**Team Project Sprint #3**

Team Name:

|  |  |  |  |
| --- | --- | --- | --- |
| **Information provided by the student team** | | **To be used by grader** | |
| **Student name** | **Specific contributions to this sprint** | **Team Score** | **Individual Score** |
| Mannava Jyothi Krishna | Developed the core game logic (game\_logic.py, ai.py), including piece placement, move handling, and mill detection. |  |  |
| Mannava Vignesh | Designed and implemented the graphical user interface (gui.py), ensuring a clear, organized, and visually engaging layout. |  |  |
| Shaik Sumayya Fathima | Created and expanded automated test cases (test\_game.py) and human vs computer functionality (human vs computer) to validate game functionalities. Assisted with GUI refinement (gui.py). |  |  |
| Shreya Saraf | Managed project integration, documentation, coordinated team meetings, and assisted with developing game logic (game\_logic.py). Designed the AI’s decision making algorithm using minmax with alpha-beta pruning. |  |  |

A student without specific contributions shall not receive credit.

1. **Updated User Stories**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **User Story Name** | **User Story Description** | **Priority** | **Estimated effort (hours)** | **Actual effort (if completed)** | **Status (completed, toDo, inProgress)** | **Developer names** |
| 1 | Board Visualization | As a player, I want to see a clear and organized game board to easily identify where to place my pieces. | High | 5 | 5 | Completed | Mannava Vignesh |
| 2 | Piece Placement | As a player, I want to place my pieces on the board to participate in the game. | High | 8 | 8 | Completed | Mannava Jyothi Krishna, Mannava Vignesh |
| 3 | Invalid Move Handling | As a player, I want to receive an error message when I attempt to place a piece on an occupied | Medium | 4 | 4 | Completed | Mannava Jyothi Krishna |
| 4 | Turn Switching | As a player, I want the game to switch turns after a successful move to ensure fair play for both players. | High | 3 | 5 | Completed | Mannava Vignesh, Shaik Sumayya Fathima |
| 5 | Mill Formation Detection | As a player, I want the game to detect when I form a mill so I can remove an opponent's piece. | Medium | 5 | 5 | Completed | Mannava Vignesh, Shreya Saraf |
| 6 | Replay and record | As a player, I want the game to record when I press record and replay it, with options of taking a step back and moving a step forward | Medium | 10 | 10 | Completed | Sumayya Fathima |
| 7 | Save game and Restart | As a player, I want the game to be saved in a JSON file and then reload it and continue | Medium | 8 | 8 | Completed | Mannava Vignesh |
| 8 | AI Piece Placement | As a computer opponent, I want to place a piece on the board so that I can play against the human player. | High | 8 | 8 | Completed | Mannava Jyothi Krishna |
| 9 | Computer Piece Movement | As a computer opponent, I want to move my pieces during the movement phase to simulate gameplay in phases 2 and 3. | High | 8 | 8 | Completed | Sumayya Fathima |
| 10 | Decision-Making (Minimax) | As a computer opponent, I want to make intelligent moves based on a minimax algorithm with alpha-beta pruning to challenge the human player effectively. | Medium | 12 | 12 | Completed | Shreya Saraf |
| 11 | Mill Detection for Computer vs Human | As a computer opponent, I want to detect when a mill is formed so that I can remove an opponent's piece. | Medium | 6 | 6 | Completed | Shreya Saraf |
| 12 | Record Replay for Computer vs Human | As a player, I want the game to record when I press record and replay it, with options of taking a step back and moving a step forward | Medium | 8 | 8 | Completed | Mannava Jyothi Krishna |

1. **Updated Acceptance Criteria (AC)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID and Name** | **AC**  **ID** | **Description of Acceptance Criterion** | **Status (completed, toDo, inPprogress)** | **Developer Names** |
| 1. Board Visualization | 1.1 | Given the game starts, When the GUI is launched, Then the board should display all 24 positions in an organized layout. | Completed | Mannava Vignesh |
|  | 1.2 | Given the board is displayed, When a player places a piece, Then the piece should appear at the selected position with the correct color. | Completed | Mannava Jyothi Krishna, Mannava Vignesh |
| 2. Piece Placement | 2.1 | Given it is a player's turn, When the player clicks on an empty position, Then a piece should be placed, and the turn should switch. | Completed | Mannava Jyothi Krishna |
|  | 2.2 | Given a player has placed all 9 pieces, When the player attempts to place another piece, Then an error message should indicate no pieces left. | Completed | Mannava Vignesh, Shaik Sumayya Fathima |
| 3. Invalid Move Handling | 3.1 | Given a position on the board is occupied, When a player attempts to place a piece there, Then an error message should be displayed. | Completed | Mannava Vignesh |
| 4. Turn Switching | 4.1 | Given a player successfully places a piece, When the move is completed, Then the turn switches to the opposing player. | Completed | Mannava Vignesh, Shreya Saraf |
| 5. Mill Formation Detection | 5.1 | Given a player places a piece that completes a mill, When the mill is formed, Then the player should be prompted to remove an opponent's piece. | Completed | Mannava Jyothi Krishna, Shreya Sara |
|  | 5.2 | Given a mill has been formed, When the player removes an opponent's piece, Then the opponent's piece is removed, and turn switches. | Completed | Mannava Jyothi Krishna, Shreya Saraf |
| 6. Replay and Record | 6.1 | Given the game is in progress, When the player clicks record, Then the game should record each move for replay functionality. | Completed | Shaik Sumayya Fathima |
|  | 6.2 | Given a recorded game, When the player selects replay, Then the moves should play back sequentially with options to step forward or back. | Completed | Shaik Sumayya Fathima |
| 7. Save Game and Restart | 7.1 | Given the player chooses to save the game, When saved, Then the game state should be saved in a JSON file with all relevant details. | Completed | Mannava Vignesh |
|  | 7.2 | Given a saved game file, When loaded, Then the game should reload with all pieces, moves, and turns correctly in place. | Completed | Mannava Vignesh |
| 8. AI Piece Placement | 8.1 | Given it is the AI’s turn, When the AI chooses a position, Then the AI places a piece at the selected position without errors. | Completed | Mannava Jyothi Krishna |
| 9. Computer Piece Movement | 9.1 | Given it is the AI’s turn in phases 2 or 3, When the AI decides to move a piece, Then the piece moves to a valid, available position. | Completed | Shaik Sumayya Fathima |
| 10. Decision-Making (Minimax) | 10.1 | Given it is the AI’s turn, When the board state is evaluated, Then the AI selects the best move using the minimax algorithm with alpha-beta pruning. | Completed | Shreya Saraf |
| 11. Mill Detection for Computer vs Human | 11.1 | Given a mill is formed by the AI, When the mill is detected, Then the AI removes an opponent's piece as allowed. | Completed | Shreya Saraf |
| 12. Record Replay for Computer vs Human | 12.1 | Given a game with human vs computer mode, When replayed, Then all moves should be recorded and replayable. | Completed | Mannava Jyothi Krishna |

1. **Updated Implementation Tasks**

Include the tasks from the previous report and highlight the new tasks with a different color.

Summary of production code

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User Story ID and Name** | **AC ID** | **Class Name(s)** | **Method Name(s)** | **Developer Name(s)** | **Status** | **Notes (optional)** |
| 1. Board Visualization | 1.1 | GameGUI | calculate\_positions, create\_board | Mannava Vignesh | Completed | Organized layout of board positions |
|  | 1.2 | GameGUI | update\_board | Mannava Vignesh | Completed | Correct piece placement and color |
| 2. Piece Placement | 2.1 | Game, GameGUI | place\_piece, handle\_click | Mannava Jyothi Krishna | Completed | Turn switching implemented |
|  | 2.2 | Game | place\_piece | Mannava Vignesh | Completed | Error message on exceeding piece limit |
| 3. Invalid Move Handling | 3.1 | Game | place\_piece, handle\_click | Mannava Vignesh | Completed | Error displayed for occupied positions |
| 4. Turn Switching | 4.1 | Game | switch\_player | Mannava Vignesh, Shreya Saraf | Completed | Turn switch functionality |
| 5. Mill Formation Detection in Human Vs Human | 5.1 | Game, GameGUI | check\_mill, prompt\_remove\_pieceMannava Jyothi | Mannava Jyothi Krishna, Shreya Saraf | Completed | Mill detection and prompt |
|  | 5.2 | Game | remove\_piece, handle\_remove\_click | Mannava Jyothi Krishna, Shreya Saraf | Completed | Piece removal after mill formation |
| 6. Replay and Record | 6.1 | Game, GameGUI | start\_recording, play\_replay | Shaik Sumayya Fathima | Completed | Record and replay functionality |
|  | 6.2 | GameGUI | step\_forward, step\_back | Shaik Sumayya Fathima | Completed | Replay controls for navigation |
| 7. Save Game and Restart | 7.1 | Game | save\_game, load\_game | Mannava Vignesh | Completed | JSON-based game state saving |
| 8. AI Piece Placement | 8.1 | Computer Player | get\_move | Mannava Jyothi Krishna | Completed | Initial position logic for AI |
| 9. Record Replay for Computer vs Human | 8.2 | GameGUI | start\_recording, replay\_game\_state, play\_replay | Mannava Vignesh | Completed | Enhanced replay feature |

Summary of automated test code (directly corresponding to some acceptance criteria)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User Story ID and Name** | **Acceptance Criterion ID** | **Class Name (s) of the Test Code** | **Method Name(s) of the Test Code** | **Description of the Test Case (input & expected output)** | **Status** | **Developer Name(s)** |
| 1. Board Visualization | 1.1 | TestGameGUI | test\_board\_display | Verify the board displays all positions in an organized layout | Completed | Shaik Sumayya Fathima |
| 2. Piece Placement | 2.1 | TestGamePiecePlacement | test\_piece\_placement | Place pieces and verify placement and turn switching | Completed | Shaik Sumayya Fathima |
| 3. Invalid Move Handling | 3.1 | TestGamePiecePlacement | test\_invalid\_move\_handling | Attempt to place on occupied position, expect failure message | Completed | Shreya Saraf |
| 5. Mill Formation Detection | 5.1 | TestGamePiecePlacement | test\_mill\_detection | Form mill and verify mill detection and removal prompt | Completed | Shreya Saraf |
| 6. Replay and Record | 6.1 | TestReplay | test\_record\_playback | Verify record and playback functionality with navigation | Completed | Mannava Vignesh |
| 7. Save Game and Restart | 7.1 | TestGameSaveLoad | test\_save\_load\_game | Verify game state is saved and loaded accurately | Completed | Mannava Vignesh |
| 8. MinMax Piece Placement | 8.1 | TestAIPlayer | test\_ai\_piece\_placement | Verify AI places pieces correctly during phase 1 | Completed | Shaik Sumayya Fathima |
| 9. Computer Piece Movement | 9.1 | TestAIPlayer | test\_ai\_piece\_movement | Verify AI moves pieces correctly during phases 2 and 3 | Completed | Shaik Sumayya Fathima |
| 10. Decision-Making (Minimax) | 10.1 | TestAIPlayer | test\_minimax\_decision | Verify AI makes optimal moves using minimax | Completed | Shaik Sumayya Fathima |

Summary of manual test cases (directly corresponding to some acceptance criteria)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User Story ID and Name** | **Acceptance Criterion ID** | **Test Case Input** | **Test Oracle (Expected Output)** | **Status** | **Notes** | **Developer Name(s)** |
| 1. Board Visualization | 1.1 | Launch the game GUI | Board displays all 24 positions in an organized layout | Completed | Visual inspection | Mannava Vignesh |
| 2. Piece Placement | 2.1 | Player W places at position 0 | Piece appears, turn switches to Player B | Completed | Visual inspection | Shaik Sumayya Fathima |
| 3. Invalid Move Handling | 3.1 | Attempt to place at occupied position | Error message displayed, move rejected | Completed | Error handling verification | Shaik Sumayya Fathima |
| 5. Mill Formation Detection | 5.1 | Player forms a mill at positions 0, 1, 2 | Mill detected, prompt to remove opponent’s piece | Completed | Mill detection confirmation | Mannava Vignesh |
| 6. Replay and Record | 6.1 | Press record, perform moves, then replay | Moves replayed with step controls | Completed | Functional replay check | Shaik Sumayya Fathima |
| 7. Computer Piece Placement | 7.1 | Start game against AI, let AI take its turn | AI places a piece without errors | Completed | Verified via gameplay | All team members |
| 8. Computer Piece Movement | 8.1 | Proceed to phase 2, observe AI moves | AI moves pieces to valid positions | Completed | Verified via gameplay | All team members |
| 9. Decision-Making (Minimax) | 9.1 | Play against AI and attempt to win | AI makes challenging moves, attempts to block player | Completed | Verified via gameplay | All team members |

Summary of other automated or manual tests (not corresponding to the acceptance criteria)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Number** | **Test Input** | **Expected Result** | **Class Name of the Test Code** | **Method Name of the Test Code** | **Status** | **Developer Name(s)** |
| 1 | Place pieces sequentially until limit | Players cannot place more than 9 pieces | N/A | N/A | Completed | Shaik Sumayya Fathima, Shreya Saraf |
| 2 | Attempt to place piece outside board | Error message displayed | N/A | N/A | Completed | Shaik Sumayya Fathima, Shreya Saraf |

1. **Summary of Source Code**

|  |  |  |  |
| --- | --- | --- | --- |
| Production or test code | Source code file name | # lines of code | Developer names and contributions (% of the source code) |
| Production Code | ai.py | 60 | Mannava Jyothi Krishna (60%), Mannava Vignesh (40%) |
| Production Code | game\_logic.py | 195 | Mannava Jyothi Krishna (40%), Mannava Vignesh (60%) |
| Production Code | Gui.py | 441 | Shreya Saraf |
| Production Code | Main.py | 55 | Shaik Sumayya Fathima (50%), Shreya Saraf (50%) |
| Production Code | Utils.py | 103 | Mannava Jyothi Krishna (25%), Shreya Saraf (40%), Mannava Vignesh (25%) |
| Test Code | Test\_game.py | 313 | Shaik Sumayya Fathima (50%), Shreya Saraf (50%) |
| Total | | 1167 |  |

1. **Design Documentation**
2. **User Interface Design**

**Contributors:** Mannava Vignesh

The game's user interface (UI) is built using Tkinter, providing a graphical representation of the Nine Men's Morris board. Key features of the UI include:

* **Board Display:** A visually organized board showing all 24 positions, with lines connecting adjacent positions to represent possible moves.
* **Piece Representation:** Pieces are represented by colored circles—white for Player W and black for Player B (AI). Empty positions are indicated by the board's background color.
* **Interaction Mechanism:** Players interact with the game by clicking on positions to place or move pieces. The UI handles user input and updates the board accordingly.
* **Status Updates:** A status label displays current game information, such as whose turn it is, the current phase of the game, and special messages (e.g., mill formation alerts).
* **Recording and Replay Controls:** Buttons are provided for recording games, replaying moves, and navigating through the replay (rewind, play/pause, step forward/backward).
* **Game Saving and Loading:** Menu options allow players to save the current game state to a file or load a previously saved game.

### 2. Class Diagram

**Contributors:** All team members

The following is an overview of the main classes and their relationships:

* Game **Class**
  + Manages the game state, including the board, player turns, phases, piece counts, and game logic for placing and moving pieces.
  + Methods: place\_piece(), move\_piece(), remove\_piece(), switch\_player(), update\_phase(), check\_win\_condition(), get\_possible\_moves(), is\_over(), make\_move(), save\_game(), load\_game().
* AIPlayer **Class**
  + Implements the AI opponent using the minimax algorithm with alpha-beta pruning.
  + Methods: \_\_init\_\_(), get\_move(), minimax().
* GameGUI **Class**
  + Handles the graphical user interface, including drawing the board, handling user interactions, and updating the display.
  + Methods: create\_menu(), calculate\_positions(), create\_board(), update\_board(), handle\_click(), ai\_move(), save\_game(), load\_game(), start(), start\_recording(), replay\_game\_state(), play\_replay(), and various helper methods for replay controls.
* Utils **Module**
  + Provides utility functions and data structures, such as the adjacency list, mill configurations, and helper functions.
  + Functions: is\_adjacent(), check\_mill(), count\_pieces(), evaluate\_board(), print\_board().
* Main **Module**
  + Entry point for the application, managing game mode selection and initializing the game and GUI.
  + Functions: main().
* TestGame **Module**
  + Contains automated tests for validating game logic and AI functionality.

### 3. Algorithm Design

**Contributors:** Shreya Saraf, Mannava Jyothi Krishna

#### ****Minimax Algorithm with Alpha-Beta Pruning for Nine Men's Morris AI****

The AI uses a depth-limited minimax algorithm with alpha-beta pruning to determine the best possible move. The algorithm evaluates possible future game states to a certain depth and chooses the move that maximizes the AI's chances of winning while minimizing the human player's opportunities.

**Algorithm Steps:**

1. **Initialization (**get\_move **Method):**
   * Set best\_score to negative infinity.
   * Iterate over all possible moves for the AI:
     + Create a deep copy of the current game state.
     + Apply the move to the copied game.
     + Recursively evaluate the move using the minimax function with decreased depth.
     + If the evaluation score is greater than best\_score, update best\_score and best\_move.
2. **Recursive Evaluation (**minimax **Method):**
   * **Base Case:** If the specified depth is zero or the game is over, return the heuristic evaluation of the board.
   * **Possible Moves:** Generate all possible moves for the current player (maximizing or minimizing).
   * **Maximizing Player:**
     + Initialize max\_eval to negative infinity.
     + For each move:
       - Apply the move to a copied game state.
       - Recursively call minimax with maximizing\_player set to False.
       - Update alpha with the maximum of alpha and the evaluated score.
       - If beta ≤ alpha, prune the branch.
     + Return max\_eval.
   * **Minimizing Player:**
     + Initialize min\_eval to positive infinity.
     + For each move:
       - Apply the move to a copied game state.
       - Recursively call minimax with maximizing\_player set to True.
       - Update beta with the minimum of beta and the evaluated score.
       - If beta ≤ alpha, prune the branch.
     + Return min\_eval.
3. **Heuristic Evaluation (**evaluate\_board **Function):**
   * **Piece Advantage:** Calculate the difference in the number of pieces on the board.
   * **Mill Formation:** Count the number of mills formed by each player.
   * **Mobility:** Consider the number of possible moves available.
   * Combine these factors to compute a heuristic score that reflects the board's favorability for the AI.
4. **Move Generation (**get\_possible\_moves **Method):**
   * **Phase 1 (Placing Pieces):** Generate all empty positions where a piece can be placed.
   * **Phase 2 (Moving Pieces):** Generate all valid moves to adjacent positions.
   * **Phase 3 (Flying Phase):** If a player has only three pieces, they can move to any empty position.

### 4. Extensibility

**Contributors:** All team members

#### Extension to Twelve Men's Morris

**Adaptations Required:**

* **Board Expansion:**
  + Increase the board size to accommodate additional positions specific to Twelve Men's Morris.
  + Update the adjacency\_list and mills in utils.py to reflect the new board configuration.
* **Game Logic Modifications:**
  + Adjust initial piece counts to 12 per player.
  + Modify game phase transitions and conditions based on the rules of Twelve Men's Morris.
  + Update methods in the Game class to handle additional pieces and positions.
* **User Interface Adjustments:**
  + Modify the calculate\_positions and create\_board methods in GameGUI to render the expanded board.
  + Ensure the GUI accommodates the larger number of pieces and maintains usability.

**Open-Closed Principle Application:**

* **Open for Extension:**
  + The design allows new game variants by subclassing existing classes (e.g., creating a TwelveMensMorrisGame class that extends Game).
  + New features can be added through inheritance and composition without altering existing code.
* **Closed for Modification:**
  + Core classes like Game, AIPlayer, and GameGUI are designed to be robust and require minimal changes.
  + By adhering to well-defined interfaces and abstraction layers, existing functionality remains unaffected by extensions.

1. **Findings from the Code Review Exercise**

Use the following template to document the findings from the code review.

Participant names: Mannava Jyothi Krishna, Mannava Vignesh, Shaik Sumayya Fathima, Shreya Saraf

Class that was reviewed: Game class

|  |  |  |  |
| --- | --- | --- | --- |
| **Checklist** | **Checklist Item** | **Findings** | |
| Coding Standards | Naming conventions | The Game class and its methods adhere to consistent naming conventions. Class names use CamelCase (e.g., Game, AIPlayer), while methods and variables use snake\_case (e.g., place\_piece, current\_player). Constants are defined in uppercase where appropriate. | |
| Ordering convention of method arguments | Method arguments follow a logical and consistent order throughout the class. For instance, methods typically have self as the first parameter, followed by other parameters in an order that reflects their usage. | |
| Meaningful and valid comments | Most methods include docstrings that explain their purpose and functionality. However, detailed comments regarding preconditions and postconditions are lacking in some methods. Enhancing comments with these details would improve code understandability. | |
| Consistent style of code blocks | The code maintains a consistent style for code blocks. Indentation is uniform (4 spaces), and there is consistent use of blank lines to separate logical sections of code, enhancing readability. | |
| Consistent indentation | Indentation is consistent throughout the Game class, adhering to PEP 8 guidelines. There are no noticeable deviations from the standard indentation practices. | |
| … | Other coding standards, such as line length and spacing around operators, are generally well-followed. The code is clean and readable, with appropriate use of whitespace and adherence to style guidelines. | |
| Design Principles | Good class abstraction and interface | The Game class provides a clear abstraction of the game's mechanics, encapsulating the state and behaviors necessary for gameplay. Methods are logically grouped, and the class interface is intuitive for other components to interact with. | |
| Appropriate visibility of each variable, method, and class | While Python does not enforce access modifiers, the code uses naming conventions (e.g., prefixing with \_) to indicate private methods and variables. Public methods are appropriately exposed, and internal workings are kept encapsulated. | |
| Design by contract (pre/post-conditions) | The code ensures that methods perform as expected, but explicit documentation of preconditions and postconditions is limited. Adding these details in the docstrings would strengthen the reliability and clarity of the code. | |
| Is the Open-Closed Principle violated? | The Game class is designed to be extendable without modifying existing code, adhering to the Open-Closed Principle. New features can be added through subclassing or composition, keeping the core class closed for modification. | |
| Is the Single Responsibility Principle violated? | The Game class primarily focuses on managing the game state and logic, aligning with the Single Responsibility Principle. It delegates other concerns (e.g., UI handling) to other classes, maintaining a clear separation of responsibilities. | |
| Code Smells | Magic numbers | The use of magic numbers is minimized. Constants such as initial piece counts and board size are defined at the class level or in the utils module. This practice enhances code readability and ease of maintenance. | |
| Unnecessary global / class variable | There are no unnecessary global or class variables. Variables are defined within appropriate scopes, and the class maintains only those attributes necessary for its operation. | |
| Duplicate code | Minimal code duplication exists. Common functionalities are abstracted into helper methods or utility functions. This reduces redundancy and potential errors from maintaining multiple copies of similar code. | |
| Long methods | Some methods, particularly in the Game class (e.g., move\_piece, get\_possible\_moves), are relatively long and could be broken down into smaller helper methods. Refactoring these methods would improve readability and maintainability. | |
| Long parameter list | Methods generally have concise parameter lists. There are no methods with excessively long parameter lists that would suggest the need for refactoring or introducing data structures to encapsulate parameters. | |
| Over-complex expression | The code avoids over-complex expressions. Calculations and logic are broken down into understandable steps, enhancing clarity. | |
| Switch or if-then-else that needs to be replaced with polymorphism | The code uses if-else statements appropriately. There are no instances where a switch-case or extensive if-else chain could be replaced with polymorphism, given the language's capabilities and the application's requirements. | |
| Variable or method name whose intent is unclear | Variable and method names are descriptive and convey their purpose effectively. There are no ambiguous names that could confuse readers or maintainers of the code. | |
| Any similar methods in other classes? | No unnecessary duplication of methods across different classes was found. Methods are appropriately defined within their respective classes, and shared functionality is correctly abstracted. | |
| **Bugs** | **Buggy code snippet** | **What is the bug?** | **Why it is a bug?** |
| **In** place\_piece **method, lack of win condition check:** def place\_piece(self, position): ... | The game does not check for a win condition after placing the last piece in phase 1. | This could allow the game to continue even if a player has won after the placement phase, violating game rules. A win condition check should be added. |
| **In** get\_possible\_moves**, not checking for player mobility:** def get\_possible\_moves(self, player): ... | The method does not account for scenarios where a player has no possible moves but still has pieces on the board. | This oversight could prevent the game from recognizing a stalemate or loss condition when a player cannot move, leading to incorrect game progression. |
| In move\_piece, inadequate validation of flying phase moves: | Players may be able to 'fly' pieces before reaching the flying phase (when they have only three pieces). | This violates the game rules. Additional checks are needed to ensure that players can only move to any position (fly) during the appropriate game phase. |

1. **Meeting Minutes (only during this sprint)**

Report the minutes of all meetings, including, but not limited to: project/sprint planning meeting, stand-up meeting, backlog grooming, retrospective meeting, and pair programming session.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date** | **Time and Duration** | **Place** | **Participant Names** | **Purpose of the Meeting** | **Specific Action Items** |
| 10/25/2024 | 10:00 AM - 11:30 AM | Zoom | All team members | Sprint planning for Sprint #3 | Assign tasks related to AI implementation |
| 10/28/2024 | 2:00 PM - 3:00 PM | Zoom | All team members | Stand-up meeting to discuss progress | Address challenges with AI decision-making |
| 11/1/2024 | 4:00 PM - 5:30 PM | Zoom | All team members | Code review session | Evaluate code against checklists, identify improvements |
| 11/3/2024 | 9:00 AM - 11:00 AM | Zoom | All team members | Integration testing and bug fixing | Test AI functionality and replay features |
| 11/6/2024 | 9:00 AM - 11:00 AM | Zoom | All team members | Sprint retrospective | Reflect on sprint achievements and areas to improve |

1. **Buddy Ratings**

If you don’t feel comfortable to include your ratings in this report, you may email your ratings to the instructor or grader.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Rating giver* | *Rating receiver* | | | | |
|  | Full Name 1 | Full Name 2 | Full Name 3 | Full Name 4 |
| Full Name 1 | X | 1 | 1 | 1 |
| Full Name 2 | 1 | X | 1 | 1 |
| Full Name 3 | 1 | 1 | X | 1 |
| Full Name 4 | 1 | 1 | 1 | X |
|  | *Average* | 1 | 1 | 1 | 1 |

## VI. Summary

* **Mannava Jyothi Krishna** focused on developing the AI player, implementing the minimax algorithm with alpha-beta pruning (ai.py), and integrating the AI logic with the existing game mechanics. Participated actively in the code review process.
* **Mannava Vignesh** enhanced the graphical user interface (gui.py) to accommodate AI gameplay, ensuring smooth interaction between human and AI players. Implemented recording and replay features compatible with AI moves. Contributed to the code review.
* **Shaik Sumayya Fathima** expanded automated test cases to cover AI functionalities, ensuring the AI's moves and strategies behave as expected. Assisted with code review and identified areas for improvement.
* **Shreya Saraf** designed and optimized the AI's decision-making process using the minimax algorithm with alpha-beta pruning. Updated acceptance criteria to reflect the AI's capabilities. Led the team in conducting a thorough code review.