Project Sprint #2

The SOS game is described in CS449HomeworkOverview.docx. You should read the description very carefully.

Your submission must include the GitHub link to your project and you must ensure that the instructor has the proper access to your project. You will receive no points otherwise.

GitHub link: https://github.com/jsb58p/CS449---Software-Engineering-Project

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 required. You should practice object-oriented programming, making your code easy to extend. It is required to separate the user interface code and the game logic code into different classes (refer to the TicTacToe example). xUnit tests are required.

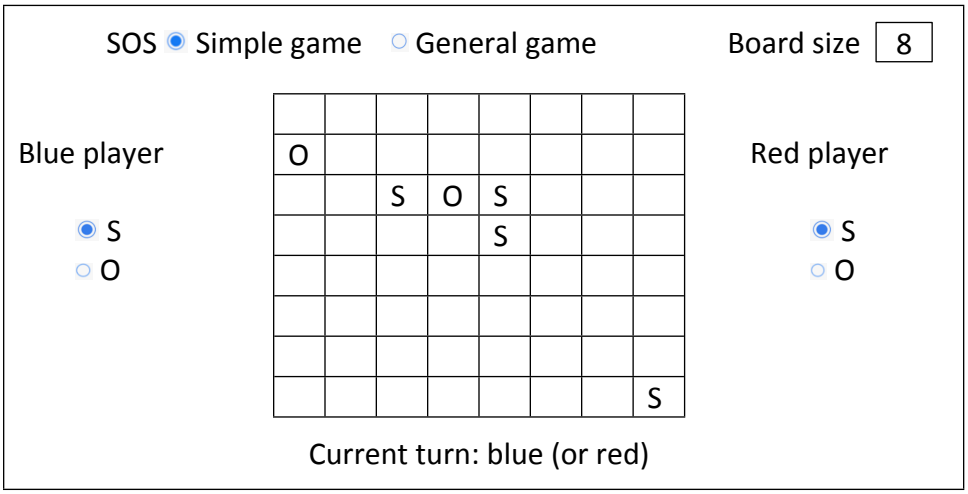


Figure 1. Sample GUI layout of the Sprint 2 program

Deliverables:

1. **Demonstration (8 points)**

   Submit a link to 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. No points will be given without a video link.
   YouTube/Panopto link:
   https://umsystem.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=33236ec1-f365-48e8-8411-b37d01745d2d

	Feature
1	Choose board size
2	Choose game mode
3	Start a new 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
RadioButtonGroup.java	Production code	71
SOSGameGUI.java	Production code	183
SOSGame.java	Production code	133
TestSOSGame.java	Test code	85
	Total	472

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 Choose a board size	1.1	SOSGame	SOSGame(int boardSize, GameMode gameMode) constructor, getBoardSize()	Complete	Board size is set via constructor and can be retrieved.
		SOSGameGUI	start(Stage primaryStage), updateBoardDisplay()	Complete	Spinner in the GUI allows user to select board size 3-10.
2 Choose the game mode of a chosen board	2.1	SOSGame	GameMode enum (SIMPLE, GENERAL), SOSGame(int boardSize, GameMode gameMode) constructor	Complete	GameMode enum defines the two game modes. Constructor accepts and stores the selected mode.
		RadioButtonGroup	RadioButtonGroup(String label1, String label2, Boolean isVertical) constructor, getSelectedButton(), selectFirst()	Complete	Used to create radio buttons for “Simple game” and “General game” with toggle group to ensure only one is selected.
		SOSGameGUI	start(Stage primaryStage), getSelectedButton()	Complete	Instantiates RadioButtonGroup with game mode options. When “New Game” is clicked, passes current selection to SOSGame constructor.

3 Start a new game of the chosen board size and game mode	3.1	SOSGame	SOSGame(int boardSize, GameMode) constructor, getBoard(), getBoardSize()	Complete	Constructor initializes the board array with empty cells determined by spinner value.
		SOSGameGUI	start(Stage primaryStage), updateBoardDisplay()	Complete	"New Game" button event handler uses board size and game mode selections, calls updateBoardDisplay() to show board.
	3.2	N/A	N/A	N/A	Acceptance criteria 3.2 is no longer relevant, board size and game mode is selected when application loads and cannot be unselected.
4 Make a move in a simple game	4.1	SOSGame	isCellEmpty(int row, int col), makeMove(int row, int col, char letter)	Complete	Validates that selected cell is empty before allowing move. Then update the board array with the chosen letter, 'S' or 'O'.
		RadioButtonGroup	getSelectedButton()	Complete	Returns which radio button is selected ('S' or 'O') for the current player.
		SOSGameGUI	updateBoardDisplay()	Complete	When an empty vcell is clicked, updates button display with current player's chosen letter 'S' or 'O' from RadioButtonGroup.
	4.2	SOSGame	getCurrentPlayer(), switchPlayer()	Complete	Track which player's turn it is and switch after each valid move.
		SOSGameGUI	updateBoardDisplay()	Complete	Calls switchPlayer() and updates the "Current Turn" label.
6 Make a move in a general game	6.1	SOSGame	isCellEmpty(int row, int col), makeMove(int row, int col, char letter)	Complete	Validates that selected cell is empty before allowing move. Then update the board array with the chosen letter, 'S' or 'O'.
		RadioButtonGroup	getSelectedButton()	Complete	Returns which radio button is selected ('S' or 'O') for the current player.
		SOSGameGUI	updateBoardDisplay()	Complete	When an empty vcell is clicked, updates button display with current player's

					chosen letter ‘S’ or ‘O’ from RadioButtonGroup.
	6.2	SOSGame	getCurrentPlayer(), switchPlayer()	Complete	Track which player’s turn it is and switch after each valid move.
		SOSGameGUI	updateBoardDisplay()	Complete	Calls switchPlayer() and updates the “Current Turn” label.

Additional note: General game currently uses same implementation as simple game because there are no scoring or game-ending conditions.

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
You are required to use ChatGPT to create at least 2 unit tests. You also need to ensure that the generated user stories are correct, and refine them if not. At the end of the submission, provide the screenshots of your ChatGPT prompts and answers, along with errors ChatGPT made and you fixed. You may also use another LLM, including hosted locally. Points will be deducted if no screenshots are provided.

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	TestSOSGame	testChooseBoardSize()	**Input:** Create SOSGame with size 5 and another with size 7. **Expected Output:** getBoardSize() returns 5 for first game and 7 for second game
2 Choose the game mode of a chosen board	2.1	TestSOSGame	testChooseGameMode()	**Input:** Create SOSGame with GameMode.SIMPLE and another with GameMode.GENERAL **Expected Output:** getGameMode() returns SIMPLE for first game and GENERAL for second game
3 Start a new game of the chosen	3.1	TestSOSGame	testStartNewGameState()	**Input:** Create new SOSGame with size 5 **Expected Output:** All board cells contain ‘ ‘ (empty),

board size and game mode				getCurrentPlayer() returns Player.BLUE
4 Make a move in a simple game	4.1	TestSOSGame	testMakeMoveInSimpleGame()	**Input:** Check if cell (1,1) is empty, make move at (1,1) with 'S'. **Expected Output:** isCellEmpty(1,1) returns true before move, board[1][1] contains 'S' after move.
	4.2	TestSOSGame	testMakeMoveInSimpleGame()	**Input:** Make a move, then call swithcPlayer() **Expected Output:** getCurrentPlayer() changes from BLUE to RED after switch
6 Make a move in a general game	6.1	TestSOSGame	testMakeMoveInGeneralGame()	**Input:** Make a move, then call switchPlayer() **Expected Output:** isCellEmpty(2,2) returns true before move, board[2][2] contains 'O' after move
	6.2	TestSOSGame	testMakeMoveInGeneralGame()	**Input:** Make a move, then call switchPlayer() **Expected Output:** getCurrentPlayer() changes from BLUE to RED after switch

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 Choose a board size	1.1	Change spinner value to 8 and click "New Game"	8x8 grid displays with 64 empty cells	The grid displays as expected.
2 Choose the game mode of a chosen board	2.1	Select "General game" radio button and click "New Game"	Game starts in general mode (Shown in console output)	The console output verifies the chosen game mode is selected.
3 Start a new game of the chosen board size and game mode	3.1	Set board size to 5, select "Simple game", and click "New Game".	5x5 grid displays, "Current Turn: Blue Player" displays in blue text, console output shows game size and simple mode.	The grid displays as expected and the console outputs the game size and mode as expected.
4 Make a move in a simple game	4.1	Blue player selects 'S' radio button and clicks on empty cell at (0,0).	'S' appears in cell (0,0)	The selected letter appears in the correct cell.
	4.2	Blue player makes move.	"Current Turn: Red Player" displays in red text.	The Current Turn label changes as expected.
6 Make a move in a general game	6.1	Red player selects 'O' and clicks empty cell at (2,3).	'O' appears in cell (2,3).	The selected letter appears in the correct cell.

	6.2	Red player makes move.	“Current Turn: Blue Player” displays in blue text.	The Current Turn label changes as expected.

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**

ChatGPT prompts/responses:

"Choose a board size
Choose the game mode of a chosen board
Start a new game of the chosen board size and game mode
Make a move in a simple game
Make a move in a general game
4.1 Create Automated tests directly corresponding to the acceptance criteria of the above user stories." Use JUnit 5. Google style, include comments. No wildcard imports.

```
package sprint2.product;

import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;

/**
 * A custom VBox containing two radio buttons with a shared toggle group.
 */
public class RadioButtonGroup extends VBox {

    private ToggleGroup toggleGroup = new ToggleGroup();
    private RadioButton rb1;
    private RadioButton rb2;

    /**
     * Creates a RadioButtonGroup with two radio buttons labeled as specified.
     * Default vertical layout.
     *
     * @param label1 the label for the first radio button
     * @param label2 the label for the second radio button
     */
    public RadioButtonGroup(String label1, String label2) {
        this(label1, label2, true);
    }


    /**
     * Creates a RadioButtonGroup with two radio buttons labeled as specified.
     *
     * @param label1 the label for the first radio button
     * @param label2 the label for the second radio button
     * @param isVertical true for vertical layout, false for horizontal layout
     */
    public RadioButtonGroup(String label1, String label2, boolean isVertical) {
        super(10); // spacing between buttons

        rb1 = new RadioButton(label1);
        rb2 = new RadioButton(label2);
        rb1.setToggleGroup(toggleGroup);
        rb2.setToggleGroup(toggleGroup);

        if (!isVertical) {
            // Create an HBox to hold the buttons horizontally
            HBox horizontalContainer = new HBox(10);
            horizontalContainer.getChildren().addAll(rb1, rb2);
            this.getChildren().add(horizontalContainer);
        } else {
            // Vertical adds buttons directly to this VBox
            this.getChildren().addAll(rb1, rb2);
        }
    }
```

```
    /**
     * Gets the text of the currently selected radio button.
     *
     * @return the text of the selected button, or null if nothing is
selected
     */
    public String getSelectedButton() {
        RadioButton selected = (RadioButton)
toggleGroup.getSelectedToggle(); //typecast generic Toggle object to
RadioButton object
        return (selected != null) ? selected.getText() : null;
    }

    /**
     * Sets the first radio button as selected by default.
     */
    public void selectFirst() {
        rb1.setSelected(true);
    }
}


package sprint2.product;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.control.Spinner;
import javafx.scene.control.SpinnerValueFactory;
import javafx.stage.Stage;
import sprint2.product.SOSGame.Player;

/**
 * This class represents the GUI for the SOS game.
 */
public class SOSGameGUI extends Application {

    private SOSGame game;
    private GridPane boardGrid;
    private RadioButtonGroup bluePlayerRadio;
    private RadioButtonGroup redPlayerRadio;
    private Label instructionLabel;
    /**
     * Starts the JavaFX application and sets up the main window.
     *
     * @param primaryStage the primary stage for this application
     */
    @Override
    public void start(Stage primaryStage) {

        // Setting up main window.
        BorderPane root = new BorderPane();
        root.setPadding(new Insets(10));

        // Top section.
        HBox topSection = new HBox(30);
        topSection.setAlignment(Pos.CENTER);
```

```
        Label titleLabel = new Label("SOS");
        titleLabel.setTextFill(Color.BLUE);
        titleLabel.setStyle("-fx-font-size: 20px;");

        Spinner<Integer> boardSizeSpinner = new Spinner<>();
        boardSizeSpinner.setValueFactory(new
SpinnerValueFactory.IntegerSpinnerValueFactory(3, 10, 3));
        boardSizeSpinner.setMaxWidth(60);

        RadioButtonGroup gameMode = new RadioButtonGroup("Simple
game", "General game", false);
        gameMode.selectFirst();
        gameMode.setPadding(new Insets(30));

        Button newGameButton = new Button("New Game");
        newGameButton.setOnAction(e -> {
            int size = boardSizeSpinner.getValue();
            String selectedMode = gameMode.getSelectedButton();
            SOSGame.GameMode mode = selectedMode.equals("Simple
game")
                    ? SOSGame.GameMode.SIMPLE
                    : SOSGame.GameMode.GENERAL;
            game = new SOSGame(size, mode);
            updateBoardDisplay();
            instructionLabel.setText("Current Turn: Blue Player");
            instructionLabel.setTextFill(Color.BLUE);
            System.out.println("New game started: " + size + "x" + size + ",
" + mode);
        });

        topSection.getChildren().addAll(titleLabel, gameMode,
boardSizeSpinner, newGameButton);
        root.setTop(topSection);

        // Center section.
        VBox centerSection = new VBox(20);
        centerSection.setAlignment(Pos.CENTER);
        centerSection.setPadding(new Insets(30));

        boardGrid = new GridPane();
        boardGrid.setAlignment(Pos.CENTER);
        boardGrid.setHgap(0);
        boardGrid.setVgap(0);

        centerSection.getChildren().addAll(boardGrid);
        root.setCenter(centerSection);

        // Left Section.
        VBox leftSection = new VBox(15);
        leftSection.setAlignment(Pos.TOP_CENTER);

        Label blueLabel = new Label("Blue Player");
        blueLabel.setStyle("-fx-font-weight: bold;");

        bluePlayerRadio = new RadioButtonGroup("S", "O");
        bluePlayerRadio.selectFirst();

        leftSection.getChildren().addAll(blueLabel, bluePlayerRadio);
        root.setLeft(leftSection);

        //Right section.
        VBox rightSection = new VBox(15);
        rightSection.setAlignment(Pos.TOP_CENTER);

        Label redLabel = new Label("Red Player");
        redLabel.setStyle("-fx-font-weight: bold;");
```

```
    redPlayerRadio = new RadioButtonGroup("S", "O");
    redPlayerRadio.selectFirst();

    rightSection.getChildren().addAll(redLabel, redPlayerRadio);
    root.setRight(rightSection);

    // Bottom section.
    VBox bottomSection = new VBox(20);
    bottomSection.setAlignment(Pos.CENTER);
    bottomSection.setPadding(new Insets(10));

    instructionLabel = new Label("Current Turn:");
    CheckBox checkbox1 = new CheckBox("Record Game");

    bottomSection.getChildren().addAll(instructionLabel, checkbox1);
    root.setBottom(bottomSection);

    // Create scene and show window.
    Scene scene = new Scene(root, 400, 400);
    primaryStage.setTitle("SOS Game");
    primaryStage.setScene(scene);
    primaryStage.show();
}

/**
 * Updates the board display to match the current game state.
 * Clears the existing grid and creates new cells based on board size.
 */
private void updateBoardDisplay() {
    boardGrid.getChildren().clear();
    int size = game.getBoardSize();
    char[][] board = game.getBoard();
    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            final int r = row;
            final int c = col;
            Button cell = new Button(String.valueOf(board[row][col]));
            cell.setMinSize(40, 40);
            cell.setMaxSize(40, 40);
            cell.setAlignment(Pos.CENTER);
            cell.setStyle("-fx-border-color: black; -fx-border-width: 1;");
            // Wire up cells on game board.
            cell.setOnAction(e -> {
                if (game.isCellEmpty(r, c)) {
                    String selectedLetter;
                    if (game.getCurrentPlayer() == Player.BLUE) {
                        selectedLetter = bluePlayerRadio.getSelectedButton();
                    } else {
                        selectedLetter = redPlayerRadio.getSelectedButton();
                    }
                    game.makeMove(r,  c,  selectedLetter.charAt(0));
                    cell.setText(selectedLetter);
                    // Switch players and show turn.
                    game.switchPlayer();
                    Player nextPlayer = game.getCurrentPlayer();
                    if (nextPlayer == Player.BLUE) {
                        instructionLabel.setText("Current Turn: Blue Player");
                        instructionLabel.setTextFill(Color.BLUE);
                    } else {
                        instructionLabel.setText("Current Turn: Red Player");
                        instructionLabel.setTextFill(Color.RED);
                    }
                }
            });
            boardGrid.add(cell, col, row);
        }
    }
}
```

```
    /**
     * Main entry point for the application.
     *
     * @param args command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
}



package sprint2.product;

/**
*Represents the SOS game logic.
*Handles game size, rules, and player turns.
*/
public class SOSGame {

    /**
     * Represents the game mode options.
     */
    public enum GameMode {
        SIMPLE,
        GENERAL
    }

    /**
     * Represents the player colors.
     */
    public enum Player {
        BLUE,
        RED
    }

    private Player currentPlayer;
    private int boardSize;
    private GameMode gameMode;
    private char[][] board;


    /**
     * Creates a new SOS game with the specified settings.
     *
     * @param boardSize the size of the board (e.g. 3 for 3x3)
     * @param gameMode the game mode (SIMPLE or GENERAL)
     */
    public SOSGame(int boardSize, GameMode gameMode) {
        this.boardSize = boardSize;
        this.gameMode = gameMode;
        this.board = new char[boardSize][boardSize];
        this.currentPlayer = Player.BLUE;
        for (int i = 0; i < boardSize; i++) {
            for (int j = 0; j < boardSize; j++) {
                board[i][j] = ' ';
            }
        }
    }

    /**
     * Updates the board size.
     * Used when user changes the spinner value.
     *
     * @param boardSize the new board size (between 3 and 10)
     */
    public void setBoardSize(int boardSize) {
        this.boardSiz boardSize;
    }
```

```
    /**
     * Updates the game mode.
     * Used when user selects Simple or General game.
     *
     * @param gameMode the new game mode
     */
    public void setGameMode(GameMode gameMode) {
        this.gameMode = gameMode;
    }

    /**
     * Gets the current board state.
     *
     * @return the 2D char array representing the board.
     */
    public char[][] getBoard(){
        return board;
    }

    /**
     * Gets the current board size.
     *
     * @return the board size
     */
    public int getBoardSize() {
        return boardSize;
    }

    /**
     * Checks if a cell is empty.
     *
     * @param row the row index
     * @param col the column index
     * @return true if the cell is empty, false otherwise
     */
    public boolean isCellEmpty(int row, int col) {
        return board[row][col] == ' ';
    }

    /**
     * Gets the current player.
     *
     * @return the current player
     */
    public Player getCurrentPlayer() {
        return currentPlayer;
    }

    /**
     * Switches to the other player.
     */
    public void switchPlayer() {
        currentPlayer = (currentPlayer == Player.BLUE) ? Player.RED :
Player.BLUE;
    }

    /**
     * Places a letter at the specified position.
     *
     * @param row the row index
     * @param col the column index
     * @param letter the letter to place ('S' or 'O')
     */
    public void makeMove(int row, int col, char letter) {
        board[row][col] = letter;
    }
}
```

Here is a **JUnit 5** test class written in **Google Java Style** that covers the acceptance criteria based on the user stories you provided. These tests validate:

1. Choosing a board size
2. Choosing the game mode
3. Starting a new game with the selected settings
4. Making a move in simple mode
5. Making a move in general mode

Each test is accompanied by **comments**, and wildcard imports are avoided.

✅ Test Class: `SOSGameTest.java`

java Copy code

```java
package sprint2.product;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.junit.jupiter.api.Assertions.assertNull;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

/**
 * Unit tests for the SOSGame and related components.
 * Covers functionality based on acceptance criteria from user stories.
 */
public class SOSGameTest {

  private SOSGame simpleGame;
  private SOSGame generalGame;

  /**
   * Sets up test games before each test.
   */
  @BeforeEach
  public void setUp() {
    simpleGame = new SOSGame(5, SOSGame.GameMode.SIMPLE);
    generalGame = new SOSGame(7, SOSGame.GameMode.GENERAL);
  }

  /**
   * Tests that the board size can be selected and is set correctly.
   */
  @Test
  public void testChooseBoardSize() {
    assertEquals(5, simpleGame.getBoardSize(), "Board size should be 5");
    assertEquals(7, generalGame.getBoardSize(), "Board size should be 7");
  }

  /**
   * Tests that the correct game mode is selected when creating a game.
   */
  @Test
  public void testChooseGameMode() {
    assertEquals(SOSGame.GameMode.SIMPLE, simpleGame.gameMode, "Game mode should be SIMPLE");
    assertEquals(SOSGame.GameMode.GENERAL, generalGame.gameMode, "Game mode should be GENERAL");
  }

  /**
   * Tests that a new game starts with an empty board and blue player.
   */
  @Test
  public void testStartNewGameState() {
    char[][] board = simpleGame.getBoard();
    for (int i = 0; i < board.length; i++) {
      for (int j = 0; j < board[i].length; j++) {
        assertEquals(' ', board[i][j], "All cells should be initially empty");
```

```
        }
      }
      assertEquals(SOSGame.Player.BLUE, simpleGame.getCurrentPlayer(), "Game should start with BLUE
    }

    /**
     * Tests making a move in a simple game.
     */
    @Test
    public void testMakeMoveInSimpleGame() {
      assertTrue(simpleGame.isCellEmpty(1, 1), "Cell (1,1) should be empty initially");
      simpleGame.makeMove(1, 1, 'S');
      assertEquals('S', simpleGame.getBoard()[1][1], "Cell (1,1) should contain 'S'");
      simpleGame.switchPlayer();
      assertEquals(SOSGame.Player.RED, simpleGame.getCurrentPlayer(), "Player should switch to RED")
    }

    /**
     * Tests making a move in a general game.
     */
    @Test
    public void testMakeMoveInGeneralGame() {
      assertTrue(generalGame.isCellEmpty(2, 2), "Cell (2,2) should be empty initially");
      generalGame.makeMove(2, 2, 'O');
      assertEquals('O', generalGame.getBoard()[2][2], "Cell (2,2) should contain 'O'");
      generalGame.switchPlayer();
      assertEquals(SOSGame.Player.RED, generalGame.getCurrentPlayer(), "Player should switch to RED"
    }

    /**
     * Tests RadioButtonGroup selection logic.
     */
    @Test
    public void testRadioButtonGroupSelection() {
      RadioButtonGroup group = new RadioButtonGroup("S", "O");
      group.selectFirst(); // Should select "S"
      assertEquals("S", group.getSelectedButton(), "First button 'S' should be selected");
    }

    /**
     * Tests no selection returns null in RadioButtonGroup.
     */
    @Test
    public void testRadioButtonGroupNoSelection() {
      RadioButtonGroup group = new RadioButtonGroup("S", "O");
      assertNull(group.getSelectedButton(), "No button should be selected by default without selectF
    }
}
```

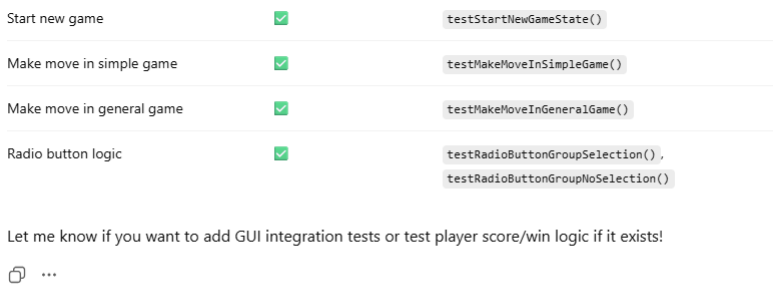
Copy code

💡 Notes:

- This class should be placed in your `test` directory, under the same package: `sprint2.product`.
- If you're using **Maven or Gradle**, make sure the JavaFX module is configured properly for tests (e.g., mock or exclude GUI code when unnecessary).
- `SOSGame.gameMode` is accessed directly here for test purposes. Consider adding a `getGameMode()` method if you want stricter encapsulation.

✅ Summary of Coverage:

Feature	Covered?	Test Method
Choose board size	✅	`testChooseBoardSize()`
Choose game mode	✅	`testChooseGameMode()`



Start new game	✅	`testStartNewGameState()`
Make move in simple game	✅	`testMakeMoveInSimpleGame()`
Make move in general game	✅	`testMakeMoveInGeneralGame()`
Radio button logic	✅	`testRadioButtonGroupSelection()`, `testRadioButtonGroupNoSelection()`

Let me know if you want to add GUI integration tests or test player score/win logic if it exists!

The ChatGPT automated tests were mostly correct. I removed the test functions **testRadioButtonGroupSelection()** and **testRadioButtonGroupNoSelection()** since they didn't work and the other tests covered the user stories. I also renamed the class to **TestSOSGame** and changed the package to **package sprint2.test;**. I added an import, the **import sprint2.product.SOSGame**;. I removed **import static org.junit.jupiter.api.Assertions.assertNull;** and the **import sprint2.product.RadioButtonGroup;** since they aren't being used. I added a public function **getGameMode()**, which I used in **testChooseGameMode()** because the variable that the ChatGPT code was trying to access was the **private GameMode gameMode;** variable from the **SOSGame.java** class.