Project Angelis

Virtual Reality Wave Survival First Person Shooter

The Rifters
Fernando Colon
James “Isaac” Ebbert
Samuel Jonesi
Noah Leiter
Julia Olsen
Kurt Steinke

University of Missouri
College of Engineering
Capstone I
December 16, 2014

Table of Contents

I. Executive Summary

Project Angelis is a first person shooter video game we will develop for the Windows operating system. The game is made for use with the Oculus Rift, the Sixsense STEM System, and the Virtuix Omni. The development group consists of Fernando Colon, lead tester and Oculus implementation lead, James "Isaac" Ebbert, lead animator and combat architect, Sam Jonesi, scrum master and lead modeler, Noah Leiter, sound designer and economy designer, Julia Olsen, level designer and art lead, and Kurt Steinke, team lead, AI designer, and STEM implementation lead. We will be using Unreal Engine 4 for development, with Git as our version control. Components will be tested and integrated in the University of Missouri's Virtual Reality lab where the Oculus Rift will be available to the group.

In addition to being a first-person shooter, the game archetype is that of wave survival from video games such as *Call of Duty's* Nazi Zombies mode or horde mode from *Gears of War*. The objective of the Project Angelis game is to survive against waves of intelligent AI that are trying to "kill" the player and his AI allies. The game ends when the human player's health reaches zero. After every wave the player gets a chance to buy weapons, ammo, and friendly reinforcements to prepare for the next wave. There are a total of 50 waves that increase in difficulty as you progress from one to fifty. Once the player beats level 50 the game ends. The player will be using a wide array of futuristic weapons to combat enemies and will require advanced strategy to beat the game at it's highest level.

The game takes place in the near-future on a space station in the Alpha Centauri system. The ship consists of a single circular level, with optional objectives spread throughout. We are looking to create a fast-paced, tension-filled environment that immerses the player within the world of the game.
In order to accommodate for the Oculus Rift, two main measures will be taken. These measures will be designed to maximize playability and minimize nausea associated with the Oculus Rift. One of these will be making sure the frame rate never drops below 60fps. The other accommodation will be implementing menus and information about the game state in a way that that is organic to the Oculus Rift. An example of this is the pause menu. The menu will slowly saturate and desaturate the screen to make the experience as comfortable as possible for the user. Another example would be the player's health bar. Traditionally, the bar would be displayed in the corner of the HUD, however, this causes disorientation for Oculus users. We will be implementing the heath bar as a graphical interface on the avatar's arm that the player can rotate to view. By using these unconventional means of UI design, we are hoping to promote a user-friendly experience for those using the Oculus Rift.

II. Acronyms and Terminology

- AI - Artificial Intelligence
- API - application programming interface, libraries in C++ are APIs
- Avatar - the player's character in game
- C++ - An object oriented programming language implemented on a wide variety of hardware and operating system platforms
- Epic Games - creator/developer of UE4
- FPS - First-Person Shooter
- fps - frames per second
- git - file-saving utility that improves group file editing
- GUI - Graphical User Interface
- HMD - Head Mounted Display
- HUD - Heads Up Display, a user interface archetype typically seen in first person shooters
- IDE - Integrated Development Environment
- Mind Map - a diagram that provides an abstraction of entities. Using this, we can provide a high-level perspective and see the relationships between various entities.
- NPC - non-playable characters
- Oculus - Company that created and owns the rights to the virtual reality piece of hardware known as "The Rift"
- Omni - Dynamic Trackpad used with Oculus for movement, etc.
- OOP - Object Oriented Programming
- QWERTY - keyboard setup involving the first 6 letters in top left being 'qwerty'
- Rift - Virtual Reality head mounted display that is a visual user interface output device for a user. Created by the company Oculus
- SDK - Software Development kit
- Sixense - Creator of STEM system
- STEM - Motion Tracking system used for virtual reality hand user interface
- Unreal Engine 4 (UE4) - game engine developed by Epic Games, first showcased in the 1998 first-person shooter game *Unreal*.
- Virtuix - Creator of Omni peripheral
- Visual Studio 13 - Microsoft Windows 7/8 IDE
- WASD - typical keyboard control setup for FPS genre computer games for movement (w=up, s=down, a=left, d=right)

III. Project Overview

Recent advancements in virtual reality hardware for the everyday consumer has propelled the gaming industry into a new state of player immersion never before seen at its current scope. While advancements in consumer hardware technology such as the Oculus Rift, Virtuix Omni, and Sixsense STEM have given video game developers the ability to immerse their players in games in new ways, very few game designers and developers have fully utilized these technologies to create games built from the ground up with virtual reality in mind. Most games developers have simply added support for virtual reality hardware input as an afterthought.

For our senior capstone project, our team, The Rifters, will be creating a first person perspective combat game, where players will fight waves of enemies in increasing difficulty. The game will be designed with virtual reality hardware implementation as a priority and will feature integration with the Oculus Rift, and Sixsense STEM system, and will support the Virtuix Omni treadmill.

A. Team Member Responsibilities

Fernando Colon: *Lead Tester, Oculus Implementation Lead*
James “Isaac” Ebbert: *Lead Animator, Combat Architect*
Samuel Jonesi: *Scrum Master and Lead Modeler*
Noah Leiter: *Sound Designer and Economy Designer*
Julia Olsen: *Level Designer and Lead Art Designer*
Kurt Steinke: *Team Lead, AI Designer, STEM Implementation Lead*

B. Project Mentor & Coach

Dr. Fang Wang: *Assistant Teaching Professor at the University of Missouri, Columbia*

IV. Research

Out of the three peripherals we are using, the Oculus Rift is the one that has by far the most development. That being said, it is still underused and has much room for improvement. When the Rift is added onto a game, its ability to integrate with the game's graphics often has as many bad points as it does good.

An example of this phenomenon in action is when *Skyrim* was retrofitted to support the Oculus Rift. The main gameplay actions such as jumping, walking, and melee combat seem to work fine with the Oculus. However, navigating menus is nigh impossible. This is an especially big problem in *Skyrim* where much of the time spent in gameplay comes from navigating menus for inventory and spells. Here is a quote from Penny Arcade Report's Ben Kuchera that demonstrates this:

> The Rift does not do well with menus, in-game text, or any user interfaces that aren't purely graphical. It's a major shortcoming of the hardware, and it makes games like *Skyrim* that throw many menus of that kind at you intolerable to play in a serious way. You'd have to remove the headset every time you need to read anything, much less compare weapons or assign skills.

There is certainly much interest in creating video games that will work well with the Oculus Rift. The best example of this is the upcoming video game *Star Citizen*. This is a first-person shooter/mmorpg game set in space that is famously featuring the Oculus.

The scale of the interest of this game is the truly amazing part. *Star Citizen* is not being produced by a corporation, but is instead being crowdfunded and has set the *all-time crowdfunding record at $63 million*. It is said that in capitalism, people vote with their wallets. To have such a large group of people invest so much money in a product that has not even been made yet makes it clear that there is a market for a more interactive game experience that simple controllers, televisions, and personal computers cannot give.

One of the members of this group has a personal experience that seems to support this idea. He attended a conference as part of his internship at the Cerner Corporation and the keynote speaker's speech was on the idea that consumer use of hardware is the next big revolution in technology. The new interest in peripherals seems to suggest that video games will be one of the biggest industries affected by this.

This same member used the Oculus Rift before the project began. He was at the Ohio Supercomputer Center for unrelated business. The Ohio Supercomputer Center at the time was working on using the Oculus Rift with 3D graphics to create a training simulator for medical workers.
To create a video game using the Oculus Rift and other peripherals that are even less explored can actually be quite useful for non video game applications such as simulators for training. In computer science, games are often used to create a methodology for solving a simpler problem before expanding this methodology to suit the real world. Video games can be used for this purpose as they combine the problems of real-time programming and 3D graphics. Add the Oculus Rift and we add another element of realism that could make simulators even more useful.

V. Requirements

A. Functional

- Fully integrate the Oculus Rift
  - Fit lens for screen
  - Develop models that do not leave screen
  - Head/screen move simultaneously
- Fully integrate the STEM system
- Supports Virtuix Omni hardware input
- Support Gamepad Controller
- Must have a menu
- Feature one gamemode
- Be a first-person shooter
- Support keyboard input QWERTY (WASD movement)
- Will have one level
- Will have enemy AI
- Will have friendly AI
- Will be single player
- Include ambient noises
- Include weapon sounds
- Allow the player to purchase weapons
- Allow the player to interact with the environment
- Allow the player to switch weapons
- Allow the player to reload a weapon
- Keep track of the player's score
- Must be able to be paused
- Allow the user to change input (STEM/Keyboard/Gamepad)
- Playable on a Windows 7 machine

B. Non-Functional

- Frame rate must not drop below 70fps on optimal build
- Gamemode is wave-survival
- Have 50 waves
- No effects applied to player camera (rain)
  - Don't treat it like a camera (no cracks, wetness, etc.)
  - Reduce motion blur as much as possible
- Minimum hardware specification:

  - Windows 7
  - CPU 2.4 GHz
  - RAM: 4GB
  - DirectX 11
- Game must run on the Unreal Engine 4
- Minimize user VR disorientation
- Aim for ESRB Game Rating: Mature
- Purchasing weapons must be done using the player's points
- Two-story level
- Score increases when the player completes an optional objective
- Score increases when the player kills an enemy
- Have a variety of weapons
- Lights go off when the enemy AI damage a power core
- Blast doors remain closed longer when the player interacts with them

VI. Game Design

A. Core Mechanics

One of the very important parts of a game being immersive is the way the character interacts with the world. In our case, the primary interaction will be the use of weapons to defeat the invading enemies. Through the use of the STEM and Oculus systems the player will have complete control of the arms and head, and as such will be a very different experience than the standard first person shooter. In order to reflect this difference, we will need to design the weaponry in a way that feels natural for this type of control. The STEM allows for a controller to be in each hand, this means that the guns will need to be usable with one hand, and we have decided to allow the player to dual wield weapons.

Weapon Types

There will be two main types of weapons in this game, projectile weapons and melee weapons. All weapons will be powered by a universal clip system, meaning that all weapons will drop and use the same ammo, in our case it will be an energy clip system. The projectile weapons will start with the standard weapons that exist today retrofitted to fit the futuristic story. Then there will be high tech futuristic weapons, these will primarily shoot energy projectiles of varying sizes and styles. For simplicity and immersion we will be sticking to the more recognizable weapons such as pistols, small machine guns, shotguns, assault rifles and grenades. The melee weapons will consist of offensive and defensive weapons, such as an energy shield or energy sword. Below is a full list of the weapons we intend to include within the game.

1. Conventional: Futuristic No-special Properties
    - Pistol
    - Magnum
    - SMG
    - Shotgun
    - Assault Rifle
    - Grenades
2. High Tech: Futuristic & Special Properties
    - Pistol (Unique)
    - SMG (Unique)
    - Shotgun (Unique)
    - Assault Rifle (Unique)
    - Rail Gun
    - Grenade(Unique)
3. Unconventional weapons
    - Healing ray
    - Banana Gun

Ammo System

For our implementation we are going with a universal ammo system that runs on energy instead of individual clips and bullets. The player will be able to pick up ammo by simply walking over it inside of the game and it will be added to the players total stock in the same way that most games. Each weapon will then use up a specific amount of this energy as displayed on Combat HUD described below. This will allow for easy balancing between weapons by being able to tweak the consumption of energy for a weapon based on its firing attributes.

Combat HUD

While in combat it is important to display important information such as health and ammo to the player. Since the Oculus Rift does not do well with HUD elements that stay on the player camera, we will be putting this information on holographic projections. The health and other information about the game progress will be displayed on the players arm and will be visible in the same way that a person looks at a wrist watch (See *Figure 1*).

The ammo information will then be projected just behind the weapon that the player is using, so that the player can always see how much ammo they have left without the standard HUD style display (See *Figure 2*).

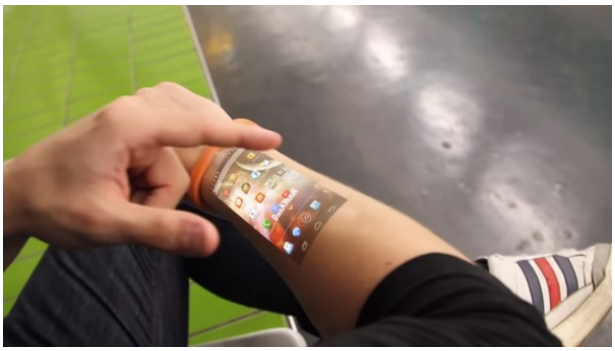


Figure 2: http://www.kitguru.net/laptops/mobile/android/matthew-wilson/circlet-projects-android-on-to-your-wrist/

Figure 1: http://www.egmnow.com/platforms/xbox_360/dead-space-3-weapon-crafting-gameplay/

Dual Wielding

Because the weapons in our game will all be one handed and most of the time the player will have a some kind of weapon in each hand, significant changes need to be made to how the player interacts with the environment and their inventory. Our plan for switching weapons is to have spots located on the characters body that when the hands enter trigger events such as reloading and switching weapons. The weapon switching will occur when the player puts the Stem controller in the area on the outside of the leg, much like the action of holstering a weapon. There will also be a spot most likely somewhere on the torso for reloading, since the weapons are working on energy it will act in much the same way as charging a battery by plugging it into a spot on the characters suit. After switching weapons to melee or putting a weapon away so that the hand is empty the player will be able to attack at close range. By calculating the speed of movement in the STEM controller it will be possible to set a

minimum for a value that turns the hand or the weapon in the hand into an attack. With this the player will be able to punch or move a melee weapon in a way that causes damage to enemies.

Environment Interaction

When it comes to the player interacting with the game, the STEM and Oculus combo makes the experience much more complex. There are two obvious interactions that we have found to be significant, picking up/holding an object, and touching an object to activate its properties. The STEM has many buttons able to handle these events. There is a button located on the pointer finger much like a trigger and multiple buttons on the thumb area next to the joystick. Our plan is to have the trigger button be the activate, this will cause actions such as a weapon to fire and a button that the hand is on to push. Then we can make one of the buttons for the thumb to be pick up/hold and the interaction with the environment should feel very natural to the player.

Here is a list of the environment interactions we plan to have available to the player.

- Interact with Shop
- Interact with Blast Door
- Interact with Power Generator
- Turn Off/On Lights

Economy System (Noah Leiter Research Portion)

For the playability portion of our project, it is important to look at 3 key characteristics: gameplay, achievement, and game economy. Beyond classic optional objectives, the most popular optional gameplay elements consistently center around achievements and game economy. Rick Lane from IGN rightfully claims that achievements "provide an illusory challenge which satisfies the desire for accomplishment (1)" ; for any new game or system, it is no longer a bonus, but has increasingly become a requirement. Vili Lehdonvirta, co-writer of *Virtual Economies: Design and Analysis*, states that the "joy of economics comes from being an economic actor (2)," rather than being an economist-most would not like to draw out economic formulas for their day job, but would love to see how they can influence an economy (particularly a virtual one with no real negative consequence). Optional objectives, some sense of achievement, and a working game economy will be important to ensure our game is highly playable.

For optional objectives, we are gathering information from several established horde-mode models: these include *Call of Duty: Black Ops* (copyright Activision) Nazi zombies, *Gears of War*'s (copyright Epic Games) horde mode, *Halo 3: ODST*'s (copyright Microsoft) Firefall, and *Team Fortress 2*'s (copyright Valve) Mann vs. Machine (for wiki's on each, see sources). After researching and compiling the extra objectives from these and a few others, the most popular and helpful optional objectives seem to be turrets, defensive placeables (like placeable cover), environmental challenges (such as light variations), and unique difficulty increases (such as firefight's skulls). The simplest way to implement the placeables would be to have them be part of the map, spawnable in certain areas, where they would be static for their duration. Once you purchase each (or, for the lights, when you hit the switch), the boolean for their existence in a certain part of the map would move to TRUE until their own HP or time expires. We could also have the turrets be spawnable wherever the user places them, but this is a (far) future goal. Here is the list of possible optional objectives, and their classification:

- user-placed (maybe) or map-available sentry guns ; static map element (dynamic if user-placed)
- infinite-ammo turret ; static map element

- cover wall ; static map element
- extra door ; static map element
- light switch ; static environment element
- difficulty meter ; interface element

Our design intention is not to achieve all of these, but some kind of optional objective will be important for playability. In the end, there may be more, fewer, or even completely different optional objectives, mostly depending on playtesting.

The second part of playability consists of player achievement. Again, we looked at the popular horde modes above- after compiling the popular achievements from these, we see most of them can be implemented by adding counters to actions (such as shots, deaths, kills, etc.). Pending any design problems, they should be easy to implement (we may end up having to discard complex achievements, if playtesting for them does not work out). Also, if they are not capable of being implemented, it will be easy to at least create a high-score board, using the currency acquired as an indicator of point value, giving players some sense that they accomplished something in the game. To see a more complex breakdown of the counters, see the Points and Economy Breakdown (Appendix H) section below.

Beyond objectives and achievements, we believe it is very key to the success of a horde mode to include some kind of economy. After each kill or wave, the player gets points, which are equivalent to currency, to trade in to get backup, ammo, guns, and other helpful items. For a crude chart of how percent increase will work after each wave, see figure 3 below ; for the enemy infantry bonuses, see 'things to include.' All of these are a compilation of the modes researched to provide a stable, fun, economy that is worth taking part in.

For playability, objectives, achievements, and a dynamic game economy will be the key aspects we will work on for the game. However, there are a couple extra design elements we may throw in eventually- these include, but are not limited to, easter eggs, weapon upgrades, armor upgrades, and 'fun' achievements (like punch a horse, or defeat an enemy by dropping a turret on him). The immediate goal, however, is to add some simple, but fun, elements to overall improve user experience.

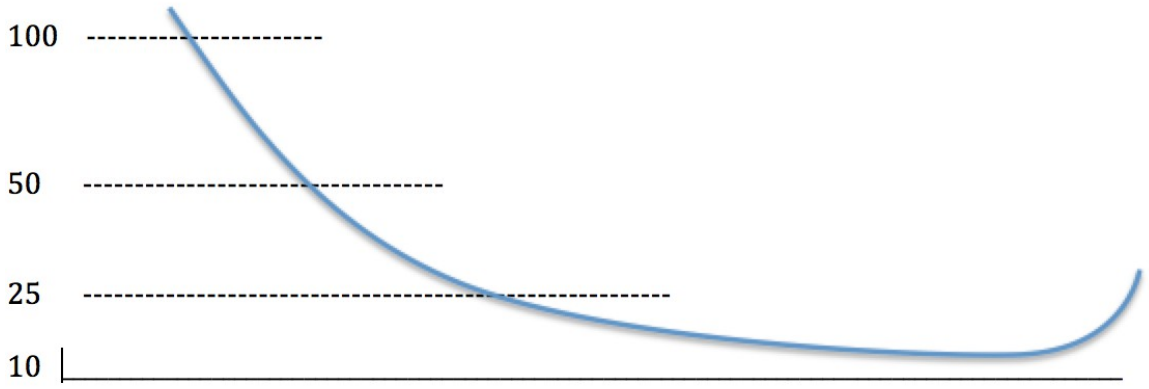


Figure 3: Wave Enemy Increase

B. Level Design

Flow

Level design is very important in all games as it sets the mood, pacing, and challenge of any given area. For a story-driven FPS you want the player to feel like they have choices in where they can go while still leading them through to an end goal. For a multiplayer based FPS you want the play to move in circles, never coming upon any dead ends with designated open spaces for combat. While Project Angelis is a single-player game that does contain minimal story elements, the environment will be much more circular. We want the player to freely move around the ship, constantly checking corridors and rooms for enemy units. While there will be specific spawn points that the enemies can come from, we want the player to feel like they need to keep watch on all corridors of the ship, not just the central room.

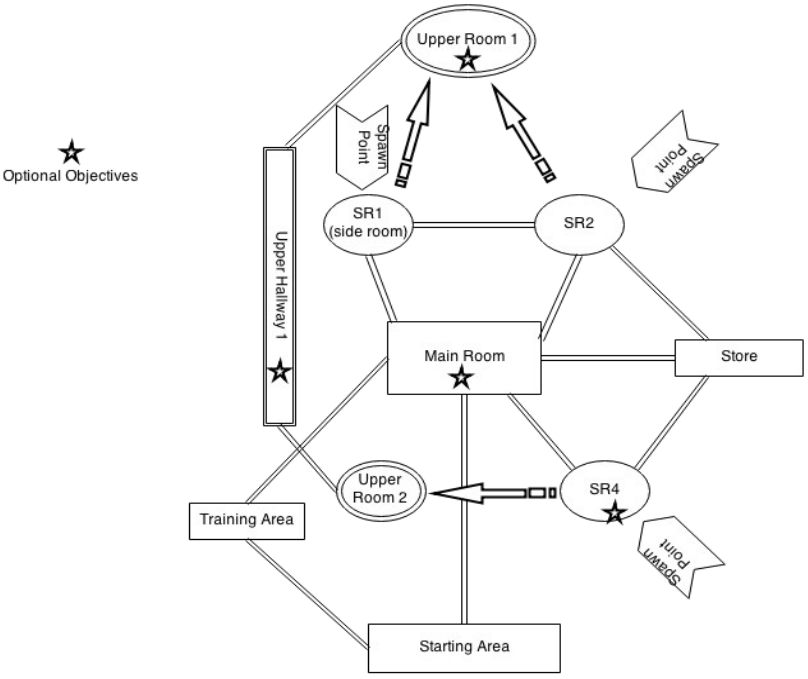


Figure 4: Level Flow

As previously mentioned, how the player moves through an environment is key to gameplay and termed *level flow* for game designers. The picture above (see *Figure 4*) diagrams the flow for our intended level. Each area has multiple entrances and exits to give the player choices, thereby empowering them. Empowering the player is very important in order to keep them playing the game. The player starts in the "Starting Area" that he can choose to exit by either going to a "Training Area" or "Main Room." Once in the "Main Room," either by way of the "Training Area" or not, the player is presented with plenty of options of where to go next. We want to make sure the player never feels trapped in any given area, or feels like he must turn around to get where he wants to go.

Environment Interaction

What the player can *do* within the level is the next area of design. It is recommended that level designers write a walkthrough of how the player will play through a given level. This can be found in the Appendix F. There are a couple objects that the player can interact with: the store, the black box, the terminals at the blast doors (optional objective), and the power cores (optional objective).

The store will consists of a couple oversized buttons (two arrow button and one purchase button) that the player can hit and a holographic projection of the different weapons. By hitting the arrow buttons the player can loop through the weapons until he find the desired one. The purchase button will create an instance of the gun on the counter for the player to pick-up. The black box will be similar to the store. The "box" will be a separate terminal with only a purchase button. When pressed, the player will receive a random weapon.

The optional objectives are objects the player can interact with to gain additional points. These also will affect some aspect of the world around them. The terminal are found by the blast doors and

can be interacted with for 10 seconds (time dependent on play-testing). Interacting with the terminal will prolong the enemies from entering through that door for a yet to be determined amount of time. The power cores are responsible for the lights within the ship. While the enemy AI's main goal is to kill the player, they can also damage the power cores to turn off the lights. The player can then interact with these cores (also for 10 seconds) to turn the lights back on and receive a score boost.

Physical Map

The physical map itself has a couple components that need to be defined: cover, heights, choke points, and focal points. Please note that terrain is also traditionally included in this list, but does not affect our level design since it is an indoor environment. Cover is the physical elements of the map that the player can hide behind, reducing the range that the player can fire at an enemy and vice versa. It is important that both the player and the enemy AI both have sufficient cover to balance the game. Within our map, cover will be implemented in a number of ways. First, hallways and corners can act as cover and create smaller areas for confrontation. Additionally, random crates and ship's parts (power cores, etc.) can act as cover. By placing objects strategically within the ship we can limit where the player can fire and be fired upon, creating a more interesting environment.

Heights add a lot of choices for players. By creating a second level with a window overlooking the main room we are allowing the player more options of how he wants to play the game. The player can take a sniper rifle up top to shoot the enemy as they come in, but must watch for assault from either end of the corridor. Additionally, the ramps leading up to the second floor can present the player with the height advantage over the enemy if he chooses to use it.

Choke points are parts of a map where the either the player or the enemies are forced to walk through a narrow space. This can be a hallway, door frame, crevasse, etc. For our map, there are a number of choke points the player can utilize to better his odds of winning. Each of our hallways can be considered to have a choke point at either end. If the player can bait the enemies into coming through one, then they have a better chance of picking them off one by one. However, with the circularity of our map, chances are the player will soon be attacked from behind. Additionally, if the player is trying to enter a new room from the hallway, then the enemy AI has the advantage since the player will be walking through a choke point. Each set of blast doors could also be considered a choke point and will need to be playtested to determined if this is too big of an advantage for the player (if it is, we can widen the doors and spread the enemy out so that it is no longer considered a choke point).

The last element of the physical map that needs to be defined is that of focal points. In order for the player to be able to orient themselves within our map and always know exactly where they are, level designers use focal points. Each level should have one main focal point followed by additional focal points for each successive area. The main focal point of our map will be the ship's power core in the main room. Each room will then have its own focal point. The focal point for the upstairs hallways will the be window overlooking the main room. The focal points for the first-floor rooms will be additional smaller power cores, work stations for the ship (circular tables, etc.), the storefront, and the blast doors.

Mental Map

A player's mental map consists of the level's pacing, tension, navigation, and challenge. Pacing is defined as the swing between chaos and calm. For our level, the chaos comes during the waves of enemies and the calm comes in the downtime between waves. It is necessary to give the player both of these extremes in order to create an enjoyable gameplay experience.

Tension is how much strain the game puts upon the player. We are looking to create a scaling intensity to our game that starts off less intense and increases as the waves progress. The basic intensity

of the games comes from the circular nature of our map. This circularity forces the player to continually check their back for approaching enemies. The scaling intensity comes from the enemies themselves. With each wave the enemies get stronger and are harder to defeat, creating more tension on the player. The last element that adds tension goes hand-in-hand with our optional objective of the power core. When the enemy AI destroy the power cores, the lights within the area are turned off, leaving only ambient light for the player to see by. Low lighting is a natural intensifier for the player as it limits their sight of the enemy.

Navigation is tightly linked with a level's focal points. Basically, it how easily the player can orient themselves within a given level. In order to help the player navigate through our level, in addition to focal points, we will use visual directions. Since the player will not have a long field of vision, we will have painted directions within the ship to help the player navigate. For instance, if the player is in the training room then one hallway would say "Command Center →" and the other would say "Barracks →."

The final aspect of the mental map is that of challenge. Challenge is defined as the goal of a given level. For our map, the goal is for the player to score as many points as possible. This is done by killing individual enemies, performing optional objectives, and beating waves.

C. Sound Design

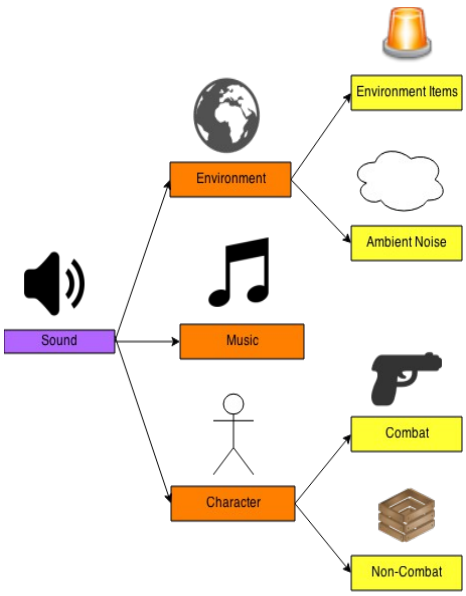


Figure 5: Sound Design

Sound design can greatly enhance a video game, but it must be consistent and subtle. For example, if a player reloads there gun and hears the "click" of the clip locking in place one time, then they should expect to hear that again the next time. However, the "click" should not overpower the other sounds within the game or replace them. Project Angelis will feature ambient noise in addition to a music soundtrack. The music will be royalty free beats that fit with the game's pace. Additionally, the game menu will feature it's own soundtrack.

Weapons sounds are the most pressing for us to find or engineer. Each gun will have it's own unique firing sound, in addition to a reloading sound. If a pistol sounds just like a shotgun then the game becomes unbelievable and player loses immersion. Thankfully, there are a lot of free sounds that have already been engineered available on the web. Our task will be to locate the ones that could realistically match the weapons we have, creating our own for ones we cannot find. This search includes sounds necessary for the melee weapons, fist hitting and sword swing, not just the firearms. Each time the trigger is pulled or swung, a weapon should create a sound just like it would in real life.

We want to create a very subtle ambient noise within the ship for the player to feel immersed. A low hum of the engine and some randomly placed distant "clanks" should do the trick. Basically, it is just important for the player to always be hearing *something* other than just the music so that they feel immersed in the world. The ambient noise will be the hardest to get right and should take the most tweaking so as to not annoy the player with repetition or a badly toned hum.

The more interesting sounds will come from the environment around the player. Lights should have a low buzz as the player nears them, alarms should be louder as the player gets closer to the source, and footsteps should be heard anytime the player moves. Additionally, if the player hits into an

object then there should be an appropriate collision noise, as well as when they pick up an item or repair an optional objective. The armor the player wears within the game will also have some sounds attached to it that gives the player a warning when their health is low. Again, most of these should be available with a relatively quick search on the internet and some attributions on our credits list.

The final aspect of sound design is that of NPC voices. In order to orient the user within our world and direct them to certain places, we want to use voices. These will need to be voice acted by one of us or someone else we find. We would like to random shouts coming from the friendly and enemy AI as well. These shouts would add verisimilitude and help immerse the player in the world we are creating.

D. AI Design

Project Angelis will feature a robust and complex AI system that will include both friendly and enemy AI. AI entities, hereby known as agents, will be split into three main types, light, medium, and heavy. Each of these types or "classes" correspond to a collection of attributes, behaviors, and modifiers that characterize and classify that AI agent. AI agents are organized into tactical squads that operate under one "squad leader." The AI that that governs these squads is a centralized system and is implemented in a Behavior Tree.

Enemy AI

Each wave in the game roughly 20 enemy AI agents spawn. These 20 enemies are a composition of light, medium, and heavy classes. At round one, the composition of these 20 enemies is easier and therefore is made up of mostly light class AI. As the wave number increases, the composition of these 20 enemies becomes made up of more and more difficult enemies. By level 10 almost all of the enemies will either be medium or large class. However, after the wave count passes a multiple of 10 and hits the the single digit 1 again, the composition is reset. So waves 11, 21, 31, 41. The composition of the wave will go back to like it was at level 1. However, after it is reset, static modifiers are added to the game that increase the difficult permanently. These modifiers include but are not limited to: increased enemy accuracy, health, damage, speed, health regeneration, intelligence, loadout luck, and ammo reserves. After every loop of 1-10, 11-20, 21-30, 31-40, 41-50 more and more modifiers are added.

Friendly AI

In the space station, the player is accompanied by a small force of allied soldiers. These soldiers can either be put in "sentinel" mode, "search and destroy" mode, or "follow" mode. In sentinel mode, the closest objective is guarded until the AI agent dies. There are two primary objectives that this could be: protecting the player or protecting a power generator. An allied AI will not leave a generator just because the player walks by. However the player has the ability to tell an allied AI to follow them at which point the ally AI will now guard the player. The player can at any point tell the Ally to stop following and will go back into sentinel mode. Lastly, a player can tell the allied soldier to "search and destroy" which will cause them to hunt and engage any remaining enemies. When there are more than three allied agents nearby they will form a temporary squad and will use squad like tactics similar to the enemy AI.

AI Categories

- Light:
    - Fastest moving, most agile
    - Primarily equipped with short range weapons requiring up close and aggressive movement tactics
    - Low base health, frail, typically weaker than the other types
    - Less likely to be designated squad leader
- Medium
    - Primarily equipped with medium range, semi-powerful weapons
    - Highly variable weapon loadouts
    - Moderate base health stats
    - Moderate base speed stats
    - Most likely to be designated squad leader
- Large
    - Slowest moving, least agile
    - Primarily equipped with high-power weapons
    - Often equipped with specialty weapons
    - High base health

Light class AI, are the fastest moving and most agile AI agents. To trade off with the increased mobility, light agents are frail and have low base health stats. They are least likely to be designated squad leader. They are primarily equipped with short range weapons that require them to be in close range. This causes their behavior to be very aggressive. They are most likely to perform flanking maneuvers than any other class because of their speed. Due to their shocktrooper like mentality, they have the least priority given to their self-preservation behavior tree.

Medium class AI, are the jack-of-all trades AI agents. Their base movement speed and health stats are balanced. Due to their balanced nature they have highly variable weapon loadouts, ranging from powerful weapons to weak weapons and from short range to long range. They are most likely to be designated squad leader.

Large class AI, are the slowest moving and least agile AI agents. Their base health stats are very high making them powerful juggernauts in battle. They are primarily equipped with powerful weapons but may sometimes be equipped with Melee weapons to balance out their health. Unless given a Melee weapon, their behavior is very reserved and shoots from a medium to long distance playing the role of grenadier or support.

E. Art Design

Note: Art design has been very challenging for us as a group as none of us are artists. As such, all of our concept art has been taken from Google searches from the internet. As you see, each picture has been cited accordingly as we want to give credit to those who do have artistic talent.

Character and Weapon Art

There is not a lot of designing we will be doing for the characters within our game as we plan on using pre-made characters for our actors. As such, they will probably be in modern military garb, like the soldier on the right (See *Figure 6*). If possible, we will try to keep the enemy uniforms dark in color so as to not draw too much attention to them.

The weapon art we will have a little more choice in as there are already pre-made sci-fi weapons available. The gun to the right (See *Figure 7*) shows the style we are going for. Each of the guns needs to be held single-handedly so as to allow for easy use with the STEM. Additionally, we plan on creating any weapons that we cannot find and/or altering the ones we do to fit our needs. These alterations could be making them one-handed, adding a spot for ammo amount to be displayed, etc. Since the guns will be seen the most clearly by the player, we plan on spending the time and money to make sure these assets are exactly how we want them to appear.

Figure 6: http://lordhayabusa357.deviantart.com/art/UNSC-Female-Trooper-396658984

Figure 7: http://digital-art-gallery.com/picture/gallery/pistol

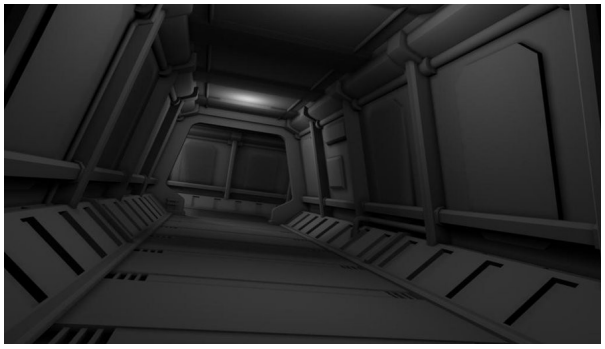


Figure 8: http://fc06.deviantart.net/fs71/i/2010/040/c/e/Spaceship_Interior_WIP_by_mhofever.jpg

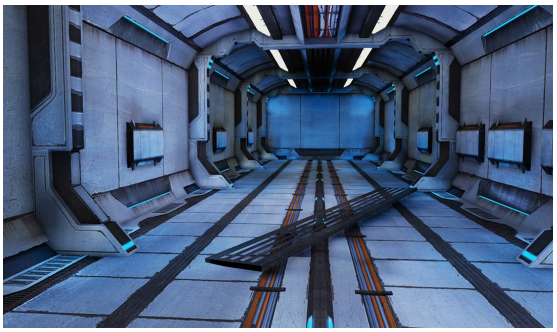


Figure 9: http://galleryhip.com/spaceship-interior.html

Level and Atmosphere Art

This first image (See *Figure 8*) gives a good representation of the basic interior structure we are looking for. We want the hallways to feel open while still giving the illusion of being within a space station. The hexagonal design adds physical space compared to a straight-walled corridor and immediately gives the viewer the impression of futuristic technology.

This second image (See *Figure 9*) represents better the amount of elements we want on the walls, ceiling, and floor. While the first picture gave the impression of being within space, this second image fills out the picture. The rectangular attachments to each wall section could have light emitting from them, and the beans going along the top of the ceiling adds to the illusion of a working ship. The same goes for the bottom design elements of the wires running through the floor. By adding elements such as these to our design we can present the player with a more functional ship that adds to their immersion within our created world.

Figure 10: http://www.daz3d.com/forums/viewthread/13848/

We want to keep the starting area very bare so the player is forced to exit without trying to do much exploring. By starting them in a bedroom aboard the ship, they get the feeling that they are waking up within the world just as their avatar is. Additionally, the bedroom allows for the user to recall their own life where they wake up and start the day by walking outside of the bedroom. This connection to real life will help orient the player within our world. The image on the right (See *Figure 10*) shows a basic bunk area within a space station with ladders going up and down to get to various areas. We would eliminate those and single present the user with a wide-open doorway for them to exit, giving them plenty of room to get used to the virtual reality experience.

Figure 11: http://starwars.wikia.com/wiki/File:Ebon_Hawk_interior_KotOR.jpg

The next picture (See *Figure* 11) comes from the Ebon Hawk, a smuggler ship within the Star Wars universe. This picture shows a good representation of what we want the command center to look like for the user, or the "Store" on the earlier flow design. The "Store" will not only consists of a place to purchase weapons, something similar to the circular table in the middle of the Ebon Hawk, but will also have the user's current score shown on the screens in the room. Since there can be HUD elements, this alternative gives the user access to their score without plastering it on the inside of their eyelids.

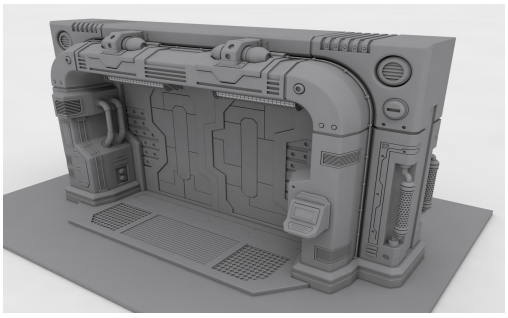


Figure 12: http://www.c4dcafe.com/ipb/topic/76418-free-video-tutorial-scifi-blast-door-part-1-the-outer-frame/

The following image (See *Figure 12*) presents the idea of our blast doors well. We want the doors to open in the middle and slide outwards, revealing the enemies behind them. Additionally, we want there to be some sort of computer system, like shown on the front right in the given image, in order for the player to jam the doors and prevent the enemies from entering for a given amount of time (another optional objective). While our doors may not be as detailed as the ones in the image, it gives us a good starting basis for our own design.

The next two images (See *Figures 13* and *14*) we found for inspiration deal with the lighting we want within our level. The first shows the color palette of reds, blues, and some green. The second is a better representation of the amount of lighting we want within the level. While we want the user to feel under pressure, under-lighting a level can be disastrous and no fun for the player to play through. We want the user to be able to see the world they entered, not stumble blindly through the dark. However, if enemy manage to turn off the lights and the player is given the option to repair them as them as an optional objective, then the first would represent the amount of light in addition to the color palette we are looking at.

Figure 13: http://galleryhip.com/spaceship-interior-background.html

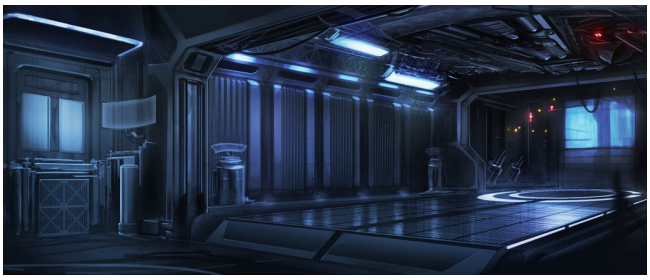


Figure 14: https://forums.robertsspaceindustries.com/discussion/115162/team-shard-collective-ax114-boomslang-mk4-new-renders/p17

This final image (See *Figure 15*) is something we are considering adding to our level to help orient the user. By placing a window, or windows, in the level we give the player a sense of the outside world around them and broaden their understanding of the world we've created. However, it is unclear whether or not seeing the outside would make the player more or less isolated. This design decisions is something we will have to discuss and test through once we have the rest of the ship in place.

Figure 15: http://galleryhip.com/future-spacecraft-interior.html

F. Story Design

Another strategy for giving a player full immersion into a game is to give them a story that is meaningful to the style of the artwork and the location. With that in mind, early on in our development we created the story below to work from. Once the story was in place, design decisions became easier because we knew the world we wanted to create.

Setting

The year is 2142, planet earth has hit a total population of over 17 billion. One hundred years ago an inhabitable planet was found within reach of human control in the Alpha Centauri system. Despite the great value of this planet, two warring governments are locked in a struggle for control of the planet. Naksas, who believe the planet should only be open for people of the upper class to escape the overcrowding of planet earth; and Simoursi, who believe the planet should be open to all people no matter what status. Both factions have already built space stations orbiting the new planet. Tensions are running high and it seems that a war could break out at any time.

Character

The player's avatar is a soldier of the Simoursi stationed on the ship orbiting the habitable planet. Having recently been promoted to Squad Leader, the player can call for reinforcements from patrolling vessels during the game (see Economy). At the beginning of the game, the player is woken from sleep by shouting soldiers and alarm bells - the Naksas are about to board. The player hears enemy and friendly AI making comments about the other side during the following battles, alluding to the history of the two warring factions on Earth. Through this, the player should be able to gather a rough history of the world we have created and the reason for the incoming assault.

G. UI Design

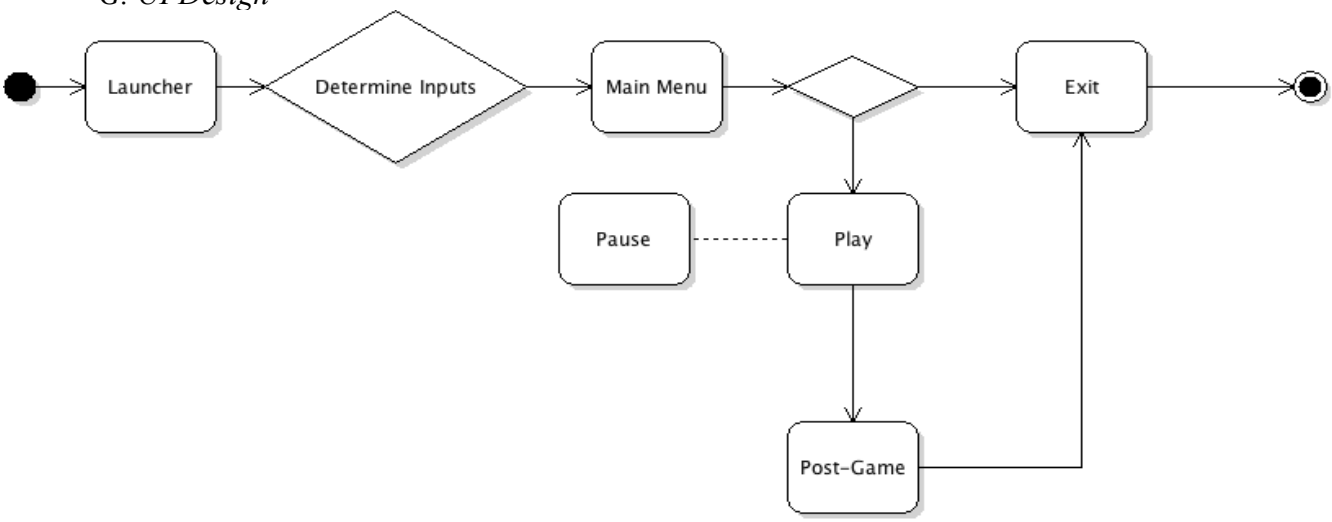


Figure 16: Menu Flow

Before we discuss the in-game UI, we have to discuss the game's launcher. Since we are supporting multiple input styles, the game's engine will need to know what configuration to use while it loads. This will be presented as a series of choices to the user that asks them what hardware they will be using to play the game. The game will then take this input as a parameter, and set up the control accordingly.

There are two components that go into the in-game UI design for a video game: the menus and the player's HUD. Traditionally, menus are complex and give the users a plethora of options and HUDs display the player's health, ammo, minimap, etc., on the corners of the screen. Since we will be developing Project Angelis with the Oculus Rift in mind, neither of these options will present a good user experience for the player. Instead, we will be using an unconventional approach to UI design.

The main menu of the game will be simplistic, consisting only of a play and exit button. The play button will load the game's level and start the user at wave one of the game. The exit button will close the game. We plan on using part of the game's level as the background to give the user a consistent look and feel to the game.

The pause menu will be mapped to a the player's controller. When pressed, the round will slowly desaturate and the player will no longer be able to move his avatar. However, he will still have control of his head. By keeping the rotational control in place, we hope to eliminate any sudden disorientation the player could feel from a sudden solid screen. There will be no options on the pause menu so as to limit this disorientation as well. When the player resumes, the game will slowly re-saturate and the game will continue.

Figure 17: http://www.dvice.com/archives/2009/10/laser_watch_pro.php

The player's HUD presents what is arguably the most important information within a game: the player's health, ammo, weapon info, etc. As described in the core mechanics section the best implementation for a Oculus Rift is a projection style display. The projections for the player information will be coming from a device attached to the players wrist. Similar to what is seen here (See *Figure 17*), only there will be projections going to the arm and to the back of the gun. Because of the role the Simoursi are taking the projection will have lighter colors for a more friendly tone.

VII. Testing/Data Collection and Analysis Plan

We are going to maintain a log for testing. Each log entry will consist of three parts. The first part of the log entry will be to say if we accomplished the specific goal we are testing (did we add this gameplay feature properly) or not. If an element has not been successfully added, we need to record our observations on what went wrong, and what the likely suspects for bugs are. We also need to record nausea levels using a simple 1-10 scale. If there is a specific reason for high levels of nausea, we need to record what that is. Finally, we need a miscellaneous section to record anything that could potentially go wrong in the future, as well as observations of what is working so we know which actions to do more of in the future.

A very important note about testing: Test results may override some decisions we have made in this report. The development of video games in general is heavily dependent on playtesting. As such, depending on how our testing progresses, we may need to change some of the design decisions that are in this document. The biggest reason why we need to change an element of the gameplay is if it is causing nausea. Any and all changes and their reasoning will be noted before the end of the semester. What we will turn in is guaranteed to be the scope of the project presented in this document and non-trivial. However, the dynamic nature of the user experience of video games dictates that we may need to change the way certain elements of the game are done.

VIII. Project Management Plan

To make this game properly, we will continuously be adding components we make to the functioning game as soon as they are made. To make the game any other way would be to set ourselves up for failure and an end-of-the-semester rush that would be worse than it will already be.

The way we are constantly testing our additions along with possibly changing our requirements make this project a textbook case for agile development methodologies. We are not going to be adhering to any specific agile/scrum methodology other than test-oriented. We will have one-week long scrums with a one-hour scrum meeting every week to discuss the goals we did and did not accomplish, what tasks we need to put in the back-log, and what the goals of the next scrum are.

Each scrum-specific meeting will mark both the beginning and the end of a scrum. We will most likely have our meeting be on Monday, that way we can devote the rest of the work week towards the scrum. Meanwhile, weekends will be used to finish up the scrum when we need them to. We can roughly divide each week into "preparation," "creation," and "integration," phases. Preparation would be the scrum meeting where we decide what to accomplish for the week. "Creation" would be when we work either alone or in pairs on our specific assignment. "Integration" will be when we get our specific assignments both working and tested on the master copy in the VR lab. Integration will most likely be done sometime in the weekend. It should have a report generated that we can then review that Monday and use to guide the following scrum.

To organize the project, we are using the free management software Trello. This will be useful in order to maintain a scrum board along with a backlog. Trello in general seems to be more user friendly than github's management functions, which will be vital to help us save man-hours. Trello has many functions that will come in handy. Boards can be used to separate our project into components. Once that has been accomplished, we can add lists in order to create tickets for our scrum. We can also add members through Google Plus, which is convenient as we are already using Google Plus for much of our communication anyway. We can also view the activity of others which will make double-checking each other's work, along with reviewing the scrums much easier. Finally, notifications can be used as needed to remind members of meetings.

The first part of our project is going to be to create a specific plan on how we intend to accomplish our goals. We already know what we want to do, now the question is how we are going to do it. Each member of the group will need to research their specific area and come up with a list of tasks. After that, the group can get together and discuss which tasks need to be done in what order. This allows for each member to decide what they want to do, but also allows a group consensus.

Another one of our early tasks need to be the creation of what video game designers refer to as "the gym." This is a room that is divided from the levels that the player is expected to enter and is made for the sole purposes of testing game and character mechanics out. Most of our time is going to be spent testing this game out, so it is vitally important that creation of this room is one of the first tasks we perform so we can begin testing on our characters.

We can divide our project into four main categories: core mechanics, level design, art design, sound design. We will most likely be working on the former three simultaneously, with certain group members loosely assigned to certain tasks, but free to float between tasks as needed. This will give us a good balance of flexibility to accomplish goals, but with enough stability that each person can become a subject matter expert.

Tentative Schedule

Below is our tentative outline for how the scrums will be divided.

Week 1	Setting up the project and starting the gym
Week 2	Asset Prototype requisition and additions to the gym
Week 3	Character completely functional
Week 4	Game physics finished (projectile, etc.)
Week 5	Level needs to be completely functional, any changes after this point superficial
Week 6	Finish work on game economy
Week 7	Finish game mode development (a user can sit through 50 levels of horde mode now)
Week 8	Assuming STEM and Omni are available by this time, integrate STEM with user controls
Week 9	Integrate Omni with user controls
Week 10	Finishing touches on art design
Week 11	Finish integration of AI; at this point, the game is the equivalent of the finished product on mute
Week 12	Begin planning/work for sound design
Week 13	Finish sound design
Week 14	Finish any and all “rough patches” for presentation
Week 15	Finish preparation for presentation

IX. Development Costs and Resource Needs

Item	Cost Per Unit	Funded By	Have/Ordered
Oculus Rift	$350.00	School & Team	Yes
Sixense STEM	$300-$600	School	Yes
Virtuix Omni	$500	School	Yes
Gamepad Controller	$15	Team	No
Unreal Engine 4	Free	N/A	Yes
Assets	Vary	School & Team	No

Figure 18: Cost Breakdown

Project Angelis requires a few costly items to achieve the full virtual reality experience we want to create. The principal hardware needed is the Oculus Rift that costs $350.00 for Development Kit 2 and Sixense's STEM that costs between $300.00 and $580.00 depending on the kit you choose. As it stands, the University already has two Oculus Rifts in its possession and our team has purchased a third. This effectively eliminates any cost for the University coming from the Oculus. The STEM system, however, did need to be purchased and was ordered for $580.00.

Additional hardware costs include that of the Virtuix Omni, costing approximately $500.00 per unit not including additional shoes. However, we have been informed that the school is already awaiting a shipment of two of these systems early next Spring. While this still cost the school money, it was not directly influenced by our project and happened before we pitched our idea. As such, the total hardware costs from the school total $580.00.

We predict the software costs for our project to be split up between the school and our team. Originally, each team member was paying for their own subscription to Unreal Engine 4, coming in at a cost of $19 a month. However, due to a new student promotion with GitHub, we now have access to UE4 for free for the next year, removing that cost from our team. The other aspect of software that will come into play for our project is that of assets. The Unreal Store has a number of assets that would greatly reduce our production time for the project. Some of these we may be asking the school to purchase, and some will come out of our own pockets. We estimate no more than $200.00 would be spent by the school on assets from the Unreal Store.

A final note that should be mentioned here is that of assets available for academic use that would not be allowed in a commercial product. While our future wish for the game is to be available for purchase, we have decided that for the scope of this project we will make use of all free assets available to students with the understanding that these would need to be removed or replaced later. By doing this, we reduce the overall cost for both the University and our team.

X. Identification of required independent learning

Below is a list of the required independent learning areas we will need to master as a group. Some of these are more complicated than others, but all are necessary to complete the game. While this list may seem daunting, we feel that we are up to the challenge.

- Implementation of Oculus Rift
- Implementation of STEM, STEM SDK
- Implementation of Omni
- Unreal Engine 4
- Artificial Intelligence
- Level Design
- Hit Detection/Game Mechanic paradigms

The first three requirements are for hardware, namely, the Oculus Rift, STEM, and Omni. We already have a head start on the Rift as we have set up the VR development room and have successfully gotten the Rift to work with UE4. The STEM will require more time to map the character rig to the hand movements of the player. The buttons on the STEM will be an easy map of the keyboard buttons that will already be in place. The same goes for the Omni. Assuming that it arrives in time, the Omni should be a straight map to WASD.

The last four areas of independent learning are focused on the game design and implementation. Some areas, such as AI and Level Design, we have already started, and finished, researching. This will make their implementation much smoother next semester when it comes to coding the game. The real challenges will come with UE4 and game mechanics like hit detection. Since we will coding the game for keyboard and mouse play first (while always keeping in mind the Oculus, etc.), there are lots of good resources available on these topics. We plan on using these resources and skipping the step where we reinvent the wheel so to speak. UE4 is just another coding environment to be learned and mastered.

XI. Project Outcomes

A first person shooter that effectively integrates the Oculus Rift, the STEM System, and Omni into a cohesive game experience. While there is already some work on using these peripherals, there is very little that integrates them. There are also few video games made that utilize the STEM and the Omni in particular. Most of what is available is mostly a demonstration. By creating a first person shooter than can utilize all three technologies, we are not only creating something that requires the novel use of both software and hardware, but also creating something that has not been made before and can be used for others to base their development off of.

XII. Future Work

Once we have fulfilled the above requirements and gameplay for Project Angelis, we want to focus on getting it market ready. There are a number of factors that would go into this, the first being to make sure we only have assets that have been fully paid for for commercial use. In other words, eliminating or replacing all assets that we found free for academic use. This could be a costly venture, but one in which we all eager to partake.

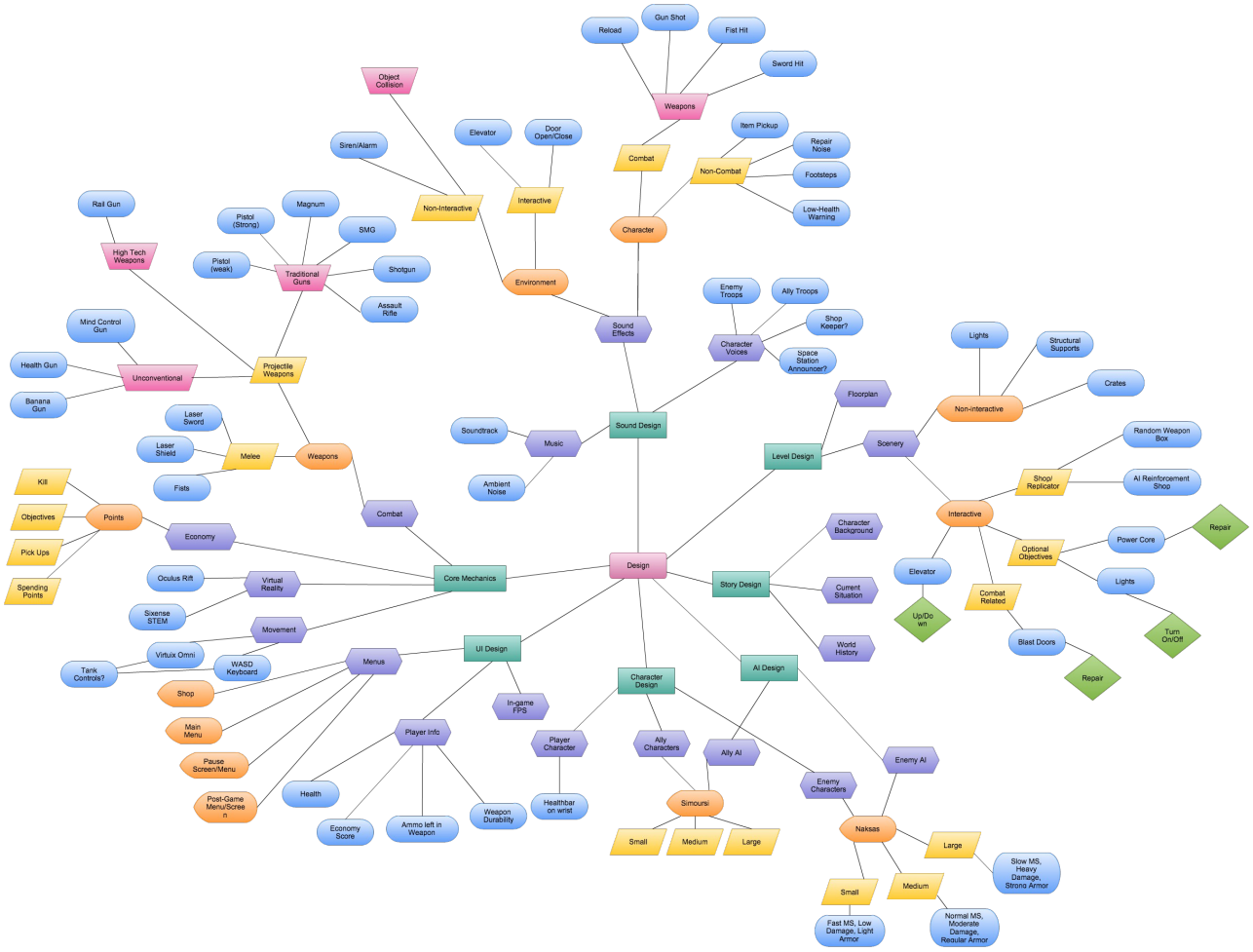
Another component of making the game market ready would be to include a terms a use agreement. This is something that none of us have any experience in and would require a lot of research and possibly a consultation with a lawyer. While we plan to eliminate as much virtual reality sickness as possible from our game, it is still possible individuals will experience disorientation and we want to legally protect ourselves from any repercussions.

Not only do we want to get the game market ready, but we also want to enhance the user's enjoyment of the game. To that end, we want to allow for a multiplayer mode so that user's can invite their friends to share in this new experience. Additional levels would also be created to add variety.

The benefit of creating a game like Project Angelis is that the expansion possibilities are endless. We can only come up with new objectives, optional or otherwise, power up abilities for the players, additional weapons, etc. Project Angelis represents the basis of a fully immersive virtual reality FPS that our team can take in any direction we choose.

XIII. Appendices

A. Mind Map



B. APIs, Tutorials, References

- *STEM API - https://stem.torproject.org/api.html*
- UE4 SDK - https://docs.unrealengine.com/latest/INT/Engine/index.html
- Oculus SDK - https://developer.oculus.com/
- Omni SDK - *coming soon*
- UE4 Tutorials - https://www.youtube.com/playlist?list=PLZlv_N0_O1gaCL2XjKluO7N2Pmmw9pvhE
- "The Dos and Don'ts of Designing VR Games" - http://www.gamasutra.com/view/news/220480/The_dos_and_donts_of_designing_VR_games.php
- *Game Coding, Complete Fourth Edition* by Mike McShaffry and David "Rez" Graham
- *Beginning Game Design* by John Fiel and Marc Scattergood
- "First Person Shooter C++ Tutorial" - https://wiki.unrealengine.com/First_Person_Shooter_C%2B%2B_Tutorial
- Game Dev Tutorials - http://gamedevelopment.tutsplus.com/articles/fantastic-gamedev-tutorials-from-across-the-web--gamedev-3384
- *Fundamentals of Game Design* by Ernest Adams

C. Team Member Biographies

Fernando Colon

I'm Fernando Colon. My technical expertise consists of most of the courses here at Mizzou. Some of the courses include Java, 2050, Database, Team Based Mobile App Dev, SWE and WWW. I'm in Science and Engineering of the WWW currently. I had a short internship for the Vet School as a PHP programmer. What I am looking for is a project that is looking to take advantage of the Oculus Rift and potentially the Virtuix Omni. Working in Virtual Reality is something I've been looking forward too and want to make a career within the VR community. A career in the video game industry is what I am aiming for so preferably a group that wants to make an FPS game using the Oculus Rift. I also want to learn how to develop using the Rift because an opportunity like this doesn't come often, so the fact that our department has it for this course is something I believe should be taken advantage of. I don't believe schools in the past few years actually use this in their capstone being that its barely 2 years old. So the experience gained from this could prove invaluable.

James "Isaac" Ebbert

Hi there to whoever is reading this, my name is James Isaac Ebbert, I go by Isaac. I have had considerable experience with computer hardware throughout high school and in getting my associates degree. At Mizzou I spent my first year in biological engineering and then two years in computer science, then this last May I switched my major to IT. As such I am currently in kind of a transition period, most of my experience has been in hard coding, but I have decided that I am much more passionate about the artistic side of IT/CS. My favorites are 3D modeling and animation (I have taken the first class and I am in the second class this semester for 3D modeling and Animation in Maya), and web design. I have practiced with Maya, Unreal, Unity, Cryengine3, and most of the Adobe programs, I am well versed in C, and know a good amount of Java, PHP, HTML, and SQL.

I would like to get into a project that is working on something in the game design or web design area. Preferably the game design as I am an avid gamer myself and hope to someday make games either professionally or just as a hobby in the indie developing platform.

Samuel Jonesi

Sam Jonesi originally entered Mizzou majoring in meteorology. After changing to computer science, he has been given many wonderful opportunities including researching for Mizzou's Virtualization, Multimedia, and Networking Lab, researching for Washington University in St. Louis, and interning at the Cerner Corporation. After graduating, he will be pursuing a full-time career with AT&T in St. Louis.

In addition to computer science, Sam has completed four years of ROTC training at the University of Missouri with Air Force ROTC detachment 440. Part of his training included 28 days of officer field training in Maxwell Air Force Base Alabama and Camp Shelby Joint Forces Training Center in Mississippi. He also was given an opportunity to escort one of the joint chiefs of staff at the Columbia Salute to Veterans Memorial Air Show.

On the rare occasions when he can get free time, Sam is found at the Mizzou Rec Center and is currently training to complete a marathon in White Sands Air Force Base New Mexico called the Bataan Death March.

Noah Leiter

My name is Noah Leiter, and I am hoping to earn my BS in Computer Science. The following is a description of my technical skills, experience, interests, and ideas. I admittedly have limited experience and programming skills. I did not have the fortune of having a computer or internet before coming to college, so I discovered my interest in the field of Computer Science after I had spent a year at Mizzou. I also have not had much time to spend working on my own projects outside of class since then. I can program very well in C, C++, Java, and PHP/HTML. I am very willing and very able, however, to learn any language/utility desired in order to be a useful member of any team. I am an incredibly quick learner, and am fascinated by all aspects of the field. It is also worth noting that I am an English minor, so I can proofread/peer-review almost anything, and I can write a marvelous essay at rocket-esque speed.

As I mentioned above, I am interested in all facets of the field of Computer Science. While I would do well in my niche with languages I've implemented with, I almost prefer to use what I know on a utility that I am unfamiliar with, or in a fashion with which I lack experience. I relish the challenge and experience. Regardless of the department of study, I am rapt with the possibilities presented through programming.

To close, here are a few of my ideas: Star Fox-like simulation/game for use with Oculus Rift technology, Asteroids remake (for a smaller group), a fully-functional cash register/inventory program, a campus marauder's map, and a quiz website (like a daily brain-buster/submit your own jeopardy-like questions app).

Julia Olsen

Hello, my name is Julia Olsen. I found the IT department at Mizzou in my sophomore year after taking a film production course and decided that the technical aspects of films appealed greatly to me. Since Mizzou currently has only a film studies program, I looked into the IT department and found Leslie and Chip's classes and went for it. As such, I had to take CS1040 – turns out, I liked programming just as much as I liked film. So now it is my senior year and I know more than I ever thought I would.

Since I didn't find IT till later in my college career I don't have a lot of the background experience that a lot of others have. However, I have worked hard to gain all the knowledge I can. I don't feel like I should go over my experiences in 2050, Database, Software Engineering, WWW, etc., as everyone in the program has had to take those classes. But I will say a little about my skills that I find important: I am a fast learner and I work hard. I do not accept poor grades, and will work all night if it means getting those extra five points. This is not to say that I just hunt grades, but it is important for me to figure out why something isn't working and not give up in order to be the best programmer I can be.

As I mentioned earlier, film is a huge passion of mine. I have worked on a number of local independent productions, mainly as the AD (assistant director). It take a lot of dedication and problem solving to be an AD for a set, and you can find yourself in ridiculous situations, but it has always been worth it to me. The sense of community that a set can have is amazing, and teamwork is so necessary. In addition to problem solving and teamwork, my time around film has taught me all about the visual experience and what makes up a good shot, how to get good sound versus bad, and made my best friend hate watching movies with me. Basically, I have spent some time cultivating a good eye for visuals.

Currently, I have been playing around with C++, C#, and Unity in my spare time (when I am not playing LoL or spending all my money on Steam). And obviously, I will now start learning Unreal for this project. Game design is something that I am very interested in for my future career.

Kurt Steinke

Hello senior capstone classmates! My name is Kurt Steinke and I'm an Information Technology major with a minor in Computer Science and Geology. Within the field of Information Technology my greatest interests lies in software development for mobile iOS, Windows based desktop applications,and the creation and modification of video games. The languages I'm most comfortable writing in are Objective-C, C++, PHP, and Java. I have experience in database and web development languages as well but I find I do not like them as much as the other languages/platforms I have listed. Over the summer I worked as a software engineering intern at an energy and utility company known as National Information Solutions Cooperative, where I worked in the research and development department.

Within the R & D department I specifically worked in the “Innovations” and new technology group. I was apart of a team that managed and developed a nation wide cloud-based smart-grid. One of the main projects I worked on while I was there was implementing a prototype application that could predict a person's electric bill based on trends in their past usage history and based on weather forecasts. Following my graduation, I have made plans to work at NISC. I really enjoy creating software which pushes the boundaries of the new technology and I've also always wanted to create a video game. That is why for my senior capstone project, I have teamed up with Julia Olsen to lead a team that will be working on a video game created from the ground up for new virtual reality technologies. These virtual reality hardware technologies include the Oculus Rift, Omni, and potentially the Sixsense STEM system. Our game will be a first person shooter most likely set in space although the game details have not been finalized. The game will be built upon the Unreal engine 4 for which utilizes a combination of C++, a scripting language, and C#. If are interested in working on the project with us feel free to email me at krsbqb@mail.missouri.edu. For more information about our project please visit the Project and Ideas Forum on blackboard and look for the “Oculus FPS PC Game” thread created by me, Kurt Steinke.

D. Meeting Logs

Previous: We had a lot of meetings before now, but starting taking notes as of 10/8/14.

Wed 10/8/14

Weapons will probably be easier to implement and make more sense in context if they are all futuristic weapons. The general idea is that all the weapons are "lazer" all weapons run on energy, say energy clips have X amount of energy I was thinking we could just show this as a % and each gun uses X amount of energy. Energy clips could be created in a tiered fashion, ie tier 1 has 25 energy and tier 2 has 80 energy and tier 3 has 300 energy, that way bigger guns can only be shot with tier 2 or 3 clip because it uses more energy than the tier 1 or tier 2 clip has, i think this would also allow for downward compatibility if we wanted to try that.

Isaac- I was thinking we could use holographic projection on the back of the gun to show the ammo and do something like go red when low and flash when out of ammo. We could also have a holographic projection come up on the arm when the stem user turns the control past something like 110 degrees like the movement of looking at a watch. display the total ammo clips for each of the tiers as well as the hp and other info.

Friday 10/10/14

Recap: Went through and created a game flow diagram. Talked about the merits of having a tutorial room and whether or not that should be a requirement for the player to enter the wave mode. Two schools of thought arose: 1) the player has to hit x number of targets before waves start spawning, and 2) the player much reach a certain physical space (proximity trigger) in order to start the waves (thereby not having to hit the targets unless desired).

Also discussed how the game should "end" and what that will look like as there is no traditional ending to endless waves. Options: 1) the player dies (no respawn allowed/they run out of lives), 2) they hit some sort of self-destruct button on the station/leave in an escape pod and "abandon" ship. Either way, there is a need for showing the player's score, whether through an animation-like event, or a "screen" surrounding my black that the user can look away from.

With these discussion the pause menu came up. We brought up the idea that any options that are traditionally on a pause or start menu will have to be on a computer (so setting sound volume, etc will not be controllable "in game", including a "start" button). If we choose to go that route, how will the game show that it is paused visually without disoriented the user (greyed out screen and only your head can move? completely frozen screen in the last position you looked? a minimized game screen that is greyed out and surrounded by black?). We did not come up with any conclusions, just played around with some ideas.

Wednesday 10/15/2014

Established and outlined what the weapon/ammo system will be.

Things we decided on meeting 2014/10/15

- Tiered Clips with differing usage
- Clip pickups(enemies drop clips)
- Weapon store for purchasing new weapons
- Weapon storage areas(places where weapons can be stored for later use without being deleted

Wednesday 10/22/14

We went over the ideas we had come up with for the research papers. Talked them through, decided who wanted to research what. After we split them up, we decided that some of the topics not chosen were ideas that should still be researched and added to the final report.
Additionally, we want back through the original list of requirements we made and categorized them better into functional versus nonfunctional. This also led to the addition of more requirements, which we added as well.

For Next Meeting: everyone should come up with an outline of their research papers so we can discuss the direction you want to take/see how much they overlap and change them if necessary. Topics chosen can be found under ResearchPapers/Topics on Drive.

Wednesday 10/29/14

Went over what the player can interact with within the world: weapons, store, power generators, blast doors, etc. We decided that most of the interactions will include a holding down a button on the stem and placing the controller towards your body, or else pushing (big) buttons in world. Rather than elevators, we decided that we will use ramps to get in between floor on the map. Lots of discussion around these topics and more along these lines.

Currently, we decided that waves will spawn at the press of the button - later we will playtest this to decide if we want a timer or some other function to spawn a new wave.
The store will be a series of buttons and hologram pictures of guns. The user will be able to unlock the guns in some ways (whether wave-based or blueprint drops, etc).
We also discussed the creation of our gym to test every aspect of the game.

Wednesday 11/7/14

We started out by talking about the economy of the game: what type of points should the player get, how many, etc. This lead to an in-depth discussion with lots of game design decisions made. We decided that the points should be called Fame/Infamy, the user gets a couple thousands on average per round, and spends these points to get weapons from either the store or black box. The player will get a different amount of points for different styles of kills/different types of enemies killed. There is a round bonus as well that scales with the current wave. The scaling is based on multiplication, with enemy speed, health, and amount changing with each wave. In addition, we gave a rough estimate of the price of weapons. Lots of progress.

Wednesday 11/12/14

Talked about what we want in a mentor and how we would present our ideas to one. Also talked about how often we want to meet next semester both with the mentor and not. In addition, we went over random Oculus thoughts. We concluded with a discussion about the final paper outline and the next steps we need to take to complete it.

For next time - outline for final design paper
- outline for research paper

Wednesday 11/19/14

How we will write the final Report:

Under each of the Game Design folders there is a “Paper Paragraphs” doc that has the outline for that section listed. What we decided to do was to write the individual sections under those docs and transfer all the text at the end. Anyone can work on any section, we are just trying to make sure we don’t have two people write the same stuff and waste time. If you want to work on a section that is not

under Game Design, there is a Misc folder under Report with a doc for each of the other topics. The VR specific topics are under VirtualRealityHardware.

For After Thanksgiving Break:

- try to have research papers finished, or at least a rough draft
-start working through whatever paper topics you want, adding well-written paragraphs
-everyone come up with ONE project name and we will vote on JUST those

Wednesday 12/3/14

This meeting consisted of some development room setup: getting logged into the work station, starting to download Unreal, getting the Oculus running, etc. We then spent some time discussing project logos and names. The meeting ended with going over the presentation and working on powerpoint slides.

E. Estimated Costs

Item	**Amount**	**Cost Per Item**	**Total Cost**	
Salary	x6	$50,000	$300,000	
Benefits	x6	$50,000	$300,000	
Office Space		$600/month	$7,200	
Computers	x6	$2,000	$12,000	
Unreal Engine	x6	$20/month per person	$1,440	
Utilities		$150/month	$1,800	
Internet		$100/month	$1,200	
Oculus	x6	$350	$2,100	
STEM	x6	$580	$3,480	
OMNI	x6	$500	$3,000	
Company Provided Food		$200/month	$2,400	
Office Equipment	x6	$400	$2,400	
Startup Costs			$5,000	
Assets	x6	$100/month per person	$7,200	
Misc.		$1000/month	$12,000	
Furniture			$10,000	
			$671,220	**Total**

F. Level Walkthrough

The player wakes up in a bedroom. There are sirens and alarms going off in the distance and a friendly NPC is yelling at them to "Get a move on, the enemy is coming!" The NPC continues to tell them to report to the command center as soon as possible, but be sure to be ready for a fight. If they are not ready, they can "Head to the training area, but be quick about it." The player is presented with an open doorway to walkthrough. Once out of the bedroom, they are in a hallway that splits off in two direction. Written on the wall is "Training Area →" and "Command Center ←" respectively (Note: the command center is the same thing as the Main Room on the Level Flow Diagram).

The player chooses to walk down to the training area. The hallway opens up into a wide room with a weapons rack. The weapons rack contains a starting pistol and melee weapon (a sword). To the player's left are humanoid dummies with targets painted on them. They player picks up one weapon at a time and practices his skills with both. They don't do much damage, but it is good practice. Alarms are still going off in the distance and the computerized voice of the ship's warning systems reminds them that. "Warning: Enemy Approaching. All units report to the Command Center." The player exits the training areas on the far end of the room (again, the hallway is marked "Command Center →").

Once down the resulting hallway, the player enters the command center. It is a large room with NPCs and an oversized computer screen on the closest wall that is showing the player's current score. In the center is a humming power core that goes up to the ceiling. There are six hallways leading off in different directions, including the one the player just walked through. The NPC commander greets the player and says the enemies ships are boarding in 60 seconds. "Hurry up and get to your post, soldier. We are about to be surrounded. If you don't have a weapon, there are a couple extra on the table." If the player skipped the training area, there are now extra starting weapons available on the table next to the commander.

The game now opens up and the player can go wherever they like. There are 3 docking bays where the enemies can board (marked with paint like the directions on the walls), an upstairs hallway which has a window overlooking the command center (good sniper position), and a shop where the player can purchase weapons based on his score. The goal is to survive as long as possible. The player heads to one of the docking bays as a countdown timer starts on the ship's intercom. The airlock doors have a health bar on top that is slowly decreasing as they enemy is hacking their way inside. The player notices a computer terminal by the doors and goes to look at it. "Hold down to put up a firewall." The player has found an optional objective. By holding down the computer terminal button, the player can add an extra firewall to the door's security, slowing the enemies approach. The player quickly does this, holding down the button for 10 seconds (Note: this time will need to be playtested).

The player decides to run to another blast door and perform the same operation. With the time almost out on the 60 second clock, the enemies break through the third blast door and the game is on. As the enemies rush into the main room the player starts to fire from the corridor, ducking behind a crate for cover. He then continues to fire at them, killing two, as he backs up into another hallway. By this time, the original set of blast doors is open and the player is forced to run past the second set of enemies and up into the upper hallway. He picks off two more shooting through the window before the enemies start firing from down the hallway. They get a few hits in, but the player dashes down the opposite side and tries to perform a loop to get behind them, firewalling the last door as he goes.

This process continues until all of the enemies are dead and the player is told that the first assault has been rebuffed, but another is approaching in 60 seconds. The player can now choose to firewall two terminals, go to the store to purchase a better weapon, or repair the smaller power cores to turn back on some lights (the other optional object that player ran into during the round).

The player chooses to purchase a new weapon with the points he earned during the wave, and heads up top to get the jump on the approaching enemies.

G. Basic Game Questions

1. Write a high concept statement: a few sentences that give a general flavor of the game. You can reference other games, movies, books, or any other media if your game contains similar characters, actions, or ideas.

The year is 2142, planet earth has hit a total population over 17 billion. Ten years ago an inhabitable planet was found within reach of human control. Despite the great value of this planet, 2 warring governments are locked in a struggle for control of the planet. Naksas, who believe the planet should only be open for people of the upper class to escape to the overcrowding of planet earth, and Simoursi who believe the planet should be open to all people no matter what status. Both factions have already built their space stations orbiting the new planet, a war has started on the space stations, and the winner will decide the fate of humanity.

2. What is the player's role? Is the player pretending to be someone or something, and if so, what? Is there more than one role? How does the player's role help to define the gameplay?

The player is a member of the Simoursi faction, playing as a member of a squad/station. His/her role is to survive the oncoming assault, with secondary (optional) objectives such as ___. The player is leading the combat team in order to defend the base. This allows for the story to progress and gives the player a key role in the upcoming battle.

3. Does the game have an avatar or other key character? Describe him/her/it.
Yes, player will be controlling an avatar of their choosing.
Have the health bar on the arm (so the user can raise their left arm to view the health bar)

How should the player feel? Somewhat afraid, but also like a badass
What should his AI allies talk about? Allusion/jokes

Weapon Choices: dual wield pistol weapons

Ammo Pickups

4. What is the nature of the gameplay, in general terms? What kinds of challenges will the player face? What kinds of actions will the player take to overcome them?
Survival
Lights being turned off/ enemy power ups/ more and more people
Turn lights on/ call backup

5. What is the player's interaction model? Omnipresent? Through an avatar? Something else? Some combination?
Halo model = can't see player face, but has a character

6. What is the game's primary camera model? How will the player view the game's world on the screen? Will there be more than one perspective?
First person perspective with NO HUD elements

7. Does the game fall into an existing genre? If so, which one?
FPS Shooter, Horde, Wave-Survival
Possible horror/thriller elements

8. Is the game competitive, cooperative, team-based, or single-player? If multiple players are allowed, are they using the same machine with separate controls or different machines over a network?
Competitive Single Player -- multiplayer possibly later

9. Why would anyone want to play this game? Who is the game's target audience? What characteristics distinguish them from the mass of players in general?
Novelty value of Oculus, feel like a badass (jedi) Jedi-Rambo
20 to 35 year olds; developers game more than player's game - M (17+)
--more hard-core gamers, more interested in game design, picky

10. What machine or machines is the game intended to run on? Can it make use of, or will it require, any particular hardware such as dance mats or a camera?
Windows 7 & 8 64 bit
Oculus, with Omni and STEM

11. What is the game's setting? Where does it take place?
Space Station, 2142 -- around a planet in the Alpha Centauri System

12. Will the game be broken into levels? What might be the victory condition for a typical level? What does the player do?

1. kill enemies
2. fix doors
3. repair cover
4. repair lights/generator
5. buy weapons from a random box (or a store)-- gets a "key" (bronze/silver/gold) based on score for each wave
6. can requisition friendly AI

7. give AI current weapon (maybe not..)

How do we want to reward the player?
--view score when it goes up (score goes on the wall -- “Stress Level of the Station”)

1. base weapon: pistol (weak)
2. Pistol (strong)
3. Deagle - Type Magnum
4. SMG
5. Assault Rifle?
6. Shotgun
7. Sawed-Off Shotgun
8. Laser Sword
9. Laser Shield (would need a sort of healthbar)
10. Laser Axe? laser banana?
11. Rocket Launcher
12. scarab gun
13. AI reinforcement (weak)
14. AI reinforcement (strong)
15. Mind gun? Change enemy AI into friendly AI
16. Heal gun?
17. Rail gun?
18. Ammo

13. Does the game have a narrative or story as it goes along? Summarize the plot in a sentence or two.
No, but there is comic relief? Store Owner. Random AI quotes in between each wave.
Further you make it in the game, more important space station becomes

H. Points and Economy Breakdown

1. counters
    1. kills
        1. current kills
        2. user total kills
        3. all total kills
        4. friendly kills
        5. self kills
        6. enemy friendly fire kills
        7. sentry/environment kills
        8. special kills (like kills per gun type)
    2. points
        1. current
        2. user total
        3. spent
        4. total spent
        5. spent on certain things
    3. time
        1. current game time
        2. total play time
    4. games played
        1. games played total
        2. games played total by user
        3. rounds completed
    5. current game wave counter
2. statistics
    1. achievement progress
    2. player stats
        1. ex. Kills/game etc.
    3. upgrade progress
3. active/inactive objectives
    1. difficulty modifiers
        1. increase enemy health
        2. increase enemy damage
        3. increase enemy numbers
        4. etc.
    2. environmental effects
        1. lights on/off
            1. they/us disorientation?
        2. Wave on/off
        3. Endgame cinematic/begin cinematic?
        4. other
    3. optional objectives
        1. turret
        2. defense helpers (walls, etc.)
4. economy
    1. wave constant

    1. 1 Wave = 1000 points = $1000 = 1U (unit)
2. wave rewards
    1. wave formula (t = total, u = wave unit, n = current wave)
        1. initialize previous t = u
        2. while (game active) {
            1. t = (previous t) + bonus
            2. bonus = u((n^2-1))\2(2(n-1)^2-1) //(integer divide)
            3. previous t = t
            4. counter++ }
        3. when counter reaches last one, reward additional u
3. kill rewards
    1. size = 1, 2, or 3 (s)
    2. formula will likely be
        1. reward = u*s^2/(s*16)
        2. sm=63, med=125, lrg=188, 10=625, 80=5000
            1. leaves room to play with levels (bosses, etc.)
4. upgrade costs
    1. unit uniform
    2. 1 upgrade unit = p = 250, level = l
    3. formula
        1. cost = p + u * ((l-1) / 8)
        2. range between 250 (level 1) – 750 (level 5)
5. gun costs
    1. begin cost determined by each gun when allocated
    2. begin cost = b
    3. upgrade formula (if we have gun upgrades)
        1. cost = b + u * ((l-1) / 8)
6. optional objective costs (maybe, may run on timed)
    1. optional objective unit = uniform = o = 200
    2. optional objective levels assigned in allocation
    3. formula: cost = o * assigned level
    4. examples
        1. turret = level 5 (1000 cost)
        2. cover wall = level 1 (200 cost)
        3. light cover? = level 3 (600 cost) ← probably not
            1. thing enemies must break to turn lights off
        4. extra helper AI = levels 2-4 (4-800)
        5. door to next room? = level 4-20 (800-4000)
            1. wood door = lvl 4 = ammo/cover room
            2. steel door = lvl 6 = entry to store
            3. blue door = lvl 10 = really good armory
            4. black door = lvl 20 = secret room

I. Research Papers

See Core Mechanics - Economy (p. 10) for Noah Leiter's research paper. References below.

Sources

1. Lane, Rick. *Why are You Addicted to Achievements.* IGN. 2011. Web Article.
2. Lehdonvirta, Vili & Castronova, Edward. *Virtual Economies: Design and Analysis*. MIT. 2014. Print.

Wiki's

- gears http://gearsofwar.wikia.com/wiki/Horde
- gears 2.0 challenges and rewards http://gearsofwar.wikia.com/wiki/Horde_2.0#Earning_Points_and_Money
- firefight http://halo.wikia.com/wiki/Firefight
- tf2 https://wiki.teamfortress.com/wiki/Mann_vs._Machine
- nazi zombies http://callofduty.wikia.com/wiki/Points_(Zombies)

Kurt Steinke
CMP_SC 4970
Dr. Dale Musser
12/16/2014

Artificial Intelligence in Game Design

Creating an engaging, fun, yet challenging artificial intelligence engine for the modern first person shooter game is a complicated and arduous task. Today's experienced video game players have come to expect highly intelligent AI, that while challenging to the player,do not make the game feel unwinnable or make the player feel frustrated. The behaviors and decisions made the AI entities in the game can be constructed in numerous different ways with each implementation having it's own pros and cons. With the advent of numerous army-like factions becoming a staple in the modern first person shooter genre, the use of organizing individual AIs into groups or "squads" to act and think as a group is becoming an important aspect of creating artificial intelligence that really appears intelligent.

What is game artificial intelligence? Well according to Wikipedia, "Artificial intelligence (AI) is the intelligence exhibited by machines or software.[1]" Simple enough right? Wrong! Artificial intelligence is actually a highly specialized field of study and its specializations often have very different goals compared to each other[1]. Let's examine another definition of AI, "...a system that perceives its environment and takes actions that maximize its chances of success.[2]" There we go! That definition sounds right doesn't it? In a game you have AI entities known as "Player Characters" which seek to simulate what a real human player would do in the environment. One would think that

the definition provided by Marcus Hatter would directly fit the actions and programming logic powering these Player Characters. However this is not the case for game artificial intelligence. This leads us to one of the biggest aspects of understanding game AI, “Even though a game tries to pursue realism and world consistency, the game’s primary purpose is to please the player that plays the game. Therefore, the player is the most important concept to be regarding when making a game.[3]” Meaning that the AI’s main goal is to make the player have fun. Often times this means taking actions that won’t “maximize” its chances of success. In general three main rules are followed to ensure that a game is fun to play against [4]. Do not try to beat the player, the AI must be predictable, and be sure the player can understand and learn the behaviors of the AI. As long as the player perceives the AI entities as “intelligent”, they are. Even if they are not actual choosing the most optimized strategy. “Our goal is to design agents that provide the illusion of intelligence, nothing more.[5]”

When it comes to creating the “illusion of intelligence” in a first-person shooter game there are many ways to make the aI appear smarter, but one of the most effective and impressive ways to do it is to place AI entities into groups or “squads “and have them work together in tandem [6]. When grouped together the AI can send messages back for between each other and can coordinate actions as a group. Generally there are two ways to do this: a decentralized system or a centralized system [7]. In a centralized system such as Halo 2[8] or FEAR[9], all AI entities communicate directly to each other. They can request help and share information with people other AI entities in their squad. This basically forms a web of one-to-one communication channels between AI. They

are working together albeit not working as one “mind”. In contrast, a centralized coordination system is a system in which the squad itself is an AI entity, all squad members check in with this AI and the “squad AI” makes the decisions for the group and orders other AI do things based on the squads needs.[6] Often this is implemented by promoting an existing member of the squad to squad leader. However this does not have to be the case as there have been numerous examples such as in the video game Unreal Tournament 3 where a central intangible mind acted as the main driving intelligence of a squad[10]. Regardless of implementation, squad drive behaviors creates a very realistic and intelligent looking AI to the experienced player.

Although creating an intelligent but not *too* intelligent artificial intelligence engine may seem daunting at first. With the write game AI design mind set, an AI can be created that is both fun and challenging. An AI architect can use squad based behavior and decision to give the illusion of very intelligent and crafty to AI entities. As long as the AI architect always remembers to put the player’s satisfaction and fun before the design of the AI, creating an AI for the modern first-person shooter that is fun, challenging, and intelligent is completely doable.

Bibliography

[1] "Artificial Intelligence." *Wikipedia*. Wikimedia Foundation. Web. 17 Dec. 2014. <http://en.wikipedia.org/wiki/Artificial_intelligence#cite_note-Definition_of_AI-1>.

[2] Hutter, Marcus (2005). *Universal Artificial Intelligence*. Berlin: Springer. ISBN

[3] Buckland, Matt. AI Techniques for Game Programming. 2002

[4] Gilgenbach M. (2006), Fun Game AI Design for Beginners. In S. Rabin (Ed.), AI Game Programming Wisdom 3 , Hingham: Charles River Media Inc.

[5] Buckland, Matt (2005) Programming AI by Example. Jones & Bartlett Learning

[6] Verweij, Tim. A hierarchically-layered multiplayer bot system for a first-person shooter" 2007

[7] Sterren, W. van der (2002). Squad Tactics: Team AI and Emergent Maneuvers. In S. Rabin (Ed.), AI Game Programming Wisdom, Hingham: Charles River Media Inc.

[8] Isla, D. (2005). Handling Complexity in the Halo 2 AI, Game Developers

[9] Orkin J. (2006), Three states and a Plan: The A.I. of F.E.A.R. Game Developers

[11] Hoang, H. (2005), MSc Thesis, Using HTN to Coordinate Unreal Tournament Bots, Lehigh University, USA

Julia Olsen
CMP_SC 4970, *Research Paper*
Dale Musser
12/6/2014

FPS Level Design

For any game, the level design can make or break it. Players come into games with certain expectations based upon the perceived genre of the title. If the player is expecting a role-playing game (RPG) and not given an open world to explore, then they will be disappointed. Similarly, if a player is expecting a first-person-shooter (FPS) and is not given sufficient cover to hide behind, they will feel cheated. Understanding the genre of the game is the first step to a successful level design for any game.[1] Once the genre is identified, the designer can focus on genre-specific designs.

There are two main areas of design when it comes to creating FPS levels. The first is the physical map that includes cover, terrain, focal points, etc. and the second is what I will call the "mental map" that consists of the map pacing, tension, navigation and challenge. Both of these must work seamlessly together to create a cohesive design. We will first explore the physical map components and then see how these work together with the mental map to create a fully enjoyable and immersive player experience.

Cover is the first element of the physical map that comes into play. The amount of cover a level includes can determine the gameplay style of that map. If the player cannot see more than a few yards ahead of him at any given time, then he is more likely to be fighting with a melee weapon than a sniper rifle. Little to no cover will leave the player feeling vulnerable to attack

[1]John Fiel and Marc Scattergood, *Beginning Game Level Design* (Boston, MA: Thomson Course Technology, 2005), 4.

and uncomfortable. Generally speaking, you want to provide the player with adequate cover so as to limit their range of shooting and put emphasis on close combat skills as well as long-range.[2] Additionally, if one half of a map has cover and the other does not it will feel unfair to the player. Making sure that every player on a map, whether human or AI, has an equal chance of winning is important for overall enjoyment.[3]

Looking at the "Sword Base" multiplayer map from *Halo: Reach* is a good example of how cover plays into map design. The map consists of two sides of a complex with a wide open area splitting up the sides. The players can either use the hallways for cover on either side and shoot across the gap or risk running across one of the walkways to attack the enemy team close range. By placing bullet-proof glass walls around rooms, having sharp corners, and using tall obstacles such as stairways to limit the player's views, "Sword Base" utilizes cover to equalize the balance of power between the two starting spawn points.

Terrain serves both an artistic and functional purpose. As art, terrain determines how believable an outdoor level appears. Since players are all aware of what nature should look like, they expect games to mimic what they already know and are disappointed when or if they fall short. This presents a balancing act for the game's designer as they choose between realistic terrain and manageable assets.[4] Terrain's functional purpose is to define the player's boundaries within a given map. Water, mountains, deserts, etc., all present a visually distinct edge that make logical sense to be impassable and therefore frustrate the player less than invisible walls.[5][6] Using

---

[2] Mike Stout, "Designing FPS Multiplayer Maps - Part 2," On Game Design, October 1, 2008, http://www.ongamedesign.net/designing-fps-multiplayer-maps-part-2/.
[3] Fiel and Scattergood, *Beginning Game Level Design*, 102.
[4] Ibid., 40.
[5] Ibid., 46.
[6] Andrew Rollings and Ernest Adams, *Andrew Rollings and Ernest Adams on Game Design* (Indianapolis, Ind.: New Riders, 2003), 63.

elevation within a level can give the player an advantage in combat by giving them a better angle of attack.[7] Additionally, these heights serve to help orient the player by providing a vantage point for them to observe their surroundings.[8]

While not a FPS, the mountains in *Elder Scrolls V: Skyrim* present a good example of how poor terrain design can be exploited by the player. While the designers intended the player to take a certain path up a mountain, it was possible to scale some mountains by leaping sideways up them instead, looking for the flattest angle. This represents two key components of terrain design, namely, that players want to go where they are not supposed to, and that playtesting terrain for usability is very important.[9]

One aspect of focal points is that of distinguishability. In order to orient the player within your level, each area of the map needs specific features that draw the player's eye and allow them to keep track of their location.[10] Generally speaking, there is one major focal point of a map followed by individual focal points in each successive area.[11] A designer wants the focal points to be visually appealing so as to draw the player's eye towards them. This gaze can help direct the player to various important elements or quest objectives, drawing them towards or pointing out important objects within the level.[12]

"Ice Station" on *Timesplitters 2* uses a large building in the center of the map as its focal point. By making it visible to the entire area, the player is instantly able to orient himself around

---

[7] Mike Stout, "Designing FPS Multiplayer Maps - Part 1," On Game Design, August 1, 2008, http://www.ongamedesign.net/designing-fps-multiplayer-maps-part-1/.
[8] Steve Gaynor, "Basics of Effective FPS Encounter Design (via F.E.A.R. and F.E.A.R. 2)," Fullbright Design.
[9] Fiel and Scattergood, *Beginning Game Level Design*, 46.
[10] Christian Güttler and Troels Degn Johansson, "Spatial Principles of Level-design in Multi-player First-person Shooters," In *Proceedings of the 2nd Workshop on Network and System Support for Games*, 158-170 (New York, NY: ACM, 2003), 168.
[11] Stout, "Designing FPS Multiplayer Maps - Part 1."
[12] Alex Galuzin, "How to Plan Level Designs and Game Environments," World of Level Design.

the building and know where he is, spatially speaking. Additionally, there are smaller buildings within each of the four corners of the map, providing area focal points. With these in place, the player is able to locate the points of interest, determine where the battles are likely to take place, and also know where he is located.

The flow of a FPS depends on the game type. In a single-player FPS, the flow should be linear in movement, drawing the player through the level to reach an end goal.[13] However, for a multiplayer FPS, the flow should be more circular and open, allowing for battling teams to fight and run into each other fairly easily. Single player levels still give the player choices in which route to take, but they also allow for dead ends, something that the circular nature of multiplayer maps forbid.[14]

The first campaign mission in *Gears of War 2* takes the player from the streets of a city, through a parking garage, and out towards another street. The player starts at one end and is forced to go forward to continue with the mission. This is in contrast to the multiplayer map "Day One," which is also set on a series of streets, but instead of a straightforward path to the finish, this map takes place in and around the buildings at an intersection. The environments and setting are similar, but because of the way the map is laid out, it has a very different feel from the campaign level. This illustrates how changing the design can change the flow of a level, as well as how single-player levels differ from multiplayer levels.

Certain levels will implement a design element called a choke point that can be used to increase or decrease the amount of tension a player is feeling.[15] A choke point is any narrow

---

[13] Stout, "Designing FPS Multiplayer Maps - Part 1."
[14] Ibid.
[15] Kenneth Hullett and Jim Whitehead, "Design Patterns in FPS Levels," In *FDG '10: Proceedings of the Fifth International Conference on the Foundation of Digital Games* (New York, N.Y.: ACM Press, 2010), 82.

passage that the player or enemy units are forced to walk through: a doorway, hall, etc. If the player is the one walking through the choke point then it increases their tension as they are more vulnerable to attack. However, if the player is the one firing from the choke point, then it decreases the tension as they can pick the enemy off as they walk through it. In the *Call of Duty: Black Ops 2* multiplayer map "Aftermath," there are three long corridors full of choke points that separate the two starting points. This area comprises the theater for most of this level's engagements, because players from either team will arrive here at the same time. Once the game is underway, either team can set up ambushes for the other at these choke points, making them dangerous areas to move through.

The first elements of the mental map we will explore are tension and pacing. These elements work hand in hand to determine how a player feels when playing a level. Pacing determines how hectic or relaxed the player is as they work though the game. A good game pace will present a balance between the two, giving the player downtime in between fights to regain their composure.[16] Tension is the strain put upon the player by the game. This can be done by the use of a game timer, non-playable character (NPC) voices, approaching enemies, or any other game mechanics that create a sense of urgency.[17]

Call of Duty created a game mode called Zombies that demonstrates good tension and pacing. The game mode consists of the player fighting off waves of incoming zombies for as long as they can live. The player starts out with only a pistol and combat knife, increasing the tension by forcing the player into close combat. Additionally, the zombies can spawn all around the player, creating more tension by the player always having to watch their back. The pacing is

[16] Fiel and Scattergood, *Beginning Game Level Design*, 14.
[17] Hullet and Whitehead, "Design Patterns in FPS Levels," 80.

well-defined with each round end. As the wave ends, the player is given downtime where the zombies cannot spawn. This ebb and flow of tension demonstrates good pacing within a FPS.

Navigation within a level employs visual cues for the player to pick up on. These include the focal points that direct the player to points of interest and provide navigational markers, and lighting and atmosphere that can lead the player to a desired area.[18] Heights can also aid in navigation by providing a player a spot to overview the entire map and create their own mental map of the area and the key points within it.[19] Navigation also differs greatly between single-player and multiplayer maps. While multiplayer maps direct the player to points of collision with an opposing team and are circular in flow, single-player maps have a start and end and direct the player down a pre-determined path.[20]

The main goal of any game is to entertain the player.[21] Game designers accomplish this goal through the challenges they present to the player within each given level. These challenges are meant to test the player's abilities while still remaining doable. If a challenge cannot be figured out or accomplished then the game loses its entertainment value and merely frustrates the player.[22] For a FPS, the goal of each level can change base upon the game type. For example, if we look at *Timesplitters 2* again, there is a campaign mission ("Chicago") which takes place around a night club. The player's goal is to infiltrate a nightclub to kill a mob boss. However, this same map is expanded and used later for a multiplayer deathmatch called "Nightclub" in which the goal is simply to score the most points.

---

[18] Ibid.
[19] Gaynor, "Basics of Effective FPS Encounter Design (via F.E.A.R. and F.E.A.R. 2)."
[20] Güttler and Degn Johansson, "Spatial Principles of Level-design in Multi-player First-person Shooters," 163.
[21] Rollings and Adams, *Andrew Rollings and Ernest Adams on Game Design*, 43.
[22] Hullet and Whitehead, "Design Patterns in FPS Levels," 80.

How the player plays the game should be up to them. However, designers can manipulate the player into a certain gameplay style by limiting the assets that are available to them, such as only providing a shotgun to the player when you want them to engage the enemy in close combat.[23] However, it is also important to respect the player and provide a consistent, fulfilling game experience.[24] One way to do this with an FPS is the previously mentioned game modes. By creating new maps for each different style of play, the designers is catering to how the player wants to enjoy the game. Some maps can be used for multiple modes, or altered to fit a certain mode, but by creating multiple game modes the designers have catered to any given player's style.

FPS level design is a combination of the map's mental and physical components. The level designer should have respect for the player by meeting their expectations of the genre. Players expect levels that include the factors I discussed here. By using these elements, a level designer caters to the player's needs and can present a fully immersive and enjoyable experience.

[23]Desi Quintans, "Level Design and Flow for FPS Maps." Level Design and Flow for FPS Maps - Desiquintans.com.
[24] Fiel and Scattergood, *Beginning Game Level Design*, 20..

Bibliography

Fiel, John, and Marc Scattergood. *Beginning Game Level Design*. Boston, MA: Thomson Course Technology, 2005. *eBook Academic Collection (EBSCOhost)*, EBSCO*host* (accessed December 14, 2014).

Galuzin, Alex. "How to Plan Level Designs and Game Environments." World of Level Design. December 1, 2011. Accessed December 15, 2014.

Gaynor, Steve. "Basics of Effective FPS Encounter Design (via F.E.A.R. and F.E.A.R. 2)." Fullbright Design. February 16, 2009. Accessed December 14, 2014.

Güttler, Christian, and Troels Degn Johansson. "Spatial Principles of Level-design in Multi-player First-person Shooters." In *Proceedings of the 2nd Workshop on Network and System Support for Games*, 158-170. New York, NY: ACM, 2003.

Hullett, Kenneth, and Jim Whitehead. "Design Patterns in FPS Levels." In *FDG '10: Proceedings of the Fifth International Conference on the Foundations of Digital Games, June 19-21, 2010, Monterey, CA*, 78-85. New York, N.Y.: ACM Press, 2010.

Quintans, Desi. "Level Design and Flow for FPS Maps." Level Design and Flow for FPS Maps — Desiquintans.com. Accessed December 15, 2014.

Rollings, Andrew, and Ernest Adams. *Andrew Rollings and Ernest Adams on Game Design*. Indianapolis, Ind.: New Riders, 2003.

Stout, Mike. "Designing FPS Multiplayer Maps - Part 1." On Game Design. August 1, 2008. Accessed December 15, 2014. http://www.ongamedesign.net/designing-fps-multiplayer-maps-part-1/.

Stout, Mike. "Designing FPS Multiplayer Maps - Part 2." On Game Design. October 1, 2008. Accessed December 15, 2014. http://www.ongamedesign.net/designing-fps-multiplayer-maps-part-2/.

VR Disorientation

Isaac Ebbert

12/15/14

Strategies for Avoiding Virtual Reality Disorientation

1. Introduction
    a. Types of disorientation
    b. Steps the Oculus Rift has taken to avoid disorientation
    c. Spatial updating and its importance
2. Hardware adaptations to the Oculus
    a. positional tracking
    b. low persistence OLED
    c. built in latency tester
    d. High refresh rate
3. Visual Effects that cause disorientation
    a. animations and particle effects to avoid
    b. load times
4. Camera Effects that cause disorientation
    a. UI/HUD elements and solution
    b. Cut scenes for VR
    c. jump cuts
5. Camera positions that cause disorientation
    a. Clipping
    b. User control of camera

Intorduction

Virtual Reality sickness is caused from Virtual Reality systems that produce effects similar to that of motion sickness.(*Cited 1*) The most common symptoms are general discomfort, headache, stomach awareness, nausea, vomiting, pallor, sweating, fatigue, drowsiness, disorientation, and apathy, as it is induced from the virtual perception of movement it is different from motion sickness which requires actual motion. The Oculus team has found that one of the leading causes of VR disorientation is caused by a low latency in the headset, *(Cited 3)* “Latency is the time it takes between your head moving to a new orientation and the correct image arriving on your retinas. In real, old-fashioned reality, the latency is effectively zero. In VR, latency is widely recognized as a key source of disorientation and disbelief (the brain cannot be fooled)”. While having proper correlation between physical motion and virtual updating of motion, it is also very important to keep spatial updating in mind during development of VR games. In a study done by Simon Fraser University they studied the effect of physical motion in VR to attempt to measure Spatial updating with and without physical motion que’s.*(Cited 4)* Spatial updating is the brains way of keeping track of the bodies relative position in space and keeping it updated based on perceived changes. This is important in life as it assists in navigating the world and it becomes even more important in VR as the perceived changes in environment can vary drastically based on the developers production of the virtual space.

Hardware adaptations to the Oculus

In the newest iteration of the Oculus Rift, Oculus has added in IR LED’s to be picked up by a camera and used to capture positional movement. This combined with the high frequency accelerometers and gyroscopes has allowed the oculus to track movement with a very high precision. Despite this there is still a need to decrease the latency further, however at this time the only possible solution is to use tricks in the programming to simulate a decrease in the latency of the hardware. The Oculus has even been installed with a latency tester in order to insure the needed low latency is achieved. The Oculus best practices documentation includes some areas for focus for minimizing motion sickness causing problems.*(Cited 2)*

- *Your code should run at a frame rate equal to or greater than the Rift display refresh rate, vsynced and unbuffered. Lag and dropped frames produce judder which is discomforting in VR.*
- *Ideally, target 20ms or less motion-to-photon latency (measurable with the Rift’s built-in latency tester). Organise your code to minimize the time from sensor fusion (reading the Rift sensors) to rendering.*
- *Game loop latency is not a single constant and varies over time. The SDK uses some tricks (e.g., predictive tracking, TimeWarp) to shield the user from the effects of latency, but do everything you can to minimize variability in latency across an experience.*
- *Use the SDK’s predictive tracking, making sure you feed in an accurate time parameter into the function call. The predictive tracking value varies based on application latency and must be tuned per application.*
- *Consult the OculusRoomTiny source code as an example for minimizing latency and applying proper rendering techniques in your code*

As mentioned above one part of the programming for a game on the Oculus rift is to include predictive head tracking in order to further minimize latency. Predictive head tracking is achieved by making calculations about the position and orientation of the head based on the previous information provided by the sensors and the information currently being supplied by the sensors. This predictive tracking is tuned from application to application by the working developers.

Visual Effects that cause disorientation

The Oculus best practices document goes into a lot of detail regarding the visual effects that have been found to cause disorientation, as such when designing our project we will be keeping these in mind.

● *The images presented to each eye should differ only in terms of viewpoint; post-processing effects (e.g., light distortion, bloom) must be applied to both eyes consistently as well as rendered in z-depth correctly to create a properly fused image.*

● *Avoid visuals that upset the user's sense of stability in their environment. Rotating or moving the horizon line or other large components of the user's environment in conflict with the user's real-world self-motion (or lack thereof) can be discomforting.*

● *Viewing the environment from a stationary position is most comfortable in VR; however, when movement through the environment is required, users are most comfortable moving through virtual environments at a constant velocity. Real-world speeds will be comfortable for longer—for reference, humans walk at an average rate of 1.4 m/s.*

Camera Effects that cause disorientation

Along with all of the visual effects to avoid disorientation there are even more concerns when it comes to the camera that the player sees through. The Oculus developers have already made significant progress in detailing proper camera usage for a FPS game utilizing the Rift. These are the details from the Oculus best practices documentation. *(Cited 2)*

●*Maintain VR immersion from start to finish – don't affix an image in front of the user (such as a full-field splash screen that does not respond to head movements), as this can be disorienting.*

● *The display should respond to the user's movements at all times, without exception. Even in menus, when the game is paused, or during cutscenes, users should be able to look around.*

● *Provide the user with warnings as they approach (but well before they reach) the edges of the position camera's tracking volume as well as feedback for how they can re-position themselves to avoid losing tracking.*

● *Zooming in or out with the camera can induce or exacerbate simulator sickness, particularly if they cause head and camera movements to fall out of 1-to-1 correspondence with each other. We advise against using "zoom" effects until further research and development finds a comfortable and user-friendly implementation..*

● *UIs should be a 3D part of the virtual world and sit approximately 2-3 meters away from the viewer—even if it's simply drawn onto a floating flat polygon, cylinder or sphere that floats in front of the user.*

● *Don't require the user to swivel their eyes in their sockets to see the UI. Ideally, your UI should fit inside the middle 1/3rd of the user's viewing area; otherwise, they should be able to examine it with head movements.*

● *Use caution for UI elements that move or scale with head movements (e.g., a long menu that scrolls or moves as you move your head to read it). Ensure they respond accurately to the*

user's movements and are easily readable without creating distracting motion or discomfort.
● Strive to integrate your interface elements as intuitive and immersive parts of the 3D world. For example, ammo count might be visible on the user's weapon rather than in a floating HUD.
● Draw any crosshair, reticle, or cursor at the same depth as the object it is targeting; otherwise, it can appear as a doubled image when it is not at the plane of depth on which the eyes are converged.

Camera positions that cause disorientation

The final concern when it comes to camera usage for the oculus rift is primarily related with how to relate physical limitations for the hardware into the unlimited digital world so that the user can maintain an immersive experience. The Oculus best practices documentation has noteworthy concerns regarding this as well. *(Cited 2)*

● Under certain circumstances, users might be able to use positional tracking to clip through the virtual environment (e.g., put their head through a wall or inside objects). Our observation is that users tend to avoid putting their heads through objects once they realize it is possible, unless they realize an opportunity to exploit game design by doing so. Regardless, developers should plan for how to handle the cameras clipping through geometry. One approach to the problem is to trigger a message telling them they have left the camera's tracking volume (though they technically may still be in the camera frustum).
● We recommend you do not leave the virtual environment displayed on the Rift screen if the user leaves the camera's tracking volume, where positional tracking is disabled. It is far less discomforting to have the scene fade to black or otherwise attenuate the image (such as dropping brightness and/or contrast) before tracking is lost. Be sure to provide the user with feedback that indicates what has happened and how to fix it.
● Augmenting or disabling position tracking is discomforting. Avoid doing so whenever possible, and darken the screen or at least retain orientation tracking using the SDK head model when position tracking is lost.
● Movement in one direction while looking in another direction can be disorienting. Minimize the necessity for the user to look away from the direction of travel, particularly when moving faster than a walking pace.

Clipping

As mentioned in the bullets one of the main concerns for camera position is clipping. Our solution for this problem is going to be a fade to black so that when the players camera/head is inside or outside of geometry they cannot see anything and they return to the area that they are supposed to be in.

User control of camera

The final concern when it comes to disorientation prevention for the Oculus is to make sure that the player is always in direct linear control of the camera that is linked to the viewport for the Rift. Any changes or loss to control can lead to serious discomfort and is one of the main causes for Virtual Reality sickness. *(Cited 1)*

Works Cited

1. Virtual Reality Sickness. Wikipedia.org. Accessed 12/15/2014 http://en.wikipedia.org/wiki/Virtual_reality_sickness
2. Oculus Best Practices. Oculus. December 5. 2014 version. http://static.oculus.com/sdk-downloads/documents/Oculus_Best_Practices_Guide.pdf
3. The Latent Power of Prediction. Steve LaValle. https://www.oculus.com/blog/the-latent-power-of-prediction/
4. Can Physical Motions Prevent Disorientation in Naturalistic VR?. Simon Fraser University. Abstract. http://www.researchgate.net/publication/254026628_Can_physical_motions_prevent_disorientation_in_naturalistic_VR/links/02e7e529a743c431f2000000

Samuel Jonesi
Capstone I - *Research Paper*
12/16/14

Video Game Programming

Introduction

The main part of the project I am a part of is the use of new peripherals to create a more realistic experience in gaming. These peripherals include the Oculus Rift, the STEM System by Sixense and the Omni treadmill by Virtuix. While I am pleased with the project I have chosen and the ways it is innovating, what really fascinates me are the methodologies and algorithms behind the software of video games and how they can benefit computer science as a whole.

My first experience in video game programming was in my microprocessor engineering class. In this class, one of the projects involved creating an "asteroids" style video game in the x86 programming language. Creating this game was the first time I had done any programming project of this size before. It required me to be creative and use solutions that before seemed like the wrong thing to do, like using an infinite loop as part of my program. The following paper will give a broad overview of related programming methodologies and techniques that are used in video game design, including the use of game loops, finite state machines, tweening, and multithreading.

The Game Loop

When I was creating the space invaders video game, I wondered how I would be able to constantly keep processing running. I came up with a simple solution that had already been thought of decades before: keep an infinite loop running. Along with this loop, I would have other processes are work in order to set up the main game loop to accommodate the user as well as perform tasks required for after the game is over. Again, this is nothing new.

Game loops have been the workhorse for video games as long as there were video games. One early type of video game that demonstrates this quite well is the text-based video game.

```
You are crawling over cobbles in a low passage. There is a dim light at
the east end of the passage.

? east

You are in a small chamber beneath a 3x3 steel grate to the surface.
A low crawl over cobbles leads inward to the west.
```

Input and output taken from the game Colossal Cave Adventure[8].

The text-based video game has a loop that is constantly waiting for user input. Once that input is given, it will respond with the appropriate output. Modern video games are, of course, much more sophisticated than this, but the basic idea remains the same. Instead of simple commands, mouse and keyboard, or game controller, or even a peripheral controller such as the Oculus Rift is used. Output is given in the form of graphics, sound, and/or

vibrations instead of text. The simple idea of having the video game wait for input from the player, update its state, then returning appropriate output is still the same, however [6].

At its simplest, the video game loop can be thought of as continuously performing three instructions [6]:

```
while (true)
{
  processInput();
  update();
  render();
}
```

This is not powerful enough for even the simplest video game, however. Video games require functionality outside of mere gameplay. They often require a saved state for the user, along with an options menu, credit sequence, and sometimes a record of high scores. In fact, the main game loop will often account for a mere 10% of the code [6]. The following can be thought of as a more complete example of video game instructions:

a. Game intro and interface
b. Game level interface that allows users to select options such as weapons
c. Game level initialization and loading of objects
d. The game loop proper
    i. Handle the gameplay
    ii. If player wins goto sequence a
    iii. If player loses, goto failure sequence then goto a if user gives up or b if user wants to try again[1]

It is best practice to allocate and deallocate arrays for any objects that will be active during gameplay before the main loop begins. This is because the developers cannot risk gameplay being stopped mid-game due to a problem with allocation or deallocation of memory[1].

The game loop itself for any non-trivial video game is a difficult piece of real-time programming with many moving parts. It is the most important part of the video game code and accounts for 90% of the time the game is spent in [6]. The many tasks that are required to make a game function in the game loop itself include, but are not limited to:

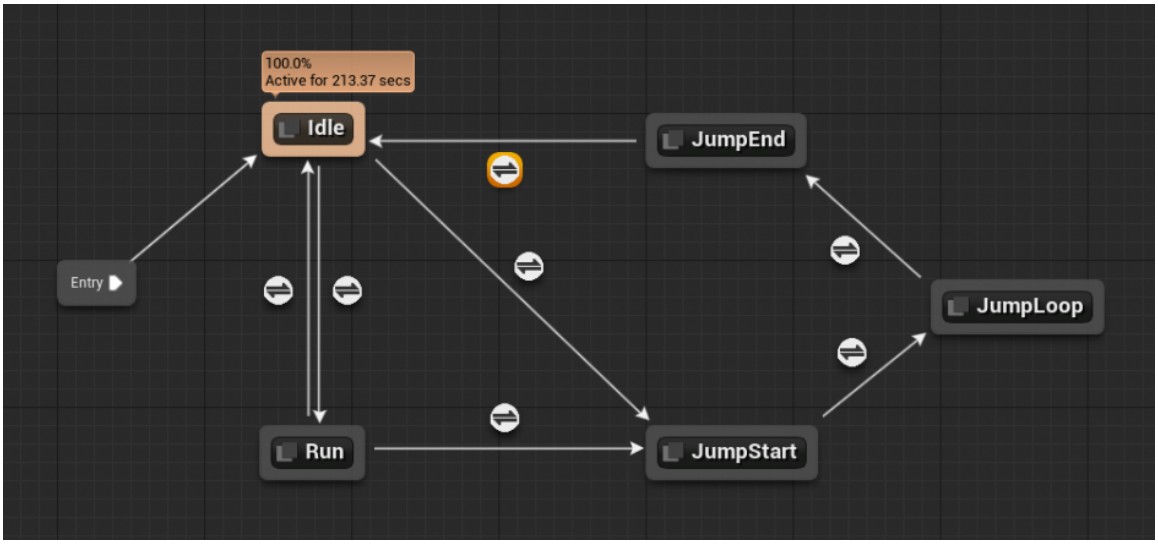
-Reading user input (including network data)
-Calculating user parameters based on user input
-Calculating non-playable character AI
-Draw graphics
-Handle sound effects[1]

An example of the game loop I would have used in my asteroids game is:

```
while(true){
        moveAsteroids();
        checkAsteroidCollision();
        if(playerClicked() == true)
                fireBullet();
        moveBullets();
        checkBulletCollision();
        if(playerClicked() == true)
                fireBullet();
        if(health==0)
                break;
}
```

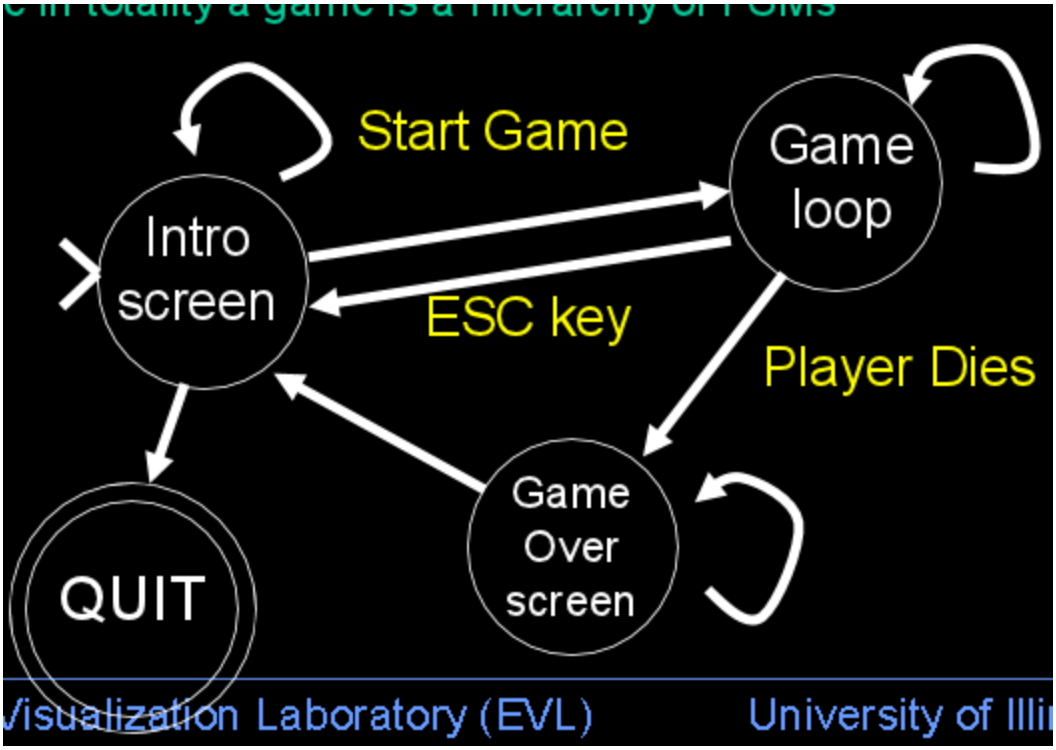
In my asteroids game, the user input included the mouse to aim and fire the weapon. The parameters included where the "bullet" that was fired moved. The non-playable character AI could be seen as calculating where the asteroids moved. Drawing the graphics included drawing the space ship, bullets, and asteroids where they were meant to be drawn and erased where they were not. There were no sound effects.

The Use of Finite State Machines in Gaming



A finite state machine to be used in animation. Taken from the Unreal Engine tutorial on making a first person shooter my group will be using [7].

Finite State Machines are one of the most useful abstractions in gaming. They consist of states, inputs, and transitions. The game as a whole can be seen as a finite state machine, with the components mentioned in the previous section as states in the finite state machine. Each of these components contain finite state machines within themselves. Video games can thus be seen as a hierarchy of finite state machines.



An abstraction of video games as a Finite State Machine. In the real world, each node would contain a finite state machine in itself.

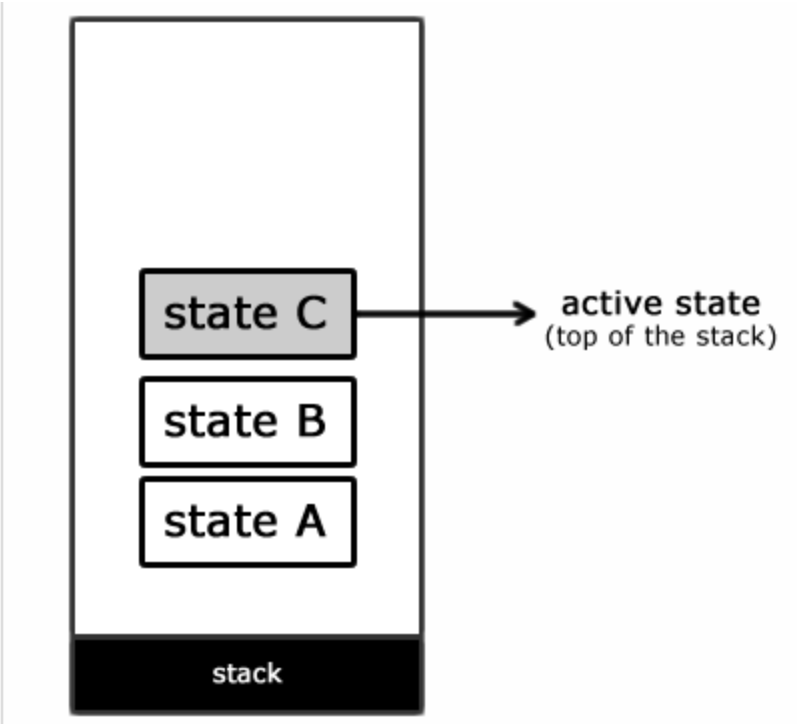
One of the reasons why the hierarchy of finite state machines is important is because, without it, the game's code would quickly devolve into spaghetti code. A hierarchy of FSMs allows for proper modularization of code in which inputs, outputs, and transitions of state must happen in real-time. At the lower level, they can be used to keep track of several objects such as bullets or enemies. The states themselves can be simple or complex and keep track of properties such as position and health [1].

```
01  public class FSM {
02      private var activeState :Function; // points to the currently activ
03
04      public function FSM() {
05      }
06
07      public function setState(state :Function) :void {
08          activeState = state;
09      }
10
11      public function update() :void {
12          if (activeState != null) {
13              activeState();
14          }
15      }
16  }
```

An example of what finite state machine code might look like [6].

In this implementation, states are represented through functions. The update() function is called once per frame. It calls the function pointed to by the activeState() property. The setState() property points the finite state automaton to a new state, which may not necessarily be contained by the FSM. This grants the FSM more flexibility, important for video games in particular [4].

Sometimes it is appropriate for a finite state machine to transition to the state it was before it was in the current state instead of a completely new one. For this, FSMs need a form of memory. One solution that has been used for this purpose is FSMs with a stack.



A diagram of a finite state machine with stack.

When the state needs to go back to the state where it was before, it can simply pop state C and move to state B instead of having to remember complex rules. An example is in the *Metal Gear Solid* series. Whenever the main character is spotted by enemy soldiers, an alarm is sounded, and the game shifts state. It is possible that the game keeps a stack of states and that when the alarm is over, the state is popped and the game returns to its normal non-alert state.

One of the differences between implementing an "ordinary" FSM and a stack-based FSM is that while non-stack based FSM's only require the method changeState(), the stack-based FSM requires pushState() and popState(). With these two methods, the current state can get updated while previous states are kept in memory. This can lead to more complicated programming, as each state needs to know not only when it is to be updated, but when it needs to pop itself from the stack or when another state needs to be pushed onto the stack [5].

To put it another way, the finite state machine has a current state along with an input queue. The process() method of this state checks what state the FSM is in, then checks the input queue to see if any messages are relevant for changing states. If there is, it performs the necessary actions which might include changing state. If not, it breaks. This process might be continuously running, or it might be a part of the state of another FSM that is not active.

An example is a bullet. A bullet may have the following states:

- **DormantState**
- Bullet particles hidden

- Stays this way until it is told to set the currentState to initState

- **InitState** //activate the bullet
- Initialize the location of the bullet and show it on screen
- current state becomes the movestate
- **Move State** //Move bullet
- Move bullet along trajectory
- check if collided with an object
- If collided:
- If object==tank then tank.input1="hit"
- currentstate = BulletExploded
- break;
- **BulletExplodeState:**
- Bullet gets hidden
- Enable explosion
- currentState = WaitForExplosionState
- break;
- **WaitForExplosionState**:
- explosionCounter++
- If explosionCounter = 100 then explosion is over; currentState = dormantState;
- break;[1]

One of the problems with having a game loop is maintaining consistency no matter what the processing power of the hardware is. This is especially important if the hardware is single-threaded and thus one core is required to take care of every part of the game loop. Old video games did not address this problem because there was not a wide variety of hardware performance back then. The way old video games would often calculate movement is:

```
PosX = PosX + some_unit_distance
```

where the bigger the some_unit_distance, the more the object at PosX moved[1].

A real example of this is a simple Flash game I played online called "RaidenX," a 2D shooter game. When I played this nearly a decade ago, it had a game speed that was just fine. When I play it today, however, it is so fast that it is nearly unplayable. Another more-famous example is the Space Invaders video game, originally programmed in a way that the movement speed of the invaders never actually changed, but once there were fewer enemies to render, the hardware would naturally make them move faster.

The proper way of enforcing the real-time game speed is to consider the processing speed of the game loop. This time needs to be recorded and kept as a variable (like say, dt). This elapsed time can be used to determine where an object should be during the next iteration of the game loop. An example of this in action is, if we want a car to move 30 feet per second, we could move it like so:

```
posX = posX + (speedX * dt)
```

In this example, speedX could be 30 feet per second. We multiply that by the number of seconds that have elapsed and that gives us the next position to render the object that was at posX[1].

It is worth noting that video games that are shared across networks make consistent speeds of objects between players more difficult. If player 1 is using a modern gaming PC while player 2 is using a consumer-grade PC that is 5 years old, the difference in the framerate and calculations between the two would be like night and day. If not programmed and properly accounted for, the same object, such as a projectile or a player, could be in two different places at once. It could be at once place on player 1's PC and at a completely different place at player 2's PC! As if that wasn't enough, the lack of concurrency between the objects could cause the physics engine to blow up and cause objects to spontaneously launch themselves into the air [6].

Tweening

Tweening takes the idea of calculating graphics updates based on the time of the game loop, and applies it more globally. Before tweening can be applied, first one must consider the update rate for the calculations. During each calculation, the state of the world and the amount of time needed to perform the calculation is recorded. The amount of time needed to perform a single calculation is divided by the amount of time necessary to perform the entire game loop. This is the *tween value*. It can be used to determine how far between two different states a variable, such as position is. This allows for variables to be updated more continuously and not be dependant on the main game loop[1].

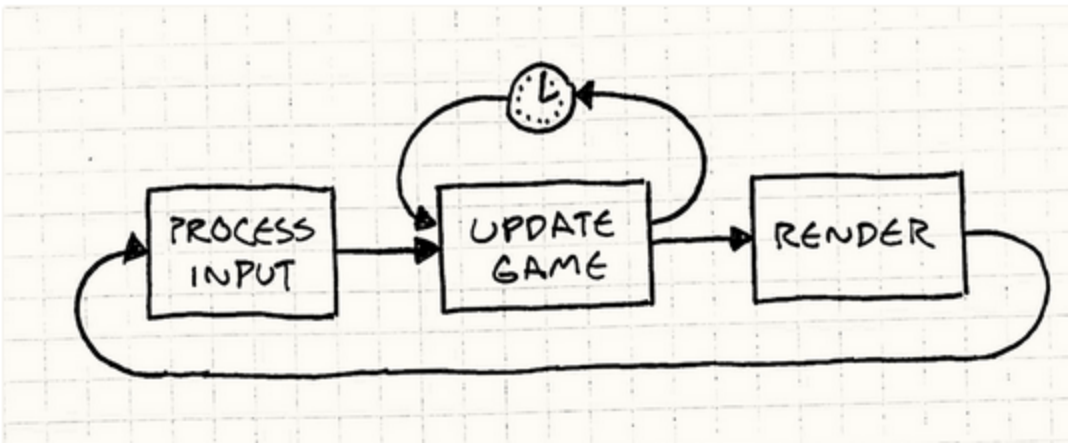
Tweening is *not* the rate of updating the graphics, however. Updating graphics and updating variables need to be seen as two separate functions. By keeping them independent, the graphics may not always update perfectly (even the best games on the best PCs with the best networks will occasionally have "hiccups"), but it can always be kept accurate to the variables at that moment.

Here is what code for tweening might look like [6]:

```
double previous = getCurrentTime();
double lag = 0.0;
while (true)
{
  double current = getCurrentTime();
  double elapsed = current - previous;
  previous = current;
  lag += elapsed;

  processInput();

  while (lag >= MS_PER_UPDATE)
  {
    update();
    lag -= MS_PER_UPDATE;
  }

  render();
}
```

It starts off by initializing previous to the current time and setting the lag equal to 0. After that, the loop begins. It finds the current time and the time elapsed between the current and the previous time. It then sets the "previous" time to the current (to initialize the next calculation) and adds the elapsed time to the lag. After that, it processes the input. The interesting part is that, while the lag is greater than the number of milliseconds per update, it will continually update the game state while reducing the amount of lag. Finally, after the game state is updated, the game is rendered.

Here is a visualization of this process [6]:



One must be careful in selecting granularity for updating. Update too often precious time is wasted. Update too slowly, and the game becomes choppy. It is especially important that the update granularity not be so short that it takes more time to update than the amount of time between updates. That would cause the game to lock up completely. The good news is

that a safeguard is sometimes put in place that allows for a maximum number of iterations before the game loop quits [6].

Multi-threaded Game Loop

Often the many sub-components of a video game can significantly slow down the main game loop. This is especially true of the AI. The solution to this is to use multithreading to allow multiple game loops to compute simultaneously. This is prevalent in modern, graphics-intensive games. An example of this is Far Cry IV, where the minimum requirements for the CPU include the quad-core AMD Phenom II X4 [2], or the dual-core dual-threaded Intel i5 [3][4].

Examples of processes that can be broken up into multiple threads include the input, the computation, the graphics, and the sound. These loops need to progress as fast and as independently as possible[1]. The graphics often use the GPU to render itself due to the high parallelization that graphics allows, but discussion of this is beyond the scope of this report.

Sharing Variables in a Multi-threaded Game Loop

Due to the complexity and real-time nature of video games, a study in video game programming is not unlike a study in operating systems. A multi-threaded game loop adds an additional element of difficulty when it comes to sharing variables properly. Global variables need to be shared across multiple threads. Variables in forked processes are local to the process. In forked processes, variable sharing is done using shared memory when the Unix API is being used. Threading and forking lead to the additional risk of a variable being changed while another thread is using it. Mutexes are required to maintain variable integrity across threads. This leads to additional time overhead, so mutexes cannot be used for every variable[1].

Triple Buffering

A common compromise between security and performance is known as triple buffering. As the name implies, a triple buffer is a buffer that stores a single variable and copies it three times. This buffer is to be shared among more than one process. When one process updates the variable, the local buffer can then swap with the other two to allow other processes using the triple buffer to update accordingly. An example is, once a new position is computed, its buffer can be swapped with the other two so the draw process can update the graphics.

These buffers can be implemented with an array of pointers. This array can then have the mutex locked on it so it can safely perform the swap without any part of the buffer getting updated and having inconsistent data[1].

Design Decisions

Before making a game loop, there are many crucial design decisions that must be made. First, the question you must answer is who *owns* the game loop? Are you using the platform's game loop, the game engine's loop, or are you writing the whole thing yourself?

If you have the platform own the game loop, you have the advantage of simplicity, because you are letting the platform take as much work off your shoulders as you can. Part of this simplicity includes a game loop that is guaranteed to work well with the platform as it is constructed by it. What you are giving up is control over the timing. This means the platform will call sections of code as it sees fit. When making a web browser based game like Flappy Birds, the progammer is forced to use this form of the game loop.

Another way of programming is to use a game engine loop. This has the advantage that basically any other library has. The programmer does not have to write, test, and debug this code, so if it is a quality game engine, it should work fairly well. The downside is that, unlike standard libraries like stdio.h, the game engine needs to fit the task of a specific video game. This means that it will not be optimal.

This is the way that my group will be making our game loop: using the Unreal 4 Engine. It is vitally important when making our game loop that we ensure that hiccups are an extremely rare occurrence, or better yet, never happen. The game must be kept at a constant frame rate of 60 fps or better. This is to ensure that a gamer using the Oculus Rift headset does not get sick.

The final way of making a game loop is to simply write it oneself. This has the advantage of being tailor-made for the video game. If our group had infinite time, we would write the game loop ourselves to focus especially on consistent graphics rendering. Writing the game loop has many downsides. In the case of our group, the downside is that there would be many man-hours lost creating the game loop that the engine already would have taken care of. In addition to simply writing the game loop, it would have to be tested and debugged. Even a small bug can have huge repercussions.

Conclusion

This paper was an introduction to certain techniques used specifically for video game development. There are many, many more techniques used specifically in video game programming that could be mentioned in this paper. Most, if not all of these techniques are going to be taken care of for us by the Unreal IV Engine. It is useful to keep these techniques in mind when making the video game, however, as knowledge of these mechanics will aid in both development and debugging of the product. By gaining real-world experience in how these elements of programming video games interact with the Oculus Rift, the Sixsense STEM, and theVirtuix Omni, this project is a worthy of being a capstone for not only information technology, but computer science as well.

1: Leigh, Jason. "Game Software Design." CS 426. Electronic Visualization Lab. University of Illinois at Chicago, Chicago. 1 Feb. 2013. Lecture.
2: AMD Phenom™ II Processors." AMD Phenom™ II Processors. Advanced Micro Devices, Inc., 1 Jan. 2014. Web. 27 Nov. 2014.
<http://www.amd.com/en-us/products/processors/desktop/phenom-ii>.
3: "Intel® Core™ I5 Processors." Intel. Intel Corporation. Web. 27 Nov. 2014.
<http://www.intel.com/content/www/us/en/processors/core/core-i5-processor.html>.

4: "Far Cry 4 System Requirements." Far Cry 4 System Requirements and Far Cry 4 Requirements for PC Games. Web. 27 Nov. 2014. <http://www.game-debate.com/games/index.php?g_id=8147&game=Far Cry 4>.
5: http://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867
6: Nystrom, Robert. "Game Loop." *Game Programming Patterns*. Print.
7: "First Person Shooter C Tutorial." - *Epic Wiki*. Web. 16 Dec. 2014.
8: Manual, Rob. "How Adventure Games Came Back from the Dead." *PCWorld*. 5 Feb. 2013. Web. 17 Dec. 2014. <http://www.pcworld.com/article/2026802/how-adventure-games-came-back-from-the-dead.html>.