Nine Men's Morris Project Report - Sprint3

By Yuxiang Feng(32431813), Yifan Wang(32459238), Tianyi Liu(27936619)

Team Name: **G**enerative **A**dversarial **N**etwork (short: GAN)

0. Project Package Tree

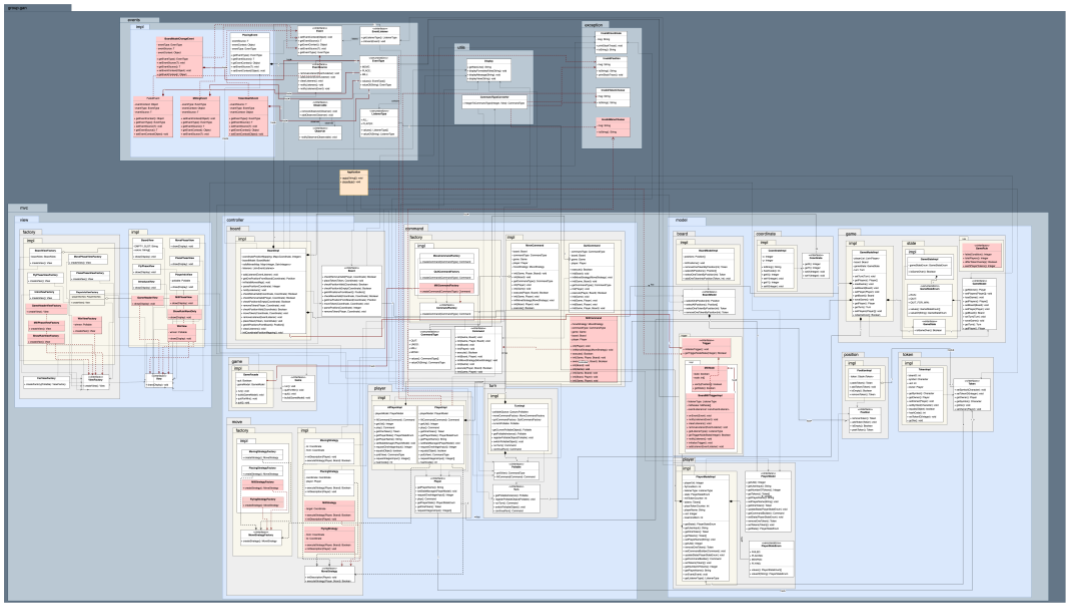
```
.
├── main
│   ├── java
│   │   └── group
│   │   └── gan
│   │   ├── events
│   │   │   └── impl
```

```
│   ├── exception
│   ├── mvc
│   │   ├── controller
│   │   │   ├── board
│   │   │   │   └── impl
│   │   │   ├── command
│   │   │   │   ├── factory
│   │   │   │   │   └── impl
│   │   │   │   └── impl
│   │   │   ├── game
│   │   │   │   └── impl
│   │   │   ├── move
│   │   │   │   ├── factory
│   │   │   │   │   └── impl
│   │   │   │   └── impl
│   │   │   ├── player
│   │   │   │   └── impl
│   │   │   └── turn
│   │   │       └── impl
│   │   ├── model
│   │   │   ├── board
│   │   │   │   ├── impl
│   │   │   │   ├── trigger
│   │   │   │   └── impl
│   │   │   ├── coordinate
│   │   │   │   └── impl
│   │   │   ├── game
│   │   │   │   ├── impl
│   │   │   │   ├── rule
│   │   │   │   ├── state
│   │   │   │   └── impl
│   │   │   ├── player
│   │   │   │   └── impl
│   │   │   ├── position
│   │   │   │   └── impl
│   │   │   ├── token
│   │   │   │   └── impl
│   │   │   └── view
│   │   ├── factory
│   │   │   └── impl
│   │   └── impl
│   └── utils
└── resources

└── test
```

The whole project adopts `full interface` development and `MVC` architecture mode, and uses `JUnit` as a unit testing tool. Due to the need for full interface development, we use the naming method commonly used in the industry. A folder named `impl` is at the same level as its interface file. The implementation class of the interface of the same level is stored in the `impl` folder.

1. Class Diagram

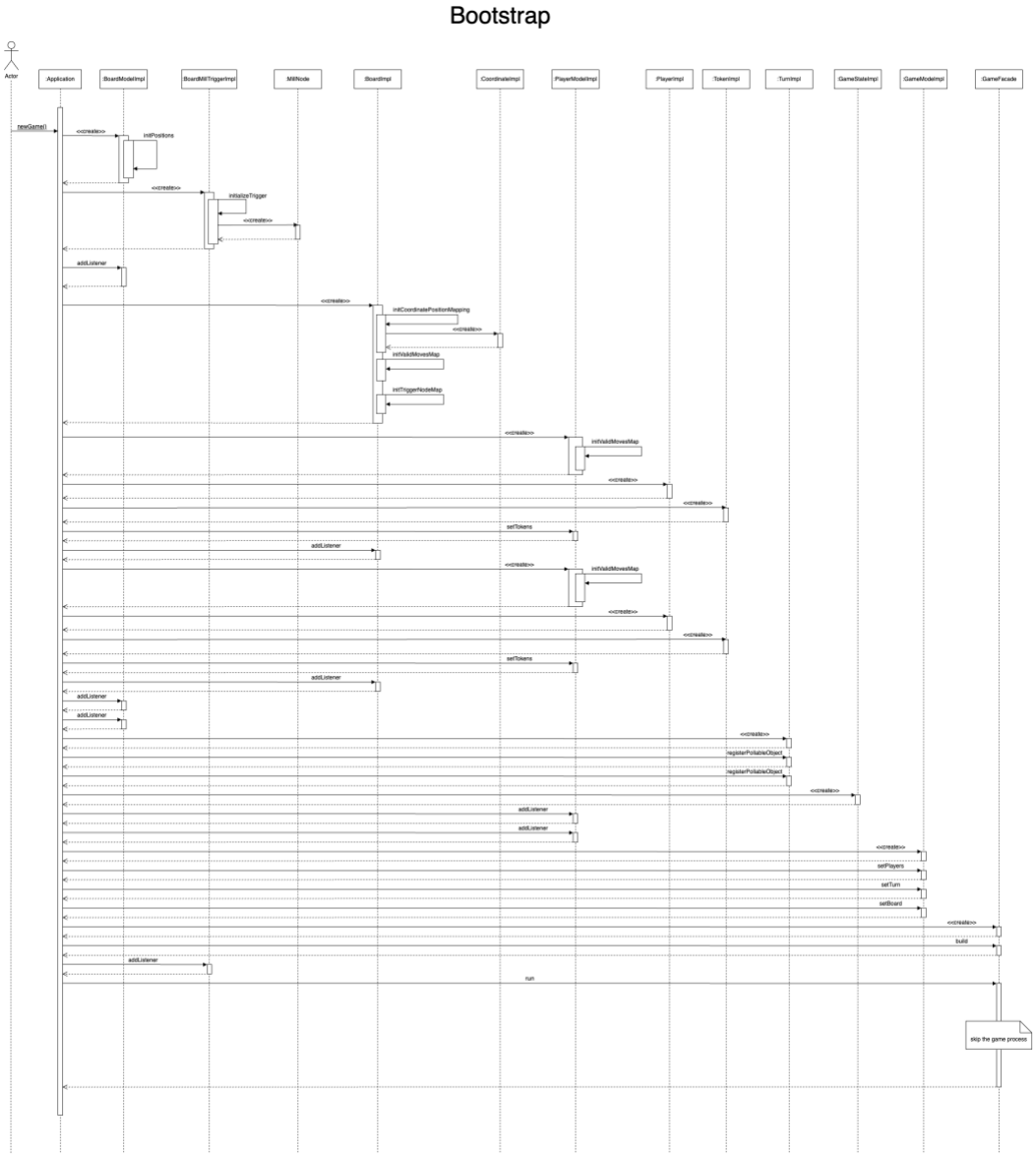


Full Size Link:
https://drive.google.com/file/d/1Ozh4pBfu2e_btftUhGYjQlTs9ed1V8i9/view?usp=sharing

In the Class Diagram, the parts that have been modified compared with the last time are marked in red. Application class is the entry point of the whole project. The most important change in Class Diagram is that we have added a trigger about mill. Because there are some differences between Mill and BoardModel. Therefore, Mill data will not be saved in the BoardModel class, but will be stored and verified by MillTrigger.

2. Sequence Diagram

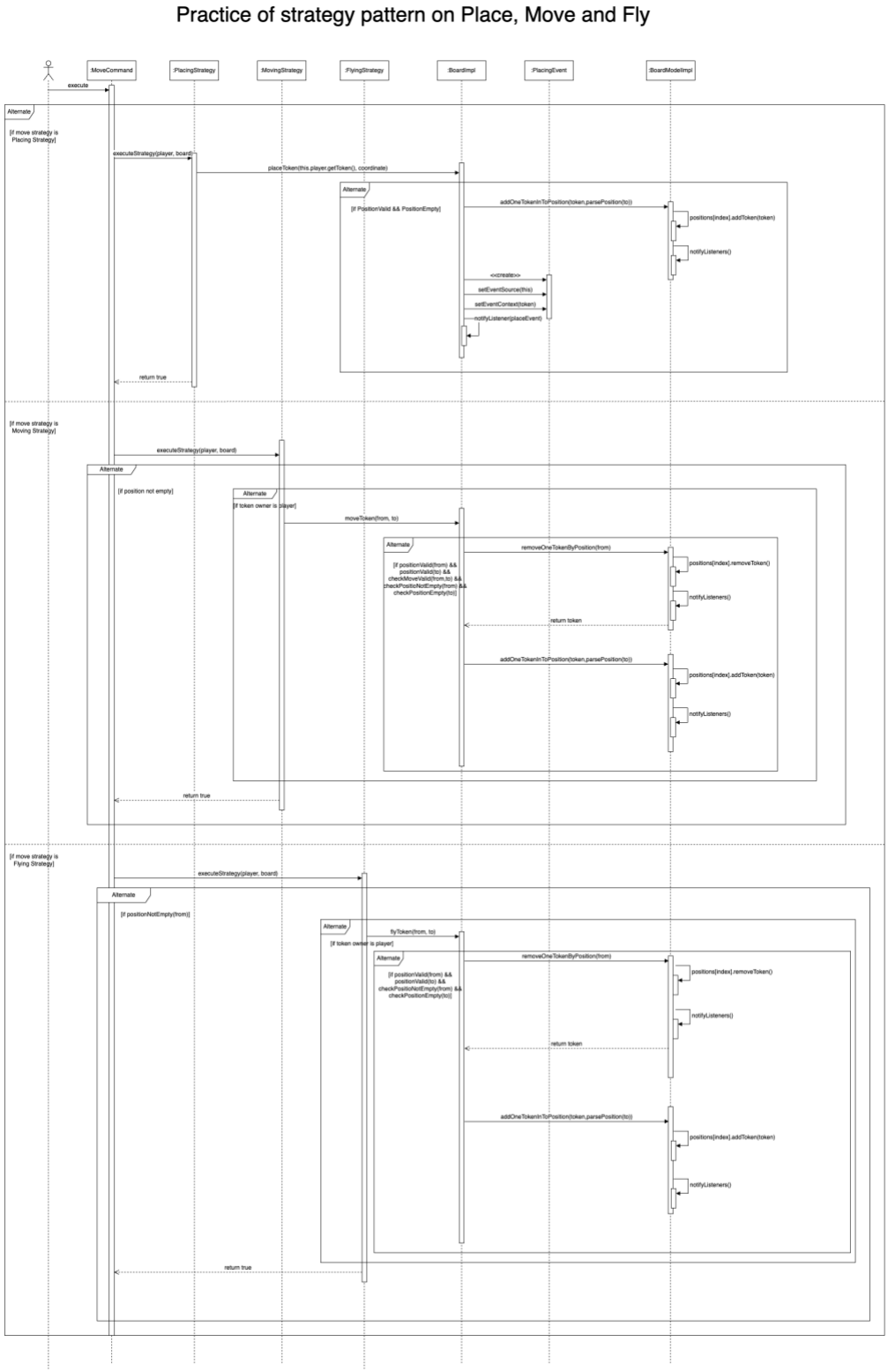
2.1 Bootstrap



Full Size Link:
https://drive.google.com/file/d/15KXWDGG9Du1hWj2W7ls1Uf1ooewCYp-V/view?usp=sharing

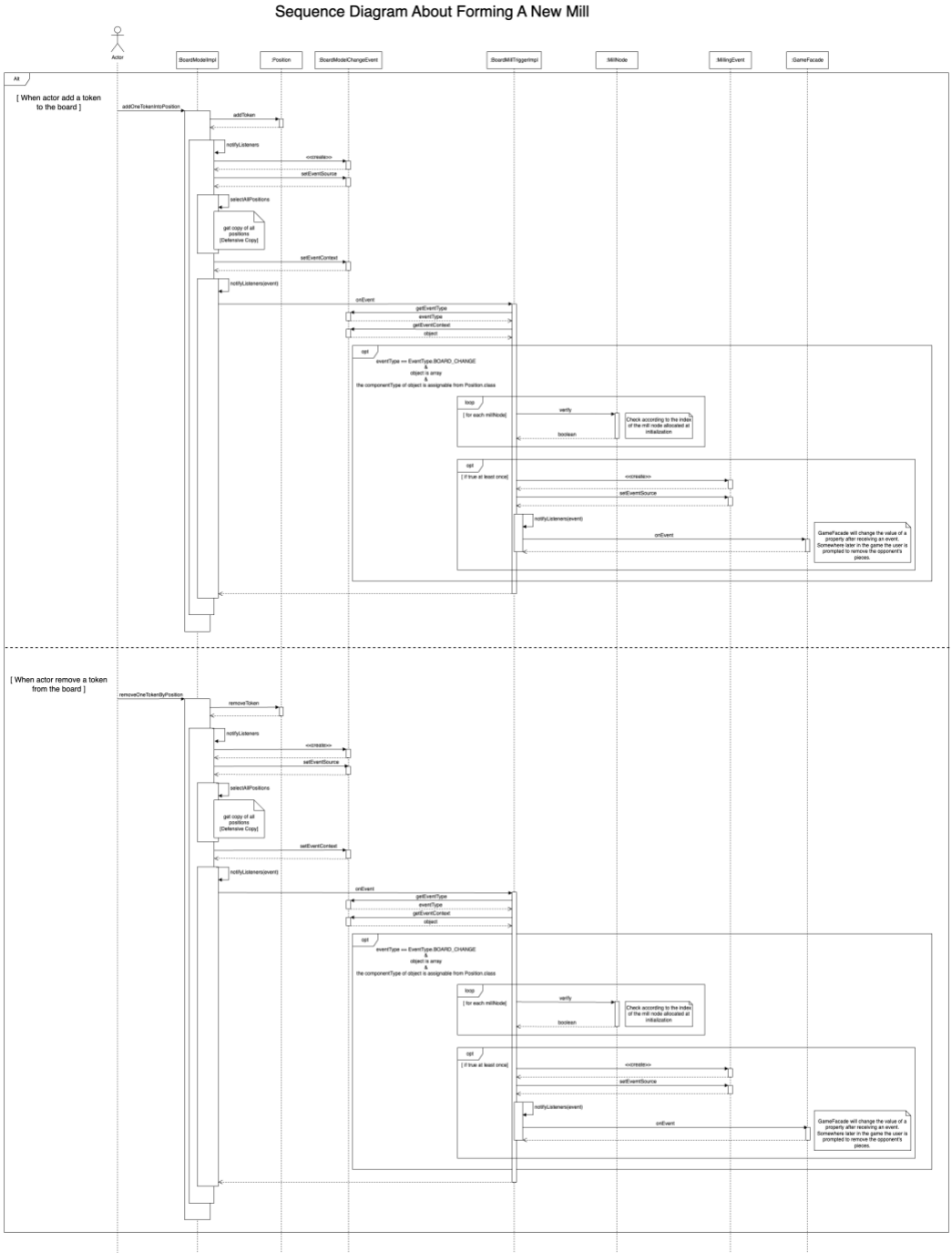
2.2 Non-trivial Interaction

2.2.1 Practice of strategy pattern on Place, Move and Fly



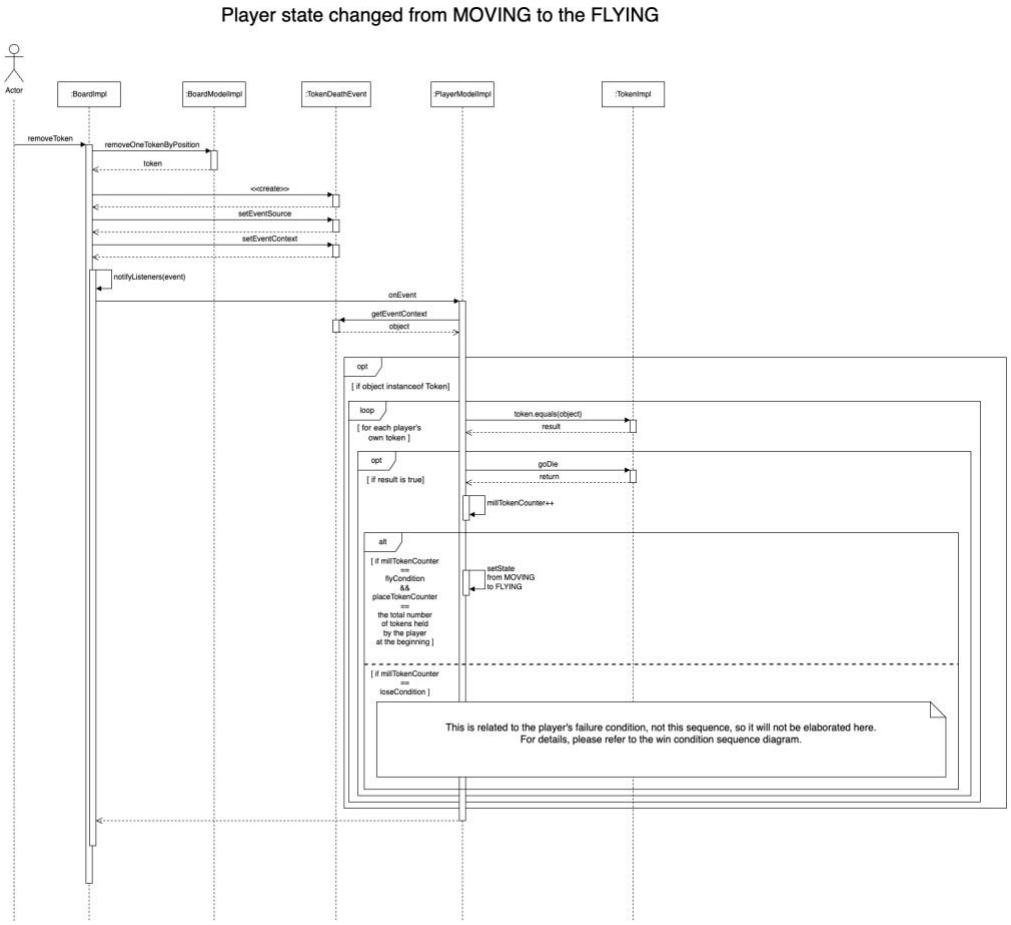
Full Size Link:
https://drive.google.com/file/d/1xrWJMlbNErIpS1rL30FHBTSBj3jThsOk/view?usp=sharing

2.2.2 Sequence Diagram About Forming A New Mill



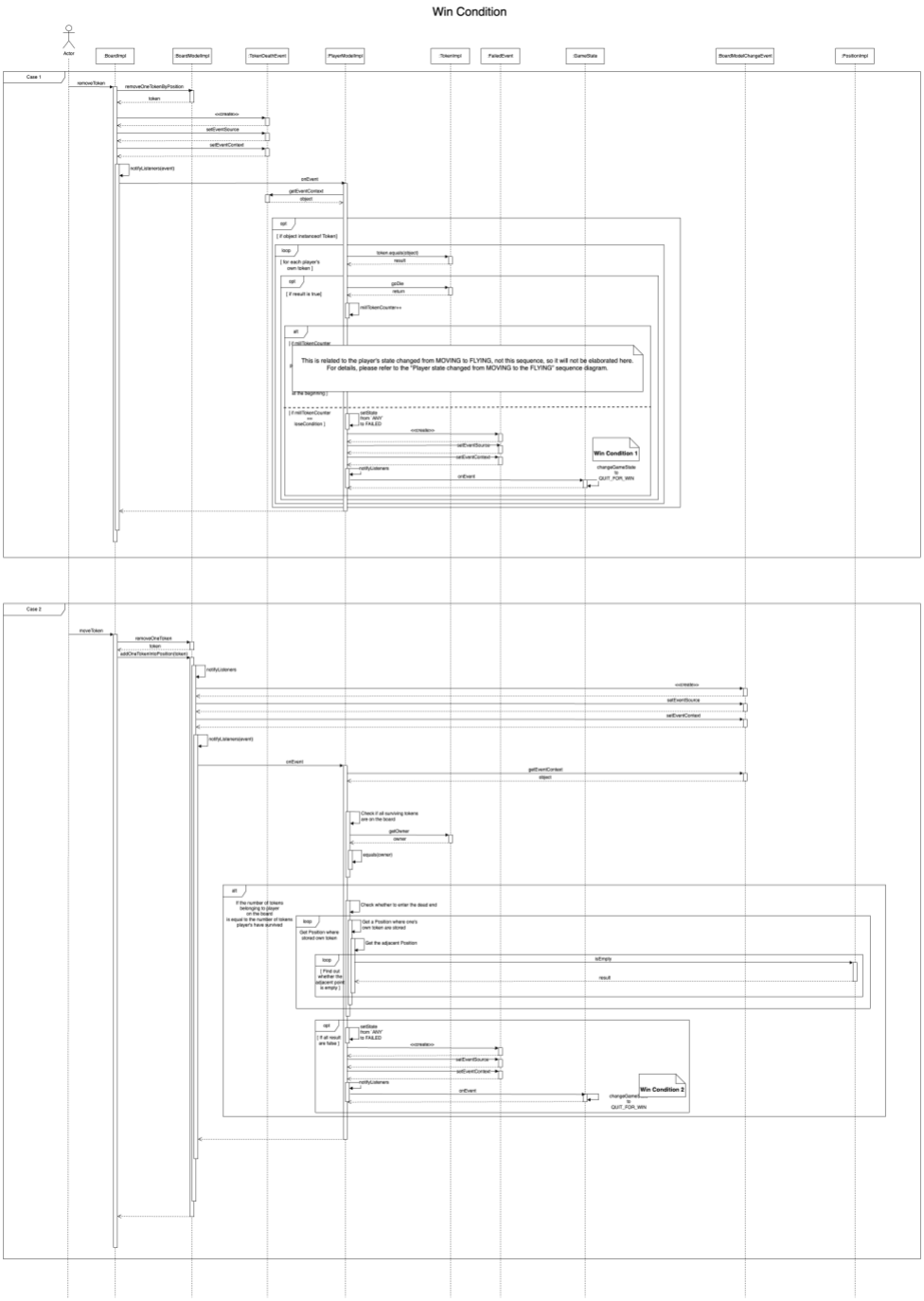
Full Size Link:
https://drive.google.com/file/d/1ByCtENtjAaKDg8q8gIU_UIIYdzIPuwWJ/view?usp=sharing

2.2.3 Player state changed from MOVING to the FLYING



Full Size Link:
https://drive.google.com/file/d/1gdkCu0vJ23F9sw1naOp6GwkT4HIQbEGq/view?usp=sharing

2.2.4 Win Condition



Full Size Link:
https://drive.google.com/file/d/11O7fIxyMBvvneVxtl3JjTYMxsiYpysgK/view?usp=sharing

3. Design Rationale

3.1 Explain why you have revised the architecture

Since we used the MVC architecture pattern and the full interface development pattern in Sprint2. So we didn't make any changes to the architecture in Sprint3. We will continue to use the MVC architecture pattern and full interface development. This is because we believe that the MVC architectural pattern and full interface development have provided sufficient scalability and modularization, and successfully achieved separation of concerns. Fully meet the design principles and apply multiple design patterns (such as factory pattern, strategy pattern, command pattern, state pattern, listener and facade pattern) in the development process. To sum up, we did not make any changes to the architecture in Sprint3.

3.2 Explain quality attributes. Why are they relevant and important to your game?

3.2.1 Maintainability

We have adopted the development method combining MVC architecture and full interface development. This makes the game system modular. At the same time, we write corresponding abstraction layer interfaces for different functional modules. After the interface is designed, the entire game system will be highly abstract. The implementation details of each different functional module will be completely hidden. This allows the coupling of the entire game system to be fully reduced.

When we reduce the degree of coupling to this level, the superiority of maintainability will be highlighted. Because the specific details of each class are hidden. Each class (including its concrete concrete class) will only communicate through the methods provided by the interface. This means that a unified constraint will be formed between each class. Problems with any one module will be caught quickly.

For example, when I modify the `initCoordinatePositionMapping()` function in the Board Controller. I accidentally put

```
int x = Integer. parseInt(coordinateArray[0].trim());
int y = Integer. parseInt(coordinateArray[1].trim());
```

to the wrong writing

```
int x = Integer. parseInt(coordinateArray[1].trim());
int y = Integer. parseInt(coordinateArray[0].trim());
```

This will reverse the Coordinate's x and y mapping.

Then when the user enters the coordinates (3,6) on the chessboard, the position of the chess piece will become (6,3).
Since the only class that reads board mapping files in the entire game is

BoardModel, we can quickly determine the location of the problem and fix it. This is the benefit that MVC brings us. MVC makes each class have a smaller scope of responsibility. After reducing the granularity of each function, the scope of responsibility of each class is limited. Problems always appear in a small area, and we can quickly locate and solve them.

This is the excellent performance of maintainability after the combination of MVC and full interface development.

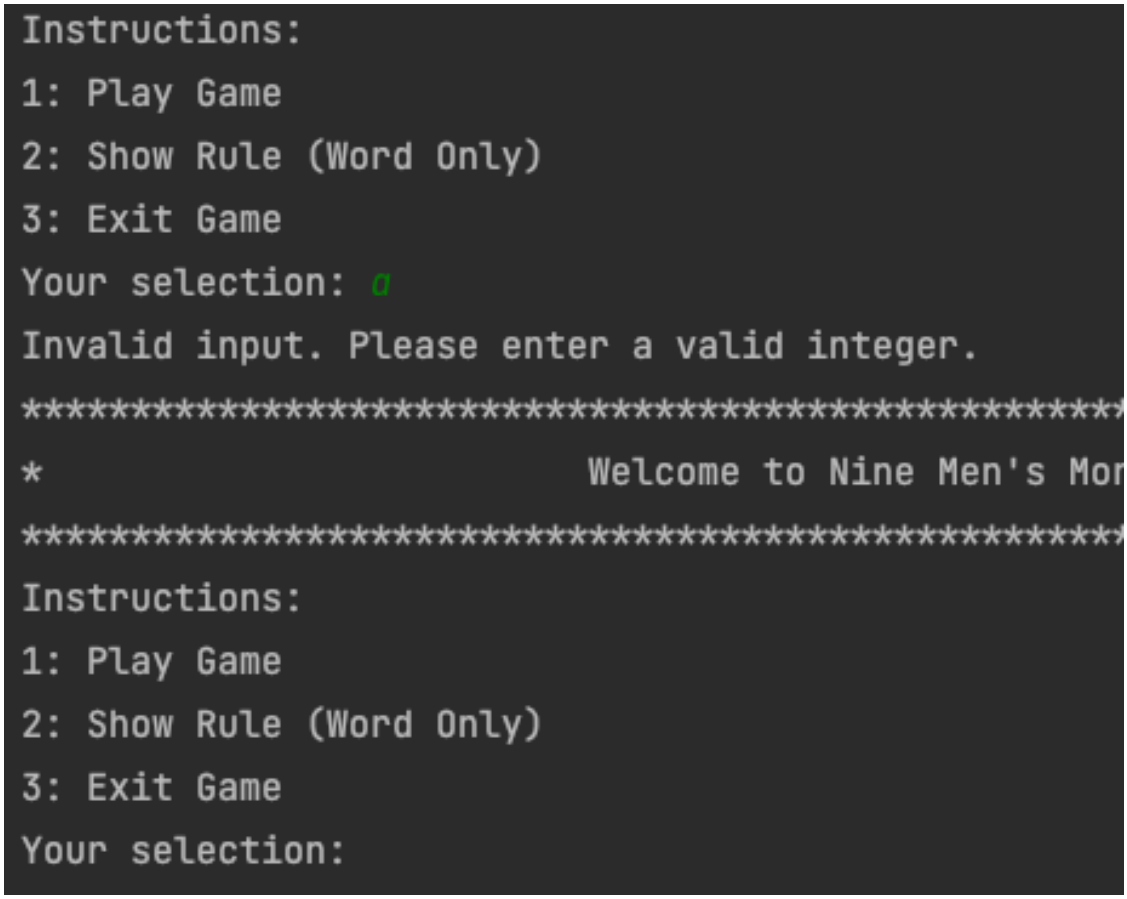
Maintainability is very relevant and important to our project. If our project has a high maintainability, we will always be able to quickly discover and solve problems in the subsequent development process. Reduced code development and maintenance costs. At the same time, maintainability is also an important indicator for evaluating code quality.[1] So maintainability is very important in our project.

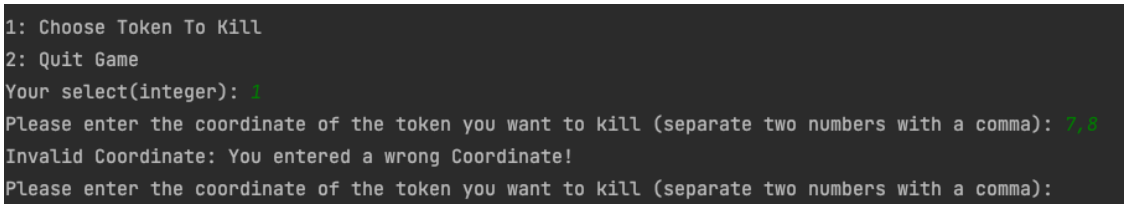
3.2.2 Usability

Our projects excel in usability. This is because we have designed an intuitive interface and made full use of prompts and interaction methods that conform to operating habits.

In order to more convincingly explain how our project has excellent performance in usability, I will use the five indicators summarized by ISO and Jakob Nielsen for evaluation. [2] [3]

- **Learnability**: User can easily start his first play. This is because the entire project is output based on Terminal. This is a simple and straightforward output method. We provide rich and satisfying hints. We put numbers before each option. Since the coordinate system on the chessboard is the same as the coordinate system used in real life, users can easily understand the coordinate system of the chessboard and perform input operations.
- **Efficiency**: Users will be able to operate the game quickly. There will be no more than 3 options in the different menus of the game, and each option has an independent function, so the user only needs to enter a number to enter the function of the corresponding option. All options in the game are manipulated with numbers. Therefore, the player always performs fast operations in the number area of the keyboard.
- **Memorability**: The difficulty for users to re-establish proficiency is very low. Because the operation of the whole game is extremely simple. And because we have a lot of text prompts as mentioned above, users can quickly recall past operations.
- **Errors**: Users can easily recover from errors and continue the game seamlessly. Our project has a perfect error catching mechanism. When the user enters coordinates that do not exist or enters an incorrect format. Our game system can always detect and catch errors at the first time and will prompt the user to guide the user to make the correct input. At the same time, since we manually catch user errors, the program will not crash due to wrong sending. Therefore, when the user sends an error, the program will not crash and exit, but will prompt the user to enter again after returning to the previous input state after the correct input.





- **Satisfaction**: The whole game is very enjoyable. We have done a lot of design in the game's start and end screens. And when the mill is formed, we will use eye-catching prompts to tell the player that a "kill" can be performed. This will increase the entertainment of the game and make the user more enjoyable.

To sum up, our program has extremely high usability.

3.2.3 Scalability

When we use the MVC architecture pattern, the coupling between modules is reduced, so that different modules can be developed separately. At the same time, because we use full interface development, dependency inversion and interface isolation are realized. The above reasons provide good scalability for our project.

Specifically, although our View is only output on the Terminal at this stage, we can expand the entire project to have a richer GUI mode. We just need to make a new class implement any of the other visualization techniques. This class needs to implement the View interface. Then, we only need to replace the products in the corresponding factory class.

Similarly, even on the Terminal, we can also expand. For example, now we are a 9MM game, we can expand it into a twelve men's morris game [4] [5], but we don't need to modify any source code, we only need to modify the configuration file of the required resources. This is because in the design process of the entire interface, we have carried out a higher level of abstraction, which makes the interface of the entire game system not only applicable to 9MM, but also applicable to many board games. This design perspective provides a strong guarantee for scalability.

Since the source code does not need to be modified, we also provide excellent scalability while satisfying the open-closed principle.

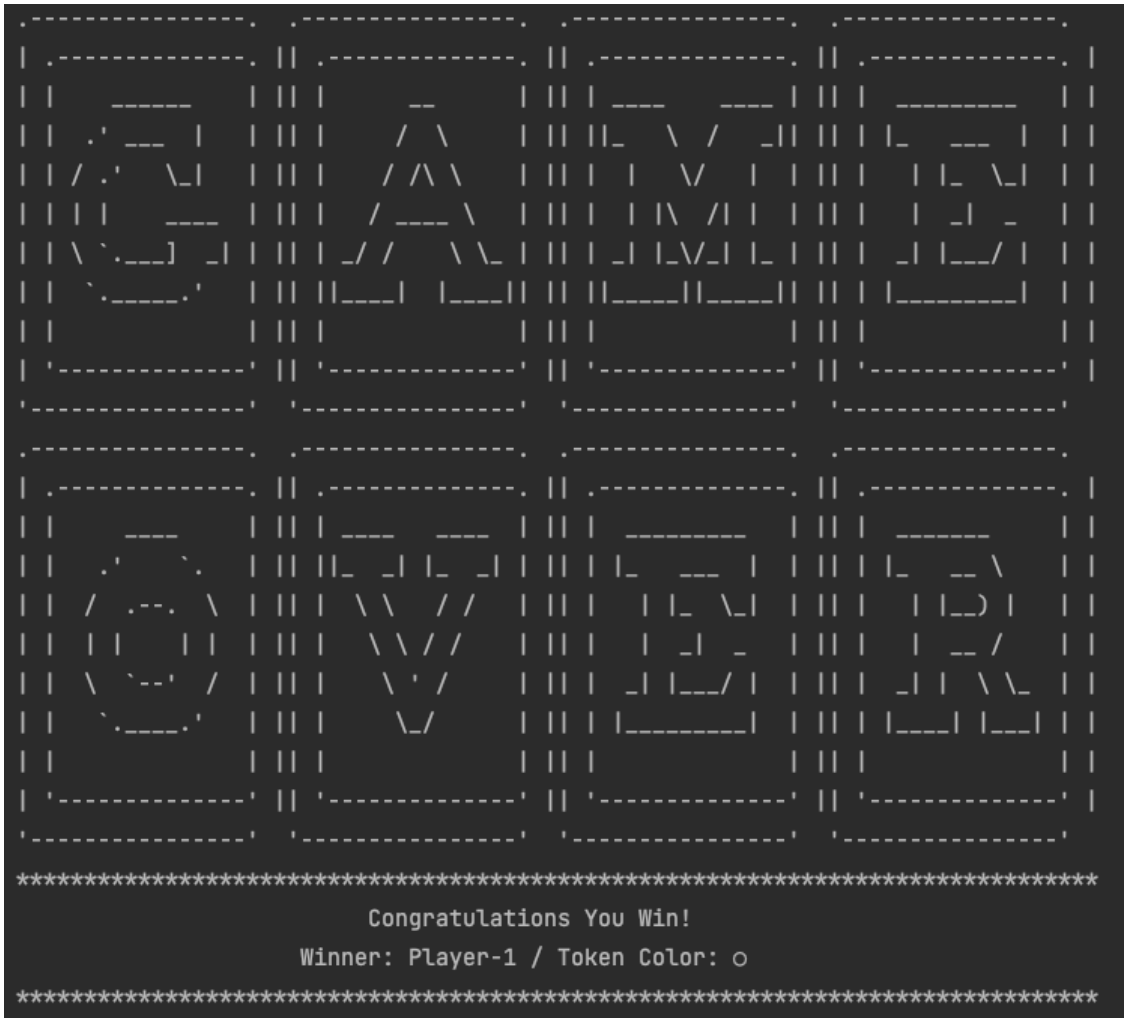
Scalability is extremely important in this project. Because we need to expand the basic functions in the next development. In this process, excellent scalability allows us not to have too much involvement with other classes in the expansion of functions. At the same time, we can easily expand without modifying other parts of the original system.

3.3 Explain human value. Why is it relevant and important to your game?

3.3.1 Achievement

Players can constantly practice their skills while playing the game. Improvement in technology brings a sense of accomplishment. At the same time, players will gain a sense of accomplishment after defeating AI or real players. These senses of accomplishment are similar to those of us playing other games and winning. At the same time, the game uses more eye-catching output prompts in the design to inform players that they have won. We used ASCII Art to output "Game Over" prominently

and inform the winning players. This output method is very impactful and will make the winning player feel very fulfilled.



3.3.2 Stimulation

Playing nine men's morris is very exciting for players who like a challenge. Because this game is not common, most people should be exposed to it for the first time. They will love to know and learn the game. Our advanced features in this project involve both human players and AI players. Players can challenge real people and AI, which will bring a different sense of excitement. Because we can design AI to be challenging. At this time, players will feel very exciting, which is the difficulty of the game they hope to achieve. At the same time, if two real players with the same ability are playing. Then the game between them will also become very exciting.

3.3.3 Self-Direction

During the game, players need to independently think about how to move each piece, formulate a strategy to form a mill, and make every decision, which can realize the value of self-direction. Specifically, players need to manually operate

each step during the game. We will not give hints about game solutions. This means that the player needs to think about his actions alone. This gives the player full freedom.

3.3.4 Universalism

Throughout the game, the same rules apply to both human and AI players. This means the game is absolutely fair. At the same time, from the relevant proof of the 9MM game. The expected win rate is the same throughout the game [6], which means it's a very fair game. And fairness is precisely part of universalism.

3.3.5 Security

Our game does not require networking, and it only uses the basic Terminal for output. This means that players don't have to worry about problems with their computers due to abnormal game crashes (we have avoided all situations that may cause game crashes). At the same time, because the game does not need to be connected to the Internet, it can be played normally. This means players don't have to worry about any network at all. Players are unlikely to encounter any scams or privacy issues, which is extremely attractive to security-conscious players.

4. Video Demo Link

- Please check out the Video demo in the **submission** checklist.
- Or please use the links below to view: https://youtu.be/LlszET_OpPE

5. Reference

[1] ISO. (2014, March). *ISO/IEC 25000:2014 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*. ISO. https://www.iso.org/standard/64764.html

[2] Wikipedia Contributors. (2020b, April 2). *Jakob Nielsen (usability consultant)*. Wikipedia. https://en.wikipedia.org/wiki/Jakob_Nielsen_(usability_consultant)

[3] Wikipedia Contributors. (2019, April 9). *Usability*. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Usability

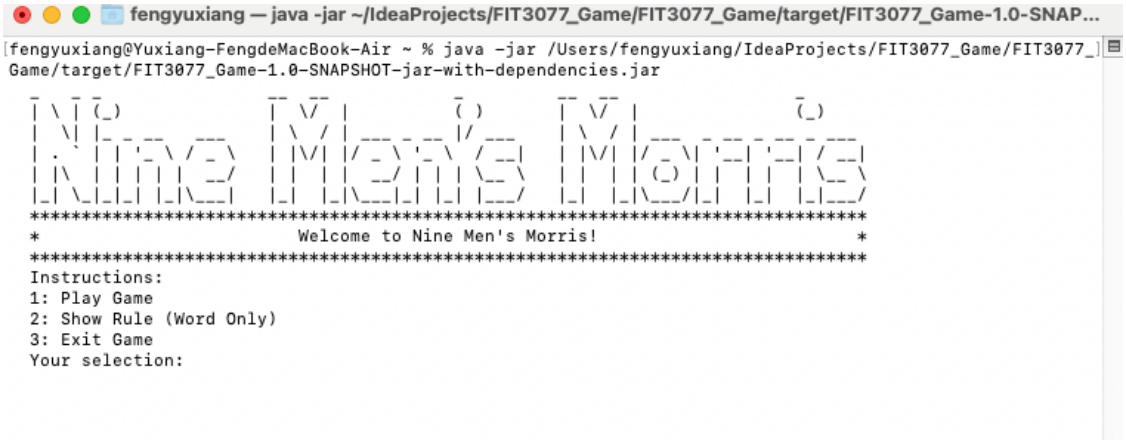
[4] Wikipedia Contributors. (2020c, October 26). *Nine men's morris*. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Nine_men%27s_morris

[5] Wikipedia Contributors. (2020a, March 20). *Morabaraba*. Wikipedia. https://en.wikipedia.org/wiki/Morabaraba

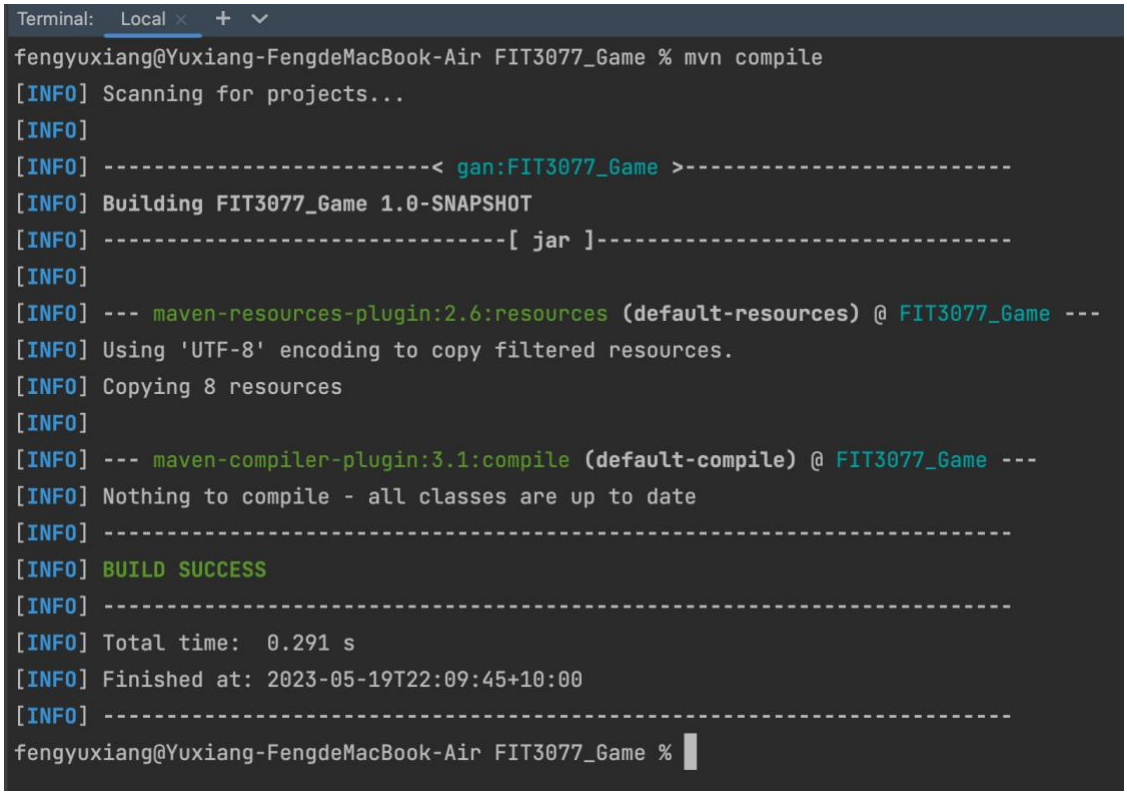
[6] Gasser, R. (1996). SOLVING NINE MEN'S MORRIS. *Computational Intelligence*, *12*(1), 24–41. https://doi.org/10.1111/j.1467-8640.1996.tb00251.x

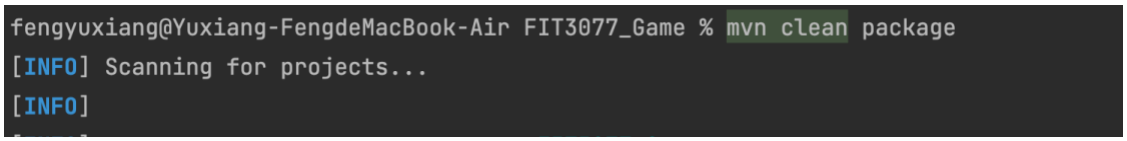
6. Executable Instruction

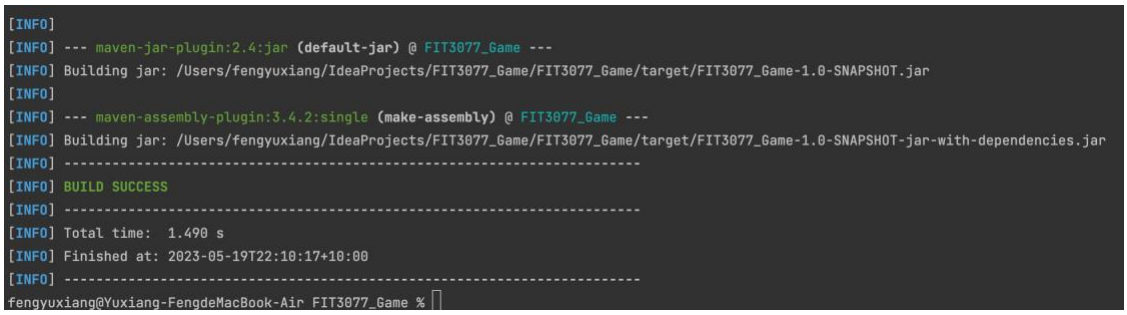
Please use the `java -jar` command in Terminal and enter the path to the "**FIT3077_Game-1.0-SNAPSHOT-jar-with-dependencies**" file located in the **Target** folder. (You can drag the file into Terminal for automatic path filling.) (Do not use "FIT3077_Game-1.0-SNAPSHOT", it is unusable)



If you want to recompile the whole project. Please open Terminal in the same folder as the pom.xml file, and enter `mvn compile` to compile the source code. Then use `mvn clean package` to compile the jar package based on the source code compiled at this time.


Terminal: Local
fengyuxiang@Yuxiang-FengdeMacBook-Air FIT3077_Game % mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< gan:FIT3077_Game >------------------------
[INFO] Building FIT3077_Game 1.0-SNAPSHOT
[INFO] --------------------------------[jar]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ FIT3077_Game ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 8 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ FIT3077_Game ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --
[INFO] BUILD SUCCESS
[INFO] --
[INFO] Total time: 0.291 s
[INFO] Finished at: 2023-05-19T22:09:45+10:00
[INFO] --
fengyuxiang@Yuxiang-FengdeMacBook-Air FIT3077_Game %


fengyuxiang@Yuxiang-FengdeMacBook-Air FIT3077_Game % mvn clean package
[INFO] Scanning for projects...
[INFO]


[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ FIT3077_Game ---
[INFO] Building jar: /Users/fengyuxiang/IdeaProjects/FIT3077_Game/FIT3077_Game/target/FIT3077_Game-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-assembly-plugin:3.4.2:single (make-assembly) @ FIT3077_Game ---
[INFO] Building jar: /Users/fengyuxiang/IdeaProjects/FIT3077_Game/FIT3077_Game/target/FIT3077_Game-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] --
[INFO] BUILD SUCCESS
[INFO] --
[INFO] Total time: 1.490 s
[INFO] Finished at: 2023-05-19T22:10:17+10:00
[INFO] --
fengyuxiang@Yuxiang-FengdeMacBook-Air FIT3077_Game %