**Project Sprint #4**

Implement all the features that support a player (**human or computer**) to play a simple or general SOS game against another player (**human or** **computer**). The minimum features include **choosing human or computer for red and/or blue players**, **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 computer component must be able to play complete simple and general games. You are encouraged to consider basic strategies for winning simple or general games (e.g., against a poor human player). Optimal play is not required.

The following is a sample GUI layout. You should use a class hierarchy to deal with the computer opponent requirements. If your current code has not yet 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 Human  Icon  Description automatically generated S  Icon  Description automatically generated O  Icon                          Description automatically generated Computer | Chart, line chart  Description automatically generated | Red player  Icon  Description automatically generated Human  Icon  Description automatically generated S  Icon  Description automatically generated O  Icon  Description automatically generated Computer |
|  | Current turn: blue (or red) | New Game |

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

**Total points: 24**

1. **Demonstration (8 points)**

Submit a video of no more than five minutes, clearly demonstrating that you have implemented the computer opponent features and written some automated unit tests.

1. A complete simple game where the blue player is a human, the red player is the computer, and there is a winner
2. A complete general game where the blue player is the computer, the red player is a human, and there is a winner
3. A complete simple game where both sides are played by the computer
4. A complete general game where both sides are played by the computer
5. Some automated unit tests for the computer opponent.

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

1. **User Stories for the Computer Opponent Requirements (1 points)**

* **User Story Template**: As a <role>, I want <goal> [so that <benefit>]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **User Story Name** | **User Story Description** | **Priority** | **Estimated effort (hours)** |
| 8.1 | Computer makes a move | As a computer player, I want it to be able to place either a S or an O on the board randomly so that the game can continue. | 1 | 10 |
| 8.2 | Computer makes a SOS | As a computer player, I want it to be able to place either a S or an O on the board randomly so that an SOS can be formed. | 1 | 5 |
|  |  |  |  |  |

1. **Acceptance Criteria (AC) for the Computer Opponent Requirements (4 points)**

Add or delete rows as needed.

|  |  |  |  |
| --- | --- | --- | --- |
| **User Story ID and Name** | **AC**  **ID** | **Description of Acceptance Criterion** | **Status (completed, toDo, inPprogress)** |
| Story 8 | 8.1 | AC 8.1 <It’s the computer’s turn to make a move>  Given it is the computer’s turn to make a move  When the other player has made their move  Then the computer should randomly put an S or an O in a random spot on the board. | Completed |
| 8.2 | AC 8.2 <SOS is formed by the computer>  Given it is the computer’s turn to make a move  When the other player has made their move  And the computer has put a S or an O in a random spot the makes a SOS.  Then the computer should be awarded a point(s)  And the computer should go again. | Completed |
| … |  |  |
| … |  |  |

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

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| Board.java - 634 lines | AC1TestCases.java – 40 lines | Source Code Total Lines: 1,602 |
| Box.java – 97 lines | AC2TestCases.java – 34 lines | Test Code Total: 263 |
| ComputerPlayer.java – 22 lines | AC3TestCases.java – 58 lines |  |
| GameMode.java – 53 lines | AC4TestCases.java – 47 lines |  |
| GeneralSOSGame.java – 151 lines | AC6TestCases.java – 47 lines |  |
| GUI.java – 216 lines | AC8TestCases.java – 34 lines |  |
| HelloApplication.java – 32 lines |  |  |
| HelloController.java – 8 lines |  |  |
| HumanPlayer.java – 18 lines |  |  |
| PlayerBox.java – 87 lines |  |  |
| PlayerParent.java – 18 lines |  |  |
| SimpleSOSGame.java – 137 lines |  |  |
| SOSGame.java – 129 lines |  |  |
| Total | | 1,865 lines of code |

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

1. **Production Code vs New User stories/Acceptance Criteria (2 points)**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Story ID and Name** | **AC ID** | **Class Name(s)** | **Method Name(s)** | **Status (complete or not)** | **Notes (optional)** |
| 8 | 8.1 | ComputerPlayer | checkToDrawSOrO | Complete | Checks to put an S or O down. |
|  | 8.1 | GeneralSOSGame | gameRules | Complete | Does the gameRules for a general game. |
|  | 8.1 | PlayerParent | drawS, drawO | Complete | Parent class that lets child classes draw an S and an O. |
|  | 8.1 | SimpleSOSGame | gameRules | Complete | Does the gameRules for a simple game. |
|  | 8.2 | Board | checkForSOS | Complete | Checks for SOS horizontally, vertically, and diagonally. |

1. **Tests vs New User stories/Acceptance Criteria (2 points)**

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

6.1 Automated tests 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)** |
| 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. |
| 8 | 8.1 | AC8TestCases | ComputerMakesAMove | Checks if a blue computer player can make a move and if a red computer player can make a move in different game modes. | |

6.2 Manual tests directly corresponding to some acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID and Name** | **Acceptance Criterion ID** | **Test Case Input** | **Test Oracle (Expected Output)** | **Notes** |
| 8 | 8.1 | Set the red player as computers and have them make moves in a Simple and General Game |  |  |
|  | 8.2 | Blue and Red computer should form an SOS in a Simple and General Game |  |  |

6.3 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. **Present the class diagram of your production code (3 points) and describe how the class hierarchy in your design deals with the computer opponent requirements (3 points)**?

A screenshot of a computer

Description automatically generated with medium confidence

For the class hierarchy for players, I created a player parent class (PlayerParent) that does the actual drawing of S and O (since both players will need to do that) and a function checkToDrawASOrO that isn’t implemented. This is because the human player will have a different way to check to play a S or an O than a computer will have to check to play S or an O. If it is a human player, then the program will check to see if the S is selected or the O is selected. If it is a computer, then it will randomly pick a S or an O.