FIT3077, Assignment 1, Sprint 1
Due: 5pm Thursday, 6 April 2023

Made by:
Team Bug Busters

Table of contents:

1. Team information

- Team Bug Busters

Members:

- Team member information
  - basic info like name, contact detail
  - Strengths and weakness
  - Fun fact about yourself

No.	Name	Contact	Strengths	Weakness	Fun Fact
1.	Bryan Brady	bbra0010@student.monash.edu	Familiar with Python, C++ (mostly), Java, have done fit2094, fit2081, etc.	Not so experienced with java OOP programming.	It took me 3 days of suffering to finally defeat Malenia.
2.	Tanzim Islam Khan	tkha0014@student.monash.edu	Intermediate with OOP as done FIT2099 last semester	Less experience using python for OOP	I can fly planes

3.	Glenn Eric Wenardy	gwen0001@student.monash.edu	Familiar with MVC, js, database, have done FIT2099	No experience with coding UI game	I can beatbox

- Meeting and work schedule for team + how distributed workload is managed

Meeting schedule & Workload distribution:

Meeting Agenda 1:

Date: 31/03/2023

Time: 8:00pm - 9:00pm:

Location: Discord voice call.

Attendants: Bryan Brady, Tanzim, Glenn Eric

Agenda:

1. View all the tasks required to do
2. Discussion of how the assignment should be done
3. Distribute links for important diagrams for the assignment.

Meeting Agenda 2:

Date: 05/04/2023

Time: 9:00pm - 10:30pm:

Location: Discord voice call.

Attendants: Bryan Brady, Tanzim, Glenn Eric

Agenda:

1. Clear up any confusions about any problems
2. Add on important bits of information such as the team information and the technology stack + justification

Workload Distribution:

We have decided in our standups that we would be working in each part equally, helping each other out if any problems arise and communicating it in a clear and concise manner on the discord chat room.

Technology stack and justification

- Java, JavaFX, IntelliJ, OpenGL

    - Technology stack and justification
        - We decided to use Java as our programming language to make code the game as most of us have the most experience in java programming
        - Also, Java is well known for its powerful object oriented programming supports as well as making 2D games
        - We also use JavaFX in deploying our application on desktop
        - As for graphic game rendering, we decided to use OpenGL as it is one of the well known API's. Thus, having more community support when we run into errors later

These are the current technologies stack that we think we might need, further addition or removal of these technologies are possible.

Discarded Alternatives :

1. Python
- While python is fast for deployment, we did not end up choosing python for our project as our area of expertise does not align with it.

2. User stories

Basic requirement:

1. As a player, I want to be able to decide to be player 1 or 2 so I can decide If I want to go first or second.
2. As a piece, I want to be able to differentiate between the pieces so I can know which side I am on.

3. As a board, I want to have nodes where pieces can be placed so that the game can be played.
4. As an application, I want to be able to display the score for both players so they can see their score.
5. As a board, I want to have links where pieces can move between nodes so that the game can be played.
6. As a player, I want to know if it is my turn or not so I know when it's my turn to move.
7. As a piece, I want to be able to be placed in an empty space so the game can be played.
8. As a player, I want to know how many pieces I have to place left so I can stop my opponent from setting up a mill.
9. As a board, I want to be able to prevent any illegal moves so that the game can be fairly played.
10. As a piece, I want to know when I can slide to an open node so I can prevent the opposing piece from making a mill or to set up a mill.
11. As a player, I want to be able to take one of my opponent's pieces when I create a mill at any point in the game so I slowly make my way in winning.
12. As a board, I want to prevent a piece that is taken from returning into the board so the game is balanced.
13. As a piece, I want to be able to move to wherever open node I want so I can prevent or even create a possible mill
14. As a board, I want to end the game when there are less than three of the same pieces so I can declare who the winner is.
15. As a board, I want to end the game when an opposing player cannot move a piece so I can declare the winner.
16. As a board, if a piece moves to a section where 2 mills can be created, I want to only remove one piece from the opponent's side so I can uphold the rules of the game.
17. As a player, I want to see my score throughout the game so I can see how well I did.

Advanced requirement:

A) " Considering that visitors to the student talent exhibition may not necessarily be familiar with 9MM, a tutorial mode needs to be added to the game. Additionally, when playing a match, there should be an option for each player to toggle "hints" that show all of the legal moves the player may make as their next move. "

1. As a player, I want to play a tutorial mode so I can learn how to play the game
2. As an application, I want to be able to provide information in tutorial mode so the player can understand how the game works
3. As a piece, I want to be able to move to a predetermined node if in tutorial mode so the player can have a better understanding of how the game works.

4. As a player, I want to be able to toggle hints on or off so I can know what legal moves are possible if I forgot the rules of the game.

3. Basic Architecture

The class diagram below represents the architecture of the 9MM game:

Rationale :

1. **Application** [Dependency: **Board, Player**]

- The **application** can be considered as the starting point of the game. The game runs by running the Application module which displays the users different options which they can choose how they want to interact with the game itself. The application will allow the user to play the game with or without displaying a tutorial.

- **Application** needs to know both the board and player since that is what essentially defines the application. The "game" itself and the players.

2. **Tutorial** [Dependency: **Application**]

- This is one of the advanced requirements from the client which asks us to implement a tutorial functionality that will guide the user to play the game.

- **Tutorial** is dependent on **Application** as tutorial is run on the application and a tutorial cannot exist without an application.

3. **Player** [Dependency: **Action**]

- The game is played between two players by default. This class will be used to generate players for the game itself

- This class holds information about the players such as which colour piece they are assigned to, number of pieces left, and score.

- **Players are** dependent on **Action** because players are the ones that will be making the action, whether it is moving the pieces or killing an opponent's piece.

4. **Action** [Abstract Class]

- This class is an abstract class that controls the action of the player and it has two child classes **Move** and **Kill.**

- **Move** is an action that allows the player to move their chips to a permitted spot.

- **Kill** is an action that allows the player to remove an opponent's chips once they get a three in a row.

5. **Board** [Association: **Player**]

- This generates the board for the player to play the game in. This is obviously, if not, the most important entity since without a board, the game cannot happen at all. The board can be accessed from its coordinates.

- **Board** is associated with **Player** because the players will be moving their chips on the board and players don't have direct access to the chips, such that the board needs to know the players and have player as its attribute.

6. **Chips** [Association: **Board**]

- This generates the chips that the players will use to play the game with. The chip can vary from 0 to many since the players start with 6 chips and it starts decreasing since the chip can be removed if the opponent player has 3 chips in a row.

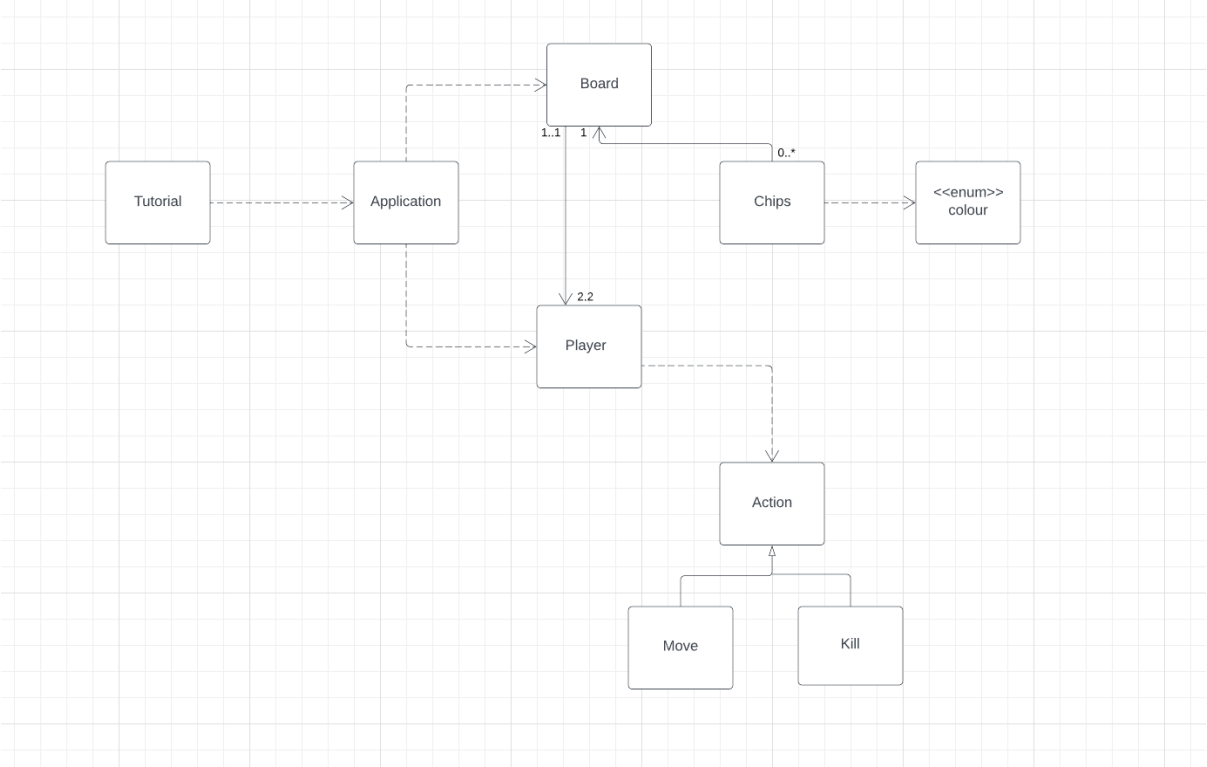
- It is associated with the Board because **Chips** can exist on its own as well as the board but it needs to know which board is on since the players are going to use the chips on that board to play. E.gThe board needs to know where the chips are being placed and the chips need to know where it is placed.

7. **Color** [Enum: Black, White]

- Since there are two types of chips, black and white. Instead of using an abstract class, using an enum would be the better option as both black and white chips will behave the same and the only difference it has is which player it is assigned to.

Assumptions :

No assumptions were made for this design.



4. Basic UI design

- Low fidelity prototype demonstrating interactions

[High resolution version is in the other file in GIT or in https://www.figma.com/file/3NqvcJguRnyufqinrGGwGa/9-Man-Morris-Prototype?node-id=0%3A1&t=SCGHV2lXGKMoVN2n-1]