**Project Sprint #5**

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/commits/main/**

The main tasks of this assignment are:

1. Adding the feature of recording a game into a text file(or a lightweight database) and replaying from text file.
2. Conducting a code review exercise.
3. Summarizing the lessons learned from Sprint 0 through Sprint 5.

The following is a sample GUI layout of the final product, where “Replay” is optional.

|  |  |  |
| --- | --- | --- |
| 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  Replay |
| Record game | Current turn: blue (or red) | New Game |

Figure 1. Sample GUI layout of the final product

**Total points: 16**

1. **Demonstration (10 points)**

Submit a link to a video of no more than 5 minutes, clearly demonstrating that you have implemented all the features in the following table. In the video, you must explain what is being demonstrated. No points will be given without a video link.

**YouTube/Panopto link: https://umsystem.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=cb2de90c-9688-422c-b572-b2d10019b6ba**

|  |  |
| --- | --- |
|  | **Feature** |
| 1 | A complete simple game of two human players is recorded |
| 2 | A complete general game of two human players is replayed |
| 3 | A complete simple game of human-computer players is replayed |
| 4 | A complete general game of human-computer players is recorded |
| 5 | A complete simple game of computer-computer players is recorded |
| 6 | A complete general game of computer-computer players is replayed |

1. **User Stories and Acceptance Criteria for the Record/Replay Requirements (1 points)**

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

Add or delete rows as needed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **User Story Name** | **User Story Description** | **Priority** | **Estimated effort (hours)** |
| 20 | Save Game Feature | As a player, I want to save the current game state to a CSV file so that I can use this data later. | High | <1 Hours |
| 21 | Replay Game Feature | As a player, I want to upload the saved game data to replay the game. | High | 2 Hours |

|  |  |  |  |
| --- | --- | --- | --- |
| **User Story ID and Name** | **AC**  **ID** | **Description of Acceptance Criterion** | **Status (completed, toDo, inPprogress)** |
| 20 Save Game Feature | 20.1 | AC 20.1 After the user clicks the “Save Game” button, the game data is saved to a CSV file. | Completed |
| 20.2 | AC 20.2 The CSV file includes the player’s name “listed as Red/Blue Player”, and color. Following this, it contains the player, row, column, and letter placed. | Completed |
| 21 Replay Game Feature | 21.1 | AC 21.1 After the user clicks the “Replay” button, the game allows the user to input game data saved to a CSV file. | Completed |
| 21.2 | AC 21.2 The CSV file once loaded should replay the entire game completed game for the user. | Completed |
| 21.3 | AC 21.3 The CSV file once loaded should allow the user to pick up in a game where they left off. | Completed |

1. **Code Review (2 points)**

Apply source code review to one or two most important classes (and other classes if time permits) and report the findings. In addition to looking for bugs, the review should check: (1) whether the entire project has followed the coding standard in a consistent manner, (2) whether the project has followed the design principles introduced in class, and (3) whether there are code smells that indicate the need for refactoring. The following checklists provide basic guidelines. You may add new items to each of the checklists.

Make sure your answers resulted from the code review exercise. If there is no finding for an entry, you should provide an explanation. For example, if your answer to “Are the naming conventions violated?” is no, you should describe a naming convention and present an example. You will receive no credit for this part if your answers are simply yes or no without additional information.

Classes that have been reviewed:

Date/time duration of the code review exercise:

|  |  |  |  |
| --- | --- | --- | --- |
| **Checklist** | **Checklist Item** | **Findings** | |
| Coding Standards | Are the naming conventions violated? | No violations. Public types (GameController, GridSystem), methods (ReplayGameAsync, MakeMove), and properties (PlayerOneScore) follow PascalCase. Private fields (gameLogic, overlayCanvas) and parameters/local variables use camelCase. Which is consistent and follows the reommended Microsoft’s C# conventions. | |
| Is the ordering convention of method arguments violated? | Minor recommended adjustments my game logic. The seven-parameter GameController constructor interleaves two flags (redIsCpu, blueIsCpu) with two data values (redLetter, blueLetter). While functional, grouping all boolean flags first and then data parameters, or encapsulating them in a small settings object. This could reduce the chance of misordering at the call site. | |
| Any comments meaningless or inconsistent with the code? | I believe I could only find one minor gap. In GridSystem.ResetBoard(), the comment describes clearing children and row/column definitions but omits mention of clearing overlayCanvas. Children. The implementation is correct; updating the comment to include “clear the overlay canvas” would ensure full alignment between comment and code. | |
| Any code block has an inconsistent formatting style? | None found during code review. All methods across both classes use K&R brace style, consistent blank lines between members, and uniform indentation. There is no deviation in formatting between different parts of the codebase. | |
| Any indentations inconsistent? | No issues found during code review. The entire solution uses 4 space indentation with no mixed tabs or spaces. Nested blocks and continuation lines align in a uniform manner. | |
| Design Principles | Any class/method not well-modularized? | Minor point. GridSystem currently handles both grid initialization (creating rows/columns/buttons) and overlay rendering (drawing lines). Splitting these concerns, one class for layout/setup and another for overlay drawing. This would enhance modularity and make each part easier to test and maintain, though the combined approach remains clear at this project size. | |
| Any class with poor abstraction? | Small coupling. GameController directly calls Dispatcher.UIThread.Post to perform end of replay UI updates. Routing UI updates solely via the OnMoveMade event (subscribed by MainWindow) would fully detach core game logic from platform specific threading concerns and improve adherence to the Dependency Inversion Principle. NOTE: THIS WAS RUSHED WHEN IMPLEMENTED. | |
| Is the visibility of any variable, method, and class inappropriate? | MoveHistory is publicly writable. Exposing this mutable List<ScoreEvent> allows external code to modify game history. Converting this to a read-only property (IReadOnlyList<ScoreEvent>) backed by a private list preserves encapsulation while still providing necessary information to the UI. | |
| Is design by contract (pre/post-condition) violated? | No violations. Methods like MakeMove return a boolean to indicate invalid moves rather than throwing exceptions. Preconditions (valid row/col ranges) are enforced by the underlying BaseGame implementations, and post-conditions (state update) follow immediately. Contracts are upheld implicitly and documented through return values. | |
| Is the Open-Closed Principle violated? | I believe that is it acceptable at the current scale of the program. The constructor’s switch on gameMode must be extended to add new modes, which is straightforward given only three game types. If additional modes are anticipated, introducing a factory or registration mechanism would allow new game types to be added without modifying existing code. | |
| Is the Single Responsibility Principle violated? | I believe that this is mostly respected. Each class has a primary, cohesive responsibility: GameController manages move flow, GridSystem handles UI grid/overlay, and threading concerns are minimal. The only minor SRP crossover is the single Dispatcher.UIThread call in the controller; that logic could be moved entirely into the UI layer if strict separation is desired. | |
| Code Smells | Are there magic numbers? | Yes, but only in a minor way. Grid size limits (3 and 12) in SetGridSize and the 500 ms delay in ReplayGameAsync are hard coded. Although these values are obvious in context, defining named constants such as MinGridSize, MaxGridSize, and ReplayDelayMilliseconds would improve readability and facilitate future changes. | |
| Are there unnecessary global / class variable? | No true “globals”. All state is confined within class instances. The only potential over exposure is the public MoveHistory, which should be read-only rather than fully mutable. | |
| Is there duplicate code? | Minor duplication. Both SaveGameAsync and LoadGameFromCSV configure FileStream/StreamReader with identical FileShare options. This boilerplate could be extracted into a small helper (e.g. OpenSharedStream(path, mode)) to adhere to DRY, but its current form remains clear and easy to follow. | |
| Are there long methods? |  | |
| Is there any long parameter list? | Unfortunately yes, this does occur. The GameController constructor takes seven parameters, mixing operational flags, data, and callbacks. While workable now, if I were to use a dedicated GameSettings or builder pattern, it would streamline instantiation and reduce cognitive load. | |
| Is there over-complex expression? | I don’t believe so. The code contains no deeply nested or overly complex expressions; logic is straightforward and easy to read. | |
| Is there switch or if-then-else that needs to be replaced with polymorphism | Yes, noted. The switch on gameMode in the controller could be refactored into a factory pattern to avoid modifying the constructor when new game types are added. However, with only three modes, the current approach is concise and maintainable. | |
| Any variable or method name whose intent is unclear? | Minor adjustments could be made in GridSystem.UpdateOverlayCanvas, the parameter name baseGame could be shortened to game without loss of clarity. Overall, naming is descriptive and consistent. | |
| Any similar methods in different classes? | Beside the previously mentioned the minor duplication of SaveGameAsync and LoadGameFromCSV. I wasn’t able to directly identify any near-duplicate logic across the reviewed classes. Each method serves it owns distinct purpose within the overall program. | |
| … |  | |
| **Bugs** using (var stream = new FileStream( | **Buggy code snippet** | **What is the bug?** | **Why is it a bug?** |
| FileMode.OpenOrCreate, | Using FileMode.OpenOrCreate when saving the CSV can leave leftover bytes if the new file content is shorter than the existing file. | Those leftover bytes become “ghost” lines in the CSV that LoadGameFromCSV will read as additional (and invalid) moves. This could potential lead to corrupting the game load process and leading to malformed or unexpected behavior. |
| Public mutable MoveHistory   (While this is a bug for our SOS project. This is intentionally left for further exploring exploitation of the game). | public List<ScoreEvent> MoveHistory { get; } = new List<ScoreEvent>(); // track all the moves | Exposing MoveHistory as a public, mutable List<ScoreEvent> allows external code to arbitrarily clear or modify the move list. | External code (or tests) could call MoveHistory.Clear() or insert bogus events, which would corrupt replay sequences and score calculations, breaking game logic integrity and UI sync. |

1. Summarize the lessons learned from the entire project by answering the following questions from the perspectives of development processes, coding, design, refactoring, and testing **(3 points)**:

* What did you personally gain from the project?
* What does your project do well, and what could your project do better?
* How could you improve your development process if you develop a similar game from scratch?

Minimum requirement for (4): One full page single spaced, font size no bigger than 12 points.

Throughout this project, I feel that I gained a ton of invaluable experience. The first thing I can think of is being able to explore a completely new development environment, C#, (.NET), and Avalonia, as well as create a more practical program in a college course. I don't personally intend to go work within the game development industry. However, I do believe that the foundation of what I learned will allow me to further build on becoming more intentional with the way I develop programs as a whole. Another thing that I really enjoyed about this project is to understand the reason to adopt TDD. Through developing my program, I began to realize that I was wasting a lot of time on creating the program and making multiple updates before testing. This led me to situations, not the fact Sprint 2 was submitted at 11:56 PM. I eventually forced myself to quickly move away from the way I was developing and start creating the tests for the logic prior to implementing everything else, including the UI. While I have had experience writing code for the last 4-5 years, it has never been within a professional environment outside of creating simulated HTML phishing webpages and scripting for operations (pushing admin to endpoints or autolocking devices after 5 minutes). Additionally, I am incredibly happy for what my extra credit feature is. I won't go too far into detail as that will be submitted in the following Sprint. This vulnerable game allowed me to explore exploiting a LOLBIN for the first time in a new programming language, which will absolutely be crucial as I continue to explore the realm of cybersecurity research and exploitation development.

Overall, I am very proud of the final project I have created! This is the first desktop application that I have built that has had an GUI, which felt incredible when it finally loaded on my screen. I believe that the program does following most of the principles we were taught. I assumed that application development would be similar to the architecture of web development, which is somewhat true. However, I did realize that the reason we create such open architecture is to allow us to continue to build and improve our work as developers or the clients requirements. I will admit that I did get a bit burnt out due to as you put in class today, hell week. I do feel a bit "lazy" for not ensuring that the newly implemented replay feature was fully separated from the game logic. I feel that I could have applied the principles more aggressively and create an interface for my replay logic from GameController. A few of the things that I feel like could have went better with the program would be to follow through with refactoring more. I feel like some of my logic may be "overly" complex and to spread out, however, I think taking this approach allows me to be more intentional and granular with my approach to development. Another thing I wish that I had the time to explore is the true power of Avalonia. This semester I am also taking the Human Computer Interface course which has taught me so much about the empathy and intention of the what the user interface actual is. From the research that I've done, there is a lot the framework which I didn't even get to explore due to the lack of time. Since I am going to utilize this game to expand the security concepts for a tabletop exercise with our summer intern. I believe that improving on this will definitely something that I am going to continue to improve over the course of the summer. Ensuring to try to following the ELEGANCE of design.

The first thing that I would do differently is start from the perspective of the user and not the developer. Having such a hyper analytical brain and ADHD, I always tend to make the things I do more complex than they need to be. For example, during Sprint 1, I wanted to get as much of the "brain dump" out as possible into code for the vulnerable game state. My hyper focusing on my non-required feature caused me to shift the focus away from the requirements on more than one occasion during the project. Another thing that I would want to remember is recognizing the importance of TDD. If I were to develop another ground up game, I would absolutely focus solely on a test driven development approach. Not only would doing this allow me to reduce the amount of time I spent debugging, but also allow me to ensure that whatever requirements are necessary are fully complete, tested, and verified. It's odd that I understand this from both perspectives though. On one hand, I want to ensure as the developer that I am creating the "coolest" version of the program. But the UIUX class has really engrained the importance of ensuring that your user is comfortable within the interface. I would take more time to wireframe the design before even writing a line of code. While I did attempt to do this, I didn't follow through with the original design and ended with what I feel, is a not great interface. Providing myself with more time to think from more than one perspective would allow me to ensure that the game I am developing will not only meet requirements, but also visually appeal to the customer(s).

On a side note, I do feel that I was able to abstract the information we learned to be able to practically apply it within my current career. This specifically has allowed me to dive further into the thought of potentially exploring very low level vulnerability development, exploitation, and detection techniques. This experience is completely invaluable, and I want to thank you for allowing me to explore this more and utilize this space to gain more knowledge.