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**.

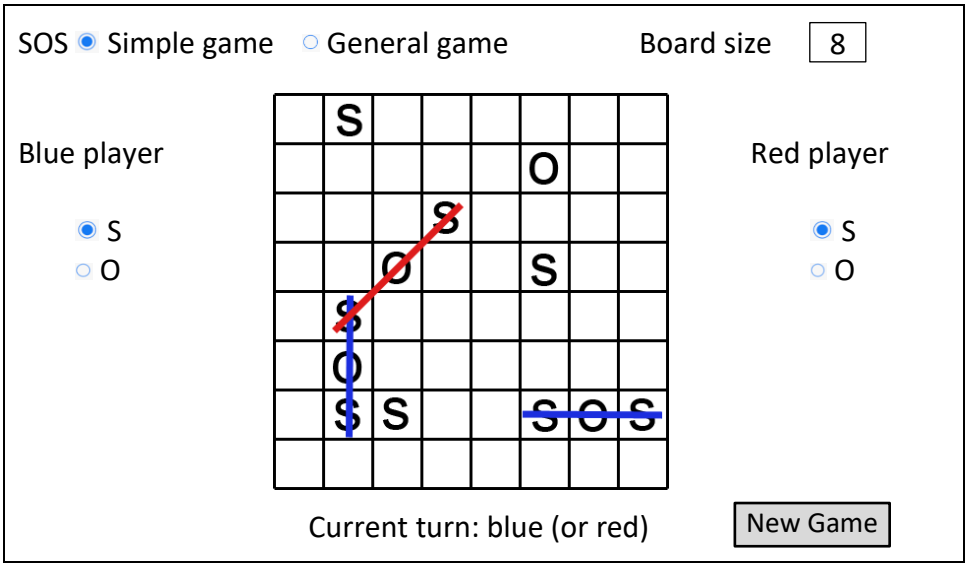

Figure 1. Sample GUI layout of the working program for Sprint 3

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
	Total	

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	AC ID	Class Name(s)	Method Name(s)	Status (complete or not)	Notes (optional)

1. Choose a board size	1.1	AC 1.1 Selecting board size
		Given that the player has started the SOS game and is presented with the initial game menu.
		When the player selects the "Choose board size" option from the menu
		Then the player should be presented with a screen that displays a range of available board sizes, with the smallest and largest sizes clearly labeled and highlighted.
			The class **board** initialized the size of board, the cell, etc
			
			`public void actionPerformed(ActionEvent e)`
			
			which is in the class GUI is the function that makes sure that the right board size is chosen by the player. If it's not the case a message will be prompted
			
			There's also the function that check if the size is valid or not which is called:
			
			`public boolean validSize(int size)`
				Completed	

	1.2	AC 1.2 Changing board size during the game Given that the player has selected a board size and begun a game. When the player attempts to switch to a different board size during the game Then the game should display a confirmation dialog or prompt, informing the player that their progress will be lost	This part is not implemented yet but it'll be inside the initboard like an error message. An I'll have to include a replay button just like the New game on the bottom.	Incompleted	The replay button is not ask in this sprint that's why this AC is not implemented yet.

		if they switch to a new board size, and asking them to confirm their choice before proceeding.			

2. Choose the game mode of a chosen board	2.1	AC 2.1 Selecting the game mode of a chosen board
		Given a player has selected a board for the game
		When the player is prompted to choose a game mode,
		Then the player must be presented with 2 options: simple game, and general game.
		The player must be able to select one of these options, and the game must proceed accordingly based on the mode selected.
			The two-mode are written as Button and only one can be chosen in the class GUI
			The simple mode is implemented as
			
			`JRadioButton simpleGame = new JRadioButton("Simple Game");`
			
			The general mode is implemented as:
			
			`JRadioButton generalGame = new JRadioButton("General Game");`
			
			There's also a class called simple where the specificity of what's going to happen in the simple mode is going to be located like checking if there's a win or not
				Completed	

3. Start a new game of the chosen board size and game mode	3.1	AC 3.1 Start a new game with the chosen settings Given a player has selected a board size and game mode When the player chooses to start a new game Then the game must implement the rules specified for the mode chosen such as time limit, or reduced lives for incorrect moves.	This part is implemented by the button called New game so when we want to start a new game we just choose the board size and the game mode and click on it the play. And the function that allows that is in the class GUI as is called: `public void actionPerformed(ActionEvent e)`	Completed	

4. Make a move in a simple game	4.1	AC 4.1 Valid move in a simple game.
		Given a player is in an SOS game,
		When the player attempts to make a valid move which is placing "S" or "O" in an unoccupied cell,
		Then the game board should be updated accordingly, and the turn should be passed to the next player.
			This AC is not fully implemented that's why when I run the test code I didn't have an error but a warning. Its implementation is located in the class board and the function doing the job is :
			
			`public boolean makeMove(int row, int column)`
			
			but the turn choice and set up are done by:
			
			`public char getTurn()`
			
			`public void setTurn(char t)`
				Completed	

	4.2	AC 4.2 Invalid move in a simple game.	This part is completed with this function	Completed	

		Given a player is in a SOS game, When the player attempts to make an invalid move like trying to place “S” or “O” in an occupied cell, Then the game board should not be updated, and the player should be prompted to make a valid move. The turn should remain with the same player until a valid move is made.	`public boolean makeMove(int row, int column)`		
5. A simple game is over	5.1	AC 5.1 Simple game is over in a tie Given that all cells on the game board of a simple SOS game are occupied with letters, When there’s no player who has formed an “SOS” sequence, Then the game should end in a tie, and the players should be notified of the tie.	We have the message “it’s a drawn” when there’s no winner: `printStatusBar()`	Complete	
	5.2	AC 5.2 Simple game is over with a Win. Given that a player has formed an “SOS” sequence in a simple game mode, When there are no more empty cells on the game board, Then the game should end, and the player who has formed the “SOS” should be declared the winner.	This part is also in progress just missing the points recorded to stated if we have a winner or not. In the simple mode, the function that’s doing that is: `public void checkForWin()`	In-Progress	
6. Make a move in a general game	6.1	AC 6.1 Valid move in a general game Given a player is in a general SOS game. When the player makes a move by placing either an “S” or an “O” in an empty cell. Then the game should correctly update the	The move are implemented through the function: `public void actionPerformed(ActionEvent e)`	Complete	

		grid with the player's move, and the turn should be passed to the next player.			
	6.2	AC 6.2 Invalid move in a general game Given a player is in a general SOS game, When the player attempts to make an invalid move, such as placing a letter in a cell that is already occupied or attempting to place a letter outside the bounds of the game board, Then the game board shouldn't be updated, and the player should be prompted to make a valid move. The turn should remain with the same player until a valid move is made.	This part is completed, you can see on the console a message saying that "this cell is already occupied" `public boolean makeMove(int row, int column)`	Completed	
7	7.1	AC 7.1 General game is over in a tie Given that all cells on the game board of a general SOS game are occupied with letters, When there's no player who has formed an "SOS" sequence, Then the game should end in a tie, and the players should be notified of the tie.	This part is implemented but it's not fully functional because the points of each player is not recorded. The function that's doing that is: `printStatusBar()`	In-progress	
	7.2	AC 7.2 General game is over with a Win Given that a player has formed an "SOS" sequence in a general game mode, When there are no more empty cells on the game board, Then the game should end, and the player who has formed the most "SOS" sequences should be declared the winner.	This is also in progress and should be done in the function: `printStatusBar()`	In progress	

	7.3	AC 7.1 General game is over with same amount of "SOS" sequence. Given that all cells on the game board of a general SOS game are occupied with letters, When both players have formed the same number of "SOS" sequences, Then the game should end in a tie, and the players should be notified of the tie.			

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	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	1.1	Testboard	`public void testInitBoard()`	
	1.2	incomplete		
2	2.1	TestGUI	`public void test()`	
3	3.1	TestGUI	`public void test()`	
4	4.1	Testboard	`public void testMakeMove()`	
	4.2	TestBoard		

6	6.1	TestBoard and TestGeneral	`public void testMakeMove()`
			
			`public void test()`	

	6.2	TestBoard and TestGeneral		
7	7.1	In-progress		
	7.2	In-progress		

4.2 Manual tests directly corresponding to some acceptance criteria

User Story ID	Acceptance Criterion ID	Test Case Input	Test Oracle (Expected Output)	Notes
1	1.1			
	1.2			
	…			
2	2.1			
	…			

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)

In the design hierarchy, both Simple Game and General Game inherit from the Board class. The Board class contains the basic functionalities and attributes that are common to both games, such as the board size, number of moves, and the current game state.

However, as the Simple Game and General Game have different requirements, they have different implementations for the checkForWin() method in their respective classes.

In the Simple Game class, the checkForWin() method checks if either player has achieved the minimum number of points required to win the game. If so, it sets the game state to the appropriate winning state. Otherwise, if the maximum number of moves have not been reached, it sets the game state to PLAYING, or else it sets the game state to DRAW.

In the General Game class, the checkForWin() method checks if the maximum number of moves have been made or not. If not, the game state is set to PLAYING. Otherwise, it checks the number of points each player has, and sets the game state accordingly, i.e., B_WON, R_WON, or DRAW.

Therefore, the class hierarchy in the design deals with the common and different requirements of the Simple Game and the General Game by having a common base class that contains shared attributes and functionality,

while allowing the derived classes to implement their unique requirements through overriding methods or adding new methods.