**What tests did you perform?** **What were the results? Were the results as expected? How did we remedy the issue?**

**Player and Floor Collision:** We wanted the player to be able to glide on the floor, the collision worked as expected on first try.

**Pipe Movement:** We wanted the pipes to slide to the left of the screen. The pipes moved towards the left of the screen as expected on first try.

**Randomized Pipe Gap:** The pipe gap was too large and was sometimes out of the screen. We shrunk the gap and added a range/limitation that the center of the pipe can be nearing the edges of the screen.

**Linear Jumping:** It was hard to jump and felt unrealistic/unnatural. The player immediately falls as soon as the jump tick counter reaches 0. We adjusted gravity to make it lighter but left the mechanic as is for now until we worked on the exponential/curved jumping.

**Pipe Collision/Math**: Rather than creating two game objects for both the top and bottom pipe, we used math to calculate the ‘safezone’ y-position that the player must be in when the pipe and player are on the same x.

On the first attempt, the player immediately died because the math was just wrong.

The second attempt, the player would die after passing the pipe because the pipe still considered the player to be inside of the pipe. We adjusted this by checking if the players left had passed the pipes right.

On the second attempt, it worked like we wanted.

**Player Death:** We wanted the player to die when they make contact with the pipes. We had a variable to cancel the players user input. It worked as expected and the player would fall. Then we added a tick counter to delay the death screen pop up. It took a couple of tweaks to find the right amount of time to wait before the screen appears.

**Adding textures to pipe using for loop:**  Noah wanted to add more design to the game so he drew three images. The first is the bottom of the pipe (which we didn’t use), then second is the body, and the third was the top of the Echo dot. We used a for-loop to apply the images onto the pipes. The body images would be loaded in first and then the top part would overlay the body image. It took several attempts.

Attempt one, the image was stacked on top of each other, it did not yet follow the pipes x position.

Attempt two, the images were now following the pipes x position. But the images were still stacked on top of each other.

Attempt three, the images were now separated into their own rows but the math was off. We added an offset for the last pipe so that the top image would be offset by half its height. This was to ensure that the user doesn’t get confused by bad collision.

**Pipes exiting screen on win/death or pass:** To create an ‘exit’ effect where the pipes would widen and disappear, we added a variable to check if the pipe has already been passed by the player. Then we checked to see if the player either died or the game was complete. When it does, we continuously incremented the pipe gap variable by one every time the update method was called. When the right side of the pipe was less than 0, out of screen, we removed the pipe object from the Pipes array list so that it can be deleted. The animation, checks, and taking the pipe object out of the list all worked first try.

**Pipe head changing color according to the current player status:** We drew different variations of the pipe top. On default, the ring is blue. When the ‘pipe passed’ variable is true, the ring turns green. When the player dies, the ring turns red. When the player wins, the ring turns orange. The orange and red rings were added last. The different images were loaded as variables and then we used an if statement to check which conditions were met to know which top image to load. The orange ring is top priority, red is second, then green. If no conditions for those were met, the default is blue. There were no problems with this and it worked as expected.

**Curved Jumping/ Fall Acceleration:** We wanted a smoother feeling jump. One where the start of the jump feels faster and slowly decelerates when reaching the peak height. Then there is a slight delay before the player begins its descent. We achieved this by using two tick counters. The first is the jump tick which is reset to the max tick jump (20) and the fall tick is set to 0 when the player presses the space bar. The tick variable is decremented on every frame update until it reaches -2. We modified a gravity formula (<https://www.omnicalculator.com/physics/acceleration>) by adjusting the constants, applying the tick variable, and removing final speed. The tick variable adjusts the velocity, it moves fast at 20, and slows down while it decreases. The calculation is only applied when the tick is greater than 0. The reason why the tick can reach -2 is because we wanted a small delay before descent begins. Descent is the same calculation as the acceleration formula except negative and with a smaller constant. (0.75 is jump, 0.5 is fall). When the jump tick reached -2, the fall tick would begin increasing from 0 until the player presses jump again.

On the first attempt, the player jumped continuously.

The second, the player jumped normally but fell to hard and quick.

The third, we added the -2 check to add a delay before falling.

The fourth, it worked but we needed to tweak the different variables to make it feel more ‘fitting’ with the length of the pipes.

After X amount of tests tweaking the different variables, we got what the results we were satisfied with.

**Point Counter:** The point counter was simple enough, we stored a points variable inside the player class and had a method to increase the points variable by one every time a pipe was passed. When the pipe object is updated, it checks to see if the player has successfully passed the pipe and then it calls the player method to increment the point’s variable. Then the pipe sets its ‘passed’ variable to true so that the other mechanics can operate.

On the first try, the point was added as soon as the player’s right passed the pipes left. We didn’t want this so we adjusted the math to check if the player had completely passed the pipe before adding the point. Results were as desired.

**Progress Bar:** The progress bar was a class that extended game object. When the display method is called, the game draws a rectangle that is a third of the screen positioned on the top-center. Another bar is drawn on top of the rectangle that represents the ‘completed’ percentage. It used division to create a ratio that adjusts the size of the rectangle. E.g. 15 points/ 30 points required = 50%. Meaning half of the rectangle. Afterwards, we added a truck on top of the end of the progress bar to spice it up.

There were no problems with it, we just tweaked the sizes of the progress bar a couple times before getting the right size.

**Stars awarded depending on point progress/ Death Screen:** We used circles and rectangles to draw a rounded box. It took several tries to get the right sizes. We then removed the fixed numbers and replaced them with variables so that we could adjust the box with ease. We used a for-loop to draw stars on the screen. It used a for loop that ranges from one to five and division to check how many points are needed to earn a star. Currently, the player needs to pass 30 pipes to win, 30/4 means 7.5 pipes to earn a point. It is divided by four because the lowest rating you can give a review is one star.

It took X amount of times to draw the rectangle box.

We initially wanted the stars to appear large and on top of the rectangle box but we adjusted it to appear on the top right of the box.

The first try, the two stars were drawn on the very right of the box and the rest clipped off of the box.

The second, the boxes appeared on the top right of the box but there was no margin so the fifth star was touching the edge of the box. We added an offset so that the last star is a couple pixels from the edge.

The third, the stars were filled from right to left. We tried to fix this by rewriting the placement equation and using a-for loop that counts from 5 to 1. It didn’t work…

The fourth attempt, we count from 1 to five again but now used an if statement to check how many stars the player is supposed to have. If five minus the current iteration was a filled star, the filled\_star image would be drawn. The x coordinate equation was also adjusted. It previously drew from right to left but we added - ((starWidth \* star) + gap \* star) to draw from left to right. It worked.

**Randomized Reviews:** We used Random and lists to generate randomized reviews. When a result object is made, a random number from one to twelve is chosen and then used to pick a month in the list of months. Then a random year is chosen within two hundred years from 1996. Lastly a day is randomly chosen from 1-31.

When the player dies, we check the points and see how many stars the player received. We use this to check which list to pick a review from. Then we simply draw the reviews onto the screen.

On the first try, we used placeholder text to find the best locations to draw the username, date, and descriptions.

Afterwards, we replaced the placeholder text with actual information. Unfortunately, the descriptions were changing on every frame. We fixed this by checking to see if a review had already been chosen, if it has, do not pick another review description.