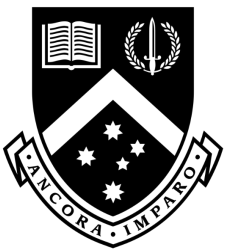


MONASH University

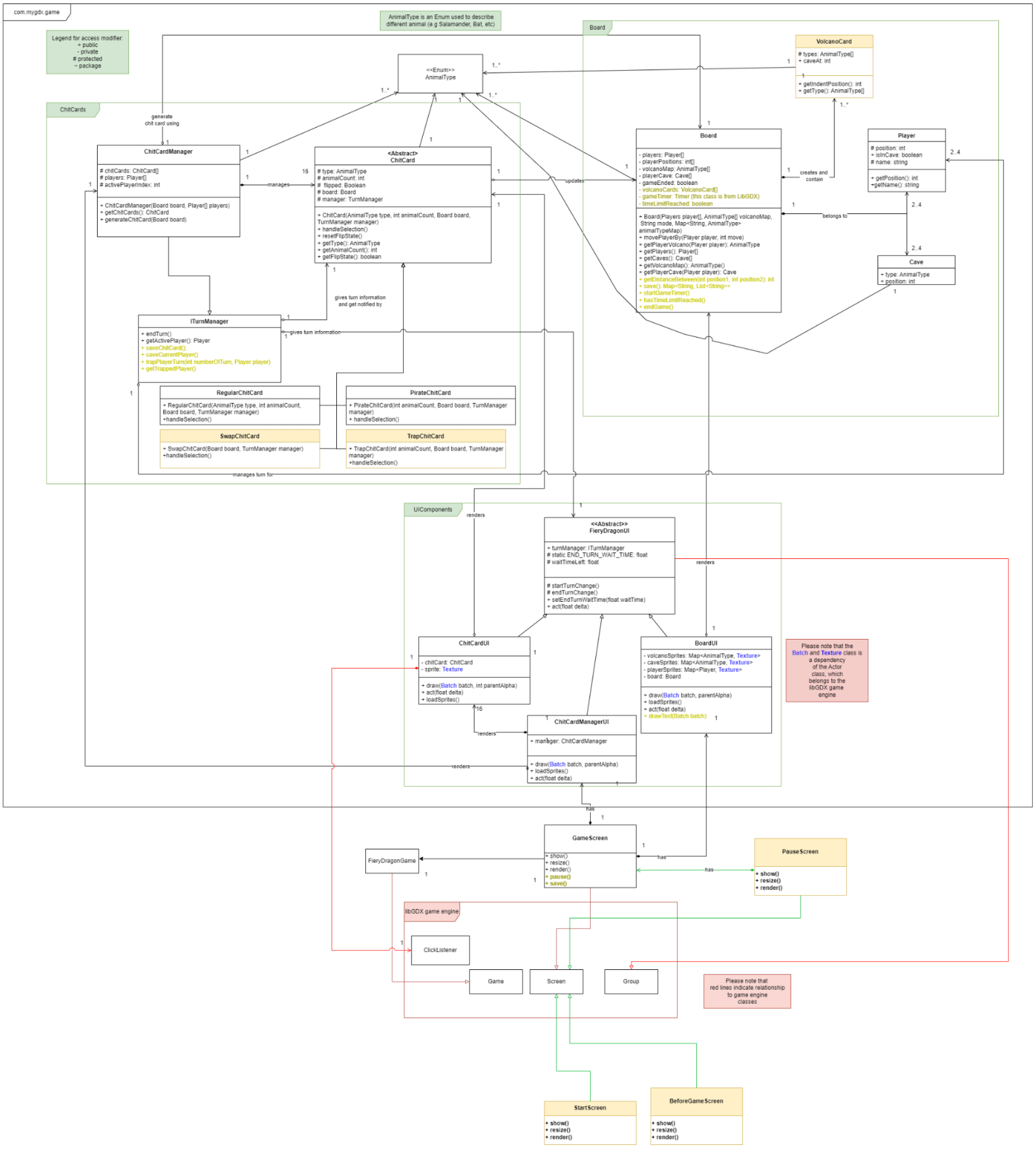
FIT3077 Sprint 4 (20%)

Team: CL_Monday06pm_Team377

1. Object-Oriented design

New addition and changes are highlighted in yellow

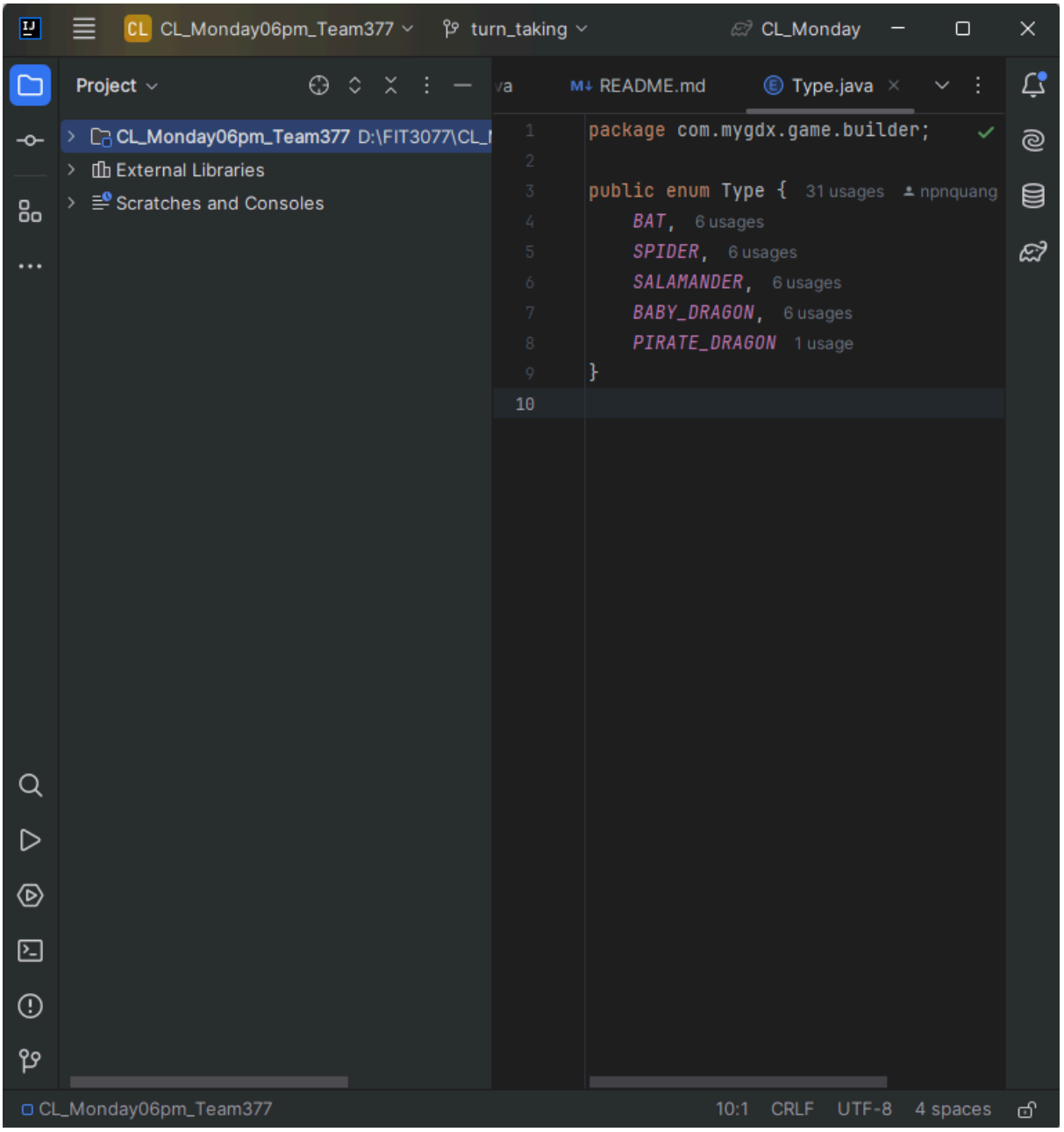
CLASS DIAGRAM



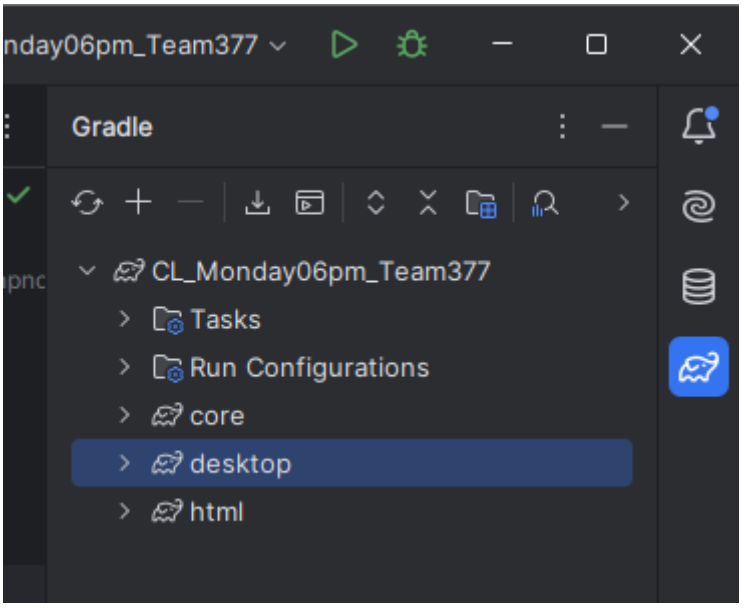
2. Tech-based software prototype

2.1. How to compile an executable

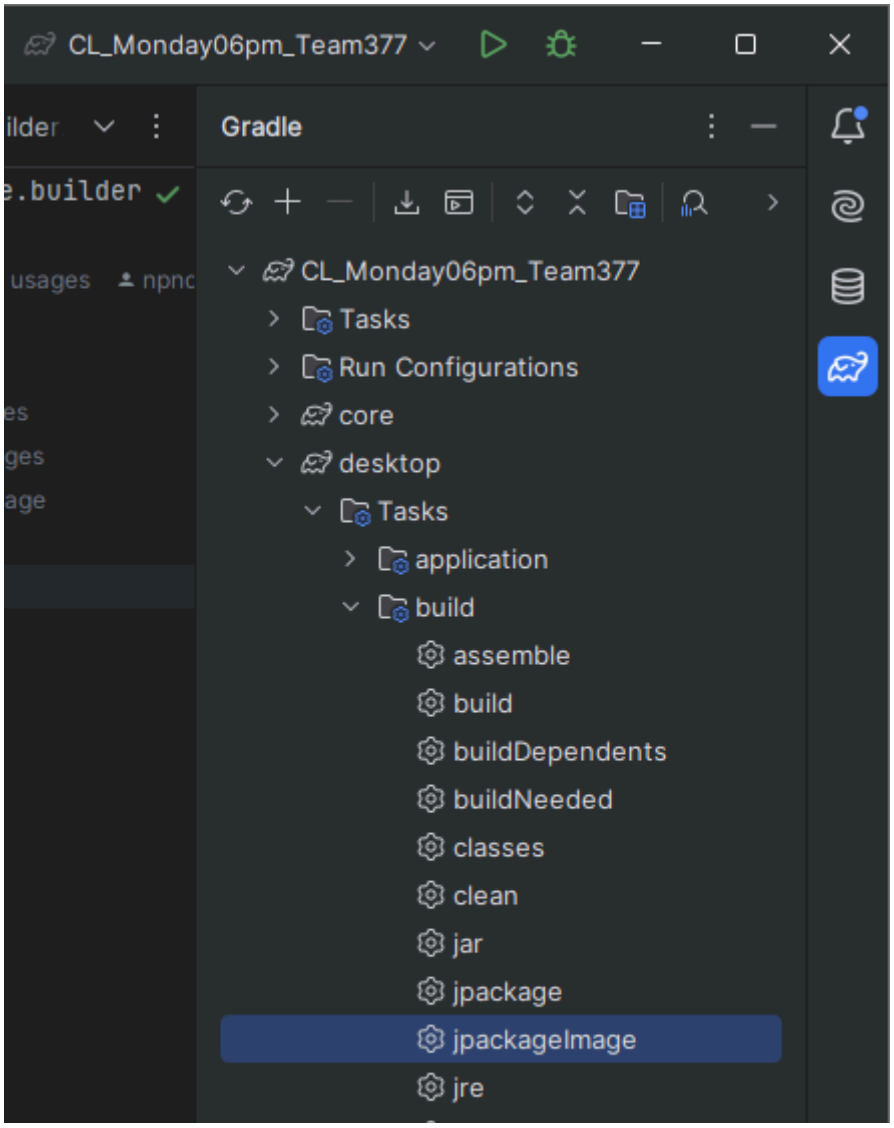
- Step 1: Open IntelliJ Idea software



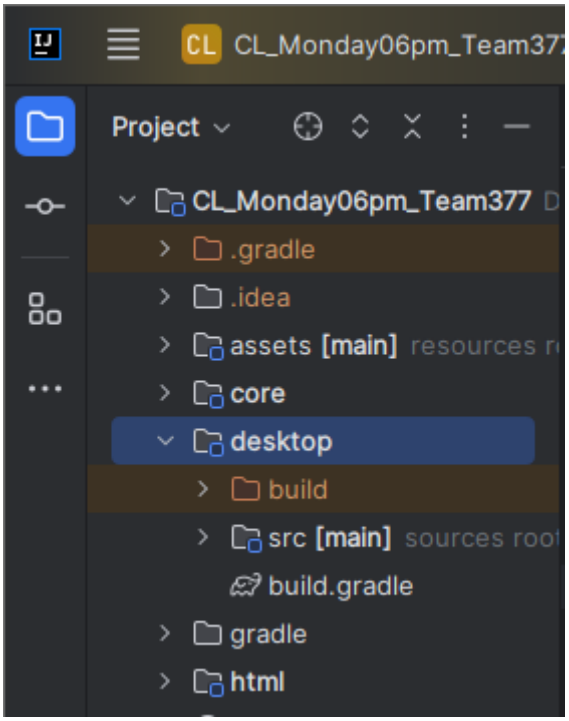
- Step 2: Go to Gradle panel on the right side bar



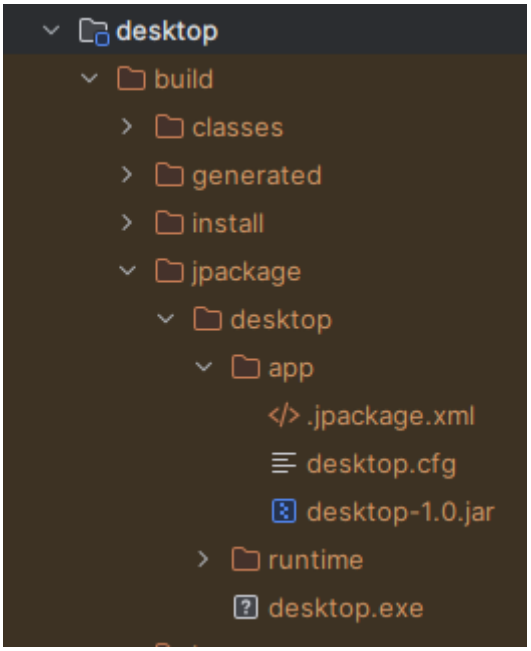
- Step 3: Go to **desktop** -> **Tasks** -> **build** and run jpackageImage task



- Step 4: After the task is finished, a **build** folder will be generated inside **desktop**



- Step 5: Go to **build** -> **jpackage** -> **desktop** -> **app**



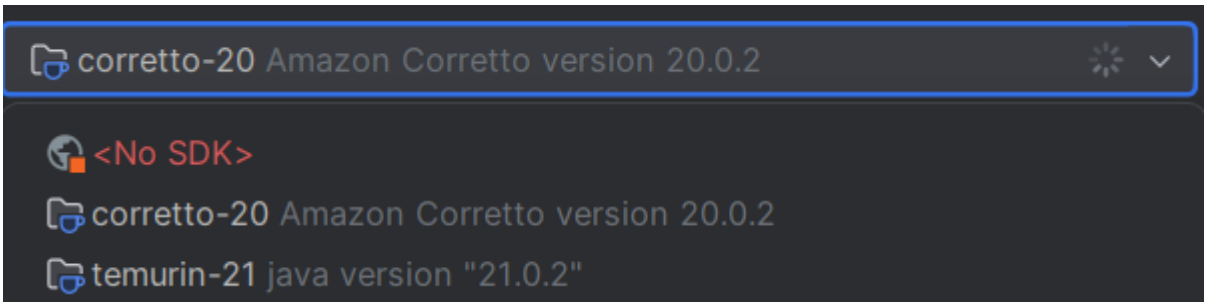
- Step 6: Run the **desktop-1.0.jar** file to launch the prototype

Note: Please see the following instructions to run the jar file

2.2. How to run the executable

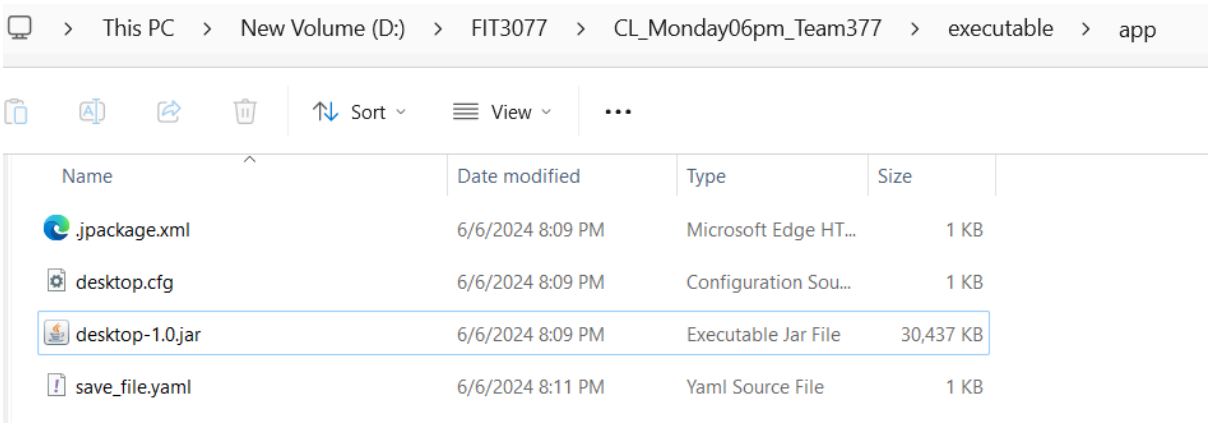
Here are the system requirements to run the game.

- SDK: either corretto-20 or temurin-21



- Java Virtual Machine (JVM):

```
C:\Users\Nhat Quang>java -version
java version "21.0.2" 2024-01-16 LTS
Java(TM) SE Runtime Environment (build 21.0.2+13-LTS-58)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.2+13-LTS-58, mixed mode, sharing)
```



Navigate to the executable folder, take the **save_file.yaml** from the **assets** folder and put it into the **executable/app** folder. Here you can modify the **save_file.yaml** file, to load your game configuration.

- **saved**: This is to indicate that if the previous game is saved. Set this to “true” if you want to load the game configuration from this yaml
- **load_option**: custom or default, this is handled by the game. It is set to “custom” if you choose to load your custom game configuration, “default” if you choose to load a default configuration. **You do not need to change this option**
- **chitCardNumber**: a list of numbers for chit cards in the game
- **chitCardFlipped**: a list of strings to indicate if the chit cards are currently flipped
- **chitCardType**: a list of types of chit cards
- **boardCustom**: a list of types of slot on the board, this can be of any length
- **currentPlayer**: the index of the player who is having their turn, zero-based. Maximum value is 3.
- **playerPositionCustom**: the position of the player on the board
- **playerDistanceFromCaveCustom**: the distance of each player from their cave, can be negative or positive.
- **playerNumber**: the number of players on the board, maximum is 4, minimum is 1
- **volcanoCardSize**: the size of all volcano cards on the board

- **cavePosition**: the positions of the caves, zero-based, ranges between 0 and length of the board minus 1

Next, click on the **desktop-1.0.jar** file to run the game.

3. Reflection on Sprint 3 Design

3.1. Required Extension 1: New Dragon Card 2

One of the required extensions is to implement New Dragon Card 1 or 2 as mentioned in the assignment specification, for this sprint, we decided to implement New Dragon Card 2, and we named it the Swap Chit Card in game. The Swap Chit Card functionality is to swap the position of the token belonging to the player who is playing the current turn with the another token closest to it on the board. For this extension, we needed to add extra functionality to the Board class to calculate distance between tokens, and a new ChitCard subclass to handle this new behaviour.

Since we anticipated that new Chit Card behaviours is something that was highly probable for Sprint 4, implementation of this extension was relatively straightforward and easy. In our sprint 3 design, the ChitCard class uses the Strategy design pattern, leaving behaviour on user selection highly customizable. Supporting functions such as calculating distance between tokens was built on top of existing methods. Overall, little couplings were encountered and a substantial amount of code was reused without needing modifications.

In conclusion, we found that our Sprint 3 design allows for heavy customization and extension of Chit Card behaviours. And this highlights good adherences to the SOLID principles, especially Liskov Substitution Principle. This design is a good exercise in good use of the Strategy pattern and how it can help with extensibility.

3.2. Required Extension 2: Loading and saving

In Sprint 4, we decided to add two new screens to the which were “PauseScreen” and “StartScreen” and modified the “GameScreen” by adding the dispose function and “BeforeGameScreen” (we changed its name from “MenuScreen” to “BeforeGameScreen”). It was not difficult to extend the screen since LibGDX provided the package for the screen. We simply extended the “Screen” class in the LibGDX package and it didn't require any additional classes creation.

The design of our system supports the extension of saving and loading in this sprint. In general, the amount of changes to the classes was not large. Most of the changes were made to the game components Board, ChitCardManager and GameScreen. The changes were simply to enable the game components to parse the information from the save file as well as encode their information into the yaml file for later loading. From the previous sprint, the class Board and ChitCardManager already had their lists of elements stored, and retrieving such information to put into the save file only requires a for loop. For loading functionality, the construction of the Board

and ChitCard does require some hardcode. However, I would not consider them as code smell since those are required code to parse the text from the save file. In addition, the information of the whole board is concentrated in two classes, Board and ChitCardManage. Thus, the changes introduced are concentrated in some specific classes, rather than being spread out. From my point of view, it was quite a manageable task in this sprint.

3.3. Self-defined extension 1: Trap Chit Card

Taking inspiration from many other board games (i.e Jail in Monopoly, Skip Cards from Uno and Exploding Kittens, etc), we want to implement a new game rule that would skip a player's turn if the user selects a Trap Chit Card, this would raise the stake when the user try and select. This extension aims to provide the human value of **Stimulation** by forcing the player to be daring. This extension requires some additional functionality for the class that manages turn, in this case it relates to the ChitCardManager class and ITurnManager interface, additionally we also need to create a subclass of Chit Card to accommodate this new behaviour/strategy.
Similar to the Swap Chit Card, our design in Sprint 3 made it particularly easy to implement this new chit card behaviour due the use of the Strategy design pattern. However, a challenge arose when adding the supporting functions in the Chit Card Manager class, as some modifications to existing methods were needed to support the complex logical and rendering behaviour of the Trap Chit Card. This somewhat violated the Open-Close Principle and was proof of slight coupling, an alternative approach we could've made was to create a subclass of ChitCardManager or a new class that implemented the ITurnManager interface, this would resolve the issue we had, at the expense of time, something that we didn't have a lot of.
The challenges we encountered gave us some insight into technical debt and the project-related tradeoffs we need to make when designing. If we can redesign our Sprint 3, we would have made Chit Card Manager class be able to cope with different turn-ending behaviour. But more generally, we should've managed our time better and allocated more time brainstorming the possible changes for this class. In hindsight, the design for this class was rushed due to limited time.

3.4. Self-defined extension 2: Human value, Healthy - Limited play time

In regards to this extension, I could take advantage of the codebase to easily implement this feature to limit the playtime of players. The existing system has the class 'Board' to handle every action related to the Board including moving players or checking whether the game has ended or not. Hence, I made use of the 'gameEnded' attribute to implement the 'endGame' function which is called when the timer has reached the time limit. Another advantage about this system when I implemented my extension was we created the 'BoardUI' class which had the

responsibility to draw the board and any initial messages to players. As my extension was not allowed to kick players out of the game when time ended, but just send out messages to warn them instead. This ‘BoardUI’ class was perfect to return a message whenever a player has reached their time limit and another message at the initial stage of the game for the player to notice about the limited playtime.

3.5. Designing/Implementing an explicit Volcano Card class

3.5.1. Overview

This change posed the greatest challenge for our team during Sprint 4 as it directly clashed with one of our assumptions in Sprint 3, that volcano cards are a physical aspect of the game that did not need to be explicitly implemented. Another reason was because all of our other extensions did not make use of this new explicit Volcano Card, so we found it hard to motivate any of our design changes. In the end, due to a lack of time, we settled with just fulfilling the functional specifications required by the assignment and didn’t make use of the new VolcanoCard class other than an intermediate step when arranging the board. In order to make our game able to interpret this new format, the best practice was to make use of the Adapter design pattern to translate the Volcano Cards into a tile array that our existing board can understand, the process was relatively straightforward the existing Board logic does allow for flexibility in board length and layout, and other assumption about the board holding in this sprint. We however didn’t create a separate class for this adapter but consolidated the logic within Board, this is obviously a bad practice that can lead to a bloated class that violates SRP, but given the limited time we had, we cannot afford extra complexity that can result in regression and errors for a feature that we barely used, so we have prioritise getting it done over getting it perfect. The bulk of the challenges lies in the changing the UI, which we go into detail in the following section.

Overall, the reflection in this section revealed our biggest oversight when designing Sprint 3, is that we were geared toward predicting possible change and preparing for that, when what we should’ve done was make our design as robust as possible to cope with any change, because in real life, changes are unpredictable.

3.5.2. Issues with existing logic

Sprint 4 implementation witnessed the expansion of the board’s components such as volcano card and piece’s cave. In the Sprint 3 implementation, we focused on developing the constant amount of volcano cards and distance between caves. There were some additional hard-coded variables and code smell in the board UI design. This means that, whenever the amount of the card changed, the cave distance would not be constant and it can result in failure to display the cave on the board in the border around the board game. While expanding the cave distance, we found it challenging since we had to modify the “Board” class to edit the board from

equal distance in the range of 24 volcano cards to any random position in the game. Fortunately, we managed to successfully meet the extension and published it on Git. If we were able to go back to sprint 3 again, we would implement the code so that the cave could be allocated anywhere around the board.

4. Video presentation

Drive Link:
https://drive.google.com/file/d/1XA3_n1z_Df9iFCJLUmq_iZPAjWBYRbDS/view?usp=drive_link

Youtube link: https://youtu.be/2w-YcGxQ23Y