Software Requirements Specification

for

Py-Uno

Version 1.0 approved

Prepared by Cruz Lopez & Melanie Garza

Texas State University Computer Science Program

February 29th, 2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Py-Uno	2/29	Initial Document	1.0

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document is crafted to outline the requirements and specifications for the development of an Uno game using Python. The primary objective of this document is to provide a clear and detailed blueprint for the game's design and functionality, serving as a guiding framework for the students involved in the project.

1.2 Problem Statement

Despite the popularity of Uno as a card game, existing digital versions often fall short in delivering a comprehensive experience that includes both engaging single-player modes with challenging AI and a seamless, intuitive multiplayer platform. Additionally, there is a lack of customizable features that allow players to modify game rules and aesthetics according to their preferences. This project aims to develop an Uno game using Pygame that addresses these gaps by providing an intuitive, cross-platform gaming experience that supports both single-player and multiplayer modes. It will offer customizable game elements to enhance engagement and meet the diverse needs of Uno players, leveraging modern software development tools and technologies to create a superior digital Uno game.

1.3 Document Conventions

This document adheres to the IEEE outline format to organize information into distinct sections and subsections, enhancing readability and comprehension of the Py-Uno application's functionalities. For typographical conventions, keywords are **highlighted** in **bold** to stand out, while specific font styles and sizes are designated for headings Times 18 pt for section headings, Times 14 pt for subsections) and body text (Arial 12 pt). Italicization is used for emphasis and to denote technical terms upon their first appearance.

Revisions and updates to this document, including any changes to the document conventions, are meticulously recorded in the Revision History table. This approach ensures that document evolution is transparent and easily traceable. The initial version of this document is 1.0, with subsequent updates increasing the version by 0.1 (e.g., 1.1, 1.2).

1.4 Intended Audience and Reading Suggestions

This Software Requirements Specification (SRS) document is designed for a diverse audience, including:

1. **Professors**: To evaluate the adherence to software engineering principles, the robustness of the design, and the completeness of the documentation.
2. **Possible Employers:** To assess the practical skills, innovation, and real-world applicability of the project.
3. Other Students: Who may use this document as a learning tool or for collaborative purposes.
4. **External Developers:** Who may fork this project will find detailed descriptions of the game's functional and non-functional requirements.
5. **Project Managers:** Can refer to the project scope and milestones to track progress and resource allocation.
6. **Testers:** Who wish to document bugs should focus on the 'User interface' and 'Functional Requirements' sections to understand the game mechanics and testing criteria.
7. **Possible Documentation Writers:** Will benefit from the comprehensive overview of game rules and mechanics to create user manuals and help guides.

The SRS is structured to cater to these varied interests. Professors and employers may focus on the 'Project Scope' and 'Design Rationale'. Other students might benefit from the 'General Overview' and 'Functional Requirements'.

Reading should commence from the 'General Description' for a comprehensive understanding, followed by sections relevant to the reader's specific interest or role in the project.

1.5 Product Scope

This requirements document delineates the scope of the Py-Uno application, developed as part of our capstone project for the Computer Science program at Texas State University. Py-Uno is a digital adaptation of the classic card game Uno, created using Python. This project aims not only to provide an engaging game experience but also to serve as a practical application of our programming, design, and project management skills learned through our coursework.

The key functions of the application include:

- Offering a single-player experience, allowing users to play solo and engage with the game mechanics.
- Enabling a multiplayer mode, facilitating social interaction and teamwork among peers.
- Providing customizable game settings, such as the number of rounds, to demonstrate our ability to create flexible software.
- Incorporating AI opponents with varying difficulty levels, showcasing our skills in AI and game strategy implementation.

As a capstone project, Py-Uno is intended to demonstrate our ability to conceptualize, design and implement a software solution that is both functional and user-friendly. This endeavor aligns with our academic goals of applying theoretical knowledge in a real-world scenario, enhancing our software development skills, and preparing for a professional career in the field of computer science. This project is also a reflection of our commitment to create software that is not only technically proficient but also enjoyable and accessible to a wide range of users.

1.6 References

This document references the following sources to provide a comprehensive background and framework for the Py-Uno project:

1.5.1 Technical Documentation:

- Python Documentation: Python
- Pygame Library Documentation: Pygame
- Pygwidgets Library Documentation: Pygwidgets
- Python Coding Style Guide: PEP 8 – Style Guide for Python Code.

1.5.2 External resources:

- Object Oriented Principles: "Object-Oriented Python" by Irv Kalb ISBN-13: 9781718502062. This book provides the object-oriented design approach used in the project.
- This SRS document is derivative of the Software Requirements Specification template created by Karl E. Wiegers Copyright © 1999. Permission was granted to use, modify, and distribute the template.

1.5.3 Project Proposal:

Initial Project Proposal: This project was first proposed by Cruz Lopez and Melanie Garza. The proposal outlines the initial concept and objectives for the Py-Uno game project. The proposal can be found here.

1.5.4 Game Rules Reference:

Uno Game Rules: The official rules for the Uno card game as owned by Mattel. The rules can be found here.

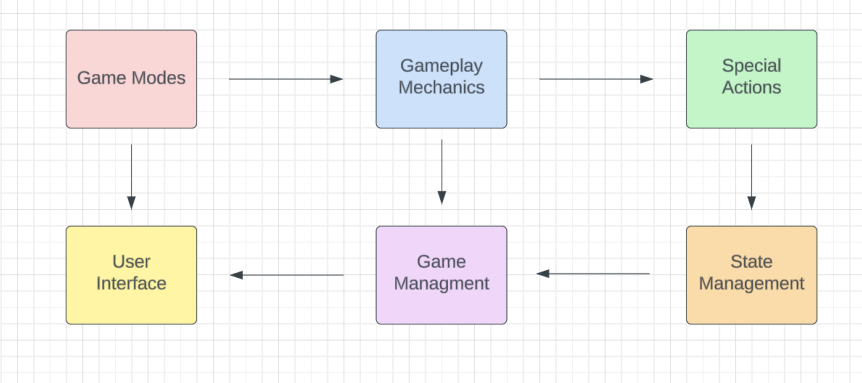
2. Overall Description

2.1 Product Perspective

This digital Uno game is a new, standalone product, designed to bring the traditional card game experience into the digital realm using Python. It incorporates object oriented programming in order to ensure a scalable and maintainable codebase. The game features both single-player against an AI and multiplayer, adhering to the classic Uno rules with added functionalities such as the automated turn management and a round-based system for extended play. The system is structured around key components: the User interface for player interaction, Game Logic to enforce rules and manage gameplay, AI for single-player mode, and a Networking component for multiplayer interactions.

2.2 Product Functions

- **Game Modes:**
  - Single-player mode against AI opponents
  - Multiplayer mode for playing with friends or other online players.
- **Gameplay Mechanics:**
  - Automated turn management to ensure smooth transitions between players
  - Card matching based on color, symbol, or number to progress in the game
  - Drawing mechanisms for when players cannot match the current top card of the discard pile
  - Round-based system allowing players to set and play a series of rounds, with an overall winner determined by the number of rounds won.
- **Special Actions:**
  - Utilization of special cards (skip, reverse, draw two, wild, and draw four) to influence gameplay
  - Ability to declare “Uno” when down to the last card
  - Feature to challenge players who fail to declare “Uno”
- **Game Management:**
  - Adherence to official Uno rules and incorporation of traditional card types
  - In-game timer to maintain pace and engagement
  - State management to track players’ hands, the deck, and the discard pile, including reshuffling the discard pile into a new deck when necessary
- **User Interface:**
  - Intuitive and interactive interface for game setup, in-game actions, and viewing game progress
  - Visual representation of players’ hands, the draw pile, and the discard pile



2.3 Operating Environment

2.3.1 Operating Systems:

- Windows 10
- Mac OS
- Linux

2.3.2 Software Requirements:

- Python 3
- Pygame library
- Pygwidgets library

2.3.3 Hardware requirements:

- 700MHz Processor
- 512 MB RAM
- 200 MB Storage

2.3.4 Network Requirements:

- Stable internet connection

2.4 Design and Implementation Constraints

Design and Implementation constraints include:

- **The Model View Controller (MVC):** This will be our first time applying the MVC pattern to an application. We expect a learning curve in the first sprint. The scope of the application will remain uncertain until its completion, so it's possible that we're adding unnecessary overhead if the application is too small. Regardless, applying an MVC pattern comes with a tight coupling between the view and the controller. If there is a need to change either the view or the controller, corresponding changes in the other will also be necessary. .

- **Agile Development Process:** The project will include at least three sprints, each preceded by sprint planning and followed by a sprint review. Each sprint will be two weeks in length. All tasks will be tracked in Jira and all code will require approval via pull request before being merged.

- **The Uno official rules:** Adherence to the Uno official rules will guide the development of game logic and features.

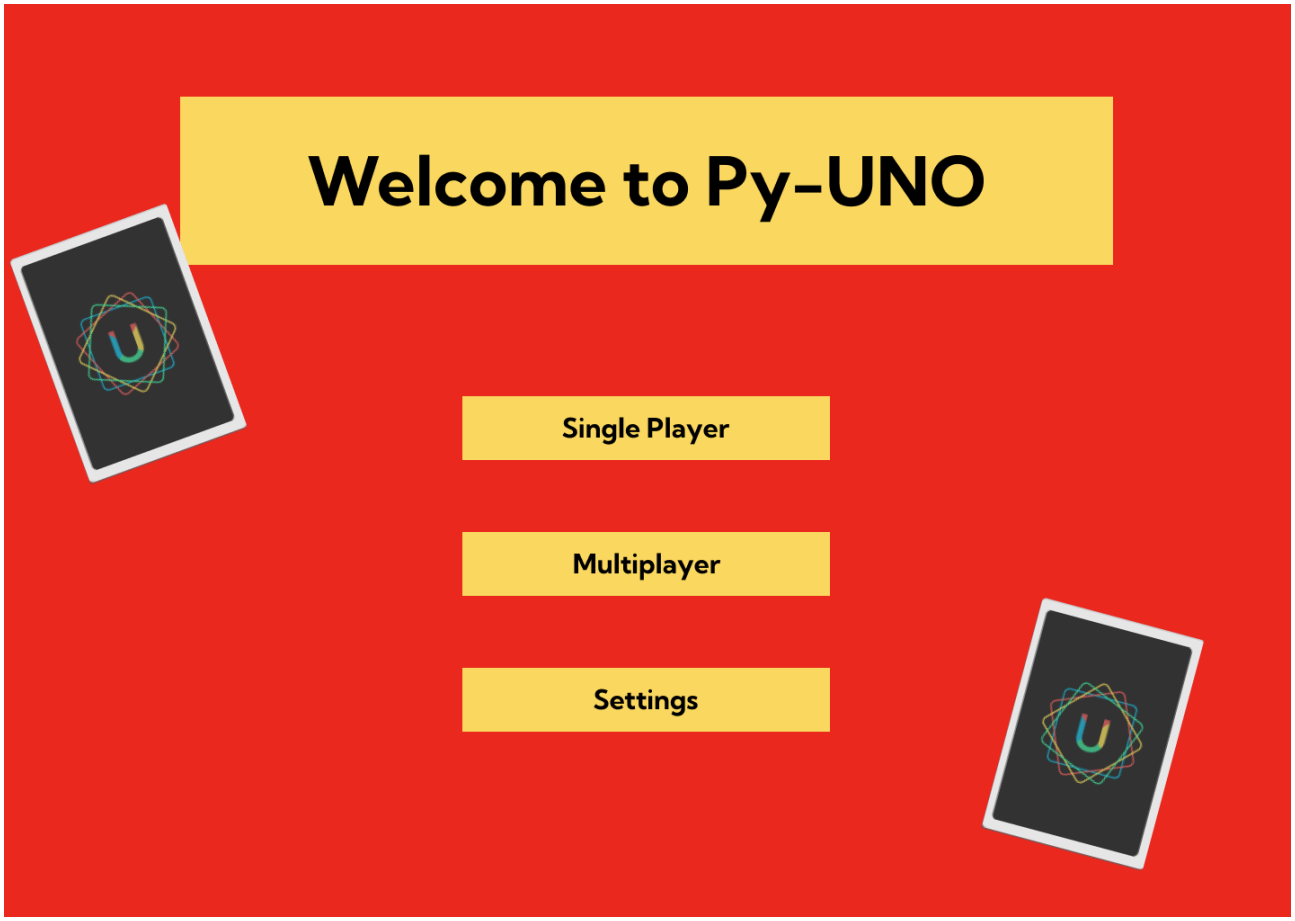
- **Language Limitations:** Python is an interpreted language, which can make it slower than compiled languages such as C++ or C#. Due to its interpreted nature, errors in Python may not become apparent until the code is executed. Furthermore, even though there is a community dedicated to Python game development, it is dwarfed by the communities dedicated to Unity and Unreal. The smaller community and resources for Python game development could pose challenges in advanced stages of the project.

3. External Interface Requirements

3.1 User Interfaces

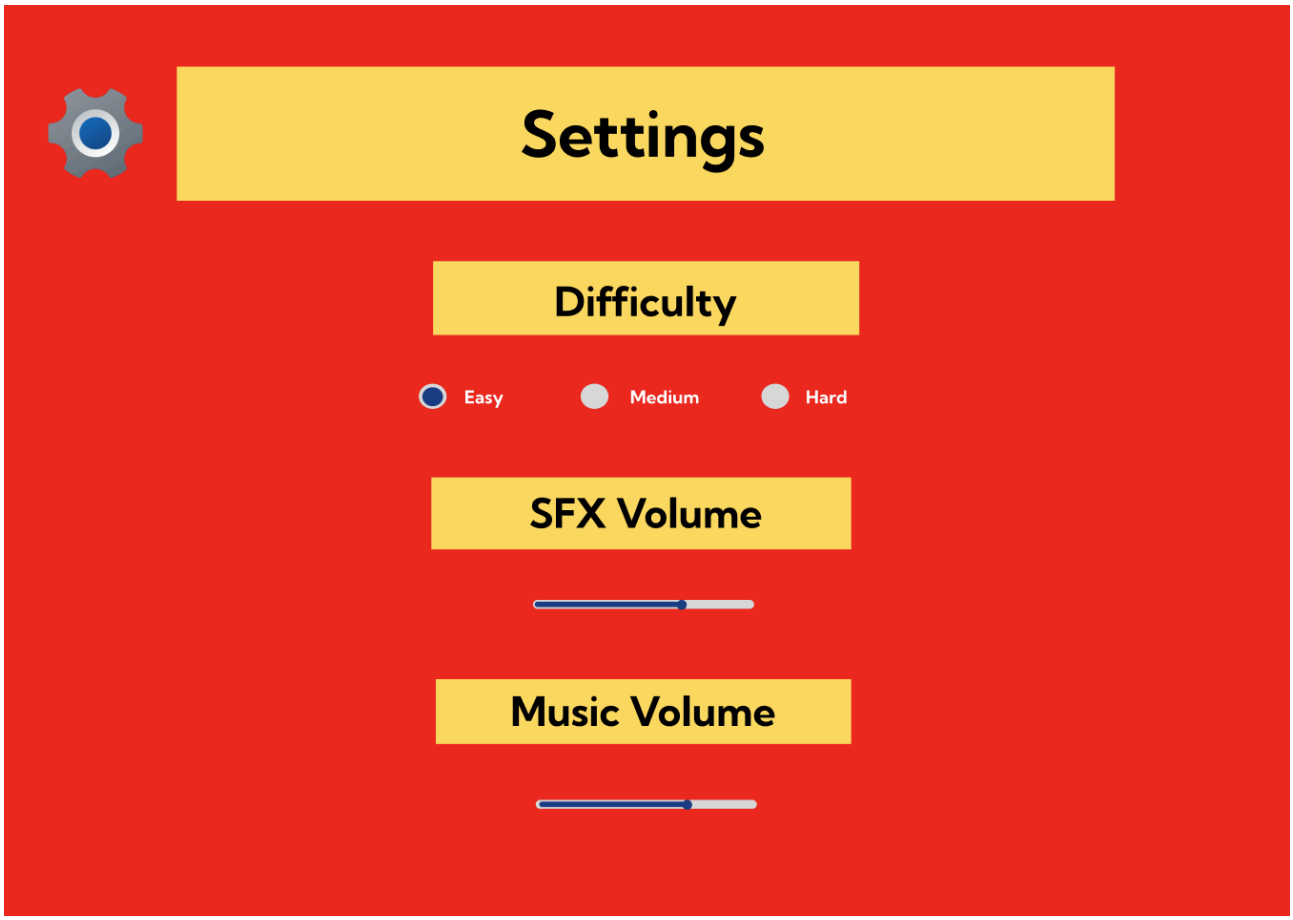
The user interfaces for the Py-Uno game are designed to provide an intuitive and user-friendly experience, ensuring that players can easily navigate and interact with the game. Each interface is structured to facilitate a seamless flow from one action to the next.

3.1.1. Home Page



The home screen welcomes players to Py-Uno with a yellow banner atop a red background. Players are greeted with three key choices in the form of yellow buttons: “Single Player”, “Multiplayer”, and “Settings”. The layout is straightforward, ensuring that the player can quickly navigate to their preferred game mode or adjust settings with ease.

3.1.2. Settings Page Screen

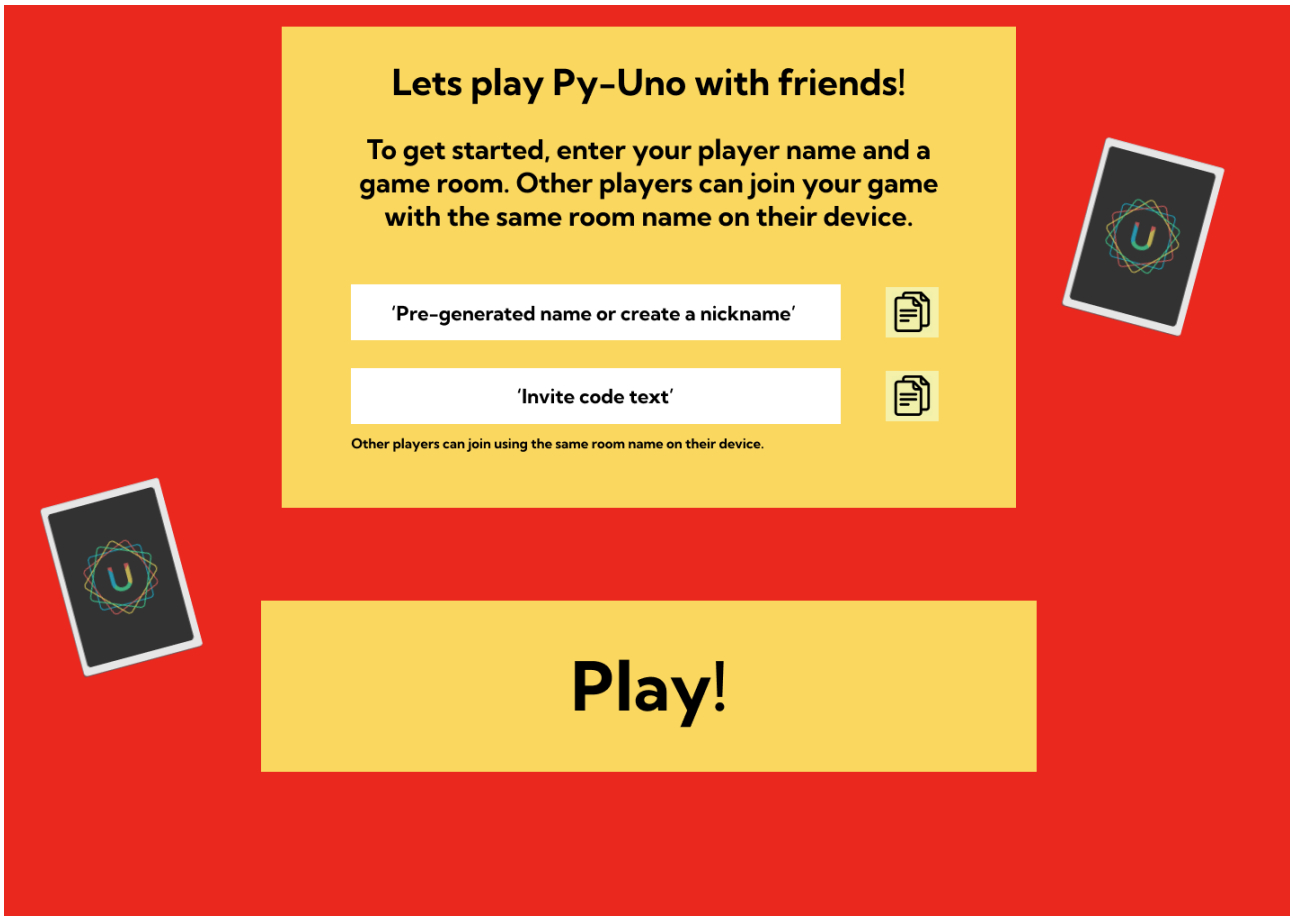


The settings page was made so that players can personalize their game experience by adjusting “Difficulty”, “SFX Volume”, and “Music Volume” through marked buttons and a volume bar for adjusting. The gear icon at the top signifies customization, while the consistent color scheme maintains the game’s energetic vibe.

3.1.3. Single Player Lobby Screen

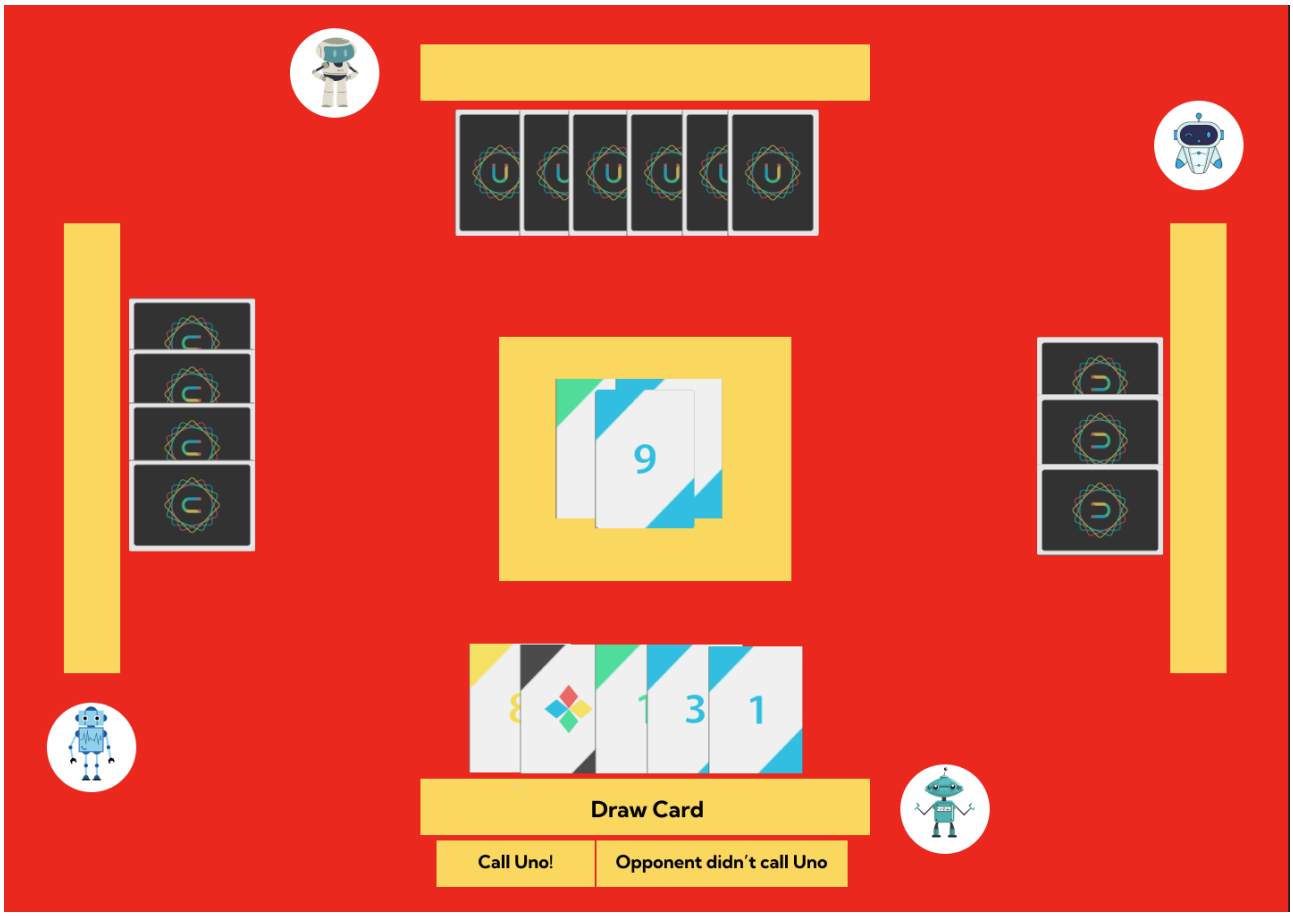
Before dividing into a solo game, players are prompted to select their challenge level on this screen by choosing the number of NPC opponents. The interface is uncluttered, focusing the player's attention on the "1", "2", or "3" NPC options, framed by Uno card graphics. A sizeable "Play" button awaits the player's choice, positioned centrally.

3.1.4. Multiplayer Lobby Screen



Designed to kickstart the multiplayer experience, this screen allows players to set up their game room. It features fields for “Pre-generated name or create a nickname” and “Invite code text”, along with a prominent “Play” button. The side graphics of Uno cards add a thematic touch, while the instructions provide clear guidance for new players joining the fun.

3.1.5. Game Session Screen



During an intense game session, this interface keeps the player immersed with a clean and organized layout. The player's cards are displayed at the bottom, with options such as "Draw Card", "Call Uno!" and "Opponent didn't call Uno" within easy reach. The central area showcases the discard pile and the current card in play, surrounded by the hidden hands of AI or multiplayer opponents, symbolizing the game's progression and interaction.

3.1.6 End of Game Session

This screen presents players with a simple choice after a game of Uno concludes. It features two buttons - "Play Again?" and "Quit", offering players a straightforward option to either continue the fun or exit the game. Player avatars are positioned at the corners, indicating the multiplayer aspect, while the central area remains clear for focus on the decision at hand.

3.2 Software Interfaces

3.2.1 Overview of Pygame and Pygwidgets

Py-Uno is developed using Python , with Pygame serving as the primary framework for game development, handling graphics, sound and game mechanics. Pygwidgets is utilized for creating and managing interactive widgets with the python environment.

3.2.2 Purpose and Use of Methods

- **Pygame Methods:**
  - **`pygame.mixer.Sound()`:** Used for losing and playing sound effects in the game.
  - **`pygame.display.set_mode()`:** Sets the window size for the game, creating the man displace surface.
  - **`pygame.image.load()`:** Loads image assets, such as sprites and backgrounds.
  - **`pygame.mixer.play()`:** Plays the loaded sound effects at specific game events.
  - **`pygame.display.update()`:** Updates the contents of the entire display.
- **Pygwidgets Methods:**
  - **`pygwidgets.TextButton()`:** Creates interactive text buttons for the game's user interface.
  - **`pygwidgets.enable() / pygwidgets.disable()`:** Enables or disables widgets. Widgets are disabled/enabled depending on what card is at play or who's turn it is.
  - **`pygwidgets.HandleEvent()`:** Handles events related to pygwidgets, such as button clicks.
  - **`pygwidgets.ImageCollection()`:** Manages a collection of images. For the Py-Uno application, ImageCollection stores the card images when a deck is initialized.
  - **`pygwidgets.replace()`:** Replaces one widget with another. The replace() method is utilized when transitioning the back of a card to the front.
  - **`pygwidgets.DisplayText()`:** Displays text to the screen, used for game information, scores, etc.

3.2.3 Integration and Data Flow

The game leverages the Pygame library for its core functionality, including rendering graphics and playing sounds. Pygwidgets is integrated into this environment to enhance the user interface with interactive and dynamic elements. For example, pygwidgets.TextButton() is used in conjunction with pygame.display.update() to render interactive buttons on the screen.

3.2.4 Dependencies and Versioning

Use these specific versions to ensure compatibility and proper functionality of the game.

- **Python Version:** 3.7
- **Pygame Version:** 2.0.1
- **Pygwidgets Versions:** 1.0.4

3.2.5 Constraints Considerations

- Due to performance considerations, sound effects (handled by `pygame.mixer`) are optimized for minimal latency.
- The game's UI, managed by Pygwidgets, must be responsive and work seamlessly with Pygame's event loop.
- All images are preloaded using `pygame.image.load()` to reduce in-game loading times.

3.3 Communications Interfaces

Py-uno requires a robust and efficient communication interface to ensure seamless gameplay and interaction among players over the internet. This section outlines the communication requirements and standards used by Py-Uno to facilitate online multiplayer functionality.

3.3.1 Network Communication Protocol:

- **TCP/IP:** Py-Uno utilizes the Transmission Control Protocol (TCP/IP) for reliable data transmission between clients (players) and server. The **`socket`** library in Python is extensively used to establish and manage TCP connections, enabling real-time game state updates and player interactions.

3.3.2 Programming Libraries and Modules:

- **Socket:** This library is for implementing the TCP/IP protocol, enabling Py-Uno to establish connections, send messages, and receive data over the network.
- **_Thread(Threading Module):** To manage concurrent operation without interrupting the main gameplay loop, Py-Uno leverages Python's `_thread` module for multithreading. This allows the server to handle multiple client connections simultaneously, ensuring responsive gameplay for all players.
- **Sys:** The **`sys`** module is used for interacting with the Python interpreter, primary for retrieving command-line arguments that may specify server configurations or player preferences.

3.3.4 Data Transfer Rates and Synchronization:

- Py-Uno is designed to be efficient in its network communications, requiring minimal bandwidth to transmit game updates and player actions. The game optimizes data transfer rates to accommodate players with varying internet speeds, ensuring a consistent and fair gameplay experience.
- Synchronization mechanisms are implemented to ensure that all players have a consistent view of the game state.

4. System Features

Py-Uno will feature both a single player and multiplayer mode, with adjustable AI, music, and sound effects. The game mechanics include all traditional Uno cards and mechanics for calling “Uno” or impose penalties on those who neglect to do so.

4.1 Single Player Game

4.1.1 Description and Priority

- **Description:** This mode enables players to engage in a game of Uno against computer-controlled opponents using AI, with all standard rules of Uno applied.
- **Priority:** High
- **Benefit:** 9
- **Penalty:** 8
- **Cost:** 5
- **Risk:** 4

4.1.2. Functional Requirements

REQ-1 (SinglePlayerMode-Start): The application must allow users to start a single player game from the main menu.
REQ-2 (SinglePlayerMode-AIDifficulty): The application must allow users to adjust the AI difficulty through the settings page.
REQ-3 (SinglePlayerMode-GamePlay): The application must validate the user's moves and respond with appropriate system actions (e.g. play a card, draw a card, etc.)
REQ-4 (SinglePlayerMode-EndGame): The system must determine and display the game's outcome and provide options for replay or exit to the main menu.

4.1.3 Use Case

Use Case Name:	**Starting a Single Player Game**
Brief Description	This mode enables players to engage in a game of Uno against computer-controlled opponents using AI, with all standard rules of Uno applied.
Pre-Conditions	The player is on the main menu and selects 'Single Player'.
Post-Conditions	A single player game is initialized with the specified number of AI opponents.
Events Flow	1. Player chooses 'Single Player' mode from the main menu. 2. Player is prompted to enter the number of AI opponents (1-3). 3. Player inputs the desired number and selects 'Start Game'. 4. The game initializes, dealing cards to the player and AI opponents. 5. The game proceeds to the gameplay phase.
Alternate Flow and Exceptions	If the player enters an invalid number of AI opponents, an error message is displayed, and they are prompted to enter a valid number.

Use Case Name:	**End of Game Decision**
Brief Description	Players decide whether to play another round or quit after a game concludes
Pre-Conditions	A game round has ended
Post-Conditions	The player either exits to the main menu or begins a new game.
Events Flow	1. The end-of-game screen displays the option to ‘Play again?’ or ‘Quit’ 2. The player selects their choice 3. If ‘Play again?’ is selected, a new game initializes 4. If ‘Quit’ is selected, the game closes or returns to the main menu

Alternate Flow and Exceptions	If the player does not make a selection within a certain time, the system defaults to returning to the main menu

4.2 Multiplayer Game

4.2.1 Description and Priority

- **Description:** Allows a player to host a new multiplayer game session and invite others to join.
- **Priority:** High
- **Benefit:** 9
- **Penalty:** 7
- **Cost:** 7
- **Risk:** 7

4.2.2 Functional Requirements

REQ-5 (MultiplayerMode-CreateRoom): The application must allow users to create a room for others to join.
REQ-6 (MultiplayerMode-JoinRoom): The application must allow users to join an existing game room.
REQ-7 (MultiplayerMode-GamePlay): The application must synchronize game state across all player's systems in real-time.
REQ-8 (MultiplayerMode-EndGame): The application must accurately determine and display the winner to all players and offer options for a new game or exit.

4.2.3 Use Case

Use Case Name:	**Hosting a Multiplayer Game**
Brief Description	Allows a player to host a new multiplayer game session and invite others to join.
Pre-Conditions	The player is on the main menu and selects 'Multiplayer'.
Post-Conditions	A new multiplayer game session is created, and the player can invite others.
Events Flow	1. Player selects 'Multiplayer' mode from the main menu. 2. Player is provided with a pre-generated name or the option to enter a custom name. 3. An invite code is generated and can be copied to the clipboard.

	4. Player shares the invite code with other players to join the game. 5. Once other players have joined, the host selects 'Start Game'. 6. The game initializes and transitions to the multiplayer game board.
Alternate Flow and Exceptions	If no other players join within a specified time frame, the player is notified, and the session is closed or the player is given the option to wait longer.

Use Case Name:	**Join a Multiplayer Game**
Brief Description	Allows a player to join a multiplayer session.
Pre-Conditions	The player is on the main menu and selects 'Multiplayer'.
Post-Conditions	Player joins a multiplayer session.
Events Flow	1. Player selects 'Multiplayer' mode from the main menu. 2. Player is provided with a pre-generated name or the option to enter a custom name. 3. The player paste the invite code in the required field and press join. 4. The game initializes and transitions to the multiplayer game board.
Alternate Flow and Exceptions	If the room the player is trying to join is no longer available, an error message is displayed.

Use Case Name:	**Using Special Game Actions**
Brief Description	Players use special actions during gameplay, such as calling “UNO” or challenging a player.
Pre-Conditions	The player is in an active game session.
Post-Conditions	The special action taken affects the game accordingly
Events Flow	1. During their turn, the player selects a special action button (ex: ‘Call Uno!’ or ‘Call Out’ 2. The system processes the action, applying the game rules. 3. The gameplay continues reflecting the outcome of the special action.

Alternate Flow and Exceptions	If a player fails to call “Uno” when required, the system will prompt for a penalty.

Use Case Name:	**Copying Multiplayer Invite Code**
Brief Description	Players can copy the multiplayer session invite code to their clipboard to share with friends.
Pre-Conditions	The player has created a multiplayer game session.
Post-Conditions	The invite code is copied to the player’s clipboard for easy sharing.
Events Flow	1. The player selects the 'Copy to clipboard' button next to the invite code text field. 2. The system copies the invite code to the clipboard. 3. The player can then paste and share the code with other potential players.
Alternate Flow and Exceptions	If the copy function fails, the player is prompted to manually select and copy the invite code.

Use Case Name:	**Changing Player's Nickname**
Brief Description	Player changes their pre-generated nickname in the multiplayer lobby.
Pre-Conditions	The player is in the multiplayer lobby.
Post-Conditions	The player has a new nickname for the session.
Events Flow	1. The player clicks on the pre-generated name field. 2. The player types in a new nickname. 3. The player confirms the change, and the new nickname is displayed.
Alternate Flow and Exceptions	If the player enters an inappropriate name, the system prompts them to choose a different nickname.

4.3 Game Settings

4.3.1 Description and Priority

- **Description:** Allows players to adjust game settings such as difficulty level and volume controls.
- **Priority:** Medium

- **Benefit:**5
- **Penalty:** 3
- **Cost:** 2
- **Risk:** 2

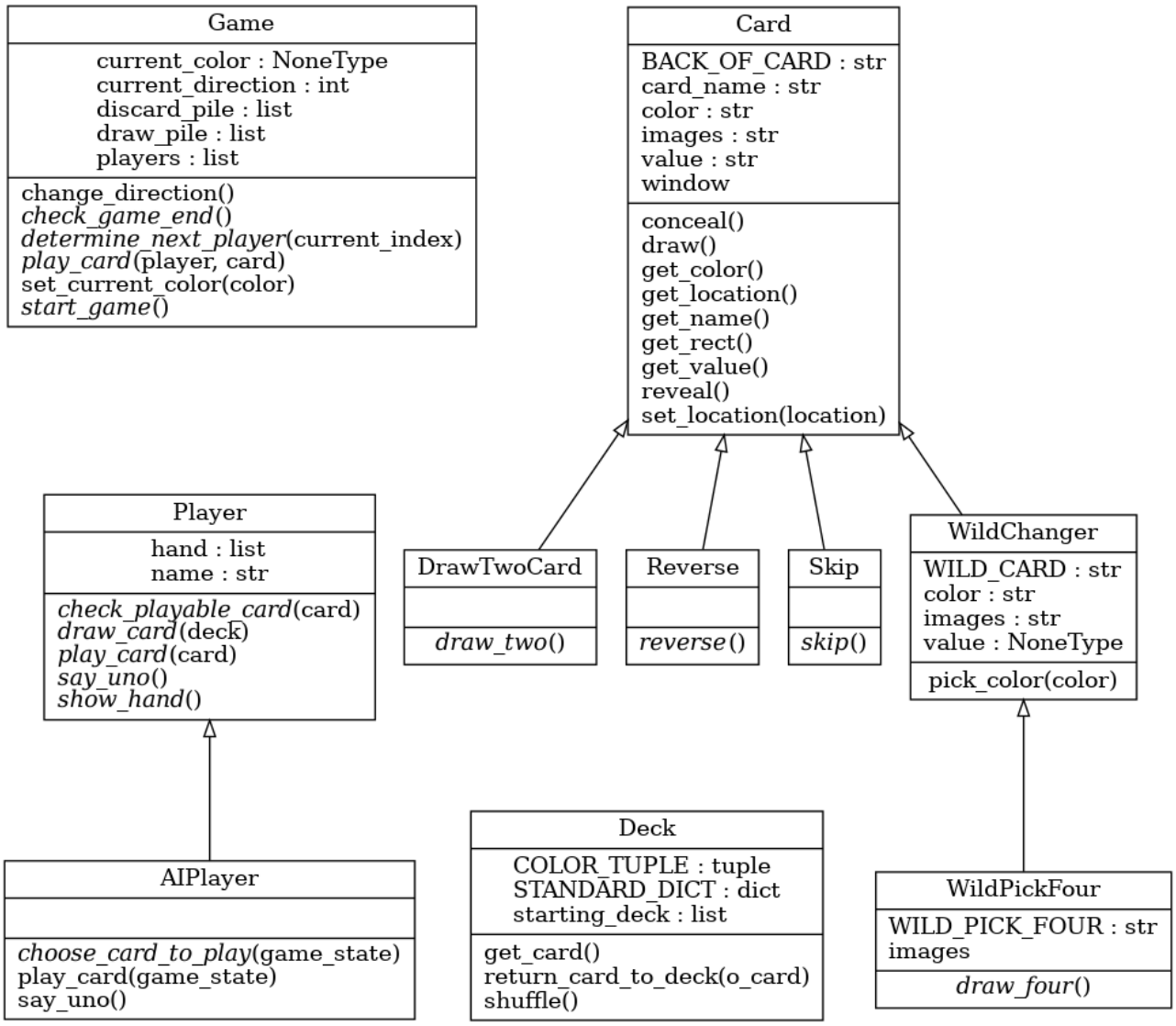
4.3.2 Functional Requirements

REQ-9 (SettingsPage-Adjustments): The application must allow users to adjust AI difficulty, Music Volume, and Sound Effects Volume.

4.3.3 Use Case

Use Case Name:	**Accessing game settings**
Brief Description	Allows players to adjust game settings such as difficulty level and volume controls.
Pre-Conditions	The player is on the main menu of the game
Post-Conditions	The players' settings choices are saved and applied
Events Flow	1. Player selects the 'Settings' option from the main menu. 2. The settings menu opens with options for difficulty, sound effects volume, and music volume 3. The player adjusts the settings according to their preferences. 4. Player confirms the changes, which are then saved. 5. The player returns to the main menu or continues to game play with the new settings in effect.
Alternate Flow and Exceptions	If the player exits the settings menu without saving changes, the original settings remain in effect.

4.4 Class Diagram



5. Other Nonfunctional Requirements

5.1 Performance Requirements

Optimize the game for stable framerate and quick response time. Ensure that transitions between menus and game screens are fast. (Main menu, settings page, info page, game board).

5.2 Reliability and Robustness

Implement error handling for when the game disconnects, or if the invite code expires.

5.3 Usability

Design the game interface to be user friendly and intuitive, from the start menu to the gameplay itself.

5.4 Additional Considerations

Ensure compatibility with different screen resolution and hardware. Regularly update the game to be compatible with the latest version of Pygame and Python. Optimize resource usage like memory and CPU to ensure smooth gameplay.

6. Appendix A: Art Assets

6.1 Number of Cards (0-9):

These are the basic Uno cards, each with a number from 0 to 9. They are colored(red, yellow, green. or blue) and are played by matching the top card of the discard pile either by number or color.

6.2 Draw Two Card:

When this card allows the player, the next player must draw two cards from the draw pile and forfeit their turn. This card is also color coded, and can only be played on matching color or another Draw Two Card.

6.3 Reverse Card:

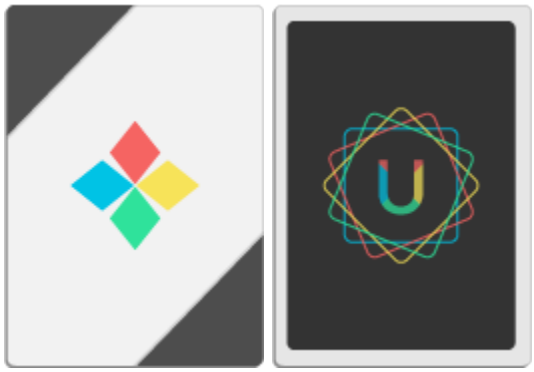
The Reverse card is played and the next player must draw two cards from the draw pile and forfeit their turn. This card is also color coded and must match.

6.4 Skip Card:

This card skips the turn of the next person in the sequence. it is color coded and must be played on a matching color or another Skip card.

6.5 Wild Card:

This card allows the player to change the current color in play. it can be played on any card, and the player who plays it decides what color to switch to.



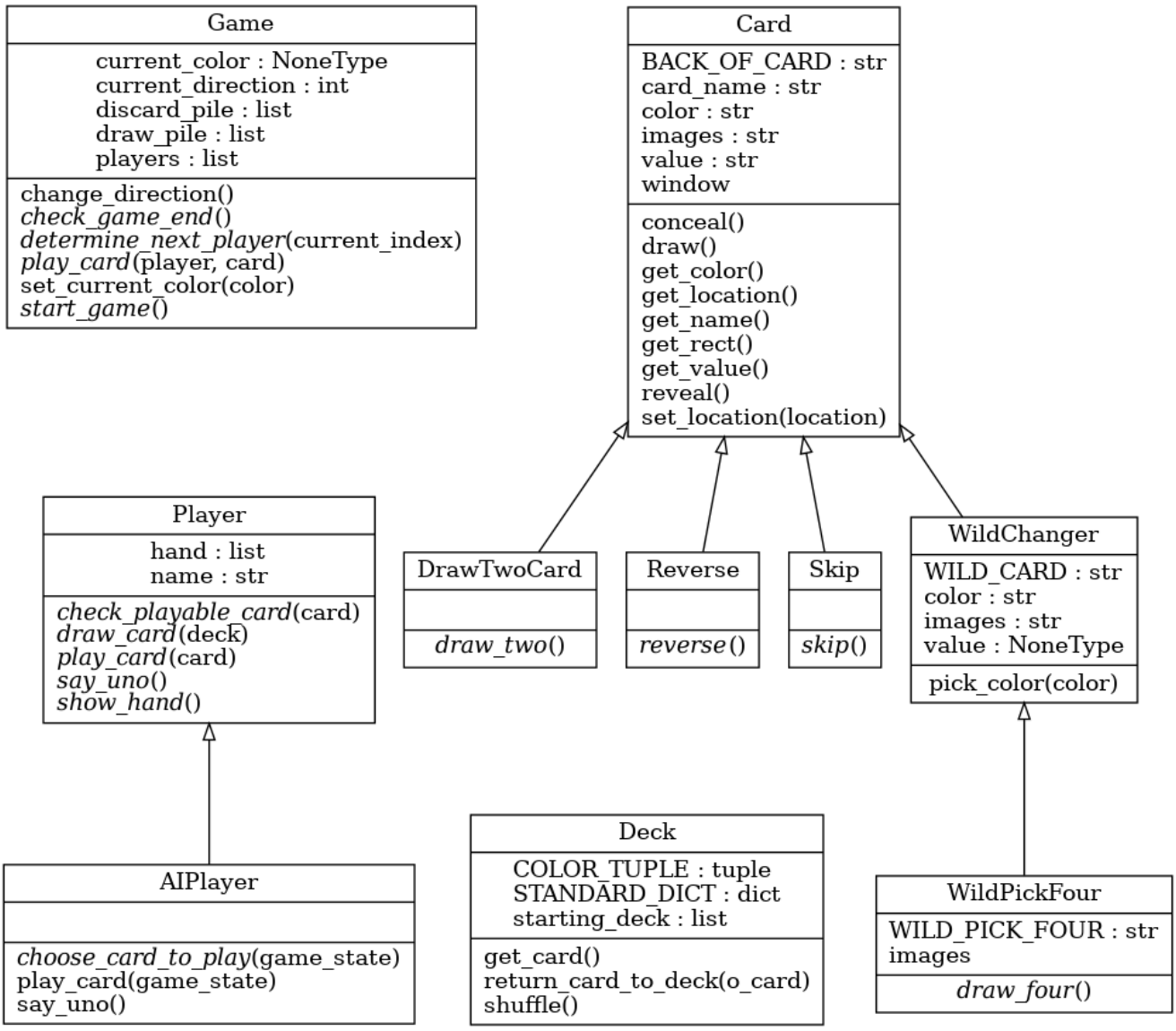
6.6 Wild Draw Four Card:

This card allows the player to change the current color in play and making the next player draw four cards. It can be played when the player has no other cards and forces the next player to draw four.

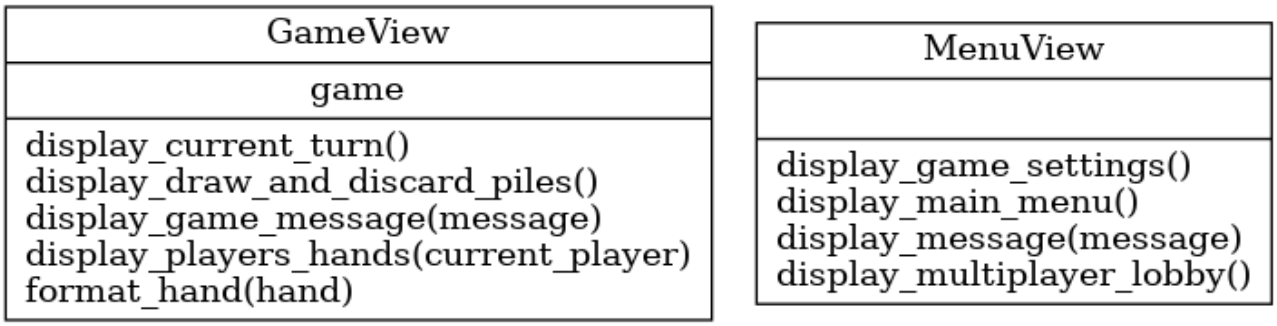
Appendix B: Analysis Models

MVC Class Diagrams

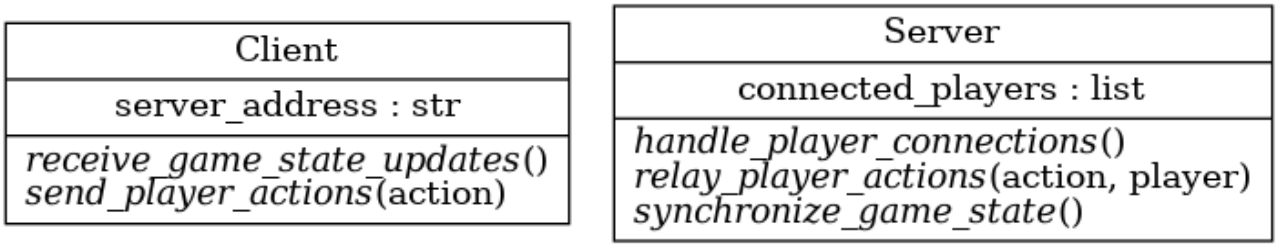
Model



View



Controller



Appendix C: Resources

Below is a list of resources used for this project.

- **Figma**: used to create the UI Mockups.
- **Lucidchart:** used to create the simple flow diagram.
- **Pylint:** To create the UML diagrams.
- **Git:** for version control.
- **Github:** For code repo.
- **GitKraken:** Used for convenience and
- **Jira:** For task management and sprint planning.
- **Pygame and Pygwidgets:** Used to create UI components.
- **OpenGameArt:** We sourced our Uno assets from here.
- **Object Oriented Principles:** "Object-Oriented Python" by Irv Kalb ISBN-13: 9781718502062. This book provides the object-oriented design approach used in the project.
- **Software Requirements Specification template:** This SRS document is derivative of the Software Requirements Specification template created by Karl E. Wiegers Copyright © 1999. Permission was granted to use, modify, and distribute the template.