Michael Deng

Compsci 308

Game Analysis

**Time spent on project:**

I probably spent a totality of 30 hours on this game. I started learning about stages, scenes, and panes on Wednesday. From Wednesday to Saturday, I spent the majority of my time familiarizing myself with some of javafx’s syntax and trying to clumsily build my game. Then on Sunday, I took a look at how Prof. Duvall structured his game on GitHub and I decided to completely scrap my game and start from scratch. I spent the majority of Sunday afternoon and evening completely rewriting my game. By Monday night, I was completely done with the main functionality of my game and by Tuesday night, I had finished writing in all my comments. Wednesday and Thursday were dedicated to refactoring and improving my code. I did not begin GitHub-ing my work until the majority of the game’s functionality was completed. In the future, I will remember to start at the beginning. In the beginning, before I finished the majority of my game’s functionality, I would work on average about 4 solid hours a day. On Sunday, when I decided to completely rewrite my game, I spent a good 7 to 8 hours on it. This past week, I only had to spend roughly an hour per day to get everything I wanted done.

**Code Design:**

My code is designed with four classes: Main.java, Gameloop.java, Sprite.java, and Cookie.java. Because my game only has two objects (the player’s character and the cookies falling from the sky), I decided that no abstract class or super class was needed because the two objects were so different from each other, it would not have been helpful anyway. The game loop was my main class, where I populated the game’s scene, constantly updated the sprite’s and cookies’ movements, and handled any events that may cause the game to pause or stop. My Main.java was simply used to open the splash page and start the game whenever the player pressed a button. What Main.java would usually be used for is handling different scenes and groups they may occur in the game.

My game design followed the same implementation as the one suggested in class, which was to have a main game loop that updated the different classes of the game by looping through one method called updateSprites() once per frame. What’s nice about this implementation is that whenever a programmer wants to add another object to the application, say a bullet object that allowed the player to shot down cookies, all they would have to do is create a new class for the objects and then in that class add in a method that constantly checks the condition of that object. Then all that has to happen is that method being called in updateSprites() in GameLoop.java. Another way this implementation makes further additions to the code easier is if the programmer wanted to add in new scenes to the game. Right now, my game only has two scenes (the splash scene and the game scene), but if they programmer wanted add in let’s say a pause screen or a congratulations scene at the end of the game, it could be easily done in Main.java by initializing a new scene on the current stage.

What’s also nice about breaking the different objects in the game up into different classes is that fact that they can be individually scrutinized and tested. For example, if one where to want to test something specific to cookies, my code is modular enough that only a change to Cookie.java would be needed, while no other methods would need to be touched. In this way debugging the code also becomes easier because the programmer would be to single out problems in specific classes.

**What I liked about my code:**

What I mainly liked about my code were my two classes that created by game objects: Sprite.java and Cookie.java. Even though I did not like my excessive use of if statements in either of the two classes, I believe both are a pretty decent case of code that is both shy and dry. My main method in Sprite.java is updateSprite, which is responsible for moving the sprite based on what arrow keys are pressed. The two methods called within the method are activateKeyEvents(), which activates a lot of key board events, and moveSprite(), which moves the sprite based on those keyboard events. The method is only 10 lines long and contains two much larger and much more important methods which handle the majority of the work. In this sense, the code is very shy, because it does specific details of how the code works but rather just returns the necessary commands to make the sprite move. In sprite.java, moveSprite() is the method that moves the sprite depending on what key is pressed, instead of writing in every case and writing sprite.setX() a bunch of times, I created 6 methods that define the movements of the sprite and implemented those in moveSprite(). Now not only is that method shorter and much more readable, but code is repeated much less in that method, making it more dry.

Another example of where I tried making my code more dry was in Cookie.java, where I implemented where the cookies would fall in calculateCookieXLocation(). I use several methods to adjust how “intelligently” the cookies would fall based on level. Each level gets harder and harder by implementing new forms of intelligence but also by calling the previous levels’ methods that determine how cookies fall. In this way, no code has to be duplicated and the methods remains dry.

**Problems I had with my code:**

My biggest problem with my code was there were too many continuous if statements. Granted, I did not know of a better way to do it, but I have quite a few methods where I have a bunch of “else if” statements to the point where it was confusing. For example, when I was writing my method that handled all of my KeyEvents, I decided to use a whole list of if statements because I wanted to be able to activate multiple keys at the same time and not have to break out of a “select case” implementation. Another example where I didn’t know a better method than if statements was when I tried to give some intelligence to my falling cookies. Basically, I wanted cookies at higher levels to “predict” where the player would go to try to escape the falling cookies. I populate the cookies in a cookie array and I give them each an ID equal to their index in the array. Then I have a trail of if statements that say “if ID == 1, if ID == 2, if ID == 3”, etc.

Another issue I had with my code was the amount of Booleans I used. Once again, I didn’t have a better way of implementing what I wanted to do, but whenever my application encountered some game event, such as my player getting hit by a falling a cookie or the player pressing the key to skip a level, I would handle that event by flipping a Boolean. My game loop continuously checks for a Boolean to be flipped, and once it is, the game loop fires another method that handles that particular event. If possible, I would have liked to implement a way to handle events without the game loop having to constantly check the conditions of my Booleans.

I do not like the fact that in most of my classes, half of the methods are public and the other half are private. In the future I would like to find a better way to homogenize my method types but still allow other classes to implement the methods of other classes.

One minor thing I did not like about my code was the fact that I had to duplicate a couple functions, such as my createLabel method. I wanted methods in three separate classes to all create labels when needed. The problem is that I would not have been able to do this with one method unless I had create an object of each one of these classes in each of the other classes. So I basically just had three of the same methods in three separate classes. I would like to have had a better way to do that.

**Things I would have changed after thinking about it:**

Instead of having a bunch of if statements that check what level I’m currently on, I believe a better implementation would to almost just have a separate class dedicated to levels and having levels have certain characteristics and elements. Because my project only had 4 levels and only one of the levels was different from the other three, it may not have been too necessary. But if we consider the case where I expand this game and have a bunch of different levels that have different objectives and do different things, the string of if statements this issue would create would blow completely out of proportions and cause poor readability and adaptability. If levels have their own set of attributes and methods, I believe they would have been much easier to implement and much more readable then a chain of if statements.

Another way I could have idealized my game design for this project would have been to implement some abstract class for falling objects and then extend that class with other classes of cookies based off of speed, intelligence, size, falling direction, etc. I feel that my code cluttered a lot and became a lot less readable by implementing a bunch of if statements that checked for what level the player was on, and if there were different classes of cookies that all inherited the same abstract falling object class, then they could just have a level attribute and I would be able to remove all of my if statements.

**Readability:**

All of my code does what I want it to do, the exception being sometimes the bounding box is a little bigger than the image, allowing cookies to hit the bounding box and not the image. My code to fix this issue did not work out the way I wanted it to.

I implemented quite a few getters and setters for both my Cookie.java and Sprite.java classes. I believe that they were needed in order to best share information across classes. Quite a few of my global variables are public as well as opposed to private, which is because I needed to reference them in other classes. In hind sight, it would have been a much better idea to standardize my code to just using getters and setters or global variables, because it would have made my code much more consistent and readable. Just using getters and setters would have also increased the shyness of my code by showing the programmer less in the game loop.

My GameLoop.java, was definitely not the shyest code I could have written, which I guess is evident by the number of comments I included in a couple of the methods. One of my biggest problems was that I did not create separate methods for all of my if statement checks, making a few of my methods blocky and difficult to read. I will elaborate further on this in my code masterpiece.

**Extensibility:**

It is absolutely not clear how to extend the code to add a new level or a new feature. This is partially because the idea behind the game was so simple to begin with and partially because I did not account for needing to add anything else to the game.

The way I should have designed the code was to have an abstract class for falling objects and have a class used to define constants for the program. I ran into the issue where the placement of the cookies, the speed of the player, and a lot of other game factors were dependent on constant numbers. I have several static constants in each class, but the easier and more readable thing to do would have been to have a class dedicated to constants where I could call the numbers from. Not having a separate abstract falling object class also makes modifying existing code harder because it requires I added more and more complicated methods to the class Cookies.java. If I had implemented the abstract class, all I would have needed to do would be to changes a few parameters based on the level because all of my different cookies would all have the same basic functionality.

My use of if statements also greatly limits my extensibility because it requires I stack layers of if statements on top of each other as the amount of different game events increases. I do not have a good solution to this problem, but an excellent implementation would be one that requires I use less “if statement ladders”, both reducing the amount I need to write and increases my codes readability and shyness.

My class Cookie.java has a couple of places where the over-specificity of the code takes away from its usefulness. When I implement the cookiePlacement() methods, I hard code in specific cookieIDs I want to manipulated based on certain levels. Both the current level and the ID of the cookie are hard coded values in my if statements, making them only applicable to one scenario and making them difficult to change if I wanted to added future levels and game additions.

**Conclusion:**

My code was simple enough and clean enough, regardless of everything I mentioned before, that I did not run into too many errors or problems while writing the code. Whenever I got an error or my code refused to compile, it was usually because I did not fully understand the JavaFx syntax I was writing and I was returning an integer instead of a double or what not.

The code that required the most amount of editing was GameLoop.java, this is because it was the class that was responsible for updating the sprites’ movements and collisions, as well as the class responsible for handling an important game events. Whenever another class was edited, this class was usually edited as well because was not initially designed with shyness in mind.

In order to reduce the occurrence of these defects and code smells in the future, it would be quite wise for me to actually do a little planning before I actually jumped into coding. I was so excited to just learn how to use JavaFX that I forget to stop and actually write out how I wanted my code design to be implemented. Polymorphism and the use of abstract classes and inheritance are all things I will consider in the future, even when the project is small and not a lot of coding needs to be done. This is because regardless how small the program is, implementing changes and adding extensions will become so much easier when I used inheritance.

I need to start using abstract classes and learn how to better use different types of classes and methods. I need to keep writing good, dry code in my classes like what I did with Sprite.java and Cookie.java. I need to stop writing so many if statement ladders and I need to stop neglecting the shyness of my code.

If there is one thing I can do to improve my grade, it is to make a couple of my if statement ladders in Gameloop.java more shy and break it up into a couple of different methods, which will make my code a lot more readable.

**Test Plan:**

In class, Prof. Duvall mentioned that the creation of cheats was originally used in order to test programs. So, one way I would continue to test my program would be to continue adding in more and more cheats in order to try to find out if anything were wrong with my program. I have already have cheats that allow my player to pass through cookies and skip levels. Other cheat I could implement to test code could be a cheat that variably changes the speed or frequency of the falling cookies or a cheat that always me to change the picture of the sprite. In this way, I would be able to test how dynamically the speed at which falling cookies and the amount of falling cookies is implemented and how intelligently I implemented assigning pictures to my sprite.

Another thing I would test in my code is the efficiency of my code. The way I have the cookies fall is I loop through a for loop and I have the cookies fall one by one over a short period of time. In this way, I could avoid having to program in concurrency and the use of multiple threads. Yet I would want to put this design choice under stress and test just how far I could go before my computer begins to slow down and my game begins to glitch. I currently have an arraylist of cookies and I would write a program that continuously increments the number of cookies in that arraylist to observe at what point the cookies begin to slow the game down.

One part that could easily be testing for correctness is the placement of the cookies in regards to the play. I already have a bunch of different methods that dictated where the cookies fall and at what speed they fall. All I would have to do is reduce the number of cookies in the arraylist to one and I could easily observe were the cookie falls and how fast it falls by simply putting different methods in the runFallAction() method belonging to Cookie.java.

A much more difficult thing to test would for my program would be the correctness of the bounding box for the sprite. The size of the bounding box for the sprite is what determines were the sprite gets hit and because the bounding box is a rectangle and not the exact shape of the human image I placed within the bounding box, sometimes the bounding box is hit but not the actual image. This creates a problem because then players won’t understand how they could have been hit when the image of the sprite was clearly not. The hard part about testing this is that the bounding box is only larger than the image in some parts of the image. The only way to see where the bounding box is larger is by running the game and having cookies fall on the player again and again, while at the same time trying to hit the sprite in different places in order to see which parts of the bounding box need to be fixed and which parts do not. It is a tedious process where replicating results is particularly difficult to do.

Some example tests I would make for my code, besides the example I mentioned above, would be to write a program that makes a cookie intersect the bounding box of the sprite many times in random locations. This would be to specifically test the boundaries of the bounding box because, particularly in the case of my game, how big or small the bounding box is compared to the image of the sprite is extremely important. Another way I would test my code would be to create different scenarios where I could test individual cookies and how fast they fall and where they fall relative to the sprite. The new test script would also automate the sprite to make the testing more thorough.

In order to make my code more easily testable, it would help to make my code more modular and actually simply a few of my methods further in order to better test parts of my code more specifically. Also creating an abstract class that defines the environment of my game would also be helpful because it would allow me to create my game be also allow me to create development environments in order to selectively test parts of my program.

**Code Masterpiece:**

In my GameLoop.java, there is a method called checkForEvents(), which contains a pretty decently sized if statement ladder. Not only that but the code as a whole was completely unreadable because the if statements were not shy. So what I did was create a separate method checking for the validity of each if statement with an appropriately labeled name. Even though I did not know how to get rid of the if statement tree, the code is now much more readable because the conditions of each if statements are much more shy.