# FIT3077

# Software engineering: Architecture and design

## Sprint 3 - Documentation

Group name: Master Byter

**Written by:**

Chen Jac Kern

Chong Jet Shen

Chua Wen Yang

## 1. Screenshot of Contributor Analytics

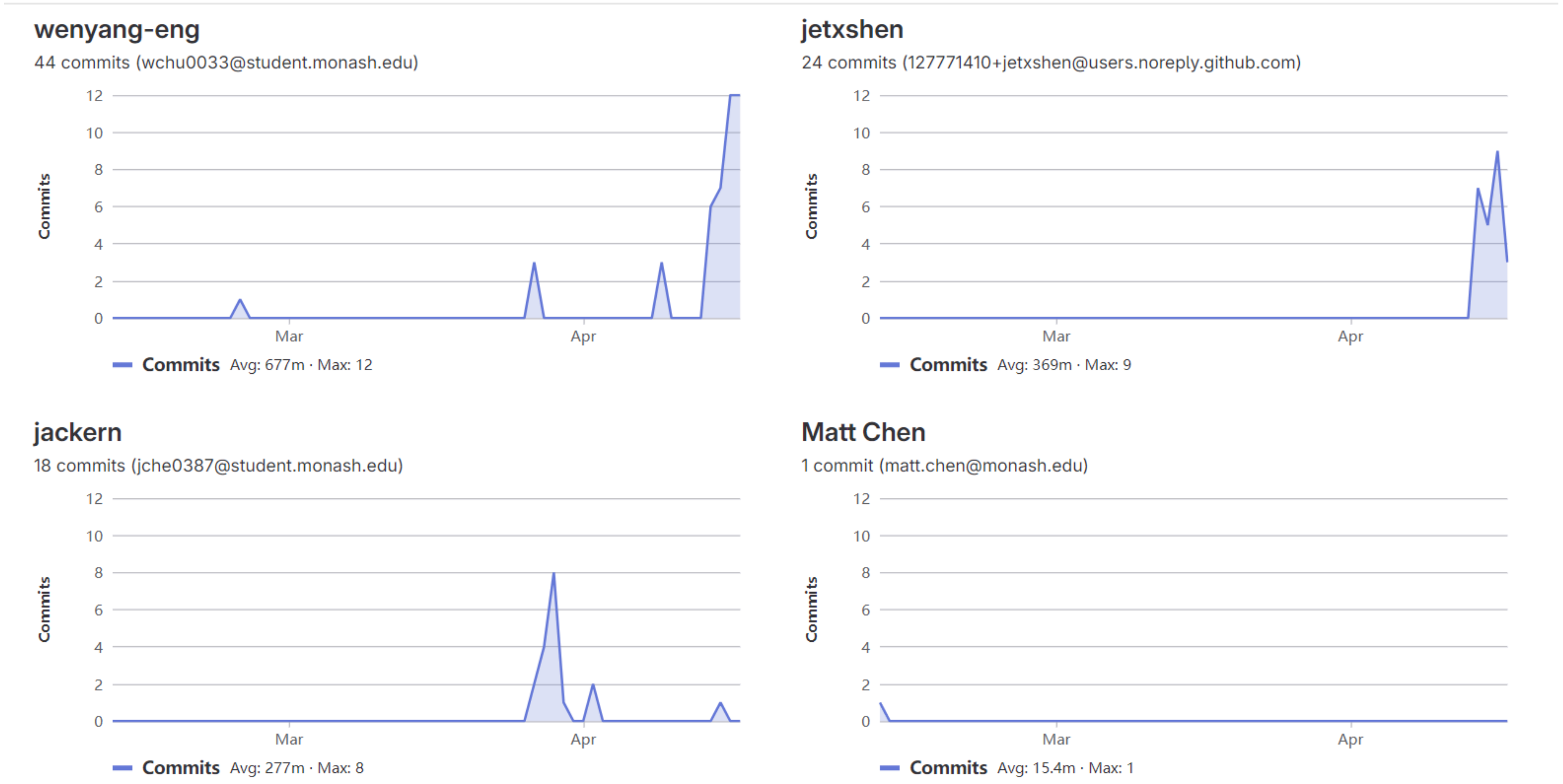


*Figure 1. Screenshot of Contributor Analytics*

## 2. Review of Sprint 2 Tech-based Software Prototypes

### 2.0 Chosen Assessment Metrics Criteria

| Factor | Characteristics | Design Principle | Metric | Acceptable value |
|---|---|---|---|---|
| Functional Suitability | Functional completeness | Complete initialisation of game board | The number of the volcano cards. | 24 |
| | | | The number of dragon cards. | 16 |
| | | | The number of caves. | 4 |
| | | | The number of tokens. | Between 2 to 4 |
| | | Complete key functionalities of the game. | The dragon cards in the game are flippable.<br>0: Incomplete<br>1: Complete | Between 2 to 5 |
| | | | The token moves the same steps as the number of animals on the flipped dragon cards.<br>0: Incomplete<br>1: Complete | |

| | | | | |
|---|---|---|---|---|
| | | | The turn is changed to the next player if the current player flips the dragon card that contains a different animal from the current position's animal of the current player.<br>0: Incomplete<br>1: Complete | |
| | | | The turn is changed to the next player if the current player needs to move to the occupied position.<br>0: Incomplete<br>1: Complete | |
| | | | The player wins if this player moves to its caves.<br>0: Incomplete<br>1: Complete | |
| | Functional Appropriateness | Only include the functions or classes that are related to the game in design. | The number of unnecessary classes or functions | 0 |
| Reliability | Faultlessness | Avoid any error occurring during the game implementation. | The number of errors occur in the game | 0 |
| Performance Efficiency | Time Behaviour | Response of the button and game actions should be fast enough. | Time taken to perform every action in the game | Less than 5 seconds |
| Maintainability | Reusability | Reuse the code or | The number of abstract classes | Between 2 to 10 |

| | | | | |
|---|---|---|---|---|
| | | attributes by calling the functions or instantiating classes | The number of classes and functions in the code | Between 5 to 15 |
| | Modifiability | Avoid introducing defects to degrade the software quality | The number of downcasting | 0 |
| Flexibility | Adaptability | Design has been adapted across other technologies. | The number of different GUI technologies the design can be applied to. | 1 |
| | Scalability | Loose coupling between the classes | The number of interfaces or abstraction layers | Between 2 to 5 |
| | | Encapsulate all the attributes | The number of private attributes | Between 2 to 10 |
| Interaction Capability | Appropriateness Recognizability | Use meaningful and descriptive names for functions and classes | The number of function or class names which are ambiguous. | 0 |
| | User Engagement | Include interactions for navigation of software | The number of buttons which interact with being clicked. | Between 5 to 10 |

*Table 1: Table of assessment criteria*

## 2.1 Solution 1 (Chua Wen Yang)

## A. Assessment

### i. Functional completeness

Jet Shen's Review:
The number of volcano cards, dragon cards, caves and player tokens in this implementation is within the accepted value in the measurement criteria. The game board is initialised with the flipping of dragon cards completely functional.

Jac Kern's Review:
All volcano cards, dragon cards, caves and player tokens are implemented within the acceptable value. Two key functionalities are met which are setting up the game board and flipping of the dragon cards

### ii. Functional appropriateness

Jet Shen's Review:
No classes and functions were never used. Every class and function has been used whether it is being used directly or through an abstraction.

Jac Kern's Review:
All classes and functions are used appropriately

### iii. Faultlessness

Jet Shen's Review:
No errors were found when testing running the game. Every part of the GUI displayed works within the functional requirements given.

Jac Kern's Review:
No errors were displayed in the terminal and no crashes occur when running the game.

### iv. Time behaviour

Jet Shen's Review:
The only function within this code implementation which involves time is the flipping of dragon cards. This functionality fits within the acceptable value of less than 5 seconds.

Jac Kern's Review:
Board is built within a second and time taken to flip dragon cards is within the acceptable value

## v. Reusability

Jet Shen’s Review:
No abstract classes but 2 interfaces were used for implementation of the actions and the animals. And 16 classes which have specific responsibilities within them.

Jac Kern’s Review:
Two interfaces were implemented for Action and Animal and placed in respective folders. There are a total of 16 other classes which can be divided into subcategory classes of Action, Animal, Button, Controller, Game and Board

## vi. Modifiability

Jet Shen’s Review:
No downcastings were identified within the entire system. All functions are correctly used by their own classes and not their parent classes.

Jac Kern’s Review:
No downcastings were found in the implementation

## vii. Adaptability

Jet Shen’s Review:
The use of Java Swing is utilised in this code implementation. Which helps with displaying all the components needed to form the game.

Jac Kern’s Review:
JavaSwing was implemented in this code implementation, which allows all components to be displayed accordingly

## viii. Scalability

Jet Shen’s Review:
There were no abstract classes but 2 interfaces used. Which is used for actions and animals, which is made easier to scale up. The number of private attributes used to encapsulate the attributes fits within the acceptance value.

Jac Kern’s Review:
There were 2 interfaces included and no abstract classes, therefore it meets the acceptable value of loose coupling. A lot of attributes are private so it meets the acceptable value

## ix.Appropriateness recognizability

Jet Shen’s Review:
All classes and functions used have appropriate names which makes it easy to recognise to anyone reading the code.

Jac Kern’s Review:
Functions implemented have documentation and names used were easy to understand therefore there is no ambiguous classes and attributes

### x. User engagement

Jet Shen’s Review:
16 buttons are implemented to represent the 16 dragon cards. All buttons are fully functional, displaying the animal and the number of them when clicked on, and flips back after.

Jac Kern’s Review:
16 buttons, which are the dragon cards can be clicked to flip them so it meets the acceptable value. However, buttons specified to be built in sprint one’s basic IU design was not implemented

## B. Summary of key findings in solution

All in all, this teammate’s design and implementation has managed to fulfil the functional requirements chosen in Sprint 2. The use of model view controllers helped distinguish the responsibilities of all the key components of the Fiery Dragons game. Every class and function are easy to recognise and this implementation. And the use of interfaces follows the acceptable value of our assessment metrics criteria which helps with further extension and modifiability.

## 2.2 Solution 2 (Chong Jet Shen)

## A. Assessment

### i. Functional completeness

Wen Yang's Review:
The number of the volcano cards, dragon cards, and caves meets the acceptable value. The number of tokens seems to be restricted to 4 but this can be modified in the actual implementation. All key functionalities are included in the UML, sequence diagram, and rationale with classes (FlipAction, WinningAction, and NextPlayerAction) and methods (moveToken(), winGame()).

Jac Kern's Review:
All volcano cards, dragon cards, caves and player tokens are implemented within the acceptable value but the respective images are not displayed. Only the functionality of setting up the game board was implemented.

### ii. Functional appropriateness

Wen Yang's Review:
The ExitAction class might be not necessary in the design because the user can click on the exit button to exit the game, and this can be done within a method.

Jac Kern's Review:
Classes are set up appropriately but some methods have not been completely implemented

### iii. Faultlessness

Wen Yang's Review:
The design seems not to lead to any errors, every game component has been properly set up to the correct position.
Jac Kern's Review:
No errors were displayed in the terminal and no crashes occur when running the game.

### iv. Time behaviour

Wen Yang's Review:
The design of the game actions does not require large memory and is not overcomplicated, so the time taken to perform any actions would be efficient enough.

Jac Kern's Review:
Board is built within a second and within the acceptable value

## v. Reusability

Wen Yang's Review:
There is one abstract class in the design solution (Animal) and it is inherited by 4 subclasses, and there are 17 classes in the design.

Jac Kern's Review:
An abstract class was implemented for different animal types and number of classes and func

## vi. Modifiability

Wen Yang's Review:
There are 2 downcastings from Graphics to Graphic2D, however, these downcastings are not because of the design issues. Therefore, the downcasting issues in this case can be ignored.

Jac Kern's Review:
Five downcasting instances were spotted which is when downcasting from Graphics to Graphics2D for the board components in order to draw them

## vii. Adaptability

Wen Yang's Review:
The design can be applied to the JavaSwing framework, and it runs in the windows.

Jac Kern's Review:
JavaSwing was implemented in this code implementation, which allows all components to be displayed accordingly

## viii. Scalability

Wen Yang's Review:
There is one abstract class and one interface in the design solution, so there are a total of 2 abstraction layers, which ensures the loose coupling in the design. From the encapsulation perspective, there might need 2 more private attributes to reach the acceptable value.

Jac Kern's Review:
There is one abstract class, which is Animal class, used to differentiate between different types of animals. There is also an interface class, used to declare behaviour of Action classes such as flip action and next player action. There were no private attributes implemented so they should be implemented to reach the acceptable value.

## ix.Appropriateness recognizability

Wen Yang's Review:
All the classes and functions have clear and descriptive names, this increases the clarity of each component in the design.

Jac Kern’s Review:
Names for classes and functions used are straightforward and easy to understand

### x. User engagement

Wen Yang’s Review:
Although the current design does not include button components, these components can be easily incorporated into this design.

Jac Kern’s Review:
No button components have been implemented but button layout is being shown

## B. Summary of key findings in solution

The overall design decisions made by this teammate in terms of classes and functions used has been made clear of their intentions. The level of abstraction meets the assessment metrics criteria acceptable value along with the clear names of classes and functions which makes clear their intentions. However, this teammate was only able to fully implement the key game functionality of initialising the game board, which is considered incomplete. Where buttons were not implemented as they were supposed to which ends up with no way of testing the user engagement.

## 2.3 Solution 3 (Chen Jac Kern)

### A. Assessment

#### i. Functional completeness

Jet Shen’s Review:
The number of volcano cards, dragon cards, caves and player tokens displayed meets the acceptable value in the measurement metric. The entire game board managed to be initialised with the flipping of dragon cards completely functional. With the additional feature of randomised positions of dragon cards after restarting the game.

Wen Yang’s Review:
All of the game components have met the acceptable value, and most of the key functionalities are included in the UML diagram with the methods flipDragonCard(), stepsTaken(), and playerStartRound(). For the winning situation, based on the design rationale, it is validated by the steps taken.

#### ii. Functional appropriateness

Jet Shen’s Review:
All functions and classes were used with none of them being unnecessary or redundant.

Wen Yang’s Review:
The functions and classes in the design are necessary and perform their own responsibilities.

#### iii. Faultlessness

Jet Shen’s Review:
No errors were found when testing running the game. Every part of the GUI displayed works within the functional requirements given.

Wen Yang’s Review:
The design is not error-prone and all game components are displayed in the correct position.

#### iv. Time behaviour

Jet Shen’s Review:
Every action performed in this implementation fits within the 5 second boundary from our assessment metric acceptable value.

Wen Yang’s Review:
Based on the implementation of the flipping card and clicking on the button, the time taken is less than 5 seconds which meets the acceptable value.

## v. Reusability

Jet Shen's Review:
No abstract classes were used. Instead an enum was used for animals where they were used to handle the abstraction. A total of 10 classes were used in this whole code implementation.

Wen Yang's Review:
There is no abstract class in the design solution and implementation. The number of classes and functions fall into the acceptable value (10 classes and 15 functions) in the UML diagram. For implementation, it includes more reusable functions (getter and setter methods) which enhance the reusability of the software.

## vi. Modifiability

Jet Shen's Review:
As no abstraction was used, no down castings were possible.

Wen Yang's Review:
There is no downcasting found in the code implementation.

## vii. Adaptability

Jet Shen's Review:
Only Java Swing was utilised in this code implementation. Which helps with displaying all the components needed to form the game.

Wen Yang's Review:
The design can be applied to the JavaSwing framework, and it runs in the windows.

## viii. Scalability

Jet Shen's Review:
No layers of abstraction were identified within this code implementation. However the number of private attributes used to encapsulate them fits within the acceptance value.

Wen Yang's Review:
There might need 2 more abstract classes or interfaces to meet the acceptable value for the loose coupling. For the encapsulation perspective, most attributes are private (except constant attributes for magic numbers, but I think it is acceptable) so it meets the acceptable value.

## ix.Appropriateness recognizability

Jet Shen's Review:
Every class and function are easy to understand with none of them being ambiguous.

Wen Yang’s Review:
All the classes and functions have clear and descriptive names, this increases the clarity of each component in the design.

### x. User engagement

Jet Shen’s Review:
All dragon card buttons are fully functional. Additional buttons like the game rule, restart and start game button are fully functional and work within the functional requirements.

Wen Yang’s Review:
All GUI buttons and the position of the game components are properly functioned. It is better to make a solution that makes the frame fits a different computer size (by only modify the size of the frame, all panels and buttons’ sizes and positions changes based on the modified frame size)

## B. Summary of key findings in solution

Overall, the implementation made by this teammate has fulfilled all the functional requirements: initialising the game board and flipping of dragon cards and an additional requirement of the player winning the game. However, some level of abstraction should be considered as no abstract classes nor interfaces were used in this implementation. Which fails to meet our assessment metrics criteria acceptable value.

## 2.4 Conclusions of tech-based software prototypes assessment

Based on all the reviews and how well each of our teammates based on our own measurement metrics, we as a team have decided to use Wen Yang’s code.First of all, Wen Yang’s code from Sprint 2 meets all the functional requirements they have chosen: setting up the initial game board and flipping of dragon (“chit”) cards.

To add on, their code implementation has ticked all the boxes in terms of meeting the acceptable value of our team’s assessment metrics criteria. The use of object-oriented design principles with an appropriate level of abstraction along with proper uses of design patterns has convinced the whole team that we will further extend with Wen Yang’s code base.

However, we would have to make the following changes based on the other teammate’s design and implementation. First of all, the main menu panel was not implemented so we took ideas from Jac’s code to implement the main menu panel, with some implementation of buttons too. Other than that, the game panel, which is the board, is redesigned such that it has buttons so that functionality such as restarting the game, showing game rules and exiting the game is implemented.

By using Path, we also implemented the functionality of movement of tokens as it was not implemented for Sprint 2. Action classes was also not completely implemented for sprint 2 but now MoveForwardsAction, MoveBackwardsAction and DoNothingAction are implemented to handle how the tokens move under conditions such as token will only move forward if dragon cards are selected correctly and there are no tokens occupied at destination.

Next, we also improved the prototype by requiring the players to select the amount of players and their respective ages so that the game starts with the youngest player and goes in a clockwise direction. Lastly, we also allow the player to choose not to flip another dragon card if they prefer not after selecting a correct dragon card by displaying a window. In conclusion, these implementations helped us to improve more on the prototype and completed all the required functionalities.

# 3. Object-Oriented Design

## 3.1 CRC cards of main classes

| **Class 1: Game** | |
|---|---|
| Process each player’s turns | Token, ChitCardController, ChitCard |
| Flip chit cards | ChitCardController, ChitCard |
| Move players | Token, ChitCard |
| Knows all the players in the game | Game |
| Creates game page navigation buttons | GameRuleButton, RestartButton, ExitButton |
| Knows if a player wins the game | Game |

Description:
The Game class is a key class that facilitates the entire Fiery Dragons game. This includes all key game functionalities like initialising the game board, flipping of chit card, movement of player based on the animal and value on the flipped chit card.

| **Class 2: ChitCardController** | |
|---|---|
| Creates chit cards | GamePanel, Deck |
| Knows the location of all chit cards | ChitCardController |

Description:
The ChitCardController class is a key class that controls all the chit cards in the game. This helps create the specified number of chit cards, along with the animal and value of each chit card.

| **Class 3: VolcanoCardController** | |
|---|---|
| Creates Volcano Cards | GamePanel, VolcanoCard |
| Knows the volcano cards adjacent to each cave | VolcanoCardController |
| Knows all the volcano cards in the game | VolcanoCardController |

Description:
The VolcanoCardController class is a key class that controls all the volcano cards in the game. This helps create the specified number of volcano cards set in the game, along with knowing which volcano cards are adjacent to the caves.

| **Class 4: TokenController** | |
|---|---|
| Creates player Tokens | TokenController, Token |
| Knows all the tokens in the game | TokenController |
| Compute the paths each token takes in the game | TokenController, VolcanoCard, Cave |

Description:
The TokenController class is a key class that controls the movement of token cards in the game. This helps specify the path each token takes in the game for each token to reach their designated destination.

| **Class 5: CaveController** | |
|---|---|
| Creates Caves | GamePanel, Cave |
| Knows all the caves in the game | CaveController |

Description:
The CaveController class is a key class that controls the number of caves in the game, along with being able to initialise the caves according to the number of players in the game.

| **Class 6: Home** | |
|---|---|
| Creates the home page | Home |
| Creates home page navigation buttons | StartButton, GameRuleButton, ExitButton |

Description:
The Home class is a key class that composes the entire home page, which is the landing page of our Fiery Dragons game before starting the game.

### 3.1.5 Alternative distribution of CRC cards

| **Class 1: Game** | |
|---|---|
| Process each player's turns | Token, ChitCardController, ChitCard |
| Flip chit cards | ChitCardController, ChitCard |
| Move players | Token, ChitCard |
| Knows all the players in the game | Game |
| Creates game page navigation buttons | GameRuleButton, RestartButton, ExitButton |
| Knows if a player wins the game | Game |
| Creates the home page | Game |
| Creates home page navigation buttons | StartButton, GameRuleButton, ExitButton |

Rejection justification:
Previously our team had the idea of facilitating the entire game within the Game class, which also includes the main page before entering the game. However, this gives this class too many responsibilities.

In our team's initial lofi prototype, we decided to have a home page, which is the landing page of the game before entering the game. As we realised this landing page could be separated from our Game class, we decided against this distribution of responsibilities of the Game class and proceeded with making a separate Home class to handle the landing home page.
The highlighted rows indicate the responsibilities we have removed from the Game class and added to our final implementation of the Home class.

| **Class 2: ChitCard** | |
|---|---|
| Creates chit cards | GamePanel, Deck |
| Knows the location of all chit cards | ChitCardController |
| Sets the animal and value of each chit card | ChitCard |
| Knows the animal and value of each chit card | ChitCard |

| Class 3: VolcanoCard | |
|---|---|
| Creates Volcano Cards | GamePanel, VolcanoCard |
| Knows the volcano cards adjacent to each cave | VolcanoCard |
| Knows all the volcano cards in the game | VolcanoCard |
| Sets the the animal on each space | VolcanoCard |
| Adds token to each volcano card space | VolcanoCard |
| Removes token from each volcano card space | VolcanoCard |
| Knows if each volcano card space is occupied by another token | VolcanoCard |

| Class 4: Token | |
|---|---|
| Creates player Tokens | Token |
| Knows all the tokens in the game | Token |
| Compute the paths each token takes in the game | Token, VolcanoCard, Cave |
| Knows its position on the board | Token |
| Knows the order of player's turn in the game | Token |
| Knows whether to move forward, backward, or stay put | Token, Move |

| Class 5: Cave | |
|---|---|
| Creates Caves | GamePanel, Cave |
| Knows all the caves in the game | Cave |
| Knows the position of the cave on the game board | Cave |
| Knows which animal is associated with each cave. | Cave, Animal |

Rejection justification:
The following justification applies to all Class 2-5 above. This is because the alternative solution mentioned involves keeping all responsibilities within each component class (ChitCard, VolcanoCard, Token, Cave), whereas our chosen responsibility involves a separate ‘controller’ class.

As there are a lot of responsibilities to be handled within each component of the game, and many of these responsibilities are different from each other. Our team has chosen to separate the responsibilities of each component to their own ‘controller’ class.

This is based on the Model View Controller Design Pattern, where a ‘controller’ acts as an intermediary between what players can see and the logic behind certain actions of the game. Separating certain responsibilities away from the actual component of the class to utilise the controller classes, will keep the code to be more modular and flexible for further potential extensions.

Furthermore, our component classes responsibilities become more clear as our chosen distribution of responsibilities for each component class only involves attributes solely related to their designated component. While their controller classes facilitate the other responsibilities which involves each component of the game.

The responsibilities highlighted are the responsibilities which have been moved to their respective controller classes in our final distribution of responsibilities shown in our CRC cards.

## 3.2 Revised class diagram

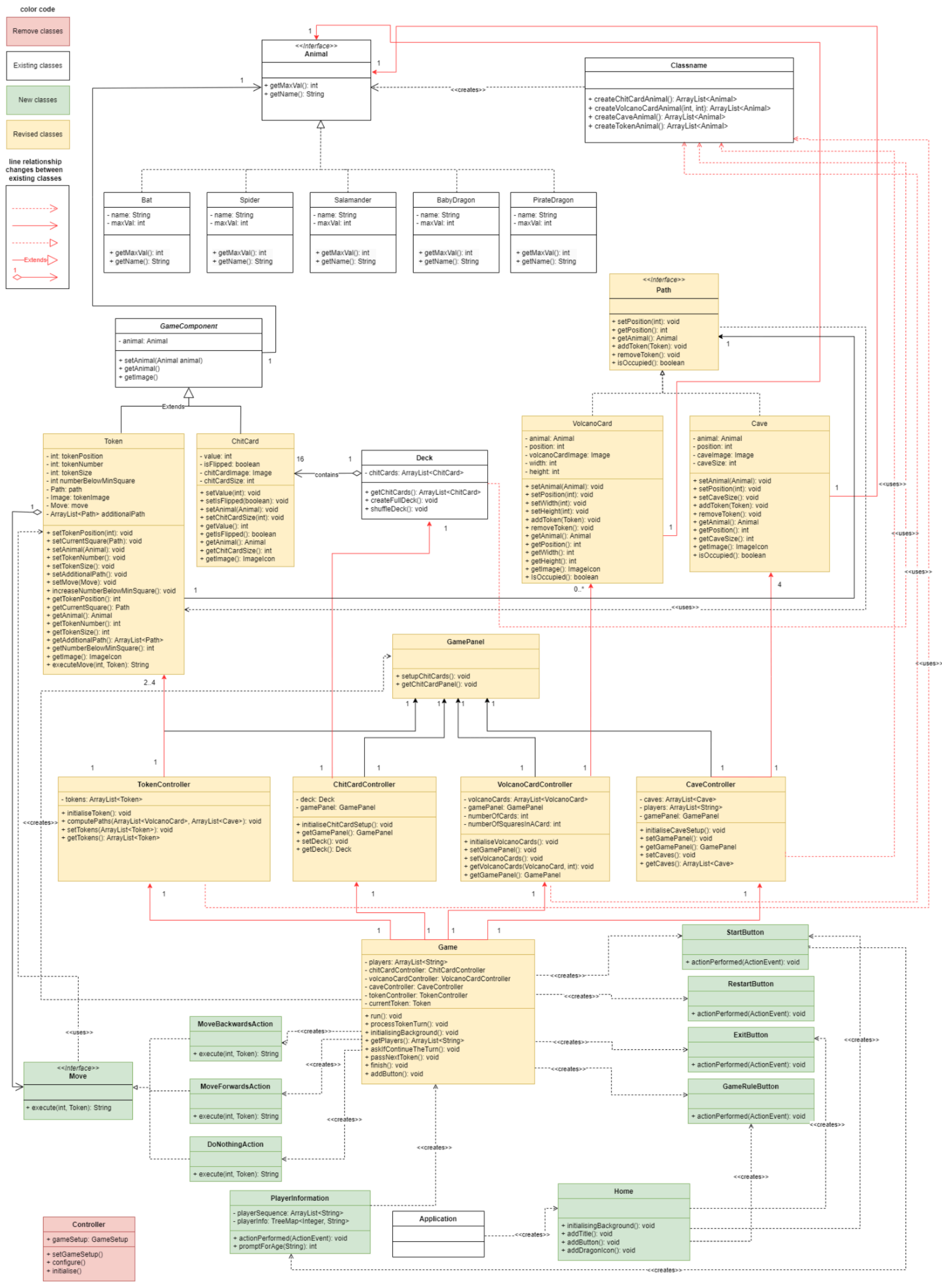


*Figure 2 Revised UML class diagram*

In this revised UML diagram, Controller class is removed from the previous UML class diagram because we intend to use the method of the subclasses (TokenController, ChitCardController, VolcanoCardController, and CaveController) which the superclass (Controller) does not consist of in the Game class. For example, the getTokens(), getVolcanoCards(), and so on which are included in the subclasses, but not included in the Controller class. Therefore, the polymorphism between the Controller class and its subclasses is not applied in this design.

To implement the movement of the dragons in the game, Move interface, MoveForwardsAction class, MoveBackwardsAction class, and DoNothingAction class are added in the design solution. The purpose of adding these classes and interfaces is to avoid all the key functionalities implemented in the Game class (avoiding god class). Besides, this approach satisfies the Dependency Inversion Principle (DIP) because these classes implement the Move interface, whereas the Move interface is the attribute of the Token class. It also follows the Open Closed Principle (OCP) since adding a new movement class will not affect the existing classes.

Furthermore, we added classes of StartButton, RestartButton, ExitButton, and GameRuleButton to improve the user experience and user engagement in our design. These classes are created and used in the Game, Home, and PlayerInformation classes. Although the button functionalities can be executed as the methods in those classes, creating classes for each of them will ensure the Single Responsibility Principle (SRP) is followed in the design, and hence avoiding the classes which have the button functionalities become the God classes. For instance, GameRuleButton is only responsible for showing the game rules in point form with the dialog.

Additionally, some classes are revised to implement all key functionalities and to reduce some redundant methods. For example, in the previous design, there are isOccupied() methods and canTokenEnter() methods in the Path interface. However, these two methods are actually redundant because they both have the same functionality of confirming whether the token can move into the Path. Therefore, canTokenEnter() is removed from the interface, so only isOccupied() implements this functionality.

In the current design, three design patterns are included: FactoryMethod (creational), Model, View, and Controller pattern (structural), and Strategy pattern (behavioural) are applied to the design solution. For the Factory method, AnimalFactory consists of factory methods such as createChitCardAnimal to create animals for each game component. The Animal interface is the product, and each animal class (Spider, Bat, Salamander, BabyDragon, PirateDragon) is the concrete product. For the MVC pattern, Model is each game component class (Token, Cave, ChitCard, VolcanoCard), View is the GamePanel because each game component will be displayed with this class, and each controller is the Controller. Lastly, for Strategy pattern, each action class (DoNothingAction, MoveForwardsAction, MoveBackwardsAction) is the concrete strategy with the Strategy interface of Move, whereas the context and client classes are the Token and Game class respectively.

# 4. Executable

## 4.1 Description of executable

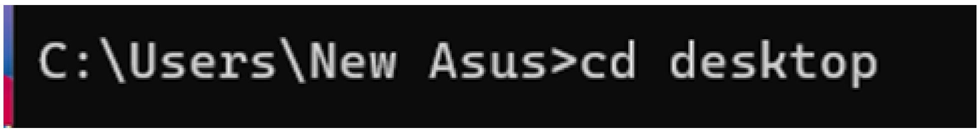
Windows is the main platform for the executable file. Before opening and running the file, you would like to have Java SE Development Kit 22.0.1 and OpenJDK-22 (Oracle OpenJDK version 22.0.1) installed on your machine.

Here is the website link for installation:

https://www.oracle.com/my/java/technologies/downloads/#jdk22-windows

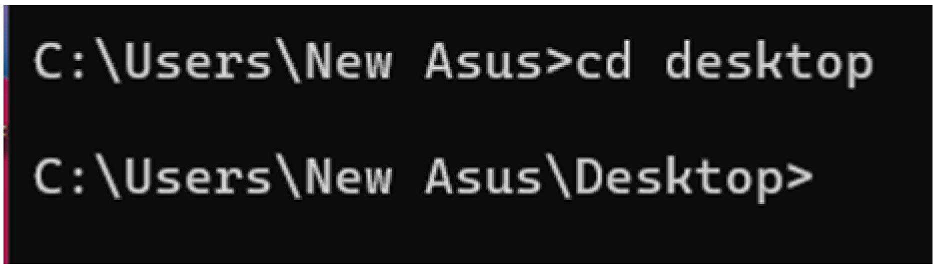
## 4.2 Instructions of running executable

1. Download the sprint_3.jar zip folder.
2. Unzip the folder then copy the jar file to the desktop.
3. Open the command prompt and type: cd desktop
   For example:



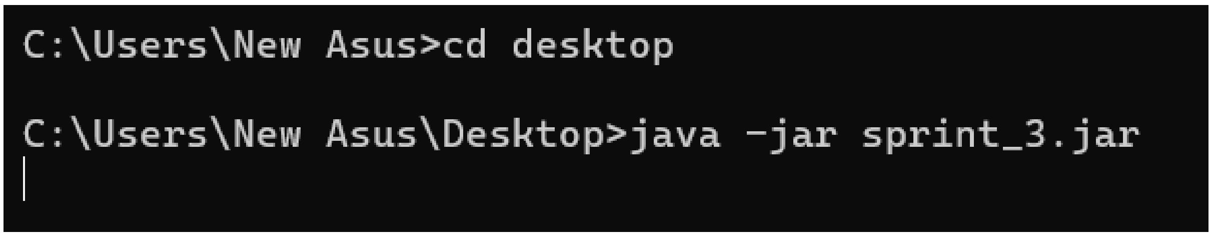
```
C:\Users\New Asus>cd desktop
```

4. Press Enter, you should find that on the next line, there is "\Desktop"



```
C:\Users\New Asus>cd desktop

C:\Users\New Asus\Desktop>
```

5. Then continue to type: java -jar sprint_3.jar



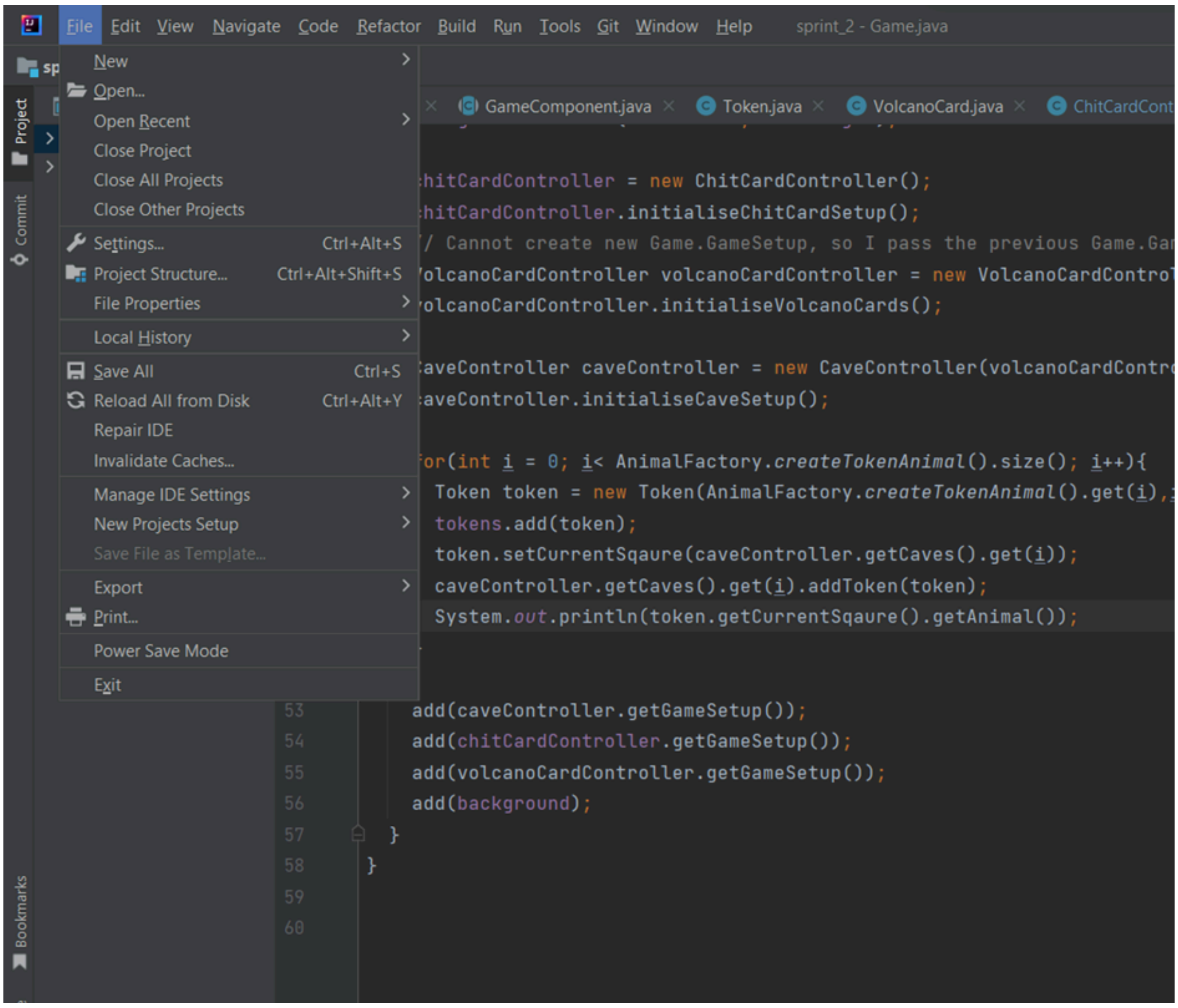
```
C:\Users\New Asus>cd desktop

C:\Users\New Asus\Desktop>java -jar sprint_3.jar
```

6. Press Enter again, the game should be open.

## 4.3 Instructions of building executable

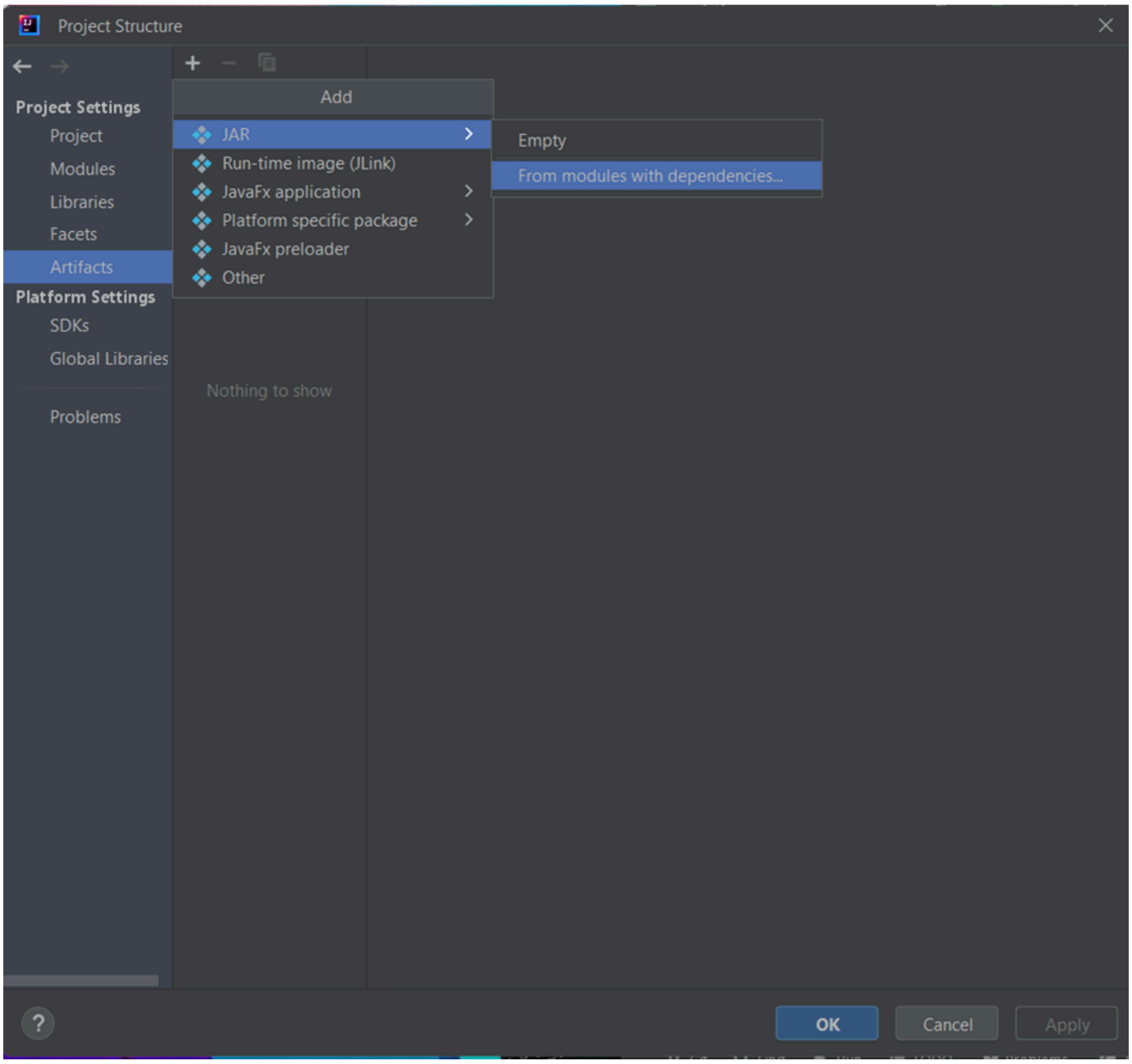
Our team applied IntellijIDEA as the IDE to create the project, here are the steps to build the executable .jar file:

1. First, click on the File -> Project Structure…

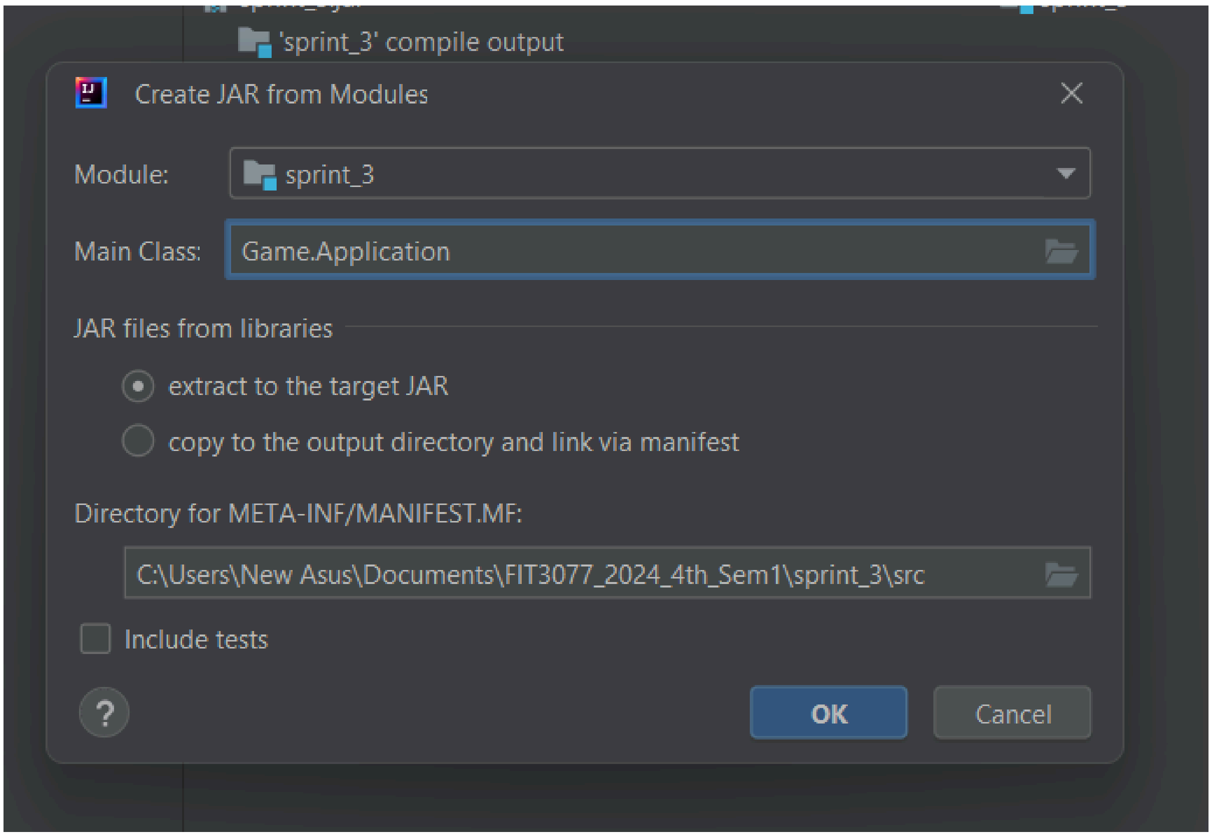


```
hitCardController = new ChitCardController();
hitCardController.initialiseChitCardSetup();
/ Cannot create new Game.GameSetup, so I pass the previous Game.Ga
olcanoCardController volcanoCardController = new VolcanoCardContro
olcanoCardController.initialiseVolcanoCards();

aveController caveController = new CaveController(volcanoCardContr
aveController.initialiseCaveSetup();

or(int i = 0; i< AnimalFactory.createTokenAnimal().size(); i++){
 Token token = new Token(AnimalFactory.createTokenAnimal().get(i),
 tokens.add(token);
 token.setCurrentSqaure(caveController.getCaves().get(i));
 caveController.getCaves().get(i).addToken(token);
 System.out.println(token.getCurrentSqaure().getAnimal());


        add(caveController.getGameSetup());
        add(chitCardController.getGameSetup());
        add(volcanoCardController.getGameSetup());
        add(background);
    }
}
```

2. Next, navigate from “Artifacts” -> click on “+” signal -> select “JAR” -> “From modules with dependencies”, then click Apply -> OK.



3. Choose the main class, and click OK.


'sprint_3' compile output

Create JAR from Modules

Module: sprint_3

Main Class: Game.Application

JAR files from libraries

● extract to the target JAR

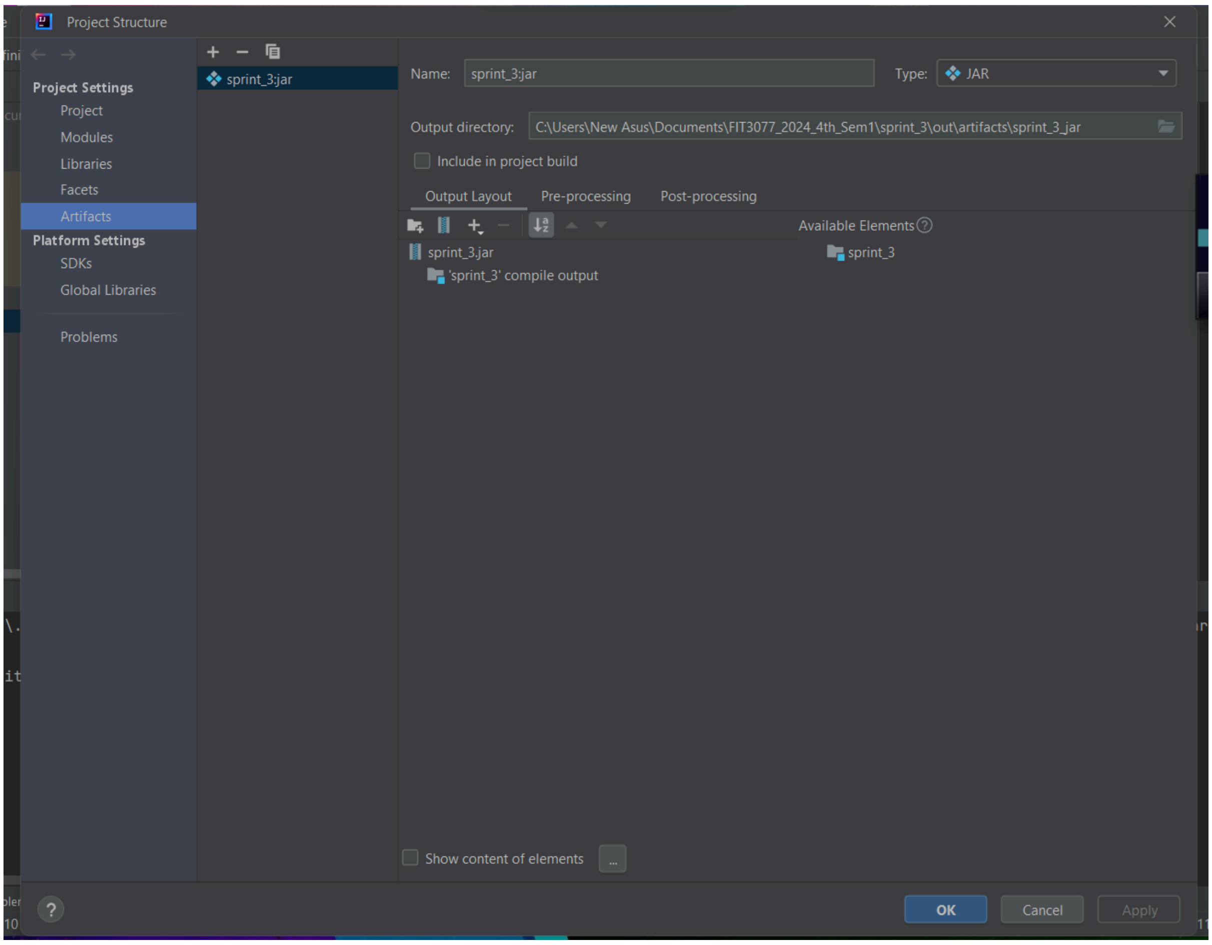
○ copy to the output directory and link via manifest

Directory for META-INF/MANIFEST.MF:

C:\Users\New Asus\Documents\FIT3077_2024_4th_Sem1\sprint_3\src

☐ Include tests

4. Navigate to Apply -> OK.


Name: sprint_3:jar

Type: JAR

Output directory: C:\Users\New Asus\Documents\FIT3077_2024_4th_Sem1\sprint_3\out\artifacts\sprint_3_jar

Include in project build

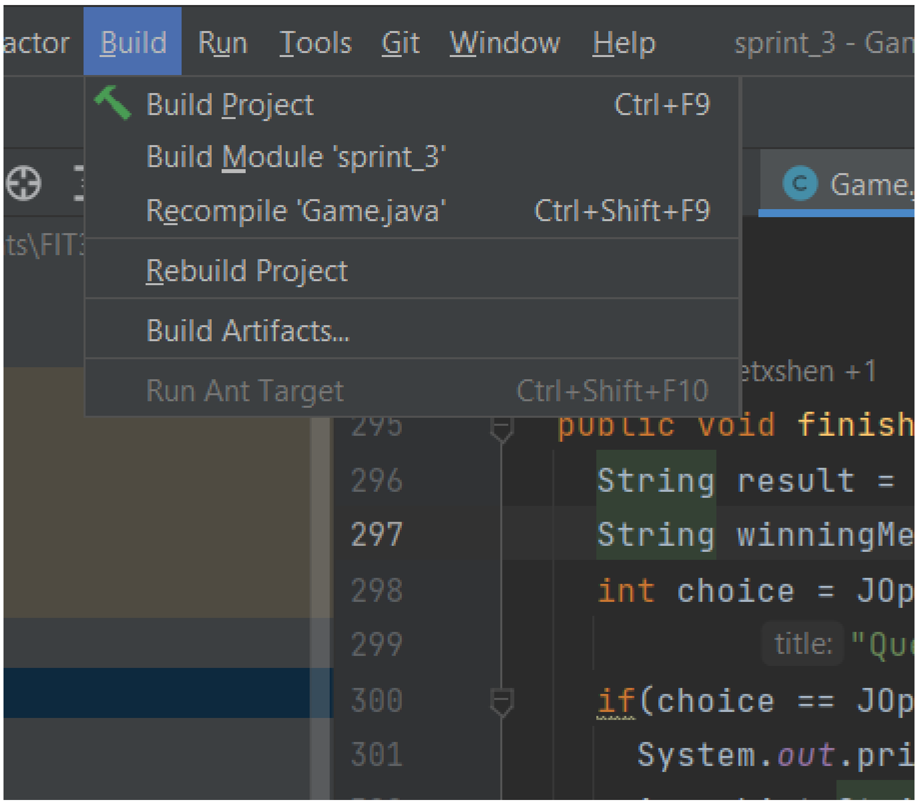
Output Layout | Pre-processing | Post-processing

sprint_3.jar
'sprint_3' compile output

Available Elements: sprint_3

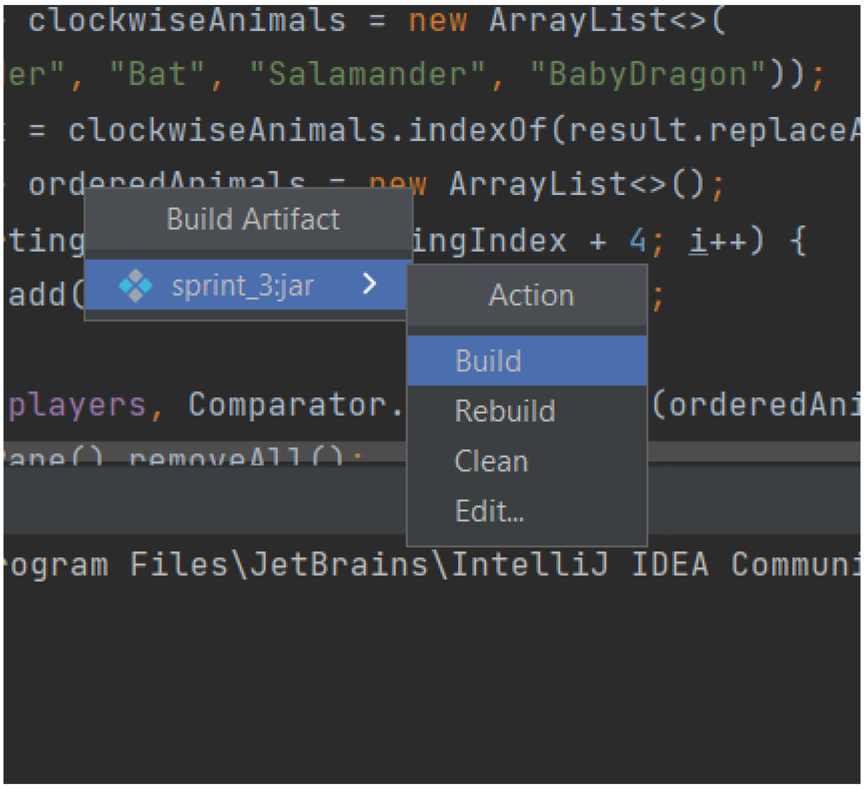
Show content of elements

5. Next, click on “Build” -> “Build Artifacts”.



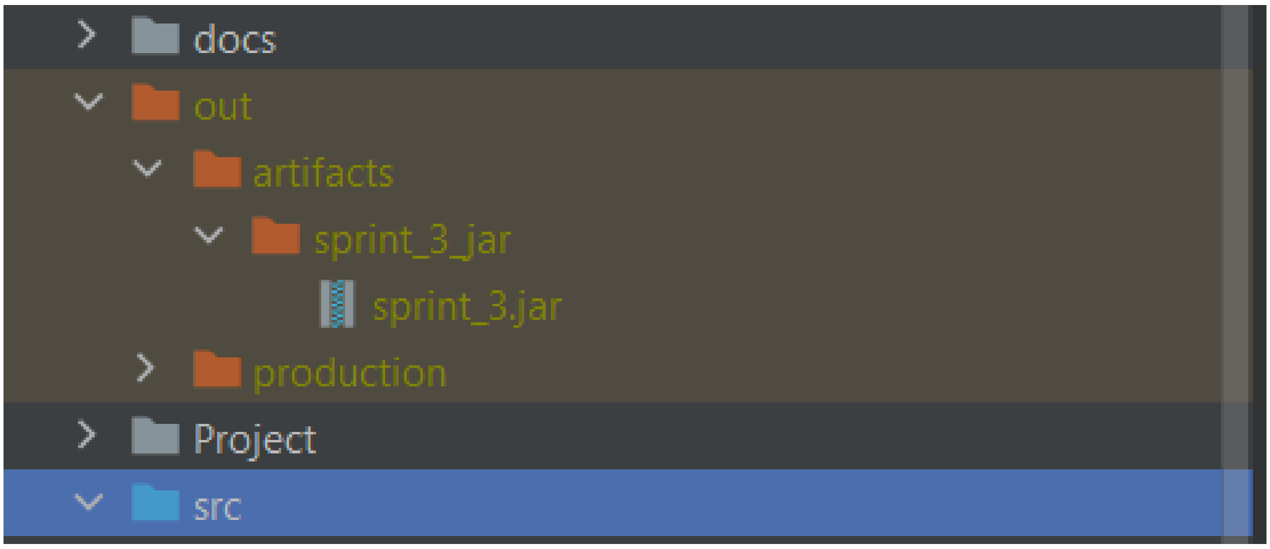
```
public void finish
    String result =
    String winningMe
    int choice = JOp
        title: "Qu
    if(choice == JOp
        System.out.pri
```

6.  Click on “Build”.



```
clockwiseAnimals = new ArrayList<>(
er", "Bat", "Salamander", "BabyDragon"));
= clockwiseAnimals.indexOf(result.replaceA
orderedAnimals = new ArrayList<>();
ting                  ingIndex + 4; i++) {
add(                                  ;

players, Comparator.                  (orderedAni
ane().removeAll();

ogram Files\JetBrains\IntelliJ IDEA Communi
```

7. Then you can see the -jar file (executable) in the directory.



## 5. Additional Notes

Here are some links for Sprint 3 for reference:

1. Sprint Contribution Log: https://docs.google.com/spreadsheets/d/1yn-9gWATjubar48b2LuoespdicjzFGZUf7PUqDMGB68/edit?usp=sharing

2. Draw.io for UML diagram: https://drive.google.com/file/d/1hC5FDw3NGPq00-kAAr5fgbs6KIzglGJM/view?usp=sharing