Final Project

Art Dealer Simulation

Software Engineering

Professor Fadi Wedyan

Lewis University

July 7, 2025

**Table of Contents**

[Introduction 3](#_Toc202786120)

[Objectives: 3](#_Toc202786121)

[Game Logic and Patterns 3](#_Toc202786122)

[User Interface and Interactivity 5](#_Toc202786123)

[Process Model 6](#_Toc202786124)

[Use Cases 8](#_Toc202786125)

[Use Case 1: Start a New Game 8](#_Toc202786126)

[Use Case 2: Select Cards 8](#_Toc202786127)

[Use Case 3: Submit Cards for Dealer Evaluation 9](#_Toc202786128)

[Use Case 4: Guess the Dealer's Pattern 10](#_Toc202786129)

[Use Case 5: View Feedback on Card Selection 10](#_Toc202786130)

[Use Case 6: Restart the Game 11](#_Toc202786131)

[UML Model 12](#_Toc202786132)

[Use Case Diagram: 12](#_Toc202786133)

[Deployment Diagram: 13](#_Toc202786134)

[Class Diagram: 13](#_Toc202786135)

[State Diagram: 16](#_Toc202786136)

[Activity Diagram: 17](#_Toc202786137)

[Customer Journey Map 18](#_Toc202786138)

[Persona 19](#_Toc202786139)

[Testing Strategies 21](#_Toc202786140)

[Scope of Testing 21](#_Toc202786141)

[Testing Types 21](#_Toc202786142)

[Functional Testing 21](#_Toc202786143)

[Non-Functional Testing 22](#_Toc202786144)

[User Acceptance Testing (UAT) 22](#_Toc202786145)

# Introduction

The Art Dealer Game is a Python-based simulation that helps kids in grades three through five practise their pattern recognition skills. As the owner of a gallery, it is your job to find out how often the software salesperson buys things. You need to make sets of four cards using a conventional 52-card deck as a guide.

## Objectives:

The primary objective of the player is twofold:

* Lay out four cards that the art dealer will purchase.
* Correctly identify the pattern the dealer is following from a list of possible patterns.

The game uses two Python libraries: Tkinter for the user interface and pygame for sound. It's a pleasant and interesting approach for students to learn. Players can look at their stats and get tips on how to improve after every game. The game ends when someone tries to guess or find a matching pattern.

You might print patterns of logic and math on the playing cards. A range of numbers, a maximum level for sums, or certain combinations of card suits are all examples of these kinds of limits. Using these patterns in the classroom makes it easier for students to be involved, think critically, and work together to solve problems.

## Game Logic and Patterns

The Art Dealer Game is driven by the logic of card patterns, specifically designed for students in grades 3-5. The game starts by creating a shuffled deck of 52 standard playing cards, consisting of four suits (Hearts, Diamonds, Clubs, and Spades) and values ranging from Ace (A) to King (K). At each turn, the player selects four cards from this deck and attempts to match a predefined pattern that the software (acting as the art dealer) is following.

The patterns used in the game are as follows:

1. All Even Numbers: The cards laid out must have even values (e.g., 2, 4, 6, 8, 10).
2. All Odd Numbers: The cards must have odd values (e.g., 1, 3, 5, 7, 9).
3. Sum of Values Greater than 20: The total value of the four cards must exceed 20 (with face cards counted as 10).
4. Two Red and Two Black Cards: The hand must consist of exactly two red-suited cards (Hearts, Diamonds) and two black-suited cards (Clubs, Spades).
5. All Cards of Different Suits: Each card must belong to a different suit (one from each of Hearts, Diamonds, Clubs, and Spades).
6. Two Picture Cards (Jack, Queen, King): The player’s hand must contain exactly two face cards (J, Q, K).
7. Prime Numbers and a Face Card: The hand must include at least one prime number (2, 3, 5, 7, 11) and at least one face card (J, Q, K).
8. Two Cards Add Up to 10: Two of the cards in the player’s hand must sum to 10 (e.g., 4 and 6, 3 and 7).
9. All Cards Greater than 5: The values of all cards must be greater than 5 (e.g., 6, 7, 8, 9, 10).
10. One Card of Each Rank (1-10): The hand must consist of cards that represent unique values from 1 to 10.

These patterns are checked using lambda functions that evaluate the selected cards. The player continues guessing until they correctly match the art dealer’s pattern or exhaust their guesses. The game provides immediate feedback after each round, helping players adjust their strategy for the next attempt.

## User Interface and Interactivity

The game is designed using the Tkinter library for a simple yet interactive graphical user interface (GUI). The GUI allows players to interact with the game smoothly by selecting cards and making guesses about the dealer's pattern. Here are the key features of the user interface:

* **Card Selection Area:** The player chooses four cards at random from the digital deck that is shown on the screen. This window shows all of the cards, so the player can quickly choose which ones to utilize.
* **Pattern Guess Area:** After getting their cards, the player can use a pattern-based database to guess what the dealer will do next. The best projections decide who wins after each round.
* **Feedback and Results:** The game will tell you if your hand is the same as the dealer's after each round. If the player's patterns match exactly, they will see a message saying "victory" and be able to celebrate with music or lights.
* **Game Flow Control:** You can use the UI to start games, guess what will happen in them, and find new games to play. Players can keep trying different combinations until they find one that works best for them.

The game's design makes it easy and quick to fix problems. The pygame package lets you add sound effects to the game, which makes it a lot more fun for kids. This fun part of the game will keep kids interested while also helping them get better at maths and recognising patterns.

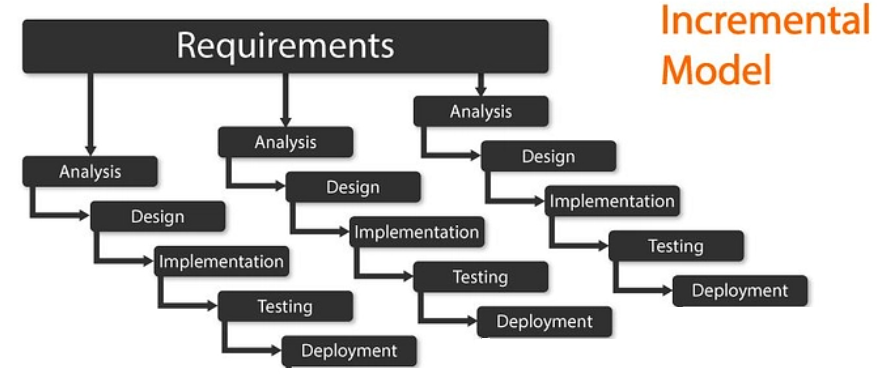
.

# Process Model

The Incremental Process Model breaks down an existing system into its parts before adding new features. With each upgrade, you get a full edition of the program, and with each update, the app grows better. If you want to learn how to build more complex systems, you need to start thinking this way.

Steps of the Incremental Process Model in the Art Dealer Game:

1. **Requirements Gathering:**
2. It's really important to plan ahead. The main goal of the Art Dealer Game was to create a card game in Python that would help students in grades three through five learn how to find patterns in math and logic.
3. **Design:**
4. They thought long and hard about a lot of different patterns to make sure the user experience was easy to understand. The framework was built up piece by piece, starting with the most basic tasks like choosing and moving cards around. This was done to provide way for pattern recognition logic.
5. **Implementation:**
6. The first step in making the game was to put the playing cards together and mix them up. Later versions of the game made it harder by introducing tools that could find patterns. These algorithms looked for patterns in the distribution and sum of even and odd numbers.
7. **Testing and Feedback:**
   1. With each increment, testing was likely done to ensure that the game logic was working correctly. For example, after implementing patterns like "All Even Numbers" or "Two Red and Two Black Cards," those specific features would have been tested before moving on to implement the next set of patterns.
8. **Deployment and Refinement:**
   1. Once all the patterns and game logic were implemented, the entire game was tested and refined. The user interface was likely adjusted and improved in this phase, making the game more user-friendly and engaging for students.



# Use Cases

## ****Use Case 1: Start a New Game****

|  |  |
| --- | --- |
| **Use Case ID** | UC-1 |
| **Use Case Name** | Start a New Game |
| **Primary Actor** | Student (Player) |
| **Goal** | Start a new game session |
| **Preconditions** | The game is installed and running |
| **Main Success Scenario** | 1. The player opens the game application. 2. The game interface loads with options to start a new game. 3. The player clicks the "Start New Game" button. 4. The game shuffles the deck of 52 cards. 5. A new game session begins. |
| **Postconditions** | A new game session starts with a shuffled deck of cards. |
| **Extensions** | None |

## ****Use Case 2: Select Cards****

|  |  |
| --- | --- |
| **Use Case ID** | UC-2 |
| **Use Case Name** | Select Cards |
| **Primary Actor** | Student (Player) |
| **Goal** | Select four cards to lay out |
| **Preconditions** | The game session is active, and the deck is shuffled |
| **Main Success Scenario** | 1. The player views the available cards. 2. The player selects four cards. 3. The selected cards are displayed in the player’s hand. |
| **Postconditions** | The selected cards are displayed as the player’s hand. |
| **Extensions** | If less than four cards are selected, the game prompts to select more cards. |

## ****Use Case 3: Submit Cards for Dealer Evaluation****

|  |  |
| --- | --- |
| **Use Case ID** | UC-3 |
| **Use Case Name** | Submit Cards for Dealer Evaluation |
| **Primary Actor** | Student (Player) |
| **Goal** | Submit selected cards for evaluation |
| **Preconditions** | The player has selected four cards |
| **Main Success Scenario** | 1. The player clicks the "Submit" button. 2. The game checks the selected cards against predefined patterns. 3. Dealer accepts or rejects the cards. 4. Feedback is given based on the result. |
| **Postconditions** | The player receives feedback on whether the cards match the dealer’s pattern. |
| **Extensions** | The player can select new cards if the cards do not match. |

## ****Use Case 4: Guess the Dealer's Pattern****

|  |  |
| --- | --- |
| **Use Case ID** | UC-4 |
| **Use Case Name** | Guess the Dealer's Pattern |
| **Primary Actor** | Student (Player) |
| **Goal** | Guess the correct pattern from the dealer |
| **Preconditions** | The player has submitted cards for evaluation and received feedback |
| **Main Success Scenario** | 1. The player guesses the dealer's pattern from a provided list. 2. If the guess is correct, the player wins. 3. If the guess is incorrect, the player is prompted to guess again (up to 3 times). |
| **Postconditions** | The game ends or prompts the player to make another attempt. |
| **Extensions** | After 3 incorrect guesses, the game stops and offers a new attempt. |

## ****Use Case 5: View Feedback on Card Selection****

|  |  |
| --- | --- |
| **Use Case ID** | UC-5 |
| **Use Case Name** | View Feedback on Card Selection |
| **Primary Actor** | Student (Player) |
| **Goal** | Receive feedback based on the selected cards |
| **Preconditions** | The player has submitted four cards for dealer evaluation |
| **Main Success Scenario** | 1. The game checks the selected cards against the dealer's pattern. 2. The player receives feedback on whether the cards match the pattern. 3. If the cards match, the game indicates success; if not, the player is prompted to try again. |
| **Postconditions** | The player receives feedback on the current selection, allowing them to adjust their next move. |
| **Extensions** | None. |

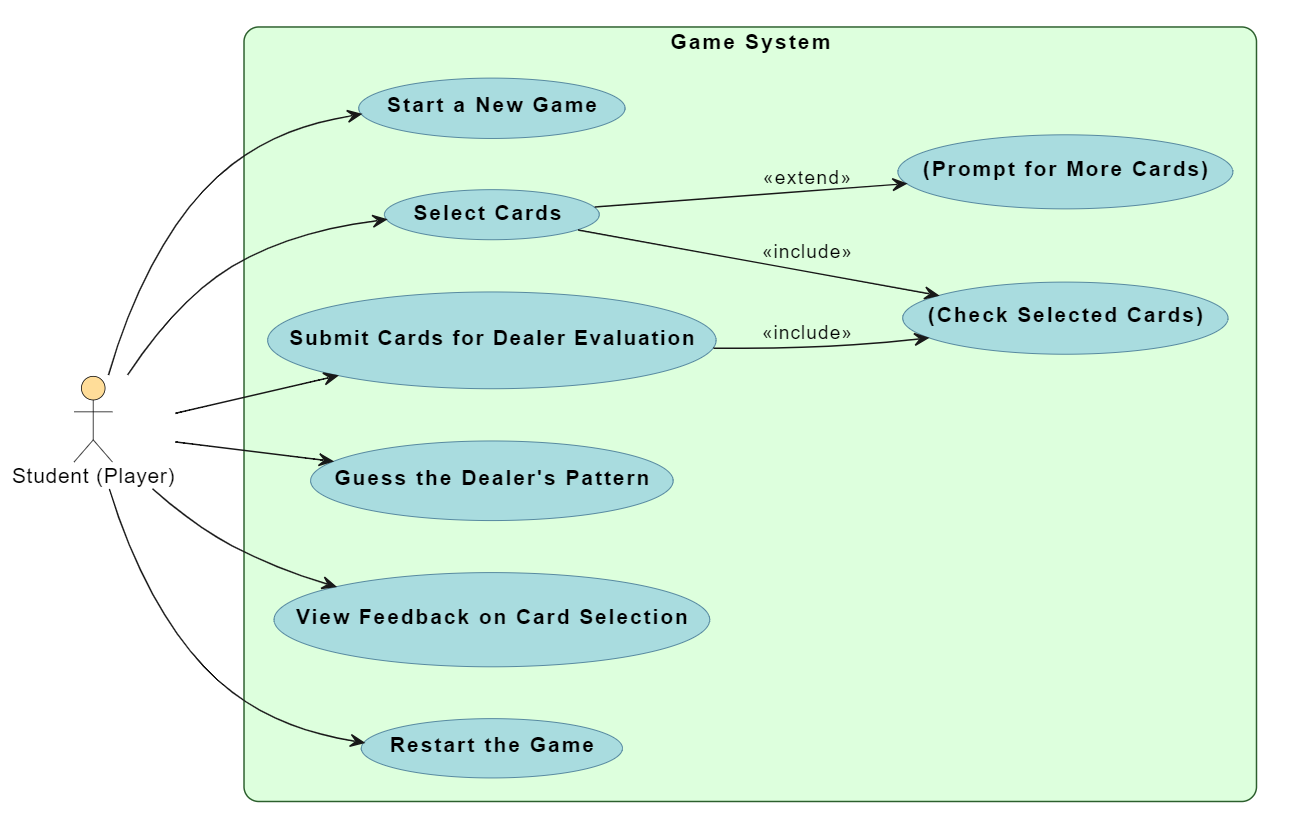
## ****Use Case 6: Restart the Game****

|  |  |
| --- | --- |
| **Use Case ID** | UC-6 |
| **Use Case Name** | Restart the Game |
| **Primary Actor** | Student (Player) |
| **Goal** | Restart the game after a round has ended |
| **Preconditions** | The player has completed or ended a game session |
| **Main Success Scenario** | 1. The player clicks the "Restart Game" button. 2. The game reshuffles the deck. 3. A new game session begins with the same difficulty level. |
| **Postconditions** | The game restarts with a fresh deck and a new pattern for the dealer. |
| **Extensions** | The player can choose to restart immediately after finishing the game or end the session. |

# UML Model

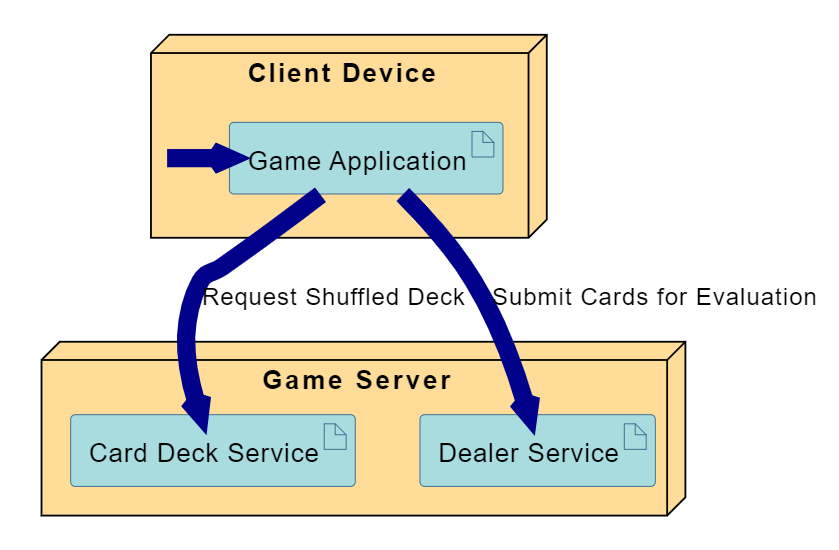
## Use Case Diagram:

A Use Case Diagram visually represents the interactions between a user (actor) and a system to capture its functionalities and behaviors. It identifies the different ways a user might interact with the system to achieve specific goals, such as starting a game, selecting options, or submitting actions. The diagram consists of actors (e.g., "Student (Player)") who engage with use cases (e.g., "Start a New Game"), and it can include relationships like <<include>>, which shows mandatory steps in a use case, and <<extend>>, which highlights optional behaviors. By outlining these interactions, the Use Case Diagram provides a high-level overview of the system’s functionality and the user's role within it.



## Deployment Diagram:

A Deployment Diagram illustrates the physical architecture of a system, showing how software components are deployed across hardware nodes. In this case, the diagram depicts the interaction between the Client Device (representing the player's device) and the Game Server, where the key services for the game are hosted. The Game Application runs on the client, and communicates with the Card Deck Service (responsible for shuffling and managing the deck) and the Dealer Service (which evaluates the player's card selections) on the server. The arrows represent data flow, such as requests for shuffled decks or submitting cards for evaluation. This diagram helps visualize how different parts of the system interact and how the client communicates with server-side services during gameplay.



## Class Diagram:

This Class Diagram represents the core components of a card-based game system, where players interact with the game by selecting cards, submitting them for evaluation, and receiving feedback from a dealer. The diagram captures five primary classes: Player, Game, CardDeck, Dealer, and Card, detailing their attributes and behaviors as well as their relationships.

Player Class:

* Attributes: playerId, playerName, and selectedCards (a list of Card objects).
* Methods:
  + startNewGame() initializes a new game session.
  + selectCard() allows the player to choose a card.
  + submitCards() submits the selected cards for evaluation.
* Relationship: A Player plays exactly one Game and selects zero or more Cards.

Game Class:

* Attributes: gameId, currentPlayer (a reference to a Player), dealer (a reference to a Dealer), and deck (a reference to a CardDeck).
* Methods:
  + start() begins a new game.
  + restart() restarts an existing game.
  + evaluatePlayerCards() evaluates the player's selected cards.
* Relationships: The Game interacts with both the CardDeck and Dealer. The game is responsible for shuffling the deck and interacting with the dealer to evaluate the player's selected cards.

CardDeck Class:

* Attribute: cards, a list of Card objects representing the full deck.
* Methods:
  + shuffle() randomizes the order of cards in the deck.
  + dealCard() returns a card to the player.
* Relationship: The Game manages a CardDeck and calls on it to shuffle and deal cards.

Dealer Class:

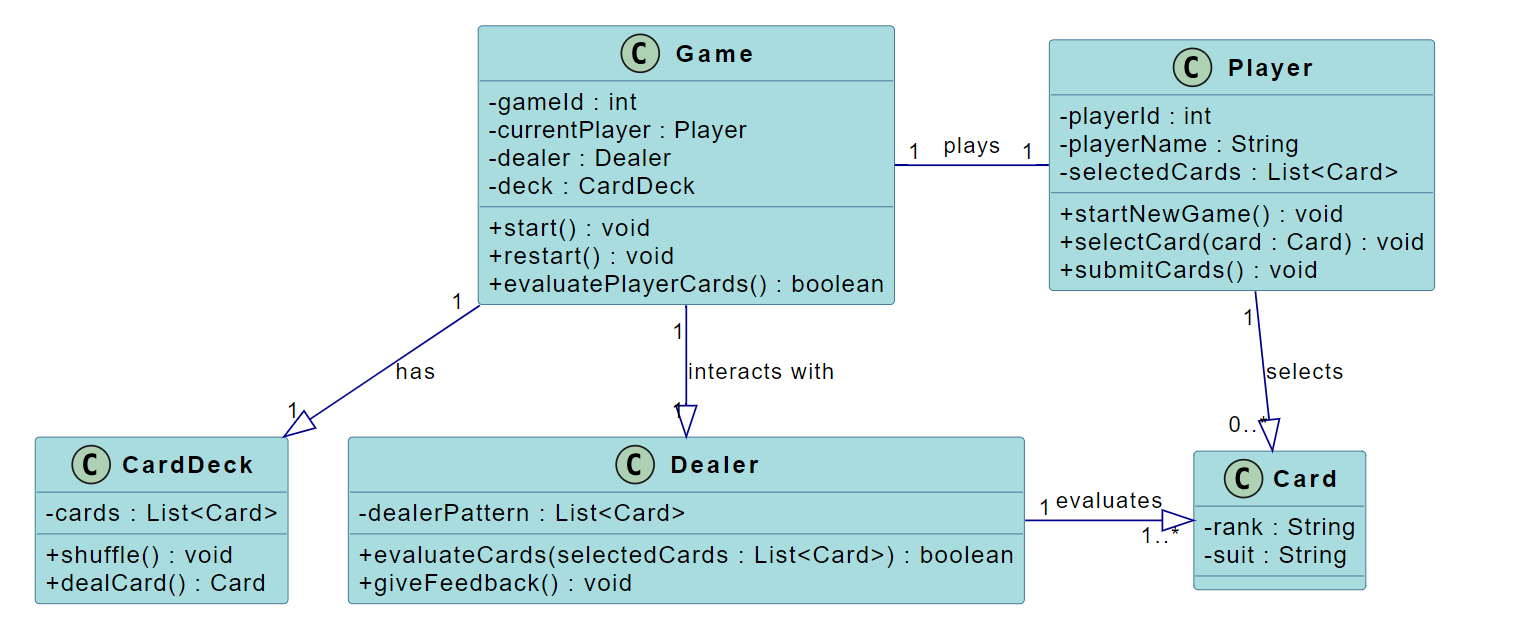
* Attribute: dealerPattern, a list of Card objects representing the pattern against which the player's cards are evaluated.
* Methods:
  + evaluateCards() checks whether the player's selected cards match the dealer's pattern.
  + giveFeedback() provides feedback based on the evaluation.
* Relationship: The Dealer evaluates the player's selected cards and interacts with the Game to process the results.

Card Class:

* Attributes: rank and suit, representing the value and type of a card (e.g., Ace of Spades).
* Relationship: The Player selects one or more Cards, and the Dealer evaluates these cards.

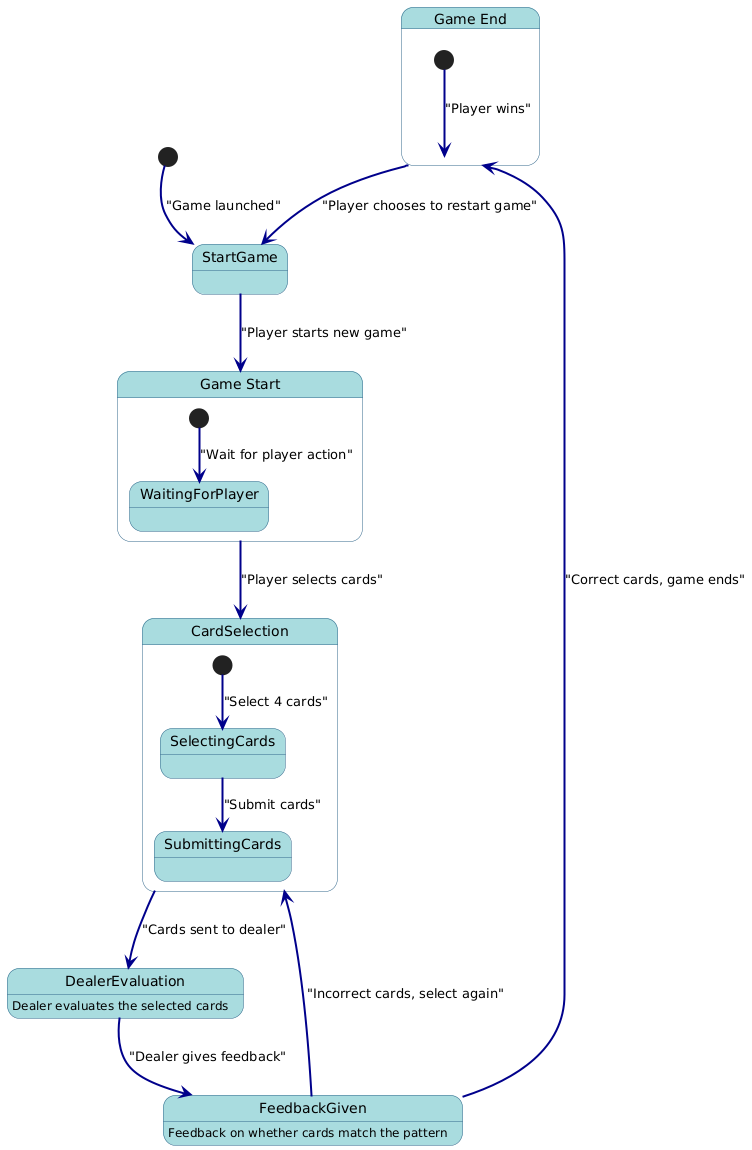
Relationships:

* Multiplicity:
  + A Player plays exactly one Game.
  + A Game interacts with one Dealer and manages one CardDeck.
  + A Player can select zero or more Cards (0...\*), while the Dealer evaluates one or more cards (1...\*).



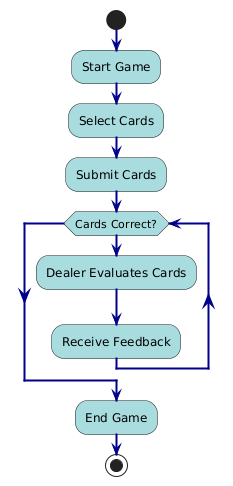
## State Diagram:

This State Diagram represents the flow of a card-based game. It starts with the game being launched (StartGame), transitioning to the Game Start state where the system waits for the player to take action. The player can then move to the Card Selection state to select and submit cards for dealer evaluation. The game transitions to the Dealer Evaluation state, where the dealer checks the player's cards. After the evaluation, feedback is provided in the FeedbackGiven state. If the cards are incorrect, the player returns to the Card Selection state; if correct, the game ends, reaching the Game End state. The player can choose to restart the game from there.



## Activity Diagram:

This Activity Diagram represents the flow of a card-based game where the player selects and submits cards for evaluation. The game starts with "Start Game", followed by the player selecting and submitting cards. A while loop checks if the submitted cards are correct. If the cards are incorrect, the loop repeats, allowing the player to resubmit cards until they are correct. Once the correct cards are submitted, the game ends with the "End Game" activity. The loop ensures continuous evaluation until the correct result is achieved.



# Customer Journey Map

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stage | Player Actions | Player Thoughts | Player Emotions | Touchpoints |
| Discovery | Finds the game (through app store, recommendation, etc.) | "This game looks interesting!" | Excited, Curious | App Store, Game Website, Ads |
| Installation | Installs the game on their device | "Let me see how it works." | Enthusiastic, Eager | Game download/install process |
| Game Start | Launches the game and starts a new session | "I hope this is fun and easy to play." | Curious, Hopeful | Game interface (main menu) |
| Card Selection | Selects cards to play in the game | "Which cards should I choose?" | Engaged, Focused | Game screen (card selection) |
| Submit Cards | Submits the selected cards for dealer evaluation | "Will these cards be accepted?" | Anticipation, Slightly Anxious | Submit button (game interface) |
| Dealer Feedback | Receives feedback on the card selection (correct/incorrect) | "Did I make the right choice?" | Excited (if correct), Frustrated (if wrong) | Dealer feedback screen |
| Game End | Game ends (either win or lose) | "That was fun! Should I play again?" | Happy (if win), Determined (if lost) | Game completion screen (result) |

# Persona

|  |  |
| --- | --- |
| Category | Details |
| Name | David Lee |
| Age | 32 |
| Occupation | Software Engineer |
| Location | Suburban Area, United Kingdom |
| Tech Savviness | High (frequent gamer, tech enthusiast) |
| Preferred Devices | PC, Smartphone |
| Hobbies | Gaming, coding, reading sci-fi, card tournaments |
| Background | David enjoys strategy and card games. He plays during short breaks and seeks mental challenges in his gameplay. |
| Goals | 1. Improve strategy2. Mental challenge3. Achieve milestones4. Quick, engaging play |
| Frustrations | 1. Games lacking depth2. Unclear feedback3. Unbalanced gameplay4. Long game sessions |
| Motivations | - Competition- Skill growth- Relaxation |
| Gaming Preferences | - Strategy and card games- Favorite games: Gwent, Civilization VI, PokerStars |
| Journey with Game | Finds games through gaming blogs, appreciates quick-start games with strategic depth, and enjoys replayability via rankings and challenges. |
| Quote | “I love games that challenge me intellectually and let me improve with every session.” |
| Interaction with Game | Prefers short, challenging sessions with feedback and progression (achievements and levels). |

# Testing Strategies

The primary objective of this testing strategy is to:

Ensure the game meets functional and non-functional requirements. Identify and resolve any defects in gameplay mechanics, user interface, and backend systems. Validate the game’s performance, security, and user experience across multiple platforms and devices.

## Scope of Testing

The scope of testing will include:

* Functional Testing: Testing the core functionalities of the game, including card selection, dealer evaluation, feedback mechanisms, and game progression.
* Non-functional Testing: Testing the performance, security, usability, and compatibility across various devices.
* User Acceptance Testing (UAT): Verifying that the game meets the expectations of end users.

## Testing Types

### Functional Testing

This type of testing will focus on validating each feature of the game.

* + Unit Testing: Verifying individual functions, such as shuffling cards, evaluating selected cards, and submitting feedback.
  + Integration Testing: Ensuring modules (e.g., card selection and dealer evaluation) work together smoothly.
  + End-to-End Testing: Testing the complete game flow from the start of a new game to card submission, feedback, and game conclusion.
  + Smoke Testing: Performing a basic test to verify that the game’s key functions (e.g., card selection, game start, dealer evaluation) work after each build.

### Non-Functional Testing

These tests validate the game's performance, scalability, and user experience.

* + Performance Testing: Ensuring the game runs smoothly under normal and peak loads (handling multiple players simultaneously without lag).
  + Security Testing: Checking the game for vulnerabilities such as unauthorized access, data security issues, and payment system flaws (if any).
  + Usability Testing: Validating the user interface for ease of use, accessibility, and intuitiveness.
  + Compatibility Testing: Ensuring the game works across different devices (smartphones, tablets, PC) and operating systems (iOS, Android, Windows).

### User Acceptance Testing (UAT)

* + End users or focus groups will test the game to ensure it meets their expectations in terms of gameplay, difficulty, and enjoyment.