David McCullough

ATEC 4373.003

28 April 2017

Semester Project, 3D Platformer: Post Mortem

My final project was a 3D platformer, and I chose to implement swimming and wall-kicks as extra features. I’ve developed several 2D platformers before, so I wanted to see how effective it would be to translate my existing knowledge of platforming mechanics into a different space.

**Things that went right**

I expected the camera system to be the most difficult aspect of this project. While a 3D orbital camera was new for me, I had the system working with less lines of code than I thought would be necessary. While my first iteration did not work perfectly (sometimes the camera wouldn’t detect a collision and would end up inside a wall), discussing the pitfalls of using a raycast instead of a spherecast helped me get the camera collisions working almost immediately.

In class, we also discussed alternative movement methods, one of which I ended up implementing. Examining the applied acceleration movement code used in the Source Engine was informative and helped me improve both the feeling and the utility of my PC’s movement while learning something myself.

I am also happy with how the collectable pickups in my game turned out. While it was a very straightforward process, having a decent variety between each pickup’s look and feedback added at least some character to the otherwise plain looking project. The XP capsules especially added satisfying layer of feedback that took a relatively short amount of time to implement.

**Things that went wrong**

As mentioned, I was happy with the result of my movement system. However, early in the project, the PC’s movement was wonky and unsatisfying. This was because I tried using the same method I had previously used in a 2D medium, but this route didn’t hold up. I had separated movement speed along the x and z axis into two respective float variables and tied them directly to the keys which influenced them. This resulted in movement that only worked in cardinal directions that had their own frictions. I fixed this by using the applied acceleration method discussed earlier, adding a new acceleration vector based on input to an existing velocity vector which is influenced by friction.

My swimming system introduced an issue which I didn’t end up fixing because it wasn’t necessarily causing any major gameplay problems, but could have some effects down the line if this was developed further. Water in my game is a trigger. When the player enters this trigger, their state switches to the swimming state, and changes to either the falling or walking state when exiting the water. I gave the ability for the player to hold spacebar to float upwards in the water. Doing so while at the top of the water mesh causes the player to move just far enough out of the water to enter a falling state only to immediately renter the swimming state upon reentering the trigger. While this gives off a nice “bobbing” motion, I couldn’t help but think switching states back and forth from swimming to falling so rapidly could lead to some pitfalls. For one, if the player tries to jump during the couple of frames they are in the falling state, it won’t register. To the player, it still appears that they are swimming so they should be able to jump on out of the water. This isn’t a huge flaw, but it’s there, and I never did think of a good solution.

Balancing my other extra, wall-kicking also proved difficult. I wanted wall-kicking to only be a viable way of reaching greater heights if there were two walls to do it on. This meant I would need the PC to kick off the wall hard enough that attempting to accelerate back toward the same wall was impossible. This was the case initially, but the horizontal force seemed to heavy. I ended up turning the air acceleration down, which did prevent the player from climbing the same wall, but now precision jumps became too difficult. I attempting to balance this out, but I never got a good result within my timeframe, so currently the player can scale a wall without needing another one to bounce off.

**Hindsight changes**

If I started over, I would put more time developing the walking state functionality. Currently, it transitions to the falling state as soon as the floor underneath the PC is gone. The PC isn’t bound to the ground either, so moving down a sloped surface caused the PC to enter the falling state, as it ramps on and off the slope. Finding a way to detect a sloped surface, and then stick to it would allow for the inclusion of sloped surfaces in the level, enabling more options for gameplay.

I also would have made a pickup base class and create other pickups that inherit from it. I didn’t foresee any big differences in functionality between collectables, so they all share one class and are differentiate by a single enum variable. This resulted in some hacked logic, which could have been avoided with polymorphism.

**Continuations**

If I developed this project further, I would maintain the player state system and the basic movement system. Both of these features are a solid base that could easily be tweaked or built upon down the road.