5-20-2015

Crystalline

Nicholas Buonarota

John Dunham

Alexander Herdzik

Matthew Mundell

Crystalline

By

Nicholas Buonarota

John Dunham

Alexander Herdzik

Matthew Mundell

Project submitted in partial fulfillment of the requirements for the degree of
Master of Science in Game Design and Development

Rochester Institute of Technology

B. Thomas Golisano College of Computing and Information Sciences

May 20, 2015

Acknowledgements

We would like to give a heartfelt thank you to all of our committee members for helping us craft *Crystalline* into the game it is today. These people include Dr. Jessica Bayliss, Kevin Bierre, Alberto Bobadilla, Chris Cascioli, and Jesse O'Brien. In addition, we would like to thank our supporting professors at the Rochester Institute of Technology and the Interactive Games and Media Faculty in general. Additional thanks go out to all who playtested our game, including the designers at Raven Software, playtesters at Imagine RIT and RPI GameFest, and family and friends who played and tested our game while in development.

Figure 1 Meme of Han Solo (Han Solo Meme, n.d.)

Executive Summary

Crystalline is a fast action arena shooter with a focus on gunplay. The core objective of this project was to create a fun multiplayer First Person Shooter. To achieve this goal as a team we had to best leverage the tools and technology available to us.

As First Person Shooter games typically have teams far larger than our own, we had to work hard and smart on *Crystalline*. Unreal Engine 4 was used in lieu of Unity or an in-house engine, saving hours of development time and allowing us to focus on gameplay and assets more. Thanks to Unreal Engine 4, we were able to produce a game that, based on playtesting, appears to meet our core objective. Due to the limited time available for the project, there are still far more designed features to be implemented. However, the core gameplay has been completed leaving opportunity for expansion and future work.

This document is divided into nine chapters and an appendix. Chapter 1 will introduce readers to the core concepts of *Crystalline*. Market analysis and background research are covered in Chapters 2 and 3 respectively. The prototypes and general process that took *Crystalline* from concept to game are outlined in Chapter 4. Chapters 5 and 6 outline the core design of the final iteration of *Crystalline,* technical or otherwise. Chapter 7 describes overall visual designs of the game, both 2D and 3D. Playtesting data is reported and assessed in Chapter 8, and a post mortem is detailed in Chapter 9. This document concludes with an appendix containing an asset bible.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Tagline

“The 8 player networked free for all, where magic fantasy weapons meet fast action gunplay.”

1.2 Vision Statement

Crystalline is a fast-action, arena shooter which takes place in the far future. In *Crystalline,* players use both their twitch reflexes and planning skills to harvest crystals and defend their space fleet. By modifying their weapon, players have a free reign on how they approach their goal of planetary domination.

1.3 Elevator Pitch / Paragraph Summary

Crystalline is an 8 player networked first-person shooter that takes place in the far reaches of outer space. *Crystalline’s* gameplay revolves around combat in which players modify a single weapon to defeat their opponents. The weapon sandbox can be thought of like the magical weapons from *Dungeons and Dragons* or other fantasy adventure games. In *D&D*, “magic runes” can often be inserted into weapons giving them extra abilities. For example, if you insert a “fire rune” into your sword, then it becomes a magic sword of fire and does additional fire damage. Likewise in our weapon sandbox, if you insert a “precision modifier crystal” into the crystal gun it becomes more accurate and behaves more like a single shot rifle than a fully automatic rifle. Our weapon system brings a new and unique approach to First Person Shooter (FPS) weapon sandboxes by incorporating some of these upgrade mechanics found in Role Playing Games (RPG’s).

2. Game Genre Background and Market Analysis

2.1 Genre Background

Genre: First Person Shooter

Fast skirmishes punctuated with strategic planning to take control of the battlefield define the typical match of *Crystalline*. The core control metaphor and combat mechanics draw heavily from the Action and First Person Shooter genres, the current top selling video game genres (Entertainment Software Association 2013). The FPS elements take cues from franchises such as *Tribes* and *Halo,* focusing on fast paced combat and an interesting weapon sandbox. As a point of innovation in the FPS genre we've co-opted elements from the classic Sega Genesis game *Gunstar Heroes*, namely the power up slotting mechanic, to make the weapon sandbox far more accessible and customizable for the entry level player. In addition to the FPS genre, *Crystalline* was originally planning to borrow select elements from the MOBA genre. The primary game modes in *Crystalline* were also meant to have a lane structure in which the player must simultaneously capture and defend "lanes" similar to a MOBA game such as *DOTA 2* and *League of Legends*.

2.2 Competitive Analysis

2.2.1 The Market

Abstract: Here we acknowledge similar games in the marketplace, how they impacted our design, how our game is different from those products, and finally how we plan to improve upon ideas and mechanics expressed in these games.

Title	Gameplay	Mechanics	Takeaway
Battleborn	FPS MOBA	Team based play, unique characters	team-oriented play
Battlefield	FPS	large maps, vehicles	details matter
Borderlands	FPS, RPG	Co-op play, Massive Weapon Sandbox, Skill Trees	Interesting Weapon Sandbox, Multiplayer Experience
Call of Duty	FPS	action	solid experience that people return to year after year
Firefly	TV Series	space cowboy	look and feel
Halo	FPS	arena multiplayer	game feel
League of Legends	MOBA	Outpost Destruction, Levelling up	Lanes are good, Shorter matches means happier players
Natural Selection	FPS / RTS	unique combinations of genre, adding more to the average FPS match	too much can overwhelm new players
Nuclear Dawn	FPS / RTS	unique combinations of genre, adding more to the average FPS match	simply mashing two genres together doesn't always work
Star Wars: Battlefront	Third Person Shooter, Conquest	Node Capture, Unique Roles	Feeling of a large scale war, node capture
Tribes Series	FPS, CTF	Jet pack, Skiing, Base Equipment and Deployables	Focus on Base strategy and upgrading
Titanfall	FPS	High-tech weapons, fast-paced gameplay	People like fun mechanics

Table 1 Summary of Similar Games Analysis

Battleborn (Gearbox Software n.d.) **-** After the success of *Borderlands*, Gearbox is now working on Battleborn, an FPS Multiplayer Online Battle Arena (MOBA). Preliminary gameplay video and news shows a game like *League of Legends* or *Defense of the Ancients (DotA)*, but in the first person perspective. Teams of 5 unique characters face other human or computer controlled players, and each character has unique weapons and abilities.

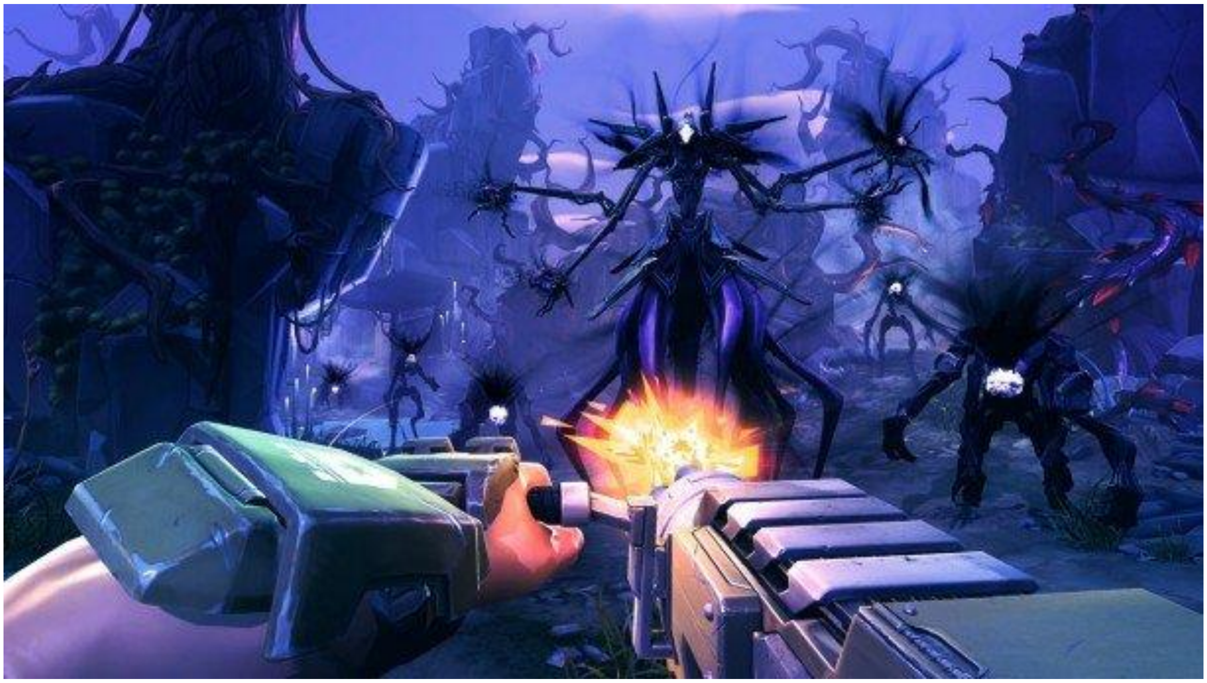


Figure 2 Battleborn Gameplay (PC Gamer 2014)

Battleborn takes the traditional elements of a MOBA, specifically the unique characters and team-based play, and adds an FPS perspective. We wanted to have team-based play and a lane based map like in MOBAs, but our gameplay is based more on traditional FPS games where characters are not unique innately and more than 5 players per team are supported. *Battleborn* is a MOBA, and *Crystalline* was originally intended to be a FPS with some elements of MOBAs.

Battlefield Series (DICE 2013) **-** A popular FPS, the *Battlefield* games feature a focus on large maps, teamwork and vehicle warfare, when compared to other traditional first-person shooters. The PC games in the series are mainly focused on online multiplayer. The *Battlefield* series has served more than 50 million players worldwide as of 2012, and with a popular hit iteration after iteration, it should not be ignored.

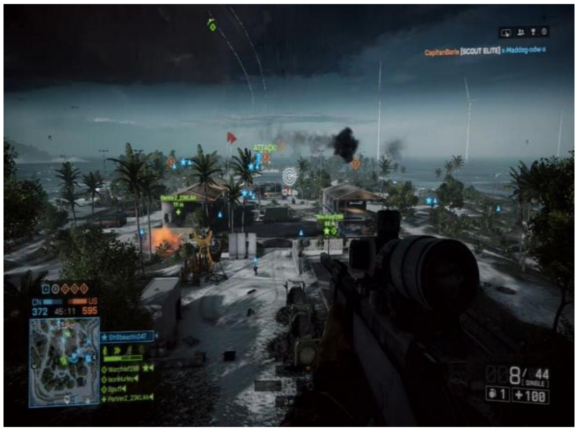


Figure 3 Battlefield 4 Gameplay (BF4Central 2015)

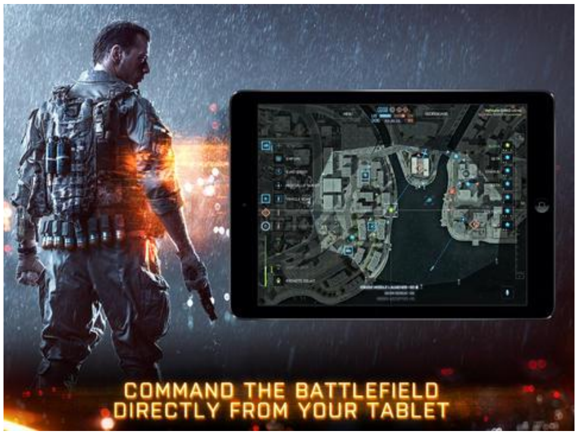


Figure 4 Commander Tablet Application (Tom's Hardware 2014)

In *Battlefield 4*, there was a second screen tablet “Commander” application. This was a feature that our team was very impressed with and influenced a few aesthetic features throughout the game development.

Borderlands series (Gearbox Software 2012)**-** *Borderlands* is Gearbox’s flagship FPS/RPG franchise. Boasting a near infinite amount of guns, character classes, skill trees and more dubstep than recommended by a sane physician. *Borderlands* is largely story driven, but the real fun comes in the four player co-op that allows players to explore the world of Pandora, rich in atmosphere and character, utterly destroying anything in their path with a litany of firearms, explosives and powers. Not surprisingly this is fun, and now with three games in the series it shows no signs of stopping.

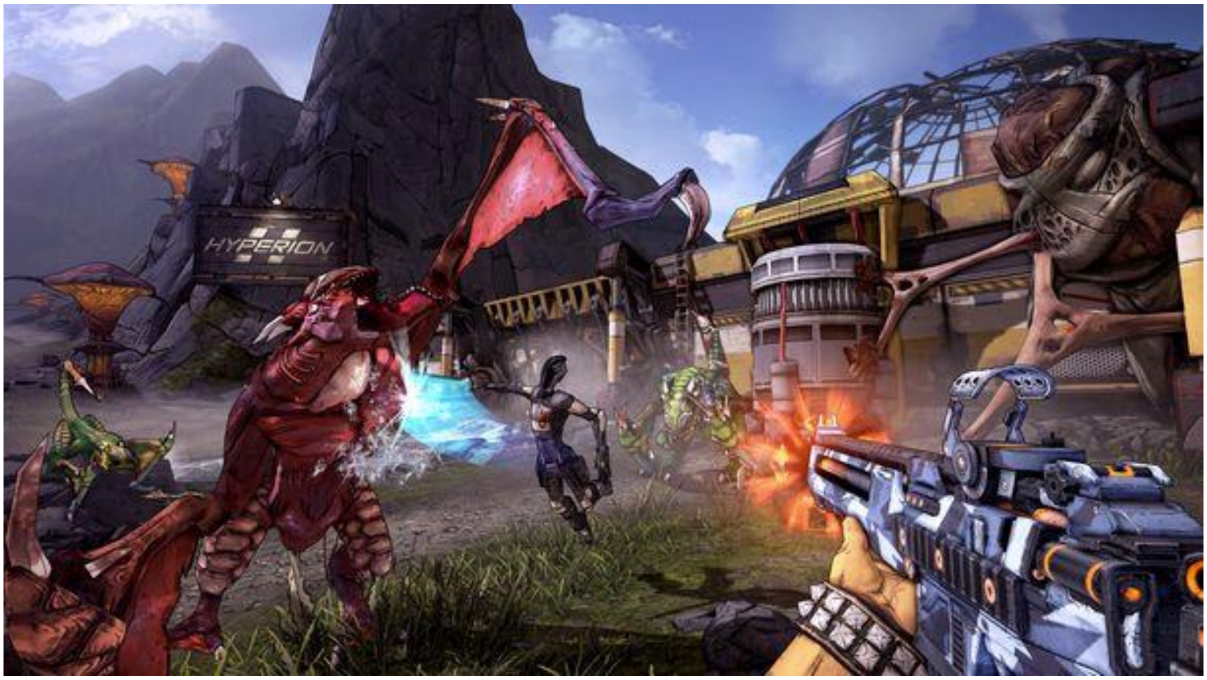


Figure 5 Borderlands 2 co-op gameplay (Next Gamer 2012)

Borderlands integrates RPG and FPS elements in a near flawless manner. The player doesn't feel limited by the skill progression, as they can refund their skills and try and find a combination that works for them. Additionally, the RPG skills *feel good* to use: when the player phases in and out as a Siren they can feel her power and ability, but are still able to use a gun and pick the enemy off without getting their hands dirty.

The co-op feels like it adds to the game rather than detracting. This largely due to the fact that players are still able to explore the environment as they see fit, but can share in crazy fights and experience the alien scenery together. *Borderlands* is good at making emergent stories in this way. That being said, not having friends can be less interesting. The game becomes a bit of a grind fest when a player is alone and while the humor is great in the single player, jokes alone can't carry player interest. Additionally, while the weapon sandbox is great the weapons sometimes feel "floaty", as a result the satisfaction from using weapons can be sometimes diminished.

Borderlands influenced *Crystalline's* emphasis on an interesting weapon system and helped guide how to integrate disparate genre mechanics. That being said *Borderlands'* weapon system is not always accessible to new players, a fact we are attempting to account for in our weapon sandbox's power up based system.

Call of Duty Series (Activision 2015) - The *Call of Duty* series is one of the most known FPS intellectual properties on the market. From titles ranging from WWII to future sci-fi settings, the series has sold over 100 million copies. The series is one of the top grossing FPS games and has dominated the multiplayer console FPS market for the last few years. Its faced paced action is top notch.

Figure 6 Call of Duty 2 (Takahashi 2015)

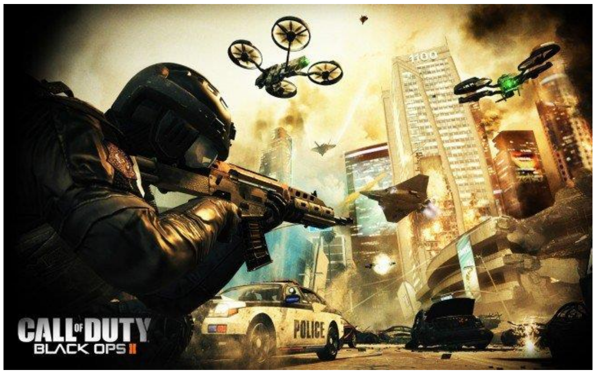


Figure 7 Call of Duty: Black Ops II (Takahashi 2015)

This is not a game series to be ignored if you are planning on developing an FPS and going to market. One major feature that the last few *CoD* games have had is the "custom starting load out" feature. This lets players spawn with exactly the weapons they want. In *Crystalline*, we plan to implement a different type of weapon system, which encourages player movement on the battlefield and player on the spot adaptability.

Firefly

"[N]ine people looking into the blackness of space and seeing nine different things"
-Will Wheaton (quoted in Brioux[2004])

An iconic work in the realm of space westerns, *Firefly* has inspired our aesthetic and thematic choices throughout the process. The series practically codifies the notion of the "space cowboy" instilling a sense of wonder and adventure in the viewer. *Firefly* also captures a sense of independence and survival that incorporates technology in a manner we've found important to our design at all stages. The character designs are striking, but simple, inspiring the notion that these characters have travelled the half 'verse, and are willing to travel the other half to live freely and survive. As *Crystalline* at its core is about two sides attempting to survive in space, we've found this general aesthetic feel to be valuable.

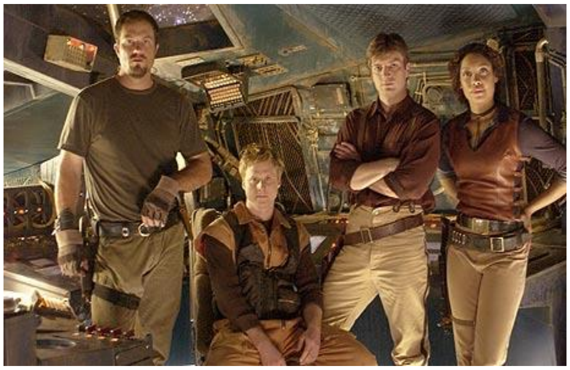


Figure 8 Firefly Clothing (Massey 2012)

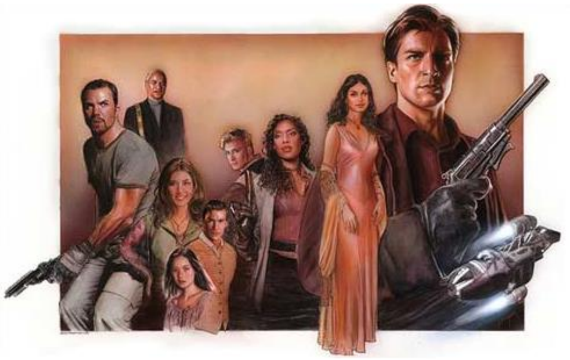


Figure 9 Firefly "space cowboy theme" (Palmer n.d.)

Halo Series (Microsoft 2015)- *Halo* is a first person shooter with a heavy sci-fi narrative campaign and a genre revolutionizing multiplayer experience for console FPSs. The game series got many positive reviews for its multiplayer experience. Bungie had a runaway hit. It was so good that many console FPSs tried to emulate its qualities for the next 10 years. *Halo: Combat Evolved* the first entry in the series got a lot of things right in transferring the FPS experience from a PC to the console and set up a great foundation for the series multiplayer experience by creating a quality local multiplayer.

Halo 2 innovated in the multiplayer space even further, revolutionizing how console multiplayer was delivered at a fundamental level. Some of the major additions to *Halo 2* included Xbox Live, the networking multiplayer aspect for Microsoft's Xbox. *Halo 2* set industry standards for Xbox Live by establishing and highlighting the party system. The party system took the "friends list" and let players have pre-game lobbies that carried over from match to match. This was Bungie's attempt to “virtualize” the couch multiplayer experience they refined from *Halo: Combat Evolved*. The other major contribution was the idea of a “quick play” system. Opposite to the traditional, host a server - join a server model, “quick play” had the player push a button, which effectively said, “I want a play a game. Find me other players that want to play. Pick a host. Pick a game type. Pick a map. Get me into the game as fast as possible.” In addition to these innovations, the *Halo* series had many other great qualities to admire. The attention to detail and game feel is another quality of the series that we highly respect.

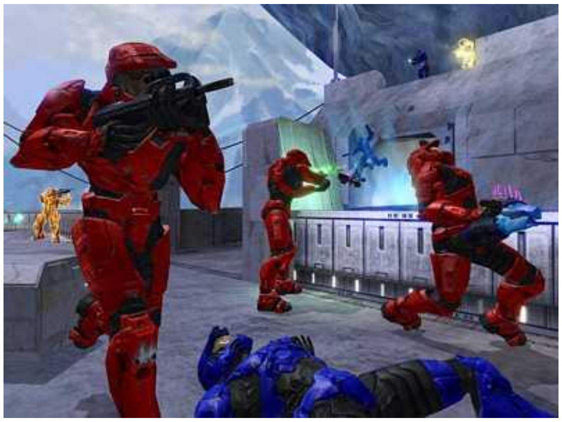
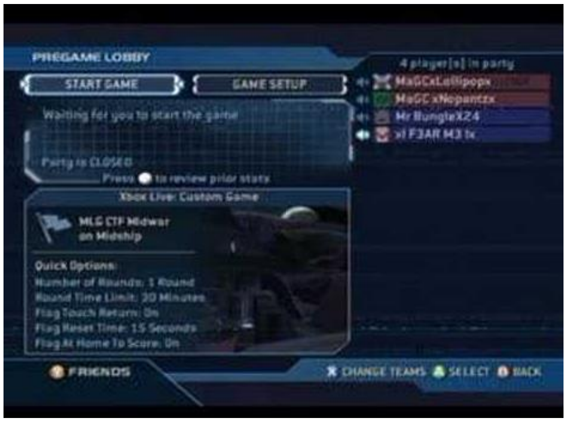


Figure 10 Halo 2 multiplayer gameplay (jkdmedia 2005)



4 player(s) in party
MaGCxLollipopx
MaGCxNepantzx
Mr BungleX24
xI F3AR M3 Ix

Waiting for you to start the game

Party is CLOSED

Press ● to review prior stats

Xbox Live: Custom Game

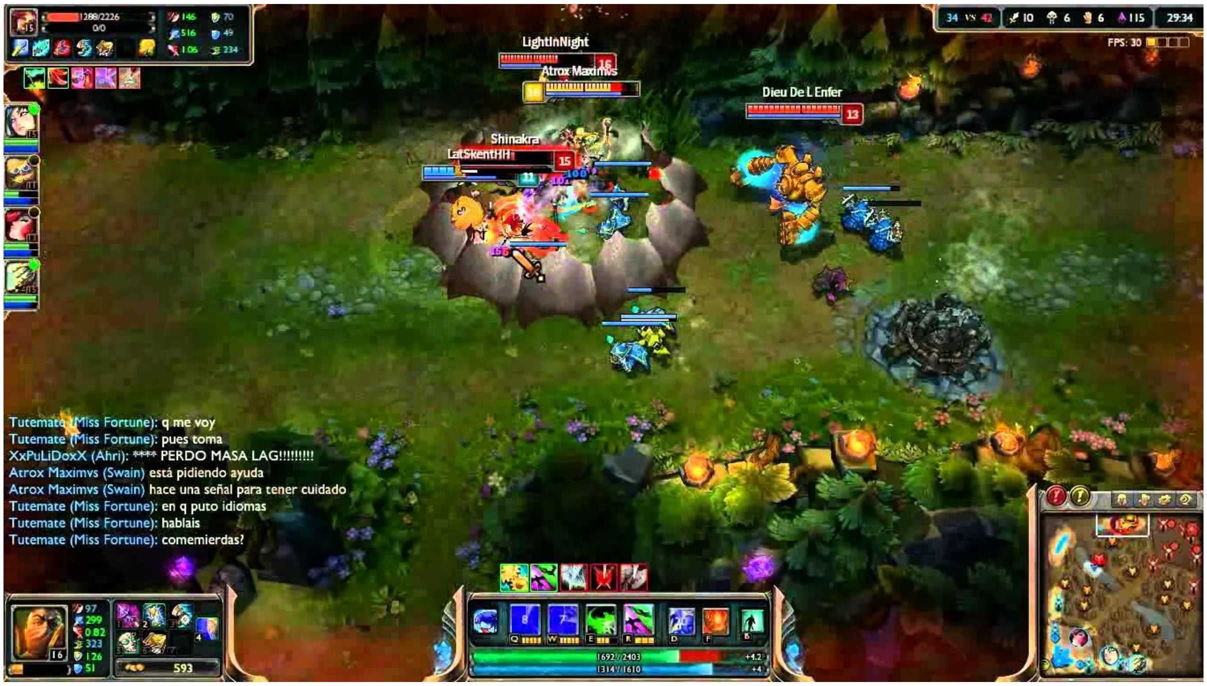
MLG CTF Midwar on Midship

Quick Options:
Number of Rounds: 1 Round
Round Time Limit: 30 Minutes
Flag Touch Return: On
Flag Reset Time: 15 Seconds
Flag At Home To Score: On

Figure 11 Halo 2's pre-game lobby: The Virtual Couch (Microsoft 2015)

The Halo series did a good job of creating a quality action arena experience. We plan to expand upon that game feel and incorporate strategy elements from other genres to create an altogether new and exciting experience.

League of Legends (Riot Games 2015)- *League of Legends* is a MOBA game set in a fantasy realm, and is one of the most popular games in the world. The player takes control of one of over 100 characters to take down towers, slay champions, and win the game by destroying the enemy nexus. Though the end result of development has not much in common with *League of Legends*, our main similarity to *League of Legends* in our original design is the use of lanes to focus the player’s attention. Having lanes almost acts as a funnel for the action and combat that takes place in a game. As a result, players can see where the action is, or if there is no action, where it should be soon. We think having areas of focus like lanes in our games would keep our players engaged. In addition, *League of Legends* has smaller minions that periodically spawn to give players a sense of game flow between skirmishes with actual players and completing objectives. In *Crystalline's* early iterations, we had drones that were able to be spawned. Though drones were not a great threat to the player, they gave the player something to shoot at or assist on the way to the "front lines" of the current game. Drones also functioned as support to help capture objectives, as minions do in *League of Legends*.



Tutemate (Miss Fortune): q me voy
Tutemate (Miss Fortune): pues toma
XxPuLiDoxX (Ahri): **** PERDO MASA LAG!!!!!!!!!
Atrox Maximvs (Swain) está pidiendo ayuda
Atrox Maximvs (Swain) hace una señal para tener cuidado
Tutemate (Miss Fortune): en q puto idiomas
Tutemate (Miss Fortune): hablais
Tutemate (Miss Fortune): comemierdas?

Figure 12 League of Legends gameplay (Tictac Town n.d.)

The main difference between League of Legends and our original concept is that *League of Legends* is 100% MOBA, whereas *Crystalline* is mostly a First-Person Shooter. In contrast to "Hard MOBAs" like *League of Legends* and DotA, *Crystalline* is primarily a First-Person Shooter, and was going to borrow mechanic from the MOBA genre. Our combat is mostly PVP, instead of a mix of PVP and PVE like MOBA games are. In addition, we are hoping to have a game community that is more positive than that of *League of Legends*, whose community is notoriously toxic. One way we plan to achieve this is through quicker matches (losing isn't so bad if you start another match in 10 minutes).

Natural Selection Series (Unknown Worlds Entertainment 2010)**-** *Natural Selection* is a multi-player, team oriented video game series that combines the FPS genre with the Real Time Strategy (RTS) genre. It is set in a science fiction universe in which a human team fights an alien team for control of resources and territory in large indoor facilities. The gameplay is very asymmetric. Human marines rely highly on various tech and guns, while the alien team relies primarily on melee attacks and bio-evolution. Both teams have one player act as a commander, who is given a top-down view of the map, and tasked with placing buildings, researching upgrades, and supporting the FPS players on the front.

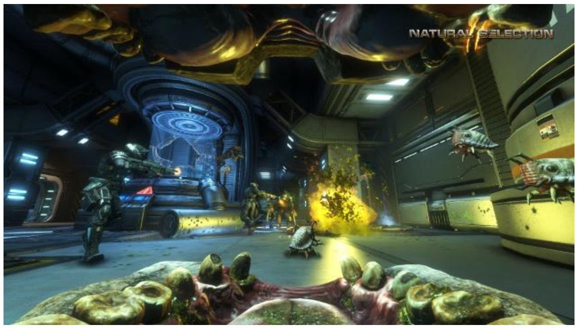


Figure 13 An alien player as a Gorge, the healer class. The unusual alien "mouth cam" can be seen here. (Natural Selection 2 n.d.)

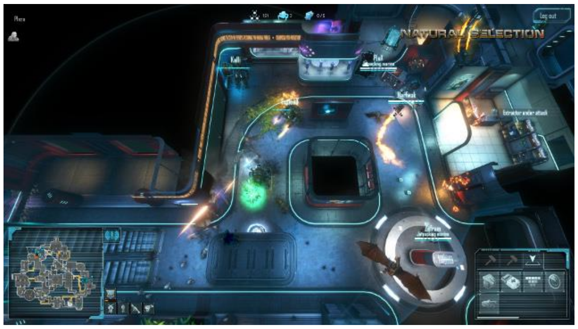


Figure 14 A marine commander viewing an alien attack on the human team's base. (Natural Selection 2 n.d.)

Natural Selection has a very high barrier for entry. There are many things going on in this world. If the player doesn't know everything about it, then they will have a very difficult time playing with other players in this real time environment. With *Crystalline*, we decided to address this problem by having a very low barrier for entry and a system of patterns that are very easily identifiable for new players, yet still offer depth to explore for players who have the time to spend learning the ins and outs of our systems.

Nuclear Dawn (Interwave studios 2011)- *Nuclear Dawn* is a post-apocalyptic FPS-RTS hybrid. Gameplay consists of one player on each team acting as a commander, who directs the team from an overhead view. The other players are foot soldiers with standard FPS player capabilities. While an interesting and novel concept, *Nuclear Dawn* simply merged the two genres into one game, 50 / 50. But not nailing the gameplay merge between these two similar, but radically different genres *Nuclear Dawn* suffers from balance issues and stagnant gameplay.

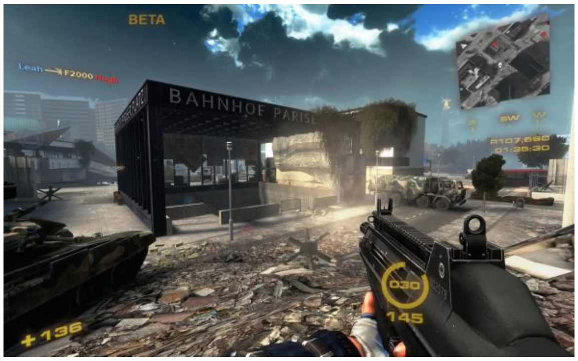


Figure 15 The FPS view on the Gate Map (Interwave studios 2011)

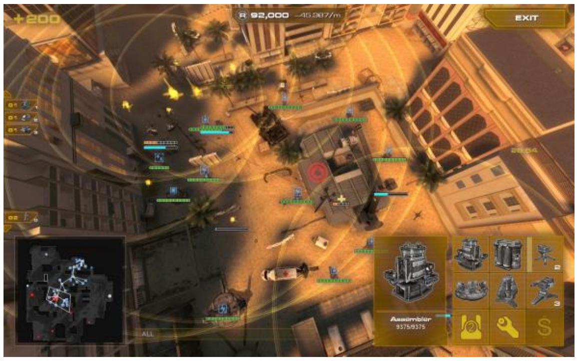


Figure 16 The commander view on the Oasis map (Interwave studios 2011)

With *Crystalline*, we are creating an FPS that has elements of strategy games that fit and complement each other. We are being very selective in what elements we bring to the FPS genre, in an effort to avoid the pitfalls that *Nuclear Dawn* fell into.

Star Wars: Battlefront Series (LucasArts 2005)**-** *Battlefront* is a game from LucasArts putting the players into the iconic battles of the Star Wars universe. The player chooses one of several classes of troopers or droids to play as and then spawns at a node belonging to the player’s team. Then, along with a small army of AI units, the player moves to an uncaptured or enemy node and stands near it to capture it. Over time, each side takes casualties which reduces the number of reinforcements available per team. A team wins by either capturing all the nodes and holding them for a time, or by reducing the enemy team’s reinforcement count to 0. In the battlefield there are also points where the engineering class of each faction could build turrets or repair droids that provide health and ammo.

Figure 17 Large scale warfare in Star Wars: Battlefront II (LucasArts 2005)

What we like about Battlefront and Battlefront 2 is the feeling of being in a large scale battle. This is mainly accomplished by having a large reinforcement count that implies a larger war going on than just the currently spawned units. By capturing nodes, the player can feel like they are making a meaningful contributing to this larger war on their own. Players also have roles to play in the battlefield through the distinctive class system, where they can either be a generic trooper, a sniper, an engineer, a heavy trooper with a rocket launcher for destroying vehicles, or a commando with a unique ability. Despite these positives, there is not much more there beyond the capturing of outposts, and the game also suffered from the fact that once a player's side starts losing, it is difficult to make a comeback.

The feeling of playing a part in a larger battle is an important element that we want the player to experience in our game. We use a similar concept of node capturing so that a 'line' of battle forms as teams contest the various points. That way players can feel like they are making that meaningful contribution when they help to push the line and capture the next node. We originally planned to have drones to supplement the players like the allied troops in Battlefront. We also plan to visually create the effect of a larger war by having combat effects in the 3D skybox beyond the playable space.

Titanfall (Respawn Entertainment 2014)- Respawn's flagship sci-fi multiplayer First-Person Shooter, set on a variety of colonized planets in the far future. Players make use of futuristic technology such as jetpacks and giant mechs in order to gain an edge over their enemies. Gameplay is fast-paced and relies heavily on using your weapons to your advantage. One of the main features of Titanfall are the giant mechs, which players are able to pilot. Using your mechs effectively is the key to victory in *Titanfall*.

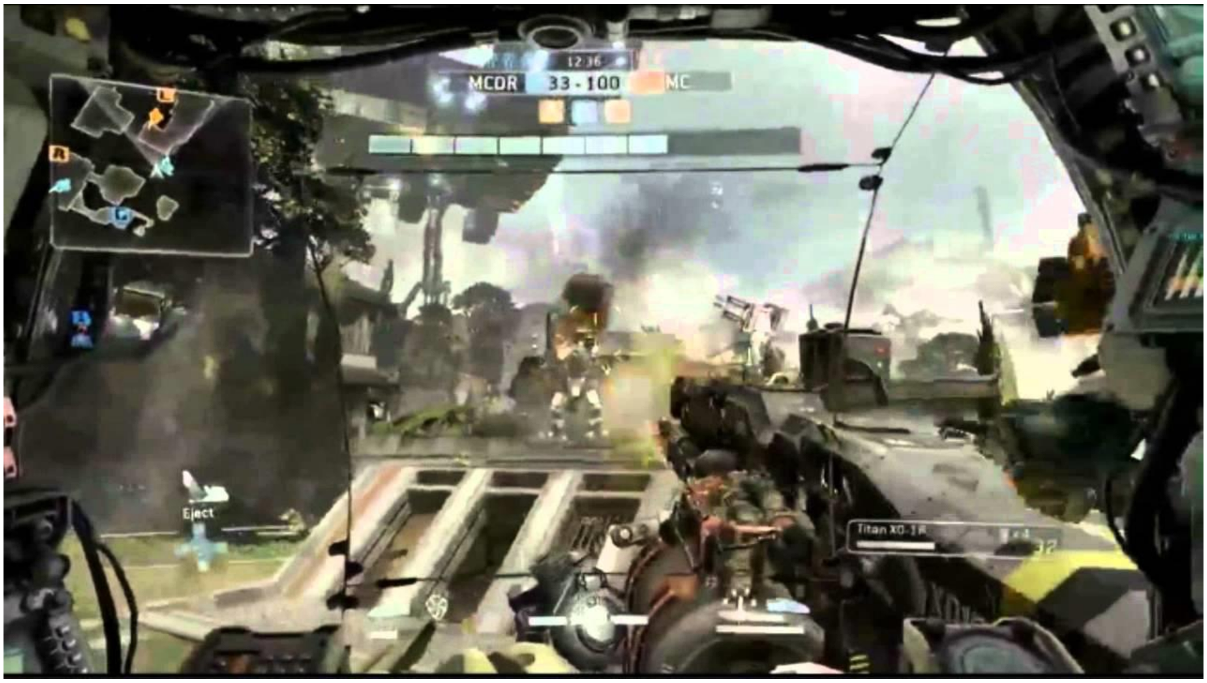


Figure 18 Titanfall gameplay (Respawn Entertainment 2014)

One of the things that we wanted to bring in from Titanfall was the feel of fast-paced multiplayer gameplay with strategy elements involved. Titanfall's gameplay mechanics and weapons are highly conducive to frantically-paced running and shooting. Even some of Titanfall's items like the lock-on Smart Pistol do most of the aiming work for you, so focus is put more on the wall-running and parkour elements of the game. We are currently striving to have *Crystalline's* combat pacing match that of Titanfall's. One thing about Titanfall that we don't want to mirror is the lack of strategy elements. One notable thing about Titanfall is that it is an almost purely FPS experience. Even the mech pilot's interface mirrors that of the standard soldier's. As a result, the game is less about thinking and more about positioning and fast reflexes. Though it would be nice to have those elements in *Crystalline*, we also want *Crystalline* to be an exercise in strategy as well. We believe we can achieve this through having victory being tightly coupled with completing objectives.

Tribes Series (Hi-Rez Studios 2012)**-** Beginning with Dynamix's *Tribes* and *Tribes 2* and later *Tribes Ascend* when the franchise was picked up by Hi-Rez Studios, the Tribes franchise is fast-paced sci-fi FPS that sets itself apart by featuring large, open landscapes rather than the tight maps of traditional games in the genre. This is made possible due to each player's jet pack and the ability to ski. Skiing is a mechanic by which you negate friction and move along the terrain quickly, picking up speed as you fall and using your jetpack to get up slopes without losing any momentum. This allows skilled players to gain surprising amounts of speed and cover whole maps even in heavy armor. Each team also has a base with essential equipment, such as inventory stations that allow the players to get weapons and tools they don't start with, and defensive turrets. This makes base defense and repair an important part of the gameplay experience. Typical games are capture the flag matches for up to 32 players, and players get points for kills, capturing the flag, and repairing and defending the base.

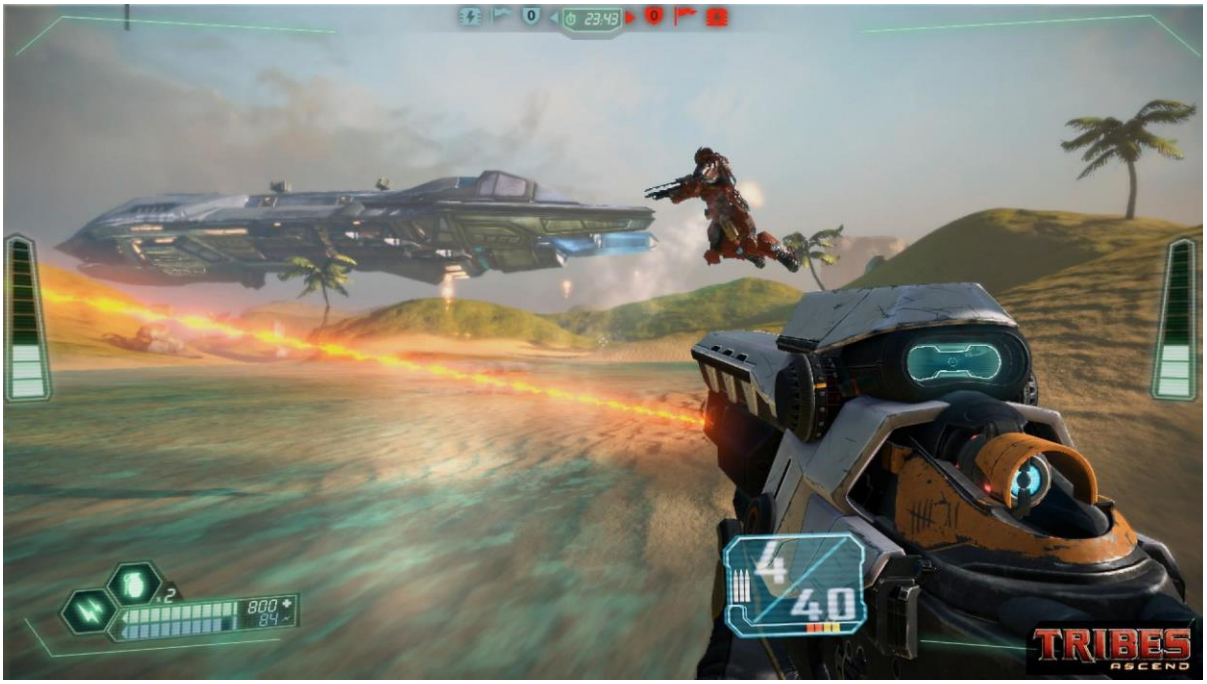


Figure 19 Tribes Ascend Gameplay (Hi-Rez Studios 2012)

The mechanics we like most about *Tribes* are the skiing which literally makes the gameplay fast-paced, and the emphasis on base defense and equipment interactivity. The downside of *Tribes* is that it can get repetitive. Each player has access to the same things at all times, with the exception of the last game in the series, *Tribes Ascend*, which was free-to-play and had unlockable weapons and equipment that could be obtained by paying or by getting experience in the game.

We wanted to take the emphasis on base equipment and make it possible to build things in our game. In earlier iterations of *Crystalline*, the outposts represented small bases, and each had a small number of slots on which equipment can be built. Our game is unique from the *Tribes* series in that it also originally included a resource management system in the form of crystals, which were currency for building upgrades. Our weapon system also allows players to modify their weapons in the field by slotting different crystals in them, and use unique abilities depending on which crystal is slotted in.

2.2.2 Our Game's Place in the Market

While the FPS genre may be saturated, it is not so much so that a niche can't be found. Canonical games (*Halo, Call of Duty, and Battlefield*) typically don't stray too far from the norm in their design for the sake of safety, or when they shift seriously (*Halo 2*) they redesign it. While there are a lot of FPS's that straddle genre lines (*Borderlands*, *Battleborn*) they typically aren't very accessible to new players. Over specificity of skill trees and builds can sometimes be daunting for new players (*League of Legends*) scaring them away before they can even get immersed in the game.

A brief assessment of the market yields that accessibility is an area that FPS games tend to ignore, typically targeting a hardcore crowd, the stereotypical "gamer". Players are more diverse than that, leaving a well of untapped potential that *Crystalline* intends to address. Every design decision is intended to produce an experience that welcomes new players with open arms. Where *Borderlands* has a near infinite weapon sandbox, *Crystalline* has a compact system in which the player need only learn three power ups to access a litany of options. Where some games seek depth through complexity and number of choices (*League of Legends*) *Crystalline* targets depth through the nuances of a basic power up system.

Crystalline's success lies in the fact that it doesn't attempt to be a combination of different genres, becoming too bloated to easily comprehend. Rather, *Crystalline* recognizes that mechanics should complement the overall aesthetic of a game.

3. Background Research

3.1 Game Design Research

3.1.1 Level Design (Nicholas Buonarota)

3.1.1.1 Introductions and Definitions

Level design is the “craft”, “procedure”, or “process” of creating a playspace that revolves around the set of rules in your game. Some say it is, “The Art of Level Design” others say, “The science of Level Design”. It’s a little bit of both. At some points along the process its more one than the other. But in the end just like game design, level design is both an art and a science.

Level design is essentially secondary mechanic design. The real purpose of level design is to highlight the main mechanics in your game. Let’s look at an example in context. The main mechanic in a First Person Shooter is Player 1 shoots Player 2. The level design in the game should highlight that. The level designer needs to create a space that enables Player 1 shooting Player 2. If Player 1 has a sniper rifle, then create a space where they can use a sniper rifle effectively. If Player 1 has a shotgun, then create a space where they can use a shotgun effectively. For example, tight corners, small rooms, one / two way entrances.

If the game in question is a platformer named *Super Mario Brothers* (*Nintendo 1985*), then the main mechanic of the game is “Mario jumps from platform to platform”. The level design should highlight that. The level designer needs to create a space that enables Mario to jump from platform to platform. In this case, the level designer needs to think about jump distances. For example, near platforms will make your level easy, platforms on the borderline of your character’s jump distance will make your level hard, and platforms beyond your character’s jump distance will make your level impossible. Are there obstacles in the way of jumping to the next platform? If Player 1 has a double jump power, then create a space where they can use that ability to maybe reach a secret area. These are all examples of questions the level designer can ask themselves during the creation process.

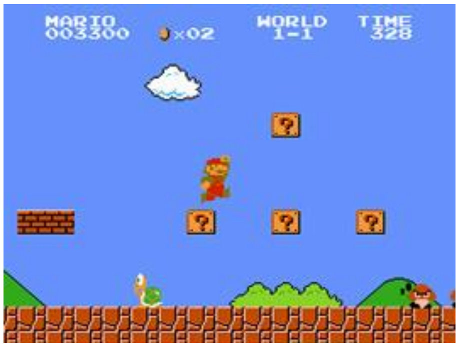


Figure 20 Mario punching and kicking the sky (Nintendo 1985)

That’s it! That’s everything you need to know about level design. You can leave now and stop reading. But if you want a better understanding of level design theory, processes, workflows, and more context examples, then stick around, there is more great stuff on the way!

Rudolf Kremers has written one of the best books on level design that I have ever read. In his introduction he goes on to say that level design is an all-encompassing field and there are few jobs in the games industry as satisfying or important as that of a level designer. It is however a relatively new official position in the industry and is often misunderstood or misrepresented, which is regrettable.

Level design is exciting and important. Good or bad level design can make or break a game, so it is unfortunate and is surprising how little reference material and documentation there is for people who wish to get into level design and have a deeper understanding of the field. Most of what is out there is community written wiki articles on the websites of mod communities, and more recently a slim few Game Developers Conference (GDC) presentations. These GDC presentations are categorized under “Level Design in a Day”. LDD has been a regular series of presentations for the last few years at GDC. It is a day-long event, usually on the Tuesday of GDC from 9-4PM. It gathers a bunch of people who present on level design and level design related topics. The lunch session is also converted to a portfolio review session, which is helpful for those who wish to get critique from others.

Because of these things, most people who work as level designers are self-taught or have learned via apprenticeship on the job. This is less than ideal. If only there were books & books of information on this subject. Well, that was one of the goals of Kremers' book. He wishes to bring some of that knowledge to the public. On a personal note, that is also one of my goals. Over the years I have been collecting information, such as design documents, presentations, and post mortems. I've been taking notes, collecting resources, and writing tutorials on these subjects. As I've been collecting this raw information, I've been processing it, trying to make it easier to understand with a low barrier for entry in order to release it to the community and public.

"Bad level design can ruin a good game." and "A bad game cannot be saved by good level design." Before one can understand the function of level design, one must understand the function of game design. Unfortunately, designers don't always agree on what game design is, but that's okay because design is just a collection of opinions anyway. Some people agree more on certain things than other things, but in the end it's all just opinions.

One of the most famous definitions of game design is from Sid Meier. He says, "A game is a series of interesting choices." Further explanation of "interesting" often includes the following qualifiers.

- No choice should be consistently better than the others. (Or it would make the other choices uninteresting or redundant.)
- The choices shouldn't be the same. (It becomes meaningless to differentiate between choices.)
- Choices must be informed. (Lest they become arbitrary or random.)

Yet of course there are games that do away with interesting choices. Example, DDR, "Hit the right button at the right time!"

This isn't the time to go into detail, but a good designer needs to be familiar with both Ludologist and Narrativist viewpoints. Some designers focus more on one than the other. Neither is "right", but a good designer needs to be familiar with both.

Good designers are also aware of the external goals of their game. Good designers look at matters of function and purpose. If the designer knows what game design is supposed to achieve, then they should be able to reverse engineer and have ideas of how game design might function to achieve that end purpose.

Case study: A simple chair.

- A chair's design is subject to many things, but the main identifiable goal is to allow a person to sit on it.

This goal leads to other related requirements that describe what the chair has to be:

- strong enough to support the weight of most people.
- stable
- affordable (in the sense of cost)
- moveable
- aesthetically pleasing

At this point, a designer takes these goals and requirements and turns out some functional designs. So the first step in game design is to identify the game's external goals and interpret them so that they are represented in the rules.

Now let's apply the same case study to games:

Game: goal is to provide a fun and profitable gameplay experience.

This leads to other requirements on what the game has to be:

- pretty
- easy to learn
- hard to master
- of sufficient quality
- showcase high production values

However, it is important not to confuse these goals with the game's internal or intrinsic goals.

Before more details can be given, some more terms need to be defined. Everyone has their own definition of what a "game" is. According to Kremers, a game is defined as, "An often predetermined, agreed-upon set of rules, which are designed to facilitate gameplay. The motivation behind the creation of a game itself can be diverse, for example including commercial, educational, artistic, or other elements.”.

From that we can infer what it means to be a game designer, aka, someone who practices game design. According to Kremers, “A game design is a coherent set of rules that formalizes a game’s content in such a way that it facilitates appropriate gameplay, in order to achieve the game’s fundamental goals.” In layman’s terms, “creating a set of rules that guides the act of play toward a particular goal.”

So if that is game design, then what is level design? A good designer asks this question and looks to game design’s definition as a starting point. As stated before, after reading papers, watching presentations, and experiencing the workflow, it has become my opinion that level design is the craft, procedure, and process of creating a playspace that revolves around the set of rules in your game.

Why is level design important to game design? Well, it is important that your playspace revolve around your rules and mechanics of your game because the rules is what guides your players toward your “game vision”. Level design is a supporting role. Your playspace needs to highlight your mechanics. Your playspace needs to support rules and convince the player that these are the best freaking rules in the history of playing games.

A good designer should note the similarities here. Level design has been around for a very long time. In a historical context, we can find good level design in sports, board-game layouts, pinball machines, and Dungeons and Dragons. There is probably a good homework assignment in here for someone teaching a class on level design.

Here are some pretty famous game designers’ opinions on level design:

- Jay Wilbur: “making sure the player is taught the rules of play”
- Sam Sharami: “responsible for the implementation of gameplay in a title.”
- John Romero: “maps are where the game takes place.”

As a summary of the views so far, level design takes all the disparate elements of the game and make them gel. It is responsible for the implementation of a good game. The level is where the game takes place. These opinions combined with historic examples provide us with a good foundation of what level design truly is.

As a summary of the function of level design so far, video games are still in a state of infancy. There are a lot of opinions on what is “law” and “right”. A lot of designers say different things. A good designer should back up their opinions with data and examples. They need to help answer the question of “why?”. Together all these opinions have value. These designers are people with passion about these ideas. Each designer’s opinion has a single thread of truth usually somewhere in their argument. We can all learn and grow from each other’s differentiating opinions.

As a summary of game design compared to level design so far, it is important to note that level design is not game design, but they are co-dependant and interrelated. A designer needs to understand both to be good at level design. After all, how can a designer interpret game design without knowledge of it? And conversely, how can we define rules for an experience we don’t understand?

A game designer designs the gameplay rules. The level designer designs how the player is confronted with those rules. A game designer formulates the rules, while the level designer interprets them for maximum results. A good level designer will always produce maximum results. To a certain degree one represents theory, while the other represents practice.

Just as a theatrical play needs a performance to be complete, a video game’s rules need gameplay to occur to be complete. The basic purpose of level design is to interpret the rules, and to translate them into a construct, a level, that best facilitates play. According to Kremers, level design is applied game design, not a separate function subordinate in a game design hierarchy, but as a description of its main function and purpose. To understand certain level design issues, we have to understand certain game design issues. They are different sides of the same coin. Again, comparing game design and level design to our theatrical metaphor from before, is a written play superior to its theatrical performance? The answer is no. Neither is better than the other, especially not alone. Only together can one element rise to its full potential.

These observations, bar-room discussions, and debates of what is and is not level design and what its function is are not unimportant. If we don't understand the nature of our work, we cannot defend our work. We can't iterate on the process and definition to improve upon it. It is important to understand how we got here. We need to be able to explain it to others, as much as to explain it to ourselves, why we made these choices in the first place. We need to know the function of our work is, what our responsibilities are so we can make clear defensible choices from the get go.

3.1.1.2 Teaching Mechanisms

"A videogame is an artificial construct that, when well made, rewards the player with fun." ~ Rudolf Kremers. Much of traditional play is all about teaching "skills" and "testing" the player's proficiency. It is rewarding to master a task. It is even more rewarding to show to others that you have mastered a task and then receive their praise and admiration. Games teach skills. Gameplay allows the players to put these skills to the test in a controlled setting. A good game strikes a balance between teaching skills and providing enjoyable "testbeds" to try out the new skills.

"Good level design teaches the player how to play and enjoy the game." ~ Ed Byrne. Good level design is not just teaching the player the rules of the game, but also allowing the player to use those rules in a way that is rewarding and fun. Much of the fun comes to the surface when the player is tested.

Teaching mechanisms are meaningless unless that which is taught is tested and put into practice. In other words, there is no point in learning new skills if you never use these skills. Testing is a part of teaching. It can be assumed that when we speak of teaching mechanisms, we also speak of testing mechanics.

3.1.1.3 Modular Design Processes

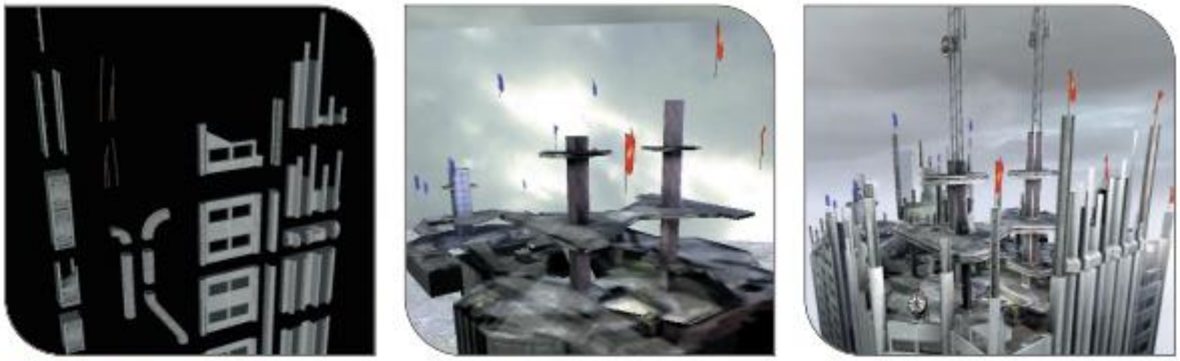


FIGURE 4A-4C (LEFT TO RIGHT). A relatively small selection of instanced prefabs (left) can turn a basic level (center) into a far more complex world (right).

Figure 21 Figure from Lee Perry's article on modular design (Perry 2002)

Introduction

Lee Perry, the lead level designer on Epic’s Gears of War, wrote an article for “Game Developer Magazine’ that explains how we go about creating complex worlds from modular static mesh pieces, aka modular level design. Although the article was written about Unreal Engine 2, Unreal’s approach has remained very much the same for UE3 and UE4. The approach of modular level development is even more valid now, given the increased complexity of each mesh.

Necessity is the mother of invention. The modular level design approach arose from the need to have great-looking, high detailed levels, environments, and worlds without having to build and texture every nook and cranny of the environment. Audiences have high expectations. Environment creation has become much more complex since the days of Doom and Quake. Players expect more detail in the geometry now. Flat walls with textures don’t but it anymore. The standard now is more 3D geometry that “pops”.

Sometimes, before we can improve our practice, we need to study our past. In the past there was rarely an exclusive “level designer”. Level designer was usually a sub role that another team member also handled in addition to their other hats. For example, there were larger roles on the project that would be likely to also grab the level design role. Some specific examples include the designer, who focused on the design and fun of the level, the artist, who focused on making the level pretty, and the programmer, who understood the engine and maximized efficiency on their levels.

For the purposes of not having this section go on for 30 pages on its own I have cut the in depth analysis of these roles, their positive and negative impacts on a team, and how they aren't a viable option in a modern game development production cycle. But should you have an interest in reading about these topics, I do have additional slideshows and notes in my personal google drive folder that I would be happy to share with you. Please email me at nickbuonarota@gmail.com, with the subject line pertaining to this topic, and I will continue the conversation with you.

Modular Design

Studying the past you can see why we can to where we are today. A common practice of many studios is modular design. Modular design is a better solution that uses prefabricated modular component based geometry in creative ways to achieve level of detail that players expect to see in their game worlds.

An efficient modular set should be reusing high detailed components in new creative combinations, saving development time, and have a focused base development that doesn't create wasted assets. Other secondary components, that make a superb modular set, include having a pre-planned scale, utilizing grids, lots of planning, starting with the basics, utilizing the concepts of "mirror-mirror", selecting proper object origins, not forgetting to accessorize, compartmentalizing your accessories, selecting specific custom pieces, and creating objects with the end goal of assembly in mind. Again, these terms and definitions deserve explanation of their own, so if you are curious on the topic and not satisfied with the conclusion summary below, then please feel free to email me.

In Conclusion

Even if the modular construction doesn't seem like something your project can utilize, there are a few key benefits to consider before making that decision. First and First and most importantly. After thoughts are rampant and a fact of life in the game industry. How many times have you heard, "Oh wouldn't it be cool if…" Working with prefabs offers a lot of freedom to go back and edit an original piece, while having every instance of that piece automatically update throughout the entire world. For example, if a designer doesn't like that texture on that monitor, then they can implement it for now and go back and change it later.

Sometimes entire pieces of geometry can change as long as the new piece fits the same function and grid space. Working with "Lego-like" chunks of geometry allows for easy re-working on the designers end without having to go back and consult with artists. Are testers telling you that you need an additional exit in that room? No problem in modular land!

There is also a technical advantage of using a modular set. This is for the hard core programmers in the crowd. The "We do not believe it is good enough" people. In modular level design, you can load the components into memory and work with many instances of the same object. Instead of 100's of custom pieces, you have 10 pieces used over and over and over. Even the densest-detail environment can get away with only using a handful of meshes in memory.

Modular level design also presents an opportunity for maintaining consistency. With bigger teams and multiple level designers and artists, consistency becomes an issue. By using a prefab set, even a fast sprawling, robust level will maintain a consistent style throughout. A feat much harder to do when everything is custom geometry.

Another benefit of modular design is that, modular design is very compatible with LOD scaling. As you can swap out components for lower LOD components if a user's computer isn't that powerful.

On paper, this workflow can maximize skill efficiency. Modularity allows team members to concentrate on doing what they are best at. Level designers with strong gameplay skills needn't worry about creating the loads of detail required in high-end graphics. With a modular construction set, they focus on laying out the game and not get bogged down trying to create what should be considered art assets. Let the artists do that!

Finally, modularity also contributes to community and shelf life of a game. Working in a modular setting with prefabs lays the groundwork for low barrier for entry level editors for your community should you decide to make one. For example: Elder Scrolls Series, Halo's Forge, Age of Empires, Starcraft, et, et. These titles that have used modular / tile based techniques have made their editability a key selling point. Editability helps extend the shelf-life and creates a stronger user community.

With all the great things, it should also be noted that there are potential limitations and drawbacks. Sometimes developers see the words "modular" and "component-based" and they panic. Old school graph-paper Dungeons & Dragons levels and Dungeons come to mind. Ninety degree hallway corners, and evenly spaced doorways creep into their nightmares. The bottom line is that modular level design is indeed more restrictive than having a staff of dozens create an entire level as custom geometry and unique art work. But for most teams, that kind of content creation isn't feasible. It's important to find the right level of modularity for your project and your team.

Still some designers find it difficult to work with geometry they didn't create. This is a common scenario for designers who came up through the trenches and upbringing of where they had to do it all. It's part of what made them a valuable asset to the team. This modular construction makes them feel threatened. But soon they'll realize that being able to see one's gameplay ideas realized quickly is much more valuable.

Future of Modular Design

As a plan for your team for further down the road, modular practices early on can help future version of your product. Content can easily be ported to a procedural world generating algorithm for randomized dungeons and worlds. Possible for content to have future potential. Sequels of Series, DLC, Expansion Packs. Examples: Old school Disney animations; Modern Pixar movies. This technique has been around in other industries for a while. It's not just a fad. It's useful.

3.1.1.4 First Person Shooter Level Design

Level design for an FPS isn't that different from level design in general if you follow the same idea that you create your playspace around your game's rules. But there are some things that a designer can be aware of specifically for first person shooters to give them an upper hand and make the difference between an A level and an A+ level.

Design Theory: Terms - Areas

"Areas" for lack of a better term are important to your map. Having standard terms is useful for describing ideas to others in the field or on your team. Here are some Left 4 Dead specific area definitions, "Close Quarters, Flow, Narrow Flow, Wide Flow, Capillaries, Masking, Sanitizing, Tube, King of the Hill, Fish in a Barrel, One-Way, Return, Holdout."

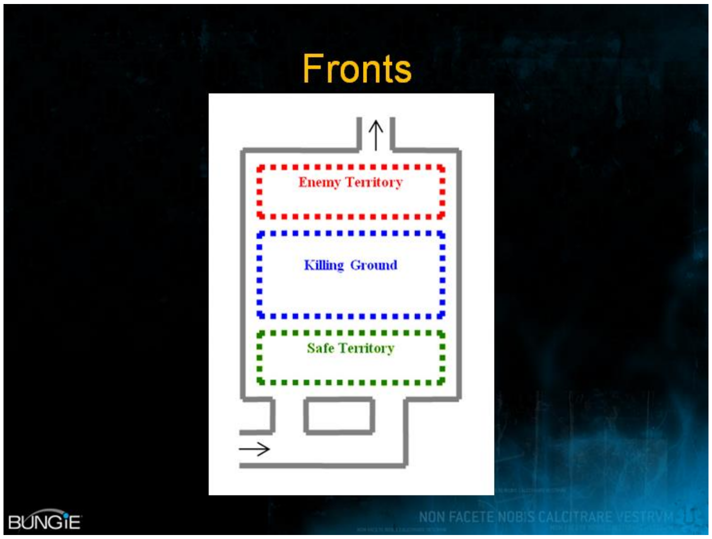
Close Quarters for example, is defined as a series of small combat spaces where the survivors must deal with the infected at a close proximity. Typically, the infected can get within 50 feet of the survivors before they are vulnerable. For example, the apartment building at the beginning of No Mercy campaign in Left 4 Dead is a CQB space. (Close Quarters Battle)

In this space, zombies have the advantage of appearing at close range. Zombies don't have ranged attacks. Usually survivors have the ability to shoot before zombies get close and can attack. CQB takes that advantage away from players. Zombies and players come around tight corners and have the possibility of quickly becoming overwhelmed. Although the infected have the advantage of appearing at close range, the survivors don't have too many entrances to cover and protect themselves. Some weapons that are useful here are melee and close range weapons.

All the L4D terms are available as a part of the L4Dictionary on the source SDK wiki here: "https://developer.valvesoftware.com/wiki/Left_4_Dictionary" A good designer should analyze these spaces. They aren't L4D specific. The concepts can be generalized to other FPSs and even other genres. You should utilize this knowledge of these areas in your maps.

Design Theory: Terms - Fronts

Designers at Bungie like to split up their campaign encounters into areas with fronts. They define fronts as, "the line in the battlefield between the player and his enemies. It is created by the paths and cover in the space." The fronts themselves are divided into three parts, a safe territory, a killing ground, and an enemy territory.



Fronts

Enemy Territory

Killing Ground

Safe Territory

NON FACETE NOBIS CALCITRARE VESTRVM

Figure 22 Bungie's "fronts"

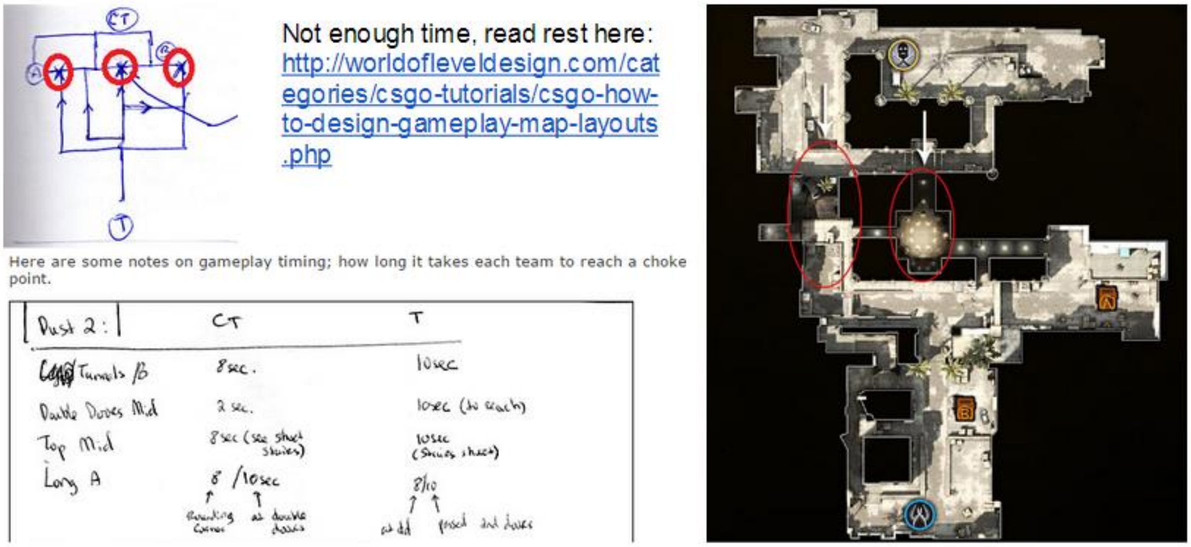
Safe territory is a space that the player can safely retreat to if he starts to get overwhelmed. Bungie designers believe that there should always be sufficient cover in this area for the player to recharge their shields (a mechanic of Halo). This should also be where the player enters the space nearly all of the time.

On the complete other side of the encounter space from safe territory is enemy territory. Here the enemy AI has total control and it is very dangerous for the player. This is nearly always where the exit of the space is found.

Most of the combat happens in the Killing Ground. This is the area in the middle. Bungie designers believe that sight lines in this area should be clear enough that the player can see the battle playing out and changing. For example, if the AI retreats then the player should be able to tell.

In the example that Bungie gives, the terms are specifically given in the context of a single player campaign. But as good designers we need to break down this design and look for similarities across game types and genres. There are a few articles on “worldofleveldesign.com” that focus on “choke point design” in the Counter-Strike series. If you read the article, then you’ll notice that the design seems very similar to the “fronts” of the Halo 3 presentation. This middle meet up zone, aka “choke point” as Alex of WoLD says, or “killing ground” as Marc Zak designer at bungie says, is an area of the map that sees the most action. It is where it all goes down. It is where Player 1 meets Player 2.

Another large point in the “choke point” articles is made on timing. Because Counter-Strike is multiplayer and not single player, designers can’t hold the player’s hand as much and therefor it's harder to gauge pacing of when players will have certain encounters. Here, designers rely a lot on timing, or how long it takes a player to arrive at a point with no obstacles presented to them. This technique is useful in determining where/when opposing teams will meet up, and where your multiplayer map’s “choke point” or “killing ground” is.



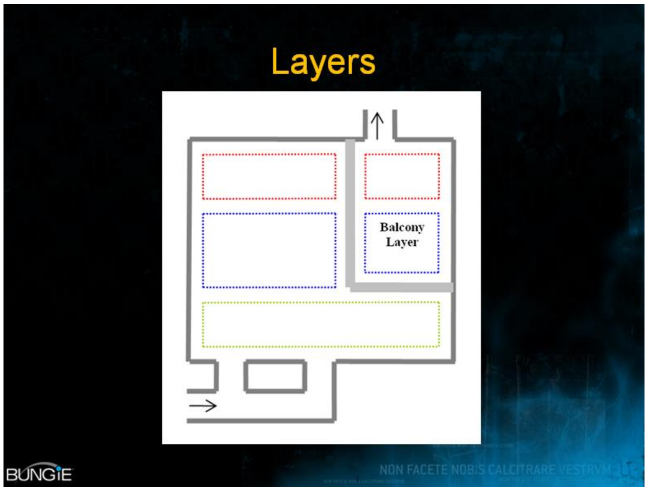
Not enough time, read rest here: http://worldofleveldesign.com/categories/csgo-tutorials/csgo-how-to-design-gameplay-map-layouts.php

Here are some notes on gameplay timing; how long it takes each team to reach a choke point.

Dust 2:	CT	T
~~Cage~~ Tunnels /B	8sec.	10sec
Double Doors Mid	2 sec.	10sec (to reach)
Top Mid	8sec (see short stairs)	10sec (Stairs short)
Long A	8 /10sec ↑ Rounding Corner ↑ at double doors	8/10 ↑ at dd ↑ [illegible] and doors

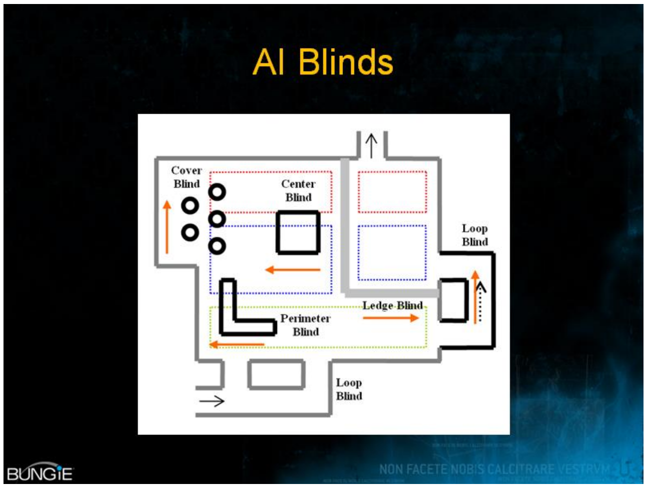
Figure 23 Counter-Strike's "choke points"

Another thing about these areas is that they need to be populated by both enemies and geometry. Other terminology used for populating these spaces include “layers” and “blinds”. Layers are areas within a space that can’t be moved between trivially. These layers are used together during combat. They are not considered separate spaces. Combat can occur between layers, but the player will use different tactics depending on which layer he is in. Some example layers in halo maps are balcony layer, chasm layer, etc.



Layers

Figure 24 Bungie's "layers"



AI Blinds

Figure 25 Bungie's "AI Blinds"

Blinds are environment features that allow the player to move a short distance without the AI being able to see them. In Halo, they are used to hide and recover shields after taking too much damage, force the AI to search for you, and attack an enemy from an unexpected direction. Bungie believes that every space should have AI blinds. Blinds are best when they allow the player to switch between two territories, for example, retreating from the Killing Ground into Safe Territory, or between two layers. Some examples of different types of blinds include perimeter blind, loop blind, center blind, ledge/hill blind, and cover blind.

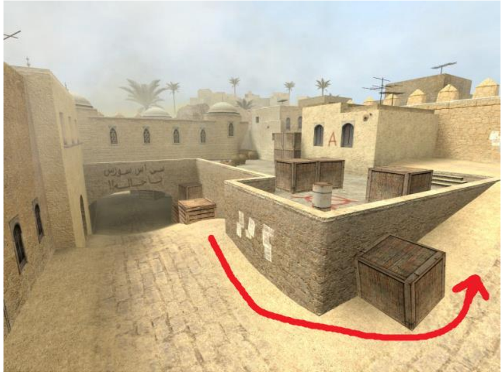
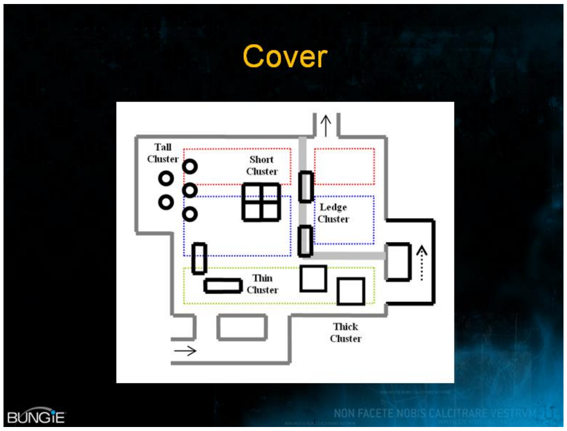


Figure 26 Counter-Strike's Dust 2 using a "Ledge Blind" in addition to having 2 layers

Bungie also likes to give terminology to their cover types. Cover is a staple of the FPS genre. Having different types of cover help give shape, texture, and flair to a fight. Most of the time cover in a space should be grouped. These groups will naturally be useful as Safe Territory or Enemy Territory, and the spaces between them will make good killing grounds. Examples of common terms for the cover clusters include tall cover cluster, short cover cluster, ledge/hilltop cluster, thick cluster, and thin clusters.



Cover

Figure 27 Bungie's "cover clusters"

Lastly an FPS playspace should also support and complement movement. If your FPS supports a jump ability, then you can take some established tricks from 3D platformers and apply them to your map. Types of movement that Designers at Bungie identified for Halo 3 include Player Shortcuts, One-way Paths, Ninja Paths, and Vehicle Flow.

Player Shortcut Paths are routes that the AI cannot normally take, but the player can. They usually take less than 5 seconds to traverse, and they are best when they connect two territory-types or two layers. They are also often AI blinds. Every once in a while, the AI will use these paths through hints or scripting, which makes the AI seem more lifelike and intimidating. One-Way Paths are paths that require the player to commit to a course of action and don’t allow him to retreat. They very often lead directly into Enemy Territory and can lead to a successful flanking…or a quick death. The player must jump down from a ledge or balcony and cannot jump back up to escape. Ninja Paths are just fun. They make the player feel like a ninja. Usually they do not shorten the time it takes to get somewhere (often they take longer than a direct path) but they are just cooler. Sometimes they allow the Player to attack from an unexpected direction, as well. Jump chains, crouch-jumps. Lastly if your game supports vehicles, like Halo 3, you need to be aware of how these vehicles move through the environment. Vehicles usually have different sizes and speeds than an infantry unit does. If you expect to have both infantry and vehicles in the same combat space, then you need to think about those different needs and create a space that works to both ends.

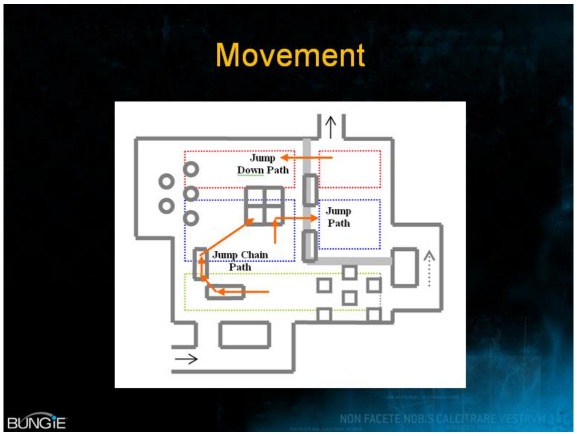


Figure 28 Bungie's "movement" diagram

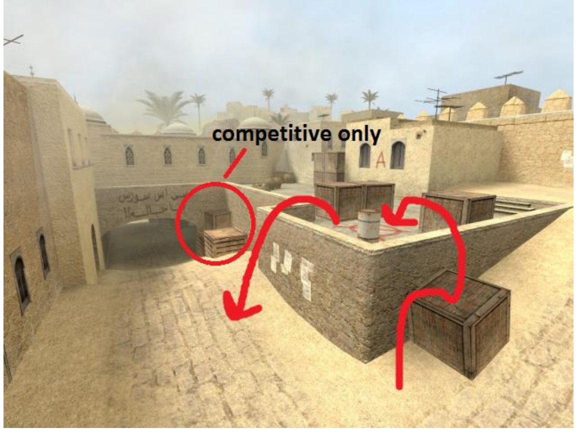


Figure 29 Dust 2's Bombsite A showing movement terminology in action

In conclusion, the terminology can be often broken down into general cases that can be applied to different game modes and even sometimes different genres. Level design is just “applied game design" a good designer will remember that everyone and everything is your teacher, and that you can often learn something in a place that you may not expect.

3.2 Technical Research

3.2.1 Networking Architecture Research (Alexander Herdzik)

Since we essentially wanted to make an FPS game since our group was created, a lot of the networking research was done with this in mind. The first place that we went to was the GDC vault. Though there were not many results by searching for "networking", we found one talk that was immensely useful for us, especially in regards to tips for creating a functional, lightweight networking architecture. The talk "I Shot You First: Networking the Gameplay of HALO: REACH" by David Aldridge gave a lot of insight into the approaches of AAA developers for writing the networking for FPS games. One main takeaway from this talk was the idea of each object that needs to be networked having a priority number. Data from high-priority objects gets sent over the network more often than low-priority objects. However, the idea is that all objects get synced up and gets their data sent eventually. As a result, the networking architecture becomes "scalable". Players with lower bandwidth will have a smaller amount of data sent to and from their machines, but they will get all data eventually. The difference between high-bandwidth and low-bandwidth players is that low-bandwidth players will receive mostly high-priority object data, where as a player with almost unlimited bandwidth will receive all data regardless of bandwidth equally. The talk also goes through common simplified approaches to networking, such as deterministic networking, the TCP protocol, and the Quake method of constantly sending all game data in one blob. Finally, the talk goes into detail regarding the networking protocols of Halo: Reach, and ways to integrate these protocols into your own game. This part was especially useful regarding ideas on how to structure our networking architecture. If we were building our own engine, we would likely base our networking architecture around a simplified version of how it is done in Halo: Reach.

One problem with the Halo: Reach networking architecture is the fact that networking is done through clients acting as servers, with no central server involved. If we needed to host a server, we would take advantage of server space given to us through Microsoft Azure, and as a result, we looked at client-server architectures used in FPS games. The main example that we found was the Source Multiplayer Architecture, in which the server takes precedence over state events and positions of objects. One main tactic were planning on implementing before using the Unreal Engine is server-side lag compensation. The way this works, is that the server keeps a record of where all plays are from a second before the current time, up to the current time. When bullets are fired, the server estimates where the other players were on the client's computer when these bullets were fired, and attempts to calculate a hitbox to see if the bullet hits anything. This way, we can tell if the player shoots who he thinks he shot, even with significant lag. By reading the Source Multiplayer Networking documentation, we were given insights into how to handle problems like low bandwidth and lag. Since the server code is open source as well, we have downloaded the code and are planning on taking a closer look at it to get a good idea of how AAA-grade networking architecture is constructed from a programming standpoint.

Finally, in order to get a good handle on creating an engine's networking, we first needed to get a good handle on game networking and learning all that we need to understand in order to have our networking layer running as quickly and smoothly as we can. Our main tutorial on setting up a networking interface was Glenn Fieldler's tutorial, "Networking for Game Programmers". Glenn Fielder currently works at Respawn Entertainment. This tutorial starts at explaining the difference between TCP and UDP (and which protocol we should favor, goes through concepts such as virtual connections, reliability and flow control, and multiplayer debugging, and ends with a brief history of FPS multiplayer games, and the growth of the genre.

Overall, though there aren't many super-advanced strategies and algorithms learned from this tutorial, this base of knowledge is of immense help in the development of the networking component of our game.

Upon beginning of our capstone project in the Unreal Engine, much of the research dealt less with designing a networking architecture and more with researching how to best utilize the Unreal Engine in the context of our game. Unreal has networking “built-in”, but a lot of the documentation surrounding it consists of a series of tutorial videos and definitions of the classes involved. Though this may be excellent in getting a basic game running, extending this architecture to include features such as an in-game lobby was a full-time task. Implementing many networked menu features using the Blueprints feature available in Unreal Engine 4 proved challenging, especially since the testing and documentation for the Unreal Engine was not as plentiful as that for the Unity Engine, for example.

The main resource utilized for learning Unreal Engine’s networking architecture was Billy Bramer’s Blueprint Networking Tutorials, found on the Unreal Engine’s website. This resource consists largely of a series of YouTube videos, starting with basic networking terminology, then running through simple networking demos, then finally running through adding networking features to an existing Third Person Game template. This site was an excellent source to get started on Blueprints in Unreal, as well as how to properly structure your code so that it does not get too unwieldy. However, one issue was the lack of supporting documentation on the project if you did not want to watch a YouTube video. In addition, the videos did not delve into a lot of the modules and pieces of the Unreal Engine that you would need to work with to create your own networking features. Overall, this site gave us a good start into networking with Unreal Blueprints.

3.2.2 Terrain Feel (John Dunham)

3.2.2.1 Introduction

Terrain tends to be one of the more undervalued elements of a game's design. The majority of the time, studios will simply throw a height map at the problem and pick up the slack with meshes for any details not easily represented in a height map (which aren't always coherent). Too often the terrain of a game is auxiliary to the design of the game, despite being one of the most visible and interacted with elements of the games simulated space.

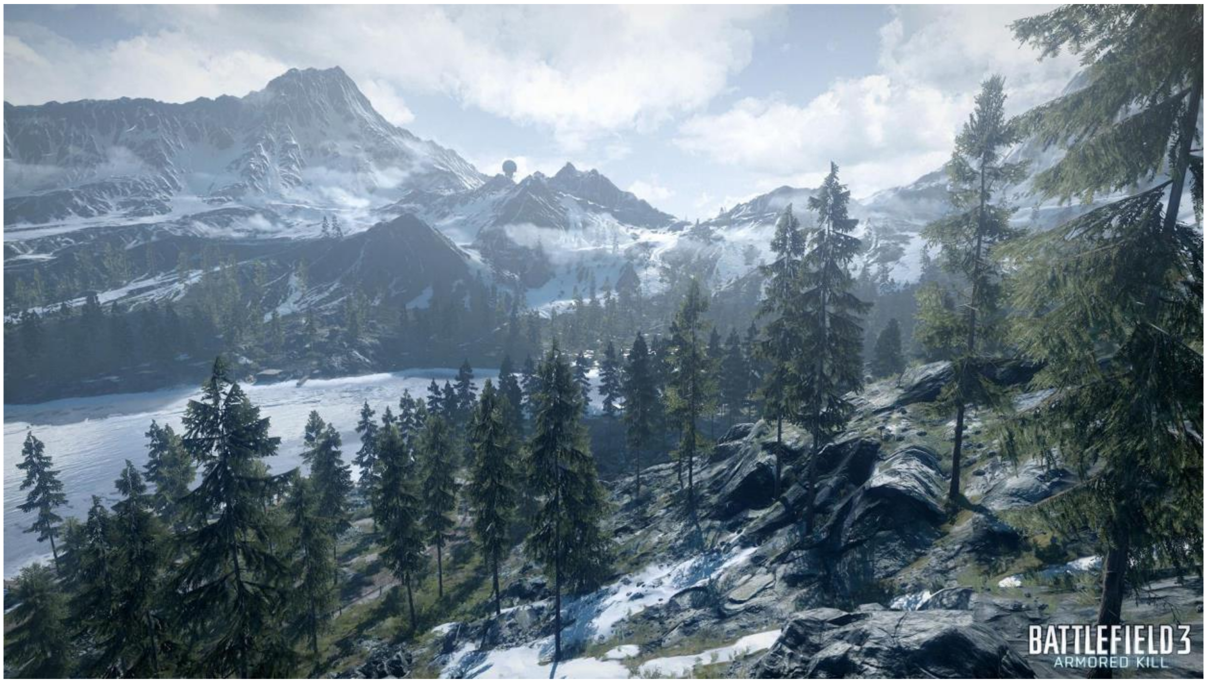


Figure 30 Example of a simulated terrain (DICE 2012)

The visuals of the terrain: how far the user can see, how detailed the height map is and whether or not there are artifacts in the texture stitching all contribute to the realism or lack thereof for the player. As seen in Figure 30 Example of a simulated terrain , a screenshot from DICE's Battlefield 3, solid visuals can contribute significantly to a player's immersion in the simulated space. In fact DICE has such command on engineering the visuals for their terrain that still shots look almost as good as real life photographs, a fact that is certainly noticed by the players.

In addition to visuals, terrain demands a fair amount of simulation/physics to immerse the player. If a player can walk through a tree in a exploration game, or falls through the ground it can be insanely jarring. Likewise, if the player can't move through the space in a meaningful way they'll lose interest in even the most beautiful of vistas. It may be postulated therefore, that good terrain design and development requires a juxtaposition of visuals and simulation. For the purpose of this research this juxtaposition will be referred to as *Terrain Feel*.

The notion of *Terrain Feel* is the core of this research. While the best way to represent a highly detailed terrain, or how to optimize how the pipeline transmits the current state of terrain on the CPU to the GPU may contribute to the notion of *Terrain Feel*, these subjects are auxiliary in context. *Terrain Feel* is a repurposing/specialization of the notion of game feel proposed by Steve Swink (2009) in his book *Game Feel*. Swink postulates that game feel consists of three core components: Spatial Simulation, Real-Time Control and Polish. *Terrain Feel* focuses in particular on the importance of Spatial Simulation and Polish (visuals) and attempts to enhance and support the overall feel of a game.

3.2.2.2 Types of Terrain Representations

As terrain can be represented in a number of ways for the sake of clarity and brevity, this research will focus upon three means for representing terrain: Height Maps, Vector Field Terrain and Voxels.

Height Maps are the most commonly used means of representing terrain, implementations existing in nearly every commercial engine ranging from *Ogre3D* to *Unity*. This particular representation is probably the easiest of the different representations discussed here to get up and running, making it a prime choice for most studios (less time is less money spent). The height map is also astoundingly fast and has an extremely small memory footprint. Additionally, representing terrain in this manner can look incredibly realistic if used properly since it is built upon the notion of topographical maps.

The Frostbite 3 engine developed by Swedish developer DICE has excellent examples of height map-based terrain done right (see Figure 30 Example of a simulated terrain). The terrain in the Frostbite engine is a work of art combining procedural texturing, terrain decoration (trees, rocks and grass), destructible terrain and much more. (Widmark 2012) DICE did a good job at making terrain feel real, they straddle the line between polish and spatial simulation, allowing the player to lose themselves in the game. That being said as with all things, in using a height map the Frostbite engine trades a certain degree of flexibility for speed. The chief issue with such systems is that both sharp gradations (e.g. cliffs) and overhangs are not easily represented and typically need to be faked in order to look appropriate.

The vector field height map is a potential solution to the aforementioned drawback. A vector field height map is a variation upon the traditional height map in which three data points (x, y, z) are stored on each point of a generally high resolution grid. As the basic height map only stores the height value at each grid point it is easy to see the advantages of a vector field, chiefly that vector fields are capable of representing overhangs in the terrain directly.

Despite the potential for this representation, it has only seen use in one game: *Halo Wars* (2009). *Halo Wars*' terrain was created in response to the fact that for RTS games terrain is massive, taking up a major portion of screen real estate, and to innovate in that space would result in the "Biggest bang for the buck". (McAnlis 2009) As a result, vector fields found a place in *Halo Wars* terrain rendering system. With it the *Halo Wars* team was able to do things that are close to, if not impossible to do, with a height map alone: overhangs, cliffs with minimal mess distortion and artist definable vertex densities. As with height maps, however, this system is not without its flaws. It is far harder to handle collisions in the traditional sense due to overhangs which may not play nice with the broad phase AABBs, a fact noted by McAnlis (2009) in his talk on the technology.

Voxels are something wholly different in regards to terrain representations. Ostensibly, there is no limit to the types of terrain that may be represented by voxels given enough space and a fast enough computer. That being said voxels are very heavy on both computation time and storage, potentially taking up far more space than even a mesh representation of the terrain. Additionally, voxels aren’t supported natively on graphics cards, requiring a rendering algorithm such as Constrained Elastic Surface Nets to even see them in some capacity. (Gibson 1991) While the technology is interesting, Pursuing voxels to represent terrain is perhaps too time consuming given the potentially limited rewards.

As such, the vector field height map seems to have the most promise as a research avenue. There is little written on the subject beyond McAnlis’ 2009 talk at GDC leaving open a number of pressing questions in regards to implementation and impact upon *Terrain Feel* in games outside of the RTS genre. Vector field terrain also is a sound decision in that it can be built with a layered approach, first implementing a height map based system for other members of the team to work in so their progress isn’t stymied while research into vector fields is ongoing.

3.2.2.3 Terrain LOD Systems

LOD or Level of Detail is incredibly important, as it allows the game to run smoothly and prevent unnecessary computations from impacting the feel of other gameplay components. In terrain, LOD is especially key. If an open world game were to render the whole terrain at once the frame rate would absolutely plummet. In fact in modern games if the terrain just renders what the user can see without any level of detail the game would likely grind to a halt. This is a non-option when pursuing good *Terrain Feel*, as such a LOD system is necessary in designing a terrain system with good feel. As hardware tessellation was used in *Halo Wars*, the only other game with a vector field height map, tessellation seems like a good jumping off point for terrain LOD (McAnlis 2009).

3.2.2.4 Terrain Feel Implementation

Ultimately this research acts as a background and jumping off point for future study. Through the process of design iteration it was deemed that terrain was not necessary in the context of *Crystalline*.

This is partially due to the fact we switched to Unreal, but primarily due to the fact that a modular art set better represented the spatial spaces of our game world. Were *Crystalline* to be expanded this research may be furthered in the context of world editors, however, no formal plans were created at the time of writing.

3.2.3 Graphics (Matthew Mundell)

An important element that we all agreed on is that our game should have a visually rich, vibrant world. To that end, graphical research will include studying both standard techniques widely agreed upon as well as some of the newest techniques in storing, manipulating and rendering data. The goal is that in doing this research we can create an efficient, robust rendering system which provides high quality graphics in our product, is easy to use, and suggests techniques for specific effects. Furthermore, it should be robust enough that effects can largely be generated procedurally, such that variations in color, decals, secularity, sizes, or more complex parameters can be done programmatically to avoid creating assets in large quantities.

First we want to have a system capable of producing a reasonable image using basic techniques to render 3D and 2D geometry. Typical modern games include: deferred rendering to support many lights, normal mapping to simulate high levels of detail on simple geometry, hardware instancing to batch-render objects, and post processing to generate special effects after a scene is rendered to a 2D image. In addition to this baseline, we want a system for managing different rendering components and objects easily and with minimal memory footprint. One newer technique is to encode texture data into half the number of channels necessary to reduce the total number of textures which must be sent over to the GPU (Engel 2014). Some of these techniques, deferred rendering in particular, presents problems when trying to create certain effects like anti-aliasing or transparent objects. Therefore emphasis has been placed on researching how to implement these effects on top of a deferred rendering system.

The direction of many graphical systems recently is to use compute shaders to exploit the speed of GPUs to create effects too prohibitively slow to render. The graphical research for this project is heavily towards rendering effects that use this technology. We hope that with the experience of using compute shaders for graphics, we can potentially extend them for use in GPGPU operations such as updating of objects or AI calculations on the GPU. Some of the major effects we will focus on include: particle systems, bloom, and volumetric lighting (a.k.a. god rays).

Particle systems are now almost ubiquitous in games, representing effects like smoke, fire, bullets, rain or snow, or even whole meshes. When DirectX 10 was released, the approach to particles changed greatly from traditional vertex shader computation over a series of triangles, to storing particles as single points that can be expanded on a geometry shader. Then with DirectX 11, the heavy calculation of particle movement and physics was moved again to the compute shader (Latta 2004). Games such as *Infamous: Second Son* use compute and geometry shaders to dynamically generate the form of particle systems as well as ribbons which follow the character. We want to use particle systems to generate effects such as the bullets in our game, the impacts and explosions that they create, volumetric fog, and visual indicators of events like if a base is being taken over sparkles can swirl around it. In addition, they can be used as intermediate steps to create other graphical techniques such as volumetric lighting (Engel 2014).

3.2.3.1 Animation

An important piece of many modern games is animation. Typically animation is handled by first creating the animation in an art tool such as Autodesk's Maya. The process involves creating a model, rigging it with a skeleton and control objects to help manipulate the skeleton, skinning the model to the skeleton, and then animating it by setting the model into various poses using keyframes along a timeline. The keyframed data is then interpolated over time as a curve which can be manipulated to tweak the animation. This data, the skeleton, and the model in its base pose (the way it looks prior to any animation) are then exported from the modeling tool.

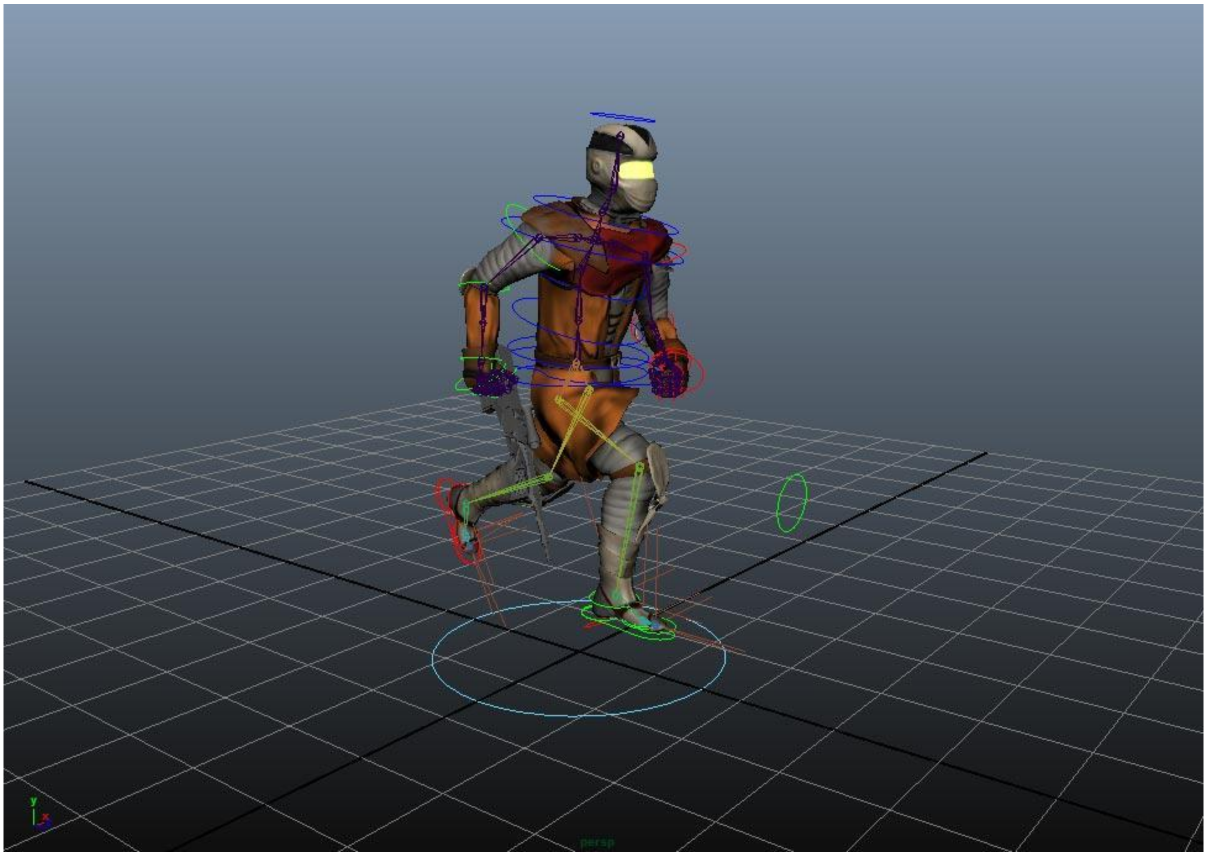


Figure 31 Our animated character in Maya

The problem with this technique is that by itself it creates a small number of very specific visual effects and creating the many possible poses and actions of a game quickly becomes unfeasible. To resolve this graphics programmers use techniques like animation blending, layered animation, skeleton retargeting, and parametric animation, just to name a few. Animation blending allows artists to blend two or more animations together. This may be used to transition one animation into another, by playing first 100% of one animation and 0% of the other, then the next frame playing 90% and 10%, and so on, creating a smooth transition that may not be perfect, but will not distract the player. Layered animation means to additively perform one animation in addition to another. For example, the legs of a character may perform the ‘Run’ animation while the arms of the character may perform the ‘Shoot’ animation. This is significant because it allows games to use animation sequences more efficiently. The work of creating the ‘Run’ animation just has to be done once, and it will work regardless of what the arms are doing. Skeleton Retargeting is a technique to reuse one skeleton (and its associated animations) on a different model. This allows for even more efficiency in producing animation-related assets by reusing work for multiple models. Parametric animation refers to driving the animation through values that change based on events in the game, such as changing whether the character is idling, walking or running or blending somewhere between those based on its speed. This allows for greater flexibility and interactivity with the character and combined with the other animation techniques, gives artists and programmers a great amount of control over the final appearance of a character in game.

A common problem posed by animation in games is how to handle dynamically changing geometry attached to animated character such as weaponry and equipment. One approach is to include these in the animations themselves and create one for each combination of equipment that will be on the character. However, this is unfeasible in games with large amounts of content. In Unreal Engine 4, our engine of choice, animators use a device called a socket to handle this problem. A socket is a named point in space attached to a bone on a skeleton asset. The socket can then be moved or oriented in any way in relation to the bone. Then a weapon or other object can be 'attached' to the socket so that it transforms along with the socket, like any child object in a scene hierarchy. That means that whenever the bone performs any animation, the object will move in an appropriate way to match it; this is the technique we use to implement our character's weapon inventory.

Another issue with animation is how to process events which occur at particular times during an animation. A good example of this is when feet hit the ground in a running animation. Typically a game will play a footstep sound bite and/or have a puff of dust or footprint decal appear when and where the foot hits the ground. However, this event is based entirely on the timing of the animation rather than by any controlled input or game event. To get that data, the animation must be tagged in some way to inform the engine when that event occurs. Unreal's Persona system uses the concept of a 'notify' to achieve this. A notify is just a marker that can be placed at any point along a particular animation's timeline and can be used to propagate an event that can be captured in code to trigger something happening. This way an artist can easily tell the engine when an animated event occurs without having to manually record the frame for every such event in all his animations.

3.2.3.2 Physically Based Rendering

Traditionally, many games approximate the visual appearance of objects using attributes like diffuse color and specularity. Advanced effects like reflection are then added through specialized materials and shaders. This approach is common because it is relatively simple to implement and provides reasonable results which we have enjoyed in computer generated graphics for years. But it is still just an approximation, and as graphics and processing power have increased over the years, the need for more realism and more advanced effects has grown as well. Therefore a new approach has recently become popular called physically based rendering. As the name implies, physically based rendering means using more physically realistic parameters and methods to determine how light really interacts with a scene to produce the final visuals.

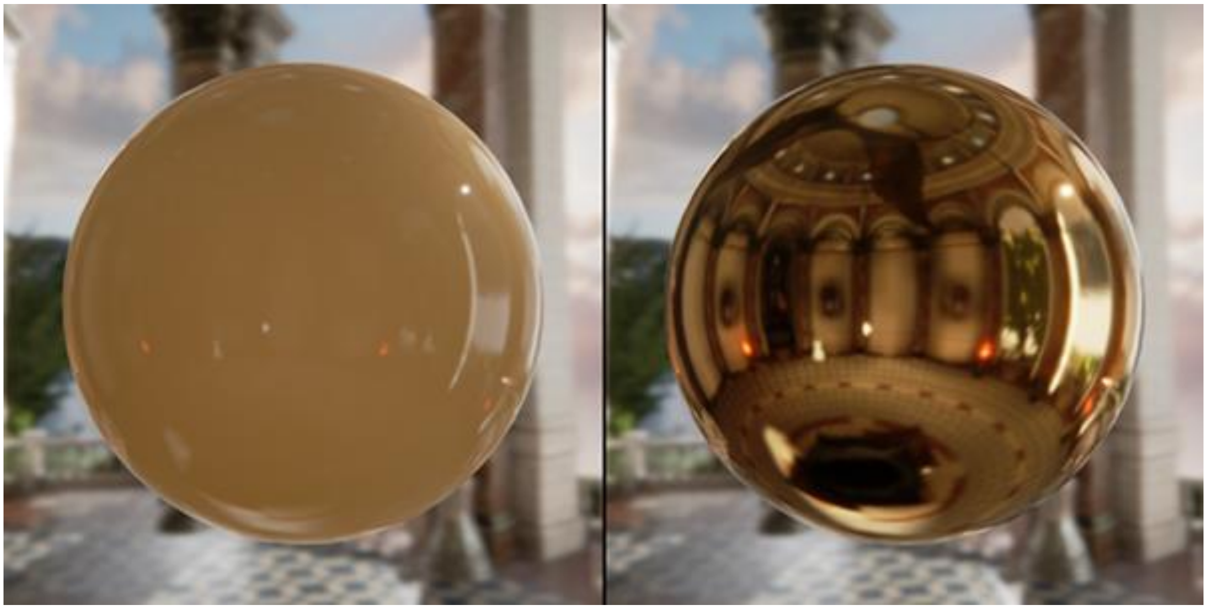


Figure 32 Physically based rendering in action (Walker 2014)

Like traditional materials, physically based rendering systems still use a base color attribute, but the similarities end there. A metallic attribute tells the engine how metallic a material is, usually

0 (plastic) or 1 (metal). This affects how reflective the surface is. Then roughness describes how smooth or rough a surface is at a near-microscopic level. This is similar to specularity in a traditional system; A very rough surface (near 1) is going to have little reflection or highlight, whether it is metal or plastic. A very smooth surface (a roughness near 0) will be highly reflective and have a bright highlight. A specular attribute also exists in physically based materials but has a slightly different usage than in the traditional system; it refers to how bright the highlights will be rather than where they will be, and is closer to the concept of 'gloss' in the traditional system.

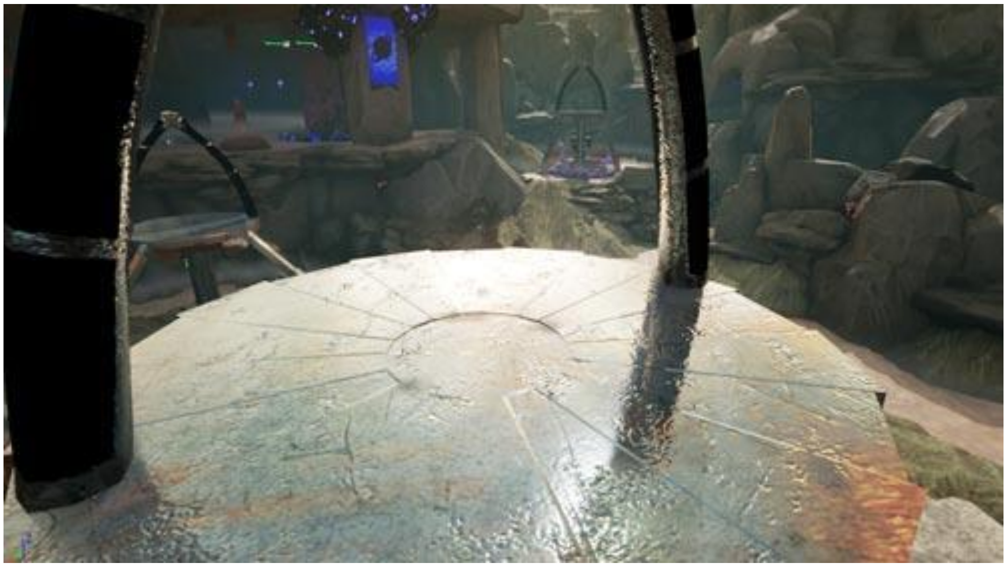


Figure 33 A metallic surface in game

Artists of the traditional system often *approximate* metallic surfaces and non-metallic surfaces using a black and white 'specular map' so that some areas have bright highlights, appearing metallic, and others have no or very soft highlights to appear like cloth or leather. Using the physically based system, materials can be thought of as how they would be in the real world, so artists can instead say this piece *is* smooth metal and that piece *is* rough plastic and so on. This actually makes the process more intuitive, but requires different kinds of artistic assets and therefore artists have to learn how to use the new system to take full advantage of it. The power of physically based systems come from the fact that by using realistic properties, the visual appearance of materials becomes more consistent and intuitive to work with across different environments and lighting models, and it allows for advanced effects like subsurface scattering, reflection or refraction to be achieved more easily.

3.2.3.3 Skybox

Oddly enough, one of the largest visual pieces of a game is often one of the least explained: the Sky. The sky in most games is a at least in part a texture rendered on to the sides of a cube or sphere that moves with the player and is always rendered behind everything else in the scene. Some may blend between textures to create different environments or times of day, and sometimes games use the concept of a ‘3D skybox’ using models and other effects to create far-away geometry or activity that can’t actually be reached by the player. This may include explosions in the distance, clouds generated by particle effects, mountains and other landscaped geometry, etc. But the difficulty with creating a unique sky is in creating the texture to draw across the skybox (or sphere). The general technique is to take a panoramic picture or just a large image, and break it up into 6 chunks. Then, imagine an unfolded cube on top of the image. The sky will be drawn across the faces of this cube (or projected from these faces onto a sphere). The problem is the edges that need line up seamlessly. If you imagine folding the box up, you can see where the edges will line up and where you need to blend them together. However this task is more easily said than done and the process can be slow and tedious.

A new potential approach is to take advantage of a tool like Mudbox. Mudbox is a high-poly sculpting tool allowing 3D artists to create fine details in models at high resolution and bake them into textures to render in game. Mudbox includes a feature allowing artists to paint directly on a mesh in 3D space rather than using the typical approach of exporting a UV layout and drawing over it in Photoshop. This feature gives us a new option to create skyboxes. First, a cube is made in Maya and it is UV unwrapped so that each face is its own UV Shell (no edges are connected). Then the UVs are carefully aligned in a row and each face should be oriented according to how the tool of choice, in this case Unreal, imports cubemaps. Then the cube should be smoothed into a sphere for visualizing the final sky easier. We didn't start with a sphere because then the UVs would not be arranged as 6 squares. Then we can bring this reusable model into Mudbox. One must be first be careful to set Mudbox's preferences to have no edge bleed and to paint on overlapping uvs, which are set differently by default. Then we take any textures we like and paint them like a stencil over the cube, crossing edges naturally, and Mudbox automatically paints the texture onto the correct places on the 2D UV layout. This greatly simplifies the process of creating seamless edges and allows to quickly layer different textures or make changes to the overall skybox very quickly. The final work can be exported from the paint layer in Mudbox as the usual 2D image of a completed skybox necessary to render in game. Some transparent pixels along the edges may still need cropped out in Photoshop, but this step can be automated or negated with more careful alignment of UVs from the original mesh. The final output from Mudbox still needs to be exported from Photoshop as a cubemap, and for this Unreal recommends Nvidia's free Texture Tools Photoshop plugin.

Figure 34 Carina Nebula through ultraviolet filter

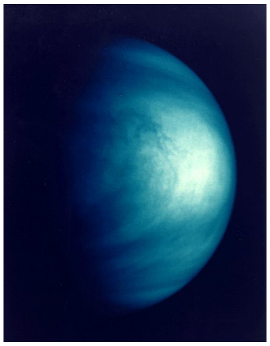


Figure 35 Venus through ultraviolet filter

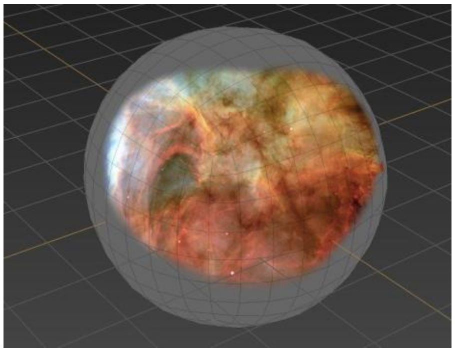


Figure 36 Painting the source images as layers on skybox in Mudbox

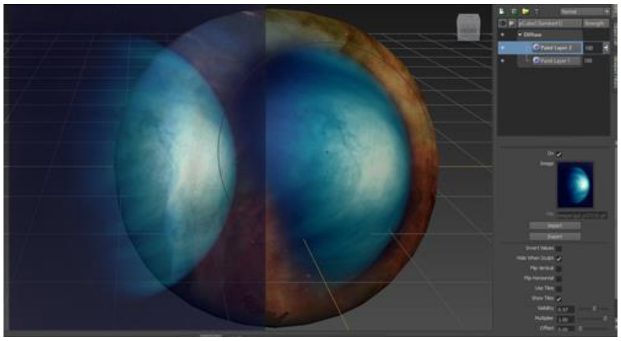


Figure 37 Painting the source images as layers on skybox in Mudbox

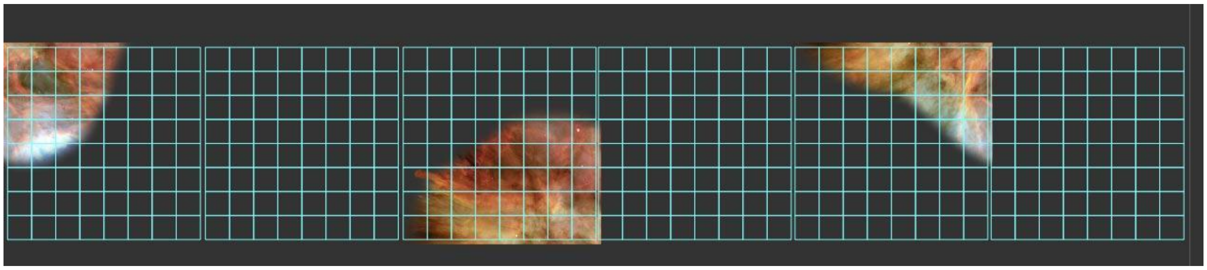


Figure 38 Skybox being painted in Mudbox

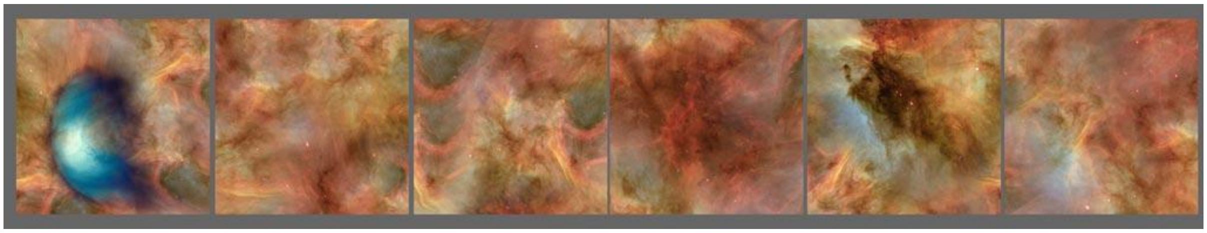


Figure 39 Final Skybox output from Mudbox

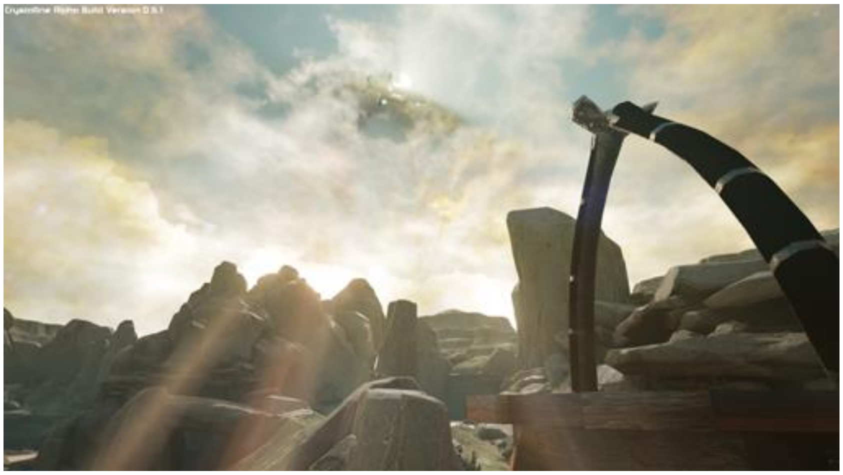


Figure 40 Final in-game sky showing light shaft occlusion and bloom

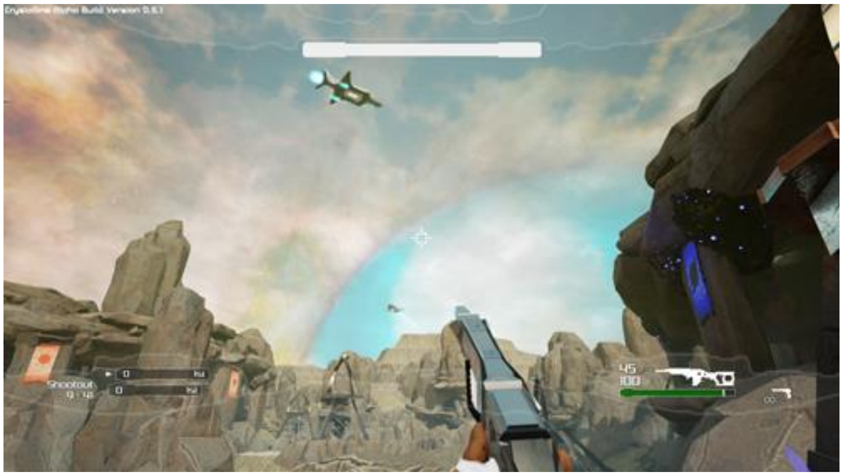


Figure 41 Final in-game sky showcasing planet and spaceships

4. Game Development Process

4.1 Team Structure

Our team is made up of Nick Buonarota, John Dunham, Alex Herdzik, and Matt Mundell, specializing in level design, game feel, networking, and graphics respectively. We decided in the beginning of the project that instead of designating a single leader or producer, we would instead share the responsibilities of starting meetings and writing down notes from day to day.

4.2 Defining the Game's Aesthetic and Genre

During summer months of 2014, the team had many discussions about creating a game called *Overscoped* . The goal of *Overscoped* was to create an epic experience that broke and blended genre boundaries of various shooters, strategy, and role playing games, ultimately creating a new genre of superb war gaming experience. After a few months of pre-planning and a week of focused design, *Overscoped* was living up to its name.

As a team, we discussed why we liked *Overscoped* so much, but more specifically, the core game element pillars, which we felt strongly about. With those focuses in mind, the team split up to try to recreate those experiences in a much more scoped down, bare bones, minimal setting. These prototypes, while capturing the scope, felt lacking. They felt like pieces that were an element of a whole, and overall, they were shallow experiences.

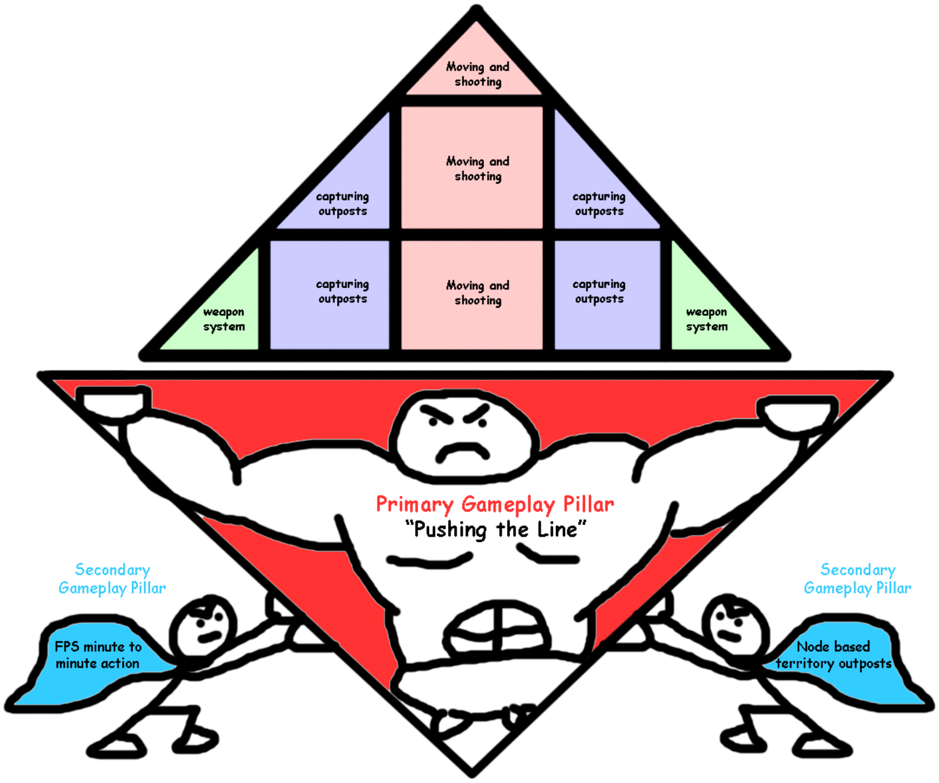
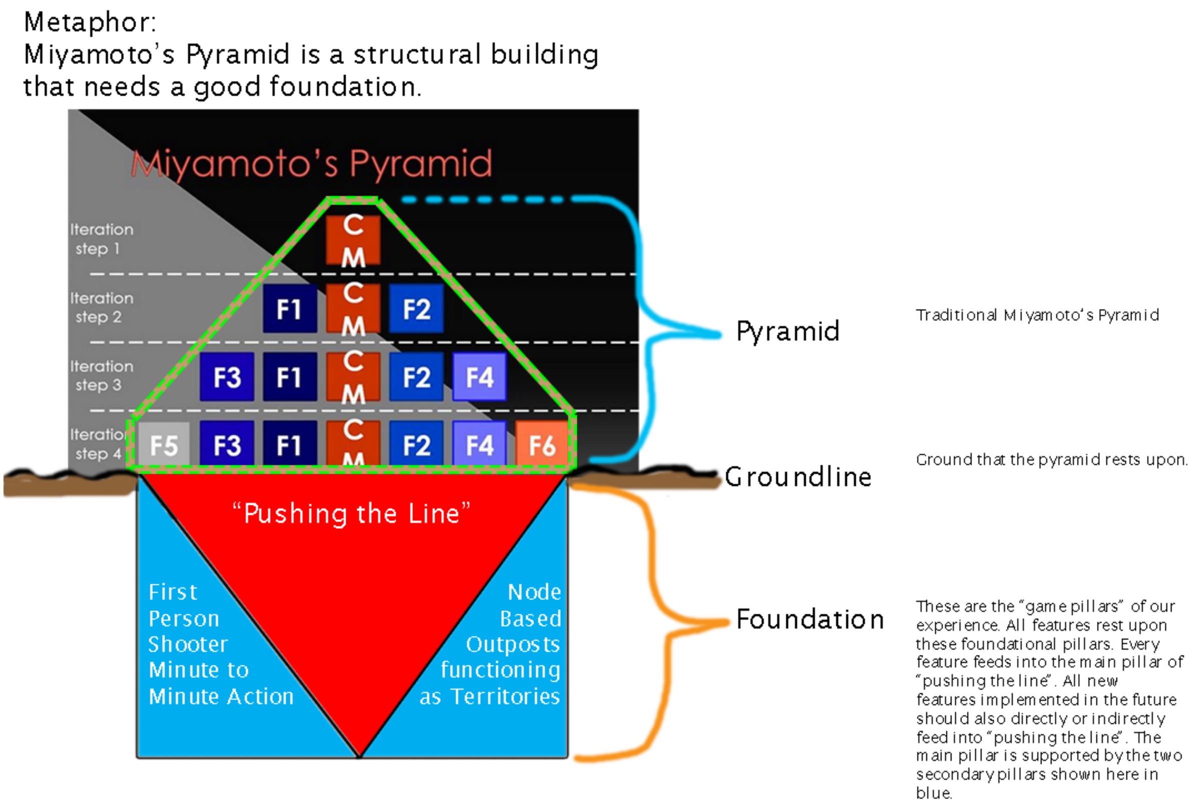


Figure 42 Game pillars concept image

After more discussion and analysis, we further defined the true core pillars of our experience. We examined the two prototypes, *Colony Crisis* and *Toy Box Arena*, and then compared them to the original idea of *Overscoped* . It was at that moment in which we discovered the experience we needed to create, blending the core elements from the three experiences. Interestingly, we kept a lot of the aesthetic feel of *Overscoped* scoped, yet we efficiently implemented that feel with many of the design choices and inspiration from the two scoped down prototypes of *Colony Crisis* and *Toy Box Arena*.


Metaphor:
Miyamoto's Pyramid is a structural building that needs a good foundation.
Miyamoto's Pyramid
Iteration step 1
Iteration step 2
Iteration step 3
Iteration step 4
C M
F1
F2
F3
F4
F5
F6
"Pushing the Line"
First Person Shooter Minute to Minute Action
Node Based Outposts functioning as Territories
Pyramid
Groundline
Foundation

Traditional Miyamoto's Pyramid

Ground that the pyramid rests upon.

These are the "game pillars" of our experience. All features rest upon these foundational pillars. Every feature feeds into the main pillar of "pushing the line". All new features implemented in the future should also directly or indirectly feed into "pushing the line". The main pillar is supported by the two secondary pillars shown here in blue.

Figure 43 Miyamoto's Pyramid Metaphor

At this time, all of our mechanics rested upon three pillars of gameplay experience, one primary pillar and two secondary pillars. The core of the gameplay experience was focused around the concept of "pushing the line", a term we came up with to define the literal line of contention between the two teams in our game. Everything went back to, and was related to, this concept of "pushing the line" and if it wasn't, then we questioned what purpose the feature had in our game and what it actually contributed to the experience.

The first secondary pillar was, "First Person Shooter minute to minute action." This was one of our major gameplay experiences that we wanted to deliver. The FPS minute to minute action was a major supporting and contributing factor to the main pillar of "pushing the line." The other secondary pillar was, "node based outposts functioning as territories." This was the other major core gameplay experience that we needed to deliver. The node based outposts supported the primary pillar of “pushing the line” because connecting the dots together between nodes is what actually created the line that you push.

Because of all these factors, our game wasn't a straight up First Person Shooter deathmatch arena like *Doom* or *Quake*, but our game still fell under that large genre umbrella because we had shooting mechanics and it was seen from the first person perspective. To be more specific, and to help differentiate between other FPS games such as *Doom* and *Quake*, we attempted to further quantify our game’s genre and aesthetic as an “Action-Strategy First Person Shooter”.

4.3 Prototypes

4.3.1 Overscoped

First Person Shooter, RTS, RPG, Large Scale, Warfare

Overscoped was our initial concept for the capstone: conceived last spring and discussed for the duration of the summer of 2014. In the original design we conceived was a layered FPS war-game in which players would interact with the world as both commander and combatant. The player could both determine where new buildings and features could go on the map, but also how their individual player advanced.

Setting

Overscoped was to follow a “space cowboy” aesthetic, drawing heavily from *Firefly* as an inspiration. As a team, we found this aesthetic gave us the most leeway for setting, we weren’t simply limited to earthlike weapons, combat, or environments. The notions of giant rocks floating around, lasers blasting across the battlefield, and space ships in the skybox fills the imagination.

Gameplay

The core goal of the *Overscoped* gameplay was to fuse the high action combat of First Person Shooters with the large scale war gaming of a Real Time Strategy game. Our design work focused on making this juxtaposition not jarring and generally fun.

One of the plans utilized a second screen app that the player could manipulate to control the flow of combat, as though they were the commander in an RTS. For example, a player on the second screen may use it to launch an airstrike on an enemy fortification, or send a battalion of soldiers to take a base. Obviously this tactic would have heavily leveraged networking, so it would have been no mean feat to make the transition feel “right”.

We additionally toyed with taking an approach closer to the engineer from *Team Fortress 2*, allowing the player to physically place objects in the game world. Defense turrets, vehicle spawners, and weapon depots were just three of the multitude of buildings proposed. In the same vein the notion of a home base was toyed with, which would have its own set of rules and enhancements.

Prototype

The prototype for *Overscoped* implemented the basic multiplayer connection and attack turrets, but not much else. Before getting deep into the prototype we realized as a group that *Overscoped* lived up to its name. As we examined where we could scale back or cut features, we came to the realization that we had inadvertently coupled gameplay elements too tightly. Meaning that if we were to cut one element the other elements would suffer as well. This conundrum led the group to split up for two weeks and work on new game ideas and prototype them to see if we could distill the core essence from *Overscoped*. As a result *Toy Box Arena* and *Colony Crisis* were born.

4.3.2 Toy Box Arena

Action, Shooter, Fast Pace, Arena, Thunderdome

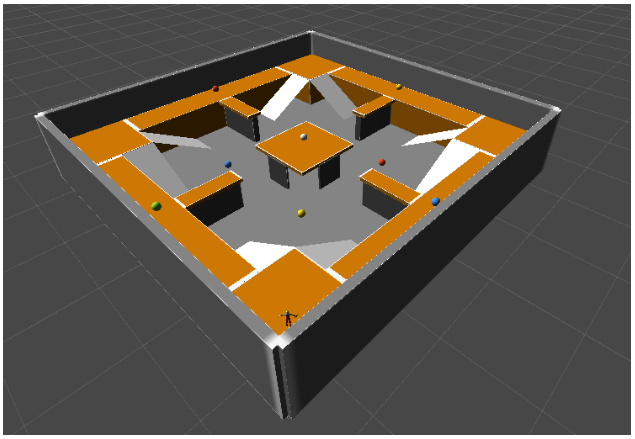


Figure 44 Toy Box Arena prototype

Gameplay

Players have five minutes in a thunderdome arena deathmatch. The arena is the same every time. The arena is a small mid-range thunderdome, with areas and components designed around and for specific gameplay styles (defensive play or aggressive play for example) and gameplay mechanics such as jumping and shooting). Players get points by killing other players and/or completing objective goals.

In the center of the arena, there is a big red button that is powered off the entire game. At the end of the five minutes, the big red button is powered up and glows bright red. The players then rush to the big red button and try to be the first one to push it. If the player in the lead with the most points presses the button first, then the match is over and the game goes to the post-game lobby and recap screen. If anyone else is the first to reach the button and push it, then the game goes into sudden death overtime!

In sudden death overtime, the big red button in the center of the map disappears and reveals a large trap door in the floor. The trap door opens up, and gives access to a secret dungeon battle zone. The players then all get sent down the hole to this new dungeon arena to fight in sudden death.

The dungeon arena is procedurally generated each match if sudden death mode is activated, so each dungeon arena is unique and never the same. Being a new map that the user has never seen before, the mystery of the unknown further adds to the adrenaline rush of sudden death overtime.

4.3.3 Colony Crisis

Shooter, Action, DPS Race, Upgrade Game

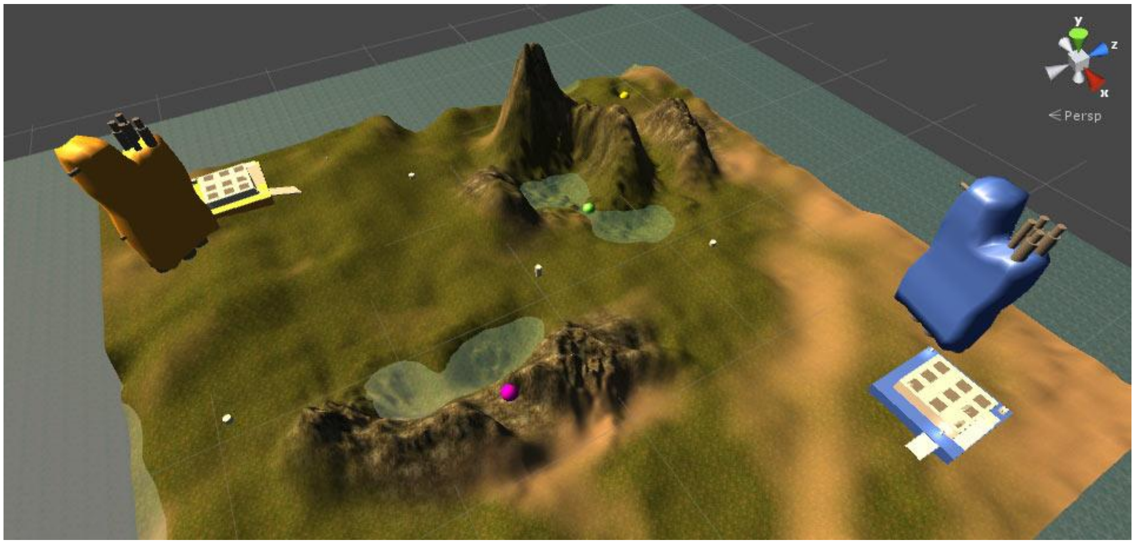


Figure 45 Colony Crisis prototype

Two factions have landed mother ships on a planet with the intent of aligning that planet to their cause. Each team begins with a base and a small amount of funding to take the planet. When the player buys an upgrade on the base, it can either be a weapon which will begin firing on the enemy mothership, or a civilian upgrade which will influence the local population. Teams will begin making money each second from helping the colonists or each shot by protecting them from the other team. Then teams can use this new income to continue getting more and bigger upgrades, until they either destroy the other team or take over the planet's local population, forcing the other team to leave. While the player's base only attacks the enemy's mother ship, players may attempt to attack the enemy base and upgrades to slow their progression. They may steal resources or hack into the base computers to cause the enemy's equipment to backfire.

In addition to gaining income through base upgrades, players can explore the environment for one time and persistent resources which can be gathered and used as boosts for weapons and upgrades or to sell to the local population for quick cash. Some of these resources make weapons more powerful against other types of resources. For example, if one team upgrades their ship's armor with a purple 'Ion' crystal so it repairs over time and the other team upgrades a turret with a red 'Volatile' crystal so it shoots faster, the turret will do extra damage against the upgraded ship. Players can deploy hovering turrets to protect passages or resources, or deploy flags to declare their ownership of the planet, giving that team more cultural points per second depending on how far from their base they have managed to claim. Aerial drones can be purchased to gather resources automatically, or to drop bombs on a targeted position. Player's will have the opportunity to shoot down these drones and deployable turrets while their base attempts the takeover of the planet.

For more detailed design information written for *Colony Crisis*, see **Appendix A.5**.

4.3.4 Crystalline Prototype

After trying different ideas resulting in *Toy Box Arena* and *Colony Crisis,* we decided we needed to agree on elements we liked from both while maintaining the scope of the project. Thus, after a lot of thought, *Crystalline* was born. The original vision of *Crystalline* included the blend of genres we originally wanted in *Overscoped*, using mechanics and ideas drawn from *Toy Box Arena* and *Colony Crisis* to reduce its overall scope.

4.3.4.1 Developing the weapon system

Both *Toy Box Arena* and *Colony Crisis* had one thing in common: powerups. *Crystalline's* weapon system was borne from the fusion of the power up system of the prototypes and the more traditional FPS fare of *Overscoped*. In order to increase the depth of the weapon sandbox we looked into how *Gunstar Heroes* (Sega 1993) managed power ups.

Gunstar Heroes* to *Crystalline

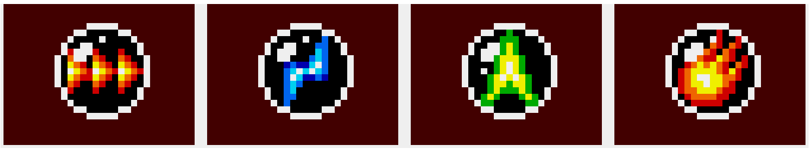


Figure 46 Gunstar Heroes Powerups - Force, Lightning, Chaser, Fire (Sega 1993)

Gunstar Heroes had a straightforward gun power up system consisting of 4 basic power ups: Force, Lightning, Chaser and Fire. Force is high powered and fast; Lightning is a piercing shot that passes through enemies; Chaser homed on enemies; and Flame a close ranged weapon. These power ups could then be combined in a novel manner producing interesting weaponry. For example, Chaser and Lightning produces a homing beam that will track and pass through enemies dealing damage.

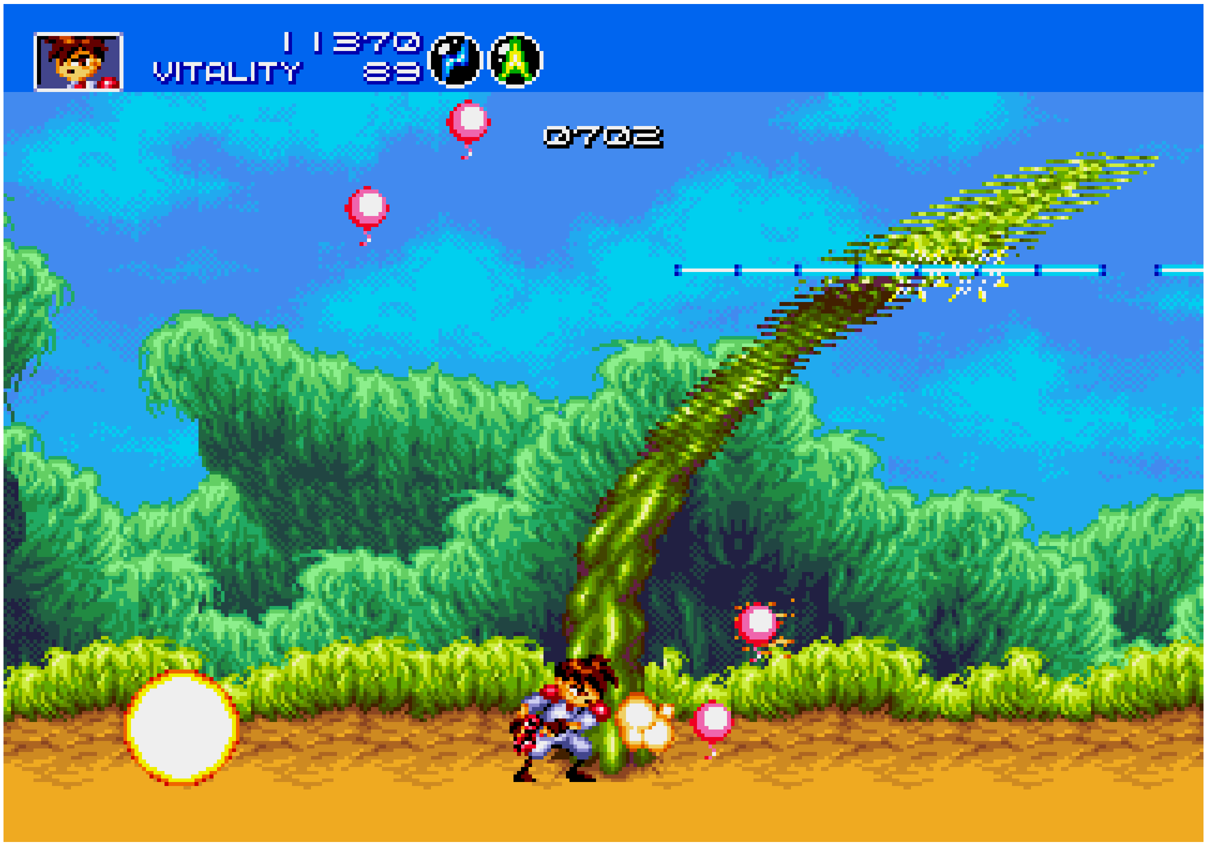


Figure 47 Gunstar Heroes Lightning and Chaser Power Up combination (Sega 1993)

Not only did this system promote fun mechanics, but it also produced a rather deep weapon sandbox, 14 weapons in total, with only four simple building blocks that were readily accessible to a new player. Upon playing through *Gunstar Heroes*, it was discovered that the rough powerups from our prototypes could be readily transferred to a similar system.

Colony Crisis featured different colored crystals that either increase Firepower, Speed of the bullet, or Rate of Fire. In *Toy Box* we had four power up types: Speed, Defense, Offense and Trap. As we had determined that the game would be largely an FPS it was decided that Traps would not be easily accessible to a new user. Additionally, as weapons are offensive by their nature Offense and Defense were transitioned to Power and Range.

With a new set of power ups in hand we brainstormed how to best convey a power up system in a FPS. Since this power up system would effectively take the place of our weapon sandbox it was determined that a “morphing” gun that channels the energies of the eponymous crystals of *Crystalline* would be the most interesting and novel means to articulate this mechanic.

4.3.4.2 Crafting the Crystal Gun

The crystal gun was initially perceived as a firearm with some kind of slotting system, where the slots would hold the various crystal types. A supersoaker-esque aesthetic was conceived in which crystals would be slotted into the top of the weapon and the gun would morph both its shape and ammunition to respond to the new crystal energies, producing new and exciting power ups. The crystal gun would have two ports for such augmentations resulting in ten possible weapon permutations. At this phase it was also decided that the crystal gun would be ammunition based, as a number of the proposed weapons (Rocket Launcher, Sniper Rifle, Shotgun, etc.) would be hard to balance otherwise. During development, it was also decided that the gun design would be scoped down to have only one available crystal slot instead of two, to reduce the amount of available weapon combinations.

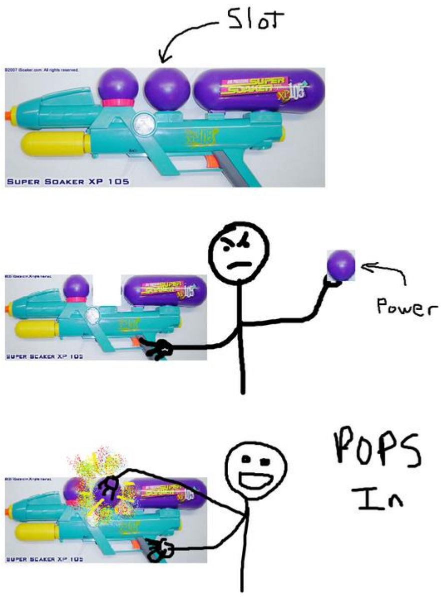


Figure 48 Initial mockup of the slotting crystals in the Crystal Gun

In an effort to prevent the player from becoming entirely defenseless in the event that their crystal gun had no ammunition, a side arm was included in the design. This sidearm, a pistol, would be a cooldown based weapon that is less effective than the crystal gun, but reliable enough that a player could use it in a firefight. The developers of *Star Wars: Republic Commando* had a similar issue in their weapon system and adding in an unlimited ammo overheating pistol seemed to solve the issue, so we decided to implement the same solution.

An important fact to note about the pistol is that it does not interact with the crystal system. If we were to include the pistol in the crystal system it would destroy some of the simplicity in the sandbox we were seeking: it would add a full four new guns (if only one crystal was slotted) as opposed to one. As the weapon system seeks to simplify and not complicate our weapon sandbox it was determined that one firearm affected in such a manner would suffice.

4.3.4.3 Color Blindness

As the crystal gun’s central mechanic revolves around slotting crystals into one gun, we didn’t want to only have one crystal silhouette and multiple colors to identify the different crystal types. We decided that each crystal needs to have a unique silhouette, color, and purpose. Not only does this help players with color blind issues, but also reinforces each crystal's unique contribution to the weapon sandbox, making it easier for players to identify the crystals and their role during gameplay.

4.3.4.4 Narrative through Mechanics

We struggled with the narrative for several meetings before we began to get a real handle on it. Narratively, *Crystalline* pulls heavily from *Colony Crisis* taking cues from the resource contention that characterized it. Effectively, *Crystalline* is about a struggle between two spacefaring societies attempting to harvest the same planet for life sustaining crystals in short supply. It became our challenge to represent these narrative elements through gameplay.

First and foremost, it was necessary to determine some means to express that these factions were spacefarers. Through some blue skying it was determined that we would have low orbit space ships in our skybox that would be collecting energy from the crystal collection drills on the map via energy pulses. Additionally, escape pod respawns were proposed to not only give players an overview of the map, but the impression that they were dropping to the battlefield from space.

Despite both the crystal gun and resource aggregation mechanics being responsible for crystal element of the narrative, we found it important to reinforce the importance of crystals through other means. To express the intent of the drills that players were tasked with capturing we spawned the crystal pickups on the exterior of the drill. We also determined that the Heads-up Display (HUD) should explicitly outline the current crystal resource totals for both teams, as well as the currently equipped crystals to further outline the importance of the crystals in a heavier handed manner.

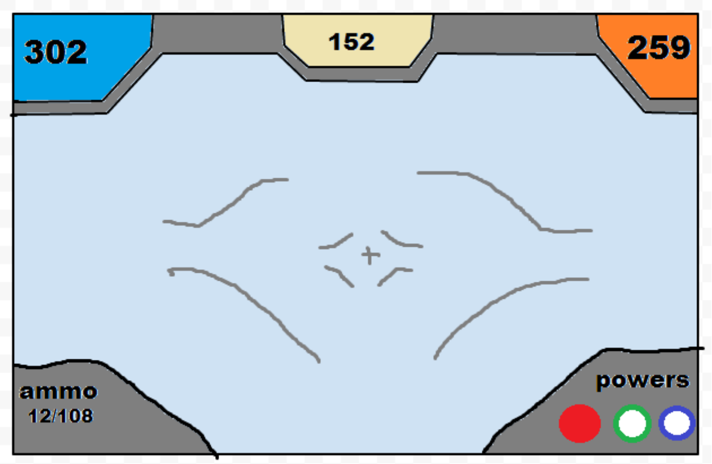


Figure 49 Heads Up Display Mockup

4.4 Tools

To facilitate production of a large scale project like *Crystalline*, we took advantage of a number of tools and resources. These primarily fell into two categories, one for organizing the project itself, and one for organizing our team and relevant documents. As our design process evolved so too did our tools of choice in an effort to maximize our productivity or remedy communication issues. Regular, frequent use of these or similar tools and making sure everyone accepts and understands them is essential to maintaining the development process.

4.4.1 Google Drive

Out of familiarity and ease of use, the first tool we turned to was Google Drive. It allows us to organize our documents, schedules and resources, as well as work simultaneously on the same files, such as this design document or our pitch presentations. This was our core resource for storing all of our work except for the game itself.

4.4.2 Source Control

The game itself and the various prototypes were stored on a Bitbucket repository utilizing Git for source control for most of development time. We used Atlassian SourceTree as our interface for the repo. Team members had a positive experience with Gitflow, a feature of SourceTree, and so we chose to use it in our project. Gitflow involves keeping two main branches, called ‘master’ and ‘develop’. The ‘master’ branch is where final stable builds of the game should be, and ‘develop’ is where new work gets pushed to. From the develop branch, individuals make ‘feature’ branches for specific features in the game, like branches for ‘weapons’, ‘character’, etc. These branches then merge into develop as the features are completed. The advantages of this process are that the develop and master branches remain relatively stable and individuals can more easily interact with the build of the game on develop, and using feature branches makes it easy for people to switch what they are currently working on for whatever reason.

For approximately the last month of development, we switched from using Git and SourceTree to store our game repository to Perforce for storage capacity reasons. Though it took a little while to get used to, we eventually found it to be roughly as usable as Git. One advantage to using Perforce was its easy integration with Unreal Engine. By being able to check out files directly in-editor, it saves a lot of time with tracking down files in a separate window and adding them to the change list. In addition, the Perforce system handles binary files a little better, partly thanks to its ability to check out files so that only one person is editing it at a time. However, because the switch to Perforce was made so late in the process, we were not able to take full advantage of all the features Perforce has to offer.

4.4.3 Trello

To facilitate our team's organization and process, we decided to use Trello. Trello is a web service for creating and sharing to do lists and reminders. We tried other tools to perform a similar task but found this one to be the easiest to use. Using Trello, developers are able to create multiple lists of things to do, organising them by category, then moving them to 'completed' as tasks are finished. The items on the lists are called 'cards' by Trello and can be moved between lists by simply clicking and dragging, such as when someone completes a task and moves it to the 'Done' list. Ideally we can look at it at any time to see what people are currently doing and what is left to be done, and any other comments or activity.

4.4.4 Engine

Choosing the technology to build our game on was not an easy task. Originally, the group was using a custom from-scratch Direct X 11 Engine written in C++, begun in a previous class, where we made a game called 'Alpha'. It included features such as model importing, hardware instancing, frameworks for post processing and particle systems, and tessellated terrain, among others. We had planned to improve and use this engine for our capstone project, aware that the challenge of making an engine meant the tech would be the shining point and the final game would have to be less ambitious.

Over the summer and in the early months of Capstone, we implemented transform-parenting to support a full scene hierarchy system, several debugging and logging features, a line rendering and billboarding tool, integrated Havok Physics, and began on a networking framework in the Alpha Engine. Meanwhile, we began prototyping the game in Unity because we had a lot of experience using it for rapid prototyping. However over the year several major updates to existing engines came out and we constantly debated whether we wanted to work on our own engine or if we wanted to use an existing one and focus more on gameplay. When our entire group took a class that used Unreal Engine 4, we decided from seeing all of its features that we would use that for our final development in the second semester instead of continuing with our own.

Crystalline began using Unreal Engine version 4.5. During development Unreal released versions 4.6 and 4.7, which we upgraded to after carefully testing to see if it would break the game in any way. The completed game uses Unreal Engine 4.7.6.

4.5 Schedule

In general, we used the first semester of Capstone to design and prototype our game and the second semester to develop and polish it. Our primary development milestones included:

1. Finish deciding on our idea by October
2. Finish Prototyping by December
3. Have a playable game by the 2015 Game Developers Conference (early March)
4. Have a finished, but not necessarily a bug-free and completely polished game by the middle of April
5. Finish polishing for Imagine RIT and the RPI Game Festival May 2.
6. Finish working on this document and our defense presentation by the middle of May

In the beginning we tried to anticipate and schedule what we would be doing in great detail. However as time went on, our plans changed and tasks scheduled far in advance no longer applied. We also stretched our schedule and continued making changes and fixes long after our original ‘finish’ date of May 2. But changes and problems are to be expected, and having the plan at all helped us to not to get too far off track.

4.6 Design Process

Our overall design process was agile and iterative to meet the natural demands of an evolving game concept. However, we did not adopt a specific named process like Scrum.

In the very beginning of the project we decided on a ‘meeting leader’ role and a ‘note taker’ role which would be rotated every meeting. This worked for maybe two weeks before it became too much to keep track of. We started having regular meetings every Monday, Wednesday, and Friday afternoons to discuss design while otherwise working on the Alpha Engine as we knew it would need a few more basic features to prepare for developing our game.

When we realized that our initial idea was too much, we decided to split up into pairs and do two rapid prototypes and show them to each other to get more ideas on the table. We designed both of these by writing up a short document about their gameplay and features and then prototyping in Unity. When we revealed them, both groups were naturally attached to their own and we stalled for a bit. We began iterating on them to see if we could reach a point where everyone was happy, but this point did not come. Eventually we had a major meeting where we sat down and reviewed all the ideas we had ever had. We basically took elements we liked from all of them and created our final idea, *Crystalline*, and began designing it in more detail and prototyping it.

We then began a somewhat successful process of having meetings around the whiteboard in the lab, writing our ideas together, and then taking a picture of it. One person would then volunteer or be asked to type of these notes into design one sheets on the topics they covered. Ideally this one sheet would be reviewed at the next meeting, iterated upon, and then someone would prototype the concept in question. Once they were finished, it would be reviewed and iterated upon. It was around this time that we began using the online organization tool Trello to help us keep track of what people were writing or prototyping.

As the prototype was developed, the bulk of our time began to change from design to developing its features. As this happened, the frequency of our meetings and our use of Trello began to suffer. Around the same time we had stopped working on the Alpha Engine at all to put our time into the design and prototype, and so the decision was made to use Unreal Engine 4 instead.

When the second semester began we started our primary development on *Crystalline* in a fresh UE4 project. At this point the four of us began to focus on four major tasks that needed completing corresponding to our personal passions. These were the weapon mechanics and sandbox, the level design, the player character animation and other graphics systems, and the networking and menu system. Each of these took a lot of learning and work to get into a usable state. This caused us to work very independently and may have been the weakest part of our process as a team. We didn't establish a regular meeting time and had stopped using Trello to organize who was doing what.

Once we reached a point where these four features needed integrating and other features needed implementing, we realized that we needed to pull ourselves together as a team. We accepted that we had let things slip and agreed to begin having daily meetings and review what each person was doing on Trello at each meeting. In this way *Crystalline* began taking its final shape.

At the same time we also agreed to do more frequent builds of the game to test among the 4 of us or ask other teams around the lab to help us do an 8 player test. When we began having important milestones come up at which to demonstrate our game, we began to discuss the game in terms of what problems were most important to fix based on our playtests. Instead of our regular meetings we would point out what we each think is the biggest issue and then agree on the biggest problems as a team. Then one person would take responsibility for each issue and we would begin a sprint to work on them. We would then discuss everyone's progress on their assigned issues at the next meeting. This was largely successful in getting our game ready for major events such as Imagine RIT and the RPI GameFest. After these events we considered the game more or less finished, and continued doing playtests to find areas in need of polish.

5. Game Design

Crystalline is a competitive, 3D, 8 player networked, First Person Shooter, where the run and gun action focuses around modifying your gun for specific situations. This game runs in the Unreal Engine for the PC platform and is geared towards FPS fans. It is set in a distant sci-fi setting reminiscent of *Star Trek*, *Star Wars*, and *Firefly*.

In the beginning of the second semester of Crystalline development, we shifted our design from a unique combination of strategy and FPS to just FPS with a polished weapons system. This overall change in game design was due to a number of factors. One major constraint which prompted this change was our lack of time. We had decided to switch the engine of choice to develop this game in from the Unity Engine to the Unreal Engine. This meant that we were starting from scratch in regards to making a game, which gave us more to do in the same amount of time. As a result, we needed to cut features so that Crystalline would be within our scope. The MOBA-oriented aspects of our game were then removed, as well as the ability to combine weapon crystals. Cutting these features helped keep our project in scope enough to feasibly create a polished product at the end of development time.

One other reason why our goals changed is our desire to have a novel game, but not a game that was over-scoped. All of the features we cut were cut so that we were able to focus on refining our core concept, instead of creating an unpolished game in broad scopes, which is what our prototype was. By shifting our focus from showcasing a novel mixture of strategy and FPS to showcasing a novel, polished weapon system, we were able to create a better game in the end.

5.1 Features

Crystalline is a full featured first person shooter created in UE4.

- Networked multiplayer.
- Diverse weapon sandbox.
- Multiple game types.
- Mouse and keyboard support.

5.2 Game Narrative

In a time when living on Earth is a thing of the past, humanity now roams the stars in large space fleets. Maintaining a supply of resources to sustain the fleet is no simple task. Periodically fleets need to land on exotic planets and harvest natural resources. The most essential resource of them all is the Arwenian space crystal, a crystal that forms in deep space on the surface of asteroids and deep within outer rim planets' surfaces. These crystals make deep space flight possible as the life support systems and engines of the space fleet run on energy produced from these crystals. Having a solid reserve of these crystals is important for the survival of the space fleet.

There are two major space fleets that roam the stars. One of the factions, The Blue Rings, a federalist autocracy, believes the crystals need to be collected and stored on the fleet, only to be distributed throughout the rim by a select few. Their goal is to maintain supply and ensure that everyone gets a fair and equal share. The other faction, The Orange Suns, an anarchic collective, believes the crystals belong to those risking their lives to harvest them from these harsh environments. Their goal is to ensure that freeloaders aren't rewarded and that only the strongest and bravest survive the frontier of space.

These two civilizations often meet during crystal harvesting operations. Inevitable conflict arises due to their difference in thought of how these crystal resources should be managed. Some of these conflicts become large battles that often turn into war.

5.3 Game Types

5.3.1 Game Type Descriptions

5.3.1.1 Shootout

Kill opponents to gain points. The first player to reach the point limit or the player with the most points when the match time runs out is the winner. This game mode is similar to the traditional "deathmatch" of First Person Shooters.

5.3.1.2 Lord of the Waterhole

There is only one hill zone on the map that never moves. Gain 1 point per second spent on the hill. First player to reach the point limit or the player with the most points when the match time runs out is the winner. This game mode is similar to the traditional "King of the Hill" of First Person Shooters.

5.3.1.3 Crazy Lord of the Waterhole

There is only one hill zone on the map that moves ever so often. Gain 1 point per second spent on the hill. First player to reach the point limit or the player with the most points when the match time runs out is the winner. This game mode is a variation of the traditional "King of the Hill" of First Person Shooters.

5.3.2 Game Options

Each game mode can be customized with options. The two options we offer to the users are "points to win" and "time limit." Both options have drop down menus that change and update based on the game mode selected.

5.4 Characters

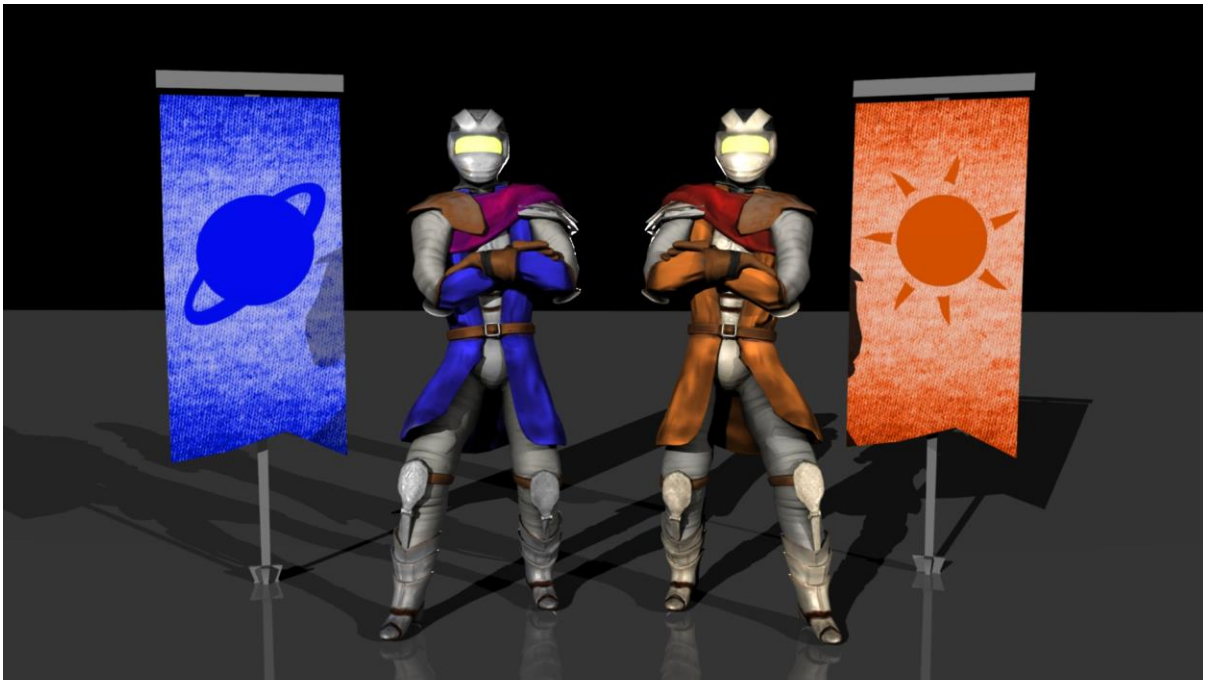


Figure 50 Player characters and faction emblems

5.4.1 Player Character

Crystalline's player troopers have a dusty western look over a sci-fi space base suit. This aesthetic is to push the combination of the two genres and reinforce the space western themes of rugged factions fighting over valuable crystal prospects on the space frontier.

The character's color can be chosen by players in the lobby. When a player is damaged, their color gets a shield overlay which indicates that they have been hit and how much health they have left. This is designed to provide diegetic feedback to players in addition to the HUD indicator upon scoring a hit.

5.4.2 Artificial Intelligence

In addition to the player controlled characters there is support for AI bots who can move and fight but cannot use the crystals to enhance their Crystal Gun. While our focus was on player versus player combat, we added support for bots to make it easier for the game to demonstrated or played with fewer players. The bots can only use the basic assault rifle, and they do not attempt to go towards the target in the King of the Hill game modes. The bots take advantage of Unreal’s blackboard and state machine systems. These utilize a precomputed navigation mesh based on the static geometry in the scene.

The basic operation of the bots is to loop through the available valid opponents and choose the nearest one to start moving towards. Within a certain range they begin firing. Because they were originally too difficult, they were changed to fire in 2-3 second bursts and with a higher degree of spread on the bullets than on a player gun. When a player spawns in the tutorial room, they hit an ‘AI Immunity Trigger’ which marks them as invalid targets. Upon exiting the tutorial rooms they are changed back to valid targets. This prevents bots from targeting a player in the tutorial room and camping outside the door. Bots cannot spawn in the tutorial room and skip straight to the outdoor spawn points.

5.5 Weapon Sandbox

There are two weapons in the game, the Crystal Gun and the Pistol.

5.5.1 Crystal Gun

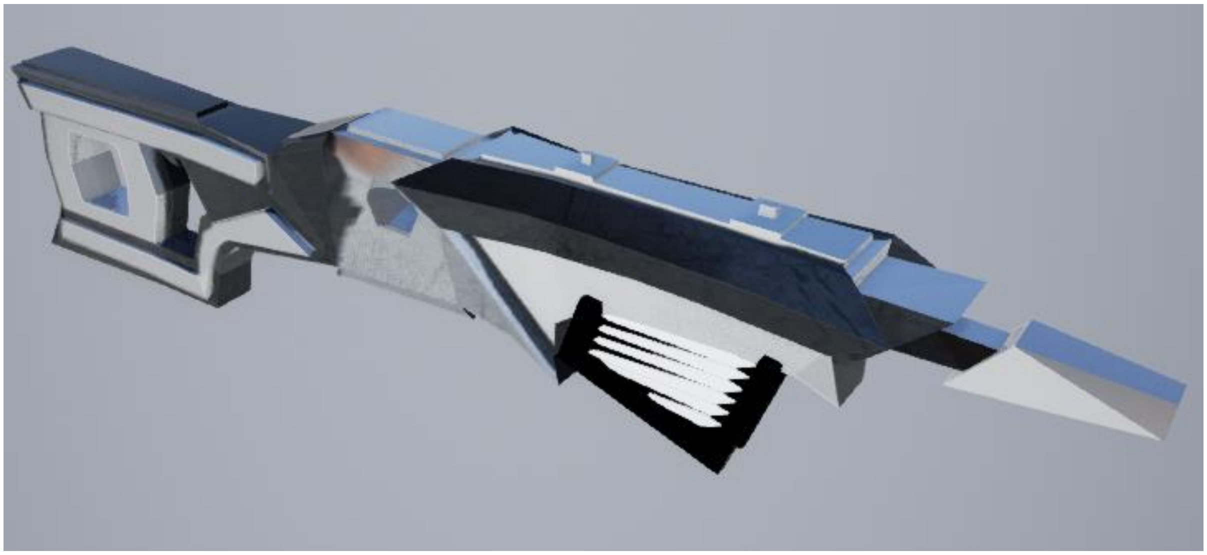


Figure 51 Crystal Gun

The Crystal Gun is *Crystalline's* primary weapon. The weapon is characterized by its fully automatic fire with minimal spread. Headshots are rewarded at a rate of 1.5 times base damage and operates at a short to medium range. Sustained fire from the core Crystal Gun should result in a kill after about half of the clip is exhausted.

The Crystal Gun's core functionality lies in its ability to be modified for distinct roles on the battlefield. This modification is carried out through the pushing and popping of crystals found in the game world. As the gun does not change ammo is shared between all four iterations of the Crystal Gun (Core, Power, Precision, Energy) and consumed at rates dependent upon factors relating to the modifier. The default Crystal Gun uses minimal ammo as it plays to a more "spray and pray" play style in comparison to its modifications.

5.5.1.1 Power Modifier

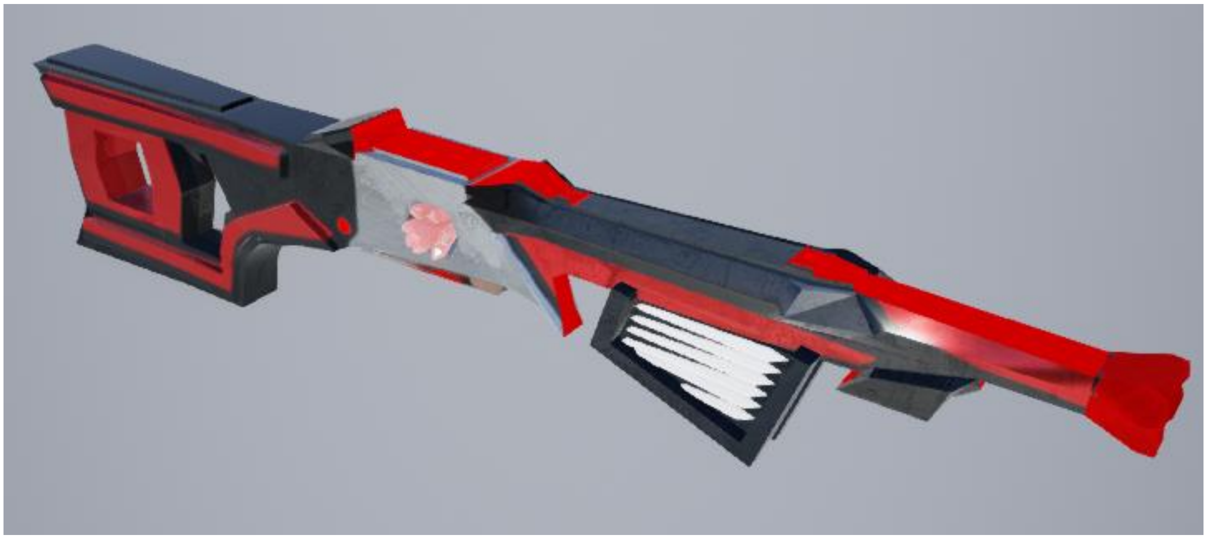


Figure 52 Crystal Gun with Power Modifier

The Power Modifier for the Crystal Gun is best compared to a "Haymaker Punch": devastating when it hits an opponent, but forces the player into a dangerous situation. When the Power Crystal is slotted into the Crystal Gun the fire rate of the gun is reduced to semi-automatic, damage is greatly increased, range is far shorter, and it has a far greater ammo consumption. Additionally, each shot now fires a spread of bullets that land within a conical region emitted from the barrel of the weapon. This modifier is catered to short range engagements and is typically beaten by the Precision Modifier when the engagement starts at long range.

5.5.1.2 Precision Modifier

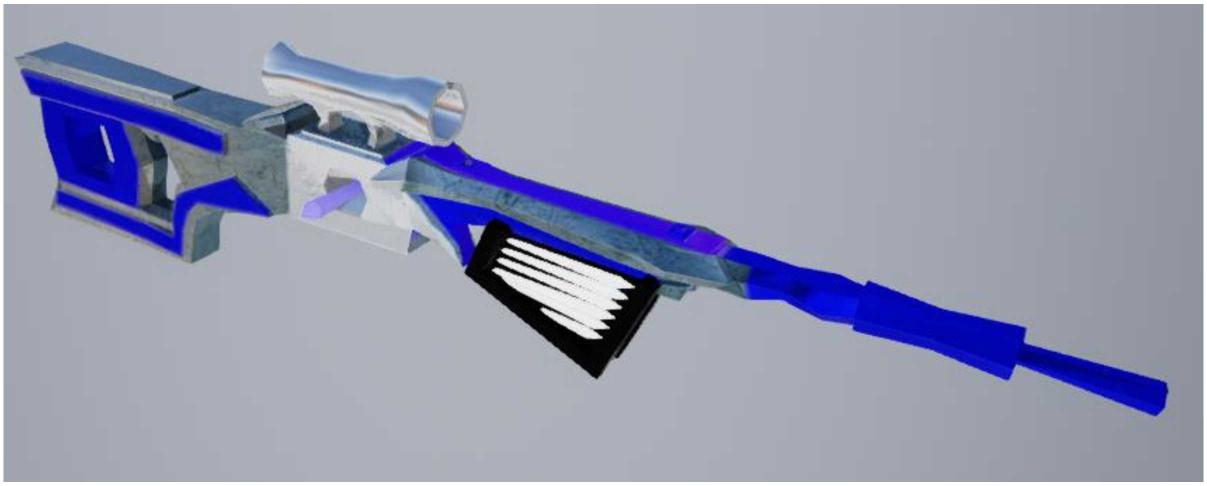


Figure 53 Crystal Gun with Precision Modifier

The Precision Modifier for the Crystal Gun is best compared to a precision strike to a pressure point, valuing accuracy over rate of fire. When the Precision Modifier is slotted into the Crystal Gun the fire rate is reduced to semi auto, damage is increased, the effective range extended, and the ammo consumption moderately increased. Headshots are rewarded at a rate of 2 times base damage for this modifier and encouraged. The modifier also has a scope feature allowing the player to better place long distance shots. This modifier is best suited for open spaces and long distances, typically losing to the Power Modifier when the engagement starts at short range.

5.5.1.3 Energy Modifier

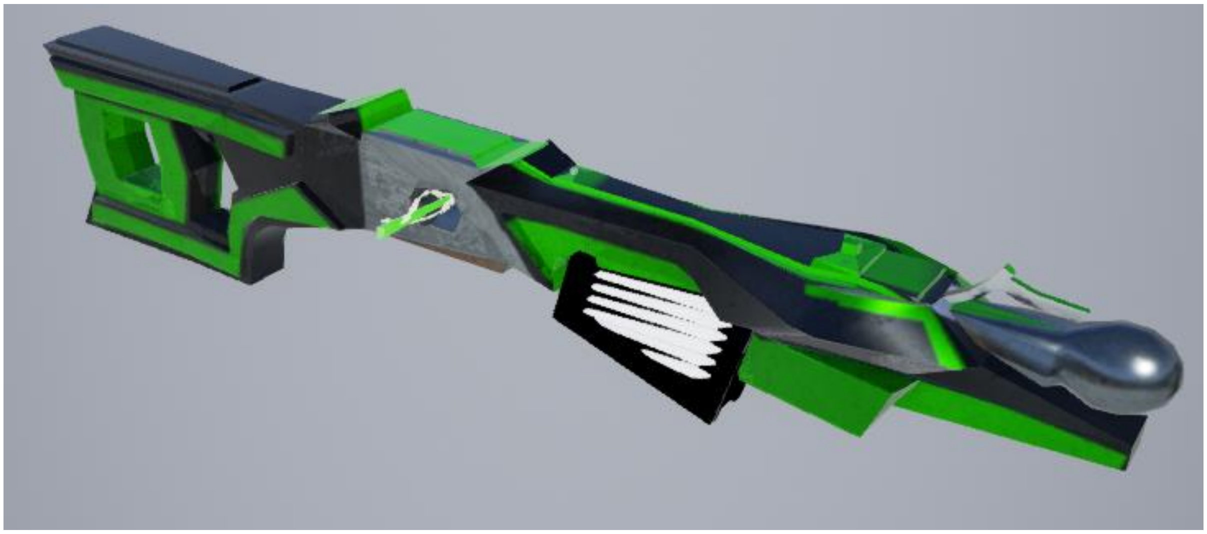


Figure 54 Crystal Gun with Energy Modifier

The Energy Modifier for the Crystal Gun is best compared to a flurry of blows: damage caused by individual strikes is less emphasized than the sheer number of times it hits. When the Energy Modifier is slotted into the Crystal Gun the fire rate is increased, damage is reduced, and the ammo consumption is reduced. The modifier also introduces a mechanic to the Crystal Gun, wherein if the player lands a successful hit, an energy beam connects between the target and the weapon barrel. So long as the angle between the target and line of fire remains with a defined conical region the player will deal damage while firing. This modifier is best suited for mid-range encounters typically only losing to skillful default Crystal Gun users.

5.5.2 Pistol

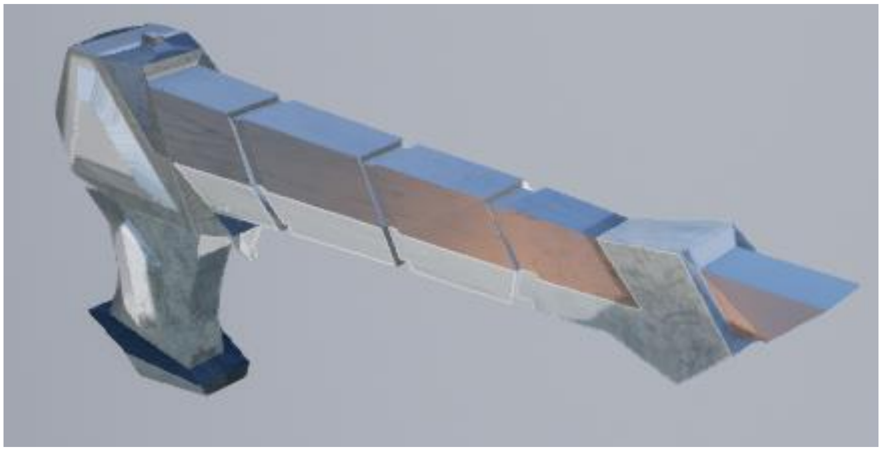


Figure 55 Pistol

Crystalline's Pistol is a fallback secondary firearm. Unlike the Crystal Gun, the Pistol has a heat based ammunition system. If used properly the player can shoot indefinitely with no penalty until they can acquire ammunition for the Crystal Gun. If the player shoots too much too quickly the "heat" gauge of the Pistol will max out and trigger a "cooldown" state that prevents the player from firing. If the player fires to overheat and lands every shot the Pistol is balanced to deal a lethal amount of damage. The Pistol is not capable of being upgraded.

5.6 Combat

The moment to moment gameplay of *Crystalline* is centered on gun play and combat. In a typical encounter with another player the determining whether their current modifier is appropriate for the engagement. If the modifier is deemed inappropriate the player will then either eject their modifier (if appropriate) or seek a new modifier in the immediate vicinity. While the player seeks the appropriate weapon configuration they will likely seek cover appropriate to the circumstances while taking shots at their opponent. The victor of the encounter will be determined on the basis of skillfulness. After completing such an encounter the players will repeat until a victor is determined by the rules of the game type.

5.7 Player Health

Each player has a regenerating shield that represents their overall health. When the player receives damage it is inflicted upon their shield. As the player's shield receives damage it is reflected both on the player's HUD and a world space shield surrounding the player. When the player's shield reaches zero the player's actual health may be damaged. The health of the player is not visible to the user and exists to allow the player time to hide when their shield zeroes out. If the player's health reaches zero the player has been killed, after being killed the player then respawns at a random spawn point on the map. When the player does not receive any damage after a set period of time their health is reset and the shield regenerates over a fixed period to its maximum value.

5.8 Multiplayer Environment

Crystal Canyon

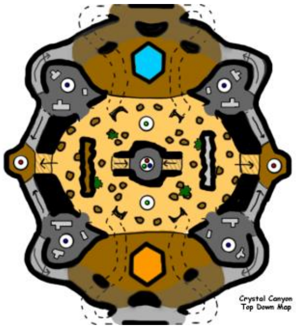
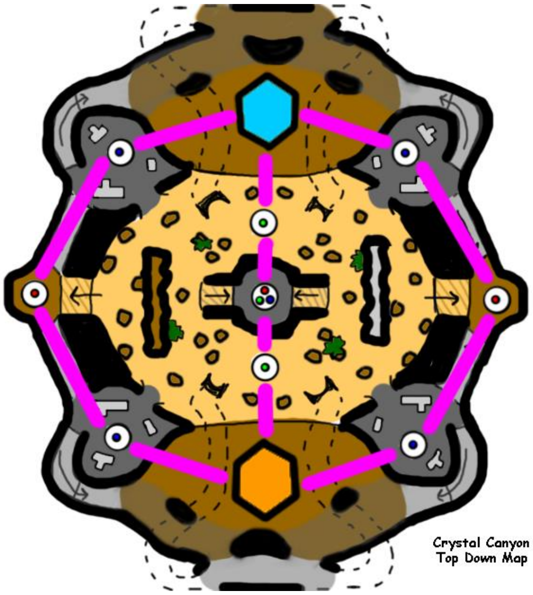
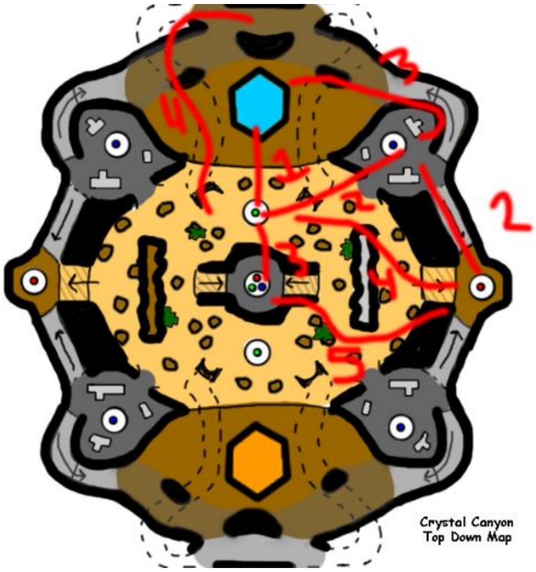
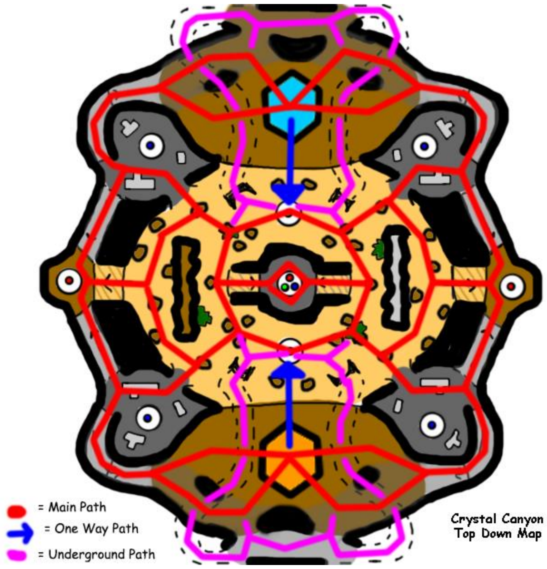
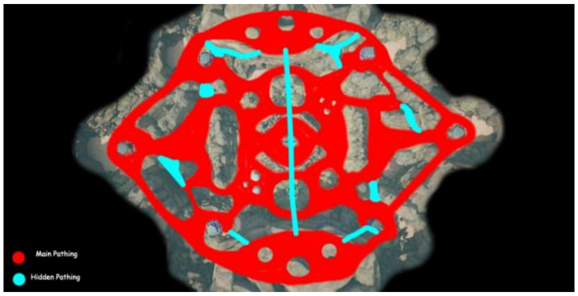
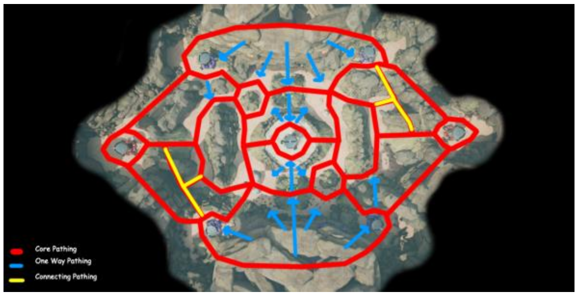
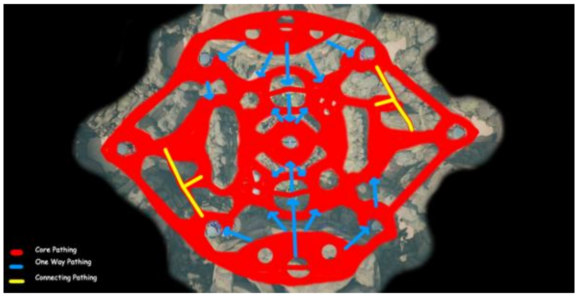


Figure 56 Standard map. Main showcase map for capstone. Designed for the conquest game type

This section is written in mind for the game design and level design choices of the multiplayer map of “Crystal Canyon” for information on the environment design in relation to visual design, then please see section 7.1.3 Environment Design. If you are looking for information on design of the map during iteration, then see section 8.4.1 Map Iteration.

The original high concept document for “Crystal Canyon” contained this information: ● Designed for the “Contention” gametype. ● Symmetric ● Fight on planet ● Team based ● Static designed map ● Medium to large map ● Narrative: Two teams fighting on the planet surface over crystal harvesting drill sites.	 *Figure 57 Representation of “Lanes” in pink. 3 lanes total: left, middle, and right..*

High concept of scale: - Relational unit scale. (time) - Triangular relational scale (positions)	 *Figure 58 Representation of "relationships" between areas*
First Pass Design of Player Flow in the Map	 = Main Path = One Way Path = Underground Path Crystal Canyon Top Down Map *Figure 59 Map flow pass one*
Diagram of area of play Red: Main Pathing Light Blue: Hidden Pathing	 *Figure 60 Map flow pass two*

Core Pathing Diagram Red: Core Pathing Blue: One way pathing Yellow: Connecting Pathing	 *Figure 61 Map flow pass three*
Area of Play with Pathing Red: Area of play Blue: One way pathing Yellow: Secondary connecting Pathing	 *Figure 62 Area of Play*

The initial design layouts of Crystal Canyon were implemented around supporting the “Contention” gametype, which never actually shipped with the game. During development, the map’s layout changed due to this. Since the only gametypes that the game supported were shootout, lord of the waterhole, and crazy lord of the waterhole, the map had to be iterated upon throughout development to ensure that the environment played well with these game types.

About 70% of the map’s initial design was carried over to the final version. Most of the changes were opening up more sections of the map, making areas less defensible, and easier to invade. Navigating between areas also became easier with creating more ways to get from section to section. Some major goals here was to one “make sure there are fun interesting spots for one player to shoot and engage another player in gun combat,” two “make sure there are smaller areas on the overall map for players to use specific weapons and assume ‘roles’ ,” three “make sure the ‘hill zones’ are fun to defend and fun to attack, ” and four “make sure that the map design complements the weapon sandbox and the game modes, and that the design never conflicts with either of these things.” Finally, due to the nature of the gametypes being FFA only and mostly related to deathmatch in essence, there was more of a subliminal design to make many loops in the core pathing of the environment

The last section that influenced the final design was the modular art set that we purchased on the UE4 marketplace. This art set gave us the opportunity to create some extra “flair” areas to help make gameplay more fun. These areas were implemented due to seeing the art pieces, and having ideas along the lines of, “Oh, wow, if we put this piece here, and rotate it like this, then it would create this cool opportunity for interesting gameplay because it gives the player the opportunity to do this [cool thing]”.

5.9 Camera

The player views the world of *Crystalline* through a first person view. This view, a standard of the genre, allows the player to become their avatar and feel immersed in the combat. The camera moves with the player’s avatar. As the player moves the camera also moves, as to maintain the first person view.

5.10 Graphical User Interface

Crystalline's in game GUI went through several iterations over the course of development, each reflecting the core mechanics of gameplay and user feedback.

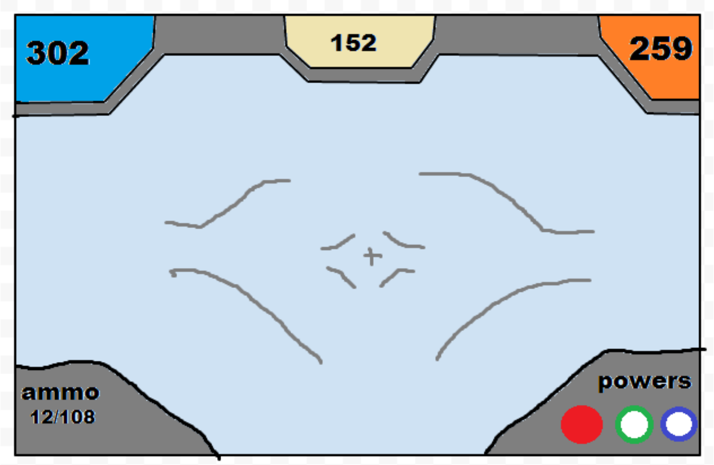


Figure 63 HUD Concept

The Primary iteration of the HUD emphasized the original game type, making the scores of each team extremely prominent in the left and right upper corners. At its core this design was utilitarian and slap dash for the sake of the prototype.

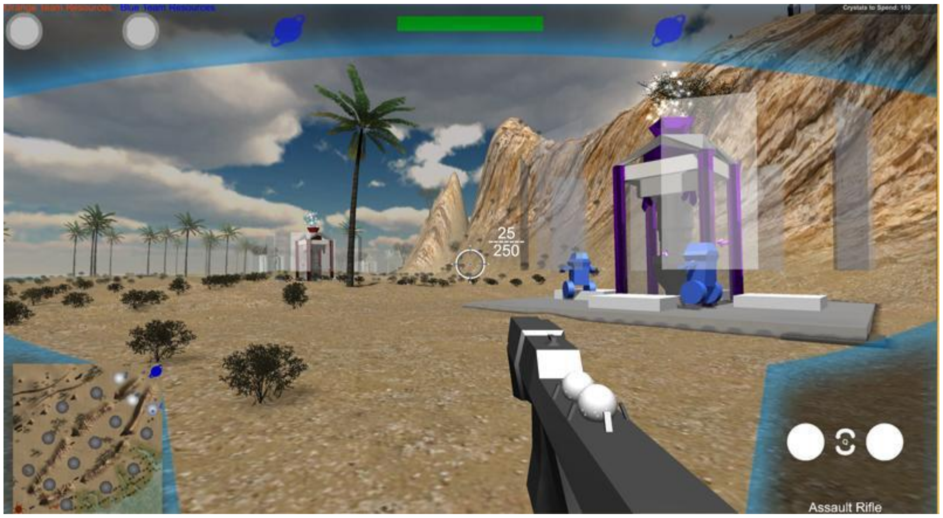


Figure 64 HUD Iteration 2

The second iteration of the HUD constituted the first in game implementation. Score indicators were moved to the upper left quadrant and replaced with circular indicators. This iteration also introduced a minimap and moved the ammo indicator to a more central location.

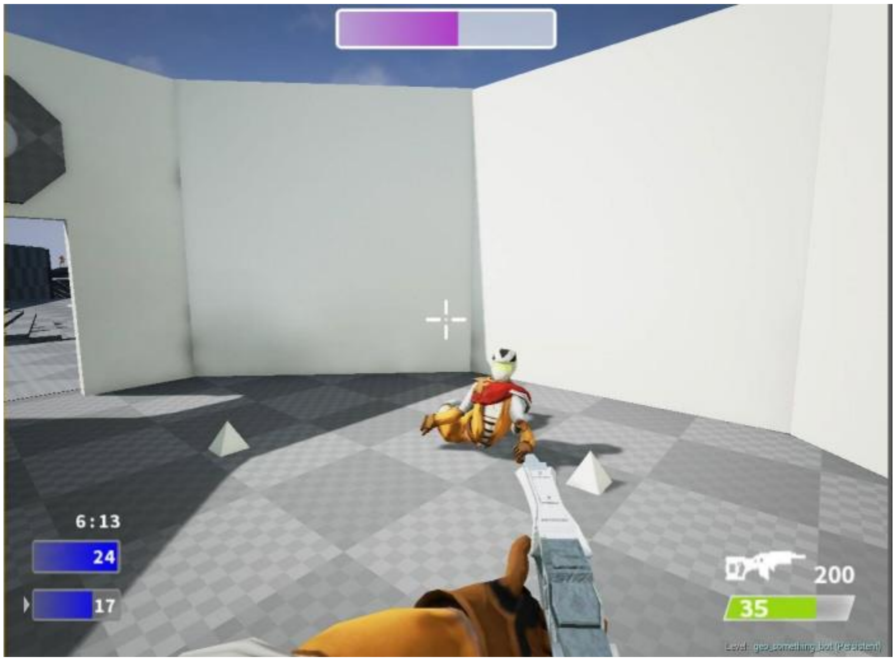
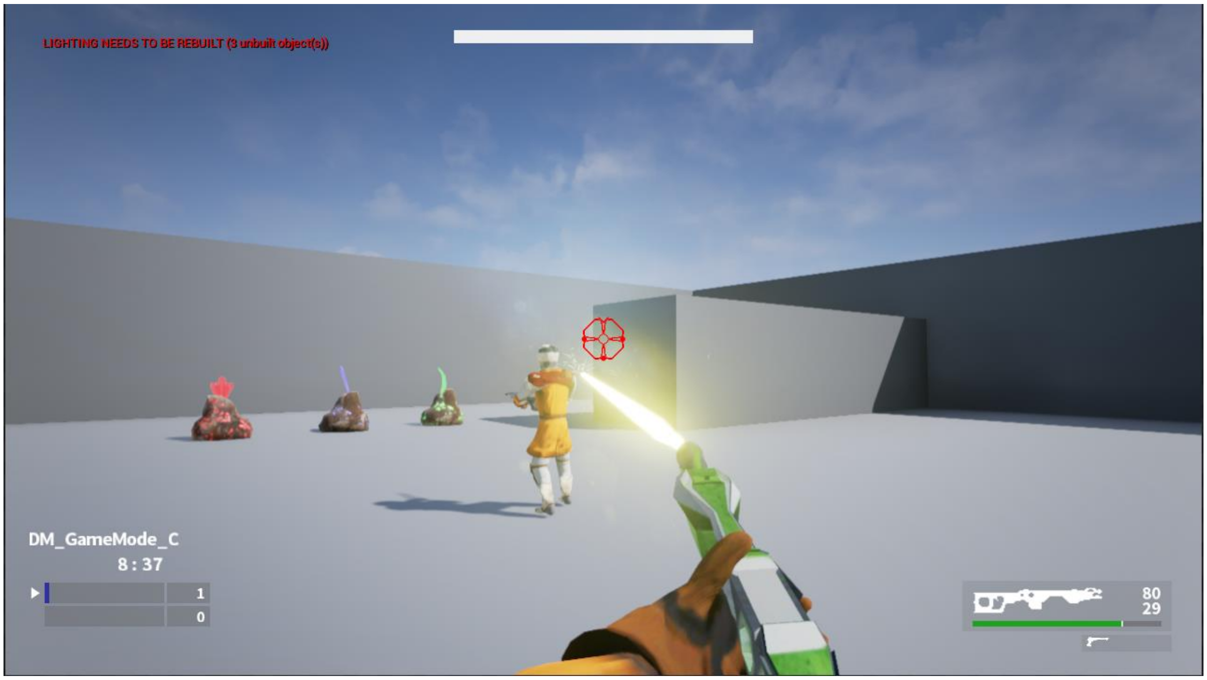


Figure 65 HUD Iteration 3

Upon transitioning to UE4 a new HUD was designed, placing an emphasis on rule of thirds and more cohesive visual elements. Several cues were drawn from games such as *Halo, Borderlands* and *Half-Life 2* in an effort to build off of semiotics that are more accessible to players of traditional First Person Shooters.



LIGHTING NEEDS TO BE REBUILT (3 unbuilt object(s))

Figure 66 HUD Iteration 4

This was primarily inspired by minimalist styles, abandoning gradients and moving the HUD elements out of the primary action of the player’s view. This iteration was a step up from the previous in terms of element cohesion, but a step backwards in terms of readability. Many players complained that the HUD elements were too far removed from where they were looking on the screen. Additionally, most players failed to recognize the shield bar’s purpose.

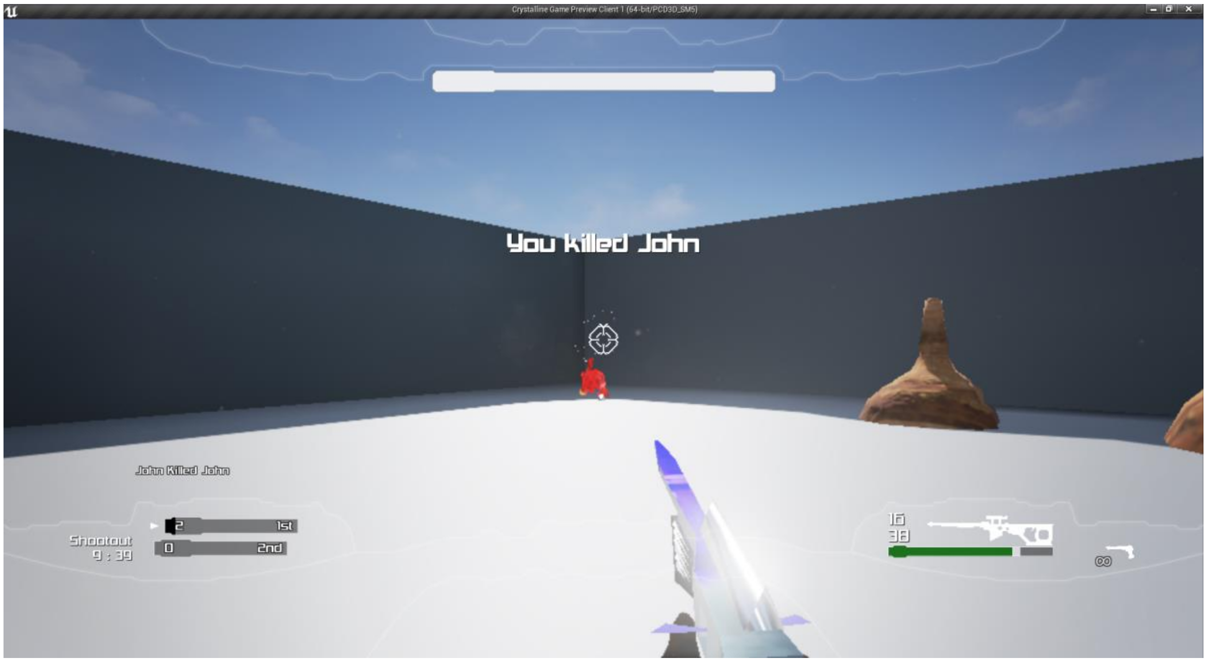


Figure 67 HUD Final Iteration

The final revision of the HUD abandoned minimalism in favor of a more **Meta** interface. Attempting to place the HUD back inside of the character's helmet a texture overlay was created for the helmet interior. Sections of the interior could be selectively flashed to draw attention to HUD elements that previously went unnoticed in playtests: the shield bar. This iteration also features a new message system, a feature missing from all prior iterations. Finally, this HUD was characterized by an improved widgeting system that allowed for rapid refinements. Further iteration would place the HUD into 3D space, shooting for a diegetic interface.

5.121 Control Scheme Summary

The control scheme is the genre established standard FPS controls. One the keyboard we have "WASD" for movement and strafing, "Space" for jumping, "Q" for last used weapon, "E" for use, "R" for reload, and "Tab" for score. On the mouse we have "mouse axis movement" for aiming, "right click" for zoom, "left click" for shoot, and "mouse scroll" for weapon swap.

We also have an alternative "lefty mode" on the keyboard that surprisingly majority of FPS games on the PC don't ship with by default. But it should be known that some PC games let players remap controls, and some console games have a lefty mode option.

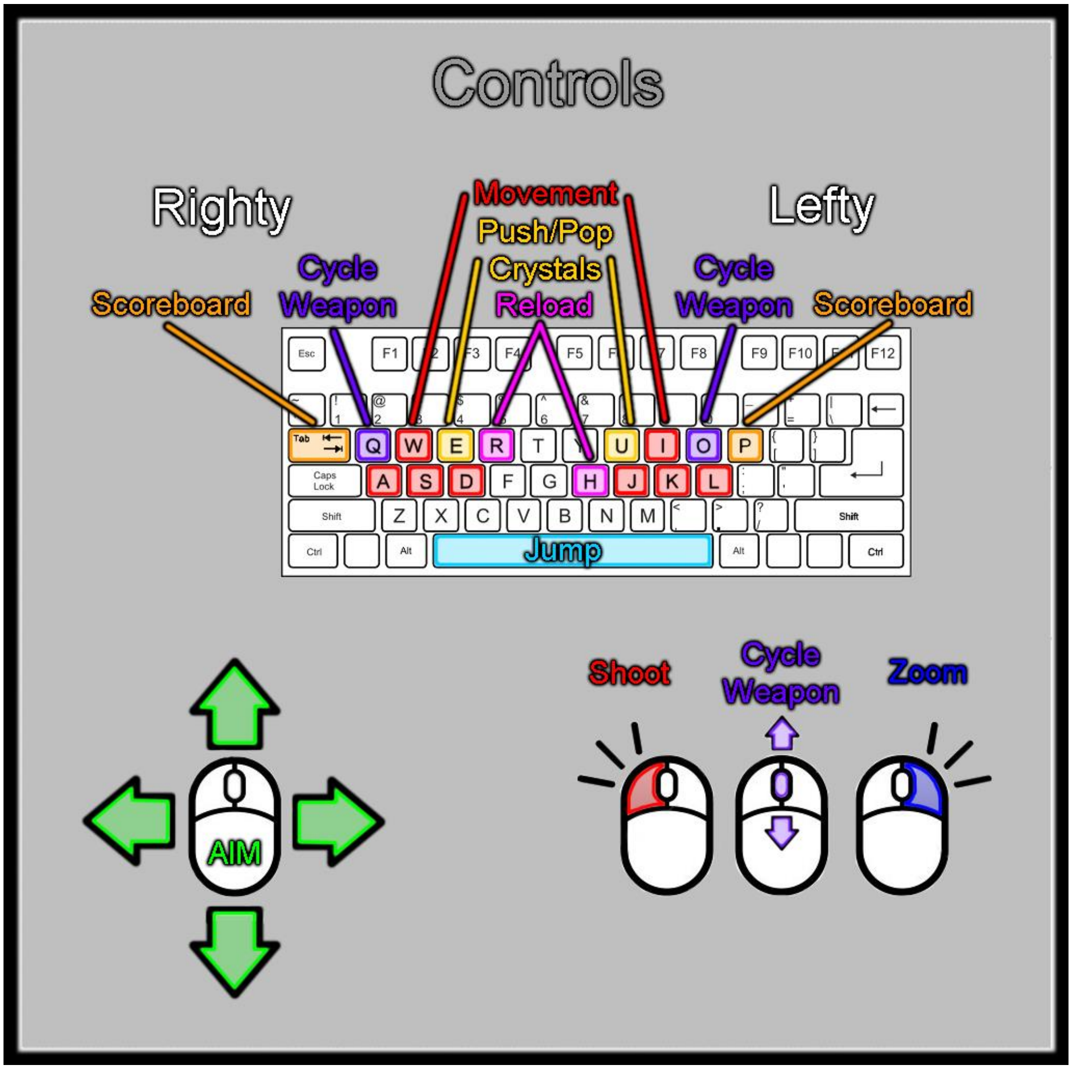


Figure 68 Controls

5.12 Player Personalities

All-around Alex

Alex prefers to be well rounded and prepared for all situations. Rather than specializing in a specific role Alex will try to play all roles simultaneously, or fall into where there is a gap in the team dynamic. Alex will usually stick to using the crystal gun without any modifiers because it has the best all-around average stats. Alex may also switch to a specific modifier if the situation calls for it, but Alex won't be afraid to pop out the crystal and switch back to the base crystal gun mid fight.

Headshot Harry

Harry likes to go for headshots. He'll use the precision modifier most of the time to help him achieve his goal. Harry would rather fail at getting a kill trying to get a headshot, than get a kill by landing body shots.

Newbie Nick

Nick is a new player not that familiar with the genre of First Person Shooters. It is initially difficult for him to master the keyboard and mouse controls, and often he fails to keep his target within his gun's sights. Due to this, Nick will default to using the energy modifier once he figures out that it has an aim assist lock on feature. Nick will continue to use this modifier as his go to choice, slowly branching out and experimenting with other modifiers until he becomes familiar and accustomed to PC FPS gameplay and controls.

Power Punch Paul

Paul likes to go for the "one hit kill". Because of this, he will default to using the power modifier for his crystal gun. The power modifier sacrifices range and ease of hitting successive enemies for more damage per shot. Due to this, Paul will insert the power modifier and run towards his enemies to try to finish them off in a single hit at close range.

Rushing Rihanna

Rihanna doesn’t really care about the game type or the points. She just wants to rush into the action and get her adrenaline pumping. She’ll grab a crystal modifier if it is along her path to the action, but she won’t go out of her way to pick up a modifier.

Sniping Sam

Sam prefers to stay on the outskirts of skirmishes and fights, picking off the wounded parties for an easy double kill. To help facilitate this playstyle Sam will use the precision modifier most of the time to both keep distance and get the final powerful kill shot.

Winning Wendy

Wendy doesn’t care about specific guns or play styles. She only cares about winning. She will use her map knowledge to get around and control her enemies. She will use the specific crystal modifier to give her the advantage in that particular situation. She will target first place players first because they pose the biggest threat to her winning, but will also go after weaker players because they provide easy points in the shootout game mode. In king of the hill, she will kill opponents who get in her way, but will use other players as a tool to help kill other players in the hill and wait for the perfect moment to strike to secure victory.

6. Technical Design

Technical resources can sometimes blur the line between editors and engines. The development team decided to use Unreal Engine 4 as the engine driving our capstone game, *Crystalline*. UE4 is half editor and half engine. The editor components gave the team full control of the play environment and play state. The editor component was a very helpful tool for designers and developers, as the team could do what they needed and see the results of their work instantly.

A potential problem with a combination editor/engine, like UE4 is, is the multi-user revision control. Many files in the editor are binary and not easily archived in version control. The team did eventually see these potential problems arise late into development.

When starting any new project it is important to determine appropriate scope. Engine scope is no different. Should the developer build an engine or purchase an existing engine? A good practice is to look at the core features of your game, and ask, "Can an existing engine handle my game's mechanics." Just because the engine is new and custom built, doesn't mean it is better. The game experience should be the driving force behind engine and technical design choices.

6.1 Target Platforms

Due to the overhead of targeting consoles or a litany of devices it was determined that our target platforms would be our personal machines and the lab machines as they were the most readily testable. Additionally, minimal effort was devoted to this task as Unreal handled the majority of platform specific details. Minimum and maximum system requirements have been estimated using similar projects in the Unreal Engine.

6.1.1 Minimum System Requirements

OS: Windows 7 64 bit
CPU: Dual-Core 2.0 GHz
RAM: 4GB
Graphics: DirectX(R) 11 Compatible Graphics Card.
Network: Broadband Internet Connection.
HDD Space: 2 GB

6.1.2 Maximum System Requirements

OS: Windows 7 64 bit
CPU: Quad-Core 2.4 GHz
RAM: 8GB
Graphics: DirectX(R) 11 Compatible Graphics Card.
Network: Broadband Internet Connection.
HDD Space: 2 GB

6.2 C++ Style

As *Crystalline* is a project of no small scale, it is necessary to follow a coherent style for code to remain legible. This style guide was derived from the approximate style guide that seemed to be used internally by Epic Games in an effort to make the code visually coherent. Time constraints and deadlines severely abridged this guide to the bare minimum:

File Names : *CG[Name].h* / *CG[Name].cpp*

e.g. *CGWeapon.h* / *CGWeapon.cpp*

Class Names : *UCG[Name].h* / *ACG[Name].cpp*

e.g. *ACGWeapon*

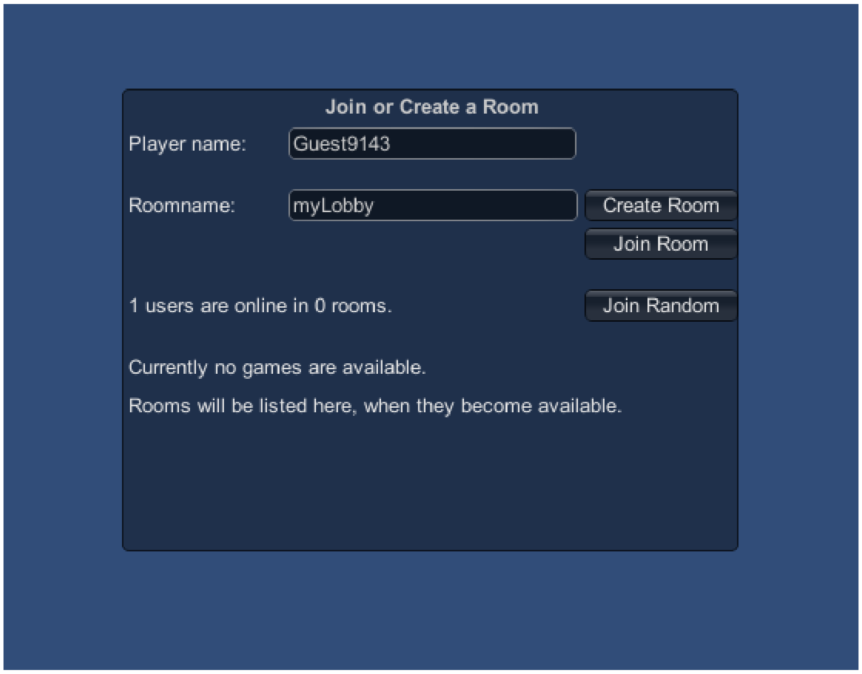
Function Name : First letter is upper case, followed by camel case.

e.g. *void FireWeapon()*

Member Variable Names : First letter is upper case, followed by camel case.

e.g. *uint32 HitDamage;*

6.3 Menu System



Join or Create a Room

Player name: Guest9143

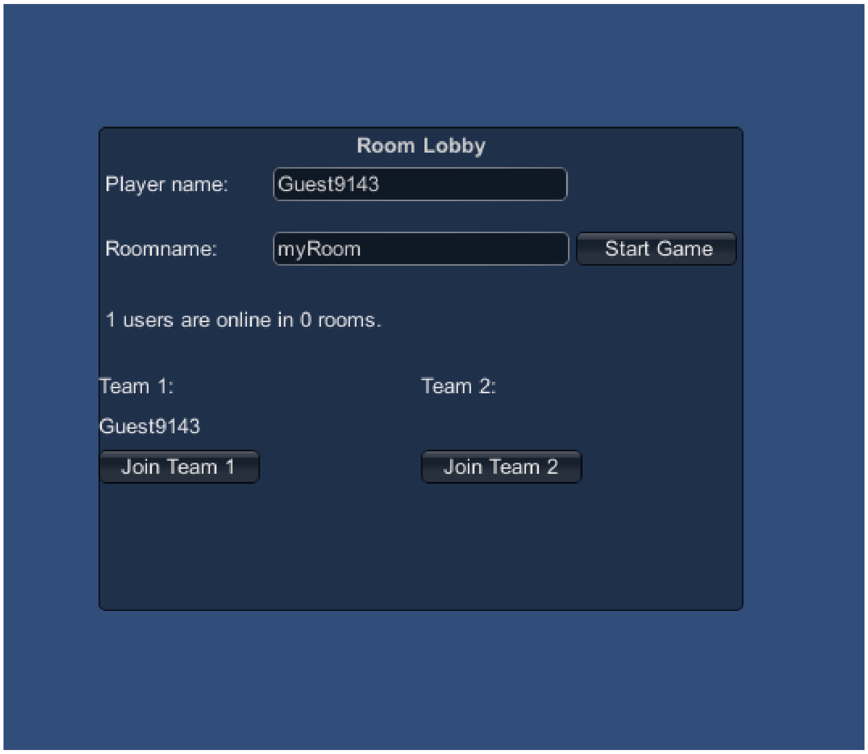
Roomname: myLobby

1 users are online in 0 rooms.

Currently no games are available.

Rooms will be listed here, when they become available.

Figure 69 Pregame Lobby: prototype 1



Room Lobby

Player name: Guest9143

Roomname: myRoom

1 users are online in 0 rooms.

Team 1: Guest9143

Team 2:

Figure 70 Pregame Lobby: prototype 2

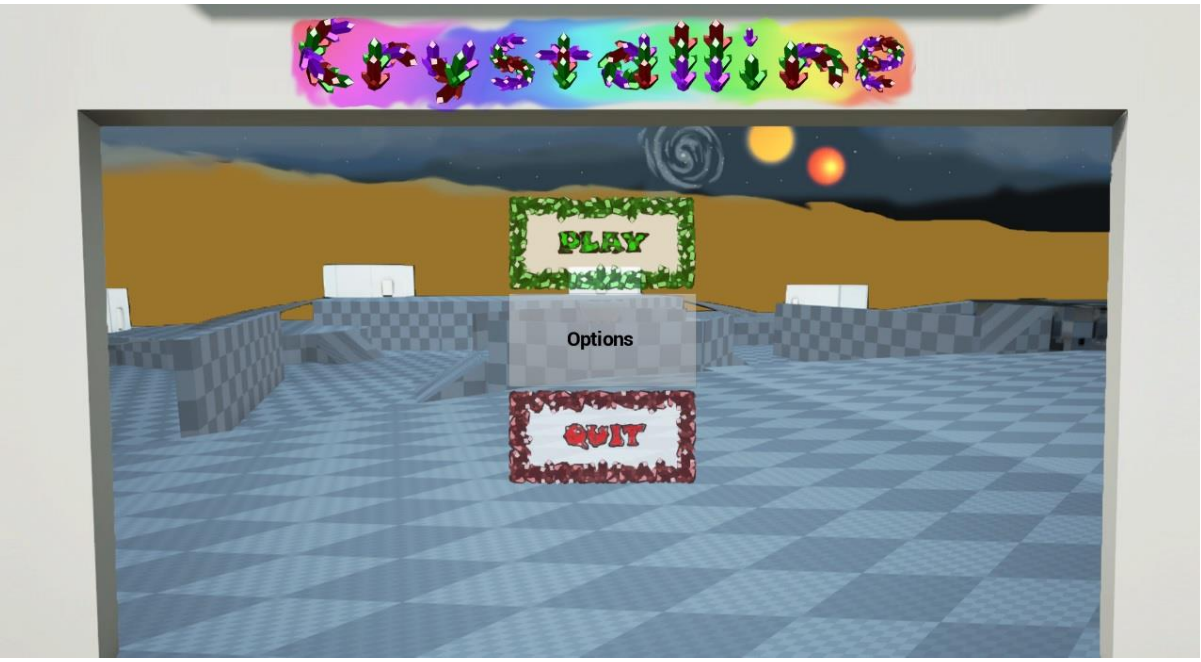


Figure 71 Development Main Menu Screen

Our menu system serves multiple goals in terms of game design. One main objective for our menu system is to give our first-time players a good first impression of our game. When our players first begin to play our game, we want them to get a clear vision of our game aesthetic and what the game will look and feel like. Another main objective of our menu system is to get our players into a game quickly and easily, without having to deal with any confusion or technical errors. This goal is where the technical design of our main menu system comes in.

6.3.1: Main Menu Level

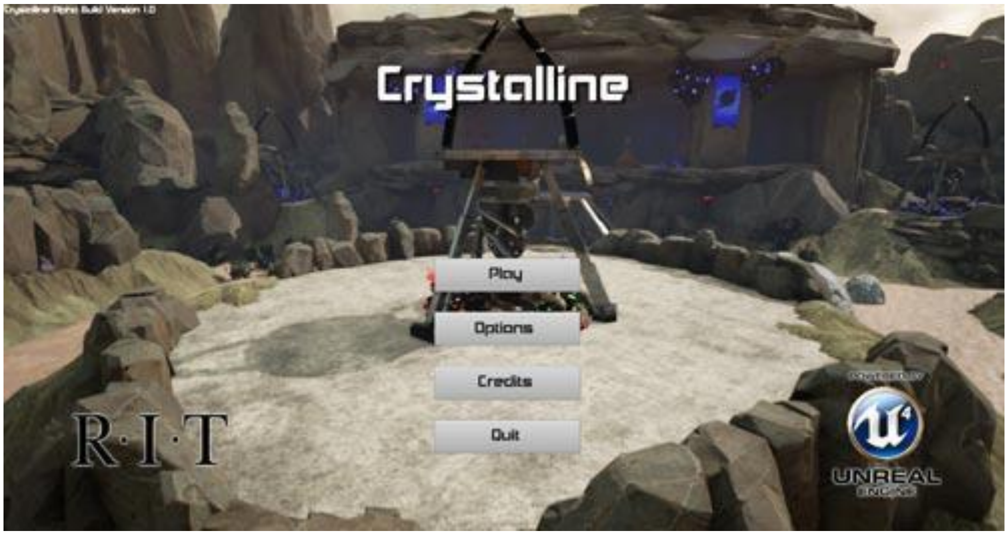


Figure 72 Final Main Menu

Our players run an Unreal map called *Main Menu Level* when the game begins. This provides a context for the system to know that the player is beginning the game, and will start in the "Title Screen" game state. The actual state in question that the current player is in is kept track of by an enumerator in our extension of the Unreal Game Instance class, called *ACGGameInstance* . For the *Main Menu Level* map, what game state the player is in correlates to which part of the main menu the player is currently looking at. The actual menu objects (items, buttons, etc.) that are displayed in our menu system are displayed via Unreal Motion Graphics Widgets (Or UMG Widgets). These Widget objects are held in the *ACGGameInstance* class, so that the *ACGGameInstance* class can switch which Widget object is being displayed to the players whenever necessary. Each screen shown to the player in the *Main Menu Level* map is a Widget held by the *ACGGameInstance* class. Whenever a player presses a button to switch what screen is shown, the *ACGGameInstance* class hides the first Widget and displays the second Widget. The interactions between these widgets in the *Main Menu Level* is shown visually in the diagram below.

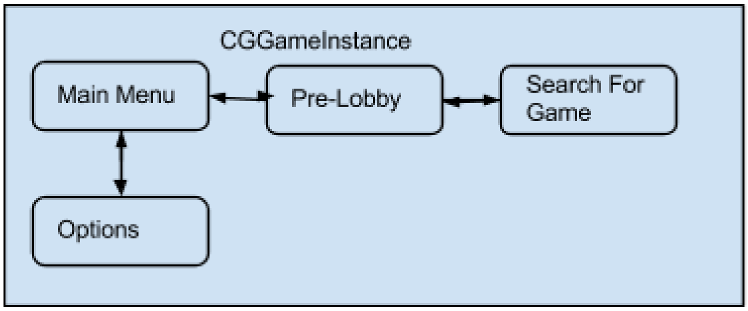
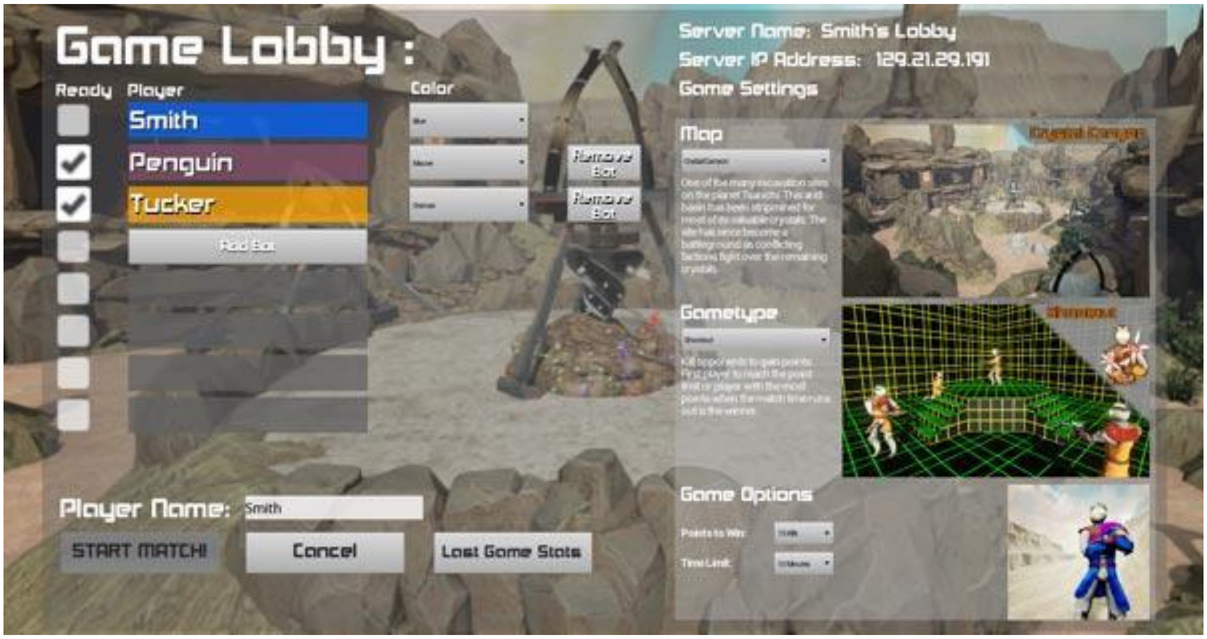


Figure 73 Diagram of Main Menu Level Interactions

6.3.2: Lobby Level



Smith
Penguin
Tucker

Player Name: Smith

Server Name: Smith's Lobby
Server IP Address: 129.21.29.191
Game Settings

Map

One of the many excavation sites on the planet [illegible]. This arid basin has been stripmined for most of its valuable crystals. The site has since become a battleground as conflicting factions fight over the remaining crystals.

Gametype

Kill opponents to gain points. First player to reach the point limit or player with the most points when the match time runs out is the winner.

Game Options

Points to Win:
Time Limit:

Figure 74 Lobby Screen

From the *Pre-Lobby* and the *Search For Game* screens, players are able to Host a Game and Join a Game, respectively. By hosting or joining a game, the player is then sent to the *Lobby Level* map from the *Main Menu Level* map. The main reason for this is that, from a technical design standpoint, the *Lobby Level* map is where most of the networking in the menu system takes place, whereas no networking is involved on the *Main Menu Level* map, with the exception of when you search for a game on the *Search For Game* screen. By dividing up the non-networked and networked portions of the Main Menu, we can better divide up our code from a design standpoint so that the non-networked sections don’t need to work with the multiplayer aspects of the game in mind. The logic of the *Lobby*

Level map is contained within a combination of a HUD class extended for the *Lobby Level* map, called *LobbyHUD*, and an extension of the Unreal Player Controller class, called *MenuPlayerController*. The *MenuPlayerController* class contained a majority of the code dealing with sending data back and forth, whereas the *LobbyHUD* class deals with showing the data on the screen. From the *Lobby Level* map, the client hosting the game is able to start the game by pressing the Start Game button, found on the UMG widget shown by *LobbyHUD*. By setting up a menu flow from Main Menu to Lobby to Game, we have a natural flow for the player, and gives us clearly organized areas for placing networked and non-networked classes and code.

6.4 Character

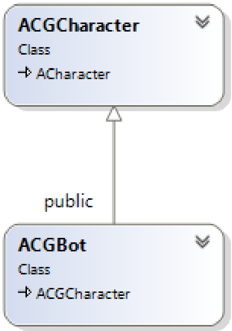


Figure 75 Generated UML for ACGCharacter

The *Unreal Engine* utilizes an actor based system for player control and representing the player in the world. In *Crystalline* the Player and Bot pawns are represented by *ACGCharacter* and *ACGBot* respectively. *ACGCharacter* manages all things directly tied to the state of the player character (e.g. shields) and maintains references to related subsystems.

ACGCharacter contains references to the following:

- Inventory of the player (Weapons on hand and crystals)
- The currently equipped weapon.

ACGCharacter defines the following behaviors:

- Receiving damage.
    - Reporting death to the Game Mode.
- Managing the character mesh and animations.
- Player Input (movement, looking, changing weapons, popping crystals, etc.)

ACGBot redefines several functions for bot specific responses.

6.5 Inventory

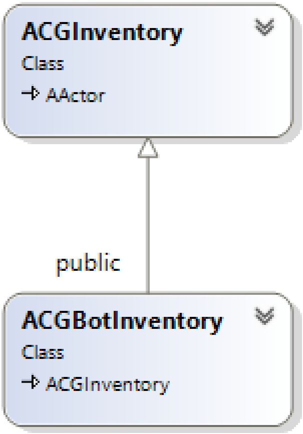


Figure 76 Generated UML for the ACGInventory

ACGInventory manages the weapons the player currently has access to and defines what happens when the player pushes or pops a crystal.

6.5.1 Inventory Construction

When the player first spawns all of their potential weapons are spawned as well. The Pistol *ACGWeapon* is maintained as a separate variable from the *ACGCrystalGun*s. When an *ACGCrystalGun* is spawned it is initialized to the default ammo amount and placed in an array, with the 0th index representing the base crystal gun. When all of the crystal guns are spawned the 0th gun is cached in the *CrystalGun* variable and the currently held crystal (*TierOneCrystal*) is updated to match the crystal of that weapon.

This implementation was adopted after several iterations for ease of maintenance, rather than raw space efficiency or speed.

6.5.2 Crystal Swapping

When the player triggers either a pop or push event on their weapon (assuming it's a crystal gun) the inventory will invoke its *ModCrystalGun* routine. This routine will first search the *CrystalGun* array for a gun with a crystal matching the target crystal, caching it. The crystal types are then compared and the appropriate animations are assigned to the pending gun and current gun. Finally, *TierOneCrystal* is updated to contain the new crystal and the *CrystalGun* variable receives the cached weapon.

6.6 Weapon System

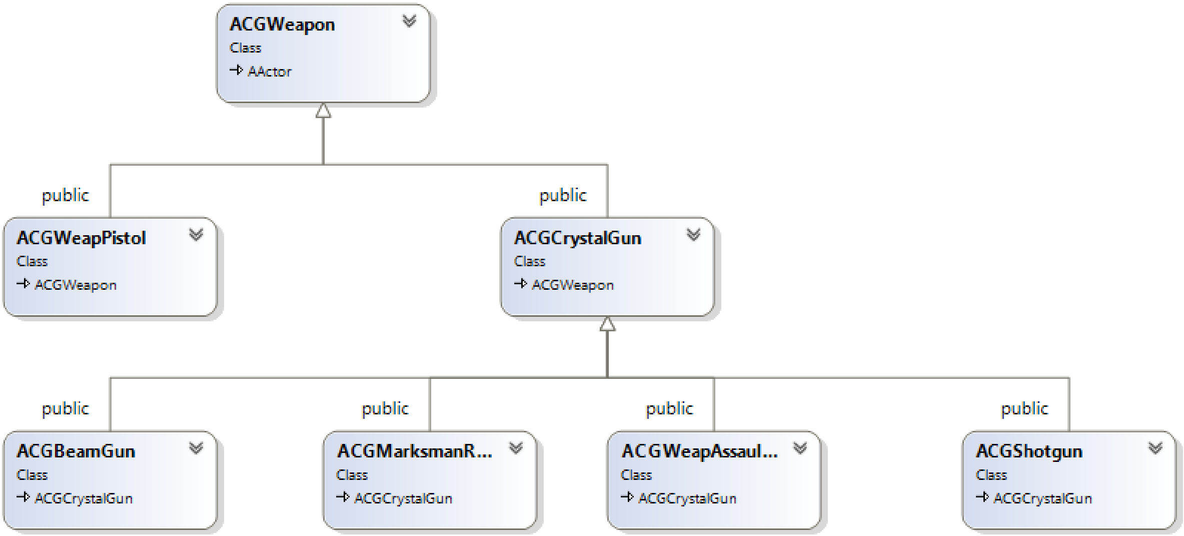


Figure 77 Generated UML for the Weapon System

Due to the nature of *Crystalline's* weapon system an inheritance hierarchy is utilized to prevent code from being duplicated and weapons such as the Pistol getting hooks for the crystal swapping mechanics. In short all weapons that are in the core weapon sandbox are children of *ACGCrystalGun*.

6.6.1 ACGWeapon

ACGWeapon defines the core functionality of the weapon system. It has networking for weapon fire (hitscan and projectile) and reload operations. Additionally, it defines hooks for ammo usage and getters that are specific to the weapon for UI feedback. Finally, *ACGWeapon* utilizes a state system to appropriately define when a player may and may not perform an action with a weapon, diagrammed below.

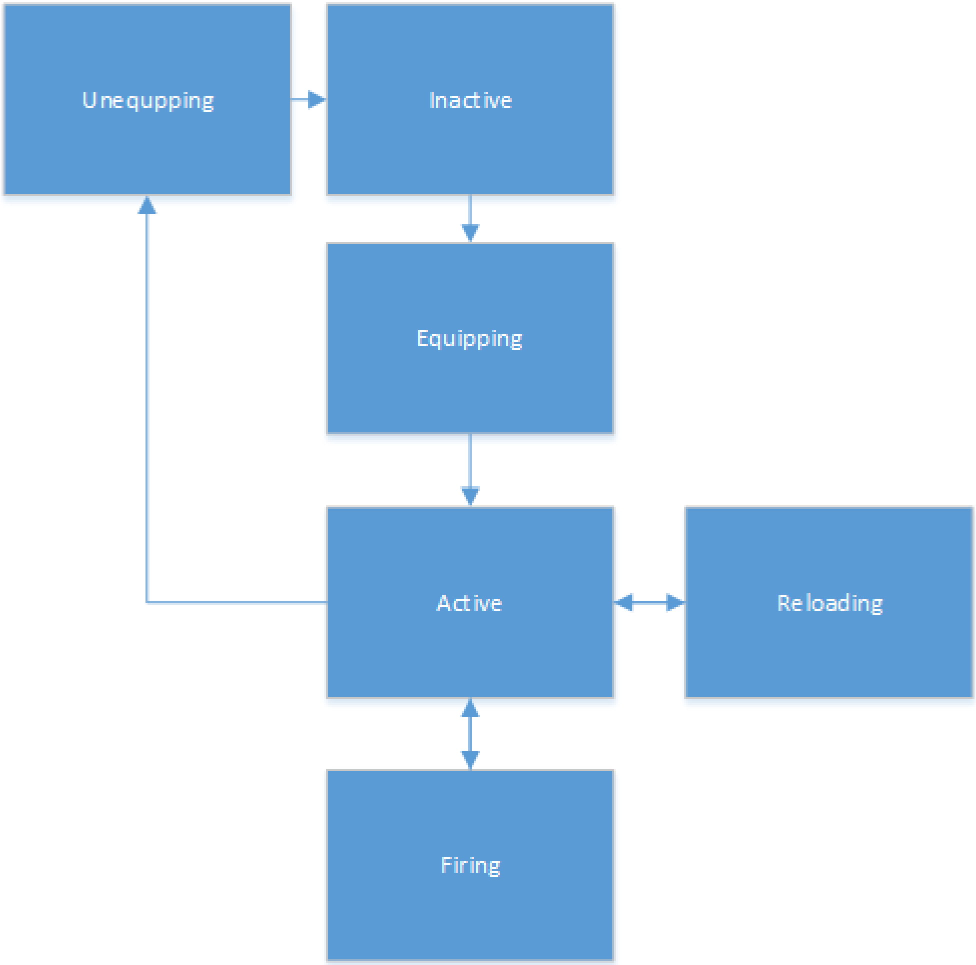


Figure 78 State Diagram for the weapon state machine

6.7 UI Widget System

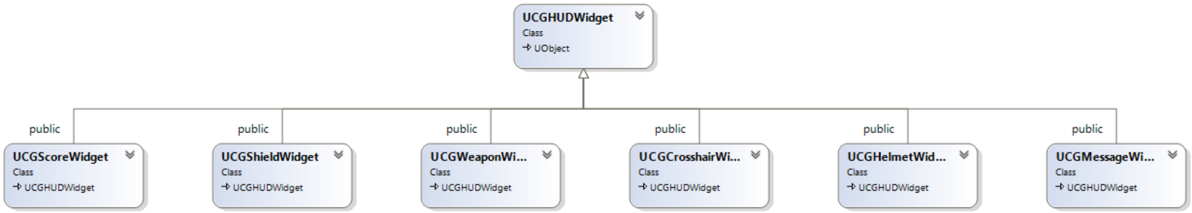


Figure 79 Generated UML for the UI Widget System

Despite *Unreal Engine 4* having a relatively robust user interface system in the form of UMG (*Unreal Motion Graphics*), it was found to be insufficient for in game UI due to poor documentation and lackluster C++ hooks. A widget system was devised to fulfill this shortcoming of Unreal.

6.7.1 UCGHUDWidget

Crystalline's widget system has a root object type in the form of the *UCGHUDWidget.* This object defines the core behavior for the system and how it interacts with the built in Unreal HUD class.

Each widget has three extensible core functions: **PreDraw**, **Draw** and **PostDraw**. **PreDraw** determines whether or not the widget should draw this cycle, caches references to the owning Pawn, HUD, and Canvas, and carries out any other additional preparation for drawing the widget. **Draw** performs the actual canvas calls for the widget, this function is exclusively implemented by the child widgets. **PostDraw** destroys the cached object references, as they are now dated.

UCGHUDWidget also has several utility functions to standardize certain drawing functions. **DrawBrush** handles texture drawing, providing rotation, resolution scaling, and alignment settings. **DrawScaledText** handles text operations to ensure that the text is rendered consistently.

A collection of *UCGHUDWidgets* is maintained with the **PreDraw**, **Draw** and **PostDraw** calls being called on each draw call for the HUD. Z-ordering is implicitly implied by position in array with higher indices having a higher z-index.

6.7.2 UCGMessageWidget

UCGMessageWidget is a configurable widget that defines a message reporting element on the HUD. The widget manages a collection of messages containing text data, the current life of the message, and whether or not it should render. This collection is implemented as a Queue popping off the oldest message when a new message is added and the queue is full, or it reaches the end of its lifespan. The age of a message is incremented in the **PreDraw** call for the widget, popping any messages that "age out".

UCGMessageWidget's are managed by a map structure with the key being a reference to the type of message the widget represents (e.g. Dialog, Action, State, etc.). This key is specified in the instance of the widget as an enumerated type via the UE4 editor. This map is then used to report things such as kills, game status and more.

7. Asset Overview

7.1 Visual Design

Our goal for *Crystalline* was always to have a vibrant visual experience. We settled early on having a futuristic setting and as our narrative about mining crystals evolved, we began referring to our target visual aesthetic as 'Space Arizona.' We want players to feel like they are on an alien world fighting for scarce resources. Because of our team member's passions for 3D art and graphics, and because Unreal Engine 4 offers many tools to facilitate such assets, we chose to pursue a realistic, fully 3D art style.

7.1.1 Character Design

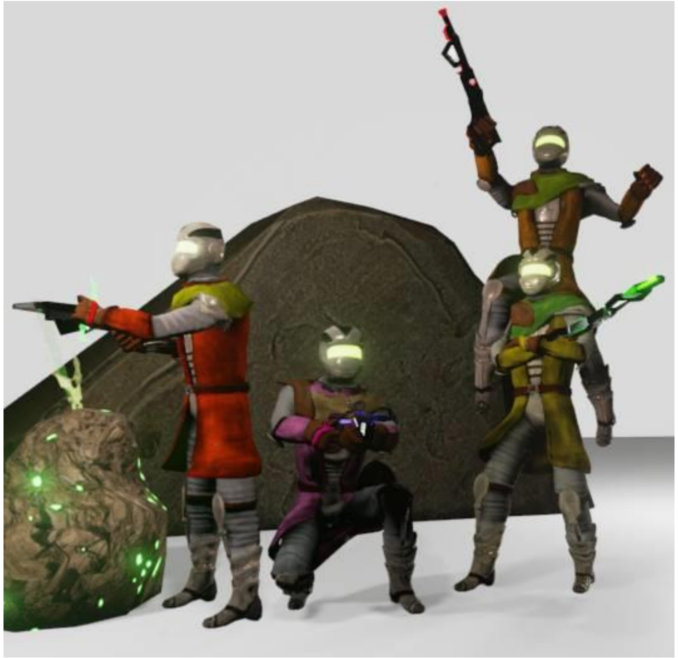


Figure 80 Crystalline's Character

As a multiplayer first person shooter, our player character is a key element of the game experience. We designed the character to support the narrative and aesthetic, drawing on the steampunk art style with a mix of futuristic and western elements. Because the character would represent multiple human players in one game, the character was designed to be neutral so as not to impose specific traits to players, as opposed to having an aggressive or technical look. The intention was that if one player is playing aggressively and another is playing more defensively or even just exploring the map, that personality would be defined for other players by their actions rather than by the appearance of the character. Having different characters or styles to choose from would have been a nicer alternative, but was not attempted due to time constraints.

For a typical modern game, a main character can have upwards of 30,000 polygons (Stirling). As a multiplayer game, up to 7 of these models could be on screen at any one time, so we aimed a little lower. Our character model has approximately 15,000 polygons at full detail and a 5,000 level of detail model which is usually rendered instead. The character was modeled in Maya and sculpted in Mudbox, where it was also painted to create diffuse, specular, normal, and ambient occlusion maps as well as a glow map to make the visor glow slightly. Because Unreal uses a physically based rendering system, these had to be tweaked slightly and the specular map came to represent the metallic and roughness values. Combined, these textures were able to create a detailed character out of a relatively low-poly in game mesh.

To animate the character it was first fully rigged and skinned in Maya. Since the local player would only see the arms of the character, these were extracted from the fully rigged model and separated into a different file. Animations were then generated usually at 30 to 60 frames each. To get these into Unreal, first the skeleton alone is exported, then the mesh and its skinning data is exported separately. This facilitates using the same skeleton on multiple meshes. Then the skeleton root is exported with a baked animation range applied for each separate animation. Each of these files can then be imported into Unreal to create animation sequence assets which can be applied to the character in game.

Originally we had planned to have teams and the character was designed with an orange color scheme and a blue color scheme. As the game evolved and free-for-all deathmatch was the first mode to be completed, the character material was redesigned to support any player color. This was achieved by making the base color a very light grey close to white. A texture mask was made for the areas of the character which needed full color, and a separate mask was made for the character's scarf. The material made in Unreal uses both masks and a dynamic parameter representing the chosen color to lightly tint the entire character towards that color, paint the primary areas fully with that color, and then apply a hue shift to change the color of the scarf to complement the primary color slightly.

To provide visual feedback when a player hits an opponent, the character material was expanded again to support a shield effect. This uses a fresnel term to color around the character and is additively combined with the color of the character. The dynamic color value for the shield is normally black, so no color is added and the shield is invisible. When a hit is received, the color value is changed in code to an interpolated value between blue (full shield) and red (empty shield) and it becomes visible. The value then interpolates back to black over time.

7.1.2 Weapon Design

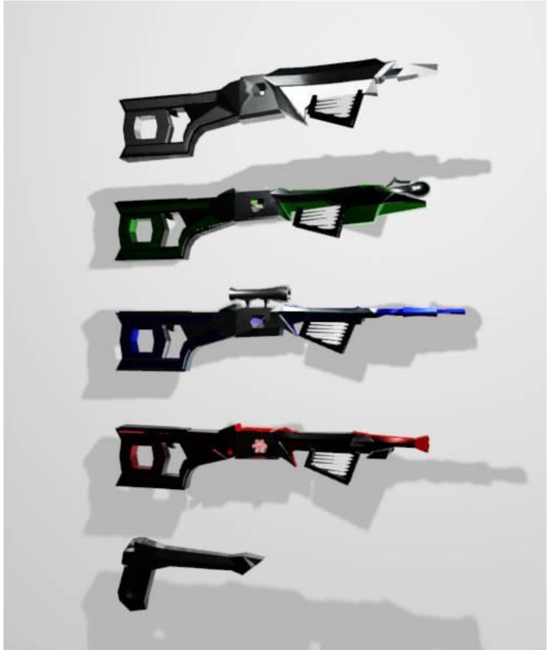


Figure 81 Crystalline's Weapons

The primary weapon of *Crystalline* is the assault rifle, and it is designed to be the standard troop weapon with a neutral grey color. The appearance of the enhanced crystal weapons use the assault rifle as a basis since it is what turns into crystal guns. They were designed with a crystalline aesthetic in mind, with many hard edges and sharp points. Each is colored like the crystal it is comprised of and it resembles recognizable favorites like the shotgun and the sniper rifle. The crystal itself appears in the mesh of each gun to show the player how they use the crystals to create the guns. Each weapon also has a crystal bullet magazine like the dropped ammo available in the game to reinforce for players what the ammo drops do.

The secondary weapon of *Crystalline* is the pistol, which is designed to be much simpler than the other weapons. It mimics the appearance of a standard handgun while maintaining the hard-edged aesthetic of the other weapons, as if made out of similar processes and materials.

Since they are so often far away from the player, each weapon has a high LoD model and a low LoD model. At full detail the most complex gun, the beam gun, has about 2,800 polygons, and about 800 at low detail.

The crystals themselves are each made with a silhouette meant to mimic the ability of the guns they create. The marksman rifle crystal is long and pointed, the shotgun crystal spreads out and is shorter, and the beam crystal has a solid green emerald-like crystal surrounded by twirling yellow crystals to mimic the electrical appearance of the beam gun once it has locked onto a target. To create the appearance of crystals all these meshes use hard shading along the edges, transparency, and a fresnel term to create refraction as you look through them. Then to give them the impression that these are powerful crystals full of energy, an extra texture pattern is added to the emissivity of the crystal's color with UV's that animate over time, making it appear that energy is flowing up through the crystals.

7.1.3 Environment Design

As players traverse the final frontier of space, they fight on worlds similar to that of dusty western landscapes searching for prospects and hoping to hit the motherload of crystals. The worlds are inspired by real world landscapes similar to Southern California, New Mexico, and Arizona. Real world strip mines also influence how these battle arenas should look. Everything in this world should feel dirty and used, as if people live here and have been living here for a long time. We want to connect players to the themes of struggle and prospect of the United States old western frontier to the themes of hope and technology of futuristic space travel. As the design evolved we began referring to this aesthetic as "Space Arizona."

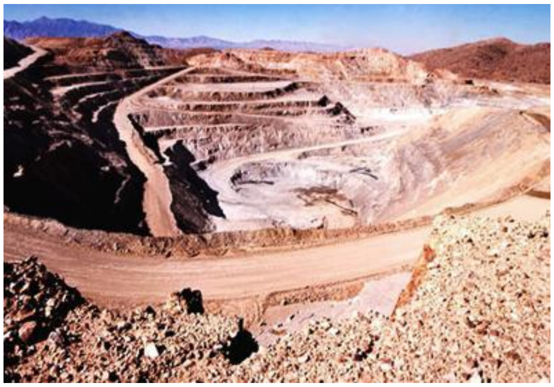


Figure 82 Example of an Arizona strip mine (Nature's Crusaders 2009)

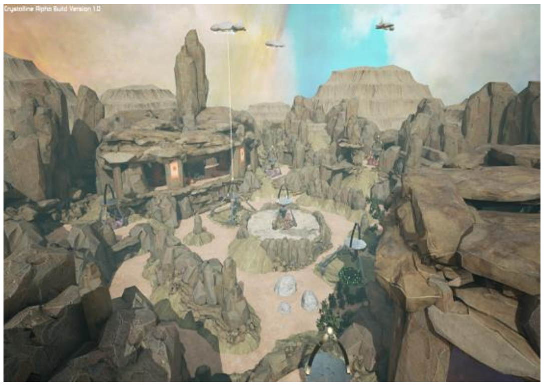


Figure 83 A screenshot from the game map, "Crystal Canyon"

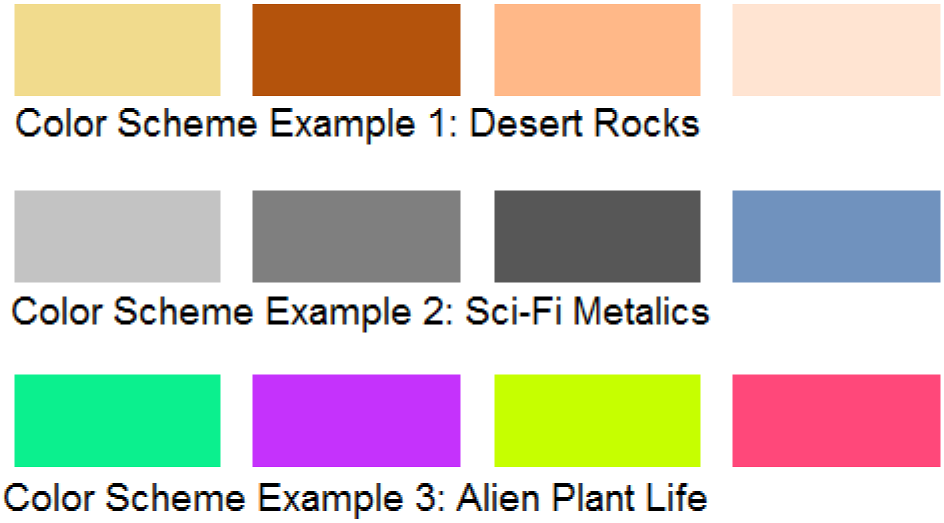

Color Scheme Example 1: Desert Rocks
Color Scheme Example 2: Sci-Fi Metalics
Color Scheme Example 3: Alien Plant Life

Figure 84 Color Palette

The colors and mood of the aesthetic suggest dirty and dusty browns, contrasting shiny and durable metallic greys, and every now and then touches of vibrant alien wildlife through unique purples, yellows, and greens, to suggest an otherworldly presence.

The terrain and structure of the environment are mostly comprised of desert rocks acquired from the Arid Desert content pack available on the Unreal Store and other models made ourselves. The desert pack rocks are the most complicated meshes in our game in terms of polygon counts and unfortunately do not include the editable model files for adding LoD meshes. However they utilize hardware tessellation in the materials applied to them to minimize their impact on performance.

The other structures and geometry of the world of Crystalline are mostly comprised of rusty metal drills, crystal rocks being drilled, energy collectors on top of the drills getting energy from the crystals in the rock, and spaceships which fly by to pick up the energy. All of the metallic materials use cubemap reflection to approximate the environment and sky, and screenspace reflections to accurately mirror specific objects. Many of the other materials take advantage of emissive color and Unreal’s bloom effect. For example, the crystal portions of the drilled rocks use a texture mask and a high color value (> 1) to emit a glow around the crystals. In dark caves, this produces an effect of being surrounded by energy and color.

7.1.4 UI Design

7.1.4.1 HUD

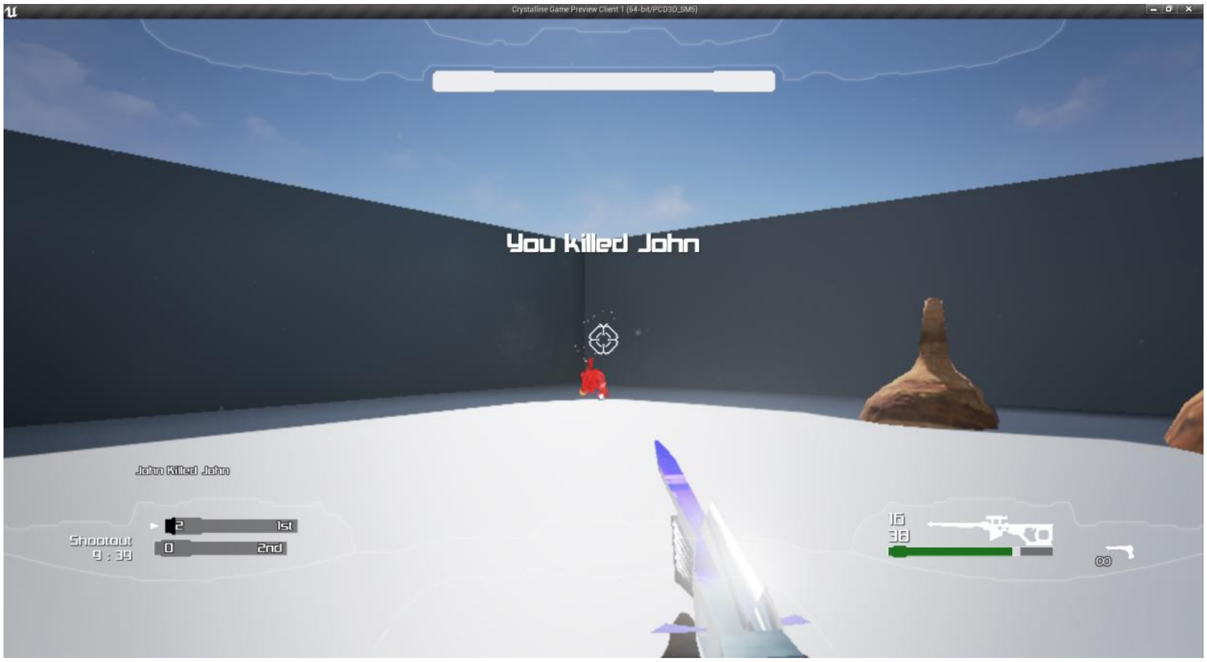


Figure 85 Final HUD iteration

Crystalline's HUD went through a number of iterations (detailed in **Section 5.16**) with the final iteration reproduced above. As mentioned in the prior section the final design paradigm enforced by *Crystalline's* HUD may be categorized as a Meta representation of the HUD elements. Meta indicates that despite existing on a 2D plane the interface elements have an in world analog. The *Crystalline* HUD represents the interior of the player character's helmet, with the lower and upper halves featuring textures that represent the edges of the visor.

The individual HUD elements (Shield Bar, Weapon Info and Game Info) represent hologram projections separate from the visor, in future iterations of the HUD they would move at a rate different than the visor to better indicate a spatial difference. Chamfered edges were used to convey a quasi-futuristic sense while retaining a degree of minimalism. The HUD ultimately focused on emulating a futuristic display, in contrast to the arid desert scenery of the environment. For more information regarding the HUD please refer to **Section 5.16 Graphical User Interface**.

7.1.4.2 Menus

Generally, our Menu Style is both constrained and influenced by the default menu system available with Widgets in the Unreal Engine. We wanted to have a feel that is reminiscent of a realistic AAA game. As a result of this, though there aren’t any animations in our menu system, our menu system needed to be feature-rich, such as a character preview for each color and the ability to add and remove bots in the lobby system. On that note, the majority of work in the menu system was spent on the lobby screen. We wanted to make something that was reminiscent of the lobby system in Age of Empires 2. You can see its influence in the placement of player name and game option placement, as well as the buttons for readying up and canceling being similar to the way it was done in Age of Empires 2 as well. Our choice of using dropdown menus for choosing the game length and point amount to win instead of Unreal’s number slider is also influenced by our choice to imitate Age of Empire 2’s lobby system.

As far as our menu “flow” goes, we tried to design our game so that the amount of button presses from starting the game to being in a match is as small as possible. By clicking play, you are immediately asked if you are joining or hosting a game. Hosting a game will take you to your lobby, and joining will ask you for the IP address for the person who is hosting the game. When everyone is in the lobby and readied up, the game starts when the host starts the game. By polishing the flow of the menu system, we are able to create a more streamlined, compact experience for the player. For more information regarding the Menu system please refer to **Section 5.20 Menus**.

7.2 Audio Design

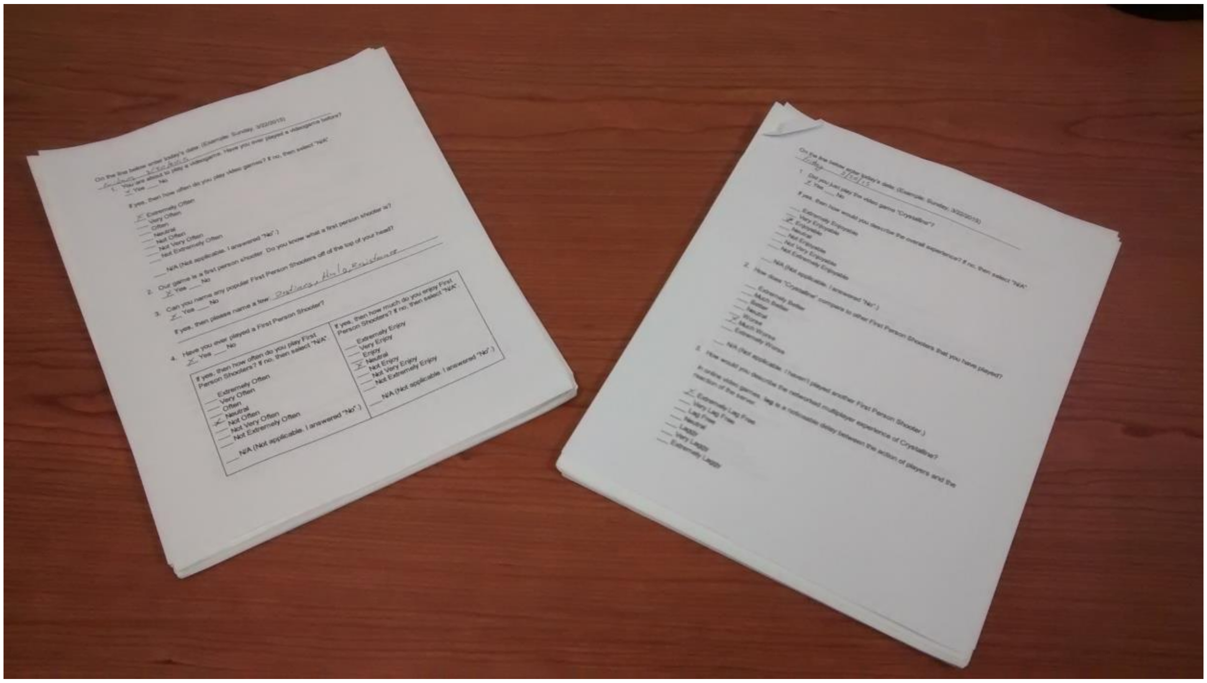
The sounds you hear in *Crystalline* are designed to be as realistic as possible, to keep in tune with our realistic art style. With the exception of our footstep noises, the audio samples used in our game are either licensed under a Creative Commons license, or modified from an Unreal Engine example project. Since our main map is on a desert-oriented planet, the ambient noise found in the game takes the form of a constant, yet quiet wind heard from all over the map. This ambient noise is designed to further immerse the player in our universe, and give players a better idea of the relative barrenness of the planet's environment. Most of the more active noises heard in our game are our weapon firing noises. With realism in mind, we tried to have the sound of the gun firing match the style of crystal that is currently slotted. For example, the red crystal firing noise is loud and explosive. In contrast, the blue crystal's shot sounds a little higher-pitched, giving the idea that the bullet is travelling far faster and therefore with more velocity than the projectiles fired from a gun with the red crystal attached. Our green crystal firing noise is a modified generic "electricity" noise, to match the beam style of the weapon projectile itself.

While walking around the Crystal Canyon map, you are also able to hear the drills found throughout the map, as long as you are near enough to them. This sound was made by combining a modified hum from a refrigerator, with the sound of falling rocks. Another sound of note is the sound of footsteps when you move around, which was made in-house.

8. Playtesting and Results

8.1 Procedure

In late March, the team formalized playtesting by developing two playtesting surveys. One for before users played *Crystalline* and one for after the users completed the playtest event. The final version of all the playtesting surveys can be found in Appendix X.



On the line below enter today's date: (Example: Sunday, 3/22/2015)
[illegible]

1. You are about to play a videogame. Have you ever played a videogame before?
X Yes ___ No

If yes, then how often do you play video games? If no, then select "N/A"
X Extremely Often
___ Very Often
___ Often
___ Neutral
___ Not Often
___ Not Very Often
___ Not Extremely Often
___ N/A (Not applicable. I answered "No".)

2. Our game is a first person shooter. Do you know what a first person shooter is?
X Yes ___ No

3. Can you name any popular First Person Shooters off of the top of your head?
X Yes ___ No
If yes, then please name a few: [illegible]

4. Have you ever played a First Person Shooter?
X Yes ___ No

If yes, then how often do you play First Person Shooters? If no, then select "N/A" ___ Extremely Often ___ Very Often ___ Often ___ Neutral X Not Often ___ Not Very Often ___ Not Extremely Often ___ N/A (Not applicable. I answered "No".)	If yes, then how much do you enjoy First Person Shooters? If no, then select "N/A" ___ Extremely Enjoy ___ Very Enjoy ___ Enjoy X Neutral ___ Not Enjoy ___ Not Very Enjoy ___ Not Extremely Enjoy ___ N/A (Not applicable. I answered "No".)

On the line below enter today's date: (Example: Sunday, 3/22/2015)
[illegible]

1. Did you just play the video game "Crystalline"?
X Yes ___ No

If yes, then how would you describe the overall experience? If no, then select "N/A"
___ Extremely Enjoyable
X Very Enjoyable
___ Enjoyable
___ Neutral
___ Not Enjoyable
___ Not Very Enjoyable
___ Not Extremely Enjoyable
___ N/A (Not applicable. I answered "No".)

2. How does "Crystalline" compare to other First Person Shooters that you have played?
___ Extremely Better
___ Much Better
___ Better
___ Neutral
X Worse
___ Much Worse
___ Extremely Worse
___ N/A (Not applicable. I haven't played another First Person Shooter.)

3. How would you describe the networked multiplayer experience of Crystalline?
In online video games, lag is a noticeable delay between the action of players and the reaction of the server.
X Extremely Lag Free
___ Very Lag Free
___ Lag Free
___ Neutral
___ Laggy
___ Very Laggy
___ Extremely Laggy

Figure 86 First iteration of pre-playtest and post-playtest surveys

Each playtesting survey has a specific goal. The pre-playtest survey's goal is to tell us about our players. We don't have the resources to organise specific focus group testing, so with the pre-playtesting survey we ask questions such as, "How often do you play video games?" and "Are you familiar with the First Person Shooter genre?" We gathered this type of information to create graphs about the data we collect from the post-playtest survey. The data from the pre-playtest survey might help us put the post-playtest survey data into context. Providing info such as, "Hey, I noticed that we have a lot of playtesting data from players who play video games, but who don't play many First Person Shooters. We might need to find some more playtesters who fill that focus group." The pre-playtesting survey is designed to be short. It only has four questions and fits on one page. Questions included are either “Yes or No” questions or Likert Scales.

Unlike the pre-playtest survey, the post-playtest survey is designed to give us information and feedback about our game. The post-playtest survey is a bit longer than the pre-playtest survey and totals in at ten questions. This is because we wanted to get more information, but still not force our playtesters to fill out an annoying amount of questions. In the post-playtest survey, we ask questions such as, “How would you describe the overall experience of *Crystalline*?” and “How does *Crystalline* compare to your expectations of a First Person Shooter?” Each question in the post-playtest survey is based on the likert scale, except for the last questions which is open ended, leaving room for playtesters to express their opinions freely.

Later on in our development, including for Imagine RIT, we moved our playtesting survey to a Google Form, so that it could be filled out and submitted more efficiently, just by the player moving to that tab, and filling things out that way.

8.2 Data

Our final iteration of our playtesting survey is hosted on Google Forms, and consists of 17 multiple-choice mandatory questions, and 8 short-answer optional questions that were usually focused on giving players the opportunity to add any extra comments. This was the format used for our Imagine RIT playtests. 70 playtest results were received in total over the course of *Crystalline* development time, with 59 of these being from the Imagine RIT playtests. The following data results were compiled from all 70 playtest results.

The first notable result from our playtest was that the majority of people who played *Crystalline* were at least somewhat acquainted with First-Person Shooters.

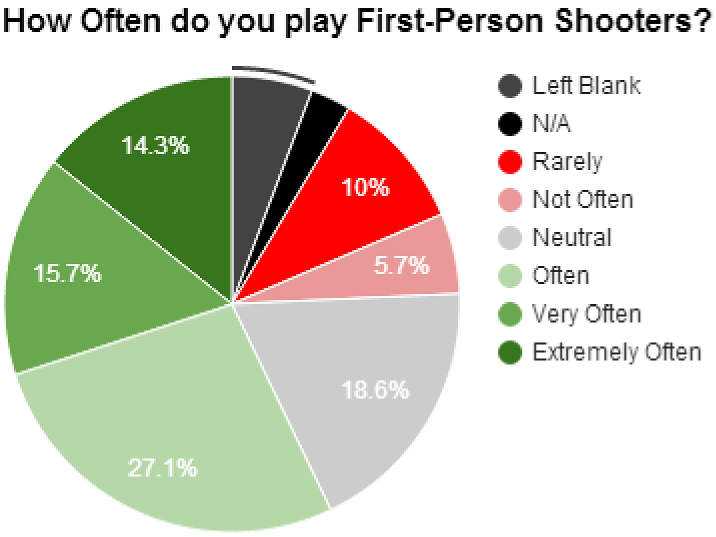


Figure 87 How often playtesters play FPS games

Of our 70 playtests, roughly 15.7% of our players answered “Rarely or Not Often” to the question “How Often do you play First-Person Shooters?” This is contrasted with the amount of results claiming to play First-Person Shooters “Often, Very Often, or Extremely Often, with 57.1% of our players choosing this result.

Another noteworthy result is how much our playtesters enjoyed our game.

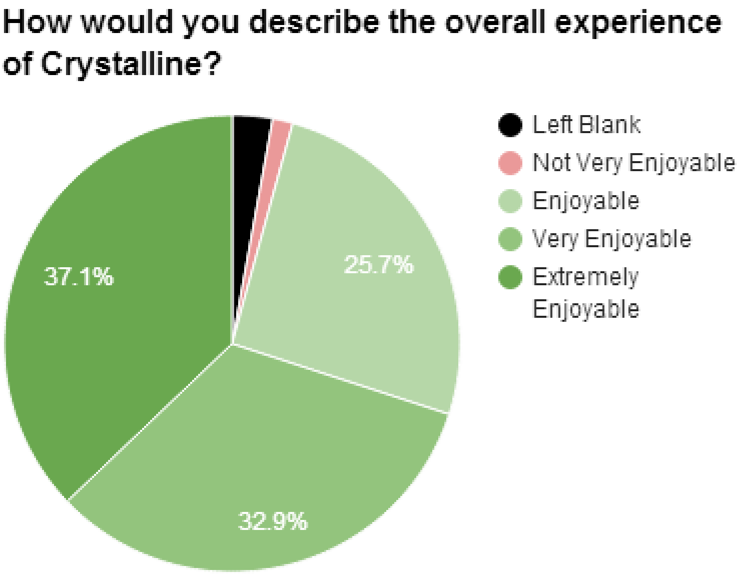


Figure 88 How playtesters felt about Crystalline

According to our data, only one player rated our game as "Not Very Enjoyable". The rest of our playtesters rated our game positively, or left the question blank. In addition, the answers to the question "How Does *Crystalline* compare with your expectations of a First-Person Shooter?" were positively notable in this regard too.

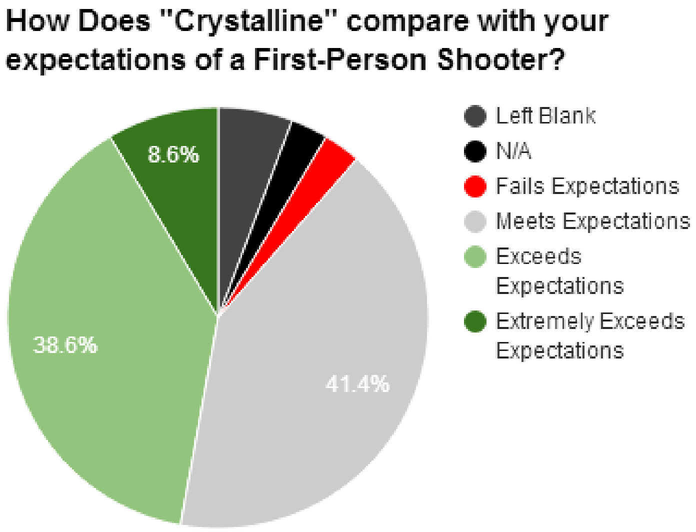


Figure 89 How Crystalline compares to other FPS games to playtesters

A total of 47.2% indicated that *Crystalline* exceeded expectations of a First-Person Shooter, with 41.4% of results saying that *Crystalline* met expectations. From this point in the playtesting survey, the questions move on to the opinions of the player of specific parts of *Crystalline*. One example is the player's opinion of the Weapon Sandbox.

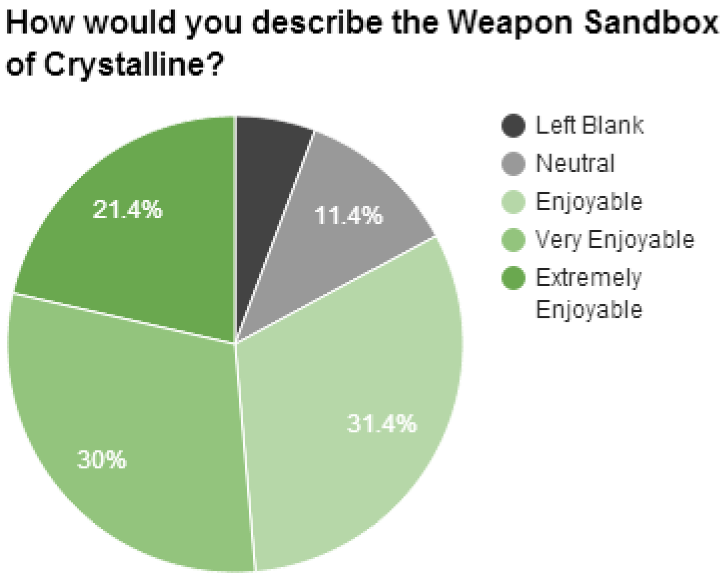


Figure 90 How playtesters felt about the Weapon Sandbox of Crystalline

As shown here from the results of this question, our weapon sandbox is thought of in a generally positive light. The opinions on the tutorial were slightly more mixed.

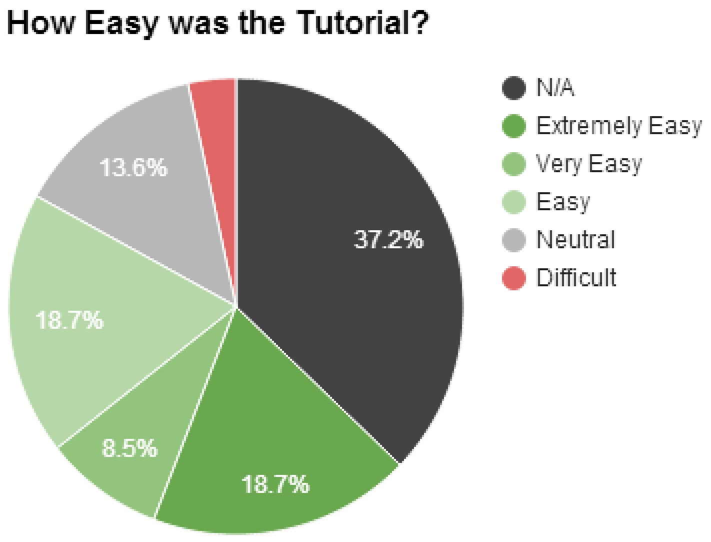


Figure 91 How playtesters felt about the tutorial of Crystalline

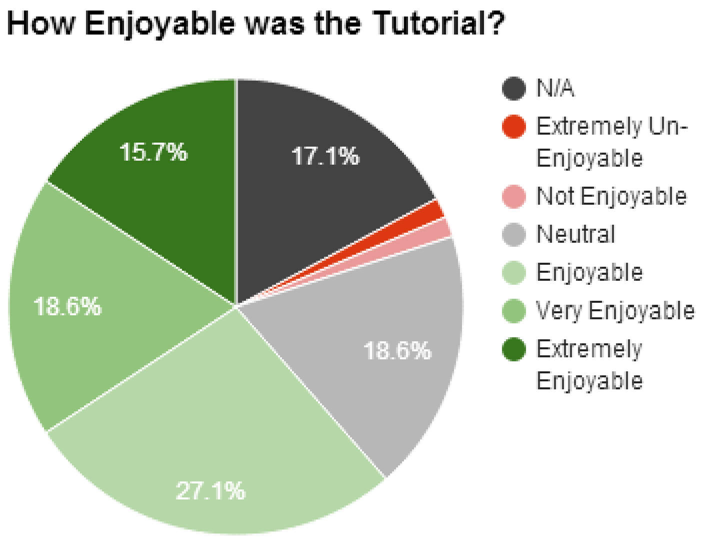


Figure 92 How playtesters felt about the tutorial's difficulty

Due to the wording of this question, many players did not entirely fill out the question, and many players did not give an opinion of both how enjoyable and how easy the tutorial was.

As far as the visual experience of *Crystalline* goes, the results were very positive.

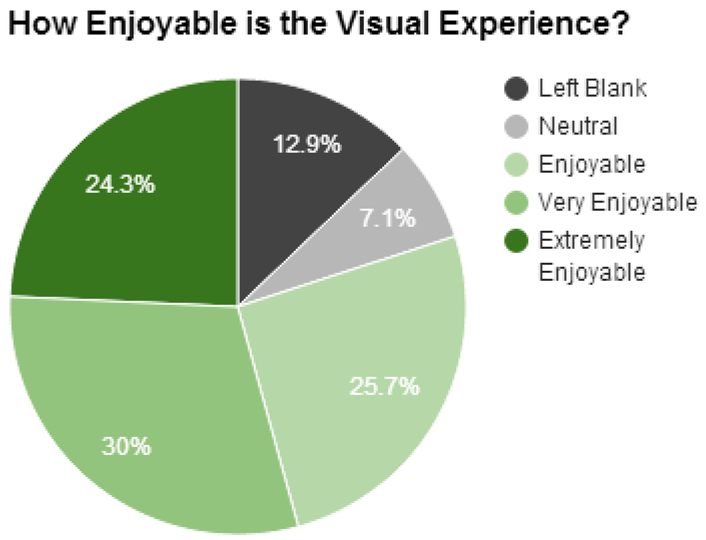


Figure 93 How playtesters felt about the visuals of Crystalline

As shown here, 80% of all players gave positive feedback on this visual experience of our game. We received similar results for how comprehensible out HUD was.

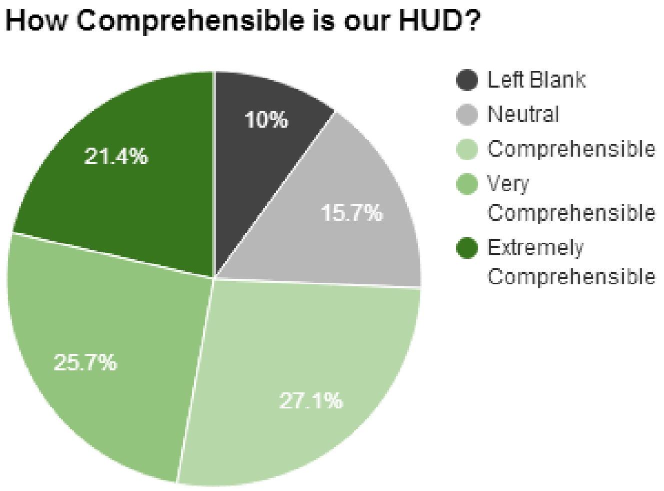


Figure 94 How playtesters felt about the HUD of Crystalline

The response towards our HUD was largely positive too. 74.2% rated our HUD as being comprehensible.

In addition to the playtesting data that was received by our playtesting surveys, a copy of our game was sent to a team of designers at Raven Software. Since the copy of *Crystalline* we sent to them was fairly early in development, much of the feedback we received was already on our to-do list of things to fix. However, there was a lot of strong feedback unrelated to specific features. For example, one piece of advice was to narrow our scope so that we take the core gameplay experience to 100% polish. Another good example of strong feedback we received from Raven is regarding our map. The designers thought that our map was too open, and had too many dead ends. As a result, our map went through some tweaking in accordance with their advice. We also received feedback related to the bugginess of our spawn points and only having to do the tutorial room once per game, which was fixed before we even received this feedback from Raven specifically.

8.3 Conclusions

The most valuable conclusions drawn from this data often relates to the optional comments given by some playtesters in regards to specific things we can improve about our game. For example, a lot of our feedback regarding our weapons sandbox was related to which crystals the player believed were the most powerful, and which crystals the player believed needed to be made more powerful. Usually, the blue crystal and the red crystal seemed to be the most powerful, with the green crystal lagging behind in apparent power. We took this feedback into account while tweaking our weapons sandbox. Other players wanted to be able to combine crystals, which is valuable feedback to have in regards to what we might have added given more time.

Most of our playtests were run as a combination of a networking stress test, debugging session, and design test in mind. Since *Crystalline* is a networked multiplayer game, it's important to make sure everything stays in sync with players. One area that we were definitely looking for feedback with is the mini-tutorial at the beginning of the map, which gives the players a quick run-through of the crystal-slotting mechanic, how to shoot, and how to jump. Though we show the controls during the loading screen of the game, we felt it was helpful to give the player some extra pointers regarding how to play our game. The playtests concluded that on the whole, the tutorial room is fairly easy, though some people needed pointers to get through. Throughout development, playtests were a great way to get a feel for whether or not the players needed more guidance or pointers. Overall, the conclusions that we received from our playtests were that people generally enjoyed our game, and a lot of feedback was how to best expand our game.

8.4 Iterations

Here we discuss how we took the data and conclusions from our playtests and implemented the feedback to create a better experience for our players.

8.4.1 Map Iterations

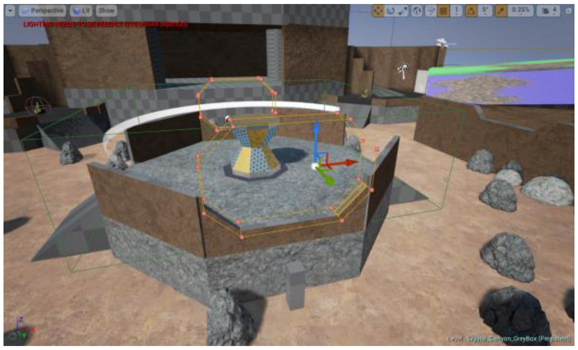


Figure 95 Subtractive Geometry in Editor

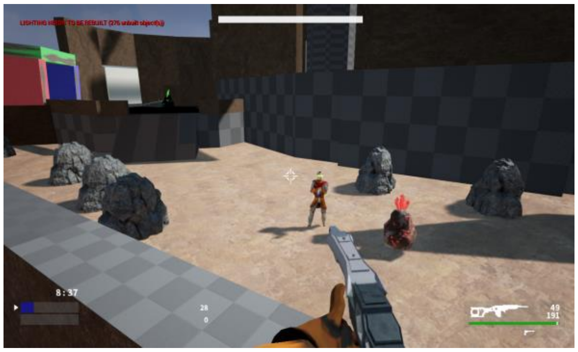


Figure 96 A screenshot from in the hill

We had a playtesting event in which we noticed that there were some balance issues with the King of the Hill game mode on the Crystal Canyon map. At the time, the only hill zone in the map was located directly in the center on the platform with the gold rotating drill. The hill had a bit too much cover for players who were already inside. To try to fix this, Nick placed some subtractive geometry on some of the walls around the hill to open the hill up a bit more.

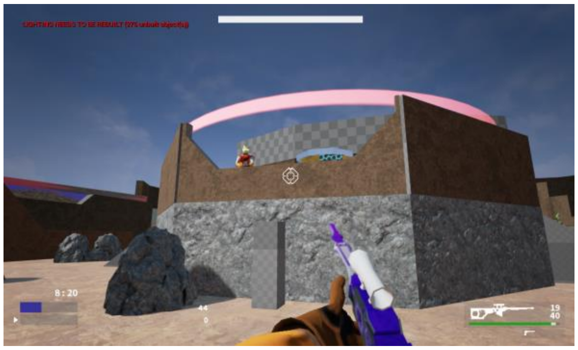


Figure 97 : A screenshot from a player outside and under the hill

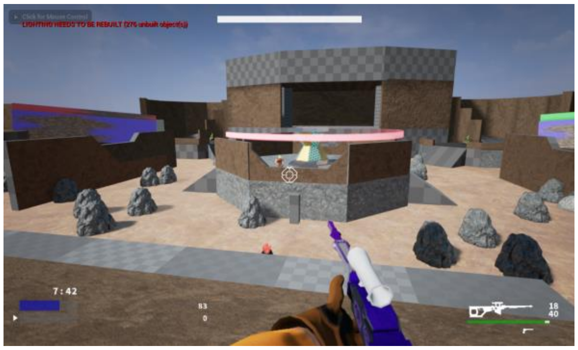


Figure 98 A screenshot from a player outside and far from the hill

Originally this map was designed for another game mode in mind, but we cut that game mode from the game. And now that our game only supports Deathmatch and King of the Hill, tweaks to the map need to happen to ensure that the two game modes that we offer are fun to play in our environment. Luckily at the time of this incident, the grey box Maya version of the map wasn’t too far along and those assets for the middle area weren’t made yet. So this design tweak didn’t impact production flow at all.

9. Post Mortem

The making of Crystalline was a difficult endeavor. Some of the choices we made or, more often, didn't make hurt our process and led to features getting cut and tension between teammates. At the same time, some of the tools and techniques we used were able to make up for these issues and allowed us to finish strong. Reviewing and remembering both the strong and weak points of any process are critical to personal and professional growth so that we can improve on our future endeavors.

9.1 What Went Right

9.1.1 Unreal Engine 4

9.1.1.1 UE4 Resources

Unreal Engine 4 is at its core a First Person Shooter Engine. As such it has built in functionality supporting first person players, networked replicated scoring, names and game modes. Upon comparing the functionality of UE4 to competitors it was found that we could get to actually implementing gameplay features faster for our game, allowing us to go way farther than if we had used Unity or an in house engine.

UE4 had a number of tutorials and sample code that could be quickly referenced while implementing features. For example, Unreal has several sample projects demonstrating networked gunplay that gave us a good jumping off point. We were then able to focus more on the crystal system and gun mechanics. Dozens of such sample projects existed to onboard us quickly to the engine. While UE4 has great sample projects and documentation, it should be noted that UE4 is still being developed and some documentation has not yet been written.

9.1.1.2 UE4 Networking

One strong advantage of using UE4 for the development of *Crystalline* is how easy it makes Networking. Using UE4's built-in replication of objects and Remote Procedure Calls, a lot of the work was taken care of in regards to setting up the multiplayer component of *Crystalline.* In addition, using UE4's built-in networking component allowed us to forgo the task of searching for and integrating a 3rd-party library for multiplayer networking, such as Photon Networking, or writing a networking library from scratch. UE4's networking component saved a huge amount of time, allowing us to focus on other aspects of the project.

When utilized correctly, UE4's networking library resulted in a largely lag-free experience. That being said, one small caveat to using UE4's networking was its rather obtuse documentation. Though there were a few tutorials we found on the subject (See Section 3.2.1), much of the "specifics" of how the networking actually functioned were left out. This necessitated our looking into the engine itself, conducting experiments within the engine, or asking through the Unreal Forums.

9.1.2 Artistic Workflow

Choosing Unreal was probably the best choice we could have made when it comes to creating the visually rich world we wanted to make. One of the primary reasons for switching to Unreal was its visually rich effects system, saving us from having to develop one of our own. In general, Unreal's UI and automated processes made it possible for us to create the visual experience were going for. UE4's multiple control schemes are similar to existing 3D tools. This leads to a short learning curve for developers already familiar with such tools. The editor offers many customizable options, profiling tools, everything is color coded and assets generally have automatically created thumbnails so you know what each is just from skimming over them.

Unreal also comes with a full suite of animation tools providing features like animation blending, events called ‘Notifies’ for playing particle effects or sounds inside an animation, and access to play animations from code which allowed us to network them properly. The GUI for animations in Unreal is called Persona. Persona includes a preview window for each animation showing changes to settings in real time. It loads any animation related assets that apply to the same skeleton in the same window, so you can access them quickly. The GUI for particle systems, Cascade, and the Material Editor similarly have previews and tools for changing fine detail in any asset.

Any particular asset could be composed of multiple meshes, materials, textures, animations, and particle systems, and once these are all set up, one can change any individual component without breaking the others. Because Unreal has reference to the source file for all its assets, it can quickly refresh changes made to files if they are re-exported from external tools. All the materials, code interaction, animation, particles or whatever else applies to the asset in question remains stable and the change can be seen in its final in-game state immediately. This made iterating on existing assets much easier than it could have been and allowed us to make many tweaks and improvements over time.

9.1.3 Modular Art Set

Using a modular art set for our environment was definitely something that our team did right. Once the modular set was completed, updating the grey box level to its detail pass only took 3-4 days of constant sprint. If we hadn’t used a modular set, then that time frame would have probably been more like 3-4 weeks.

Our modular set is split up into 2 sections, the Arid Desert pack, purchased from the UE4 marketplace and created by Tris Baybayan, and an assortment of rocks created by our team members to highlight gameplay elements and visual themes that were unique to our game and not general enough to be found on a marketplace.

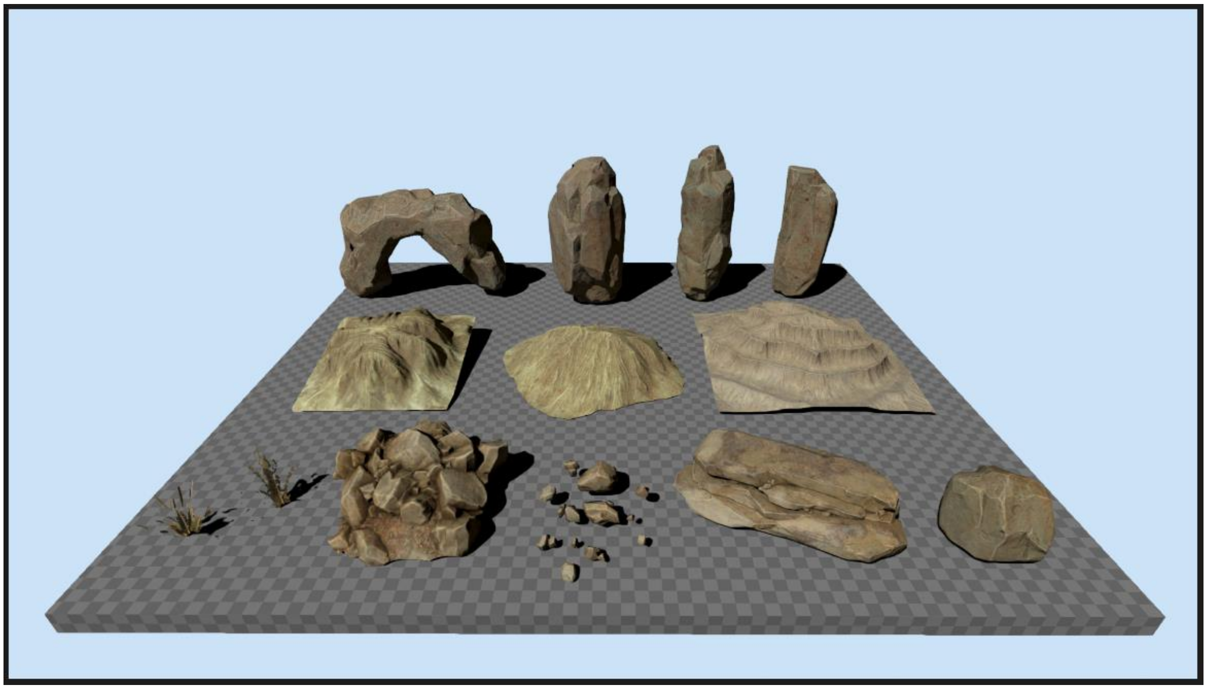


Figure 99 Arid Desert Modular pack by Tris Baybayan (the arid desert pack was used to create the majority of the environment that the player interacts with in Crystalline)

For the Arid Desert pack, the combination of professional quality, variety of the pack, and matching theme was a perfect choice for our game. In all honestly, without the Arid Desert pack, we couldn't have had the turnaround time that we did with the final art pass of the environment, and we would not have had a visually appealing environment for the Imagine RIT and RPI builds. We also probably would not have had it for the final capstone deliverable either. Making this purchase saved us many hours on production, and was a positive net gain for our work efficiency.

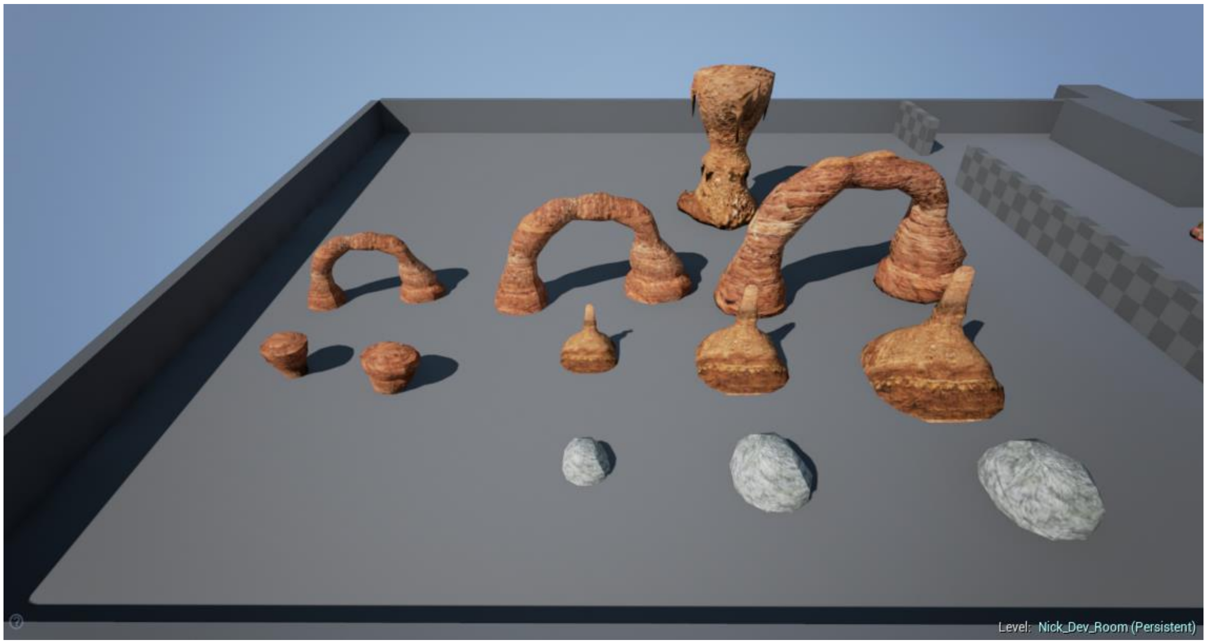


Figure 100 Custom Modular pack by Crystalline Team

Initial investment on creating the modular set paid off large dividends for our team. The practice lends itself to having independent units that can be used together to create more complex structures, a single modular asset can be rotated, scaled, repositioned and even combined with another single modular asset to create multiple and diverse sections of a map. And when you combine one asset with another that can also be rotated, scaled, and repositioned, the possible combinations increase exponentially. In theory, your modular set has a finite number of combinations, but it is up to the imagination of your designer and artist to tap into that number and see these combinations in their mind. The designer should feel free to use a single modular asset in creative ways, even if that is outside of the intended role of that asset, for that is one of the potential combinations the asset offers. Thinking outside the box, the modular methodology can and should be used to prove concepts, break up repetition, and test forms, shapes, and composition. Truly with a modular set, the potential is only limited by the imagination of the designer.

9.2 What Went Wrong

9.2.1 Communication

Though we were communicating very well for the first semester of development, the second semester fell apart rapidly from a communication standpoint. One reason for this is our lack of a regular meeting time for most of the second semester. Originally in our first semester, we met every Monday, Wednesday, and Friday to discuss what we were working on and what we had to do. In our second semester, however, we did not meet at a regular time. Instead we chose to talk to each other about our progress more irregularly. This resulted in a lack of communication within the team. Our team worked very well in the first semester of development, and the fact that our schedules meshed less well in the second semester may have contributed to our lack of regular meetings. One way to remedy this could have been to utilize a group chat program like Google Hangout to facilitate communication. In addition, if we had established a regular meeting time early on and stuck to these meetings, it would have been easier to continue this routine throughout the second semester.

A contributing factor to our lack of communication was not understanding what each individual was working on at any given moment. Some tools we feel would have helped with this problem (and we later used) were the programs Trello and Google Hangout. When we utilized these programs, most of us found them to be helpful in visualizing our progress and keeping in touch with how different team members were doing. In addition, since we did have schedules that had a hard time lining up, using these programs regularly would have helped even more.

One problem we had relating to communication was that some members of our team felt as though they weren’t sure where they were in relation to the “bigger picture” of the game. Sometimes our roles for the game were blurred or team members felt discouraged from a lack of progress. Clearer tasks and better tracking of such tasks, through services such as Trello, would have likely ameliorated these issues.

We had trouble communicating goals in the short term, and we also had trouble expressing our long term goals and priorities with the project. We may have not known what we were really hoping to get out of the experience and it may have changed over time. In any case, these intentions were never stated formally, leading to different priorities between team members. While the reasons were all probably important, without more regular meetings and understanding of each other's feelings, people would go to work on different tasks at different times, creating a tension between us.

Having a healthy work-life balance is a challenge to achieve at best and life interfered with everyone's ability to work well at one point or another. This is a problem common to any group project, but in our case we could have handled it better. More openness and meetings would have helped us to pick up each other's slack when someone was having problems.

9.2.2 Architecture Design

While we did a lot of design, next to none of it was in regards to the actual technical architecture of our game. The majority of our core systems had little to no formal definition, whether that be through UML diagrams or plain old English, with the notable exception of the prototypical weapon system (which was ultimately discarded in the switch to UE4). The core of the problem likely lies in the fact that our process was so iterative, we frequently wouldn't have the whole specification before implementation, making much of the code prototypical.

We attempted to establish a set of formal definitions using Mini Design Documents (MDD), design one sheets for programmers. The core structure was as follows: state the problem, specify resolution parameters, and propose a solution. Once an MDD was completed, it should be read by at least one other set of eyes before the writer does any work, after which the programmer would report how the process worked. Ultimately, the goals of this process are to foster communication and self-document our architecture and code base. However, we failed to implement this process, most likely due to the upfront time investment it required.

In a similar vein, documentation was frequently lacking. Certain code segments were easier to trace than others, primarily linked to the commenting habits of the programmer. As with the MDDs, a formal documentation rule set was never formally defined likely due to the upfront time cost of teaching all the members of the team best practices in such a limited span. This documentation issue was further compounded by Unreal Blueprints.

While Blueprints support documentation, the visual programming can become extremely hard to maintain and parse visually. Simple operations that may take a line of C++ code can spawn a multitude of lines connecting one node to another, crisscrossing every which way. Naturally, this can lead to some serious issues down the line when programming without formal planning. In one example, a quick and dirty solution was roughed out in blueprints that wasn't extensible and had some serious limitations (See Figure 101 Blueprints in action). While not uncommon in traditional programming languages, it was found that Blueprints tended to lend themselves to more slapdash solutions.

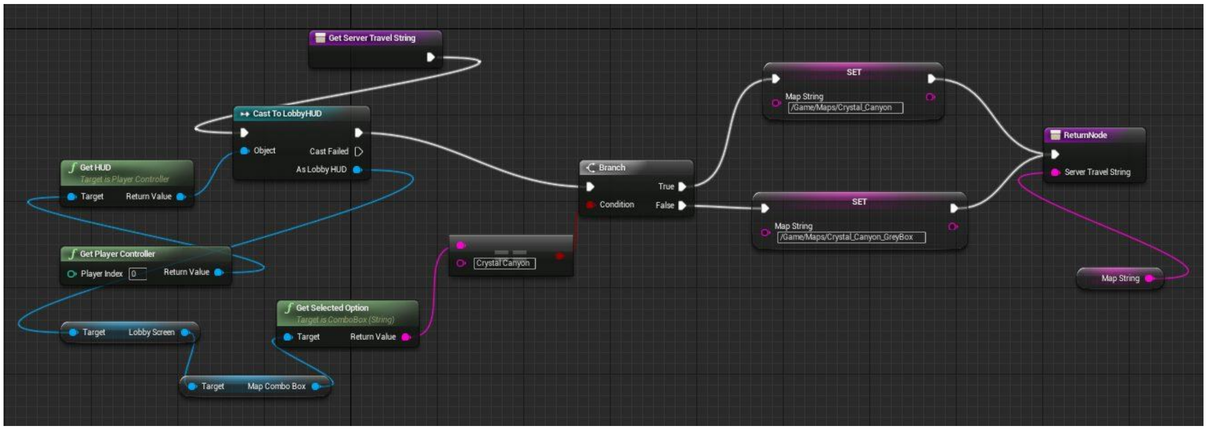


Figure 101 Blueprints in action

Were *Crystalline* to be remade, it is likely that the Architecture Design could be made far stronger through the execution of the MDD process described above. That being said, where *Crystalline* got documentation correct, it was generally well implemented or at the very least easily modified.

9.2.3 Scope

One aspect of *Crystalline's* development that definitely needed to be tweaked was scope. Throughout development, we were plagued with scoping issues. At the game's inception, it was a Real-Time Strategy game and a First-Person Shooter game rolled into one, and was even given the name *Overscoped,* with the intention of creating an entirely new super over the top action wargame genre. While we cut many features during the design phase, the game was as over-scoped as the initial design, if not expanded upon by adding more features. One thing that may have helped is a more stringent greenlighting process. This could have brought up our issues of scope especially in regard to team size, objectives, and project development time.

Crystalline had a fairly over-scoped design initially, but the developers themselves are not without their faults as well. Throughout development, we had the tendency to overestimate our ability to get work done. There are multiple factors which contributed to this, including our lack of accounting for other life factors such as classes and communication issues. In addition, many of us had other things to attend to in our lives, like jobs and job searches, which took a lot of time away from capstone. Our 4 members also tended to have a hard time scheduling times to work, as our schedules did not align very often.

On this note, many of our issues pertaining to scope may have been related to our pride and passion for the game we had crafted and over-scoped, making it a pain to let go of a feature an individual was passionate about or to compromise a feature for the good of the team. Since we were all very passionate about games, and the game we were making in particular, having to cut features was a painful proposition. In summary, don't be afraid to kill your babies.

9.3 Future Work

9.3.1 Crystal Combinations

While the current weapon sandbox is interesting, earlier iterations of our design had a far more robust take on how the Crystal Gun should work (See **Section 4.3.4.1**). In the initial design, the weapon had slots for two modifications, allowing the player to mix and match crystals and create a weapon that better suited their needs. For example, a Power and Precision crystal may create a Rocket Launcher.

Ultimately this feature was cut due the time constraints; even the bare minimum implementation of this system would have doubled the size of the weapon sandbox. With each new weapon added to the sandbox, the weapons would have to be rebalanced, new assets would need to be created, and bugs would have to be squashed. Given more time, these issues would be able to be addressed and the game as a whole would likely benefit, as the core gunplay would be a richer, deeper experience.

9.3.2 Additional Maps

Crystal Canyon

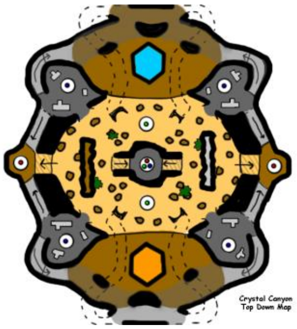


Figure 102 Original Map Concept

This is different from the “Crystal Canyon” that shipped with *Crystalline*. This version of Crystal Canyon is specifically designed for the Contention gamemode in which the drill stations are capturable outposts that teams control to gain points. The geometry of crystal canyon would need to change to help better focus on this gamemode.

ARAM (All Random All Mid) Map

Built with the modular environment set and designed for the conquest game type, this map was intended to be a long straight canyon. The game type of Contention on this map will basically feel like “tug of war”. One idea for this map would be to have a small trickling river through the middle of it. (the river caused the erosion of the canyon). On one end, the river comes from above, creating a waterfall, while on the other end, the river opens up and falls down into a large ravine, this side also has a wide open vista.

Raid Map

This was intended to be an asymmetric map for use in the Raid gamemode. One team would be defending a structure like a landed mothership, while the other team attacks from multiple lanes or access points. Originally we also discussed having the defending team’s base interior be generated procedurally so that new strategies would have to be formed each time and more experienced players would have less advantage over newer players.

Map Editor

Another thing that would be cool is that if users could create and edit maps in game via an in game editor.

High concept milestones:

- 1st: Component toy chest (a set of modular pieces).
- 2nd: also have a terrain plain that can be edited, pulled, pushed and smoothed
- 3rd also have it so that you can pop in / pop out live edit system similar to Far Cry and Halo Forge.

9.3.3 Additional Game Modes

Contention

Contention is the gamemode originally designed for *Overscoped* and *Crystalline*.

Some high concept ideas about this concept were:

- Symmetric.
- Fight on the planet.
- Team Based.
- Map is statically designed.
- Medium to large map.
- Narrative: Two teams fighting on the planet surface over crystal harvesting.

This game mode still needs to be actually designed, but the high concept has been established.

Raid

Raid was the first secondary gamemode in mind for the *Crystalline* project.

Some high concept ideas about this concept were:

- Asymmetric.
- Invade the mothership, attackers and defenders.
- Team based.
- Map is built procedurally.
- Medium sized map.
- Narrative: Attacking party invades defender's ship for their crystal reserves.

This game mode still needs to be actually designed, but the high concept has been established.

Deathmatch - aka Shootout

With Contention being the main gamemode, deathmatch was a secondary gamemode in the original plan and had a different place in *Crystalline's* sandbox. Should Contention and Raid become actual game modes, then deathmatch needs to be reworked thematically to fit into the new narrative. It should also be noted that we changed the name of this gamemode to “Shootout”

Some high concept ideas about this concept were:

- Arena map.
- Free for all mode first and foremost - but supports teams.
- No capture nodes.
- No bases or outposts.
- Map is statically designed/ Small sized map.
- Focus on deathmatch, pvp, and weapon pick-ups.
- Narrative: Holodeck simulation room to prep soldiers for battle.

This game mode would need to be further designed to fit into the new Crystalline aesthetic, but the high concept has been established and many design choices should still apply from the shootout gamemode.

Bibliography

Activision. 2015. "Call of Duty."

Aldridge, David. 2011. "I Shot You First: Networking the Gameplay of HALO: REACH." *GDC Vault.* Bungie LLC. Accessed October 13, 2014. http://www.gdcvault.com/play/1014345/I-Shot-You-First-Networking.

Andersson, Johan. n.d. "Terrain Rendering in Frostbite Using Procedural Shader Splatting." DICE. http://dice.se/wp-content/uploads/Chapter5-Andersson-Terrain_Rendering_in_Frostbite.pdf.

Bacik, Michal. 2015. "Rendering the Great Outdoors: Fast Occlusion Culling for Outdoor Environments." *Gamasutra.* Accessed October 14, 2015. http://www.gamasutra.com/view/feature/131388/rendering_the_great_outdoors_fast_.php?print=1.

Baybayan, Tris. n.d. "Arid Desert Pack."

Bekir_VirtualDJ. 2011. "Electric." 2011, October 31. Accessed May 19, 2015. https://www.freesound.org/people/Bekir_VirtualDJ/sounds/132834/.

BF4Central. 2015. "Battlefield 4 AK-12." Accessed May 19, 2015. http://bf4central.com/wp-content/uploads/2013/07/battlefield-4-ak-12-screenshot-1.jpg.

blaukreuz. 2012. "avl07.wav." August 8. Accessed May 19, 2015. https://www.freesound.org/people/blaukreuz/sounds/162672.

Boe, Black. 2006. "wind.ogg." August 26. Accessed May 19, 2015. https://www.freesound.org/people/Black%20Boe/sounds/22331/.

Bramer, Billy. 2014. "Blueprint Networking Tutorials." Epic Games, April 16. https://www.unrealengine.com/blog/blueprint-networking-tutorials.

Brioux, Bill. 2004. "CANOE - JAM! Firefly: Firefly Series Ready for Liftoff." November 11. Accessed May 19, 2015. http://jam.canoe.ca/Television/TV_Shows/F/Firefly/2002/07/22/734323.html.

Bui, Tom. 2014. "Embracing User Generated Content." *Steam Dev Days.* https://www.youtube.com/watch?v=SRyUpR4qOxU&list=UUStZs-X5W6V3TFJLnwkzN5w.

Burgess, Joel. 2013. "Modular Level Design for Skyrim." *GDC.* http://www.slideshare.net/JoelBurgess/gdc2013-kit-buildingfinal.

Butcher, Chris. 2008. "E Pluribus Unum: Matchmaking in HALO 3." *GDC.* .http://halo.bungie.net/images/Inside/publications/presentations/gdc2008_butcher_chris_matchmaking.ppt.

Byrne, Ed. 2005. "Game Level Design." Boston, Massachusetts.

Cantlay, Iain. 2011. "DirectX 11 Terrain Tessellation." Nvidia, January. https://developer.nvidia.com/sites/default/files/akamai/gamedev/files/sdk/11/TerrainTessellation_WhitePaper.pdf.

Cervin, Albert. 2012. "Adaptive Hardware-Accelerated Terrain Tessellation." Linköping University, November 14. Accessed May 19, 2015. http://dice.se/wp-content/uploads/adaptive_terrain_tessellation.pdf.

Chernyak, Ulyana. 2014. "The MOBA Market Expansion." *Gamasutra.* http://gamasutra.com/blogs/UlyanaChernyak/20140722/221552/The_MOBA_Market_Expansion.php.

Claypool, Mark. n.d. "Networking for Games." Worcester Polytechnic Institute. http://web.cs.wpi.edu/~imgd4000/d07/slides/Networking.pd.

DeLeon, Vi. n.d. "Halo 3 Flood Alien Level Autopsy." http://www.bungie.net//images/Inside/publications/presentations/!Halo3_Flood_Alien_Level_Autopsy.zip.

DICE. 2012. "Alborz_01.jpg." August 8. Accessed May 19, 2015. http://blogs.battlefield.com/2012/08/inside-dice-alborz/.

—. 2013. "Battlefield 4." Stockholm: Electronic Arts, October 29.

Engel, Wolfgang. 2014. "GPU Pro5 ." Boca Raton: CRC Press.

Entertainment Software Association. n.d. "Essential Facts about the Computer and Video Game Industry." The Entertainment Software Association. https://www.theesa.com/facts/pdfs/ESA_EF_2013.pdf.

Epic Games. 2015. "ShooterGame." February 4.

Fieldler, Glenn. n.d. "Networking for Game Programmers." Accessed October 14, 2014. http://gafferongames.com/networking-for-game-programmers/.

Galuzin, Alex. 2013. "CS:GO How to Design Gameplay Map Layouts (Complete In-Depth Guide)." December 10. http://worldofleveldesign.com/categories/csgo-tutorials/csgo-how-to-design-gameplay-map-layouts.php.

Gearbox Software. n.d. "Battleborn." Plano.

—. 2012. "Borderlands 2." Plano, September 18.

Gibson, Sarah. 1991. "Constrained Elastic SurfaceNets: Generating Smooth Models from Binary Segmented Data." MITSUBISHI ELECTRIC RESEARCH LABORATORIES, December. http://www.merl.com/publications/docs/TR99-24.pdf.

Griesemer, Jamie. 2002. "The Illusion of Intelligence: The Integration of AI and Level Design in Halo." *GDC.*

n.d. "Han Solo Meme." Accessed May 19. http://www.quickmeme.com/p/3vn9yx.

Hi-Rez Studios. 2012. "Tribes: Ascend." April 12.

Interwave studios. 2011. "Nuclear Dawn." September 26.

jkdmedia. 2005. "Halo 2 Multiplayer Map Pack - XB - Review." *GameZone.* July 19. Accessed May 19, 2015. http://www.gamezone.com/reviews/halo_2_multiplayer_map_pack_xb_review.

Kremers, Rudolf. 2010. "Level Design: Concept, Theory, & Practice." Natick, Massachusetts: A K Peters, Ltd.

Latta, Lutz. 2004. "Building a Million-Particle System." *Gamasutra.* July. Accessed October 10, 2015. http://www.gamasutra.com/view/feature/130535/building_a_millionparticle_system.php.

Lipp, Markus, Peter Wonka, and Michael Wimmer. 2009. "Parallel Generation of L-Systems." http://peterwonka.net/Publications/pdfs/2009.VMV.Lipp.ParallelGenerationOfLSystems.final.pdf.

LucasArts. 2005. "Star Wars: Battlefront II." October 31.

Luna, Frank. 2012. "Introduction to 3D Game Programming with DirectX 11." Dulles, VA: Mercury Learning and Information.

Massey, Heather. 2012. "The Hollywod Elevator Pitch: Useful For Sci-Fi Romance." *The Galaxy Express.* January 29. Accessed May 19, 2019. http://www.thegalaxyexpress.net/2012/01/hollywood-elevator-pitch-useful-for-sci.html.

McAnlis, Colt. 2009. "GDC Vault - HALO WARS: The Terrain of Next-Gen." Accessed October 9, 2014. http://www.gdcvault.com/play/1277/HALO-WARS-The-Terrain-of.

Microsoft. 2015. "Halo."

—. 2009. "Halo Wars." Februrary 26.

n.d. "Natural Selection 2 ." *Natural Selection 2 Wiki.* Accessed May 19, 2015. http://www.wikiwand.com/en/Natural_Selection_2.

Nature's Crusaders. 2009. *Nature's Crusaders.* September. Accessed May 19, 2015. https://naturescrusaders.files.wordpress.com/2009/09/arz_strip_mine.jpg.

Next Gamer. 2012. "Im Test: Borderlands 2- Back to Pandora." October 6. Accessed May 19, 2015. http://www.next-gamer.de/wp-content/uploads/2012/10/Borderlands-2.jpg.

Nguyen, Hubert. n.d. "GPU Gems 3 - Foreword." *NVIDIA Developer Zone.* Accessed October 14, 2014. http://http.developer.nvidia.com/GPUGems3/gpugems3_pref01.html.

Nintendo. 1985. "Super Mario Bros." September 13.

Oyarijivbie, S. 2012. "Modular Content Design."

Palmer, Jason. n.d. *Firefly Art Print.*

PC Gamer. 2014. "Battleborn Gameplay." September 21. Accessed May 19, 2015. http://36646d87786feafc0611-0338bbbce19fc98919c6293def4c5554.r0.cf1.rackcdn.com/images/YrS-hq183zpw.878x0.Z-Z96KYq.jpg.

Perry, Lee. 2002. "Modular Level and Component Design - Or: How I learned to Stop Worrying and Love Making High-Detail Worlds." Game Developer Magazine, November. 30-35.

Peters, Zi. 2013. "Naughty Dog’s Uncharted Level Design Process." 2013, March 14. http://zipeters.com/2013/03/14/naughty-dogs-uncharted-level-design-process/.

Prusinkiewicz, Przemyslaw, and Aristid Lindenmayer. 1990. "The Algorithmic Beauty of Plants." New York: Springer-Verlag. http://algorithmicbotany.org/papers/abop/abop.pdf.

psmilek. 2011. "Voxel Terrain Engine - Work in Progress - YouTube." Accessed October 9, 2014. http://www.youtube.com/watch?v=Rc1ztktWxJs.

Respawn Entertainment. 2014. "Titanfall." March 11.

Riot Games. 2015. "League of Legends." Los Angeles.

Schell, J. 2008. "The Art of Game Design: A Book of Lenses." Burlington, Massachusetts: Morgan Kaufmann.

Sega. 1993. "Gunstar Heroes." September 9.

2013. "Shotgun Cock Reload Layered (Powerful)." September 2013. Accessed May 19, 2015. https://www.freesound.org/people/Paul368/sounds/200966.

Snelling, S, and S Garozzo. 2015. "Community Level Design for Competitive CS:GO." *GDC.* http://www.gdcvault.com/play/1021868/Community-Level-Design-for-Competitive.

n.d. "Source Multiplayer Networking." Accessed October 14, 2014. https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking.

stewdio2003. 2014. "CP_Power_Down01.aif." May 22. Accessed May 19, 2015. https://www.freesound.org/people/stewdio2003/sounds/238367.

Stirling, Rick. 2007. "Yes, but how many polygons?" RSArt, August 27. http://www.rsart.co.uk/2007/08/27/yes-but-how-many-polygons/ .

Swink, Steve. 2009. *Game Feel.* CRC Press.

Takahashi, Dean. 2015. "How Treyarch evolved its own brand of Call of Duty multiplayer." *Yahoo! Games.* April 26. Accessed May 19, 2015. https://games.yahoo.com/news/treyarch-evolved-own-brand-call-173008704.html.

ten by twenty. n.d. "Akashi Font." http://tenbytwenty.com/?xxxx_posts=akashi .

Tictac Town. n.d. "League of Legends." *Tictac Town.* Accessed May 19, 2015. http://www.tictactown.com/tr/league-of-legends-577-oyundetay.

Tom's Hardware. 2014. "App smartphone e tablet per giocare come un professionista." *Tom's Hardware.* July 23. Accessed May 19, 2015. http://www.tomshw.it/files/2014/07/immagini_contenuti/57626/5.jpg.

Unknown Worlds Entertainment. 2010. "Natural Selection." San Francisco.

Walker, Jordan. 2014. "Physically Based Shading in UE4." May 1. Accessed May 19, 2015. https://forums.unrealengine.com/showthread.php?4049-Physically-Based-Shading-in-UE4.

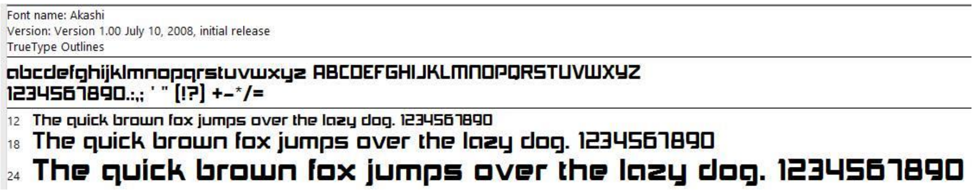
Widmark, Mattias. 2012. "Terrain in Battlefield 3: A Modern, Complete and Scalable System." EA Digital Illusions (DICE), March. http://dice.se/wp-content/uploads/gdc12_terrain_in_battlefield3.pdf.

Zak, Mark. 2008. "Environment Design in Halo 3." *GDC.* http://halo.bungie.net/images/Inside/publications/presentations/Bungie_EnvDesign_GDC.pptx.

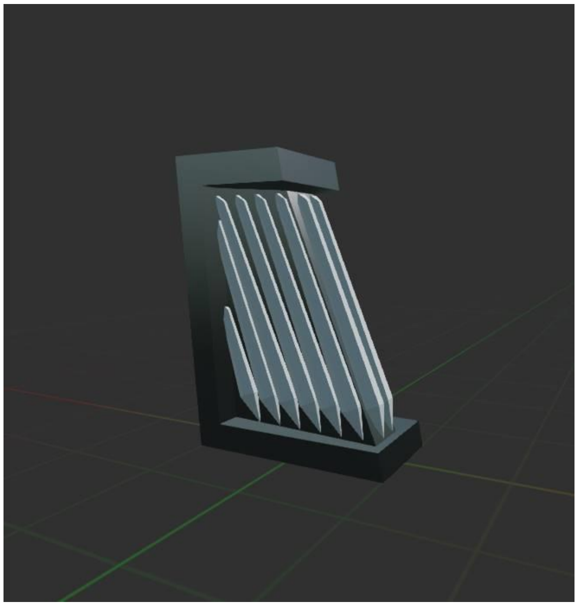
Appendices

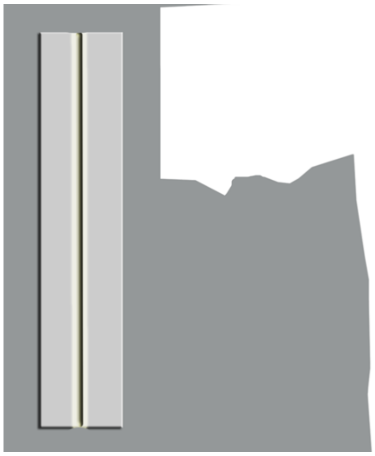
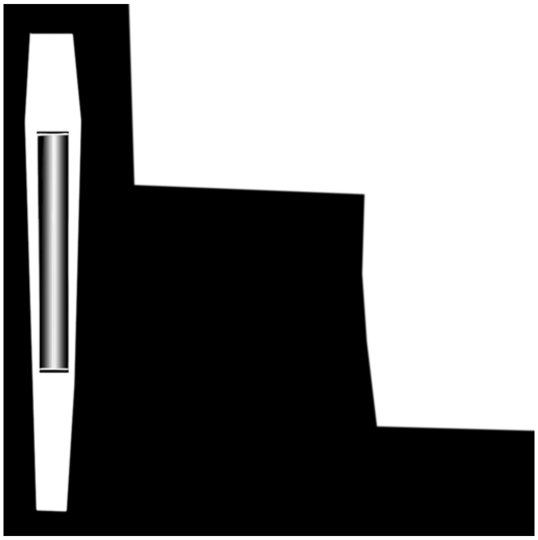
A.1 Visual Assets

A.1.1 Akashi Font

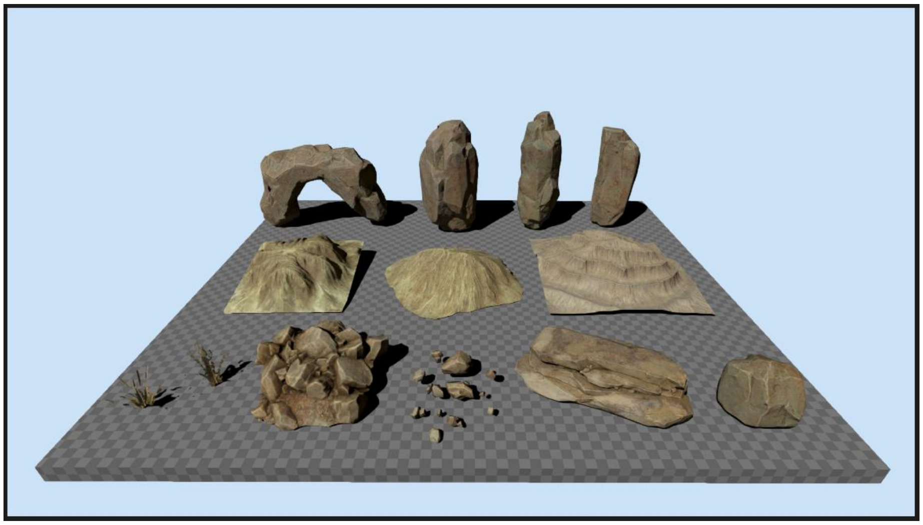
 Font name: Akashi Version: Version 1.00 July 10, 2008, initial release TrueType Outlines abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890.:,; ' " [!?] +-*/= 12 The quick brown fox jumps over the lazy dog. 1234567890 18 The quick brown fox jumps over the lazy dog. 1234567890 24 The quick brown fox jumps over the lazy dog. 1234567890	Akashi Font (ten by twenty n.d.)

A.1.2 Ammo

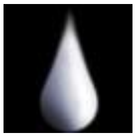
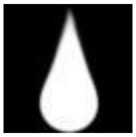
	Ammo 352 polygons

	Ammo Color
	Ammo Emissivity Mask

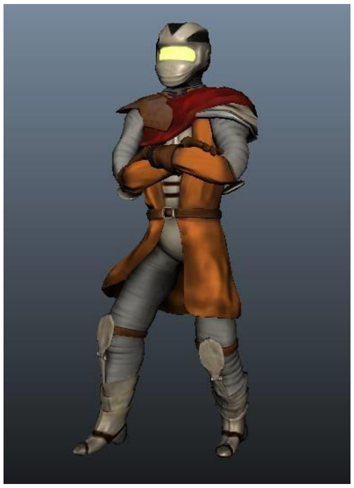
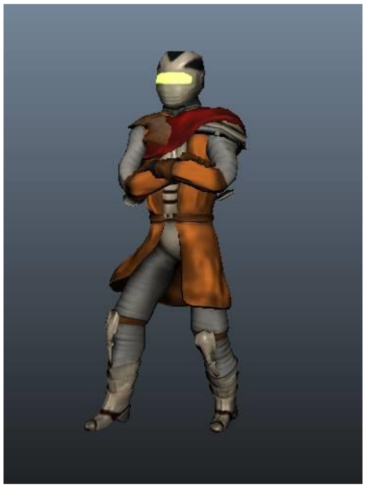
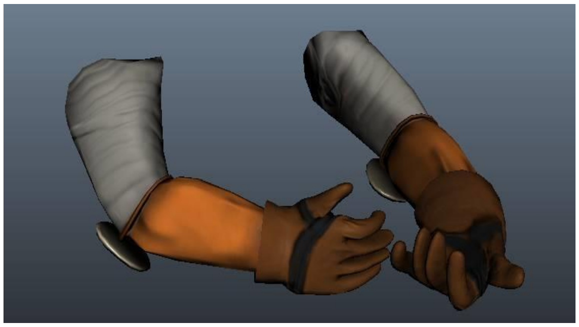
A.1.3 Arid Desert Pack

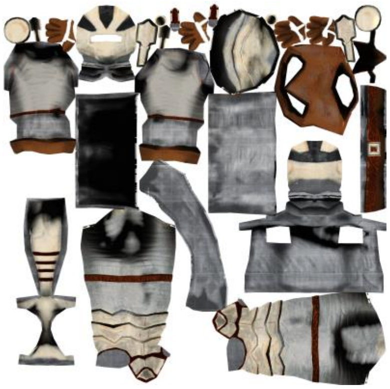
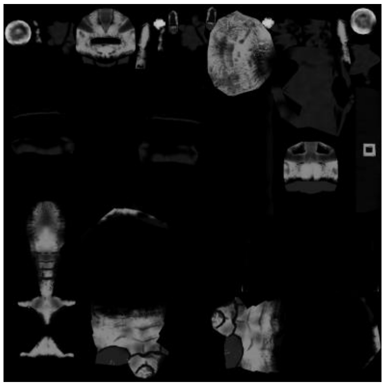
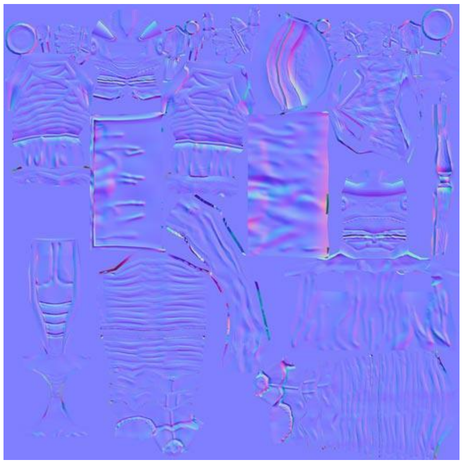
	Arid Desert Modular Pack (Baybayan n.d.)

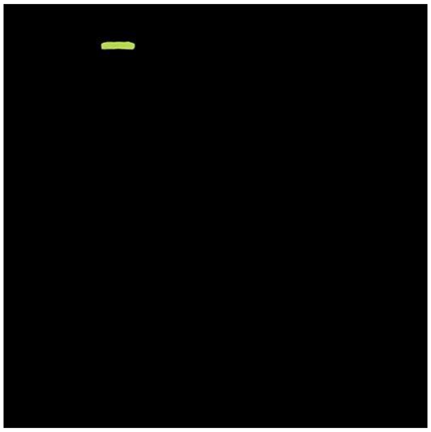
A.1.4 Cave Drip

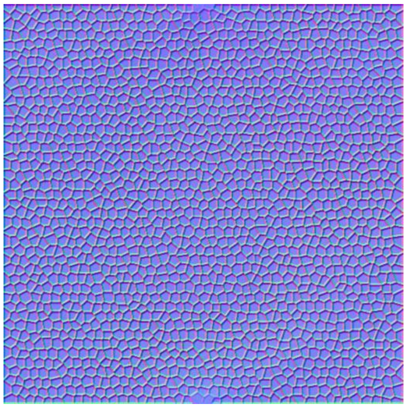
	Cave Drip Color
	Cave Drip Mask

A.1.5 Character

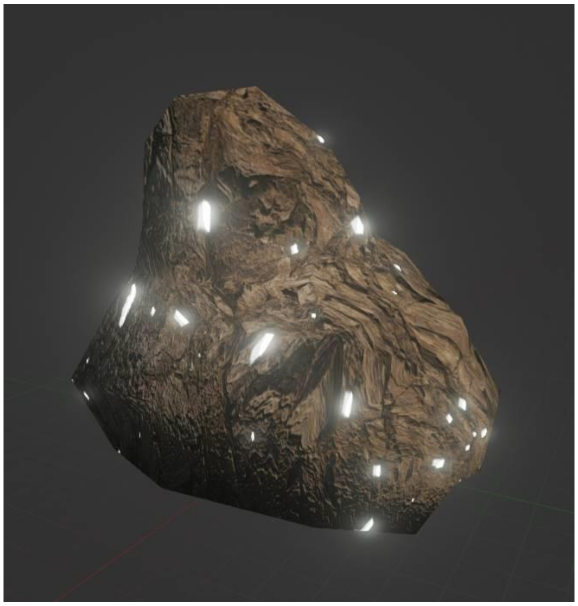
	Character Model, Full Detail 14,584 Polygons
	Character Model, Low Detail 5,570 Polygons
	Character Model, First Person Arms 3,836 Polygons

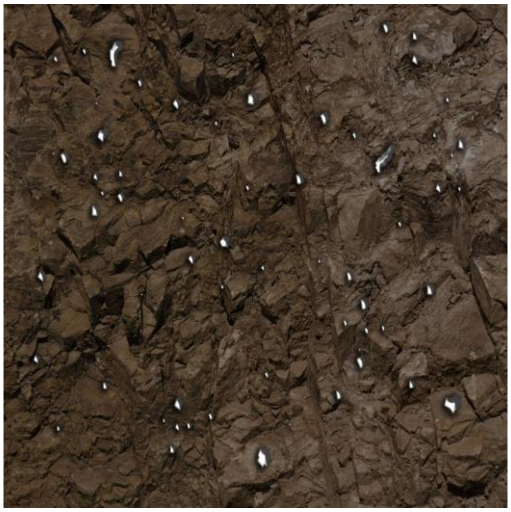
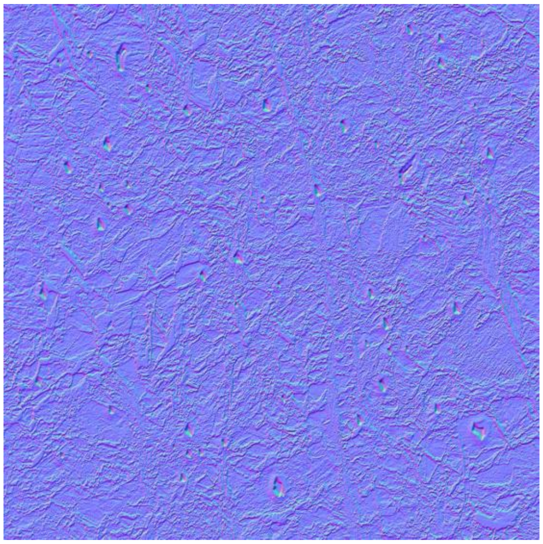
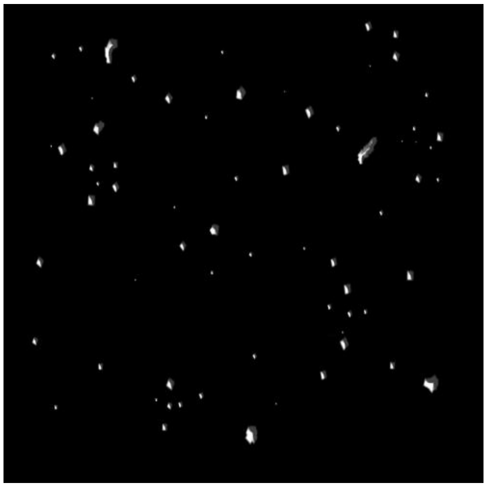
	Character Color
	Character Specularity/Roughness
	Character Normals

	Character Emissive Color
	Character Color Mask 1
	Character Color Mask 2

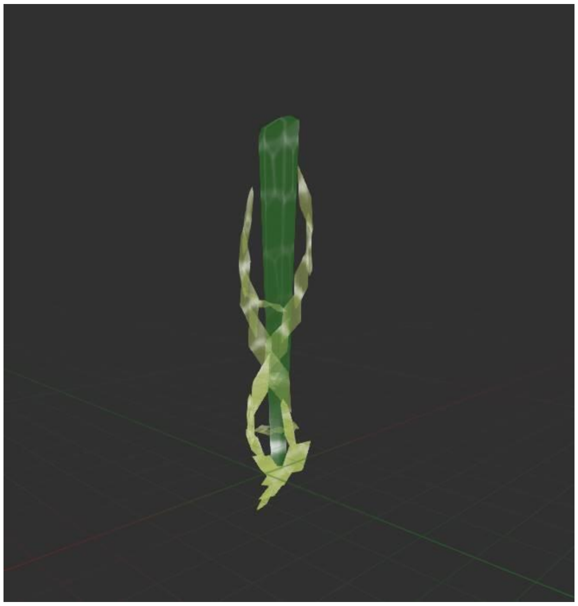
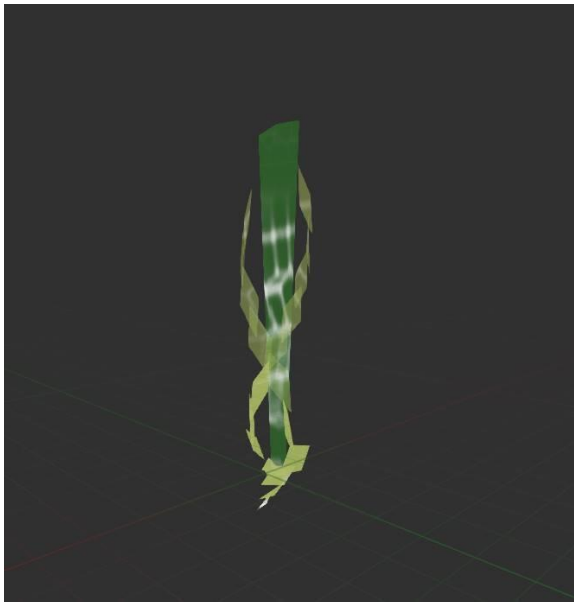
	Character Shield Normals

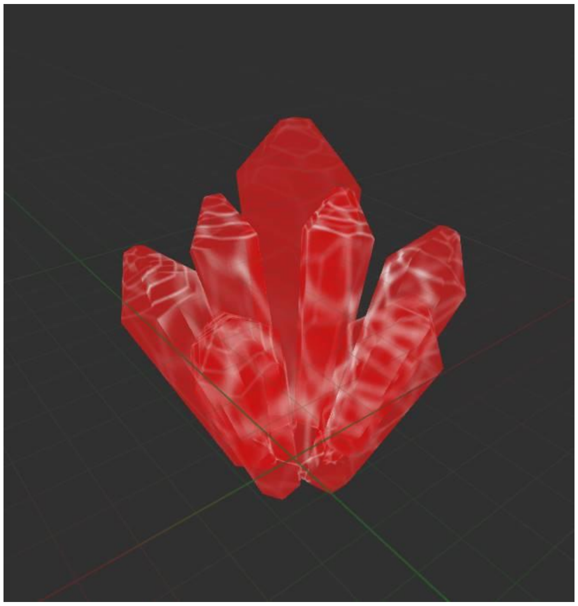
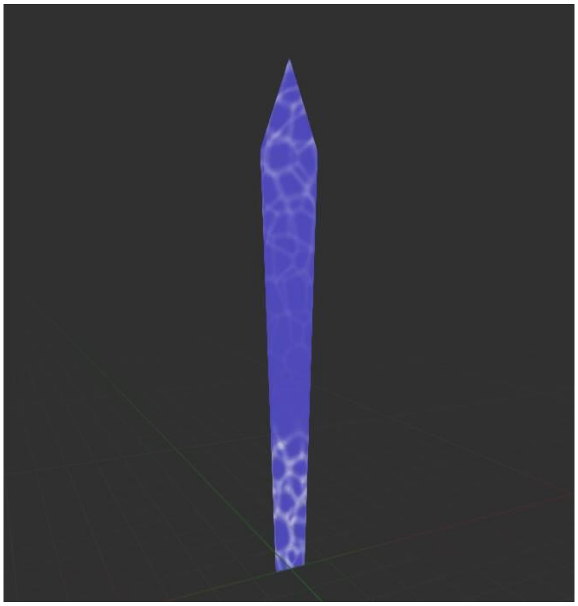
A.1.6 Crystal Rock

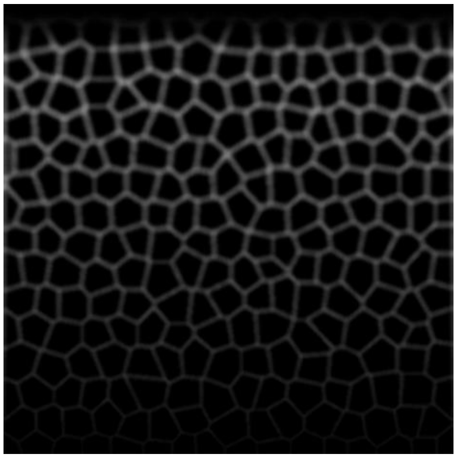
	Crystal Rock 144 Polygons

	Crystal Rock Color
	Crystal Rock Normals
	Crystal Rock Emissivity Mask

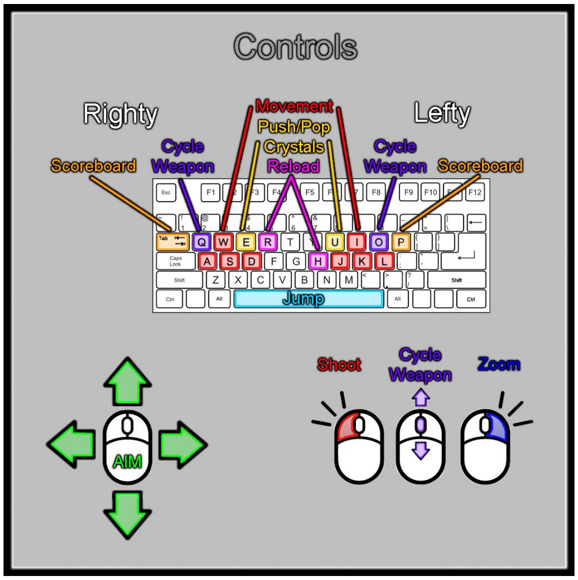
A.1.7 Crystals

	Energy Crystal, Full Detail 1588 Polygons
	Energy Crystal, Low Detail 382 Polygons
	Energy Crystal Color

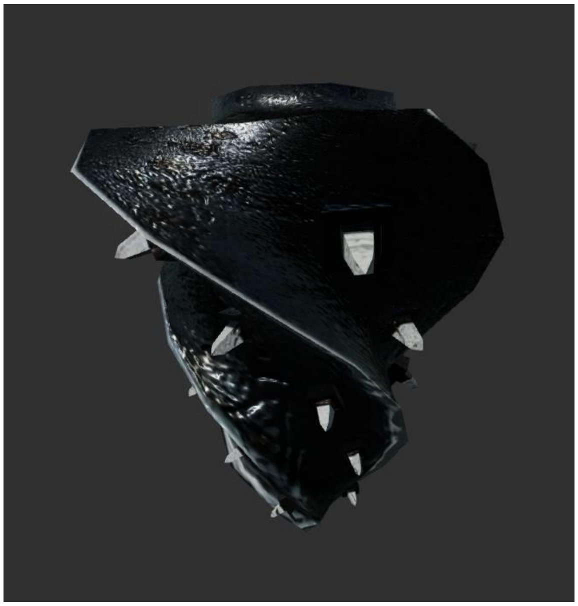
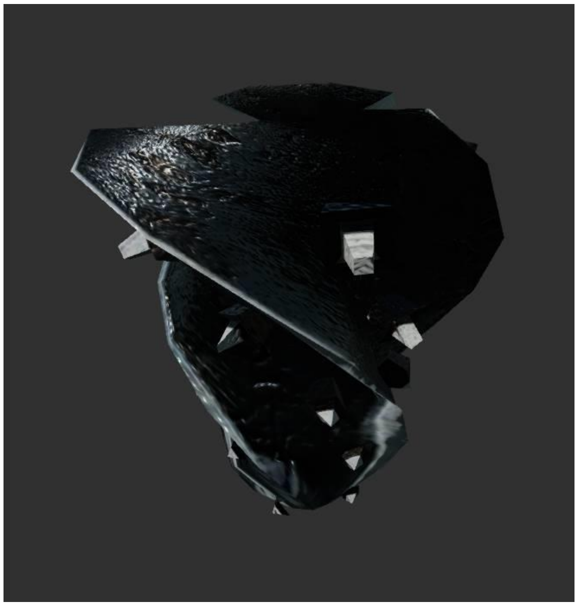
	Power Crystal 262 Polygons
	Accuracy Crystal 22 Polygons

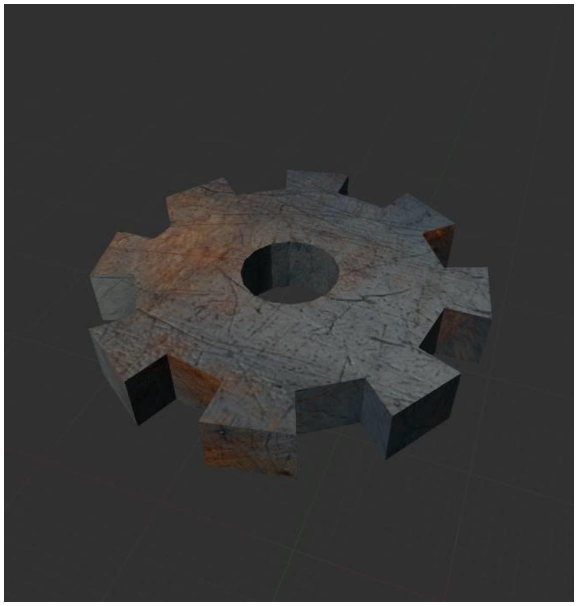
	Crystal Emissivity Mask

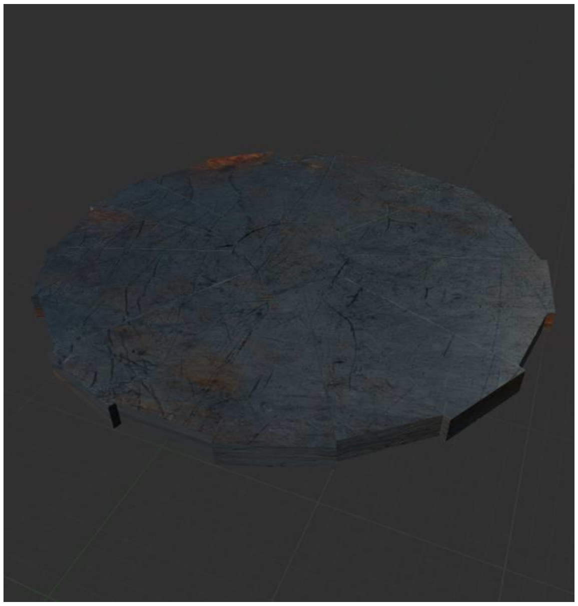
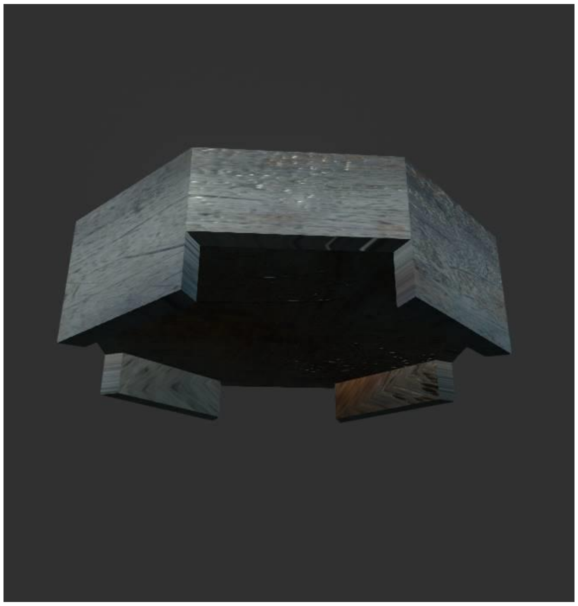
A.1.8 Controls Image

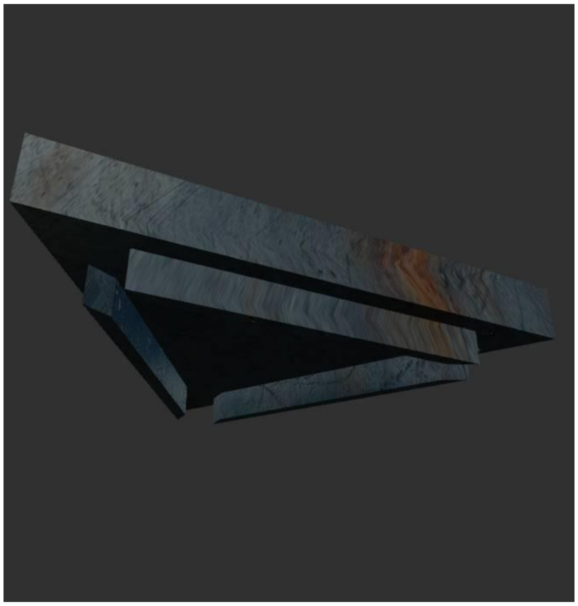
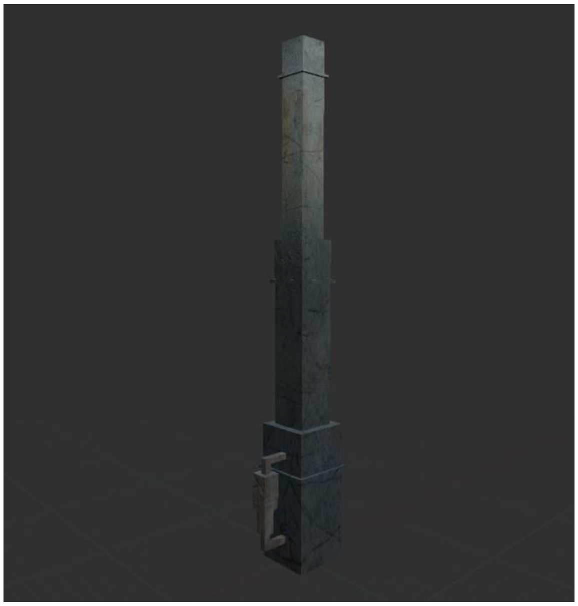
 Controls Righty Lefty Movement Push/Pop Crystals Reload Cycle Weapon Scoreboard Cycle Weapon Scoreboard Jump AIM Shoot Cycle Weapon Zoom	Controls Image

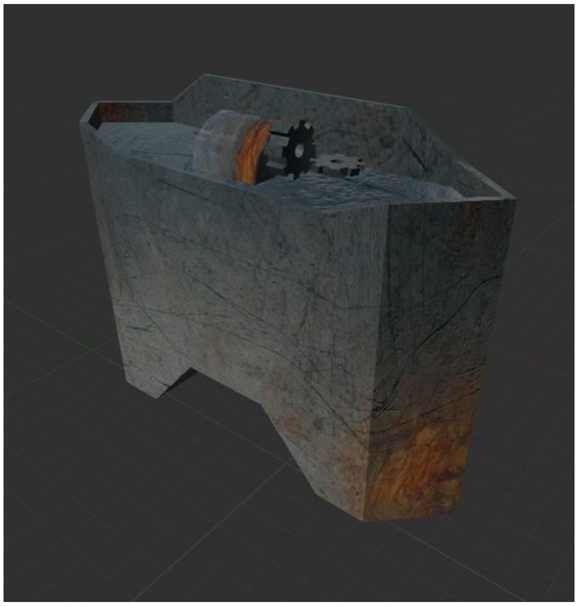
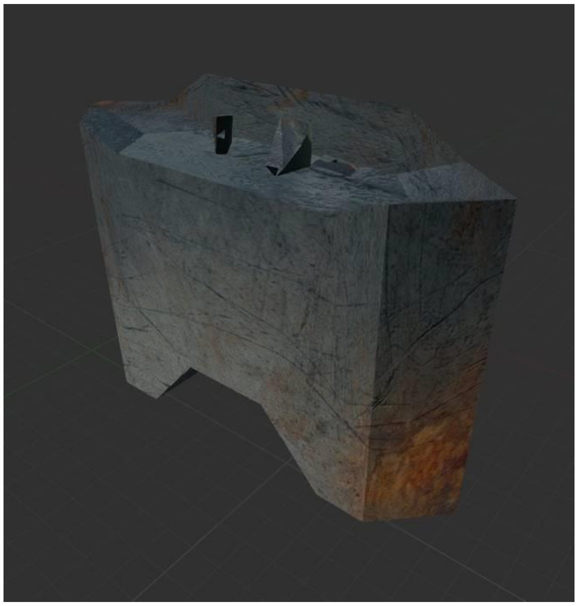
A.1.9 Drill

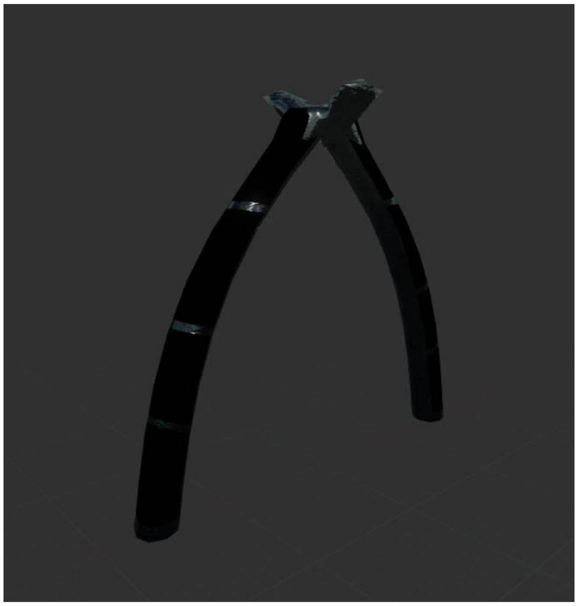
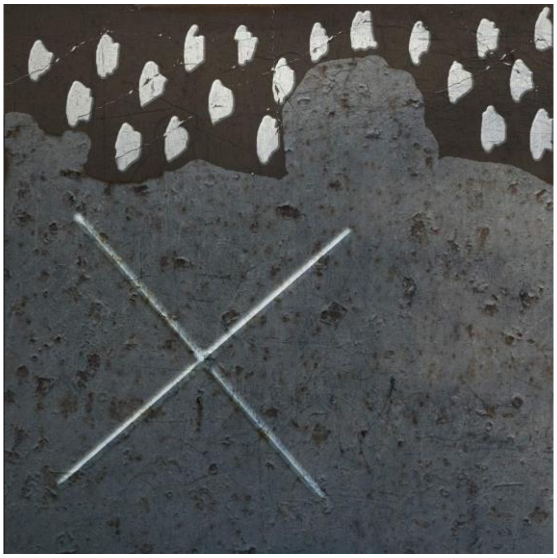
	Drill, Full Detail 1,232 Polygons
	Drill, Low Detail 460 Polygons

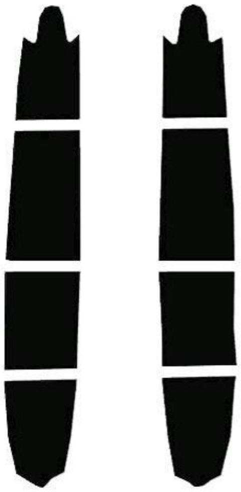
	Drill Gear 192 Polygons
	Drill Cylinder 32 Polygons

	Drill Cover 360 Polygons
	Drill Top Quad 94 Polygons

	Drill Top Tri 66 Polygons
	Drill Leg 264 Polygons

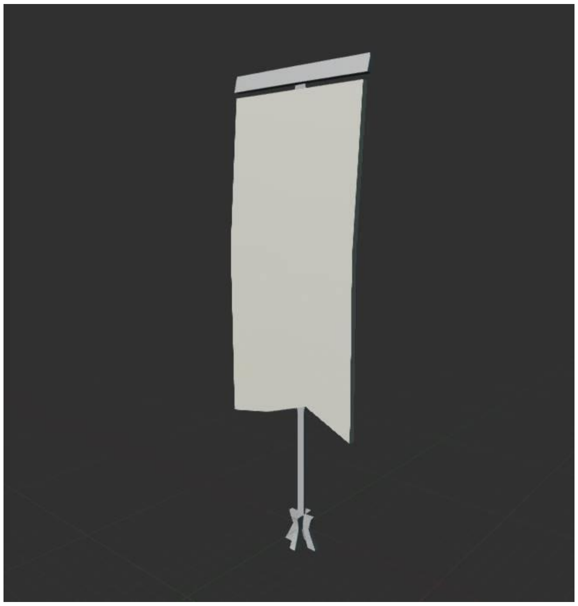
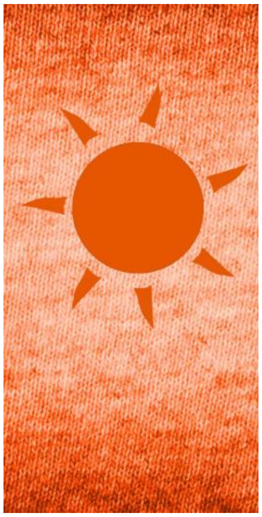
	Drill Foundation, Full Detail 708 Polygons
	Drill Foundation, Low Detail 122 Polygons

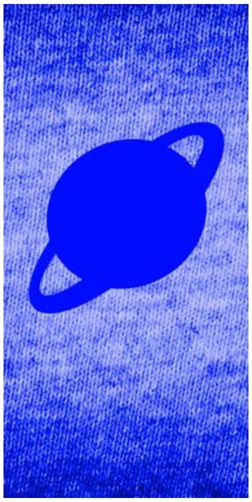
	Drill Energy Collector 496 Polygons
	Drill Color
	Drill Emissivity Mask

	Drill Color (For all other pieces)
	Drill Energy Collector Color
	Drill Energy Collector Color Mask

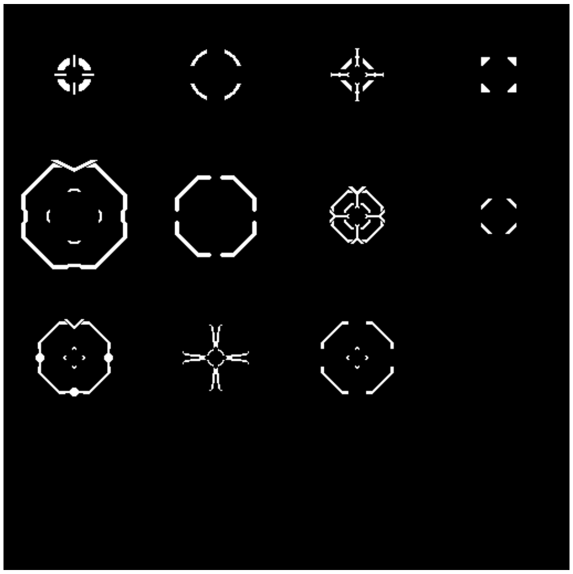
	Drill Dirt Particle

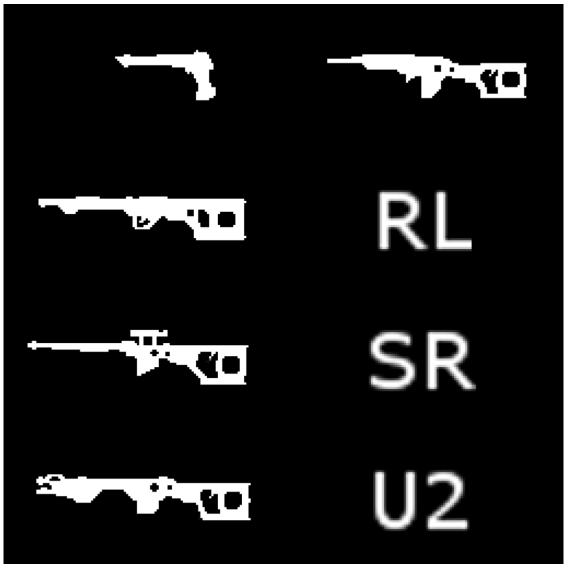
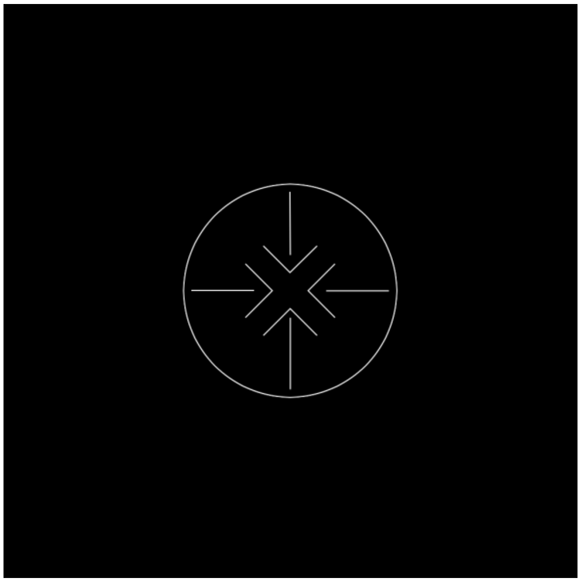
A.1.10 Flag

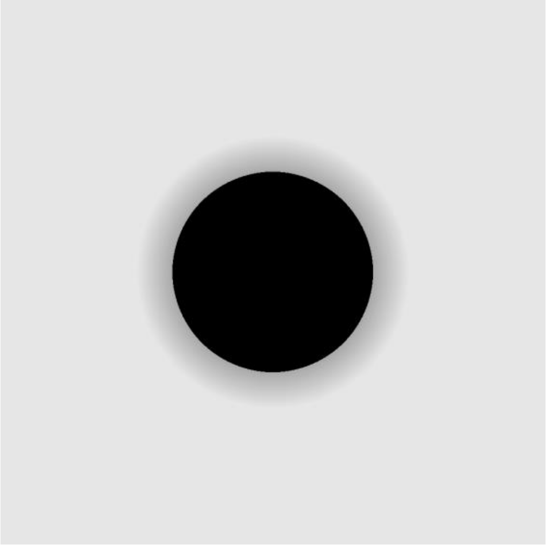
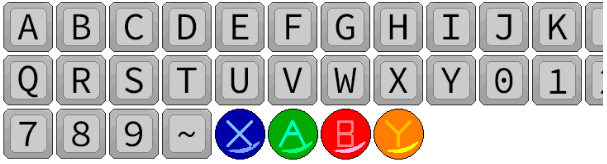
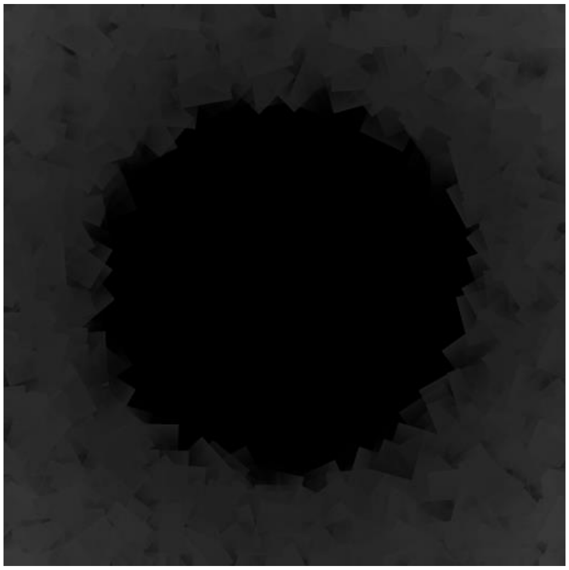
	Flag 160 Polygons
	Flag Color, “Team Orange Suns”

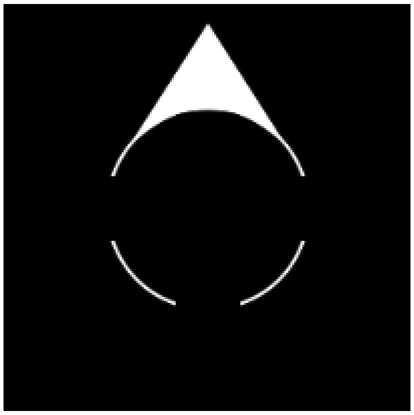
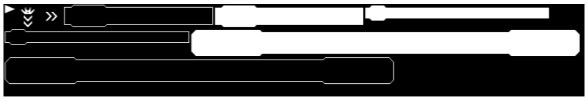
	Flag Color, “Team Blue Rings”

A.1.11 HUD

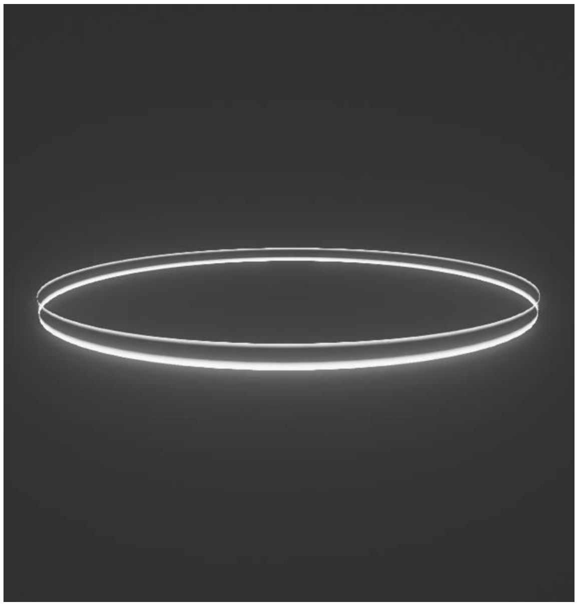
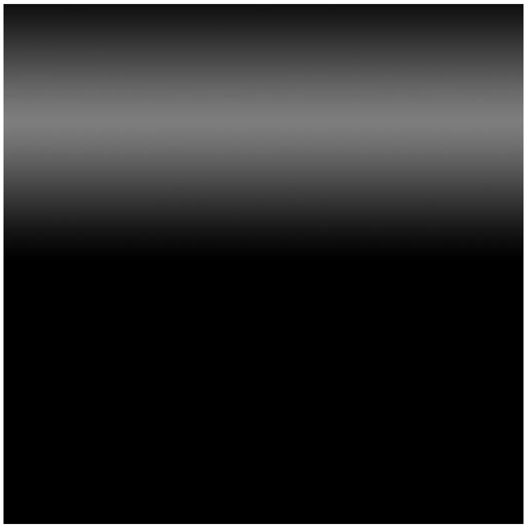
	Crosshair Atlas

	Weapon Icons
	Marksman Rifle Zoom Overlay

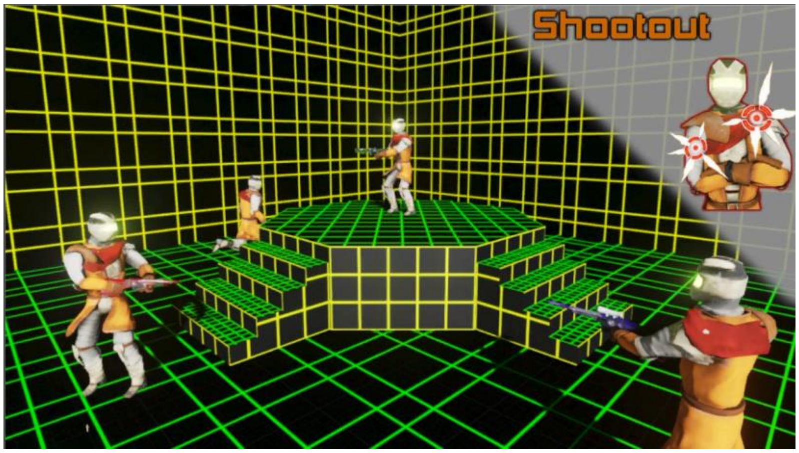
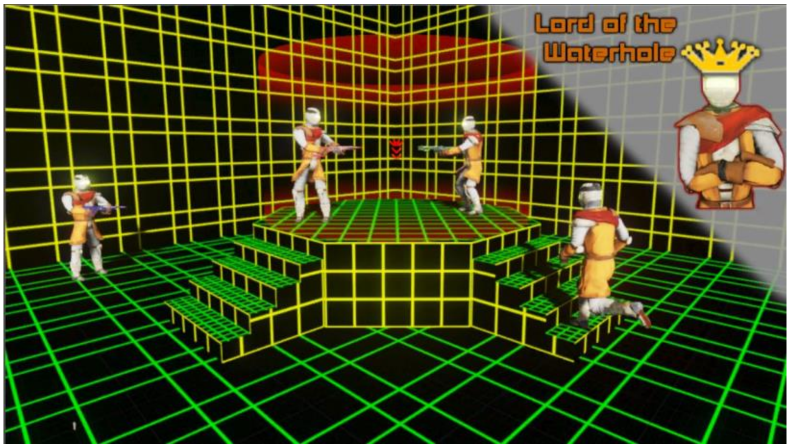
	Marksman Rifle Zoom Mask
 A B C D E F G H I J K Q R S T U V W X Y 0 1 7 8 9 ~ X A B Y	Input Key Atlas
	Hit Overlay

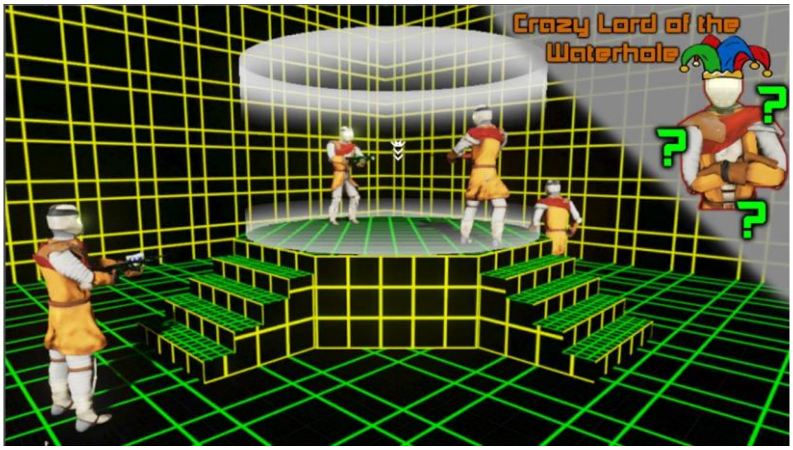
	Hit Indicator
	HUD Element Atlas
	Lower Helmet
	Lower Flasher
	Info Flasher
	Upper Helmet
	Upper Flasher

A.1.12 King of the Hill Indicator

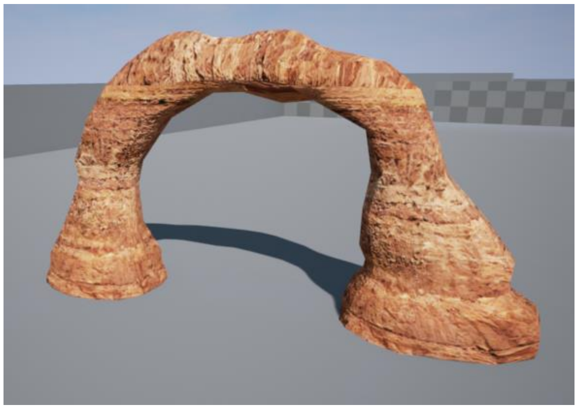
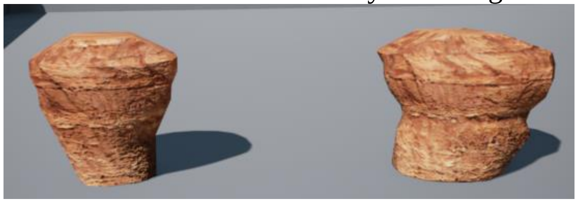
	King of the Hill Indicator (3D) 128 Polygons
	King of the Hill Indicator Color Mask

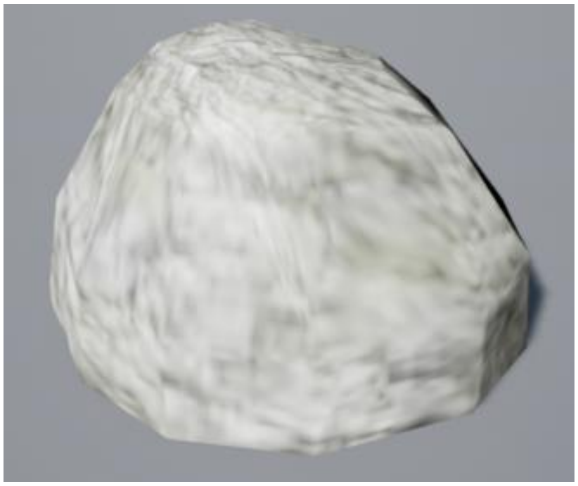
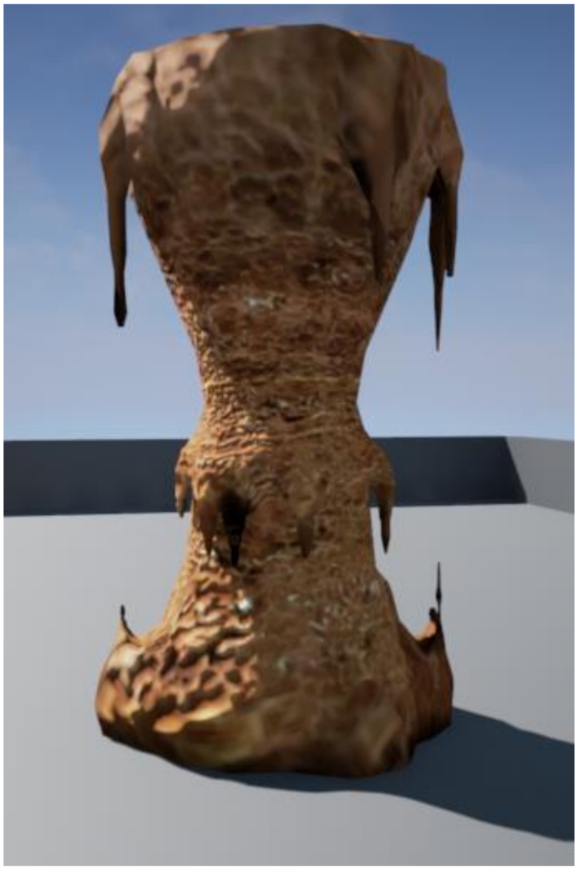
A.1.13 Lobby Images

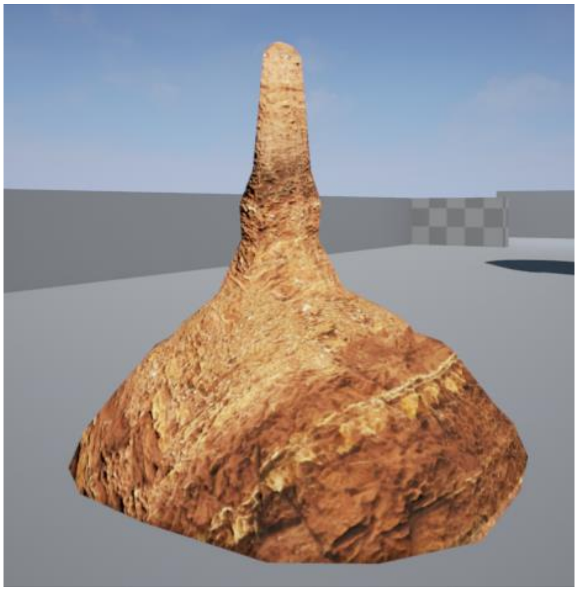
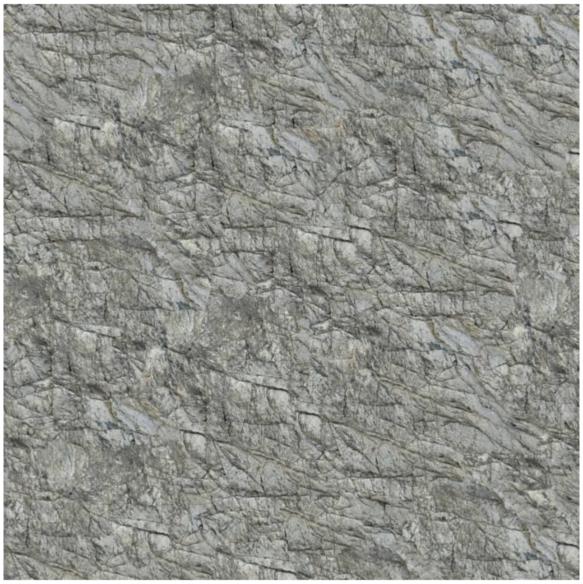
Crystal Canyon	Lobby Image, Map: Crystal Canyon
	Lobby Image, Game Mode: Shootout
	Lobby Image, Game Mode: Lord of the Waterhole

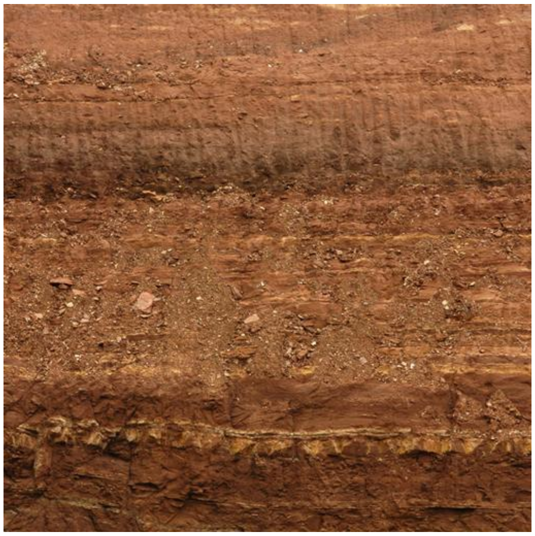
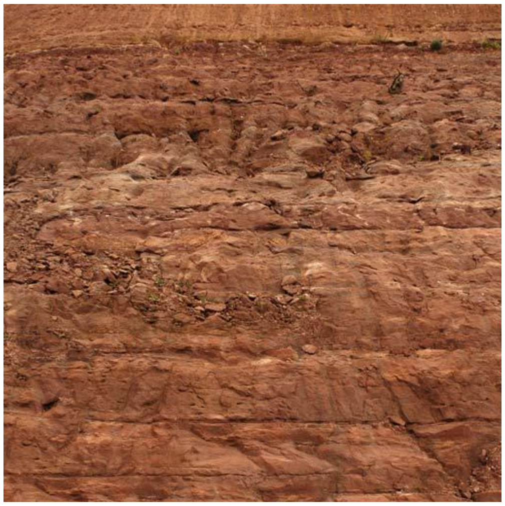
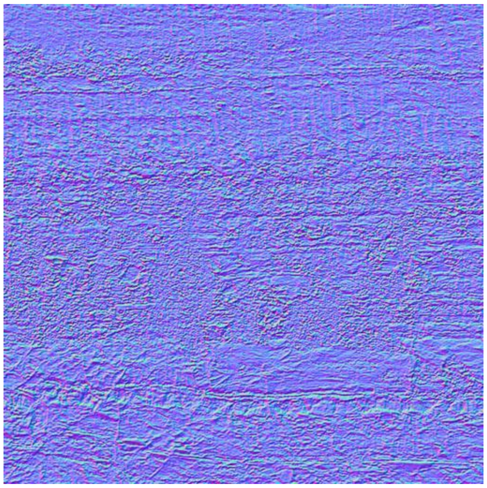
	Lobby Image, Game Mode: Crazy Lord of the Waterhole

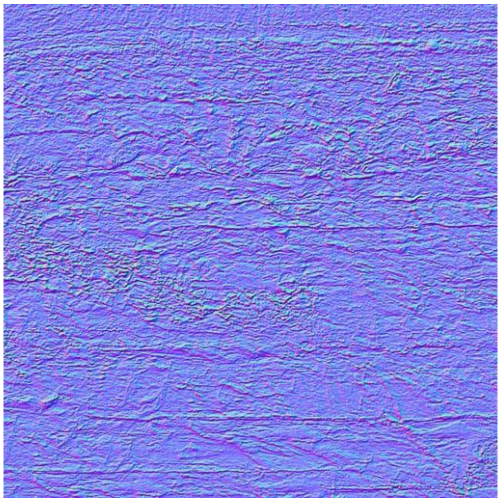
A.1.14 Modular Art Set

 Shown above - Normal archway rock usage  Shown above - Alternative archway rock usage	Archway Rock 992 Polygons [Created in 3 sizes (small, medium, and large) for ease of use in the modular set. Small, medium, and large sizes were determined via player scale to offer unique cover types. It was decided to make 3 exports as to save the designer time in editor removing the need to set the scale to certain numbers.] [Also shown is the arch rock flipped 180 degrees around the X-Axis. This is an example of how modular design can effectively create multiple assets out of a single asset.]

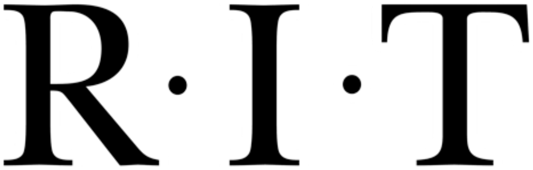
	Boulder Rock 108 Polygons [Created in 3 sizes (small, medium, and large) for ease of use in the modular set. Small, medium, and large sizes were determined via player scale to offer unique cover types. It was decided to make 3 exports as to save the designer time in editor removing the need to set the scale to certain numbers.]
	Cave Rock 1580 Polygons

	Spike Rock 384 Polygons [Created in 3 sizes (small, medium, and large) for ease of use in the modular set. Small, medium, and large sizes were determined via player scale to offer unique cover types. It was decided to make 3 exports as to save the designer time in editor removing the need to set the scale to certain numbers.]
	Small Rock Color From CGTextures.com

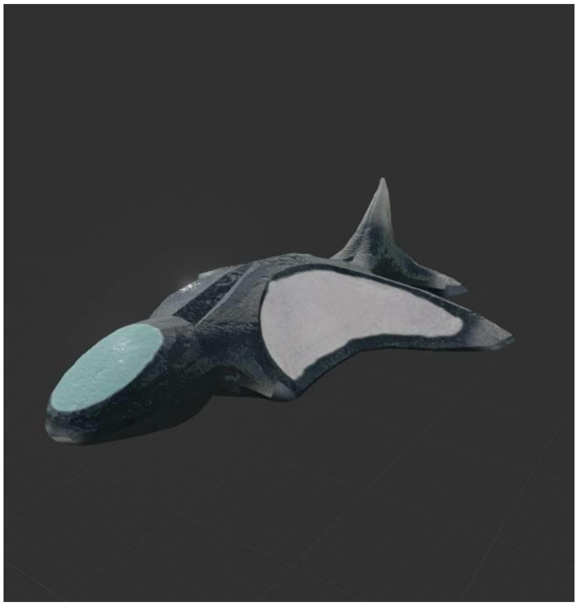
	Desert Rock Sediment 1 Color From CGTextures.com
	Desert Rock Sediment 2 Color From CGTextures.com
	Desert Rock Sediment 1 Normals

	Desert Rock Sediment 2 Normals

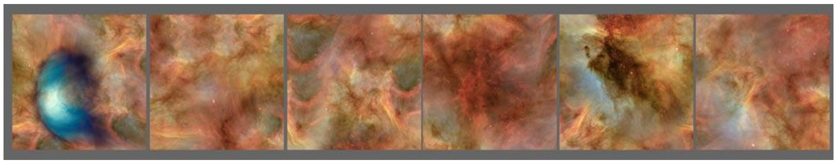
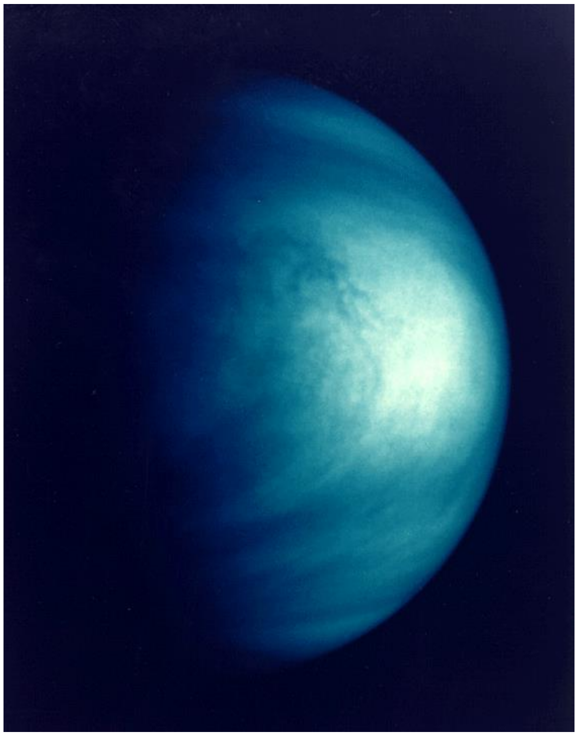
A.1.15 RIT Logo

	RIT Logo https://www.rit.edu/upub/logos

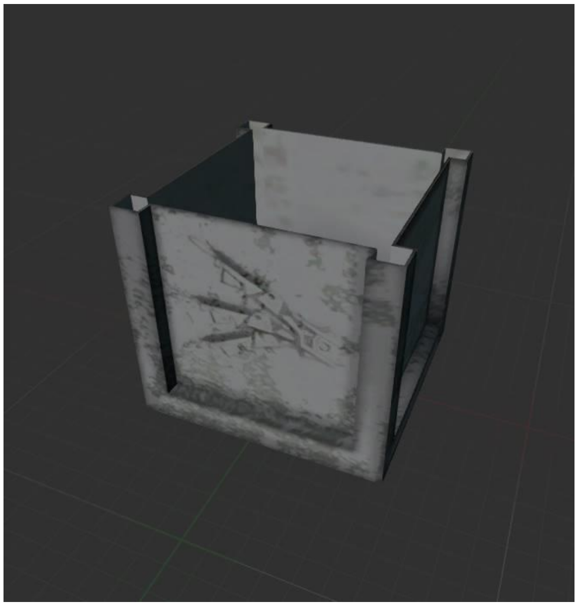
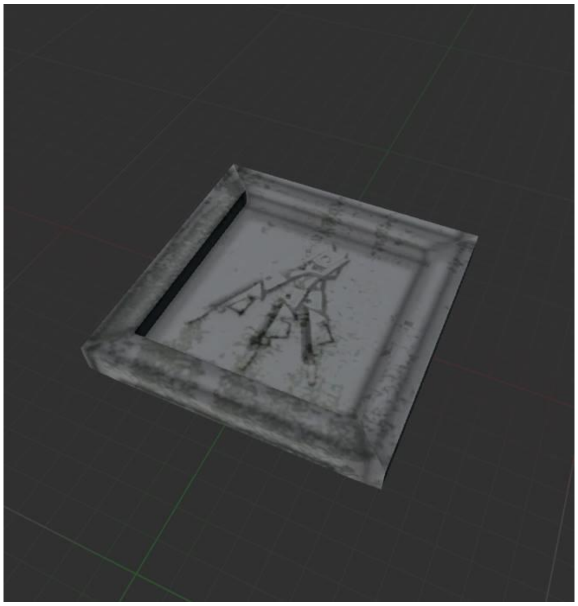
A.1.16 Ship

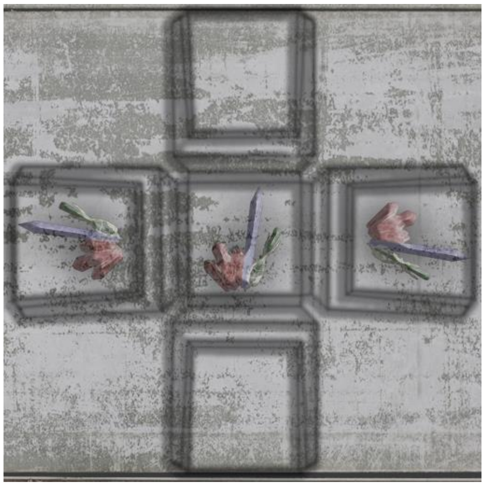
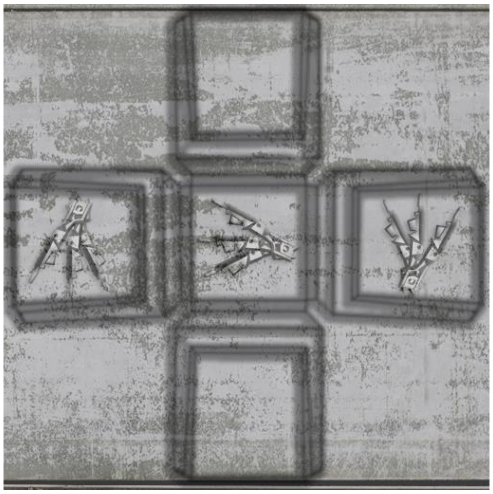
	Ship 720 Polygons
	Ship Color
	Ship Emissivity Mask

A.1.17 Skybox

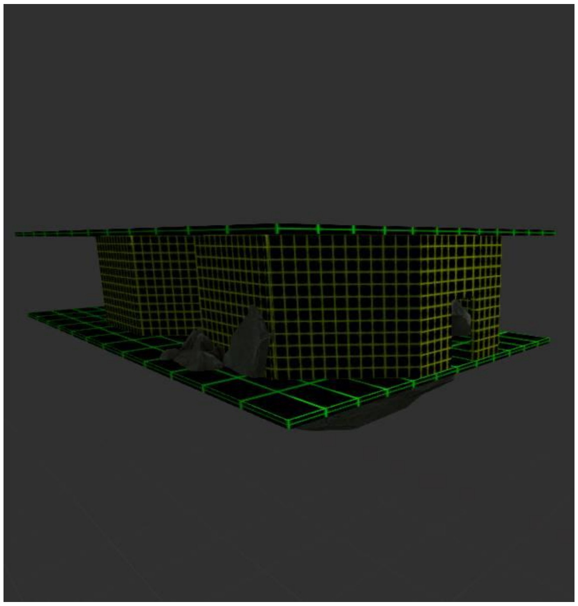
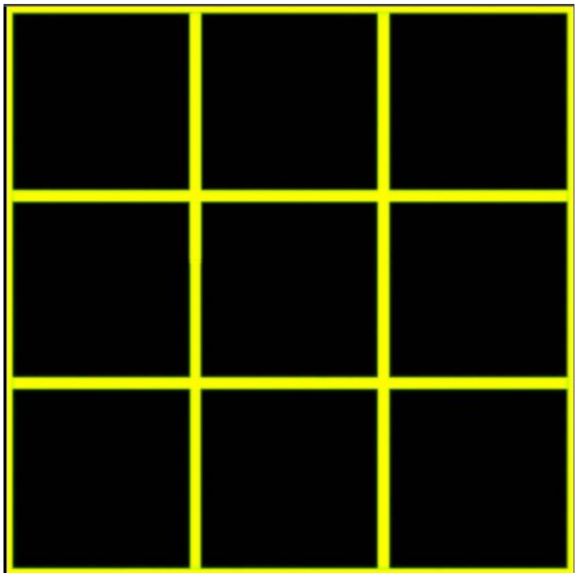
	Skybox
	Skybox Source Image 1 The Carina Nebula http://www.freeimageslive.com/galleries/space/nebula/pics/hst_carina_ngc3372_0006.jpg
	Skybox Source Image 2 Venus through an Ultraviolet Filter http://www.freeimageslive.com/galleries/space/planets/pics/gal_p37218.gif

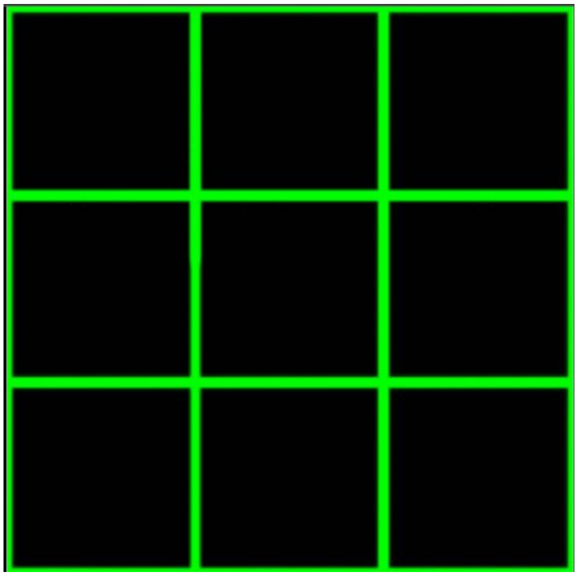
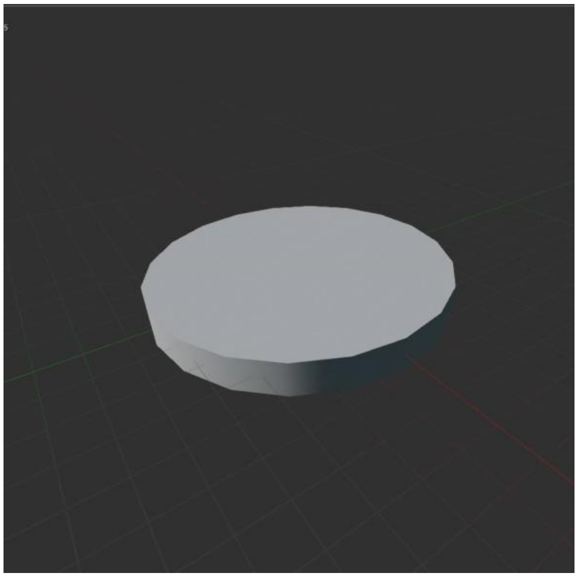
A.1.18 Space Crate

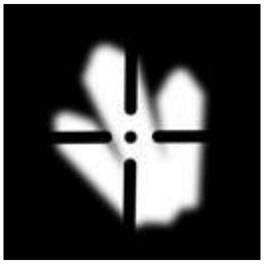
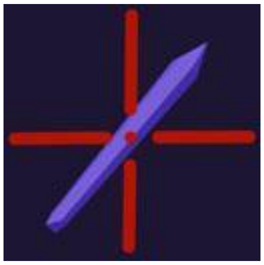
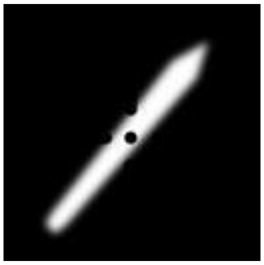
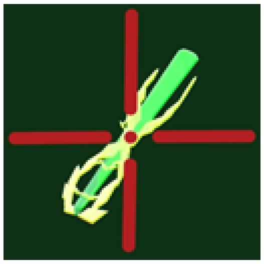
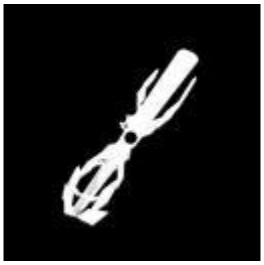
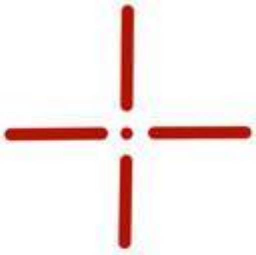
	Space Crate 188 Polygons
	Space Crate Lid 124 Polygons

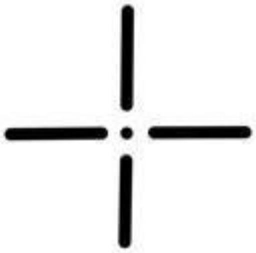
	Crystal Crate Color
	Ammo Crate Color

A.1.19 Tutorial Room

	Tutorial Room BSP Geometry and Arid Desert Rocks
	Tutorial Room Color 1

	Tutorial Room Color 2
	Button 60 Polygons
	Button Shotgun Color

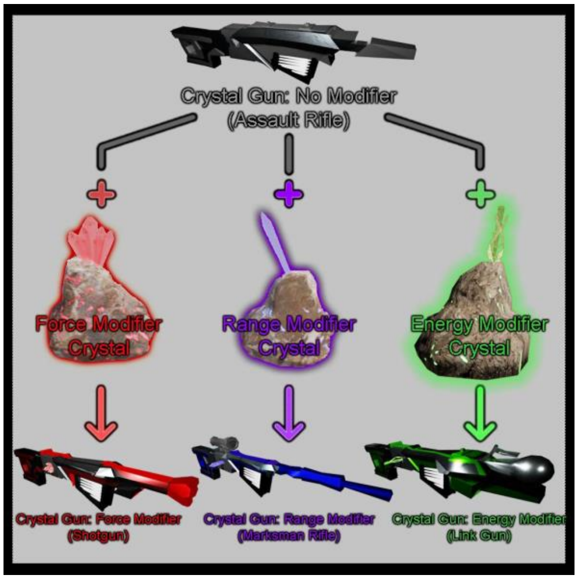
	Button Shotgun Emissivity Mask
	Button Marksman Rifle Color
	Button Marksman Rifle Emissivity Mask
	Button Beam Gun Color
	Button Beam Gun Emissivity Mask
	Button Assault Rifle Color

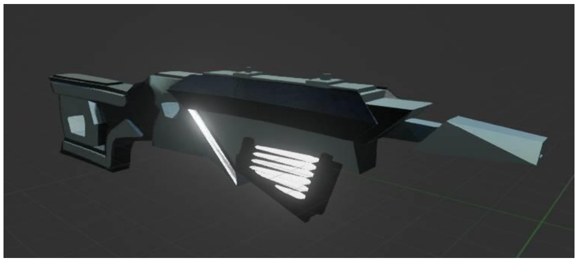
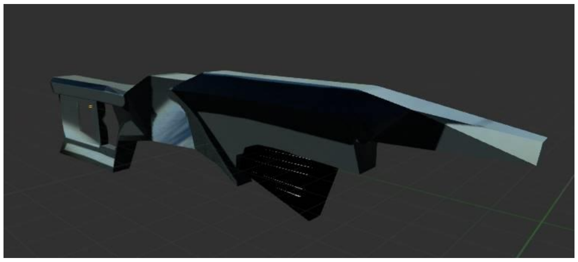
	Button Assault Rifle Emissivity Mask

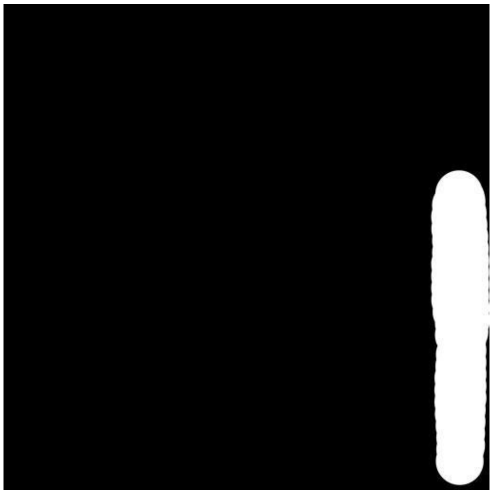
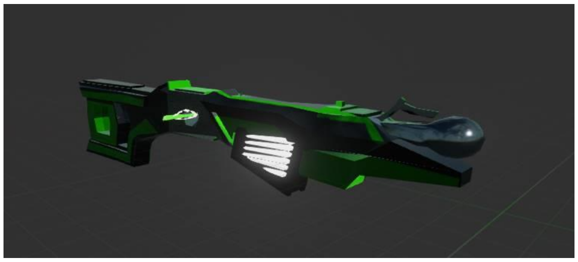
A.1.20 Unreal Logo

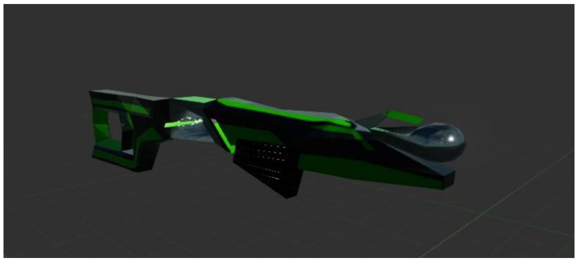
POWERED BY UNREAL ENGINE	Unreal Engine 4 Logo

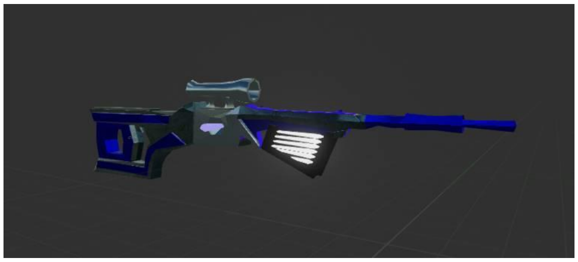
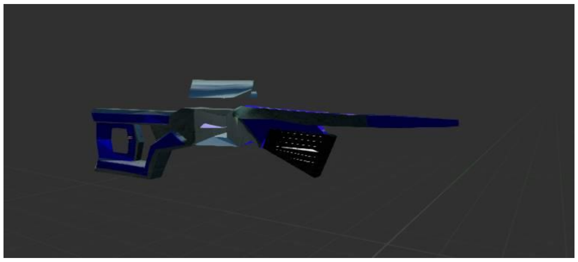
A.1.21 Weapons

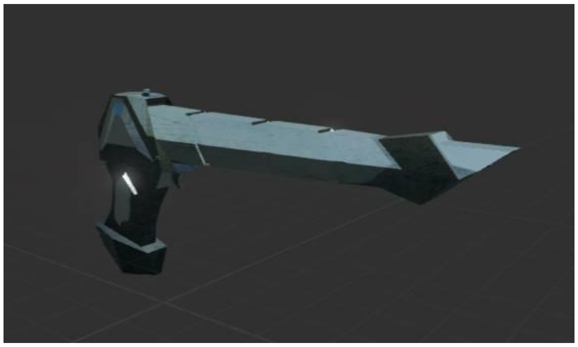
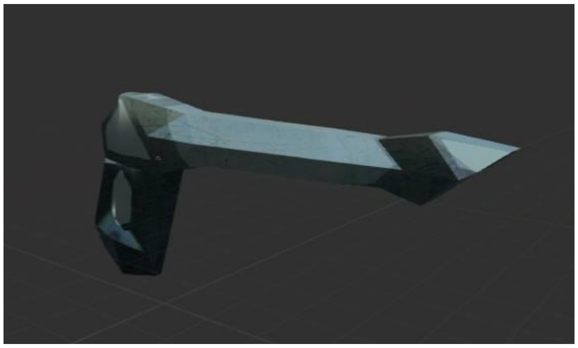
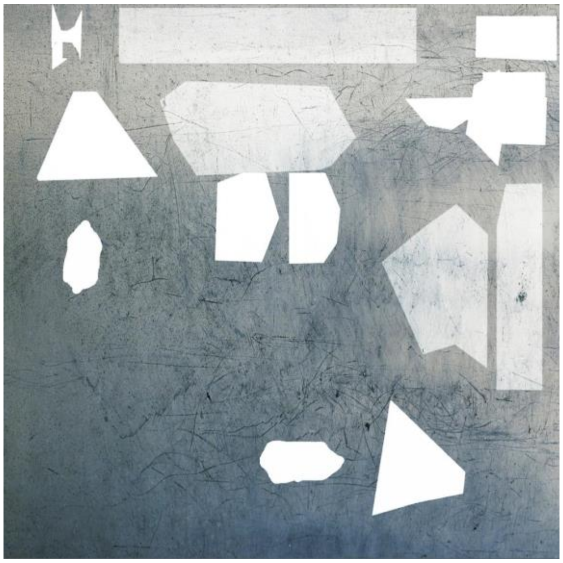
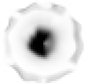
 Crystal Gun: No Modifier (Assault Rifle) Force Modifier Crystal Range Modifier Crystal Energy Modifier Crystal Crystal Gun: Force Modifier (Shotgun) Crystal Gun: Range Modifier (Marksman Rifle) Crystal Gun: Energy Modifier (Link Gun)	Weapon System Overview Image

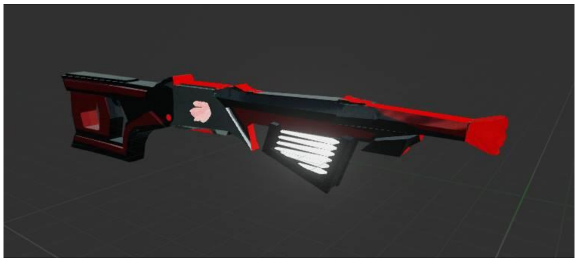
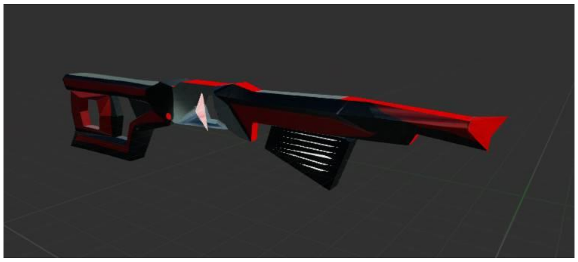
	Weapons, Assault Rifle, Full Detail 1,248 Polygons
	Assault Rifle, Low Detail 400 Polygons

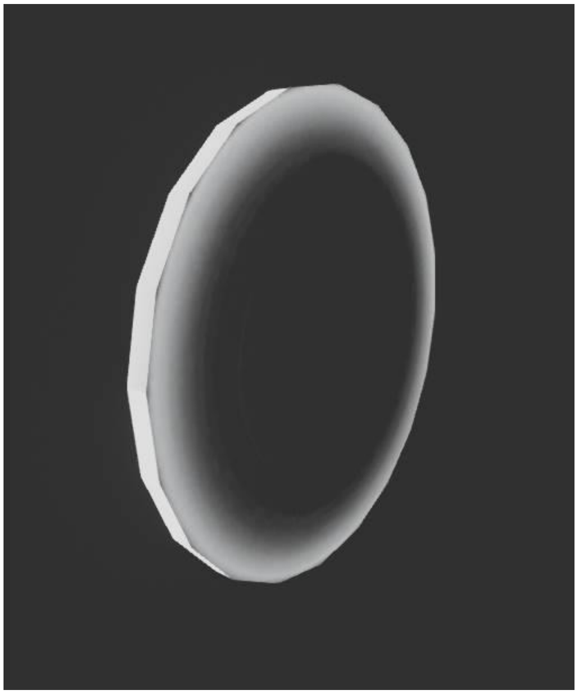
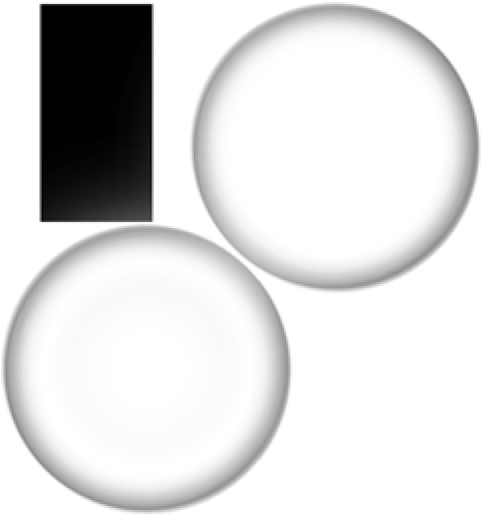
	Assault Rifle Color
	Assault Rifle Emissivity Mask
	Assault Rifle Bullet Hole
	Beam Gun, Full Detail 2,836 Polygons

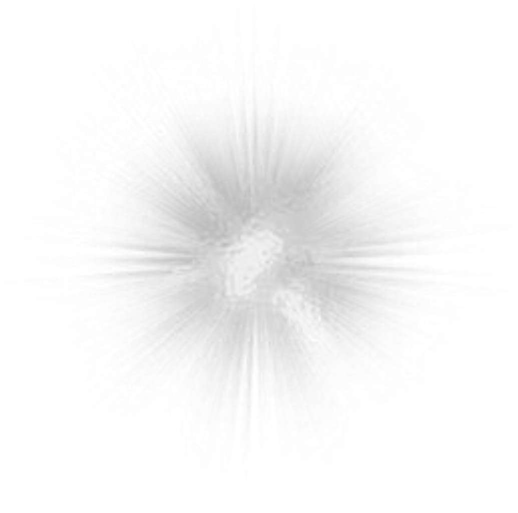
	Beam Gun, Low Detail 821 Polygons
	Beam Gun, Color
	Beam Gun, Emissivity Mask
	Beam Gun Bullet Hole

	Marksman Rifle, Full Detail 1,584 Polygons
	Marksman Rifle, Low Detail 534 Polygons
	Marksman Rifle Color
	Marksman Rifle Emissivity Mask
	Marksman Rifle Bullet Hole

	Pistol, Full Detail 644 Polygons
	Pistol, Low Detail 148 Polygons
	Pistol Color
	Pistol Bullet Hole

	Shotgun, Full Detail 1,800 Polygons
	Shotgun, Low Detail 780 Polygons
	Shotgun Color
	Shotgun Emissivity Mask

	Shotgun Bullet Hole
	Transform Effect Halo Cylinder
	Transform Effect Halo

	Transform Effect ‘Shiny’ This was a generic ‘shiny’ particle image we made in Photoshop and reused for many effects including the ship’s engines.

A.2 Sound Assets

Sound name	Where sound plays	Source
Dusty Bowl Sand Ambient Noise	Always plays and loops	(Boe 2006)
Ambient hum of the mining drill	Always plays and loops, but can only be heard near the mining drills	(blaukreuz 2012)
Assault Rifle Shot	When Assault Rifle shoots	(Epic Games 2015)
Beam Gun Shot	When Beam Gun shoots	(Bekir_VirtualDJ 2011)
Marksman Rifle Shot	When Marksman Rifle shoots	(Epic Games 2015)
Pistol Shot	When Pistol shoots	(Epic Games 2015)
Shotgun Shot	When Shotgun shoots	(Epic Games 2015)
Inserting a crystal	When the player inserts the crystal into the gun	(stewdio2003 2014)
Popping a crystal	When any crystal is ejected from the gun	(stewdio2003 2014)
Reload	When the player reloads a weapon	(Shotgun Cock Reload Layered (Powerful) 2013)
Footsteps	When players move	Recorded by Matt

Table 2 Sound Assets

A.3 Feedback Data

A.3.1 Playtesting Survey

1. How often do you play video games?

___ Extremely Often
___ Very Often
___ Often
___ Neutral
___ Not Often
___ Not Very Often
___ Not Extremely Often

___ N/A (Not applicable. I don’t play video games.)

2. Our game is a First Person Shooter. Are you familiar with the First Person Shooter genre?
___ Yes ___ No

3. Can you name any popular First Person Shooters off of the top of your head?
___ Yes ___ No

If yes, then please name a few: ____
____

4. Have you ever played a First Person Shooter?
___ Yes ___ No

If yes, then how often do you play First Person Shooters? If no, then select “N/A”. ___ Extremely Often ___ Very Often ___ Often ___ Neutral ___ Not Often ___ Not Very Often	If yes, then how much do you enjoy First Person Shooters? If no, then select “N/A”. ___ Extremely Enjoy ___ Very Enjoy ___ Enjoy ___ Neutral ___ Not Enjoy ___ Not Very Enjoy

___ Not Extremely Often ___ N/A (Not applicable. I answered “No”.)	___ Not Extremely Enjoy ___ N/A (Not applicable. I answered “No”.)

1. How would you describe the overall experience of *Crystalline*?

___ Extremely Enjoyable
___ Very Enjoyable
___ Enjoyable
___ Neutral
___ Not Enjoyable
___ Not Very Enjoyable
___ Not Extremely Enjoyable

2. How does “*Crystalline*” compare to your expectations of a First Person Shooter?

___ Extremely Exceeds Expectations
___ Exceeds Expectations
___ Meets Expectations
___ Neutral
___ Fails Expectations
___ Greatly Fails Expectations
___ Extremely Fails Expectations

___ N/A (Not applicable. I haven’t played another First Person Shooter.)

3. In online video games, **lag** is a noticeable delay between the action of players and the reaction of the server.

How would you describe the networked multiplayer experience and lag of *Crystalline*?

___ Extremely Lag Free
___ Very Lag Free
___ Lag Free

___ Neutral
___ Laggy
___ Very Laggy
___ Extremely Laggy

Optional comments about lag: ____
____
____

4. In video games, a **weapon sandbox** is described as a collection of weapons that a player can choose from, with each weapon fulfilling one or more distinctive playstyles.

How would you describe the weapon sandbox experience of *Crystalline*?

___ Extremely Enjoyable
___ Very Enjoyable
___ Enjoyable
___ Neutral
___ Not Enjoyable
___ Not Very Enjoyable
___ Not Extremely Enjoyable

Optional comments about the weapon sandbox: _________________________________
____
____

5. In video games, **visual experience** is a broad term, but here we are asking, “is the game pretty?” or other feelings like, “the character model looks nice, the particle effects are cool, the textures seemed very detailed”. All these things impact the visual experience of the game.

How would you describe the visual experience of *Crystalline*?

___ Extremely Enjoyable
___ Very Enjoyable
___ Enjoyable
___ Neutral

___ Not Enjoyable
___ Not Very Enjoyable
___ Not Extremely Enjoyable

Optional comments about the visual experience: _________________________________
____
____

6. In video games, a **HUD** stands for Heads Up Display. A term taken from the military to help describe all of the information available to a player during gameplay.

   How would you describe the in game HUD of *Crystalline*?

   ___ Extremely Comprehensible
   ___ Very Comprehensible
   ___ Comprehensible
   ___ Neutral
   ___ Not Comprehensible
   ___ Not Very Comprehensible
   ___ Not Extremely Comprehensible

   Optional comments about the HUD: ____
   ____
   ____

7. In video games, **level design** is a large topic on its own, but basically it boils down to "was the environment fun to play in?" And did you have fun in all the game types that you played on the map?

   How would you describe the level design of the map, Crystal Canyon, that you played in *Crystalline*?

   ___ Extremely Enjoyable
   ___ Very Enjoyable
   ___ Enjoyable
   ___ Neutral

___ Not Enjoyable
___ Not Very Enjoyable
___ Not Extremely Enjoyable

Optional comments about the level design: _______________________________________
____
____

8. In video games, a **tutorial** is the part of the game where the player learns about some of the main actions of the game.

We chose to place our tutorial inside of the spawn rooms on our map. How would you describe the tutorial puzzle of inserting a crystal, aiming, shooting the target, then leaving the room, ejecting the crystal, and finally exiting the respawn room in *Crystalline*?

___ Extremely Enjoyable ___ Very Enjoyable ___ Enjoyable ___ Neutral ___ Not Enjoyable ___ Not Very Enjoyable ___ Not Extremely Enjoyable	___ Extremely Easy ___ Very Easy ___ Easy ___ Neutral ___ Difficult ___ Very Difficult ___ Extremely Difficult

Optional comments about the tutorial: ____
____
____

1. In video games, navigating between menus, game states, and options are extremely important to a quality user experience.

How was your experience navigating through the menus and setting up a game in *Crystalline*?

___ Extremely Easy to Navigate ___ Very Easy to Navigate ___ Easy to Navigate ___ Neutral	___ Extremely Easy to Understand ___ Very Easy to Understand ___ Easy to Understand ___ Neutral

___ Difficult to Navigate ___ Very Difficult to Navigate ___ Extremely Difficult to Navigate	___ Difficult to Understand ___ Very Difficult to Understand ___ Extremely Difficult to Understand

Optional comments about the pre-game lobby and post-game lobby menu system:

____

____

____

1. Please feel free to express any other comments, concerns, and opinions that you have that were not covered in our survey. Thank you for playing *Crystalline*. We hope you enjoyed the experience, and are grateful for your feedback.

A.3.2 Internal Playtests

- 4/17/15 with Bayliss and Alberto with Raven Build
    - Put the word Graphics or something on the option so you know what you are changing. - Bayliss
    - Do graphics settings not carry over when level changes? - Bayliss
    - Saucy is a horrible name - Bayliss
    - You can get out of the tutorial room with a special weapon because of the colliders - Alberto
    - Shotgun still feels OP - Alberto
    - Beam gun looks cool but “dont feel like I’m doing anything different than with the pistol or something” - Alberto
        - Having a lock on hud image for the beam gun might help, that only the firing player sees
        - “It’s extremely hard for me to tell when I’m hitting something with the green weapon” - Bayliss
    - Delay on respawn was ‘annoying’ - Bayliss

    - Would expect a headshot with marksman rifle to be instant kill when the shotgun can instant kill. - Alberto
    - Orange uniform blends in to orange level textures which is hard on older people - Bayliss
- 4/17/15 Internal 2 Player Game with Raven Build
    - Crazy King works and is fun
    - Host gets the lobby screen but is also in the world so he can move around like a spectator when a game ends and he goes back to the pre game menu
    - Settings should default to High
    - Crazy King of the Hill text overlaps with Map text in pregame menu
- 4/15/15 with Bierre
    - Visuals look nicer
    - Would like to stop spawning in the tutorial room every time
- 4/15/15 with RPI Pre-Judging
    - Chad suggested we should have the “shoot color to clear obstacle” mechanic back since we bring it up in the tutorial.
- 4/13/15 with CrashTest
    - Bayliss says to have data “in a spreadsheet condensed into charts” with the most important being up in section 8 and the rest here. Don’t put every last individual survey here
- 4/9/15 with Bayliss
    - Main menu still broken (yesterdays build)
    - Kill/Death streaks would be nice
    - Ammo drop radius could be increased or visibility increased
    - Loading screen didn’t show
    - Frustrating not to know where you are when you spawn

    - Map looks the same everywhere
    - Guns look nice, wouldn’t mind them just the way they are
    - Put more specific questions on survey
        - Do you prefer certain areas of the map?
        - Do you prefer a certain weapon?
- 4/8/15 with Bierre
    - Still spawning in room with another player sometimes
    - 3D Sound still buggy? Test.
    - Game crashes from time to time possibly from using the shotgun?
    - Crystal pickup radius needs to be larger
    - Text about picking up/popping crystals needs to be more visible
    - Bierre kept hitting Q accidently because it’s the ‘weapon swap’ button in TF2
    - Make 1 and 2 switch weapons, Q maybe swap weapons, F for crystal pop.
- 3/19/15
    - Shot gun needs work – too hard or inconsistent to use
    - Reducing strafe speed improved the ease of using weapons
    - The pistol should fire slower, be auto fire rather than click as fast as you can
    - Marksman rifle felt good this time, took about 5 tries to kill a moving target
    - Beam rifle bug (shooting off into the distance while no one is using it) still there
- 3/18/15 With Bierre
    - Thinks the ammo representation works out pretty well as long as players understand it from the start
    - Some textures like the dirt one in the map can make motion sickness in people – bierre in particular
- 3/18/15
    - Strafing is too effective to get away from bullets

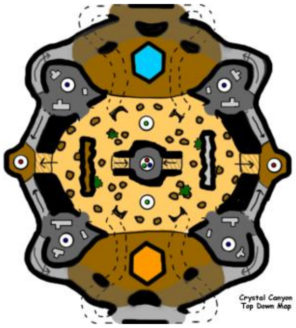
- Guns are still too ‘skillful’ to use, partly because of strafing, weapon shake, the bullets go further than the reticle indicates
    - New geometry changes are good for providing cover, changing how people move through space
    - Steam based network play breaks on second game, and games appear but cannot be connected to if you are in the wrong mode
    - King of the Hill zone needs more feedback to show which game you are in (I had no idea it was KotH until I realized I wasn’t getting points for killing)
    - Feedback about how close someone is to winning only appears for host
    - Should work on having alternatives for playing other than being logged into steam
- 3/11/15
    - Better gun Silhouette differences especially from first person perspective
    - Fix 3D Sound
    - Standing in drill
    - Beam gun bug (continues firing off into distance)
    - Still spawning in same room: suggest getting rid of room for after first time you spawn.
- 3/9/15
    - Menu stays up on clients unless you spam resume
    - If host quits everyone gets knocked out to menu
    - If you host and then cancel it goes blank instead of to menu
    - Beam gun way overpowered and has long range, marksman rifle underpowered
    - Spawning in the same room as someone else
- 11/19/14 Bierre’s Feedback (End of Fall Semester)
    - no capture drones because they could overwhelm

- Players pickup crystals per kill, get a streak: become higher value target. Have to return to outpost to beam up to ship.
- Explosive hide and seek
- Controllable turrets

A.4 Designed but not Implemented

A.4.1 Multiplayer Maps

Crystal Canyon



ARAM (All Random All Mid) Map

Built with the modular set, designed for conquest game type.

Modular Raid Map

Built via algorithm.

Arena Map

Designed for deathmatch game type.

Example Map

Designed for deathmatch using the modular set to showcase what you can do with the modular set and still make a good map.

A.4.2 Map Editor

1st: Component toy chest (a set of modular pieces). 2nd: also have a terrain plain that can be edited, pulled pushed and smoothed, then 3rd also have it so that you can pop in / pop out live edit system similar to farcry and halo forge.

A.4.3 Original Crystal Based Gun Sandbox

The Crystal Gun and Refined Crystals form the backbone of the weapon sandbox in *Crystalline*. Each refined crystal has a theme associated with it.

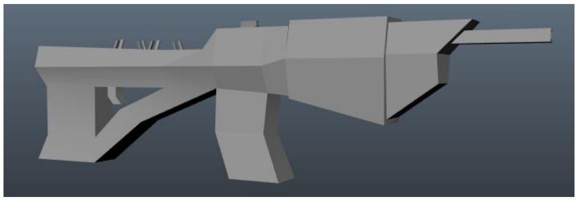
RED has a theme of power and damage.

PURPLE has a theme of precision and range.

GREEN has a theme of energy and speed.

These crystals can be inserted into the crystal gun's upgrade slots in various combinations. The primary/secondary slot do not affect gun type. They only affect active ability slot. For example: This gives the player the choice of having the marksman rifle with either the incendiary ammo active ability or the cryo ammo active ability.

Assault Rifle

	Primary Slot: None Secondary Slot: None Active Ability: None

The *Assault Rifle* is a mid-range weapon that's the king of the general case. A player using the assault rifle should outperform the more advanced weapon types if the other player is not using the specialized weapon in the correct case. For example if the *Assault Rifle* faces off with a *Shotgun* at mid to far range, the *Shotgun* should most of the time lose. However, if the *Assault Rifle* faces a *Shotgun* at close range, the *Assault Rifle* will almost always lose (always if the *Shotgun* user is competent).

Rough Time To Kill:

Long: ~2 Clips

Mid : ~1 Clip

Short: ~1 Clip

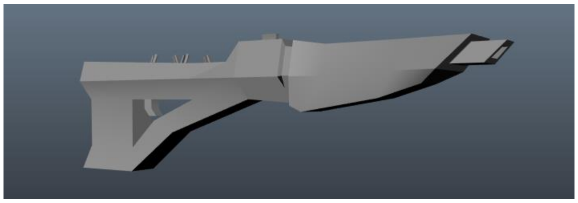
Fire Speed : Fast

Damage per Shot : Low-Medium

Range : Mid

Spread : Tight, progressively looser as the player continues shooting.

Shotgun

	Primary Slot: Red Crystal Secondary Slot: None Active Ability: Incendiary Ammo

The *Shotgun* is an up close and personal weapon. If the player specs into a Power Crystal their goal is to utterly annihilate whatever they come up against. However, the player sacrifices some of their ability to be versatile. The *Shotgun* is worthless at long range and the player must make sure their shots really count in the close range as the gun is slow to shoot in quick succession.

Rough Time To Kill:

Long: Unlikely

Mid : 4 Shots

Short: 2 Shot

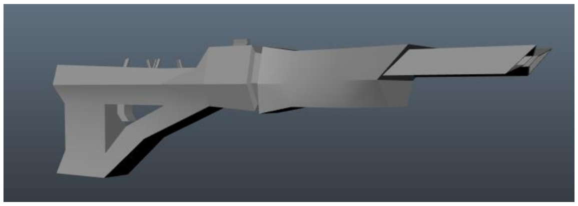
Fire Speed : Slow

Damage per Shot : High

Range : Short

Spread : Loose

Double Barrel Shotgun

	Primary Slot: Red Crystal Secondary Slot: Red Crystal Active Ability: Incendiary Ammo Special Ability: ???

A perennial favorite the *Double Barrelled Shotgun* is a force of nature. A single shot fires more “buck shot” than its close brethren the *Shotgun*. The *Double Barrelled Shotgun* should feel like an absolute monster to fire, with the kickback exceeding every other gun in the game.

Rough Time To Kill:

Long: Unlikely

Mid : 2 Shots

Short: 1 Shot

Fire Speed : Slow

Damage per Shot : Extremely High

Range : Short

Spread : Loose

Sniper Rifle

	Primary Slot: Purple Crystal Secondary Slot: None Active Ability: Cryo Ammo

The *Sniper Rifle* is the weapon of choice for players who like to assess the situation. From about halfway across the map a competent player with the *Sniper Rifle* should be able to place some serious heat on their opponents. A shot from the sniper rifle should be considered lethal, but at close range due to the precision of the instrument it is expected that most players will not always hit their mark.

Rough Time To Kill:

Long: 1 Shot

Mid : 1 Shot

Close: 4 Shots

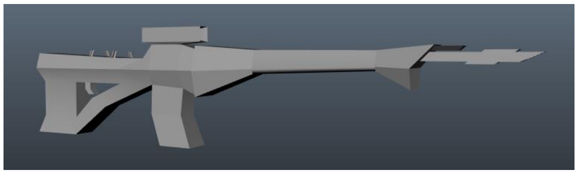
Fire Speed : Medium

Damage per Shot : High

Range : Long

Spread : Extremely tight

Railgun

	Primary Slot: Purple Crystal Secondary Slot: Purple Crystal Active Ability: Cryo Ammo Special Ability: ???

The epitome of range and accuracy the *Railgun* is a ranged monster. It is effectively a *Sniper Rifle* with enhanced range and the ability to shoot through multiple opponents. As a trade off the *Railgun* has low ammo capacity and a slow reload speed.

Rough Time To Kill:

Long: 1 Shot

Mid : 1 Shot

Close: 2 Shots

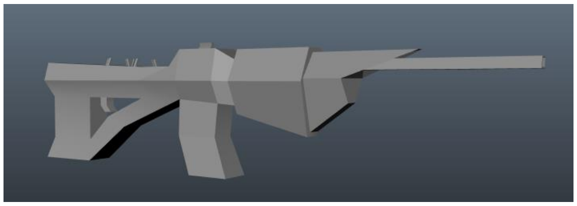
Fire Speed : Slow

Damage per Shot : Extremely High

Range : Extremely Long

Spread : Extremely tight

Marksman Rifle

	Primary Slot: Purple Crystal Secondary Slot: Red Crystal Active Ability: Cryo Ammo

The *Marksman Rifle* is a single shot mid-range weapon. Its core purpose is for tactical players who want the accuracy of a *Sniper Rifle* in the mid to close range with a punch. It's not as strong as either the *Sniper Rifle* or *Shotgun*, but it make up for it in versatility and low kick for easier aiming.

Rough Time To Kill:

Long: 6 Shots

Mid : 3 Shots

Close: 3 Shots

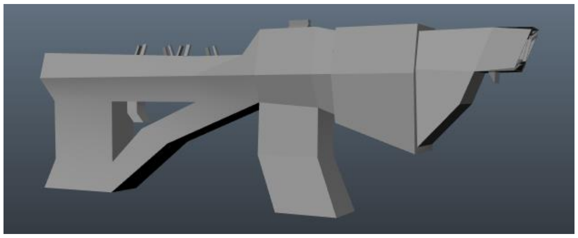
Fire Speed : Medium/Fast

Damage per Shot :Medium

Range : Close-Mid

Spread : Extremely Low

Burst Rifle

	Primary Slot: Purple Crystal Secondary Slot: Green Crystal Active Ability: Cryo Ammo

Like the *Marksman Rifle* the *Burst Rifle* a tactical mid-range firearm. Unlike the *Marksman Rifle* the *Burst Rifle* trades punch for the ability to fire three shots in rapid succession. The *Burst Rifle* is generally to be used by players who aim less than their Marksman counterparts.

Rough Time To Kill:

Long: ~6 shots

Mid : ~3 shots

Close: ~3 shots

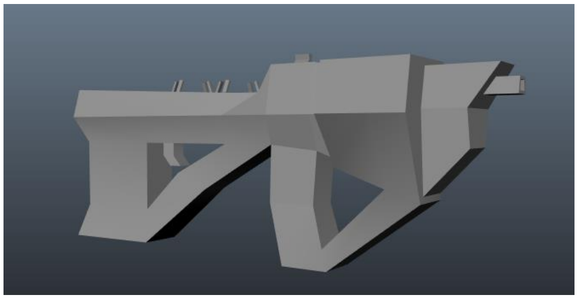
Fire Speed : Mid (3 bullets per shot)

Damage per Shot : High (burst of three)

Range : Mid

Spread : Tight

Submachine Gun

	Primary Slot: Green Crystal Secondary Slot: None Active Ability: EMP Ammo

The *Submachine Gun* is a close cousin to the *Assault Rifle* with the key distinction in that the *SMG* is effectively an over glorified bullet hose. The key of the SMG is that it gets the bullets from the magazine of the weapon to the enemy as quickly as humanly possible. The *SMG* should do minimal damage per bullet, and its viability falls off in mid-range. The reload speed should be relatively fast to accentuate the speed of the gun and encourage the player empty the magazine.

Rough Time To Kill:

Long: Unlikely

Mid : ~4 Clips

Close: ~2 Clips

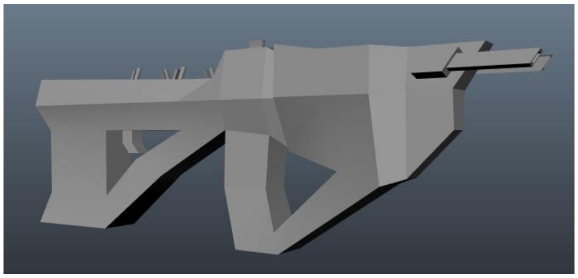
Fire Speed : Very Fast

Damage per Shot : Low

Range : Close to Mid

Spread : Loose to very Loose

Double Barreled Submachine Gun

	Primary Slot: Green Crystal Secondary Slot: Green Crystal Active Ability: EMP Ammo Special Ability: ???

The *Double Barreled Submachine Gun* epitomizes spray and pray. It doubles the bullet hose that is the *SMG* and increases the speed twofold. This gun is the most unusual of the sandbox, but does not disagree with the general aesthetic of other weapons in the sandbox.

Rough Time To Kill:

Long: Unlikely

Mid : ~2 Clips

Close: ~1 Clips

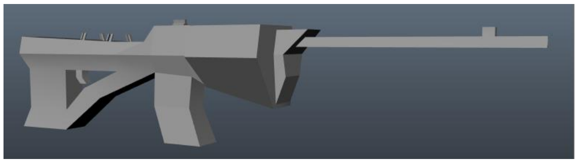
Fire Speed : Extremely Fast

Damage per Shot : Low

Range : Close to Mid

Spread : Loose to very Loose

Heavy Machine Gun

	Primary Slot: Green Crystal Secondary Slot: Red Crystal Active Ability: EMP Ammo

The *Heavy Machine Gun* is a *SMG* with a punch. The HMG is designed for the player who likes the punchiness of the shotgun, but is willing to trade the high risk sure thing for a lower risk higher fire speed alternative that doesn't always hit its mark.

Rough Time To Kill:

Long: 40 shots

Mid : 25 shots

Close: 20 shots

Fire Speed : Fast

Damage per Shot : High

Range : Mid

Spread : Tight first shot, exponentially progressively looser as the player continues shooting.

A.4.4 Gameplay:

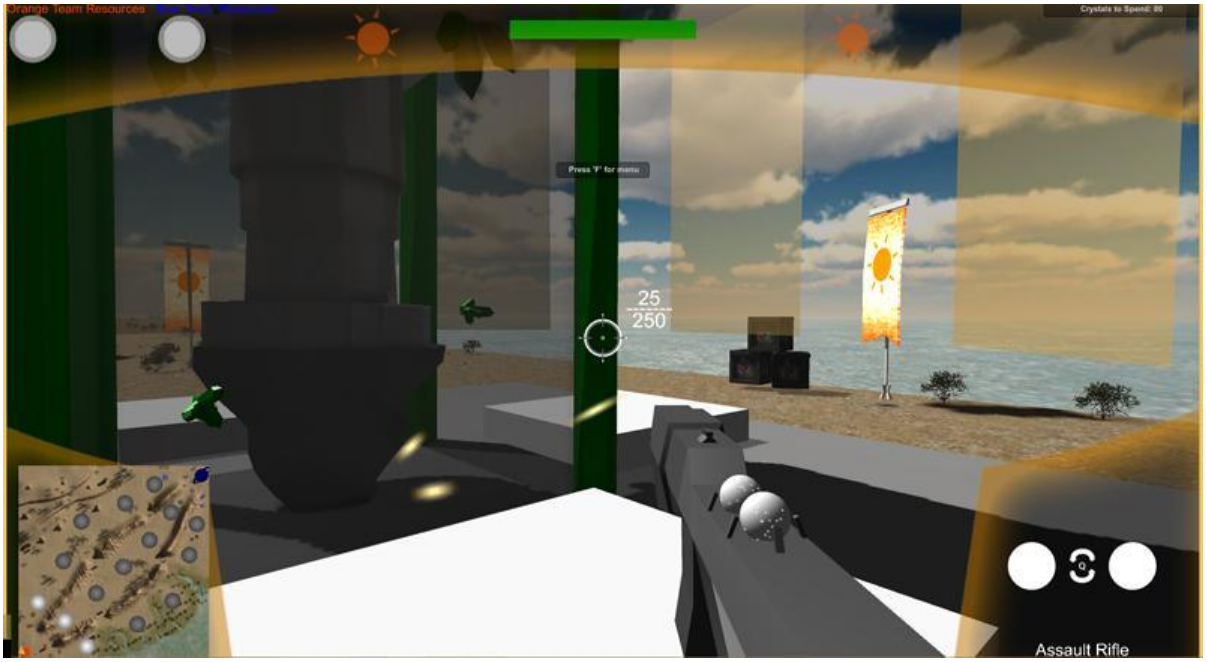
The gameplay of *Crystalline* can be analyzed into three identifiable components of shooting, capturing, and upgrading. These components all focus around the primary gameplay pillar of pushing the line. Together they create an experience that is more than just a sum of the parts.

A standard component of any first person shooter is the twitch action shooting mechanic. Players will shoot, move, throw a grenade, dodge bullets, melee, move, and then repeat this process over and over again. This 30 seconds of fun, as coined by Jamie Griesemer of Bungie, can be stretched out to become an entire game. This core loop is essential in any first person shooter.

As the game progresses, players will need to capture outposts to succeed. These outposts harvest the crystals that the player factions are fighting over in the game narrative. Players are challenged with capturing and protecting these outposts while fighting enemy troops. By constantly holding these outposts, players are rewarded with crystals that they can use to upgrade both their weapon and the outpost.

A key feature of *Crystalline* that feeds into many gameplay areas is the crystal based upgrade system. This system improves captured outposts and is the backbone behind our game's weapon system. As a reward, over time holding outposts adds unrefined crystals to a team's bank. Players can used these unrefined crystals in the bank to purchase upgrades for controlled outposts. One example of such an upgrade is an auto defense turret. As the game goes on, refined crystals form on the side of the outpost drills. Players can grab these refined crystals and insert them into their primary weapon, the crystal gun. The crystal gun, starts off as a generic all-purpose assault rifle, but when a player inserts a refined crystal into the rifle, its weapon properties are altered and it becomes a new type of rifle. Via the three refined crystals and the combinations of these crystals, we create an entire weapon sandbox for *Crystalline*.

A.4.5 Drilling Outposts



The Outposts should be considered the core elements of gameplay for the purpose of level design, as they are the direct representation of the moving “line” of combat. Outposts represent the player presence on the map at a given time in a physical manner, representing the state of play for the team as a whole (**more outposts == more power**). To win the game a team **MUST** capture outposts in an effort to capture the enemy faction’s base or gain resources.

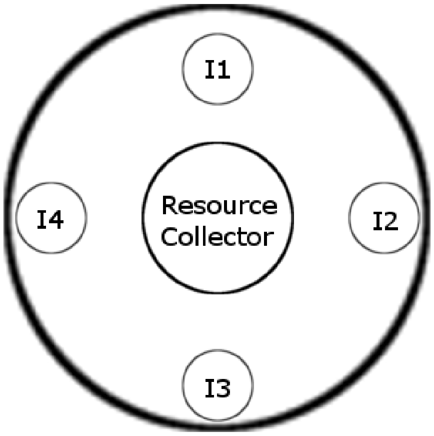
Each outpost is to have at least one type of resource which is “mined” by the outpost. The outpost additionally has a set of “buckets” (exact number determined by the number of resources it gathers) unlocked as one team has ownership over an outpost for a predetermined period of time/resource gain. Power ups such as turrets, shields or item factories may be placed in these buckets at the expense of resources (typically those mined by the outpost).

Outpost Layout/Contents

The outpost itself can be imagined as a fortifiable drilling platform with the following qualities:

- Each outpost will have no less than one “drill” tower.

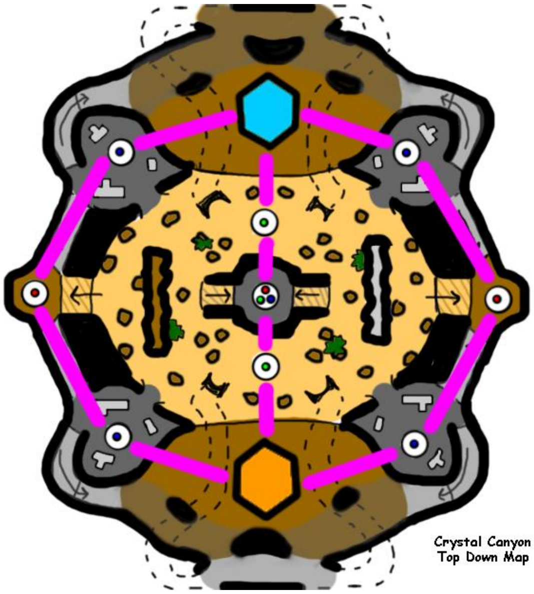
- This drill tower will harvest one resource.
- An outpost will have several **improvement** slots which a player may build upgrades upon.



Outpost Lanes

The Outposts in ***Crystalline*** may be thought of as lanes of attack.

- An outpost must be connected to at least one other outpost on the map.
- Outpost towers must have line of sight to any connected outpost towers.
- Team bases qualify as outpost towers for the sake of the above rules.



Difference between Bases and Outposts:

- Capturing an enemy base will result in victory
- Bases produce more resources from the start than a standard outpost
- Bases come with some upgrades, such as catapult to fire the players the the closest outposts
- Bases are the standard spawn-points for the user.
- Bases are bigger and take longer to capture

Outpost Capturing

The act of capturing requires a team to gain capture points for an outpost. The number of capture points required varies based upon the number of resources that the outpost contains, starting at 10 for one resource and 20 for 3 or more. An outpost may be captured in two ways: physically ("Player Capture") and via "Energy Beam".

Type	Base Capture Points per Second
Player	1
Energy Beam	0.5
AI Energy Beam	2
Capture Points per Second Calculation	$p < 5$:CPpS*(-.125p2+1.125p) p 5:CPpS * 2.5
Outpost Capture Points	$r \leq 3$:-1.5r2+11.1r r>3 :20

Player Capture occurs when a player character is physically at an outpost location. The act of capturing is mechanically similar to capturing the hill in Halo's King of the Hill gametype. When a player enters a zone connected their outpost network they begin to initiate a capture. Each Allied player in the zone of capture increases the capture points per second by the factor defined in the **Capture Points per Second Calculation** above. Enemy players may enter the zone of capture, even if the node is not connected to their network, to contest zone captures. A player may NOT capture a node unconnected to their network. When the capture points reach the required total ownership of the outpost falls to the team who successfully reached the total.

A.4.6 Crystal Resources

- The only resource currency in our game is the crystals that can be harvested at outposts.
- These crystals are the narrative point of contention between the two main teams.
    - These crystals are essential for space age living in our universe.
    - The power ships and are used for upgrading tech, etc, etc.
    - They enable advanced life support systems required for space living and flight.

Winning the Game

1. Capturing the enemy’s base (Military Victory)
2. Acquiring the “goal” amount of resources (Economic Victory)

Capturing the enemy’s base (Military Victory)

- Capturing resources nodes leading up to the enemy base, then capturing the enemy base itself
- When the enemy’s base has been captured, the game is over, and victory goes to the team who captured the enemy’s base.
- Favors aggressive play through rapid node capturing

Acquiring the “goal” amount of resources (Economic Victory)

- A “goal” amount of resources is set before the beginning of the game
- Essentially exists so that if there is no military victory after a certain amount of time, a victor is decided by the amount of overall resources gathered.

- The way this is explained in game is that for every amount of resources gathered, a percentage of it is beamed to the mothership for that team, and the rest is kept to further your cause as spendable resources. When the “goal” amount of resources is reached by what has been beamed to your mothership, then you can leave the planet, as when the “goal” is reached, this represents the planet having been completely stripped of resources, and leaving far less for the other faction.
- Favors more conservative play, and you can just try your hardest to defend 51% of the nodes, and then win eventually.

A.5 Colony Crisis Design Doc

Passion

John: 3D environment on terrain provides a lot of opportunity to play with terrain rendering and generation techniques. With no building placement in game, we don't have to worry about adding that feature to terrain. The existing Terrain with overhangs and tunnels could be used to hide more important features or upgrade platforms or routes to the enemy base. Deformable terrain could take advantage of all the big guns we have going on if you wanted to go that route, or the interior of the base could become more important through stealing resources or hacking or planting bombs there and it could be generated procedurally.

Alex: The idea is able to support anything from a single player game to as many players as the network can handle. Having a single player option also allows us to test it more easily, and takes pressure off of me since it will still be playable if something goes wrong with networking. There is a lot of potential for this game in terms of networking architectures, and I plan to experiment with a few different approaches and models before settling on one that works best for keeping the game and the objects inside it in sync. If networking doesn't go well however, I can shift focus to working on the AI for the single-player version of the game.

Matt: There are numerous places where graphical effects can be implemented, starting with particle systems on the projectiles crossing the screen. I keep thinking of this scene from Stargate Atlantis where the good guys are under attack and are going to die but the bombardment of their shield looks so pretty that they stop and stare at it for a while like its romantic. I imaging our game having this constant barrage of projectiles going across the sky where players are safe to watch it, as well as cool effects for shields, resources that can be reflective and shiny, etc.

Nick: Because the game is based on the costs and effects of upgrades, and all the upgrades can just be static geometry, it offers a lot of options for modders to change the costs, effects or upload their own geometry and textures to be used as upgrades. A game configuration tool could be used to easily increase the length of a game by adjusting upgrade costs and team health or the style of game by constraining people to certain upgrades (like match types where only war upgrades or only peace upgrades are allowed). And even a map editor tool would be feasible since the only requirement for the map to work is that *if* military victory is enabled both team's mother ships have to be visible to at least *one* turret platform, which could easily be checked for programmatically.

Scope

This idea was designed so that the minimum playable game would be a complete experience but small in scope. Other features can then be built on top as desired.

Layer 1 - Minimum Viable Product (MVP):

1. 3D Environment with static base and mother ship for each team.
2. Ability to walk over the terrain.
3. Platforms for building upgrades.
4. GUI for presenting current credits, base health, team DPS (damage), CPS (credits), and IPS (influence). GUI for presenting a menu to buy upgrades at static upgrade points on the base, as well as at computers near the base for upgrades like a shield around the base or team-wide bonuses. GUI based tutorial/instructions on what to do when you start, and a win/loss screen. Menu for buying upgrades should only appear when near the appropriate platform/computer when you press a button.
5. A balanced cost-curve for purchasing upgrades which get more expensive per level/upgrade.
6. An AI routine for deciding what upgrades to buy for the enemy base. (This can just be random at a minimum, or based on cheapest upgrade available. If resource bonuses are powerful, the AI may need an advantage to be challenging in single player as it cannot gather resources for itself in the MVP).

7. Assets: For multiplayer, a player character model with run, jump, shoot, and idle animations would be required. Everything else can be represented as static geometry.
8. Audio for bullets, explosions, and background music/ambient noise.

This minimum product would basically be an Idle game, which we know can be addictive. The next layer of features would add FPS and strategy elements:

Layer 2:

1. Resources to be harvested.
2. Aerial drones that gather resources. These could also either shoot the player or drop deployable turrets like the player. (My intention is not to make ‘Grunts’ or ‘Bots.’ Aerial drones do not need pathing or navmeshes or dynamically changing goals and planning; they can just go up, seek to a resource, base or player, and then drop down to gather the resources or shoot the player. If the resource is one time and taken by a player or if a player its after dies, it can just get a new random resource or player to attack. The worst part would be some sort of flocking behavior to make sure they don’t try to crowd on top of each other *if* we have a lot of them and they are allowed to go to the same spot.)
3. Deployable turret/flag. The turret will just hover in one spot and use the already existing Transform.LookAt() to aim and shoot at players or drones. It doesn’t need complex building graphics or anything; I imaging it like in Star Wars Jedi Knight when you just hit a button and a little seeker ball appears and hovers around you. The flag will increase your IPS based on how far it is from your base, like you are literally claiming more and more territory.
4. Resource-Upgrade system for turrets. Putting a red crystal in a turret makes it stronger, a green crystal makes it give more credits, etc.
5. Ability to start and stop turrets to synchronize their audio into a chorus of firepower (whether they are guns or fireworks)

Layer 3:

1. Resource-Upgrade system extended to the player's weapon, drones, ship, and base. So a red crystal in a drone makes it fly faster or something.
2. Killing someone with a resource in hand or in their weapon drops them to be looted.
3. Team upgrades – Boosts for the whole team, like a shield that goes up around the base and ship, or effects such as "Sanctions – When the enemy kills someone on your team, they lose Influence" which would be available as an expensive peaceful-victory upgrade.
4. Ethics based upgrades/victory conditions.
    a. I don't know what to call it exactly or where to put it in these layers but the idea is this: Say team 1 gets an upgrade "Bio Weapons" which makes them do more damage but can kill the local population of the planet. The other team might also be militaristic but they don't want to use Bio Weapons, so because they are fighting someone who is, they become the 'Just' military and the others become the 'Domination' Military or something like that. BUT to take it a step farther maybe the team using Bio Weapons is weaker and the other team is using something like "Drone Strikes" maybe the first team is actually a "Freedom Fighter" Military and the second is an "Overwhelming" Military. However we name them, the point is to address real life issues and to make people argue over which team is the 'good' team. Ideally, each would have a different win/loss cutscene (but those are pony list). Influence based victories could be like 'Peaceful,' 'Cultural', 'Propaganda', and 'Economic' for instance. Or economic can be another whole set.
5. Shield – (If it were separate from a temporary team upgrade) another piece of the base that can be upgraded in terms of power, regeneration, etc with resources. Maybe it requires a certain upgrade to be able to pass through it. If it's boosted with a red crystal it takes more damage from turrets with yellow crystals for example.
6. Hacking or sabotaging the enemy base, stealing resources from it, etc.

7. Upgrade points in the environment rather than at the base that either team can build on. These can have limits to what can be built on them, like if it can’t get Line of Sight on either base only teleporters, resupply points, or things like that can be built there.

Layer 4:

1. Base interior (procedural?)
2. Scripted events like one time resources that provide a temporary global effect like all turret platforms go haywire and fire at random targets.
3. If we have terrain with overhangs or tunnels rarer resources that provide special effects could be hidden in those sorts of places.
4. Audio for player voice commands like “Run!” or “Help!”, audio for the local population and base people to randomly say stuff and make the game feel more alive.

Layer 5 - Super Pony List:

1. Additional levels like one where both teams are working together to DPS a third thing like an alien ship or something.
2. Procedural terrain - only constraint is that *some* turret platforms can see one of the motherships.
3. Grunts
4. Changeable player equipment
5. Vehicles
6. Cutscenes for the different victory conditions
7. etc. etc. etc.

Networking

Things that would have to be passed through the network include:

1. Player positions

2. Player bullets
3. Team DPS, IPS, and CPS when it changes. If possible, sending over the individual bullet information would be more accurate, but DPS is all that is needed at a minimum.
4. Current team health every second or when it changes.
5. When someone buys an upgrade
    a. My thought for this is each player on a team would get a percent of the team's income. They can then put that money towards an upgrade, and other players would see that when they check the upgrade menu. Then they can finish the purchase with their own money. If a player puts 250 credits towards a gun turret on a platform and another player has more money and completely buys a firework turret (builds influence) on that platform the first player gets his money back. This approach would promote teamwork without requiring a chat interface.
6. Chat. See above (5-a) but chat wouldn't be required to be playable.

A.6 Modular Asset Set Guidelines

If you are using a modular set, then fail earlier and often. Start working yesterday. Because yesterday is now, and tomorrow is never.

You should first brainstorm about your environment. Identify what the general aesthetic is. Get a color pallet agreed upon. Ask yourself questions like, "What happened yesterday in this environment? What happened years ago? What is the weather like here? What is the plant life?" Asking questions like these will help you solidify and guide your initial concept.

Next you should start with concepts and napkin sketches, basically rough drawings. From these sketches you can create some more detailed concept images and landscapes. In your next meeting you can use these landscapes as references to help identify common modular set pieces. For example, if your environment is a desert rock canyon like ours, then you might start to notice that there are a lot of large outdoor rocks in your landscapes, which might be a hint to include a few large rocks into your modular set. If your environment is in an urban city space, and you notice your landscape has a lot of city buildings, then part of your modular set may include a variety of building components such as windows, or building facades.