FIT3077 - Architecture and Design

Sprint Four

MA_Tuesday08am_Team123

Swift Crafters

Team Members:

Maliha Tariq

Loo Li Shen

Khor Jia Wynn

Dizhen Liang

MONASH University

Table of Contents

1. Extensions Implemented

2. Object-Oriented Class Design

MONASH University

3. Reflection on Sprint 3 Design

General challenges and successes
Overview of the extensions implemented

3.1 Difficulty Levels of Incorporated Extensions

3.1.1 Extension 1

Description
Level of difficulty
Factors contributing to difficulty (e.g., distribution of system intelligence, code smells, design patterns used)
Reflection on what made this extension easy/difficult to address

3.1.2 Extension 2

Description
Level of difficulty
Factors contributing to difficulty (e.g., distribution of system intelligence, code smells, design patterns used)
Reflection on what made this extension easy/difficult to address

3.1.3 Self-Defined Extension 3 - “Equality Boost”

Description
The "Equality Boost" extension adds a feature to the game where if a player consistently reveals incorrect cards three times in a row, they receive an "Equality Boost" card. This boost allows them to automatically move one step forward on their next turn and still take their turn normally. This feature aims to balance the gameplay and offer a second chance to struggling players.

Level of difficulty
The level of difficulty for implementing this extension is moderate.

Factors contributing to difficulty (e.g., distribution of system intelligence, code smells, design patterns used)

Distribution of System Intelligence:
Player Class: The intelligence for tracking incorrect reveals and managing the "Equality Boost" is centralized in the Player class. This involves maintaining counters and state flags that track the player’s performance and boost status.
FieryDragonsApplication Class: The main application class had to be adjusted to handle new game logic for incorrect reveals, awarding boosts, and using boosts effectively.

Code Smells:
Complex Conditional Logic: The logic for handling incorrect reveals and boosts adds conditional complexity to the handleCircleClick method, which could be prone to errors if not carefully managed.
State Management: Introducing new state variables like incorrectRevealCounter and hasEqualityBoost increases the risk of state management issues if not properly initialized and updated.

Design Patterns Used:
Observer Pattern: The interaction between different components (e.g., player turn manager, chit card adapter, and text display manager) follows a pattern where changes in one component (e.g., player's state) trigger updates in others.
Factory Pattern: The game heavily relies on factory classes for spawning game entities, and the new feature had to integrate seamlessly with these existing factories.

Reflection on what made this extension easy/difficult to address
Easy Aspects:
Modular Design: The game's design, which uses factory patterns and clearly separated responsibilities (e.g., player state management, UI updates), made it straightforward to add new features without deeply entangling with existing logic.
Centralized State Management: Managing the incorrect reveal counter and equality boost state within the Player class provided a clear and contained way to implement the feature.

Difficult Aspects:
Conditional Logic Complexity: Adding the equality boost logic introduced new conditional checks that increased the complexity of the handleCircleClick method. Ensuring that all conditions (e.g., incorrect reveals, end of turn, equality boost activation) were correctly handled required careful thought and testing.
State Synchronization: Keeping track of the player's state across different parts of the game (e.g., UI updates, game logic) required meticulous updates to ensure that the player's incorrect reveal counter and equality boost status were accurately reflected in gameplay.

3.1.3 Extension 4

Description
Level of difficulty
Factors contributing to difficulty (e.g., distribution of system intelligence, code smells, design patterns used)
Reflection on what made this extension easy/difficult to address

3.2 Constraints Introduced by Sprint 3 Design/Implementation

General constraints identified
Specific constraints for each extension

3.3 Retrospective on Sprint 3 Design/Implementation

Changes you would make if starting over
How these changes would ease the incorporation of extensions

3.4 Future Strategies and Techniques

Strategies and techniques to improve future practice
Thought process on how they might work in similar situations

MONASH University

4. Executable Instructions

Instructions:
The executables were built using Maven based on the pom.xml file. The output is a zip file and it should be located in a directory called 'target'.

You should be using Java SDK version 22.
To produce the zip file: In your IDE, run the command `mvn javafx:jlink` using Maven.
In IntelliJ, this can be done by pressing the Ctrl key twice and then searching for this command.

5. Sprint Contributions

Wiki GitLab

Contributor Analytics