# Software Engineering 2 - Final Project Submission


# 1. Introduction

## Our Story: The Boys Are Jack In Town

When we first came together as a group at Universidad Complutense de Madrid, we weren't just looking to create a Blackjack game, we wanted to make something that felt alive, personal, and straight-up *fun*. We kicked things off with a brainstorming session over card games like *Spit*, *Trash*, and *Blackjack* itself, diving into what made them exciting (or not). Very quickly, Blackjack 2.0 was born, a bold take on the classic, packed with features we actually wanted to play ourselves.

But this wasn't just about slapping some code together. We were dreaming up wildcards, AI dealers, custom themes, betting systems, achievements, leaderboards, multiplayer madness, and even calling image generation APIs to bring wildcards to life! If we were gonna build something, we wanted it to be Blackjack but better. Our goal was to build a Blackjack game that not only follows core rules, but also pushes innovation in gameplay mechanics, UX, and engagement. We aimed for a modular, extensible product with the same spirit that made card games timeless, but reimagined for a new generation.

So that's how we created BlackJack 2.0 which is a modern, feature-rich Blackjack game developed for the Software Engineering 2 course. The project aims to provide a clean user interface, modular architecture, and enhanced gameplay mechanics, going beyond traditional Blackjack rules. Key features include multiplayer support, betting mechanics, achievements, wild card variants, sound effects, and localization.

The project evolved iteratively over multiple sprints. We began with core game logic and terminal-based interaction, gradually integrating GUI components, betting systems, achievements, multiplayer functionality, and polished visuals and audio. The project journey reflects an ongoing refinement of Scrum practices, technical design, and teamwork.

# 2. Scrum

## 2.1. Structure and Operation of the Scrum Team

- Team Members:
  - Hanna Szalai - Scrum Master
  - Kate O'Reilly - Product Owner
  - Finn Farrell - Git Expert
  - Haroun Riahi - Team Advocate
  - Emre Eryilmaz - Document Lead
  - David Jimenez - Developer
- Coordination Methods:
  - Tools: GitHub for version control, issue tracking, project board; daily meetings (online/in-person hybrid)
  - Agile Inception Document: Defined team roles, goals, risks, and coordination methods. Updated regularly to reflect project evolution.
  - Communication: Daily stand-ups, sprint planning, reviews, and retrospectives.
- Evolution:
  - Initial struggles with Git and Java by some members were resolved through tutorials and pairing sessions.
  - The hybrid working style allowed flexibility, with in-person meetings for key discussions and online coordination via GitHub.

## 2.2. User Stories

| ID | Name | Sprint # | Weight | Priority | Done by |
|---|---|---|---|---|---|
| 1 | As a UI/UX designer, I want to research user needs and expectations, so that I can design an interface that is intuitive and user-friendly. | 1 | S | P0 | Haroun |
| 2 | As a developer, I want to be able to remove cards from the player's hand. | 1 | L | P1 | Kate, Hanna, Finn |
| 3 | As a user, I want to be able to quit the game, aka terminate | 1 | S | P1 | Kate |
| 4 | I want to make the dealer draw cards according to the game rules so that the game follows standard Blackjack behavior | 1 | M | P1 | Hanna |

| | | | | | |
|---|---|---|---|---|---|
| 5 | I want to allow the player to start a new round after a game ends so that they can continue playing. | 1 | S | P0 | Kate |
| 6 | As a developer, I want to research about the best possible UI for the game | 1 | L | P0 | Haroun |
| 7 | As a dealer, I want to start a new game so that I can deal initial cards to the player and myself | 1 | S | P1 | Kate |
| 8 | I want to compare the player's and dealer’s hands so that a winner is determined. | 1 | L | P1 | Finn |
| 9 | As a user, I want to play with the game (hit, stand) in the terminal | 1 | L | P2 | Hanna |
| 10 | As a player or dealer, I want to draw a random card (or multiple random cards) from the deck, so that I can play the game fairly with randomized card distribution. | 1 | M | P1 | Haroun |
| 11 | As a developer, I want to delimit the first prototype, so that I can establish a clear scope and ensure a structured initial implementation of the game. | 1 | M | P1 | Haroun, Hanna, Kate, Finn |
| 12 | As a dealer, I want to deal a card to a player so that they can build their hand | 1 | S | P1 | Hanna |
| 13 | As a player, I want to see my current hand so that I can know which cards I have | 1 | M | P1 | Hanna |
| 14 | As a player, I want to calculate my score so that I can know my current score | 1 | M | P1 | Hanna |
| 15 | As a player, I want to receive a card so that I can build my hand | 1 | M | P1 | Hanna |
| 16 | As a developer, I want to set up the project structure, so that the game has a solid foundation for future development. | 1 | L | P0 | Kate |
| 17 | As a user, I want a glossary for the first prototype so that I can easily understand key terms and concepts used in the prototype. | 1 | S | P1 | Finn |
| 18 | As a developer I want to create and store the typical 52 deck of cards, so the user can have all the available cards. | 1 | M | P1 | Kate, Finn |

| | | | | | |
|---|---|---|---|---|---|
| 19 | As a developer I want to break down class structures for first prototype from user stories | 1 | S | P1 | Kate |
| 20 | As a developer I want to print the cards stored in the players hand in an appropriate format | 1 | M | P1 | Kate |
| 21 | As a researcher, I researched different card options that we can add to the existing game in the next sprint. | 1 | S | P0 | Dave |
| 22 | As a developer, I want to prevent the player from choosing multiple actions at once ( multiple choices in a row) | 1 | S | P1 | Hanna |
| 23 | I want to immediately determine the winner if either the player or dealer gets a Blackjack (21 with two cards) so that the game ends correctly. | 1 | S | P2 | Haroun |
| 24 | As a player, I want to see a clear message when I win or lose so that I understand the outcome of the game. | 1 | M | P1 | Finn |
| 25 | As a developer, I want to refactor the project into the MVC pattern by the end of the sprint. | 2 | M | P1 | Hanna |
| 26 | As a researcher, I want to explore innovative Blackjack mechanics, so that I can suggest new rule variations to enhance replayability. | 2 | S | P0 | Dave |
| 27 | As a player, I want correct messages to display when wildcards are drawn | 2 | M | P2 | Kate |
| 28 | As a researcher, I want to study different regional variations of Blackjack, so that I can identify unique rule sets to incorporate into our game. | 2 | S | P0 | Dave |
| 29 | As a player, I want to be able to draw a Blackjack Bomb card, so that I can instantly win the game upon receiving it. | 2 | L | P1 | Kate |
| 30 | As a researcher, I want to explore different rules for the game. | 2 | L | P0 | Dave, Emre |
| 31 | As a player I want to see a menu page with options when I start the game. | 2 | M | P1 | Hanna |
| 32 | As a player, I want the Joker Wild card to act as any value between 1 and 11, so that I can strategically adjust my hand’s score. | 2 | M | P1 | Kate |

| | | | | | |
|---|---|---|---|---|---|
| 33 | As a player, I want to be able to automatically split my hand when I draw a Split Ace card, so that I can play two separate hands without manually choosing to split. | 2 | L | P1 | Kate |
| 34 | I want to reveal the dealer’s hidden card at the appropriate moment so that the game follows standard Blackjack rules. | 2 | M | P1 | Finn |
| 35 | As a developer, I want to improve code comments and documentation so that the project is easier to maintain. | 2 | M | P2 | Finn |
| 36 | As a player, I want to have a DEMO to interact with a graphical interface instead of just the terminal so that the game is more engaging. | 2 | M | P1 | Kate, Finn |
| 37 | As a developer, I want to implement that the user can restart the game after each round. | 2 | M | P2 | Kate |
| 38 | As a developer, I want to integrate a Singleton Design Pattern in my project. | 3 | XS | P0 | Hanna |
| 39 | As a player, I want to place bets so that I can gamble within the game | 3 | M | P1 | Hanna, Haroun |
| 40 | As a developer I want to see the dealer's balance and bets each round. | 3 | M | P2 | Hanna |
| 41 | As a developer, I want to create a database for the achievement information. | 3 | M | P2 | Hanna |
| 42 | As a developer I want to implement different languages to make the game for accessible for all | 3 | L | P0 | Finn |
| 43 | As a player I want to pause the game when I need a break or get interrupted. | 4 | M | P0 | Haroun |
| 44 | As a player I want to see my achievements from the game. | 4 | M | P1 | Hanna. Haroun |
| 45 | As a user, I want to be able to play multiplayer at the same computer | 4 | M | P1 | Kate |
| 46 | As a developer, I don't want the special card text to overwrite the game text | 4 | M | P0 | Finn |
| 47 | As a developer, I want to ensure that the game correctly resets bets after each round and ends | 4 | M | P1 | Haroun |

| | | | | | |
|---|---|---|---|---|---|
| | the game when a player or dealer runs out of money | | | | |
| 48 | As a player I want to have settings to pause the game, exit the game and stop the music. | 4 | M | P1 | Emre |
| 49 | As a player, I would like to see nicer backgrounds maybe hear some songs | 4 | S | P2 | Hanna, Haroun |
| 50 | As a user, I want the gui to be bigger and easier to use. | 4 | XS | P2 | Finn |
| 51 | As a user, I want the betting system to be in a different place so it is easier to use. | 4 | S | P2 | Finn |
| 52 | As a developer, I want to save / Load with new Main. | 5 | L | P0 | Finn, Dave |
| 53 | As a player, I want to save/load the game. | 5 | L | P0 | Finn, Dave |
| 54 | As a player, I want to hear sound effects during the game. | 5 | S | P1 | Hanna |
| 55 | As a player, I want to be able to choose the difficulty setting of my AI dealer | 5 | M | P0 | Haroun |
| 56 | As a developer, I want to create a new UI window for the achievements. | 5 | M | P1 | Hanna, Haroun |
| 57 | As developers, we want to organise everything that will have to be done by the end of the term for our final submission. | 6 | XL | P0 | Hanna, Haroun, Kate, Dave, Finn, Emre |
| 58 | As a developer, I want to create UML diagrams for our project's documentation. | 6 | M | P1 | Finn |
| 59 | As a developer, I want to create Sequence Diagrams based on my main user stories. | 6 | M | P2 | Hanna |
| 60 | As a player, I want badges to visually update from grey to colorful when unlocked, so that I can feel rewarded for achievements | 6 | S | P1 | Hanna |
| 61 | As a developer I want to fix even more bugs. | 6 | L | P2 | Kate, Hanna, Finn, Haroun |

| | | | | | |
|---|---|---|---|---|---|
| 62 | As a user, I want to be able to play multiplayer with friends using Java sockets. | 6 | L | P0 | Kate |
| 63 | As a developer, I want to refactor the code to adhere MVC principles for the final submission. | 6 | XL | P2 | Kate, Hanna, Finn |
| 64 | As a developer I want to refactor the GameState class to adhere to MVC principles | 6 | S | P1 | Kate |
| 65 | As the scrum master, I want to create and continuously modify the main README.md and USERMANUAL.md files. | 6 | M | P1 | Hanna |
| 66 | As a player, I want to undo my last betting or gameplay action, so that I can recover from mistakes and improve strategy. | 6 | L | P2 | Hanna |
| 67 | As the Scrum Master, I want to create a LICENSE for the project. | 6 | S | P2 | Hanna |
| 68 | As a player I want to be able to undo my last hit or stand move using the Command Design Pattern | 6 | L | P0 | Hanna |
| 69 | As a developer, I want to make it so that the gui bug in the menu is resolved | 6 | S | P2 | Kate |
| 70 | As a developer I want to fix the language bugs. | 6 | M | P1 | Finn |
| 71 | As a developer, I want to add Javadocs to every class. | 6 | L | P2 | Hanna, Kate |

# 2.3. Sprint Plannings

Throughout the project, we used sprint planning meetings to organize our tasks, estimate the effort needed, and set clear goals.
After each sprint review and retrospective, we tried to apply what we learned to make the next sprint planning better.

### *Sprint 1 Planning (2025/02/03 – 2025/02/17):*

At the beginning, our main goal was to build the basic structure of the Blackjack game in the terminal.
We planned to:

- Create the card deck.
- Allow basic player actions (hit, stand).
- Calculate and compare hand scores.
- Set up a clean project structure and a glossary.

Since it was our first sprint, the planning was simple and not very detailed. We underestimated how much time Git setup and team coordination would take.

## *Sprint 2 Planning (2025/02/18 – 2025/03/04):*

After Sprint 1's retrospective, we realised we needed better task division and more clear code comments.
For Sprint 2, we planned to:

- Start adding a graphical interface (GUI demo).
- Create a Main Menu (Start Game, Rules, Exit).
- Implement a Restart Game function.
- Research different Blackjack rule variations.

This time, we assigned tasks more clearly to avoid confusion, and we also started using GitHub Issues properly. We planned both coding and research tasks.

## *Sprint 3 Planning (2025/03/04 – 2025/03/18):*

From Sprint 2, we saw that working without clear deadlines for big features slowed us down.
For Sprint 3, we focused on:

- Implementing a Betting System.
- Adding Multilingual Support.
- Creating a Database for Achievements.
- Applying the Singleton Design Pattern.

We gave each major task a specific responsible person and set small internal deadlines, which helped the team organize better.

## *Sprint 4 Planning (2025/03/18 – 2025/04/01):*

After Sprint 3, we understood we needed to break complex tasks into smaller ones.
In Sprint 4, we planned:

- Implement Local Multiplayer (divided into smaller sub-tasks).
- Improve GUI layout for better readability.
- Add Pause/Resume functionality.
- Add background images and music.

We also planned to be more flexible if priorities changed during the sprint, which helped when the teacher added new requirements.

## *Sprint 5 Planning (2025/04/01 – 2025/04/22):*

After Sprint 4, we realized the importance of leaving more time for final debugging.
In Sprint 5, we focused on:

- Finalizing Save/Load system.
- Adding Sound Effects for better user experience.
- Creating a new Achievements Window.

We scheduled complex tasks first in the sprint to make sure we had enough time to test and fix issues before the final submission.

### *Sprint 6 Planning (2025/04/23 – 2025/05/06):*

Sprint 6 was the final sprint before submission. The goal was to polish the project and tie up loose ends:

- Refactor project to fully follow MVC architecture.
- Finalize multiplayer logic (Java Sockets).
- Fix known bugs and finalize GUI.
- Add Javadocs and README/USERMANUAL content.
- Add license to our project
- Implement final Undo and Language fixes.

This sprint was about stability, documentation, and delivery.

## 2.4. Sprint Reviews

At the end of each sprint, we held a sprint review to check what we had achieved, what still needed work, and how close we were to what we had originally planned.
Each review helped us understand where we were improving and what we had to fix for the next sprint.

### *Sprint 1 Review (2025/02/03 – 2025/02/17):*

- We managed to make a basic but functional Blackjack game that you could play on the console. Players could hit, stand, and the game would correctly calculate scores and decide the winner.
- We also created the 52-card deck and handled the drawing of random cards.
- Setting up Git was messy at the beginning because not everyone was used to it, but it got better by the end of the sprint.

**Definition of Done:**

> Game playable fully in the terminal; deck works; players and dealer can take actions; score calculation is correct; project and glossary are organized.

### *Sprint 2 Review (2025/02/18 – 2025/03/04):*

- We created our first GUI demo! It was basic but it made the game look much more professional.
- The Main Menu was added (Start, Rules, Exit), and it made the game more user-friendly.
- We also added the ability to restart the game without having to quit and re-open it, which made testing much easier.
- Documentation was better organized after this sprint compared to the first one.

**Definition of Done:**

> GUI is launched properly; Main Menu buttons work; Restart Game button resets everything; documentation improved and clear for all members.

## *Sprint 3 Review (2025/03/04 – 2025/03/18):*

- The betting system was fully functional — players could bet and win or lose money, and the balance updated correctly after each round.
- The game was also translated into 6 different languages, which made it feel a lot more polished.
- We also set up a basic Achievements database to start tracking player progress across games.
- A Singleton pattern was introduced for some parts of the project (like managing the deck), and even though it was a bit new to some of us, it helped structure the code.

**Definition of Done:**

> Betting and balance work as expected; game text changes with language settings; achievements start being recorded; Singleton pattern implemented properly.

## *Sprint 4 Review (2025/03/18 – 2025/04/01):*

- We finished one of the hardest things: local multiplayer. Now two players could take turns playing on the same computer.
- We resized and reorganized the GUI (bigger fonts, cleaner buttons) to make the game easier to understand.
- A Pause button was added, letting players take a break in the middle of a game, and background music was added too.
- We also fixed an annoying bug where special card texts would overwrite other important messages during gameplay.

**Definition of Done:**

> Multiplayer mode available; pause/resume menu working; improved GUI layout; background music plays without lag; no more text-overwriting bugs.

## *Sprint 5 Review (2025/04/01 – 2025/04/22):*

- We added a Save and Load system, which means players could now save a game and continue later — something we're really proud of.
- We added sound effects for major actions like winning, losing, or drawing cards, which made the game more immersive.
- Finally, we created a new Achievements Window that displayed all the unlocked badges nicely with better visuals.

**Definition of Done:**

> Save/Load working perfectly; Undo action available per turn; sound effects correctly triggered; Achievements window shows player progress properly.

## *Sprint 6 Review (2025/04/23 – 2025/05/06):*

- We refactored the entire codebase to follow the MVC architecture, which made our project much more maintainable and modular.

- We implemented multiplayer using Java Sockets, allowing players to host or join games with synchronized state across all clients.
- The save/load was refactored to more accurately follow the Memento pattern.
- We added full Javadoc documentation, completed the README and USERMANUAL, and made sure the language system worked in all supported languages.
- We also polished the GUI, including badge animations and final bug fixes, delivering a smoother user experience.

**Definition of Done:**

> Code follows MVC cleanly; Multiplayer runs with full sync; Javadocs written for all classes; README and USERMANUAL finalized; GUI bugs resolved; Language system functions correctly.

# 2.5. Sprint Retrospectives

Reflections on productivity and Scrum practices.

After each sprint, we took a bit of time to sit together and talk about what went well, what didn't, and what we could do better for the next sprint.
Looking back, we definitely got more productive and better at using Scrum as the semester went on.

**Sprint 1 Retrospective :**

- At the start, we were mostly figuring things out. Git confused half the team, and organizing tasks was super basic.
- Productivity wasn't bad, but we wasted time setting up GitHub, branches, and just getting the project to run for everyone.
- We also noticed people were picking random tasks instead of planning properly.
- After this sprint, we agreed to start using GitHub Issues seriously and make sure every task had someone responsible.

**Sprint 2 Retrospective:**

- Things got better. We actually used issues and branches properly, and short daily meetings helped a lot.
- Productivity went up because everyone knew what they had to do, and we stopped stepping on each other's toes.
- One thing we realized though: we sucked at estimating time for big tasks like the GUI.
- So for the next sprint, we promised to break down big tasks into smaller pieces to avoid last-minute panic.

**Sprint 3 Retrospective :**

- Scrum was working pretty well: meetings were faster, and the GitHub board made it clear what was going on.
- Giving clear ownership of stories helped a lot.
- We also learned that setting mini-deadlines mid-sprint (instead of everything at the end) made progress way smoother.

**Sprint 4 Retrospective :**

- This sprint felt a lot more professional.

- We split Multiplayer into mini-tasks like "deal with player turn" and "update balances", and it helped make the feature way less scary.
- Daily meetings were now super quick (like 5 minutes tops) because mainly everyone knew what they were doing.
- Even when the teacher threw new requirements at us halfway through, we adapted fast without freaking out.

**Sprint 5 Retrospective :**

- By this point it was just flowing naturally.
- We planned the tough stuff (like Save/Load) first, left some room for bugs at the end, and it worked perfectly.
- Everyone was contributing, and the GitHub board kept everything clean.

**Sprint 6 Retrospective :**

- This sprint was all about the final polish.
- We fully refactored the game to follow MVC, completed multiplayer support, and tied up all loose ends like Javadocs and final documentation.
- The team worked hard and pushed through a lot of tasks in the final days — except for one member who didn't contribute until the very last minute.
- That created some unnecessary pressure, but the rest of the team adapted and delivered.
- In the end, we're proud of how the project turned out and how far we came in both code quality and teamwork.

# 2.6. Product Backlog

Evolution of the backlog, changes, and motivations.

When we started the project, our product backlog was super simple, just the bare minimum to make a working Blackjack game in the terminal. As the weeks went by, we kept adding more ideas, new features, and also stuff our professor suggested during reviews. It honestly grew a lot more than we expected, but it also helped us keep track of everything.

| Sprint | Feature/Story | Motivation |
|---|---|---|
| 1 | Deck creation, score logic | Core game setup |
| 2 | GUI Demo, Restart Game | First user-friendly interface |
| 3 | Betting System, Language Support | Adds personalization and strategy |
| 4 | Multiplayer, Pause Menu | Enhances replayability and user control |

| | | |
|---|---|---|
| 5 | Save/Load, Sound FX | Completes functionality and polish |
| 6 | Undo, Java Multiplayer | |

## Sprint 1

The first version of the backlog had just the essentials:

- Build the deck (52 cards, shuffled)
- Implement player and dealer hands
- Add basic game logic: hit, stand, calculate score
- Compare hands to decide who wins
- Set up the Java project structure and a mini glossary

We were basically just trying to get something that worked — no interface, just terminal-based gameplay. It wasn't much, but it got the ball rolling.

## Sprint 2

After seeing the terminal version in action, we realized it wasn't very user-friendly. So for the second sprint, we expanded the backlog to include:

- First version of a GUI (with simple buttons and visuals)
- A main menu (Start Game, Rules, Exit)
- The ability to restart the game without closing it
- Better comments and code documentation
- A bit of research on different Blackjack rules

This sprint helped the game feel more like an actual game, not just a command-line app.

## Sprint 3

By this point, we wanted the game to feel more complete and personal, so the backlog grew again with:

- A betting system (with balance updates)
- Multilingual support (we added 6 languages)
- A small database to track achievements
- Singleton pattern in a few key classes like the game manager

Some of these were a bit ambitious, but they made the game feel more like something we'd actually play (or show off).

## Sprint 4

This sprint had the craziest backlog because we added multiplayer and made a bunch of UI improvements. Tasks included:

- Local multiplayer mode (two players taking turns)
- GUI redesign (bigger buttons, cleaner layout)
- Pause/Resume functionality with a proper menu
- Background image and looping music
- Fixing an issue where card messages would overwrite each other
- Moving and redesigning the betting UI to make it easier to use

This was the moment the game started to feel real. A lot of stuff got added here, but it paid off.

## Sprint 5

This sprint was all about laying the groundwork for the final submission. We prioritized implementing core quality-of-life features and restructuring logic-heavy components. While the number of tasks was moderate, the technical depth of the work increased significantly. Key tasks included:

- Save and Load functionality
- Undo last move
- Sound effects(game actions like drawing cards, winning, and losing)
- GUI Achievements Window
- Refactoring of GUI

This sprint marked a shift from just "building features" to polishing and tightening the core game loop and user experience. Complex logic (like undo state rollback and save consistency) was carefully handled through design patterns and deep testing.

## Sprint 6

This sprint focused on wrapping up all major development and polishing features in preparation for the final submission. While the backlog was shorter than previous sprints, the tasks were more complex and detail-oriented. Key tasks included:

- Save and Load game progress
- Undo last move (implemented using the Command pattern, allowing a single undo per turn).
- GUI for achievements was visually improved (badges, animations).
- Focus on *synchronization issues*, *multiplayer finalization*, and *bug squashing*.
- Much more attention to *documentation*, *UML*, *README*, and final packaging.

This sprint was all about ensuring the product was complete, tested, and aligned with MVC and documentation requirements. While no new features were introduced, a lot of polish, bug-fixing, and integration took place to prepare for release.

This was the moment the project transitioned from "almost there" to a finished, presentable, and playable product. Every member contributed critical finishing touches to ensure the quality of Blackjack 2.0.

# 2.7. Sprint Backlog

Throughout the project, we used the **GitHub Project Board** to manage and visualize our user stories, sprint goals, and individual tasks. We followed an Iterative Development setup with the following columns:

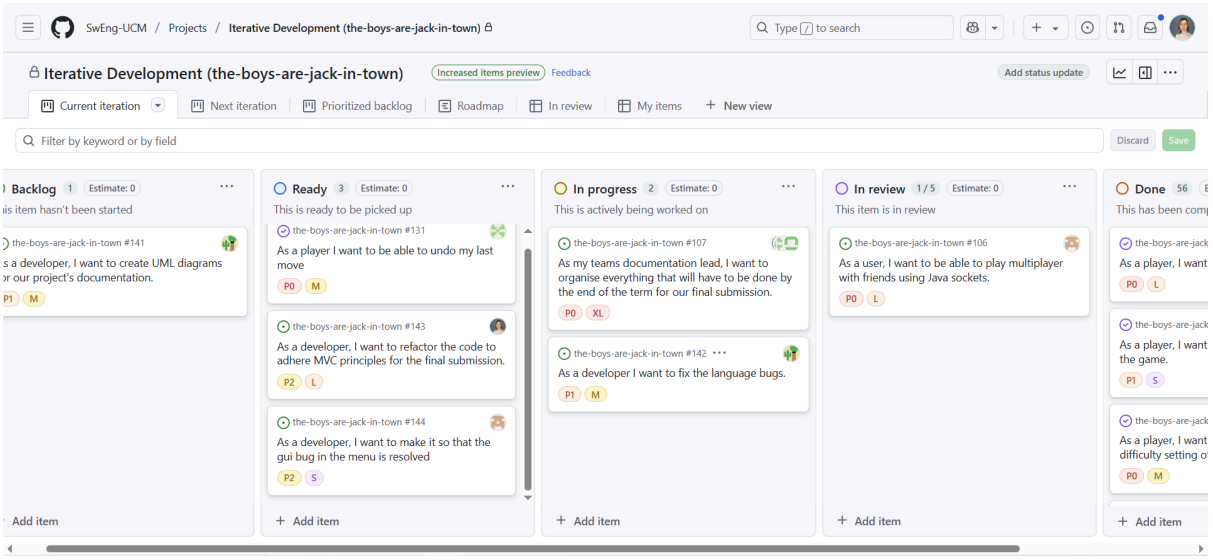
- **To Do** – All new user stories and backlog items waiting to be picked up.
- **In Progress** – Tasks currently being developed by assigned team members.
- **Review / Blocked** – Tasks needing review, feedback, or unblocked by another dependency.
- **Done** – Completed and tested tasks for the sprint.

During each sprint planning meeting, we selected a subset of user stories from the product backlog and moved them to "To Do." Team members then self-assigned tasks or were assigned by the Scrum Master depending on availability. As work progressed, we moved cards across the board to reflect status changes, which helped all members monitor progress in real time.

We also used labels to categorize tasks by type (e.g., "GUI", "Logic", "Research", "Documentation") and milestones to group tasks per sprint. Comments within the cards allowed for discussion, quick bug reporting, and linking relevant commits or pull requests.

This system allowed the team to stay aligned, avoid duplicated work, and maintain a clear workflow even during hybrid (in-person + remote) coordination.

*Here is an example we captured at the very last sprint:*



# 2.8. Description of the Work Performed by Each Group Member

*Individual contributions, roles, and special circumstances.*

## *Member 1: Finn Farrell*

**Primary Role(s):**

Git Expert

**Sprint Contributions:**

**Sprint 1:**

- As a developer, I want to be able to remove cards from the player's hand.
- I want to compare the player's and dealer's hands so that a winner is determined.
- As a developer, I want to delimit the first prototype, so that I can establish a clear scope and ensure a structured initial implementation of the game.
- As a user, I want a glossary for the first prototype so that I can easily understand key terms and concepts used in the prototype.
- As a developer I want to create and store the typical 52 deck of cards, so the user can have all the available cards.
- As a player, I want to see a clear message when I win or lose so that I understand the outcome of the game.

**Summary:** Developed core game mechanics, including card removal, hand comparison, deck creation, and win/lose messaging, while defining the prototype's scope and adding a glossary for clarity.

**Sprint 2:**

- I want to reveal the dealer's hidden card at the appropriate moment so that the game follows standard Blackjack rules.
- As a developer, I want to improve code comments and documentation so that the project is easier to maintain.
- As a player, I want to have a DEMO to interact with a graphical interface instead of just the terminal so that the game is more engaging.

**Summary:** Enhanced game realism by revealing the dealer's hidden card, improved code documentation, and introduced a graphical interface demo for better user engagement.

**Sprint 3:**

- As a developer I want to implement different languages to make the game accessible for all players.

**Summary: Implemented multilingual support to make the game accessible to a wider audience.**

**Sprint 4:**

- As a developer, I don't want the special card text to overwrite the game text
- As a user, I want the gui to be bigger and easier to use.
- As a user, I want the betting system to be in a different place so it is easier to use.

**Summary:** Refined the GUI by preventing text overlap, enlarging the interface, and repositioning the betting system for improved usability.

**Sprint 5:**

- As a developer, I want to save / Load with new Main.
- As a player, I want to save/load the game.

**Summary:** Added functionality to save and load games via a new main system, enabling persistent gameplay.

**Sprint 6:**

- As a developer I want to create UML diagrams for our projects documentation.
- As a developer I want to fix the language bugs.
- As a developer, I want to refactor the code to adhere to MVC principles for the final submission.

**Summary:** Finalized the project with UML diagrams, language bug fixes, and code refactoring to fully adopt the MVC architecture for submission.

**Technical Tasks & Features Worked On:**

Worked on a lot of the early implementation of core game mechanics, creating decks, determining winners, revealing hidden cards and displaying game messages.

Created and populated the wiki's GitHub section with all the necessary tools for working on the project, keeping in mind not to *"information overload"* since Git has an extensive library that can be overwhelming to those who haven't used it. Designed branch naming standards to allow easier navigation of projects work while it was being implemented in real time.

Implemented the games language system to support 6 world languages including English, Castellano, Hungarian French, Arabic and endangered Gaeilge. Ensured that all meaningful messages and texts in the game were correctly translated and displayed to the best of their ability.

Touched up the graphical user interface when components I had implemented affected the group's efforts to deliver a stylish and comfortable user experience. Added revolving title text to the menu screen.

Produced the game's Save/Load features to allow dynamic saving of game states, and loading of previous games in accordance with the Memento behavioural design pattern. This required installing and implementing the Jackson library .jar files to facilitate the serialization and deserialization of java objects.

Performed a portion of the refactoring of the GUI class to cut the large file down by separating the initialization and layout of components and the attaching of their respective action listeners into smaller, more digestible subclasses. Namely GUIComponentInitializer, GUILayoutBuilder and GUIEventBinder.

Fixed numerous bugs in preparation of the final submission and release of the groups BlackJack 2.0 project.

**Collaboration & Support to Others:**

Worked alongside David in the implementation of the Save and Loading, and later Kate in the refactoring of said system.

Gave Git advice/help to team members who needed it, especially those who were unfamiliar with how git worked.

Helped Hanna out in her substantial undertaking in the refactoring of the GUI classes to fit the MVC design pattern.

Provided Kate with a laptop and a helping hand in testing her work on the Java sockets and the recording of the demos for the sprint review.

**Special Circumstances (if any):**

## *Member 2: David Jiménez Calderón*

**Primary Role(s):**

N/A

**Sprint Contributions:**

Sprint 1:

- As a researcher, I want to explore innovative Blackjack mechanics, so that I can suggest new rule variations to enhance replayability.
- As a researcher, I want to explore different rules for the game.
- As a developer, I want to delimit the first prototype, so that I can establish a clear scope and ensure a structured initial implementation of the game.
- As a researcher, I researched different card options that we can add to the existing game in the next sprint.

Sprint 2:

- As a researcher, I want to study different regional variations of Blackjack, so that I can identify unique rule sets to incorporate into our game.

Sprint 3:

- As a researcher, I researched different card options that we can add to the existing game in the next sprint.
- As a researcher, I researched for different betting features

Sprint 4:

- As a developer, I looked carefully at the spelling in Spanish throughout the game.
- As a developer I want to perform a large scale refactoring so that we have easier code integration going forward. (multiplayer)

Sprint 5:

- As a player, I want to save/load the game.

Sprint 6:

**Technical Tasks & Features Worked On:**

- Multiplayer
- Language (spanish)

**Collaboration & Support to Others:**

- Finn helped me introduce me to github

- Collaborated with Kate (pair programming) on the large scale multiplayer refactor
- Helped Finn implement some of the save/load
- Hannah helped me on how to commit

**Special Circumstances (if any):**

Zero knowledge on java (not taking tp1 or tp2)

## *Member 3: Kate O'Reilly*

**Primary Role(s):**

Product Owner

**Sprint Contributions:**

Sprint 1:

- As a developer, I want to be able to remove cards from the player's hand.
- As a developer I want to print the cards stored in the players hand in an appropriate format
- As a developer I want to break down class structures for first prototype from user stories
- As a developer I want to create and store the typical 52 deck of cards, so the user can have all the available cards.
- As a developer, I want to set up the project structure, so that the game has a solid foundation for future development.
- As a developer, I want to delimit the first prototype, so that I can establish a clear scope and ensure a structured initial implementation of the game.
- As a dealer, I want to start a new game so that I can deal initial cards to the player and myself
- I want to allow the player to start a new round after a game ends so that they can continue playing
- As a user, I want to be able to quit the game, aka terminate

**Summary**: In Sprint 1, I established the foundational structure of the Blackjack game by setting up the project architecture, defining core class structures, implementing a standard 52-card deck, enabling card removal and formatted display, supporting game lifecycle features like starting new games or rounds, and allowing the user to quit—laying the groundwork for functional gameplay.

Sprint 2:

- As a developer, I want to implement that the user can restart the game after each round
- As a player, I want to have a DEMO to interact with a graphical interface instead of just the terminal so that the game is more engaging
- As a player, I want to be able to automatically split my hand when I draw a Split Ace card, so that I can play two separate hands without manually choosing to split.
- As a player, I want the Joker Wild card to act as any value between 1 and 11, so that I can strategically adjust my hand's score
- As a player, I want to be able to draw a Blackjack Bomb card, so that I can instantly win the game upon receiving it.
- As a player, I want correct messages to display when wildcards are drawn

**Summary**: In Sprint 2, I enhanced gameplay dynamics by integrating special wildcard behaviors (Split Ace, Joker Wild, Blackjack Bomb), implemented GUI-based interaction for improved engagement, enabled game restarts after rounds, and ensured appropriate messages display for wildcard events to create a more interactive and strategic playing experience.

Sprint 3:

- As a player I want the logic of the split ace to function correctly.
- As a player I want to create a functioning GUI

**Summary**: In Sprint 3, I focused on establishing a fully functional GUI and ensured that the Split Ace logic behaved correctly, enhancing both the visual experience and core gameplay mechanics for players.

Sprint 4:

- As a developer I want to perform a large scale refactoring so that we have easier code integration going forward.
- As a player I want to be able to play multiplayer with a friend on the same computer

**Summary**: In Sprint 4, I performed a large-scale refactoring to streamline future code integration and added support for local multiplayer, allowing two players to play together on the same computer.

Sprint 5:

- As a developer I want to create separate classes for server side and client side code when using Java sockets.
- As a user I want to have the option to decide to act as a host or client when playing multiplayer.

**Summary**: In Sprint 5, I significantly advanced the multiplayer infrastructure by cleanly separating server-side and client-side responsibilities using Java sockets, enabling scalable and maintainable communication logic. You also implemented a robust mechanism allowing users to choose whether to host or join a game session, laying the foundation for flexible and user-driven multiplayer gameplay.

Sprint 6:

- As a developer I want to debug the multiplayer functionality
- As a developer I want to refactor the save/load to ensure it follows the Memento pattern.
- As a developer I want to add Javadocs comments to all of the classes in the View, those associated with the Java sockets, and also the main GUI class.
- As a developer I want to fix the bug with the currentPlayerIndex
- As a developer I want to update the final README and User Manual so they are ready for final submission.

**Summary:** In Sprint 6, I tackled critical polish and stability tasks to finalize the project. I rigorously debugged the multiplayer functionality, resolving key issues like the currentPlayerIndex bug to ensure smooth turn-based gameplay. I refactored the save/load system to properly adhere to the Memento design pattern, enhancing maintainability and design consistency. I also elevated the codebase quality by comprehensively documenting the View, socket-related classes, and main GUI with detailed Javadocs. Finally, I prepared the project for submission by updating the README and User Manual, ensuring clarity and usability for both users and reviewers.

**Technical Tasks & Features Worked On:**

- Multiplayer architecture (host/client distinction)
- Wildcard game logic (Split Ace, Joker Wild, Blackjack Bomb)
- Core GUI development and integration
- Save/load system refactor (Memento pattern)

- Game restart and quit mechanisms
- Contributed UML and the structure of everyone's individual contribution section to the final documentation.
- Took on a significant amount of the final debugging.
- Major refactoring tasks across multiple sprints
    - Fully refactored GameManager, BlackjackGUI with Hanna and Finn, and related controllers in Sprint 6 to eliminate MVC violations and prevent GUI-model coupling. These 2 classes were the hardest to refactor because of them being 1000+ lines each.

**Recurring Sprint Tasks**

- Created Sprint Goals
- Managed the Iterative Development Board
- Hosted Planning Poker in sprints 5 and 6
- Created the User Stories and assisted in writing the wiki
- Created and presented almost every Sprint Presentation and demo video.
- Took on any additional tasks requested by the Scrum Master
- Managed the Releases for every sprint

**Collaboration & Support to Others:**

- Collaborated with David (pair programming) on the large scale multiplayer refactor
- Collaborated with Finn by refactoring his save/load
- Worked with Hanna on the final refactoring and inclusion of Javadocs.
- Worked with Haroun in the creation of the initial GUI.
- Was supported by Finn in Java Sockets testing.
- Worked with Hanna in dividing all of the final debugging between us.

**Special Circumstances (if any):**

Significant illness in Sprint 3

## *Member 4: Haroun Riahi*

**Primary Role(s):**

Team Advocate

**Sprint Contributions:**

Sprint 1:

- Delimit first prototype
- As a developer, I want to delimit the first prototype, so that I can establish a clear scope and ensure a structured initial implementation of the game
- As a player or dealer, I want to draw a random card (or multiple random cards) from the deck, so that I can play the game fairly with randomized card distribution
- I want to immediately determine the winner if either the player or dealer gets a Blackjack (21 with two cards) so that the game ends correctly

- As a developer, I want to research about the best possible UI for the game
- As a UI/UX designer, I want to research user needs and expectations, so that I can design an interface that is intuitive and user-friendly

Sprint 2:

- As a player, I want to have a DEMO to interact with a graphical interface instead of just the terminal so that the game is more engaging

Sprint 3:

- As a player, I want to place bets so that I can gamble within the game

Sprint 4:

- As a player, I want to see my achievements from the game
- As a player, I would like to see nicer backgrounds maybe hear some songs
- As a player, I want the betting system to correctly add my winnings to my balance when I win, deduct my losses when I lose, and return my bet in case of a tie
- As a developer, I want to ensure that the game correctly resets bets after each round and ends the game when a player or dealer runs out of money
- As a player I want to be able to pause the game and resume it
- As a player I want to be able to control music through the pause menu
- As a player, I want to pause the game when I need a break or get interrupted

Sprint 5:

- As a developer, I want to create a new UI window for the achievements
- As a player, I want to be able to choose the difficulty setting of my AI dealer

Sprint 6:

- As a player, I want the Instructions window to be clear, updated, and visually appealing
- As a player, I want the interface to show correct button styles and hover effects
- As a player, I want all text to be translated based on the selected language
- As a player, I want the window banner (title) to match the selected language
- As a player, I want the Options panel to match the casino-themed interface
- As a developer i want to fix even more bugs !!
- As developers, we want to organise everything that will have to be done by the end of the term for our final submission.

Technical Tasks & Features Worked On:

- Implemented random card drawing logic for both dealer and player, ensuring fairness and proper distribution.
- Developed core winner-determination logic
- Initiated the Java Swing GUI, later extended in collaboration with teammates.
- With Hanna’s help i helped implement a betting system .
- Fixed bugs in the gambling system, ensuring accurate functionality and edge-case handling.
- Implemented full French language support and improved multilingual system

- Designed a custom pause menu with multiple buttons and a volume slider for music.
- Enhanced UI with:
  - Animated backgrounds and card table graphics.
  - Custom icons for menus, buttons, and cards.
  - User-friendly layout, stylish main menu, and improved component alignment.
- Implemented difficulty setting using the Strategy design pattern, accessible from the options menu.
- Applied consistent UI/UX polish across sprints to improve player experience.

Collaboration & Support to Others:

- Fulfilled the role of Team Advocate, ensuring team cohesion, motivation, and active participation and teams building .
- Facilitated smooth sprint retrospectives, feedback sharing, and resolution of blockers.
- Helped teammates debug and fix:
  - Broken translations and missing text entries.
  - Path problems related to resources (images, icons).
  - Incomplete or non-working features across different components.
- Participated in sprint planning, stand-ups, and encouraged shared ownership of the game's development.
- Collaborated closely with the UI team (notably Kate) to progressively improve the GUI design.
- Ensured everyone in the team had a voice, contributed equally, and stayed focused on delivering a complete and polished project.

overall ,thanks to the collaboration, dedication, and support of every team member, we were able to overcome challenges, combine our strengths, and successfully deliver a complete and polished project

**Special Circumstances (if any):**

## *Member 5: Hanna Szalai*

**Primary Role(s):**

Scrum Master

**Sprint Contributions:**

Sprint 1:

- I want to make the dealer draw cards according to the game rules so that the game follows standard Blackjack behavior
- As a developer, I want to be able to remove cards from the players hand.
- As a user, I want to play with the game (hit, stand) in the terminal
- As a developer, I want to delimit the first prototype, so that I can establish a clear scope and ensure a structured initial implementation of the game.
- As a dealer, I want to deal a card to a player so that they can build their hand.
- As a player, I want to see my current hand so that I can know which cards I have.
- As a player, I want to calculate my score so that I can know my current score.
- As a player, I want to receive a card so that I can build my hand.
- As a developer, I want to prevent the player from choosing multiple actions at once ( multiple choices in a row).

**Summary**: I focused on implementing fundamental Blackjack gameplay logic, including card actions, score calculations, and enforcing valid player input. I also helped set the project scope by shaping the first prototype.

Sprint 2:

- As a developer, I want to refactor the project into the MVC pattern by the end of the sprint.
- As a player I want to see a menu page with options when I start the game.

**Summary**: I led the initial architectural transition to MVC (which we did not maintain onwards so at the end it was hell of task to transition) and implemented the main menu UI, laying the groundwork for a modular and scalable project structure.

Sprint 3:

- As a developer, I want to integrate a Singleton Design Pattern in my project.
- As a player, I want to place bets so that I can gamble within the game
- As a developer I want to see the dealer's balance and bets each round.
- As a developer, I want to create a database for the achievement information.

**Summary**: I implemented core betting mechanics, set up Singleton managers for centralized logic control, and helped lay the foundation for achievements tracking.

Sprint 4:

- As a player I want to see my achievements from the game.
- As a player, I would like to see nicer backgrounds maybe hear some songs

**Summary**: I focused on building the early Achievements UI, supported the integration of music, and refined the audio experience in collaboration with other team members.

Sprint 5:

- As a player, I want to hear sound effects during the game.
- As a developer, I want to create a new UI window for the achievements.

**Summary**: I contributed major UI improvements and implemented sound effects.

Sprint 6:

- As developers, we want to organise everything that will have to be done by the end of the term for our final submission.
- As a developer, I want to create Sequence Diagrams based on my main user stories.
- As a player, I want badges to visually update from grey to colorful when unlocked, so that I can feel rewarded for achievements
- As a developer i want to fix even more bugs.
- As a developer, I want to refactor the code to adhere MVC principles for the final submission.
- As the scrum master, I want to create and continuesly modify the main README.md and USERMANUAL.md files.
- As a player, I want to undo my last betting or gameplay action, so that I can recover from mistakes and improve strategy.
- As the Scrum Master, I want to create a LICENSE for the project.
- As a player I want to be able to undo my last hit or stand move using the Command Design Pattern
- As a developer, I want to add Javadocs to every class.

**Summary**: Sprint 6 was my most productive sprint. I led multiple refactors, implemented key architectural patterns, polished the GUI, documented the entire system, and ensured our final submission met all functional and technical goals.

**Technical Tasks & Features Worked On:**

- Served as **Scrum Master** for the entire project duration: organized Sprint Planning, Reviews, Retrospectives, and daily standups.
- Kept project **wikis** and **documentation** updated after **every sprint** with accurate, well-written entries and synced status with GitHub issues.
- Maintained the GitHub Project Board: kept cards and milestones up to date, categorized with labels (e.g., GUI, Logic, Docs), and assigned tasks to team members.
- Managed team coordination in hybrid (in-person + remote) settings to ensure task visibility and role clarity.
- **Fully refactored** GameManager, BlackjackGUI with Kate and Finn, and related controllers in Sprint 6 to eliminate MVC violations and prevent GUI-model coupling. These 2 classes were the hardest to refactor because of them being 1000+ lines each.
- Updated the **README.md and USERMANUAL.md** with game instructions, build/run guide and snapshots.
- Sprint 6 was my most productive sprint, where I implemented, reviewed, documented, and finalized more features than any previous sprint.
- **In summary:** I contributed across every layer of the project: game logic, architecture, GUI, sound, multiplayer, documentation, testing, and team management. **What I'm most proud of is the Undo system**, the **Achievements logic and interface**, and the **MVC refactor**, because they combined deep technical design with user-facing impact and significantly improved the quality and structure of the final game.

**Collaboration & Support to Others:**

- Helped David and Emre understand how to commit, resolve merge conflicts. Set up an SSH key and downloaded VS Code for both of them.
- Gave Kate code review help and debugging assistance during her refactor of multiplayer and Save/Load logic.
- Provided the documentation structure for the submission PDFs.
- Led all Sprint Plannings, Retrospectives, and facilitated daily meetings as Scrum Master.
- Collaborated with Kate, Finn in the final MVC decoupling and with Haroun in the final GUI refactoring.
- Finally, always provided help outside of class, during the evenings/nights and weekends if anyone was stuck or needed help.

**Special Circumstances (if any):**

- Was sick from March 31st to April 6th, (part of Sprint 5) but caught up quickly and delivered multiple tasks afterward.
- Took on extra responsibilities during the final sprint to offset delayed contributions from others.

## *Member 6: Fehmi Emre Eryilmaz*

**Primary Role(s):**

Documentation Lead

**Sprint Contributions:**

Sprint 1:

Sprint 2:

Sprint 3:

Sprint 4:

As a Document Lead, I started with the documentation.

As a Player, I want to implement an undo function.

Sprint 5:
As a player, I would like to add  images on cards

Sprint 6:

As a Document Lead, we want to finish the final document with full detail and fast as possible.

**Technical Tasks & Features Worked On:**
undo

card backgrounds

**Collaboration & Support to Others:**
Everybody was helpful at the start because I arrived late.

I could not do the undo function so Hanna did it herself.

**Special Circumstances (if any):**

I arrived to school 3 weeks late

Not much knowledge on java. This is my second time taking a class on Java.

# 2.9. Conclusions and Self-Assessment

Our team began this course with a passion for creativity but limited coordination and Git knowledge. Through the five sprints, we learned how to break down work efficiently, how to give ownership to user stories, and how to respond to setbacks. By Sprint 3, we had matured significantly in task estimation, documentation, and modular design.

Scrum ceremonies gave us structure, but our collaboration and flexibility were what carried the project forward. Challenges like adapting new design patterns (Singleton, Memento), integrating localization, and reworking GUI layouts taught us to think like engineers and collaborators. Most importantly, we realized the value of version control, clean interfaces, and keeping our users in mind.

Looking back, our greatest success was turning a console prototype into a fully playable, extensible, and enjoyable game with multiplayer, achievements, and modern UI. Given more time, we would improve our testing strategy (with JUnit or CI tools) and polish the multiplayer UX. Still, we're proud of what we've built and how we worked as a team.

# 3. UML Design

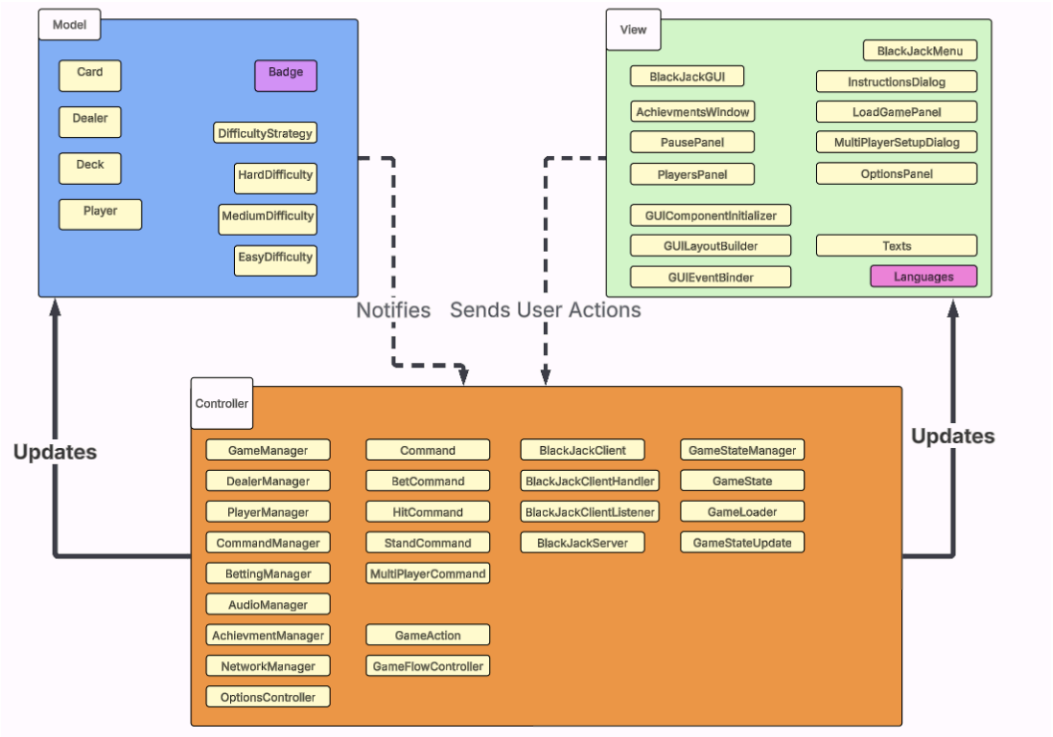
## 3.1. Final Architecture

Our final project architecture follows the tried and tested **Model View Controller** (MVC) design pattern. This pattern separates the project into 3 subsections, namely the model, view and controller, each performing a set of core functions in a highly modular fashion. This configuration promotes cleaner code and improves maintainability since a component can be modified without need to make changes to the other components it interacts with.

This design revolves around sending and receiving information based on the location of a component relative to the 3 packages. First, the user performs an action, such as starting the game or pressing a button. This request is handled by the **Controller**. Next the **Controller** updates the **model.** The **Model** handles the updated data and notifies the **Controller** that changes have been made. The **Controller** then updates the **View** with the newly gathered data. This tells the **View** to update the GUI which is then displayed to the user. Our user can then interact with the **View** and any new actions will be sent back to the **Controller.**

The MVC strategy has been particularly useful in the delivery of User Stories #13, #24, #31, #33, #42, #53, #54, #56 among others, some of which will be discussed further in section 3.3. The pattern has permitted the implementation of new features in an agile manner since the addition of the code needed to run the new functionality does not interfere with the existing code.

Basic overview of the MVC architecture.

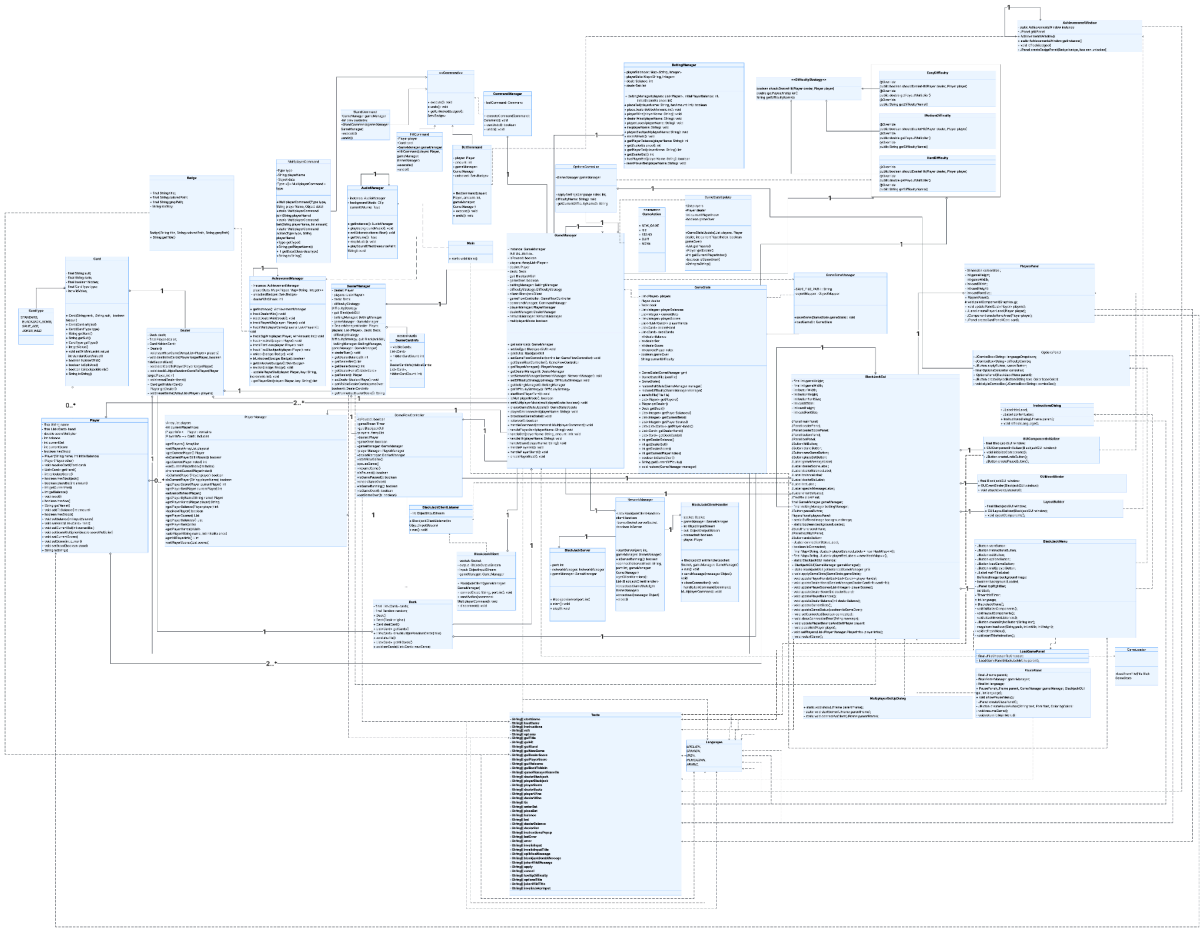



*Where the Yellow objects are classes, and Purple are enums. A more detailed description can be found in section 3.2.*

## 3.2. Final Structure

The final structure of our project is a highly modular approach, implemented after large scale refactoring to permit the extension of the source code. This meant that we ended up with more classes but ultimately a more flexible composition. Overly verbose classes, such as **GameManger** were broken up into subclasses to be called upon such as **PlayerManager, DealerManager, GUIComponentInitializer** among others. The relationships between classes is demonstrated in the diagram below. A dotted line with an arrow head representing the use of one class by another (**Dependency**) . A solid line with a diamond head represents when a class has a number of instances of another class (**Aggregation**). The **cardinality** of such relations can be found along the line connecting the classes. We can also see the application of design patterns (Memento, Singleton and Strategy), all of which are described in further detail in section 3.4.

*Projects final structure - uml diagram displaying class attributes, methods and relationships.*



*Link to an online version can be found below.*

*https://github.com/SwEng-UCM/the-boys-are-jack-in-town/blob/main/resources/doc%20pics/Project-UML-.png*

# 3.3. User Stories

*5 recommended user stories. For each, include design evolution, UML sequence/activity diagrams, class diagrams, and explanations.*

### User Story #53 : As a player, I want to save/load the game.

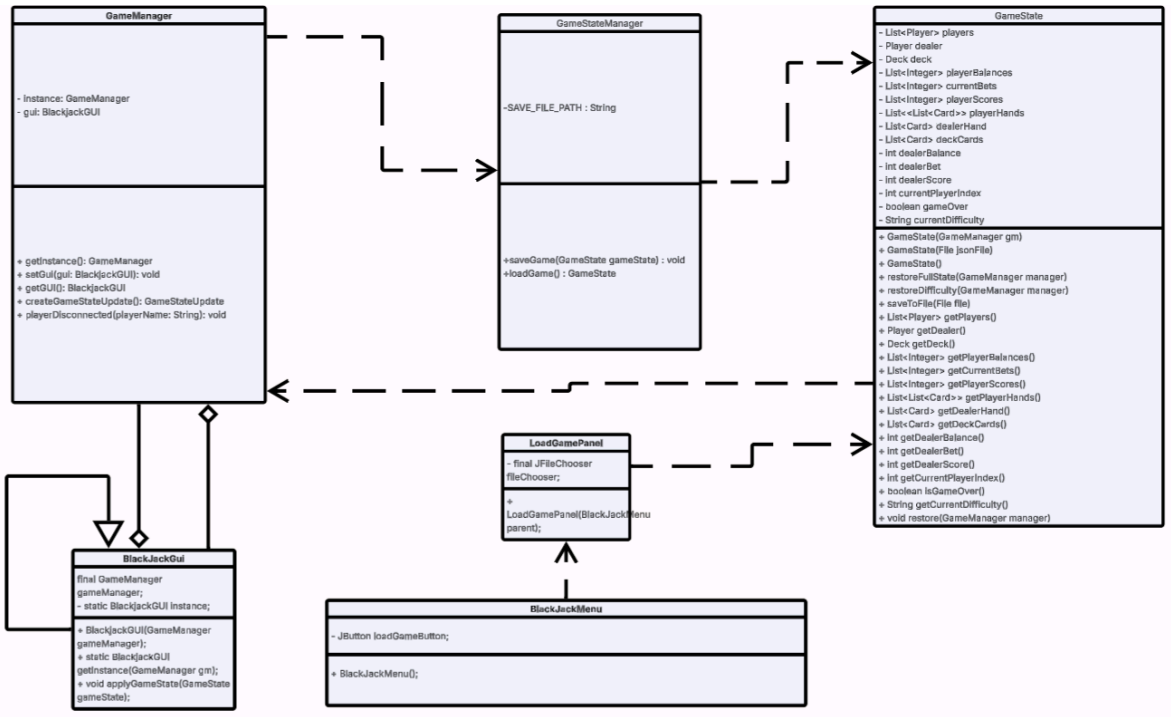
To carry out this User Story David and Finn first began by researching the common practices in regards to the implementation of saving and loading features. The most widely used method seemed to be the **"Memento"** design.

Next, we delegated our **Originator**, **Memento** and **Caretaker** components to the **GameManager, GameState** and **GameStateManager** respectively. Initially David and Finn had assigned the functionality of the careTaker to the **LoadGamePanel**, however this was later moved to a new GameStateManager class by Kate to help keep the code "Clean" and more robust.

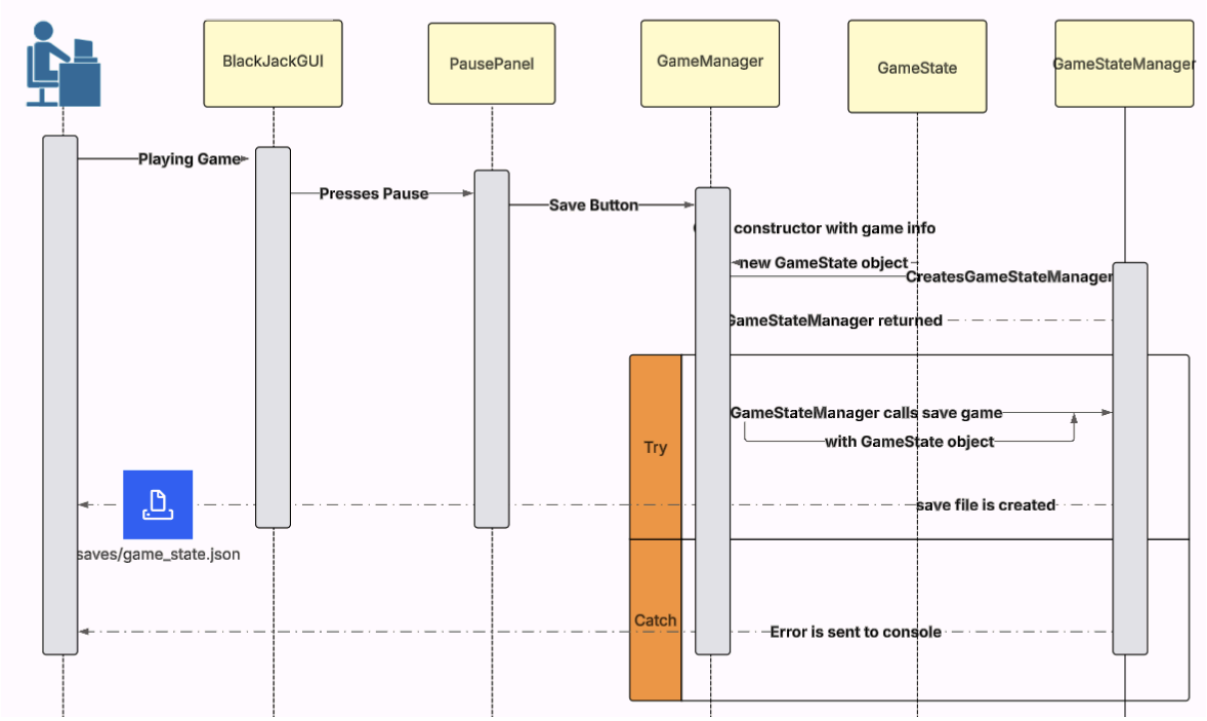
The **GameManager** handles the saving of the game to a **GameState** via the **GameStateManager**. The **GameStateManager** then saves the **GameState** locally in the file *"saves/game_state.json"*. The **GameState** stores the necessary information regarding a moment in time of the game's attributes. This **GameState** can then be loaded in by selecting the "Load Game" button in the **BlackJackMenu**. After a file has been loaded the **GameManager** calls to the **GameState** to restore itself. This populates the **GameManager** with the corresponding information from the **GameState**. Finally the **BlackJackGUI** is updated with the new **GameManger** data and the user can then resume the game they had been playing at the time of the save.

The design took a number of iterations to perfect. Bugs and breaks were handled by Finn, David and later Kate as they came up. One of the largest was the design of the initial plan, whereby the Loading was being implemented first, rather than the Saving which we later came to realise is the more pragmatic way of carrying out such a procedure as it will provide an easier saving (json, in our case) format for the loading to be carried out with.

Simplified UML of classes involved:



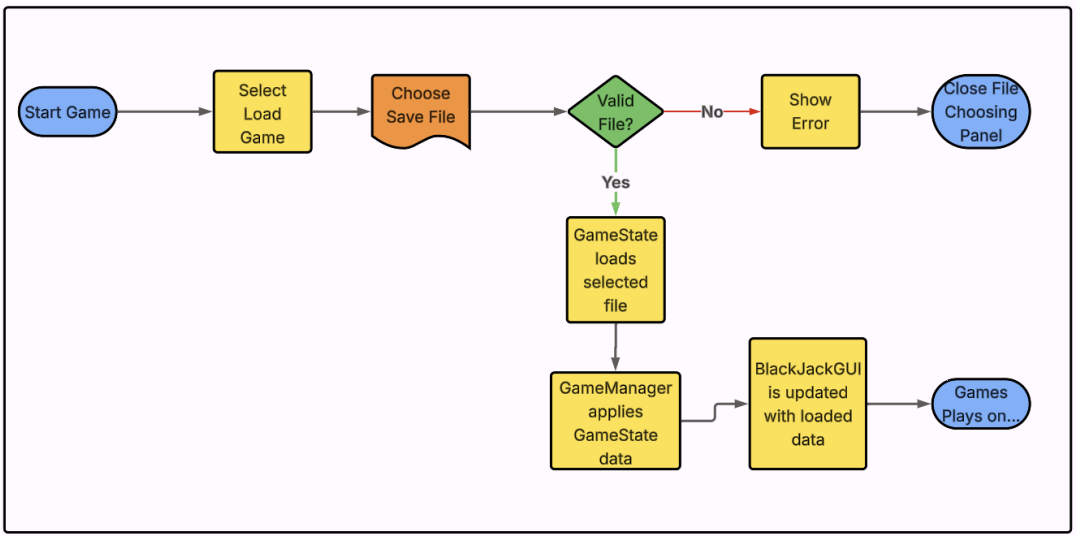
UML Sequence diagram detailing the saving procedure:




Once the saving procedure had been implemented, the loading feature was added. To do this we first created the LoadGamePanel that appears on the Main Menu. This handles the choosing of valid JSON files from the games save files. The JSON format was chosen because it was easiest to work with and implement given that our project was made using Java. Any invalid files are returned with an error message.

The pattern we implemented again follows the Memento design pattern, where once the Load is performed the **careTaker (GameStateManager)** applies the **Memento (GameState)** to the **Originator (GameManager)** and the game resumes play.

Activity Diagram demonstrating the Loading feature:



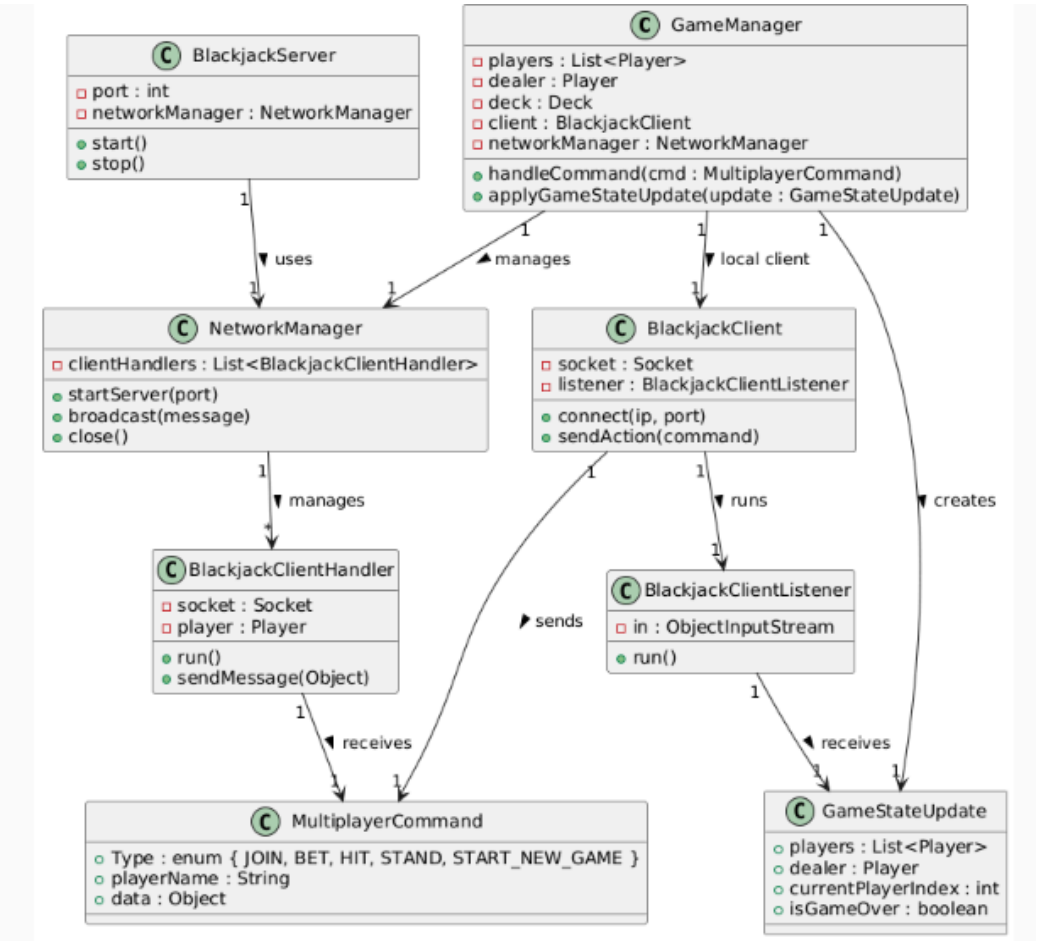
## User Story #106 : As a user, I want to be able to play with a friend using Java Sockets.

To fulfill this user story, Kate single-handedly designed and implemented the entire multiplayer functionality using Java Sockets. She began by researching real-time multiplayer patterns and found that Java's low-level Socket and ServerSocket classes were ideal for implementing a synchronous client-server Blackjack game.

I developed the multiplayer system around a clean separation of concerns:

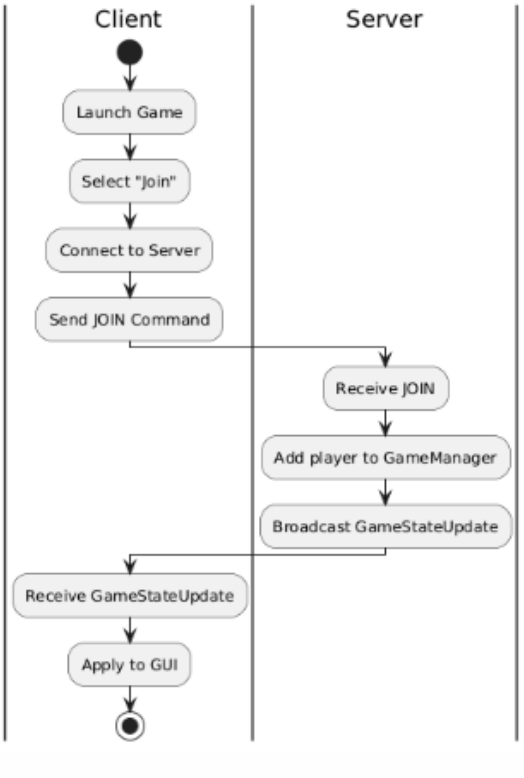
- The server logic was encapsulated in a BlackjackServer class.
- The client logic was handled by BlackjackClient, with a listener thread (BlackjackClientListener) responsible for receiving game state updates.
- The communication layer was abstracted using NetworkManager, which handled broadcasting messages and client connection management.
- Each player joining the game was handled through a JOIN command, and their actions (e.g., BET, HIT, STAND) were serialized as MultiplayerCommand objects.

*UML Class Diagram — Multiplayer Architecture*



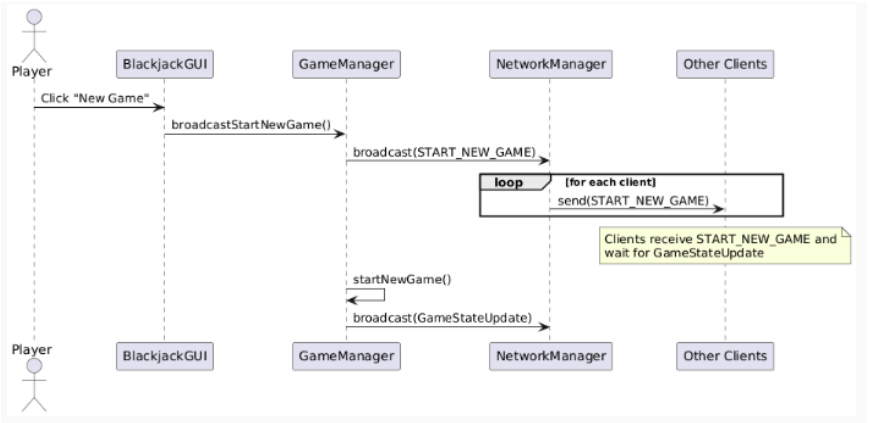
To prevent illegal moves and ensure fairness, I implemented strict turn validation on the server. Only the player whose turn it was — tracked via PlayerManager's currentPlayerIndex — was permitted to send actionable commands. Clients had their buttons disabled unless it was their turn.

*Activity Diagram — Client Join and Sync*




The most complex part came with synchronizing the start of new rounds. To solve this, I created a START_NEW_GAME command that the server could broadcast. When received by clients, they would not try to deal cards themselves, but instead wait for the server to send a GameStateUpdate, which kept everyone fully in sync.

*Sequence Diagram — Start New Game (Multiplayer)*



## User Story #81 : As a player, I want to place bets so that I can gamble within the game.

The UML class diagram illustrates the structure and interaction of the core classes involved in the **betting system** of the game. This user story evolved over several sprints — from the initial placement of bets to full integration with game outcomes (win/loss/tie) and GUI updates.

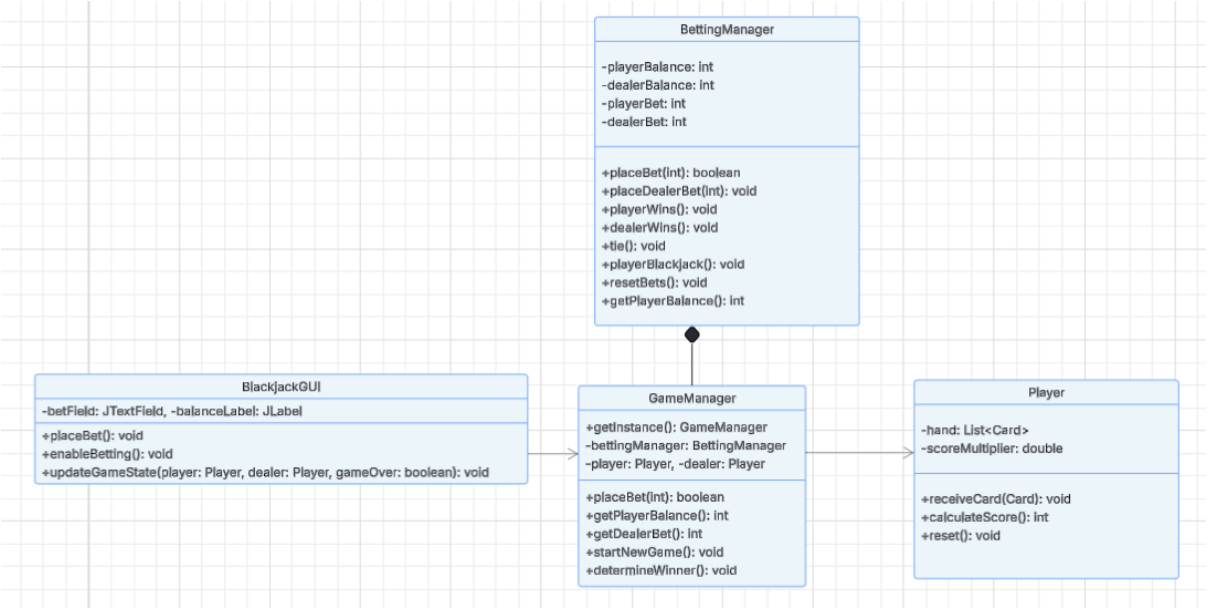
**Sprint 3: (where we implemented the functionality)**

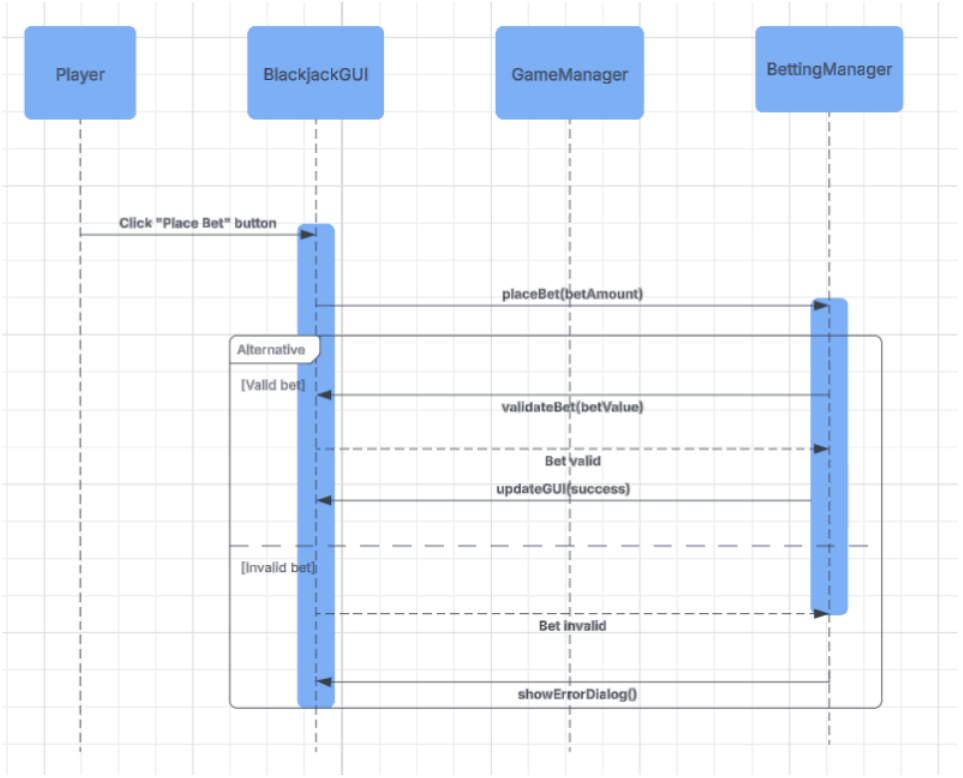
**Key Classes and Responsibilities**

- **BettingManager** (Controller Layer)
    - Acts as the central authority for managing all betting operations.
    - Stores both player and dealer balances and current bet values.
    - Provides functions to:
        - Place a bet for both player and dealer (placeBet, placeDealerBet)
        - Apply payouts (playerWins, dealerWins, tie, playerBlackjack)
        - Reset bet state after each round
    - Uses clean encapsulation and provides getters for balance/bet display.

- **GameManager** (Orchestrator / Game Logic)
    - Singleton pattern ensures centralized game flow.
    - Interacts with BettingManager to:
        - Deduct and apply bets
        - Trigger payouts based on game outcome
        - Reset states when appropriate
    - Also serves as an intermediary between GUI and game logic.

- **BlackjackGUI** (View Layer)
    - Contains components like betField, balanceLabel, and placeBetButton to gather user input and display current values.
    - Calls gameManager.placeBet() to place the bet and updates UI accordingly.
    - Reflects updated balances and dealer bets each round and after outcome is determined.
- **Player** (Model Layer)
    - Stores card hand and calculates score, but betting is abstracted out of the Player class — a good design decision to preserve **single responsibility**.




**Sprint 6:**

**BettingManager (Controller Layer)**

- Remains the **core class** responsible for managing all betting logic.
- Now supports **multiple players** via:
    - Map<String, Integer> playerBalances
    - Map<String, Integer> playerBets
- Responsibilities:
    - Place and validate bets for players and dealer:
        - placeBet(playerName, amount)

        - placeDealerBet(amount)
    - Handle payouts after outcomes:
        - playerWins(playerName), dealerWins(), tie()
    - Expose balance/bet data:
        - getPlayerBalance(playerName), getPlayerBet(playerName)
    - Reset bet values after each round
- **Does not directly own Player objects**, maintaining loose coupling from the model.

**PlayerManager (Controller Layer)**

- Manages all Player instances and provides access via name or turn.
- Responsibilities:
    - Add and retrieve players (addPlayer, getPlayerByName, getCurrentPlayer)
    - Interface to retrieve Player objects when BettingManager needs verification
- Acts as the **bridge between identifiers (names)** and Player instances in a multiplayer context.

**DealerManager (Controller Layer)**

- Responsible for dealer logic, including dealer betting and turn execution.
- Interacts directly with BettingManager to:
    - Place dealer bets automatically based on dealer balance
    - Apply payout logic depending on dealer result
- Encapsulates dealer's state and logic, reducing coupling in GameManager.

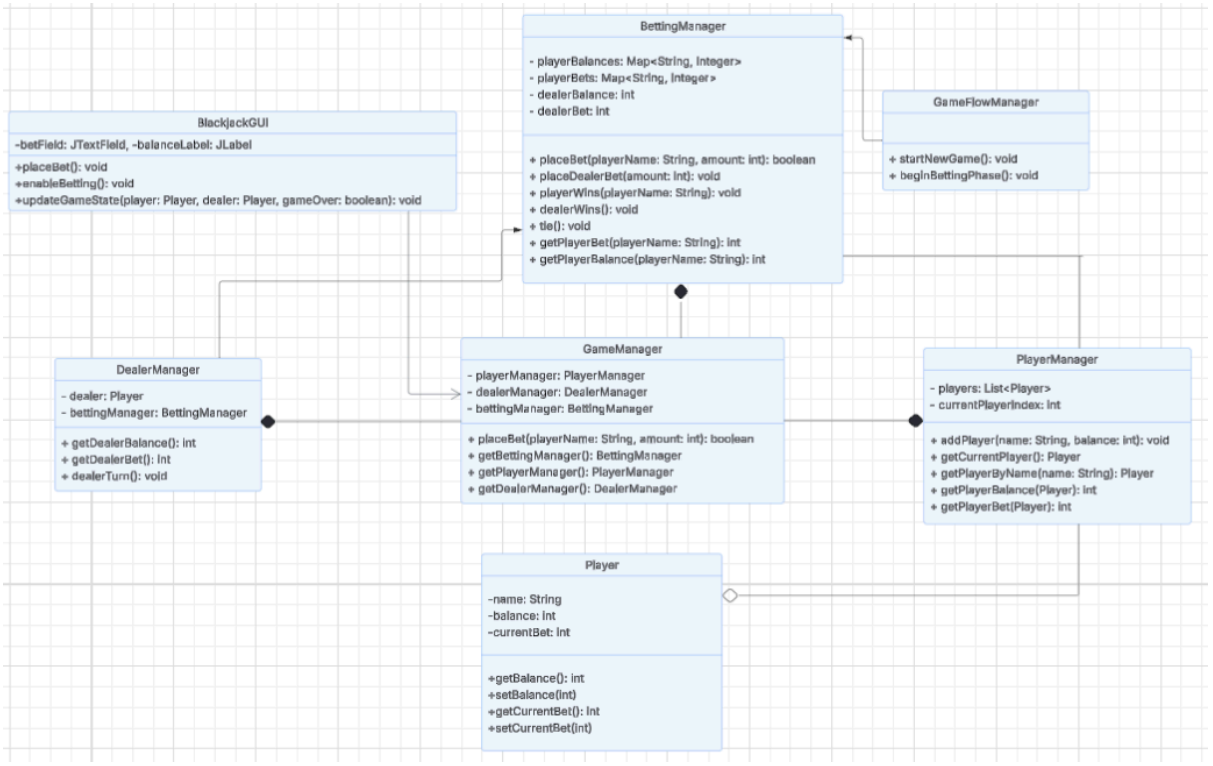
**GameFlowController (Orchestration Layer)**

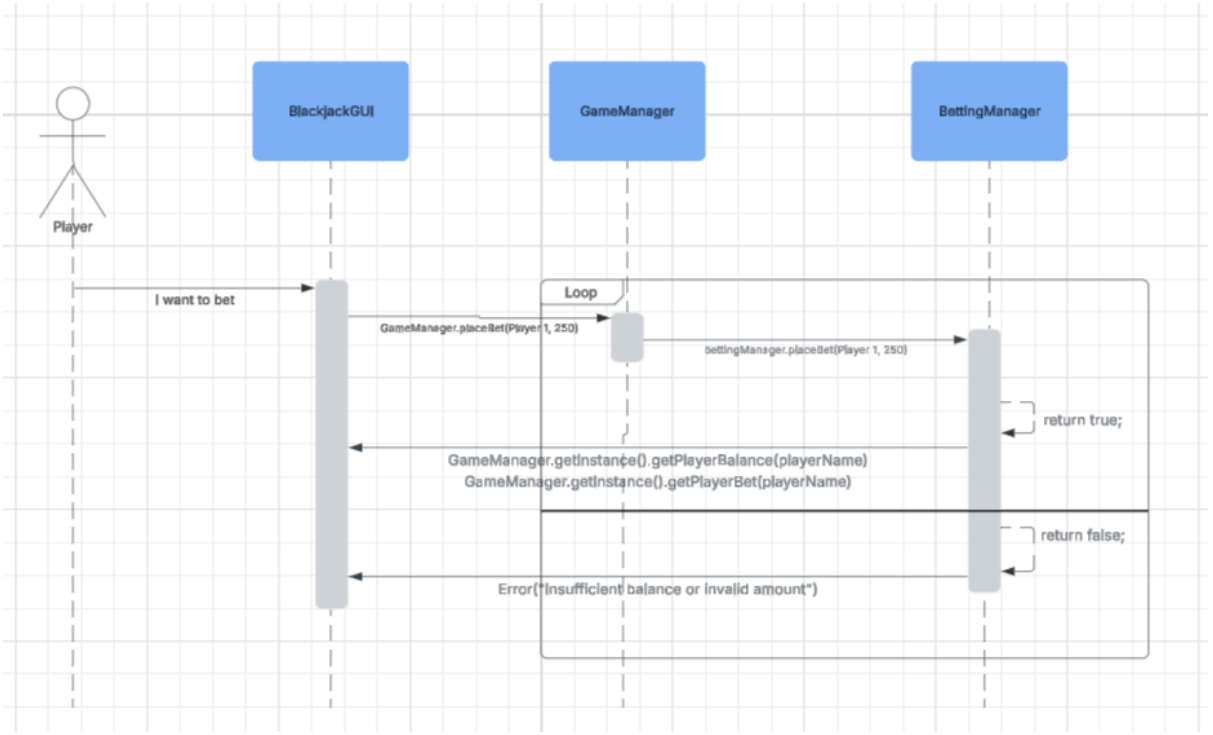
- Coordinates round-level flow, including:
    - Starting new games
    - Calling dealer logic
    - Initiating dealer bet logic at round start
- Interacts with PlayerManager, DealerManager, and BettingManager
- Ensures betting is completed **before gameplay starts**.

**GameManager (Central Controller / Singleton)**

- Orchestrates the overall game state and routes function calls between GUI and controllers.
- Holds instances of:
    - BettingManager
    - PlayerManager
    - DealerManager

- Handles UI event delegation (e.g., placeBet(), getPlayerBalance()).


BettingManager
- playerBalances: Map<String, Integer>
- playerBets: Map<String, Integer>
- dealerBalance: int
- dealerBet: int
+ placeBet(playerName: String, amount: int): boolean
+ placeDealerBet(amount: int): void
+ playerWins(playerName: String): void
+ dealerWins(): void
+ tie(): void
+ getPlayerBet(playerName: String): int
+ getPlayerBalance(playerName: String): int
GameFlowManager
+ startNewGame(): void
+ beginBettingPhase(): void
BlackjackGUI
-betField: JTextField, -balanceLabel: JLabel
+placeBet(): void
+enableBetting(): void
+updateGameState(player: Player, dealer: Player, gameOver: boolean): void
DealerManager
- dealer: Player
- bettingManager: BettingManager
+ getDealerBalance(): int
+ getDealerBet(): int
+ dealerTurn(): void
GameManager
- playerManager: PlayerManager
- dealerManager: DealerManager
- bettingManager: BettingManager
+ placeBet(playerName: String, amount: int): boolean
+ getBettingManager(): BettingManager
+ getPlayerManager(): PlayerManager
+ getDealerManager(): DealerManager
PlayerManager
- players: List<Player>
- currentPlayerIndex: int
+ addPlayer(name: String, balance: int): void
+ getCurrentPlayer(): Player
+ getPlayerByName(name: String): Player
+ getPlayerBalance(Player): int
+ getPlayerBet(Player): int
Player
-name: String
-balance: int
-currentBet: int
+getBalance(): int
+setBalance(int)
+getCurrentBet(): int
+setCurrentBet(int)

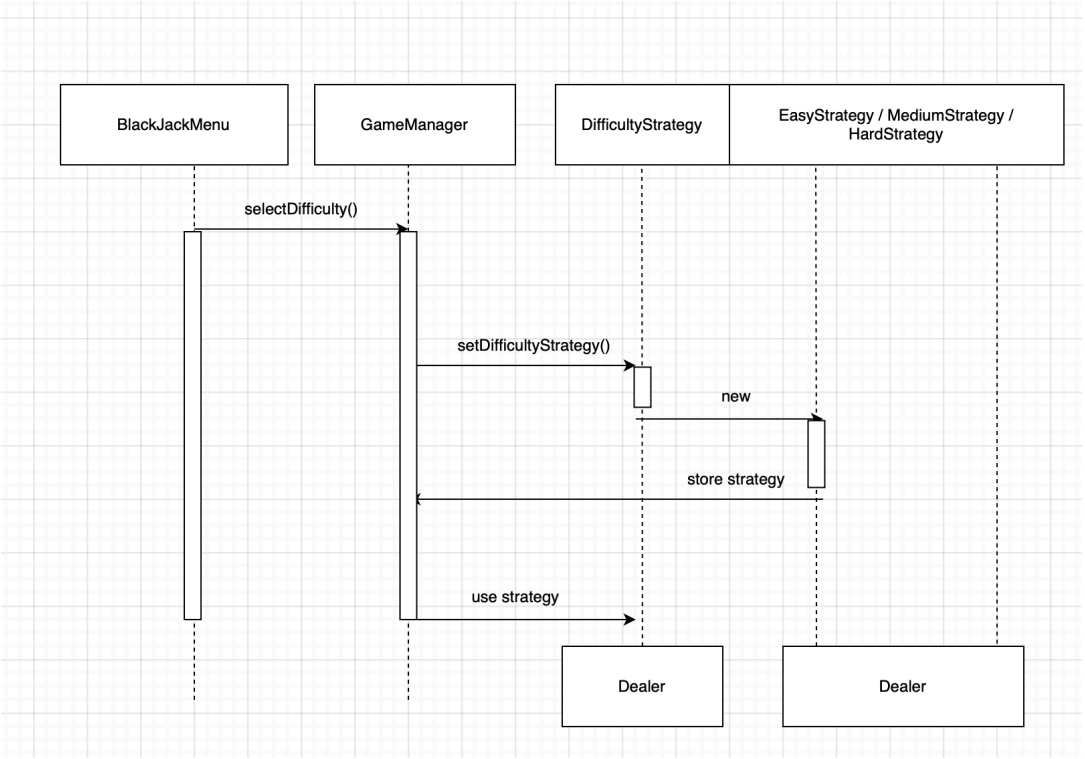

BlackjackGUI
GameManager
BettingManager
Player
I want to bet
Loop
GameManager.placeBet(Player 1, 250)
BettingManager.placeBet(Player 1, 250)
return true;
GameManager.getInstance().getPlayerBalance(playerName)
GameManager.getInstance().getPlayerBet(playerName)
return false;
Error("Insufficient balance or invalid amount")

## User Story : As a player, I want to be able to choose the difficulty setting of my AI dealer #130 , sprint 5

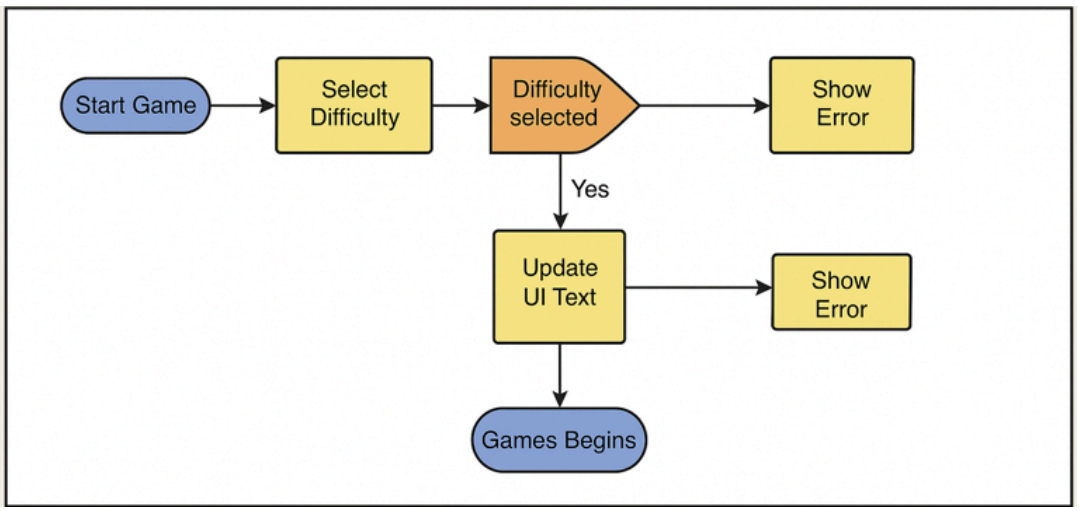
To realise this User Story, Haroun began by exploring ways to let players customize the challenge level of the game. After evaluating different approaches, chose the Strategy Pattern to allow flexible and scalable difficulty configuration for both the dealer and AI players. This method enabled a clean separation of behaviors across difficulty levels.
Next, assigned the Strategy pattern's components to the appropriate classes:

- Context: GameManager, which stores the current strategy.
- Strategy Interface: DifficultyStrategy, defining common behavior.
- Concrete Strategies: EasyStrategy, MediumStrategy, and HardStrategy—each modifying dealer decisions and card logic.
- The selection logic was triggered via the BlackJackMenu, where the player chooses the difficulty using a dropdown or buttons before starting the game.

Initially, the dealer behavior was hardcoded (hit until 17). this was refactored using the Strategy Pattern.Then, a setDifficultyStrategy() method was introduced in GameManager, allowing dynamic injection of the selected strategy. The dealer now adapts to the selected strategy when playTurn() is called. After,, difficulty also started influencing the AI Players' behavior.




Once the difficulty system was fully integrated, we validated it by simulating rounds at each difficulty level. In Easy, the dealer rarely hits beyond 16. In Medium, they follow classical rules. In Hard, the dealer uses an optimized draw strategy, and the AI players make probabilistic choices.
The use of the Strategy Pattern allowed us to encapsulate and isolate difficulty behaviors, adhering to solid OOP principles such as the Open/Closed Principle (new strategies can be added without changing existing logic). The player now experiences a progressively challenging Blackjack game that feels fair and adaptive.




**UserStory# : As a player, I would like to see nicer backgrounds, maybe hear some songs, see images on cards.**

First of all, To enhance the visual polish of the game, Emre implemented dynamic custom backgrounds for the cards shown in the GUI. Haroun and Hanna implemented the general background, songs and effects. This feature adds immersion and helps differentiate normal and special card types like blackjack_bomb, split_ace and joker_wild.

### Technical Implementation for Cards

Each “Card” object in the model includes a “CardType”, which can be standard or one of the special cards. Based on this type, the correct image path is dynamically resolved in the GUI. The logic for loading and scaling images is encapsulated in the “createCardPanel(Card card)” method in “BlackjackGUI”.

### Synchronization Across Clients (Multiplayer)

In multiplayer mode, the server ensures that all clients render the correct card visuals, audios and songs by broadcasting a “GameStateUpdate” object. This object includes all cards with their exact types, ensuring each client can load and display the correct image.

**Determine Image Path**

- Uses switch “(card.getType())” logic.
- If it's a special card: load from "resources/img/special_card.png".
- If it's a regular card: construct path like "resources/img/number_of_suits.png".

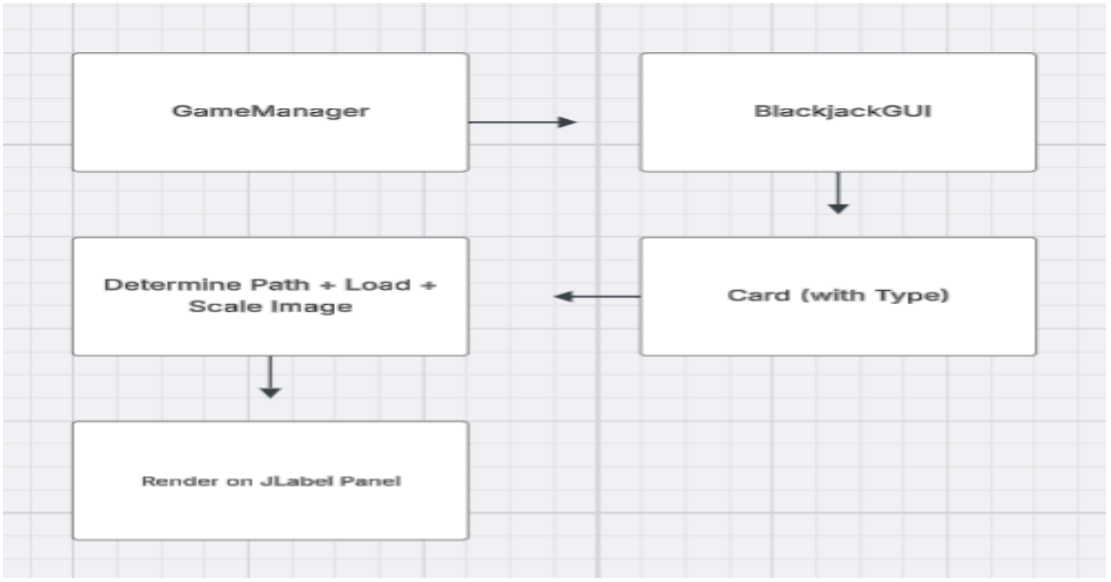
**Audio Integration**

The audio system is designed to enhance player immersion through background music and sound effects during gameplay. It adds atmosphere and feedback to user interactions like placing bets, hitting, standing, or game win/loss events.

**Volume Control:**

- The volume is a float value between 0.0 (mute) and 1.0 (max).

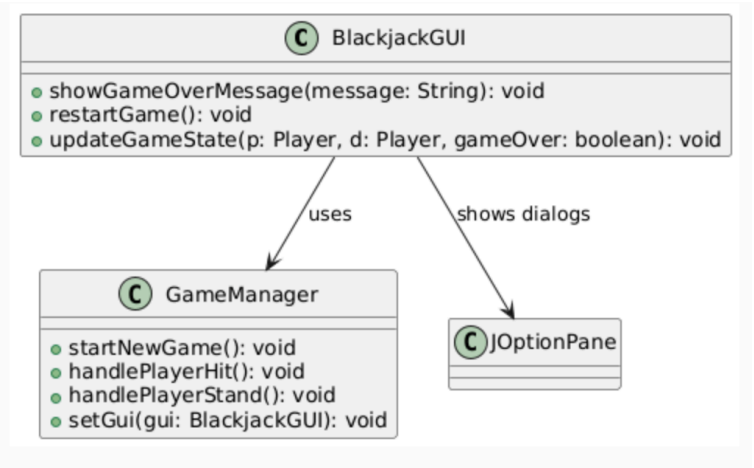
- Users can adjust the volume in real-time from the “Pause Menu” via a slider, which calls “AudioManager.setVolume(float)”.
- All audio operations are wrapped in “try-catch” blocks to prevent crashes due to missing files or incompatible formats.

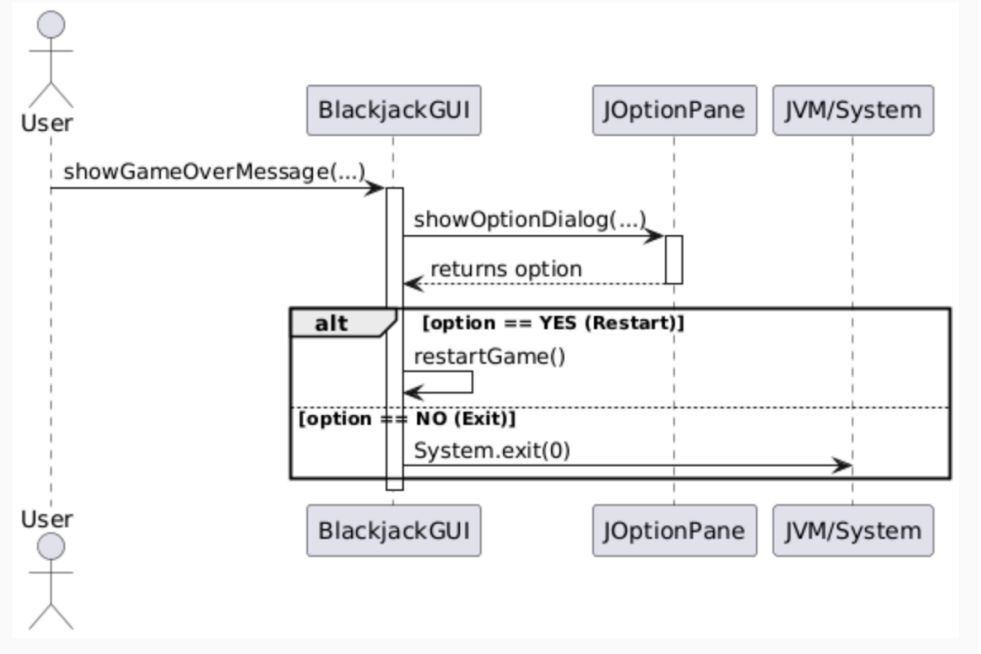



## User story #40: As a user, I want to be able to exit the game, aka terminate

Design Evolution

- Sprint 1 (Terminal version):
  - Early prototype ran in the console; to quit the game the user pressed a specific key (e.g. “Q”), and the main loop checked for this input and called System.exit(0).
  - This was a quick-and-dirty approach: game-over logic and termination were both handled in Main.java’s loop.
- Sprint 2 (GUI integration):
  - When the project moved into Swing, quitting via a menu option or button made sense.
  - In BlackjackGUI the “Exit” choice was incorporated into the end-of-game dialog (showGameOverMessage).
  - Rather than sprinkle System.exit(0) calls around, all termination points now funnel through one method: if the user clicks “Exit” on the dialog, System.exit(0) is invoked.






Explanation of Implementation:

1. Single Exit Point
- By centralizing termination in showGameOverMessage, the code avoids duplicated System.exit calls and keeps all end-of-game cleanup in one place.
2. User Flow
- After each round completes, GameManager notifies the GUI, which then calls showGameOverMessage(String message).
- The user sees a modal JOptionPane asking “Would you like to start a new game?” with “Restart Game” and “Exit” options.
3. Termination
- If “Exit” is chosen, showGameOverMessage executes: System.exit(0); which cleanly shuts down the JVM and closes the application window.
4. Restart Alternative
- If the user selects “Restart Game”, the same method calls restartGame(), which reinitializes the game state and balances without terminating the application.

# 3.4. Design Patterns

**Memento**

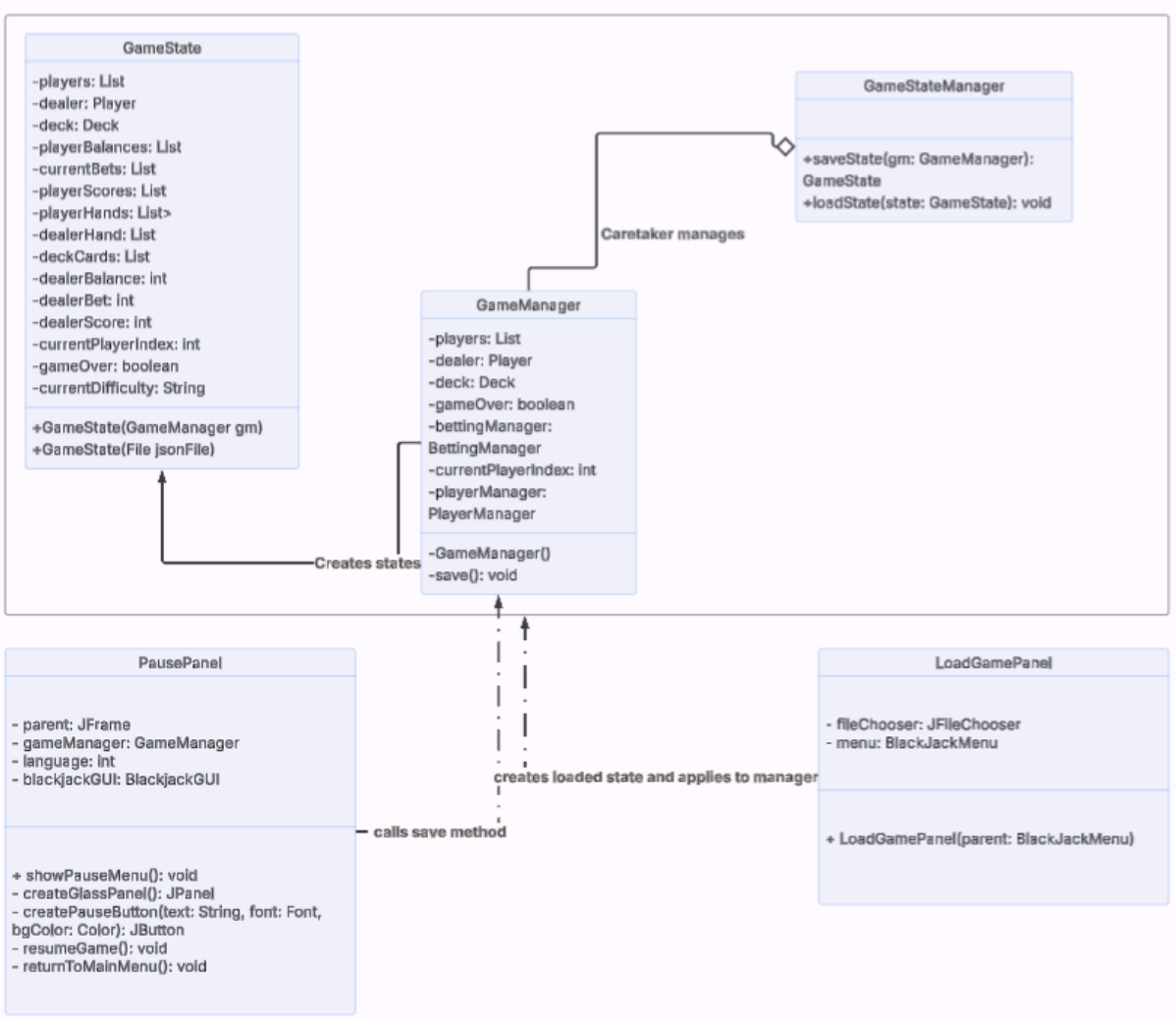
The Memento design pattern is a software engineering design pattern that allows an object to be saved and restored (or “loaded”) without revealing the internal implementation of the object's internal details. The implementation involves 3 main components:

**The originator :** The originator class is the object whose state we want to save and restore.

**The Memento :**This class stores the necessary information regarding the originator class. Its constructors are populated with “Screenshots” of game data that will be saved or loaded in.

**The CareTaker :** CareTaker classes save Mementos to files or take in a file in order to be processed and then applied by the Originator.

In our project we used the Memento to facilitate the Saving and Loading of .JSON files, as described above in User Story #53. Below we demonstrate the UML class diagram representation of how these objects interact with one another to perform Saving and Loading features.



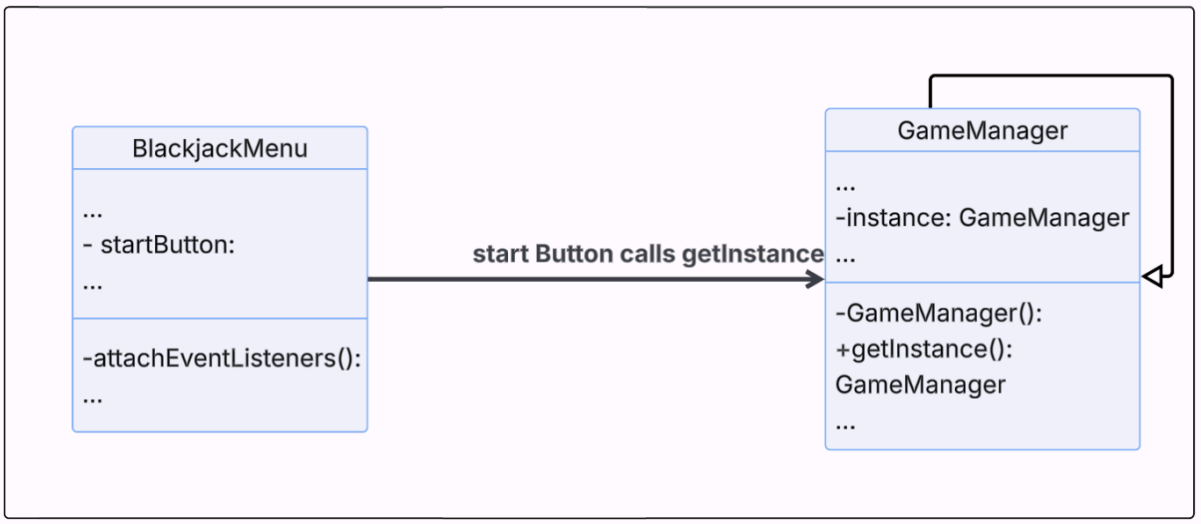

**Singleton**

The Singleton design pattern is a creational design pattern used to create an object of a class that has one and only one instance. These are often employed in cases where only one instance of a class is needed to perform tasks such as controlling data. In our case, we employ the Singleton design pattern to create our **GameManager**, **AchievemntsManager** and **AudioManager,** among others, which are responsible for the running of our BlackJack 2.0 game and do not require more than one occurrence.

To do this we create a private constructor and calle it from our public static method getInstance, which if an instance of the respective Manager exists, it is returned. If an instance does not exist, a call is made to the private constructor and a new Manager object is created.

This helps keep code clean and compact, while also restricting the ability to create unnecessary objects in our code base.

An example of this pattern can be seen below. When we start a game, our BlackJackMenu calls the GameManagers *getInstance()* method. Since this is the start of the game, a GameManager object will not yet exist. Therefore the method will provide one and the game will start.

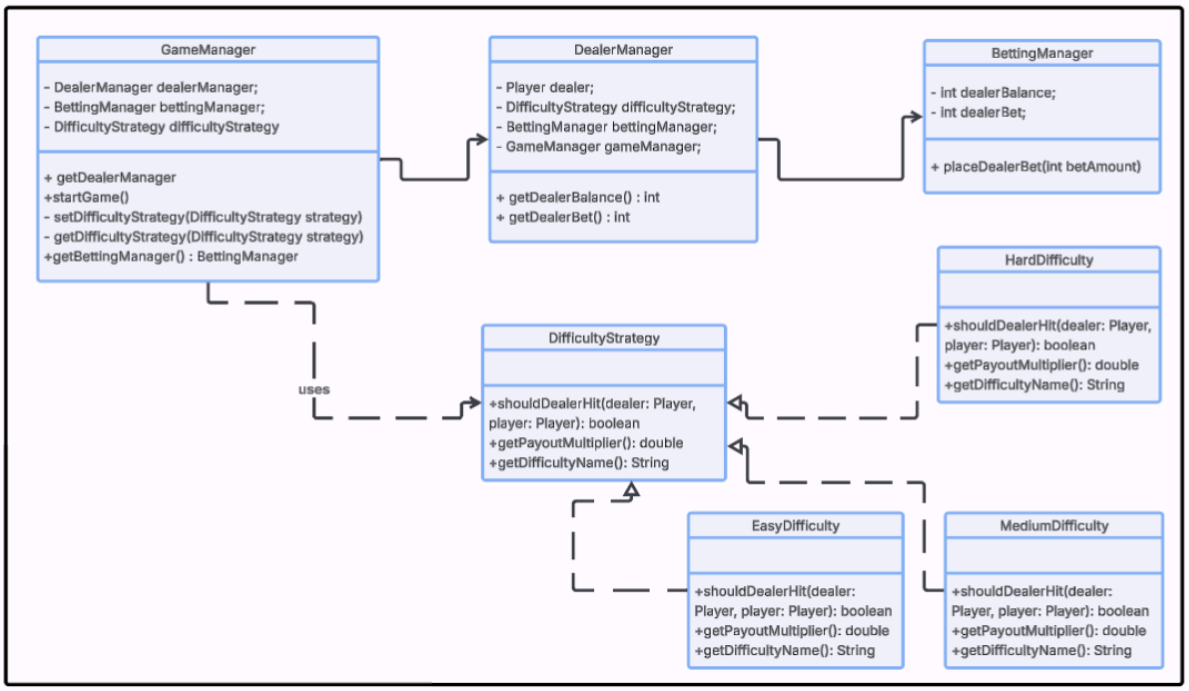



**Strategy**

Our project uses a Strategy design pattern to provide the Difficulty system, allowing the player to play against varying levels of Dealer behaviours and dish out diverse Payouts. This design pattern allows us to select an algorithm's behaviour at runtime. Furthermore we have implemented a system that will allow us to update the algorithm used by the player if they decide to change the difficulty level in the options.

To do this a DealerStrategy interface was defined, providing all its implementations the needed methods to handle whether the dealer should perform an action and calculate the bet it should place. Next the concrete difficulty classes were written **EasyDifficulty, MediumDifficulty** and **HardDifficulty**, each with different values for their respective calculations. These are then used by our "Context" class **(GameManger)** to create the **DealerManager** which handles the dealer actions, such as whether to hit or stand on a given hand and call to the **BettingManager** when it's time to place a bet.

The use of this Strategy allows our code to adhere to the Open/Closed principle: Open for extension, Closed for modification. It also keeps code clean removing unnecessary if statements.




# 4. Product & Code

Details of the repository, code structure, user manual, executable, Javadoc, test coverage, and snapshots.

## 4.1 Repository

The complete source code and documentation for the project is hosted on GitHub:

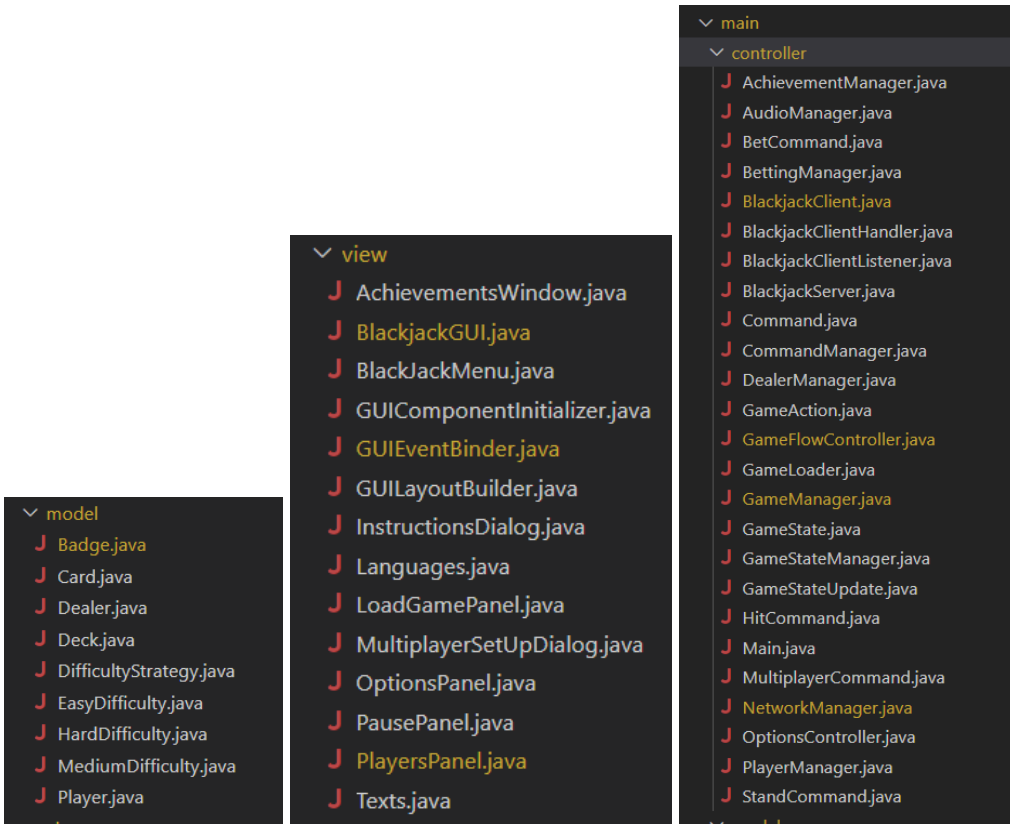
**GitHub Repository – The Boys Are Jack In Town**

The repository includes:

- Java source files
- GUI assets (images, icons, sound effects, background music)
- Executable JAR file
- Javadoc HTML documentation
- Agile documents (user stories, inception, sprint reviews)
- Screenshots of the final GUI are included in the README and User Manual.

The project is open source and organized to allow easy navigation and extension by future developers.

## 4.2 Code Structure




The codebase is organized following a more or less clean **Model-View-Controller (MVC)** architecture. Each major component of the game is clearly separated to improve readability, modularity, and maintainability:

### Folder & Package Structure

- **model/** – Contains core classes related to game logic and state management, such as GameState, Player, Dealer, Deck, and Card.
- **view/** – Holds all GUI components, including BlackjackGUI, MainMenu, AchievementsWindow, and popups for betting.
- **controller/** – Manages user interactions and game flow logic, including GameFlowController, PlayerManager, DealerManager, and OptionsController.

### Code Quality and Documentation

- Each class is defined in its own file with a clear, descriptive name.
- All public classes and methods include properly formatted **JavaDoc comments**.
- Naming conventions follow standard Java best practices.
- JavaDocs are manually generated.

## Modularity and Reusability

- GUI and model components are decoupled; views interact only through controllers.

- Feature-specific logic (e.g., achievements, audio, networking) is encapsulated in dedicated packages or classes.

## Final Notes

- The final version of the code compiles without any errors.
- Dependencies are managed internally, and the code is exportable as a single executable JAR.

## Citation:

This project was developed with the assistance of OpenAI's ChatGPT (GPT-4), an AI language model used to support software design, implementation, debugging, and documentation generation during the development process.

OpenAI. (2025). ChatGPT (GPT-4) [Large language model]. https://openai.com/chatgpt