Maternal Instinct

Interactive Media and Game Development and

Computer Science

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:

Nicholas Defossez _____

Alex Quartulli _____

John Tordoff _____

Kimani Gyening _____

Advised By:

Professor Britton Snyder

Professor Keith Zizza

Professor Robert W. Lindeman

Abstract

Maternal Instinct is a third person game that explores using directional sound as the main mechanic for navigation, game mechanics, and storytelling. Where most games focus on visual cues to guide the player, *Maternal Instinct* relies on audio cues. To this end, visual cues alone should not be sufficient to play. This game was created as a Major Qualifying Project for both Interactive Media and Game Development and Computer Science majors. This report discusses the intended design goals *Maternal Instinct*, the process by which *Maternal Instinct* was developed, challenges the project team faced during the development process, how *Maternal Instinct* was received by players, and the team's reflection on the overall direction the project had taken.

Acknowledgements

We would like to thank our advisors, Britton Snyder, Keith Zizza, and Robert Lindeman, for their time and effort in guiding this project. We would also like to thank Sasha Abdurazak for providing a written narrative introduction sequence and voice acting in accordance with the narrative.

Table of Contents

List of Figures

1. Introduction

Maternal Instinct is a third person game designed to force players to navigate primarily aurally, instead of navigating visually as most games have trained them to do. The main character is a tribal huntress, who must travel through a dark forest, braving traps and wild animals, in order to rescue her son from a group of kidnappers, intent on sacrificing him to appease a god.

Our main design goal was to force the players to use their ears, rather than their eyes in order to navigate the dark forest. Additionally, as a Computer Science/Interactive Media and Game Design double MQP, we had a goal of a tech-heavy component. We decided to create a procedurally generated temple for the mother to explore. With all assets being placed in the world at runtime, the team created a unique temple for each player to explore.

2. Gameplay

Maternal Instinct is a third person action game, based around training players to respond to audio cues rather than visual ones. The main character is armed with a simple spear, and she must defend herself by listening for panthers, dodging, and then striking back, rather than charging straight into a fight.

2.1. Win Condition

A player wins once he or she has traveled through the forest, defeats the great panther god which awaits at the foot of the temple, and reunites the mother and son once more. In order to do this players must fight their way through many panthers along the trail using positional sound to listen for hazards.

2.2. Setting

Maternal Instinct is set in a jungle, based on the environments native to South America. The game takes place in the past. The mother's spear is used primarily as a hunting tool as was common amongst Aztec society.

Though *Maternal Instinct's* setting is inspired by South American culture, it does not take place in any true Earth location or time period. This allowed us to have the freedom that we needed for the story, while still drawing on familiar sights, sounds, and cultures that do not often get featured in video games. It was important to us that we do not make yet another game featuring a stereotypical white male action hero.

We made an effort to stay as authentic as we could to the source material, while giving that material our own unique flavor. As shown in Figure 1, we did a lot of research into various South American cultures, dress, and weapons.

Figure 1: Reference image of Mayan Spear Heads. [4]

2.3. Player Experience

We were focused first and foremost on forcing our players to navigate through the jungle by ear, listening for audio cues and reacting based on those sounds, rather than navigating by sight, and reacting to visual stimuli.

2.4. Narrative

Maternal Instinct tells the story of a mother trying to get her son back. On a regular hunting trip the two capture and kill a panther, which they plan to sell at a nearby village. When the mother and son arrive, the villagers greet them warmly. That evening, the village is attacked by panthers. The villagers realize their guests had killed a sacred animal, stealing away the son with intentions to sacrifice him to

the panther god. The mother quickly hurries after the angry villagers and - alone - braves the treacherous jungle at night, intent to let nothing happen to her son.

2.5. Gameplay Challenges

In traveling through the jungle at night, the mother must not only navigate primarily by ear, but also stay safe in the dark. Panthers prowl in the area, but the path she is traveling is treacherous in its own right. The path the villagers have taken is rife with old, falling trees and unstable cliffs.

2.5.1. Navigation

While the jungle at night has very low visibility, the mother is able to locate her son through positional sound. As she chases down the villagers, the mother hears her son calling out to her, or villagers talking up ahead. These audio cues direct the mother towards her kidnapped son.

2.5.2. Panthers

The panthers are the most frequent obstacle standing between the mother and her child, and one of the most dangerous. They stalk her just out of sight, and pounce from the darkness when she drops her guard. Fortunately, the panthers make a growling sound as they stalk around the mother, and each time they pounce it gives a distinct audio cue for her to react to. If the mother dodges in time, the panther sails harmlessly by, and becomes stunned momentarily so that the mother can strike at it with her spear. If not, it claws her and then runs back out of sight, once again stalking, and preparing for another pounce.

2.5.3. Falling Trees

Certain dead trees in the jungle will make a loud cracking sound as they sway in the breeze, signaling to the mother that they might fall down and crush her. If she is careful she can avoid these hazards, but if not she might find herself crushed. These hazards appear intermittently throughout the

forest, and require the player to pay close attention to the soundscape surrounding them to survive. Figure 2 shows a player who has successfully avoided a falling tree trap.

Figure 2: Mother standing in front of a fallen tree that she successfully avoided.

2.5.4. Falling Rocks

Finally, the mother must also be wary of the rocky cliffs along the path. The cliffs host unstable boulders which will roll down and crush the mother as she walks by. As these boulders begin to roll, the surrounding rock makes a distinct crumbling sound, alerting the mother to their presence and allowing her to avoid the traps.

2.5.5. Spitting Snake Traps

The temple the son is being brought to is for all manner of beasts, snakes included. Statues created to ward off trespassers were built in the beasts' image. Statues of snake heads line the walls of the temple, ready to spit poison. The statues spit poison quickly, but they alert the player by hissing as they spit. By listening for the spit, the player can avoid these traps.

3. Development Environment

Maternal Instinct was developed using the Unreal Engine, with the Unreal Development Kit 3.0. Art assets were generated using Autodesk Maya, 3DS Max, Adobe Photoshop, ZBrush, and SpeedTree. Sound files were recorded in the field, wherever possible, or sourced from online free sound libraries, and then modified using Reaper. Art assets were rigged and animated in Autodesk Maya, and 3DS Max. After early prototyping of concepts for procedural texture generation in both Unreal Engine and Unity, we decided upon Unreal Engine to base our game. Additionally, we knew that Unreal Engine had excellent sound support, which we would require for this project.

3.1. Engine

Early on, we did a brief test of both UDK and Unity, in order to determine which engine would be better able to support the procedural texture generation that we were hoping to accomplish. In order to conduct these tests, we procedurally modified a brick wall texture, sourced from an online tutorial on how to create a brick wall texture that did not seem to repeat itself, as a proof of concept for our eventual goal of generating a brick wall texture from scratch via shader code.

3.1.1. UDK

The shader written for UDK performed very well, with realistic looking bump mapping and lighting. We were able to color the brick wall by randomly shifting the positioning of a color texture, and blending that texture with a grayscale texture of the bricks. Using a similar technique, we were also able to blend in a randomly placed bump map of a crack on top of the walls, adding additional details. The goal in this test was not to procedurally create an entire wall, but instead we wanted to test how easily we would be able to apply various techniques to textures that we would create on the fly. Our goal was to eventually create the crack textures at runtime, and place them randomly onto the wall, as well as project randomized weathering effects onto the wall.

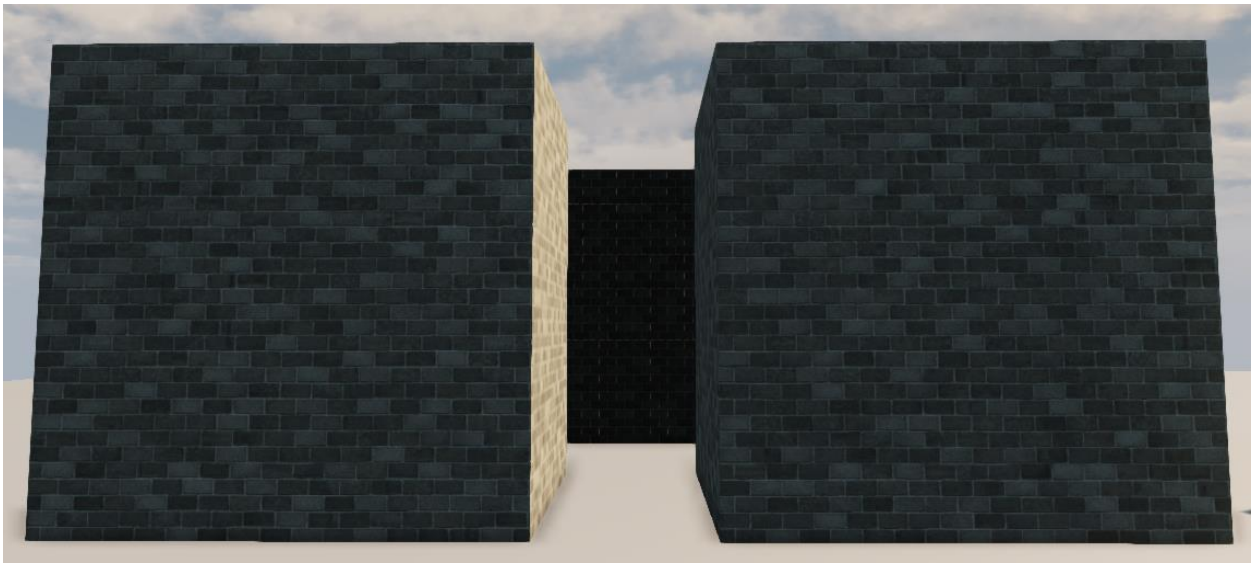

Figure 3: Example of randomized brick coloration in the UDK. You can see that the brick wall texture does not simply repeat across the two samples, but is instead unique on each.

3.1.2. Unity

Though Unity performed nearly as well as UDK did in this shader test, the lighting compared unfavorably to UDK's lighting engine. The bump mapping did not look quite as nice as it did within UDK, giving the texture an overall lesser appearance. Finally, the UDK provided support for the SpeedTree tool, which allows for the easy creation of trees and foliage. SpeedTree was essential for the development of our jungle environment.

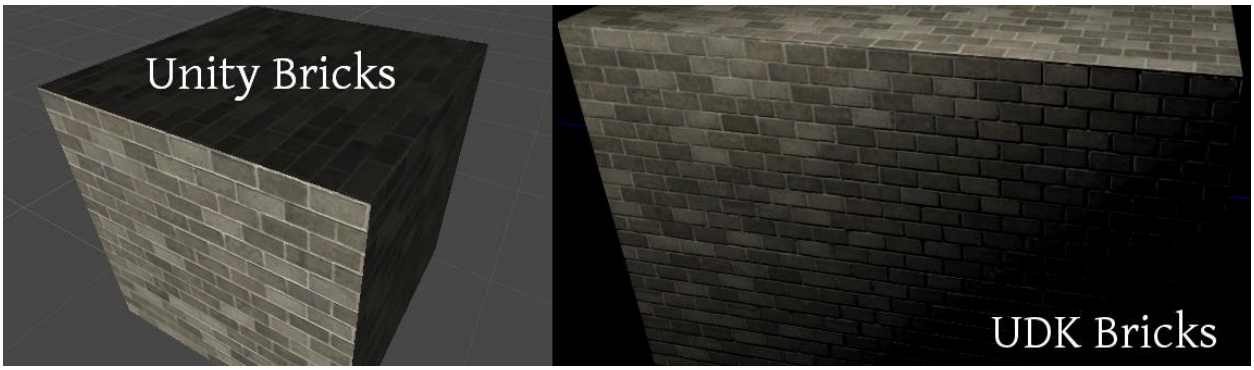

Figure 4: A comparison of the brick shader in both Unity and the UDK.

3.1.3. Engine Performance

Despite our initial tests, and though Unreal excelled in certain areas, there were other unwanted features. While the engine does have great sound support and excellent lighting, it is monolithic and as such it is very difficult to add or remove core features. Because *Maternal Instinct* is a third person action game, we did not require certain features provided in Unreal Engine which were tailored towards first person shooters. Early on, we spent time learning what features we did not wish to include from Unreal and removed those features.

Finally, though our initial tests showed promising results in working with UDK's shader language, it ultimately proved very difficult to use in the generation of full textures, parallax mapping, and bump mapping from scratch. While we were able to get a plain tiling brick wall texture generated very quickly, unfamiliarity with UDK's visual shader language rendered us unable to finish the shader before the end of the project.

3.2. Art Tools

As mentioned above, art assets were created using Autodesk Maya, 3DS Max, Adobe Photoshop, ZBrush and SpeedTree. High-polygon models were sculpted in ZBrush, then retopologized in order to reduce the polygon counts to amounts suitable to work in the game engine. For characters and some additional high-polygon models, diffuse and normal maps were initially generated in ZBrush, and were later refined in 3DS Max. Low-polygon models were generated in Maya and textures were gathered from online sources, such as cgtextures.com. Characters were rigged and animated using both Maya and 3DS Max. Most of the trees in the environment were generated using SpeedTree, which is fully integrated with the Unreal Development Engine. Many of the textures for environment assets in the jungle were gathered from cgtextures.com. Adobe Photoshop was used to refine some diffuse and normal maps, as well as to generate concept art for the game, and a sequence of splash images that corresponded to narrative dialogue during the game's introduction.

4. Sound Design

4.1. Imagining the Soundscape

One of the greatest challenges in the audio design for a game that is based around providing the player with directions almost solely through sound is to create a convincing soundscape for the game. The soundscape had to be strong enough to stand on its own in order to immerse the player in the jungle that they find themselves in while still allowing important gameplay sound cues to stand out and be easily identifiable by the player. In order to accomplish this, we used a number of references of what a thick jungle at night would sound like, both in the games industry and in the real world. Games such as *Amnesia: The Dark Descent, The Elder Scrolls V: Skyrim,* and the *Bioshock* series feature powerful and compelling environmental sounds with an immersive ambient track and countless periodically occurring or triggered sounds that help to engage players. With the help of these inspirations, we used a number of base ambient tracks for each of the primary environments in the jungle level to act as a sound floor, with the other sounds layered on top. Some examples of the other sounds used to increase the texture of the soundscape were the sounds of monkeys moving through the trees, flies and cicadas, rustling bushes and creaking trees and the use of four different sets of footstep sounds for each different ground material the player encounters. The soundscape needed to be carefully volume balanced as well. All of the important sounds for gameplay had to be easily recognizable from the ambient sounds, so that the player understood which sounds signified necessary immediate action. In addition, all of the sounds that comprise the jungle ambience are 3d positional sounds so as to train the player to see with their ears and identify the locations of these sounds.

4.2. Sound as a Guide

In addition to a strong and immersive ambience, *Maternal Instinct* is a game focused around using sound cues as the primary method of guiding the player through the level. We sought to

accomplish this with a few different categories of guiding sounds. First and foremost, due to the dark lighting of the level that restricts the players' view, we chose to guide the player through the level with positional dialogue that triggers as the player progresses. These voice over clips were recorded and placed throughout the level as a method of telling the story of the level as well as guiding the player through it and were carefully placed and edited so that they would properly sound as though coming from a distance in front of the player. The other form of guiding sound effect that we added were the sounds that are used to warn the player of an upcoming danger. These included sounds made by traps throughout the level, as well as environmental dangers. In addition, the cat enemies that the player fights have distinct sound effects used to warn the player when they are about to attack.

The most important aspects of these "guiding" sounds was to ensure that they could be easily heard and located in the level. This relied on not only ensuring that the sounds were loud enough to hear and distinguish from the background ambience, but also ensuring that the sound files' properties were right for the job and also to ensure that the UDK sound-cues' settings were properly adjusted.

4.3. Creating the Sounds

The sounds of *Maternal Instinct* were acquired in two different ways and were then edited for use in the game. Numerous environmental sounds were recorded out in the field using a Zoom H2 Microphone borrowed from the WPI Academic Technology Center. These sounds include footsteps, rustling bushes, creaking doors, crumbling sand and rocks, and more. Other sounds that could not be reasonably acquired by recording them personally were downloaded from Freesound.org under the Creative Commons 0 License. All sounds were then edited using Reaper in order to get them ready for the game.

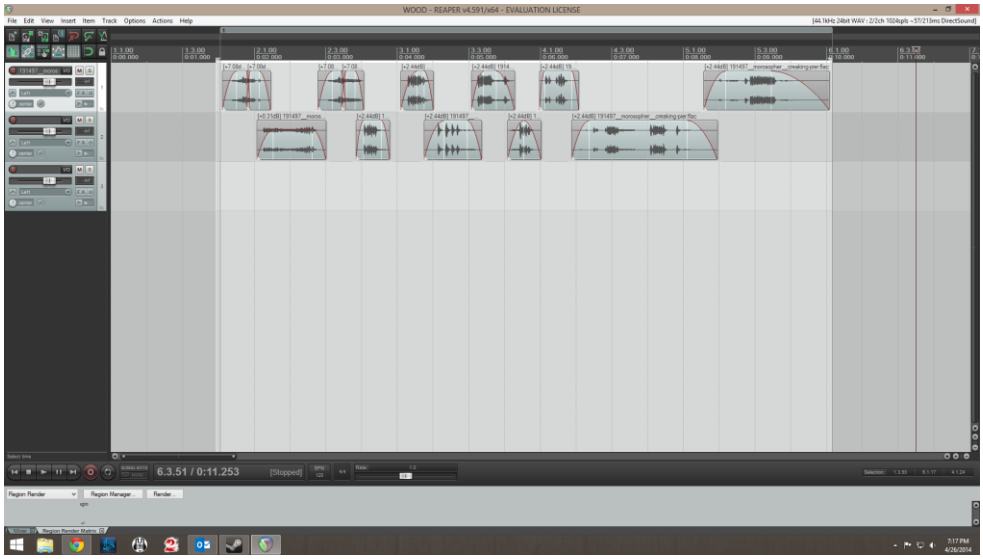

Figure 5: Editing footstep sound files in Reaper, an Audio Workstation by Cockos.

Particular attention was placed on ensuring that the sounds operated well according to their role. Looping ambient sounds had to be edited to ensure that they looped properly and that their volume level matched up with the other ambient sounds in the level, and the active "guiding" sounds were modified with Equalization to boost the mid ranges and reduce areas where the bass was too strong, as sounds are more easily located in a 3d environment when they have a strong mid-range component. Other effects such as reverb, compression, and pitch changes were added in Reaper as needed to simulate distance or adjust volume and improve quality. The sounds are all in the game industry standard 16 bit, 44.1khz sample rate and exported as either stereo or mono .wav files, for maximum quality in game.

4.4. Unreal Sound Integration

Once the sound effects had been created, the next step was to bring them into the game engine. One of the reasons the team had chosen the Unreal Development Kit as the game engine was that the UDK has an extremely powerful and customizable sound engine. The process for integrating

sounds into the UDK was twofold: First, the sound files were imported into UDK's content browser and were then categorized as SoundNodeWaves, a format that is comparable to a .wav file outside of the UDK.

Figure 6: SoundNodeWaves and SoundCues in the UDK content browser.

However, SoundNodeWave files only contain the basic sound file information, and can only be used as simple sounds in the UDK editor. Any more complex uses of sounds needed to be set up in a "SoundCue" object, which allows for more advanced uses such as positional sound, sound randomization and movable sounds.

The SoundCue editor allows the application of a number of different effects to SoundNodeWave files that are imported into it. It uses a visual scripting style similar to Kismet in order to accomplish this, and it allows the addition of multiple effects to sound files within the game engine itself, rather than having to export numerous sound files with these different effects from Reaper.

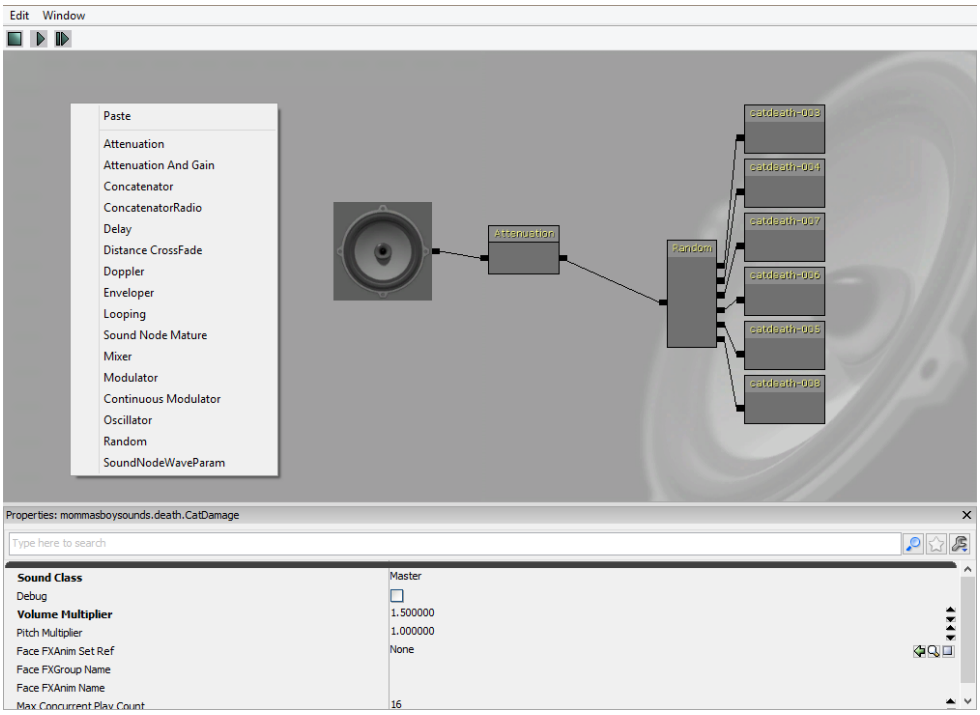

Figure 7: The SoundCue Editor, displaying adding the Attenuation effect and randomization effect for the Cat Minion's "OnHit" event sound.

Once all of the necessary SoundCues had been created for the sound effects, they were brought into the UDK map editor itself. The UDK editor uses a number of powerful tools for building a strong sound environment along with a visually compelling level. Sounds are brought into the level using a variety of ambientsound objects, which have varying attributes based on their purpose. Each sound object placed in the level can have differing settings, allowing for different attenuation ranges, low-pass filter attenuation, variable volume and pitch, and they can be spatialized. The combination of these settings allowed us to make each sound act properly in the environment and to make it easy for the player to hear the most important sounds and isolate their location in the environment, in addition to sounding excellent.

Another important tool of the UDK's sound environment was the use of ReverbVolumes. The UDK uses a number of "Volume" type tools, which create an area of space within the level that certain settings are applied to. A ReverbVolume isolates one such area and designates it as its own separate

sound space from the rest of the level. Any sounds inside a ReverbVolume are much fainter outside, and likewise, sounds outside the volume become much quieter when inside the volume. Also, reverb effects are applied to any sounds played inside a ReverbVolume, and the reverb settings can be applied to make the volumes environment sound like a particular type of environment, such as a forest, a wide open room, or a stone corridor. In *Maternal Instinct*, we used ReverbVolumes to separate the jungle segments of the level from the mountainous segments of the level. This allowed us to achieve a smooth transition from the jungle's ambience to the mountains, as hearing the harsh winds of the mountain whilst in the jungle would break immersion, as would hearing the flies, cicadas and other ambience of the jungle while on the barren mountains. The ReverbVolumes used for this had very conservative reverb settings due to being in a jungle, and a lengthy fade time for when the player left the jungle, so the transition between jungle ambience and mountain ambience was smooth and not a hard switch.

The final aspect of how we used the UDK's sound engine to our advantage was the powerful integration of trigger-able sounds and interactive moving sounds within the game editor. With the exception of sounds that were associated with the character controller or enemy pawns, none of the triggered or otherwise interactive sounds in the level needed scripting in order to be placed in the game level. From the dialogue that helps to guide the player through the level to the falling rock traps or falling trees, all of those sounds could be put in the game and tested all without having to change or compile scripts. The tool used to accomplish this was Unreal Kismet. Kismet is a powerful visual scripting tool used in the UDK, where objects in the game editor can be linked together with a variety of functions that can be both premade and custom.

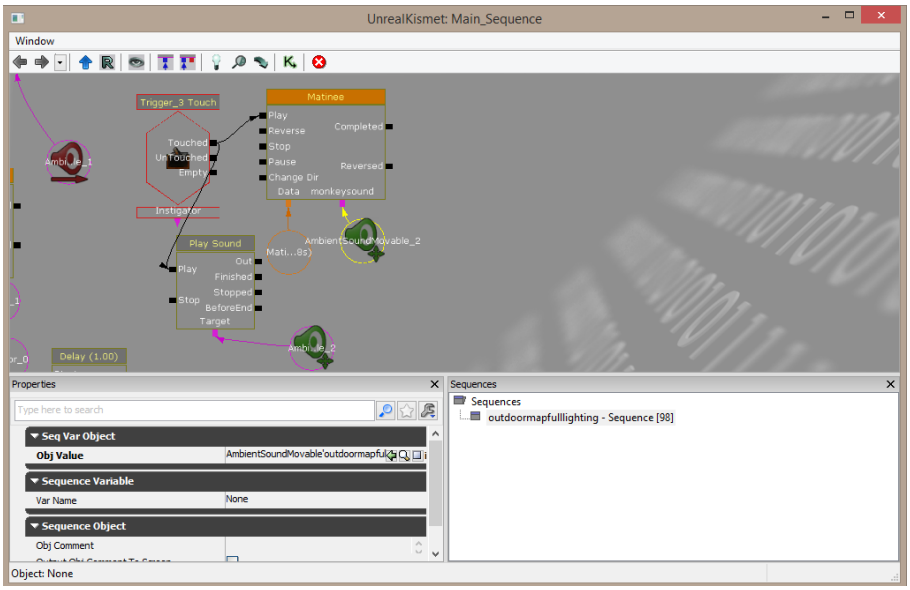

Figure 8: One of the falling rock traps scripted in Kismet along with its moving sound effects.

Using Kismet and Matinee (an Unreal tool that is used to create cut scenes or other moving objects) we were able to implement a variety of moving sounds throughout the level that were triggered either as the player moved through the level or on timers, which helped us to enhance the feel of the living jungle soundscape.

4.5. Narration

As a part of telling the story of the game, we chose to include a short intro sequence using a series of splash images and narration to tell the backstory of the game. This intro sequence was intended to provide a method of introducing the story without costing a large amount of development time like an in-game cut scene would. The narration for this intro was performed and recorded by Sasha Abdurazak and the script was also written by Sasha after he analyzed the story envisioned by the group. The use of an audio intro sequence in this manner as opposed to simply using text to tell the backstory helped to solidify *Maternal Instinct* as an audio focused game.

5. Artistic Goals

Overall we wanted to work in an art style that closely resembled the real world but wasn't required to be hyper-realistic. This decision was made in order to allow our artists to have some freedom in designing characters and environments. This section discusses the initial artistic objectives we sought to meet in creating our characters and environments, and the process by which we designed early and final concepts for both.

5.1. Character Creation

In accordance with our setting and narrative, we decided to establish our primary main character and protagonist as a mother and female hunter. The primary goal of our character artist was to establish a visual design of the main character that would most closely followed that description. Our character artist looked at reference images of women in various tribal cultures. Initially, these images were not restricted to old South American civilizations, and even included some African and Native American tribes. There were some potential issues regarding the physical attributes and cultural garb of the women in many of the reference images. In some images the women were not fully clothed and certain body parts were exposed, as are cultural customs in certain societies, but not in others such as the United States. In other images the females were scantily clad and some physical attributes were accentuated in a manner that serves to sexualize and objectify the female body (or could be easily interpreted as sexualization). While relative lack of clothing in other cultures may be understood by some, our country is currently living in a culture that's become increasing hyper-sexualized, to the point where any of lack of clothing may be interpreted as a sexual message, whether intended or not.

Figure 9: Reference of over-sexualized female characters. [8][9]

Our character artist felt that without clear indication of cultural significance, our surrounding audience may misconstrue any potential lack of clothing on our main character in the manner described, as many games have done intentionally. In order to avoid allowing for this kind of misinterpretation, our character artist based his designs on references of women who wore more clothes but whose clothing still had a tribal look and feel to them. Examples of this style include Korra and Korra's mother from the *Avatar: The Legend of Korra* series as shown in Figure 10.

Figure 10: References of women who are not sexualized. [2][3]

Although we drew some of our main character's initial design from the second character shown in Figure 9, much more of the design was influenced by the characters in Figure 10. The characters in the latter figure wear longer clothes because they live in a colder climate, so we shortened the clothing on our main character to make them more suitable to warmer weather, as seen in Figure 11.

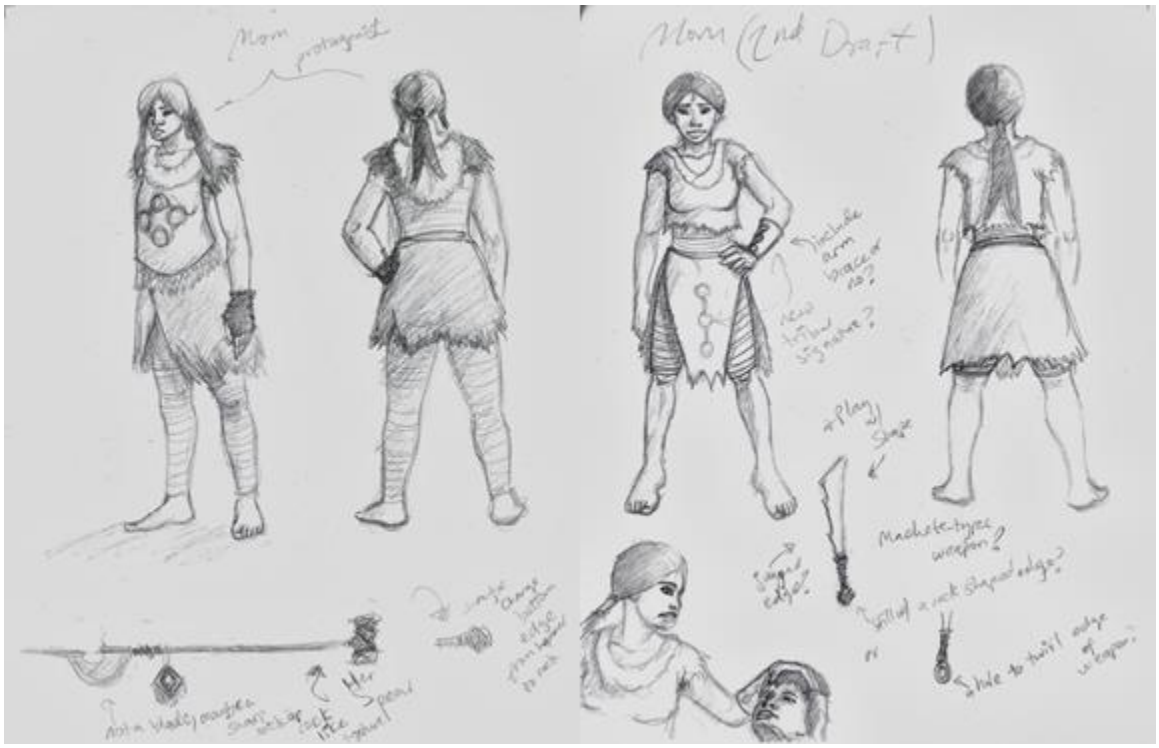

Figure 11: Main character's initial design (left) and final design (right)

Our overall consensus on the initial design was that she appeared to look too brutish and her outfit appeared to be too heavy for her to be wandering in a warm, tropical jungle. After looking at more reference, our character artist came up with a newer design on the right (Figure 11), to which we all felt addressed both criticisms raised toward the initial design. As a result, we decided to stick with the newer design.

5.2. Son Design

The initial and final designs of the main character's son were similarly based on her respective designs (Figure 12). As a result, criticisms of his design were similar; the clothes were too bulky for the character's climate. However, we all agreed that there wasn't as much about him that needed to be changed, so there is not much of a difference between his initial and final designs.

Figure 12: Son's initial design (left) and final design (right)

5.3. Enemy Design

In addition to the main characters, we also wanted to establish characters that provided a serious and continuous threat to both the mother and son's survival. We wanted to make the threats fierce, dangerous and consistent with our game's setting and story. Within the game's story we created an enemy hierarchy that starts from sacred animals that serve as minions to a god-like form of that corresponds to said animal. The two animal types we decided on were cats and snakes. The cats were

intended to be the primary threat, as the mother would be fending off several of them throughout her journey in the jungle. Later once she arrived at the temple, she would have to face a cat-god, the first boss. Then, as she wandered through the temple, she would have to avoid snake-like traps and eventually face a snake-god, of which is the final and most powerful boss. Initial concepts were generated in order to meet these conditions. In order to create a ferocious looking cat creature, we first looked at reference for wild cats, such as cougars and panthers. Another one of our team members also provided a concept of how ferocious he wanted our cat creatures to be (Figure 13). Our character artist then took characteristics from this image and other references and ended up with the design in Figure 14.

Figure 13: Reference cat beast sketch (left) and a cougar (right). [1]

Also shown in Figure 15 is a concept of the cat-god, derived from the appearance of the cat minion creatures. For the cat-god, our character artist wanted to create a look that's less ferocious and more "elegant". After looking at reference our artist tried to visually convey that elegant look in a series of concept sketches in the image on the bottom.

Figure 14: Design for cat minion.

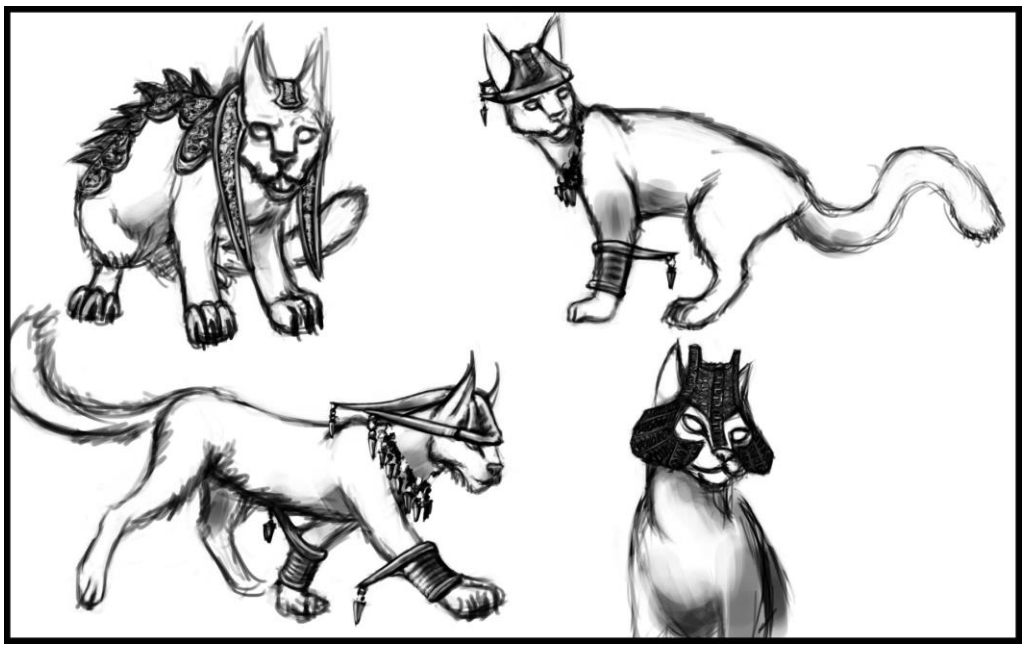

Figure 15: Design for cat god.

As for the snakes, the character artist took a slightly different approach toward establishing their visual design; reference was taken from common snakes and various unrelated sources and combined to create snakes that were unusual in appearance and had specific methods of attack. Their designs were generated with the intention of having the snakes correspond to sounds that were unique

compared to sounds made by other enemies in the game. For example, the snakes with the insect-like design would make some sort of buzzing noise, while the snakes with crown-shaped heads could make a sound that corresponds to their more primitive look (Figure 17). Lastly, the snake-god's design did not follow the process of the other snakes, instead drawing the most influence from South American culture. Specifically, our snake god was based on Quetzalcoatl, a god the Aztecs served. Our artists looked at various references to Quetzalcoatl, many of which were different from each other. The final design of the snake god was a combination of these references, but with an overall snake-like appearance (Figure 17).

Figure 16: Reference image for snake god. [6]

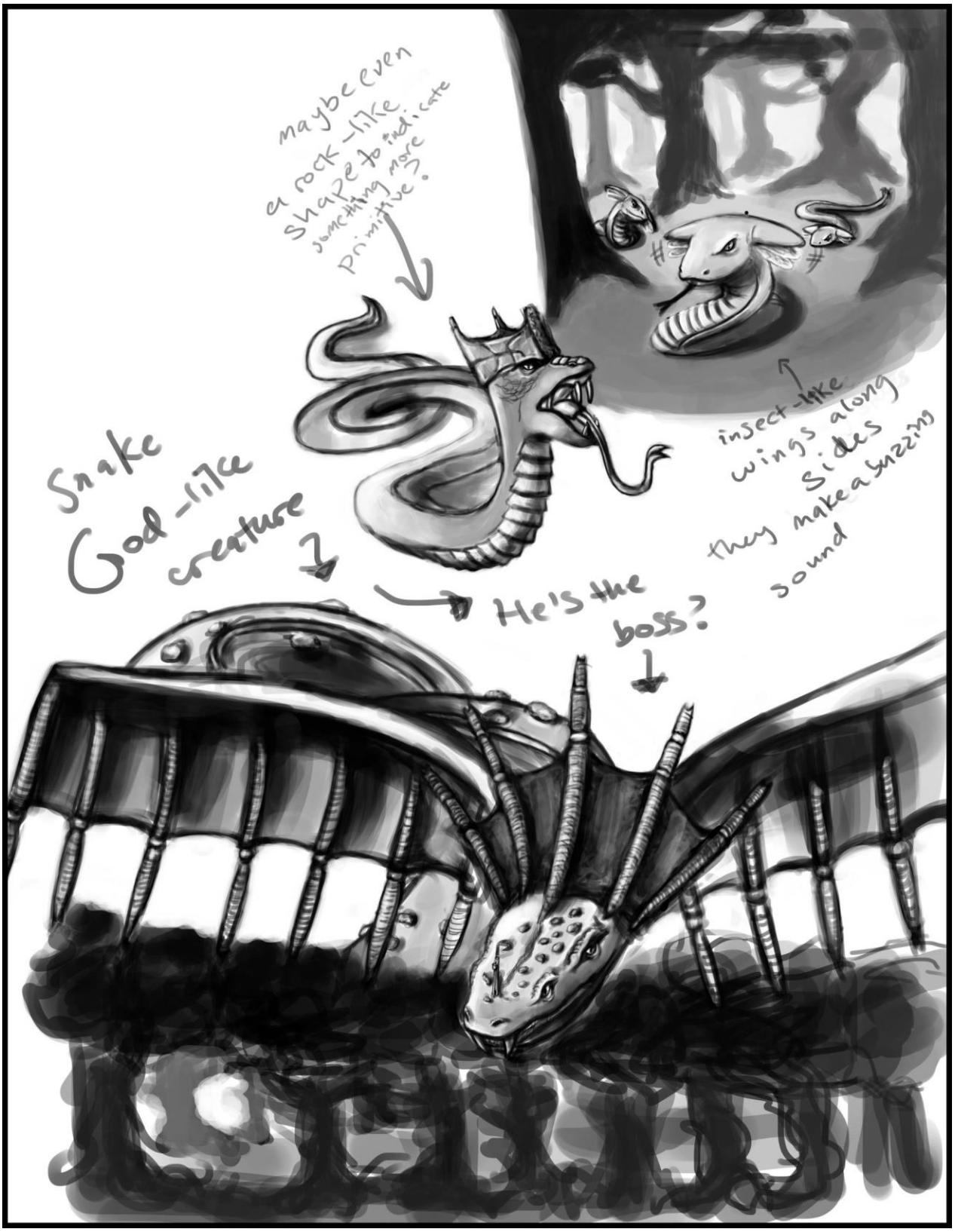

Figure 17: Design for snake god.

6. Environment Creation

Our overall environment consist of two parts, a jungle and a temple. Both places were influenced by South American forests and temples, respectively (Figure 13). Early concept art for the temple can be seen in Figure 14.

Figure 18: Reference image of a South American jungle (left) and a Mayan temple (right). [5][7]

6.1. The Jungle

Building the jungle itself proved to be a substantial but worthwhile challenge. The first and most notable part of any jungle is the flora. In order to create a convincing environment that matched the sound of the level, we would need many different varieties of trees and underbrush. To accomplish this, we used SpeedTree, a powerful toolkit for creating a variety of flora that integrated well with the UDK. This allowed us to efficiently create a large number of different trees and smaller plants to put within our jungle.

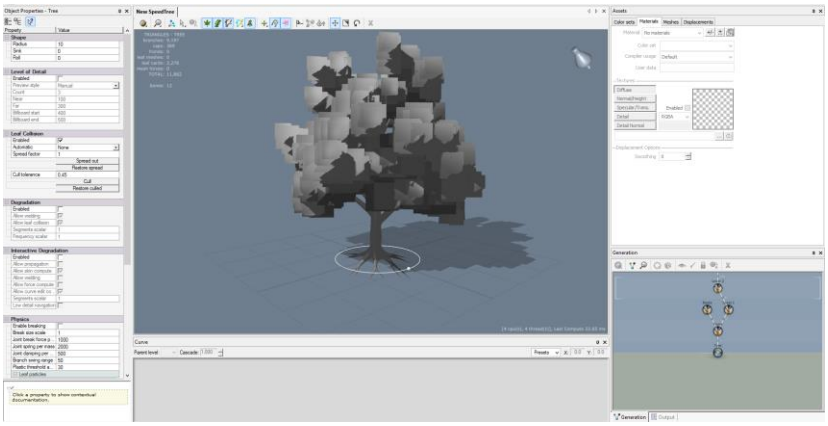

Figure 19: An in-progress SpeedTree in the SpeedTree Modeler.

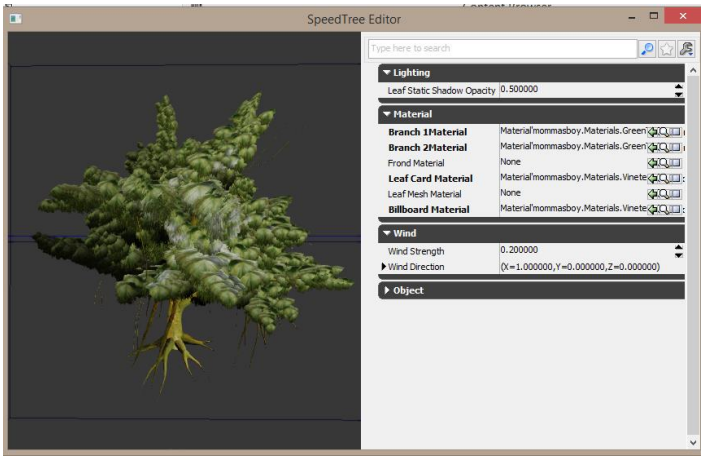

Figure 20: A finished SpeedTree in the UDK SpeedTree editor window.

When designing the types of trees that we wanted in the jungle, we decided to focus on giving each tree a unique silhouette. When referencing thick jungles or forests in other games, we discovered that the most powerful effects were gained when light would shine through thick trees and fog and create powerful sun rays and silhouette the trees. This is shown in Figure 21 and Figure 22.

Figure 21: A compelling forest scene in The Witcher 2: Assassins of Kings (2011). [11]

Figure 22: The dark forests of Tomb Raider (2013) helped to drive the mood of the game. [12]

To create the most interesting and engaging jungle silhouettes, we specifically modeled trees with curving, gnarled trunks and branches and numerous vines. These features made the jungle really stand out and appear foreboding and dangerous. Placement of lights throughout the level even further enhanced this as we placed many spotlights pointing in through the trees at a low angle which made the silhouettes even stronger.

In addition to the varieties of trees we created, the jungle also needed to have a thick layer of underbrush and many different objects to fill up the space. For this, we created a number of bushes and other types of underbrush in SpeedTree as well as a number of static meshes using ZBrush. We ended up creating around 5 different kinds of underbrush and 2 different rocks. Even with only 2 rock meshes, we were able to create numerous varied rock formation. These formations worked very well with the dim lighting, creating silhouettes.

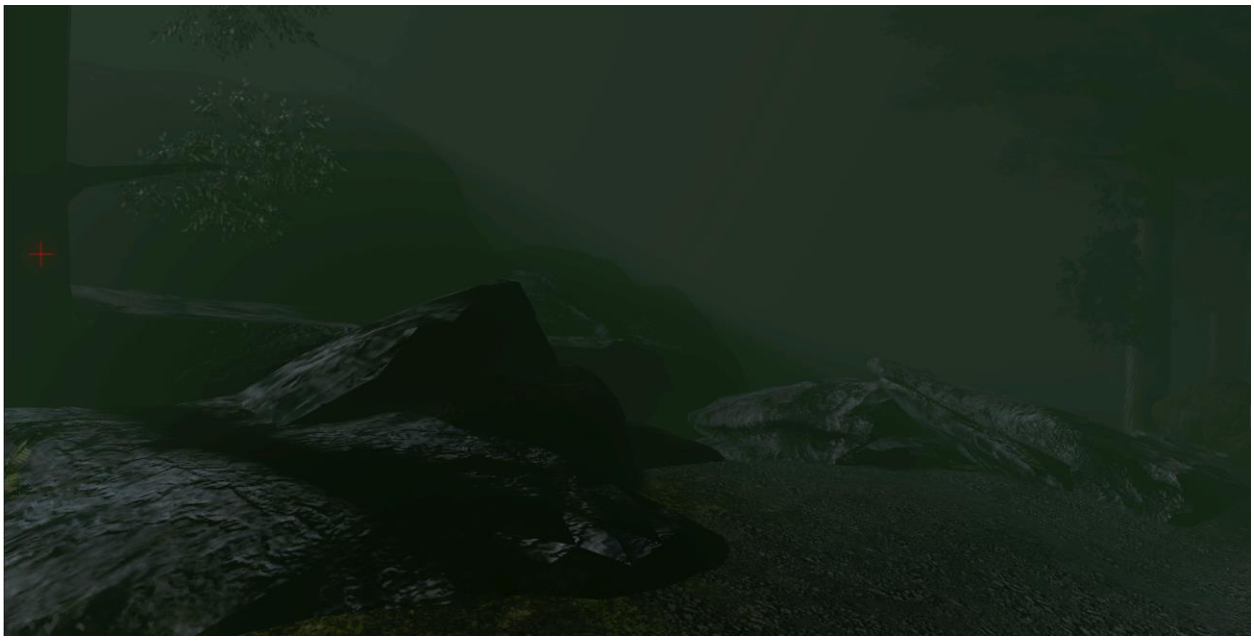

Figure 23: An example of how 2 rock meshes can form a complex rock formation.

In addition to the rocks and underbrush created for the level, we also created a number of other static meshes to be used in the jungle, including tree trunks and fallen trees, as well as a rope bridge that would allow the player to cross the rivers in the jungle.

Figure 24: The rope bridge is an example of one of our static meshes throughout the level.

Figure 25: The SpeedTrees and fallen tree static mesh's silhouettes stand out in the jungle.

All of these meshes work together with the trees and terrain of the level to create a thick, dark jungle environment that works with the Mother character and the Cat enemies and create a compelling visual environment for our sound design to work through.

6.2. The Temple

Along with the jungle level, we also created a temple for the technical requirement of the project. This temple was to be procedurally generated, including the placement of static meshes and application of textures to the temple walls. In accordance to this, the artists created a number of static meshes to be placed inside the temple as well as to create the entrance facade of the temple.

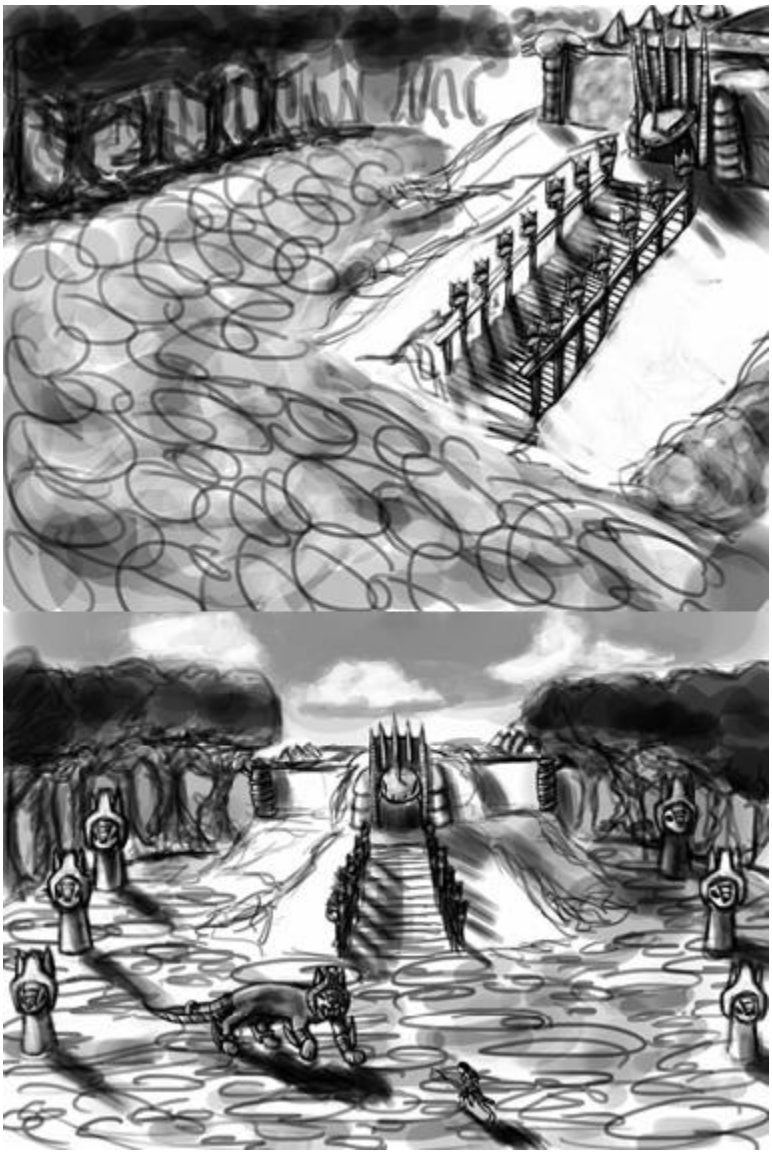

Figure 26: Concept art for the temple.

The entrance to the temple in the jungle level was built using a few meshes, as it merely needed to be something imposing for the player to see as they approached the end of the jungle. First, some simple stone blocks were modeled to be used both as stairs and as the wall of the temple, as you can see below. Then, a snake head entrance was modeled, along with pillars according to the concept art above. The pillars were repeated in between each stone block, and with proper application of lighting, the temple entrance became a final goal for the player to achieve in the jungle.

Figure 27: The temple entrance as it appears in the jungle level.

For the interior of the temple, a few static meshes were created to be procedurally placed in the temple. These include pillars from the temple entrance as well as two different snake heads. These were designed based on statues built for the Aztec god, Quetzalcoatl and are used in the level as traps as well as decorations.

Figure 28: A snake head statue mesh (left) Reference for a Quetzalcoatl Head Statue (right). [10]

7. Technical Components

As we used Unreal Development Kit to develop Maternal Instinct, all of our coding was done in Unrealscript. Unrealscript provided an object oriented scripting language with useful built-in features, such as state code. All traps, enemies, and player behaviors were implemented as Unrealscript classes. Additionally, all features related to procedural generation were implemented in Unrealscript.

7.1. Procedurally Designed Temple

Although the temple was never fully implemented, the procedurally designed temple featured some impressive technical feats, and it was very much a part of our project. We first designed an algorithm for temple generation which would ensure that the player would always pass through a number of challenges in a specific order, and also ensured that our algorithm would never run out of room to place additional rooms. This algorithm creates rooms just-in-time so that each room could be tailored towards the player's skill level. The procedural generation has three tiers, deciding the layout of the rooms in the temple, the contents of each room, and the textures applied to the objects in the room.

7.1.1. Temple-Specific Gameplay

While most gameplay elements were moved from the temple to the jungle environment, when we realized that we would not have time to fully implement the temple in a satisfying way, there are a few pieces of gameplay which were ultimately cut from the game entirely. We had initially intended to have players work their way through the temple and ultimately discover and defeat a boss based on the Mayan Quetzalcoatl. To train players for that encounter, we would introduce them to each of the boss's attacks and their associated sound cues through traps within the temple.

7.1.2. Falling Brick Trap

This trap triggers a sound above the player of a crumbling rock, and he or she must quickly dodge to the side, or else face a crushing death. During the fight with the Quetzalcoatl-inspired boss the temple would begin collapsing, and the player would have to listen for these falling rocks throughout the encounter.

7.1.3. Spitting Snake Trap

This trap is a snake-head statue which spits poison at the player as he or she walks by. The trap makes a hissing sound, and then shoots out an attack at the player which they must quickly dodge out of the way of. This was going to be one of the Quetzalcoatl-inspired boss's primary attacks, and the player would have to listen for the hisses that accompanied it to dodge out of the way.

7.1.4. Temple Generation Algorithm

We spent a lot of time early on developing an algorithm which would not only procedurally generate rooms for the player to travel through, but also generate these rooms in a way that ensured that we would be able to introduce the player to certain challenges in order. It also needed to ensure that the rooms would be visually interesting, and not feel as if they were all copies of what came before them. To do this, we created a three tiered algorithm which procedurally generates the rooms in increasingly fine detail. Using three tiers also has the advantage of independence. If one tier of the algorithm didn't work out then it had no impact on the remaining tiers.

Tier 1: High-level Room Placement

In the first pass of room creation the algorithm simply figures out how much space it has to place a room, and whether or not any room would fit there. This part of the algorithm is based off a grid. The grid spaces are claimed if there is currently a room intersecting that grid space. Additionally, placed rooms will have doors that span the grid cells. Rooms cannot be placed if the room would need

to claim a grid cell that is already claimed. Likewise, placing the room would claim a cell that an existing door already leads into then it cannot be placed.

The high-level algorithm ensures that there is always somewhere to go. This constraint is ensured because doors will always have at least one grid cell available to generate the next room. If there is no compatible room that could fit in that space then the algorithm automatically fills the cell in with a staircase. The staircase only takes up one cell, so it is guaranteed to fit. As long as the grid cells on the next vertical level are empty then the staircase will lead to a fresh slate where the algorithm can continues.

In order to guarantee that the next level of grid cells is empty we make the distinction of whether or not a door is on the main path or not. Initially the first room's door is on the main path. If the path splits then one of the split's doors will be marked as the main path. Only a door on the main path may generate a staircase. Otherwise, the algorithm will generate a dead end instead of a staircase. Since there is exactly one unexpanded door on the main path there will always be exactly one door capable of creating a staircase. Since the doors on the non-main path cannot generate staircases and there are no staircases going back down we ensure that the next level of grid cells will always be empty.

The last constraint we needed to ensure is that we can have a sense of order. For example, we wanted to ensure that a room with a snake head trap can only be placed if a room with a panther has already been placed. We implemented this by giving forcing room definitions to specify what prerequisite tags the room needs before it is placed, as well as what tags the room satisfies once it's placed. Using the above example, the room with the snake head trap would require the "Cat" tag and satisfy the "Snake Head" tag. The algorithm maintains a list of what tags have been satisfied. Each time a room is placed, the prerequisites that the room satisfies are added to the list. When the algorithm is looking for compatible rooms it only considers a room compatible if all of its required prerequisites have been added to the list. Figure 29 shows an example of a generated temple.

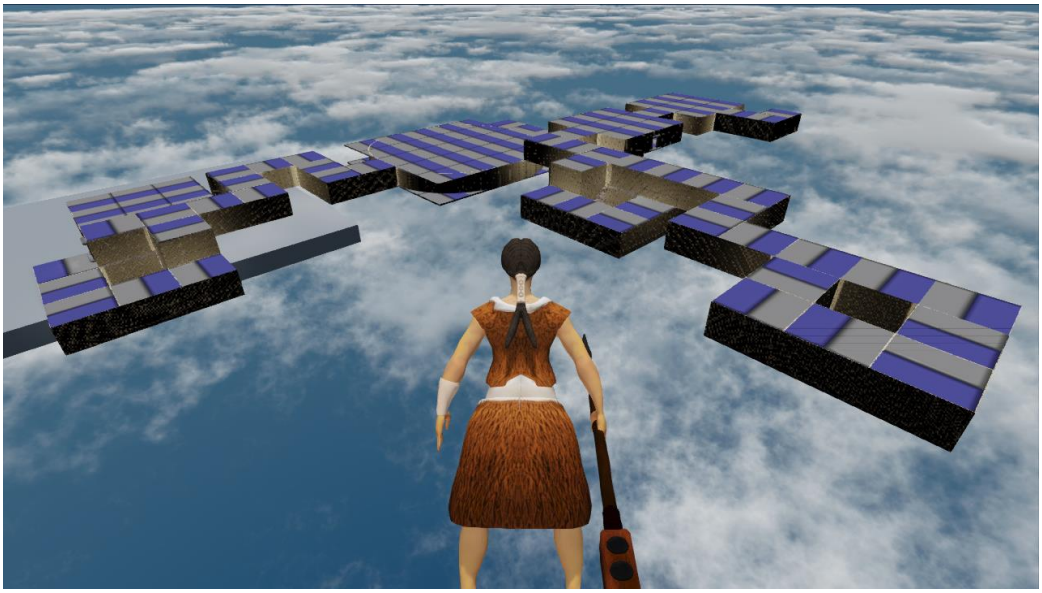

Figure 29: Outside perspective of a partially generated temple.

Tier 2: Placement of Meshes and Objects

In the second tier, the algorithm begins placing actual geometry into the world. Where the first tier manages the space and decides what rooms to spawn, the second tier fills in that space. This is where the architectural meshes are placed, where the traps are placed, and where the lights are placed. It's up to each room to place objects in the room. The only constraint is that the meshes need to be confined to the space that the room claimed.

Tier 3: Procedural Modification

In this final pass, the algorithm procedurally generates and project decals onto the items in the room, in order to simulate weathering and damage of those items. Decals could be placed to simulate water damage, erosion, crumbling bricks, or other such damage. The modifications were made using a mixture of shaders and UDK's material editor. Unfortunately, this portion of our algorithm was never completely finished due to issues implementing the procedural textures in UDK. Figure 30 shows a generated room. All objects except for the player were placed procedurally. Additionally, the wall's texture was created procedurally.

Figure 30: Inside view of a generated temple room.

7.2. Cat Enemies

At the heart of the cat enemies is a state machine. The desired cat behavior includes stalking behavior, pouncing behavior, stunned behavior, and fleeing behavior. Each of these behaviors is implemented as a state. An overview of the states and their transitions is shown in Figure 31.

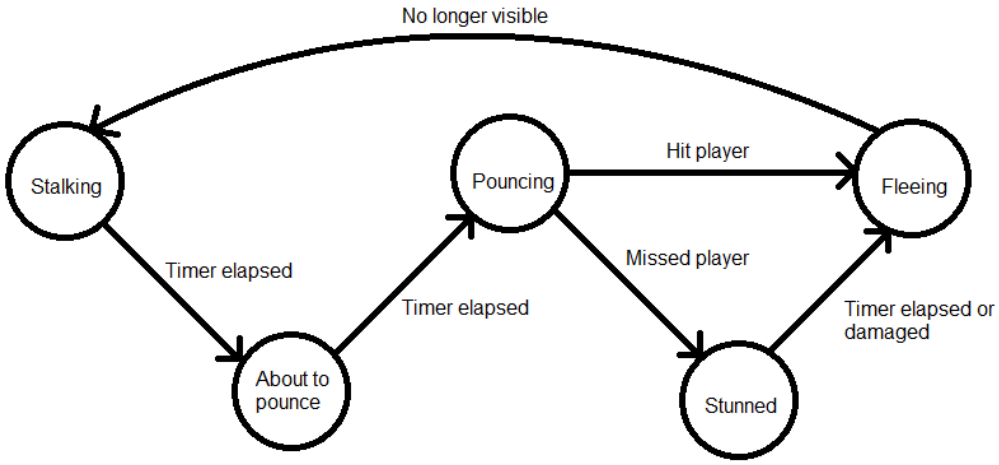

Figure 31: Cat state machine represented as a diagram.

7.2.1. Stalking

The cats spend most of their time in the stalking state. Here, the cat is invisible, invulnerable, and the player cannot interact with it. While in this state, the cat plays a growl sound in various locations around the player.

7.2.2. Pouncing

The pouncing behavior is actually two separate states. The first state is for when the cat is getting ready to pounce. This state solely exists to play a distinct sound that the cat makes before it pounces. Playing the sound before the cat actually pounces gives the player a window to react. By varying the time that this state lasts (after which it actually pounces) we can alter how difficult it is to dodge the cat.

Once the prior state finishes the cat enters its actual pouncing state. Here, the cat becomes visible, placing itself where the last growl sound came from. It then jumps toward the player, dealing damage if the two collide.

7.2.3. Stunned

The cat enters the stun state after it pounces, but only if it failed to hit the player. In the stunned state the cat is temporarily immobilized and vulnerable to damage. This allows a player who successfully dodged the cat a small window to retaliate. Once hit, or after a predetermined time period, the cat exits the stunned state.

7.2.4. Fleeing

After the cat successfully hits the player or the stunned state has ended, the cat runs off into the fog to begin stalking again. As it runs it fades out, providing a smooth transition between being visible and invisible.

7.3. Traps

There are four traps that appear throughout the game, though only two appear in the jungle level. The four traps are falling rock traps, falling tree traps, spitting snake traps, and falling brick traps. The former two appear in the jungle, while the latter two appear in the temple. Since a description of the traps was described prior to this section, the technical section will only include the implementation details of the traps.

7.3.1. Falling Rock

The falling rock traps required very little code to create. Since each falling rock trap requires a unique path of travel, the rocks are controlled using UDK's Matinee feature. The Matinee allows the level designer to specify a path for the rock to follow. Once triggered through Kismet, the rock moves down the path. If the rock collides with the mother then she is killed. The script for the falling rock only specifies that the rock should kill the mother upon colliding with her.

7.3.2. Falling Tree

The falling tree trap was implemented using a mixture of Kismet and state code. The tree script has three states corresponding to when it's standing, falling, and fallen. A Kismet trigger signals when the tree should start to fall. Additionally, the script allow the level designer to specify the area onto which the tree falls through Kismet. Figure 32 shows how a falling tree encounter is implemented in Kismet. Upon triggering the trap, the ambient warning sound stops playing and Kismet tells the tree to start falling.

The falling tree script is was designed with SpeedTree integration in mind. By using a SpeedTree component rather than a static or skeletal mesh component we allowed the level designer to use any tree designed in SpeedTree. Normally this would require a modeled mesh, so integrating SpeedTree saved our artists valuable time.

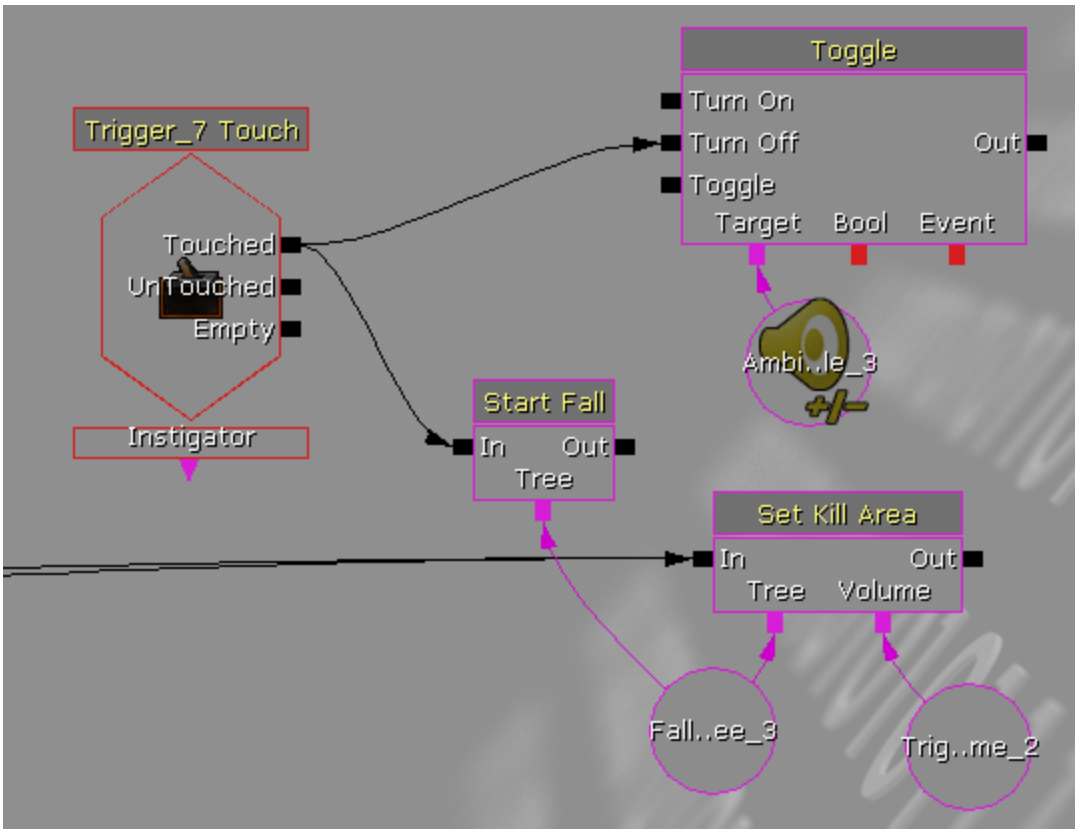

Figure 32: Kismet implementation of a falling tree trap.

7.3.3. Spitting Snake Trap

The spitting snake traps were relatively simple. Upon being triggered the statue spawned a projectile and fired it in the direction the statue was facing. There were plans to use the trap in a variety of situations, so the trap needed both Kismet support for manual level design and non-Kismet support for procedural level generation. The statue itself only required an outside source to tell it to fire. If done via Kismet, a Kismet node told the statue to fire. This proved unnecessary though, as the trap was never placed manually. Instead, all occurrences of the trap were generated procedurally. In this case, the generator provided a special collision object through Unrealscript. Upon collision with the object, the object itself signaled the trap to fire.

7.3.4. Falling Brick Trap

The last trap, the falling brick trap, only appeared in the temple and only supported placement through Unrealscript. The procedural generator specified where to place a specialized trigger through Unrealscript. Upon collision with the trigger object, the trigger object itself created the falling brick and removed itself from the game. The brick followed rigid body physics, so its path was determined by the engine. Once the brick collided with anything, be it a player or the floor, it removed itself from the game and dealt damage if necessary.

7.4. Player

The player character and combat system is made up of three classes, a Controller, a Pawn, and a Weapon, each of which extend their respective UDK class. The Controller is responsible for handling player input, while the Pawn is responsible for the player's character within the game world, and rendering information about the player. Finally, the Weapon is responsible for the combat system.

7.4.1. Player Controller

The player controller was the simplest of the three classes to implement, as there were very few changes required from the base class. The only changes made were the addition of code to allow a player to dodge.

7.4.2. Player Pawn

The player pawn was where most of the coding for the mother character's non-combat capabilities was done. Within the pawn class was coding for the 3rd person camera that we implemented, as well as additional code to handle the player's dodge, which worked in conjunction with the previously mentioned code in the player controller.

7.4.3. Player Weapon

The mother's spear contains all of the code for the combat system in the game. This combat system was designed based on a tutorial found online for a melee combat system within UDK, and then heavily modified and customized, in order to suit our needs. In order to accomplish our needs, we had to add additional code within the Pawn class, providing a number of public functions that the weapon could call, in order to handle certain actions which would influence the player's state within the game world. (Basic Melee Combat System)

8. Conclusion

In the end, we achieved most of our goals with *Maternal Instinct*. Our chief goal was to create a game where the player had to rely on compelling, positional sounds to guide them through the level and encounter threats that would be near undefeatable without understanding and reacting to the sounds that they heard. *Maternal Instinct* accomplishes this goal, with numerous traps and enemies that use sound, not in a gimmicky fashion but in an immersive and enjoyable aural experience that enhances gameplay. We also wanted to create art for the game that matched well with the sound but did not overtake it in importance, and with our dark, foreboding landscape with lighting that creates an environment where seeing is not the be-all, end-all of navigation we believe that we have achieved that as well.

The technical goals of *Maternal Instinct* were not only to create enemies and traps that could use the art and sound that were created, but also to create a temple level that was procedurally generated on three levels. While this temple level was unfortunately not completed in time, and ultimately cut from the project, we succeeded in our other goals of creating enemies and traps which would use the art and sound in meaningful ways. Additionally, though the third tier of the procedural generation algorithm did not work well enough that it met our standards for satisfaction, the first two tiers worked very well, and the temple level is completely playable, though it is not as visually appealing as we would have liked.

8.1. Post Mortem

User response to *Maternal Instinct* was very positive when we demoed it at PAX East, and though the game was lacking polish, users enjoyed the sound-based mechanics that we had

designed. We had done tests originally and found that UDK seemed more likely to produce a favorable result in our procedural texture generation. However, when we got to writing our brick wall shader, it turned out that UDK's unfamiliar shader language was too much of an obstacle to overcome in the given amount of time.

References

[1] Cougar. Retrieved October 2013, from: http://4.bp.blogspot.com/--jEwLoAO26k/TV-Ml8IkIzI/AAAAAAAAA14/YcPLkbSKd7M/s1600/cougar-picture.jpg

[2] Legend of Korra art. Retrieved October 2013, from: http://2.bp.blogspot.com/-D2j4oGqFE2o/T9H6LDxadTI/AAAAAAAAFLY/SBS_wrkZOpc/s1600/korra-the-legend-of-korra-2.jpg

[3] Legend of Korra art. Retrieved October 2013, from: http://www.cosplayisland.co.uk/files/costumes/5963/67271/CI_67271_1345491019.jpg

[4] Mayan Flints. Retrieved October 2013, from: http://www.edgarlowen.com/mayan-flints-11363a.jpg

[5] Mayan temple example. Retrieved October 2013, from: http://photos.demandstudios.com/225/194/fotolia_8869902_XS.jpg

[6] Quetzalcoatl. Retrieved October 2013, from: http://dragonpictures.us/images/gallery/uploads_big/dragons/Quetzalcoatl-lightning-green-dragon-215.jpg

[7] Rainforest example. Retrieved October 2013, from: http://theburtonwire.com/wp-content/uploads/2012/09/Rainforest-e1347932358660.jpg

[8] References of over-sexualized female character. Retrieved October 2013, from: http://cache.desktopnexus.com/thumbnails/1163591-bigthumbnail.jpg

[9] References of over-sexualized female character. Retrieved October 2013, from: http://idrawgirls.com/tutorials/wp-content/uploads/2012/11/painting-tutorial-female-hunter-chief.jpg

[10] Reference of a Quetzalcoatl statue Retrieved October 2013 from: http://media.web.britannica.com/eb-media/34/20534-004-4179E3E3.jpg

[11] Reference of a forest from The Witcher 2: Assassins of Kings Retrieved April 2014 from: http://chrismweb.com/2011/05/17/the-witcher-2-assassins-of-kings-review-part-1/

[12] Reference of a jungle in Tomb Raider (2013) Retrieved April 2014 from: http://forums.guru3d.com/showthread.php?t=334097&page=30

Sound File Attribution

The following sound files were downloaded from Freesound.org and used under the Creative Commons Attribution License.

poppyshaker.wav
http://www.freesound.org/people/NoiseCollector/sounds/41614/
Rockfall in mine.wav
http://www.freesound.org/people/Benboncan/sounds/60085/
Fly.wav
http://www.freesound.org/people/Benboncan/sounds/81970/
Fly00.wav
http://www.freesound.org/people/dobroide/sounds/7973/
Fighting Game Grunts - YoungFemale.wav
http://www.freesound.org/people/AderuMoro/sounds/213295/

The following sound files were purchased under license from SoundDogs.com

JAGUAR_GROWLING__60038503