Connor Nacov Course 4080.330.03 Flash Game Project

**Abstract**

I based this game off of one I made for my High School Computer Programming class. Basically, different letters start at the top of the screen and work their way down, and they need to be destroyed before they reach the bottom. This is accomplished by typing that letter into the keyboard. The tone would be lighthearted and entertaining, with the goal being to simply survive as long as possible and get a good score.

**High-Level Architecture**

After the user chooses to start a new game, the required assets are loaded into the game class, such as external XML for some values, all of the arrays and variables, and layers are added to the stage, each one for letters and the turret specifically. A timer is set up to tick every two seconds and starts playing, meanwhile an event for the Keyboarder API is set up to get user input, and an Enter\_Frame handler is set up to call an update method continuously.

Every time the timer ticks, a new letter is added to the stage. When this happens, the letter is randomly determined and selected, and is created with a falling speed that gets incrementally faster as time goes on. Also, each tick reduces the delay between ticks, incrementing difficulty the longer the game goes.

The update function calls the update functions of each letter on the screen, telling them to go further down the screen by their falling speed. It also checks to see if any letters hit the bottom, in which case a game over is triggered.

When the user types in any letter, the Keyboarder event checks that key against every letter in the letter array. If a match is found, that letter is destroyed and removed from the stage/array, and points are awarded based on how far from the bottom it was destroyed at.

**Design Decisions**

The reason I decided to base this off a project I worked on before was my familiarity with the concept (I didn’t need to brainstorm possible fun ideas) and also I already had a rough idea on the fun factor and difficulty curve.

I added in the idea of a static turret to shoot down the letters that was not controlled by the mouse or arrow keys, because a degree of accuracy would defeat the point of the main concept, which was to type the letters you see. But at the same time I didn’t want them to just disappear for no reason, so the turret and laser beams could give the user a better sense of cause and effect.

If I had more time and motivation, I might have added specific stages, maybe bonus enemies or boss ‘words’, but figured the single letter/survival setup fit the theme well enough, that eventually, the game will end, so get the highest score possible before your reflexes fail you.

The final decision was on the scoring system. I had to decide between a numerated score, and just a simple time trial. I eventually went with the scoring system because the way the curve is set up, most people would end up failing right around the same times, as eventually the letters would fall too fast to react. The scoring based on proximity to the ground gives a motivation to be quick on the draw.

**Technical Decisions**

I went with timeline coded screens instead of the outside screen API because I figured it was unnecessary, and would have little more than an ascetic impact on the finished product. Plus it would be simple enough to contain elements in their respective frames for easy organization and tweaking.

Choosing the random letters was easy once I realized I didn’t need to make a new symbol for each letter, just one letter symbol with 26 different frames, and using a randomly determined ID to choose one. This way I could also put the explosion animation in the same timeline at the end, and play it for each letter.

Also, this is more of an issue than a decision, but I could only get the main screen music to play the one time, at the start of the animation, and stop it when a new game was called. I couldn’t figure out a simple way of restarting it if the user returned to the screen. (I couldn’t just play it from the Back buttons because then when someone returned from the credits or instructions screen, there would be two tracks playing at once.) It’s not that major of an issue since the player would have little reason to go back to the screen from the game over if they had seen the credits or instructions beforehand.

**Extras**

As stated before, my ‘extra’ was the Keyboarder API for input, because it worked well enough for iceteroids and was easy to implement. One thing I think would have been cool is a particle physics setup for when the letters explode, maybe randomly generated colorful shrapnel would fall down the screen, but I just couldn’t find the time to handle it, so it was just an idea.