Nine Men's Morris
Sprint 3 Deliverables

Team Name:

Brute Force

Team Details:

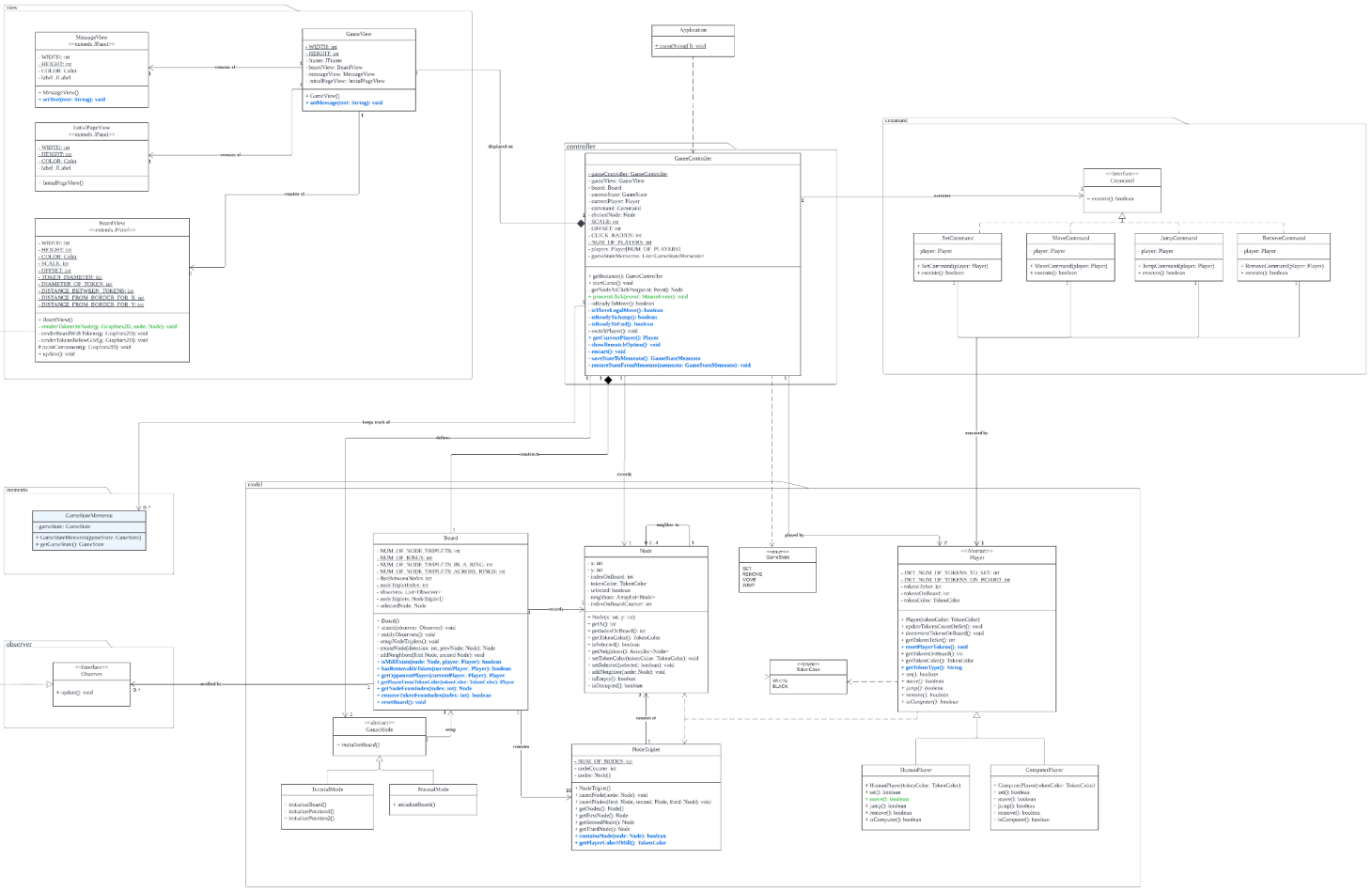
Huang GuoYueYang 32022891 ghua0010@student.monash.edu
Kuah Jia Chen 32286988 jkua0008@student.monash.edu
Tee Shun Yao 32193130 stee0005@student.monash.edu
Ong Di Sheng 31109667 dong0009@student.monash.edu

Table of Contents

Class Diagram

[Lucidchart link]

For this updated class diagram, **Modifications** are indicated by the color **GREEN**, while **Newly Added** elements are represented in **BLUE**. The use of color helps to visually distinguish between the different types of changes made to the class diagram. A detailed explanation of all the **Modifications** and **Newly Added** elements can be found in the design rationale section, where each class and method alteration is described in detail.



Sequence Diagram

[LucidChart Link for All Sequence Diagrams]

Initialization

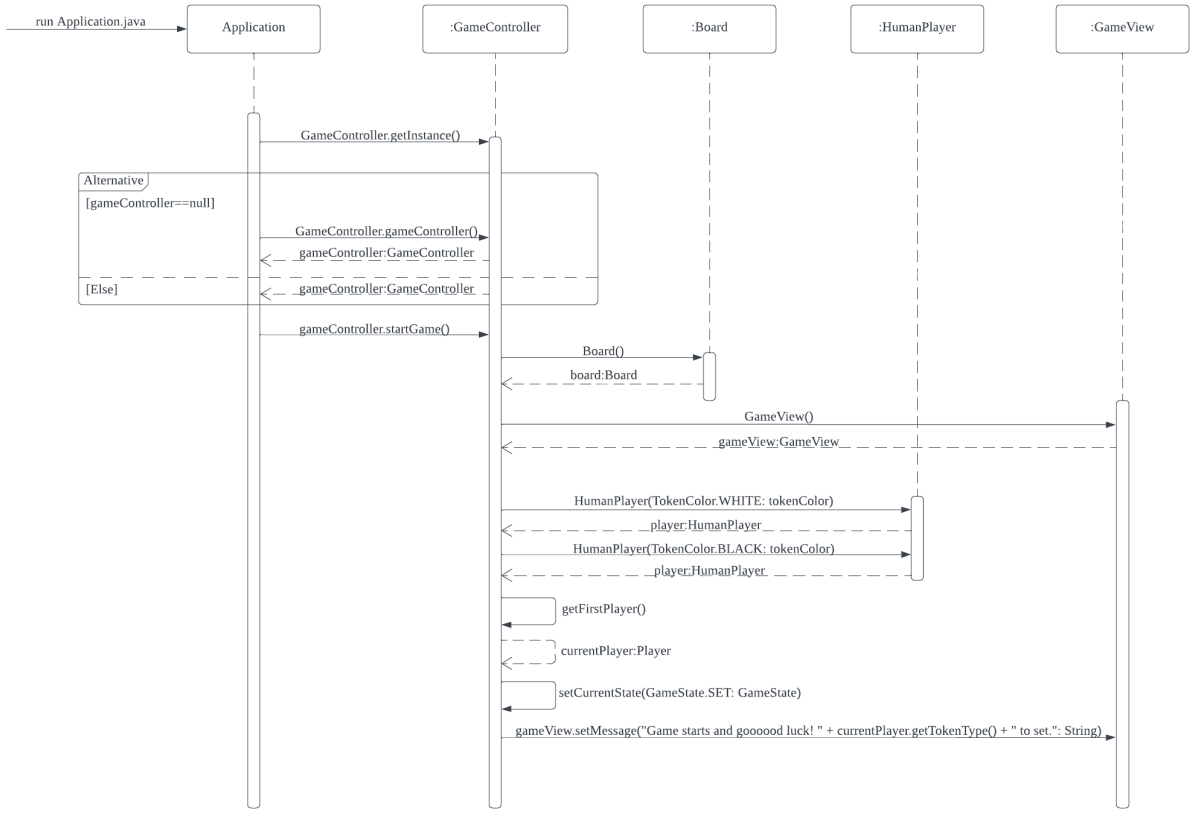


Figure 2: Sequence diagram for the bootstrap (or initialization) process of the 9MM game

Set

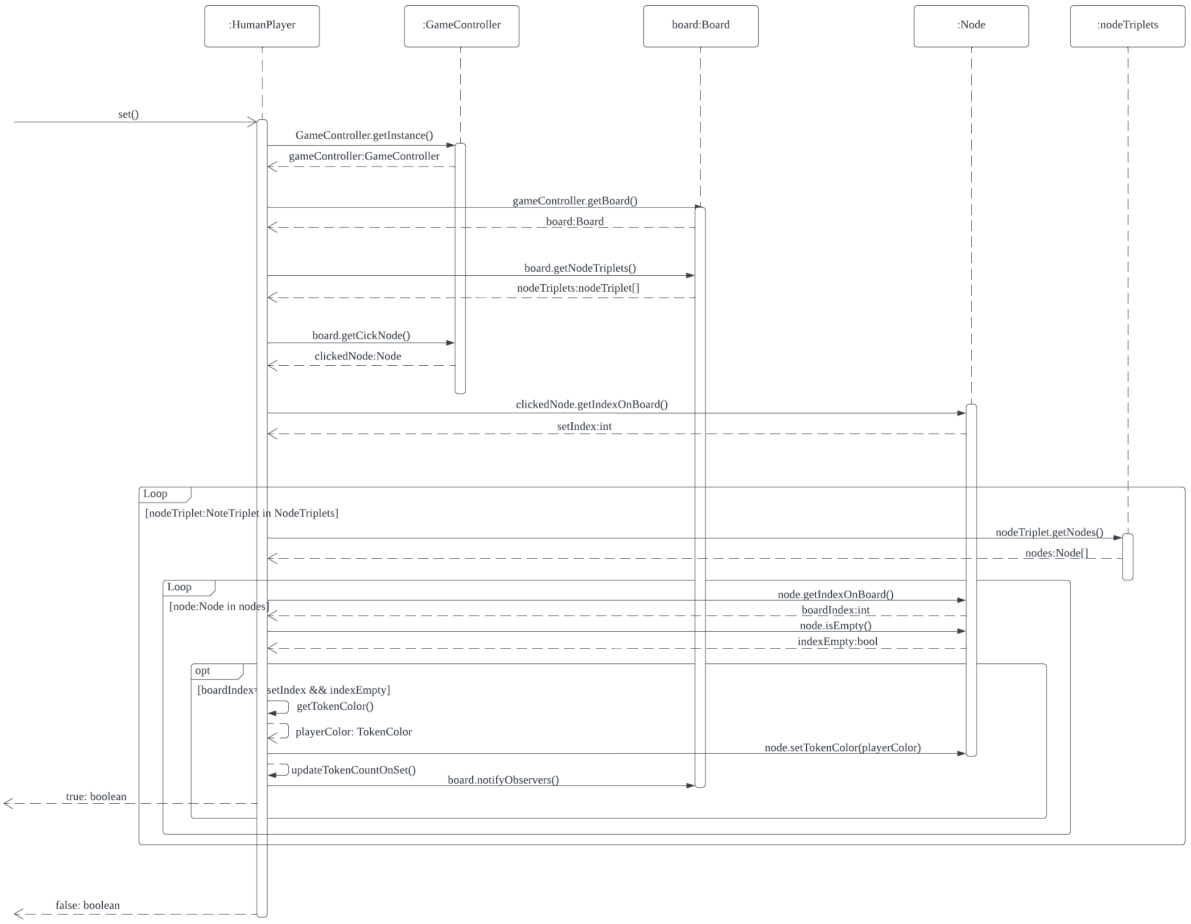


Figure 3: Sequence diagram for Set Logic

Move

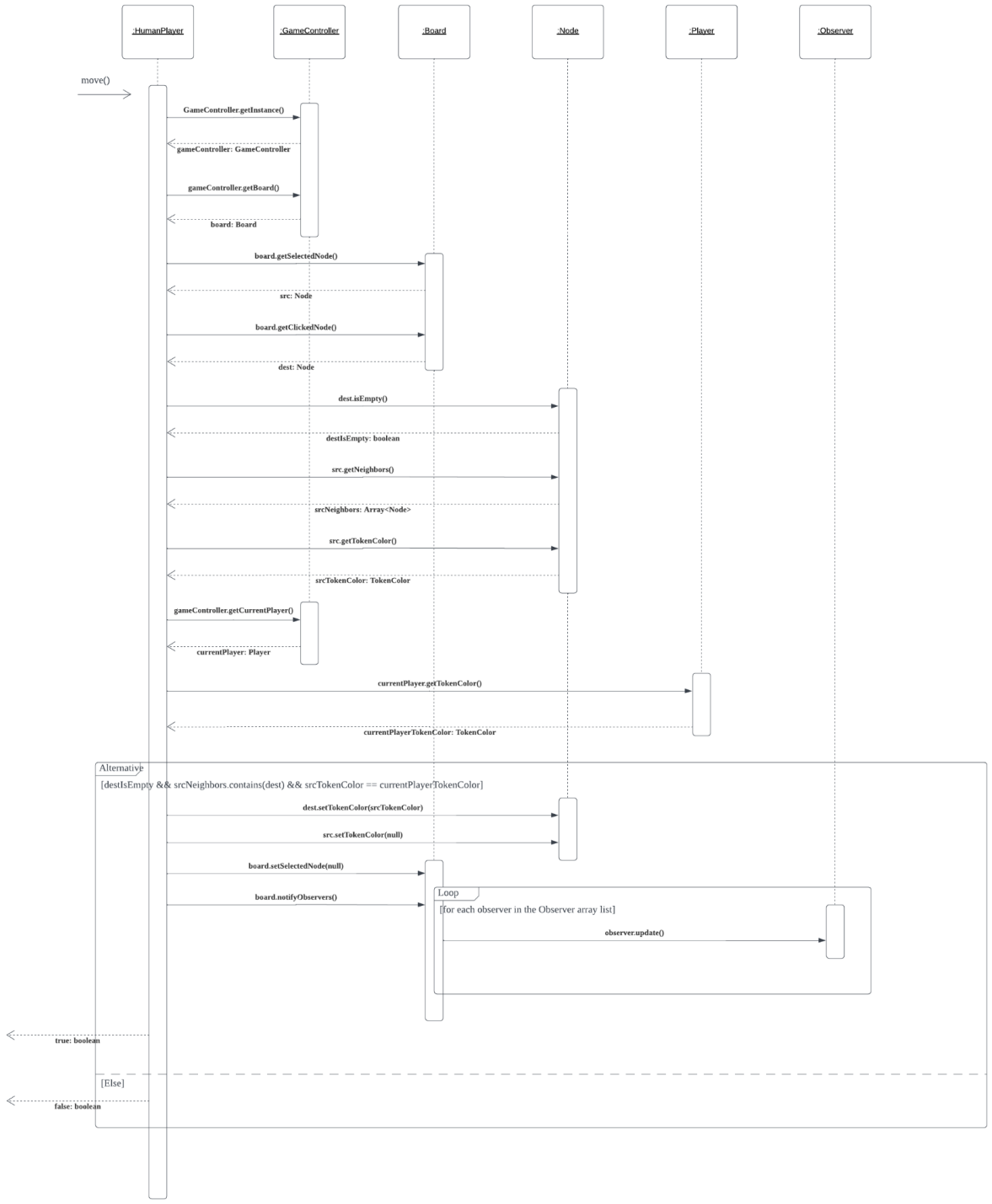


Figure 4: Sequence diagram for Move Logic

Remove

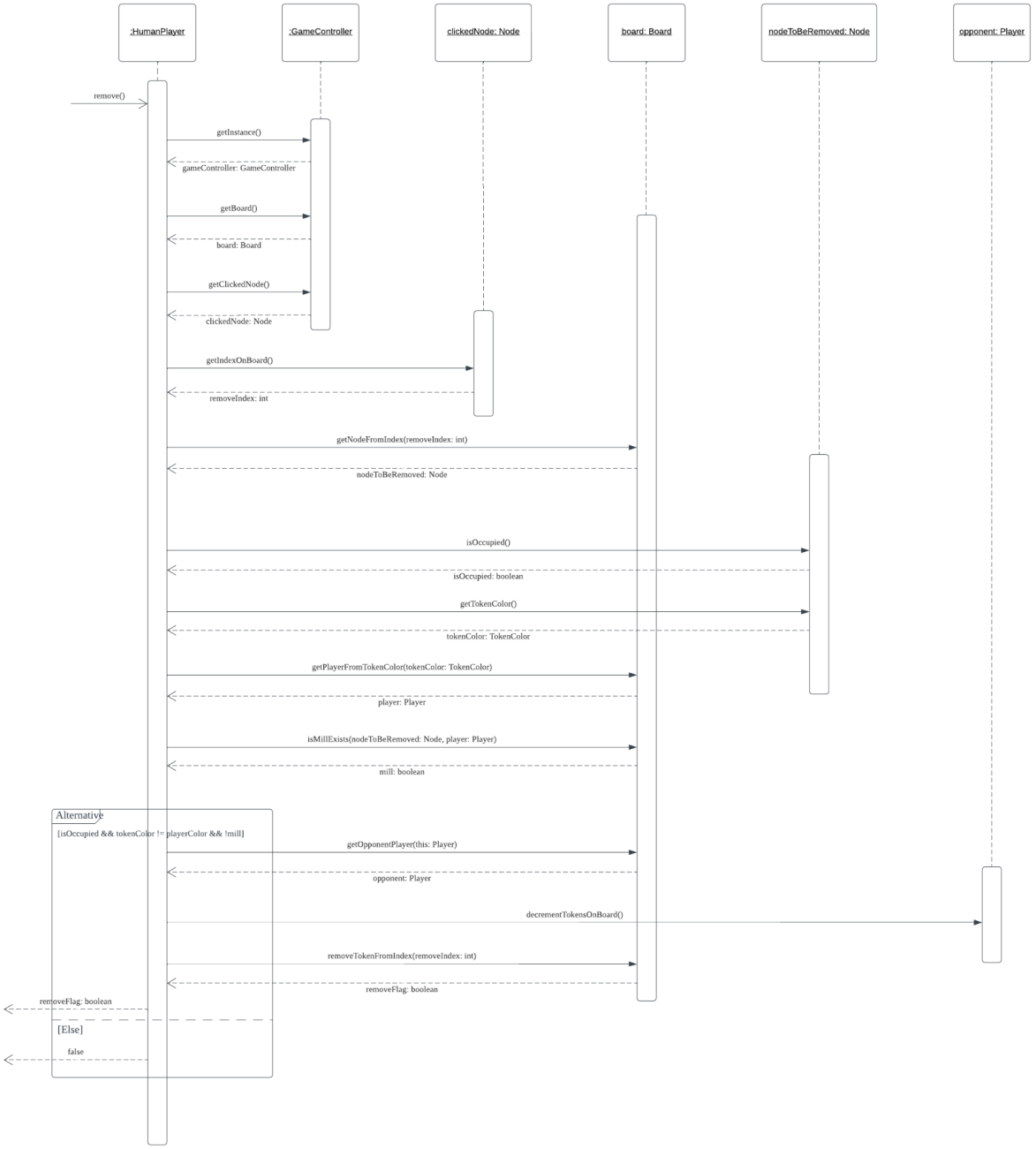


Figure 5: Sequence diagram for Remove Logic

Jump

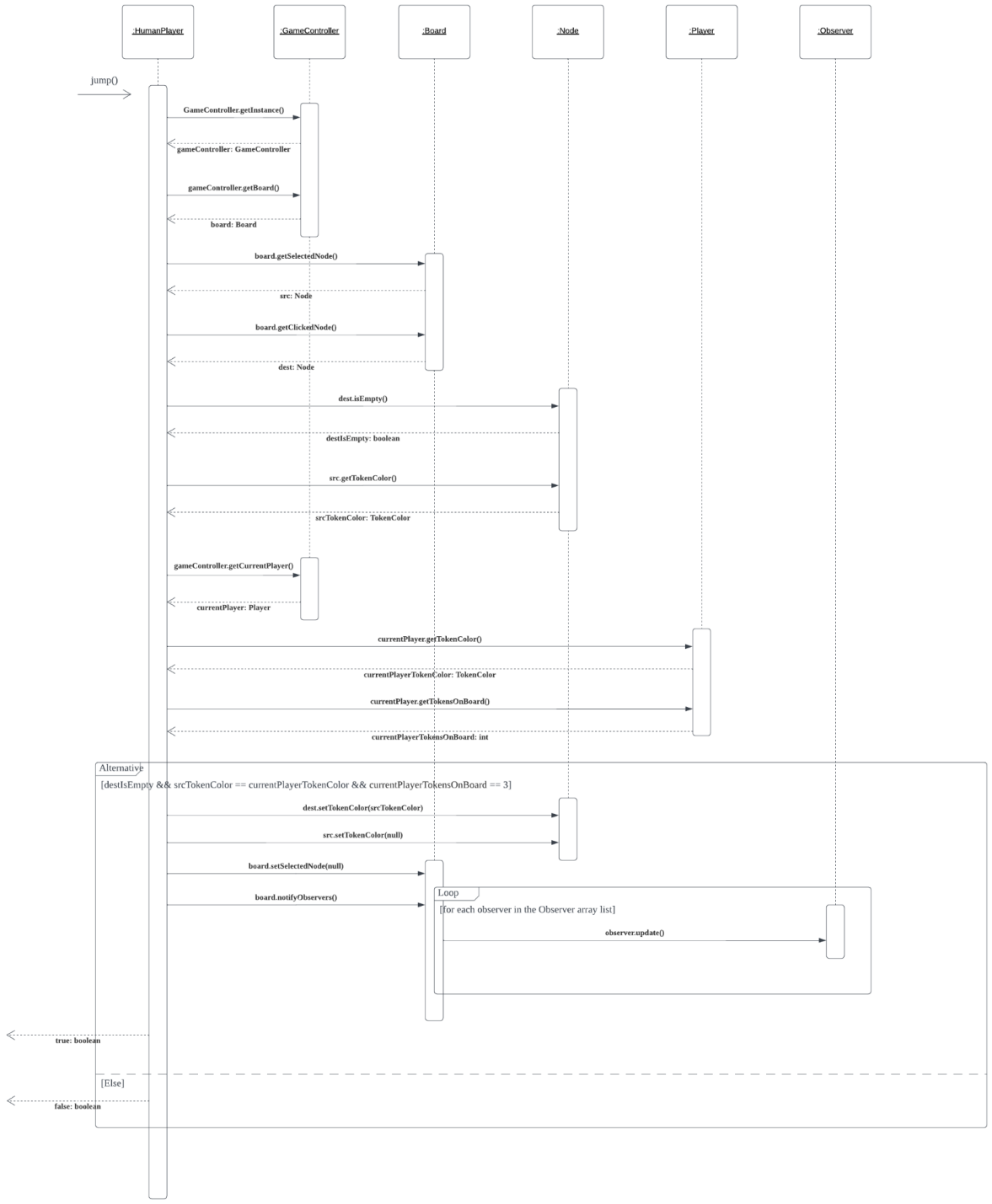


Figure 6: Sequence diagram for Jump Logic

Design Rationales

1) Why Revised Architecture and What Has Changed and Why?

Revisions from Sprint 2

Several revisions were made from Sprint 2 to enhance the overall structure and functionality of our project. These revisions encompassed a range of modifications, including additions of new classes as well as the introduction and removal of methods to existing classes.

i. Methods Additions and Modifications to Existing Classes

1. Board class
    a. isMillExists method (Newly Added Method)
    This method plays a vital role in determining whether the current player has successfully formed a mill based on the clicked node on the game board. It achieves this by systematically iterating through each node triplet on the board. During this process, the method performs two essential checks. Firstly, it examines if the node triplet contains a mill that belongs to the current player by considering the token color of the node triplet and the current player. At the same time, it also determines if the node triplet contains the node clicked by the current player. If these conditions are met, the method returns "true," indicating the formation of a mill. Otherwise, after evaluating every node triplet, the method returns "false," indicating the absence of a mill. This functionality provides a crucial mechanism for accurately identifying mill formations, enabling strategic decision-making and progression within the game.

    b. hasRemovableToken method (Newly Added Method)
    This method is responsible for determining whether the current player can remove any tokens from the opponent. It accomplishes this by iterating through each node triplet on the game board and checking if there is any node within the triplet that belongs to the opponent player and is not part of a mill formation. If such a node is found, the method returns "true," indicating the presence of a removable token. On the other hand, if no eligible node is found after evaluating all node triplets, the method returns "false." This functionality is vital in enabling the

current player to strategically remove the opponent's tokens when they are not part of a mill formation, creating opportunities for gaining an advantage in the game.

c. getOpponentPlayer method (Newly Added Method)
   This method is responsible for retrieving the opponent player based on the given input player. It achieves this by examining the token color of the input player. If the input player's token color is white, the method returns the black player. Similarly, if the input player's token color is black, it returns the white player. This allows for a straightforward and efficient way to determine the opponent player without additional complexity. By leveraging the token color information, this method ensures consistent and reliable retrieval of the opponent player in a concise and intuitive manner.

d. getPlayerFromTokenColor method (Newly Added Method)
   This method is responsible for retrieving the player based on the given token color. The functionality of the method is as follows: if the input token color is white, it will return the first player of the game, as the first player is associated with the white color. Otherwise, if the token color is black, it will return the second player, as the second player is associated with the black color. This concise implementation ensures efficient and accurate retrieval of the player object associated with a specific token color, enhancing the overall clarity and effectiveness of the game's codebase.

e. getNodeFromIndex method (Newly Added Method)
   This method retrieves a node from the board based on a given index. It accomplishes this by iterating through each node triplet on the board and comparing the indexOnBoard attribute of each node with the input index. If a node is found with a matching index, it is returned; otherwise, null is returned after checking every node triplet. This approach ensures that the method efficiently locates and returns the desired node based on the provided index.

f. removeTokenFromIndex method (Newly Added Method)
   This method serves the purpose of removing a token from the game board based on a given index. By iterating through each node triplet on the board, the method checks if the indexOnBoard attribute of any node in the triplet matches the input index. If a match is found, indicating the presence of a token at that index, the color of the corresponding node is set to null, effectively removing the token from the board. To ensure the board view reflects this change, the method

invokes the notifyObservers method, triggering an update and rendering of the latest state. Subsequently, the method returns true to indicate a successful token removal. If no match is found after checking every node triplet, the method returns false to signify that no token was removed. This method seamlessly removes a token from the specified index, ensuring the board view reflects the updated state and preserving the game's coherence.

g. resetBoard method (Newly Added Method)
   The addition of the resetBoard method serves a crucial purpose. This method is responsible for resetting the board's node triplets by reassigning the initial values of distBetweenNodes and nodeTripletIndex, and invoking the setupNodeTriplets() method. By doing so, the node triplets within the board are reconfigured, and any existing tokens placed on the nodes are removed, causing all the tokens to reappear underneath the grid.

   The resetBoard method is primarily called by the restart() method in the GameController class. It enables the board to be reset in the event that the user chooses to have a rematch after one of the players has already emerged victorious in the game. This functionality allows for a seamless transition between game rounds, ensuring a fresh and fair gameplay experience for the players.

2. NodeTriplet class
   a. containsNode method (Newly Added Method)
      This method allows us to determine if a specific input node is present within a node triplet. This is achieved by checking if the first node in the triplet contains the input node, followed by the second node, and then the third node. If any of these nodes contain the input node, the method returns true, indicating that the node triplet does indeed contain the desired node. Alternatively, if none of the nodes in the triplet match the input node, the method returns false, signifying its absence. By utilizing this method, we can effectively determine the presence or absence of a specific input node within a node triplet, providing valuable functionality for various operations and validations in the context of our Nine Men Morris game.

   b. getPlayerColorIfMill method (Newly Added Method)
      This method is used to determine if a node triplet forms a mill and, if so, returns the color of the player who owns the mill. It checks the token color of the first, second, and third node in the node triplet. If the token color of all three nodes is white, it indicates a mill belonging to

the white player, and the method returns white. Similarly, if the token color of all three nodes is black, it signifies a mill belonging to the black player, and the method returns black. If neither of these conditions is met, indicating that there is no mill, the method returns null. This allows for easy identification of mill ownership based on the color returned by the method.

3. GameController class
    a. isReadyToJump method (Newly Added Method)
    The purpose of the isReadyToJump method is to determine whether any of the players can execute a jump in the game. It achieves this by checking two conditions: the number of tokens on the board must be equal to three, and the player must not be in the process of placing tokens.

    By evaluating these conditions, the method will return true if both criteria are met, indicating that a jump can be performed. Conversely, if either of the conditions is not fulfilled, the method will return false.

    b. isReadyToEnd method (Newly Added Method)
    The isReadyToEnd method serves the purpose of determining whether the game is over. It does so by evaluating two specific conditions: the number of tokens on the board for any player must be less than three, the player must not be in the process of placing tokens and the current player have no legal move to perform.

    When all of these conditions are met, the method will return true, indicating that the game is ready to end. This signifies that one player has fewer than three tokens remaining on the board or the current player has no legal move, suggesting a victory for another player. On the other hand, if any of the conditions is not satisfied, the method will return false, signifying that the game is still ongoing.

    c. getCurrentPlayer method (Newly Added Method)
    The getCurrentPlayer method serves as a getter that retrieves an instance of the Player class, representing the current player during a particular turn in the game. This method plays a crucial role in validating move and jump commands by ensuring that the clicked node shares the same token color as the current player.

    By comparing the token color of the current player with the selected node, along with other necessary conditions, the getCurrentPlayer method helps determine the validity of the move or jump command. If

the token colors match, it signifies that the command is valid, allowing the corresponding action to be executed.

Additionally, this method facilitates the visual enhancement of the currently selected token. By examining whether the token color of the current player matches the token color of the selected node, it enables the adjustment of the token's opacity. This visual can assist players in identifying their selected token and reinforces their understanding of the current game state.

d. showRematchOption method (Newly Added Method)
The showRematchDialog method serves the purpose of presenting a dialog box to the user when the game has ended. This dialog box displays a message indicating that the game has ended and provides the option to initiate a rematch.

Upon selecting the "Yes" option, the game will be reset to its initial state, allowing the user to play again. This enables the players to engage in a new round, providing them with the opportunity for a rematch and further enjoyment of the game. Conversely, if the "No" option is selected, the game will be terminated, concluding the gameplay session.This provides a convenient and intuitive way for players to decide whether they want to initiate a rematch or end the game.

e. restart method (Newly Added Method)
The restart method plays an essential role in facilitating a game restart by resetting important class attributes and notifying the user about the restart process.

First, the method sets the value of the currentState attribute, which represents the game's current state, to its default setting. By performing this action, the game can be restarted from the beginning, wiping off all prior progress. The original player for the new game session is also reflected in the currentPlayer attribute, which tracks the player presently performing their turn. This makes sure that the player's turn order is correct when the game restarts.

The resetBoard() method from the Board class is used by the restart method to restart the game completely. Using this technique prepares the game board for future gameplay by clearing any tokens or game states that may have been there. Similarly to that, the method calls the Player class's resetPlayerTokens() method, which returns each player's

tokens to their initial state. By doing this, we can make sure that each player starts the new game with an equal number of tokens.

Additionally, the Board class's notifyObservers() method is called by the restart method. By using this technique, the board status can be updated and sent to observers, such as graphical user interface (GUI) components. By doing this, a quick refresh of the game board display to reflect the reset state is made, giving a visual depiction of the new gaming session.

The restart method ensures a seamless and bug-free restart of the game by carefully carrying out these essential stages. Users are given the confidence to start a new game session knowing that all game elements have been properly reset, resulting in a fun and uninterrupted gaming experience.

f. saveStateToMemento method (Newly Added Method)
This method implements the Memento design pattern to store the current game state. This functionality is crucial when the current player forms a mill and needs to remove a token from the opponent. Before transitioning to the "REMOVE" state, the method saves the current game state, allowing us to restore it later once the token removal is completed. By encapsulating the game state within a memento object, we ensure its integrity and provide a way to revert back to the previous state seamlessly.

g. restoreStateFromMemento method (Newly Added Method)
This method is an essential component of the game's functionality, enabling the restoration of the previous game state from a provided GameStateMemento object. When the current player completes the token removal phase, it is vital to revert back to the previous state, ensuring a seamless transition and continuity in the gameplay. By restoring the game to the precise state before the removal process, it guarantees a smooth and uninterrupted experience, allowing the player to proceed from the correct point in the game without any disruptions or discrepancies. This functionality ensures that the game operates flawlessly, maintaining the integrity and flow of the gameplay.

h. processClick method (Modified Method)
In this method, we handle the current state and command of the game. Based on the game situation, we will decide what is the next step and how the game should progress. There are 7 messages set for the game. One message is used to announce the commencement of the game. Four messages, which refer to set, move, jump and remove, will

be used during the respective gameState in the form of “{Star/Moon}’s turns to {action}”. There is one more message which tells the user when a mill is formed but the remove action is not available because all the opponent’s tokens are in the mill. Another message is used to indicate the end and the winner.

i. isThereLegalMove method (Newly Added Method)
The isThereLegalMove method serves the purpose of determining if the current player has any legal moves available. Its functionality is crucial for ensuring the game's progress and detecting potential game-ending conditions.

By evaluating the game state and considering the rules of movement, the method checks if there are any valid moves that the current player can make. If at least one legal move is found, the method will return true, indicating that the player has viable options. Conversely, if no legal moves are available, the method will return false, signaling that the player is unable to make a move and potentially leading to a game-ending condition. By incorporating it into the IsReadyToEnd method, we can accurately assess if the game should continue or if it has reached a point of completion.

4. Player class
    a. resetPlayerToken method (Newly Added Method)
    The resetPlayerTokens method plays a crucial role in the game's rematch functionality. When invoked, it reverts the player's token-related attributes, namely tokensToSet, and tokensOnBoard, back to their initial values. This ensures a clean slate for the players when they decide to rematch the game, effectively resetting their token inventory and removing any tokens placed on the board during the previous session. By incorporating resetPlayerTokens into the rematch process, we guarantee that the game starts anew, providing players with a fresh opportunity to strategize and compete.

    b. getTokenType method (Newly Added Method)
    This method returns “Star” or “Moon” depending on the color of the token WHITE or BLACK. In our UI, the tokens are represented in star and moon instead of a white and black token, as it will be better to refer to the actual star or moon when displaying the messages.

    c. move method (Modified Method)
    During Sprint 2, the game allowed players to move tokens freely without considering the neighboring relationship between nodes or the

current player's turn. However, in Sprint 3, we recognized the need for stricter game rules. As a result, we made significant modifications to the game's functionality to enforce these rules.

In Sprint 3, players are now required to adhere to specific constraints when moving tokens. Firstly, the current player must be the one who performs the move. Additionally, a token can only be moved to a neighboring node, preventing players from making invalid or unrealistic moves.

To achieve these objectives, we thoroughly revised and updated the relevant functions and logic within the game. By implementing these changes, we ensured that the game now operates according to the desired rules and provides a more engaging and strategic gameplay experience for the players.

5. MessageView class
    a. MessageView constructor (Newly Added Method)
    Modify the font type, font size, font alignment, font color, and background color. This is to make the UI aesthetically pleasing.
    b. setText method (Newly Added Method)
    This method will call the setText method of Jlabel. This determines what message is displayed in the window. For different game states such as move, set, end game, and so on, we will display messages to inform the players what is going on.

6. GameView class (Newly Added Method)
    a. setMessage method
    This method will call the setText method of the messageView in the gameView class. This provides the capability of GameController to access and modify messages displayed by MessageView via GameView.

ii. Newly Added Classes

1. GameStateMemento
    This class applies the memento design pattern to store a snapshot of the game state at a specific point in time. It serves as a container for capturing and preserving the state of the game. By creating an instance of GameStateMemento, we can save the current state of the game. This allows us to later restore the game to this saved state if needed. The GameStateMemento acts as a reliable snapshot or backup of the game,

ensuring that we can revert back to a previous state and continue the game seamlessly.

Changes in Design Pattern

i. Introduction of Memento Design Pattern

Within our Nine Men Morris game, we have integrated the Memento design pattern to address a specific game scenario. When a player forms a mill and has the opportunity to remove an opponent's token, the game state transitions to the "REMOVE" mode. However, it is vital to preserving the current game state before entering this mode, ensuring a seamless restoration once the token removal process is completed. To accomplish this, we have implemented the "saveStateToMemento" method, which stores the current game state within a GameStateMemento object. This memento serves as a snapshot of the game's state at a specific point in time. Once the token removal is finalized, the "restoreStateFromMemento" method retrieves the saved state from the GameStateMemento object, effectively returning the game to its previous state.

ii. Removing Observer Implementation from MessageView

Initially, our Nine Men Morris game included the MessageView class, which implemented the Observer interface to leverage the Observer design pattern. As an observer, the MessageView was designed to be notified by the Board class, which acted as the Observable, of any modifications such as token placements on the game board. This approach ensured that the MessageView remained synchronized with the dynamic state of the board, promptly updating and presenting relevant messages to the players.

However, upon further analysis, it became evident that the logic for effectively displaying the flow of messages resided in the GameController class, specifically within the processClick method. This method allowed for direct manipulation of the message being displayed by utilizing the setMessage method within the GameView class, which in turn would invoke the setText method of the MessageView class. Implementing the Observer interface in the MessageView class would have proven challenging, as the notifyObservers method did not include the latest message as an input parameter, making it difficult for the MessageView to directly access and display the message within its update method. This would have necessitated finding an alternative means for MessageView to retrieve the latest message from other classes within its update method.

After careful consideration of these trade-offs, the decision was made to streamline the design and improve maintainability. Instead of relying solely on the Observer pattern, we opted to directly call the setMessage method within the processClick method of the GameController class. By dividing the processClick method into distinct states, we were able to easily set the appropriate message using setMessage at the appropriate moments.

This design decision helps to simplify the implementation, eliminating potential complications and reducing the risk of errors. By leveraging the direct message setting in the GameController class, we achieve a more cohesive and efficient approach to communicating the current state of the game to the players.

2) Three Quality Attributes Considered In Design

To ensure a positive user experience, customer satisfaction, and overall functionality, it's essential to develop a high-quality game. For our Nine Man Morris Game, we have identified three critical quality attributes that are particularly relevant and important to our design: usability, portability, and reliability. These qualities are vital for creating a game that is easy to use, can run on different platforms and devices, and operates consistently and dependably. Our ultimate goal is to provide a game that not only meets but exceeds the expectations of our target audience. By prioritizing these superior qualities, we aim to increase player engagement and success, leading to a more successful game overall.

Usability

Usability plays a vital role in shaping the overall user experience and success of a game. We gave usability top priority when creating our 9MM game to ensure that players could easily navigate, understand, and become fully immersed in the experience. The main goal of our usability initiatives is to provide players of all skill levels with an intuitive and easily accessible experience.

Our user interface has been carefully developed to be both aesthetically pleasing and user-friendly, which is a crucial component of usability. Our interface smoothly directs players through the gameplay process, minimizing confusion and allowing them to concentrate on the game's essential features. It does this with the use of clear iconography and a coherent layout as shown in Figure 7.

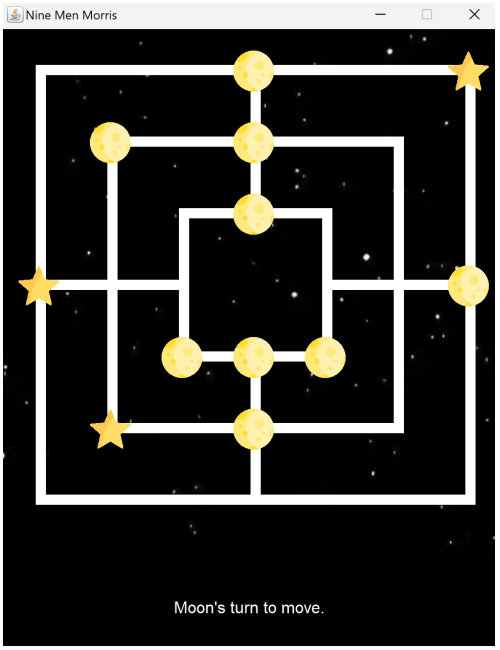


Figure 7: the user interface of our 9MM game

We have given careful consideration to the smoothness and reactivity of token movements in order to further accord with usability considerations. When players move tokens, they move them smoothly around the board, and jumping and removal operations are carried out without any visual or animation glitches. We improve the overall user experience and produce a sense of smoothness and realism by making sure these interactions are flawless and error-free. The addition of a visual indication that modifies the chosen token's opacity has improved our game's usability. Users can quickly recognize their selected token thanks to this functionality, making the gaming experience more simple and entertaining. Figure 8 shows the opacity of a token is changed when it is selected.

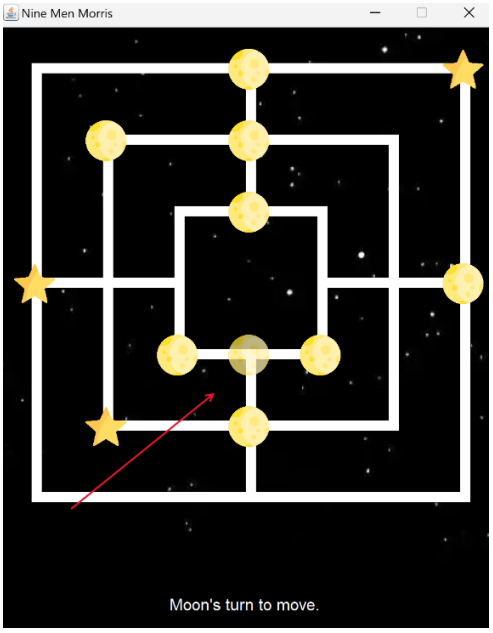


Figure 8: Change in opacity of the selected token

We have engaged in numerous playtesting sessions to collect insightful user feedback in our attempt to achieve ideal usability. We have been able to improve game mechanics, detect and fix potential usability concerns, and continuously enhance the user experience due to this vital advice. We have been able to confirm and improve the usability features of our 9MM game by actively integrating players in the testing phase, ensuring that it offers a pleasurable, approachable, and immersive gameplay experience.

In conclusion, the usability of our 9MM game is a high priority because it has a big impact on how much fun and happiness a player can have while playing. We want to develop a game that is straightforward, engrossing, and offers a great user experience for players of all backgrounds and skill levels through careful user interface design, smooth token movements, and continuous player feedback.

Portability

Portability is a critical quality attribute that refers to a software's ability to be easily transferred from one environment to another without requiring significant modifications. The capability of an application of software to run on several types of hardware, operating systems, or platforms is the emphasis on portability. We chose Java as the programming language to implement because portability was a consideration when we were creating our 9MM game. Java is the perfect language for a portable game like 9MM since it is platform-independent and compatible with a variety of hardware and operating systems.

Our 9MM game runs without any additional modifications on a variety of PC platforms, Android, and iOS due to our selection of Java and extensive testing. To ensure that it functions properly on each device, our team tested the game on a variety of them. A key component of portability was the game view's ability to remain the same size while adapting to various screen sizes and resolutions. This unrestricted accessibility makes the game more common and expands its user base by allowing it to reach a larger audience.

Overall, by concentrating on portability, our 9MM has been swiftly and effectively adapted to various hardware setups, allowing it to appeal to a wider audience. We made sure that our game is portable by keeping portability in mind during development and thorough testing, resulting in a wonderful user experience on a variety of platforms and devices.

Reliability

A crucial quality attribute for any software system, including games, is reliability. It relates to the system's capacity to operate solidly and consistently over time in a variety of situations. For our 9MM game, reliability was an essential aspect of the development process. We played the Nine Man Morris game multiple times before beginning to code the game in Java to make sure we were familiar with all potential game-breaking circumstances. This assisted us in learning the game's rules and implementing the game in accordance with them exactly, ensuring that the game would function consistently and dependably throughout time under a variety of circumstances. We also made sure that the game ran without issues, crashes, or other problems that can have a detrimental effect on the player experience.

We thoroughly tested the game to find and fix any potential problems that might have hampered its performance in order to guarantee reliability. This testing involved using numerous devices and network configurations to play the game. By doing this, we made sure the game would function reliably and consistently across a variety of hardware and network setups. Overall, reliability was a primary priority for us while we created the 9MM game. We developed a game that operates consistently and stably, giving players a seamless and uninterrupted gaming experience across diverse devices and network situations, by thoroughly testing the game and executing it in accordance with the specific game regulations.

3) Human Value Relevant To Game

Creativity

In the design of the Nine Men Morris game, the human value of creativity has been explicitly considered. Creativity is relevant and important to the game as it elevates the overall gameplay experience, empowers players' self-expression, and fuels strategic innovation. Figure 9 exemplifies the visually captivating and creatively designed board that serves as a gateway to a world of boundless imagination. The use of star and moon tokens adds a touch of brilliance and symbolic depth to the game. These tokens not only enhance the aesthetic appeal but also serve as a canvas for players to express their creativity by associating their moves with celestial elements. Furthermore, the space-themed backdrop featuring cosmic landscapes stimulates players' imaginations and invites them to envision themselves as interstellar adventurers. By immersing players in the vastness of space, the game fosters an environment that encourages imaginative thinking and inspires players to explore new strategies and possibilities. Through these design choices, it becomes apparent that creativity is a fundamental human value deliberately considered in the

development of the Nine Men Morris game. By embracing creativity in the game's design, the Nine Men Morris game offers players a unique and captivating experience, tapping into their imaginative capabilities and fostering a deeper connection with the game.

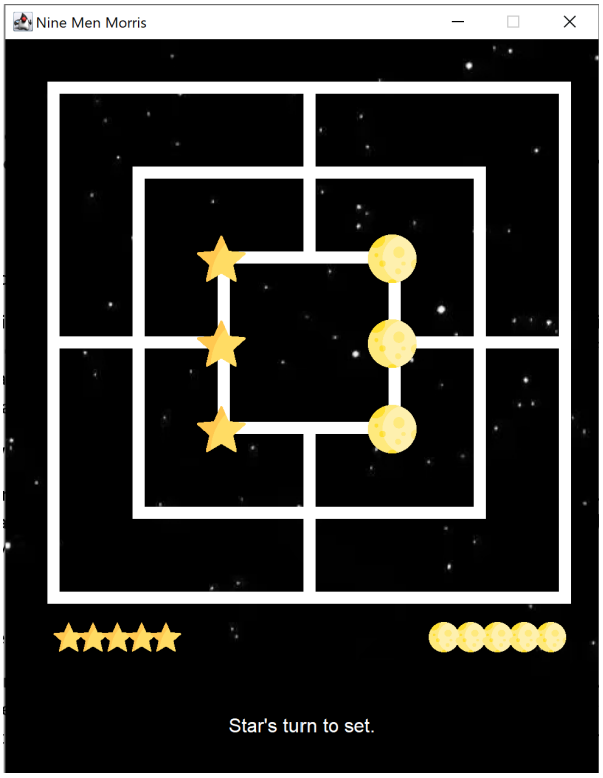


Figure 9: the visually captivating and creatively designed board

Clean

The clean value applies to our software in terms of code and user interface. The code is written with care to ensure its readability, ease of maintenance, functionality, and extendability. All the classes and interactions among them are designed with software design principles in mind. Proper comments are provided for the code consistently. Separation of concerns is applied to make debugging easier. From the user interface perspective, the necessary elements are presented in an aesthetically pleasing way. Board and tokens displayed clearly show the current situation. Clear, short, and precise messages are given to indicate the game state and what steps should be taken by the player to proceed. Our team believes that it is a clean piece of software.

Video Demonstration

[Youtube Link for our Demonstration]

Screenshots of The 5 Situation

Situation 1: Empty board at the beginning of the game

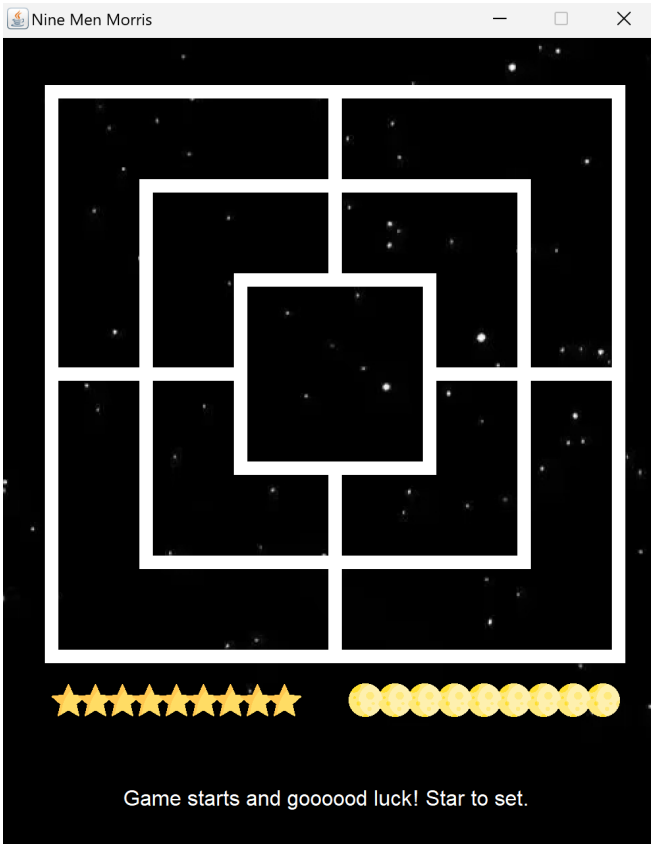


Figure 10: Initial board at the beginning of the game

Situation 2: Set, Move and Jump

1) Set

Before the players can start moving the tokens, both Player 1 and Player 2 are required to place 9 tokens each on the board. Initially, all the tokens are positioned underneath the grid. The message panel prominently displays the player's turn, indicating who should place a token. Upon clicking a node on the board, the corresponding tokens are revealed and displayed on that specific node, allowing for the game to progress smoothly.

a. Initial Board Configuration without any Tokens Placed

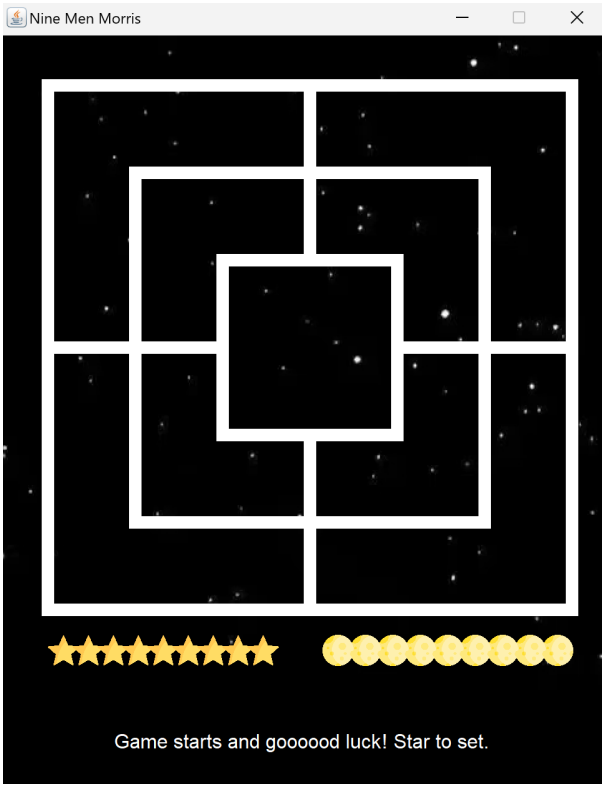


Figure 11: Initial Board Configuration without any Tokens Placed

b. Both Players in the Token Placement Phase

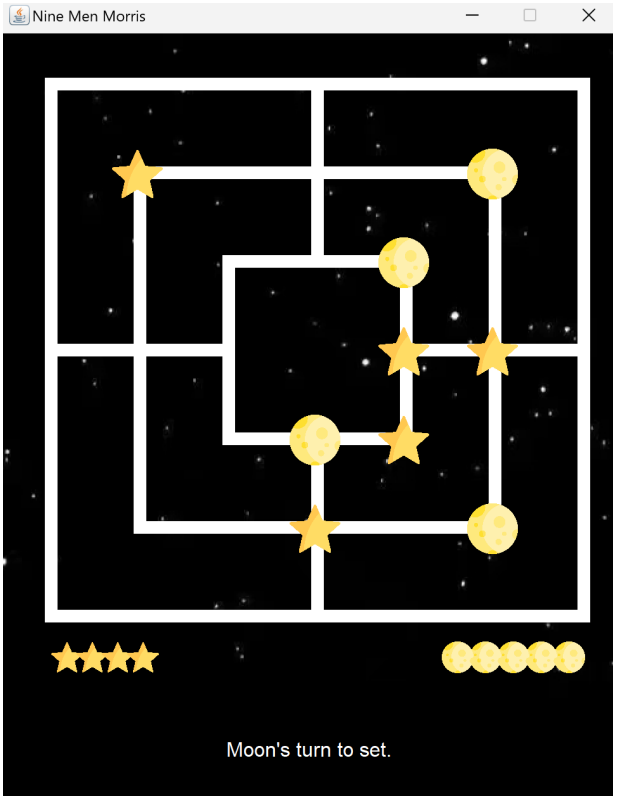


Figure 12: Both Players in the Token Placement Phase

c. Tokens Successfully Placed and Ready for Player Movement

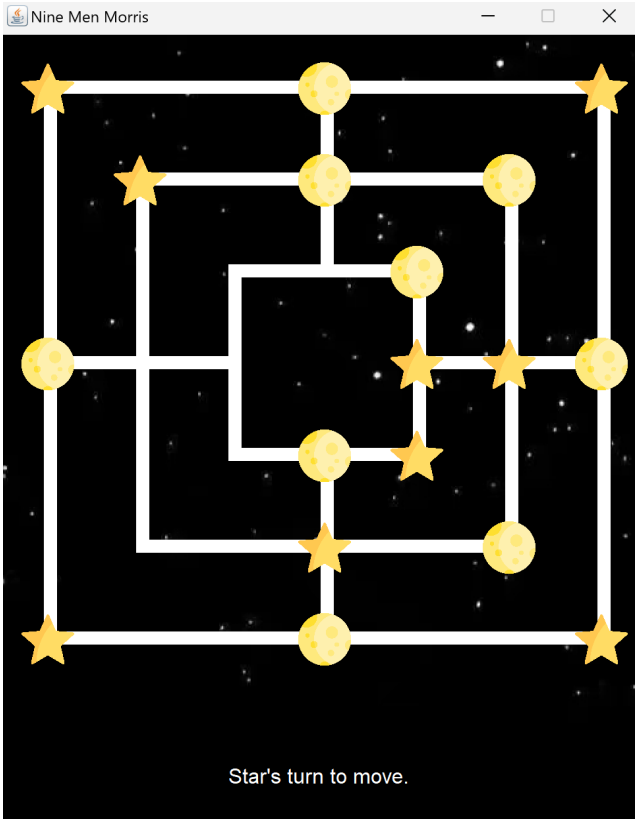


Figure 13: Tokens Successfully Placed and Ready for Player Movement

2) Move

After successfully placing all the tokens on the board, the gameplay progresses to the 'move' action, where both players take turns. The player who possesses the star token initiates the first move, followed by the player with the moon token. This alternating pattern between the star and moon tokens continues throughout the game, ensuring fair and engaging gameplay for both players.

a. Pre-Move Stage for Star Token

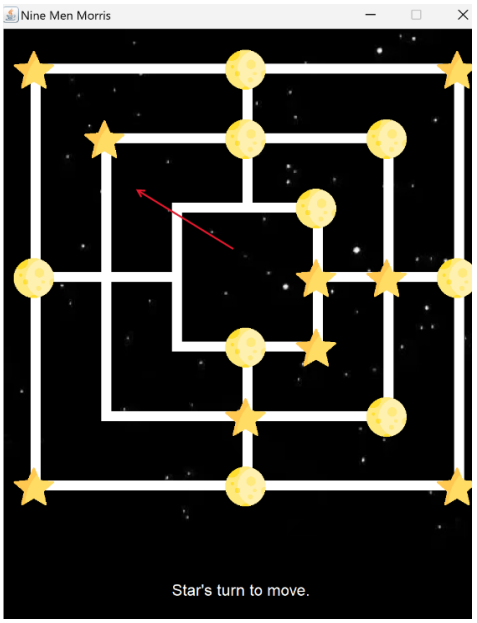


Figure 14: Pre-Move Stage for Star Token

Post-Move Stage for Star Token

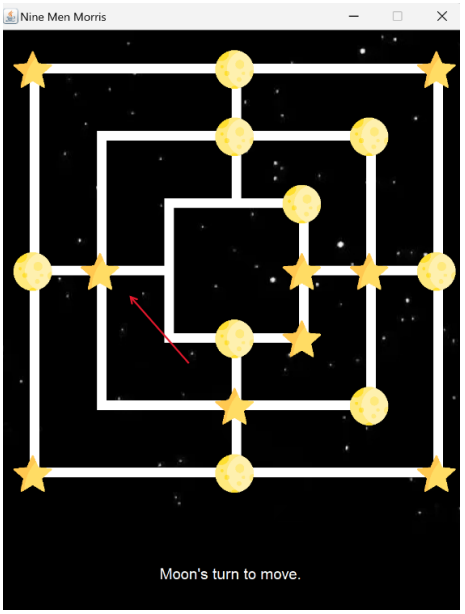


Figure 15: Post-Move Stage for Star Token

b. Pre-Move Stage for Moon Token

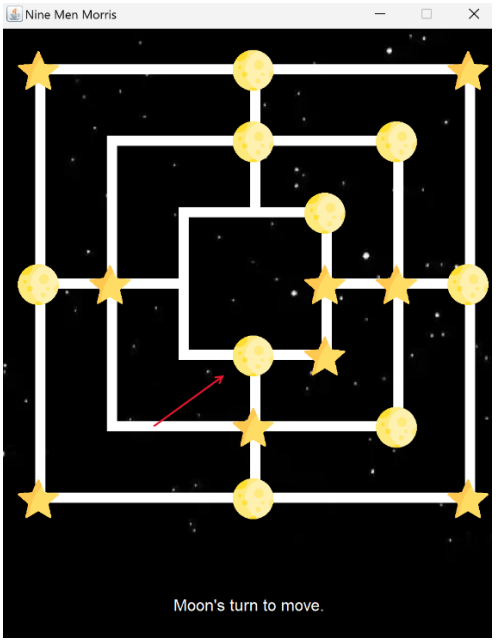


Figure 16: Pre-Move Stage for Moon Token

Post-Move Stage for Moon Token

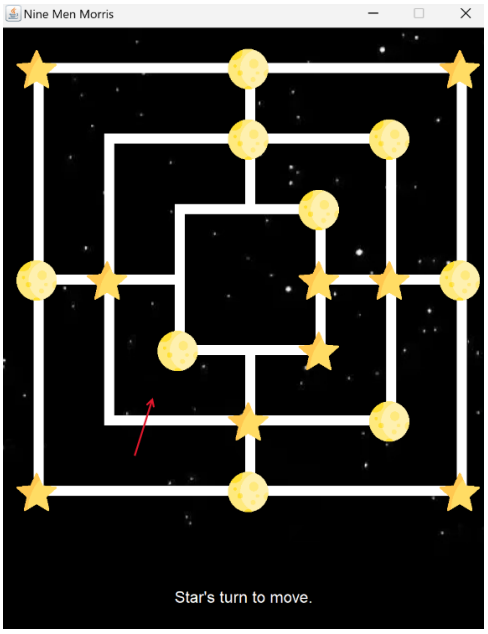


Figure 17: Post-Move Stage for Moon Token

Please note that the move action will only be executed if the player intends to move to a neighboring position of their token. Since unsuccessful move actions do no because of any changes to the board, no screenshots are provided for those situations.

3) Jump

The Jump action is only allowed when a player has exactly 3 tokens remaining on the board. In this situation, the player has the freedom to move any of their three tokens to any position on the board. In the provided screenshot, we can observe that the player using the star tokens has only 3 tokens remaining, enabling them to perform the jump action and strategically move their tokens to different positions on the board.

a. Pre-Jump Stage for Star Token

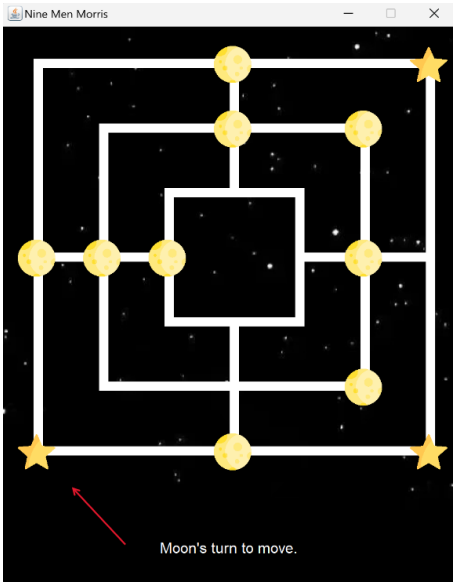


Figure 18: Pre-Jump Stage for Star Token

b. Post-Move Stage for Star Token

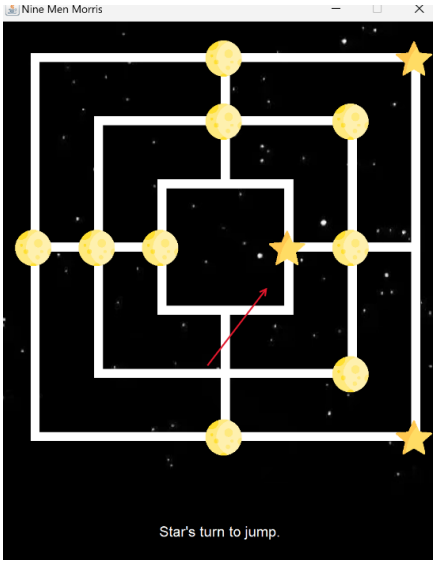


Figure 19: Post-Move Stage for Star Token

Situation 3: Detection of Mill

A mill is formed when a player successfully aligns three of their tokens in a straight non-diagonal row along one of the board's lines. Once a mill is formed, our game can detect this mill automatically and notify the players in the message panel below by displaying a message informing the player who forms the mill to remove a token from the opponent.

a. Before forming a mill by Player Star

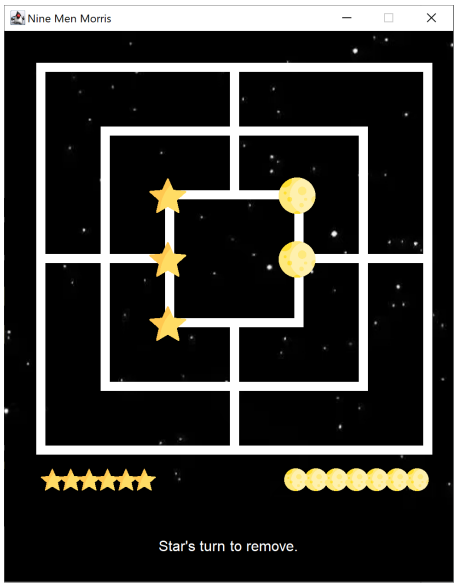


Figure 20: Before forming a mill by Player Star

b. After forming a mill by Player Star

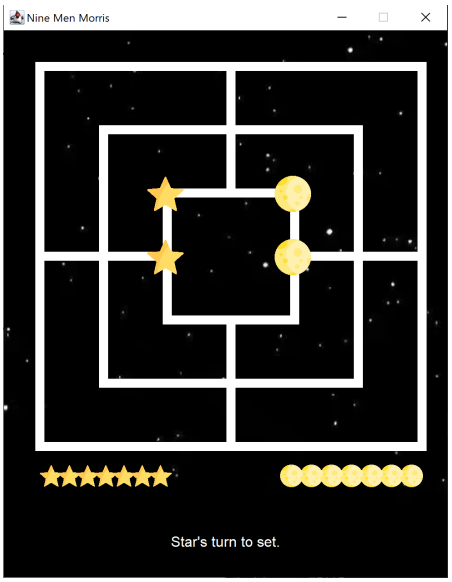


Figure 21: After forming a mill by Player Star

Situation 4: Correct Remove Action When Forming a Mill

When a player forms a mill, the player can remove one of the opponent's tokens that is not part of a mill. With that being said, those that are part of a mill are not allowed to be removed.

a. Before token removal by Player Moon

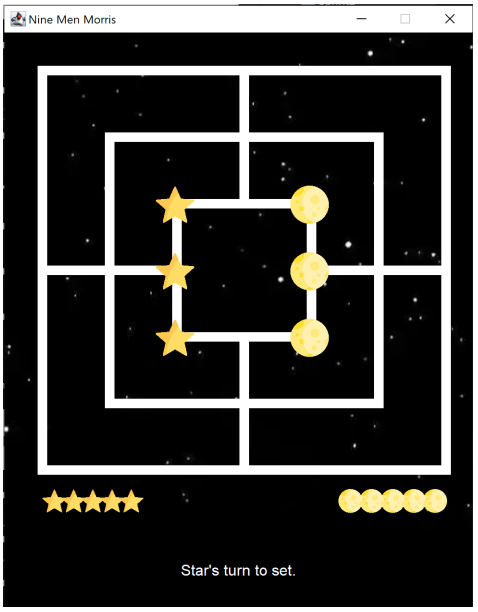


Figure 22: Before token removal by Player Moon

b. After token removal by Player Moon

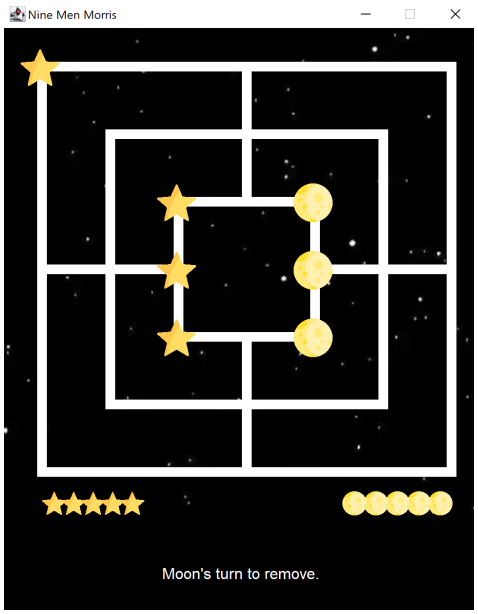


Figure 23: After token removal by Player Moon

Edge Case (all of the opponent's tokens are in the mill)

An interesting edge case whereby a player forms a mill, but all of the opponent's tokens are also in the mill. The game handles this by switching the turn to the opponent to continue the game. This ensures that the game can proceed smoothly even when all opponent's tokens are in mills.

a. Before forming a mill by Player Star

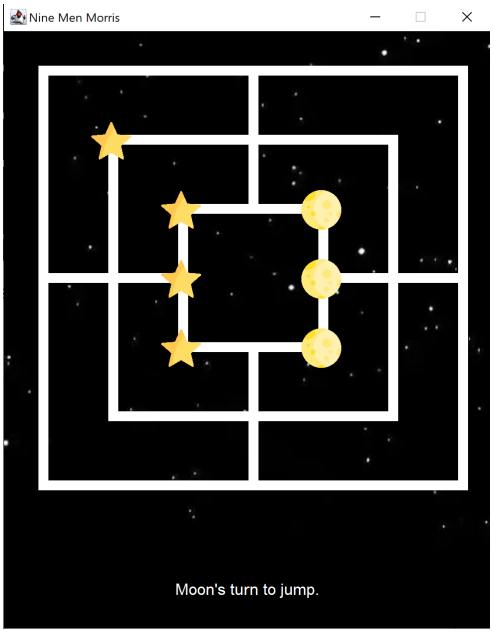


Figure 24: Before forming a mill by Player Star

b. After forming a mill by Player Star

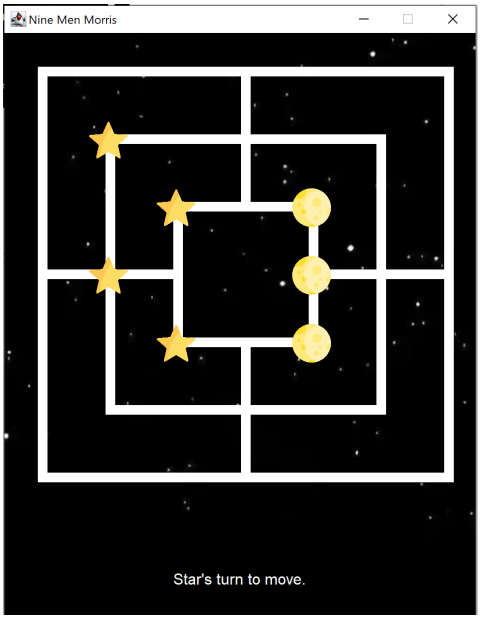


Figure 25: After forming a mill by Player Star

Situation 5: Detection of the end of a game

The game will only end in two conditions. The first condition occurs when one of the players is left with only 2 tokens. The second condition occurs when either player has any legal moves left. This happens when it's a player's turn to move, but none of their tokens can be moved.

1) First Condition: One player has less than two tokens(Star Lose, Moon Win)

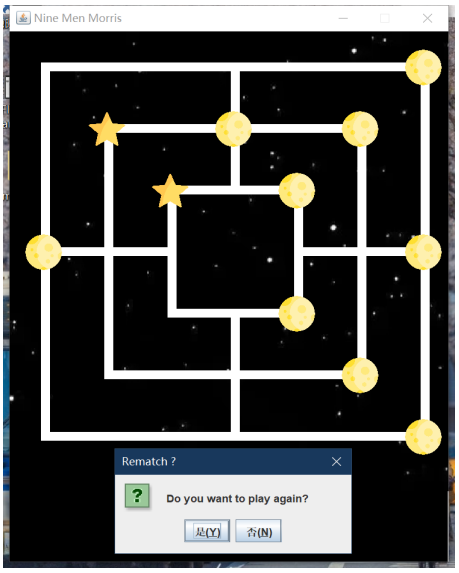


Figure 26: End game due to one player has less than two tokens

2) Second Condition: No more legal move(Star Lose, Moon Win)

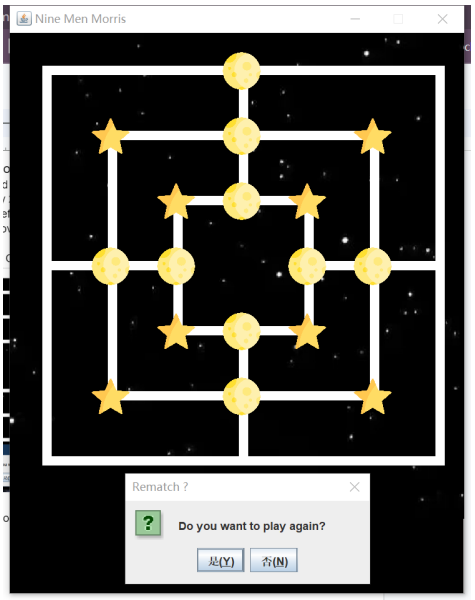


Figure 27: End game due to one player has no legal move