

FIT3077: SOFTWARE ENGINEERING: ARCHITECTURE AND DESIGN

Sprint 3 (25%)

Design Rationale



MONASH
University

Done By

Shillong Xiao (Norman) (30257263)

Sok Ear (32442688)

Kisara Batugedara (30912539)

Faculty of Information Technology

Note for the project

The working game prototype can be seen under **FX branch**.

Design Rationale

Key Changes in Architecture

Implementation of Game Stages

One key change in the architecture of our 9 Men's Morris Game was to implement different game stages which we had not taken into consideration in our previous sprint. This key change was necessary as we needed to take into account how the game was going to keep track of each stage of play which are as follows:

- **Initial Placement Stage:** This is the stage where both players place all their 9 tokens on the board
- **Sliding Move Stage:** This is the stage where players are allowed to slide move their tokens on the board from one position to another.
- **Finished Stage:** This is the stage where a winner is found in the game and the game must record that it has reached the end.

We implemented this easily using a Game Stage Enumeration and have the Game class have a direct association with it. This means that the Game class has a Game Stage attribute that records the current stage of the game as it gets played.

The reason why we went with this approach is because it becomes the game can keep track of the different stages and all the conditions that allow it to change from one stage to another. For example, during the initial placement stage, the tokens cannot slide into another position until all the 18 tokens have been placed on the board and hence cannot transition into the sliding move stage. Likewise, the game knows it should transition into the finished stage when a winner is detected.

In this approach, the responsibilities of each stage is encapsulated hence it improves code readability and ensure that the game progresses smoothly. Additionally, it enables easier testing of individual stages and promotes modular design, allowing for easier expansion or modification of the game in the future.

Advantages	Disadvantages
<ul style="list-style-type: none"> - Improved Code Organization: Implementing game stages helps in organising the codebase by separating different stages of the game.. - Clear Flow and Progression: Game stages provide a structured flow for the game, allowing players to move through distinct phases in a logical order. - Separation of responsibilities: Dividing the game into stages helps separate different aspects of the game logic, such as initial placement, sliding moves and finished game stages. This separation allows for better encapsulation, easier testing and more modular design. 	<ul style="list-style-type: none"> - Increased Complexity: Introducing game stages adds an additional layer of complexity to the codebase. Managing the transitions between stages, maintaining the state of the game, and handling stage-specific behaviours can increase the overall complexity of the code.

Conclusion: After careful consideration, despite it being complex in terms of development, we decided it was best to go ahead and implement the Game stages as it ensures a smooth sailing experience in terms of gameplay and it helps the application be in line with the actual rules of the 9 Men's Morris Game.

Implementation of a FxController Class to Integrate Game Logic with GUI

Another key change in the architecture is to have an FxController Class that serves as the intermediate between the Game Logic and the GUI. The purpose of this class is to separate the Game Logic from the GUI but at the same time make sure they work together to reflect the necessary functionalities in the game. This ensures that each component focuses on its specific responsibilities. This separation improves the maintainability and extensibility of the codebase.

By separating the GUI and the Game Logic, we can improve or replace the GUI without affecting the underlying Game Mechanics. This ensures Flexibility in implementing different user interfaces or adapting to changes in the GUI Frameworks. This approach also promotes code reusability across different platforms or interfaces, such as a desktop application, a web application or a mobile app by simply developing separate GUI components that interact with the controller, which will be beneficial for future development.

Advantages	Disadvantages
<ul style="list-style-type: none"> - Seperation of responsibilities: Using a controller class helps separate the game logic from the GUI components. This seperation promotes a more modular and maintainable codebase, making it easier to understand, modify and extend. - Code Reusability: The controller class acts as an intermediary between the game logic and GUI. By encapsulating the game-specific functionality within the controller, you can reuse the game logic across different interfaces or platforms without the need to modify it. 	<ul style="list-style-type: none"> - Increased Complexity: Introducing a controller class adds an additional layer of abstraction and complexity to the codebase. Developers need to carefully manage the interaction between the controller, game logic, and GUI components which can increase overall complexity of the code.

Conclusion: After careful consideration, despite it being complex in terms of development, we decided it was best to go ahead and implement the FxController class as it ensures a working GUI with the correct gaming logic that represents accurate game rules to the actual 9 Men's Morris Game.

Implementation of MillMoveAction to accommodate Mills in the game

Another key change in the architecture was to have a subclass called MillMoveAction that inherits from the parent class MoveAction. This ensures that when a 'Mill' occurs, the player that achieved a Mill, gets to remove a token from the opponent. The removing functionality of the token gets validated inside this class as it overrides the validate method inside the MoveAction class.

We decided to go with this approach as it allows us to encapsulate the logic and behaviour specific to mill moves in one central location. This class can handle the validation of mill formation, determine which opponent's pieces to be removed and enforce the rules related to mill moves. Through encapsulation, the code organization gets improved and becomes easier to understand and maintain.

This approach also helps in code reusability as it creates a reusable component that can be used in multiple parts of the code base. Mill moves occur in various stages of the game, such as initial placement stage and sliding move stage. Having a dedicated class to handle mill moves allows you to reuse the logic in different scenarios, promoting code reuse and reducing code duplication thus preventing the DRY principle.

Since the mill move logic is isolated and distinct from other parts of the game, it improves code testability as it makes it easier to modify or extend the mill move behaviour without impacting other parts of the game.

In terms of flexibility, having a dedicated mill move action class allows you to modify the mill move behaviour in case additional rules or variations to the mill move logic are introduced in the future.

Quality Attributes (Non-Functional Requirements)

Usability

We believe that usability is actually very relevant when it comes to checking quality attributes. The reason why is because usability is because it indicates how effective and easy it is for the players (end-users) to learn and use the game/system.

Advantages	Disadvantages
<ul style="list-style-type: none">- Improved User Experience: Usability requirements focus on enhancing the user experience by making the game more intuitive, efficient, and user-friendly.- Increased adoption and acceptance: By incorporating usability requirements, you can make the game more accessible and easier to use for a wide range of users.- Enhanced Productivity and Efficiency: Usability requirements focus on improving the efficiency of user interactions, reducing cognitive load, and streamlining workflows. This leads to increased productivity for users, saving time and effort in completing tasks.	<ul style="list-style-type: none">- Development Effort: Implementing Robust usability requirements can require additional development effort such as iterative design, usability testing and iteration based on user feedback.- Trade-Offs with other NFRs: Prioritising usability requirements may require trade-offs with other functional or non-functional requirements such as performance, security, etc. Balancing these can be challenging and may require a compromise.

Conclusion: After careful consideration, we decided usability is a non functional requirement that we consider relevant in our game design as we want to make a game that is very effective, smooth sailing and satisfying for the end-users to use.

To test our game in terms of usability, we must first declare few of the quality expectations in terms of usability.

The following are the Usability quality expectations we have for our 9 Men's Morris Game:

R.1. Tokens must move smoothly when moved.

R.2. Players must be able to identify whose turn it is.

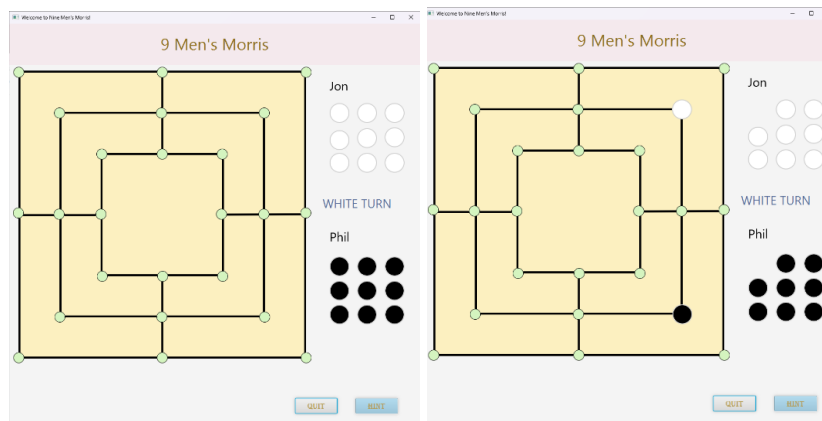
R.3. Players must be given feedback if they make a move that is not in alignment with the rules.

R.4. Players must be alerted when a mill has occurred and prompt them to hit the mill button to remove a token from the opposition.

R.5. Players must be alerted when the game has finished and who won the game.

R.1. Tokens must move smoothly when moved.

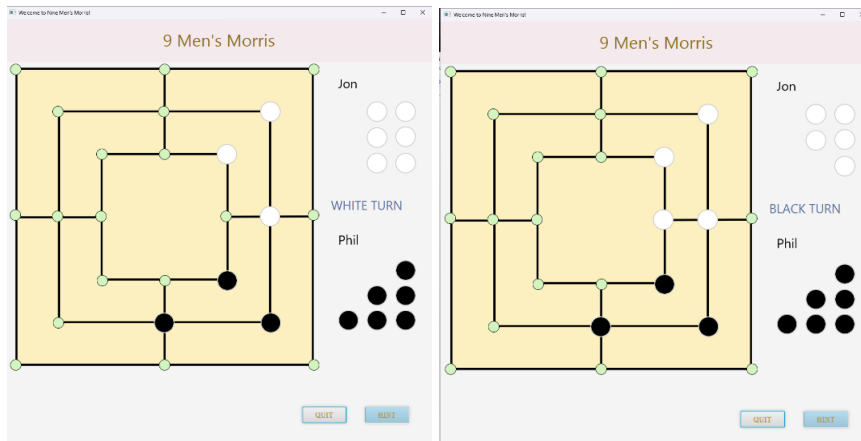
Our game does ensure that the tokens when selected move very smoothly to ensure satisfying visual in our gameplay.



As can be seen from the above screenshots, the tokens move smoothly into the positions selected by the players.

R.2. Players must be able to identify whose turn it is.

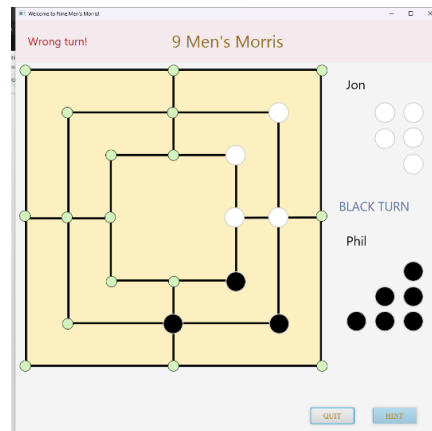
Our game makes it clear whose turn it is by indicating the colour of the token that needs to be placed for that particular turn.



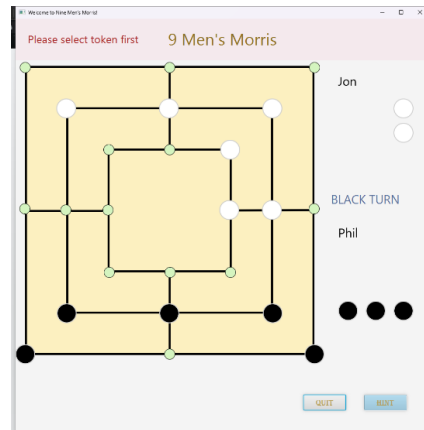
As can be seen from the right hand side of the above screenshots, it indicates the colour of the token of the player has in their turn.

R.3. Players must be given feedback if they make a move that is not in alignment with the rules.

Our game gives feedback to the players if they make an invalid move.



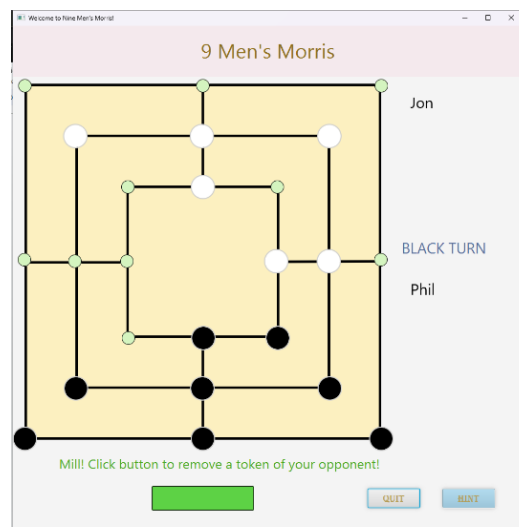
In the above screenshot, Jon makes an invalid move as it is not their turn. The feedback from the game can be seen in the top left corner of the application where it says "Wrong Turn!".



In the above screenshot, Phil did not select a token first before selecting the destination position of the token and hence the game has given feedback "Please select token first".

R.4. Players must be alerted when a mill has occurred and prompt them to hit the mill button to remove a token from the opposition

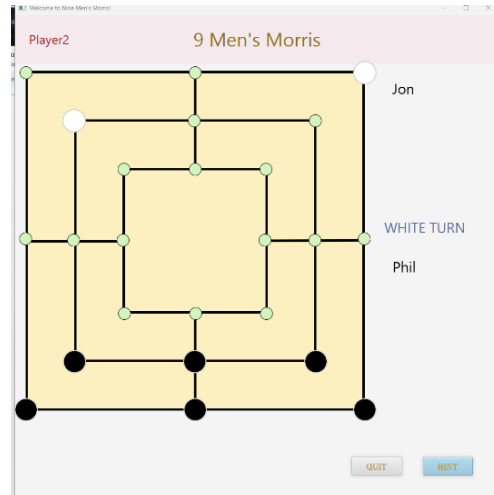
Our game gives the player who achieved a mill a prompt to select the token they want to remove from the opposition and then select the mill button.



As can be seen from the above screenshot, when Phil achieves a mill, the game gives them feedback in a prompt that says "Mill!! Click button to remove a token of your opponent".

R.5. Players must be alerted when the game has finished and who won the game.

Our game alerts the players who the winner is when one of the player's have less than 3 tokens left.



As can be seen from the above screenshot, Phil has won the game since the number of tokens left on Jon's side is less than 3. Therefore, the game displays the player number who won the game on the top left corner.

Reliability

We believe that Reliability serves as a very important parameter that is highly relevant in our game. This is because Reliability determines whether the application can run without any failure for a given amount of time under predefined conditions.

Advantages	Disadvantages
<ul style="list-style-type: none">- Increased Trust and Confidence: Reliability requirements focus on the game's ability to consistently perform its intended functions without failure or errors.- End-User Satisfaction: A reliable game enhances end-user satisfaction by minimizing downtime, errors and disruptions. Users can depend on the game to function reliably, leading to a positive user experience and increased customer loyalty.	<ul style="list-style-type: none">- Design Complexity: Ensuring reliability often involves designing complex systems with fault-tolerant mechanisms and redundancy. This complexity can increase the design and implementation challenges, making the game harder to understand, maintain and troubleshoot.- Difficulty in Quantification: Reliability is a complex concept to measure and quantify accurately. Defining and setting precise

<ul style="list-style-type: none"> - Improved Productivity: Reliability requirements ensure that the system is available and functioning properly when needed. This leads to improved productivity for users, as they can consistently access and use the game. 	reliability targets can be challenging, as it depends on multiple factors such as game complexity, usage patterns, and environmental changes.
---	---

Conclusion: After careful consideration, we decided Reliability should be one of the key Non-Functional requirements as it ensures a product/game that works at a higher percentage of the time. Unfortunately, a given system can fail or not work accordingly due to errors and this is where maintainability comes in where the system can be improved upon different time periods.

Flexibility

Adaptation 1:

The program uses generalisation of MoveAction which can lead to increased flexibility when more move types are to be implemented in the future.

Advantages	Disadvantages
<p>By incorporating this approach, it enhances the program's flexibility when there are rule changes, thereby making the program more modifiable and capable of supporting a changing</p> <p>This creates encapsulation for move functionality which is isolated from the rest of the code. If the move function were to change in the future, this can be done so easily without heavily impacting the rest of the system.</p> <p>It will be useful if there is an undo move functionality to implement in the future. This is as the detail of each move is stored in a class instance.</p>	<p>By allowing the handling of multiple moves, it can introduce higher complexity in the code, resulting in a more intricate codebase that is costly to maintain in the future.</p> <p>It can potentially deviate the program too far from its original rules, which can make the game harder to manage.</p>

After careful consideration, this adaptation is implemented due to having increased flexibility for move actions opens up the opportunity for the game to be extended. This could be done so with minimal cost thereby endorsing improvements of the game rules which could be beneficial for the users.

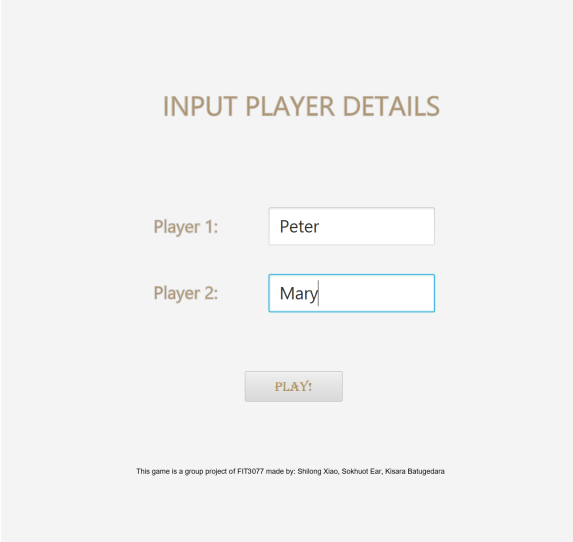
Human value:

Freedom:

We believe the key to get players engaged in the game is to provide as much freedom as possible. In our game, we try to allow players to customise their own player names which can be displayed on the UI. We also considered all the possible interactions a player can make during the game. We provide explicit feedback messages on the UI when a player hits a button or tries to make an illegal move. In this way, it would help players to learn the game and it prevents them from making the wrong move by accident.

Here, we have listed a few examples from the game:

1. Players can input their names.



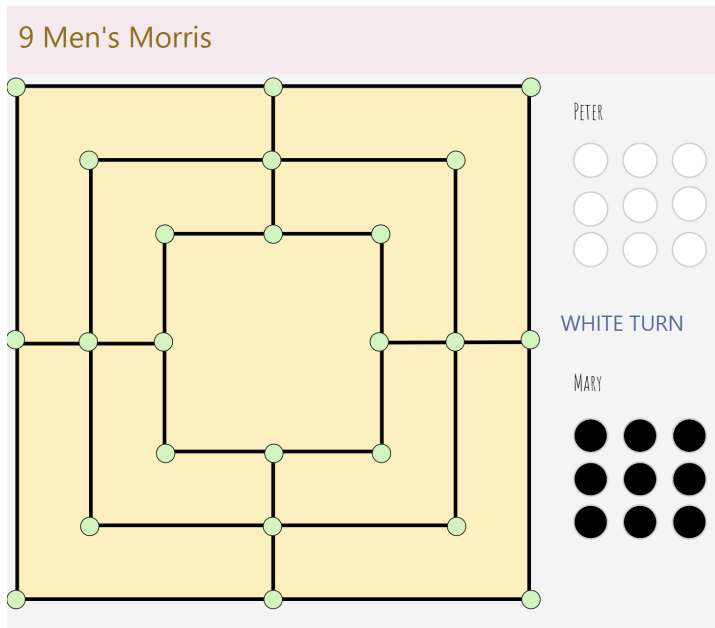
INPUT PLAYER DETAILS

Player 1:

Player 2:

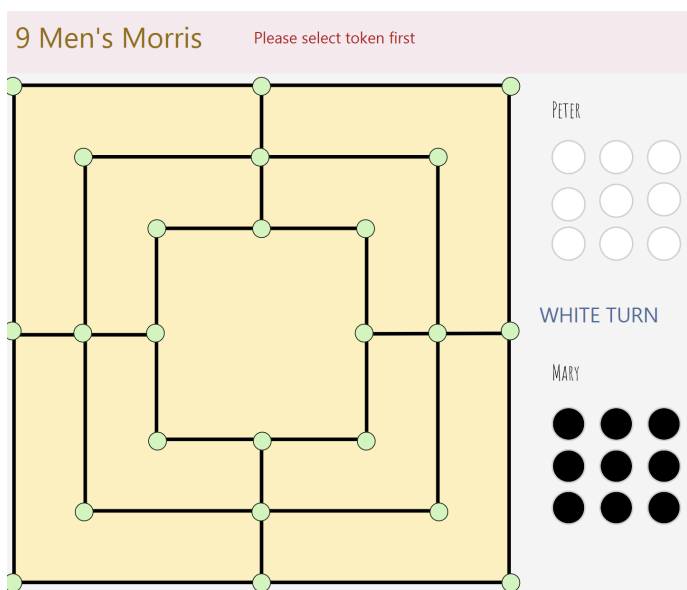
This game is a group project of FIT3077 made by: Shilong Xiao, Sokhool Eai, Kinsara Butagedara

When the player hits the play button, their name will be carried to the game mode and be displayed in the game. It helps player to understand white colour they are playing and they can easily identify the game progress by comparing with the opponent.



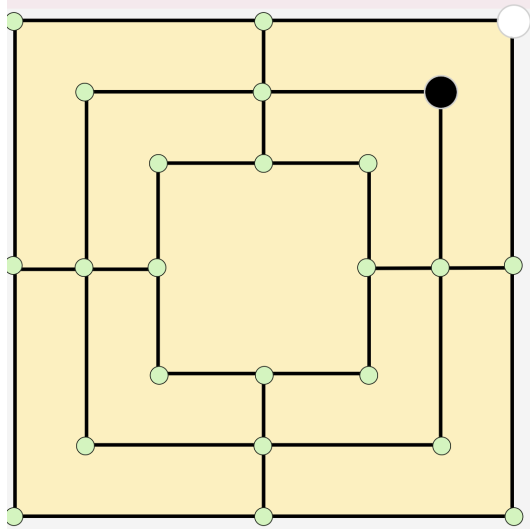
2. Our game considers multiple scenarios when players can possibly make illegal moves. Players can fully enjoy the freedom of interacting with the elements on the interface without considering the illegal moves by accident. Because the game will automatically stop them from the action happening and also provide feedback indicating where goes wrong.

It helps players to learn the game and prevents the mistouch. Below are the example of miss touching positions and tokens.



9 Men's Morris

Wrong turn!



PETER



WHITE TURN

MARY

