# Nine Men's Morris
# Sprint 4 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


**Table of Contents** **2**
**Revised User Stories - Advanced Requirement** **3**
User Stories - Blue is the newly added user stories 3
I. Computer VS Human 3
II. Hints Button 3
III. Tutorial Mode 4
What We Changed and Why We Changed For User Stories 4
I. Computer VS Human 4
II. Hints Button 5
III. Tutorial Mode 5
**Guide To Build and Run Executable** **6**
**Design Rationales** **8**
Why Designed The Architecture This Way 8
Design Pattern Used Since Previous Sprints 8
Observer 8
Singleton 9
Command 9
Memento 9
Revised Class Diagram 10
What and Why Revised The Class Diagram 11
i. Newly Added Classes 11
ii. Methods Additions and Modifications to Existing Classes 13
When Advanced Feature Was Finalized 21
Advanced Feature 1: Human VS Computer 21
Advanced Feature 2: Tutorial Mode and Hints 22
**Video Demonstration** **23**

# Revised User Stories - Advanced Requirement

## User Stories - Blue is the newly added user stories

### *I. Computer VS Human*

1. As a player, I would like to have the option to engage in a game against the computer as a player, so that it enables me to enhance my abilities and practice consistently, regardless of the availability of other opponents.
2. As a player, I would like if the computer player waits for a brief moment after I make my move before placing its token. This way, I can have sufficient time to react and observe which piece the computer player placed/moved and its new position.
3. As a player, I want the computer's moves to be carried out with optimal efficiency, so that it ensures the gameplay retains its smoothness and enjoyment, devoid of lengthy delays between moves.
4. As a player, I want the computer to select moves randomly from the available valid moves, so that it ensures the game retains its element of unpredictability and remains challenging.
5. As a game engine, I want to offer the capability for individuals to engage in gameplay against the computer as a single player, so that it allows players to enjoy the game in the absence of a human opponent.
6. As a game engine, I would like to incorporate a random move selection algorithm into the game engine for the computer player, so that the computer can choose moves randomly from the available valid moves at any given time.
7. As a game board, I want to establish the essential framework that allows individual players to engage in a computer-controlled match, so that players can enjoy the game in a solo mode.
8. As a game board, I want to establish and oversee the acceptable actions for the computer player in my role as the game board, so that the computer can make informed decisions during its turn.

### *II. Hints Button*

1. As a player, I would like to have a button for me to press to show the hints of “all next legal moves”, so that I can proceed with the game nicely.
2. As a game board, I want to ensure that the legal moves shown during the “hints” option are accurately displayed so that the players can make informed decisions about their next moves.

3. As a player, I want to be able to toggle the “hints” feature on or off during a match, so that I have the flexibility to choose whether or not to see the legal moves available to me.
4. As a player, I want the legal moves indicated by the hints to be visually distinct from other elements on the game board, such as highlighting, so that I can easily identify the available moves.
5. As a player playing against the computer, I want the hint button to be disabled during the computer player’s move, so that I won’t accidentally interfere with the system.

### *III. Tutorial Mode*

1. As a player, I want to have a step-by-step guide available in the tutorial mode accessible from the game's main menu, so that I can understand and learn how to play the game easily and effectively.
2. As a game display, I want the tutorial mode to provide clear explanations of the game mechanics so that players can gain a better understanding of how to play the game.
3. As a game display, I want to display every possible rule and situation with informative messages to introduce the game thoroughly.
4. As a game engine, I want to be able to provide an interactive tutorial mode so that players can make moves according to the instructions.
5. As a player, I want to be able to start a real game right after the tutorial ends. This allows me to quickly apply what I have learned from the tutorial mode.

## What We Changed and Why We Changed  For User Stories

### *I. Computer VS Human*

During Sprint 4, our focus remained on the user stories related to computer VS human gameplay, which were originally established in Sprint 1. However, we also took this opportunity to introduce a series of new user stories that primarily emphasized enhancing the overall player experience. By incorporating these additional user stories, we aimed to address specific issues from the player's perspective and further improve the game's functionality and enjoyment. This allowed us not only to maintain the existing features but also to expand and enrich the game with fresh perspectives and improvements tailored to the player's needs and preferences.

## *II. Hints Button*

In the initial stages of Sprint 1, our focus was primarily on providing a button for players to show the hints of all next legal moves. At that time, we did not anticipate the need for additional functionalities. However, as we progressed, we recognized the importance of incorporating certain changes to enhance the overall user experience. We realized that allowing players to toggle the "hints" feature on and off during a match would provide them with the flexibility to choose whether or not to view the available legal moves. Additionally, we understood the significance of enhancing the visual distinction of the indicated legal moves from other elements on the game board in order to facilitate easy identification by the players. With the addition of eye-catching highlighting, we ensure that the legal moves stood out clearly, thereby enabling players to make informed decisions with greater ease and efficiency. Lastly, considering the player's experience when playing against the computer, we added the user story to disable the hint button during the computer player's move. This change was introduced to avoid unintended disruptions or interference with the system caused by the human player accidentally clicking the hint button while the computer was making its move. All in all, these iterative changes are driven by our commitment to improving user experience and delivering a more enjoyable gameplay environment.

## *III. Tutorial Mode*

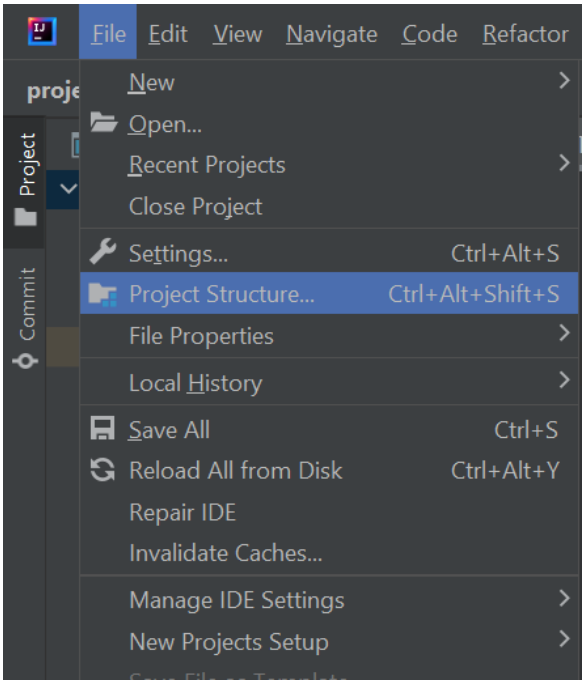
In the later stage of the implementation, there are two possible ways of designing the user stories. The first one being an interactive-based tutorial which enables the players to make moves accordingly and the other one is like a slideshow-based demonstration where the player will click the”Next” button to watch the board and message only. Our team decided that the first option will be better for the player to learn. Thus, our implementation will go through every single situation, including initialize and set, move, remove, jump, winning condition 1 and winning condition 2. For each situation, we display necessary instructions and explanations. For the two winning conditions, we only show the winning positions and do not require actions from the player. For the other situations, we prompt the user to make a move, then proceed to the next situation using the “Next” button. After the tutorial ends, players can choose to start a real game with a computer or human player, this enables players to quickly apply the stuff taught in tutorial and enjoy the game.

# Guide To Build and Run Executable

**Building an Executable Jar file**

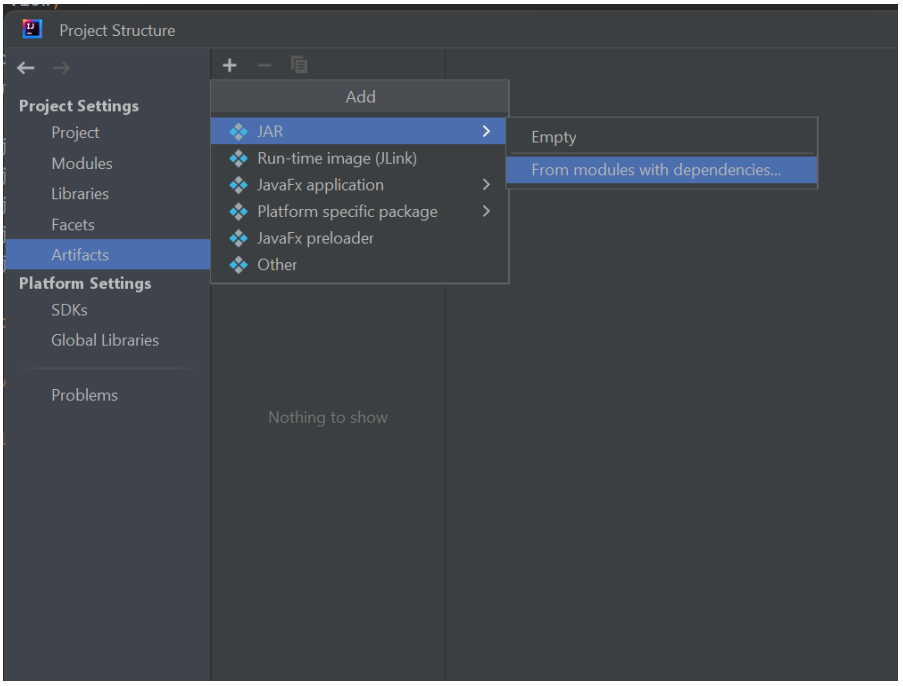
To build a file, our team utilizes the capabilities of IntelliJ IDE.

First, locate the `File -> Project Structure`.

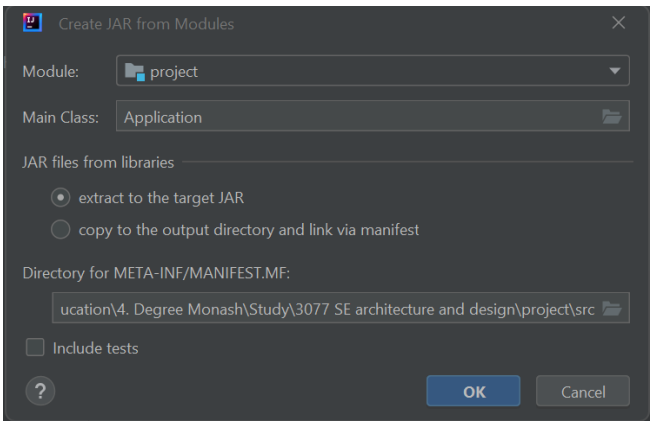



Next, click on the `Artifacts`, followed by the `"+"` icon.

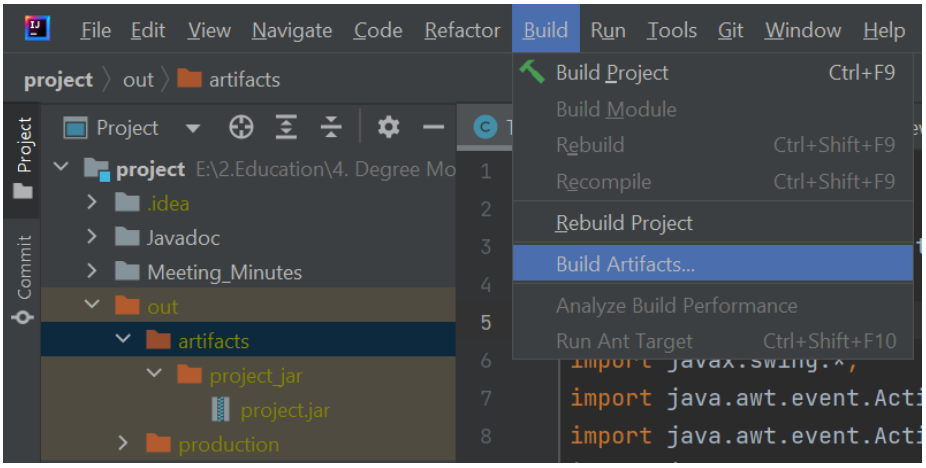
Then click on the `Jar` and `From modules with dependencies...`




Next, create the Jar from modules, make sure that the `Main class` is located at the Application.java file which contains the main method. Click OK.

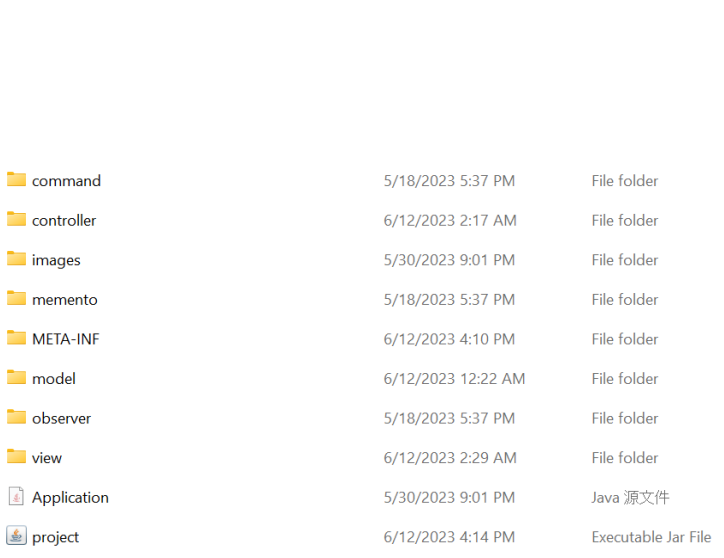


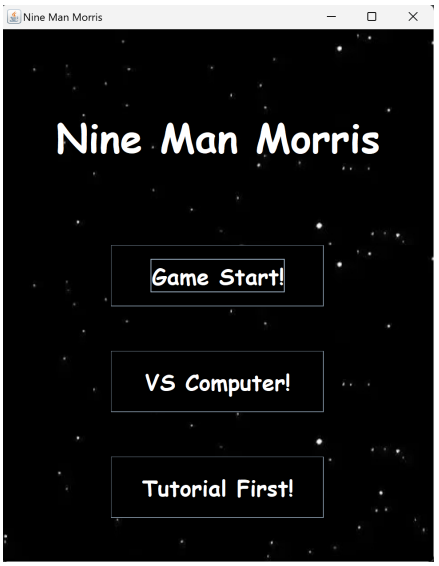

The next step is to click on the `Build` and then `Build Artifacts`. The jar file will be created in the path project\out\artifacts\project_jar\project.jar. Copy the project.jar file to the src file.




**Running an Executable Jar file**

To run the Jar file, we can locate the file in the file explorer and click on the project executable Jar file. The game interface will appear and users can start playing the game.



# Design Rationales

## Why Designed The Architecture This Way

**Since the first sprint, our project has consistently adhered to the Model-View-Controller (MVC) pattern, and we have not made any modifications to it due to its numerous advantages.** The MVC architecture forms the core structure of our game. The "View" component handles user interaction with the interface and includes classes such as GameView, BoardView, MessageView, and InitialPageView, which define the visual presentation of the application. A well-designed view is essential for providing an engaging user experience, as it determines how the application appears and captures user input. On the other hand, the "Model" encompasses the game logic, consisting of critical components like the board, nodes, players, token colors, game state, and more.

It governs the functioning of the game and ensures that the code accurately represents the rules and mechanics. The model's correctness is of utmost importance, as any inconsistencies can lead to bugs and disrupt the game's balance. By separating the internal logic from the external interface, we gain better control over the game's manipulation. We prioritize achieving a 100% correct model to ensure correctness. Meanwhile, the "Controller" handles the interaction between the model, view, and user. It processes user requests, such as mouse clicks, retrieves the clicked node, manages player turns, verifies the game state, and performs related tasks. The controller invokes model methods to drive the game's execution and instructs the view on what to display. The adoption of MVC allows us to effectively distribute responsibilities and simplifies code maintenance.

## Design Pattern Used Since Previous Sprints

### Observer

To ensure real-time updates and improve efficiency in rendering changes to the user interface (UI), we adopted the observer pattern. By registering the UI component (BoardView) as an observer to the game model, it receives automatic notifications whenever the board pieces change, such as when a token is set. This approach not only enables prompt UI updates but also promotes code extensibility, as new observer classes can be added without modifying the game model.

## Singleton

The singleton design pattern ensures that a class can only have a single instance accessible through a public method from anywhere. In our game, we implemented the GameController class to handle user clicks and manage the game's current state. We opted for the singleton design because we deemed it unnecessary to have multiple GameController objects. This approach guarantees the creation of a single instance and provides convenient global access to the class's functionalities.

## Command

The command pattern transforms a request into a self-contained object, separating the implementation details from the GameController. By using a Command interface and concrete classes, the GameController can initiate commands and delegate their execution. This approach reduces coupling, supports code reusability, and enables the introduction of new commands without affecting existing code.

## Memento

The Memento design pattern is implemented in Sprint 3 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 preserve the current game state before entering this mode, ensuring a seamless restoration once the token removal process is completed. To accomplish this, we leverage the "saveStateToMemento" method implemented previously, 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 then retrieves the saved state from the GameStateMemento object, effectively returning the game to its previous state.

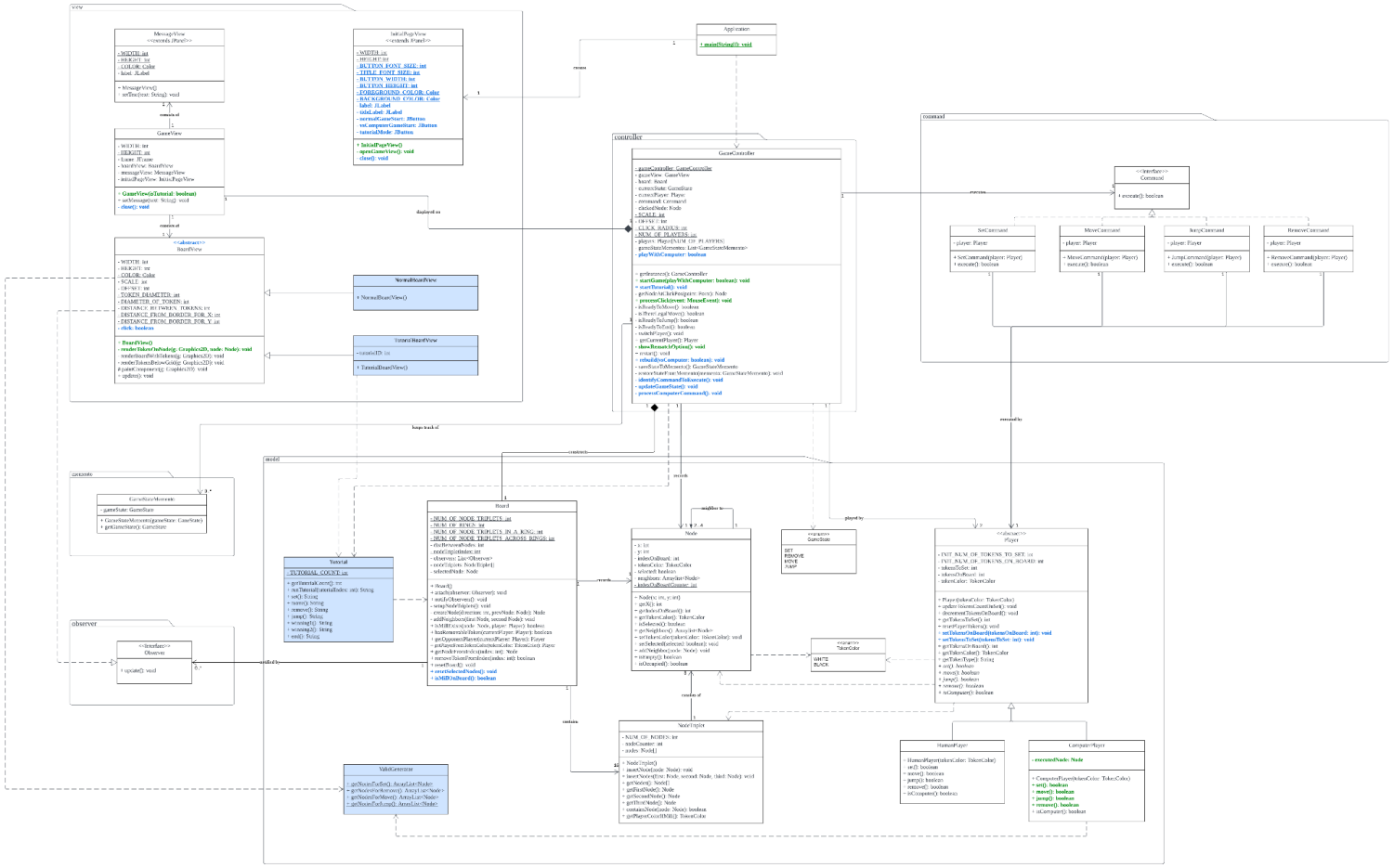
# Revised Class Diagram

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.

Before: [Lucidchart Link for Sprint 3 Class Diagram]

After: [Lucidchart Link for Sprint 4 Class Diagram]

Final Class Diagram:



# What and Why Revised The Class Diagram

## i. Newly Added Classes

1. ValidGenerator
   The ValidGenerator class plays a crucial role in generating valid and legal moves based on the current game board's situation. Its primary purpose is to provide methods for identifying and retrieving specific types of nodes that can be utilized by players to execute different actions in the game.

   By employing the ValidGenerator class, both the HumanPlayer and ComputerPlayer classes can generate valid moves for each action without duplicating code. This implementation adheres to the ***"Don't Repeat Yourself"*** principle, promoting code reusability and reducing redundancy.

   To ensure adherence to the ***Single Responsibility Principle***, this class is solely responsible for generating valid and legal moves based on the current game board's situation. Additionally, the class follows the ***"Command-Query Separation"*** principle by ensuring that none of its methods perform both command (modify state) and query (retrieve information) operations. This separation improves code clarity and maintainability.

2. Tutorial
   The Tutorial class is dedicated to managing the game tutorial by offering various scenarios for setting, moving, removing tokens, and illustrating winning conditions. Its primary function is to guide users through interactive examples, aiding their comprehension of the game mechanics. Once the tutorial is complete, users have the option to initiate a genuine game against either computer or human opponents. This class plays a vital role in delivering an engaging and educational learning experience, empowering users to grasp and apply the game's rules and strategies effectively.

   To ensure adherence to the ***Single Responsibility Principle***, the Tutorial class exclusively handles the game tutorial without mixing unrelated functionalities. Additionally, it upholds the ***"Command-Query Separation"*** principle, ensuring that its methods strictly perform either command operations (modifying state) or query operations (retrieving information). This clear separation enhances the readability and maintainability of the code.

   Furthermore, by avoiding excessive use of literals, the Tutorial class employs private constant class attributes to represent important values. This approach

enhances code readability and maintainability, making it easier to understand and modify the code when needed.

3. NormalBoardView
   The NormalBoardView class is an essential component of a board game application, specifically designed to provide a visual representation of the normal game board. It serves as an extension of the BoardView class and is primarily responsible for rendering the game board and managing user interactions within it.

   During initialization, the NormalBoardView sets up various view components, including a "Hint" button that offers players assistance by providing hints regarding valid moves. This button enables users to receive guidance during gameplay. Additionally, the class effectively handles mouse click events occurring on the game board. It forwards these events to the GameController, which processes them further to determine the appropriate actions based on the game's rules.

   A notable feature of the NormalBoardView is its integration with the ValidGenerator class. This class plays a significant role in identifying and determining valid moves according to the current state of the game. By leveraging the functionality provided by ValidGenerator, the NormalBoardView can highlight specific nodes on the game board that correspond to valid actions such as placing tokens, removing opponent tokens (excluding those forming a mill), moving player tokens to neighboring unoccupied nodes, or jumping to vacant nodes.

   By strictly focusing on the display and management of user interactions with the game board, the NormalBoardView adheres to the ***Single Responsibility Principle***, ensuring a clear separation of concerns within the board game application.

4. TutorialBoardView
   The TutorialBoardView class serves as a specialized view component designed specifically for the tutorial board in the game. It extends the BoardView class to provide a dedicated user interface tailored for tutorial-based gameplay, enhancing the overall tutorial experience.

   Upon creation, TutorialBoardView initializes the tutorial ID and integrates a "Next" button into the view. This button enables users to navigate through the tutorial steps, progressing at their own pace. The class also handles mouse click events, enabling interaction with the tutorial board and facilitating user engagement.

Through the implementation of mouse click event handling, TutorialBoardView effectively communicates user actions to the GameController, which manages the game's underlying logic. Additionally, it prints the coordinates of the click, providing feedback on user interactions. Furthermore, it triggers the execution of computer commands after a brief delay, simulating a dynamic tutorial experience that guides players through the game.

To enhance tutorial progression, the class includes an ActionListener that is attached to the "Next" button. This listener retrieves the associated GameView from the GameController and iterates through each tutorial step. As it advances, it executes the corresponding tutorial logic and displays relevant messages in the GameView, keeping players informed and guiding them through the tutorial seamlessly.

By strictly focusing on delivering a dedicated user interface for tutorial-based gameplay, the TutorialBoardView upholds the principles of the ***Single Responsibility Principle***. This principle ensures a clear separation of concerns within the broader context of the board game application, improving maintainability and promoting modular design.

### ii. Methods Additions and Modifications to Existing Classes

1. GameController
    a. playWithComputer attribute (Newly Added Attribute)
    This attribute is utilized to store the current game mode, specifically indicating whether the game is being played against a computer opponent. This attribute helps in distinguishing between the different gameplay scenarios and enables appropriate functionality and behavior adjustments tailored to the computer mode.

    b. startGame method (Modified Method)
    The startGame method plays a vital role in initializing all the essential components and displaying relevant messages to the players. In order to accommodate the inclusion of a computer player as one of the participants in our Nine Man Morris game, we have made modifications to this method. Now, when the user chooses to play against the computer, the startGame method ensures that the computer player instance is added to the player list and sets the playWithComputer attribute to true. It will initialize a gameView made for normal games, which contains a hint button in the middle.

    c. processClick method (Modified Method)

After careful analysis, we identified that the processClick method contained excessively long code, leading us to the decision of decomposition. By breaking down the method into smaller, more manageable sections, we can improve code readability and maintainability. Additionally, we recognized the presence of common code within the method, which will be extracted into separate methods. This approach not only streamlines the process of handling user interactions but also facilitates the invocation of these methods when processing commands from the computer player. The objective is to minimize repetitive code and increase code reusability throughout the game execution, resulting in a more efficient and maintainable implementation.

d. identifyCommandToExecute method (Newly Added Method)
   This method serves the purpose of determining the appropriate command to be executed based on the current game state. It analyzes the game state and creates a command instance that aligns with the specific conditions and requirements of that state. By dynamically generating the command based on the state, the method ensures that the game proceeds with the correct actions and logic in accordance with the ongoing gameplay.

e. updateGameState method (Newly Added Method)
   This method is responsible for updating the game state based on the current situation of the game. It executes the appropriate command and performs necessary verifications to determine the resulting game state. By executing the command and analyzing the outcomes, the method accurately reflects the current state of the game, allowing for proper gameplay progression and decision-making.

f. processComputerCommand method (Newly Added Method)
   This method is designed to handle the computer command based on the current game state and its situation. Similar to the processClick method, it invokes the identifyCommandToExecute and updateGameState methods to determine and execute the appropriate command. By encapsulating the logic for processing the computer command within this method, it ensures consistent and accurate gameplay for the computer player in accordance with the game state.

g. startTutorial method (Newly Added Method)
   This method serves as the method to initiate a tutorial mode game. It will create a gameView and boardView specifically for tutorials, which contains a “Next” button in the middle of the application. Other than

creating the views, board and player, a welcome message for the tutorial mode is displayed.

h. Rebuild method (Newly Added Method)
This method will be used to start a new game after the tutorial ends. The tutorial mode uses TutorialBoardView, which is different from the NormalBoardView. In order to switch from tutorial view to the normal game view, the restart method is not enough as it does not change the view. This rebuild methods fill the gap and set the correct view and correct player, which can be human or computer to prepare for a new game.

i. showRematchOption (Modified Method)
At first, this method worked by bringing up a dialog at the end of the game, giving the player the option to either end the game or continue playing. However, we have recently introduced an exciting new feature called "Computer VS Human." Now, when the game reaches its conclusion, players have even more choices available to them. They can opt to continue playing in the current mode, switch to the new "Computer VS Human" mode(if currently in normal mode), switch to the normal mode(if currently in "Computer VS Human" mode), or gracefully conclude the game. This enhanced feature gives players the flexibility to tailor their gaming experience to their preferences.

2. InitialPageView
    a. Constructor
    The constructor of the InitialPageView class is responsible for setting up the initial page of the Nine Men Morris game. It initializes the JFrame, sets its properties such as title, size, and location, and creates the necessary components such as labels and buttons. The constructor also adds action listeners to the buttons to handle user interactions. Finally, it displays the JFrame to make it visible to the user.

    b. openGameView method (Newly Added Method)
    The openGameView method is a private method used within the InitialPageView class. It is invoked when the user clicks on the "Game Start!" or "VS Computer!" buttons. This method is responsible for initiating the game by creating an instance of the GameController class and calling its startGame method. The method accepts a boolean parameter, playWithComputer, which indicates whether the user wants to play against the computer or not. Based on this parameter, the GameController will be configured accordingly. By invoking the startGame method, the game is initialized and the appropriate game view is displayed to the user, allowing them to begin playing.

c. close method (Newly Added Method)
   The close method in the InitialPageView class closes the initial page view by disposing of the JFrame instance, ensuring a smooth transition to subsequent views or actions in the application.

3. Player
   a. setTokensOnBoard method (Newly Added Method)
      In previous versions, the tokensOnBoard will be updated automatically when an action such as set or remove is performed. In sprint 4, to make the tutorial mode, our team decided to make a series of predefined positions. The tokensOnBoard counter should be set properly as it affects the winning condition.

   b. setTokensToSet method (Newly Added Method)
      In previous versions, the tokensToSet attribute will be decremented automatically when a set action is performed. In sprint 4, to make the tutorial mode, our team decided to make a series of predefined positions. The tokensToSet counter should be set properly as it helps the game engine to determine if the next gameState is a Set State.

4. ComputerPlayer
   a. set method (Modified Method)
      The set() method is responsible for the computer player's placement of a token on the game board. It retrieves the game controller and the node triplets from the board. It identifies empty nodes on the board and selects a random empty node. Then, it sets the token color of the selected node to the computer player's token color, updates the tokens count on the set, notifies the board observers of the change, and stores the executed node.

   b. move method (Modified Method)
      The move() method handles the movement of a token by the computer player. It retrieves the game controller and the node triplets from the board. It identifies the occupied nodes controlled by the computer player that have at least one empty neighbor. It selects a random occupied node and a random empty neighbor node. If the destination node is empty, is a valid neighbor of the source node, and has the same token color as the current player, the token is moved from the source node to the destination node. The board is then notified of the change, and the executed node is stored.

   c. jump method (Modified Method)

The remove() method handles the removal of an opponent's token by the computer player. It retrieves the game controller and the board. It identifies the opponent's tokens that are not part of a mill configuration. It selects a random token from the removable tokens and retrieves its index on the board. The method calls the board's removeTokenFromIndex() method to remove the token from the specified index. If successful, it returns true; otherwise, it returns false.

d. remove method (Modified Method)
The jump() method handles the jumping of a token by the computer player. It retrieves the game controller and the node triplets from the board. It identifies the computer player's occupied nodes and all empty nodes on the board. It selects a random occupied node and a random empty node. If the destination node is empty and the source node has the same token color as the current player, the token is moved from the source node to the destination node. The board is then notified of the change, and the executed node is stored.

5. Application
    a. main method (Modified Method)
    The Application class serves as the entry point for the Nine Men Morris game. Its main method is responsible for initiating the game by instantiating the InitialPageView class and creating an instance of it. This allows the game to start and provides the initial page view to the user.

6. ValidGenerator (Newly Added Class)
    a. getNodesForSet method
    This method serves the purpose of identifying and retrieving all the unoccupied nodes or positions on the game board. By utilizing this method, we can obtain a comprehensive list of available positions that players can consider for placing their game tokens during the SET phase. At the same time, this method is also leveraged in the hint feature to highlight the vacant positions on the game board, offering players valuable information about the available options.

    b. getNodesForRemove method
    This method is responsible for identifying and retrieving opponent tokens on the game board that can be removed, excluding those that are part of a mill formation. By utilizing this method, players can access a comprehensive list of positions containing opponent tokens eligible for removal. At the same time, this method also plays a crucial role in the hint feature by highlighting the opponent's tokens that can be strategically removed from the game board.

c. getNodesForMove method
   This method plays a pivotal role during the MOVE phase of the game. When the player has not yet selected a node on the board, this method is utilized to identify all the player's tokens that can be moved if one of their neighbouring nodes is unoccupied. However, once the player has clicked on a movable token, the method retrieves all the neighbouring nodes of the selected token that are currently vacant. At the same time, this method is instrumental in the hint feature as it not only highlights tokens that can be moved but also identifies the empty positions adjacent to the selected token that are viable options for relocation. By utilizing this functionality, players are provided with clear visual cues, indicating both the movable tokens and the available empty positions to facilitate their decision-making process during the MOVE phase.

d. getNodesForJump method
   This method plays a vital role during the JUMP phase of the game. Initially, this method identifies all tokens belonging to the player, as they are inherently jumpable to other nodes on the board. However, once the player clicks on a jumpable token, the method retrieves all currently vacant nodes on the board, representing the potential destinations to which the token can be jumped. At the same time, this method is used in the hint feature by highlighting tokens that can be jumped and also identifying available empty positions as potential jump destinations, providing valuable guidance to players during the game.

6. Board
   a. resetSelectedNodes method (Newly Added Method)
      This method serves the purpose of clearing any highlighted nodes in the game. By iterating through each node on the board, the method identifies if a node is currently selected. If a selected node is found, its selected state is reset to false, effectively removing the highlighting. This functionality ensures a clean and visually uncluttered game board, providing players with a fresh starting point devoid of any previous selections.

7. BoardView
   a. BoardView abstract class
      The class is changed from a concrete class to an abstract class. Before sprint 4, the boardView only needs to handle the normal game view. After sprint 4, there are tutorial views and normal views, each requiring a different implementation for constructors. Meanwhile, they share the same code for the other methods. This abstraction makes the code easier to maintain and extend for all kinds of board views. This obeys

the ***Open-Closed Principle (OCP)*** and ***Don't repeat yourself*** principle**.**

b. click attribute (Newly Added Attribute)
   This attribute serves as an indicator of the current click state on the board. It plays a dual role, both tracking whether a click event is currently active and providing functionality to toggle off the hint features if they were previously enabled. By utilizing this attribute, the system can accurately determine the click status and adjust the hint display accordingly.

a. Constructor
   Constructor method now only sets height and width and attaches the board to the boardView. The other implementations will be implemented in the subclass.

b. renderTokenOnNode method (Modified Method)
   This method has been enhanced to include a highlight feature for the hint functionality. It checks if the current node has been selected to be highlighted. If the node is marked for highlighting, a pink-colored oval shape is used to visually distinguish and emphasize the node. This visual cue effectively draws attention to the selected node, aiding players in identifying important elements during gameplay.

8. NormalBoardView
   a. Constructor
      The NormalBoardView class extends BoardView. It has a constructor which contains the implementation of a normal game board view. This is modified from the previous code used in BoardView when it is a concrete class. The constructor has been modified to incorporate a toggleable hint button, which provides players with the ability to reveal and highlight all the possible legal moves based on the current game state (SET, REMOVE, MOVE, or JUMP) when clicked. The inclusion of a hint button greatly enhances the player's experience by providing informative guidance and empowering them to make informed decisions. Additionally, when playing in computer mode, the hint button is automatically disabled during the computer's turn. This precautionary measure prevents human players from inadvertently clicking the hint button and unintentionally interfering with the ongoing computer move. By disabling the hint button during computer moves, the risk of accidental disruptions is minimized, ensuring a seamless and uninterrupted gaming experience for the players.

9. TutorialBoardView

a. Constructor
   The TutorialBoardView class extends BoardView. It has a “Next” button in the middle of the interface. It is used to proceed from one tutorial to another. It has a counter to track the current tutorial id. The tutorial id is used to loop through all tutorials from the Tutorial class. For each tutorial, a predefined position is set and the relevant message returned from Tutorial class will be displayed.

10. Tutorial
    a. TUTORIAL_COUNT attribute
       This is a static final variable which records the number of tutorials available in the class. It is together with the tutorial id to loop through all tutorials. It is set to 7, as there are 6 tutorials and 1 ending case.

    b. runTutorial method (Newly Added Method)
       The method switches through the tutorialID and runs the corresponding tutorial. It has cases 0 to 6 which refers to set, move, jump, remove, winning condition 1, winning condition 2 and ending case. It also returns a message which is the description of each tutorial.

    c. set method (Newly Added Method)
       This method starts with an empty board and prompts the user to set a star token at an empty node. It covers the set case and introduces the game. The gameState is SET.

    d. move method (Newly Added Method)
       This method starts with some tokens already placed on the board. The player will be prompted to select a token and move it to an adjacent empty node. The gameState is MOVE.
    e. jump method (Newly Added Method)
       This method starts with the star side having only 3 tokens left. The user will be guided to move a token to any empty node on the board. The gameState is JUMP.

    f. winning1 method (Newly Added Method)
       This method shows the winning condition that the player with only 2 tokens left will lose the game. This method does not require the player to perform action, the winning position is shown.

    g. winning2 method (Newly Added Method)
       This method shows the winning condition that the player with no legal moves left will lose the game. This method does not require the player to perform action, the winning position is shown.

h. end method (Newly Added Method)
   This method ends the tutorial. Then a small window will pop up and ask the player if he or she wants to start a real match after learning from the tutorial. There are two options, to play with a computer or play with humans.

11. GameView
    a. close method (Newly Added Method)
       The close method in the GameView class closes the game view by disposing of the JFrame instance, ensuring a smooth transition to subsequent views or actions in the application.

# When Advanced Feature Was Finalized

## Advanced Feature 1: Human VS Computer

The Advanced Feature "Computer VS Human" was completed by JiaChen and YueYang. JiaChen is responsible for implementing the "Jump" and "Remove" actions, while YueYang is in charge of the "Set" and "Move" actions. According to the sprint plan, we aimed to complete the advanced features of the Nine Man Morris game by sprint 4. Due to an extension, we officially finished developing the feature at 6 pm on June 11th, successfully addressing all encountered bugs. The majority of our code does not require extensive rewriting, due to the inherent extensibility of our software design. Our architecture allows for seamless extension without significant impact on the original codebase.

During Sprint 2, we introduced the singleton design pattern to enhance the GameController class. This pattern enables convenient global access to a single GameController instance throughout the application. We strategically employed this design choice in the ValidGenerator class, specifically for implementing the functionality related to the computer player feature.

By leveraging the singleton pattern in the GameController class, we achieved effortless access to vital game information, such as the current token positions on the board and the available empty positions. This information proved crucial for enabling the computer player feature to identify valid moves and generate random moves for computer players. The utilization of the singleton pattern streamlined the integration of the computer player functionality while minimizing disruptions to the existing codebase.

We assessed the difficulty of this feature to be moderate. Although we faced several challenges during the implementation process, overall progress was steady. One issue we encountered involved the immediate placement of the computer player's token when the human player made a move. This lack of delay prevented the human

player from reacting in time and observing the computer player's move. To resolve this, we introduced a timer that delayed the computer player's action by a second or two, providing the human player with enough time to react and strategize effectively.

Furthermore, we discovered a bug after implementing all the actions of this feature. Occasionally, the computer player failed to respond autonomously after the human player made a move or removed tokens. It required a manual click on the screen to prompt the computer player to continue the game. Recognizing a potential issue with the logic of the "processClick" function, we meticulously examined the code and reorganized the logic. After three days of diligent debugging, the logic was restored to its intended functionality.

We are pleased with the smooth performance of this advanced feature. Our collaborative efforts have paid off, resulting in a well-functioning and seamless experience.

## Advanced Feature 2: Tutorial Mode and Hints

The tutorial mode is written by Shun Yao. Some challenges are met during the design and implementation. The design is finalized at the later stage of Sprint 4. We do not need to rewrite the majority of the code as our software is easy to extend without affecting the original code as much as possible. Initially, we plan to have a GameMode abstract class with a NormalMode and a TutorialMode class extending it. The initial idea is to apply a state design pattern. However, challenges like the methodology to switch between tutorials, methodology of delivering a tutorial, increasing complexity of codes to add tutorial arise. This increases the difficulty of requirements.

To address them, we acknowledge that a separate class is needed to store all the collections of different tutorials for different situations. We also agree on the fact that there will be predefined positions with guidance for each rule or situation. This results in the Tutorial class. Every tutorial is written in a single method and the runTutorial method will be used to loop through all of them. The code has high readability, maintainability and extensibility. It is very easy to locate a specific tutorial when debugging and to create a new tutorial. Another acknowledgement is that the boardView for normal game and tutorial will be different. A “hint” button is needed for the normal game and a “Next” button is needed for a tutorial. Thus there are normalBoardView and TutorialBoardView classes. Our team also considers the possibility of adding a timer for the switching of different tutorials. This may lead to issues where the user needs to wait even if the instructions are completed. Another potential issue is that the user wants to explore the tutorial longer but the timer runs out of time. Having these considerations, a “Next” button will be a superior solution.

The experience of this requirement is that a good design makes the coding work easier after we change it from the initial design.

The hint feature, developed by DiSheng, was finalized and completed on June 11th, aligning closely with our initial decisions from Sprint 1 while undergoing minor design refinements in Sprint 4. The implementation process proved to be smooth and efficient, largely attributed to the effective design practices we had employed throughout the project. In particular, during Sprint 2, we employed the singleton design pattern for the GameController class, which provided a convenient global access point to the GameController instance. This design choice proved advantageous in the ValidGenerator class, where the hint feature's functionality is implemented. By leveraging the singleton pattern in the GameController class, we were able to easily access critical game information, such as the current tokens on the board and the empty positions available. This information was essential for the hint feature to highlight the available legal moves and guide players towards making strategic decisions.

However, we encountered a challenge during the implementation of the hint feature, specifically in implementing the toggle functionality. While we were successful in enabling the hints to be toggled on, we faced difficulty in providing a seamless way to toggle them off once they were activated. This limitation could potentially affect the overall gameplay experience, as players may prefer the flexibility of closing the hints after they have gained a better understanding. Through iterative testing and debugging, we were able to identify the root cause of the issue and implement a solution. By keeping track of the state of the hint feature using a boolean flag, we were able to toggle it on and off effectively. When the toggle button is clicked, it triggers the corresponding event handler, which updates the state of the hint feature and adjusts the rendering accordingly. With this, the players are able to effortlessly toggle the hints on or off, providing them with a personalized gaming experience tailored to their preferences.

# Video Demonstration

Youtube Link for our Demonstration