**Project Sprint #4**

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/sidnsude10/CS449**

Implement all the features that support a player (**human or computer**) to play a simple or general SOS game against another player (**human or** **computer**). The minimum features include **choosing human or computer for red and/or blue players**, **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 computer component must be able to play complete simple and general games. You are encouraged to consider basic strategies for winning simple or general games (e.g., against a poor human player). Optimal play is not required.

The following is a sample GUI layout. You must use a class hierarchy to deal with the computer opponent requirements. If your current code has not yet 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 Human  Icon  Description automatically generated S  Icon  Description automatically generated O  Icon                          Description automatically generated Computer | Chart, line chart  Description automatically generated | Red player  Icon  Description automatically generated Human  Icon  Description automatically generated S  Icon  Description automatically generated O  Icon  Description automatically generated Computer |
|  | Current turn: blue (or red) | New Game |

Figure 1. Sample GUI layout of the working program for Sprint 4

**Total points: 24**

1. **Demonstration (8 points)**

Submit a link to a video of no more than five minutes, clearly demonstrating that you have implemented the computer opponent features and written some automated unit tests. No points will be given without a video link.

**YouTube/Panopto link:**

1. A complete simple game where the blue player is a human, the red player is the computer, and there is a winner
2. A complete general game where the blue player is the computer, the red player is a human, and there is a winner
3. A complete simple game where both sides are played by the computer
4. A complete general game where both sides are played by the computer
5. Some automated unit tests for the computer opponent.

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

1. **User Stories for the Computer Opponent Requirements (1 points)**

* **User Story Template**: As a <role>, I want <goal> [so that <benefit>]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **User Story Name** | **User Story Description** | **Priority** | **Estimated effort (hours)** |
| 8 | Play Against Computer | As a player, I want to play against a computer opponent so that I can practice alone. | 1 | 4 |
| .9 | Computer vs Computer | As a developer, I want two computers to play against each other so I can demo auto-play. | 2 | 3 |
|  |  |  |  |  |

1. **Acceptance Criteria (AC) for the Computer Opponent Requirements (4 points)**

Add or delete rows as needed.

|  |  |  |  |
| --- | --- | --- | --- |
| **User Story ID and Name** | **AC**  **ID** | **Description of Acceptance Criterion** | **Status (completed, toDo, inPprogress)** |
| 8 Play against Computer | 8.1 | The game must allow a human to play against a computer. | completed |
| 8.2 | The computer must be able to play as the first player (Blue) | completed |
| 8.3 | The computer must choose legal moves and not overwrite cells. | completed |
|  | 8.4 | The game ends and correctly declares a winner or draw. | completed |
|  |  |  |
| 9 Computer vs Computer | 9.1 | The game supports computer vs computer play in both modes. | completed |
|  | 9.2 | The computer alternates turns correctly without crashing. | completed |
|  | 9.3 | The game determines a valid winner or draw at the end. | completed |
|  |  |  |  |

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

|  |  |  |
| --- | --- | --- |
| **Source code file name** | **Production code or test code?** | **# of lines of code** |
| Main.py | production | 35 |
| Sos.game.py | production | 204 |
| Sos.gui.py | production | 257 |
| test\_board\_size.py | test | 33 |
| test\_computer\_player.py | test | 50 |
| test\_game\_mode.py | test | 30 |
| test\_game\_over.py | test | 57 |
| test\_general\_game.py | test | 80 |
| test\_new\_game.py | test | 30 |
| test\_simple\_game.py | test | 66 |
|  |  |  |
|  | Total | 842 |

1. **Production Code vs New User stories/Acceptance Criteria (2 points)**

Summarize how each of the new user story/acceptance criteria is implemented in your production code (class name and method name etc.)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Story ID and Name** | **AC ID** | **Class Name(s)** | **Method Name(s)** | **Status (complete or not)** | **Notes (optional)** |
| 8 Play against computer | 8.1 | ComputerPlayer, SOSGUI | make\_move(), computer\_move() | complete |  |
|  | 8.2 | ComputerPlayer | make\_move(), \_\_init\_\_() | complete |  |
|  | … |  |  |  |  |

1. **Tests vs New User stories/Acceptance Criteria (2 points)**

Summarize how each of the new user story/acceptance criteria is tested by your test code (class name and method name) or manually performed tests.

6.1 Automated tests directly corresponding to some acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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)** |
| 8 Play against Computer | 8.1 | TestComputerPlayer | test\_computer\_can\_make\_move() | Create empty board, call make\_move(), assert that an "S" or "O" appears in a valid cell |
|  |  |  |  |  |
|  |  |  |  |  |
|  | 8.2 | TestComputerPlayer | test\_computer\_random\_letter\_selection() | Call make\_move() multiple times, confirm both "S" and "O" are used over several runs |
|  |  |  |  |  |

6.2 Manual tests directly corresponding to some acceptance criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story ID and Name** | **Acceptance Criterion ID** | **Test Case Input** | **Test Oracle (Expected Output)** | **Notes** |
| 8 Play against Computer | 8.1 | Start a game with Blue as Computer | Computer makes move without human interaction |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | 8.2 | Watch several computer-only games | Mix of "S" and "O" appears on board |  |
|  |  |  |  |  |

6.3 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. **Present the class diagram of your production code (3 points) and describe how the class hierarchy in your design deals with the computer opponent requirements (3 points)**?

Player (abstract)

┌───────────────┐

│ + name │

│ + make\_move() │

└─────┬─────────┘

│

┌──────┴─────────┐

│ │

HumanPlayer ComputerPlayer

**|**

**|**

┌──────────────┐

│ + letter │

│ + make\_move() │

└──────────────┘

────────────────────────────────────────────

SOSGame

┌─────────────────────────────┐

│ + board\_size │

│ + board │

│ + current\_turn │

│ + current\_player\_name │

│ │

│ + getCell() │

│ + validateMove() │

│ + makeMove() │

│ + checkForSOS() │

│ + isGameOver() │

│ + getGameState() │

└──────┬───────────┬─────── ─┘

│ │

┌──────┘ └───────┐

SimpleSOS GeneralSOS

┌──────────────┐

│ + blue\_score │

│ + red\_score │

└──────────────┘

\* The class hierarchy uses a base Player class with two types: HumanPlayer and ComputerPlayer. This setup lets both player types share a common structure, but act differently when it’s their turn. The human player waits for input through the GUI, while the computer picks a random valid cell and letter using its own make\_move() method. This clean split makes it easy to support human vs computer, or even computer vs computer games. Since SOSGame handles the rules, and SimpleSOS / GeneralSOS handle specific game logic, the players don’t need to worry about how scoring or game-over checks work. The structure is modular, easy to test, and lines up well with Sprint 4's computer opponent goals.