**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/SalmanM1/CS449**](https://github.com/SalmanM1/CS449)

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.

|  |  |  |
| --- | --- | --- |
| SOS Icon  Description automatically generated Simple game Icon  Description automatically generated General game Board size  8 | | |
| Blue player  Icon  Description automatically generated S  Icon  Description automatically generated O | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  | | O |  |  |  |  |  |  |  | |  |  | S | O | S |  |  |  | |  |  |  |  | S |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | S | | Red player  Icon  Description automatically generated S  Icon  Description automatically generated O |
|  | Current turn: blue (or red) |  |

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://youtu.be/9nhD9IrJNB0**](https://youtu.be/9nhD9IrJNB0)

|  |  |
| --- | --- |
|  | **Feature** |
| 1 | Choose board size |
| 2 | Choose game mode |
| 3 | Initial game of the chosen board size and game mode |
| 4 | “S” moves |
| 5 | “O” moves |
| 6 | Automated unit tests |
| 7 | Turn indication and management |
| 8 | Color-coded player moves |
| 9 | Letter selection buttons |
| 10 | Game ends when board is full |
| 11 | Input validation and error handling |

1. **Summary of Source Code (1 points)**

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| game.py | Production Code | 94 |
| game\_ui.py | Production Code | 211 |
| test\_game.py | Test Code | 98 |
| Total | | 403 |

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

1. **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 | GameUI | create\_widgets(), start\_game() | Complete | User enters board size via an input field (Entry widget) instead of a dropdown list |
|  | 1.2 | GameUI | start\_game(), messagebox.showerror() | Complete | Validates that board size > 2; displays error message if invalid |
| 2. Choose the game mode of a chosen board | 2.1 | GameUI | create\_widgets(), start\_game() | Complete | User selects game mode via radio buttons ('Simple' or 'General') |
| 3. Start a new game of the chosen board size | 3.1 | Game, GameUI | start\_new\_game(), start\_game(), create\_board() | Complete | Starts a new game with the selected board size and game mode |
| 4. Make a move in a simple game | 4.1 | Game, GameUI | select\_letter(), on\_cell\_click(), is\_move\_valid(), make\_move(), messagebox.showwarning() | Complete | Players select 'S' or 'O' and place it on an empty cell; invalid moves are rejected |
| 5. A simple game is over | 5.1 | Game | (To be implemented) | toDo | SOS detection and victory condition not yet implemented |
|  | 5.2 | Game, GameUI | check\_game\_over(), messagebox.showinfo() | Partial | Game ends when the board is full; however, SOS detection is not yet implemented |
| 6. Make a move in a general game | 6.1 | Game, GameUI | select\_letter(), on\_cell\_click(), is\_move\_valid(), make\_move(), messagebox.showwarning() | Complete | Similar functionality to the simple game; players make moves on empty cells |
|  | 6.2 | Game | (To be implemented) | toDo | SOS detection and granting extra turns not yet implemented |
| 7. A general game is over | 7.1 | Game | (To be implemented) | toDo | Counting SOS sequences and determining the winner not yet implemented |
|  | 7.2 | Game | (To be implemented) | toDo | Determining a draw based on equal SOS counts not yet implemented |

1. **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)** |
| 3. Start a new game of the chosen board size and game mode | 3.1 | TestGame | test\_start\_new\_game\_initialization | Input: Call start\_new\_game() method. Expected Output: The game board is initialized to an empty state, and the current player is set to 'Blue'. |
| 4. Make a move in a simple game | 4.1 | TestGame | test\_move\_on\_occupied\_cell | Input: Make a move at cell (0, 0), then attempt to make another move at the same cell (0, 0). Expected Output: The first move is successful, the second move returns False, indicating the move is invalid. |

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 | Enter a valid board size (e.g., 5) | Game starts with a 5x5 board | Tested via GUI interaction |
|  | 1.2 | Enter an invalid board size (e.g., 2 or abc) | Error message displayed; game does not start | Tested via GUI interaction |
| 2. Choose the game mode of a chosen board | 2.1 | Select 'Simple' or 'General' game mode | Game starts in the selected game mode | Tested via GUI interaction |
| 4. Make a move in a simple game | 4.1 | Attempt to place a letter on an occupied cell via GUI | Warning message displayed; move is not allowed | Observed during gameplay |

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 | Filling the board completely without forming any SOS sequences | Game ends with a draw | TestGame | test\_check\_game\_over\_full\_board |
| 2 | Checking that the switch\_player method correctly alternates players | Current player switches between 'Blue' and 'Red' after each move | TestGame | test\_switch\_player |
| 3 | Retrieving the current game mode using get\_game\_mode() | Returns the correct game mode ('simple' or 'general') | TestGame | test\_get\_game\_mode |

**A screenshot of a computer program

Description automatically generatedChatGPT Unit Tests**

In the first screenshot, I prompted ChatGPT to write a unit test that verifies a player cannot make a move on an occupied cell and that the make\_move method returns False in such a case. ChatGPT provided a test where the Game class was instantiated without parameters, and the make\_move method was called with a player argument.

Upon reviewing the generated code, I noticed some discrepancies:

Instantiation Issue: In my Game class, the constructor requires parameters for board\_size and game\_mode. ChatGPT's code instantiated Game() without any arguments.

Method Signature Mismatch: The make\_move method in my Game class does not accept a player parameter; it manages the current player internally.

To correct these errors, I modified the test code as follows:

Instantiation Correction: Updated the setUp method to instantiate Game with the required parameters:

self.game = Game(board\_size=3, game\_mode='simple').

Method Call Correction: Removed the player argument from the make\_move method calls since my method does not require it. After making these corrections, the test accurately reflects my game's logic and successfully verifies that attempting to make a move on an occupied cell returns False.

A screenshot of a computer

Description automatically generated

In the second screenshot, I asked ChatGPT to create a unit test that checks whether the start\_new\_game method initializes the game board to an empty state and sets the current player to Blue. ChatGPT provided a test that again instantiated the Game class without parameters and hard-coded the expected board state.

I identified a few issues with the generated test:

Instantiation Issue: Similar to the first test, the Game class was instantiated without the required parameters.

Hard-Coded Board State: The expected board was hard-coded with dimensions that might not match the actual board size used in the test.

To address these issues, I made the following corrections:

Instantiation Correction: Updated the setUp method to include the necessary parameters:

self.game = Game(board\_size=3, game\_mode='simple').

Dynamic Board Generation: Replaced the hard-coded expected board with a dynamic generation using a list comprehension to match the board size: expected\_board = [[None for \_ in range(3)] for \_ in range(3)].

These adjustments ensured that the test accurately verifies that the start\_new\_game method initializes the board correctly and sets the current player to Blue.