

Requirement Analysis Document -

Group 16

1 Introduction

The goal of this project is to develop a computer-based version of the classic card game UNO by Mattel. The primary focus is to preserve the core gameplay mechanics while enabling a multiplayer experience. Additionally, the design will allow for adaptability, making it possible to support various versions of UNO by utilizing object-oriented principles.

2 GUI sketches

The GUI is designed to be visually appealing and easy for players to navigate and interact with. We want it to be simple to understand the state of the game and make moves. It consists of a start view, a lobby view and a game view. The implementation will be in JavaFX. We have a figma prototype to display our vision.

2.1 Start View

The start view, our main menu, serves as the starting point. In the middle of the screen there's a large play button, and a question mark button for instructions and rules. Pressing play switches the player over to the lobby.



2.2 Lobby View

The Lobby View handles multiplayer setup, allowing players to join a room and see who else is in the game. When enough players have joined, pressing the play button starts the game.



2.3 Game View

Once the game begins, the Game View becomes the main interface, showing the current game state, player hands, and actions. In the central area a discard pile and draw pile is displayed. Only the player's cards are visible. There is also going to be a turn indicator and animations for example a hover animation when choosing cards to play. Additionally, a button will be provided to announce when a player has UNO, one card left in their hand.



3 User stories

We have three epics that address core aspects of the game. One for the model, view and multiplayer support.

The goal of the epic centered around the model is to ensure the game mechanics function according to the rules of UNO. The view epic focuses on creating a visually engaging and intuitive interface for interacting with the game. Lastly, the multiplayer epic aims to enable players to connect and enjoy UNO with friends in a multiplayer environment.

[21] - EPIC - Model

2 cards

As a player:

I want to be able to play cards, draw them from a deck into my hand, all according to the rules of the game Uno, with the game preventing me from making illegal moves.

Because:

This lets me play the game.

Acceptance criteria:

- Card class and card factory
- Deck class and deck factory
- Player class with a hand of cards, able to add and remove cards from hand.
- A rules class, able to interpret special cards, when a player must draw cards and what cards are able to be played when.

[13] - EPIC - View

2 cards

As a player:

I want to be able to have a visually pleasing way to interact with and view the game.
So that:

The game is more fun to play.

Acceptance criteria:

- The `GameView` class should display the current state of the game.
- The `CardView` class should ensure a card can be displayed clearly with its value/action and color.
- The `LobbyView` class should display the lobby and allow players to join the game.
- The `LobbyWaitingView` class should display the waiting screen, showing the players who are currently waiting for the game to start.
- The `StartView` class should display the main menu.

[21] - EPIC - Multiplayer Support

2 cards

As a player:

I want to be able to play with other people.

Because:

It is more fun to play with friends

Acceptance criteria:

- Host the game and manage player connections.
- Handle connection setup between the server and clients.
- Connect to the server to participate in the game.
- Allow users to send actions to the server.

The epics are divided into user stories that focus on specific features or functionality that needs to be implemented. For example, the model epic is divided into 5 stories. There is one for managing the deck and one for rules when playing cards, shown below.

[2] - Create and Manage the Deck

2 cards

As a player:

I want the game to have a deck of cards that I can draw from,
So that:
I can play the game with a consistent set of cards.

Acceptance criteria:

- Implement a `Deck` class that holds a collection of cards.
- Implement a `DeckFactory` to create the deck at the start of a game.
- Allow players to draw cards from the deck.
- A method to shuffle the deck.

[5] - Basic Rules for Card Play

2 cards

As a player:

I want the game to prevent me from playing invalid cards,
So that:
The game follows Uno rules and is fair.

Acceptance criteria:

- Implement a `Rules` class that validates if a card can be played based on the current card on the discard pile.
- Provide clear feedback if an invalid move is attempted.

4 Stages/DoD

Our team follows a structured approach to complete user stories, divided into five stages:

1. Stories: Everything we want to implement
2. Backlog: higher priority stories we plan to start implementing
3. In progress: working on the implementation.
4. In review: reviewing and testing the finished implementation.
5. Completed: Stories are marked as complete once acceptance criteria are met, and the implementation has been tested either by users or by passing all automated/manual tests

We consider a user story done when all of the acceptance criteria has been met and the implementation passes all tests, either by unit tests or by a user testing by playing the game.

To keep track of our process we use a kanban board, we estimate required time and complexity using the Fibonacci sequence and colour the cards to show urgency/importance with red being the highest priority.

Here is a link to the rest of the user stories:

<https://app.milanote.com/1T9muel5vtnB98/oopp-game?p=AGLautC3VrV>

We use MilaNote to track them, you might have to create an account for access.

5 Domain model

The domain model consists of three entities: Deck, Player and Game. The Deck contains a collection of cards, the Player holds a hand of cards, and the Game manages the overall gameplay, including player interactions and the state of the game.

There is a value object called Card, which represents the individual playing cards in the game.

Four service objects, GameLogic, PlayerHandler, DeckHandler and Model, are responsible for handling the operational logic of the game. The Model is also responsible for initializing the game.

Two factories, CardFactory and DeckFactory, are responsible for creating instances of cards and the deck.

