**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 | Have a single computer player | As a player I want to be able to select the option to play against one computer player, so that I can play a game against the computer whenever I want. | 1 | 2 |
| .. |  |  |  |  |
| 9 | Have two computer players | As a player I want to be able to select the option for both players to be computer players so that I can watch a game happen. | 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 Have a single Computer Player | 8.1 | AC 8.1 Playing against a computer player and someone wins  Given the gamemode is set to either simple or general and one of the players is a robot  When the rules for either game mode rules are met and either the human or the robot has scored more points  Then the game should be over and display the score and the choice to play again. | Complete |
| 8.2 | AC 8.2 Playing against a computer player and it’s a tie  Given the game mode is set to simple or general and one of the players is a robot  When the rules for either game mode are met and the human and the robot have scored the same number of points or the board is full with no points  Then the game should display there was a tie and give the choice to play again | Complete |
| … |  |  |
| 9 story nine if needed | 9.1 | AC 9.1 Two computers play against eachother and one wins  Given the game mode is either simple or general and the two players are both computers  When the game rules for a win are met and one player has more points than the other  Then display which player won how many points they had, and prompt the human to ask if they should play again. | Complete |
| 9.2 | AC 9.2 Two computers play against eachother and no one wins  Given the game mode is either simple or general and the two players are both computers  When the board is completely full of tokens and the scores are the same for both players  Then display that there was a tie and ask the human player if they would like the robots to play again. | Complete |

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

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| Start.py | Production | 11 |
| GUI.py | Production | 393 |
| Board.py | Production | 252 |
| Tests.py | Test | 168 |
| Total | | 824 |

**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 Have a single computer player and a winner | 8.1 | App, Board | buildStartMenu, buildMainGame, robotMoves, placePiece, checkSPlacedPoint, checkOPlacedPoint, checkSimpleWin, noOpenSpaces, getGenrealWinner, chooseRandomEmptySpace,  getRandomToken, reset | Complete |  |
| 8 Have a single computer player and a tie | 8.2 | App, Board | buildStartMenu, buildMainGame, robotMoves, placePiece, checkSPlacedPoint, checkOPlacedPoint, checkSimpleWin, noOpenSpaces, getGenrealWinner, chooseRandomEmptySpace,  getRandomToken, reset | Complete |  |
| 9 Have two computer players and a winner | 9.1 | App, Board | buildStartMenu, buildMainGame, robotMoves, placePiece, checkSPlacedPoint, checkOPlacedPoint, checkSimpleWin, noOpenSpaces, getGenrealWinner, chooseRandomEmptySpace,  getRandomToken, reset | Complete |  |
| 9 Have two computer players and a tie | 9.1 | App, Board | buildStartMenu, buildMainGame, robotMoves, placePiece, checkSPlacedPoint, checkOPlacedPoint, checkSimpleWin, noOpenSpaces, getGenrealWinner, chooseRandomEmptySpace,  getRandomToken, reset | Complete |  |

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 Have a single computer player and a winner | 8.1 | UnitTestClass, App | StartGameWithOneRobot | With the input of selecting any board size and any game mode we should and selecting player 1 as a robot, we should expect the outcome of the assertion to be that oneRobot is true and twoRobots is false |
| Have a single computer player and a tie | 8.2 | UnitTestClass, App | StartGameWithOneRobot | With the input of selecting any board size and any game mode we should and selecting player 1 as a robot, we should expect the outcome of the assertion to be that oneRobot is true and twoRobots is false |
|  | … |  |  |  |
| 9 Have two computer players and a winner | 9.1 | UnitTestClass, App | StartGameWithTwoRobots | With the input of selecting any board size and any game mode and selecting player 1 and player 2 to be robots we expect the outcome of the assertion to be oneRobot is false and twoRobots is true. |
| Have two computer players and a tie | … | UnitTestClass, App | StartGameWithTwoRobots | With the input of selecting any board size and any game mode and selecting player 1 and player 2 to be robots we expect the outcome of the assertion to be oneRobot is false and twoRobots is 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 Have a single computer player and a winner | 8.1 | Select red to be a robot and make human moves to set up a winning scenario for your self. | Game should end with blue player winning assuming it’s possible. And ask the human if they would like to play again. |  |
| Have a single computer player and a tie | 8.2 | Select red to be a computer player and play against red to try to end in a tie avoiding SOS’s. | Game should end with a tie screen showing and asking the human if they would like to play again. |  |

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)**?

**How many class hierarchy deals with the computer opponent requirements**

* Since there really isn’t much of a difference between two humans playing each other, one human and one robot, or two robots playing each other, the game rules didn’t need to change. What ended up happening was if the blue player is a robot since they have the first turn on launch, check in the main game loop to see if blue is a robot, if so launch into a special function that lets the robot select a token randomly and a random available space. After that, we either hold until a human player selects an available button, or we see that the other player is a robot as well and relaunch the robot choice function. Both the human choice and robot function use the same logic for actually updating the board GUI and the board’s logic side in the board class. And we still use the board class to keep track of score and if someone has won or not yet. The only difference between this sprint and the last sprint is adding a check to see if the opposite player is a robot, if not continue as normal, but if it is a robot launch into a special function then return to the normal function.

**Class Diagram of Production Code**

