HW5: Portfolio (167.5pts)

Replace all the highlighted placeholder text, below. You may change the style of this template, but please provide all the responses requested and keep the same sections / order.

**Usability note**: If you double-tap/click the placeholder text, that text should become selected so that you can easily type over it.

# Your Team Name (0.5pt)

**Which team** are you on?

|  |
| --- |
| ASDFJkl; |

# Project Description (5pts)

Who is the **target audience** of your software?

|  |
| --- |
| My target audience is people who want to play or learn to play the board game shogi. |

What is the **purpose** of your software? (e.g., what problem does it solve?)

|  |
| --- |
| My software provides a way to play shogi and displays possible moves for each piece in case the players do not know which pieces do what. |

What **service** does your software **provide**?

|  |
| --- |
| My software provides an image manipulation service that can rotate images. |

What **service** does your software **use**?

|  |
| --- |
| My software will use my own service as well as a Wikipedia scraper service to get information about each piece and the rules of the game. |

# Status Updates (30pts)

Do at least four status updates on four different days.

Screenshots of status updates:

|  |
| --- |
|  |
|  |
|  |
|  |

# Code Smell Refactoring (40pts)

**Instructions:**

* **Evaluate** your code against each of the 11 code smells mentioned in the reading.
* **State** whether your code does or does not have the smell.
* **Refactor** all smells you find.

|  |  |
| --- | --- |
| **Code smell** | **Has smell or not? (**e.g., Yes) |
| Obsolete comment | Yes, one of the functions that I use to create pop ups for my GUI started out as an error alert pop up but has since been converted into a general use alert pop up functionality – the function comment still explicitly says that it is used for errors. |
| Commented-out code | Yes, the init method for my GUI initially had many definitions and objects created within it, but I eventually made a function to perform the tasks outside of the init method. The code I commented out as backup is still there. |
| Redundant comment | No, I tend to not enjoy writing comments – if anything, my problem is that there are areas and functions that do not have comments but probably should. |
| Long comment | Yes, the function I used to visualize the program state before I created the GUI has a very long comment describing what everything means, even though it is fairly intuitive. |
| Long function | Yes, the code that runs when a place on the board is clicked is over 150 lines long due to several long if-else statements checking to see what needs to be done, even though much of it is the same, just in different branches. |
| Function with many jobs | No, I try to not force functions to do things they were not intended to do since it can affect the primary functionality – instead, I will copy the code and modify the copy if I need a function with similar functionality to one that I already have. |
| Function with many parameters | No, I have learned the hard way how tedious it can be to create functions with too many parameters – I try to keep parameters at two or fewer whenever possible and often get necessary information within the functions themselves. |
| Duplicate code | Yes, I have a line of code that handles conversion from human-readable coordinates to its location in an array – if I change how the array or coordinates are handled, I will have to change every instance of this line. |
| Long lines | Yes, creating objects for the GUI can take many parameters and thus be very long. |
| Inconsistent conventions | No, I try to be as consistent as possible with conventions since inconsistencies tend to bother me. |
| Vague naming | No, I try to be specific about a variable’s purpose when possible to avoid problems when trying to debug or read my code in the future. |

**Before-and-after** versions of refactored code (copy-paste table for each refactoring):

|  |  |
| --- | --- |
| **Name of code smell:** | Obsolete comment |
| **Before**: |  |
| **After**: |  |

|  |  |
| --- | --- |
| **Name of code smell:** | Commented-out code |
| **Before**: |  |
| **After**: |  |

|  |  |
| --- | --- |
| **Name of code smell:** | Long comment |
| **Before**: |  |
| **After**: |  |

|  |  |
| --- | --- |
| **Name of code smell:** | Long function |
| **Before**: |  |
| **After**: |  |

|  |  |
| --- | --- |
| **Name of code smell:** | Duplicate code |
| **Before**: |  |
| **After**: |  |

|  |  |
| --- | --- |
| **Name of code smell:** | Long lines |
| **Before**: |  |
| **After**: |  |

# Implementation (52pts)

**Instructions**:

* **Complete at least one user story of your choice.** 
  + Must add *new* functionality to your project.
* **Create a video**:
  + Show your entire project (think about what you’d like to show a prospective employer---or a classmate who hasn’t seen your project)
  + Mention which functionality you implemented for Sprint 5

**Video location:**

|  |
| --- |
| https://www.youtube.com/watch?v=ewIq5fRQHxQ |

# Rubber Duck Debugging (10pts)

Try RDD with a teammate (take turns being the duck), or an inanimate object like a rubber duck. Write a short reflection on the RDD experience. (At least one paragraph)

|  |
| --- |
| Since I was unable to get together with a teammate, I tried out rubber duck debugging with a small bowl filled with orange peels, and it was surprisingly more helpful than I thought it would be. One of the big issues I tried to solve during this sprint was how to get a popup to return information to the main window and destroy itself at the same time. Putting the return statement in front of the destroy function means that the code would never reach the destroy function, while doing the opposite means that whatever needed to be returned was destroyed. As I walked my bowl of orange peels through a completely different section of the code, I explained the purpose of one of the classes, its use of get and set functions, and local variables. It then occurred to me that I could do the same thing with the popup problem I had; I needed to create a class with methods that create the popups and save the information I wanted to return to a local variable. Once the popup destroyed itself, the information was still stored in the local variable, which I could then access with a get method. This solution is much better practice than what I was planning as a last-ditch effort, which was liberal use of global variables and imports. Other than that, rubber duck debugging also helped me remove some unnecessary checks, wherein one function performed checks to validate data before passing it to another function which did those same checks. The two functions were written in different sprints, hence I had forgotten that I had already implemented the sprints. |

# GitHub (30pts)

**Commit your code** to your repo.

Screenshot evidence of one or more commit this Sprint:

|  |
| --- |
|  |

Create a **release**.

* Instructions: <https://docs.github.com/en/github/administering-a-repository/releasing-projects-on-github>
* Contact me if you want to do releases a different way

Screenshot of your release in GitHub (or other approved location):

|  |
| --- |
|  |

# 