MONASH University

FIT3077 Software Engineering: Architecture and Design

Sprint One (20%) Specifications

Group: MA_Wednesday04pm_Team888

Members: Arvind Siva, Lee Sing Yuan, Mohamed Areeb Ilham Riluwan, Tharani Prathaban

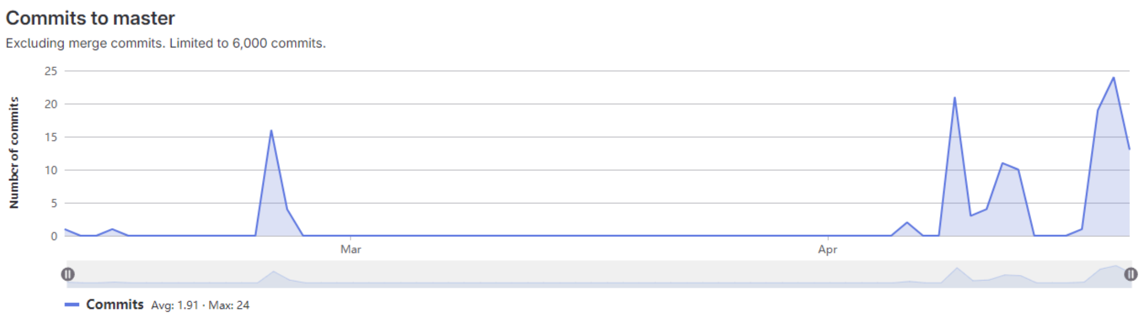
Due date: Wednesday, 27 March 2024

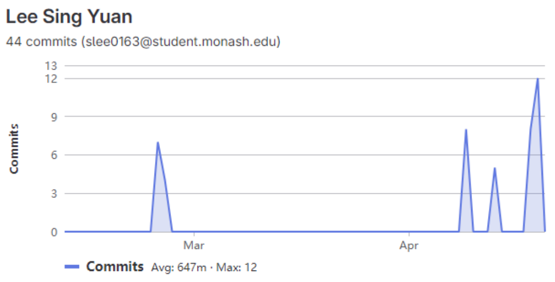
Company: Nexus DevOps

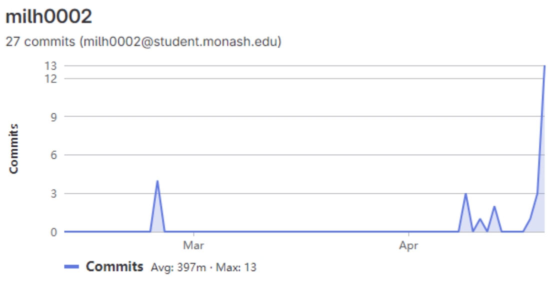
1. Introduction

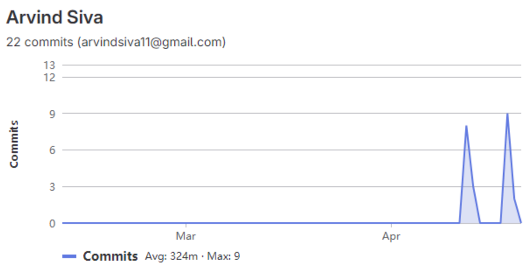
In the ever-evolving landscape of software development, iterative prototyping plays a crucial role in refining and enhancing product functionalities. Sprint 2 saw the culmination of diverse approaches from team members, each contributing a tech-based software prototype aimed at advancing our game development project. This report serves as a comprehensive review of the Sprint 2 tech-based prototypes, evaluating their strengths and weaknesses against predefined assessment criteria. By delving into the completeness, rationale, understandability, extensibility, code quality, and user interface aesthetics, we aim to gain insights into the efficacy of each solution direction. Furthermore, based on the outcomes of this assessment, we will outline a strategy for the creation of the tech-based prototype for Sprint 3, elucidating the integration of ideas and elements from Sprint 2 prototypes and identifying areas for further improvement. Through this comprehensive analysis, we endeavour to steer our development process towards delivering a robust and engaging gaming experience in subsequent sprints.

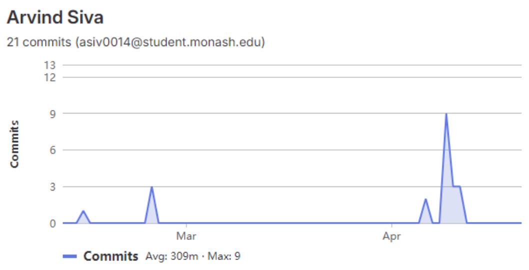
1.1. Contributor analytics

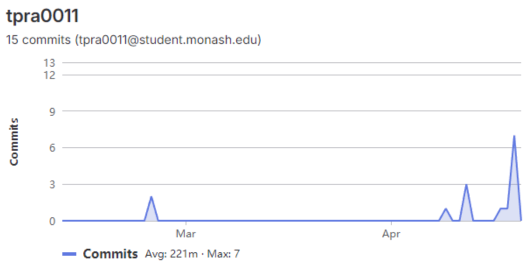


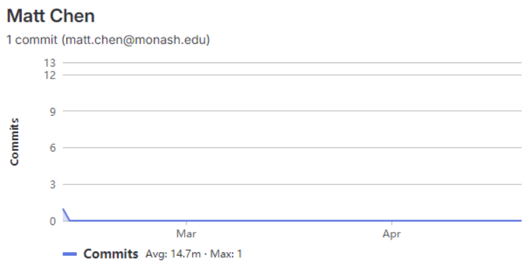












2. Assessment Criteria

2.1 Functional Stability

2.1.1 Functional Completeness

Design Principles	Metrics	Threshold
Feature 1 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	**> 80%** if required to be done from sprint 2, **N/A** if not required from sprint 2
Feature 2 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	
Feature 3 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	
Feature 4 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	
Feature 5 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	

2.1.2 Functional Correctness

Design Principles	Metrics	Threshold
Feature 1 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	**> 80%** if required to be done from sprint 2, **N/A** if not required from sprint 2
Feature 2 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	
Feature 3 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	
Feature 4 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	
Feature 5 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	

2.1.3 Functional Appropriateness

Design Principles	Metrics	Threshold
There exist functions which allow the game to deliver **feature 1** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	**> 3** if required to be done from sprint 2, **N/A** if not required from sprint 2
There exist functions which allow the game to deliver **feature 2** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	
There exist functions which allow the game to deliver **feature 3** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	
There exist functions which allow the game to deliver **feature 4** to the users with minimised mum functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy	

	3 is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	
There exist functions which allow the game to deliver **feature 5** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	

2.2 Performance Efficiency

2.2.1 Resource Utilisation

Design Principles	Metrics	Threshold
Reduce duplication of object instances that are required in multiple places	The **total memory** required by the game (MB). The lesser the memory used the better it is.	**Maximum 20MB** for Jar file only
Reuse UI elements without recreating them multiple times To further iterate. This is defined as reducing the amount of references pointing to the images in the resources.	● Presence of reusing assets for **dragon card UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **volcano tile UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **cave UI elements** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **player token UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability.	● **Yes** ● **Yes** ● **Yes** ● **Yes**

2.3 Maintainability

2.3.1 Modifiability

Design Principles	Metrics	Value
Presence of abstract classes with inheritance	• **Minimised** code redundancy using abstraction and inheritance for **Dragon Card. Yes** if achieved, **No** if not • **Minimised** code redundancy using abstraction and inheritance for **Animals. Yes** if achieved, **No** if not	• **Yes** • **Yes**

2.4 Flexibility

2.4.1 Scalability

Design Principles	Metrics	Threshold
Dynamic configurations	• VolcanoCard can handle different configurations. **Yes** if capable, **No** if incapable • The board can handle different configurations. **Yes** if capable, **No** if incapable • Dragon cards can handle different configurations. **Yes** if capable, **No** if incapable	• **Yes** • **Yes** • **Yes**

2.4.2 Adaptability

Design Principles	Metrics	Threshold
Create an executable that can run on all major desktop platforms	The OS that can run the executable.	**Windows** is the main target operating system

2.5 Interaction Capability

2.5.1 User Engagement

Design Principles	Metrics	Threshold
Ensure UI follows UI principles to be visually appealing	On a scale of **1 to 5**: **1** is absence in UI design principles **2** is basic application of UI design principles **3** adequate application of UI design principles **4** Significant application of UI design principles **5** Excellent application of UI design principles	**3**
Program's ability to respond to the user interaction	Time taken for the UI to change based on moves in the game	**0.001 to 0.1 ms**

2.5.2 User Error Protection

Design Principles	Metrics	Threshold
Prompt the user if incorrect data was entered	On a scale of **1 to 5,** based on the amount of time an error is detected and the user is prompted where: **1** delayed detection and absent feedback **2** moderate detection duration and absent feedback **3** reasonable detection duration with correct feedback **4** short detection duration and correct feedback **5** immediate detection and correct feedback	**3**
Maximise reduction of manual input from users such as text input box.	On a scale of **1 to 5,** based on the amount of reduction of manual input from the user where: **1** minimised reduction of manual user input	**3**

Another form of reduction could consist of limiting users to clicking on certain entities only or within a certain border only.	**2** limited reduction of manual user input **3** moderate reduction of manual user input **4** significant reduction of manual user input **5** maximum reduction of manual user input	

2.5.3 Appropriateness Recognizability

Design Principles	Metrics	Threshold
The UI design is consistent throughout the application. Consist of colour, theme, button placement etc.	On a scale of **1 to 5:** **1** inconsistent design **2** limited consistency **3** moderate consistency **4** good consistency **5** excellent consistency	**3**
The game mimics the original board game as much as possible	On a scale of **1 to 5**: **1** not the same as the actual game at all **2** minimal resemblance to the actual game **3** partial resemblance to the actual game **4** strong resemblance to the actual game **5** uncanny resemblance to the actual game	**4**

3.1. Prototype 1 (Arvind):

3.1.1. Assessment against criteria (individual ISO evaluation)

3.1.1.1. Functional Stability

Functional Completeness

Design Principles	Metrics	Value
Feature 1 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	100%
Feature 2 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	100%
Feature 3 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	100%
Feature 4 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	100%
Feature 5 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	100%

Functional Correctness

Design Principles	Metrics	Value
Feature 1 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	100%
Feature 2 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	100%
Feature 3 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	100%
Feature 4 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	100%
Feature 5 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	100%

Functional appropriateness

Design Principles	Metrics	Value
There exist functions which allow the game to deliver **feature 1** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	4
There exist functions which allow the game to deliver **feature 2** to the users with minimised functional	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous	4

redundancy.	redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	
There exist functions which allow the game to deliver **feature 3** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	4
There exist functions which allow the game to deliver **feature 4** to the users with minimised mum functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	5
There exist functions which allow the game to deliver **feature 5** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	5

3.1.1.2. Performance Efficiency

Resource Utilisation

Design Principles	Metrics	Value
Reduce duplication of object instances that are required in multiple places	The **total memory** required by the game (MB). The lesser the memory used the better it is.	17MB
Reuse UI elements without recreating them multiple times To further iterate. This is defined as reducing the amount of references pointing to the images in the resources.	• Presence of reusing assets for **dragon card UI element** which optimises the space complexity. Values range from **Yes**, means reusability is present while **No** means absence of reusability. • Presence of reusing assets for **volcano tile UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. • Presence of reusing assets for **cave UI elements** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. • Presence of reusing assets for **player token UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability.	• Yes • Yes • Yes • Yes

3.1.1.3. Maintainability

Modifiability

Design Principles	Metrics	Value
Presence of abstract classes with inheritance	• **Minimised** code redundancy using abstraction and inheritance for **Dragon Card. Yes** if achieved, **No** if not • **Minimised** code redundancy using abstraction and inheritance for **Animals. Yes** if achieved, **No** if not	• Yes • Yes

3.1.1.4. Flexibility

Scalability

Design Principles	Metrics	Value
Dynamic configurations	● VolcanoCard can handle different configurations. **Yes** if capable, **No** if incapable ● The board can handle different configurations. **Yes** if capable, **No** if incapable ● Dragon cards can handle different configurations. **Yes** if capable, **No** if incapable	● Yes ● Yes ● Yes

Adaptability

Design Principles	Metrics	Value
Create an executable that can run on all major desktop platforms	The OS that can run the executable.	Windows

4.1.1.5. Interaction Capability

User Engagement

Design Principles	**Metrics**	**Value**
Ensure UI follows UI principles to be visually appealing	On a scale of **1 to 5**: **1** is absence in UI design principles **2** is basic application of UI design principles **3** adequate application of UI design principles **4** Significant application of UI design principles **5** Excellent application of UI design principles	3
Program’s ability to respond to the user interaction	Time taken for the UI to change based on moves in the game	0.001 to 0.1 ms

User Error Protection

Design Principles	Metrics	Value
Prompt the user if incorrect data was entered	On a scale of **1 to 5,** based on the amount of time an error is detected and the user is prompted where: **1** delayed detection and absent feedback **2** moderate detection duration and absent feedback **3** reasonable detection duration with correct feedback **4** short detection duration and correct feedback **5** immediate detection and correct feedback	3
Maximise reduction of manual input from users such as text input box. Another form of reduction could consist of limiting users to clicking on certain entities only or within a certain border only.	On a scale of **1 to 5,** based on the amount of reduction of manual input from the user where: **1** minimised reduction of manual user input **2** limited reduction of manual user input **3** moderate reduction of manual user input **4** significant reduction of manual user input **5** maximum reduction of manual user input	3

Appropriateness Recognizability

Design Principles	Metrics	Value
The UI design is consistent throughout the application. Consist of colour, theme, button placement etc.	On a scale of **1 to 5:** **1** inconsistent design **2** limited consistency **3** moderate consistency **4** good consistency **5** excellent consistency	3
The game mimics the original board game as much as possible	On a scale of **1 to 5**: **1** not the same as the actual game at all **2** minimal resemblance to the actual game **3** partial resemblance to the actual game **4** strong resemblance to the actual game **5** uncanny resemblance to the actual game	4

3.1.2. Summary of key findings

Arvind was tasked with completing requirements 1 and 3. However, we as a team have found that Arvind has coded out the whole game completing all the features. We have not found any critical bugs in his code that affect the gameplay. Since all the features were completed and the code works correctly as per game rules we have scored Arvind’s code 100% for all the functional completeness and functional correctness design principles. Since the whole game was completed all 5 design principles covering the 5 requirements were evaluated against Arvind’s code. For features 4 and 5 we found no redundancy in the code as all repetitive logic was made into methods to reduce code repetition. For features 1,2 and 3 there was minimal presence of repeated code in some instances especially for feature 3 when it comes to the logic for moving the players. However, these repetitions were minimal so scores of 4 were given to features 1,2 and 3.

Next, we looked into the performance efficiency of the code by evaluating the resource utilisation. The code had reduced duplication of object instances required in multiple places by making use of the Singleton for the GameEngine, AnimalFactory and DisplayMain. This was further enhanced with the FlyWeight design pattern by ensuring only a single Animal instance of each animal exists in the game, greatly reducing the number of animal instances. Animal instances are greatly needed for various game components and it reduces resource utilisation. As such the total memory used by the game is very low at 10MB. Moreover, the code doesn't reuse the UI elements by not recreating them at every user interaction for most of the metrics evaluated. Only the DragonCard UI element was recreated at every user interaction. Since most of the UI elements are created once and reused it improves resource utilisation.

Next, Arvind’s code has a high maintainability. Looking into modifiability the code had abstraction and inheritance involved for DragonCard. Different DragonCards can be created with different movements to cater to different player movement logics. However, for the Animal class, it is not highly modifiable as it does not use abstraction and inheritance. The Animal class was using enum to differentiate it which is not highly modifiable if more animals were added to the game.

Looking into the flexibility of the code, the code is highly scalable as it passes all the metrics associated with dynamic configurations. The code can handle any number of game components making it easier to scale up the game. The code can handle a variety of board, volcano card and dragon card configurations. However, the code is not highly adaptable as the executable produced from the code only runs on Windows. This could be an issue with the game being compiled using JDK and JavaFX on Windows. This reduces adaptability as the executable is unable to run on a large variety of platforms.

Next, on interaction capability, in user engagement, the code is adequate in applying UI design principles to be visually appealing. It needs more UI design principles to further improve its user engagement. The program has a very high speed in responding to user interactions due to its high-performance efficiency. This high-speed interaction makes user engagement good in this code. The code was moderate for user error protection as it was able to detect errors in user input but did not often prompt the user to explain the error. However, the error is detected and the incorrect data is not put into the game which makes it moderate to prompt the user on error data. The code moderately reduces the amount of input from the user to reduce errors by using dropdown menus and automatically moving the player upon flipping a dragon card. On appropriateness recognizability, the game had a moderate consistency of UI design throughout the application. There were a bit of inconsistencies but most UI themes were consistent making it moderate. The game improves appropriateness and recognizability with a strong resemblance to the original Fiery Dragons board game. The UI was very close to the actual Fiery Dragons board but not exactly the same, giving it a strong resemblance to the actual game.

3.2. Prototype 2 (Lee Sing Yuan):

3.2.1. Assessment against criteria (individual ISO evaluation)

3.2.1.1. Functional Stability

Functional Completeness

Design Principles	Metrics	Value
Feature 1 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	100%
Feature 2 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	100%
Feature 3 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	60%
Feature 4 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	100%
Feature 5 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	100%

Functional Correctness

Design Principles	Metrics	Value
Feature 1 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	100%
Feature 2 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	100%
Feature 3 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as**	60%

	per game rules	
Feature 4 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	100%
Feature 5 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	100%

Functional appropriateness

Design Principles	Metrics	Value
There exist functions which allow the game to deliver **feature 1** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	3
There exist functions which allow the game to deliver **feature 2** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	4
There exist functions which allow the game to deliver **feature 3** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of	4

	redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	
There exist functions which allow the game to deliver **feature 4** to the users with minimised mum functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	4
There exist functions which allow the game to deliver **feature 5** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	4

3.2.1.2. Performance Efficiency

Resource Utilisation

Design Principles	Metrics	Value
Reduce duplication of object instances that are required in multiple places	The **total memory** required by the game (MB). The lesser the memory used the better it is.	35MB
Reuse UI elements without recreating them multiple times. To further iterate. This is defined as reducing the amount of references pointing to the images in the resources.	● Presence of reusing assets for **dragon card UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **volcano tile UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **cave UI elements** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **player token UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability.	● No ● No ● No ● No

3.2.1.3. Maintainability

Modifiability

Design Principles	Metrics	Value
Presence of abstract classes with inheritance	• **Minimised** code redundancy using abstraction and inheritance for **Dragon Card. Yes** if achieved, **No** if not • **Minimised** code redundancy using abstraction and inheritance for **Animals. Yes** if achieved, **No** if not	• Yes • Yes

3.2.1.4. Flexibility

Scalability

Design Principles	Metrics	Value
Dynamic configurations	● VolcanoCard can handle different configurations. **Yes** if capable, **No** if incapable ● The board can handle different configurations. **Yes** if capable, **No** if incapable ● Dragon cards can handle different configurations. **Yes** if capable, **No** if incapable	● Yes ● Yes ● Yes

Adaptability

Design Principles	Metrics	Value
Create an executable that can run on all major desktop platforms	The OS that can run the executable.	Windows

3.2.1.5. Interaction Capability

User Engagement

Design Principles	Metrics	Value
Ensure UI follows UI principles to be visually appealing	On a scale of **1 to 5**: **1** is absence in UI design principles **2** is basic application of UI design principles **3** adequate application of UI design principles **4** Significant application of UI design principles **5** Excellent application of UI design principles	4
Program's ability to respond to the user interaction	Time taken for the UI to change based on moves in the game	0.001 to 0.1 ms

User Error Protection

Design Principles	Metrics	Value
Prompt the user if incorrect data was entered	On a scale of **1 to 5,** based on the amount of time an error is detected and the user is prompted where: **1** delayed detection and absent feedback **2** moderate detection duration and absent feedback **3** reasonable detection duration with correct feedback **4** short detection duration and correct feedback **5** immediate detection and correct feedback	1
Maximise reduction of manual input from users such as text input box. Another form of reduction could consist of limiting users to clicking on certain entities only or within a certain border only.	On a scale of **1 to 5,** based on the amount of reduction of manual input from the user where: **1** minimised reduction of manual user input **2** limited reduction of manual user input **3** moderate reduction of manual user input **4** significant reduction of manual user input **5** maximum reduction of manual user input	3

Appropriateness Recognizability

Design Principles	Metrics	Value
The UI design is consistent throughout the application. Consist of colour, theme, button placement etc.	On a scale of **1 to 5:** **1** inconsistent design **2** limited consistency **3** moderate consistency **4** good consistency **5** excellent consistency	4
The game mimics the original board game as much as possible	On a scale of **1 to 5**: **1** not the same as the actual game at all **2** minimal resemblance to the actual game **3** partial resemblance to the actual game **4** strong resemblance to the actual game **5** uncanny resemblance to the actual game	4

3.2.2. Summary of key findings

As a team, we have determined that Sing Yuan has completed approximately 80% of the game's coding. However, rigorous testing revealed that the logic for the second feature did not perform as expected. Despite this issue, our evaluation and investigation showed that Sing Yuan's code contains minimal redundant functions.

In addition to functional redundancy, we have identified that Sing Yuan's code consumes a significant amount of system memory, which is suboptimal and could be improved. Therefore, this factor will be considered when integrating ideas and code from different team members.

On the other hand, Sing Yuan's code has demonstrated high adaptability to changes, resulting in a high maintainability evaluation. By effectively utilising inheritance and abstraction, his code will facilitate ongoing development and improvements as the project progresses.

In addition to maintainability, Sing Yuan's code is designed with significant flexibility, allowing it to run various configurations of the game.

However, when evaluating flexibility, it was found that Sing Yuan's executable ran only on Windows OS and failed on both Mac OS systems tested. This issue may be related to the specific machines used for testing, and while it is a concern, it does not significantly impact the consideration process.

In terms of user engagement, Sing Yuan's User Interface incorporated various design principles, such as Norman's 7 principles, with a focus on Visibility and Mapping. However, there is room for improvement. He did not provide clear instructions on how to play the game without referring to the readme.md file. Additionally, Sing Yuan did not apply Norman's feedback principle, as he did not validate user inputs or provide feedback when errors occurred. Furthermore, he did not adhere to Shneiderman's 8 principles, particularly in preventing errors by minimising the amount of manual input required from the user.

Despite the aforementioned issues, Sing Yuan's User Interface adheres to Shneiderman's principle of striving for consistency, consistently following a specific format in terms of colour schemes, button positions, and other elements. Additionally, his User Interface closely mimics the real-life board game. However, there is still room for improvement.

In conclusion, Sing Yuan's code is functionally correct for features 1,2,4,5, scalable, maintainable and has applied various key User Interface principles which will be taken into consideration upon integration.

3.3. Prototype 3 (Tharani):

3.1.1. Assessment against criteria (individual ISO evaluation)

3.1.1.1. Functional Stability

Functional Completeness

Design Principles	Metrics	Value
Feature 1 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	70%
Feature 2 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	100%
Feature 3 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	0%
Feature 4 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	0%
Feature 5 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	0%

Functional Correctness

Design Principles	Metrics	Value
Feature 1 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	60%
Feature 2 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	100%
Feature 3 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	0%
Feature 4 of the game	The **percentage of requirements** required	0%

works as per the game rules	for the feature that is **working correctly as per game rules**	
Feature 5 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	0%

Functional appropriateness

Design Principles	Metrics	Value
There exist functions which allow the game to deliver **feature 1** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	3
There exist functions which allow the game to deliver **feature 2** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	5
There exist functions which allow the game to deliver **feature 3** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of	N/A

	redundancy **5** is absence of redundancy	
There exist functions which allow the game to deliver **feature 4** to the users with minimised mum functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	N/A
There exist functions which allow the game to deliver **feature 5** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	N/A

3.1.1.2. Performance Efficiency

Resource Utilisation

Design Principles	Metrics	Value
Reduce duplication of object instances that are required in multiple places	The **total memory** required by the game (MB). The lesser the memory used the better it is.	34.4 MB
Reuse UI elements without recreating them multiple times To further iterate. This is defined as reducing the amount of references pointing to the images in the resources.	● Presence of reusing assets for **dragon card UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **volcano tile UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **cave UI elements** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **player token UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability.	● Yes ● Yes ● Yes ● Yes

3.1.1.3. Maintainability

Modifiability

Design Principles	Metrics	Value
Presence of abstract classes with inheritance	• **Minimised** code redundancy using abstraction and inheritance for **Dragon Card. Yes** if achieved, **No** if not • **Minimised** code redundancy using abstraction and inheritance for **Animals. Yes** if achieved, **No** if not	• Yes • N/A

3.1.1.4. Flexibility

Scalability

Design Principles	Metrics	Value
Dynamic configurations	• VolcanoCard can handle different configurations. **Yes** if capable, **No** if incapable • The board can handle different configurations. **Yes** if capable, **No** if incapable • Dragon cards can handle different configurations. **Yes** if capable, **No** if incapable	• Yes • Yes • Yes

Adaptability

Design Principles	Metrics	Value
Create an executable that can run on all major desktop platforms	The OS that can run the executable.	Tested on MacOS only

4.1.1.5. Interaction Capability

User Engagement

Design Principles	Metrics	Value
Ensure UI follows UI principles to be visually appealing	On a scale of **1 to 5**: **1** is absence in UI design principles **2** is basic application of UI design principles **3** adequate application of UI design principles **4** Significant application of UI design principles **5** Excellent application of UI design principles	4
Program's ability to respond to the user interaction	Time taken for the UI to change based on moves in the game	0.1ms - 5s

User Error Protection

Design Principles	Metrics	Value
Prompt the user if incorrect data was entered	On a scale of **1 to 5,** based on the amount of time an error is detected and the user is prompted where: **1** delayed detection and absent feedback **2** moderate detection duration and absent feedback **3** reasonable detection duration with correct feedback **4** short detection duration and correct feedback **5** immediate detection and correct feedback	N/A
Maximise reduction of manual input from users such as text input box. Another form of reduction could consist of limiting users to clicking on certain entities only or within a certain border only.	On a scale of **1 to 5,** based on the amount of reduction of manual input from the user where: **1** minimised reduction of manual user input **2** limited reduction of manual user input **3** moderate reduction of manual user input **4** significant reduction of manual user input **5** maximum reduction of manual user input	3

Appropriateness Recognizability

Design Principles	Metrics	Value
The UI design is consistent throughout the application. Consist of colour, theme, button placement etc.	On a scale of **1 to 5:** **1** inconsistent design **2** limited consistency **3** moderate consistency **4** good consistency **5** excellent consistency	5
The game mimics the original board game as much as possible	On a scale of **1 to 5**: **1** not the same as the actual game at all **2** minimal resemblance to the actual game **3** partial resemblance to the actual game **4** strong resemblance to the actual game **5** uncanny resemblance to the actual game	5

3.1.2. Summary of key findings

Feature 1, which involves user inputs such as the number of players, player names, ages, and token colour selection, met 70% of the requirements for functional completeness but only 60% for functional correctness. The missing elements include prompts for these user inputs, which are essential for the initial game setup. The absence of these inputs impacts both the completeness and correctness, highlighting a significant area for future development. Feature 2 was exemplary, achieving 100% in both functional completeness and correctness. This feature handles the flipping of dragon cards in the middle of the board, revealing the card on the other side and randomising their positions every time the game is run. This indicates that Tharani implemented this feature thoroughly and accurately according to the game rules. Features 3, 4, and 5 were not implemented in this iteration, resulting in 0% completion and correctness. As per the sprint 2 requirements, Tharani focused solely on Features 1 and 2, which is why the other features were not addressed. Future iterations should prioritise these features to enhance the game's functionality and completeness.

Tharani's prototype of the game uses 34.4 MB of memory, which is within an acceptable range for a game like FieryDragons and comparable to similar games. This indicates efficient resource utilisation. The reuse of assets for various UI elements, such as the dragon card, volcano tile, cave, and player token, was effectively implemented without any challenges, improving performance by optimising space complexity and reducing memory footprint.

Maintainability was assessed by examining the use of abstraction and inheritance. Tharani successfully minimised code redundancy for Dragon Cards by using an abstract class extended by other dragon card types like bat and salamander. However, for animals, she opted to define the animal type as a string within each dragon card and other components, which could benefit from further abstraction to enhance code maintainability and clarity.

The game demonstrated significant flexibility, capable of handling different configurations for VolcanoCard, the board, and Dragon cards. This was achieved using the Collections.shuffle() method, which ensures varied and dynamic gameplay. This flexibility enhances the game's replayability and user engagement. Currently, the executable runs only on MacOS. Although the jar file for this JavaFX application is designed to be cross-platform, testing was limited to MacOS. Future iterations should expand compatibility to other operating systems to broaden the user base.

The prototype scored well on user engagement, with a UI design that follows significant design principles, rated a 4 on a scale of 1 to 5. However, it lacked user instructions, which Tharani acknowledged as an oversight due to time constraints and the absence of this as a core requirement in sprint 2. Future iterations should include clear and comprehensive user instructions to improve user experience. User error protection was not implemented, as user input processing before the game starts was not addressed. Implementing error detection and feedback mechanisms is crucial for enhancing user interaction and minimising errors during gameplay.

The game’s resemblance to the original board game significantly enhances user experience. The design of the game board followed the exact ring pattern of the original, and the use of actual game element pictures for the UI representation contributed to a visually appealing and familiar interface. This resemblance was rated a 5, indicating an uncanny resemblance to the actual game, which is vital for user satisfaction and engagement.

Overall, Tharani's prototype shows great potential, with strong performance in functional implementation and UI design. Key areas for improvement include completing the missing features, enhancing user interaction with prompts and error protection. By addressing these areas, future iterations can provide a more comprehensive and engaging user experience, fully realising the potential of the FieryDragons game.

3.4. Prototype 4 (Areeb):

3.4.1. Assessment against criteria (individual ISO evaluation)

3.4.1.1. Functional Stability

Functional Completeness

Design Principles	Metrics	Value
Feature 1 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	80%
Feature 2 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	N/A
Feature 3 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	N/A
Feature 4 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	50%
Feature 5 of the game works as per the game rules	The **percentage of requirements fulfilled** to complete the feature	N/A

Functional Correctness

Design Principles	Metrics	Value
Feature 1 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	90%
Feature 2 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	N/A
Feature 3 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	N/A
Feature 4 of the game	The **percentage of requirements** required	70%

works as per the game rules	for the feature that is **working correctly as per game rules**	
Feature 5 of the game works as per the game rules	The **percentage of requirements** required for the feature that is **working correctly as per game rules**	N/A

Functional appropriateness

Design Principles	Metrics	Value
There exist functions which allow the game to deliver **feature 1** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	3
There exist functions which allow the game to deliver **feature 2** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	N/A
There exist functions which allow the game to deliver **feature 3** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of	N/A

	redundancy **5** is absence of redundancy	
There exist functions which allow the game to deliver **feature 4** to the users with minimised mum functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	2
There exist functions which allow the game to deliver **feature 5** to the users with minimised functional redundancy.	Minimisation of redundant functions,on a scale of **1 to 5** where: **1** is presence of numerous redundancies **2** is moderate presence of redundancy **3** is slight presence of redundancy **4** is minimal presence of redundancy **5** is absence of redundancy	N/A

3.4.1.2. Performance Efficiency

Resource Utilisation

Design Principles	Metrics	Value
Reduce duplication of object instances that are required in multiple places	The **total memory** required by the game (MB). The lesser the memory used the better it is.	14MB
Reuse UI elements without recreating them multiple times To further iterate. This is defined as reducing the amount of references pointing to	● Presence of reusing assets for **dragon card UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means	● Yes ● Yes ● Yes ● Yes

the images in the resources.	absence of reusability. ● Presence of reusing assets for **volcano tile UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **cave UI elements** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability. ● Presence of reusing assets for **player token UI element** which optimises the space complexity . Values range from **Yes**, means reusability is present while **No** means absence of reusability.	

3.4.1.3. Maintainability

Modifiability

Design Principles	Metrics	Value
Presence of abstract classes with inheritance	● **Minimised** code redundancy using abstraction and inheritance for **Dragon Card. Yes** if achieved, **No** if not ● **Minimised** code redundancy using abstraction and inheritance for **Animals. Yes** if achieved, **No** if not	● Yes ● Yes

3.4.1.4. Flexibility

Scalability

Design Principles	Metrics	Value
Dynamic configurations	● VolcanoCard can handle different configurations. **Yes** if capable, **No** if incapable ● The board can handle different configurations. **Yes** if capable, **No** if incapable ● Dragon cards can handle different configurations. **Yes** if capable, **No** if incapable	● Yes ● Yes ● Yes

Adaptability

Design Principles	Metrics	Value
Create an executable that can run on all major desktop platforms	The OS that can run the executable.	MacOS

3.4.1.5. Interaction Capability

User Engagement

Design Principles	Metrics	Value
Ensure UI follows UI principles to be visually appealing	On a scale of **1 to 5**: **1** is absence in UI design principles **2** is basic application of UI design principles **3** adequate application of UI design principles **4** Significant application of UI design principles **5** Excellent application of UI design principles	2
Program's ability to respond to the user interaction	Time taken for the UI to change based on moves in the game	0.001 to 0.1 ms

User Error Protection

Design Principles	Metrics	Value
Prompt the user if incorrect data was entered	On a scale of **1 to 5,** based on the amount of time an error is detected and the user is prompted where: **1** delayed detection and absent feedback **2** moderate detection duration and absent feedback **3** reasonable detection duration with correct feedback **4** short detection duration and correct feedback **5** immediate detection and correct feedback	5
Maximise reduction of manual input from users such as text input box. Another form of reduction could consist of limiting users to clicking on certain entities only or within a certain border only.	On a scale of **1 to 5,** based on the amount of reduction of manual input from the user where: **1** minimised reduction of manual user input **2** limited reduction of manual user input **3** moderate reduction of manual user input **4** significant reduction of manual user input **5** maximum reduction of manual user input	3

Appropriateness Recognizability

Design Principles	**Metrics**	**Value**
The UI design is consistent throughout the application. Consist of colour, theme, button placement etc.	On a scale of **1 to 5:** **1** inconsistent design **2** limited consistency **3** moderate consistency **4** good consistency **5** excellent consistency	5
The game mimics the original board game as much as possible	On a scale of **1 to 5**: **1** not the same as the actual game at all **2** minimal resemblance to the actual game **3** partial resemblance to the actual game **4** strong resemblance to the actual game **5** uncanny resemblance to the actual game	3

3.4.2. Summary of key findings

The team identified several areas for improvement in Areeb's code, particularly in the user interface (UI) and overall functionality of the application. The UI requires enhancements in image quality and functionality, as current resizing issues cause dragon cards to move when the screen is extended, disrupting the user experience. These UI problems have resulted in incomplete Req4 functionality, as it relies on the implementation of other requirements that are hindered by these issues. All UI components are currently housed within a single board class, indicating a need for a more modular approach to improve maintainability and scalability. Additionally, the use of inheritance is minimal, suggesting potential for architectural improvements to enhance code reuse and organisation. The board implementation is simple and efficient, generating cards randomly rather than hardcoding them, which enhances the game's replayability and reduces manual configuration efforts. Furthermore, the process of adding caves and players is streamlined based on the number of tiles on the board, ensuring players face each other for a more intuitive and balanced gameplay experience. Addressing these issues would significantly enhance the overall user experience and functionality of Areeb's application.

4. Plan for Sprint 3:

4.1. Incorporated Ideas/Elements:

Prototype	Incorporated Key Ideas	Justification
Tharani's Prototype	● GameComponentsGenerator class	Tharani's implementation efficiently generates game components such as dragon cards, volcano tiles, caves, and player tokens and leaves the responsibility of generating and managing it to one particular class. This component streamlines the setup process, and ensures Single Responsibility Principle
Bernard's Prototype	● PlayerManager class ● Animal abstract class	Bernard's PlayerManager handles player-related functionalities, including player registration and turn management. Integrating PlayerManager enhances the game's multiplayer aspect, providing seamless player interactions. Bernard's implementation includes an Animal abstract class, providing a foundation for different animal types. Utilising the Animal abstract class reduces code duplication and enhances flexibility, allowing for easier implementation of new animal types with shared functionality.
Arvind's Prototype	● GameEngine ● State abstract class	Arvind's implementation of the game logic includes a GameEngine class that fully controls the logic of game actions, tracks the current state, and ensures a single instance manages the game This design centralises game logic, facilitating easier maintenance and scalability of the game system. Arvind's implementation of the game includes the State class extended by a

		few game states such as start, end, nextTurn and so on which is used by GameEngine class to inform the display (UI) what to render based on the current game state.This design ensures a clear separation of concerns between game logic and UI rendering, enhancing code organisation and readability.
	● AnimalFactory class	Arvind’s implementation of representing the different animal types in the game includes the AnimalFactory class that provides singleton access to shared Animal instances for use in DragonCard, VolcanoCard, and Cave. This is used together with a flyweight design pattern to ensure that any VolcanoCard or DragonCard or Cave having the same Animal share the same instance which reduces duplication of the same Animal instance. This design enhances resource management and promotes consistency in animal representation across different game components.
	● InnerDisplay interface	Arvind’s implementation of showing the game display includes breaking down display functionalities into separate classes using interfaces to render the board, player details, and dragon cards within the main display. This design promotes modularity and extensibility, allowing for easy customization and future expansion of display elements.

4.2. New Ideas/Elements:

New Ideas	Justification
DragonCardManager class	The DragonCardManager class handles storage and metadata for dragon cards, including tracking the last flipped card, initialization, access, manipulation, shuffling, and determining if all cards are flipped open. Implementing this design ensures efficient management and interaction with dragon cards, promoting better organisation,

	maintainability, modular development, and code scalability within the game architecture.
DisplayDetails class	The DisplayDetails manages the visual representation of player turns, last flipped dragon card details, and action buttons for game progression within the UI. Implementing DisplayDetails enhances player interaction and engagement by providing clear visual cues and options for game actions. This design promotes a user-friendly experience, facilitating smooth gameplay and progression through intuitive UI elements.

5. Review Conclusion:

The selection of the prototype for Sprint 3 is justified by its comprehensive feature set, which integrates key components from Tharani's, Bernard's, and Arvind's prototypes. This amalgamation covers all essential aspects of gameplay, encompassing setup, player management, game logic, and UI rendering. By combining these elements, the prototype ensures a holistic and robust gaming experience that meets the requirements for functionality and engagement. Additionally, Arvind's code base will serve as the foundational framework for Sprint 3, given its highest degree of completeness across all aspects. His prototype already encompasses essential features, making it the most comprehensive among the Sprint 2 prototypes. Rather than refining from scratch, Arvind's code will be augmented by adopting key ideas from other prototypes to further enhance its functionality and user experience, ensuring a polished and robust final product while emphasising seamless integration, modularity, and user-centric design principles. The overarching goal of Sprint 3 is to deliver an immersive and enjoyable gaming experience for players.

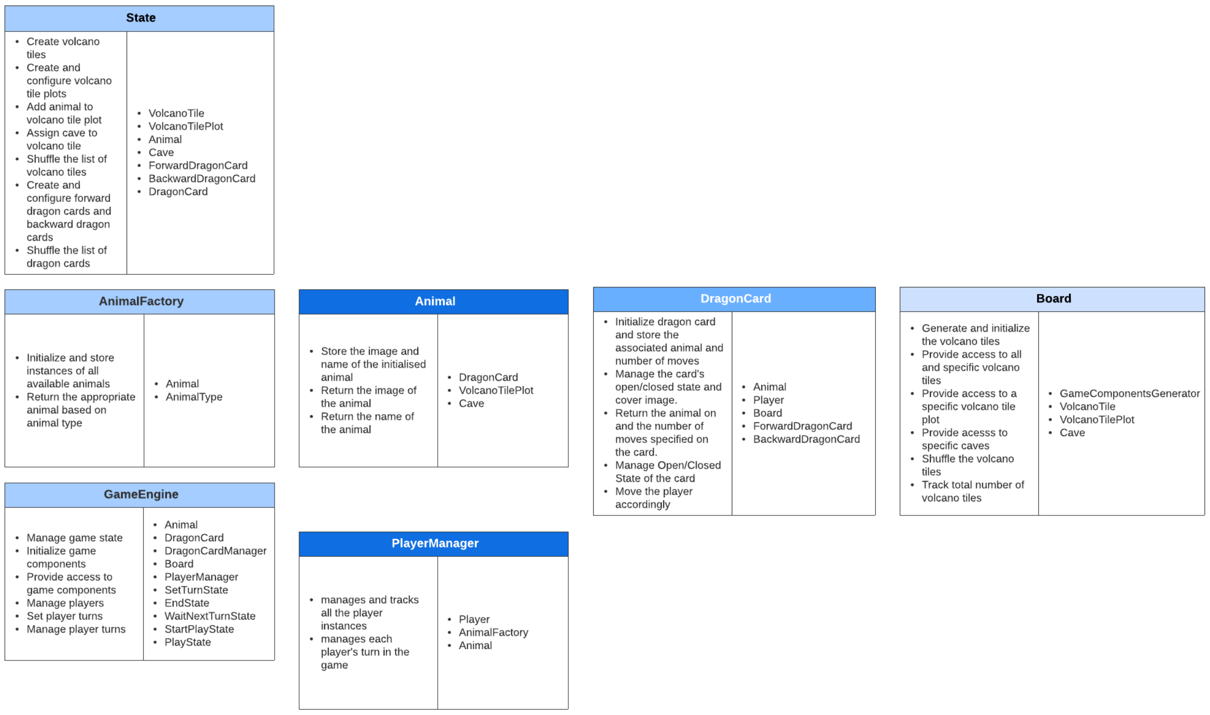
6. Description of executable

The executable currently in the repository was created from a windows' Operating System. If you would like to run it on other platforms, Please rebuild the artefact following the steps in the Readme.md. The instructions to build the artefact is also present in the Readme.md

It could be summarise like this:

1. Clone the project on your local machine
2. Open the project in Intellij
3. Wait for Maven and all the project dependencies to set up
4. In Intellij, navigate to File > Project Structure > Artefacts
5. Press the add button and select JAR
6. Select Main as the main class
7. Press OK
8. Press Apply, then press OK
9. In Intellij, navigate to Build > Build Artefacts
10. Press Build for FieryDragons:jar

7. CRC cards:



State	
• Create volcano tiles • Create and configure volcano tile plots • Add animal to volcano tile plot • Assign cave to volcano tile • Shuffle the list of volcano tiles • Create and configure forward dragon cards and backward dragon cards • Shuffle the list of dragon cards	• VolcanoTile • VolcanoTilePlot • Animal • Cave • ForwardDragonCard • BackwardDragonCard • DragonCard

AnimalFactory	
• Initialize and store instances of all available animals • Return the appropriate animal based on animal type	• Animal • AnimalType

Animal	
• Store the image and name of the initialised animal • Return the image of the animal • Return the name of the animal	• DragonCard • VolcanoTilePlot • Cave

DragonCard	
• Initialize dragon card and store the associated animal and number of moves • Manage the card's open/closed state and cover image. • Return the animal on and the number of moves specified on the card. • Manage Open/Closed State of the card • Move the player accordingly	• Animal • Player • Board • ForwardDragonCard • BackwardDragonCard

Board	
• Generate and initialize the volcano tiles • Provide access to all and specific volcano tiles • Provide access to a specific volcano tile plot • Provide acesss to specific caves • Shuffle the volcano tiles • Track total number of volcano tiles	• GameComponentsGenerator • VolcanoTile • VolcanoTilePlot • Cave

GameEngine	
• Manage game state • Initialize game components • Provide access to game components • Manage players • Set player turns • Manage player turns	• Animal • DragonCard • DragonCardManager • Board • PlayerManager • SetTurnState • EndState • WaitNextTurnState • StartPlayState • PlayState

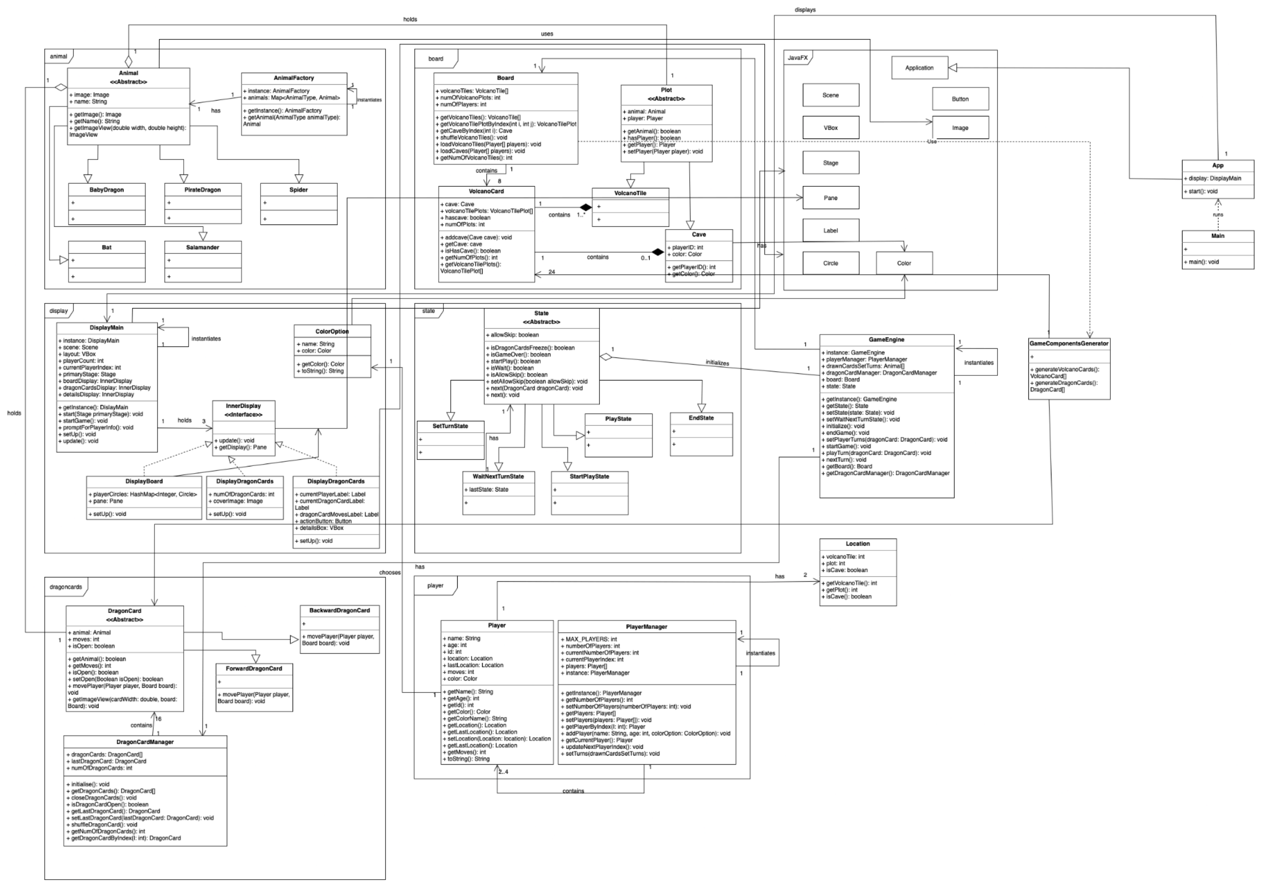
PlayerManager	
• manages and tracks all the player instances • manages each player's turn in the game	• Player • AnimalFactory • Animal

Class	Purpose
State	The class is used exclusively by the GameEngine to inform the display (UI) what to render based on the current game state and what actions to perform on user interaction.
GameEngine	Controls the logic of game actions, tracks the current state, and ensures a single instance manages the game state.
PlayerManager	This class is responsible for managing player-related operations in the game.
AnimalFactory	Provides singleton access to shared Animal instances for use in DragonCard, VolcanoTilePlot, and Cave.
Animal	Represents an abstract class that represents an animal that can be used on a DragonCard, in a VolcanoTilePlot, and in a Cave.
Dragoncard	Represents an abstract dragon card that has an associated animal and number of moves, and can move a player on the board.
Board	Manages the collection of VolcanoTile objects that make up the game board.

Discarded Alternatives	Reason
Animal class is not implemented as an abstract class	Led to code duplication and reduced flexibility, as each animal class would have to implement the same methods independently for different animal types
DragonCard class is not implemented as an abstract class	Lacked polymorphism, making it difficult to create and manage different types of dragon cards uniformly.
State class is not implemented	Resulted in complex game logic being spread across multiple classes, leading to a lack of clear separation of concerns and making the code harder to maintain and extend.
PlayerManager class is not implemented	Without a factory pattern, it became cumbersome to manage the creation of Animal instances, increasing the risk of inconsistent object creation and making it harder to manage shared instances.
GameComponentsGenerator class is not implemented	Having the Player class handle everything led to a violation of the single responsibility principle, making the Player class overly complex and difficult to manage.

8. UML Diagram:

The class diagram's original image is also present within the GitLab repository in the Documents folder. Please refer to that png if this png is unclear.


holds
uses
displays
animal
Animal <<Abstract>>
+ image: Image
+ name: String
+ getImage(): Image
+ getName(): String
+ getImageView(double width, double height): ImageView
AnimalFactory
+ instance: AnimalFactory
+ animals: Map<AnimalType, Animal>
+ getInstance(): AnimalFactory
+ getAnimal(AnimalType animalType): Animal
instantiates
has
BabyDragon
PirateDragon
Spider
Bat
Salamander
board
Board
+ volcanoTiles: VolcanoTile[]
+ numOfVolcanoPlots: int
+ numOfPlayers: int
+ getVolcanoTiles(): VolcanoTile[]
+ getVolcanoTilePlotByIndex(int i, int j): VolcanoTilePlot
+ getCaveByIndex(int i): Cave
+ shuffleVolcanoTiles(): void
+ loadVolcanoTiles(Player[] players): void
+ loadCaves(Player[] players): void
+ getNumOfVolcanoTiles(): int
Plot <<Abstract>>
+ animal: Animal
+ player: Player
+ getAnimal(): boolean
+ hasPlayer(): boolean
+ getPlayer(): Player
+ setPlayer(Player player): void
contains
VolcanoCard
+ cave: Cave
+ volcanoTilePlots: VolcanoTilePlot[]
+ hascave: boolean
+ numOfPlots: int
+ addcave(Cave cave): void
+ getCave: cave
+ isHasCave(): boolean
+ getNumOfPlots(): int
+ getVolcanoTilePlots(): VolcanoTilePlot[]
VolcanoTile
Cave
+ playerID: int
+ color: Color
+ getPlayerID(): int
+ getColor(): Color
1..*
0..1
24
JavaFX
Application
Scene
VBox
Stage
Pane
Label
Circle
Button
Image
Color
Use
App
+ display: DisplayMain
+ start(): void
runs
Main
+ main(): void
display
DisplayMain
+ instance: DisplayMain
+ scene: Scene
+ layout: VBox
+ playerCount: int
+ currentPlayerIndex: int
+ primaryStage: Stage
+ boardDisplay: InnerDisplay
+ dragonCardsDisplay: InnerDisplay
+ detailsDisplay: InnerDisplay
+ getInstance(): DislayMain
+ start(Stage primaryStage): void
+ startGame(): void
+ promptForPlayerInfo(): void
+ setUp(): void
+ update(): void
ColorOption
+ name: String
+ color: Color
+ getColor(): Color
+ toString(): String
InnerDisplay <<Interface>>
+ update(): void
+ getDisplay(): Pane
DisplayBoard
+ playerCircles: HashMap<Integer, Circle>
+ pane: Pane
+ setUp(): void
DisplayDragonCards
+ numOfDragonCards: int
+ coverImage: Image
+ setUp(): void
DisplayDragonCards
+ currentPlayerLabel: Label
+ currentDragonCardLabel: Label
+ dragonCardMovesLabel: Label
+ actionButton: Button
+ detailsBox: VBox
+ setUp(): void
state
State <<Abstract>>
+ allowSkip: boolean
+ isDragonCardsFreeze(): boolean
+ isGameOver(): boolean
+ startPlay(): boolean
+ isWait(): boolean
+ isAllowSkip(): boolean
+ setAllowSkip(boolean allowSkip): void
+ next(DragonCard dragonCard): void
+ next(): void
SetTurnState
PlayState
EndState
WaitNextTurnState
+ lastState: State
StartPlayState
initializes
GameEngine
+ instance: GameEngine
+ playerManager: PlayerManager
+ drawnCardsSetTurns: Animal[]
+ dragonCardManager: DragonCardManager
+ board: Board
+ state: State
+ getInstance(): GameEngine
+ getState(): State
+ setState(state: State): void
+ setWaitNextTurnState(): void
+ initialize(): void
+ endGame(): void
+ setPlayerTurns(dragonCard: DragonCard): void
+ startGame(): void
+ playTurn(dragonCard: DragonCard): void
+ nextTurn(): void
+ getBoard(): Board
+ getDragonCardManager(): DragonCardManager
GameComponentsGenerator
+ generateVolcanoCards(): VolcanoCard[]
+ generateDragonCards(): DragonCard[]
Location
+ volcanoTile: int
+ plot: int
+ isCave: boolean
+ getVolcanoTile(): int
+ getPlot(): int
+ isCave(): boolean
chooses
dragoncards
DragonCard <<Abstract>>
+ animal: Animal
+ moves: int
+ isOpen: boolean
+ getAnimal(): boolean
+ getMoves(): int
+ isOpen(): boolean
+ setOpen(Boolean isOpen): boolean
+ movePlayer(Player player, Board board): void
+ getImageView(cardWidth: double, board: Board): void
BackwardDragonCard
+ movePlayer(Player player, Board board): void
ForwardDragonCard
+ movePlayer(Player player, Board board): void
DragonCardManager
+ dragonCards: DragonCard[]
+ lastDragonCard: DragonCard
+ numOfDragonCards: int
+ initialise(): void
+ getDragonCards(): DragonCard[]
+ closeDragonCards(): void
+ isDragonCardOpen(): boolean
+ getLastDragonCard(): DragonCard
+ setLastDragonCard(lastDragonCard: DragonCard): void
+ shuffleDragonCard(): void
+ getNumOfDragonCards(): int
+ getDragonCardByIndex(I: int): DragonCard
16
player
Player
+ name: String
+ age: int
+ id: int
+ location: Location
+ lastLocation: Location
+ moves: int
+ color: Color
+ getName(): String
+ getAge(): int
+ getId(): int
+ getColor(): Color
+ getColorName(): String
+ getLocation(): Location
+ getLastLocation(): Location
+ setLocation(Location: location): Location
+ getLastLocation(): Location
+ getMoves(): int
+ toString(): String
PlayerManager
+ MAX_PLAYERS: int
+ numberOfPlayers: int
+ currentNumberOfPlayers: int
+ currentPlayerIndex: int
+ players: Player[]
+ instance: PlayerManager
+ getInstance(): PlayerManager
+ getNumberOfPlayers(): int
+ setNumberOfPlayers(numberOfPlayers: int): void
+ getPlayers: Player[]
+ setPlayers(players: Player[]): void
+ getPlayerByIndex(I: int): Player
+ addPlayer(name: String, age: int, colorOption: ColorOption): void
+ getCurrentPlayer(): Player
+ updateNextPlayerIndex(): void
+ setTurns(drawnCardsSetTurns): void
2..4