Plane Attack!

Gabriel Butterick

Michael Costello

Overview:

A side scrolling shooter in which the player uses the wasd keys to control a plane and guide it through an onslaught of straight flying, non-shooting gray planes.

Results:

We accomplished keeping the player within the bounds of the screen, making the player able to fire on command, making the bullets have hitboxes, making the player have a hitbox, making enemies that move and have hitboxes, having the background move to simulate flying better. We used object oriented classes to control everything in our game from the background to the bullets and made them all work together in relative harmony. We learned to better use Git Hub and became more capable in pair programming.

Implementation:

We tried to separate the major components of the game into different classes to keep them unique and separate, which ended up being to our benefit. As the code grew longer, this level of organization was all that stood between us and hopelessness. They are all brought together in the Scroller Model class which serves to update everything using data from virtually every class. It is possible that we could have done everything with fewer classes and used inheritance instead of repeating functions down the line, however, we chose to stick with our method because it helped stabilize and separate things as we programmed even when we became lost or confused. The more we condensed it, the harder it would be to see how all the pieces fit together and learn what worked and why. Our major components consisted of the movement, the collision boxes, and the accepting of player input. At a higher up level, that's all our game is about.

As you can see from our UML list diagram, we did not use inheritance, and instead kept a relatively linear structure throughout the program. We started with a class and put all the necessary functions into that class. This could be the source of some inefficiency, as there are clear repeats in the code. We chose a list format for the diagram because a flow chart would be fairly crowded and ungainly, as nothing moves beyond the second step.

Class: drawable surface:

    Init

    Get surface

    Get rect

Class: plane:

    Init

    Update

    Get drawables

Class: Background:

    Init

    Get drawabless

    Update

    Collided with

Class: Bullet:

    Init

    Update

    Get drawables

    Collided with

Class: Enemy:

    Init

    Collided with

    Update

    Get drawables

    Is dead

Class: ScrollerModel:

- Init
- Get plane drawables
- Get bullet drawables
- Get enemy drawables
- Is player dead
- Is bullet dead
- Plane update
- Bullet update
- Enemy update
- Background update

Class: Scroller View:

- Init
- Draw

Class: SideScroller:

- Init
- Run

Reflection:

What went best in our project was how we divided up work. We would start with the same base code and each hack away at it for a few hours. If either of us were successful, we would join the new code to the old code, and restart the process with the updated code. It made for sure, steady progress, and kept our individual codes from falling too far out of sync. What could be improved would definitely be the efficiency level of our code. In the end, we still felt like we didn't understand object oriented programming well enough to take full advantage of it. Our project was easily scalable and well within the scope of our abilities. We pushed ourselves from the MVP onwards, working in unknown territory to get parts of the game we needed to fit together. We have learned valuable group programming skills, ranging from using GitHub to pair programming. Moving forward, we will have a better idea of how long some things take to implement, the danger of hiccups, the importance of well structured, well commented code, and we now know that asking for help is necessary sometimes. There were a few times when some base structural changes were made to the code, which caused implementing the other person’s features to become much more difficult. We learned that structural changes need to be well documented and talked out ahead of time in order to be beneficial.