CTF_Shift: A Custom Team Fortress 2 Map

Taylor Yust, Scott Benton, Luke Godfrey, Dylan Reile

Note: Except where otherwise specified, text was written by Taylor Yust

Abstract

One of the most prominent applications of virtual worlds is in the game industry. The success of a game hinges on its ability to craft compelling environments that effectively explore the design space afforded by its mechanics.

Team Fortress 2, an online multiplayer action game developed by Valve, is no exception. In an attempt to further our own experience with game development and virtual world construction, our team seeks to design and build an engaging custom map using critical game design methodology. We will tackle the “Capture the Flag” game type, which is notorious in the Team Fortress 2 community for often resulting in broken gameplay. [1]

1.0 Problem

When most amateur designers approach map development, they fail to think of the design in a critical manner and instead move forward with only a vague notion of what would be “cool.” For example, rather than understanding a vertical space as providing a sort of increased “flow” in the form tactical and mechanical advantage, many designers would see it as an “awesome tower,” unable to look beyond its literal representation.

This approach often leads to poorly designed maps with balancing issues. The “Capture the Flag” (CTF) maps in Team Fortress 2 are an excellent example of this. The mechanics often lead to stalemates with the dominant strategy being to “turtle” near your team’s flag. This has led to a lack of popularity in competitive CTF play, which is a concern for fans of the game. [2] [3]

In a more general sense, the lack of critical thinking skills has been damaging to the game industry’s ability to construct compelling game designs. It also hurts our ability to properly “gamify” serious subjects, such as health and education.
2.0 Objective

The objective of this project is to develop a deep and compelling custom CTF map for Team Fortress 2 using critical game design methodology.

3.0 Background

3.1 Key Concepts

Team Fortress 2 – A popular team-based multiplayer online action game developed by Valve. Players work in teams of 12-16 using a combination of nine different classes ranging from Soldier to Medic to Engineer. They are challenged to accomplish overarching objectives such as capturing the enemy team’s intelligence, capturing a control point, or pushing a cart to a target destination.

Capture the Flag – A game type wherein each team is attempting to capture the opponent’s “flag” (represented as “intelligence” in the “50s spy action” theme of the TF2 universe). The flag is normally located at a team’s base, so a player and his allies must overcome an enemy team’s defenses to grab their flag and return to their base all while preventing the enemy from doing the same to their own flag.

If an enemy carrier of your team’s flag is defeated, the flag will be dropped where the carrier last stood. A countdown timer will start, and once it finishes the flag will “respawn” back at its team’s base. However, if the enemy can grab the flag, even temporarily, the timer resets, so there is a large incentive for a team to heavily defend a dropped flag until it respawns.

Source – The game engine that Team Fortress 2 was built with.

Hammer – The tool editor that Valve provides to create custom Team Fortress 2 maps within Source.

Hammer, the map editor for Source
3.2 Related Work

There has been a distinct lack of work done in relation to critical game and map design, certainly in the realm of Team Fortress 2.

4.0 Design

4.1 Design Goals

1. The primary design goal is to develop a working custom CTF Team Fortress 2 map.
2. Beyond just a barebones map, we would also like to create a pleasing aesthetic and ensure that the map is optimized for network play.
3. A long-term goal is to publish the map on map hosting websites and see the map played on public servers.
4. Finally, beyond simply being played, we would like the map to effectively address the issues of the CTF game type and provide deep and interesting gameplay (which is ultimately subjective, but generally understood through player feedback consensus).

4.2 High Level Architecture

In designing this map, we first attempted to understand critically why CTF maps fail so often. We came to the conclusion that most maps incentivize “turtling,” which is the practice of heavily defending a strategic location or object. It is restrictively difficult to tackle an enemy’s defense to steal their flag, and failed attempts often leave the assaulting team’s defenses open for a counter-steal. As a result, neither team feels it is to their advantage to play aggressively.

Furthermore, the value of CTF as a game type is this abstract idea of an “ever-moving point of interest,” in that the interesting gameplay comes from chasing a enemy that has stolen a flag and quickly and dynamically adjusting both offensive and defensive strategies relative to the locations of dropped flags until they respawn. In this way, gameplay is constantly shifting around the map. However, due to the overly defensive nature of most CTF maps, if an enemy base is miraculously routed, grabbing the flag is essentially an instant point gain since the path back must have already been mostly cleared to reach the flag in the first place.

Our solution is to create a map design that favors offensive strategies and makes grabbing an enemy team’s flag relatively easy. However, instead of making the return journey painless, we wanted a design that provided more challenge on the way back to base, creating the dynamic gameplay of “ever-moving point of interest” as described above.

We accomplished this by taking advantage of “gravity.” Gravity is an assumed mechanic in most games since “it’s realistic,” but critically understanding gravity as an explicit game mechanic allows a designer to learn how to best leverage its potential. By understanding gravity as an abstract “flow” that pushes and emphasizes spaces at the end of its vector, we can apply the mechanic to our goal of creating a space that is “easy to get in, but hard to get out.”
Thus, we’ve placed each team’s flag at the base of parallel vertical “shafts.” Gravity naturally pulls players to the bottom of a shaft to easily steal the opposing team’s flag, but afterwards gravity suddenly acts as a force of opposition that makes escape difficult for flag carriers.

With our general game design approach established, we developed a more detailed map, the general flow and layout of which can be seen above in a sketch. To represent this map as a location that “makes sense” and fits the TF2 aesthetic, we placed the map in a “flooded mineshaft” environment.

4.2.1 Initial Development and Division of Labor

Taylor acted as team lead and manager for the project. Since Scott, Luke, and Dylan had little experience with Team Fortress 2 and map development, Taylor established an explicit map design by applying the ideas above.

Actual development required the use of Hammer, Valve’s proprietary tool to develop custom maps for their games. Downloading Hammer required having an account with Steam, Valve’s software distribution service. After installing Steam, it was a simple matter of downloading the Team Fortress 2 client (which, as a “Free-To-Play” game, cost us nothing) as well as the Source
SDK, a collection of tools used in the development of software that utilizes Valve’s Source engine. Hammer is one of the tools included in the Source SDK.

Using Hammer, Taylor developed a map composed of “primitives” (such as “cubes” and “wedges”) based on the initial map design and sketches. The map would compile and run, albeit without games rules in place (doors would not open, there were no team flags, etc.), and there were no hand-placed textures or lighting; every object had generic “dev” textures, and all surfaces were displayed with default lighting values. However, it was playable.

One major advantage of the design is that one side of the map “mirrors” the other, making it symmetrical. This is ideal for competitive play to ensure equal and fair playing conditions, but it also essentially halves the effort for the development team, which is important given our lack of experience.

Taylor then divided the map up into three explicit sections that could be developed independently. Every primitive was tied to one of three “visgroups”, which are collections of game objects that can be all turned “off” or “on” from the editor. Each visgroup carried the name of its assigned team member, so any question of responsibility could be answered through the editor itself: If a team member turned off every visgroup but their own, whatever was left was theirs to develop.

We took advantage of Dropbox as a means of sharing files and assets. Taylor set up a folder for each team member as well as a backup folder should anyone need a fresh copy of the latest version of the map. The map file was duplicated several times and a copy was placed in each team member’s assigned folder. From there, each team member could develop their assigned map section independently from the others without worrying about file or design conflicts. On reaching certain milestones, Taylor would take each team member’s copy and merge their changes into a new map file that would then be duplicated and distributed back to the team members. This allowed team members to see and build off one another’s work without interfering with them.

Over the course of development, Taylor also added other map-wide elements. For example, Taylor handled setting up skyboxes (providing the illusion that there is an open-ended sky), sunlight, shadows, LOD (level of detail) control, water rendering, and cubemaps (which enable shiny surfaces to “reflect” their surroundings).

4.2.2 Map Overview

As mentioned before, the layout of the map is designed to resemble a pair of flooded mineshafts. Each team has a mineshaft with their flag at the bottom of the shaft. The team “spawn room” is
at the top of each mineshaft and is connected to a “team base” structure embedded into a wall of the mineshaft. While the mineshaft is fairly open and exposed, the base acts as a sort of “indoor” structure that provides cover opportunities but lies in close proximity to its affiliated team, giving that team more influence over the space. Both the mineshaft and the base provide routes for a flag carrier to climb his way out after having stolen the flag.

Right in the center of the map between the tops of the two team mineshafts is a common battleground area. It has some light structures and terrain to encourage team clashes, which should be a regular occurrence given that all flag carriers must pass through it on the way to their respective bases. This space is connected to two side passages that wind their way to the sides of each team mineshaft, providing a means of flanking the enemy team. They’re tactically designed to dislodge certain defenses from within each team mineshaft so that frontal attacks can be more successful.

Because of the symmetrical design, team members were assigned to develop just one side of the map and would later have their work reflected onto the other side, at which point subtle team-specific changes could be made. Each team member was also assigned certain mechanics and functionality that they had to implement.

The assignments were as follows:

- Scott Benton – Team Respawn and Base, Spawn Room Functionality
- Luke Godfrey – Team Mineshaft, Door and Filter Functionality
- Dylan Reile – Central Battleground and Side Passage, CTF Mechanics and Rules

4.2.3 Team Respawn and Base, Spawn Room Functionality (This section is written by Scott Benton.)

Starting with the initial map composed of primitives that Taylor had designed, Scott's first task was to implement the mechanics for the team respawn rooms. Since Taylor had already added in the player spawn entities with the initial map, the first step was to add in the functionality that would allow players to change classes without dying while inside of their spawn area. This turned out to be a simple task, as he only needed to fill the spawn room with a brush-based entity called 'func_respawnroom' and adjust the properties so that it applied to the appropriate team color. Since the respawn room for each side was comprised of two connected rooms, he needed to create the entity in both rooms and the section connecting them and then group them with the same settings to allow the functionality in all areas of the team respawn room.

Having completed this, he tested the sections to ensure that a player would be able to immediately change their class without their character dying or suffering a time penalty before they could rejoin the gameplay. Having little prior experience with Team Fortress 2 or game design experience, Scott considered the functionality of this feature as he was working on it. The importance of the mechanic became especially clear during his play testing as he attempted to run around the map with each of the individual classes, as he found the time delay increasingly frustrating when simply trying to switch classes. In relation to the idea of "ever-moving point of
interest" with this map, the need to dynamically adjust the class choice to fit a constantly changing strategy during gameplay was well established.

The next feature that needed to be added to the respawn rooms was the addition of resupply cabinets that would restore full health and ammunition to all players on the team that interact with them. Since the team respawn area was comprised of two connected rooms with a total of three exits, Scott decided to place one resupply cabinet in each room in places that were easy to access as a player was entering or leaving the area and yet still relatively safe from enemies who could shoot into the area when the doors were open.

A resupply cabinet consists of two entities: a 'prop_dynamic' entity which represents the resupply locker in the map, and the 'func_regenerate' entity which gives functionality to the resupply locker model. After creating and placing the two 'prop_dynamic' entities in each base and providing them names, Scott then created the 'func_regenreate' brush-based entities in a larger area around the props and adjusted the properties so that each 'func_regenreate' entity was associated with the name of a 'prop_dynamic' entity, and setting them to their respective teams. During this step, he was very careful to make sure that the entities matched up with the walls of the rooms carefully and that they did not leak into areas that would allow players to resupply where they shouldn't.

Having finished adding the respawn room and resupply mechanics, the next milestone was to add in some lighting elements to the respawn areas and the team base. Initially, Scott had difficulty with understanding how to properly use the 'light_spot' entities in the map. There were several settings and properties that were not entirely intuitive to beginners, and angle values took guesswork to figure out which angles would cause the entities to point in the desired direction. Brightness level of the lighting was also something of confusion, as the base setting of 200 was hardly enough to light the immediate sections around the light spots. Only when the values were set excessively high, such as around 2000, did the entity light up the area as intended. However, as a result, the lighting was more likely to look unrealistic. Having not played the game for too long prior to working on this step, Scott was unfamiliar with what values seemed appropriate and was content with how the light was being portrayed at those settings.

In an effort to ease work flow later, Scott opted to go ahead and add in light props for every light spot entity he placed. Having a rough theme of each team's base in mind, he selected appropriate 'prop_static' models and situated them the best he could to make them feel naturally placed.
Then, he made sure to place the light entity right before each model to give the effect that the model was the source of the light. While trying to place the light spot entities carefully was tedious and sometimes confusing because of needing to do so with three different window views of the map, Scott didn't have much trouble getting all of the lighting in place.

The final component of Scott's section of the project was to add textures and props to the team respawn rooms and bases so that they would look like a playable map as opposed to a map in development. Scott started by working on the red team first, which tend to be more red, earthy, and wooden. With this in mind, he modeled the red team's areas around a wooden structure that was built into the walls of the mineshaft, primarily using two different types of wooden textures for the walls and flooring/ceilings. The end result gave off a rustic feel, which was the effect that Scott had desired.

Blue team textures, on the other hand, were more industrial or metallic. Finding appropriate textures to use for the blue team was slightly more difficult and as a result Scott borrowed textures from the standard Team Fortress 2 maps. Scott used five main textures for the blue team area, for the floors, ceilings, walls, the exteriors of the base, and sides of inclined surfaces.

After he had finished adding in textures to both sections, Scott wanted to add in props for some of the inclines that connected rooms. The logical choice seemed to be staircases, though he often encountered an issue with the standard props not being the appropriate sizes to fit where he wanted them. The problem was made a little more difficult with the fact that props could not be resized without breaking them down and adjusting the model outside of Hammer. In most instances, he got around that problem by taking smaller models and using multiples of them side-by-side. For metallic staircases, however, he had the luxury of being able to copy a grouped object from other maps that was able to be resized to his needs.

On the whole, Scott feels like he is walking away from this project with a lot more experience in regards to game development as a whole, the ability to critically analyze components or features and how they reflect on the gameplay, and a greater understanding of the software and tools are used in professional development. With Hammer in particular, however, Scott was constantly frustrated by a few details, due perhaps to his lack of experience with the program, with the inability to resize the right sidebar which cut off large portions of text inside drop-down menus being the largest of them. Despite that, he is eager to use this experience as a basis to start working on other level and map design projects in the future.
4.2.4 Team Mineshaft, Door and Filter Functionality (This section is written by Luke Godfrey.)

Luke Godfrey had the tasks of implementing the logic for the automatic doors and designing each team’s mine shaft. As stated above, one advantage of this particular map’s design is that the entire map is symmetric. This essentially halved the work in the initial design process; once one side of the map was finished, it could be duplicated and mirrored for the other side. With regards to the team-specific mine shafts, Luke was able to design the red team’s shaft and duplicate and mirror it for the blue team’s shaft, along with a few thematic tweaks.

The first aspect Luke implemented was the logic required for the doors. The doors provide a gateway between the spawn room and the team mine shaft. The reason doors are necessary, rather than simply having an empty archway, for example, is that each team’s spawn room should only be accessible to its respective team. The red team should not have access to the blue team’s spawn room, and vice versa. The door logic would have to include a filter that chooses to open for the correct team only.

In the Hammer editor, there are two main types of visible objects: textured “brushes” (which are simple geometric shapes, like a cube) and props. Brushes were used for most of the map, and are usually implemented as solid objects that the player can’t walk or fall through, such as a wall or a door. They also “seal off” the map to prevent memory leaks. These brushes can then be textured to look like the objects they are supposed to represent. The textures range from rock to metal to wood to an invisible “no draw” texture used for invisible boundaries or the faces of a texture block that the player can’t see. Props, on the other hand, are 3D models that offer a greater level of detail, but are generally more expensive to render, less versatile, and do not “seal off” the map to prevent memory leaks.

To implement the doors, Luke used a combination of brushes and props. Each door was made up of a door prop (the visible part), an invisible “no draw” texture block (what the player actually ran into), and logic that connected the two pieces. The logic was associated with the texture block, and the prop was set up so that it recognized the texture block as its “parent.”

The final piece for each door was a trigger; a designated block of space near the door that fired an event that triggered the door to open or close. The trigger space was set up so that anytime a player entered the trigger area, it would check if the player was on the correct team, and if so, would open the door (in this case by sliding it up and out of the way). When all players have exited the trigger area, the trigger closes the door (by sliding it back down into its original
position). In addition, a “player clip” wall was added in the doorway to prevent the opposite team from following an opposing player into his or her spawn room.

After implementing the doors, Luke proceeded to design the red team’s mine shaft. The first step was to replace the existing blue and orange development textures with “no draw” textures, so that only visible surfaces would actually have drawn textures. To match the theme of a mine shaft, the walls and some of the floors were then textured with a rock wall texture. The main tower in the middle of the mine shaft was supported by four pillars, which were textured as wooden beams. Most of the floors, especially the floors supported by the wooden beams, were textured as wooden platforms.

In addition to simple textures, Luke used a number of props. Some of the wooden platforms appeared to be awkwardly jutting out from the central tower, so Luke added a support prop that made those platforms appear more natural. At the bottom of the main tower, just above the water level, the original map had one of the four sides of the tower blocked off with a texture block. Luke replaced it with a fence prop, accompanied with an invisible “no draw” texture block for collisions. Similarly, he removed a part of the floor and added a “no draw” block with a prop that looked like a couple of loose boards. In more than one location, Luke replaced the texture block ramps with “no draw” ramps and used a more realistic-looking prop that looked like wooden stairs.

After adding textures and props, the map looked much more like a mine shaft. However, when compiled, the map was still very dark. The next addition to the team mine shaft area was lights. Luke added a number of lights, each with a prop to correspond to it. Hanging lamps and mounted lights fit the theme of a mine shaft, so that is what was used to add light to the map.

After finishing the red team mine shaft, Luke had Taylor copy everything and mirror it for the blue team mine shaft. Although the map looked good and was completely textured at that point, the blue team’s shaft didn’t fit the blue motif (“metal” or “industry”). Several of the props were replaced with similar metal props. The wooden stairs, for example, were replaced with metal stairs. The loose boards covering a hole at the bottom of the tower were replaced with a section of chain-link fence on its side.

Several textures were also replaced in the blue team’s mine shaft. The wooden support beams for the central tower were replaced with steel beams, and the wooden platforms were replaced with metal plates.
One of the most useful tools for designing the team mine shafts, beside the Hammer editor itself, was the TF2 Prop Library. The TF2 Prop Library is a community-made file that can be opened in Hammer as a map, but is more like a visual catalog of available props. Luke was able to use this resource to find all the props he used in his section of the map, including the props for each door, all of the different light fixture props, the fences and gate props, the support props, and the wooden boards prop.

As Luke had no previous experience using the Hammer editor, this was an excellent learning opportunity. Understanding the process of designing a map from a critical viewpoint proved to be a key point of the project. Luke was able to utilize a number of online tutorials, as well as the hands-on help provided by Taylor, to learn while he worked. Beyond the scope of this project, the concepts learned and experience gained by Luke will equip him to work on other, similar projects in the future. Beside Team Fortress, Hammer can also be used to design maps and levels for Portal, Left 4 Dead, and several other Valve games.

In addition to using the Hammer editor, Luke learned some crucial concepts in effective game design. Jumping right into the creation phase can cause gameplay elements to break, such as other capture-the-flag maps in Team Fortress 2. Careful and critical planning, on the other hand, reduces the potential of any failure and results in greater success.

4.2.5 Central Battleground and Side Passage, CTF Mechanics and Rules (This section is written by Dylan Reile.)

My contribution to the project consisted of functioning capture-the-flag (CTF) mechanics and developing the pleasing aesthetic of the central battleground and team side passages.

Capture-the-Flag Mechanics

After Taylor provided the primitive environment filled with development textures (default textures with a neutral aesthetic before the initial art pass), the next phase was to add working gameplay mechanics. The map must be told what type of game will be played within it. This is accomplished via placing pre-designed “gametype entities” provided by Valve at arbitrary locations within the map. These entities are essentially objects possessing states and together compromise an in-game object-oriented programming language.

Gametype Object - This object defines the gametype (in this case, CTF).
- Scoring Object – This object defines the round scoring; such as the number of points given for capturing a flag, the number of points that win a round, the time limit of each round, and the number of rounds.

- Intelligence Object- Also known as the flag, this object must be captured by the opposing team and brought to their own base object in order to score a point. This object has several states including the time it takes a dropped intelligence to return to its original spawning point and the team to which it is assigned (red or blue).

- Base Object – This object has state dimensions (width/height/length) and a single method that responds to the intelligence object being placed within its parameter by rewarding the capturing team with a point and resetting the position of the intelligence object.

Central Battleground and Team Side Passages Art Pass

After the team had a completely functioning game with which to work and playtest, the majority of development time was used on the art pass. The art pass consisted of replacing the default development textures with textures that were aesthetically pleasing to end-users and produced a virtual environment consistent with the Team Fortress 2 universe. This pass included lighting, textures, and props which while not directly affecting gameplay, are significantly important for establishing a playerbase.

- Lighting – If no manually placed sources of light (light entities) exist within the map, the map will light all areas of the map with default lighting. This default lighting is harsh, unrealistic, aesthetically unpleasant and considered by the game’s playerbase as extremely unprofessional. The first step in resolving this issue is to place a single light entity within the map, causing the map to recognize the manually placed object and turn off the default lighting. Without the default lighting, it is easy to recognize areas of the map that are poorly-lit and strategically place light sources to lighten them. These light entities act as objects with definable states, including axis, luminosity, direction, and position. For the central battleground and team side passages, I utilized point-light entities.

- Textures – After gameplay mechanics are finished, the default development textures are replaced with textures that make sense in that environment. As our map takes place within flooded mineshafts, I replaced the default textures in the central battleground and team side passages with wet, earthy textures that are consistent with that setting. The
walls, ceilings, and floors were assigned wet, earthy textures while the ramps, tunnels, and guard-rails were replace with a type of wood that the miners of the time would have access to.

- Props – Unlike light objects, props are essentially purely aesthetic objects; they are merely pre-rendered and methodless. While often overlooked, props are essential to the overall aesthetic of a map. Unfortunately, I do believe that I mostly overlooked this in my sections. For instance, my point-light entities, when played by an end-user will appear to be disembodied sources of light without so much as a street light or lamp emitting them. I will bring the knowledge of their importance into future projects dealing with virtual worlds.

Contributing to this project has gained me a wealth of experience in virtual worlds and specifically game development. This experience includes the knowledge of and methods to avoid a thousand pit-falls that can ruin a game before it even begins being developed. I will discuss some of these below:

<table>
<thead>
<tr>
<th>Risk</th>
<th>Risk Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom-up Approach</td>
<td>A game should be developed with a stringent top-down approach. The final, desired end-user experience and goal of the game should be heavily considered before development brainstorming even begins.</td>
</tr>
<tr>
<td>Fine-Grain Control</td>
<td>Developers will frequently desire the maximum possible control over their projects and begin developing with “fine-grain” tools. While noble in intention, this is oftentimes the equivalent of using a pencil in MS Paint to fill an image instead of the paint bucket. This can be avoided by using larger tools to texture large areas of the map concurrently.</td>
</tr>
<tr>
<td>No Version Control</td>
<td>A strict separation of tasks should be established and enforced via version controlled software such as Dropbox, Subversion, Git, or Skydrive. This allows each team member to “own” a specific area of the map and for mistakes to be easily revoked.</td>
</tr>
</tbody>
</table>

4.2.6 Problems and Obstacles

While the team was able to meet the milestones of the schedule for the most part, we were not able to have large-scale playtesting and feedback. This was an anticipated barrier, given that the voluntary participation of third party custom map server administrators was not guaranteed. The timing of our development schedule just did not line up. In addition to that, the team was already overwhelmed with learning new technology, tools, and development methodologies that getting the initial map pass finished occupied the majority of the team’s attention. There would not have been enough time to act on feedback from playtesters anyway.

Another issue was that of a cut section of the map. Initially, there were plans for a “flooded cave” just below the central battleground. It would have connected the two flooded mineshafts from their bottoms, accessible via a buried underwater passageway. However, there was a
technical issue related to how Source renders water: Adjacent, connected bodies of water must share the same water surface height, or else geometry beyond the lowest water surface will either not be rendered or suffer from rendering issues.

Since the idea was to swim from the low elevation mineshaft bottom into the flooded cave (from which players can emerge at a higher elevation, thus serving as a unique means of moving players to a higher height), this limitation posed a major obstacle. We could have either lowered the render quality of the water (which fixes the surface height rendering issue in Source) or just remove the section. Since there were already concerns about team member workloads and whether or not the flooded cave would even be necessary for gameplay, Taylor decided to seal off the area and remove it from the compilation. (However, the section still exists in the map file as a disabled visgroup, should future playtesting suggest it would be worth returning to.)

4.2.7 Future Work

The map still needs a lot of work before a release is viable, but there is a solid foundation. Most importantly, the mechanics and functionality are all there, minus a few details (for example, defined spectator camera viewpoints). As the original designer and an avid TF2 fan, Taylor plans to take the results of the project and bring it to completion within the near future, hopefully over the following summer. This work will mostly consist of art passes, playtesting, and design iteration.

A visual illustration of the flooded cave’s rendering issues
4.2.8 Screenshots and Visual Progression
4.3 Risks

<table>
<thead>
<tr>
<th>Risk</th>
<th>Risk Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team inexperience with TF2</td>
<td>Taylor designed the map since he has extensive TF2 experience, leaving other team members to tasks that do not require deep understanding of the game</td>
</tr>
<tr>
<td>Team inexperience with finishing and publishing maps</td>
<td>Leverage prior failed project experience to avoid pitfalls; low scope; effective milestones; using established game types and mechanics, focusing solely on design</td>
</tr>
<tr>
<td>Lack of time to adequately teach Hammer and build the map</td>
<td>Build the map in distinct stages, beginning with a fully working map that uses only primitive shapes and textures</td>
</tr>
<tr>
<td>Dependence on teammates and file sharing</td>
<td>The map has been divided into independent sections that can be distributed and developed without relying on the progress of teammates. The primitive map already works and compiles, so teammates will be responsible for their own sections and can be swapped out for default “working” sections if need be. Teammates can succeed as individuals even if the project as a whole fails.</td>
</tr>
</tbody>
</table>

4.4 Tasks

1. Analyze and understand the issues with the CTF map type in TF2
2. Design a CTF map that incorporates these ideas
3. Implement a working map made with primitives
4. Test design and mechanics privately and on public testing servers (if possible)
5. With a (near) final design, execute an “art pass”
6. Optimize map for network play
7. Finalize and release map

4.5 Schedule

<table>
<thead>
<tr>
<th>Task</th>
<th>Fall</th>
<th>Spring</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Analyze</td>
<td>Late January</td>
<td>Early February</td>
</tr>
<tr>
<td>2. Design</td>
<td>Early February</td>
<td>Early March</td>
</tr>
<tr>
<td>3. Implement</td>
<td>Mid-March</td>
<td>Early April</td>
</tr>
<tr>
<td>4. Test</td>
<td></td>
<td>Late April</td>
</tr>
<tr>
<td>5. Art Pass</td>
<td></td>
<td>May</td>
</tr>
<tr>
<td>6. Optimize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Release</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.6 Deliverables

- Primitive Map VMF (Valve Map File)
- Final Map VMF (Valve Map File)
- Final Report

5.0 Key Personnel

Note: Team Member descriptions are written by their respective students

**Taylor Yust** – Yust is a first year graduate student in the University of Arkansas computer science program. He has experience developing games for smartphone platforms and the Unity game engine, and his familiarity with the Source SDK and Hammer will aid the team’s efforts. Having attended multiple conferences (including the Game Developers Conference) with his gaming software, his background has made him uniquely qualified for this project.

**Scott Benton** – Benton is a third year undergraduate student in the University of Arkansas’ computer science program. He has extensive experience in web design and development, and is passionate about learning about game design and development. He has contacts that have experience with map-making for various games and Hammer that should help answer key questions for the project.

**Luke Godfrey** – Godfrey is a third year undergraduate in the University of Arkansas’ computer science program. He has experience with web technologies and has created several websites. He is also an iPhone developer, and has an app in the Apple App Store. His dedication to his faith as a Christian is what motivates him to work hard in every project in which he is involved.

**Dylan Reile** – Reile is a student taking Advanced Virtual Worlds at the University of Arkansas. He has prior familiarity with Hammer and has an interest in map-making.

**Dr. Craig Thompson** – Instructor for the course and advisor to the project, Dr. Thompson has a keen interest in virtual worlds and its overlap with gaming. He is exploring the application of game design in “gamifying” serious subject matter, such as health and education.

6.0 References

