Carolyn Knapp
CS 449
SOS Game - Sprint 2
Due: 3/10/2024

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.

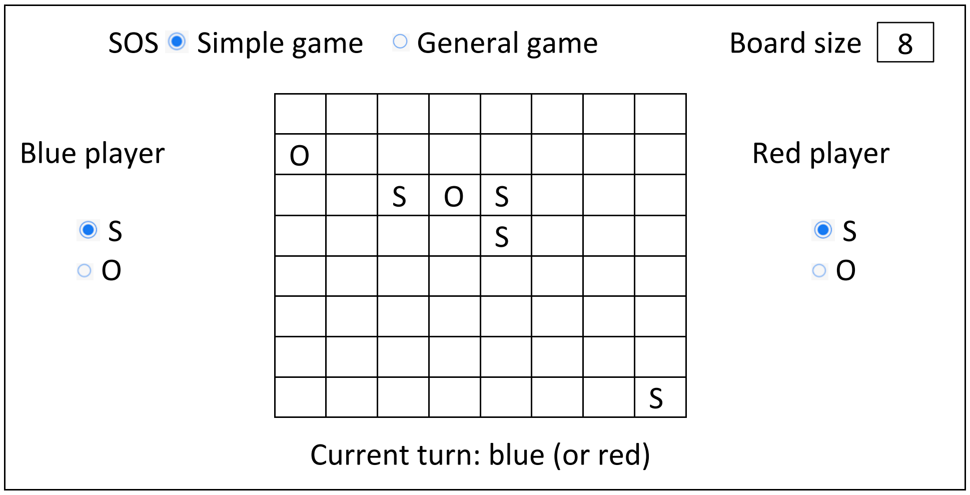

Figure 1. Sample GUI layout of the Sprint 2 program

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

sprint2.product.Board.java		102
sprint2.product.GUI.java		309
sprint2.test.BoardTest.java		83
	Total	494

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	1.1	`Board`	`setBoardSize(int size)`	Complete	
	1.2	`Board`	`setBoardSize(int size)`	Complete	
2	2.1	`Board`	`setGameMode(Boolean mode)`	Complete	
	2.2	`Board`	`setGameMode(Boolean mode)`	Complete	
	2.3	`Board`	`setGameMode(Boolean mode)`	Complete	
3	3.1	`Board`	`Board()`	Complete	
	3.2	`Board`	`Board(int size)`	Complete	
	3.3	`Board`	`Board(int size, boolean mode)`	Complete	
4	4.1	`Board`	`makeMove(int row, int column, char letter)`	Complete	
	4.2	`Board`	`makeMove(int row, int column, char letter)`	Complete	
	4.3	`Board`	`makeMove(int row, int column, char letter)`	Complete	
5	5.1			Not	
	5.2			Not	
	5.3			Not	
6	6.1			Not	
	6.2			Not	
	6.3			Not	
7	7.1			Not	
	7.2			Not	
	7.3			Not	

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	1.1	`BoardTest`	`testSetAndGetBoardSize()`	Input size = 5 and setBoardSize(size); Expected: size = .getBoardSize
	1.2	`BoardTest`	`testDefaultBoardSize()`	Expected: .getBoardSize = 3
2	2.1	`BoardTest`	`testSetAndGetGameMode()`	.setGameMode to true for simple game; Expected: .getGameMode = true
	2.2	`BoardTest`	`testSetAndGetGameMode()`	.setGameMode to false for general game; Expected: .getGameMode = false
3	3.1	`BoardTest`	`testDefaultConstructor()`	Expected: .getBoardSize = 3, .getGameMode = true, and .getCurrentPlayer = 'B'
	3.2	`BoardTest`	`testConstructorWithSize()`	Input size = 5 Expected: .getBoardSize = 5, .getGameMode = true, and .getCurrentPlayer = 'B'
	3.3	`BoardTest`	`testConstructorWithSizeAndMode()`	Input size = 7, mode = false Expected: .getBoardSize = 7, .getGameMode = false, and .getCurrentPlayer = 'B'
4	4.1	`BoardTest`	`testMakeMove()`	Input row = 0, column = 0 and letter = 'X'; Expected .getCell(0,0) = 'X'. .getCurrentPlayer = 'R', .makeMove(1,1,'O') = true, .getCell(1,1) now = 'O'

				.getCurrentPlayer now = 'B'
	4.2	`BoardTest`	`testMakeMoveOnInvalidCell()`	Input row = -1, column = 0, and letter = 'X' Expected: false for cell out of bounds
	4.3	`BoardTest`	`testMakeMoveOnInvalidCell()`	Input row = 0, column = 0, and letter = 'X' Then try to makeMove(0,0,'O') Expected: false for cell already occupied
5	5.1			
	5.2			
	5.3			
6	6.1			
	6.2			
	6.3			
7	7.1			
	7.2			
	7.3			

4.2 Manual tests directly corresponding to the acceptance criteria of the above user stories

User Story ID and Name	**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 of the above user stories

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