MONASH University

ENGINEERING

SPRINT 1

TABLE OF CONTENTS

1 Introduction

The project’s goal is to create an object-oriented design for a game named Nine Men's Morris that will be fully functional implemented in the last sprint. Our team, The Brewing Java will be asked to advance this game’s development and demonstrate that the stated Sprint milestones are met in each Sprint. In early sprints, we will concentrate on the basic elements of the game, such as User stories, Basic Architecture and UI Design, Tech Stack and Rationales. During the final sprint, we will be asked to expand our implementation with new features.

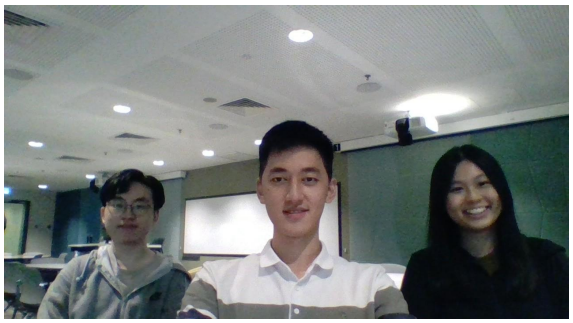
In this document, we will first introduce the team members, with the team information provided, such as team name, photo, team membership and team schedule. Then, we will cover the programming languages, APIs and technologies we plan to use and provide justification on our final decision. We will create a list of user stories that covers both basic Nine Men's Morris gameplay and chosen advanced requirements by applying all aspects of the INVEST acronym. For the advanced requirement, we chose the first one shown in Sprint one brief, which is to implement a tutorial mode

We will also highlight the basic architecture of our project which includes domain model and rationale for each chosen domain. Moreover, low-fidelity prototype drawings of the proposed interface for the game are drawn. An appendix with all the links attached to the documents such as Work Breakdown Structure, Domain Model and Prototypes. With that, it concludes our report on sprint one deliverables.

2 TEAM INFORMATION

1. Team Name and Team Photo

Team Name: The Brewing Java
Team Photo:



2. Team Membership

Name	Email	Technical & Professional Strengths	Fun Fact
Fong Zhiwei	zfon0005@student.monash.edu	Object-oriented design	Night Owl
Kennedy Tan	ktan0087@student.monash.edu	Software development methodologies	Plays futsal
Soh Meng Jienq	msoh0007@student.monash.edu	UX design	Plays piano

3. Team Schedule

Our team decided to have a regular meeting every Saturday 10am - 12pm. A discussion will be made in WhatsApp group to confirm the agendas every Friday. The meeting will be held on Discord and one of the members will be responsible to note down a meeting minute. It contains the meeting content and it will be very useful if any of the members is not available for the meeting and for members to check on the details in the meeting.

For our team, we created a Work Breakdown Structure (WBS) which is attached in the appendix below this document. Project was broken down into smaller, manageable tasks. All members are required to update the WBS while they are working on this project, so that we can monitor each other's progress. We found this way is easier for us to manage and plan the project as workload was distributed evenly. If any of the members falls behind on his/her tasks, we will discuss among ourselves and help on the member's task accordingly.

4. Technology Stack and Justification

Python is considered as one of the options initially, as all of the members in our team have learnt Python before but due to the slow speed and high memory usage, it is not a suitable frontend programming language. Python performs generally slower than other programming languages, such as C++, Swift and Java. In addition, it has insufficient memory and slow processing power, thus it is more suitable to implement a server-side program. However, our project is planned to be a desktop application. Besides that, it is harder to apply OOP in Python, compared with Java. Therefore, we decided to not use it as our programming language.

Java is also an Object-Oriented programming (OOP) language that facilitates the development of applications on any platform. With OOP, a huge problem is broken down into solvable or manageable parts. During troubleshooting, we know which class to check on when there is an error instead of going line-by-line through all the code. Above all, OOP increases code reusability through inheritance, so that we do not have to repeat the same code multiple times, but instead create a child class that inherits the methods of the parent class. Hence, we decided to utilise Java for this project for the backend development of our application. Adding on to this, as all our team members have the knowledge and experience on Java from previous units taken, so we all agreed on the decision made.

For graphical user interfaces (GUI), we will be using Java Swing API. We compared the Swing API and Java Abstract Window Toolkit (AWT) API. We found out that Java Swing components are lightweight and powerful while Java AWT components are heavyweight as its components depend on Operating System. Thus, Java Swing has more functionality in contrast with Java AWT. For instance, Swing provides a richer set of high-level components such as tree, tabbed panes and list box.

In consequence, we chose Java and Swing API for our application to reach our goal of this project, which is demonstrating how we properly apply object-oriented design.

3 USER STORIES

We come out with 16 user stories for this project, that covers both basic Nine Men's Morris gameplay and chosen advanced requirements based on the regular template, “As a (role), I want to (desire action/reaction), so that (benefit).” In the process of creating user stories, we make sure that all our user stories are not redundant and fulfil all aspects of INVEST criteria.

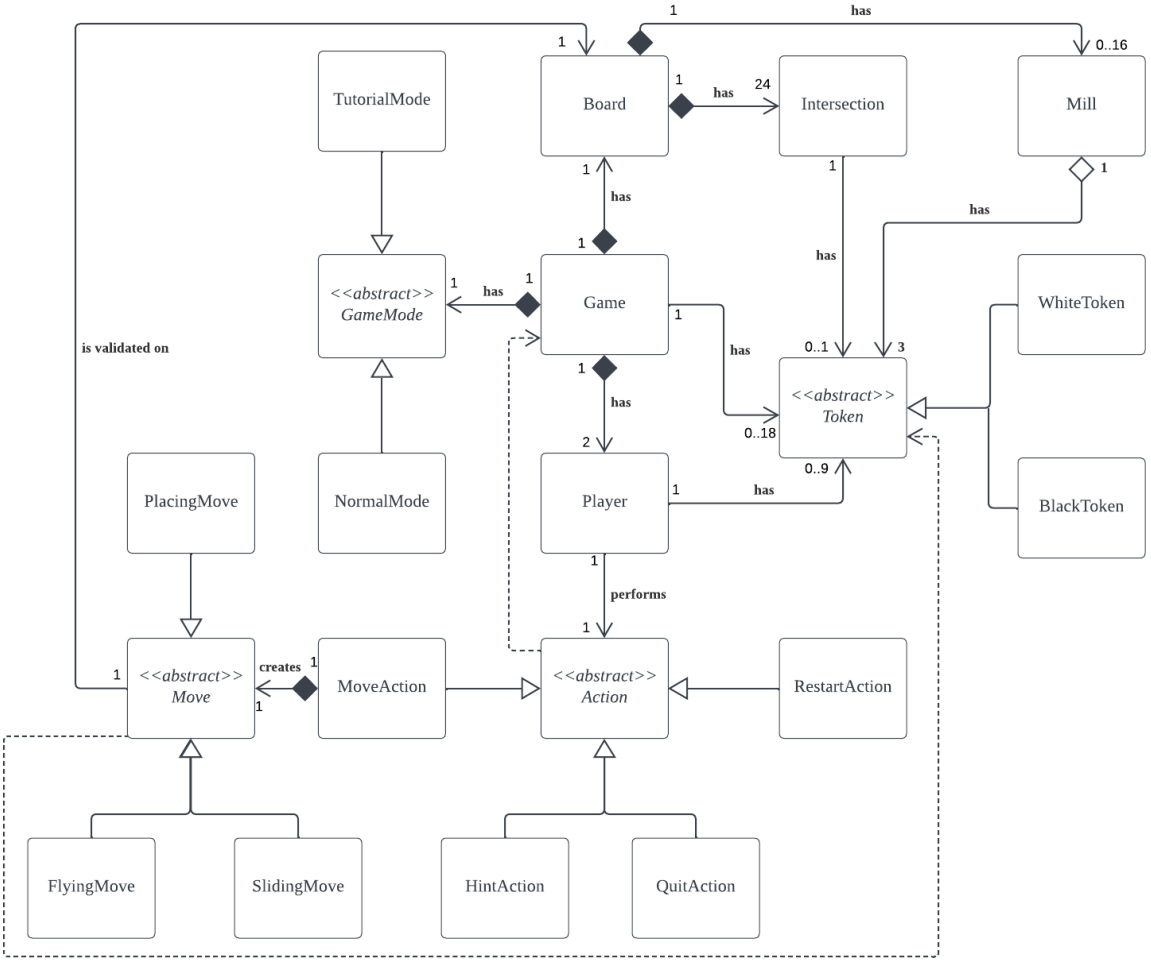
User Stories (Basic and advanced requirements)

- As a visitor, I want to have a tutorial mode in the game, so that I can learn how to play the game.
- As a user, I want to have hints of all legal moves, so that I know where to place the next piece.
- As a user, I want to be unable to make an illegal move, so that I cannot cheat in the game.
- As a user, I want my piece to have a different design than my opponent’s piece, so that I can differentiate my piece from my opponent’s piece.
- As a user, I want to be able to remove one of my opponent’s pieces when a mill is achieved, so that I can increase my winning possibility.
- As a user, I want to move my piece to an adjacent point after 18 pieces have been played, so that I can make a mill.
- As a user, I want to know the result of the game, so that I know who wins the game or it is a draw.
- As a user, I want to know the amount of remaining pieces, so that I can plan my future moves easily.
- As a user, I want to place one piece at a time onto the board, so that the players can take turns to play all their pieces.
- As a user, I want to be able to place my piece only on an empty place, so that it is only possible to only have a single piece in one spot on the board.
- As a user, I want my opponent to not remove any pieces that I formed as a mill, so that I can protect my mills on the board.
- As a user, I want to be able to move my piece to any empty intersection without the limitation of only moving to an adjacent dot after I have been reduced down to the last 3 pieces, so that it can increase my winning possibility.
- As a user, I want the game to end when either of the players has no possible move or has reduced to only 2 pieces on the board, so that I can quickly start a new match.
- As a user, I want to have 2 players in a single game so that I can play the game with an opponent.
- As a user, I want to be able to restart the game, so that I can start a new match anytime.
- As a user, I want to be able to exit the game, so that I can stop playing at any time.

4 BASIC ARCHITECTURE

Domain Model

We come out with a domain model that covers both Nine Men's Morris gameplay and the chosen advanced requirements. It is shown below.



Rationale

GameMode is a composition of Game. Abstract class GameMode is a child class of a parent class Game. Every Game of Nine Men's Morris has a GameMode and GameMode will not exist if Game does not exist. GameMode is an abstract class because there will be multiple child classes that inherit abstract class GameMode into specific different game modes.

NormalMode extends GameMode, **TutorialMode extends GameMode.** NormalMode and TutorialMode inherit GameMode abstract class as they are specific game modes for the Game class. This relationship satisfies the **Open-closed principl**e of the SOLID principle as the GameMode parent class is able to be extended into multiple child classes without modification to the parent class. This also satisfies the **Don't Repeat Yourself** (DRY) principle. For instance, if we are adding more modes in the future, we can simply extend them from GameMode, instead of repeating the code that is similar in every class.

Board is a composition of Game. Board is a child class of a parent class Game. Every Game of Nine Men's Morris has a single Board and Board will not exist if Game does not exist. Board is where the game of Nine Men's Morris will be played.

Intersection is a composition of Board. Intersection is a child class of parent class board. The Board in every game of Nine Men's Morris has 24 Intersections and these Intersections cannot exist if there is no Board. Each Intersection acts as a place where the player can place their tokens.

Player is a composition of Game. Player is a child class of a parent class Game. Every Game of Nine Men's Morris has 2 players and players will not exist if there is no game. 2 players play the game of Nine Men's Morris by taking turns to perform actions.

Player---<<has>>--->Token and Game---<<has>>--->Token. Each Player has 9 objects of Token class at the start of the game and it may decrease as the game of Nine Men's Morris goes on. Each Game starts with all its Intersections of the Board as empty and it can be filled with 18 objects of Token class maximum if both Players placed all their Token objects.

Intersection---<<has>>--->Token. Every Intersection on the board is empty when the game of Nine Men's Morris starts and each Intersection can be filled with a single object of Token class.

WhiteToken extends Token, **BlackToken extends Token.** WhiteToken and BlackToken inherit the Token abstract class. WhiteToken and BlackToken inherit all the methods and attributes from the Token abstract class. The only difference between WhiteToken and BlackToken is their colour to differentiate between them as they are held by different Players. This relationship satisfies the **Open-closed principle** of the SOLID principle as the Token parent class is able to be extended into multiple child classes without modification to the parent class. This also satisfies the **Don't Repeat Yourself** (DRY) principle.

Token has an aggregation relationship with Mill. A Mill is formed by having 3 objects of Token class from the same owner (e.g. 3 WhiteTokens or 3 BlackTokens) placed in a certain

Intersections of the Board. A Mill has to have 3 objects of Token class but objects of Token class can exist without Mill class, thus they have an aggregation relationship.

Mill is a composition of Board. Mill is a child class of a parent class Board. Every Board can have 0 to 16 Mills and the Mills cannot exist if there is no Board as there will be no possible way to form them.

Player---<<performs>>--->Action. Player creates an object of the Action abstract class to perform various actions when it is his turn. Players can use actions to perform actions onto the game such as restarting the game, quitting the game and turning on hints. Players can also use actions to perform moves onto the board and doing this will end the player's turn.

RestartAction extends Action, **QuitAction extends Action,**
HintAction extends Action, MoveAction extends Action.
These 4 types of Actions are concrete classes that extends the abstract class Action, they inherit the attributes and methods of Action but they each have their own specialised methods or attributes to help me affect the Game in a different way, RestartAction will restart the Game into a new Game, QuitAction will stop and exit the current Game, HintAction can be toggled to show the valid moves that the Player can perform, MoveAction will allow the Player to affect the Board. This relationship satisfies the **Open-closed principle** of the SOLID principle as the Action parent class is able to be extended into multiple child classes without modification to the parent class. This also satisfies the **Single Responsibility Principle** which defines that a class should only have one job. Instead of having all the methods such as restart, quit, hint and move in the Action class (God class), we create classes for each action ensuring that each class is only responsible for a single part of the game's functionality.

Move is a composition of MoveAction. Abstract class Move is a child class of a parent class Move. Every MoveAction has an object of Move abstract class and the Move cannot exist if there is no MoveAction so there is no possible way to move affect the Board without a MoveAction. Move is used to affect the status of the board.

Move---<<is validated on>>--->Board. Any object of Move class will be validated on the Board to ensure it is a valid move based on Nine Men's Morris rules before it is able to affect the Board..

PlacingMove extends Move, **FlyingMove extends Move,**
SlidingMove extends Move. These 3 concrete classes extend the Move abstract class. PlacingMove is used by the Player to place a Token onto an empty intersection on the Board. FlyingMove is used by the Player to move a piece to any intersection if and only if they have less than 3 tokens left. SlidingMove is used to move a token to an adjacent intersection. This relationship satisfies the **Open-closed principle** of the SOLID principle as the Move parent class is able to be extended into multiple child classes without modification to the parent class. This also satisfies the **Don't Repeat Yourself** (DRY) principle.

Discarded Alternatives

Game---<<has>>--->Hint, Game---<<can>>--->Quit, Game---<<can>>--->Restart. We originally designed for Game to have an association relationship between Game with Hint,

Quit and Restart for it to have these functions. However, we think that these actions at its core are the same as the MoveAction the Player used to affect the Board. To satisfy the **Don't Repeat Yourself** (DRY) principle, we moved these classes to be a child class of an abstract Action class that covers all the actions that the Player can make.
Game---<<has>>--->GameMode. We originally designed for Game to have a direct association relationship with concrete class GameMode. However, we realised that this approach is not scalable and it violates the Single Responsibility Principle of SOLID principles. This is due to GameMode having to manage multiple types of game modes in its class if there is a need to increase the number of game modes. We decided to change GameMode into an abstract class where different game modes can directly inherit its main attributes and methods.

Game---<<has>>--->Player. We originally designed for Game to have an association relationship with the Player. However, we think that there is no meaningful way for a Player to exist if Game does not exist. Hence we designed Player as a composition of Game.

5 BASIC UI DESIGN

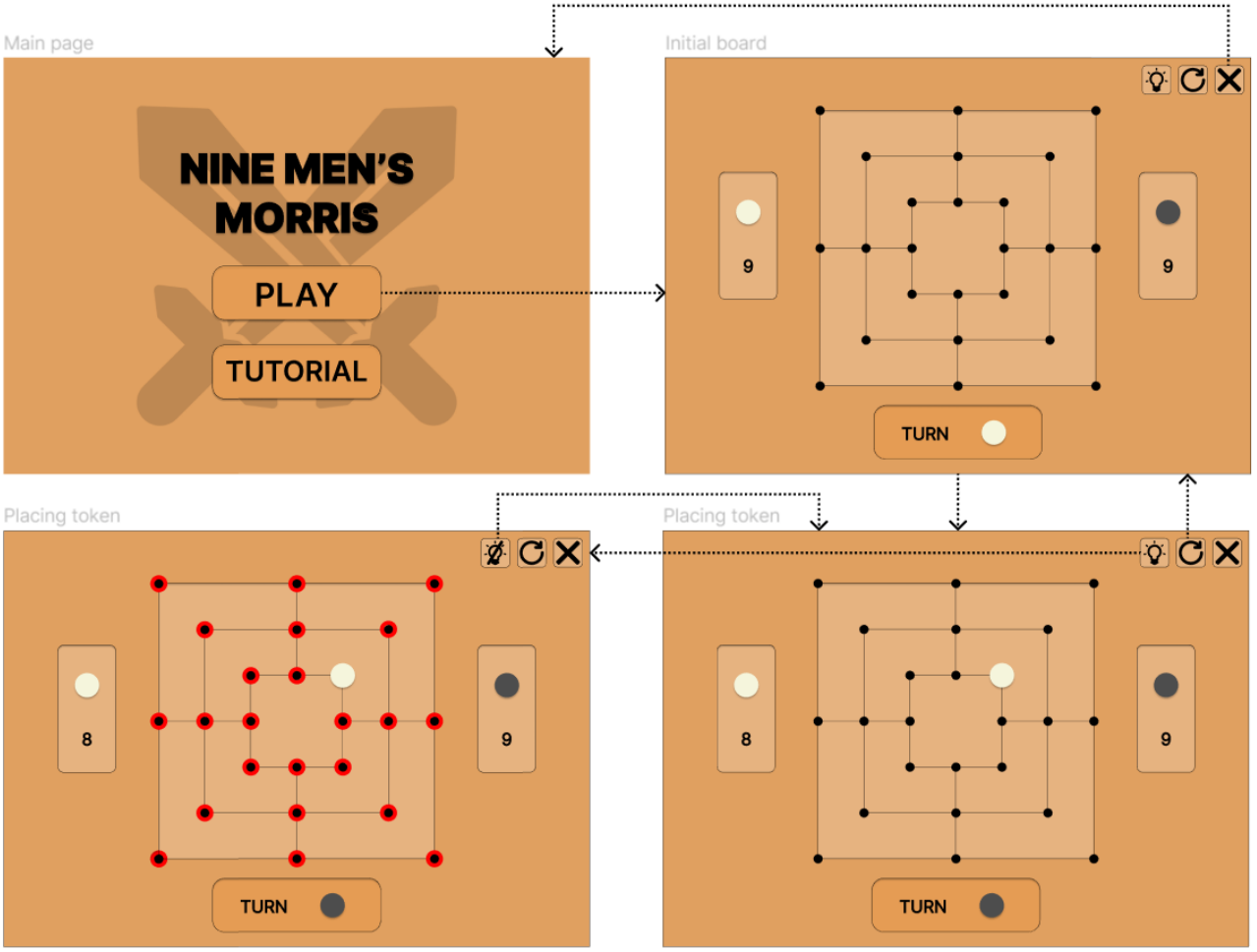
Low-Fidelity Prototype Drawings

There are two modes in our game, which are game and tutorial mode.

Game Mode

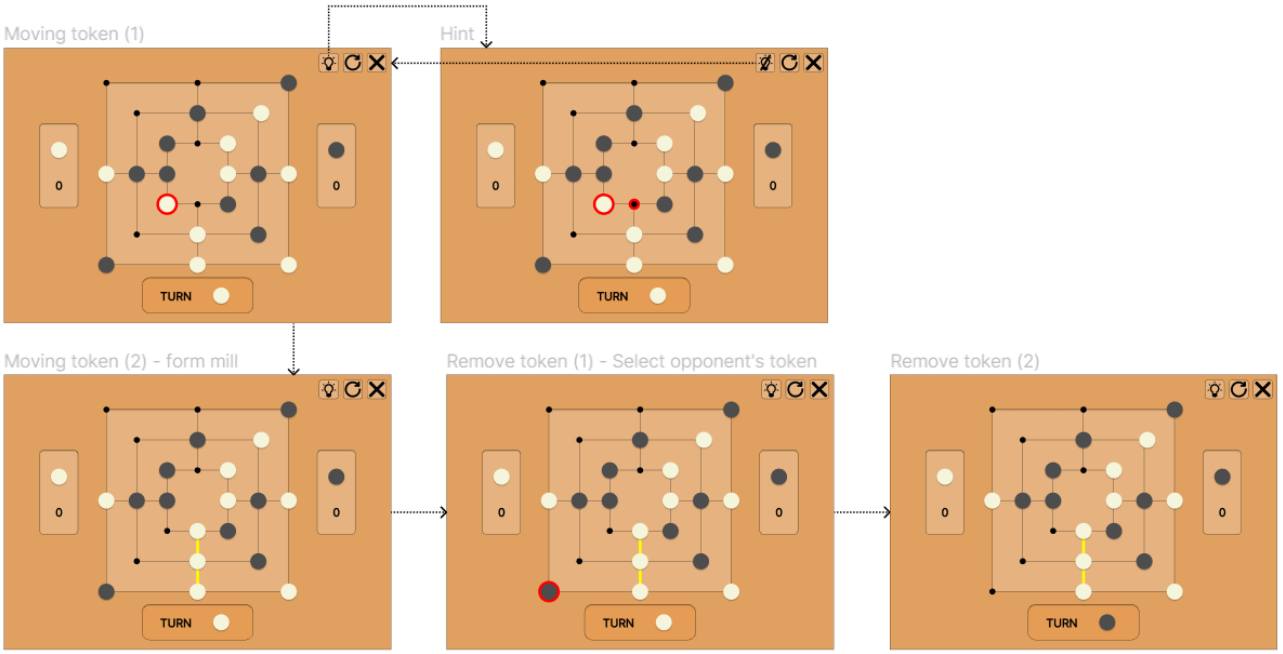
When players press “PLAY” to start a match, it will bring them to an initial board which consists of 24 intersections. Each player has 9 tokens in their respective colour, which are either black or white. The game always begins with a player holding white tokens, and they take turns to play.

On the top right corner, there are 3 buttons, which represent hint, restart and close. The hint button will display all the legal moves of the player. And, when the player presses the hint button again, the hints will be disabled. Restart button will bring them to the initial board to start a new game. Close button will lead them to the main page.

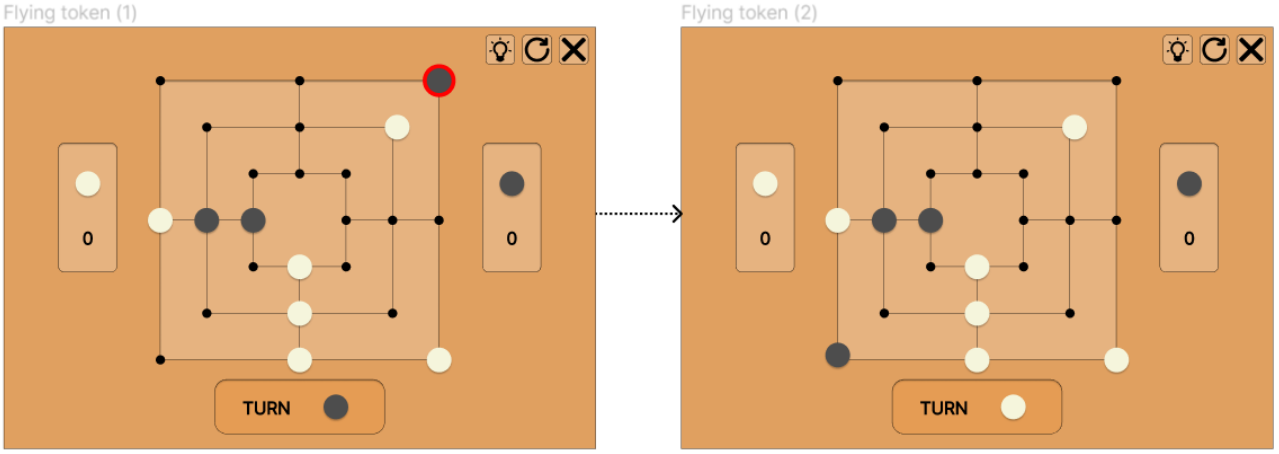


After both of the players place their 9 tokens, they take turns to select and move their token to an empty adjacent intersection. When a player forms a mill, which is 3 tokens in a line, he/she can remove one of the opponent’s tokens that is not in any mill.

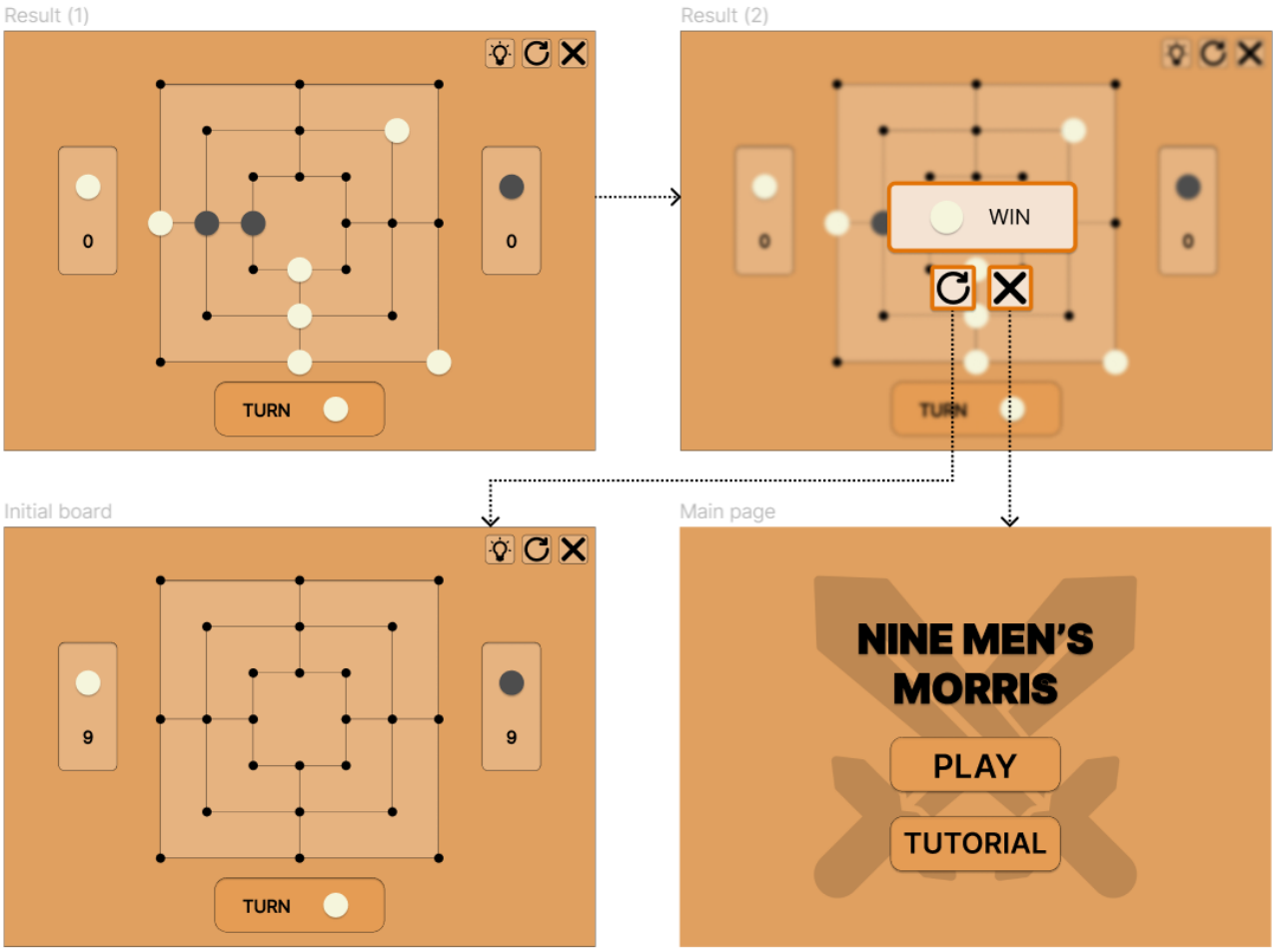
The hint button in the moving phase is slightly different, as it will require the player to select the token that he/she wants to move. And, the legal moves of the selected token will be shown in red circles. When the player presses the hint button again, the hints will also be disabled.



When the player only has 3 tokens left, the player can move his/her token to any empty intersection without the limitation of only moving to the adjacent point.

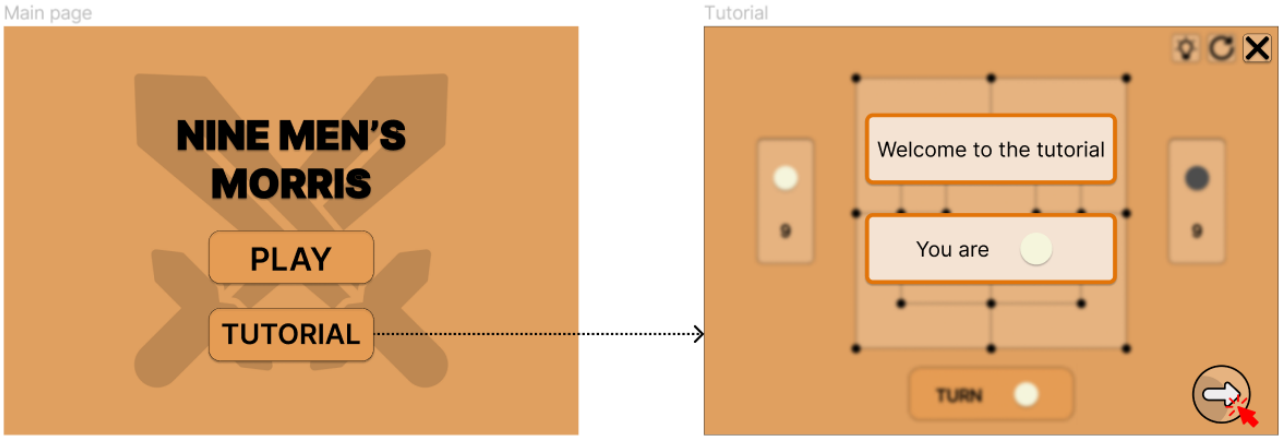


When one of the players has less than 3 tokens, the player loses the game. The result page will show the winner in his/her respective token's colour. There are two buttons in the result page, which are restart and close. The restart button will bring the players to the initial board to start a new game for them. While the close button will bring them to the main page to exit the game.

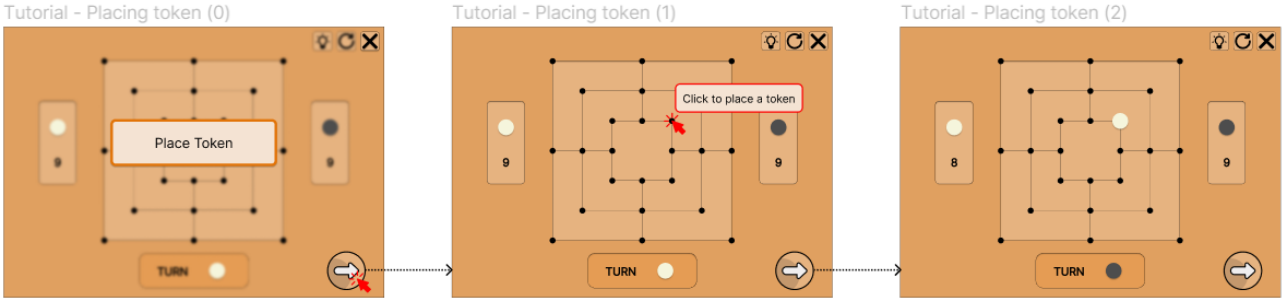


Tutorial Mode

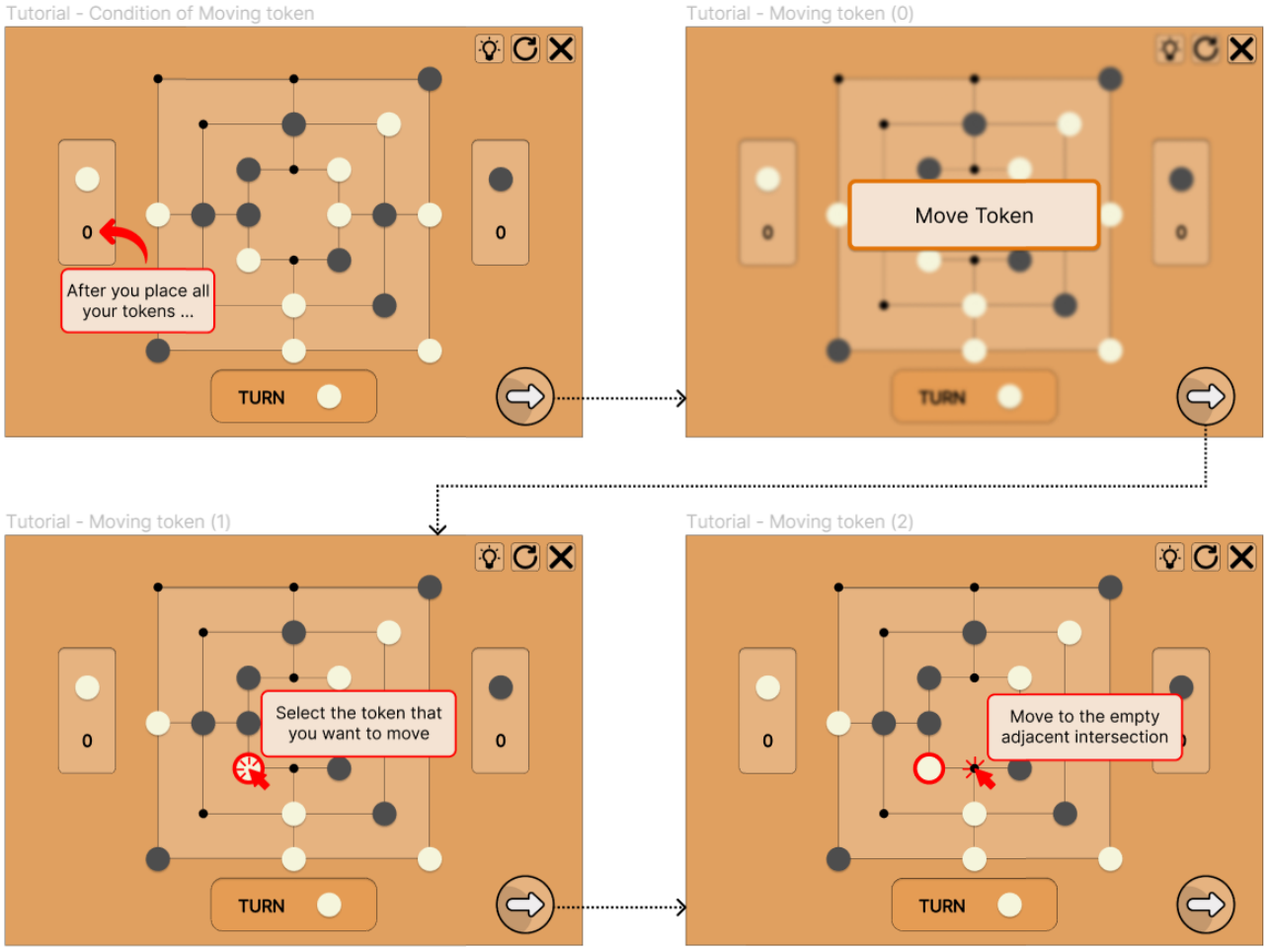
The tutorial will teach the player how to play the game. Once the player enters the tutorial, it will automatically be allocated as holding white tokens. All the pages in the tutorial have a next button at the bottom right corner, and the player is required to press it to continue the tutorial. There is also a close button at the top right corner of all the tutorial pages, so that the player is allowed to exit the tutorial anytime.



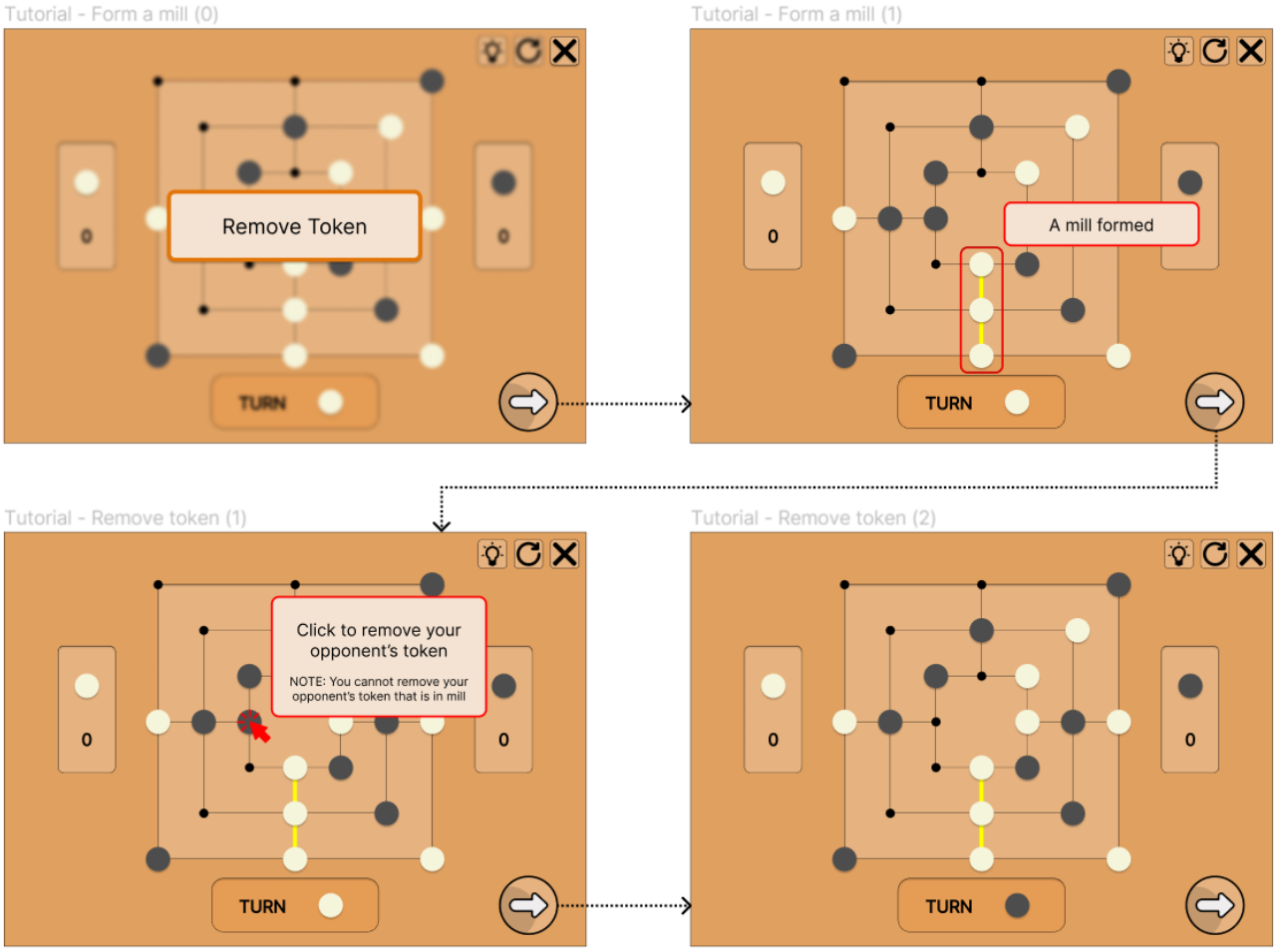
The tutorial will guide the player to place a token on the board. A red arrow will be using to guide the player to follow the tutorial with a supported message.



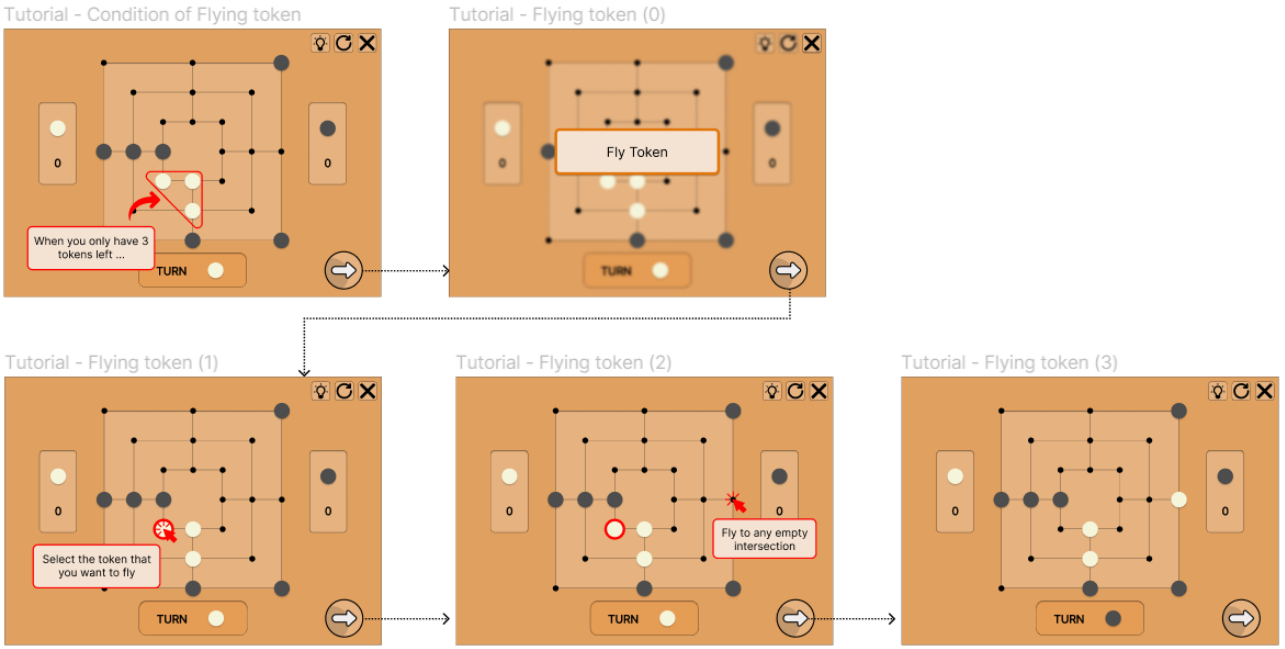
After teaching the player how to place a token, the next step is to teach the player to move his/her token.



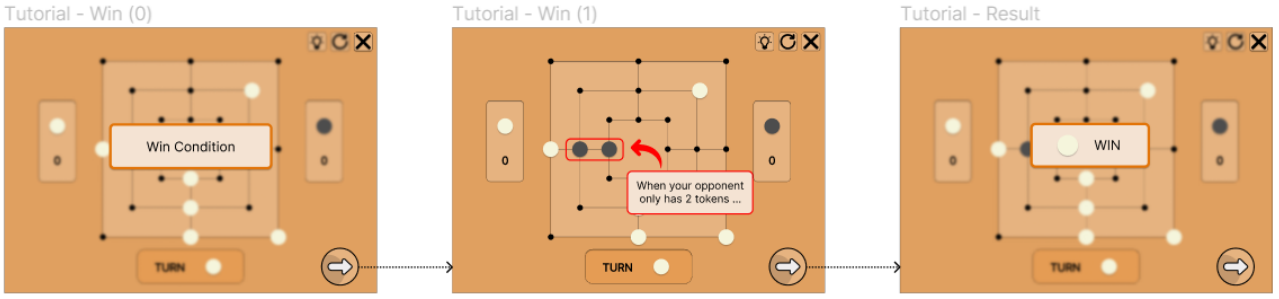
After the player moves his/her token, a mill is formed. The 3 tokens in line will be highlighted by a yellow line. And, the player will be guided to remove the opponent's token. A note is given to tell the player that it is not allowed to remove the token in mill.



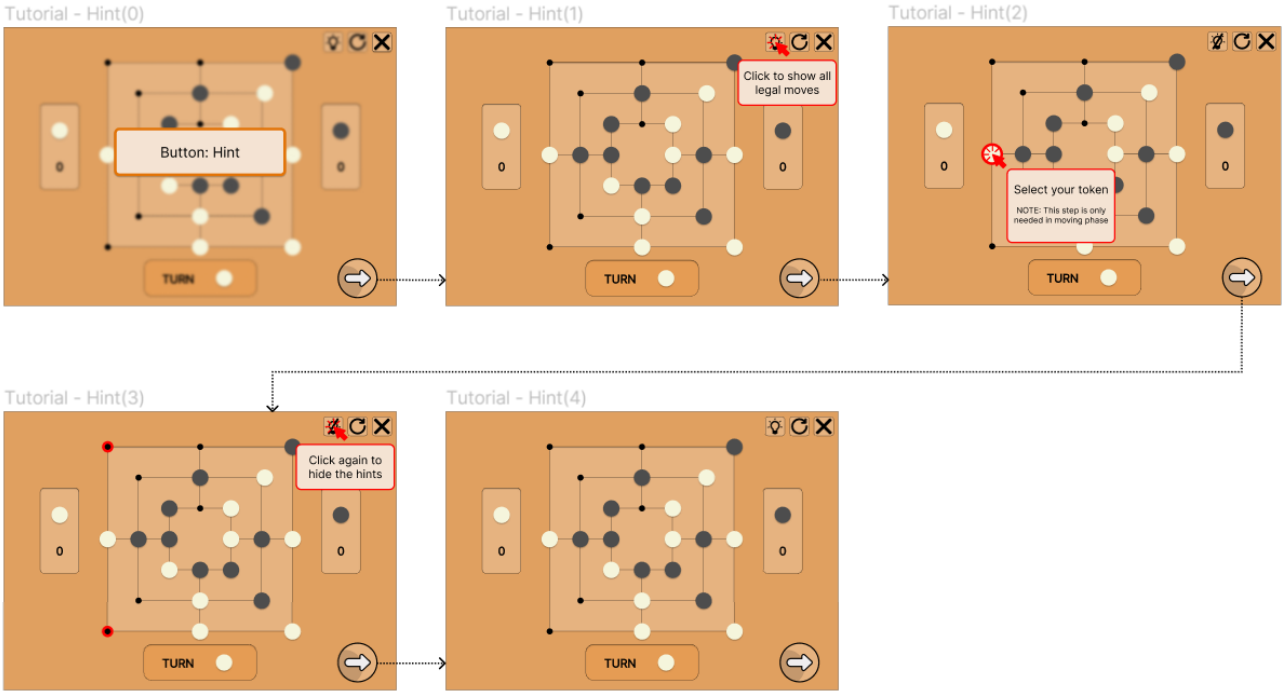
The tutorial will be followed by teaching the player how to fly his/her token when the player only has 3 tokens.



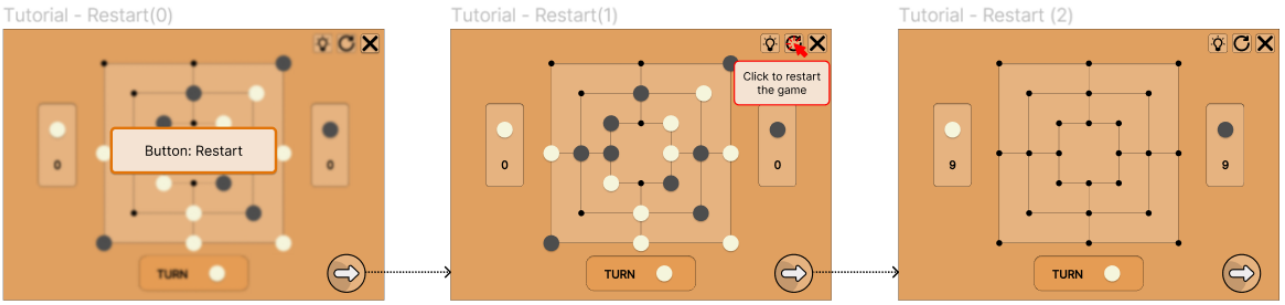
After the tutorial of flying tokens, it will tell the winning condition of nine men's morris, which is when the opponent has less than 2 tokens.



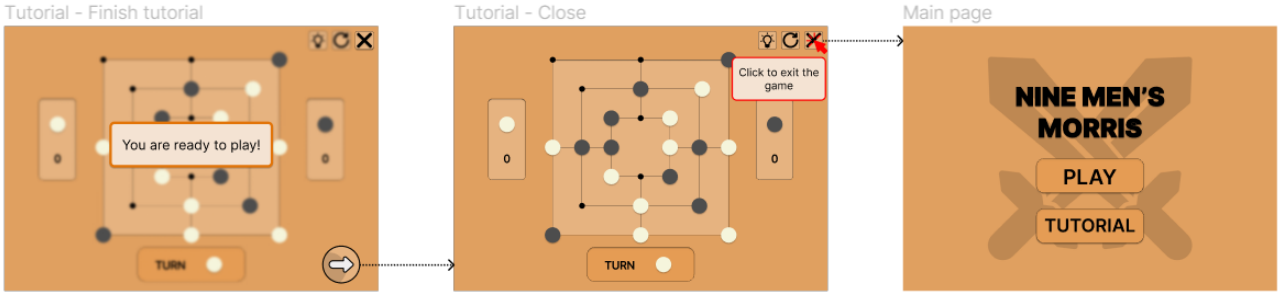
The function of the buttons will also be covered in the tutorial. It will start by teaching the hint button.



After that, the tutorial will teach about the restart button.



Lastly, the tutorial will wrap up by teaching the close button and bring the user back to the main page.



5 APPENDIX

Work Breakdown Structure

https://docs.google.com/spreadsheets/d/10B28lCSSwzXOpijzydfFUe32XY_a1BlKvHQG8Hlq6rk/edit?usp=sharing

Low-Fidelity Prototype Drawings

https://www.figma.com/file/bFVr5a2RvkB7s3ONMxl7Fv/UI-design?node-id=0%3A1&t=aLRQKU3xO6h7clQW-1

Domain Model

https://lucid.app/lucidchart/72bd3d5f-67f6-485c-8377-ad7802a7a456/edit?viewport_loc=-108%2C-155%2C2219%2C1097%2C0_0&invitationId=inv_ca1797a5-fe22-4640-8cd0-134bc8e9cfe4

Reference List

EDUCBA. (2019). *AWT vs Swing | Learn The Top 11 Most Valuable Comparisons*. [online] Available at: https://www.educba.com/awt-vs-swing/ [Accessed 2 Apr. 2023].