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/Tarycx/CS499SOSProject

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.

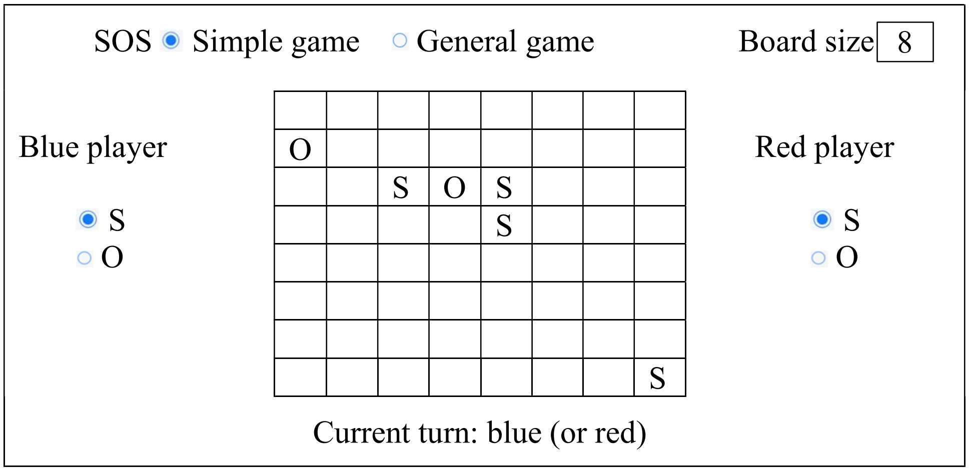


Figure 1. Sample GUI layout of the Sprint 2 program

Deliverables:

1. Demonstration (8 points)

Submit a 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://www.youtube.com/watch?v=ZkmKbG5Wef0
Note: VS Code extension(Test Runner for Java) is being used to support with unit testing environment.

	Feature	
1	Choose board size	Complete
2	Choose game mode	Complete
3	Initial game of the chosen board size and game mode	Complete
4	"S" moves	Complete
5	"O" moves	Complete
6	Automated unit tests	Complete
…		

2. Summary of Source Code (1 points)
Note: Test Code in separate directory

Source code file name	Production code or test code?	# lines of code
SOSGame.java	Production	10
SOSBoard.java	Production	75
SOSBoardTest.java	Testing	65
GameMenuGUI.java	Production	98
SOSGameGUI.java	Production	175
GameMenu.java	Production	37
GameMenuTest.java	Test	70
GeneralGameMoveTest.java	Test	47
SimpleGameMoveTest.java	Test	47
	Total	624

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	GameMenu
		GameMenuGUI
		SOSBoard
			setBoardSize(int)
			boardSizeSpinner
			SOSBoard(int, string)
				complete	

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

	1.3	GameMenuGUI GameMenu	boardSizeSpinner setBoardSize(int)	complete	

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

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

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

	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	

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	

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 using ChatGPT. You also need to ensure that that the generated user stories 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. Points will be deducted if no screenshots 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	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	testSetInvalidBoardSize()	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	testSetGameTypeGeneral()	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 game mode	3.1	GameMenuTest	testStartNewGame_CustomSettings()	Input: Custom board size 5 and game mode "General Game" Output: board size 5, game mode General
	3.2	GameMenuTest	testDefaultGameType()	Input: No custom game type is set

				Output: The default game type should be "Simple Game"
			testDefaultBoardSize()	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	testMakeMove_EmptyCell()	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”
		SOSBoardTest	testTogglePlayer()	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
6. Make a move in a general game	6.1	GeneralGameMoveTest	testMakeMove_EmptyCell()	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”
		SOSBaordTest	testTogglePlayer()	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

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			
	…			

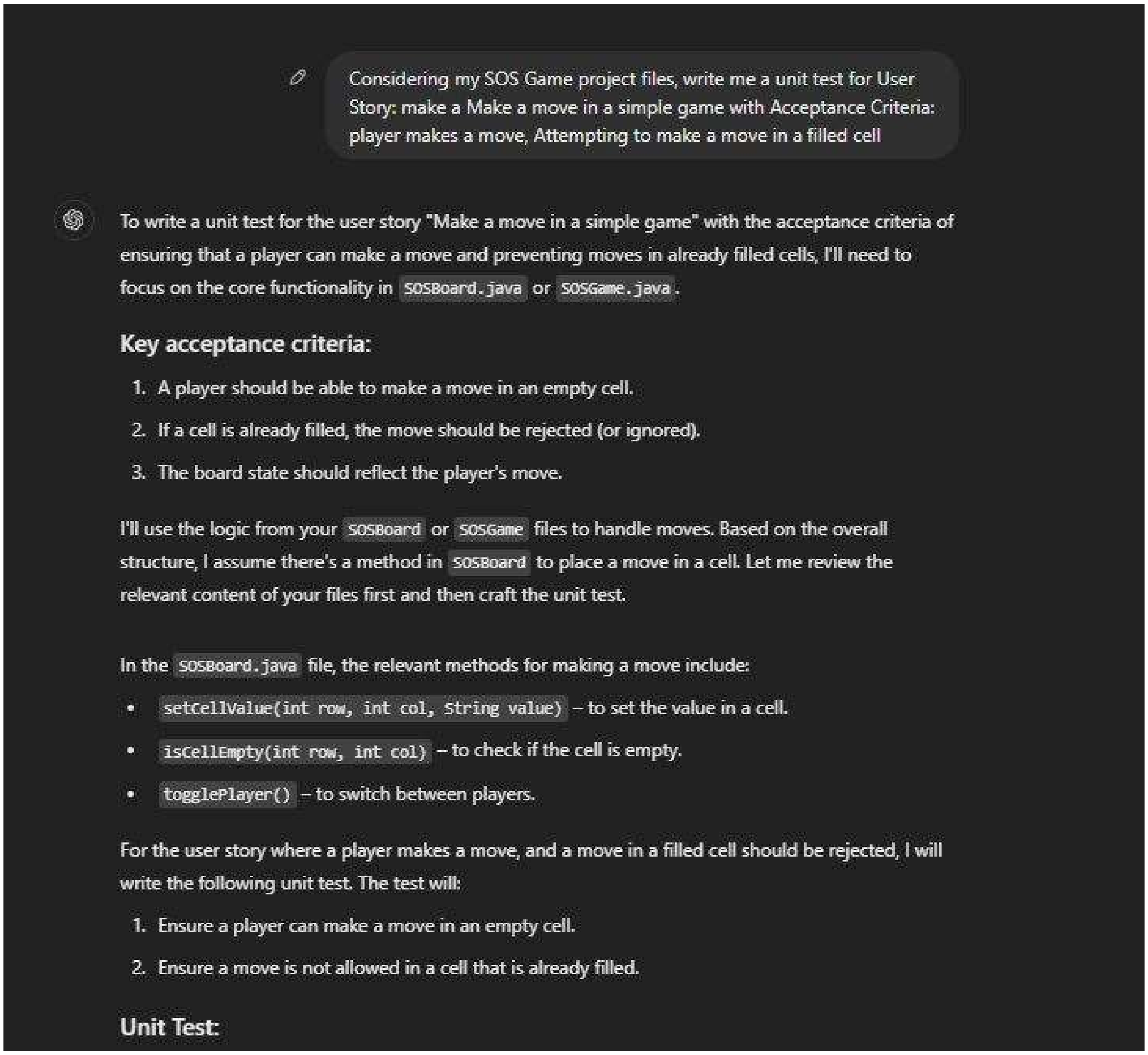
4.3 Other automated or manual tests not corresponding to the acceptance criteria of the above user stories

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

Screenshot (2 ChatGPT Unit Test Prompts and answers):

1. User Story: Make a move in a simple game

   Notes: LLM was fed my code files for analyzation. Results from the LLM where detail and met my requirements



Considering my SOS Game project files, write me a unit test for User Story: make a Make a move in a simple game with Acceptance Criteria: player makes a move, Attempting to make a move in a filled cell

To write a unit test for the user story "Make a move in a simple game" with the acceptance criteria of ensuring that a player can make a move and preventing moves in already filled cells, I'll need to focus on the core functionality in `SOSBoard.java` or `SOSGame.java`.

Key acceptance criteria:

1. A player should be able to make a move in an empty cell.
2. If a cell is already filled, the move should be rejected (or ignored).
3. The board state should reflect the player's move.

I'll use the logic from your `SOSBoard` or `SOSGame` files to handle moves. Based on the overall structure, I assume there's a method in `SOSBoard` to place a move in a cell. Let me review the relevant content of your files first and then craft the unit test.

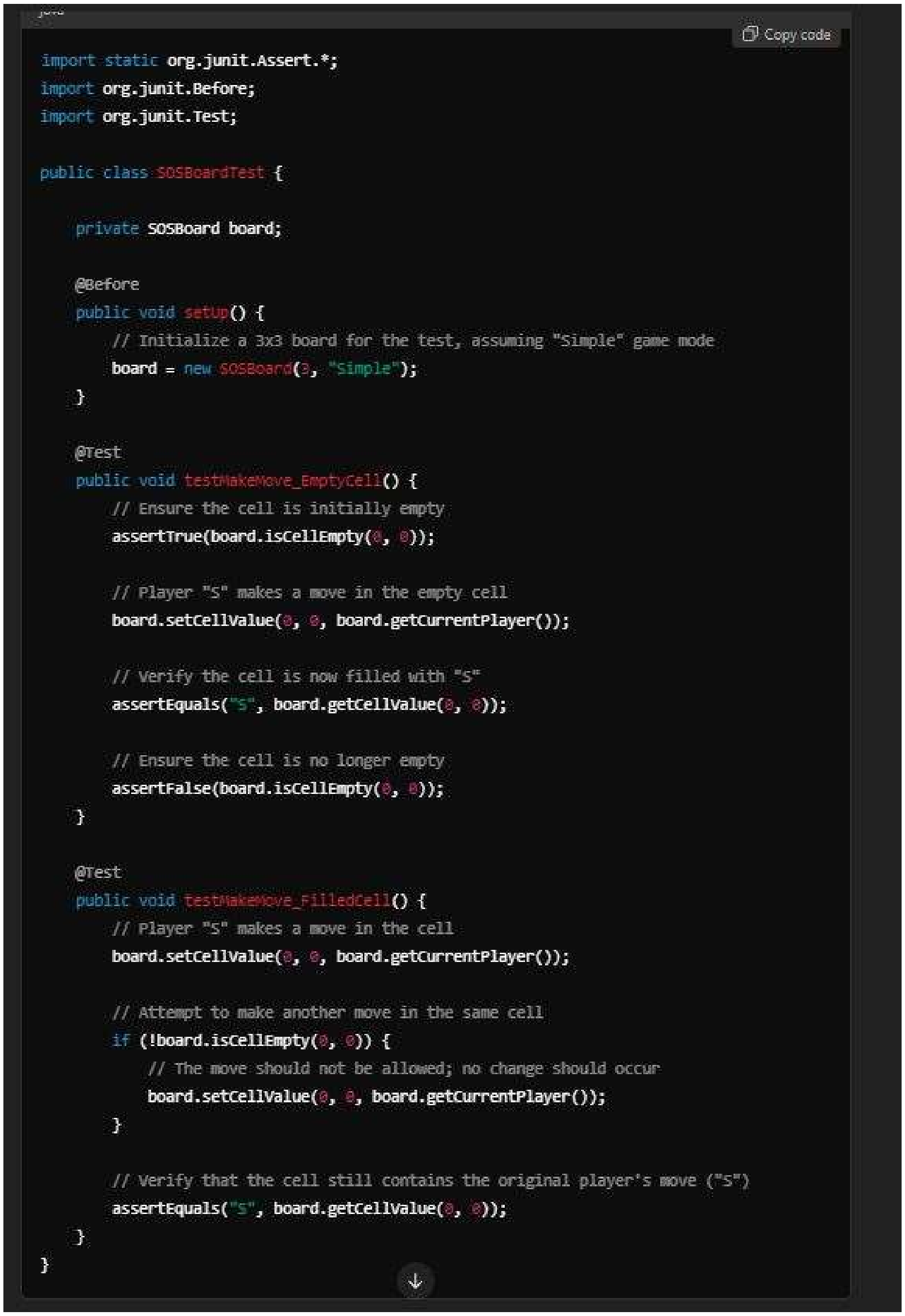
In the `SOSBoard.java` file, the relevant methods for making a move include:

- `setCellValue(int row, int col, String value)` – to set the value in a cell.
- `isCellEmpty(int row, int col)` – to check if the cell is empty.
- `togglePlayer()` – to switch between players.

For the user story where a player makes a move, and a move in a filled cell should be rejected, I will write the following unit test. The test will:

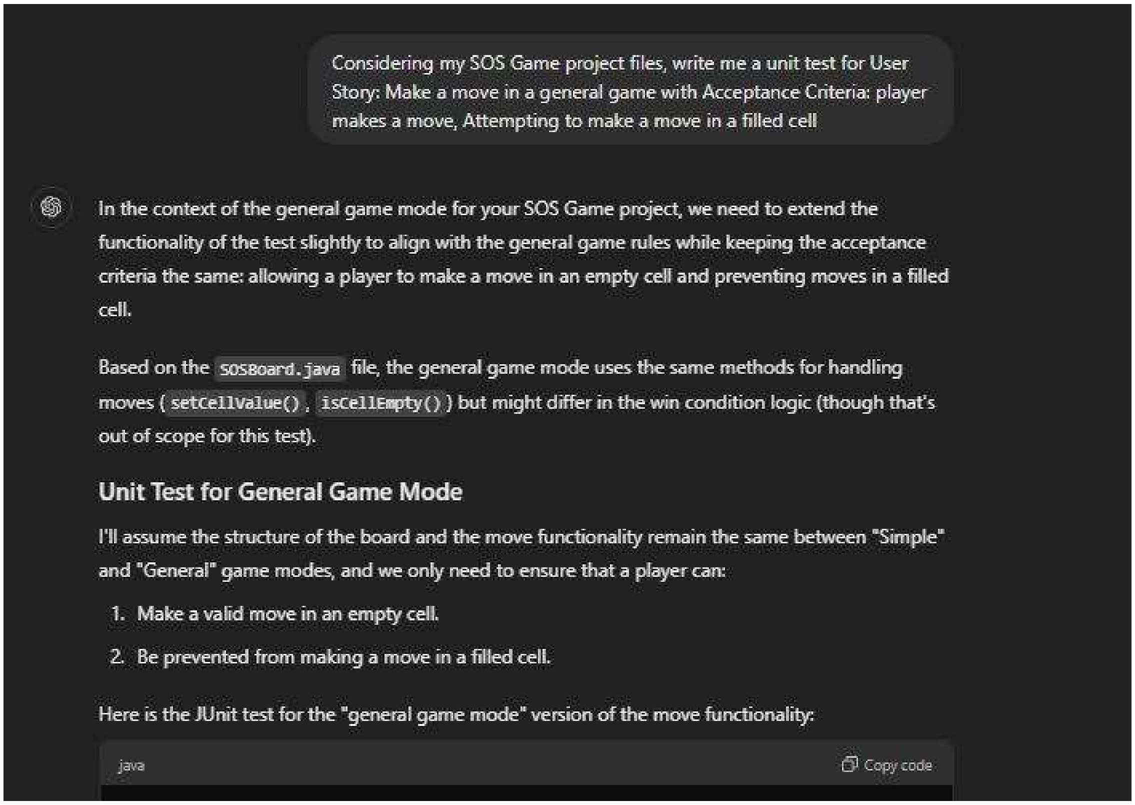
1. Ensure a player can make a move in an empty cell.
2. Ensure a move is not allowed in a cell that is already filled.

Unit Test:



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

public class SOSBoardTest {

    private SOSBoard board;

    @Before
    public void setUp() {
        // Initialize a 3x3 board for the test, assuming "Simple" game mode
        board = new SOSBoard(3, "Simple");
    }

    @Test
    public void testMakeMove_EmptyCell() {
        // Ensure the cell is initially empty
        assertTrue(board.isCellEmpty(0, 0));

        // Player "S" makes a move in the empty cell
        board.setCellValue(0, 0, board.getCurrentPlayer());

        // Verify the cell is now filled with "S"
        assertEquals("S", board.getCellValue(0, 0));

        // Ensure the cell is no longer empty
        assertFalse(board.isCellEmpty(0, 0));
    }

    @Test
    public void testMakeMove_FilledCell() {
        // Player "S" makes a move in the cell
        board.setCellValue(0, 0, board.getCurrentPlayer());

        // Attempt to make another move in the same cell
        if (!board.isCellEmpty(0, 0)) {
            // The move should not be allowed; no change should occur
            board.setCellValue(0, 0, board.getCurrentPlayer());
        }

        // Verify that the cell still contains the original player's move ("S")
        assertEquals("S", board.getCellValue(0, 0));
    }
}
```

2. User Story: Make a move in a general game
Notes: LLM was fed my code files analyzation. LLM results where simpler but met the requirements



Considering my SOS Game project files, write me a unit test for User Story: Make a move in a general game with Acceptance Criteria: player makes a move, Attempting to make a move in a filled cell

In the context of the general game mode for your SOS Game project, we need to extend the functionality of the test slightly to align with the general game rules while keeping the acceptance criteria the same: allowing a player to make a move in an empty cell and preventing moves in a filled cell.

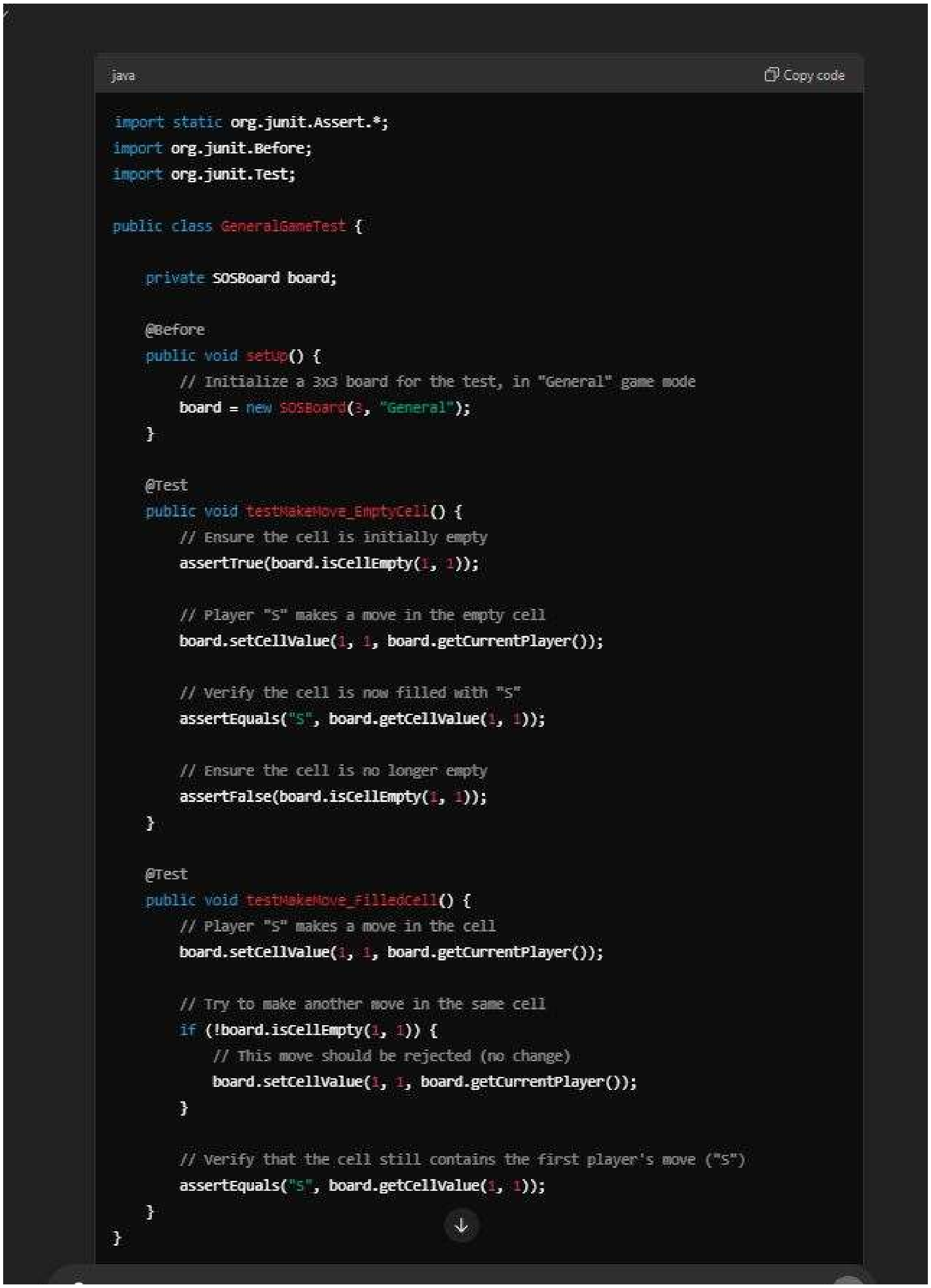
Based on the `SOSBoard.java` file, the general game mode uses the same methods for handling moves (`setCellValue()`, `isCellEmpty()`) but might differ in the win condition logic (though that's out of scope for this test).

Unit Test for General Game Mode

I'll assume the structure of the board and the move functionality remain the same between "Simple" and "General" game modes, and we only need to ensure that a player can:

1. Make a valid move in an empty cell.
2. Be prevented from making a move in a filled cell.

Here is the JUnit test for the "general game mode" version of the move functionality:



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

public class GeneralGameTest {

    private SOSBoard board;

    @Before
    public void setUp() {
        // Initialize a 3x3 board for the test, in "General" game mode
        board = new SOSBoard(3, "General");
    }

    @Test
    public void testMakeMove_EmptyCell() {
        // Ensure the cell is initially empty
        assertTrue(board.isCellEmpty(1, 1));

        // Player "S" makes a move in the empty cell
        board.setCellValue(1, 1, board.getCurrentPlayer());

        // Verify the cell is now filled with "S"
        assertEquals("S", board.getCellValue(1, 1));

        // Ensure the cell is no longer empty
        assertFalse(board.isCellEmpty(1, 1));
    }

    @Test
    public void testMakeMove_FilledCell() {
        // Player "S" makes a move in the cell
        board.setCellValue(1, 1, board.getCurrentPlayer());

        // Try to make another move in the same cell
        if (!board.isCellEmpty(1, 1)) {
            // This move should be rejected (no change)
            board.setCellValue(1, 1, board.getCurrentPlayer());
        }

        // Verify that the cell still contains the first player's move ("S")
        assertEquals("S", board.getCellValue(1, 1));
    }
}
```