FIT3077 Sprint One Report

3 Man’s Mischief (Team 38)

Josiah Schuller, Caleb Ooi, Steven Pham

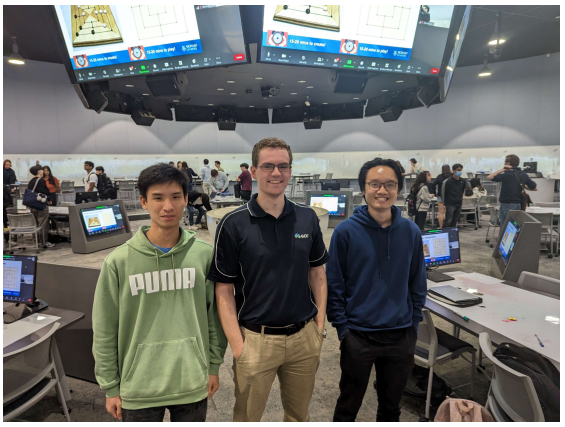
Table of Contents

Team Information

Team Name and Team Photo

Team name: 3 Man’s Mischief

Team photo:



Team Membership

Josiah Schuller:

- **Email:** jsch0049@student.monash.edu
- **Mobile:** 0450 744 519
- **Technical strengths:**
    - Experience building back-end of applications in Python (including Chess: https://github.com/josiahschuller/chess2023)
    - Experience building full-stack applications. This includes working at AMOG Consulting on the Smart Mooring Integrity Checker (https://amog.consulting/products/smart-mooring-integrity-checker-smic-2).
    - Experience building front-end applications using HTML, JavaScript and CSS
    - Experience building front-end applications using ReactJS
    - Experience building back-end applications using Python and NodeJS
    - Experience with postgreSQL databases
    - Experience building CI/CD pipelines
    - Experience using and managing Git repositories
    - Experience using and developing APIs
    - Experience developing unit tests
    - Some experience programming in C++, Java and Haskell
- **Professional strengths:**
    - Quick to see and respond to team communication
    - Natural at facilitating and minute-taking in meetings
    - Encouraging team members to communicate and valuing their contributions to discussion - not being a “conversation hog”
    - Keeping team members on track with their work
    - Natural at presenting work to stakeholders and bosses multiple layers up the chain of command
- **Fun fact:** I love hot cross buns! Chocolate or toasted traditionals with butter are top notch!

Caleb Ooi:

- **Email:** cooi0004@student.monash.edu
- **Mobile:** 0414 464 326
- **Technical strengths:**
    - Experience building front-end web applications using HTML, JavaScript and CSS
    - Experience building android applications using Java
    - Experience building back-end applications using NodeJs
    - Experience with SQL and NoSQL database such as SQL lite and MongoDb
- **Professional Strengths:**
    - Highly adaptable and flexible, able to quickly adjust to changing environments.
    - Highly motivated and able to learn new technologies and processes quickly
    - A team player with excellent communication skills, able to collaborate effectively with colleagues and stakeholders to achieve shared goals and objectives.
- **Fun fact:** I enjoy watching anime. Currently my favourite anime is Attack on Titan.

Steven Pham:

- **Email:** spha0018@student.monash.edu
- **Mobile:** 0435 046 630
- **Technical strengths:**
    - Experience building front end applications using HTML, JavaScript and CSS
    - Experience building user interfaces with Python Tkinter
    - Experience using Object-Oriented Programming
    - Experience building 3D games with Unreal Engine 5 https://github.com/Edenste/ProjectGrapple
    - Experience using APIs to develop projects
    - Experience using Git to develop and manage repositories
    - Experience developing unit tests
- **Professional strengths:**
    - Speaking confidently and clearly when presenting to stakeholders and clients
    - Explaining high-level concepts in an easy to understand format to teammates and clients
    - Understanding and interpreting the needs and wants of a client into technical requirements
- **Fun fact:** Fantasy books are a guilty pleasure of mine. I go crazy for anything written by Brandon Sanderson.

Team Schedule

Regular Meeting Schedule

The team will hold a regular meeting each **Thursday from 6 PM to 8 PM.** If a person or the team is unavailable on this day, a *backup day* has been scheduled for **Saturday from 12 PM to 2 PM**. The team will discuss each individual's progress on the project during the week and collaborate to resolve any issues that have arised. Additional meetings may be scheduled if needed.

Regular Work Schedule

The team will choose user stories and work on their own time to implement them at their discretion. If any issues arise they can be communicated through the shared communication channel called Discord where the team will assist the individual. If the problem is beyond the scope of text-based communication then the issue will be brought up at the regular meeting time outlined in the Regular Meeting Schedule. Additional meetings may be scheduled if needed.

Workload Distribution and Management

The project will be developed using the Scrum methodology and the user stories will be measured by the team quantitatively in terms of difficulty and length of time to complete using Planning Poker. This measurement will then be used to evenly distribute the workload between the team so that each team member has an equal amount of work to complete. This measurement will be managed and updated between sprints so that the workload stays evenly distributed.

In the event that a team member is unable to fully complete their distributed workload by a deadline, the team will assist and the workload will be evenly distributed to the rest of the team.

The expected workload is to be 3 to 4 hours of work on the project per week for each team member.

Technology Stack and Justification

Teams' expertise

Josiah: Java, Python, Javascript
Caleb: Java, C, Python, Javascript
Steven: Java, C++, Python, Javascript

Based on each team member's expertise, the programming language we are considering are Java, Python and Typescript. The table below highlights some of the pros and cons of using these languages for our projects.

Languages	Pros	Cons
Java	1. Fully supports OOP 2. Strongly typed-easy to debug	1. Verbose Syntax- can be made easier with good IDE such as intellij 2. Can be complex with different build tools such as Gradel or Maven
Python	1. Simple 2. Supports a varieties of libraries	1. Dynamically typed-harder to debug 2. Does not fully support OOP such as encapsulation
Typescript	1. Smart strongly typed, with the ability to infer types 2. Compiles to JavaScript, can be used with HTML and css	1. Complicated build process and configuration required with the TypeScript compiler

Technology stack

For the UI, our team is currently considering a terminal-based UI. However, based on research, there are a few libraries/frameworks that can be used for each language. For Java applications, libraries such as JavaFX and Swing can be used to create desktop applications. For Python applications, libraries like Tkinter and PyGUI can be used. As TypeScript is compiled to JavaScript, TypeScript can be used with HTML and CSS. Additionally, there are frameworks such as ReactJS and NextJS that can be used for developing UI.

Our final choice of technologies was based on several factors, including the team's expertise, project requirements and the level of support available for development, which takes into account factors such as object-oriented design implementation, type support. Based on these factors, we have decided not to use Python for this project. Although Python does support type checking and has some basic type annotation features, it lacks a strongly typed system. The lack of type reinformace can make the development process slower as it can be difficult to debug and causes type-related errors at runtime.

Between Java and Typescript, we have decided to select Java. While all our members have experience with developing with Typescript, some of us do not have the experience of using it to develop in a object oriented style. All of members are familiar with using Java to develop object oriented projects. Additionally, Java supports OOP designs out of the box along with tools from IDE making it easy to get started on the project without having to worry about to configuration, whereas Typescript being build on Javascript requires some configuration before the project.

User Stories

User Interface

- As a player, I want to be able to see the current layout of the board, so that I know the current state of the game.
- As a 9MM board, I want to be able to see the number of pieces that have been taken, so that I know the progress of the game.
- As a player, I want to be able to tell whose turn it is, so that I know who needs to play their move.
- As a player, I want to be able to see the vacant spots on the board, so that I can decide where to put my piece.
- As a 9MM board, I want to be able to see the mills that are present on the board, so that I know which pieces are protected.
- As a player, I want to be able to see which pieces I can remove from the board when I get a mill, so that I can make better decisions.
- As a player, I want to be able to navigate the different game modes of the game with a User Interface, so that I can easily start a new game or choose another mode.

Gameplay

- As a 9MM piece, I want to be able to slide into vacant spots if I'm not a mill, so that I can form a mill
- As a 9MM piece, I want to be able to be placed on the board, so that moves may be played.
- As a player, I want to be able to move my piece to any part of the board when I only have three pieces left, so that I can play my moves from a weak position.
- As a player, I want to be able to remove a piece of my opponent's when I form a mill, so that I can win the game
- As a player, I want the game to end when me or my opponent has fewer than three pieces, so that a winner can be found.
- As a player, I want the game to end when me or my opponent has no more legal moves, so that a winner can be found.

Miscellaneous

- As a developer, I want to be able to test the implementation of the game in the terminal, so that I can verify and fix any errors easily.

Advanced requirement

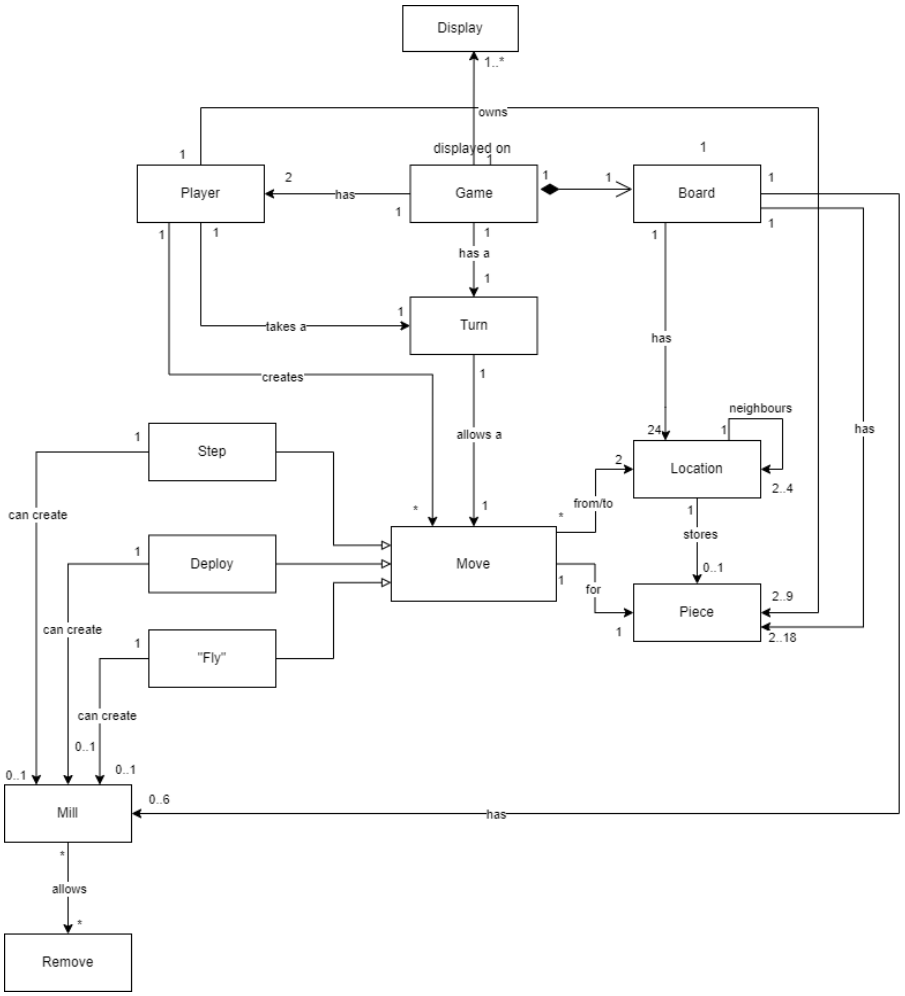
- As a player, I want to be able to undo my last move, so that I can play a better move instead.
- As a player, I want to be able to save the state of my game, so that I can continue playing later.
- As a player, I want to be able to reload the state of a previous game, so that I can continue playing it.
- As a game developer, I want to be able to save the state of a game, so that I can diagnose bugs when I find them.
- As a player, I want to be able to see what my previous move was, so that I can make a decision on whether to undo or not.

Basic Architecture

Domain Entities

- **Display:** the interface where the game is viewed on
- **Game:** the game itself.
- **Player:** a player playing one side of the game. There are 2 players in the game
- **Board:** the board of the game, on which the pieces are placed
- Line: a line on the board which connects 3 locations/points. There are 16 line on the board.
- **Location:** a point on the board (connected to other Locations by a Line). There are 24 locations in a board, which can be occupied by a token.
- **Token/Piece:** the piece played (each player has 9)
- **Turn:** describes the current state in the game.
- **Move:** a move played by the player.
- **Deploy:** a type of move describing when a token/piece is deployed onto the board
- **Step:** a type of move describing when a token/piece is moved across one line to an adjacent Location
- **Fly:** a type of move describing when a token/piece flies across the board
- **Remove:** a type of move describing when a token is removed from the board. It happens when a mill is formed.

Initial Domain Model



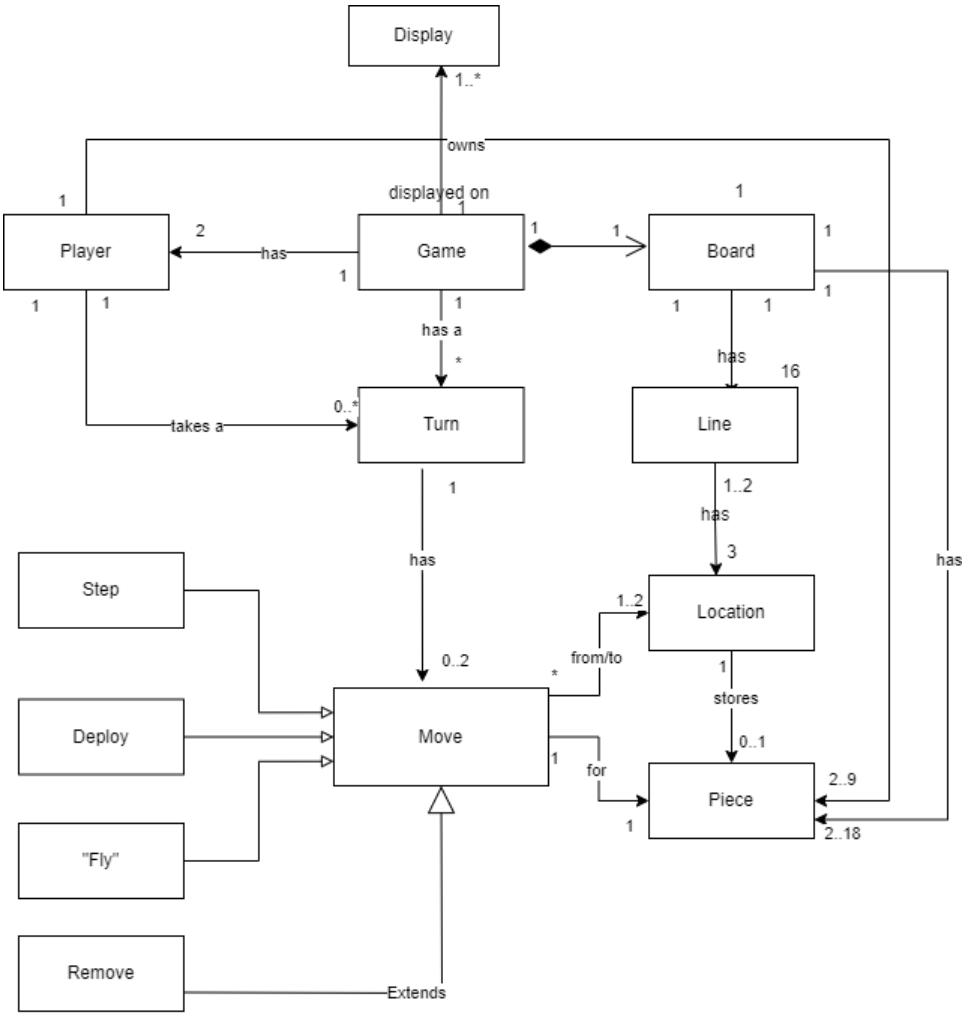
We made a change to our initial domain model as shown above by removing the mill entity and instead representing mills using the lines on the board and the locations of the pieces.

Initially, we used a unary relationship within the Location entity to represent the position of a piece relative to other positions, such as other pieces that can be on the top, bottom, left or right of the location. However, we discovered a limitation with this approach when it came to identifying when a mill had been formed. As the Location entity only knows about adjacent positions, it cannot determine if a mill has been formed across non-adjacent positions.

To overcome this issue, we decided to use the Line entity to represent unique positions on the board. By doing so, we can associate each line with the three points that form a mill, making it easier to identify and track mills on the board.

Using the Line entity also allows us to simplify our domain model by removing the need for the Mill entity altogether. Additionally, we can still use the Location entity to represent the positions of the pieces relative to the lines and other pieces, allowing us to determine which mills a piece belongs to.

Final Domain Model



Justification

- To highlight how the game is displayed to the user, the game object has a associates relationship with display object with a one to many relationship as a game can be displayed into multiple displays.
- A composite relationship is used between the Game entity and the board entity as a board doesnt exist when the game is finished.
- The board has a one to many relationship with the piece object as that is used to represent the individual pieces that are placed by the player on the board. A player has 2 to 9 pieces that can be on the board.
- To represent the a mill and the position of the pieces, the line entity and the location entity are used. A board has 16 lines including 12 from each 3 squares and 4 across the center of the board. Each line has 3 locations in which the location can be either free or occupied. A line with all 3 location occupied by pieces from the same team represents a mill. As the lines in the game overlapped, a location can overlap and be

store in up to 2 lines e.g the middle location of a line always overlaps with another line.

- The relationship between the Game entity and the Turn entity is used to represent the progression of the game which each turn represents a different stage of the game. The turn entity can be used to restore the game state to a certain stage allowing for the undo or save functionality. A player also has 0 to many turns representing the moves a player has played in the game.
- The move object represents the actions of the player in each turn. A move can involve changing the occupation of 1 or 2 locations from either vacant or occupied either by removing a piece or placing a new piece to that location or move a placed piece from a old locatio to a new location. As a turn can have more than one moves, a move is only restricted to one piece. There are a few variation of moves including step, deploy, fly and remove.

Basic UI Design

Initial game state

The game begins with an empty board and nine tokens to the side for each player.

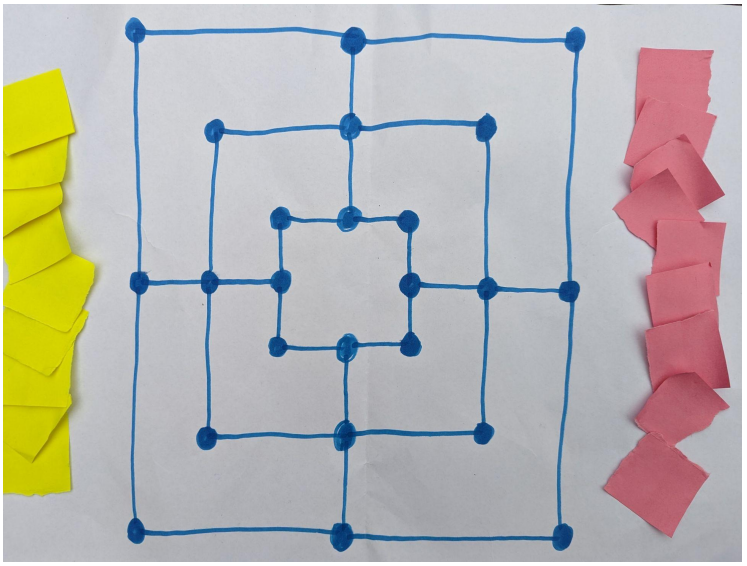


Figure 1

Phase 1: Placing pieces

Each player takes turns placing a piece on a point on the board.

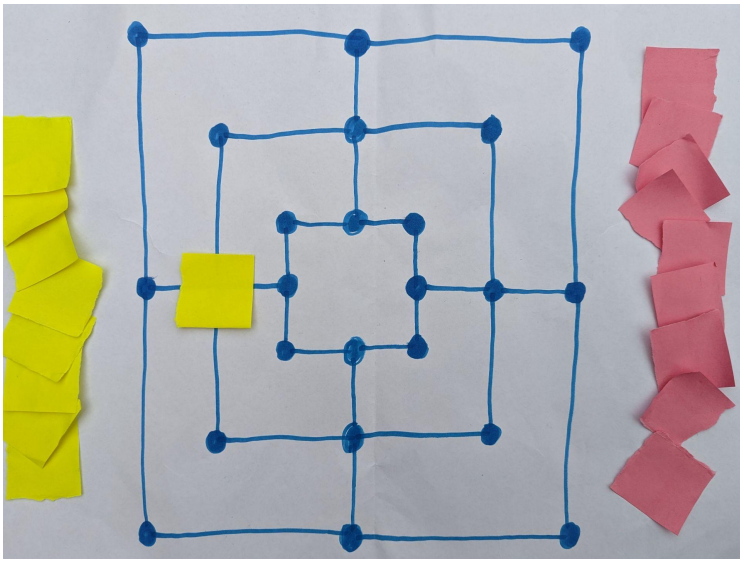


Figure 2

Forming a mill

At any point in the game, if the player's move forms an array of three pieces in a row (as seen in Figure 3), this is a mill (or more rather, a mill has been formed). When a mill is formed, the player who formed the mill can remove one of the opponent's pieces. Note: this piece being removed cannot be in a mill unless there are no other pieces to pick from. Figure 4 shows the board after an opponent's piece has been removed.

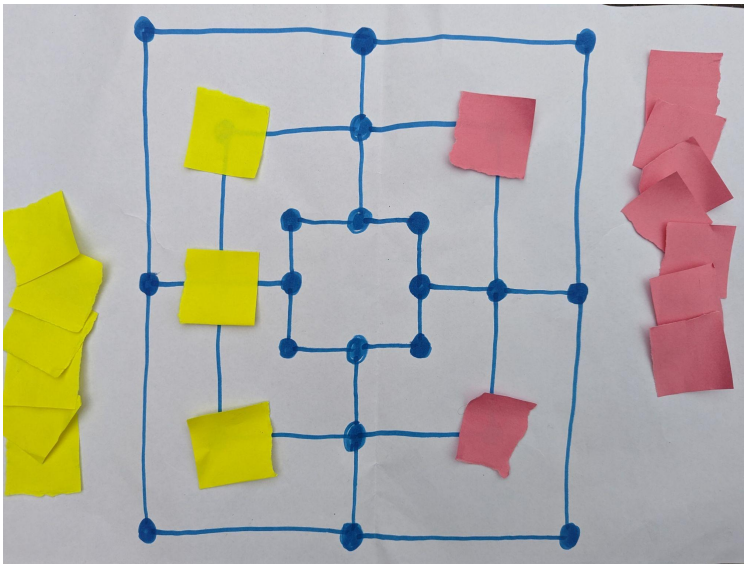


Figure 3

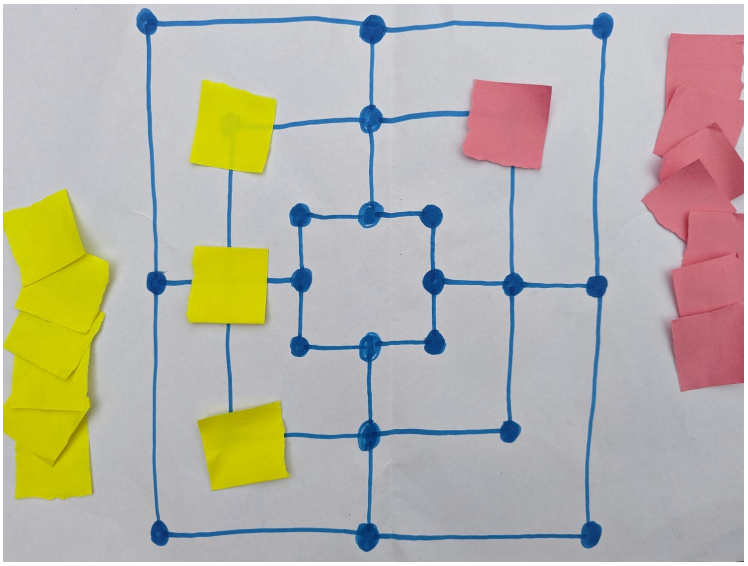


Figure 4

All pieces placed

Once all pieces have been placed (as shown in figure 5), the game moves to the second phase. Each player now takes turns moving a piece across a line to an adjacent point. Rules for mills still apply. Figure 6 displays the board after yellow has moved the top right piece to the middle right.

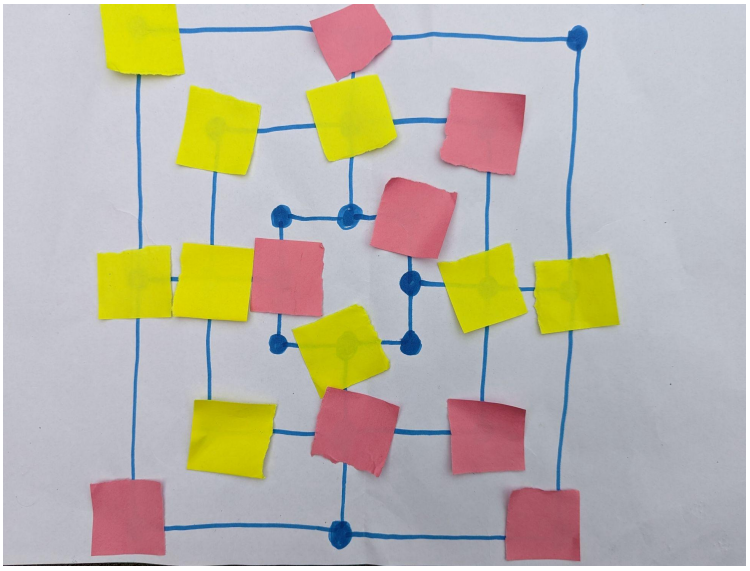


Figure 5

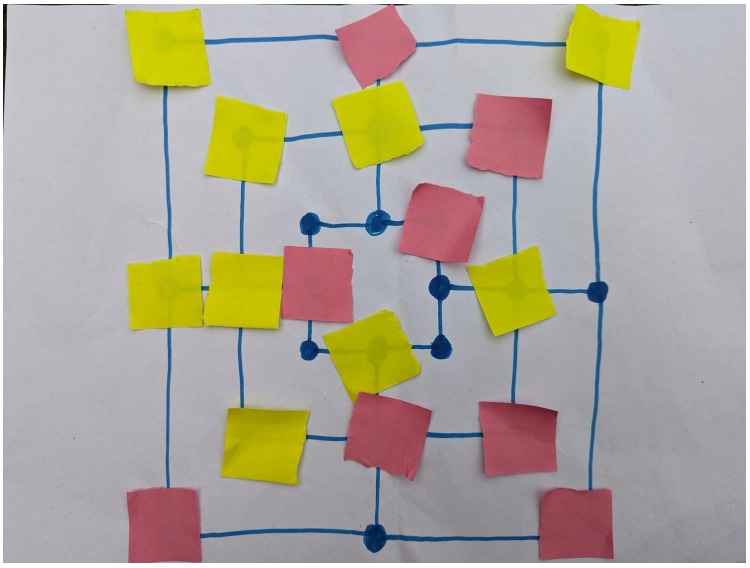


Figure 6

Flying

When a player has only three pieces left, these pieces can fly! Figure 7 shows a game where pink only has 3 pieces left. Figure 8 is the board after the bottom left piece has taken off and flown away!

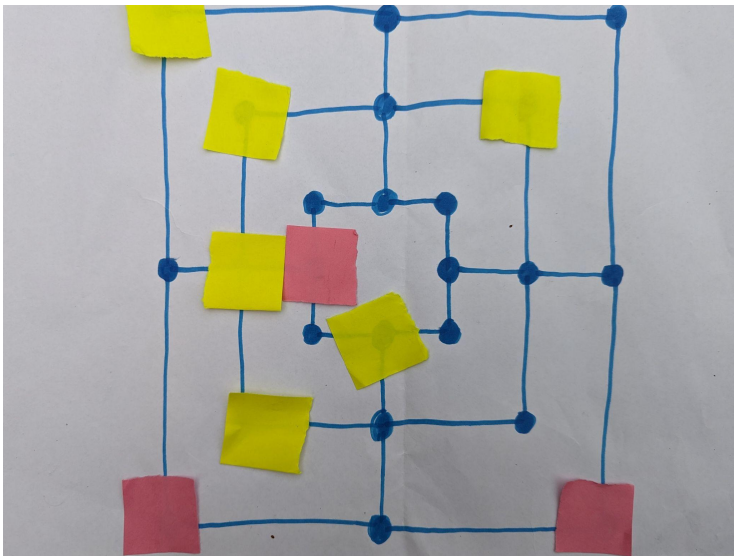


Figure 7

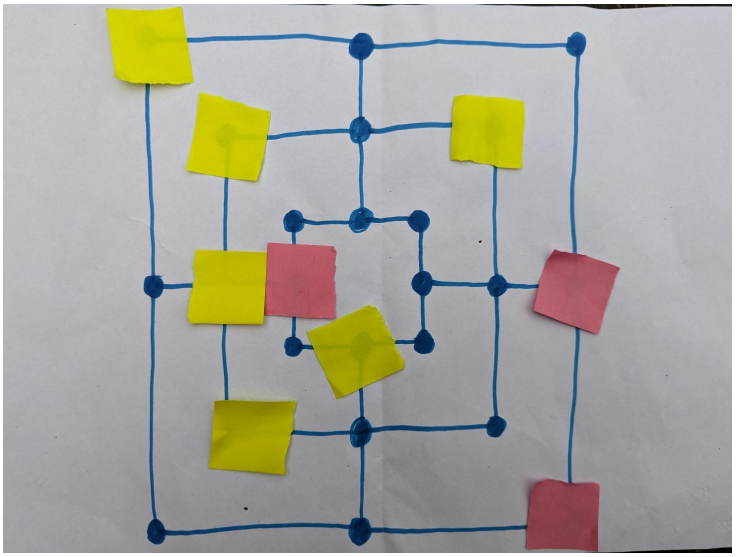


Figure 8

Win conditions

A player wins when their opponent has two pieces left on the board (see figure 9). A player can also win when it is their opponent's turn and their opponent has no possible moves (see figure 10)

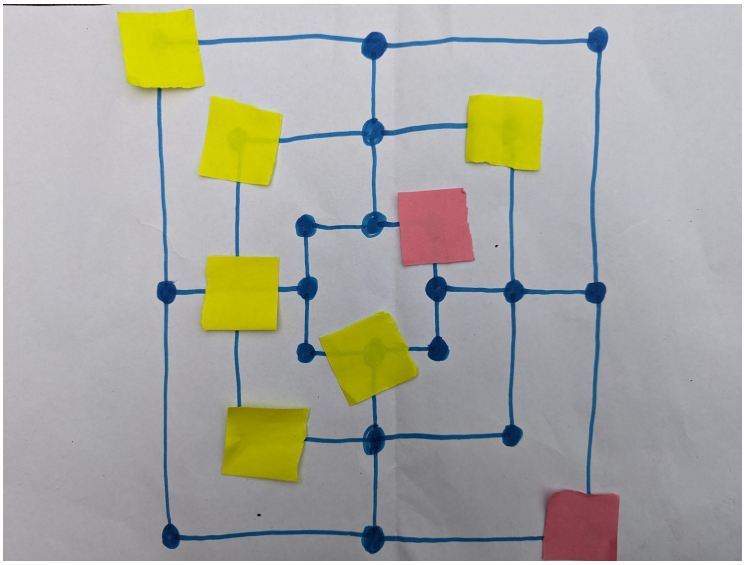


Figure 9

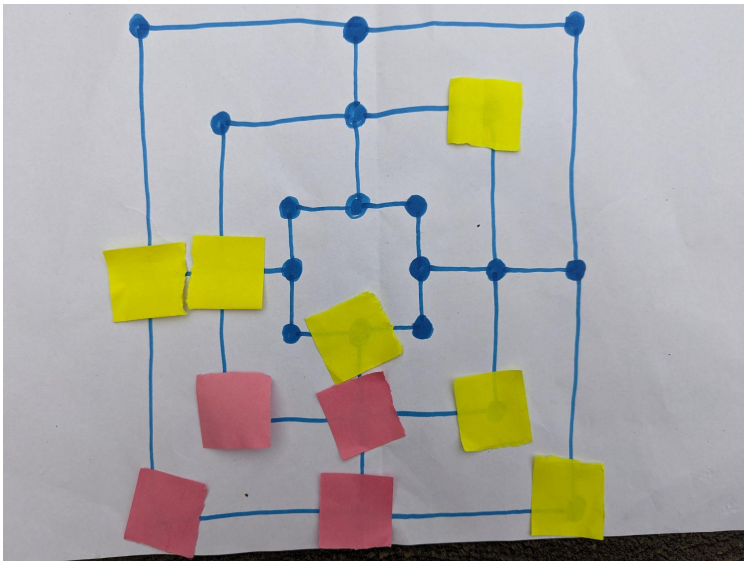


Figure 10