**Project Sprint #4**

The SOS game is described in CS449HomeworkOverview.docx. You should read the description very carefully.

Your submission must include the GitHub link to your project and you must ensure that the instructor has the proper access to your project. You will receive no points otherwise.

**GitHub link: https://github.com/gwb9h3/SOS-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 link to a video of no more than five minutes, clearly demonstrating that you have implemented the computer opponent features and written some automated unit tests. No points will be given without a video link.

**YouTube/Panopto link: https://umsystem.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=c5080f77-fbb2-4434-a5db-b22600577cc1**

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 | A Computer Player is playing | As a player, I want to have the option to play against a computer player so that I can play the game on my own. | 1 | 3 |
| 9 | Start a game with two computer players | As a player, I want to be able to see two computers play against each other to see the computers strategy. | 2 | 1 |
|  |  |  |  |  |

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)** |
| 8. A computer player is playing | 8.1 | AC 8.1 Valid Computer Move Placement  Given: the computer selects the best move on the board or a random move,  When: the system places a “S” or “O” in the cell,  Then: the chosen letter should appear in the cell and points should be awarded if necessary. | Completed |
| 8.2 | AC 8.2 Turn Switch After Valid Move  **Given:** a computer has made a valid move, **When:** the move is completed, **Then:** the turn should switch to the other player, and the board should list the current turn. | Completed |
| 8.3 | **AC 8.3: Prevent Further Moves After Game Over**  **Given:** the game is marked as over (by the win Boolean), **When:** a computer attempts to make a move, **Then:** the move should be prevented from making a move. | Completed |

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

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| Main.java | Production | 14 |
| GameController.java | Production | 70 |
| BoardData.java | Production | 126 |
| Move.java | Production | 19 |
| PlayerMove.java | Production | 35 |
| ComputerMove.java | Production | 157 |
| GameBoard.java | Production | 141 |
| SimpleGameBoard.java | Production | 48 |
| GeneralGameBoard.java | Production | 40 |
| GameView.java | Production | 217 |
| GameBoardTest.java | Test | 47 |
| GameBoardTest2.java | Test | 69 |
| Total | | 983 |

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.Valid Computer move placement | 8.1 | MakeComputerMove.java GameBoard.java | handleButtonClick cpuGame & ComputerMove Class | Complete | The computer move class handles executing the move on the back end and finds the best available move for the board while the handle button click and cpuGame are for either starting a game and passing off the turns. |
| 8. Turn Switch After Valid Move | 8.2 | GameBoard.java | handleButtonClick cpuGame | Complete | After the computer move is processed the turn is switched to the opposing player (either a computer or real player) |
| 8. Prevent Further Moves After Game Over | 8.3 | GameBoard.java | hadleButtonClick cpuGame | Complete | There is a boolean value that is returned from the make move functions that will update whether the game is over or not. |

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)** |
| 8.A computer player is playing | 1.1 | GameBoardTest2.java | testValidComputerMovePlacement | The test case does a move and checks to see if the board has had an updated cell inside. |
| 8.A computer player is playing | 8.2 | GameBoardTest.java | testTurnSwitchAfterValidComputerMove | The test case sets the turn to the cpu and then makes the move. Following the move the board updates the turn which is the same as found in the code. It should then show the next players turn as true |

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 A computer player is playing | 8.3 | The player has completed a move to end a simple game | The computer will not follow up with another move | #This is handled via a simple boolean value check before the computer can make a move. |

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)**?

**The two class hierarchies are on the following two pages. The first class hierarchy handles most of the differences between simple and general games but does hold a lot of the code that handles the computer opponents moves being played inside of the gameboard.java file. The majority of the code that handles the computer moves is found inside of the Move class hierarchy. The first Move class holds the initialization of the execute function but the playerMove and computerMove classes both have different implementation of the function. This allows for the same function to be used while doing different things for each different case.**

A notebook with writing on it

Description automatically generated

A notebook with writing on it

Description automatically generated