FIT3077: Software engineering: Architecture and design

S1 2023

Monash University Malaysia

Sprint Four

Nine Men's Morris

Team The Three Tokens:

Priyesh Nilash Patel 32182058

Rachel Ng Chew Ern 31424290

Hee Zhan Zhynn 31989403

Sprint 4: Design Rationale

Selected Advanced Requirement: Implement new game mode to play against computer

Sprint 4 Demo: https://youtu.be/6x5lyS0EDVE

Updates to User Stories

Original User Stories in Sprint One:

1. As a player, I want to start a new game so that I can play the game.
2. As a player, I want to be able to see the full board, so that I can see all tokens
3. As a player, I want to be able to see the total tokens that are alive so that I can strategize my game.
4. As a player, I want to read the rules of the game so that I understand how the game works.
5. As a player, I want to place tokens on the board to create a mill.
6. As a player, I want to be able to play the game with another player so that I can improve my skills.
7. As a player, I want to be able to move my pieces on the board to create a mill or block my opponents.
8. As a player, I want to capture my opponent's tokens so that I can increase my chances of winning.
9. As a player, I want contrasting token colours between me and my opponent so that I can easily differentiate the tokens.
10. As a player, I want to have a bug-free experience so that I can enjoy the game.
11. As a player, I want to clearly see the token that I have selected so that I know which token I am moving.
12. As a player, I want to be able to play the game on the same device with my friend so that we can have fun together.
13. As a player, I want to know when the game is over so that I can see if I have won or lost.
14. As a player, I want to play the game without any internet connection so that I can play it whenever and wherever I want.
15. As a player, I want to quit the game so that I can close it once it is over.
16. As a game board, I want to ensure all moves made are legal so that the game can be played fairly
17. As a game board, I want to ensure that the game rules are enforced so that no player can cheat.
18. As a game board, I want to be sure all positions are distinct so that I can differentiate between them.
19. As a token, I want to be part of the game so that the players can use me to play the game.
20. **As a computer player, I want to be sufficiently challenging so that the player can improve their skills.**
21. **As a computer player, I want to make decisions using a good heuristic function, so that I have a high chance of winning the game.**
22. **As a computer player, I want to be another option for a human player to play with so that they can still play without another human player.**

Updates:

20. [REVISED] As a human player, I want to play against an AI that simulates the capability of another human player, so that I can improve my strategy skills even without having another person to play with.
21. [REVISED] As a computer player, I want to make moves that are valid yet random, so that I can be an unpredictable opponent for the human player.
23. [NEW] As a player, I want to select my preferred game mode after clicking start so that I can immediately start playing against the opponent of my choice.

Design Rationales (explaining the Whys'):

1. Explain *why* you have designed the architecture for the advanced requirement(s) the way you have.

In our application, we have employed object-oriented programming principles to facilitate **AI movement** within the RootLayoutController class, instead of creating a separate AI player class.

Our overall architecture is structured around two main controllers: the SceneController class, responsible for managing the transition between game scenes such as the main menu, and the RootLayoutController, which handles various gameplay functionalities, including the drag and drop actions performed by human players. This design choice adheres to the **Single Responsibility Principle (SRP)**, as each controller is dedicated to a specific aspect of the application.

To incorporate AI functionality, we needed to simulate valid moves that a player could make. This allowed us to seamlessly integrate AI movement within the RootLayoutController, treating it as if it were executed by a human player. Since the RootLayoutController already handles and processes user input, such as the drag and drop actions, we found it logical to extend its capabilities to include the programmatically generated input required for AI movement. Consequently, the dragging and dropping of tokens are now handled by a single class, aligning with the principles of SRP.

We deliberately chose not to create a separate AI player class, primarily due to the potential drawbacks it may introduce. Implementing an AI player class would introduce additional dependencies between classes and could complicate the maintenance of our codebase. By keeping the AI functionality within the RootLayoutController, we maintain a streamlined and cohesive structure, making it easier to manage and modify the code in the future.

2. Explain *why* you have revised the architecture if you have revised it.

As we initially intended to implement AI functionality in its own Player class, our early designs included a standalone AIPlayer class, which is a sibling to the HumanPlayer class. However, as we moved into the process of implementing the advanced requirement, we found out that it would make more sense to contain AI-related functions in RootLayoutController to **increase cohesion** while **reducing coupling**. This is because RootLayoutController receives and coordinates the input used to manipulate pieces on the game board, which means that it should be responsible for simulating AI movements as well. Therefore, seeing that there wasn't a real need for it, we removed the AIPlayer class from our architecture.

In addition to that, we also added a GameMode enum to indicate the mode that the player has selected to play in. This is mainly used to initialise the appropriate listeners and to inform the RootLayoutController to handle scenarios involving an AI player if this game mode is selected. As the game mode will remain constant once it is selected, we decided that it will be best represented as an enum.

3. **Explain when your advanced feature was finalised (e.g. it is the same as we decided from sprint one; or we changed it in Sprint 3) and how easy/difficult it was to implement. e.g. was it easy to implement due to good design practice/pattern(s) that you have applied in the earlier Sprints (provide evidence)? Or was it difficult (such that you needed to rewrite the majority of the code) for the advanced feature?**

*Original: As a computer player, I want to make decisions using a **good heuristic function**, so that I have a high chance of winning the game.*

The advanced feature of implementing a good heuristic function for the computer player was not finalised as originally planned. Due to time constraints, we were unable to fully develop the intended heuristic function, which is supposed to provide the player with a challenging experience. However, we did incorporate a basic functionality where the AI player randomly selects a move from the currently available valid moves.

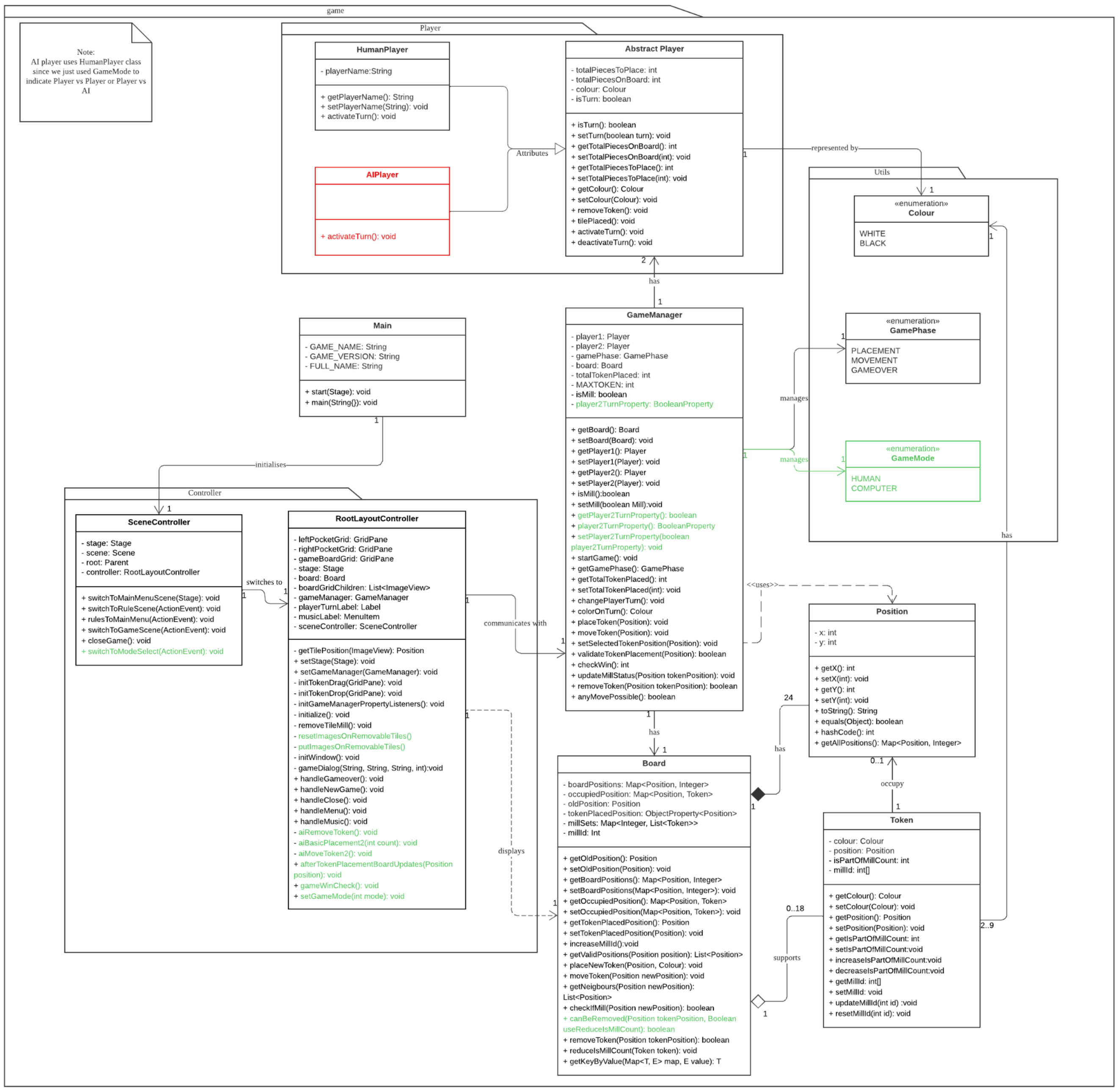
The implementation of this advanced feature was relatively straightforward due to the good design practices and patterns we had applied from Sprint 1, particularly the adherence to the Single Responsibility Principle (SRP). For earlier Sprints, we implemented two controller classes (SceneController and RootLayoutController) to handle the front-end and back-end respectively. Since the RootLayoutController class already handled the drag-and-drop actions for human players, adding the AI logic to randomly select a move among the valid options was a natural extension of its existing functionality.

Fortunately, we did not encounter major difficulties during the implementation of this advanced feature. By leveraging the modular and well-organised structure of our codebase, we could incorporate the necessary changes without the need for extensive code rewriting. The only significant addition was the introduction of a new *GAMEMODE* to indicate the user's preference to play against an AI player.

In summary, while the original plan to implement a sophisticated heuristic function was not realised due to time limitations, we successfully integrated a basic random move selection for the computer player. The implementation process was facilitated by effective design practices, particularly the adherence to SRP, which minimised the need for extensive code modifications.

Final UML Diagram

Note: *Please access the full-resolution UML diagram in the Sprint 4 folder in our GitLab*



game
Note: AI player uses HumanPlayer class since we just used GameMode to indicate Player vs Player or Player vs AI
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
represented by
1
1
2
has
1
Utils
«enumeration»
Colour
WHITE
BLACK
1
«enumeration»
GamePhase
PLACEMENT
MOVEMENT
GAMEOVER
1
manages
«enumeration»
GameMode
HUMAN
COMPUTER
1
manages
1
has
Main
- 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
- player2TurnProperty: BooleanProperty
+ getBoard(): Board
+ setBoard(Board): void
+ getPlayer1(): Player
+ setPlayer1(Player): void
+ getPlayer2(): Player
+ setPlayer2(Player): void
+ isMill():boolean
+ setMill(boolean Mill):void
+ getPlayer2TurnProperty(): boolean
+ player2TurnProperty(): BooleanProperty
+ setPlayer2TurnProperty(boolean player2TurnProperty): 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
<<uses>>
Controller
SceneController
- stage: Stage
- scene: Scene
- root: Parent
- controller: RootLayoutController
+ switchToMainMenuScene(Stage): void
+ switchToRuleScene(ActionEvent): void
+ rulesToMainMenu(ActionEvent): void
+ switchToGameScene(ActionEvent): void
+ closeGame(): void
+ switchToModeSelect(ActionEvent): 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
- resetImagesOnRemovableTiles()
- putImagesOnRemovableTiles()
- initWindow(): void
- gameDialog(String, String, String, int):void
+ handleGameover(): void
+ handleNewGame(): void
+ handleClose(): void
+ handleMenu(): void
+ handleMusic(): void
- aiRemoveToken(): void
- aiBasicPlacement2(int count): void
- aiMoveToken2(): void
+ afterTokenPlacementBoardUpdates(Position position): void
+ gameWinCheck(): void
+ setGameMode(int mode): void
1
communicates with
1
1
displays
1
1
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
+ 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 useReduceIsMillCount): boolean
+ removeToken(Position tokenPosition): boolean
+ reduceIsMillCount(Token token): void
+ getKeyByValue(Map<T, E> map, E value): T
1
has
24
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>
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