FIT3077 Sprint 4

Team Information

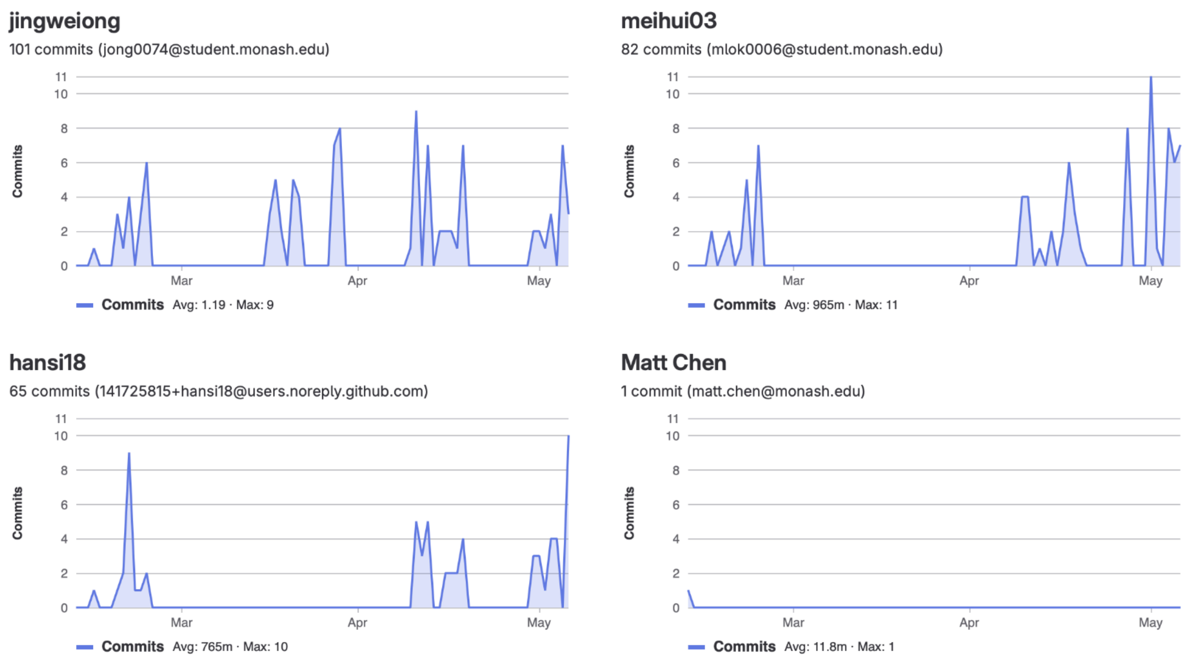
Team: MA_Tuesday12pm_Team003

List of members

1. Ong Jing Wei
2. Lok Mei Hui
3. Hansikaa Aggarwal

Content

Screenshot of the "Contributor Analytics"



Reflection Extension 1

A new dragon card has been introduced where, when flipped, the player's token must move backwards on the board until it finds an unoccupied cave (any cave without a token, not necessarily the one it started from). Once it finds a free cave, it positions itself there. The player does not lose their turn and can continue flipping the next Dragon card. If the player's token is already in a cave, it remains in place and the player's turn continues as usual.

Level of difficulty: medium

Animal Interface

For the new dragon card, a new animal, camel, is introduced into the system. With the implemented animal interface class in Sprint 3, it allows for consistency in handling different types of animals within the system, providing a common contract for accessing animal information. Thus, the camel class can just implement this interface class, which ensures that it adheres to the same standards as other animal classes in the system, making it easier to manage and interact with different types of animals in a unified manner. This approach promotes code reusability, maintainability, and extensibility, as new animals can be seamlessly integrated into the system by implementing the same interface.

Action class
The utilisation of an abstract action class in Sprint 3 significantly facilitated the incorporation of new actions, such as the one required for the new dragon card, which is the camel card. This design decision not only offered a systematic approach but also encouraged code reusability and maintainability. By extending the abstract class, implementing the new action, ToCaveAction became straightforward, ensuring consistency and scalability within the system.

handlePlayerTurn function
For the handlePlayerTurn, the previous implementation in Sprint3 is kinda lengthy, it has led to code smells. This is due to the presence of numerous if statements, which is often a sign of potential code smells, such as excessive branching and poor readability. Thus, violating the Single Responsibility Principle (SRP) by combining multiple responsibilities into a single method. Other than that, some logic like determining the current position and handling forward movement, was repeated within the function, leading to duplication.

But consolidating multiple conditional statements into individual functions, it can enhance readability and maintainability. Besides, by decomposing the complex logic into smaller, more focused methods, each responsible for a specific action or condition, the code will become more modular and easier to understand. This approach then follows the principle of single responsibility, ensuring that each function has a clear and concise purpose. Additionally, using meaningful function names can improve code comprehension and make it easier to debug or modify in the future. Therefore, we can now easily create a new function for checking when a camel card is flipped.

Reflection Extension 2

When a player flips this second new dragon card, their token swaps positions with the nearest token on the Volcano, whether it is ahead or behind. However, the player loses their turn, and it becomes the next player's turn. Tokens in caves cannot be swapped, even if they are the closest; consider these tokens as having an infinite distance. If a token is near its cave after nearly completing a circuit around the Volcano, it has to go around the Volcano again if it gets swapped “past” its cave.

Level of difficulty: Easy

Animal Interface
The implementation of the rabbit class as a new animal followed the Animal interface introduced in Sprint 3, adhering to the Open Closed Principle. This design choice facilitated seamless integration of new animals into the system without requiring extensive modifications to existing code. By maintaining a standardised approach to handling animal types, the system remained flexible and extensible, supporting future enhancements with minimal effort.

Action class
Extending the abstract Action class for the SwapAction allowed for the incorporation of new game actions without modifying existing functionality. This design decision upheld the principles of modularity and scalability, ensuring that the system remained adaptable to changing requirements. By leveraging inheritance, the integration of the SwapAction was achieved with minimal disruption to the existing codebase.

Flipping of Dragon Card
Due to the fact that we have refactor the handlePlayerTurn function into several functions that obey the Single Responsibility Principle (SRP), we can now easily add a new function called checkRabbitCard to handle when a player has flipped a dragon card with a 'R' symbol on it.

Readjusting Counter(number of moves to own cave)
When a dragon token that is close to its cave is swapped passed its cave, it will have to go around the Volcano Card again. Therefore we need to adjust the counter, so that the exact number of moves required for the player to reach its cave is accurate after the swapping.
As the logic of dragon token movement falls under the move function in DragonToken class, we can just add a if statement for checking if the current counter is -1, as -1 indicates that the dragon token has "past" its cave and thus we can readjust the counter over here.

Reflection Extension 3

Implementing the save and load game extension was a medium challenging task. The primary reason for this difficulty was the need to research effective methods for saving and loading game state, particularly in a way that aligns with object-oriented principles. By using JSON for serialisation and deserialization, we managed to keep the implementation clean and cohesive.

Level of difficulty: Medium

The save and load functionality did not significantly impact the existing codebase. This is because we designed the extension in an object-oriented manner, ensuring each class took responsibility for saving and loading its own state. This separation of concerns allowed us to integrate the new functionality without needing to make extensive changes to the existing logic.

In line with object-oriented design principles, each class was made responsible for its own state management. We implemented methods such as saveGame and loadGame for each class, enabling them to serialise and deserialize their attributes to and from JSON. This approach ensured that the code remained modular and reusable.

One of the main challenges was determining the best way to serialise complex objects, such as game state, into JSON format and then deserialize them back into their respective classes. This required thorough research into available libraries and techniques for JSON handling in Java. We opted for a well-documented library that supported deep serialisation and deserialization, which facilitated the process.

Reflection Extension 4

For the self defined extension, we have incorporated a new JPanel before the Fiery Dragon GameBoard. This panel asks about the player's details where we ask for the number of players and their choice of animal token. The game will allow a minimum of 1 player and a maximum of 4 players. As we begin the game, the game board appears where all 4 tokens are placed in their respective caves. The selected animal token players will play the game as usual, however the non selected players will be computerised players. These computerised players will have the ability to randomly select a dragon card and make a move accordingly. Hence this extended feature increases the player's engagement in the game and allows the player to choose their own token.

Level of difficulty: medium

InitialPage class
This new class is responsible for creating the new JPanel where we take details of players. As this class is introduced, we have modified the main page from GameBoard to InitialPage as this later creates the creates and display for GameBoard. This class uses JComboBox for taking the player's details where we ask the number of players and the dragon token of their own choice. The selected animal choices are stored in a list which is used later for randomTurn method. There is a Play Game button, which directs us to the game board.

GameManager class
The list of selected animals is initialised in this class. In the switchTurn method, we check whether the current player animal name is in the list of selected animals. If it is, then the game functions normally as it did before. But if the current player animal name is not in the selected animal list, then we use the new method, performRandomTurn. This method will randomly select any one of the dragon cards, and then pass it onto the onCardClicked method. After this, the game will perform as usual and then switchTurn to the next player.

Player Class
The Player class has been refactored to set the selected players as human or non human players. A new boolean attribute isHuman is introduced where we check if the player is human (selected) or not. We also made a new method which sets the select player as human.

These changes are made so that we can check every time a player turn's changes and if it's a human or non-human player and accordingly perform the performRandomTurn method.

Description of the self defined extension

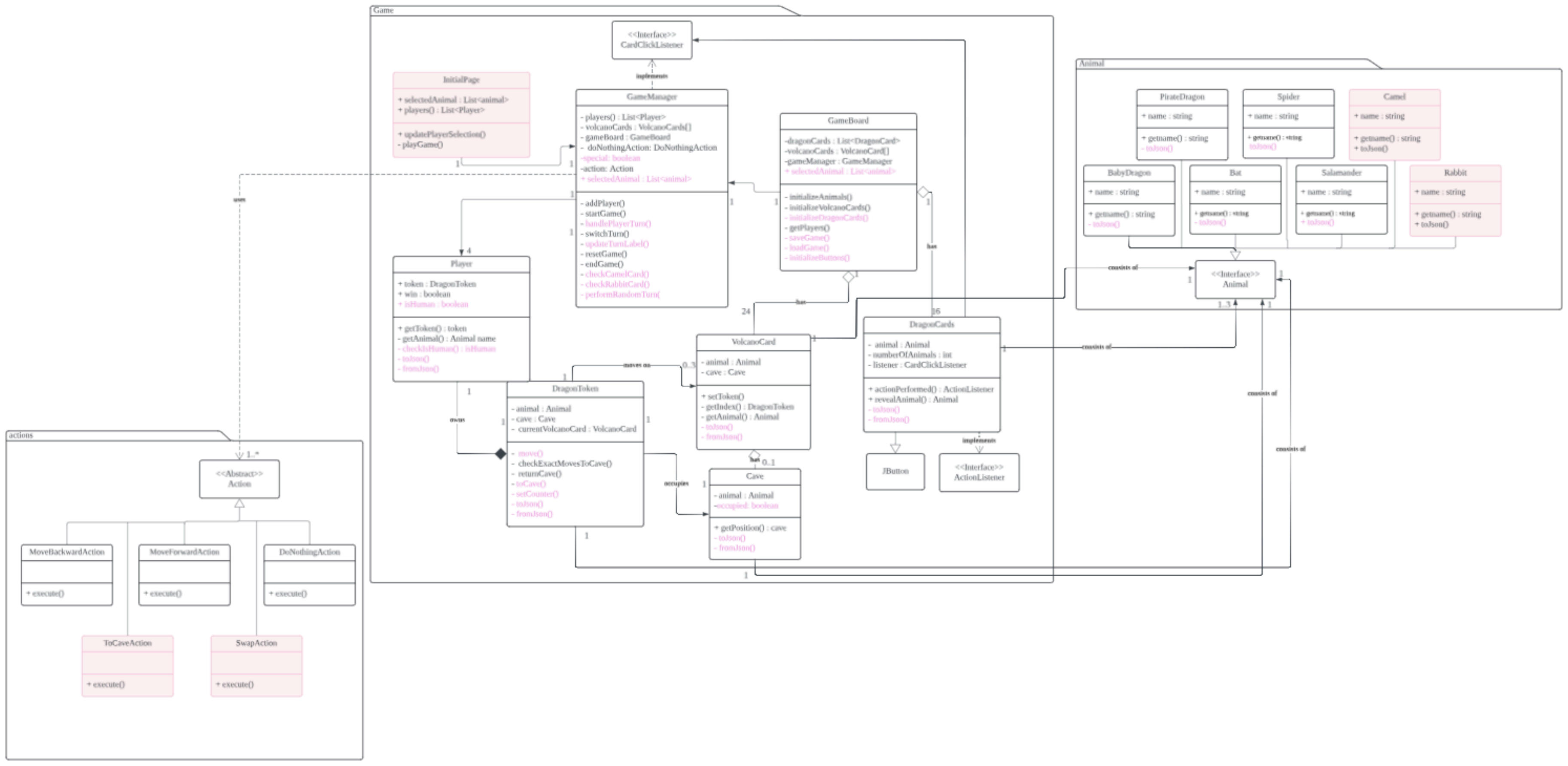
This self defined extension adheres to many human values like freedom, variation, excitement, capability, success , equality and many more. This feature manages to keep the player's engaged throughout the game and gives them freedom to choose their own token animal. We have also ensured that every player of the game has equal chances of winning the game whether it be manual or computerised users. The detailed description of the human values that extended functionality adheres to are:-

Self Direction & Stimulation
Taking the details of the player and asking their choices addresses 'Self-Direction' values particularly the freedom, and choosing their own goals. Allowing the player to select an animal of their own choice gives freedom to the players and also allows them to choose any player of their choice. Thus, the player is not bound to play with a specific animal. As they chose the animal of their choice, they also get to have different turns in the game. This also promotes the 'stimulation' human value as the player gets variation in the game. Players can get variation whenever they begin by picking their own favourite animal.

Achievements and Hedonism
As the game begins, all the players will play the game (including selected and non selected animals). The selected animal players will choose the dragon card of their choice and move their token accordingly however the non selected animal players will be randomly choosing the dragon cards and move accordingly and switch to the next player. This ensures that all players are equally ambitious to win the game and can be successful. This will promote player engagement within the game, fostering a sense of pleasure and enjoyment in the game. With the computerised players and the manual players, the game will be more exciting as there are many chances of clashing of players and other movements. Thus, the more the player, the merrier the game.

Universalism
As we allow all the players to play the game, we are addressing a sense of equality within all players. This promotes fairness in the game as every player has equal chances to either win or lose the game. This human value is one of the most crucial ones as the game is not biassed towards any player and every player is given a turn to flip the dragon cards.

Class Diagram



Executable

Executable file is located under the Project directory. The command line to run this file is

```
java -cp "Sprint4_MA_Tuesday12pm_Team003.jar:lib/json-smart-2.4.10.jar"
game.InitialPage
```

This class file version is 63.0, where a more recent version of Java Runtime is required.

The jar file is created using command:

```
javac -g -cp "./lib/json-smart-2.4.10.jar" -d out actions/*.java Animal/*.java game/*.java

jar cfe Sprint4_MA_Tuesday12pm_Team003.jar game.InitialPage -C out .
```