Assignment Sprint 3 - Design Rationale

Section 1 - Additions and Changes to Architecture

1.1 GameState Enum Class - (Ref: Appendix 1A)

We have decided to add a new enum class ***GameState*** to store all the possible states of the game. The enum ***GameState*** provides a standard way for all classes to track the game's current state. ***GameManager*** uses this to decide player actions and call the appropriate action handler methods.

1.2 GameManager Class - (Ref: Appendix 1B)

GameManager now uses ***GameState*** Enums, combined with blocks of switch statements, to invoke the correct handler methods for specific player actions during their turn. The new method *getGameState()* allows ***GamePanel*** (formerly ***Canvas***) to display the game's current state as a message on the GUI. This new change improves communication in between classes, and tracking the game for improved decision making.

Since game over is a state of the game, the new methods *noLegalMoves()* and *hasLessThanThreePieces()* are added to check if the win conditions are met. We have decided to add a new button in the game that allows the player to replay the game. The new method *replay()* will reset the game when the replay button is clicked.

1.3 Position Class - (Ref: Appendix 1C)

In the revised architecture, we have added a new attribute id to the ***Position*** class which uniquely identifies each instance of the ***Position***. It enables us to easily retrieve a particular position from the game board based on its id. The new method *hasNeighbour()* is added to check if a given position is adjacent to a specific position. This method helps to reduce redundancy and promotes code reuse.

1.4 GameBoard Class - (Ref: Appendix 1D)

The revised architecture for the ***GameBoard*** class includes a new method *checkAllPiecesInMill(i:int)*, that is introduced to check if all the pieces of the opponent on the board are part of mills or not. The most important reason to introduce this method was to cater for a special game scenario - after a player creates a mill they are asked to remove one of the opponent's pieces from the board which is not part of a mill, but all the opponent's pieces are part of mills. Therefore, this method checks if all the pieces of the opponent are part of mills or not, and performs appropriate measures.

1.5 View Package Classes: Display, GamePanel (formerly Canvas), and MainMenuPanel - (Ref: Appendix 1E)

One of the major changes is that the ***Display*** class now stands independently as a manager of panels. Display initialises the main window Jframe (changed to a global variable because *gameOverMessage()* uses it) via *initializeDesktopWindow()* and *createMainFrame()*. Display is only concerned with placing or swapping JPanel objects onto the main window via the *setCanvas()*, *swapToGamePanel()*, and *swapToMainMenuPanel()* methods, which are invoked by panel classes (***GamePanel***, ***MainMenuPanel***) upon requesting a change. This

change was made as the introduction of more GUI screens, such as the main menu feature, which required panel swaps when the user goes from the main menu to the game, or back. This also further enforces Display's single responsibility principle and reduces complexity.

Meanwhile, ***Canvas*** was renamed to ***GamePanel***, and has association with ***GameManager*** instead of ***Display***. This shortens the chain of communication, as ***GameManager*** no longer needs to go through ***Display*** to communicate with ***GamePanel***. The following additions were made to ***GamePanel***:

- whitePiecesLeft and blackPiecesLeft: It is added to hold the current number of white/black pieces left to be placed on the game board, it helps in drawing the correct number of pieces in the piece bar.
- isGameOver: A boolean attribute which will be true when game over condition is met, this attribute is used to display game over message and winner text.
- message: It added to hold the message to be displayed to the players which serve as prompts to indicate the valid actions the players can take at a particular moment.
- *finalMessage()*: The reason for its inclusion was that we needed a method to display game over message and winner text. Therefore, this method updates isGameOver boolean and redraws the display.
- *setMessage()*: is a setter for the new attribute message. It is also used by the ***GamePanel*** to modify the message that needs to be displayed.
- *setGameOver()*: is a setter for the new attribute isGameOver. It is used by the ***GamePanel*** to indicate whether the game is over or not.
- *updateMessage()*: is added to change the message to be displayed to the players based on the current game state, i.e. change the attribute message in ***GamePanel***.
- *gameOverMessage()*: is added to display the game over message based on the player who wins the game.

Further changes were made to the *reDrawPieces()* method (formerly *updateCanvas()*) that now accepts 2 integers. The integers refers to the number of white and black pieces left to be placed on the board. The reason for this change is that this method is responsible for making changes to the Game's GUI therefore using these 2 integers helps us to draw the correct number of pieces, in the piece bar, left to be placed on the game board.

A class was also created for the main menu, ***MainMenuPanel***, which initialises all the elements, buttons and their functionalities, for the main menu JPanel. This was made its own class, as it is a separate screen from the other panels with its own responsibility of displaying the title and team name, and allowing the user to choose between different game modes.

1.6 Actions Package: PieceAction Classes - (Ref: Appendix 1F)

An optimization made to Action related Classes, namely ***PieceAction*** and its children, is that all indexes corresponding to positions on the gameboard, "to" and "from" variables, have been removed from all subclasses of ***PieceAction***. Instead, they inherit the superclass class PieceAction's "to" and "from" variables to reduce code duplication (DRY principle). If an index is not in use (Ex: ***PlaceAction*** only has a destination: int to), it is initialised to -1, a

non-existent position. Similarly, the Mill handling methods, *separateMillSet*() for breaking up mills, and *removeFromMill*() for removing individual target pieces from a mill set, are inherited from the ***PieceAction*** superclass by the Action subclasses, so that any action that mutates pieces in a mill only needs to call the superclass methods, rather than host a copy of those methods.

1.7 Piece class

In the revised architecture, we have added a new attribute partOfMill in the ***Piece*** class. This attribute holds a boolean value which will be true if the ***Piece*** is a part of a mill. We have added two new methods *isPartOfMill()* and *setPartOfMill()*. Other classes, such as ***JumpAction***, will call *isPartOfMill()* to check if a Piece is a part of a mill. When a mill is formed, ***GameBoard*** will set the partOfMill to true by calling *setPartOfMill()*. This follows encapsulation where the attribute of a class can be assessed through setter and getter only.

Section 2 - Quality Attributes

2.1 - Usability

This implementation of 9 Men's Morris fulfils the Usability non-functional requirement via several design choices in its user interface. The implementation of a Graphic User Interface with a minimalist design in the main menu makes it simple for users to navigate from screen to screen, whilst providing sufficient detail on what the game is about. The chosen background colour was wood brown to create a sense of familiarity with the original wood brown board of the 9 Men's Morris game, whilst providing sufficient contrast so that both white and black elements of the GUI (Pieces, text, buttons) can be easily read. (Ref: Appendix 2.1A)

Similarly, the game also has visual aids to assist players in the game. The message text on top of the board displays the next available move a player can make or indicates the winner when the game is over, to aid the player in learning how the game mechanics works. Likewise, the display of a piece bar for both teams is a useful indicator for how many pieces a player has to place on the board, and the turn indicator to display the current player's turn. These increase the game intractability by providing useful feedback to players. (Ref: Appendix 2.1B)

2.2 - Correctness

A 9 Men Morris game's accuracy attribute highlights the importance of ensuring the game's operation is reliable and accurate. It involves making sure that the game adheres to the 9 Men Morris rules. This includes ensuring that pieces are placed, slide, jump and removed in accordance with regulations. Referring to appendix 2.2A, 2.2B, 2.2C and 2.2D, respectively show that the piece is placed, the piece slides, the piece jumps and the piece is removed in accordance with the 9 Men Morris rules and regulation. Thorough testing is important in identifying and correcting any logical problems or inconsistencies in the game.

Moving on, sufficient validations were included to prevent players from making invalid moves, which helps to keep the game's integrity and also ensures that the players can only carry out legal moves. During a player's turn, the game correctly uses a piece from that player's piece bar, and places that piece in the clicked spot on the board.(Ref: Appendix

2.2A) Further, the game accurately detects and manages end game conditions, such as recognition of the winning player.(Ref: Appendix 2.2E)

It was ensured that the 9 Men Morris game completely demonstrates the game rules, manages user input correctly and provides a consistent gaming experience by prioritising correctness.

2.3 - Maintainability

Maintainability is one of the quality attributes we have explicitly considered in our design. To achieve this, we have applied OOP principles to our design, such as encapsulation and inheritance throughout our progress. The use of encapsulation ensures that each class' internal implementation details are hidden from outside. For example, declaring the attributes as private and only allow the other classes to access the values of the attributes through the getter methods (refer to Appendix 2.3A). This reduces the risk of side effects when making changes to a part of the code. Furthermore, the use of inheritance helps in achieving code reuse and adding new functionalities without affecting the existing code. For example, calling the superclass' implementation using the super() method (refer to Appendix 2.3B). We have also applied SOLID principles, such as the Single Responsibility Principle. The use of this principle ensures that each class is only responsible for a single, specific task which makes it easier to identify and fix an error.

Besides that, we have added Javadoc and inline comments in the code (refer to Appendix 2.4A-C). It helps developers to understand the purpose and usage of the code, making the code easier to maintain and modify in the future.

Overall, our design has explicitly considered maintainability as a key attribute. By applying OOP principles along with Javadoc and comments, we have developed a game that is easier to maintain, extend and improve in the long run.

Section 3 - Human Values

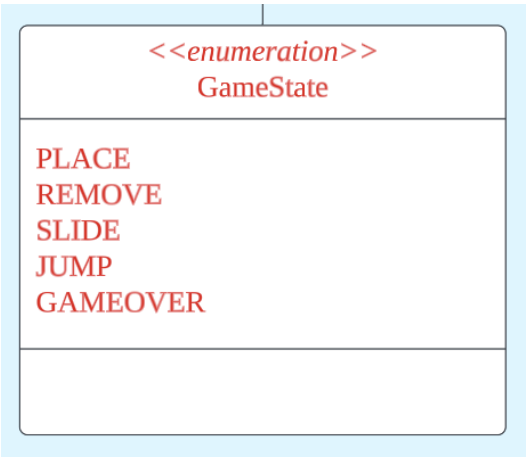
One of the human values that this implementation of 9 Men's Morris honours is: **Respect For Tradition.** This game was implemented as close as possible to the original game from 14th Century Europe, even mimicking the wooden colour of the original game board (Encyclopædia Britannica, inc, n.d.) for a sense of familiarity. Thus, preserving the cultural heritage of ancient gaming with minimal cost.

Likewise, another human value in this implementation is: **Helpfulness.** The game aids users through GUI by displaying Piece Bar and Header Message that instructs the user about his next move. Furthermore, the gameplay keeps users entertained, and helps them develop logical thinking, by using the mechanics such as mill forming and positioning of pieces to develop strategies, and win the game.

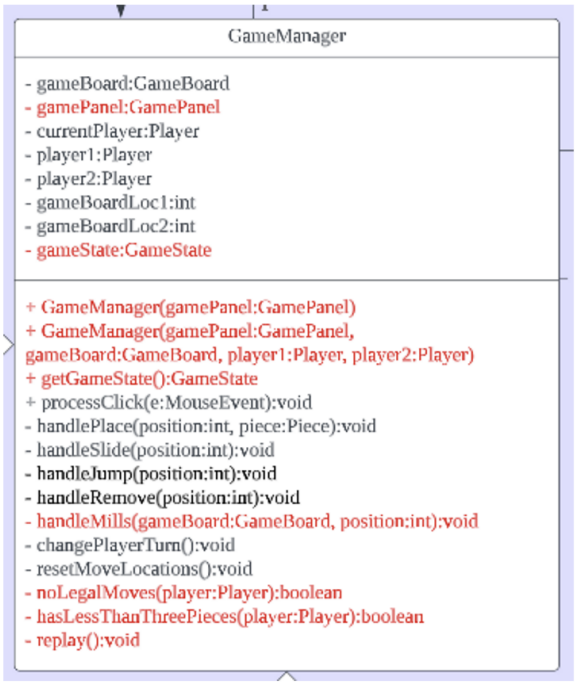
Appendix

Section 1 - Additions and Changes to Architecture

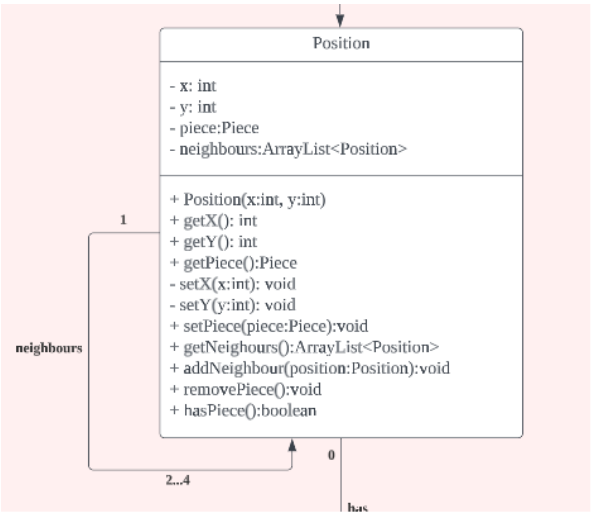
1A - Newly Added GameState Enum Class

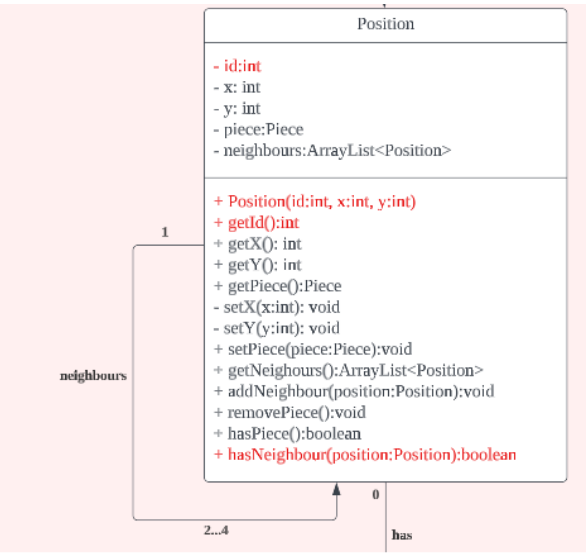


1B - Old GameManager (Left) vs New GameManager (Right)

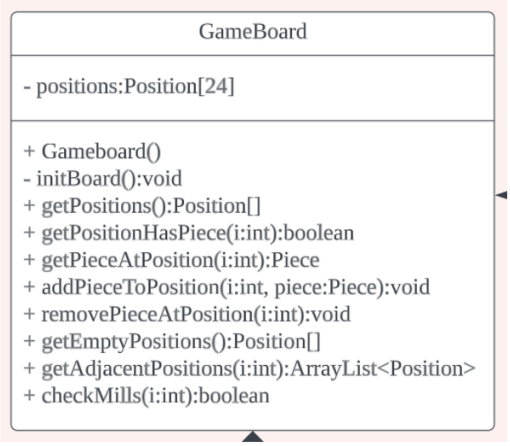


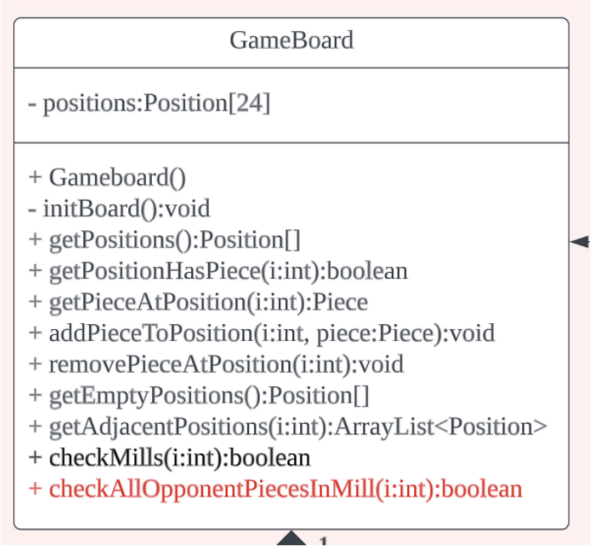
1C - Old Position Class (Left) VS New Position Class (Right)



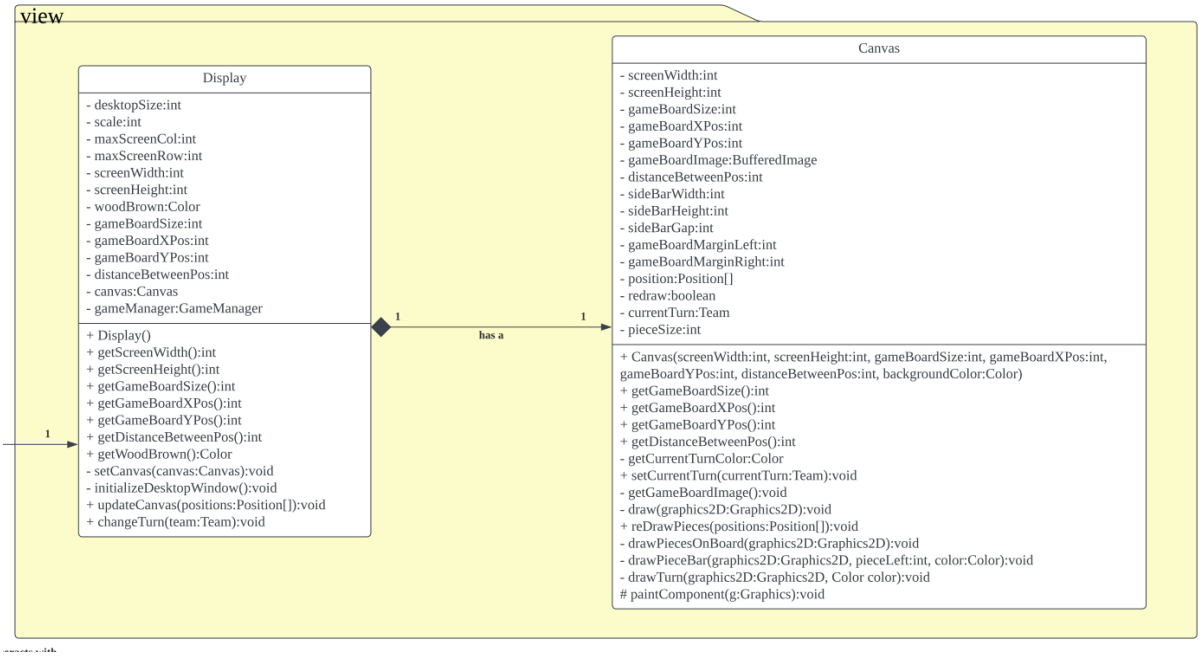


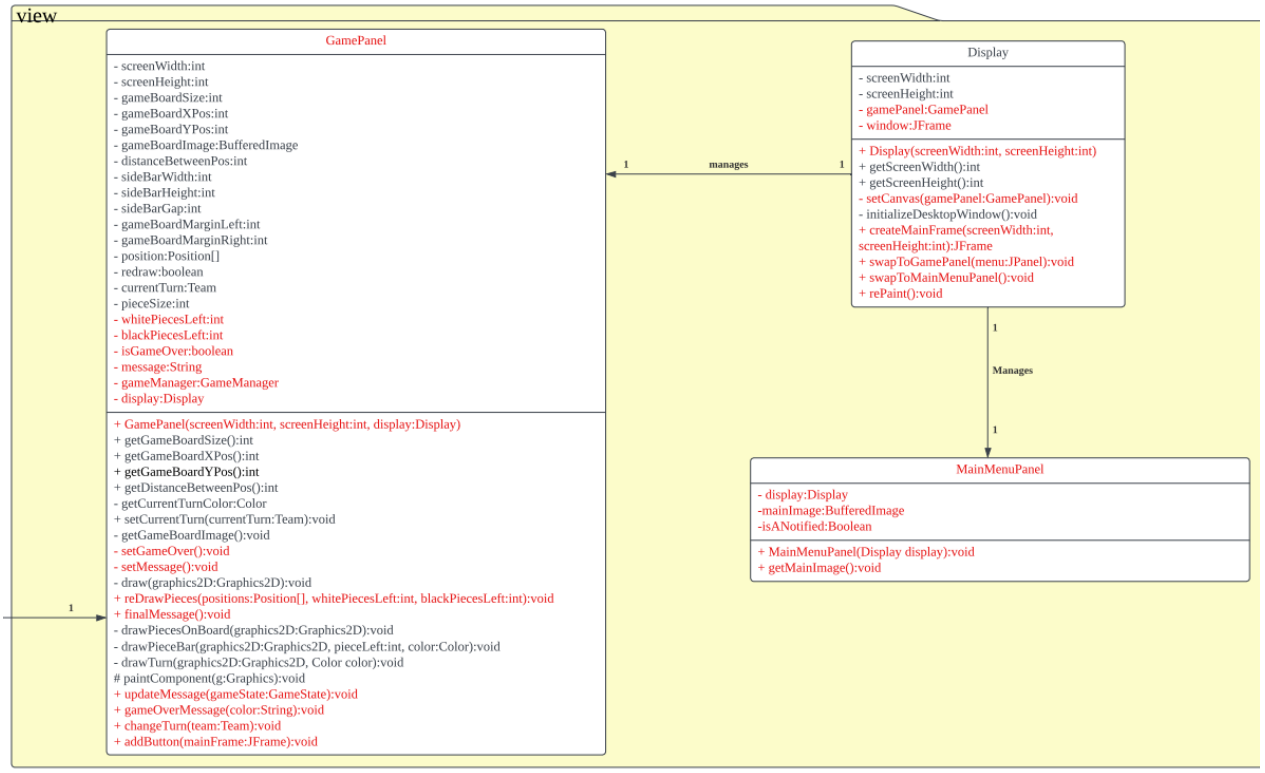
1D - Old GameBoard Class (Left) Vs New GameBoard Class (Right)



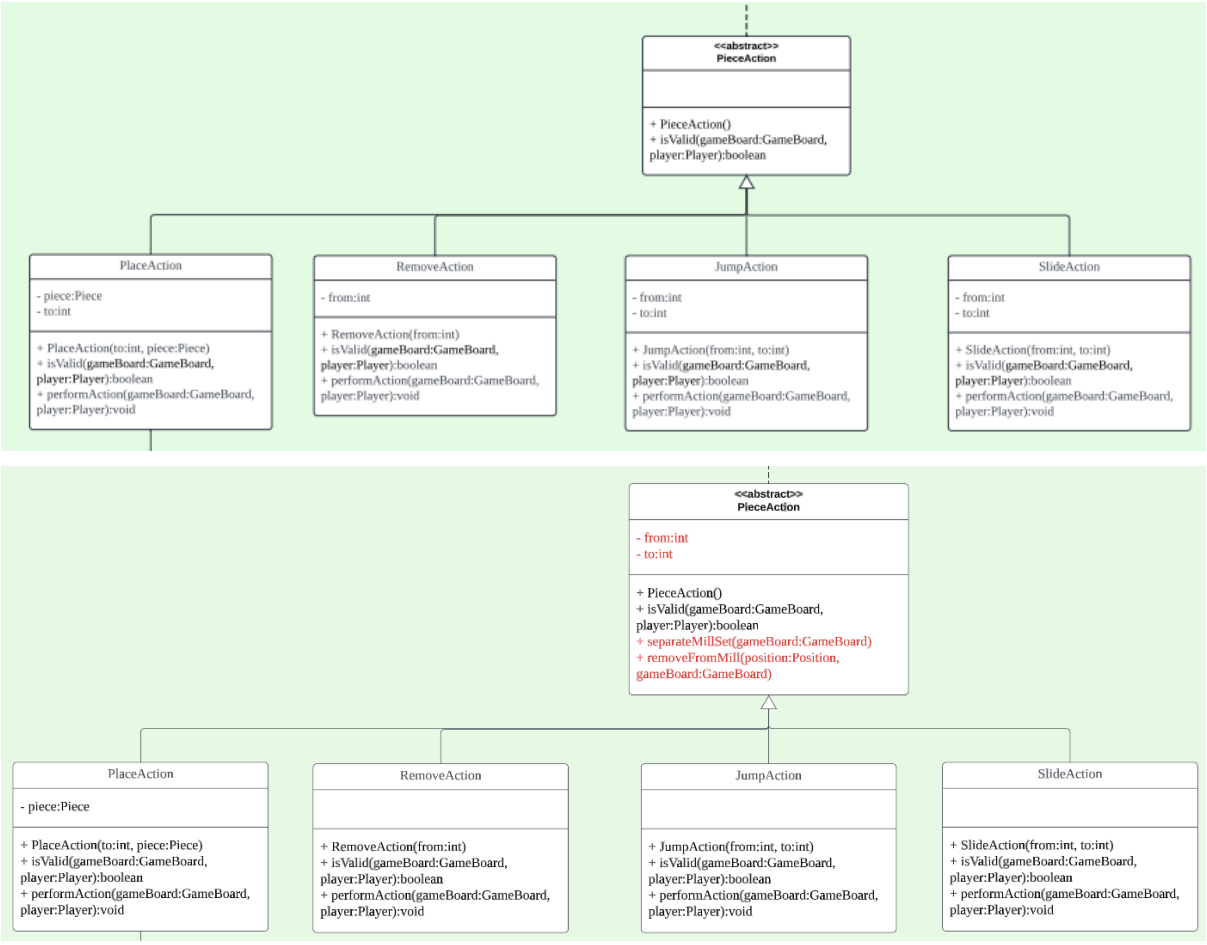


1E - Old View Package: Display, Canvas (Top) vs New View Package: GamePanel, Display, MainMenuPanel (Bottom), showing all related classes:

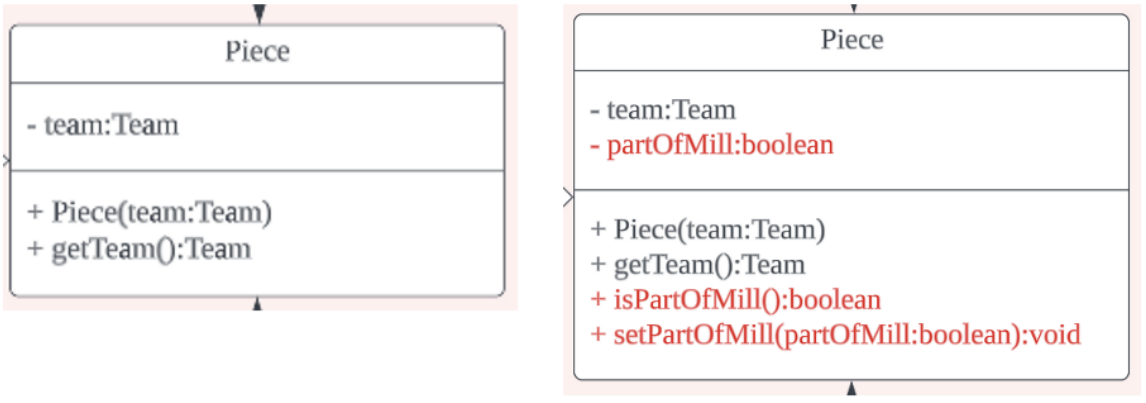




1F - Old Action Classes (Top) Vs New Action Classes (Bottom)

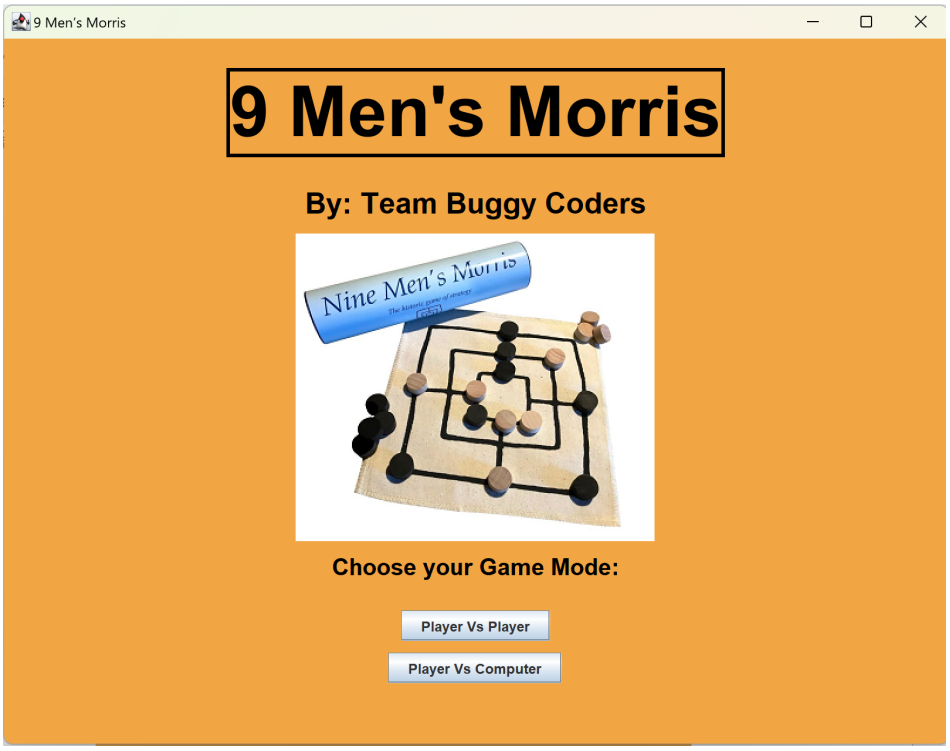


1G - Old Piece Class (Left) Vs New Piece Class (Right)

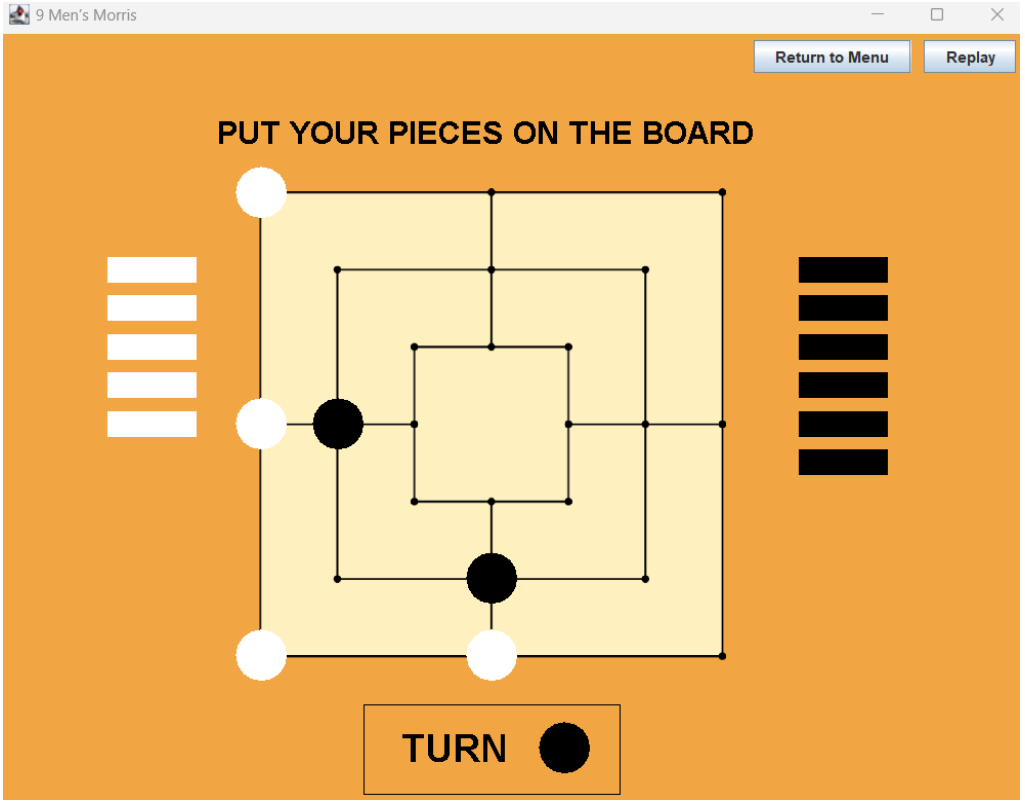


Section 2 - Quality Attributes

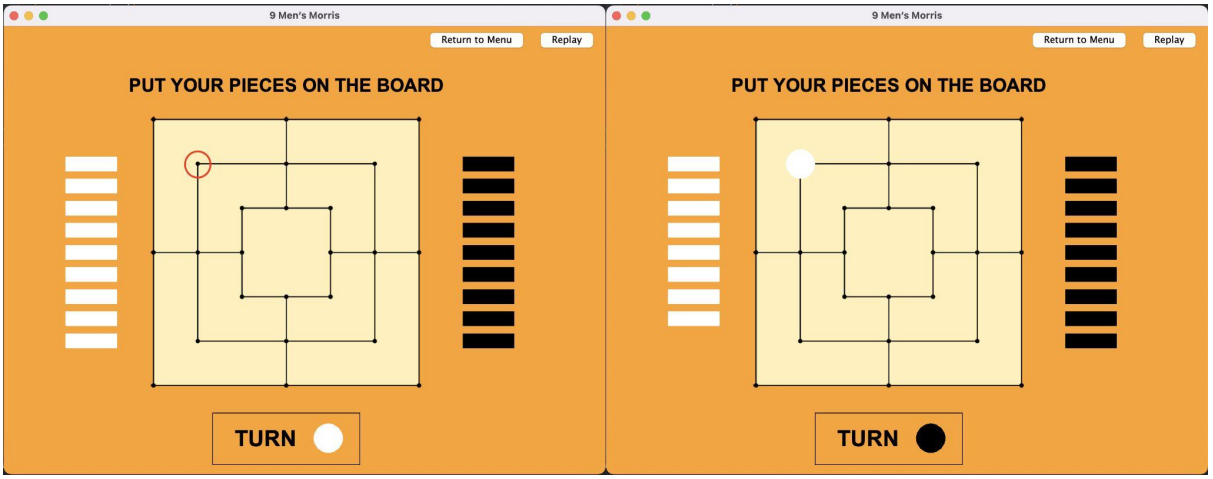
2.1A - Main Menu of the 9 Man Morris Window



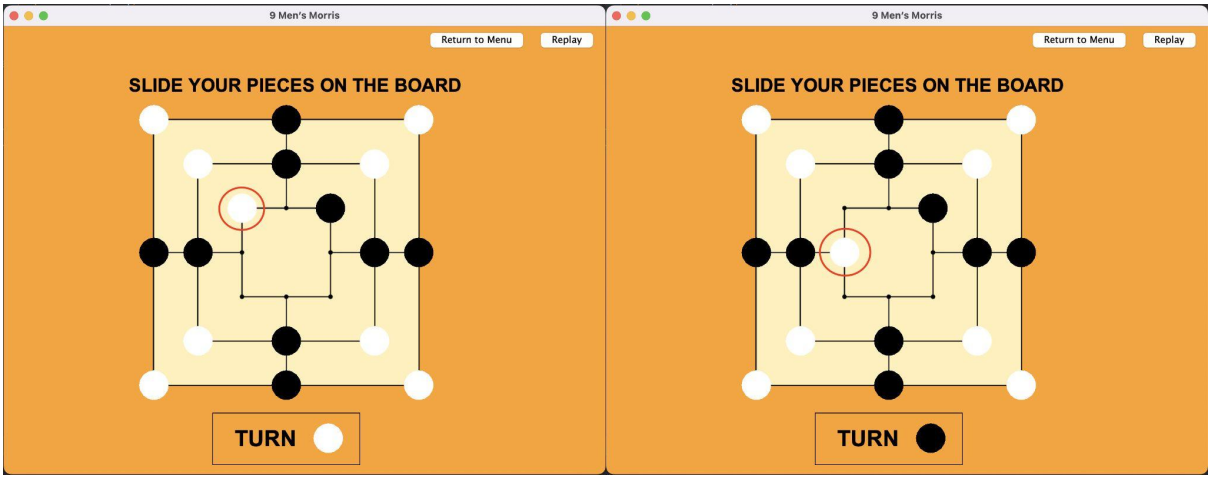
2.1B - Visual Aids: Message Text, PieceBar, and Turn Indicator.



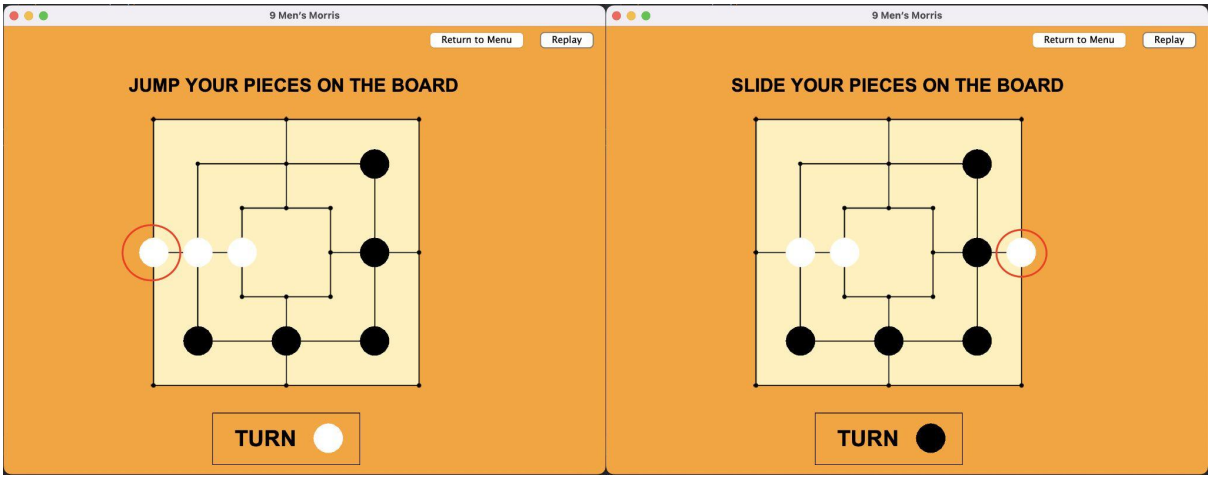
2.2A - PlaceAction and White Piece is removed from the white piece bar



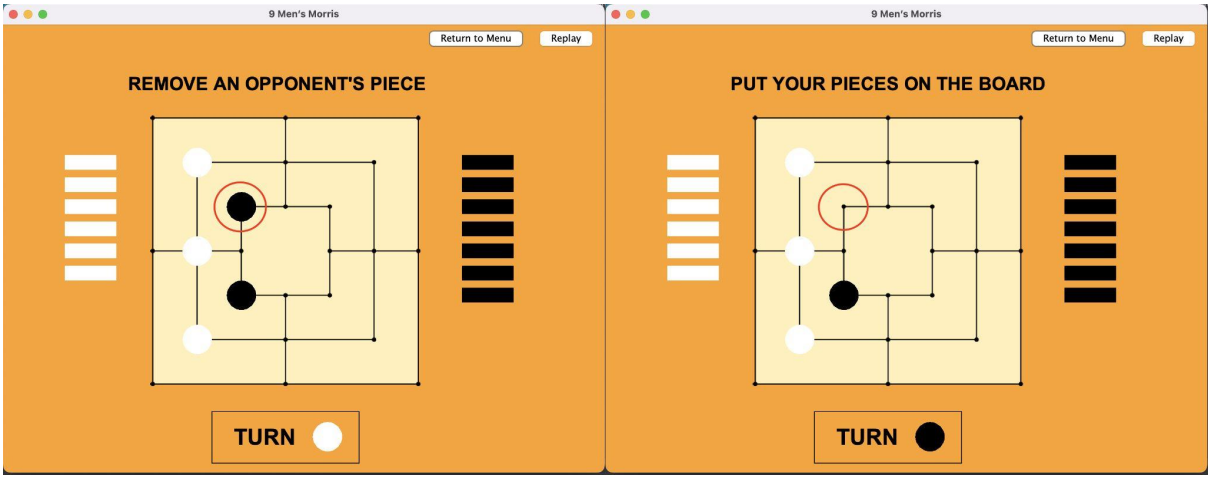
2.2B - SlideAction



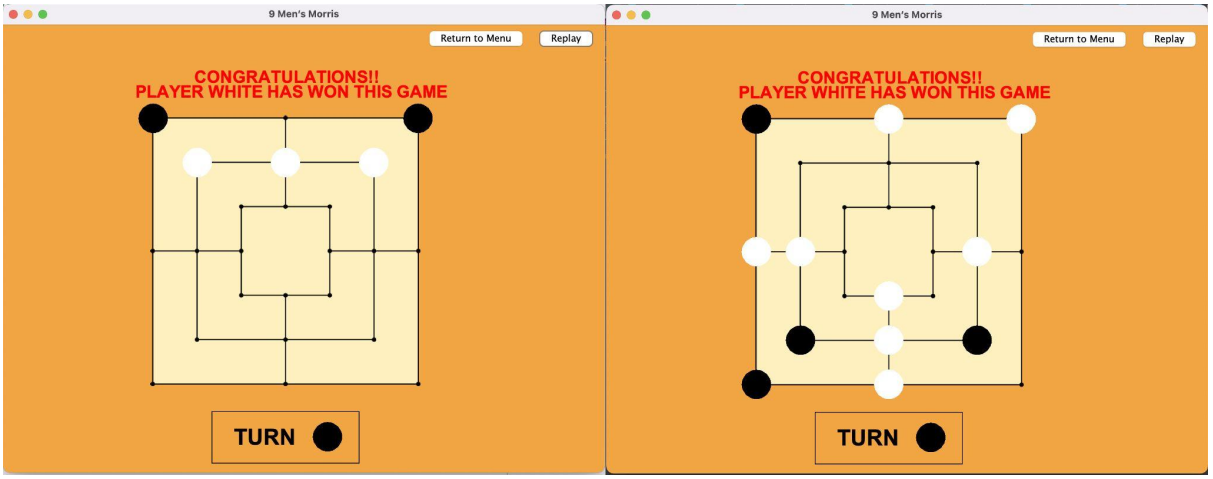
2.2C - JumpAction



2.2D - RemoveAction



2.2E - GameOver (Less than 2 pieces or No legal moves left)



2.3A - Encapsulation

Piece class:

```java
public class Piece {

    // The team (color) the piece belongs to
    2 usages
    private Team team;

    // Whether the piece is a part of a mill
    2 usages
    private boolean partOfMill;

    /**
     * Creates a new Piece object with the given team.
     *
     * @param team The team (color) of the piece
     */
    anneo
    public Piece(Team team) { this.team = team; }

    /**
     * Returns the team (color) of the piece
     *
     * @return The team (color) of the piece
     */
    anneo
    public Team getTeam() { return team; }
```

A method in GameBoard to get the team of the piece:

```java
    public boolean checkAllOpponentPiecesInMill(int i) {
        Team team = positions[i].getPiece().getTeam();
        for (Position x: positions) {
            if (x.hasPiece()){
                if (x.getPiece().getTeam() != team){
                    if (!x.getPiece().isPartOfMill()){
                        return false;
                    }
                }
            }
        }
        return true;
    }
```

2.3B - Inheritance

```java
public class RemoveAction extends PieceAction{

    /**
     * Create an instance of a RemoveAction with the given parameters.
     * @param from indicating the position of opposition's piece which the player wish to remove
     */
    1 usage   How Yu Chern +1
    public RemoveAction(int from) {
        super( to: -1, from);
    }
```

2.4A - Javadoc for constructors

```java
/**
 * GameManager is a class that communicates between the model (GameBoard, etc.) and the View (Display, etc.), to correctly
 * manage and update the state of the game.
 *
 * Reporting to the Game Classes (Board of Directors), GameManager is like the CEO.
 */
13 usages  anneo +3
public class GameManager {
```

2.4B - Javadoc for methods

```java
/**
 * Creates a GameManager instance with the given parameters. Can be used to toggle PvP or PvC modes.
 * @param gamePanel The Display class to manage the GUI
 * @param gameBoard The GameBoard class to manage the Model and Mechanics
 * @param player1 The first player
 * @param player2 The second player
 */
1 usage  anneo
public GameManager(GamePanel gamePanel, GameBoard gameBoard, Player player1, Player player2) {
    this.gamePanel = gamePanel;
    this.gameBoard = gameBoard;
    this.player1 = player1;
    this.player2 = player2;
    this.currentPlayer = player1;
}
```

2.4C - Inline comments

```java
@Override
public void performAction(GameBoard gameBoard, Player player) {

    // Add a new piece on destination position
    gameBoard.addPieceToPosition(this.to, new Piece(gameBoard.getPieceAtPosition(this.from).getTeam()));

    // Check if the piece from start position is a part of mill
    boolean isPartOfMill = gameBoard.getPieceAtPosition(from).isPartOfMill();

    // Remove the piece from start position
    gameBoard.removePieceAtPosition(from);

    if (isPartOfMill) {
        // Handle Removal of Piece from Mill
        super.separateMillSet(gameBoard);
    }
}
```

References

Encyclopædia Britannica, inc. (n.d.). *Nine men's Morris*. Encyclopædia Britannica. https://www.britannica.com/topic/Nine-Mens-Morris