CS471/571 Team Project

Instructions for the Midterm Report

Turn in a single zip file containing the report, source code files, compiled code files, and sample data (if any).

The midterm report should include the following titled sections. Reports must be typed and fonts should be consistent throughout the report. Points will be deducted for problems with correctness, completeness, clarity, structure, and writing style.

**Cover Page.** Provide the names of all team members and the table of contents.

**Section I. Team Organization and Buddy Rating** (1-2 pages):

1. Describe how tasks were divided among members and describe each member’s work in detail.
2. Provide a table of buddy ratings: each student will rate the performance of each of the other members of his/her group with a rating between 0 and 1. The average of the ratings each student receives from other members will be multiplied by the project grade to determine individual grades. To safeguard against a capricious rating, if a group member receives one and only one buddy rating below 0.8, it will be discarded.

**Section II. Product and Sprint Backlog**

1. User stories

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User story ID** | **User story description** | **Priority** | **Status** | **Estimated effort (hours)** |
| 1 | As a user, I need a GUI to interact with the game. | A | Complete | 2 |
| 2 | As a user, I need to be able to start a new game. | B | Complete | 1 |
| 3 | As a user, I need to be able to play with another person. | A | Complete | 6 |
| 4 | As a user, I need to see player pieces on a game board. | B | Complete | 2 |
| 5 | As a user, I need to see whose turn it is. | C | Complete | 1 |
| 6 | As a user, I need to know what to do during a turn. | C | Complete | 1 |
| 7 | As a user, I need to know when someone wins. | B | Complete | 2 |
| 8 | As a user, I need to be able to play against a computer opponent. | D | ToDo | 8 |
| 9 | As a user, I need to see where I can move pieces. | C | Complete | 2 |
| 10 | As a user, I need to be able to remove the opposing player's piece when a mill is formed. | A | Complete | 2 |
| 11 | As a user, I need to be able to move anywhere when I only have 3 pieces on the board. | B | Complete | 1 |
| 12 | As a user, I need to not be able to remove a piece within a mill unless it is the only option. | A | Complete | 1 |
| 13 | As a user, I need to not be able to place a piece on top of an opponent's piece. | A | Complete | 1 |
| 14 | As a user, I need to only be able to move a given piece to a empty space on the board next to it, unless having 3 pieces in play. | A | Complete | 2 |
| 15 | As a user, I need to be able to interact with the GUI environment using the mouse. | B | Complete | 2 |

1. Acceptance Criteria

|  |  |  |
| --- | --- | --- |
| **User story ID** | **Acceptance criteria** | **Status** |
| User story 1 | Display game board | Pass |
| Display player pieces | Pass |
| Display new game button | Pass |
| User story 2 | New Game button resets game board | Pass |
| New Game button resets player turn | Pass |
| User story 3 | Player pieces change as each player places pieces | Pass |
| Player label displays current player's turn | Pass |
| User story 4 | Pieces show up on game board when that space is clicked | Pass |
| User story 5 | Label updates showing current player's turn | Pass |
| User story 6 | Label updates showing instructions for current player's turn | Pass |
| User story 7 | Label updates showing winner of the game | Pass |
| No more pieces can be played until new game button clicked | Pass |
| A winner is found when a player has no valid moves | Pass |
| A winner is found when a player has less than 3 pieces in play on the game board | Pass |
| User story 8 | Option to select computer opponent | Fail |
| Computer opponent places pieces on the game board when it is its turn | Fail |
| Computer opponent plays the game somewhat intelligently | Fail |
| User story 9 | Highlight game board spaces where a currently clicked piece can move | Pass |
| User story 10 | Game recognizes when 3 pieces of the same player are lined up on the same column or row | Pass |
| Game keeps player turn as current player's when that player forms a mill | Pass |
| Can remove one of the opposing player's pieces when current player creates a mill | Pass |
| User story 11 | Once a player has only 3 pieces in play passed the first phase, player should have the option to move to any open space on the game board | Pass |
| Game board should highlight every space without a player currently occupying it when current player is attempting to move a piece when having 3 pieces | Pass |
| User story 12 | Not allow a piece to be removed when that piece is contained within an opponent's mill | Pass |
| If attempting to remove a piece, if the only pieces available to be removed are within an opponent's mill, allow one of those pieces to be removed | Pass |
| User story 13 | Attempting to place a piece on top of another piece currently placed on the game board does nothing | Pass |
| User story 14 | During phase when moving pieces, clicking on my own piece and then clicking an empty space next to that piece should show that piece in the new space and remove it from the old one | Pass |
| User story 15 | Using the mouse to click close button closes the game | Pass |
| Using the mouse to click new game button starts a new game | Pass |
| Using the mouse to click places on the game board places pieces on the game board | Pass |

1. Tasks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User story ID** | **Task description** | **Owner (developer)** | **Estimated effort (hours)** | **Status** |
| Story 1 | Create GUI frame/panels | Brendon | 0.5 | Pass |
| Board panel displays circles representing player pieces | Brendon | 1 | Pass |
| Create new game button | Brendon | 0.5 | Pass |
| Story 2 | Register new game button clicked | Brendon | .3 | Pass |
| Reset game by creating new Logic object | Brendon | .3 | Pass |
| Constructor for Logic resets game state by initializing data | Adam | .3 | Pass |
| Story 3 | Display player's played piece on game board | Brendon | 1 | Pass |
| Logic updates pieces played for the given player if valid placement | Adam |  | Pass |
| Logic does not update if pieces played on occupied spaces | Adam |  | Pass |
| Update GUI player label when turn changes | Brendon | 0.5 | Pass |
| Once a piece is placed, Logic updates to other player | Adam |  | Pass |
| Story 4 | GUI update to show current player piece displayed at location where player clicked | Brendon | 1 | Pass |
| Logic updates the locations occupied by the player that just placed a piece | Adam | 1 | Pass |
| Story 5 | GUI update player label with each change of player turn | Brendon | .5 | Pass |
| Logic updates by setting current player after each player places or moves pieces | Adam | .5 | Pass |
| Story 6 | GUI update instructions label to show current instructions for the player | Brendon | 1 | Pass |
| Story 7 | GUI update instructions label to display game winner | Brendon | .5 | Pass |
| Logic check for winner to see if a player won the game | Adam |  | Pass |
| Logic check winner after each player turn if a player has no valid moves | Adam |  | Pass |
| Logic check winner after each player turn if a player has less than 3 pieces in play | Adam |  | Pass |
| Story 8 | Add radio button to GUI to select 1 or 2 player game | Brendon | 1 | In Progress |
| GUI updates pieces on game board when Computer Logic plays a piece | Brendon | .5 | In Progress |
| Logic updates when computer pieces are played | Adam |  | In Progress |
| Computer Logic places pieces using Logic methods | Brian |  | In Progress |
| Computer Logic makes decisions on where to move based on current state of game board | Brian |  | In Progress |
| Story 9 | GUI update when in moving piece phase by showing available spaces to move when pieces are clicked | Brendon | .5 | Pass |
| Logic checks for available moves for current player for current piece in question | Adam | 1.5 | Pass |
| Story 10 | Logic checks for a mill being formed when 3 pieces line up in a row or column for current player | Adam |  | Pass |
| Logic keeps player turn as current player when a mill is formed | Adam |  | Pass |
| Logic attempts to remove one of the opposing player's pieces, selected by the user | Brian |  | Pass |
| GUI updates by no longer displaying the piece that was removed | Brendon | .5 | Pass |
| Story 11 | When a player has 3 pieces, the list of valid moves will contain all empty spaces. | Adam | .7 | Pass |
| GUI highlights empty spaces on game board, when a player with 3 pieces chooses a piece to move. | Brendon | .3 | Pass |
| Story 12 | When a piece is clicked, if it is in a mill, cannot be removed if there are other available options. | Bendon(GUI)/  Adam(Logic) | .5 | Pass |
| When a piece is clicked to remove, if in a mill, will only be removed if the only options are in a mill. | Brendon(GUI)/  Adam(Logic) | .5 | Pass |
| Story 13 | When a piece is moved or placed, it cannot be moved onto another piece. | Adam |  | Pass |
| Story 14 | When a piece is clicked in the appropriate phase, and then an empty valid move is clicked, the piece is moved to the new location. | Brendon | 2 | Pass |
| Story 15 | Clicking the Close button terminates the program | Brendon | .5 | Pass |
| Clicking the New Game Button resets the board, and lets players place pieces. | Brendon | .5 | Pass |
| Clicking on the Board places a player piece, if that spot is empty | Brendon | 1 | Pass |

1. Meetings

|  |  |  |  |
| --- | --- | --- | --- |
| Meeting # | Time/place | Participants | Topics and decisions |
| 1 | 9/15, 6:00 – 6:15, Engineering Bldg | Brian, Brendon, Adam | User Stories, Basic Design, Deciding meeting place |
| 2 | 9/24,6:00-6:15,  Metageek Lab | Brian, Brendon, Adam | Basic Board/GUI development. Board class and functions created. |
| 3 | 10/1,6:00-6:15,  Metageek Lab | Brian, Adam | Logic/GUI development.  Logic constructor, place pieces, beginning of move pieces function. GUI displays board, uses current functions. |
| 4 | 10/8,6:00-6:15,  Metageek Lab | Brian, Brendon, Adam | Finished move pieces, began check moves, and remove pieces, and check for win. GUI can use added functions. |
| 5 | 10/15,6:00-6:15,  Metageek Lab | Brian, Brendon, Adam | Finished check moves, remove, check win. GUI finished. |
| 6 | 10/22,6:00-6:15,  Metageek Lab | Brian, Brendon, Adam | Bugfixes, GUI beautification. |

**Section III. Summary of Test-Driven Development and Refactoring** (as many pages as needed)

1. Provide brief descriptions of the tests (not code), including the developer name(s). Every team member should provide at least four different tests.

|  |  |  |  |
| --- | --- | --- | --- |
| Test # | Description of test case (test input and oracle) | User story # and Task # | Developer(s) |
| mutatorTests() | Tests each mutator to check if they work. |  | Adam |
| playerOneFlyTest03() | Determines if player one, with only 3 pieces, has a valid moves consisting of all empty spaces on the board. | User Story 11  Task 1 | Adam |
| playerTwoFlyTest03() | Same Test as above, but for player two. | User Story 11  Task 1 | Adam |
| LogicTestPlacePiece() | Places one piece for each player, and checks those places on the board to see if they match. Places a mill for player one, checks if phase changes appropriately, then places all pieces to check phase change into move pieces phase. | User story 3  Task 2 | Adam |
| LogicTestInvalidPiece() | Places player one piece at 0,0 then asserts that a piece cannot be placed at 0,0 again | User Story 13  Task 1 | Adam |
| NodeCreateTest() | Tests if a Node object can be created | User Story 4  Task 2 | Brian |
| NodeSetPlayer1Test() | SetPlayer to 1:  Player == 1 | User Story 4  Task 2 | Brian |
| NodeSetPlayer2Test() | SetPlayer to 2:  Player == 2 | User Story 4  Task 2 | Brian |
| NodeSetPlayer0Test() | SetPlayer to 0:  Player == 0 | User Story 4  Task 2 | Brian |
| BoardCreateTest() | Tests if a Board object can be created | User Story 1  Task 2 | Brian |
| BoardAllEmptyNodeTest() | All Board Nodes have no players | User Story 2  Task 3 | Brian |
| SetBoardNodeP1Test() | Set Node Player1 :  Node == Player1 | User Story 3  Task 2 | Brian |
| SetBoardNodeP2Test() | Set Node Player2 :  Node == Player2 | User Story 3  Task 2 | Brian |
| BoardCheckMill() multiple | Create Mill:  checkMill == true | User Story 10  Task 1 | Brian |
| BoardCheckNotMill() | Create Non Mill combinations:  checkMill == false | User Story 10  Task 1 | Brian |
| LogicCreateTest() | Tests if Logic object can be created | User Story 2  Task 3 | Brian |
| CheckMovesTest() multiple | Match all valid moves from given position:  Move list == checkMoves() | User Story 9  Task 2 | Brian |
| CheckMovesWrongPhase() | Set phase 1 and check Moves:  checkMoves =={} | User Story 9  Task 2 | Brian |
| CheckMovesWrongPlayer() | Set node for player 1 and check Moves for player 2:  checkMoves =={} | User Story 9  Task 2 | Brian |
| LogicTestMovePiece() | Place pieces up to phase 2, attempt to move a piece to a valid spot:  movePiece == true | User Story 14  Task 1 | Brian |
| LogicTestInvalidMovePiece() | Move to a spot that is already occupied:  movePiece ==false | User Story 14  Task 1 | Brian |
| LogicTestRemovePiece | Remove a valid piece when mill is formed:  removePiece == true | User Story 10  Task 3 | Brian |
| LogicTestRemovePieceInMill() | Remove valid piece within mill if only option:  removePiece == true | User Story 12  Task 2 | Brian |
| LogicTestNoValidMoves() | Given player has no valid moves and is loser:  isLoser == true | User Story 7  Task 3 | Brian |
|  |  |  |  |
|  |  |  |  |

1. Provide brief descriptions of refactorings, including the developer name(s). Each team member should provide at least two different refactorings.

|  |  |  |
| --- | --- | --- |
| Refactoring # | Description of the refactoring (problem and solution) | Developer (s) |
| Refactor 1: | Check mill function had too many if statements, it was pruned down using modulus operator. | Adam |
| Refactor 2: | Remove pieces pruned down, passes node to be removed to a separate function that was initially repeated in an if statement. | Brian |
| Refactor 3: | Check win, split into an internal function and passes in required variables, instead of multiple if statements to check the correct player. | Brian |
| Refactor 4: | Remove Pieces. Added mill protection for opponents pieces, unless they are the only possible option. | Adam |
| Refactor 5: | Check Moves: simplified the process for checking if flying was possible for a player, moved to a separate function with the list of player pieces passed in, instead of if statements to determine the correct route. | Adam |
| Refactor 6: | Made hard-coded values for the GUI into constants so we could easily fine-tune the appearance of the game board | Brendon |
| Refactor 7: | Changed how the GUI components passed around mouse listeners to be more specific to our needs. | Brendon |
|  |  |  |
| Needs Refactoring | Check moves could definitely use refactoring, but we could not think of a better way. | All |

1. **Section IV. Summary of Pair Development** (as many pages as needed)

**Section IV. Summary of Pair Development**

1. Provide the date/time, place, developer names, and tasks for each pair development session.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Date/time/duration | Place | Developer names | Tasks |
| 1 | 9/15, 6:00 – 8:30,  2.5 hours | Engineering Bldg | Adam, Brian, Brendon | Rough GUI design, Logic design, test cases |
| 2 | 9/24,6:00-8:30,  2.5 hours | Metageek Lab | Adam, Brian, Brendon | Node class methods, Board class methods, checking for mills, testing mills, Basic GUI functionality |
| 3 | 10/1,6:00-8:30,  2.5 hours | Metageek Lab | Adam, Brian | Logic methods, placing pieces, testing placing pieces, GUI updating when pieces placed |
| 4 | 10/8,6:00-8:30,  2.5 hours | Metageek Lab | Adam, Brian, Brendon | Logic methods, removing pieces, testing removing pieces, GUI updating when removing pieces, bug fixes |
| 5 | 10/15,6:00-8:30  2.5 hours | Metageek Lab | Adam, Brian, Brendon | Logic methods, checking win conditions, testing win conditions, GUI updating when a win occurs, refactoring, bug fixes |
| 6 | 10/22,6:00-8:30,  2.5 hours | Metageek Lab | Adam, Brian, Brendon | Code cleanup, refactoring, testing, bug fixes |

**Section V. Lessons Learned**

*Brian Dunn*

1. What did you personally gain from the project?

The most important thing I've gained from this project so far is a better understanding of the Agile process and test driven development. For all of my past projects there was no set process in place that I would use to complete the project. Sometimes I would start one project a completely different way than I would start the next, depending on what the specifications of the projects were and what data and/or code I was originally given.

The agile process has given me a way to tackle projects to where I can progress through them in a similar manner, perhaps more efficiently than I had in the past. Test driven development has opened my eyes to a new way of developing a program. I have written tests in the past, but they typically came after I had an original prototype of a program written. The problem I found with doing it that way was that there were times where I would have to make drastic changes to my program, taking up much more time than if I had developed tests beforehand and then programmed based on the tests.

1. What does your program do well, and what could your program do better?

Our program performs to the specifications given and is a relatively robust implementation. It helps the user figure out what to do by providing an instruction label detailing the current phase of the game. Also, there is a color coded player label so players always know which pieces belong to which player. It properly changes phases and stops when a winner is found.

One thing our program could do better is the code could be refactored more as there are many lines of duplication. We refactored quite a bit by adding in new methods, however there are still other areas where there are many similar if statements. Another thing we could do to improve the program is implementing GUI testing. Unfortunately, my attempts at getting automated GUI testing working haven't worked out as I have hoped. So we've been having to do GUI testing manually by playing the game and picking up on errors that we hopefully discover. The problem seems to be that we are using awt components for the game board and swing for the labels and buttons. The swing components I can interact with, however the awt components don't seem to work properly when using GUI testing environments.

Also, something else I would like to add, other than the required computer player implementation, is sound effects, specifically some sort of error tone if the user attempts to do something invalid like place a piece on top of another piece.

1. How could you improve your development process if you develop a similar game from scratch?

I would have a better breakdown of the individual sprints. For the development of this game we weren’t 100% sure where to start so we just started adding various pieces in little by little and then tried to make them all fit in the end. Instead I would plan out the sprints by first figuring out what objects are going to interact with what others and in what ways. I would also probably start with the logic of the game first as that is the backbone of the whole process instead of starting with the GUI implementation. The GUI needs to know very little about the logic of the game, it just needs to update based on what the logic does.

*Brendon McCoy*

1. What did you personally gain from the project?

One thing I personally gained from working on this project was experience creating GUIs in Java. My past experience with that was fairly limited and had all been done a while ago, so I had to relearn some of the concepts and reference my old Java GUI project. The other big thing I gained from this project was experience coordinating with other team members to make sure our induvial components of the project could communicate properly with each other. We also tried to write our components in a way that would make it as easy as possible for the other components to use.

1. What does your program do well, and what could your program do better?

Our program does function properly and allows two players to successfully play a game of Nine Men’s Morris. It does a reasonably good job at giving clear instructions to the players at each phase of the game. I also does a good job of showing all the valid moves for a selected piece. The GUI will also scale well to different window sizes

However, our program does not yet support playing against a computer component. Also, while our program does not allow a player to make an invalid move, it could do a better job at displaying invalid moves. It could also be improved by displaying which of your opponent’s pieces you are free to remove when you form a mill. Finally, our game could do something more interesting when one of the players wins.

1. How could you improve your development process if you develop a similar game from scratch?

We could certainly improve our project management process. Since we had never worked with each before and were not yet super familiar with the agile process, we initially didn’t know how best to organize the project. However, we’ve gotten better at that as we have developed this project and could easily apply from the start on a new similar project. I would also utilize test-driven development more and write more test before doing the implementation.