MONASH University

<u>Team Information</u>

JTR Industries Pty Ltd

Last updated: 1/04/2023

Table of Contents:

Team Members:

This section details technical information relating to all current members of the company "JTR Industries Pty Ltd". Contact information is private and intended to be shared only amongst relevant stakeholders and internal company employees.

Thanh Nguyen (Jessica):

Jessica Nguyen is a third-year software-engineering student, currently interning at 'JTR Industries Pty Ltd'. She has had previous experience with agile project management and works well within structured team environments. Jessica has a very keen eye for aesthetics as she also studies design.

Technical / Professional Strengths:
- Java
- Javascript
- HTML
- CSS

Contact details:
- Email: tngu0151@student.monash.edu
- Facebook: https://www.facebook.com/rubynguyen.1999

Tuan Le:

Tuan Le is a third-year IT student, currently interning at 'JTR Industries Pty Ltd'. He has had previous experience with agile project management and works well within structured team environments. Tuan is currently writing an epic fantasy novel as a hobby.

Technical / Professional Strengths:
- Java
- Javascript
- Python
- HTML
- CSS
- C++

Contact details:
- Email: mlee0096@student.monash.edu
- Facebook: https://www.facebook.com/tuan.leminh.18400

Ryan Tran

Ryan Tran is a third-year software-engineering student, currently interning at 'JTR Industries Pty Ltd'. He has had previous experience with agile project management and works well within structured team environments. In his spare time, Ryan can be found hanging out with his Calico cat, "Kuja".

Technical / Professional Strengths:
- Java
- Javascript
- Python
- HTML
- CSS
- React

Contact details:
- Email: rtra0013@student.monash.edu
- Facebook: https://www.facebook.com/ryan.tran.50702/

Team Logistics:

This section details the intended meeting schedule for the current project, "Nine Man's Morris". It also discusses how division of workload amongst JTR Industries team members will be distributed and managed fairly.

Figure 1. Availability matrix of all JTR Industries members. Created on when2meet.com
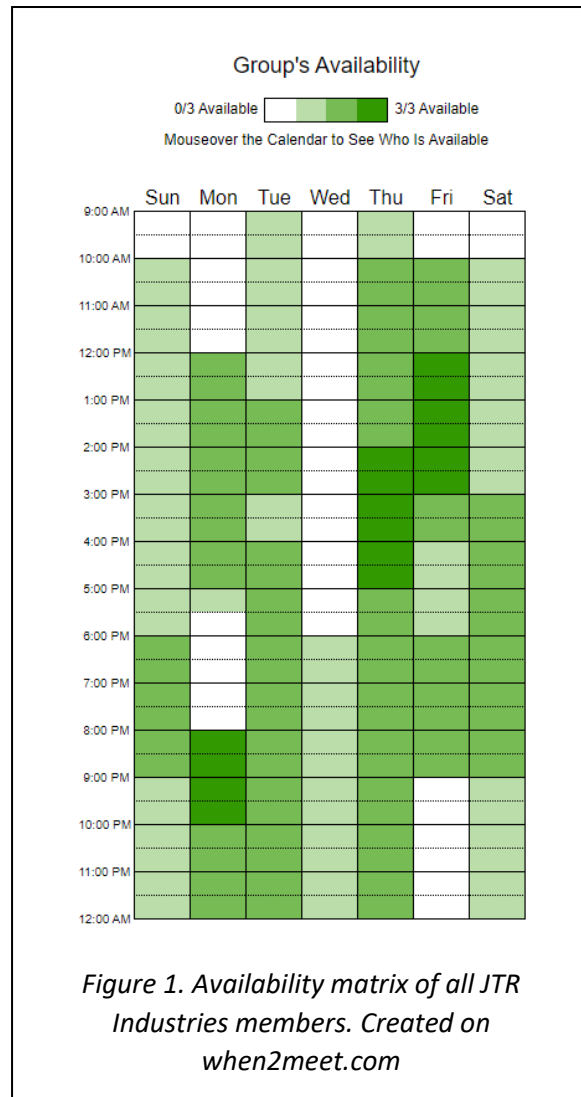
Meetings:

This team will meet regularly in-person at the following times:

- Monday (8:00pm – 9:00pm)
- Thursday (2:00 – 3:00 pm)

These times were chosen based on team member availabilities (see figure 1). These times are especially convenient as they follow in-person applied and consultation sessions.

Please note that additional meetings may be organized on a week-to-week basis, dependant on workload.

Division of workload:

Workload will be distributed evenly amongst team members at the start of each sprint. Team members will meet in person after identifying all user stories pertaining to the current sprint.

User stories and features will then be assigned to team members, taking into consideration the complexity and nature of the work.

Care will be taken during this meeting to ensure each member does an appropriate amount across all facets of the project (documentation, coding, wireframing etc.).

Technology Stack:

This section details the proposed technology stack to be used for the "Nine Man's Morris" project. Included is the justification for each choice and alternative considerations.

Programming language:
- Java

Java was chosen as the main programming language for this project due to several reasons. Primary amongst them, was because all members of JTR Industries have had previous experience coding with Java through FIT2099. Java is also at its core a fully object-oriented language which very much suits this unit, as it will help enforce sound class-based object-oriented design.

Python was also considered as the main programming language due to two team members having experience with it. Python's relatively simple and concise syntax also made it an appealing choice for this project as it would have helped with the code readability. However, with Python being a multi-paradigm language, it does not enforce object-oriented principles to the same degree as Java. Therefore, the final consideration was given to Java.

API:
- Java2d
- Swing (JFC)

We have not yet finalized our choices for the API technologies we would use for this project but listed above are APIs we have identified as being potentially useful.

Considering that the design of "Nine Man's Morris" is comprised of relatively simple shapes, we believe the Java2d API would be appropriate for this use case. This API is more light weight than its cousin Java3d (which was also considered but deemed overkill for our use purposes.

Swing is our primary candidate for our GUI application needs. We considered Swing alongside Java AWT as both could potentially achieve the same things. We ultimately went with Swing as, it is comparatively more light-weight, powerful and faster than AWT. Swing also supports MVC patterns (unlike AWT) which synergises well with OOP.

Design:
- Figma
- Lucidchart

With one of our team members already having experience with using Figma (Jessica Nguyen), thought that Figma would be a good choice for our Lo-Fi Prototyping needs. Figma is a web-based design tool, with many useful functionalities for designing prototypes.

Chief among these tools is its collaborative functionalities, which allows for multiple users to work on the same design page with live syncing. This was especially useful for our purposes, as having the shared workspace would allow for our team members to easily see which elements have already been implemented. Additionally, by having all Lo-Fi drawings be consolidated in a single workspace, we thought it would help us keep a coherent and cohesive design style, as each drawing could easily be referred to.

Lucid chart was our chosen tool for creating our domain model. This was a very easy choice, as all team members had previous experience with using the program, which would likely eliminate any hiccups with learning how to design on a different platform. Additionally, Lucid chart also supports the importation of third-party shape libraries, such as "UML class diagrams" and "entity relationships" which would have been especially useful for our modelling purposes.

User Stories:

- As a User,
  I want to have a simple, minimal UI style,
  So that it will be easy to intuitively use.

- As a User,
  I want to have a board with great contrast,
  So that my colour-blindness is not an issue with viewing the board.

- As a Player,
  I want to have a game tutorial,
  So that I can learn how to play the game.

- As a Player,
  I want to be able to choose between human and computer opponents,
  So that I can have different game experiences.

- As a Player,
  I want to experience three difficulty levels of computer opponent,
  So that I can enjoy a game, appropriate to my skill level.

- As a Player,
  I want to move my token,
  So that I can form three tokens in a row.

- As a Player,
  I want to see a match history record,
  So that I can review my play and improve.

- As a Game Board,
  I want to recognize when a "mill" (three in a row) occurs,
  So that a player can be allowed to move a piece.

- As a Game Board,
  I want to recognize when all 18 pieces have been set,
  So that I can allow for sliding and removing moves to be played.

- As a Game Board,
  I want to keep track of all a player's mills,
  So that a piece cannot be illegally removed from a mill.

- As a Game Board,
  I want to recognize when a player has fewer than three pieces left (in endgame),
  So that a winner can be chosen.

- As a Game Board,
  I want to recognize when a player has three pieces left (in endgame),
  So that, that player can unlock "Jump" moves.

- As a Game Board,
  I want to ensure that moves are made in alternation between players,
  So that a fair game can be played.

Domain Model:
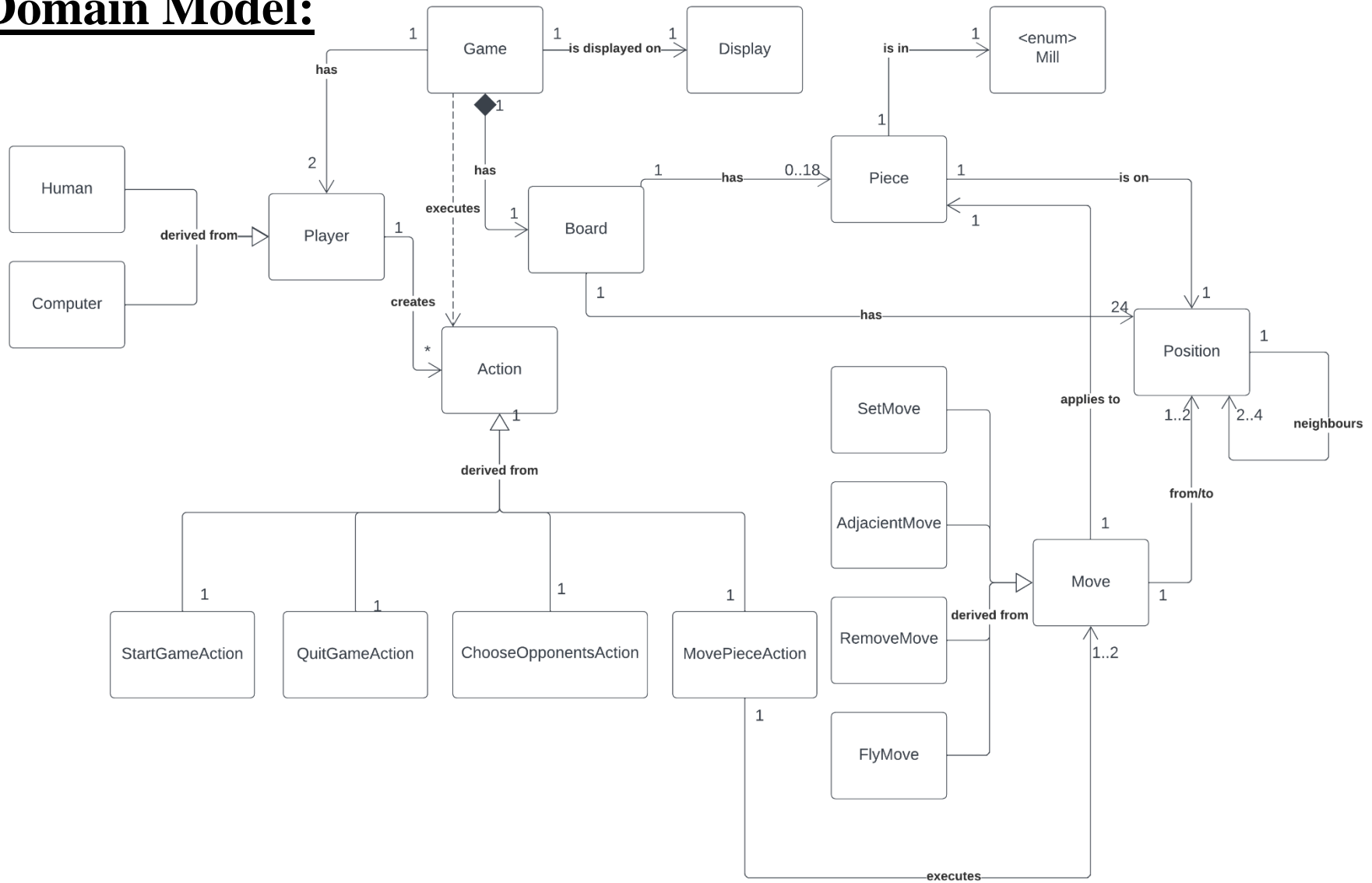
Domain Model Rationale:

- Player
- Human
- Computer

Having chosen "Computer" as our advanced requirement, we recognized that there were many similarities between the "Human" and "Computer" players. They will likely execute the same types of moves and actions and share many attributes and methods. In seeing this overlap, we determined that creating a new Abstract entity called "Player" and leveraging the inheritance model for this relationship would be appropriate.

This "Player" entity will capture many of the similar responsibilities between "Human" and "Computer", and thus allow us to only implement them once.

- Action
- StartGameAction
- QuitGameAction
- ChooseOpponentsAction
- MovePieceAction

We also recognized that a "Player" may create various types of actions which are executed by the game. Therefore, we chose to capture these various actions under an abstract class called "Action". This not only eliminates code repetition (adhering to DRY principles), but also reduces the number of dependencies coming from the "Game" class (which would otherwise need to have dependencies with each concrete action class).

- Move
- SetMove
- AdjacentMove
- RemoveMove
- FlyMove

Our reasoning behind the design surrounding "Move" and its concrete classes, follows the same reasoning behind the "Action" classes. With various phases and contexts within "Nine Man's Morris" allowing for different types of movements, we decided that following a command pattern in implementing "Moves" would be very appropriate. There would otherwise be far too many responsibilities consolidated within one class which would need to execute on the game's state (Board, Pieces, Position) across a wide range of outcomes.

- Game

A "Game" class is typical and appropriate for most implementations of this nature, and Nine Man's Morris is no exception. There must be a central point from which all choices and outcomes are coordinated through or executed on. Nine Man's Morris as a concept in our design, cannot itself exist without boards and pieces, which the "Game" is ultimately responsible for initializing. "Game" is also home to all the central logic which bedrocks Nine Man's Morris.

A human player is also not capable of playing the game well, without seeing some visual representation of the game state. Thus, "Game" is logically needed as an intermediary to pass consolidated meaningful information onto "Display" for visual representation (as display by itself knows nothing).

- Display

Where "Game" homes the program's logic, "Display" homes the game's graphics. Display's responsibilities are restricted to only the visual technicalities, hence its separation from the "Game" class. "Display" could be technically over-loaded, as it may setup graphical APIs (Java2d, Swing, SDL), in addition to other display elements. Therefore, it may need to be further compartmentalized and broken to better fit object-oriented design. This will be fleshed out later, when the team becomes more experienced and knowledgeable of the requirement's nuances.

- Board

The "Board" is an essential component of the Nine Man's Morris Game. It is important to note the composition relationship between "Board" and "Game". This is because we likely need to access much of the "Board" class's behaviors, within the "Game" class. A litmus test of the Liskov Substitution Principle on the "Game" and "Board" relationship (which is failed) can also be used to rule out inheritance as a potential relationship type. Furthermore, thinking intuitively, a "Board" and "Game" relationship is not like a typical inheritance relationship like "Car" and "Vehicle".

- Piece

Pieces too are essential components of the Nine Man's Morris game.

- \<enum\> Mills

We recognized that a key mechanism of Nine Man's Morris, was that board pieces in "mills" would not be permitted to move or be removed. Therefore, it is important to have some sort of identifier for "Pieces" to hold, to be able to recognize their mill status.
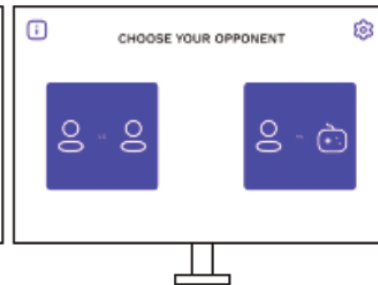
- Position

"Position" will be a class that denotes various board location points. One important design note is that "Position" may have associations with itself, through which its neighbors' positions are noted. This is potentially helpful in identifying open spots for adjacent moves and chains of pieces which could form mills.
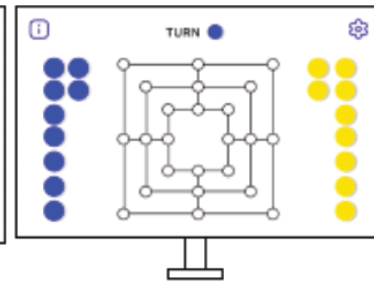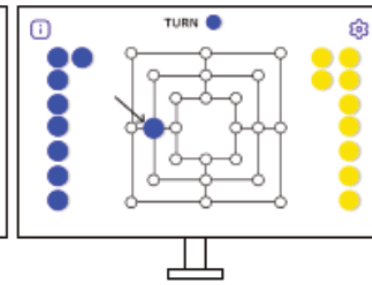
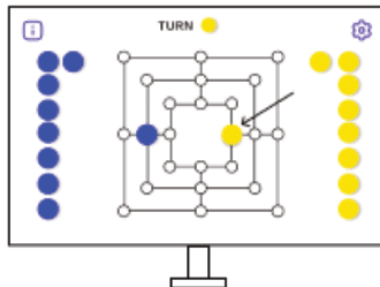# Basic UI Design:

START PAGE
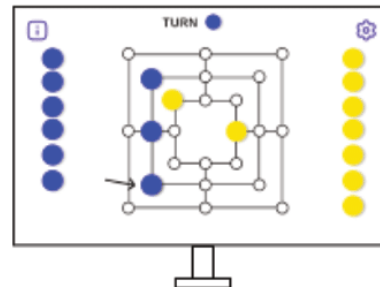
CHOOSE OPPONENT

GAME START

GAME PLAY - PLAYER TURN

ℹ️ Stands for games rule

Phrase 1: Setting up the pieces
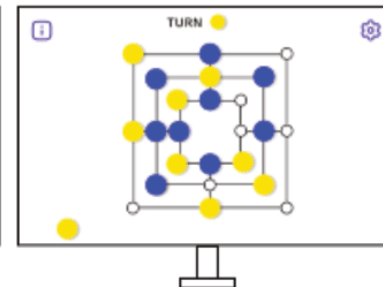Here the player with the blue pieces goes first. The piece can be put at any empty points on the board.
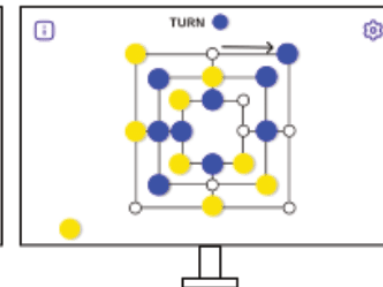
GAME PLAY - COMPUTER TURN

GAME PLAY - MILLS

If any player forms a 'Mill' on the board (3 pieces on a row or column), they can choose any pieces of the opponent that is not in a mill to remove from the board.
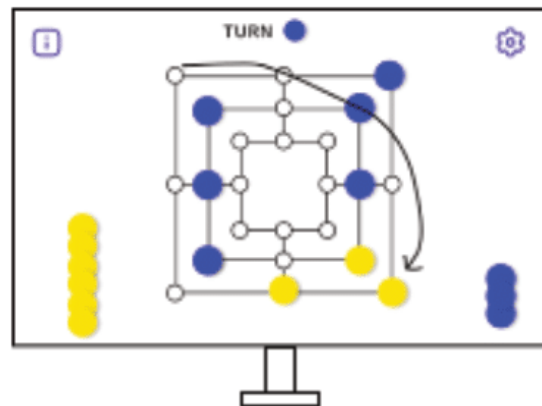
GAME PLAY - ALL PIECES ON BOARD

The players will alternately place pieces on the board, until they run out of pieces.

GAME PLAY - MOVE PIECE

Phrase 2: Moving pieces
One all players have completed placing their pieces, they will move their pieces to its empty adjacent squares. Each player can only move one of their pieces per turn and only move them vertically or horizontally following the lines on the game board.

GAME PLAY - FLYING PIECE

Phrase 3: Flying
When player only has 3 pieces on board, they can 'FLY' the piece to any position on board to form MILLS.
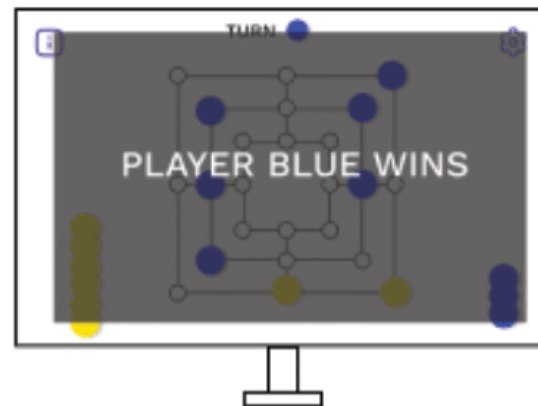
WIN - LOOSE ANNOUNCEMENT

Win - Loose
Player will win when they only have less than 3 piece.