YOUTUBE VIDEO LINK

https://youtu.be/WKfKIZhIC80

DESIGN RATIONALE

Explain *why* you have revised the architecture, if you have revised it. *What* has changed should be covered in the previous point. This one is about *why* it changed.

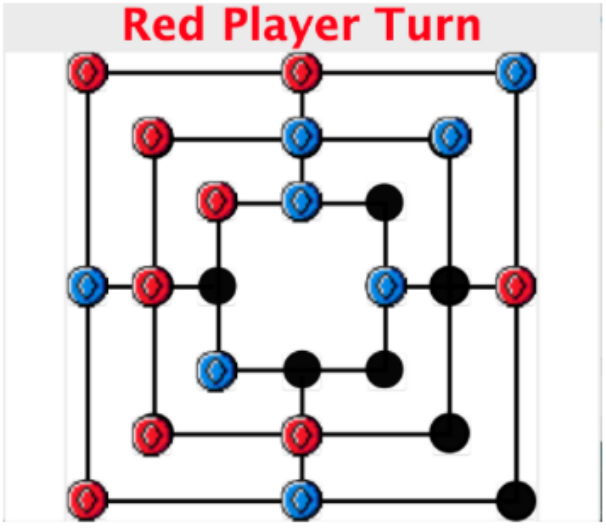
We have not made many changes in our architecture design, as we had a lot of discussion and modification in order to have the structure of the project. So, in this sprint, the only major change that we made was having the Player class managing the Action interaction instead of Game class.

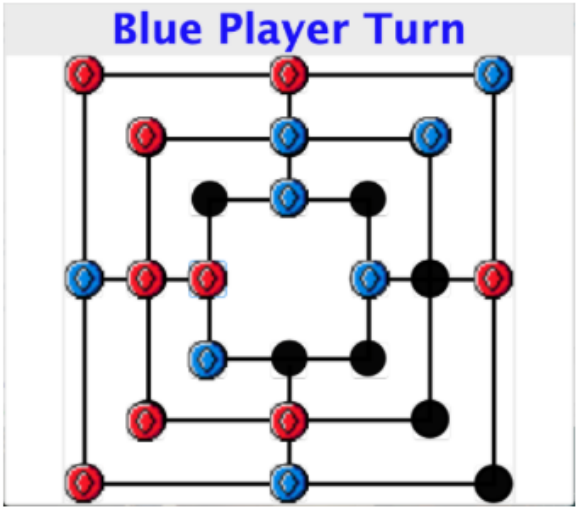
The reason for this change is we believe that the player is who makes the interaction, and the function should be executed right when the interaction occurs, while Game is only responsible for toggling player turn, keeping the game running and displaying the UI. So we moved the ActionListener from Game class to Player.

Explain 2-3 quality attributes (as non-functional requirements, e.g. usability, flexibility) that you consider relevant to the 9MM game and have explicitly considered in your design. Why are they relevant and important to your game? Show (provide evidence) how your design manifests these non-functional requirements.

The quality attributes that our group considers relevant to the 9MM game are: Correctness, Fairness and Strategy. And we have considered all attributes above into our game design.

Fairness: As in the 9MM game rule, players will take turns to place/move one of their pieces. This brings a feeling of excitement and calculation when players start their turn. So we must made sure to keep that quality in our game, as it is one of the game's crucial characteristics.





When a player turn starts, the game will change its label to inform players. During that time, the other player can not interact with their pieces. This is to maintain Fairness quality. As in coding, we want to make sure that all of the classes share the same amount of responsiblity. This helps us maintain, debug and modify it easier.

Strategy: In order to win the game, players must remove 7 pieces from their opponents with limited moving features. We have to consider both the game rules and circumstances that might happen. This helps us build a more consistent and stable algorithm for the game operation. We have also considered this quality in our diagrams as well. Imagining and visualising the diagrams save us lots of time and effort in implementing it in code.

```
if (gamePhase == Status.PHASE_1) {


if (pieceSet.getPieceSetSize() == 0) {
    //Testing phase 3
    gamePhase = Status.PHASE_2;
    System.out.println("Phase 2 starts");
}
```

```
if (board.getNumberOfBluePieces() == 3) {
    this.addStatus(Status.ACTIVE_FLY);
    gamePhase = Status.PHASE_3;
} else if (board.getNumberOfRedPieces() == 3) {
    this.hasStatus(Status.ACTIVE_FLY);
    gamePhase = Status.PHASE_3;
}
```

```
if (piece.hasStatus(Status.IN_MILL)&&(!board.isAllMill(!isRed)) ){
    previousPhase=gamePhase;
    gamePhase=Status.PHASE_REMOVE;

}
```

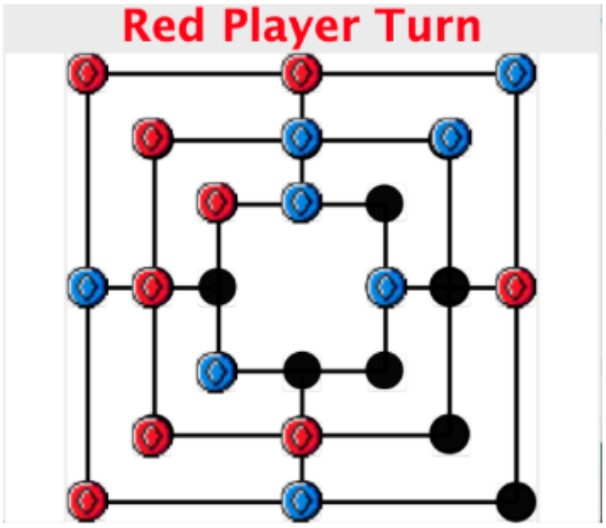
Correctness: Correctness is the critical attribute that helps us make a functional application. We always have to make sure that both coding and UI are implemented correctly.

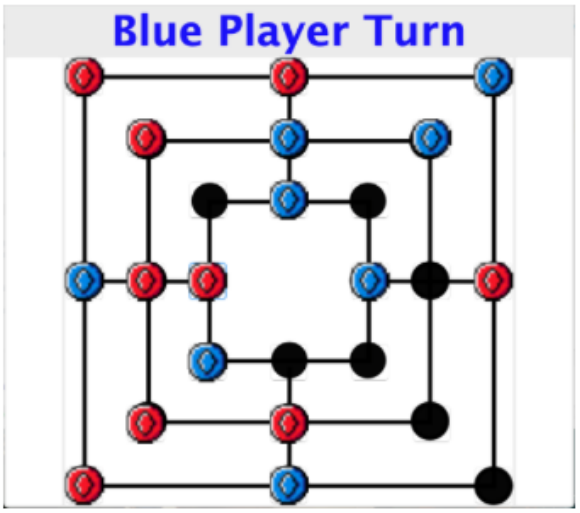
```
public enum Status {
    /**
     * Is on Red side
     */
    7 usages
    RED,
    /**
     * Is on blue side
     */
    5 usages
    BLUE,
    /**
     * This piece is in a Mill
     */
    9 usages
    IN_MILL,
    /**
     * Is not in a Mill
     */
    7 usages
    OUTSIDE_MILL,
    /**
     * Is in remove piece phase
     */
    6 usages
    PHASE_REMOVE,
```

```
for(int layer =0; layer <3; layer++){
    cor_layer = layer*LAYER_DISTANCE;
    for (int i =0; i<8; i++){
        Position position = new Position(layer,i);
        layeredPane.add(position, index: 2);

        switch (i) {
            case 0:
                position.setBounds(cor_layer, cor_layer, BUTTON_SIZE, BUTTON_SIZE);
                break;
            case 1:
                position.setBounds((round(BOARD_LENGTH-BUTTON_SIZE)/2), cor_layer, BUTTON_SIZE, BUTTON_SIZE);
                break;
            case 2:
                position.setBounds( x: BOARD_LENGTH-cor_layer-BUTTON_SIZE, cor_layer, BUTTON_SIZE, BUTTON_SIZE);
                break;
            case 3:
                position.setBounds( x: BOARD_LENGTH-cor_layer-BUTTON_SIZE, round((BOARD_LENGTH-BUTTON_SIZE)/2), BUTTON_SIZE, BUTTON_SIZE);
                break;
            case 4:
                position.setBounds( x: BOARD_LENGTH-cor_layer-BUTTON_SIZE,  y: BOARD_LENGTH-cor_layer-BUTTON_SIZE, BUTTON_SIZE, BUTTON_SIZE);
                break;
            case 5:
                position.setBounds(round((BOARD_LENGTH-BUTTON_SIZE)/2),  y: BOARD_LENGTH-cor_layer-BUTTON_SIZE, BUTTON_SIZE, BUTTON_SIZE);
                break;
            case 6:
                position.setBounds(cor_layer,  y: BOARD_LENGTH-cor_layer-BUTTON_SIZE, BUTTON_SIZE, BUTTON_SIZE);
                break;
```

In coding, we used a lot of default values. This helps avoid unnecessary changes in number and value, as well as ensure the game flow consistently.





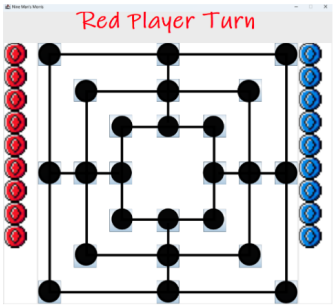
In UI design, each position will have a specific number/location on the screen. So when a player touches a position on screen, it can recognize the position correctly and place the piece inside it.

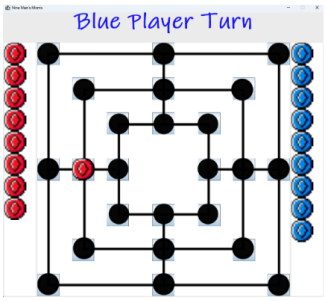
This helps the board look clean and organised. It is also crucial to use the correct piece color with the player colour to keep the game functional.

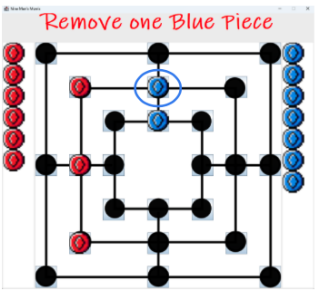
Explain at least one human value (from Schwartz's theory, e.g. achievement, tradition, freedom) that you consider relevant to the 9MM game and have explicitly considered in your design. Why is it relevant and important to your game? Show (provide evidence) how your design manifests this value.

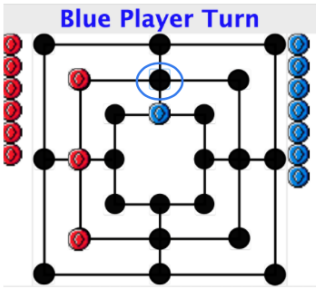
There are many human values that we have considered in our design that are related to the 9MM game. However, we have considered 2 most important are:

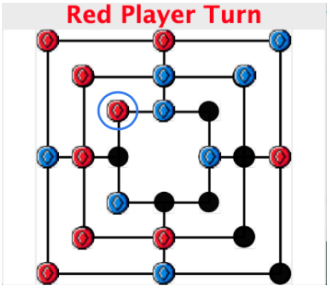
Achievement: In 9MM, removing an opponent's piece or winning the game is a form of achievement. In our project, building well-structured diagrams, creating a high-quality solution, and seeing the game running fluently and correctly all represent a form of achievement. And it is important because it always motivates us to find a better solution. This will always make our application work better as well as improving our skills.

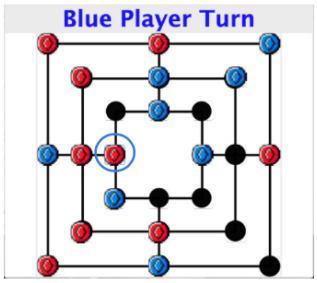












Self-direction: in the 9MM game, players will have to make their own decision to make the right move, and there is no limitation of time, they have a certain amount of freedom of thought.
This happens the same in our design.
In coding, we have faced multiple issues, crashes, errors that need to be fixed. We also make sure that our functions are implemented in an efficient and effective way. So we always have to come up with new ideas and make modifications.
In UI design, we have to learn a new language (JavaSwing) to create a functional UI.

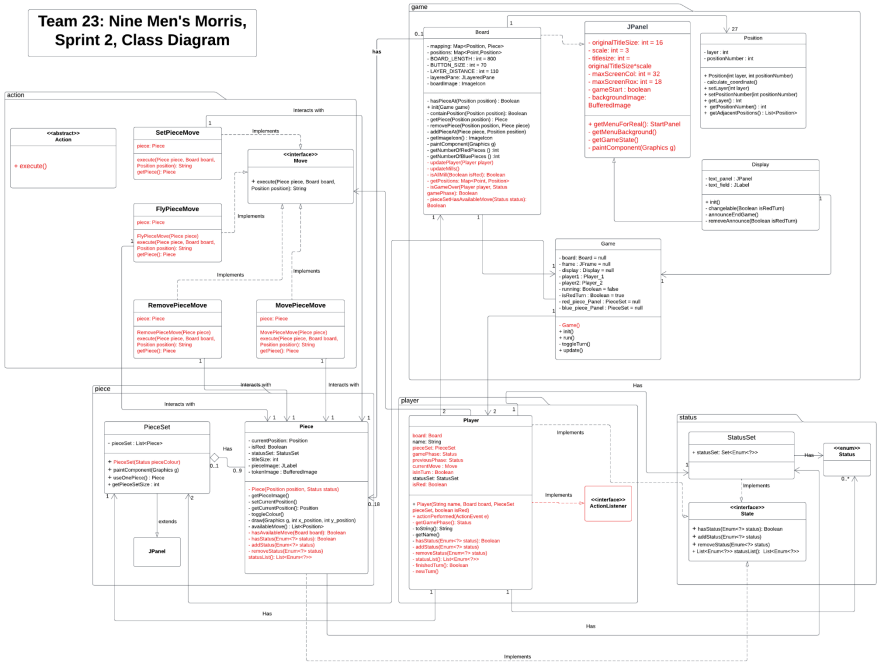
```
frame = new JFrame();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setResizable(false);
frame.setTitle("Nine Man's Morris");
frame.setBackground(Color.WHITE);
```

```
public class Player implements State, ActionListener
```

```
public class Display extends JPanel{
```

(JavaSwing example)

(changes in class diagram)

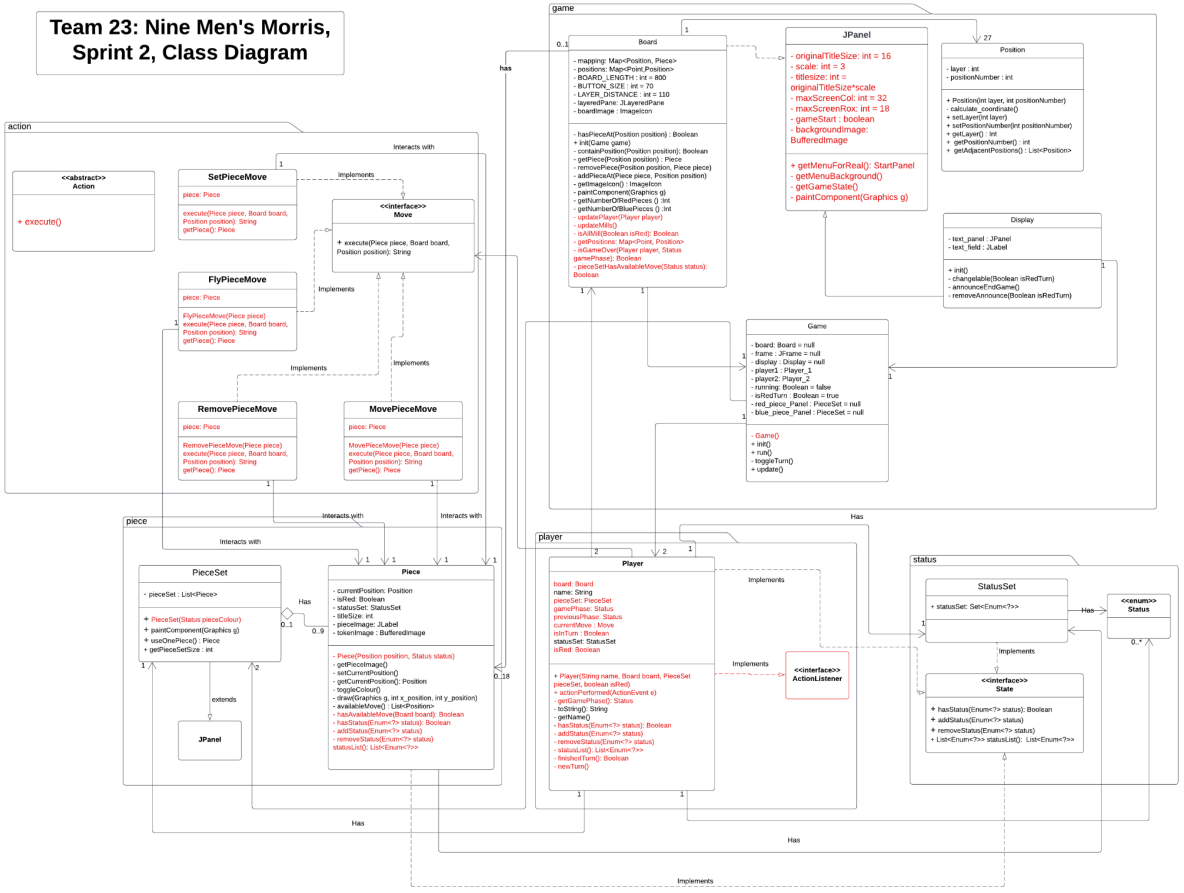


DIAGRAMS

UPDATED CLASS DIAGRAM

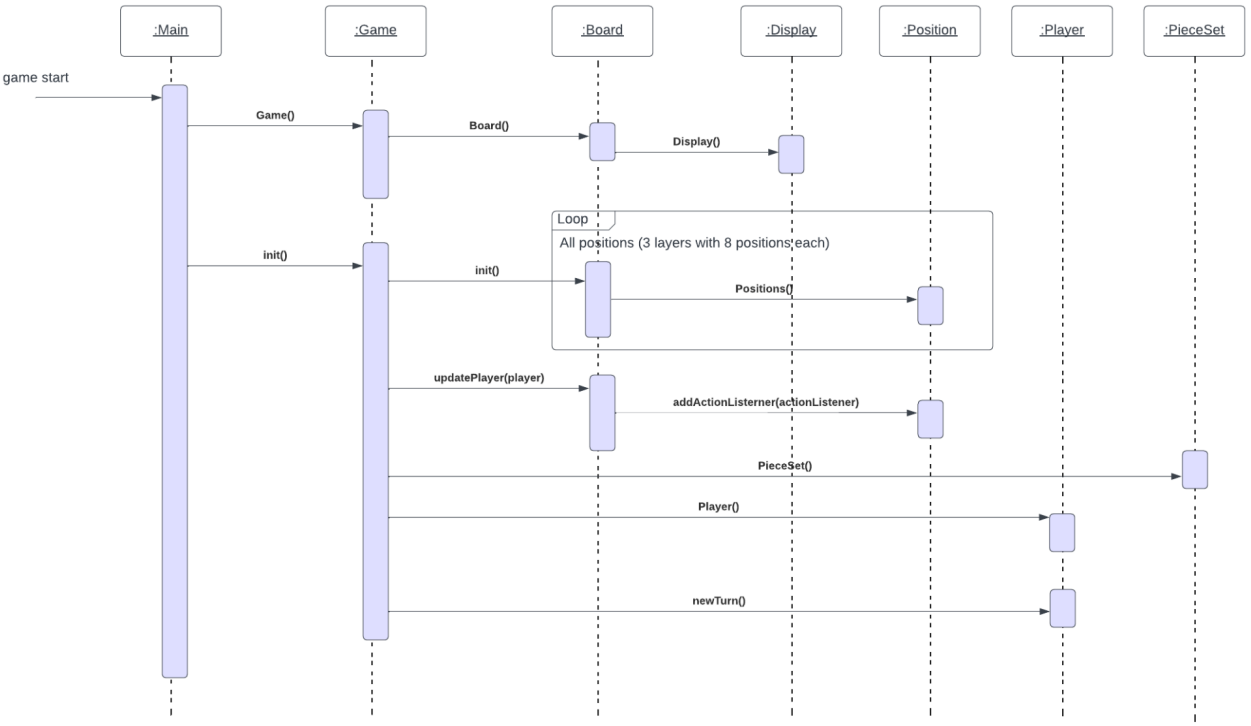
With red highlights changes

Team 23: Nine Men's Morris, Sprint 2, Class Diagram

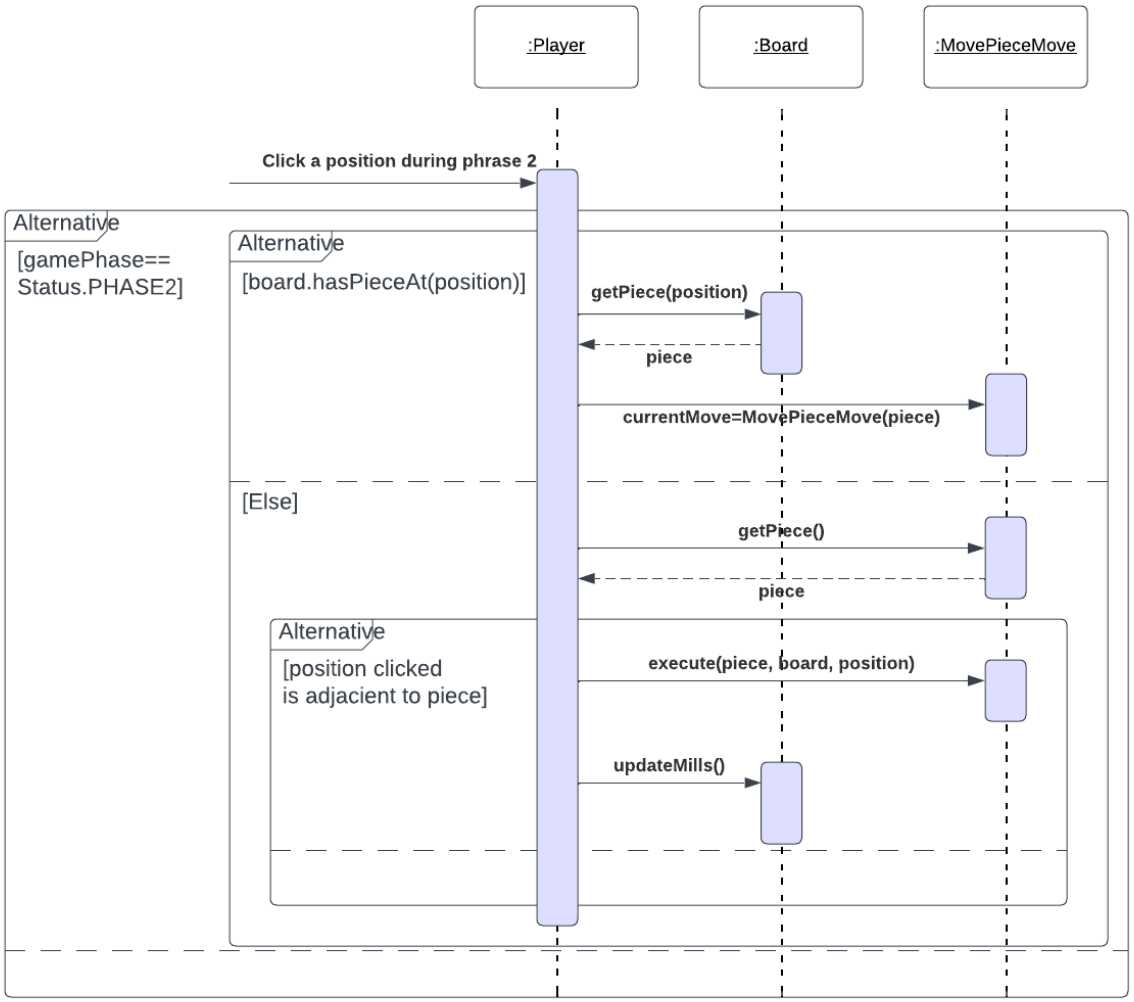


SEQUENCE DIAGRAMS

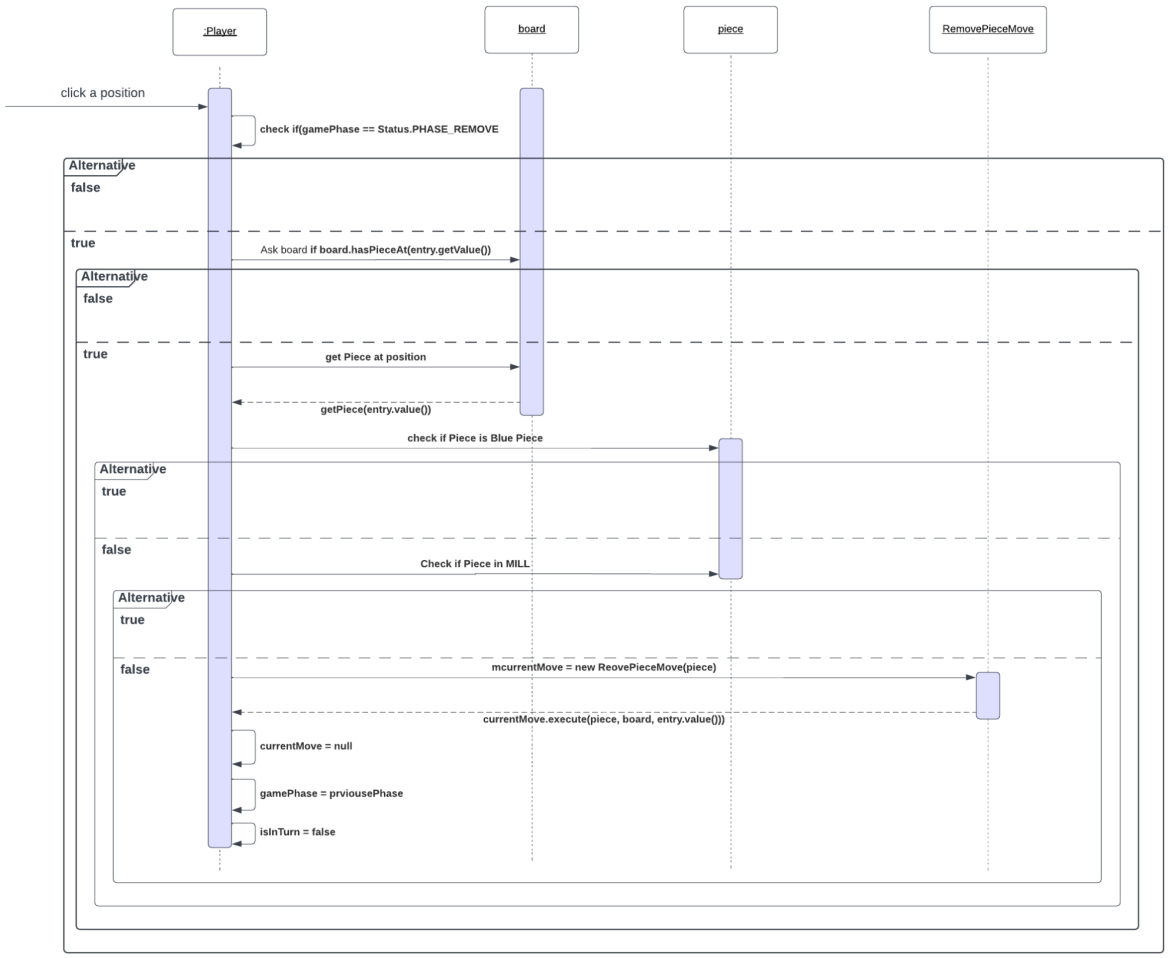
INITIALISATION


:Main
:Game
:Board
:Display
:Position
:Player
:PieceSet
game start
Game()
Board()
Display()
Loop
All positions (3 layers with 8 positions each)
init()
init()
Positions()
updatePlayer(player)
addActionListerner(actionListener)
PieceSet()
Player()
newTurn()

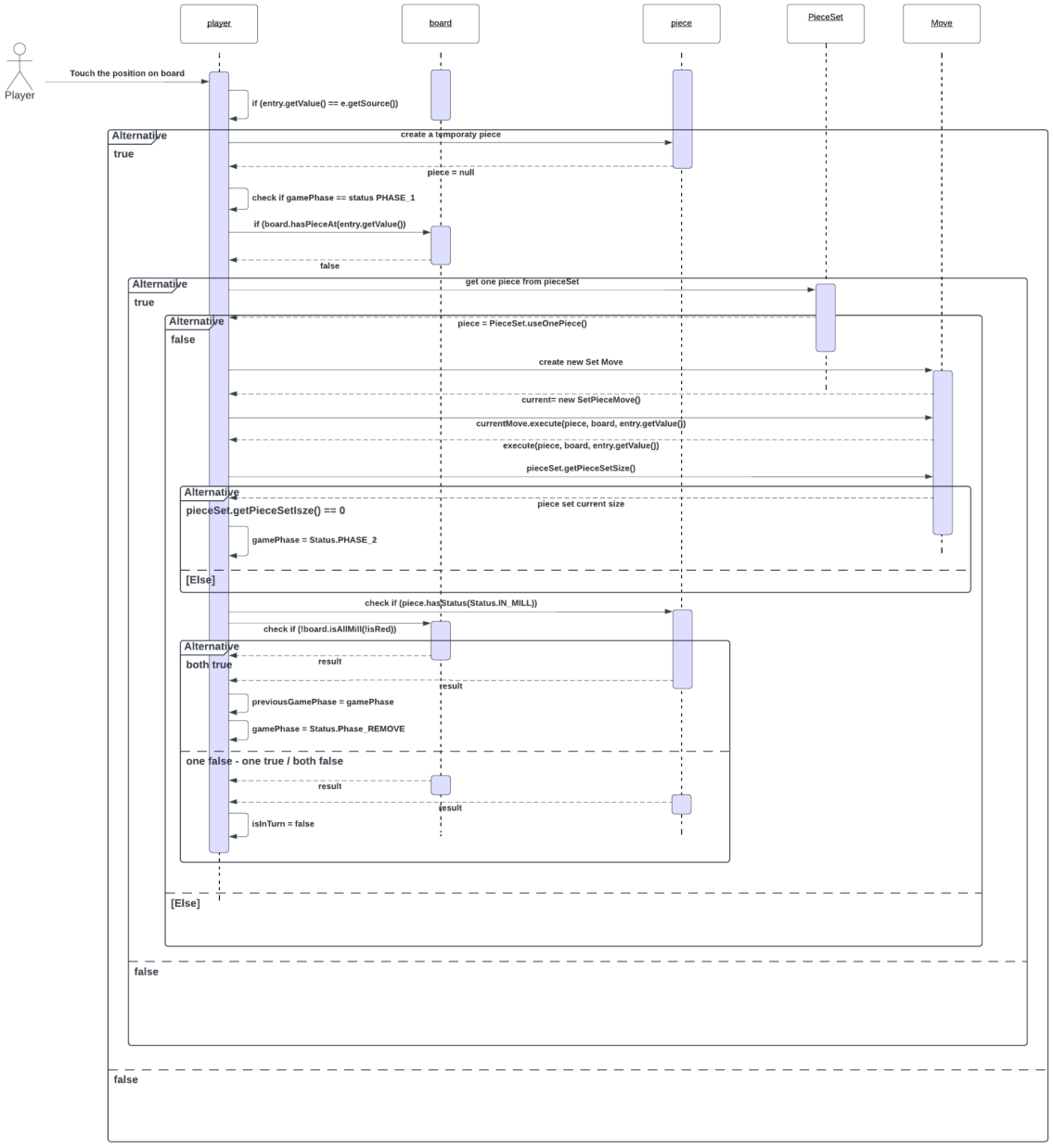
MOVE PIECE


:Player
:Board
:MovePieceMove
Click a position during phrase 2
Alternative
[gamePhase==
Status.PHASE2]
Alternative
[board.hasPieceAt(position)]
getPiece(position)
piece
currentMove=MovePieceMove(piece)
[Else]
getPiece()
piece
Alternative
[position clicked
is adjacient to piece]
execute(piece, board, position)
updateMills()

RED PLAYER REMOVE BLUE PIECE



SET PIECE


player
board
piece
PieceSet
Move
Player
Touch the position on board
if (entry.getValue() == e.getSource())
Alternative
true
create a temporaty piece
piece = null
check if gamePhase == status PHASE_1
if (board.hasPieceAt(entry.getValue())
false
Alternative
true
get one piece from pieceSet
Alternative
false
piece = PieceSet.useOnePiece()
create new Set Move
current= new SetPieceMove()
currentMove.execute(piece, board, entry.getValue())
execute(piece, board, entry.getValue())
pieceSet.getPieceSetSize()
Alternative
pieceSet.getPieceSetIsze() == 0
piece set current size
gamePhase = Status.PHASE_2
[Else]
check if (piece.hasStatus(Status.IN_MILL))
check if (!board.isAllMill(!isRed))
Alternative
both true
result
result
previousGamePhase = gamePhase
gamePhase = Status.Phase_REMOVE
one false - one true / both false
result
result
isInTurn = false
[Else]
false
false

CHECK IF BLUE PIECE IS IN MILL?