FIT3077 Sprint 3

Team Information

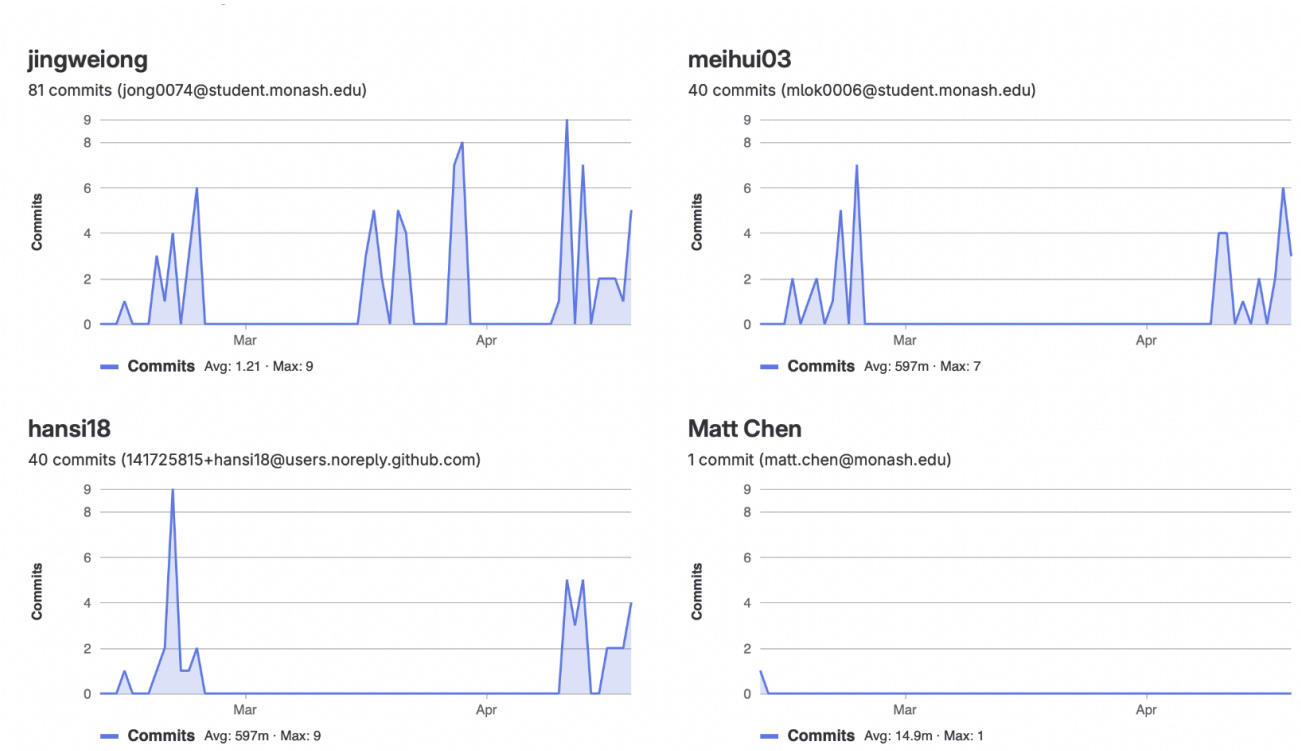
Team: MA_Tuesday12pm_Team003

List of members

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

Content

Screenshot of the “Contributor Analytics”



REVIEW OF SPRINT 2 TECH BASED SOFTWARE PROTOTYPES

Below is the definition of how the quality aspects are measured:

quality aspect	1	2	3	4	5
Functional Completeness :completeness of the design	several major omissions	a few (not many) major omissions	many minor omissions	minor omissions only	no omissions detected (which does not mean there are none, but you are not aware of any)
Functional Correctness :Correctness of the design	several major errors	a few (not many) major errors	many minor errors	minor errors only	no errors detected (which does not mean there are none, but you are not aware of any)
Functional Appropriateness:Rationale behind the Chosen Solution Direction	Unclear or unsupported rationale	Some rationale provided, but not fully supported	Clear rationale provided with minor gaps	Strong rationale provided with minor gaps	Strong rationale provided with no gaps
Appropriateness Recognizability :Understandability of the Solution Direction	Unclear or confusing	Somewhat understandable, but with significant issues	Understandable with minor issues	Clear and understandable with minor improvements	Clear and intuitive
Modifiability	Highly rigid and difficult to modify	Some areas difficult to modify	Moderately modifiable	Easily modifiable with minor improvements	Highly modifiable
Maintainability	code lacks organisation and documentation, difficult to understand and maintain	code has some organisation and documentation but may require significant effort to maintain	code is reasonably well-organised and documented, requiring occasional maintenance	code is well-organised and documented, requiring only minor improvements	code is well-structured, well-documented and easy to maintain

User Engagement	user interface is dull and unengaging	Some engagement, but lacks depth	some interactive elements or visual feedback to keep users interested	engaging with visual cues that enhance the user experience	highly engaging, with interactive element that captivates users

1. Ong Jing Wei

Functional Completeness

1. Initial Setup of the Game Board:
    - Score: 5
    - Explanation: The gameboard is fully set up according to specifications, using JPanel and paintComponent effectively. All necessary game components such as volcano cards, dragon tokens, dragon cards, and caves are correctly initialised.

2. Movement of dragon tokens based on their current position as well as the last flipped dragon card:
    - Score: 5
    - Explanation: All dragon cards can flip, and dragon tokens can move out from their caves and move to correct volcano card based on the number of animals on the flipped dragon card.

Functional Correctness

1. Initial Setup of the Game Board:
    - Score: 4
    - Explanation:
        a. Dragon cards remain concealed until flipped, positioned correctly in the middle of the gameboard
        b. Volcano cards are randomly arranged with attached caves
        c. Dragon tokens begin at their respective caves
        d. Dragon cards are shuffled at the start, however there is a minor issue that arises with the pirate dragon cards constantly appearing at the same position

2. Movement of dragon tokens based on their current position as well as the last flipped dragon card:
    - Score: 5
    - Explanation:
        a. Tokens move only if their animal matches the flipped dragon card
        b. Tokens move backward when a pirate dragon card is flipped
        c. Tokens move to the correct next volcano card based on the flipped dragon card's animal count
        d. Dragon cards reveal and hide after a few seconds, maintaining gameplay experience

Functional Appropriateness

1. Initial Setup of the Game Board:
    - Score: 5
    - Explanation:
        a. Game components are implemented using object-oriented principles, preventing the gameBoard class from becoming a god class where each game component, like caves initialised by VolcanoCard, and player instantiated their own token, enhancing code organisation

2. Movement of dragon tokens based on their current position as well as the last flipped dragon card:
    - Score: 5
    - Explanation:
        a. A CardClickListener class manages dragon card flipping, ensuring continuous handling of this game aspect.
        b. Dragon tokens have a move function, simplifying their movement between volcano cards.

Appropriateness Recognizability

1. Initial Setup of the Game Board:
    - Score: 4
    - Explanation:
        a. Game components are structured in an object-oriented manner, maintaining the code clarity and preventing the gameBoard class from becoming too complex
        b. Drawing of game components involves calculating coordinates, which adds complexity to the gameBoard, especially during interactions like flipping dragon cards as it needs to calculate the coordinates of mouse and dragon cards.

2. Movement of dragon tokens based on their current position as well as the last flipped dragon card:
    - Score: 5
    - Explanation: The move function abstracts code complexity and appropriately allocated responsibilities to game objects, enhancing code readability.

Modifiability

1. Initial setup of game board:
    - Score: 4
    - Explanation:
        a. The code's structure promotes modifiability, with separate methods for initialising game components facilitating easy addition of new elements without modifying the code
        b. The Animal abstract class enables seamless integration of new animals without altering existing code

c. The attached caves are hardcoded to specific volcano cards may hinder future modifications and require refinement for enhanced flexibility.

2. Movement of dragon tokens based on their current position as well as the last flipped dragon card:
    - Score: 3
    - Explanation:
        a. Token movement modification is straightforward due to a separate function dedicated to this purpose.
        b. Modifying dragon card behaviour may require adjustments to the CardClickListener to improve future usability as it handles most of the game logic

Maintainability

1. Initial setup of game board:
    - Score: 4
    - Explanation:
        a. The clear division of functionalities into distinct methods improves the code's maintainability, allowing for easier troubleshooting and updates

2. Movement of dragon tokens based on their current position as well as the last flipped dragon card:
    - Score: 4
    - Explanation:
        a. With each class handling specific responsibilities, maintenance becomes easier, following the principles of object-oriented programming except for CardClickListener

User Engagement

1. Initial setup of game board:
    - Score: 4
    - Explanation:
        a. Lack of visual differentiation between tokens and caves diminishes user experience as they have the same colour
        b. Gameboard is well-structured, enabling smooth interactions such as flipping dragon cards and token movement

2. Movement of dragon tokens based on their current position as well as the last flipped dragon card:
    - Score: 5
    - Explanation:
        a. Smooth dragon card flipping and automatic reversal enhance gameplay interactivity. Dragon tokens move seamlessly, enriching the player's experience.

Summary

The code solution exhibits a high level of functional completeness, with all required game components properly implemented. However, minor issues in functional correctness, such as repeated

appearances of specific dragon cards, require attention. The code's object-oriented approach enhances appropriateness and modifiability but introduces complexity in coordinate calculations, impacting code recognizability. Despite these challenges, the code promotes maintainability through structured organisation and facilitates user engagement through smooth gameplay interactions. Next, the code ensures all aspects of dragon card flipping and token movement function smoothly, providing a complete and correct gaming experience. Responsibilities are appropriately distributed among classes, simplifying maintenance and modifications.

2. Hansikaa

Functional Completeness

1. Initial setup of the gameboard
    - Score : 5
    - Explanation : The gameboard has been completely set up with all components of the game like volcano cards, cave, dragon cards and dragon token Made use of JPanel and JButtons for effective working of the game and accomplishing the requirement of initial setup of game board.
2. Winning the game
    - Score : 5
    - Explanation : Implemented an instance where a dragon token correctly moves back to its cave when the correct dragon card is flipped. A display pop up box appears when the player wins.

.

Functional Correctness

1. Initial setup of the gameboard
    - Score : 4
    - Explanation:
        A. The dragon cards are correctly implemented with each card being assigned randomly.
        B. The dragon tokens are correctly placed in their caves respectively.
        C. The volcano cards are correctly outlined however the naming of each section to animal is not correctly implemented .
        D. The caves are correctly placed on the outer ring of the volcano card however assigning it to each animal is not correctly implemented.
2. Winning the game
    - Score : 3
    - Explanation :
        A. The instance where the dragon token can move closer to the cave by one step is not implemented.
        B. Movement of token is hard coded to perform only one case on winning the game

Functional Appropriateness

1. Initial setup of the gameboard
    - Score : 4

- Explanation : The game correctly implements the components of the game without having any god class. All the components of the game are performing their own functions as needed. However, assigning animals to each component of the game is not implemented effectively.

2. Winning the game
    - Score : 3
    - Explanation : The winning cases of the game are implemented to check only a few winning cases due to which the implementation of the winning game is not effective. The winning case works correctly but does not check all the conditions.

Appropriateness Recognizability

1. Initial setup of the gameboard
    - Score : 5
    - Explanation : The setup of the game is using appropriate implementation for game components. Using JButtons for dragon cards for effective implementations are few of the appropriate steps taken for maintaining the code complexity. The game is structured in such a manner that it is easily understandable by any user and avoids any sorts of complexity.
2. Winning the game
    - Score : 4
    - Explanation : The instance of winning the game is performed only for a single token which requires easy implementation. The winning case is performed for the single token in a manner which is easy to understand and readable to the users.

Modifiability

1. Initial setup of the gameboard
    - Score : 4
    - Explanation :
        A. The dragon cards are implemented appropriately using JButtons which avoids any code complexity and can be easily extended for further modifications
        B. The volcano cards are perfectly sectioned for each animal and can be modified by adding graphics for better user experience.
        C. The dragon token and caves are initialised by the volcano cards and extend its functionality as needed. However, few implementations of the alignment are not easily modifiable.
2. Winning the game
    - Score : 3
    - Explanation : Winning the game only checks for a few cases of winning the game and is not correctly implemented due to which it can be hard to make modifications in the game for further use.

Maintainability

1. Initial setup of the gameboard
    - Score : 4
    - Explanation : The game is setup correctly according to the specifications however there are few instances where code readability is hard and few of the coding standards are not followed appropriately. Due to these reasons, it can be hard to maintain the code quality.

2. Winning the game
    - Score : 3
    - Explanation : All the winning cases are not tested appropriately due to which the code quality is poor and is hard for a user to read and understand. These are the reasons due to which it can be hard to maintain the code.

User Engagement

1. Initial setup of the gameboard
    - Score : 5
    - Explanation : The game has been structured correctly as to the specifications provided. The user interface is designed in a manner where it's easier for the user to flip the dragon cards and keep them engaged in the game.
2. Winning the game
    - Score : 5
    - Explanation : The winning case is designed in such a way that the user easily gets to know when the player has won. The display popup box after winning the game helps the user to easily realise that they have won the game.

Summary

The code presented in sprint 2 consists of a high level of functional correctness, appropriateness recognizability and user engagement for the gameboard. The setup of the game board includes all the game components according to the specifications, following an appropriate approach for game design. However the code falls short in terms of functional correctness, modifiability and maintainability. Certain components of the game do not perform their functions correctly, and some parts of the code are designed to pass specific test cases, which hampers modifiability and makes the code difficult to maintain. Despite these issues, the game provides a strong user experience, with a well-aligned interface that meets game requirements and engaging display pop-up boxes that keep users involved.

3. Mei Hui

Functional Completeness

3. Initial Setup of the Game Board:
    - Score: 5
    - Explanation: The game board is correctly set up with Dragon cards using JButton components.The others complement such as volcano cards, caves and dragon tokens are also set up on the board, therefore fulfilling the requirement for the initial setup.
4. Flipping of Dragon Cards:
    - Score: 5
    - Explanation: Mei Hui has implemented the flipping of Dragon cards, allowing players to reveal the animal on the card by clicking on it. The functionality is correctly implemented using JButton components and event handling.

Functional Correctness

1. Initial Setup of the Game Board:

- Score: 4
- Explanation:
    i. Dragon cards are arranged randomly within the “ring” of the volcano cards, where they cannot be distinguished(all dragon cards look the same before flipping).
    ii. The dragon board is also set up correctly with no missing caves or caves not in the middle of volcano cards.
    iii. The tokens are in the correct position at the start of the game, which is in their respective caves and are distinguished from each other.
    iv. However, the volcano cards are not arranged according to the original game design, as mei hui has the volcano cards arranged randomly.
    v. Overall, the setup of the game board is functional, but there are minor deviations from the original game layout.

2. Flipping of Dragon Cards:
    - Score: 4
    - Explanation:
        i. The type of animal and number of animals on the Dragon cards are revealed when flipped.
        ii. However, the Dragon cards do not hide back after some time, as there is no implementation for hiding the cards.
        iii. The flipping functionality is partially implemented but lacks the feature to hide the dragon cards back after a certain time.

<u>Functional Appropriateness</u>

1. Flipping of Dragon Cards:
    - Score: 5
    - Explanation:
        - Mei Hui chose to implement the Dragon cards as JButton components, which is a suitable choice for implementing the flipping of the cards.
        - This is because by using JButton components, it simplifies event handling and provides a familiar user interface element for interacting with the dragon cards.
        - This approach aligns well with the game's requirements and allows for straightforward implementation of card interactions.

<u>Appropriateness Recognizability</u>

Score: 5

Explanation:

- Mei Hui's solution direction is generally understandable as it follows common UI patterns and utilises standard Swing components like JButton for representing the Dragon cards.
- Players can easily recognize the cards and their interactions, such as clicking to reveal the animal on the Dragon cards.

<u>Modifiability</u>

Score: 3

Explanation:

- The code structure allows for moderate flexibility, with separate methods for creating various components of the game (volcano cards, caves, dragon tokens, dragon cards), making it relatively easy to make changes or add new features.
- However, some areas of the code may be tightly coupled, such as the dependencies between the Board class and its components, which could require refactoring to improve modifiability.
- Overall, the code is fairly modifiable, but there are some areas that may require attention to ensure ease of modification in the future.

Maintainability
Score: 4
Explanation:

- The code is well-structured and organised, with separate methods in Board class for creating volcano cards, caves, dragon tokens, dragon cards and players which improves readability and maintainability.
- Proper comments are provided to explain the purpose of each method and key sections of code, facilitating understanding and future modifications.

User Engagement
Score: 4
Explanation:

- The use of colour-coded caves and visual representations of the components like dragon tokens, dragon cards, caves and volcano cards engages the users and makes the game more visually appealing.
- Animations or effects could further enhance user engagement, but the current implementation already provides a good level of interaction.

Summary
The code demonstrates good functional completeness, functional correctness, functional appropriateness, appropriateness recognizability, maintainability and user engagement, with a well-structured organisation that makes it relatively easy to maintain. However, there are areas for improvement, particularly in modifiability, where some tight coupling may require refactoring to enhance flexibility. User engagement is enhanced through visual cues and colour-coded elements, but additional animations or effects could further improve the overall experience. Overall, the code provides a solid foundation for the Dragon Card Game, with opportunities for refinement to ensure its long-term maintainability and user engagement.

Justification of the tech-based prototype for Sprint 3

After reviewing sprint 2, we have decided to choose the best implementation for each component of the game while setting up the initial game board. This will ensure that when adding any new functionality, we will be easily able to modify the game. We also ensured that every team member is able to contribute to at least one of the components of the game for equal distribution of the game.

Implementation of volcano cards and caves

Jing Wei's implementation offers clear and understandable logic for attaching caves to specific volcano cards, enabling players to track their progress toward these caves. This design enhances player engagement and strategic decision-making by providing a clear indication of the remaining steps to reach the destination cave associated with a volcano card. Players can strategically plan their moves to reach the caves and gain advantages in the game. Additionally, the responsibility for creating caves lies with the volcano cards themselves, preventing the GameBoard from becoming a god class and distributing responsibilities effectively across the game components. This approach ensures a more organised and maintainable code structure.

Dragon Card implementation

We have decided to utilise the dragon card implementation from Mei Hui and Hansikaa's Sprint 2. Their design employs JButton to represent the dragon cards. This choice is advantageous because JButton offers built-in event handling for mouse clicks, simplifying the implementation of actions when a card is clicked. By directly attaching action listeners to the buttons, we streamline the code and minimise the risk of errors associated with manually tracking mouse coordinates. This approach not only enhances the reliability of the interaction logic but also makes the codebase cleaner and easier to maintain.

Having a Manager class that manage the game logic

Based on Jing Wei’s implementation, the GameManager class effectively handles the game logic, including the movement of dragon tokens based on the "flipped" dragon card. This design adheres to the Single Responsibility Principle (SRP) from the SOLID principles, ensuring that each class has a clear and focused responsibility. Using the Singleton pattern for the GameManager eliminates potential conflicts from multiple instances managing the game state, ensuring a consistent and reliable game experience. It also simplifies accessing the game state and logic from different parts of the application without worrying about instance conflicts.

Switch player implementation

From Mei Hui’s Sprint 2 implementation, the switchTurns method effectively manages the transition between player turns. We will continue using this method in Sprint 3, as it provides a reusable solution for changing turns whenever necessary. This approach reduces code duplication and enhances the cleanliness and maintainability of the codebase. By centralising the turn-switching logic within a single method, we ensure consistency across the game’s flow, making future modifications and expansions more straightforward and less error-prone.

Winning Game Situation

From Sprint 2, the mechanism for flipping dragon cards will be retained. This includes the smooth flipping animation and the timing function to reveal and conceal card content. The established logic for moving dragon tokens based on the flipped dragon card's animal count will be incorporated. Although Hansikaa attempted to implement the key game functionality, her approach involved placing the game logic within the dragon cards class and hardcoding the winning condition. This approach has limitations in flexibility and maintainability. Hence, we integrate JingWei's GameManager instance class to manage game states, including tracking flipped dragon cards, token positions, and the winning condition. An endGame function is developed within the GameManager to handle the winning game logic. This function will be responsible for determining when a player has won and resetting the game state. By creating a GameManager instance class, all game states and transitions are centrally managed, leading to clearer and more maintainable code.

New ideas

i) Message Indicating which player's turn

This message is displayed on the screen of the gameboard so that players can easily know whose turn it is and there's less chance of making mistakes, such as attempting to move out of turn. This can lead to smoother gameplay and a more enjoyable experience. Besides, when players know whose turn is coming up, they can plan their moves more strategically. This adds depth to the gameplay and encourages thoughtful decision-making.

ii) Counter for each Dragon Token

We introduced a counter for each dragon token object to track the number of steps remaining for the token to reach its cave, essential for winning the game. This enhancement aligns with object-oriented principles by encapsulating the state and behaviour within the dragon token class. By integrating the counter directly into the dragon token object, we ensure that each token independently manages its own progress, leading to a more modular and cohesive design. This encapsulation simplifies the game logic, as the GameManager can query each token for its step count, promoting better data integrity and reducing the risk of errors. Additionally, this approach enhances the maintainability and scalability of the codebase, as changes to the step tracking mechanism are localised within the dragon token class, making future modifications more straightforward.

iii) Action abstract class

We created an Action abstract class and extended it with MoveForwardAction, MoveBackwardAction, and DoNothingAction to enhance the game's design. This approach follows object-oriented principles by defining a common interface for different actions, ensuring a consistent way to execute actions on dragon tokens. By extending the Action class, we encapsulate specific behaviours within their respective classes, promoting code reusability and maintainability. This design allows us to easily add or modify actions in the future without changing the core logic, making the system more flexible and easier to manage.

CRC Cards

1.Description: The Cave class represents specific locations with unique properties, such as the type of animal they contain. A CRC card for Cave ensures that its role in the game, particularly its association with animals and its interactions with dragon tokens, is well-defined and managed.

Cave	
Know its animal type	Animal
Know its position	
Know the volcano card attached to it	Volcano Card

2.Description: The GameManager is chosen as it is responsible for overseeing the entire flow of the game, ensuring that all game rules and sequences are followed correctly

GameManager	
Manage the switching turn of players	
Check if the type of animals on dragon card and volcano card matches	
Know when to end the game	
Add players to the game	Player
Know when to reset the game	

3.Description: GameBoard is chosen as it is important for setting up all game elements and ensuring they are displayed and managed correctly

GameBoard	
Initialize and setup dragon cards	DragonCard
Initialize and setup volcano cards	Volcano Card
Initialize and setup caves	Cave
Initialize and setup dragon token	DragonToken
Initialise players	Player
Initialise label for displaying player’s turn	JLabel

4.Description: The DragonToken class is central to the movement mechanics of the game. It interacts directly with the game board, volcano cards and player actions. By having a CRC card for DragonToken, we can clearly define its responsibilities in tracking and moving tokens, ensuring it handles the movement logic correctly based on game rules.

DragonToken

Knows the volcano card it is on	Volcano card
Knows the cave that it belongs to	Cave
Knows the type of animal it represent	Animal
Knows its size	
Know which volcano card to move to when dragon card is flipped	Volcano Card

5.Description: The DragonCard class plays a crucial role in the game by representing the cards players interact with, which dictate the movements of the dragon token. These cards impact the flow and outcome of the game significantly. A CRC card for DragonCard ensures its responsibilities in determining and triggering specific game actions are well-defined, contributing to a clear and engaging gameplay experience.

DragonCard	
Display the type of animal and number of animals when flipped	Animal
Can be flipped when clicked by player	JButton
Notify game manager when it's flipped	GameManager
Know when to flipped back after a certain time	Timer

6.Description: The VolcanoCard class is responsible for knowing which cave it belongs to and determining valid moves for dragon tokens. Creating a CRC card for VolcanoCard helps ensure that it correctly manages its associations and movement rules, which are fundamental to game progression.

VolcanoCard	
Knows whether a dragon token is on it or not	Dragon Token
Can determine if a cave is attached to it	Cave
Knows its animal	Animal
Knows its position	

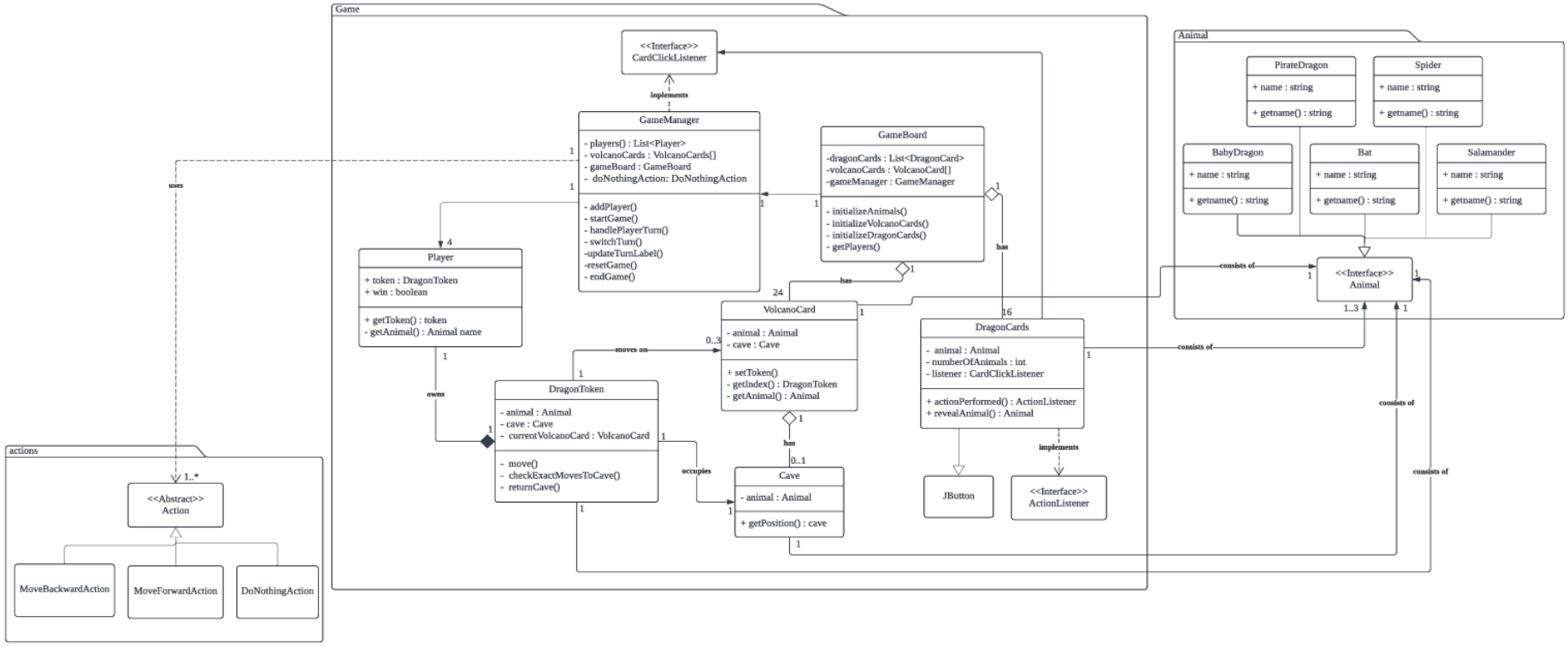
Discarded

GameManager	
Manage the switching turn of players	

Check if the type of animals on dragon card and volcano card matches	
Know when to end the game	
Add players to the game	Player
Move the dragon tokens when there are matching types of animals on dragon card and volcano card	DragonToken

We have decided to remove the responsibility of moving the dragon tokens from GameManager, instead we assign this responsibility to the Action abstract class. By doing so, we adhere to the Single Responsibility Principle(SRP) as each class should have only one reason to change. Besides, the code becomes more modular and easier to maintain as each class has a clear purpose and does not become overly complex.

Class Diagram



Executable

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

java -jar Sprint3_MA_Tuesday12pm_Team003.jar

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 actions/*.java Animal/*.java game/*.java
jar cfe Sprint3_MA_Tuesday12pm_Team003.jar game/GameBoard actions/*.class Animal/*.class game/*.cla