Fat Squirrel: The Game

Over the course of its development, the prototype for this game went through multiple metamorphoses. Whether they were design, logic, or aesthetic, changes were progressively made to achieve a smooth game that was entertaining to play. While the main theme, fat squirrel eating food in the Diag, never changed, the original idea to develop a resource-investment game quickly went out the window. A few reasons for this divergence were the convoluted of code hierarchy and cluttered game objects that would be incurred. After pitching the original idea to my colleagues, we decided that having a functional *and* entertaining upgrading system would likely be infeasible in the time allotted to prototype. However, we did brainstorm that by designing the game to be in first person (rather than third), I could avoid the time required to model and rig such a dynamic object as a squirrel. Choosing this option also gives the player a stronger connection with the character than a third-person camera (in my opinion).

With the upgrading-resources idea out of the question, I needed to find a new “hook” for my game - a purpose. I had the food idea, but as Austin pointed out, there needs to be both a decision and a focus on squirrels. Having brainstormed this and discussed with other friends, I realized that an interesting attribute of the squirrel is that it collects food and stores it away. This improved the game concept to having a collection-and-delivery system, however the final implementation required a choice. Therefore, I emphasized that food would weigh the squirrel down, and he could then choose to sacrifice a bit of food (long term goal) for a speed increase to escape an enemy (short term goal). Adding the students spawning food and the dogs patrolling the area just added nice aesthetics/plot.

My next dilemma was whether or not to add a timer. A timer is nice because it adds an extra degree of difficulty for the player (has to now avoid dogs *and* collect enough food in time), and is clearly displayed. However, I thought it was a much more creative, artsy, and juicy idea to make the trees become the timers. Instead of an ugly countdown in the middle of the screen, the player can look at the hue of the leaves to gauge time remaining, and dismiss the colors subconsciously as environment at other times.

A feature that I kept for the majority of the prototyping process was a leaping component that was built into the “move forward” controls for the squirrel. I liked this aspect because, being a first person game, a hopping effect would emphasize the movement of the squirrel. When I had my friends playtest my game, this was probably the most disliked feature, as it made it hard to control (and they said it felt more like a bunny’s hop). I attempted to make the jumping movement more fluid, however I kept having issues bumping colliders and spinning in strange directions. Regretfully, I needed to remove this attribute right before the build as I felt it could cause the player motion sickness.

A new mechanic that I really enjoyed learning and messing with was iTween. iTween is a helper class/script that allows for easy AI pathing design. Instead of having to manually go into the code and enumerate all the points an AI (student, dog, etc) needs to visit, iTween allows you to drag and drop game objects into the scene with a path lerped through them all. This was a little tricky to set up at first. However, since I was instantiating all my students and dogs, all I had to do was assign them to various iTween paths, and the rest was taken care of. There were two significant downsides of this, however. Firstly, when the player came into view of the dog and it had to leave the path in order to chase them (and then possible return back to the path), pausing the iTween, storing the point the dog left the path, returning to that point, and resuming the lerp was tricky. Additionally, due to the combination of all instantiations of dogs using one of three paths, and the same with three other paths for students, deleting instantiations that were on iTween paths threw some pretty nasty errors. After lots of debugging, I was able to edit the source iTween code I downloaded in order for it to work correctly in my game. I’m still not quite sure which method of AI pathing design I like better, but for more complicated paths I have a feeling iTween’ing will work well in the future.

The last area I wanted to tackle and improve in my game design was my aesthetics, GUI/UI, and juice. In the beginning I only added a few sound effects: when player got damaged, when collected food, etc. Already that little change helped the feel of the game. The next really big steps were adding music and various “screens”: menu, death, and win. Once these were implemented (as artistically as one could expect from a CS major with no art background), I really felt proud of where my game ended up.

Sure the enemies are boxes and perhaps the NPC helpers are capsules. Art can always be improved in post. However, this prototyping definitely let me explore various mechanics and improve the speed at which I design games. I think I only really realized near the end of the two weeks how much I had improved since when I first started using Unity. Being able to quickly implement RayCasting or iTweening now makes me feel like my horizons in game dev have broadened greatly. I look forward to pushing these boundaries even further in Project 3.