Semester 1, 2024

FIT3077 – Software engineering: Architecture and design

Sprint 3: Review of Sprint 2 Tech-based Software Prototypes

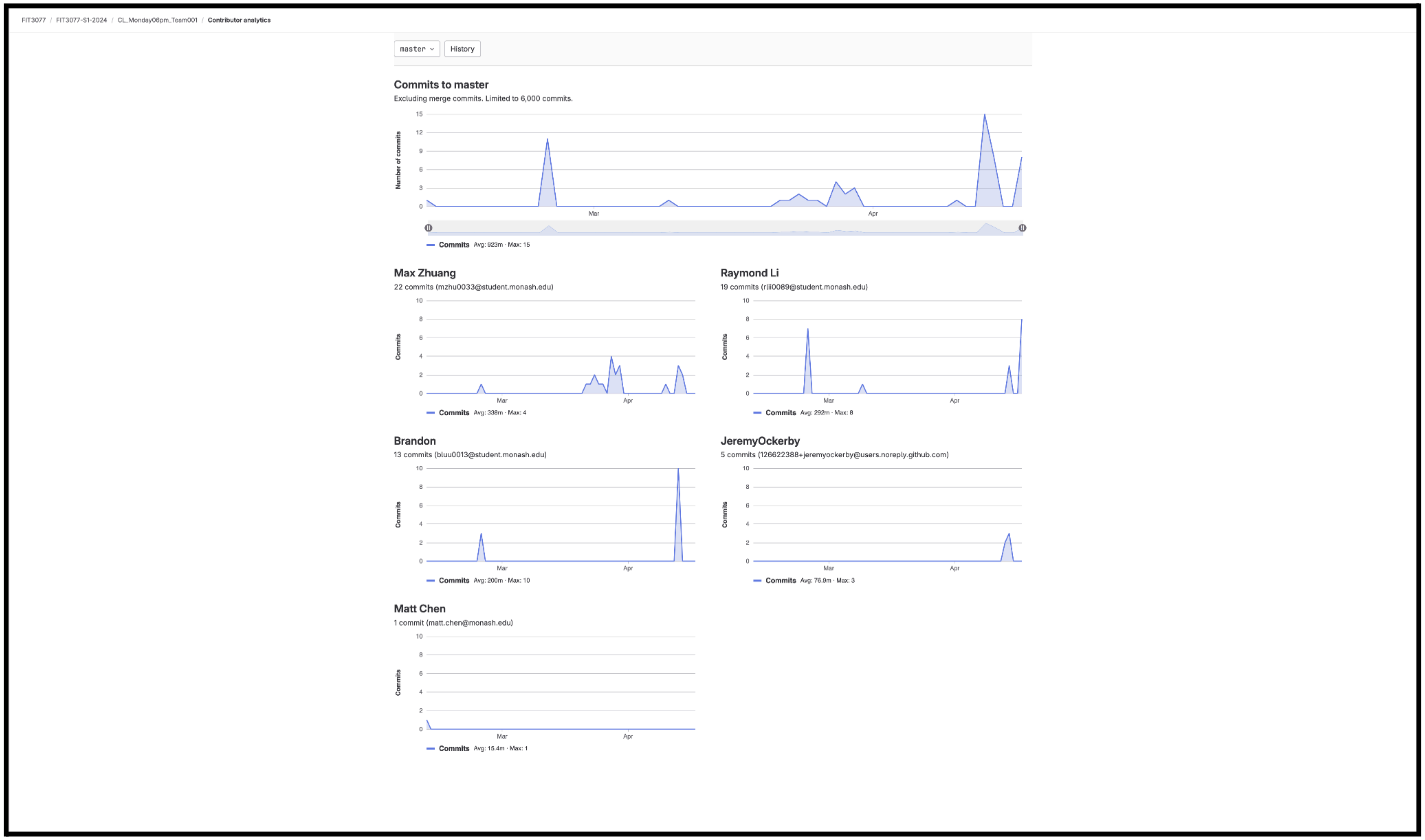
Team: Chitbusters

Members: Brandon Luu, Jeremy Ockerby, Max Zhuang, Raymond Li

Date of Submission: 20 May 2024

Table of Contents

Contributor Analytics


FIT3077 / FIT3077-S1-2024 / CL_Monday06pm_Team001 / **Contributor analytics**

Commits to master

Excluding merge commits. Limited to 6,000 commits.

Number of commits
15
12
9
6
3
0
Mar
Apr
Commits Avg: 923m · Max: 15

Max Zhuang

22 commits (mzhu0033@student.monash.edu)

Commits
Mar
Apr
Commits Avg: 338m · Max: 4

Raymond Li

19 commits (rlii0089@student.monash.edu)

Commits
Mar
Apr
Commits Avg: 292m · Max: 8

Brandon

13 commits (bluu0013@student.monash.edu)

Commits
Mar
Apr
Commits Avg: 200m · Max: 10

JeremyOckerby

5 commits (126622388+jeremyockerby@users.noreply.github.com)

Matt Chen

1 commit (matt.chen@monash.edu)

Accurate as of 18/05/24

Sprint 2 Review

Review Criteria

Functional Completeness/Correctness

Number of key functionalities covered and functional correctness of implemented features.

Functional Appropriateness

Clarity of design decisions and the adherence to game requirements.

Appropriateness Recognisability

Clarity of code structure, ease of navigation and understandability of comments/documentation.

Modifiability

Flexibility of code architecture, ease of adding new features and adherence to SOLID principles.

Maintainability

Consistency with coding standards, readability and maintainability.

User Engagement

Visual appeal, intuitiveness of user interactions and responsiveness.

Brandon's Implementation

Functional Completeness/Correctness

The implementation adheres to the main functional requirements mentioned in the brief, which included the initialisation of the gameboard, chit cards, and dragon characters. It also included two key functionalities being the flipping of the chit cards, and the changing of turns between players, but omitted key features such as movement, and winning of the game. Overall, the functional completeness was fulfilled, exceeding the number of additional key features, but could be further improved.

Functional Appropriateness

Although this implementation adheres to the basic functionalities of the Fiery Dragon game, it heavily lacks any meaningful documentation or in-line comments to clarify and/or justify the creation of any and all attributes or methods created.

Appropriateness Recognisability

The code structure is quite straightforward, having functions and attributes being very easily understandable with self-explanatory labels/names. However, the lack of commenting and JavaDoc documentation makes it difficult to understand the logic behind many areas of the code. Additionally, the lack of modularity makes it a bit difficult to navigate through.

Modifiability

Other than having areas for other key features, the code lacks the flexibility to have new features added to it without a struggle. It lacks many aspects of SOLID, having no abstractions or interfaces present. It does feature separate classes for main entities, but very much benefit from applying many of the SOLID principles to further improve the modifiability.

Maintainability

The code generally follows the basic coding conventions, making it quite easy to follow along with and read. Maintaining the code may be quite difficult in some aspects, such as the main driver class, as they contain areas of code that could be refactored to create less 'clutter' in terms of initialising variables.

User Engagement

This implementation utilises a console based design, having the game board reprinted after every move. It clearly indicates to players key information, such as the state of the board, and chit card options, and also provides players with messages as to what has happened during the game, such as which player's turn it is and reasonings behind movements or lack thereof. However, having the chit card display be a bit more compact can make viewing the board and cards easier as the long list of cards can cause players to constantly scroll between the two.

Jeremy's Implementation

Functional Completeness/Correctness

This prototype implemented the initial game board setup by creating and randomising the order of chit cards along with preparing the board for four players. However, determining the player order and initialising the game board for games with less than four players were both overlooked. The flipping of chit card functionality was additionally completed with validation for user inputs that were tested in multiple scenarios. Movement of dragon tokens and winning the game were not included within this prototype. Overall, this implementation includes the required features for Sprint 2 but requires modification to the missed elements of initialising the game board.

Functional Appropriateness

As mentioned earlier, the prototype overlooks elements of the rules including determining the player order by age and creating a game board for games with less than four players. JavaDocs have been included in every class and method however the use of in-line comments could be added to explain coding choices to fellow developers.

Appropriateness Recognisability

Code structure follows standard Java conventions with all class methods containing appropriate names. JavaDocs are included throughout the code which aids other developers with understanding the purpose of each method and class. Further documentation could be included in the design rationale to explain the reasons for some methods and in-line comments would aid with explaining longer methods that exist in the source code.

Modifiability

The prototype follows some SOLID principles where extending the game would not be an issue such as the actor and animal abstract classes which could simply add more animal chitcards and tiles. However areas such as the board initialisation in this prototype have been made for only four players, which could create issues for refactoring in the future as this particular section does not have a very extendable design.

Maintainability

Current implementation follows all Java coding conventions and JavaDocs for every method and class. Separate methods have been created for frequently used code such as user input validation to reduce repetition of code. Overall, some methods are quite extensive and may need clarification to be understood by other developers which could impact maintainability.

User Engagement

The implementation is console based, inspired by the Sprint 1 low fidelity prototype where the current player is given the choice to choose a chit card from a list of hidden options. A running commentary with the current player, animal that needs to be chosen and state of the board are printed before and after each turn which makes it clear how the game is progressing. Instructions for making a move are also included every turn with validation and meaningful error messages along with the opportunity to redo a selection for incorrect inputs.

Max’s Implementation

Functional Completeness/Correctness

The key functionalities covered by the prototype include setting up the game board, movement of character around the board and winning of the game. However, movement is implemented through temporary functions to help demonstrate how the game is won and does not include matching the chit with the current animal. Additionally only a four player based game is implemented. The implemented features all satisfy its functional correctness however, further testing can be done to improve fault tolerance.

Functional Appropriateness

The prototype solution appropriately follows the intended functionality of the real life game ‘Fiery Dragon’. Documentation is included to explain classes, methods and attributes and how they relate with the physical game, however further in line comments could have been included to further clarify how the computation in more complex methods act according to how the game is supposed to operate.

Appropriateness Recognisability

The structure of code is organised in a reasonable manner, with classes sorted into packages and class/variable names describing what they represent. Documentation of all classes, attributes and methods are included. Comments are included to help navigate complex methods, however further comments can be added to assist in this regard, and approaches of complex methods could be summarised and included as part of documentation.

Modifiability

There is a degree of adherence to SOLID principles with a few abstractions used to reduce code repetition, complying with the Single Responsibility principle and Liskov’s Substitution principles. However, the ease of adding new features can be improved as the flexibility of the design is limited.

Maintainability

Java coding standards are followed and consistent. Complex algorithms are generally split into multiple smaller methods to ensure readability and to avoid god classes supporting the maintainability of the code, however, this can still be further improved with heavier classes such as Gameboard.

User Engagement

The prototype covers the essential functionalities outlined in the design, including board setup, player initialisation and randomised positioning of dragon cards. However, some features, like change of player turn and player movement are not yet fully implemented. Functional correctness has been satisfied, only requiring moderate additions and further testing to cover the full game.

Raymond's Implementation

Functional Completeness/Correctness

The prototype covers the essential functionalities outlined in the design, including board setup, player initialisation and randomised positioning of dragon cards. However, some features, like change of player turn and player movement are not yet fully implemented. Functional correctness has been satisfied, only requiring moderate additions and further testing to cover the full game.

Functional Appropriateness

The chosen solution direction seems to align with the basic requirements of the game. However, there's a lack of documentation or comments explaining the rationale behind certain design choices. For instance, the decision to use a specific data structure or algorithm for board setup is clarified through discussion but is not clearly justified in the code.

Appropriateness Recognisability

The code structure is reasonably organised but some sections could be further modularised or commented for clarity. Variable names and method names are generally descriptive which help in understanding. However, there's room for improvement in terms of documenting complex algorithms or design patterns used.

Modifiability

The code demonstrates some degree of extensibility with separate classes for different game entities and actions. However, there are instances of tight coupling and lack of abstraction, which may hinder easy extension. Refactoring the methods and classes to adhere more strictly to SOLID principles would enhance modifiability.

Maintainability

The source code follows basic coding standards and is relatively readable. However, there are instances of code smells such as long methods or classes which indicate the potential areas for refactoring. Additionally, reliance on conditional statements could be reduced to improve maintainability and avoid potential runtime errors. Code duplication is reduced to a strong degree, further supporting maintainability.

User Engagement

The user interface uses a text based design to depict the game in real time. Whilst it is simple and uses characters to represent animals, the structure of the game board and layout of the game is maintained. However, chit chards can be included in the middle of the board and dragon tokens can walk across volcano cards, rather than inside the board to further mimic a real life physical game of Fiery Dragon.

Review Conclusion

Selected Implementation

A unanimous decision was made to proceed with Max's implementation and will serve as the base of the Sprint 3 software prototype. However, elements of the remaining three implementations will be combined into the code base of Max's implementation to cover other functionalities of Fiery Dragon. More specifically, the functionality of flipping and matching chit cards to tile animals as well as transition of player turns will be taken from Brandon's implementation. User interface and parts of the initial setup of the game board will be taken from Raymond's implementation and player order sorting as well as data validation will be taken from Jeremy's implementation.

Justification

The primary rationale behind using Max's implementation as a base for the Sprint 3 prototype, is that upon the utilisation of code metrics to analyse each of the 4 implementations, all members believed that Max's implementation was the most extensible and allowed easy integration of functionality completed in other implementations.

Brandon's implementation was used for the functionality of flipping chit cards, which included checking whether the player should move as well as the transitioning of player turns, as the code was deemed to be most reliable and modifiable. Brandon's design of these concepts also allowed for the smoothest integration process into Max's code base, and therefore, these features from Brandon's implementation were chosen to be incorporated into the Sprint 3 prototype. Jeremy's implementation was the most robust with strong data validation through the program as well as the most efficient player ordering functionality, and hence, those two features from Jeremy's implementation were included into the Sprint 3 prototype. Lastly, Raymond's implementation was agreed to have the most engaging user interface system and parts of the initial set up of the game board could be taken and combined with Max's implementation to eliminate some of its rigid components, allowing for an even more flexible design of the game. Thus, the user interface system and the initial board setup components were chosen to be included into the Sprint 3 Prototype.

Class-Responsibility-Collaboration Cards

Game	
Description: The Game class is responsible for holding the information relating to the current instance of the game, and allowing the communication between the players and the game. This includes key attributes such as the players, order of the chit cards, and the gameboard.	
Responsibilities:	**Collaborators:**
● Store GameBoard	● GameBoard
● Generate ChitCards	● ChitCards & Engine
● Store Players	● Player
● Process Player’s Turn	● Player & ChitCard
● Determine Winner	● Player & GameBoard

GameBoard	
Description: This class represents the board that players will be using through their playthrough. It will contain all the volcano cards and player’s starting positions, as well as be responsible for a player’s movement around the board.	
Responsibilities:	**Collaborators:**
● Initialise game board	● Location
● Set starting positions	● VolcanoCard, DragonLocation, AnimalLocation, CaveLocation, Location
● Move player	● Player, DragonCharacter, DragonLocation, Location
● Validate moves	● DragonCharacter, Location
● Check chit cards match	● DragonCharacter, ChitCard, DragonLocation, CaveLocation, AnimalLocation

Engine	
Description: The Engine class's main responsibility is to be in charge of all methods involved with setting up a new instance of the game.	
Responsibilities:	Collaborators:
● Generate Game Board	● GameBoard
● Generate Volcano Cards	● Cave, TopVolcanoCard, BottomVolcanoCard, RightVolcanoCard, LeftVolcanoCard, Tile
● Generate Chit Cards	● ChitCard, Actor
● Create Players	● Player, DragonCharacter

Location	
Description: This class represents a particular location on the 2D game board and keeps track of what character will be displayed to the user interface. Also checks if there is a dragon, animal or cave present at the particular location.	
Responsibilities:	Collaborators:
● Checks if a dragon is present at location	● DragonLocation
● Checks if an animal is present at location	● AnimalLocation
● Checks if a cave is present at location	● CaveLocation

Menu	
Description: This class is responsible for displaying/printing messages to the players regarding things such as the current state of the game and list of chit cards. Additionally, it is responsible for interpreting user inputs when selecting a chit card.	
Responsibilities:	Collaborators:
● Display Opening screen	● None
● Display victory screen	● Player
● Display current state of the board throughout the game	● GameBoard
● Display Chit cards to the player	● ChitCard
● Take user inputs	● None

VolcanoCard	
Description: This class represents its physical counterpart in the Fiery Dragons game. It stores a list of three tiles, as well as the cave for that tile, if it is needed.	
Responsibilities:	Collaborators:
● Set up three individual tiles and their location	● Tile, Location
● Attach to cave (if needed)	● Cave, CaveLocation
● Set up ‘walkable’ locations along the board	● Location, GameBoard

Alternative responsibility distribution

1. Originally the responsibility of setting up the locations of all the tiles was done by the GameBoard class. However, this idea was later rejected as the location of the tile class will vary based on which volcano card it is a part of. Thus, it was decided that the VolcanoCard class should be responsible for setting up the tiles and initialising them with their respective locations.

2. Another alternative that was considered was having the game board class be responsible for deciding and holding the locations that are 'walkable' by dragon tokens. However, this idea was rejected due to the GameBoard class handling too many responsibilities, violating the Single Responsibility Principle and becoming a 'god' class. This responsibility was then distributed to the VolcanoCard class, which determines the 'walkable' locations along the board, and the DragonCharacter class which holds the sequence of locations the particular dragon token will travel in the game.

Class Diagram

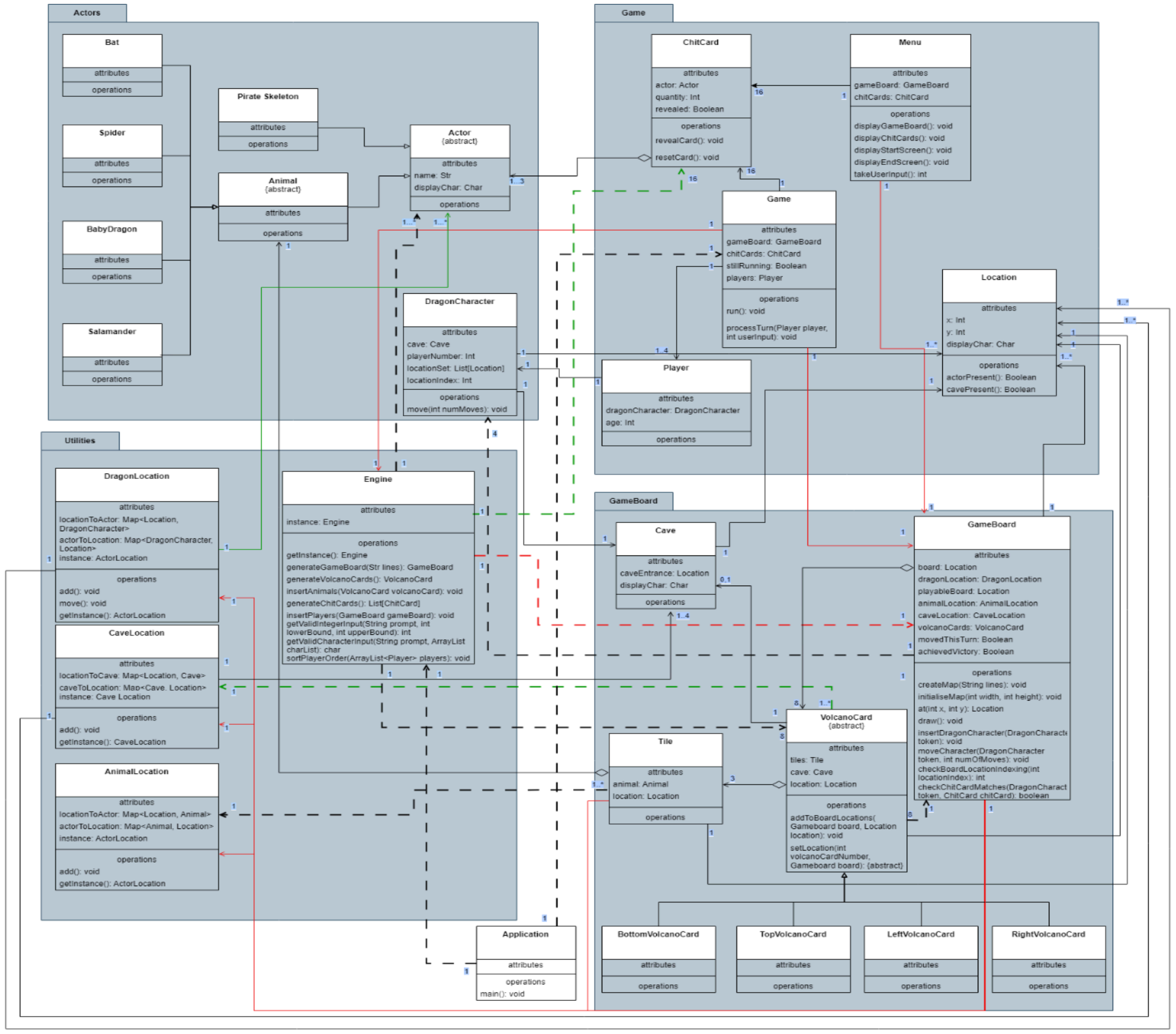


Figure 1. Completed Class Diagram of Fiery Dragons Implementation

[Click here to see clearer diagram]

Executable

Description

The executable was compiled as a .jar file.

The original compilation environment was on a MacOS running java version 18.0.2

Instructions

Step 1: Open a terminal and navigate to the directory containing the .jar file.

Step 2: Enter 'java -jar Game.jar' to run the executable.

Video Demonstration

Description

The video was taken on Zoom. It can be found in the Moodle submission, as well as the GitLab Sprint 3 folder.

Video Length: 7 minutes 50 seconds

File Name: Chitbusters-Sprint-3-Demonstration.mp4

File Format: MP4