FIT3077 Software Engineering: Architecture and Design

Fiery Dragons Game Software

Sprint Three

Luminary Team
[MA_Tuesday12pm_Team002]

Written by:
Koe Rui En
Tay Ming Hui
Wong Jia Xuan

Contents

1. Review of Sprint 2 Tech-based Software Prototype

1.1 Assessment Criteria

The quality characteristics are selected by using the ISO/IEC 25010 quality model, we will assess each other's quality of work both internally (the source game itself) and externally (the overall game). By assessing each other's work based on these quality characteristics, we can gain insights into strengths, drawbacks and areas for improvement. It promotes a structured and holistic approach to quality assurance and helps ensure that the project meets the desired standards and expectations.

Thus, we will define suitable assessment criteria that cover completeness, rationale, understandability, and extensibility of the solution direction, as well as the quality of the source code and aesthetics of the user interface, to assess each software prototype built in Sprint 2, as shown in Table 1.1.1 below.

Factor	Characteristic	Design Concept	Metric	Acceptable Value	Accepted/ Result
Functional Suitability (Game Features)	Functional Correctness	Randomisation Algorithm	The position of dragon cards are randomised when every game is loaded.	When flipping dragon cards, they will not match the previous game if the same position of the same dragon card is flipped when loading a new game.	☐
			The position of caves are randomised when every game is loaded.	The caves will be attached to different volcano cards' cuts when every new game is loaded.	☐
			The position of volcano cards are randomised when every game is loaded	The position of volcano cards is different when every new game is loaded.	☐
		Interactive Card Flipping Rule	The player can only flip one Dragon Card to move the Dragon Token.	No multiple Dragon Card selections before moving the Dragon Token.	☐
		Correct Card	Display the	100% accuracy in card content	☐

		Display and State	correct type of card and the card affects the state accurately.	display and all flipped cards show and affect with correct information.	
		Movement Calculation Implementation	Accuracy of Dragon Token movement	100% accuracy calculation where the movements that can be made by dragon token.	☐
		Change of turn to the next player	The flipped dragon(“chit”) card is different with the current position of the volcano card	A mismatched uncovered dragon card occurs, all uncovered dragon cards should be covered and the next dragon card will affect the next dragon token.	☐
			The player clicks the end button to end his/her turn.	A “End Turn” button is clicked, all uncovered dragon cards should be covered and the next dragon card will affect the next dragon token.	☐
			The player moves his/her token to the occupied position.	The player cannot his/her token when there is another token occupying the position of the volcano card.	☐
		Winning Conditions Logic	Correctness of winner determination	100% accuracy in winner determination across all scenarios.	☐
		Game End Trigger	Reliability of Game End Trigger	The game end should function correctly 100% when the winner is determined.	☐
	Functional Completeness	Board Setup Validation	Number of Volcano Cards	8 Volcano cards	☐
			Number of Dragon Cards and the initial state of Dragon Cards are covered.	16 Dragon Cards and covered.	☐
			Number of Tokens	Between 2-4 Tokens	☐

			Number of Caves	4 Caves	☐
	Functional Appropriateness	Intuitive Gameplay Mechanics	User Understanding	The peers who are not in FIT3077 can demonstrate a basic understanding of the game mechanics within the first gameplay.	☐
Performance Efficiency (Game Time Efficiency from the user's perspective)	Time Behaviour	Response of game actions	Response time of each action performed in the game	Response times should ideally be less than 5000ms when every action is performed.	☐
		Implement effective resource management to ensure graphics are loaded and managed in a way that minimises delays.	Response time of graphics loaded when starting a new game	Response times should ideally be less than 100ms when the user clicks on/off the button.	☐
Interaction Capability (User Interface)	Appropriateness Recognizability	User Interface Clarity	User Comprehension Test	At least 2 people in the group should be able to describe the game setup and basic rules.	☐
			Initial Setup Game	Setup should take no longer than 1 minute for new players.	☐
	User Engagement	Visual and Audio Stimuli	Session Length	An average session length of at least 5 minutes, indicating sustained engagement	☐
		Engaging Game Mechanics	Player Satisfaction Survey	Rate satisfaction on a scale of 1 to 5, 1 is the lowest, 5 is the highest, covering overall enjoyment. Aim for ratings of 3 or higher.	Rate in each subsection of section 1.2
Reliability (Code Quality)	Faultlessness	Robust Error Handling	Error Rate	An error rate of less than 0.1%, not major logic error, ensuring player experience a smooth gameplay.	☐
Maintainability (Code	Modifiability	Modular Architecture	Number of modules affected by typical	A single change on one component should affect no more than 3 main modules.	☐

Quality)			changes		
		Reliance on case analysis and /or down-casts	Number of down-casts within methods	0 down-cast	☐
	Modularity	Inheritance-based Design	Number of Interfaces/ Abstract Classes	Between 2-5 interfaces/abstract classes	☐
			Number of subclasses	Between 3-5 subclasses	☐
		Number of comments	Number of comments (in-line/javadocs) associated with the class	Between 20-30 [High quality]	☐
		Line of codes	Number of lines of codes in a class and method	Between 20-30 [High quality]	☐
Flexibility (Code Quality)	Scalability	Dynamic Resource Allocation	Scalability of game features	The volcano cards can have more than 3 Squares	☐
	Adaptability	Design can be adapted across other technologies	Number of GUI technologies can be applied to the design.	1 GUI	☐
Chidamber & Kemerer metrics suite(Design Quality, Code Quality)	WMC (Weighted Methods per Class)	N/A	Maximum WMC	The WMC is as low as possible	Defined in section 1.2
			Minimum WMC		
			Medium WMC		
	DIT(Depth of Inheritance Tree)		Maximum DIT	The DIT depends on purpose, a trade-off between reuse and ease of understanding.	
			Minimum DIT		
			Medium DIT		
	NOC (Number of children)		Maximum NOC	The NOC depends on purpose, a trade-off between reuse and requirements of method testing.	
			Minimum NOC		
			Medium NOC		

	CBO (Coupling Between Objects)		Maximum CBO	The CBO is as low as possible	
			Minimum CBO		
			Medium CBO		
	RFC (Response for Class)		Maximum RFC	The RFC is as low as possible	
			Minimum RFC		
			Medium RFC		
	LCOM (Lack of Cohesion of Methods)		Maximum LCOM	The LCOM is as low as possible	
			Minimum LCOM		
			Medium LCOM		

Table 1.1.1: Assessment Criteria Along With Metrics to Assess the Software Prototype

I. Additional: Assessment Criteria's Glossary

1. Error Rate:

Definition: The frequency of errors or bugs encountered by players during gameplay sessions.

1.1.2 Methodology of CK Metrics Stated In Assessment Criteria

We defined suitable assessment criteria for the quality characteristics of the ISO/IEC 20510 quality model, focusing on functional suitability, maintainability, performance efficiency, and user engagement. Additionally, CK metrics are used to gauge the quality of the code across various prototypes. The key findings will be presented by assessing the CK Metrics Suite using the CK metrics tool from GitHub[1]. The CK Metrics Suite consists of six metrics calculated for each class: WMC, DIT, NOC, CBO, RFC, and LCOM1, as described below.

1. **WMC (Weighted Methods per Class):**

- Definition:
  - WMC is the sum of the complexity of the methods of a class.
  - WMC = Number of Methods (NOM) when all method's complexity is considered as unity.
- Viewpoints
  - A higher WMC suggests more time and effort needed to develop and maintain the class.
  - The larger NOM the greater the impact on children.
  - Classes with many methods may be less reusable and more tailored to specific needs.

2. **DIT(Depth of Inheritance Tree):**

- Definition:
  - The maximum length from the node to the root of the tree.
- Viewpoints:
  - A higher DIT can mean more complexity and more inherited methods, which can be good for reuse but harder to understand.
  - A lower DIT suggests simpler structures with less reuse of methods from parent classes.

3. **NOC (Number of children):**

- Definition:
  - Number of immediate subclasses subordinated to a class in the class hierarchy.
- Viewpoints:
  - A greater NOC is:
    - the greater is the reuse.
    - the greater is the probability of improper abstraction of the parent class.
    - the greater the requirements of method testing in that class.
  - Small values of NOC, may be an indicator of lack of communication between different class designers.

4. **CBO (Coupling Between Objects):**

- Definition:
  - It is a count of the number of how many other classes a class interacts with.
- Viewpoints:
  - Lower CBO:
    - Improve modularity and promote encapsulation.
    - Indicates independence in the class, making its reuse easier.
    - It makes it easier to maintain and test a class.

5. **RFC (Response for Class):**

- Definition:
  - It is the total count of unique methods that can be executed in response to a call to any method of the class.
- Viewpoints:
  - A higher RFC indicates more complex interactions within the class, making it harder to test and maintain.

6. **LCOM (Lack of Cohesion of Methods):**

- Definition:
  - It measures how different the methods of a class are in terms of the class variables they use.
- Viewpoints:
  - High LCOM:
    - It indicates that it Increases complexity and does not promote encapsulation and implies classes should probably be split into two or more subclasses.
    - Helps to identify low-quality design.

Table 1.2.1 below shows a summary of the CK Metrics Suite guidelines, which consist of goals, levels, complexity, reusability and encapsulation, and modularity.

Metrics	Goal	Level	Complexity (To develop, to test and to maintain)	Reusability	Encapsulation, Modularity
WMC	Low	▼	▼	▲	
DIT	Trade-off	▼	▼	▼	
		▲	▲	▲	
NOC	Trade-off	▼	▼	▼	
		▲	▲	▲	
CBO	Low	▼	▼		▲
RFC	Low	▼	▼		
LCOM	Low	▼	▼		▲

Table 1.1.2.1: CK Metrics Guidelines[2]

1.2 Review of Sprint 2 Tech-Based Software Prototypes

This document presents a detailed review of the software prototypes developed by each team member during Sprint 2. The purpose of this review is to evaluate each prototype against a set of predefined criteria based on the ISO/IEC 20510 quality model and the Chidamber and Kemerer (CK) Metrics Suite. This evaluation will guide our design for Sprint 3.

Review of each software prototype is done at the next page.

1.2.1 Software Prototype 1: Koe Rui En

Factor	Characteristic	Design Concept	Metric	Acceptable Value	Accepted /Result
Functional Suitability (Game Features)	Functional Correctness	Randomisation Algorithm	The position of dragon cards are randomised when every game is loaded.	When flipping dragon cards, they will not match the previous game if the same position of the same dragon card is flipped when loading a new game.	☑
			The position of caves are randomised when every game is loaded.	The caves will be attached to different volcano cards' cuts when every new game is loaded.	☑
			The position of volcano cards are randomised when every game is loaded	The position of volcano cards is different when every new game is loaded.	☑
		Interactive Card Flipping Rule	The player can only flip one Dragon Card to move the Dragon Token.	No multiple Dragon Card selections before moving the Dragon Token.	☐
		Correct Card Display and State	Display the correct type of card and the card affects the state accurately.	100% accuracy in card content display and all flipped cards show and affect with correct information.	☐
		Movement Calculation Implementation	Accuracy of Dragon Token movement	100% accuracy calculation where the movements that can be made by dragon token.	☐
		Change of turn	The flipped	A mismatched uncovered dragon	☐

		to the next player	dragon("chit") card is different with the current position of the volcano card	card occurs, all uncovered dragon cards should be covered and the next dragon card will affect the next dragon token.	
			The player clicks the end button to end his/her turn.	A "End Turn" button is clicked, all uncovered dragon cards should be covered and the next dragon card will affect the next dragon token.	☐
			The player moves his/her token to the occupied position.	The player cannot his/her token when there is another token occupying the position of the volcano card.	☐
		Winning Conditions Logic	Correctness of winner determination	100% accuracy in winner determination across all scenarios.	☐
		Game End Trigger	Reliability of Game End Trigger	The game end should function correctly 100% when the winner is determined.	☐
	Functional Completeness	Board Setup Validation	Number of Volcano Cards	8 Volcano cards	☑
			Number of Dragon Cards and the initial state of Dragon Cards are covered.	16 Dragon Cards and covered.	☑
			Number of Tokens	Between 2-4 Tokens	☑
			Number of Caves	4 Caves	☑
	Functional Appropriateness	Intuitive Gameplay Mechanics	User Understanding	The peers who are not in FIT3077 can demonstrate a basic understanding of the game mechanics within the first gameplay.	☑
Performance Efficiency (Game Time Efficiency from the user's perspective)	Time Behaviour	Response of game actions	Response time of each action performed in the game	Response times should ideally be less than 5000ms when every action is performed.	☐
		Implement effective	Response time of graphics loaded	Response times should ideally be less than 100ms when the user	☐

		resource management to ensure graphics are loaded and managed in a way that minimises delays.	when starting a new game	clicks on/off the button.	
Interaction Capability (User Interface)	Appropriateness Recognizability	User Interface Clarity	User Comprehension Test	At least 2 people in the group should be able to describe the game setup and basic rules.	☑
			Initial Setup Game	Setup should take no longer than 1 minute for new players.	☐
	User Engagement	Visual and Audio Stimuli	Session Length	An average session length of at least 5 minutes, indicating sustained engagement	☐
		Engaging Game Mechanics	Player Satisfaction Survey	Rate satisfaction on a scale of 1 to 5, 1 is the lowest, 5 is the highest, covering overall enjoyment. Aim for ratings of 3 or higher.	4
Reliability (Code Quality)	Faultlessness	Robust Error Handling	Error Rate	An error rate of less than 0.1%, not major logic error, ensuring player experience a smooth gameplay.	☐
Maintainability (Code Quality)	Modifiability	Modular Architecture	Number of modules affected by typical changes	A single change on one component should affect no more than 3 main modules.	☑
		Reliance on case analysis and /or down-casts	Number of down-casts within methods	0 down-cast	☑
	Modularity	Inheritance-based Design	Number of Interfaces/Abstract Classes	Between 2-5 interfaces/abstract classes	☑
			Number of subclasses	Between 3-5 subclasses	☑
		Number of comments	Number of comments (in-line/javadocs)	Between 20-30 [High quality]	☐

			associated with the class		
		Line of codes	Number of lines of codes in a class and method	Between 20-30 [High quality]	☐
Flexibility (Code Quality)	Scalability	Dynamic Resource Allocation	Scalability of game features	The volcano cards can have more than 3 Squares	☑
	Adaptability	Design can be adapted across other technologies	Number of GUI technologies can be applied to the design.	1 GUI	☑
Chidamber & Kemerer metrics suite(Design Quality, Code Quality)	WMC (Weighted Methods per Class)	N/A	Maximum WMC	The WMC is as low as possible	16
			Minimum WMC		1
			Medium WMC		6
	DIT(Depth of Inheritance Tree)		Maximum DIT	The DIT depends on purpose, a trade-off between reuse and ease of understanding.	6
			Minimum DIT		1
			Medium DIT		1
	NOC (Number of children)		Maximum NOC	The NOC depends on purpose, a trade-off between reuse and requirements of method testing.	0
			Minimum NOC		0
			Medium NOC		0
	CBO (Coupling Between Objects)		Maximum CBO	The CBO is as low as possible	9
			Minimum CBO		0
			Medium CBO		2
	RFC (Response for Class)		Maximum RFC	The RFC is as low as possible	25
			Minimum RFC		0
			Medium RFC		6
	LCOM (Lack of Cohesion of Methods)		Maximum LCOM	The LCOM is as low as possible	47
			Minimum LCOM		0
			Medium LCOM		2

Table 1.2.1.1: Assessment Criteria Along With Metrics to Assess the Rui En's Software Prototype

Summary of key findings

This software prototype adopts the Model-View-Controller (MVC) architecture, separating the game's internal representations (model), the user interface (view), and the controller classes that link the model and view. Although not a Gang of Four (GoF) pattern, MVC effectively organises code into distinct roles, promoting clarity and maintainability. The UML diagram is clear and completed that covers all the key functionalities to be implemented.

- Completeness of the Solution Direction
  In Sprint 2, the prototype successfully implemented the initial setup of the game board, including randomised positioning for dragon cards, and the functionality for flipping dragon ("chit") cards. The prototype meets the functional requirements specified for Sprint 2, setting up the game board and displaying the relevant game components. However, it allows for multiple dragon cards to be clicked at the start without moving the dragon, indicating a partial fulfilment of functional correctness. The sequence diagrams of the 5 main features required in sprint 2 fulfil the requirements of functional completeness.

- Rationale Behind the Chosen Solution Direction (Functional Appropriateness)
  The design adheres to the Single Responsibility Principle(SRP) and Open/Closed Principle (OCP), enhancing maintainability and scalability. It integrates design patterns like the Factory Method for creating creatures to minimise code repetition and the Observer Pattern to synchronise the GameWindow (observer) with Game (subject), ensuring the UI remains updated with game changes. Using this design pattern will inevitably increase the number of classes.

- Understandability of the Solution Direction (Appropriateness Recognizability)
  The MVC structure is well-known among developers, aiding in the rapid understanding of the application's flow and structure. Documentation and modular design further enhance understandability.

- Extensibility of the Solution Direction (Modifiability)
  While the MVC architecture generally supports extensibility, this prototype faces challenges due to the presence of numerous fixed variables that require individual updates, indicating less flexibility for expansion without significant changes. The high coupling observed through the CBO metrics in certain classes suggests potential difficulties in modifying one component without affecting others. On the other side, this design

- Quality of the Written Source Code (Maintainability)
  - Cohesion and Coupling: The maximum of CBO is 9 which is the highest among the members, indicating there is at least a class that is not so independent, which makes it not easier to maintain and test. The NOC is 0, which may indicate the lack of communication between different classes. The median of RFC is 6 and the maximum

of RFC is 25, which indicates that it is harder to test and maintain if compared with the best one.

- Complexity: This design has the highest LCOM among the members, which is 2, indicating it is the most complex. it may not promote a better encapsulation and implies classes should probably be split into two or more subclasses.

Figure 1.2.1.1 shows the result of the CK metric after running Rui En's code, and the result was analysed and mentioned above.

class	type	cbo	wmc	dit	noc	rfc	lcom
Model.Card.DragonCard	class	3	15	1	0	12	43
Model.Card.CaveCard	class	4	11	5	0	12	30
Model.Card.VolcanoCard	class	3	8	5	0	9	17
Handler.CaveCardHandler	class	3	6	1	0	5	0
Model.Creatures.PirateDragon	class	2	4	1	0	1	2
UI.GameBoardSetup	class	5	16	5	0	25	27
UI.GameWindow	class	3	2	6	0	13	1
Model.Creatures.Bat	class	2	4	1	0	1	2
Model.Creatures.BabyDragon	class	2	4	1	0	1	2
Model.Creatures.Creature	interface	0	2	1	0	0	1
Model.Player	class	2	14	1	0	6	47
Model.Creatures.Salamander	class	2	4	1	0	1	2
Controller.DragonCardController	class	4	10	1	0	13	22
Model.Card.CardComponent	interface	1	3	1	0	0	3
Model.Creatures.Spider	class	2	4	1	0	1	2
Handler.DragonCardDeck	class	4	8	1	0	9	0
Handler.VolcanoCardHandler	class	4	8	1	0	9	0
Model.Dragon	class	1	10	1	0	8	20
Application	class	1	1	1	0	1	0
Model.Creatures.CreatureFactory	class	7	9	1	0	6	28

Figure 1.2.1.1: The CK metric of Rui En's code

- Aesthetics of the User Interface (User Engagement)
  The user interface's aesthetics have not been fully implemented. The buttons for actions that can be executed by the player are not implemented. The game board UI is similar to the actual game design.

1.2.2 Software Prototype 2: Tay Ming Hui

Factor	Characteristic	Design Concept	Metric	Acceptable Value	Accepted /Result
Functional Suitability (Game Features)	Functional Correctness	Randomisation Algorithm	The position of dragon cards are randomised when every game is loaded.	When flipping dragon cards, they will not match the previous game if the same position of the same dragon card is flipped when loading a new game.	☑
			The position of caves are randomised when every game is loaded.	The caves will be attached to different volcano cards' cuts when every new game is loaded.	☐
			The position of volcano cards are randomised when every game is loaded	The position of volcano cards is different when every new game is loaded.	☐
		Interactive Card Flipping Rule	The player can only flip one Dragon Card to move the Dragon Token.	No multiple Dragon Card selections before moving the Dragon Token.	☐
		Correct Card Display and State	Display the correct type of card and the card affects the state accurately.	100% accuracy in card content display and all flipped cards show and affect with correct information.	☑
		Movement Calculation Implementation	Accuracy of Dragon Token movement	100% accuracy calculation where the movements that can be made by dragon token.	☐

		Change of turn to the next player	The flipped dragon("chit") card is different with the current position of the volcano card	A mismatched uncovered dragon card occurs, all uncovered dragon cards should be covered and the next dragon card will affect the next dragon token.	☐
			The player clicks the end button to end his/her turn.	A "End Turn" button is clicked, all uncovered dragon cards should be covered and the next dragon card will affect the next dragon token.	☐
			The player moves his/her token to the occupied position.	The player cannot move his/her token when another token occupies the volcano card's position.	☐
		Winning Conditions Logic	Correctness of winner determination	100% accuracy in winner determination across all scenarios.	☑
		Game End Trigger	Reliability of Game End Trigger	The game end should function correctly 100% when the winner is determined.	☐
	Functional Completeness	Board Setup Validation	Number of Volcano Cards	8 Volcano cards	☑
			Number of Dragon Cards and the initial state of Dragon Cards are covered.	16 Dragon Cards and covered.	☑
			Number of Tokens	Between 2-4 Tokens	☑
			Number of Caves	4 Caves	☑
	Functional Appropriateness	Intuitive Gameplay Mechanics	User Understanding	The peers who are not in FIT3077 can demonstrate a basic understanding of the game mechanics within the first gameplay.	☑
Performance Efficiency (Game Time Efficiency from the user's perspective)	Time Behaviour	Response of game actions	Response time of each action performed in the game	Response times should ideally be less than 5000ms when every action is performed.	☑
		Implement effective resource	Response time of graphics loaded when starting a	Response times should ideally be less than 100ms when the user clicks on/off the button.	☐

		management to ensure graphics are loaded and managed in a way that minimises delays.	new game		
Interaction Capability (User Interface)	Appropriateness Recognizability	User Interface Clarity	User Comprehension Test	At least 2 people in the group should be able to describe the game setup and basic rules.	☑
			Initial Setup Game	Setup should take no longer than 1 minute for new players.	☑
	User Engagement	Visual and Audio Stimuli	Session Length	An average session length of at least 5 minutes, indicating sustained engagement	☑
		Engaging Game Mechanics	Player Satisfaction Survey	Rate satisfaction on a scale of 1 to 5, 1 is the lowest, 5 is the highest, covering overall enjoyment. Aim for ratings of 3 or higher.	5
Reliability (Code Quality)	Faultlessness	Robust Error Handling	Error Rate	An error rate of less than 0.1%, not major logic error, ensuring player experience a smooth gameplay.	☐
Maintainability (Code Quality)	Modifiability	Modular Architecture	Number of modules affected by typical Changes	A single change on one component should affect no more than 3 main modules.	☑
		Reliance on case analysis and /or down-casts	Number of down-casts within methods	0 down-cast	☑
	Modularity	Inheritance-based Design	Number of Interfaces/Abstract Classes	Between 2-5 interfaces/abstract classes	☑
			Number of subclasses	Between 3-5 subclasses	☑
		Number of comments	Number of comments (in-line/javadocs) associated with the class	Between 20-30 [High quality]	☐

		Line of codes	Number of lines of codes in a class and method	Between 20-30 [High quality]	☐
Flexibility (Code Quality)	Scalability	Dynamic Resource Allocation	Scalability of game features	The volcano cards can have more than 3 Squares	☑
	Adaptability	Design can be adapted across other technologies	Number of GUI technologies can be applied to the design.	1 GUI	☑
Chidamber & Kemerer metrics suite(Design Quality, Code Quality)	WMC (Weighted Methods per Class)		Maximum WMC	The WMC is as low as possible	23
			Minimum WMC		1
			Medium WMC		4
	DIT(Depth of Inheritance Tree)		Maximum DIT	The DIT depends on purpose, a trade-off between reuse and ease of understanding.	8
			Minimum DIT		1
			Medium DIT		2
	NOC (Number of children)		Maximum NOC	The NOC depends on purpose, a trade-off between reuse and requirements of method testing.	5
			Minimum NOC		0
			Medium NOC		0
	CBO (Coupling Between Objects)		Maximum CBO	The CBO is as low as possible	6
			Minimum CBO		0
			Medium CBO		2
	RFC (Response for Class)		Maximum RFC	The RFC is as low as possible	25
			Minimum RFC		0
			Medium RFC		5
	LCOM (Lack of Cohesion of Methods)		Maximum LCOM	The LCOM is as low as possible	87
			Minimum LCOM		0
			Medium LCOM		0

Table 1.2.2.1: Assessment Criteria Along With Metrics to Assess the Ming Hui's Software Prototype

Summary of key findings

This software prototype adopts an Object-Oriented (OO) design without using an MVC structure, which makes the overall design less complex. The UML diagram is completed but some key functionalities are not covered. However, it is believed to be the simplest software prototype based on the design and implementation.

- Completeness of the Solution Direction
  In Sprint 2, the prototype successfully implemented the initial setup of the game board, including randomised positioning for dragon cards, and the functionality for flipping dragon ("chit") cards. The prototype meets the functional requirements specified for Sprint 2, setting up the game board and displaying the relevant game components. However, it allows for multiple dragon cards to be clicked at the start without moving the dragon, indicating a partial fulfilment of functional correctness. The sequence diagrams of the 5 main features required in sprint 2 fulfil the requirements of functional completeness.

- Rationale Behind the Chosen Solution Direction (Functional Appropriateness)
  The design adheres to the Single Responsibility Principle (SRP) and Open/Closed Principle (OCP), enhancing maintainability and scalability. It utilises design patterns like the Bridge Pattern for creating ChitCards, Volcano Cards, Chit Board, and the Volcano CardBoard. Facade Pattern provides a simplified interface to manage the game's setup and execution. The Structure Pattern by using manager classes is implemented for this game to centralise functionality, encapsulate data, and provide a controlled interface, improving code organisation and maintainability.

- Understandability of the Solution Direction (Appropriateness Recognizability)
  The classes are easy to understand and straightforward. The rest of the members can understand the meaning of creating the related classes.

- Extensibility of the Solution Direction (Modifiability)
  The maximum of CBO is 6, which is the lowest among the members. The lower CBO indicates independence in the class and makes it easier to reuse. Furthermore, the maximum of NOC is 5, which is the 'animals.Animal' class, which ensures communication between the designs of different classes.

- Quality of the Written Source Code (Maintainability)
  - Cohesion and Coupling:
    - It has the lowest RFC among the members, making it easier to test and maintain.
  - Complexity:
    - It has the lowest LCOM which ensures the complexity is lower and promotes encapsulation and is identified as high quality design.

Figure 1.2.2.1 shows the result of the CK metric after running Ming Hui's code, and the result was analysed and mentioned above.

class	type	cbo	wmc	dit	noc	rfc	lcom
CaveCardManager	class	5	15	1	0	15	0
animals.PirateDragon	class	1	1	2	0	0	0
WinningPopupPanel	class	0	1	5	0	8	0
VolcanoCardManager	class	3	7	1	0	16	0
components.Token	class	5	14	8	0	15	8
animals.Bat	class	1	1	2	0	0	0
animals.Animal	class	1	4	1	5	1	4
components.ChitCard	class	3	6	6	0	9	2
components.VolcanoCard	class	3	4	6	0	8	2
animals.Salamander	class	1	1	2	0	0	0
VolcanoCardBoard	class	2	10	1	0	5	0
BoardSetup	class	3	23	5	0	25	87
ChitBoard	class	3	7	1	0	5	0
ChitCardManager	class	6	14	1	0	18	52
Game	class	5	7	6	0	18	0
animals.Spider	class	1	1	2	0	0	0
components.CaveCard	class	3	4	6	0	8	0
animals.BabyDragon	class	1	1	2	0	0	0
components.GameComponent	class	1	3	5	4	0	1
animals.AnimalFactory	class	6	6	1	0	2	15
Main	class	1	1	1	0	0	0

Figure 1.2.2.1: The CK metric of Ming Hui's code

- Aesthetics of the User Interface (User Engagement)
  The user interface is similar to the physical game board. The user interface showed the movement of the dragon token by dragging and also showed a clickable button. Although the actions that can be executed by the player are not fully implemented, the overall user interface is matched with the low-fidelity design in Sprint 1.

1.2.3 Software Prototype 3: Wong Jia Xuan

Factor	Characteristic	Design Concept	Metric	Acceptable Value	Accepted /Result
Functional Suitability (Game Features)	Functional Correctness	Randomisation Algorithm	The position of dragon cards are randomised when every game is loaded.	When flipping dragon cards, they will not match the previous game if the same position of the same dragon card is flipped when loading a new game.	☑
			The position of caves are randomised when every game is loaded.	The caves will be attached to different volcano cards' cuts when every new game is loaded.	☐
			The position of volcano cards are randomised when every game is loaded	The position of volcano cards is different when every new game is loaded.	☐
		Interactive Card Flipping Rule	The player can only flip one Dragon Card to move the Dragon Token.	No multiple Dragon Card selections before moving the Dragon Token.	☐
		Correct Card Display and State	Display the correct type of card and the card affects the state accurately.	100% accuracy in card content display and all flipped cards show and affect with correct information.	☐
		Movement Calculation Implementation	Accuracy of Dragon Token movement	100% accuracy calculation where the movements that can be made by dragon token.	☐

		Change of turn to the next player	The flipped dragon("chit") card is different with the current position of the volcano card	A mismatched uncovered dragon card occurs, all uncovered dragon cards should be covered and the next dragon card will affect the next dragon token.	☐
			The player clicks the end button to end his/her turn.	A "End Turn" button is clicked, all uncovered dragon cards should be covered and the next dragon card will affect the next dragon token.	☐
			The player moves his/her token to the occupied position.	The player cannot his/her token when there is another token occupying the position of the volcano card.	☐
		Winning Conditions Logic	Correctness of winner determination	100% accuracy in winner determination across all scenarios.	☐
		Game End Trigger	Reliability of Game End Trigger	The game end should function correctly 100% when the winner is determined.	☐
	Functional Completeness	Board Setup Validation	Number of Volcano Cards	8 Volcano cards	☑
			Number of Dragon Cards and the initial state of Dragon Cards are covered.	16 Dragon Cards and covered.	☑
			Number of Tokens	Between 2-4 Tokens	☑
			Number of Caves	4 Caves	☑
	Functional Appropriateness	Intuitive Gameplay Mechanics	User Understanding	The peers who are not in FIT3077 can demonstrate a basic understanding of the game mechanics within the first gameplay.	☐
Performance Efficiency (Game Time Efficiency from the user's perspective)	Time Behaviour	Response of game actions	Response time of each action performed in the game	Response times should ideally be less than 5000ms when every action is performed.	☑
		Implement effective	Response time of graphics loaded	Response times should ideally be less than 100ms when the user	☑

		resource management to ensure graphics are loaded and managed in a way that minimises delays.	when starting a new game	clicks on/off the button.	
Interaction Capability (User Interface)	Appropriateness Recognizability	User Interface Clarity	User Comprehension Test	At least 2 people in the group should be able to describe the game setup and basic rules.	☑
			Initial Setup Game	Setup should take no longer than 1 minute for new players.	☑
	User Engagement	Visual and Audio Stimuli	Session Length	An average session length of at least 5 minutes, indicating sustained engagement	☐
		Engaging Game Mechanics	Player Satisfaction Survey	Rate satisfaction on a scale of 1 to 5, 1 is the lowest, 5 is the highest, covering overall enjoyment. Aim for ratings of 3 or higher.	3
Reliability (Code Quality)	Faultlessness	Robust Error Handling	Error Rate	An error rate of less than 0.1%, not major logic error, ensuring player experience a smooth gameplay.	☐
Maintainability (Code Quality)	Modifiability	Modular Architecture	Number of modules affected by typical Changes	A single change on one component should affect no more than 3 main modules.	☐
		Reliance on case analysis and /or down-casts	Number of down-casts within methods	0 down-cast	☑
	Modularity	Inheritance-based Design	Number of Interfaces/Abstract Classes	Ensure each module interacting with another has at least one interface or abstract class, limiting direct class-to-class dependencies.	☐
			Number of subclasses	Between 3-5 subclasses	☐
		Number of comments	Number of comments (in-line/javadocs)	Between 20-30 [High quality]	☐

			associated with the class		
		Line of codes	Number of lines of codes in a class and method	Between 20-30 [High quality]	☐
Flexibility (Code Quality)	Scalability	Dynamic Resource Allocation	Scalability of game features	The volcano cards can have more than 3 Squares	☑
	Adaptability	Design can be adapted across other technologies	Number of GUI technologies can be applied to the design.	1 GUI	☑
Chidamber & Kemerer metrics suite(Design Quality, Code Quality)	WMC (Weighted Methods per Class)		Maximum WMC	The WMC is as low as possible	19
			Minimum WMC		0
			Medium WMC		3
	DIT(Depth of Inheritance Tree)		Maximum DIT	The DIT depends on purpose, a trade-off between reuse and ease of understanding.	6
			Minimum DIT		1
			Medium DIT		1
	NOC (Number of children)		Maximum NOC	The NOC depends on purpose, a trade-off between reuse and requirements of method testing.	0
			Minimum NOC		0
			Medium NOC		0
	CBO (Coupling Between Objects)		Maximum CBO	The CBO is as low as possible	7
			Minimum CBO		0
			Medium CBO		2
	RFC (Response for Class)		Maximum RFC	The RFC is as low as possible	17
			Minimum RFC		0
			Medium RFC		6
	LCOM (Lack of Cohesion of Methods)		Maximum LCOM	The LCOM is as low as possible	25
			Minimum LCOM		0
			Medium LCOM		1

Table 1.2.3.1: Assessment Criteria Along With Metrics to Assess Jia Xuan's Software Prototype

Summary of key findings

This software prototype adopts Model-View architecture, separating the game's internal representations (model), the user interface (view). Some unimportant classes are created to build the UI. The UML diagram is completed but some key functionalities are not covered. The testing code, UI code and game mechanics code are not grouped in a package which creates ambiguity.

- Completeness of the Solution Direction
  In Sprint 2, the prototype successfully implemented the initial setup of the game board, including randomised positioning for dragon cards, and the functionality for flipping dragon (“chit”) cards. The prototype meets the functional requirements specified for Sprint 2, setting up the game board and displaying the relevant game components. However, it allows for multiple dragon cards to be clicked at the start without moving the dragon, indicating a partial fulfilment of functional correctness.

- Rationale Behind the Chosen Solution Direction (Functional Appropriateness)
  The design adheres to the Single Responsibility and Open/Closed Principles, enhancing maintainability and scalability. It utilises design patterns like the Factory Method for creating game components, the Abstract Factory for creating Dragon Cards, and the Flyweight Pattern to implement the Dragon Card types. The Flyweight Pattern increased the class complexity.

- Understandability of the Solution Direction (Appropriateness Recognizability)
  It is challenging to figure out the code as the UI codes and testing code are all in the same package.

- Extensibility of the Solution Direction (Modifiability)
  Based on the CK metrics, there are a few aspects of modifiability tactics, which are increased cohesion, reduced coupling and defer binding. It is good to have an LCOM that is 0, there are few classes' LCOMs that are not 0, indicating that the design of the class is poor.

- Quality of the Written Source Code (Maintainability)
  Modifiability: The prototype shows some areas of concern with a high degree of coupling in classes like GameBoardPanel and VolcanoCardPanel, which might hinder future scalability and flexibility. The CK metrics highlight these issues with several classes showing high CBO and WMC scores, suggesting complex and tightly coupled code.
  Modularity: While the architecture supports modularity through its Model-View separation, the inclusion of some "unimportant" classes could dilute the overall modularity and clarity of the architecture.

Figure 1.2.3.1 shows the result of the CK metric after running Jia Xuan's code, and the result was analysed and mentioned above.

class	type	cbo	wmc	dit	noc	rfc	lcom
GameBoardPanel	class	8	3	5	0	12	0
TestSquareLabel	class	3	2	1	0	11	1
MainApplicationFrame	class	2	2	6	0	10	1
GameComponent	interface	0	1	1	0	0	0
AnimalCard	class	6	10	1	0	9	25
TestVolcanoCardPanel	class	4	1	1	0	10	0
DragonToken	class	1	14	1	0	1	0
SquareLabel	class	2	4	5	0	17	0
TestDragonCardPanel	class	4	1	1	0	8	0
ImageRepository	class	2	6	1	0	6	0
AnimalSpecies	class	1	3	1	0	2	1
GameComponentTest	class	7	4	6	0	11	6
CavePanel	class	2	2	5	0	11	1
Player	class	1	10	1	0	0	21
DragonCard	interface	3	2	1	0	0	1
VolcanoCardPanel	class	3	19	5	0	14	1
CardLabel	class	1	3	5	0	9	0
Square	class	2	5	1	0	1	6
Volcano	class	2	5	1	0	1	2
AnimalFactory	class	1	2	1	0	2	0
Cave	class	3	6	1	0	1	3
Game	class	3	8	1	0	2	3
DragonPirateCard	class	4	7	1	0	10	1
Animal	interface	0	0	1	0	0	0
GameBoard	class	9	10	1	0	9	10
DragonCardPanel	class	3	3	5	0	6	0

Figure 1.2.3.1: The CK metric of Jia Xuan's code.

- Aesthetics of the User Interface (User Engagement)
  The user interface's aesthetics have not been fully implemented. The UI failed to show the game board. The actions which can be executed by the player are not implemented.

1.3 Conclusions of Teach-Based Software Prototypes Assessment

1.3.1 Conclusion of CK Metrics of Software Prototypes

The following Table 1.3.1 to Table 1.3.6 summarises the findings using CK Metrics for each member's software prototypes.

SUMMARY STATISTICS FOR THE WMC METRIC

Member	Metric	Median	Max	Min
Rui En	WMC	6	16	1
Ming Hui	WMC	4	23	1
Jia Xuan	WMC	3	19	0

Table 1.3.1: Summary statistics for the WMC metric

SUMMARY STATISTICS FOR THE DIT METRIC

Member	Metric	Median	Max	Min
Rui En	DIT	1	6	1
Ming Hui	DIT	2	8	1
Jia Xuan	DIT	1	6	1

Table 1.3.2: Summary statistics for the DIT metric

SUMMARY STATISTICS FOR THE NOC METRIC

Member	Metric	Median	Max	Min
Rui En	NOC	0	0	0
Ming Hui	NOC	0	5	0
Jia Xuan	NOC	0	0	0

Table 1.3.3: Summary statistics for the NOC metric

SUMMARY STATISTICS FOR THE CBO METRIC

Member	Metric	Median	Max	Min
Rui En	CBO	2	9	0
Ming Hui	CBO	2	6	0
Jia Xuan	CBO	2	7	0

Table 1.3.4: Summary statistics for the CBO metric

SUMMARY STATISTICS FOR THE RFC METRIC

Member	Metric	Median	Max	Min
Rui En	RFC	6	25	0
Ming Hui	RFC	5	25	0
Jia Xuan	RFC	6	17	0

Table 1.3.5: Summary statistics for the RFC metric

SUMMARY STATISTICS FOR THE LCOM METRIC

Member	Metric	Median	Max	Min
Rui En	LCOM	2	47	0
Ming Hui	LCOM	0	87	0
Jia Xuan	LCOM	1	25	0

Table 1.3.6: Summary statistics for the LCOM metric

1.3.2 Creation of Tech-Based Software Prototype for Sprint 3

After analysing software prototypes developed by Koe Rui En, Tay Ming Hui, and Wong Jia Xuan respectively, using the quality model of ISO/IEC 25010 and Chidamber and Kemerer CK Metrics Suite, we have decided to stick with Tay Ming Hui's product prototype for the further development in Sprint 3 without integrating any ideas/elements of each of the tech-based prototypes from Sprint 2 from the other team members. The considerations for this course of action are:

1. Design Patterns and Architecture:
    a. The Bridge Pattern was used to implement ChitCards, Volcano Cards, Chit Board, and Volcano CardBoard. This pattern enhances flexibility and organisation of the code by abstracting the implementation of an entity from its abstraction.
    b. A Facade Pattern was used to implement a simplified interface that can control setup and launching of the game and enhance its maintainability.
    c. Structure Pattern/Manager Classes - This pattern was used to eliminate redundancy and to promote better maintainability.

2. CK Metrics:
    a. Low LCOM - Guarantees better encapsulation and modularity; hence, the design is higher in quality.
    b. Low RFC - Makes testing and maintenance easier because the complexity of interactions is reduced.
    c. DIT - Good balance of potential reuse without complexity in the structure being too high.
    d. CBO - Lower CBO value ensures better modularity and ease of maintainability.

3. Functional Suitability:
    a. Complete Implementation: Implementation was successful in generating the initial board setup with randomization for positioning of dragon cards and successfully accomplishing the flipping of dragon cards.
    b. User Interface: The user interface is highly representative of the physical game board; it is clear and intuitive.
    c. Maintainability:
        i. Modular Architecture: The use of Bridge and Facade design patterns gives the structure better modularity, which in turn helps to make the code more maintainable and extensible.
        ii. Code Clarity: Clear separation of concerns and adhering to the Single Responsibility Principle will make code more readable and maintainable.

4. SOLID Principles
    a. Single Responsibility Principle(SRP): Each class in her design has a single responsibility, making the system easier to understand and maintain.
    b. Liskov Substitution Principle (LSP): All derived classes in her design can be substituted for their base classes without affecting the correctness of the program. This ensures that the system remains robust and reliable.
    c. Open/Close Principle (OCP): Her design allows for easy extension without modifying existing code. This is achieved through the use of abstract classes, allowing new functionalities to be added with minimal changes.

Based on the considerations and summaries of CK Metrics that we had mentioned above, Ming Hui's prototype stands out due to its strict adherence to SOLID principles, high cohesion, and low coupling. These qualities ensure that her design is not only more robust and reliable but also easier to maintain and extend. In comparison, the other prototypes lack these critical design qualities, making them less optimal for long-term development and maintenance.

b. Future Improvements

Although the prototype built by Tay Ming Hui is robust, there are certain risks and areas of improvement that need to be addressed in Sprint 3 so that the final product meets all the quality standards and the expectations of the users. Some new ideas are discussed and may be incorporated in the prototype in order to improve it for Sprint 3.

1. Scalability:
    a. The game should be scalable to support increased complexity; for example, more players, more features, etc. should not have a significant impact on the performance of the game.
    b. Future expansion Plan for additional features or new game modes that may be included in future sprints, ensuring that the architecture will support such expansions without major refactoring.

2. Code Quality and Refactoring:
    a. Code complexity: Although the current design is modular, further work will be required to minimise any residual code complexity. Refactor any class that has high complexity or coupling.
    b. Error handling: Implement strong error-handling mechanisms to ensure graceful handling of any unexpected situation or user input.

3. User Interface Enhancements:
    a. Visual and audio feedback: Add some more visual and audio feedback such as background music and label the current player's name and turn to the user interface in order to make the game more engaging, interesting, and more fun to its users.

4. Performance Optimization:
    a. Response time: Ensure that the response time for the game actions and graphics to load is at the specified performance criteria. Optimise the code and resources to lower latency.
    b. Resource management: Develop good resource management to ensure that the game runs smoothly on the platform with no excess CPU or memory usage.

5. Documentation and Code Comments:
    a. Proper Documentation: Document well such that the architecture, design patterns, and key functionalities are described properly.
    b. Inline Comments: Use comments to explain complex logic within the code, which is useful for further developers.

6. Reliability:
    a. Reliability Testing: Stringent testing so that all reliability issues will be tracked and resolved, and the game will sustain its work under any conditions with the same quality.

In this manner, the final prototype is going to be robust, maintainable, scalable, user-friendly, and performant. This prototype by Tay Ming Hui will be at a good level, so with these additions, it will be immensely improved to fit the project's needs and give the best gaming experience.

2. Object-Oriented Design

2.1 Class-Responsibility-Collaboration (CRC)

The section outlines the six main classes of the consolidated design using CRC cards. The purpose of these selected six main classes is also identified and well-justified in each subsection of Section 2.1.

2.1.1 AnimalFactory class

AnimalFactory	
Manufacture instances of Animal subclasses, including baby dragon, bat, spider, salamander, and pirate dragon, corresponding to different game components such as volcano cards, chit cards, and cave cards	Bat BabyDragon PirateDragon Spider Salamander

Table 2.1.2: CRC card of AnimalFactory class

Purpose of AnimalFactory class:

The AnimalFactory class serves as a centralised factory for creating various types of Animals utilised in the Fiery Dragons game components, such as dragon cards, volcano cards, and cave cards. Its introduction helps to avoid tight coupling between the creator (AnimalFactory) and all concrete products (Bat, Spider, BabyDragon, Salamander, and PirateDragon) derived from the abstract product (Animal). By encapsulating the creation logic within factory methods, AnimalFactory simplifies the process of creating and managing animals for different game components. This approach enhances code reusability and maintainability, adhering to the Single Responsibility Principle (SRP) by focusing solely on animal creation. Moreover, it aligns with the Open/Closed Principle (OCP) by enabling the introduction of new types without disrupting existing code. As various components consisting of animals may be introduced in future game implementations, the AnimalFactory class can be seamlessly utilised across each, promoting efficiency and consistency.

2.1.2 Game class

Game	
Initialises 2 to 4 players for the Fiery Dragons' game	Token
Sets up game board for the Fiery Dragons game	BoardSetup ChitCardManager VolcanoCardManager CaveCardManager
Executes the main game loop of Fiery Dragons' game	Token ChitCardManager CaveCardManager VolcanoCardManager BoardSetup WinningPopupPanel
Checks the animal on the flipped chit card same as the animal on the current volcano card that the dragon token occupies	ChitCardManager Token
Calculates the number of steps that the token needs to be moved	Token
Covers all the uncovered chit cards after the current player's turn is over	ChitCardManager
Processes player turns	Token ChitCardManager CaveCardManager VolcanoCardManager
Moves the token forward or backward based on the calculated number of steps.	Token JPanel
Checks the winner of game	Token
Displays name of players	PlayerInfoDialog

Table 2.1.1: CRC card of Game class

Purpose of the Game class:

The purpose of the Game class is to represent the main logic and user interface for the Fiery Dragons game. It encapsulates the main logic and user interface, providing players with an immersive gaming experience. This includes initialising the game state and setting up essential game elements such as volcano cards, chit cards, and cave cards. Within the main game loop, it manages player turns, movements, and interactions, ensuring smooth gameplay progression. Moreover, the Game class determines the winner of the game, bringing closure to the gaming session. Hence, this design decision aligns with the Single Responsibility Principle (SRP) and Open/Closed Principle (OCP), where the Game class manages the whole game system, and it can also be extended if any new features are introduced in future game implementation.

2.1.3 HomePage class

HomePage	
Sets up the homepage of the Fiery Dragon to be rendered on the screen	JPanel JLabel ImageIcon
Renders game rule on the home page	InfoButton
Plays music on the background of the game	MusicButton
Displays text fields for players to enter their information	PlayerInfoDialog

Table 2.1.3: CRC card of HomePage class

Purpose of HomePage class:

The HomePage class serves as the initial interface displayed to players when they start the game. It provides a welcoming screen with various interactive elements, allowing players to navigate into the game or access additional features. In the home page, it contains a start button for players to start the game. A pop-up will appear for players to enter their information and send this information back to the game system. This design adheres to the Single Responsibility Principle (SRP) as it focuses on providing the home page interface and initiating the game. It does not handle game logic or other unrelated functionalities. The HomePage class also separates the interface layer (GUI) from the logic of the game, ensuring a clear distinction between user interface components and underlying game mechanics.

2.1.4 ChitCardManager class

ChitCardManager	
Configures chit cards for rendering on the game board	ChitBoard JLabel BoardSetup
Configures interaction with chit cards	ChitBoard MouseListener JLabel
Provides the type of animal on the flipped ChitCard	Animal
Provides the movement shown on the flipped chit card	
Provides the flipped chit card	ChitCard
Initialises a animal and movement obtained from the flipped chit card	Animal
Covers all the uncovered chit cards	ChitCard
Clears the animal on the flipped chit card	Animal
Clears all the chit cards and the clicked chit cards	ChitCard JLabel

Table 2.1.4: CRC card of ChitCardManager class

Purpose of ChitCardManager class:

The ChitCardManager class is designed to handle the initialisation of chit cards on the game board and their interaction. It encapsulates functionalities related to configuring chit cards on the game board, managing mouse interactions with them, and handling the state of flipped chit cards after a player's turn. This consolidation of related functionalities within a single unit enhances maintainability and readability. By focusing solely on managing the chit cards, this design decision adheres to the Single Responsibility Principle (SRP). Since it manages the interaction with chit cards, the class allows for the extension of further functionalities related to interaction without modifying the current implementation. This promotes the Open/Closed Principle (OCP) and reduces the responsibility of the Game class to handle all interactions with chit cards. This decision making prevents the Game class from becoming a GOD class, which violates the Single Responsibility Principle (SRP) and may make the Game class hard to maintain, and also increase the complexity of Game class.

2.1.5 CaveCardManager class

CaveCardManager	
Initialises animals on cave cards	AnimalFactory
Configures cave cards for rendering on the game board	CaveCard BoardSetup
Shuffles all cave cards	CaveCard
Configures the position of all dragon tokens with their respective cave card for rendering on the game board	Token BoardSetup CaveCard
Provides the dragon tokens corresponding to their respective cave cards.	Token
Provides the position of cave where the dragon token is placed	CaveCard Token

Table 2.1.5: CRC card of CaveCardManager class

Purpose of CaveCardManager class:

The CaveCardManager class is designed to handle several key functionalities related to cave cards in the game. It manages the initialisation of animals on cave cards and configures them to be rendered on the game board. Additionally, players have the option to place cave cards randomly, and the class facilitates this by implementing a shuffle mechanism for cave cards. Furthermore, as cave cards serve as starting locations for all dragon tokens, the class is responsible for setting the position of each token on its respective cave card. This means that it encapsulates tasks related to configuring both cave cards and dragon tokens on the game board, promoting maintainability and flexibility through consolidation of related functionalities. This design adheres to the Single Responsibility Principle (SRP) by focusing solely on managing cave cards. This specialisation allows for easy extension of functionalities specific to cave cards without altering existing implementations, aligning with the Open/Closed Principle (OCP). By reducing the responsibility of the Game class to handle cave card configurations, the CaveCardManager prevents it from becoming overly complex and violating the SRP, thus facilitating easier maintenance of the game code base.

2.1.6 VolcanoCardManager class

VolcanoCardManager	
Configures volcano cards for rendering on the game board	VolcanoCardBoard BoardSetup
Initialises the path of the player's dragon token taken on the volcano cards	GameComponent VolcanoCard

Table 2.1.6: CRC card of VolcanoCardManager class

Purpose of VolcanoCardManager class:

The VolcanoCardManager class is introduced to handle several key functionalities related to volcano cards in the game. It configures volcano cards and arranges them to be rendered on the game board. Since each player's dragon token follows its own path on the volcano cards, it also handles the initialisation of these paths. This consolidation of related functionalities within a single unit enhances maintainability and readability. By focusing solely on managing the volcano cards, this design adheres to the Single Responsibility Principle (SRP). Additionally, the class allows for the extension of further functionalities without modifying the current implementation if any new features related to volcano cards are introduced. This promotes the Open/Closed Principle (OCP) and reduces the responsibility of the Game class in setting up the volcano cards on the game board. This decision prevents the Game class from becoming a GOD class, which violates the Single Responsibility Principle (SRP) and may make the Game class hard to maintain and increase its complexity.

2.1.7 Discard Alternative Distribution of Responsibilities and Justifications

In the development of our Fiery Dragons game, we explored various alternative distribution of responsibilities to ensure the most effective and maintainable architecture. Below, we present the significant alternative and discuss the reasons for its rejection.

Alternative Distribution of Responsibilities for Class1: Game Class

Game	
Initialises 2 to 4 players for the Fiery Dragons' game	Token
Sets up game board for the Fiery Dragons game	BoardSetup ChitCardManager VolcanoCardManager CaveCardManager
Executes the main game loop of Fiery Dragons' game	Token ChitCardManager CaveCardManager VolcanoCardManager BoardSetup WinningPopupPanel
Checks the animal on the flipped chit card same as the animal on the current volcano card that the dragon token occupies	ChitCardManager Token
Calculates the number of steps that the token needs to be moved	Token
Covers all the uncovered chit cards after the current player's turn is over	ChitCardManager
Processes player turns	Token ChitCardManager CaveCardManager VolcanoCardManager
Moves the token forward or backward based on the calculated number of steps.	Token JPanel
Checks the winner of game	Token
Configures volcano cards for rendering on the game board	VolcanoCard BoardSetup

Configures cave cards for rendering on the game board	CaveCard BoardSetup
Configures chit cards for rendering on the game board	ChitBoard JLabel BoardSetup
Initialises the path of the player's dragon token taken on the volcano cards	GameComponent VolcanoCard
Provides the flipped chit card	ChitCard
Initialises a animal and movement obtained from the flipped chit card	Animal
Covers all the uncovered chit cards	ChitCard
Clears the animal on the flipped chit card	Animal
Shuffles all volcano, cave and chit cards	CaveCard ChitCard VolcanoCard
Configures the position of all dragon tokens with their respective cave card for rendering on the game board	Token BoardSetup CaveCard

Table 2.1.5: CRC card of Alternative Distribution of Responsibilities of Game class

Justification of Alternative Distribution of Responsibilities for Game Class:

Table 2.1.5 above shows the alternative distribution of responsibilities of Game Class. We know that the purpose of Game class is to control all the main flow of the game, including initialisation configuration, and interaction of all game components, into a single Game class. Before we introduce any manager classes such as VolcanoCardManager, CaveCardManager and ChitCardManager, to handle these responsibilities, the Game class needs to directly handle all aspects of the game, from the player management to component interactions. Hence, there are few significant reasons we decided to discard this alternative distribution of Game class's responsibilities as discussed below.

Reasons of Discard:

- Violation of Single Responsibility Principle (SRP)
  The Game class needs to handle multiple responsibilities simultaneously, potentially leading it to become a "GOD class" that handles unnecessary responsibilities which could be managed by other classes. This decision not only separates all the concerns of the Game class but also makes the class harder to maintain and extend.

- Increase Complexity
  Since the Game class manages all game logic, it would become extremely complex. This complexity makes the Game class difficult to understand, maintain, and extend, reducing its overall robustness.

- Scalability Issues
  As the game grows with more features, the Game class approach would not scale well, potentially leading to an increase in faults in the implemented game features and higher maintenance overhead.

2.2 UML Class Diagram of Fiery Dragons

This section will include the revised version of UML Class Diagram for Fiery Dragon's game of Sprint 3 as shown in Figure 2.2.1 below.

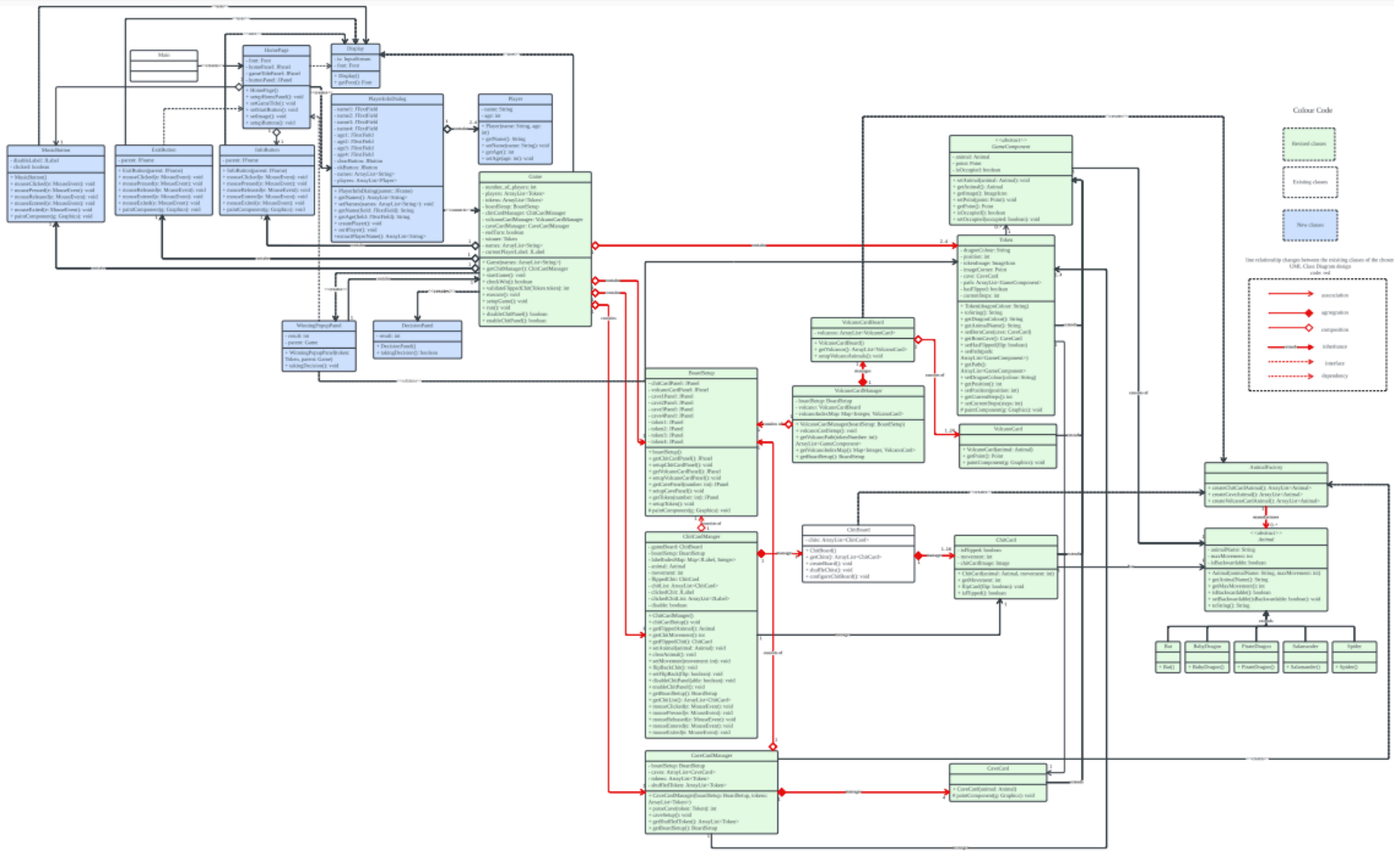


Figure 2.2.1: The revised version of UML Class Diagram for Fiery Dragon's game of Sprint 3

** Note:
If the UML class diagram is blurry, please refer to attachment at the end of documents.

3. Executable Deliverable

This section will provide detailed instructions on how to build and run the executable for the software prototype from the source code. Additionally, it will specify the target platforms on which the software prototype can be run, as well as the settings for the modules.

3.1 Instructions for Running Executable Files of Software Prototype

**** Instructions before running the executable .jar file:**

Target Platform: Windows & MacOs

Required SDK: openjdk-22 (Oracle OpenJDK version 22.0.1) or 22 (Oracle OpenJDK version 22.0.1)

Required Environment: JDK-22 and Java™ Platform SE binary

Required JDK Development Kit:

- For Windows: x64 Installer
- For MacOS: ARM64 DMG Installer or x64 DMG Installer

** Notes:

For MacOs, after clicking on the .jar file, the user must click “Open Anyway”.

Settings -> Privacy & Security -> “Open Anyway”

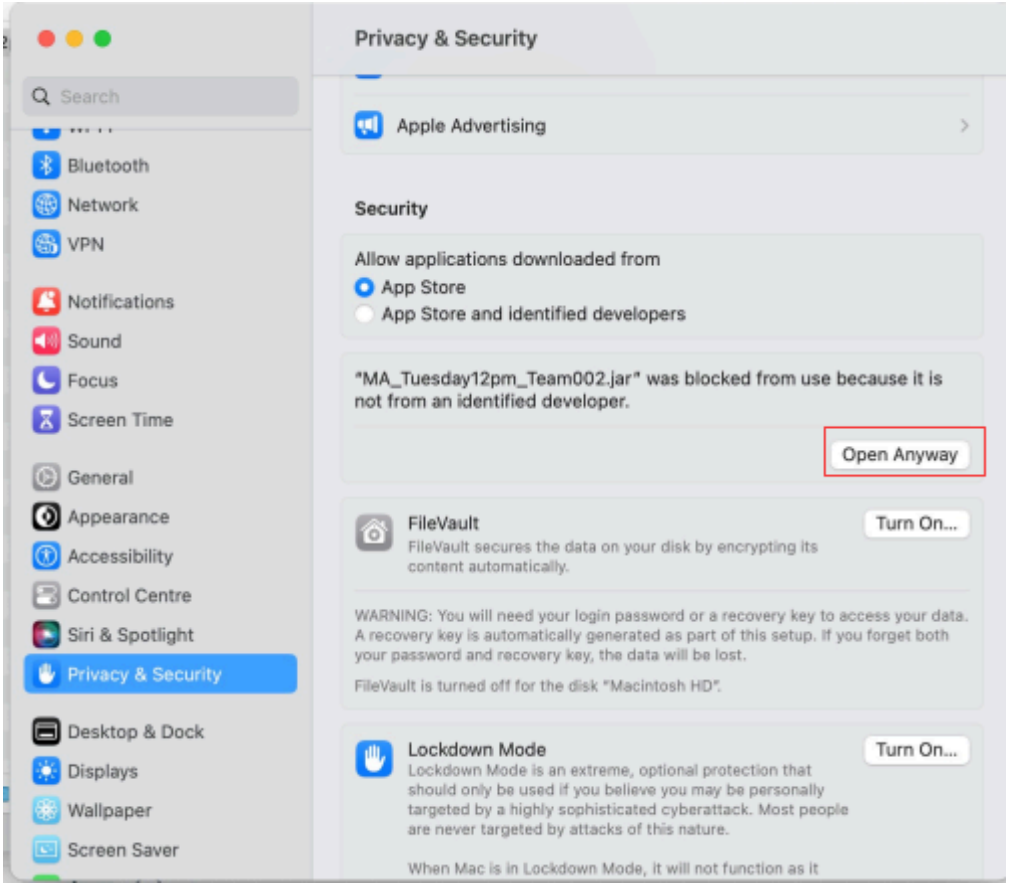


Figure 3.1.1 The Settings for macOS user

How to run the application:

1. Download the MA_Tuesday12pm_Team002_jar zip folder.

2. Unzip the folder and locate the MA_Tuesday12pm_Team002.jar file.

3. There are two ways to run the application:
    a. via the command prompt of the Windows operating system.
    b. using Java™ Platform SE binary with JDK-22, where the product is **x64 Installer** for **Windows**, while the product is **ARM64 DMG Installer** or **x64 DMG Installer** for **MacOS**.

4. For command prompt, enter in “java -jar MA_Tuesday12pm_Team002.jar” and run the file.

5. To run on Java Platform SE binary, ensure that JDK-22 and Java Platform SE binary are downloaded before double-clicking the jar file. If **double-clicking does not open** the game, right-click and choose the option "**Open with > Java™ Platform SE binary**," or click "Choose another app" and select **"Java™ Platform SE binary**" if it does not appear in the recommended options as shown in Figure 3.1.2.

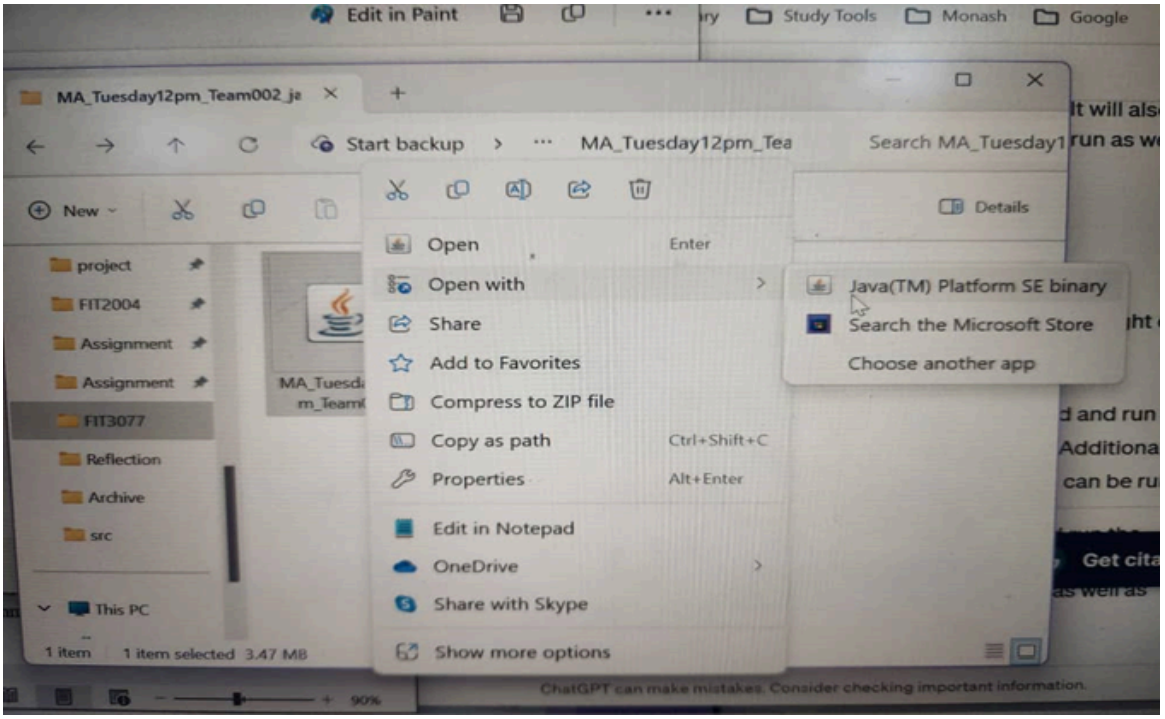


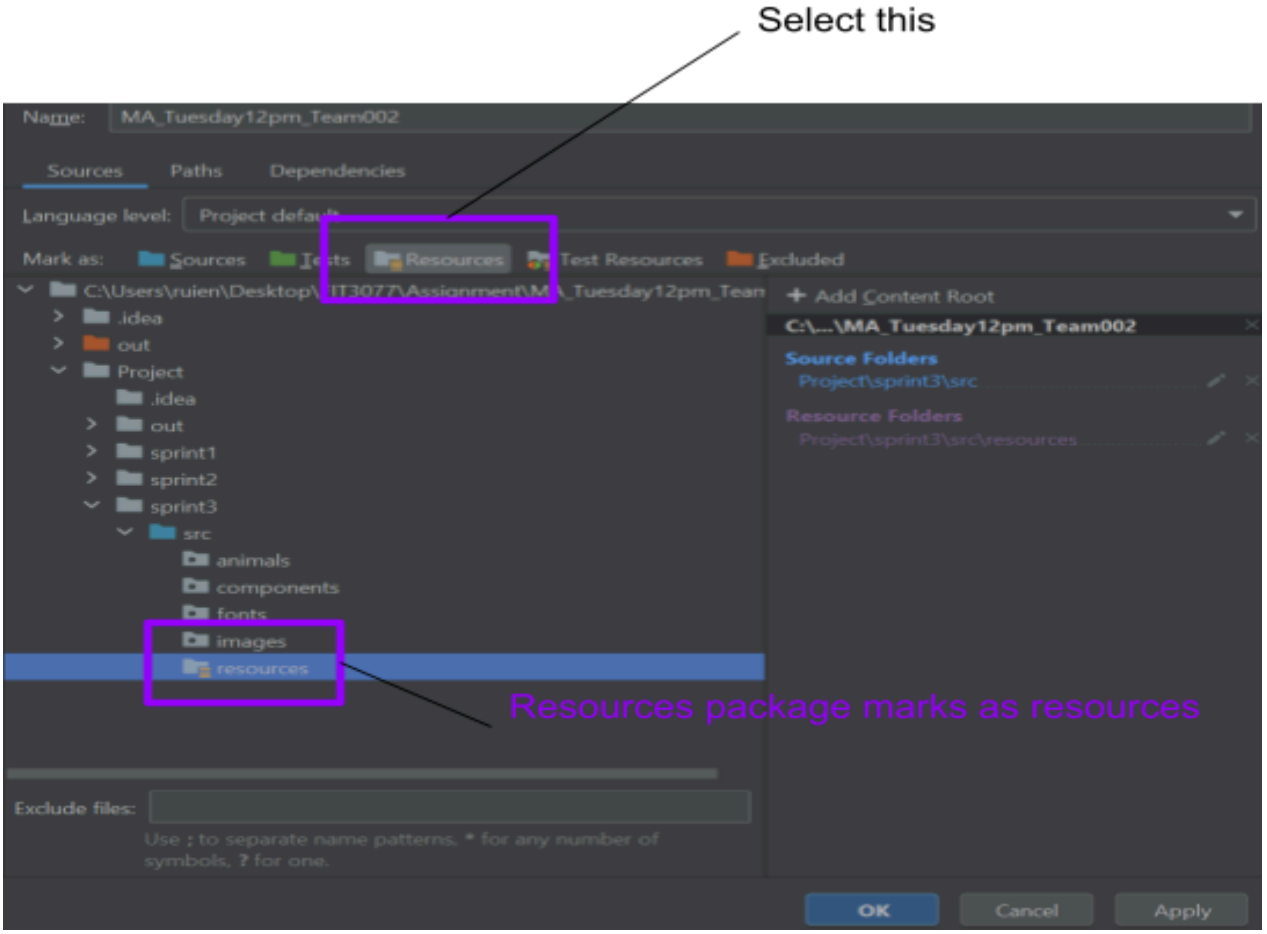
Figure 3.1.2: Instruction on using (b) to run the Fiery Dragons application

3.2 Instructions for Building Executable File of Software Prototype

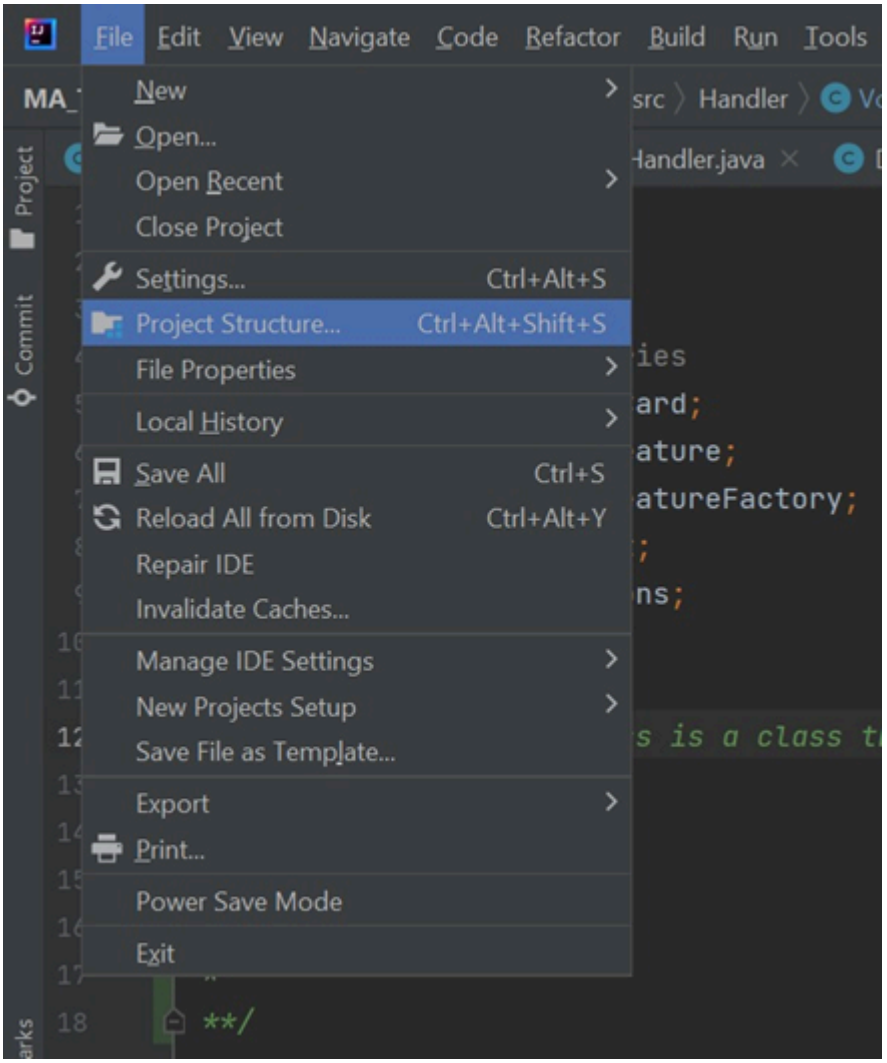
This section will demonstrate how our team built the executable file of our software prototype in IntelliJ, as shown in the attached figures, along with descriptions provided below.

How to build the executable jar file:

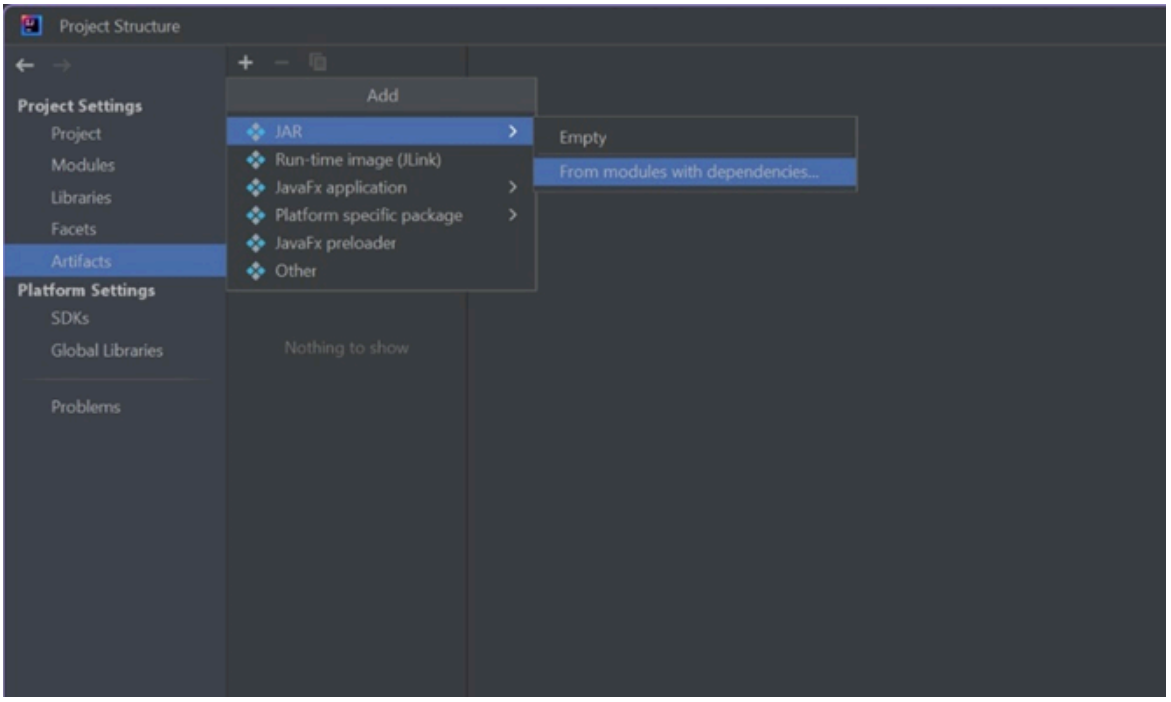
Before building the executable JAR file, we will create a package called 'resources' and mark this package as resources within the module in the project structure.



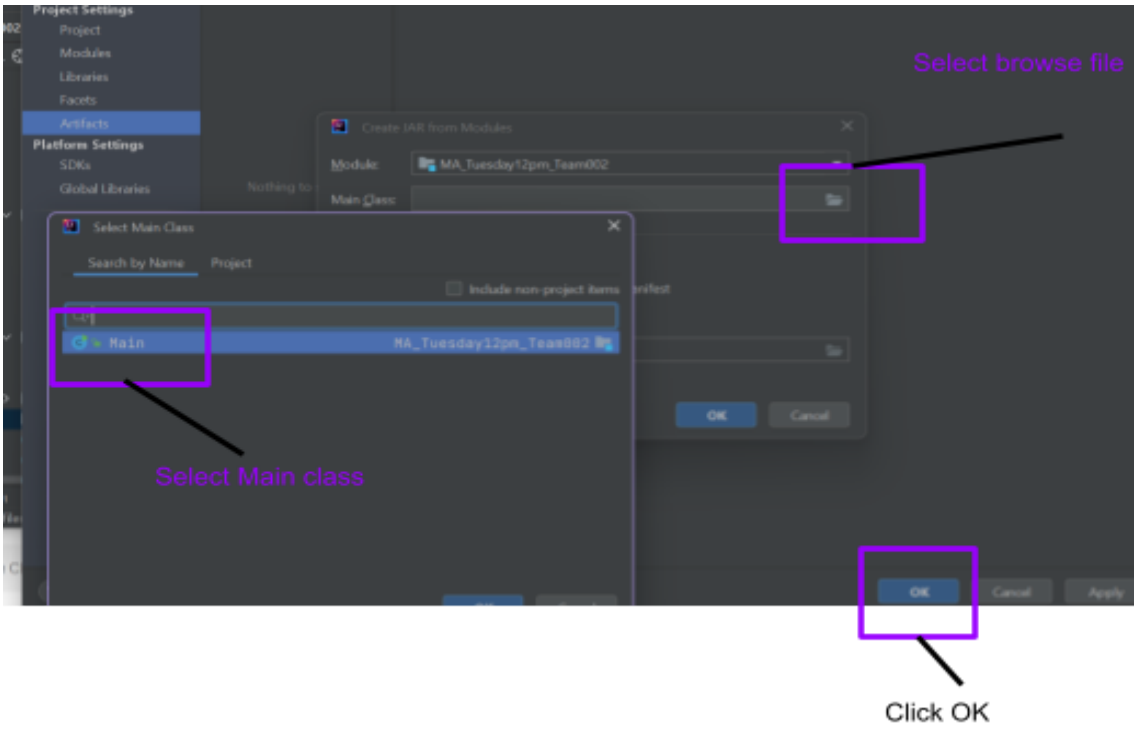
1. In the main menu, go to File | Project Structure and select Artifacts.



2. Click the Add button, point to JAR and select From modules with dependencies.

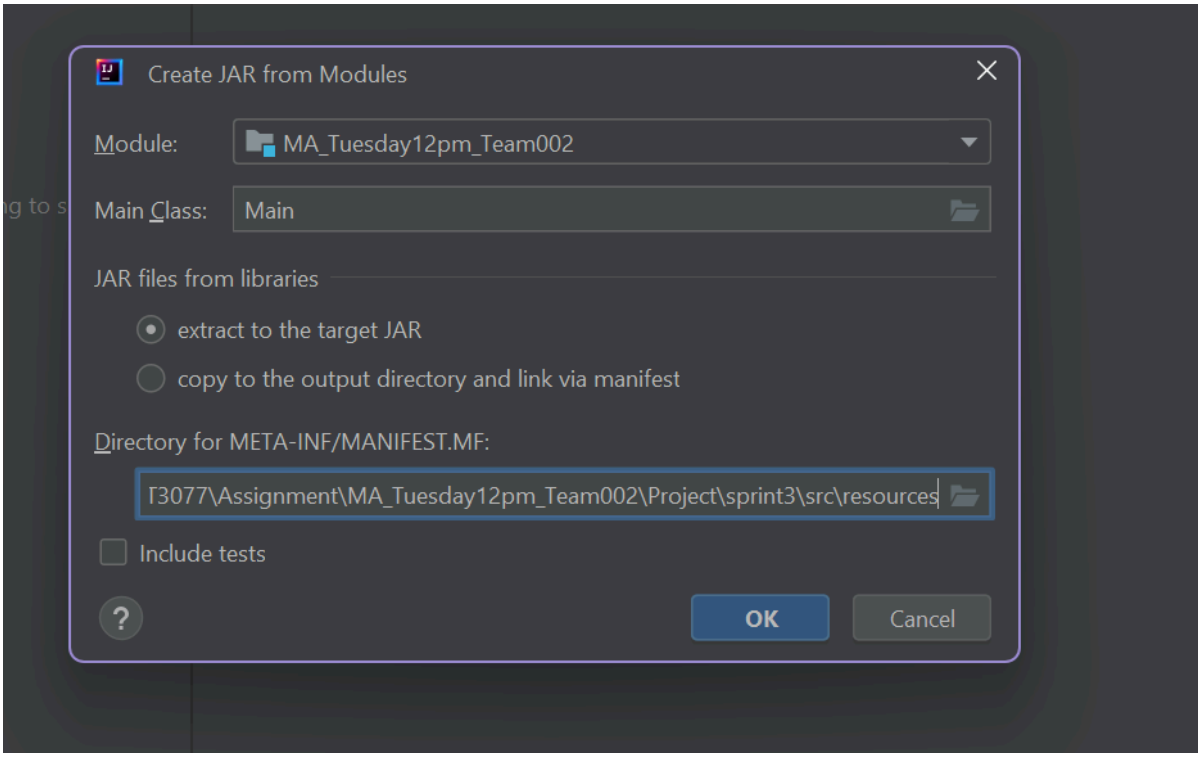


3. To the right of the Main Class field, click the Browse button and select Main class (main) in the dialog that opens.

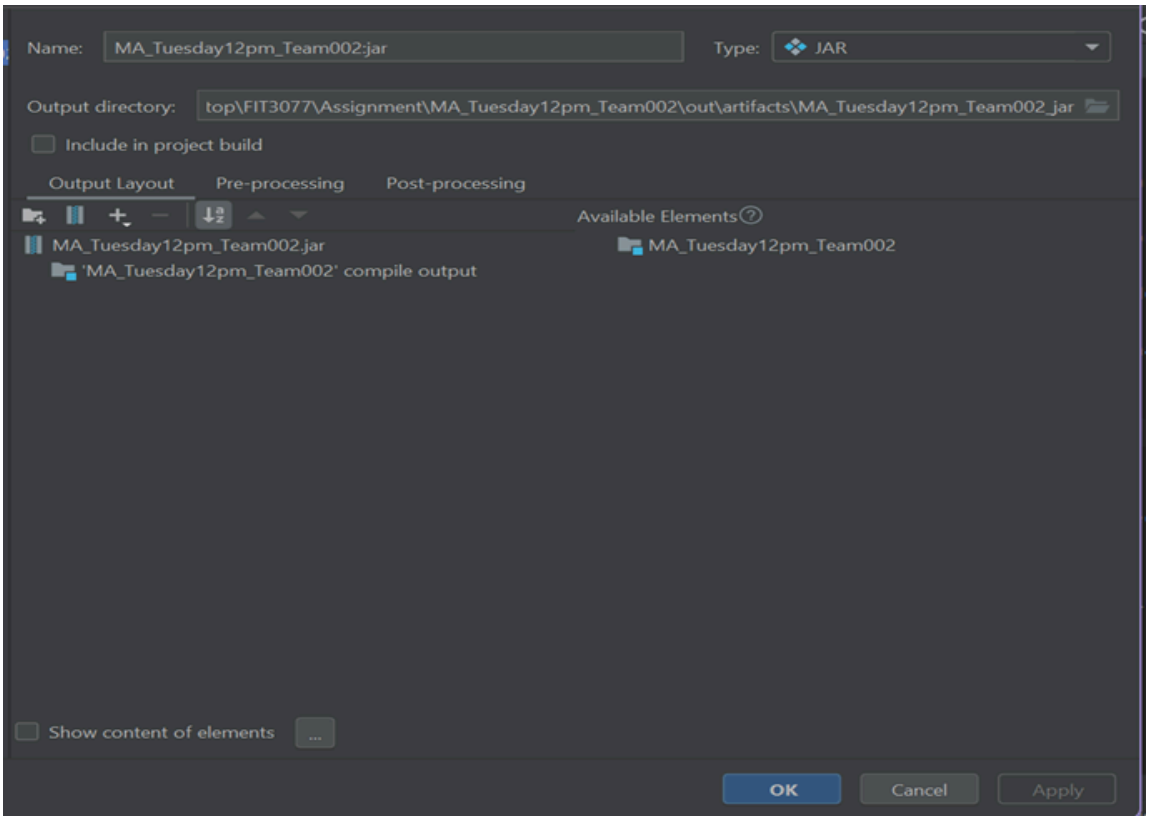


\

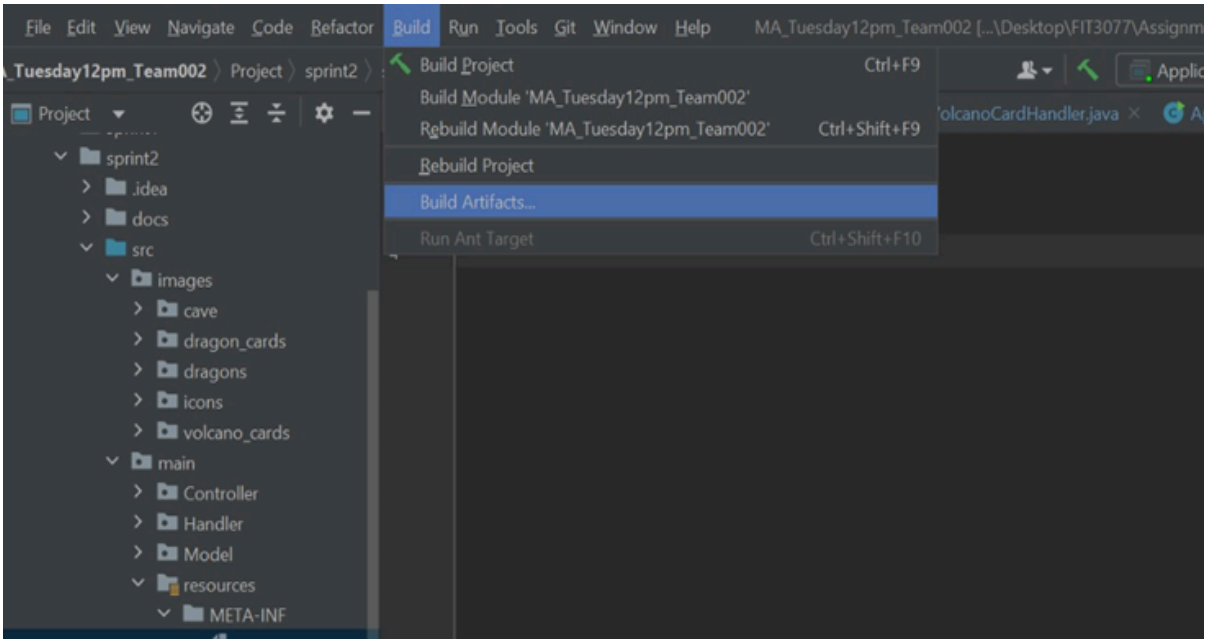
4. Change the directory for META-INF/MANIFEST.MF to src/resources in the Create JAR from Modules.



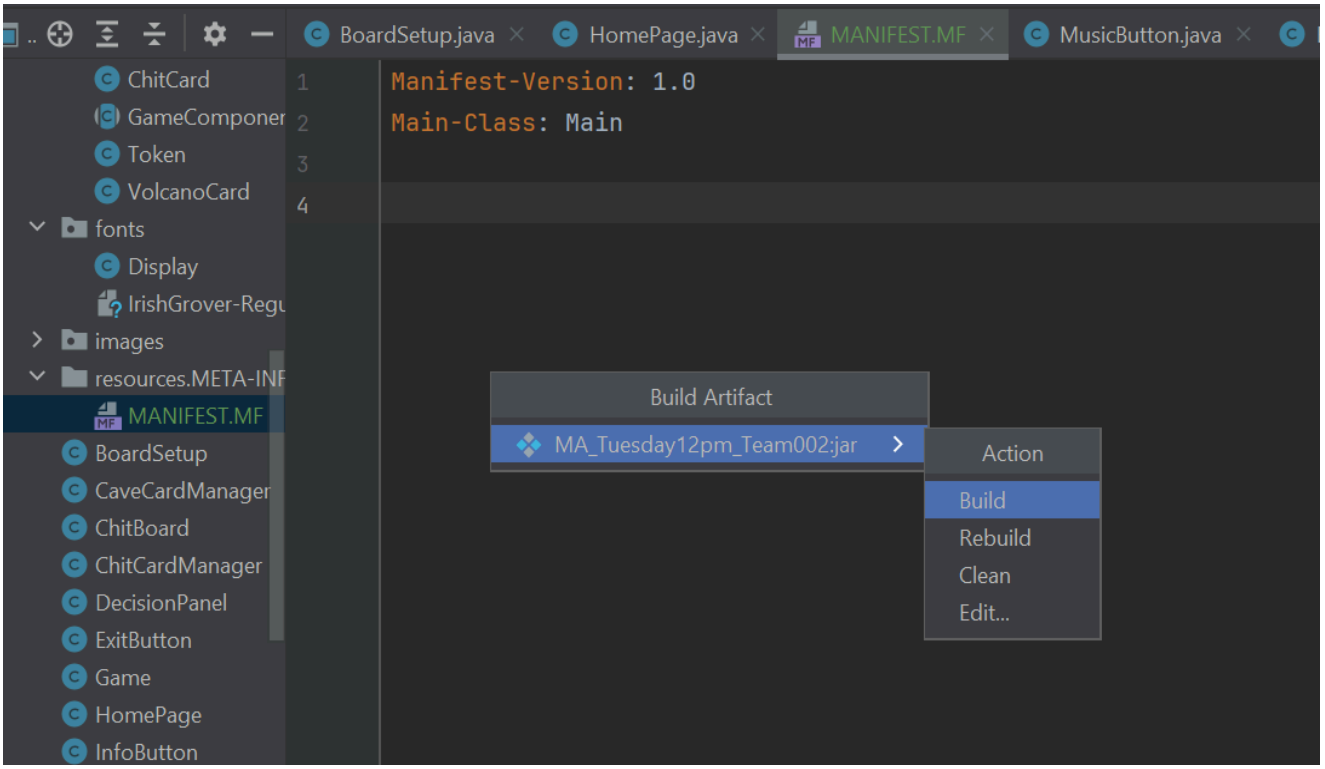
5. In the Artifacts section, select Apply and close the dialog.



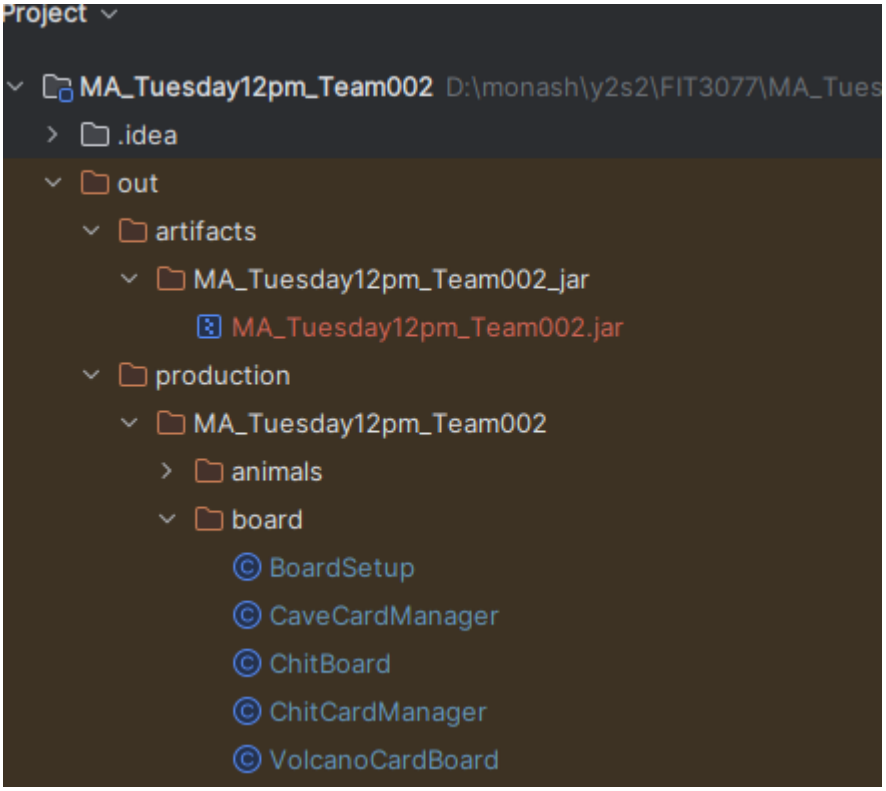
6. Then, in the main menu, go to Build | Build Artifacts.



7. Point to MA_Tuesday12pm_Team002:jar and select Build.



8. Now, the JAR file is in the out/artifacts folder.



4. Appendices

Appendix 1: Important Links

This section will include all the important links, GitLab repository, contribution logs, wiki section in GitLab repository and Google Drive that facilitate our team's collaboration more effectively and efficiently.

Google Drive Link:

🗀 FIT3077-Submission

GitLab Repository Link:

https://git.infotech.monash.edu/FIT3077/fit3077-s1-2024/MA_Tuesday

Contribution Logs:

FIT3077 - MA_Tuesday12pm_Team002 -Contribution Logs

Wiki Section in GitLab Repository Link:

https://git.infotech.monash.edu/FIT3077/fit3077-s1-2024/MA_Tuesday12pm_Team002/-/wikis/home

Appendix 2: Git Commit History - “Contributor Analytics”

This section will include a screenshot of recent "Contributor Analytics," visualising the commit history of all team members to ensure that each team member contributes to Sprint 3.

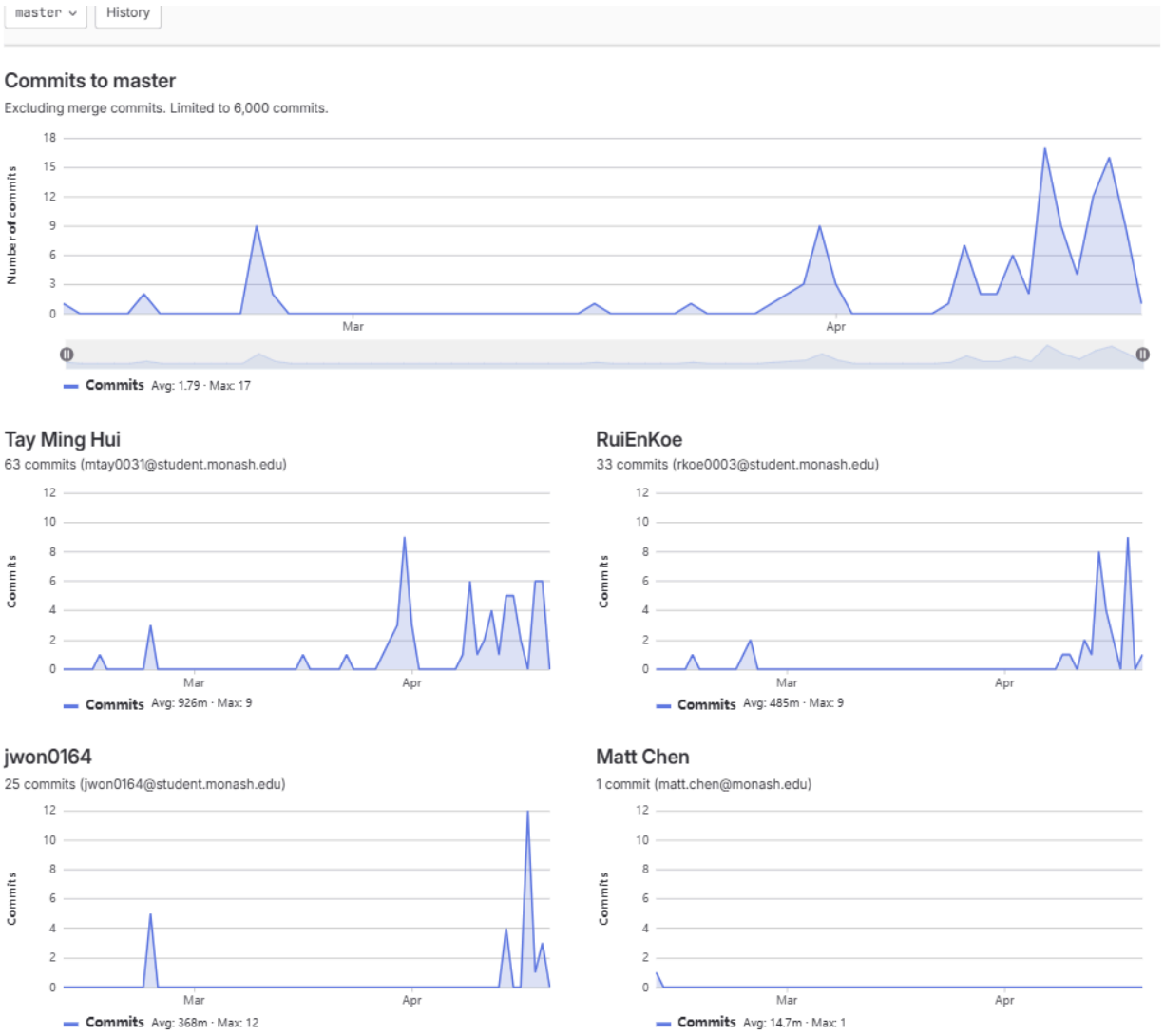


Figure 1: A screenshot of the Luminary Team’s “Contributor Analytics”

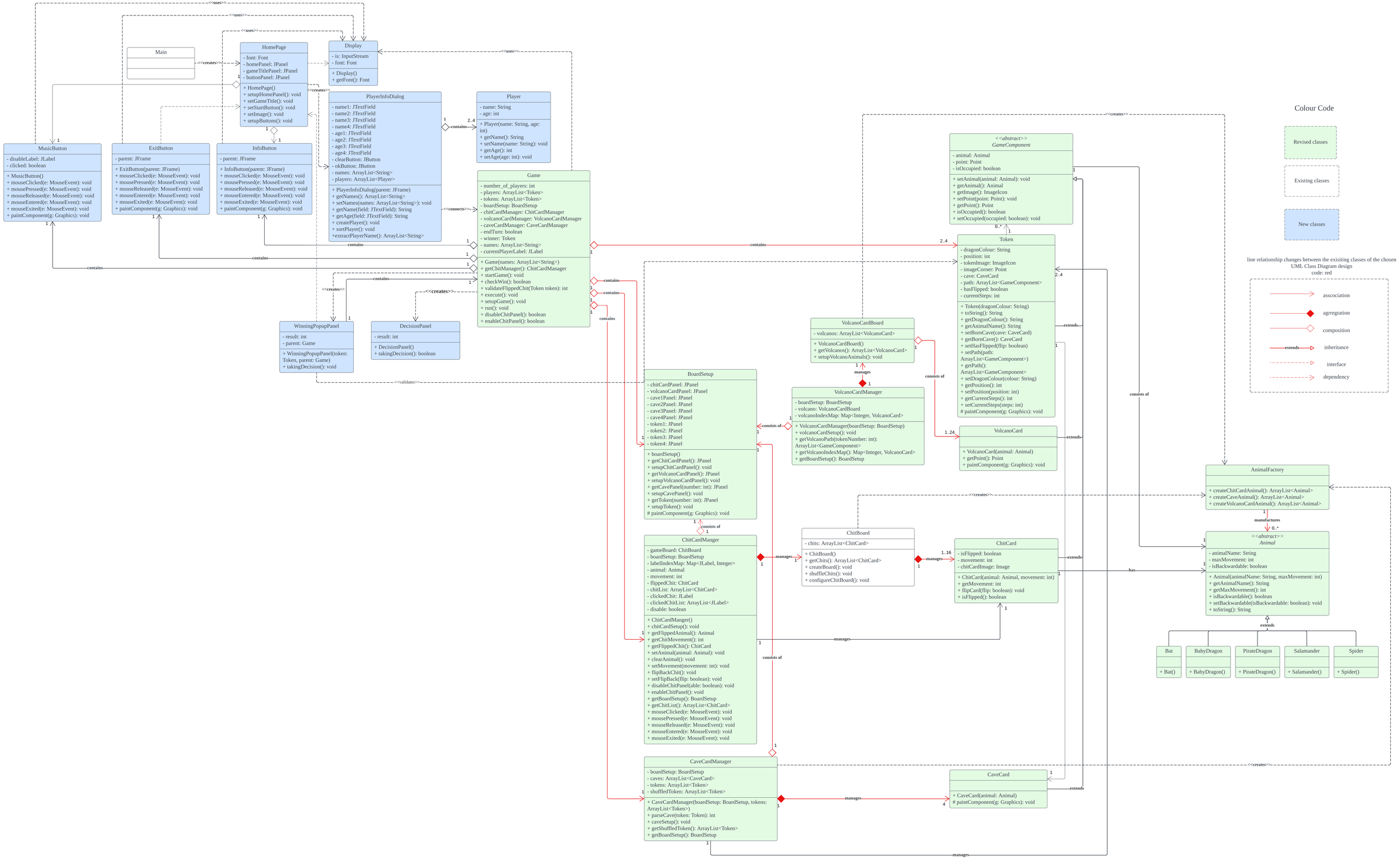
Appendix 3: References

[1] M. Aniche. “mauricioaniche/ck,” GitHub. Accessed: May. 13, 2024. [Online]. Available: https://github.com/mauricioaniche/ck/tree/master

[2] “CK metrics (Chidamber & Kemerer Suite of Metrics).” YouTube. Accessed: May. 13, 2024. [Online]. Available: https://www.youtube.com/watch?v=g_uTDj102Cg

Appendix 4: Acknowledgement

I acknowledge the use of ChatGPT. The prompts used include “help me to refine my writing”, etc. The output from these prompts was used to help me to learn Java Swing better.


Main
HomePage
- font: Font
- homePanel: JPanel
- gameTitlePanel: JPanel
- buttonPanel: JPanel
+ HomePage()
+ setupHomePanel(): void
+ setGameTitle(): void
+ setStartButton(): void
+ setImage(): void
+ setupButtons(): void
Display
- is: InputStream
- font: Font
+ Display()
+ getFont(): Font
MusicButton
- disableLabel: JLabel
- clicked: boolean
+ MusicButton()
+ mouseClicked(e: MouseEvent): void
+ mousePressed(e: MouseEvent): void
+ mouseReleased(e: MouseEvent): void
+ mouseEntered(e: MouseEvent): void
+ mouseExited(e: MouseEvent): void
+ paintComponent(g: Graphics): void
ExitButton
- parent: JFrame
+ ExitButton(parent: JFrame)
+ mouseClicked(e: MouseEvent): void
+ mousePressed(e: MouseEvent): void
+ mouseReleased(e: MouseEvent): void
+ mouseEntered(e: MouseEvent): void
+ mouseExited(e: MouseEvent): void
+ paintComponent(g: Graphics): void
InfoButton
- parent: JFrame
+ InfoButton(parent: JFrame)
+ mouseClicked(e: MouseEvent): void
+ mousePressed(e: MouseEvent): void
+ mouseReleased(e: MouseEvent): void
+ mouseEntered(e: MouseEvent): void
+ mouseExited(e: MouseEvent): void
+ paintComponent(g: Graphics): void
PlayerInfoDialog
- name1: JTextField
- name2: JTextField
- name3: JTextField
- name4: JTextField
- age1: JTextField
- age2: JTextField
- age3: JTextField
- age4: JTextField
- clearButton: JButton
- okButton: JButton
- names: ArrayList<String>
- players: ArrayList<Player>
+ PlayerInfoDialog(parent: JFrame)
+ getNames(): ArrayList<String>
+ setNames(names: ArrayList<String>): void
+ getName(field: JTextField): String
+ getAge(field: JTextField): String
+ createPlayer(): void
+ sortPlayer(): void
+extractPlayerName(): ArrayList<String>
Player
- name: String
- age: int
+ Player(name: String, age: int)
+ getName(): String
+ setName(name: String): void
+ getAge(): int
+ setAge(age: int): void
Game
- number_of_players: int
- players: ArrayList<Token>
- tokens: ArrayList<Token>
- boardSetup: BoardSetup
- chitCardManager: ChitCardManager
- volcanoCardManager: VolcanoCardManager
- caveCardManager: CaveCardManager
- endTurn: boolean
- winner: Token
- names: ArrayList<String>
- currentPlayerLabel: JLabel
+ Game(names: ArrayList<String>)
+ getChitManager(): ChitCardManager
+ startGame(): void
+ checkWin(): boolean
+ validateFlippedChit(Token token): int
+ execute(): void
+ setupGame(): void
+ run(): void
+ disableChitPanel(): boolean
+ enableChitPanel(): boolean
WinningPopupPanel
- result: int
- parent: Game
+ WinningPopupPanel(token: Token, parent: Game)
+ takingDecision(): void
DecisionPanel
- result: int
+ DecisionPanel()
+ takingDecision(): boolean
<<abstract>>
GameComponent
- animal: Animal
- point: Point
- isOccupied: boolean
+ setAnimal(animal: Animal): void
+ getAnimal(): Animal
+ getImage(): ImageIcon
+ setPoint(point: Point): void
+ getPoint(): Point
+ isOccupied(): boolean
+ setOccupied(occupied: boolean): void
Token
- dragonColour: String
- position: int
- tokenImage: ImageIcon
- imageCorner: Point
- cave: CaveCard
- path: ArrayList<GameComponent>
- hasFlipped: boolean
- currentSteps: int
+ Token(dragonColour: String)
+ toString(): String
+ getDragonColour(): String
+ getAnimalName(): String
+ setBornCave(cave: CaveCard)
+ getBornCave(): CaveCard
+ setHasFlipped(flip: boolean)
+ setPath(path: ArrayList<GameComponent>)
+ getPath(): ArrayList<GameComponent>
+ setDragonColour(colour: String)
+ getPosition(): int
+ setPosition(position: int)
+ getCurrentSteps(): int
+ setCurrentSteps(steps: int)
# paintComponent(g: Graphics): void
VolcanoCardBoard
- volcanos: ArrayList<VolcanoCard>
+ VolcanoCardBoard()
+ getVolcanos(): ArrayList<VolcanoCard>
+ setupVolcanoAnimals(): void
BoardSetup
- chitCardPanel: JPanel
- volcanoCardPanel: JPanel
- cave1Panel: JPanel
- cave2Panel: JPanel
- cave3Panel: JPanel
- cave4Panel: JPanel
- token1: JPanel
- token2: JPanel
- token3: JPanel
- token4: JPanel
+ boardSetup()
+ getChitCardPanel(): JPanel
+ setupChitCardPanel(): void
+ getVolcanoCardPanel(): JPanel
+ setupVolcanoCardPanel(): void
+ getCavePanel(number: int): JPanel
+ setupCavePanel(): void
+ getToken(number: int): JPanel
+ setupToken(): void
# paintComponent(g: Graphics): void
VolcanoCardManager
- boardSetup: BoardSetup
- volcano: VolcanoCardBoard
- volcanoIndexMap: Map<Integer, VolcanoCard>
+ VolcanoCardManager(boardSetup: BoardSetup)
+ volcanoCardSetup(): void
+ getVolcanoPath(tokenNumber: int): ArrayList<GameComponent>
+ getVolcanoIndexMap(): Map<Integer, VolcanoCard>
+ getBoardSetup(): BoardSetup
VolcanoCard
+ VolcanoCard(animal: Animal)
+ getPoint(): Point
+ paintComponent(g: Graphics): void
ChitCardManger
- gameBoard: ChitBoard
- boardSetup: BoardSetup
- labelIndexMap: Map<JLabel, Integer>
- animal: Animal
- movement: int
- flippedChit: ChitCard
- chitList: ArrayList<ChitCard>
- clickedChit: JLabel
- clickedChitList: ArrayList<JLabel>
- disable: boolean
+ ChitCardManger()
+ chitCardSetup(): void
+ getFlippedAnimal(): Animal
+ getChitMovement(): int
+ getFlippedChit(): ChitCard
+ setAnimal(animal: Animal): void
+ clearAnimal(): void
+ setMovement(movement: int): void
+ flipBackChit(): void
+ setFlipBack(flip: boolean): void
+ disableChitPanel(able: boolean): void
+ enableChitPanel(): void
+ getBoardSetup(): BoardSetup
+ getChitList(): ArrayList<ChitCard>
+ mouseClicked(e: MouseEvent): void
+ mousePressed(e: MouseEvent): void
+ mouseReleased(e: MouseEvent): void
+ mouseEntered(e: MouseEvent): void
+ mouseExited(e: MouseEvent): void
ChitBoard
- chits: ArrayList<ChitCard>
+ ChitBoard()
+ getChits(): ArrayList<ChitCard>
+ createBoard(): void
+ shuffleChits(): void
+ configureChitBoard(): void
ChitCard
- isFlipped: boolean
- movement: int
- chitCardImage: Image
+ ChitCard(animal: Animal, movement: int)
+ getMovement: int
+ flipCard(flip: boolean): void
+ isFlipped(): boolean
CaveCardManager
- boardSetup: BoardSetup
- caves: ArrayList<CaveCard>
- tokens: ArrayList<Token>
- shuffledToken: ArrayList<Token>
+ CaveCardManager(boardSetup: BoardSetup, tokens: ArrayList<Token>)
+ parseCave(token: Token): int
+ caveSetup(): void
+ getShuffledToken(): ArrayList<Token>
+ getBoardSetup(): BoardSetup
CaveCard
+ CaveCard(animal: Animal)
# paintComponent(g: Graphics): void
AnimalFactory
+ createChitCardAnimal(): ArrayList<Animal>
+ createCaveAnimal(): ArrayList<Animal>
+ createVolcanoCardAnimal(): ArrayList<Animal>
<<abstract>>
Animal
- animalName: String
- maxMovement: int
- isBackwardable: boolean
+ Animal(animalName: String, maxMovement: int)
+ getAnimalName(): String
+ getMaxMovement(): int
+ isBackwardable(): boolean
+ setBackwardable(isBackwardable: boolean): void
+ toString(): String
Bat
+ Bat()
BabyDragon
+ BabyDragon()
PirateDragon
+ PirateDragon()
Salamander
+ Salamander()
Spider
+ Spider()
Colour Code
Revised classes
Existing classes
New classes
line relationship changes between the exisiting classes of the chosen UML Class Diagram design
code: red
asscociation
aggregation
composition
inheritance
interface
dependency
<<uses>>
<<creates>>
<<connects>>
<<validates>>
contains
consists of
manages
extends
has
manufactures