FIT 3077 Assignment (Team Information):
9 Men's Morris Game Application

Presented by:

Ong Chien Ming - 31861318
Syed Zubin Hafiz - 3227671
Garret Yong Shern Min - 31862616
Mario Susanto - 31103146

Team Information

1 Team Name & Photo

1.1 Team Name:

Stemming from the nature of the team which are new to the industry and on their way to find the proper road leading to their goals of reaching HD; our team name:

Team GreenPath

1.2 Team Photo:

2 Team Membership

2.1 List of Members:

1. Syed Zubin Hafiz - 3227671
2. Ong Chien Ming - 31861318
3. Garret Yong Shern Min - 31862616
4. Mario Susantoo - 31103146

2.2 Contact Information

Our team has agreed to use WhatsApp as the primary vector of communication. The following is each of our team member's contact details: -

1. Syed Zubin Hafiz
   H/P :- +60 18-361 6508
2. Ong Chien Ming
   H/P :- +60 12-242 9868
3. Garret Yong Shern Min
   H/P :- +6017 307 8780
4. Mario Susanto
   H/P :- +62 821-1120-8808

2.3 Technical & Professional Strengths

Syed Zubin Hafiz

Has experience in the following languages :

- Java
- Python
- HTML + CSS
- Vega Lite

Has experience in the following topics:

- Data structures and algorithms
- Mobile app development
- Object Oriented Design
- Software development practices
- Data Visualization

Ong Chien Ming

Has experience in the following languages:

- Java
- JavaScript
- Python
- C
- HTML + CSS

- MicroPython
- Vega Lite

Has experience in the following topics:

- Data Structures and Algorithms
- Mobile App Development
- Object Oriented Design
- Software Development Practices
- Artificial Intelligence/Deep Learning

Garret Yong Shern Min

Has experience in the following languages:

- Java
- JavaScript
- Python
- SQL
- HTML

Has experience in the following topics:

- Data Structures and Algorithms
- Mobile App Development
- Object Oriented Design
- Usability
- Software Development Practices
- Leadership and innovation

Mario Susantoo

Has experience in the following languages:

- Java
- JavaScript
- Python
- R
- SQL
- HTML + CSS

Has experience in the following topics:

- Data structures and algorithms
- Mobile app development
- Object Oriented Design
- Software development practices

2.4 Fun Fact of Each Member

Syed Zubin Hafiz

I love listening to music, and want to buy a guitar pretty soon. Left the one I owned back home.

Ong Chien Ming

I am a cheerleader and love to participate in sports. I go to the gym a lot to keep myself fit and also acts as a coping mechanism for our stressful student life. My music taste is very varied as I listen to different genres of music and also different languages.

Garret Yong Shern Min
Is an avid foodie

Mario Susantoo
Likes sushi

3 Team Schedule

3.1 Meeting and Work Schedule

There will be regular team meetings that will be hosted once a week between Fridays to Sundays which will be voted for via WhatApp. These meetings are held through Zoom Meetings and last for 1-2 hours. More meetings can and will be scheduled should the need arise.

Meeting minutes will be written by Ong Chien Ming to keep track of the progress of the project.
Meeting minutes links up to 1/4/2023:
29/3/2023 - Meeting minutes 29/3/2023
1/4/2023 - Meeting minutes 1/4/2023

There is no fixed work schedule but there are early deadlines set up to provide a buffer in the event that there is one member whose work needs to be worked on for longer.

3.2 Workload Distribution

To fairly distribute the work, we would first have to analyse the weight of each task and assign some value to the task based on their perceived difficulty and apparent difficulty. The first round of distribution would be based on the points of perceived difficulty; this would be achieved by hosting a planning poker session to allocate the points and then the tasks would be distributed while ensuring that all members are shouldering an equal number of points for the tasks to ensure fairness. After that, each task can be reallocated difficulty points based on their apparent difficulty which is the amount of difficulty a task turns out to possess after conducting some research and trial. The members who realise the task they are undertaking is heavier than they intended, are allowed to deallocate some of their tasks and have them delegated to others; this works for the converse as well.

4 Technology Stack and Justification

4.1 Technology Stack

For the project, Java will be used as the programming language, while Java Swing will be used for the graphical user interface.

4.2 Justification

Java was chosen as the base language due to it being designed specifically for Object Oriented Programming. This is important not only because OOP allows for extensibility and maintainability, but also because the guest of honour that the software is intended for is passionate about OOP. Furthermore, Java is performant while also being cross-compatible between multiple platforms, which is necessary as the platform that the software is intended

to be run on during the Open Day is not stated. Finally, as every member of the development team is familiar with Java, using the language would improve productivity as less time will be used to understand the technology stack.

Three other programming languages were considered: Python, JavaScript, and C++. The main advantages of Python are that the language is highly flexible, along with the fact that the development team is highly familiar with it. However, we decided not to choose Python because the language, including most third party graphical libraries, are not designed to support OOP to the extent that Java does. We furthermore considered JavaScript as the language to be used as it is designed with a graphical user interface in mind, but we decided against using it as JavaScript's dynamic typing means that errors are more likely to occur, and because JavaScript's support of OOP is somewhat limited. Finally we considered C++, which is fast and OOP oriented, but we decided against using C++ as no team member is familiar with the language, and because applications created with C++ may not necessarily be cross-compatible with multiple platforms.

Java Swing was the GUI library chosen for the project. One of the reasons for this is that it is widely used, meaning that solutions to problems during development can be found more easily. Furthermore, some team members are familiar with the library, which allows for increased productivity as less time is required to understand the library. It could be noted that while JavaFX was also considered as a GUI library, it was not chosen due to its complexity, and because a variant of XML would need to be learnt in order to use it.

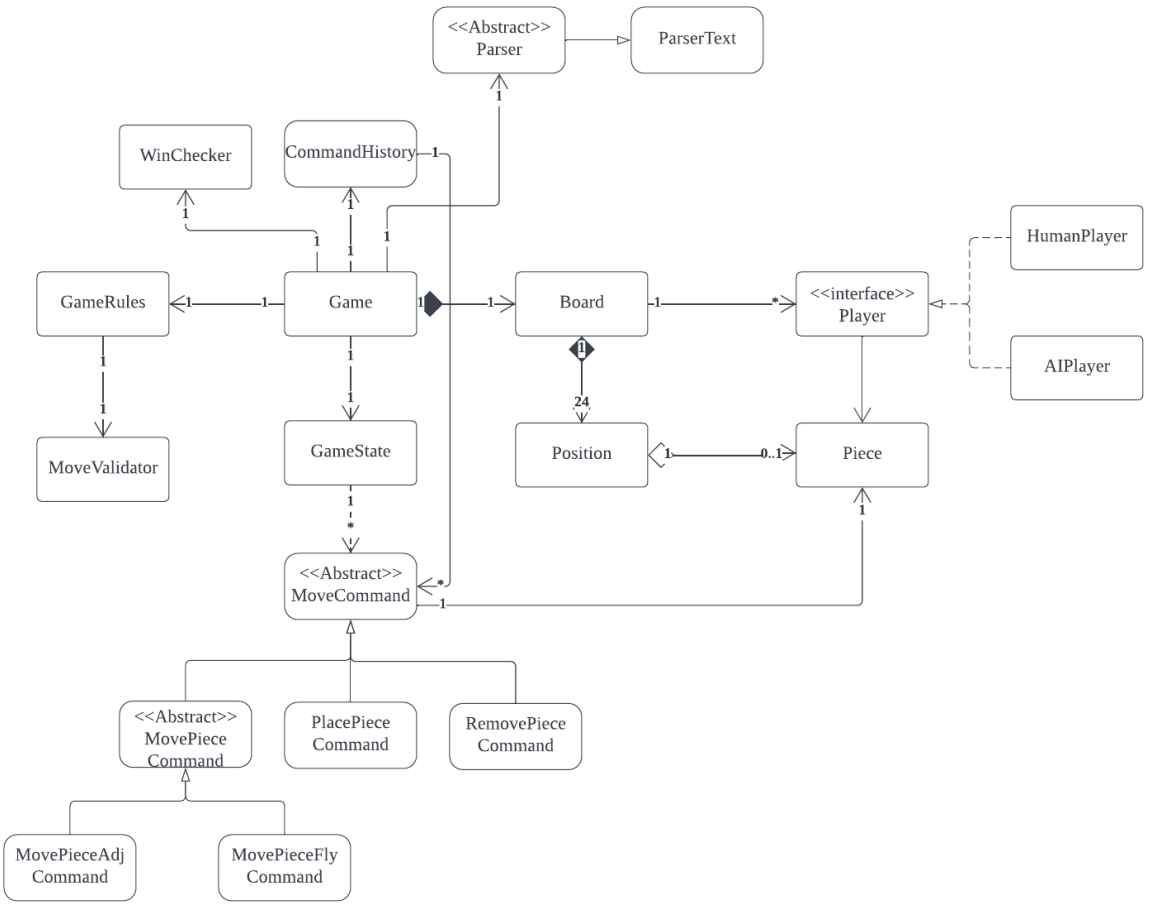
User Stories

Outside the game
As a user, I want a menu screen so that I can select between playing alone or with someone else
As a user, I want to be returned to the main menu after a game ends so that I can start a new game
As a user, I want a single player mode in the menu so I can play against the computer
As a user, I want the computer to automatically make moves after my turn so that it can act as a proper 2nd player
Starting the game

As a user, I want to start the game with 9 tokens so that I can place them on the board
As a user, I want to be able to undo my actions so that I can undo my mistakes
As a user, I want to form a mill when I align 3 men along a line so that I can retire one of the opponent's men
As a user, I want to only be able to make legal moves so that no one can cheat and the game flow is smooth
As a user, I want the starting player to be randomised so that each player has a fair chance to start first
Moving Phase
As a user, I want to be able to move my pieces to adjacent positions after I place all my men so that I can form additional mills
As a user, I want to be able to move my pieces to any empty position when I have three pieces left so that I have a chance to win despite my opponent's numerical advantage
Ending the Game
As a user, I want the game to end once one player has less than 3 men so that I can move on to the next game.
As a user, I want the player with lesser than 3 men to lose so that the game can end and a winner can be declared

Basic Architecture

Domain Model



Alongside the basic requirements, our team has chosen to do B and C, where the game supports undoing previous moves, saving the game, and playing against a computer

Justifications

Game

The Game class serves as the main orchestrator of the Nine Men's Morris game. It manages the game's state, players, board, and game rules. It is responsible for coordinating the interactions between various game components and handling the overall game flow. As such, it holds instances of various classes relevant to the functionality of the game, such as the Board class and the GameRules class.

Whilst designing the domain model, an option was considered where the various outlying classes, such as Board, WinChecker, GameState, and GameRules were to be integrated directly into the Game class. This approach was however decided against as it may result in the Game class being excessively cluttered, thus resulting in the god class code smell and therefore hampering readability and maintainability.

Board

The Board class represents the 9MM game board. It is responsible for maintaining the positions of all game pieces and providing methods to interact with the board, such as placing, moving or removing pieces. The board furthermore has the ability to access the player that is running a turn so that it would be able to modify the state of the board accordingly.

Player interface

The Player interface defines the common methods that all types of players must implement. This abstraction allows for different player types to be introduced without changing the code that interacts with players.

There were multiple possible alternatives for implementing this feature, such as having a concrete Player class with subclasses or with an abstract Player class. The idea of a concrete Player class was decided against because of the need to implement the advanced requirement of playing against a computer, so having a singular class would hurt the extensibility of the code. Meanwhile, while an abstract class works similarly to an interface, an interface is preferable as a human player and a computer player is unlikely to share any code because of the significant differences in their operation.

HumanPlayer

The HumanPlayer class implements the Player interface for human players. It is responsible for handling user input and converting it into game moves, and therefore allowing the game to interact with human players seamlessly.

AIPlayer

The AIPlayer class implements the Player interface for AI players. It is responsible for generating moves using AI algorithms, such as a search algorithm. This class allows the game to interact with AI players as if they were human players.

Piece

The Piece class represents a game piece in the Nine Men's Morris game. It is responsible for tracking the owner(player) and the current position of the piece on the board. This class provides a way to identify and manipulate individual game pieces during gameplay.

An approach that was initially chosen was to integrate the piece into the position by indicating the piece using a variable, such as an enumerator. While this would likely be possible as Pieces exclusively stay on Positions, it could potentially harm extensibility and may result in unforeseen problems with the code. Furthermore, doing so may break the Single

Responsibility Principle as As such, the approach selected was modified to have a separate Piece.

Position
The Position class represents a position on the game board. It is responsible for tracking the coordinates and any associated piece, This class enables querying and manipulation of the board state during gameplay.

GameState
The GameState class represents the state of the game at any point in time. It is responsible for tracking the information like the current player, the number of pieces placed, and the game phase. This class provides an easy way to manage and query the state of the game.

GameRules
The GameRules class represents the rules of the Nine Men's Morris game. It is responsible for validating moves, determining if a move forms a mill, and checking if a player has won or if the game has reached a draw. This class encapsulates the game's rules and logic.

Move
The Move class represents a move in the game, including placing a piece, moving a piece, or removing a piece. It is responsible for storing the details of the move, such as the piece involved and the positions affected. This class provides a way to record and manage moves during gameplay.

MoveCommand interface
The MoveCommand interface represents a move command and provides methods for executing and undoing the move. This abstraction allows for different types of move commands to be introduced and managed in a uniform way.

PlacePieceCommand
The PlacePieceCommand class implements the MoveCommand interface for placing a piece on the board. It is responsible for executing and undoing the placement of a piece during gameplay.

MovePieceCommand
The MovePieceCommand class implements the MoveCommand interface for moving a piece on the board. It is responsible for executing and undoing the movement of a piece during gameplay,allowing players to move their pieces according to the game's rules.

RemovePieceCommand
The RemovePieceCommand class implements the MoveCommand interface for removing a piece from the board. It is responsible for executing and undoing the removal of a piece during gameplay, which typically occurs when a player forms a mill and is allowed to remove an opponent's piece.

CommandHistory

The CommandHistory class manages the move command history for undo and redo functionality. It is responsible for maintaining a record of executed MoveCommands and providing methods to undo or redo moves. This class enables players to revert their moves, which is particularly useful when implementing features like an Undo Move button in the game's user interface.

Parser

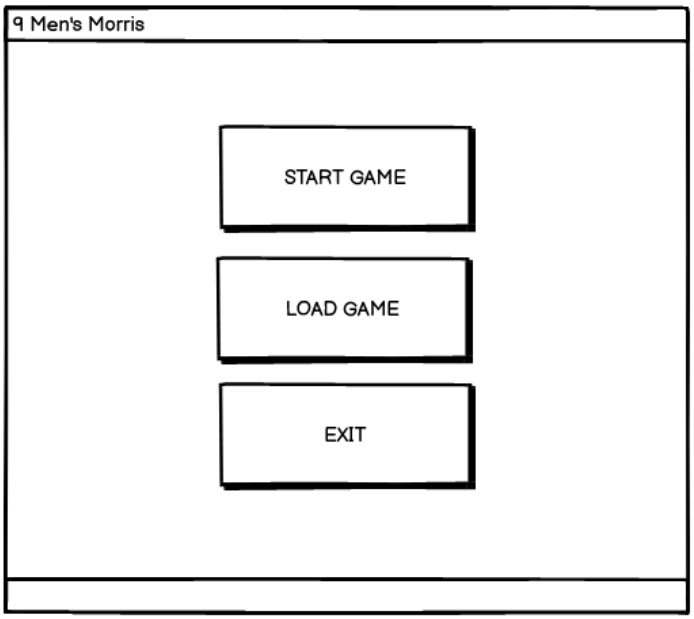
This Parser abstract class will be a class that will lay the framework for other Parser classes that will extend it. As it is said, the initial implementation will utilise the saving of the game state, which will be stored in a .txt file. However, this abstraction will allow for the extension of other parsers from and to different file types. This is necessary as the requirements note the possibility of extending the software to save the game with different file types, meaning that subclasses to handle such situations are necessary.

ParserTxt

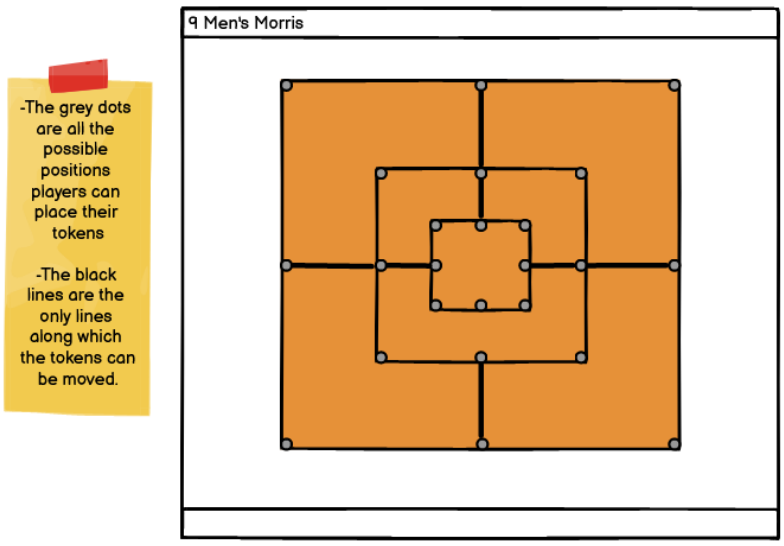
This class is the concrete class responsible for the parsing of the current game state as a txt file. It is also responsible for the parsing of the saved game state back into an active game state from a previously saved .txt file.

Basic UI Design

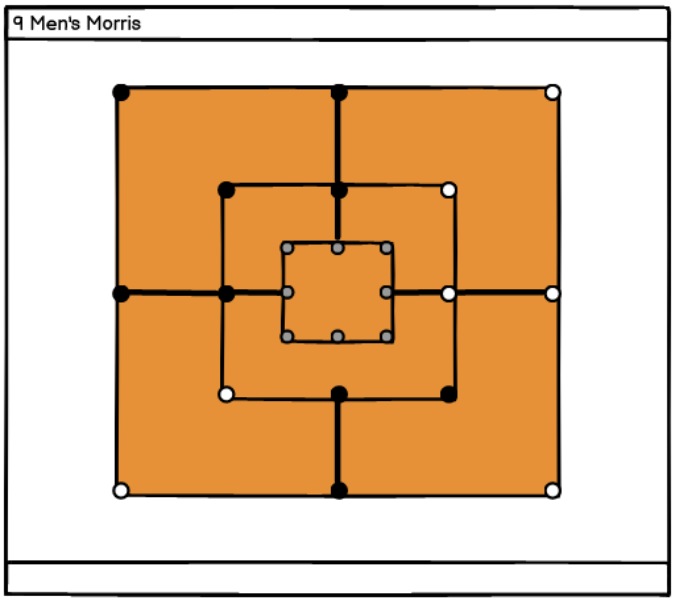
Starting Screen



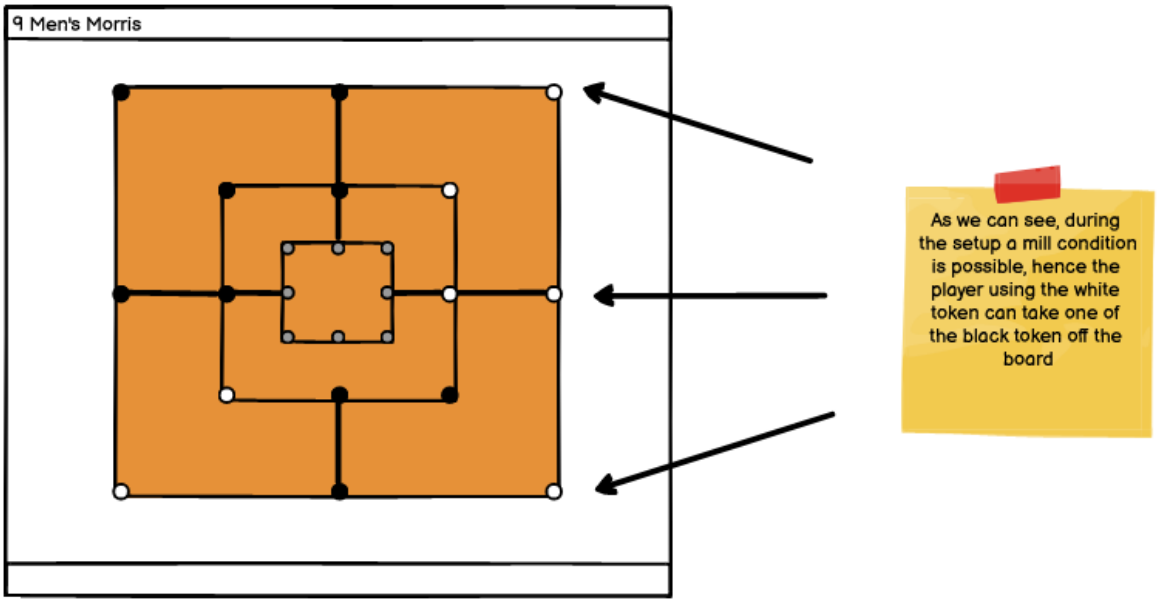
Game Board

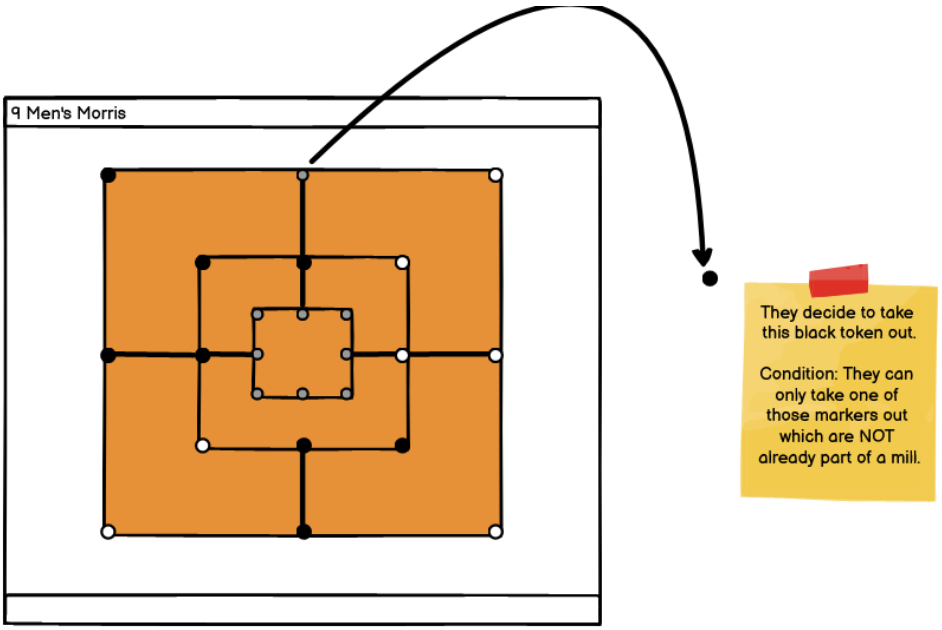


Game Board with Tokens Placed

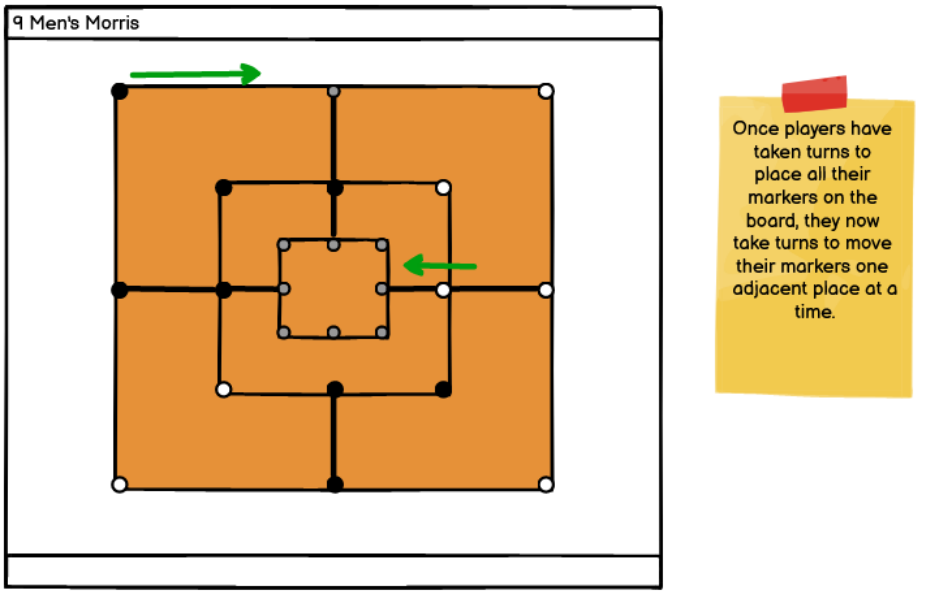


Mills

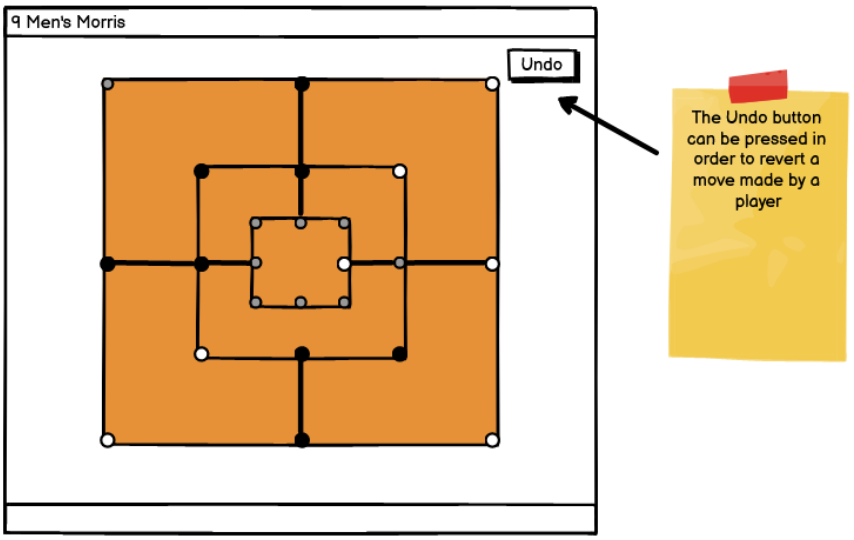



9 Men's Morris
They decide to take this black token out.
Condition: They can only take one of those markers out which are NOT already part of a mill.

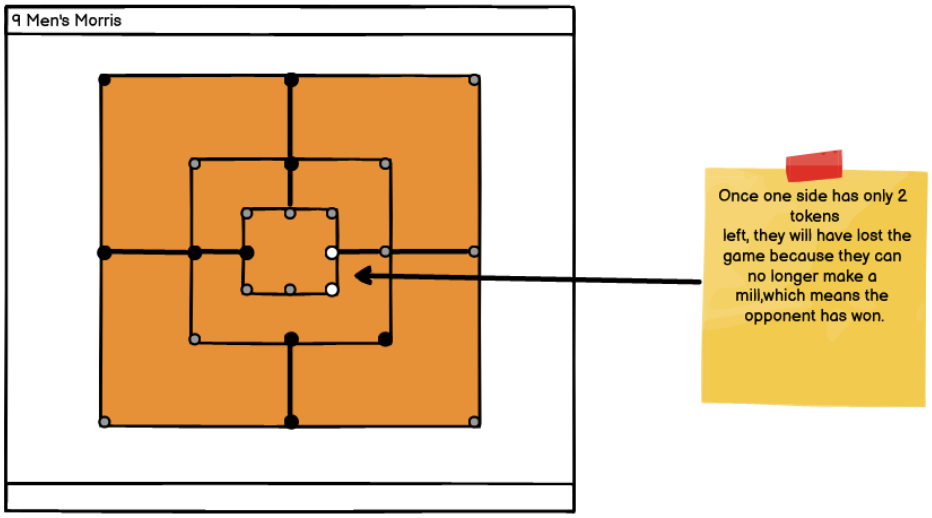
Move Phase


9 Men's Morris
Once players have taken turns to place all their markers on the board, they now take turns to move their markers one adjacent place at a time.

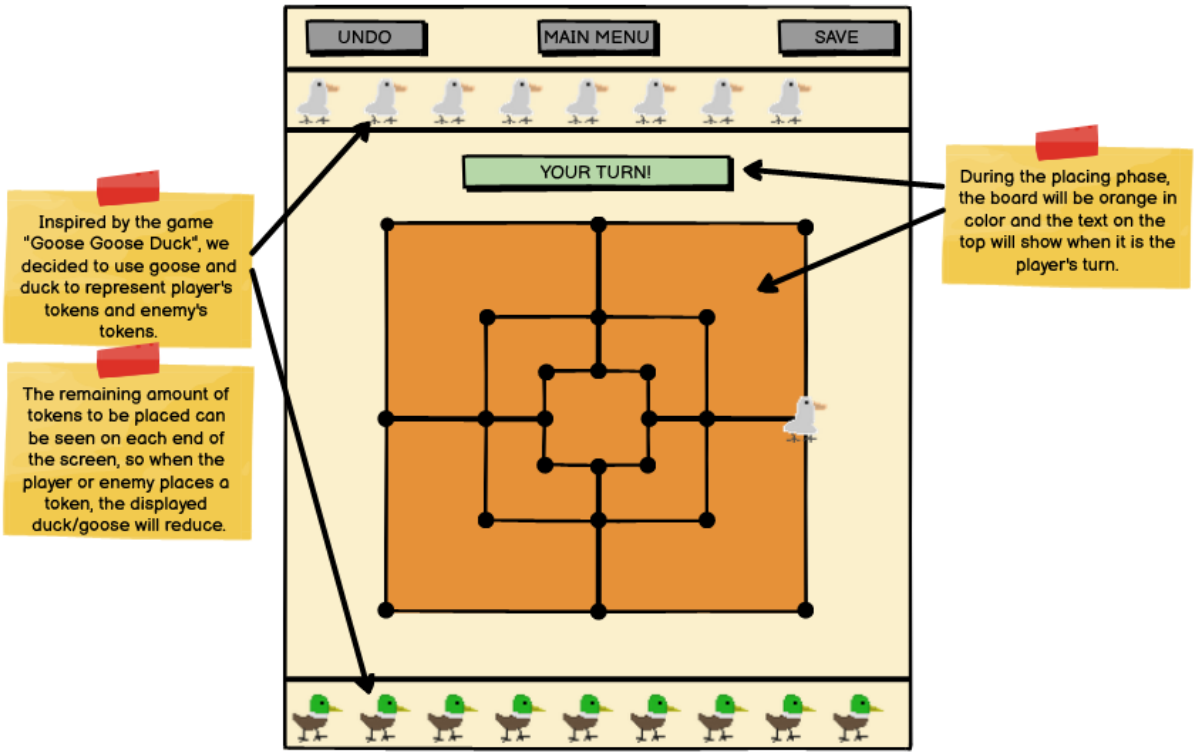
Undos

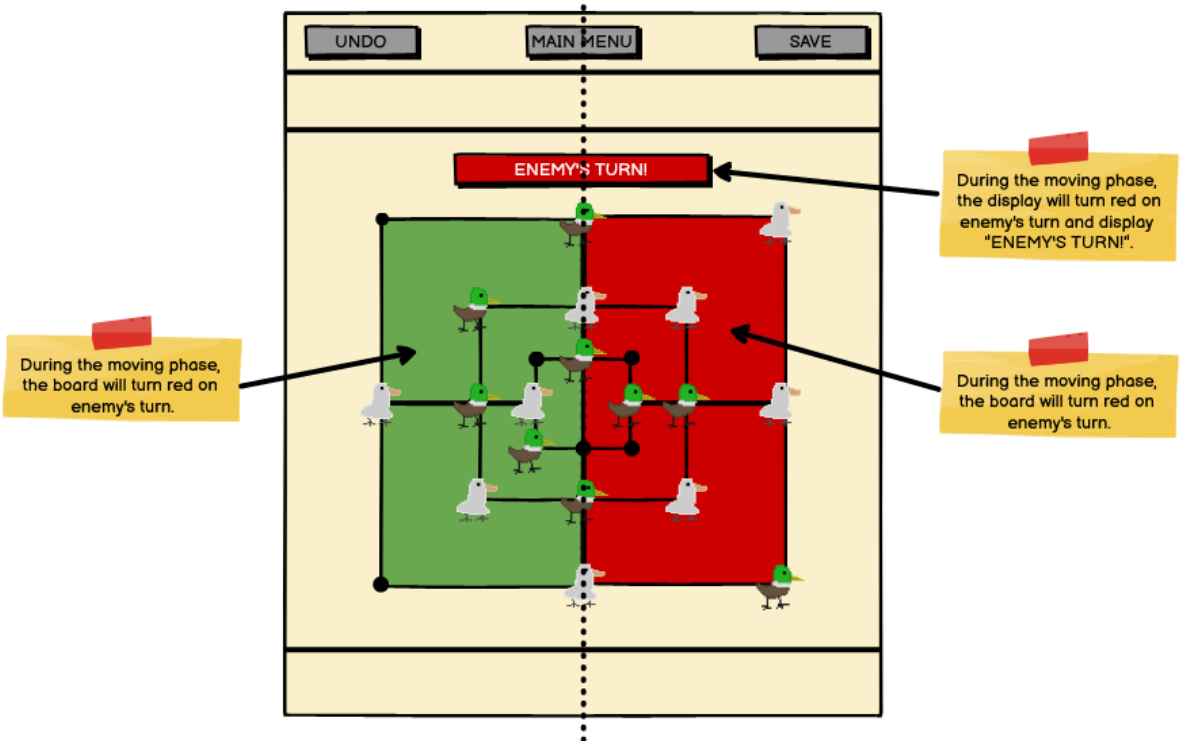


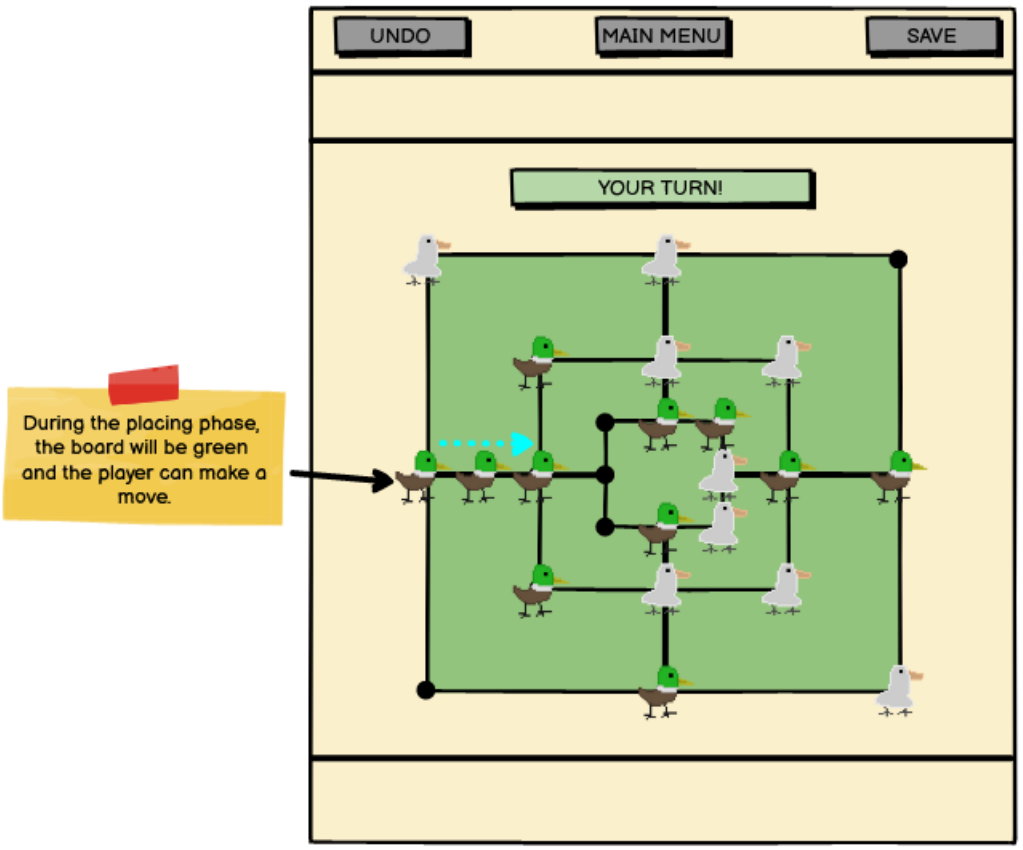
Win condition

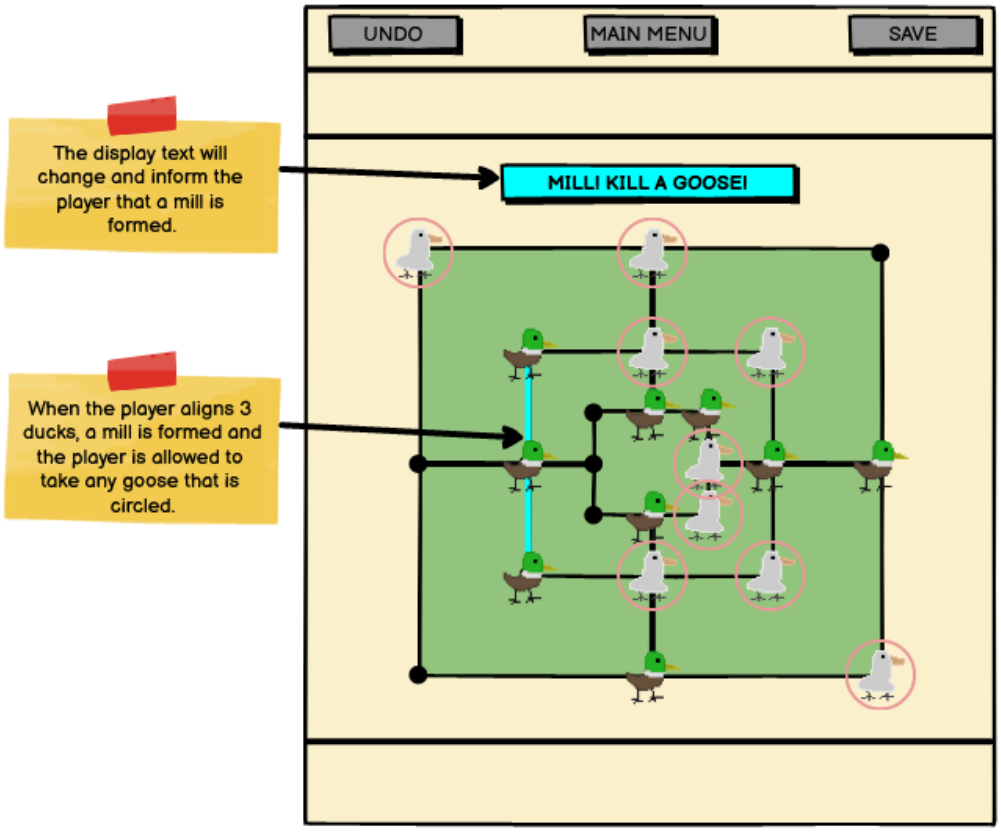


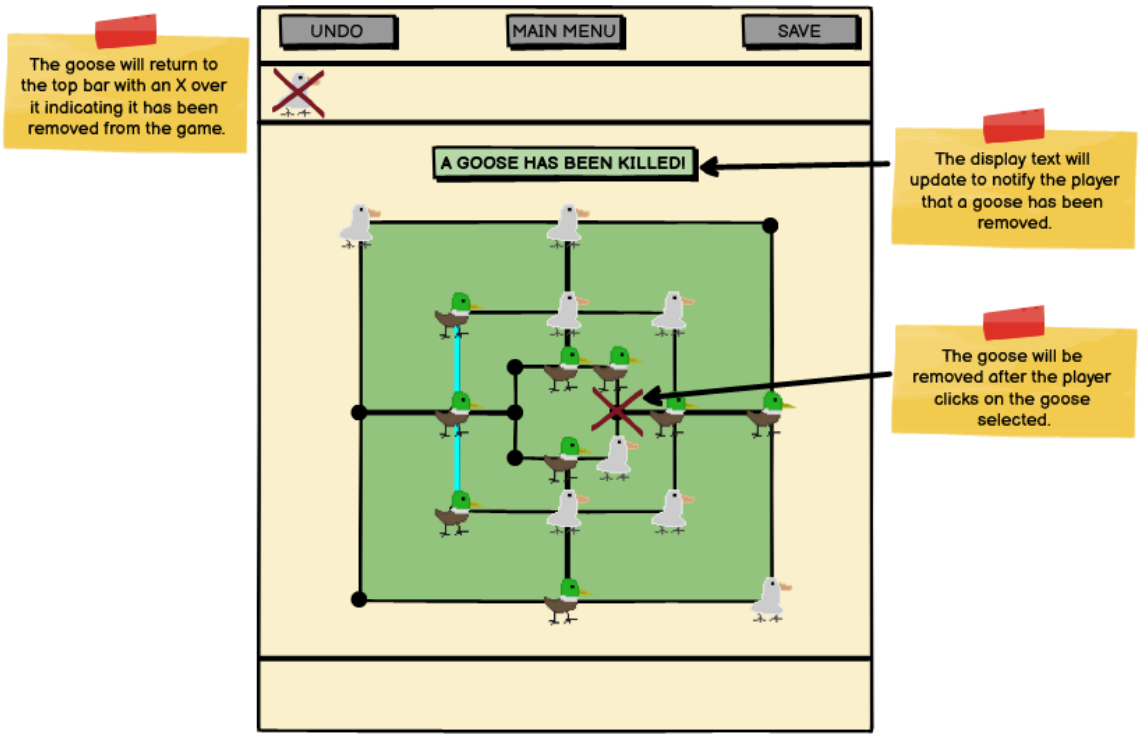
Detailed UI Design

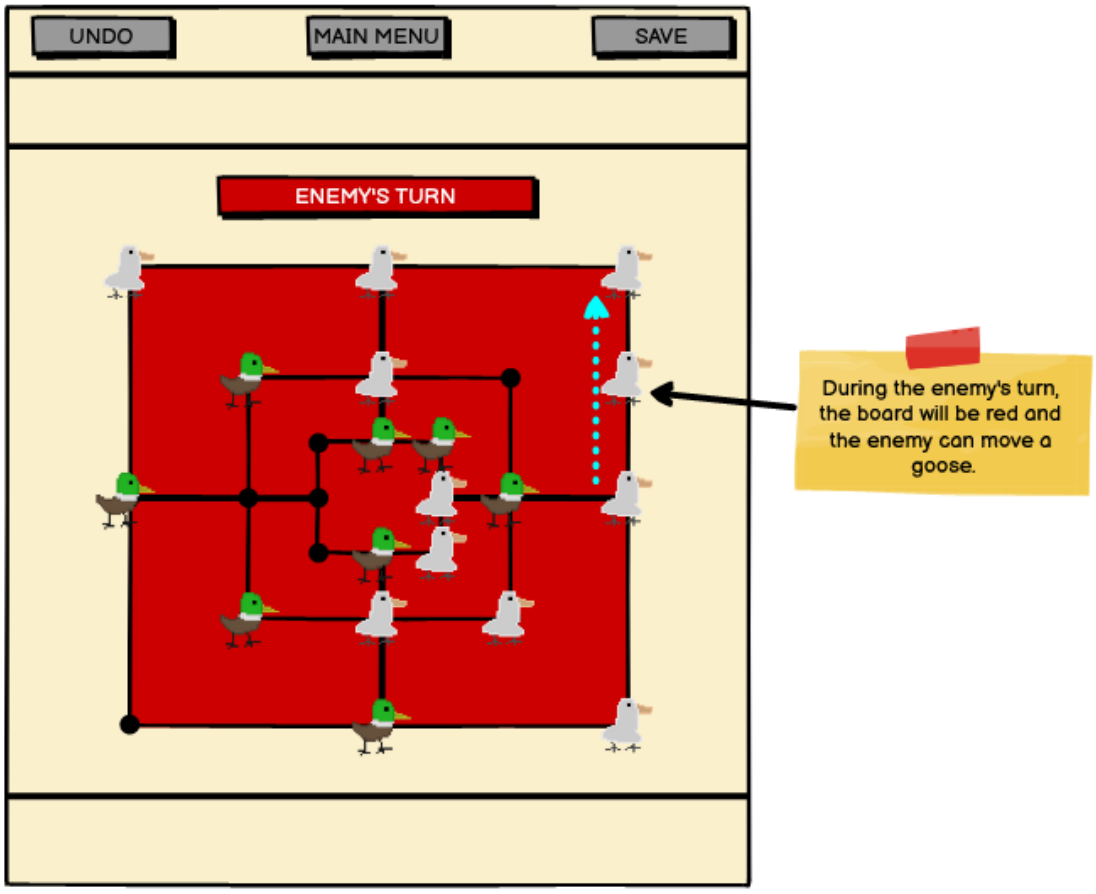


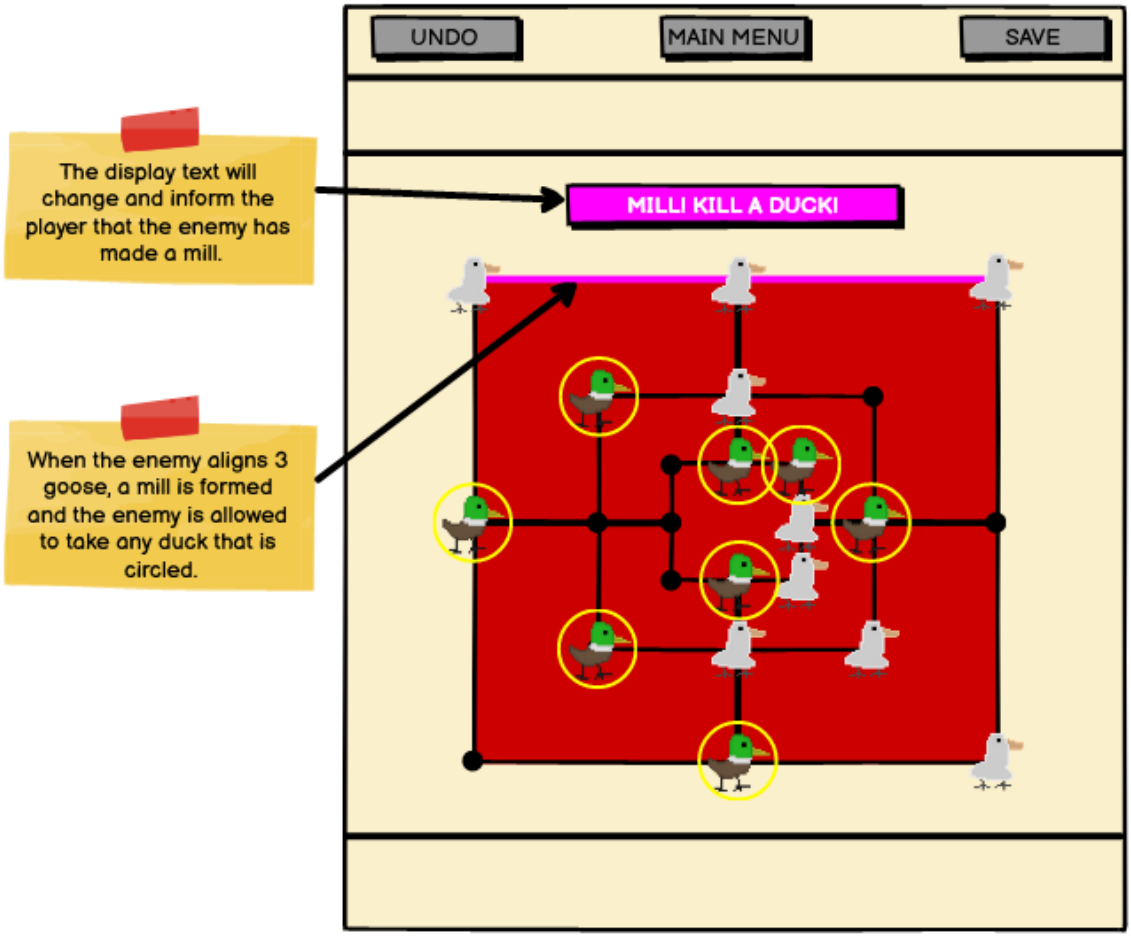


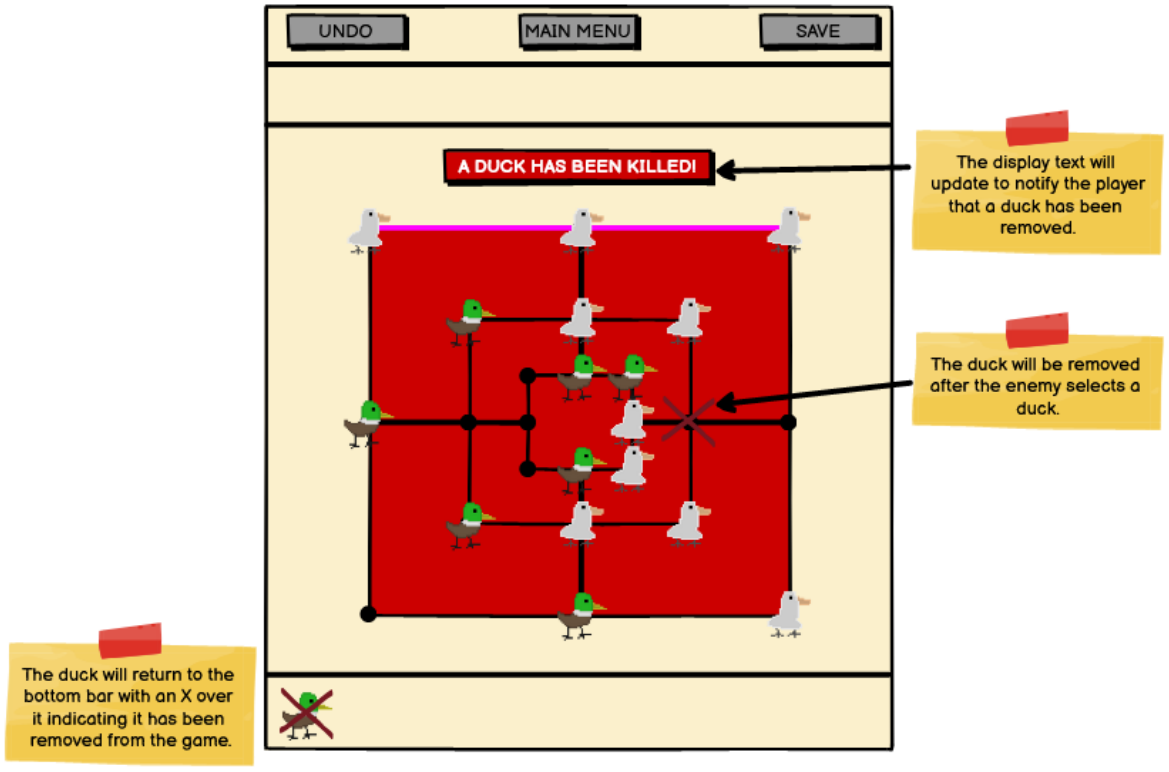

UNDO
MAIN MENU
SAVE
YOUR TURN!
During the placing phase, the board will be green and the player can make a move.


UNDO
MAIN MENU
SAVE
MILL! KILL A GOOSE!
The display text will change and inform the player that a mill is formed.
When the player aligns 3 ducks, a mill is formed and the player is allowed to take any goose that is circled.

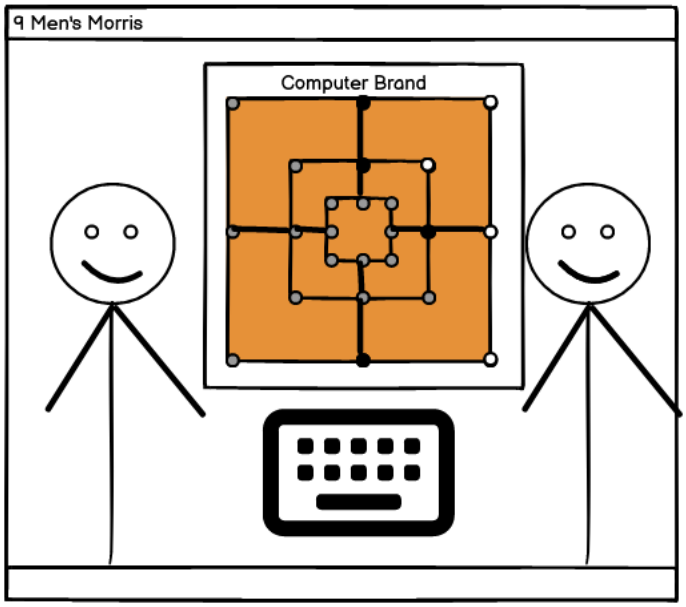

The goose will return to the top bar with an X over it indicating it has been removed from the game.
UNDO
MAIN MENU
SAVE
A GOOSE HAS BEEN KILLED!
The display text will update to notify the player that a goose has been removed.
The goose will be removed after the player clicks on the goose selected.


UNDO
MAIN MENU
SAVE
ENEMY'S TURN
During the enemy's turn, the board will be red and the enemy can move a goose.

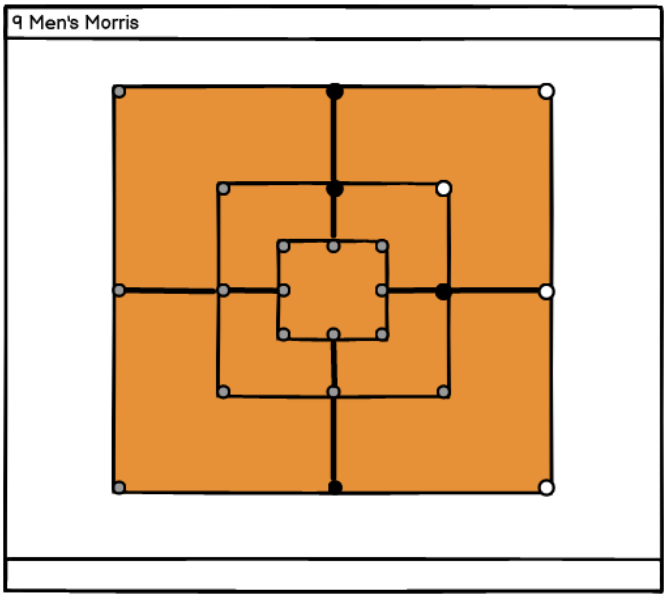

UNDO
MAIN MENU
SAVE
The display text will change and inform the player that the enemy has made a mill.
MILLI KILL A DUCK!
When the enemy aligns 3 goose, a mill is formed and the enemy is allowed to take any duck that is circled.


UNDO
MAIN MENU
SAVE
A DUCK HAS BEEN KILLED!
The display text will update to notify the player that a duck has been removed.
The duck will be removed after the enemy selects a duck.
The duck will return to the bottom bar with an X over it indicating it has been removed from the game.

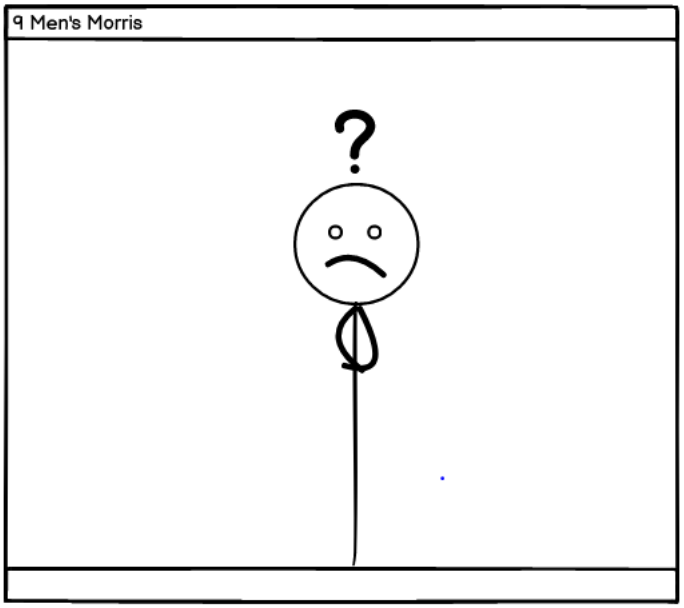
Draw Condition



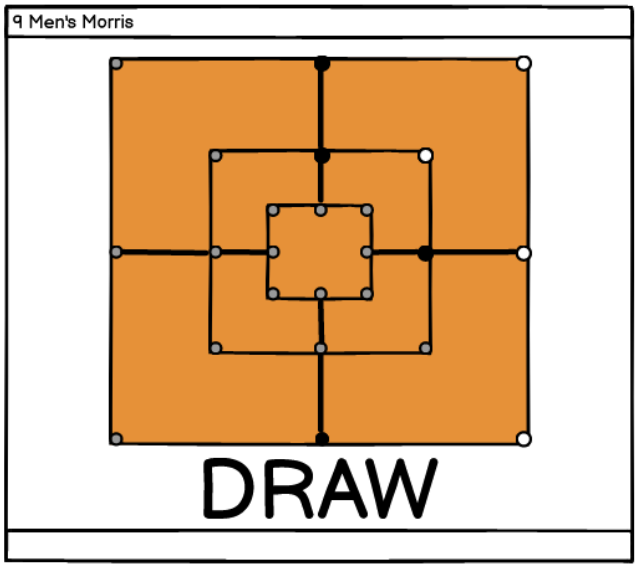
- Andrew is excited to play Nine Men's Morris with his friend
- Andrew's friend is just as good as Andrew



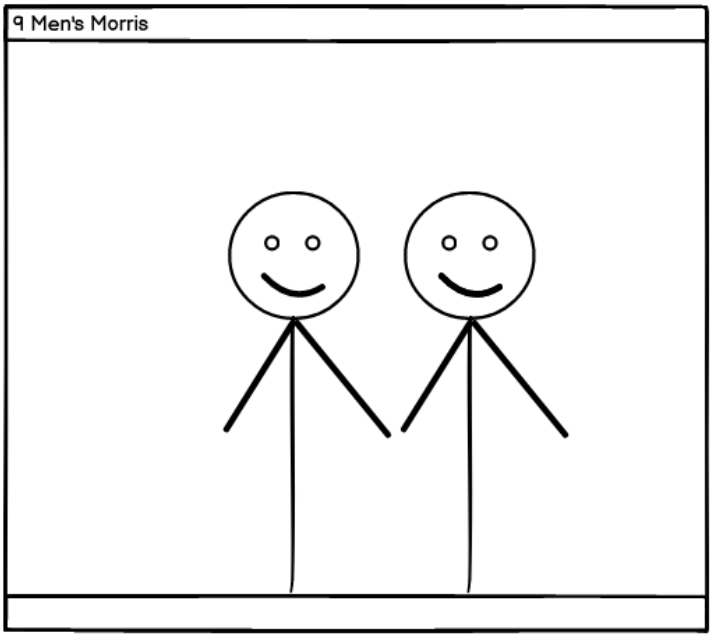
- Andrew played a very close game with his friend
- It comes to Andrew's turn but due to the situation, Andrew cannot make any moves as all his pieces are blocked



- Andrew realises due to not being able to make his move, the game cannot proceed as there are no more moves
- Andrew feels frustrated

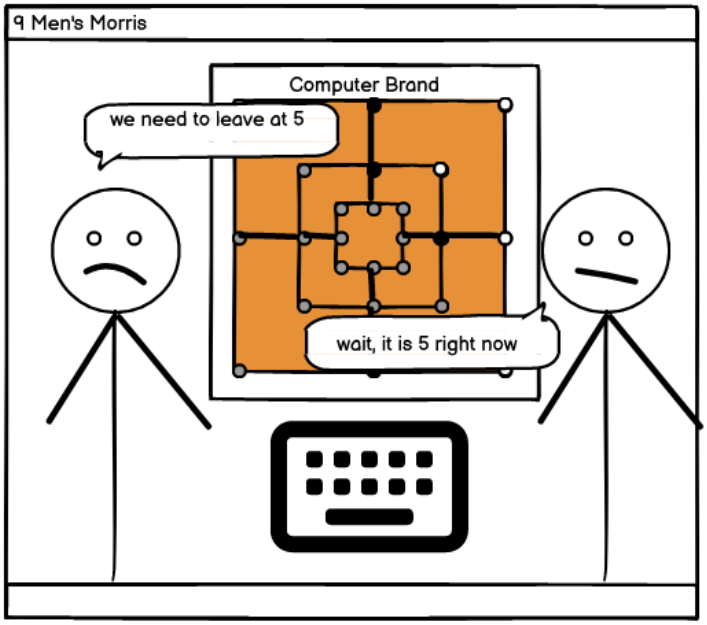


- The game auto-resolves and declares that there are no legal moves to be made, and declares a draw

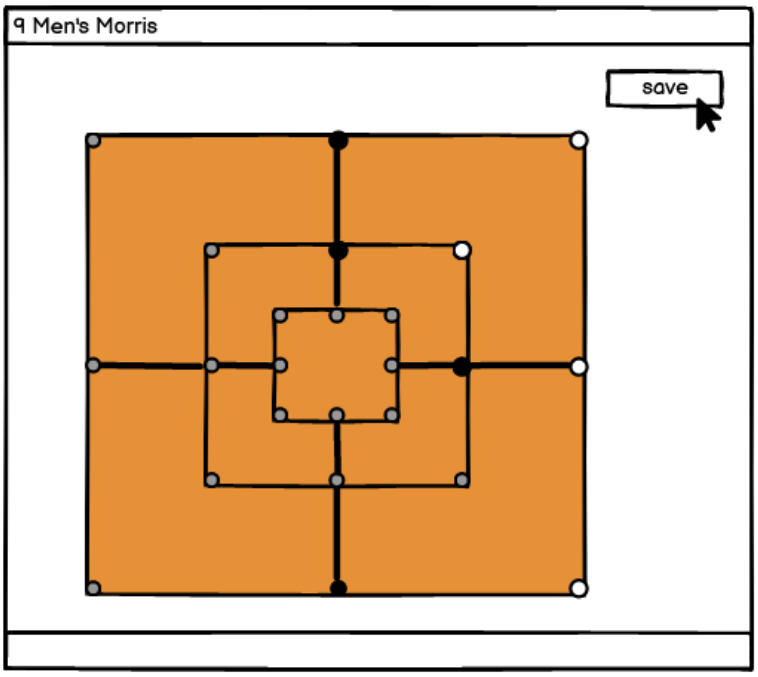


- The confusion and worry Andrew feels have vanished
- Andrew and his friend feel happy to have the game resolved in a satisfactory way (no random winner announced)

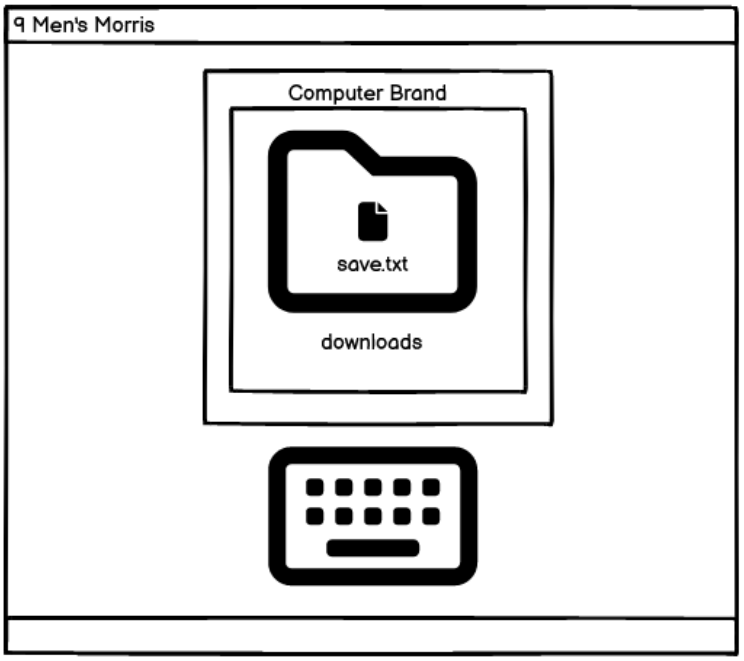
Save Function



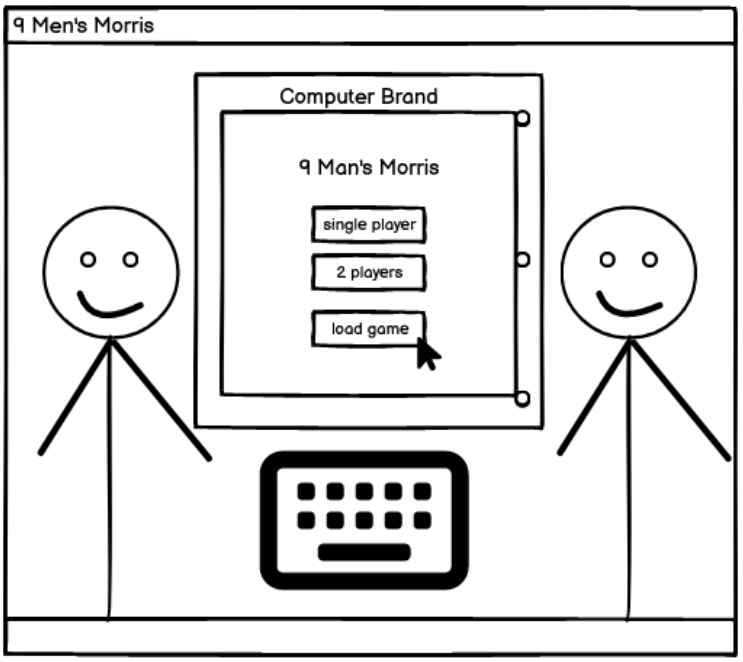
- Andrew is playing Nine Men's Morris with his friend
- Andrew and his friend have to head out soon but thought they had time to finish the game
- But alas, time is up and they have to leave



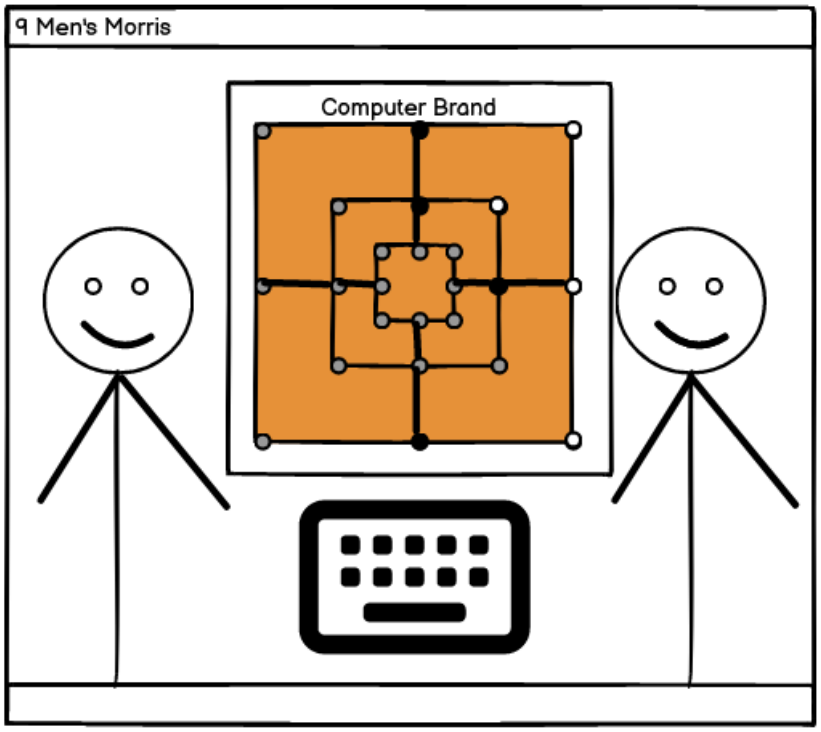
- Andrew wants to continue the game with his friend, so he clicks on the save game button



- Andrew's current game is saved as a txt file

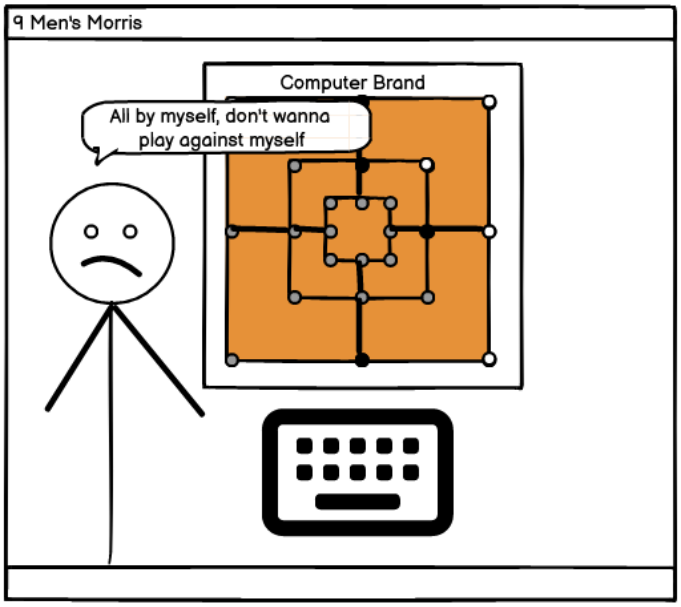


- Once Andrew and his friend get back, they start the game and on the menu screen clicks on the load game button
- Andrew and his friend select the previously saved txt file and the game resumes from the previous game state

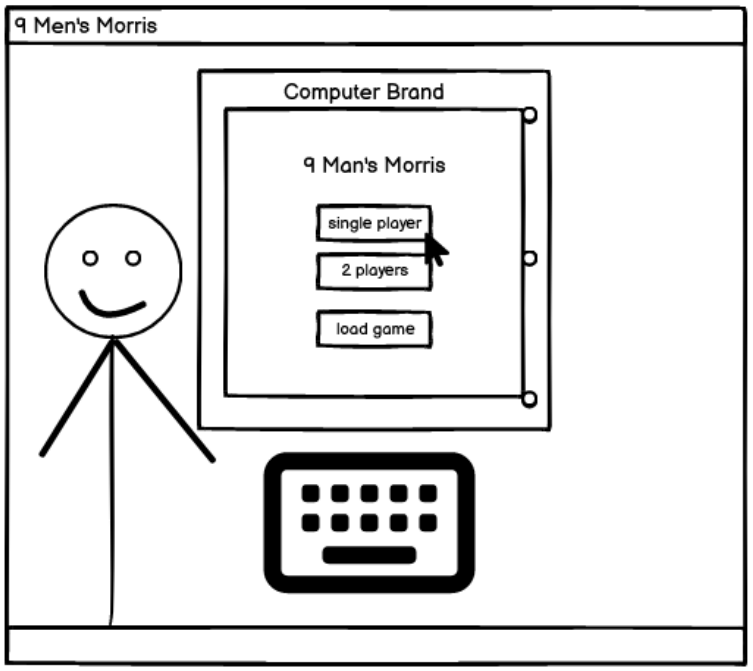


- Andrew and his friend is able to complete their game and they enjoyed playing with each other

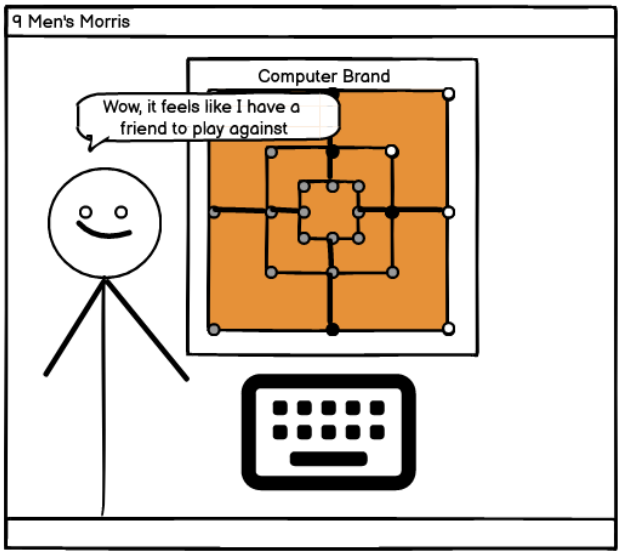
Computer Player



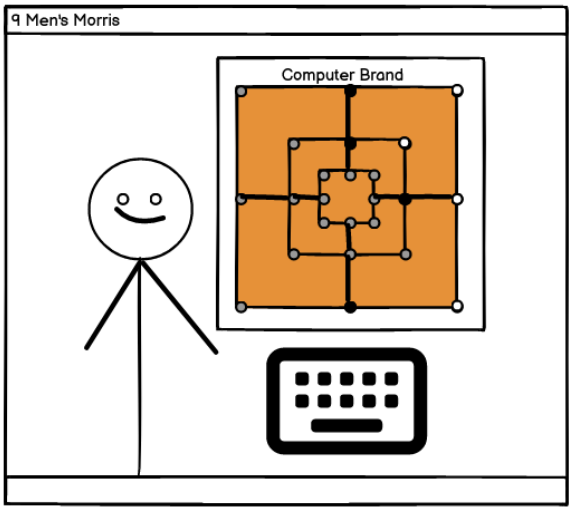
- Andrew wants to play Nine Men's Morris with his friend
- Andrew realises he has no friends
- Andrew does not want to play against himself
- Andrew is sad



- Andrew then realises there is a single player option
- Andrew clicks on the single player option



- The game loads up as usual
- However, once Andrew is done with his move, the computer makes a move after him
- Suddenly Andrew feels like he is not playing alone



- While Andrew searches for actual people to play 9 Man's Morris with him, he will continue to play 9 Man's morris with the computer forever.
- Despite the bleak future, Andrew is contented