**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**.

Github Link : https://github.com/gwb9h3/SOS-Sprint-3

|  |  |  |
| --- | --- | --- |
| 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.**

**Panopto Link: https://umsystem.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=a3b5801a-eea8-4aa5-99c4-b21e0058ed7e**

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 |
| GameView.java | Production | 209 |
| GameController.java | Production | 183 |
| Main.java | Production | 15 |
| BoardData.java | Production | 118 |
| GameControllerTest.java | Test Code | 65 |
| GameModeSelecitonTest.java | Test Code | 40 |
| Total | | 630 |

**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 and Name** | **AC ID** | **Class Name(s)** | **Method Name(s)** | **Status (complete or not)** | **Notes (optional)** |
| 1.Choose a board size | 1.1 | GameView & Game Controller | Stored under the gameView.getBtnEnter action listener | Complete | More classes will be added to store the buttons data in an array when it is necessary to find SOS points. Therefore this code may get moved to a different file and just be called inside this class at that point. |
|  | 1.2 | GameView & Game Controller | Stored under the gameView.getBtnEnter action listener | Complete | Same as previous |
| 2. Choose a game mode | 2.1 | GameView & Game Controller | Stored as radio buttons in the top pannel of GameView and functionality is found in the the gameView.getBtnEnter action listener | Semi-Complete | The game will not start unless the game mode is chosen but the rules for the game mode have not been implemented as the SOS points have not been implemented |
|  | 2.2 | GameView & Game Controller | Stored as radio buttons in the top pannel of GameView and functionality is found in the the gameView.getBtnEnter action listener | Semi-Complete | The game will not start unless the game mode is chosen but the rules for the game mode have not been implemented as the SOS points have not been implemented |
| 3. Start a new game of the chosen board size and game mode | 3.1 | GameView & Game Controller | In gameView the enter button starts it and the code is stored in the the gameView.getBtnEnter action listener | Complete | The game starts and the board is initialized when started but the backend points trackers and rules need to be implemented when the SOS trackers are required |
|  | 3.2 | GameView & Game Controller | In gameView the enter button starts it and the code is stored in the the gameView.getBtnEnter action listener | Complete | The game starts and the board is initialized when started but the backend points trackers and rules need to be implemented when the SOS trackers are required |
| 4. Make a move in a simple game | 4.1 | Game Controller | The action is performed in the gameView.getBtnEnter action listener | Semi-Complete | Move record will be kept once the matrix code is created in a separate file. Placement works perfectly fine though |
|  | 4.2 | Game Controller | The action is performed in the gameView.getBtnEnter action listener | Complete | Invalid moves cannot be done because the button to place a S or O does not work once an S or O is already in a tile. |
|  | 4.3 | Game Controller | The action is performed in the gameView.getBtnEnter action listener | Complete | Turn switching is fully functional and is shown below the board |
|  | 4.4 | Game Controller | The action is performed in the gameView.getBtnEnter action listener | Not Complete | The board still allows for moves to be done once a SOS is formed due to the points not yet being tracked in the game |
| 5. Make a move in a general game | 6.1 | Game Controller | The action is performed in the gameView.getBtnEnter action listener | Semi-Complete | Move record will be kept once the matrix code is created in a separate file. Placement works perfectly fine though |
|  | 6.2 | Game Controller | The action is performed in the gameView.getBtnEnter action listener | Complete | Invalid moves cannot be done because the button to place a S or O does not work once an S or O is already in a tile. |
|  | 6.3 | Game Controller | The action is performed in the gameView.getBtnEnter action listener | Complete | Turn switching is fully functional and is shown below the board |
|  | 66.4 | Game Controller | The action is performed in the gameView.getBtnEnter action listener | Complete | The board does not allow for more moves once a general game is complete because all of the tiles allow for only one move to be done before becoming solid. |

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.

You are required to use free ChatGPT version to create 2 unit tests using ChatGPT. You also need to ensure that that the generated unit tests are correct, and refined them if not. At the end of the submission, provide the screenshots of your chatgpt prompts and answers, along with errors chatgpt made and you had to correct. You may also use LLMs hosted locally. 2 points will be deducted if no screenshots provided.

|  |  |
| --- | --- |
| **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 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. Choose a board size | 1.1 | GameControllerTest.java | testInvalidBoardSizeInputOutOfRange() | Test inputs a board size of 2 and it should return null |
|  | 1.2 | GameControllerTest.java | testValidBoardSizeInput() | Test inputs a board size of 5 and should return a list of 25 buttons |
| 2. Choose the game mode of a chosen board | 2.1 | GameModeSelectionTest.java | testSimpleGameModeSelected() | Test imitates selecting the simple game mode radio button and checks the getter function to see if it is reflecting being selected |
|  | 2.2 | GameModeSelectionTest.java | testGeneralGameModeSelected() | Test imitates selecting the general game mode radio button and checks the getter function to see if it is reflecting being selected  (HAD TO BE EDITED MANUALLY DUE TO CHANGES IN SOURCE CODE) |

4.2 Manual tests directly corresponding to some acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID and Name** | **Acceptance Criterion ID** | **Test Case Input** | **Test Oracle (Expected Output)** | **Notes** |
| 3. Start a new game of the chosen board size and game mode | 3.1 Trying to start the game with invalid board size | Input board size of 2 | Game does not initialize a board and sends an error message |  |
|  | 3.2 Trying to start the game with a valid board size and invalid game selection | Board size of 4 selected but no game mode selected | The game does not initialize a board and sends an error message |  |
|  | 3.3 Starting a game with a valid game mode and valid board size | Game mode of Simple or general and board size  3<=x<=10 | The game initializes the board of given size and game mode |  |
| 4. Making a move in a simple game | 4.1 Valid Move Placement | Place an S or O in an empty tile on the board | The game will put the S or O in that tile |  |
|  | 4.2 Invalid Move Handling | Attempt to place an S or O in a full tile on the board | The game will not allow the button to be pressed to fill the tile as it is already full |  |
|  | 4.3 Turn switch after valid move | After a valid move from either player | The game will switch the text at the bottom and give control to the opposite player |  |
| 5. A simple game is over | 5.1 A Player has Gotten an SOS | Game mode is initialized with proper size and simple game is chosen. Blue player places S in top left, Red player Places O in top middle, Blue player places S in top right. | The game should recognize the SOS that has been placed on the board and send a message that the Blue player has won |  |
|  | 5.2 The Board is Full | The Blue and Red player go back and forth placing an S in each box on a 3x3 board | The game should recognize that the board is full and send a message that the game is a tie |  |
| 6. Making a move in a general game | 6.1 Valid Move Placement | Place an S or O in an empty tile on the board | The game will put the S or O in that tile |  |
|  | 6.2 Invalid Move Handling | Attempt to place an S or O in a full tile on the board | The game will not allow the button to be pressed to fill the tile as it is already full |  |
|  | 6.3 Turn switch after valid move | After a valid move from either player | The game will switch the text at the bottom and give control to the opposite player |  |
|  | 6.4 Prevent further moves after a board is full | After all tiles have filled up on the board | The game will no longer allow any new moves to be placed |  |
| 7. A general game is over | 7.1 The Board is full | The players go back and forth placing a S and an O on a 3x3 board with at least one valid SOS until the board is full | The game should recognize that the board is full and announce the player with the most SOS points wins |  |
|  | 7.2 The game is a Tie | The players go back and forth placing a S and an O on a 3.3 board where there is no valid SOS’s or they tie in points when all boxes are full. | The game should recognize that the board is full and announce that the game is a tie. |  |

4.3 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, Input is non-numerical | abc | null | GameControllerTest.java | testInvalidBoardSizeInputNonNumeric() |
|  |  |  |  |  |

1. **Describe how the class hierarchy in your design deals with the common and different requirements of the Simple Game and the General Game**? **At least 1/2 of a page is required.** **(4 points)**

For my implementation of the SOS game there are a set of radio buttons at the top of the program for the players to choose whether they are playing either a Simple game or they are playing a General Game. This is an easy way for players to pick what mode they would like to enjoy and the game does not allow a board to be created until one of these two options have been chosen. All of the code in my program is the same between a Simple Game and a General Game because the only rule that differs between the two options is whether the program should stop after a SOS has been created or if the program should stop when all of the tiles have been filled. Therefore I only had to create two functions to test whether a Simple Game has been completed and if a General Game has been completed. Both of these functions are implemented inside the code right next to each other and only run if their individual radio button is selected. This is the simple way that I was able to keep a hierarchy for my code as these two game modes inherit all of the shared rules/functionality that is needed between the two modes. This saves on lines of code and makes the program much more simple.

A black and white screen with white text

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a black and white screen

Description automatically generated