FIT3077 Software Engineering: Architecture and Design

Assignment 1: Sprint One

MA_Friday2pm_Team5

Faculty of Information Technology, Monash University

Semester 1, 2023

Table of Contents

Team Information **2**
- Team Name and Team Photo 2
- Team Membership 3
- Team Schedule 4
  - Team Meetings 4
  - Work Schedule 4
- Workload Distribution 4
- Technology Stack and Justification 6
  - Development Tools 6
    - Programming Language 6
    - Framework - Swing v.s. JavaFx 7
  - Project Management Tools 8
    - Communication 8
    - Documentation 8
    - User Story Board 8

User Stories **9**
- Figures 10

Domain Model **11**
  - Basic Requirements 11
    - Overview 12
    - Capturing Interactions - Actions 12
    - Assumptions 13
  - Advanced Requirements (and Display) 13

Lofi Prototype **15**

Team Information

Team Name and Team Photo

Monash Java Lamas - MA_Friday2pm_Team5

Team Membership

Student Name: **Gan Jia Horng** (Right most in photo)
Student Id: 32579160
Student Email: jgan0022@student.monash.edu
Technical Strengths:

1. Keen to pick up new languages - Although not really needed for this project.
2. Very interested in design patterns! Reason I took the unit in the first place.
3. Experience with enterprise resource systems (ERP).
   Worked with SAP for my internship

Professional Strengths:

1. Leadership. Willing to provide directions and lead the team
2. Self-motivation. Deadlines are a great motivator!

A fun fact about me: I often watch LoL eSports! Although I am too busy to play the game nowadays, I am very caught up with the games and results. For me it is one of the most enjoyable entertainment out there with how exciting each game is!

Student Name: **Ravindu Santhush Ratnayake** (Left most in photo)
Student Id: 32634269
Student Email: rrat0006@student.monash.edu
Technical Strengths:

1. Solidity - I develop smart contracts and create dApps on the Ethereum blockchain.
2. React - a javascript framework I use to create reusable UI components

Professional Strengths:

1. Adaptability to new environments - Can smoothly adapt with the working styles of new team members

A fun fact about me: I am a huge fan of the Resident Evil Video Game Series from Capcom. The first game I ever played was Resident Evil 4(2005). A new remake was released very recently (March 24, 2023) and now I am busy playing the game and procrastinating on my university work. 🙂

Student Name: **Danesh Carmel Domingo Mariapan** (Middle in photo)
Student Id: 31965601
Student Email: dmar0038@student.monash.edu
Technical Strengths:

1. Experience with industry software development process and mobile application development using Flutter and Dart during internship
2. Interests in Software Engineering as well as data analytics / aspects in Data Science

Professional Strengths:

1. Enjoy trying new things, meeting new people and exposing myself to new cultures, ideas and environments
2. Easily adaptable to work styles, gained experience in collaboration, discussions and working with an agile team during internship

A fun fact about me: I loved playing sports growing up, especially: table tennis, badminton, football - I am quite a competitive person in game / sporting environments!

Team Schedule

Team Meetings

Weekly meetings are conducted regularly every Friday after the workshop to ensure everyone is making progress with their delegated tasks. The goal of the meeting is to prevent underestimating complexity of a task so that we can set realistic expectations for the development of our project towards the end of a sprint.

Meetings are also taken as a good opportunity to address issues and doubts each team member might have with each other as well as our tutor for any clarifications that might be necessary. It is also for the team to reorganise and manage their assigned tasks in favour of a higher priority need, as well as future concerns and tasks.

Work Schedule

The team does not have a set work schedule due to differing personal schedules as well as commitments to other units and responsibilities as students. However, the User Stories each member is assigned to has an agreed upon due date to allow time for review and understanding the team's progress. We will be able to track collaboration efforts using Git to hold each other accountable with commit history as proof for their efforts. Git provides a detailed history of all changes made to the codebase, so team members can freely view code change history to determine who made what changes and when.

Workload Distribution

At the start of each Sprint, the team will first determine and select the User Stories that are needed to be prioritised for the next Sprint referring to their priority. Team members then select the User Story they wish to implement. In the event that 2 team members wish to implement the same User Story, the third team member can act as a tiebreaker to determine who is more suitable for the task. The assigned User Story will then be worked on for the duration of the entire Sprint.

Under each user story contains a task list that has the specifics of our team's implementation expectations. When every criteria is fulfilled, the developer should change the status to "completed" and move the User Story to the "completed" list. We expect each team member to pick up the same amount of responsibilities and ensure the workload is distributed equally.

Sprint 1 Distribution:

- Team Information - This task is evenly between all team members, each team member is responsible for filling in his own respective information regarding their personal information, descriptions and schedules

- User Stories - This task is evenly split between all team members to initially produce an appropriate number of user stories after study on the project brief / material. After the user stories are produced, all team members will collaborate on identifying the more effective and relevant user stories to the project and eliminate redundant or unnecessary user stories after discussion

- Basic Architecture - This task is done in collaboration with contributions from all team members. Team members participate in sharing and visualising ideas on the initial domain model. The team will then decide on the final domain model of the project after discussion and combining relevant parts and individual designs of the domain model. The documentation and justification will then be produced by the whole team based on the final domain model.

- Basic UI Design - This task is done in collaboration with contributions from all team members. Team members initially share ideas and designs of the UI structure and its components. After discussions, a final low-fidelity (low-fi) prototype will be created by combining all the effective ideas and parts of individual design.

Technology Stack and Justification

Development Tools

Programming Language

The chosen programming language for development significantly impacts how our end product is developed. Based on initial ideas of available object-oriented programming languages, the team has narrowed down the possible options to the three main languages that would be the most efficient and effective for this project and also that all team members would have had experience working with personally or in their previous units.
These programming languages are: **Java, Typescript and Python**.

Java
Java is the poster child of object-oriented programming languages. It is an attractive option for the team since all the members have prior experience developing with it. Since object oriented modelling is closely tied to how the code will be resembled when developing in Java, it is an added advantage of the language for this unit that has an emphasis on Software Architecture. The main drawback is developing the GUI, which will require the usage of additional Java frameworks which will be brand new experience working with for all team members.

TypeScript
Another point of consideration is TypeScript. TypeScript supports object oriented modelling with simpler syntax than Java, although it is not the main paradigm of the language. The main strength of the language is the graphical interface, as it can directly render anything on the web without any frameworks. The drawback of this language is that it does not enforce following typing and is more troublesome to set up with a lot of overhead, not to mention its high learning curve of the technology stack which may significantly impact the delivery schedule and workflow of the team.

Python
While Python does have object oriented support, it is not the best in terms of its development experience since it does not enforce typing. Our team is concerned if its feature will make programs harder to debug and encourages usage of bad design patterns or principles that do not follow an object-oriented approach. While Python has the simplest syntax out of all three options, it does not offer much else in terms of significant benefits for the team and using it for this specific project. The most popular GUI of the language - tkinter is also harder to develop with for larger projects.

The table next page summarises the features of all the candidate programming languages. Ultimately, we decided to go with Java as it is tied with the spirit of the unit and it is the most experience our team had out of the other options.

Table Summary: Features of Java, TypeScript, Python

	Team Experience	**GUI**	**OOP Support**
Java	Yes	Bad	Great
TypeScript	No	Yes	Good
Python	Yes	Bad	Bad

Framework - Swing v.s. JavaFx

For a better user experience, the project needs to have a graphical user interface (GUI) to display the game board and allow cursor interactions as opposed to console user interface which is troublesome to navigate with. Since our team decided to use Java, the next logical step is to decide which GUI framework supported by Java to use for the project. For Java, there are two popular GUI options for the team to choose from which to develop with, namely JavaFX and Swing,

JavaFX has the option to use FXML and CSS-like styling - a technology similar to web pages the team had experience with in other units. In our experience, this is vital since it uses a declarative style of creating UI which reduces code dramatically. Additionally, JavaFX features a drag and drop UI creation which we believe will smoothen our UI development. While it won't be the most attractive UI, the concession makes sense as it was not the main focus of the project.

Swing is an older GUI framework than JavaFX. The main drawback for us is the need for imperative code to declare every single component of the application. As compared to FXML, it is way more verbose as everything has to be instantiated manually and declared at which position the component should appear.

JavaFX might require more resources to learn for developers already familiar with Java and Swing, as it introduces new UI components and ways of building UIs with FXML and CSS-like styling. However JavaFX is easier to learn due to its modern, clean API and separation of UI design and logic with FXML and CSS-like styling. In contrast, Swing uses older APIs and might be more challenging for beginners like us.

Our team decided to go with JavaFX as the GUI framework for the project. JavaFX is preferred due to its status as a modern framework relative to Swing and having clean separation of design and logic.

Project Management Tools

Communication

The team primarily uses WhatsApp for communication. It is sufficient for communication purposes of a group of size 3. Occasionally, if meetings are required, a virtual Zoom meeting will be scheduled and the link for it will be shared via WhatsApp group chat.

Documentation

Design documents will be created on Google Drive. Google Drive eases collaboration between team members by showing live changes. Git is not suitable for non-text files which makes it difficult to share some of the documents. While Markdown files can be used to create documentation, the overhead of committing and pushing to git is troublesome and the advantages do not apply with how the team handles documents.

User Story Board

User Stories allocation and tracking will be done on the Trello board which is shared among all team members. User stories are created in the first sprint to breakdown the requirements of the project.

User Stories

For the advanced requirements, our team chose Advanced Requirements C:
" A single player may play against the computer, where the computer will randomly play a move among all of the currently valid moves for the computer, or any other set of heuristics of your choice"

1. As a player, I want to be able to place tokens on board in the first phase so that I can capture the tokens if I form a mill[1].
2. As a player, I want to move my tokens to a connected position in the midgame so that I can capture the opponents' tokens if I form a mill.
3. As a player, I want to move my tokens to any position when I have less than 3 tokens so that I can capture the opponents' tokens if I form a mill.
4. As a white player, I want to place a token on an empty board so that I can begin the game.
5. As a player, I want to remove my opponent's piece when he only has 3 pieces so that I can win the game.
6. As a player, I want to be able to see the game board so that I know where the tokens are.
7. As a player, I want to see whose turn it is, so that I can keep track of the game progress.
8. As a player I want to remove opponent's tokens that do not form a mill after forming a mill with my tokens so that I can weaken my opponent.
9. As a player I want to remove my opponent's tokens when all of them form a mill after forming a mill with my tokens so that I can weaken my opponent. (Figure 1 setup)
10. As a game board, I want to alternate turns between both players so that they can place/move/capture tokens on board.
11. As a game board, I want to put a victory message for the winner when his opponent has less than 3 tokens so that the winner can be notified.
12. As a game board, I want to put a victory message for the winner when his opponent has no legal moves so that the winner can be notified. (Figure 2 setup)
13. As a game board, I want to ensure players cannot place their token on a spot that is already occupied, so that the official game rules are maintained.
14. As a game board, I want to display the number of tokens not yet placed, so that the player knows which stage of the game he is in.
15. As a token, I want my colours to be identifiable visually, so that the player knows which side I am on.

Advanced requirements:

16. As a player, I want to be able to play against a computer player so that I can play 9MM when another human is not available.
17. As a computer player I want to randomly play moves that are legal so that the game will become fun and unpredictable for the human player.

[1] Mill = Same token colour 3 positions in a row

Figures

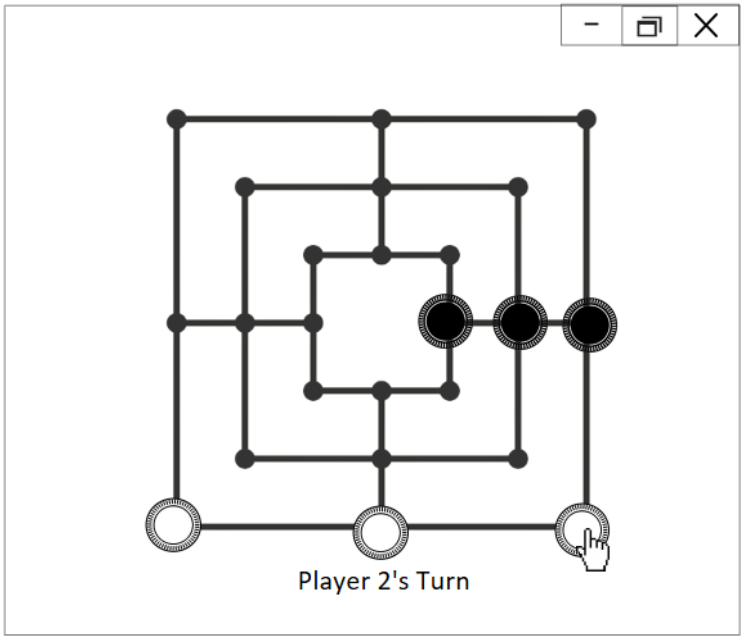


Figure 1: Black formed a mill in the middle row.
Black can still capture one of white pieces since all of white tokens form a mill.

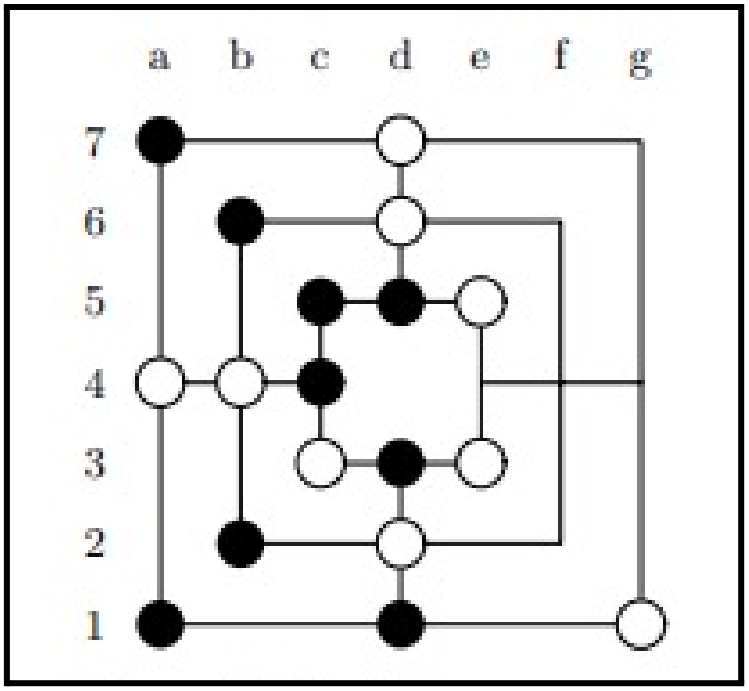
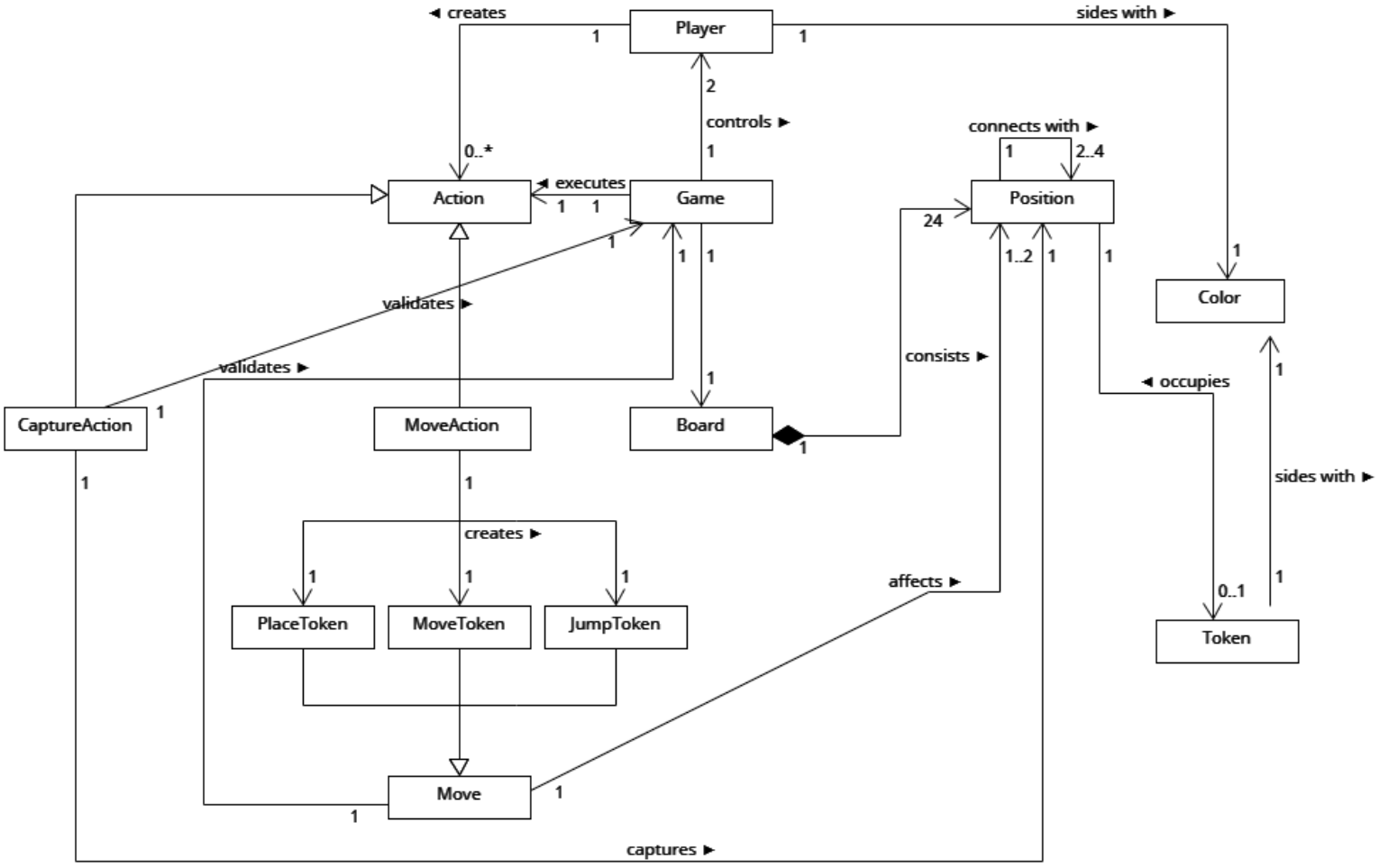


Figure 2: Black has lost the game, as he has no more legal moves. [*Source*]

Domain Model

Basic Requirements



Domain Model for 9 Men's Morris

The domain model is designed to cover both basic requirements of the 9 Man Morris (9MM) game. The advanced requirements will be covered in the below section. Each entity represents a specific aspect of the game, and relationships are established based on how they interact with another entity.

Overview

The Game class is the formulation of 9MM. It encompasses all aspects of the 9MM game, from the game rules to the state of the game. The main loop is also contained in the Game class alternating between the two Players, until one of them meets the victory condition. At a Player's turn, their intention to interact with the Game is abstracted by the Action class. Before the Player's intention is executed (Move, CaptureAction), it is first validated by the Game class as it is responsible for verifying if the action is in accordance with 9MM rules.

The Board class is part of the composition to Game class. The difference between a Board and Game is that Board serves as a data structure to store where Positions and Tokens are, whereas Game is the logic and the game rules are formed. Hence, Game alters the Board data based on the game rules implemented inside Game. This is also a decoupling move that ensures the responsibilities between both entities are not overloaded.

A Position is a unit that collectively forms the Board and they represent a single placeable location in 9MM. It is connected with other Positions just as drawn in a 9MM Board. Each Position may or may not be occupied by a coloured Token, and the Token's colour ultimately helps the Game to identify which side the Token is on. The Game identifies which token can Move or CaptureTokens interact with based on the colour of the Token and the current Player's colour.

Capturing Interactions - Actions

Fundamentally, Players create move actions but delegate testing of validity of moves to Game class. The motivation behind abstracting the player's interaction as Action is inspired by the Command Pattern. The Action class allows specific Actions to be parameterized and instantiated. For instance, the Player may want to Move or Capture a token, and both the commands are encapsulated inside an object. They have to be subsequently verified before executed by Game. The need for a distinction between Action and MoveAction, CaptureAction is to support the specific interactions for the game. As such, additional future interactions (e.g. Undo) with the Game can be done by extending the Action class.

Across different stages of the game, the MoveAction has a different subtype, which is to Place, Move or Jump a specific Token. They implement the Move interface similar to the Strategy Pattern where we capture the Move abstraction and encapsulate different implementations in derived classes. Move will inform Game of its desired Move that concerns which Position, and if it is a legal move it will execute the desired MoveAction. A case may be made that Move class should affect tokens in the domain model instead of Position, but our approach is to allow Move to inform Game which Position is affected by player action. In the end, the proposed Move has to be validated before executed by the Game class to check if it is legal, so we are not really concerned about privacy leaks.

Following closely after a MoveAction, a player may place 3 tokens in a row and will want to capture a token. The player will create a CaptureAction to take away one of the opponent's tokens, after verifying it is a legal move from Game. Alternatively, placing CaptureAction under Move subclass is under debate of the team as they both affect Position on board and have to be verified by Game, but due to their logical differences between moving a friendly token and capturing a piece we decide not to inherit the Move class for the time being.

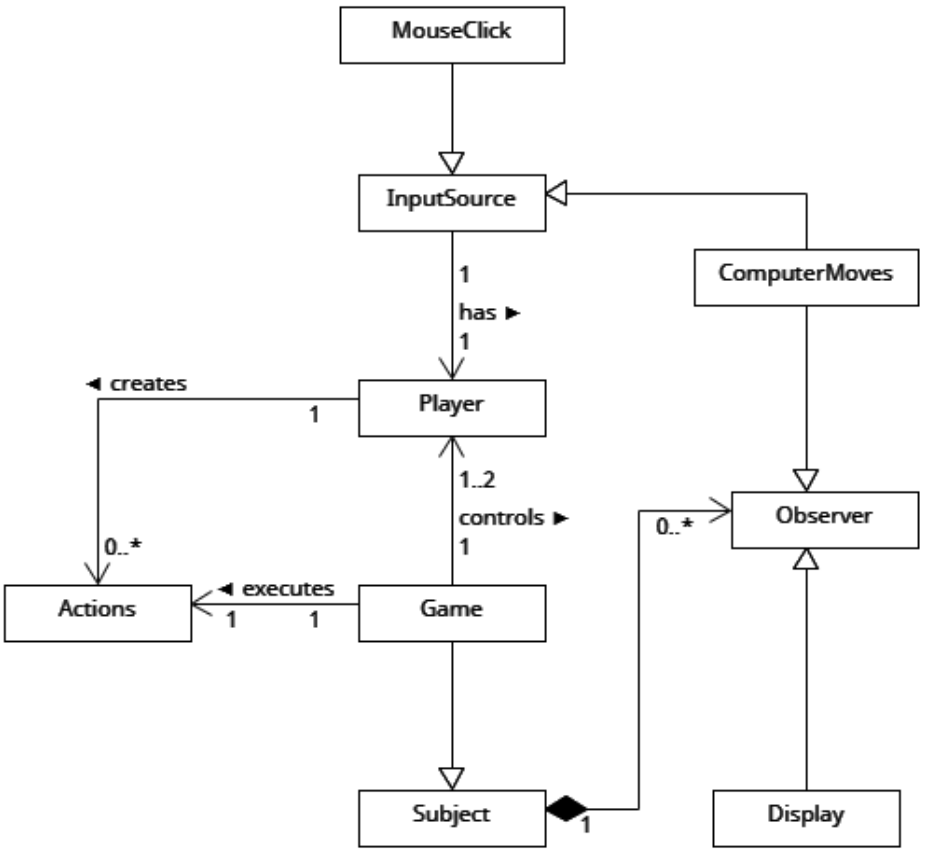
Assumptions

Forming a Mill is handled by the Game class internally. We believe the assumption is justified given the Game has the responsibility of determining legal moves, as such given that it has access to the game state it should have no trouble checking if the tokens are formed 3 in a row after a move.

Advanced Requirements (and Display)

For the advanced requirements, our team chose Advanced Requirements C:
A single player may play against the computer, where the computer will randomly play a move among all of the currently valid moves for the computer, or any other set of heuristics of your choice.

Our main concern about this additional requirement is it is less related to the 9MM domain but the application. As such, we need to dive into more technical terms in this requirement to give the full picture.



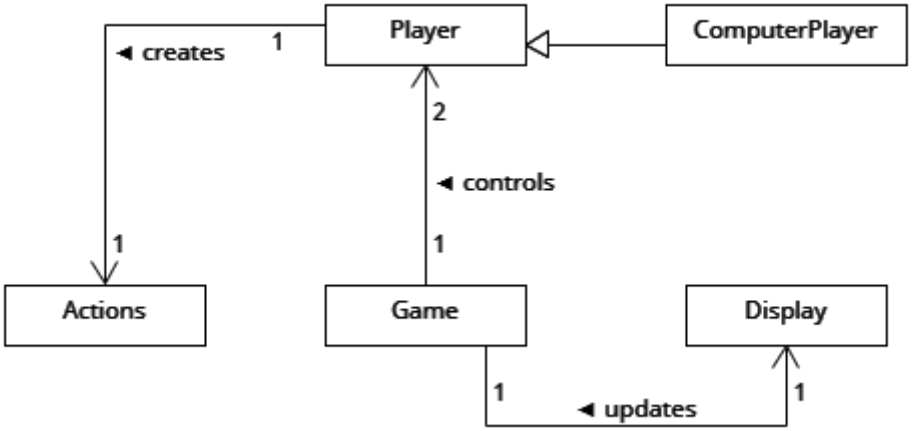
Model has been simplified for clarity purposes.

Each Player has also been extended to include an input source that determines how they interact with the Game. As such, the main characteristic that differentiates between Human and Computer players lie in different InputSources, where for MouseClick it will wait for a mouse click to create the Action,

The Game class has been extended to inherit the Subject class. In doing so, it has a list of Observers that are able to receive notifications when an event of interest occurs within the Game. The Observer pattern makes the code extensible without hindering the game logic such that it is able to notify other entities that are interested in the state of the game. In this case, we have identified 2 classes in particular that are the Game's observers, namely ComputerMoves and Display.

The ComputerMoves class has been identified as an observer as it needs to inform the Computer the state of the Game in order to calculate the next move. The algorithm to pick the computer's moves can be generated after receiving the state of the Game Board. In the future, optimal moves can be implemented here to beat 9MM[2], but for this project, our group only intends to let the Computer make random moves. On the other hand, Display can update the graphical user interface whenever notified by the Game class.

Another alternative design that has merit for discussion is having a ComputerPlayer class instead. Intuitively, ComputerPlayer is a subclass of Player as they share different implementations of the same behaviours.



However, in doing so Player and ComputerPlayer class are tightly coupled, and the Player class taking precedence with no good reason. Moreover, Player sees the board through the display and creates Actions whereas the Computer Player needs the game state as input to calculate the next move. Hence, this method is discarded.

The domain designs covered above separates the role of the input (InputSource), model (Game) and display. We believe this approach towards our domain design paves the way for Model-View-Controller when we discuss the software architecture of the game.

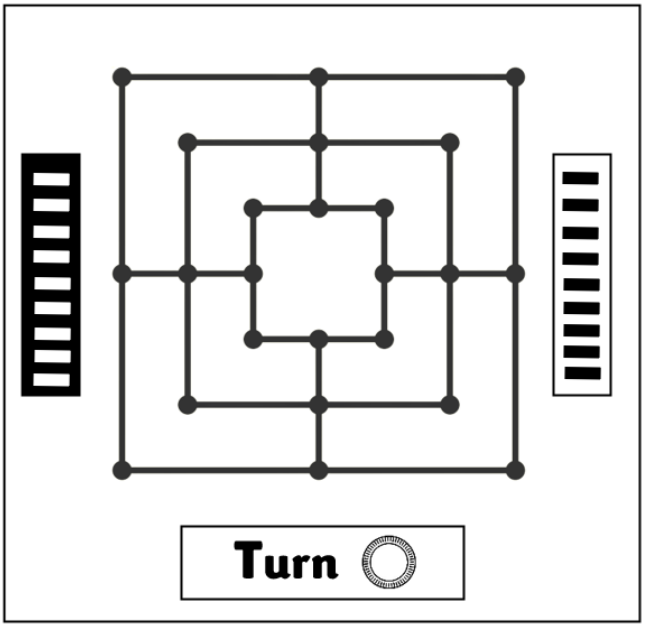
[2] http://library.msri.org/books/Book29/files/gasser.pdf

Lofi Prototype

1. Welcome Screen

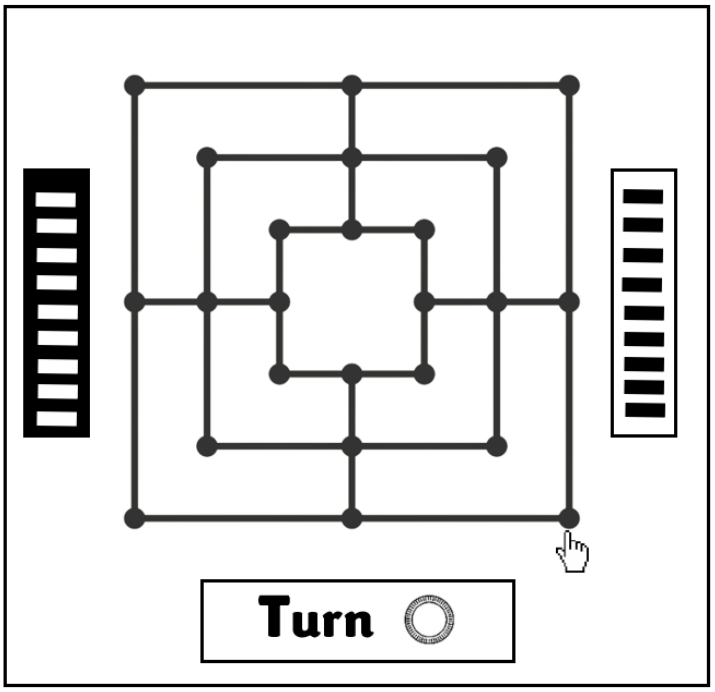
A simple but aesthetic layout displaying the “9 Men’s Morris” game title and a striking “Start” button inviting the player to play the game.

2. Initial Board

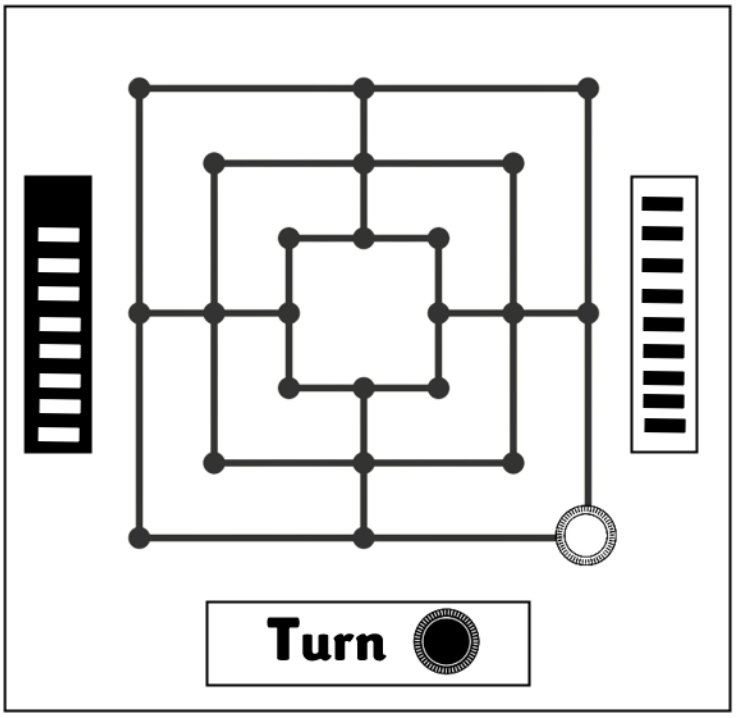


A clear board layout consisting of: 24 connected and placeable token locations and the token holders showing the number of each player’s available tokens.

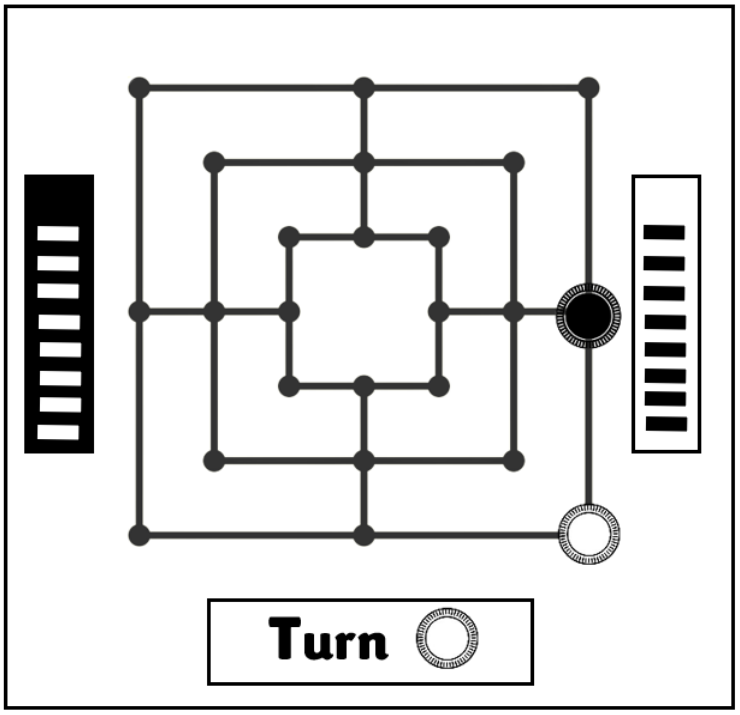
3. Placing Markers on Board



Player 1’s (White) Turn and placing a token on an available location

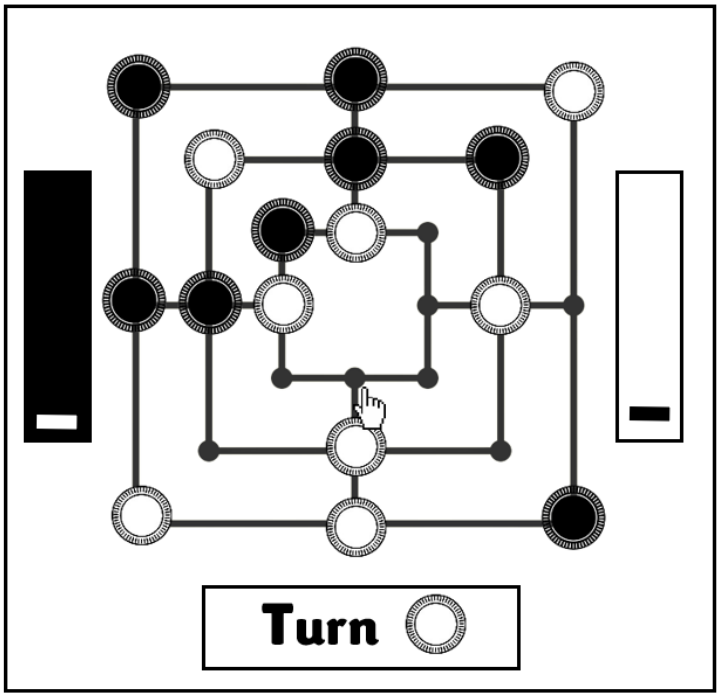


Player 2’s (Black) Turn after the previous move has been made. The left bar shows the number of tokens white has left.

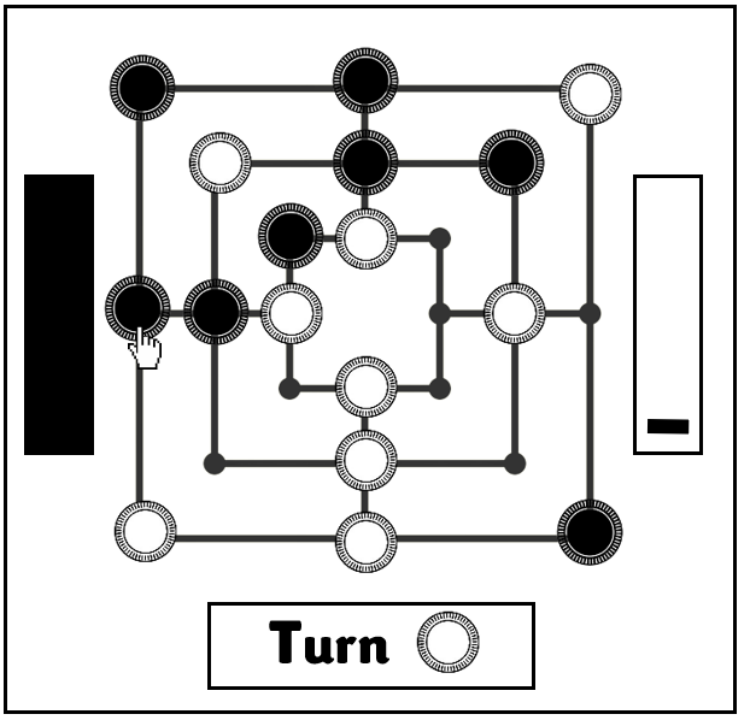


Back to Player 1’s Turn (White) to place a token on an available location.
The process repeats throughout the entire game.

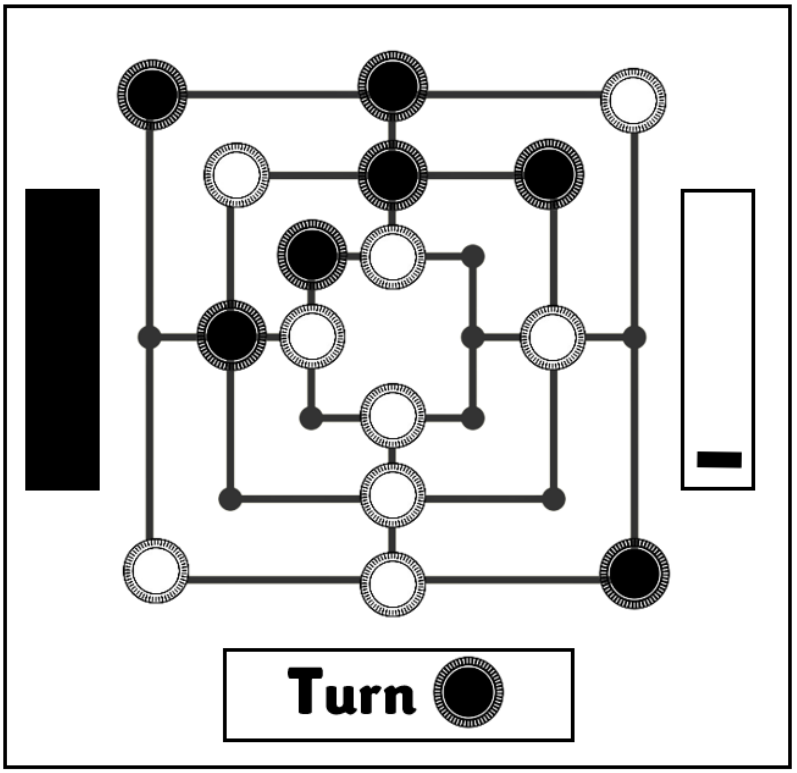
4. Removing tokens



Player 1 (White) places 3 tokens in a row, forming a “Mill” (tokens have to be either horizontally or vertical placed, not diagonally) - this gives the player a game advantage.

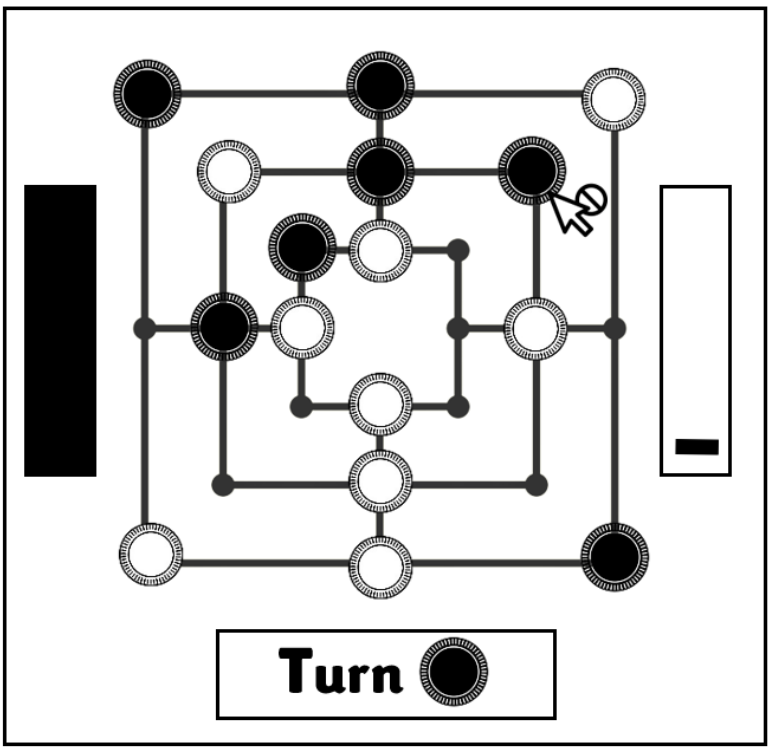


Player 1 (White) is now able to select an opponent's token of his/her choice to remove - (token cannot be part of a “Mill” on the opponent’s side unless all of opponent’s tokens are)



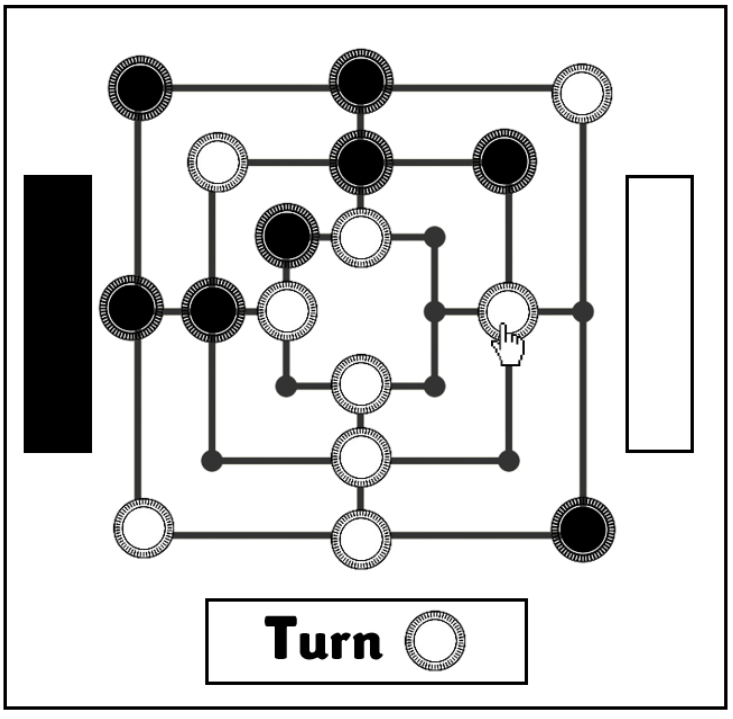
Player 2’s turn (Black) after a black token has been removed from the game board.

5. Invalid moves

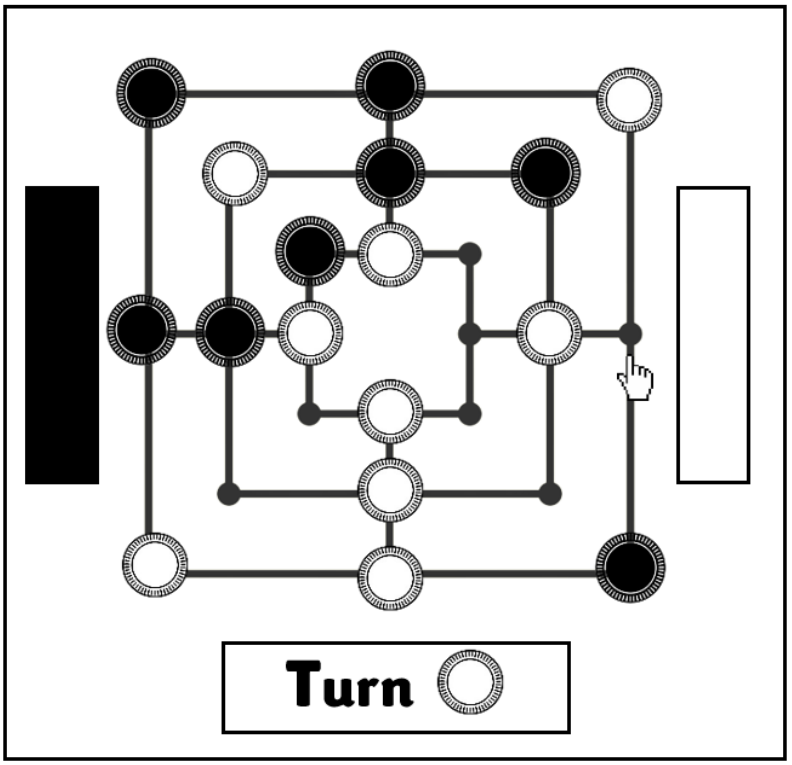


Player 2's turn (Black) where the player tries to place a token at the occupied location. This move violates the game rules and is indicated with an "invalid" icon next to the cursor.

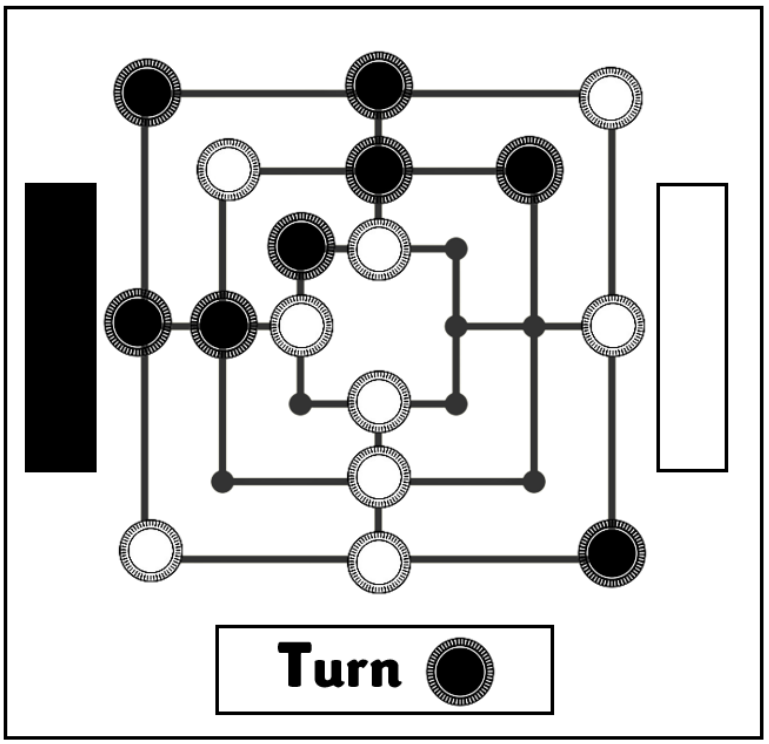
6. Moving Tokens



Mid-game: Player 1 (White) selects one of his/her own pieces to be moved.

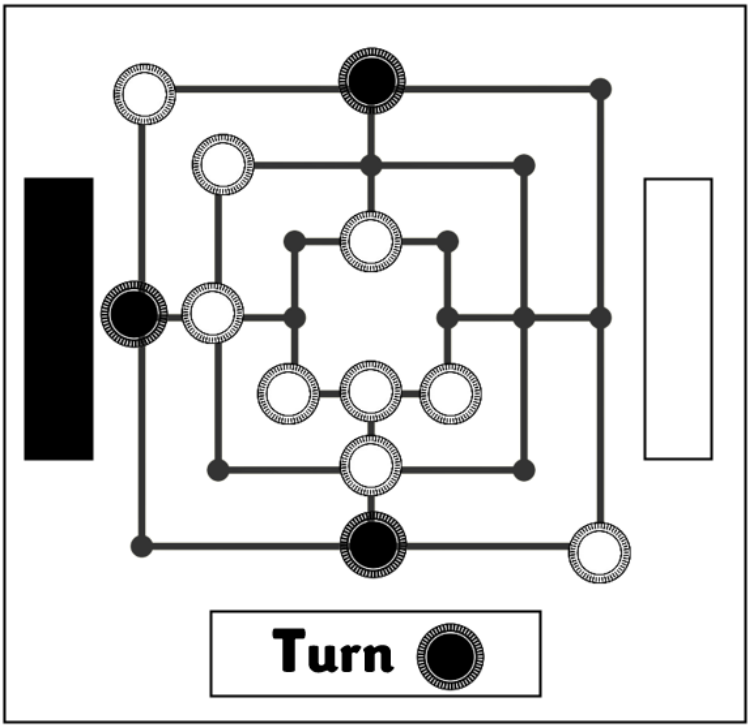


Player 1 (White), then clicks on one of the four empty positions connected to his current token to move it to the desired position.

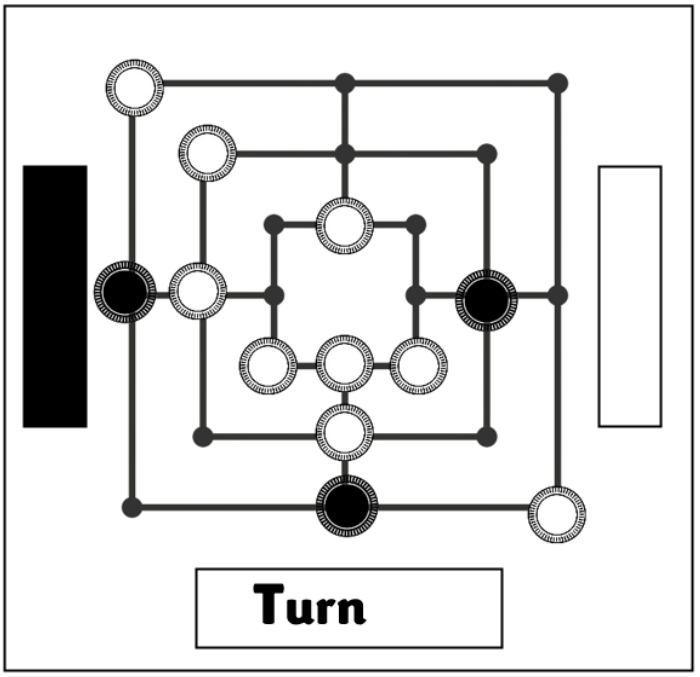


After the White piece has been moved, it goes back to the next player’s turn (Black) and the game continues.

7. Jumping / Flying

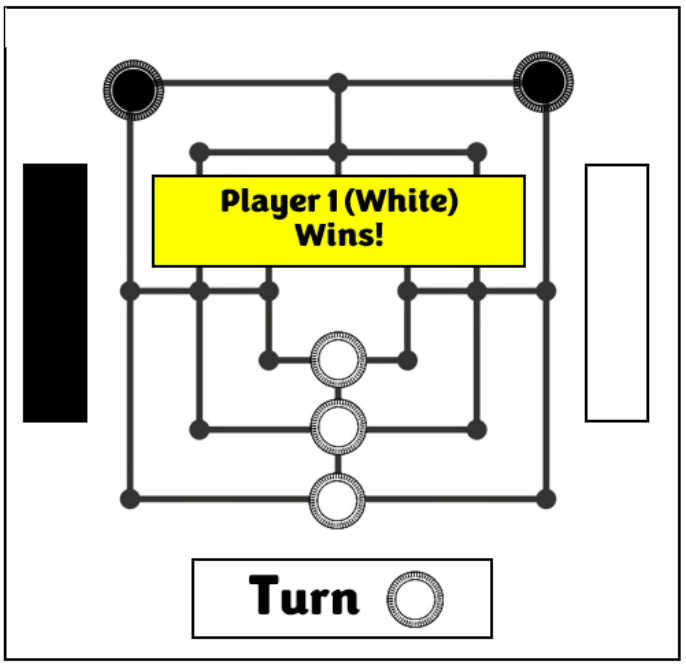


When a player has been reduced to his/her last 3 pieces, a game advantage is given to the Player to “fly” or “hop” a piece to any vacant position.



In this instance - Player 2 (Black), hops his/her piece from the middle node on the top row to the middle node on the middle row (left side), since he/she only has 3 pieces left.

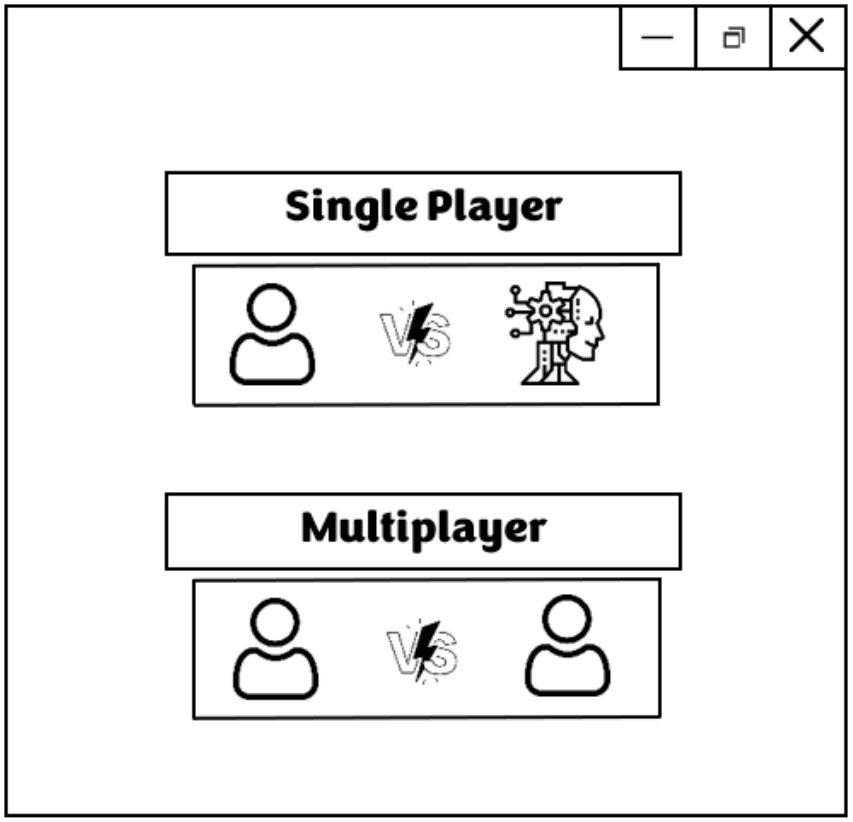
8. Game Victory Screen



Once a Player is reduced to two pieces (or has no legal moves), the game is over and the other Player has won. A victory screen is shown displaying the Winning Player.
The player may now quit the game to reset the game board.

Advanced Feature

For the advanced requirements, our team chose Advanced Requirements C:
“ A single player may play against the computer, where the computer will randomly play a move among all of the currently valid moves for the computer, or any other set of heuristics of your choice”



The player has the option to choose between “Single Player” (playing against the computer AI) or “Multiplayer” (playing against another human player) after hitting the “Start” button in the home screen menu. In “Single Player”, the rest of the gameplay remains the same, but after the player’s turn, the computer plays its turn instantly without a waiting period, it goes back to the player’s turn again and the game cycle continues until the game ends.