

***FIT3077: Software Architecture
and Design***

Sprint 3

MNE Tech ©

***Ethel Lim
Nethara Athukorala
Mahesh Suresh***

Table Of Contents

Tech-based Basic Software Prototype	3
Prerequisite:	3
Instructions on downloading Java SDK:	3
Instructions to run the software prototype:	3
Sprint 2 Class Diagram	5
Sprint 3 Revised Class Diagram	6
Architecture and Design Rationale	7
Game package	7
Actors package	7
Player	7
Double Player	7
SinglePlayer	7
Actions package	7
Action	8
Place	8
Slide	8
Jump	8
Remove	8
Game	8
Display	9
Board	9
Position	9
Token	9
Rule	10
Mode	10
Design Architecture	11
Non Functional Requirements	12
Schwartz's Theory For Human Values	13
Communication Diagram	16

Tech-based Basic Software Prototype

Prerequisite:

- Java SDK version 20 (the latest version)

Instructions on downloading Java SDK:




- Go to the website <https://www.oracle.com/au/java/technologies/downloads/>
- Download according to your OS : Window , Linux , Mac




Linux	macOS	Windows
Product/file description	File size	Download
x64 Compressed Archive	180.81 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.zip (sha256)
x64 Installer	159.95 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.exe (sha256)
x64 MSI Installer	158.74 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.msi (sha256)





- Please follow the youtube video instruction for setting up the Java SDK 20 for Window users : <https://www.youtube.com/watch?v=AUL--F5Wdh8>
- Please follow the youtube video instruction for setting up the Java SDK 20 for Linux users : <https://www.youtube.com/watch?v=Lin1q9S4zTU>
- Please follow the youtube video instruction for setting up the Java SDK 20 for Mac users : <https://www.youtube.com/watch?v=ZeDBQ0d2P5M>

Instructions to run the software prototype:




- Download the main repository from the gitlab

 **project** 
Project ID: 40069 

  Star 0  Fork 0



 156 Commits  5 Branches  0 Tags  41.3 MB Project Storage


FIT3077 Project repository


main  project /  


Find file

Web IDE

Clone 

 **merge to main**
elim0050 authored 16 minutes ago

bf101537 






Add README

Add LICENSE

Add CHANGELOG

Add CONTRIBUTING

Set up CI/CD

Name	Last commit	Last update
 .idea	merge to main	16 minutes ago
 Sprint 1 docs	Sprint 1 documents	1 month ago
 Sprint 2 docs	documents	3 weeks ago
 game	merge to main	16 minutes ago
 out	created artifacts and build jar file	23 minutes ago

- Select 'out' folder and select 'artifacts' folder

main

project / out / artifacts / project.jar /

+

Lock


History

Find file


Web IDE


↓

Clone

 created artifacts and build jar file
elim0050 authored 24 minutes ago

f9b6de9f



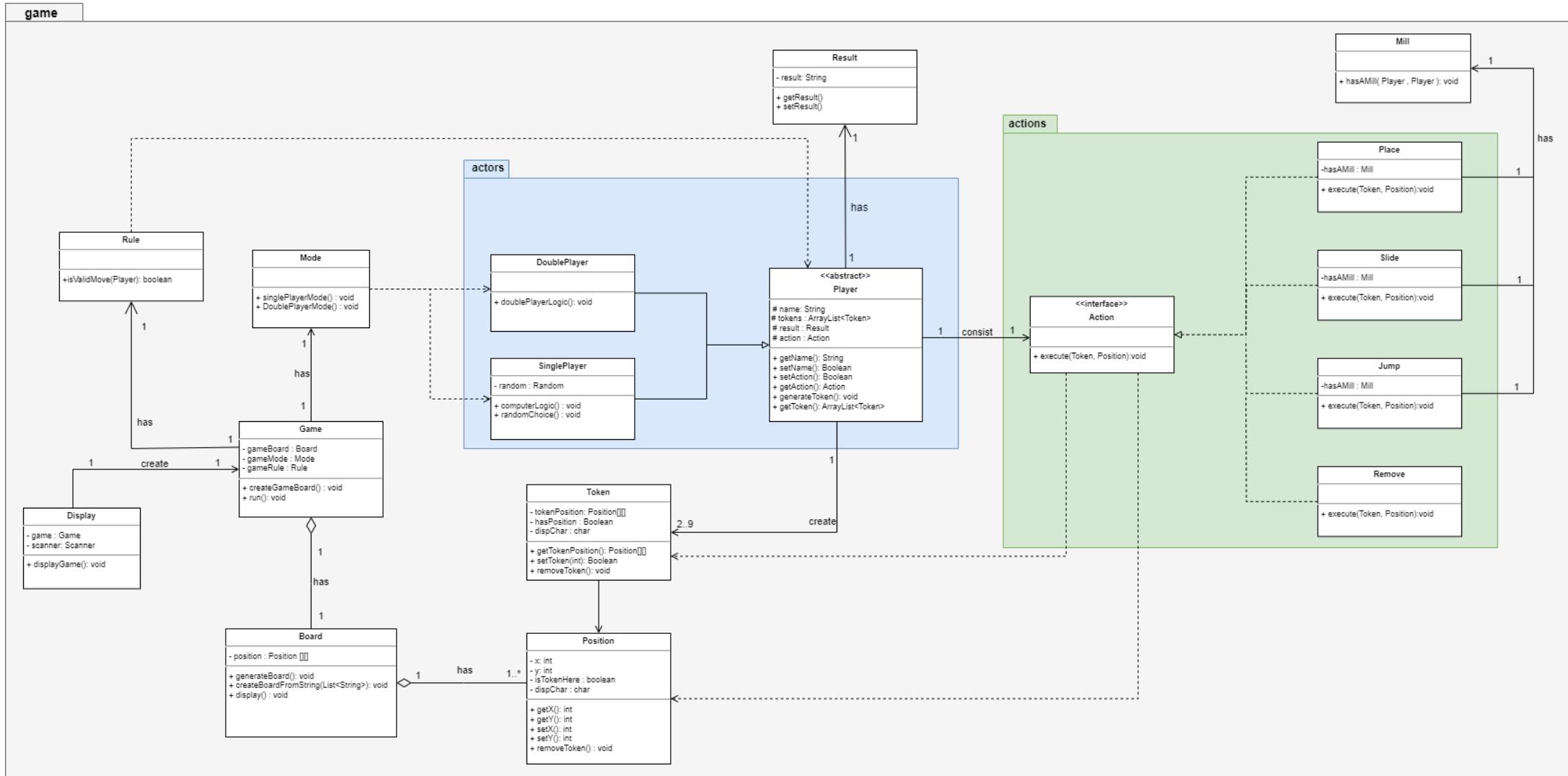
Name	Last commit	Last update
..		
 project.jar	created artifacts and build jar file	24 minutes ago

- double click on project.jar to run it on your desktop (check your prerequisite)

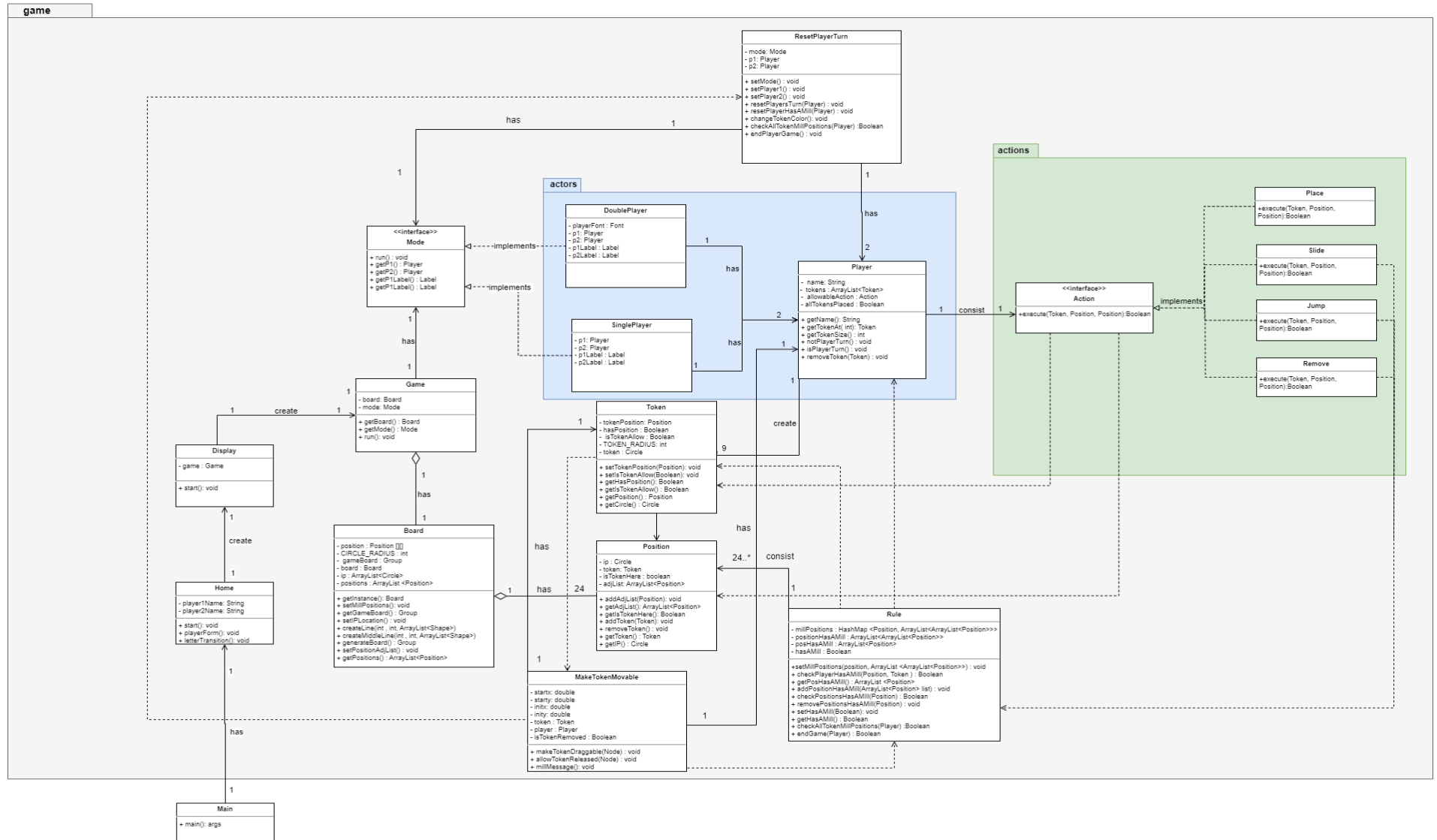
DemoVideo Link

Link : <https://youtu.be/6hPiSiojjsq>

Sprint 2 Class Diagram



Sprint 3 Revised Class Diagram



Architecture and Design Rationale

- Please take note that the highlighted classes below have been modified or newly added compared to the previous sprint.
- All of the classes in the class diagram adheres to the SOLID principle of Single Responsibility Principle (SRP), meaning that each class has a single, clearly defined responsibility or task to perform. Additionally, it makes the code more modular and easier to understand, as each class can be developed and tested independently.

Game package

All the classes are placed into a game package since they are part of the game.

Actors package

The actors package consists of the classes that are related to the user/player. The classes within the actors package include: Player abstract class, DoublePlayer and SinglePlayer classes.

Player

Player class is created to reflect on the player that is playing the game. It stores attributes like the player name, tokens and allowable action. Thus, it is associated 1 to 9 with Token since each player is given 9 tokens to start the game, and associated 1 to 1 with the Action class, to execute checking on the player actions. Moreover, it has methods to set the players turn to correctly define the players round.

Double Player

DoublePlayer class is a class that runs whenever a player chooses to play as a double player mode. This class implements the Mode interface since it is one of the mode options for the player to choose. It creates two player instances which are associated 1 to 2 with the Player class. Then it overrides the run method defined in the Mode interface to be able to run the double player mode game logic.

SinglePlayer

SinglePlayer class is a class that runs whenever a player chooses to play as a single player mode. In this mode, the player will play the game against the computer. This class implements the Mode interface and is similar to how the DoublePlayer class works.

Actions package

The actions package is concatenated with classes that are related to the action performed by the token. The following classes are present in the actions package: Action, Place, Slide, Jump, Remove classes.

Action

The Action class is the top-level class that encompasses all the possible actions that can take place in the game and is used to check if the player performs a valid action. It has a dependency relationship with the Token and Position classes. As the action involves the player choosing a token and moving it to a new position in which each action leads the token to a different position on the board. Since it has a dependency relationship with the two classes, the Action class is decoupled from the specific implementation of the game logic. This decoupling makes it easier to maintain, as changes made in other classes wouldn't affect the Action class.

It was an interface as there were a number of actions with different behaviour to be performed by the player. This allows more flexibility, as different actions can implement this class to have their own implementation way and own function.

Place

Place class implements the Action class since it is an action that needs to execute when the player is placing the tokens to the board at the start of the game. Hence, it implements the abstract method 'execute' defined in the Action class to check if a player executed rightly.

Slide

Slide class is one of the actions that can be performed by the token when all tokens are placed on board, so it implements Action class. It limits the position where the token can be moved to, so it checks if the player is sliding the token correctly on the board.

Jump

Jump class is an action to be performed by the token whenever the player is left with 3 tokens on the board. Hence, it implements the Action class to implement the abstract method defined. The Jump class represents the action of jumping a token to a valid position on the game board regardless of lines or adjacency.

Remove

Remove class is an action that removes the token of the opponent player from the board. Whereby this action will execute whenever the player has a mill. Since it removes a token, it also implements the Action class to check if the token is removable.

Game

The Game class is the main class of the Nine Men's Morris game and it is responsible for coordinating the different components of the game, such as the board, the rules, and the game mode. Therefore, this class has an association relationship 1 to 1 with the Mode class and an aggregation relationship 1 to 1 with the Board class to allow it to coordinate the different components of the game.

Instead of composition , it has an aggregation relationship with the Board class. As it must instantiate a game board to start the game, while the Board class can exist on its own without having the Game class to exist.

Display

Display class is created to the final outcome of the Nine Men's Morris game which allows the user to interact with the gameboard. This class will therefore have a 1 - 1 relationship with the Game class as the display will only need one game instance, since only one game was needed throughout the program.

Board

The Board class is created as a Singleton pattern whereby only one instance of Board class can be created. This was made to avoid multiple instances of Board class, as the game can only have one game board. Hence, the class will be responsible for initialising the board UI and displaying the current board status.

This class has an aggregation relationship 1 to 24 to the Position class instead of composition, as a game board is made up of 24 intersection points while the Position does not require a gameboard to exist. This was done as the Board class needs Position to exist while Position can exist without having a Board, resulting in an aggregation relationship.

Position

The Position class represents each intersection point created in the game board. It is associated 1 to 1 with the Token class in order to store the Token instance whenever the position has a Token placed on it. Hence, it has attributes that check if it is a token on the position and consist of a list of adjacent positions to ease the process of checking Slide Action performed by the player.

These attributes are essential in order to determine the current position of the tokens on the gameboard and for validating player moves. Hence, the class consists of methods for removing and placing tokens.

Token

Token class is created for managing the tokens used in the game. It contains several methods that allow for setting the position of a token, removing a token from the game board, and checking if a position is currently occupied or not. Therefore, this class has an association relationship 1 to 1 with the Position class and is dependent on the MakeTokenMovable class. As each Token instance needs to add on actions that allow the user to move the token on the board UI.

The only primary change was the relation between the rule and token class which would allow rules to access and view tokens that were placed on the board, this would allow us to

identify if mills were made or not based on the tokens placed instead of checking the whole board ultimately improving efficiency.

Rule

The Rule class is created to set up the game rules which is specifically used to check if a player has a mill. It consists of methods to check if a player has a mill and checking if the current position is a mill position. Hence, it has a hashmap that is used to store position as a key and the corresponding positions that will form a mill as a list in the value. Thus, it is associated 1 to 24...* with the Position class, as it stores all of the position instances.

Moreover, this class has a dependency relationship with the Player class as it needs to be able to access player information to enforce the game rules. A separate class was created for the Rule class instead of having methods checking all in the Game class to avoid having “god” classes.

ResetPlayerTurn (new class)

ResetPlayerTurn class is a class created to reset the player turn each time a player finishes executing. It is associated 1 to 2 with the player class and is associated 1 to 1 with the Mode class. This class is like a helper class that only consists of static methods, which need to be executed after a player finishes moving a token on the board.

MakeTokenMovable (new class)

MakeTokenMovable class is a class that adds actions on each token instance to allow the token to move on the game board UI. This class is also used to read the token moved by the player and perform checking according to the movement, Hence it is associated 1 to 1 with the Token and Player class.

Mode

The Mode class is responsible for handling the different game modes available in the Nine Men’s Morris game, such as the single player and double player modes. It is created as an interface that allows both the SinglePlayer and DoublePlayer classes to be implemented.

It is made into an interface to satisfy the Open-closed principle, Liskov Substitution principle and Dependency inversion principle. It allows handling multiple game modes effectively without the need of dependency or association relationships between other classes.

Hence, the Game class can directly call the Mode interface to run the game logic without creating extra instances of the DoublePlayer or SinglePlayer class.

Home (new class)

Home class is responsible for the initial page of the game on launch. It extends the Application class and overrides the start method in order to launch the game. Additionally, it has a method to get the names of the players before the start of the game.

Main (new class)

Main class is added to run the whole game application.

Design Architecture

The design architecture was revised to form a stronger and more secure design pattern compared to previous ones. Looking back at the previous design patterns, it is insufficient consideration for future implementation. It is lacking for separation of concerns, such as not properly separating responsibilities between different classes , which can easily form a GOD class, leading to code duplication and harder to make changes in the future. Also, it includes only some dependency between classes which makes it harder to connect all classes together. Moreover the design patterns also show an absence of high level design patterns which should be used to guide overall design structure.

Therefore, a revised class diagram was generated for this sprint. As usual, we make use of creational design patterns that make a Singleton for the Board class to create only one instance of the Board class throughout the game which enhances flexibility and provides code reusability. Moreover, behavioural design patterns are applied to the Action interface and Mode interface. As both of the classes were made to an interface to make use of the strategy design patterns. Such as the Action interface which allows actions: remove, place, jump and slide to implement different behaviour on the same method. Same for the Mode interface which allows different game modes to run with different logic.

Additionally, these design patterns adhere to a high level design pattern which is the Model-View-Controller. This design architecture can separate a design into three interconnected components, which is the model, view and controller.

- Model
 - Classes : Token, Player , Rule , ResetPlayerTurn, MakeTokenMovable , Board
 - These classes exist to implement the overall game logic , which represent the game rules , token rules or check for the token movement.
- View
 - Classes : Display , Home
 - The two classes are displayed as an UI that allows capture for user interaction.
- Controller
 - Classes : Game, Mode, Action
 - These classes are created to send information between the view and the model classes. Such as the Game class that creates a board and passes it to the Display class which allows different components to coordinate accordingly. The Mode class captures different game modes and sets the game mode to the classes in the model to handle different game logics.

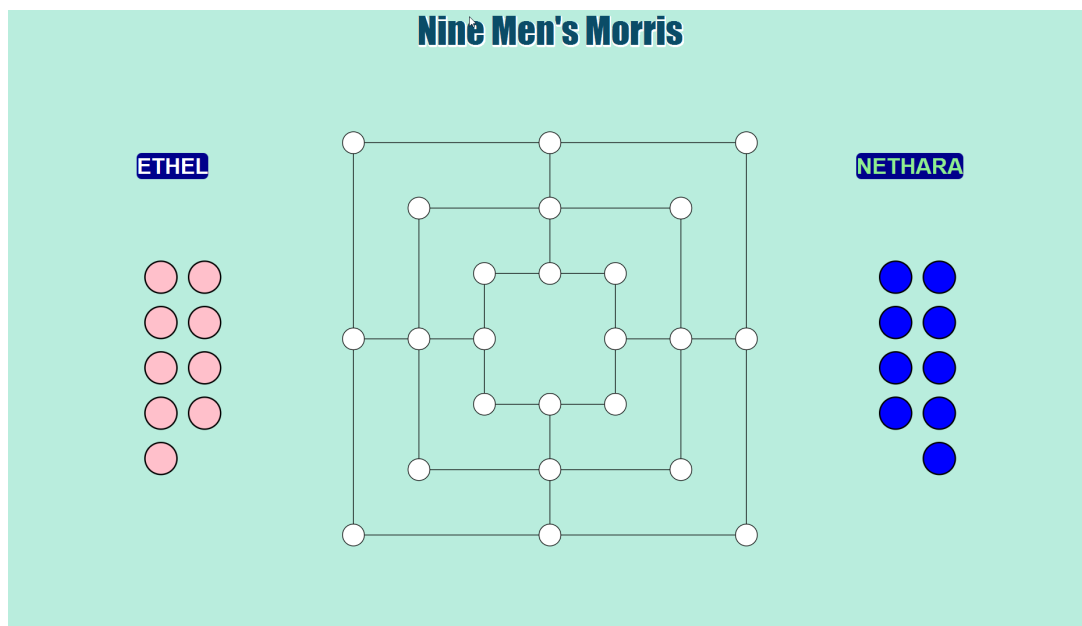
Hence, the use of Model-View-Controller(MVC) architecture provides stronger design patterns compared to before.

Non Functional Requirements

Usability

Usability is the measurement of how easy it is for users to perform desired actions, within our implementation of the game, we made this a priority. The impact of having an easy to play game in this instance is because Nine Men's Morris is originally a board game and conveying it into a digital game means that it's importance to capture the essence of the basic movements and simplicity of playing it, this ultimately allows players to feel more captivated within the game as well as also allowing them to better grasp the game instead of feeling discouraged about the difficult in carrying out tasks.

We achieved usability by implementing a simple UI interface that allows players to directly interact with pieces allowing for movement to be simply done through drag and drop. In doing so we were able to capture the feel of players playing the game via board and also managed to reduce the difficulty in carrying out operations as alternatives may have instead been using inputs via keyboard to directly control options displayed. Moreover, we created an UI that allows players to choose to play as double player or single player mode in order to get player choices.



Performance

Performance is the measure of how quickly a system is able to respond to user inputs, our implementation incorporates this quality attribute through use of a singleton as a building board for the game. Performance is especially important within the game as it is necessary for us to be able to have actions performed quickly in accord with the user's actions in order to create fluidity between both the interface and the backend. Having only one instance of the board determines that our code has been designed in a method in which we edit existing pieces on the board and adjust them accordingly, in turn reducing the computation and resource needed as we don't generate a new board every time a task is performed.

```

/**
 * create only one board instance and return the board instance
 */
4 usages  elim0050
public static Board getInstance(){
    if (board == null){
        board = new Board();
    }
    return board;
}

```

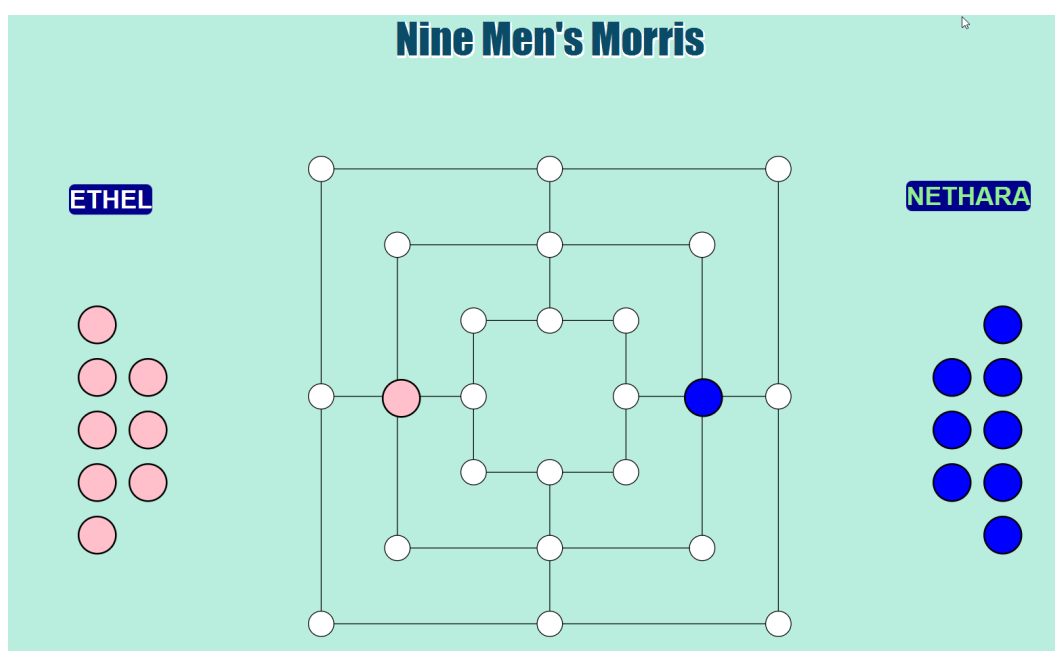
Schwartz's Theory For Human Values

Creativity

In accordance with Schwartz's theory the focus on creativity was a factor for the creation of our Nine Men's Morris game. Creativity is important to our game as it better engages players to play the game and improves focus, in turn retaining information and increasing game time and enjoyability.

A key part of the original Nine Men's Morris is the nature of the game being played on a board, in order to capture similarities in terms of both ease of movement as well as token movement it was decided that we would create an interactive drag and drop UI to allow players to be better immersed within the game.

By using a drag and approach method to allow players to interact with both board and token it allowed us the chance to creatively implement a familiar feature that players who have played on the board version may understand, and also allows players to better familiarise themselves with the original game. This ultimately allows players to play a truer version of the game and enjoy a more genuine experience of what is to be offered.



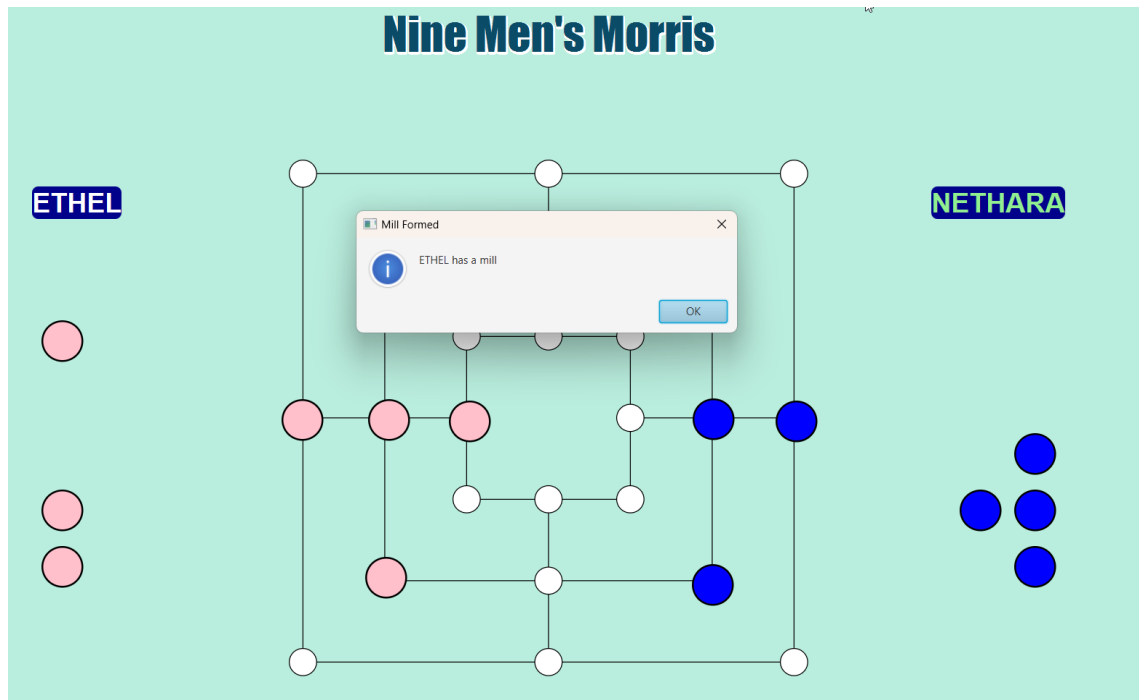
Achievement

The value of achievement, as described in Schwartz's theory of basic human values, holds great relevance when applied to the game of Nine Men's Morris. This value encompasses the pursuit of personal success, competence, and recognition, which align perfectly with the dynamics of the game.

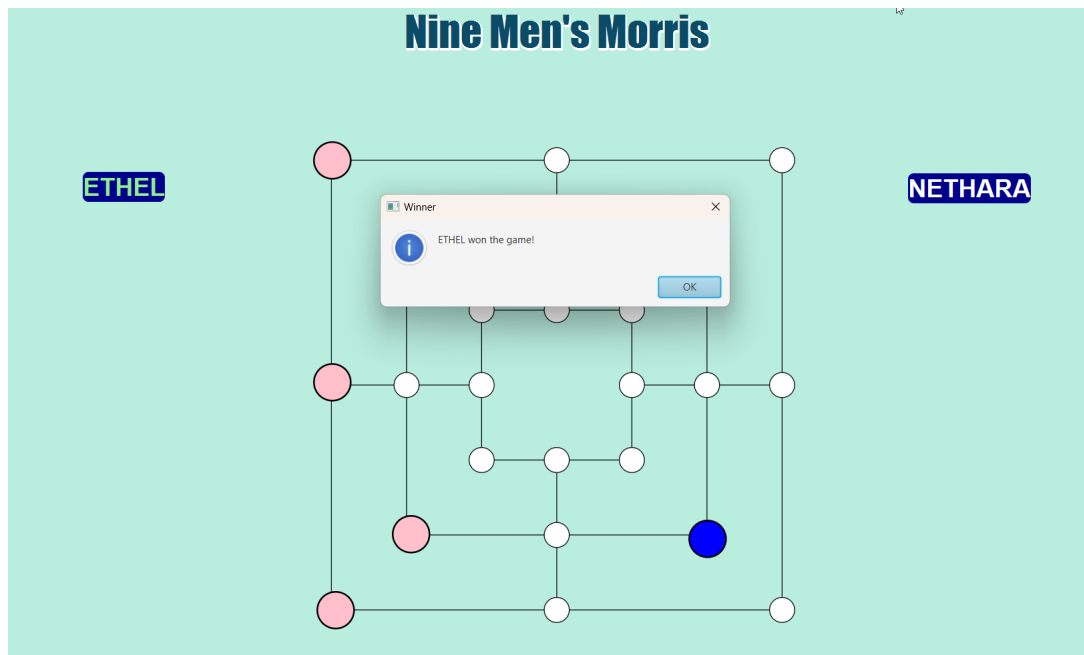
Nine Men's Morris is a strategic game that demands careful planning, anticipation of opponent moves and strategic positioning of pieces on the board. The primary objective is to create "mills" by forming rows of three tokens and subsequently removing the opponent's tokens or blocking their moves entirely. For individuals who value achievement, the game becomes a platform to show their skills, hence they continuously improve their gameplay to showcase their competence. Achievement-oriented players continuously seek to improve their skills on the game by spending more time learning the necessary tactics in order to challenge their opponent and win the game.

Overall, in our implementation, it can be seen that the value of achievement plays a significant part in the Nine Men's Morris game. After a player wins the game, as the opponent has less than two tokens, a message will be displayed on the screen with the winner's name. Doing so, enables the player to have a sense of accomplishment after playing the game.

Player has a mill :

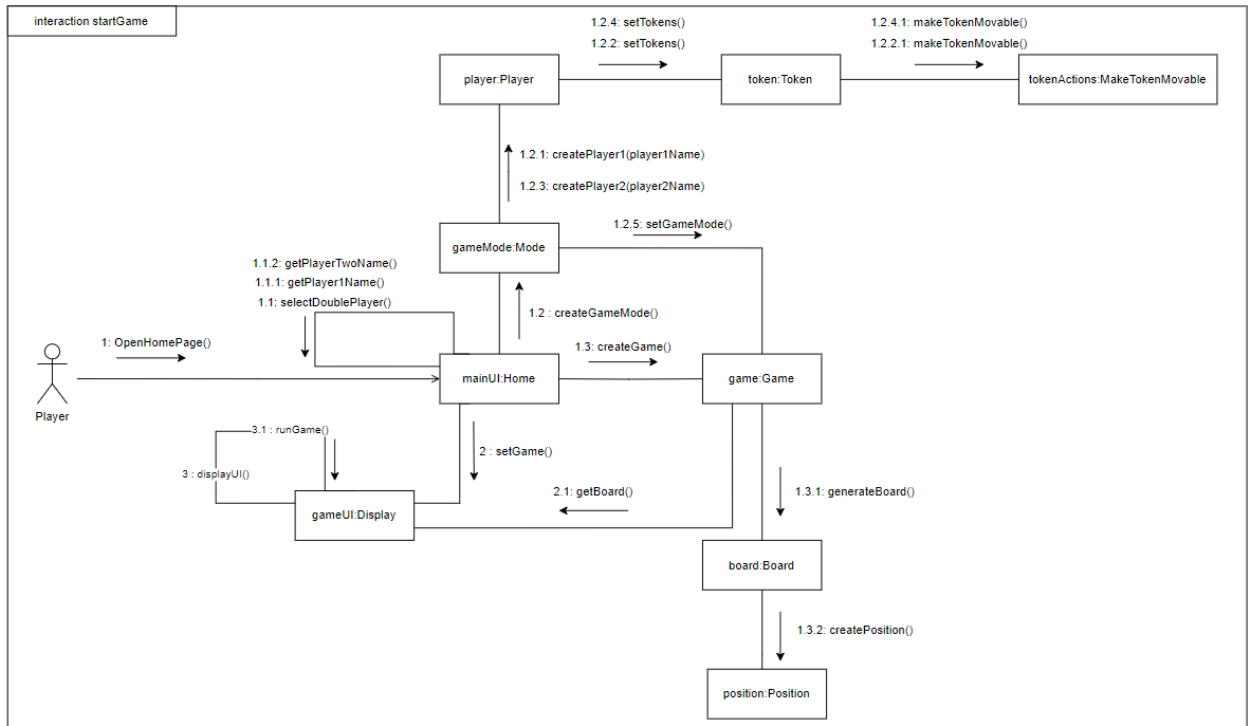


Player win the game :

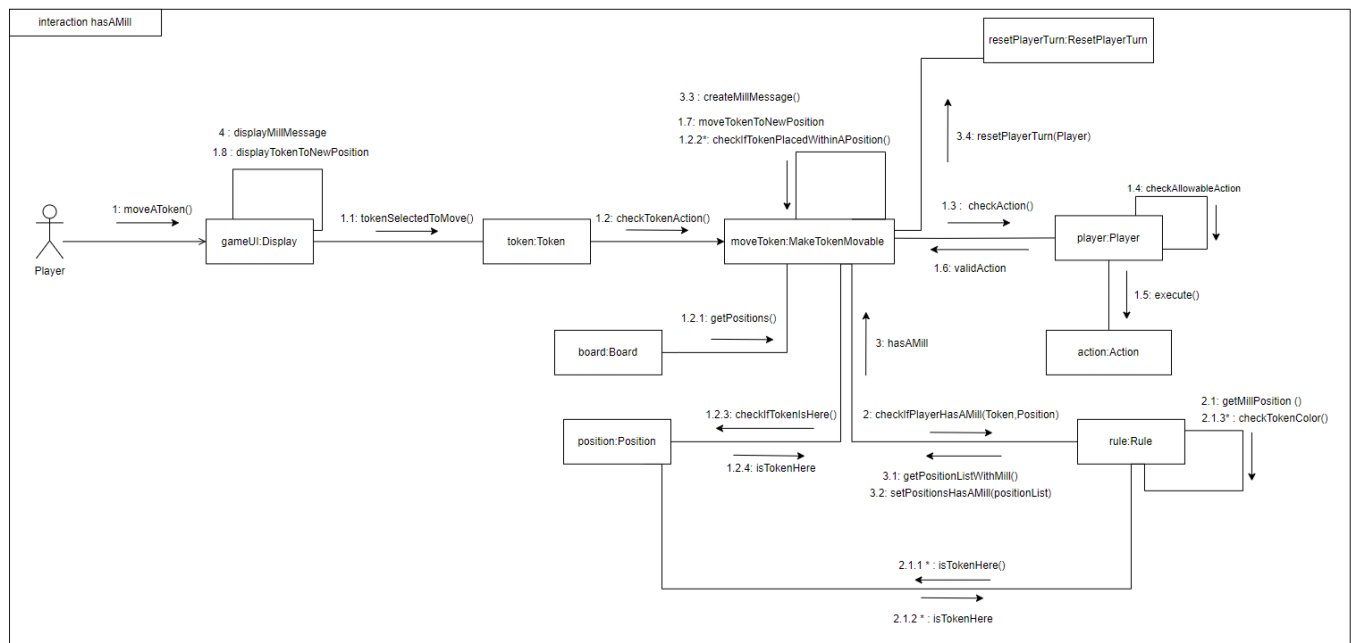


Communication Diagram

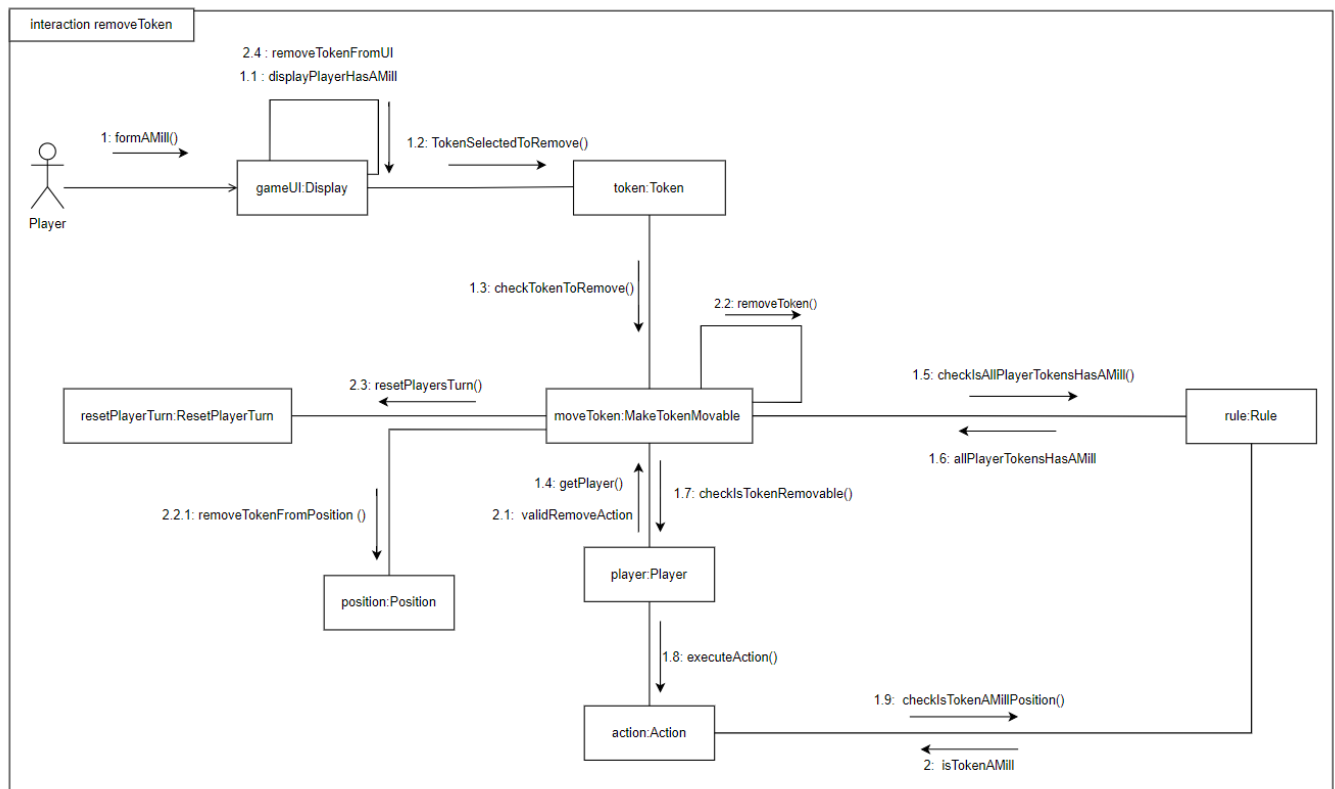
Start of the game :



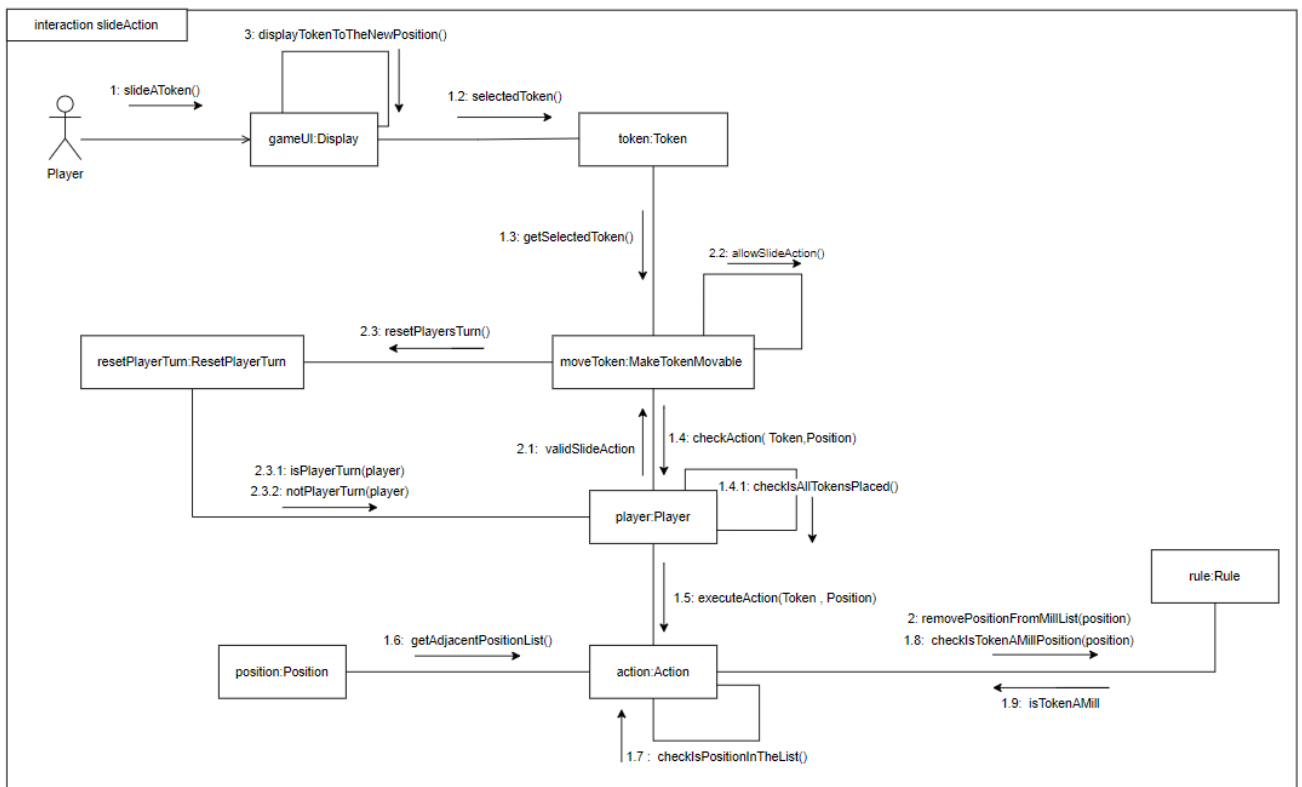
Player Form A Mill On Board :



Player select opponent player token to remove :



Player Slide Token on the Board :



Player left with 3 tokens and perform jump/fly action on the board :

