FIT3077: Software Architecture and Design

Sprint 4

MNE Tech ©

Ethel Lim
Nethara Athukorala
Mahesh Suresh

Table Of Content

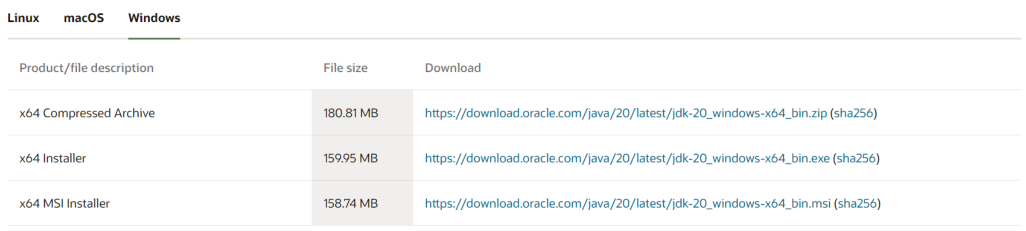
Tech-based Basic Software Prototype

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

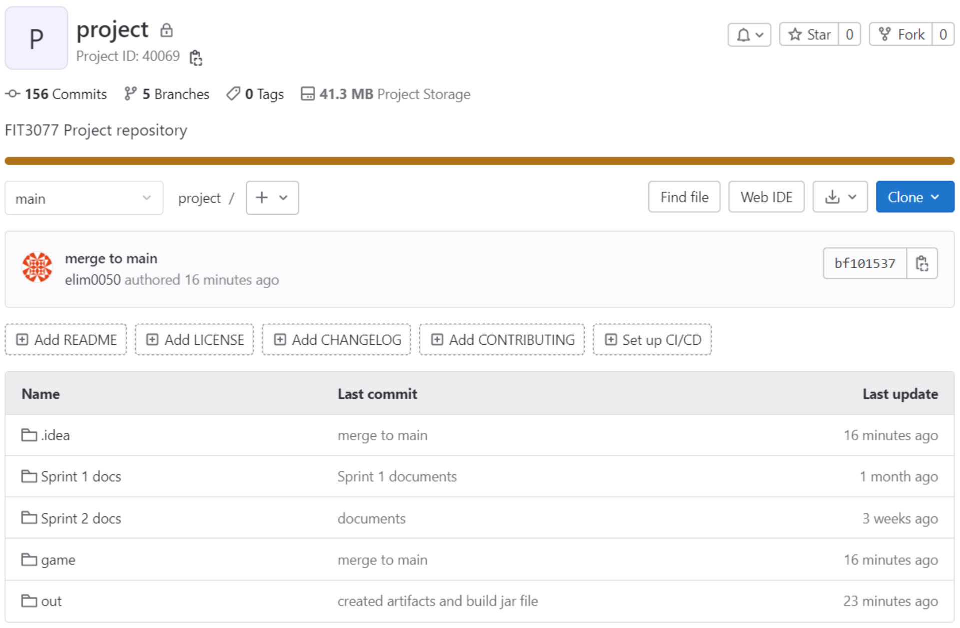


Product/file description	File size	Download
x64 Compressed Archive	180.81 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.zip (sha256)
x64 Installer	159.95 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.exe (sha256)
x64 MSI Installer	158.74 MB	https://download.oracle.com/java/20/latest/jdk-20_windows-x64_bin.msi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

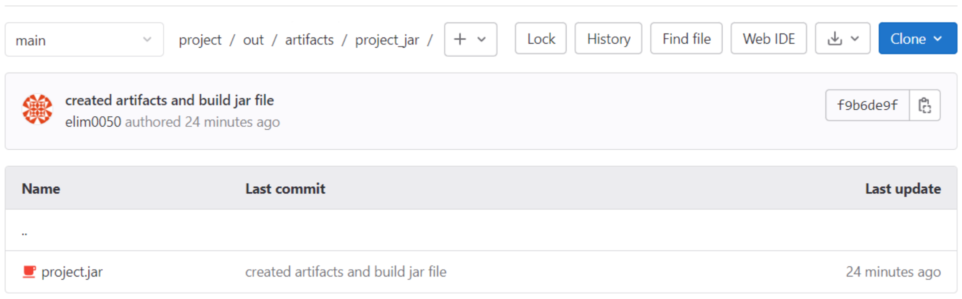
156 Commits 5 Branches 0 Tags 41.3 MB Project Storage

FIT3077 Project repository

merge to main
elim0050 authored 16 minutes ago
bf101537

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



created artifacts and build jar file
elim0050 authored 24 minutes ago
f9b6de9f

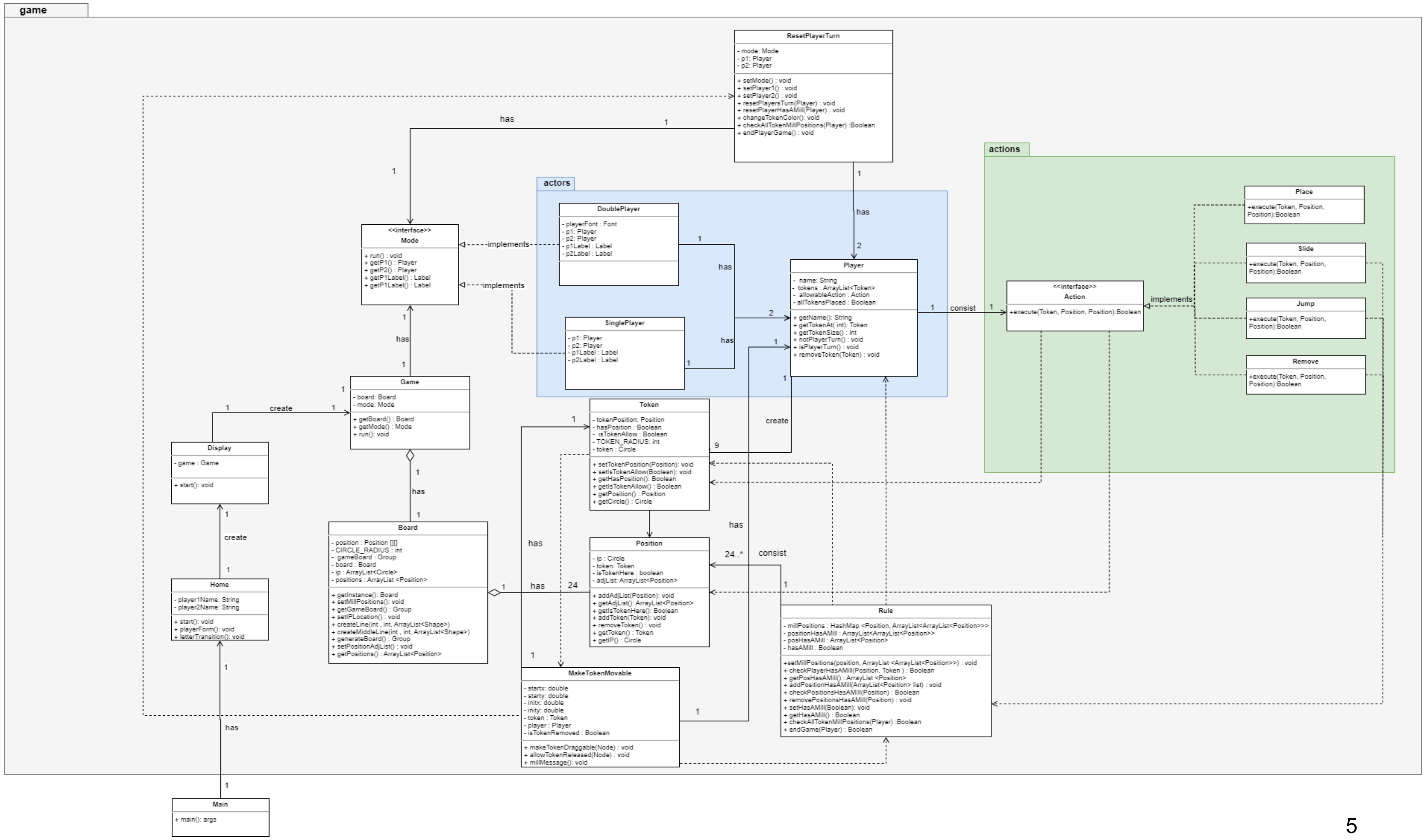
Name	Last commit	Last update
..		
project.jar	created artifacts and build jar file	24 minutes ago

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

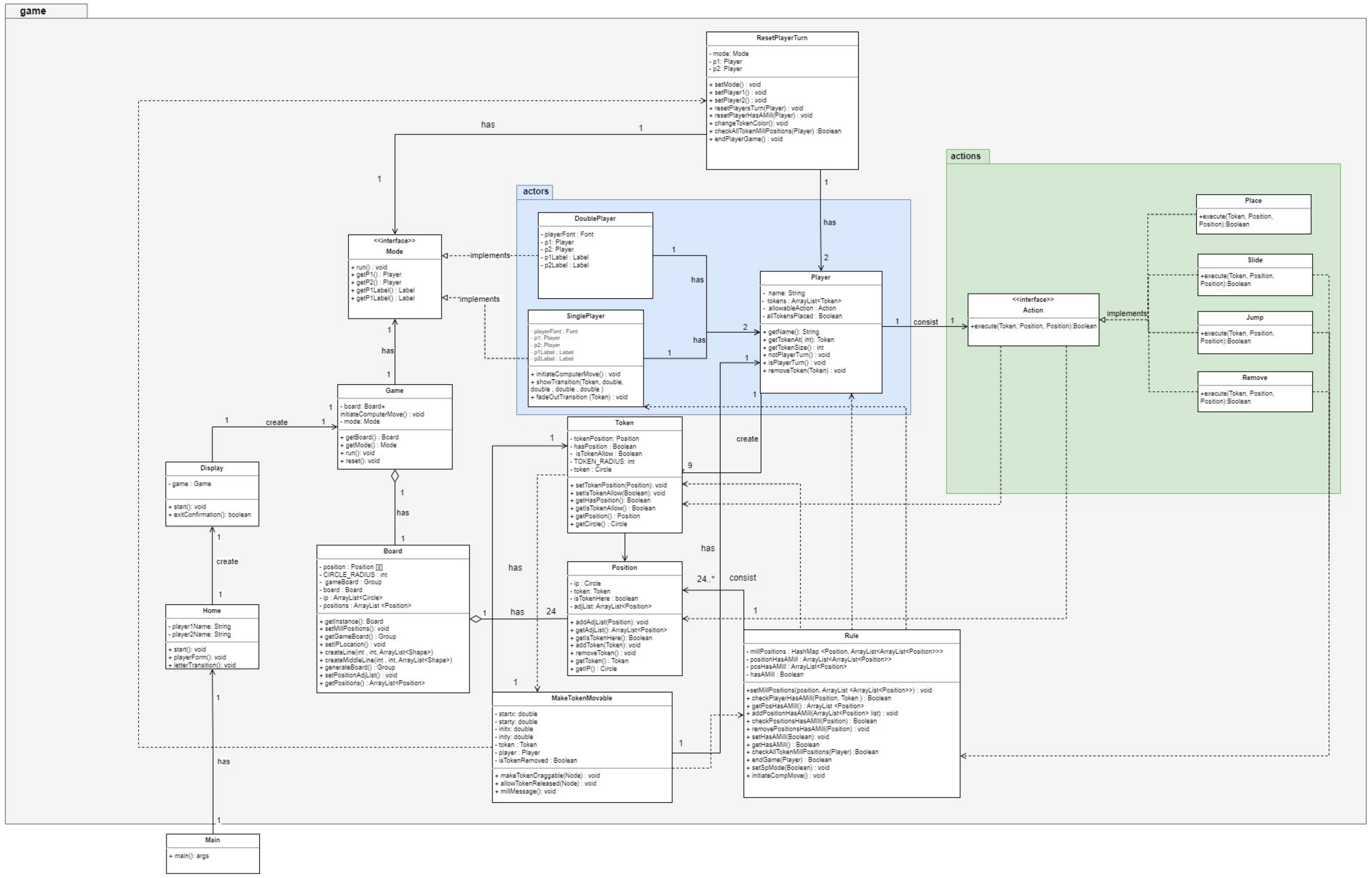
Demo Video Link

Video Link: https://www.youtube.com/watch?v=zfFuxtAF93s

Sprint 3 Class Diagram


game
ResetPlayerTurn
- mode: Mode
- p1: Player
- p2: Player
+ setMode() : void
+ setPlayer1() : void
+ setPlayer2() : void
+ resetPlayersTurn(Player) : void
+ resetPlayerHasAMill(Player) : void
+ changeTokenColor(): void
+ checkAllTokenMillPositions(Player) :Boolean
+ endPlayerGame() : void
has
actors
DoublePlayer
- playerFont : Font
- p1: Player
- p2: Player
- p1Label : Label
- p2Label : Label
SinglePlayer
- p1: Player
- p2: Player
- p1Label : Label
- p2Label : Label
<<interface>>
Mode
+ run() : void
+ getP1() : Player
+ getP2() : Player
+ getP1Label() : Label
+ getP1Label() : Label
implements
implements
Player
- name: String
- tokens : ArrayList<Token>
- allowableAction : Action
- allTokensPlaced : Boolean
+ getName(): String
+ getTokenAt(int): Token
+ getTokenSize() : int
+ notPlayerTurn() : void
+ isPlayerTurn() : void
+ removeToken(Token) : void
consist
actions
<<interface>>
Action
+execute(Token, Position, Position):Boolean
implements
Place
+execute(Token, Position, Position):Boolean
Slide
+execute(Token, Position, Position):Boolean
Jump
+execute(Token, Position, Position):Boolean
Remove
+execute(Token, Position, Position):Boolean
Game
- board: Board
- mode: Mode
+ getBoard() : Board
+ getMode() : Mode
+ run(): void
create
Display
- game : Game
+ start(): void
create
Home
- player1Name: String
- player2Name: String
+ start(): void
+ playerForm(): void
+ letterTransition(): void
Board
- position : Position [][]
- CIRCLE_RADIUS : int
- gameBoard : Group
- board : Board
- ip : ArrayList<Circle>
- positions : ArrayList <Position>
+ getInstance(): Board
+ setMillPositions(): void
+ getGameBoard() : Group
+ setIPLocation() : void
+ createLine(int , int, ArrayList<Shape>)
+ createMiddleLine(int , int, ArrayList<Shape>)
+ generateBoard() : Group
+ setPositionAdjList() : void
+ getPositions() : ArrayList<Position>
Token
- tokenPosition: Position
- hasPosition : Boolean
- isTokenAllow : Boolean
- TOKEN_RADIUS: int
- token : Circle
+ setTokenPosition(Position): void
+ setIsTokenAllow(Boolean): void
+ getHasPosition(): Boolean
+ getIsTokenAllow() : Boolean
+ getPosition() : Position
+ getCircle() : Circle
create
Position
- ip : Circle
- token: Token
- isTokenHere : boolean
- adjList: ArrayList<Position>
+ addAdjList(Position): void
+ getAdjList(): ArrayList<Position>
+ getIsTokenHere(): Boolean
+ addToken(Token): void
+ removeToken() : void
+ getToken() : Token
+ getIP() : Circle
consist
24..*
24
Rule
- millPositions : HashMap <Position, ArrayList<ArrayList<Position>>>
- positionHasAMill : ArrayList<ArrayList<Position>>
- posHasAMill : ArrayList<Position>
- hasAMill : Boolean
+setMillPositions(position, ArrayList <ArrayList<Position>>) : void
+ checkPlayerHasAMill(Position, Token) : Boolean
+ getPosHasAMill() : ArrayList <Position>
+ addPositionHasAMill(ArrayList<Position> list) : void
+ checkPositionsHasAMIll(Position) : Boolean
+ removePositionsHasAMill(Position) : void
+ setHasAMill(Boolean): void
+ getHasAMill() : Boolean
+ checkAllTokenMillPositions(Player) :Boolean
+ endGame(Player) : Boolean
MakeTokenMovable
- startx: double
- starty: double
- initx: double
- inity: double
- token : Token
- player : Player
- isTokenRemoved : Boolean
+ makeTokenDraggable(Node) : void
+ allowTokenReleased(Node) : void
+ millMessage(): void
Main
+ main(): args

Sprint 4 Revised Class Diagram


game
actors
actions
ResetPlayerTurn
- mode: Mode
- p1: Player
- p2: Player
+ setMode() : void
+ setPlayer1() : void
+ setPlayer2() : void
+ resetPlayersTurn(Player) : void
+ resetPlayerHasAMill(Player) : void
+ changeTokenColor(): void
+ checkAllTokenMillPositions(Player) :Boolean
+ endPlayerGame() : void
<<interface>>
Mode
+ run() : void
+ getP1() : Player
+ getP2() : Player
+ getP1Label() : Label
+ getP1Label() : Label
DoublePlayer
- playerFont : Font
- p1: Player
- p2: Player
- p1Label : Label
- p2Label : Label
SinglePlayer
- playerFont : Font
- p1: Player
- p2: Player
- p1Label : Label
- p2Label : Label
+ initiateComputerMove() : void
+ showTransition(Token, double, double , double , double)
+ fadeOutTransition (Token) : void
Player
- name: String
- tokens : ArrayList<Token>
- allowableAction : Action
- allTokensPlaced : Boolean
+ getName(): String
+ getTokenAt(int): Token
+ getTokenSize() : int
+ notPlayerTurn() : void
+ isPlayerTurn() : void
+ removeToken(Token) : void
<<interface>>
Action
+execute(Token, Position, Position):Boolean
Place
+execute(Token, Position, Position):Boolean
Slide
+execute(Token, Position, Position):Boolean
Jump
+execute(Token, Position, Position):Boolean
Remove
+execute(Token, Position, Position):Boolean
Game
- board: Board+ initiateComputerMove() : void
- mode: Mode
+ getBoard() : Board
+ getMode() : Mode
+ run(): void
+ reset(): void
Display
- game : Game
+ start(): void
+ exitConfirmation(): boolean
Home
- player1Name: String
- player2Name: String
+ start(): void
+ playerForm(): void
+ letterTransition(): void
Main
+ main(): args
Board
- position : Position [][]
- CIRCLE_RADIUS : int
- gameBoard : Group
- board : Board
- ip : ArrayList<Circle>
- positions : ArrayList <Position>
+ getInstance(): Board
+ setMillPositions(): void
+ getGameBoard() : Group
+ setIPLocation() : void
+ createLine(int , int, ArrayList<Shape>)
+ createMiddleLine(int , int, ArrayList<Shape>)
+ generateBoard() : Group
+ setPositionAdjList() : void
+ getPositions() : ArrayList<Position>
Token
- tokenPosition: Position
- hasPosition : Boolean
- isTokenAllow : Boolean
- TOKEN_RADIUS: int
- token : Circle
+ setTokenPosition(Position): void
+ setIsTokenAllow(Boolean): void
+ getHasPosition(): Boolean
+ getIsTokenAllow() : Boolean
+ getPosition() : Position
+ getCircle() : Circle
Position
- ip : Circle
- token: Token
- isTokenHere : boolean
- adjList: ArrayList<Position>
+ addAdjList(Position): void
+ getAdjList(): ArrayList<Position>
+ getIsTokenHere(): Boolean
+ addToken(Token): void
+ removeToken() : void
+ getToken() : Token
+ getIP() : Circle
MakeTokenMovable
- startx: double
- starty: double
- initx: double
- inity: double
- token : Token
- player : Player
- isTokenRemoved : Boolean
+ makeTokenDraggable(Node) : void
+ allowTokenReleased(Node) : void
+ millMessage(): void
Rule
- millPositions : HashMap <Position, ArrayList<ArrayList<Position>>>
- positionHasAMill : ArrayList<ArrayList<Position>>
- posHasAMill : ArrayList<Position>
- hasAMill : Boolean
+setMillPositions(position, ArrayList <ArrayList<Position>>) : void
+ checkPlayerHasAMill(Position, Token) : Boolean
+ getPosHasAMill() : ArrayList <Position>
+ addPositionHasAMill(ArrayList<Position> list) : void
+ checkPositionsHasAMill(Position) : Boolean
+ removePositionsHasAMill(Position) : void
+ setHasAMill(Boolean): void
+ getHasAMill() : Boolean
+ checkAllTokenMillPositions(Player) :Boolean
+ endGame(Player) : Boolean
+ setSpMode(Boolean) : void
+ initiateCompMove() : void
has
implements
implements
create
consist
24..*
24
9
2
1

User Stories

In order to meet the advanced requirement of enabling a single player to play against the computer, with the computer playing a move randomly among all currently valid moves, or based on a selected set of heuristics (advanced requirement C), our team has utilised user story 17. This user story outlines the necessary features and functionality required to support this requirement, and serves as a guide for the implementation of this feature. By implementing user story 17, we can ensure that the advanced requirement is fully met and that the resulting software meets the desired specifications and expectations. All of the following user stories are written such that they follow the INVEST criteria.

Changes to user stories:

1. A new user story (No. 1) was added in order to choose the mode of the game.
2. A new user story (No. 7) was added in order for the player to be able to remove a token from the mill of the opponent player for the game to continue.
3. A new user story (No. 8) was added so that the player is alerted when a mill is formed.

Table 1: User Stories For Nine Men's Morris

No.	**User Story**
1	As a player, I want to be able to choose the mode before I start the game so that I can play with another person or with the computer
2	As a game player, I want to be able to enter my name before I start playing so that it is easier to see which tokens belong to me.
3	As a game player, I want to be able to select my pieces on the board by clicking or tapping on the desired location so that I know which token I have selected.
4	As a game player, I want to be able to move my tokens from one position to another when it's my turn, so that the game can continue.
5	As a game player, I want to be able to arrange my tokens on the game board, so that I can continue the game.
6	As a game player, I want to have a mill by forming a straight row of three pieces along one of the board's lines to eliminate one piece of the opponent token.
7	As a player, if I have a mill formed, I want to remove a token from the mill of the opponent if there are no isolated tokens to be removed so that the game can continue.
8	As a player, I want to be informed if I have formed a mill so that I can remove a token from the opponent player
9	As a game player, I want to win the game by removing the opponent's tokens until there are fewer than two pieces.

10	As a game player, I want to be able to see the remaining pieces for both players so that I can track the progress of the game.
11	As a game player, I want to be able to exit the game if I feel like I cannot win, so that I do not have to continue playing until the end.
12	As a game rule, each player should be able to start moving their tokens without following the line of the board when they are left with three pieces.
13	As a game rule, players should move their tokens according to the line of the board for each round to ensure no illegal moves were taken.
14	As a game rule, each player should be given 9 pieces of tokens before the start of the game, so that the game can be played fairly.
15	As a game rule, the player should not be able to take any pieces from a mill of the opponent, so that it allows the game to progress.
16	As a gameboard, I want to ensure that no illegal moves are made so that a fair game can be played.
17	**(Advanced Requirement)** As a game player, I want to be able to play against a computer opponent so that I can play the game against another real player.

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. The game logic for how the computer will move a token will be stored in this class since it is responsible for player versus computer game logic.

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. A new method was created for the user to exit the current game if they no longer want to continue playing.

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. This class also handles checking if the game is a single player mode.

Additionally, this class also has a dependency relationship on the SinglePlayer class , to hold the computer game logic.

ResetPlayerTurn
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
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. This class also handle the automated move of the token from the computer

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
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
Main class is added to run the whole game application.

Design Architecture

In this sprint , we briefly revise our architecture to cope with the advance requirement we chose. Since we chose to allow the player to play games with the computer , we added some methods to take care of the computer game logic. A SinglePlayer class was created since sprint 1 as we have never changed the choice of the advance requirement. Hence, the SinglePlayer class holds the computer game logic and ensures the game to be played in a single player mode to allow all the classes to be played in single player mode.

In the revised class diagram, there are a few methods added to handle the computer logic. The SinglePlayer class holds how the computer game logic will work and then the Rule class will be informed it is a single player game , and added methods to initiate the computer move. So we can apply checking in the MakeTokenMovable class to ensure it is a single player game and can safely trigger a computer move using the Rule class.

Since we didn't really change on the design architecture, our design architecture will be similar as Sprint 3:

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. (For advanced requirement)

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.

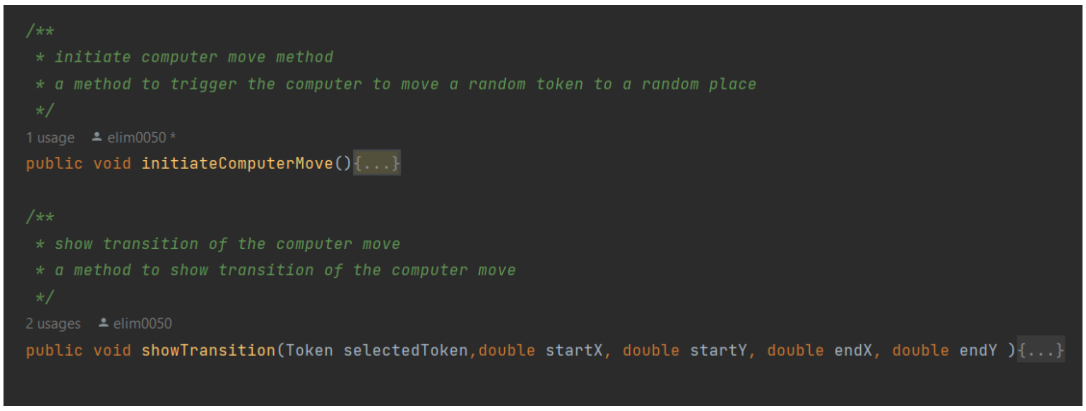
Advance Requirement

We chose to allow a single player to play against the computer, in which the computer will generate random moves on random tokens on the board. We decided this requirement since sprint 1 and have never changed. Implementing the advance requirement requires adding methods to different classes in order to handle the computer game logic, so that random tokens can be generated to random positions on the board.

As mentioned above, we only added code to the current existing classes and rarely modify the previous code. As we have considered implementing the same requirement since sprint 1, we created the design structure that allows us to easily add code in the future. This shows that all of our classes have carefully followed the Open-Closed Principle, which permits us to extend our code without modifying the inner structure. Moreover, all the classes are not highly dependent on each other, hence implementing this requirement is not considered as difficult, it is fairly simple to implement it by adding the computer game logic in the SinglePlayer class and ensuring all classes are informed in a single player mode.

<u>Methods added</u>

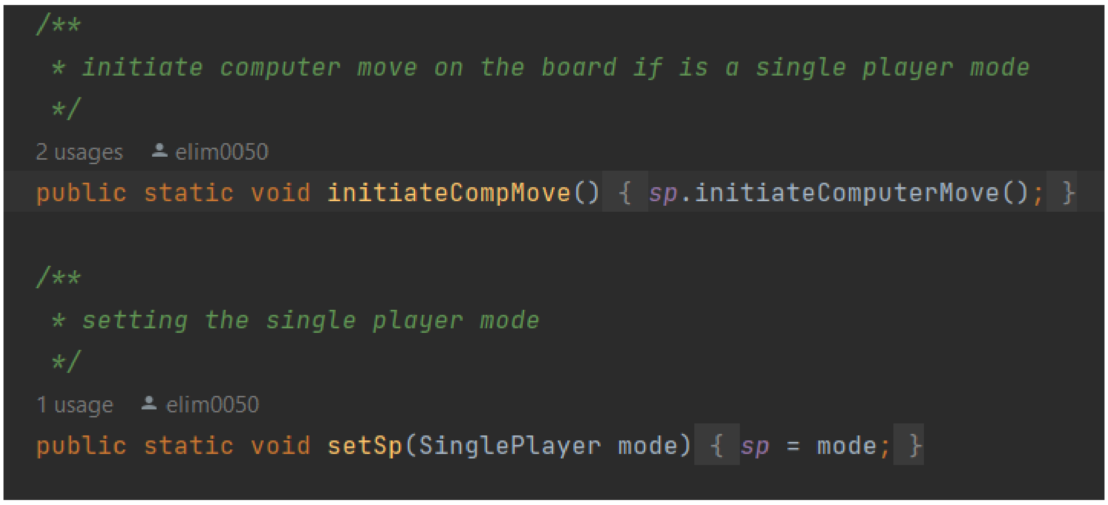
SinglePlayer class hold the method to initiate a computer token move on the board :



```java
/**
 * initiate computer move method
 * a method to trigger the computer to move a random token to a random place
 */
1 usage  elim0050 *
public void initiateComputerMove(){...}

/**
 * show transition of the computer move
 * a method to show transition of the computer move
 */
2 usages  elim0050
public void showTransition(Token selectedToken,double startX, double startY, double endX, double endY ){...}
```

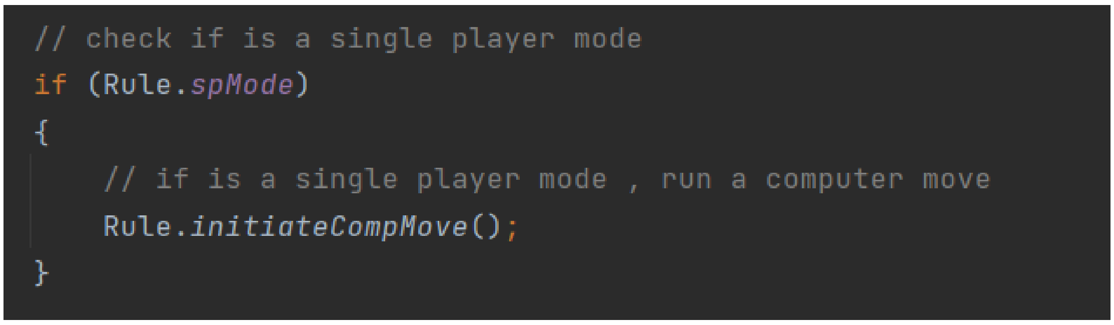
Rule class hold the method on initiate the computer move:



```java
/**
 * initiate computer move on the board if is a single player mode
 */
2 usages  elim0050
public static void initiateCompMove() { sp.initiateComputerMove(); }

/**
 * setting the single player mode
 */
1 usage  elim0050
public static void setSp(SinglePlayer mode) { sp = mode; }
```

Check in the MakeTokenMovable Class:

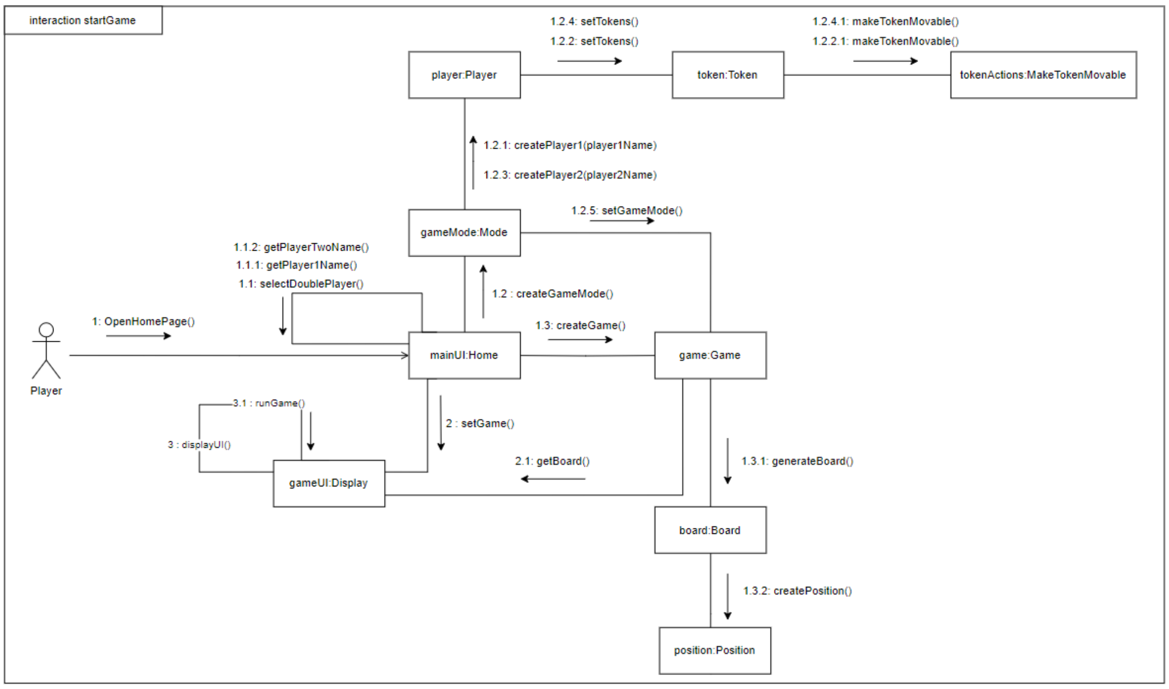


```java
// check if is a single player mode
if (Rule.spMode)
{
    // if is a single player mode , run a computer move
    Rule.initiateCompMove();
}
```

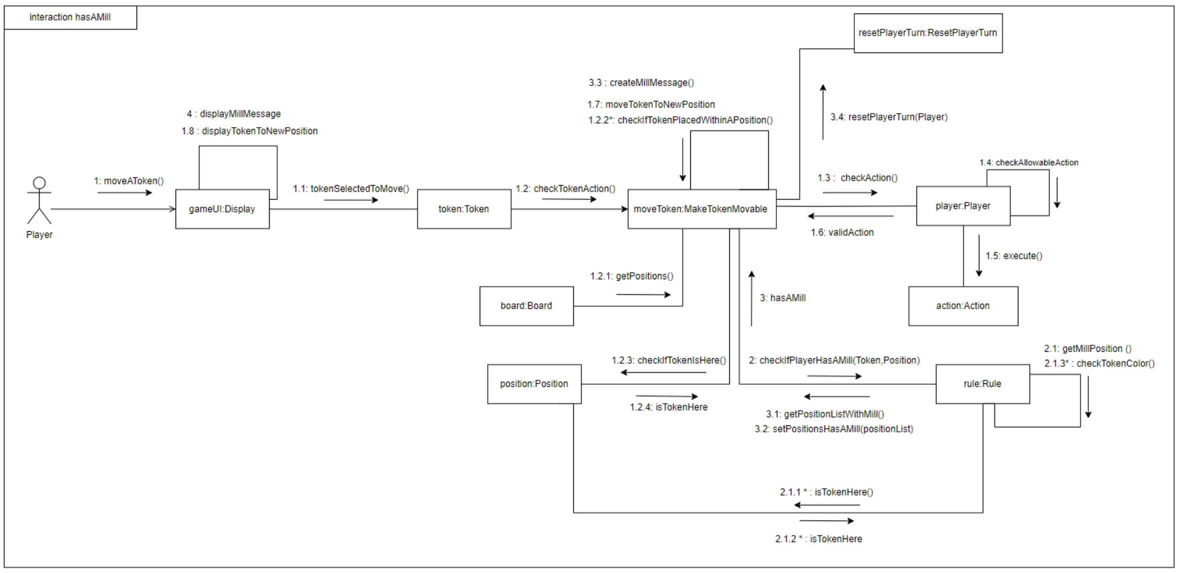
Therefore, we extend the code in the SinglePlayer class and let the Rule class hold the method in order to be called in the MakeTokenMovable class. Hence the MakeTokenMovable class does not need to further depend on the SinglePlayer class to call the method. This reduces the complex dependency between classes.

Communication Diagram

Start of the game :

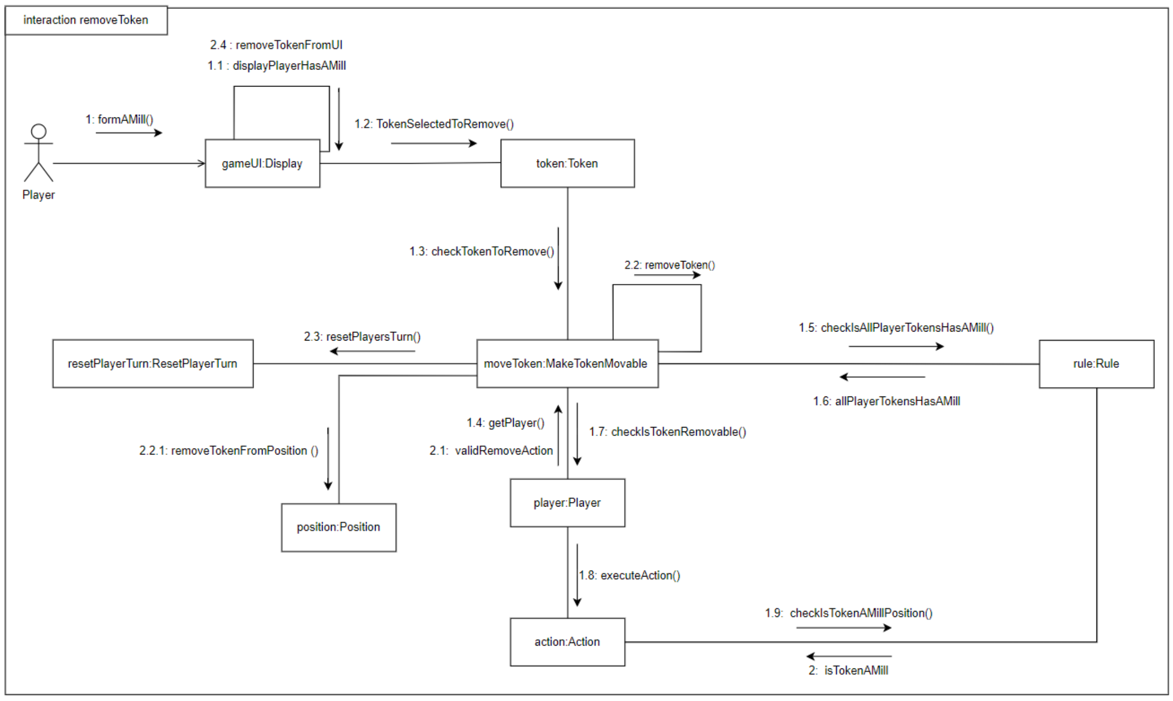


Player Form A Mill On Board :

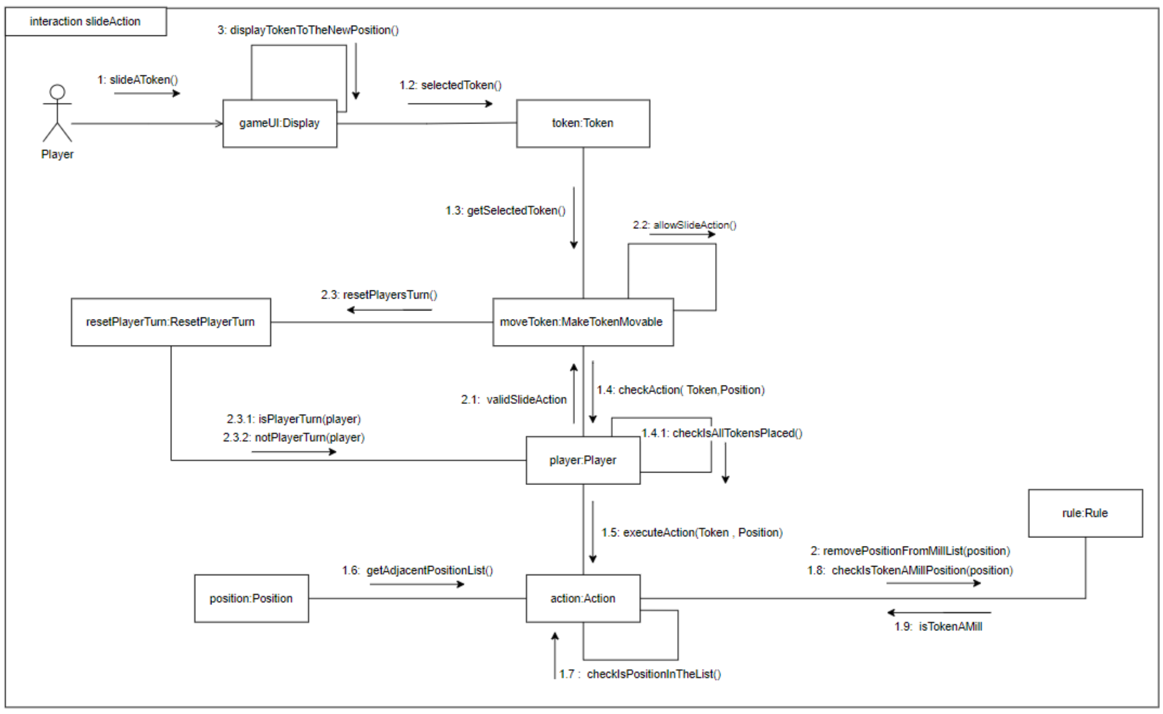


interaction hasAMill
resetPlayerTurn:ResetPlayerTurn
3.3 : createMillMessage()
1.7: moveTokenToNewPosition
1.2.2*: checkIfTokenPlacedWithinAPosition()
3.4: resetPlayerTurn(Player)
4 : displayMillMessage
1.8 : displayTokenToNewPosition
1.4: checkAllowableAction
1: moveAToken()
1.1: tokenSelectedToMove()
1.2: checkTokenAction()
1.3 : checkAction()
Player
gameUI:Display
token:Token
moveToken:MakeTokenMovable
player:Player
1.6: validAction
1.5: execute()
1.2.1: getPositions()
board:Board
3: hasAMill
action:Action
2.1: getMillPosition ()
2.1.3* : checkTokenColor()
1.2.3: checkIfTokenIsHere()
2: checkIfPlayerHasAMill(Token,Position)
position:Position
rule:Rule
1.2.4: isTokenHere
3.1: getPositionListWithMill()
3.2: setPositionsHasAMill(positionList)
2.1.1 * : isTokenHere()
2.1.2 * : isTokenHere

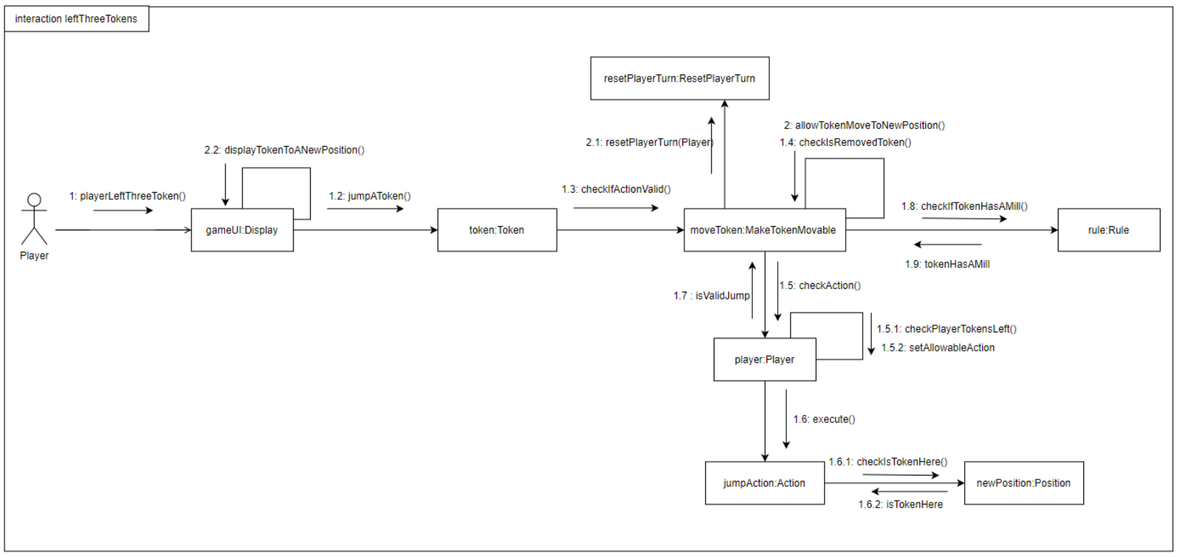
Player select opponent player token to remove :


interaction removeToken
2.4 : removeTokenFromUI
1.1 : displayPlayerHasAMill
1: formAMill()
Player
gameUI:Display
1.2: TokenSelectedToRemove()
token:Token
1.3: checkTokenToRemove()
2.2: removeToken()
2.3: resetPlayersTurn()
resetPlayerTurn:ResetPlayerTurn
moveToken:MakeTokenMovable
1.5: checkIsAllPlayerTokensHasAMill()
rule:Rule
1.6: allPlayerTokensHasAMill
1.4: getPlayer()
2.1: validRemoveAction
1.7: checkIsTokenRemovable()
2.2.1: removeTokenFromPosition ()
position:Position
player:Player
1.8: executeAction()
1.9: checkIsTokenAMillPosition()
action:Action
2: isTokenAMill

Player Slide Token on the Board :


interaction slideAction
3: displayTokenToTheNewPosition()
1: slideAToken()
Player
gameUI:Display
1.2: selectedToken()
token:Token
1.3: getSelectedToken()
2.2: allowSlideAction()
2.3: resetPlayersTurn()
resetPlayerTurn:ResetPlayerTurn
moveToken:MakeTokenMovable
2.1: validSlideAction
1.4: checkAction(Token,Position)
2.3.1: isPlayerTurn(player)
2.3.2: notPlayerTurn(player)
1.4.1: checkIsAllTokensPlaced()
player:Player
rule:Rule
1.5: executeAction(Token , Position)
2: removePositionFromMillList(position)
1.8: checkIsTokenAMillPosition(position)
position:Position
1.6: getAdjacentPositionList()
action:Action
1.9: isTokenAMill
1.7 : checkIsPositionInTheList()

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


interaction leftThreeTokens
resetPlayerTurn:ResetPlayerTurn
2.1: resetPlayerTurn(Player)
2: allowTokenMoveToNewPosition()
1.4: checkIsRemovedToken()
2.2: displayTokenToANewPosition()
1: playerLeftThreeToken()
1.2: jumpAToken()
1.3: checkIfActionValid()
1.8: checkIfTokenHasAMill()
Player
gameUI:Display
token:Token
moveToken:MakeTokenMovable
rule:Rule
1.9: tokenHasAMill
1.7 : isValidJump
1.5: checkAction()
1.5.1: checkPlayerTokensLeft()
1.5.2: setAllowableAction
player:Player
1.6: execute()
1.6.1: checkIsTokenHere()
jumpAction:Action
newPosition:Position
1.6.2: isTokenHere