Kevin Jian

CS 308

9/7/13

Project 1: Game, Analysis

**Project Design**

I approached this project first by reading through the assignment and writing an extensive README that documented all of the features I would be building, as well as the mechanics behind each feature. During this time, I also thought about how I would be coding each part. I found that this made it really easy to transition straight into coding, and because I thought about how each part would be coded from the start, I was able to manage my expectations and didn’t find myself overwhelmed by the work that I had to do. Time wise, I spent around three hours a day working on some aspect of the project, starting on August 28th when it was assigned to September 4th when it was due, making it around 24 hours. I spent around an hour brainstorming my game, figuring out the mechanics, and writing out the README. Another two or three hours was spent going through the JGame engine documentation and tutorial, and examining the sample games provided. During this time I also experimented with the engine by creating test programs. From there, it was about 8 hours of coding and debugging until I finished all the basic parts of the game, including the title screen, first and second modes, and the end state. The next 5 hours was spent adding more features, as well as refining or changing the mechanics that were actually in place. Around 6 hours was spent where I attempted to refactor my code and remove code smells, and finally close to three more hours spent debugging the final project to get rid of any remaining issues.

Since my original plan and README was fairly extensive, my vision for the game was very clear to me. With that said, the end project that I submitted was almost exactly what I had in mind when I planned the game. It also took about as much time as I expected, so either I made a really simple game or I just had simple expectations for what I was creating. The mainframe of the game with the two modes was completed by Saturday, well ahead of the deadline, and the next 4 days was spent refining the code, making it more visually appealing, making the gameplay work better for the player experience, and making the game clearer to understand. Three to four hours a day of focused coding seems manageable to me (so far at least). During the planning process, I also had levels of goals, and beyond the ones required for the game, I included stretch goals if I had time. These included adding graphics and drawing my own sprites for the game. These graphics didn’t affect the game mechanically, but simply made the game more visually appealing and easier to play for the user. I also did adjust the game goals a little bit. The original plan was that the first mode would have the user in control of a hook, with the goal to reach the bottom of the sea where there was treasure, having to dodge fish and other objects along the way. In the current version, I found that dodging the fish was much more enjoyable, so I decided to make the first mode continue infinitely until a fish is hit, rather than have it stop at a set time if the player survived to the bottom. The rest of the game is pretty much the same.

Going through line by line, the code seems fairly readable to me. There are quite a bit of magic numbers that I took out and gave accurate variable names to. One problem I ran into was if I was scaling something by half, I would have halfVariable = originalVariable/two, with two being equal to 2, since I couldn’t really come up with a better name for that. I would go back and possibly rename some of these to better reflect what they represent if I were to fix make the code more readable. There were also dependencies when I had to use class instance variables to store factors such as time and score, which were used continuously in my game, within the first, second, and end game mode. Since the JGame interface doesn’t let us add arguments to methods such as doFrame(), I couldn’t really find a way around these dependencies, but if we were to rewrite this without using the JGame engine, I would’ve made these methods able to take input to pass information between game states.

At the same time JGame also made the game easy to extend, as to add new modes all one has to do is create a new game state and use the setGameState() method to move there.

The nextState method is something I created to eliminate duplicity, since I found that every time I switched game states, I would remove all objects as well, so I consolidated this state transition into one method. However, this is an example of a place where my code specificity makes it less useful because it is less general. The nextState method also clears the enter key input, since one of the cheats I use to skip modes is to press enter. Clearing the key is required because pressing the enter key for too long would accidently skip more modes than intended, but the clear key is also overly specific since I also use the nextState method to return to the title screen, which involves pressing esc. In this case, clearing the enter key is useless.

Another part that’s overly specific is in my first mode, when I’m creating the fish objects. I create two fish every so often with them being identical except one starts off moving left while the other moves right. I should have just had one fish object created but randomized the parameters so that the one fish object would spawn either moving left or right, and increased the fish spawn rate to compensate.

Even though the game already matches my original vision, if I were to add additional features it would be very simple to. In the first mode, one idea would be to create a special fish that doubles your points, and I could do that by just extending the current fish class to create one that moves faster, making it harder to catch, and is worth more points by using a multiplier. I could also add different obstacles in the first mode, such as tires and bags to the current boots. Furthermore, currently the second mode only has you follow the fish going left and right. I could make it more challenging by having the player match the fish moving left, right, up, and down. Since the algorithm for the fish movement has already been created, this would simply require adding to the fish object’s move function to cause it to move randomly in two new directions, and would require me to add two more sprite orientations to the media table for the fish. In the sense of these new features, it would not be too difficult to implement them.

To test the code to look for bugs, I went through an entire process of various situations where I would enter the different game modes, test reaching the winning and losing scenario in each mode, try the cheats specific to each mode, and playing the game repeatedly to see if it broke down anywhere. I tried these in different combinations with each other as well, in case there was a unique interaction between some states. I also texted fringe cases, and let the program run for extended periods of time in different modes to see if these ran into bugs anywhere. An example of the process was as follows:

* I went to the title screen, entered the first mode, won the first mode, entered the second mode, won the second mode, pressed enter to end the game and return to title screen
* Title screen, entered the first mode, won the first mode, entered the second mode, lost the second mode, pressed enter to end the game and return to the title screen
* Title screen, entered the first mode, lost the first mode, pressed enter to end the game and return to the title screen
* Title screen, entered the first mode, cheat to exit the first mode back to title screen
* Title screen, entered the first mode, won the first mode, cheat to exit the second mode back to title screen
* Title screen, entered the first mode, lost the first mode, cheat to exit the end screen back to the title screen
* Title screen, entered the first mode, won the first mode, won the second mode, cheat to exit the end screen back to the title screen
* Title screen, entered the first mode, cheat to make the first mode easier, won the first mode, entered the second mode, cheat to make the second mode easier, enter to end the game and return back to the title screen
* Title screen, entered the first mode, cheat to make the first mode last forever, cheat to end the first mode, cheat to end the second mode, enter to end the game and return back to the title screen.

The list went on, but all of these went in different combinations with each other, and I ran several of the scenarios after I made major changes to my code to ensure that each step along the process was working as intended. These parts were all fairly easy to test since for the most part they just required two objects colliding into each other, or objects to reach a certain point within the playfield and observe if the correct reaction to that interaction occurred. Parts that were hard to test for were specific cases in the game, such as surviving the first game mode for a certain amount of time, and hitting the boot object instead of the fish object, which would cause the game mode to end. Cases such as these were simply hard to test because they required an extreme amount of precision when playing the game, and to eliminate potential factors for bugs, especially when not using cheat codes in order to eliminate potential factors for bugs.

Magic numbers was definitely one of the issues I had the biggest problem with. Even after speaking to a TA, it was hard to distinguish what were magic numbers, and I ended up just declaring all of the numbers that I used that weren’t 0 as variables. The benefit to this however is that whenever I wanted to change a factor of the game, especially when I was tweaking the numbers to make the game just the right difficulty for the player, I was able to adjust them all from one location without scanning through my code for the line that included that information.

Another issue I ran into was duplicity. All of my objects have specific interactions when hitting the border of the playfield, and I wanted to eliminate the individual code I had in each object class for these by consolidating them in a method that would check boundary. I tried fixing this by using inheritance, such as making my fish and hooked fish classes extend an all fish class, but I kept running into limitations with the JGame engine and how they treated objects that I couldn’t get around. I ended up spending a lot of time working on this, but eventually I just went back to my original methods. If I had more time or if I understood JGame more, I probably could have solved this, especially since professor Duvall explained inheritance in yesterday’s class. Some methods such as checkEscapce and nextState are very similar as well, and could be put into one overall method.

I initially had a lot of unnecessary code as well. I had originally made object classes for each of the objects I had in the game, from the hook, reel, to fish and hooked fish. As I got more used to JGame however, I realized I was able to eliminate the code for creating these object classes by simply initializing them in states such as startGame() and doFrame(), and I utilized this for objects such as the boots and the reel.

There was also the initial problem when I had other people test my game that they didn’t react quick enough to the switch from the first game mode to the second mode. To remedy this, I added a pause between the modes to let the user collect himself. If I were to write it now, instead of a pause, I would have the game require a user input after finishing the first mode to enter the second mode.

For the future, I will definitely be watching out for magic numbers, especially since making them variables helped speed up my debugging process greatly. I will also try to implement inheritance as a way to reduce duplicity. I feel like by using what I learned during this project will definitely improve my coding and refactoring process for next time.