**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/gwb9h3/SOS-Sprint-5.git**

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: 10**

1. **Demonstration (4 points)**

Submit a link to a video of no more than 8 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=17ba56a8-2292-4877-84e6-b234005e412a**

|  |  |
| --- | --- |
|  | **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 |

If you have implemented the “replay” feature for extra credit, you should include its demonstration in the video.

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 | Record Game | As a user, I want to be able to record a game so that I can read through the record of moves or load game. | 1 | 1.5 |
| 21 | Load game | As a user, I want to be able to load a game so that I can see a visualization of past games that have been played. | 2 | 1.5 |

|  |  |  |  |
| --- | --- | --- | --- |
| **User Story ID and Name** | **AC**  **ID** | **Description of Acceptance Criterion** | **Status (completed, toDo, inPprogress)** |
| #20 Record Game | 20.1 | AC 20.1 User selects record game  Given: The user selects the Record Game checkbox  When: The game detects that there are no issues with the game starting.  Then: The game will write each move to a file that can be used later on to load a game. | Completed |
| #21 Load game | 21.1 | AC 21.1 The user does not have a game to load  Given: The user does not have a proper game file to load stored in files.  When: The user selects the Load button  Then: The game will output an error and tell the user they must have a game file to load in. | toDo |
| 21.2 | AC 21.1 The user has a game file  Given: The user has a proper game file to load stored in files.  When: The user selects the Load button  Then: The game will populate the given board size with each of the moves from the file in storage. | toDo |

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: GameBoard.java & BoardData.java

Date/time duration of the code review exercise: 11/25/24 7:00 PM

|  |  |  |  |
| --- | --- | --- | --- |
| **Checklist** | **Checklist Item** | **Findings** | |
| Coding Standards | Are the naming conventions violated? | There are no naming conventions violated in the two previous files but the GameView file does have oversimplified variables that have lead to confusion when using getters in functions. | |
| Is the ordering convention of method arguments violated? | Within these two files there are no functions that share all of the same arguments to be passed in so there is no violations of the method arguments. | |
| Any comments meaningless or inconsistent with the code? | The comments that are within the code for these two files are not meaningless but there could be more comments explaining the code throughout more consistently. | |
| Any code block has an inconsistent formatting style? | Blocks of code have a consistent formatting style where there is a newline between different sections of code but code within sections do not have excess spaces between lines. | |
| Any indentations inconsistent? | Indentations follow a consistent pattern throughout the code where brackets close off newlines following the last line of code in a given section. See small attached image: | |
| Design Principles | Any class/method not well-modularized? | GameBoard class has had the functions restructured to be more modular creating two children classes named SimpleGameBoard & GeneralGameBoard that have different implementation of a few functions.  BoardData is a class that does not have any children and contains more raw code as opposed to function calls. One of the ways that the class could be better modularized would be adjusting the way that scores are updated for the makeSMove and MakeOMove class to reduce the lines of code involved. | |
| Any class with poor abstraction? | At this moment I cannot find a case of poor abstraction in the two functions. In the GameBoard class most of the code has been hidden behind function calls to simplify the code making it much more readable and the classes called have been simplified and are now easier to work on compared to before.  In BoardData the process of making a move was split into two functions that can be called and while the code is complicated to look at, it works well. | |
| Is the visibility of any variable, method, and class inappropriate? | Most of the variables throughout the code are either private or protected so they cannot be fully accessed globally unless called through a getter or a setter, most of the functions that are used in all of the classes are public so they can be called in each other. | |
| Is design by contract (pre/post-condition) violated? | No pre/post condition violations have been found at this moment in the code. A majority of the pre conditions that are within the code are called in a class named GameController so once those preconditions pass then it moves onto the classes that are listed above. There are a lot of preconditions inside the handleButtonClick or MakeMove functions but they all work as intended. | |
| Is the Open-Closed Principle violated? | It is not violated, The principle is followed in the GameBoard class where SimpleGameBoard & GeneralGameBoard implement the code from GameBoard but do directly edit the code from the parent class. | |
| Is the Single Responsibility Principle violated? | While the program did not initially follow the single responsibility principle the code has been split up into a number of classes that all focus on one task and call other classes to implement their features making the code more readable and easier to change down the line. | |
| Code Smells | Are there magic numbers? | The functions inside of the Move classes could be viewed as containing magic numbers because there are numerous statements checking each of the tiles around the move to be made to see if there are points getting earned while there is more than likely a way to simplify this code down much more. | |
| Are there unnecessary global / class variable? | There are no globle variables inside any of the code as they tend to cause problems trying to track down where they are edited. | |
| Is there duplicate code? | There is some duplicate code inside of GameBoard where a move is getting done and the turn is switching but it is duplicated for the red and blue player. This could be simplified down to only being put down once. | |
| Are there long methods? | There is a list of buttons that gets passed in functions that is used as the table visible to the player. | |
| Is there any long parameter list? | No, the highest number of parameters that a function will take is 3 parameters. | |
| Is there over-complex expression? | There are a few complicated expressions inside the of the createGame function but they are not overly complicated as each part of the expression is necessary. | |
| Is there switch or if-then-else that needs to be replaced with polymorphism | The if else statements in the makeSMove and makeOMove functions would be much better if a way to simplify the functions was used. | |
| Any variable or method name whose intent is unclear? | In the two classes listed above all of the method or variable names are straightforward to try and reduce confusion when reading code. | |
| Any similar methods in different classes? | There are similar methods in classes but they are usually sibling classes that have slightly different code inside their implementation of methods inherited. | |
| **Bugs** | **Buggy code snippet** | **What is the bug?** | **Why is it a bug?** |
|  | This is not the exact spot of the bug but there is no implementation for a reset button to clear the board. | If another game is started after a previous game without closing the program and restarting it the game will not function as intended due to stored values not being reset. |

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 (5): One full page single spaced, font size no bigger than 12 points.

This is the first class that I have taken at the University of Missouri Kansas City where the main homework portion of the course was focused on one incremental project that students work on over the course of the semester. Most of the other courses offered in the Computer Science degree path do small projects that focus on a topic and are moved past within a week or two of the start date. This course on the other hand had us students creating a larger-scale project that required much more attention to detail compared to week-long projects. Getting more practice working on larger projects with ample documentation at each step of the way helped me grow as a programmer as it made me track my progress in my projects instead of coding non-stop without much testing or planning. This has been the second main growth area that this course has helped me develop this semester. While this class put a large emphasis on the documentation steps that are often required in the field to keep code orderly and on the right path, other courses tended to treat the documentation phase as an afterthought that students would rush to finish after the project has already been completed instead of using it as a guide along the way. Having the user stories that we crafted for sprint 0 was instrumental in the guidance we needed to be able to complete this project while making sure to take each step to create quality code. Along with the user stories I learned how to create detailed acceptance criteria, useful for making sure code works as intended, and mitigating bugs slipping through the cracks of the code. I personally believe that these documentation steps have been the best area of growth that the course has given me.

I am quite proud of the project that I have spent the majority of the semester working on. I believe that I was able to come quite close to the examples in terms of user interface which I am proud of. The interface that I worked on is quite simple and easy for people to pick up on how to use. My favorite part of the user interface is the way that I represent connected SOS’s on the board. Instead of spending lots of time attempting to draw connecting lines on top of the buttons for my board, I decided it would be much more simple to just change the background color of the buttons that have been pushed in by users. This makes it easy for the users of my program to see when an SOS has been completed on the board and even updates if a tile is a part of both a blue and a red point. These details that I worked on gave me a great sense of accomplishment that I was proud to show off to fellow students in the class. There is one glaring issue with my program though and that is the lack of support to play multiple games in a row using the same instance of the program. I have not implemented a reset button which in theory wouldn’t be too difficult because I was focused on the required portions of the program. This is one feature that I think would greatly increase the quality of my program compared to where it is at the moment.

There are a few things that I would change if I were to completely start this project over from scratch but my main focus would have to be creating a detailed diagram of the different classes that I believe would be required for the program. Having all of these classes listed out on paper and drawing in the connections that I think would be necessary would help me with creating an organized class hierarchy which was missing in my earlier sprints and cost me lots of points on my submission. It would also make working on my project easier in the long run and more simple to change down the road if necessary. The second major change that I would implement would be to change just about every variable name inside of my GUI file. All of the buttons, switches, text fields, and lists are labeled very generic without an identifying feature which made connecting these buttons to my controller extremely difficult as I was forced to check my GUI file to see which button I had to reference each time as to not mix them up. The last change that I would like to implement in my code if I were to start over is the way that I checked for points when making a move. As is at the moment I have hard coded in each different possible test the program has to check when a move is placed which could be done in much fewer lines of code and would be more simple to read. In the moment it was the easiest way to deal with trying to tally points for a move but looking back on it now I know there are better ways that I could have handled accomplishing this task.