**Team Project Sprint #3**

**Team Report**

Instructions

Please read the instructions carefully. All members of your team should discuss the instructions together to ensure that everyone is on the same page.

**Objectives**

1. Update and complete the user stories and acceptance criteria of the target software that allows a human player to play against either a human or a computer opponent.
2. Complete the implementation of all user stories, including improvements on the previous sprint. Note that the computer opponent should make a reasonable attempt at beating the human player. For example, it can defeat the worst human player.
3. Conduct a code review exercise as a team (for at least one hour) and report the findings. Everyone should have read the following instructions before the exercise.

In this exercise, each team will apply code review practices to the Board class of the Mill game (and other classes if time permits) in their team project. One member should lead the code review and another member should take notes and document the findings. The review should be constructive. Do not criticize the developers who wrote the code under review. There is no need to test the program or make changes during the review exercise. The findings can be used to improve your project after the review.

In addition to looking for bugs, the review should check: (1) whether the entire project has followed the coding standard in a consistent manner, (2) whether the project has followed the design principles introduced in class, and (3) whether there are code smells that indicate the need for refactoring. The following checklists provide basic guidelines. You may add new items to each of the checklists.

Checklist #1: Coding Standards

* Are there any violations of naming conventions?
  + Packages, classes, methods, variables, constants
  + Production code vs test code
* Is the ordering convention of method arguments followed in each method?
* Are all the comments meaningful and valid?
  + Are the precondition and post-condition of each method documented?
* Is the same style used for all curly braces of code blocks?
* Consistent indentation?
* …

Checklist #2: Design Principles

* Does each class have a good abstraction and good class interface?
* Is the visibility of each variable, method, and class (private, protected, public, default) appropriate?
* Design by contract: for each public method, is Design by Contract followed? If so, is the specified precondition reasonable and available?
* Is the Open-Closed Principle violated?
* …

Checklist #3: Code Smells

* Is there any magic number or unnamed constant?
* Is there any unnecessary global or class variable?
* Is there duplicate code?
* Are there long methods?
* Does any method have a long parameter list?
* Is there any over-complex expression?
* Is there any switch or if-then-else statement that should be replaced with polymorphism?
* Is there any variable or method name whose intent is unclear?
* Are there similar methods in multiple classes?
* …

1. Review the software design of the final production code.

* Summarize the user-interface design, using a combination of screenshots and textual descriptions.
* Describe the software architecture of the final production code using class diagram(s). It should capture the main classes and their relationships.
* Describe the algorithms for the computer opponent to place a piece, remove a piece, move and fly a piece. The descriptions should be understandable without referring to the source code. For example, you may use pseudo code.
* Discuss how your code can be extended for the variants of Nine Men's Morris, including Six Men’s Morris and Twelve Men’s Morris. What classes and methods need to be changed, and how? How was the Open-Closed Principle applied (i.e., which functions or classes are open for extension, but closed for modification)?

**Deliverables and Grading Policy**

Please zip the project report, the demonstration video, and all source code into one .zip file before submitting.

Please don’t upload rar files and don’t submit links.

1. Project Report (**33 points**)

The project report should include the following sections:

* 1. Updated complete user stories using the template discussed in class. **(1 points)**

Provide a complete list of user stories and estimated/actual efforts for the target software that allows a human player to play against either a human or a computer opponent.

* 1. Updated complete acceptance criteria using the template discussed in class. **(4 points)**

Provide complete acceptance criteria for all the user stories.

* 1. Implementation tasks **(12 points)**

Describe the production code, automated test code or manual test cases for all the user stories. For each acceptance criterion of every user story, you need to implement at least one test (either test code or manual test case).

* 1. The design of the final production code (**12 points**)

User-interface design (**3 points**)

Software architecture (**3 points**)

Algorithm (**3 points**)

Extensibility (**3 points**)

* 1. Findings of the code review exercise **(2 points)**
  2. Minutes of ALL meetings, including, but not limited to: project/sprint planning meeting, stand-up meeting, backlog grooming, retrospective meeting, and pair programming (or development) session. **(2 points)**
  3. A table of buddy ratings. Individual members may email their buddy ratings to the instructor or teaching assistant.

Each team only needs to submit one report. For an individual member to receive the credit for this part of the project, the team’s project report must include explicit evidence of his/her contribution (e.g., his/her name is listed as a developer).

1. Demonstration (**5 points**)

Submit a 5-minute video in the .mp4 format. It should clearly demonstrate that:

1. your project has completed the implementation of all required features for the computer opponent.
2. A complete game won by the human player
3. A complete game won by the computer opponent without flying
4. A complete game won by the computer opponent with flying
5. for each acceptance criterion of each user story for the computer opponent, your project has implemented either an automated test method or performed an acceptance test manually.
6. your project has some unique features or enhancements (optional).

Grading of the demonstration is based on completion of the required functions (**3 points**), and overall presentation (**2 points**) using the following evaluation rubric:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Poor | Fair | Good | Very Good | Excellent |
| Was the demonstration logically organized |  |  |  |  |  |
| Were points made clearly and concisely |  |  |  |  |  |
| Were the grader or instructor’s questions answered satisfactorily |  |  |  |  |  |

3. Source Code

You will not receive credit if the source code is not submitted. Make sure your project report is consistent with the source code.

**Team Project Sprint #3**

Report Template

Team Name: Overworked and Understaffed

Team Members: Joshua Koni, Elizabeth Nastoff, Ruby Rios, Marley Symmonds

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 user, I want to be able to visualize the board so I can see the game. | Very High – Sprint 1 | 3 | 4 | Completed | Elizabeth Nastoff |
| 2 | Piece Visualization | As a user, I want to be able to place my pieces so I can see the game. | High – Sprint 1 | 3 | 5 | Completed | Joshua Koni |
| 3 | Turn-based Gameplay | As a user, I want the system to track whose turn it is so I can only play on my turn. | High –  Sprint 1 | 2 | 3 | Completed | Joshua Koni, Elizabeth Nastoff,  Marley Symmonds |
| 4 | Piece Differentiation | As a user, I want to know which color I am so I can know which pieces are mine. | High – Sprint 1 | 1 | 2 | Completed | Joshua Koni, Elizabeth Nastoff,  Marley Symmonds |
| 5 | Start Screen Functionality | As a user, I want to be able to select whether I want to play against a computer or another player, so I can choose who I play against. | High– Sprint 2 | 3 | 3 | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
| 6 | End Screen Functionality - Close | As a user, I want to be able to close the game so I can end the application when I am done. | High– Sprint 2 | 1 | 1 | Completes | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
| 7 | End Screen Functionality - Restart | As a user, I want to be able to reset the game so I can start a new game. | High – Sprint 2 | 1 | 2 | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
| 8 | Player vs Player Functionality | As a user, I want to be able to play against another player on the same device so I can play against my friends when we are using the same computer. | High– Sprint 2 | 4 | 6 | Completed | Elizabeth Nastoff, Marley Symmonds |
| 9 | “Placing Pieces” Phase Production | As a user, I want to be able to place my pieces anywhere on the board where there is not a piece currently during the Placing Pieces phase of the game, so that I can play the game. | High – Sprint 2 | 4 | 3 | Completed | Joshua Koni,  Elizabeth Nastoff, Marley Symmonds |
| 10 | “Moving Pieces” Phase Production | As a user, I want to be able to move my pieces one space on the board where there is not a piece currently during the Moving Pieces phase of the game, so that I can play the game. | High– Sprint 2 | 4 | 4 | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
| 11 | “Flying” Phase Production | As a user, when I have only three pieces left, I want to be able to move my pieces anywhere on the board where there is not a piece currently during the Flying phase of the game, so that I can play the game. | High – Sprint 2 | 4 | 3 | Completed | Elizabeth Nastoff, Joshua Koni, Marley Symmonds |
| 12 | Mill Functionality | As a user, when I have moved three pieces into a “mill” (all of the pieces are in consecutive spaces in a row or column), I want to be able to remove one of my opponents’ pieces, so that I can play the game. | High – Sprint 2 | 2 | 4 | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
| 13 | End Screen – Win/Loss | As a user, I want to know if I have won or lost so I can know if the game is done. | Medium – Sprint 2 | 1 | 2 | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
| 14 | Computer Selection | As a user, I want to be able to play against the computer so I can play solo. | Medium - Sprint 3 | 1 | 1 | Completed | Elizabeth Nastoff, Ruby Rios |
| 15 | Computer Functionality- “Placing Pieces” Phase Production | As a user, I want the computer to place and move pieces so I can play against it. | Medium – Sprint 3 | 4 | 4 | Completed | Elizabeth Nastoff, Ruby Rios |
| 16 | Computer Functionality- “Moving Pieces” Phase Production | As a user, I want the computer to place and move pieces so I can play against it. | Medium – Sprint 3 | 4 | 4 | Completed | Elizabeth Nastoff, Ruby Rios |
| 17 | Computer Functionality- “Flying” Phase Production | As a user, I want the computer to place and move pieces so I can play against it. | Medium – Sprint 3 | 4 | 4 | Completed | Elizabeth Nastoff, Ruby Rios |

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 | When an option is selected from the start screen for player opponent, then 16 lines and 24 circles will appear on the screen in the formation of the Nine Men’s Morris board. | Completed | Elizabeth Nastoff |
| 2 Piece Visualization | 2.1 | Given a board is displayed on the screen, when the screen is clicked on a circle where there are not any existing pieces and it is the “Placing Pieces” phase, then a piece is placed on the screen at that location. | Completed | Joshua Koni, Elizabeth Nastoff |
|  | 2.2 | Given a board is displayed on the screen, when the screen is clicked on a circle where there is an existing piece and it is the “Placing Pieces” phase, then a piece will not be placed on the screen at that location. | Completed | Joshua Koni, Elizabeth Nastoff |
|  | 2.3 | Given a board is displayed on the screen, when the screen is clicked in a location where a piece cannot go and it is the “Placing Pieces” phase, then a piece will not be placed on the screen at that location. | Completed | Joshua Koni, Elizabeth Nastoff |
| 3 Turn-based Gameplay | 3.1 | Given a game is started, when it is the user’s turn, then they will be able to make an action according to what phase the game is in. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 3.2 | Given a game is started, when it is the opponent’s turn, then they will be able to make an action according to what phase the game is in. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
| 4 Piece Differentiation | 4.1 | Given a board is displayed on the screen and the board is clicked in a place where there are not any existing pieces, when it is the user’s turn and it is Phase 1, then the piece placed is in the color of the user. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 4.2 | Given a board is displayed on the screen and the board is clicked in a place where there are not any existing pieces, when it is the opponent’s turn and it is Phase 1, then the piece placed is in the color of the opponent. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
| 5 Start Screen Functionality | 5.1 | When the application has been opened, then a screen will be displayed with buttons for choosing an opponent and for viewing the rules. | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
|  | 5.2 | Given the application has been opened, when the button for viewing the rules is clicked on, then a list of rules for the game will be displayed, as well as a button to return to the start screen. | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
| 6 End Screen Functionality – Close | 6.1 | Given that a game has ended and an end screen appears, when the user clicks the “close” button, the application will close. | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
| 7 End Screen Functionality – Restart | 7.1 | Given that a game has ended and an end screen appears, when the user clicks the “restart” button, the application will reset (the board will appear on the screen with no pieces on it, a new coin toss will be performed, the game will start in phase 1)  . | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
| 8 Player vs Player Functionality | 8.1 | Given that a game has started and the option to play against another player has been selected on the start screen, when the user is playing the game, then they will be able to play against another player on the same computer with full game functionality. | Completed | Elizabeth Nastoff, Marley Symmonds |
| 9 “Placing Pieces” Phase Production | 9.1 | Given that a game has started, when it is the user’s turn, less than nine pieces of the players have been placed on the board, and the place where the user clicks does not have any existing pieces on it, then the user will be able to play down a piece in this space. | Completed | Joshua Koni,  Marley Symmonds,  Elizabeth Nastoff |
|  | 9.2 | Given that a game has started, when it is the opponent’s turn, less than nine pieces of the opponents have been placed on the board, and the place where the opponent clicks does not have any existing pieces on it, then the opponent will be able to play down a piece in this space. | Completed | Joshua Koni,  Marley Symmonds,  Elizabeth Nastoff |
|  | 9.3 | Given that a game has started, when it is the user’s turn, less than nine pieces of the players have been placed on the board, and the place where the user clicks does have an existing piece on it, then the game will do nothing. | Completed | Joshua Koni,  Marley Symmonds,  Elizabeth Nastoff |
|  | 9.4 | Given that a game has started, when it is the opponent’s turn, less than nine pieces of the opponents have been placed on the board, and the place where the opponent clicks does have an existing piece on it, then the game will do nothing. | Completed | Joshua Koni,  Marley Symmonds,  Elizabeth Nastoff |
| 10 “Moving Pieces” Phase Production | 10.1 | Given that both players have placed all nine of their pieces, when the user clicks on a piece that is theirs and then click on a spot on the board that does not have any existing pieces on it, then the user will be able to move a piece to this space (the piece will disappear from its original position and reappear where the user has clicked). | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 10.2 | Given that both players have placed all nine of their pieces, when the opponent clicks on a piece that is theirs and then click on a spot on the board that does not have any existing pieces on it, then the opponent will be able to move a piece to this space (the piece will disappear from its original position and reappear where the user has clicked). | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 10.3 | Given that both players have placed all nine of their pieces, when the user clicks on a piece that is theirs and then click on a spot on the board that has an existing piece on it, then the game will do nothing. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 10.4 | Given that both players have placed all nine of their pieces, when the opponent clicks on a piece that is theirs and then click on a spot on the board that has an existing piece on it, then the game will do nothing. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 10.5 | Given that both players have placed all nine of their pieces, when the user clicks on a piece that is not theirs or a spot on the board that does not have a piece on it and then click on any spot on the board then the game will do nothing. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 10.6 | Given that both players have placed all nine of their pieces, when the opponent clicks on a piece that is not theirs or a spot on the board that does not have a piece on it and then click on any spot on the board then the game will do nothing. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
| 11 “Flying” Phase Production | 11.1 | Given that the user has only 3 pieces remaining on the board, when it is the user’s turn, the user clicks on one of their own pieces, and the place where the user clicks does not have any existing pieces on it, then the user will be able to move a piece to this space (the piece will disappear from its original position and reappear where the user has clicked). | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 11.2 | Given that the opponent has only 3 pieces remaining on the board, when it is the opponents turn, the opponent clicks on one of their own pieces, and the place where the opponent clicks does not have any existing pieces on it, then the opponent will be able to move a piece to this space (the piece will disappear from its original position and reappear where the user has clicked). | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 11.3 | Given that the user has only 3 pieces remaining on the board, when it is the user’s turn, the user clicks on one of their own pieces, and the place where the user clicks has an existing piece on it, then the game will do nothing. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 11.4 | Given that the opponent has only 3 pieces remaining on the board, when it is the opponent’s turn, the opponent clicks on one of their own pieces, and the place where the opponent clicks has an existing piece on it, then the game will do nothing. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 11.5 | Given that the user has only 3 pieces remaining on the board, when it is the user’s turn, the user clicks on a piece that is not their own or a space on the board that does not have any pieces and any other spot on the board, then the game will do nothing. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 11.6 | Given that the opponent has only 3 pieces remaining on the board, when it is the opponent’s turn, the opponent clicks on a piece that is not their own or a space on the board that does not have any pieces and any other spot on the board, then the game will do nothing. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
| 12 Mill Functionality | 12.1 | Given that a game is going, when 3 pieces of the users are aligned consecutively vertically or horizontally, then the player has formed and mill and gets to remove one of their opponent’s pieces from the board. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
|  | 12.2 | Given that a game is going, when 3 pieces of the opponents are aligned consecutively vertically or horizontally, then the opponent has formed and mill and gets to remove one of the user’s pieces from the board. | Completed | Joshua Koni, Elizabeth Nastoff, Marley Symmonds |
| 13 End Screen – Win/Loss | 13.1 | Given that a game has ended, the user has won, and the game was player vs player, when the end screen is displayed, then it will say that the user has won. | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
|  | 13.2 | Given that a game has ended, the opponent has won, and the game was player vs player, when the end screen is displayed, then it will say that the opponent has won. | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
|  | 13.3 | Given that a game has ended, the user has won, and the game was player vs computer, when the end screen is displayed, then it will say that the user has won. | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
|  | 13.4 | Given that a game has ended, the opponent has won, and the game was player vs computer, when the end screen is displayed, then it will say that the user has lost. | Completed | Joshua Koni, Elizabeth Nastoff, Ruby Rios |
| 14 Computer Selection | 14.1 | Given that a game has started and the start screen is displayed, when the user selects to play against a computer, then the user will begin a game against a computer. | Completed | Elizabeth Nastoff, Ruby Rios |
| 15 Computer Functionality- “Placing Pieces” Phase Production | 15.1 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is given the option to place their first piece, the computer will place their piece on a random place. | Completed | Elizabeth Nastoff, Ruby Rios |
|  | 15.2 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is still in the placing pieces phase of gameplay, when given an option to place a piece, if the ability to place a piece in a mill is available, the computer will complete a mill. | Completed | Elizabeth Nastoff, Ruby Rios |
|  | 15.3 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is still in the placing pieces phase of gameplay, when given an option to place a piece, if the user has 2 adjacent pieces next to each other, the computer will block the user from completing a mill. | Completed | Elizabeth Nastoff, Ruby Rios |
|  | 15.4 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is still in the placing pieces phase of gameplay, when given an option to place a piece, if the ability for the computer to complete a mill is not available for either player, if there is a place where the computer can play a piece adjacent to where it’s previously placed a piece, the computer will place the piece there. | Completed | Elizabeth Nastoff, Ruby Rios |
|  | 15.5 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is still in the placing pieces phase of gameplay, when given an option to place a piece, if the ability for the computer to complete a mill or form a new mill is not available for either player and there is no adjacent spots on the board for the computer to place its piece, the computer will place a piece in one of the remaining open spots on the board. | Completed | Elizabeth Nastoff, Ruby Rios |
| 16 Computer Functionality- “Moving Pieces” Phase Production | 16.1 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is in the moving phase, when given an option to move a piece, if the ability to move a piece into a mill is available, the computer will complete a mill. | Completed | Elizabeth Nastoff, Ruby Rios |
|  | 16.2 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is in the moving phase, when given an option to move a piece, if the user has 2 pieces adjacent to each other, and the computer can move a piece in a place that will prevent a mill from being formed, the computer will block a mill. | Completed | Elizabeth Nastoff, Ruby Rios |
|  | 16.3 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is in the moving phase, when given an option to move a piece, if the ability to move a piece in mill or block a user’s mill is not available, if the ability is open for the computer to move a piece adjacent to another piece that it already has, it will do so. | Completed | Elizabeth Nastoff, Ruby Rios |
|  | 16.4 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is in the moving phase, when given an option to move a piece, f the ability to move a piece in mill, block a user’s mill, or to move a piece adjacent to an existing piece it already has is not available, the computer will move a piece at random. | Completed | Elizabeth Nastoff, Ruby Rios |
| 17 Computer Functionality- “Flying” Phase Production | 17.1 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is in the flying phase, when given an option to move a piece, if the ability to move a piece into a mill is available, the computer will complete a mill. | Completed | Elizabeth Nastoff, Ruby Rios |
|  | 17.2 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is in the flying phase, when given an option to move a piece, if the user has 2 pieces adjacent to each other, the computer will block a mill. | Completed | Elizabeth Nastoff, Ruby Rios |
|  | 17.3 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is in the flying phase, when given an option to move a piece, if the ability to move a piece in mill or block a user’s mill, if the ability is open for the computer to move a piece adjacent to another piece that it already has, it will do so. | Completed | Elizabeth Nastoff, Ruby Rios |
|  | 17.4 | Given that a game has started and the option to play against a computer has been selected on the start screen, when the computer is in the flying phase, when given an option to move a piece, f the ability to move a piece in mill, block a user’s mill, or to move a piece adjacent to an existing piece it already has is not available, the computer will move a piece at random. | Completed | Elizabeth Nastoff, Ruby Rios |

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 | Board, Main | drawBoard() | Elizabeth Nastoff | Completed |  |
| 2 Piece Visualization | 2.1 | Board | drawBoard(), handleMouseClick(s,t) | Joshua Koni, Elizabeth Nastoff | Completed |  |
|  | 2.2 | Board | drawBoard(), handleMouseClick(s,t) | Joshua Koni, Elizabeth Nastoff | Completed |  |
|  | 2.3 | Board | drawBoard(), handleMouseClick(s,t) | Joshua Koni, Elizabeth Nastoff | Completed |  |
| 3 Turn-based Gameplay | 3.1 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck() | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 3.2 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck() | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
| 4 Piece Differentiation | 4.1 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck() | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 4.2 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck() | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed | . |
| 5 Start Screen Functionality | 5.1 | Main, Menu | drawMenu(screen) | Joshua Koni, Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 5.2 | Main, Menu | drawMenu(screen),  drawInstruction(instruction, height, screen) | Joshua Koni, Elizabeth Nastoff, Ruby Rios | Completed |  |
| 6 End Screen Functionality – Close | 6.1 | Main, Menu | drawMenu(screen),  handleMenuClick(screen, mousePosition) | Joshua Koni, Elizabeth Nastoff, Ruby Rios | Completed |  |
| 7 End Screen Functionality – Restart | 7.1 | Main, Menu, Board | drawMenu(screen),  handleMenuClick(screen, mousePosition),  clearBoard() | Joshua Koni, Elizabeth Nastoff, Ruby Rios | Completed |  |
| 8 Player vs Player Functionality | 8.1 | Board, Menu | drawBoard(),  handleMenuClick(screen, mousePosition),  handleMouseClick(s,t),  turnCheck() | Elizabeth Nastoff, Marley Symmonds | Completed |  |
| 9 “Placing Pieces” Phase Production | 9.1 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  phaseOne(location),  isNotTaken(num) | Joshua Koni,  Marley Symmonds,  Elizabeth Nastoff | Completed |  |
|  | 9.2 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  phaseOne(location),  isNotTaken(num) | Joshua Koni,  Marley Symmonds,  Elizabeth Nastoff | Completed |  |
|  | 9.3 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  phaseOne(location),  isNotTaken(num) | Joshua Koni,  Marley Symmonds,  Elizabeth Nastoff | Completed |  |
|  | 9.4 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  phaseOne(location),  isNotTaken(num) | Joshua Koni,  Marley Symmonds,  Elizabeth Nastoff | Completed |  |
| 10 “Moving Pieces” Phase Production | 10.1 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isAdj(current\_spot, potential\_spot),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 10.2 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isAdj(current\_spot, potential\_spot),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 10.3 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isAdj(current\_spot, potential\_spot),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 10.4 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isAdj(current\_spot, potential\_spot),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 10.5 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isAdj(current\_spot, potential\_spot),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 10.6 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isAdj(current\_spot, potential\_spot),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
| 11 “Flying” Phase Production | 11.1 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 11.2 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 11.3 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 11.4 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 11.5 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 11.6 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isNotTaken(num),  clickOne(pieceLocation),  clickTwo(pieceLocation) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
| 12 Mill Functionality | 12.1 | Board | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isMill(placed\_pieces, new\_piece) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
|  | 12.2 | Game\_Class | drawBoard(), handleMouseClick(s,t),  turnCheck(),  isMill(placed\_pieces, new\_piece) | Joshua Koni, Elizabeth Nastoff, Marley Symmonds | Completed |  |
| 13 End Screen – Win/Loss | 13.1 | Main, Menu, Board | handleMouseClick(s,t),  hasWon(),  drawMenu(screen) | Joshua Koni, Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 13.2 | Main, Menu, Board | handleMouseClick(s,t),  hasWon(),  drawMenu(screen) | Joshua Koni, Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 13.3 | Main, Menu, Board | handleMouseClick(s,t),  hasWon(),  drawMenu(screen) | Joshua Koni, Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 13.4 | Main, Menu, Board | handleMouseClick(s,t),  hasWon(),  drawMenu(screen) | Joshua Koni, Elizabeth Nastoff, Ruby Rios | Completed |  |
| 14 Computer Selection | 14.1 | Menu, Board, AIHeuristic | drawMenu(screen), handleMouseClick(s,t), | Elizabeth Nastoff, Ruby Rios | Completed |  |
| 15 Computer Functionality- “Placing Pieces” Phase Production | 15.1 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  phaseOne(location),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed | Acceptance Criteria for User Story 15 have been updated since Sprint 2. |
|  | 15.2 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  phaseOne(location),  AIPhase1(takenSpots, whitePlaced, blackPlaced),  makeOrPreventMill(openSpots, blackPlaced, whitePlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 15.3 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  phaseOne(location),  AIPhase1(takenSpots, whitePlaced, blackPlaced),  makeOrPreventMill(openSpots, blackPlaced, whitePlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 15.4 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  phaseOne(location),  AIPhase1(takenSpots, whitePlaced, blackPlaced),  potentialMill(openSpots, blackPlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 15.5 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  phaseOne(location),  AIPhase1(takenSpots, whitePlaced, blackPlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed |  |
| 16 Computer Functionality- “Moving Pieces” Phase Production | 16.1 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  AIphase2(takenSpots, whitePlaced, blackPlaced),  phase2Mill(moves, blackPlaced, whitePlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed | Acceptance Criteria for User Story 16 have been updated since Sprint 2. |
|  | 16.2 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  AIphase2(takenSpots, whitePlaced, blackPlaced),  phase2Mill(moves, blackPlaced, whitePlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 16.3 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  AIphase2(takenSpots, whitePlaced, blackPlaced),  phase2CloseMill(moves, openSpots, blackPlaced, whitePlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 16.4 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  AIphase2(takenSpots, whitePlaced, blackPlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed |  |
| 17 Computer Functionality- “Flying” Phase Production | 17.1 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  AIFlying(takenSpots, whitePlaced, blackPlaced),  makeOrPreventMill(openSpots, blackPlaced, whitePlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed | Acceptance Criteria for User Story 17 have been updated since Sprint 2. |
|  | 17.2 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  AIFlying(takenSpots, whitePlaced, blackPlaced),  makeOrPreventMill(openSpots, blackPlaced, whitePlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 17.3 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  AIFlying(takenSpots, whitePlaced, blackPlaced),  potentialMill(openSpots, blackPlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed |  |
|  | 17.4 | Board, AIHeuristic | drawBoard(), handleMouseClick(s,t),  AIFlying(takenSpots, whitePlaced, blackPlaced),  pickByRandom(options) | Elizabeth Nastoff, Ruby Rios | Completed |  |

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)** |
| 2 Piece Visualization | 2.1 | PieceLocationTest(unittest.TestCase) | Piece\_Location(x, y. spots\_List),  test\_Piece\_Location\_Test() | Piece\_Location() returns true if the x and y values input are in the range of an item in spots\_List. test\_Piece\_Location\_Test() checks that this works for every coordinate on the board  The expected output is null. | Working | Joshua Koni, Ruby Rios |
|  | 2.3 | PieceLocationTest(unittest.TestCase) | Piece\_Location(x, y. spots\_List),  test\_Piece\_Location\_Test() | Piece\_Location() returns true if the x and y values input are in the range of an item in spots\_List. test\_Piece\_Location\_Test() checks that this works for every coordinate on the board  The expected output is null. | Working | Joshua Koni, Ruby Rios |
| 3 Turn-based Gameplay | 3.1 | CheckTurnTest(unittest.TestCase) | def CheckTurnTest (turn) | def CheckTurnTest (turn) gets which turn the game is at, and checks that the color of the piece and the player putting down that piece are correct.  The expected output is null. | Working | Joshua Koni, Ruby Rios |
|  | 3.2 | CheckTurnTest(unittest.TestCase) | def CheckTurnTest (turn) | def CheckTurnTest (turn) gets which turn the game is at, and checks that the color of the piece and the player putting down that piece are correct.  The expected output is null. | Working | Joshua Koni, Ruby Rios |
| 4 Piece Differentiation | 4.1 | CheckTurnTest(unittest.TestCase) | def CheckTurnTest (turn) | def CheckTurnTest (turn) gets which turn the game is at, and checks that the color of the piece and the player putting down that piece are correct.  The expected output is null. | Working | Joshua Koni, Ruby Rios |
| 8 Player vs Player Functionality | 8.1 | CheckTurnTest(unittest.TestCase) | def CheckTurnTest (turn) | def CheckTurnTest (turn) gets which turn the game is at, and checks that the color of the piece and the player putting down that piece are correct.  The expected output is null. | Working | Joshua Koni, Ruby Rios |
| 9 “Placing Pieces” Phase Production | 9.1 | PieceLocationTest(unittest.TestCase) | Piece\_Location(x, y. spots\_Dict),  Piece\_Location\_Test(spots\_Dict) | Piece\_Location() returns true if the x and y values input are in the range of an item in spots\_List. test\_Piece\_Location\_Test() checks that this works for every coordinate on the board  The expected output is null. | Working | Joshua Koni, Ruby Rios |
|  | 9.2 | PieceLocationTest(unittest.TestCase) | Piece\_Location(x, y. spots\_List),  test\_Piece\_Location\_Test() | Piece\_Location() returns true if the x and y values input are in the range of an item in spots\_List. test\_Piece\_Location\_Test() checks that this works for every coordinate on the board  The expected output is null. | Working | Joshua Koni, Ruby Rios |
|  | 9.3 | PieceLocationTest(unittest.TestCase) | Piece\_Location(x, y. spots\_List),  test\_Piece\_Location\_Test() | Piece\_Location() returns true if the x and y values input are in the range of an item in spots\_List. test\_Piece\_Location\_Test() checks that this works for every coordinate on the board  The expected output is null. | Working | Joshua Koni, Ruby Rios |
|  | 9.4 | PieceLocationTest(unittest.TestCase) | Piece\_Location(x, y. spots\_List),  test\_Piece\_Location\_Test() | Piece\_Location() returns true if the x and y values input are in the range of an item in spots\_List. test\_Piece\_Location\_Test() checks that this works for every coordinate on the board  The expected output is null. | Working | Joshua Koni, Ruby Rios |
| 12 Mill Functionality | 12.1 | CheckMillTest(unittest.TestCase) | def CheckMillTest() | test\_CheckMillTest() returns true if the pieces provided produces a mill and false if the pieces provided do not form a mill. It compares these to the isMill function.  The expected output is all combinations of mills, and whether they are mills or not. | Working | Marley Symmonds |
|  | 12.2 | CheckMillTest(unittest.TestCase) | def CheckMillTest() | test\_CheckMillTest() returns true if the pieces provided produces a mill and false if the pieces provided do not form a mill. It compares these to the isMill function.  The expected output is all combinations of mills, and whether they are mills or not. | Working | Marley Symmonds |
| 15 Computer Functionality- “Placing Pieces” Phase Production | 15.1 | class ComputerTest() | def testPlaceRandomFirstPiece() | def testPlaceRandomFirstPiece() simulates the first opportunity the computer has to place a piece, and ensures that it places a piece randomly. | Working | Ruby Rios |
|  | 15.2 | class ComputerTest() | def testMakeMillPhase1Test() | def testMakeMillPhase1Test() simulates where there is a mill available for the computer to make, and makes sure that the computer places its piece to complete the mill. | Working | Ruby Rios |
|  | 15.3 | class ComputerTest() | def testBlockMillPhase1Test() | def testBlockMillPhase1Test() simulates where there is a mill available for the computer to block, and makes sure that the computer places its piece to block the mill. | Working | Ruby Rios |
|  | 15.4 | class ComputerTest() | def testAdjPhase1Test() | def testAdjPhase1Test()  simulates where there is not a place for the computer to make or block a mill, and makes sure that when there is the option to do so, the computer places its piece adjacent to a piece it has previously placed. | Working | Ruby Rios |
|  | 15.5 | class ComputerTest() | def testRandomPhase1Test() | def testRandomPhase1Test() simulates that when there is not an opportunity to form/block a mill or place a piece adjacent to an existing piece, that the computer will place a piece at random. | Working | Ruby Rios |
| 16 Computer Functionality- “Moving Pieces” Phase Production | 16.1 | class ComputerTest() | def testMakeMillPhase2Test() | def testMakeMillPhase2Test() simulates where there is a mill available for the computer to make, and makes sure that the computer moves its piece to make the mill. | Working | Ruby Rios |
|  | 16.2 | class ComputerTest() | def testBlockMillPhase2Test() | def testBlockMillPhase2Test() simulates where there is a mill available for the computer to block, and makes sure that the computer moves its piece to block the mill. | Working | Ruby Rios |
|  | 16.3 | class ComputerTest() | def testAdjPhase2Test() | def testAdjPhase2Test()  simulates where there is not a place for the computer to make or block a mill, and makes sure that when there is the option to do so, the computer moves its piece adjacent to a piece it has previously placed. | Working | Ruby Rios |
|  | 16.4 | class ComputerTest() | def testRandomPhase2Test() | def testRandomPhase2Test() simulates that when there is not an opportunity to form/block a mill or place a piece adjacent to an existing piece, that the computer will move a piece at random. | Working | Ruby Rios |
| 17 Computer Functionality- “Flying” Phase Production | 17.1 | class ComputerTest() | def testMakeMillPhase3Test() | def testMakeMillPhase3Test() simulates where there is a mill available for the computer to make, and makes sure that the computer moves its piece to make the mill. | Working | Ruby Rios |
|  | 17.2 | class ComputerTest() | def testBlockMillPhase3Test() | def testBlockMillPhase3Test() simulates where there is a mill available for the computer to block, and makes sure that the computer moves its piece to block the mill. | Working | Ruby Rios |
|  | 17.3 | class ComputerTest() | def testAdjPhase3Test() | def testAdjPhase3Test()  simulates where there is not a place for the computer to make or block a mill, and makes sure that when there is the option to do so, the computer moves its piece adjacent to a piece it has previously placed. | Working | Ruby Rios |
|  | 17.4 | class ComputerTest() | def testRandomPhase3Test() | def testRandomPhase3Test() simulates that when there is not an opportunity to form/block a mill or place a piece adjacent to an existing piece, that the computer will move a piece at random. | Working | Ruby Rios |

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 | print(“Board has been drawn”)  return True | “Board has been drawn” prints whenever a change to the board is made. | Working | More tests done on board functionality in other tests. | Joshua Koni, Ruby Rios |
| 2 Piece Visualization | 2.2 | print("Piece has already been placed!")  Input cases for all 24 circles checked | For each circle, on first click:  x, y, i[0], i[1]  Upon repeat click:  x, y, i[0], i[1]  Piece has already been placed! | Working | Test worked when first implemented. Currently, with the beginnings of a piece counter, the test does not work, as the counter only allows 9 pieces to be placed down by each player. | Ruby Rios |
| 5 Start Screen Functionality | 5.1 | print(“Button has been pressed. Start Game.”) | “Button has been pressed. Start Game.”, before the board is drawn | Working |  | Ruby Rios |
|  | 5.2 | print("Button has been pressed. Instructions Menu Printed.") | “Button has been pressed. Instructions Menu Printed.", before the Instructions Menu is drawn | Working |  | Ruby Rios |
| 6 End Screen Functionality – Close | 6.1 | print(“Button has been pressed. Close Program.”) | “Button has been pressed. Close Program.”, before the program is closed. | Working |  | Ruby Rios |
| 7 End Screen Functionality – Restart | 7.1 | print(“Button has been pressed. Restart Game.”) | “Button has been pressed. Restart Game.”, before the start screen is redrawn. | Working |  | Ruby Rios |
| 9 “Placing Pieces” Phase Production | 9.1 | print("Phase 1 in progress, Valid piece placed.") | “Phase 1 in progress, Valid piece placed.", after a piece is placed at a valid location on the board | Working |  | Ruby Rios |
|  | 9.2 | print("Phase 1 in progress, Valid piece placed.") | “Phase 1 in progress, Valid piece placed.", after a piece is placed at a valid location on the board | Working |  | Ruby Rios |
|  | 9.3 | print("Phase 1 in progress, Invalid piece not placed.") | "Phase 21in progress, Invalid piece not placed." after a piece is not placed at an invalid location on the board | Working |  | Ruby Rios |
|  | 9.4 | print("Phase 1 in progress, Invalid piece not placed.") | "Phase 1 in progress, Invalid piece not placed." after a piece is not placed at an invalid location on the board | Working |  | Ruby Rios |
| 10 “Moving Pieces” Phase Production | 10.1 | print("Phase 2 in progress, Valid piece placed.") | “Phase 2 in progress, Valid piece placed.", after a piece is moved to a valid location on the board | Working |  | Ruby Rios |
|  | 10.2 | print("Phase 2 in progress, Valid piece placed.") | “Phase 2 in progress, Valid piece placed.", after a piece is moved to a valid location on the board | Working |  | Ruby Rios |
|  | 10.3 | print("Phase 2 in progress, Invalid piece not placed.") | "Phase 2 in progress, Invalid piece not placed." after a piece is not moved to an invalid location on the board | Working |  | Ruby Rios |
|  | 10.4 | print("Phase 2 in progress, Invalid piece not placed.") | "Phase 2 in progress, Invalid piece not placed." after a piece is not moved to an invalid location on the board | Working |  | Ruby Rios |
|  | 10.5 | print("Phase 2 in progress, Invalid piece not placed.") | "Phase 2 in progress, Invalid piece not placed." after a piece is not moved to an invalid location on the board | Working |  | Ruby Rios |
|  | 10.6 | print("Phase 2 in progress, Invalid piece not placed.") | "Phase 2 in progress, Invalid piece not placed." after a piece is not moved to an invalid location on the board | Working |  | Ruby Rios |
| 11 “Flying” Phase Production | 11.1 | print("Phase 3 in progress, Valid white piece placed.") | "Phase 3 in progress, Valid white piece placed." after a piece is moved to a valid user location on the board | Working |  | Ruby Rios |
|  | 11.2 | print("Phase 3 in progress, Valid black piece placed.") | "Phase 3 in progress, Valid white piece placed." after a piece is moved to a valid opponent location on the board | Working |  | Ruby Rios |
|  | 11.3 | print("Phase 3 in progress, Invalid white piece not placed.") | "Phase 3 in progress, Invalid white piece not placed." after a piece is not moved to an invalid user location on the board | Working |  | Ruby Rios |
|  | 11.4 | print("Phase 3 in progress, Invalid black piece not placed.") | "Phase 3 in progress, Invalid black piece not placed." after a piece is not moved to an invalid opponent location on the board | Working |  | Ruby Rios |
|  | 11.5 | print("Phase 3 in progress, Invalid white piece not placed.") | "Phase 3 in progress, Invalid white piece not placed." after a piece is not moved to an invalid user location on the board | Working |  | Ruby Rios |
|  | 11.6 | print("Phase 3 in progress, Invalid black piece not placed.") | "Phase 3 in progress, Invalid black piece not placed." after a piece is not moved to an invalid opponent location on the board | Working |  | Ruby Rios |
| 13 End Screen – Win/Loss | 13.1 | print("White Win Menu printed") | “White Win Menu printed", before the win screen for the white player is drawn. | Working |  | Ruby Rios |
|  | 13.2 | print("Black Win Menu printed") | “Black Win Menu printed", before the win screen for the black player is drawn. | Working |  | Ruby Rios |
| 14 Computer Selection | 14.1 | print(“Button has been pressed. Start Game against AI.”) | “Button has been pressed. Start Game against AI”, before the board is drawn | Working |  | Ruby Rios |

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

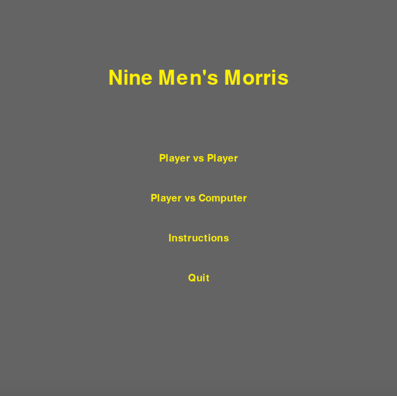
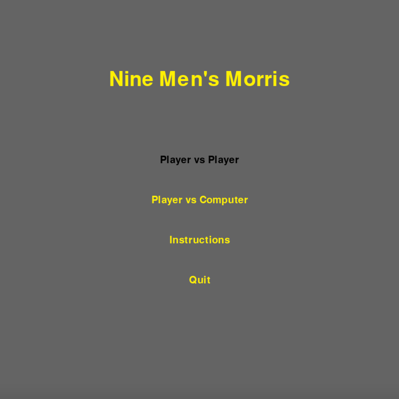
List the names of the team members who contributed to this section.

Summarize the user-interface design, using a combination of screenshots and textual descriptions.

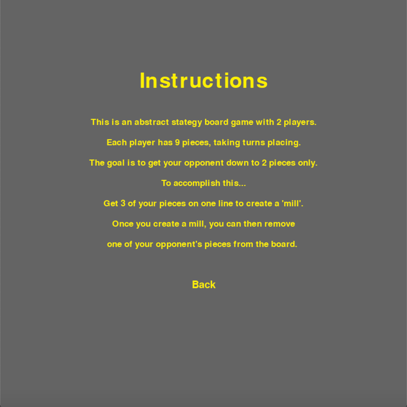
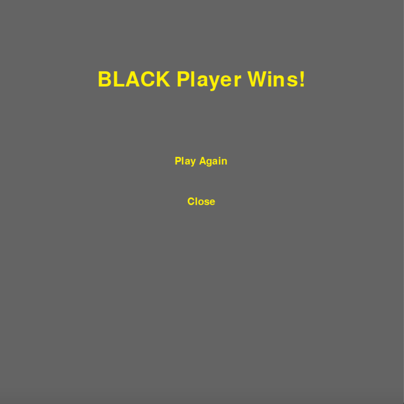
Team Members: Ruby Rios

We worked to create a functional user experience through use of visual indicators to the player. With our menu, we implemented functionality that changes text color on hover, so that the user had an indicator to inform them about where they can click. We also have instructions for the game as a screen within the menu, as to help the player get an understanding of how to play the game properly. Within the game itself, we implemented instruction text on the top of the screen to let the players know which player’s turn it is and what action they should take. In addition, in the moving pieces phase of the game, to help the player with moving their piece properly, we implemented a feature to help indicate which piece is currently selected. If they decide that they would like to change the piece they’re moving, they can click that same piece, and the yellow indicator will disappear, indicating that the piece has been deselected. On the board itself, there are small round circles to help the user determine where they can place pieces. We tried to choose colors for text and board objects that would help make visual tasks like reading and differentiation of players easier.

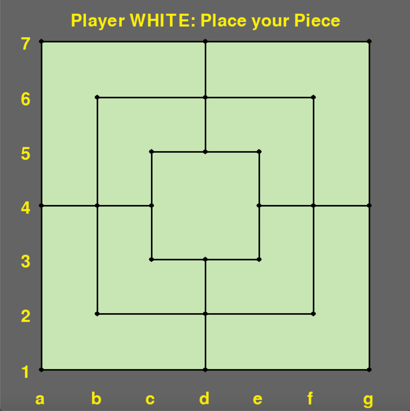
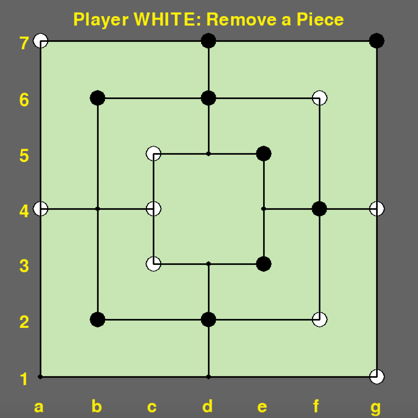
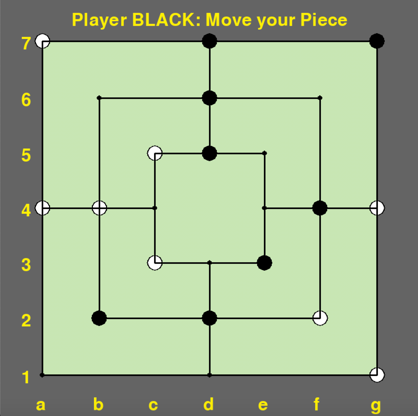
**User Interface Design – Screenshots**

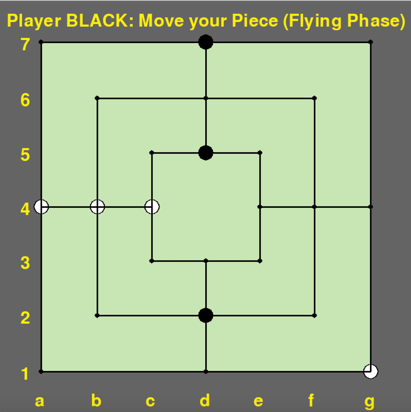
 

Main Menu Main Menu (with hover over Player vs. Player)

Instructions Screen Game Over/Winner Screen



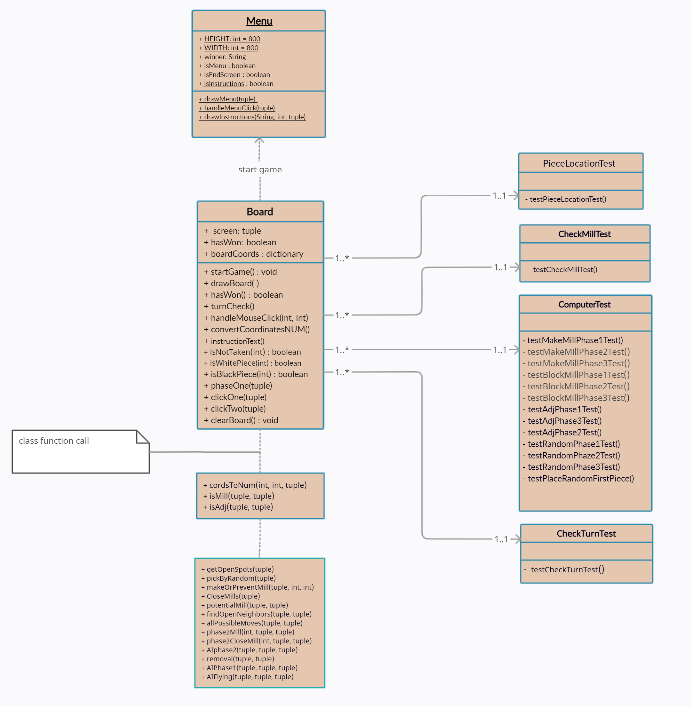
During the game, varying instructions given at the top of the screen depending on user/phase.

1. **Software Architecture**

List the names of the team members who contributed to this section.

Provide a class diagram that captures the main classes and their relationships in your final program.

Team Members: Joshua Koni



1. **Algorithm Design**

List the names of the team members who contributed to this section.

Describe the algorithm design of the computer opponent (e.g., using pseudo code). The description should be understandable without referring to the source code.

Team Members: Elizabeth Nastoff, Ruby Rios

The algorithm for the computer opponent has similar logic for each phase, even if the way it behaves will depend on the actions it’s permitted by the phase. To ensure that the computer takes its proper action, each of the phases have separate functions, as well as a separate function for removing pieces. In these functions, the first thing it will check is whether there is the ability for the computer to make a mill or block a mill, either by placing a piece, moving a piece, or flying a piece, depending on the phase. If it has an option to do both, it will make this choice at random. If it does not have the option to make or block a mill, if the computer has the ability to move/place a piece next to another piece it has control over, it will make that action. If the computer cannot do any of the above actions, it will move or place a piece at random. If the computer creates a mill and can remove a piece, it will first look to see if it can remove a piece from a mill of the user. If it cannot do this, then it will see if it can remove a piece of the user’s which has an adjacent user piece next to it. If it cannot do either of the above, it will remove a piece at random.

For the first piece that the computer places on the board, it will place this piece at random.

1. **Extensibility**

List the names of the team members who contributed to this section.

Discuss how your code can be extended for the variants of Nine Men's Morris, including Six Men’s Morris and Twelve Men’s Morris. What classes and methods need to be changed, and how? How was the Open-Closed Principle applied (i.e., which functions or classes are open for extension, but closed for modification)?

Team Members: Ruby Rios

For variants of Nine Men’s Morris, the Board class would need to changed, as well as some of the functions in the GameLogic and AIHeuristic files. In the Game Logic file, the isMill() and isAdj() functions would need to be updated to work with the new board piece numbers. In the AIHeruistic file, getOpentSpots() would need to be updated to support the new number of pieces and closeMills() would need to be updated with the new mills. In the Board class file, the drawBoard() and phaseOne() functions would need to be updated, as the drawBoard() would need to draw a different board to match the variation of Nine Men’s Morris, and phaseOne() would need to be updated, as it uses the turn number to determine how many pieces have been placed on the board, and if the piece number changes, the corresponding turn number should change. However, with the exception of phaseOne(), all of these functions refer to the board makeup itself. The functions for gameplay would still work, all that would need to be updated are the few functions that refer to the different placement of pieces on the board, and beyond drawBoard(), which would need different math to refer to where to draw the boxes on the board and what percentage of the screen those boxes should make up, the rest of these updates would be easy to make, as it would just be making basic updates to piece location numbers instead of coordinate numbers. We tried to apply the Open-Close principle through adding in functions involving adjacent and mills in separate files outside of the Board class.

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

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

Participant names: Joshua Koni, Elizabeth Nastoff, Ruby Rios, Marley Symmonds

Class that was reviewed: We reviewed all of the code within the project from Sprint 2. This is excluding the new test code and Computer class written for this Sprint.

|  |  |  |  |
| --- | --- | --- | --- |
| **Checklist** | **Checklist Item** | **Findings** | |
| Coding Standards | Naming conventions | We found that while most variables, function names, and classes were properly written in camel case, a few of these items were written in snake case. These variables were fixed accordingly. | |
| Ordering convention of method arguments | The ordering convention of method arguments was consistent with all methods we looked at. | |
| Meaningful and valid comments | Comments were scarce throughout the code, and the pre-conditions and post-conditions of functions were not given. These issues were fixed accordingly. | |
| Consistent style of code blocks | We did not find issues in regards to the consistent style of code blocks. Curly brace style was consistent within our code. | |
| Consistent indentation | We did not find issues in regards to consistent indentation. Indentation seemed to be consistent throughout our code. | |
| … | No additional notes were added beyond these categories. | |
| Design Principles | Good class abstraction and interface | While there might be some more class abstraction and class interface that could be done, | |
| Appropriate visibility of each variable, method, and class | We found that we had not previously considered visibility of variables, methods, and classes in the design of our code, and, as such, have many places in our code where visibility could be improved. However, due to our team having limited experience with determining what constitutes as proper visibility, we have decided to work on this in a later iteration, as it may pose a risk to our ability to complete the sprint. | |
| Any violation of the command-query separation principle | There were a few functions that violated the command-query separation principle, with the most notable of such being the handleMouseClick function in the Board class. | |
| Design by contract (pre/post-conditions) | We had not previously been using design by contract in our code. We added pre/post-conditions after this code review. While we currently feel like all of our pre and post conditions are reasonable and available, this could be disputed with another code review, as all of these conditions were created recently. | |
| Is the Open-Closed Principle violated? | While there were some examples of where the open-closed principle might be better applied in our code, we found our code for the most part displayed the open-closed principle. | |
| Is the Single Responsibility Principle violated? | We believe that the single responsibility principle is followed, as each class and method seem well suited to handling it’s assigned goal. | |
| Code Smells | Magic numbers | Our constants were all added into a separate globals class, which we believe is helpful for their visibility. We did not find any unnamed constants or magic numbers within our code. | |
| Unnecessary global / class variable | We did find some unused global variables in our code in regards to colors that we did not use. We have gotten rid of those to clean up our code. | |
| Duplicate code | We did not find any duplicate code within our project. | |
| Long methods | We did recognize that there were some longer methods in our code, particularly in our Board and Menu classes. However, for this Sprint, we recognized that while we have worked to break shorten these methods, doing more to shorten these methods would pose a risk to our ability to complete this sprint on time, and have decided to work on this in a later iteration. | |
| Long parameter list | We did not find any long parameter lists within our code. | |
| Over-complex expression | We did recognize that some expressions in our code might be a little harder to read and understand, but simplifying these would be difficult. We added comments in order to better clarify these sections. | |
| Switch or if-then-else that needs to be replaced with polymorphism | We did recognize that there were some are some longer if-then-else expressions in our code that should be replaced with polymorphism. However, for this Sprint, we were not certain on our ability to implement polymorphism successfully in these cases, and have decided to work on this in a later iteration. | |
| Variable or method name whose intent is unclear | We did not find unclear method or variable names within our code. | |
| Any similar methods in other classes? | We did not find duplicate methods amongst the classes. | |
| … | No additional notes were added beyond these categories. | |
| **Bugs** | **Buggy code snippet** | **What is the bug?** | **Why it is a bug?** |
|  |  |  |
|  |  |  |
|  |  |  |

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** |
| 11/24 | 1:00-4:00 PM  3 hours | Microsoft Teams | Joshua Koni,  Elizabeth Nastoff, Ruby Rios, Marley Symmonds | Retrospective meeting, Project/Sprint Planning, and Backlog Grooming | Specific tasks were assigned to each member to get started on the next sprint. In addition, each member of the team is to go through the code for the upcoming code review. |
| 11/29 | 2:00-2:30 PM  30 minutes | Discord | Elizabeth Nastoff, Ruby Rios | Pair programming Session to work on the design on the AI code | Specific tasks were assigned to each member at the meeting. |
| 11/30 | 1:00 – 3:00 PM  2 hours | Microsoft Teams | Joshua Koni,  Elizabeth Nastoff, Ruby Rios, Marley Symmonds | Code Review- Everyone talks through what they found in their individual reviews of the code, and then reviewed the code together. | No new action items were added at this meeting. Everyone is to continue working on their already assigned tasks. |
| 12/3 | 1:00-1:30 PM  30 Minutes | Microsoft Teams | Joshua Koni,  Elizabeth Nastoff, Ruby Rios, Marley Symmonds | Stand-up Meeting | New tasks were assigned to members who had completed their tasks. |
| 12/8 | 1:00-2:45 PM  1 hour and 45 minutes | Microsoft Teams | Joshua Koni,  Elizabeth Nastoff, Ruby Rios, Marley Symmonds | Stand-up Meeting and Pair-Programming Session | Members who had completed their tasks were assigned new tasks. |
| 12/9 | 9:45 PM - 3:45 AM  6 hours | Microsoft Teams | Joshua Koni,  Elizabeth Nastoff, Ruby Rios, Marley Symmonds | Stand-up Meeting and Pair-Programming Session | Members who had completed their tasks were assigned new tasks. All team members are to review the sprint documentation before our next meeting. |
| 12/10 | 6:00-7:00 PM  1 hour | Microsoft Teams | Joshua Koni,  Elizabeth Nastoff, Ruby Rios, Marley Symmonds | Presentation Meeting- Meeting to record the presentation video | We had some technical glitches with our video, so the plan is to meet back in an hour and try again. |

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* | | | | |
|  | Joshua Koni | Elizabeth Nastoff | Ruby Rios | Marley Symmonds |
| Joshua Koni | X | 1.0 | 1.0 | 1.0 |
| Elizabeth Nastoff | 1.0 | X | 1.0 | 1.0 |
| Ruby Rios | 1.0 | 1.0 | X | 1.0 |
| Marley Symmonds | 1.0 | 1.0 | 1.0 | X |
|  | *Average* | 1.0 | 1.0 | 1.0 | 1.0 |