**Design Review**

**1) Time Review**

I spent approximately 25 hours in total working on this project. On average, I would spend three to four hours a day, spread across the period from Friday, the 10th of January, to Friday, the 17th, of January. In general, I would probably spend one and a half hours each day working on particular features that I wanted to add to my game, one hour on debugging and testing the code, thirty minutes on reading the documentation of the JGame implementation, and very little time actually refactoring the code or documenting the code.

For the first part of the project, I was mainly working on basic features. I knew that I wanted to do several things: 1) Make one level filled with small enemies that would need to be shot down or avoided and 2) Make another level that had one boss monster which had the ability to teleport and travel down the screen. A majority of the time was spent trying to perfect the movement and shooting mechanisms of these two features. Along the way, there were small problems with collisions that needed to be fixed, and so those were debugged as they occurred. Towards the end of the project, more time was spent developing graphics and learning how to use GIMP in order to create a more beautiful (although not necessarily better designed game).

I think the time would have been much better spent if I had first spent a few hours planning out all of the basic features of the program and then placed daily goals on what I would like to do for that day and how I would go about implementing it into my program. I believe that this would have made my code 1) more understandable 2) more concise and 3) more flexible.

**2) Status - A. Readability**

In general, the code seemed to fulfill its intended purpose. The game was created with the intention of having the user's plane be fragile - killable in one collision with either an enemy or a bomb. The enemies were also fulfilled their intended purpose through the code as the boss monster ended up having 100 hit points that would go down by 5 each time it was shot, and the first level's monsters would end up being destroyed upon collision with any object.

Because of the simplicity of the game, there does not seem to any back channel dependencies aside from a possible dependency of the level of the game on whether the object exists or not (it is located in the Boss class and the level is incremented when the boss is killed). In our case, it doesn't matter because we only have two levels, but this will surely limit the flexibility of the project if more levels were to be added. Each of the objects are independent, and their actions are either independent of key settings, or dependent on only one key setting. Additionally, the cheat codes are only dependent on one key setting.

There are several things that could have been done to make the code more readable. The first is limiting each method to have one important function, and then assigning new methods for new functions. Additionally, there were many extraneous "if statements" that should have been grouped together. The code had to be "hardcoded" and check for properties of each level, rather than having some gameState that should have been called that would change the properties of each function. Even though there were game states that were available in the standard game that accounted for whether the user was playing a level, lost a level, or beat the game, there was not enough time to refactor the code so that it would enable future implementation of new levels. Lastly, it probably would be better to just move each class (Boss, Player, Enemy) to a new file class in the package, rather than sticking them all inside of the Invasion file so that it would be

easier to see what each class actually does.

**2) Status - B. Extensibility**

An abstract level class or interface ought to be created with each level extending and modifying methods and properties of the abstract class that are specific to each level. There are many aspects of the game that are affected by the specificity of the code, but the main areas dealt with movement and shooting mechanisms of the objects in each level. Additionally, there are problems that will arise should a new level with a different set of objects arise. The heaviest modifications resulting from implementation of a new level would be located within the Player and Boss classes, as those are the most level-dependent classes.

**3) Conclusions**

The most difficult things to figure out were why certain objects were disappearing when only a small portion of the code that seemingly had nothing to do with the resulting disappearance of objects was changed. In general, it was not because the small portion of code was changing, but because there was a pre-existing problem that caused the small change to subsequently result in the disappearance of certain objects. Additionally, it was hard to figure out that the disappearances were generally a result of collisions between the object and the projectiles that it was emitting. There were several ways to fix that: make it so that the projectiles would never be close enough to collide or to disable the collisions between the objects, which is perhaps a safer, more encompassing method of debugging.

It seemed as if the player and the boss classes required the most editing because every time a new feature was added, both of these classes would have to be modified, likely because they

had a litany of “if” statements and weren’t designed to be flexible and survivable.

A good way to prevent having to spend hours just modifying small details of code would be to limit each function to do one thing and then refactor classes that do the largely the same thing and extend the superclass by slightly modifying the functions for each specific instance (in reference to levels, mostly).

The main thing I ought to start doing is simplifying functions and preventing repeats in the code. Something I ought to keep doing is being specific and making sure that I test for every possible case within my code. One thing I ought to stop doing is writing so many if statements - they aren’t fun to read and are even less fun to modify. Additionally, it’s easy to forget which if statements do what and to have to change a long list of things that aren’t grouped together each time a new feature is added.

If I could work on one part right now to improve my grade, it would not be to add more features or even change the graphics, but it would simply be to make my code more concise and easier to read. I would add a level superclass or abstract class and then create level classes for each of my stages in the game.

**Test Plan**

One way to test the game is to write JUnit Tests that will test for whether projectiles are firing, whether collisions are occurring, whether objects are being removed when they collide, and whether objects are appearing correctly when new lives are given or when the game is reset.

Things that are easy to visualize in the game - such as collisions, appearance of objects - would probably be easiest to test out while the finer details - such as whether things are appearing at the exact timing that they should be - would probably be much more difficult to work out.

Some example tests that could be made for my own code would be:

1. To check for collisions between objects
2. To make sure that firing doesn't kill objects
3. To check timing for the teleportation of the Boss object
4. To see if all objects are removed after each LifeLost and GameOver

Some things that could be done to make automated testing easier are:

1. Breaking code down into more methods and classes so that individual components could be tested.

*Note: I tried to implement interfaces so that I would not have as many "if statements" in my Player object, but I was unable to quite figure out how to do so without having to restate all of the constants and redefining all of the invasion functions. Also, doing so completely broke down the functionality of the game.*

Acknowledgments:

- Bradley Sykes, for helping me understand how to use interfaces a bit better.
- Jeremiah Siochi, for working with me on some game mechanics such as collisions and shooting as well as some graphic design logistics from Standard Game
- Joel Luther, for helping me out with the graphic design and teaching me how to use GIMP

to create better images than the ones available online.

---