University of North Carolina Asheville
Journal of Undergraduate Research
Asheville, North Carolina
May 2022

Integrating Unreal Engine 4 and the IF Magic microcontroller through Blueprint programming and level sequencer animations

Sarah Hendricks
Computer Science
The University of North Carolina Asheville
One University Heights
Asheville, North Carolina 28804 USA

Faculty Advisor: Dr. Victoria Bradbury

Abstract

The Rattlin' Bog Project is a collaboration of premises from the fields of New Media Art and Computer science. This paper presents highlighted points of interest of the Rattlin' Bog project from a programmer's point of view. The Rattlin' Bog project was a team effort. The research explained in this paper is that of using Unreal Engine to fit the needs of this project. The processes recounted range from the amalgamation of character and environmental animation using Unreal Engine's level sequencer, to that of the integration of the IF Magic Microcontroller which obtains user input from a physical computing device and then transfers it to UE4's Blueprints visual scripting language, which controls the game play.

1. Introduction

This paper outlines the process of creating an in-progress new media and computer science project, Rattlin' Bog, that integrates the Unreal Game Engine with an IF Magic microcontroller. This project is being implemented by a team, working under the direction of Dr. Victoria Bradbury, that included a 3D modeling, rigging and animation expert, Clara Tracey, Shiasia Beasley, an artist prototyping the sculptural component, David Freund, a composer and Sound designer, and Sarah Hendricks, a programmer, implementing all of these components together through Blueprints, Unreal Engine's Visual scripting language.

From a conceptual standpoint, Rattlin' Bog is a commentary on the political climate in America today. There are references within the project that Speaks critically to Alt right misappropriation of Norse mythologies, the Gadsden flag, and Trump's 2016 campaign catchphrase, Draining the Swamp, as well commentary on climate change and its effect on ecosystems and their inhabitants. Rattlin' Bog states that draining the swamp would damage our culture and ecosystem beyond repair. Instead, we need to come together and fill the proverbial swamp to make it a place of prosperity once more. While these are compelling ideas as they come together in this project, this paper is focused on the programming in Unreal Engine, and the different approaches that were needed to accomplish the combining of these many moving parts.

The aim of the game is for a single user to fill the bog with water to bring tranquility to all its inhabitants, and to allow the plant life to flourish once more. The user will engage with a physical device, yet to be determined, to fill a reservoir with water. As the reservoir is filled with water, the onscreen bog too will fill with water. Through Blueprints and the power of the UE4 level sequencer tool, all these game assets, and characters, are coming together to create an engaging and unique user experience. There are not many artistic physical computing games like Rattlin' Bog. If Rattlin' Bog could be compared to another artist's work one could look at the works of Jonah Brucker-Cohen who is a Ph.D. from Trinity College Dublin and is a researcher and artist. His work on Data Materialities, a 2016 art gallery hosted by ACM SIGGRAPH , takes the premise of people's consumption of data via technology, and creates physical devices to speak on how immersed people are as a whole in their technologies. His piece Printcade, showcases this

idea. ” The Art Gallery for SIGGRAPH 2016 intends to bring large-scale, immersive physical displays to the conference. As computer graphics change daily and become more powerful and closer to “real world” renderings, it becomes more difficult to discern the differences between real and augmented realities. ” [1] The work of Jonah Brucker-Cohen is reminiscent of some aspects of Rattlin’ Bog in the way that it combines computing and physical interactive displays integrated with Technologic themes.

2. Definitions

Unreal Engine 4 (UE4) is a physics engine and real time 3D graphics tool that can aid in the creation of projects that range from game development and animation to architecture concepts.

Blueprints in UE4 allow for visual scripting that can extend the lower-level C++ scripting language. Both Blueprints and C++ can work in tandem to create high-level gaming content. Blueprints make coding for game development much more user-friendly for coders of any experience level, however, that does not mean that they are simplistic. For example, below is a blueprint that drives most of the animation of Rattlin’ Bog. This blueprint has quite a bit of functionality. The main input nodes, which are the first nodes that are in red, take button presses and then execute a line of code to make everything else work. If one wanted to, they could write this exact code in C++. Though there are certain features that C++ has for the more advanced and obscure tasks that a programmer may wish to use, blueprints perform most tasks that one would need in a user-friendly way. “ Unreal Engine uses the text-based programming language, C++. In addition, Unreal Engine uses visual scripting called Blueprints which utilizes a faster programming option via drag-and-drop. ” [2]

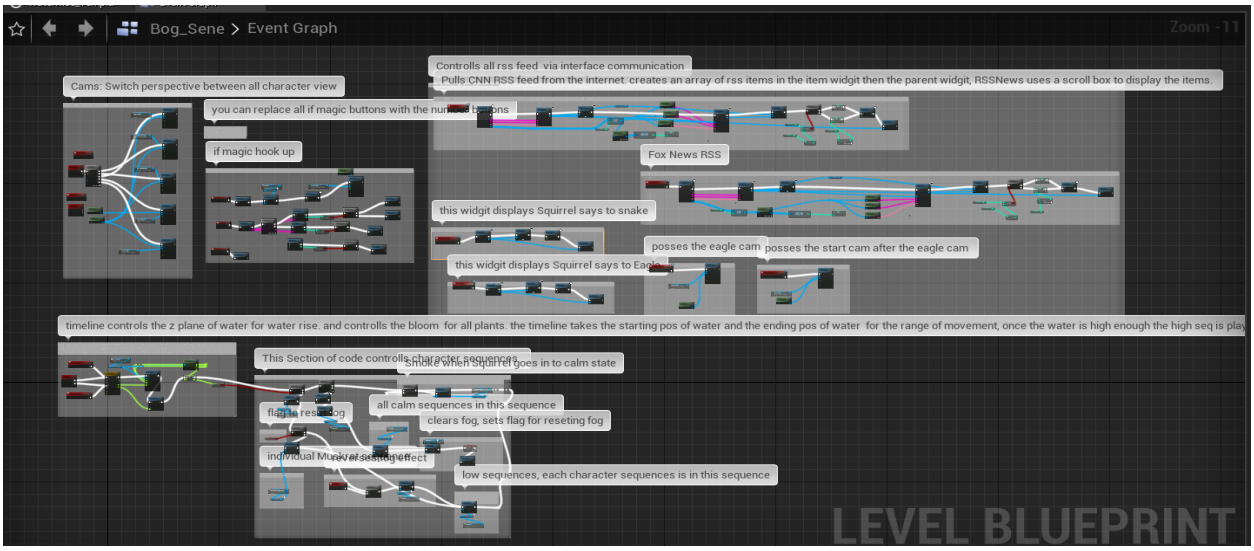

Figure 1. The Level Blueprint

The level sequencer tool is a streamlined animation, and visual effects tool built into UE4 that allows a developer to sequence animations and implement control over the virtual environment in an efficient manner. All the characters in Rattlin’ Bog, along with the atmospheric lighting, and fog were triggered using the level sequencer tool.

The IF Magic is a microcontroller that allows for the construction of a myriad of devices that can easily interface with Unreal Engine. The IF Magic has modules such as buttons, joysticks, and sensors that easily plug into ports on the device and can then be programmed in Unreal Engine using the blueprints visual scripting language.

The IF Magic is a startup created by and Lance Chantiles, and his team, who are based in New York and have agreed to allow the Rattlin’ Bog Team to be Official beta testers for their device.

A transform is composed of three vectors. An X plane, a Y plane, and a Z plane. This gives an object a location. The X and Y plane give the left and right coordinates while the Z plane gives the up and down coordinates.

RSS Feeder is a plug in that allowed For the integration of RSS feed into the game

Blueprint interfaces are a primary form of inter blueprint communication. By using interfaces, a blueprint can make calls from one blueprint to another.

3. Process

The project began with pre-animated character assists, a bog landscape, a water texture and mesh pack from the Unreal marketplace, and a large tree, which resides in the center of the swamp and serves as a centerpiece, and focal point for all of the characters to orbit. The characters include an Eagle, a Squirrel, a snake, and a muskrat. All of which have roots in mythologies or symbolically represent the American people or America as a whole. Here is a screenshot of the landscape.

Figure 2. the Bog

At the beginning of the project, all of the characters were stationary in their starting positions, and the water was in its low state. The Task was to get all the characters moving, have the tree and all plant life bloom, and to get the water to rise up and fill the bog. When the water is low the characters interact in an aggravated manner, and when the water is high, all of the characters are calm and enter contented animations. A large snake is wrapped around the tree's roots, on the trunk of the tree, a squirrel sits with an acorn, and at the top of the tree, is an eagle laying in its nest. The tree was bare along with the plants around the landscape. The snake was to slither around the roots biting and chewing them, and the eagle was to pace around the nest all while the squirrel ran up and down the tree to either of them talking back and forth to each of them all While a muskrat ran around the bog. Once the water rises, the Snake sits happily at the bottom of the tree while the eagle flies in a circle around the tree and the Squirrel eats its acorn while the muskrat swims happily.

The water was a flat plain that was created originally with a brush. Brushes in UE4 do not have any programmability. They are basically placeholders for the finished product. The water needed to rise by taking input from a player controller and filling the bog. The water was replaced with a flat plain mesh and given a blueprint that allows the

water to be programmable yet keep the same look that it had to begin with. In the water's new state, it was given a transform. Using its transform, the water takes values from a timeline node and uses that information to move along its Z plane which the water allows the water to rise and fill the bog. The yellow node in figure 3 is a timeline node. The inside of the node is shown below in figure 4. The Graph inside of the node, called a Float Track, lets you set the range of parameters that the water uses to rise up and down.

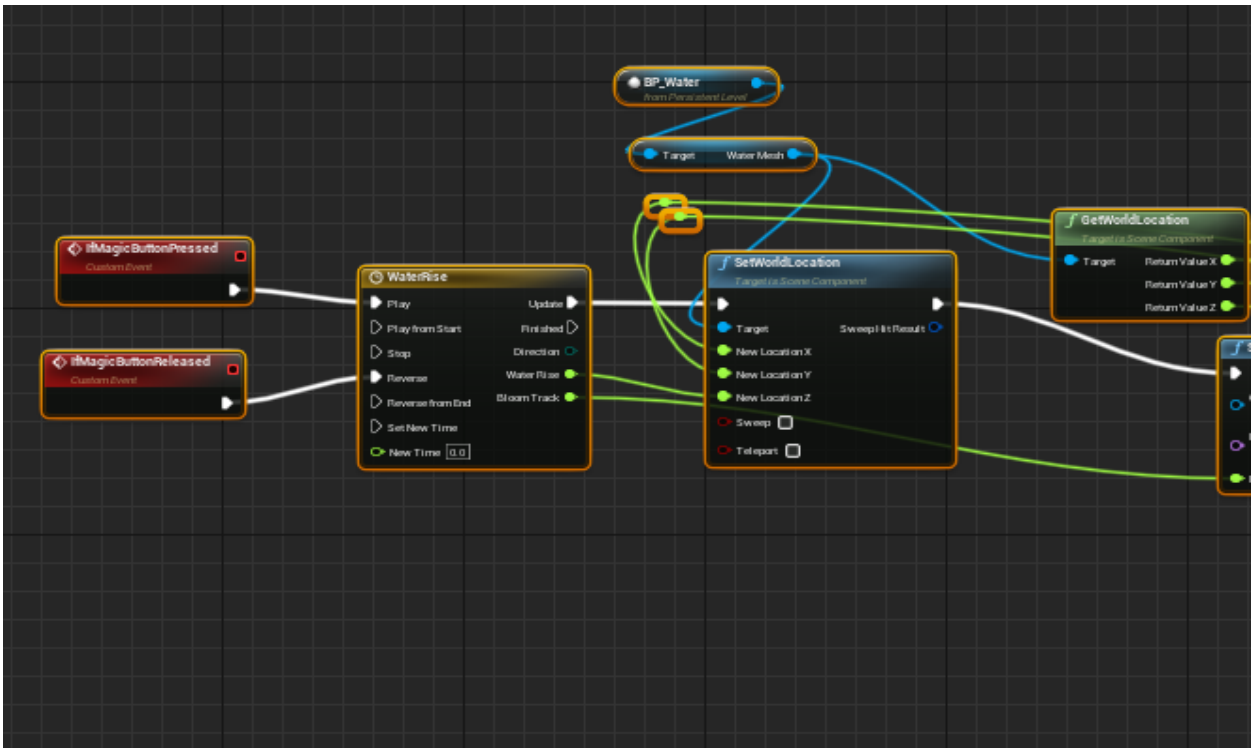

Figure 3. the Timeline Node

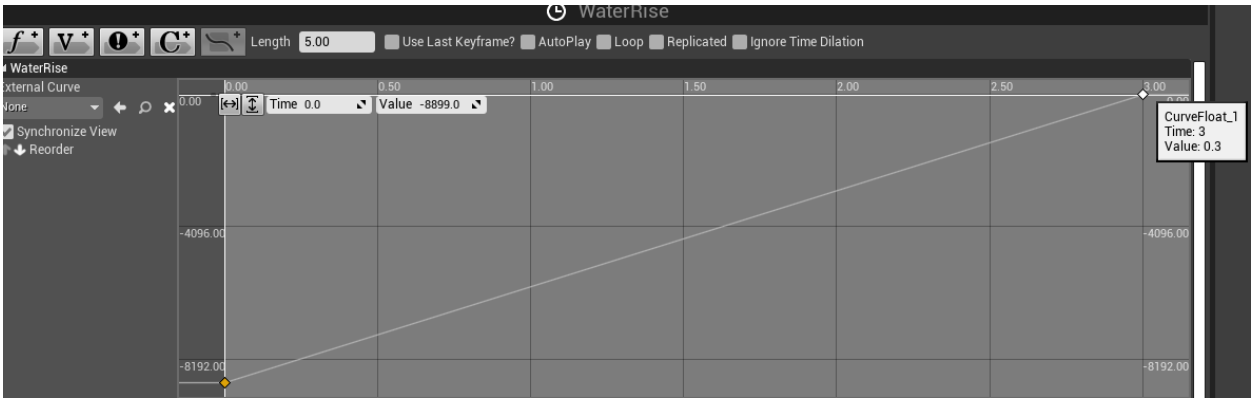

Figure 4. Inside the Timeline

The values in the timeline node range from the water's starting position to its risen position. Once the water reaches its full state a Boolean variable is set to tell the character's high-water sequence to play, which encompasses all the character's calm animations. Otherwise, while the water is in its starting position, the character's low sequence plays which contains all the characters' aggravated animations. Along with the water's timeline, a bloom timeline was implemented to control the bloom of all plants. As the water rises, the blooming of all plants slowly goes from no

leaves on any plant to a beautiful and luscious landscape. The Bloom timeline was encoded using the same timeline node and works in the same way.

The bloom system is an interesting point in this game's development. The system uses a material parameter collection called MPC.

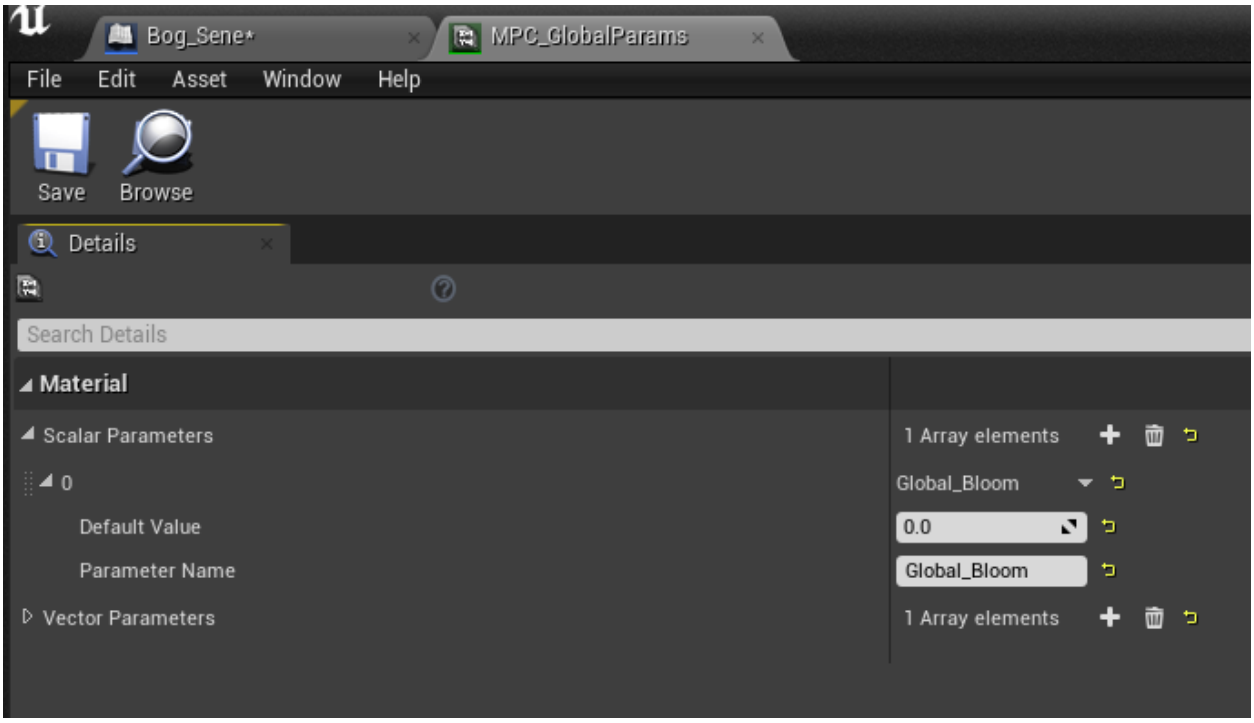

Figure 5. the MPC

an MPC offers a way to put a list of parameters together to be referenced efficiently. The values that give the range that scales the bloom of the plant life were included in a material parameter collection. The MPC is then accessed via the Timeline mentioned above. Figure 6 shows how the parameter, Global Bloom, contained in the MPC, is connected in the code execution line located in the level blueprint. There are two float tracks located within the timeline node the second track is the track being used for the set scalar parameter nod on the left side of the picture you can see that the set world location node shown in the execution line just before the set scalar parameter node which controls the waters rising ability work simultaneously when a button is pressed, or in this case when the physical computing device simulates a button press. Here is the official description form unreal engine documents for better clarification. " A Material Parameter Collection is an asset that stores an arbitrary set of scalar and vector parameters which can be referenced in any Material. It is a powerful tool that artists can use to get global data into many Materials at once. It can also be used to drive per-level effects, such as snow amount, destruction amount, wetness, etc., that would otherwise require setting individual parameter values on many different Material Instances in your level. " [3]

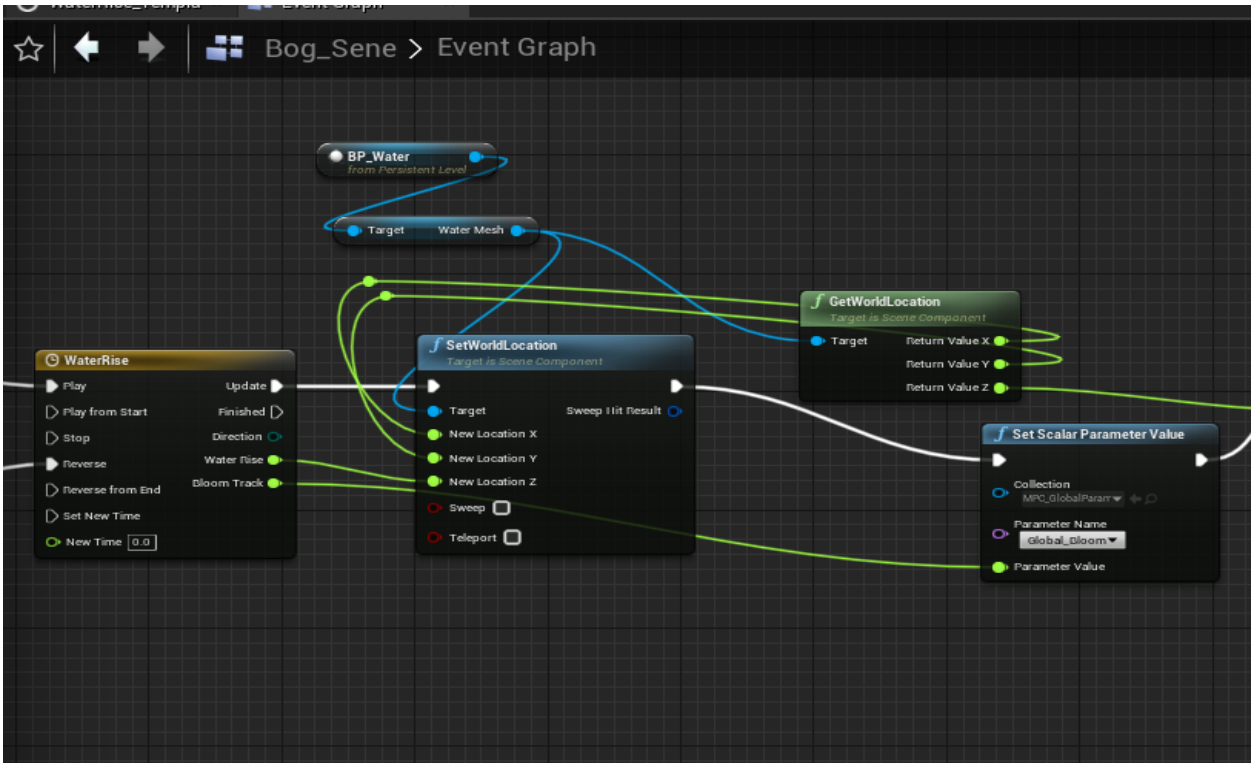

Figure 6. the MPC in use

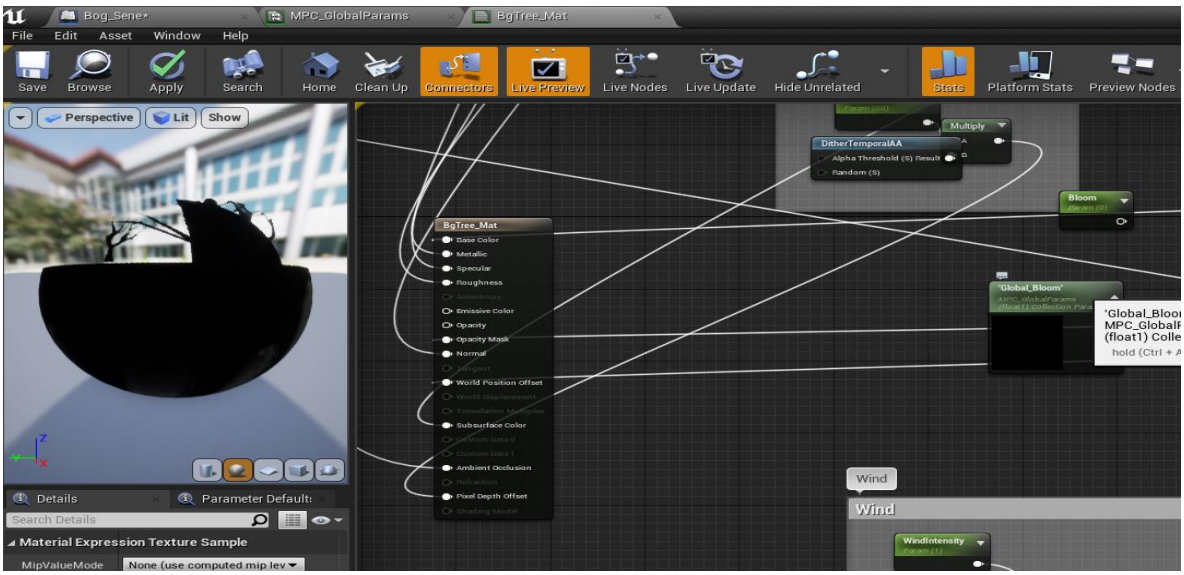

Figure 7. Inside the Material

Once The MPC is constructed, it can be placed into a material assist. This allows access to the bloom properties for the timeline to control. Shown in Figure 7 is a partial event graph where a material for a tree is shown. The global bloom node is on the right of the image. This node is accessed through communications from the main blueprint that controls the scene in the game. It gets player input, then uses the timeline system mentioned above to bloom the leaves of the tree. Clara, who is the animation and 3d modeling specialist set up these materials, which helped bring the

bloom system to life. Because of the way the materials were constructed, all the plant materials have the ability to gain input data, which then can control attributes of the material such as opacity and roughness based on the information gained from the scalar parameter, Global bloom, contained within the MPC, which is indirectly controlled by the player input.

When input was first implemented, the game would not receive the keypresses from the player controller. A few methods were tried such as using the direct communication node, enable player input, which did nothing. It turned out that the game was not Constructed with a game mode base. A Game mode base is what sets the default configurations such as the default spawn location, which is where the beginning view generates, and where the instructions for the game to recognize player input comes from. After the Game mode base was added, the project received input just fine. There were quite a few different roadblocks that had to be overcome within the development process. The biggest roadblock was figuring out the animation systems within UE4 and learning how they translated to the needs of the Rattlin' Bog project.

4. Integrating If Magic

Unreal Engine 4 is a great assist to any developer or development team that wishes to go beyond the normal confines of the everyday game, or project development. Unreal Engine is helping our team to realize a nonconventional format of user input. In the case of Ratlin' Bog, The Goal of the If Magic Is to create what is essentially an art piece that serves as a player controller. The If Magic will be masked with wooden cut outs in the shape of the characters and the central tree which they all revolve. When a user interacts with the wooden cutouts, it will trigger a button press that tethered to the If Magic, which will then control the on-screen interaction of the characters and environment. This part of Ratlin' Bog is currently a work in progress and is being implemented by Dr.Bradbury and Shiasia Beasley.

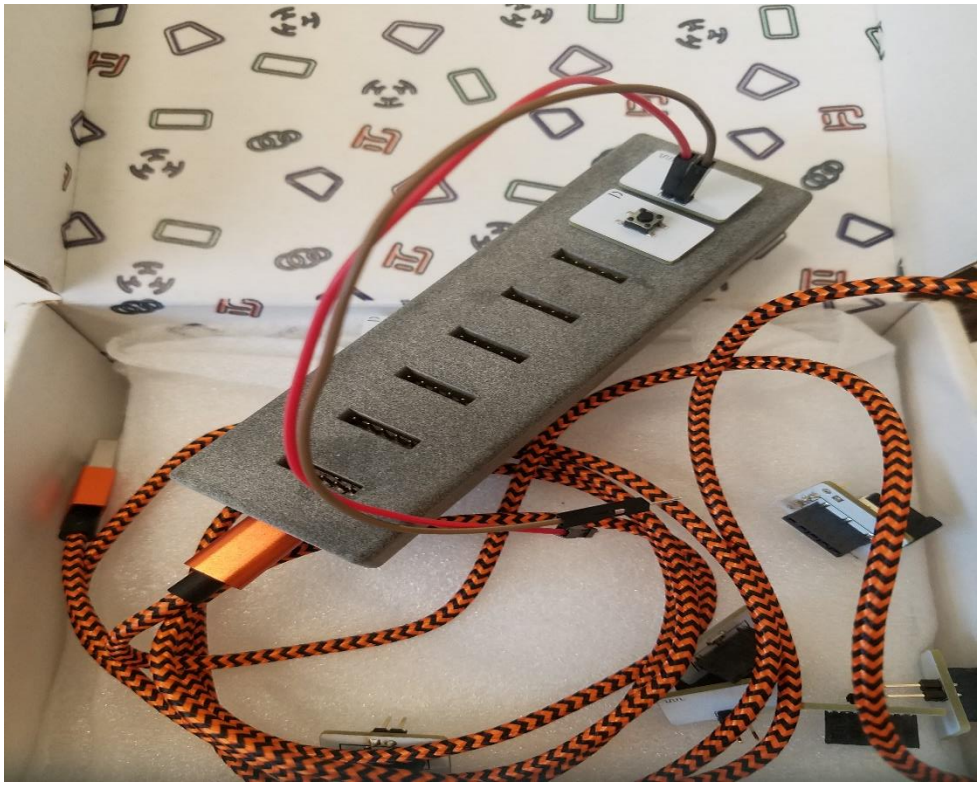

Figure 8 The If Magic

Figure 9. Character Cutouts

Figure 9 shows an example of the squirrel and branch cut out which has been crafted with a laser cutting tool. Once all the pieces are made, they can then be unified with the If Magic, and incorporated into the Rattlin' Bog instillation as a user controller. The wood cut out instillation will most likely be a wall hanging that a user can interact with while viewing the game on a large screen. Depending on which wood cutout a player moves, it will trigger a view change of the scene, or it will fill the bog with water just like a key press on a regular controller or keyboard.

Programming the If Magic device was relatively straightforward, and successfully provided the onscreen effects that were needed to carry out the project. By using the IF Magic Plugin, Blueprints enables a connection to the device through serial communications via COM port. Through this method, Blueprints can get communication from the physical computing device providing input to the game. In this case, Button presses are communicated through the connection to create custom events to occur within the level blueprint. Through the custom events, every other line of code is executed.

The process that led to the integration of the IF Magic was a smooth one. Before the If Magic was integrated into the project, the player input was coming from the keyboard. UE4 makes input configuration relatively easy. There is a list in the project settings where a designer can program key mappings to trigger certain events. Figure 10 is the action mappings setting list where any input controller or device that is standard to UE4 can be programmed. For instance, the water was being filled when a player pressed the R key. Once the R key is programmed to be the key that triggers that event, that input gets a node that can be used in the blueprint where the action happens.

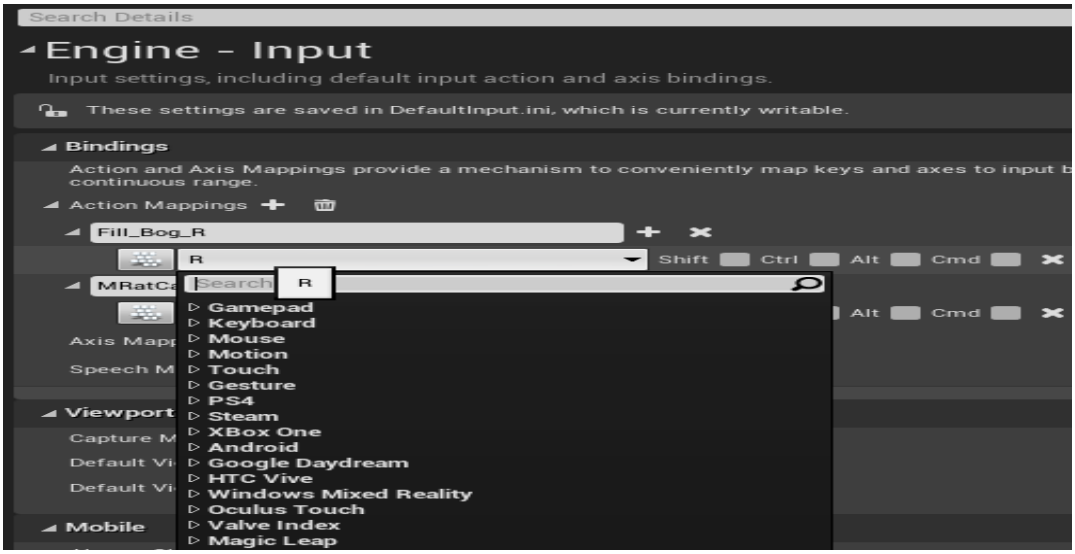

Figure 10. Inputs

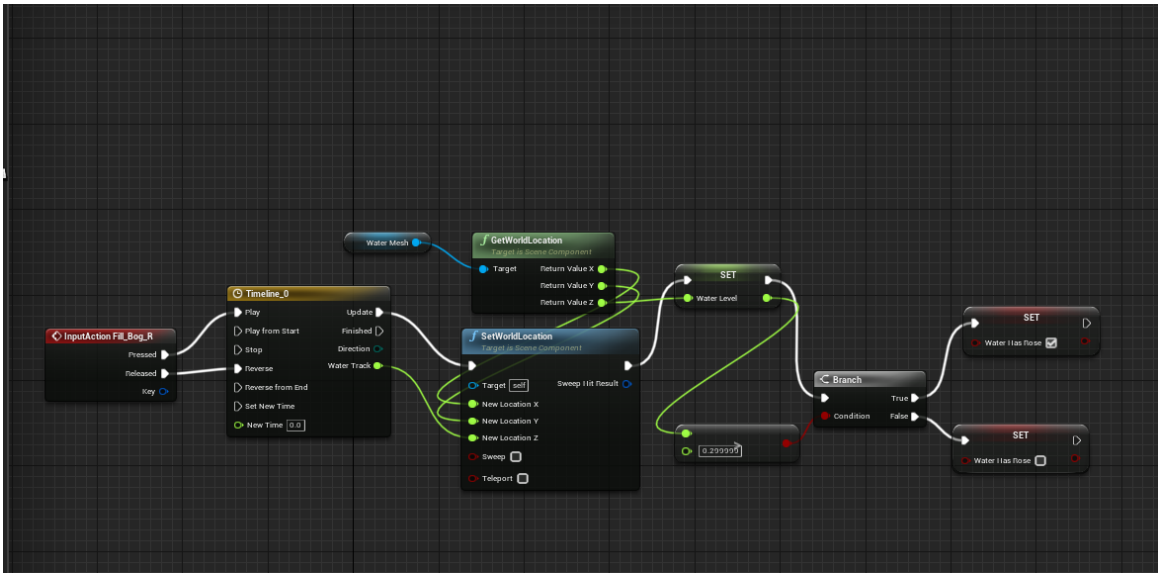

Figure 11. original Input

The red node to the left of the graph recognizes when the R key is pressed. Once a player pressed that key the timeline node tells the water to rise. This concept was basically the same for the IF Magic, instead of using the standard input game settings, the IF Magic communicates directly with blueprints.

Once the blueprint establishes the port in which the device is connected, a programmer has access to its library functions where button presses, among other inputs can be used. Figure 12 shows an image of the If Magic input nodes being switched out with the R key press. It works in almost the exact same way, though there is a bit more logic required to receive the button press from the IF Magic device. Once the If Magic is integrated into the logic, the physical device can then be built around the If Magic.

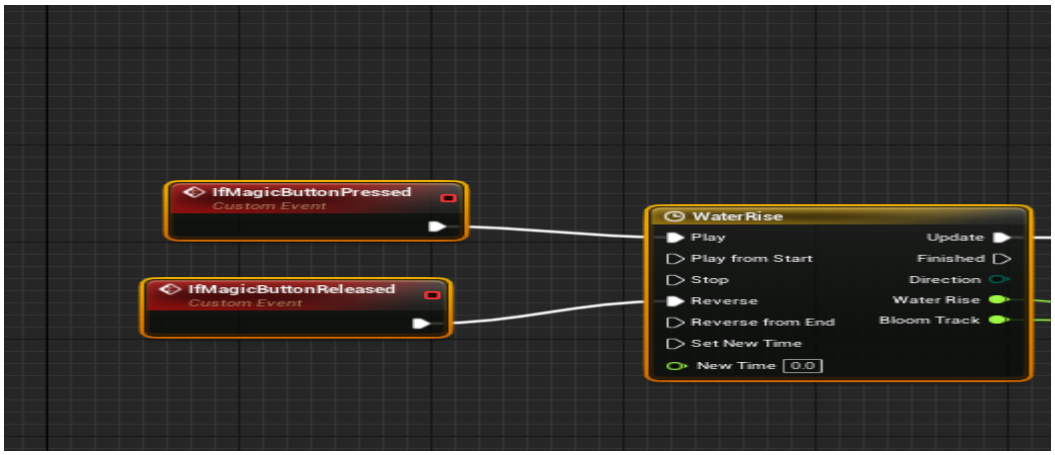

Figure 12. If Magic Input

5. Level Sequencer

The main tool that was chosen to aid in the completion of the Rattlin' Bog project was UE4's level sequencer tool. The level sequencer tool was chosen because of its ability to execute the animations for the characters correctly. A Sequence is played out by a play head which scans across all of the content with in the sequence. The sequence carries out tasks through time. The characters in Rattlin' Bog were modeled to move in an animated series of motions that rely on an exact starting position. Because the characters were constructed in this way, the best way to carry out the tasks of bringing them to life was to use the level sequencer tool. Not only could the sequencer tool be used to correctly execute the character's animations, but it could also be used to gather all sequences into a single level sequence to have all the characters move at the same time.

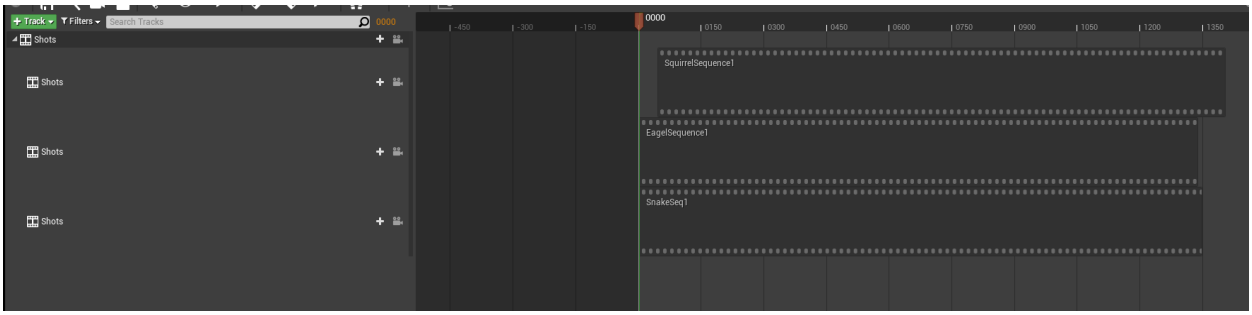

Figure 13. A sequence of level sequences

In many cases, a character will not only have a skeletal mesh, which is like a character's bone structure, but will also have animations such as walking or running in place, and a character blueprint that allows the characters to be programmed. In these cases, character movement can be achieved within the level sequencer, blend space, which is an animation blending tool, or animation blueprints. These tools apply the stationary animation to the character, and then control their placement in the world, called their transform. Characters set up this way can gain motion using UE4 itself. The characters in Rattlin' Bog were modeled and animated in such a way that they did not have just

stationary animations but had full pre-determined animation that were meant to only execute in one way, because of this design, there was no other feasible option but to trigger the animations using the level sequencer tool.

The characters and their animations were created in Maya, because of the way the characters were animated. they project out of their transforms when moving. For example, when the squirrel runs up and down the tree, its transform is stationary below it, and does not move with it. The only character that was implemented in UE4 in the manner described above was the muskrat. Because of this, the muskrat is the only character that is eligible for other forms of animation, and movement manipulation. Using the level sequencer, it was possible to attach a camera to the muskrat's transform and get a personalized shot of the muskrat while it runs around the scene.

At first there were many options considered for implementing the tasks that were needed to complete the project. Animation blueprints work in tandem with a character blueprint, and an AI controller or a player controller. A character blueprint is applied to a character using its skeletal mesh which includes its transform. Then the animation blueprint controls a character's animations using a state machine. The state machine communicates that when a condition occurs in game, the character enters a state of animation; these conditions are called transition rules in the state machine. The beginning thought was to use this method to get all the characters to carry out their animations. It was only a matter of time to see why this method would not work. There were two reasons, the first reason was that the characters, like the squirrel, were leaving their transform behind as various animations were carried out, meaning that there was no way to get the character location to carry out the different states of animations needed for the state machine. The other reason was that all the character animations changing from their aggravated animations when the water is low to their calm animations when the water is high, was based on one condition, the water level. In the state machine that controls the animations within the animation blueprint, there were not enough transition rules/conditions to execute all the characters' animations. Each character had at least two to three animations for each water level. Without more rules to run the state machine, there was no way to use this method. After consulting with teammates about these issues it was clear that because of the given animation assist for all the characters, the level sequencer was the only way to carry out the task of getting all the characters to move correctly. The Level sequencer provides a linear workflow where a character's animations, and transform can be keyframed to provide a smooth transition from movement to movement." Tracks and content in Sequencer are animated by creating Keyframes (also referred to as Keys) with defined properties at specific points along the timeline. When the play head reaches a key in the timeline, the properties are updated to the values defined at those points. Properties can either gradually change, or interpolate, between keyframes, or change immediately to the specified value upon reaching the keyframe."[4] There is a track bar in which you add the character assist. In the figure below you can see where these animations, and movements fit together.

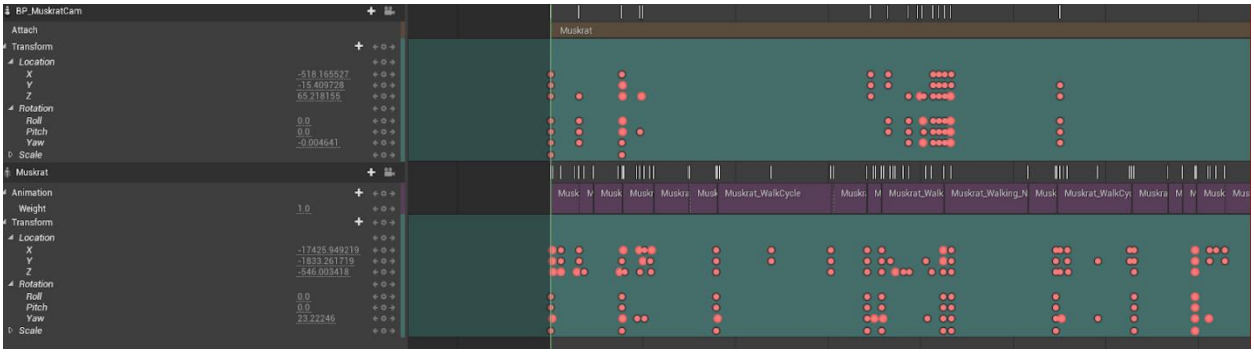

Figure 14. The Muskrat Sequence

Above is the Muskrats sequence. The brown bar at the top of the image is the camera track for the muskrat. The purple bars are the stationary walking animations that have been fitted together, and the orange dots are the keyframes of the muskrat being moved via its transform coordinates through space and time in the level. In opposition to this, is the squirrel's level sequence shown below.

Figure 15. The Squirrel Sequence

There is quite a difference between the two sequences, though they both carry out the needed movements of each character. Because there are no transform coordinates that move along with the squirrel, there are no transform keyframes to provide movement for the squirrel. The squirrel's movement is handled directly within its animations which are seen in the animation track with the purple bars. Because of the difference in animation, the keyframes that are seen in the animation track are used to blend animation. Animations can be executed without the keyframes, but they can provide a smoothness that without them is not present.

The squirrel sequence is the most significant sequence of all the characters in Rattlin' Bog. Not only is squirrel the main antagonist in the game, Squirrel also drives all of the dialog that happens behind the scenes. Below is another example of the Squirrel's level sequence. The green tracks are event tracks. Each of the tracks are responsible for triggering the RSS Dialog to appear on screen as the squirrel speaks with either the snake or eagle. In the section below titles RSS Feeder there is further discussion of how this operates.

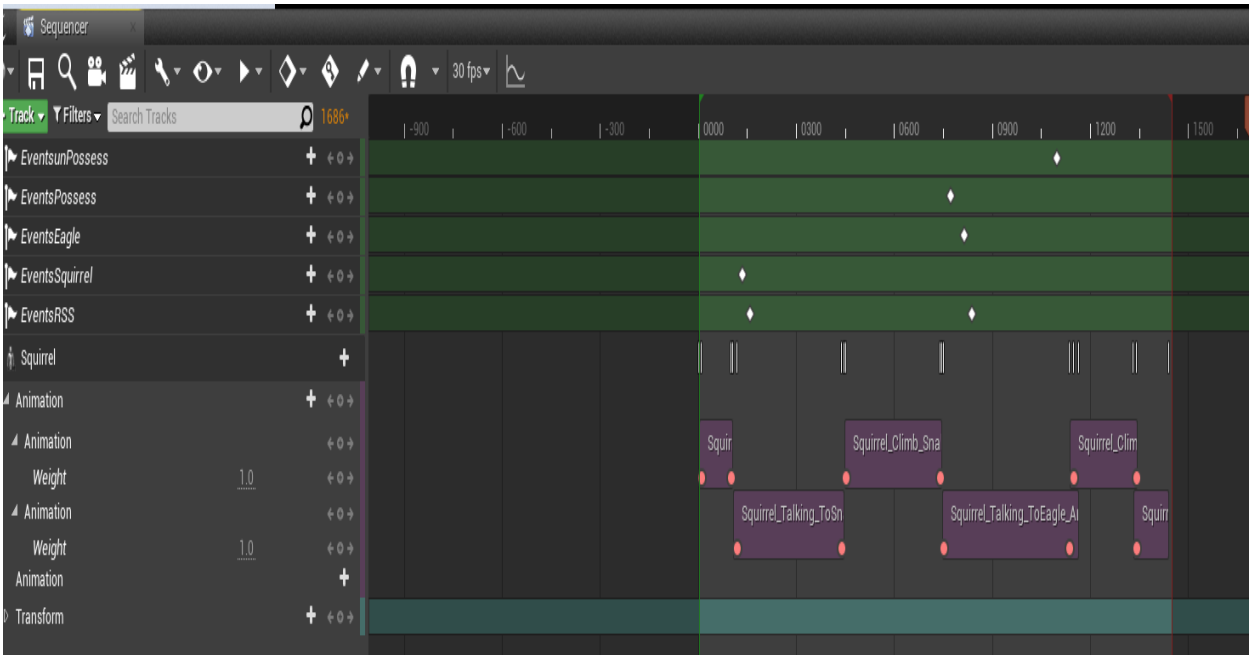

Figure 16. Squirrel's Dialog event tracks

6. RSS Feeder

As the Squirrel character runs up and down the main tree talking to first the Snake and then to the Eagle, RSS feed is displayed on the screen as a representation of what the Squirrel is saying to each character. When the Squirrel is speaking with the Snake, the RSS that is displayed on screen is from Fox news, and when the Squirrel is speaking with the Eagle, the RSS feed is from CNN.

RSS Feeder allows for the use of a high-level API that gets XML from the RSS feed of choice and allows for the articles within the feed to come into Unreal engine. The Plug In is set up to bring the articles into Unreal engine by fetching the articles and using widgets to display the text on the screen.

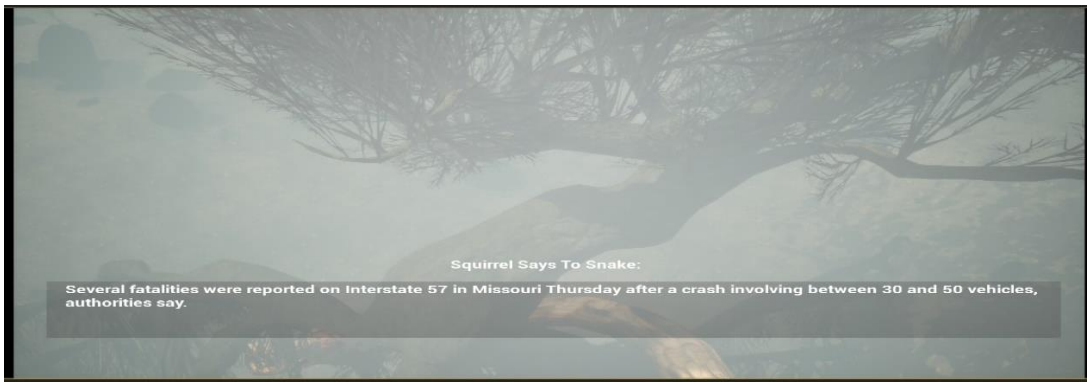

Figure 17. RSS Feed

The RSS Feeder plug in is versatile and modifiable which allowed for the right aesthetic to the Epic Bog environment. The widgets were modified to display text at the bottom of the screen. Above the RSS feed, the design allowed for a smaller text box widget to appear with the heading Squirrel says to Snake, or Squirrel says to Eagle before their respective RSS feeds Appear on screen.

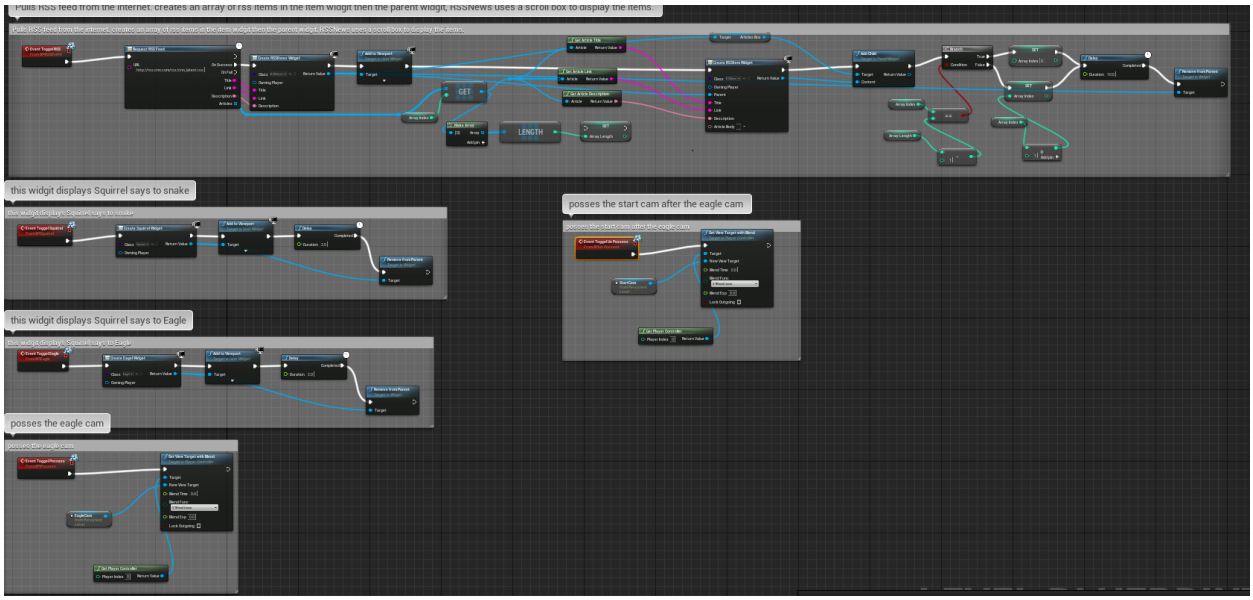

Figure 18. Event Driven RSS Feed from the Level Blueprint.

The communications on the back end that allow for this RSS Feed dialog to take place are that of Interface events that are implemented in the level blueprint. All of the red nodes are the events which are tied to all of the event tracks in the sequence shown in figure 16. This logic allows the text to appear on screen. Figure 19 shows all the interfaces implemented in the level blueprint. By telling the level blueprint that it needs to implement these interfaces, The level blueprint has access to the other blueprints tied to the interfaces.

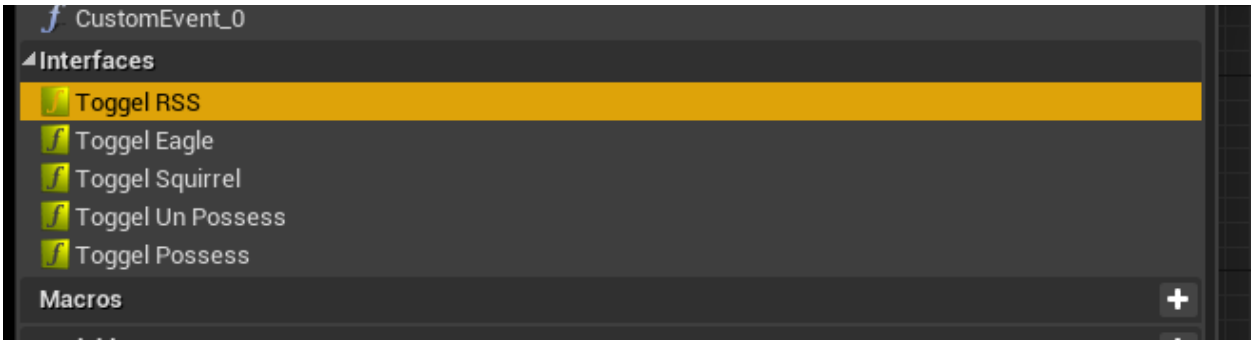

Figure 19. Interface Implementation in the level Blueprint

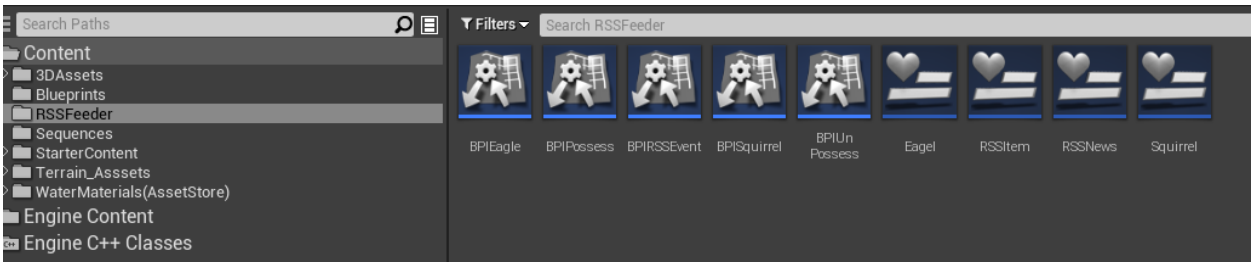

Figure 20. The Interface Blueprints

.

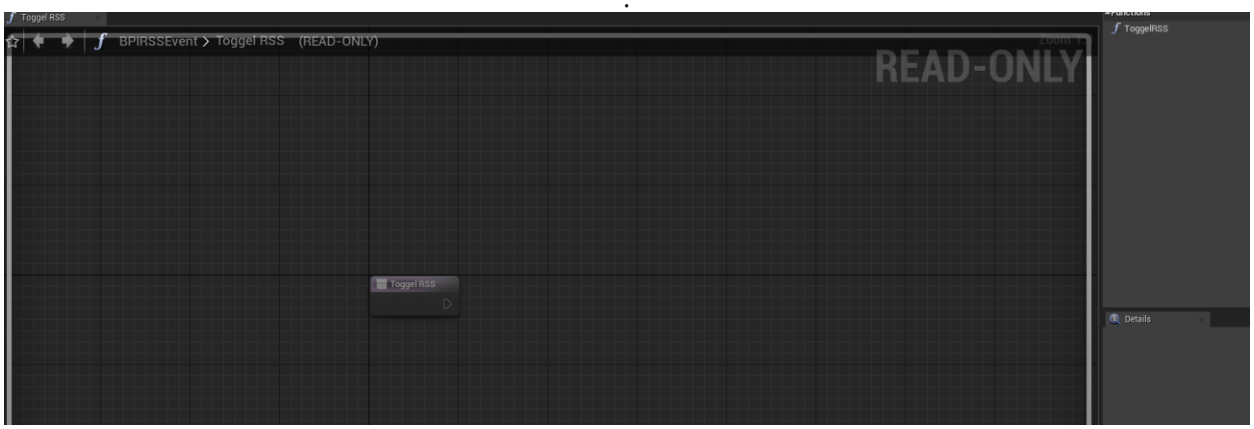

Figure 21. Inside BPIRSSEvent Interface

Figure 22 shows the inside of the event track Event RSS which can be seen in Figure 16 in the Level Sequencer section. The Event tracks in the sequence shown in figure 16 can be thought of as the sender blueprint. By double clicking on the white dots shown in the event track, a developer can create this kind of function which triggers the event that is constructed on the receiving blueprint which has implemented the same interface that the sender blueprint has, in this case, the receiver is the level blueprint shown in figure 18 which has all the same interfaces as the sender blueprint which is the sequence shown in figure 16. It could be conceptualized that the event tracks state that when the key (the white dots shown on each of the green tracks in figure 16) is reached, execute the tied event.

The event is then constructed in the level blueprint, the receiver, which contains the logic for a widget to appear on screen which is the squirrel's dialog.

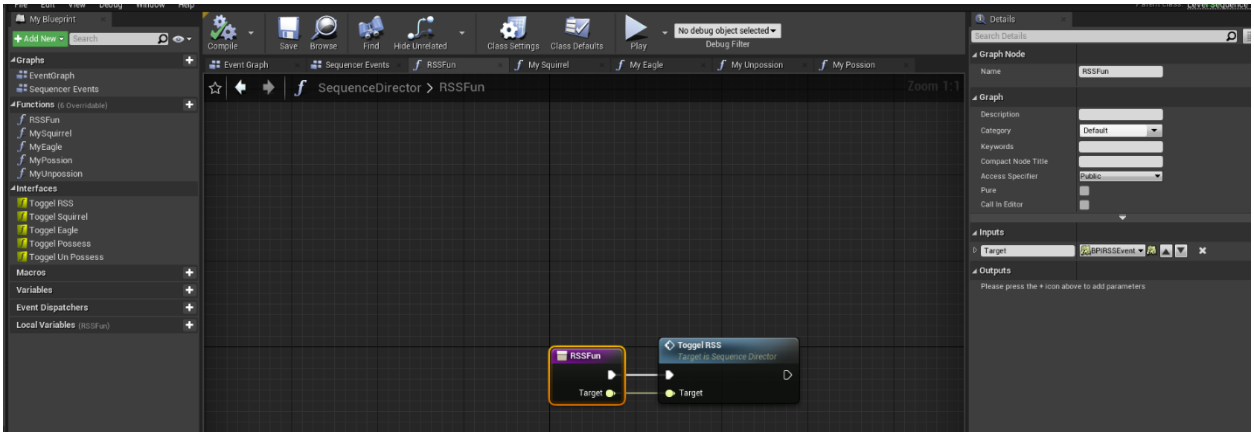

Figure 22. Inside the event track Toggle RSS Found in the Squirrel level sequence

7. Conclusions

The process of writing this paper and evaluating the programming process was that of backtracking and remembering what it was like to not know how the project would turn out, and not fully understanding how to get there. Step by step information was pulled into the paper by recalling the thought process of taking what the game should do, and using intuition, and logic to implement the code to carry out what was needed. The process of creating such a unique game as an art piece like Rattlin' Bog is a solitary adventure. There are not many tutorials online that relate to what we did in this project. For example, most of the tutorials online that relate to animation in UE4 are not like what was used in the Rattlin' Bog project. The tutorials are for Maximo, a popular UE4 assist website, characters, or other UE4 predefined characters like the Manikin that the UE4 templates come with. There is very little information out there for a programmer to use that is based on character assists that are custom-made. This was not the only unique thing about this project. There are not many projects to reference that are using devices like the IF Magic to gain input that controls the animated game world. Creating a physical device like the one that is intended for this project is a rare idea. When a person is part of the development of a project that is little researched, or not implemented often, that person has knowledge or experience that few people have. Having worked on something unique gives a person a unique perspective that could translate to other projects in the future.

At this point, Rattlin' Bog remains a work in progress. There is still work that needs to be completed to realize the project in its filial form. It will be interesting to implement post-processes such as perfecting the project as an executable that can be run apart from the UE4 IDE. Once Rattlin' Bog is Packaged, then the Physical device can then be brought into the project. From there it will only be a matter of tweaking how the physical device cooperates with the game's programming. Once Rattlin' Bog is a completed piece, it will surely entertain all those who come to see it and hopefully make them think about not only the technology that was used to create it but also the ideas it is trying to convey.

Working under Dr. Bradbury was a wonderful opportunity. Gaining the experience of working with the Epic Bog team was beneficial in many ways. The Organization of the project and team lead to collaboration on every aspect of the development process. There were multiple times where reaching out to team members proved to be helpful in resolving an issue. Working with animation was the most challenging aspect of the Epic Bog Project. After Reaching out to Thomas Townsend, the other Programmer on the Team, A solution for handling the animation aspect of the game became clear. Working with the Epic Bog Team garnered Growth not only in programming, and development, it provided a space to build skills of communication and collaboration.

8. Areas of Future Research

The Techniques gained from the development of Rattlin' Bog can be used with several future projects. The tools that brought Rattling' Bog to life can be applied to game development such as VR, AR and other platforms. The roadblocks that were overcome throughout this project are valuable learning opportunities. In the future, if a project opportunity required animated characters, for example, the predetermined method in which the animations were executed in the Rattlin' Bog Project could be improved upon. In the case of Rattlin' Bog it was sufficient enough for the characters to not be blueprinted, and non-programmable, but In larger projects, characters would most likely need to be programmed and manipulated via animation blueprints and state machines. There will be work on the road ahead that will build on top of this learning experience. Game development in this method can translates to purely foundational concepts of programming. Having the experience of using UE4 in this way has allowed for practice of essential skills that can be applied to work such as business or even academic programming. Every project that one can be a part of helps build skills in ways that may not seem important or relevant to other fields at the time, but in many cases will later be reflected on in the moment of need.

9. References

1.SIGGRAPH Conferences, "SIGGRAPH 2016 Preview: An Interview with Jonah Brucker-Cohen," blog.siggraph.org, https://blog.siggraph.org/2015/09/siggraph-2016-preview-an-interview-with-jonah-brucker-cohen.html/

2.idTech, "C++ Vs. Blueprints: pros and cons Which Should be Used, and When?" idtech.com https://www.idtech.com/blog/c-vs-blueprints-differences

3.UnrealEngine, "Material Parameter Collections," unrealengine.com, https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Materials/ParameterCollections/

4.UnrealEngine, "Keyframing," unrealengine.com, https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/Sequencer/Overview/Keyframing/