Project Sprint #4

Implement all the features that support a player (**human or computer**) to play a simple or general SOS game against another player (**human or computer**). The minimum features include **choosing human or computer for red and/or blue players**, **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 computer component must be able to play complete simple and general games. You are encouraged to consider basic strategies for winning simple or general games (e.g., against a poor human player). Optimal play is not required.

The following is a sample GUI layout. You should use a class hierarchy to deal with the computer opponent requirements. If your current code has not yet considered class hierarchy, it is time to refactor your code.

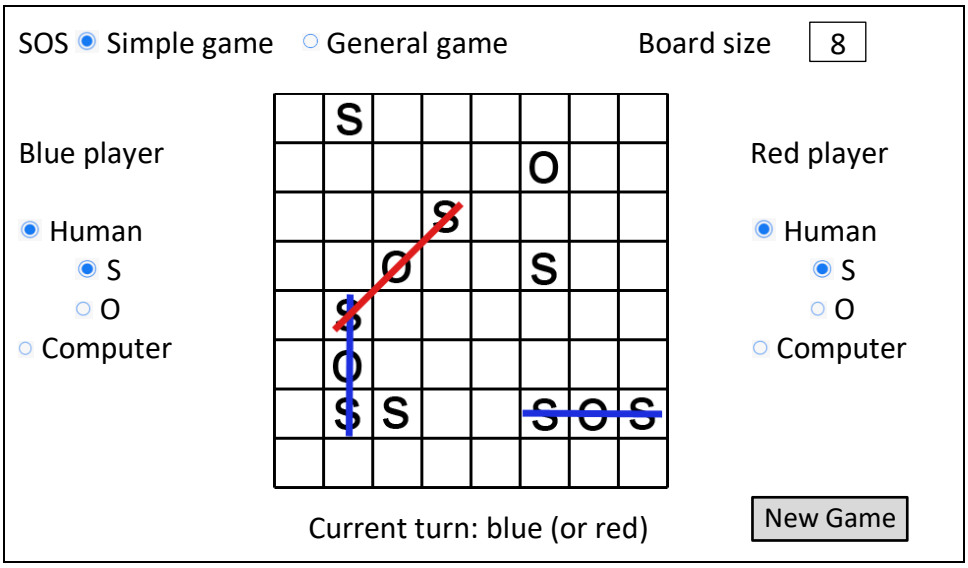

Figure 1. Sample GUI layout of the working program for Sprint 3

Total points: 24

1. **Demonstration (8 points)**

   Submit a video of no more than five minutes, clearly demonstrating that you have implemented the computer opponent features and written some automated unit tests.
   1) A complete simple game where the blue player is a human, the red player is the computer, and there is a winner
   2) A complete general game where the blue player is the computer, the red player is a human, and there is a winner
   3) A complete simple game where both sides are played by the computer
   4) A complete general game where both sides are played by the computer
   5) Some automated unit tests for the computer opponent.

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

2. **User Stories for the Computer Opponent Requirements (1 points)**

   - **User Story Template**: As a <role>, I want <goal> [so that <benefit>]

ID	User Story Name	User Story Description	Priority	Estimated effort (hours)
8		As a computer opponent, I need to form SOS, to win simple game	2	
9		As a computer opponent, when the board is full, I need to form more SOS's, to win the general game	3	
10		As a player, I need to choose between human and computer opponent	1	

3. Acceptance Criteria (AC) for the Computer Opponent Requirements (4 points)

Add or delete rows as needed.

User Story ID and Name	AC ID	Description of Acceptance Criterion	Status (completed, toDo, inPprogress)
8	8.1	AC 8.1 <scenario description> Given an empty board and is blue player's computer opponent turn When blue player makes a valid move Then the token is placed and is red player's turn	Complete
	8.2	Given red player turn When computer opponent is chosen and makes a valid move Then the token is placed and is blue player's turn	Complete
	8.3	Given a simple game and is blue player's computer opponent turn When the computer forms SOS Then the computer opponent of blue player won and the game is over	Complete
9	8.4	Given a simple game and is red player's computer opponent turn When the computer forms SOS Then the computer opponent of red player won, and the game is over	Complete
	9.1	Given a general game and is blue player's computer opponent turn When the board is full and blue player's computer opponent forms more SOS than red player's computer opponent Then the computer opponent of blue player won, and the game is over	Complete
	9.2	Given a general game and is red player's computer opponent turn When the board is full and blue player's computer opponent forms more SOS than red player's computer opponent Then the computer opponent of red player won, and the game is over	Complete
	9.3	Given a full board When blue player's score = red player's score or there are no SOS in the board Then the game is a draw	complete

4. Summary of All Source Code (1 points)

Source code file name	Production code or test code?	# lines of code
Board.java	Production	598
SosGui.java	Production	535
TestBluePlayerMoves.java	Test	150
RedPlayerTest.java	Test	118
TestEmptyBoard.java	test	43

Total	1444

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

5. Production Code vs New User stories/Acceptance Criteria (2 points)

Summarize how each of the new user story/acceptance criteria is implemented in your production code (class name and method name etc.)

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() initBoard()	complete	
2 start a new game of a chosen size and mode		Board	getCell() changeTurn()	complete	
4.Make a move in a simple game	4.1 4.2 4.3 4.4 4.5 4.6	Board	makeSmove() makeOmove()	complete	
5 simple game is over	…	Board	isDraw() simpleCheck()	completed	
6 make a move in general game		Board	setBpScore() setRpScore() getRpScore() getBpScore()	complete	
7 general game is over			generalCheck() isFull() isDraw() generalRedCheck() generalBlueCheck() compareScore()	complete	
		SOsGUI	updateBoardSize()	complete	
			setContentPane()	Complete	
			mouseClicked()	Complete	
			mouseEntered()	Complete	
			paintComponent()	Complete	
			drawGridLines()	Complete	
			drawboard()	Complete	
			printStatusBar()	Complete	

6. Tests vs New User stories/Acceptance Criteria (2 points)

Summarize how each of the new user story/acceptance criteria is tested by your test code (class name and method name) or manually performed tests.

6.1 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 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
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
		TestBluePlayerMoves	testEmptyBoard() testBluePlayerTurnisFull()	Display empty board
		TestBluePlayerMove	testIsDraw() testsetBpScore() testgeneralCheckTest() testSimpleTestHor() testSimpleTestVer() testSimpleTestDiag()	Expected blue won Expected red won Expected a draw

			testSimpleTestAntiDiag()	
		testRedPlayer	testsetRpScore() testGetRpScore()	

7. Present the class diagram of your production code (3 points) and describe how the class hierarchy in your design deals with the computer opponent requirements (3 points)?

My program is not done yet, I haven't used class hierarchy so far. My plan is to use a general class which extends Board class, and override.

Board.java
currentGameState: GameState TOTALROWS: int TOTALCOLUMNS: int Turn: String bpScore: int rpScore: int Grid: Cell
Board() initBoard(): void setRows(int): void setColumns(int): void getTotalRows(): int getTotalColumns(): int getCell(int, int): Cell changeTurn(): void makeSmove(int, int): void makeOmove(int, int): void setBpScore(int): void getBpScore(): int setRpScore(): void getRpScore(): int isFull(): Boolean isDraw(): Boolean simpleCheck(String): Boolean generaleRedCheck(int, int, int): int generalBluePlayer(int, int, int): int compareScore(): void getGameState(): GameState

The following classes extend Board class

SosGUI.java
CELL_SIZE: int GRID_WIDTH: int GRID_WIDTH_HALF: int CELL_PADDING: int SYMBOL_SIZE: int SYMBOL_STROKE_WIDTH: int CANVAS_WIDTH: int CANVAS_HEIGHT: int gameBoardCanvas: GameBoardCanvas gameStatusBar: JLabel
SosGui(Board) updateBoardSize(): void setBoardSize(): Boolean setContentPane(): void mouseClicked(MouseEvent): void mouseEntered(MouseEvent): void printStatusBar(): void

TestBluePlayerMoves
testBlueplayerTurnMoveVacantCell() testBlueplayerTurnMoveVacantCellS() testBlueplayerTurnMoveNonVacantCell() testBlueplayerTurnInvalidRowMove() testBlueplayerTurnInvalidColumnMove() testBlueplayerTurnisFull() testsetBpScore() testgetBpScore() testIsDraw() testgeneralCheckTest() testsimpleTestHor() testsimpleTestVer() testsimpleTestDiag() testsimpleTestAntiDiag()

TestRedPlayerMoves
testRedplayerTurnMoveVacantCell() testRedplayerTurnMoveVacantCellS() testRedplayerTurnMoveNonVacantCell() testRedplayerTurnInvalidRowMove() testRedplayerTurnInvalidColumnMove() testRedplayerTurnisFull() testsetRpScore() testgetRpScore() testgeneralCheckTest()

testsimpleTestHor() testsimpleTestVer() testsimpleTestDiag() testsimpleTestAntiDiag()
TestEmptyBoard
testInvalidRow() testInvalidColumn() testNewBoard()