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**.

GitHub Link: https://github.com/Tarycx/CS499SOSProject
Youtube Link: https://www.youtube.com/watch?v=1ykFt4GnA44

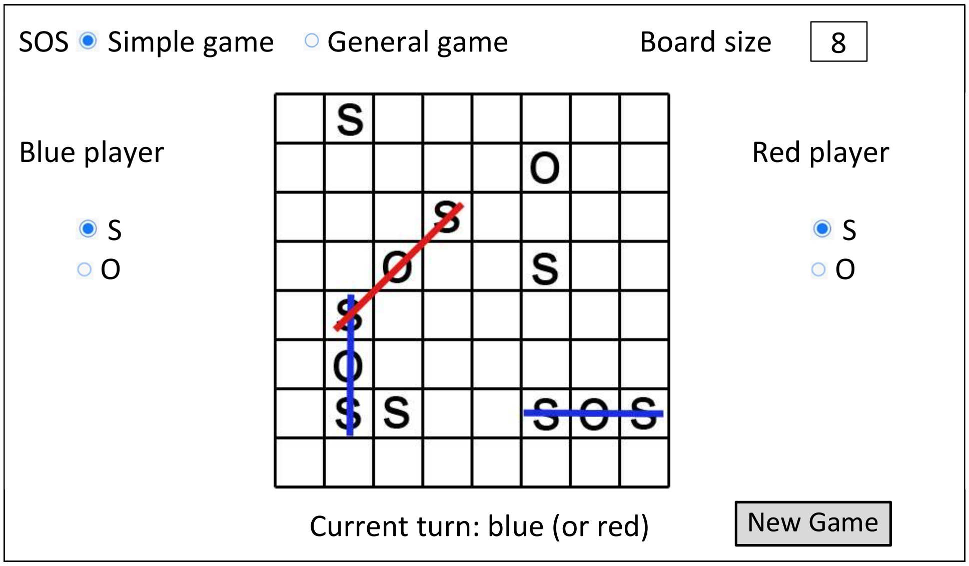


Figure 1. Sample GUI layout of the working program for Sprint 3

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.

   Youtube Link: https://www.youtube.com/watch?v=1ykFt4GnA44

2. Summary of Source Code (1 points)

Source code file name	Production code or test code?	# lines of code
SOSGame	Production	10
GameMenuGUI	Production	98
GameMenuGUITest	Testing	68
SOSBoard	Production	222
SOSBoardTest	Testing	73
SOSGameSimple	Production	119
SOSGameSimpleTest	Testing	111
SOSGameGeneral	Production	199
SOSGameGeneralTest	Testing	127
SOSGameGUI	Production	222
SOSGameGUITest	Testing	68
SimpleGaemMoveTest	Testing	90
GeneralGameMoveTest	Testing	94
	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	GameMenu
		GameMenuGUI
		SOSBoard
			setBoardSize(int)
			boardSizeSpinner
			SOSBoard(int, string)
				complete	

	1.2	GameMenu
		GameMenuGUI
			GameMenu()
			setBoardSize(int)
			boardSizeSpinner
				complete	

	…				
2. Choose the game mode of a chosen board	2.1	SOSBoard GameMenu	SOSBoard(int) StartGame()	complete	

	2.2	SOSBoard
		GameMenu
			SOSBoard(int,string)
			StartGame()
				complete	

3. Start a new game of the chosen board	3.1	GameMenu
		GameMenuGUI
			GameMenu()
			startGame()
			GameMenuGUI(startGame)
				complete	

size and game mode					
	3.2	SOSGameGUI GameMenu	newGameButton GameMenu()	complete	

4. Make a move in a simple game	4.1	SOSGameGUI
		SOSBoard
			SOSGameGUI(SOSBoard)
			setCellValue(int, int, String) togglePlayer()
				complete	

	4.2	SOSBoard
		SOSGameGUI
			isCellEmpty(int, int)
			SOSGameGUI(SOSBoard)
				complete	

5.a simple game is over	5.1	SOSGameSimple
		SOSBoard
			checkForSOS(int row, int col)
			checkWinCond()
			makeMove()
				complete	

	5.2	SOSGameSimple
		SOSBoard
			checkForSOS(int row, int col)
			checkWinCond()
			makeMove()
				complete	

	5.3	SOSGameSimple SOSGameGUI SOSBoard	resetBoard()	complete	

6. Make a move in a general game	6.1	SOSGameGUI
		SOSBoard
			SOSGameGUI(SOSBoard)
			setCellValue(int, int, String) togglePlayer()
				complete	

	6.2	SOSBoard
		SOSGameGUI
			isCellEmpty(int, int)
			SOSGameGUI(SOSBoard)
				complete	

7. A general game is over	7.1	SOSGameGeneral
		SOSBoard
			makeMove()
			checkForSOS(int row, int col)
			checkWinCond()
				complete	

	7.2	SOSGameGeneral
		SOSBoard
			makeMove()
			checkForSOS(int row, int col)
			checkWinCond()
				complete	

	7.3	SOSGameGeneral SOSGameGUI SOSBoard	resetBoard()	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.

You are required to use free ChatGPT version to create 2 unit tests using ChatGPT. You also need to ensure that that the generated unit tests are correct, and refined 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 had to correct. You may also use LLMs hosted locally. 2 points will be deducted if no screenshots provided.

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. Choose a board size	1.1	GameMenuTest	testSetValidBoardSize()	Input: The board size is explicitly set to 5 using setBoardSize(5) Output: The board size should be updated to 5
	1.2	GameMenuTest	testDefualtBoardSize()	Input: No custom board size is set Output: The default board size should be 3
	1.3	GameMenuTest	testDefualtBoardSize()	Input: The board size is set to an invalid value 12 Output: An IllegalArgumentException should be thrown, as the board size is outside the valid range of 3-10
2. Choose the game mode of a chosen board	2.1	GameMenuTest	testSetGameTypeSimple()	Input: The game type is explicitly set to "Simple Game" Output: The game type should be updated to "Simple Game"
	2.2	GameMenuTest	testSetGameTypeSimple()	Input: The game type is explicitly set to "General Game" Output: The game type should be updated to "General Game"
3. Start a new game of the chosen board size and	3.1	GameMenuTest	testStartNewGame_Custom Settings()	Input: Custom board size 5 and game mode "General Game" Output: board size 5, game mode General

game mode				

	3.2	GameMenuTest	testDefaultGameType()
			testDefaultBoardSize()
				Input: No custom game type is set 5 Output: The default game type should be "Simple Game"
				Input: No custom board size is set Output: The default board size should be 3

4. Make a move in a simple game	4.1	SimpleGameMoveTest
		SOSBoardTest
			testMakeMove_EmptyCell()
			testTogglePlayer()
				Input: The cell at position (0, 0) on a 3x3 board in "Simple" game mode Output: empty cell at (0,0) is filled with "S"
				Input: game begins with player color "Blue" placing S Output: player color should change to "Red" current letter should change to "O" and then after second toggle back to original state

	4.2	SimpleGameMoveTest	testMakeMove_FilledCell()	Input: First, the cell at position (0, 0) is filled with "S" Output: no changes, (0,0) should have original move

5.A simple game is over	5.1	SOSGameSimpleTest	testGameEndsWithFirstSOSByBluePlayer()
			testGameEndsWithFirstSOSByRedPlayer()
				Input: Blue player completes an "SOS" sequence Output: Game should end, and Blue should be the winner
				Input: Red player

				completes an "SOS" sequence Output: Game should end, and Red should be the winner
	5.2	SOSGameSimpleTest	testGameEndsInDrawWhenBoardIsFullWithoutSOS()	Input: Fill the board without forming any "SOS" sequence Output: Game should end in a draw
	5.3	SOSGameSimpleTest		

6. Make a move in a general game	6.1	GeneralGameMoveTest
		SOSBaordTest
			testMakeMove_EmptyCell()
			testTogglePlayer()
				Input: The cell at position (0, 0) on a 3x3 board in "general" game mode filled with “S” Output: empty cell at (0,0) is filled with “S”
				Input: game begins with player color “Blue” placing S Output: player color should change to “Red” current letter should change to “O” and then after second toggle back to original state

	6.2	GeneralGameMoveTest	testMakeMove_FilledCell()	Input: First, the cell at position (0, 0) is filled with "S" Output: no changes, (0,0) should have original move

7.A general game is over	7.1	SOSGameGeneralTest	testPlayerScoresMultipleSOSSequencesAndWins()
			testRedPlayerWinsWithMoreSOSSequences()
				Input: Arrange: Player "Blue" creates two "SOS" sequences Output: Game should end, and Blue should be

				the winner with 2 points Input: Blue and Red players create multiple "SOS" sequences, with Red scoring more Output: Game should end, and Red should be the winner with a higher score: 2
	7.2	SOSGameGeneralTest	testGameEndsInDrawWithEqualSOSCounts()	Input: Both players create one "SOS" sequence each Output: Game should end in a draw since both players scored equally
	7.3	SOSGameGeneralTest	testGameEndsWhenBoardIsFull()	Input: Fill the board without creating an "SOS" that ends the game prematurely Output: the board is full and the game ends

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

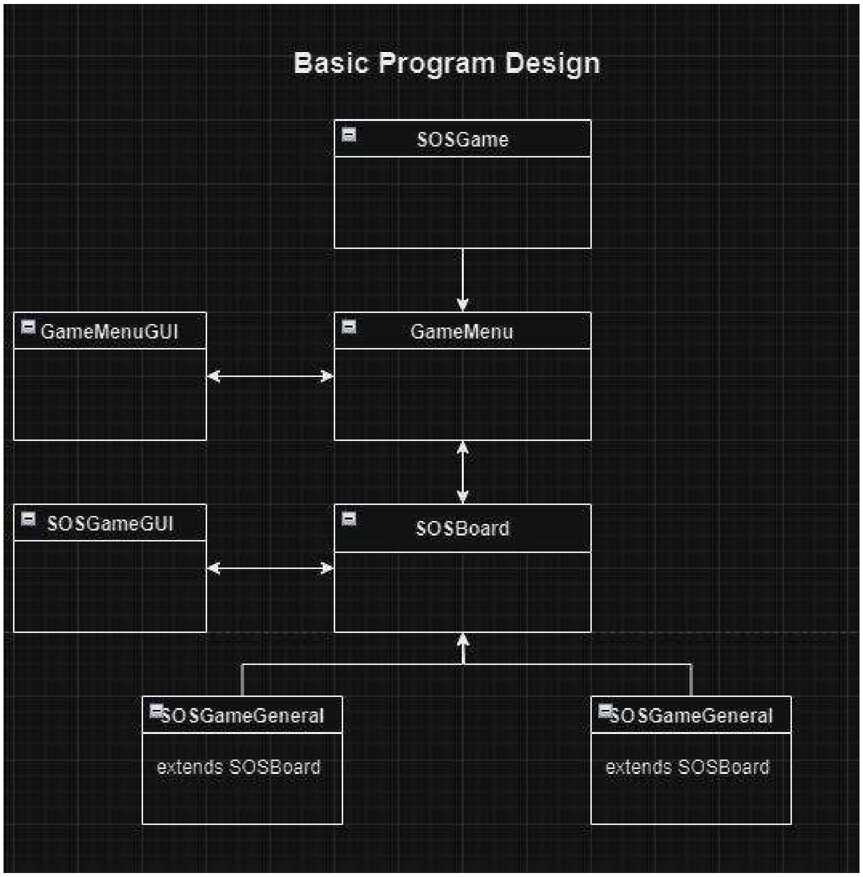
5. Describe how the class hierarchy in your design deals with the common and different requirements of the Simple Game and the General Game? At least 1/2 of a page is required. (4 points)

In my SOS Game my class hierarchy between Simple Game (SOSGameSimple) and General Game(SOSGameGeneral) are denoted as child classes of board object(SOSBoard.). SOSBoard servers as the foundational base for each game type. It's main functionality relates to board state, player turns, cell behavor, sequence tracking, SOS checking, and other general board functions. SOSBoard provides build in object (SOSSequence) and list container to hold SOS Sequence objects to hold information about each completed SOS sequence and update line data

Simple Game denoted by the class: SOSGameSimple servers as a subclass of SOSBoard to provide simple game functionality. This utilized board state functionality and SOSSequence object directly from SOSBoard. The Simple game class overrides functions for makeMove(main logic for handling a player moves on the SOS game board and handles setting and checking win conditions), checkForSOS and following SOS directional checkers from SOSBoard parent.

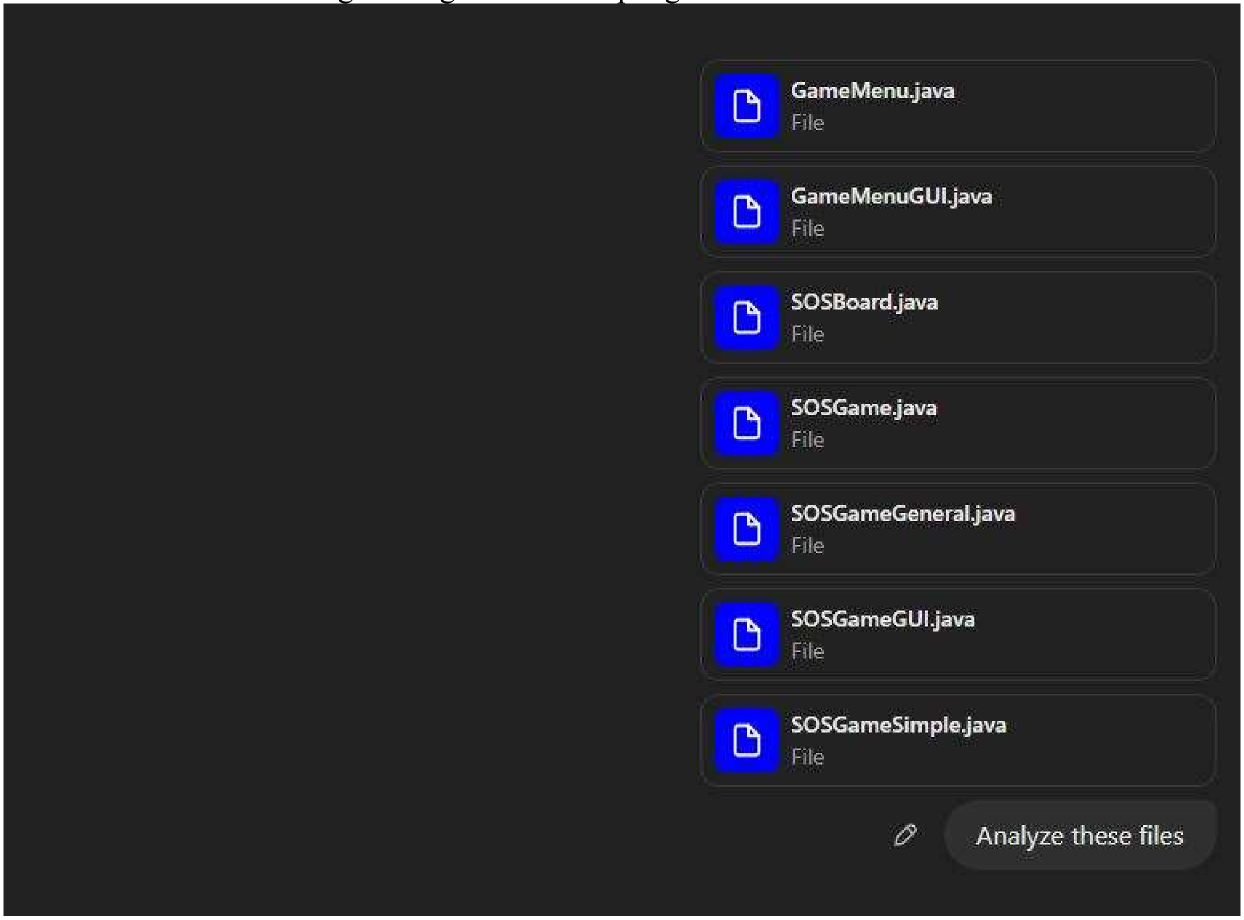
General Game denoted by the class: SOSGameGeneral servers as a subclass of SOSBoard to provide general game logic in program. This class demands much more complexity compared to Simple game class. This class utilizes board state functionality, SOS directional Checkers and SOSSequence object directly from SOSBoard parent class. The General game class overrides function for checking SOS sequence, make move function (main logic for handling a player moves on the SOS game board and handles setting and checking win conditions). General game logic has it own built in sos sequence marker, nested boolean containers, SOS checking iterator, player scoring methods.

Simple game and general game class are both dependent on many SOSBoard class methods. The subclass of SOSboard both rely heavily on board state logic, directional sequence tracking, and SOS sequence tracking methods and objects. The biggest differecnce between these subclasses is the main logic for handling a player moves on the SOS game board, setting and checking win conditions.



ChatGPT Unit Testing ScreenShots:

Files provided to and analyzed by LLM
Unit Tests are focused on general game and simple game class features



GameMenu.java
File

GameMenuGUI.java
File

SOSBoard.java
File

SOSGame.java
File

SOSGameGeneral.java
File

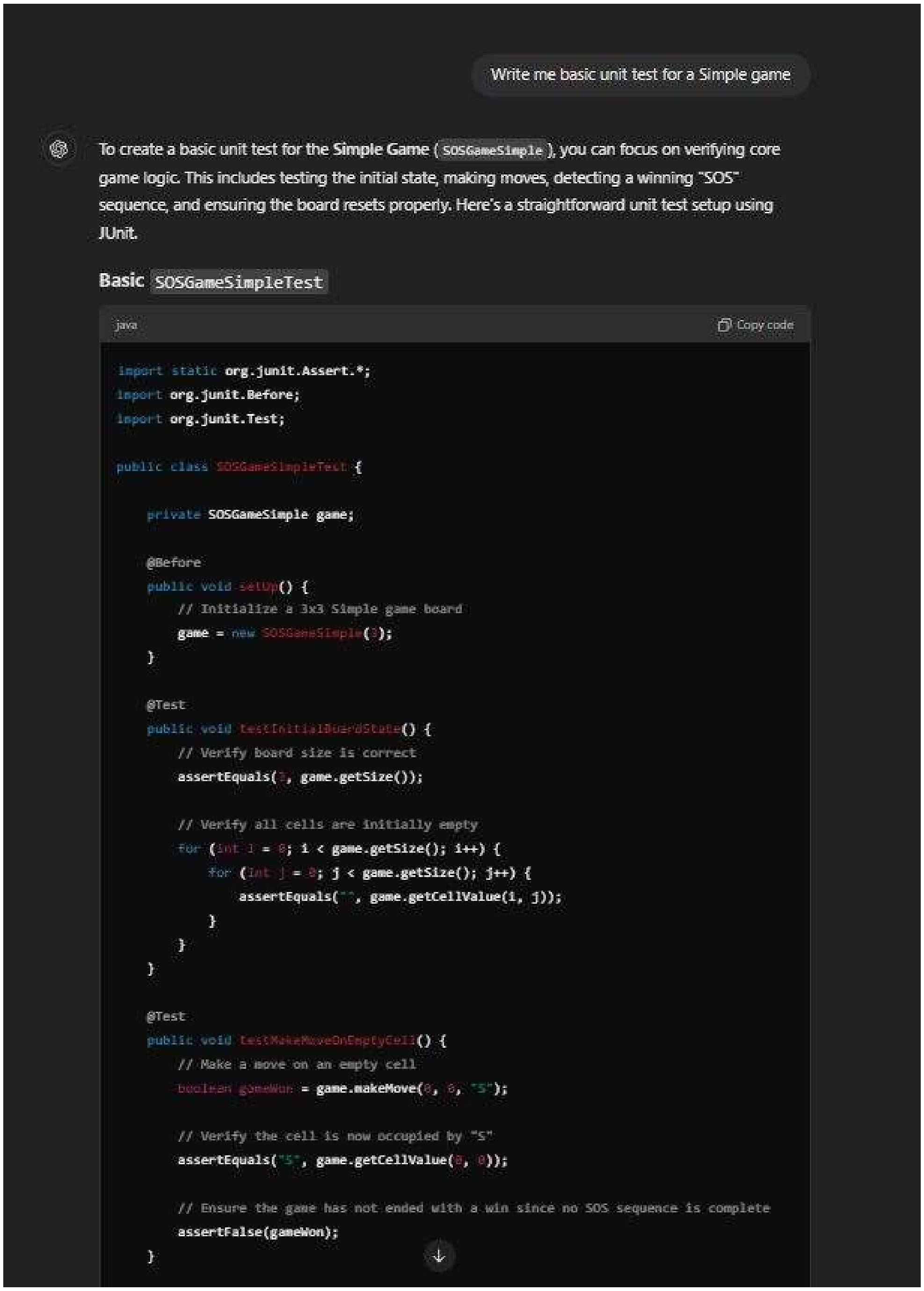
SOSGameGUI.java
File

SOSGameSimple.java
File

Analyze these files

Unit Test 1 (Simple Game Logic Testing)

Notes: For the most part the output was what I was looking for but there where issues with LLM incorrectly using my simple game logic. Many class methods were not properly implanted. Corrections where not complex. To prevent this in the future a more descriptive prompt might be useful

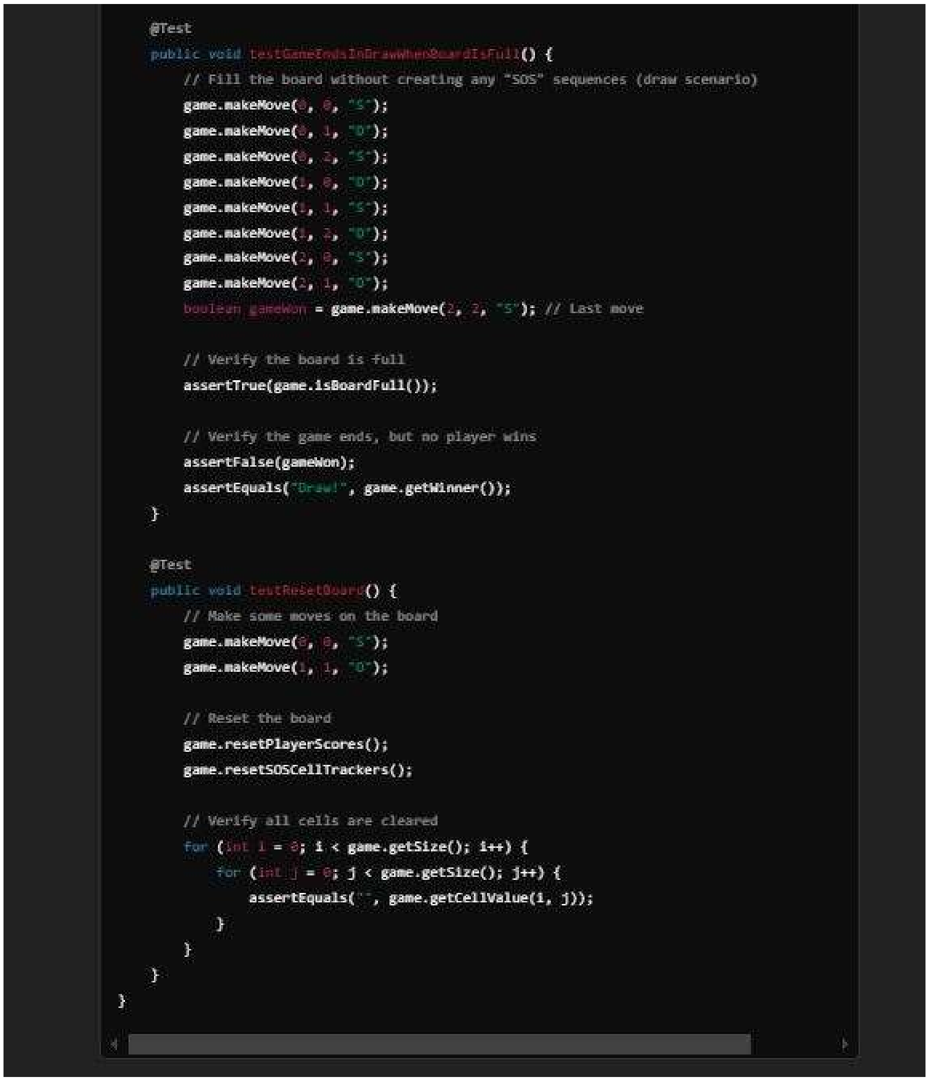


Write me basic unit test for a Simple game

To create a basic unit test for the **Simple Game** (`SOSGameSimple`), you can focus on verifying core game logic. This includes testing the initial state, making moves, detecting a winning "SOS" sequence, and ensuring the board resets properly. Here's a straightforward unit test setup using JUnit.

Basic `SOSGameSimpleTest`

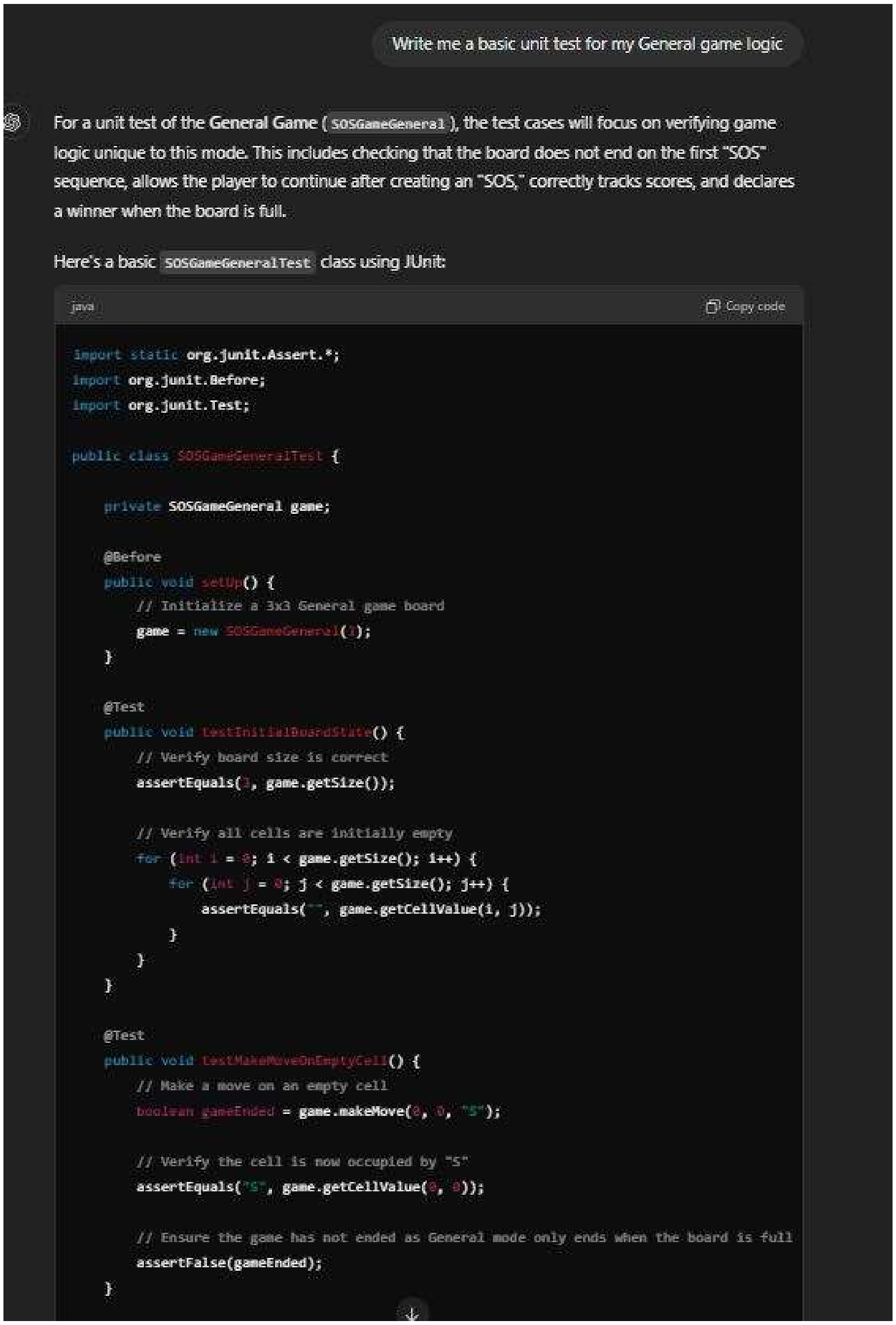
```java
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class SOSGameSimpleTest {

    private SOSGameSimple game;

    @Before
    public void setUp() {
        // Initialize a 3x3 Simple game board
        game = new SOSGameSimple(3);
    }

    @Test
    public void testInitialBoardState() {
        // Verify board size is correct
        assertEquals(3, game.getSize());

        // Verify all cells are initially empty
        for (int i = 0; i < game.getSize(); i++) {
            for (int j = 0; j < game.getSize(); j++) {
                assertEquals("", game.getCellValue(i, j));
            }
        }
    }

    @Test
    public void testMakeMoveOnEmptyCell() {
        // Make a move on an empty cell
        boolean gameWon = game.makeMove(0, 0, "S");

        // Verify the cell is now occupied by "S"
        assertEquals("S", game.getCellValue(0, 0));

        // Ensure the game has not ended with a win since no SOS sequence is complete
        assertFalse(gameWon);
    }
```



```
    @Test
    public void testGameEndsInDrawWhenBoardIsFull() {
        // Fill the board without creating any "SOS" sequences (draw scenario)
        game.makeMove(0, 0, "S");
        game.makeMove(0, 1, "O");
        game.makeMove(0, 2, "S");
        game.makeMove(1, 0, "O");
        game.makeMove(1, 1, "S");
        game.makeMove(1, 2, "O");
        game.makeMove(2, 0, "S");
        game.makeMove(2, 1, "O");
        boolean gameWon = game.makeMove(2, 2, "S"); // Last move

        // Verify the board is full
        assertTrue(game.isBoardFull());

        // Verify the game ends, but no player wins
        assertFalse(gameWon);
        assertEquals("Draw!", game.getWinner());
    }

    @Test
    public void testResetBoard() {
        // Make some moves on the board
        game.makeMove(0, 0, "S");
        game.makeMove(1, 1, "O");

        // Reset the board
        game.resetPlayerScores();
        game.resetSOSCellTrackers();

        // Verify all cells are cleared
        for (int i = 0; i < game.getSize(); i++) {
            for (int j = 0; j < game.getSize(); j++) {
                assertEquals("", game.getCellValue(i, j));
            }
        }
    }
}
```

Unit Test 2 (General Game Logic Testing)

Notes: For the most part the output was what I was looking for but there where issues with LLM incorrectly using my class methods. Correction where not complex. To prevent this in the future a more descriptive prompt might be useful

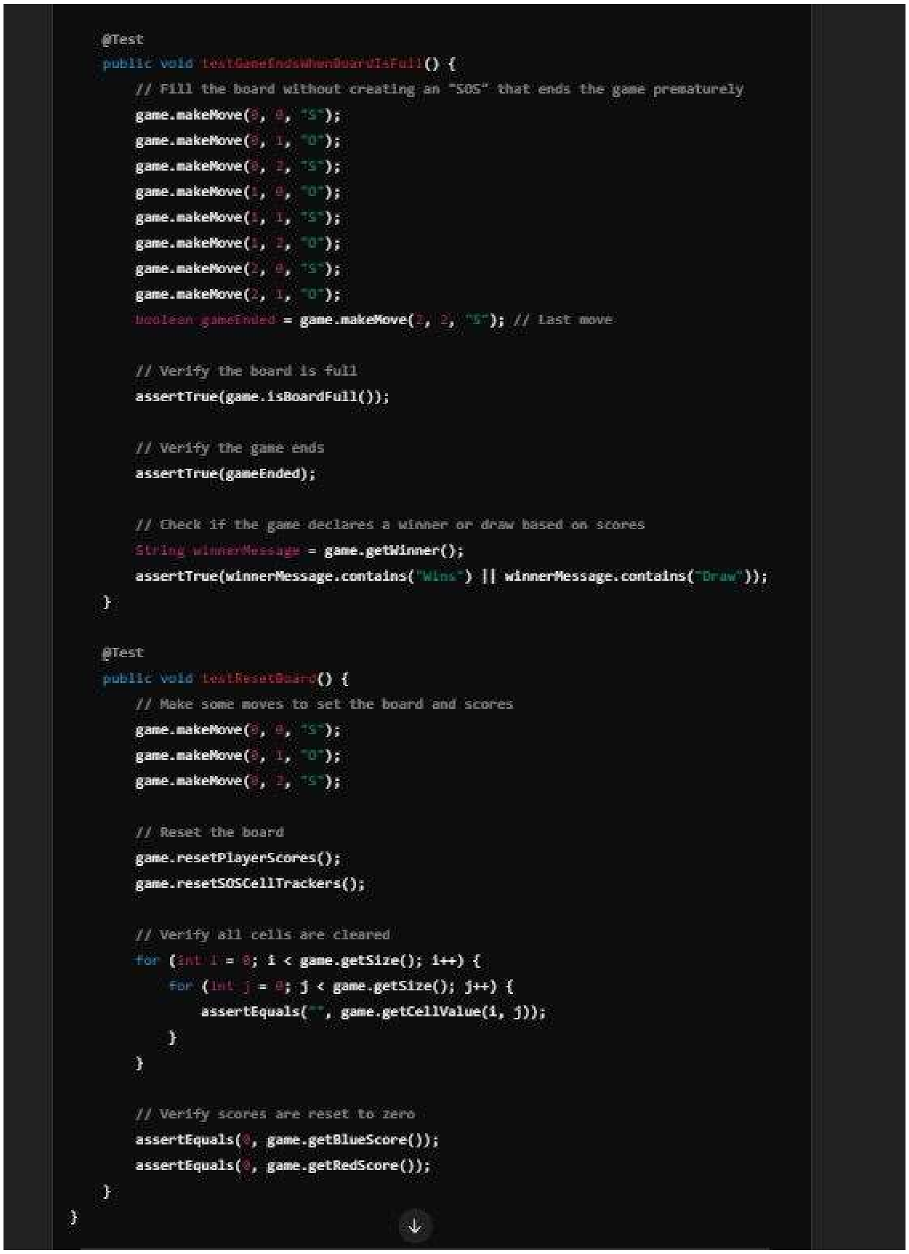


Write me a basic unit test for my General game logic

For a unit test of the **General Game** (`SOSGameGeneral`), the test cases will focus on verifying game logic unique to this mode. This includes checking that the board does not end on the first "SOS" sequence, allows the player to continue after creating an "SOS," correctly tracks scores, and declares a winner when the board is full.

Here's a basic `SOSGameGeneralTest` class using JUnit:

```java
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class SOSGameGeneralTest {

    private SOSGameGeneral game;

    @Before
    public void setUp() {
        // Initialize a 3x3 General game board
        game = new SOSGameGeneral(3);
    }

    @Test
    public void testInitialBoardState() {
        // Verify board size is correct
        assertEquals(3, game.getSize());

        // Verify all cells are initially empty
        for (int i = 0; i < game.getSize(); i++) {
            for (int j = 0; j < game.getSize(); j++) {
                assertEquals("", game.getCellValue(i, j));
            }
        }
    }

    @Test
    public void testMakeMoveOnEmptyCell() {
        // Make a move on an empty cell
        boolean gameEnded = game.makeMove(0, 0, "S");

        // Verify the cell is now occupied by "S"
        assertEquals("S", game.getCellValue(0, 0));

        // Ensure the game has not ended as General mode only ends when the board is full
        assertFalse(gameEnded);
    }
```



```
    @Test
    public void testGameEndsWhenBoardIsFull() {
        // Fill the board without creating an "SOS" that ends the game prematurely
        game.makeMove(0, 0, "S");
        game.makeMove(0, 1, "O");
        game.makeMove(0, 2, "S");
        game.makeMove(1, 0, "O");
        game.makeMove(1, 1, "S");
        game.makeMove(1, 2, "O");
        game.makeMove(2, 0, "S");
        game.makeMove(2, 1, "O");
        boolean gameEnded = game.makeMove(2, 2, "S"); // Last move

        // Verify the board is full
        assertTrue(game.isBoardFull());

        // Verify the game ends
        assertTrue(gameEnded);

        // Check if the game declares a winner or draw based on scores
        String winnerMessage = game.getWinner();
        assertTrue(winnerMessage.contains("Wins") || winnerMessage.contains("Draw"));
    }

    @Test
    public void testResetBoard() {
        // Make some moves to set the board and scores
        game.makeMove(0, 0, "S");
        game.makeMove(0, 1, "O");
        game.makeMove(0, 2, "S");

        // Reset the board
        game.resetPlayerScores();
        game.resetSOSCellTrackers();

        // Verify all cells are cleared
        for (int i = 0; i < game.getSize(); i++) {
            for (int j = 0; j < game.getSize(); j++) {
                assertEquals("", game.getCellValue(i, j));
            }
        }

        // Verify scores are reset to zero
        assertEquals(0, game.getBlueScore());
        assertEquals(0, game.getRedScore());
    }
}
```