FIT3077 Software Engineering: Architecture and Design

Fiery Dragons Game Software

Sprint Four

Luminary Team
[MA_Tuesday12pm_Team002]

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

Contents

1. Object-Oriented Design

1.1 Revised UML Class Diagram of Fiery Dragons

This section will include the revised version of UML Class Diagram for Fiery Dragon's game of Sprint 4 as shown in Figure 1.1.1 below.

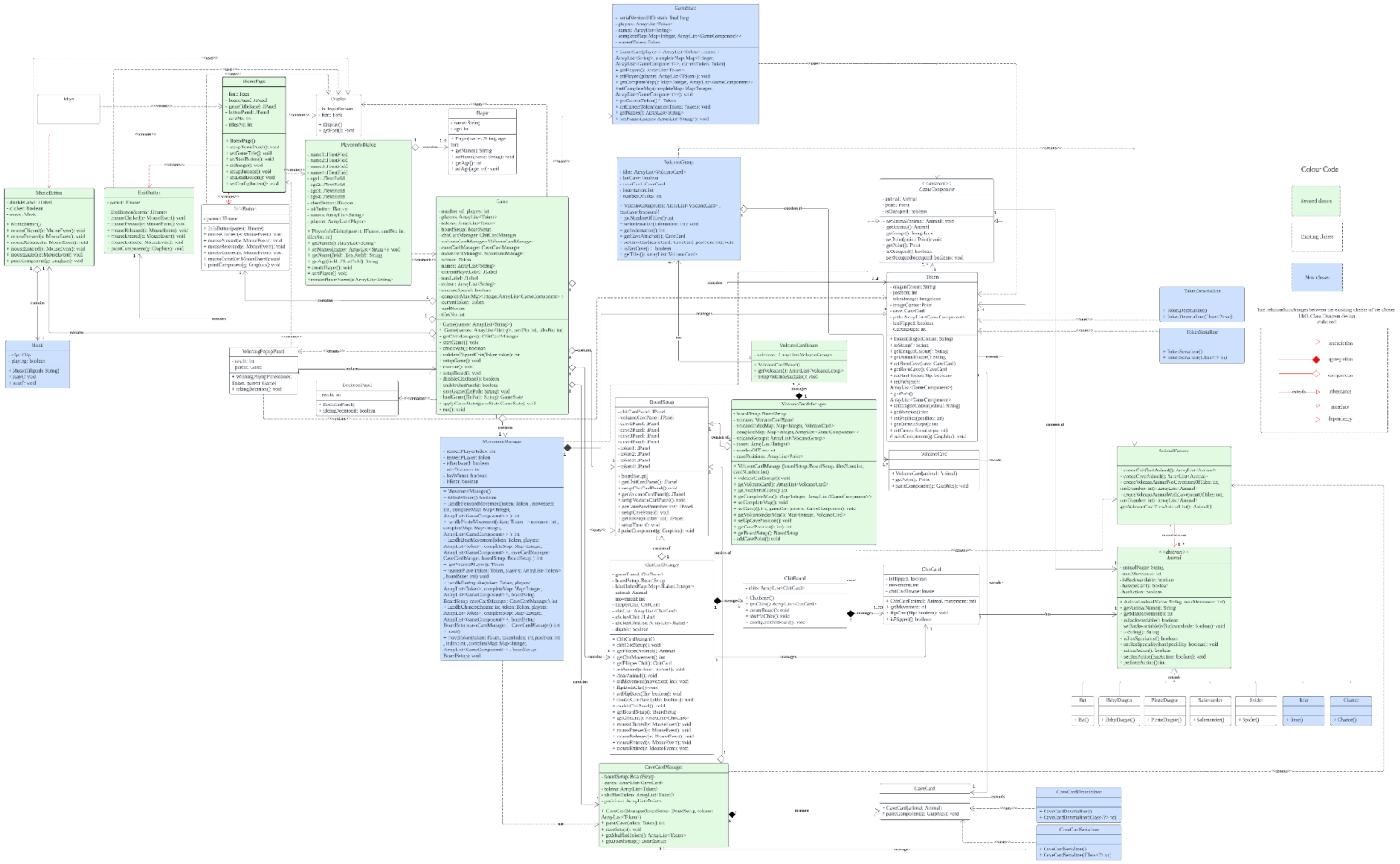


Figure 1.1.1: The revised version of UML Class Diagram for Fiery Dragon's game of Sprint 4

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

**** Remarks for UML class diagram:**
VolcanoCard - volcano card tiles
VolcanoGroup - volcano cards
VolcanoCardBoard - to configure animals on the tiles
VolcanoCardManager - to configure the arrangement of volcano cards with dynamic tiles numbers on the gameboard

2. Reflection on Sprint 3 Design

Reflecting on previous designs is significantly important for software developers and team members. This practice allows teams to assess the strengths and weaknesses of earlier designs before incorporating new extensions into the existing design and implementation. In this section, we will reflect on our design and implementation from Sprint 3 before incorporating the required and self-defined extensions in Sprint 4.

2.1 Reflection on Sprint 3 Design: Incorporation into Required Extensions

2.1.1 Required Extension 1: New Dragon Card 2: Swap the Position of the Current Dragon Token with the Nearest Token

In Sprint 4, we will incorporate a new Dragon card, Dragon Card 2, as an extension, which can swap the position of the current player's token with the nearest token on the game board. This extension posed a moderate level of difficulty to be incorporated into the implementation and design that had been implemented from Sprint 3. We encountered a few constraints in our Sprint 3 design and implementation that made addressing this extension moderately challenging.

In the Sprint 3 design, we assigned a specific path for each token to ensure that only a token from its specific cave could re-enter that cave. We made this design decision because we were focused on meeting the basic requirements for Sprint 3 without considering the possibility of future extensions. Later, we realised that incorporating the extension, which involves swapping the position of the current token with the nearest token, posed difficulties. This was because we needed to swap the position of the current dragon token regardless of its location, even if the current token to be swapped was still in the cave, and any of the other players' tokens were outside the cave. The exception was when the current dragon token was still in the cave and did not need to be swapped. As a result, our get_path method to generate a specific path for each token in the VolcanoCardManager class could no longer be used. To address this issue, we took a few days to review our design and implementations from Sprint 3, particularly in the VolcanoCardManager and Game classes, to appropriately handle the movement of the current token after being swapped with the nearest player's token when the player flips the new dragon card. During this review, we discovered the presence of code smells, especially in the Game class. We had distributed responsibilities too heavily within the Game class during Sprint 3, which unintentionally resulted in it becoming a 'God' class by centralising too many responsibilities. This violated the Single Responsibility Principle (SRP). Specifically, we designed Game class to handle the calculation of the token's moving steps for each scenario of dragon cards applied to the token within the same method used to validate the flipped dragon card in the Game class. This decision led to the ValidateFlippedChit method becoming long and complicated, making it difficult to extend and less modular.

Looking back at Sprint 3, if we could start over again, there are several strategies we would employ to make incorporating this extension easier. We would implement a more flexible event-driven architecture that would allow for dynamic interactions between game elements, such as the interactions between the players' dragon tokens and the volcano cards board. Instead of having each player's dragon token move on its own path, we would change to a universal path map so that every token could move anywhere on the same map, promoting movement flexibility. We believe that if we made these changes, our codebase would be easier to extend with new features like token swapping, while maintaining code cohesion and readability. Moreover, we would alleviate the responsibility of the Game class in handling movement and step calculations by introducing a new class, such as a MovementManager class, focused on managing all the dragon tokens' movements on the volcano cards. By doing so, we would enhance the modularity of our codebase by abstracting common game mechanics, such as movement and step calculations, into reusable components. This would facilitate the integration of new features by reducing code duplication and promoting code reusability. Besides, we would avoid centralising responsibilities in the Game class, which unintentionally violated the Single Responsibility Principle (SRP), since the Game class was supposed to handle the overall execution of the gameplay.

From the issues we discovered when reflecting on our design and implementation of Sprint 3, we gained valuable insight: we should consider not only the current requirements but also potential future extensions when designing software. A flexible and extensible design can save significant time and effort when new features need to be incorporated. Moving forward, we will prioritise designing our systems with extensibility in mind. This involves anticipating possible future changes and designing the architecture to accommodate them easily.

2.1.2 Required Extension 2: Saving and Loading The Game From/To an External File Using Suitable Text File Format

Overview

In order to improve the user experience of the Fiery Dragons game, our team added functionality to save and load game states from an external file. This exercise in implementation is essential in helping players resume their game at a later time. Since it is supported almost universally by all programming environments, we have adopted JSON as the format for the text file, considering its readability and that it is straightforward to use if compared with other formats.

Implementation Details

The save and load functionality turned out to be tough to do since we started with a lot of design decisions and constraints right from Sprint 3. This is because our architecture did not account to centralise the control of the game state, which would facilitate the addition of saving and loading functionalities. We also made quite a few additions to our codebase to achieve solutions to these problems. Jackson library is picked as the first option because it provides the related jar files and they can be added manually to the project without using Maven or modifying the XML file.

Engaged New Classes:

1. GameState: A class that manages the game state for saving and loading.
2. CaveCardSerializer, CaveCardDeserializer: Serialize/Deserialize CaveCard objects.
3. TokenSerializer, TokenDeserializer: Serialize and Deserialize Token objects.

Capturing Game State: The GameState class captures the current state of the game, including player positions, animal types on each tile, and the state of each card (volcano and cave cards).
Serialization: Class SaveState uses JSON serialization to convert the object GameState into a JSON string. It has two different serializers: one for cave cards named CaveCardSerializer, and another one for player tokens named TokenSerializer.
Writing to File: Serialized JSON string is written into an external file game_save.json
Loading the Game
Reading the File: SaveState class reads the file game_save.json and deserialize it back JSON string into a GameState object
Reinitializing Game State : GameState object is used to reinitialize the game board. The AnimalFactory class reproduces the animals in the board while restoring every tile and card to its saved state.
Deserialization: The CaveCardDeserializer and TokenDeserializer classes are responsible for the deserialization of the cave cards and player's tokens, making sure that they are correctly reloaded.

Challenges that we met

While developing the save and load functionality, many challenges came up with regard to the constraints of our design in Sprint 3:

1. The JSON library:

The current project does not use Maven to build the project, therefore, Jackson library is chosen as the library provides the related jar files.

2. The Swing library component:
   The code does not use the MVC(Model-View-Controller) structure, so the code is mixed with the UI code, which uses the Swing library, therefore it is hard to implement the Serialization interface to the current code.

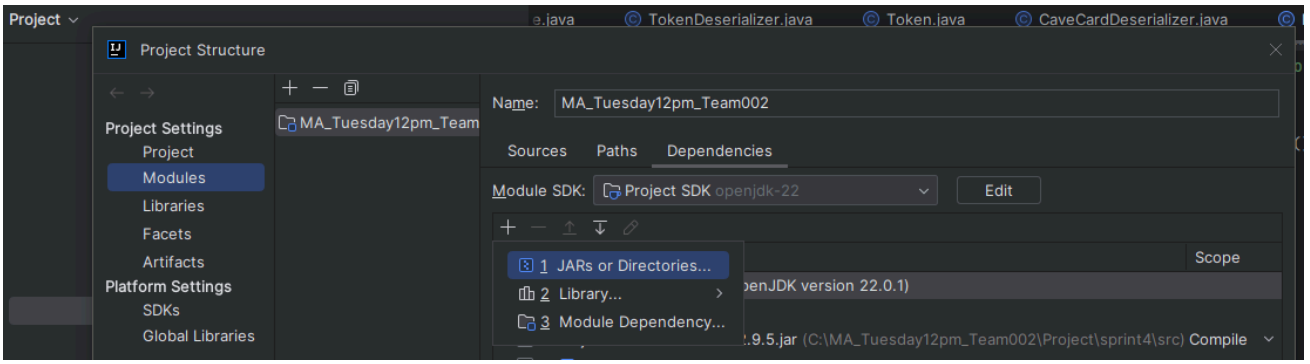
Future Improvements

By reflecting on our sprint4 development, we have identified several strategies that should be made for incorporating the extensions easier in the future:

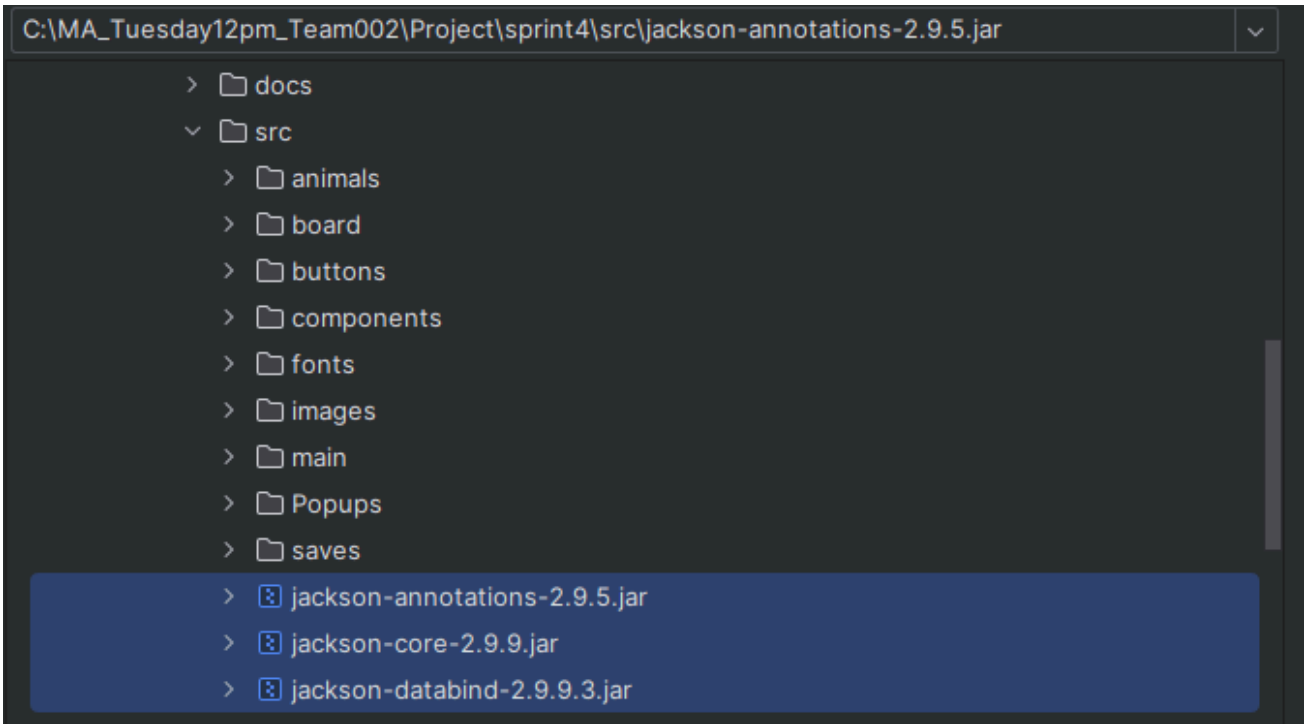
- **MVC model design**: Designing components in a modular-view-controller fashion can help isolate changes and make the system more adaptable to new requirements. Especially for serialisation.
- **Extensibility Planning**: Anticipating potential future extensions and incorporating flexibility into the initial design can save significant time and effort when new features need to be added.

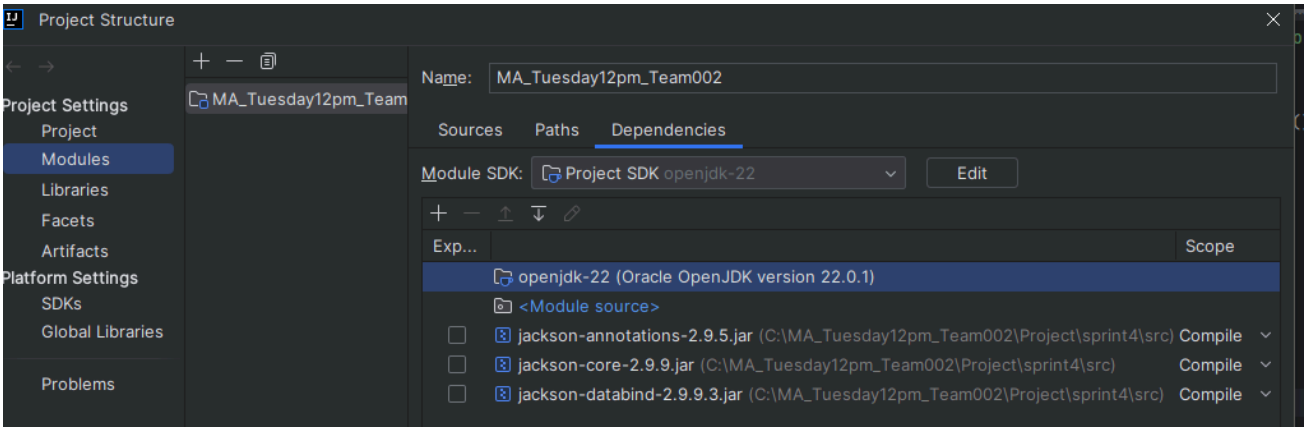
To use the Jackson file

1. Select File-> Project Structure -> Project Settings - Modules -> Click Dependencies -> Click "+" -> Select "JARs or Directories"



2. Select the three jar files under the sprint4/src


C:\MA_Tuesday12pm_Team002\Project\sprint4\src\jackson-annotations-2.9.5.jar
docs
src
animals
board
buttons
components
fonts
images
main
Popups
saves
jackson-annotations-2.9.5.jar
jackson-core-2.9.9.jar
jackson-databind-2.9.9.3.jar


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
MA_Tuesday12pm_Team
Name:
MA_Tuesday12pm_Team002
Sources
Paths
Dependencies
Module SDK:
Project SDK openjdk-22
Edit
Exp...
Scope
openjdk-22 (Oracle OpenJDK version 22.0.1)
<Module source>
jackson-annotations-2.9.5.jar (C:\MA_Tuesday12pm_Team002\Project\sprint4\src) Compile
jackson-core-2.9.9.jar (C:\MA_Tuesday12pm_Team002\Project\sprint4\src) Compile
jackson-databind-2.9.9.3.jar (C:\MA_Tuesday12pm_Team002\Project\sprint4\src) Compile

2.2 Reflection on Sprint 3 Design: Incorporation into Self-Defined Extension

2.2.1 Chance Chit Cards

Our team decides to implement a ‘chance chit cards’ extension which consists of 5 different events as our self-defined extension in Sprint 4. Having this decision as we found having chance chit cards can enhance the interaction between players with the game, to invoke their curiosity and adventurousness during the gameplay. This extension posed a similar level of difficulty with the extension, token swapping, we discussed in the previous section, which is also a moderate level of difficulty to be incorporated into the implementation and design that had been implemented from Sprint 3. The constraints we found in our design and implementation are also similar to the token swapping extension, as this extension also involves token’s movement on the volcano cards.

As we discussed previously, we assigned a specific path for each token to ensure that only a token from its specific cave could re-enter that cave in our Sprint 3 design and implementation to meet the basic requirements of the Fiery Dragon gameplay. We later realised that incorporating this extension caused the Game class to over-handle the movement and step calculation of the tokens on the game board, unintentionally resulting in it becoming a 'God' class by centralising too many responsibilities. Specifically, we designed a method in the Game class to handle the calculation of the token's moving steps for each scenario of dragon cards within the same method used to validate the flipped dragon card. With the introduction of new functionalities for the ‘chance chit cards’ to make the game more thrilling, we would need to extend the ValidateFlippedChit method. While our code supported the extensibility of new features, this approach made the method increasingly complex and complicated, thereby increasing its cyclomatic complexity and leading to code smells. The poorly written code base significantly hindered our ability to add new extensions. It took us several days to review and refactor the existing code to understand how to incorporate the new features without breaking existing functionality. This delay was primarily due to the over-centralization of responsibilities in the Game class, making it difficult to isolate and modify specific behaviours without affecting other parts of the code.

Reflecting on Sprint 3, if given the opportunity to start anew, we would implement several strategies to facilitate the incorporation of extensions. A key strategy would involve centralising movement and step calculations within a dedicated class, such as a MovementManager. This class would focus solely on managing all movements and steps of the dragon tokens on volcano cards. By segregating this functionality into a separate class, we could easily extend it to accommodate any future functionalities related to step calculations. For instance, when introducing new chance card effects like 'Dragonquake,' where all tokens move back except the current player's token, we could seamlessly integrate this feature into the MovementManager class. This approach enhances the modularity, extensibility, and reusability of our codebase. Additionally, it mitigates code duplication, which is a violation of the DRY (Don't Repeat Yourself) principle, and reinforces adherence to the Single Responsibility Principle and Open/Closed Principle. By adopting this strategy, we ensure that our codebase remains scalable and maintainable, capable of accommodating future changes and

extensions with minimal effort. Moreover, it fosters a more organised and cohesive development process, promoting collaboration and efficiency among team members.

From the issues we discovered when reflecting on our design and implementation of Sprint 3, we gained valuable insight: we should emphasise modular design approaches that encourage the separation of concerns and promote code reusability. Breaking down functionalities into smaller, self-contained units can make the system more adaptable and easier to maintain. From the lesson we had learnt, We will adopt a flexible design approach in further Sprint as well as future software projects that allows for easy integration of new features and extensions. This involves designing components with clear interfaces and minimising dependencies to facilitate future modifications.

3. Self-Defined Extension

Human values are significantly important to consider in software engineering, as they can facilitate the design of user-centric software that meets the actual needs and values of users. Schwartz proposed the Theory of Basic Human Values, which serves as a universal framework across cultures and helps to define human behaviour. The theory outlines ten core value categories, which will be considered, implemented, and manifested in the design and implementation of our self-defined extensions of the Fiery Dragon game. Hence, we will describe our self-defined extension to be implemented and address the human values related to our extension in this section.

3.1 Chance Chit Cards

After discussions among team members, we decided to implement chance chit cards as our self-defined extension to be added to our base game. Our extension particularly addresses Stimulation, which is one of Schwartz's basic human values.

We introduced various new chance effects for the chance chit cards. These chance effects included “Do Nothing”, “Advance to Move Forward”, “Pirate Dragon’s Haul”, “Dragonquake” and “Hidden Fortune”. The player’s token movement varies depending on the card effects. The descriptions and mechanics of these chance card effects are further elaborated below.

- **Chance Card Effect 1: “Do Nothing”**
  - Card Description: *"Uh Oh! \nNothing here."*
  - Mechanics: When a player draws the Do Nothing card, their token will remain in its current position and its turn will be ended.

- **Chance Card Effect 2: “Advance to Move Forward”**
  - Card Description: *"Surprise! Move 2 steps forward."*
  - Mechanics: When the current player draws the Advance to Move Forward card, their token will move 2 steps forward on the game board.

- **Chance Card Effect 3: “Pirate Dragon’s Haul”**
  - Card Description: *"Pirate Dragon looted your cave. Move 2 steps backward."*
  - Mechanics: When the current player draws the Pirate Dragon’s Haul card, their token will move 2 steps backward on the game board.

- **Chance Card Effect 4: “Hidden Fortune”**
  - Card Description: *"You discovered a buried treasure. Move three steps forward."*
  - Mechanics: When the current player draws the Hidden Fortune card, their token will move 3 steps forward on the game board.

- **Chance Card Effect 5: “Dragonquake”**
    - Card Description: *"Earthquake!!! All dragons except you move backward 1 step."*
    - Mechanics: When the current player draws the Dragonquake card, the tokens of the other players will move 1 step backward on the game board, except for the token of the current player and the current player’s turn will be ended.

To ensure players have opportunities to draw these chance cards, we design chance chit cards to be placed with the dragon chit cards and randomly arrange them in the middle of the game board so that players can randomly draw the cards and perform the effects.

By introducing these mechanics of chance chit cards, we add a layer of unpredictability and excitement, keeping the gameplay fresh and engaging. This also encourages players to take risks and seek out novel experiences, aligning with the value of Stimulation, which emphasises the need for excitement, novelty, and challenge in life. Furthermore, this new behaviour enhances the engagement of players with the possibility of both positive and negative outcomes, enhancing the overall excitement of the game.

4. 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”

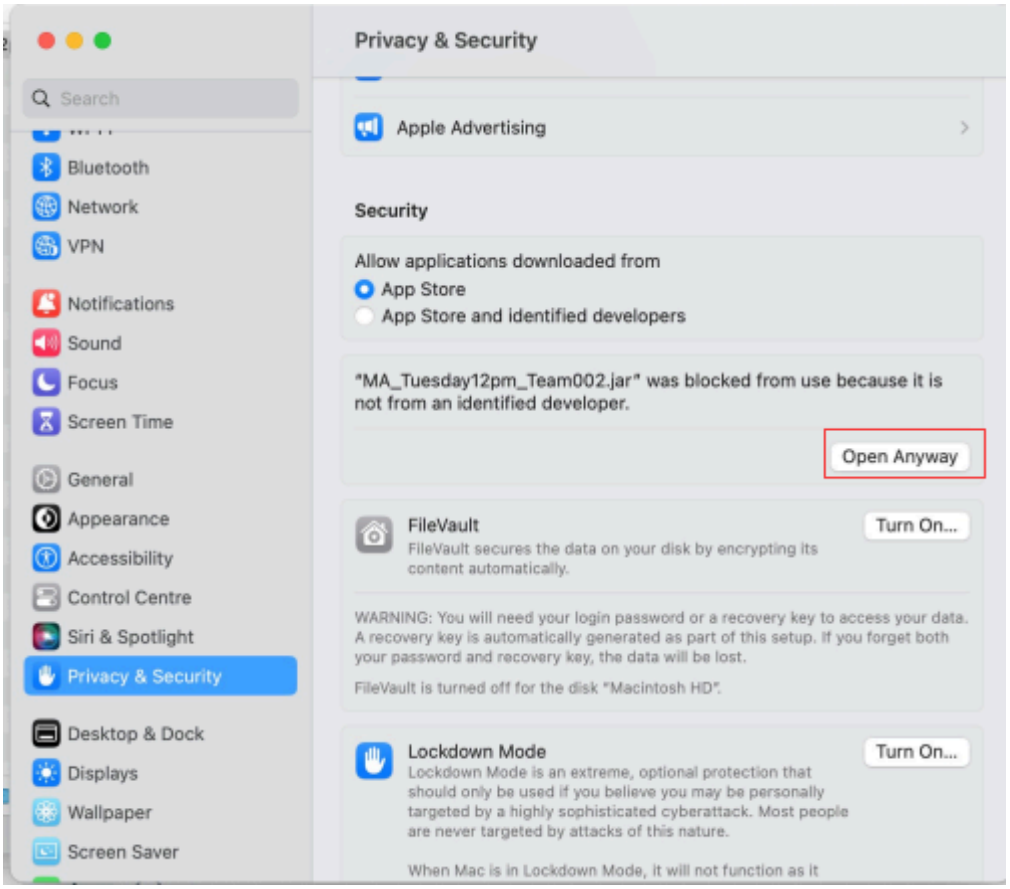


Figure 3.1.1 The Settings for macOS user

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.

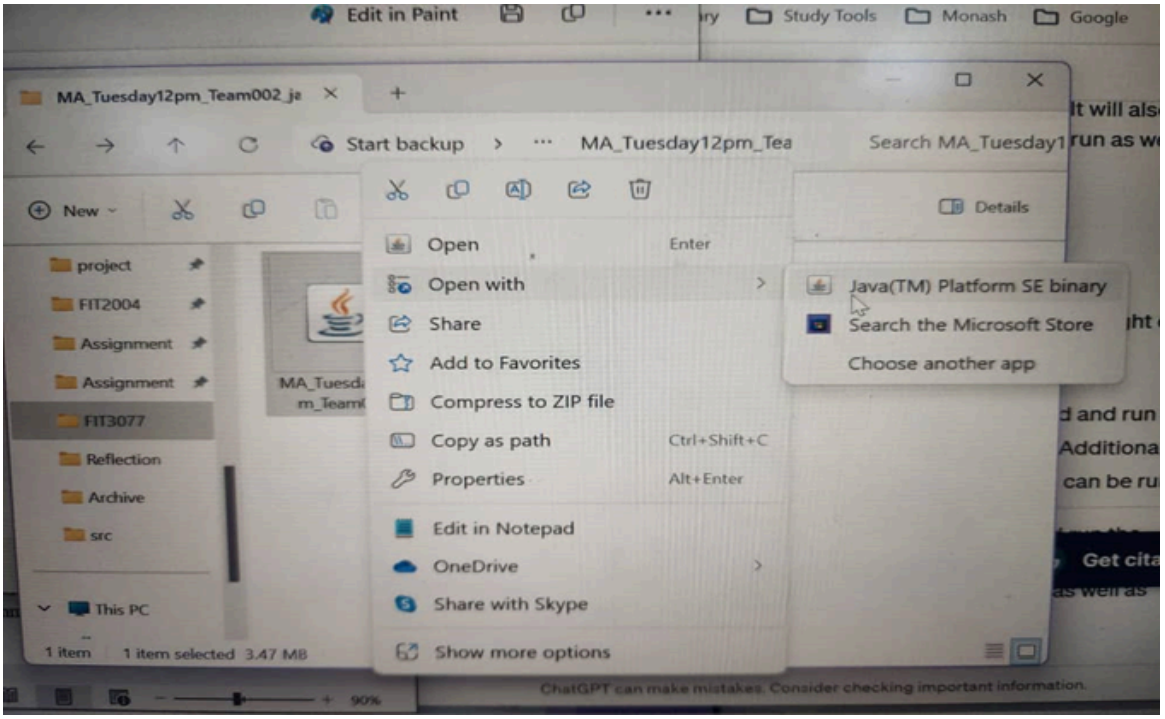


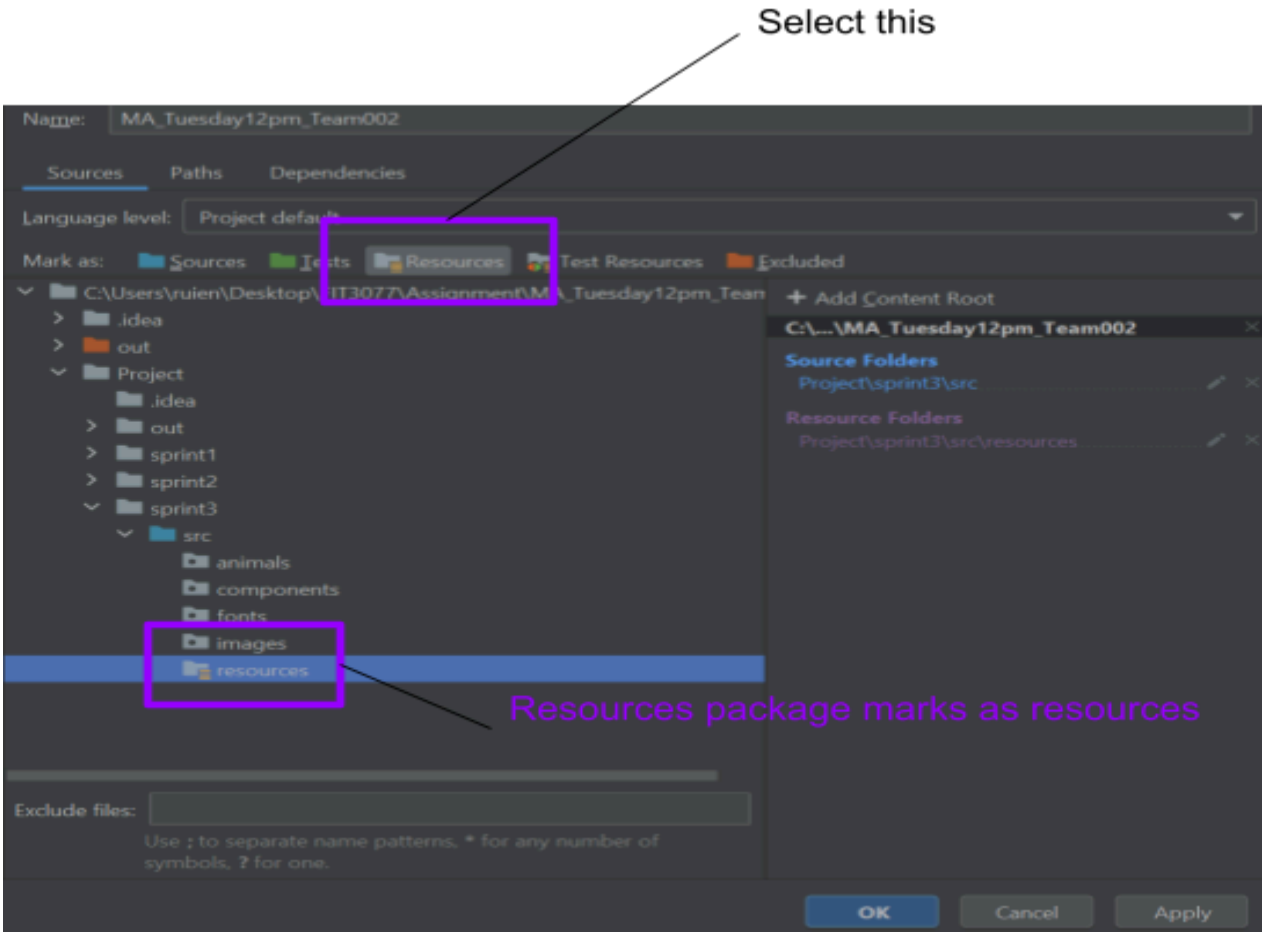
Figure 3.1.2: Instruction on using (b) to run the Fiery Dragons application

3.2 Instructions for Building Executable File of Software Prototype

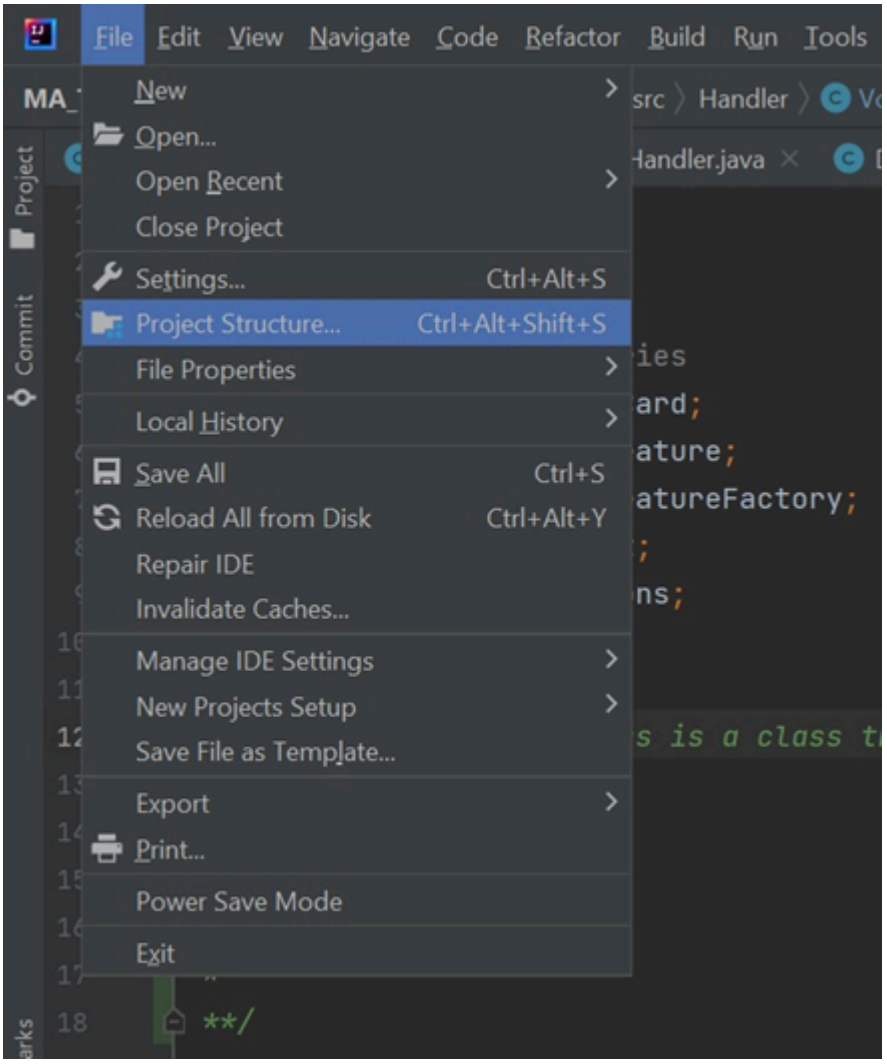
This section will demonstrate how our team built the executable file of our software prototype in IntelliJ, as shown in the attached figures, along with descriptions provided below.

How to build the executable jar file:

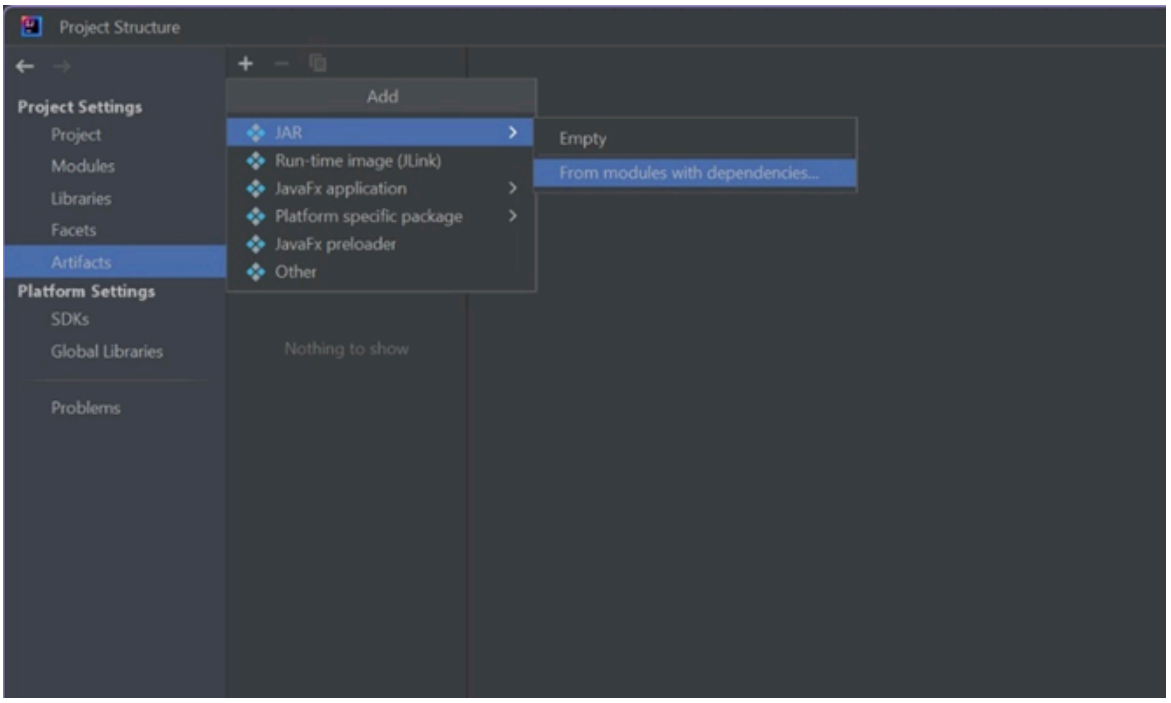
Before building the executable JAR file, we will create a package called 'resources' and mark this package as resources within the module in the project structure.



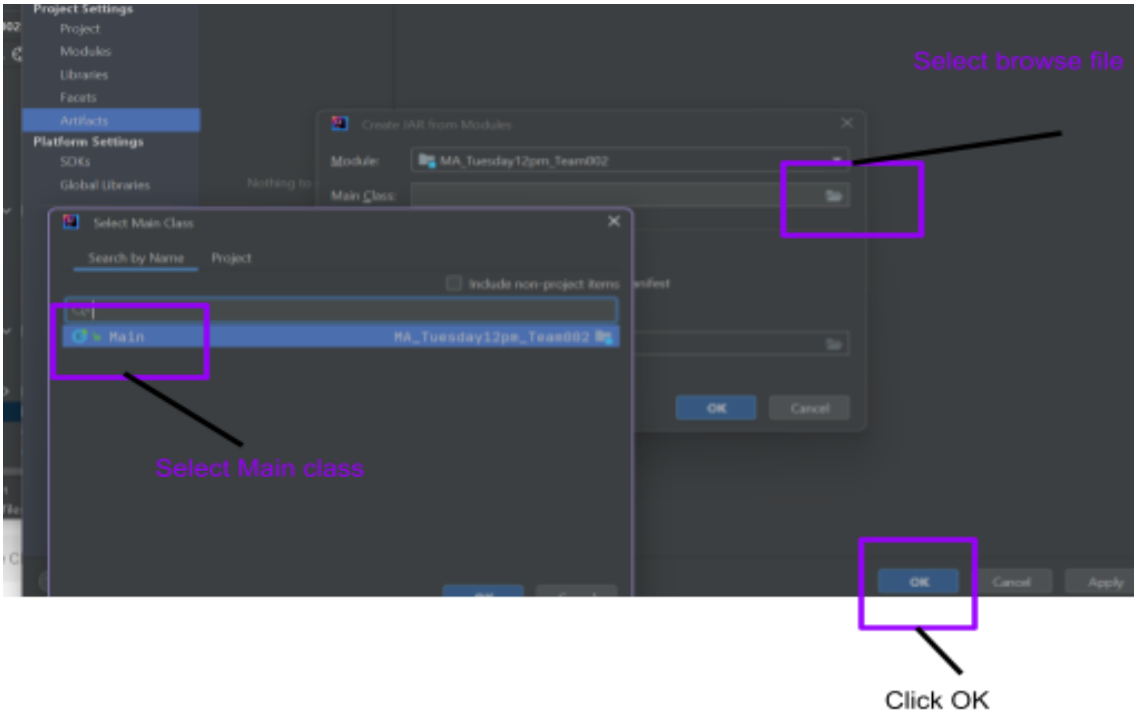
1. In the main menu, go to File | Project Structure and select Artifacts.



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

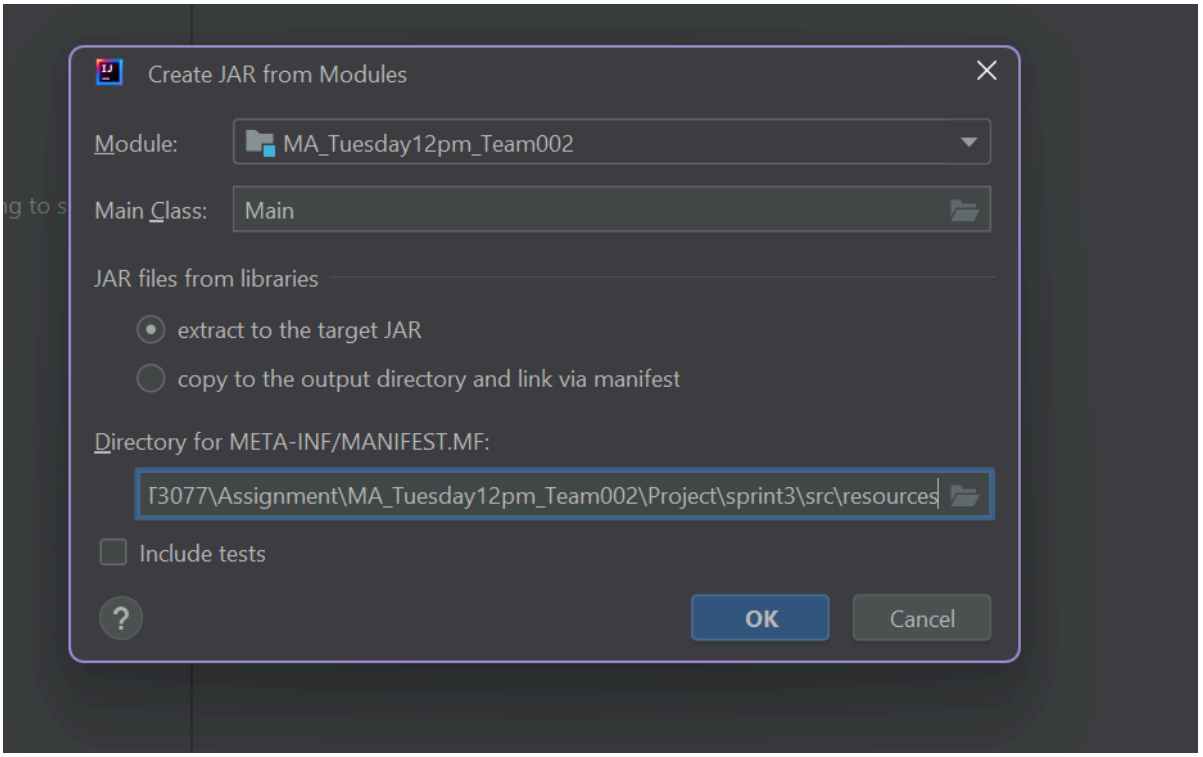


3. To the right of the Main Class field, click the Browse button and select Main class (main) in the dialog that opens.

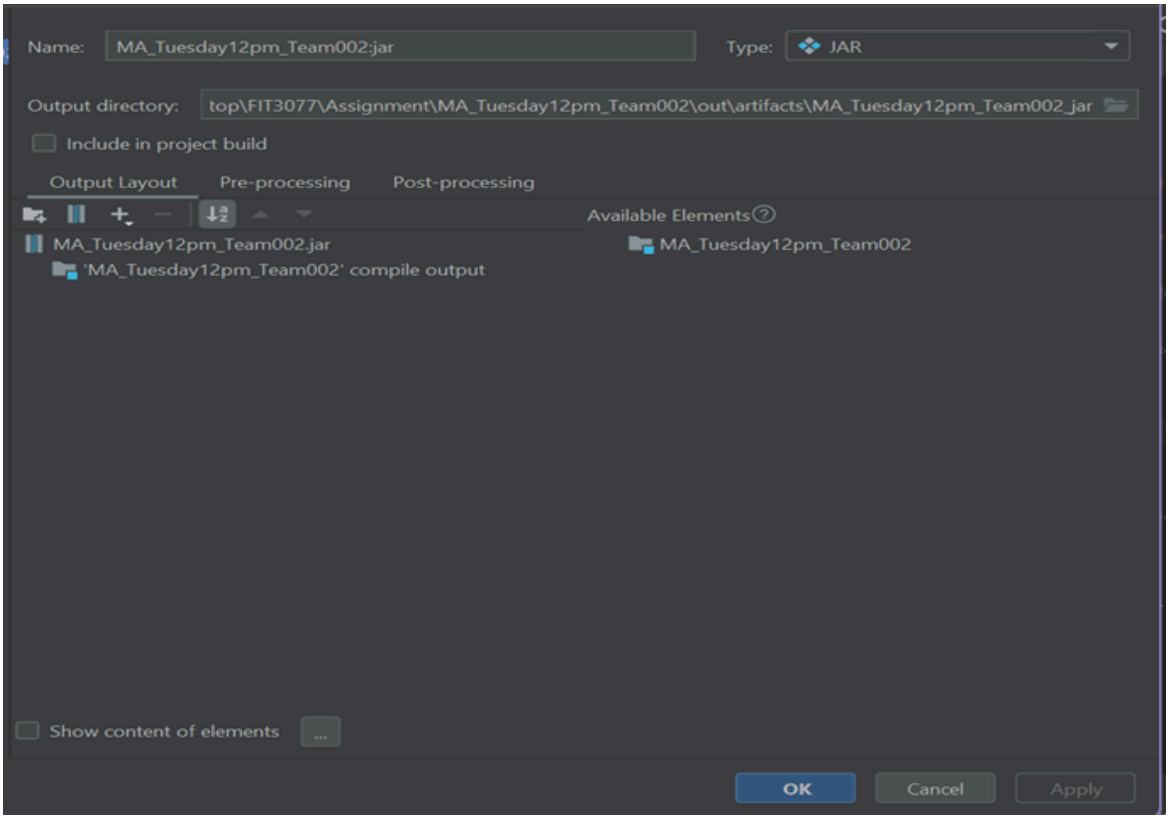


\

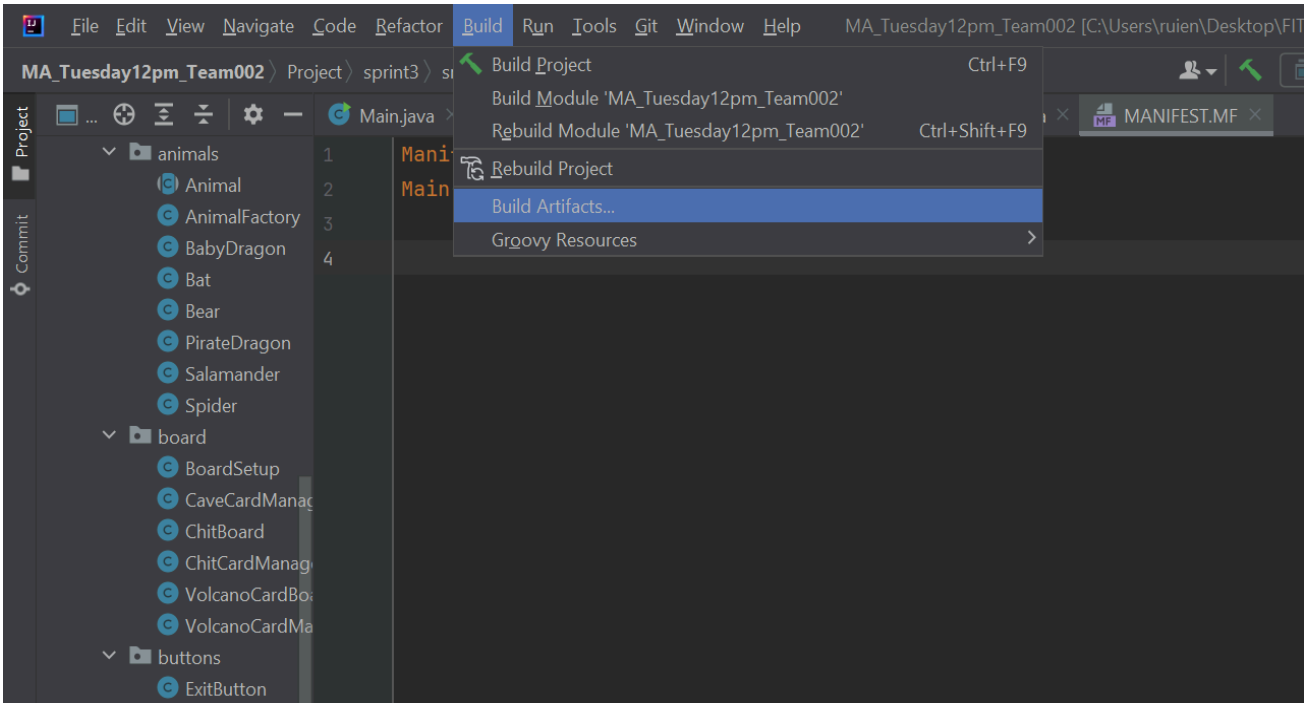
4. Change the directory for META-INF/MANIFEST.MF to src/resources in the Create JAR from Modules.



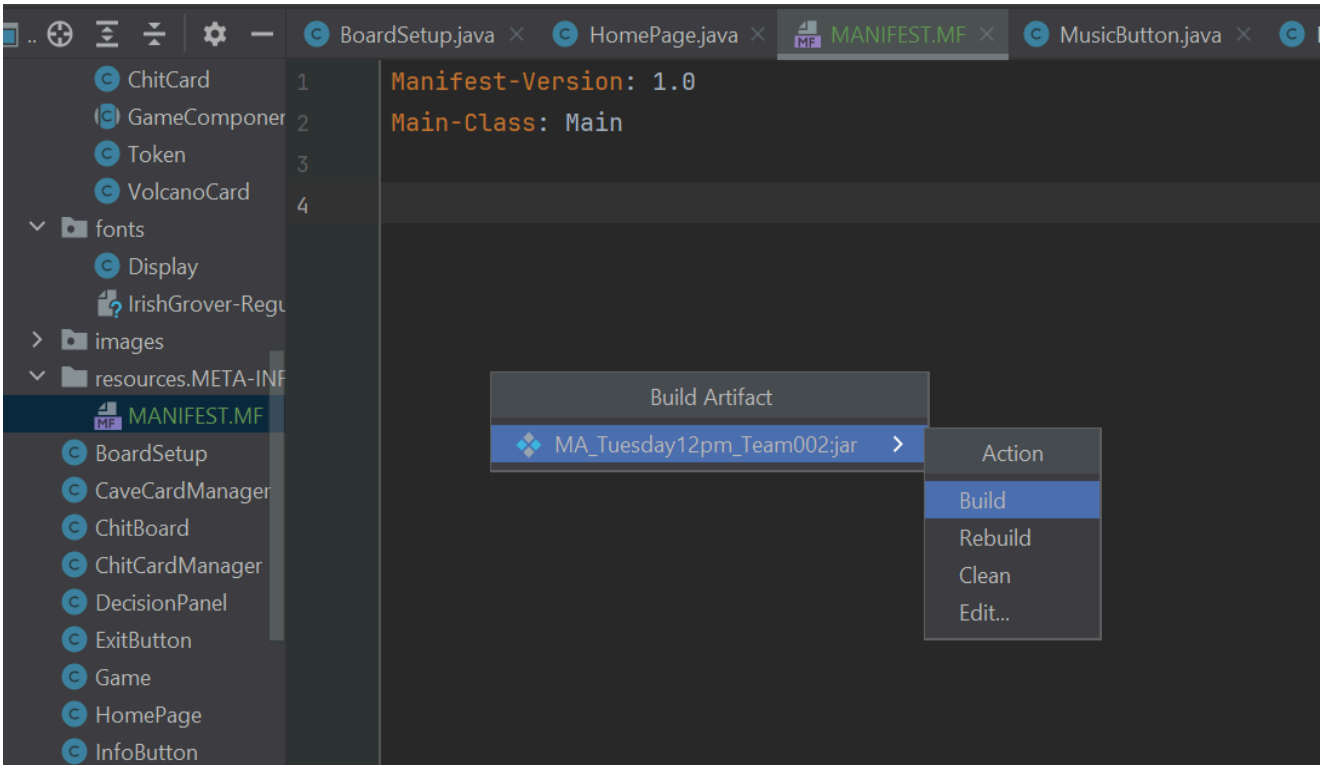
5. In the Artifacts section, select Apply and close the dialog.



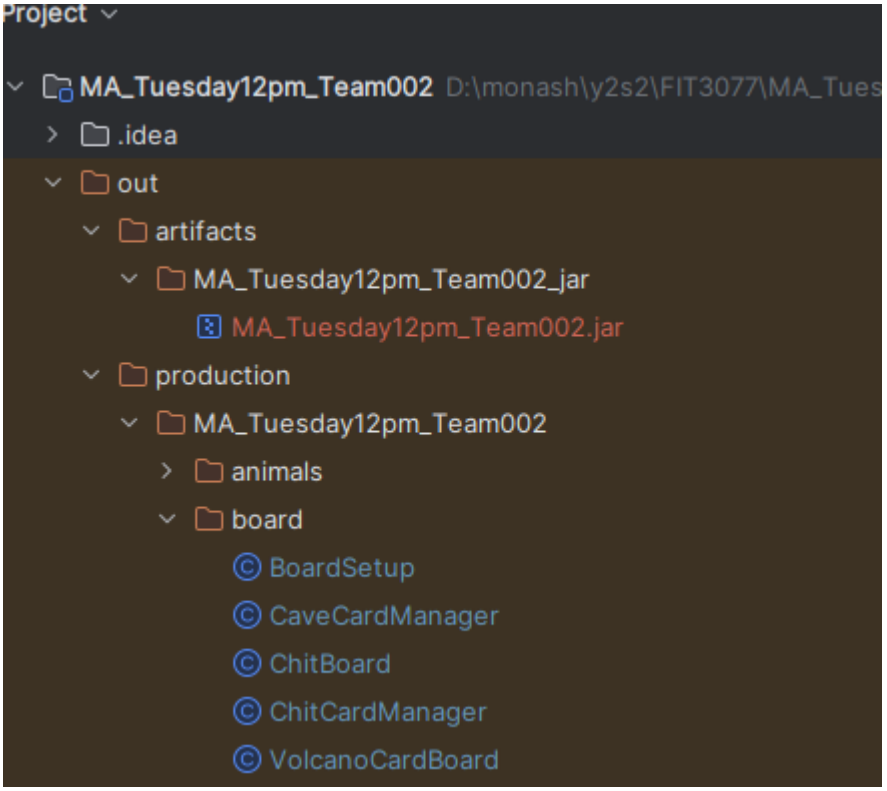
6. Then, in the main menu, go to Build | Build Artifacts.



7. Point to MA_Tuesday12pm_Team002:jar and select Build.



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



5. 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_Tuesday12pm_Team002

Contribution Logs:

FIT3077 - MA_Tuesday12pm_Team002 -Contribution Logs - Sprint 4

Wiki Section in GitLab Repository Link:

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

UML class diagram:

https://lucid.app/lucidchart/7468c9fe-a027-4a6c-ac38-72d87be469c5/edit?viewport_loc=-2574%2C-2351%2C6992%2C4997%2CJglHht-fxdBE&invitationId=inv_65731c31-0d76-445b-9130-16db24c09260

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 4.

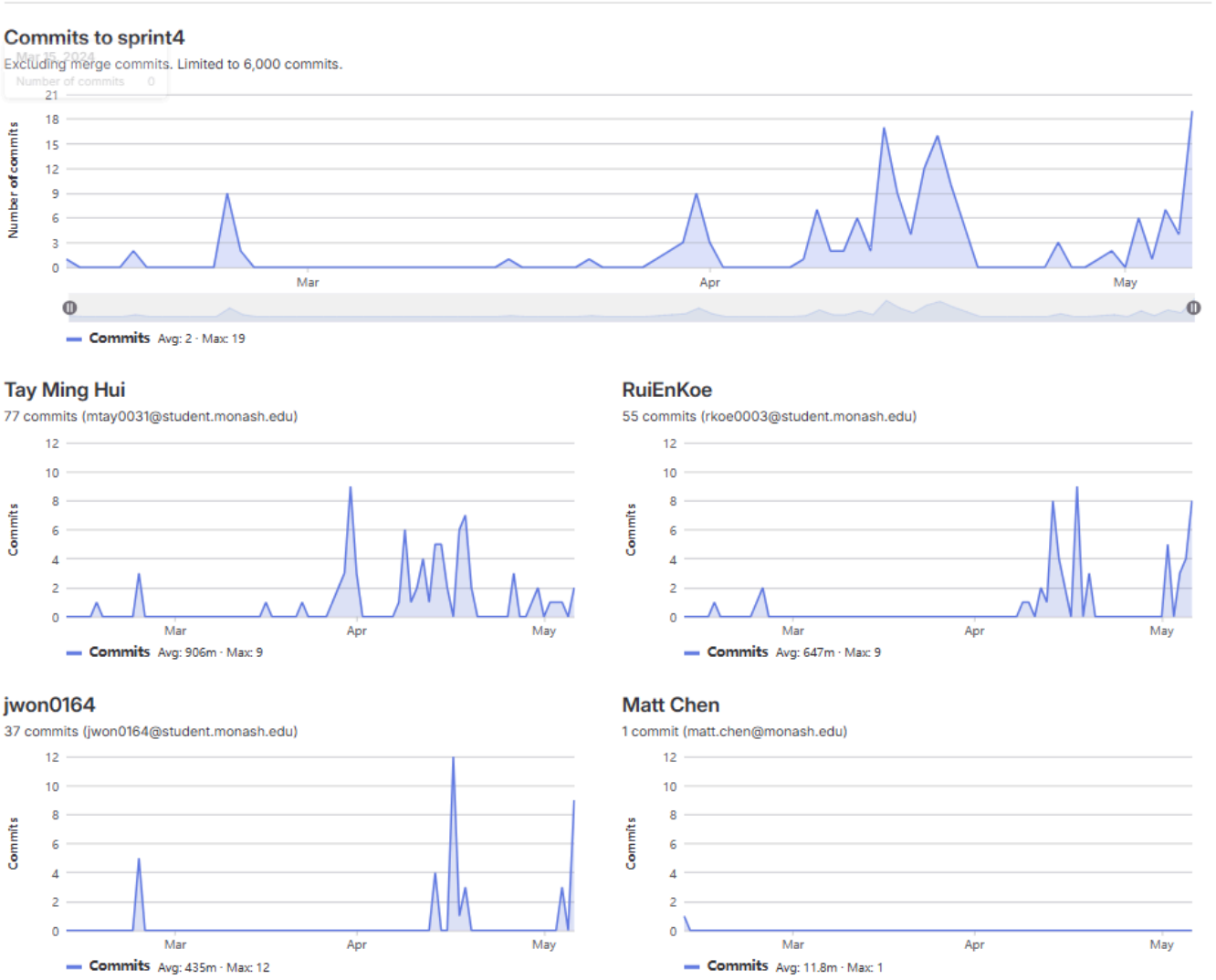
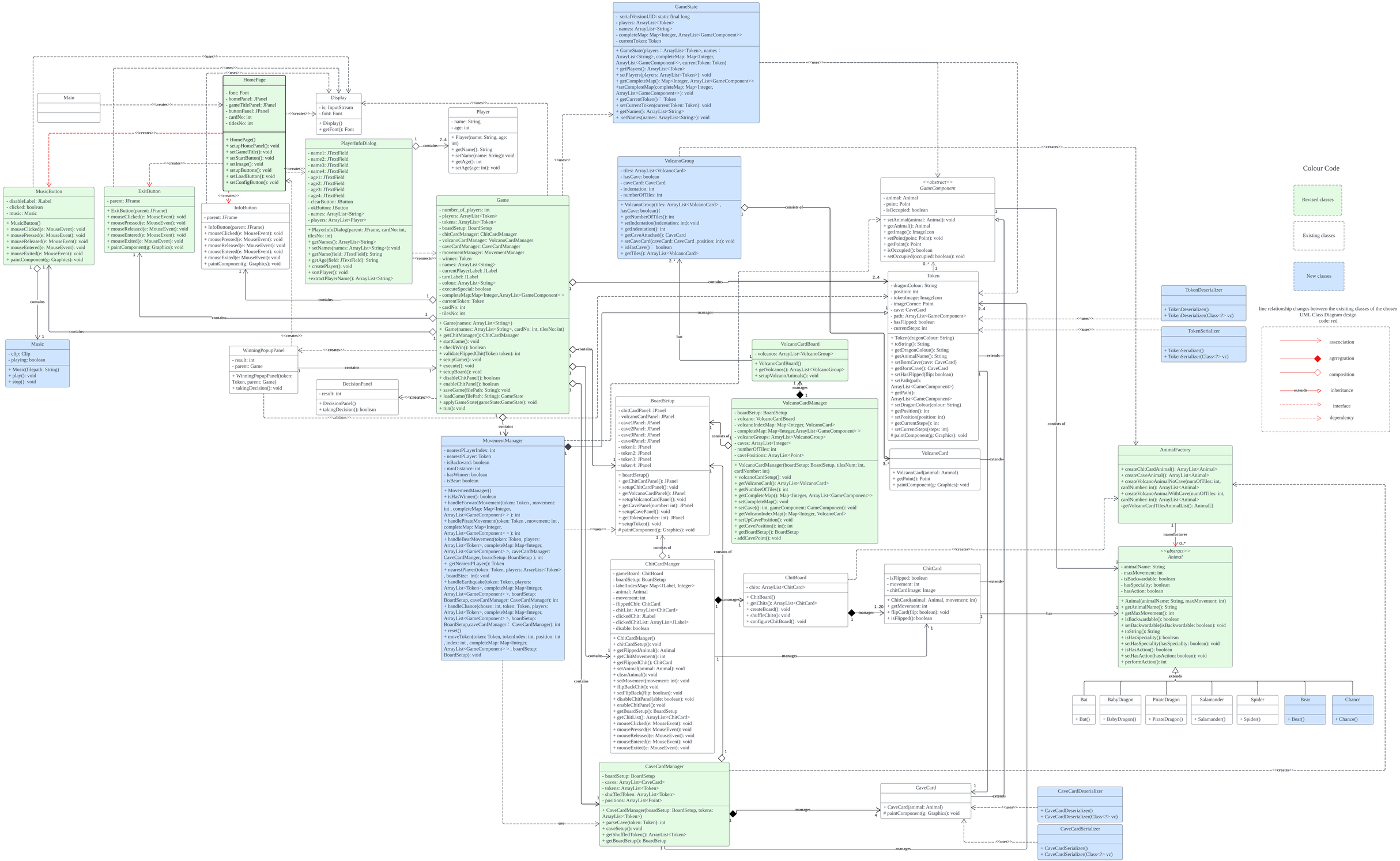


Figure 1: A screenshot of the Luminary Team’s “Contributor Analytics”

Appendix 3: 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
- cardNo: int
- tilesNo: int
+ HomePage()
+ setupHomePanel(): void
+ setGameTitle(): void
+ setStartButton(): void
+ setImage(): void
+ setupButtons(): void
+ setLoadButton(): void
+ setConfigButton(): void
Display
- is: InputStream
- font: Font
+ Display()
+ getFont(): Font
GameState
- serialVersionUID: static final long
- players: ArrayList<Token>
- names: ArrayList<String>
- completeMap: Map<Integer, ArrayList<GameComponent>>
- currentToken: Token
+ GameState(players : ArrayList<Token>, names : ArrayList<String>, completeMap: Map<Integer, ArrayList<GameComponent>>, currentToken: Token)
+ getPlayers(): ArrayList<Token>
+ setPlayers(players: ArrayList<Token>): void
+ getCompleteMap(): Map<Integer, ArrayList<GameComponent>>
+setCompleteMap(completeMap: Map<Integer, ArrayList<GameComponent>>): void
+ getCurrentToken() : Token
+ setCurrentToken(currentToken: Token): void
+ getNames(): ArrayList<String>
+ setNames(names: ArrayList<String>): void
Player
- name: String
- age: int
+ Player(name: String, age: int)
+ getName(): String
+ setName(name: String): void
+ getAge(): int
+ setAge(age: int): 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, cardNo: int, tilesNo: int)
+ getNames(): ArrayList<String>
+ setNames(names: ArrayList<String>): void
+ getName(field: JTextField): String
+ getAge(field: JTextField): String
+ createPlayer(): void
+ sortPlayer(): void
+extractPlayerName(): ArrayList<String>
MusicButton
- disableLabel: JLabel
- clicked: boolean
- music: Music
+ 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
Music
- clip: Clip
- playing: boolean
+ Music(filepath: String)
+ play(): void
+ stop(): void
WinningPopupPanel
- result: int
- parent: Game
+ WinningPopupPanel(token: Token, parent: Game)
+ takingDecision(): void
DecisionPanel
- result: int
+ DecisionPanel()
+ takingDecision(): boolean
Game
- number_of_players: int
- players: ArrayList<Token>
- tokens: ArrayList<Token>
- boardSetup: BoardSetup
- chitCardManager: ChitCardManager
- volcanoCardManager: VolcanoCardManager
- caveCardManager: CaveCardManager
- movementManager: MovementManager
- winner: Token
- names: ArrayList<String>
- currentPlayerLabel: JLabel
- turnLabel: JLabel
- colour: ArrayList<String>
- executeSpecial: boolean
- completeMap:Map<Integer,ArrayList<GameComponent> >
- currentToken: Token
- cardNo: int
- tilesNo: int
+ Game(names: ArrayList<String>)
+ Game(names: ArrayList<String>, cardNo: int, tilesNo: int)
+ getChitManager(): ChitCardManager
+ startGame(): void
+ checkWin(): boolean
+ validateFlippedChit(Token token): int
+ setupGame(): void
+ execute(): void
+ setupBoard(): void
+ disableChitPanel(): boolean
+ enableChitPanel(): boolean
+ saveGame(filePath: String): void
+ loadGame(filePath: String): GameState
+ applyGameState(gameState:GameState): void
+ run(): void
VolcanoGroup
- tiles: ArrayList<VolcanoCard>
- hasCave: boolean
- caveCard: CaveCard
- indentation: int
- numberOfTiles: int
+ VolcanoGroup(tiles: ArrayList<VolcanoCard> , hasCave: boolean){
+ getNumberOfTiles(): int
+ setIndentation(indentation: int): void
+ getIndentation(): int
+ getCaveAttached(): CaveCard
+ setCaveCard(caveCard: CaveCard ,position: int): void
+ isHasCave() : boolean
+ getTiles(): ArrayList<VolcanoCard>
<<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
TokenDeserializer
+ TokenDeserializer()
+ TokenDeserializer(Class<?> vc)
TokenSerializer
+ TokenSerializer()
+ TokenSerializer(Class<?> vc)
VolcanoCardBoard
- volcanos: ArrayList<VolcanoGroup>
+ VolcanoCardBoard()
+ getVolcanos(): ArrayList<VolcanoGroup>
+ 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>
- completeMap: Map<Integer,ArrayList<GameComponent> >
- volcanoGroups: ArrayList<VolcanoGroup>
- caves: ArrayList<Integer>
- numberOfTiles: int
- cavePositions: ArrayList<Point>
+ VolcanoCardManager(boardSetup: BoardSetup, tilesNum: int, cardNumber: int)
+ volcanoCardSetup(): void
+ getVolcanoCard(): ArrayList<VolcanoCard>
+ getNumberOfTiles(): int
+ getCompleteMap(): Map<Integer, ArrayList<GameComponent>>
+ setCompleteMap(): void
+ setCave((: int, gameComponent: GameComponent): void
+ getVolcanoIndexMap(): Map<Integer, VolcanoCard>
+ setUpCavePosition(): void
+ getCavePosition(i: int): int
+ getBoardSetup(): BoardSetup
- addCavePoint(): void
VolcanoCard
+ VolcanoCard(animal: Animal)
+ getPoint(): Point
+ paintComponent(g: Graphics): void
MovementManager
- nearestPLayerIndex: int
- nearestPLayer: Token
- isBackward: boolean
- minDistance: int
- hasWinner: boolean
- isBear: boolean
+ MovementManger()
+ isHasWinner(): boolean
+ handleForwardMovement(token: Token , movement: int , completeMap: Map<Integer, ArrayList<GameComponent> >): int
+ handlePirateMovement(token: Token , movement: int , completeMap: Map<Integer, ArrayList<GameComponent> >): int
+ handleBearMovement(token: Token, players: ArrayList<Token>, completeMap: Map<Integer, ArrayList<GameComponent> >, caveCardManager: CaveCardManger, boardSetup: BoardSetup): int
+ getNearestPLayer(): Token
+ nearestPlayer(token: Token, players: ArrayList<Token> , boardSize: int): void
+ handleEarthquake(token: Token, players: ArrayList<Token>, completeMap: Map<Integer, ArrayList<GameComponent> >, boardSetup: BoardSetup, caveCardManager: CaveCardManager): int
+ handleChance(chosen: int, token: Token, players: ArrayList<Token>, completeMap: Map<Integer, ArrayList<GameComponent> >, boardSetup: BoardSetup,caveCardManager : CaveCardManager): int
+ reset()
+ moveToken(token: Token, tokenIndex: int, position: int , index: int , completeMap: Map<Integer, ArrayList<GameComponent> > , boardSetup: BoardSetup): 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>
- positions: ArrayList<Point>
+ 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
CaveCardDeserializer
+ CaveCardDeserializer()
+ CaveCardDeserializer(Class<?> vc)
CaveCardSerializer
+ CaveCardSerializer()
+ CaveCardSerializer(Class<?> vc)
AnimalFactory
+ createChitCardAnimal(): ArrayList<Animal>
+ createCaveAnimal(): ArrayList<Animal>
+ createVolcanoAnimalNoCave(numOfTiles: int, cardNumber: int): ArrayList<Animal>
+ createVolcanoAnimalWithCave(numOfTiles: int, cardNumber: int): ArrayList<Animal>
-getVolcanoCardTilesAnimalList(): Animal[]
<<abstract>>
Animal
- animalName: String
- maxMovement: int
- isBackwardable: boolean
- hasSpeciality: boolean
- hasAction: boolean
+ Animal(animalName: String, maxMovement: int)
+ getAnimalName(): String
+ getMaxMovement(): int
+ isBackwardable(): boolean
+ setBackwardable(isBackwardable: boolean): void
+ toString(): String
+ isHasSpeciality(): boolean
+ setHasSpeciality(hasSpeciality: boolean): void
+ isHasAction(): boolean
+ setHasAction(hasAction: boolean): void
+ performAction(): int
Bat
+ Bat()
BabyDragon
+ BabyDragon()
PirateDragon
+ PirateDragon()
Salamander
+ Salamander()
Spider
+ Spider()
Bear
+ Bear()
Chance
+ Chance()
Colour Code
Revised classes
Existing classes
New classes
line relationship changes between the exisiting classes of the chosen UML Class Diagram design
code: red
association
aggregration
composition
extends
inheritance
interface
dependency