Senior Project Final Report

Michael Campfield

 For my senior project, I have created a first person shooter (referred to as ‘fps’ henceforth). Using Unity3d version 2.6, I have built a 5 level game, comprising of fictional factions: the Cube-u-nists and the Capsulists. These two countries have been at war before the game even started, with the player experiencing and witnessing the conclusion of the later parts of the war. This is done with a combination of game play & cut scene cinematics, which utilize a combination music, voice acting, physics, and coordinated “behind the scenes” tricks used to create an experience not seen often from students in general.

 This project had two main objectives: to demonstrate the knowledge and experience I have gained in the computer science degree program, and to take my passion for games to the next level. There will be some who will question the content of my game and the intent behind its creation. I must point out that I take the view that if violent but well thought out movies with an underlying sense of purpose and meaning behind the violence is allowed, why can’t violent video games be given a chance to have some underlying sense of reason? Why can’t video games take that same emotional response that movies convey, and make the player live in that moment? Witnessing an event is not the same as living through it. Wouldn’t a reenactment of the Normandy landing, if done correctly and accurately, show just how brave the men had to be to face something of that magnitude?

 The creation of my fps was done with an “agile” development model. At no point did I try to make all of the decisions all at once, nor did I ever bury myself with the details of the project, losing sight and time. Small snippets of code were written and tested at a time, not allowing it to be integrated until it passed all my tests, thus future problems. Almost code is broken up into its own file, and most code doesn’t exceed 10 lines per block, allowing for cleaning and easier to manage code to be integrated. Also utilized was the “2 of 3” decision making model. This model says you can practically have two of three attributes with regards to your project: time, features, or quality. Time describes how long it will take to complete the project. In my case time was non-negotiable, the deadline was fixed and there wasn’t much I could do about it, thus time was the first of my two attributes. Features are the number cool, interesting, or useful things you’re finished product will have. This was my second chosen attributes for two reasons: many CS students have to toil away at mountains of code just to get simple games working, and often times these “games” just qualify as games. They often serve the purpose more of getting a grade than to actually be played. Unity3d, the engine I’m working in, allows me to focus most of my energy into designing the game itself, giving me an advantage that most students don’t have when making games. This allows me to add a lot of cool little features that make the game feel more like a game than an actual project. I was able to focus more on what game play mechanics would be used in the game, as well giving an actual story for the player to actually get. The third attribute, quality was not chosen for a couple of simple reasons: I’m only a CS major. I deal with the under workings of software, not the visual aspects of media, so most things in my game look simple. Second, I know nobody is expecting the greatest from me with regards to this project. I’m only one person working with a budget of zero dollars on an NMU issued laptop, using a free version of a 3D engine, in a 3 month time frame. So quality was sacrificed.

 While the development style was agile, production of the game has proceeded with a general pattern: Brain storming sessions formed a large portion of the game’s direction and plot. The decisions made during these brain storming sessions would become the basic blueprints that all other phases of construction would refer to, and must follow. “Shells” of the game levels were next laid out. This involved just putting basic models out, and laying down underlying textures used for visual effect. Next came the “major construction” phase. Building off of the shell in each level, I fleshed out the details of each level, setting the mode, adding more visual detail, invisible walls, and rough cinematics at the end of this phase. The final stage of production was adding the finest of details, things like weapons, ammo pickups, health, finishing the cinematics, and placing enemies, music ques. At any given point however, production would take a detour to work on the game mechanics, and testing certain features. How would I change cameras? Add music? What about the destructive environments I promised? Combat system? These were things I took time to figure out, leaving whatever level I was working on at the time to sit where it was.

 While Unity3d did a lot of the grunt work for me, I by no means could afford to slack off. The biggest challenge I faced was simply figuring out solutions to non-obvious problems. Certain features such as my damage indicator, which displays blood on the screen after a certain amount of health is lost, ran into strange difficulties that were not explained in the user’s manual at the website. This particular problem was when I attached a GUITexture component to the camera; the texture that was supposed to be displayed over the camera wouldn’t appear at times when I moved. For reasons still unexplained, the texture, which could be moved up and down within the editor or in code, was moving at the same time the player moved, regardless of the fact that I never wrote code to do such a thing, and this behavior didn’t match the normal behavior of Unity. It took some time to figure out, but eventually, I was able to replicate the problem, and find a solution by detaching the GUITexture component off of the camera. Little things like this occurred frequently during development, and while most of weren’t consistent with normal Unity behavior, I still was able to find solutions to these problems.

 While there were many technical issues and details to work out, another large aspect was the design of the game itself. The plot, characters, voice over work, storytelling elements, and cinematic elements all had to be placed. Throughout the course of the project, questions abound regarding a lot of different game play and storytelling came to mine. Whenever the ‘refined’ construction part of the level building phase began, I constantly asked myself how I envisioned this before I started building this level during the brainstorming sessions. Using these blueprints, I already had a layout decided for me. All I needed to do was remember what decisions were made, and follow them accordingly. However, these brainstorming sessions told the basics of how the plot would play out. The finer aspects of the presentation were decided not from a development stand point, but from a movie director’s view. What could be done relatively easily, but still have positive impact on a player’s experience. Fading in and out, multiple cameras being used, speech and music ques, triggered explosions and many other cinematic elements were painstakingly decided and placed to give the player an actual game experience, not just another project to be worked on and forgotten.

 Game play mechanics were another aspect that had many hours put into it to ensure that they were coherent, fluid, and felt okay. One of the mechanics used was a non-traditional health bar. This is a marriage of the classic health bar that was a staple in classic fps games, and a key aspect of the contemporary “regenerative” health system. In contemporary fps games today you can acquire a certain amount of damage, and as this damage builds, the screen is filled (often with blood) with images that obscure the screen, making it more difficult to see where you are going. The main argument behind this that although you have regenerative health, until you stop behind cover and allow your health to recover, you will have a very difficult time seeing just what is happening. Furthermore, this tells the player the state of his avatar without having to keep looking at some health bar somewhere away from the action. The screen filling mechanic was taken from that system, and put together with the classic health bar system, which had an arbitrary number of “health points” that the player had to keep at least one point in order to keep playing. Instead of a health bar, what happens is after 33 and 66 health has been lost out of a total of 100,blood will fill the screen obscuring the players view, but initially not that badly (during the 66-33 health phase). Once the player has less than 33 health left, more blood will fill the screen, blood will begin dipping front of the player’s eyes, and a heartbeat will begin sounding in the background, signaling to the player that death will ensue if a health pack is not found soon. Once a health pack is found, health will be brought back to 100, clearing the screen of blood, and stopping the heartbeat.

 Another game play mechanic that needed to be decided was how many weapons the player will carry, and is their infinite ammo? What kind of weapons would be used? Unfortunately, I took the path of least resistance and choose to keep the weapons realistic, and stick to a two weapon system. This two weapon system needed to allow no more than two weapons to be used at a time, forcing the player to make a decision when it comes time to get a new weapon. The only interesting thing about my weapons is my rocket launcher, which will be used to destroy buildings in the game.

 Learning new things that I had never been taught to do was a huge part of this project as well. Prior to the project, I had babbled in making 3d models in a free 3d model editor named Anim8tor, will some interesting results. I even used it to create a virtual model that was used in another project for Argonne. Once the project begin however, I needed to really figure out just how models were manipulated and rendered in the engine to reduce if not eliminate tearing and “disconnected” parts of models from not being manipulated as they should be. An example is saving a 10 piece model as one solid piece then importing it into Unity allows you to simply flip a single Boolean render variable versus saving it as a group of 10 pieces which would require you to flip the render Boolean in each and every part in code. Doing this would require predefining each component, then flipping them all individually. Small details like this came up often, and there was no end all solution for all of them. Often times experimenting was required, and at times new insights were gained on the software itself and new techniques had to be developed to overcome certain problems. Constant looks ups at Unity’s JavaScript reference pages were required, often times boiling down to just looking and down a list methods and class in the library just to see if I can find something that looks remotely like something I can use to solve the problem I was working on. Furthermore, this project took me into directions that are outside of computer science, such as directing voice actors to vocalize emotions, mixing and preparing sound and music in the game, and coming up with creative ways of progressing the story.

 This project to me was more than just an opportunity to gain credits. Being a gamer, just making a game with no passion is tantamount to sacrilege. I have a unique opportunity to make a gun the way I would make it. Despite the large limitations before me, the game from the beginning was designed with hours of careful planning and thought. Often times, ideas and mechanics couldn’t be made at all, or at the quality that I had aspired and planned on having. Often times the phrase that played in my head was “I would love to do this, but that just isn’t an option at this point”.

 Overall, this project has been very fun and very frustrating. The time and energy put into the game was immense, but the finished product will be great. A demo will probably be ready by the April 29 2011 presentation.