Trabajo de Fin de Grado

Aplicación didáctica para el juego del bridge

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Naixin Chen Zhou

TUTORIZADO POR:
Agustín Rafael Trujillo Pino
Gabriele Salvatore De Blasio

Junio 2025

ULPGC Universidad de Las Palmas de Gran Canaria | EII ESCUELA DE INGENIERÍA INFORMÁTICA

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D. Naixin Chen Zhou, autor del trabajo Aplicación didáctica para el juego del bridge, correspondiente a la titulación de Grado en Ingeniería Informática

SOLICITA

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida, haciendo constar que

[X] se autoriza / [ ] no se autoriza la grabación en audio de la exposición y turno de preguntas.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).

[X] No está previsto.

Y para que así conste firma la presente. (fecha en firma electrónica)

El/La estudiante

Fdo.————————

A rellenar y firmar **obligatoriamente** por el/la/los/las tutores

En relación con la presente solicitud, se informa:(firmar donde corresponda)

Positivamente (en caso de detección de copia, esta firma quedará invalidada)	Negativamente (justificación en TFT05)
Fdo.————————	Fdo.————————

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Agradecimientos

Deseo agradecer a mis tutores de Proyecto de Fin de Grado, Agustín Rafael Trujillo Pino y Gabriele Salvatore De Blasio, por su guía a lo largo de este trabajo. Su feedback en el aspecto del diseño enfocadas en la accesibilidad, la usabilidad y la experiencia de usuario fue crucial para lograr una interfaz de usuario amigable y fácil de usar.

También quiero agradecer al profesor de Bridge del curso extracurricular, Manuel Fernando Duque de Lara, de quien obtuve todo el conocimiento fundamental sobre el juego. Su enseñanza fue indispensable para comprender la base del juego. Este conocimiento resultó de vital importancia para la digitalización del juego de manera precisa.

Resumen

Este trabajo presenta el desarrollo en Unity del juego de cartas Bridge, centrado en las fases de subasta (limitada a aperturas Sin Triunfo sin intervención) y carteo.

El sistema implementado ofrece:

- Asistencia interactiva: El jugador puede solicitar pistas durante la subasta y el carteo, recibiendo recomendaciones basadas en estrategias básicas

- Jugadores IA: Los oponentes utilizan heurísticas para tomar decisiones coherentes con el nivel principiante

El proyecto busca hacer accesible el Bridge a nuevos jugadores, combinando mecánicas tradicionales con herramientas de ayuda interactiva.

Summary

This Project presents the development of the card game Bridge in Unity, focusing on the bidding phase (limited to No Trump openings without intervention) and card play.

The implemented system features:

- Interactive assistance: Player can request hints during bidding and card play, receiving recommendations based on basic strategies

- AI opponents: Computer controlled players use heuristic-based decision making adjusted for beginners

The project aims to make Bridge more accesible to new players by combining traditional gameplay with interactive learning tools.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Este capítulo aborda el contexto, la motivación detrás del proyecto y la problemática que busca resolver.

1.1. Contexto y motivación

El Bridge es un juego de cartas de gran prestigio, reconocido mundialmente por su complejidad estratégica, su profundidad intelectual y su capacidad para fomentar el razonamiento lógico y la comunicación. Por su complejidad, a menudo es percibido como un juego difícil de aprender para los principiantes, por sus reglas de subasta y carteo, la necesidad de un compañero y oponentes, y la carencia de herramientas interactivas que guíen el proceso de aprendizaje de forma didáctica y accesible.

La motivación principal de este trabajo es el deseo de hacer accesible al juego de cartas Bridge a los jugadores principiantes. Tradicionalmente, la enseñanza del juego se ha basado en clases presenciales, libros y la práctica constante con otros jugadores. Sin embargo, estos recursos no están al alcance de todos y pueden generar una curva de aprendizaje inicial frustrante. Un factor clave en la motivación de este trabajo fue la participación en un curso extracurricular de Bridge.

En este curso además de la enseñanza de las reglas y convenios de Bridge, se evidenció la curva de aprendizaje a la que se enfrentan los principiantes. La experiencia didáctica en el curso resaltó la necesidad de herramientas didácticas interactivas que faciliten la asimilación

de conceptos de Bridge.

Este trabajo se ha realizado con el propósito de facilitar el aprendizaje de las reglas básicas de Bridge, los convenios establecidos por la comunidad y estrategias básicas. Al digitalizar el juego se busca ofrecerles a los jugadores la posibilidad de jugar en caso de no tener en su cercanía gente interesada en Bridge.

1.2. Problema a resolver

El problema principal que este proyecto busca solventar es la alta barrera de entrada para principiantes en el juego del Bridge, esta barrera se caracteriza por:

- **Complejidad de reglas**: la subasta y carteo poseen un conjunto amplio de reglas y convenios que son difíciles de dominar sin una guía constante.
- **Falta de oponentes**: el Bridge es un juego de cuatro jugadores, lo que dificulta la práctica individual y autónoma.
- **Falta de interacción**: en partidas reales, los jugadores carecen de una explicación clara y concisa de la razón de las decisiones tomadas.
- **Falta de herramientas didácticas interactivas**: a pesar de que existen páginas web que permite el juego del Bridge, pocas ofrecen la posibilidad de obtener explicaciones de las jugadas.

La aplicación desarrollada busca solucionar estos problemas, ofreciendo a los jugadores una plataforma donde puedan aprender y practicar de manera iterativa y autodidacta.

Capítulo 2

Bridge

Este capítulo tiene como objetivo describir el juego del Bridge, se abordarán sus orígenes, la comunidad, los conceptos fundamentales y las reglas básicas.

2.1. Orígenes e historia del Bridge

El Bridge moderno es el resultado de las evoluciones de los juegos de cartas de bazas. Sus orígenes se remontan al Whist, un juego popular en Inglaterra en el siglo XVI, que sentó las bases de la mecánica de bazas y la coordinación en parejas. A lo largo de los siglos, el Whist fue refinado y se le añadieron conceptos como la subasta y la declaración de triunfo.

El término `Bridge` aparece por primera vez a finales del siglo XIX, con el "Biritch, or Russian Whist", un juego que tiene origen en Rusia, pero no fue hasta el siglo XX que se convirtió en el juego actual. El "Bridge-Whist" evolucionó hacia el `Bridge de Contrato` [13] y popularizado alrededor de 1920. Se introdujo un sistema de puntuación con penalización por los contratos no cumplidos y recompensaba a las subastas ambiciosas.

Desde entonces, el `Bridge de Contrato` se ha convertido en el juego de cartas más practicado a nivel mundial, con una larga historia de campeonatos, convenciones de subasta, libros dedicados a estrategias, etc.

Ilustración 2.1: Cartas de Bridge y subasta

Ilustración 2.2: Campeonato internacional de Bridge

2.2. Comunidad

El Bridge, más allá de ser un juego de cartas, es fundamentalmente un juego social. La comunidad global de Bridge es muy grande y activa, contando con millones de jugadores activos en clubs, torneos, online y en persona alrededor de todo el mundo.

2.2.1. Clubes y asociaciones

En ciudades y pueblos de todo el mundo, los jugadores se encuentran en clubes y asociaciones para participar en ligas y torneos, o simplemente jugar de forma casual. Estas asociaciones como la Asociación Española de Bridge (AEB) [3] en España o la American Contract Bridge League (ACBL) [2] en Estados Unidos, organizan eventos y promueven la enseñanza de Bridge.

2.2.2. Torneo y competiciones

Los torneos y competiciones de Bridge se organizan en todo el mundo, desde eventos locales hasta campeonatos internacionales, como el Campeonato Mundial de Bridge, organizado por la Federación Mundial de Bridge (WBF) [12]. Estos eventos no son solamente competiciones, sino también una oportunidad para los jugadores de conocer a otros jugadores y compartir sus conocimientos.

2.2.3. Plataformas Online

La digitalización de Bridge ha expandido enormemente la accesibilidad del juego. Plataformas como Bridge Base Online (BBO) [8] permiten a los jugadores jugar en línea de forma gratuita contra oponentes de todo el mundo.

2.3. Contexto General del Juego de Bridge

El Bridge es un juego de cartas para cuatro jugadores, utilizando la baraja inglesa estándar de 52 cartas (cuatro palos: picas, corazones, diamantes y tréboles; y 13 cartas por palo (2,3,4,...,Q,K,A). Los jugadores se dividen en dos parejas: Norte-Sur y Este-Oeste. El juego se desarrolla en tres fases: la subasta, el carteo y la puntuación.

- **La Subasta:** Es la primera fase del juego, consiste en una comunicación codificada entre los jugadores, que determina los jugadores atacantes y contrato final (número de bazas que los atacantes se comprometen a ganar y el palo de triunfo o sin triunfo).

- **El Carteo:** Es la segunda fase del juego, en la que los jugadores que ganaron la subasta (atacantes) intentan cumplir el contrato establecido en la subasta.

- **La Puntuación:** Es la última fase del juego, en la que se determinan las puntuaciones correspondientes a cada pareja dependiendo del contrato, las bazas ganadas, además de conceptos más avanzados como vulnerabilidad.

Las fases de subasta y carteo se desarrollan con interacción de los jugadores, estas se juegan por turnos en el sentido de las agujas del reloj, mientras que la puntuación no requiere interacción.

2.3.1. Conceptos claves de valoración de Manos

Para comprender las decisiones que se toman en la fase de subasta, es esencial conocer los siguientes conceptos:

- **Puntos de honor (High Card Points):** puntuación asignada a las cartas de honor (J=1, Q=2, K=3, A=4) en cada palo. Son la base de muchas decisiones de subasta.

- **Puntos de distribución (Distribution points):** puntuación adicional que se le asigna a una mano durante la subasta, dependiendo de la distribución de los palos en la mano.

    - **Fallo (Void):** 3 puntos, si un palo es vacío.

- **Semifallo (Singleton):** 2 puntos, si hay únicamente una carta en un palo.
- **Dubletón (Doubleton):** 1 punto, si hay 2 cartas en un palo.

- **Puntos de longitud (Length points):** puntuación adicional que se le asigna a una mano durante la subasta. Por cada carta en un palo con una longitud mayor a 8 se le asigna 1 punto.

2.3.2. Conceptos claves del Carteo

Para comprender el funcionamiento de la fase de carteo, es esencial conocer los siguientes conceptos:

- **Abridor (Opening Leader):** jugador que primero juega durante la fase de carteo
- **Palo de salida (Leading Suit):** representa el palo de la primera carta que se juega en una baza

2.3.3. Reglas básicas de Bridge

A continuación se presentan las reglas básicas de Bridge:

- **Subasta:**
  - Un jugador puede realizar una de las siguientes acciones durante su turno:
    - **Declarar (Declare)**: poner una apuesta representada por un nivel y un rango, esta debe ser superior a todas las apuestas anteriores
    - **Doblar (Double)**: incrementar los puntos que se obtiene en caso de ganar y los que se pierde en caso de perder
    - **Redoblar (Redouble)**: respuesta a un doblar que incrementa aún más los puntos de ganar y perder

    - **Pasar (Pass)**: pasar el turno

- La subasta termina cuando: no hay ninguna apuesta (4 pases seguidos) o hay una apuesta y hay tres pases seguidos

- El jugador que realiza la apuesta más alta gana la subasta, él y su pareja son denominados atacantes. El primero de la pareja que haya realizado la apuesta con el mismo rango que el contrato final es denominado declarante

- **Carteo:**

    - La pareja del declarante es denominada muerto y este muestra sus cartas tras la primera carta jugada. Este no puede realizar ninguna acción durante el carteo, en su turno juega el declarante

    - Un jugador debe jugar una carta del mismo palo que el palo de salida, si no tiene puede jugar una carta de otro palo

    - El ganador de una baza se determina dependiendo de la modalidad de juego:

        - **Modalidad de juego sin triunfo:** el ganador de la baza es el que juega la carta del mismo palo que el palo de salida y de mayor número

        - **Modalidad de juego con triunfo:**

            - Si ninguna carta es del mismo palo que el contrato final, se determina el ganador igual que en la modalidad de juego sin triunfo

            - Una carta del mismo palo que el contrato final siempre tiene mayor valor que cartas de otro palo

            - Si hay más de una carta del mismo palo que el contrato final, el ganador es el que jugó la carta de mayor número

Capítulo 3

Estado actual y objetivos iniciales

En este capítulo se presentan los objetivos planteados en el TFT01 y una descripción del estado actual de la temática del trabajo.

3.1. Estado actual

Con respecto al estado actual de la temática del trabajo, el desarrollo de aprendizaje del Bridge es un campo con un potencial considerable. Plataformas como **Bridge Base Online (BBO)** [**8**] dominan el juego en línea, ofreciendo una vasta funcionalidad que incluye partidas contra robot, partidas contra jugadores humanos, torneos. Sin embargo, BBO[8], aunque permite jugar con distintas modalidades y ofrece una experiencia muy completa a jugadores veteranos, no está diseñado como una herramienta de aprendizaje para principiantes que buscan aprender a jugar Bridge.

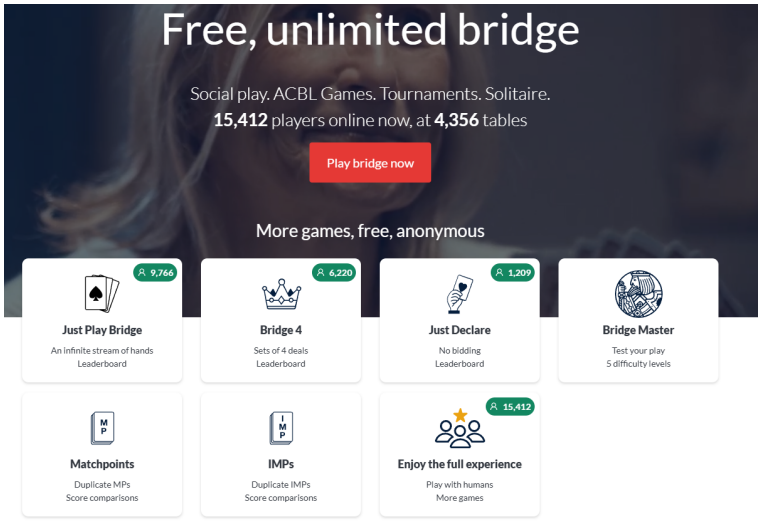

Ilustración 3.1: Modalidades de juego en Bridge Base Online

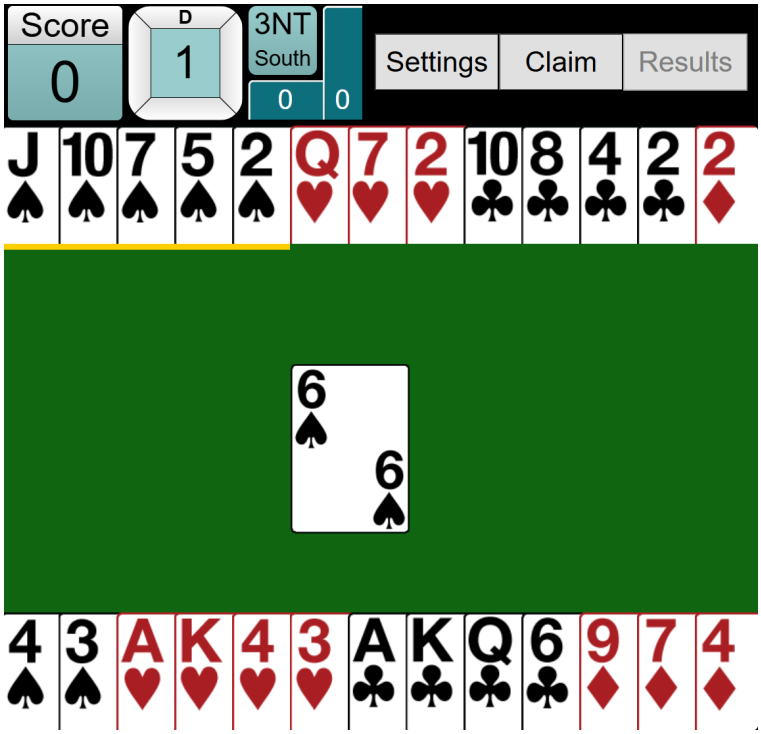

Ilustración 3.2: Interfaz de Bridge Base Online

Por otro lado, existen recursos más enfocados a la enseñanza de Bridge, como el **Sayc Bidding Practice (SAYC)** [1] que se centran en la práctica de de la subasta y las convenciones específicas. Este recurso si bien es muy útil para ayudar a la memorización de distintas aperturas, carece de una experiencia de juego completa, ya que no implementa la fase de carteo.

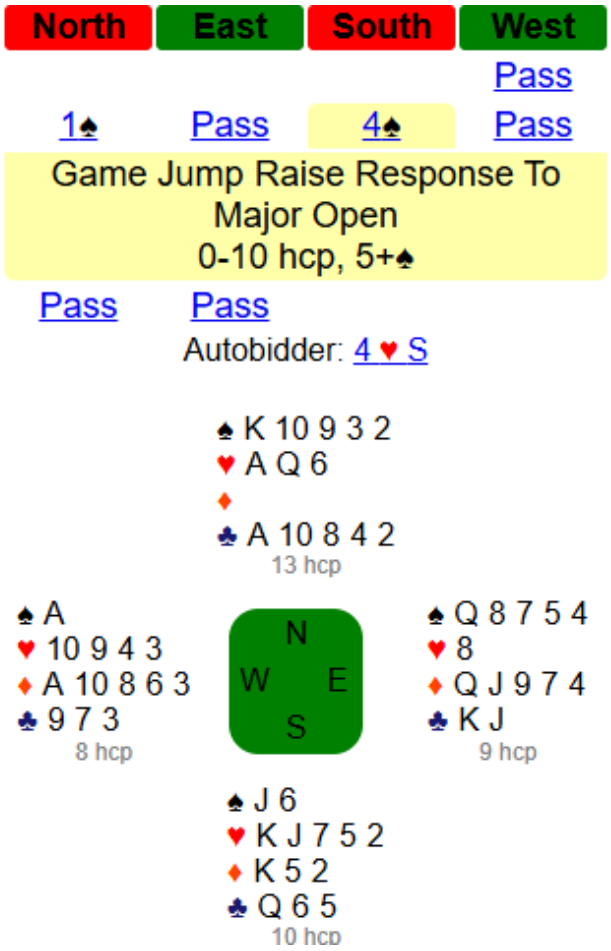

Ilustración 3.3: Interfaz de Sayc Bidding Practice

Este trabajo busca precisamente juntar las dos herramientas, proporcionar tanto el aprendizaje con elementos didácticos en tiempo real como una experiencia de juego completa, permitiendo al usuario jugar tanto la fase de subasta como la de carteo.

3.2. Objetivos

Los objetivos principales planteados para este trabajo, inicialmente previstos en el TFT01, son los siguientes:

- **Implementación de reglas y mecánicas oficiales del Bridge:** crear una implementación digital del juego de Bridge que replique las reglas y mecánicas oficiales, asegurando el cumplimiento de las normativas del juego.

- **Posibilidad de jugar contra la máquina:** implementar jugadores controlados por la máquina que se ajuste al nivel de un jugador principiante a través de uso de convenios y heurísticas básicas.

- **Práctica de las distintas fases del juego (subasta y carteo):** ofrecer la posibilidad de pedir pistas durante tanto la subasta como el carteo, recibiendo recomendaciones basadas en estrategias básicas, además de una explicación detallada del porqué de la recomendación.

- **Interfaz amigable para principiantes:** proveer una interfaz gráfica de usuario intuitiva y amigable, que facilite la comprensión de la dinámica del juego y la interacción con los distintos elementos visuales.

En cuanto a las desviaciones respecto a los objetivos iniciales, cabe mencionar que aun que la fase de puntuación (Scoring) forma parte de la mecánica del juego, no se llegó a implementar, priorizando la solidez de las mecánicas de subasta, carteo y asistencia didáctica.

Aparte de los objetivos planteados inicialmente en el TFT01, se han agregado algunas características adicionales como las siguientes:

- **Configuración de la partida:** permitir al jugador elegir el rango de punto de honor y la distribución de la mano, tanto del propio jugador como de su pareja.

- **Ver historial de juego:** permitir al jugador ver el historial del juego actual de su partida, este incluye las cartas que se jugaron, el estado de la subasta y el estado del carteo.

Capítulo 4

Aportaciones del trabajo

En este capítulo se presentan las contribuciones fundamentales de este trabajo. Se detallarán las principales aportaciones del proyecto al campo de las aplicaciones didácticas de Bridge, así como las competencias específicas que se han cubierto con este TFG. Finalmente, se indicará el alineamiento del TFG con los objetivos de desarrollo sostenible.

4.1. Principales aportaciones

Las principales aportaciones de este TFG residen en la implementación digital didáctica de Bridge completa y accesible dirigida a los jugadores principiantes. Se destacan las siguientes características del trabajo:

- **Herramienta didáctica integrada:** a diferencia de plataformas orientadas a jugadores experimentados como BBO [8], o aplicaciones de práctica de subasta aisladas como SAYC [1], el proyecto ofrece una herramienta didáctica que abarca la totalidad de una partida de Bridge (subasta y carteo).

- **Sistema de asistencia en tiempo real:** la característica más importante de la aplicación y valiosa es la implementación de un módulo de ayuda que ofrece sugerencias y consejos dependiendo del contexto de la partida con explicaciones detalladas. Esto permite al jugador entender el porqué de las decisiones estratégicas del Bridge.

- **Jugadores IA:** los jugadores controlados por la máquina no solo juegan cualquier

carta legal, sino que también toman decisiones coherentes siguiendo las sugerencias de la herramienta didáctica. Esto ofrece un grado de dificultad al juego, lo cual es fundamental para simular una partida real.

- **Flexibilidad y configuración de la partida:** permitir al jugador elegir el rango de punto de honor y la distribución de la mano, tanto del propio jugador como de su pareja, lo que permite al jugador practicar escenarios específicos, lo cual es crucial para el estudio y aprendizaje de convenciones de subasta o técnicas de carteo particulares.

- **Arquitectura modular:** el diseño del software bajo el patrón arquitectónico Model-View-ViewModel (MVVM) y el enfoque de Event Driven Development (EDD) han resultado en una arquitectura robusta, fácilmente extendible y de simple mantenimiento, lo que permite futuras ampliaciones como el sistema de puntuación.

4.2. Competencias específicas

A continuación se presentan las competencias específicas que se han cubierto con este TFG:

4.2.1. CI6

"Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos."[6]

Este proyecto ha requerido de la aplicación de algoritmos complejos para la satisfacción de restricciones al generar las cartas del jugador y su pareja, además de la lógica de la subasta y carteo. La adopción de la metodología Desarrollo guiado por pruebas (TDD) ha permitido diseñar y validar soluciones complejas, como la correcta gestión del estado de la partida y las interacciones entre jugadores.

4.2.2. CI7

"Conocimiento, diseño y utilización de forma eficiente de los tipos y estructuras de datos más adecuados a la resolución de un problema."[6]

La naturaleza de Bridge, con sus cartas, manos, palos y bazas ha requerido de la utilización de estructuras de datos específicas adaptadas a la resolución de problemas de juego. Esto incluye la representación de la baraja, las manos de los jugadores, el historial de bazas, entre otros.

4.2.3. CI8

"Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados."[6]

La aplicación ha requerido de un exhaustivo estudio de las reglas de Bridge, así como su traducción a un modelo software. Se ha seleccionado el lenguaje de programación C# y el motor de juego Unity por su flexibilidad y gran soporte para el desarrollo de juegos en 2D. Tanto el modelo (Model) como el vista-modelo (ViewModel) han sido creados desacoplados de Unity, lo que permite una fácil sustitución por otros motores de juegos si es necesario como Godot.

4.2.4. C15

"Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica."[6]

La implementación de los jugadores IA se ha basado en la utilización de heurísticas, que permiten a los jugadores realizar jugadas que de manera general suele estar acertadas, además de convenios estándares establecidos en la comunidad de Bridge.

4.2.5. C16

"Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería del software."[6]

Para el desarrollo del proyecto se ha seguido la metodología iterativa e incremental, permitiendo la implementación de nuevas funcionalidades y refinamiento de las existentes. Por otro lado, se han aplicado los principios SOLID (Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) para distintas partes del código, que se detallaran en el capítulo 6, Desarrollo.

4.3. Alineamiento con los objetivos de desarrollo sostenible

Cuadro 4.1: Grado de relación del TFT con los objetivos de desarrollo sostenible.

ODS	Grado de relación			
	0 No procede	1 Bajo	2 Medio	3 Alto
1 Fin de la Pobreza	X			
2 Hambre cero	X			
3 Salud y Bienestar	X			
4 Educación de calidad			X	
5 Igualdad de género	X			
6 Agua limpia y saneamiento	X			
7 Energía Asequible y no contaminante	X			
8 Trabajo decente y crecimiento económico	X			
9 Industria, Innovación e Infraestructuras	X			
10 Reducción de las desigualdades	X			
11 Ciudades y comunidades sostenibles	X			
12 Producción y consumo sostenibles	X			
13 Acción por el clima	X			
14 Vida submarina	X			
15 Vida de ecosistemas terrestres	X			
16 Paz, justicia e instituciones sólidas	X			
17 Alianzas para lograr objetivos	X			

4.3.1. ODS 4: Educación de calidad

Al ofrecer una herramienta didáctica para el aprendizaje de Bridge, el TFG contribuye al fomento del acceso a conocimientos y al desarrollo de habilidades cognitivas y lógicas (pensamiento estratégico, resolución de problemas, inferencia, memoria), alineándose con la meta de promover oportunidades de aprendizaje.

Capítulo 5

Análisis y Diseño

En este capítulo se detalla la fase de análisis y diseño del proyecto. Se verá la metodología de desarrollo, la selección de herramientas, pasando por el diseño de la arquitectura y la definición precisa de los casos de uso del sistema.

5.1. Metodología

La metodología aplicada a lo largo del trabajo fue el desarrollo iterativo-incremental, con un enfoque particular en la parte incremental del proceso. Este enfoque permitió desarrollar el sistema de manera progresiva, añadiendo funcionalidades (incremental) y mejorando la calidad de las funcionalidades existentes (iterativa).

Las razones principales para la elección de este enfoque sobre otros métodos como SCRUM, son las siguientes:

- Desarrollo en solitario: al ser un proyecto individual, no es necesario tener en cuenta las necesidades de otros miembros del equipo a través de reuniones periódicas

- Flexibilidad: el Bridge es un juego con una gran profundidad estratégica con numerosas convenciones. El enfoque incremental permitió implementar las reglas de juego de manera modular, una regla de juego puede ser implementada y probada (test) de manera independiente, mientras que el carácter iterativo permitió establecer un prototipo funcional básico desde la repartición de cartas hasta el carteo, garantizando la viabilidad

del proyecto completo. Estas funcionalidades fueron más tardes ampliadas y refinadas a lo largo del tiempo.

- La ausencia de un product manager eliminó la presión de presentar las iteraciones en fechas preestablecidas. Esta autonomía permitió que cada iteración fuera guiaba por objetivos internos y, especialmente, por la estabilidad de los incrementos implementados anteriormente, ya que es crucial que el juego digital se comporte de manera exacta al juego original. En lugar de forzar una implementación frágil para cumplir con plazos externos a cambio de la acumulación de deudas técnicas y robustez.

- Objetivo claro: el Bridge al ser un juego ya existente, solo era necesario digitalizarlo y añadir el carácter didáctico, por lo que se conocía con certeza desde un principio qué es lo que se esperaba del proyecto. Esto facilitó el establecimiento de las fases del desarrollo: primero el modelo del juego como las cartas, manos, jugadores, bazas. A continuación el controlador que gestionaría el flujo del juego y por último, la interfaz gráfica que permitiría al jugador humano interactuar con el juego.

5.2. Análisis y selección de herramientas

La primera fase del proyecto consistió en un análisis de las tecnologías y herramientas que se emplearían para el desarrollo del juego. Esta etapa fue fundamental para establecer un marco de trabajo robusto, eficiente y adecuado para un juego de cartas interactivo. La elección del motor de juego, en particular, determinaría el lenguaje de programación principal y la estructura del proyecto.

Se consideraron tres motores de juegos: Unity, Unreal Engine y Godot. La evaluación se basé en la facilidad de uso para el desarrollo de juegos en 2D, la curva de aprendizaje, la comunidad y los recursos disponibles en plataformas como YouTube, GitHub, etc.

Unity

Unity [11] es un motor de juegos reconocido por su flexibilidad y facilidad de uso para tanto el desarrollo de juegos en 2D como en 3D, Unity utiliza C# como lenguaje de programación principal. Este cuenta con una gran comunidad y una amplia variedad de recursos como tutoriales, documentación, assets, etc. Su curva de aprendizaje es accesible para desarrolladores familiarizados con lenguajes de programación parecidos como Java.

Unreal Engine

Ilustración 5.1: Unity

Ilustración 5.2: Unreal Engine

Unreal Engine [7] destaca por sus capacidades gráficas realistas y su uso intensivo en el desarrollo de juegos AAA, este motor de juegos utiliza C++ como lenguaje de programación principal. Su curva de aprendizaje es significativamente más pronunciada que los otros dos motores, el desarrollo en C++ implica una mayor complejidad en el desarrollo y depuración.

Godot

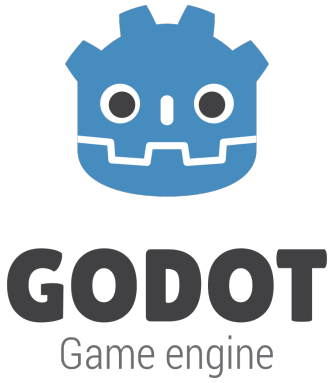

Ilustración 5.3: Godot

Godot [9] es un motor de juegos de código abierto, es conocido por su flexibilidad y una comunidad creciente, este motor de juegos utiliza GDScript como lenguaje de programación principal parecido a Python, también es compatible con C#. Al ser un motor de juegos desarrollado recientemente, la cantidad de recursos disponibles es limitada.

Tras un análisis de las tres opciones, se optó por elegir el motor de juego Unity. Esta decisión se fundamentó en factores como:

- **Curva de aprendizaje y facilidad de uso:** su interfaz intuitiva y la abundancia de recursos online permiten una desarrollo ágil y eficiente.

- **Experiencia previa en Informática Gráfica:** la familiaridad adquirida con el entorno de Unity y sus distintas funcionalidades en la asignatura optativa Informática Gráfica, proporcionó un punto de partida considerablemente más favorable que los otros dos motores.

- **Lenguaje de programación:** la similitud sintáctica con Java, lenguaje ampliamente utilizado a lo largo de la carrera, redujo considerablemente el esfuerzo de aprendizaje.

- **Mejor soporte para el desarrollo de juegos en 2D:** entre las tres opciones, Unity ofrece la mayor cantidad de funcionalidades para el desarrollo de juegos en 2D.

Adicionalmente, se consideró la posibilidad de desarrollar la aplicación utilizando entornos de programación de propósito general como JavaFX o Java Swing, pero esta opción fue descartada por la complejidad de la implementación de animaciones, objetos interactivos desde cero. Esto hizo inviable su elección para los plazos del trabajo.

5.3. Herramientas de desarrollo complementarias

Una vez seleccionado Unity [11], como motor de juego, la siguiente etapa consistía en la selección de las herramientas de desarrollo complementarias. Estas herramientas facilitarían la gestión del código fuente y la depuración. La elección de las herramientas fue basada en su integración con Unity y su amplia uso en la ingeniería del software.

Control de versiones: Git y GitHub

Para la gestión de versiones de código fuente, se optó por Git y GitHub. Git es el programa estándar de control de versiones en el mundo de la informática, su posibilidad de registrar cada modificación, revertir a versiones anteriores, y gestionar ramas de desarrollo facilitó la gestión de la historia del proyecto. GitHub es una plataforma para alojar repositorios de Git, proporcionando un respaldo seguro del código en la nube, además de posibilitar la difusión del instalador del programa de una manera centralizada en su apartado de Releases. Una alternativa a GitHub es GitLab, pero su uso es menos extendido a nivel global.

Editor de texto: Visual Studio Code

Para la edición de código C#, se seleccionó Visual Studio Code. Se eligió por su ligereza, alta personalización a través de extensiones y soporte oficial para C# y Unity. Estas extensiones permiten la refactorización y depuración del código, directamente desde Visual Studio Code.

Herramienta de Pruebas: Unity Test Runner y NUnit

Esta herramienta de pruebas está incluida en el editor de Unity, este framework permitió la ejecución de pruebas unitarias y de integración, para validar la lógica y robustez del juego.

5.4. Diseño de la arquitectura

Para garantizar una separación clara de responsabilidades, modularidad, testeabilidad y mantenibilidad del juego, se optó por la implementación del patron arquitectónico MVVM (Model-View-ViewModel).

El patrón MVVM es un patrón de diseño de software que se basa en la separación del modelo (Model), la vista (View) y la interacción del usuario (ViewModel). Este patrón permite una mayor flexibilidad y mantenibilidad del código.

Los componentes principales de esta arquitectura se definen de la siguiente manera:

- **Modelo (Model):** representa la lógica del juego. Es completamente independiente del motor de juego y de la interfaz gráfica. Contiene todas las reglas y mecánicas del Bridge (subasta, carteo, cálculo de baza, etc), así como el estado de la partida (historial de la subasta, cartas jugadas, turno, etc), y la lógica de los jugadores IA. Cada fase o concepto del juego está representado por una clase, por ejemplo, la clase `BiddingPhase` representa la fase de subasta, `PlayPhase` la fase de carteo, etc.

- **Vista (View):** representa la interfaz gráfica del juego. Contiene la lógica de la interfaz gráfica, como capturar los eventos de Clic del usuario, además de la representación gráfica de los distintos elementos del juego, como las cartas, los menús, los botones, etc. La vista obtiene los datos del ViewModel a través de enlace de datos (Data binding), técnica que establece una conexión automática entre la interfaz de usuario View y los datos del ViewModel, permitiendo la actualización automática de los elementos gráficos cuando los datos cambian.

- **Modelo-Vista (ViewModel):** capa de abstracción entre la vista y el modelo. Expone los datos del modelo a través de enlace de datos a la vista, y proporciona comandos a la vista para actualizar los datos del modelo. Por ejemplo, el vista-modelo contiene datos como las bazas ganadas, consejos del asistente, y estos datos son mostrados en la vista.

- **Coordinador:** es el componente que coordina la interacción entre los componentes, es el responsable de iniciar, avanzar y detener el juego. Este componente es el encargado de gestionar el estado del juego, como la posición del jugador, el turno, etc.

Flujo de interacción entre componentes

1. Usuario interactúa con la vista

2. La vista invoca un comando del modelo-vista

3. El modelo-vista delega la petición al coordinador

4. El coordinador actualiza el estado del juego llamando a los métodos del modelo

5. Los modelos procesan la lógica y actualizan su estado

6. El coordinador notifica al modelo-vista de los cambios a través de eventos

7. El modelo-vista actualiza sus datos

8. La vista actualiza los datos en la pantalla

5.5. Definición de los casos de uso

Tras elegir el motor de juego y plantear el diseño de la arquitectura, se procedió a la definición de los casos de uso del juego. Estos casos de uso representan las principales operaciones que el usuario puede realizar en el juego. En la Figura 5.4 se muestran los casos de uso del juego:

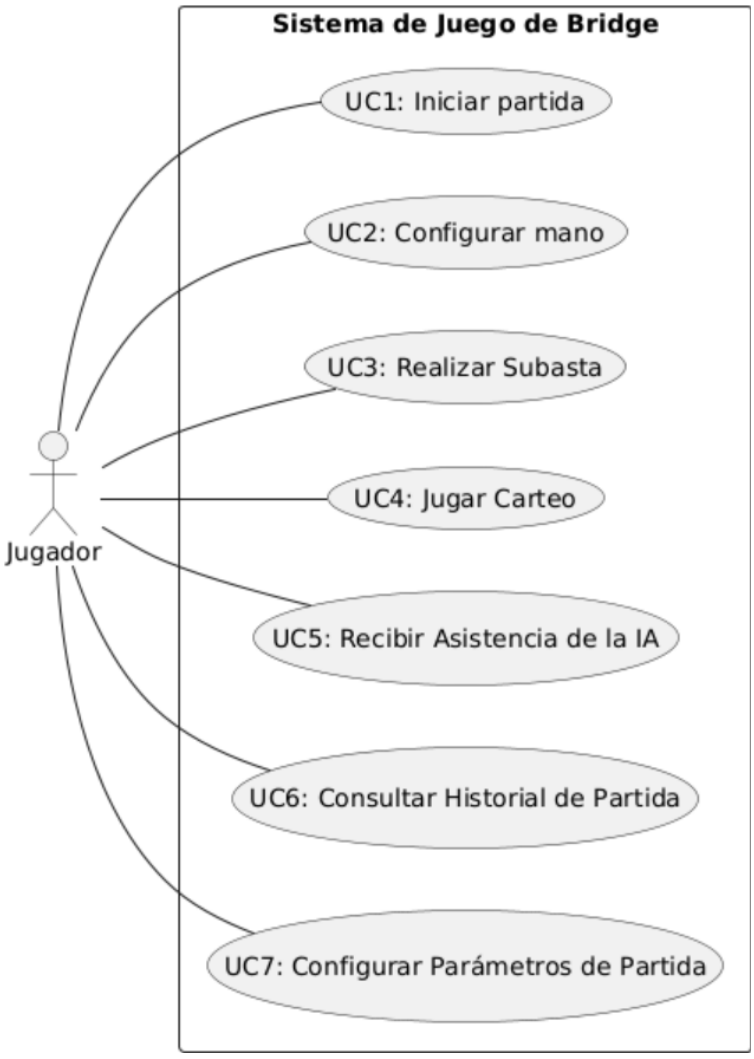

Ilustración 5.4: Casos de uso del juego

A continuación se detallan los casos de uso:

Cuadro 5.1: Caso de uso: Iniciar partida

ID Caso de Uso	UC1
Nombre	Iniciar partida
Descripción	Permite al jugador comenzar una nueva partida de Bridge
Actor	Jugador
Precondiciones	El jugador debe estar en el menú principal del juego.
Postcondiciones	El jugador es dirigido al menú de configuración de mano.
Flujo principal	1. El jugador inicia el juego 2. El jugador hace clic en el botón `Jugar`

Cuadro 5.2: Caso de uso: Configurar mano

ID Caso de Uso	UC2
Nombre	Configurar mano
Descripción	Permite al jugador configurar la mano que quiere jugar y la de su pareja.
Actor	Jugador
Precondiciones	El jugador debe estar en el menú de configuración de mano.
Postcondiciones	El jugador es dirigido a la pantalla de subasta.
Flujo principal	1. El jugador modifica los punto de honor y selecciona si quiere una mano equilibrada 2. El jugador hace clic en el botón `Aceptar`

Cuadro 5.3: Caso de uso: Realizar subasta

ID Caso de Uso	UC3
Nombre	Realizar subasta
Descripción	Permite al jugador subastar
Actor	Jugador
Precondiciones	El jugador debe estar en la pantalla de subasta.
Postcondiciones	La subasta es mostrada en la mesa y pasa turno al siguiente jugador
Flujo principal	1. El jugador selecciona una subasta 2. El jugador hace clic en el botón `Confirmar`

Cuadro 5.4: Caso de uso: Jugar carta

ID Caso de Uso	UC4
Nombre	Jugar carta
Descripción	Permite al jugador indicar la carta que quiere jugar
Actor	Jugador
Precondiciones	El jugador debe estar en la pantalla de carteo.
Postcondiciones	La carta elegida es eliminada de la mano del jugador, se muestra en la mesa y pasa turno al siguiente jugador
Flujo principal	1. El jugador hace clic en la carta que quiere jugar

Cuadro 5.5: Caso de uso: Carta de la IA

ID Caso de Uso	UC5
Nombre	Recibir Asistencia de la IA
Descripción	Permite al jugador pedir pista durante la partida sobre que jugada realizar.
Actor	Jugador
Precondiciones	El jugador debe estar en la subasta o en el carteo.
Postcondiciones	Se muestra en pantalla la pista.
Flujo principal	1. El jugador hace clic en el botón `Pista`

Cuadro 5.6: Caso de uso: Consultar Historial de Partida

ID Caso de Uso	UC6
Nombre	Consultar Historial de Partida
Descripción	Permite al jugador ver las subastas realizadas y las cartas que se han jugado a lo largo de la partida.
Actor	Jugador
Precondiciones	El jugador debe estar en la pantalla de subasta o carteo.
Postcondiciones	Se muestra en pantalla el historial de partida
Flujo principal	1. El jugador hace clic en el botón `Subasta` o `Carteo`

Capítulo 6

Desarrollo

En este capítulo se profundiza en el proceso detallado de la implementación del proyecto. Tras establecer las bases en el análisis y diseño (Capítulo 5), esta sección se enfoca en explicar cómo las decisiones teóricas se traduce en código funcional.

Para una comprensión más detallada, se presentará extractos de códigos de los elementos más relevantes del juego. Se detallarán los aspectos claves como los motores de subasta y carteo, la construcción de la interfaz gráfica, animaciones, la lógica de las bazas, etc.

6.1. Prototipo inicial

Antes de la fase de implementación definitiva, se realizó un pequeño prototipo inicial, este prototipo forma una parte integral del proceso de este TFG, ya que en él abarca una fase de exploración, familiarización tecnológica y prueba de conceptos.

6.1.1. Propósito del prototipo

Los objetivos principales del prototipo fueron:

- **Familiarización tecnológica**: adquirir conocimientos práctico básico de las herra-

mientas y tecnologías a emplear, en concreto el motor de juegos Unity y el lenguaje de programación C#. Esto incluye comprender el ciclo de vida de los scripts de Unity, la gestión de escenas, los objetos de juego, el uso del editor, el sistema de interfaz de usuario, etc.

- **Prueba de conceptos**: implementar una versión muy simplificada de las mecánicas esenciales de Bridge, como el reparto de cartas, la visualización de las cartas, realización de la subasta, etc. Esto permitiría evaluar el nivel de complejidad del juego, las posibles interacciones entre componentes, la gestión de turnos asíncronos, etc.

En esta versión simplificada del juego, el jugador podía realizar la subasta y el carteo. Los jugadores IA durante la fase de subasta únicamente pasaban de turno, y durante el carteo, jugaban cualquier carta legal.

6.1.2. Arquitectura inicial y desafíos técnicos

Durante esta fase de prototipado, el planteamiento inicial de la arquitectura fue subóptimo, ya que predominó un diseño de alto acoplamiento y bajo reparto de responsabilidades. Estas primeras versiones del código frecuentemente mezclaban la lógica de la interfaz de usuario, la lógica del juego y la gestión del estado del juego en un mismo script.

Un ejemplo de los problemas que surgieron fue el uso de un elemento central `GameManager` monolítico cuya función no era únicamente gestionar el estado de la partida, sino que también contenía responsabilidades relacionadas con la interfaz de usuario, lógica de las bazas, etc. Los principales problemas que surgieron fueron:

- **Dificultad de mantenimiento**: cualquier cambio en una parte del sistema podía tener consecuencias inesperadas en otras partes, haciendo difícil la depuración y mantenimiento del código.

- **Dificultad de escalabilidad**: al agregar nuevas funcionalidades al sistema, se debía modificar la unidad central que estaba acoplado al resto del sistema.

- **Baja testabilidad**: la mezcla de responsabilidades hacía prácticamente imposible realizar pruebas unitarias. Un test requería la ejecución de casi el juego completo.

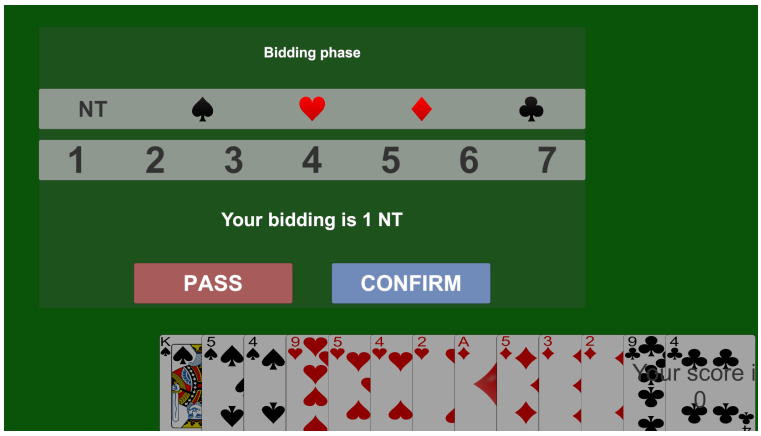

Ilustración 6.1: Prototipo con MVC, efecto colateral provocando el mal cálculo de las posiciones de las cartas

6.1.3. Lecciones aprendidas y familiarización tecnológicas

A pesar de los desafíos técnicos que surgieron durante esta fase, el resultado general del prototipo fue positivo por su valor en el aprendizaje:

- **Habilidad básica en Unity y C#**: esta fase permitió aprender la utilización del editor de Unity, la creación de objetos dinámicos, la gestión de coordenadas de los objetos, la creación de interfaz de usuario, manejo de eventos de UI.

- **Diseño de la interfaz de usuario**: se probaron distintas paletas de colores, tamaños de fuente, distribución de los elementos gráficos, etc. Esto permitió aprender a diseñar una interfaz de usuario que se adapte a las necesidades del juego.

- **Identificar las áreas más críticas del juego**: se identificaron las partes del juego que requerían más atención como el correcto funcionamiento de las bazas, la generación de mano, la gestión de turnos, etc.

6.1.4. Planificación para el desarrollo definitivo

La experiencia y los desafíos encontrados durante la fase del prototipo inicial, especialmente el alto acoplamiento y la dificultad para la testabilidad, fueron claves para un planteamiento estratégico para el desarrollo definitivo. La solución elegida se basó en una combinación de principios y patrones de diseños:

- **TDD**: el uso de TDD se convirtió en una prioridad fundamental para el desarrollo del juego. Este enfoque permitiría validar cada módulo del juego individualmente, y en caso de efectos secundarios, poder identificarlos lo antes posible.

- **Patrón arquitectónico MVVM**: el cambio de MVC a MVVM asegura que el Modelo y la Vista permanezcan desacoplados, con un ViewModel actuando de intermediario, transformando los datos del Modelo en un formato que pueda ser consumido por la Vista.

- **Desarrollo orientado a eventos EDD**: vía principal de comunicación entre los componentes, implementado mediante el Patrón observador. La idea principal es reemplazar la comunicación directa y acoplada entre componentes por un sistema de eventos. En lugar de un único controlador encargado de gestionar el estado del juego, este simplemente emitiría los eventos que se produjeran en el juego y cada uno de los componentes interesados en estos eventos se registrarían como observadores de dichos eventos.

Esta planificación sentó las bases para el desarrollo del juego, asegurando una buena separación de responsabilidades, flexibilidad e integridad del juego.

6.2. Desarrollo del juego

Como se mencionó anteriormente, la metodología de desarrollo elegida fue iterativa-incremental, sin embargo, cabe destacar que durante las fases iniciales el enfoque era casi en su totalidad incremental. Esto se debe a que módulos pequeños como las manos, las bazas, el cálculo de punto de honor, etc. no resultaban muy complejas, por lo que se optó por implementar cada uno de ellos en su totalidad antes de avanzar al siguiente módulo. En iteraciones posteriores, donde la interconexión de los distintos módulos crecía, el enfoque se volvió más iterativo que incremental, la prioridad era llegar a un producto mínimo viable y asegurar que cada una de las partes del juego se integraba de manera correcta.

6.2.1. TDD

El proceso de desarrollo se guió fundamentalmente por los principios de Desarrollo guiado por pruebas (Test Driven Development - TDD), por lo que se ha combinado tanto la implementación como la validación del proyecto en el mismo proceso.

El Desarrollo guiado por pruebas (TDD) es una técnica de desarrollo que se basa en la escritura de pruebas antes de la implementación de cualquier código funcional. Esta técnica es el pilar del desarrollo de este proyecto, ya que permitió asegurar la correcta implementación de las mecánicas y reglas de un juego tan complejo como el Bridge. El ciclo de desarrollo Rojo-Verde-Refactor (Red, Green, Refactor) se aplicó de la siguiente manera:

1. **Rojo (Fallo de prueba):** se escribe una prueba unitaria para una funcionalidad específica. Esta prueba fallaría, ya que el código que trata de probar no habría sido implementado.

2. **Verde (Prueba pasa):** se implementa el código mínimo necesario para que la prueba pase. Esto incluye la implementación de las clases y métodos necesarios para que la prueba se ejecute correctamente.

3. **Refactor (Mejora del código):** una vez el código pasa la prueba se mejora el código. Esto implica la eliminación de código innecesario, mejora de nombre de variables, eficiencia y legibilidad, sin alternar su comportamiento funcional.

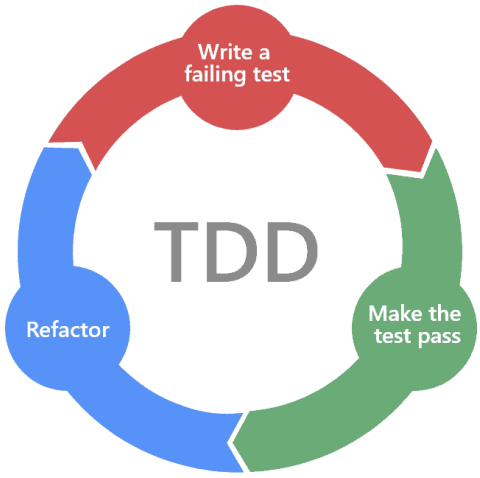

Ilustración 6.2: Ciclo iterativo de TDD

La aplicación de TDD fue especialmente útil para la implementación de la lógica de la subasta y carteo, ya que permitió identificar cualquier fallo en la aplicación de las reglas de juego, especialmente cuando se llegaba a un estado inconsistente como jugadas fuera de turno, carteo de mano vacío, jugadas ilegales, etc. Para ello las pruebas unitarias fueron diseñados para validar que el sistema lanzara las excepciones esperadas al detectar dichas inconsistencias, garantizando así la integridad del estado de la partida.

Un aspecto crucial para facilitar la aplicación de TDD y asegurar el máximo desacoplamiento durante las pruebas fue el diseño de los modelos utilizando interfaces. De este modo,

las clases se comunicarían entre sí mediante los métodos de las interfaces y no de una implementación concreta (Principio O de los principios SOLID), abierto a la extensión cerrado a la modificación. Lo que llevó al siguiente punto, la utilización de la librería de aserciones de Moq [5]. Con Moq, fue posible crear objetos mock que se comportan como si fueran objetos reales adaptadas a las necesidades de las pruebas. Esto garantizó que las pruebas unitarias realmente probasen el comportamiento de la funcionalidad sin verse afectada por factores externos.

6.2.2. Fase inicial: Modelo

Durante la fase inicial, se implementaron los modelos más fundamentales del juego, ya que estos encapsulan toda la lógica de negocio pura y ele estado de la partida, completamente independiente de la interfaz gráfica o del motor de juego. Cabe destacar que la fase de desarrollo del Modelo, con la aplicación rigurosa de TDD, abarcó aproximadamente un 70 % del tiempo total de desarrollo. A pesar de que esta sección pueda parecer la más corta en la memoria debido a su naturaleza no visual y la intención de no saturar la memoria de código. Entre los modelos más importantes se encuentran:

- **(Deck) Baraja:** clase que representa la baraja, contiene atributos como la lista de cartas `Cards` y métodos como `ShuffleCards` y `DealCard`
- **(IHand) Mano:** interfaz que representa la mano, contiene atributos como la lista de cartas `Cards`, bool `IsBalanced` que indica si la mano es equilibrada y métodos como `AddCard` y `RemoveCard`
- **(IPlayer) Jugador:** interfaz que representa a un jugador del juego, contiene atributos como la posición `Position` (Norte, Este, Sur, Oeste), mano `Hand`, métodos como `PlayCard` y eventos como `OnCardChosen` y `OnCallChosen`
- **(IBidding) Subasta:** interfaz que representa la fase de subasta, contiene atributos como el contrato final `Contract`, lista de jugadores `Players`, jugador actual `CurrentPlayer` y métodos como `MakeCall`
- **(IPlay) Carteo:** interfaz que representa la fase de carteo, contiene atributos como rondas ganadas por atacantes `RoundsWonByAttackers`, lista de jugadores `Players`, jugador actual `CurrentPlayer` y métodos como `PlayCard`

La implementación del Modelo se realizó siguiendo estrictamente TDD, como se detalló

previamente, cada una de las interfaces fue diseñada y construida a partir de pruebas unitarias. Esto garantizó que la lógica del juego fuese correcta y el código robusto sin efectos secundarios.

```
public void Play_TricksWonByAttackers_ShouldBeIncreased_WhenTrickWinnerIsAttacker() {
    Mock<IBidding> mockAuction = CreateMockAuction();
    mockAuction.Setup(a => a.Declarer).Returns(southPlayer); // Declarer and his partner are attackers (south and north)
    IPlay play = new PlayPhase(mockAuction.Object);
    play.PlayCard(new Card(Rank.Two, Suit.Clubs), westPlayer);
    play.PlayCard(new Card(Rank.King, Suit.Clubs), northPlayer);
    play.PlayCard(new Card(Rank.Queen, Suit.Clubs), eastPlayer);
    play.PlayCard(new Card(Rank.Jack, Suit.Clubs), southPlayer);
    Assert.AreEqual(1, play.TricksWonByAttackers);
}
```

Algoritmo 6.1: Ejemplo de test unitario, contador de rondas ganadas

En el código 6.1, se verifica el correcto incremento del contador de bazas ganadas por atacantes cuando este gana. En esta prueba se ve claramente el beneficio de utilizar `Moq` [5], permitiendo modificar el comportamiento de la interfaz `IBidding` (que representa la fase de subasta), en este caso, fijando el `Declarer` a un valor específico.

Esto elimina la necesidad de:

- Crear una instancia real y completamente funcional de la fase de subasta
- Ejecutar toda la lógica de la fase de subasta hasta que se decida un declarante

Esta capacidad de aislar cada uno de los módulos del juego contribuye a la velocidad, fiabilidad y granularidad de las pruebas unitarias.

6.2.3. Fase inicial: Coordinador (GameManager) e Inicializador (GameInitializer)

GameManager

El `GameManager` es el componente central del juego, pero a diferencia del prototipo, su rol es puramente de orquestación, sin lógica de negocio directa. Su función principal es gestionar el ciclo de vida de la partida, así como la transición entre las fases (subasta, carteo, fin de partida) y la interacción entre modelos.

Aquí es donde se aplica los principios de desarrollo orientado a eventos, en lugar de tener una referencia a cada objeto que está a la espera de una acción del juego, el `GameManager` emite eventos como `OnPhaseChanged`, `OnPlayMade` y `OnTrickEnd`. Estos eventos serán capturados por la Vista y ViewModel para ser procesados. Esto es un claro ejemplo del principio de Inversión de Dependencias (D) de los principios SOLID, donde tanto el `GameManager` como los componentes dependen de abstracciones (eventos/interfaces de observador), y no de implementaciones concretas.

```
public void ProcessPlay(Card card, IPlayer player) {
    _play.PlayCard(card, player);
    OnPlayMade.Raise(this, (card, player));
}
```

Algoritmo 6.2: Método ProcessCall de la clase GameManager

En el código 6.2, se muestra el método `ProcessPlay` del `GameManager`, este método delega la acción de jugar una carta a la instancia de `IPlay` y emite un evento cuando se realiza la jugada. De esta manera el `GameManager` no necesita saber nada sobre la implementación de la fase de juego, sino que simplemente delega la acción a la instancia de `IPlay` y notifica al `ViewModel` cuando se realiza la jugada, garantizando un bajo acoplamiento entre componentes.

GameInitializer

El `GameInitializer` es el punto de entrada de una partida, en él se define las implementaciones concretas de los componentes participantes. Este componente crea los jugadores, crea la baraja, reparte las cartas, crea la instancia de `GameManager` y comienza la partida.

```
void Awake() {
    List<IPlayer> Players = new List<IPlayer>(
        new ComputerPlayer(South),
        new ComputerPlayer(East),
        new ComputerPlayer(North),
        new ComputerPlayer(West)
    );

    Deck deck = new Deck();
    deck.Shuffle();

    foreach (IPlayer player in Players) {
        player.ReceiveCards(deck.DealCards(13));
    }
    IGameManager gameManager = new GameManager(player: Players, dealer: Players[0]);
    gameManager.StartGame();
}
```

Algoritmo 6.3: Método Awake simplificado de la clase GameInitializer

En el código 6.3, se muestra el método `Awake` simplificado del `GameInitializer`, en él

se elige las implementaciones concretas como `ComputerPlayer` y `GameManager`, reparte las cartas a cada jugador, inicializa el `GameManager` y comienza la partida.

En este punto, el modelo del juego está completamente operacional. Los jugadores IA están configurados para realizar cualquier jugada legal, lo que significa que se puede iniciar la partida y esta avanzará de forma autónoma por las fases de subasta y carteo.

A partir de este punto, el desarrollo entra en la fase de iteración. La naturaleza incremental de la implementación que fue dominante hasta ahora, pasa a un segundo plano para dar paso a mejoras y refinamientos continuos.

6.2.4. Fase intermedia: Modelo-Vista y Vista

Modelo-Vista

En esta fase, se implementa el componente Modelo-Vista, que hace de intermediario entre el Modelo y la Vista. Este tiene campos como int `WonTricksCount` que almacena la cantidad de rondas ganadas por el jugador, bool `ShowDummy` que indica si el muerto debe mostrar su mano o no, etc. Estos campos se derivan del estado de la partida y se actualizan en tiempo real. La vista se vincula a estos campos mediante `DataBinding`, permitiendo que el usuario vea la información actualizada en tiempo real.

```
public BindableProperty<int> WonTricksCount => BindableProperty<int>.Bind(() => {
    var declarer = _game.Declarer;
    var partnerOfDeclarer = PlayerUtils.PartnerOf(declarer, _game.Players.ToList());

    bool isPlayerAttacker = _humanPlayer == declarer || _humanPlayer == partnerOfDeclarer;
    return isPlayerAttacker ?
        _game.TricksWonByAttackers :
        _game.TricksWonByDefenders;
});
```

Algoritmo 6.4: Campo WonTricksCount de la clase GameViewModel

En el código 6.4, se muestra el campo `WonTricksCount` de la clase `GameViewModel`, este campo actualiza su valor en tiempo real con el valor que almacena el `IGameManager`.

Vista

La interfaz de de usuario de la aplicación se estructura en varias pantallas: Inicio, Configuración de Mano, Subasta y Carteo.

Inicio

Esta pantalla es la primera que se muestra al iniciar la aplicación, se muestra la pantalla de inicio con un botón `Jugar` y un botón `Salir`. Tiene un diseño minimalista, con el nombre del juego, los iconos de cada palo y un par de botones.

Ilustración 6.3: Pantalla de inicio básica

Ilustración 6.4: Pantalla de inicio final

En iteraciones posteriores, se implementó la posibilidad de ajustar la resolución de pantalla (Figura 6.4).

Configuración de Mano

Esta pantalla se muestra cuando el usuario elige "Jugar" en la pantalla de inicio. El jugador puede configurar los puntos de honor y la distribución de la mano que quiere jugar.

Junto con esta pantalla, se implementó la clase `HandGenerator`. Esta clase es la encargada de generar manos a partir de una baraja dado el rango de punto de honor y la distribución

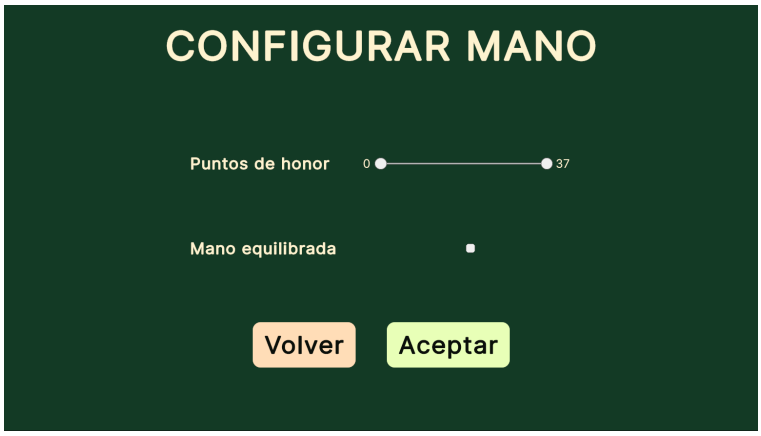

Ilustración 6.5: Pantalla de configuración de mano básica

de palos especificadas por el usuario. Esto da un control importante sobre la condición inicial de partida al jugador, permitiéndole practicar con configuraciones de mano específicas.

Aunque a primera vista generar una mano no parecía excesivamente complejo, esta parte del proyecto se convirtió rápidamente en uno de los retos más difíciles de solventar del proyecto. La generación de mano aleatoria en combinación de cumplir las restricciones establecidas escondía una serie de complejidades algorítmicas inesperadas que requirió de una cantidad considerable de tiempo y esfuerzo.

La principal dificultad que presentaba residía en la generación `aleatoria` en combinación con el `cumplimiento de restricciones`. Esto llevó a varios problemas:

- **Bucle infinito:** el algoritmo de vez en cuando caía en un bucle infinito. A medida que se seleccionaban cartas para la mano, se llegaba a un punto donde era imposible satisfacer el rango de punto de honor como la distribución de palos de manera simultánea. Determinar este "punto de no retorno" era extremadamente complejo, ya que implicaba un análisis predictivo (look-ahead) de las múltiples combinaciones posibles con las cartas restantes de la baraja.

- **Predicción:** no había una forma efectiva de predecir con antelación que se había llegado a un estado sin solución. El algoritmo seguía intentando combinaciones que no llevaría a una solución.

- **Imposibilidad de unit test:** debido a la naturaleza pseudo-aleatoria de la generación de mano, no había una forma de probar la funcionalidad, ya que su solución no era determinista. Para la misma baraja, la misma distribución de palos, y el mismo número de puntos de honor, se obtendría una mano diferente.

Dada la naturaleza del problema, se consideró la utilización de herramientas especializadas en programación con restricciones [14]. Específicamente, Minizinc [10], una herramienta de programación declarativa vista a lo largo de la carrera de Ingeniería Informática. Este tipo de lenguajes especialmente diseñados para resolver problemas de combinatoria encajaba perfectamente con el desafío presente de generación de mano con restricciones.

Sin embargo, la integración de Minizinc con Unity no era posible, no existe ninguna librería que permita el uso de Minizinc en C#. Por lo tanto, se decidió implementar el algoritmo de generación de manera inteligente.

La solución final fue una combinación de algoritmos inteligentes y una estrategia de selección de cartas por fases dentro del propio `HandGenerator`.

1. **Elección estratégica de cartas iniciales:** el objetivo es no restringir demasiado la mano con las primeras cartas, ya que la elección de cartas posteriores dependerá de ellas. Para ello, se priorizó alcanzar el mínimo de punto de honor establecido, para posteriormente llenar la mano hasta trece cartas.

2. **Prioridad a cartas de honor:** para cumplir una de la restricciones lo más pronto posible, se priorizó la selección de cartas de honor (J=1, Q=2, K=3, A=4).

3. **Relleno de la mano:** una vez alcanzado el mínimo de punto de honor, se rellena la mano hasta trece cartas, asegurando que la mano cumpla con las restricciones establecidas.

```
public static IHand Generate(Deck deck, bool balancedHand, int minHCP, int maxHCP) {
    while (true) {
        IHand hand = new Hand();
        List<Card> remainingCards = new(deck.Cards);
        List<Card> currentAttemptCards = new();
        List<Card> honorCards = remainingCards.Where(c => c.HCP > 0).ToList();
        List<Card> nonHonorCards = remainingCards.Where(c => c.HCP == 0).ToList();
        honorCards.Shuffle();
        nonHonorCards.Shuffle();
        int currentHCP = 0;
        foreach (var card in honorCards) {
            if (currentHCP + card.HCP <= maxHCP &&
                currentAttemptCards.Count(c => c.Suit == card.Suit) < targetSuitLengths[card.Suit] &&
                currentAttemptCards.Count < 13) {
                currentHCP += card.HCP;
                currentAttemptCards.Add(card);
            }
            if (currentHCP >= minHCP && currentAttemptCards.Count >= 7) // Early exit
                break;
        }
        List<Card> allRemainingSorted = new();
        allRemainingSorted.AddRange(honorCards.Except(currentAttemptCards)); // Add remaining honors
        allRemainingSorted.AddRange(nonHonorCards);
```

```
        allRemainingSorted.Shuffle();

        foreach (var card in allRemainingSorted) {
            if (currentAttemptCards.Count == 13) break; // Hand is full
            if (currentAttemptCards.Count(c => c.Suit == card.Suit) < targetSuitLengths[card.Suit]) {
                if (currentHCP + card.HCP <= maxHCP || currentAttemptCards.Count < 13) {
                    currentAttemptCards.Add(card);
                    currentHCP += card.HCP;
                }
            }
        }

        if (currentAttemptCards.Count == 13) {
            hand.AddCards(currentAttemptCards);
            if (hand.HCP >= minHCP && hand.HCP <= maxHCP && hand.IsBalanced) {
                Debug.Log("Found:␣" + iter);
                return hand;
            }
        }
    }
}
```

Algoritmo 6.5: Método Generate de la clase `HandGenerator`

Esta combinación de heurísticas y pasos secuenciales permitió obtener una mano que cumpliera con las restricciones establecidas, haciendo que `HandGenerator` fuera robusta y funcional.

Ilustración 6.6: Pantalla de configuración de mano final

En iteraciones posteriores, se implementó la posibilidad de configurar la mano del compañero (Figura 6.6).

Pantalla de Subasta

En esta pantalla se muestra la mano del jugador, el historial de juego, y la caja de subastas. El jugador puede hacer clic en el los cartuchos de la caja de subasta para indicar su elección

y hacer clic en el botón `Confirmar` para confirmar la elección.

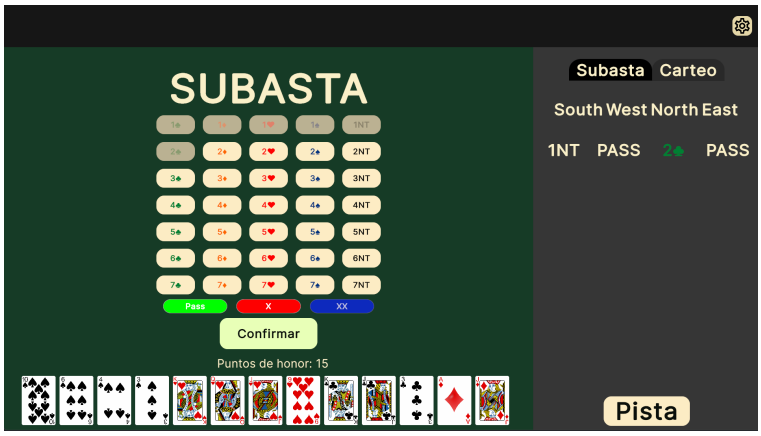

Ilustración 6.7: Pantalla de subasta

Para optimizar el proceso de desarrollo y evitar la creación manual y repetitiva de elementos en la interfaz, tanto los cartuchos de la caja de la subasta como la mano del jugador se generaron dinámicamente desde código.

Ilustración 6.8: Template de la pantalla de subasta

```
private VisualElement CreateCardView(Card card) {
    VisualElement cardView = new();
    cardView.style.backgroundImage = LoadImage(card);
    cardView.style.marginLeft = 5;
    cardView.style.marginRight = 5;
    return cardView;
}

private StyleBackground LoadImage(Card card) {
    string rank;
    if ((int)card.Rank > 10)
        rank = card.Rank.ToString();
    else
        rank = ((int)card.Rank).ToString();
    string textureName = rank.ToLower() + "_" + card.Suit.ToString().ToLower();
    Texture2D cardTexture = Resources.Load<Texture2D>("Cards/" + textureName);
    return new StyleBackground(cardTexture);
}
```

Algoritmo 6.6: Código de generación de mano dinámicamente

En el código 6.6, se muestra el método `CreateCardView` que se encarga de crear la vista de las cartas del jugador, este carga la imagen correspondiente de la carta y la aplica a la vista.

Pantalla de Carteo

En esta pantalla se muestra la mano del jugador, el historial de juego, y la mano del muerto. El jugador puede hacer clic en cualquier carta legal para jugar la carta.

Ilustración 6.9: Pantalla de juego

Tanto en la pantalla de subasta como en la de juego, el usuario puede ver el historial de juego a la derecha de la pantalla. El diseño de esta pantalla está basado en la interfaz de usuario de Chess.com (Ver Figura 6.10), donde el usuario puede ver información relevante sobre la partida.

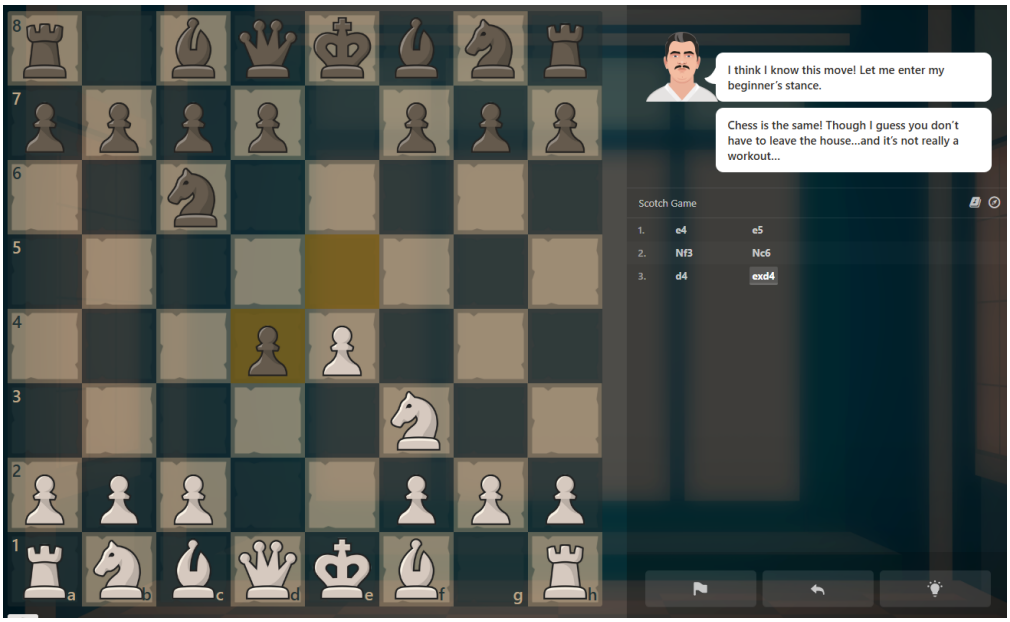

Ilustración 6.10: Interfaz de usuario de Chess.com

Animación

Para elevar la experiencia visual y fluidez de la aplicación, las animaciones de las cartas se gestionaron utilizando la librería de tweening [15], `DOTween` [4]. Esta herramienta facilitó la creación de animaciones de movimiento y de transición de manera sencilla y eficiente.

El `tweening` es una técnica de animación digital que genera los fotogramas entre dos "keyframes" (posición inicial y final). Por ejemplo, para realizar la animación de la carta al jugarse se define un lugar inicial y final para cada jugador, y `DOTween` se encarga de interpolar el movimiento. Resultando en un movimiento fluido y natural.

```
private void AnimateCardToPlayArea(Card card, IPlayer player, Action animationCompleteCallback) {
    var (targetContainer, originContainer, _) = _playerCardSlotSpawnMap[player];

    // Create card object at origin
    GameObject cardObject = Instantiate(cardPrefab, originContainer.transform.position, Quaternion.identity);
    _playedCards.Add(cardObject);

    // Initialize card UI
    CardUI cardUI = cardObject.GetComponent<CardUI>();
    cardUI.Initialize(card);

    _animationsInProgress++;
    // Animate card to target position
    cardObject.transform.DOMove(targetContainer.transform.position, 1.0f)
        .SetEase(Ease.OutQuad)
        .OnComplete(() => {
            animationCompleteCallback?.Invoke();
            _animationsInProgress--;
            UpdateHandsUI();
```

```
        })
        .SetDelay(0.5f);
}
```

Algoritmo 6.7: Código de animación de la carta al jugarse

El uso de `DOTween` fue clave para conseguir una aplicación visualmente más atractiva, dinámica y fluida. Además de mejorar estéticamente el juego, también contribuye a una experiencia de usuario más amigable y agradable.

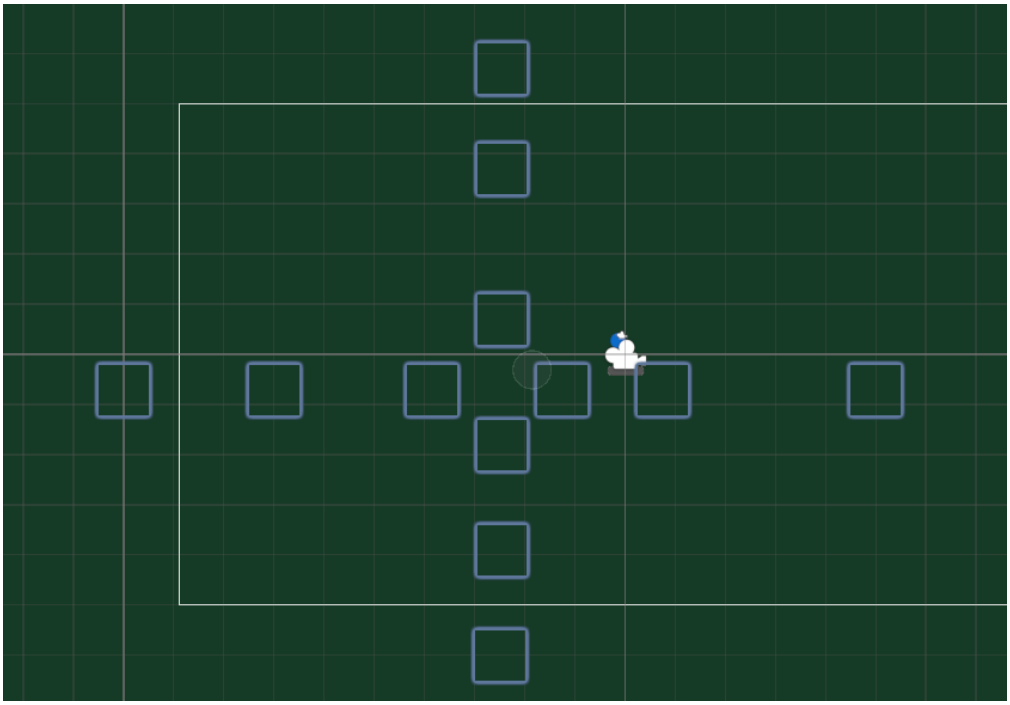

Ilustración 6.11: Escena de juego con lugar inicial y final de las cartas

En la escena de juego (Figura 6.11), se muestra el lugar inicial y final de las cartas. Los cuadrados más centrados representan el lugar final el la carta (mesa), los cuadrados intermedios representan el lugar inicial de la carta (mano), y los cuadrados más alejados representan el lugar donde se dirigen las cartas cuando un jugador gana la baza.

6.2.5. Fase final: Sistema de asistencia didáctica

Como última fase del proyecto, se implementa el sistema de asistencia didáctica, que permite al usuario solicitar pistas durante la subasta y el carteo.

Durante el desarrollo del sistema de asistencia didáctica se presentó un desafío de diseño

particular. Inicialmente, se consideró implementar la lógica de decisión mediante el patrón estado (State Pattern), ya que la naturaleza secuencial y transiciones predecibles de la subasta, hizo que la implementación de una máquina de estados pareciera una opción idónea. Sin embargo, este enfoque tras un par de iteraciones resultó en nuevamente un problema de escalabilidad. Cada subasta podía resultar en más de cinco distintas respuestas, y cada nueva capa de profundidad multiplicaba exponencialmente el número de estados posibles, este enfoque se volvió insostenible rápidamente. Este problema llevó a buscar un patrón alternativo más flexible: el Patrón estrategia (Strategy Patten).

Este enfoque nuevo con el Patrón estrategia permitió evitar la explosión de estados. En su lugar, se definieron distintas estrategias que podían ser seleccionadas en tiempo de ejecución dependiendo del contexto de la partida. Un componente central como la clase `BiddingEngine`, gestionaría una colección de estrategias disponibles. Cada estrategia implementaría una interfaz común `IBiddingStrategy` que incluiría un método `IsApplicable(Context)`. Este método permitiría al `BiddingEngine` determinar si una estrategia es aplicable en un determinado contexto.

```
class BiddingEngine : IBiddingEngine {
    List<IBiddingStrategy> _strategies;
    BiddingEngine(List<IBiddingStrategy> strategies) => _strategies = strategies;

    List<BiddingSuggestion> GetBiddingSuggestions(context) {
        var suggestions = new List<BiddingSuggestion>();
        foreach (strategy in _strategies) {
            if (strategy.IsApplicable(context))
                suggestions.AddRange(strategy.GetSuggestions(context));
        }
        return suggestions;
    }
}
```

Algoritmo 6.8: Código simplificado del BiddingEngine

El BiddingEngine recibe como parámetro una lista de IBiddingStrategy, que son almacenados internamente, y en el método GetBiddingSuggestions, se recorre la lista de IBiddingStrategy y se aplica cada uno de ellos pasándole como parámetro el contexto de la partida actual, devolviendo una lista de sugerencias de subasta.

A continuación se muestra un ejemplo de una estrategia de subasta:

```
class OpenerStrategy : IBiddingStrategy {
    bool IsApplicable(BiddingContext context) {
        if (context.Calls.Count == 0) return true;
        if (context.Calls.Where(c => c.Type == CallType.Bid).Count() == 0) return true;
        return false;
    }

    List<BiddingSuggestion> GetSuggestions(context) {
        List<BiddingSuggestion> suggestions = new();
```

```
        if (context.Hand.HCP < 12)
            // Can't open with less than 12 HCP
            suggestions.Add(
                new(message: OpeningPass(context.Hand.HCP),
                call: new Pass(null)));
        else if (context.Hand.HCP >= 15 && context.Hand.HCP <= 17 && context.Hand.IsBalanced)
            // 15-17 HCP and balanced hand, can open with 1NT
            suggestions.Add(
                new(message: OpeningOneNT(context.Hand.HCP),
                call: new BidCall(new Bid(1, Strain.NoTrump), null)));
        else
            suggestions.Add(
            new(message: Unknown,
            call: new Pass(null)));
        return suggestions;
    }
}
```

Algoritmo 6.9: Código simplificado de la estrategia de subasta del abridor

En este caso, la estrategia implementa dos métodos: IsApplicable y GetSuggestions. IsApplicable se encarga de determinar si la estrategia es aplicable en un determinado contexto, en este caso, se verifica que se encuentra en la apertura (no hay subastas previas).

GetSuggestions se encarga de devolver una lista de sugerencias de subasta, en este caso, se verifica si la mano tiene menos de 12 puntos de honor, si no, se devuelve una sugerencia de pasar el turno. Si la mano tiene de 15 puntos de honor a 17 puntos de honor y es equilibrada, se devuelve una sugerencia de abrir con 1NT. En caso contrario, se devuelve una sugerencia de pasar el turno. Esta es la apertura básica del convenio SAYC.

Por último, se muestra un ejemplo de lo flexible y escalable que es utilizar el patrón estrategia en la aplicación.

```
// Create bidding strategies list
List<IBiddingStrategy> biddingStrategies = new() {
    new BiddingStrategy.OpenerStrategy(),
    new NoInterventionStrategy(),
    new OneNTResponseStrategy(),
    new OneNTOpenerResponseStrategy(),
    new TransferStrategy(),
    new TransferResponseStrategy(),
    new TransferOpenerResponseStrategy(),
    new StaymanResponseStrategy(),
    new StaymanOpenerResponseStrategy(),
    new StaymanSecondResponseStrategy(),
};

// Create bidding engine
IBiddingEngine biddingEngine = new BiddingEngine(biddingStrategies);
```

Algoritmo 6.10: Fragmento de código del inicializador de partida

Para incrementar los convenios de subasta en el sistema de asistencia didáctica, solo se necesita agregar nuevas estrategias en la lista de estrategias. Las sugerencias de carteo están implementadas de la misma manera.

En el estado actual del juego, los jugadores IA utilizan todas las estrategias de los sistemas de asistencia didáctica, algunas estrategias de carteo implementadas son:

- **Honor sobre honor (convenio):** si el jugador es el segundo en jugar en la baza y el primero ha jugado una carta de honor, entonces:

  - Si el jugador tiene una carta mayor, jugarla

  - Si el jugador tiene no tiene una carta mayor, jugar la carta de menor nivel

  Esta estrategia tiene como objetivo obligar al contrincante a jugar otra carta de honor incluso mayor, efectivamente intercambiando dos cartas de honor por una.

- **Apertura secuencia de honor (convenio):** si el jugador es el abridor y tiene una secuencia de cartas de honor (AKQ, KQJ, QJ10), jugar la carta de mayor nivel. Esta estrategia tiene como objetivo arrastrar a los oponentes, obligándoles a jugar cartas del mismo palo, eliminando su posibilidad de fallar.

- **Apertura cuarto del palo quinto (convenio):** si el jugador es el abridor, tiene palo quinto (5+ cartas en un palo) con carta de honor, jugar la cuarta mayor carta de dicho palo. Esta estrategia tiene como objetivo, comunicar al compañero que se posee al tres cartas de mayor nivel en la mano y se posee una carta de honor.

- **Descarta la carta menor (heurística):** si el jugar no tiene carta del mismo palo que el palo de salida, se descarta la carta menor. Esta estrategia tiene como objetivo, evitar que el jugador "malgaste" cartas de valor durante una baza perdedora.

- **Carta menor en baza ganadora (heurística):** si el jugador es el último de la baza y su compañero ha jugado la carta ganadora de la baza, se juega la carta menor. Esta estrategia tiene como objetivo, evitar que el jugador "malgaste" cartas de valor en una baza ganadora asegurada.

Con estas estrategias simples de carteo, hace que el juego sea significativamente más divertido y desafiante, en comparación con el jugador IA que juega de forma aleatoria.

Capítulo 7

Resultados

En esta sección se presentan los resultados del trabajo, mostrando cómo la aplicación final cumple con los objetivos planteados en el TFT01, además de incluir algunas funcionalidades adicionales.

7.1. Cumplimiento de objetivos

La aplicación desarrollada es un juego de Bridge en solitario completamente funcional, diseñado como una herramienta didáctica de aprendizaje y práctica. El juego implementa las fases de subasta y carteo, permitiendo al usuario solicitar pistas durante ambas fases.

Los objetivos planteados en el TFT01 son estos:

- **Práctica de las distintas fases del juego (subasta y carteo)**
- **Interfaz amigable para principiantes**
- **Implementación de reglas y mecánicas oficiales del Bridge**
- **Posibilidad de jugar contra la máquina**

Los objetivos 'Práctica de las distintas fases del juego (subasta y carteo)' e 'Interfaz amigable para principiantes' pueden ser comprobados visualmente en las figuras 7.1 y 7.3, respectivamente. En estas figuras se puede observar las pistas que el usuario puede solicitar para facilitar su aprendizaje. Además, el diseño tiene un contraste de color claro y legible.

Complementando la funcionalidad y el diseño, el uso de la librería `DOTween` [4], como se detalló en la sección 6, permite mejorar la experiencia de usuario, además de hacer que el juego se sienta más amigable y agradable.

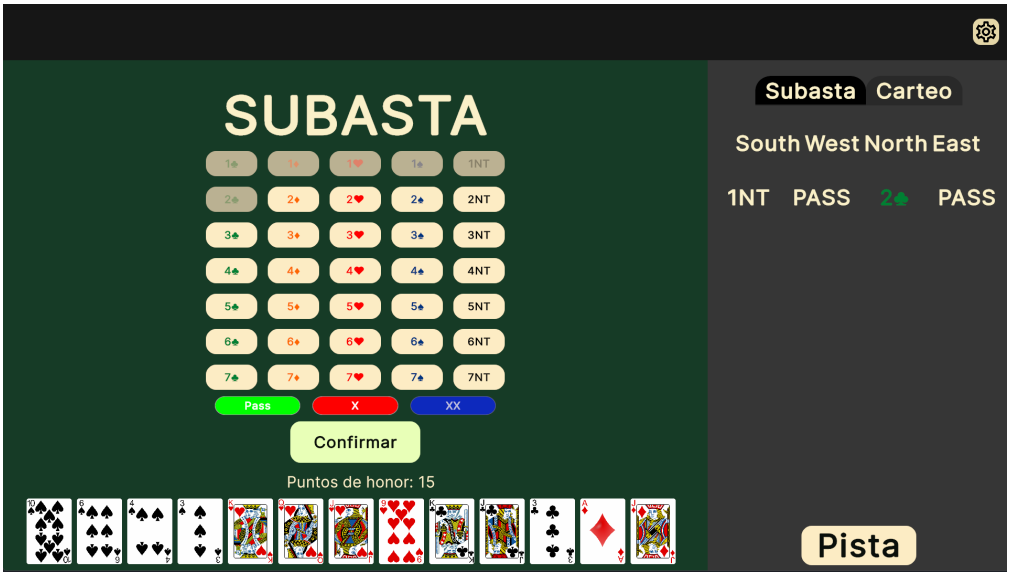

Ilustración 7.1: Pantalla de subasta

En la pantalla de la subasta, el usuario puede subastar o pasar el turno.

El usuario puede pedir pista durante la subasta, y se muestra en pantalla la pista.

En la pantalla de carteo, el usuario puede jugar y también solicitar pistas.

Para comprobar que los objetivos 'Implementación de reglas y mecánicas oficiales del Bridge' y 'Posibilidad de jugar contra la máquina' se cumplen, es necesario ver el código fuente o ejecutar el juego, ya que en una imagen no se puede ver la lógica del juego, aunque en las figuras 7.1 y 7.3 se puede observar a la derecha, el historial de la subasta y el historial del carteo, implicando el juego de la IA. Tanto el código fuente, como el instalador de la aplicación se encuentran disponibles en el repositorio de GitHub [16].

Ilustración 7.2: Pantalla de pista de subasta

Ilustración 7.3: Pantalla de carteo

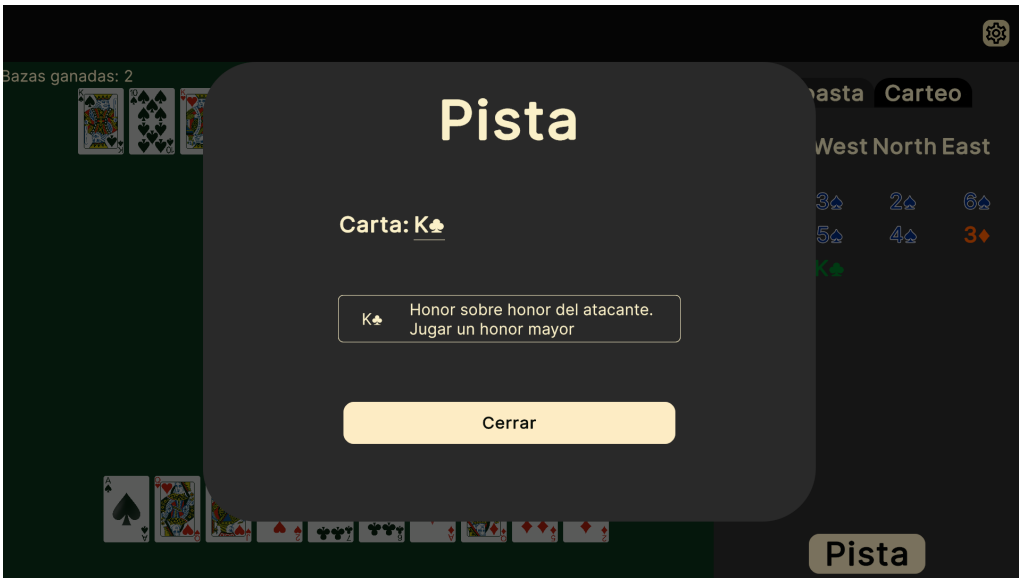

Ilustración 7.4: Pantalla de pista de carteo

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

En este proyecto se ha logrado desarrollar un programa didáctico para el juego de Bridge, que no solo cumple con los objetivos planteados en el TFT01, sino que también sienta una base sólida y extensible para futuras mejoras.

Se ha logrado establecer un modelo de juego completamente funcional, que permite al usuario experimentar tanto la fase de subasta como la de carteo con la posibilidad de solicitar pistas durante cualquier momento del juego, ofreciendo un soporte crucial para la comprensión de las reglas y estrategias. Además, la posibilidad de configurar la mano del compañero y ver el historial de la subasta y el carteo, lleva el juego a un nivel de experiencia de usuario más alto de lo que se esperaba inicialmente.

La interfaz de usuario ha sido un punto clave en la experiencia de usuario, mediante unas paletas de colores de alto contraste y animaciones fluidas se ha conseguido una interfaz amigable y agradable, que no solo mejora la estética y usabilidad del programa, sino que también hace que el juego sea accesible y atractivo incluso para quienes no tienen experiencia previa en Bridge.

En conclusión, con este trabajo no solo se provee una herramienta didáctica para el aprendizaje y práctica de Bridge, sino que también se espera incentivar a nuevos jugadores a explorar y unirse a la comunidad de Bridge, contribuyendo en la difusión del juego.

8.2. Trabajo futuro

Aunque el proyecto actual ha logrado cumplir todos los objetivos planteados, representa solo una fracción del vasto mundo del Bridge. Por lo tanto, existen diversas áreas de mejora que se pueden integrar al juego existente, elevando el programa a un nivel de experiencia de usuario, usabilidad y utilidad que no se ha alcanzado hasta el momento.

Las principales áreas de mejora se centran en la expansión de estrategia/inteligencia de los jugadores y el asistente, la diversificación de modos de juego y agregar funcionalidades claves para completar el juego.

- **Más estrategias de subasta:** se puede agregar más estrategias de subasta, como por ejemplo, las diferentes variantes del Forcing, convención de Smolen, apertura de palos menores, apertura de palos mayores, etc.

- **Más estrategias de carteo:** se puede agregar más estrategias de carteo, como por ejemplo, comunicación de las manos, bloqueo/desbloqueo, ceder/pasar, adversario peligroso, timing, etc.

- **Intervenciones durante la subasta:** hasta ahora se ha implementado estrategias de apertura en donde los oponentes siempre pasan, pero en un juego real, los oponentes pueden intervenir y cambiar el curso del juego drásticamente

- **Sistema de vulnerabilidades:** agregar vulnerables/no vulnerable a las parejas, incrementando el factor riesgo/recompensa de la subasta.

- **Sistema de puntuación:** se debe implementar la fase final del Bridge, donde se calculan los puntos correspondientes a cada pareja.

- **Diferentes dificultades:** ofrecer la posibilidad de elegir el nivel de dificultad de los oponentes

- **Modo de juego Puzzle:** implementación de un modo de juego "puzzle" que permita al usuario enfrentarse a situaciones específicas donde deben resolver problemas de subasta o carteo

- **Juego online:** agregar la posibilidad de jugar en línea con amigos y desconocidos, transformando el juego actual no únicamente como una herramienta didáctica, sino

que también en una plataforma social y competitiva.

- **Sistema de historial de partidas:** implementar un sistema de historial de partidas, permitiendo al usuario ver sus mejores partidas

- **Sistema de análisis:** implementar un sistema de análisis de juego, donde se explique los errores y las mejores jugadas.

- **Mejoras en la interfaz de usuario:** mejorar la estética con más animaciones, opciones de personalización, mejor visualización de información, etc.

- **Sistema de audio:** agregar efectos de sonido, como sonidos de cartas, sonidos de movimiento, sonidos de error, etc.

- **Integración de bases de datos de partidas famosas/historicas:** permitir a los jugadores comparar sus mejores partidas con las mejores jugadas de competiciones internacionales.

- **Elección de variantes de Bridge:** permitir al jugador elegir entre varias variantes de Bridge

- **Soporte multi-idioma:** permitir a los jugadores elegir entre idiomas diferentes

Estas mejoras tienen potencial de transformar el proyecto en una herramienta mucho más completa, versátil y atractiva para los usuarios que podría incluso llegar al nivel de plataformas como **Bridge Base Online (BBO)** [**8**].

Capítulo 9

Manual de Usuario

Este capítulo contiene la guía práctica para el usuario final. Se detallarán los pasos para instalar y utilizar la aplicación, a través de instrucciones de uso y capturas de pantalla.

9.1. Instalación y uso

9.1.1. Instalación

Para instalar la aplicación, se debe descargar el instalador desde el repositorio de GitHub [16] en el apartado de `Releases`, véase la Figura 9.1. Una vez descargado, el instalador se ejecutará y se le pedirá permiso para instalar la aplicación en el dispositivo.

Dentro del apartado de `Releases`, se encuentra la versión más reciente de la aplicación. Para descargar el instalador, se debe hacer clic en el botón `Assets` y luego en el archivo del instalador, véase la Figura 9.2.

Una vez descargado, el instalador se ejecutará y se le pedirá permiso para instalar la aplicación en el dispositivo.

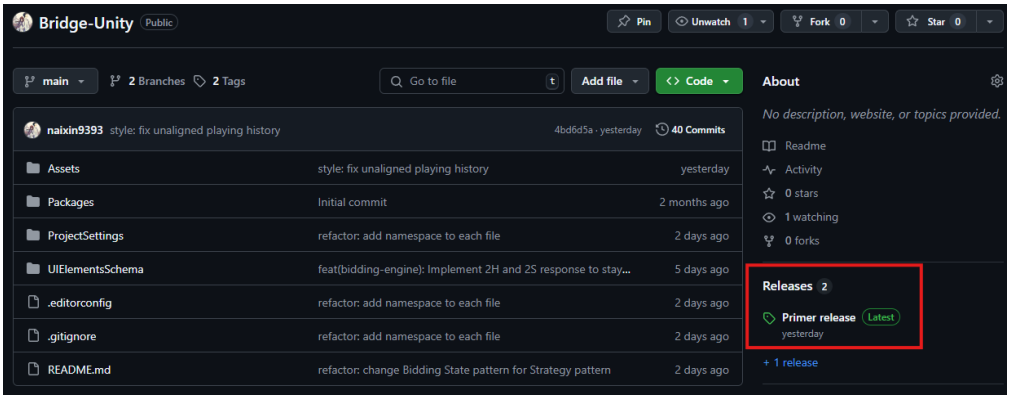

Ilustración 9.1: Apartado de Releases

Ilustración 9.2: Instalador

9.1.2. Uso

Al abrir la aplicación, se mostrará la pantalla de inicio, véase la Figura 9.3. En esta pantalla, hay tres botones: `Jugar`, `Resolución` y `Salir`.

Pantalla de inicio

Ilustración 9.3: Pantalla de inicio

- **Jugar**: se abre la pantalla de configuración de mano, véase la Figura 9.5.
- **Resolución**: se abre un menú para cambiar la resolución de la pantalla, véase la Figura 9.4.
- **Salir**: se cierra la aplicación.

Menú de resolución

Ilustración 9.4: Menú de resolución

En el menú de resolución, se puede cambiar la resolución de la pantalla haciendo clic sobre la opción deseada, véase la Figura 9.4.

Pantalla de configuración de mano

Ilustración 9.5: Pantalla de configuración de mano

En la pantalla de configuración de mano, véase la Figura 9.5, se puede configurar los punto de honor y la mano que se quiere jugar y la de su pareja.

- **Volver**: se vuelve a la pantalla de inicio, véase la Figura 9.3.
- **Aceptar**: se guarda la configuración y se dirige a la pantalla de subasta, véase la Figura 9.6.

Pantalla de subasta

En la pantalla de subasta, véase la Figura 9.6, hay varias funcionalidades: subastar, pasar turno, solicitar pista, ver historial de la subasta.

Para subastar y pasar turno, el usuario debe hacer clic sobre el botón correspondiente y a continuación hacer clic en el botón `Confirmar`.

Para solicitar pista, el usuario debe hacer clic sobre el botón `Pista`, se mostrará la pista en pantalla, véase la Figura 9.7.

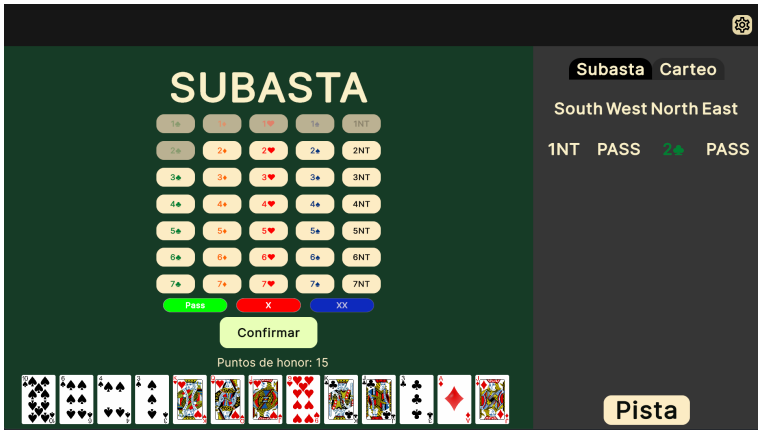

Ilustración 9.6: Pantalla de subasta

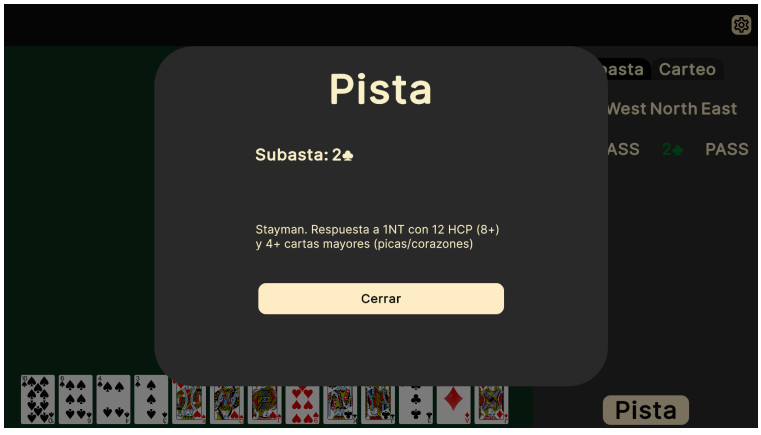

Ilustración 9.7: Pista de subasta

Adicionalmente, el usuario puede hacer clic sobre cualquiera las subasta en el historial de la subasta, véase la Figura 9.8, para ver la explicación de dicha subasta.

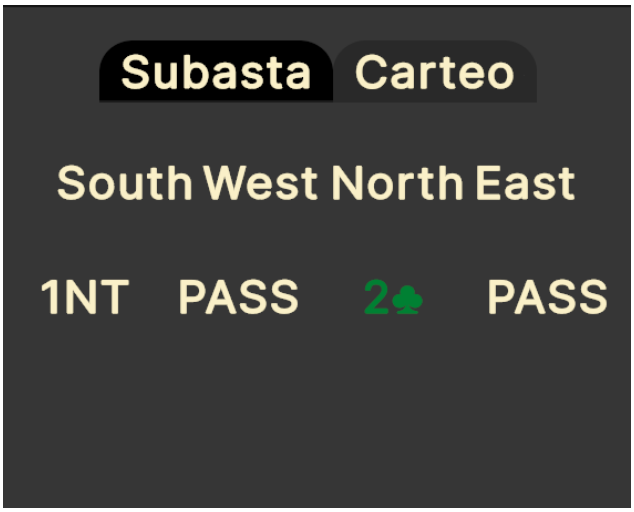

Ilustración 9.8: Historial de la subasta

Tras tres pases seguidos, se dirige a la pantalla de carteo, véase la Figura 9.9.

Pantalla de carteo

Ilustración 9.9: Pantalla de carteo

En la pantalla de carteo, el usuario puede jugar y también solicitar pistas al igual que en la pantalla de subasta.

Para jugar, el usuario debe hacer clic sobre una de las cartas legales (cartas no sombreadas). Una vez hecho clic, se pasará turno al siguiente jugador.

Para solicitar pista, el usuario debe hacer clic sobre el botón `Pista`, se mostrará la pista en pantalla, véase la Figura 9.10.

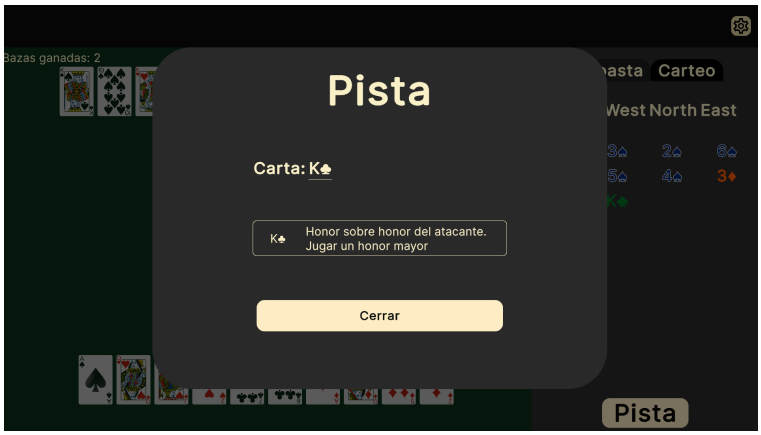

Ilustración 9.10: Pista de carteo

Al igual que en la pantalla de subasta, el usuario puede hacer clic sobre cualquiera de las cartas en el historial de la subasta, véase la Figura 9.11, para ver la explicación de dicha jugada.

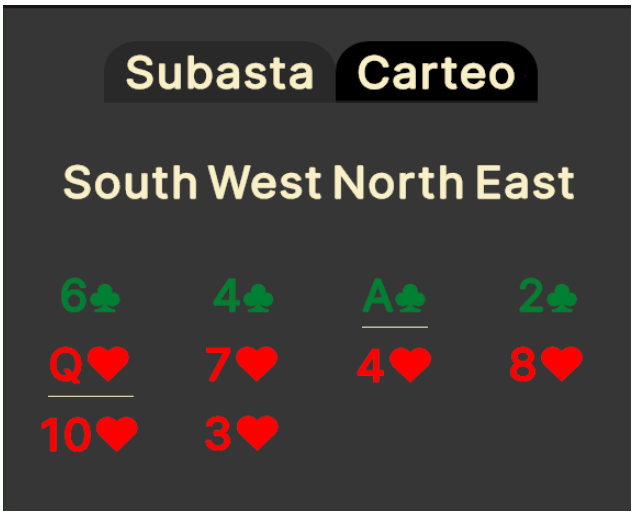

Ilustración 9.11: Historial de la carta

Tras finalizar las trece bazas, se muestra el menú de resultados, véase la Figura 9.12.

En el menú de resultados, se pueden ver los detalles de la partida. En este menú hay un botón para volver a la pantalla de inicio.

Ilustración 9.12: Menú de resultados

Bibliografía

[1] (2013). Sayc bidding practice. http://www.saycbridge.com/. Accedido: 2025-05-31.

[2] ACBL (1937). American contract bridge league. https://www.acbl.org/. Accedido: 2025-06-04.

[3] AEB (1950). Asociación española de bridge. https://www.aebridge.com/. Accedido: 2025-06-04.

[4] Demigiant (2025). Dotween. https://dotween.demigiant.com/. Accedido: 2025-06-04.

[5] devlooped (2009). Moq. https://github.com/devlooped/moq. Accedido: 2025-05-31.

[6] EII (2022). Competencias específicas. https://www.eii.ulpgc.es/sites/default/files/2022-05/Grado%20en%20Ingenier%C3%ADa%20Inform%C3%A1tica%20-%20memoria%20del%20plan%20de%20estudios%20-%202019.pdf. Accedido: 2025-05-31.

[7] Games, E. (1998). Unreal engine. https://www.unrealengine.com/. Accedido: 2025-05-31.

[8] Gitelman, F. (2001). Bridge base online. https://www.bridgebase.com/. Accedido: 2025-05-31.

[9] Juan Linietsky, A. M. (2014). Godot. https://godotengine.org/. Accedido: 2025-05-31.

[10] MiniZinc (2007). Minizinc. https://www.minizinc.org/. Accedido: 2025-06-04.

[11] Technologies, U. (2005). Unity. https://unity.com/. Accedido: 2025-05-31.

[12] WBF (1958). Federación mundial de bridge. https://www.worldbridge.org/. Accedido: 2025-06-04.

[13] Wikipedia (1925). Bridge de contrato. https://www.wikiwand.com/en/articles/History_of_contract_bridge. Accedido: 2025-06-04.

[14] Wikipedia (2025a). Constraint programming. https://en.wikipedia.org/wiki/Constraint_programming. Accedido: 2025-06-04.

[15] Wikipedia (2025b). Tweening. https://en.wikipedia.org/wiki/Inbetweening. Accedido: 2025-06-04.

[16] Zhou, N. C. (2025). Repositorio de github de bridge. https://github.com/naixin9393/Bridge-Unity.

Glosario

Abridor el jugador que primero juega durante la fase de carteo, es el que se sienta a la izquierda del declarante. 7

Arrastrar técnica de carteo, consiste en extraer todas o la mayoría de las cartas de un palo de los oponentes. 47

Atacante representa uno de los jugadores de la pareja que gana la subasta. 6, 8, 65, 66

Baza representa las 4 cartas que se juega en una ronda en la fase de carteo. En una baza cada jugador debe jugar una carta. 3, 6–8, 15, 23, 28–31, 44, 47, 61, 66, 68

Contrato representa la apuesta más alta de la fase de carteo. 3, 6, 8, 33, 65, 68

Declarante representa el primero de los atacantes que realiza la apuesta con el mismo rango que el contrato final. 8, 65

Declarar acción durante la subasta de poner una apuesta representada por un nivel y un rango, esta debe ser superior a todas las apuestas anteriores. Las apuestas de menor nivel son 1 tréboles, 1 diamantes, 1 corazones, 1 picas, 1 sin triunfo, las más altas son 7 tréboles, 7 diamantes, 7 corazones, 7 picas, 7 sin triunfo. 7

Desarrollo guiado por pruebas metodología de desarrollo que se basa en la escritura de pruebas antes de la implementación del código. 14, 31, 32

Desarrollo orientado a eventos paradigma de arquitectura en el que los componentes se comunican y reaccionan a través de eventos utilizando el patrón de diseño Observer.

31, 35

Doblar acción durante la subasta de incrementar los puntos que se obtiene en caso de ganar y los que se pierde en caso de perder. 7, 67

Dubletón representa un palo en una mano con 2 cartas. 7

Enlace de datos técnica que establece una conexión automática entre la interfaz de usuario View y los datos del ViewModel. 23

Fallar técnica de carteo, consiste en jugar una carta de un palo diferente al palo de salida para ganar bazas bajo contratos con triunfo. 47

Fallo representa un palo en una mano vacío, es decir, no tiene cartas. 6

Mano representa las 13 cartas aleatorias que un jugador recibe al inicio de la partida. 6, 7, 12, 14, 15, 31, 33, 37, 58, 66, 67

Mock técnica de simulación de objetos para la prueba unitaria. 33

Muerto representa el jugador que muestra sus cartas tras la primera carta jugada. 8, 36, 42

Nivel representa el número de bazas por encima de 6 que los atacantes deben ganar. 7, 47, 65

Palo de salida es el palo de la primera carta que se juega en una baza. 7, 8, 47, 66

Pasar acción durante la subasta de pasar el turno durante la subasta. 8

Patrón Estado patrón de diseño que permite definir una clase de objeto que mantiene el estado de un objeto, y permite cambiar el estado del objeto a través de métodos. 45

Patrón Estrategia patrón de diseño que permite definir una familia de algoritmos que pueden ser elegido en tiempo de ejecución dependiendo del contexto. 45, 46

Patrón Observador patrón de diseño que permite a los objetos observadores observar y recibir notificaciones de los eventos que ocurren en los objetos de sus interés. 31

Posición representa la posición del jugador en la mesa, es decir, Norte, Sur, Este, Oeste. 33

Programación declarativa paradigma de programación declarativa que se utiliza para resolver problemas combinatorios complejos. 39

Punto de honor puntuación asignada a las cartas de honor (J=1, Q=2, K=3, A=4) en cada palo. 6, 12, 14, 26, 31, 37–39, 46, 58

Rango representa el palo de triunfo o sin triunfo que se jugará en la fase de carteo. 7, 8, 65

Redoblar acción durante la subasta de responder a un doblar que incrementa aún más los puntos de ganar y perder. 7

Semifallo representa un palo en una mano con una sola carta. 7

Sin triunfo modalidad de juego en la que ningún palo tiene preferencia sobre otros. 67

SOLID conjunto de cinco principios de diseño de software

1. **S** (Single Responsibility Principle): un componente debe tener una única responsabilidad
2. **O** (Open/Closed Principle): un componente deben ser abiertos a la extensión y cerrados a la modificación
3. **L** (Liskov Substitution Principle): las clases deben ser sustituibles por sus subclases
4. **I** (Interface Segregation Principle): las clases no deben ser forzados a implementar interfaces que no utilicen
5. **D** (Dependency Inversion Principle): los módulos de alto nivel no deben depender de los módulos de bajo nivel

. 33, 35

Subasta fase de la partida de Bridge en la que los jugadores deciden el contrato final. 3

Tweening técnica de animación, consisten en la generación de fotogramas entre dos "key-frames" (posición inicial y final) para realizar una animación. 43

Whist juego de cartas de bazas popular en Inglaterra en el siglo XVI. 3