FIT3077 Sprint 1 Deliverables

Team: Red Kat

By:
Linden Beaumont - lbea0010@student.monash.edu
Aaron Boak - aboa0005@student.monash.edu
Zayn Hatter - zhat0011@student.monash.edu

*Photoshop was approved for our group due to extenuating circumstances

Team Information

Team Membership:

- Linden Beaumont

  Contact:
  Email - bea0010@student.monash.edu

  Technical and professional strengths:
  Experience in Python and UFT (testing automation software). Works well in a team (did IT Professional Practice FIT1050)

  Fun Fact:
  I live in Frankston

- Zayn Hatter

  Contact Details:
  E-mail - zhat0011@student.monash.edu
  Discord Username - zn#4453

  Technical Strengths:
  Technology stack consists of Java, Python, HTML, JavaScript, Swift & SQL

  Professional Strengths:
  Attentive listener and good communication skills

  Fun Fact:
  I've built and modified PCs

- Aaron Boak

  Contact:
  Email - aboa0005@student.monash.edu
  Discord - 4_za_4s#5922

  Technical and professional strengths:
  Experience in Java, Python C, Worked in professional environments, Did IT professional practice,

  Fun Fact:
  Was MVP for my team in HTB CTF

Team Schedule and Communication:

- Team meeting Online on discord every Saturday 5pm
- On campus in the workshop
- Regular messages on Discord

In terms of messaging we decided to use Discord because we all already use it and it is easy to use. We decided not to use Facebook Messenger or Microsoft Teams because we do not regularly use them. Additionally, Microsoft Teams browser version is slow to load on some of our computers making it an accessibility issue.

Technology Stack & Justification:
We decided to use Java because it is a very common OO language that we are all familiar with.

We decided not to use Typescript even though we have all also used it before because we wanted to create a standalone application. Java enables this easily whereas Typescript would have been best if we wanted to create a webpage.

We are not using any APIs because we do not need to access them for this project.

We anticipated that we will need help with the interaction with the objects of our OO language as it can be confusing in how to best implement SOLID and DRY principles.

Advanced Feature chosen:

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.

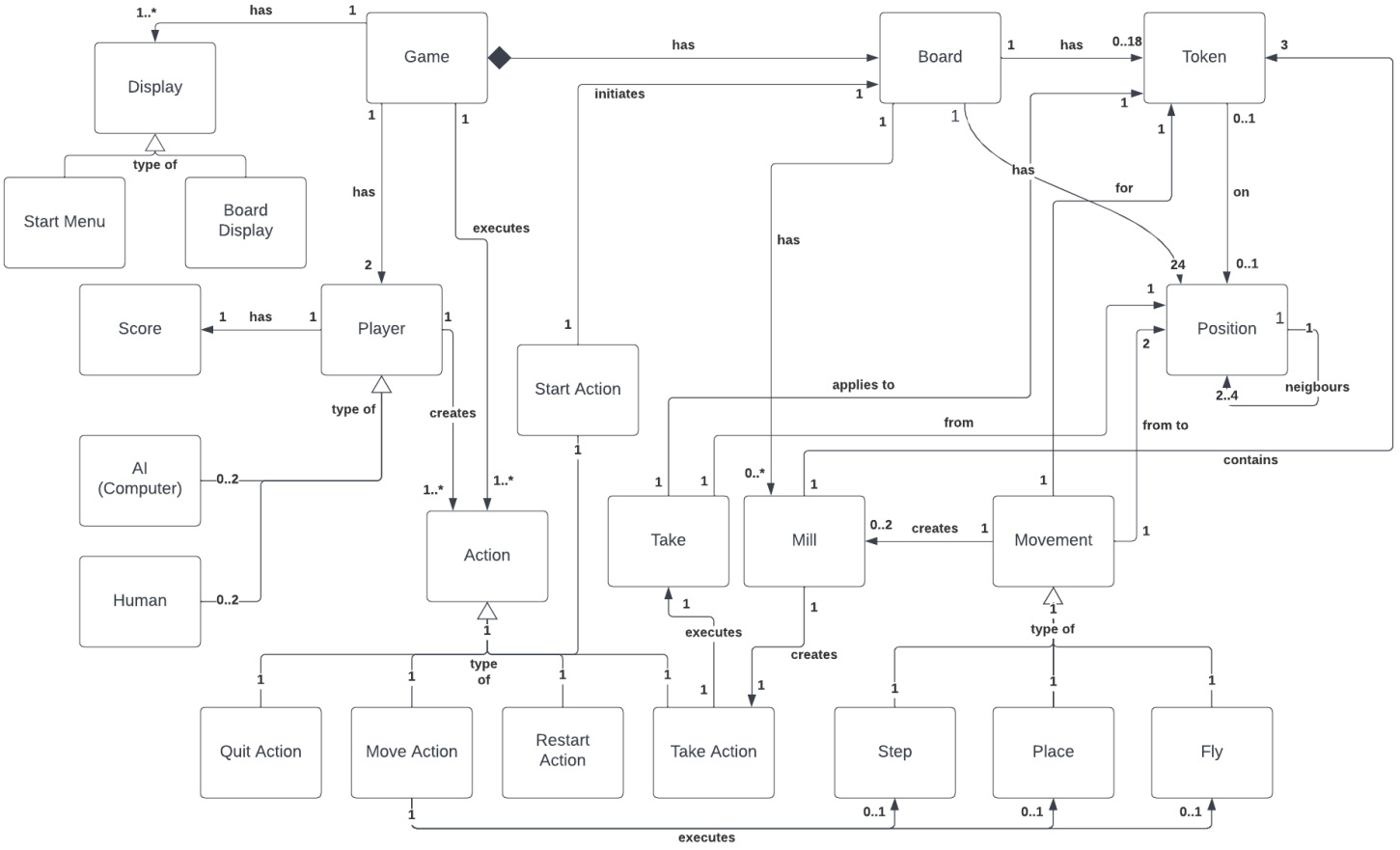
We chose this feature as it seemed the most interesting and has potential to be expanded on by creating more complex heuristics/implementations of the AI. This means we can start off with the basic implementation with random moves to get it working and then add more complexity later on.

This is perfect for sprints where you further refine the product each sprint. We can see how development goes, and then later base our AI aims on that.

User stories:

Story Name	Story Description
Main game:	
Create 9MM board UI	As a display, I want to have a board that shows up in the console and can have pieces placed onto it, so that the player can view the game
Create display for remaining pieces	As a board, I want to be able to know how many pieces a player has so that I can allocate the player the correct number of pieces
Turns	As a board of the game I want there to be turns (player1 -> player2 -> player1 -> etc) so that movement can take place correctly
Placing pieces	As a piece to be able to place myself on the board in an empty slot while there are pieces left (placing pieces removes 1 from the remaining pieces counter) so that I can go in play
3 in a row remove	As a mill I want placing three pieces in a row to allow me to remove any of my opponent's pieces so that I can create a win condition
3 in a row protection	As a mill I want that if there are other opponents pieces on the board I can't remove a piece in a 3 in a row so that I can create win conditions properly
Moving pieces	As a piece I want to be able to move to an adjacent empty slot so that I can change positions
3 Pieces left/Flying	As a player I want that if I have three pieces left then I can move my pieces anywhere so that the end game is more intense and so that I can have the correct actions
Win Condition	As a board I want the game to end when a player has reached a win condition (which can happen due to only having two pieces or being unable to move) so that the game can be won
Testing	As a board I want to ensure I work properly so that actions created by players have their intended consequences
Restart game	As a game I want to be able to restart after a win/lose so I can be played multiple times
Computer Extension:	
Implement the AI player	As an AI, I want to be able to do all actions a player can, so that I can play the game
Option for AI player	As a player I want an option to make player 1 or 2 a human or AI so that the I can play against an AI or human
AI heuristic	As an AI, I want to be able to read the state of the board, so that I can make informed decisions on my next move

Domain Model



Entities:

- *Game:* Entity that keeps track of the state of the game

- *Start Action:* starts and initialises the game

- *Action:* where the user can execute certain actions
- *Quit Action:* where the user can quite the game
- *Move Action:* where the user can execute a particular move
- *Restart Action:* where the user can restart the game
- *Take Action:* where the user can execute a take action

- *Take:* where the user can take their opponent's token from the board
- *Mill:* where the user can create a mill on the board
- *Board:* consists of the board for the game
- *Token:* represents a token that a player can have on the board
- *Position:* encompasses the position of each token on the board

- *Movement:* executes the movement for each token on the board
- *Step:* where the user can move their token by one position (step) on the board
- *Place:* where the user can place their token to a certain available position on the board

- *Fly:* where the user can move their token to any available position on the board regardless of the distance

- *Player:* encompasses the player (user) of the game
- *AI (Computer):* essentially is a computer/robot which acts as a player
- *Human:* is an actual player which can be played by a user (human)

- *Display:* UI for the user see and interact with, only humans need this
- *Start Menu:* exhibits the commencement of the game
- *Board Display:* displays the state of the board on the user's screen

Relations:

- *Movement → Position:* Movement affects 2 Positions because once a movement is made, 2 positions are required for a token to move from and to.
- *Movement → Token:* Movement is primarily built for 1 Token as it allows a token to be moved throughout positions on the board.
- *Movement → Mill:* Movement creates a Mill because mills (3 pieces in a row) are only created after a piece has moved into a mill.
- *Mill → Take Action:* Mill creates Take Action as mills provide a user with the opportunity to take or remove one of the opponent's tokens from the board. The take action is only created once when the mill is first created.

- *Start Action → Board:* Start Action initiates Board because it starts the game which consequently creates the board for the user
- *Board → Token*: Once the Board is initiated, along with the Board, Tokens for each player are also created. There can be 0 tokens at the start and at most 18 as each player only has 9 tokens.
- *Board → Position:* The Board also concurrently creates Position in order to create space for tokens on the board. Every board has 24 positions.
- *Board → Mill:* Board has an association with Mill as mills can exist on the board and need to be kept track of when they are created and destroyed.

- *Mill → Token:* Mill has an association with Token because the primary component behind the creation of a mill is tokens. Mills contain 3 tokens as it makes them untakeable in normal circumstances.
- *Take → Token:* Take directly applies to Token because Take consists of taking/removing a single token from the board.
- *Token → Position:* Similarly, Token has an association with Position as a token eventually takes a particular position on the board as the game progresses. A token doesn't need a position to exist in the game, which is why a token can have 0 or 1 positions.
- *Position → Position:* Position has its own association as multiple positions on the board can have other positions surrounding it, meaning that it is surrounded by neighbours. This is related to how a mill will know to exist.

- *Game → Board:* Game has a Board to provide the user with a space to play the game.
- *Game → Player:* Similarly, the Game has 2 Players which are part of the main components behind the creation and execution of a game.

- *Game → Display:* Game has a Display which provides the user with a screen to interact with in order to successfully play and complete the game.

- *Player → Score:* Player has a Score as it helps in keeping track of how many games each player has won.
- *Game → Action:* The game entity will execute any action that is created within it. Game will validate these actions when executing them to make sure the state of the game is progressing as intended, i.e. only place actions at the start of the game.
- *Player → Action:* Player creates Action because once the player interacts with the board and tokens an Action has to be created in order to successfully execute and complete that specific action.
- Players create actions not just movements because we want more than just moving pieces in the game.
- The player will choose an action when the game is waiting for one, Game dictates what actions can be done on any given turn.

- *Move Action → Step, Place & Fly:* Move Action executes the child classes of Movement (Step, Place and Fly) because when a player selects an action to move a token, they will be provided with those three movements as potential options. Only 1 of these potential actions can be executed in a given turn.

Inheritance:
- *Action → Quit Action, Move Action, Restart Action & Take Action:* Action is the parent class of Quit Action, Move Action, Restart Action and Take Action as these classes are all intended to be actions and will extend and inherit attributes such as variables and methods from the Action class which is indicated by the inheritance association within the diagram.

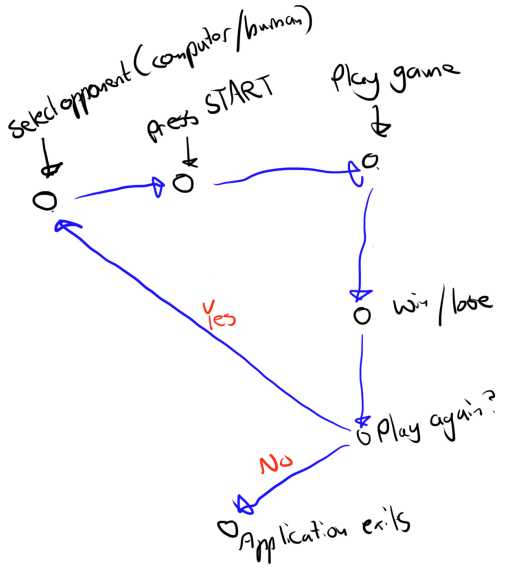
- *Player → AI (Computer) & Human:* Player is the parent class of AI (Computer) and Human as these classes extend and inherit attributes such as variables and methods from the Action class which is demonstrated via the inheritance association within the diagram.

- *Display → Start Menu & Board Display:* Display is the parent class of Start Menu and Board Display as these classes extend and inherit attributes such as variables and methods from the Display class which is conveyed by the inheritance association within the diagram.

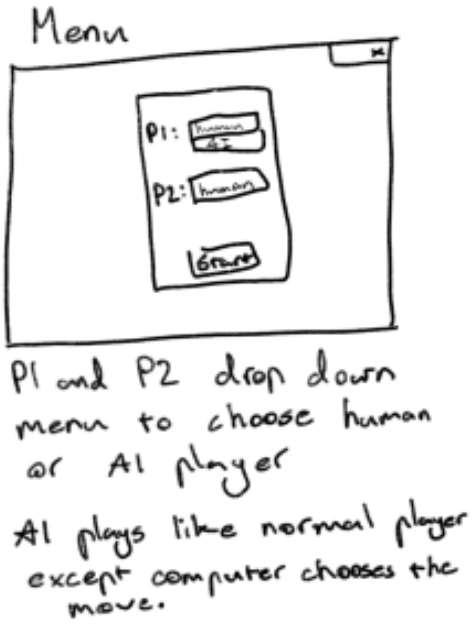
- *Movement → Step, Place & Fly*: Multiple movements are available as there are many different ways of moving a piece depending on the state of the game.
- The user can either select to Step, Place or Fly their token which is directly correlated with Move Action because the user is concurrently also executing a movement action.

Basic UI Design

Game cycle



Start Menu



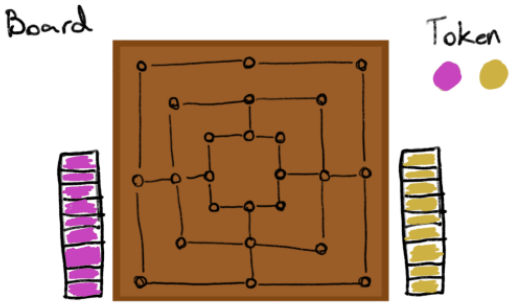
P1 and P2 drop down menu to choose human or AI player

AI plays like normal player except computer chooses the move.

Game over menu

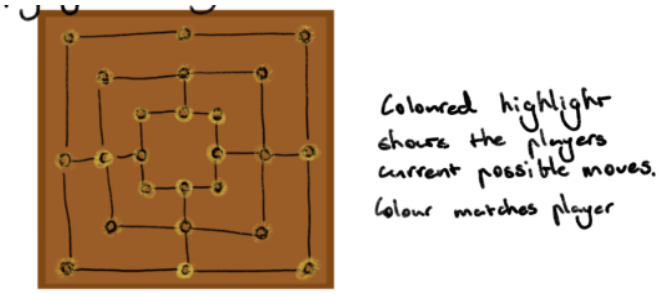
Token Design (chose piece design not used in other diagrams for simplicity)

Board design

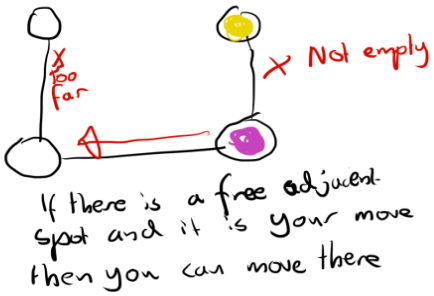


Here each circle is an available position. Positions connected with black lines are adjacent.

Showing available moves

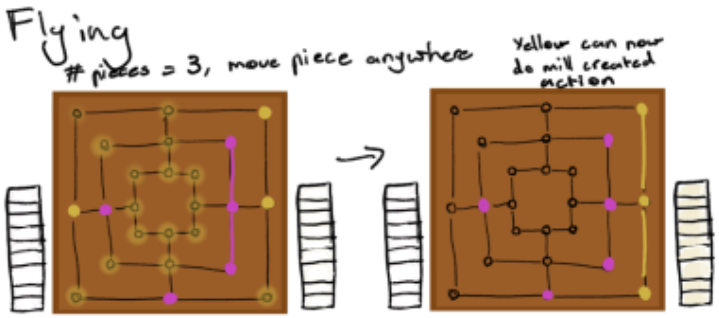


Moving pieces normally

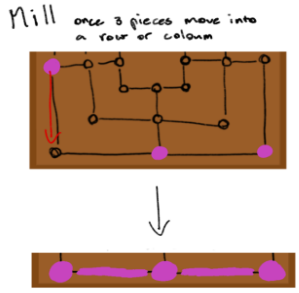


If there is a free adjacent spot and it is your move then you can move there. You can only move once per turn.

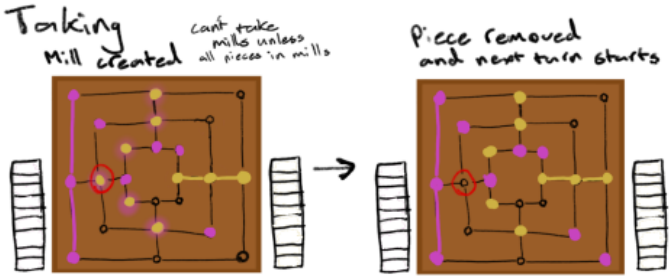
Moving when a player on has three pieces left



Mill definition (3 in a row)



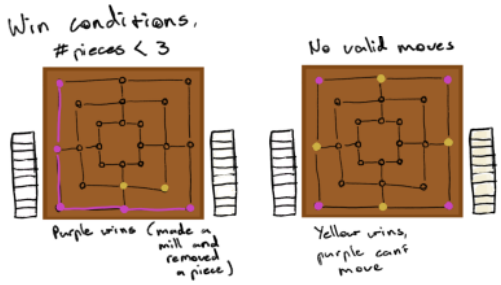
Using a mill (remove opponent's piece)



You only remove a piece on the turn you get a mill. On the turn you make a move to create a three in a row.

If purple just got a mill he can remove one of yellow's pieces that is not in a mill. If yellow only has pieces in a mill. If yellow only has pieces in a mill, purple can remove any piece.

Win Condition



Start of Game