SPRINT THREE

FIT3077 Software Engineering:
Architecture & Design

Optimitics

Version 3

19/05/2023

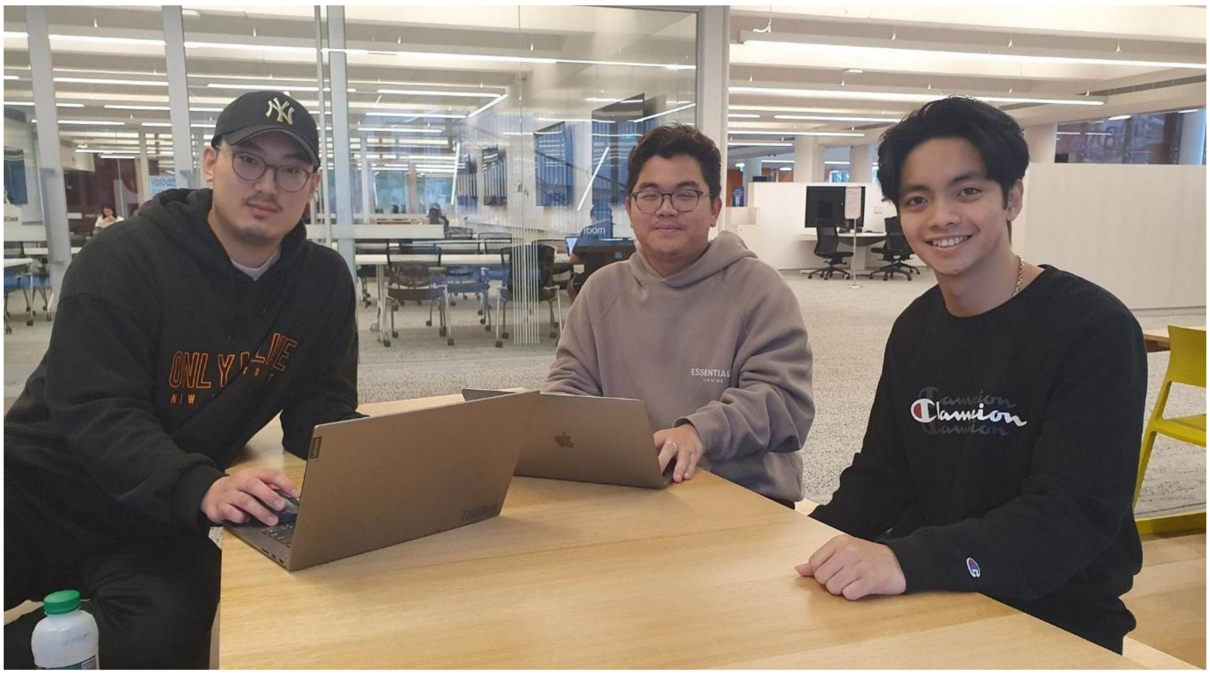
Chandra Putra – 30709458
Deokkyun Kil – 32053819
Hong Kien Nguyen – 29853168

Table of Contents

Team Information

Team name & Team Photo

Optimistics



Team member

Name	Email Address	Technical Strength	Professional Skill	Fun Fact
Deokkyun Kil	dkil0003@student.monash.edu	Java, Python, C++, CakePHP	Teamwork, Creative and strategic thinking	Korean, Working in Korean restaurant
Chandra Putra	cput0004@student.monash.edu	Java, HTML, CakePHP	Cooperation & Collaboration, Real-life problem solving	Listens to Lana Del Rey religiously
Hong Kien Nguyen	hngu0053@student.monash.edu	Java, C++, CakePHP, Html, Javascript	Communication, Collaboration, Time Management	Can not handle spicy food

Team Schedule

Regular meeting schedule

Online Meeting (Zoom)	9.00pm Tuesdays https://monash.zoom.us/j/9556780254?pwd=dUt1NnAzUWRkOFdrb1VsbmNLd3NTdz09 meeting ID: 955 678 0254 passcode: 451882
Face-to-face Meeting (On campus)	6.00pm Thursday

Regular work schedule

Chandra	Tuesday: 2-7pm Wednesday: 11-7pm Friday: 3-9pm Saturday: 2-7pm P.S. Time is relative to shifts given but usually work on the days listed.
Deokkyun	Monday: 10-10pm Tuesday: 10-8pm Wednesday: 4-7pm Thursday: 10-8pm Saturday or Sunday: 1-11pm (Decided on the previous Sunday)
Hong Kien	Monday: 12pm-8pm Tuesday: 12pm-4pm Wednesday: 12pm-6pm Thursday: 12pm-6pm

Workload Distribution

Distribution of workload will be distributed equally to each team member where there is a mutual agreement within all team members during team meetings or discussions with a condition where all team members have to contribute or give an input to every aspect of the development or project.

Furthermore, in special circumstances only one team member is in charge of certain tasks. Tasks are listed below which are subjects to changes regarding the situation in the future.

1. Documents/reports formatting
2. Generating meeting link
3. Ensuring no merge conflict
4. Reminding each other regarding development
5. Generating initial project/file to work on

Technology Stack & Justification

Technology	Justification / Alternatives
Java (ORACLE) & IntelliJ IDEA	Justification: - The main technology for developing the Application. - Since Java is an object-oriented language, it allows us to create modular programs and reusable code. - One of the advantages of Java is its ability to move easily from one computer system to another, which is platform independent. It is suitable for developing a standalone application
	Alternatives: C++, C - Steep learning curve since not all team members familiar with the language
GitLab	Justification: - Core part of co-developing a project where developers could keep track of who contributed on which part. - The main technology for Version control - Act as a backup where history of development is shown
	Alternatives: GitHub - Essentially provide the same tools but extra account needs to be created given us students already been provided with one for GitLab - Need extra tutor's approval - Not within Monash server
Lucid Chart	Justification: - A comprehensive tool that provides collaboration in creating any kind of diagram - All the team members already familiar with it in previous units
	Alternatives: Diagrams.net - Essential features are covered but collaboration is not as simple as lucid chart has to offer
Figma	Justification: - All the team members already familiar with it in previous units - Offers extensive prototyping tools - Good starting point for low-fidelity prototype which can extended into high-fidelity if needed in a single place
	Alternatives: Balsamiq - Limited uses with free account

	- Does not offer student account
Facebook Messenger	Justification: - The main technology for Regular communication - Team members can connect socially - Simple messaging app to keep contact with each other
	Alternatives: WhatsApp, Instagram - Need to create another account to use - Unnecessary extra features - Instagram main feature is not messaging or texting
Zoom	Justification: - Connect team members regardless of space which is important for us busy/active students with tight work schedule - Provide all necessary online meeting tools with simplicity at its core - Convenient to send invitation or make recurring meeting - Less distractions
	Alternatives: Discord - A bit of learning curve - More distractions with another servers present - Screen sharing is limited to free user
Google Drive	Justification: - Offers quick and efficient web-based paperwork tools - File-sharing with minimal effort
	Alternatives: OneDrive - Web-based tools are not as efficient as Google Drive - Extensive tools yet complicated to use if not familiar
MongoDB	Justification: - Has a separate application that act as GUI which helps to view the database - NoSQL that gives a lot of flexibility in storing entry or document - Compatibility with object-oriented programming
	Alternatives: MySQL, Oracle SQL - Relational database seem a bit restrictive in this case where development are not set in stone where how we develop could change depending on how each sprint goes - Relational database requires more time to adjust to changes due to nature having to set attributes early in the stage

API	Swing (+JButton) - offers a suite of tools for creating graphical user interfaces (GUIs), as well as for enhancing Java's rich graphics functionality and interactivity.

User Stories

Basic Requirements

Setup:

As a player,

I want the game to have clear and concise instructions

so that I can easily understand the game mechanics.

As a player,

I want to see all placed tokens on the board

So that I know the status or state of the game.

As a game board,

I want to ensure that there are 2 players

So that the game can be started.

As a game board,

I want to show instruction to players,

So that players can play easily.

As a game board,

I want to check if player’s move follows the game rule

So that a fair game can be played.

As a player,

I want to be able to quit to the menu during the game from the initial setup phase

So that I can change settings or start over.

Gameplay:

First stage (placing a token)

As a white token player,

I want to place a token anywhere on the board at the beginning of the game

So that I can build my opening and start the game according to the rules.

Second stage (moving a token)

As a player,

I want to move my token adjacently to available position

So that I can move the pieces precisely according to the game rules.

As a player,

I want to be notified if I have formed a mill

So that I am given the option to remove an opponent's piece.

As a player,

I want to remove any available token (i.e., not in a mil) from the opponent when I place 3 tokens in row

So that I can reduce my opponent's available tokens to win the game.

Third stage (flying a token)

As a player,

I want to move my token anywhere on the board when I have only 3 tokens left

So that I can prevent my opponent from having 3 tokens in a row.

End of Game:

As a game board,

I want to check how many token each player has on the board

So that the game can be concluded once a player has less than 3 tokens.

Advanced Requirements

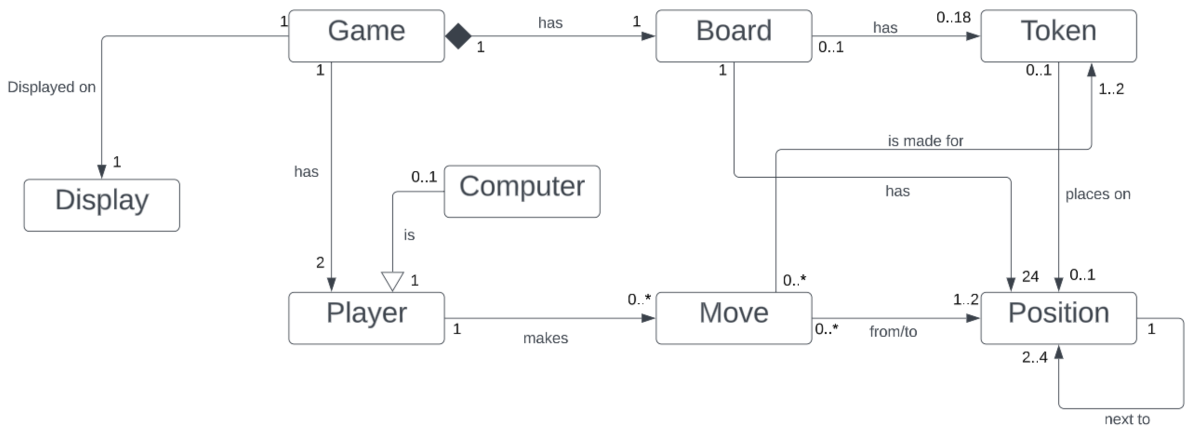
If ‘c’ is chosen (Our team choose this for now),

As a player,

I want to be able to choose a game mode,

So that I can play against another human or against the computer

Basic Architecture



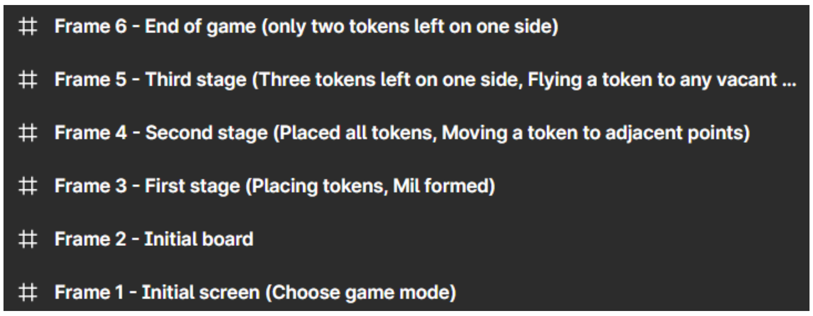
Advance requirement C is selected here as part of the model diagram.

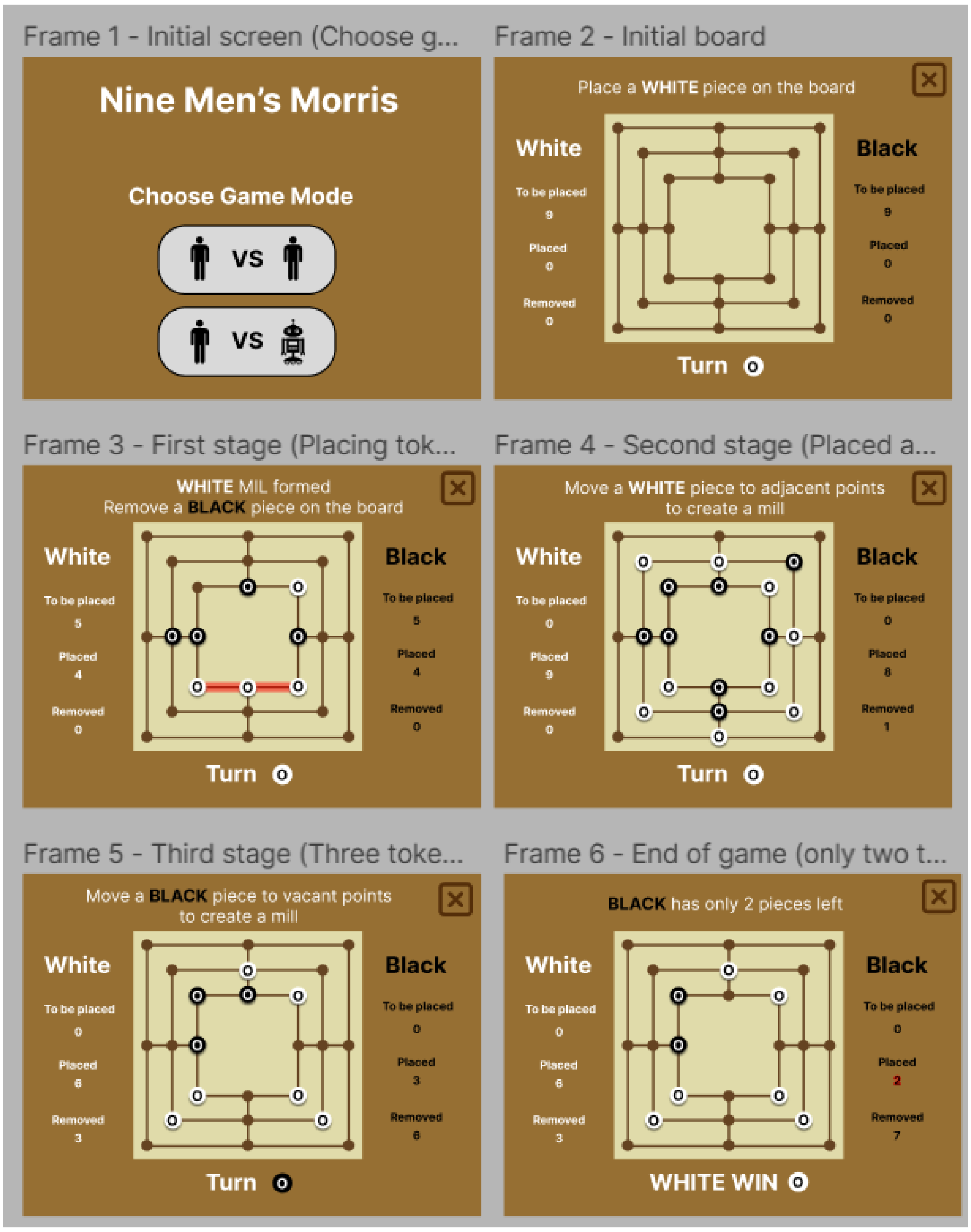
Relationship	Justification
Display & Game	<u>Notation:</u> The Game and Display are related but neither is a part of the other, so the notation is Association. <u>Multiplicity:</u> One game can be displayed on one to many displays. A display can display only one game.
Game & Player	<u>Notation:</u> The Game and Player are related but neither is a part of the other, so the notation is Association. <u>Multiplicity:</u> One game has two players. A player can only belong to one game.
Player & Computer	<u>Notation:</u> The Computer can act as a Player so it can inherit all the attributes from the Player entity, so the notation is Inheritance. <u>Multiplicity:</u> A player can be 0 to 1 computer. A computer can act as a player.
Player & Move	<u>Notation:</u>

	The Move and Player are related but neither is a part of the other, so the notation is Association. Multiplicity: A player can make 0 to many moves. A move can be made by only one player.
Computer & Move	Notation: The Move and Computer are related but neither is a part of the other, so the notation is Association. Multiplicity: A computer can make 0 to many moves. A move can be made by only one computer.
Game & Board	Notation: The Game is composed of one Board, and the Board can not exist without a Game, so the notation is Composition. Multiplicity: A game has only one board. A board can only belong to one game.
Board & Token	Notation: The Board and Token are related but neither is a part of the other, so the notation is Association. Multiplicity: A board can have 0 to 18 tokens. A token belongs to 0 to 1 board (placed or not).
Board & Position	Notation: The Board and Position are related but neither is a part of the other, so the notation is Association. Multiplicity: A board has 24 positions. A position belongs to only one board.
Move & Token	Notation: The Move and Token are related but neither is a part of the other, so the notation is Association. Multiplicity: A move can be made for 1 or 2 tokens (movement of a player's token and removal of another player's token). A token can have 0 to many moves.
Move & Position	Notation: The Move and Position are related but neither is a part of the other, so the notation is Association. Multiplicity: A move can be made that affects 1 or 2 positions. (movement of a player's token position and removal of another player's token position). A position has 0 to many moves made onto it.

Token & Position	<u>Notation:</u> The Token and Position are related but neither is a part of the other, so the notation is Association. <u>Multiplicity:</u> A token can be in 0 to 1 position, with 0 being out of board. A position allows 0 to 1 token to be on.
Position & Position	<u>Notation:</u> The Position and another Position are related but neither is a part of the other, so the notation is Association. <u>Multiplicity:</u> A position is next to 2 to 4 adjacent positions.

Basic UI Design

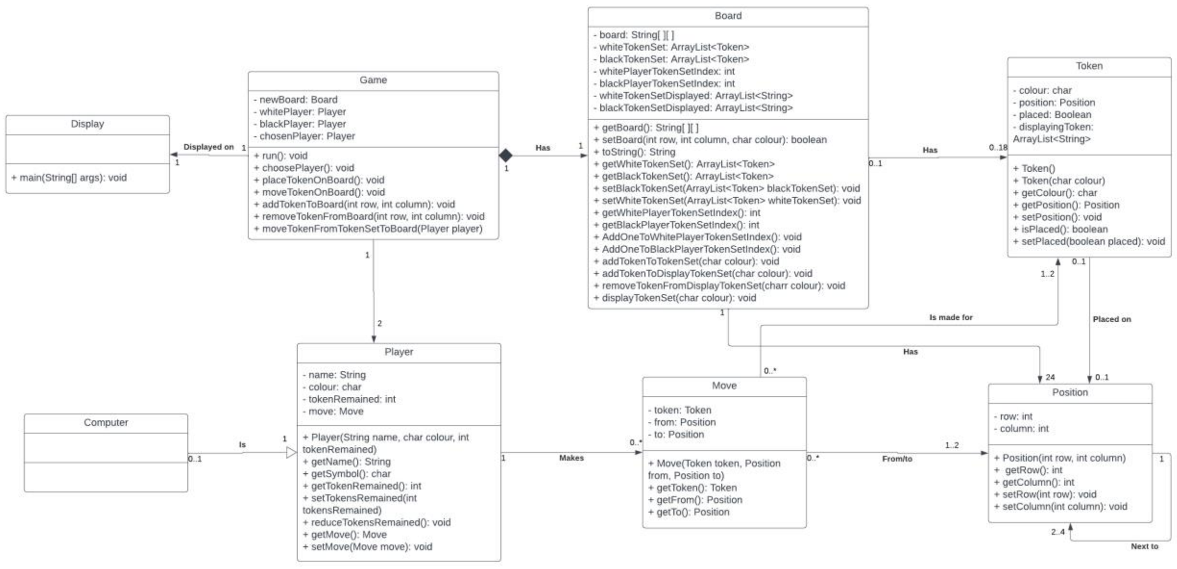

Frame 6 - End of game (only two tokens left on one side)
Frame 5 - Third stage (Three tokens left on one side, Flying a token to any vacant ...
Frame 4 - Second stage (Placed all tokens, Moving a token to adjacent points)
Frame 3 - First stage (Placing tokens, Mil formed)
Frame 2 - Initial board
Frame 1 - Initial screen (Choose game mode)



Attributes such as font, colour and size will be changed and improved as the project progresses.

SPRINT 2

Class Diagram


Display
+ main(String[] args): void
Displayed on
1
1
Game
- newBoard: Board
- whitePlayer: Player
- blackPlayer: Player
- chosenPlayer: Player
+ run(): void
+ choosePlayer(): void
+ placeTokenOnBoard(): void
+ moveTokenOnBoard(): void
+ addTokenToBoard(int row, int column): void
+ removeTokenFromBoard(int row, int column): void
+ moveTokenFromTokenSetToBoard(Player player)
Has
1
1
Board
- board: String[ ][ ]
- whiteTokenSet: ArrayList<Token>
- blackTokenSet: ArrayList<Token>
- whitePlayerTokenSetIndex: int
- blackPlayerTokenSetIndex: int
- whiteTokenSetDisplayed: ArrayList<String>
- blackTokenSetDisplayed: ArrayList<String>
+ getBoard(): String[ ][ ]
+ setBoard(int row, int column, char colour): boolean
+ toString(): String
+ getWhiteTokenSet(): ArrayList<Token>
+ getBlackTokenSet(): ArrayList<Token>
+ setBlackTokenSet(ArrayList<Token> blackTokenSet): void
+ setWhiteTokenSet(ArrayList<Token> whiteTokenSet): void
+ getWhitePlayerTokenSetIndex(): int
+ getBlackPlayerTokenSetIndex(): int
+ AddOneToWhitePlayerTokenSetIndex(): void
+ AddOneToBlackPlayerTokenSetIndex(): void
+ addTokenToTokenSet(char colour): void
+ addTokenToDisplayTokenSet(char colour): void
+ removeTokenFromDisplayTokenSet(charr colour): void
+ displayTokenSet(char colour): void
0..1
Has
0..18
Token
- colour: char
- position: Position
- placed: Boolean
- displayingToken: ArrayList<String>
+ Token()
+ Token(char colour)
+ getColour(): char
+ getPosition(): Position
+ setPosition(): void
+ isPlaced(): boolean
+ setPlaced(boolean placed): void
1..2
0..1
Is made for
Placed on
1
Has
24
0..1
1
2
Player
- name: String
- colour: char
- tokenRemained: int
- move: Move
+ Player(String name, char colour, int tokenRemained)
+ getName(): String
+ getSymbol(): char
+ getTokenRemained(): int
+ setTokensRemained(int tokensRemained)
+ reduceTokensRemained(): void
+ getMove(): Move
+ setMove(Move move): void
Computer
0..1
Is
1
1
Makes
0..*
0..*
Move
- token: Token
- from: Position
- to: Position
+ Move(Token token, Position from, Position to)
+ getToken(): Token
+ getFrom(): Position
+ getTo(): Position
0..*
From/to
1..2
Position
- row: int
- column: int
+ Position(int row, int column)
+ getRow(): int
+ getColumn(): int
+ setRow(int row): void
+ setColumn(int column): void
1
2..4
Next to

Design Rationale

○ Explain two key classes that you have debated as a team, e.g. why was it decided to be made into a class? Why was it not appropriate to be a method?

Issue: Should game become a class or a method?	
Alternative A: Game as a class **Advantages**: ● Flexible to changes in the future. ● One central communication solution that acts as a class manager. ● Managing classes could be done in a more structured way. ● Able to localise variable initialisation. **Disadvantages**: ● Longer code	**Alternative B**: Game as a method **Advantages**: ● Only need one handler to handle game operation. **Disadvantages**: ● Code can be too messy with the intention to initialise most game components in a single method.
Decision: A – Game as a class With our team's intention in designing the code, we believe the game class will act as a main class or manager class where most game components would be declared and	

instantiated. Hence, game logic can be handled accordingly. And future changes can be done easily through one the classes instead of directly modifying the method.

Issue: Should move become class or a method?	
Alternative A: Move as a class **Advantages**: ● Debugging can be done easier when it comes to logic errors. ● Able to separate handlers on performing game movement logic. **Disadvantages**: ● Extra class to be maintained.	**Alternative B**: Move as a method **Advantages**: ● Less class to be created. ● Player can communicate to token directly to perform moves. **Disadvantages**: ● Another class may have too many responsibilities to process move's responsibilities.
Decision: A – Move as a class While move itself sounds like a method or a function where in this case each token all have the same move, there is more to it in Nine Men's Morris when designing it an object-oriented way. Making move a class allows it to be more flexible in a sense where in the future some changes may be required where move could be more complex.	

○ Explain two key relationships in your class diagram, e.g. why is something an aggregation not a composition?

Issue: Should computer class aggregate or compose from player class?	
Alternative A: Aggregation **Advantages**: ● Changes made could be more efficient. ● Able to reuse its superclass since the two classes introduced will have similar logic. **Disadvantages**: ● Future changes in superclass might break its subclass.	**Alternative B**: Composition **Advantages**: ● Changes to be make can be more specific. **Disadvantages**: ● More classes to maintain. ● Longer and even more complicated code to implement in the beginning. ● Code can get messy.

Decision: A – Aggregation Computer class shares the same responsibilities as the player class. However, computer class will have slightly different responsibilities due to the nature of itself where in this case a computer could perform its tasks automatically without any outside input. Hence, given the business logic or in this case the game logic, computer class will have the same set of actions as players do with tweaking to allow automatization.

Issue: Should board class aggregate or compose from game class?	
Alternative A: Aggregation **Advantages**: ● Changes made would be quicker or efficient. **Disadvantages**: ● Changes to game does not necessarily affect board. ● Tight coupling in this case	**Alternative B**: Composition **Advantages**: ● Future changes specific to the class itself ● Loose coupling ● Align with game logic where two do not share the same responsibilities. **Disadvantages**: ● Longer code in the beginning ● Code has to be properly structured in the beginning.
Decision: B – Composition We believe by using composition would result in loose coupling between game and board classes, due to our intention in making game class more a manager class and with this game class would become less of what we called god class since most of the game's requirements would not be heavily centred in game class's methods.	

○ Explain your decisions around inheritance, why did you decide to use (or not use) it? Why is your decision justified in your design?

Issue: In designing the architecture is the use of inheritance would be appropriate in some classes?	
Alternative A: Using inheritance **Advantages**: ● Minimise code duplication. ● Easier to debug where the problem is. ● Only one class needs to be updated if an error occurred or change of logic. **Disadvantages**: ● Tight coupling between classes may occur when inappropriately implemented. ● Changes to superclass might affect every of its subclasses even though the changes may not be required in the subclass	**Alternative B**: Not using inheritance (one class with more methods (for example)) **Advantages**: ● A standalone class ● Classes have their own behaviour or responsibilities. **Disadvantages**: ● Code duplication occurs when classes have similar nature.
Decision: A – Using inheritance For example, here computer class inherits from player class. Where rather than creating extra methods in player that they player itself would not really use unless the player is a computer in this case, so making methods in a computer class that inherits all attributes and methods from player class would be more appropriate to separate the use of it's two and allow further changes	

○ Explain how you arrived at two sets of cardinalities, e.g. why 0..1 and why not 1...2?

Issue: Should player to move cardinalities be 0..* or 0..1?	
Alternative A: 0..* **Advantages**: ● Human error is more tolerable in this case to allow player to perform another in a single turn. **Disadvantages**: ● Human error may persist without proper guide.	**Alternative B**: 0..1 **Advantages**: ● Human error input would be more likely to be prevented. **Disadvantages**: ● Player could only do up to one move at a time. ● Less human error means prevention needs to be make early
Decision: A – 0..* We believe this is more aligned to the game's logic. Furthermore, we also consider taking an opponent's token as a single turn rather than another turn since taking a token is a consequence of forming a 3-token row when a player makes a move and there might be a scenario where a player could not perform a legal or appropriate move. Hence, player could have a 0 to many moves.	

Issue: Should move to position cardinalities be 1..2 or 0..2?	
Alternative A: 1..2 **Advantages**: ● Memory allocation could be reserved since no instantiation in the beginning is needed. **Disadvantages**: ● Extra variable to maintain to keep the token counter.	**Alternative B**: 0..2 **Advantages**: ● All tokens are instantiated to ensure no missing token. **Disadvantages**: ● For loop in the beginning of the game which may seem irrelevant since all token does not to be placed in the beginning
Decision: A – 1..2 Even though placing a token is considered as a move, we think that initial placement of a token is an initialisation of a token into a board or position so it's still a movement from one position to the same position. Hence, 1 to 2 cardinalities make more sense with code design.	

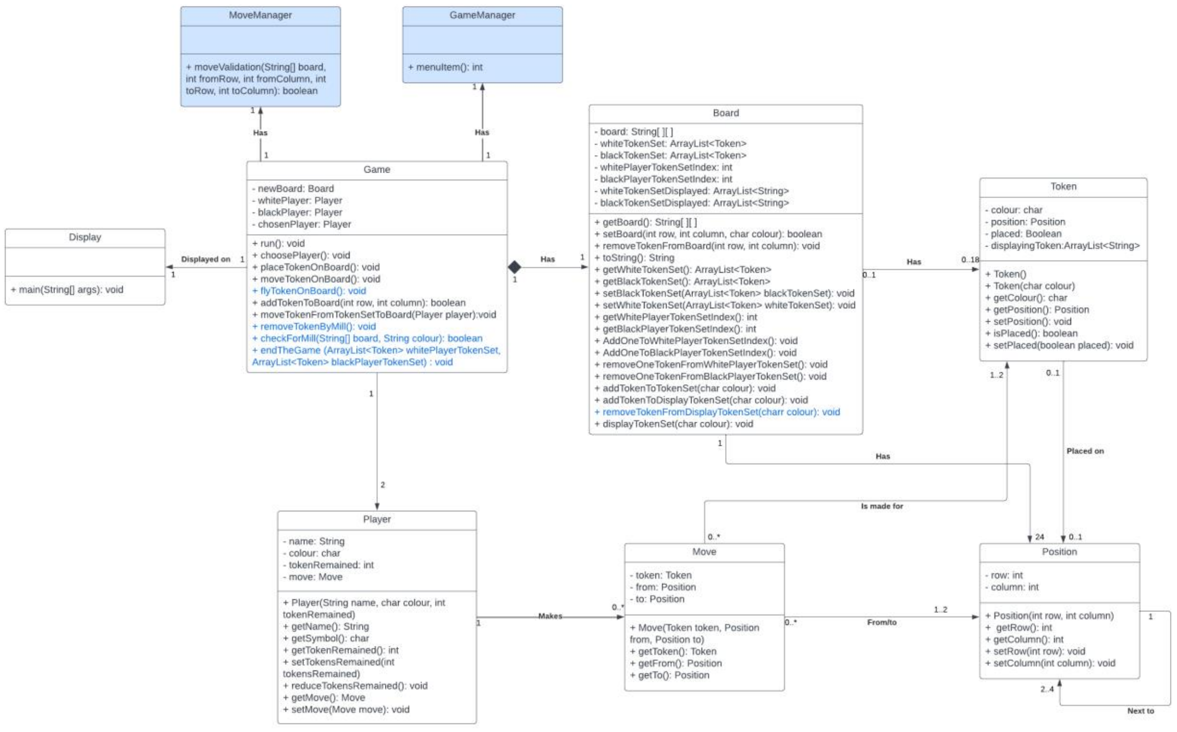
○ Explain why you have applied a particular design pattern for your software architecture? Explain why you ruled out two other feasible alternatives?

Issue: Which object-oriented programming design pattern is best used for Nine Men's Morris game?	
Alternative A: Creational patterns **Advantages**: ● Provide tools for creating objects that improve flexibility and allow for code to be reused. ● Abstractions allow more changes in the future. ● Development does not have to be done all at once. **Disadvantages**: ● Some classes might become too complicated to implement due to interface.	**Alternative B**: Structural patterns **Advantages**: ● Explain how to combine objects and classes into more complex structures, while maintaining the structures' flexibility and efficiency. ● Relationships between objects are clearly defined. **Disadvantages**: ● Code may become too complex due to interfaces and classes defined. ● More complicated to connect classes to interfaces.
Alternative C: Behavioural patterns **Advantages:** ● Ensure that objects can communicate and assign their roles effectively with each other. ● Dependencies between communicating objects are reduced. ● Each object has a set of responsibilities. **Disadvantages:** ● Limited when it comes to expanding objects' responsibilities. ● Might overcompensate on handling communication between objects.	
Decision: It depends on the situation, for example, the computer class which will extend the Player class meaning giving the abstraction between them, in here, the option A, creational pattern, is used.	

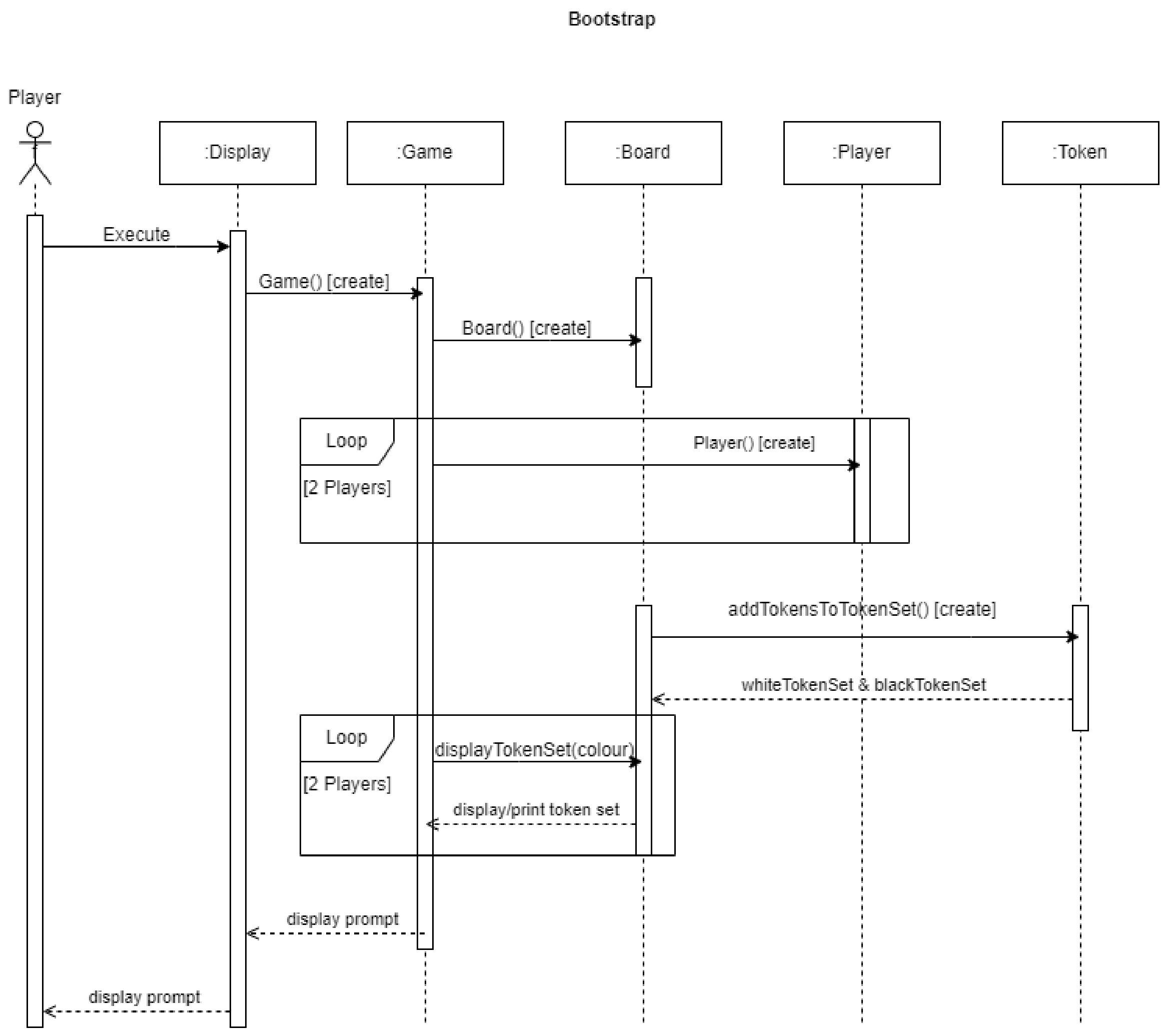
SPRINT 3

Class Diagram

Before is available at sprint 2 documentation above and changes are highlighted in blue

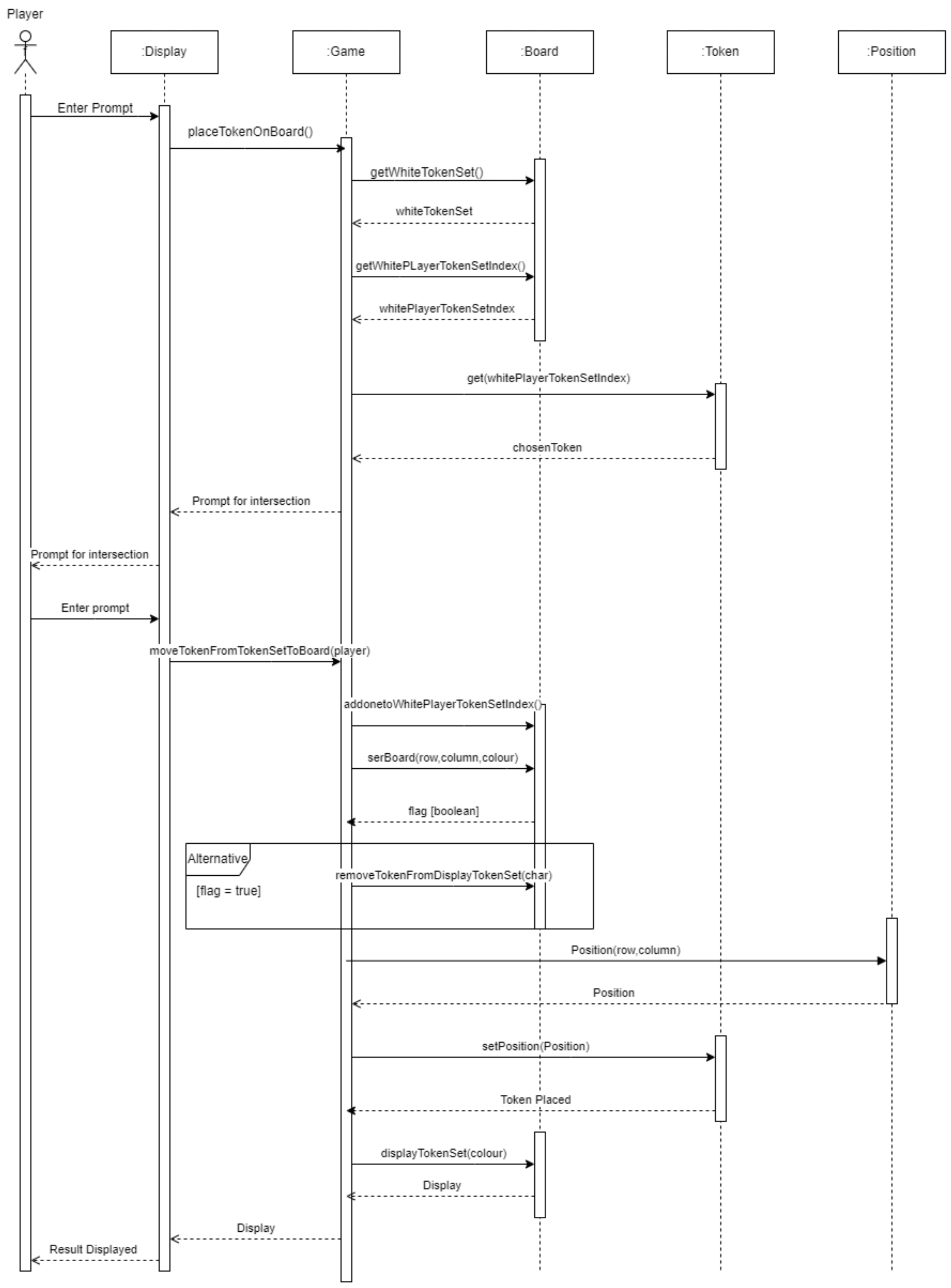

MoveManager
+ moveValidation(String[] board, int fromRow, int fromColumn, int toRow, int toColumn): boolean
GameManager
+ menuItem(): int
Has
1
1
Has
1
1
Game
- newBoard: Board
- whitePlayer: Player
- blackPlayer: Player
- chosenPlayer: Player
+ run(): void
+ choosePlayer(): void
+ placeTokenOnBoard(): void
+ moveTokenOnBoard(): void
+ flyTokenOnBoard(): void
+ addTokenToBoard(int row, int column): boolean
+ moveTokenFromTokenSetToBoard(Player player):void
+ removeTokenByMill(): void
+ checkForMill(String[] board, String colour): boolean
+ endTheGame (ArrayList<Token> whitePlayerTokenSet, ArrayList<Token> blackPlayerTokenSet) : void
Display
+ main(String[] args): void
Displayed on
1
1
Has
1
1
Board
- board: String[ ][ ]
- whiteTokenSet: ArrayList<Token>
- blackTokenSet: ArrayList<Token>
- whitePlayerTokenSetIndex: int
- blackPlayerTokenSetIndex: int
- whiteTokenSetDisplayed: ArrayList<String>
- blackTokenSetDisplayed: ArrayList<String>
+ getBoard(): String[ ][ ]
+ setBoard(int row, int column, char colour): boolean
+ removeTokenFromBoard(int row, int column): void
+ toString(): String
+ getWhiteTokenSet(): ArrayList<Token>
+ getBlackTokenSet(): ArrayList<Token>
+ setBlackTokenSet(ArrayList<Token> blackTokenSet): void
+ setWhiteTokenSet(ArrayList<Token> whiteTokenSet): void
+ getWhitePlayerTokenSetIndex(): int
+ getBlackPlayerTokenSetIndex(): int
+ AddOneToWhitePlayerTokenSetIndex(): void
+ AddOneToBlackPlayerTokenSetIndex(): void
+ removeOneTokenFromWhitePlayerTokenSet(): void
+ removeOneTokenFromBlackPlayerTokenSet(): void
+ addTokenToTokenSet(char colour): void
+ addTokenToDisplayTokenSet(char colour): void
+ removeTokenFromDisplayTokenSet(charr colour): void
+ displayTokenSet(char colour): void
Has
0..1
0..18
Token
- colour: char
- position: Position
- placed: Boolean
- displayingToken:ArrayList<String>
+ Token()
+ Token(char colour)
+ getColour(): char
+ getPosition(): Position
+ setPosition(): void
+ isPlaced(): boolean
+ setPlaced(boolean placed): void
1..2
0..1
Placed on
1
Has
Is made for
1
2
Player
- name: String
- colour: char
- tokenRemained: int
- move: Move
+ Player(String name, char colour, int tokenRemained)
+ getName(): String
+ getSymbol(): char
+ getTokenRemained(): int
+ setTokensRemained(int tokensRemained)
+ reduceTokensRemained(): void
+ getMove(): Move
+ setMove(Move move): void
Makes
1
0..*
0..*
Move
- token: Token
- from: Position
- to: Position
+ Move(Token token, Position from, Position to)
+ getToken(): Token
+ getFrom(): Position
+ getTo(): Position
0..*
From/to
1..2
24
0..1
Position
- row: int
- column: int
+ Position(int row, int column)
+ getRow(): int
+ getColumn(): int
+ setRow(int row): void
+ setColumn(int column): void
1
2..4
Next to

Sequence Diagrams



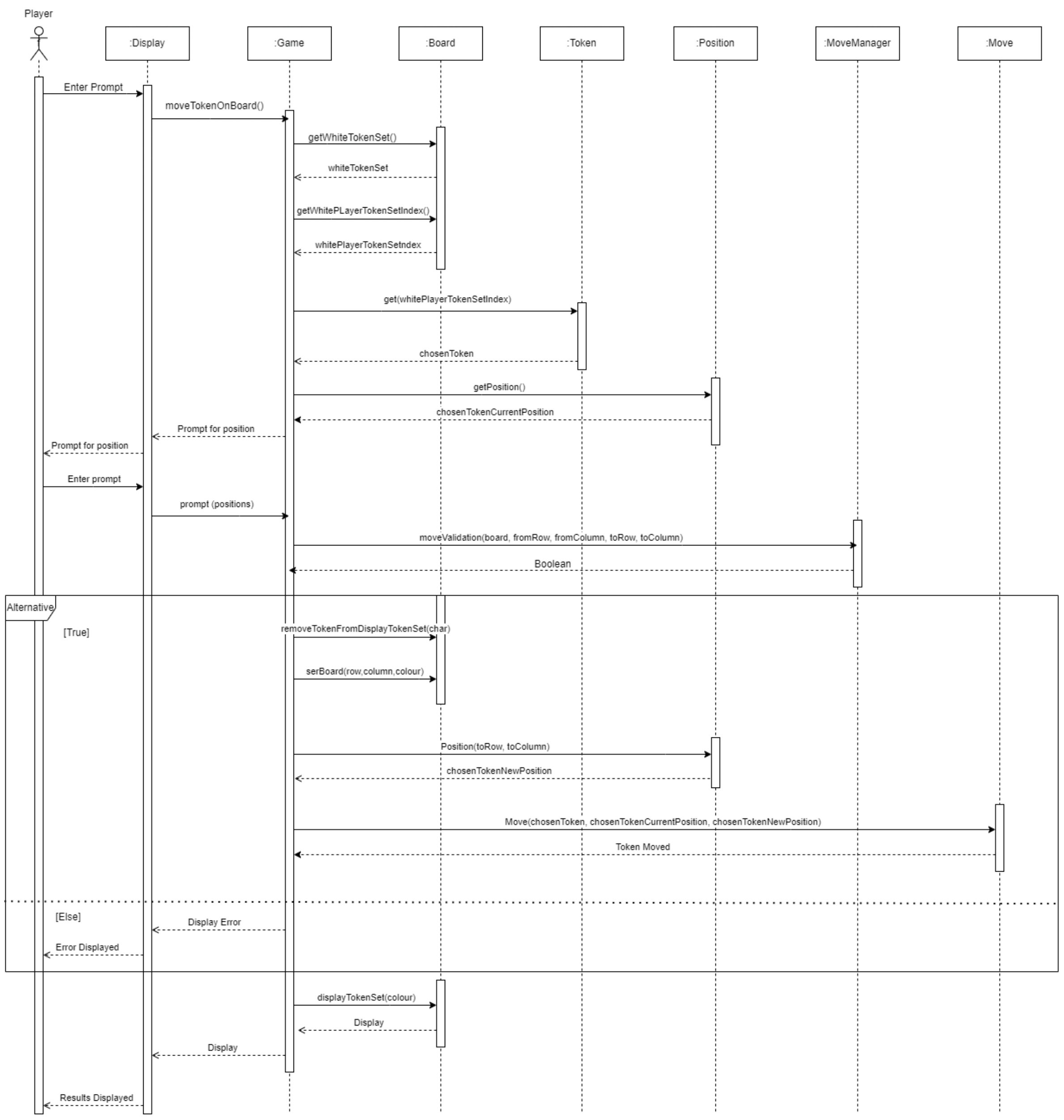
Placing token

Given white player is already being chosen



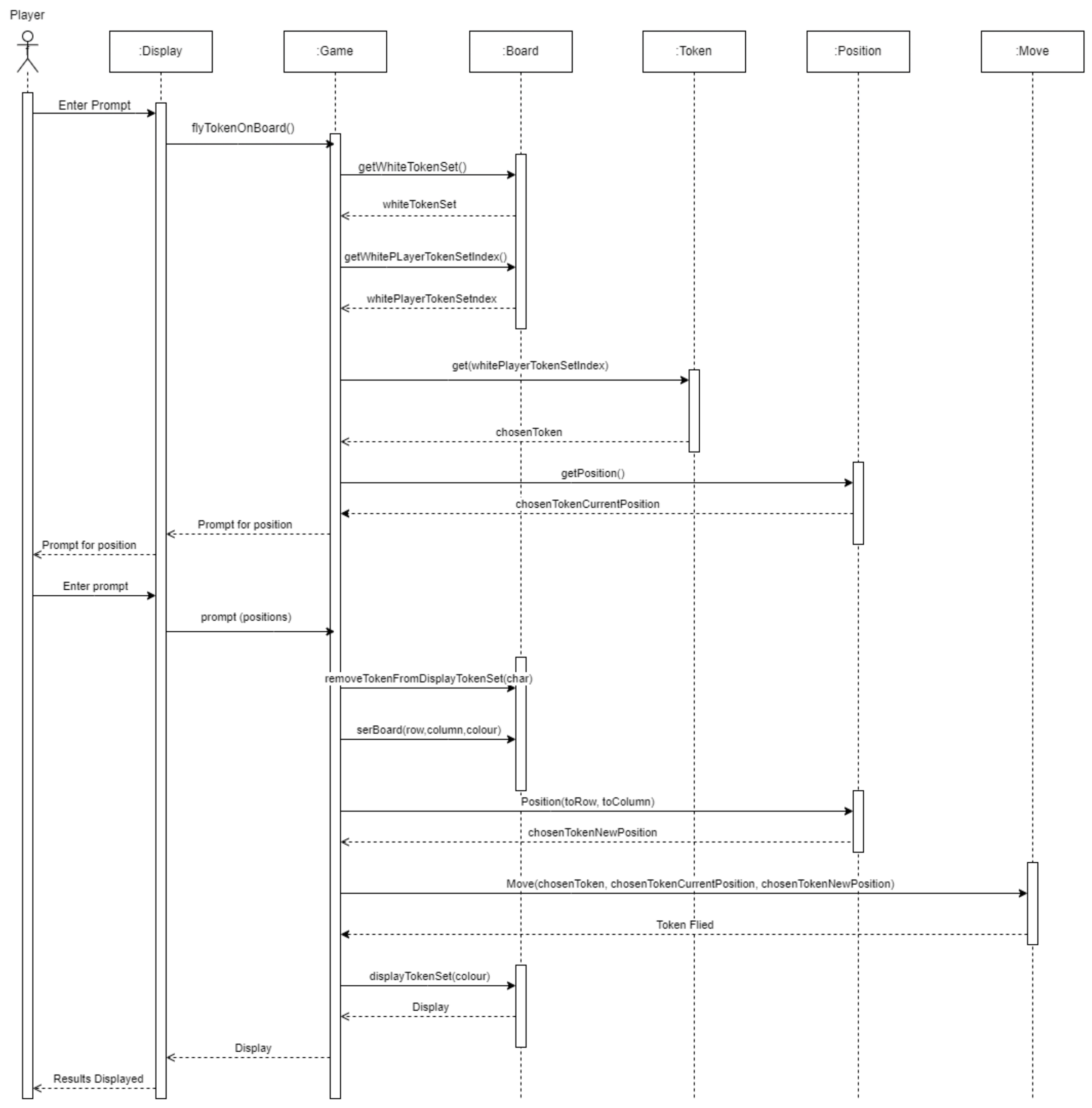
Moving Token

Given white player is already being chosen

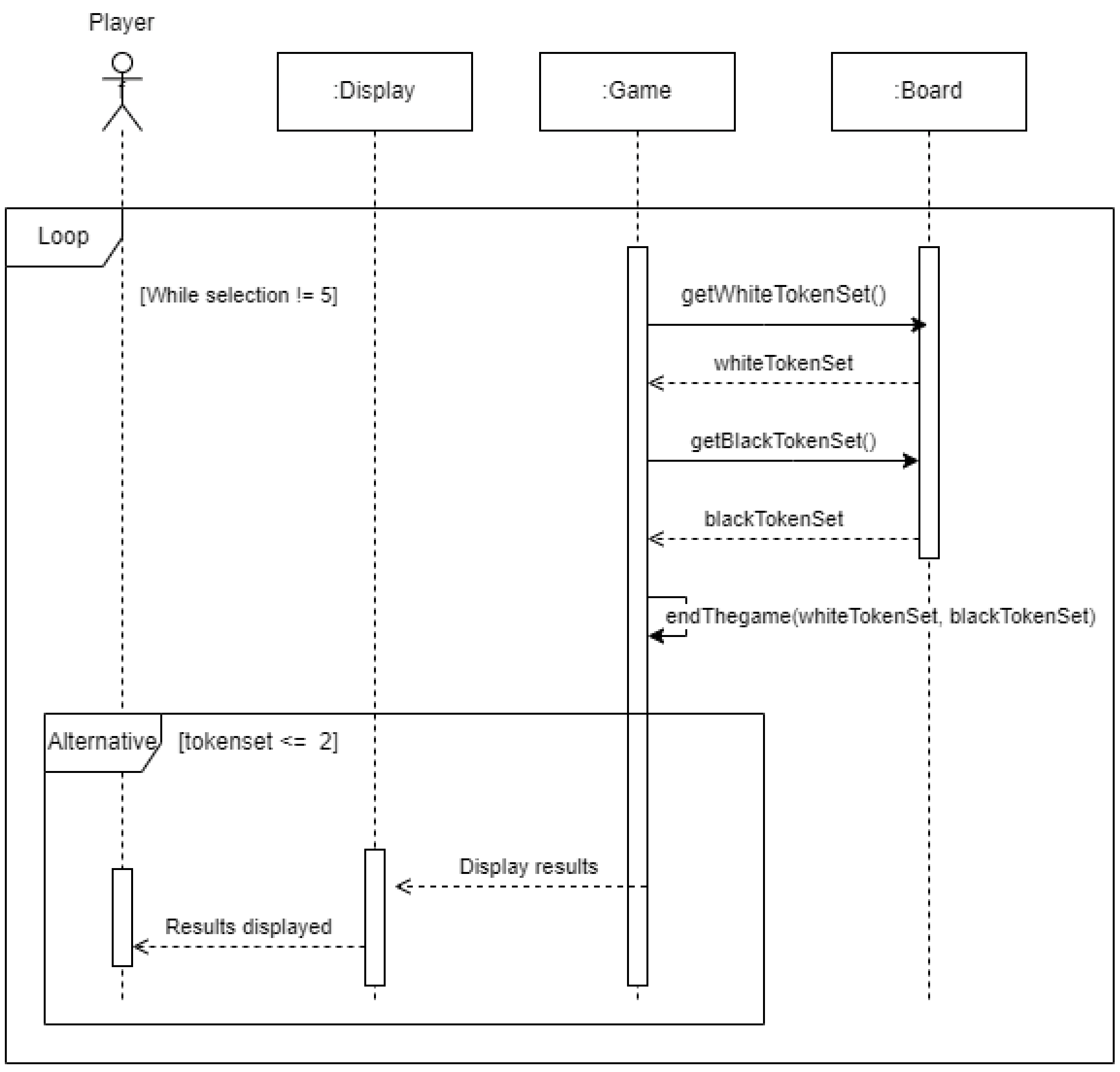


Flying Token

Given white player is already being chosen



Winning Condition



Design Rationales

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.

Issue: Should the design architecture be revised?	
Alternative A: Revise **Advantages**: • Responsibilities are evenly spread and be more appropriately related to the game logic. **Disadvantages**: • Code changes which require more time	**Alternative B**: Not Revising **Advantages**: • Teams familiar with code with. Hence, be more aware what each class responsibilities are. **Disadvantages**: • Heavily rely on board class
Decision: **A** We decided to revise the architecture to fit this sprint's requirements as we feel like the last architecture is too catered towards last sprint's requirement which is to display and allow to place all nine tokens. Hence, the architecture is updated to fit the responsibilities such as the mil function, winning condition, human values, and quality attributes we decided are non-trivial towards the game. By introducing new classes as shown on the class diagram.	

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.

Issue: Which quality is most appropriate to be implemented in the game design?	
Alternative A: Reliability **Advantages**: • Game is playable. • Crucial to meet the end condition of the game. **Disadvantages**: • Thorough testing may be required to ensure reliability	**Alternative B**: Portability **Advantages**: • Game can be played everywhere and any time. • Increases the audience. **Disadvantages**: • Another set of display which means more code to fit the need of a mobile • Time constraint
Alternative C: Usability **Advantages**: • Ease of use • Accessible **Disadvantages**: • Constant maintenance to ensure the best of the quality.	**Alternative D**: Efficiency **Advantages**: • Game easier to load. • Can be distributed easily in a sense that it does not require much space to safe. **Disadvantages**: • Code may break if cutting up on the wrong part of the design.
Decision: **A, C & D** We think that all of the listed alternatives above are non-trivial to the well-being of the game. However, we think that the three alternatives that we decided to include are essentials for this sprint and portability is not truly our main objective given the time constraints and our resources as a team in terms of mobile development to accommodate this portability attribute. With reliability quality attribute, we ensure that the game run smoothly and able to serve its purposes and operating with the correct logic with the aims of smooth and reliable gameplay. Usability, we ensure that the game is easy to access and play with by giving set of instructions in correct wording and displaying all necessities on the display. And lastly efficiency attribute, we aim to keep the code as simple as it could get and ensuring the game can run efficiently from the beginning to the end like booting up the game with executable program and ending the game with a single command whilst maintaining the space required to run the game.	

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.

Issue: Which human value is most appropriate to be implemented in the game design?	
Alternative A: Social Recognition **Advantages**: • Playing the game is awarded and not just for pleasure itself. • Motivates player to continue to play the game **Disadvantages**: • Rivalry may exist which could introduce an unwanted culture towards the game if it grows to be toxic enough	**Alternative B**: Helpful **Advantages**: • Inclusive towards new player • Less error in playing the game. **Disadvantages**: • More steps to be implemented. • Changes towards the display may be required depending on how complex the helpful value is implemented here
Alternative C: Intelligent (computer) **Advantages**: • Single player option available • Can be played anytime. **Disadvantages**: • More steps to be implemented. • Game can be monotone if implemented incorrectly	**Alternative D**: Pleasure **Advantages**: • Key point of all games • Big selling point **Disadvantages**: • Players gets addicted to the game.
Decision: **D** The team has decided to include all human value included above with some exception that we have not fully implemented them. Hence, only pleasure included in this sprint. We believe by implementing these human values the gaming experience would be enhanced. And able to serve its purposes accordingly which are to be enjoyable and playable. For social recognition we decided to keep track of how many each player win in a session or app runtime. As for pleasure, we do not have a concrete example of it rather than the playthrough itself to be enjoyable. Furthermore, we plan to implement intelligent value in the next sprint by introducing computer to the game. To allow player to enjoy the game without relying on the existence of another player whilst also challenging the player due to unpredictable computer movement.	

Video Demonstration

Link:

https://drive.google.com/file/d/1RPEMOIFY74MfUGNW7riG7Q0HJSHpTy6G/view?usp=share_link

Executable Instructions

1. Make sure latest JDK Development Kit is installed on local machine which in this case we use Windows as our main operating system.
2. Clone the project from GIT.
3. Once cloned, go to "out/artifacts/FIT3077_project_jar" folder
4. FIT3077-project.jar should be under the folder.
5. Copy the file as absolute path.
6. Go to command prompt.
7. Run the command java -jar "Pathname".
8. Example:
   java -jar C:\Users\Chandra\IdeaProjects\project1\out\artifacts\FIT3077_project_jar\FIT3077-project.jar
9. Enter (Project should be executed).

Reference

(n.d.). SVG Repo - Free SVG Vectors and Icons. Retrieved April 2, 2023, from https://www.svgrepo.com/

Nine Men's Morris | Online Games for Kids. (n.d.). Toy Theater. Retrieved April 2, 2023, from https://toytheater.com/nine-mens-morris/

Wood color hex code is #BA8C63. (n.d.). Color-Name.com. Retrieved April 2, 2023, from https://www.color-name.com/wood.color