FIT3077 Sprint Three Report

Team: MA_Friday2pm_Team10
Team Name: The Algorithm Alchemists

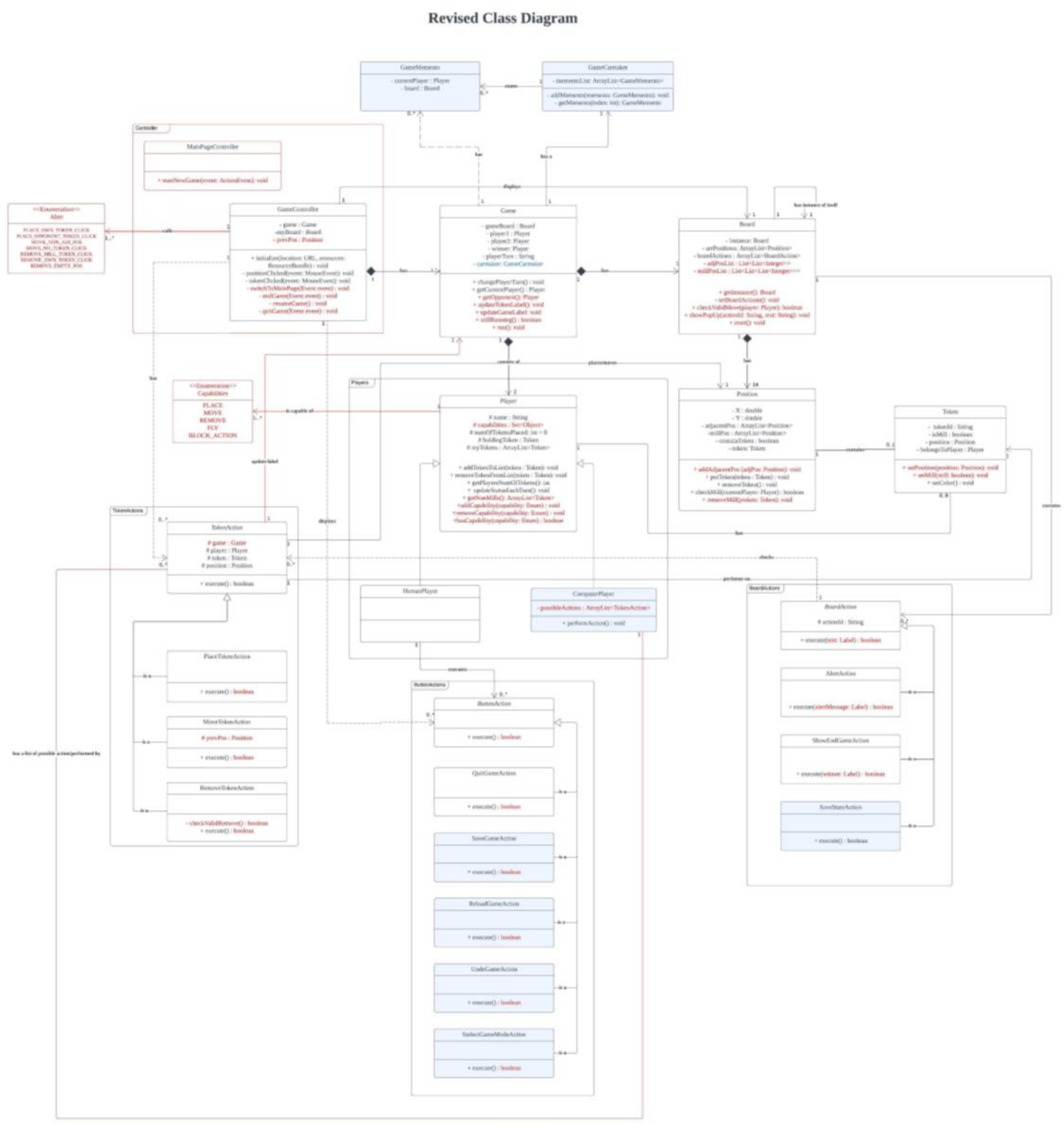
Team Members:
Lai Qui Juin 32638809
Leong Yen Ni 32023685
Lim Jia Ying 32423063
Lo Kin Herng 32023995

Table of Contents

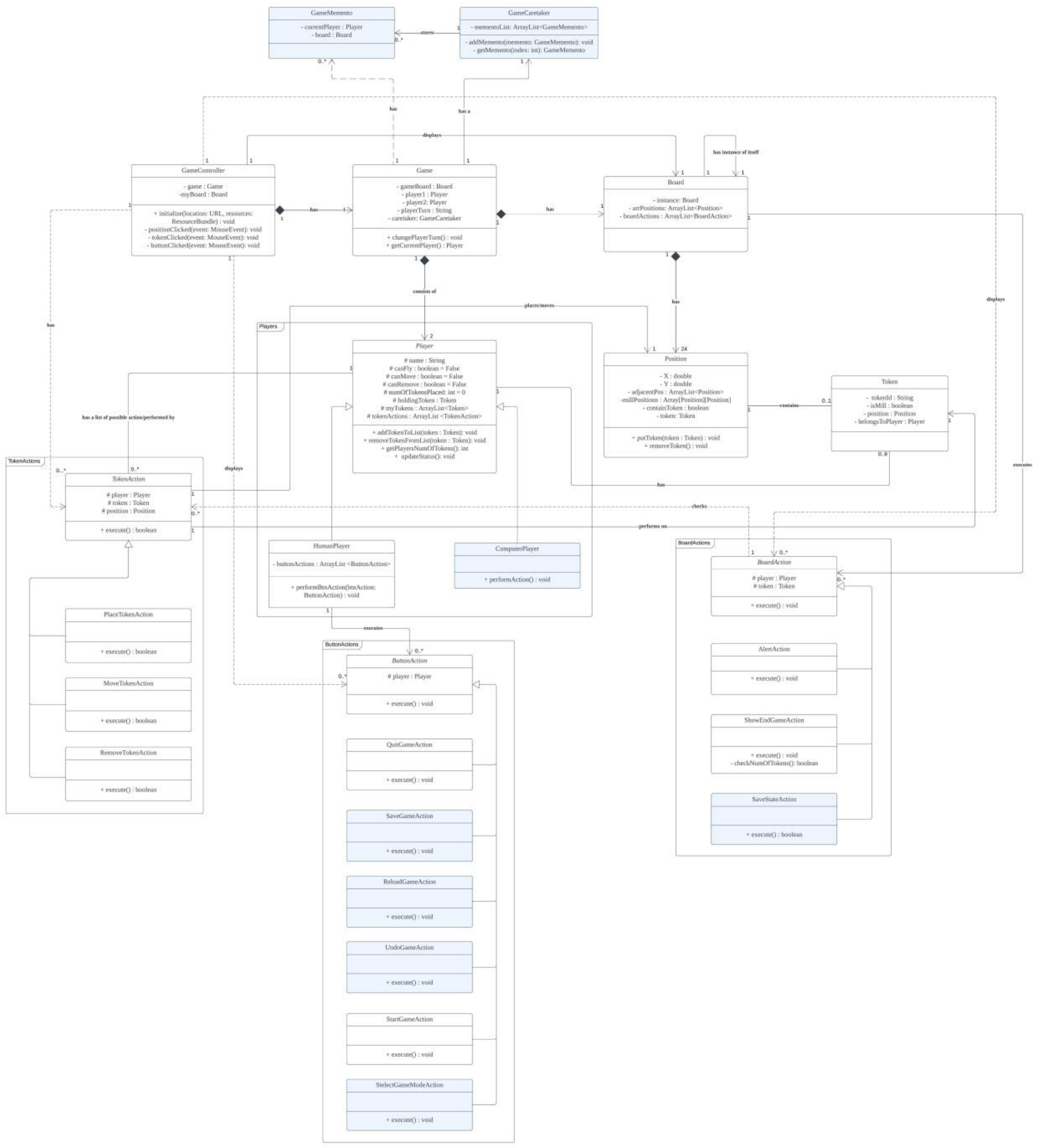
Revised Class Diagram

https://lucid.app/lucidchart/e66943cd-742a-4ce3-8dd3-ba4812ec3b4b/edit?viewport_loc=288%2C-6313%2C9658%2C3888%2CpueX~aQuz0TP&invitationId=inv_52e98539-aab1-4b23-8d36-c80cdd648ae7

1. Sprint 3 Revised Class Diagram



2. Sprint 2 Class Diagram (For comparison purposes)

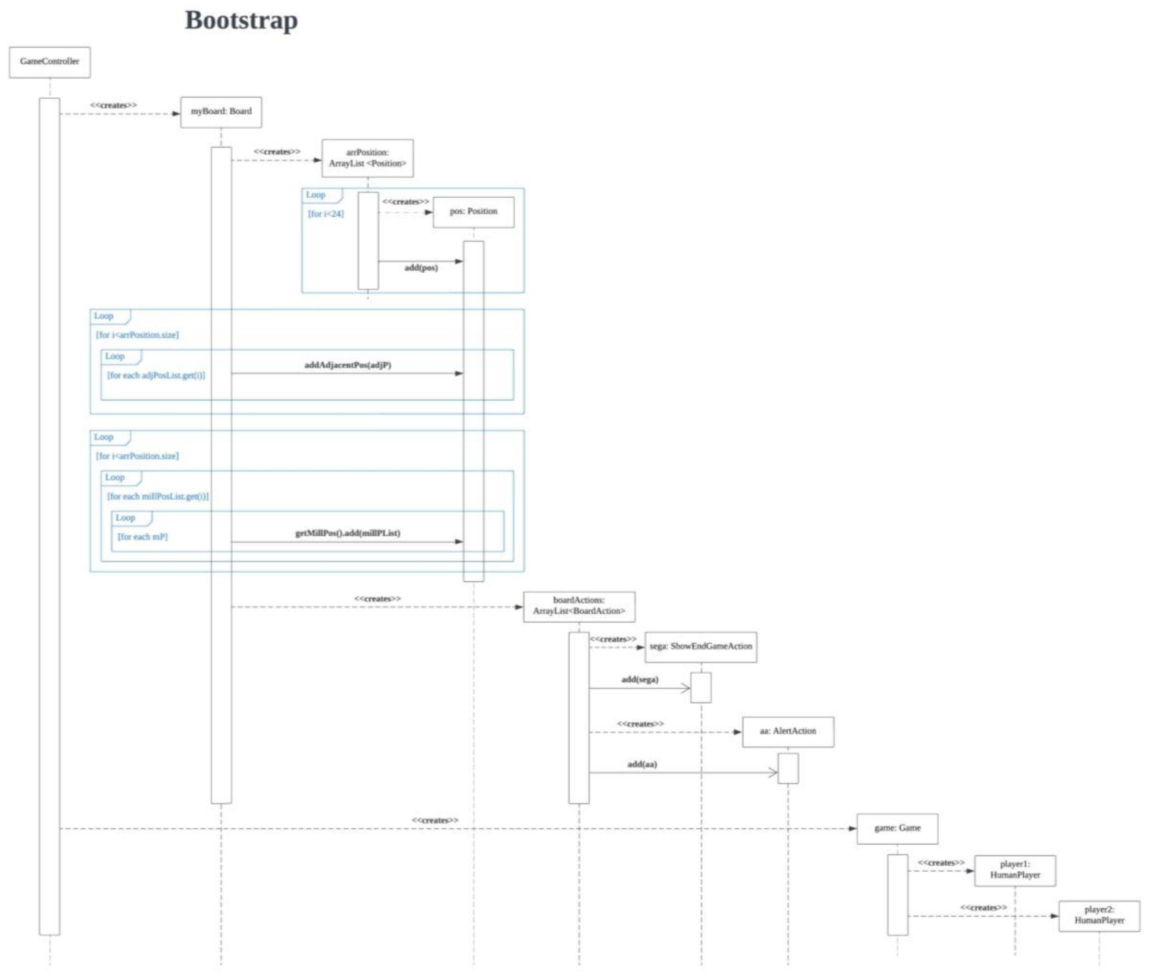


Sequence Diagrams

https://lucid.app/lucidchart/e66943cd-742a-4ce3-8dd3-ba4812ec3b4b/edit?viewport_loc=288%2C-6313%2C9658%2C3888%2CpueX~aQuz0TP&invitationId=inv_52e98539-aab1-4b23-8d36-c80cdd648ae7

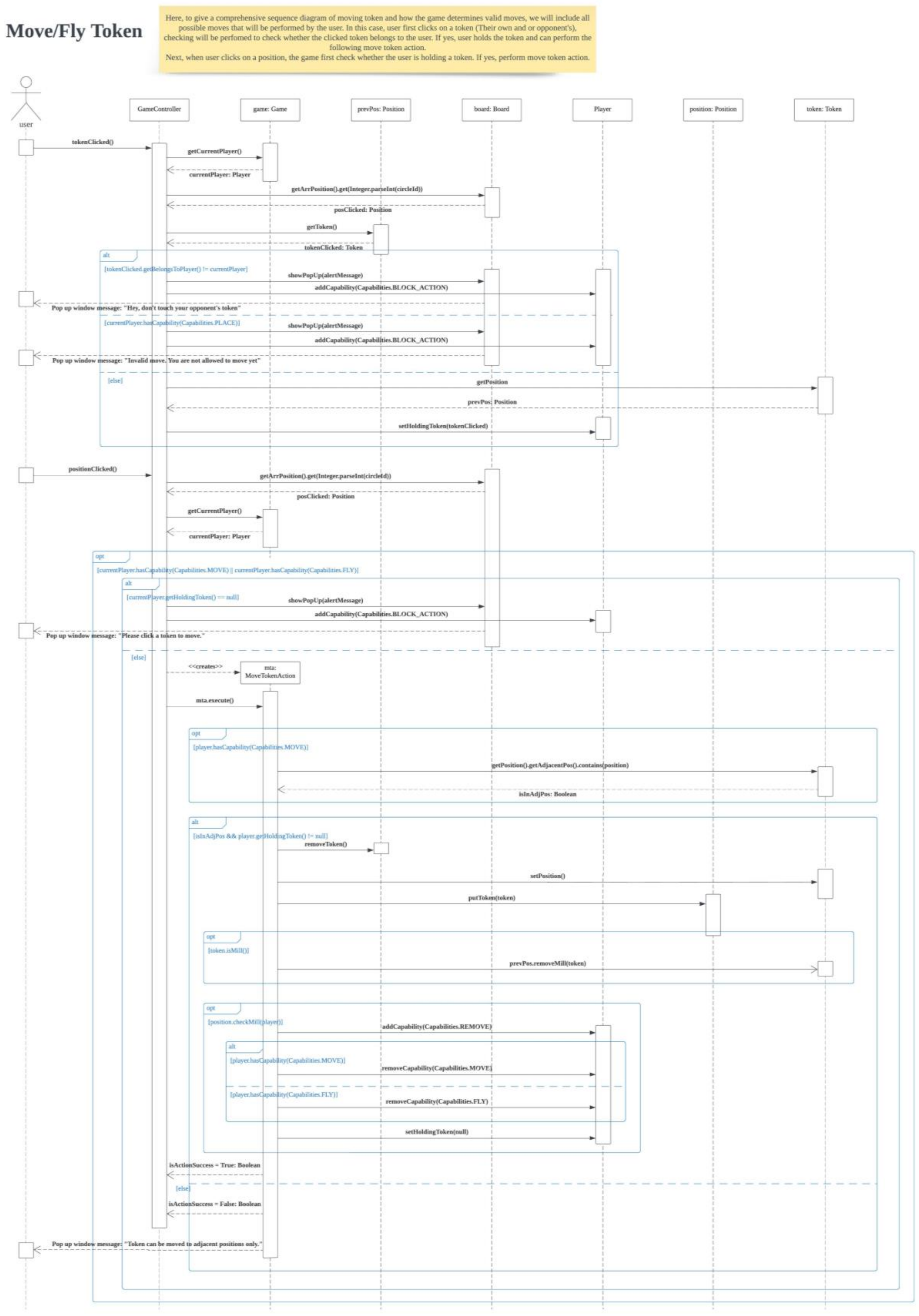
We have created a total of five sequence diagrams:

1. An illustration of the bootstrap (or initialisation) process of the 9MM game



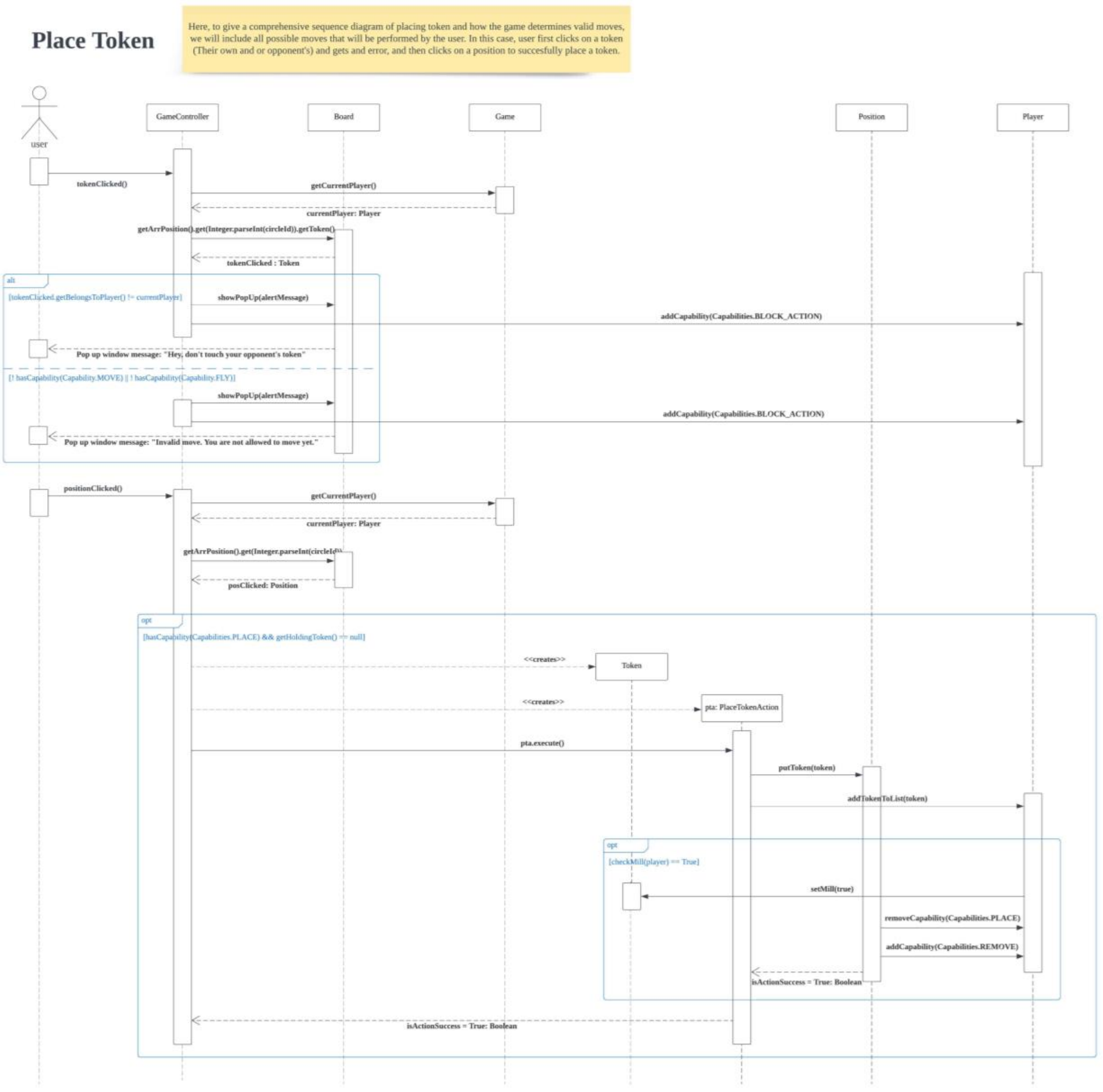
2. An illustration of four different scenarios:

a. Move/fly token


Here, to give a comprehensive sequence diagram of moving token and how the game determines valid moves, we will include all possible moves that will be performed by the user. In this case, user first clicks on a token (Their own and or opponent's), checking will be perfomed to check whether the clicked token belongs to the user. If yes, user holds the token and can perform the following move token action.
Next, when user clicks on a position, the game first check whether the user is holding a token. If yes, perform move token action.

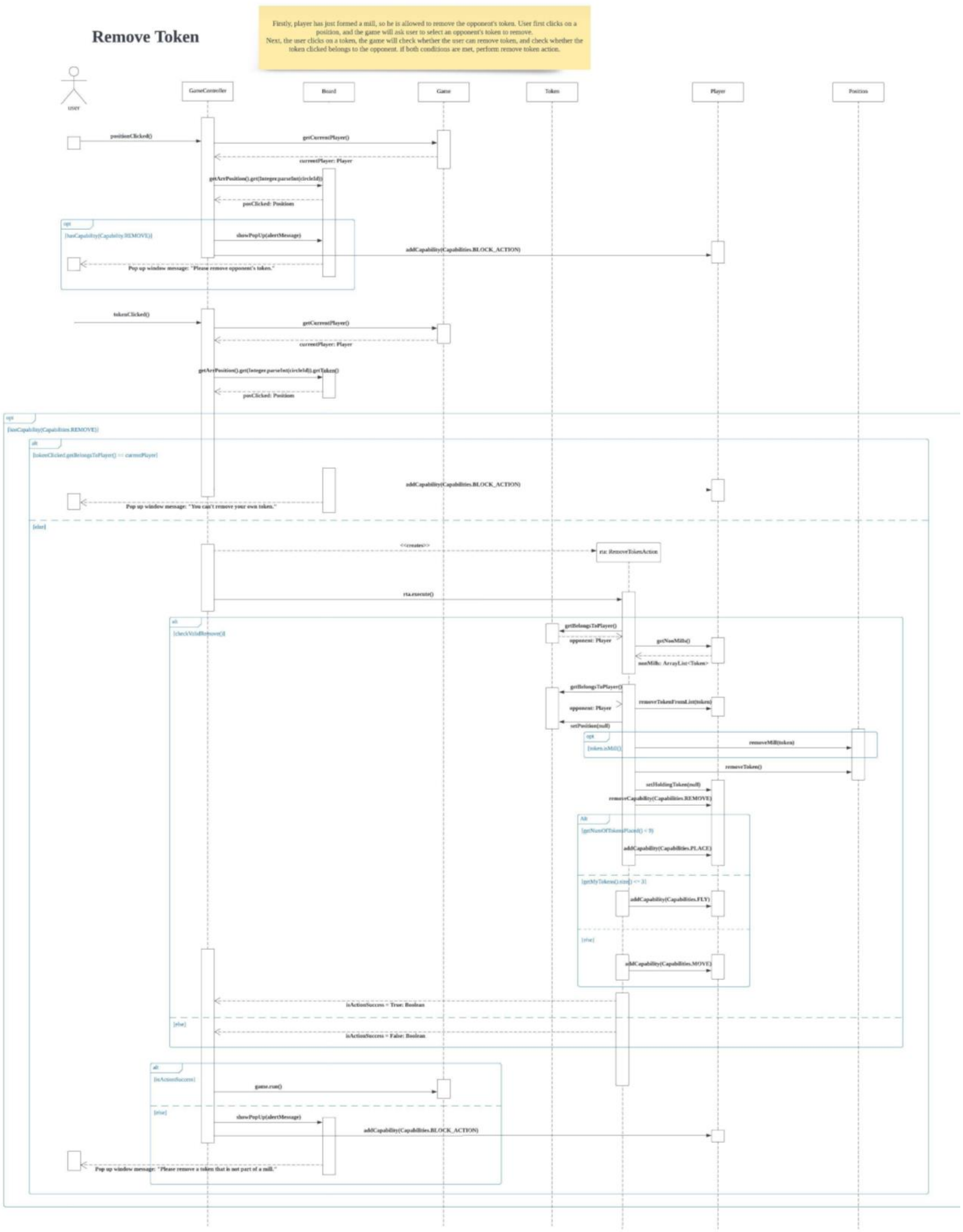
Move/Fly Token
user
GameController
game: Game
prevPos: Position
board: Board
Player
position: Position
token: Token
tokenClicked()
getCurrentPlayer()
currentPlayer: Player
getArrPosition().get(Integer.parseInt(circleId))
posClicked: Position
getToken()
tokenClicked: Token
alt
[tokenClicked.getBelongsToPlayer() != currentPlayer]
showPopUp(alertMessage)
addCapability(Capabilities.BLOCK_ACTION)
Pop up window message: "Hey, don't touch your opponent's token"
[currentPlayer.hasCapability(Capabilities.PLACE)]
showPopUp(alertMessage)
addCapability(Capabilities.BLOCK_ACTION)
Pop up window message: "Invalid move. You are not allowed to move yet"
[else]
getPosition
prevPos: Position
setHoldingToken(tokenClicked)
positionClicked()
getArrPosition().get(Integer.parseInt(circleId))
posClicked: Position
getCurrentPlayer()
currentPlayer: Player
opt
[currentPlayer.hasCapability(Capabilities.MOVE) || currentPlayer.hasCapability(Capabilities.FLY)]
alt
[currentPlayer.getHoldingToken() == null]
showPopUp(alertMessage)
addCapability(Capabilities.BLOCK_ACTION)
Pop up window message: "Please click a token to move."
[else]
<<creates>>
mta: MoveTokenAction
mta.execute()
opt
[player.hasCapability(Capabilities.MOVE)]
getPosition().getAdjacentPos().contains(position)
isInAdjPos: Boolean
alt
[isInAdjPos && player.getHoldingToken() != null]
removeToken()
setPosition()
putToken(token)
opt
[token.isMill()]
prevPos.removeMill(token)
opt
[position.checkMill(player)]
addCapability(Capabilities.REMOVE)
alt
[player.hasCapability(Capabilities.MOVE)]
removeCapability(Capabilities.MOVE)
[player.hasCapability(Capabilities.FLY)]
removeCapability(Capabilities.FLY)
setHoldingToken(null)
isActionSuccess = True: Boolean
[else]
isActionSuccess = False: Boolean
Pop up window message: "Token can be moved to adjacent positions only."

b. Place token


Here, to give a comprehensive sequence diagram of placing token and how the game determines valid moves, we will include all possible moves that will be performed by the user. In this case, user first clicks on a token (Their own and or opponent's) and gets and error, and then clicks on a position to succesfully place a token.

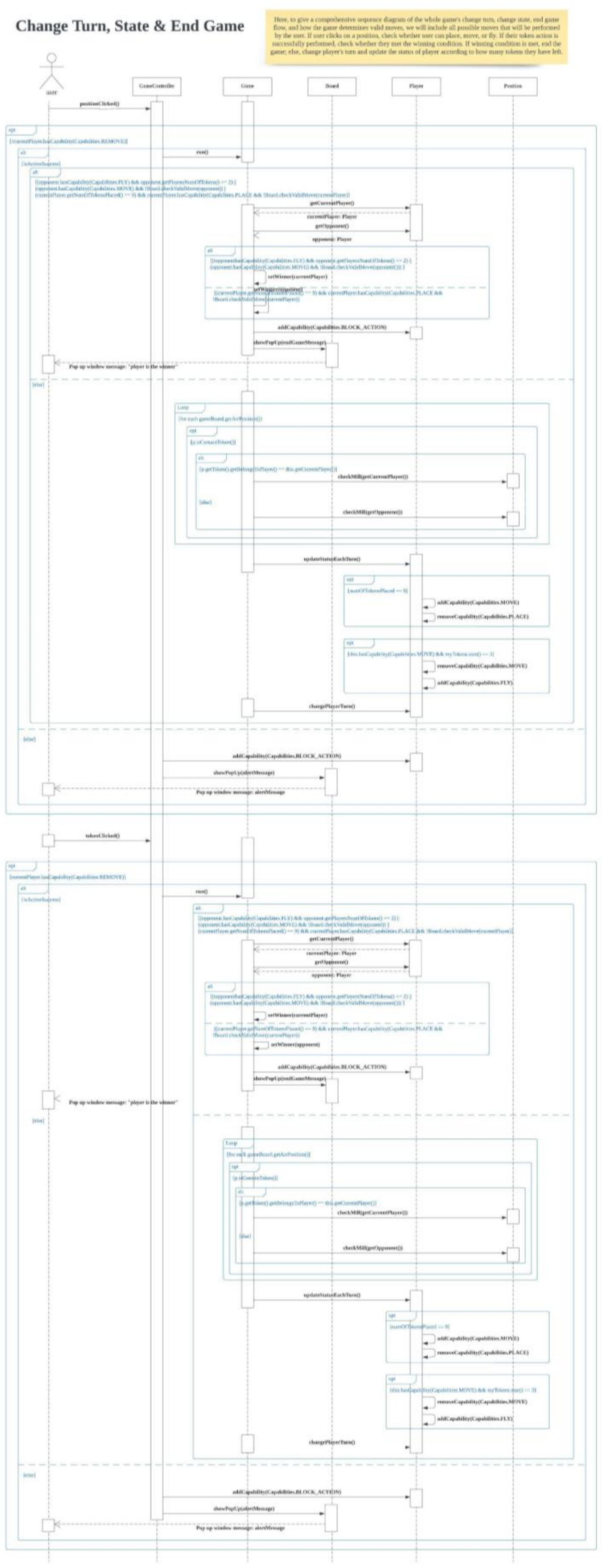
Place Token
user
GameController
Board
Game
Position
Player
tokenClicked()
getCurrentPlayer()
currentPlayer: Player
getArrPosition().get(Integer.parseInt(circleId)).getToken()
tokenClicked : Token
alt
[tokenClicked.getBelongsToPlayer() != currentPlayer]
showPopUp(alertMessage)
addCapability(Capabilities.BLOCK_ACTION)
Pop up window message: "Hey, don't touch your opponent's token"
[! hasCapability(Capability.MOVE) || ! hasCapability(Capability.FLY)]
showPopUp(alertMessage)
addCapability(Capabilities.BLOCK_ACTION)
Pop up window message: "Invalid move. You are not allowed to move yet."
positionClicked()
getCurrentPlayer()
currentPlayer: Player
getArrPosition().get(Integer.parseInt(circleId
posClicked: Position
opt
[hasCapability(Capabilities.PLACE) && getHoldingToken() == null]
<<creates>>
Token
<<creates>>
pta: PlaceTokenAction
pta.execute()
putToken(token)
addTokenToList(token)
opt
[checkMill(player) == True]
setMill(true)
removeCapability(Capabilities.PLACE)
addCapability(Capabilities.REMOVE)
isActionSuccess = True: Boolean
isActionSuccess = True: Boolean

c. Remove token


Remove Token
user
GameController
Board
Game
Token
Player
Position
positionClicked()
getCurrentPlayer()
currentPlayer: Player
getArrPosition().get(Integer.parseInt(circleId))
posClicked: Position
opt
[!hasCapability(Capability.REMOVE)]
showPopUp(alertMessage)
addCapability(Capabilities.BLOCK_ACTION)
Pop up window message: "Please remove opponent's token."
tokenClicked()
getCurrentPlayer()
currentPlayer: Player
getArrPosition().get(Integer.parseInt(circleId)).getToken()
posClicked: Position
opt
[hasCapability(Capabilities.REMOVE)]
alt
[tokenClicked.getBelongsToPlayer() == currentPlayer]
addCapability(Capabilities.BLOCK_ACTION)
Pop up window message: "You can't remove your own token."
[else]
<<creates>>
rta: RemoveTokenAction
rta.execute()
alt
[checkValidRemove()]
getBelongsToPlayer()
opponent: Player
getNonMills()
nonMills: ArrayList<Token>
getBelongsToPlayer()
opponent: Player
removeTokenFromList(token)
setPosition(null)
opt
[token.isMill()]
removeMill(token)
removeToken()
setHoldingToken(null)
removeCapability(Capabilities.REMOVE)
Alt
[getNumOfTokensPlaced() < 9]
addCapability(Capabilities.PLACE)
[getMyTokens().size() <= 3]
addCapability(Capabilities.FLY)
[else]
addCapability(Capabilities.MOVE)
isActionSuccess = True: Boolean
[else]
isActionSuccess = False: Boolean
alt
[isActionSuccess]
game.run()
[else]
showPopUp(alertMessage)
addCapability(Capabilities.BLOCK_ACTION)
Pop up window message: "Please remove a token that is not part of a mill."

Firstly, player has just formed a mill, so he is allowed to remove the opponent's token. User first clicks on a position, and the game will ask user to select an opponent's token to remove.
Next, the user clicks on a token, the game will check whether the user can remove token, and check whether the token clicked belongs to the opponent. if both conditions are met, perform remove token action.

d. Change Turn and Change State



Here, to give a comprehensive sequence diagram of the whole game's change turn, change state, end game flow, and how the game determines valid moves, we will include all possible moves that will be performed by the user. If user clicks on a position, check whether user can place, move, or fly. If their token action is successfully performed, check whether they met the winning condition. If winning condition is met, end the game; else, change player's turn and update the status of player according to how many tokens they have left.

Change Turn, State & End Game
user
GameController
Game
Board
Player
Position
positionClicked()
opt
[!currentPlayer.hasCapability(Capabilities.REMOVE)]
alt
run()
[isActionSuccess]
getCurrentPlayer()
currentPlayer: Player
getOpponent()
opponent: Player
setWinner(currentPlayer)
setWinner(opponent)
addCapability(Capabilities.BLOCK_ACTION)
showPopUp(endGameMessage)
Pop up window message: "player is the winner"
[else]
Loop
[for each gameBoard.getAllPositions()]
[p.isContainToken()]
[p.getToken().getBelongToPlayer() == this.getCurrentPlayer()]
checkMill(getCurrentPlayer())
checkMill(getOpponent())
updateStatusEachTurn()
[numOfTokensPlaced == 9]
addCapability(Capabilities.MOVE)
removeCapability(Capabilities.PLACE)
[this.hasCapability(Capabilities.MOVE) && myTokens.size() == 3]
removeCapability(Capabilities.MOVE)
addCapability(Capabilities.FLY)
changePlayerTurn()
showPopUp(alertMessage)
Pop up window message: alertMessage
tokenClicked()

Design Rationale

Revised architecture:

The **attributes and methods highlighted in red** in the class diagram are the elements that we have **added or changed,** whereas those in **blue** are the **advanced requirements.**

One major adjustment that we have made is about the capabilities of players to perform an action on their tokens. Instead of relying on the three separate booleans (canMove, canFly, canRemove) to keep track of the movement capabilities of each player, we decided to use a Capabilities Set. **An enum class**, **Capabilities**, is created to **store the constants of the capabilities of the players.** The reason for this is to **maintain the extensibility of the system**. If we were to add or modify movement capabilities in the future, using an enum class simplifies the process. A new constant can just be easily added into the class, instead of creating a new boolean variable in the Player class, which is in line with the **Open Closed Principle**. This also demonstrates a more concise representation of the movement capabilities as only one single variable in the Player class represents the entire set of capabilities.

Furthermore, we have introduced an additional **enum** class called **"Alert"** to **store all types of alerts associated with possible errors that the player will make** while playing the game. Just as mentioned earlier, employing an enum class facilitates seamless addition or removal of alerts in the future, **ensuring flexibility and ease of maintenance**. We will explain more about how the alert works in our architecture in the non-functional requirements section.

Moreover, in our previous architecture, the execute() method within each action class was designed to return no value. However, in this sprint, we made a modification to enhance functionality. We updated the **execute() method** to **return a boolean value**, **indicating whether the action was successfully executed or not.** The execute() method returns False when a Player attempts to perform an invalid action. This adjustment is made to align with our intention of implementing a popup window that alerts players when an invalid move is attempted.

Non-functional requirements:

1. Reliability:
   We have prioritised the **reliability** of our game to **ensure it operates consistently by following the rules and logic of the game.** When implementing the game, we have performed thorough testing to identify and fix any potential bugs or issues that could impact the execution of the game. Considering that players may not be familiar with the game, it is very likely for novice players to misclick or play against the game rules, therefore it is our responsibility to execute error handling for all possible scenarios as follows:

PLACING TOKEN	○ Player accidentally clicks on his own token ○ Player accidentally clicks on opponent’s token
MOVING TOKEN	○ Player tries to move to non-adjacent position when they can't fly ○ Player tries to move to non empty position ○ Player clicks on position before selecting a token ○ Player accidentally clicks on their opponent’s token
REMOVING TOKEN	○ Player tries to remove a token from opponent’s mill when opponent has other tokens which are not in a mill ○ Player accidentally clicks on their own tokens ○ Player forgets to remove a token, which they click on an empty position.

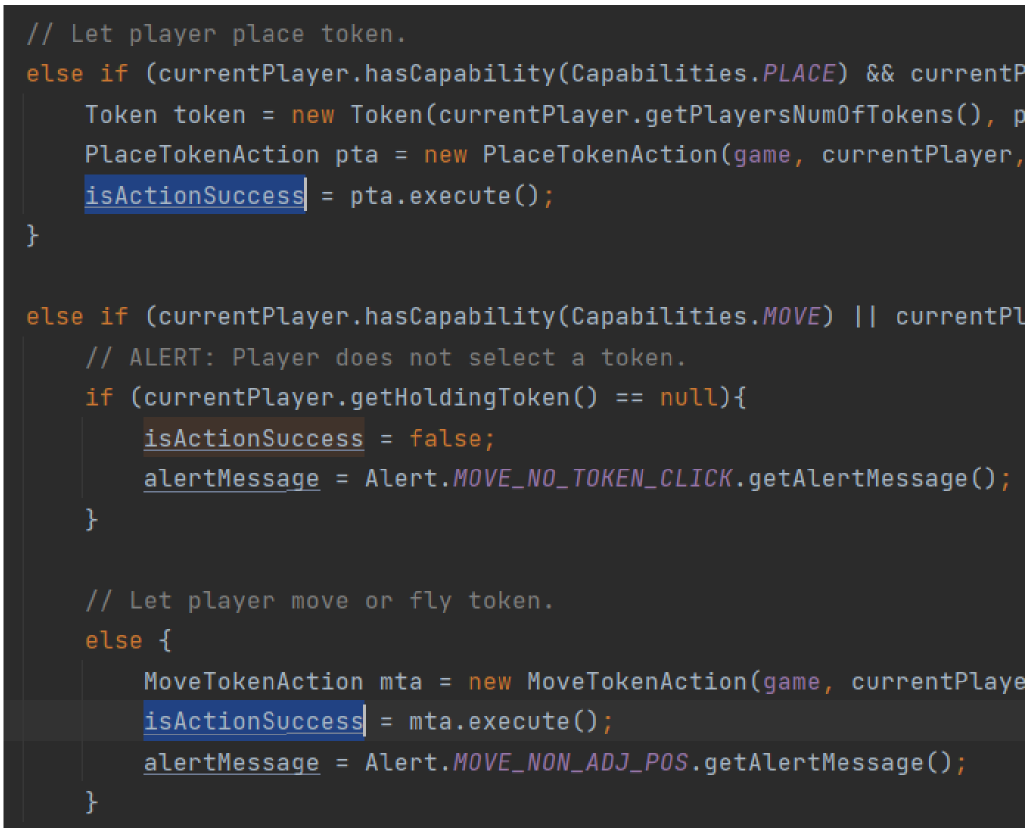
These errors are declared in the Alert class as mentioned in the revised architecture. **To handle them properly, a pop-up window with an alert message corresponding to the invalid move will appear in the middle of the screen.** This is implemented in the GameController class where the board will execute the AlertAction class to display the pop-up window. With that, players are informed about the errors made and can proceed to make the correct move. Screenshots are provided at the end of this report as demonstration of the alerts.

2. Maintainability
   Maintainability can be defined as the ability to maintain a system.This includes aspects such as updating and modifying the code. With this in mind, we have **adhered to OOP principles** such as SOLID and DRY when designing our game. We have included design patterns such as singleton for the Board class.

   Another aspect of maintainability that we focused on is **code readability**. We have achieved this by **providing documentation for all classes and methods.** We have also **included in-line comments** as well for code sections that are more complex.

   Moreover, we **apply modularity** by **grouping classes into modules** and **using functions to keep our code modular**. Using functions makes our code easier to read, understand and debug.

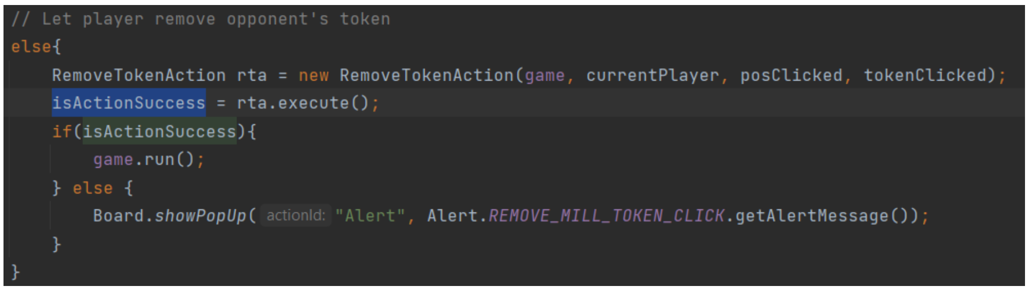
   Lastly, although we have 4 software developers on our team, we have tried to **use similar coding styles to make the code more consistent.** An example of this is the implementation of placing, moving and removing tokens. The game controller class consistently uses an “isActionSuccess” flag to decide if the move is valid, even though the code was designed by 3 different people.



```
// Let player place token.
else if (currentPlayer.hasCapability(Capabilities.PLACE) && currentP
    Token token = new Token(currentPlayer.getPlayersNumOfTokens(), p
    PlaceTokenAction pta = new PlaceTokenAction(game, currentPlayer,
    isActionSuccess = pta.execute();
}

else if (currentPlayer.hasCapability(Capabilities.MOVE) || currentPl
    // ALERT: Player does not select a token.
    if (currentPlayer.getHoldingToken() == null){
        isActionSuccess = false;
        alertMessage = Alert.MOVE_NO_TOKEN_CLICK.getAlertMessage();
    }

    // Let player move or fly token.
    else {
        MoveTokenAction mta = new MoveTokenAction(game, currentPlaye
        isActionSuccess = mta.execute();
        alertMessage = Alert.MOVE_NON_ADJ_POS.getAlertMessage();
    }
```



```
// Let player remove opponent's token
else{
    RemoveTokenAction rta = new RemoveTokenAction(game, currentPlayer, posClicked, tokenClicked);
    isActionSuccess = rta.execute();
    if(isActionSuccess){
        game.run();
    } else {
        Board.showPopUp( actionId: "Alert", Alert.REMOVE_MILL_TOKEN_CLICK.getAlertMessage());
    }
}
```

The 2 screenshots above show our implementation of token actions using the same "isActionSuccess" flag for place token, move token and remove token actions.

3. Usability

   We have adopted **usability** as one of the quality attributes in our 9MM game. We aim to **deliver an interface that is both easy to use and user-friendly,** catering to a wide range of users including those with limited knowledge about the game.

   To achieve this, we have implemented several features. **Clear instructions are displayed under the game board during each player's turn.** This ensures that players are guided throughout the game and understand the actions they need to take.

   Besides, we offer **informative feedback to players** to keep them informed and engaged. For example, a green stroke will appear around the token when a player clicks on it. This **visual cue helps players to understand their actions and selections** faster.

Additionally, we have **incorporated a green line around the square** indicating the number of tokens belonging to respective players on the game board to **inform the users about the current player's turn** during each round. This element prevents confusion and ensures that users remain aware of the progression of the game, especially when they are playing as both players.

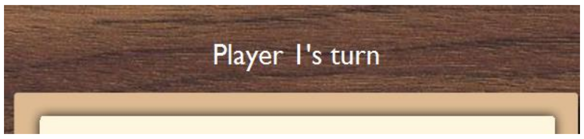
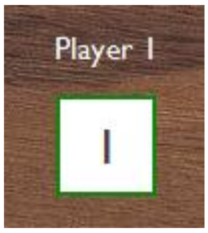
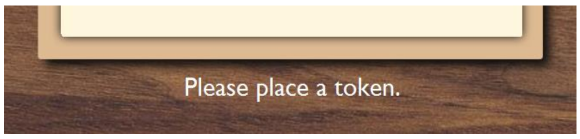
Lastly, **a confirmation window pops up when the player clicks on the quit game button** located at the top right of the game interface. The confirmation popup window provides an additional step to **prevent accidental game exits** and allows the player to confirm his intention to quit the game.

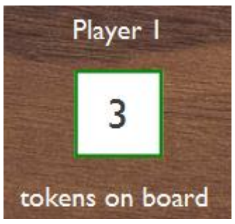
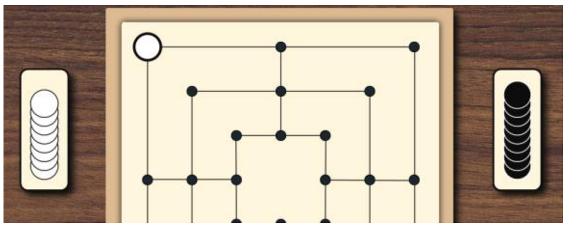
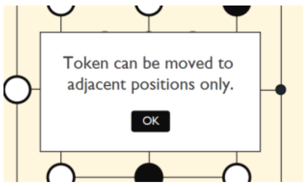
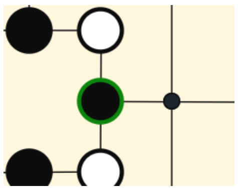
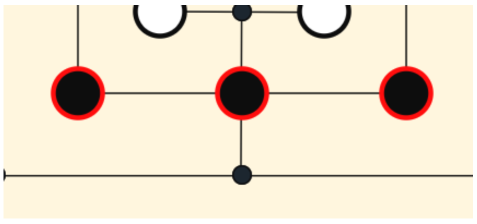
Please refer to the table under the "Human Values" section for screenshots of the examples mentioned above.

Human values:

Based on Schwart's theory, we have demonstrated the **value of benevolence** in our game. Benevolence can be defined as valuing compassion, kindness and helping others. We think that helping the player by **providing useful "indicators" allows the player to have a better user experience** and be able to enjoy the game more. With our user friendly design, all players of varied masteries (even kids) can play and enjoy the game.

We show **kindness** in our game by **including plenty of labels to help the player keep track of things and make suitable moves**:

We have a label to indicate the current player's turn.	 Player I's turn
This label is also highlighted with a green line to indicate the current player's turn.	
We have a label to show the player what action he can make at each turn.	 Please place a token.

We have a label to indicate how many tokens a player currently has on the board.	
We have a side panel to indicate how many tokens a player has placed on the board.	
We display custom error messages for any invalid move that the player makes.	 Token can be moved to adjacent positions only.
We have a green cursor to indicate the token that a player has selected.	
We have a red cursor to indicate the mills on the board.	

Moreover, we also **show compassion** in our game. For example, we understand that humans can make mistakes such as misclicking a token, which is why our design allows players to rechoose the token that they want to move. Our **custom error messages allow the player to make mistakes** and **understand why their move is invalid.**

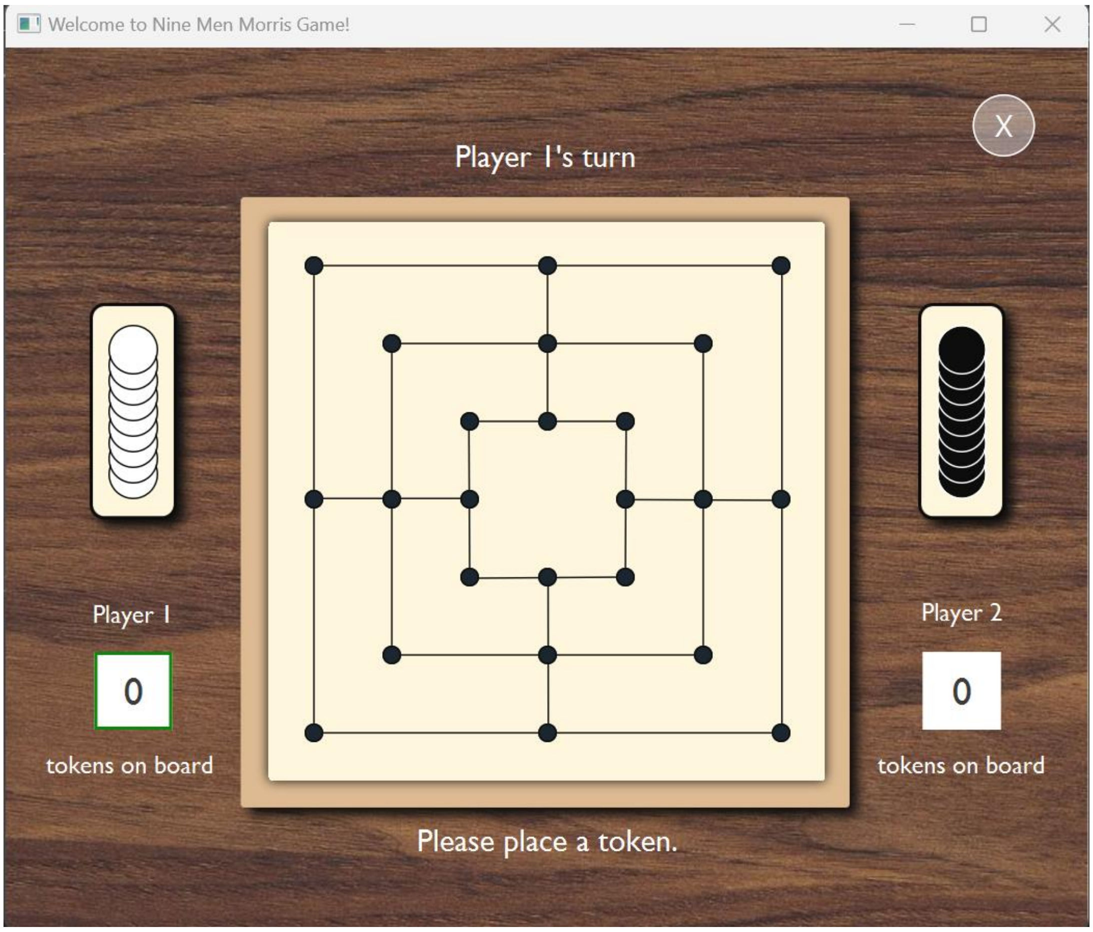
Lastly, we have also demonstrated the **value of Tradition** by **choosing white and black as the colour of our tokens**. We have also **incorporated the traditional look by using a wooden board.**

Demonstration of Game

Link to demonstration video: https://youtu.be/NIkaDyBskcA

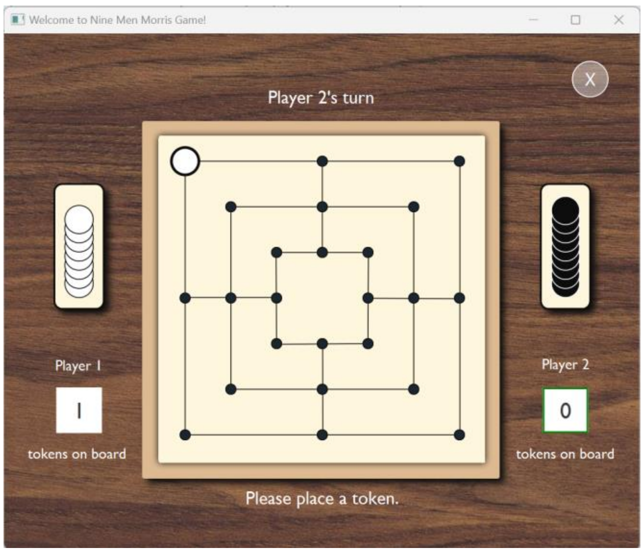
Screenshots of Game Situations

a. An empty board at the beginning of the game



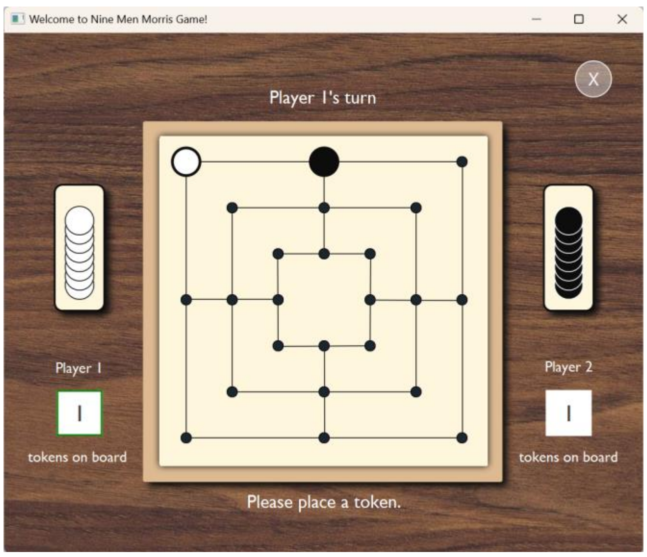
b. Moves of all three phases of the game

i. Place token



Player 2's turn

Please place a token.



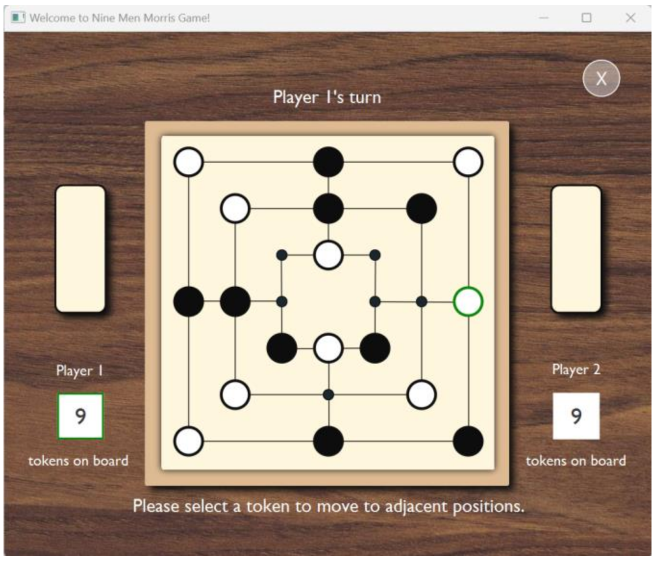
Player 1's turn

Please place a token.

Player 1 (White) placed a token.

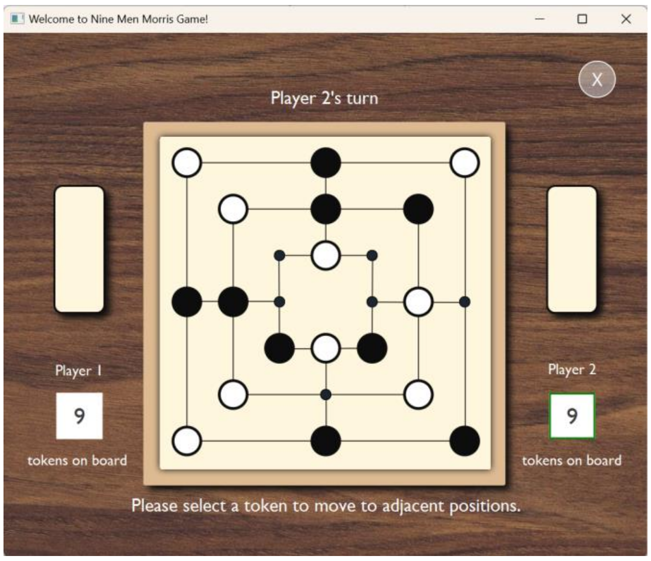
Player 2 (Black) placed a token.

ii. Move token



Player 1's turn

Please select a token to move to adjacent positions.

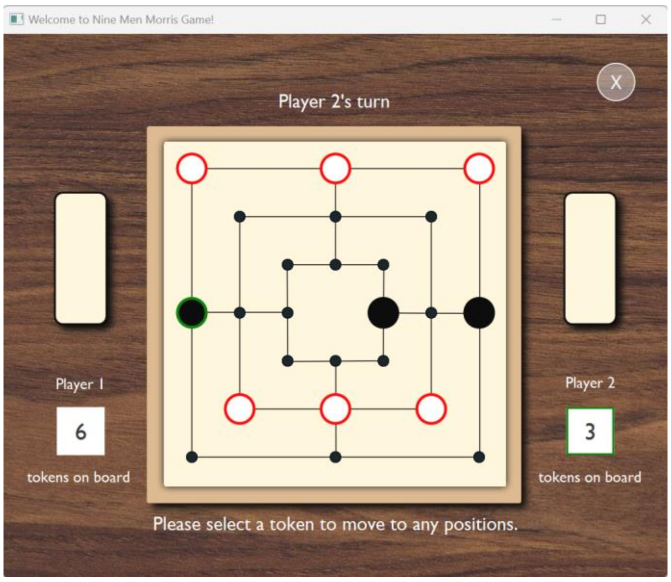


Player 2's turn

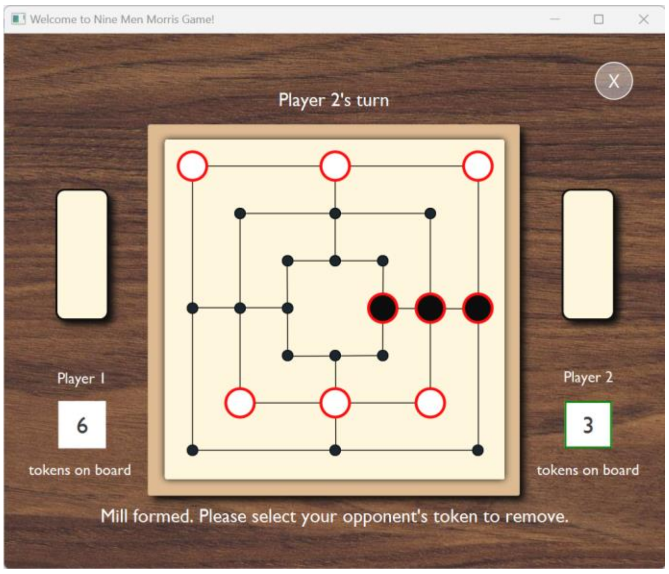
Please select a token to move to adjacent positions.

Player 1 (White) moves its token to an adjacent position, from right to left.

iii. Fly token



Please select a token to move to any positions.

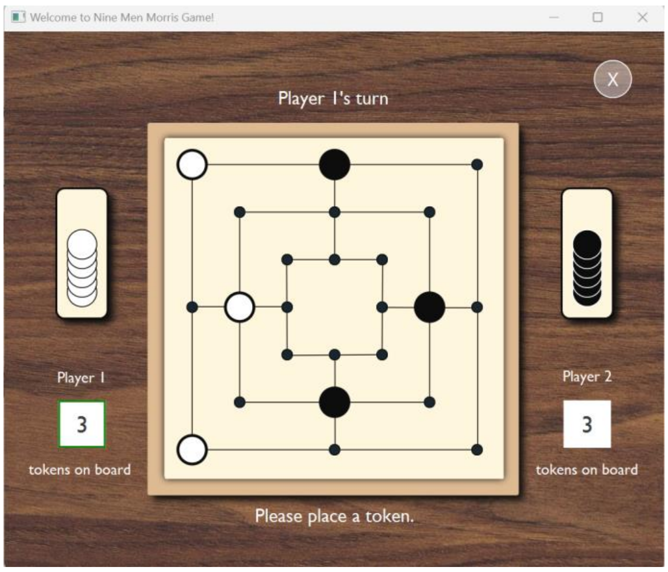


Mill formed. Please select your opponent's token to remove.

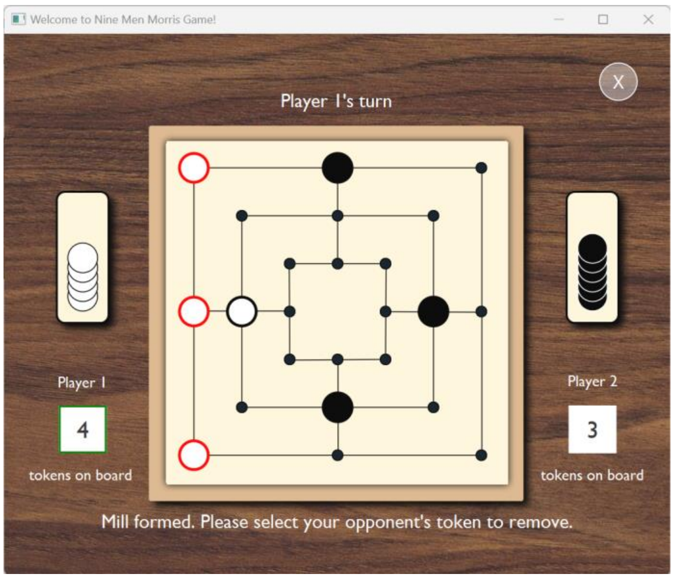
Player 2 (Black) has 3 tokens left on the board, so he can fly his tokens. He moves his token on the left to a non-adjacent position.

c. Detection of new mills

i. Placing phase



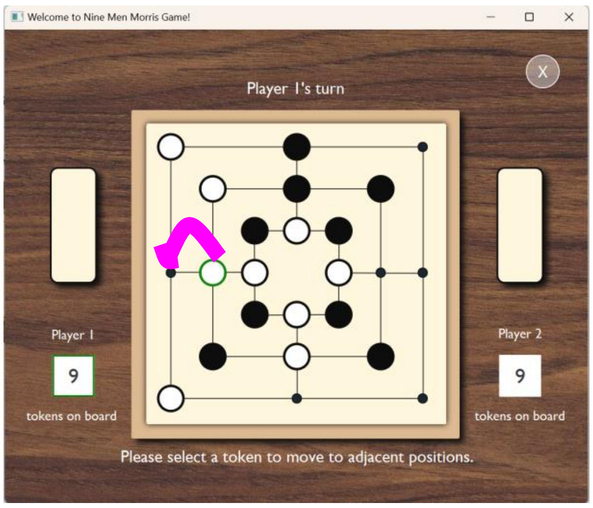
Please place a token.

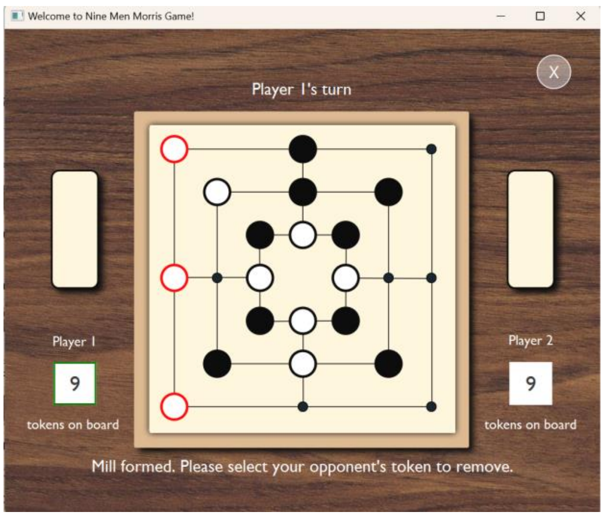


Mill formed. Please select your opponent's token to remove.

In the placing phase, Player 1 (White) places a token to form a mill.

ii. Moving phase

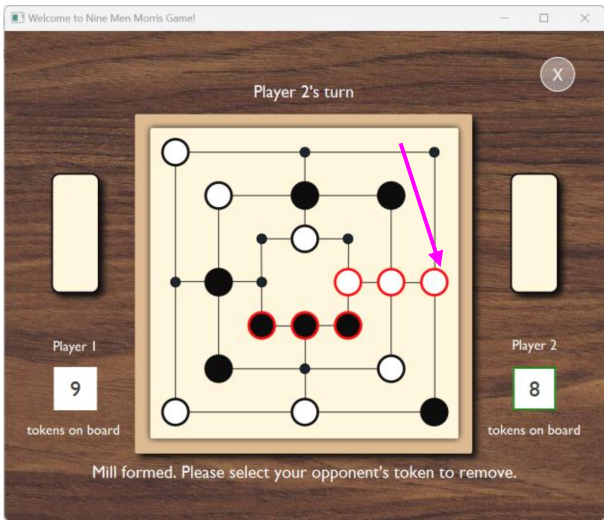

Welcome to Nine Men Morris Game!
X
Player 1's turn
Player 1
9
tokens on board
Player 2
9
tokens on board
Please select a token to move to adjacent positions.

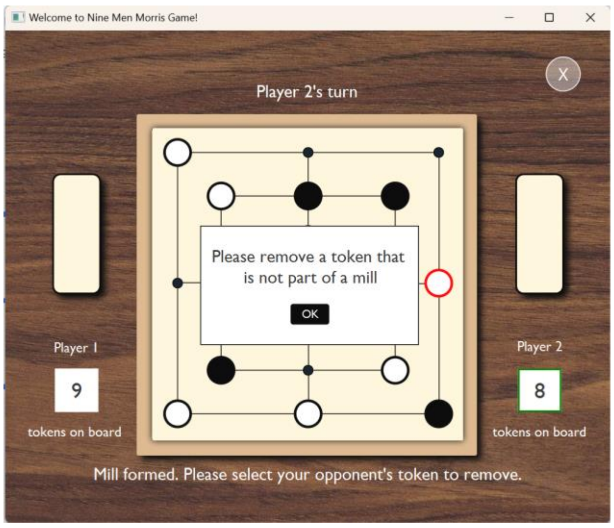

Welcome to Nine Men Morris Game!
X
Player 1's turn
Player 1
9
tokens on board
Player 2
9
tokens on board
Mill formed. Please select your opponent's token to remove.

In the moving phase, Player 1 (White) moves its token from right to left to form a mill.

d. Correct removal of pieces after a new mill creation

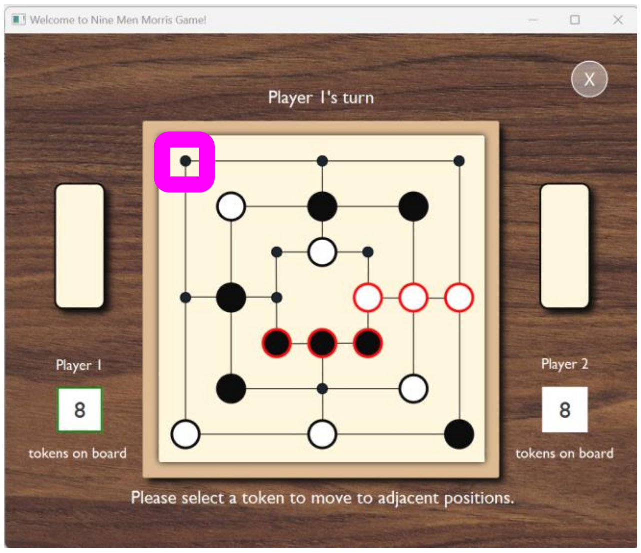
i. Remove a token that is not a mill.


Welcome to Nine Men Morris Game!
X
Player 2's turn
Player 1
9
tokens on board
Player 2
8
tokens on board
Mill formed. Please select your opponent's token to remove.


Welcome to Nine Men Morris Game!
X
Player 2's turn
Player 1
9
tokens on board
Player 2
8
tokens on board
OK
Mill formed. Please select your opponent's token to remove.

Please remove a token that is not part of a mill

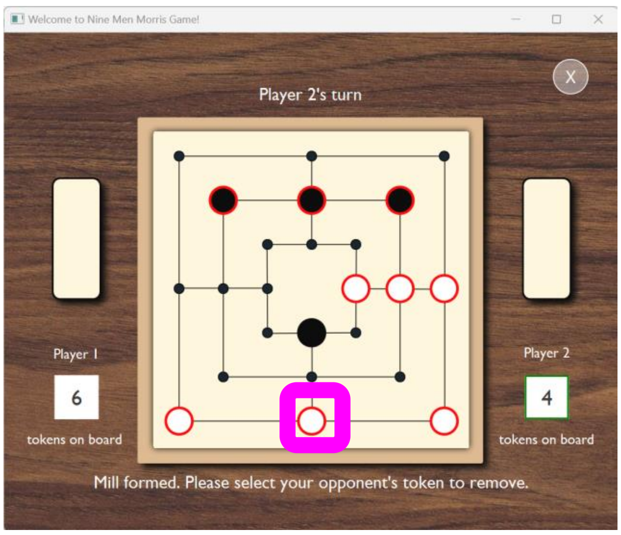
Player 2 (Black) formed a mill, and had to remove the opponent's token. An alert window will pop up if he tries to remove a token that is part of a mill when there are other non-mill tokens that can be removed.



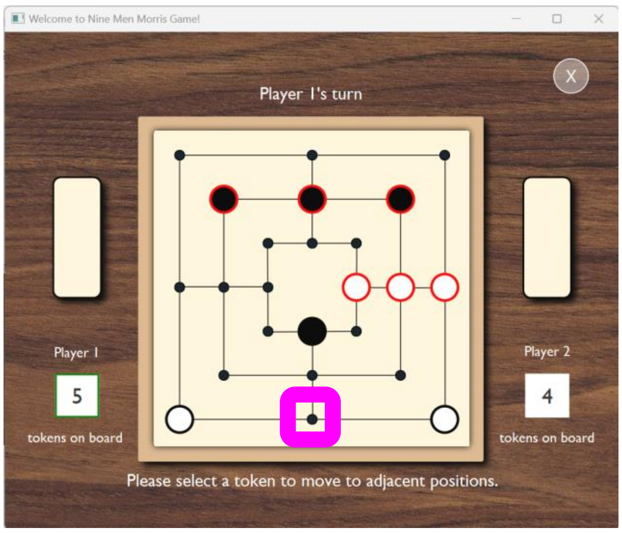
Please select a token to move to adjacent positions.

Player 2 (Black) removed the opponent's token at the top left corner.

ii. Remove a token that is a mill.



Mill formed. Please select your opponent's token to remove.

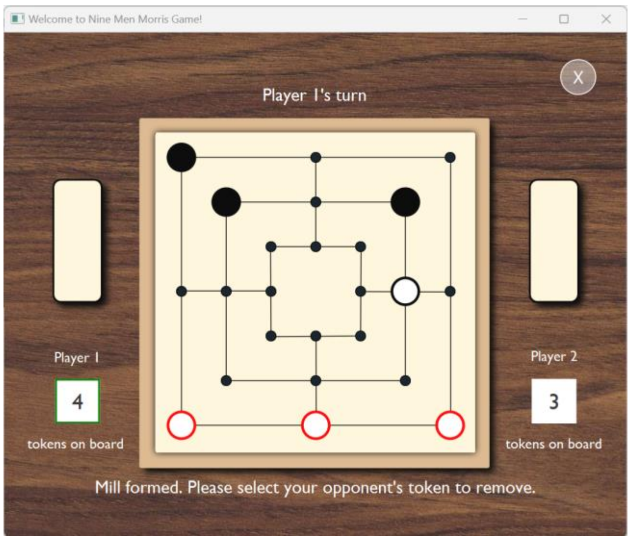


Please select a token to move to adjacent positions.

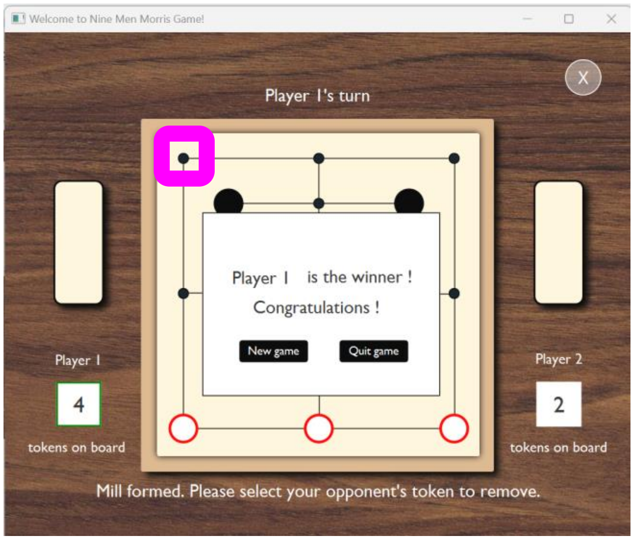
Player 2 (Black) formed a mill, and had to remove the opponent's token. Opponent does not have any non-mill tokens, hence Player 2 can remove tokens that are part of a mill.

e. Detection of the end game

i. Player left with two tokens



Mill formed. Please select your opponent's token to remove.

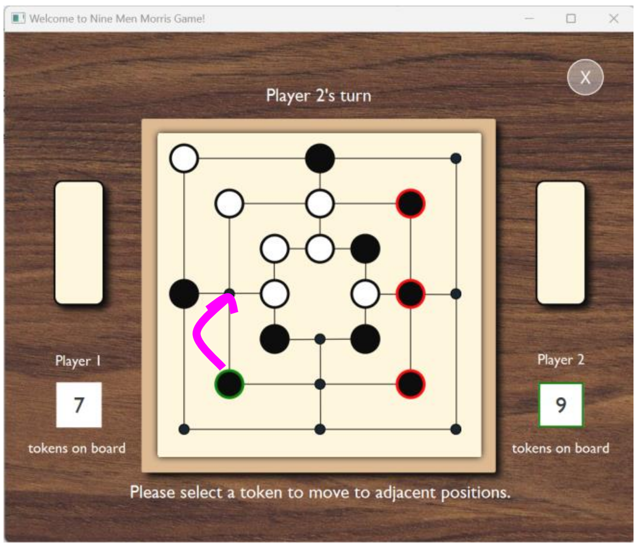


Player 1 is the winner !
Congratulations !

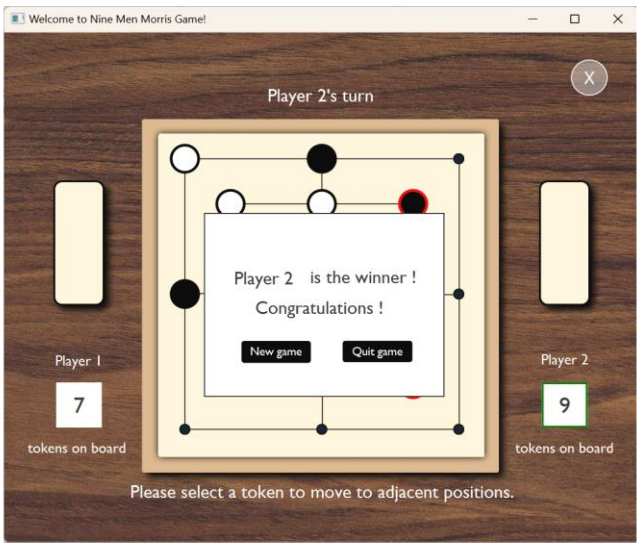
Mill formed. Please select your opponent's token to remove.

Player 1 (White) formed a mill, removing the opponent's token at the top left corner. Opponent is now left with two tokens, hence Player 1 wins the game.

ii. No more possible valid moves



Please select a token to move to adjacent positions.



Player 2 is the winner !
Congratulations !

Please select a token to move to adjacent positions.

Player 2 (Black) moves his token upwards, leaving Player 1 with no more possible valid moves, hence Player 2 wins the game.