# FIT3077: Software engineering: Architecture and design

# S1 2023

## Monash University Malaysia


*Sprint One*

*Nine Men's Morris*

**Team The Three Tokens:**

Priyesh Nilash Patel 32182058

Rachel Ng Chew Ern 31424290

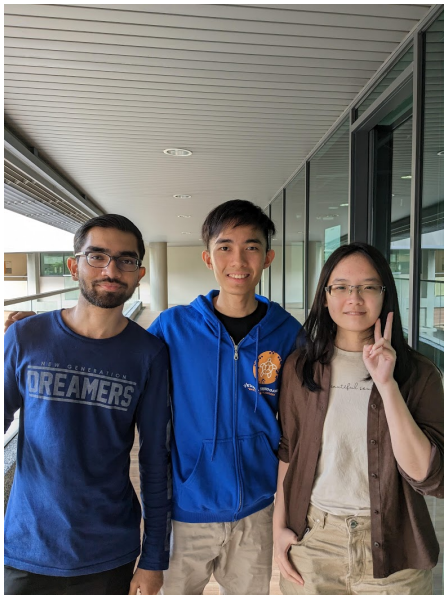
Hee Zhan Zhynn 31989403

# Table of Contents

# Team Information

---

# The Three Tokens




## Team Introduction

We are excited to introduce our team, **The Three Tokens**. The team name is a direct inspiration from our current project, the Nine Men's Morris where three tokens are needed to form a mill to remove enemy pieces. As the game progresses, the last three tokens become incredibly valuable and gain additional power. Losing any of the last three tokens will end the game in defeat. With this in mind, we aim to bring the same level of skill and uniqueness of each member to this project. Our team consists of 3 members, as pictured above. From left to right, we have Priyesh, Zhan Zhynn and Rachel, all of whom are third year students at Monash University. A brief description of their individual strengths and a fun fact about themselves is elaborated below.

## Team Membership

| **Priyesh Nilash Patel** (Pnil0001@student.monash.edu) | |
|---|---|
| Strength 1 | I am always inquisitive. |
| Strength 2 | I have good communication skills. |
| Strength 3 | I can speak 5 languages. |
| Fun fact | I am not a morning person. |

| **Hee Zhan Zhynn** (zhee0035@student.monash.edu) | |
|---|---|
| Strength 1 | I am a team player. |
| Strength 2 | I am good at time management. |
| Strength 3 | I can work under pressure. |
| Fun Fact | I have a pet dog. |

| **Rachel Ng Chew Ern (**rngg0013@student.monash.edu) | |
|---|---|
| Strength 1 | I am goal oriented. |
| Strength 2 | I have experience in Java, Python, HTML, CSS, Javascript, Typescript, C and SQL. |
| Strength 3 | I am good at technical writing e.g. drafting user guides. |
| Fun Fact | I may roll my eyes at lame puns but I secretly enjoy them. |

## Team Schedule

**Team Meeting Schedule**

1-hour meeting scheduled on every Thursday night, 10pm to 11pm, subjecting to urgency and number of issues pending discussion.

| Past Meetings | Summary |
|---|---|
| 27/03/2023 | Everyone came up with suggestions on which user stories to include and selected the final ones which will define our project scope. |
| 30/03/2023 | Discussion on which domain entities to include in the domain model and the relationships between them. Also discussed the preliminary low-fi design and which interactions to depict. Worked on the model together on LucidChart. |
| 01/04/2023 | Short meeting to review each other's work and provide feedback. |
| 02/04/2023 | Final pre-submission meeting to go over everything and put finishing touches on deliverables. Discussed justifications and made final updates to everything. |

| **Workload Distribution** | |
|---|---|
| Design | ● Everyone is involved in the brainstorming session.<br>● Design is iterated upon in LucidChart and Figma until we reach the final design.<br>● Everyone takes turns updating the design when there are any changes or extensions. |
| Implementation | ● Stories are divided into subtasks and posted on Trello.<br>● Effort estimation meeting to determine the number of man hours each task will take.<br>● Each member takes up tasks based on area of expertise, such that total hours spent are roughly equal.<br>● Weekly meetings to gauge everyone's progress and redistribute tasks if effort estimation was inaccurate.<br>● Progress can also be monitored on Trello. |

## Tech Stack

| | |
|---|---|
| Java | ● Priyesh and Rachel used it in their internships.<br>● Native support for object-oriented programming.<br>● Static typing for a more organised structure<br>● Long-established language, robust and ease of use<br>● Supports multiple platforms e.g. Windows, macOS<br><br>Discarded alternative:<br><br>● Java is more intuitive than Python for object-oriented programming.<br>● The team lacks experience in other modern alternatives to Java such as C#, as it it not taught in the Monash syllabus |
| JavaFX | ● Easy to learn and use<br>● Abundant online resources for game development<br>● Modern replacement for Java Swing<br>● Compatible and cross-platform Java GUI<br>● Specifically designed for building rich graphical user interfaces especially for games.<br>● Can be styled with CSS. Java Swing needs a third-party source for CSS.<br>● Enables the developers to write clear, manageable code in Java, which is easy to update or debug. Writing code for desktop applications in JavaScript is far more complicated, and cross-platform compatibility of .NET apps is inconsistent<br><br>Discarded alternative:<br>● ReactJS is another good alternative for front end design as it also supports Javascript and CSS. However, it is not specifically designed for game UIs and may not provide the necessary tools required for game development. |
| GitLab | ● Version control to make it easier to manage code.<br>● Allows collaborative software development work among the team members.<br><br>Discarded alternative:<br>● No alternatives were considered as the project repository was set up and provided by Monash. |
| IntelliJ IDEA | ● Free and open source<br>● Widely used in the industry<br>● Team members' familiarity with using IntelliJ<br>● Provides extensive support for Java-related projects such as easy debugging, code completion, formatting etc.<br><br>Discarded alternative:<br>● VSCode was considered due to its flexibility and support for multiple languages, but its autocomplete and debugging features pale in comparison to IntelliJ. |

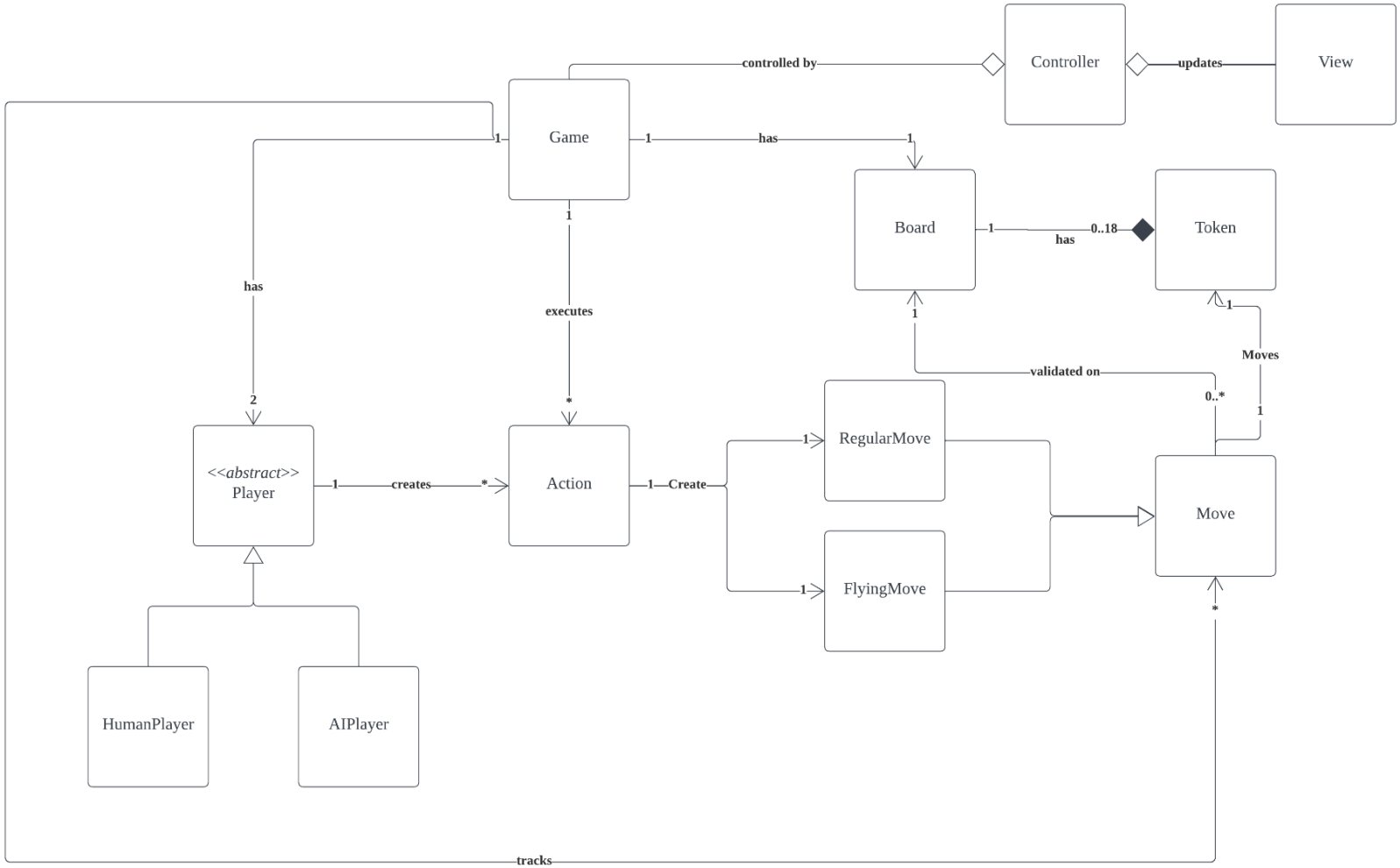
# User Stories

---

**The advance requirement chosen by our team is to develop a computer player for the game. (C)**

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.**

# Domain Model



## Justification

Player is an abstract parent class where there are 2 children classes under it at the moment. Generalisation is used to create a hierarchical relation between the classes. This is because both human players and computer players are types of players which will have similar functionality to a certain extent, the major difference lies in whether the move comes from user input or is automatically generated. This method also reduces redundant code and allows easier expansion if a new type of player is introduced to the game.

The Game class is the main class that has an association with the Player, Board and Action classes. It is responsible for initiating the game and handling interactions between the classes. One game has two players and one game board. The game has an aggregation relationship with the controller class, as the controller class can exist meaningfully without the game class.

The action class is created by a player who wishes to move a token. There is an association between Player and action with a one to many relationships as one player can create many actions throughout the game. The current design allows us to generalise the action class into child classes if newer features are introduced in the future.

The Move class also has a parent-child relationship with RegularMove and FlyingMove classes. As of now, these are the 2 different types of moves that a player can execute using the action class. An alternative approach that was considered was to have 2 children action classes which inherit from the action class to execute the action from the player as well as move the token. However, this would violate the single responsibility principle as one class is executing the player's action as well as moving the token. Move class also has an association relationship with the Board class as when a move is executed the board class needs to be updated with the new positions. The Game class has an association with the Move class to keep track of the moves made by the players. Finally, all moves from the player's action will be validated in the Move class.

The Board class has a composition relationship with the token class, this is because if the board ceases to exist, the tokens become obsolete and cannot function. An alternative approach of introducing a Position class was considered, though it was decided to be unnecessary as the positions of the tokens can be kept in a list to represent the row and column coordinates.

The Controller class has an aggregation relationship with the Game and View Class. The Controller class exists to direct Game requests to update the View Class. Therefore it is able to exist without the Game Class. The View Class is to update the user interface to allow users to interact with the game.

# Low-Fi Design

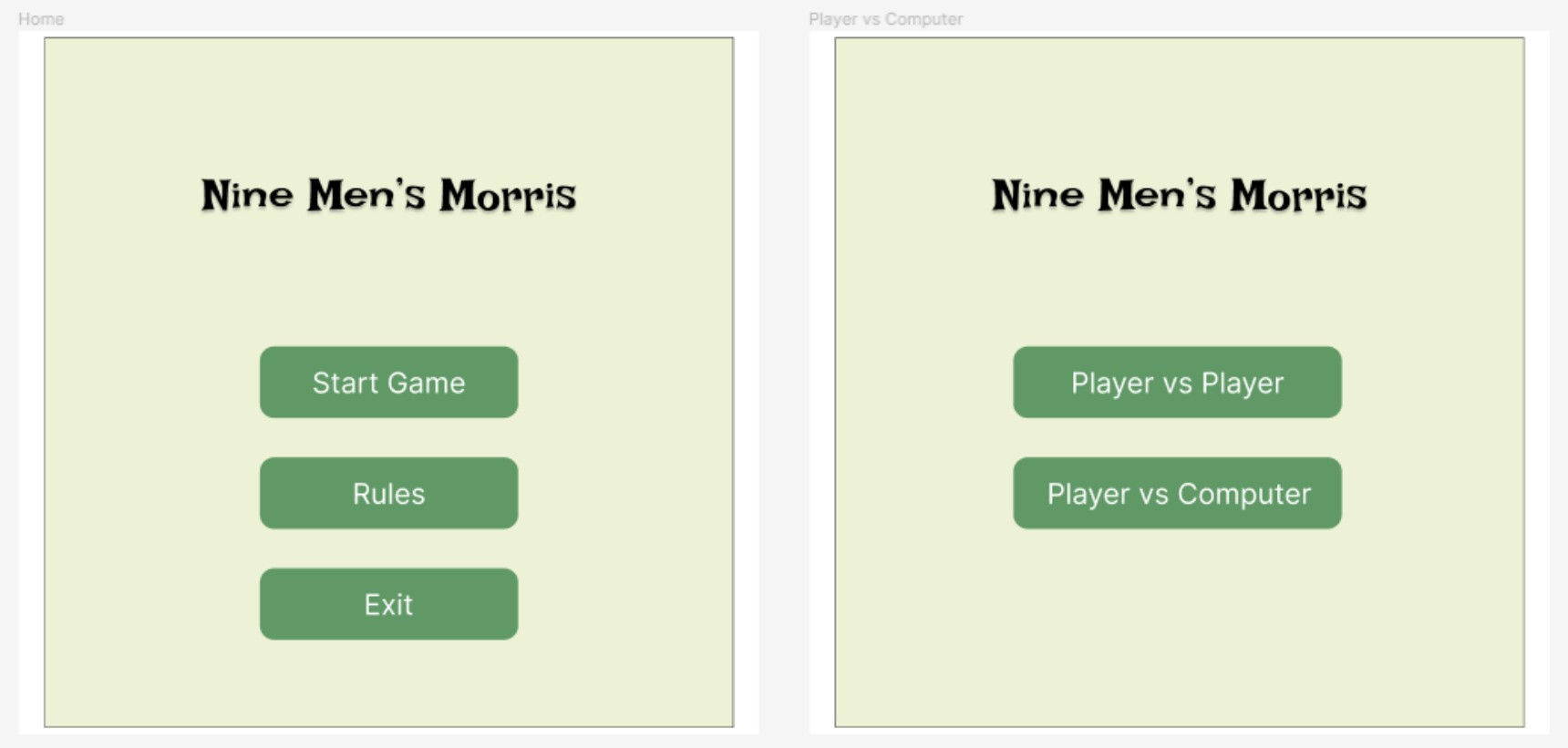



Figure 1a: Home page and advance requirement

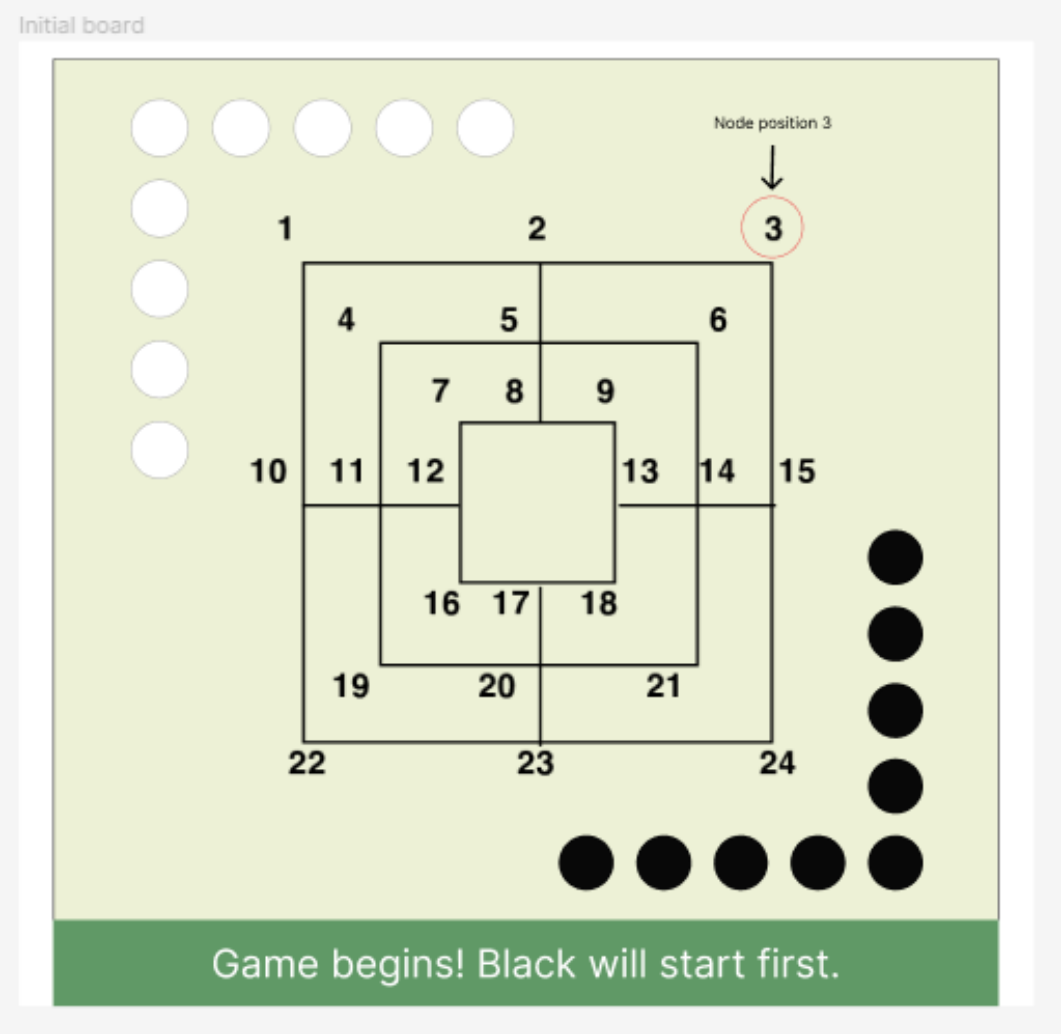



Figure 1b: Initial board

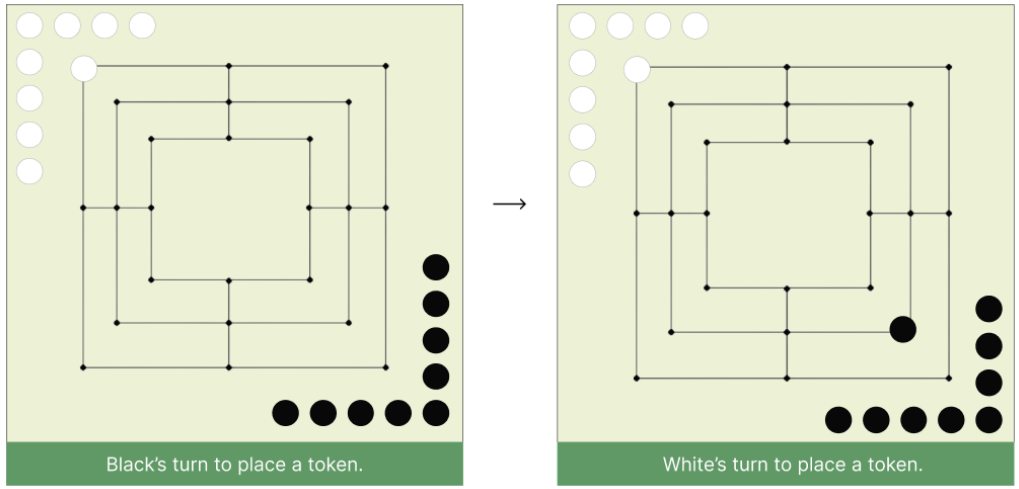



Figure 2: Design for placing token interaction

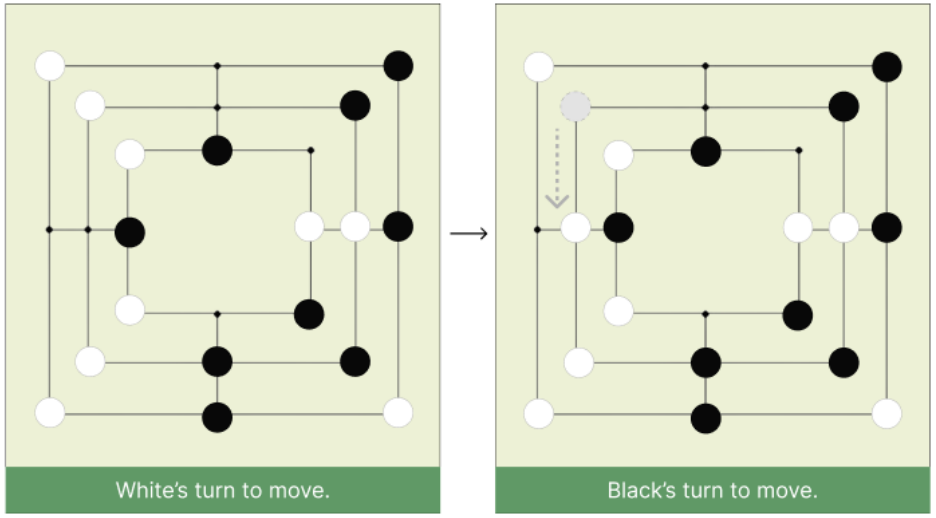



Figure 3: Design for moving token

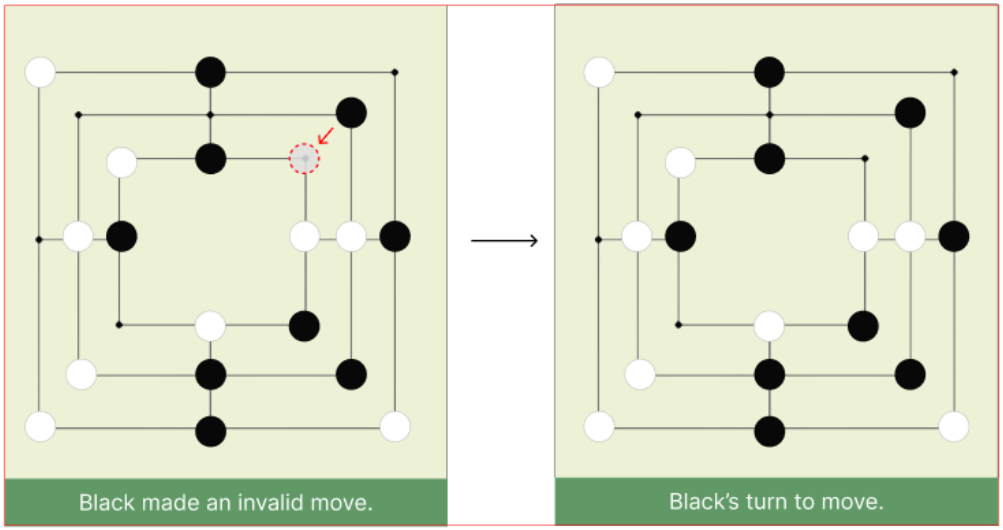



Figure 4: Design when an invalid move is made

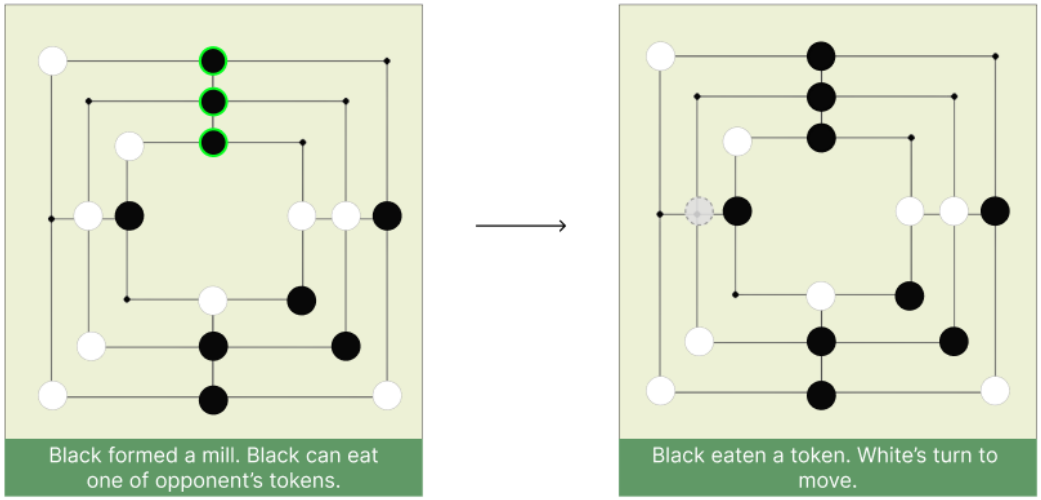



Figure 5: Forming a mill and eating an opponent's man

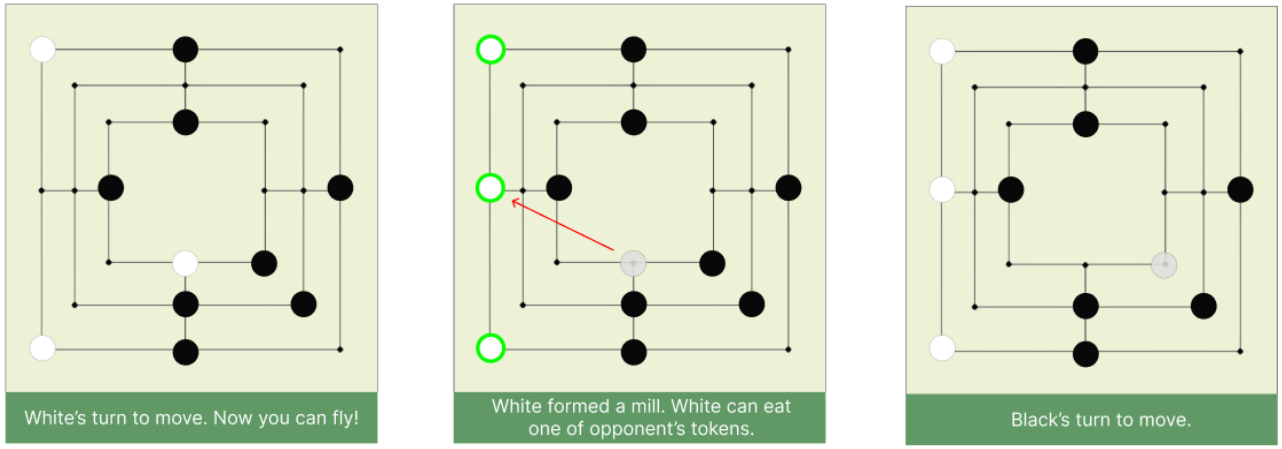



Figure 6: Flying move

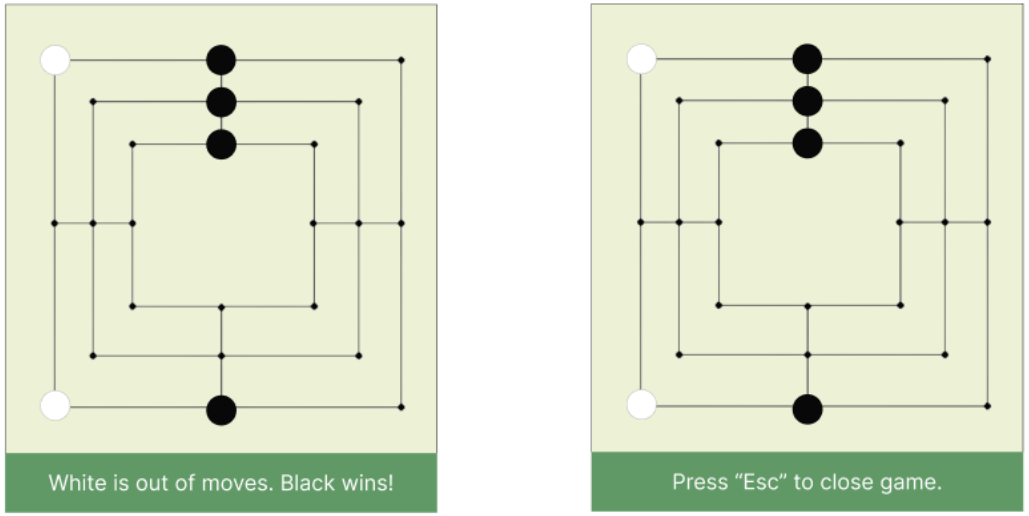



Figure 7: Wining and closing the game