Matrix Typing

Project Overview:

We created an interactive typing game in the style of the matrix to make typing fast fun and interesting to help them learn to type faster. In this project we aimed to make a fun game that challenged the player to type the correct letter when it was in the bottom of the screen. To do this we broke up our program into model and interfacing components and implemented them such that the model contained the information for the running program and the user interface processed the information and presented it to the user, accepting user input in the form of typing.

Results:

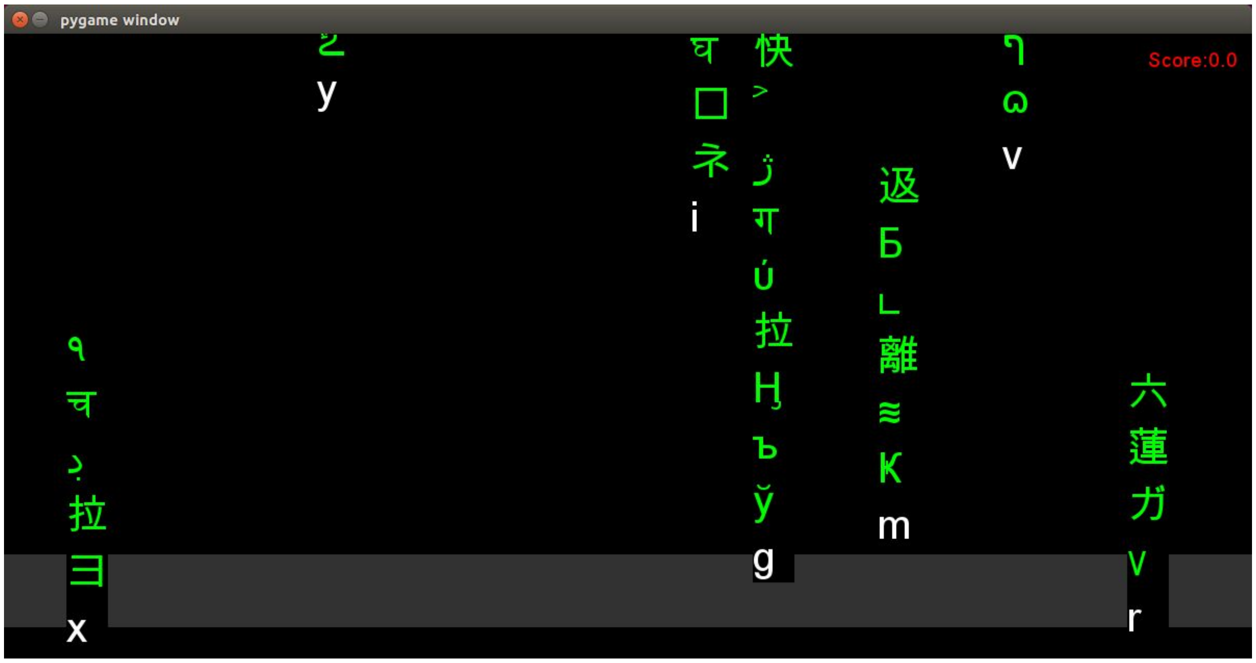

Figure 1: Example Game Window

Our final product as seen in figure 1 has white letters that fall and intersect from the top of the window and are trailed by green text. When the white letters fall into the grey box at the bottom of the screen, the player types the letter. The score, as displayed at the top right in figure 1, represents the percentage of the letters that appear that the user successfully types while the character is in the grey box. If the player types the wrong letter, or the does not type the letter while it is in the box, the player's score decreases.

Each white character that falls is trailed by a series of green unicode characters. These make the game look like the scrolling text of the movie “The Matrix”. This increases the aesthetic appeal of the game and makes it more entertaining for the player. The green text is in unicode to keep the player from getting confused by the additional text. The characters of both the white text and the green text are randomly generated as well as the length of the green text. This gives the classic cascading effect of the 1999 film.

Implementation:

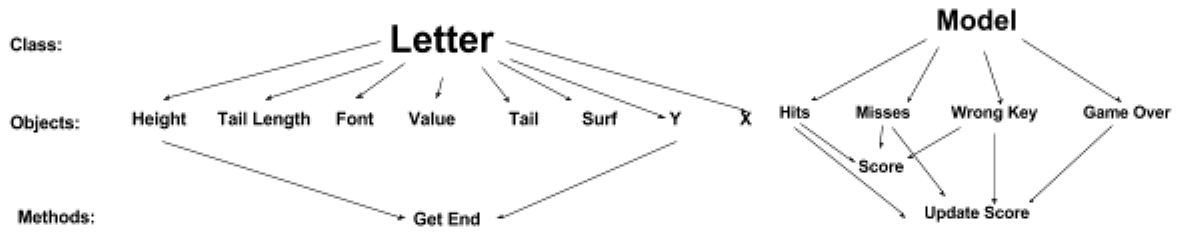

Figure 2: UML Class Diagram

We divided our code into 3 different scripts. Our model script contained the classes in which all of the information about the current status of the game was contained. This included the class model, which kept track of score and class letter which kept track of all of the information pertaining to a letter on the screen as objects with methods as seen in figure 2. This includes location in the x and y directions, alphabetic character, and font. The user interface then sets up the screen using pygame and updates the letters and score continuously. The user interface script is in charge of knowing the rules of the game, such as what to do if a letter gets to the bottom of the screen and is the key associated with that letter is not pressed at the right time.

We chose to include the rules inside of the user interface script so that it could seamlessly interact with the user's input to the system. Because the program has to do different things if the correct key is pressed, if the wrong key is pressed, and if no key is pressed, we found it easier to have the key presses feed directly into the code that is in charge of determining what the game should do with the key presses it receives. The third script contains a random unicode generator. Because unicode characters are defined as characters only in certain ranges, we had to create a function that would create an alphabet of unicode characters from which we could select one in our model script.

Reflection:

While we had a lot of setbacks in our process, we found that discussing our plan for the whole code and then dividing up the work and separately working on those parts was an effective way of working as a team to create one cohesive program. While working, we were able to go to one another and ask for help on our sections of code, allowing us to understand fully all of the code written. Understanding unicode and how to get it to display unicode in our pygame based program was a point of difficulty in our work. We initially had grander plans including adding audio to our program, due to time constraints we were unable to get to this point. Although we were unable to get to the scope that we had hoped to get to for our project, we knew that goals such as adding audio would be a stretch and as such called them stretch goals at the onset. This made it less difficult when we had to scale our scope down a little to make it feasible. We did a poor job of unit testing in our program. In the way that we went about writing our code we found it convenient to print each step and then delete the print statement once we knew that that stage was working as we expected. If we were to do this again or improve our code, unit testing would be among the many things that we would be interested in adding. Again due to a lack of time we were unable to add this even after the fact. Working as a pair on this project was made especially difficult because one of our team members was actually out of time for a significant portion of our project. This added complication to our coordination in creating our program. We both spent approximately 12 hours at least on this project. When doing similar projects in the future, we would aim to divide up the work a little more evenly and would try our best to work in the same place at the same time on our sections of the project to ensure we are working together.