PROJECT STATUS REPORT

Project Name	The UNO Game Project	**Project Duration**
Project Owner	Aykhan Ahmadzada	April 24, 2024 → May 15, 2024
Prepared by	Ahmadzada Aykhan	

Execution Steps Guideline

The UNO Game Project is a culmination of learning outcomes from the COMP132 course, demonstrating proficiency in Java programming. The project aims to implement the popular card game UNO in a digital format, showcasing fundamental programming concepts and object-oriented design principles.

Installation Instructions

No specific software or dependencies are required to run the UNO game application. You can simply open the project in any IDE that supports Java and build/run it.

Configuration Setup

No configuration setup is required for the UNO game application.

Execution Steps

1. Open the project in your preferred IDE.
2. Build the project if necessary.
3. Start the application by running the main class or executable file.
4. That's it! The UNO game should now be ready to play.

Testing Procedures

- Functional Testing:

Verify that the game rules are implemented correctly, including card drawing, playing, and discarding. Test various game scenarios, such as when a player has one card left, when a player plays a Wild card, or when a Draw Two card is played. Ensure that the game correctly handles special cards, such as Skip, Reverse, and Wild cards.

- User Interface Testing:

Test the graphical user interface (GUI) elements for responsiveness and usability. Verify that players can easily understand their turn, available actions, and game status. Test GUI components such as buttons, labels, and card displays for proper functionality.

- Error Handling Testing:

Test error handling scenarios, such as invalid user inputs or unexpected game states. Verify that appropriate error messages are displayed to the user, and the game gracefully recovers from errors without crashing.

- Performance Testing:

Test the game's performance under different conditions, such as varying numbers of players or complex game states. Monitor resource usage, such as CPU and memory, to ensure the game runs smoothly without excessive lag or slowdowns.

- Regression Testing:

After making any changes or updates to the game code, perform regression testing to ensure that existing features still work as expected. Re-test critical functionalities and edge cases to confirm that no new bugs have been introduced.

Troubleshooting

- Common Issues:
  - Installation Failure: Users may encounter difficulties in building or running the application in their IDE.
    - Solution: Double-check project configuration and ensure all dependencies are correctly installed. If issues persist, consult the project documentation or seek assistance from peers or instructors.
  - Runtime Errors: Users may encounter errors or unexpected behavior during gameplay.
    - Solution: Check console output for error messages and review code for potential bugs or logic errors. Ensure that input validation and error handling are implemented effectively to handle unexpected situations.

Project Overview

The goal of the UNO Game Project is to provide a hands-on opportunity for in-depth practice of Object-Oriented Programming (OOP) concepts and to explore Graphical User Interfaces (GUIs) in Java. Developed as a university project, the primary objectives include:

- **OOP Practice**: Utilizing OOP principles such as encapsulation, inheritance, and polymorphism to design and implement a comprehensive UNO game application.
- **GUI Mastery**: Learning to create intuitive and interactive user interfaces using Java's GUI libraries, enhancing user experience and engagement.
- **Independent Learning**: Encouraging self-directed learning and exploration of advanced Java programming concepts beyond the classroom curriculum.

The target audience for this project includes university students studying Java programming, particularly those seeking to deepen their understanding of OOP principles and GUI development.

Design Architecture

The UNO Game Project adopts a Model-View-Controller (MVC) architectural pattern, coupled with the Singleton design pattern for certain components. Here's an overview of the design architecture:

- **Model-View-Controller (MVC)**:
  - **Model**: Represents the underlying data and logic of the UNO game, including player information, card decks, game rules, and state management. It encapsulates the core functionalities of the game without directly interacting with the user interface.
  - **View**: Represents the graphical user interface (GUI) elements of the game, including the game board, card displays, player interfaces, and game status indicators. It presents the game data to the user and handles user input events.
  - **Controller**: Acts as the intermediary between the model and view components, facilitating communication and data flow. It interprets user actions from the view and updates the model accordingly, as well as updates the view in response to changes in the model.
- **Singleton Design Pattern**:
  - The Singleton pattern in the UNO Game Project guarantees that certain critical components remain as single, globally accessible instances. This ensures consistency and centralized control, enhancing the overall efficiency and reliability of the system.

By adopting the MVC pattern, the UNO Game Project achieves separation of concerns, modularity, and flexibility, allowing for easier maintenance, testing, and future enhancements.

Implemented Type and Hierarchies

The UNO Game Project encompasses various implemented types and hierarchies to facilitate the game's functionality and structure. These include:

1. **Card Types:**
   - **NumberCard:** This represents a standard numbered card in the UNO deck, characterized by a color and numerical value.
   - **ActionCard:** Represents special action cards in the UNO deck, denoted by their color and specific action, such as Skip, Reverse, or Draw Two.
   - **WildCard:** Represents wild cards in the UNO deck, which have no specific color but possess unique effects, such as allowing the player to choose the next color or forcing the next player to draw cards.
2. **Player Management:**
   - **Player:** Represents a player in the UNO game, associated with a user and a hand of cards.
   - **Bot:** Represents a bot player in the UNO game, inheriting from the Player class and providing specific functionalities for bot behavior.
3. **Enums:**
   - **ActionType:** Enumerates different actions associated with action cards, such as Draw Two, Reverse, and Skip.
   - **CardType:** Enumerates the types of cards in the UNO deck, including Number, Action, and Wild cards.
   - **Color:** Enumerates the colors of cards in the UNO deck, such as Red, Yellow, Blue, Green, and None.
   - **WildType:** Enumerates different types of wild cards, including standard Wild cards and Wild Draw Four cards.
4. **GameSession:**
   - Manages a session of the UNO game, including players, draw pile, discard pile, game state, and game logic.
   - Handles card distribution, player turns, game direction, Uno status, penalties, and game-end conditions.

These implemented types and hierarchies form the backbone of the UNO Game Project, facilitating robust gameplay mechanics and providing a structured framework for development and expansion.

Reference To Resources

In the UNO Game Project, various resources were utilized to aid in its development. These include:

1. YouTube Videos:
   - Video 1
   - Video 2
   - Video 3
   - Video 4
   - Video 5
   - Video 6
   - Video 7
2. Stack Overflow:
   - Stack Overflow was used for debugging code, seeking solutions to programming problems, and obtaining guidance from the programming community.
3. Google:
   - Google was utilized for sourcing images, icons, and colors to enhance the visual elements of the game.

These resources played crucial roles in acquiring knowledge, troubleshooting technical issues, and sourcing visual assets, contributing to the successful development of the UNO Game Project.

Video Demonstration

COMP132 Project | UNO Game Simulation (youtube.com)