FIT3077: Software engineering: Architecture and design

S1 2023

Monash University Malaysia

Sprint Three

Nine Men's Morris

Team The Three Tokens:

Priyesh Nilash Patel 32182058

Rachel Ng Chew Ern 31424290

Hee Zhan Zhynn 31989403

Sprint 3: Design Rationale

***Note**: To view the GIFs attached as figures illustrating our game, please refer to the google docs via link. It should be noted that the UML diagram is present in the PDF but not accessible through the link.*

Sprint 3 Demo: https://youtu.be/z88bg2aQVc4

Architecture

As outlined in our UML class diagram, we have added a SceneController Class to our code architecture. This class is necessary for us to handle switching between different scenes such as the main menu, game scene and the rules page. While we do have an existing controller class in RootLayoutController, we want it to exclusively be in charge of managing the game board so that it would not become a god class.

Therefore, we decided to add a SceneController class to manage the switching of scenes, such as from the main menu to the Rules page or the Game page. This is reflective of the **Single Responsibility Principle (SRP)**, as any new gameplay-related functionality will be handled by RootLayoutController while SceneController will be responsible for any new plugins or additional pages that might be added to the game, such as a leaderboard screen. Thus, by introducing the SceneController class, there is a higher level of code readability and maintainability that will greatly benefit anyone who works on the game in the future.

While SceneController was the only major modification we made, there were a number of smaller changes such as adding more attributes and methods to existing classes to help us flesh out our game and fulfil the necessary requirements. Some examples include isMill(), setMill() and updateMillStatus() in the GameManager class which handles logic relating to forming mills; as well as handleGameover(), handleNewGame(), handleClose(), handleMenu() and handleMusic() in RootLayoutController which allows the user to navigate or configure the app.

Quality

After attending the Week 8 workshop, we learned about the different quality attributes that are desirable for a software to have. We considered our 9MM implementation and came up with the following non-functional requirements that demonstrates our game's quality:

1. **Usability**
    - We consider usability to be a crucial quality aspect as our main goal in this project is to provide our users with a seamless gaming experience.
    - As our game employs the drag-and-drop mechanic for players to move their tokens, we believe that this is strongly symbolic and indicative of a Nine Men's Morris game that is played on a physical board.
    - Tokens will snap into place when the player moves it to an approximate location close to a valid board position.
    - Interface is clean, user action is snappy and fluid.

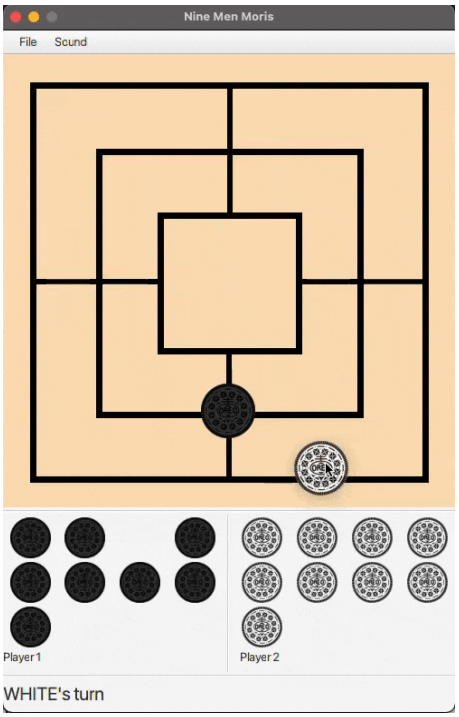


Figure 1: Token placement (gif in google docs)

2. **Communicative**
    - There are multiple measures in the game that aim to keep the user informed of what is going on, as well as to let them know the consequences of certain terminating actions.
    - We added a game status bar at the bottom of the game which indicates the current game status to the player, such as which player's turn it currently is.
    - If the player wishes to take any action that destroys the current state of the game board such as exiting the main menu or starting a new game, a dialog window will appear to prompt the user to confirm their choice.
    - It is important to provide clear communication to our users so that they have all the information they need to make the best decision.

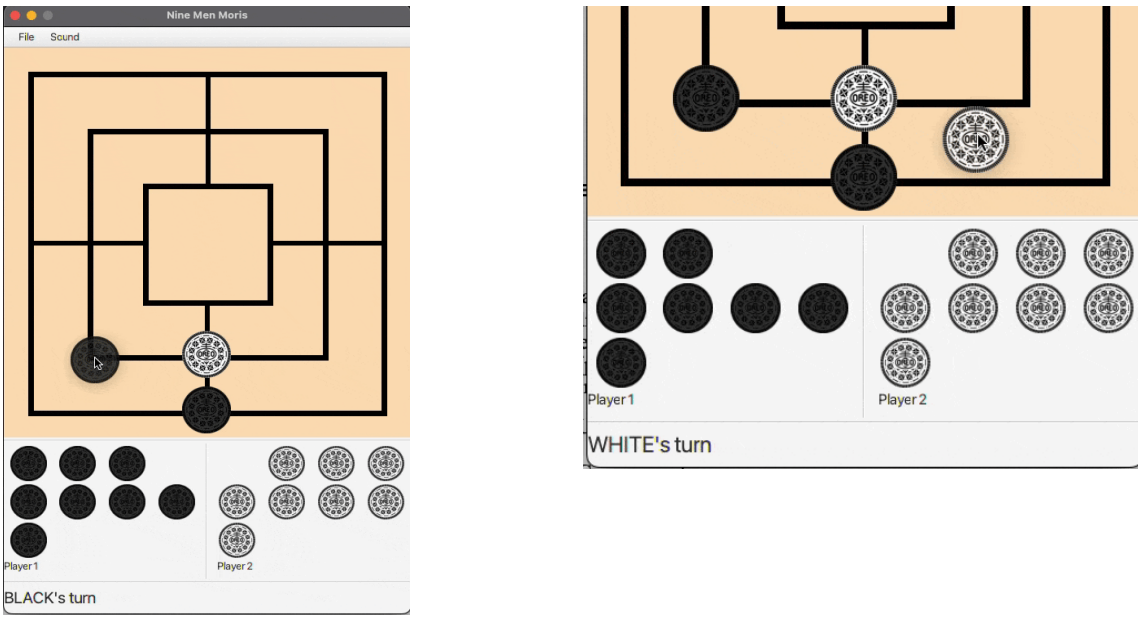


Figure 2: Preview of game status bar at the bottom

3. **Efficiency**
    - The game boots up almost immediately after running the executable file.
    - Game actions are instant, the program executes game rules without freezing or lagging.
    - We must not neglect software efficiency as it is a major contributor to user experience and usability in general.

Human Values from Schwartz's Theory

Considering that all of our actions are driven by human values, Schwartz's Theory provides us a critical framework to evaluate the contribution that our game will offer to society. After an internal discussion, our team has come to the conclusion that our game represents pleasure and creativity as human values.

1. **Pleasure**
    - When the game is booted up, a chill lofi beats music will start playing in the background. This lets the players enjoy the vibe as it will foster a soothing atmosphere, allowing them to avoid boredom when waiting for their opponent to make their move.
    - We believe that music is an element that unites people as almost everyone derives pleasure from listening to music.
    - There is also an option to mute the music if players would prefer to focus on the game without any distractions.
    - Ultimately, this is a feature that we added for the pleasure of our users.

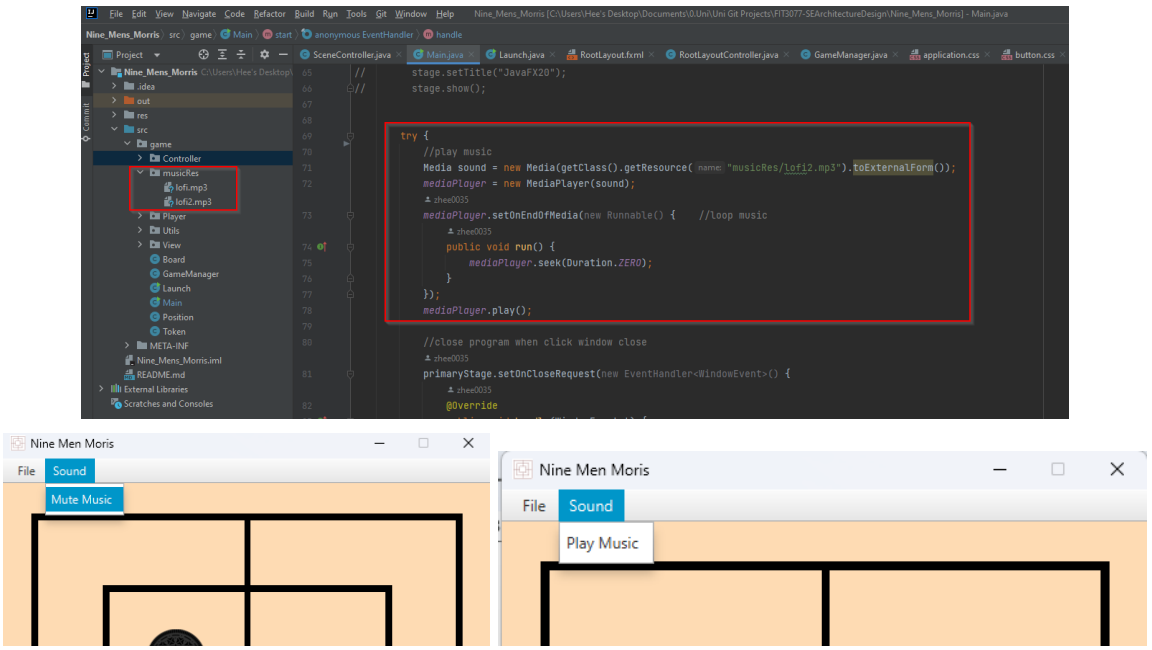


Figure 3: Controls for music

2. Creativity

- We introduced two colours of Oreo inspired tokens instead of using basic black and white tokens to add creativity and our own special flair on a 9MM game.
- Our team also put a lot of thought into the main menu design that depicts what the game is about and to attract users' attention. It displays a sense of rivalry as two players will take command of their tokens in a battle of wits, yet the casual drawing style also conveys a light hearted tone that reminds people of a friendly competition.

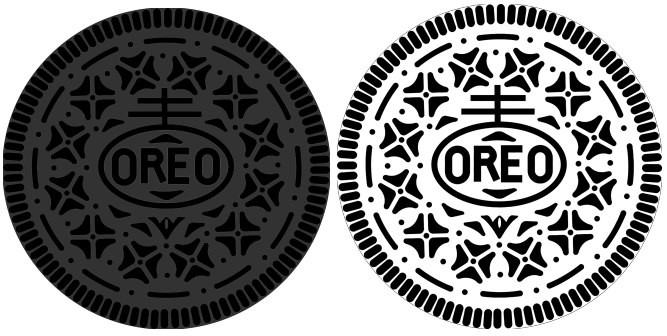


Figure 4: Tokens image

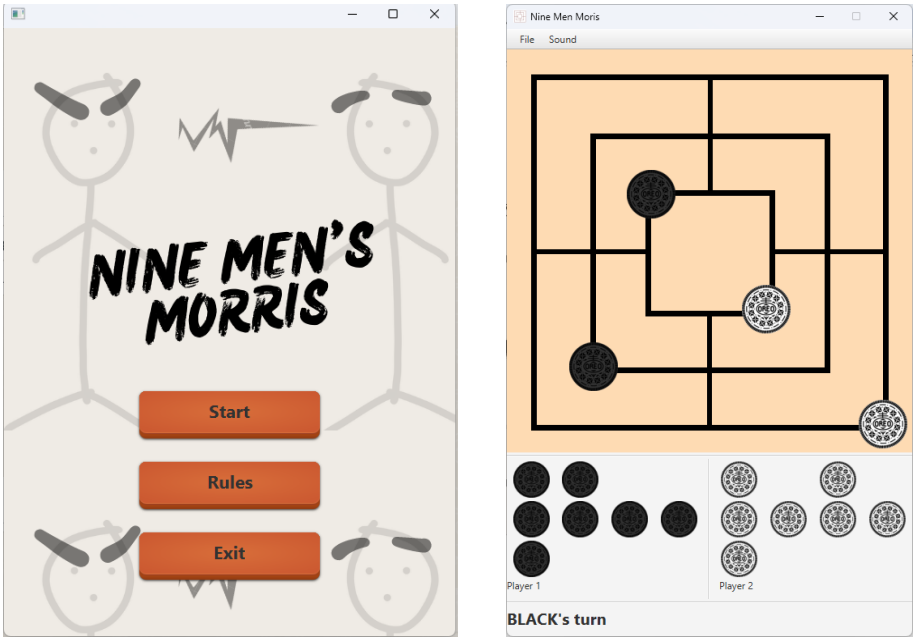
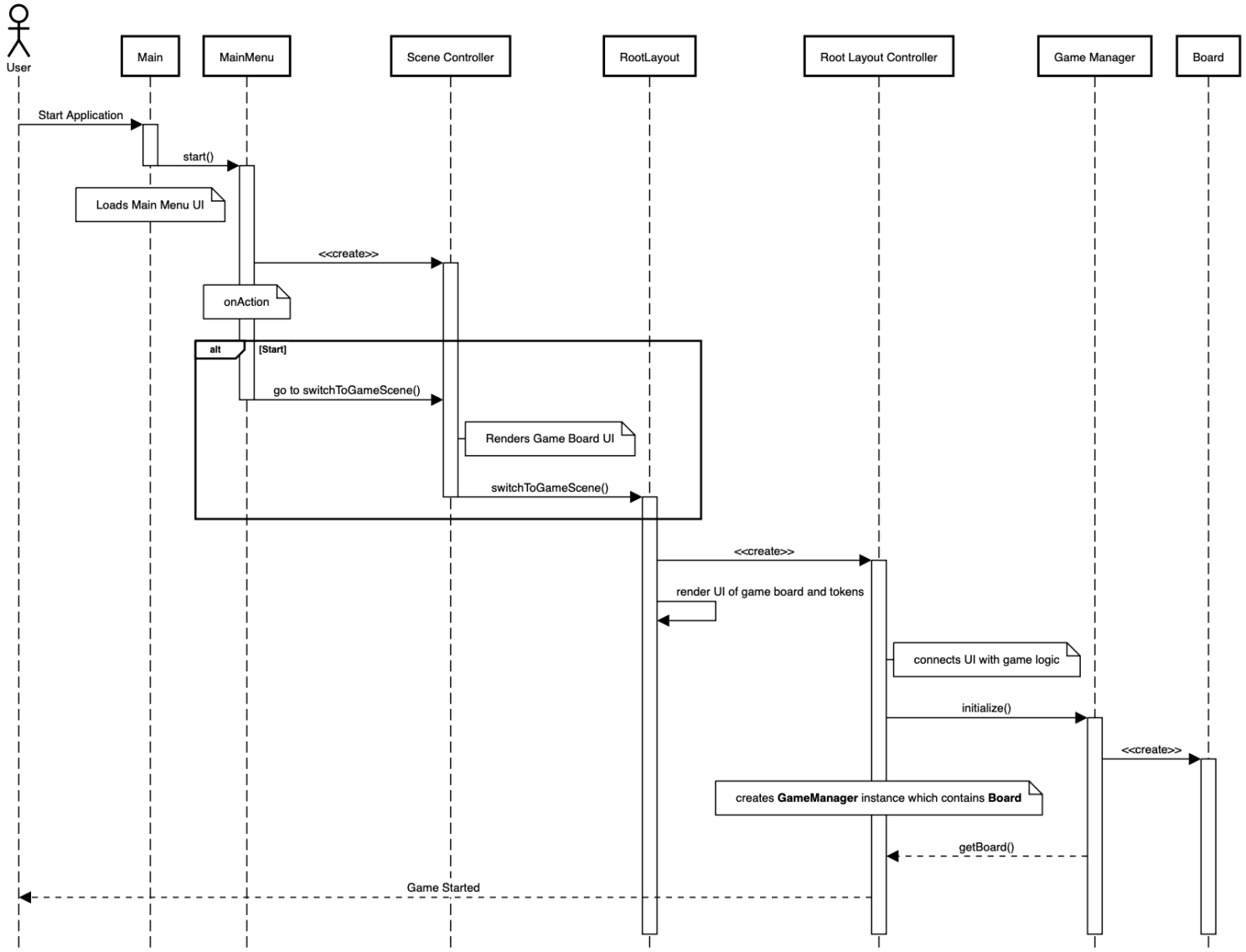


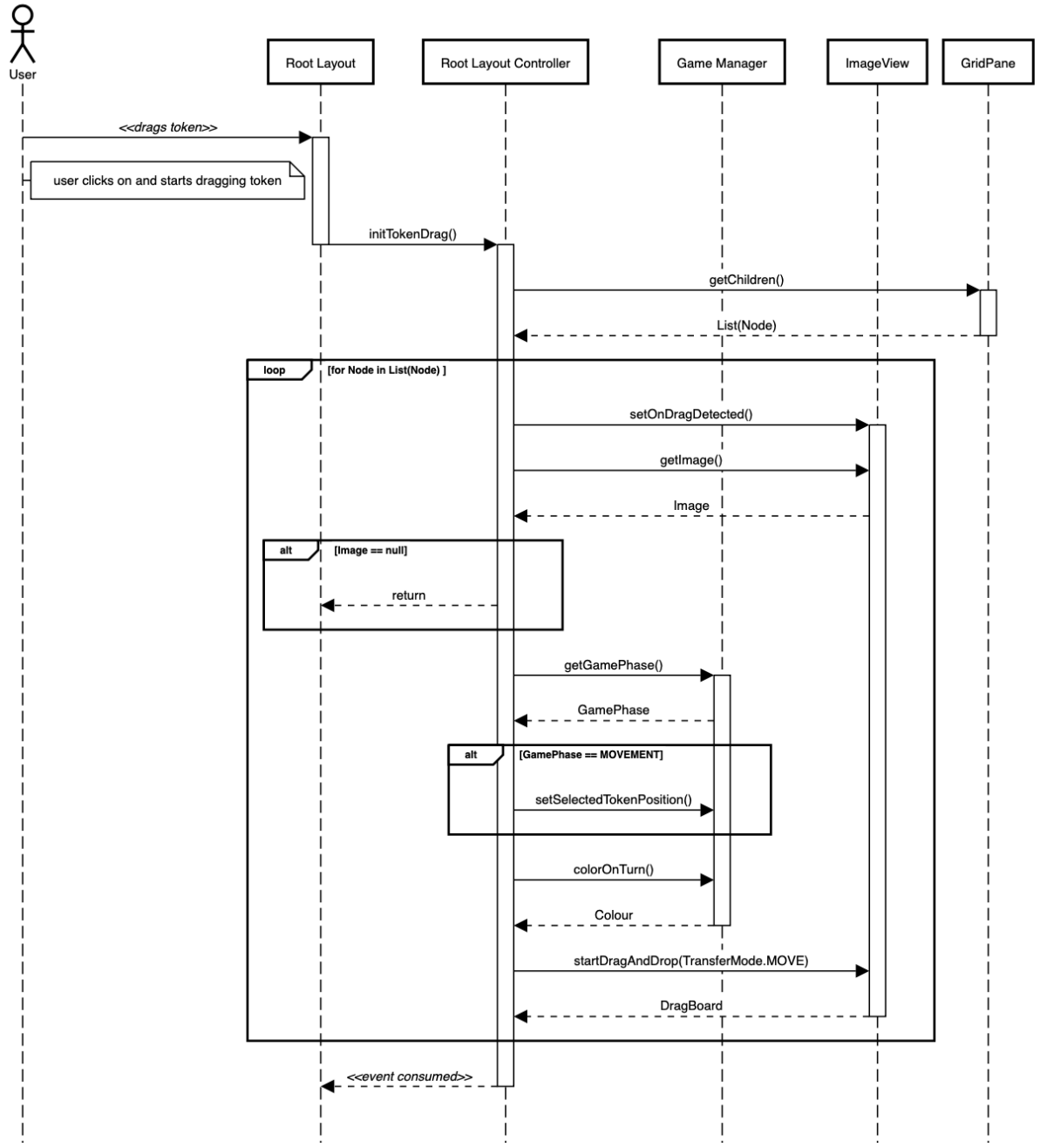
Figure 5: Game screenshot

Sequence Diagrams below.

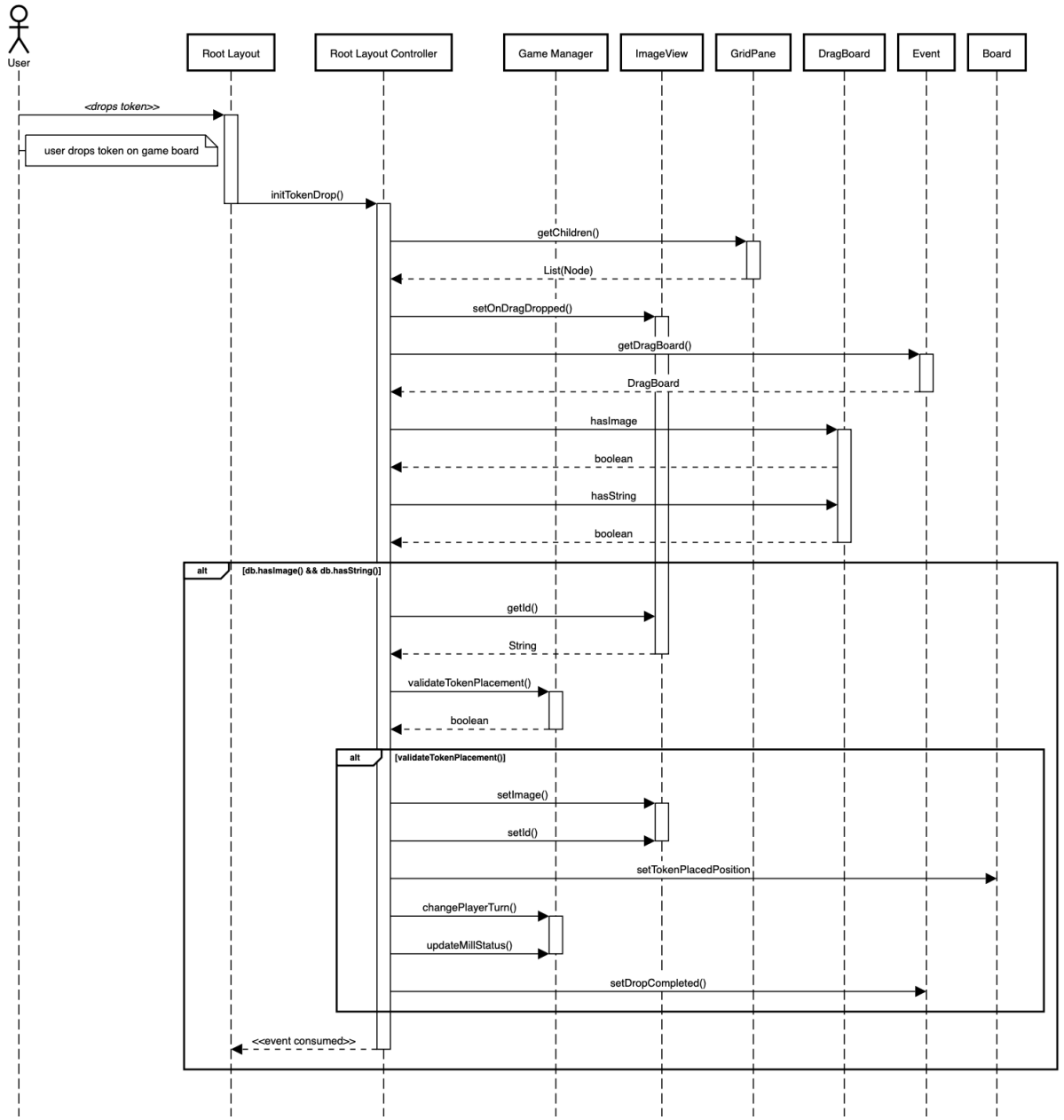
9MM Initialisation Process



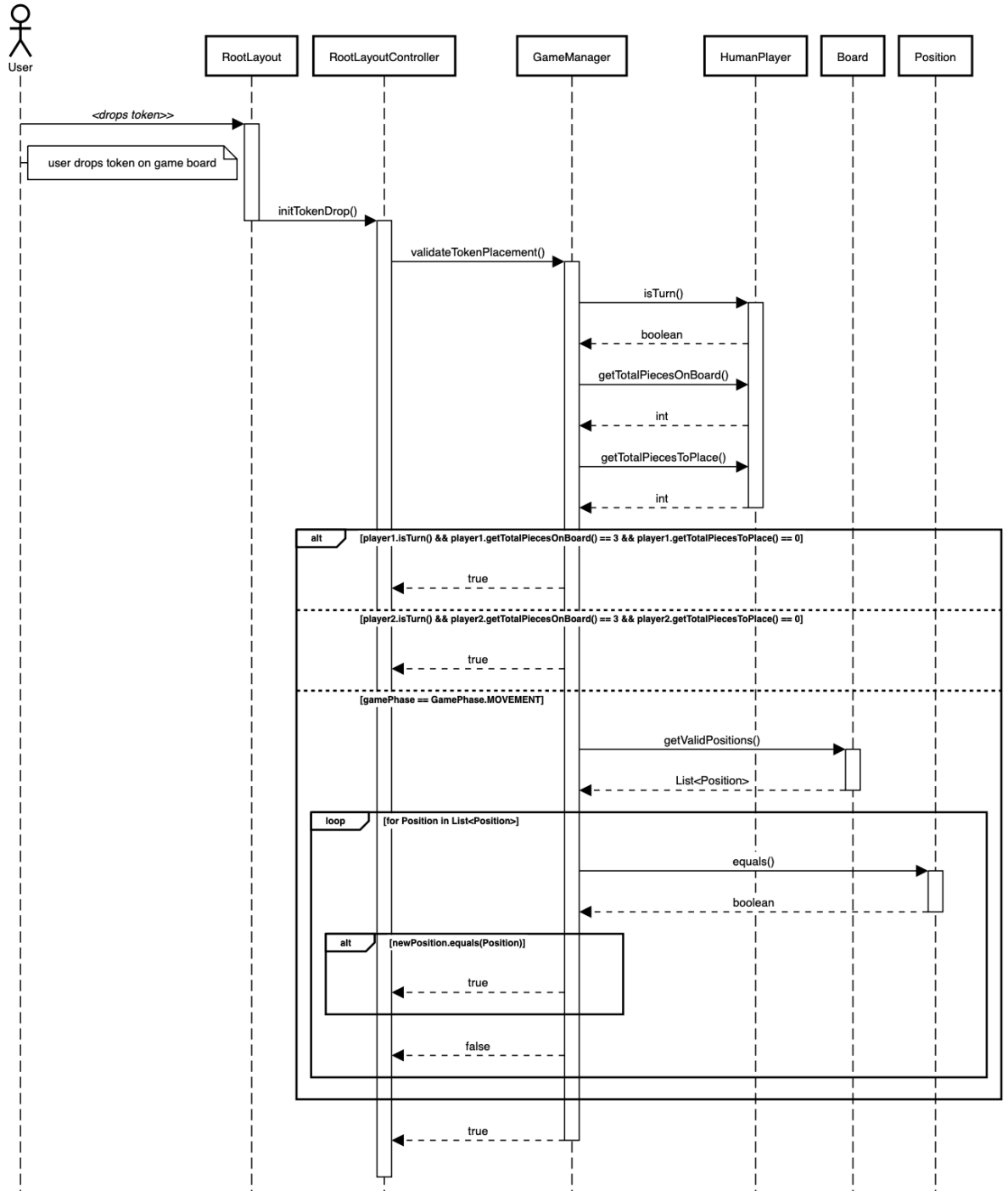
9MM Dragging Token (OnTokenDrag)



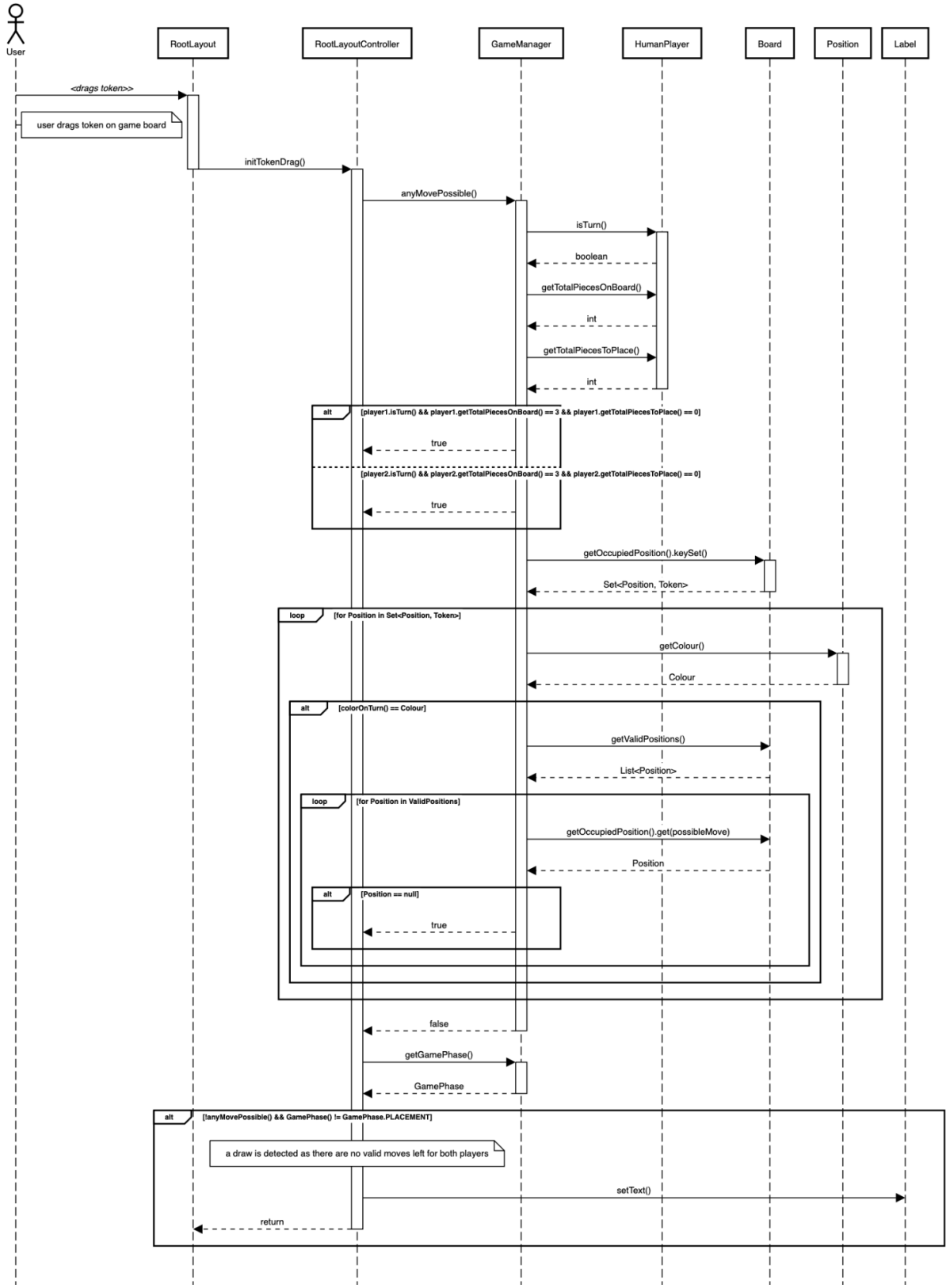
9MM Dropping Token (OnTokenDrop)



9MM Validating Token Position



9MM Checking For Draw



UML Diagram

Note: UML Diagram is in the uploaded PDF version. We are unable to attach the UML pdf in Google Docs.

game

Player

HumanPlayer

- playerName:String

+ getPlayerName(): String
+ setPlayerName(String): void
+ activateTurn(): void

AIPlayer

+ activateTurn(): void

Attributes

Abstract Player

- totalPiecesToPlace: int
- totalPiecesOnBoard: int
- colour: Colour
- isTurn: boolean

+ isTurn(): boolean
+ setTurn(boolean turn): void
+ getTotalPiecesOnBoard(): int
+ setTotalPiecesOnBoard(int): void
+ getTotalPiecesToPlace(): int
+ setTotalPiecesToPlace(int): void
+ getColour(): Colour
+ setColour(Colour): void
+ removeToken(): void
+ tilePlaced(): void
+ activateTurn(): void
+ deactivateTurn(): void

1 represented by 1

Utils

«enumeration»
Colour

WHITE
BLACK

Attributes

«enumeration»
GamePhase

PLACEMENT
MOVEMENT
GAMEOVER

2 has 1

Main

~~- controller: RootLayoutController~~
- GAME_NAME: String
- GAME_VERSION: String
- FULL_NAME: String

+ start(Stage): void
+ main(String{}): void

1 initialises 1

GameManager

- player1: Player
- player2: Player
- gamePhase: GamePhase
- board: Board
- totalTokenPlaced: int
- MAXTOKEN: int
- isMill: boolean

+ getBoard(): Board
+ setBoard(Board): void
+ getPlayer1(): Player
+ setPlayer1(Player): void
+ getPlayer2(): Player
+ setPlayer2(Player): void
+ isMill():boolean
+ setMill(boolean Mill):void
+ startGame(): void
+ getGamePhase(): GamePhase
+ getTotalTokenPlaced(): int
+ setTotalTokenPlaced(int): void
+ changePlayerTurn(): void
+ colorOnTurn(): Colour
+ placeToken(Position): void
+ moveToken(Position): void
+ setSelectedTokenPosition(Position): void
+ validateTokenPlacement(Position): boolean
+ checkWin(): int
+ updateMillStatus(Position tokenPosition): void
+ removeToken(Position tokenPosition): boolean
+ anyMovePossible(): boolean

1 manages 1

1 has 1

Controller

1

SceneController

- stage: Stage
- scene: Scene
- root: Parent
- controller: RootLayoutController

+ switchToMainMenuScene(Stage): void
+ switchToRuleScene(ActionEvent): void
+ rulesToMainMenu(ActionEvent): void
+ switchToGameScene(ActionEvent): void
+ closeGame(): void

switches to 1 1

RootLayoutController

- leftPocketGrid: GridPane
- rightPocketGrid: GridPane
- gameBoardGrid: GridPane
- stage: Stage
- board: Board
- boardGridChildren: List<ImageView>
- gameManager: GameManager
- playerTurnLabel: Label
- musicLabel: MenuItem
- sceneController: SceneController

- getTilePosition(ImageView): Position
+ setStage(Stage): void
+ setGameManager(GameManager): void
- initTokenDrag(GridPane): void
- initTokenDrop(GridPane): void
- initGameManagerPropertyListeners(): void
- initialize(): void
- removeTileMill(): void
- initWindow(): void
- gameDialog(String, String, String, int):void
+ handleGameover(): void
+ handleNewGame(): void
+ handleClose(): void
+ handleMenu(): void
+ handleMusic(): void

1 communicates with 1

1 displays 1

Attribu

<<uses>>

Position

- x: int
- y: int

+ getX(): int
+ setX(int): void
+ getY(): int
+ setY(int): void
+ toString(): String
+ equals(Object): boolean
+ hashCode(): int
+ getAllPositions(): Map<Position, Integer>

24 has 1

Board

- boardPositions: Map<Position, Integer>
- occupiedPosition: Map<Position, Token>
- oldPosition: Position
- tokenPlacedPosition: ObjectProperty<Position>
- millSets: Map<Integer, List<Token>>
- millId: Int

+ getOldPosition(): Position
+ setOldPosition(Position): void
+ getBoardPositions(): Map<Position, Integer>
+ setBoardPositions(Map<Position, Integer>): void
+ getOccupiedPosition(): Map<Position, Token>
+ setOccupiedPosition(Map<Position, Token>): void
+ getTokenPlacedPosition(): Position
+ setTokenPlacedPosition(Position): void
+ increaseMillId():void
+ ~~validateTokenPlacement(Position newPosition) : boolean~~
+ getValidPositions(Position position): List<Position>
+ placeNewToken(Position, Colour): void
+ moveToken(Position newPosition): void
+ getNeigbours(Position newPosition): List<Position>
+checkIfMill(Position newPosition): boolean
+ canBeRemoved(Position tokenPosition): boolean
+removeToken(Position tokenPosition): boolean
+reduceIsMillCount(Token token): void
+getKeyByValue(Map<T, E> map, E value): T

0..1 occupy 1

Token

- colour: Colour
- position: Position
- isPartOfMillCount: int
- millId: int[]

+ getColour(): Colour
+ setColour(Colour): void
+ getPosition(): Position
+ setPosition(Position): void
+ getIsPartOfMillCount: int
+ setIsPartOfMillCount:void
+ increaseIsPartOfMillCount:void
+ decreaseIsPartOfMillCount:void
+ getMillId: int[]
+ setMillId: void
+ updateMillId(int id) :void
+ resetMillId(int id): void

0..18 supports 1

2..9 has