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**.

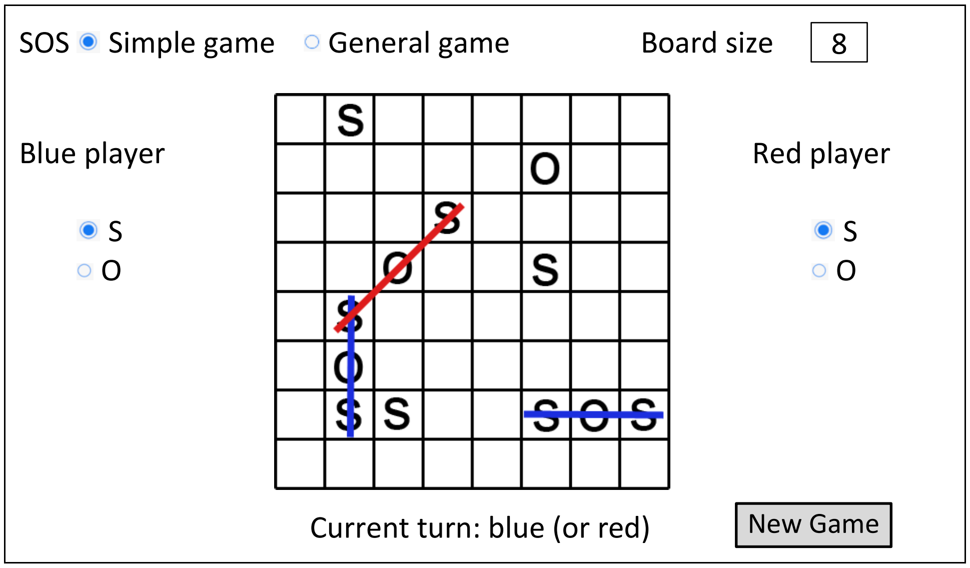

Figure 1. Sample GUI layout of the working program for Sprint 3

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	232
GameFrame.java	Production code	499
GeneralBoard.java	Production code	76
BoardTest.java	Test code	271
	Total	1078

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	setBoardSize	Complete	
	1.2	GameFrame	startGameButtonActionPerformed	Complete	Needs decoupled
	1.3	Board	setBoardSize	Complete	The default board size is set in the class variable boardSize
2 Choose game mode of chosen board	2.1	Board	setGameType	Complete	
	2.2	Board	setGameType	Complete	
	2.3	Board	setGameType	Complete	The default game mode is set in the class variable gameType.
3 Start a new game of the chosen board size and game mode	3.1	Board	initializeGame	Complete	Sets class variable inProgress to 1
	3.2	Board	initializeGame	Complete	Sets class variable inProgress to 1
4 Make a move in a simple game	4.1	Board	makeMove	Complete	
	4.2	Board	makeMove, isEmpty	Complete	makeMove calls isEmpty to ensure cell not occupied
	4.3	JPanel1		Complete	Not sure which method does it, but only the jpanel will allow a letter to be placed on it
	4.4	Board	makeMove	Complete	
	4.5	Board	makeMove, isEmpty	Complete	makeMove calls isEmpty to ensure cell not occupied

	4.6	JPanel1		Complete	Not sure which method does it, but only the jpanel will allow a letter to be placed on it
5 A simple game is over	5.1	Board	getWinner, setWinner, isFull, isSOS, isGameOver, setIsGameOver, getIsGameOver	Complete	
	5.2	Board	getWinner, setWinner, isFull, isSOS, isGameOver, setIsGameOver, getIsGameOver	Complete	
	5.3	Board	getWinner, setWinner, isFull, isSOS, isGameOver, setIsGameOver, getIsGameOver	Complete	
6 Make a move in a general game	6.1	Board	makeMove	Complete	
	6.2	Board	makeMove, isEmpy		
	6.3	JPanel		Complete	
	6.4	Board	makeMove	Complete	
	6.5	Board	makeMove, isEmpty	Complete	
7 A general game is over	7.1	GeneralBoard	getWinner, setWinner, isFull, isSOS, isGameOver, setIsGameOver, getIsGameOver	Complete	
	7.2	GeneralBoard	getWinner, setWinner, isFull, isSOS, isGameOver, setIsGameOver, getIsGameOver	Complete	
	7.3	GeneralBoard	getWinner, setWinner, isFull, isSOS, isGameOver, setIsGameOver, getIsGameOver	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
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	BoardTest	testSetBoardSize	Input: 6 Output: passed test, board size set to 6
2. Choose game mode of	2.1	BoardTest	testSetGameTypeSimple	Input: 0 Output: passed test, game type set to simple {0}
	2.2	BoardTest	testSetGameTypeSimpleNotChosen	Input: >1

a chosen board				Output:
3. Start a new game of the chosen board size and game mode	3.1	BoardTest	testInitializeGameSimple	Input: Simple GameType,BoardSize Output: passed, new simple game initialized
	3.2	BoardTest	testInitializeGameGeneral	Input: General Game Type,BoardSize Output: passed, new general game initialized
4. Make a move in a simple game	4.1	BoardTest	testMakeMoveSRed, testMakeMoveORed	Input: Letter S/O Output: passed, make move to specified cell for Red
	4.2	BoardTest	testInvalidMoveRed	Input: Letter O on occupied cell Output: passed, cell not changed, turn not changed
	4.4	BoardTest	testMakeMoveSBlue, testMakeMoveOBlue	Input: Letter S/O Output: passed, make move to specified cell for Blue
	4.5	BoardTest	testInvalidMoveBlue	Input: Letter O on occupied cell Output: passed, cell not changed, turn not changed
5 A simple game is over	5.1	BoardTest	testSimpleBluePlayerWin	Input: SOS in a Row Output: Blue or Red player wins with SOS in simple game
	5.3	BoardTest	testSimpleDraw	Input: a full board with no SOS Output: passed
7 A general game is over	7.1	BoardTest	testGeneralRedPlayerWin	Input: A full board with Red having more SOS's Output: passed
	7.2	BoardTest	testGeneralDraw	Input: An full board with neither player having more SOS's Output: passed

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**
1 Choose a board size	1.2	A value greater than 8	A Message box with an error message	Need to figure out how to add this to automated test

4 Make a move in a simple game	4.3	Attempt to place an S/O outside of the board	No letter placed, nothing changed	
	4.6	Attempt to place an S/O outside of the board	No letter placed, nothing changed	
5 A simple game is over	5.1	Tested both a Blue and A Red win from multiple angles	The correct winner was given	
	5.3	Filled the board with no SOS's in a row	Correctly given a draw	
7 A general game is over	7.1	Tested that both the Red player and Blue player would win if one had more SOS's	The correct winner was given	
	7.2	Filled the board with no SOS's for either player	Correctly given a draw	

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**

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)

Most of the methods in the parent class, Board, were the same. I had to add two new properties to General Board, the inherited class, which were redScore and blueScore. RedScore and blueScore were used to keep track of the number of SOS's each player had before the board was full. I added four new methods that dealt with the scoring in a general game: addRedPoint, addBluePoint, getRedScore, and getBlueScore. I overrode two of the parent methods: setWinner and makeMove. SetWinner was changed to compare the scores once the board was full and set the winner property with the winner of the game. I overrode makeMove and added code to to add a point to the Red player or Blue player when an SOS was created by the player whose turn it was. It then called on the setWinner method to declare the winner and end the game.