**Project Sprint #5**

The main tasks of this assignment are:

1. Adding the feature of recording a game into a text file. The user story and acceptance criteria of both record and replay are required, but the implementation of replay is for extra credit (up to 2 points in the weighted total).
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 (6 points)**

Submit 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.

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

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 a Game | As a player, I want to be able to click a “Record Game” button so that I will get a text file detailing what each move a player made. | 1 | 5 |
| .. |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **User Story ID and Name** | **AC**  **ID** | **Description of Acceptance Criterion** | **Status (completed, toDo, inPprogress)** |
| 20 story | 20.1 | AC 20.1 <Player has selected to record the current game>  Given a player has clicked the “Record Game” button.  When the game is over.  Then the text file should contain all the moves made in the game. | Completed |
| 20.2 | AC 20.2 <scenario description>  Given a player has not clicked the “Record Game” button.  When the game is over.  Then the text file should not contain all the moves made in the game. | 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: Board.java and GUI.java

Date/time duration of the code review exercise: 5/4/2022 – 5/6/2022

|  |  |  |  |
| --- | --- | --- | --- |
| **Checklist** | **Checklist Item** | **Findings** | |
| Coding Standards | Are the naming conventions violated? | No, both classes follow the naming conventions. All the variables start with a lowercase, have no spaces and start the next word with an uppercase, and describe what they are holding. | |
| Is the ordering convention of method arguments violated? | No, both classes follow the ordering convention of method arguments. The names of parameters in the method call match the names of the method arguments. | |
| Any comments meaningless or inconsistent with the code? | Yes, there are meaningless comments in the code. In Board.java class on line 604, it is redundant to say the function checks to see if the board is full when the function name is “isBoardFull”. In GUI.java on line 73, it says that it is now creating simple and general game variables. This is not needed and is meaningless. | |
| Any code block has an inconsistent formatting style? | Yes, for both classes. For both of them, some functions start their functions right after the function definition and parameter line while other functions skip a line to make it more clear. | |
| Any indentations inconsistent? | No, both functions seem to have a consist indentations. If there is a loop, if/else statement, or function declaration, then the next and following lines are indented. | |
| … |  | |
| Design Principles | Any class/method not well-modularized? | Yes for Board.java. Board.java could have better modularized the functions that check for SOS’s (checkHorizontal, checkVertical, checkLeftToRightDiagonal, and checkRightToLeftDiagonal). GUI.java does a good job of modularizing it’s function. Like it has a reset function instead of just combining the reset function’s contents at the beginning of the makeBoard function. | |
| Any class with poor abstraction? | No for both. Both classes have no need for abstraction (GUI doesn’t need one because there is only one GUI and Board doesn’t need one it takes care of all the needed functions for our board). | |
| Is the visibility of any variable, method, and class inappropriate? | No for both. Both classes keep the variables private and the functions public (except for constructors). | |
| Is design by contract (pre/post-condition) violated? | No for both. Both classes have adapted and completed their pre/post conditions. Board has added new methods to keep up with changing conditions like computer players and recording games. GUI has adapted it’s functions to include new options like if a player wants to be a computer. | |
| Is the Open-Closed Principle violated? | No for both. I have been able to add new ideas and functionality to my code without changing pre-existings one (like computer players, recording the game, forming and keeping track of SOS’s in the game. | |
| Is the Single Responsibility Principle violated? | No for both. Both classes’ functions only have one intended goal or responsibility to complete (like GUI’s resetBoard is only incharge of reseting the board for a new game and Board’s getBoxOfBoard is in charge of getting a certain box from the board using a row and column. | |
| Code Smells | Are there magic numbers? | No for Board. This classes use a variable instead of a number and the variables name gives the user a good idea (like boardSize for board). The GUI class does have magic numbers (like line 101 and 102 when the GUI sets up the actual board). | |
| Are there unnecessary global / class variable? | No for both. Neither class’s contain a global variable. | |
| Is there duplicate code? | No for GUI. GUI utilizes it’s functions and makes sure each one is optimized (like makeBoard function doesn’t duplicate code from the resetGUI function). Board class does reuse code, specifically in the checkHorizontal, checkVertical, checkLeftToRightDiagonal, and checkRightToLeftDiagonal functions. | |
| Are there long methods? | Yes for both. GUI’s makeBoard function is rather long and Board’s checkHorizontal, checkVertical, checkLeftToRightDiagonal, and checkRightToLeftDiagonal are all extremely long. | |
| Is there any long parameter list? | Yes for GUI. GUI’s drawLine function has six variables in the parameters. Board’s longest parameter is 3 variables in checkHorizontal, checkVertical, checkLeftToRightDiagonal, and checkRightToLeftDiagonal functions. | |
| Is there over-complex expression? | No for GUI. GUI keeps everything relatively simple like the if statements on lines 161 and 167. Board does have over-complex expressions, like line 167-170 for checkHorizontal function. | |
| Is there switch or if-then-else that needs to be replaced with polymorphism | No for GUI. There is only 4 cases of an if statement in the whole GUI class on line 142, 161, 167 and 235. Board class does have some overly complicated if/else statements in checkHorizontal, checkVertical, checkLeftToRightDiagonal, and checkRightToLeftDiagonal functions that could have been simplified using polymorphism | |
| Any variable or method name whose intent is unclear? | No for both. GUI class has SOSBoard that keeps track of only the SOS’s on the board and resetBoard function resets the board. The board function has gameOver which keeps track of when the game is over and addBox adds a box to the board. | |
| Any similar methods in different classes? | No for both. Each method has it’s own unique purpose for the class. | |
| … |  | |
| **Bugs** | **Buggy code snippet** | **What is the bug?** | **Why is it a bug?** |
| No bugs in the code. |  |  |
|  |  |  |
|  |  |  |

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

|  |  |  |
| --- | --- | --- |
| Source code file name | Production code or test code? | # lines of code |
| Board.java - 640 lines | AC1TestCases.java – 40 lines | Source Code – 1,737 lines |
| Box.java – 102 lines | AC2TestCases.java – 34 lines | Test Code – 260 lines |
| ComputerPlayer.java – 22 lines | AC3TestCases.java – 58 lines |  |
| GameMode.java – 53 lines | AC4TestCases.java – 47 lines |  |
| GeneralSOSGame.java – 174 lines | AC6TestCases.java – 47 lines |  |
| GUI.java – 247 lines | AC8TestCases.java - 34 lines |  |
| HelloApplication.java – 32 lines |  |  |
| HelloController.java – 8 lines |  |  |
| HumanPlayer.java – 18 lines |  |  |
| PlayerBox.java – 87 lines |  |  |
| PlayerParent.java – 18 lines |  |  |
| SimpleSOSGame.java - 159 lines |  |  |
| SOSGame.java – 177 lines |  |  |
| Total lines of code | | 1,997 lines of code |

**You will receive no credit for this assignment unless your complete source code is submitted.**

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 **(6 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.

Throughout this project, there were three things that I have learned, experienced, and realized and they can be summarized using these three questions: what did I personally gain from the project, what does my project do well and what could it do better, and how could I improve my development process if I had to develop a similar game from scratch. First, I personally gained an inside look into software engineering. Before this class, I had some ideas of what software engineers did, but it was based off of blog posts and YouTube videos, not first hand experiences. But after taking this class, I finally got a taste of what being a Software Engineer is like. You have to be able to schedule your time correctly for each Sprint, you need to have your code not only work for the present code but also be adaptable for inevitable changes, and be able to write automatic tests for your code. You also need to have character qualities like dependable, intelligent, resourceful, adaptable, relatable, and organizational skills. Without these necessary qualities it would be extremely difficult (if not already impossible) to be able to achieve the title of software engineer. Finally, you need to ready for failure. Failure is a necessary part of the development process, not a mark of an awful software engineer. It is only because we have stumbled that we truly know the wonder of running. Our shortcomings show us where the problem in our code is. That’s why we have automated tests, acceptance criteria, and user stories: to make sure we are on the right track and to show us when we have wandered off.

Second, what does my project do well and what could it do better? There are three things that my program does well and three ways it could improve. First, my project does a good job of having an interactive and intuitive GUI. Every button has a label, the turns label displayed whether it is blue or red player’s turn, and the boxes are set so that it has an action when the user touches them. This makes the user experience much better. Also, it keeps the GUI more intuitive rather than having the user guess which button does what and trying to figure out how the game works without much visual cues. Second, it does a good job of reusing code. The best example of this is the hierarchy for the players and the game mode. Keeping it in a hierarchy not only keeps the code cleaner, but also removes the need to put redundant code into multiple different classes. Third, it does a good job of separating the GUI logic and Board logic. Throughout this project, it can be easy to intertwine the how the board works with the GUI itself. However, this would not have been a desired outcome as it would lead to messy and tangled code; much like Christmas lights when you first bring them out. Instead, we want to keep these parts of the code separate so that a change in one class won’t impact the other class and vice versa. While these were three ways that my code succeed, there were three ways in which I would wish to improve my code. First, my program has some cases of having complex and unintuitive if/else statements. These situations could have very easily been transformed into a separate function to not only give the code some much needed clarity, but also increase the modularity of the code. Second, the GUI is rather basic. When it comes to the Graphical User Interface, it gets the job done and is clear to the user. But it is very much plain and some added color and details would go a long way in impressing the user. My current GUI is very much like a Ford Focus; it gets the job done efficiently, effectively, and it doesn’t break down. But I would rather my GUI be more of a Ford Mustang: fast, flashy, and impressive to everyone who views it. Finally, my code has some unnecessary comments. Many of the comments I added in to help those viewing my code actually are redundant and not needed. Instead of adding more value to the code, it’s just wasted space and extra lines for the program and I would remove them.

Finally, how could I improve my development process if I had to develop a similar game from scratch. I would do this in three ways. First, I would worry less about the deadline and more about the code. As the day that the code is due gets closer and closer, it can be easy to lose focus on the task at hand and try to hurry up writing code. But that is actually the exact opposite of what we would hope to accomplish. We shouldn’t worry about the due date, but worry instead about the clarity of the code. Yes, we want to submit the code on time; but if the code doesn’t work, then there is really no point in delivering it on time. Second, I would ask for advice from others. There is always wisdom in seeking out advice and tips from others who have done what you are doing. And if I had another similar project to develop, then I would go to other students and ask their advice on what to do. Obviously, I wouldn’t steal their ideas, but I would take their tips and gain inspiration for my own ideas. Finally, I would look at the project from a bigger picture view. It’s easy to focus on one class or one function instead of seeing the whole plan. But if we only look at the details and not the grand scheme or idea for the program, then we will lose sight of what each variable, function and class needs to do.