FIT3077 SPRINT 3

Mun Hong Ooi, Owen Zhang, Jacky Zhao, Kevin Feng

Assessment of Sprint 2 Prototypes

Additional Assessment Criteria and Metrics

Completeness of the Solution Direction

1. Criteria: Evaluate whether the solution encompasses all required functionalities as outlined in the project specifications.
2. Metrics:
    - Checklist of key functionalities required: Measure the percentage of required functionalities that are addressed in the design.
    - Feature traceability matrix: Cross-reference requirements with implemented functionalities to identify any gaps.

Rationale Behind the Chosen Solution Direction

3. Criteria: Assess the justification provided for choosing the specific solution, including how it meets the needs of the project.
4. Metrics:
    - Justification quality score: Rate the rationale on a scale from 1 (poor) to 5 (excellent) based on relevance, clarity, and depth of explanation.
    - Alignment with project goals: Evaluate how well the solution's rationale aligns with the overarching project objectives.

Understandability of the Solution Direction

5. Criteria: Determine how easily team members and stakeholders can understand the proposed solution.
6. Metrics:
    - Comprehension test results: Conduct a survey or quiz among team members or stakeholders to measure how well they understand the solution.
    - Clarity rating: Use a Likert scale from 1 (very unclear) to 5 (very clear) based on feedback from the team.

Extensibility of the Solution Direction

7. Criteria: Assess the ease with which the solution can be extended or modified in future development cycles.
8. Metrics:
    - Extensibility score: Rate the design’s extensibility on a scale from 1 (not extensible) to 5 (highly extensible), considering factors like modularity and the use of design patterns.
    - Future feature integration feasibility: Analyse how easily new features as planned for future sprints can be integrated without significant rework.

Quality of the Written Source Code

9. Criteria: Evaluate the quality of the code based on readability, adherence to coding standards, and the robustness of implementations.
10. Metrics:

- Code review scores: Use a standardised code review checklist to evaluate each piece of code and score it based on predefined standards.
- Static analysis reports: Utilise tools to analyse the code for complexity, adherence to coding standards, and presence of anti-patterns.

Aesthetics of the User Interface

11. Criteria: Judge the visual appeal and user-friendliness of the user interface.
12. Metrics:
    - User satisfaction survey: Gather feedback from users on the visual appeal and ease of use of the interface, rating on a scale from 1 (poor) to 5 (excellent).
    - Aesthetic quality rating: Conduct expert reviews to assess UI aesthetics based on modern design principles and best practices.

Owen's Prototype Assessment & Review:

Completeness of the Solution Direction

- Criteria: All key functionalities in the design.
- Observation: The game includes a main menu, card and chip elements, player setup, and basic interaction logic (click detection and win condition display). The basic game loop and display mechanisms are implemented.
- Metric Score: Appears to cover most basic requirements for a functional game but lacks details on various game modes or settings if any were initially intended.

Rationale Behind the Chosen Solution Direction

- Criteria: Justification of the chosen design and architecture.
  Observation: The object-oriented approach properly segregates responsibilities: Card, Chip, and Player handle display and state, Game orchestrates gameplay, and main initialises the game environment. This design supports maintainability and scalability.
  Metric Score: Well-rationalised in terms of modular architecture conducive to scaling and updates.

Understandability of the Solution Direction

- Criteria: Clarity and comprehensibility of the solution to developers and stakeholders.
- Observation: The use of descriptive class and method names and structured code layout enhances understandability. Comments or documentation are minimal, which can be improved for better clarity.

- Metric Score: Generally clear with scope for adding more inline documentation and comments.
-

Extensibility of the Solution Direction
- Criteria: Ability to extend the solution.
- Observation: The game architecture allows for easy addition of new cards, chips, or players, suggesting good use of object-oriented principles.
- Metric Score: High extensibility due to modular design, although deeper dependencies and interactions could be better abstracted.

Quality of the Written Source Code
- Criteria: Code quality, including readability, standards adherence, and robustness.
- Observation: The code is readable and structured. However, it employs hard-coded values and could benefit from exception handling and configuration options to enhance robustness and adaptability.
- Metric Score: Good with room for improvement in error handling and configuration.

Aesthetics of the User Interface
- Criteria: Judge the visual appeal and user-friendliness of the user Criteria: Visual appeal and user-friendliness of the UI.
- Observation: The UI design as shown in the screenshots is functional with basic interaction elements. The aesthetics are somewhat simplistic and might not engage users looking for a visually rich experience.
- Metric Score: Functional but basic; could benefit significantly from design enhancements.

Usage of Git Repository
- Criteria: The dedicated branch contains multiple commits that span over multiple days. The commits comments are meaningful and describe what work is being committed. All source code is available in the branch. Multiple versions of source files are identifiable - the commit history shows how work has progressed.
- Observation: multiple commits over one or two days, all source code is available in the branch
- Metrics: very good

Object Oriented Design
- Criteria: A clear and neat Class Diagram for the Fiery Dragons game. Usage of correct notations.The provided classes contain all the relevant attributes, methods, relationships, and cardinalities, and cover all required functionality. Good object-oriented design principles are followed.
- Observation: correct use of notations and cardinalities, all classes implemented are shown on the class diagram
- Metrics: very good

Key Functionalities

- Criteria: All of the key game functionalities are covered and it is clearly explained how they are covered.
- Observation: the key functionality is correctly and completely implemented and working perfectly, however some of the aspects of the key functionality is hard coded. The key functionality is explained.
- Metric: very good

Design Rationale

- Criteria: Design rationale covering most required areas (classes, relationships, inheritance, cardinality, design pattern) that present reasonable rationales for design choices made.
- Observation: some of the design rationale including the cardinalities and relationship are covered, however the use of design patterns are not covered.
- Metrics: acceptable

Tech work in progress

- Criteria: Given the design presented, source code for (i) the game set-up and (ii) the chosen key game functionality is provided.
- Metrics: acceptable

Executable deliverable

- Criteria: Executable provided with information what platform it can be run on, including instructions how to build the executable from the source code. Provided interface clearly shows all elements of the Fiery Dragon game. Dragon board is set up correctly (i.e. no missing caves or caves not in the middle of volcano cards). Tokens are in the correct starting positions and can be clearly distinguished from each other. Dragon ("chit") cards are arranged "inside" the ring of volcano cards, are clearly visible as "turned over", and all look the same (they cannot be distinguished). Second key game feature can be executed and fully demonstrates the expected functionality.
- Observation: executable file is provided, run instructions and targeted platform is stated. The executable file runs correctly and all implemented functionalities are working. The game include the main feature 'winning the game' and additional feature of 'flipping chit card'
- Metrics: excellent

Video

- Criteria: A video that demonstrates the implementation of the two key game functionalities is provided.Description of the file format given. The presenter provides an overview of the contents of the video at the start.
- Observation: provided introduction. Demonstrated the key functionalities. Shown the executable file, but did not explain how the executable file is created.
- Metrics: good

Owen's Overall Assessment:
The solution provides a functional and logically structured game with basic features implemented. It has a strong foundation in terms of architecture that facilitates future

enhancements and scalability. However, the project could be improved in areas of code documentation, and robustness features like error handling. Further development can focus on enhancing the visual elements, adding comprehensive inline documentation, and incorporating flexible game settings to improve both user experience and maintainability.

Kevin’s Prototype Assessment & Review:

Completeness of the Solution Direction

- Criteria: All functionalities in the main design
  This includes the initial game board initialisation and card flipping function.
- Metrics:
  - Checklist of key functionalities required: Measure the percentage of required functionalities that are addressed in the design.
  - Feature traceability matrix: Cross-reference requirements with implemented functionalities to identify any gaps.

  This is mostly done. A main menu page is displayed and you can enter the game with its button "PLAY" and the card is able to be flipped with mouse click. However, the number of dragon cards and the number of tiles displayed are incorrect with the game book but there is a minor gap in functionality. Will say 95% done.

Rationale Behind the Chosen Solution Direction

- Criteria: Assess the justification provided for choosing the specific solution, including how it meets the needs of the project.
- Metrics:
  - Justification quality score: Rate the rationale on a scale from 1 (poor) to 5 (excellent) based on relevance, clarity, and depth of explanation.
  - Alignment with project goals: Evaluate how well the solution's rationale aligns with the overarching project objectives.

  The explanation between key functionalities and the reason for defining classes are clear and logical. This also occurs at Cardinality Decisions but Decision Around Inheritance and key relationship are not clear. In Design Pattern Application, it seems to detail what the future plan for this sprint is but not having enough explanation why certain design patterns are not suitable. Some parts feel like they are not relevant in this sprint(will talk about in detail in the next Understandability section). I will give 3 on the quality score and 90% on alignment.

Understandability of the Solution Direction

- Criteria: Determine how easily team members and stakeholders can understand the proposed solution.
- Metrics:
  - Comprehension test results: Conduct a survey or quiz among team members or stakeholders to measure how well they understand the solution.
  - Clarity rating: Use a Likert scale from 1 (very unclear) to 5 (very clear) based on feedback from the team.
- I think the Rationale page contains some sections that are not necessary. For example, the key functionalities of the entire game is outlined and not mention which functionalities that he chooses to implement. This will make the project hard to understand if you didn't try the game he implements. Also, there is a section of Key Relationship between Game_board and Achievement which the latter is not even in the game. Also, having an image of a class diagram in the design rationale page may be helpful. I will give 3 for clarity.

Extensibility of the Solution Direction

- Criteria: Assess the ease with which the solution can be extended or modified in future development cycles.
- Metrics:
  - Extensibility score: Rate the design's extensibility on a scale from 1 (not extensible) to 5 (highly extensible), considering factors like modularity and the use of design patterns.
  - Future feature integration feasibility: Analyse how easily new features as planned for future sprints can be integrated without significant rework.

  The main functionalities that will be altered are flipping the game chip in game_chip.py and game board in game_tile.py and game_playscreen.py. All of those are constructed in class format so it can be easily adopted and changed by inheritance. High flexibility.

Quality of the Written Source Code

- Criteria: Evaluate the quality of the code based on readability, adherence to coding standards, and the robustness of implementations.
- Metrics:
  - Code review scores: Use a standardised code review checklist to evaluate each piece of code and score it based on predefined standards.
  - Static analysis reports: Utilise tools to analyse the code for complexity, adherence to coding standards, and presence of anti-patterns.

  Name of each file is clear outlined its functionalities and connotation in code block to give meaningful explanation. The game functionalities is too small to occur bug now but may occur later sprint and therefore, exception handling is needed to handle error and crash.

Aesthetics of the User Interface

- Criteria: Judge the visual appeal and user-friendliness of the user interface.
- Metrics:
  - User satisfaction survey: Gather feedback from users on the visual appeal and ease of use of the interface, rating on a scale from 1 (poor) to 5 (excellent).

    - Aesthetic quality rating: Conduct expert reviews to assess UI aesthetics based on modern design principles and best practices.

Minimalistic game starting and game board interface. Mouse clicks are quickly responded with the dragon card flipping. Overall 5 and Aesthetic 4.

Usage of Git Repository

- Criteria: The dedicated branch contains multiple commits that span over multiple days. The commits comments are meaningful and describe what work is being committed. All source code is available in the branch. Multiple versions of source files are identifiable - the commit history shows how work has progressed.

  Commits are shown over multiple days. Meaningful comment to see how the progress of the development has occurred.

Object Oriented Design

- Criteria: A clear and neat Class Diagram for the Fiery Dragons game. Usage of correct notations. The provided classes contain all the relevant attributes, methods, relationships, and cardinalities, and cover all required functionality. Good object-oriented design principles are followed.
- Observation: No incorrect symbol applied in the diagram. Every detail in the design rationale corresponds to the visualised diagram. Good

Key Functionalities

- Criteria: All of the key game functionalities are covered and it is clearly explained how they are covered.
- All key functionality is covered. Good

Design Rationale

- Criteria: Design rationale covering most required areas (classes, relationships, inheritance, cardinality, design pattern) that present reasonable rationales for design choices made.

Explained in **Rationale Behind the Chosen Solution Direction**

Tech work in progress

- Criteria: Given the design presented, source code for (i) the game set-up and (ii) the chosen key game functionality is provided.
- Outline is clear. Program runs smoothly without bugs. Contains text files on what functionalities are implemented.

Executable deliverable

- Criteria: Executable provided with information what platform it can be run on, including instructions how to build the executable from the source code. Provided interface clearly shows all elements of the Fiery Dragon game. Dragon board is set up correctly (i.e. no missing caves or caves not in the middle of volcano cards). Tokens are in the correct starting positions and can be clearly distinguished from each other. Dragon (“chit”) cards are arranged “inside” the ring of volcano cards, are clearly visible as “turned over”, and all look the same (they cannot be distinguished). Second key game feature can be executed and fully demonstrates the expected functionality.

- Explained in **Completeness of the Solution Direction**. Number of cards and tiles are wrong with the dragon cave not presented.

Video

- Criteria: A video that demonstrates the implementation of the two key game functionalities is provided. Description of the file format given. The presenter provides an overview of the contents of the video at the start.
- No sound in the video but did give a run down on the overall functionalities with text file with his sprint to as an overview and functionalities implemented is shown throughout the video.

Kevin's Overall Assessment:
This sprint provided an efficient solution to its goals with a few gaps that don't affect most functionalities. The documentation of the design rationale could be expressed in a more understandable way and involve unnecessary sections that could be valuable to the future sprint. Code has great flexibility and quality.

Daniel's Prototype Assessment & Review:

Completeness of the Solution Direction

- Criteria: All key functionalities in the design.
- Observation: The game includes a main menu, the game board, chip and card elements, buttons in the game and music. The game displays and runs well. All game functions are implemented correctly.
- Metric Score: Covers all requirements of a functional game, even includes other quality of life things such as music.
-

Rationale Behind the Chosen Solution Direction

- Criteria: Justification of the chosen design and architecture
- Metrics:
  - Justification quality score: 5. The justification was done in details and gave clear reasons. All justifications were relevant to the game and very in depth.
  - Alignment with project goals: The solution mentions project objectives in detail and connects each solution direction with the overarching project objective. The chosen justifications all aligned well with each of the implemented game functionalities.

Understandability of the Solution Direction

- Criteria: Determine how easily team members and stakeholders can understand the proposed solution.
- Metrics:

    - Comprehension test results: All team members understood the solution well, the solution direction is written clearly and makes very good points.
    - Clarity rating: 5. The solution direction is very clear and all team members understood the direction well.

Extensibility of the Solution Direction

- Criteria: Assess the ease with which the solution can be extended or modified in future development cycles.
- Metrics:
    - Extensibility score: 5. The design implemented includes multiple classes that can be extended or inherited when needed. The design uses very good design patterns.
    - Future feature integration feasibility: In future sprints there are plans to implement an in game chat feature, this can be very easily done with the existing game board class and other classes.

Quality of the Written Source Code

- Criteria: Evaluate the quality of the code based on readability, adherence to coding standards, and the robustness of implementations.
- Metrics:
    - Code review score: 5. The use of object-oriented programming makes the code very readable and the code is written very clearly with each function indicating what it's supposed to do.
    - Static analysis reports: The code is complex however it's very readable, multiple different files with separate classes makes the code very easy to understand. The code follows very good coding standards such as the use of comments.

Aesthetics of the User Interface

- Criteria: Judge the visual appeal and user-friendliness of the user interface.
- Metrics:
    - User satisfaction survey: 4. The interface of the game board is very well done, and the game includes UI elements that make the game very easy to play. The only main drawback is that the game lacks backgrounds and the main menu is a bit empty. However these flaws are not required for this sprint and are simply suggestions to make the game better.
    - Aesthetic quality rating: 4. All required UI elements are in the game and function well, but compared to most modern games it looks a bit empty. However this can be improved in the future so it's not that important for sprint 2.

Justification of Chosen Prototype

Daniel's prototype stands out as the most complete version of the Fiery Dragon game among all team members. It adheres to very good coding conventions, which simplifies the process for other members to modify and add new features that are not yet implemented. Additionally, the prototype enhances the user experience by incorporating quality-of-life features, such as music.

Summary Review Of All Prototypes

In summary, all above group members’ sprints have successfully provided a solution to its intended functionalities. Kevin has sufficient structured code for initialised game boards. The design rationale is simple and clear within most paragraphs but the way of organising paragraphs is confusing and some need editing. The class diagram clearly corresponds to its rationale. Owen also has sufficient structured code to all its desired functionalities in sprint. The design rationale is clear on what the relationship between each entity and the design decision is clearly outlined and resolution with sufficient reasoning. Class diagrams are overall good. Daniel has the most well received game interface and its background music. The design rationale is explained greatly and in depth. The diagram is correctly outlined.

How the Creation of The Tech-based Prototype is To Be Performed

1. What are the ideas/elements of each of the tech-based prototypes from Sprint 2 we are going to use for Sprint 3?

    - Implementation for a way of navigation between different screens.
    - Background music.
    - Optimised universal class for fundamental interactables such as buttons.

2. What new ideas/elements do we need to bring in in order to improve the prototype?

    - Boolean identification for certain game events such as chits being flipped, or for conditions such as when players move to different tiles. This could be implemented with a universal method that detects positions on the game board.

Justification

1. Utilising Elements from Sprint 2:

a. Navigation between Different Screens:

- **Rationale:** The implementation of a seamless way to navigate between different screens is crucial for enhancing user experience and ensuring that the flow of the game is intuitive and user-friendly. This functionality has been tested and optimised during Sprint 2, proving its effectiveness and reliability.

- **Impact:** It allows players to easily transition between game states without confusion, thus maintaining engagement and ensuring a smooth gameplay experience.

b. Background Music:

- **Rationale:** Background music is essential for creating an immersive gaming environment. It sets the tone and enhances the thematic elements of the game. The feedback from Sprint 2 highlighted that the inclusion of background music positively affects player immersion and satisfaction.

- **Impact:** Consistently using background music helps in maintaining an engaging atmosphere, potentially increasing the time players spend in the game.

c. Optimised Universal Class for Fundamental Interactables (such as buttons):

- **Rationale:** Developing a universal class for interactables like buttons has streamlined the development process and reduced redundancy. This optimization from Sprint 2 has improved the maintainability and scalability of the codebase.

- **Impact:** It ensures that all interactable elements behave consistently across different parts of the game, thereby enhancing the user experience and reducing potential bugs.

2. New Ideas/Elements for Sprint 3:

a. Boolean Identification for Game Events and Conditions:

- **Rationale**: Introducing a universal method for Boolean identification for specific game events (like chits being flipped) and conditions (such as players moving to different tiles) is necessary to handle game state transitions efficiently. This method will provide a clear and manageable way to track and respond to changes in the game environment, which is essential for both gameplay logic and user interface updates.

- **Impact:** Implementing this feature will improve the responsiveness of the game to player actions, making the game feel more interactive and engaging. It will also simplify the development process by providing a standardised approach to event handling and condition checking.

Class-Responsibility-Collaboration (CRC)

1.

Player	
Choose Dragon	Dragon
Initialise Game Board	Game Board
Choose Dragon Card to flip	Dragon Card

2.

Game Board	
Initialise Tiles	Tile
Initialise Cave	Cave
Initialise Dragon Card	Dragon Card
Place Dragon in Cave	Dragon, Cave
Judge winning	Cave

3.

Cave	
Know its dragon	Dragon
Know its characteristic(Animal)	

4.

Tile	
Know its characteristic(Animal)	
Know if it already has a dragon	Dragon

5.

Dragon Card	
Know its characteristic (Animal/ Dragon Pirate)	
Know its number	
Flipping	Player

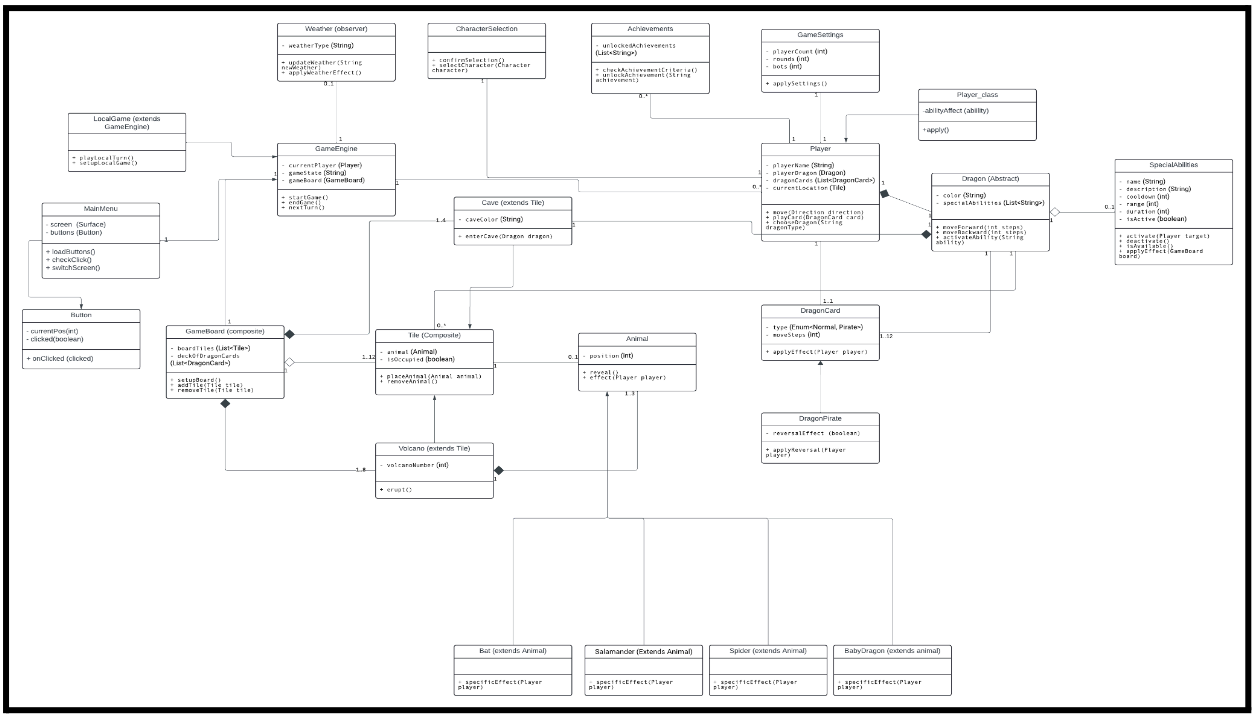
6.

Dragon	
Move	Dragon Card, Tile, Cave

Description for the purpose of the 6 main classes

1. Player is an interactive class for users that starts the game by initialising the game board and choosing the dragon. It has the responsibility of choosing which dragon card to flip.
2. Game Board manages all most spatial elements of the game by initialising all the classes participating in the game board like tile, cave and dragon card. It is also responsible for the reporting of which player wins.
3. Cave is like a data holder class that contains its dragon which is useful for judging winning condition and the animal it holds because players of the dragon have to flip a corresponding animal dragon card with the cave to start moving.
4. Tile is like a data holder class that holds an animal that lets a dragon move to another if the player flips the right dragon card with the same animal.
5. Dragon Card is the central mechanic of the game that lets the dragons know if they can move, at what direction and across how many tiles.
6. Dragon representation of the player in the game board and communicate with other classes to know what position it should be on the game board.

UML Class Diagram



Link to UML class Diagram

Executable Deliverable

After downloading the zip file and unzipping it, there should be a ‘main.exe’ file that can be executed without any extra installation. The target platform for this game is windows.