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**.

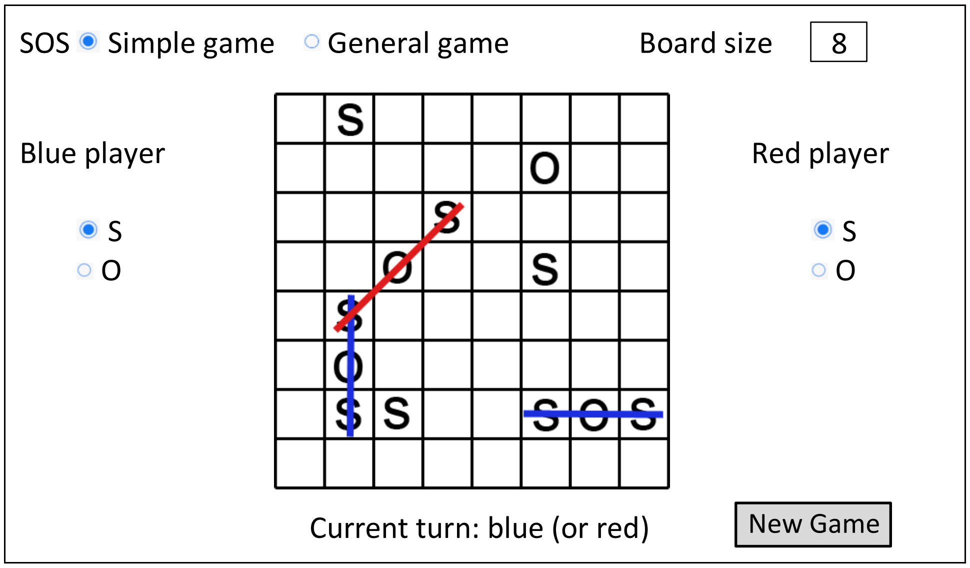

Figure 1. Sample GUI layout of the working program for Sprint 3

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.
   (a) A simple game that the blue player is the winner
   (b) A simple draw game with the same board size as (a)
   (c) A general game that the red player is the winner, and the board size is different from (a)
   (d) A general draw game with the same board size as (c)
   (e) Some automated unit tests for the simple game mode
   (f) Some automated unit tests for the general game mode

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

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

Source code file name	Production code or test code?	# lines of code
Board.java	Production code	566
Console2.java	Production code	120
GUI.java	Gui code	143
TestConsole.java	Test code	47
TestEmptyBoard.java	Test code	48

TestRedplayerMoves.java	Test code	50
TestBlueplayerMoves.java	Test code	59
TestCompleteBoard	Test code	60
	Total	1093

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

3. 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	Board()	Board()
			setRows()
			setColumns()
			getTotalRows()
			getTotalColumns()
				complete	

3.Choose the game mode of a chosen board	3.1	Board()	setGameMode()
			getGameMode()
				complete	

2 start a new game of a chosen size and mode		console	Play()		
4.Make a move in a simple game	4.1 4.2 4.3 4.4 4.5 4.6	Board()	makeMove()	complete	

5 simple game is over	…	Board
		
		Console2
			isDraw()
			simpleCheck()
			isOver()
				completed	

6 make a move in general game			makeMove()	In progress	
7 general game is over			generalCheck()	In progress	

Number	Test Input	Expected Result	Class Name of the Test Code	Method Name of the Test Code

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
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	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()	Board initial to empty cells
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	3.1 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	TestBlueplayerMoves()	`testBlueplayerTurnMoveVacantCell()`	Input row=col=0 Expected S_player placement
	4.2 illegal s player move in occupied cell	TestBlueplayerMoves()	`testBlueplayerTurnMoveNonVacantCell()`	Row=0, col=0 Expected blue player turn don’t change
	4.3 illegal move outside the board	TestBlueplayerMoves()	`testBlueplayerTurnInvalidRowMove()` `testBlueplayerTurnInvalidColumnMove()`	Row >4 not valid Col>4 not valid
	4.4 valid O player move	`TestRedplayerMoves()`	`testRedplayerTurnVacantCell()`	Row=col=0 Expected O_player placement
	4.5 illegal o player move in occupied cell	`TestRedplayerMoves`	`testRedplayerTurnMoveNonVacantCell()`	Row=1, col=0 Expected turn don’t change

	4.6 illegal o player move outside the board	TestRedplayerMoves	testRedplayerTurnInvalidRowMove() testRedplayerTurnInvalidColumnMove()	Row=5, col=5, turn don’t change, still O
		TestConsole	testEmptyBoard()	Display empty board
			testNonEmptyBoard()	Display updated board

		TestCompleteBoard	testBlueWon()
			
			testRedWon()
			
			testDraw()
				Expected blue won
				Expected red won
				Expected a draw

5. **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)**

I created GeneralGame class which extends Board class. They share the same methods and variables, however they differ on the check winner method. I didn’t finish working on GeneralGame class for this sprint.