# FIT3077 - Sprint 3

# Team 24

# SANDY

Annie Ho (33156581)
Selsa Sony (33228302)
Yokabit Fesshaye (32591675)
Navya Balraj (32529252)

Team Name: SANDY

## Team Information

| Student Name | Contact Details | Technical and Professional Strengths | Fun Fact |
|---|---|---|---|
| Selsa | Phone:<br>0403641670<br>Email:<br>sson0024@student.monash.edu | Python<br>Java<br>Javascript<br>HTML/CSS<br>Communication<br>Collaboration | I have lived in 3 countries |
| Annie | Phone: 0406034601<br>Email: ahoo0034@student.monash.edu | Python<br>Java<br>Javascript<br>HTML/CSS<br>Adaptable collaboration | I can crochet and bake |
| Navya | Phone:<br>0452625236<br>Email: nbal0016@student.monash.edu | Python<br>Java<br>JavaScript HTML/CSS<br>Active Listening Problem Solving | I enjoy crafting and building furniture |
| Yokabit | Phone:<br>0469290048<br>Email:<br>yfes0001@student.monash.edu | Python<br>Java<br>CSS<br>Communication<br>Creativity | I like to swim |

# Table of Contents

# Sprint 2 Review Assessment Criteria

| Criteria | Expectations | Metric Assessment | Evaluation Method |
|---|---|---|---|
| **Game Functionality** | | | |
| Initial Game Board Setup | Key features covered, and matches implementation | Accuracy of initial board setup and randomness of dragon card positions | Check initial board setup against game rules and verify randomness of dragon card positions |
| Flipping of Dragon Cards | Correctness and randomness of card flipping | Correctness and randomness of card flipping | Verify adherence to game rules and analyse patterns in card flips |
| Movement of Dragon Tokens | Accuracy of token movement according to game rules | Accuracy of token movement | Observe token movements and ensure consistency with game rules |
| Change of Turn | Seamless transition of turns without errors | Seamless transition of turns | Monitor transitions from one player to another |
| Winning the Game | Correct determination of game-winning conditions | Correct determination of game-winning conditions | Test various scenarios to determine accurate winning conditions |
| **Rationale of Solution Direction** | | | |
| Design Rationale | Covers all required areas with clear justifications | Completeness of design rationale | Review documentation or design specifications |
| Clarity and Justification of Design Pattern Usage | Clear identification and explanation of design patterns | Clear identification and explanation of design patterns | Check design documentation for explicit identification and explanation of design patterns |
| Appropriateness of Design Pattern Context | Correct application of design patterns in relevant contexts | Correct application of design patterns | Analyse implementation for appropriate application of design patterns |

| | | | |
|---|---|---|---|
| Justification for Lack of Design Pattern Usage | Validity of justifications for not applying design patterns | Validity of justifications for not applying design patterns | Evaluate reasons for omission of design patterns |
| Strength and Compellingness of Design Rationale | Strength and persuasiveness of justifications | Strength and persuasiveness of justifications | Evaluate persuasiveness of arguments in design rationale |
| **Understandability of Solution Direction** | | | |
| Code Comments | Adequate comments for all code blocks, functions, and complex algorithms | Comment density, clarity, and relevance | Review code for comments explaining purpose, inputs, outputs, and key logic steps |
| Naming Conventions | Descriptive and consistent naming | Consistency and descriptiveness of naming | Check variable, function, and class names for consistency and accuracy |
| Modularity | Logical modules or components for specific tasks | Cohesion and coupling of modules | Examine code structure for logical modules and assess cohesion and coupling |
| Code Reusability | Common functions or algorithms abstracted into reusable components | Frequency and scope of reusable components | Identify reusable components and assess their promotion of code reusability |
| **Extensibility of Solution Direction** | | | |
| Modular Architecture | Clear separation of concerns with specific features or functionalities | Clarity of module boundaries and adherence to the single responsibility principle | Review codebase for well-defined module boundaries and adherence to the principle |
| Loose Coupling | Modules should not necessitate extensive modifications for changes | Degree of interdependence between modules | Analyse dependencies between modules |

| | | | |
|---|---|---|---|
| Abstraction | High-level interfaces hiding implementation details | Level of abstraction used to hide implementation details | Examine interfaces and abstractions in the codebase |
| Design Patterns | Use of design patterns for flexibility and modularity | Effective use of design patterns | Identify and assess the use of design patterns |
| **Quality of Written Source Code** | | | |
| Readability | Easy to read and understand | Clarity and ease of understanding | Review code for clear variable, function, and comment names |
| Consistency | Consistent coding style and conventions | Adherence to coding style and conventions | Check for consistent coding style and adherence to best practices |
| Simplicity | Simplified complex logic | Complexity of logic and solutions | Assess complexity of logic and solutions |
| Modularity | Logical modules or components for specific tasks | Organization of code into logical modules or components | Review codebase for logical modules |
| Scalability | Designed to scale with project size and complexity | Design considerations for future growth and complexity | Evaluate codebase for scalability and potential for future expansion |
| **Aesthetics of the User Interface** | | | |
| Visibility of Game Elements | All elements clearly displayed | Clarity and completeness of displayed elements | Review interface for visibility of all game elements |
| Dragon Board Setup | Correct setup with no missing elements | Completeness of setup | Verify presence and correct positioning of all elements |
| Token Positioning and Distinction | Tokens in correct positions and easily distinguishable | Correct token placement and distinctiveness | Ensure tokens are in correct positions and easily distinguishable |

| | | | |
|---|---|---|---|
| Dragon Card Arrangement | Dragon cards arranged correctly and uniformly | Arrangement and uniformity of dragon cards | Confirm arrangement of dragon cards and their uniform appearance |
| **User Engagement** | | | |
| Responsiveness | Prompt response to user actions | Responsiveness to user actions | Assess responsiveness to user actions |
| Error Handling | Graceful handling of user errors | Informative error messages and guidance | Assess error handling and user guidance |
| Intuitiveness | Easy to use with understandable controls and interactions | Intuitiveness of interface | Evaluate intuitiveness of the interface for users of all skill levels |

# Review and Assessment of Prototypes

## Review Prototype 1 (Annie):

Completeness of Key Functionality:
This prototype follows the game's rules and features, integrating the first and second functionality of Fiery Dragons. The initial setup is complete with four caves positioned on each corner of the game window and a game board consisting of twenty-four tiles, wherein each tile has an image of a dragon card. However, each cave is not connected to a tile of the game board so players might be unsure of the card they need to flip to step onto the game board. The dragon cards that are intended to be flipped over are placed in the middle of the game board, and have randomised positions for each execution of the game. The second functionality is complete with the ability to flip the randomised dragon cards over to display sequence/s of bats, spiders, dragons, dragon egg and pirate cards. Hence, the initial board setup is accurate, complete with randomised tiles on each of the twenty four tiles, and the dragon card positions are also randomised. Overall, both functionalities are complete and correct.

Rationale of Solution Direction:
The design rationale provides a comprehensive overview of the classes, relationships, inheritance designs and more. Clear justifications were provided for many design decisions such as the purpose of key classes, like 'Cards' and 'TileManager', explaining why they were designed as classes rather than methods. Additionally, the use of inheritance was justified and cardinalities between classes were explained. The use of the Observer's pattern in the 'GamePanel' class is clear as it highlights how mouse events are triggered to specific actions in the game. Dependency Injection used in the prototypes promotes loose coupling between classes and Resource Acquisition Is Initialisation ensures proper initialisation of resources. Hence, the usage of design patterns are justified and contribute to the flexibility of the codebase, aligning with the specific requirements of the game. In essence, the rationale of the solution is strong and effectively supports the design decisions made throughout the implementation, providing for a clear roadmap for the development of the game through the class and sequence diagrams.

Understandability of Solution Direction:
In terms of naming conventions, the prototype has descriptive and consistent variable and method names where it accurately reflects its purpose. There are some comments included throughout the code, which provide clarity. The code is organised into packages and logical modules, which are cohesive as each handles specific tasks related to the module. There is decreased coupling between

‘TileManager’ and ‘GamePanel’, however, abstraction is included in ‘DragonCards’, which promote code reusability, which is also demonstrated through the use of inheritance. Common patterns, such as game loop, and standard libraries, including ImageIO, help to understand the code. Areas for improvement include more dense comments in methods with complex logic and refactoring could be considered to encapsulate related functionality within classes more effectively. In conclusion, the code provides for a solid foundation with many factors of reusability.

Extensibility of Solution Direction:
The solution direction showcases a modular architecture with clear separation of concerns, evident in modules like ‘GamePanel’, ‘TileManager’, ‘AssetSetter’, and ‘DragonCards’, each handling specific functionalities while adhering to the Single Responsibility Principle. Coupling between modules is reasonably loose, minimising extensive modifications when changes occur, although some dependencies exist, particularly between GamePanel and other modules. Abstraction is present, notably in the DragonCards hierarchy, offering a level of flexibility for extending card functionality. Design patterns, such as inheritance for card types, are effectively utilised, contributing to the overall extensibility and modularity of the codebase.

Quality of Written Source Code:
The quality of the written source code is commendable, demonstrating readability, consistency, and modularity. Variable and function names are clear, adhering to Java conventions, and the coding style is consistent throughout. While the logic is generally straightforward, some areas could be further simplified for improved clarity. The code is well-organised into logical modules like TileManager, AssetSetter, and DragonCards, promoting modularity and ease of maintenance. Design considerations for scalability are evident, with the codebase structured to accommodate future growth and complexity, leveraging design patterns like inheritance for card types.

Aesthetics of the User Interface:
The purposeful pixelated design of the user interface presents a visually pleasing and functionally effective display. All game elements are clearly visible, ensuring players can easily interact with them. Dragon cards are arranged correctly and uniformly in a circle, contributing to the overall aesthetic appeal and ensuring consistency in gameplay. Overall, the interface is well-designed, with careful attention to the visibility, completeness, and arrangement of game elements, enhancing the user experience.

User Engagement:
The code demonstrates strong user interaction and engagement, characterised by prompt responsiveness to user actions, ensuring a smooth and interactive gameplay experience. User actions elicit quick and appropriate responses, enhancing engagement and immersion. Error handling is not currently implemented but could

be added in the future to offer guidance to players. Overall, the code effectively fosters user engagement through its responsiveness.

## Review Prototype 2 (Selsa)

Completeness of Key Functionality:
This prototype also follows the game's rules and features, integrating the first and second functionality of Fiery Dragons. The initial setup is complete with four caves, positioned north, south, east and west respectively, on a circular game board that is separated into twenty four tiles. Each tile displays an image of a relevant animal in the game that matches the images under the dragon cards in the middle. The game board is not randomised but the dragon cards in the middle are. The cards are spaced evenly in the inner circle and consist of a JButton, which can be clicked to view the image on the other side. The JButton can be clicked again to be flipped back and their positions are randomised with sets of coordinates named. Both functionalities are implemented to reflect the game's requirements with randomised positioning of dragon cards, which are complete and accurate.

Rationale of Solution Direction:
The design rationale provided for Prototype 2 covers the design decisions made for the game, including aspects such as encapsulation, association, aggregation and the lack of design patterns applied. Detailed explanations are provided for the design and design patterns, such as Singleton, Decorator and Proxy, have been identified and rationale has been provided for their lack of usage. Inheritance, cardinalities and abstraction has also been included and explanations provided. However, the lack of a design pattern increases the difficulty of extending the codebase.

Understandability of Solution Design:
The code demonstrates good understandability through its well-commented blocks, providing clarity on the purpose, inputs, outputs, and key logic steps of each function and class. Comments are adequately dense and relevant, aiding comprehension without being overly verbose. Naming conventions are consistent and descriptive, enhancing readability and maintainability. The code is modular, with logical components for specific tasks, exhibiting both cohesion within modules and appropriate coupling between them. Additionally, common functionalities are abstracted into reusable components, promoting code reusability and reducing redundancy. Overall, the code exhibits a cohesive and coherent structure, making it understandable and maintainable for future development.

Extensibility of Solution Design:
The code demonstrates strong extensibility through its modular architecture, with clear separation of concerns and adherence to the single responsibility principle, facilitating easy extension of specific features or functionalities without impacting

other parts of the codebase. Module boundaries are well-defined, promoting loose coupling between modules and reducing the need for extensive modifications when introducing changes. Abstraction is effectively utilised, with high-level interfaces hiding implementation details, thereby allowing for flexibility in future modifications or enhancements. While the code demonstrates strengths in extensibility, there are also areas for improvement. The modular architecture, while generally clear, may lack some flexibility in certain areas due to tightly coupled dependencies between modules, which could result in cascading changes when modifying certain functionalities.

Quality of Written Source Code:
The code for Prototype 2 exhibits several positive aspects in terms of the quality of written source code. It is highly readable, with clear variable and function names contributing to its ease of understanding. Consistency in coding style and conventions is maintained throughout the codebase, enhancing readability and maintainability. The simplicity of complex logic is effectively achieved, making the codebase comprehensible even for those unfamiliar with the project. Furthermore, logical modules are organised for specific tasks, promoting modularity and ease of maintenance. However, there are also areas for improvement. While the code is generally well-structured, there may be instances where further simplification of logic could enhance readability and reduce complexity. Moreover, the level of scalability is low.

Aesthetics of the User Interface:
In this prototype, all game elements are clearly displayed, contributing to the visibility and completeness of the interface. The Dragon Board setup is correctly arranged with no missing elements, ensuring the completeness and correctness of the setup. Additionally, token positioning is accurate, and tokens are easily distinguishable, enhancing the user experience and gameplay. However, there are areas for improvement. While the Dragon Card arrangement is generally correct and uniform, there may be opportunities to further enhance the arrangement for improved visual appeal and consistency.

User Engagement:
The code demonstrates positive user engagement attributes in some areas but lacks in others. It is responsive to user actions, providing prompt feedback and interaction, which enhances the overall user experience. However, error handling is lacking, as the code does not gracefully handle user errors or provide informative error messages or guidance. Additionally, while the interface is generally intuitive with understandable controls and interactions, the absence of error handling mechanisms may detract from its overall intuitiveness, especially for players with various skill levels.

## Review Prototype 3 (Yokabit):

Completeness of Key Functionality:
This prototype is lacking in completeness as it does not contain key functionality. Although there are some features and functionality, the initial game board set up is not complete nor does it contain the key functionality. Once the prototype loads, it does not lead to the expected software.

Design Rationale:
The design rationale for this prototype makes an attempt at explaining the decisions behind the specific choices. It covers the basic idea of single responsibility principle and how it has been implemented throughout the prototype, as well as inheritance. The design rationale explains extensively how the use of inheritance enables for code extensibility. However, although the design rationale slightly describes the design pattern that has been used, explanation of how it benefits code base extension could have been further discussed.

Understandability of the solution direction
The code provided in this prototype demonstrates slightly decent understandability of the solution direction. As mentioned before, this design approach makes use of inheritance which enables code reusability throughout the software, preventing duplication of the same code. The code could benefit from block comments as it is lacking significantly. The code contains decent class structure and most of the functionality code also follows an organised structure.

Extensibility of the solution:
The prototype makes an attempt at providing an opportunity for extensibility of the solution. For example, the way that the code is designed provides the ability to add more players and caves due to its inheritance nature. An effort has been made to utilise the single responsibility principle which is incredibly beneficial for code extensibility as it prevents functional dependencies and accommodates for the ability to change or adjust different sections without impacting the overall source code.

Quality of the written source code
The overall quality of the code is somewhat decent. It contains clear, distinguishable variable and class names, making it easy to understand the purpose of them. There has been an attempt made to follow Java coding conventions, however some of the code is missing different concepts such as encapsulation. There has been good use of classes and packages which enable understanding of the purpose of the code written.

Aesthetics of the user interface

Use of images and fonts have been implemented in the prototype. Although it is lacking functionality and a complete game, the aesthetic for what is provided is satisfactory to some extent. The aesthetic style and texts are consistent overall and are placed in suitable areas that can be seen on the screen. There is definitely space for improvement as aesthetics is still required for all key functionalities.

User Engagement
There is not much that can be discussed about user engagement as the prototype is missing completeness of key functionality. However as mentioned before, the tech progress does provide an aesthetic user interface. Had the prototype been complete with all required functionality, it can be assumed that the user engagement would have been natural to the user. This can be supported by the intuitive home screen and menu selection page.

## Review Prototype 4 (Navya):

Completeness of Key Functionalities:
This prototype sets up the initial game board in a manner that is partially complete with some inaccuracies. The gameboard does contain twenty-four tiles, each with an animal on it. There are also four caves on the gameboard, each clearly connected to a single volcano card. Each cave also has one animal and one dragon with the number of dragons on the gameboard equal to the number of players selected. The only issue with the caves is that two of them are not fully visible so it is unclear which animal and dragon is on them. The game board also contains dragon cards in the centre which are placed at random locations at each run of the game. However, there are not always sixteen cards visible for every run of the game as sometimes they are placed on top of each other so there seems to be less than sixteen cards. The game functionality of moving the dragons based on the current position of the dragons as well as the last flipped dragon card was also attempted but does not seem to work as none of the dragon images are able to move even when the selected dragon card has the same animal as the animal which the dragon is standing on. From first glance of the functionality when running the game, it does not seem to work, however upon further analysis of the code, the logic of when to move the dragons seems accurate according to the game rules. It is not clear which dragons turn it is when the button is clicked and the statement at the bottom of the screen is also inaccurate.

Rationale of Solution Direction:
This prototype unfortunately had no design rationale at all so none of the required areas were addressed.

Understandability of Solution Direction:

All methods have been commented with explanations of what each parameter and return value of each method represents. The comments are mostly relevant, clear and they aid in understanding the code. However these comments could be more dense inside complex methods and loops so that the code is understandable by other team members. No packages or modules have been used which would assist in logical structuring of the code. This prototype has low coupling and high cohesion as related concepts are grouped within classes without depending much on other classes which was achieved by using abstractions, inheritance and polymorphism. The naming conventions used are descriptive and consistent in format as variables, functions and classes are named as expected. The ‘MovementAction’ class is an example which promotes code reusability as it can be used for every movement in the game. To improve the code reusability even more, the drawImage() function could be placed in a separate class so that images can be drawn in other functions not just in paintComponent() method as it requires the g2d variable which could rather be used as an attribute of the new class created.

Extensibility of Solution Direction:
The solution direction used in this prototype follows clear separation of concerns shown mainly in the ‘GameBoard’ and ‘GameLogic’ class. There is fairly low coupling as classes aim to follow the Single Responsibility principle wherever possible. One possible area for improvement is that the ‘TokenCard’ class generates all 16 tokens, however it would be more beneficial if it only generated one token so there will be sixteen instances of this class which could be used with different animals under each card. Abstract classes have been used like ‘Dragon’, ‘Animal’ and ‘Action’ classes which act as the constant base of these classes, but they can have different implementations. This allows for the design to be easily extended to implement a new ‘Animal’ for example if needed for future extensions. One of the main design patterns which have been used is the Singleton pattern. The ‘GameLogic’ class uses this pattern as it has only one instance which can be accessed from any part of the program and there is a global point of access to it which is in the ‘Main’ class.

Quality of Written Source Code:
This prototype contains mostly easy to read and understand code. There are clear variable and function names and comments explaining most of the code. The logic is slightly complex in the ‘GameLogic’ class in the initialise() and run() methods where more comments would increase readability. Coding style used is consistent and adheres to Java conventions and practices. The logic of the solution for moving dragons is made simple with the ‘Location’ class. There have been no modules used which would definitely benefit the modularity and readability of the code. The design considers future growth like if more animals are added which is done using abstractions and inheritance.

Aesthetics of the User Interface:

The animated style of the game board acts as an effective display with all the game components being present. The main issue is that all the Caves are not fully visible however they are all present. The tokens are easily distinguishable and the locations are randomised but all 16 tokens are also not always visible as the number of tokens is different each run of the game. Dragon cards' arrangement is accurate and are uniform in appearance.

User Engagement:
This prototype exhibits some sense of user interaction as there is always a response when the user clicks the button. However, the statement that appears is not always accurate and it is not clear which dragons turn it is each time the button is clicked. There is no possible error from the user as the only functionality is the button which simply changes the statement in the text box so no error handling or user guidance was necessary. The prototype is easy to use and the controls are understandable by users of all skill levels. However, it is not clear if anything happens in terms of the game when the button is clicked as there is no movement on the game board.

## Prototype Rationale for Sprint 3:

The team has chosen Prototype 1 as the foundation for Sprint 3 due to its adherence to key principles essential for the development of our game, Fiery Dragons. Firstly, Prototype 1 demonstrates completeness in key functionalities, encompassing the initial setup of the game board, randomisation of dragon cards, and the mechanics for revealing different tiles. This completeness ensures that the core gameplay mechanics are well-established and ready for further expansion.

Secondly, the rationale behind the solution direction in Prototype 1 provides a comprehensive roadmap for development. The design decisions, such as the utilisation of design patterns like Observer and Dependency Injection, are justified, promoting code flexibility and maintainability. This clear rationale not only guides current development efforts but also lays the groundwork for future iterations. Moreover, Prototype 1 exhibits a high level of understandability, with clear naming conventions, organised code structure, and appropriate comments throughout. These factors contribute to the ease of comprehension and maintenance of the codebase, crucial for the ongoing development process.

Additionally, the extensibility of the solution direction in Prototype 1 allows for seamless integration of new features and functionalities. The modular architecture, clear separation of concerns, and loose coupling between modules facilitate scalability and adaptability, ensuring that the game can evolve over time without significant rework. Lastly, Prototype 1 sets a solid foundation for user engagement, with responsive interactions and a visually pleasing user interface. Building upon this foundation in Sprint 3, we aim to further enhance user engagement through the

introduction of new game mechanics, expanded content, improved visual feedback, and potentially multiplayer functionality.

To enhance the prototype further in Sprint 3, we can introduce several new ideas and elements. One crucial aspect is to augment the codebase with additional comments, particularly in areas with complex logic. These comments will not only improve understandability but also aid in maintaining the codebase over time, ensuring that developers can easily grasp the functionality of different components.

While other prototypes may introduce unique ideas or elements, integrating them into Sprint 3 could introduce complexity, inconsistency, or divergence from our established design principles. By focusing on Prototype 1 as the primary basis for development, we can maintain a cohesive and streamlined approach, ensuring that the final product meets our standards and delivers a consistent user experience.

However, while we've chosen Prototype 1 as the foundation, we remain open to incorporating elements from other prototypes if they align with our goals and enhance the overall quality of the game. Any such decisions would be made based on careful evaluation and consideration of their impact on the project timeline, resource allocation, and alignment with our vision for Fiery Dragons.

In summary, Prototype 1 serves as an excellent foundation for Sprint 3 due to its completeness, rationale, understandability, extensibility, and potential for user engagement. Building upon this strong base, we can confidently proceed with further development, knowing that we have a solid framework to support the evolution of Fiery Dragons.

# CRC Cards

## Cards Class

Purpose: This class represents dragon cards, including storing details of its position and whether it is flipped or not. It also handles the loading of the front and back of the dragon cards from the provided image paths. This class also draws the card on the screen, depending on whether it is flipped.

| Cards | |
|---|---|
| Responsibilities | Collaborations |
| Store the x,y coordinates and size of the dragon cards | GamePanel |
| Store the front and back images of the | ImageIO |

| | |
|---|---|
| card | |
| Flip the card | |
| Check if a point is within the card's boundaries | |
| Draw the card on the screen | Graphics2D |

## Player Class

Purpose: This class represents a player in Fiery Dragons. It stores details such as the player's turn, images of the player's dragon and methods to switch the turns between the players. It also draws the player's dragon on the window and sets the current position of the token.

| Player | |
|---|---|
| Responsibilities | Collaborations |
| Store the player's details | GamePanel |
| Load the player's dragon token image | ImageIO |
| Manage the player's turn | PlayerTurn |
| Check if a point is within the player's dragon's boundaries | |
| Set the default values and current position of the player's dragon | |
| Draw the player's dragon on the screen | Graphics2D |

## TileManager Class

Purpose: The class manages the tiles in Fiery Dragons, which including storing an array of Tile objects, loading images from provided image paths, loading map from a text file and drawing the tiles on the screen.

| TileManager | |
|---|---|
| Responsibilities | Collaborations |
| Store the details of the tiles | GamePanel |
| Load the tile images | ImageIO |

| | |
|---|---|
| Load the map from a text file | BufferedReader |
| Draw the tiles on the screen | Graphics2D |

## FindCoordinatesWithValue Class

Purpose: Designed to find and store the coordinates of a specific value in the map file. Specifically, it stores a list of coordinates where the value ‘01’ is found in the map and reads the map file line by line, splitting each line into characters. It also provides a method to retrieve the list of coordinates.

| FindCoordinatesWithValue | |
|---|---|
| Responsibilities | Collaborations |
| Store the coordinates where ‘01’ is found in the map | BufferedReader |
| Read the map file and find the coordinates | |
| Provide a method to retrieve the list of coordinates | |

## AssetSetter Class

Purpose: Manages the placement of game objects on the game board. It also retrieves the object at a given coordinate, while storing the map of objects for GamePanel.

| AssetSetter | |
|---|---|
| Responsibilities | Collaborations |
| Store the details of the game assets (FindCoordinatesWithValue) | GamePanel |
| Place game assets on the game board | DragonCards |
| Retrieve the game asset at a given coordinate | |

## GamePanel Class

Purpose: Serves as the graphical user interface for the game. It is responsible for setting up game elements, managing user input through ‘mouseClicked’, updating game state and controlling the logic of the game.

| GamePanel | |
|---|---|
| Responsibilities | Collaborations |
| Render game elements | Graphics2D, TileManager, DragonCards, CircleManager, Player classes (GreenDragon, BlueDragon, PinkDragon, PurpleDragon), Cards |
| Manage user input | MouseEvent |
| Update game state | AssetSetter, TileManager, DragonCards, Player classes (GreenDragon, BlueDragon, PinkDragon, PurpleDragon), Cards |
| Control game flow | Player classes (GreenDragon, BlueDragon, PinkDragon, PurpleDragon), WinningPage |
| Initialise game objects | AssetSetter, ImageLoader, CircleManager, Util classes (FindCoordinatesWithValue) |
| Start and manage game thread | Thread |

## Alternative Distribution of Responsibilities

In the design of these classes, alternative distributions of responsibilities were considered but ultimately rejected for various reasons. For instance, the Cards class could have been responsible for drawing itself on the screen, but this responsibility was given to the Player class to centralise the rendering logic and reduce coupling. Similarly, the TileManager class could have been responsible for loading the map file, but this was delegated to the FindCoordinatesWithValue class to separate the concerns of tile management and map loading. The AssetSetter class could have taken on the responsibility of determining the type of object to place based on the count of each object, but this was kept within the setObject() method to encapsulate the object placement logic. These decisions were made to promote high cohesion, low coupling, and better maintainability in the codebase.

# Description of Executable

Purpose: Instructions on how to build and run the executable of this software prototype.

Target Platform(s): The executable is using Java. All platforms, including MacOS, Linux and Windows, are able to run the file as long as an appropriate JDK is installed on the user's machine.

# Instructions for Executable

**Download JDK**

1. Install JDK21 on whichever machine that will be running the software.

- Linux: [Download JDK21](https://www.oracle.com/au/java/technologies/downloads/#java21)

- macOS: [Download JDK21](https://www.oracle.com/au/java/technologies/downloads/#jdk21-mac)

- Windows: [Download JDK21](https://www.oracle.com/au/java/technologies/downloads/#jdk21-windows)
    (Example link for Windows download: [JDK 21 Windows Installer](https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.exe))

2. Complete the installation of the Java JDK by following the steps of the installer.

**SDK Set up**

3. Set up the SDK for the file by going to:
  - "File -> Project Structure..." (Shortcut: "Ctrl + Alt + Shift + S")
  - Click on the "Project" Tab.

4. Under project, click on the drop-down bar next to the sub-heading SDK.

5. Click on "+ ADD SDK" then "JDK...", find the directory of the downloaded JDK and click on the whole file to add the JDK to the project.

6. Click on "Apply" then "OK" to finish the SDK set up.

**Build Executable**

7. Set up the SDK for the file by going to:

- "File -> Project Structure..." (Shortcut: "Ctrl + Alt + Shift + S")
- Click on the "Artifacts" Tab.

8. On the top left of the window, a "+" symbol will be present, add a new artifact by clicking this.

9. Click on the JAR title.

10. Then click on the "From modules with dependencies..." title, a new window will appear.

11. Next to the title "Main Class:" click on the folder icon and either search the main class, however, it should appear on the window "Main (main)". Click ok after selecting the main class.

12. Click on "Apply" then "OK" to finish making an artifact set up.

13. After navigating to the "Build" tab on the top.

14. Once it has been found, click on the tab and find the "Build Artifacts.." and click on that option.

15. A pop-up will appear, click build.

16. An executable has been created, it will be located in the out folder:

```
fieryDragons
 ┗ out
  ┗ artifacts
  ┗ fieryDragonsGame_jar
  ┗ fieryDragonsGame.jar
```

**Run Executable**

Option 1:

17. Locate the jar file and double click to open the executable.

Option 2:

18. Change directory in the terminal to where the jar file is being kept. Write in the terminal:

```
cd
C:\Users\annie\IdeaProjects\CL_Monday06pm\out\artifacts\fieryDragonsGame_jar
```

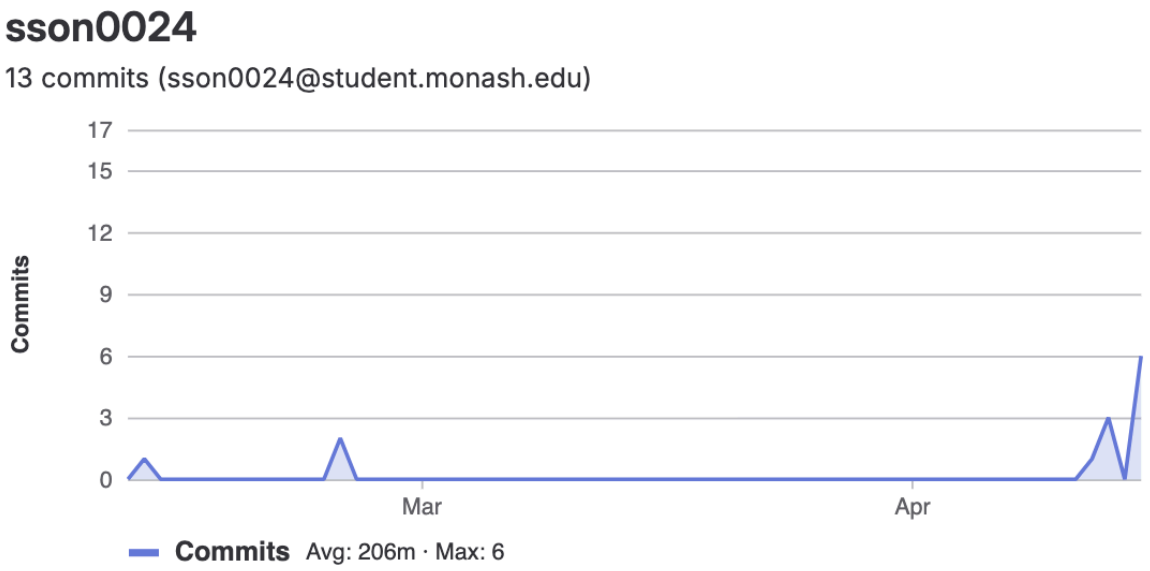
Change based on where the user has saved the jar file.

19. In the terminal write:

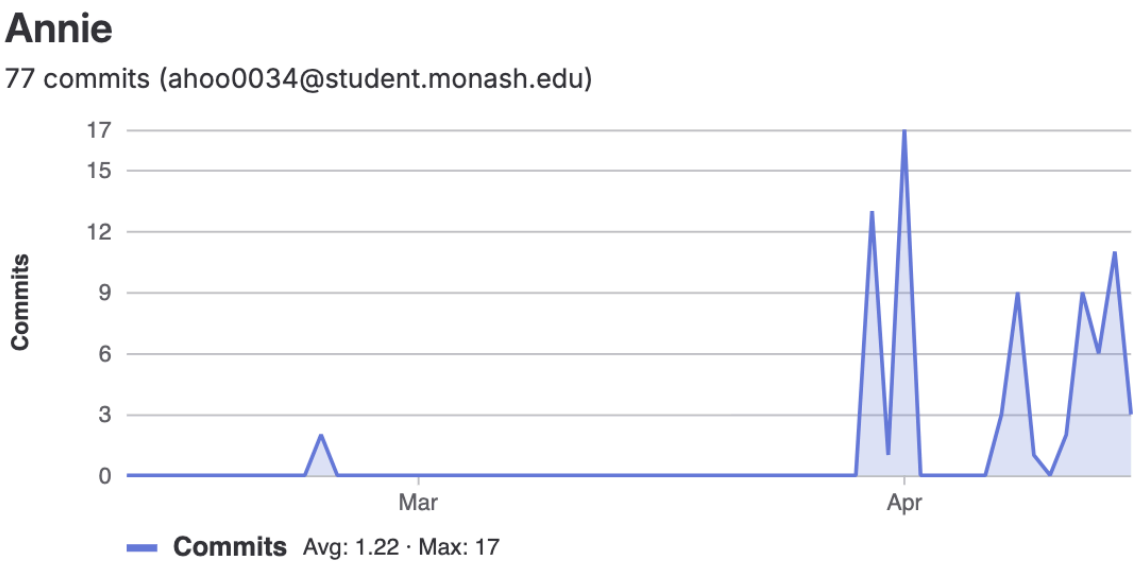
    java -jar fieryDragonsGame.jar

## Contributor Analytics

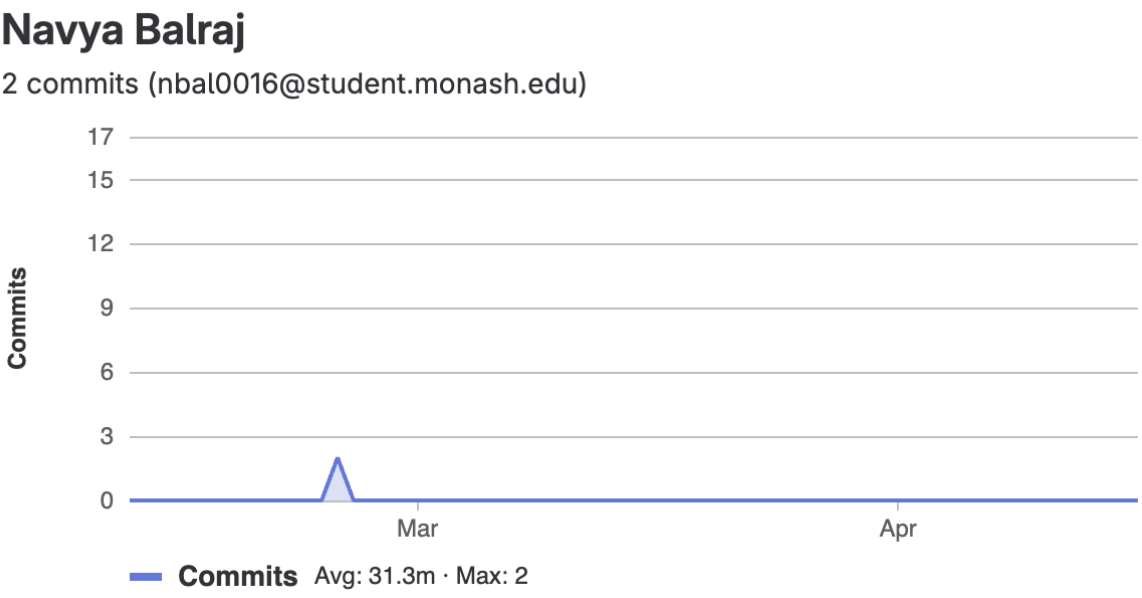
### Selsa Sony

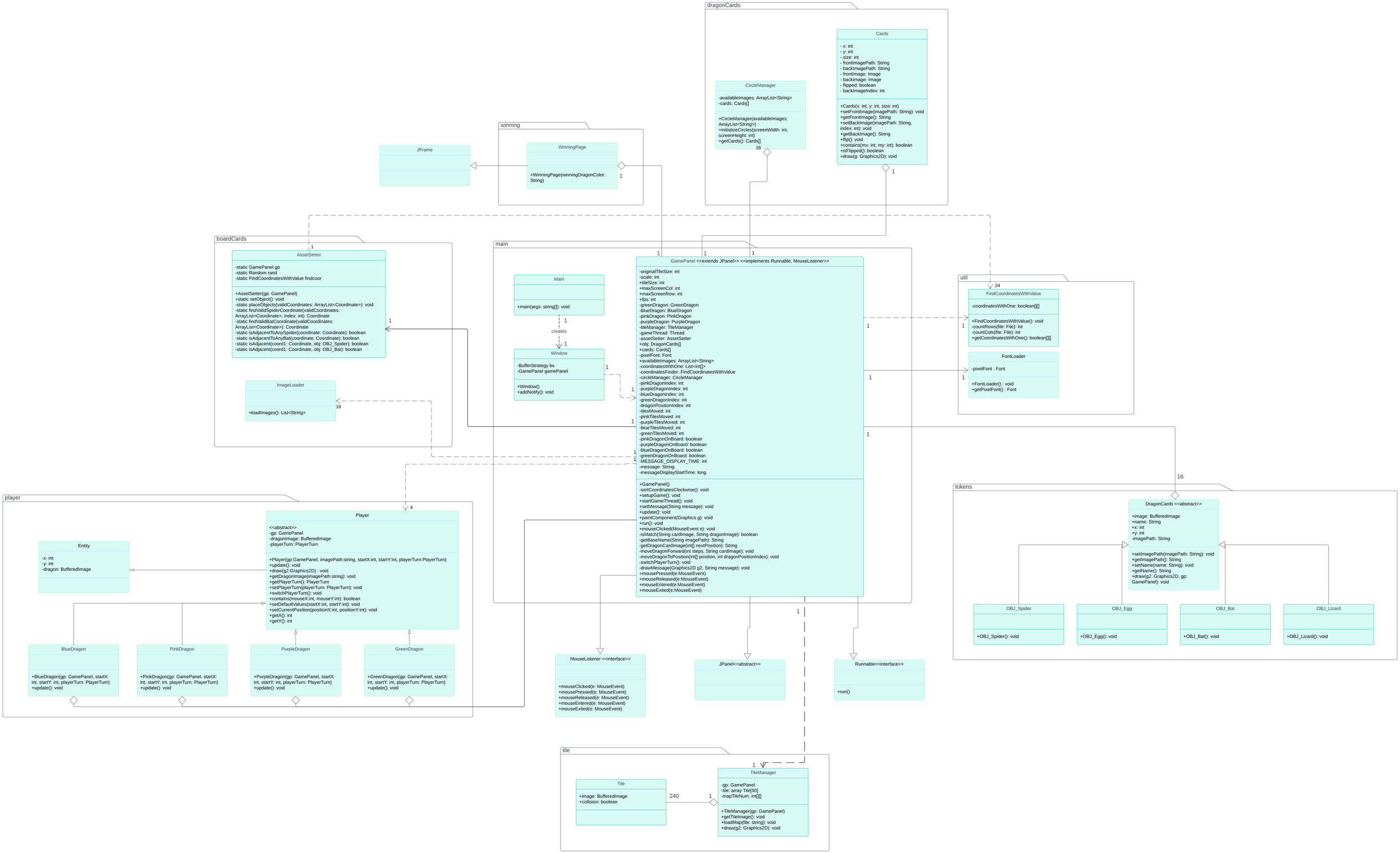


Annie Ho




Navya Balraj




dragonCards
Cards
- x: int
- y: int
- size: int
- frontImagePath: String
- backImagePath: String
- frontImage: Image
- backImage: Image
- flipped: boolean
- backImageIndex: int
+Cards(x: int, y: int, size: int)
+setFrontImage(imagePath: String): void
+getFrontImage(): String
+setBackImage(imagePath: String, index: int): void
+getBackImage(): String
+flip(): void
+contains(mx: int, my: int): boolean
+isFlipped(): boolean
+draw(g: Graphics2D): void
CircleManager
-availableImages: ArrayList<String>
-cards: Cards[]
+CircleManager(availableImages: ArrayList<String>)
+initializeCircles(screenWidth: int, screenHeight: int)
+getCards(): Cards[]
16
1
winning
JFrame
WinningPage
+WinningPage(winningDragonColor: String)
1
boardCards
AssetSetter
-static GamePanel gp
-static Random rand
-static FindCoordinatesWithValue findcoor
+AssetSetter(gp: GamePanel)
+static setObject(): void
-static placeObjects(validCoordinates: ArrayList<Coordinate>): void
-static findValidSpiderCoordinate(validCoordinates: ArrayList<Coordinate>, index: int): Coordinate
-static findValidBatCoordinate(validCoordinates: ArrayList<Coordinate>): Coordinate
-static isAdjacentToAnySpider(coordinate: Coordinate): boolean
-static isAdjacentToAnyBat(coordinate: Coordinate): boolean
-static isAdjacent(coord1: Coordinate, obj: OBJ_Spider): boolean
-static isAdjacent(coord1: Coordinate, obj: OBJ_Bat): boolean
ImageLoader
+loadImages(): List<String>
16
main
Main
+main(args: string[]): void
creates
Window
-BufferStrategy bs
-GamePanel gamePanel
+Window()
+addNotify(): void
GamePanel <<extends JPanel>> <<implements Runnable, MouseListener>>
-originalTileSize: int
-scale: int
+tileSize: int
+maxScreenCol: int
+maxScreenRow: int
+fps: int
-greenDragon: GreenDragon
-blueDragon: BlueDragon
-pinkDragon: PinkDragon
-purpleDragon: PurpleDragon
-tileManager: TileManager
-gameThread: Thread
-assetSetter: AssetSetter
+obj: DragonCards[]
+cards: Cards[]
-pixelFont: Font
+availableImages: ArrayList<String>
-coordinatesWithOne: List<int[]>
-coordinatesFinder: FindCoordinatesWithValue
-circleManager: CircleManager
-pinkDragonIndex: int
-purpleDragonIndex: int
-blueDragonIndex: int
-greenDragonIndex: int
-dragonPositionIndex: int
-tilesMoved: int
-pinkTilesMoved: int
-purpleTilesMoved: int
-blueTilesMoved: int
-greenTilesMoved: int
-pinkDragonOnBoard: boolean
-purpleDragonOnBoard: boolean
-blueDragonOnBoard: boolean
-greenDragonOnBoard: boolean
-MESSAGE_DISPLAY_TIME: int
-message: String
-messageDisplayStartTime: long
+GamePanel()
-sortCoordinatesClockwise(): void
+setupGame(): void
+startGameThread(): void
+setMessage(String message): void
+update(): void
+paintComponent(Graphics g): void
+run(): void
+mouseClicked(MouseEvent e): void
-isMatch(String cardImage, String dragonImage): boolean
-getBaseName(String imagePath): String
-getDragonCardImage(int[] nextPosition): String
-moveDragonForward(int steps, String cardImage): void
-moveDragonToPosition(int[] position, int dragonPositionIndex): void
-switchPlayerTurn(): void
-drawMessage(Graphics2D g2, String message): void
+mousePressed(e:MouseEvent)
+mouseReleased(e:MouseEvent)
+mouseEntered(e:MouseEvent)
+mouseExited(e:MouseEvent)
util
24
FindCoordinatesWithValue
-coordinatesWithOne: boolean[][]
+FindCoordinatesWithValue(): void
-countRows(file: File): int
-countCols(file: File): int
+getCoordinatesWithOne(): boolean[][]
FontLoader
-pixelFont : Font
+FontLoader() : void
+getPixelFont() : Font
player
Player
<<abstract>>
-gp: GamePanel
-dragonImage: BufferedImage
-playerTurn: PlayerTurn
+Player(gp:GamePanel, imagePath:string, startX:int, startY:int, playerTurn:PlayerTurn)
+update(): void
+draw(g2:Graphics2D) : void
+getDragonImage(imagePath:string): void
+getPlayerTurn(): PlayerTurn
+setPlayerTurn(playerTurn: PlayerTurn): void
+switchPlayerTurn(): void
+contains(mouseX:int, mouseY:int): boolean
+setDefaultValues(startX:int, startY:int): void
+setCurrentPosition(positionX:int, positionY:int): void
+getX(): int
+getY(): int
4
Entity
-x: int
-y: int
-dragon: BufferedImage
BlueDragon
+BlueDragon(gp: GamePanel, startX: int, startY: int, playerTurn: PlayerTurn)
+update(): void
PinkDragon
+PinkDragon(gp: GamePanel, startX: int, startY: int, playerTurn: PlayerTurn)
+update(): void
PurpleDragon
+PurpleDragon(gp: GamePanel, startX: int, startY: int, playerTurn: PlayerTurn)
+update(): void
GreenDragon
+GreenDragon(gp: GamePanel, startX: int, startY: int, playerTurn: PlayerTurn)
+update(): void
MouseListener <<interface>>
+mouseClicked(e: MouseEvent)
+mousePressed(e: MouseEvent)
+mouseReleased(e: MouseEvent)
+mouseEntered(e: MouseEvent)
+mouseExited(e: MouseEvent)
JPanel<<abstract>>
Runnable<<interface>>
+run()
tokens
DragonCards <<abstract>>
+image: BufferedImage
+name: String
+x: int
+y: int
-imagePath: String
+setImagePath(imagePath: String): void
+getImagePath(): String
+setName(name: String): void
+getName(): String
+draw(g2: Graphics2D, gp: GamePanel): void
OBJ_Spider
+OBJ_Spider(): void
OBJ_Egg
+OBJ_Egg(): void
OBJ_Bat
+OBJ_Bat(): void
OBJ_Lizard
+OBJ_Lizard(): void
tile
Tile
+image: BufferedImage
+collision: boolean
240
1
TileManager
-gp: GamePanel
-tile: array Tile[50]
-mapTileNum: int[][]
+TileManager(gp: GamePanel)
+getTileImage(): void
+loadMap(file: string): void
+draw(g2: Graphics2D): void

# Sprint 3 - Contribution Log for Team 24

| Task | Student Name | Date Started | Date Completed | Contribution Details |
|---|---|---|---|---|
| Fix the initial board game and fixed the directory for the images as well as refactored the GamePanel class | Annie Ho | 08/05/2024 | 09/05/2024 | - Fixed the code for the directory for the images and other files<br>- an error in the original code which caused it to not run on other team member's computers was fixed<br>- Refactored GamePanel so the length of it was more manageable |
| CRC Cards for six main classes | Selsa Sony | 14/05/2024 | 15/05/2024 | - Created CRC for Card, Player, GamePanel, AssetSetter, FindCoordinatesWithValue and TileManager classes |
| Review of prototype 2 | Annie Ho | 14/05/2024 | 14/05/2024 | - Reviewed prototype completed by Selsa in Sprint 2<br>- Based on the assessment criteria compared her prototype against the generated criteria |
| Review of Prototype 1 | Selsa Sony | 14/05/2024 | 15/05/2024 | - Assessed Annie's prototype based on criteria and metrics agreed by the group |
| Class Diagram | Annie Ho<br>Selsa Sony | 14/05/2024 | 15/05/2024 | - Refactored Annie's Sprint 2 Class Diagram to include new classes and relationships |
| Moving Dragons | Annie Ho | 11/05/2024 | 12/05/2024 | - Based on the flipping of cards the dragons would move a space on the board<br>- Created a method which enabled the dragons to be added onto the board from their cave<br>- Dragon will move 1, 2 or 3 spaces forward or 1 or 2 |

| | | | | |
|---|---|---|---|---|
| | | | | spaces backwards depending on the card flipped |
| Match dragon cards to move | Annie Ho | 13/05/2024 | 15/05/2024 | - Will check if there is a match with the card flipped and the next tile position<br>- When a skull card is flipped the dragon will be prompted to move back 1 or 2 spaces based on how many skulls they flipped<br>- Flipping a card that had multiple of the image (ie 2 eggs) checked if the next 2 tiles had that same image |
| Player's Turn | Annie Ho<br>Selsa Sony | 13/05/2024 | 14/05/2024 | - Changed the player's turn according to either correct or incorrect card flipped<br>- Displayed the text for which dragon's turn it is to move |
| Winning Page | Selsa Sony | 13/05/2024 | 14/05/2024 | - Added fonts and images for winning page |
| Collision Detect | Selsa Sony | 14/05/2024 | 14/05/2024 | - Players are unable to go on the same tile<br>- will call the switchPlayerTurn method when there's another player on the targeted tile |
| Player creation | Annie Ho | 13/05/2024 | 14/05/2024 | - created the different player classes through the formation of the abstract Player class<br>- Different colour dragons inherits from player class |
| Change of game function | Annie Ho | 16/05/2024 | 16/05/2024 | - the player moves based on the current tile that they are sitting on<br>- will move 1, 2 or 3 |
| Review of Prototype 3 | Navya Balraj | 15/05/2024 | 16/05/2024 | - reviewed Yokabit's prototype for Sprint 2 based on the metrics |
| Review of Prototype 4 | Yokabit Fesshaye | 16/05/2024 | 16/05/2024 | - reviewed Navya's Prototype |