By mid-August, most of the reports had already been drafted. At that point, one of the main elements that remained to be completed was the incorporation of the results obtained during the experimentation phase, the corresponding statistics, and their subsequent analysis. These were added once the testing process had been completed, ensuring that the report included all the final data.

In this way, the work carried out during this phase focused mainly on completing and giving the project's documentation its final form, incorporating the latest data obtained during the experimentation phase.

Document structure

The remainder of this document is organized into the following chapters:

- Chapter 2: Contextualization. This chapter introduces the theoretical and technological concepts required to understand the project's environment and underlying context.
- Chapter 3: Design. This chapter describes the different components and architectural elements of the application.

- Chapter 4: Implementation. This chapter provides a detailed explanation of the implementation of the components discussed in the design chapter.
- Chapter 5: Empirical study. This chapter presents the experimental setup, the tests conducted, and the analysis of the results obtained from the various scenarios in which the bots were evaluated.
- Chapter 6: Conclusions and future work. The final chapter summarizes the main conclusions drawn from the project and outlines potential directions for future development and research.

The material generated in this work (code, test, images file, etc.) is stored in the following GitHub repository.

Conclusions and future work

The development of this project has resulted in the creation of a No-Limit Texas Hold'em application that integrates humans and bots within the same gaming environment. Beyond providing a functional implementation of the game, the application has been designed as a simple and extensible tool aimed at experimenting with different artificial intelligence models in an imperfect-information game such as poker.

Throughout the implementation process, we encountered several challenges. The main difficulty was determining how to properly configure the game's threads and connections without leaving unresolved bugs or causing the game to become blocked. It took us some time to understand that building a network of connections capable of handling multiple simultaneous games were considerably more complex than we had initially anticipated, particularly when taking all possible error scenarios into account. Once we realized the considerable amount of time that would be required to develop an application capable of running multiple games simultaneously, we decided to focus on implementing a clearly separated architecture consisting of the client, the server, and the shared components, designed to support a single game. This structure has proven capable of supporting several types of players, managing game logic, and maintaining consistent communication between the different elements of the system.

One of the most relevant aspects of the project, and one of the greatest challenges we faced, was the creation of a bot integration mechanism that was independent of the technology used to implement each bot. Due to our limited experience as developers of artificial intelligence models, the first difficulty we encountered was implementing the different types of bots. Throughout our degree, we had not had the opportunity to acquire an in-depth understanding of all the knowledge required for this task, so we had to complement our existing knowledge through additional research and documentation. The second major challenge was finding a way to integrate the bots as easily and automatically as possible. We achieved this through the design and implementation of a distributed architecture

that separates responsibilities, making the system easier to maintain without affecting the application's core.

Furthermore, the empirical study allowed us to analyze the behaviour of the different artificial intelligence models in a variety of scenarios, including bot vs human games, mixed table, and fully automated games. The diversity of the tests carried out provided us with a comprehensive overview of all the types of bots, including aspects such as how each one operates, their strengths and weaknesses, how their playing style changes depending on the opponent's skill level, and the influence of important factors such as aggressiveness.

Overall, this project demonstrates the feasibility of using a unified approach to integrate, analyze, and compare diverse types of bots within a poker environment. In addition to meeting the requirements established at the beginning of the project, our application provides a solid foundation for future research into the integration of artificial intelligence into games such as poker. It therefore not only constitutes a functional solution for playing NLHE games but also serves as an experimentation and teaching tool for the development of new artificial intelligence techniques.

Regarding the client side, a valuable addition would be an update system capable of adapting to new server versions without requiring recompilation. This would improve the user experience and make it more convenient to introduce new functionalities.

From a graphical perspective, the application could incorporate a design and a set of functionalities similar to those found in other platforms focused on poker learning, study, and practice, such as GTO Wizard. This would allow us to provide an attractive and highly useful platform for players of all experience levels. Other less significant features that could nevertheless improve the user experience and give the application a more modern and engaging feel include a chat system based on predefined messages and reactions, as well as a clearer way of displaying the winning hand by highlighting the cards involved and indicating the type of hand achieved.

Regarding the server application, one of the first additions could be a database capable of storing a large number of user accounts, together with their game histories, playing profiles, personal statistics throughout their entire playing history, and any other relevant information. This addition should be accompanied by

improvements to the management of user connections and game execution, with the aim of supporting a large volume of users and simultaneous games, as is common in modern applications and software services.

With this new architecture, a large variety of metrics could be obtained regarding the quality and playing style of each user or bot, which would significantly increase the usefulness and relevance of the collected data.

Other particularly interesting additions to our application, which we had to discard due to time constraints and the need to prioritize other aspects of the project, would be the introduction of different "game modes". These could allow users to play either completely natural poker games or more educational sessions in which a "teacher" could have full control over the context of the game. This could be achieved by allowing the teacher to select which cards are revealed on the table or dealt to the players, undo actions, and use any other tools required to study and reproduce relevant scenarios for analytical purposes, either with other human players or with bots.

Finally, another particularly interesting feature would be the history of previous games that would allow users to replay them step by step and resume normal gameplay from a specific point, as if it were a new game. This would create a branch of the original game that could subsequently be discarded, allowing the user to return to and continue following the original course of the game.

CONTRIBUCIONES PERSONALES

Al tratarse de un proyecto realizado por tres integrantes, el trabajo se ha dividido entre los miembros del equipo para facilitar la organización y el desarrollo de las diferentes partes del proyecto. En este apartado se recoge el trabajo realizado por cada integrante durante el desarrollo del proyecto, indicando las tareas en las que ha participado y las partes que ha desarrollado de forma individual.

Carla Toapanta Taipe

Mi contribución en el proyecto se ha centrado en el desarrollo de la GUI, la correcta conexión entre cliente y servidor, la implementación de funcionalidades relacionadas con la experiencia de juego, la integración y configuración de los bots en la interfaz, la realización del formulario que los participantes del primer experimento tuvieron que completar y la participación en el experimento de un humano contra un bot para el estudio empírico.

Una parte importante de mi trabajo ha consistido en el diseño e implementación de las diferentes interfaces que constituyen la aplicación. He desarrollado las ventanas de creación de partida, y he participado en la incorporación de jugadores y bots, sala de espera y visualización de la partida.

Para lograr una interfaz intuitiva y coherente con el funcionamiento de una partida de No-Limit Texas Hold'em se necesita desarrollar varios componentes visuales. Entre las principales contribuciones realizadas destacan: el diseño y desarrollo de las ventanas de creación de partidas; la implementación de la ventana para la integración de bots a la partida de forma dinámica según el tipo de bot; la participación en la implementación de la sala de espera; el desarrollo de la interfaz principal del juego; los diseños visuales de la mesa de póker y de los componentes de cada jugador (nombre, dinero que le queda, dinero apostado, última acción realizada, avatar y la ficha del dealer en caso correspondiente); la implementación del estilo CSS para las diferentes pantallas; la configuración de elementos de navegación entre vistas y la generación dinámica de avatares personalizados en función de si un jugador es un humano o un bot.

También participé (en colaboración con mi compañero) en la conexión entre la interfaz gráfica y la lógica interna de la aplicación. Lar tareas que realicé están relacionadas con la transferencia de información entre las distintas ventanas de configuración, la sincronización de los datos de partida con los elementos visuales mostrados en pantalla, la gestión de eventos producidos durante el juego y la actualización dinámica de la información de la mesa.

Otra de mis contribuciones fue en el desarrollo de diferentes funcionalidades necesarias para el correcto funcionamiento de las partidas. En primer lugar, el sistema de temporizador de turno, incluyendo tanto la lógica de funcionamiento como su representación gráfica en la interfaz. Esta funcionalidad permite controlar el tiempo que cada jugador dispone para realizar una acción, y en caso de que el tiempo expire realizar una acción automática. En segundo lugar, la implementación de uno de los elementos probabilísticos más importantes en el póker, el equity, desarrollado por mi compañera, y que me encargué de visualizarlo dinámicamente en la partida para cada jugador. Por último, implementé funcionalidades como la posición dinámica del dealer, el indicador del turno activo y la visualización de las acciones realizadas por los jugadores. Estas características mejoran la comprensión del estado de la partida y facilitan el seguimiento de las acciones realizadas por los jugadores.

Una de las contribuciones más relevantes durante el desarrollo del proyecto fue la integración de los distintos tipos de bots a la interfaz gráfica de forma dinámica, las tareas que realicé para conseguirlo fueron: la automatización a la hora de incorporar bots a la partida, selección dinámica del tipo de bot, configuración de diferentes perfiles de juego, visualización diferenciada para cada tipo de modelo de IA.

Desarrollé y mejoré de elementos gráficos, como los avatares dinámicos según si el jugador es humano o bot y la visualización de diferentes diseños de barajas y mesas. Creados con la intención de dotar a la aplicación de una apariencia más cercana a los usuarios.

Finalmente, participé en el estudio del primer experimento basado en el enfrentamiento entre un humano contra un bot, en mi caso jugué como nivel principiante contra dos tipos de bots: bot LLM Ollama (y cuatro de sus posibles estilos

de juego) y bot red neuronal, sumando un total de diez partidas jugadas. Y diseñé el formulario para la recogida de información del participante del experimento uno.

Óscar Fabian Pineda Germán

Mi trabajo en este proyecto se ha centrado principalmente en el diseño y desarrollo de la arquitectura y configuración de la aplicación distribuida, debido a mi experiencia previa con sistemas distribuidos. En este ámbito, mi trabajo ha abarcado el diseño e implementación de los sistemas implicados en la comunicación y la lógica de las aplicaciones servidor y cliente, incluyendo la inicialización y gestión de los recursos necesarios para el desarrollo de las partidas, la definición de los protocolos de comunicación entre ambas aplicaciones y la gestión de situaciones de error, como las desconexiones de los jugadores. Asimismo, he trabajado en el desacoplamiento entre el frontend y el backend con el objetivo de facilitar el mantenimiento y evolución de la aplicación. Además de esta parte fundamental, también he participado en la sincronización y comunicación entre cliente y servidor, así como en la implementación e integración de los distintos componentes necesarios para el funcionamiento del juego de póker.

Otra parte relevante de mi trabajo ha sido el desarrollo del sistema de notificadores de jugador, diseñado para permitir el acoplamiento de diferentes tipos de bots a un jugador virtual mediante una interfaz común y bien definida. Para ello, he desarrollado la jerarquía funcional de notificadores y de los distintos tipos de jugadores, así como en la definición de una interfaz adaptada a las características de cada tipo de bot, favoreciendo así la modularidad y extensibilidad del sistema. En relación con esto, también he participado en la integración de los diferentes tipos de bots, realizando tareas de corrección de errores y calibración de su comportamiento para garantizar un funcionamiento adecuado y coherente dentro de las partidas.

También he llevado a cabo tareas de investigación, evaluación e integración de algoritmos y bibliotecas de terceros con el objetivo de agilizar el desarrollo y mejorar la productividad del equipo. Entre los recursos estudiados e integrados se encuentra el algoritmo evaluador de manos de cinco cartas de *Cactus Kev*, empleado para facilitar la evaluación de las jugadas de póker. Asimismo, he investigado y utilizado la librería *Lang4Chain* para la integración de APIs

correspondientes a diferentes modelos de lenguaje de gran escala (Large Language Models), así como la biblioteca *RLCard*, orientada al desarrollo, entrenamiento y evaluación de sistemas de aprendizaje por refuerzo aplicados a juegos de cartas. La selección de estas herramientas ha requerido analizar previamente su utilidad, posibilidades de integración y adecuación a las necesidades concretas del proyecto.

Otra de las tareas relevantes ha sido la documentación de la aplicación servidor mediante *Javadoc*, con el objetivo de proporcionar al equipo de trabajo y a futuros desarrolladores una visión clara y completa del funcionamiento del sistema y de sus principales componentes. Esta documentación busca facilitar tanto la comprensión de la arquitectura como el mantenimiento y la incorporación de nuevas funcionalidades, dejando constancia de las decisiones y elementos relevantes de la implementación.

Además de las tareas directamente relacionadas con el desarrollo, he participado en la supervisión general, planificación del proyecto y organización del equipo de trabajo. Esta participación ha incluido la coordinación de los miembros del equipo, la distribución y seguimiento de las tareas asignadas y la propuesta de diferentes soluciones a los problemas surgidos durante el desarrollo, especialmente en las reuniones periódicas de seguimiento. También he contribuido a plantear mejoras relacionadas con la calidad del software y a realizar pruebas destinadas a verificar el correcto funcionamiento de aquellas partes del sistema consideradas críticas.

Por otra parte, he participado activamente en el diseño y desarrollo de la interfaz gráfica de usuario (GUI), interviniendo tanto en la toma de decisiones relativas al diseño de la interfaz como en su arquitectura interna y funcionalidades. Además, he colaborado en la corrección de errores tanto lógicos como visuales y en la calibración y sincronización de la comunicación entre el frontend y el backend, contribuyendo a garantizar una interacción adecuada entre ambas partes de la aplicación.

Finalmente, mi participación en el estudio experimental se ha centrado principalmente en la recopilación y tratamiento de los datos obtenidos durante las

diferentes pruebas realizadas sobre el sistema. Para facilitar este proceso, he desarrollado un script en Python destinado a automatizar la recopilación y el procesamiento de los resultados, permitiendo posteriormente analizar estos datos y utilizarlos como parte del estudio experimental presentado en la memoria final.

Valeria Corina Pulido Lozada

Mi participación en el proyecto ha estado principalmente orientada al diseño y desarrollo de la GUI, la integración de los diferentes tipos de bots, el desarrollo de la monitorización e historial de las partidas y la definición e implementación de las estadísticas utilizadas posteriormente en el estudio empírico. Además, he participado en la lógica del juego, en la organización del proyecto y en la realización del estudio experimental.

Una de las contribuciones más relevantes de mi trabajo fue la integración de los diferentes tipos de bots en el framework. El trabajo comenzó con la integración del bot basado en un modelo de lenguaje local, que fue la primera implementación y sirvió posteriormente como base para incorporar el resto de tecnologías. Para esta primera integración realicé una fase previa de investigación sobre diferentes modelos disponibles, especialmente en repositorios como *Hugging Face*. Busqué modelos que pudieran ejecutarse localmente y que estuvieran orientados específicamente al juego de No-Limit Texas Hold'em. Durante este proceso instalé y probé diferentes alternativas, descartando aquellas que no se adaptaban a las necesidades del proyecto hasta seleccionar el modelo utilizado finalmente. Una vez seleccionado el modelo, desarrollé, junto con mi compañero, la estructura necesaria para integrar el LLM local y conectarlo con el flujo de ejecución de las partidas. También trabajé en la adaptación de la información de la partida y en la construcción del contexto y *prompt* utilizado por el modelo, así como en el procesamiento necesario para interpretar las respuestas generadas y obtener las acciones correspondientes. Además, definí los diferentes estilos de juego utilizados por los bots, con la clase `PokerStyle`, con el objetivo de establecer comportamientos diferenciados durante los experimentos. También contribuí en la integración del LLM online mediante un servicio externo, a partir de la investigación previa llevada a cabo por mi compañero.

Otra parte importante de mi trabajo fue el desarrollo del sistema de monitorización y del historial de las partidas, utilizado posteriormente para obtener la información necesaria para las estadísticas y los experimentos. Para ello, investigué diferentes formatos de historial y seleccioné el utilizado por PokerStars como referencia. A partir de esta referencia, desarrollé la clase `PokerHistory` para generar y mantener el historial de las manos, recopilando la información necesaria de la partida. También definí los puntos de registro e incorporé las acciones, cartas, cambios de fase y estados de los jugadores dentro del código. Además, registré información específica de los bots, como sus decisiones y tiempos de respuesta, así como otra información necesaria para los experimentos. Finalmente, configuré el sistema de *logging* para organizar los registros por partida y por mano.

También participé en la determinación de las estadísticas más relevantes para analizar el comportamiento de los bots y desarrollé el código necesario para obtenerlas a partir de la información registrada. Para ello, analicé y seleccioné las métricas que utilizaríamos para medir el comportamiento de los jugadores y desarrollé en el código los datos necesarios para calcular las estadísticas seleccionadas, así como la lógica para obtener métricas como VPIP, PFR, 3Bet, WTSD, WSD, WWSF y AF. Una parte especialmente importante de esta contribución fue el desarrollo del cálculo del equity. Para ello desarrollé `EquityCalculator`, encargado de calcular el equity de cada jugador mediante métodos estadísticos basados en simulaciones de Monte Carlo. Este cálculo se integró en las diferentes fases de la partida y posteriormente se utilizó tanto como información disponible para los bots durante la toma de decisiones como en el registro de datos empleado para el análisis de la memoria. También participé en la definición de las estadísticas y comparaciones incluidas finalmente en la memoria.

En cuanto al diseño y desarrollo de la GUI, participé en la implementación de diferentes elementos de la interfaz gráfica y en la organización de los diferentes elementos dentro de las pantallas, buscando una distribución adecuada para las funcionalidades de la aplicación. Junto con mi compañera, desarrollé componentes comunes, como botones, paneles y otros componentes reutilizables en diferentes partes de la aplicación. Además, diseñé y desarrollé las ventanas de inicio, el menú de creación y unión a una partida y el panel lateral con información adicional de la

aplicación. También participé, junto con mi compañera, en el desarrollo del estilo CSS común utilizado en las diferentes pantallas.

Participé también en el desarrollo y adaptación de parte de la lógica general de las partidas, reutilizando código procedente de proyectos y prácticas anteriores y adaptándolo a las necesidades de esta aplicación. Este trabajo incluyó la adaptación del código desarrollado previamente al nuevo framework, la modificación de diferentes componentes para incorporar las funcionalidades desarrolladas y la realización de correcciones y ajustes sobre la lógica durante las pruebas de funcionamiento.

Además del desarrollo de la aplicación, participé en la preparación de los datos y en la realización del estudio experimental. Junto con mi compañero, desarrollé el fichero de Python utilizado para procesar los historiales y preparar los datos necesarios para obtener las estadísticas, realizar las comparaciones y generar las gráficas incluidas en la memoria. También participé como jugador humano en el experimento de un jugador contra cuatro bots, realizando más de diez partidas. Finalmente, participé en la comprobación y análisis de los resultados obtenidos durante las partidas experimentales.

Por último, participé en la organización del desarrollo mediante Jira, proponiendo una estructura basada en tareas y subtareas y colaborando en la planificación y distribución del trabajo a lo largo del proyecto.

BIBLIOGRAFÍA

[1] Universidad Adolfo Ibáñez, «Gulliver y la inteligencia ridícula,» [En línea]. Available: https://www.uai.cl/columnas/artesliberales/gulliver-y-la-inteligencia-ridicula. [Último acceso: 08 07 2026].

[2] World Póker Federation, «World Póker Federation,» [En línea]. Available: https://worldpokerfederation.org/es/. [Último acceso: 20 06 2026].

[3] Vilatec Information Technology, «Jaque Mate al Farol: Cómo la IA Aprendió a Dominar el Póker Profesional,» [En línea]. Available: https://www.vilatec.com/jaque-mate-al-farol-como-la-ia-aprendio-a-dominar-el-poker-profesional/. [Último acceso: 19 06 2026].

[4] cc-poker, «La inteligencia artificial en el póquer: ¿amenaza o ayuda?,» [En línea]. Available: https://cc-poker.com/es/blog/ai-in-poker. [Último acceso: 19 06 2026].

[5] bluff central, «La IA llega a la mesa | herramientas que transforman el póker en 2025,» [En línea]. Available: https://bluffcentral.com/la-ia-llega-a-la-mesa-herramientas-que-transforman-el-poker-en-2025/. [Último acceso: 19 06 2026].

[6] David Sklansky, La Teoría del Póker, Two Plus Two, 2023.

[7] Matt Flynn, Sunny Mehta, Ed Miller, Professional No-Limit Hold 'em: Volume I, Two Plus Two, 2023.

[8] Pokerstars, «Reglas del póker online?,» [En línea]. Available: https://www.pokerstars.es/poker/games/rules/#Start. [Último acceso: 21 06 2026].

[9] TFG-G6830, «Inteligencia artificial aplicada a la teoría de juegos,» [En línea]. Available: https://uvadoc.uva.es/bitstream/handle/10324/71163/TFG-G6830.pdf?sequence=1&isAllowed=y. [Último acceso: 16 08 2026].

[10] Wikipedia, «Equilibrio de Nash,» [En línea]. Available: https://es.wikipedia.org/wiki/Equilibrio_de_Nash. [Último acceso: 16 08 2026].

[11]ActuIA, «Teoría de juego,» [En línea]. Available: https://www.actuia.com/es/tag/game-theory/. [Último acceso: 16 08 2026].

[12] IBM, «¿Qué es el machine learning (ML) o aprendizaje automático?,» [En línea]. Available: https://www.ibm.com/es-es/think/topics/machine-learning. [Último acceso: 16 08 2026].

[13] IBM, «¿Qué es el aprendizaje de refuerzo?,» [En línea]. Available: https://www.ibm.com/es-es/think/topics/reinforcement-learning#1268897081. [Último acceso: 16 08 2026].

[14] IBM, «¿Qué son las redes neuronales?,» [En línea]. Available: https://www.ibm.com/es-es/think/topics/neural-networks. [Último acceso: 16 08 2026].

[15] IBM, «¿Qué es el deep learning?,» [En línea]. Available: https://www.ibm.com/es-es/think/topics/deep-learning. [Último acceso: 16 08 2026].

[16] IBM, «¿Qué son los grandes modelos de lenguaje (LLM)?,» [En línea]. Available: https://www.ibm.com/es-es/think/topics/large-language-models. [Último acceso: 16 08 2026].

[17] Casino Barcelona, «El Póker: Un juego de cartas con un origen lejano y una historia fascinante,» [En línea]. Available: https://blog.casinobarcelona.com/el-poker-un-juego-de-cartas-con-un-origen-lejano-y-una-historia-fascinante. [Último acceso: 15 08 2026].

[18] GitHub-brianberns, «Counterfactual Regret Minimization Explaines,» [En línea]. Available: https://github.com/brianberns/CFR-Explained. [Último acceso: 16 08 2026].

[19] bubalo, «CFR Póker,» [En línea]. Available: https://www.bubalo.dev/blog/cfr_poker/. [Último acceso: 16 08 2026].