Sprint 3 - Specifications

Team Cabbage

Euan Lim, Zoe Tay Shiao Shuen and Lim Zheng Haur

Faculty of Information Technology, Monash University

FIT3077: Software Engineering Architecture & Design
Dr Jean-Guy Schneider

23 May 2023

Table of Contents

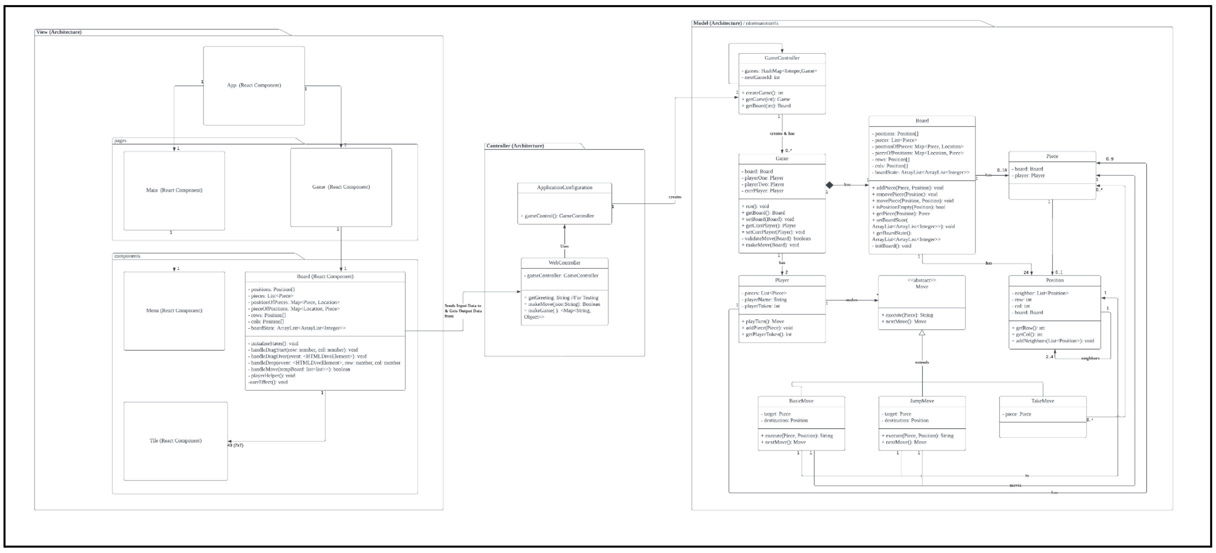
Video Link [Group Presentation]

https://youtu.be/pdNFsw8aE6U

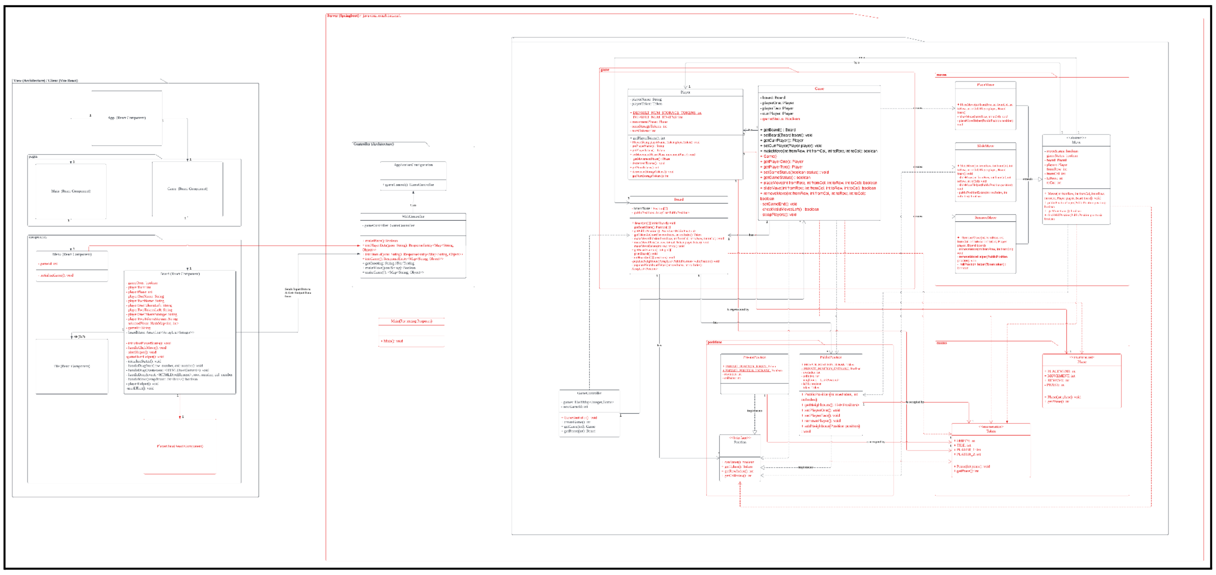
Revised Class Diagrams

Please refer to attached PDF documents **Sprint3_UMLClassDiagram_Previous.pdf and **Sprint3_UMLClassDiagram_Latest.pdf** in Git Repository for better clarity.*

Previous



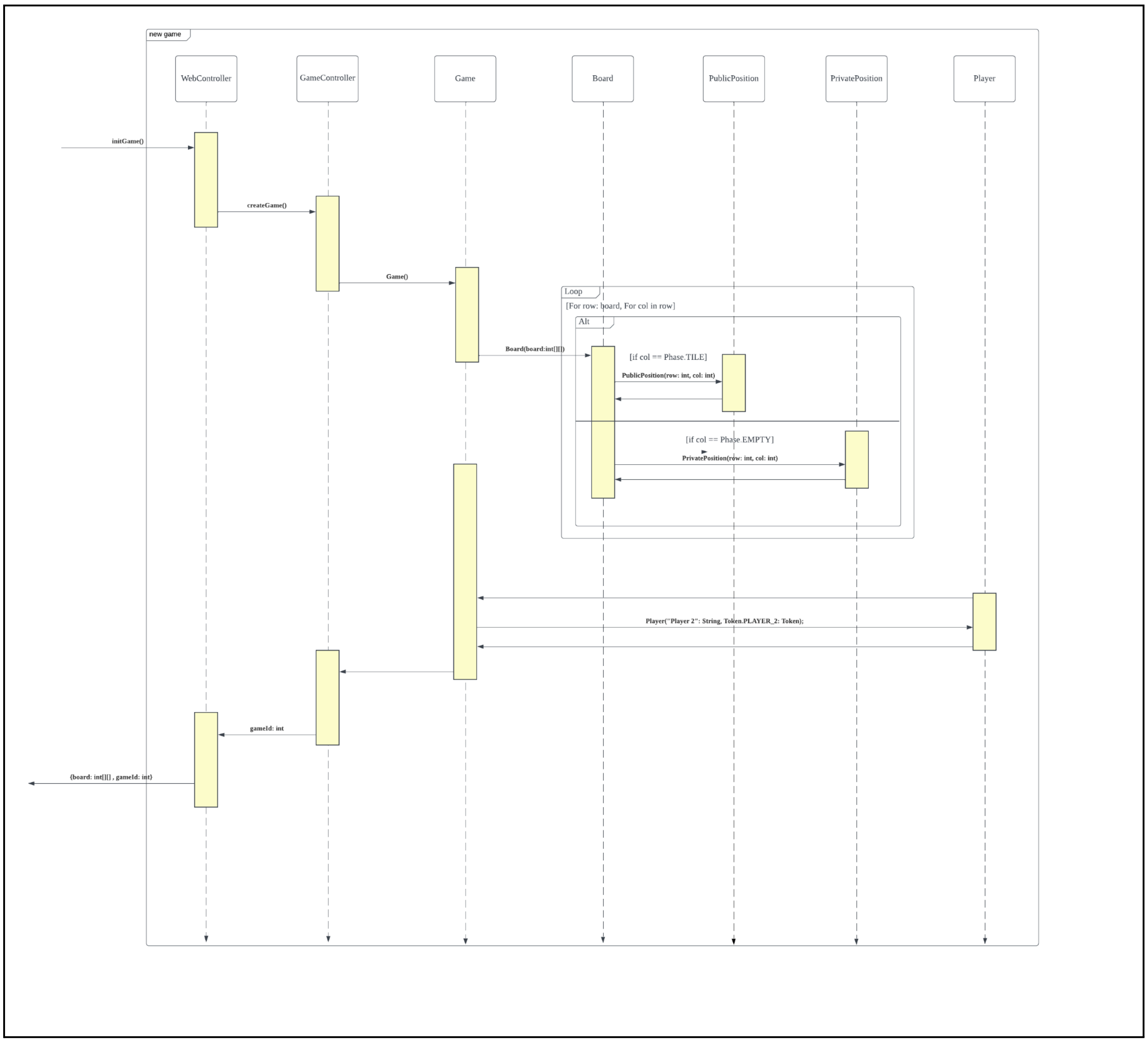
Latest



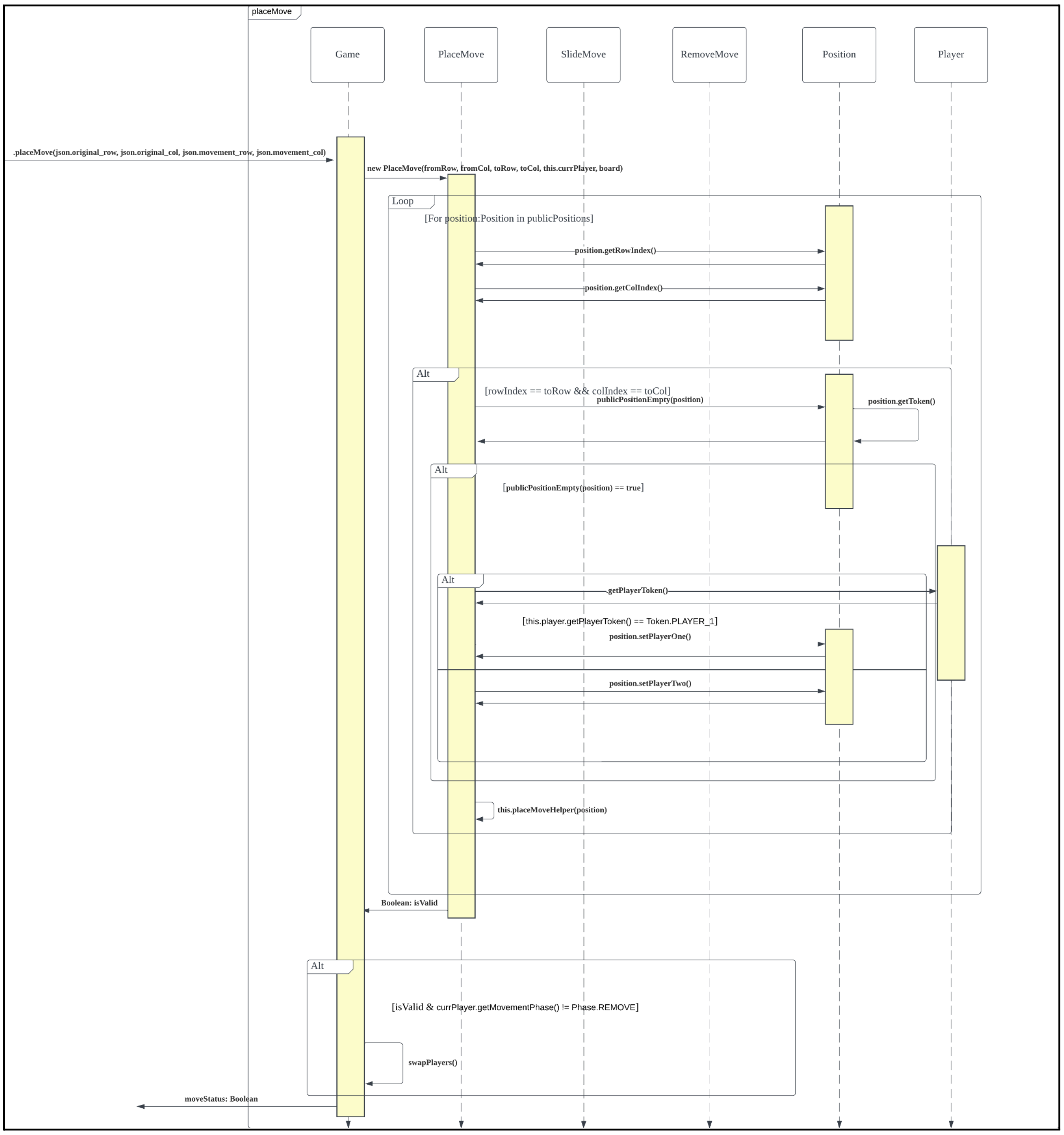
Sequence/Communication Diagrams

Please refer to attached PDF documents **Sprint3_SequenceDiagram.pdf in Git Repository for better clarity.*

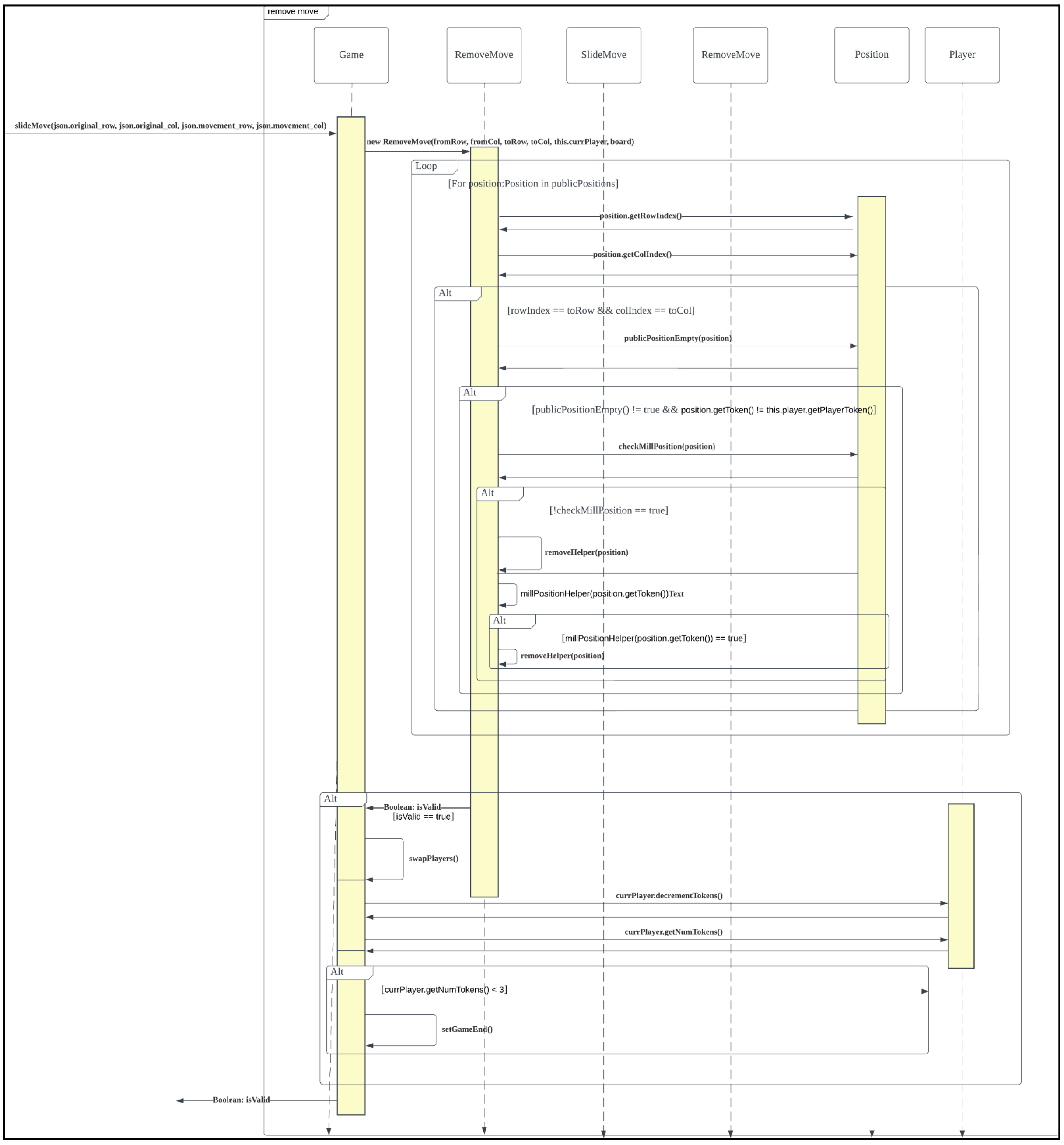
Initialisation Process



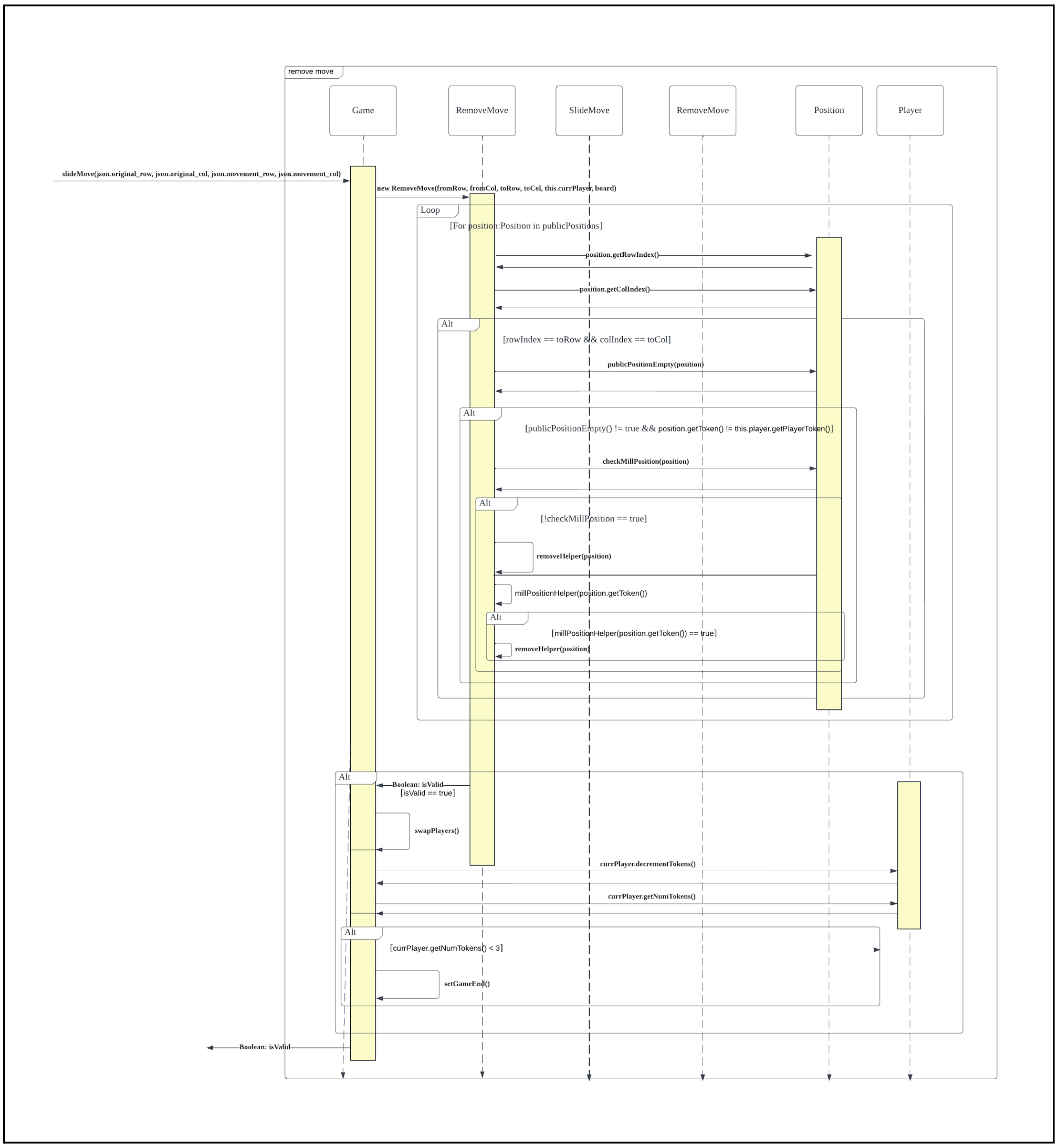
Placing a token


placeMove
Game
PlaceMove
SlideMove
RemoveMove
Position
Player
.placeMove(json.original_row, json.original_col, json.movement_row, json.movement_col)
new PlaceMove(fromRow, fromCol, toRow, toCol, this.currPlayer, board)
Loop
[For position:Position in publicPositions]
position.getRowIndex()
position.getColIndex()
Alt
[rowIndex == toRow && colIndex == toCol]
publicPositionEmpty(position)
position.getToken()
Alt
[publicPositionEmpty(position) == true]
Alt
.getPlayerToken()
[this.player.getPlayerToken() == Token.PLAYER_1]
position.setPlayerOne()
position.setPlayerTwo()
this.placeMoveHelper(position)
Boolean: isValid
Alt
[isValid & currPlayer.getMovementPhase() != Phase.REMOVE]
swapPlayers()
moveStatus: Boolean

Removing token

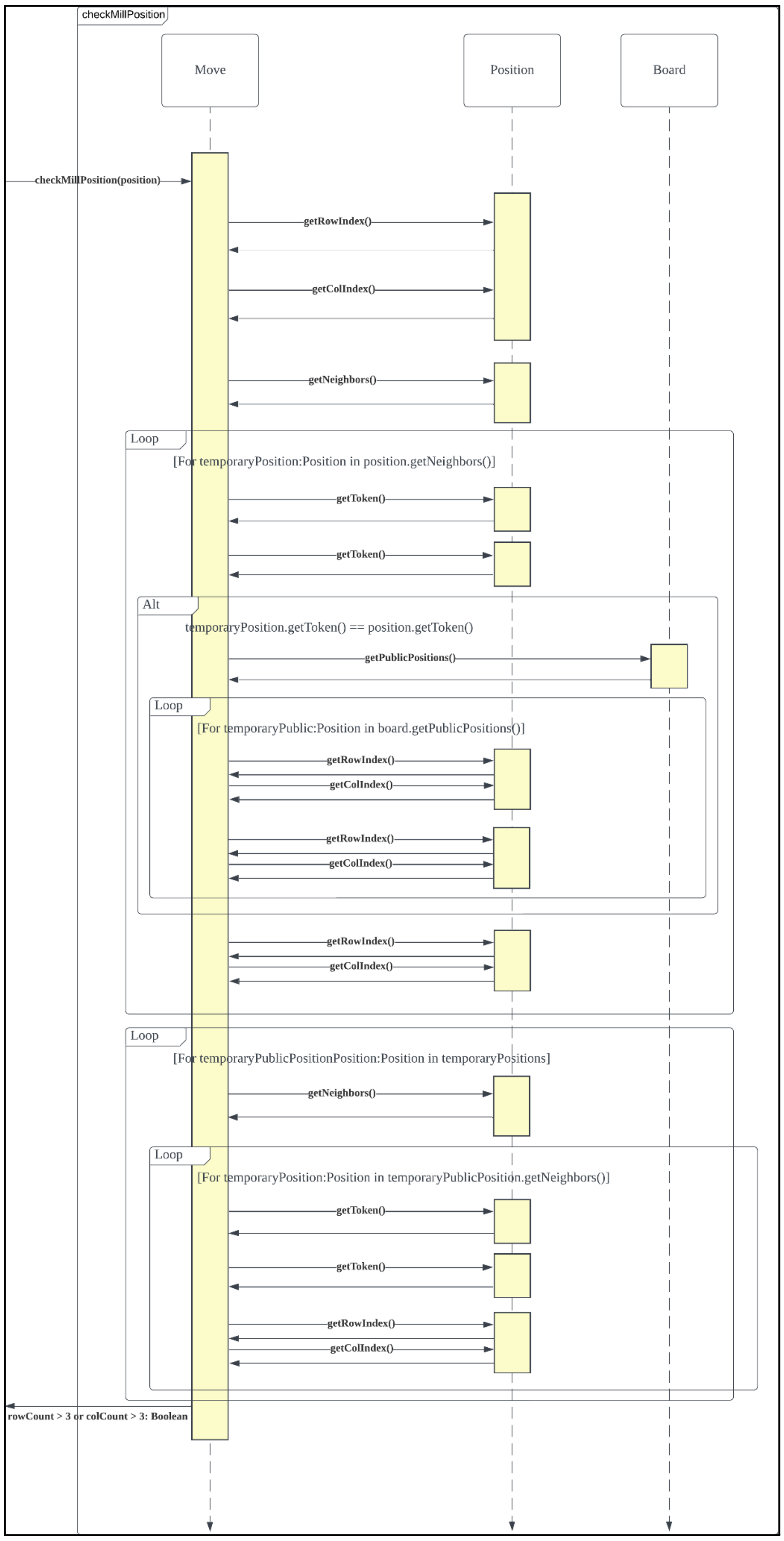

remove move
Game
RemoveMove
SlideMove
RemoveMove
Position
Player
slideMove(json.original_row, json.original_col, json.movement_row, json.movement_col)
new RemoveMove(fromRow, fromCol, toRow, toCol, this.currPlayer, board)
Loop
[For position:Position in publicPositions]
position.getRowIndex()
position.getColIndex()
Alt
[rowIndex == toRow && colIndex == toCol]
publicPositionEmpty(position)
Alt
[publicPositionEmpty() != true && position.getToken() != this.player.getPlayerToken()]
checkMillPosition(position)
Alt
[!checkMillPosition == true]
removeHelper(position)
millPositionHelper(position.getToken())Text
Alt
[millPositionHelper(position.getToken()) == true]
removeHelper(position)
Alt
Boolean: isValid
[isValid == true]
swapPlayers()
currPlayer.decrementTokens()
currPlayer.getNumTokens()
Alt
[currPlayer.getNumTokens() < 3]
setGameEnd()
Boolean: isValid

Move Token



remove move
Game
RemoveMove
SlideMove
RemoveMove
Position
Player
slideMove(json.original_row, json.original_col, json.movement_row, json.movement_col)
new RemoveMove(fromRow, fromCol, toRow, toCol, this.currPlayer, board)
Loop
[For position:Position in publicPositions]
position.getRowIndex()
position.getColIndex()
Alt
[rowIndex == toRow && colIndex == toCol]
publicPositionEmpty(position)
Alt
[publicPositionEmpty() != true && position.getToken() != this.player.getPlayerToken()]
checkMillPosition(position)
Alt
[!checkMillPosition == true]
removeHelper(position)
millPositionHelper(position.getToken())
Alt
[millPositionHelper(position.getToken()) == true]
removeHelper(position)
Alt
Boolean: isValid
[isValid == true]
swapPlayers()
currPlayer.decrementTokens()
currPlayer.getNumTokens()
Alt
[currPlayer.getNumTokens() < 3]
setGameEnd()
Boolean: isValid

Check Mill


checkMillPosition
Move
Position
Board
checkMillPosition(position)
getRowIndex()
getColIndex()
getNeighbors()
Loop
[For temporaryPosition:Position in position.getNeighbors()]
getToken()
getToken()
Alt
temporaryPosition.getToken() == position.getToken()
getPublicPositions()
Loop
[For temporaryPublic:Position in board.getPublicPositions()]
getRowIndex()
getColIndex()
getRowIndex()
getColIndex()
getRowIndex()
getColIndex()
Loop
[For temporaryPublicPositionPosition:Position in temporaryPositions]
getNeighbors()
Loop
[For temporaryPosition:Position in temporaryPublicPosition.getNeighbors()]
getToken()
getToken()
getRowIndex()
getColIndex()
rowCount > 3 or colCount > 3: Boolean

Revised Design Architecture

Pipeline Design

In this sprint, our team has decided to revise our architecture to incorporate the pipeline design pattern into our existing model view controller design architecture. The pipeline design pattern is a pattern where a process is broken down into a series of stages, where each stage is responsible for a specific purpose. Our team has incorporated this design pattern into the model of our game using different methods passing on the data from 1 method to another similar to a pipeline.

For example, we have heavily used this design pattern during the validation of a move by a player found in Game classes makeMove method. The data of the move is passed from 1 method to another respective placeMove, slideMove, removeMove; while each method is responsible for a specific type of validation such as validation of if the opponent player has valid moves after this move, validation of a present mill to follow the move with a take move etc. Similarly, for the creation or initialization of the board, we have used a pipeline design approach. We first instantiated the board, starting in the Board class's constructor and then we moved to another process of populating the board, and finally another function to populate each position on the board with their respective neighbours.

This was very beneficial as it allowed for a single source of input, and a single source for output. This was against the option of creating a super method that would perform a similar function which would be difficult to maintain and extend. Or simply abstracting it away to multiple different classes in which case would also be more difficult to maintain and extend; creating essentially one-off classes.

Creation of Enums (Strategy Design Pattern)

As our previous iteration of the board required essentially understanding magic numbers for values at the position due to the board being solely represented by an adjacency matrix of integers, this left the implementation difficult to both understand, maintain with the constant checks of int magic numbers, we decided to implement the strategy design pattern in the form of a new Token enum system for each position on the board. This Token Enum is used to identify whether the position is a public position (a position where pieces could move into), a private position (a position where pieces could not move into), a player 1 piece, or a player 2 piece. This token is used throughout our implementation to process the logic and the layout of the board for each state of the board. We discovered that using this system is a very efficient way for us to track the state of the board and to process the logic behind our game such as the validation of moves and the win conditions. Hence we implemented two new Position types; Public and Private, whereby the default token for each respectively was utilising the Token Enum. Changing the adjacency matrix of int's, into an adjacency matrix of Position instances.

Furthermore, our team has also implemented another enum system for the strategy design pattern known as the Phase Enum. These enum values are used as attributes or instance variables of the Player instances. This allows us to identify the state of the player phase such as if the player is in the phase of putting down pieces, then onto the phase of moving pieces and lastly on the phase where they can jump their pieces around the board. We believe that this is an elegant solution to track the different types of moves that a player is allowed to do in the game rather than a messy solution by computing the phase of the player based on the state of the board at every iteration.

General Refactors

Implemented a gameController that contains a dictionary of all games. Accessing via a game id for the key of the values in the dictionary. This is accessed through the url params in the browser for the user so they can revisit past games, and not lose progress. To increase organisation, we further grouped related classes into packages; game package, positions package and enums package.

Position as an interface

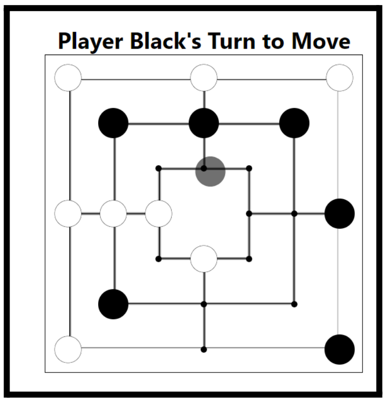
Our team has converted the position class into an interface and this interface is then implemented by the Public Position class, representing positions that are valid positions on the board where pieces can move onto, and a Private Position class, representing positions on the board where pieces could not enter. We have decided to use this approach as we require two different types of position instances to represent valid and invalid positions on the board.

This is over a parent class, as the similarities between the two classes (Public and Private) are only in the sharing of a few methods; as Private has very minimal. Hence the added complexity of utilising an inheritance relation is not required.

Quality Attributes

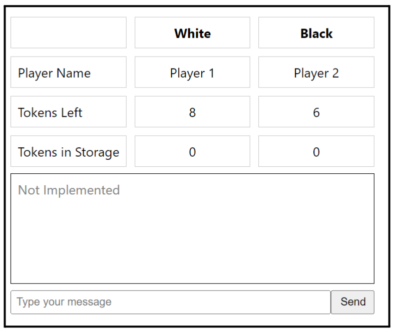
In the design of our 9 men's morris game, we have considered several quality attributes, predominantly centred around usability and flexibility.

1) In regards to Usability, we believe we have created an intuitive user interface, via the control decisions on how players make inputs. Instead of clicking buttons and selecting move types or typing in moves to the console/ prompt we opted for a more hands on approach. This is in the sense that for players to both remove and place pieces, they simply need to click on the positions of the board with their mouse etc. And for sliding or moving pieces to fly across the board, we implemented this in the form of a drag and drop whereby players drag pieces to where they want it to be. For example the dragged transparent black piece as shown below.



Hence, players are able to effortlessly drag a desired piece to the desired position,whilst visually representing the move. An intuitive mode of interacting with the game is incredibly important, as it reduces the learning curve and dramatically shortens the time it would take to type in positions for movement, or even to specifically click a button to allow for a specific move. This overall leads to a far more satisfied consumer / user of the game, and as mentioned makes the game very easy to get into.

2) Another example of usability being in regards to the clear display of the current state of the game, from the number of pieces both players currently have in and out of storage;



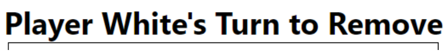
	White	Black
Player Name	Player 1	Player 2
Tokens Left	8	6
Tokens in Storage	0	0

Not Implemented

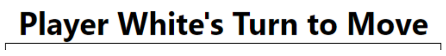
Type your message

Send

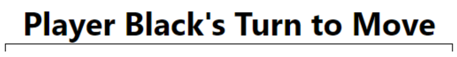
To the header prompts displaying and alerting to the player what the current move required is from either party;



Player White's Turn to Remove

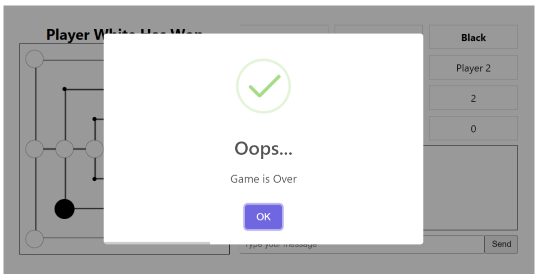


Player White's Turn to Move



Player Black's Turn to Move

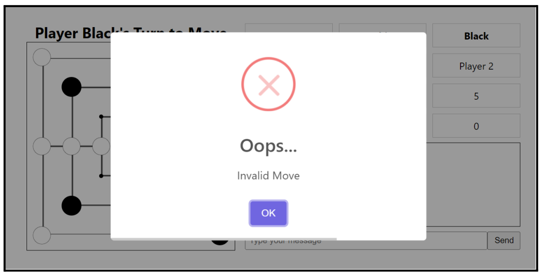
To even prompts indicating movement failures;



Oops...

Game is Over

OK



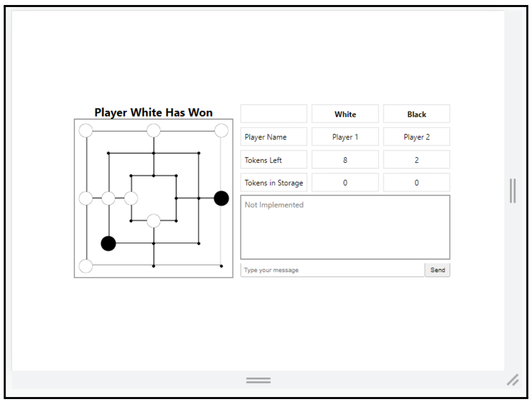
Oops...

Invalid Move

OK

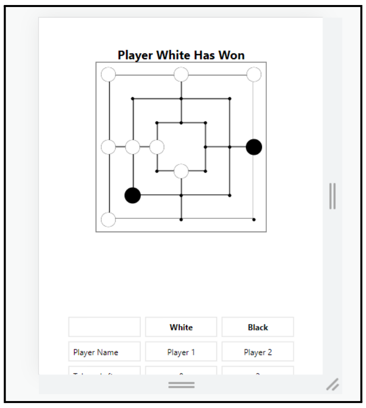
These alterations and prompts are important for helping users better understand the current state of the game, and to have visual cues of their actions displayed. Improving the overall user experience.

3) Lastly, the design in regards to the interface is also responsive through the resizing fitting for both mobile and desktop screens, whereby on a wider screen the data and the board are in line, whereas when the screen size width reduces, the data and board instead stack.



Player White Has Won

	White	Black
Player Name	Player 1	Player 2
Tokens Left	8	2
Tokens in Storage	0	0



Player White Has Won

	White	Black
Player Name	Player 1	Player 2

Schwartz’s theory of human values

We have manifested human values from Schwartz's theory in our implementation of the game 9 men morris. Among them are the value of **Universalism** where it emphasises on equality and justice. In our game design, we have implemented it in a way where the game is not biassed to one player or another and both players have equal chance of winning. In fact, according to Gasser (1996), 9 men morris is a solved game where if both players played perfectly, the game will end in a draw.

Furthermore, we have informed the design of our game to bear the value of **Self-Direction**. Self-Direction mainly regards freedom and independence as priority and through the design of our game, we have ensured that players have the freedom and independence to play the game. From the first phase of the game where players put down their tokens to the next phase of moving them along the lines of the board and the final phase where players could jump across the board to any vacant positions, we have allowed the player to have the freedom and independence to do any moves as long as they are valid in accordance to the game rules.

Finally, we have embraced the value of **Stimulation** and achievement to the players playing the game of our design. We designed the game with a vision that players could use it as a platform to battle their wits in strategic thinking, forethinking and problem solving among other players through a game of 9 men morris. Through these, players are able to feel the excitement, novelty and variation that our game could provide, manifesting the value of stimulation. Through when players make capture moves they are able to see that their open loses pieces, and also when winning, see that the game tells them they won, thus a sense of achievement.

References

Gasser, R. (1996). Solving Nine Men's Morris. *Games of No Chance, 29*. MSRI Publications. Retrieved from http://library.msri.org/books/Book29/files/gasser.pdf