Sprint 3 Deliverable 2

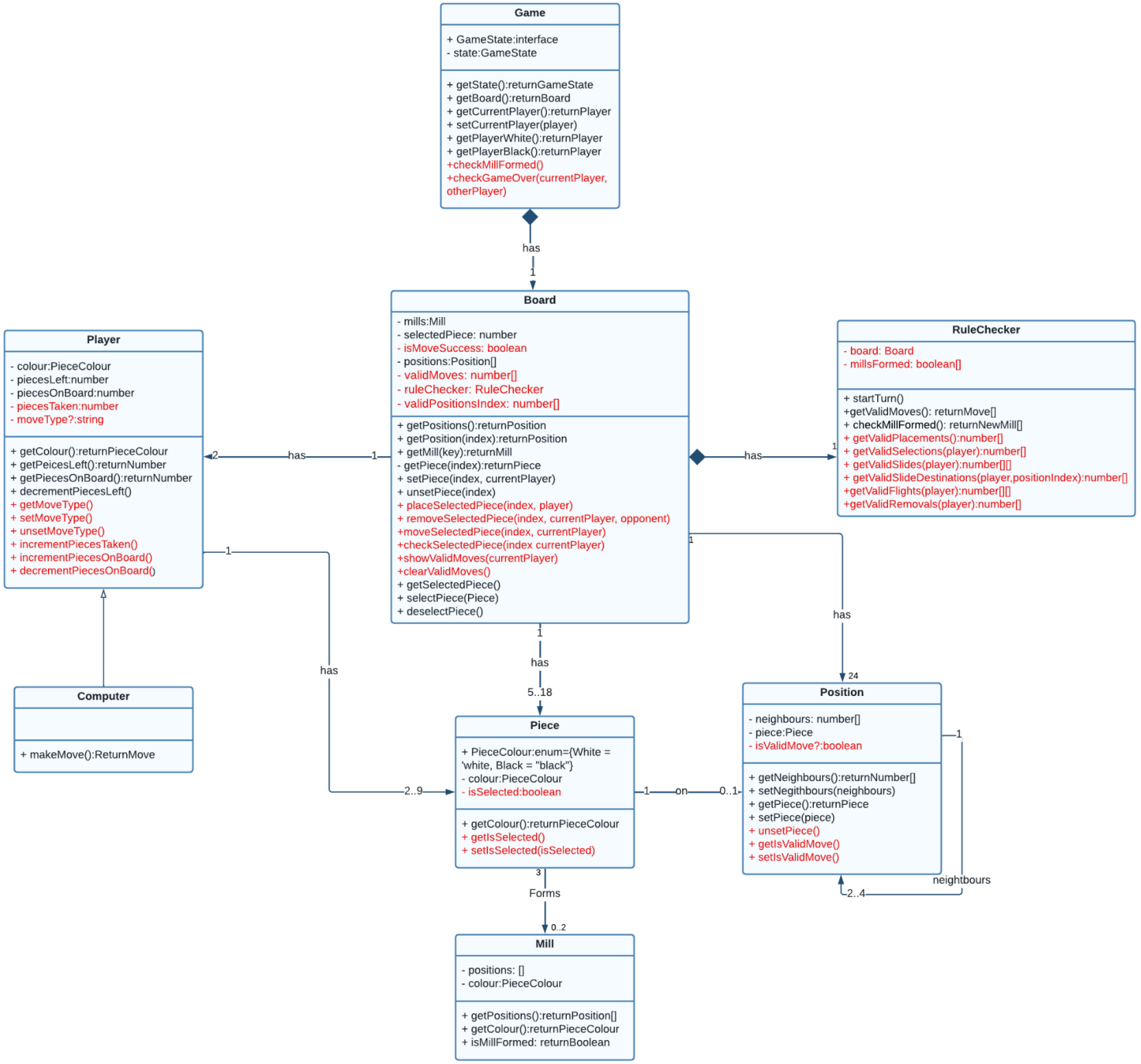
Architecture and Design Rationales

Prepared by 3 Men's Morris
Jer (Arthur) Lin, Quoc (Harry) Han and Liangdi Wang

Contents

1. Revised Class Diagram

Revised class diagram image file and all sequence diagram image files are in "/project/docs/" folder of fit3077-s1-2023/CL_Thursday4pm_Team4 GitLab repository.



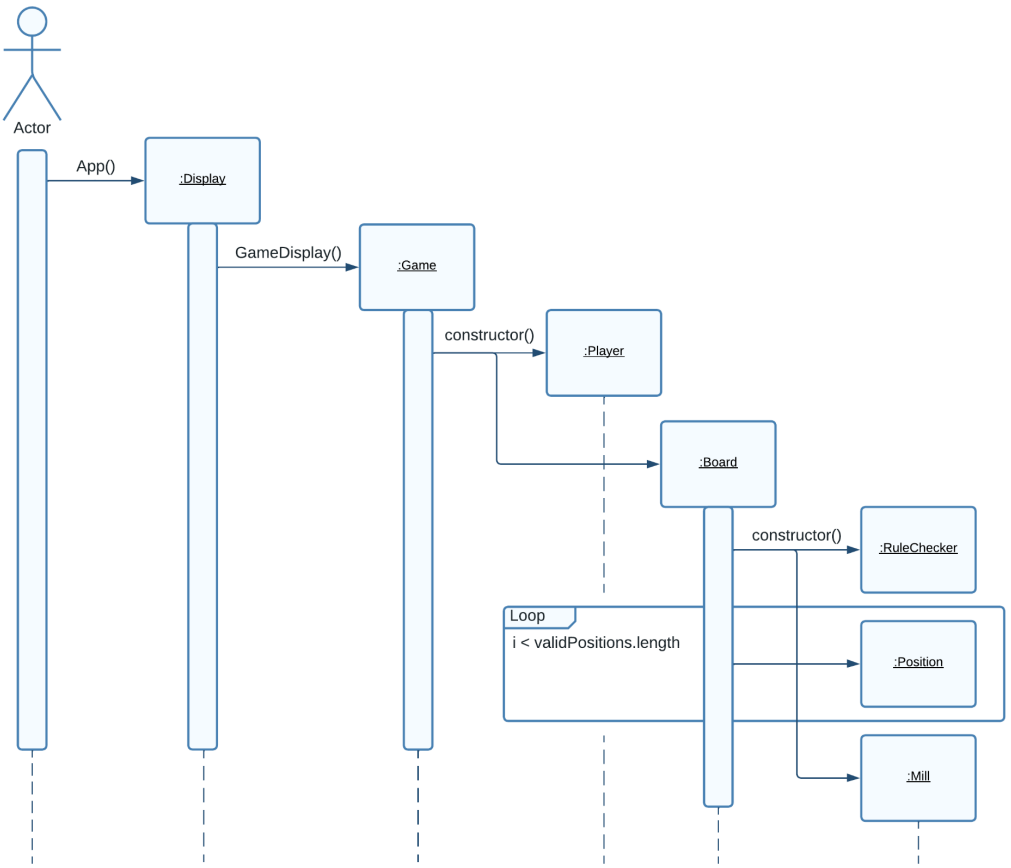
2. Sequence Diagrams

2.1. Game initialization diagram

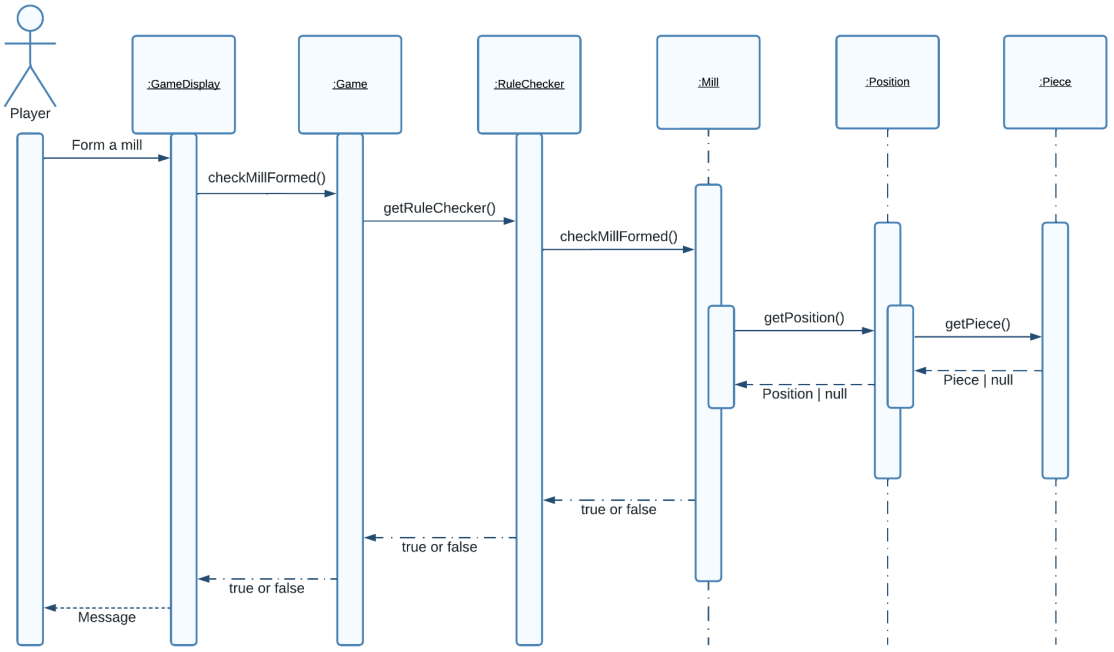
Whenever the player starts the game or clicks on the reset button, the instance of the Game class is generated and followed by the creation of a Board object and two Player objects. The initialisation of the Board object will lead to the creation of the following instances:

- 24 Position objects representing the 24 positions the players can place their pieces on the board.
- A Rule Checker object is responsible for producing a list of positions in which the player can place their piece on the board based on the player's move (place, fly and slide) and a piece which they decide to move.
- 16 Mill objects representing the 16 possible mills the player can form in the game.
    - Each mill knows the 3 Position objects that belong to that mill.

The initialisation of the Player object keeps track of a "piecesLeft" property representing the number of pieces each player can place during the game's life cycle.
By clicking on display, Piece is created on a position in the board based on the current player (current player is a property in the game object that keeps track of whose turn it is)

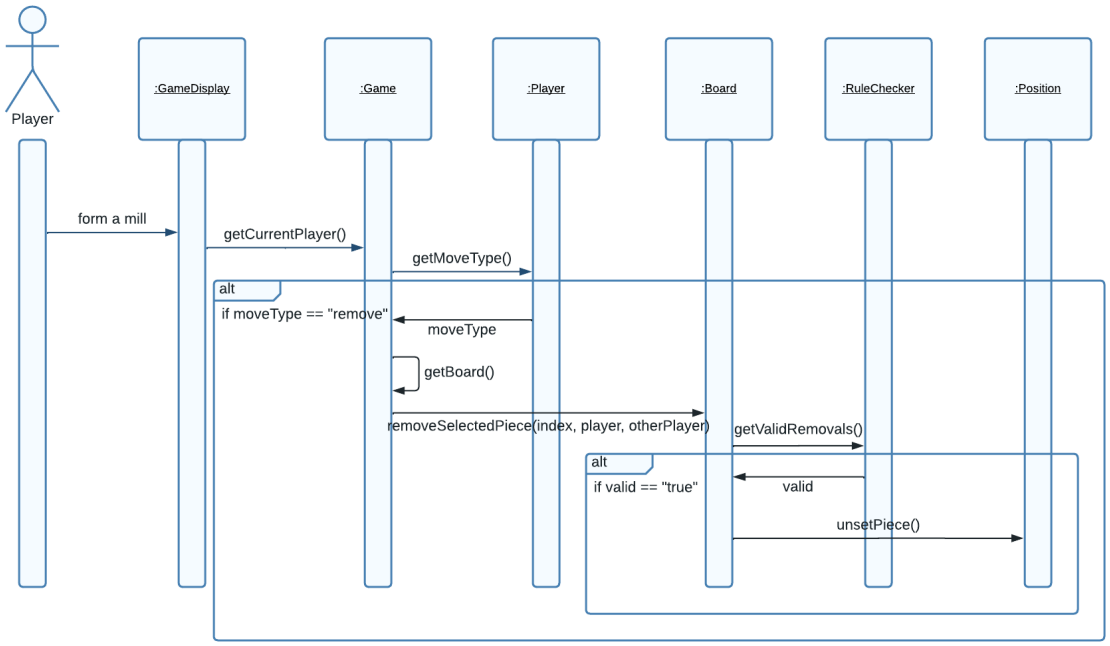


2.2. Non-trivial interaction 1 - Form a mill



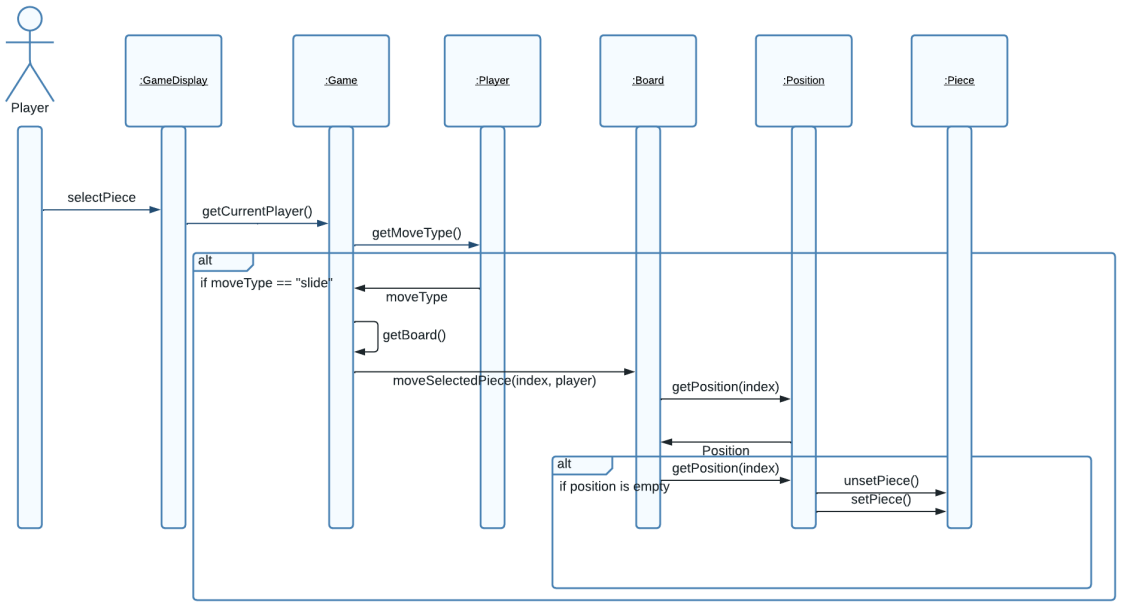
When the player forms a mill, the Game Display object will call the checkMillFormed() method from the Game class. The checkMillFormed() method is being used to tell the RuleChecker object to start the checking for the mill formed. The RuleChecker object then starts the checking by calling its method checkMillFormed() to get the reference of the mill which is considered as “be formed” by the player. Since the RuleChecker object has the reference of the Board object of the game, they can retrieve the 3 pieces and their positions in this “be formed” mill and then compare these pieces’ colour with the colour of the player. If the colour of those 3 pieces matches with the colour of the player, RuleChecker will return true otherwise return false to the Game Display object. Finally, the Game Display object will display the message to the player’s screen corresponding to the boolean data received from the RuleChecker object.

2.3. Non-trivial interaction 2 - Remove a piece



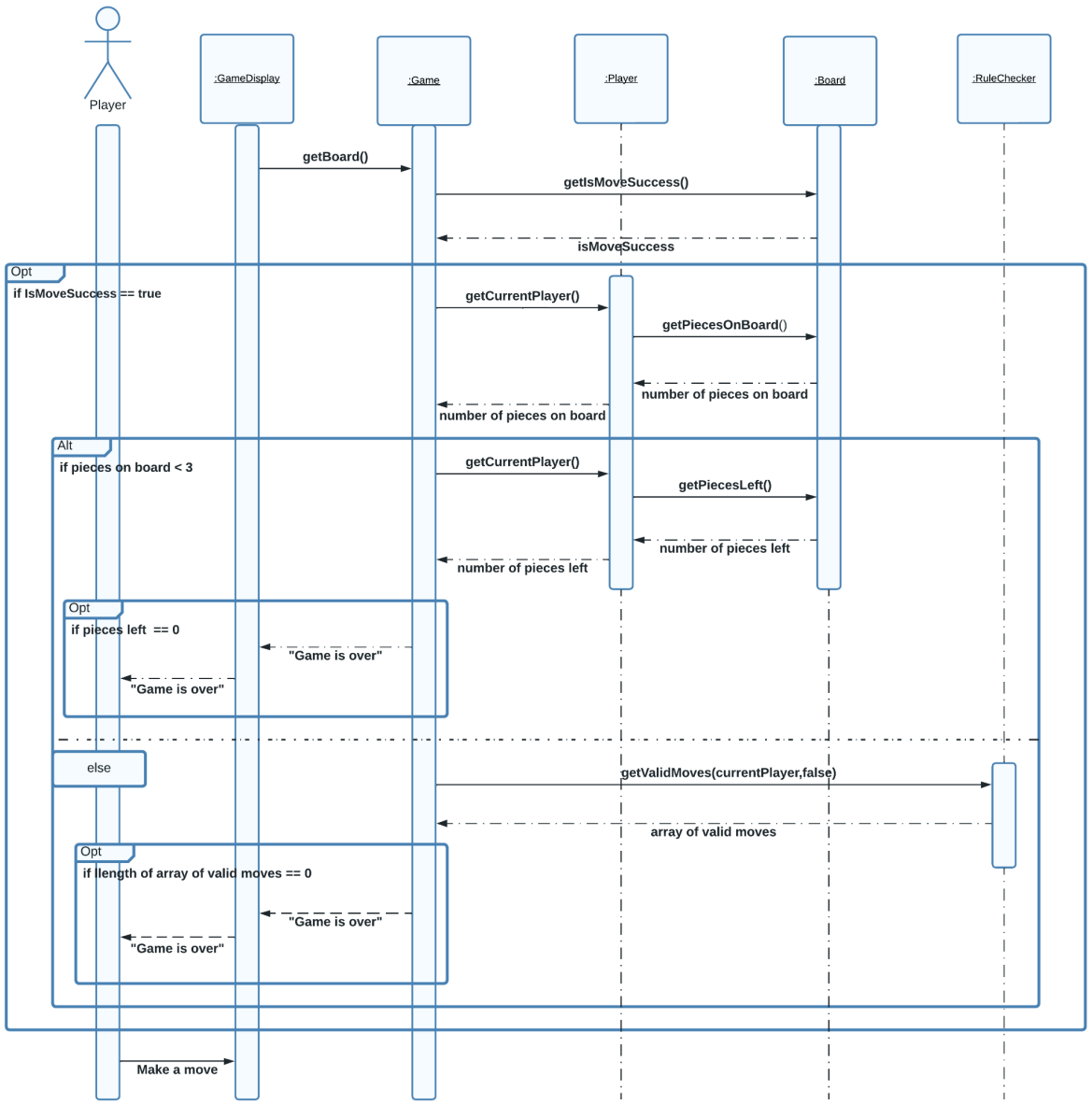
When the player forms a mill, the player will be prompted to remove a piece. The player selects the piece to remove by interacting with the GameDisplay, the GameDisplay object will then check with the Game class to check if the current player is intending to remove a piece selected. Once the confirmation is done, the board object will then proceed to call itself and remove the selected piece object from the positions. However, before the removal takes place, the board will check with the RuleChecker to see if the selected piece to remove is a legal piece to be removed. Once the check passes, the piece gets removed.

2.4. Non-trivial interaction 3 - Slide a piece



When sliding a piece, the player would interact with the GameDisplay to select the piece they wish to move. Then the second selection would be the destination. The Game would be able to know if the move type is slide by checking with the Player object to see if the intended move is slide. If so, the game will then reference the board and instruct it to move the selected piece to its destination. The board would then call the destination position to see if the position is empty (along with other checks such as if the move is legal etc). Once all the checks have been passed, the source position will remove the piece from itself and the destination position will place the piece into itself.

2.5. Non-trivial interaction 4 - Win condition



Before allowing the player to make a move, there will be a process of checking for the winner of the game. The Game will firstly tell the Board to check whether the previous player has made a valid move. If yes, the Game will then get the state of the current player and check for two things: the number of pieces they have on the board and the number of pieces they have not placed on the board. If the number of pieces the current player has on the board is less than 3 and the number of pieces they haven't placed on the board is equal to 0, the game is over otherwise the current player is allowed to make a move. However, if the number of pieces the current player has on the board is greater than 3, the Game will then ask the RuleChecker to give them an array of validMoves the current player can do. If the length of the array of validMoves is equal to 0, the game is over otherwise the current player is allowed to make a move.

3. Design Rationales

3.1. Architecture revision rationale

The fundamental architecture design in terms of tech stack for this application has not changed. This is due to the fact that tech stack change at this stage of the development process is not advisable as it would require a high amount of effort in re-factoring work. Additionally, the chosen tech stack continues to be the best solution, this is because the TypeScript models continue to allow the game object such as Board ,Pieces and Position to exist as individual objects which encourages the separation of concern when it comes to executing game logic. However, there are some design changes, listed by class:

Board

1. isPiecedMoved has been replaced by isMoveSuccess. Rather than moving the piece as the player instructed, the isMoveSuccess is now required to determine player's move is valid before performing the move
2. More responsibilities from the board are delegated to the RuleChecker class now, to prevent the board becoming a god class. Hence the new addition of a RuleChecker in and an array of ValidMoves to store all the available moves once RuleChecker computes all the possible moves for the player in the current turn. Additionally, object specific moves are also computed by other classes. For example, player objects should compute whether to slide or fly base on the pieces left
3. Without needing to handle game logic, the board is now assigned responsibilities that are more appropriate for the board game. Most of these methods involve changing the board state, some of these include placing/removing the selected piece and getting valid moves by calling the RuleChecker class and executing the move to reflect the board's state.

Game

1. To further reduce the Board class's responsibility further, the team has decided it is most appropriate the Game checks for whether a mill is formed
2. Since the game class initiates the game, it makes sense for the class to check whether the game is over and end itself

Player

1. To make the user interface more friendly, the piecesTaken for each player is now recorded. Respective methods such as incrementing and decrementing that value is also created
2. Moves available to the player (slide and fly) is also depending on the number of piece the player has left, therefore the determination of moves is assigned to the player

Position

1. Position has been modified to determine valid position based moves. For example, a position cannot be moved to if it is already occupied. Therefore, relevant methods such as getIsValidMove() is created

Piece

1. No major changes to the piece class. Other than the new addition of the isSelected attribute and respective getters and setters. Which is a required data to keep track of the state of the piece

RuleChecker

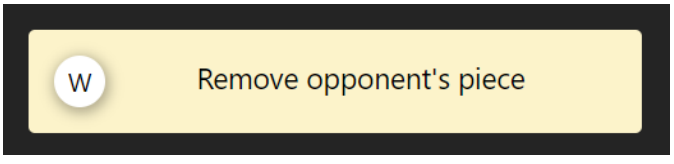
1. Although move validation has been handled by other classes (Player and Position etc). This RuleChecker is set up as an additional source of truth, it can detect is any new mills are detected and provide the removal of a piece is valid
2. Additionally, this class is designed to support the AI player in mind, as the AI player should not be able to have access to other classes. It would be most sensible to have the RuleChecker compute all the valid moves for the AI player to choose from. Hence , this class contains methods such as computing whether the player can move, fly, remove or place a piece.

The Move class is removed, the team figured that any move can be defined as an array of two [source, destination]. To remove a piece, the destination will be set to undefined. Whereas to place a piece, the source would be undefined.

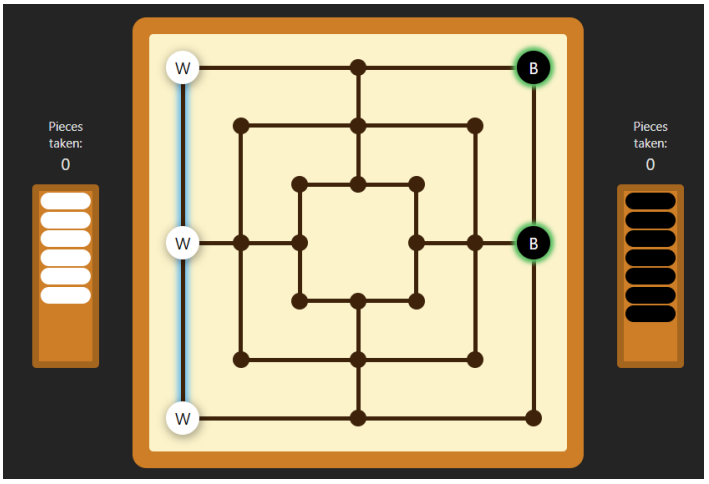
3.2. Quality attribute 1 - Usability

Usability is important for Nine Men's Morris since it determines the user experience and the ability to understand what is happening and how to play the game by interacting with the interface. Our implementation adheres to usability requirements by following several usability and accessibility guidelines.

Nielsen Usability Heuristic 1 - Visibility of system status



A status message above the board showing which player's turn it is and what they are to do.

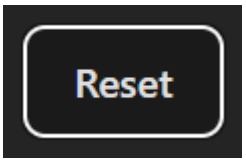


Valid moves highlighted, mills formed highlighted. Status of how many pieces left for each player and how many pieces taken are shown to indicate the state of the game.

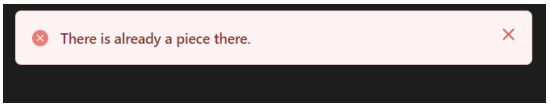
Feedback for invalid actions by the player. This also adheres to Nielsen Usability Heuristic 5 - Error prevention. Any illegal moves are prevented to stop users from being able to make mistakes. Nielsen Usability Heuristic 9 - help users recognize, diagnose, and recover from errors is also adhered to here, as the message helps users recognize why their attempted actions were invalid.

Popup showing when the game is over. Options to close the popup or start a new game are given.

Nielsen Usability Heuristic 3 - User control and freedom

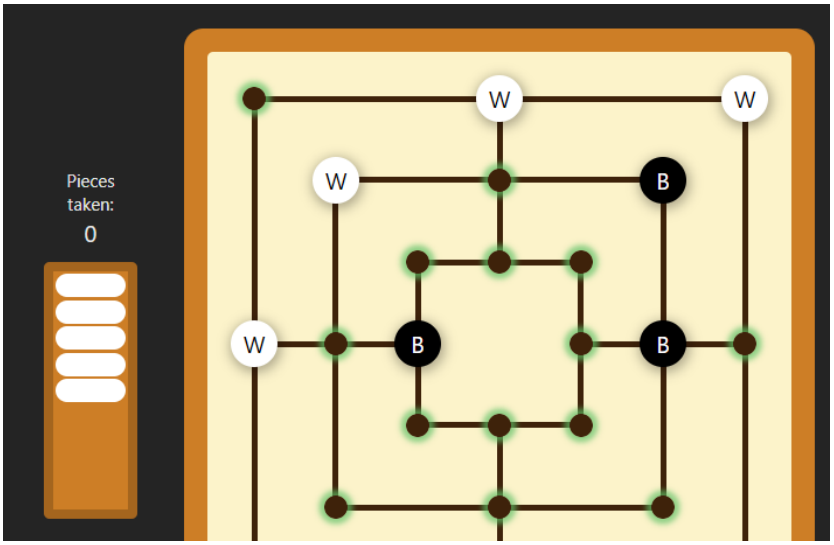


Reset button is always available for users to restart the game, so they don't feel stuck.



For pop ups, there are clear buttons to escape, and users can click anywhere on the dark parts of the screen to escape.

WCAG Principle 1 - Perceivable

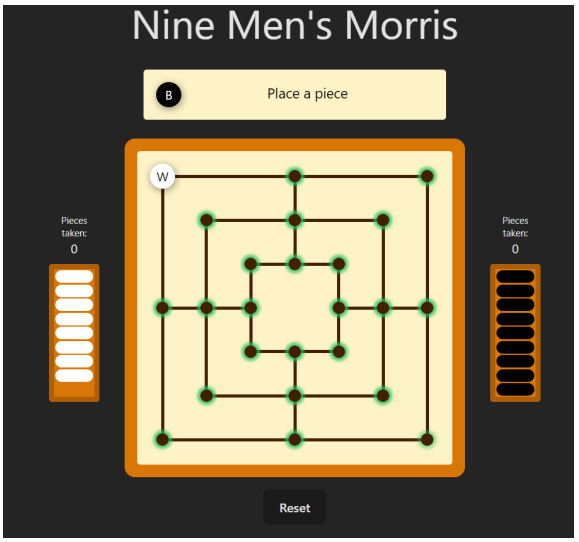


Good contrast between colours of board, mills, and pieces. Pieces are also marked with letters 'W' and 'B' to make them more easily distinguishable for people who have poorer vision. Valid position highlights have good contrast so they will be visible even to people with bad colour vision.

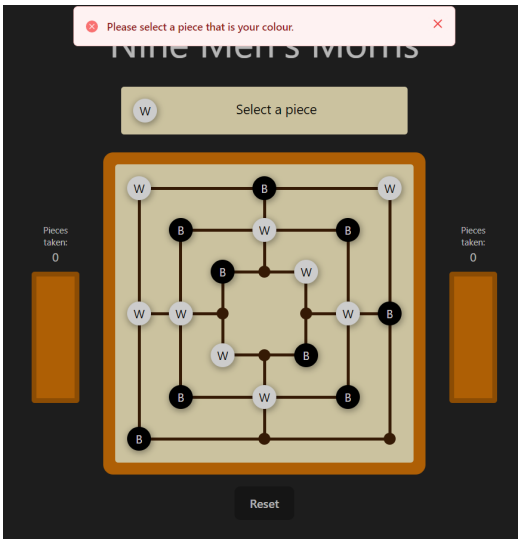
3.3. Quality attribute 2 - Reliability

Reliability is an essential quality attribute for the Nine Men's Morris game as it ensures the stability and consistency of the game's functionality. In the context of the Nine Men's Morris game, reliability ensures that the game functions as intended, without unexpected crashes, glitches, or inconsistencies. A reliable game gives confidence to the players, allowing them to focus on strategy and gameplay without worrying about technical issues. By implementing robust error handling, rigorous testing, and proper exception handling mechanisms, our design prioritises reliability to ensure a smooth and uninterrupted gaming experience.
Everytime the player makes a move, there will always be a hint for the player to follow. For example:

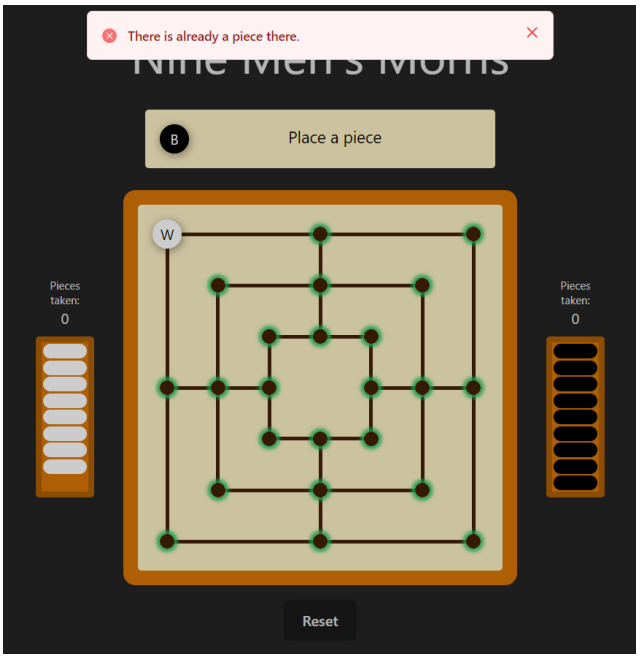
- The green outliner of dots on the board indicates that they are free to be occupied.



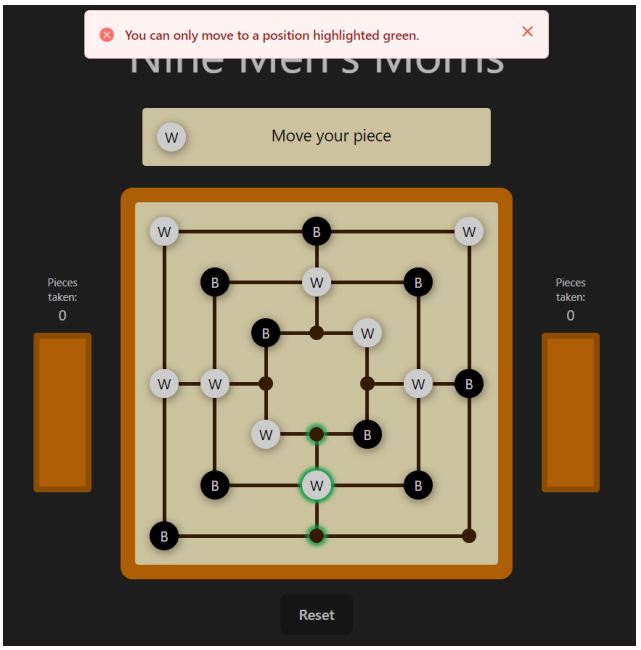
- The error prompt indicates either the dot has been occupied by other pieces or the piece which player wants to place on the board is not legally to be placed on this particular position(dot).



The above screenshot shows that the black player tried to place their piece on the position where the white piece is standing.



- The above screenshot shows that the white player tried to select the piece from that black player.



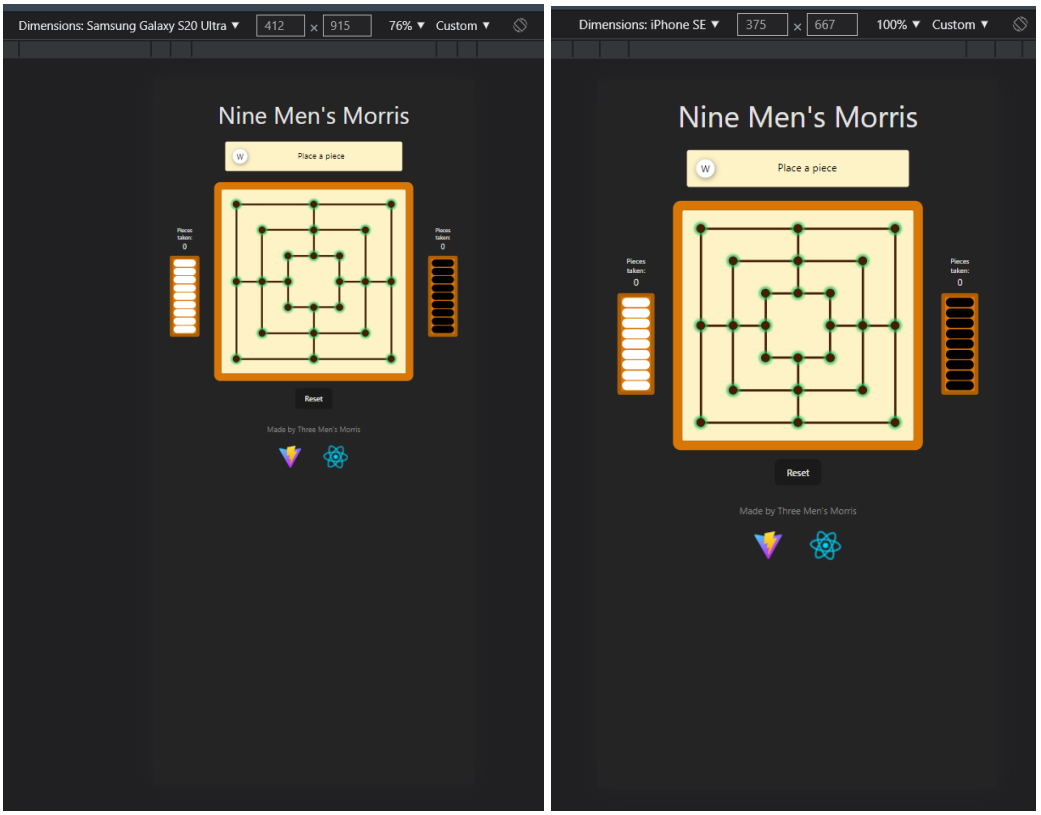
- The above screenshot shows that the white player tries to slide the piece with the green outliner to the position which is not legally to be moved into.

Overall, these UI effects and feedback has helped our 9MM to maintain the reliability of the game because the player has to stick to the game's rules when placing the pieces.

3.4. Quality attribute 3 - Portability

When designing the 9MM, we have considered that some players don't always want to play the game on their laptop/desktop since it's quite inconvenient for them to just play 1 - 2 games but need to wait 1-2 minutes for the laptop/desktop to be open. Our implementation is web-based which allows for users to play on a wide variety of devices.

Hence, our 9MM has coped with this concern by making the 9MM UI responsive, which means that players can play the 9MM game on either their mobile devices or laptop/desktop without worrying about the constraint of UI. Here are some examples of mobile device when viewing our board game:



3.5. Human value 1 - Equality

Equality is a relevant human value for the 9MM game as it promotes fairness and equal opportunities for all players. In our design, we have explicitly considered this value by ensuring that both players have an equal chance to succeed and win the game.

Besides implementing rules and mechanics that provide a level playing field, we have also considered that some players might have some impairment of vision like colorblindness. Hence, in addition to the colour used for distinguishing the difference between two players, we also use the letter to distinguish the difference between two players, such as letter "B" stands for Black player while letter "W" stands for White player. This design approach aligns with the value of equality, creating a balanced and inclusive gaming experience for all players.

3.6. Human value 2 - Successful

Success is another important human value in the context of the 9MM game. Success represents the sense of accomplishment and achievement that players derive from playing and winning the game.

We believe that the majority of players when playing a game always want to know how/when the game will end. Hence, if any of players in 9MM reach the one of the two conditions that has been discussed above in the non-trivial interaction 4 in the sequence diagram, the game's display will prompt the following message showing that the game is over and is followed by the winner's announcement.

Furthermore, the status message at the top of the screen as well as indicators of pieces taken make sure that the players know exactly what is happening and what they should strive for, helping them to gauge progress towards success.

4. Video Demonstration

YouTube unlisted link.

5. Meetings

27/04/2023, 6:00PM - 6:30PM

Location: In Person

Attendees: Liangdi, Harry (Quoc), Arthur (Jer)

Discussion: Initial work breakdown, understanding requirements.

03/05/2023, 7:30PM - 8:00PM

Location: Zoom

Attendees: Liangdi, Harry (Quoc), Arthur (Jer)

Discussion: What tasks do we still need to do, who will do what task and when.

11/05/202, 6:00PM - 6:30PM

Location: In Person

Attendees: Liangdi, Harry (Quoc), Arthur (Jer)

Discussion: How to implement basic rules and features in the code, design.

18/05/202, 6:00PM - 6:30PM

Location: In Person

Attendees: Liangdi, Harry (Quoc), Arthur (Jer)

Discussion: Explaining code, design. Go over remaining tasks, prepare for the video demo.

18/05/2023, 7:30PM - 9:00PM

Location: Zoom

Attendees: Liangdi, Harry (Quoc), Arthur (Jer)

Discussion: Record video demonstration.