**Project Sprint #3**

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

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**.

|  |  |  |
| --- | --- | --- |
| 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 | Chart, line chart  Description automatically generated | Red player  Icon  Description automatically generated S  Icon  Description automatically generated O |
|  | Current turn: blue (or red) | New Game |

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.

1. A simple game that the blue player is the winner
2. A simple draw game with the same board size as (a)
3. A general game that the red player is the winner, and the board size is different from (a)
4. A general draw game with the same board size as (c)
5. Some automated unit tests for the simple game mode
6. Some automated unit tests for the general game mode

In the video, you must explain what is being demonstrated.

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

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| game.py | Production | 227 |
| game\_ui.py | Production | 258 |
| test\_game.py | Test | 216 |
| Total | | 701 |

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

1. **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 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 | Users enter board size via Entry widget |
|  | 1.2 | GameUI | start\_game(), messagebox.showerror() | Complete | Validates board size > 2; shows error if invalid |
| 2. Choose the game mode of a chosen board | 2.1 | GameUI | create\_widgets(), start\_game() | Complete | Game mode selected via radio buttons |
| 3. Start a new game of the chosen board size | 3.1 | GameUI, SimpleGame, GeneralGame | start\_game(), start\_new\_game(), create\_game\_area(), create\_board() | Complete | Starts new game with selected options |
| 4. Make a move in a simple game | 4.1 | GameUI, SimpleGame | 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 |
|  | 4.2 | SimpleGame | check\_for\_sos(), check\_for\_sos\_s(), switch\_player() | Complete | Game detects SOS sequences and updates game state accordingly |
| 5. A simple game is over | 5.1 | SimpleGame | make\_move(), check\_game\_over() | Complete | Game ends immediately when SOS is formed |
|  | 5.2 | GameUI | update\_turn\_label(), messagebox.showinfo() | Complete | Game over message displayed; winner announced |
| 6. Make a move in a general game | 6.1 | GameUI, GeneralGame | select\_letter(), on\_cell\_click(), make\_move(), is\_move\_valid(), messagebox.showwarning() | Complete | Similar to simple game; players make moves |
|  | 6.2 | GeneralGame | make\_move(), check\_for\_sos(), check\_for\_sos\_s() | Complete | SOS detection and extra turns implemented |
| 7. A general game is over | 7.1 | GeneralGame | check\_game\_over(), make\_move() | Complete | Game ends when board is full; counts SOS sequences |
|  | 7.2 | GameUI | update\_turn\_label(), messagebox.showinfo() | Complete | Winner determined based on SOS counts; draw if equal. |

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.

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 | 1.1 | TestBaseGame | test\_initialization\_valid() | Input: Valid board size (e.g., 3). Expected: Game initializes correctly. |
|  | 1.2 | TestBaseGame | test\_initialization\_invalid() | Input: Invalid board size (e.g., 2). Expected: Raises ValueError. |
| 2 | 2.1 | TestSimpleGame | |  | | --- | |  |   test\_initialization() | Input: Start game with 'simple' mode. Expected: Game mode is 'simple'. |
|  | 2.1 (cont.) | TestGeneralGame | test\_initialization() | Input: Start game with 'general' mode. Expected: Game mode is 'general'. |
| 3 | 3.1 | TestSimpleGame | test\_start\_new\_game() | Input: Call start\_new\_game(). Expected: Board reset; current player is 'Blue' (as for my configuration Blue is the starting player by default) |
| 4 | 4.1 | TestSimpleGame | test\_make\_move\_valid(),  test\_make\_move\_invalid() | Input: Valid and invalid moves. Expected: Valid moves update board; invalid moves are rejected. |
|  | 4.2 | TestSimpleGame | test\_sos\_detection() | Input: Moves that form an SOS. Expected: Game detects SOS and ends. |
| 5 | 5.1 | TestSimpleGame | test\_game\_over\_with\_winner() | Input: Moves that lead to game over. Expected: Game ends; correct winner declared. |
|  | 5.2 | TestSimpleGame | test\_check\_game\_over\_full\_board() | Input: Fill board without SOS. Expected: Game ends in a draw. |
| 6 | 6.1 | TestGeneralGame | test\_make\_move\_with\_sos() | Input: Moves forming SOS in general game. Expected: Player gets extra turn. |
|  | 6.2 | TestGeneralGame | test\_current\_player\_does\_not\_change\_after\_sos() | Input: Player forms SOS. Expected: Current player remains the same. |
| 7 | 7.1 | TestGeneralGame | test\_game\_over\_and\_winner() | Input: Game progresses to completion. Expected: Winner determined based on SOS counts. |

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 | Enter valid board size (e.g., 5) | Game starts with a 5x5 board | Tested via GUI |
|  | 1.2 | Enter invalid board size (e.g., 2, 'abc') | Error message displayed; game does not start | Tested via GUI |
| 2 | 2.1 | Select 'Simple' or 'General' game mode | Game starts in the selected game mode | Tested via GUI |
| 4 | 4.1 | Attempt to place letter on occupied cell | Warning message displayed; move not allowed | Observed during gameplay |
| 5 | 5.1 | Form an SOS sequence in simple game | Game ends immediately; winner announced | Observed during gameplay |
| 6 | 6.1 | Form an SOS sequence in general game | Player gets extra turn; game continues | Observed during gameplay |
| 7 | 7.1 | Complete general game until board is full | Game ends; winner determined based on SOS counts | Observed during gameplay |

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** |
| 1 | Attempt to make move with invalid letter (e.g., 'X') | Move rejected; method returns False | TestSimpleGame | test\_make\_move\_invalid\_letter() |
| 2 | Attempt out-of-bounds move (e.g., (-1, 0)) | Move invalid; method returns False | TestSimpleGame | test\_is\_move\_valid\_out\_of\_bounds() |
| 3 | Check that player does not switch after invalid move | Current player remains the same | TestSimpleGame | test\_current\_player\_does\_not\_change\_on\_invalid\_move() |

1. **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 designing the class hierarchy for the SOS game I used an abstract base class “BaseGame” as the foundation for both game types.

**BaseGame Class (Abstract Base Class)**

The BaseGame class serves as the backbone, containing the core attributes and methods that both game modes need. Primary attributes include:

* board\_size, which sets the dimensions of the game board
* board, a 2D list that holds the current state of the game
* current\_player, a tracker for whose turn it is, switching between 'Blue' and 'Red'
* game**\_**over, a flag to indicate when the game has finished
* winner, which stores the winner once the game ends
* blue\_sequences and red\_sequences, which count each player’s SOS sequences

BaseGame also has several methods: \_\_init\_\_ for setup, start\_new\_game to reset the game, is\_move\_valid to verify if moves are allowed, switch\_player to alternate turns, and SOS-specific methods like check\_for\_sos\_s and check\_for\_sos, which check for an SOS sequence based on the placed letter. I also included two abstract methods, make\_move and check\_game\_over, that each game mode implements in its own way.

**SimpleGame Class (Extending BaseGame)**

The SimpleGame class inherits from BaseGame but modifies it to match the rules of a basic SOS game, where the game ends as soon as a player forms an SOS. The game stops immediately when a player forms an SOS sequence, in which the player who forms the SOS is the winner, that ends the game right away. I also implemented make\_move and check\_game\_over to detect when an SOS is formed, handling game conclusion when the board isn’t full.

**GeneralGame Class (Extending BaseGame)**

The GeneralGame class is built upon BaseGame where the game continues until the board is entirely filled. The elements of this mode include forming an SOS doesn’t stop the game; it continues until all cells are filled. When a player forms an SOS, they get an additional turn. The player with the most SOS sequences wins when the game ends. Here, make\_move manages the extra-turn logic, and check\_game\_over triggers when the board is full, at which point the player with the most sequences is declared the winner.

**A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generatedChatGPT Responses and Corrections:**

I used free mode for some automated unit tests and asked ChatGPT to write unit tests on my SimpleGame class. The responses provided by ChatGPT were poorly answered and needed correction. One was to test that if the game tries to make a move on the cell occupied by either 'X' or 'O', then it should return False and should not affect the current\_player. Instead of a unit test, the answer from ChatGPT gave some sort of implementation of the SimpleGame class. So, I had to create by hand a test which would check that the game doesn't allow making a move on an already occupied cell, doesn't change the current\_player, and the board stays intact after an invalid move.

I was asking ChatGPT to provide a unit test to check for an SOS in diagonal, and it should have tested that game\_over is set to True and the winning player is correct. Instead, it provided additional class code instead of the unit test that I asked for. I also had to manually create a test that would simulate the moves that would create an SOS diagonally, check that the game is over, and correctly assign the winning player. The final code includes these corrections, along with screenshots showing the initial prompts, and responses given by ChatGPT.