Nine Men's Morris Project Report - Sprint4

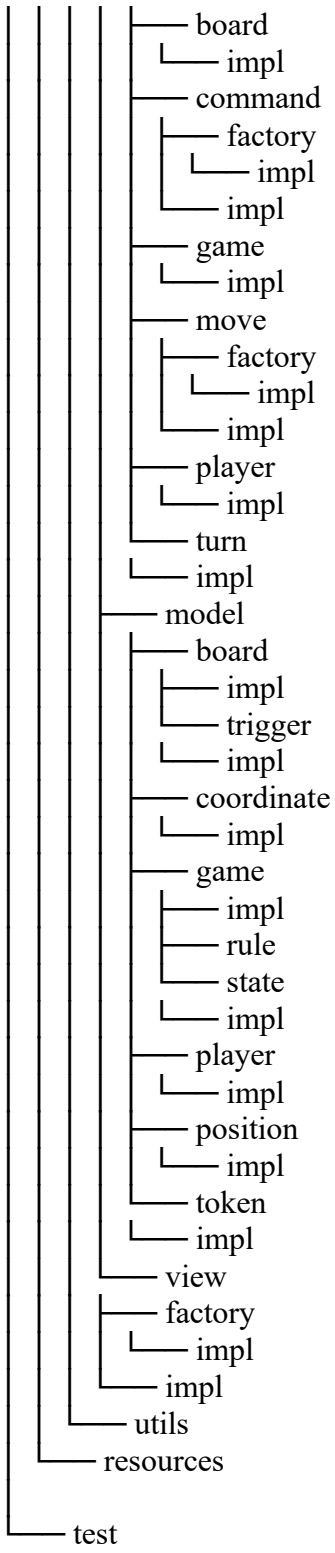
By Yuxiang Feng(32431813), Yifan Wang(32459238), Tianyi Liu(27936619)

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

0. Project Package Tree

We don't changed our architectural design, so our Project Package Tree is unchanged from Sprint3.

```
.
├── 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

1. User Story

We don't changed our advanced feature goals since Sprint1. We also don't changed our User Story.

But we want to add more details to describe our high-level functionality. The following is an additional user story.

- As a Player,

I want to play against AI Players,

So that I can practise my skills.

I want to be able to begin the match with AI Player in the main menu,

So it is convenient for me to start a match with an AI Player.

I want to make sure the AI Player does not forfeit the game,

So I can enjoy the game to the fullest.

- As an AI Player,

I want to access to all the possible moves in that turn,

So I can randomly choose one of the moves.

I want to ensure that my moves are valid,

So the game is fair.

I want to make sure I play my turn in a timely manner,

So the other player does not have to wait long.

2. Advanced Feature Goal

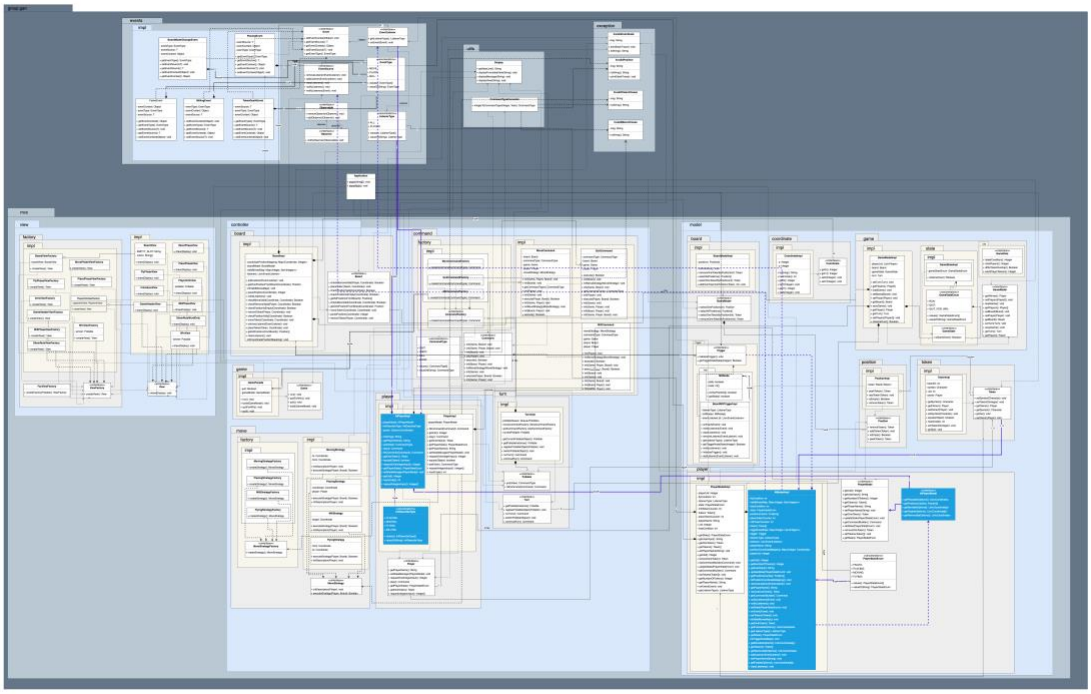
" c. A single player may play against the computer, where the computer will randomly play a move among all of the currently valid moves for the computer, or any other set of heuristics of your choice. "

(No change from Sprint1 to now)

3. Class Diagram

We didn't change our architectural design, so we just added two classes and an Interface to extend advanced functionality. Nothing about the original schema has changed. When we abstract to the interface layer, the interface interaction and interface isolation of the original architecture remain unchanged. Therefore, the following class diagram is just to clearly tell readers which classes we have added for advanced functions.

✧ *We use this color to highlight the newly added class*



Full Size Link:
https://drive.google.com/file/d/1MuwlBXHCDMj5ajLBSYAs5D0kbpCAaBXp/view?usp=sharing

4. Design Rationale

4.1 Explain why you have designed the architecture for the advanced requirement(s) the way you have

We did not modify the architecture for advanced functionality. From the beginning of the design of our design, we believed that the advanced function itself was only an extension of the basic function. Therefore, at the beginning of the design, we thought we needed to design a highly scalable architecture. Based on this theory, when we designed the entire game architecture, we blurred the specific details of the 9MM game, but focused on a more abstract and common level. This is why our architecture is designed from the interface layer, and it is also the reason why we choose full interface development.

The existing architecture designed based on this concept can not only expand advanced functions, we can even expand from 9MM games to chess. And such an extension does not lead to a modification of the architecture.

When we need to use the existing architecture to implement advanced functions, we can start from the interface layer to analyze how to extend advanced functions. Since our team chose to include AI players as our advanced feature. Therefore, we need to start thinking about how to implement it from the Player in the interface. Since we implemented the MVC architecture at the same time, the Player does not directly touch the game. This makes the Player less granular. We only need to use a brand new class to implement the Player interface and implement different functional details. AI Player has the same winning and losing conditions as Human Player. They can perform the same operations in rounds, so we don't need to modify most of the functions.

The only difference is that whenever Turn requests a Coordinate input from Player, Human Player needs to request real input from Scanner. The AI player needs to provide its own AI Player Model with options that can be operated. Since the event monitoring interface is provided in this architecture, the AI Player Model can easily monitor the information on the chessboard to provide the AI Player Controller with operable options.

The above is the process of implementing advanced functions based on the existing architecture. Due to the excellent extensibility of our original architecture, advanced functionality only involves two new classes and one interface. And did not modify the code in Sprint3. To sum up, we have completed the development of advanced functions in a non-invasive, lightweight, and agile manner under the premise of perfectly satisfying the Open and Closed Principle (OCP).

4.2 Explain why you have revised the architecture, if you have revised it. (What has changed would have been shown in the revised class diagram. This one is about why it changed)

We did not modify the schema. We still use the original architecture. Although we added a new interface, this interface inherits from PlayerModel Interface. Therefore, from a polymorphic perspective, it still belongs to the PlayerModel Interface. This is indeed the case. Although we have added an interface, in all the codes in Sprint3, we still maintain all type declarations, parameter types received by functions, etc., using PlayerModel Interface as the interactive interface. Another strong evidence that can prove that we have not modified the architecture is that we have not made any changes to the functional code implemented in Sprint3, and we have perfectly implemented the Open-Closed Principle (OCP).

4.3 Explain when your advanced feature was finalised (e.g. it is the same as we decided from sprint one; or we changed it in Sprint 3) and how easy/difficult it was to implement.

Our advanced features were confirmed in Sprint1 and no changes have been made since then. Implementation of our advanced features is extremely easy due to our good practices.

Based on the following good design practices/patterns, we can implement advanced features very easily.

4.3.1 MVC Framework

Our project implements the overall MVC architecture. The MVC architecture helps us reduce the coupling between the various classes of the project. At the same time, the relationship between Controller and Model. We can divide the functions inside the class more carefully. For example, if we were not using the MVC architecture, we would only have a Player class. With the help of the MVC architecture, the Player will be split into two classes. One of these two classes implements the operation logic of the player, and the other implements the data storage logic of the player.

When we expand the advanced functions, the operation logic of the AI player remains unchanged, and it needs to complete the functions of the Human Player. This means that the Turn class does not have to be concerned with the concrete implementation of the class it calls. Because both the Human Player and the AI player will have the same function for Turn to call, and can always get the same return type. And Turn does not need to care about the difference between AI and Human in data storage.

At the same time, since the controller and model of the AI player are divided into two different classes. Therefore, when our team is developing, it can be carried out simultaneously. Because we have provided a unified protocol in advance through the interface.

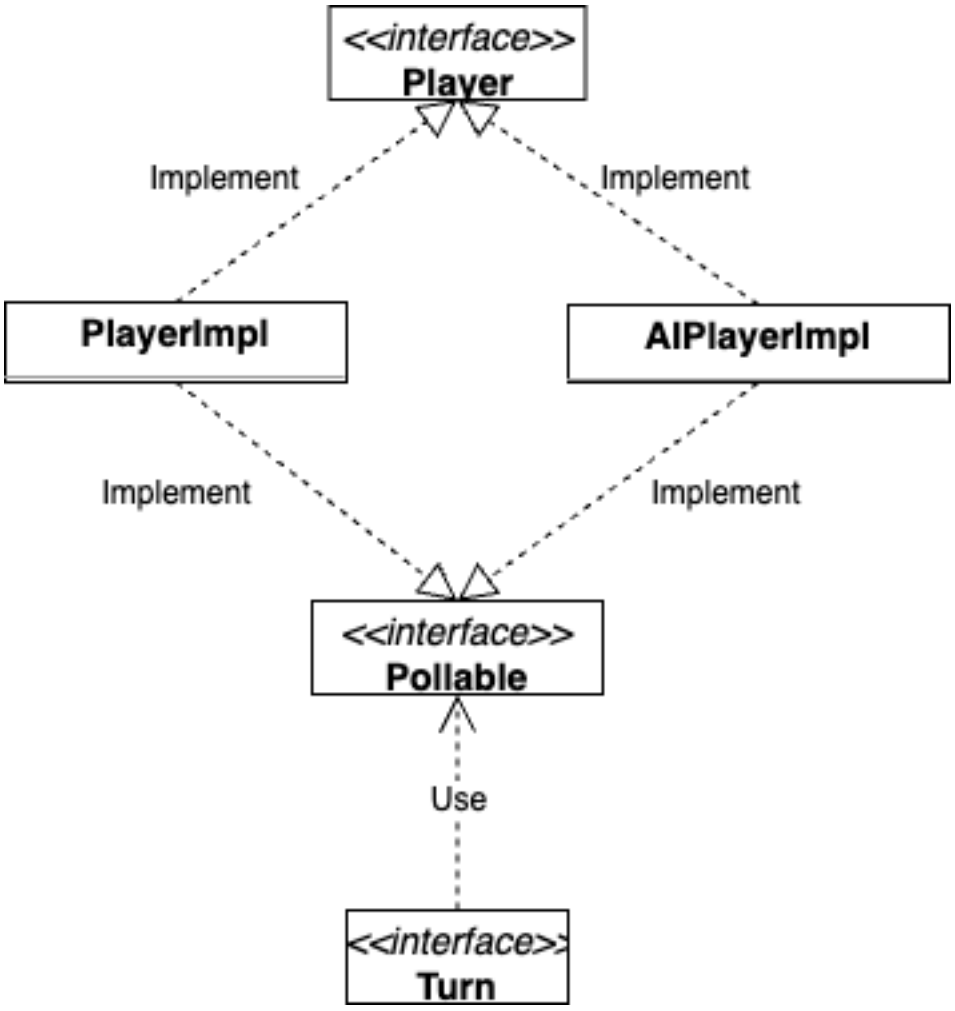
This is where MVC comes in handy when we extend advanced functionality.

4.3.2 Full Interface Development

Our project has achieved full interface development. Full interface development focuses the entire project on the interface layer. This frees us from the implementation details of the project. Instead, focus on the interface interaction of the project as a whole.

More specifically, both the Human Player Controller and the AI Player Controller implement the Player interface. This is the uniform constraint on which they implement their internal functions. At the same time, they will also implement the Pollable interface, which is the unified constraint for their interaction with Turn.

The full interface development method allows us to directly connect to the existing game system only by making the new class implement the provided interface when extending advanced functions. The scalability of the entire game has been greatly improved.

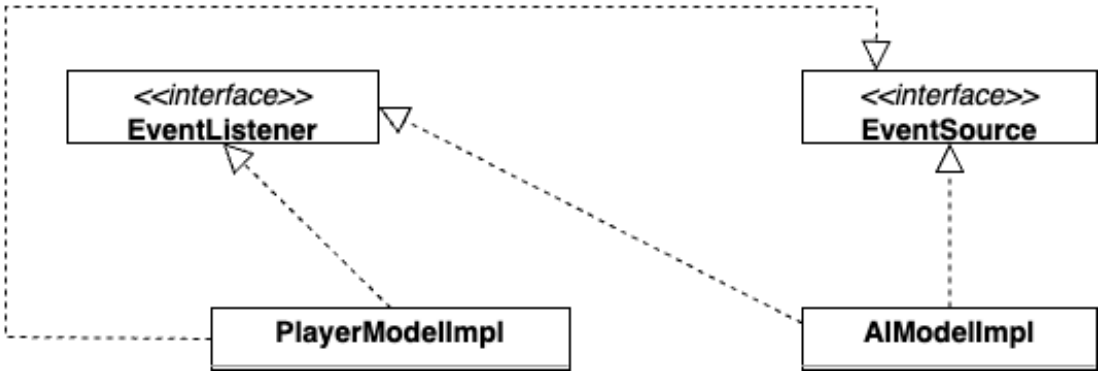


4.3.3 Event Interface

Our project implements the event listening pattern. The AI player has the same implementation as the Human Player in most functions. For example they all have the same transition state condition. (If the surviving pieces are equal to 3, the fly mode will be switched. If the surviving pieces are less than 3, the game will fail.)

Therefore, the event monitoring interface provides rich event monitoring operations. They can all monitor board changes to determine if they need to switch play states or the game has failed.

The implementation of time monitoring improves the flexibility of the whole framework, allowing us to easily extend advanced functions.



4.3.4 Combination Priority

It can be clearly seen from the statements and pictures in 2.3.2 and 2.3.3 that both the Player Controller and the Player Model implement multiple interfaces. This follows the Interface Segregation Principle (ISP) and improves code scalability. This helps us easily extend advanced functionality.

4.3.5 Dependency Inversion

Since we have achieved full interface development, we have perfectly realized dependency inversion. This reduces the coupling of various modules in our entire project, and improves the scalability, reusability and composition of the code.

5. Video Demo Link

Please use this link to view demo video: https://youtu.be/zCguFS35O_I

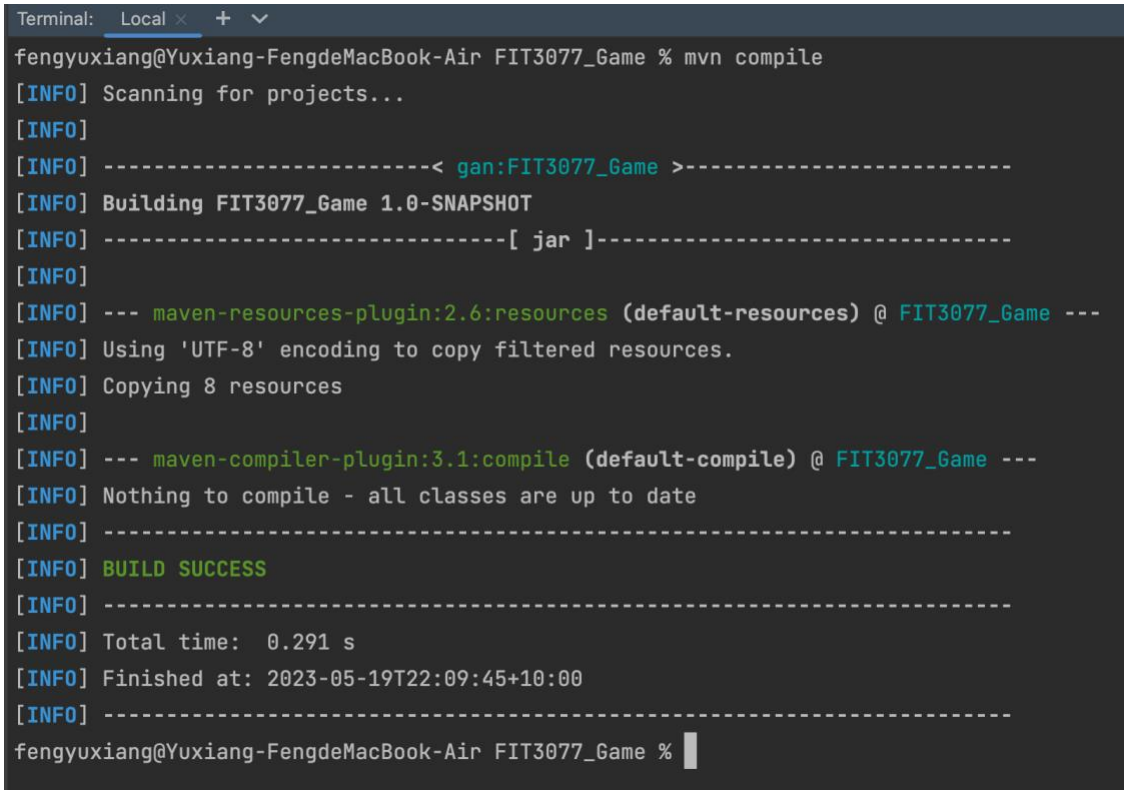
6. Executable Instruction

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**)


fengyuxiang — -zsh — 80×24
fengyuxiang@Yuxiang-FengdeMacBook-Air ~ % java -jar


fengyuxiang — -zsh — 80×24
fengyuxiang@Yuxiang-FengdeMacBook-Air ~ % java -jar /Users/fengyuxiang/IdeaProjects/FIT3077_Game/FIT3077_Game/target/FIT3077_Game-1.0-SNAPSHOT-jar-with-dependencies.jar

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.



```
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 %
```