FIT3077 Assignment 3

Team 149 - Object Oriented Dragons

Installation and execution guide at the end of this document

Review of Sprint 2 Tech Based Prototypes

Assessment Criteria Definitions

Completeness of the solution direction

This criteria for functional correctness was based on key functionalities that were stated in Sprint 2. For each key functionality accounted for in the design, an assessee would get a score. As a result of counting the number of key functionalities accounted for, this criteria was an absolute measurement. For functional completeness, assessees would get a score for the degree to which the set of functions in the design would cover all specified tasks, or how much functionality the defined functions cover. A ranking in order results in an ordinal measurement for this criteria.

Functional correctness

- ☐ set up the initial game board (including randomised positioning for dragon cards)
- ☐ flipping of dragon ("chit") cards
- ☐ movement of dragon tokens based on their current position as well as the last flipped dragon 1 card
- ☐ change of turn to the next player
- ☐ winning the game.

Functional completeness

1. Covers none - The defined functions cannot be used to cover any required functionality
2. Covers some - The defined functions can be used to cover some of the required functionality
3. Covers most - The defined functions can be used to cover most of the required functionality
4. Covers all - The defined functions can be used to cover all of the required functionality

Rationale behind the chosen solution direction

This criteria covered functional appropriateness, and determined how well the functions facilitate the execution of user stories, game rules and design patterns. Marked as a ranking, this criteria was an ordinal scale.

Functional appropriateness

1. Not appropriate - the design of the system is completely unable to facilitate the functionality needed for the system
2. Somewhat appropriate - the design of the system is able to facilitate some of the functionality needed for the system
3. Very appropriate - the design of the system is able to facilitate most of the functionality needed for the system
4. Extremely appropriate - the design of the system is able to facilitate all if not all of the functionality needed for the system

Understandability of the solution direction [Appropriateness recognizability]

This criteria covers appropriateness recognizability, and determines how understandable the facilitation of the execution of the key game features is. The complexity of the rationale is taken into consideration as a ranking, making this an ordinal scale.

Appropriateness recognizability

1. Not understandable - the rationale behind the design of the system is unable to be understood, or would require a disproportionate amount of time to grasp knowledge of implementation
2. Somewhat understandable - the rationale behind the design of the system is somewhat able to be understood and requires slightly more than expected amount of time to grasp knowledge of implementation
3. Very understandable - the rationale behind the design of the system is able to be understood quite easily and requires the expected amount of time to grasp knowledge of implementation
4. Extremely understandable - the rationale behind the design of the system is able to be understood extremely easily and requires much less time than expected to grasp knowledge of implementation

Extensibility of the solution direction

This point accounts for the modifiability of the current Fiery Dragons design. Classes and methods, along with relationships between classes are assessed on an ordinal scale. Consideration for future extensions in sprint four result in a greater ranking.

Modifiability

1. Not extensible - The solution is not extendable, would have to change overall structure
2. Somewhat extensible - The solution is somewhat extensible, would have to change some structure
3. Very extensible - The solution is mostly extensible, would have to make minor changes to structure
4. Extremely extensible - The solution is completely extensible, no structural changes required

Quality of the written source code

This criteria covers the maintainability and quality of the assessees written code. This is judged by referring to the Java Style Guide standards that can be found here:

https://google.github.io/styleguide/javaguide.html

Reliance on Case analysis and downcasts were ranked on an Absolute scale, whereas the coding standards can be measured ordinally.

Maintainability

1. Doesn't conform - the coding standards used does not conform to Google Java Coding Guidelines
2. Somewhat conforms - the coding standards used somewhat conform to Google Java Coding Guidelines
3. Mostly conforms - the coding standards used mostly conforms to guidelines as specified by the Google Java Style Guide
4. Conforms - the coding standards used completely conforms to guidelines as specified by the Google Java Style Guide

Aesthetics of the user interface [User engagement];

The aesthetics of the user interface not only cover the look of the game, but the user engagement, and how likely a player is to continue playing the game from its visual appearance. This was ranked ordinally, however this criteria may not be an accurate representation, as assessees did not have the same functionality.

User engagement

1. Not likely - User interface is not visually appealing, user would not want to continue playing
2. Somewhat likely - User interface is somewhat visually appealing, user would continue playing
3. Very likely - User interface is visually appealing, user would like to continue playing

Usability [Operability, self descriptiveness, user assistance]:

The usability of the game refers to the operability, the self descriptiveness and any relevant user assistance provided to players when playing. Again, this is measured ordinally depending on how easy it is to operate the game, and how understandable and accommodating the game is for various users.

Operability, Self descriptiveness, User assistance

1. Not understandable - User does not know how to or cannot navigate the game
2. Somewhat understandable - The user can navigate the game with difficulty
3. Mostly understandable - The user can navigate the game with little difficulty
4. Extremely understandable - The user can navigate the game intuitively

Review of Prototypes

Group Member 1: Krishna

Completeness of the solution direction [Functional correctness, Functional completeness]

- Functional Correctness: 4
  The design does not account for edge cases when moving the player. The design needs to include checking if the player can move when they are moving past the cave and winning the game.

- Functional Completeness: 5
  All core game features are implemented and designed.

Rationale behind the chosen solution direction [Functional appropriateness]

- Very appropriate
  The design covers all functionality, and each class has its own purpose without obvious god classes. The deck class currently provides no value and purpose. After clarification, it was understood that it was implemented with future extensions in mind. No design pattern was explicitly followed/utilised in this context which may introduce a few problems that are not apparent as of now.

Understandability of the solution direction [Appropriateness recognizability]

- Somewhat understandable
  Some features of the game can be difficult to understand at times, requiring a deeper look at the sequence diagrams.

Extensibility of the solution direction [Modifiability];

- Very extensible
  Various classes such as deck class, MapPiece and NormalPiece allow for various extensions with regards to how the chitcards and the game board can be modified. All extensions with regards to UI/UX can be extended through the JavaFX controllers. Extensions dealing with turns and modification of turns can also be implemented by potentially modifying the TurnManager class.

Quality of the written source code [Maintainability]

- Mostly conforms
  Many coding standards were followed., nothing obvious can be pointed out that is missing. Very little case analysis and no downcasts were used, however some repetition can be found throughout the code especially in the controller classes which can be improved.

Aesthetics of the user interface [User engagement]

- Not likely
  The user interface is a bit lacking, and the overall design of the application and the game board is simple rectangles rather than circular.

Usabiliy [Operability, self descriptiveness, user assistance]:

- Mostly understandable
  Very small number of buttons for control, easily guides the user how to navigate and play the game. Information is very clearly presented without excessive interactions. The user is able to navigate and use the application without queries, however little to no user assistance is provided.

Summary of key findings:

The key features of the Fiery Dragons board game are implemented and designed to a high level. However when implementing moving the player, the design has not considered checking for moving past a cave that the player will move to when winning the game. No apparent design pattern was utilised, and a few classes appear to have functions with the context of the base Fiery Dragons game. These classes are implemented with respect to future extensions that may be added to the game

Group Member 2: Jeffrey

Completeness of the solution direction [Functional completeness; functional correctness]

- Functional Correctness: 5
  Includes all required features
-
- Functional Completeness: 5
  All functions can be used to implement required features

Rationale behind the chosen solution direction [Functional appropriateness]

- Very appropriate
  Good logic behind all functions, and all classes have a purpose and no unnecessary classes. The Deck class currently provides no value, however prepares for later extension. There is an appropriate use of static class for Turn.

Understandability of the solution direction [Appropriateness recognizability]

- Very understandable
  The chosen approach is easy to understand with little to no clarifications required. The Sequence diagrams are logical and provide clear insight into the chosen approach.

Extensibility of the solution direction [Modifiability]

- Very extensible
  There is an extensible board design by utilising Tiles and MapPieces, keeping future sprints in mind. With regards to the Deck design, while it currently serves no purpose, it can be used later for game extensions.

Quality of the written source code [Maintainability]

- Mostly conforms
  There are no obvious deviations from coding standards and appropriate class and function names are used. In line comments have been used where necessary. Could include Javadoc for more complex classes as it is a little difficult to understand their workings. Some case analysis is used in controller class, however this may be necessary and unavoidable.

Aesthetics of the user interface [User engagement]

- Somewhat likely
  The UI representation of the game is similar to the original board game design which is a positive. Could include animal designs for dragon cards and tiles to more accurately reflect the board game. The Board tiles contain various shades rather than a single block colour for the entire board, which makes it easy to distinguish Tiles. Finally, it is easy to identify which player's turn it is.

Usability [Operability, self descriptiveness, user assistance]:

- Somewhat understandable
  Using the initial landing page is clear and easy to understand, the slider is intuitive and the start game button is clearly labelled. The implementation assumes knowledge on how the Fiery Dragons board game works to play, and clicking on cards to progress with turns. No user assistance/game instructions are provided.

Summary of key findings:

There are good UI design elements, having a clear circular board with various coloured tiles. However, images could be added instead of words to increase user engagement. The design is extensible, and the sprint 3 implementation should make use of the MapPiece, Tile, DragonCard and Deck design choices. Ensure that complex methods in Sprint 3 implementation have Javadoc for understandability in following sprints and collaboration.

Group Member 3: Zilei

Completeness of the solution direction [Functional completeness; functional correctness]

- Functional Correctness: 5
  All required features are present.
- Functional Completeness: 5
  All functions can be used to implement all requirements.

Rationale behind the chosen solution direction [Functional appropriateness]

- Very appropriate
  The use of Board as a singleton is intuitive and its implementation is appropriate. The further turn logic is easy to implement with use of the chain of responsibility pattern and makes sense in its context. The design of the Dragon card and map piece features are harder to extend due to current implementation, and classes should be considered for these. A third controller class may also be required for the settings scene.

Understandability of the solution direction [Appropriateness recognizability]

- Very understandable
  The design is easy to understand and relationships can be inferred accurately. The chain of responsibility and singleton design pattern requires a little further reading beyond just looking at the system if the user is unfamiliar, however they are excellent design choices in the context of the game.

Extensibility of the solution direction [Modifiability]

- Somewhat extensible
  The turns are extremely extensible due to the utilisation of the chain of responsibility. The Dragon cards and the Map Pieces will require more modification to the code for future extensions compared to if implemented using inheritance and separate classes.

Quality of the written source code [Maintainability]

- Mostly conforms
  There are no obvious deviations from coding standards. Appropriate function names and class names are used, and there is documentation present for complex functions, making it more understandable.

Aesthetics of the user interface [User engagement]

- Somewhat likely
  The game board closely replicates real world design of the game, and includes hand-drawn pictures of animals in the game. The text alignment in settings page can be improved and the colour scheme is very bright with harsh white, yellow and orange colours.

Usability [Operability, self descriptiveness, user assistance]

- Mostly understandable
  The buttons have clear affordances, and the slider follows natural mapping, 2 to 4 players from left to right. Clicking on a chit card flips it as expected, however the user is never told that the chit card can be flipped by clicking, which assumes that the user is already familiar with the game.

Summary of key findings

The approach of displaying images for each tile and dragon card is ideal, displaying an excellent utilisation of design patterns. The Turn logic using chain of responsibility design pattern is very applicable and also extensible if there are future conditions for whether a player should move. The current implementation of dragon cards and map pieces need to be modified for feature extensions, as they do not have their own classes. This will make sprint 4 extensions of custom board sizes difficult to implement.

Combination of ideas from each design

- Chain of responsibility from Zilei's (Turn and turnLogic classes); this design pattern is a great way to ensure that all turn logic is readable and extendable, each turnLogic class has clearly defined responsibilities.
- Singleton Board class from Jeffrey's; intuitive way for the entire system to be able to access information stored by the Board class, and ensures each reference to the Board class refers to the same object.
- (Modified) MapPiece, Tile, Deck implementation from Krishna's; creating these classes ensure that the source code is primed for extensibility in Sprint 4.
- All of our game/controller/main classes are similar

New ideas for Sprint 3 prototype

- Add an end turn button for situations where players want to end turn early if they do not want to risk going backwards.

2. Object-Oriented Design

Class Responsibility Cards

Board
The Board class in this design serves to closely mirror how the Fiery Dragons game would function in real life. Its role in this system is to be the ‘container’ for all relevant elements that would need to be interacted with, meaning it contains all relevant Dragon Cards by containing Deck and MapPieces.

The Board Class is implemented using the Singleton Design Pattern, meaning that only one instance of Board can be created, and ensures that all other classes that need to reference Board are able to, i.e. BoardController, Game, etc. This also means that the Board object is only initialised when it is first referenced.

Giving the Board class only these responsibilities means that we can avoid a god class, as currently it already stores all the most important components of the gameboard. By diverting all ‘logic handling’ to its collaborators, we can ensure that we maintain Single Responsibility Principle.

Responsibilities:	**Collaborators**
● Creating initial map pieces ● Creating initial deck	Game, MapPiece, Deck, Turn, BoardController

MapPiece
This class is used to represent the 8 map pieces that when combined together, make up the board. Each MapPiece will be composed of 3 Tiles. It has been created for the sake of abstraction, and to improve the extensibility of the Fiery Dragons game, keeping in mind the additions in the final sprint.

Each MapPiece will be responsible for initialising 3 tiles. Combined together, the 8 MapPieces composed of 3 Tiles each will make up the 24 Tile Board where the game is played. While the option of initialising this in the Board class was considered, it was rejected to uphold the Single Responsibility Principle. Additionally, having a MapPiece class would allow for future extensions specific to MapPieces, such as changing the number of Tiles per MapPiece.

Responsibilities:	**Collaborators**
● Initialising Tiles	Board, Tile

Player
This class is solely responsible for the action of flipping a chit card. When a card is flipped, this class will be responsible for telling the board what was flipped and thus letting the Turn class know how to proceed with the turns.

An alternative was to have the BoardController class take responsibility for flipping the card, however the controller class's main purpose is to render, thus the player class was handed the responsibility of flipping chit cards.

Responsibilities:	**Collaborators**
● Flipping the chit cards	Turn

Game
This class is responsible for handling all required game initialisation and allowing a player to flip a chit card. Once a chit card is flipped, this information is then passed to the Turn class for turn logic handling.

Previously, the Game class would also handle the logic for determining whether a player had won, i.e., checkForWin(), however this responsibility was eventually given to turn logic handling, where the Chain of Responsibility pattern is used.

Responsibilities:	**Collaborators**
● Initialising initial board state given input settings, e.g., number of players ● Stores a Players array to keep track of players ● Handles player flipping a chit card ● Calls turn class to handle turn logic	Turn, Board, Player, StartApplication, SceneController

Turn
The main responsibility of this class is to play out a turn and handle any changing of turns from one player to another. For every chit card a player moves, the Turn class will figure out who is flipping the card from the Game class, and also get information from the Board class, which allows turns to be calculated and carried out. It will then end the turn and proceed to the next one.

It was considered that the Turn class should handle all of the turn logic such as checking if a player is able to move from one tile to another. This was then distributed to other specialist classes to comply with SRP and not overcomplicate classes. The handling of the next turn may also be completed by the Game class, but the Game class already had distinct responsibilities to perform.

Responsibilities:	**Collaborators**
● Play out the turn ● Change of player from one to another	Board, Game, TurnHandler

Deck This class is used to represent the 16 chit cards that are displayed in the centre of the Volcano. Its only responsibility is to initialise the chit cards with their default values. An alternative location for the initialising chit cards responsibility was the Board class, however to uphold SRP and for abstraction, the chit cards should be initialised here. This design choice additionally makes the game more extensible, should there be changes to types of Decks, or card allocations in Decks.	
Responsibilities: ● Initialising chit cards	**Collaborators** Board, TileType

Differences Between UML Class Diagram, CRC and source code

MapPiece

- A cave attribute of type Tile was added to store caves for MapPieces that contained them. Relevant getters and setters were also added

Player

- An integer distanceToCave attribute was added to keep track of each player's distance from their home cave. This was used to check win conditions.

Tile

- No longer contains playerId as player locations are now being tracked by Board class.

Board

- Now stores player location data and is responsible for moving players.
- New methods were added in order to fit functionality.

Game

- Does not keep track of player location, now keeps track of current player’s turn and iterates this to the next player for each turn.

Turn

- Now a singleton class as only one will ever be created and used. Able to be referenced in Player class as dependency.

BoardController

- Now a singleton in order to reference static methods.

SettingsController

- New class implemented to facilitate variables in game set up including number of players

WinSceneController

- New class implemented to support the rendering of winning the game.

Installation and execution guide

The target operating system is Windows 10 + 11
Java Standard Edition (JSE)/ Java Development Kit (JDK) is required to run Java programs. If Java/JDK 22 is not installed, install using the following link in section 1.
https://www3.ntu.edu.sg/home/ehchua/programming/howto/JDK_HowTo.html
A direct link to the download is linked below:
https://www.oracle.com/java/technologies/downloads/
Click on Windows, then any of the 3 options should work. I recommend x64 Installer.

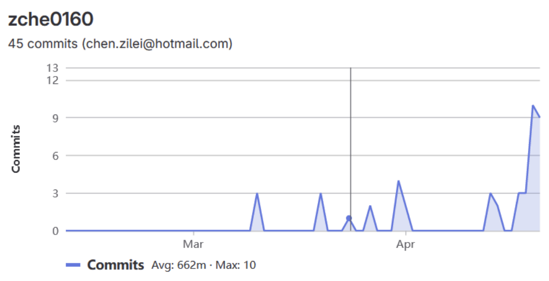
After installing Java 22, run the program by simply double clicking .jar executable

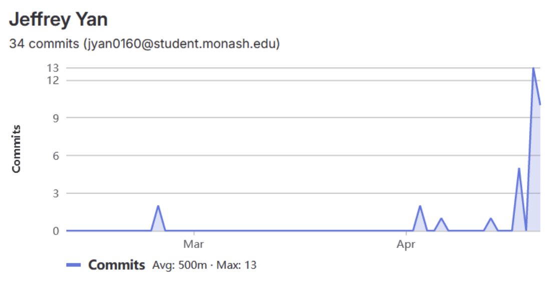
To create executable files, using Maven from IntelliJ, run the following:

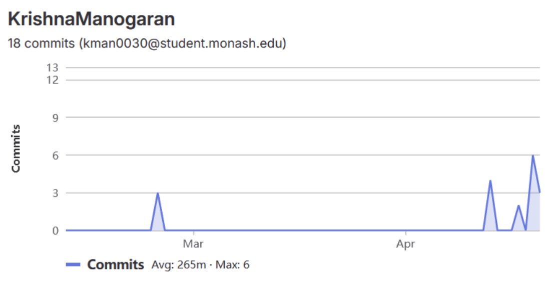
```
mvn clean
mvn install
```

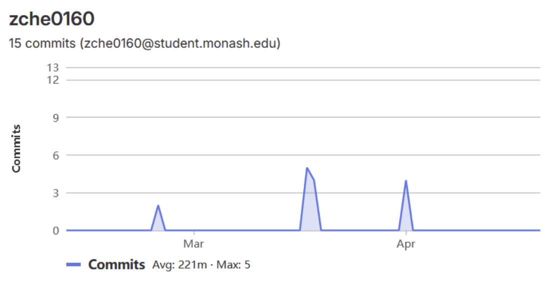
A target folder will appear outside the src folder. The .jar file we are looking for ends in -shaded.jar

Contributor Analytics


zche0160
45 commits (chen.zilei@hotmail.com)
Commits
Mar
Apr
Commits Avg: 662m · Max: 10


Jeffrey Yan
34 commits (jyan0160@student.monash.edu)
Commits
Mar
Apr
Commits Avg: 500m · Max: 13


KrishnaManogaran
18 commits (kman0030@student.monash.edu)
Commits
Mar
Apr
Commits Avg: 265m · Max: 6


zche0160
15 commits (zche0160@student.monash.edu)
Commits
Mar
Apr
Commits Avg: 221m · Max: 5