**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**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **User Story Name** | **User Story Description** | **Priority** | **Estimated effort (hours)** |
| 8 | Computer Oponent | * As a player, I want the computer opponent to make strategic moves based on the current game state, So that the game remains challenging. | High | 10 |
| .. |  |  |  |  |
|  |  |  |  |  |

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 story one | 8.1 | AC 8.1 <scenario description>  Computer Opponent Move  Given: The game is in progress.  When: It's the computer's turn.  Then: The computer should make a valid move on the game board. | completed |
| 8.2 | AC 8.2 Computer Opponent Strategy  Given: The game is in progress.  When: It's the computer's turn.  Then: The computer should make a move that maximizes its score on the game board.  Computer player makes valid moves alternating with a human player The computer player should make valid moves on the game board, taking turns with the human player | Completed  completed |
| … |  |  |
| 9 story nine if needed | 9.1 | Computer player checkbox enables computer gameplay. The checkbox for the computer player should be functional and enable computer gameplay when selected.  Computer player wins/loses/draws the game properly against a human 8.3 The game should correctly determine the outcome (win/lose/draw) when playing against a human with the computer opponent. | Completed  completed |
| … |  |  |

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

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| UserPanel | Production | 292 |
| Board  BoardLabel  Game  Main | Production  Production  Production  production | 168  23  237  19 |
| Total | | 739 |

**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 | UserPanel | playNextMove | complete | Computer player logic needs to be implemented. |
|  | 8.2 | Game | Game | compete | Win/lose/draw conditions need to be handled properly. |
|  | … |  |  |  |  |

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 |  |  |  |
|  | 1.2 |  |  |  |
|  | … |  |  |  |
| 2 | 2.1 |  |  |  |
|  | … |  |  |  |

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** |
| 1 | 1. | Click on the "Computer Player" checkbox | Verify if the player type is updated to "Computer." | Click on the "Computer Player" checkbox in the UI and check if the player type is displayed as "Computer |
|  | 1.2 | Play a game with a mix of human and computer moves. Alternate between human and computer moves. | Verify that each move is valid | Start a game with both human and computer players.  - Observe and verify that moves made by the computer are valid.  - Ensure that the game proceeds with alternating valid moves. |
|  | … |  |  |  |
| 2 | 2.1 | Play a complete game with both human and computer moves | Check the outcome of the game (win/lose/draw) and ensure it is correctly determined | Play a complete game with a mix of human and computer moves.  - Observe the game outcome and check if it's correctly determined (win/lose/draw) |
|  | … |  |  |  |

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 **Game** class is in charge of managing the overall game state, which includes player turns and interactions.

The **UserPanel** class represents each player's panel, allowing them to make selections and maintain track of their results.The **Board** class represents the game board, handles user clicks, and includes a method for the computer player to perform its move.The **BoardLabel** class stores the row and column indices and represents a label for each cell on the game board.The Move class in **UserPanel** is an inner class that represents a move in the game by storing the row, column, score, and symbol to play.

The class hierarchy addresses the computer opponent needs by giving players the option of being either human-controlled or machine-controlled. The computer player logic is implemented in the UserPanel class's playNextMove method, which calls the Move class to calculate the next move depending on the current state of the game board.