oHNO3

An OpenGL Game By Tyler Armstrong, Ralphilou Tatoy, Vedaant Vyas, and Sy Loc

Introduction

So far, on our CSE journey studying here at the University of California Merced, we have been dealing with writing code that displays in the terminal. However, for this section of our Object-Oriented Programming class, we are taking it to the next level, by making a mini-video game that uses an API to incorporate graphics, something we have not yet been exposed to. Although we will get more into detail about the game in our "Methodology/Implementation", a brief overview here is as follows. Our game, called oHNO3(pronounced "Oh No Three") is a top-down game in which a character is controlled by a user, who tries to avoid deadly acid clouds. The player has a set amount of health but can get added protection from shields, by making contact with an umbrella powerup. The goal is to survive for as long as you can. Clouds in the game will start as normal clouds, but slowly they will become filled with rain and turn dark then yellow due to pollution from the atmosphere, and rain acidic precipitates. When the player is under a cloud that is yellow and raining acid, his shields/health will deteriorate by 5 points, and when the player's health gets to 0, the player dies, and its game is over. Our game is coded in the C++ language and utilizes the OpenGL API to communicate our code to the computer's hardware so graphics can be rendered. The required library SOIL(Simple OpenGL Image Library) was used to load images, and the library SDL to load sounds. More details on the libraries will be given in the "Methodology/Implementation" section. When it comes to the platforms being used, we built our game primarily on Windows, but we also used Linux for testing the game. After the methodology, we will discuss the outcome of the game, as well as what is next for our game, and what we can do to build upon this game past the scope of this class, and what the future of the game holds.

Motivation

There were two main motivations for making this game. First, it is an important/required part of CSE 165/ENGR 140, and it takes up quite a bit of our grade. Not doing this video game project would harm our grades greatly. However, the course grade was not the only motivation to build this game. As mentioned before, UC Merced does not have a dedicated game development class or game classes that are regularly taught every year/semester, so this project was a perfect way to get experience in C++ game development. Additionally, many of us have not installed external libraries for programs before this project, such as SOIL, freeglut, glew and SDL2. Doing this project was a perfect way to get experience in using external libraries. Lastly, the largest motivation by far in building this game was to share and bring enjoyment to other people. As children, we grew up playing many games such as Super Mario Run, Brick Breaker, etc. To create a game that children nowadays can play and or learn from would be very rewarding, and the experience gained expands our future employment prospects.

Methodology

Now that we have a brief overview of the game, as well as the motivation for creating the game, let us dive into how the game was created, and the process of working together to achieve the final product. There were four of us working on the same game, however, it would not be efficient for us to work on the same computer. To mitigate this, Tyler the teams' leader set us up to use Github and git repositories, so each of us could individually work on our code without changing what others were working on, then we pushed our changes to repositories branched from the main repository containing the primary code for the project. Ralph did most of the big coding mechanics and much of the existing code was from him directly or manipulated from his code. Vedaant was put in charge of Linux compatibility and coding since his computer OS is Ubuntu, and sy was given the role of beta tester running the code and suggesting changes and fixes as problems arose. Tyler would combine and resolve conflicts from our pushed code to the main project repository, so we could then pull from that repository, compile the code, and run the game on our machines for sy to test and see where changes needed to be made.

Now that we have explained how we set up our working environment, let us take an in-depth look at how we set up the libraries that we used for this project. As mentioned in our introduction, there were numerous libraries that we installed and used for this game. We were required to use the OpenGL API, which is a common API used for building simple games and complex games. We were also required to use SOIL, which is a small C library that makes loading images easy in OpenGL using the GLUT libraries. This way, instead of being limited to colours and shapes using GLUT, we were able to use our image files, which was important for our game. We also used the library SDL, even though it wasn't required, we used this library to play sound effects, as well as to implement additional functions like more image loading and key binding. As far as installing these libraries, the difficulty depended on the operating system. On a Linux machine, installing the libraries was simple. We just needed to run some command lines in our terminal. However, when it came to installing these libraries on a Windows OS, the process was much more complicated. For windows, we had to install a packaging program called MSYS2 to install the MinGW compiler and libraries including an extra one glew so we could get the most recent OpenGL version. In the end, we're glad none of us were using a Macintosh OS since Apple has depreciated the use of OpenGL and would have been much harder to set up.

Now that we have seen how our libraries were set up, we can go into depth about the core of our game, which is the code that was used to make the game. Going over every single piece of code would be somewhat redundant, and very time-consuming. Instead, we will go over each source code file in our project, and what the uses for each of those files are, as well as what functions were implemented in relation to our libraries. We have four header files in relation to our objects. In our first header file, we code functions relating to our character. We implement functions to get the input from the user and depending on the arrow key that the user presses, we have the character move in that direction. In addition, we also have functions relating to the character interacting with other objects, such as the clouds, and the shield umbrella. Our next header file relates to our clouds. Using a time function, we program the clouds to move back and forth across the screen. As time passes, we render more clouds to the screen. We also have a function to have the clouds occasionally change into the acid rain cloud, which is what hurts the player. Our next header file is responsible for loading images, and this is where we use SOIL. We have a function that loads our chosen images onto the screen, and we also have a function that loads a font, which is used to display parameters such as time elapsed, health, shield, and current highest score. Lastly, we have a header file that is related to our powerup, which is an umbrella that we have on screen. When our character comes into contact with the powerup, he gains 10 shield points for a maximum of 50 points, which is used to provide extra health to our player. We spawn the powerup at random coordinates every 15 seconds and disappear after being picked up by the player, so the user will not know where exactly to go to receive the powerup and only receive it once. There is also a .h file called global which contains extern variables so that they can be shared between files on a global scale after being initialized, these extern variables were mainly used for timers and sounds. In total, the game utilizes 4 different player images, 3 different cloud images, 3 different game backgrounds, an umbrella image, a start screen background, and an end screen background as well as a text font file. For sounds, we have rain, an umbrella opening, and acid burning which is actually a cigarette burn sound due to the lack of accessibility to short acid burn noises. Lastly, we have a .cpp file that contains our main function, which focuses on the functions of OpenGL and SOIL.

Now that we have gone over the object header files, let us go over the rest of the files and code we used for building this project. We have a header file that is used for the detection of objects. This file works by detecting whether or not the character's image hitbox has overlapped with the powerup or a cloud if so it damages or shields accordingly. We have a function that tracks the amount of health that the user has, and decrements a set amount of health when the player comes into contact with a dangerous cloud. Once the player's health reaches 0, the game is over. We have a .cpp file in which we use for starting and ending the game. We use SOIL to load images for the “start” screen and “game over” screen. We also implement a function for the user to either choose to play the game when they start it or quit. On the “game over” screen, we provide a play again or quit option. If the player chooses to quit then the program closes, if the player chooses to play again the game returns to the “start” screen providing its options all over again. We then have a small header file providing global declarations of objects. However, the core of our game relies on our source code file containing our main function. In this file, we load and display each of our windows, including our start screen, the game itself, and the “game over” window. To generate the window, we call some Glut functions, and we specify a name for our windows, as well as the size of the windows. In this file, we call the functions that we had implemented in the rest of our header files and .cpp source files. We call the functions to move the player, move the clouds, change the state of the clouds, generate the powerup, and keep track of our score (based on time) as well as our health and shield. In a sense, if we wanted to show the user a basic overview of our code, we would show him/her this file, as it shows how the functions are called but hides how the functions are defined.

In summary, our methodology consisted of installing the required libraries, setting up a work environment, and then, coding the game. Of course, not all the coding went smoothly. Even after we got the libraries set up, we still had issues properly implementing them in C++ code. Even then, one flaw we had was related to the time functions in Linux vs. windows. Since a Linux CPU runs slower, not only did our timer run slower, but the state of the clouds changing was slower as well. However, when everything did run smoothly, it was very rewarding, and it was great to see us making progress.

Outcome

The outcome of this game was not what we originally designed since we originally wanted to make a platformer game such as Super Mario Bros, but we are still impressed by the result of the top-down implementation of the same concept. The game is fully compatible with Windows but takes a bit to play the first time after compilation after which the .exe file plays the game faster. Additionally, numerous beta tests conclude that the game runs perfectly on windows with very few bugs and no known persistent issues. The game is compatible with Linux, but there is an issue with time running much slower, and because most everyone is working with windows we don't have that much research about how we should change to be compatible with Linux to run the time functions faster. With the use of a 2d-coordinate system, we can create detection functions between our objects such as powerup, clouds, and player. Also, we can display these images in the proper position on the screen due to this system. With that, the object could move by updating the coordinates of the images and rendering each time it was changed, implemented within each respective class object. Each object moves differently such that these changes will have different formats. We used various OpenGL libraries to display our game such as SDL and glut. With the use of soil and SDL, we can render images. We are also able to display information such as health, shield, and more for the player through SDL and glut. In unity, we can make a game that is somewhat addicting and fun to play. Even though we have a completed game as of now, we do not want this to be the end of our journey. The game can still be improved and built upon. We plan on continuing our work on the game to include other options on the start screen, such as options such as mute or reduce volume. A credits page where we will have our names and maybe pictures of ourselves, or pictures of our discord avatar. If we make further progress on this game, we may even release future editions, for example, "oHNO3, Part II"

Conclusion

In the class CSE 165/ENGR 140, we have been learning about Object Oriented Programming(OOP) for the whole semester of fall 2021. After we were given instructions on how we were to complete the project, we were slightly confused because we have only about a month until the end of the semester and we had never even heard of much less learned about or how to use OpenGL or SOIL before. At first, everyone in our team thought this was going to be an interesting project because most of us did not have any experience in coding video games. We thought it would be simple to install OpenGL and SOIL on our personal computers.

When the project was initially announced, we were waiting to see how to set up OpenGL and SOIL, but we were having trouble with where exactly to look. but we still didn't see anything. After that, our team set up a meeting with every member to set up the Github repository, so everyone can push and pull everything from the git repository but was a struggle of a few days to find a solution, but once we did, everyone was able to share and everyone was able to push and pull without conflict. After we completed setting up the git repository, we still had issues setting up OpenGL and SOIL on a Windows OS. So we were having meetings and researching how to set up OpenGL and SOIL. Later, we were able to quickly set up these libraries on a Linux operating system, but still had trouble setting it up on windows. While doing this, we also found out OpenGL and SOIL on Mac OS are depreciated. We did a lot of research about it from the internet and tried installing a lot of things like GLUT, GLFW, SDL, and MSYS2 for Windows, Linux, and OS/X, from the reference materials that were provided. We still had no idea how those things work, even after reading an online guide multiple times. Then we went to Youtube to look for tutorials on how they are setting things up, and even though we found many examples of setting up OpenGL on windows, we still had issues setting up the SOIL library, since Youtube was vague about SOIL. Additionally, when we requested help on the internet we were met with many replies of “why would you use SOIL”, and no actual answers. After 3 weeks, we figured out how these things work. This was the most difficult thing we went through in the CSE 165/ENGR 140 final project. Through this project, we have learned a lot of things, such as how to facilitate teamwork in the world of CSE, and that we need to rely on more than just resources given to us in the course, we need to go above and beyond to find adequate resources.

However, our game looks simple but it is not easy to play. The idea of this game is to help people improve their eye reflexes and create a highly concentrated and improved connection between brain and hand reactions. Every time that people get stressed from work or anything, they can play this game for a few minutes such as 5 to 10 minutes to forget their stress and have a little improvement in their concentration and reaction. Not only that, but the ambiance of the game is somewhat soothing. The sound of falling rain combined with the image of a grass field is a great way to calm your mind, and you can have fun doing it while playing a game. Even though this is just a basic idea that we are working on and it does not get verified by any experts, we will continue to work on this game in the future to develop it into the Windows Store, App Store, or Google Play to allow other people download and play. If that is not plausible, then we will make a website in which users can play our game in their browser. If we want to expand our journey, we may even think about making a game like this for a console, even though we would have to greatly expand on it. However, a positive note is that we are not limited to just SOIL and OpenGL now. We can use tools such as game engines to take our game to the next level.