**Project Sprint #2**

Implement the following features of the SOS game: (1) the basic components for the game options (board size and game mode) and initial game, and (2) S/O placement for human players ***without*** checking for the formation of SOS or determining the winner. The following is a sample interface. The implementation of a GUI is strongly encouraged. You should practice object-oriented programming, making your code easy to extend. It is important to separate the user interface code and the game logic code into different classes (refer to the TicTacToe example). xUnit tests are required.

|  |  |  |
| --- | --- | --- |
| 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 | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  | | O |  |  |  |  |  |  |  | |  |  | S | O | S |  |  |  | |  |  |  |  | S |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | S | | Red player  Icon  Description automatically generated S  Icon  Description automatically generated O |
|  | Current turn: blue (or red) |  |

Figure 1. Sample GUI layout of the Sprint 2 program

**Deliverables:**

1. **Demonstration (8 points)**

Submit a video of no more than three minutes, clearly demonstrating that you have implemented the required features and written some automated unit tests. In the video, you must explain what is being demonstrated.

|  |  |  |
| --- | --- | --- |
|  | **Feature** |  |
| 1 | Choose board size |  |
| 2 | Choose game mode |  |
| 3 | Initial game of the chosen board size and game mode |  |
| 4 | “S” moves |  |
| 5 | “O” moves |  |
| 6 | Automated unit tests |  |
| … |  |  |

Here’s the video:



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

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| Board.java |  |  |
| GUI.java |  |  |
| Simple .java |  |  |
| General.java |  |  |
| TestBoard.java |  |  |
| TestGUI.java |  |  |
| Testsimple.java |  |  |
| Testgeneral.java |  |  |
|  |  |  |
| Total | |  |

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

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

Update your user stories and acceptance criteria from the previous assignment and ensure they adequately capture the requirements. Summarize how each of the following 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 |
| 6 | Make a move in a general game |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Story ID and Name** | **AC ID** | **Class Name(s)** | **Method Name(s)** | **Status (complete or not)** | **Notes (optional)** |
| 1. Choose a board size | 1.1 | AC 1.1 Selecting board size  Given that the player has started the SOS game and is presented with the initial game menu.  When the player selects the “Choose board size” option from the menu  Then the player should be presented with a screen that displays a range of available board sizes, with the smallest and largest sizes clearly labeled and highlighted. | The class **board** initialized the size of board, the cell, etc  **public** **void** actionPerformed(ActionEvent e)  which is in the class GUI is the function that makes sure that the right board size is chosen by the player. If it’s not the case a message will be prompted  There’s also the function that check if the size is valid or not which is called:  **public** **boolean** validSize(**int** size) | Completed |  |
| 1.2 | AC 1.2 Changing board size during the game  Given that the player has selected a board size and begun a game.  When the player attempts to switch to a different board size during the game  Then the game should display a confirmation dialog or prompt, informing the player that their progress will be lost if they switch to a new board size, and asking them to confirm their choice before proceeding. | This part is not implemented yet but it’ll be inside the initboard like an error message. An I’ll have to include a replay button just like the New game on the bottom. | Incompleted | The replay button is not ask in this sprint that’s why this AC is not implemented yet. |
| 2. Choose the game mode of a chosen board | 2.1 | AC 2.1 Selecting the game mode of a chosen board  Given a player has selected a board for the game  When the player is prompted to choose a game mode,  Then the player must be presented with 2 options: simple game, and general game. The player must be able to select one of these options, and the game must proceed accordingly based on the mode selected. | The two-mode are written as Button and only one can be chosen in the class GUI  The simple mode is implemented as  JRadioButton simpleGame = **new** JRadioButton("Simple Game");  The general mode is implemented as:  JRadioButton generalGame = **new** JRadioButton("General Game");  There’s also a class called simple where the specificity of what’s going to happen in the simple mode is going to be located like checking if there’s a win or not | Completed |  |
| 3. Start a new game of the chosen board size and game mode | 3.1 | AC 3.1 Start a new game with the chosen settings  Given a player has selected a board size and game mode  When the player chooses to start a new game  Then the game must implement the rules specified for the mode chosen such as time limit, or reduced lives for incorrect moves. | This part is implemented by the button called New game so when we want to start a new game we just choose the board size and the game mode and click on it the play. And the function that allows that is in the class GUI as is called:  **public** **void** actionPerformed(ActionEvent e) | Completed |  |
| 4. Make a move in a simple game | 4.1 | AC 4.1 Valid move in a simple game.  Given a player is in an SOS game,  When the player attempts to make a valid move which is placing “S” or “O” in an unoccupied cell,  Then the game board should be updated accordingly, and the turn should be passed to the next player. | This AC is not fully implemented that’s why when I run the test code I didn’t have an error but a warning. Its implementation is located in the class board and the function doing the job is :  **public** **boolean** makeMove(**int** row, **int** column)  but the turn choice and set up are done by:  **public** **char** getTurn()  **public** **void** setTurn(**char** t) | Completed |  |
| 4.2 | AC 4.2 Invalid move in a simple game.  Given a player is in a SOS game,  When the player attempts to make an invalid move like trying to place “S” or “O” in an occupied cell,  Then the game board should not be updated, and the player should be prompted to make a valid move. The turn should remain with the same player until a valid move is made. | This part is not completed yet but it’ll be a warning that I can implement in the function  **public** **boolean** makeMove(**int** row, **int** column) | Incompleted |  |
| 6. Make a move in a general game | 6.1 | AC 6.1 Valid move in a general game  Given a player is in a general SOS game.  When the player makes a move by placing either an “S” or an “O” in an empty cell.  Then the game should correctly update the grid with the player’s move, and the turn should be passed to the next player. | The move are implemented through the function:  **public** **void** actionPerformed(ActionEvent e) | Complete |  |
| 6.2 | AC 6.2 Invalid move in a general game  Given a player is in a general SOS game,  When the player attempts to make an invalid move, such as placing a letter in a cell that is already occupied or attempting to place a letter outside the bounds of the game board,  Then the game board shouldn’t be updated, and the player should be prompted to make a valid move. The turn should remain with the same player until a valid move is made. | This part is not completed yet but it’ll be a warning that I can implement in the function  **public** **boolean** makeMove(**int** row, **int** column) | Incomplete |  |

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 |
| 6 | Make a move in a general game |

4.1 Automated tests directly corresponding to the acceptance criteria of the above user stories

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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)** |
| 1 | 1.1 | Testboard | **public** **void** testInitBoard() |  |
| 1.2 | incomplete |  |  |
| 2 | 2.1 | TestGUI | **public** **void** test() |  |
| 3 | 3.1 | TestGUI | **public** **void** test() |  |
| 4 | 4.1 | Testboard | **public** **void** testMakeMove() |  |
| 4.2 | incomplete |  |  |
| 6 | 6.1 | TestBoard and TestGeneral | **public** **void** testMakeMove()  **public** **void** test() |  |
| 6.2 | incomplete |  |  |

4.2 Manual tests directly corresponding to the acceptance criteria of the above user stories

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID and Name** | **Acceptance Criterion ID** | **Test Case Input** | **Test Oracle (Expected Output)** | **Notes** |
| 1 | 1.1 | **public** **void** testInitBoard() |  |  |
|  | 1.2 |  |  |  |
|  | … |  |  |  |
| 2 | 2.1 |  |  |  |
|  |  |  |  |  |
| 4 |  | **public** **void** testMakeMove() |  |  |
|  | … |  |  |  |

4.3 Other automated or manual tests not corresponding to the acceptance criteria of the above user stories

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number** | **Test Input** | **Expected Result** | **Class Name of the Test Code** | **Method Name of the Test Code** |
| 1 |  |  | **public** **void** testSetCell() |  |
| 4 |  |  | **public** **void** testSwitchTurn() |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |