**Project Sprint #3**

Implement all the features that support a human player to play a simple or general SOS game against a human opponent and refactor your existing code if necessary. The minimum features include **choosing the game mode (simple or general), choosing the board size, setting up a new game, making a move (in a simple or general game),** and **determining if a simple or general game is over**. The following is a sample GUI layout. It is required to use a class hierarchy to deal with the common requirements of the Simple Game and the General Game. **If your code for Sprint 2 has not considered class hierarchy, it is time to refactor your code**.

|  |  |  |
| --- | --- | --- |
| SOS Icon  Description automatically generated Simple game Icon  Description automatically generated General game Board size  8 | | |
| Blue player  Icon  Description automatically generated S  Icon  Description automatically generated O | Chart, line chart  Description automatically generated | Red player  Icon  Description automatically generated S  Icon  Description automatically generated O |
|  | Current turn: blue (or red) | New Game |

Figure 1. Sample GUI layout of the working program for Sprint 3

**Deliverables: expand and improve your submission for sprint 2.**

1. **Demonstration (9 points)**

Submit a video of no more than five minutes, clearly demonstrating the following features.

1. A simple game that the blue player is the winner
2. A simple draw game with the same board size as 1)
3. A general game that the red player is the winner, and the board size is different from 1)
4. A general draw game with the same board size as 3)
5. Some automated unit tests for the simple game mode
6. Some automated unit tests for the general game mode

In the video, you must explain what is being demonstrated.

1. **Summary of Source Code (1 points)**

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| Board.java – 592 lines | AC1TestCases – 40 lines | Source = 1,329 lines |
| Box.java – 97 lines | AC2TestCases – 34 lines | ACTestCases = 226 lines |
| GUI.java – 199 lines | AC3TestCases – 58 lines |  |
| HelloApplication.java – 33 lines | AC4TestCases – 47 lines |  |
| HelloController.java – 8 lines | AC6TestCases – 47 lines |  |
| PlayerBox – 72 lines |  |  |
| SOSGame – 90 lines |  |  |
| SimpleSOSGame – 91 lines |  |  |
| GeneralSOSGame – 102 lines |  |  |
| GameMode.java – 53 lines |  |  |
| Total | | 1,555 lines |

**You must submit all source code to get any credit for this assignment.**

1. **Production Code vs User stories/Acceptance Criteria (3 points)**

Summarize how each of the user story/acceptance criteria is implemented in your production code (class name and method name etc.)

|  |  |
| --- | --- |
| **User Story ID** | **User Story Name** |
| 1 | Choose a board size |
| 2 | Choose the game mode of a chosen board |
| 3 | Start a new game of the chosen board size and game mode |
| 4 | Make a move in a simple game |
| 5 | A simple game is over |
| 6 | Make a move in a general game |
| 7 | A general game is over |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Story ID** | **AC ID** | **Class Name(s)** | **Method Name(s)** | **Status (complete or not)** | **Notes (optional)** |
| 1 | 1.1 | Board | setBoardSize  initBoard  addBox | Complete | setBoardSize sets the board size to the specified size only if it is greater than 2. initBoard and addBox sets up the ArrayList depending on the boardSize. |
|  | 1.1 | GUI | makeBoard | Complete | Function sets the board up according to Board’s specified size (specifically, Board’s boardSize). |
|  | 1.2 | Board | setBoardSize | Complete | Function will not set the boardSize to 2 or anything less than 2. |
| 2 | 2.1 | Board | getIsSimpleGame | Complete | Checks to see if it is a simple game (true) or a general game (false). |
|  | 2.1 | GameMode | getGG, getSG | Complete | Gets the value of whether the Simple Game radio button is pressed or the General Game radio button is presses |
| 3 | 3.1 | Board | initBoard  setBoardSize | Complete | initBoard sets up the board according to boardSize.  setBoardSize makes sure that the number entered in is a valid size. |
|  | 3.1 | GUI | makeBoard  resetGUI | Complete | makeBoard sets up the GUI and the board according to the boardSize.  resetGUI removes and resets all the parts of the GUI so that it can run makeBoard again. |
|  | 3.2 | Board | setBoardSize | Complete | Makes sure that the input is a valid size. |
| 4 | 4.1 | Box | Box (the constructor)  drawO  drawS | Complete | Draws an ‘S’ or an ‘O’ in a specific box. The constructor sets up the SOSGame class and it’s children to draw and maintain the rules. |
|  | 4.1 | GUI | makeBoard | Complete | In the for loop, an nxn grid is created using Boxes. These Boxes are stored in both the ArrayList in Board and in the Pane GameBoard which will be used for the GUI. |
|  | 4.1 | PlayerBox | PlayerBox(constructor)  getS  getO | Complete | Checks to see if the S radio button is clicked or if the O radio button is clicked so it can know which one to place. |
|  | 4.1 | Board | getIsSimpleGame | Complete | Gets the Boolean value that it is a Simple Game. |
|  | 4.1 | SimpleSOSGame | gameRules  drawALetter (in the parent class) | Complete | The gameRules sets up the rules that the game will play by and show whose turn it is.  The drawALetter function (in the parent class SOSGame) does the actual drawing of the letters |
|  | 4.2 | SimpleSOSGame | gameRules  drawALetter (in the parent class) | Complete | These two functions when used together combined won’t allow a new S or O to be placed in a box that is already filled. |
|  | 4.2 | Board | getIsSimpleGame | Complete | Gets the Boolean value that it is a simple game. |
| 5 | 5.1/5.2 | SimpleSOSGame | gameRules | Complete | Will output who the winner is or if it is a tie based off of how many SOS’s each player has formed. |
|  | 5.1/5.2 | Board | checkForSOS  checkHorizontal  checkVertical  checkLeftToRightDiagonal  checkRightToLeftDiagonal | Complete | These functions check to see if an SOS has been formed. |
|  | 5.1 | GUI | drawLine | Complete | Draws a line in a formed SOS, the color is based off of which player completed it. |
| 6 | 6.1 | Box | Box (the constructor)  drawO  drawS | Complete | Draws an ‘S’ or an ‘O’ in a specific box. The constructor sets up the SOSGame class and it’s children to draw and maintain the rules. |
|  | 6.1 | GUI | makeBoard | Complete | In the for loop, an nxn grid is created using Boxes. These Boxes are stored in both the ArrayList in Board and in the Pane GameBoard which will be used for the GUI. |
|  | 6.1 | PlayerBox | PlayerBox(constructor)  getS  getO | Complete | Checks to see if the S radio button is clicked or if the O radio button is clicked so it can know which one to place. |
|  | 6.1 | Board | getIsSimpleGame | Complete | Gets the Boolean value that it is a General Game. |
|  | 6.1 | GeneralSOSGame | gameRules  drawALetter (in the parent class) | Complete | The gameRules sets up the rules that the game will play by and show whose turn it is.  The drawALetter function (in the parent class SOSGame) does the actual drawing of the letters |
|  | 6.2 | GeneralSOSGame | gameRules  drawALetter (in the parent class) | Complete | These two functions when used together combined won’t allow a new S or O to be placed in a box that is already filled. |
|  | 6.2 | Board | getIsSimpleGame | Complete | Gets the Boolean value that it is a General Game |
| 7 | 7.1/7.2 | GeneralSOSGame | gameRules | Complete | Will output who the winner is or if it is a tie based off of how many SOS’s each player has formed. |
|  | 7.1/7.2 | Board | checkForSOS  checkHorizontal  checkVertical  checkLeftToRightDiagonal  checkRightToLeftDiagonal | Complete | These functions check to see if an SOS has been formed. |
|  | 7.1 | GUI | drawLine | Complete | Draws a line in a formed SOS, the color is based off of which player completed it. |

1. **Tests vs User stories/Acceptance Criteria (3 points)**

Summarize how each of the user story/acceptance criteria is tested by your test code (class name and method name) or manually performed tests.

|  |  |
| --- | --- |
| **User Story ID** | **User Story Name** |
| 1 | Choose a board size |
| 2 | Choose the game mode of a chosen board |
| 3 | Start a new game of the chosen board size and game mode |
| 4 | Make a move in a simple game |
| 5 | A simple game is over |
| 6 | Make a move in a general game |
| 7 | A general game is over |

4.1 Automated tests directly corresponding to some acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID** | **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)** |
| 1 | 1.1 | AC1TestCase | testStartBoardWithValidSize | Initiates the board and sets the boardSize to a valid size (8), then checks to so see that the board has set the value to 8 (since it is a valid input) and that isBlueTurn is true (should start as Blue Player’s turn). |
|  | 1.2 | AC1TestCase | testStartBoardWithInvalidSize | Initiates the board and sets the boardSize to a invalid size (2), then checks to see that board doesn’t set the value to 2 (since it is an illegal value) and that isBlueTurn is true (should start as Blue Player’s turn). |
| 2 | 2.1 | AC2TestCase | testSimpleGameModeSelected | Initiates the board and sets the game mode to a Simple Game. Then checks to see that the game mode is Simple Game. |
|  | 2.1 | AC2TestCase | testGeneralGameModeSelected | Initiates the board and sets the game mode to a General Game. Then checks to see if the game mode is General Game. |
| 3 | 3.1 | AC3TestCase | testStartANewGameWithValidBoardSizeAndGameMode | Starts a game with valid size and game mode. Then tries changing it into another valid size and game mode. |
|  | 3.2 | AC3TestCase | testStartANewGameWithInvalidBoardSizeAndGameMode | Starts a game with valid size and game mode. Then tries changing it into an invalid size and game mode. |
| 4 | 4.1 | AC4TestCase | testSuccesfulMoveInSimpleGame | Sets the board to an 8x8, initiates the board, set the game mode to a Simple Game, and puts an S at (1,1) on the board. Then, checks to see if there is an S at (1,1) and that it is a Simple Game. |
|  | 4.2 | AC4TestCase | testUnsuccessfulMoveInSimpleGame | Sets the board to an 8x8, initiates the board, set the game mode to a Simple Game, and puts an S at (1,1) on the board. It then tries to place an O at (1,1). Then, checks to see if there is an S at (1,1) and that it is a Simple Game. |
| 6 | 6.1 | AC6TestCase | testSuccessfulMoveInGeneralGame | Sets the board to an 8x8, initiates the board, set the game mode to a General Game, and puts an S at (1,1) on the board. Then, checks to see if there is an S at (1,1) and that it is a General Game. |
|  | 6.2 | AC6TestCase | testUnsuccessfulMoveInGeneralGame | Sets the board to an 8x8, initiates the board, set the game mode to a General Game, and puts an S at (1,1) on the board. It then tries to place an O at (1,1). Then, checks to see if there is an S at (1,1) and that it is a General Game. |

4.2 Manual tests directly corresponding to some acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID** | **Acceptance Criterion ID** | **Test Case Input** | **Test Oracle (Expected Output)** | **Notes** |
| 1 | 1 | 2 in the textField | No change |  |
|  | 1 | 4 in the textField | A board size 4x4 |  |
| 2 | 2.1 | Click on the Simple and General game radio button | Whichever radio button is clicked when clicking the New Game button, that should show as the new selected game mode. |  |
| 3 | 3.1 | Changes from a 3 in the input box to an 8 and clicks the New Game Button | Should change to an 8x8 board. |  |
|  | 3.2 | Changes from a 3 in the input box to a 2 and clicks the New Game Button | Shouldn’t change the size of the board. |  |
| 4 | 4.1 | Click the Simple Game button, the O button, and select an open box. | An O should be placed at that box |  |
|  | 4.2 | Click the Simple Game button, the S button, and select an occupied box | An S should not be placed in that box. |  |
| 5 | 5.1 | Forms a SOS on the Top Row by the Blue Player on a 8x8 board | Should display that the Blue Player has won and put a line through the SOS |  |
|  | 5.2 | Fills the entire board but no SOS on a 8x8 board | Should display that the game is a tie. |  |
| 6 | 6.1 | Click the General Game button, the O button, and select an open box. | An O should be placed at that box |  |
|  | 6.2 | Click the General Game button, the S button, and select an occupied box | An S should not be placed in that box. |  |
| 7 | 7.1 | Has the Red Player form more SOS’s than the Blue Player on a 3x3 board | Should Display that the Red Player has won and put lines through the SOSs |  |
|  | 7.2 | Have the Red and Blue Player have an even number of SOS’s formed. | Should display the game is a tie. |  |

4.3 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** |
|  |  |  |  |  |
|  |  |  |  |  |

1. **Describe how the class hierarchy in your design deals with the common and different requirements of the Simple Game and the General Game**? **(4 points)**

I first created the parent class SOSGame. This parent class had all the variables needed for the children classes (like the board of the game, the SOSBoard that keeps track of the SOS’s, the blue and red player controls and radio buttons, the GUI and the box that this selection has taken place in, and the display that shows who won or whose turn it is). The only function in the parent is the drawALetter function, since both of the children classes will need to draw a letter in order to play the game. Then comes the children: SimpleSOSGame and GeneralSOSGame. The only functions that these classes have are the constructors and the gameRules function. The gameRules functions are in charge of keeping the game going with the expected rules. And since these rules are different for each gameMode, each of the classes have their own unique gameRules function rather than the parent having one function and the children inheriting them.