Reid Daniels

10/29/2018

Rhetorical Analysis

Starting from the beginning, my game is called Scurvy Wars. It is a game about an orange who has to protect his orange brethren from bloodthirsty pirates that have scurvy. The objective is to survive as long as possible, protecting your brethren and earning points for each “kill” you get. The idea for the game came to me as my high school friends and I were talking about ideas for cool game topics. It was originally going to be based around the charge file, but while in the development I realized that the shoot game would work a lot better for me in the long run, so I went with that being as my stub game instead. Let’s start with the character design.

I got the idea for the orange pretty easily. I was brainstorming ideas for the whole game itself, and I thought a more cartoonish approach would be the best for the sort of game I had planned. The gun my orange uses is an orange carton turned on its side, shooting orange slices at the enemy. In a sense, he isn’t actually killing the enemies just feeding them oranges in order to save the majority of his people. The pirates themselves started out as simple people, but as the design went on I thought a more aquatic approach would be better. So, the enemies turned into octopus pirates, who I guess still look for citrus to get rid of underwater scurvy? The beginning screen and the intro clip both lead the player to a more understanding approach for the game, starting by showing an ocean view on an island, leading them to believe the game will be about something in the ocean. Then the intro clip lays out what is actually happening, showing the orange trying to save his people from the pirate onslaught.

For the code in the game, I used multiple classes and linked them all together to get certain parts to work. For the sake of order, I will start with the file called scurvydoc. This file very closely resembles the original stub game file given to us, providing the main screen for the game. I would say about 80% of the code is the exact same as what was given, with my changes coming in around line 30. At line 30, I don’t automatically link the button press to the game, but instead link it to the intro clip so it will play. From there I call the startGameHandler, which I remade to do what I desired. It removes the start menu and adds the intro clip to that area, providing it with coordinates so it will be placed correctly. This is also the area I included the Greensock animation. During the intro clip, towards the end, the pirates steal his box of oranges. The screen will start to turn a dark red at that point, which is accomplished by the Greensock. I then set the code to start the animation on frame one. In order to stop the clip from being a constant loop, the update event listener looks for what frame the animation is on, and once it hits 300, removes the intro clip and runs the game as regular.

The next class is the PlayerUnit. This is the character itself. It calls upon the linked movieclip for the orange. In the class I use a function called gravity to apply physics to when the player presses the space bar, making it appear like they are jumping. I set the value for the bottom or the ground to the dock, which I give the value that I desire. Next is a function called movement, where I apply the animation changes for the character. When pressing certain buttons, a or d in my game, the character will start to head that direction and play an animation on loop. If pressing space, then the character will lift his legs slightly like pushing off the ground. In order to return the orange to a state of non-animation when no button is pressed, I take it to the stand function. This just applies the original movieclip back to the animation so it does not continually run in a direction it is not even moving towards. Finally, I have a getDirection function that returns a value depending on which button is being pressed. This function is used to apply direction to the bullets I create later on in the file.

Class number three would be Bullet. This class links the bullet movieclip and changes the velocity they are moving at as well as the direction and where they spawn. Beginning with the Bullet function, it sets the location of the bullet spawn as well as giving formulas of trigonometry for later use. The setDirection function pulls from the main file that pulls from the PlayerUnit file in order to get the value associated with which direction key is being pressed. This then gets assigned to direction so it can be used later. The loop function is the most used function in this class, being used by every bullet that is created. It applies velocity to the bullet in order to propel it across the screen. The velocity is then multiplied by the direction, which is the value gotten from the PlayerUnit in order to determine direction. The final thing the loop function does is remove the bullet. It looks for the values of the end of the stage that I set it too.

The next two classes are pulled from the shoot file and are almost identical to the original. Balloon class is the exact file for the enemies in shoot. The only difference is I removed the sinMeter. Because I have no need for the enemies to move up and down, I just made them move in a single line. They are given an animation to the enemy, so it looks like their tentacles are actually moving them across the screen. The same is true to the Particle class. It is exactly the same as it was in the shoot file, except a few changes to values so the enemy will actually disappear when they hit the value I would like. I kept the same sort of falling animation as there was in the shoot file because they are fighting on a dock, then when they get hit, they fall into the water.

Finally we come to the main class. The beast of a file. This class is called scurvygame, and it is where all the hard work comes together. The giant conglomerate of variables each have a purpose, even if they are all bunched together and feel like a jumbled mess. The first function is the main called scurvygame. Here I put the elements onto the screen as well as create arrays for the enemies. Next I add currentStage, or the background of the level onto screen. Next comes the gameScore field, on the ships sail. Followed is the healthScore, added under the game score. This includes two elements, the words to tell you what it is which is the healthScore, then the pictures to represent your health which is represented with just a hb. Each enemy that hits your orange box results in a lost box, after three lost boxes the game ends. Next I add the boxes, or the orange box that is set as your base or the point that you lose health. Finally, I add the char, which is the orange itself, pulling from the PlayerUnit class. The variable charSize is used later on to apply screen borders. The final parts of that function are the balloonsLayer and the EventListeners so the other functions would activate.

The next function is called addStage. I had to include this function because when trying to apply all the eventlisteners to the stage, it would give me an error so after the intro clip the game would crash. It was very annoying, but doing it this way removed that error. Next is the loop function, which creates the bullets with help from the bulletList array. The if statement states that if there are any bullets in the array, then the following activates, meaning it will only start once a bullet is put into the array. This then leads to the loop function in the Bullet class, to apply velocity for movement. The next part is the collision, so if the bullet touches an enemy, it will be removed along with the enemies death.

The next function is called enterFrame, activating once the game is started. It includes the gravity function from the PlayerUnit class, applying gravity if they use the space bar. It also gives the speed of the character to the char.x if the specific key is pressed. Continuing down includes the onKeysDown function, which brings in the movement. If a certain key is pressed, it is passed to a Boolean which is set to true so the direction changes. It also includes the gravity value, so the height and speed that the gravity change is can be modified. This also brings in the animations from the PlayerUnit class, so they will be played when a certain Boolean value is true. OnKeysUp function is very similar, applying a false to the Booleans so the stand animation can appear.

The function fire is used to shoot the bullets. It is triggered by a mouse click, which then launches the bullet in the direction the player is facing. If the player is just standing, the bullet will not launch. It then applies the bullet to the array and pushes the array one more each time it is fired. The bulletRemoved is the function that removes the bullet from the screen, once it either hits the edge or hits an enemy, using the past code already stated.

The function chackStageBoundaries is what makes the character stay on screen. If its x is equal to one of those values, it stops it from passing. The function makeBalloons and many that follow are from the shoot file. makeBalloons, makeBalloon, purgeBalloonHandler, and purgeBalloon functions are almost exactly the same as they were, except for a few changes in makeBalloon in order to have the enemies stay in a straight line and spawn where I desired. The function update is very different, only keeping one or two rows of code while the rest being new. It brings in the endscreen MovieClip so the endscreen will appear when the character dies. This only happens if the enemies have hit the boxes child three times, resulting in the frame for hb changing to 4, one change for each hit. If the hb results in frame 4, then the whole game is removed, as well as a new movieclip called endscreen being added. As well as an endscreen appearing, it brings in a new animation that shows the character getting killed, symbolizing that you have lost. Included in the endscreen is a button that is the reset button, which calls on functions called restart and freshgame in order to replay the game, giving endless entertainment.

Overall, I believe I did a pretty good job on this game. Having a little programming experience, it was easier for me to read the code and find the errors but writing all this new code was an experience in itself. This game is far from perfect. It has many bugs, some including that when shooting and you die, the bullets stay on screen, as well as bullets not following jumping and the screen boundary for the top of the screen not working, as well as many others. I am proud of what I have been able to do though. With the help of some of the peer mentors as well as my study group, I was able to get most of the errors or problems I ran into covered very quickly, so I was never really stuck on a problem for an extended period of time. It also helped show me what my time management is like when I have to set time aside to do the game. I may not be where I want to be yet, but I believe I am on the right track. Finally, this game showed me just how hard art is. That may be one of the most difficult parts of this whole project, the drawing and animation. But, even so, I believe I have a better understanding of coding and animation process thanks to this.