MQP MBJ 1601

INTERACTIVE CINEMA

A Major Qualifying Project Report
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

by

Daniel Driggs
William Frick
Jacob Hawes
Derek Johnson
Benjamin Korza

April 28, 2016

Advised by:

Brian Moriarty, IMGD Professor of Practice
Ralph Sutter, IMGD Visual Art Instructor

Abstract

The Piper is a first-person interactive cinema experience based on the legend of the Pied Piper. Set in medieval Germany, the player assumes the role of a child being lured away from the village of Hamelin under the vengeful spell of the Piper's music. Our team consisted of two programmers, two artists, and a music/audio producer. This report discusses the design goals of The Piper, the methods by which it was developed, technical and aesthetic challenges that the project faced, and the team's reflections on the development process and final product.

Acknowledgments

Our group would like sincerely thank the following individuals for their help and support throughout this project:

- Professor of Practice Brian Moriarty, Instructor Ralph Sutter, and IMGD graduate student Caitlin Malone for their guidance and input.
- Ichiro Lambe of Dejobaan Games for lending us his Oculus Development Kit.
- Michael Voorhis for setting up and running our Perforce server.
- Helen Lisanti for providing expert voice talent in a short timeframe.
- Dillon DeSimone for providing illustrations for the opening narration.

Contents

1. Introduction

The Piper is an interactive storytelling experience, meant to retell the climax of the classic fairy tale of the Pied Piper. The player takes on the role of a child who has succumbed to the Piper's hypnotic spell. Together with a crowd of other children, the player's character is swept through the village of Hamelin and into the nearby forest, finally being led through a mysterious tunnel to an ambiguous ending. We planned on including at least one alternate ending, where the player escapes or stops the Piper's plan, but we eventually settled on a simplified experience that remains faithful to the spirit of the original source.

2. Background

One of the notable qualities of the Pied Piper legend is the number and variety of interpretations in existence. It is apparently based on a historical event which, according to early documents, occurred on June 26, 1284 in the village of Hamelin, Germany. (History Channel) The original tale was tantalizingly vague, and has been embellished with fantastic details by generations of storytellers. We studied several of versions of the legend to get an idea of where we wanted to take our game. Initially we stuck closely to Robert Browning's 1842 poem, but ultimately decided to take the story in our own direction. (Browning)

2.1. Story summary

The town of Hamelin finds itself inundated by rats. They've taken over the city and are creating havoc, leaving the townsfolk desperate to be rid of them. One day, a strange man in pied (multi-colored) clothing appears, offering to eliminate the rats for a fee. The townsfolk eagerly agree, and the Piper sets off, playing a magical melody that lures all of the rats out of the village and into a nearby river, drowning them.

When The Piper returns to collect his pay, the townsfolk refuse to uphold their agreement. The vengeful Piper plays another melody, leading the townsfolk's children away from the village, never to be seen again.

Interpretations of the children's fate differ widely. The oldest versions speak of them being "lost" or "swallowed" upon a mountain, possibly alluding to a landslide or cave collapse.

Others tell of them being drowned, like the rats. We decided to keep the ending of our version more ambiguous.

2.2. Experience goal

The experience goal of *The Piper* underwent several changes throughout development. The first draft of the goal was "To breathe new life into an old tale, immersing the player in a whimsical world which slowly devolves into a sinister, panic-inducing nightmare." This concept of contrasting between the magical, trance-like world of the Piper's spell and a more disturbing reality was largely inspired by the short film *Don't Hug Me I'm Scared* (2011). This film initially presents itself as an early childhood educational program before descending into a morbid, surreal video collage.

As work proceeded, this concept was gradually scaled back, replacing the original's "sinister, panic-inducing nightmare" with a more ambiguous ending. In addition to being smaller in scope, this ending is also more consistent with the original legend, which never explicitly describes the children's fate.

3. Technical implementation

We chose to develop *The Piper* on Epic's Unreal Engine 4 (UE4). The question of whether to use UE4 or its chief competitor, Unity, arose early in the development process. Unreal Engine 4 was chosen for its superior lighting and graphics potential, blueprint programming system, and because our team was already familiar with it.

3.1. Piper movement

The Piper travels along a spline. In order to allow the Piper to stop and conduct other actions, special volumes exist that delay the Piper when he enters them. These delays end when either a set amount of time expires or a Boolean is flipped. The delay volumes are used at several set piece locations, like when the Piper raises the bridge.

Certain out-of-view movements of the Piper are achieved using separate Piper objects; for instance, the Piper beckoning to the player at the beginning is an entirely separate object, which is destroyed when it leaves the player's doorway. The actual Piper is kept waiting behind

a well in the courtyard in front of the player's house. A similar trick was also used in an early version of the game, where the cave led to a cliff. The Piper on the cliff was also a separate object.

The Piper is programmed to stop and wait if the player falls too far behind. To prevent the Piper from constantly moving and stopping if the player straddles this threshold, two radii are used. When the player exits the outer radius, the Piper stops, and does not resume until the player enters the inner radius. This ensures that, once the Piper begins moving, he will have a reasonable distance to travel before he may be required to stop again.

Fig. 1: Original flocking in action

3.2. Child Movement

Child movement underwent two distinct iterations. The first was created by hand and allowed for greater control over the finer aspects of the children's movement, but proved difficult to optimize. It was eventually replaced with a crowd system built into Unreal Engine, known as Reciprocal Velocity Obstacle, or RVO. Below, the original, custom-built system is presented as originally implemented, followed by the built-in RVO-based one.

3.3. Original flocking method

In order to accomplish this effect an AI technique called flocking was introduced. At its most basic level, the technique works through the usage three simple steering behaviors: separation, alignment, and cohesion. Cohesion is the behavior that causes flockers to move toward the average position of their local flockmates. Alignment is the behavior that causes flockers to steer towards their local flockmates' heading. Separation is the behavior that causes flockers to avoid each other. (Reynolds) Combined, these behaviors form the basis of a general flocking technique.

In contrast to most flocking simulations, the children in this project move along a spline. This has some implications on the way their flocking can be conducted. Since the children's cohesion to the spline is the most crucial attribute of their movement, cohesive forces between children are out prioritized and therefore unnecessary. There is also no need for an alignment parameter since a child's alignment should match the direction of its velocity vector. The only behavior really needed is separation. Therefore, the children are programmed to flock away from each other by first calculating vectors of separating forces. These vectors are then applied in two different ways. The first way is by pushing children that are too close to each other away from one another. The second way involves offsetting their spline path using the forces calculated in flocking. The final behavior is a result of these two sub-behaviors combined. This resultant behavior is demonstrated in Fig. 1, where all of the children maintain a distance of roughly several yards from each other.

In terms of performance, flocking can be a computationally expensive task. The technique requires each child to do checks against all their nearby children to determine how to apply forces. As such, the algorithm employs a number of adjustable parameters in order to improve performance. For instance, the maximum number of children that the flocking algorithm will consider and the rate at which the flocking algorithm updates can be modified. By changing these parameters the algorithm's performance can be improved, but usually at the expense of accuracy. The correct balance between performance and accuracy is necessary to generate a convincing crowd of children.

3.4. Improved flocking method

The final implementation of child flocking uses Unreal Engine's built-in RVO system. This system, which has significantly fewer exposed parameters, is activated within each child's Character Movement Component and applies to certain navigational actions provided by that component. With RVO activated, children are simply instructed to pursue a target, using a Move To Actor command, and RVO ensures that they avoid each other in a crowd-like fashion.

Initially, children were instructed to simply follow the Piper. However, this led to unnatural corner-cutting behavior when the Piper was out of sight. Additionally, while children crowded around a stationary Piper realistically, they tended to form a single-file line when pursuing him from a distance. Both of these issues were addressed using a "fuzzy point" system. Rather than simply follow the Piper, each child follows an array of navigation points throughout the city. This alone solves the first issue; with points placed at either end of each straightaway in the game, children never need to cut a corner. The single-file issue was fixed by giving each point a radius; each child picks a random point in that radius, leading to slight scattering that is far more crowd-like than a single-file line.

3.5. Player movement

Restricting the player's movement to the game's linear path was an important task. In keeping with the more whimsical imagery of the original game concept, we initially envisioned using fanciful characters such as clowns, jack-in-the-boxes, or cartoon rabbits to corral the player and block off exits. Additionally, to keep with the game's intended disturbing effects towards the climax, we planned to have later instances of these characters become startlingly aggressive. This idea was eventually abandoned due to scoping issues. We considered keeping the exits and blocking them off with more simplistic soft walls (see below), but ultimately closed them off with hard barriers.

An artificial barrier known as a soft wall was designed to more subtly direct the player. Soft walls are invisible fields, with a distinct front and back. When the player is in such a field, it exerts a force upon them, proportional in strength to their depth through the field from front to back. The created effect is a soft push, gently dragging the player backward as they attempt to bypass the field. Although the final game does not use soft walls in the alleyways originally intended for the pop-up characters, soft walls are used in a few situations. One such wall blocks

off the alleyway from which the children emerge, to prevent the player from witnessing the spawning process. They are also used in a staggered fashion in the forest to ensure the player does not backtrack.

In addition to physical barriers, there are some less forceful ways in which the player is encouraged to proceed. Once the player has exited the city, an effect is activated which changes the player's screen to grayscale when they are looking away from the Piper. The technical details of this effect are explained below, under Shaders.

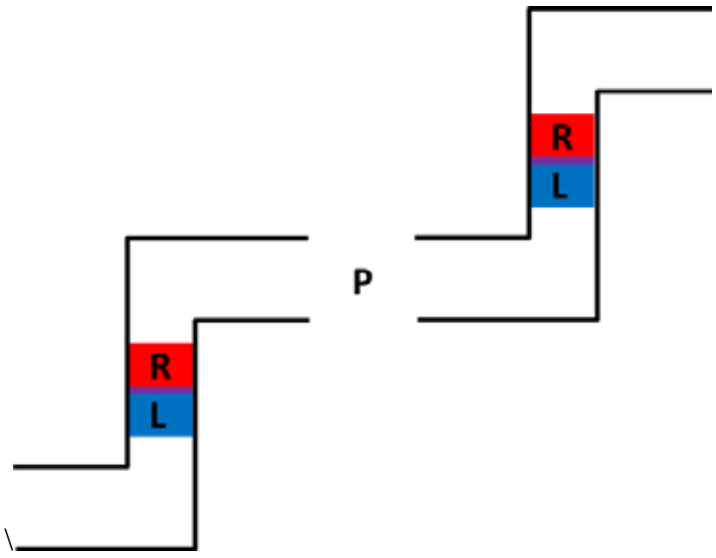

Fig. 2: The final spatial hallway setup. Zone R is inactive in the left hall, and zone L is inactive in the right. The purple area represents the intersection of both zones. The player enters one of the halls from the point labelled P.

3.6. Spatial hallways

Midway through the town, there is a pair of alleyways sitting opposite each other. Entering one of these alleys leads seamlessly to the other. These are referred to as spatial hallways, or spatial halls for short. Although this effect was originally intended to fit in with the more whimsical style originally envisioned, we kept the implementation in the final game. To ensure that the transition could be seamless, each hallway is an instance of the same class; their appearances are completely identical, and a public Boolean value is used to differentiate right from left in the Blueprint logic.

Achieving the seamless transition between the two halls was extremely difficult, and required several iterations. The first prototype simply teleported the player to a target point in the opposite hallway, which resulted in a noticeable jerk in the player's movement. To achieve seamlessness, the hallways need to account for the player's offset within the sender hallway, and recreate that offset within the receiver.

An immediate issue with this requirement is that, by definition, the player will be sent to an area of the other hallway which is within that hallway's teleport zone. Simply disabling the receiving teleport zone is not an option, as a player could, with good timing, start backing up the moment they teleport, allowing them to escape the back of the hallway. To combat this, we introduced a "nudge" mechanic which gently, almost imperceptibly, pushes the player forward after they are teleported. By disabling the receiving zone, teleporting the player there, kicking them past the zone, and then reactivating it, we can ensure that the player cannot escape nor enter an infinite loop.

A very small nudge strength is desirable, as the effect should not be noticeable by the player. In order for a small nudge to be sufficient, the zone must be small so that the player is pushed completely past its boundaries. Additionally, as the player's collision capsule is relatively thick, we added a tiny collision box to the player's center which is used for detecting teleport zones. With these zones being so small, the nudge required to push the player to safety is subtle enough to be nearly undetectable.

However, there is an additional problem created by shrinking the zones. It became possible for the player's zone to never overlap the hallway's. Due to the zones' tiny sizes, it was possible for the player to be on side of the teleporter one frame, and on the other side the next frame. To fix this issue, we used a system of overlapping zones. The hallway class has two relatively large zones, but only one is active in each instance, determined by the Boolean value which signifies whether a hallway is on the left or right. The intersection of these zones is of the same size and placement as the original, small, single zone. Since the active zone in each hallway extends back beyond the overlap, the player has plenty of room to ensure they intersect the active teleporter, but after the nudge they are no longer in the receiver's active zone.

3.7. Player look detection

We found that players were not always witnessing the Piper's climactic opening of the city gates. To improve the chances of this happening, we implemented a system to determine if the player has the gate in their sight. By detecting the rotation of the player's camera and comparing it to the direction from the player to the gate, the game detects whether they will witness the opening sequence. To ensure that the player does not accidentally activate the sequence from afar, there is a minimum distance from the gate that the player must be within before the game will recognize that they are looking at it. Additionally, since the player should never pass the Piper, there is a smaller "panic" radius around the gate which automatically opens the gate if the player enters. This ensures that a conniving player cannot trick the game by walking backwards to the gate.

We originally intended to also implement a look-detection system for the bridge raising sequence, but this was not implemented due to time constraints. We prioritized the gate because, being set in an open area, there was a higher chance that the player would not be looking at it.

3.8. Cinematics

The game is bookended by brief cinematic sequences, which underwent multiple iterations. The opening sequence was originally meant to involve the player lying in bed, being read the story of the Pied Piper, falling asleep, then waking up in the dream world wherein the game takes place. This dream concept was eventually scrapped for scope reasons; at the time of its abandonment, the sequence simply consisted of a delay and a fade in and out of black, representing sleep. The player's movement was disabled until after the fade. The fades were generated using Unreal Engine's matinee feature.

The dream concept was replaced by a simple storytelling approach; a disembodied voice tells the player a basic summary of the Pied Piper leading up to the game's events, and then fades into a bedroom. This sequence is conveyed using still images projected onto planes in 3D space; the player is placed in a black box which fades away at the sequence's conclusion.

The ending of the game was originally meant to feature the player being jolted awake, back in their bedroom, hearing the story's conclusion. With the dream intro being abandoned, we opted instead for a fade out, followed by the introduction's disembodied voice delivering the story's ambiguous conclusion. Like the original intro prototype, this fade is generated by the matinee tool within UE4.

Fig. 3: Screen space shader in effect. When the player faces in the direction opposite to the Piper, the image goes from color to black and white.

3.9. Shaders

As mentioned under Player Movement above, we used color and grayscale to entice the player to follow the game's intended path. Two different shader techniques were investigated in order to determine how to best accomplish this effect. The first involves calculating the player's angle to the Piper and applying a screen space desaturation shader whenever the player is looking away from the Piper. This has the effect shown in Fig .3. It has the limitation that it can only be cast over the entire screen, and cannot be applied on a per-texture basis. However, the technique is easy to implement in Unreal and is not very expensive computationally.

Fig. 4. Node-based material shader technique.

The second technique uses node-based shaders attached to the material components of objects. (M-H) As seen in Fig. 4, this technique has the advantage of being able to apply the desaturation effect both to objects individually and at different locations on their texture; however, it has the disadvantage of being computationally more expensive than the former solution. It also requires more work to integrate it into all of the materials using the effect.

3.10. Menu system

Two menus are present in the game. The first is the main menu, which is displayed when the player boots the game. The other is the pause menu, which can be accessed by pressing P or, on a controller, Start. These menus were both created using Unreal Engine's UMG Widget tools.

The main menu features three buttons. The play button cuts off player input, displays a loading prompt, and loads the main level. The quality button cycles through the Low, Medium, High, and Epic settings provided by Unreal Engine, running console commands in the background to change various settings. The quality button is explained in more detail in the next section. Lastly, the exit button quits the game.

The pause menu features two buttons. The resume button unpauses the game and hides the menu. The exit button returns to the main menu. Quality settings cannot be adjusted in the pause menu.

These menus proved problematic for virtual reality. By default, when rendered in screen space, they do not correctly duplicate to both eyes, leading to the menu being split between the player's eyes. This renders the menu unreadable. The suggested solution to this problem is to attach the widget to a 3D object which is spawned in the world (Creating 3D Widgets, 2016). This approach worked for the main menu, but resulted in significant difficulties for the pause menu. We found it was not possible under the current implementation of 3D widgets (currently an experimental feature) to spawn a widget which would run while the game was paused. Because of this setback, we instead used a static menu with button prompts when in virtual reality.

3.11. Quality Settings

To support a variety of players and machines, the main menu allows the player to change the game's video quality settings. These are edited via console commands. There are a handful of individual settings, such as anti-aliasing and shadow quality (Scalability Reference, n.d.). While the scalability command affects all of these at once, it is only enabled in-engine and has no effect in a packaged executable. Instead, individual commands for each setting must be used.

3.12. Metrics

Player metrics are both a valuable tool in the debugging process and a good way to determine faults that undermine the project's experience goal. In order to gather data on a player's run through, the application records various statistics and outputs them to a log file. These statistics can then be viewed by opening the log file in a text editor. The statistics recorded in this log file are listed below:

- Playtime
- Time matching Piper direction
- Time looking at Piper
- Time standing still
- Number of popups triggered
- Number of hallway teleports
- Average frames per second

The log file also records a history of data collected in real-time. This real-time data includes the time and positions of the player when certain events occur, like the player stopping or triggering pop-ups. This real-time data in combination with the recorded statistics provides a fine granularity of data to identify bugs and assess how closely the player's experience matches the projects experience goal.

4. Applications and tools

Below are the different applications that we used to create *The Piper*.

Tech	
Microsoft Visual Studio 2013	Compiling engine, C++
Art	
Adobe Photoshop	Texture creation, general image editing
Autodesk Maya	Geometry creation, UVing, texturing, rigging, animation
Pixologic ZBrush	High-poly sculpting
Marvelous Designer	Clothing for characters
SpeedTree	Tree asset creation
Substance Painter	Texture map creation
Topogun	Retopologizing high-poly assets
World Machine	Mountain creation
Quixel Suite 2.0	Texture map creation
xNormal	Texture map baking
Audio	

Adobe Audition	Sound effect editing and touching up
Cubase	Music arrangement and composition
Audiokinetic Wwise	Real-time adaptive music engine

4.1. Team communication

Most communication within the team was achieved via the application Slack. Separate channels were established for specific topics such as art and audio, but most communication occurred in the private, general-topic #students channel. On rare occasions, other communications mediums were used as backups, such as email, SMS, and Facebook Messenger. A Trello board was established for task management early on, but quickly fell into disuse.

4.2. Source control

Our source control solution for The Piper was Perforce. Used by many game and film industry companies and integrated into Unreal Engine 4, it was a superior versioning service than our previous method, Git, which will be discussed in more depth in the conclusion of this report.

4.3. Wwise integration

The project needed a flexible solution for adapting music dynamically, and in time with the tempo. To resolve this concern, Wwise was integrated into Unreal. Wwise is an interactive sound engine for games, but i’s most sought feature for this project was its flexibility with interactive music. Although Unreal Engine also has functions that handle audio, these do not allow for the same level of control as software dedicated to integrating audio like Wwise.

The expense of using Wwise is the time it takes to integrate it into the engine. The software requires building a custom version of Unreal Engine from the source code found on their github page. This is a considerable time sink, but it only lasts as long as the custom engine’s build time. Once the build is complete, all the functionality of Wwise is accessible from within Unreal.

4.4. Virtual reality integration

Virtual Reality (VR) integration was a goal that we had set out for at the beginning of the project, as the genre of an interactive cinema and immersion inducing qualities of a head mounted display (HMD) go hand in hand. We were originally given an OSVR (Open Source Virtual Reality) Hacker Dev Kit, which is produced by Sensics and endorsed by Razer. Difficulties with successful integration arose (see section 4.4). We ultimately ended up going with the Oculus Rift DK2 HMD, as integration was nearly seamless. Following Unreal's Online Documentation (https://docs.unrealengine.com/latest/INT/Platforms/Oculus/QuickStart/index.html) allowed for us to get the Oculus Rift up and running within a few hours.

4.5. Maya tools

A few tools were created in Maya using Python scripting to support the artist to create more assets much faster. Among these tools were a more efficient export tool. The exporter handles the process of moving an object to the origin of the world, freezing its values, exporting, then moving the object back to its original position. Having all these processes be automated has saved time and effort of the artists getting assets into the engine. Another tool was a mesh populator which would be used to add objects across a surface of another mesh, such as leaves and/or branches on the base model of a tree. This would have made tree creation faster.

4.6. Miscellaneous tools

We compiled the Unreal Engine 4 editor and our game with additional plugins such as Substance and SpeedTree Importer. SpeedTree Importer made it easier to import vegetation assets (such as trees) into the engine. The Substance plugin allows Unity to read Substance's proprietary .sbar file type, providing the capability for dynamic, randomized, and scalable textures and materials.

Fig. 5. Screenshot from *The Secret of Monkey Island*.

5. Art implementation

The style and aesthetic for The Piper took a while to develop, but we eventually settled on a unique mix of cartoon-like proportions with realistic texturing to provide the player with a sense of familiarity and fun. The art direction developed organically as assets were created, leaving us freedom to explore new creative avenues.

We were heavily influenced by the art of *The Secret of Monkey Island* (Lucasfilm Games, 1990). This classic adventure game's quirky angles and proportions seemed to fit our creative direction, but our modular level design process and organic approach to styling prevented us from fully achieving the same look. Nevertheless, traces of *Monkey Island's* inspiration can still be seen in the angular features of our characters and environments.

5.1. Historical art references

Note: This section of the report was provided by Caitlin Malone, a WPI IMGD graduate student, who specializes in medieval history. She assisted the team in ensuring that the assets created for The Piper were era-appropriate.

For this MQP, we researched the historical town of Hamelin, Germany and general concepts of 14th century medieval life. This time period was chosen as it corresponded with the original mention of the Pied Piper story, which was found in the church in Hamelin's market. We chose the time period of the legend, rather than the time that the legend supposedly took place, because it was a concrete date to work with and it fit with the "fairytale" inspiration for the game.

Specifically, we researched the church at Hamelin, the original story and its depiction in the town, the Pied Piper costume used by modern-day performers there, and whether or not Hamelin was a walled city at the time of the legend. We also researched the Black Forest in Germany for art inspiration for the woods outside the town. More generally, information and reference images were provided for period towns, houses and streets, with miscellaneous debris; gates, town walls, and fortifications; and clothing for the NPC children that follow the Piper.

We relied as much as possible on primary sources, particularly for features such as the town and medieval fortifications. Although authenticity was desired, artistic design was more important. For this reason we pulled from pictures not only of the town of Hamelin and the existing structures there, but also from other towns in Germany, and occasionally from towns in Austria, France and Great Britain. Although designs of houses many differ, the general use of plaster and timber was similar in all regions for the time period.

For fortifications, due to the wide variety of designs used in the time period, we restricted my search to Germany and surrounding areas like Austria. Other information, such as details of clothing and items in the streets, were pulled from medieval history secondary sources or pictures painted during analogous times. The Pied Piper costume was designed based on the costumes worn by performers in Hamelin and was used with little modification due to its fit with the "fairytale" design.

Fig. 6. Church “hero” object.

5.2. Environment

The environment of The Piper is made up of two distinct settings; the urban landscape of Hamelin, and the forest just outside of the city walls.

5.2.1. Modular meshes

Caitlin Malone provided research which showed what type of buildings were typically built during this era (see section 5.1 above). We decided to make the buildings resemble the Tudor style of house, made out of plaster walls, wooden beams, and red tiled roofs. Jetties (overhangs) are also present on our building design, though more exaggerated than they would be in real life.

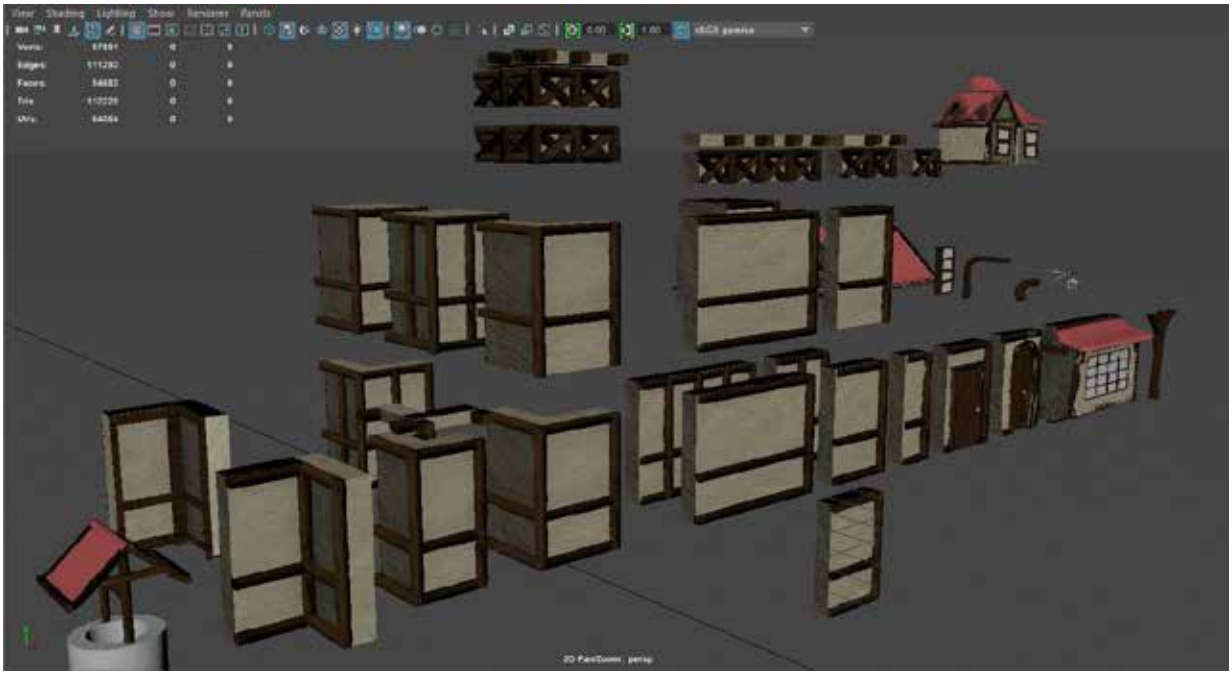

Fig. 7: Modular Pieces used to construct the city

To create the city, we constructed each piece of the buildings with a modular design in mind. Using the power of 2s and sticking to a grid system was imperative to ensure that each piece could seamlessly fit with every other piece. “Grids may fluctuate wildly between engines and projects, but whatever system you decide on, stick to it religiously, and always use even divisions of that grid when working on smaller components.” (Perry) Unreal Engine 4 uses a metric based grid system. As such, we built each building section in Maya sticking to meters. While building the walls was straightforward, with a one meter, two meter, and four meter piece, figuring out how to build modular roofs felt like it would be complicated. We decided to go the simple route. Two roof sizes were constructed, and simply scaled to fit each building if need be. This did not prove to be a problem with UV stretching, as the roof did not have to be scaled to such an extreme. The player would also never get close enough to the texture to see the stretching. By creating the roof at a power of 2 and applying a grid scale, we ensured that the roof fit each building perfectly.

Fig. 8. Example of a fully constructed building, in engine.

These pieces were then imported with the naming convention SM_Wall_x(size). Doing so allowed for the level designer to see what dimensions each piece was from the content browser. After dragging the piece into the world, grid snapping allowed for us to build each building in a timely manner, and ensured that there were no gaps between pieces as the vertices of each neighboring piece would meet at the same point in world space.

5.2.2. Unique meshes

Having purely modular grid-snapping pieces was not enough to create the city without apparent repetition. Other meshes had to be created in order to make the city feel as though it were not just a game set. Clutter, merchant tents, cemetery assets, clotheslines, and more were created to fill the world that the player explores. The hero object, the church in the center of the city, serves as the main point of attraction for players as they walk towards the city gate. All of these assets were modeled in Maya or ZBrush, and textured in Photoshop/nDo2.

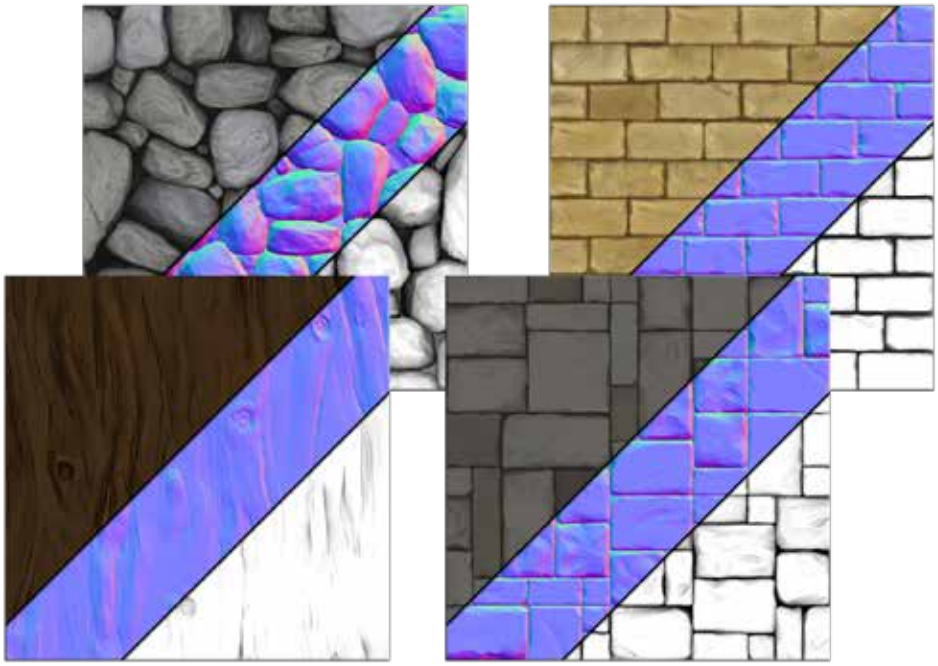

Fig. 9. Tileable textures used throughout the city.

5.2.3. Tileable textures

Tileable textures were used to quickly texture meshes. Not only are tileable textures necessary for certain modular pieces, but it allows for UVing certain assets, such as the burlap sacks that were used as clutter, quick and easy. Certain textures, such as the wood, metal, and stone wall textures, were sculpted in ZBrush and baked onto a plane xNormal. Details for the wood and metal textures were sculpted onto a plane, while the stone wall textures required physical stone bricks to be modeled. Julio Nicoletti's YouTube tutorial Seamless Texture with Maya, xNormal, Photoshop, nDo2 served as a valuable resource, and gave us insight on a workflow to create texture information. After generating the normal, height, and ambient occlusion maps from the high-poly sculpt in both xNormal and nDo2, they were all taken into Photoshop and used as RGB layer masks on color-filled layers. Different blend mode filters were applied on each layer to create a variety of different effects. This process was one of trial and error and full of tweaking, as changing the layer order or blend mode on even one layer can dramatically change the look of the texture. Other textures, such as the plaster texture, were created from picture reference Textures.com (formerly CGTextures.com), tweaked in Photoshop, and taken into nDo2 to generate the necessary maps.

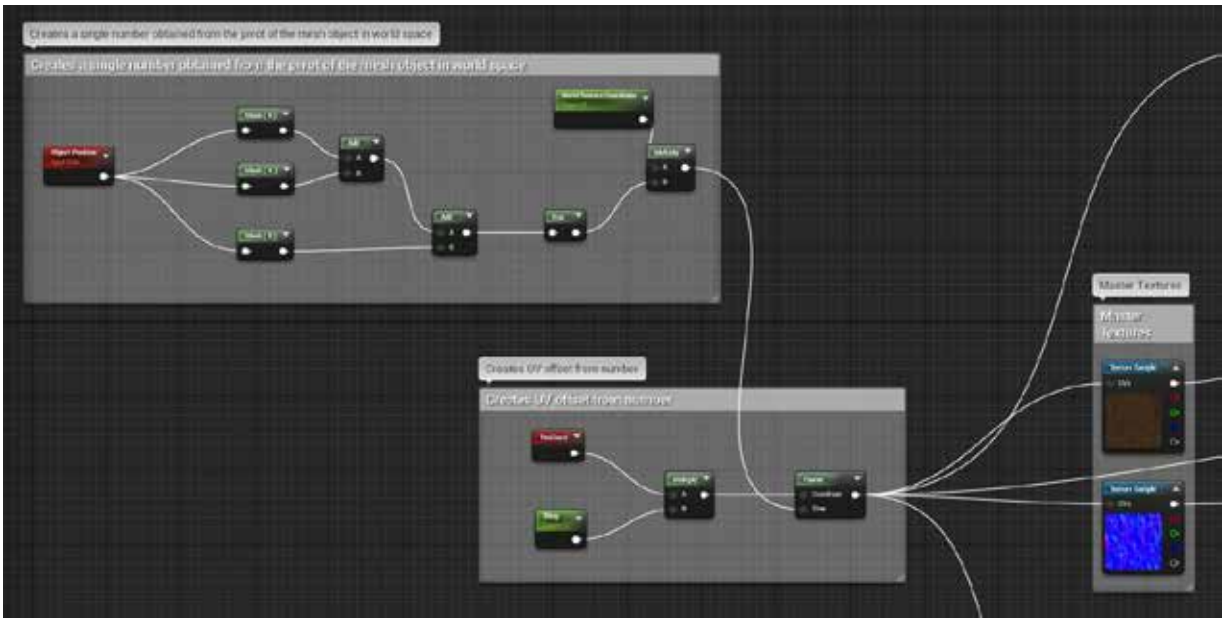

Fig. 10. Material setup to create world coordinate texture offset

To ensure that the player cannot tell that the textures are tiled, world coordinate texture offset was added to each material in UE4 that would make use of such function. As an object with this material is moved around the world, the texture that is on the object moves in UV space. For example, a plane placed at the origin with the material would have a different appearance as the same plane moved two meters to the left of the origin, as the texture in the UV space moved with the object. We dissected the “StylizedRendering” UE4 Project Example to see how they made use of this function, and modified it for our needs. The wooden beams, planks, and plaster materials share this code.

Fig. 11. Forest scene

5.2.4. Vegetation

In order to give the forest section of the environment a lush appearance, foliage was necessary. Simple planes were modeled and UVed to a single texture sheet that contains multiple foliage textures (hand painted as well as image reference). These planes were then arranged in such a way in Maya to give them the appearance that, when viewed from the side, are not merely planes. After importing them into UE4, we were able to make use of the built-in landscape tool set to paint on the foliage. The majority of the foliage is grass, with taller plants mixed in.

Trees were created in SpeedTree. As this was a new program to us, there was a small learning curve to create the trees exactly how we wanted them. Although the program is insanely powerful, the basics are easy to grasp. The tutorial SpeedTree Tutorial: Modeler Basics by SpeedTree Middleware on YouTube covered exactly what we needed to know. One thing to note is SpeedTree's built in Level of Detail system. Exported trees already contain highly optimized low-poly meshes, that even turn to billboards when the player is at a certain range. This type of optimization is something that would be insanely time consuming if done manually.

5.2.5. Lighting

The Piper takes place during the night. As such, overcoming the obstacle of having a nighttime looking environment that is still bright enough for the player to easily enjoy the visuals around them was an obstacle that we had to overcome. The main directional light source is set to a low intensity of 4.0, with a color overlay of a gray-ish blue tint. As the moon in the night sky is huge, the light that it gives off works with the shadows that it projects on the ground of both the buildings, and the characters. Light shaft occlusion is enabled, meaning that beautiful light shafts are visible through the exponential height fog as the rays are intersected by the roofs and other objects.

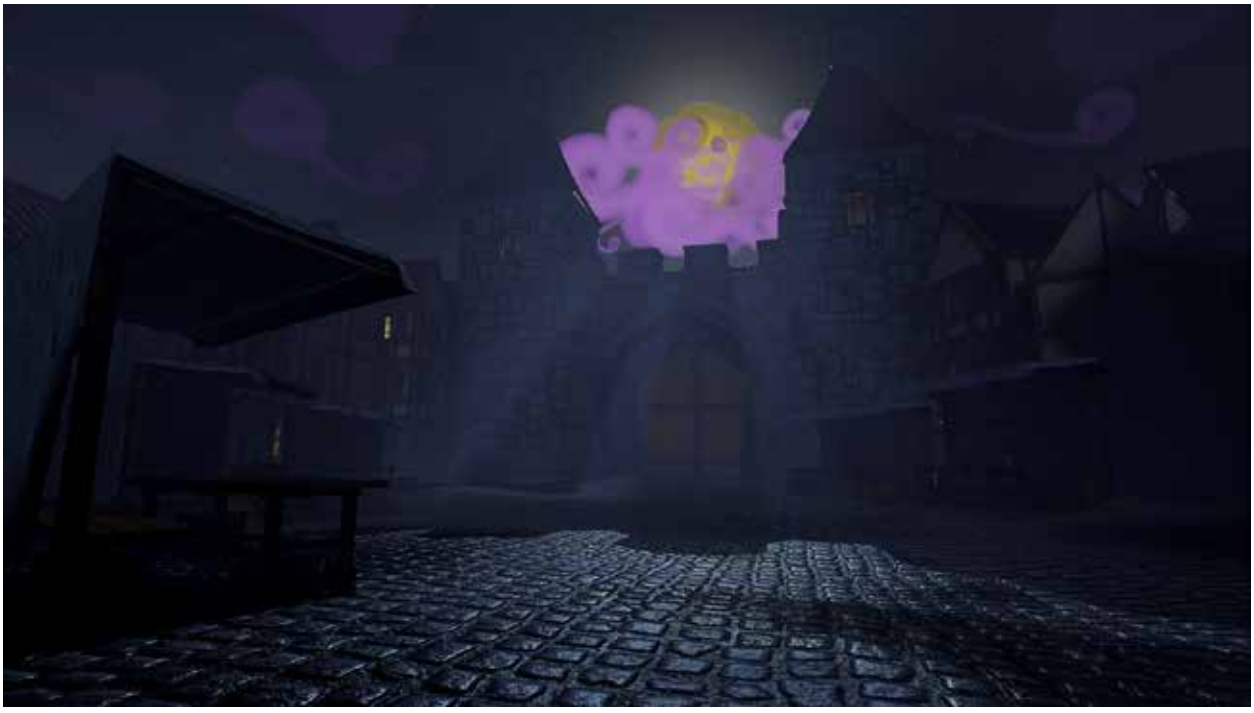

Fig. 12. Light shafts.

Lanterns are placed around the level to add extra light. Without them, the city would look dead. The light color of the lanterns is a saturated orange, to contrast with the rest of the dark blue scene. In addition to the lanterns, the lit windows of the buildings give off slight emissive light. Much like the lanterns, the light from the windows adds life to the city.

5.3. Characters

Fig. 13. Sculpt of the Pied Piper from ZBrush.

The cast of *The Piper* includes the Pied Piper himself, together with a number of Hamelin children.

The Pied Piper was designed to include angled features signifying a villainous or menacing intent. He is also intentionally proportioned to be extremely lanky to have an otherworldly or odd appearance to scare the player slightly.

The children were created to foil the Pied Piper's appearance. With smoother, more rounded features, the children are viewed as more innocent and friendly. Their proportions were also more akin to Pixar proportions with large heads and small features to provide the cartoony look we were attempting to achieve.

5.3.1. Meshes

The characters both started out as sculpts within ZBrush following the workflow of a Riot Games character modeler provided in a Zbrush Summit talk (Pixologic Zbrush). The process began with blocking out major shapes of the body to solidify the proportions of the character. Additional details were added on using dynamesh and various other brushes to obtained the desired look. A high-poly mesh of the character's body was created with proper facial and body anatomy.

Once the bodies were finalized, they had to be decimated, reducing the polygon count, in order for each piece to be made into one mesh instead of separate subtools. This singular mesh was brought in to a clothing program known as Marvelous Designer where clothing was simulated as real-world fabric on top of the body to achieve a believable look with weight and proper folds. Most of this work was performed with prior knowledge of the program, but additional tutorials were used to gain a better understanding of the functionality of some features ("Getting Started with Marvelous"). The clothing was exported back into ZBrush because it is the only 3D software which could handle the large polygon count created by Marvelous.

Fig. 14. Child sculpt from ZBrush.

Back in Zbrush, the clothing was sculpted to add a similar style to the character. For instance, the Pied Piper has many angled features, so the clothing was sculpted to reflect those angles. The children's clothes were left the same after importing because the folds were more rounded, matching the children's features.

The meshes were, again, decimated to reduce polygon count and exported to a program designed to optimize topology of a mesh known as TopoGun. This intuitive program allows for polygons to be drawn on top of a high-poly reference mesh, creating an incredibly optimized low-poly, game ready mesh fairly quickly. Proper topology flow was researched on a game development wiki and was used to ensure proper deformation in animation and easier rigging ("Topology"). The detailed information of the high-poly model was then transferred to the newly created low-poly model with textures through a process called baking. XNormal was used for this process.

The low-poly model was then brought into Maya where baked texture maps were checked and the mesh was prepared for future creative steps. Additional props such as the Pied Piper's backpack and flute were also created and arranged during this stage.

Fig. 15. Finalized Pied Piper character in engine, ready for animations.

5.3.2. Rigging

The rigs of the Pied Piper and children were created with the assistance of Epic's Animation and Rigging Toolset (ART) which comes with Unreal Engine 4. It is a plugin for Maya which contains a easy to use rigging solution and amazing animation tools (Maya Animation Rigging Toolset, 2016). The bones of the soon to be rig are placed in their appropriate locations and rotations with a mannequin rig. Once in place, the tool smooth binds the mesh to the bones, leaving the skin

weighting process to be painted by the artist. The painting skin weight process took very long, especially for asymmetrical meshes like the Pied Piper. After completion of the weighting process, the rig controls are generated and the character is done. Some additional setting needed to be experimented with to achieve proper dynamic bone movement and such.

We chose to include Nvidia PhysX APEX cloth simulation in our game to make the clothing even more believable and add dynamics. This was easily achievable with plugins and tutorials such as the one provided by Tales of Nalstone (Basic Unreal Engine 4 Apex Cloth).

Blendshapes were also used for facial animations instead of a bone based facial rig. These were created in Zbrush, applied in a specific way within Maya to be usable in engine.

Fig. 16. The Pied Piper mid-animation.

5.3.3. Animation

The animations were created in Maya using the Epic ART system used for rigging. THis toolset was chosen for its use of transform spaces over constraints, as Unreal Engine does not use constraints. This was particularly useful for animating the Pied Piper's instrument holding and playing. The flute was placed into the head's space and the hands were each placed into the flute's space effectively allowing the head to control both arms with no breakage and minimal

effort on the animator's part. The toolset also has a built-in bone picker and other efficiency improvements, making animating progress much faster.

Each character had several animations controlled in the engine by conditional animation blueprints. Depending on game events, the character animation would gradually blend into the pose of animation it would be playing next. The most common usage was switching between idling and walking without limb popping or looking awkward.

Animating was done at 30 frames-per-second with a standard keyframing workflow.

5.3.4. Texturing

Texturing for the characters was done in Substance Painter which allowed artists to paint a material directly onto the mesh and combined procedural texturing methods with hand painted ones. The tool was easy to use and had excessive tutorials (Allegorithmic).

6. Music and audio

Music is the driving force of *The Piper* experience, both for the children in-game and the player. The overall idea is that the piper lures children with pleasant and inviting music, so it was absolutely critical to get this right.

The village melody is the first thing the player hears upon waking up. It presents a joyful, hopeful tune that catches the ear. As the Piper lures the player out of the city, the music slowly builds in intensity until its climactic point at the city gates, where it explodes into a full fanfare. As the group of children approach the river just outside the city, the music quickly takes a turn for the worst, morphing into a dissonant cacophony. It rises the closer the children get to the rapids, and just before it can claim them, the melody swiftly returns to its original major key as the bridge materializes out of thin air.

Once the Piper guides the children into the forest, a new melody begins. This is something more mysterious and even mischievous, reflecting both the children's thoughts of sneaking out in the middle of the night, as well as the Piper's unknown malicious intent.

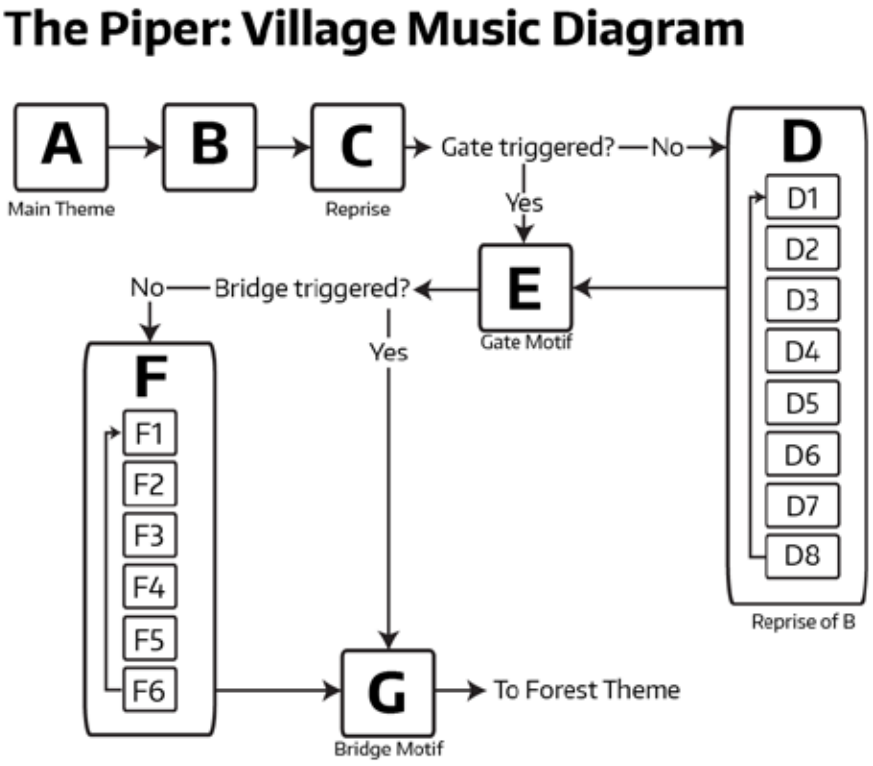

Fig. 17. Piper village music.

At the end of a forest lies a long rock tunnel. As the children enter, the forest theme begins to reverberate until the music fizzles out into a washed, ambient drone. It slowly builds in intensity as they're drawn further into the cave, and eventually cuts off into nothing as the player loses control. This is the last music heard before the narrator wraps up the story over complete silence.

The entire game was composed in such a way that we could switch music on the fly at any point while keeping the score sounding natural. The village is the most complex implementation of this idea (see Fig. 17). It starts off with the A section, with the lone piper playing his flute, poking his head into the player's room. The string section and light percussion slowly introduce themselves as he leaves. This flows straight into the B section, which brings in the brass section along with a new chord progression. After this it leads into C, a reprise of the original A section theme, albeit with more instrumentation.

Up until this point, the music has been completely linear, and will always play in this fashion regardless of the game's state. However, this is where the interactivity is brought in. If

the player reaches an in-game trigger near the end of the village by the end of section C, the music automatically transitions into the E section as the gate opens. If not, Wwise will loop a special D section over again until that condition is satisfied. Though this D section is essentially one long loop, it's broken up into eight individual 2-bar chunks. This approach may seem clunky, but it allows us to smoothly switch back to the gate theme (E) within a two bar interval. No matter which of these subsections is playing, each will transition musically into E without any pops, cracks, or other artifacts.. After the E sections finishes, a similar concept happens in the F section; we have six 2-bar loops that repeat until the player nears the bridge. Once they get close enough, we wait until the end of the current 2-bar chunk of music to transition into the bridge theme (G). Two linear sub-sections play before launching into a final loop, in which the flute plays a very soft pattern that transitions naturally into the forest theme once it is triggered.

This adaptive music setup is far from perfect, but it's still functional and mostly elegant. It ensures that players will never be met with total silence, whether they decide to spend thirty seconds in the village square or thirty minutes.

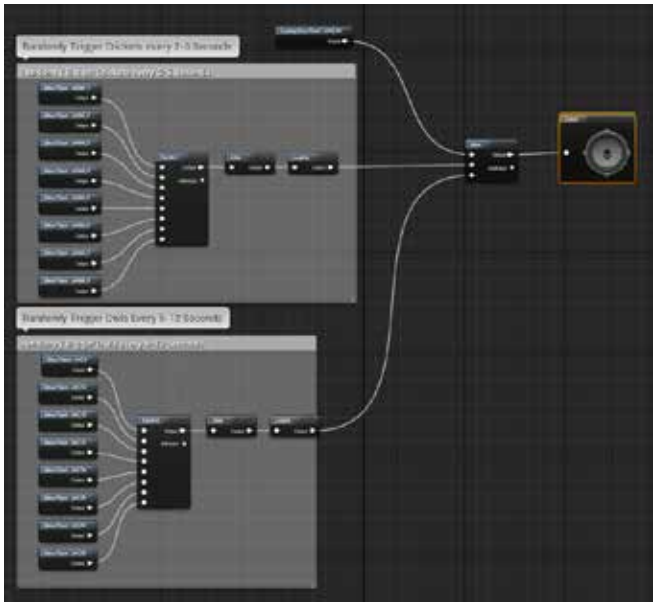

Fig. 18. Sound cues for the forest.

6.1. Sound Effects

Since the music is the primary focus, the sound effects are less emphasized. There are a few notable exceptions to this, but for the most part we kept most other sounds low key.

The entire experience plays back the player's footsteps on several surfaces, including cobblestone, dirt, wood, and leaves. They're mixed lightly as to not overpower other important effects, and as such only stick out during quieter periods in the soundtrack.

The village features a light wind blowing throughout, to serve as an ambient bed. The city gates are the most notable sound in this first segment. A long creaking rings out as the gates open, revealing the outside of the city and a bright yellow moon.

The river is the main exception to the “low key” rule, as the rapids need to sound as intimidating as they look. This is achieved using two separate audio components in Unreal, superimposed on each other. The first emits a field recording of a river from further away. It gives a general sense of a large body of water. As the player gets closer, a second loop fades in. This one is recorded much closer to the water so individual splashes, drops, and other elements of the river can be picked out. Rather than performing a simple fade with just one of these sounds, the combination of both helps sell the effect to a much greater detail.

Once the player is close enough to the rapids, the bridge forms from a number of surrounding rocks. Even though rocks floating in the air wouldn’t physically produce much sound, several samples of reversed rock falls and impacts play, giving the illusion of the stones fusing together to form the bridge.

The ambience in the forest is a bit more involved. A Sound Cue was set up to give the forest a more dynamic and randomized soundscape of its own, rather than just a single loop. Underneath everything lies a wind loop, giving a sense of slight movement in the trees above. A randomizer chooses from a set of cricket sounds, and triggers them every few seconds. Similarly, another randomizer selects various owl sounds to play at longer intervals. This more complex setup gives much more variety and interest than a stagnant loop, and helps the forest seem a little more organic.

6.2. Narration

Since the player starts out halfway through the story, we needed a way to communicate the first half in some way. We considered hinting at it several times during gameplay, but we decided to frontload it with a storybook sequence instead.

Our advisor put us in contact with Helen Lisanti, a BBC-trained voice actress, to perform the intro narration for us. She did a fantastic job, and her haunting voiceover sets the tone of our game from the very beginning quite effectively.

7. Project promotion

7.1. Web site

A basic Weebly website was created to serve as a homepage for our project. It contains a FAQ, bios of each team member, and a way for visitors to download a build of the game for testing. The website is located at www.thepipergame.com.

7.2. Twitter

A Twitter account was created in order to keep potential players up to date with production of the game. Under the handle @ThePiperGame, we posted in-progress screenshots, renders, .gifs and other assets to showcase how the game was shaping up. We were able to tag the account in our personal Twitter posts to raise awareness of our game, and show the followers on our personal accounts what we were working on.

8. Conclusion and recommendations

8.1. Scope

The scope of *The Piper* changed drastically over the game's development, generally decreasing as time went on. We initially hoped to include an alternate ending, where the player escapes, defeats, or stops the Piper. The first idea to gain traction involved a deaf child who would be immune to the Piper's spell and could be befriended through a series of puzzles. Befriending the deaf child would lead to an alternate ending where the player is rescued by their newfound friend.

This plan was found to be too complex, and was abandoned. We replaced the idea with a less intricate puzzle, where the player must locate a gold coin and deliver it to the Piper. This was significantly simpler, but rapidly approaching deadlines forced us to abandon the alternate ending altogether.

Neither of these ideas was ever implemented to any extent. We agreed to first focus on the main game, and only start developing the alternate ending(s) once we considered the rest of the experience complete. This proved invaluable, as it let us easily rescope the game when

needed. We avoided wasting time partially implementing optional features only to waste more time removing them. We recommend to any future teams that they agree upon what constitutes the core game, and ensure that it is fully developed before beginning any optional endeavors.

There were also some cut features that were not initially considered optional, and were in fact partially implemented. The game, which ends in a cave sequence, initially included an additional sequence set atop a cliff. The amount of additional art assets required for this sequence proved infeasible, and it was agreed that the game would have to end earlier. Fortunately, we identified the cave sequence as an easy cut off point. However, this was arguably a stroke of luck; the cave sequence was never designed to be a fallback ending for the game. We recommend that future teams identify points partway through the game that could be repurposed into proper endings with minimal effort in the event that the planned ending proves to be impractical to complete.

8.2. Wwise issues

Though Wwise was certainly useful due to its powerful music engine, it wasn't without problems. It took the team over a month to fully integrate it into the Unreal Engine, and it continued to present a number of obstacles throughout most of development.

The primary source of frustration stemmed from how Perforce interacts with Wwise: namely, how it doesn't. Unreal is set up to work with Perforce exceptionally well, but neither knows how to handle the inclusion of Wwise. Even though all Wwise-related folders and media are stored in the same directory as the Unreal project, they are not visible in-engine. Additionally, this means they are ignored by Unreal's internal integration with Perforce, and must be dealt with separately in the P4V client.

Over time, a workflow was developed that worked relatively smoothly: the composer would manually check out any Wwise-related folders and files (including the separate Soundbank Definition File) from P4V before making any changes to the project. Soundbanks must be generated through Unreal, rather than through Wwise normally. These are generated automatically based on whatever audio objects are attached to the specific event. After all changes are made, the project is saved and closed. Updates to source control must be applied through P4V, ensuring any music-related changes properly update on the server. This workflow was far from perfect, but it was workable.

A plug-in version of Wwise for Unreal was in development at the time of *The Piper's* development. We were able to successfully integrate Wwise into a Unreal Engine 4.11 project with extreme ease due to it being a plug-in. We were able to reap the benefits of the new update's optimized rendering for VR just in time before PAX East.

8.3. Flute issues

The original intent with the music involved recording an actual flute performance for maximum authenticity. Though this would've been great in some respects, it ultimately didn't work out.

The biggest downside to "live" recording would be the finality of it. Once a part is recorded, it stays exactly the same. Most of the game's music went through several iterations and phases during development, as plans changed for how the game was structured. If something in the flute melody needed to be altered, or a note had to cut off earlier in order for another music cue to work, it would've been difficult — if not impossible — to make the required changes.

We eventually decided to use a professional-quality sample library instead: 8dio's Claire Flute VST. It features a very detailed sampled flute, with true legato in between held notes. It was certainly worthy of being performed by the Pied Piper.

Integrating this new virtual instrument still took a significant amount of work. Up until this point, a flute in the Kontakt Factory Library was used as a temporary solution. It was far from acceptable, but was the best thing we had access to for much of the project. We didn't expect to simply throw in the new sample library and expect everything to work; the performance had to be completely reworked in order for this new flute to sound good. In some cases, the melody had to be completely changed to accommodate. All things considered, it saved much more time when compared to the original plan to record. Not only that, but the VST allowed the melody to be flexible. Though it took a fair effort to fix up timing and performance virtually, we ended up with a great sounding result.

8.4. Design issues

We had a main priority of making this experience last 3 to 5 minutes. With the time constraint in mind, the level design had to accurately reflect upon that. As such, we decided to make the streets of Hamelin simple to navigate, yet appear to expand past the player's capable range of interaction. By doing so, the illusion of the in-game city being larger than it actually is, similar to that of a movie set, exists. Not only did we want the player to easily navigate to where they needed to go to progress the story, but we also wanted the streets and street openings to be laid out in a way that shows off the assets we made.

The initial level design was laid out using UE4's BSP brushes, which are essentially primitive geometry building blocks that are used as placeholders until final art is brought into the engine. After we were happy with the design and the feel, the modular building pieces replaced the BSPs with the Tudor style houses. Minor changes to the level design were made here and there, but overall, the original city layout design remained the same.

The design outside of the city, however, changed almost completely from our initial conception to the final product. Originally, we intended on the player following the Pied Piper and the rest of the other children into a cave, where the walls behind them would crumble. They would be transported to a mountaintop, where they would see a Candyland-esque kingdom before them, past a rainbow bridge and over the clouds. Upon crossing the bridge, Candyland would disappear, and they would all fall to their death (See Fig. 20). This portion of the game was almost entirely scrapped due to scope issues.

We decided on ending the experience at the cave. Instead of walking into a dark cave, they would enter a brightly lit (although not so bright that it would hurt the player's eyes), otherworldly portal inside of a cave. This would leave the ending open to the player, as to not suggest life or death to the children.

8.5. Integration and source control

Our team management and communication method was Slack, as stated earlier. The service works on the internet, on a desktop, or on a cell phone, so we were always able to communicate. Slack has additional features when integrated with other services like Google Docs, Dropbox, Trello, and GitHub. Most of the integrations were helpful at first but quickly came into disuse as we abandoned the services associated with them. Different channels were used, but, as stated previously, most communication was done on the #students channel, only ever using another channel if we needed professor input. Overall, Slack proved vital to development process of The Piper, and we would not have had a reliable communication method without it. We highly recommend future teams to use Slack.

We set up a Trello board early on in the project. Trello is an online application to help manage, assign, and keep track of tasks. It does this through a number of lists on which individual task cards can be stored and moved around. By setting up lists and cards for tasks that needed to get done, we could have a solid idea of who's working on what, and how much is left to do. Additionally, Slack provides an integration with Trello, and a separate #task-management channel was created specifically for this purpose. However, this method of task management quickly fell out of use in favor of other methods. The team mainly kept up to date through both the usual Slack channel and later a master task list, a single Google Spreadsheet that listed everything left to accomplish. While Trello may work exceptionally for other teams, it didn't turn out to be a great fit for us.

Our first source control system utilized Git. While the tech team was already familiar with Git, we found it unsuitable for this project. Although Git technically supports any file type, it is designed with the assumption that it will be primarily used for text files. As such, it freely allows multiple users to edit the same file concurrently, knowing that it will usually be possible to combine their changes later. Even when the overlapping changes are ambiguous, Git provides an interface which allows for these conflicts to be easily seen and dealt with. However, these conveniences do not apply to binary files; as The Piper was programmed mostly in Unreal's Blueprint system rather than C++, this proved troublesome. Git's lax rules on concurrent editing led to frequent and unresolvable conflicts with file changes.

These issues with Git necessitated a switch to Perforce, a system with a concept of "checking out" files so that other users may not simultaneously edit them. Perforce was more difficult to set up than Git (see Appendix TODO for a detailed explanation of the process), but proved worthwhile. The inability to concurrently edit files was inconsequential; there was very rarely a need to do so. The mechanic of locking files for other users to prevent merge conflicts was far more useful and greatly expedited production of the game. In addition, Unreal Engine natively supports robust integration with Perforce, allowing us to use most of its functions within the editor itself. We highly recommend teams using Unreal Engine, especially if their projects are primarily Blueprint-based, use Perforce for source control.

We were originally given an OSVR (Open Source Virtual Reality) HMD courtesy of WPI. While the idea of the headset is fantastic with its low price and sufficient technical specifications, getting it to properly integrate into our Unreal Project was a hassle. Currently, Epic does not offer a native OSVR plug-in, resulting in us having to compile and add it to our engine. We were successfully able to get the HMD working in editor, however when it came to making a build of the project, we were given multiple errors. After several weeks of trying to get it to properly work, contacting customer support, and trying multiple methods of integration, we abandoned OSVR and were able to secure an Oculus Rift HMD from our good friend and WPI alumni Ichiro Lambe of Dejobaan Games. Integration of the Oculus Rift was immensely easier, as its documentation is clear and easy to follow. It should be noted that with the release of UE 4.12, OSVR native support will be offered (Sensics, 2016).

8.6. Team motivation

For the majority of the MQP timeline, we worked effectively and consistently on the development of The Piper. However, the long development led to the development of a few motivation lulls, where progress was slowed. In A-term, we were progressing rapidly with the project being so new, exciting, and, as previously mentioned in Scope, vast. We knew there would be a lot of work, but we were resilient and optimistic in finishing the game in C-term to test the game and polish it. We made great strides that term and had a greybox level that the player could walk all the way through.

Entering B-Term, our momentum halted. We concentrated on features which were not important or got hung up on artistic elements. We made progress, but not as much. Then we lost Ben Korza from the team in C-term and ran into difficulties within the team and project which set us back.

We attempted to keep our motivation for the project up with thinking of new features or simply making an effort get tasks done. The Asset Priority List we created definitely helped to keep track of how much needed to get done and who needed to do it.

Our recommendation is to, again, properly scope, have a team management system such as an asset priority list early on, celebrate successes, and stay on top of what needs to get done and by whom.

8.7. Team structure

Our team was divided into three groups: a two-man programming (“tech”) group, a two-man art group, and a single music/audio person. Early in the project, we followed a fairly strict and regular meeting schedule, with multiple full-team meetings each week. The art and tech teams also regularly held meetings amongst themselves to work on their aspects of the game. As the project went on and focus shifted from design to development, the regimented schedule was gradually phased out and replaced with a more as-needed meeting system.

Halfway through the project, Ben Korza, one of our two programmers, graduated from WPI and left the team. Knowing this in advance, the tech team took special care to ensure that Will Frick, the other programmer, would be well-versed in Korza’s contributions. This proved invaluable when revisions to the game design required changes to existing technology. While mid-project changes to team structure should obviously be avoided if possible, we recommend that these changes be carefully anticipated by the team. Leaving members should ensure that their counterparts are able to understand their work, in case it needs to be changed later on.

8.8. Engine issues

Our project was negatively impacted by a handful of bugs present in Unreal Engine. This was exacerbated by the fact that our integration with Wwise required us to freeze our engine at version 4.9, leaving future bug fixes unavailable.

As mentioned in Methods, the instability of Unreal's experimental 3D Widget feature presented severe problems for our pause menu in virtual reality. After spending a large amount of time attempting to engineer a workaround for the issue, it was decided that the feature would have to be abandoned. This is undesirable, naturally, but we feel it was the correct choice, especially as it occurred very close to our deadline.

8.9. Flocking issues

Unreal Engine provides built-in support for various technologies, such as flocking navigation. It can be tempting to design one's own interpretation of such technologies, in the interest of practice, customization, or simply out of a desire to fully understand how the technology works. However, this proved to backfire on the team. Our custom-built implementation of child flocking, while significantly more customizable than Unreal's RVO system, proved to be inefficient in terms of computing time and ultimately had to be removed. Upon implementing RVO in its place, we regretted not doing so in the first place. This would have saved development time and made testing the game significantly easier. We recommend that future teams carefully consider existing solutions to problems before engineering their own.

8.10. Pacing system issues

Due to the choreographed nature of the game, the timing of individual events was extremely important. While the various triggers used to enforce this timing were reliable and precise, they became difficult to keep track of as the game grew. We never established any sort of centralized or standardized pacing system. When new additions to the game required intercepting older events, it was sometimes extremely difficult to locate where the interception should take place. Most set pieces were activated by zones which sensed overlaps with the player or Piper; in some cases, these zones were dedicated to the purpose of sending commands to those specific set pieces. In other cases, zones carried multiple purposes; for instance, there is a zone near the river that controls the Piper's movement and animation, as well as the bridge's animation. These inconsistencies impeded the game's scalability. We recommend that, in games which heavily rely on scripted pacing and choreography, the developers implement a standardized system for how pacing is controlled, such as a centralized pacing controller object.

8.11. Documentation

The Unreal Engine 4 documentation is vast and in-depth, providing information on nearly every aspect of the engine in great detail. Nearly anything we wanted to do for the game had documentation located in the engine notes or was located in EPIC's forums and answers section. There are also many groups of developers who have knowledge of the engine, and the EPIC developers are even able to be contacted fairly easily. We made use of the documentation a lot to see the best way to perform a task.

8.12. Testing

We regret that we had little time for formal testing. We originally planned to run many formalized playtest sessions, but this fell through as deadlines approached. While we each did receive some informal feedback from friends and family, we never found the time to run true playtest sessions. Future teams should be careful to budget their time to accommodate testing.

8.13. Stretch goals

If we were to be given an extended timeline, all aspects of our current project would be taken and expanded upon. Our current level design would be just the beginning - adding a maze portion to the city map would provide a fun experience for the player as he or she would need to follow the Pied Piper in order to find their way out of the city. This guidance could be visual, but ideally it would be purely audio, as the player would have to navigate down certain hallways that they could hear the magical flute coming from.

With additional time, we would likely revisit our scrapped ideas for an alternate ending. Both the "deaf child" and "coin puzzle" concepts were extensively planned before they were abandoned, and either would likely be a profound improvement to the experience.

We also wanted to dedicate additional time to making the city feel more lively. The Pied Piper and children interacting with the environment would have contributed significantly to the feeling of the game being an actual world the player is put into. Similarly, more detail in the environment as well as environmental animation would have been nice to include. Character animations could have also been expanded upon, allowing for the player to view a character as not just a computer driven entity.

We would have also liked the music to act as a more integral part of the experience. As of now, it feels like the music is just a bed being played underneath the game, rather than being inherently tied into the experience. More interactions in both the dynamic music track itself, and more interactions between the soundtrack and the game world, would've brought the experience together into a more cohesive whole.

Fig. 19. PAX Attendee playing *The Piper*

8.14. PAX East 2016

Penny Arcade Expo (PAX) East 2016 took place during the dates of April 22nd through April 24th. We were able to secure a spot in the WPI IMGD booth, and show our project all weekend long. All together, we estimate roughly 300-400 people strapped on the Oculus Rift and played through our interactive cinema.

Because PAX is a consumer targeted show, many attendees were unfamiliar with VR. When they saw *The Piper* being presented on the Rift, people would line up just for the sole purpose of trying virtual reality. It showed once they took the headset off that a majority of the players loved what they saw. Their faces would light up, and they would say something along the lines of “that was awesome!” Although we did not have any extensive testing prior to PAX East, this event served as a major stress test to see if there were any bugs that we did not catch in the weeks prior. We were able to catch a few bugs, but overall, the weekend proved to be a success.

Works Cited

Browning, Robert. "The Pied Piper of Hamelin." Poets.org. Web. URL: https://www.poets.org/poetsorg/poem/pied-piper-hamelin. Retrieved 21 Apr. 2016.

Sloan, Becky and Pelling, Joseph. *Don't Hug Me I'm Scared*. YouTube. This Is It Collective, 29 July 2011. Web. URL: https://github.com/audiokinetic/WwiseUE4Integration. Retrieved 9 December 2015.

Reynolds, C. “Boids (Flocks, Herds, and Schools: A Distributed Behavioral Model)”. Web. URL: http://www.red3d.com/cwr/boids/. Retrieved 9 December 2015.

Epic Games. “Creating 3D Widgets.” 5 January 2016. Web. URL: https://docs.unrealengine.com/latest/INT/Engine/UMG/HowTo/Create3DWidgets/index.html. Retrieved 2 April 2016.

Epic Games. “Scalability Reference.” Web. URL: https://docs.unrealengine.com/latest/INT/Engine/Performance/Scalability/ScalabilityReference/index.html. Retrieved 2 April 2016.

History Channel. This Day in History. Web. URL: http://www.historychannel.com.au/classroom/day-in-history/663/pied-piper-lures-130-children-of-hamelin-away. Retreived 2 April 2016.

Lee, Perry. “Modular Level and Component Design.” November 2002. Web. URL: https://udn.epicgames.com/Three/rsrc/Three/ModularLevelDesign/ModularLevelDesign.pdf. Retrieved 4 April 2002.

M-H, Oliver. “3D Modeling & Texturing.” August 3, 2014. Web. URL: http://oliverm-h.blogspot.com/2014/08/ue4-localized-post-process-effects.html. Retrieved December 6, 2015.

Pixologic. “Riot Games featuring Willem Van Der Schyf.” 7 Oct. 2015. Web. URL: www.youtube.com/watch?v=NVP88LfM4jA. Retrieved 8 April 2016.

Nicoletti, Julio. “Seamless Texture with Maya, xNormal, Photoshop, nDo2.” 27 December 2013. Web. URL: www.youtube.com/watch?v=6PSrQPizlTY. Retrieved 12 December 2015.

Sensics. “With New Native OSVR Support, Sensics Brings the Epic Unreal Engine Experience to the Next Level.” 15 March 2016. Web. URL: http://sensics.com/with-new-native-osvr-support-sensics-brings-the-epic-unreal-engine-experience-to-the-next-level. Retrieved 15 March 2016.

"Getting Started with Marvelous Designer." Marvelous Designer. Web. URL: http://www.marvelousdesigner.com/learn/lessons. Retrieved 18 Nov. 2015.

"Topology." Polycount Wiki. 2 Aug. 2015. Web. URL: http://wiki.polycount.com/wiki/Topolog. Retrieved 8 Apr. 2016.

“Maya Animation Rigging Toolset.” Epic Games. Web. URL: https://docs.unrealengine.com/latest/INT/Engine/Content/Tools/MayaRiggingTool/index.html. Retrieved 8 April 2016.

“Basic Unreal Engine 4 Apex Cloth.” Tales of Nalesone. 2 Apr. 2015. Web. URL: https://www.youtube.com/watch?v=uTOELBNBt04. Retrieved 8 April 2016.

“Substance Painter Texturing for Beginners.” Allegorithmic. 13 Aug, 2015. Web. URL: https://www.youtube.com/playlist?list=PLB0wXHrWAmCx994Cb7iRFSmupYHFw5DTx. Retrieved 8 April 2016.

Appendix 1. Perforce

The team struggled to acquire a Perforce server early on in development. We contacted Michael Voorhis of the Computer Science department who graciously agreed to run a server on an old computer in his office. He was able to quickly get the server in a working order and able for us to use.

Here are a few findings (in no particular order) that may help future teams when using Perforce:

- Integration with Unreal Engine 4 is superb and easy to set up.
- The Perfoce server must have a typemap, so UE4 and Perforce know how to deal with certain files.
- Having the server physically on WPI's campus and able to SSH connect to it is ideal.
- Users must have P4V client open along with UE4 to use source control.
- Checking in large file may take some time, but it is still being pushed.
- Do not change your workspace in P4V midproject.
- Do not use streaming depots for UE4 projects.
- Sometimes, the .uproject file gets checked out then in and set to read-only. It is necessary to set this to writable manually in the file browser.
- Check in changes often so others may pull and stay updated.
- Reverting changes is on a per file basis, and is extremely useful.
- Never check out the whole project

Appendix 2: IRB Consent Agreement

Informed Consent Agreement for Participation in a WPI Research Study

Investigator: Brian Moriarty, IMGD Professor of Practice

Contact Information: bmoriarty@wpi.edu, 508 831-5638

Title of Research Study: _____

Sponsor: WPI

Introduction: You are being asked to participate in a research study. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risks or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation.

Purpose of the study: The purpose of this study is to obtain feedback on the MQP project in order to facilitate design improvements and find/address operational bugs.

Procedures to be followed: You will be asked to play a brief game lasting less than ten minutes. Instrumentation in the game software will anonymously record your activity during play. After completing the game, you will be asked to complete a brief, anonymous survey describing your subjective experience.

Risks to study participants: There are no foreseeable risks associated with this research study.

Benefits to research participants and others: You will have an opportunity to enjoy and comment on a new game under active development. Your feedback will help improve the game experience for future players.

Record keeping and confidentiality: Records of your participation in this study will be held confidential so far as permitted by law. However, the study investigators and, under certain circumstances, the Worcester Polytechnic Institute Institutional Review Board (WPI IRB) will be able to inspect and have access to confidential data that identify you by name. Any publication or presentation of the data will not identify you.

Compensation or treatment in the event of injury: There is no foreseeable risk of injury associated with this research study. Nevertheless, you do not give up any of your legal rights by signing this statement.

For more information about this research or about the rights of research participants, or in case of research-related injury, contact the Investigator listed at the top of this form. You may also contact the IRB Chair (Professor Kent Rissmiller, Tel. 508-831-5019, Email: kjr@wpi.edu) and the University Compliance Officer (Jon Bartelson, Tel. 508-831-5725, Email: jonb@wpi.edu).

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

_____ Date: _____

Study Participant Signature

_____

Study Participant Name (Please print)

_____ Date: _____

Signature of Person who explained this study

Appendix 3. IRB Protocol

- **Purpose of study**
  - To obtain playtest feedback in order to locate/address operational bugs, and to identify opportunities for design improvement.
- **Study protocol**
  - Participants are provided a computer on which to play the game. Investigators observe participants during play as instrumentation in the game software collects activity metrics. Afterward, participants are asked to fill out a short survey to characterize their subjective experience.
  - **Opening briefing for testers**
    - "Hello, and thank you for volunteering to test our game. Before we begin, could you please read and sign this Informed Consent form? [Tester signs IC form.] Thank you. During your test session, the game will be recording a variety of metrics about your play activity. When your session is complete, we will ask you to complete a brief survey about your play experience. At no point during your play session, or in the survey after, will any sort of personal and/or identifying information about you be recorded. Please begin playing when you feel ready."
  - **Metrics to be recorded during play session**
    - Total playing time.
    - Order/frequency in which in-game locations are visited.
    - Time spent between/within each location.
    - Activities performed in each location (avatar/camera movements, interactions with virtual characters and objects, idle time).
  - **Questions for post-test survey**

- What are the rules and objectives of the game? How did you discover them?
- How difficult does the game seem?
- Did it consistently hold your interest?
- Did anything about the game seem confusing or obscure?
- Did any aspects of the game stand out as particularly effective or ineffective?
- What would it have been good to know about the game before you started playing?
- How would you describe the game to someone who has never played it?
- Do you have any additional comments or questions that you would care to share?

- **Hazardous materials/special diets**
  - No hazardous materials or special diets are involved in this study.