Christopher Figueroa

7/25/23

HW 6: I Used AI to Make Flappy Bird

Overview:

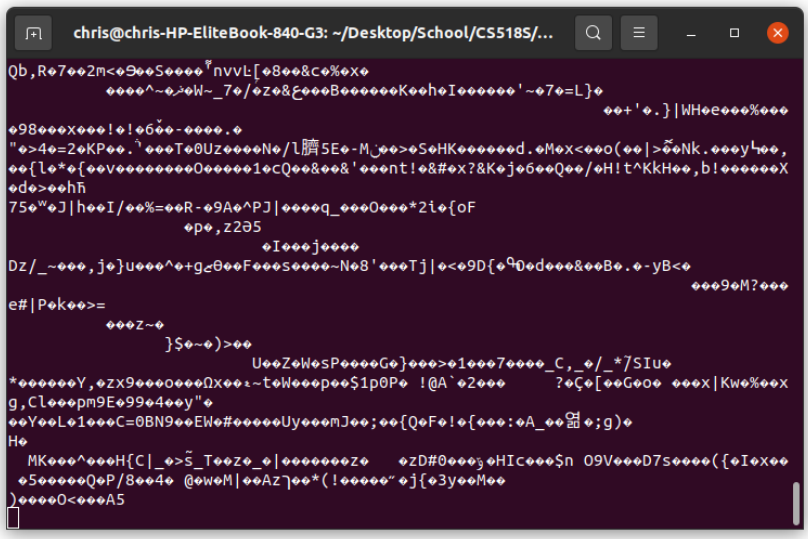
For the final assignment of the class, I utilized ChatGPT and a little bit of Rix to generate a simple game inspired by Flappy Bird. While skimming through the assignment description, I failed to notice that I had the freedom to make the game in other programming languages. For this reason I created the initial program in C++. However, once I installed Pygame into my Python install, I was able to create a Python game with graphics in the file "FlappyRocketPython.py"

The process of creating the game involved a series of steps. The first step involved the implementation of keyboard controls. The second step involved the use of ASCII art in place of image graphics. And the third step would have utilized the use of game engines in a language of my choice to use the graphics in the "Photos" folder. The third step involved the translation of the C++ into Python, and the addition of graphics into the game.

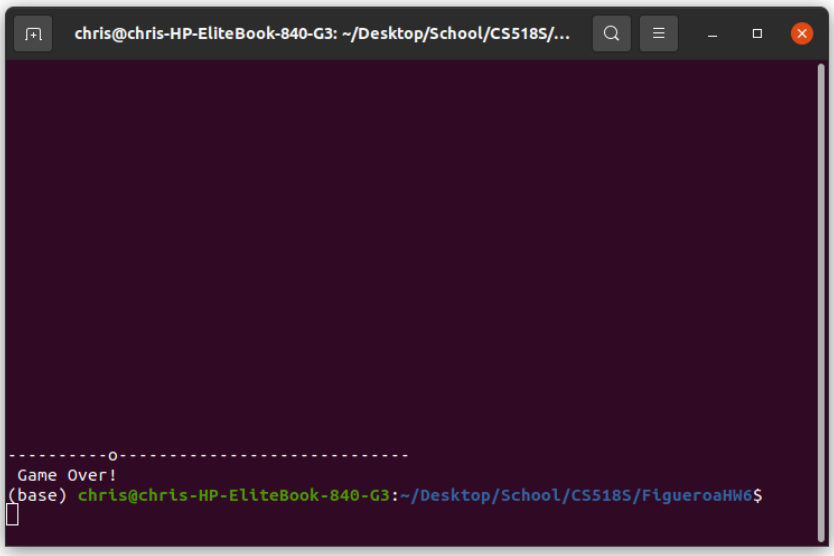
Getting the Keys to Work:

For implementation of the keyboard, I began my initial prompt with Rix. I used Rix at first because this AI yielded working ASCII code in the last assignment. I told Rix to generate a simple game that can move a '.' character with asdw keys. What I got was code that generated only two 'P' characters. After a couple of more queries, I instructed Rix to add an image background. The code instead printed out some garbage output.

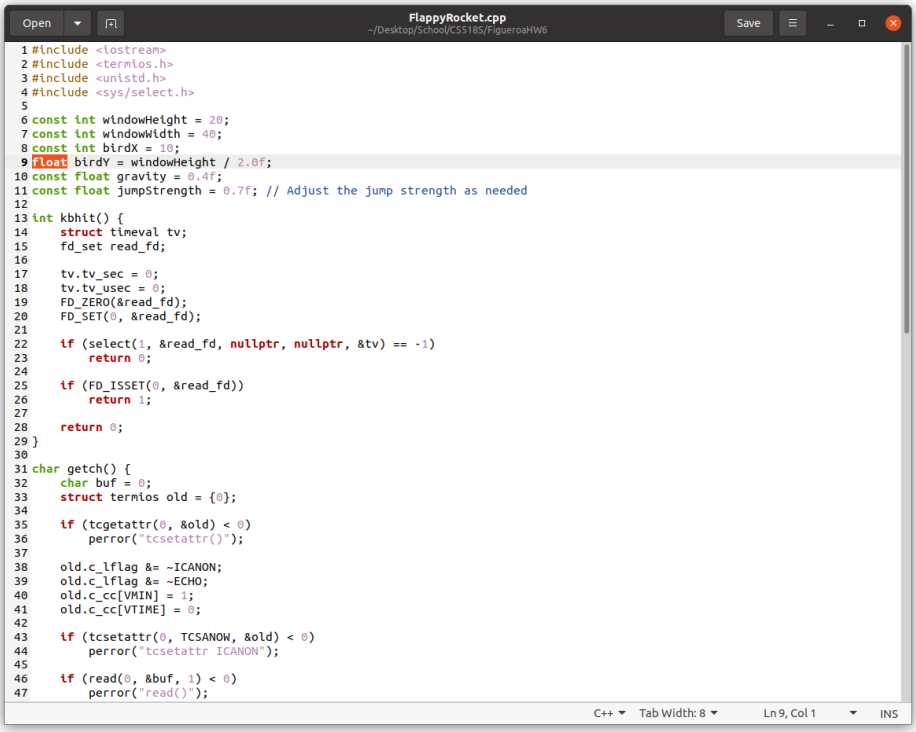
Garbage Output:



With the failure of my initial approach in Rix, I began a new implementation in ChatGPT. For this new approach, I opted to instead instruct the AI to only use one key ‘w’ to move the ‘o’ ball upward. I instructed ChatGPT to simulate a game with gravity, one in which the ball would hit a floor composed of ‘-‘ characters. After a few lines of queries, ChatGPT gave me some code that could simulate gravity and prompt a “Game Over!” message when the ‘o’ hit the floor.

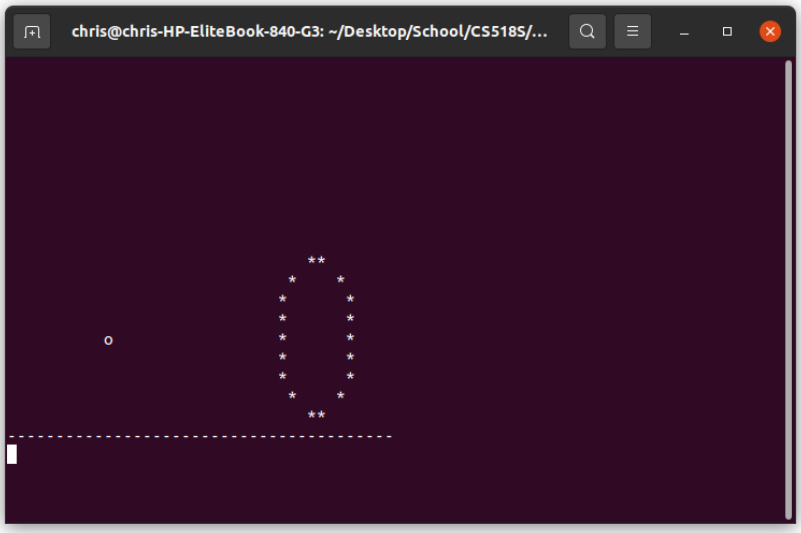


One problem that I noticed with the simulation of gravity was the velocity. Because the code used integers instead of floats to perform calculations, if the gravity was set to a value less than 1, a decimal of 0.7 would be interpreted as 0. I instructed ChatGPT to alter the code so it performed the calculations using floats. This alteration allowed the game’s gravity to be weaker.

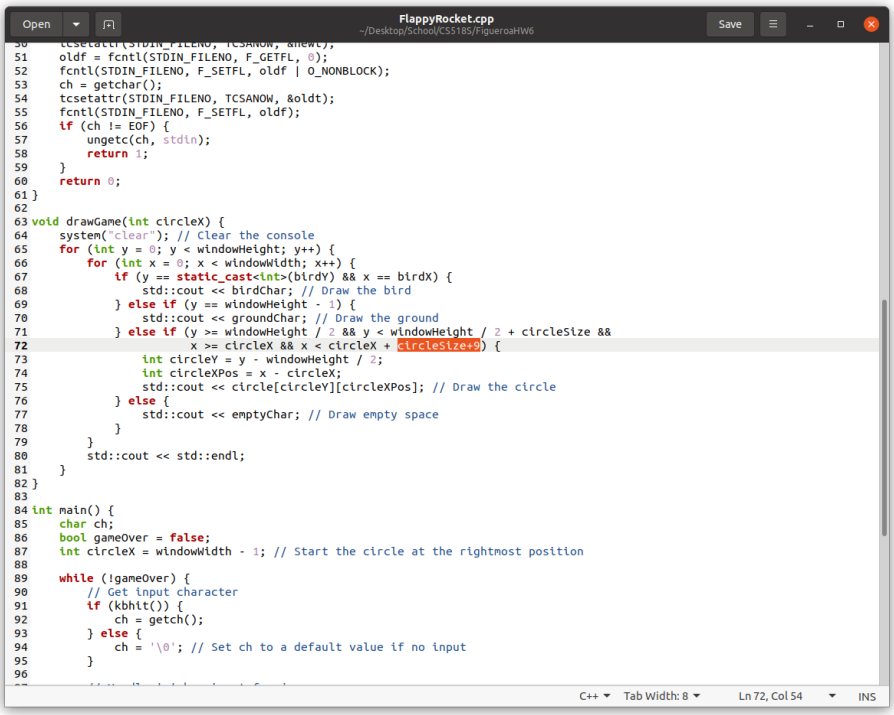


ASCII Art Graphics:

With the implementation of the keyboard ‘w’ jump control and the floating point simulation of gravity, I then proceeded to add in circles to the game. The first circle that I got ChatGPT to generate was 9-by-9. However this circle appeared too narrow for my liking. I named this game “FlappyRocket”. The circles in this game are supposed to be planets.



To fix this issue, I altered the width of the circle to be 18 instead of the initial value of 9. I also manually altered the matrix that represented the circle itself to better represent the spherical structure of planets. The screenshot below highlights the change I made to the code to accommodate the new width. I made some attempts to create some code that dynamically accepted any dimensions for the circle, however these changes resulted in “segmentation” errors.




```
51    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
52    fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);
53    ch = getchar();
54    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
55    fcntl(STDIN_FILENO, F_SETFL, oldf);
56    if (ch != EOF) {
57        ungetc(ch, stdin);
58        return 1;
59    }
60    return 0;
61 }
62
63 void drawGame(int circleX) {
64    system("clear"); // Clear the console
65    for (int y = 0; y < windowHeight; y++) {
66        for (int x = 0; x < windowWidth; x++) {
67            if (y == static_cast<int>(birdY) && x == birdX) {
68                std::cout << birdChar; // Draw the bird
69            } else if (y == windowHeight - 1) {
70                std::cout << groundChar; // Draw the ground
71            } else if (y >= windowHeight / 2 && y < windowHeight / 2 + circleSize &&
72                       x >= circleX && x < circleX + circleSize+9) {
73                int circleY = y - windowHeight / 2;
74                int circleXPos = x - circleX;
75                std::cout << circle[circleY][circleXPos]; // Draw the circle
76            } else {
77                std::cout << emptyChar; // Draw empty space
78            }
79        }
80        std::cout << std::endl;
81    }
82 }
83
84 int main() {
85    char ch;
86    bool gameOver = false;
87    int circleX = windowWidth - 1; // Start the circle at the rightmost position
88
89    while (!gameOver) {
90        // Get input character
91        if (kbhit()) {
92            ch = getch();
93        } else {
94            ch = '\0'; // Set ch to a default value if no input
95        }
96
```

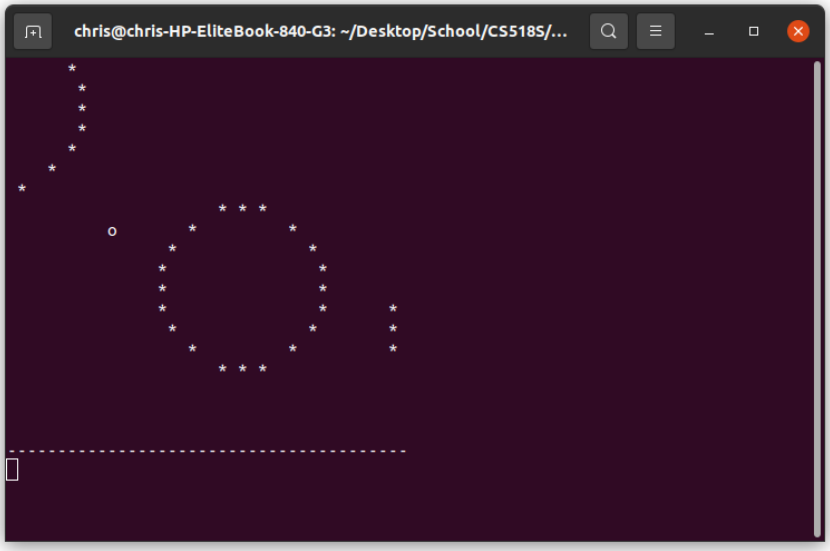
Another issue that I faced in the generation of the ASCII circles was the game's inability to detect collisions with the circles. However after a few queries, I got ChatGPT to detect the collision with the circles. I also added in a condition to throw the "Game Over!" message with the collisions with circles as well as with the ground.

Circle Collision:

The next issue that I had to overcome was the generation of multiple circles. The first set of code that ChatGPT provided me generated one circle after another all equidistant from one another. I spend several more queries trying to get more circles to appear in randomized locations. At one point I got the code to create circles that overlap. However, whenever I tried to set a distance between these randomized circles, I got a "segmentation" error.

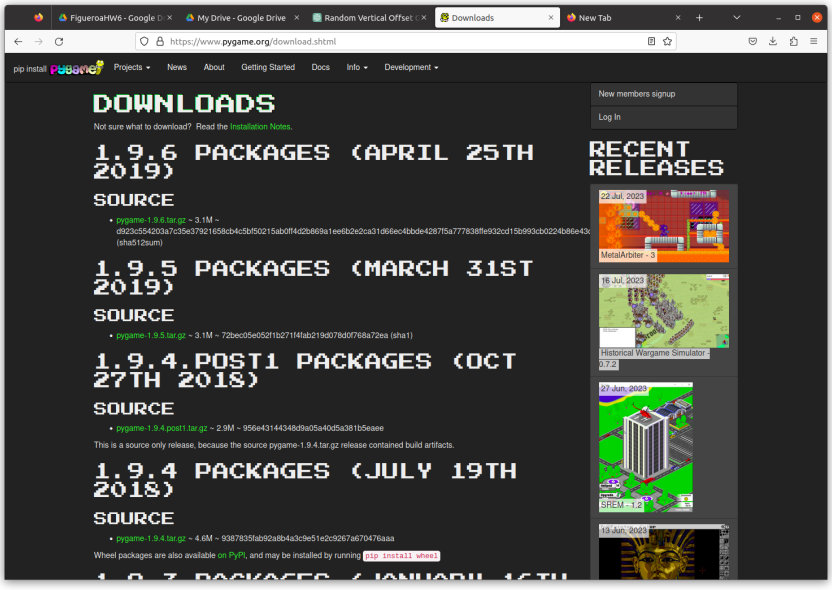
Overlapping Circles:

To avoid more segmentation errors, I reverted my code back to the point where it generated equidistant circles. Instead of generating random distances in both the X and Y axis, I queried ChatGPT to instead only generate random values for the vertical Y values. This approach generated some unpredictable results. However after some time, I was finally able to get the circles to appear at varying heights.

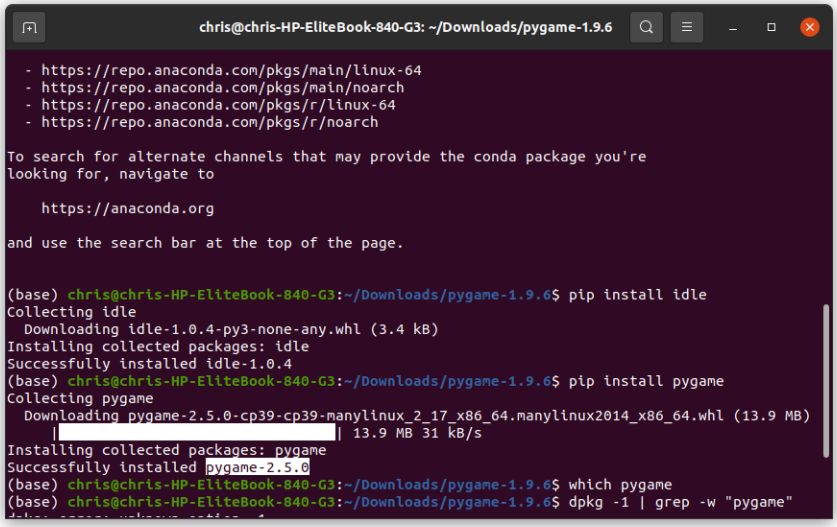


Python's Pygame Engine:

Due to the limitations of C++'s default libraries, I had to choose another platform in which to add in the graphics for the Flappy Rocket game. I selected Python's Pygame library to build to continue with the creation of my game. The installation of the library was relatively straight forward since I already had Python installed in my Linux system.

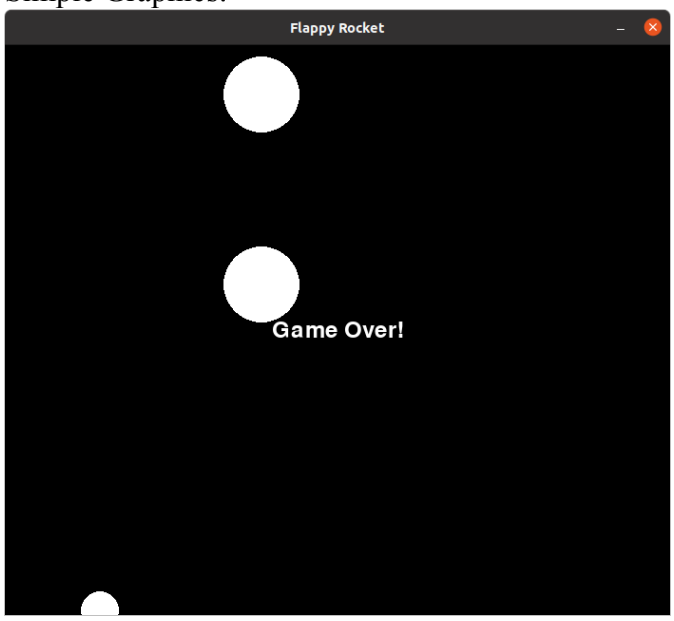


Successful Pygame Installation:



Once I installed the Pygame library, I queried ChatGPT to translate the C++ program "FlappyRocket.cpp" into Python. I ran into an index out of range error as I ran the newly created "FlappyRocketPython.py" file. After a number of queries of repeated failures, I proceeded to instruct the AI to instead use built-in Pygame graphics in place of the original ASCII graphics. The replacement of the ASCII image generation with simple graphics corrected the out of range error.

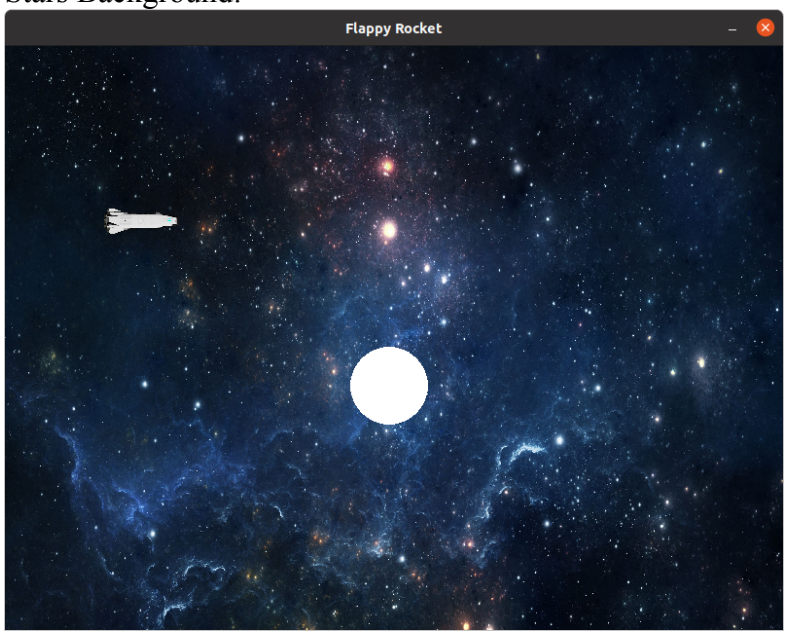
Simple Graphics:



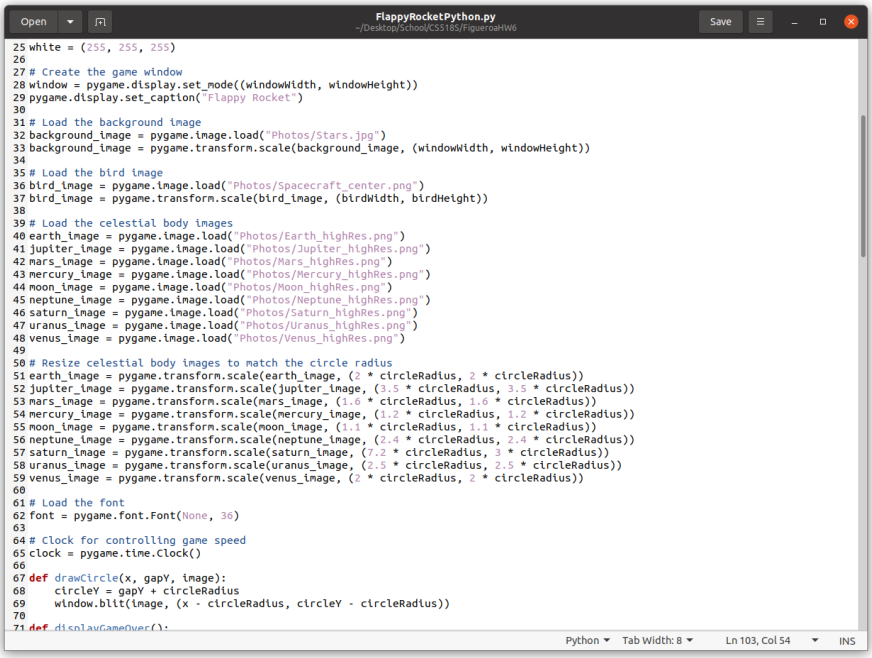
The Pygame library has the capacity to create simple geometric shapes as well as the ability to import images from folders. The two larger circles in the image above represent planets. The smaller circle represents the rocket. The game generated a "Game Over!" message when the rocket touched the bottom of the screen or one of the planets as expected.

With the success of Pygame's simple graphics generation, I then queried ChatGPT to add in images from the "Photos" folder. The first image that I added was the "Stars.jpg" file. I instructed ChatGPT to set this image as the background for the game. The stars in the newly added background appeared narrow, so I altered the width of the window in response.

Stars Background:

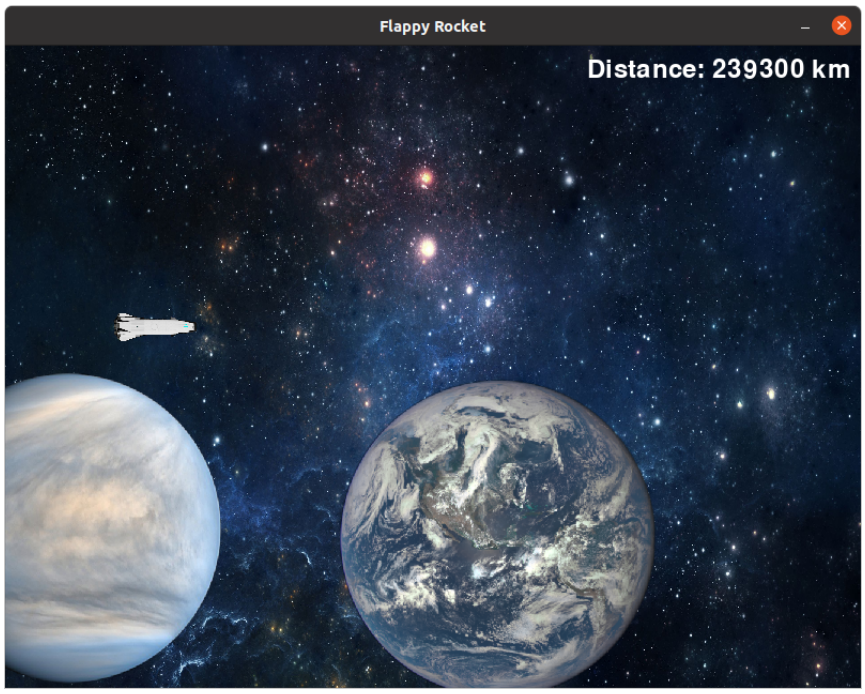


Planet Images:



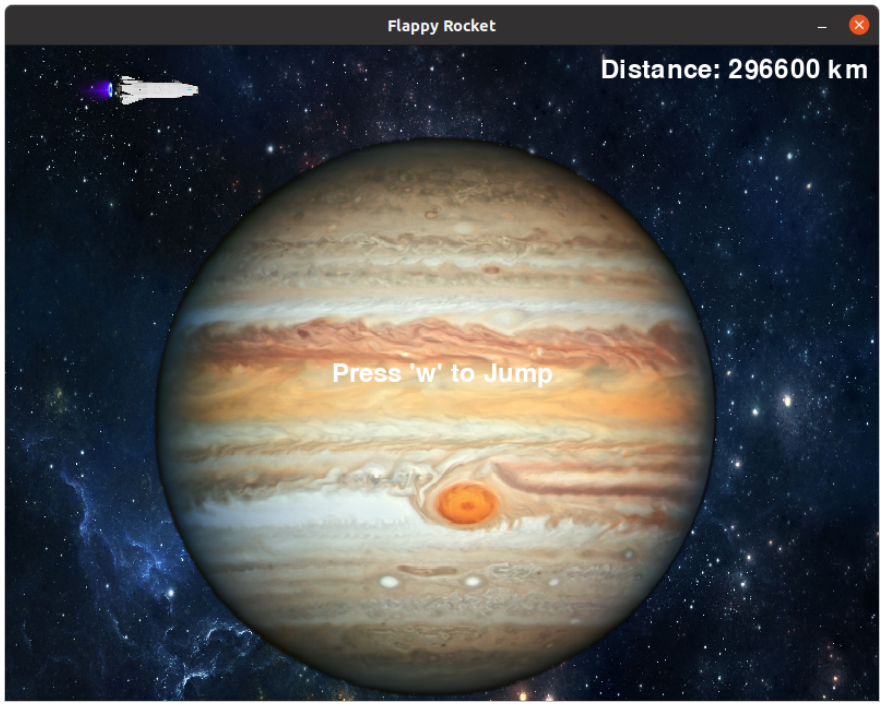
My next step involved the replacement of the existing white circles with the images of planets. I instructed ChatGPT to use 9 images, and to select at random one of those 9 images to represent the newly created planet. As of now, two planets appear on screen at once.

Planet Generation:



The planets that appear on screen vary in size. In the case of the screenshot from above, Earth and Venus appear to be the same size. If Jupiter was on screen, it would be roughly double the size of Earth. Another detail that I added to the game as a way to keep score was the inclusion of a “Distance” meter. This distance graphic is really just a renamed timer that converts miliseconds to kilometers.

Exhaust Generation:



With some additional modification and queries from the AI, I added in an exhaust animation for the rocket and an rocket sound effect from my “Sound” folder. And as you can see, Jupiter is considerably larger than Earth. Whenever the rocket hits a planet, an explosion animation appears along with a message that says “Press SpaceBar to Reset”. The screenshot below exemplifies what happens when the rocket crashes.

Game Over: