FIT3077 - Architecture and Design

Sprint Three

MA_Tuesday08am_Team123

Members:

Maliha Tariq

Loo Li Shen

Khor Jia Wynn

Dizhen Liang

MONASH University

1. Assessment Evaluation Criteria

We define the following criteria adapted from the ISO/IEC 20510 quality model to assess each team member's design and implementation.

Functional Suitability
This characteristic represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions and comprises:

- Functional completeness - The degree to which the set of functions covers all the key functionalities covered in the design.
- Functional correctness - The degree to which a product or system provides accurate results when used by intended users.
- Functional appropriateness - The degree to which the functions facilitate the accomplishment of specified tasks and objectives

Interaction Capability
The degree to which a product or system can be interacted with by specified users to exchange information is the user interface to complete specific tasks in a variety of contexts of use. This characteristic is composed of the following sub-characteristics:

- Appropriateness recognizability - Understandability of the solution direction
- User Engagement - Aesthetics of the user interface

Maintainability
- Coding Standards - Quality of the written source code (e.g., coding standards, reliance on case analysis and down-casts)
- Documentation Quality - Appropriate use of documentation for complex software components, clarity of documentation
- Error Handling - Logging is provided for debugging errors, or errors are prevented altogether.
- Code Readability - Adhering to naming conventions, easy for the reader to understand without having to consult the code author
- Modifiability - The degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality (in anticipation of extensions to the game for Sprint 4).

Our measurement scale:

1 - Poor	Several Major Omissions
2 - Fair	Few Major Omissions
3 - Satisfactory	Many Minor Omissions
4 - Good	Minor Omissions Only
5 - Excellent	No Omissions detected (which does not mean there are none, but you are not aware of any)

MONASH University

2. Review of Sprint 2 Tech-based Software Prototypes

Team Member 1: Maliha Tariq
Chosen Key Functionality: (ii) Flipping of the chit cards

Assessment Criteria	Marking	Justification
Functional Suitability		
Functional Completeness	5	Flipping of the chit cards along with the game board set up was done well and able to run smoothly without error
Functional Correctness	4	Only minor absentees where the board does not have a circular shaped line out but other than that everything that needs to be there is there
Functional Appropriateness	5	The class seems fine as there are multiple classes for each function instead of hoarding everything into one. For example the classes, “Card”, “CardFactory” and “ChitCard”
Interaction Capability		
Appropriateness Recognizability	5	The starting menu of the page is straightforward and direct allowing users to know what they want and where to look for (ie., Single Player, Play With Friends)
User Engagement	3	The design has all the right images as well as the necessary shape based off the requirements which allows the user to visualise the actual circular board shape, however as mentioned previously there wasn’t a circle outline and there are gaps between each segment parts of the cards which doesn’t provide the full user experience required
Maintainability		
Coding Standards	4	The classes and methods that are created are being used as well as implemented properly according to their needs There are also multiple classes for different uses which does not clump up everything into a single class, (eg: GameFrame, ControlPanel and GamePanel)
Documentation Quality	4	All classes are documented however some of the methods in the class aren’t documented of their functionalities, but the titles are straightforward and explains itself, However, it lacks comments explaining the purpose and usage of the enum.
Error Handling	2	Shows an GameTests class which means that error is tried and tested for, however the class doesn’t work when tried to run it
Code Readability	5	Code seems clear and concise as with good naming

		conventions for each methods and variables used
Modifiability	3	The code does show modifiability by organising functionalities into separate classes and provides basic modularity by separating UI components. However, proper error handling mechanisms (from what I can run) are missing, which could lead to unexpected behaviour and make the code less maintainable.

Total Score: 40/50
Summary:
The assessment indicates that they excel in certain aspects, earning high scores for functional suitability and appropriateness recognizability, thanks to its smooth execution and straightforward user interface. Functional accuracy is almost perfect, with very small exceptions, such as the lack of a circular board outline. The code demonstrates good maintainability with clear coding standards and documentation, although there are some gaps in method documentation and error-handling implementation. While the code is modifiable due to the organisation of functionalities into different classes, the lack of suitable error-handling techniques may impede future maintenance efforts, reducing overall maintainability.

Overall, the code has strong functional performance and user interface capabilities, while there is potential for improvement in error management and modifiability.

MONASH University

Team Member 2: Loo Li Shen
Chosen Key Functionality: (iv) Change of turn to the next player

Assessment Criteria	Marking	Justification
Functional Suitability		
Functional Completeness	3	The gameboard shows the volcano ring, with the chit cards placed inside and the tokens, however there are no images, and the view is generic. Change of turn is done by a button and it also rotates the view of the gameboard for the next player
Functional Correctness	5	The turn-changing button effectively switches player turns without errors or glitches.
Functional Appropriateness	3	**Factory Method** pattern is mentioned but not directly implemented in the code.
Interaction Capability		
Appropriateness Recognizability	4	The code exhibits a clear structure for managing capabilities and characters, facilitating easy understanding of their purpose and relationships.
User Engagement	2	Due to the absence of images on the board, the game board lacks visual appeal, impacting user engagement.
Maintainability		
Coding Standards	4	The code adheres to basic coding standards. However, some improvements could be made, such as organising imports and ensuring consistent formatting throughout the codebase.
Documentation Quality	4	The provided class summaries are informative, yet some classes lack documentation, resulting in inconsistency.
Error Handling	2	Error handling in the provided code is minimal. While some exceptions are caught and printed to the console, there is limited error handling beyond that
Code Readability	3	Variable names are descriptive, and the code structure follows standard Java conventions. However, certain methods could benefit from better organisation and decomposition.
Modifiability	3	The code demonstrates modifiability to some extent by using inheritance and composition to structure the relationships between classes. However, there could be further improvements in terms of making the code more flexible and extensible to accommodate future changes or additions to the game's functionality.

MONASH University

Total Score: 33/50

Summary:

Based on the assessment, the current implementation demonstrates functional completeness to a reasonable extent, with the gameboard effectively displaying the volcano ring and chit cards, albeit lacking images, resulting in a somewhat generic visual experience. The turn-changing mechanism, facilitated by a button, functions correctly and enhances gameplay dynamics. However, the absence of images detracts from user engagement, impacting the overall gaming experience.

While the code exhibits a clear structure for managing capabilities and characters, there are opportunities to improve error handling, enhance code readability through better organisation and decomposition of methods, and ensure consistent documentation across all classes. Additionally, implementing the Factory Method pattern, as mentioned in the design rationale, would further enhance functional appropriateness.

MONASH University

Team Member 3: Khor Jia Wynn
Chosen Key Functionality: (v) Winning the game

Assessment Criteria	Marking	Justification
Functional Suitability		
Functional Completeness	5	Functionalities are completed according to the requirements
Functional Correctness	4	Caves are missing animals. Lack of outline for the segments.
Functional Appropriateness	5	All the functions handle everything correctly
Interaction Capability		
Appropriateness Recognizability	4	The provided class diagram and rationales provide a good indication of the solution direction, with no inconsistencies with the implementation. Tokens are not initialised on cave may confuse player where to start
User Engagement	5	Clear interaction logic with player turn display on top-left corner. Excellent effect of using animation to display dragon token movement and flipping of chit cards.
Maintainability		
Coding Standards	5	Extensive and proper use of design patterns and obey design principles, **ChitCardFactory (Factory Pattern)**, **LSP** for **EntityFactory** in **SpawnFactory.** **`PlayerTurnManager`** **-> Singleton Pattern**
Documentation Quality	4	Minor documentation omissions. Provided documentation can be improved explaining important methods in more detail.
Error Handling	5	No crashes encountered when extensive play-testing is done.
Code Readability	3	Code is readable but some parts lack clarity, and require further explanation from the author. Some methods are missing documentation.
Modifiability	4	Use of design patterns is suitably justified, and the responsibilities are wrapped to the corresponding classes. **(SRP**, **OCP** in **ChitCardFactory designed to extend with new entity types)** in The design somewhat supports the extension of accommodation of different board configurations, as this logic can be controlled via the arguments passed to the constructor of the VolcanoRingFactory class, with some minor refactoring. No explicitly use of **Interface Segregation Principle**

		(ISP)

Total Score: 44/50

Summary:
All functionalities are almost perfectly implemented, and beyond the scope of chosen key functionality. The minor omissions are: 1) Cave should have animals. 2) While the player's dragon token has not exited the cave, animal type checks are done between the Cave's animal and the chit card's animal.

The rationale for the chosen solution direction is consistent with the implementation, and ample justifications are provided. The solution direction is mostly understandable with reference to the class diagram, sequence diagrams and rationales. However, the source code documentation should be more comprehensive to improve readability. The code is written with a logical flow, and usage of case analysis is appropriate for the specific solution direction. The aesthetics of the user interface are satisfactory.

MONASH University

Team Member 4: Dizhen Liang
Chosen Key Functionality: (iii) movement of dragon tokens based on their current position as well as the last flipped dragon

Assessment Criteria	Marking	Justification
Functional Suitability		
Functional Completeness	5	**Setup of game board**: Board shows the volcano ring, the caves, the tokens, and the chit cards. **Movement of dragon token**: Dragon token can move when chit cards are flipped.
Functional Correctness	3	**Setup of game board:** Minor placement issue of animal on Tile **Movement of dragon token**: 1) Logic error causing incorrect movement behaviour: Should be checking the animal type of the Tile it is on with the animal type of the last flipped chit card. The implementation incorrectly uses the animal type of the cave instead. 3) The player should be able to see all the animals on all the Tiles of the volcano ring at all times. However, whenever a dragon token moves to a new Tile, it covers the animal on it.
Functional Appropriateness	3	**Setup of game board**: Decomposing the initialisation/setup into separate functions for each of the components practises modularisation. However, assigning it the responsibilities of setting up the game board, handling user interactions, and managing state violates the Single Responsibility Principle. **Movement of dragon token**: 1) The automated approach to moving the dragon tokens discussed in the design is appropriate. However, players should not be allowed to drag the dragon token, as this would introduce confusion to the aforementioned mechanism. 2) Delegating the responsibility of 'checking if a valid move is made' to the token is acceptable.
Interaction Capability		
Appropriateness Recognizability	3	The provided class diagram and rationales provide a fair indication of the solution direction. However, some confusion arises because of some inconsistencies between the design and implementation. Some classes highlighted in the design are not used (Volcano), and some modularisation techniques (checkTokenMovement, calculateTokenPosition) discussed do not translate to the actual implementation. Consider splitting the sequence diagram into smaller, less complex diagrams to illustrate the interactions more clearly.
User Engagement	3	The image resources used are aesthetically pleasing. The

		response time is satisfactory, and the flip and movement animations are fair. The token shares the same image as the cave, and the placement of the token directly on the cave makes it difficult to distinguish between them. Consider adding distinguishing elements, or adjusting the placement of the token. As mentioned, the dragon token obscures the animal on the Tile it occupies, preventing the player from seeing it. This makes it difficult for the player to make the next move, since it forces the player to remember the animal on the Tile as well.
Maintainability		
Coding Standards	3	Misalignment between design and implementation: Volcano class was not used. Some parts of the code can be modularised. Some classes handle too many responsibilities (GOD classes): RoundGameBoard, ChitCard
Documentation Quality	3	Some documentation is provided, but lacks clarity. Missing documentation for important methods and/or variables. For example, how 'setComponentZOrder()' and repainting are used to achieve the effect of moving the token.
User handling	5	No crashes encountered when extensive play-testing is done. Logging is provided to aid with development, e.g. errors that may arise during image resource retrieval.
Code Readability	3	Code is generally readable but constrained to the documentation quality. Some parts of the code require detailed explanations from the author.
Modifiability	3	Excessive use of magic numbers affects future extensions. Some refactoring effort required to support different board configurations. Good use of inheritance: Defining the RoundComponent parent class for the UI components of Cave and DragonToken. It can be used for implementing the Luck Card in future extensions.

Total Score: 34/50

Summary:

In terms of functional requirements, correctness requires the most improvement. This should be coupled with extensive testing of the various scenarios for moving the dragon token.
Most of the rationales for the chosen solution direction are appropriate for the requirements in Sprint 2. The use of inheritance for abstracting the common UI features is appropriate and can be reused in the consolidated design.

Given the design documents and rationales, the solution direction was understandable, with only a few inconsistencies between the design and the implementation. The written rationales are easy to understand, but simpler sequence diagrams are needed to visualise interactions within the system.

To support future work, there is a need to reallocate some of the responsibilities of GOD classes to existing classes or to new dedicated classes. There is some refactoring required to support various board configurations as the extension requirement in Sprint 4.

The source code requires more comprehensive documentation to clarify semantics. Case analysis is used where necessary, but alternative solutions can be explored.

The aesthetics of the UI is satisfactory, though the animations have potential for improvement.

3. Outcome of the Assessment

Based on the above reviews, we decided to use team member 3's tech-based software prototype as a baseline for Sprint 3. The justifications are as follows:

1) It is the most functionally complete, correct and appropriate to the basic Fiery Dragons game
    a) Completed implementation for setting up game board, flipping of chit cards, change of turn, and winning the game.
    b) Extensive play-testing was done to cover all possible scenarios, and correct behaviour was displayed (with the exception of the issue mentioned above)
2) The aesthetics are acceptable and consistent. Card flip animation and dragon token movements are visually appealing.
3) The solution direction for board setup aligns with other team members and is justified
    a) Decomposition of the game board into separate components avoids a single GOD class
    b) Defining classes with different roles: Manager, Information/State Holder and Factory/Initialiser enforces separation of concerns

We decided to integrate some unique elements of the other tech-based prototypes into Sprint 3:

Sprint 2 Prototype	**Element to be integrated**
Team member 1	● Main menu of the Fiery Dragons game application. ● Defining a generic Animal class.
Team member 4	Implementation of animals on caves, as well as the logic for checking animal types when a dragon token is still in its cave.

We also propose the following new elements to improve the prototype:

a) Encapsulation of related classes into packages
b) Updated aesthetics:
    i) Player ID indicator that follows the dragon token as it moves
    ii) Wooden table texture used for background
c) New solution direction for animals: Define specific animal instances (BabyDragon, Bat, DragonPirate, Salamander, Spider) which extend the abstract Animal class. The Animal class contains a 'factory method' getAnimal() to return a new Animal instance based on the specified AnimalType(enum) and number of animals. This way, we isolate the nested if-else to a single point (Animal), and other parts of the code can perform animal type-checking using the AnimalType enum. (which also avoids having to cast types) This is open to future extensions (where new animal types are added), since we just need to:
    i) Add a new class extending Animal
    ii) Add a new AnimalType to the enums
    iii) Update the getAnimal() method to include the check for this new AnimalType
d) Decompose large methods into smaller, modular components.
e) Define a InitModel interface for classes that are responsible for instantiating classes (in the 'Model' package) for the first time during the initialisation stage of the game. This enforces separate initialisations of the View and the initialisation of the Model.

MONASH University

4. Object-Oriented Design

1. Class-Responsibility-Collaboration (CRC)

CRC Card for FieryDragonsApplication

- **Name of Class**: FieryDragonsApplication
- **Superclass**: GameApplication
- **Responsibility 1**: Initialise game (Collaborators: GameSettings, FXGL)
- **Responsibility 2**: Update game state (Collaborators: PlayerTurnManager, ChitCardFlipManager)
- **Responsibility 3**: Handle user input (Collaborators: Player, ChitCardAdapter)

Reason: main entry point of the game, extending the GameApplication class provided by the FXGL framework. It is responsible for initialising the game settings, UI, and game logic. It's a natural choice because it likely serves as the central hub through which other components are initialised and managed.

Alternative: Move the responsibility of handling specific user inputs (like card clicks) to a dedicated InputManager class. This would allow FieryDragonsApplication to focus on the game's lifecycle and state management, while InputManager translates raw input into actionable game events.

CRC Card for PlayerTurnManager

- **Name of Class**: PlayerTurnManager
- **Superclass**: N/A
- **Responsibility 1**: Manage player creation (Collaborators: Player)
- **Responsibility 2**: Handle turn transitions (Collaborators: Player, DragonToken)
- **Responsibility 3**: Move dragon tokens (Collaborators: Player, DragonToken, VolcanoCard)

Reason: This class manages player creation and the sequence of player turns, which is a critical aspect of the game's flow. It is responsible for transitioning between players' turns and is a key component in the game's state management.

Alternative: Create a TokenMovementManager that is responsible for moving tokens. This separates the logic of moving tokens from player management, allowing PlayerTurnManager to focus on turn order and state.

CRC Card for ChitCardFactory

- **Superclass**: SpawnFactory
- **Responsibility 1**: Spawn chit cards (Collaborators: FXGL, ChitCard)
- **Responsibility 2**: Randomize card positions (Collaborators: Utils)
- **Responsibility 3**: Provide card shapes (Collaborators: Circle)

Reason: The factory pattern is a common design pattern used in game development for object instantiation. This class is responsible for creating and spawning chit cards, which are essential game elements. It also holds information about the covered chit card shapes, making it a pivotal class for game interaction.

Alternative: Introduce a BoardLayoutManager that determines the layout and positioning of all game pieces, including chit cards. This way, ChitCardFactory can focus solely on the instantiation and spawning of cards.

CRC Card for ChitCardFlipManager

- **Superclass**: N/A
- **Responsibility 1**: Manage card flip animations (Collaborators: Circle, RotateTransition)
- **Responsibility 2**: Track uncovered cards (Collaborators: ChitCard, Circle)
- **Responsibility 3**: Reset cards at turn end (Collaborators: ChitCard, Circle)

Reason: This class manages the behaviour and animation of chit cards when they are flipped by the player. It is responsible for the game's interactive feedback, which is crucial for the player's engagement with the game.

Alternative: Animation logic could be moved to an AnimationManager class, which would handle all animations in the game, not just card flips. This would allow for a centralized control over animations and potentially enable reusing animation code.

CRC Card for VolcanoRingFactory

- **Superclass**: SpawnFactory
- **Responsibility 1**: Create volcano segments (Collaborators: VolcanoSegment, VolcanoCard)
- **Responsibility 2**: Arrange volcano ring (Collaborators: FXGL, Group)
- **Responsibility 3**: Randomize uncut segments (Collaborators: Utils)

Reason: This class is responsible for creating and arranging the volcano ring, which is likely a significant part of the game board. It incorporates randomness in the arrangement, adding an element of unpredictability to the game.

Alternative: The creation of game board elements could be abstracted into a BoardFactory class, which would be responsible for generating all board-related entities in a consistent manner. VolcanoRingFactory could then focus on the specifics of the volcano ring's unique arrangement.

CRC Card for DragonToken

2. **Superclass**: N/A (assuming it's a simple data class)
3. **Responsibility 1**: Represent player's token (Collaborators: Rectangle)
4. **Responsibility 2**: Move token on the game board (Collaborators: Timeline, KeyValue)
5. **Responsibility 3**: Track position and movement count (Collaborators: VolcanoCard)

Reason: The dragon token represents the player's piece on the game board and is moved according to the game's rules. It tracks the current position and movement count, which are essential for determining the game's progress and outcome.

Alternative: Moving the token could be the responsibility of a GamePieceMovementController, which would handle the logic of moving any game piece, including dragon tokens. This would allow DragonToken to act more as a data holder for the token's state, with all movement logic externalised.

Summary:
By redistributing these responsibilities, the system can achieve a clearer separation of concerns, which can lead to several benefits:

- **Improved Maintainability**: Changes to the logic of one aspect of the game (like animations) would be localized to a single class.
- **Enhanced Flexibility**: New features (like additional types of movements or animations) can be added with minimal impact on existing classes.
- **Easier Testing**: With fewer responsibilities per class, it's easier to write unit tests for individual components.

6. Class Diagram

5. Executable Instructions

YouTube Video Link: