**Project Sprint #3**

Implement all the features that support a human player to play a simple or general SOS game against a human opponent and refactor your existing code if necessary. The minimum features include **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 following is a sample GUI layout. It is required to use a class hierarchy to deal with the common requirements of the Simple Game and the General Game. **If your code for Sprint 2 has not 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 S  Icon  Description automatically generated O | Chart, line chart  Description automatically generated | Red player  Icon  Description automatically generated S  Icon  Description automatically generated O |
|  | Current turn: blue (or red) | New Game |

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

**Deliverables: expand and improve your submission for sprint 2.**

1. **Demonstration (9 points)**

Submit a video of no more than five minutes, clearly demonstrating the following features.

1. A simple game that the blue player is the winner
2. A simple draw game with the same board size as (a)
3. A general game that the red player is the winner, and the board size is different from (a)
4. A general draw game with the same board size as (c)
5. Some automated unit tests for the simple game mode
6. Some automated unit tests for the general game mode

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

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

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| Start.py | Production | 11 |
| GUI.py | Production | 261 |
| Board.py | Production | 234 |
| Tests.py | Test | 149 |
| Total | | 655 |

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

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

Summarize how each of the 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 |
| 5 | A simple game is over |
| 6 | Make a move in a general game |
| 7 | A general game is over |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Story ID** | **AC ID** | **Class Name(s)** | **Method Name(s)** | **Status (complete or not)** | **Notes (optional)** |
| 1 Choose a valid board size | 1.1 | App, Board | buildStartMenu, buildMainGame, createBoard, resetBoard, updateBoardSize | Complete | GUI passes along board size values to board class and methods to update logical board of the game, then updates GUI board with that information as well. |
| Choose an invalid board size | 1.2 | App, Board | buildStartMenu, buildMainGame, createBoard, resetBoard, updateBoardSize | Complete | The first game the user can only select values between 3 and 10, after that players can manually update board size, but the board class has preconditions of board >2 and board <11. |
|  | … |  |  |  |  |
| 2 Choose the game mode of a chosen board | 2.1 | App, Board | buildStartMenu, buildMainGame, updateGameMode, reset | Complete | On first launch the user is offered two choices of game modes to choose from, if none is chosen it defaults to simple game mode, but the user can change that before even placing the first token. |
| Choose an invalid game mode of a chosen board | 2.2 | App, Board | buildStartMenu, buildMainGame, updateGameMode, reset | Complete | The only way for a user to select an invalid game mode would be to not choose a game mode at all, which if the player does it will default to simple game mode. |
|  | … |  |  |  |  |
| 3 Start a new game of the chosen board and game mode. | 3.1 | App, Board | buildStartMenu, buildMainGame, createboard | Complete | User is prompted with a start screen where they can choose default parameters of the board, |
| Start a new game of an invalid board size and invalid game mode | 3.2 | App, Board | buildStartMenu, buildMainGame, createboard | Complete | Really not possible for the user to obtain, some things are set by default and invalid settings are not possible from the start menu due to pre and post conditions. |
|  | … |  |  |  |  |
| 4 Make a valid move in a simple game | 4.1 | App, Board | Clicked, buildMainGame, getPeice, placePiece, updateCurrentPlayerText, checkSPlacedPoint, | Complete | Assuming a valid spot exists and the game is not over, when the user selects a valid button and the game mode is set to simple, place their token in the selected spot and update board information. |
| Make an invalid move in a simple game | 4.2 | App, Board | Clicked, buildMainGame, getPeice, placePiece, updateCurrentPlayerText, | Complete | Assuming there are still valid spots on the board, and the game mode is set to simple, if the user selects a spot that is not available, already taken by themselves or the other player, do not update anything on the board and instead display an error message and allow the user to select a different available spot |
|  | … |  |  |  |  |
| 5 A simple game is over and there is a winner | 5.1 | App, Board | checkSPlacedPoint, checkOPlacedPointm checkForSimpleWin, getPlayerPoints, noOpenSpaces, getGeneralWinner | Complete | Assuming the game mode is set to simple, when somebody is the first to get an SOS in a row, col, or diag, end the game and display a message box saying who has won the game and let the players play again. |
| A simple game is over with a tie | 5.2 | App, Board | checkSPlacedPoint, checkOPlacedPointm checkForSimpleWin, getPlayerPoints, noOpenSpaces, getGeneralWinner | Complete | Assuming the game mode is set to simple, when the board is completely filled up and no one has made an SOS, display a message box that there is a tie and allow players to play again. |
|  | … |  |  |  |  |
| 6 Make a valid move in a simple game | 6.1 | App, Board | Clicked, buildMainGame, getPeice, placePiece, updateCurrentPlayerText, checkSPlacedPoint, | Complete | Assuming a valid spot exists and the game is not over, when the user selects a valid button and the game mode is set to general, place their token in the selected spot and update board information. |
| Make an invalid move in a simple game | 6.2 | App, Board | Clicked, buildMainGame, getPeice, placePiece, updateCurrentPlayerText, | Complete | Assuming there are still valid spots on the board, and the game mode is set to general, if the user selects a spot that is not available, already taken by themselves or the other player, do not update anything on the board and instead display an error message and allow the user to select a different available spot |
|  | … |  |  |  |  |
| 7 A general game is over and there is a winner | 7.1 | App, Board | checkSPlacedPoint, checkOPlacedPointm checkForSimpleWin, getPlayerPoints, noOpenSpaces, getGeneralWinner | Complete | Assuming the game mode is set to general, when the last available spot is filled up on the board, the game is over and will display who scored more points. |
| A general game is over and there is a tie |  | App, Board | checkSPlacedPoint, checkOPlacedPointm checkForSimpleWin, getPlayerPoints, noOpenSpaces, getGeneralWinner | Complete | Assuming the game mode is set to general, if all the spots on the board are filled up and either neither player scored, or they scored the same amount, end the game and display there was a tie. |

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 |
| 5 | A simple game is over |
| 6 | Make a move in a general game |
| 7 | A general game is over |

4.1 Automated tests directly corresponding to some acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID** | **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 Choose valid board size | 1.1 | Board, UnitTestClass | Test\_ChooseBoardSize | Given a valid board size update input of 10, update the board class and place a piece at the bottom right corner of the board. Expect the bottom right corner of the board should have a token and the board size logic should be 10. |
| Choose an invalid board size | 1.2 | Board, UnitTestClass | Test\_ChooseBoardSize | Given a board size update input of 15, update the board class and place a token at the bottom right corner of the board, since the board size is 11, the board size should only be updated to 10. Expect there to be a token at the bottom right corner, and the board size logic to be 10. |
|  | … |  |  |  |
| 2 Choose the game mode of a chosen board | 2.1 | Board, UnitTestClass | Test\_ChooseVaildGameMode | Given the input of selecting the General game mode, we can expect the output to be old game mode of simple and new game mode of general |
| Choose invalid game mode of the board | 2.2 | Board, UnitTestClass | Test\_ChooseInvalidGameMode | Given the input of selecting an invalid game mode called “Invalid” the board does not update the game mode. Expect the output of old game mode to be Simple and current game mode to be simple as well. |
|  | … |  |  |  |
| 3 Start a new game of chosen board size and game mode | 3.1 | Board, UnitTestClass, App | Test\_StartGeneralGame  Test\_StartSimpleGame | Given input of selecting board size, and selecting game mode, after selecting the start button, expect the output of gameStarted to be true, the board is of the proper size and isn’t an empty list, and game mode is set to either simple or general. |
|  | … |  |  |  |
| 4 Make a valid move in a simple game | 4.1 | Board, UnitTestClass | Test\_MakeSimpleGameMove | Given the game has already started and the game mode is simple, select an empty spot on the board and place a token there, we can expect the board at that location to have the token that was just placed down, and the player of who placed it down be equal to the correct player. |
| Make an invalid move in a simple game | 4.2 | Board, UnitTestClass | Test\_MakeInvalidSimpleGameMove | Given the game has already started and the game mode is set to simple, player 2 selects a spot that already has a token inside of it, we can expect the piece originally in that spot should still be there, and the player number of who placed it should not change. |
|  | … |  |  |  |
| 5 A simple game is over | 5.1 | Board, UnitTestClass | Test\_SimpleGameIsOverWin | Given the game mode is set to simple, after every token we should check if an S O S has been made, and we should check for a simple game mode win from the board class, we should expect to see the winner is equal to the person who created the SOS |
|  | 5.2 | Board, UnitTestClass | Test\_SimpleGameIsOverTie | Given the game mode is set to simple, after every turn check to see if an SOS has been created , if the board is filled up and the score is equal we can expect to see the winner is nobody. |
|  | … |  |  |  |
| 6 Make a valid move in a general game | 6.1 | Board, UnitTestClass | Test\_MakeGeneralGameMove | Given the game has already started and the game mode is general, select an empty spot on the board and place a token there, we can expect the board at that location to have the token that was just placed down, and the player of who placed it down be equal to the correct player. |
| Make an invalid move in a simple game | 6.2 | Board, UnitTestClass | Test\_MakeInvalidGeneralGameMove | Given the game has already started and the game mode is set to general, player 2 selects a spot that already has a token inside of it, we can expect the piece originally in that spot should still be there, and the player number of who placed it should not change. |
|  | … |  |  |  |
| 7 A simple game is over | 7.1 | Board, UnitTestClass | Test\_GeneralGameIsOverWin | Given the game mode is set to general, after every spot on the board has been filled up with either an S or an O token, the player with the most points is determined to be winner. |
|  | 7.2 | Board, UnitTestClass | Test\_GeneralGameIsOverTie | Given the game mode is set to general, after every turn check to see if an SOS has been created, if the board is filled up and the score is equal we can expect to see the winner is nobody. |

1. **Describe how the class hierarchy in your design deals with the common and different requirements of the Simple Game and the General Game**? **(4 points)**

* Since the only thing different about the SOS game between the two game modes is how a person wins, there isn’t too much of a difference between how a simple and general game is played in my code. After every button click, it is checked what the current game mode is set to, and it is checked based on the game mode selected, if the game should be over or not. In my GUI class, the first check made is if the game is simple, and then we see if the conditions for a simple game to be over has been complete, i.e the first person to score an SOS has done so. If it has the game is over and lets the players restart, if the game requirements have not been met, we just continue to the next round. If the game mode is set to General, we first check after a button click has been accepted, if the board has been filled up. If it has we display the winner if one player has scored more points than the other, or if they scored the same number of points we display that there was a tie. This logic uses the board class inside of the GUI class to verify if there is a winner and uses the GUI class to show the player who the winner is if there is one.