FIT3077 Software Engineering: Architecture and Design

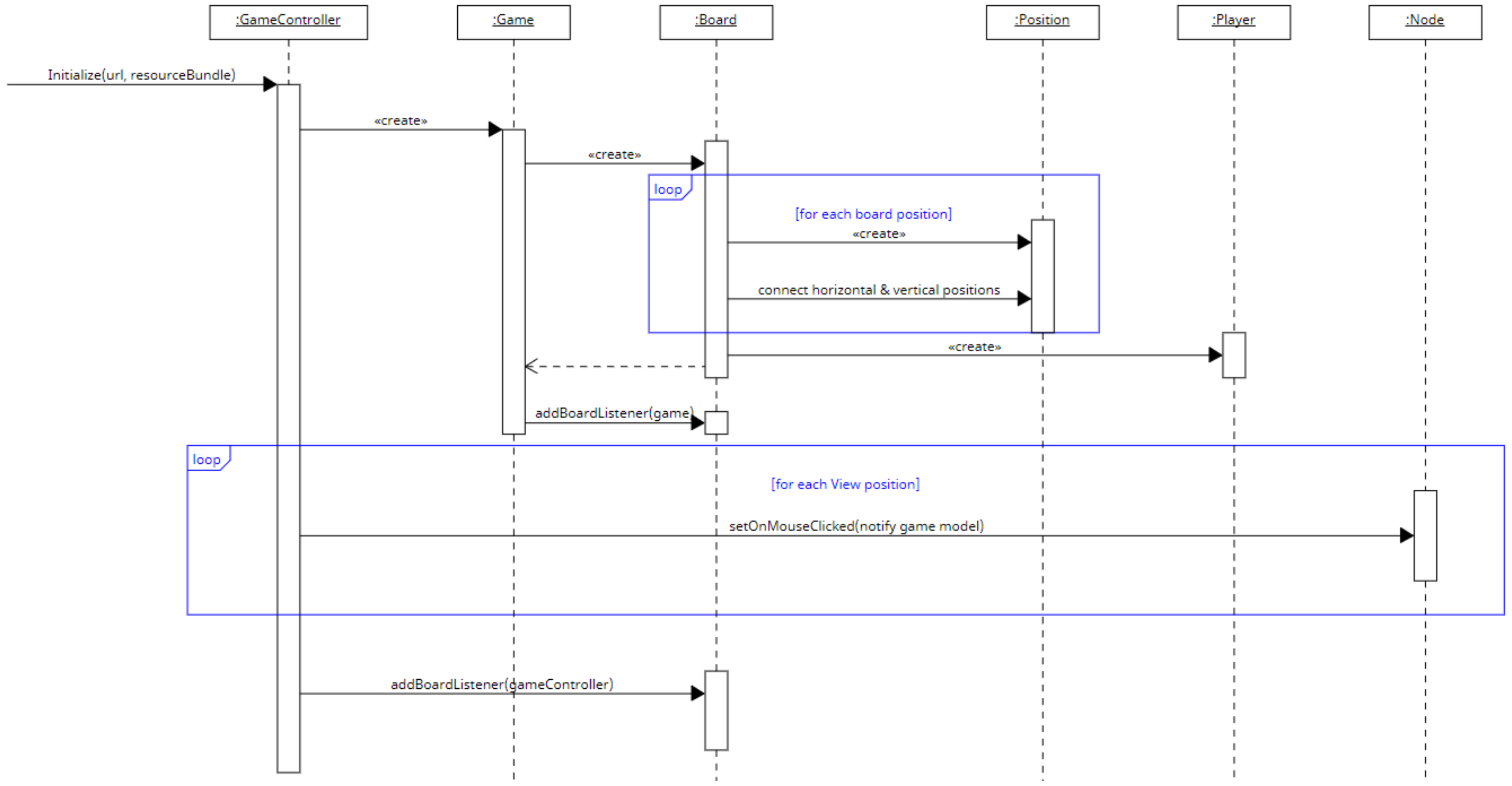
Assignment 3: Sprint Three

MA_Friday2pm_Team5

Table of Contents

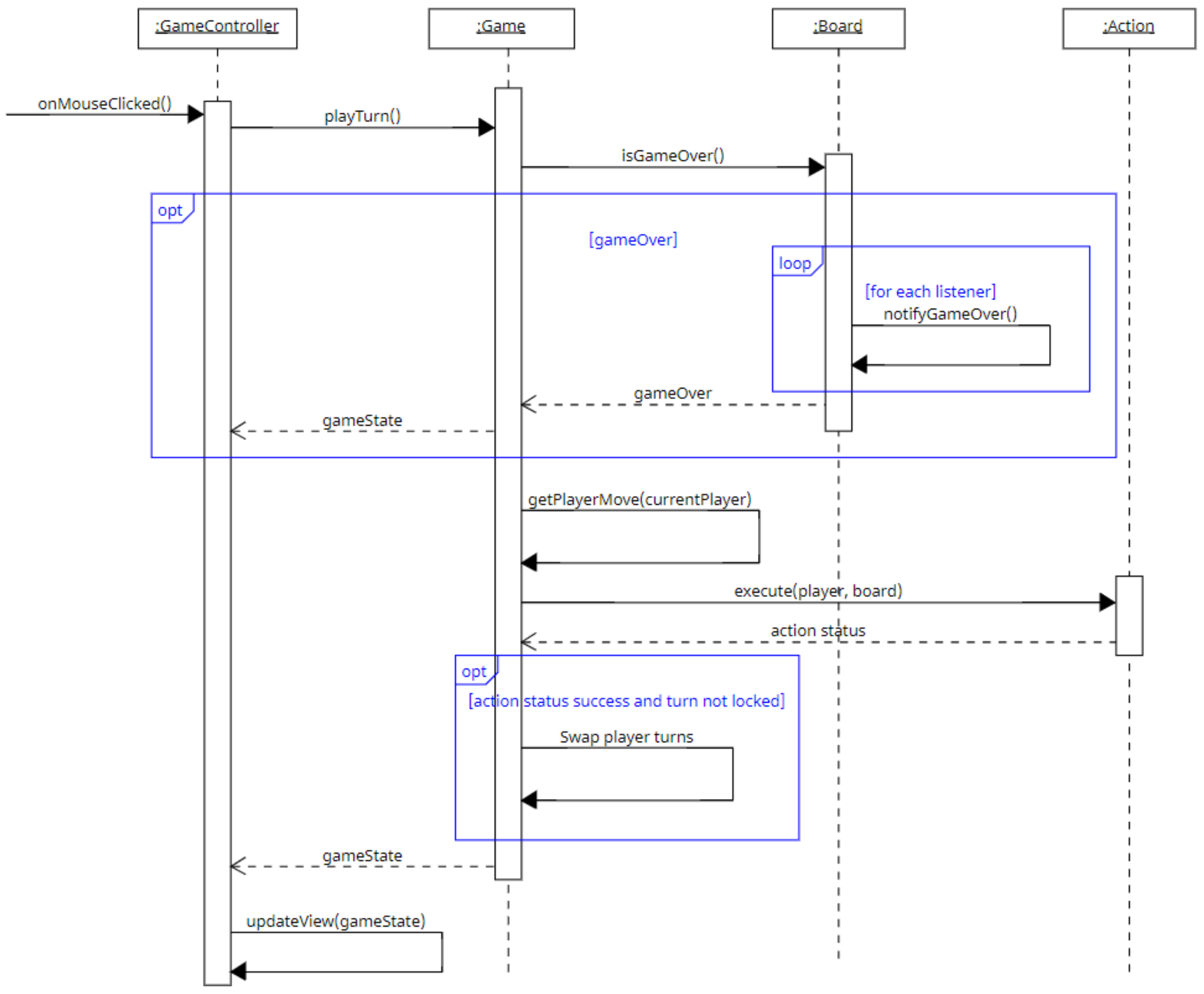
Architecture

Sequence Diagram: Bootstrap (Initialisation)

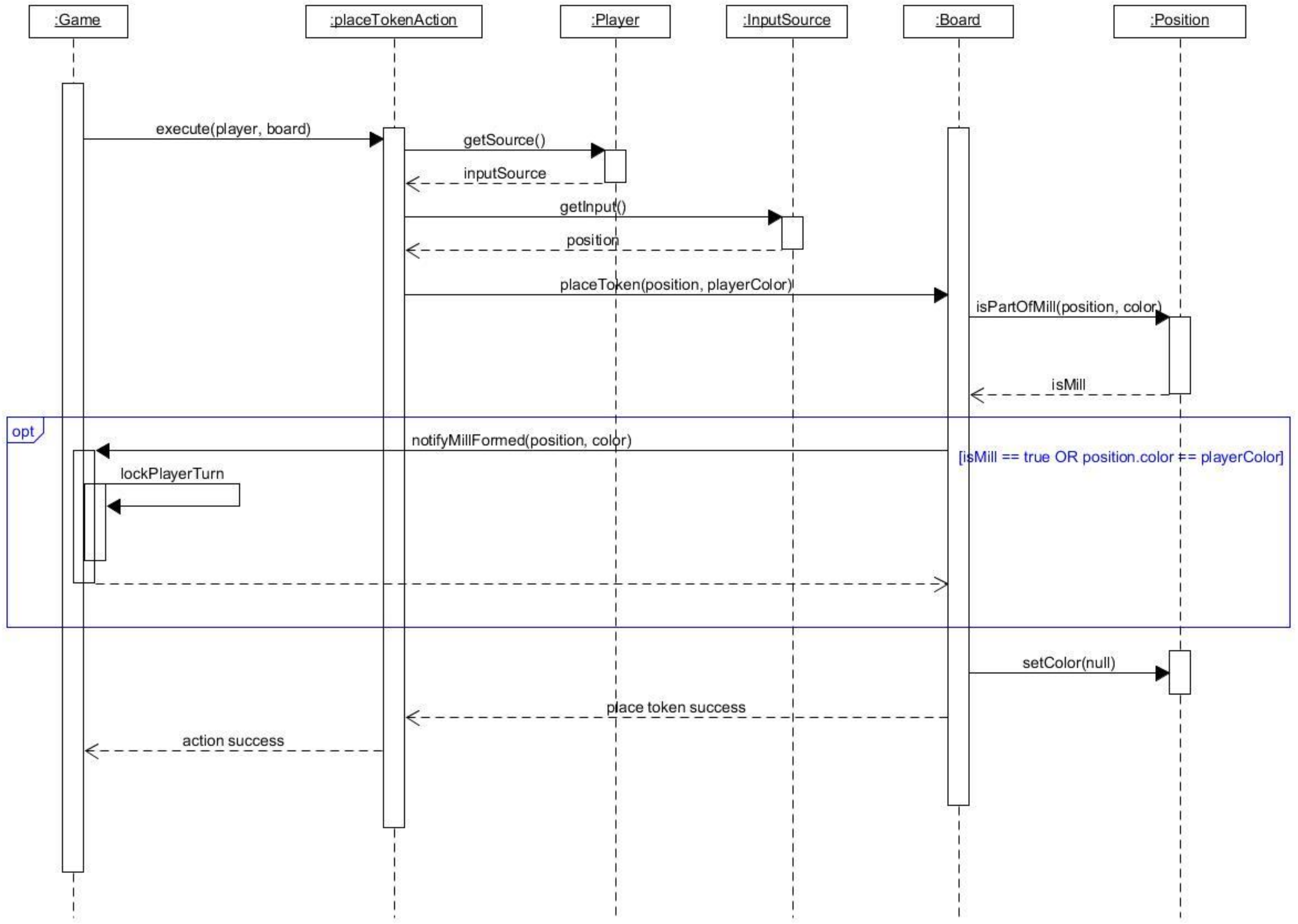

:GameController
:Game
:Board
:Position
:Player
:Node
Initialize(url, resourceBundle)
«create»
«create»
loop
[for each board position]
«create»
connect horizontal & vertical positions
«create»
addBoardListener(game)
loop
[for each View position]
setOnMouseClicked(notify game model)
addBoardListener(gameController)

Sequence Diagram: MVC Model

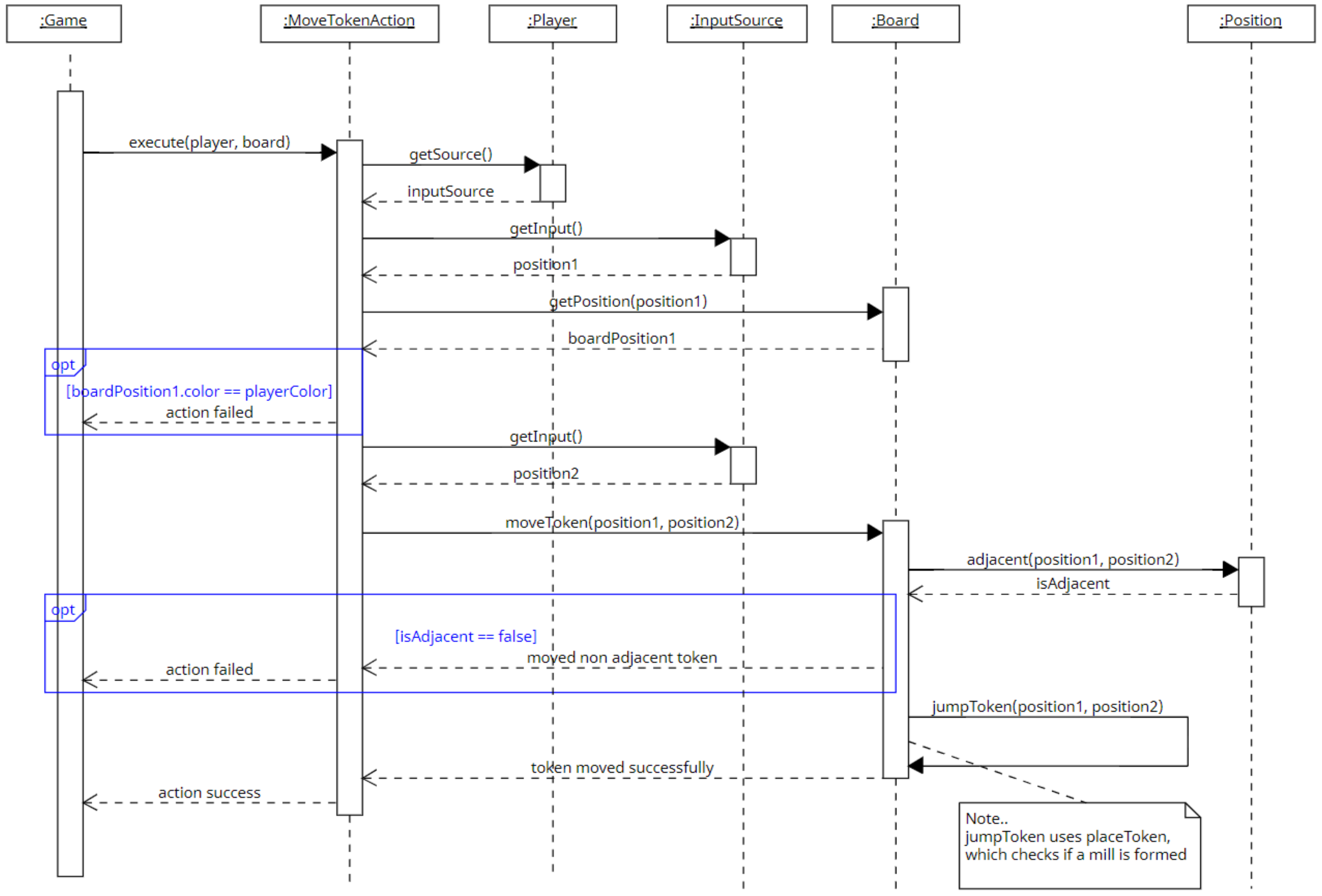
Scenario: This is the general case of how in general the game is run, starting from the controller where a mouse click is triggered to notify the game model, game model handles the game rules and controller updates the View after.



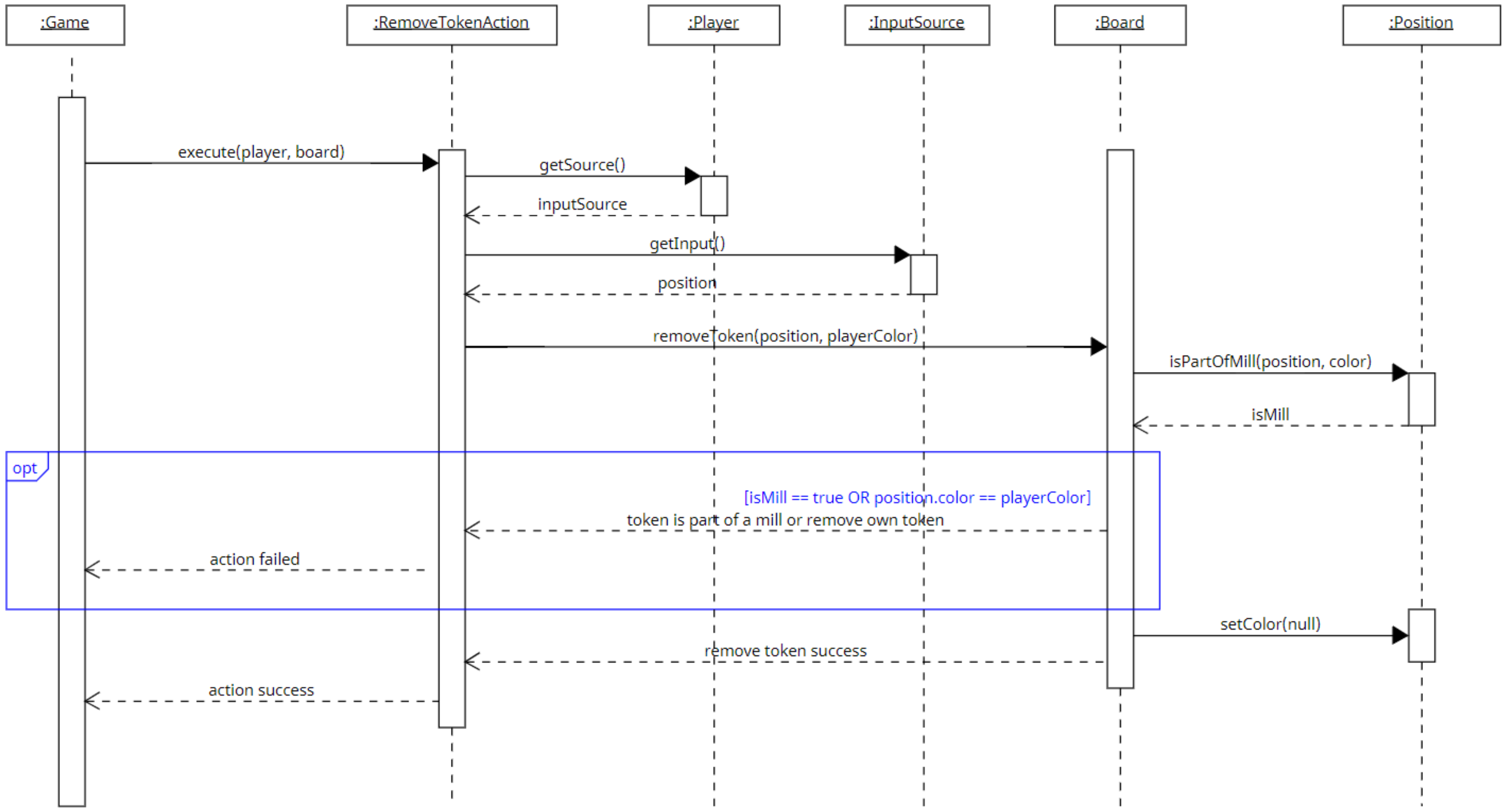
Sequence Diagram: Placing a Token and Forming a Mill



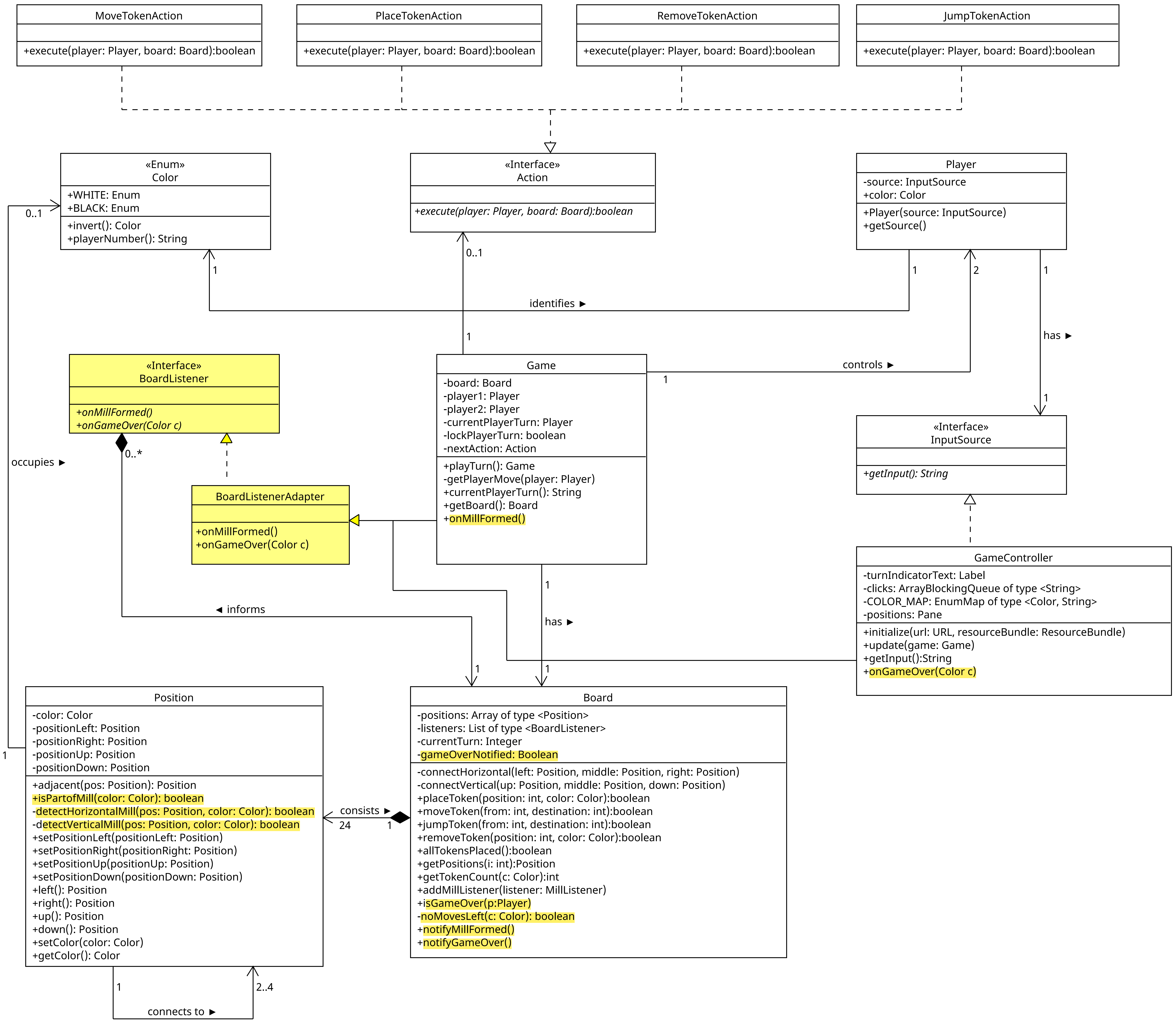
Sequence Diagram: Moving a Token to Adjacent Position (Mid-Game)



Sequence Diagram: Removing a Token

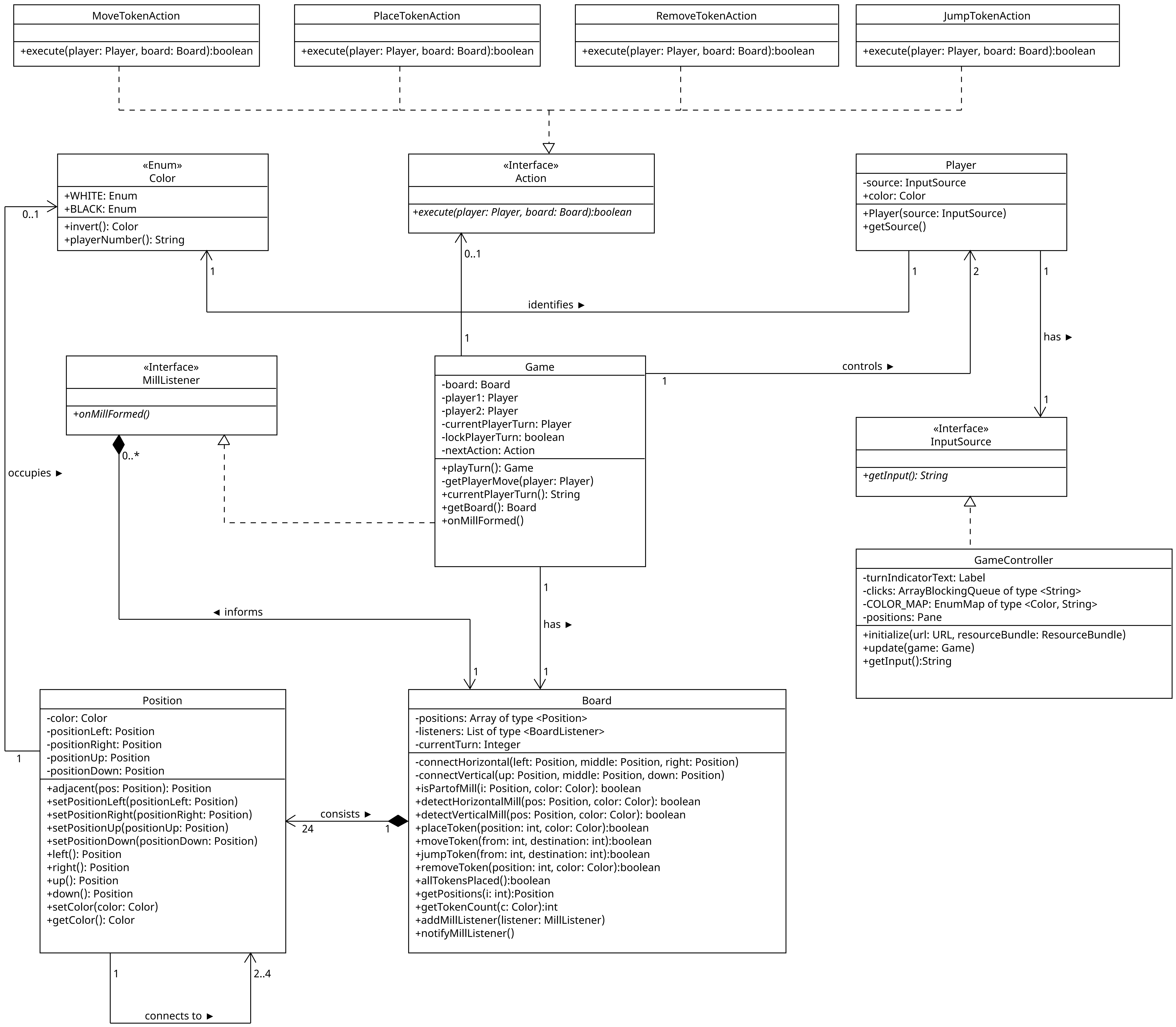


Architecture
Class Diagram
Sprint Three
Changes in Yellow


MoveTokenAction
+execute(player: Player, board: Board):boolean
PlaceTokenAction
+execute(player: Player, board: Board):boolean
RemoveTokenAction
+execute(player: Player, board: Board):boolean
JumpTokenAction
+execute(player: Player, board: Board):boolean
«Enum»
Color
+WHITE: Enum
+BLACK: Enum
+invert(): Color
+playerNumber(): String
«Interface»
Action
+execute(player: Player, board: Board):boolean
Player
-source: InputSource
+color: Color
+Player(source: InputSource)
+getSource()
0..1
1
0..1
1
2
1
identifies ►
has ►
1
controls ►
1
1
«Interface»
BoardListener
+onMillFormed()
+onGameOver(Color c)
0..*
occupies ►
BoardListenerAdapter
+onMillFormed()
+onGameOver(Color c)
Game
-board: Board
-player1: Player
-player2: Player
-currentPlayerTurn: Player
-lockPlayerTurn: boolean
-nextAction: Action
+playTurn(): Game
-getPlayerMove(player: Player)
+currentPlayerTurn(): String
+getBoard(): Board
+onMillFormed()
«Interface»
InputSource
+getInput(): String
GameController
-turnIndicatorText: Label
-clicks: ArrayBlockingQueue of type <String>
-COLOR_MAP: EnumMap of type <Color, String>
-positions: Pane
+initialize(url: URL, resourceBundle: ResourceBundle)
+update(game: Game)
+getInput():String
+onGameOver(Color c)
1
◄ informs
has ►
1
1
1
Position
-color: Color
-positionLeft: Position
-positionRight: Position
-positionUp: Position
-positionDown: Position
+adjacent(pos: Position): Position
+isPartofMill(color: Color): boolean
-detectHorizontalMill(pos: Position, color: Color): boolean
-detectVerticalMill(pos: Position, color: Color): boolean
+setPositionLeft(positionLeft: Position)
+setPositionRight(positionRight: Position)
+setPositionUp(positionUp: Position)
+setPositionDown(positionDown: Position)
+left(): Position
+right(): Position
+up(): Position
+down(): Position
+setColor(color: Color)
+getColor(): Color
consists ►
24
1
Board
-positions: Array of type <Position>
-listeners: List of type <BoardListener>
-currentTurn: Integer
-gameOverNotified: Boolean
-connectHorizontal(left: Position, middle: Position, right: Position)
-connectVertical(up: Position, middle: Position, down: Position)
+placeToken(position: int, color: Color):boolean
+moveToken(from: int, destination: int):boolean
+jumpToken(from: int, destination: int):boolean
+removeToken(position: int, color: Color):boolean
+allTokensPlaced():boolean
+getPositions(i: int):Position
+getTokenCount(c: Color):int
+addMillListener(listener: MillListener)
+isGameOver(p:Player)
-noMovesLeft(c: Color): boolean
+notifyMillFormed()
+notifyGameOver()
1
2..4
connects to ►

]

Architecture
Class Diagram
Sprint Two


MoveTokenAction
+execute(player: Player, board: Board):boolean
PlaceTokenAction
+execute(player: Player, board: Board):boolean
RemoveTokenAction
+execute(player: Player, board: Board):boolean
JumpTokenAction
+execute(player: Player, board: Board):boolean
«Enum»
Color
+WHITE: Enum
+BLACK: Enum
+invert(): Color
+playerNumber(): String
«Interface»
Action
+execute(player: Player, board: Board):boolean
Player
-source: InputSource
+color: Color
+Player(source: InputSource)
+getSource()
0..1
1
0..1
identifies ►
1
2
1
has ►
1
«Interface»
MillListener
+onMillFormed()
Game
-board: Board
-player1: Player
-player2: Player
-currentPlayerTurn: Player
-lockPlayerTurn: boolean
-nextAction: Action
+playTurn(): Game
-getPlayerMove(player: Player)
+currentPlayerTurn(): String
+getBoard(): Board
+onMillFormed()
controls ►
1
1
«Interface»
InputSource
+getInput(): String
GameController
-turnIndicatorText: Label
-clicks: ArrayBlockingQueue of type <String>
-COLOR_MAP: EnumMap of type <Color, String>
-positions: Pane
+initialize(url: URL, resourceBundle: ResourceBundle)
+update(game: Game)
+getInput():String
0..*
occupies ►
◄ informs
1
has ►
1
1
Position
-color: Color
-positionLeft: Position
-positionRight: Position
-positionUp: Position
-positionDown: Position
+adjacent(pos: Position): Position
+setPositionLeft(positionLeft: Position)
+setPositionRight(positionRight: Position)
+setPositionUp(positionUp: Position)
+setPositionDown(positionDown: Position)
+left(): Position
+right(): Position
+up(): Position
+down(): Position
+setColor(color: Color)
+getColor(): Color
1
consists ►
24
1
Board
-positions: Array of type <Position>
-listeners: List of type <BoardListener>
-currentTurn: Integer
-connectHorizontal(left: Position, middle: Position, right: Position)
-connectVertical(up: Position, middle: Position, down: Position)
+isPartofMill(i: Position, color: Color): boolean
+detectHorizontalMill(pos: Position, color: Color): boolean
+detectVerticalMill(pos: Position, color: Color): boolean
+placeToken(position: int, color: Color):boolean
+moveToken(from: int, destination: int):boolean
+jumpToken(from: int, destination: int):boolean
+removeToken(position: int, color: Color):boolean
+allTokensPlaced():boolean
+getPositions(i: int):Position
+getTokenCount(c: Color):int
+addMillListener(listener: MillListener)
+notifyMillListener()
1
2..4
connects to ►

Revised Design Rationale for Sprint 3

The software architecture of the game largely remains the same from Sprint 2, except for one particular aspect where MillListener has been changed to BoardListener and BoardListenerAdapter in Sprint 3. Some refactoring was also done to move methods to suitable classes.

BoardListener Interface

Originally, the team used listeners for mill detection in Board class that allows the player who forms a mill to have an extra turn to remove the token. This implementation requires the Board class to track a list of listeners. However, when implementing the full-fledged game for 9MM the team encountered another issue, where the Board class has another “game over" event that has to notify it to a separate list of listeners as the original interface only supports one method. In other words, we want the Board class to notify the controller that the game is over and proceed with the game victory screen. The 2 viable approaches to solve this issue is tabulated below:

Approach A: Introduce another GameOverListener interface and Board Class has to keep track of a separate list of event listeners. Advantages ● Classes that implement any listener interfaces does not have any useless methods Disadvantages ● Board class has to keep track of a new list of event listeners for different events, which violates open closed principle. ● Each event having its own interface results in lots of interfaces in the future, may be over complicated for the program	**Approach B:** Combine Game Over listener and Mill Formed Listener into a single interface. Introduce a new Adapter class that has default empty body implementations. Classes that want to listen to one of the events inherit the adapter class, and override their interested methods. Advantages ● Board class only has to keep track of a single list of event listeners. ● Listeners don’t have to implement all event handling methods if inherited from Adapter Disadvantages ● Using inheritance for listeners strongly couples them to the Adapter, and disable their option of inheriting from other classes ● Has useless methods from the parent class that may not be used ● The interface violates interface segregation principle exposing methods not every class needs
Decision The team decided to go with Approach B, as the team values the readability of the code then introducing a different listener for just a single method. While this method introduces useless inherited methods from the parent adapter class, the team thinks it is a good compromise for eliminating another list of listeners in Board. In return, we get a BoardListener and adapter class that the implementing classes only need to override the events they are interested in.	

Non-Functional Requirements - Usability

For the game to be played, a user interface needs to be added for the game. The team has 2 choices of implementing the user interface:

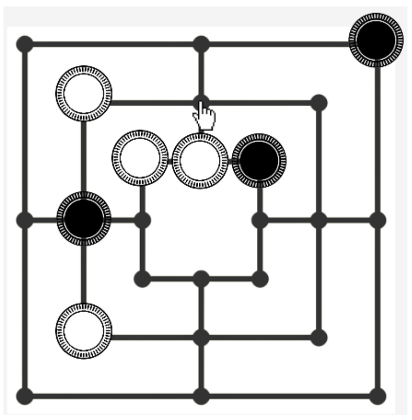
Approach A: Use a command line interface to show the board state of the player. The player has to input game moves from the keyboard. Advantages ● Easy to develop user interface ● No dependencies needed Disadvantages ● Non user-friendly controls to play the game ● Difficult to visualize game board properly in command line	**Approach B:** Use a graphical user interface to render the game. The player input game moves through their mouse cursor by clicking on the desired position. Advantages ● Good graphics that nicely depict 9MM ● Intuitive game controls with cursor Disadvantages ● Had to use dependencies for interface visualizations ● Bloated file size if dependencies not managed correctly
Decision In terms of usability, implementing a graphical user interface is the minimum requirement these days. The second approach is chosen to enable the player to interact with tokens by simply clicking on a position or a token they want to interact with. With that, the user can have a more enjoyable game experience through better visualization and intuitive controls.	

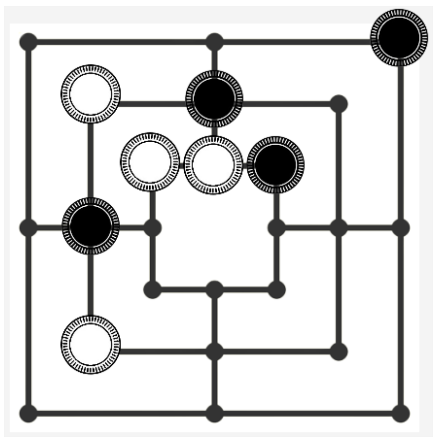
Evidence

This section showcases a few user interface elements that enhance the user experience in playing 9MM.

Intuitive Controls

The user can directly click on the board to interact with the token or position, which is intuitive and lets the player focus on playing the game with minimal distractions.


User hovers over the position they want to place the token

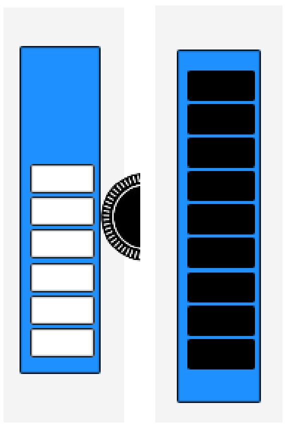

The token is placed after it is clicked

Turn Prompt

Player 1 Turn

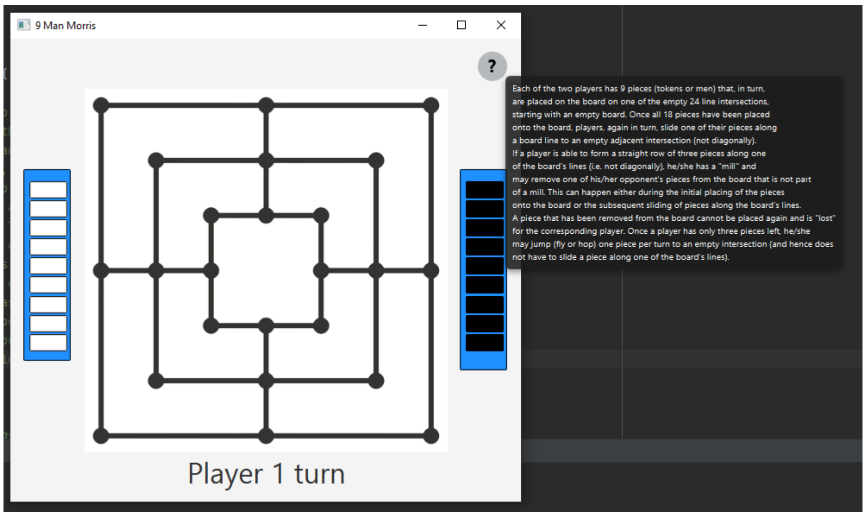
This text informs the user whose turn it is to perform an action. This prompt may be helpful for newer players who are unfamiliar with the rules of the game, and may assist them in reminding them whose turn it is to make a move on a Token.

Token Count Display



At both sides of the Game we included 2 sidebars to tell each player how many tokens they have left. This is useful to remind the players which stage of the game they are at, and how many tokens are left until they reach the next stage of the game.

Help Documentation Tooltip



Each of the two players has 9 pieces (tokens or men) that, in turn,
are placed on the board on one of the empty 24 line intersections,
starting with an empty board. Once all 18 pieces have been placed
onto the board, players, again in turn, slide one of their pieces along
a board line to an empty adjacent intersection (not diagonally).
If a player is able to form a straight row of three pieces along one
of the board's lines (i.e. not diagonally), he/she has a "mill" and
may remove one of his/her opponent's pieces from the board that is not part
of a mill. This can happen either during the initial placing of the pieces
onto the board or the subsequent sliding of pieces along the board's lines.
A piece that has been removed from the board cannot be placed again and is "lost"
for the corresponding player. Once a player has only three pieces left, he/she
may jump (fly or hop) one piece per turn to an empty intersection (and hence does
not have to slide a piece along one of the board's lines).

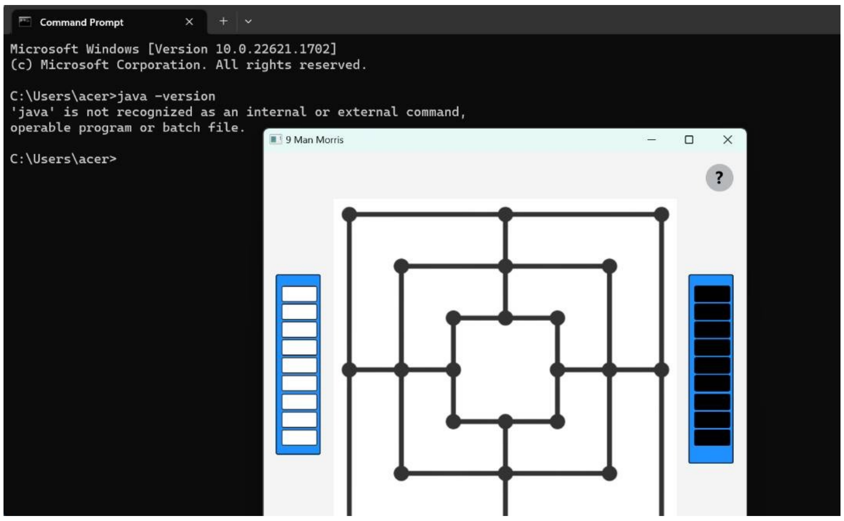
In case the player gets stuck and has no idea what to do, the player can always refer to the tooltip to read the game rules to proceed with the next course of action to take.

Non-Functional Requirements - Portability

During the development process, the team has used several frameworks and needs to find a way to export the program to the end user. In order to make sure that the program is accessible, the team discussed 2 approaches for porting the software:

Approach A: Develop the game to run natively on a chosen platform (Windows). The user does not have to do any external installations after the game is downloaded. Advantages: ● Game is ready to be played after it is downloaded, which wasn't possible before due to external dependencies Disadvantages: ● Increased file sizes. All the libraries, dependencies and the runtime environment needs to be packaged ● The software is no longer a single executable, and needs to export the entire directory to start playing the game	**Approach B:** Provide clear instructions on which versions of the dependencies are needed for the game to run. The game cannot be played without the dependency. Advantages: ● Minimal file sizes ● Single executable that can be easily exported Disadvantages: ● Users need to include the dependency to play the game ● The user may not know what went wrong in the installation process as it is manually done
Approach A is chosen for the team. The team aims for the software to be self-contained, and the only requirement for running it should be a Windows operating system. As such, the game should be bundled with all its required dependencies, libraries, and resources within a single package. This ensures that it can run independently without relying on external dependencies or additional installations, providing a very minimal setup for the user. One drawback of bundling the application together in such a way is that after installation the directory contains more files rather than a single executable (like common software applications). Despite that, we believe it is worth it for the user if it eliminates any potential user errors in the setup.	

Another motivation for choosing this non-functional requirement is after the feedback of Sprint 2, the team received feedback of not being able to open the application on the user side. We hope through this non-functional requirement we solve the issue once and for all.

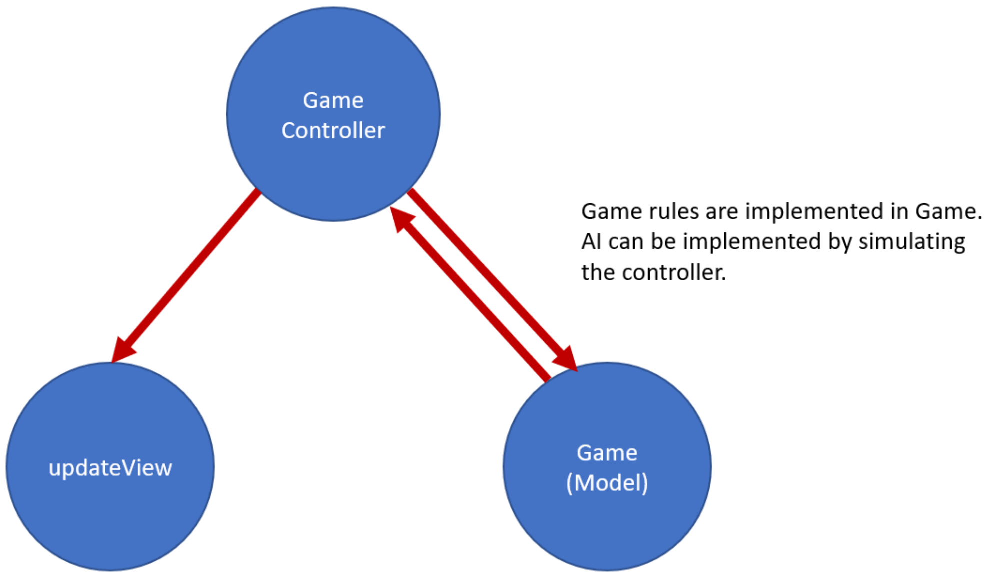


```
Microsoft Windows [Version 10.0.22621.1702]
(c) Microsoft Corporation. All rights reserved.

C:\Users\acer>java -version
'java' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\acer>
```

The game runs even without Java installed despite being a Java app. Tested on Windows 11.

Embedding Human Values - Intelligent

Schwartz theory explains the basic values that people in all cultures can recognize. To embed one human value into 9MM, the theme our team decided to go for is the “Intelligent” value under the Achievement theory. While this is part of the advanced requirements, our game demonstrates intelligent values by supporting playing against an AI in the future. The model chosen for the game, the MVC architecture supports the game model being agnostic to the input source, which provides flexibility for the team to introduce an input generator. Ideally, this generator should be able to examine the Board state and identify the next best move that challenges the player to think the most.


Game Controller
updateView
Game (Model)
Game rules are implemented in Game.
AI can be implemented by simulating the controller.

The MVC architecture can simulate inputs in the controller for the model.

By introducing an AI, we showcase a human value that may be appreciated by playing the optimal move and tests the human player’s analytical skills. In terms of Board games, they are no stranger to AI since they are always subject to mathematical study to examine the best move to obtain a winning strategy. This is inspired by Chess and Go AIs where they played unconventional lines that surprised their human counterparts. As mentioned in the Week 8 workshops, no human embedded values are useful if it cannot be tested. One way to ensure the AI we created is valuable is that we can make 2 variations of the AI - one that generates a random move and another that is based on heuristics, where the AI that makes the educated guesses should win most of the time.