**Project Sprint #2**

Implement the following features of the SOS game: (1) the basic components for the game options (board size and game mode) and initial game, and (2) S/O placement for human players ***without*** checking for the formation of SOS or determining the winner. The following is a sample interface. The implementation of a GUI is strongly encouraged. You should practice object-oriented programming, making your code easy to extend. It is important to separate the user interface code and the game logic code into different classes (refer to the TicTacToe example). xUnit tests are required.

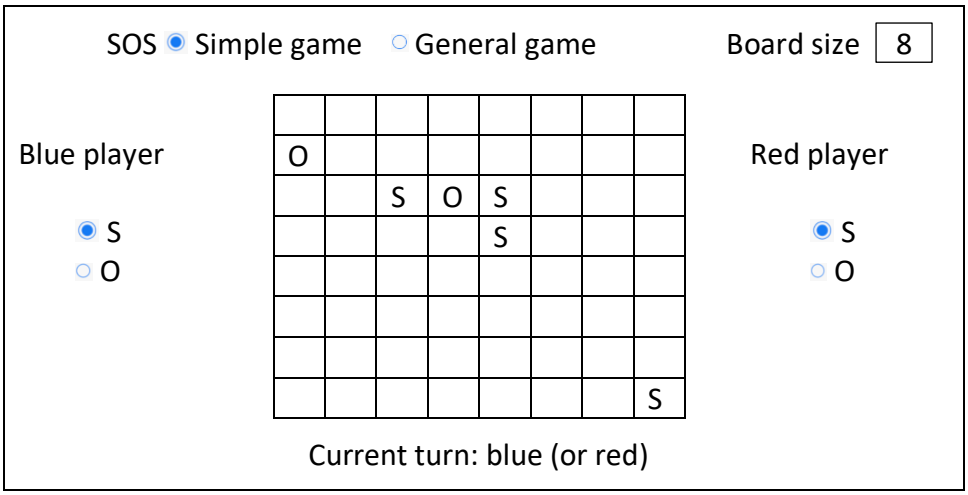


Figure 1. Sample GUI layout of the Sprint 2 program

**Deliverables:**

**1. Demonstration (8 points)**

Submit a video of no more than three minutes, clearly demonstrating that you have implemented the required features and written some automated unit tests. In the video, you must explain what is being demonstrated.

| | **Feature** | |
|---|---|---|
| 1 | Choose board size | |
| 2 | Choose game mode | |
| 3 | Initial game of the chosen board size and game mode | |
| 4 | "S" moves | |
| 5 | "O" moves | |
| 6 | Automated unit tests | |
| … | | |

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

| Source code file name | Production code or test code? | # lines of code |
|---|---|---|
| Board.java | Production code | 66 |
| Console.java | Production code | 55 |
| GUI.java | Gui code | 143 |
| TestConsole.java | Test code | 31 |
| TestEmptyBoard.java | Test code | 47 |

| | | |
|---|---|---|
| TestOplayerMoves.java | Test code | 47 |
| TestSplayerMoves.java | Test code | 54 |
| | Total | 443 |

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

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

Update your user stories and acceptance criteria from the previous assignment and ensure they adequately capture the requirements. Summarize how each of the following 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 |
| 6 | Make a move in a general game |

| 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 | Board() | Board() | complete | |
| 2.Choose the game mode of a chosen board | 2.1 | Board() | getGameMode() | Not complete | |
| 3.Start a new game of the chosen board size and game mode | 3.1<br>3.2 | Board() | getCell()<br>getTurn()<br>initBoard() | complete<br>complete | |
| Make a move in a simple game | 2.1 | Board() | makeMove() | Not complete | |
| | … | | | | |

## 4. 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 |
| 6 | Make a move in a general game |

4.1 Automated tests directly corresponding to the acceptance criteria of the above user stories

| **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 board size | 1.1 empty board | TestEmptyBoard() | testNewBoard() | |
| 2 start new game | 2.1 invalid row | TestEmptyBoard() | testInvalidRow() | Input rowIndex=4, expected null |
| | 2.2 invalid column | TestEmptyBoard() | testInvalidColumn() | Input columnIndex=4, expected null |
| 3 game mode | 2.1 choose between simple and general game | TestEmptyBoard() | testGameModeChocie() | Input simple_game, expected simple_game |
| 4 make a move in simple game | 4.1 valid s player move | TestSplayerMoves() | testSplayerTurnMoveVacantCell() | Input row=col=0 Expected S_player placement |
| | 4.2 illegal s player move in occupied cell | TestSplayerMoves() | testSplayerTurnMoveNonVacantCell() | Row=0, col=0 Expected S_player turn didn't change |
| | 4.3 illegal move outside the board | TestSplayerMoves() | testSplayerTurnInvalidRowMove() testSplayerTurnInvalidColumnMove() | Row=5, column=5, expected turn don't change |
| | 4.4 valid O player move | TestOplayerMoves() | testOplayerTurnVacantCell() | Row=col=0 Expected o_player placement |
| | 4.5 illegal o player move in occupied cell | TestOplayerMoves | testOplayerTurnMoveNonVacantCell() | Row=1, col=0 Expected turn don't change |
| | 4.6 illegal o player move outside the board | TestOplayerMoves | testOplayerTurnInvalidRowMove() testOplayerTurnInvalidColumnMove() | Row=5, col=5, turn don't change, still O |
| | | TestConsole | testEmptyBoard() | Display empty board |
| | | | testNonEmptyBoard() | Display updated board |