Hashim Abdulla
Due 10/21

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/HashAbdulla/SOS-Board-game

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.

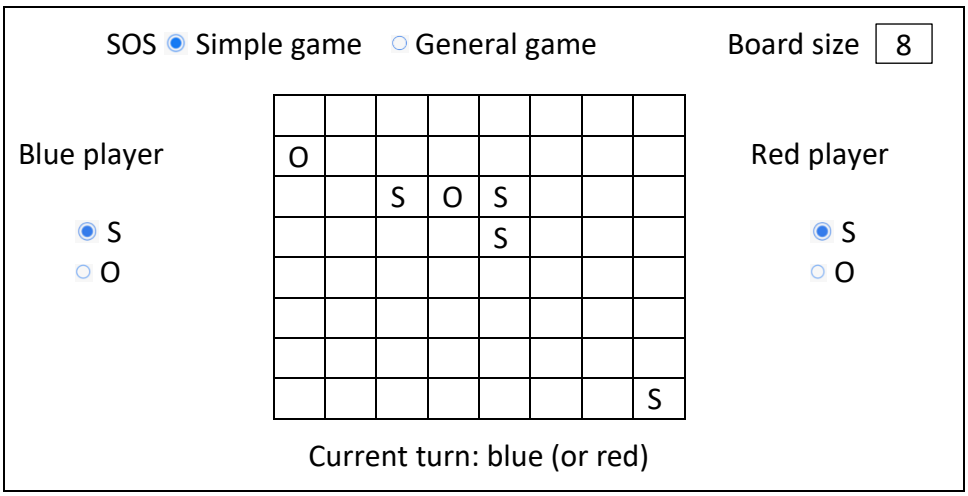


Figure 1. Sample GUI layout of the Sprint 2 program

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://www.youtube.com/watch?v=E0jRV9CArD8

	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
game_logic.py	Production	107
gui.py	Production	214
test_game.py	Test	252
Total = 573 lines of code as of Sprint #2		

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	GameBoard, SOSGame, SOSGUI	__init__(size), set_board_size(size), validate_board_size()	complete	Validates range 3-10, Spinbox widget enforces range
	1.2	GameBoard, SOSGUI	is_valid_size(size), validate_board_size(),	complete	Raises ValueError for invalid sizes, GUI shows error messagebox
2 Choose the game mode of a chosen board	2.1	SOSGame, SOSGUI	set_game_mode(mode), on_mode_change()	Complete	Radio button bound to mode_var, sets mode to SIMPLE_MODE
	2.2	SOSGame, SOSGUI	set_game_mode(mode), on_mode_change()	Complete	Radio button bound to mode_var, sets mode to GENERAL_MODE
3 Start a new game	3.1	SOSGame, GameBoard, SOSGUI	start_new_game(), __init__(size), start_new_game() (GUI)	Complete	Validates settings, creates GameBoard instance with selected size/mode

	3.2	GameBoard, SOSGUI	__init__(size), create_board_display(size)	Complete	Grid initialized with ' ' (empty), GUI creates button grid
	3.3	SOSGame, SOSGUI	start_new_game(), update_turn_label()	Complete	current_player set to blue_player, label displays "Current turn: blue" (or red)
4 Make a move in simple game	4.1	SOSGame, GameBoard, SOSGUI	make_move(row, col, letter), place_letter(), on_cell_click()	Complete	Validates empty cell, places letter, calls switch_turn(), updates button
	4.2	GameBoard, SOSGUI	test_is_cell_empty(), place_letter(), on_cell_click()	Complete	Raises ValueError if cell occupied, GUI shows error messagebox
	4.3	N/A	N/A	InProgress	SOS detection deferred to Sprint 3 per requirements
6 Make a move in general game	6.1	SOSGame, GameBoard, SOSGUI	make_move(row, col, letter), place_letter(), on_cell_click()	Complete	Same implementation as simple game (AC 4.1)
	6.2	GameBoard, SOSGUI	is_cell_empty(), place_letter(), on_cell_click()	Complete	Same implementation as simple game (AC 4.2)
	6.3	N/A	N/A	In progress	SOS scoring also deferred to Sprint 3 per requirements
	6.4	SOSGame	switch_turn()	Complete	Implemented in make_move(), switches between blue and red players

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.

(table below)

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	TestGameBoard	test_valid_board_size_creation	Input: size=5 Expected: GameBoard created with size=5, grid is 5x5 matrix of empty cells
	1.2	TestGameBoard	test_invalid_board_size_too_small	Input: size=2 Expected: ValueError raised with message "Board size must be between 3 and 10"
	1.2	TestGameBoard	test_invalid_board_size_too_large	Input: size=15 Expected: ValueError raised with message "Board size must be between 3 and 10"
	1.1	TestSOSGame	test_set_valid_board_size	Input: set_board_size(7) Expected: game.board_size == 7
	1.2	TestSOSGame	test_set_invalid_board_size	Input: set_board_size(1) and set_board_size(20) Expected: ValueError raised for both
2 Choose the game mode	2.1	TestSOSGame	test_set_simple_game_mode	Input: set_game_mode(SOSGame.SIMPLE_MODE) Expected: game.game_mode == "Simple"
	2.2	TestSOSGame	test_set_general_game_mode	Input: set_game_mode(SOSGame.GENERAL_MODE) Expected: game.game_mode == "General"
3 Start a new game	3.1	TestSOSGame	test_start_new_game	Input: set_board_size(5), set_game_mode(SIMPLE), start_new_game() Expected: board created with size=5, game_started=True
	3.2	TestGameBoard	test_board_initialized_empty	Input: GameBoard(4) Expected: All cells in 4x4 grid contain ' ' (empty)
	3.3	TestSOSGame	test_initial_turn_is_blue	Input: start_new_game() Expected: current_player.color == "blue"
4 Make a move in simple game	4.1	TestSOSGame	test_make_valid_move	Input: start_new_game(), make_move(0, 0, 'S') Expected: Cell (0,0) contains 'S', turn switches to red player
	4.1	TestGameBoard	test_place_letter_valid	Input: place_letter(1, 1, 'S') Expected: get_cell(1,1) returns 'S'
	4.2	TestSOSGame	test_make_move_on_occupied_cell	Input: make_move(0,0,'S') then make_move(0,0,'O') Expected: Second call raises ValueError "Cell is already occupied"
	4.2	TestGameBoard	test_place_letter_on_occupied_cell	Input: place_letter(0,0,'S') then place_letter(0,0,'O') Expected: Second call raises ValueError
6 Make a move in	6.1	TestSOSGame	test_make_valid_move	Input: Same as AC 4.1 Expected: Same behavior for both modes

general game				
	6.2	TestSOSGame	test_make_move_on_occupied_cell	Input: Same as AC 4.2 Expected: Same behavior for both modes
	6.4	TestSOSGame	test_turn_switching	Input: Multiple moves: (0,0,'S'), (0,1,'O'), (0,2,'S') Expected: Turn alternates: blue→red→blue

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 spinbox to 5, click New Game	Board displays as 5x5 grid	Visual confirmation in GUI
	1.1	Change spinbox to 10, click New Game	Board displays as 10x10 grid	Largest valid size
	1.2	Manually type "15" in spinbox, click New Game	Error dialog: "Board size must be between 3 and 10"	Spinbox should prevent this, but test manual entry
	1.2	Manually type "abc" in spinbox, click New Game	Error dialog: "Board size must be a number between 3 and 10..."	Non-numeric input validation
2 choose the game mode	2.1	Select "Simple game" radio button, click New Game	Info dialog shows "New Simple game started!"	Mode indicator confirmation
	2.2	Select "General game" radio button, click New Game	Info dialog shows "New General game started!"	Mode indicator confirmation
3 Start a new game	3.1	Set size=4, mode=Simple, click New Game	4x4 empty grid appears, info shows settings	Combined validation of size and mode
	3.2	Click New Game	All cells are white/empty, no text	Visual confirmation of empty board
	3.3	Click New Game	Turn label shows "Current turn: blue" in blue text	Blue always starts first
4 Make a move in simple game	4.1	Select S for blue, click cell (0,0)	Cell shows blue 'S', turn label changes to "Current turn: red"	Complete move cycle

	4.1	Select O for red, click cell (1,1)	Cell shows red 'O', turn label changes to "Current turn: blue"	Second move confirmation
	4.2	Place S at (0,0), then click (0,0) again	Error, user will not be allowed to click again	Move rejected
	4.2	Place O at (1,1), then click (1,1) again	Error, user will not be allowed to click again	Works for both letters
6 Make a move in general game	6.1	Start General game, place S at (0,0)	Cell shows blue 'S', turn switches	Same behavior as Simple
	6.2	In General game, place S at (0,0), click (0,0) again	Error, user will not be allowed to click again	Same validation as Simple

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

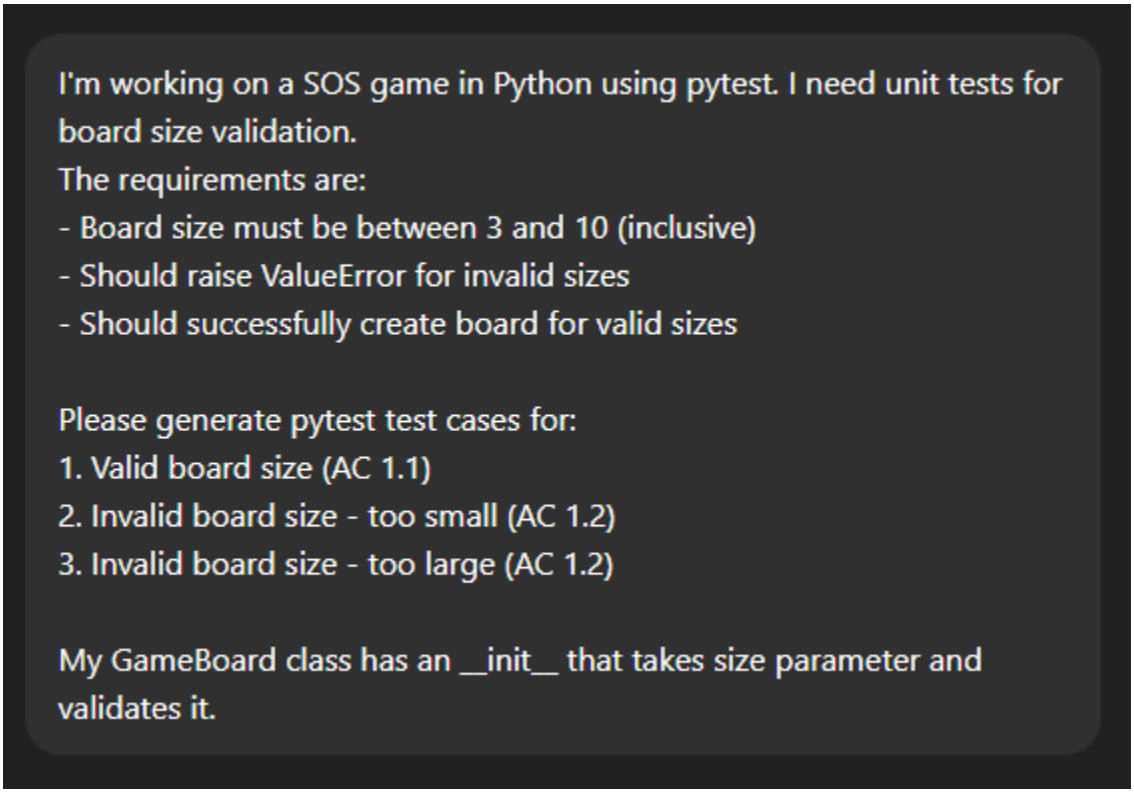
Number	Test Input	Expected Result	Class Name of the Test Code	Method Name of the Test Code
1	Check if cell is empty at various positions	Returns True for empty, False for occupied	TestGameBoard	test_is_cell_empty
2	Try to place invalid letter 'X'	ValueError: "Letter must be S or O"	TestGameBoard	test_place_invalid_letter
3	Fill entire board and check if full	is_board_full() returns True	TestGameBoard	test_board_full_detection
4	Reset board after placing letters	All cells return to ' ' (empty)	TestGameBoard	test_board_reset
5	Create player and verify attributes	Player has correct name, color, score=0	TestPlayer	test_player_creation
6	Reset player score from non-zero	Score returns to 0	TestPlayer	test_player_score_reset
7	Try to set invalid game mode "Invalid"	ValueError raised	TestSOSGame	test_set_invalid_game_mode

8	Try to make move before starting game	RuntimeError: "Please start a new game please!"	TestSOSGame	test_make_move_before_game_starts
9	Verify player scores reset on new game	Both players' scores = 0	TestSOSGame	test_players_score_reset_on_new_game
10	Multiple moves alternating turns	Turns switch: blue, red, blue, red	TestSOSGame	test_turn_switching

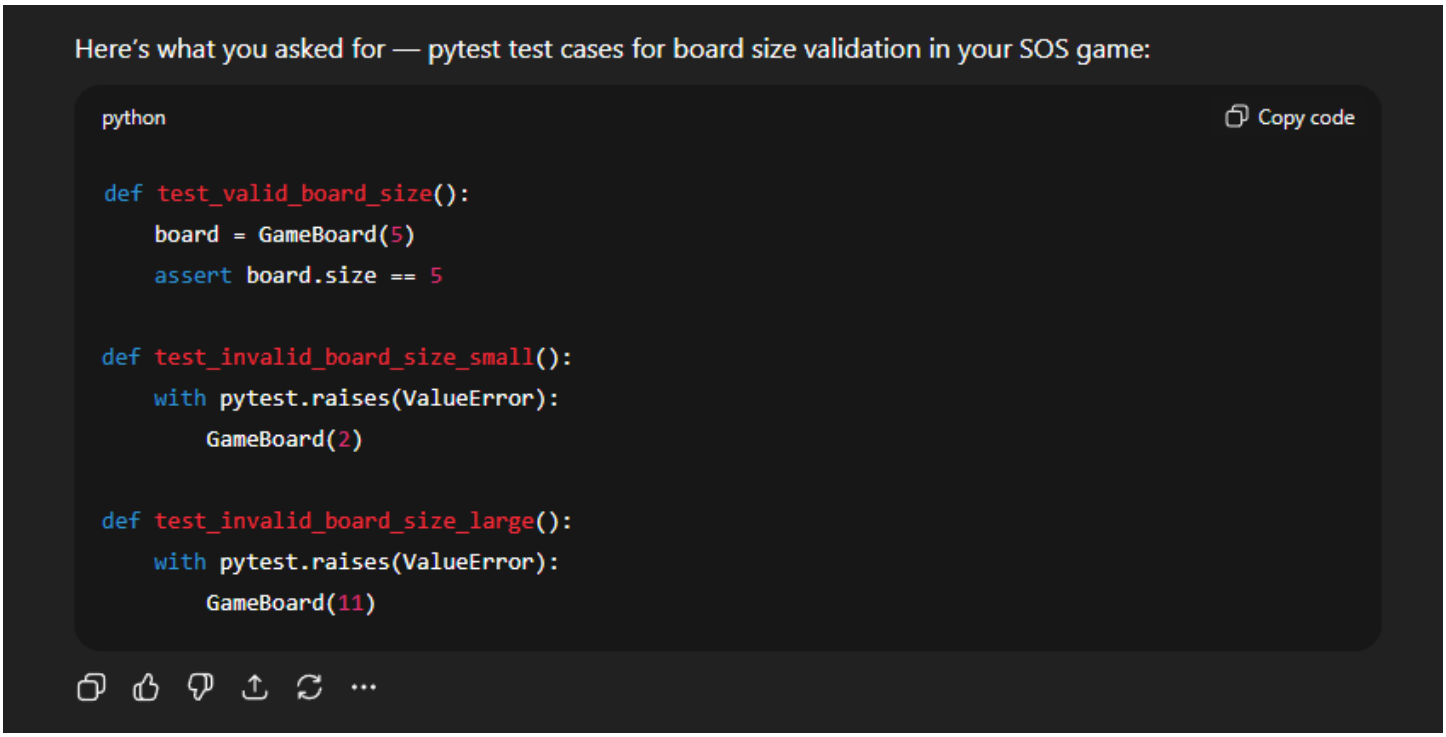
LLM PROMPTS AND FIXES:

First ChatGPT Interaction: Board Size Validation Tests

My prompt:



ChatGPT generated:



Problems I Found:

Missing test class structure: ChatGPT didn't organize tests into a class
Incomplete test names: Names didn't clearly indicate what was being tested
Missing Given-When-Then documentation: No docstrings explaining the acceptance criteria
No assertion on grid structure: First test should verify the grid is actually created correctly
Generic error matching: Didn't specify the exact error message to match

My Fixes:

```
1. class TestGameBoard:
2.     """Test cases for GameBoard class"""
3.
4.     def test_valid_board_size_creation(self):
5.         """AC 1.1 - Valid board size selection
6.         Given: Player is on start screen
7.         When: Player selects valid board size (3-10)
8.         Then: Board should be created with that size
9.         """
10.        board = GameBoard(5)
11.        assert board.size == 5
12.        assert len(board.grid) == 5  # Added: verify grid structure
13.        assert len(board.grid[0]) == 5  # Added: verify grid columns
14.
15.    def test_invalid_board_size_too_small(self):
16.        """AC 1.2 - Invalid board size selection (too small)
17.        Given: Player is on start screen
18.        When: Player enters size less than 3
19.        Then: System should raise ValueError
20.        """
21.        # Added: specific error message matching
22.        with pytest.raises(ValueError, match="Board size must be between 3 and 10"):
23.            GameBoard(2)
24.
25.    def test_invalid_board_size_too_large(self):
```

```
        """AC 1.2 - Invalid board size selection (too large)
        Given: Player is on start screen
        When: Player enters size greater than 10
        Then: System should raise ValueError
        """
        # Added: specific error message matching
        with pytest.raises(ValueError, match="Board size must be between 3 and 10"):
            GameBoard(15)
```

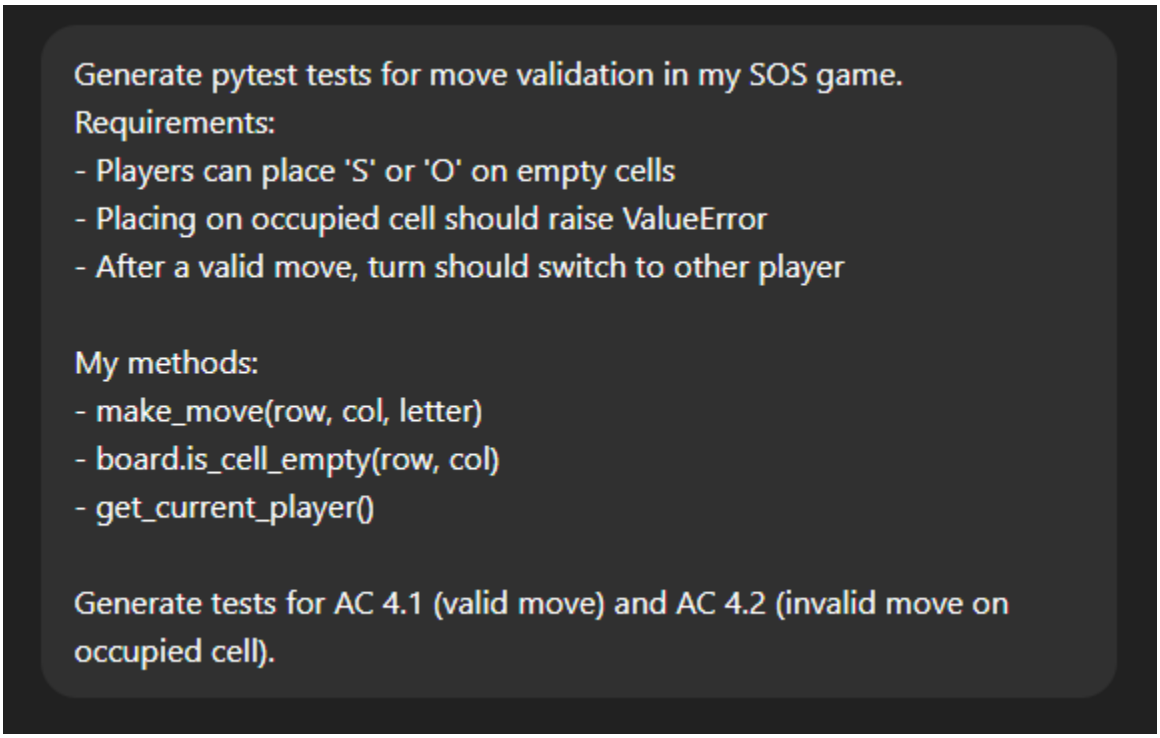
(Code outside of already-given tables in this document was formatted using the Microsoft Word "Easy Code Formatter" extension to enhance readability and presentation before converting the file to PDF.)

Why I Made These Changes:

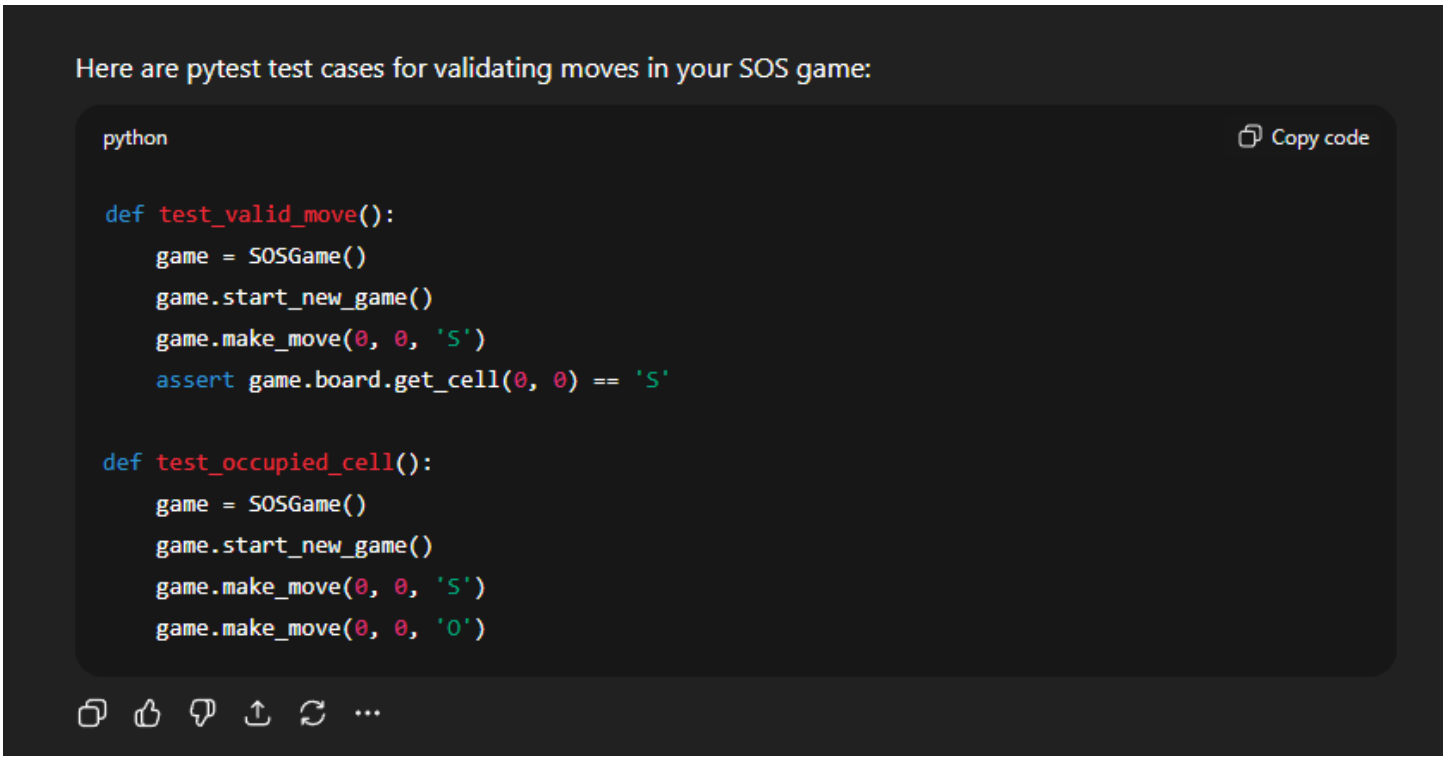
- Added test class to organize related tests
- Added comprehensive docstrings linking to acceptance criteria
- Verified not just the size attribute, but actual grid structure
- Used `match` parameter to ensure specific error messages
- Used descriptive test names that match Sprint 1 acceptance criteria

ChatGPT Interaction #2: Move Validation Tests

My Prompt:



ChatGPT generated:



Here are pytest test cases for validating moves in your SOS game:

python

Copy code

```python
def test_valid_move():
    game = SOSGame()
    game.start_new_game()
    game.make_move(0, 0, 'S')
    assert game.board.get_cell(0, 0) == 'S'

def test_occupied_cell():
    game = SOSGame()
    game.start_new_game()
    game.make_move(0, 0, 'S')
    game.make_move(0, 0, 'O')
```

Problems I Found:

Missing exception handling: Second test doesn't use pytest.raises
No turn switching verification: First test should verify turn changes
Missing player tracking: Should verify which player made the move
No Given-When-Then documentation again
Generic test names: Don't reference acceptance criteria

My Fixes:

```python
def test_make_valid_move(self):
    """AC 4.1 - Valid move placement and turn switch
    Given: It is current player's turn in a simple game
    When: Player selects S or O and clicks empty cell
    Then: System should place the selected letter in that cell and switch turns
    """
    game = SOSGame()
    game.start_new_game()

    initial_player = game.get_current_player()  # Added: track initial player
    game.make_move(0, 0, 'S')

    # Check letter was placed
    assert game.board.get_cell(0, 0) == 'S'

    # Added: Check turn switched
    assert game.get_current_player() != initial_player

def test_make_move_on_occupied_cell(self):
    """AC 4.2 - Invalid move on occupied cell"""
    game = SOSGame()
    game.start_new_game()
    game.make_move(0, 0, 'S')

```

```
    # Added: proper exception handling
    with pytest.raises(ValueError, match="Cell is already occupied"):
        game.make_move(0, 0, 'O')

```

Why I Made These Changes:

- Added proper exception handling with `pytest.raises`
- Verified turn switching behavior (critical game mechanic)
- Added Given-When-Then documentation matching my Sprint 1 AC
- Used specific error message matching
- Split into clear arrange-act-assert sections