**Project Sprint #3**

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/kimbrow-slice/SOSGame/tree/dev**](https://github.com/kimbrow-slice/SOSGame/tree/dev) **(using dev branch until we are ready for real production code with an executable)**

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.

<https://umsystem.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=1901aaf0-a049-4279-ba37-b2ae003afb97>

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

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| GUI App.axaml | Production | 9 |
| GUI Styles.axaml | Production | 74 |
| GUI GridSystem.cs | Production | 197 |
| GUI MainWindow.axaml | Production | 193 |
| GUI MainWindow.axaml.cs | Production | 355 |
| LOGIC BaseGame.cs | Production | 150 |
| LOGIC GameController.cs | Production | 52 |
| LOGIC GeneralGame.cs | Production | 91 |
| LOGIC SimpleGame.cs | Production | 68 |
| BaseGameTests.cs | Test | 79 |
| GameControllerTests.cs | Test | 34 |
| GeneralGameTests.cs | Test | 91 |
| GridSystemUnitTest.cs | Test | 62 |
| SimpleGameTests.cs | Test | 64 |
| Total | | 1419 |

**You must submit all source code via github 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** | **AC ID** | **Class Name(s)** | **Method Name(s)** | **Status (complete or not)** | **Notes (optional)** |
| 1 – Choose a board size | 1.1 | GameController.cs | StartGame() | Complete | Validates grid size is correct value before triggering setup |
|  | 1.2 | GridSystem.cs | SetGridSize () | Complete | Initalizing UI grid and passes all input from user into the logic |
| 2 – Choose the game mode | 2.1 | MainWindow.axaml.cs | IsGeneralGameModeSelected | Complete | Determine if a General or Simple Game is selected. |
|  | 2.2 | GridSystem.cs | SetGridSize() -> new Simplegame()/ new GeneralGame | Complete | Game object is initiated by user input |
| 3 – Start a new game | 3.1 | MainWindow.axaml.cs | NewGame() | Complete | Call from New Game Button to reset game with current settings |
|  | 3.2 | GridSystem.cs | ResetBoard(), SetGridSize() | Complete | Clears UI, provide the user with a new game board and clear the logic instance. |
| 4 – Make a move (Simple Game) | 4.1 | GridSystem.cs | OnCellClicked() | Complete | Validates move and calls into logic MakeMove() |
|  |  | SimpleGame.cs | MakeMove() | Complete | Place selected letter, check for SOS, handle scoring and freezing game state |
| 5 – Game over (Simple Game) | 5.1 | SimpleGame.cs | MakeMove(), IsFrozen - flag | Complete | Game ends on first SOS or board being full. |
|  |  | BaeGame.cs | IsGameOver() | Complete | Shared method to check for board completion. |
| 6 – Make a move (General Game) | 6.1 | GridSystem.cs | OnCellClicked() | Complete | This triggers move logic in a General Game |
|  |  | GeneralGame.cs | MakeMove() | Complete | Scores all SOSs from moves, retains turn if player scores. |
| 7 – Game over (General Game) | 7.1 | GeneralGame.cs | MakeMove(), IsGameOver() | Complete | End game when board is full, compares total SOS formation scores, then declares the winner in a banner. |

1. **Tests vs User stories/Acceptance Criteria (2 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 |
| 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 | GridSystemUnitTests | |  | | --- | | Grid\_Updates\_Size\_When\_SetGridSize\_Is\_Called\_With\_Valid\_Size() |  |  | | --- | |  | | Input: 5 -> Expected: Grid Size is updated to 5 |
|  | 1.2 | GridSystemUnitTests | |  | | --- | | SetGridSize\_Should\_Throw\_Exception\_When\_Invalid\_Size\_Is\_Given() |  |  | | --- | |  | | Input: 2, 13, -1, 0 -> Expected: Exception error “ERR:The grid size must be between 3 and 12” |
| 2 | 2.1 | GameControllerTests | |  | | --- | | Constructor\_InvalidGridSize\_ThrowsArgumentException() |  |  | | --- | |  | | Input: gridSize = 2 -> Expected: Throws ArgumentException |
|  | 2.2 | GameControllerTests | |  | | --- | | TestTurnSwitching() |  |  | | --- | |  | | Input: valid move at (0,0,’S’) -> Expected: Turn switches to next player |
| 4 | 4.1 | SimpelGameTests | MakeMove\_FormsHorizontalSOS\_ReturnsTrue() MakeMove\_FormsVerticalSOS\_ReturnsTrue() MakeMove\_FormsDiagonalSOS\_ReturnsTrue() | Inputs: valid SOS sequences in each direction -> Expected: MakeMove() returns true |
| 5 | 5.1 | BaseGameTests | MakeMove\_ToOccupiedCell\_ReturnsFalse() PlayerScores\_InitiallyZero() Turn\_Switches\_AfterMove() | Testing basic end game preconditions |

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 board size "5" into the board size text box and click "Start Game" | A 5x5 board is initialized in the UI | Grid buttons should match size, and game should accept player moves |
|  | 1.2 | Enter invalid board size "2" and click "Start Game" | Error message: "ERR: The grid size must be between 3 and 12." | Prevents board creation if invalid |
| 2 | 2.1 | Toggle "General" radio button and click "Start Game" | Game logic switches to GeneralGame instance | Controls switch game mode context for scoring logic |
| 3 | 3.1 | After first game completes, click "New Game" with updated size/mode | Game resets grid, mode, and logic instance | Button enables a full reset without restarting application |
| 4 | 4.1 | Place moves that form a horizontal SOS: S at (1,0), O at (1,1), S at (1,2) | SOS formation is detected, stroke is drawn, and scoring updates | SimpleGame ends immediately after this move |
| 5 | 5.1 | Form SOS in Simple Game | UI displays brush stroke from the player who created the formation. | Simple Game is put into a frozen state. All input elements disabled until reset |
| 6 | 6.1 | In GeneralGame, form SOS vertically and place another move that also forms SOS | Player retains turn after scoring. Score increments accordingly | Confirms that General mode supports multiple SOS and turn retention |
| 7 | 7.1 | Form multiple SOS in General Game until board is full | Game continues until board is full. Final winner determined based on total SOS count | Game only freezes once grid is filled; UI displays final message and allows reset |

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 | Call MakeMove() twice on the same cell | Second move returns false and cell remains unchanged | BaseGameTests | MakeMove\_ToOccupiedCell\_ReturnsFalse() |
| 2 | Call MakeMove() after game is over | Returns false and no changes are applied | SimpleGameTests | MakeMove\_GameIsFrozen\_ReturnsFalse() |
| 3 | Initialize GameController with grid size 2 | Throws ArgumentException | GameControllerTests | Constructor\_InvalidGridSize\_ThrowsArgumentException() |
| 4 | Toggle mode without clicking Start Game | No crash; no game object created until Start Game is clicked | Manual |  |
| 5 | User leaves board size field empty and clicks Start Game | Error message shown, no crash | Manual |  |
| 7 | Confirm player scores begin at 0 | PlayerScores[0] == 0, PlayerScores[1] == 0 | BaseGameTests | PlayerScores\_InitiallyZero() |

1. **(Part 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 page, excluding screenshots/diagrams (12pt, single-spaced), is required.** **(3 Points)**

My version of the SOS Game was created following the design principles we’ve talked about in class. Specifically, I used an object oriented class hierarchy that clearly separates the shared functionality from each game mode’s specific logic. At the core of the game is the BaseGame class, which acts as the foundation for both SimpleGame and GeneralGame. This hierarchy allows the game to handle common requirements like move validation and board tracking, while supporting distinct behaviors such as scoring and end-game logic, all in an organized and maintainable format.

The BaseGame class handles all of the shared behavior and state management used by both game modes. For example, it manages board initialization, turn tracking, move validation, scorekeeping, and helper methods like IsGameOver() to determine if the board is full. Both SimpleGame and GeneralGame inherit from BaseGame and override behavior based on their own rule sets.

For example, SimpleGame enforces a single SOS win condition. As soon as a player forms an SOS, the game immediately enters a frozen state and no more moves can be made. This logic is handled in the overridden MakeMove() method, which checks for the first SOS and stops the game once it’s found. On the other hand, GeneralGame allows for multiple SOS formations and lets the player keep their turn if they score. The GeneralGame’s version of MakeMove() checks for multiple SOS formations per move and continues until the board is completely full. I also added logic to compare final scores and determine the winner when the game is officially over.

This structure helps isolate core business logic in BaseGame while delegating game mode specific rules to the subclasses. Designing it this way gives me flexibility to add new game modes later on by extending BaseGame without having to modify the existing logic. I believe that following this structure is clean, scalable, and will align with the design goals spoken about in class. Below you can find screenshots of BaseGame class hierarchy as well as overridden MakeMove methods in both Simple and General games. **Class Hierarchy Folder Structure and screenshot of BaseGame.cs  
A screenshot of a computer program

AI-generated content may be incorrect.**

**Overridden MakeMove method in GeneralGame.cs  
A screen shot of a computer program

AI-generated content may be incorrect.  
  
Overridden MakeMove method in SimpleGame.cs  
A computer screen shot of a program code

AI-generated content may be incorrect.**

**(Part 2) Demonstrate how you use LLM (ChatGPT or other) to analyze how well your code adheres to the design principles discussed in class - modularity, cohesion, coupling, and encapsulation – using the definitions provided in class. Provide screenshots of your interactions with the LLM, showing the prompts used and the responses received.**

**Interpret the LLM’s feedback, refine your prompts if needed to obtain more relevant responses, and discuss any changes made to your code based on the analysis. Explain how these refinements improve adherence to the design principles.**

**At least 1/2 page is required, excluding screenshots (12pt, single-spaced).** **(2 Points)**

I attempted to analyze how well my SOS game implementation adhered to the design principles of modularity, cohesion, coupling, and encapsulation. I used ChatGPT4o model to assist with identify any potential areas for improvement within my program. I provided the model with critical source code files from my project, as well as the GitHub link to the dev branch to reference. Overall, the initial response aligned with the design principles and architecture you intend for us to achieve. ChatGPT responded with a suggestion to introduce action delegates or event handlers, similar to other logic I have used throughout the program, rather than providing direct access to the MainWindow itself.   
  
However, when I attempted to implement these suggestions, specifically decoupling GridSystem from MainWindow, I ran into multiple issues with the integration. While the abstraction worked in theory, it also introduced tons of bugs, such as turn switching, scoring logic, victory banner display, due to the lack of context I provided it (I believe this to be true?) Ultimately, I had to end up rolling back to a previous version of the program and reimplement the features that were missing.   
  
While the analysis that ChatGPT provided me was crucial and providing a great high level view of the architecture of my program, I still do not believe that the recommended fixes for this were correct. I will be diving further into more event handling documentation to find a better implementation for this abstraction in the background during Sprint 4 to ensure that the architecture COMPLETELY follows the design principles and best practices we discussed in class.   
  
I have provided screenshots of my prompts below. I have implemented some of this logic to carry over into the next sprint.

GPT 4o Initial Prompt and Response  
A screenshot of a computer

AI-generated content may be incorrect.

GPT4o response cont…  
A screenshot of a computer

AI-generated content may be incorrect.

GPT4o response cont…  
A screenshot of a computer

AI-generated content may be incorrect.

GPT4o response cont…

A screenshot of a computer

AI-generated content may be incorrect.

My response and GPT’s feedback  
A screenshot of a computer

AI-generated content may be incorrect.

GPT4o second response  
A screenshot of a computer

AI-generated content may be incorrect.

GPT4o second response cont…

A screenshot of a computer

AI-generated content may be incorrect.

GPT4o second response cont…  
A screenshot of a computer

AI-generated content may be incorrect.

GPT4o second response cont…  
A screenshot of a computer

AI-generated content may be incorrect.