Primijenjena napredna umjetna inteligencija u Unreal Enginu na bazi konkurentnog višeagentnog sustava

Huzjak, Ivan

UNIVERSITY OF ZAGREB
FACULTY OF ORGANIZATION AND INFORMATICS
VARAŽDIN

Ivan Huzjak

APPLICATION OF ADVANCED FORMS OF ARTIFICIAL INTELLIGENCE IN UNREAL ENGINE BASED ON A SPECIFIC MULTIAGENT SYSTEM

MASTER'S THESIS

Varaždin, 2023

UNIVERSITY OF ZAGREB

FACULTY OF ORGANIZATION AND INFORMATICS

V A R A Ž D I N

Ivan Huzjak

Student ID: 0016131591

Programme: Information and software engineering

APPLICATION OF ADVANCED FORMS OF ARTIFICIAL INTELLIGENCE IN UNREAL ENGINE BASED ON A SPECIFIC MULTIAGENT SYSTEM

MASTER'S THESIS

Mentor:

Bogdan Okreša Đurić, PhD

Varaždin, September 2023

Ivan Huzjak

Statement of Authenticity

Hereby I state that this document, my Master's Thesis, is authentic, authored by me, and that, for the purposes of writing it, I have not used any sources other than those stated in this thesis. Ethically adequate and acceptable methods and techniques were used while preparing and writing this thesis.

The author acknowledges the above by accepting the statement in FOI Radovi online system.

Abstract

The main idea behind this theme is to study and give some insights on how multiagent system can be applied in the task of controlling the artificial intelligence in video games. Video games are the leading revenue actor in the entertainment industry. The most popular way of controlling advanced AI behavior in video games is with behavior trees. Behavior flow in those structures needs to be regulated by set of rules. Multiagent system provides a unique new approach that requires more implementation work up front, but can speed up the rules flow setup later on. Fine tuning values that control the artificial intelligence can get tricky so the mixed approach is recommended the most.

Keywords: Multiagent System; Artificial Intelligence; Game Development; Unreal Engine; Behaviour Trees; Combat System;

Table of Contents

1. Introduction

Artificial intelligence (hereafter AI) in general is a very hot topic these days. The so-called "AI revolution" massively improved domain specific work and research. The acceptance has mostly been positive and people are eagerly waiting for the next big thing. The name itself AI is widely used for variety of applications of some kind of perceived intelligence. But what exactly is AI? The famous computer scientist John McCarthy defines AI like "the science and engineering of making intelligent machines, especially intelligent computer programs." [1, page 2] Not exactly the expected answer which would be more in the lines of particular 'thing' that is performing the intelligent action but surely the correlation can be noticed there. Most basic point of agreement would be that the AI would be some kind of intelligent machine. But what would be intelligent then? Intellect is a spectrum which living things have to some degree. It can be said that a certain animal is smart but it is usually meant that the animal is smart for an animal. Same can be applied in digital world where there are varying levels of perceived intellect. Where is the line between AI and not AI then?

It is not really a nice answer but it depends. Most software systems and applications show some kind of programmed intelligent automation which is the basis for calling it artificial intelligence. One of the most popular web sites Google Translate for, as the name suggest, translating between languages uses Google Neural Machine Translation [2] which is a sequence to sequence AI model. There is not much word around that being an AI. The point that is trying to be made is that there is no one widely accepted and recognized definition. Simple, yet understandable basis for this would point back to the start where intelligent machines or computer programs were the definition, and what is intelligent is mostly up to the interpretation.

What is not up to the interpretation are multiagent systems. This is defined by Carnegie Melon University as "a loosely coupled network of software agents that interact to solve problems that are beyond the individual capacities or knowledge of each problem solver." [3] They can work to solve a particular problem, compete in a given domain or both as it is demonstrated later in this thesis. A single agent in those systems cannot solve the presented problem because teamwork or competition is the main goal of the multiagent system approach. Simulations are a good example of such uses where single agent will not be able to provide much information. Making multiple agents with individual goals and letting them go wild will picture a much more realistic scenario.

This is the approach that has been tested in this thesis in the domain of assisting the control of artificial intelligence via directing the flow of actions in behavior tree to complement the observing parameters of simulated world. In other words adapting the behavior of an AI based on agent competition results. Even thought these agents are competing for the control of the AI unit, they can actually be perceived as working together to create a more compelling and interesting behavior.

2. Work Methods and Techniques

The approach to this thesis is mainly oriented towards research and implementing a working prototype but for the research to be complete there is an additional requirement with regards to the end result and conclusion. That requirement is to create a decent and engaging game loop which can then be tested and observed. Working AI without some kind of capable opponent controlled by the player would not result in truthful or useful final thoughts. In addition to that, since the agents only direct the behavior, there also need to be some concrete AI behaviors that are engaging and challenging at the same time. The last part is also pretty important which is creating and keeping the immersion of the player and for those, some models, animations, visual effects and sound effects will have to come in play.

Tool for creating such experience and through which the degree of usefulness of multiagent system in AI for video games will be answered is Unreal Engine. This tool has been public for quite some time now and is one of the most powerful pieces of software for creating games today. Developer team regularly brings new features which bring in even more jaw drops and expand horizons of game development possibilities. The choice of engine was not only based on popularity but on the previous experience with the engine as well. The results of this experiment could not have been made possible if there was no familiarity with the engine. I have been working on game related projects since I began to understand programming in general and I always tried to push game development related topics in my faculty projects. This thesis is also the indicator of that.

Besides knowledge of Unreal Engine, concepts that were learned during the course named Multiagent Systems, which is not particularly surprising given the name of this thesis, were applied. A simple decision system that all agents implement is designed but their decisions and proposed bids for control vary based on easy to understand variables. Behavior tree control of an AI is in the hands of an agent. Every agent has different behaviors that he advocates for but the one who proposes the highest bid is the one in control. It is not very user friendly nor productive to test the decision system in engine itself. For that endeavour, the good old Microsoft Excel was used.

3. Topic Progression

Understanding what an agent is and how it integrates with other agents to create a comprehensive cooperative structure that can also be competitive in its nature is the first discussed chapter in this thesis. Concepts are meticulously followed by examples that should elevate the adoption of presented content. Once specifics of theory are well understood, specifics of practicality are presented.

A lot needs to be done in order to fully investigate the possibilities of a multiagent system in game AI application and development. The starting point is the player itself. Making sure that the movement feels right and responsive is a key determinator on how to approach the AI itself in game development. Desired outcome is the AI that complements player capabilities and not the other way around. Besides movement, player needs to attack and defend himself against the AI. For that reason a decision to give the player a simple primary close range attack and the ability to dash for defense was made. This is obviously a design choice which is personal and creates a combat that is fluid, fast and with a lot of movement. Most interesting things to tackle in this part are probably the input queue system, animation canceling and the complexity of the animation state machine.

When that is set and done, creating the base animation handling that is independent of the behavior tree itself is explained. There is not a lot of detail here, but a general understanding of how this system works is provided. The difference between handling every animation through animation blueprint and making use of Unreal montages can be examined and broken down into positives and negatives. Montages directly alter the current animation of a character model while animation blueprint only handles animations in stages. There are not as many interesting things here as in the previous chapter, but the expansion of AI functionality that follows shall redeem that quality.

After that, the decision system is created, tested out and implemented in engine itself. Additional testing and adaptation of decision system was done in Excel as mentioned before. The implementation of such a system after the reconstruction in Excel proved to not be that difficult. Then comes the creation of agents that implement the system. They are given specific variables which tell them how to evaluate the data they are receiving.

Lastly, all behavior for the AI character is implemented. Implementation of a player, or rather controlled character, boosts efficiency of designing valid behaviors that player can react to in an interesting way. That is because there is little to no room for guesswork and everything can be tested immediately. Making one reaction the right one for certain behaviors, yet the wrong one for other behaviors, is what creates tension and engagement is exactly what is needed in any kind of combat environment in games. These behaviors are regulated by agents. Formula that drives AI bid calculation was fine tuned based on the actual AI behavior and not a modeled one. All about that process is explained in the last chapter before conclusion. Implementation is done here and all that is left is to play, observe, test, improve and conclude how the gameplay controlled by agents feels. Before all that, a related relevant theory on agents and multiagent systems will be explored in order to provide fundamental understanding

of standing pillars of this thesis.

4. Agents and Multiagent Systems

Before diving into the implementation of the multiagent system which is the core of this thesis, some theory explaining what an agent actually does and where does this specific implementation fit on the known and recognized spectrum of those systems is explored. Examples on characteristics and types of agents are provided in order to make it clear how to classify the system developed as a part of this thesis.

4.1. Agent or Intelligent Agent

Although the word agent has been mentioned frequently in this thesis, a concrete definition has not yet been provided. Up until now there is a high probability that the assumption is for it to be a certain unit related to systems and programming. If you group more of these units under the same umbrella of greater collective responsibility you get multiagent systems [4, page 295]. Question still remains, what defines one singular unit or an agent? An agent is an entity that reacts to changes in its environment [5], so there has to be a system or environment that is relevant to the agent's existence. Agent as a term does not necessarily have to be connected to computer science, the term can also be used to describe physical entities that follow the description of an agent. It is also important to highlight the differentiation between an agent and an intelligent agent. In addition to reacting towards their environment, intelligent agents are also "aware" of their previous encounters with the environment [5] and they formulate their reaction with those observations as additional factors in decision making and fulfilling a set goal.

To better demonstrate these and further characteristics that agent either has or can possess, a personal project is brought up as an example where every property will be observed on a concrete system. More information about the project can be found in the last appendix of this thesis [3]. Firstly, there is a short description of what the project does to provide a contextual lens through which agent attributes will be observed. Project is named LoL Overlay System (LOS from now on) and, as the name suggests, it places a certain overlay over League of Legends client application, which serves as an interface for finding and configuring game related data. To put it as briefly as possible, some non-essential but still important data for some users is not shown in the application itself and that is where the overlay steps in. How this looks can be observed on Figure 1. Numbers next to hero icons as well as little chests and progress bars are a result of the working overlay.

The overlay examines the client application and looks for images of characters that are rendered on the screen. It then searches for them in its memory to identify them. After identification is complete, it retrieves the corresponding data for that character and it displays that information over the client application. One representative property of an agent is the environment that is beyond the agent's control. Another representative property of an agent is a system that reacts to the environment regardless of the defined demarcated area. How would that agent become an intelligent agent or is it an intelligent agent already?

Short description of the system provided no evidence of the system being an intelligent agent. If a system has no recollection of its encounters with the environment to which its functioning would adapt there is no intelligent agent present [6, page 44]. This marks the first improvement in performance of the LOS algorithm for identification of characters, which shall not only demonstrate that systems can objectively get better at performing the job at hand but that it also improves user experience. There are over 160 characters that could appear in the defined locations on the client application. There is no 100% match when it comes to the character image that is being identified and the one in the memory. The match percentage can vary based on the image itself, so the algorithm has to go through all the images, even if the first one is the corresponding match, and check if the match percentage is greater then the current greatest. Performing this task for all 15 locations where an image can be takes some time and slows the responsiveness of the overlay system. At that moment in development, it needed 1 to 1.5 seconds to complete one pass through all locations where an image can be. Locations where images appear are constant, however, their content is not. Images that represent heroes can switch locations so the system has to be able to complete a cycle at least a few times a second to feel responsive enough. Some locations where images can appear are not populated with an image right away.

This problem was addressed by introducing memory that stores character information of characters that were previously found on the screen. Once the character selection screen is displayed, the first 5 image locations are populated, and there is no way around this. However, information about these characters is now saved in separate memory, which is used to quickly make the same check operation. What was previously an effort of checking all 15 captured areas against over 160 possible matches now, after the initial one, becomes a task of checking 15 captured areas against just 5 to 15 possible matches. That is more than 90% increase in efficiency, reducing the time for completing one loop to as low as 150 milliseconds on the top end. This is still a lot of time for a piece of software, but for a better solution, better algorithm for matching images must be used. There were a lot more optimisation steps, but this was the most significant one in terms of performance and it nicely pictures a translation from an agent to an intelligent agent. That is because the LOS system as an agent is now aware of its previous encounters with League of Legends client application.

4.2. Intelligent Agent

Through previous descriptions of intelligent agents there are some descriptors that stand out as strong indicators of the agent's existence. One of those is autonomy [7, page 39]. At no point in time does LOS need any kind of human or other interaction to continue to work properly in achieving its goal. System detects when heroes appear on the screen, identifies those heroes and shows the corresponding data to the user. It detects when there are no heroes shown, it empties the quick access memory for those that are frequently used and awaits further appearances. LOS can detect movement of application clients on the screen and follow it correspondingly. It can also detect a change in resolution of a client application and adapt its comparison library. All of these are indicators of autonomy characteristics that agents

Figure 1: LoL Client with LOS

possess.

Another common characteristic of an agent that is not as obvious to define and notice is rationality. It is often used to oppose the term omniscience [7, page 38], which cannot be realistically achieved. An agent that is working perfectly on any given task to make the best possible decisions at all times is not a realistic standard of rationality. Changing the "at all times" in the previous sentence to "as much as possible" gives a more achievable look and standard that can actually be implemented. To put it simply, rationality is goal-oriented behavior and decision-making. Described behavior of LOS fulfills that criteria where finding the right data is the main task from the agent point of view; the outcome is the rendering of that same data to the end user which is the most important element of LOS for the user.

The last characteristic that can be attributed to intelligent agents is learning [7, page 55]. This is the main distinction that would truly separate most software systems that fall into the agent or intelligent agent category from the ones that can be undoubtedly identified as artificial intelligence. These are all forms of some kind of artificial intelligence but not all of them carry its fullest potential. Humans are classified as intelligent but so are mice or squirrels, the degree of that intelligence is not the same and that is the point being made here. Process of learning in intelligent agents as a territory is vast and is the main field behind development of large AI models that have taken over society and are hugely beneficial mostly in aspects of assistance for now. Those models will surely be capable of doing fairly complex tasks on their own without any guidance or help form human observation.

Overlay does not have any capabilities on that end but it does show some short term "learning" capabilities which greatly improve its performance. There is no room for advanced kind of learning behaviour nor is there a valid need for it. Agent reacts to the state of the application and is well versed in that closed environment. Nothing it does would be vastly improved by possessing extensive learning capabilities as it is perfectly fine without those. This

goes to show that not every characteristic needs to be implemented to have a robust functioning intelligent agent that is well equipped to fulfill its mission.

4.3. Intelligent Agent Types

Complexity of any given agent is not only an indicator of invested time and great capabilities but is also determined by the goal and environment that agent is expected to work in. As could have been seen by the LOS example agent which had no need for more complexity that would elevate its performance. That is the basis of agent types that are about to be discussed. Some of those types shall be recognized in previous discussions about specific reactive nature of described LOS application. This classification is following the Russel and Norvig breakdown of agent types [7] but is more focused on concrete actions and states that agents can have depending on where in the reaction process it currently is. This shall be understood more clearly with following diagrams that describe each of those types.

4.3.1. Simple Reflex Agent

The most basic system that can be classified as an agent is the one with 3 distinctive qualities. It has to be capable of observing its environment in some way. Secondly, it has to be able to consider both the environment and set rules in the process of making a decision. Lastly, it is mandated to possess the ability to interact with the environment in some way. Interaction with the environment from both sensory and interactive perspective can be individual or as a group in case single agents are components of a multiagent system.

Exactly that is the basis of the first agent type that is known as Simple Reflex Agent. They have a defined way of environment observation and rule appliance [8] and that is what they do without any other modifications or considerations. These agents do not have any capacity to process any additional operation optimisation then what is already in the developed algorithms.

There is a figure 2 that shows exactly those elements of one agent. There are two points where agent and environment have interaction between each other. One is an observatory in its nature where the interaction is based on agents observing the environment to collect crucial data that it requires for functioning in a designed way; to be reactive or act reflexively. There can be one or more sensor elements that are instructed to collect valid data and that data creates a picture of the observed environment. That information together with specified rules dictates the processing outcome. With finalisation of processing comes the second interaction point which is agent acting upon the environment. This ends one cycle of reaction or rather a reflex. It encapsulates the process in a way that the simple yet descriptive meaning behind the agent type name emerges as a clear distinctive term.

When it comes to the example application, it is clear that the first version of LOS which had no quick access memory was indeed a simple reflex agent. It had no recollection of anything other than the processing itself and its reactive nature to the data provided by its sensory

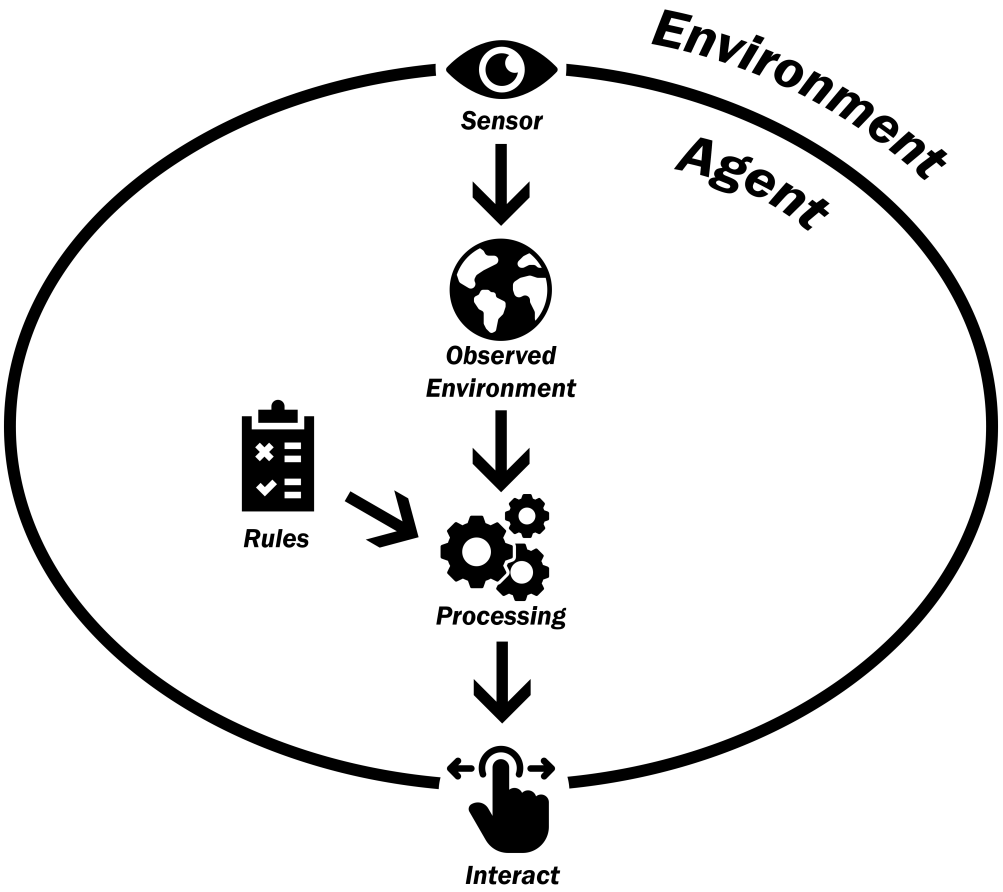

Figure 2: Simple Reflex Agent

observation of the environment which are elements of the captured screen. It finds corresponding data based on defined rules and identifiers. Found data is shown to the user in the next interaction process which concludes one cycle and that process is visible in figure 2. There can also be more than one interaction action that is done by an agent. It depends how it is looked at but if one data type or one data point of rendering is considered a different interaction it can be said that LOS has multiple outward interaction points.

4.3.2. Model-based Reflex Agent

Next type of agent raises complexity which shall be recognized as a trend in this categorisation process. The basis regarding observation, processing and acting stays untouched but there is more entanglement present inside the agent unit which can be witnessed in figure 3. Most important difference is in having memorization capacity of previous environment observations. This gives agents an essence for comparison as a factor in the decision making process. This agent is named Model-based Reflex Agent.

Figure 3 visualises creation of awareness of the observed environment that comes from sensory data. On the first run of the program there is no additional available information since the basis of previous states cannot be established in a single run so nothing changes. Every consequent run, depending on the implementation, brings forth new correlational information.

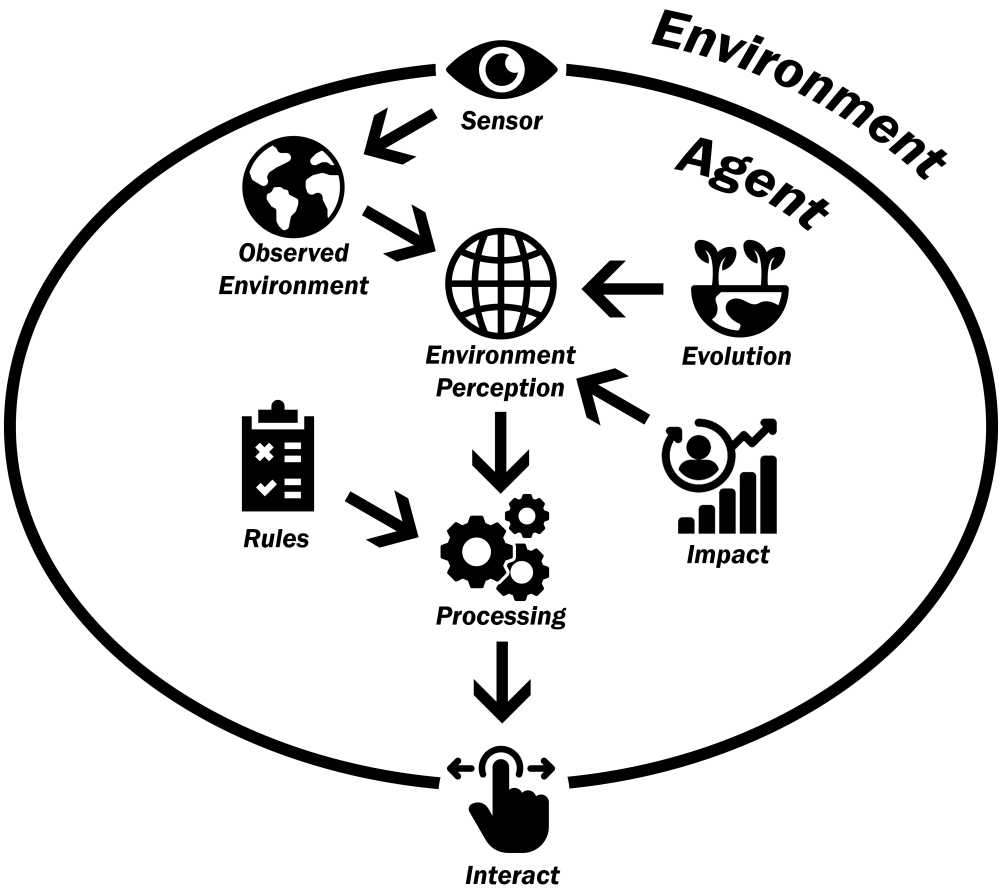

Figure 3: Model-based Reflex Agent

That information falls into one of two possible categories. It can be interpreted as change in environment or how the environment evolved from one or more previously memorised states into the current one. It can also be seen through the lens of how certain agent actions affected the environment or what kind of impact did agents leave that is reflected as noticeable change in the newly observed environment. Both of those influence the environment's perception. The rest of the figure 3 is the same as the last figure 2. Certain rules are followed while considering the environment perception to make a decision on an interaction that should end the cycle.

It can be noticed that LOS has most but not all depicted elements that outline the Model-based Reflex Agent type. After the expressed upgrade with quick access memory which was originally only purposed for improving the dreadful performance, that system as agent has also evolved in type classification. The moment that it can modify its behavior ever so slightly based on previous encounters with the environment is the evolution point. It would make a better example if there was more change in behavior rather than just in speed but it does stir into the right direction. There is, however, no available interpretation of agents' own actions and impact it leaves on the environment. This cannot be done within the circumstances and the function in which the agent operates. Results of processing are finite and definitive even though guess work is present. Environment is not reactive to the agent's inputs and this fact alone is the main cause behind the uselessness of any kind of measuring of agents' impact on the environment it observes.

4.3.3. Goal-based Agent

More profoundness comes with more complexity when it comes to agent types. Additional complexity can be detected in figure 4 where the inside of the represented agent type is becoming crowded. Defining a step towards new type independence is proactive behavior. That means that an agent is capable of perceiving how certain action he takes will affect the environment or its progress towards achieving set goals indicated by positively interpreted changes in environment.

In comparison of figure 3 and figure 4 there are basically the same elements in the initial stages depicted by figures which come to an end with environment perception. Here arises a new element called testing in the figure 4 which represents the testing nature of this agent type. Agent can have one or more actions that it can plan to invoke but it needs to confirm that that particular action will elevate the state of the environment. When testing that hypothesis both perceived evolution of the environment and the recognized impact agent left on it come into the testing process together with current or lastly observed or the most relevant environment perception. This is one of the differences that separate Goal-based agents from Model-based reflex agents. The second difference is a different approach in expressing rules. These are no longer dead set and are more unbound or open to configuration. This means that the agent itself is more flexible and mainly geared towards accomplishing wider goals then following a specific set of rules. Restrictions still exist but they are applied with less enforcement especially if planned actions lead to goal achievement. There are constraints about states that are outright prohibited but agent is more malleable. Defined goals feed into processing which values the result of action testing as well. Decision made is observable in the interaction that agent decided to take.

Example projects fall short on this agent type. It just does what it is programmed to do without questions or any kind of testing. There is some proactive behavior but it is related to the system setup rather than the agent processing cycle itself. In the setup phase of LOS, the application checks the latest version of the application client available and compares it to the latest known stored version. If there is a new version available there is a chance that some heroes have their images updated or that there is a completely new hero available which also has a unique image. In that case LOS updates its comparison library before overlaying any kind of information to the user. This is a proactive behavior but it is not in the algorithm of the running agent and it does not affect the cycle itself which disqualifies LOS as a Goal-based agent.

4.3.4. Utility-based Agent

The main blind spot of previous agent type that can be alleviated is quantitative measurement of action contribution towards one or more goals. In other words, by how much does planned action augment the environment state in the direction proclaimed by set goals as ideal. It can also be called the progression quality of certain actions. Goal-based agent does not obstruct any action that reflects positively on goal seeking properties. That is kind of a problem

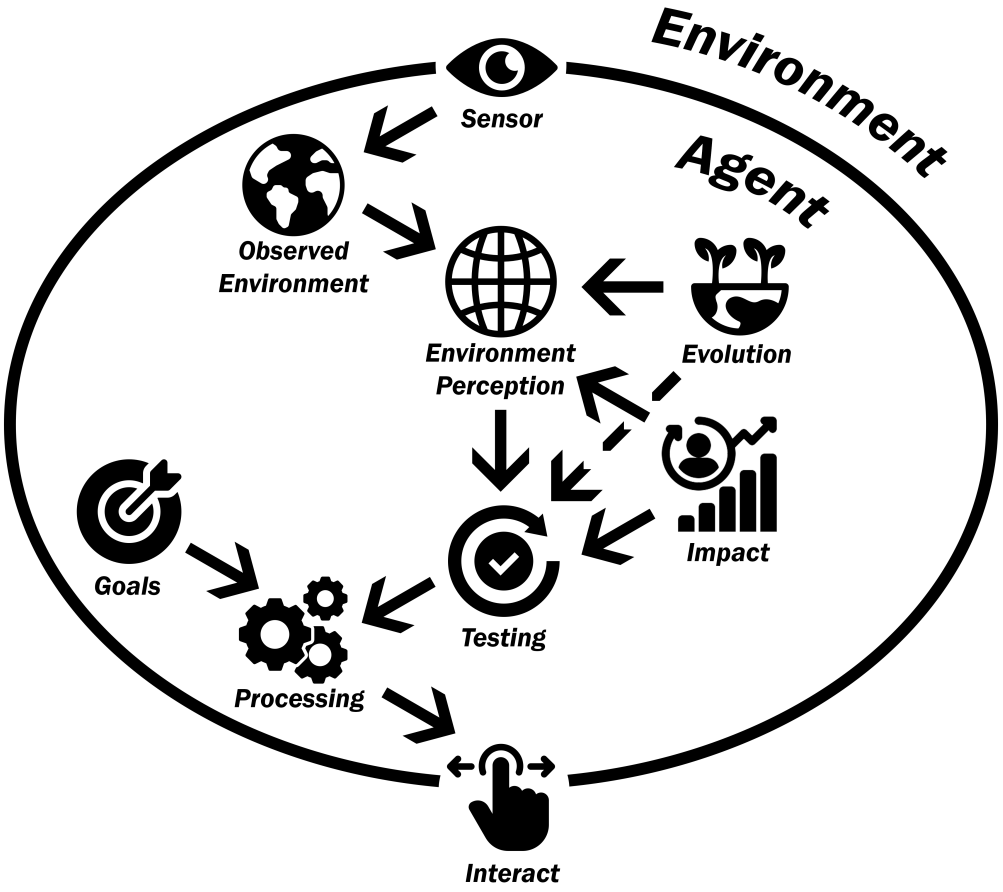

Figure 4: Goal-based Agent

since it does not find the best possible discernible solution for current circumstances, rather it provides just a solution. While that is not bad in any way, it can be optimised and it is done by additionally comparing solutions which are labeled as measuring in the figure 5.

Most of the figure 5 is the same as the figure 4 so the focus is more fixated on the differences which are evident on the left side after the known testing element. This phase discerns the action that can be taken in order to create the most probable circumstances for zeroing in on the outcome that has the greatest utility potential. For that same reason, goals are now even more substantial then original rules or current objectives. This redefines goals into utility which holds measurable data for action quality assessment. The rest of the Utility-based agent elements are the same as in Goal-based agents which derive the needed action and interact with the environment.

Since LOS did not really qualify for Goal-based agent type there is not much that can be described in that example for Utility-based agent behavior. However, evident proactive behavior can be further explored. In that concrete example, agents could be making assumptions about user behavior and hide features that are not detected as useful. Utility of the agent would be to provide the user with all requested data but also maintaining client application visual clarity as much as possible. This would end up being user specific since users perceive overlay utility based on their own needs.

In this last category there is an agent type that is a subcategory of Utility-based agents and that is Learning Agent [7, page 56]. Utility-based agents make decisions based on the expected utility of actions while taking into account the goals and preferences of specific agents. Learning mechanisms, such as machine learning algorithms, can be integrated into these agents to enhance their decision-making abilities over time and that is what makes this final distinction. These capabilities are not bound to be part of Utility-based Agents, they can be incorporated into other agent types to introduce adaptive learning capabilities. Important takeaways is that all agents have a structure that involves 3 main departments and that are percepts, reflex system and actuators [7, page 1048].

4.4. Multiagent Systems

Concept of multiagent systems was explained earlier in the thesis but some characteristics of agents inside those systems also need to be addressed. Aggregation of multiple agents that work together under a leader or on their own to achieve the end goal of a system in a way that single agent cannot is the basis of multiagent systems. That concept can be applied on LOS as well if every presented image that needs identification was offloaded to a single agent. Multithreading would surely improve overlay performance even further but would impose an additional challenge when trying to synchronize known data. It would also bring new conundrums regarding memory control and how to regulate access for 15 agents to the same memory or to, perhaps, create some copies for greater throughput or flow rate of the algorithm. Anyhow, it is clear to say that it would complicate the system by a strong margin and that is the reason it was not considered as a possibility since performance improvements were predicted to be not

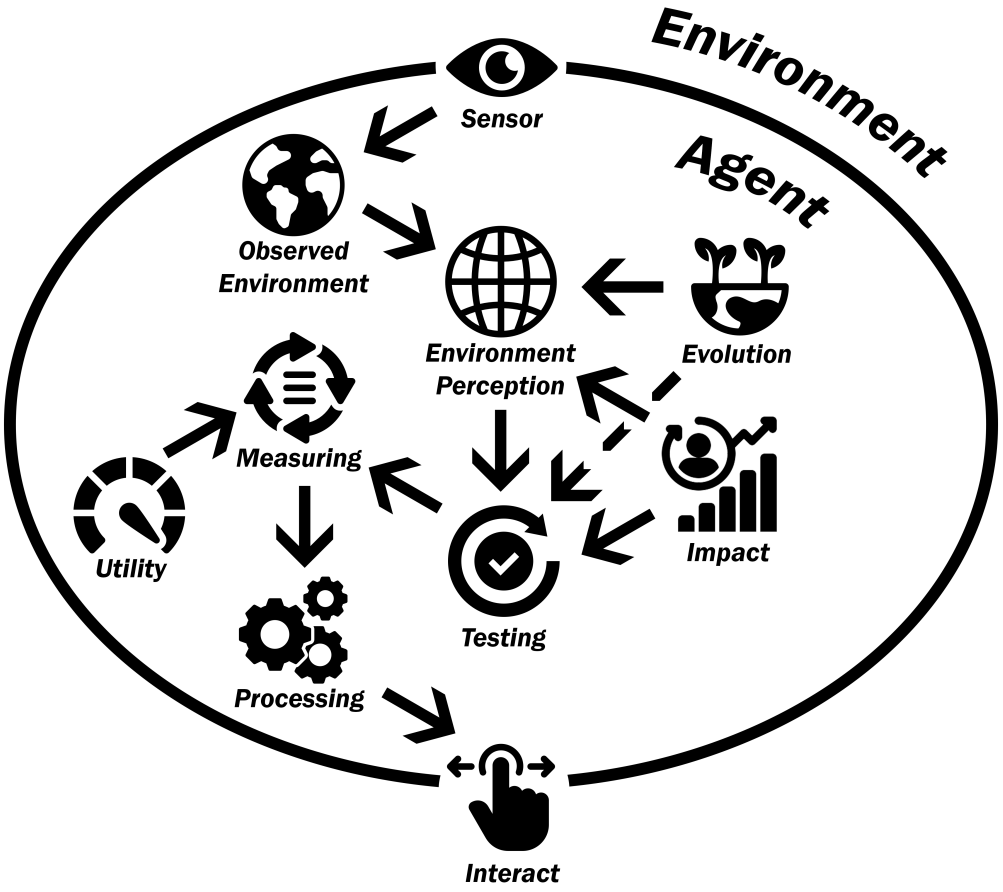

Figure 5: Utility-based Agent

that substantial.

These agent characteristics inside multiagent systems follow [9] classification into features. One of those features was teased earlier and that is the existence of a leader agent. System developed with this thesis does not have a leader agent per se. It has an agent that is currently in control of AI behaviour and it does have a system that gives control to one of the agents. None of those should be considered as leaders since in the first case, the agent in control is not only in control but also other agents do nothing when it comes to influencing AI behavior. They still compete in order to win the bid and take control but they cannot act. The system that gives control does not actively participate in the multiagent competition and is not even aware of what those agents are fighting for. When it comes to awareness, behavior (or bundle actions) is separately implemented from agents themselves, so neither the agents actually know what their victory does in the virtual world. This system would then be considered leaderless.

Next feature is what I would call concreteness or outcome concreteness. This encompasses both systems where an outcome is always clearly understandable from input parameters and the ones where different agents come with different information about the same data and some interpretation, correction and decision has to be made about which of those is more and which less accurate. Since agents in developed systems all get the same information about the environment parameters, for performance reasons, they would be considered as linear inside that system. It would be an interesting experiment to introduce some hand induced errors and make agents send both their bids and the data that they observed which would be a second variable in evaluation. This is called [9] Decision Function by the authors of the division.

Similarity or disparity in agents regarding their functionality but also characteristics and types is considered a feature as well. System that has basically the same agents is called homogeneous. Developed multiagent systems are homogeneous even though the behavior that rivalry between agents produces is not similar at all and could be considered the opposite which is heterogeneous. Systems with dissimilar agents would be more complex and potentially more resource intensive so, since available control discrepancy between usual behavior control and that of multiagent system was incomprehensive at a time, safer option was chosen as an aspect of implementation.

Recognised metric of evaluation is one of the features of agents in multiagent systems. Based on that feature agent can be identified to be of first, second or high-order. Identification number basically represents the number of parameters that all agents acknowledge, agree upon and potentially exchange when evaluating the environment. If that number is higher than 2, then the system is considered to be of high-order and it distills down multiple parameters into one single representative value. Developed system has a bid variable that all agents follow and recognise since all of them create a bid with data that is available to them. This would classify developed systems in this feature as a first-order system.

Programmed ability for agents to consider delays is unimportant for the developed system but is a feature none the less. This envelops delay handling capabilities based on their continuous presence or relevancy for the system at large. Needless to say, if delays are not

expected or rather they are not a norm, they still need to be handled in a defined way. In developed multiagent systems that would be a timeout of the control unit in its allocated await time used for expecting agent messages. By these standards, a system would be considered as without delay since delay is not a recurring theme, and when it does happen, it indicates a lot more present performance problems.

Another feature that can be present in multiagent systems is known as topology or switching. It focuses on the overall picture of the multiagent system when it comes to the presence of identifiable agents. Topology is considered static if relations between agents remain the same as well as agents who find themselves in that group. If agents can join or leave that specific system, then it is considered dynamic. In developed multiagent systems agents fight over control which can be physically seen as a behavior in the virtual world but they continue to fight for that power once they are in charge so their relations do not change in any way. That would classify them as agents in a static system.

Messaging between agents can either be continuous or event driven. If data transfer happens on a regular basis then the system is considered to be time-triggered. On the other hand if messaging is primarily event driven then the system is considered to be event-triggered. Data collection is done inside a separate entity when it comes to a developed multiagent system. This data is available to all agents at all times since it helps them to create their interpretation of value that they can provide to the behavior of the AI unit. Even though those messages do not flow every tick, since that would be ridiculously frequent and hardware intensive, it still means that the system is time-triggered.

Last feature by [9] defines agents mobility. If an agent can change its position in the environment then that would be considered a mobile agent. Likewise if that is not the case then that agent would be considered static which should not be confused with static topology of a multiagent system that was already discussed. On that metric, a developed multiagent system has agents that are static. That might seem unintuitive at first since agents do control the behavior of an AI unit which moves in virtual space but that unit is not part of that specific set of agents that fight over control of the decision making process.

When it comes to multiagent system development roadmap, there was no clear objective evidence that there is single or multiple good approaches to that endeavour since usually authors prefer their approach to any other [10, page 371]. Concept of multiagent systems in game development for behavior determination is not a particularly popular field of research. That is to be expected since it means less control and less control leads to either worse outcome or more development time. If the main goal is to create a somewhat randomised behavior that is not affecting the main game loop then it is fine, however, something like combat which usually represents at least a significant chuck of the main game loop becomes complicated.

It was decided to create a player movement and abilities before diving into the complexity that multiagent systems bring to a "simple" behavior tree. Comprehensive player capabilities became a basis for enemy AI behavior which was then tested, abbreviated and divided into categories that hold those abilities under the same overlaying idea which was then delegated to one single agent. Every agent is self interested but its main goal extends that of his own which

is creating an interesting experience for the player. This is an example of competitive nature inside of the multiagent system based on self interest. In life, self interest can be perceived as favoritism [11, page 48] when interacting with other agents even if that is not considered at all when making decisions. Same happens in this system where certain parameters favour different agents in the name of creating the most intriguing experience possible.

5. Player Implementation

The first on ToDo list is the player but nothing can be done if there is no project to work on yet. Unreal Engine version that is used for this project is 5.1.1 but the code should work in all of the versions of Unreal Engine 4 and above. To use Unreal Engine, you have to have Epic Games launcher installed which is a software that is a combination of game distribution, Unreal Engine compatible assets distribution and the Unreal Engine distribution.

In section Unreal Engine of the Epic Games launcher there must be at least one version of Unreal Engine installed to be able to start creating projects. There is no need to start on a blank project when certain minor steps can be taken with a preset. The development version of a project comes with some unnecessary starter content as well but this is not relevant for the research at hand. The chosen preset is Third Person Preset which comes with some basic input handling for the player and a basis for player character. Everything player related is kept in the Player folder so the player blueprint called BP_ThirdPersonCharacter was moved into that folder, together with Input Actions and the Input Mapping Context. Those together make a new way of handling input, which is a bit harder to comprehend at first, but ultimately gives a lot more fine control over inputs. Those can stay like that since they create a basic character movement but for additional inputs the old Input Bindings are being used which can be found in Input section of Project Settings. There is a lot of UI inside of Unreal Engine and if all details were accompanied by a figure there would be no end to this thesis.

Now that everything is set up and ready, the preparation for the code implementation of the player can begin. One last thing is important to mention before the implementation itself though. All programming logic is written in so called Unreal Engine Blueprints. This is a visual programming language that is based on nodes and connections. The easiest way to understand this is to think of nodes as representations of functions. Even variables that are present here can be seen as variation of their get or set functions. Math operations such as addition, division; arithmetic operations such as greater then, less or equal to; logical operations such as not, and; actual functions, macros, events; they all are like functions which have an input and produce output.

Connections signal which functions communicate with other functions through variables. In other words, output of which functions becomes an input to another one. Variable types of these connections are color coded, so you always know that the color red represents the boolean variable. The connection that symbolizes the flow of program is called an Execution Line. This is the equivalent of the new line or semicolon in usual programming languages.

Important caveat to have in mind is that the Execution Line which connects two Execution pins is the only line that represents the current state of execution in a program. All other pins from which a reference can be dragged out to a particular variable is only that, a reference. If a simple add one to integer function is created that takes the given integer and adds one to it, that node has one integer input and one integer output value. To demonstrate the point a variable called StartNumber is created and set to one. Calling the function three times and setting the value of StartNumber to a new one after each is a starting point. Printing the result

out by using the print node and pulling the reference to print the number from each Set Start-Number node results in the same number 4 printed out each time. Even though the number should be different each time, print statement in the Execution Line only gets the reference to said variable and, at that point in execution, that variable is always four, no matter from where the reference is provided.

5.1. Preparing Player Blueprint

For the preparation phase components are added to the main actor component. Actor in Unreal Engine is the simplest entity that can exist inside the virtual world or level that can have both the visual representation and the execution code as well. You can have three dimensional (hereafter 3D) models inside of the level that do not execute any kind of code but those are usually reserved for the environment which is not covered with this topic.

Actors can be of different types, which come with some nuances that are good to know and recognize. From the title of this section you can see that there is character in the name. That is precisely because the Character is an actor that has human-like movement generally speaking. That is because upon inspecting the Default Land Movement Mode of the Character Movement component a movement type called Flying can be seen. The step below that is the Pawn class which the Character is derived from and this is used for non-human like behavior that can be possessed by the player. This is when a player, for example, enters a car and can then drive it with different input controls and consequences of those controls. That car is of Pawn class.

Components are the sub-elements that provide actors with visual or interactable elements that can be modified and referenced in the code partition itself. These can be found in the Viewport tab of every actor. Every actor has the basic three tabs that are used to modify it. These are the mentioned Viewport, Construction Script and the Event Graph. Both Construction Script and Event Graph are places for programming with described Unreal Blueprints. The only difference is that the Construction Script is a basic programming constructor that runs the code before the actor has even appeared in the level. Code that takes as much time so that the difference between Construction Script and Even Graph can be noticeable is not used. The only difference besides that is that the Construction Script does not have access to functions that modify or take into account the tick of the game. Tick is the heartbeat of the game that takes into account the processing power of the machine that the software is run on.

Firstly an Arrow component is added which is only a single point vector that can be used to test rotation of the character. There is a difference in rotation when standing in place and moving the mouse around between the actor itself and the appropriate visual representation of the player or mesh. Mesh is a 3D model that has bones and weight-painted relations between bones and 3D model itself. The process of creating such an instance from a 3D model to an animation ready 3D model is called rigging. In order to be able to correctly resize the Capsule component for collision a mesh must be added. The chosen mesh is Paragon Kallari character that can be downloaded for free in the Epic Games Unreal Engine Marketplace. All used assets

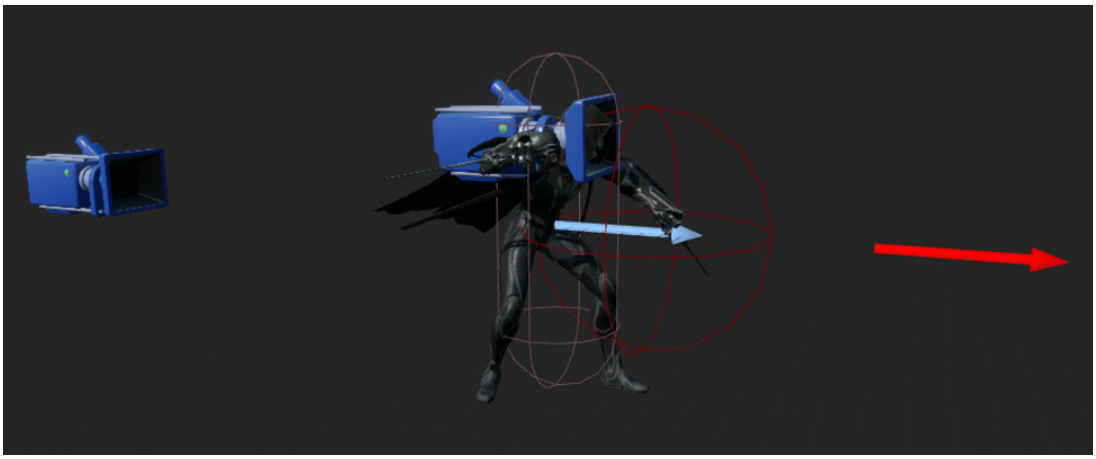

Figure 6: Modified BP_ThirdPersonCharacter Viewport

are mentioned and linked to at the end of this thesis.

The character remains in the T-pose if no Animation Blueprint is provided. Animation blueprint is named ABP_Player and one temporary State is created that has a name Idle. Idle animation of mentioned character is added to that state. More detail about the animation blueprint is provided later on, so this is just a placeholder for now. After that, in the mesh component details ABP_Player is set as the active animation blueprint and character is no more in the T-pose. Now the Capsule component for collision can be adjusted. This capsule is the main component of the actor and is called a Root component. All other components have their place in the local space in reference to the root component.

Other than that, Camera Boom component is adjusted the to be more fitting for the gameplay and another Camera component is added and attached to the mesh. This provides a basic first person gameplay capabilities so that the game loop can be tested like that as well. Other than that, another invisible component for damage system is added. It is placed in front of the character and is used for checking the collision in the events of the attack. No specific values of location, rotation, distance or scale of every component that that was modified or placed is provided but the end result can be seen in figure 6.

5.2. Manual and Automatic Camera Switching Systems

Since there was a lot of talk about cameras in the last section, that is the functionality that is described first. To the event Begin Play, which triggers the moment the actor is present or appears in the world (in this case immediately), two Set Timer By Function Name nodes are connected. These create individual threads that help in the process of achieving the sought after functionality. Details of these nodes can be seen in figure 7.

It is important to mention that in Unreal Engine there are functions and events that both act like functions. The main difference is that in inheritance the events cannot be overridden and functions cannot have nodes related to the tick, same as the Construction Script. The Track Distance is an event that simply sets the Camera Distance variable in set time interval and can

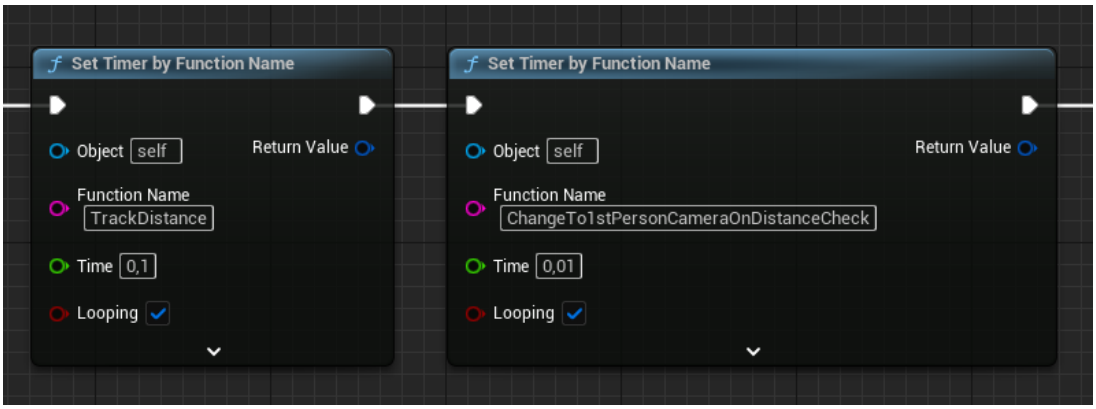

Figure 7: Timers for camera switching

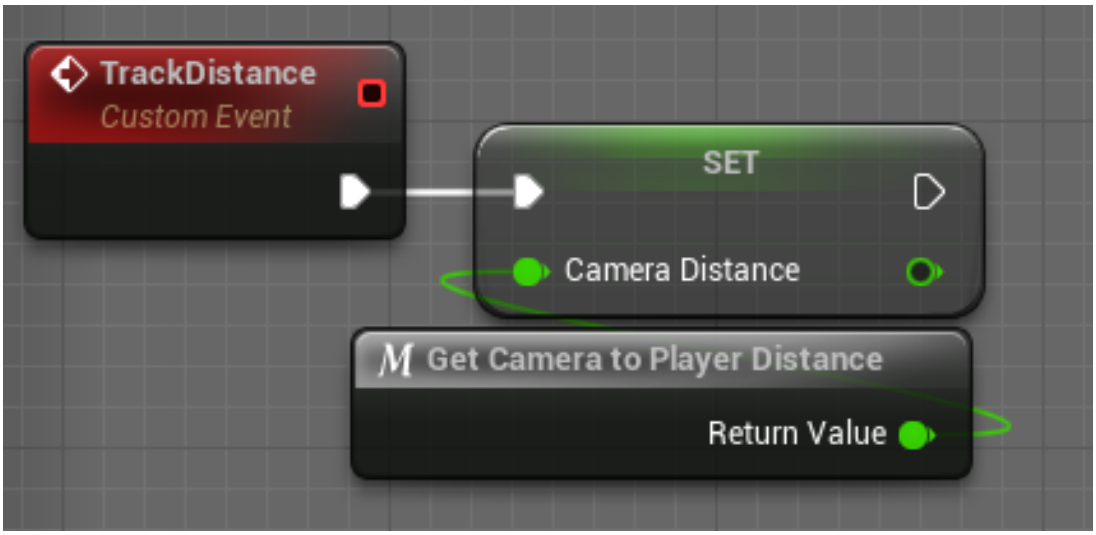

Figure 8: Custom event Track Distance

be seen in figure 8.

In the same figure, you can see the Get Camera to Player Distance, which is a custom node that is referred to as Macro. These Macros are mostly used to condense more nodes under a single name. Functions do that too, but in Macros, usually a code or nodes that do not have an execution line are written. In the figure 9 you can clearly see that none of the nodes presented have an execution pin that was present in figures 7 and 8. Here, the world location of the main camera component and measure how close it is to the actor itself. Camera boom that could have been seen in the figure 6 is a smart component that adapts the boom length depending on the objects in the world that get between the player the camera itself.

Next event that needs implementation is the long named Change To 1st Person Camera On Distance Check event. Distance is calculated to check if it is lower then set threshold. If that is the case, then the active camera can be switched by using the Set Active node with the correct camera as the target. For manual switching, some kind of easy access variable would be nice to have and for that reason variable Active Main Camera is created and has true as the initial value but here it changes to false.

Now, the same event or function could be used to change the camera back but the

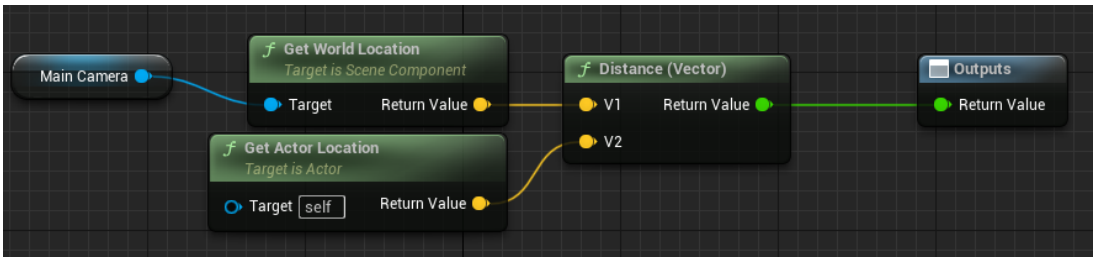

Figure 9: Macro Get Camera to Player Distance

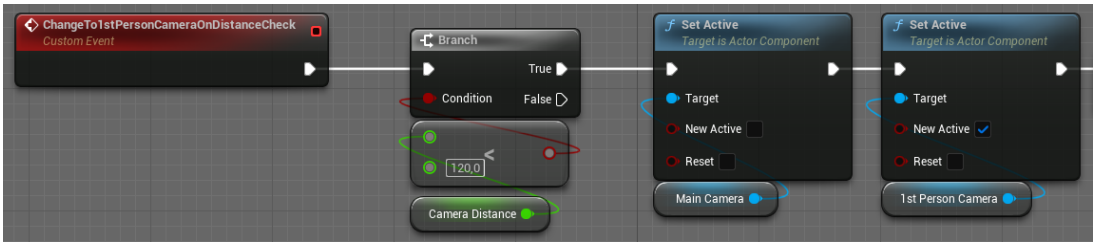

Figure 10: Custom Event Change To 1st Person Camera On Distance Check 1/2

more delicate approach is to have a different threshold for changing the camera back to the third person camera. That is because the behavior of having a specific distance where slight forward or backward movement would change the camera rapidly is unwanted. Because of unwanted rapid camera change another event is added to handle the process of switching back to the main camera. In addition to calling a new event with Set Timer by Function Name there is no need for this event itself to run anymore since this task is already completed. Because of that it needs to be paused with node Pause Timer by Function Name. This event can be seen in figures 10 and 11.

Mentioned event for switching the camera back to third person camera looks exactly the same with inverted cameras that are set as active. The variable Active Main Camera is also set to true and the event pauses itself and unpause the previously paused event. It is relevant to mention that the Set Timer by Function Name firstly checks for the existence of the timer on set function or event name and then, if it exists, it unpauses the timer rather than creating a new one. There is no need to look at this event in high detail since it is very similar to the previous one so the full event is provided in figure 12.

For manual camera switching there should be a reserved button press. For that an Input Action Event that as the name suggest triggers on the input action of the player is created. The button for camera switching is Tab key. Before doing any changes to the camera, a claim that

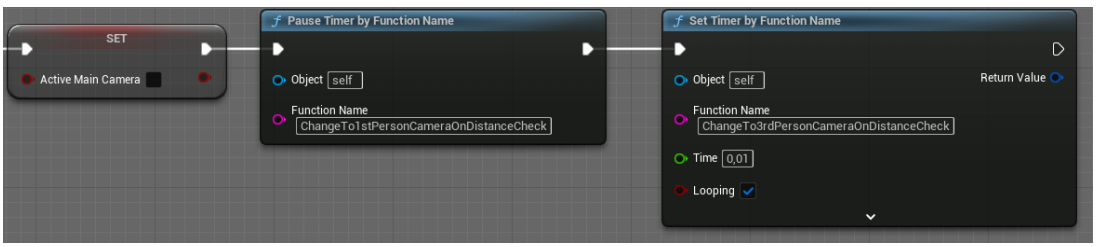

Figure 11: Custom Event Change To 1st Person Camera On Distance Check 2/2

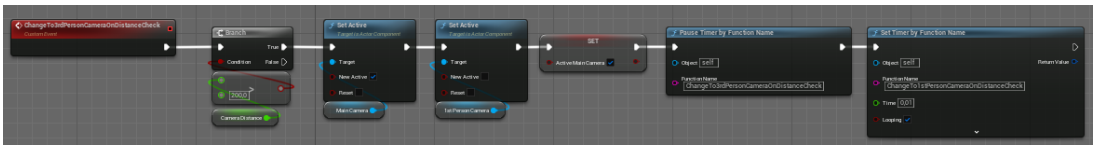

Figure 12: Custom Event Change To 3rd Person Camera On Distance Check

Figure 13: Input Action Event Tab

the player is not under already placed minimal threshold for being in third person camera has to be confirmed. To keep the constraints of the automatic switching system before proceeding, the first thing that has to be checked is if the player character is already below a set threshold. After that an active camera needs to be found and that can be done with the use of the variable named Active Main Camera that is set in last two described events. After that, only actions for pausing or unpausing the Change To 1st Person Camera On Distance Check timer have to be taken since this is the timer that changes the camera back if certain criteria is met. Finally, the value of Active Main Camera should also be changed accordingly. All this can be seen in figure 13 and this concludes all the camera nodes.

5.3. Lock On Target System

This functionality is something that does not need an automatic system. The main and only decision of whether to use it or not is on the player. In the moments where the player wants more control of the camera and make his inputs relative to the looked at rotation, player can decide not to use it. Same thing goes for if player wants his movements to be relative to the AI he is fighting where sidestepping results in circular movement around the AI. Then player can use this lock on system.

It all starts with the Left Control as the Input Action Event that starts or ends this process. Firstly, nodes that stop the lock on system are described since these nodes are a lot easier to understand. The system stops with the lock on feature if there already is a focused target which is saved in the variable called target. If said variable is not NULL it is set to NULL. After that the main loop for locking the camera to the AI is paused and the lock on indicator is set to hidden and dereferenced. One last thing that has to be taken into consideration is that the mouse control over camera is never disconnected and a fight between the new lock on system and the existing input control for the rotation of the camera component makes a visual mess. For that reason there is a Mouse Locked variable that is used to control the execution flow in the

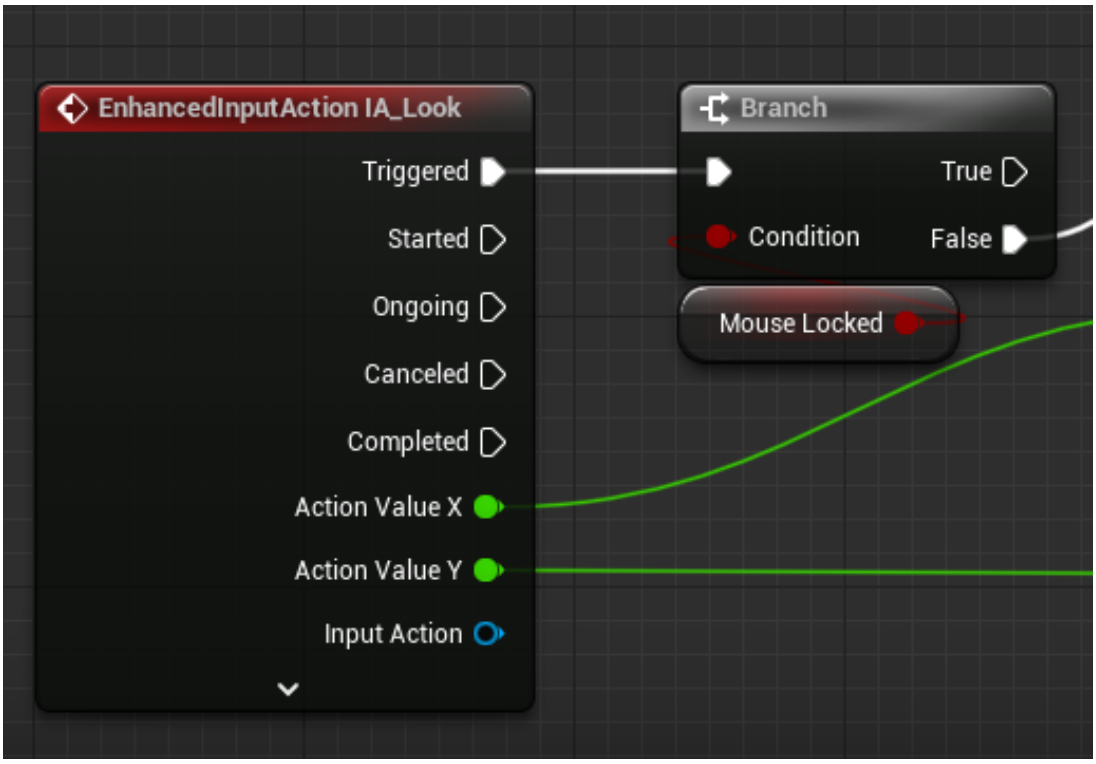

Figure 14: Mouse input action control node

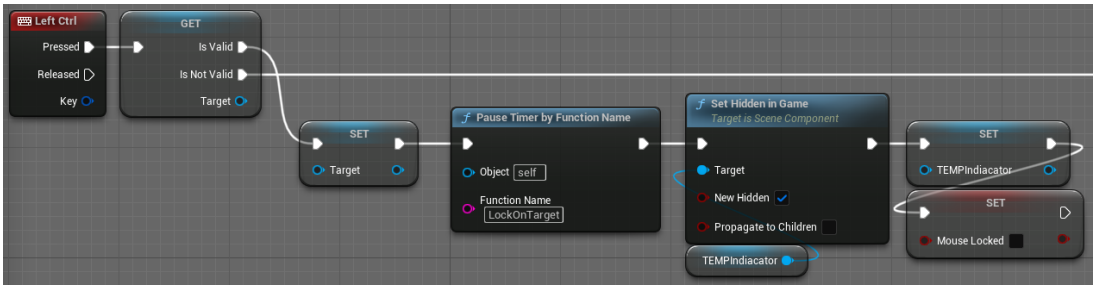

Figure 15: Input Action Event Left Control 1/3

predefined mouse input control. Figure 14 shows added control to the mouse input action and the figure 15 shows nodes that are need to stop lock on behavior.

The next step is to find if there is a valid pawn to set as a target. For that the node Sphere Trace for Objects is used. For this to work some inputs for the start, end, radius and object types that the trace looks for are needed. The starting position is trivial and to get the player location a node named Get Actor Location is used. The end position is a little trickier but not that difficult either.

Firstly, the current camera rotation must be known and that is achieved with the node Get Base Aim Rotation. To make the lock on system a little easier for the player since the AI does not have any flying capabilities the up-down rotation is removed from the equation by breaking the structure pin of the Get Base Aim Rotation and using Make Rotator node to which the Roll and the Yaw rotations are connected. From this rotator a forward vector can be created. The result of that is multiplied by how many units the trace takes place. One unit in

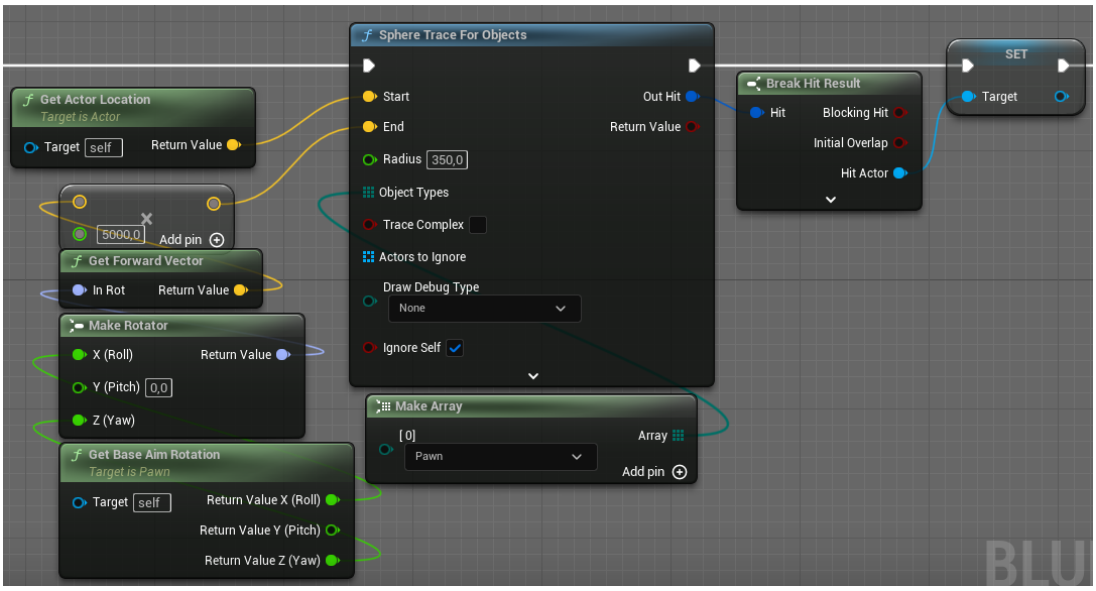

Figure 16: Input Action Event Left Control 2/3

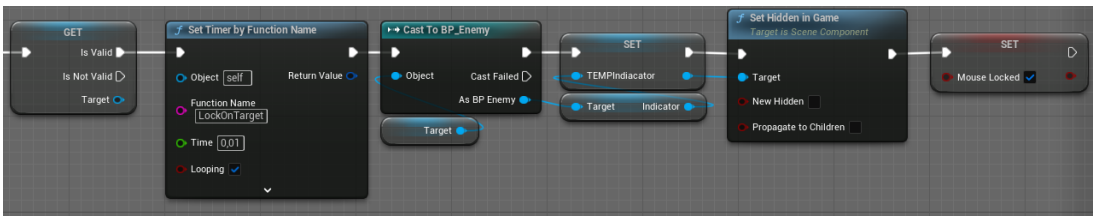

Figure 17: Input Action Event Left Control 3/3

Unreal Engine is representative of one centimeter in the real world. You do not have to abide by those conventions but it is usually easier to do so then not.

Radius is set to a custom value that seems to work the best and for the object types an array must be provided. There is a node called Make Array that has a drop down menu where the object types to look for can be set and, in this case, it only consists of one item. That item is Pawn Object type since AI is of that object type as discussed and explained earlier. If any pawn actor is found, that actor is referenced in variable Target which can bee seen in figure 16.

Now the value to which the trace result is set can be NULL if nothing was found in the sphere trace. For that reason the validity of the variable must be checked before any further steps. If the target is valid the function for camera lock on control can be started. Mentioned indicator is set to active and the mouse intervention with camera control is then locked.

Indicator is a component of the AI visual representation. It does not yet exist in the project but in order to keep everything organized specific actors and unreal objects are described one by one. All the jumping around from class to class and object to object can get confusing pretty fast. Described nodes can be seen in figure 17.

The main dish of this section is certainly the lock on loop itself. This camera control is not hard to implement and it only consists of two mayor parts. One is creating a camera offset towards which the camera should strive towards and the other one is actually commanding the

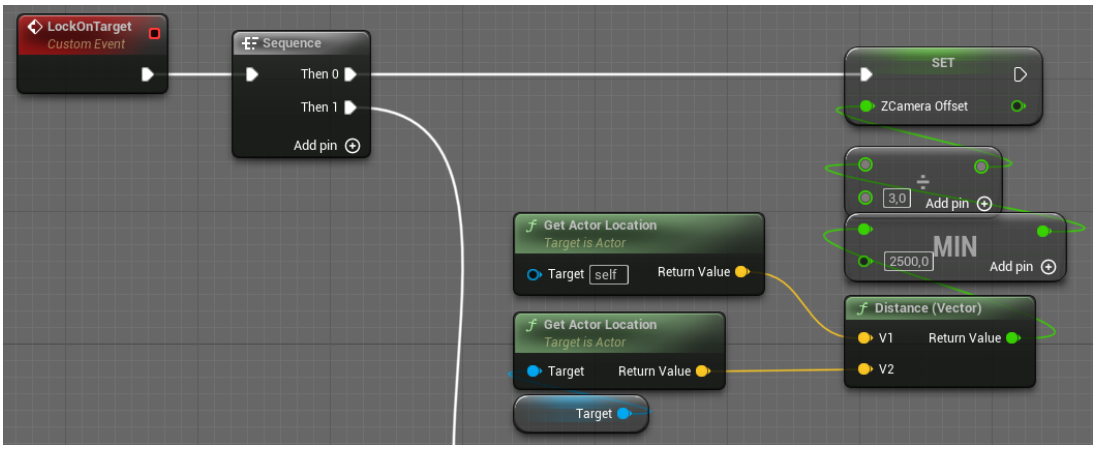

Figure 18: Custom Event Lock On Target 1/2

camera to do just that. Calculated value is stored into Z Camera Offset value. To get this value node Distance is needed again. It is used to calculate the distance between the player and the previously set target. This is the value that goes through some adjustments before being set as the camera offset. Making sure that distance stops being relevant beyond a certain points is achieved with node MIN that chooses the set default number if distance gets too large. After that the value is divided by a certain number to get a smaller or closer number to focus the camera on. The minimum value could be set to lower number to achieve this but with this approach distance is relevant on greater distance value but changes the outcome more smoothly. This can be seen in figure 18.

For actually setting the control location to the appropriate value a node named Set Control Rotation is needed. This function, however, cannot be called with the current blueprint as the caller target. Because of that node Get Controller is needed. The rotation value itself is calculated with the help of the previously extracted value.

Since the desired camera behavior is to smooth-in to the desired rotation rather then to follow it strictly the Rotator Interpolated To node is used. This node, as the name suggests, interpolates towards the target value from the current one taking into account the delta time or the time since the last run of this node, which can be set to global delta time, and the interpolation speed. Current rotation is the current camera rotation which can be accessed via Get Base Aim Rotation. Target location is calculated with adding the Z Camera Offset variable to the actor location and then finding the looked at rotation from that point and the location of the target. This effect can be achieved with different approaches and this is just one of them. Last described nodes can be seen in figure 19.

5.4. Attack System

Camera is all set up and now it is time for the offensive system to take shape. Added collision sphere for attack detection simplifies this process which can get really complicated with every attention to detail. The first check that is needed here is to make sure that player character is not in the process of dashing which is implemented later and that he is not falling

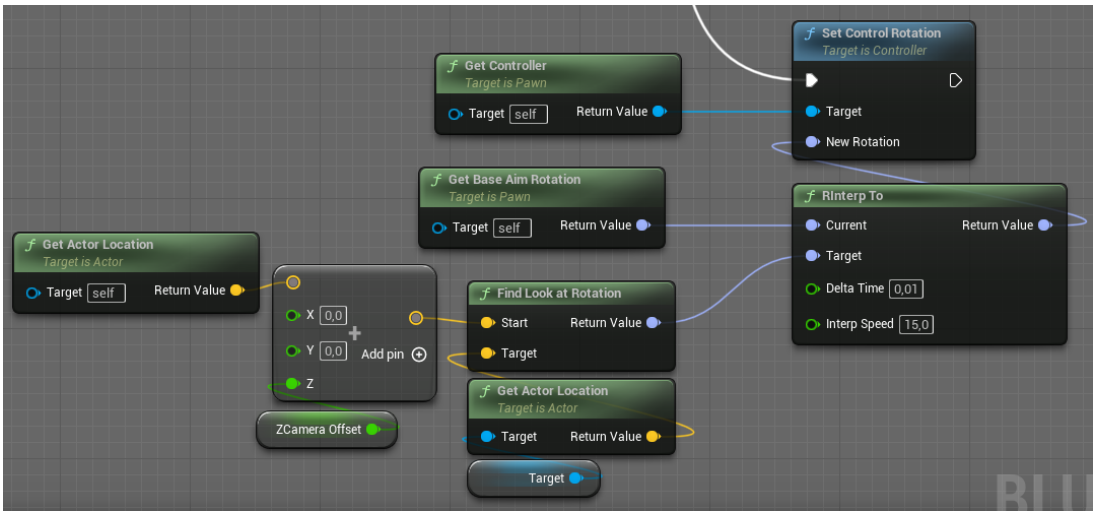

Figure 19: Custom Event Lock On Target 2/2

or in any of the stages of the jump. After that, another boolean variable is taken and used as one of the bases for simple input queue system.

Input queue systems have become the standard in modern gaming industry. Explanation of the system is actually pretty easy. If the player character is unable to perform the action at the moment of input request, save that request and let it through when conditions for performing the action are met within the certain time period.

Variable Attack In Progress, Next Attack and Current Attack Combo are the driving forces behind this cue and also providing the animation blueprint with relevant information in regards to which animations to play. If the attack in not in progress, both Attack In Progress and Next Attack are set to true and the event dispatcher named EDP Attack In Progress is called. Event dispatchers are a messaging system in Unreal Engine. When node Call of the event dispatcher appears in the execution line, it sends a message to all the actors that bind that event dispatcher to themselves.

If the Attack In Progress variable is already set to true, which means the current attack is already playing its animation, a check to find out the value of attack combo scale must take place. Later on, animation blueprint is set up in a way that it calls appropriate events on animation completion. If the animation of the attack is the first in line and is already playing after additional input action the Next Attack boolean is set to true so that after the current attack is finished the next one is already queued to start right after. When the value of Current Attack Combo becomes greater then the Max Attack Combo the value of Current Attack Combo becomes equal to one. All that can be seen in figure 20.

Accompanying this system there are three additional custom events that are called from the animation blueprint. One sets the Attacking In Progress variable to false so that the new attack can be properly set from the input action and it also messages the actors that bind the event dispatcher about such change. Similarly, there has to be an appropriate event for changing the value of the Next Attack variable. Here the event set the value to false and increments the Current Attack Combo variable. Lastly, there is an event for resetting the attack

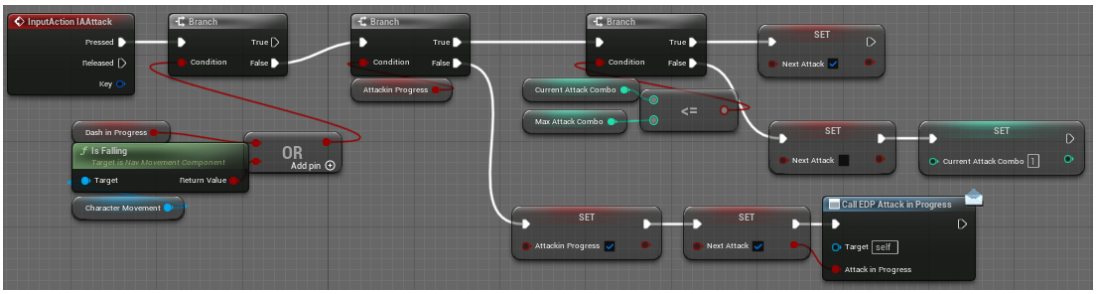

Figure 20: Input Action Event IAAttack

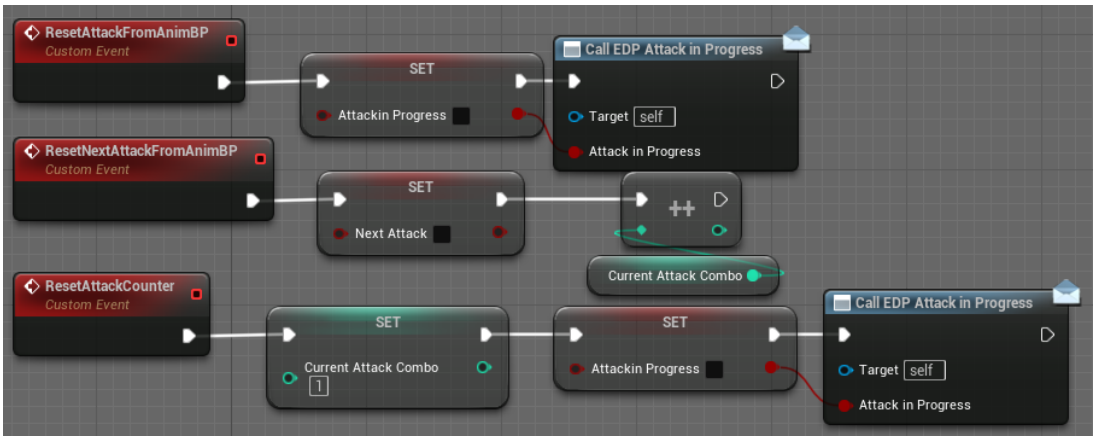

Figure 21: Custom events regarding the attack queue system

counter altogether. This sets the Current Attack Combo back to one and also refresh the Attack In Progress variable back to its false state and called the mentioned event dispatcher which can be seen in figure 21.

In order to make attack easier to hit for the player there also needs to be a custom event that shall create a certain close-gap functionality for the player. All it has to do is to add some forward momentum to the player. This is achieved with a node called Add Impulse which targets the Character Movement component. Impulse value is calculated from actor forward vector and the strength of such impulse which is stored in a variable called Sway Strength. For this process to look smooth the player Velocity also has to be set to the same value and this can be seen in figure 22. There is additional simple event called after that which is explained right away.

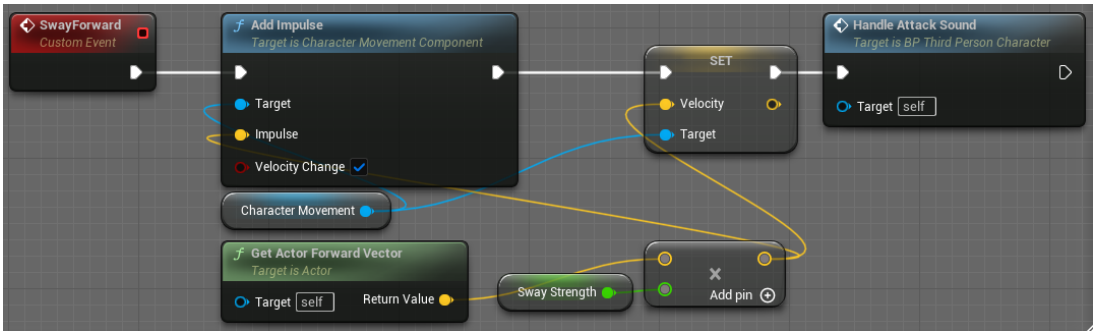

Figure 22: Custom Event Sway Forward

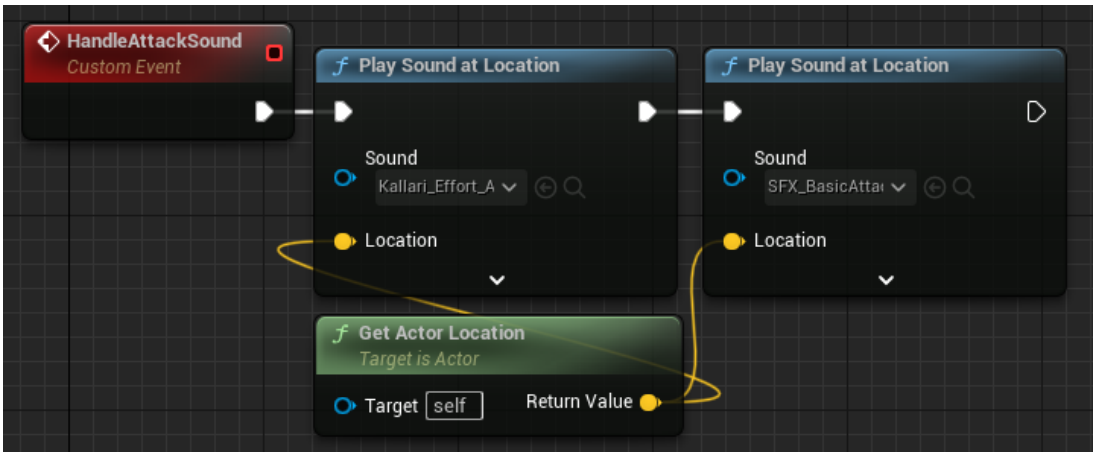

Figure 23: Custom Event Handle Attack Sound

Event Handle Attack Sound is only here to provide player with better immersion experience and has no functional value besides that. It is made out of two of the same nodes called Play Sound at Location. One of the nodes is playing the attack effort of the player character and the other one is playing the sound of the weapon cutting through the air. For the effort sound there is already a premade sound cue which plays a random of the selected sounds that make up the cue.

Cues in Unreal Engine are different from the sound itself because they can have additional space sound settings as well as simple random function which is the case with this particular cue. Unreal Engine uses '.wav' extension for the sounds it uses. The other sound is from the asset that is also imported to this project. Described event uses player location for playing both sounds and can be seen in figure 23.

Dealing damage is another part of the attacking process that needs to be accounted for. For this part, a communication with animation blueprint also needs to be in full swing. Firstly, the Damage Sphere component is used. This component was created in the first section of this chapter. Here a special events called On Begin Overlap (Damage Sphere) and On Component End Overlap (Damage Sphere) are used. These are the events that all overlapping components have in common and is precisely why it was added to the actor. In the begging of any kind of overlap there has to be a certainty that the overlapping actor is the AI itself. That can be achieved by giving that actor a certain tag. That tag for AI is Enemy. So if the target has that tag it is set as potential damage receiver in the variable Current Damage Target. Same thing goes for the end of overlap where again the player character has to sure that the overlap that ended is the one with the AI itself so that the Current Damage Target can be dereferenced as shown in figure **??**.

Two of the following three events are called from the animation blueprint. When there is a certain period inside the animation where the player has the ability to damage AI, that is in accordance with the animation itself, this events are called. One for activating the damage window and one for deactivating it. Event Set Is Damaging starts the loop event named Damage Check. On the first run of this event in which the Current Damage Target is valid the loop is interrupted and damage are applied. If, however, that does not happen until this actor receives

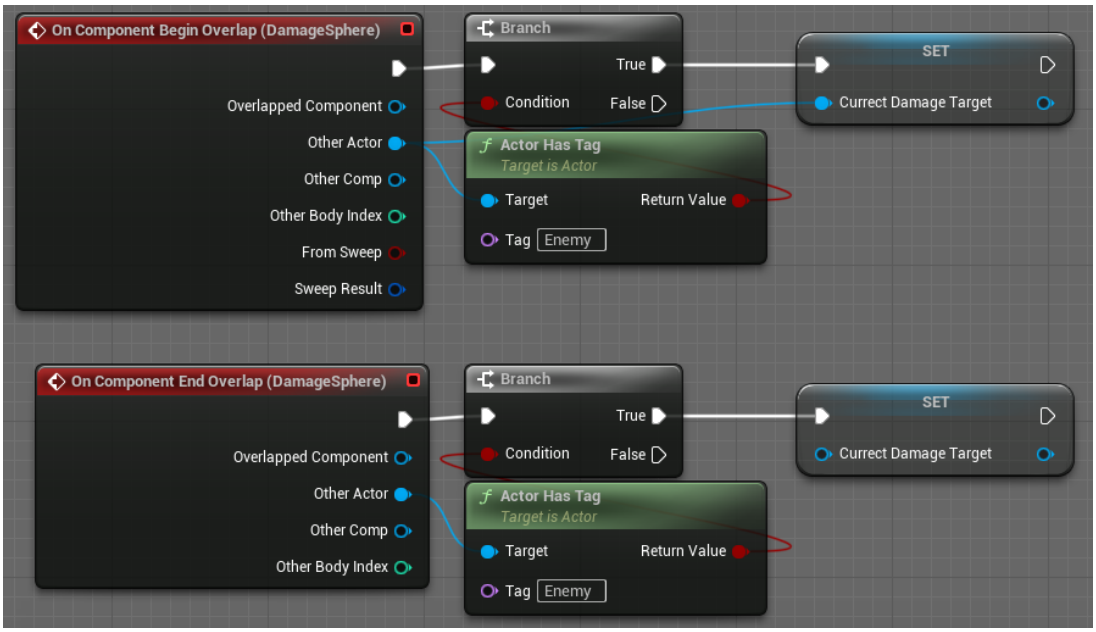

Figure 24: Events On Component Begin and End Overlap of the Damage Sphere Component

the the call for Set Is Damaging FALSE, the validity of the Current Damage Target is disregarded since the damage can only be applied in specified damage window. Explained nodes can be seen in figure 25.

Damage messaging system is present in Unreal Engine and makes this next step of applying damage really simple. All that needs to be done is to call the Apply Damage node with the Current Damage Target as the Damaged Actor argument. Base damage adopts a random value that can be both higher and lower then the amount that is stored in the Damage Dealt variable. The final amount is the sum of the Damage Dealt variable and the random number in the specified range as provided in figure 26. This is the last element of the attack system so the next section is about defensive systems and receiving damage.

5.5. Defense System and Taking Damage

In the defense department there is no block or similar functionality, rather the player is able to use dash in either left, right or backward direction. That information is derived from the current movement of the player character. For that reason there need to be some indicators on which buttons player is pressing in the event on requesting a dash ability. For that reason there are three boolean variables that indicate which of the left, backward and right buttons allocated to ’A’, ’S’ and ’D’ are pressed. This can be seen in figure 27.

For better understanding the logic in the dash itself, one particular variable is explained in detail. It is important to know how it is set and what it represents. It is used to communicate to the animation blueprint if only the forward movement key of the movement keys is pressed. Input Action Forward is defined only with the ’W’ key. Input Action Not Forward is defined with ’A’, ’S’ and ’D’ keys. Both of these events have the same goal and that is to provide information if the ’W’ key is the only key pressed in the movement keys set. You can notice that only if the

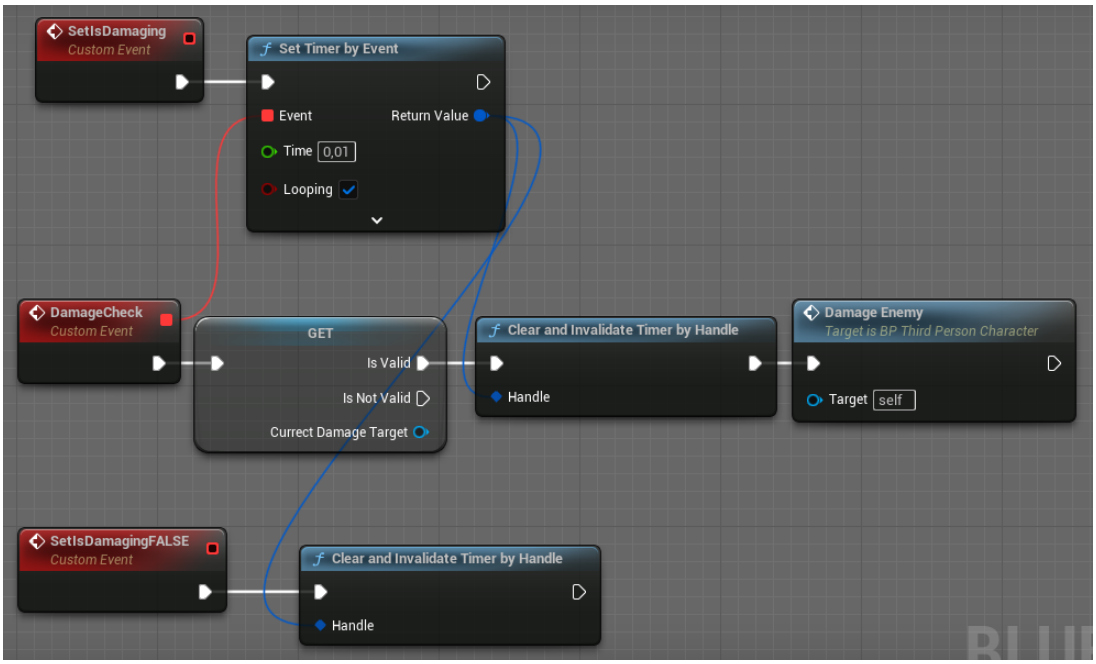

Figure 25: Custom events for Opening and closing damage window

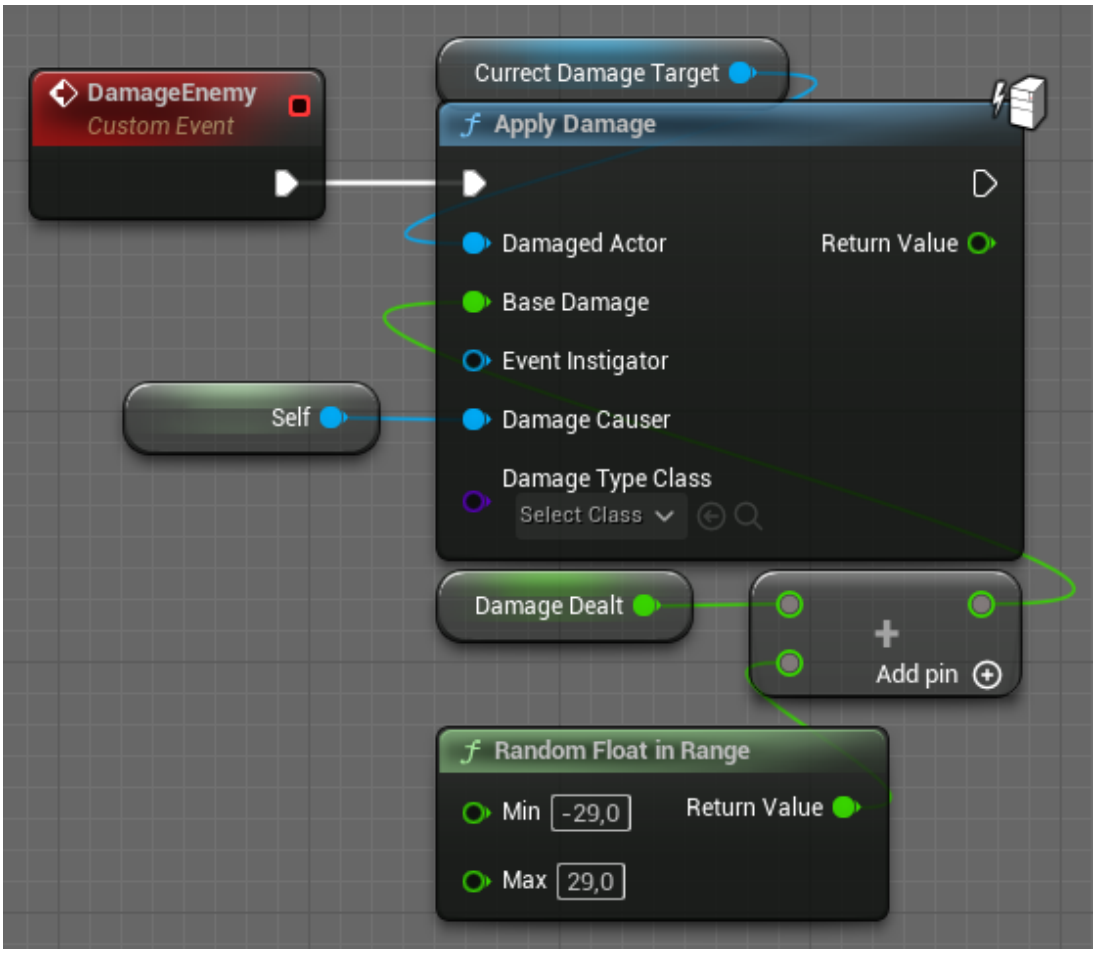

Figure 26: Custom Event Damage Enemy

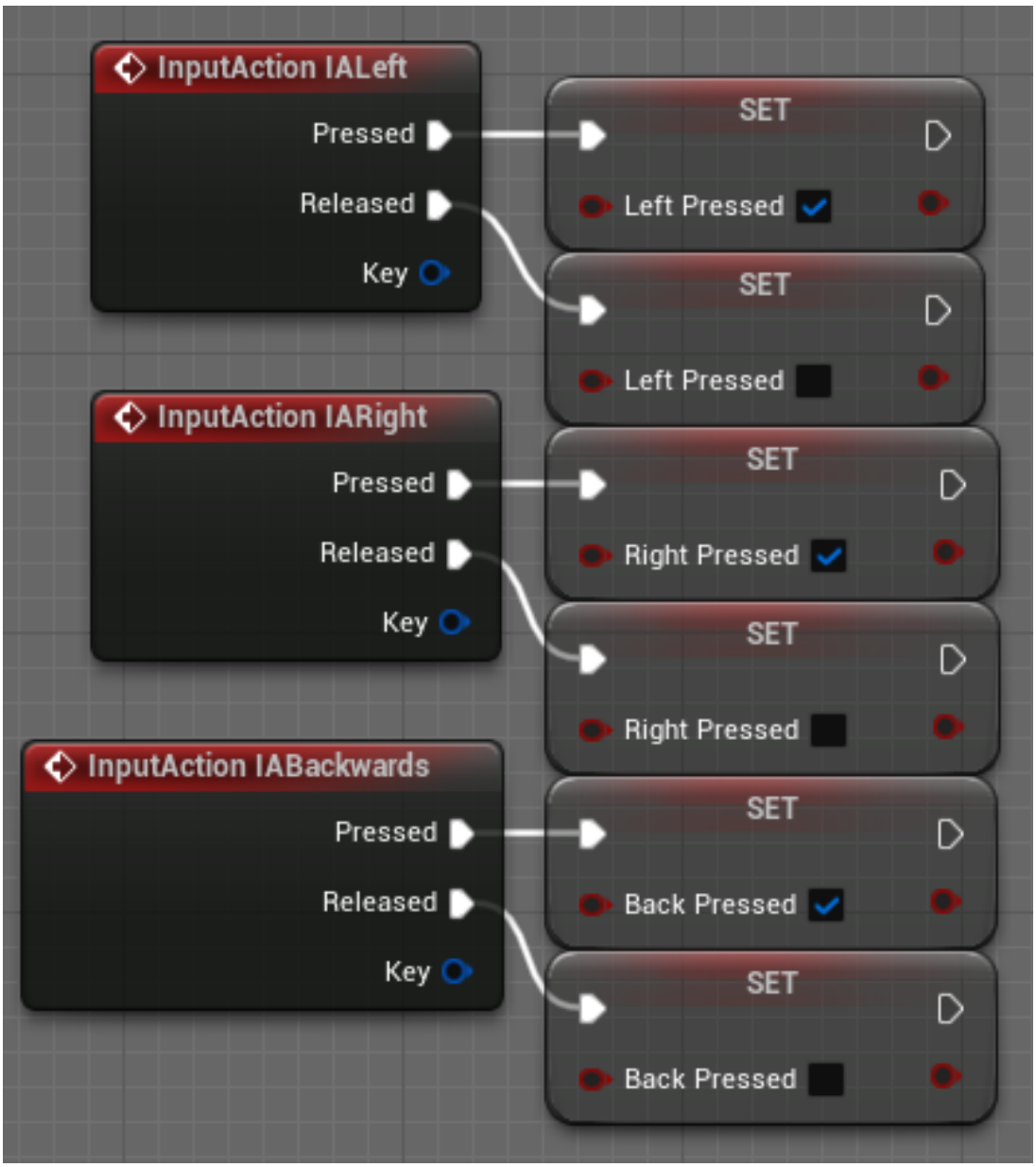

Figure 27: Input action events for boolean keyboard press values

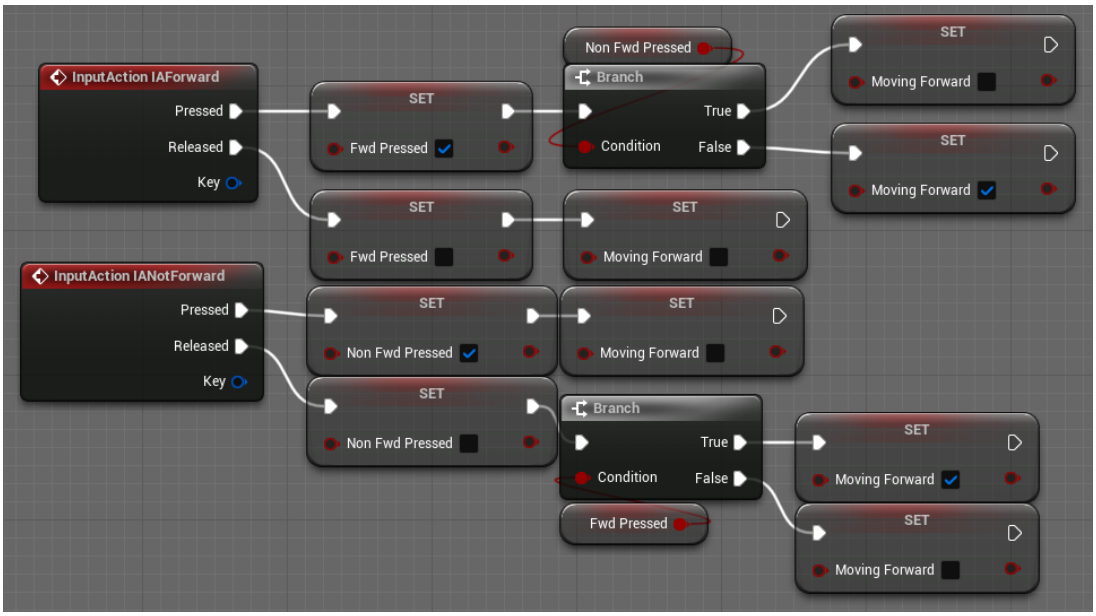

Figure 28: Input action events for defining only forward movement

forward key is pressed and no other key is pressed at that moment or if when the non forward key was released the only key left is the forward key, Moving Forward value is set to true. To be fair, if this system were to be one hundred percent accurate, a counter variable for counting keys that are pressed at the same time is needed, but this works well enough as it is and can be observed in figure 28.

Now that this explanation is out of the way, dash itself is explained in full detail. Input Action for calling dash can be started with both Shift key and the Right Mouse Button. Here a simple monitor is implemented which forbids the action of dashing while dash is already in progress. Other than that, there has to be a limit on the number of times that the player can dash in air. This limit is set to one dash only. This is done with the help of the animation blueprint which provides information about when the player character has landed on solid ground.

Besides that there is a necessity for differentiation between if the player has only 'W' key pressed in the event of the requested dash. If that is the case, since there is no forward dash, input is ignored and dash action is possible. If the player was never in the air in the first place, check for dash only proceeds without any changes to the Can Dash in Air variable. Here there is a timer for putting the value of Can Dash In Air back to true if something goes wrong in the animation blueprint but this can not be seen in figure 29 and continues the execution in figure 30.

Basic Branches that determine the dash direction can be seen in figure 30. If it is backwards, set the Dash Backwards to true; if it is left, set the the Dash Left to true; if it is right, set the the Dash Right to true and lastly, if it is none of them, unlock the monitor. There is additional Set node here which sets the camera lag speed to five. The lower the number is the higher the lag amount. The default lag amount is set t 10 since it looks more interesting that way then with no lag at all. Lag speed is set lower here so that the dash impact on movement is further accentuated except when player character is dashing backwards where the camera

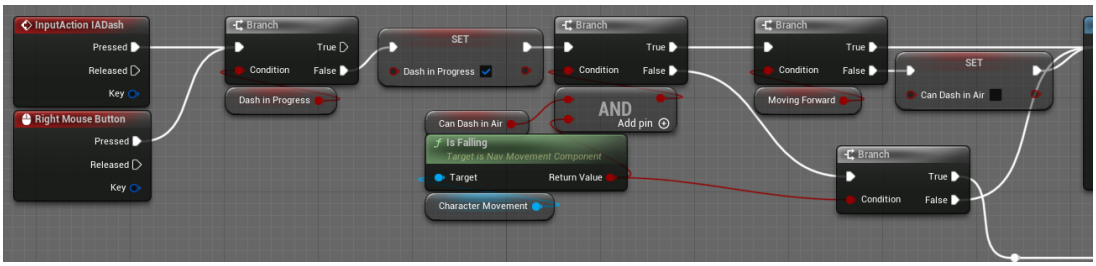

Figure 29: Input actions for dash functionality 1/4

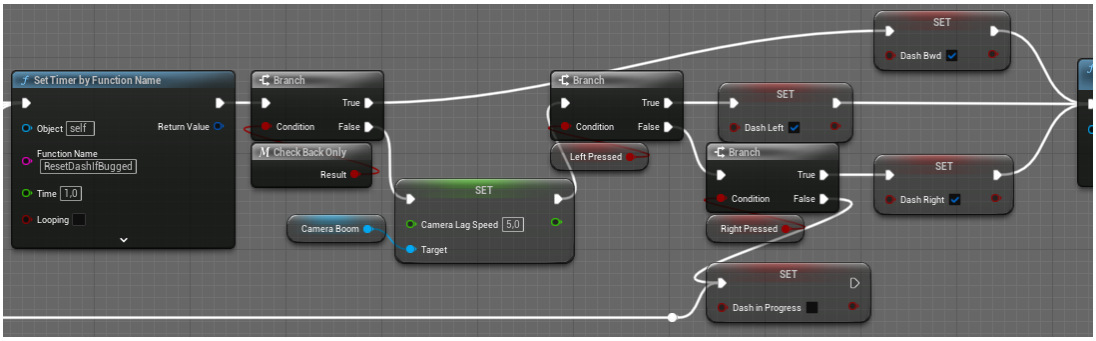

Figure 30: Input actions for dash functionality 2/4

needs to still focus on the player as usual. Described flow can be found in figure **??**.

Macro in is a simple one which checks if the 'S' key is the only pressed key at the moment of requesting a dash. That is the case if neither left nor right keys are pressed. Described macro can be seen in figure 31.

Next on the line of execution is the function that calculates shortened dash distance. This is needed because behavior of slamming into the walls or obstacles is unwanted. In a situation where there is not enough space for the full length, the dash itself is shortened. Dash itself is implemented with the use of Timeline. This function in Unreal Engine can have a graph value inside of it that changes value over defined periods of time. Here it is used in a way that the start of the dash appears more explosive and then it smooths out towards the end of it.

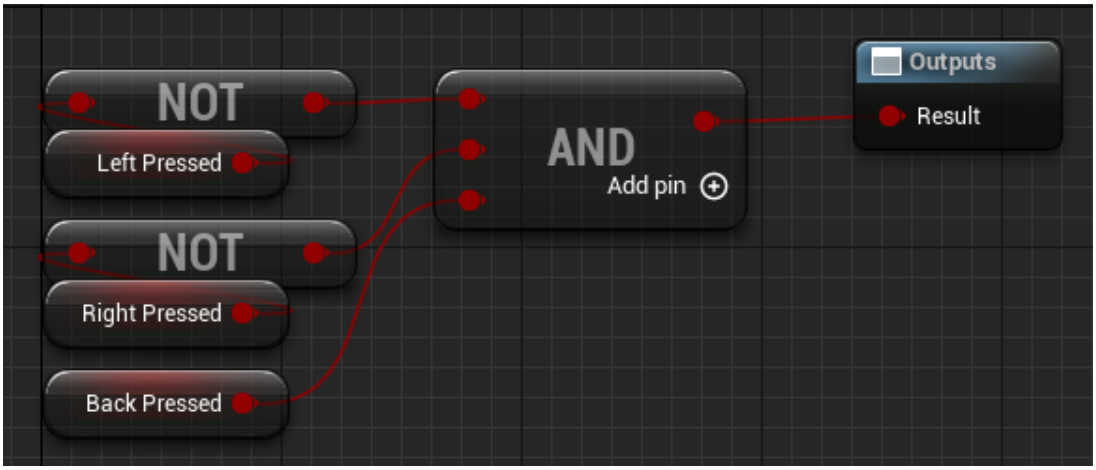

Figure 31: Macro Check Back Only

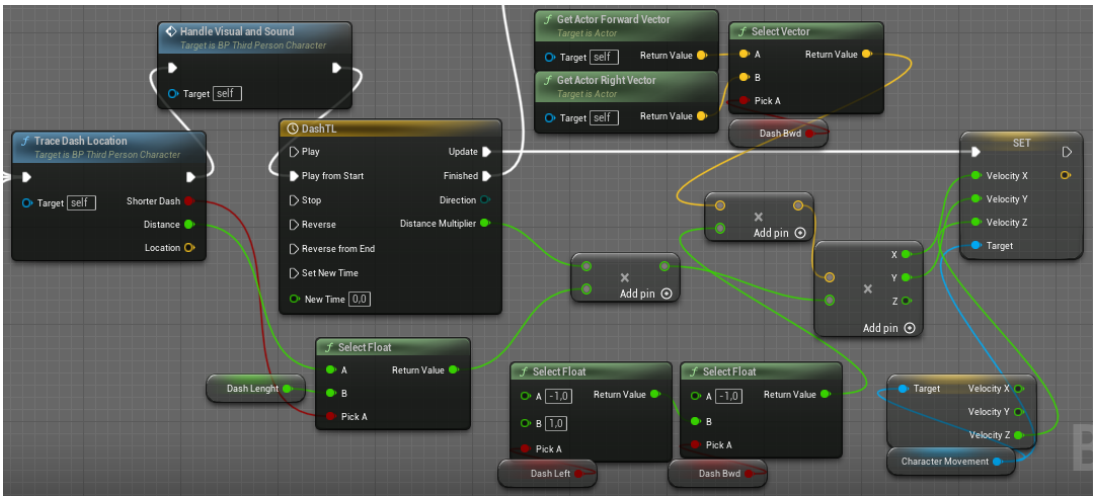

Figure 32: Input actions for dash functionality 3/4

If the function Trace Dash Location, which is examined a bit later on, creates a shorter dash distance variable then that variable is used as an endpoint in this calculation. Dash length is multiplied by the distance multiplier from the timeline and with the appropriate direction vector. If the dash is going left then a negative value of the actor right vector is needed; if the dash is going right then a positive value of the actor right vector is needed; If the dash is going backward then a positive value of the actor right vector is needed. Lastly, the momentum of the up and down motion has to be preserved and that is why Velocity Z from the Character movement component is needed. Described nodes can be seen in figure 32.

How does a function Trace Dash Location calculate new dash target distance and location? For the lock on system the Sphere Trace For Objects node was used. Here a simpler yet similar Line Trace By Channel node is used. Since trace target is nothing in particular there is no need to provide this node with array of object types.

Start of the line trace is always the same and it is the location of the player actor. End location depends on the direction of the dash. If the dash is to the right the the actor location vector is added to the multiplied actor right vector with Dash Length variable. Dash to the left negates that and so on. This has already been discussed so more detail is not provided. Selection process can be observed in figure 33.

The rest of the function is only picking the right variable to assign to the output values. If the hit was calculated then the target location is the Impact Point, otherwise the Trace End is the the Target Location. Calculated distance is the distance between that hit location and the actor itself. Boolean value of the Line Trace By Channel result is true if hit happened. This second part of the function can been seen in figure 34.

There is function named Handle Visual and Sound that could have been seen in figure 32 but since it is only there to convey visual information to the player it is described shortly after the last part of this dash main event. The only thing that is left is to reset all the dash values back to their original false ones and set the camera speed back to its original value which was ten. That is shown in figure 35.

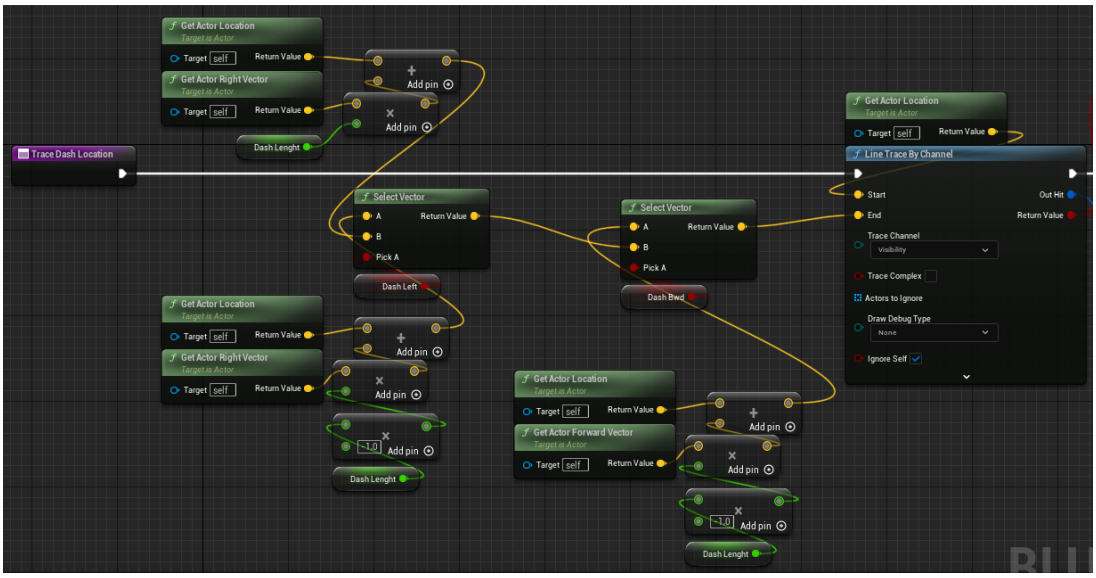

Figure 33: Function Trace Dash Location 1/2

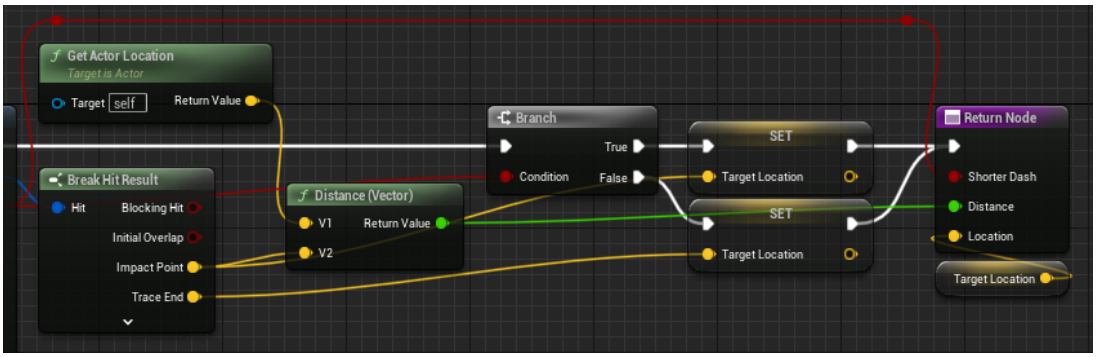

Figure 34: Function Trace Dash Location 2/2

Figure 35: Input actions for dash functionality 4/4

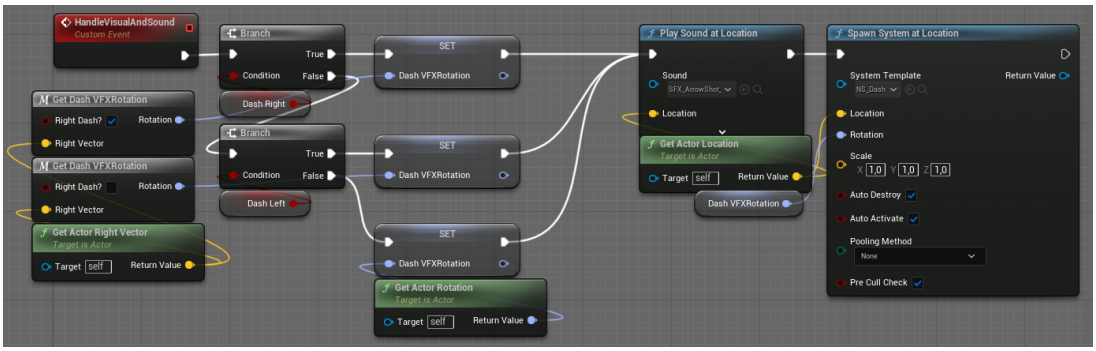

Figure 36: Custom Event Handle Visual And Sound

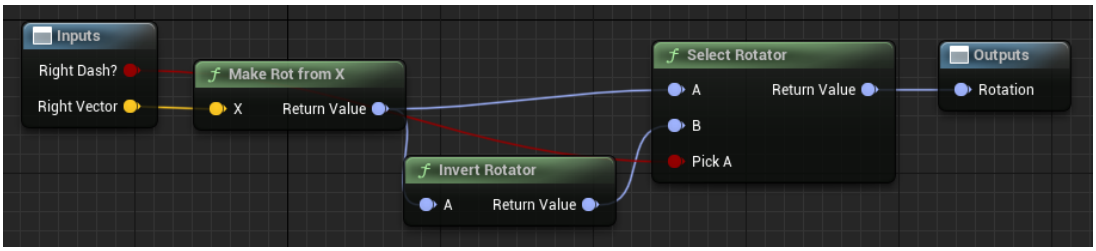

Figure 37: Macro Get Dash Visual Effects Rotation

To make the experience more engaging a sound effect and visual effect are added to dash in the Handle Visual And Sound event. The main thing that also needs to be checked here is which is the direction of the dash and calculate the rotation of the effect accordingly. Now that the Dash Visual Effects Rotation variable is set, nodes Play Sound at Location and Spawn System at Location can be called. Both the sound and the visual effect spawn at the same location which is the player actor location and the node content can be seen in figure 36.

Macro for getting the correct rotator is simple. It requires the forward vector and information if the dash direction is right or not. Firstly, it creates a rotator from the right vector and then inverts it if the dash direction is left. This simple macro can be seen in figure 37.

The main cause for having dash ability on player character is for evasive maneuvers and avoidance of taking damage. However, damage taking is to be expected and that capabilities also need to be implemented. The communication between UI and and the player actor is usually not direct and goes through the so-called controller. For that reason the Player Controller named BP Player Controller is created for the player. This player controller also needs to be set as active in the properties of the player actor. To the Event Begin Play additional execution pins are added which include events for setting up health points (hereafter HP) and getting a reference to the newly created player controller. One more event that is called is the event that has to be created inside the player controller and it is used for requesting update to the UI. Described nodes can be seen in figure 38 while their implementation is available in figure 39.

Implementation of the Set Up Heath Points consists only of one Set node that sets the value of Health Points variable to the value of Health Points Maximum variable. Custom Event Get Player Controller is also really simple and it gets and save the reference to the player controller of the Player Actor by Casting the controller object to the BP Player Controller Class. Figure 39 contains both of these events.

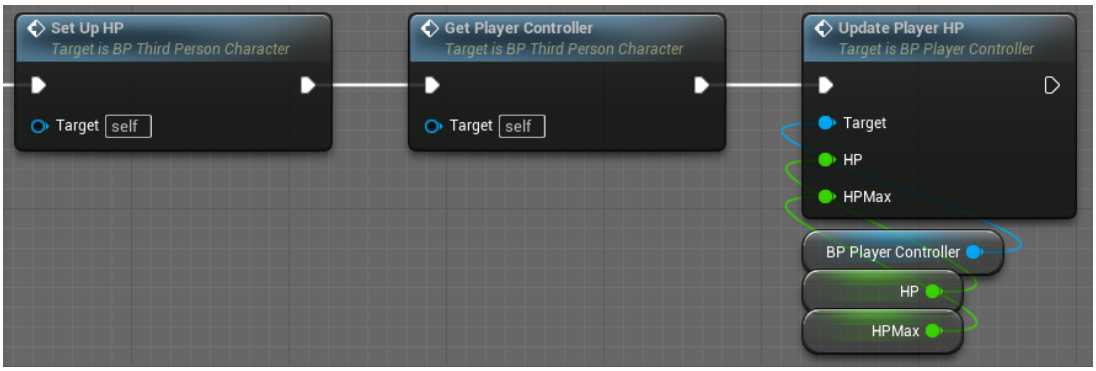

Figure 38: Event Begin Play HP setup

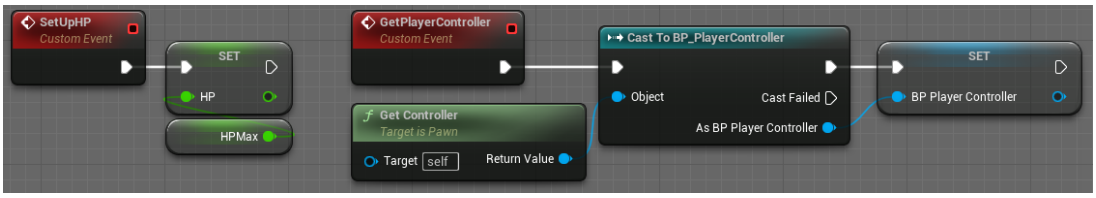

Figure 39: Custom events Set Up HP and Get Player Controller

Besides initial HP setup there needs to be some kind of handler that covers taking damage. There is an event called Event Any Damage which is called every time the actor takes damage via any of the apply damage functions very this actor is referenced as the damaged actor. The first thing that needs to be done is to subtract the damage amount from the current HP value and set that as the new value. After that the UI needs to be updated and appropriate animation and sound have to be played. These nodes can be seen in figure 40.

For hit animation and sound there is a simple set up with two custom events. One of them is called and it sets the value of the Is Hit variable to true and calls a node Set Timer by Event which acts as a delay. After defined time this node calls event Hit Recover which sets the mentioned variable back to false. Sound is also played at the actors location and all this can be seen in figure 41.

The continuation of the Event Any Damage in figure 42 goes to the check where the value of HP lower than zero is required. In that case the player character has lost all the HP and game needs to be finished. For the animation blueprint variable Is Dead is set to true, another event dispatcher is called named EDP Player Death which the AI implements. Then the appropriate end screen is called from the player controller and it ends with a custom event

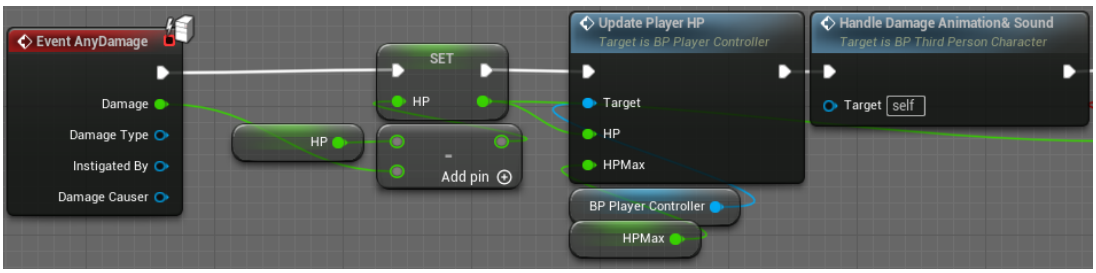

Figure 40: Event Any Damage 1/2

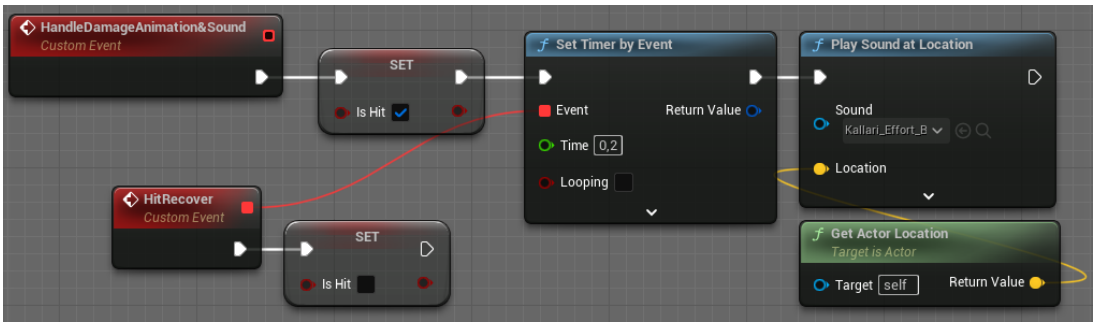

Figure 41: Custom events Handle Damage Animation And Sound and Hit Recover

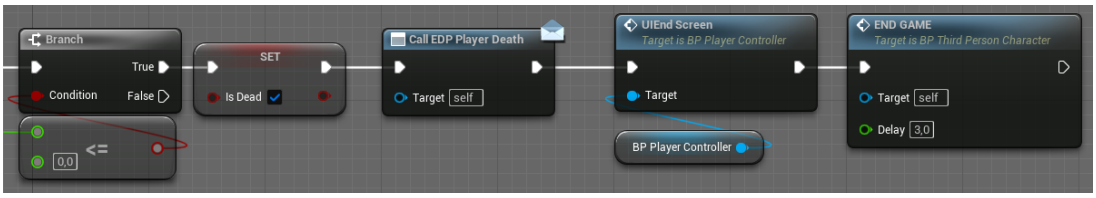

Figure 42: Event Any Damage 2/2

END GAME which is discussed in more detail.

This custom event is really simple. It disables player input and after the specified delay it closes the game application by calling node Quit Game. This set of nodes is in figure 43.

The last part of this defense system in the stun effect implementation. It is not particularly defensive in its nature but it communicates the negative effect of being hit by a specific ability to the player, just like the reduction in HP. For the animation blueprint the variable Is Stunned is set to true after which the movement is disabled. There is a visual effect that is spawned above the player character to indicate the effect as well as the appropriate sound effect. All actors that bind the event dispatcher EDP Player Stunned are also notified. These nodes are in figure 44.

After a delay of two seconds the variable Is Stunned is set back to false and the player can move again since his movement component has been activated. All actors that have bound the event dispatcher called EDP Player Unstunned are notified here. The rest of the Event Stunned nodes can be seen in figure 45.

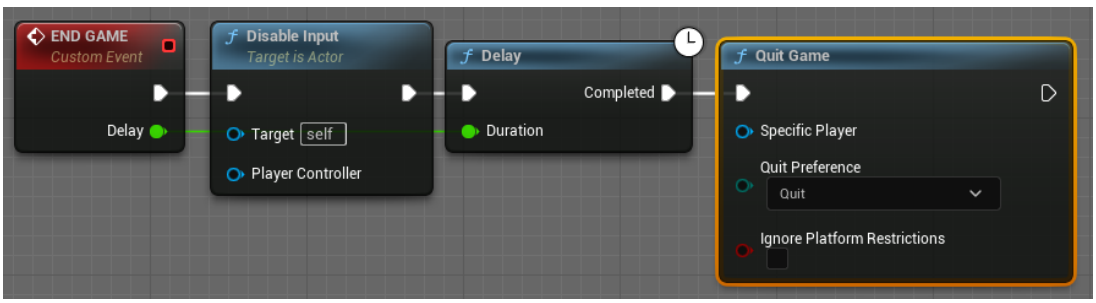

Figure 43: Custom Event END GAME

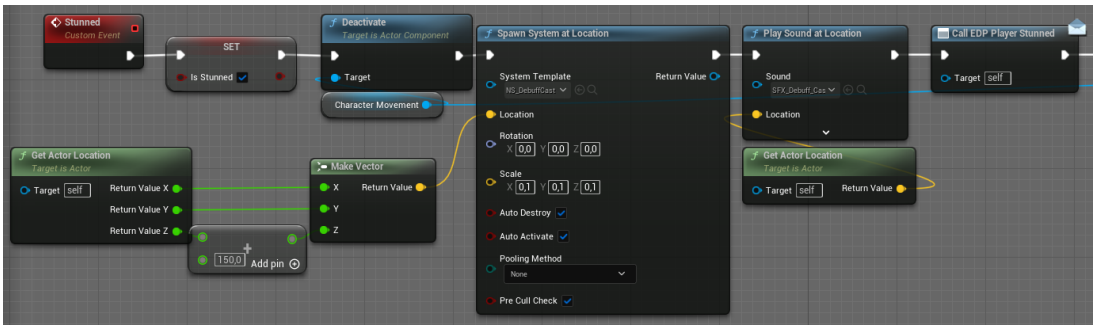

Figure 44: Custom Event Stunned 1/2

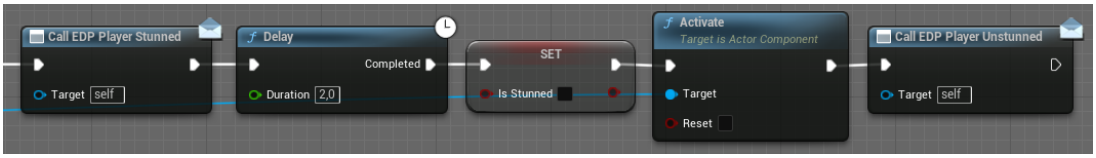

Figure 45: Custom Event Stunned 2/2

5.6. Animation Blueprint Basics

Most of animation handling is accommodated in animation blueprints of both the player character and the AI. An attempt is made here to condense this information and still provide a basic understanding on how everything is wired and how it works. There is some programming in the Animation Blueprint Event Graph but is mostly reading the specified variables from the player character main blueprint. Since the Event Graph was already mentioned, this is the first thing on the list. Most important parts are explained so that the basic idea of how this works and what it does can be formed.

The first thing on the list is the event called Event Blueprint Initialize Animation. This event is like a constructor or an Even Begin Play which was present in the previous actor class. Here, the reference to the player and its Character Movement component is saved and the Function for tracing the ground for slope angle is started. To not go into too much detail, this function traces the floor beneath the player to apply the appropriate animation from the animation Blend Space. Blend spaces are dynamic animation outputs that merge more animations into one instance of an animation and blend between them based on some set values and ranges. Contents of the described event can be seen in figure 46. The event Line Trace For Slope Angle is not provided on that figure.

Equivalent of the Event Tick for the animation blueprint is Event Blueprint Update Ani-

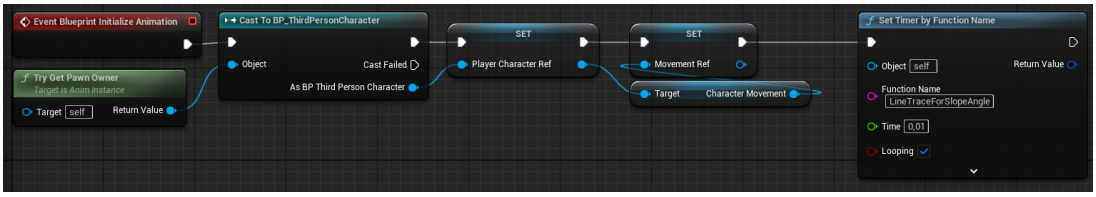

Figure 46: Event Blueprint Initialize Animation

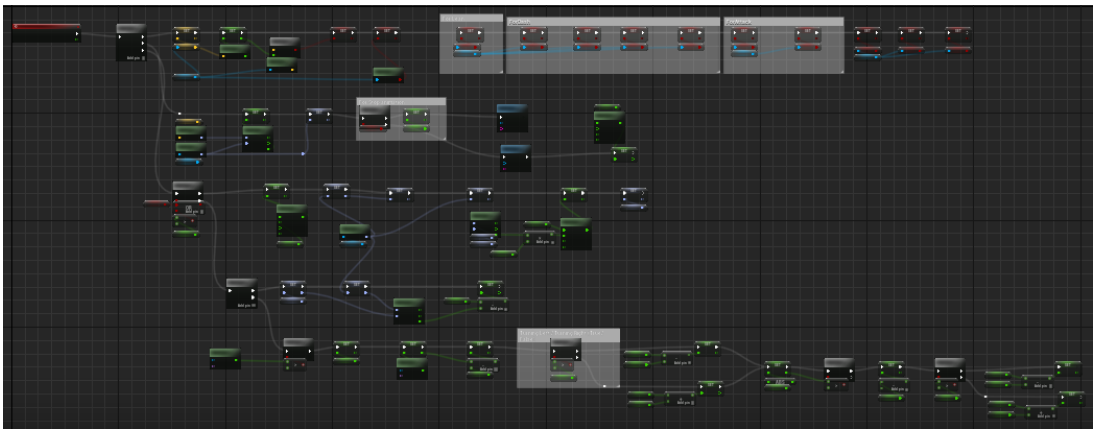

Figure 47: Event Blueprint Update Animation

mation. It is a lengthy event with few main sections of the code. The topmost section is tasked with updating the variables from the player character blueprint. This is why a lot of boolean variables were created in the player blueprint so that this animation blueprint can read those values and set animations accordingly. The second section of a sequence is taking care of the moving animation stop direction and the slope angle stop adaptation. The last and the longest part is tasked with calculating rotation of the root bone of player character mesh. It creates an effect where the actor rotation always follows the camera or mouse rotation but the root bone rotates the mesh in the other direction up to a certain threshold. This then triggers the animation for turning in the Animation Graph which is the second big element in this blueprint. Figure 47 contains all nodes of the nodes that work on explained functionality.

The last part in this Event Graph that needs mentioning is the ground of Animation Notification events which call the appropriate events in the player character blueprint and alter values in there. Animation notifications can be triggered from three main places and those are the animation sequence itself to which animation notifications can be added, animation state machine state and transition. In this project, these kinds of notifications were added for the Sway Forward event and the damage window. When it comes to animation state machine animations can be triggered at the start, during or at the end of the animation state. Same goes for the animation transitions. Once the the animation state is shown everything should be clearer. Group of notify events that call the events from the player character can be seen in figure 48.

The next thing to understand is the Animation Graph. It is shown in figure 49 and it does not look that complicated at first. Firstly there are two state machines that control the animation state. The first one is made of basic locomotion states while the other includes dashes, jump, attacks and everything else. Then there are 3 more nodes before the output pose end the graph. One node is a blend space that picks the appropriate head placement of the player character based on the calculated Yaw Offsset Root and the player aim rotation. The second node rotates the mentioned root bone by calculated offset and the last one is used for inverse kinematics, which are only applied if the player is on the ground. Inverse kinematics in this project move inverse kinematic bones to the appropriate place above the ground. This then moves the real leg and spine bones to the same place with interpolation. This means that the

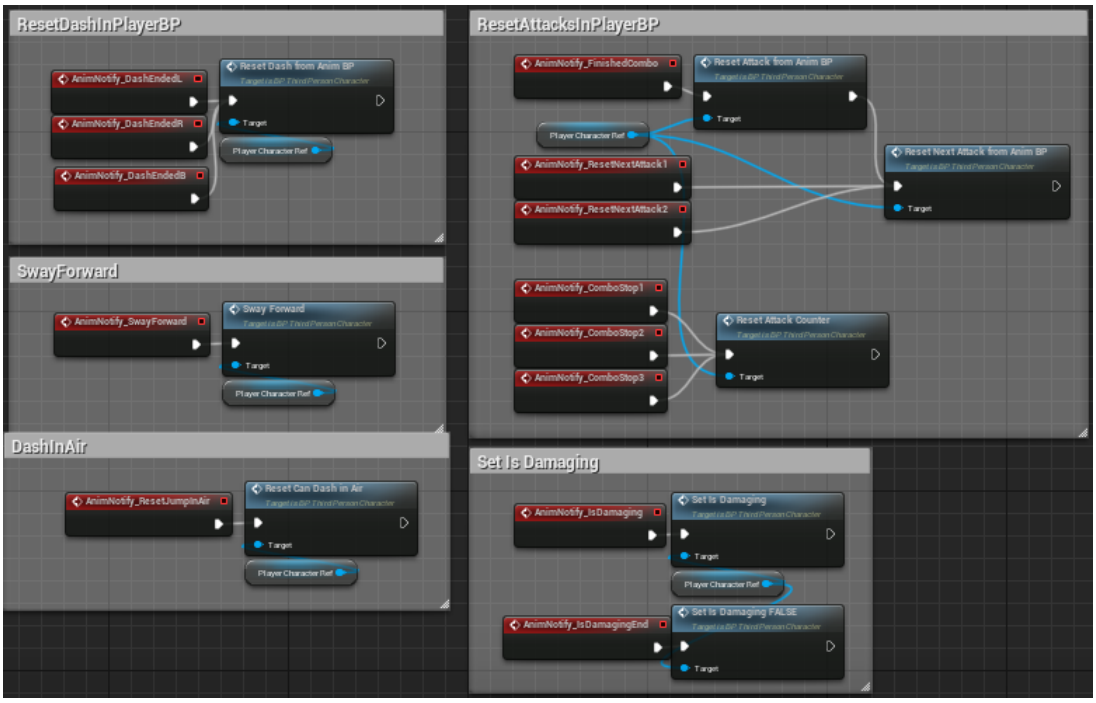

Figure 48: Group of animation notification events

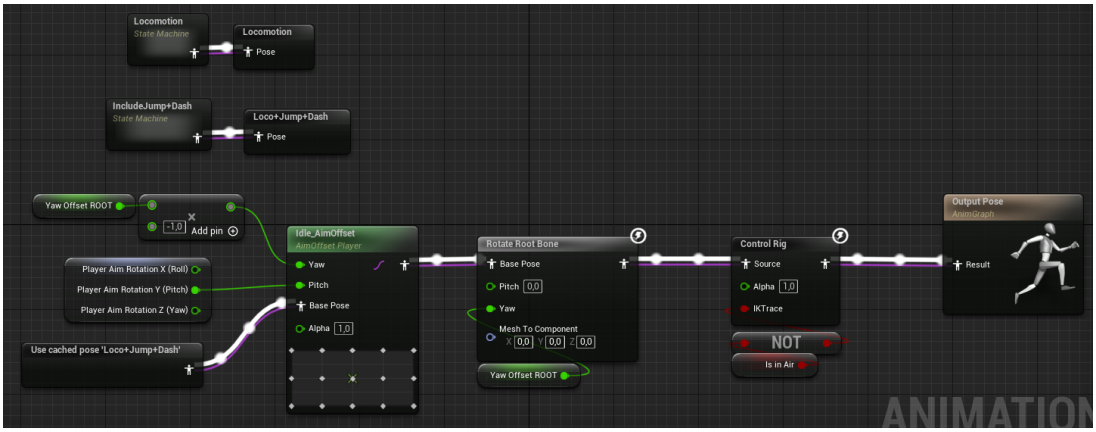

Figure 49: Group of animation notification events

player character mesh always looks like he is standing on the ground, no matter what kind of surface he finds himself in.

Figures 50 and 51 contain the two animation state machines that are present in player character. Each of those states has a defined animation and multiple transition rules as well as some animation notification events where they are needed. There are some notification events inside of animation sequences and all this needs to be set up properly so that in can work in a desired way. More detail than that is not necessary the explanation moves to the Player Controller.

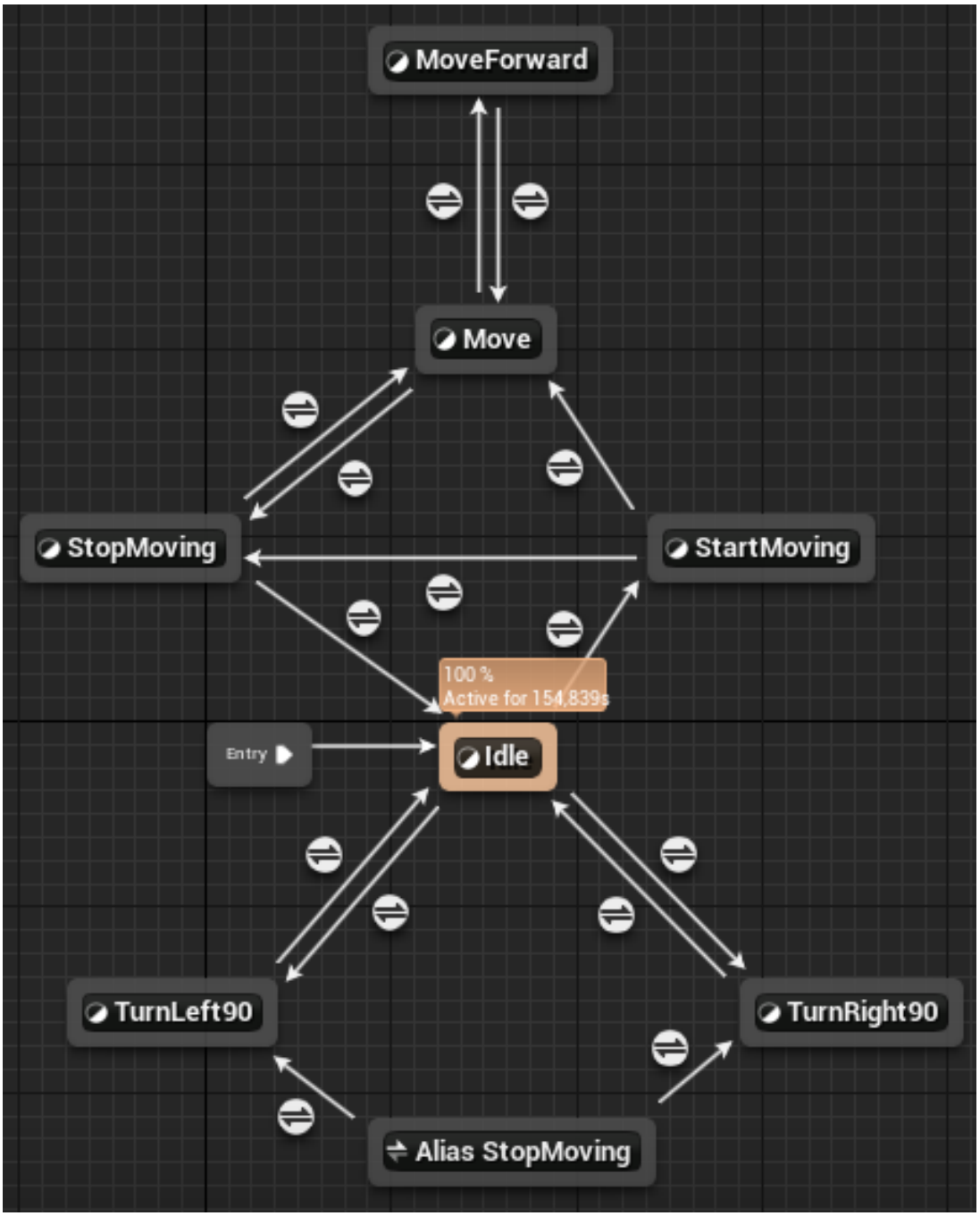

Figure 50: Animation State Machine Locomotion

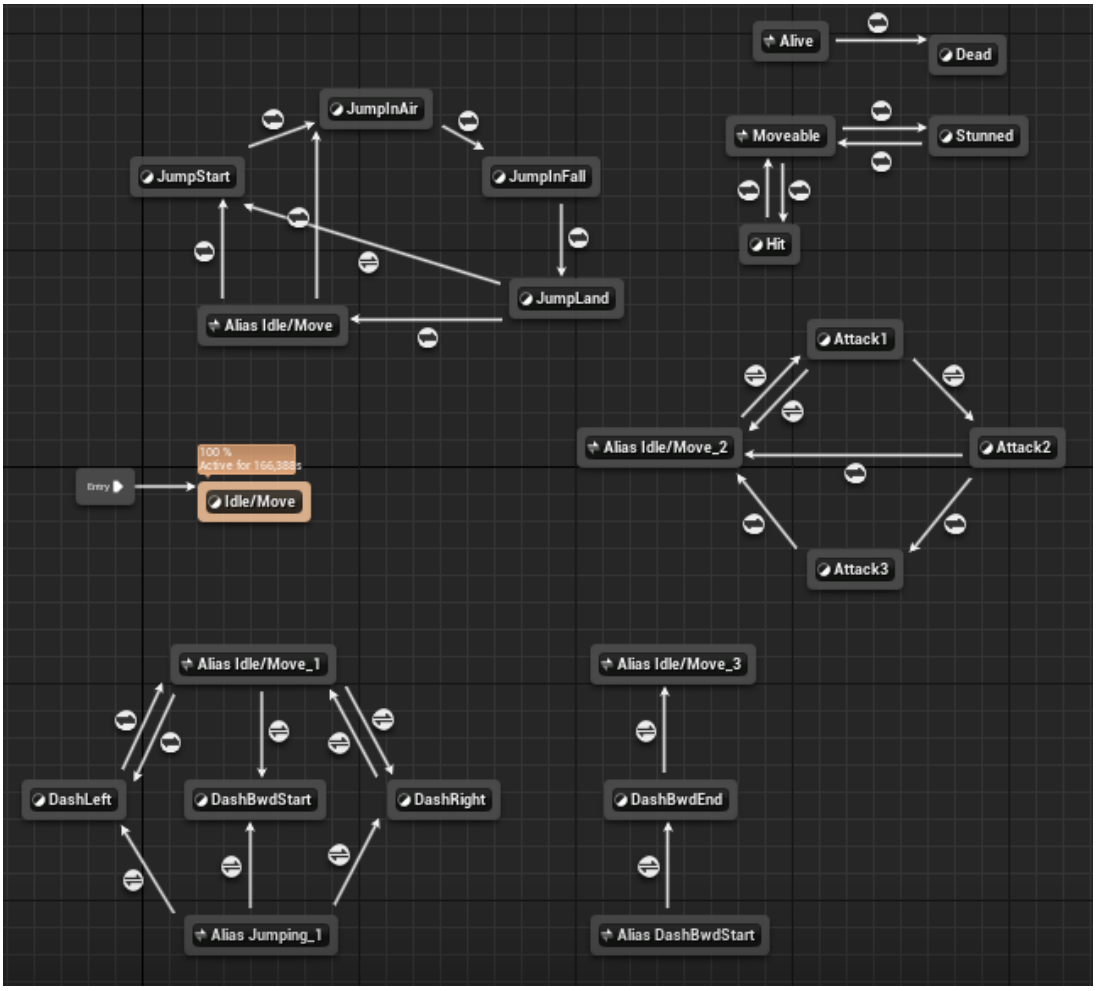

Figure 51: Animation State Machine Include Jump + Dash

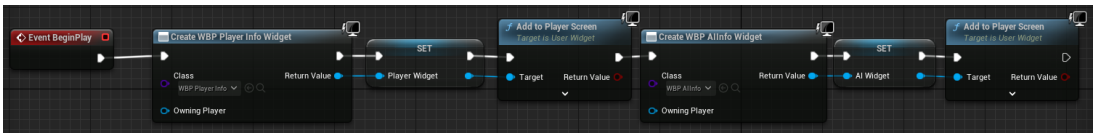

Figure 52: Event Begin Play of Player Controller

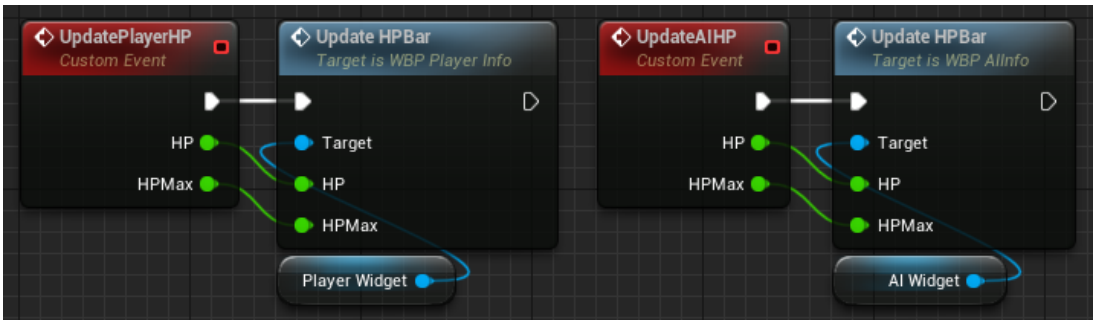

Figure 53: Custom events Update Player HP and Update AI HP

5.7. Player Controller Implementation

Player Controller is here used only for the communication between the UI and the play character. That is because an update UI system needs to be independent of the player character blueprint. Every other character that can be set as the player character can communicate with the UI elements in the same way. Event Begin Play is the logical first event to describe and is present in figure 52. This is only responsible for creating widgets which are the UI elements in Unreal Engine, saving their reference and adding them to the screen with Add to Player Screen nodes.

Two widgets that show the player and the AI HP points on a bar are created. Creation details behind widgets is not particularly relevant. The only important thing is that they update the visual representations of HP of both player and AI when called from the player controller or the AI. Requests for updating AI are implemented as events and can be seen in figure 53.

In the events of HP reaching zero for either player or AI, an appropriate screen with victorious or defeat messages appears on the screen. Since class cannot be forwarded as an argument, here there is some duplicate code. Only one of these is explained since the other only has different widget object created. Custom Event UI End Screen which is shown in figure 54 creates widget and the before adding it to the player screen the opacity is set to zero. After that the widget is added to the screen and a new looping event is called. In this event named Show End Widget the opacity is raised to the value of one and the timer function clears itself.

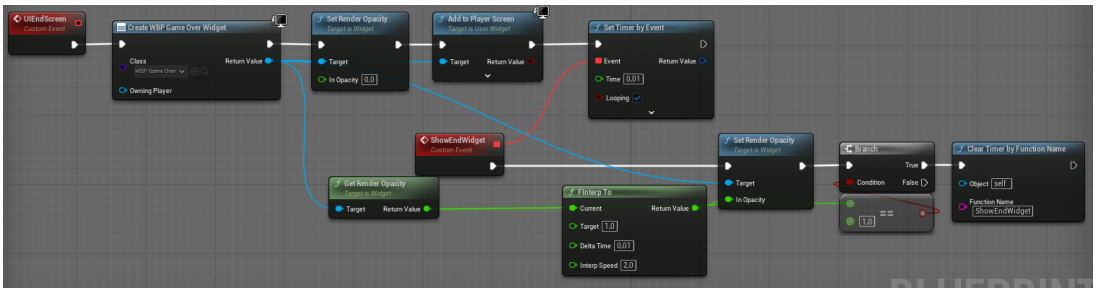

Figure 54: Custom events UI End Screen and Show End Widget

6. AI Implementation

Final product of this chapter is the implemented AI system with multiagent system as its main driving force. First, all the functionality of the AI character is taken care of like dealing and taking damage. After that the implementation moves onto the AI controller. This is the unit that receives messages from the agents and decides on who has the control over the AI unit. Then, each individual behavior of an AI is implemented as a Behavior Tree Task. There is a brief explanation of EQS System, Decoractors and Sevices in the behavior tree. After all that is done, the final step is a creation of Unreal Engine Behavior Trees.

6.1. Preparing AI Blueprint

This section is not that different from the preparation of the player character. The appropriate mesh and animation blueprint must be set with some kind of basic state with the idle animation in place so that the Capsule Collision component can be adjusted accordingly. Forward vector component is added for testing purposes. Damage sphere for dealing damage is also added to the player so that it can generate overlap events. There are of course some differences in AI actor and that is that this actor those not have any cameras, it has an indicator that was mentioned in the player implementation and it has a huge sphere for radial force generation which is used in one of the abilities. It also has the AI Perception component which is very useful for noticing actors in the environment around the AI but is not be used in this project. When all is set and done the Viewport looks like in figure 55.

6.2. AI Actor Base Functionality

As is the case with every blueprint or actor, firstly the Event Begin Play is looked at and explained. Here the first sign of the difference between the approaches to the animations here and in the player character can be noticed. The node Play Animation Montage plays a montage of a certain animation sequence. There does not need to be any difference in Animation Sequence and the Animation Montage. Here the name montage can be a bit misleading. Animation Montages in Unreal Engine are basically normal animation that are more adapted to be used in the code itself. Montages also have some more additional control over blending the animation that the montage consists of and other animation that may be active when montage is called while the Animation Sequence does not have that. One more really important thing is that playing animation on a mesh that has animation blueprint connected to it stops the animation blueprint and put the character back to the T-pose. So the animations themselves are more for the animation blueprint state machines and montages are for there to play animation in code.

Here the node Play Animation Montage is called to play one animation, after that the enemy gets and references the player character with a cast. After that the first event dispatcher is bound to this actor called EDP Attack in Progress. This information is important for the agent

Figure 55: BP_Enemy Viewport

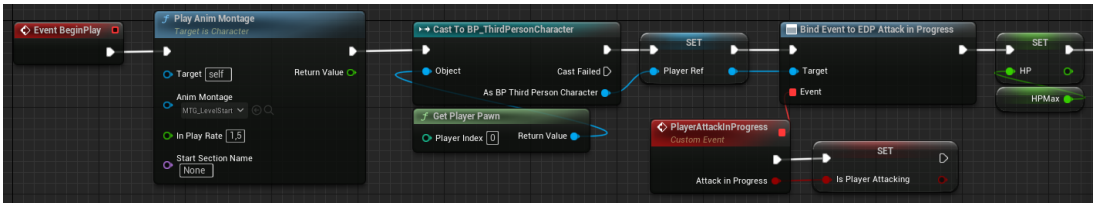

Figure 56: Event Begin Play of BP_Enemy 1/4

decision making. It can be seen that the process of binding requires a custom event that runs when the message from dispatcher is received. Here the event sets the value of variable Is Player Attacking. The last thing in figure 56 is setting the HP value to the HP Maximum value.

After that the timer starts the custom event named Distance To Player which runs every half a second. This information is also relevant to the agents in their decision making. After getting a reference to the player controller so that this actor can also communicate with the UI elements. The best approach would be for the AI controller to create its widget and then control it through it. This is not a large scale project so it is be just fine like this. Before updating the UI element there needs to be some delay because the creation is happening in another thread that does not have enough time to create both widgets. After that the Update AI HP is called and can be seen in figure 57.

Lastly, in Event Begin Play there are three more event dispatchers from the player character that are bound to the appropriate evens named Player Stun Update True and the opposite of that where the value of variable Is Player Stunned. Last bound event is named Play Death Celebration which is called when the player looses all HP. This can be observed in figure 58.

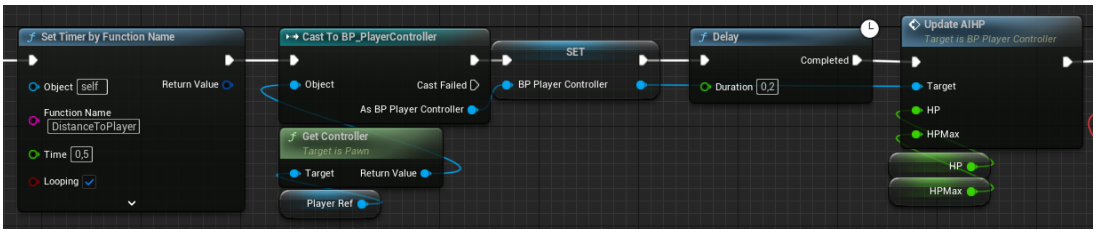

Figure 57: Event Begin Play of BP_Enemy 2/4

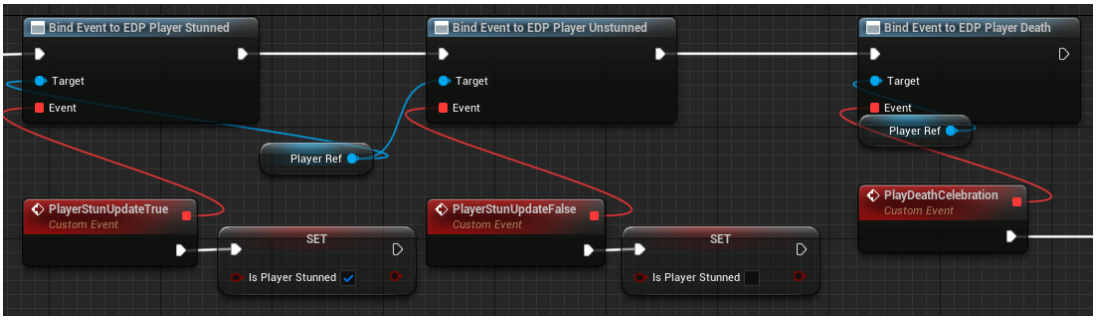

Figure 58: Event Begin Play of BP_Enemy 3/4

In this event the Blackboard value of AI named Is Alive is set to false which causes the abort function to be called in all behavior tree tasks. These nodes are present in figure 59. The same thing happens when the AI dies so it is just reusing the same variable. After that all behavior three logic must be stopped and that is done by calling the Stop Logic node with the Brain Component of the AI Controller as an argument. Lastly the mesh is tasked with playing the dance animation with some victorious voiceline being said by the AI actor.

Taking a closer look at the event Set Black Board 4 Animation Stop it is hard to avoid seeing that the blackboard has specific node functions for setting blackboard values and even has a different set function for each data type. Described can be seen in figure 60.

Figure 61 shows two simple events. One of them is in loop that is running every half a second and is measuring the distance between player and AI and saving that value into Distance variable. Usefulness of this event is seen in the implementation of the AI Controller down the line. Another event or function is for calling update on the UI, which is called on taking damage event.

Event Any Damage is pretty similar to the one already explained in the player character so the focus is more on the differences. After updating both the HP variable and UI a function for storing recent damage taken is called with the damage amount as the argument. This event

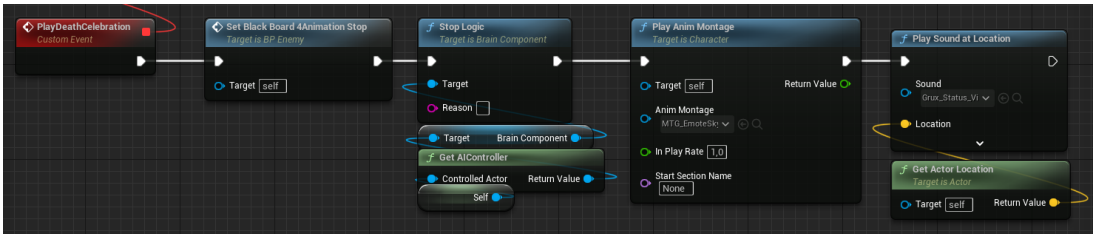

Figure 59: Event Begin Play of BP_Enemy 4/4

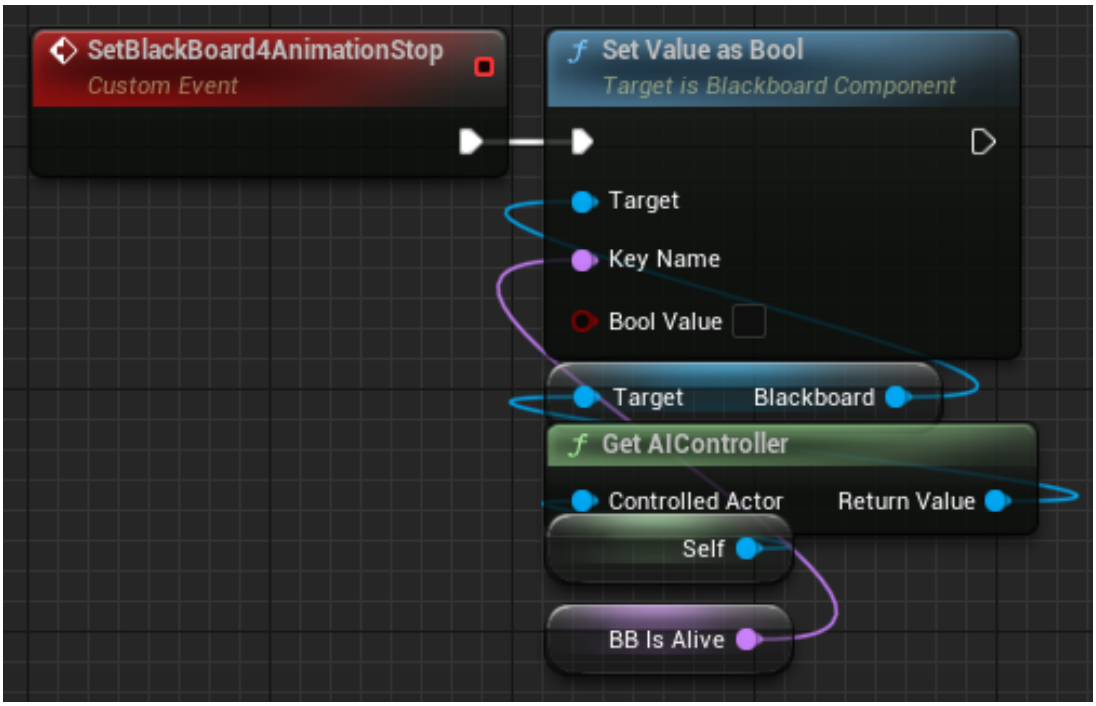

Figure 60: Custom Event Set Black Board 4 Animation Stop

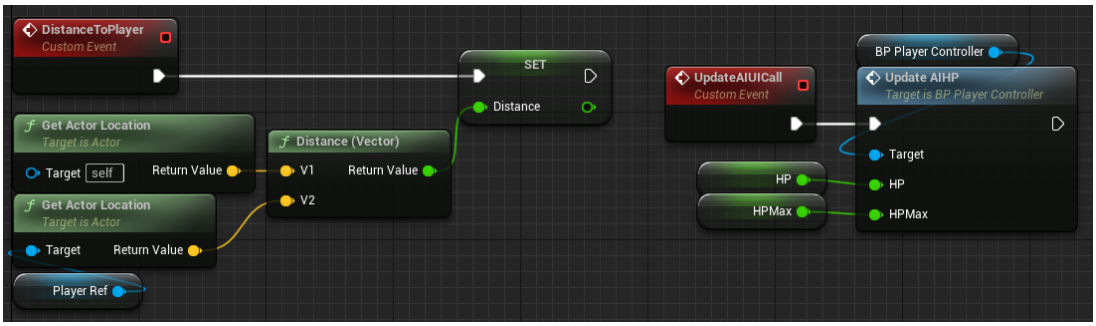

Figure 61: Custom events Distance To Player and Update AI UI Call

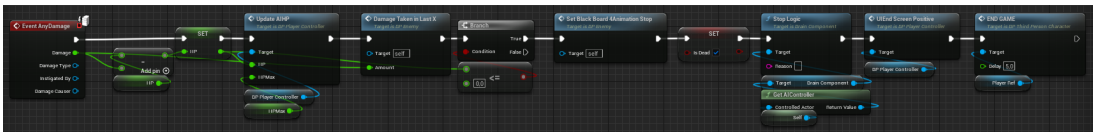

Figure 62: Event Any Damage of BP_Enemy

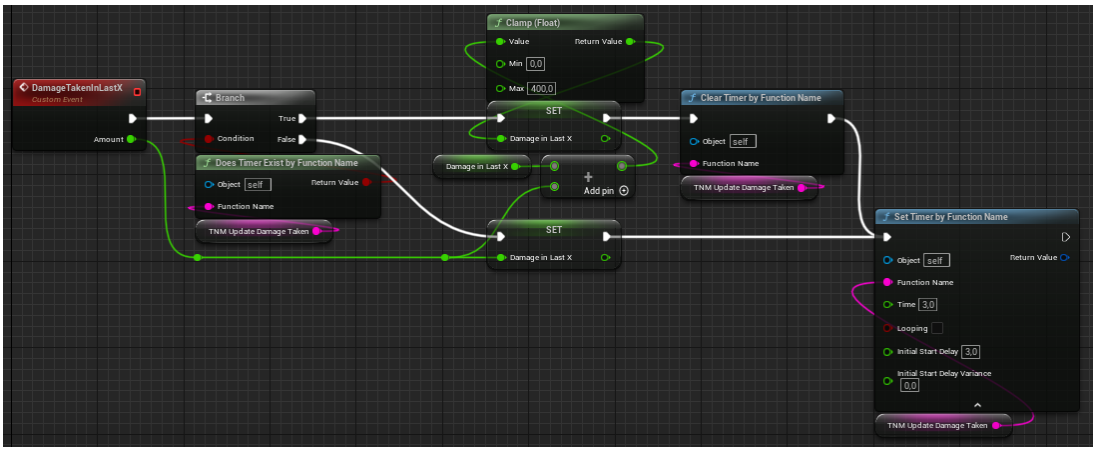

Figure 63: Custom Event Damage Taken In Last X

is explained in the next figure 63 while the focus for now stays in figure **??**. If AI has no more HP both local and blackboard values are updated. Is Dead variable is used for the animation blueprint. Stoppage of logic was already explained but the UI End Screen Positive is new. It creates an end screen and ends the program in defined five seconds.

Custom event Damage Taken In Last X may look a bit confusing at first glance but it is not that complicated actually. This event is also important for agent decision making. It starts with a branch that requests the information about the existence of the in progress function named Update Damage Taken. If such function is not running the Damage In Last X is set to the Amount value and the timer starts mentioned function or event. If the timer is already active the Amount value is added to already existing value and is clamped before being set. Next node clears the timer and the times is set yet again. Why that is done becomes much more clear upon looking at the next figure 64 which reveals the full context of this node set of figure **??**.

Here is the mission ingredient and that is the event that sets the value of Damage In Last X to zero. That means that if the information is not updated in defined interval the last damage taken is be forgotten. The other event in figure 64 is the same as in the player character so there is no need for additional explanation.

Last node complex that needs explanation is referring to the damage window in figure 65. When event Deal Damage is called the damage window is opened. Only things that can close this damage window is either the timeout or the act of dealing damage. If there is no overlap between the player and Damage Sphere component the Damage Timeout event is be called. This clears and invalidate both the Damage Tick event and itself. Damage Tick event is constantly checking for the overlap between the player and the damage component. This is more demanding approach but it also works. Based on if the attack is light or heavy

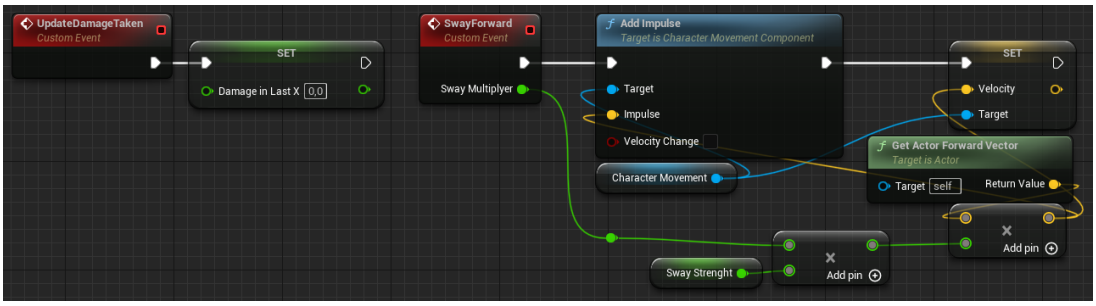

Figure 64: Custom events Update Damage Taken and Sway Forward

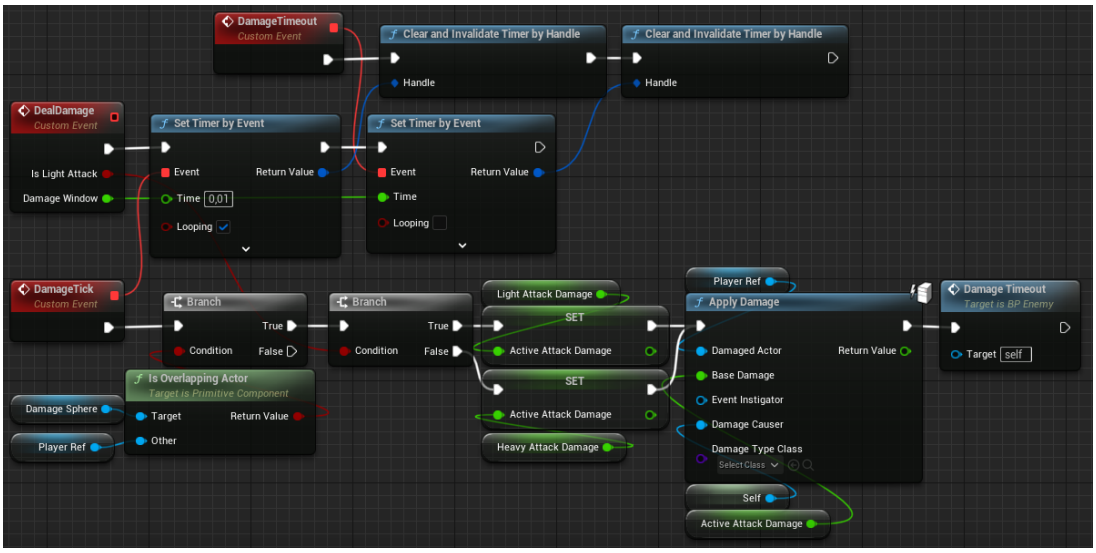

Figure 65: Custom events for creating damage window

the appropriate amount of damage is forwarded to the Damaged Actor which in this project is always the player character. After applying damage, the event Damage Timeout is called so that damage can be dealt only once per damage window.

The animation blueprint of AI character is somewhat simpler but similar to the player animation blueprint. For this reason the additional information regarding the animation control inside of animation blueprint is not provided. Next on the repertoire is AI controller.

6.3. AI Controller Implementation

In the AI controller the AI communicates with agents and the behavior tree. Node that is first in he Event Begin play execution line is the Delay node. This allows AI actor to play the appropriate animation before jumping into the smart behavior. Behavior Tree is run by Run Behavior Tree node. References that are needed here are the references to the blackboard and AI actor so that communication can be established. After that the update loop for agents is started with timer node and value of Is Alive boolean in blackboard is set to true. This concludes the content of the figure 66.

Event Update Agents is a simple event in figure 67 that sends the message to all actors

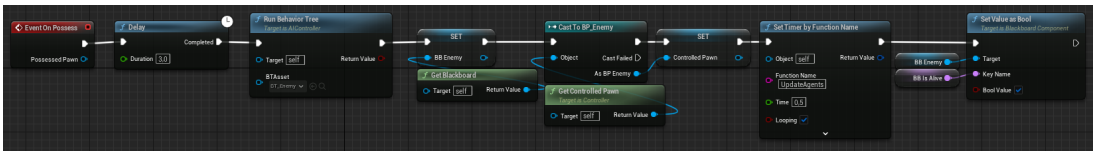

Figure 66: Event Begin Play of AIC_Enemy

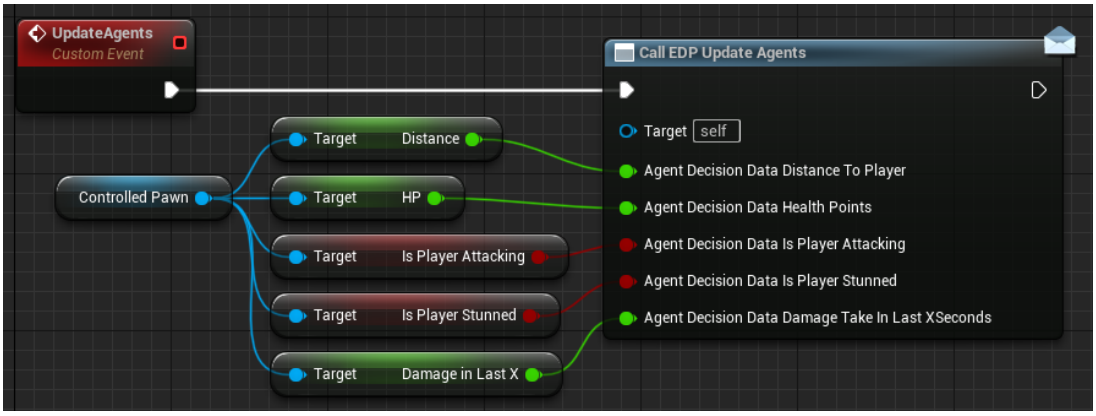

Figure 67: Custom Event Update Agents

that implement an event dispatcher named EDP Update Agents. Here the contents of the custom data structure called STR Agent Decision Data are exposed. In consists of three floats and two boolean variables. All variables are extracted from the AI character.

AI controller is unaware of the agents themselves but he can identify them by their identification number (hereafter ID). The controller needs to have information about the number of agents so that he can know if all agents have sent their bids. Event Agent Answer is the event that is called from the agents themselves when they have calculated the bid. Until all agents send their bids the function Determine Lead Agent in figure 68 is not called.

Determination of the lead agent is a simple process which looks at all agent bids and chooses the highest offer. The highest value is saved in variable named Proposition Bid which can bee seen in figure 69. This is not the only value that is stored. Index of the chosen bid which is also the ID of that agent is also saved for use case that is explained later.

The rest of the process shown in figure 70 only requires setting the Agent In Control blackboard value, calling another event dispatcher which notifies agents about the victorious

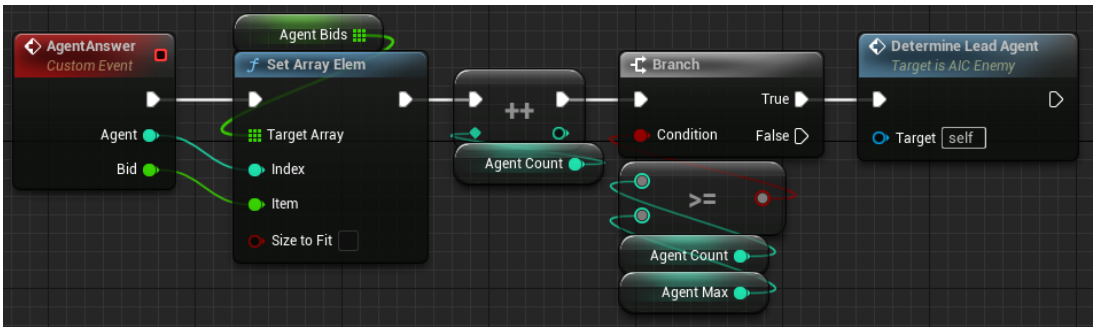

Figure 68: Custom Event Agent Answer

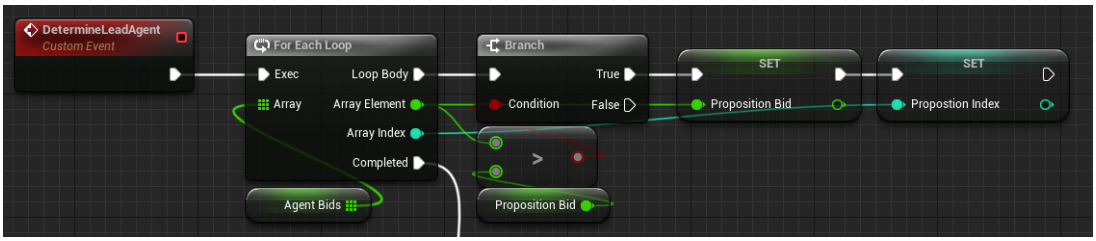

Figure 69: Custom Event Determine Lead Agent 1/2

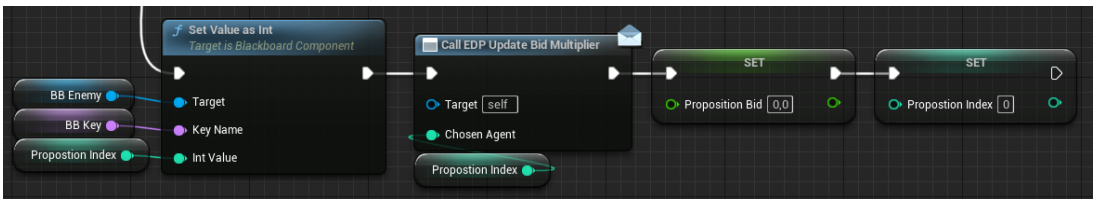

Figure 70: Custom Event Determine Lead Agent 2/2

among them and resetting both the Proposition Bid and the Proposition Index. This is all the functionality behind AI controller. Agents are implemented next.

6.4. Agent Implementation

The main idea behind the agent implementation is that each agent is given a different set of values or rules on how to interpret the given data but they all have the same formula for creating a bid. Child blueprints of the main agent actor are given IDs and custom rules on how to make decisions in creating bids. The first event that is explained is the Event Begin Play which immediately gets the reference to the AI controller. Now that this reference exists two of the event dispatchers that were created in AI controller can be bound to specific events. Lastly in figure 71, there is a function named Prepare Scalars (Normalize) which prepares the scalar values for the use inside the formula.

This function is called on the Begin Play event because the scalar value does not change in runtime and it easier to assign their values with integer rather then relational percentage which adds up to the value of one or in percentage one hundred percent. Figure 72 contains two for loops. The first loop calculates the sum of all scalars. The second loops goes through each scalar and divides it by the Scalar Sum variable in order to get a influence percent. All new values are store inside the Normalized Scalars array.

Next, one of the bound functions by event dispatchers that is displayed is the event Update Bid Multiplier which rises the bid multiplier each time the agent did not win in the bid competition. However, if he wins the bidding process his multiplier is reset to the value of one.

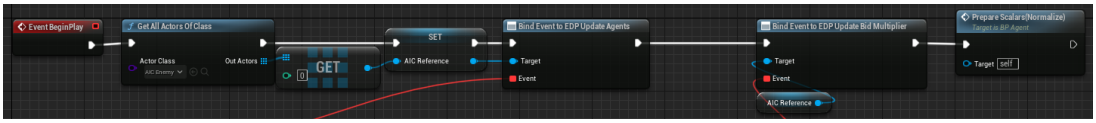

Figure 71: Event Begin Play of BP_Agent

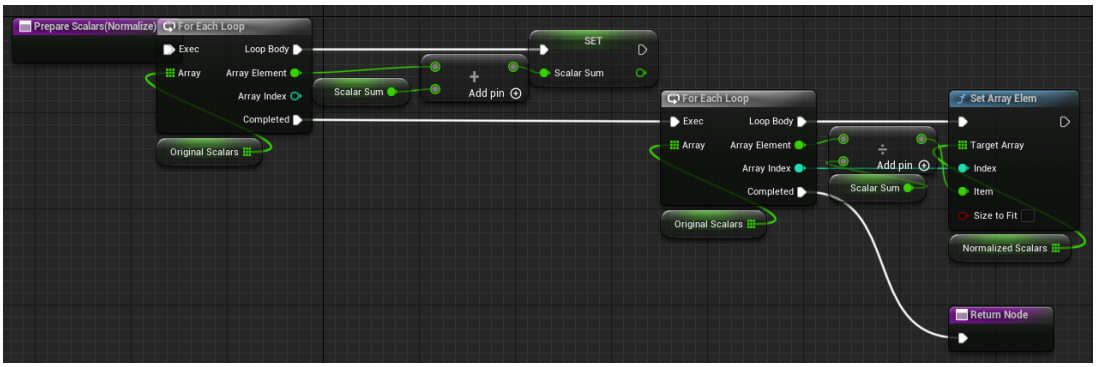

Figure 72: Function Prepare Scalars (Normalize)

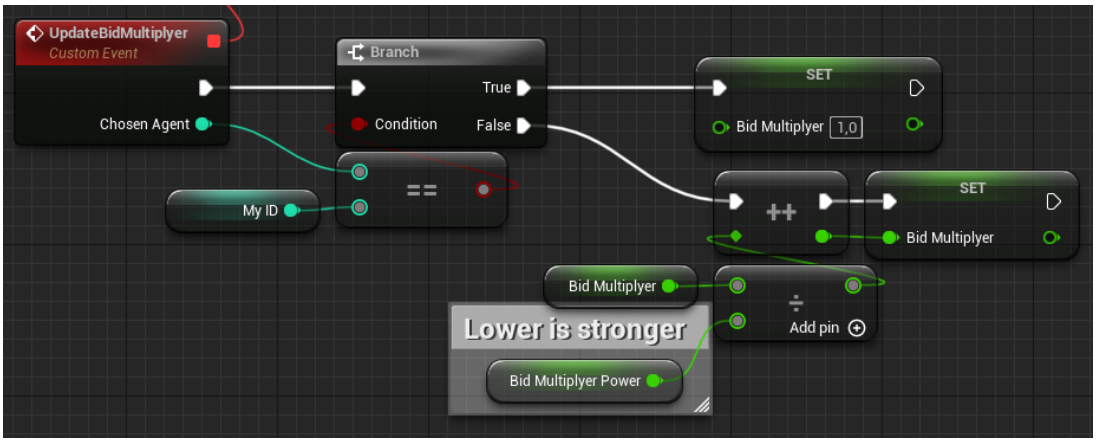

Figure 73: Custom Event Recalculate Proposition Strength

Explained behavior can be seen in figure 73 where if the winning agent has the same bid as the My ID variable the multiplier is reset.

Bid calculation happens inside this event called Recalculate Proposition Strength in figure 74. Function in which the bids are calculate is called FN Recalculate Proposition Strength. It requires an object that holds all agent decision relevant data and it pops out a bid which is then send together with the ID of the particular agent so that the AI controller can recognize the agent.

This function in figure 75 consists of three other functions which produce the bid in the end. First in the line is preparing values that are generated by the virtual environment. Based

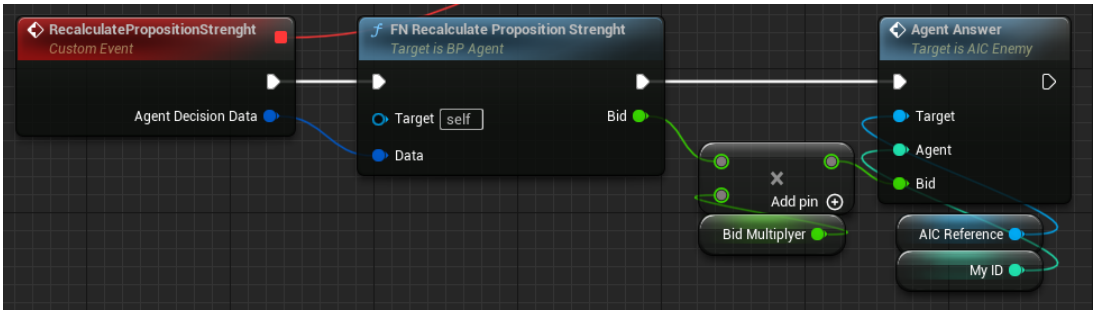

Figure 74: Custom Event Update Bid Multiplier

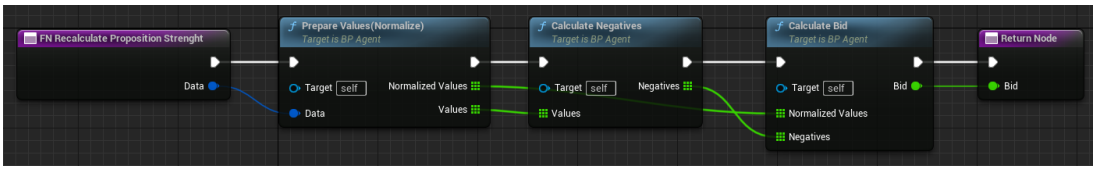

Figure 75: Function FN Recalculate Proposition Strength

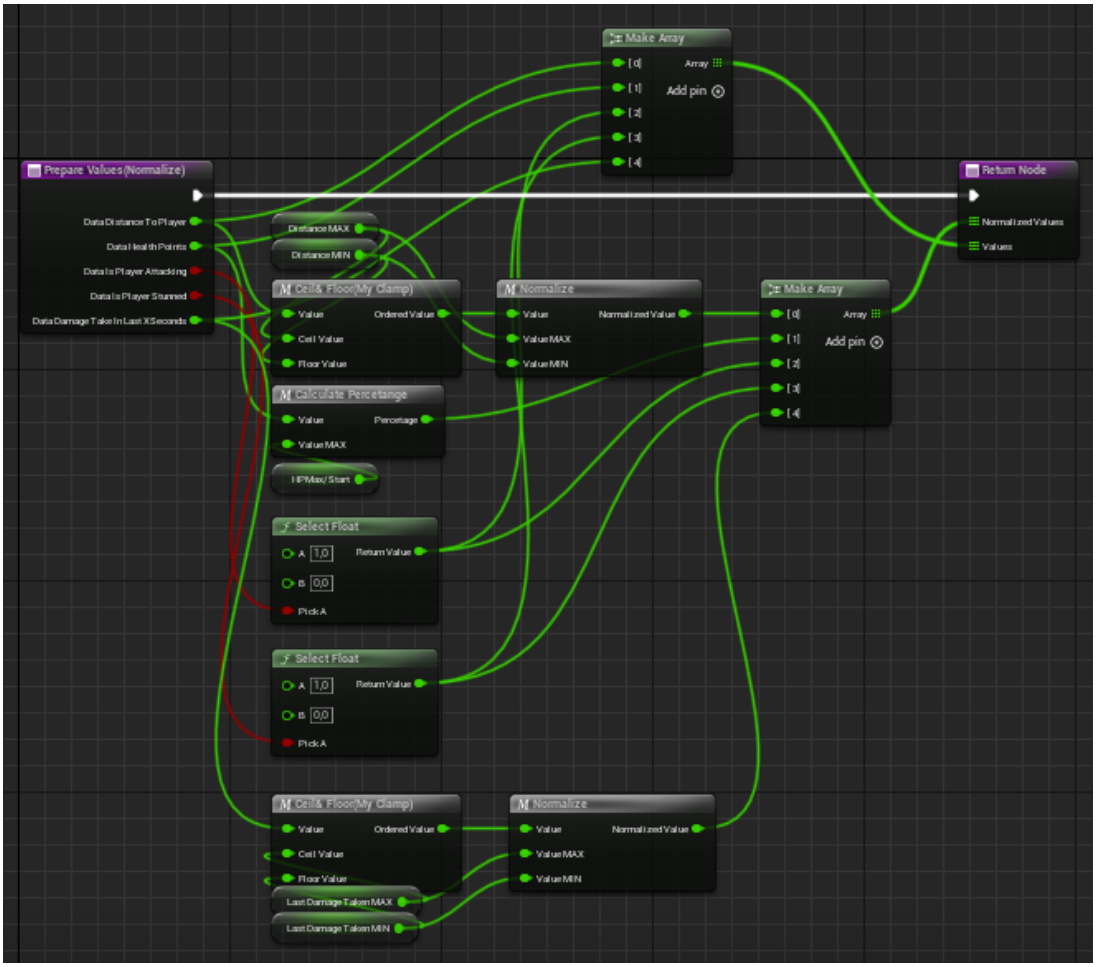

Figure 76: Function Prepare Scalars (Normalize)

on set norms the negatives are calculated which define if the result of individual values should be negated or not. It is important to mention that those names do not have any significant value outside of personal understanding of created formula. With normalized values and calculated negatives the creation of the final bid can begin.

Normalization of individual values mean a different thing depending on the given value. Distance To Player variable needs to clamped and normalized just like the variable Damage Take In Last X Seconds. HP variable needs to be converted into percentage which is also a normalization process. Two values that are left either become one or zero based on whether their boolean value is true or false. Two arrays of original and normalized values are created in figure 76 and they are both return values of this function.

Ceil and Floor macro confides the variable inside provided range. If the value of the

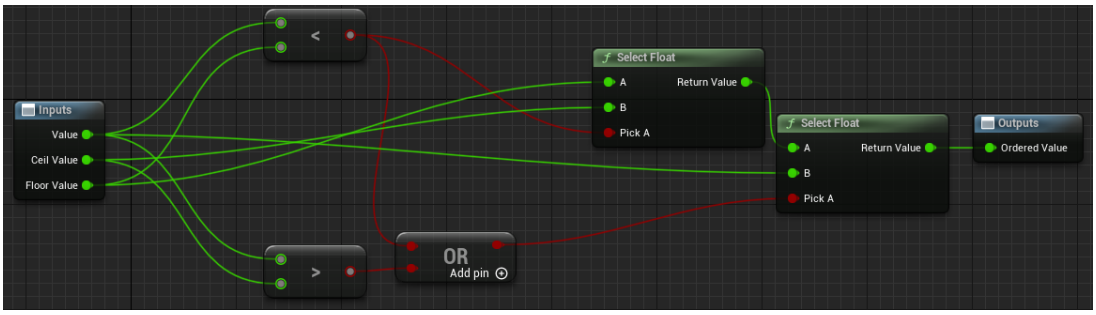

Figure 77: Macro Ceil And Floor

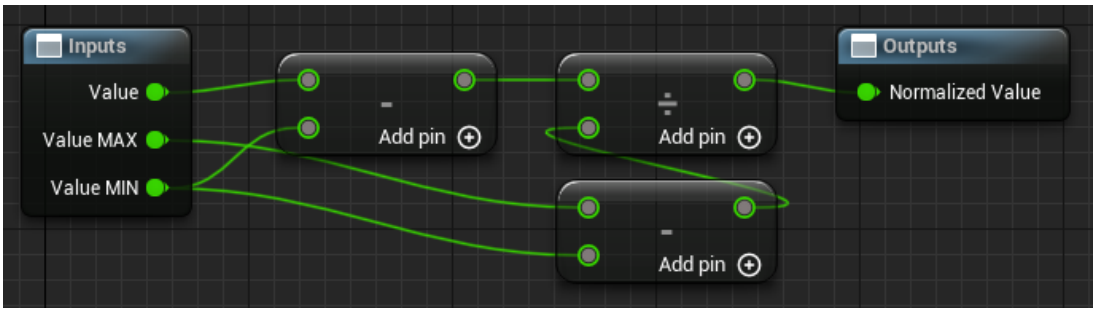

Figure 78: Macro Normalize

main variable is less then the minimal value then the minimal value is the result; if the value of the main variable is greater then the maximal value then the maximal value is the result. Only when both of those conditions are not met does the variable stay the same and this can be seen in figure 77.

Normalization is a standard process and it requires to divide the result of subtraction between the original value and the minimal value and the result of subtraction between the maximum value and the minimal value. This macro can be seen in figure 78. Output of this macro is normalized value.

Percentage is a specific kind of normalization which occurs when the minimum value is zero. In that case both of division sides do not subtract anything and they are only left with the maximum and original value. This is present in figure 79.

Negatives are calculated based on the the original values of the virtual world, the breaking point of those values which dictates the increase of the final value based on its distance to the breaking point and the norms themselves. Norms dictate if the increase in original value

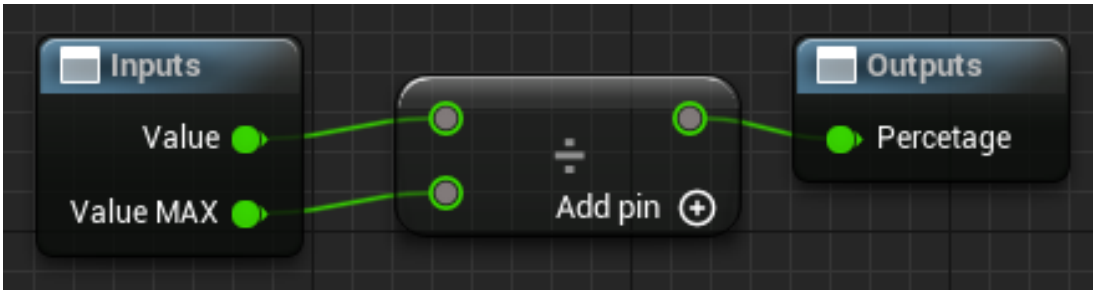

Figure 79: Macro Calculate Percentage

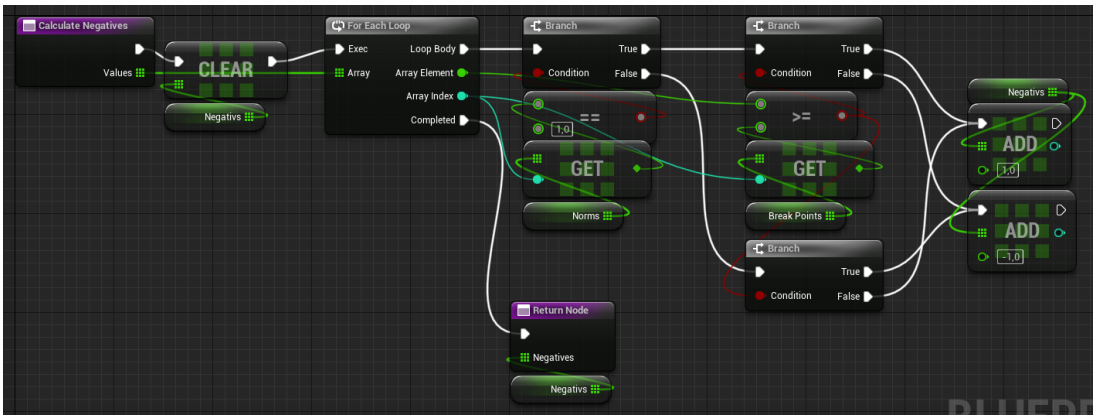

Figure 80: Function Calculate Negatives

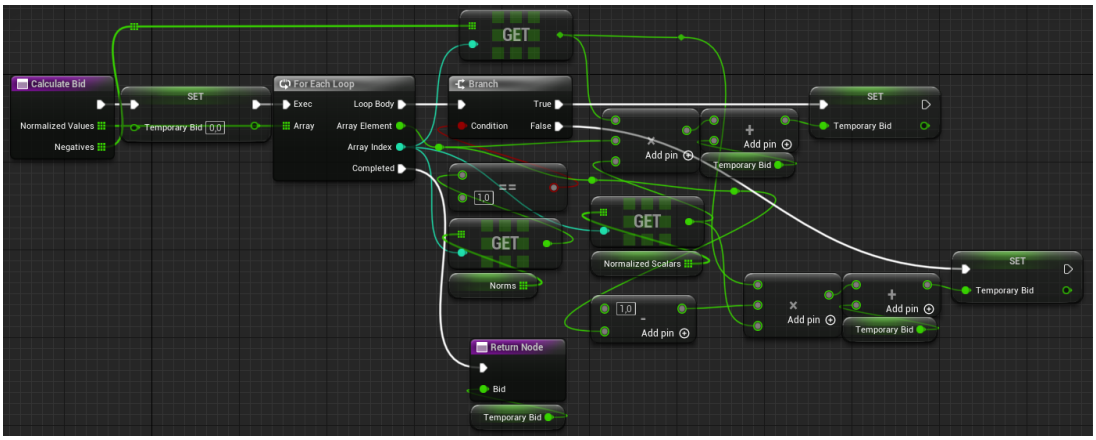

Figure 81: Function Calculate Bid

beyond the breaking should be looked as something positive or negative. With all these values in figure 80 the Negatives are either set to he value of one or minus one.

The final function that produces the bid value is named Calculate Bid and can be seen in figure 81. The decision here is between two execution lines of setting the value of the Temporary Bid variable. If the value of the norm is equal to one, number that is added to the result of Temporary Bid variable is the result of multiplication of three numbers which are the negative, the normalized value itself and the normalized scalar value. The only difference is that if the norm is equal to minus one, the normalized value becomes inverted with one minus value expression. When all the values are added together the final number represent the bid.

After the main agent blueprint is complete three child blueprints or actors can be created. All have a unique ID and different Original Scalars, Break Points and Norms. After all that is set they need to be added to the world or else their code does not run.

6.5. Implementation of Individual Tasks

Behavior Tree Task is the smallest action unit inside of a Behavior Tree. There are some prebuilt tasks that are used but the custom one give the AI a unique and interesting behavior.

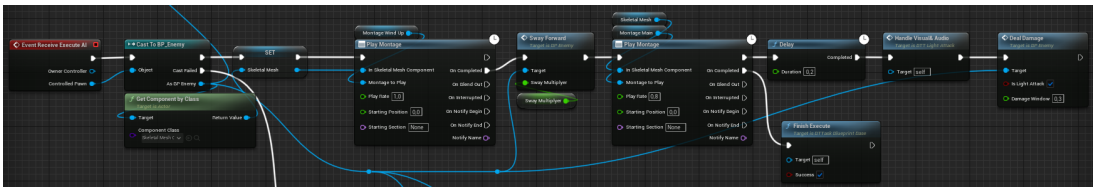

Figure 82: Event Receive Execute AI of BTT_LightAttack 1/3

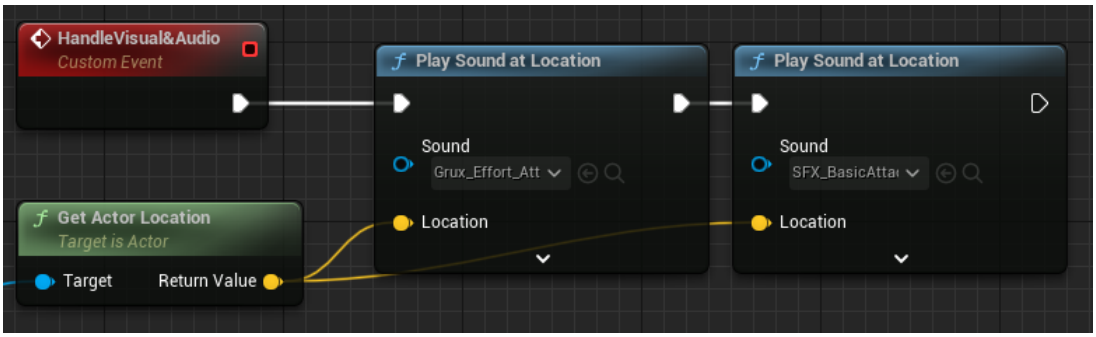

Figure 83: Event Receive Execute AI of BTT_LightAttack 2/3

Firstly, tasks regarding close range attacks are made and examined. Figure 82 is showing how each of these tasks is started and that is with casting to the AI class. For this specific use of montages a reference to the skeletal mesh is also needed. In the event of the attack, previously implemented Sway Forward event is called. After a short delay a function for handling visual and sound effects is called. This event is further examined in figure 83. For now, it is important to mention that the task also calls the custom Deal Damage event and after the animation montage is complete the task is successfully finished.

All tasks start with a specific event called Event Receive Execute AI and all the tasks need to have at least one node called Finish Execute which tells the Behavior Tree that the task is finished. Additional information about task completion is set via boolean argument Success. This can affect the behavior of the Behavior Tree so it should never be only left with one Finish Execute that fails, at least if that is not the desired behavior and it usually is not.

The event Handle Visual And Audio is pretty similar in each task that has any similarly named event so the explanation is only provided here. No matter if it is the case of visual or sound effect, everything is spawned or played at the location of the AI and besides specific chosen sound or effect there is not much to discuss here.

Each task that has some kind of animation montage playing within it has Event Receive Abort AI. This event is called if the player character or AI are left without HP. It simply stops any montages that could be playing at any moment inside the task. More tasks have the same implementation so this part of the code is not be mentioned again since there is no need for that. Besides the abort event, if by some miracle the casting to the AI actor fails the task finishes with the variable Success set to false as shown in figure 84.

The only difference between the task of light attack and the task of heavy attack is the the heavy attack has only one montage that is played and delay is slightly different to accommodate the difference in animation. This can be observed in figure 85. In an event call

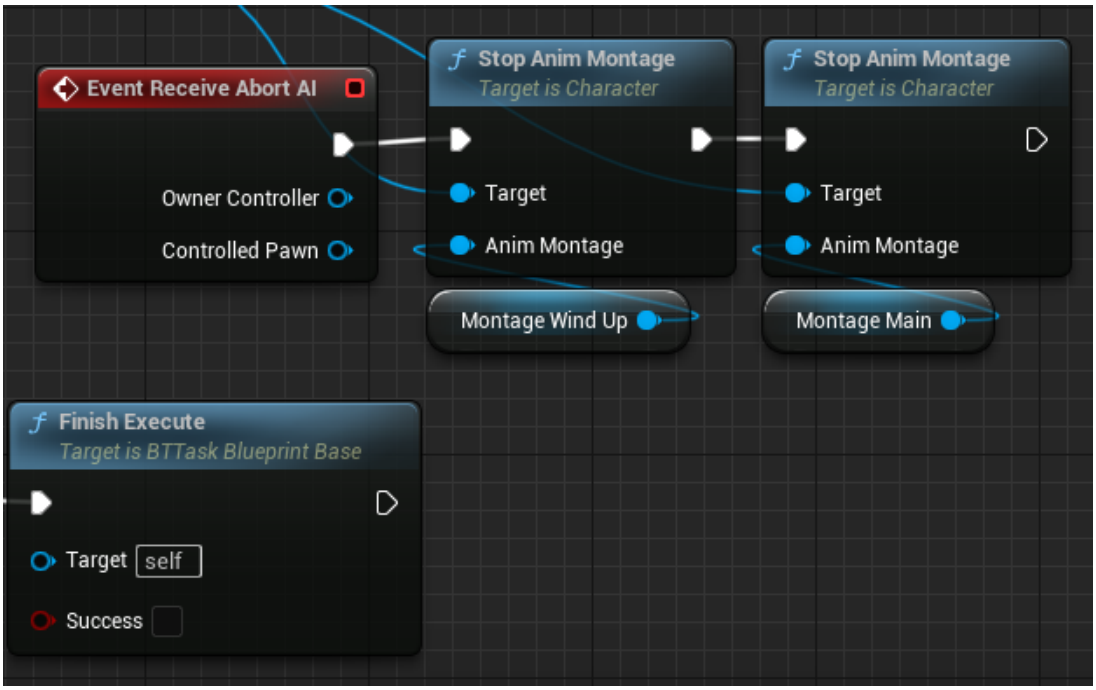

Figure 84: Event Receive Execute AI of BTT_LightAttack 3/3

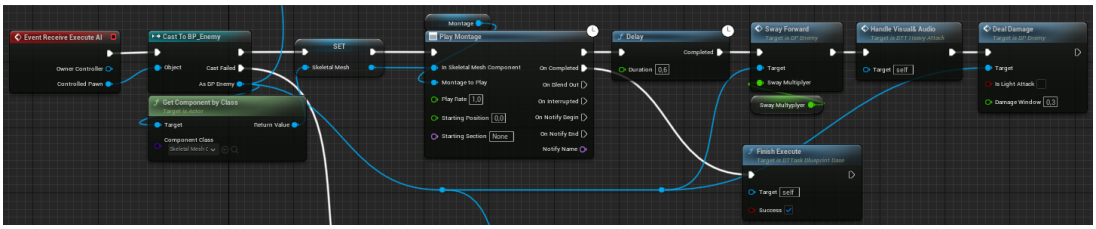

Figure 85: Event Receive Execute AI of BTT_HeavyAttack

Deal Damage the Light Attack boolean is set to false and that is it.

Next task is for the force field behavior for which the Radial Force component was added to the AI character. In the beginning of the task the local variable Current Cycle is assigned to zero. This is probably unnecessary but better safe then sorry. After the initial cast a timer that creates a looping event is called. This can bee see in figure 86.

An event Ask For Impulse plays the animation montage, fires the impulse and handles the visual and sound effects for better player experience. This steps are repeated until the Current Cycle variable is greater or equal to the Number Of Cycles variable. And that is really it. One of the future tasks has a similar behavior to this one. Described can be seen in figure 87 which is the second part of the figure 86.

Healing is done in one of the tasks as well. Firstly the heal target has to be calculated which represents the upper value that the healing process can not excess. This is calculated with the Heal 4 Percent Health variable which when multiplied by HP Maximum produces the heal amount. This amount can then be added to the current HP but also clamped at HP Maximum so that the Heal Target value can never exceed the HP Maximum value. Play rate of the animation is also calculated here so that it covers the duration period of the healing

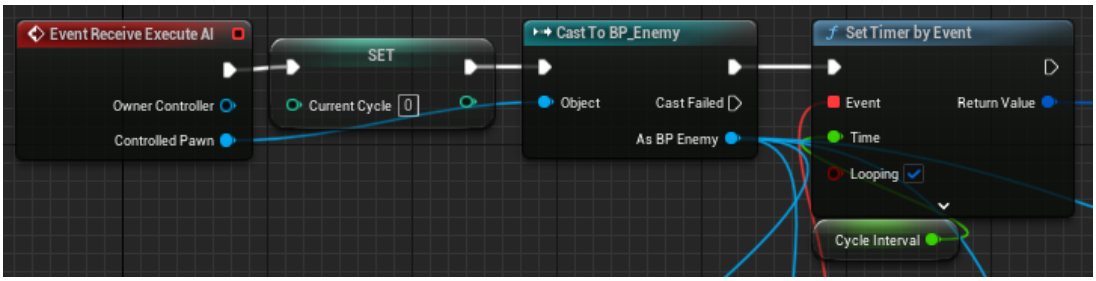

Figure 86: Event Receive Execute AI of BTT_ForceField 1/2

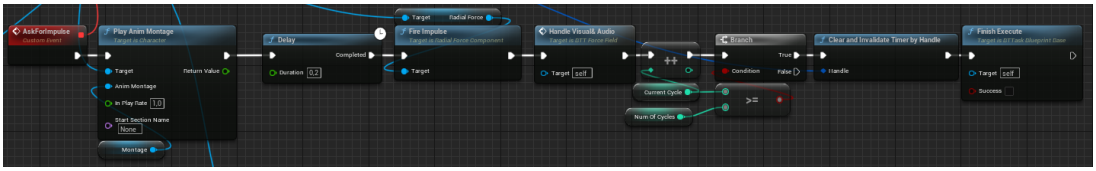

Figure 87: Event Receive Execute AI of BTT_ForceField 2/2

process. Play length is calculated by dividing the play length of animation montage with the Heal Duration variable. Function Start Visual and Audio takes care of the communicating the healing process to the player and two additional events are called and both can be seen in figure 88. One of which is not looping and the other one is.

The event that is looping is named Heal and, as the name suggest, it heals the AI over time. It interpolates the current HP value towards the calculated Heal Target and it also updates the UI. This event is interrupted by itself the the HP value becomes equal to HP Maximum value which is shown in figure 89.

Event Heal Time is called after the specified Heal Duration or if the HP value becomes equal to the HP Maximum value as stated earlier. This event invalidates both timers and after stopping the looping visual and sound effects it finishes. Figure 90 is showing the described behaviour.

Teleporting capabilities require two animation montages that are played on the start and the end of the teleportation process. Delay waits for the first animation montage to finish before teleporting the AI to the specified location. Both at the start and at the end of the teleporting process there are accompanying visual and sound effects which can be noticed in figure 91. The task is finished when the second animation montage finishes.

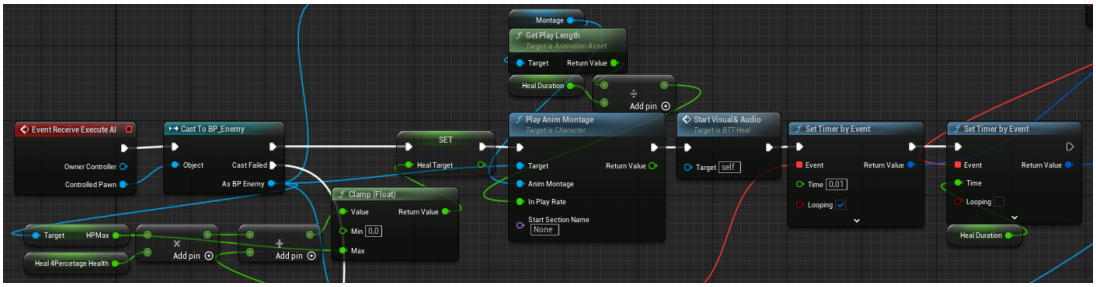

Figure 88: Event Receive Execute AI of BTT_Heal 1/3

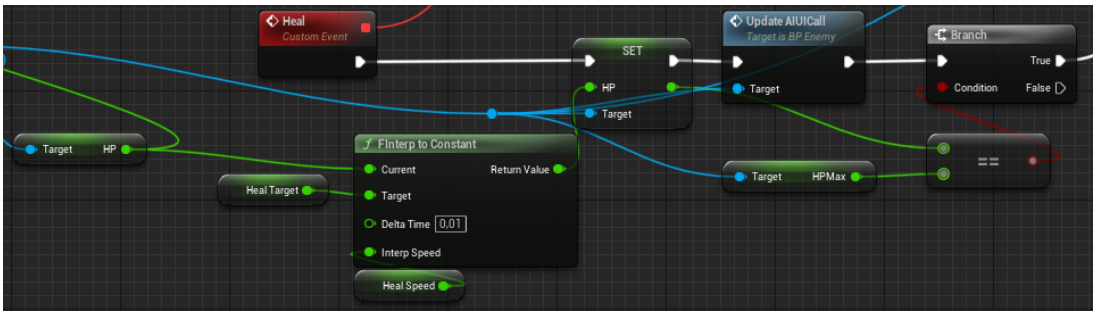

Figure 89: Event Receive Execute AI of BTT_Heal 2/3

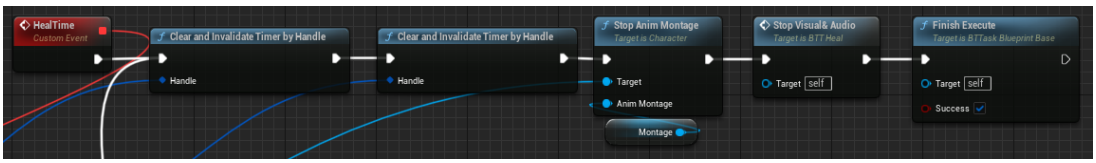

Figure 90: Event Receive Execute AI of BTT_Heal 3/3

Firing the stun projectile and an explosive projectile have extremely similar tasks so only the first of the two is explained with the help of visual presentation in figure 92. Besides playing the appropriate animation montage and spawning an actor of certain class there is nothing special going on right here. Node Spawn Actor From Class is used for spawning projectiles and this is the first appearance of this node in this project. Spawn location is pushed above the AI so that it can clearly be seen.

Task that has to spawn a burst fire looks like a crossover between force field task and the stun or explosion projectile spawning task which can be seen in figure 93. There is counter and a Spawn Actor From Class node. Spawn location is also placed above the AI and the task finishes when it has gone through all of the cycles that are defined by a variable named Burst Rounds.

6.6. EQS, Services and Projectiles

The short term EQS stands for Environment Query System and it is used to create the kind of queries that are relevant to the virtual world locations. EQS system creates, ranks and discards certain locations based on defined parameters and tests under which they run. For this project only two of all possible test were used. One test to confirm that the AI can reach a specified location with its navigational capabilities and the other is the Distance Test where locations in certain range are preferred. All used EQS are similar and there is no need to explain the difference in detail. They mostly vary in acceptable minimum and maximum distance and

Figure 91: Event Receive Execute AI of BTT_Teleport

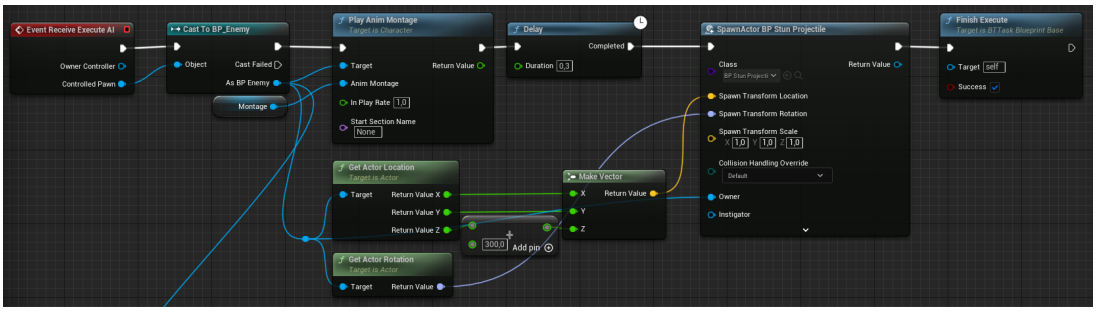

Figure 92: Event Receive Execute AI of BTT_StunProjcetile

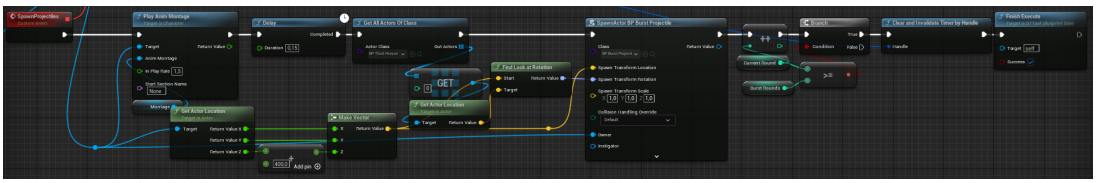

Figure 93: Event Receive Execute AI of BTT_BurstFire

the distance actor which can be the AI itself or the player character. One of the EQS is shown in figure 94. Distance to any other object besides the caller of the query needs additional setup. For that reason the object called Environment Query Context Player was created.

If a random task sequence is to be picked from the behavior tree, certain random number generation is required. This can be achieved with Services. All the created services serve the same idea of picking a random integer number. Since services can not have direct inputs more of them are created to account for all of possible numbers of needed task sequences. One of the services looks like the displayed node connections in figure 95. It only sets a blackboard value to a random integer.

Projectile classes need to be examined as well. Besides the usual collision and visual preparation two of the three projectiles are a homing projectile. In figure 96 there is an Event Begin Play of the stun projectile actor. The Homing Target component is set to a player character scene component and then the projectile is set to be a homing projectile. Spawn sound communicates the new existence of such projectile to the player.

The Event Hit of the same projectile is really simple. Whatever happens, visual and sound effects are spawned and the actor is destroyed. If however, the actor hits the player it stuns him by calling Stunned event which deactivates movement for brief period of time. Described can be seen in figure 97.

Explosive projectile has some difference to the stun projectile. Firstly it always fires an impulse on the impact location and it applies damage if the player character was in sufficient range of the explosion. This is checked by getting the overlapping actor of the Blast Radius. It can be observed in figure 98 that if there is an actor of class BP Third Person Character and it has a tag Player it takes damage.

The last type of the projectile is the is the burst projectile that can be seen in figure 99. This projectile is pretty much the simplified version of the explosion projectile. In that sense

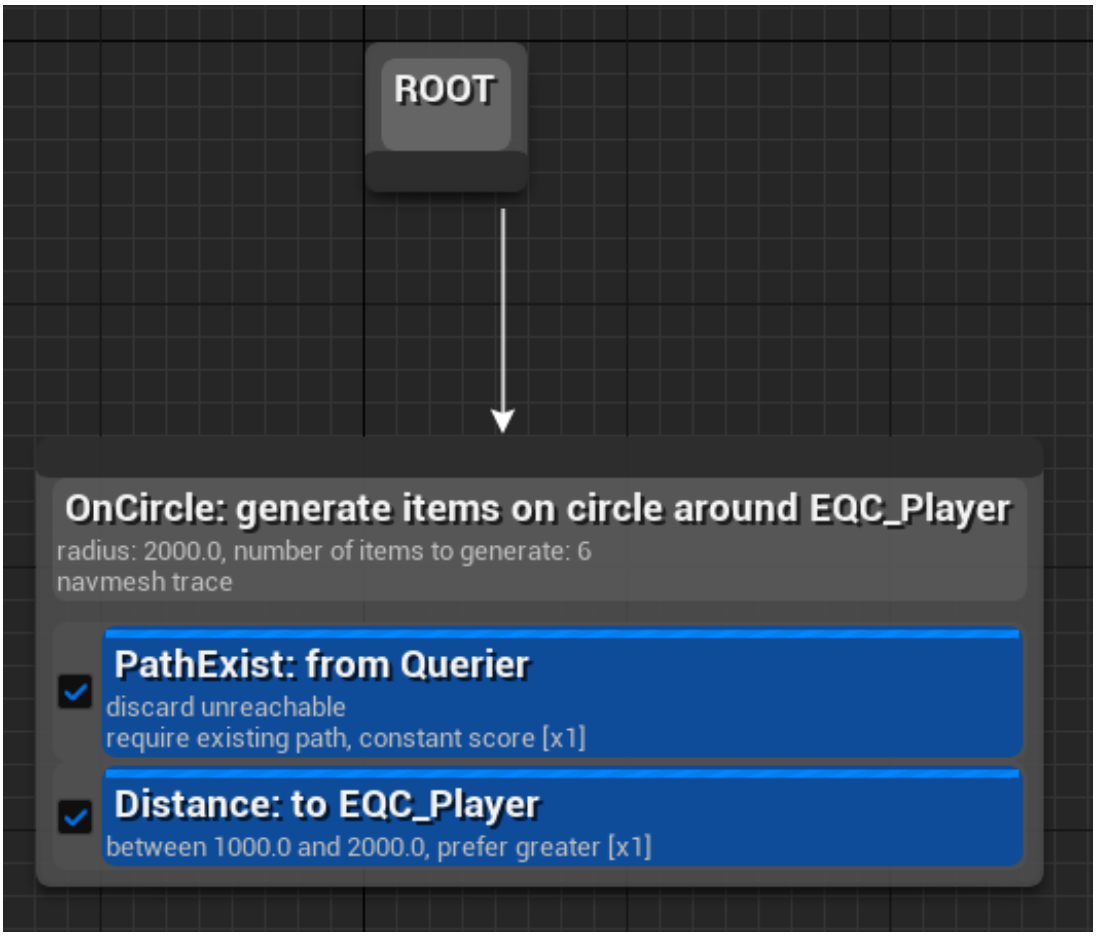

Figure 94: Environment Query System Distance Location

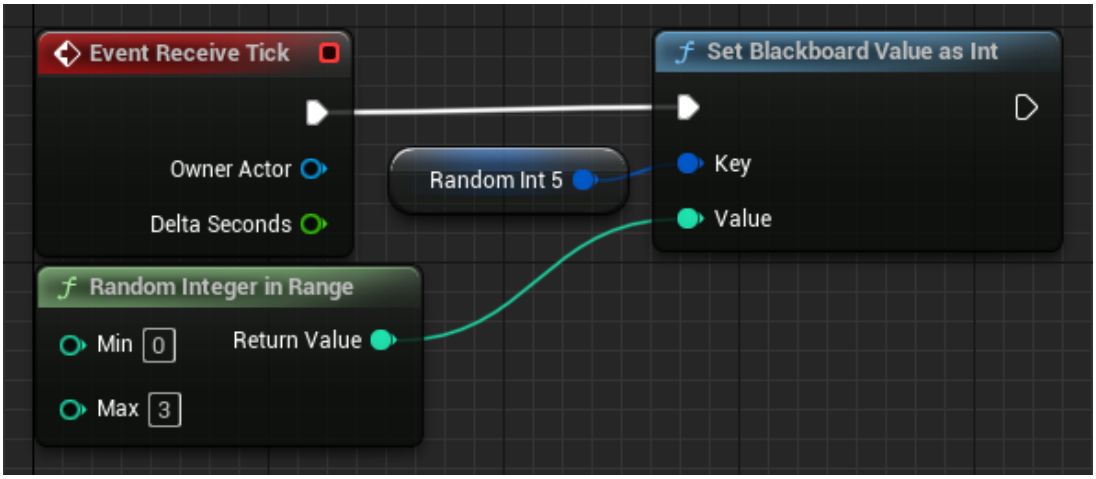

Figure 95: Behavior Tree Service Random Integer 5

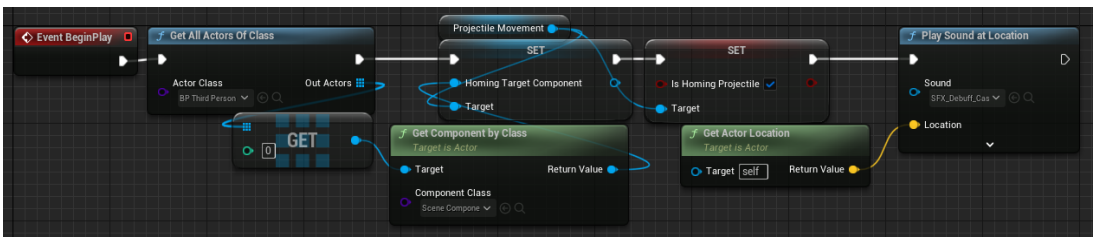

Figure 96: Event Begin Play of BP_StunProjectile

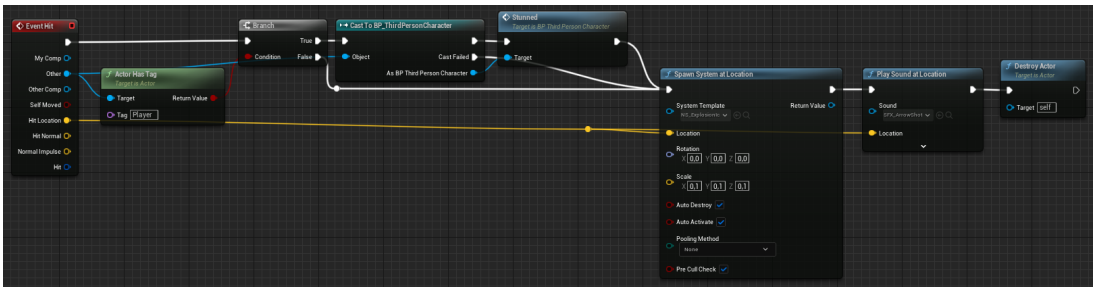

Figure 97: Event Hit of BP_StunProjectile

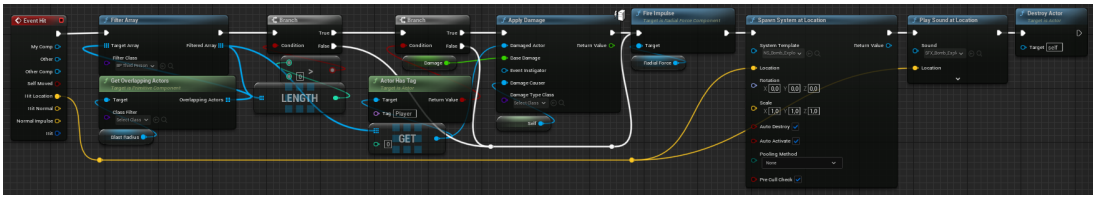

Figure 98: Event Hit of BP_ExplosionProjectile

it only damages the player if it was a direct hit and after that it also applies some damage. Visual and sound effects are by default used here, and there is nothing more to add about this actor. All projectile actors have a projectile movement component, which gives a basic projectile behaviour.

6.7. Behavior Trees Creation

To complete the creation of agent controlled AI unit behavior trees need to be implemented. The first behavior tree only has close range attacks available. It consists of main sequence which runs one of the defined queries to find a good location to which he AI then moves to. In figure 99 it can be seen that sequence has a decorator which checks for Blackboard Based Condition, more specifically if the Is Alive boolean is true. After that comes a Selector node which selects only one of the following sequences based on the result of Behavior Tree Service Random Int 5. In figure 100 it can be seen that it also has a decorator that tells the Behavior Tree that after the node has been exited out of, it can not be called for a specified cooldown time amount. All the following sequences have a decorator to define their number for execution when the random number has been picked. One of the tasks is the teleport task so that there is a random chance for AI actor to teleport away from the player character.

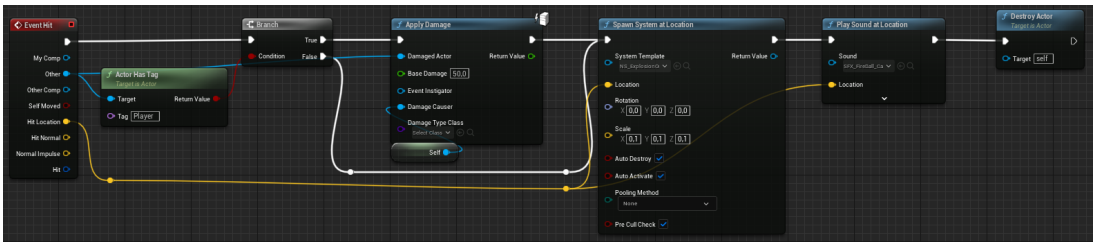

Figure 99: Event Hit of BP_BurstProjectile

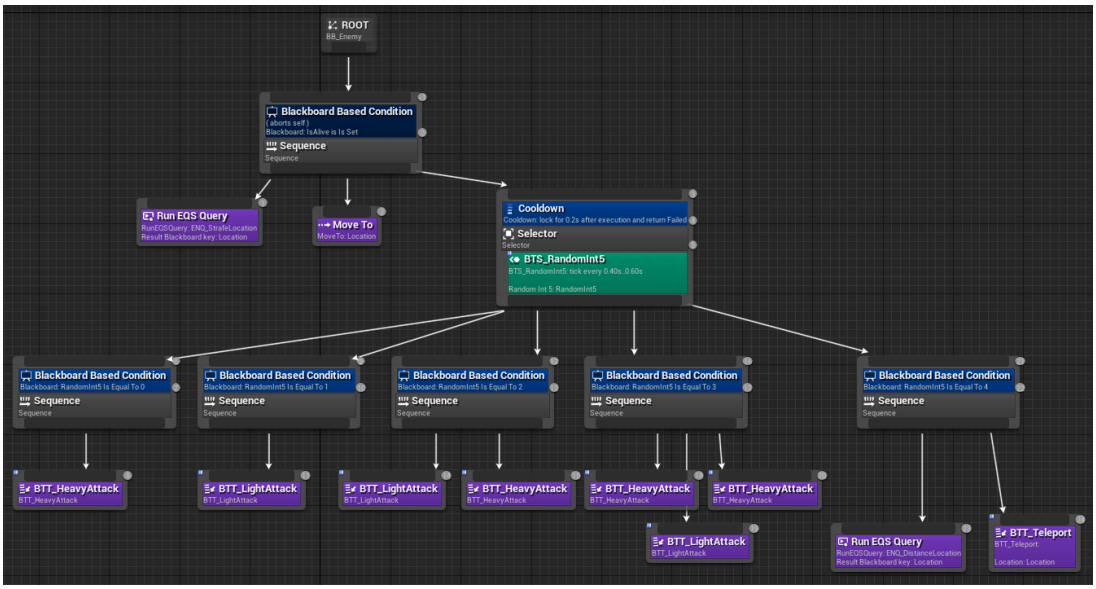

Figure 100: Behavior Tree Enemy Melee

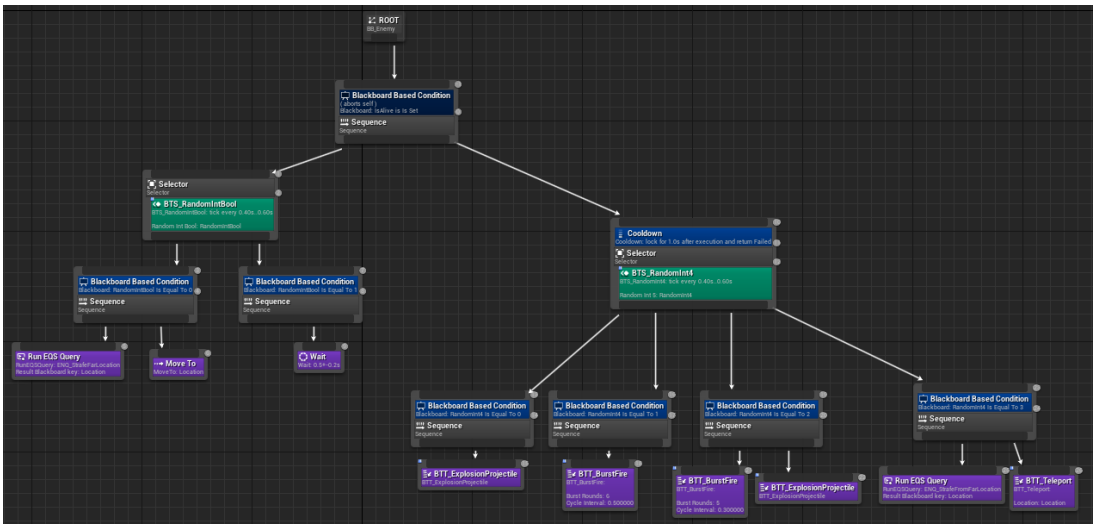

Figure 101: Behavior Tree Enemy Ranged

Ranged behavior tree also has the main sequence which connects to two selector nodes. On one side the AI either moves to a close by location or waits for a random amount of time in range. The second selector node has a cooldown of one second and randomly chooses one of the offered sequences. The main difference between the selector and sequence node is that the sequence runs all the behavior until one of them fails while the selector quits after the first successful node sequence run. Described behavior tree can be seen in figure 101. There is also a teleport sequence, which creates a random chance that AI actor comes close to the player. Other tasks are made out of spawning projectiles.

Last independent tree is the tactical tree. This tree has tasks for healing, stunning and propelling the enemy away from the AI actor. This tree proved to be the hardest to balance. It also implements a service and starts with a sequence. The first part of the sequence is only a Wait task and the second one is a selector which picks a random function out of the set three.

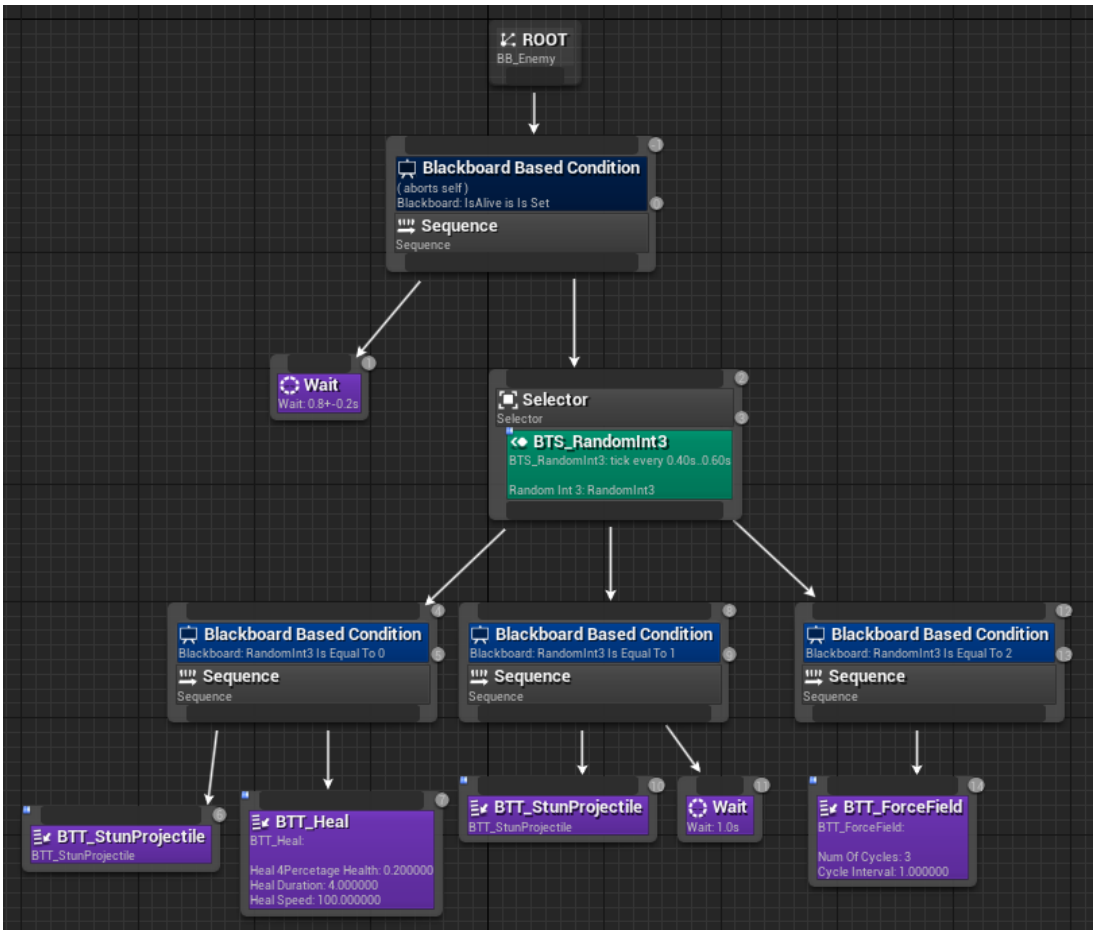

Figure 102: Behavior Tree Enemy Tactical

Described can be seen in figure 102.

All of the described trees are run from one main behavior tree with a starting selector node. Based on the variable Agent In Control it picks the appropriate behavior tree as the figure 103 suggests. All of these nodes have a decorator that aborts all behavior if the AI actor changes status to dead. Other than that, when a new agent is in control, the behavior and available abilities change. Before making any conclusions, a detailed description about all communication that is related to agents is provided in the next chapter.

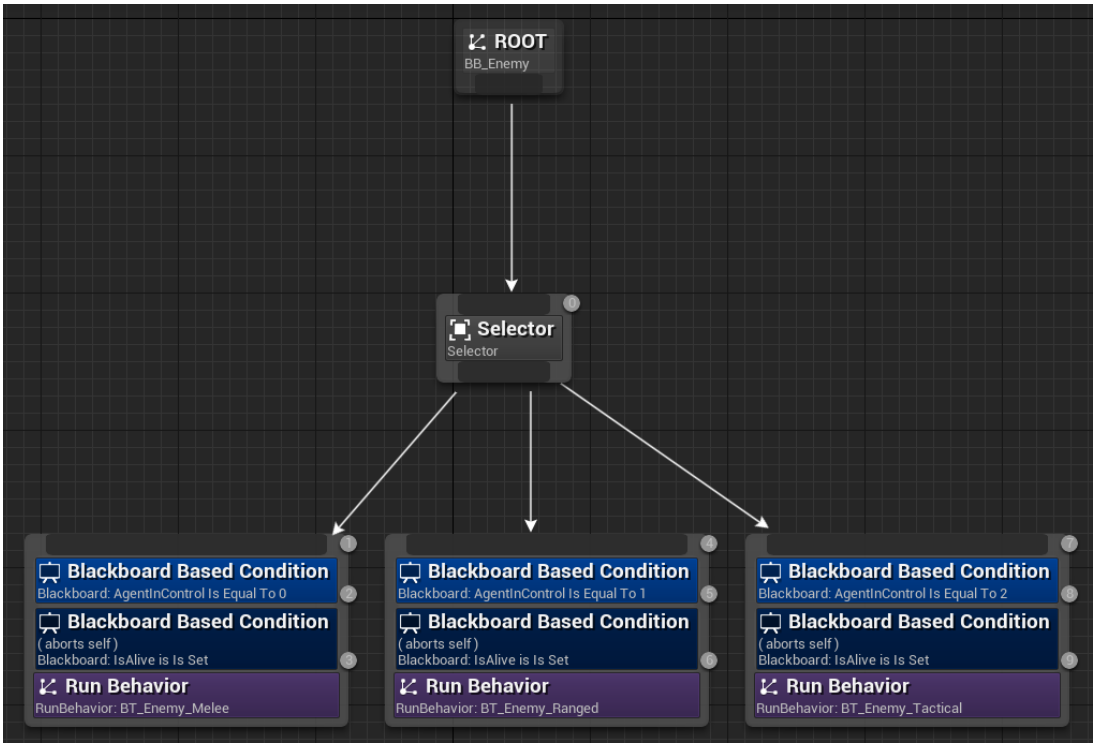

Figure 103: Behavior Tree Enemy

7. Agent Communication System

All communication that is relevant for agents and the functionality they provide is further explained in this chapter. It is easier to follow the explanation with the help of figure 104. Together with all objects that participate in the communication the connection lines are color coded to define the data that is being transferred. Consult the legend below the graph if any uncertainties arise.

Before any decisions can be made an environment data must be collected. Most of the data is collected inside the AI actor named BP_Enemy. Data that is collected here includes distance to the player, current AI HP and damage that is taken in the last three seconds. Some of the data is related to the player and the AI actor is unaware of this information. For that reason, when that data is changed inside of a player actor named BP_ThirdPersonCharacter, the AI actor is messaged accordingly. An AI actor is physically in the virtual world and that is the reason why the data collection process is done inside of this object.

Communication with other objects that are not actors is usually done via Controller objects. Controller of this AI actor is called AIC_Enemy and it can access all data of the AI actor with prior reference of course. This object periodically sends messages to all running agents who bind the event dispatcher EDP_Update Agents. This object also waits for all agents to send a response before evaluating anything. When all agents have sent their bids for AI control, an agent with the highest bid is the one that gains the control over the behavior tree. Agents are also informed about who is the winning agent which alters their bid proposals slightly.

Outgoing communication in AI Controller towards agents is simple since agents know about the AI Controller and his event dispatcher. Incoming communication is a bit different in that regard since the AI Controller does not know who is sending the message back to it. Because of that, besides the bid, agents are sending their IDs. It is important for the AI Controller to know which agent is the winning one since his ID corresponds to the behavior index inside of the behaviour tree. After finding out which agent provided the highest bid, AI Controller communicates with AI Blackboard.

There is no direct communication towards the behavior tree since this object only has algorithms or code and does not store any variables. Tasks inside of a behavior tree can have local variables which can be exposed but those too are only for behavior tree task input purposes. Because of that, the AI Controller needs to communicate with an object called Blackboard. Blackboard is only a storing object that does not have any functionality besides that but is assigned to one or more behavior trees. Values inside blackboard can not have any default values and can only be set from within the behavior tree that they are assigned to or an AI Controller which commands the behavior tree to start executing in the first place. An index of the victorious agent is then set by the AI Controller.

Every task or decorator that reads blackboard as a part of the execution process is altered by the change in value of the dependent variable. In that sense, blackboard needs to communicate to the behavior tree about the agent in control, even though it only reads that variable and does not store it anywhere. Every time a behavior tree requests information about

the agent in control from blackboard is a potential space for switching the behavior branch to the one corresponding to the new agent in control.

Bid calculation inside agents was already explained in code but it is explained again here with an example so that it is easier to comprehend what is actually happening and under which conditions some agents win or lose the bidding process. It is important to keep in mind that these examples do not take into account the increase in bid calculation upon losing the bid.

Values that go into calculating the bid vary in range and even type of the parameter. Some parameters are scalars while some are booleans. This was the first implementation that was found to be fun to play against while also being somewhat easy to adjust. Scalars in that solution are named distance, health points and last damage. They represent what their names suggest. Distance is current distance to player unit, health points is current state of available health points and the last damage represents damage taken in the last defined number of seconds. Boolean values represent the player currently stunned and is attacking. All of those values have a certain acceptable range of values. For boolean those are the only possible values which are either 0 or 1.

If value is not in that specified range, it needs to be clamped. Clamped value goes into the normalisation process where a value in range from 0 to 1 is calculated. Ranges are the same in each agent as well as the current values that are provided to the agents in real time. Differences in agents arise from 3 different main values which determine how they evaluate their value in the bidding process.

Important notice is that one variable is named negativ which should not be mistaken with negative as a sign. The simplest one to explain is the one named scale. This parameter represents the weight of that specific value in the evaluation process. The higher the number the greater the significance in change of that value. This value is also normalised based on the sum of all weights. The next variable is the norm which decides whether a higher value of a certain attribute should positively or negatively influence the end result. Another value is called break and this indicates the breaking point of evaluation in a certain range. Recognized current value of virtual world is evaluated either positively or negatively depending on which side of the breaking point the current value is at. This also influences the variable called negativ which indicates whether a value is to be positively or negatively evaluated based on where it is regarding the breaking point and how norm dictates that value evaluation. For example if value is above breaking point and norm is positive then negativ would be positive as well. If value is below the breaking point and norm is negative then the negativ would still stay positive.

At the end, all of those numbers and indicators combine in a result that is either a product of normalised current value, normalised scalar and negativ or a product of complementary value of the current value, normalised scalar and negativ. These are all individual values for each of observed values and they need to be summed up in order to present a final bid.

One value is presented in all described processes so that this formula becomes more clear. Lets say that distance is defined in range from 400 to 2000. This means that 400 is the minimum possible number value can hold and 2000 is the maximum possible number value can hold. Current distance from player to AI is measured at 1500. This value first goes into

clamping process which does not do much in this case since the value is already in acceptable range. This value is then normalised into the range of 0 to 1 which is 0.6875. Scale is defined as 10 and norm is defined as -1 or negative. This means that greater distance negatively affects the result of this specific calculation. Normalised scale results in 0.526316 which present that distance is highly important for this measurement. Breaking point is defined as 900 and this contributes to negativ value which is negative because the norm is negative and the current value is above the breaking point. Value for that current value would then be calculated as product of (1-0.6875), 0.526316 and -1 which is -0,164473684 and that is one part of the bid. All calculated parts are then summed up into a final bid and results vary between agents since they value certain values differently.

Now it is time to test and see how this AI feels to play against.

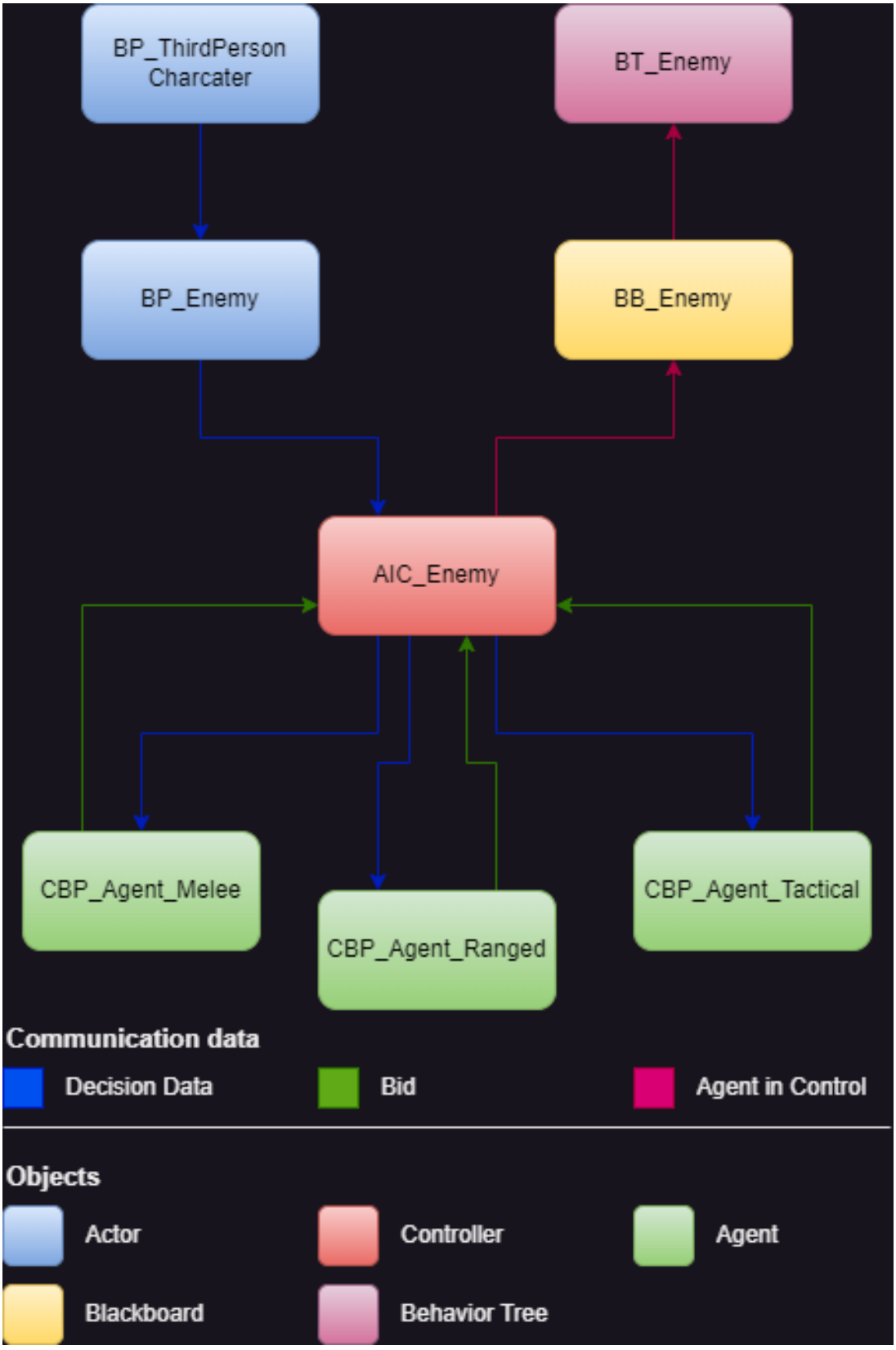

Figure 104: Agent related communication system

8. Conclusion

I am pleasantly surprised by the results of this project. I expected the agent powered behavior tree to be alright in the best case scenario. It ended up being really good and enjoyable to compete against. Things were not looking good at the start of the testing since it is kind of hard to predict how much different values affect the end behavior. For that reason I copied formulas to the MS Excel and started experimenting with values there. It was much more useful since I could see all the temporary results as well as the final result much easier. There I found out that negatives by themselves do not work with normalized scalars and values and I added both the breaking point and the new variable called norm. The formula variable called negativ then became the intermediate variable and everything fell into place after that.

Multiagent system can work in controlling the AI behavior, but I think there is something missing here. Fine tuning the initial variables can be as hard if not harder then creating the formula itself. I think there is more space for exploring the cooperating capabilities before making any final decisions. I shall certainly have in mind the multiagent approach when designing my next AI behavior but I will not be totally reliant on it to make the right decision. As far as I can tell, I think I need to dive deeper into math and algebra and find or create better formulas for calculating bids for agents. That could be a potential improvement space. The other thing that comes to mind is that agents could be better used for behavior tree flow adaptation rather then control. In this approach some different variables could be looked at. For example, the success rates of certain tasks or behaviors and various ways of scoring those rates, which would perhaps make an AI adapt to player combat style. This could make the combat more engaging at low development cost.

This project has been both very entertaining to work on and kind of hard sometimes. I do not think that I needed to push myself too hard, which is nice to observe. It tells me that I was able to overcome the challenges that were set by this project with confidence in my abilities and knowledge. It also makes me think back on the time when I started working in Unreal Engine and how often I would have gotten lost or given in. I am glad that I continued to push though and be able to create projects like this which affirm my capabilities. I am overjoyed by the thought of being able to choose my own idea and constrain it to an area of expertise of my mentor.

Bibliography

[1] S. University, *What is artificial intelligence?* 11/12/2007.

[2] M. Schuster, *Zero-shot translation with google's multilingual neural machine translation system*, 11/22/2016.

[3] A. A. R. T. Lab, *Multi-agent systems*, 2012.

[4] E. M. H. Jiang Y. Zhang, *New technologies for power system operation and analysis*, 2021.

[5] N. G. S. Bringsjord, *Artificial intelligence*, 07/12/2018.

[6] N. J. M. Wooldridge, *Intelligent agents: Theory and practice*, 1995.

[7] P. N. S. Russell, *Artificial intelligence: A modern approach; third edition*, 12/01/2009.

[8] S. Jaiswal, *Types of ai agents*, 2021.

[9] R. J. A. Dorri S.S. Kanhere, *Multi-agent systems: A survey*, 04/30/2018.

[10] J. P. J. Gomez-Sanz, *Methodologies for developing multi-agent systems*, 04/28/2004.

[11] K. L.-B. Y. Shoham, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*, 2009.

List of Figures

Appendices

1. Attachment 1 - Screenshots

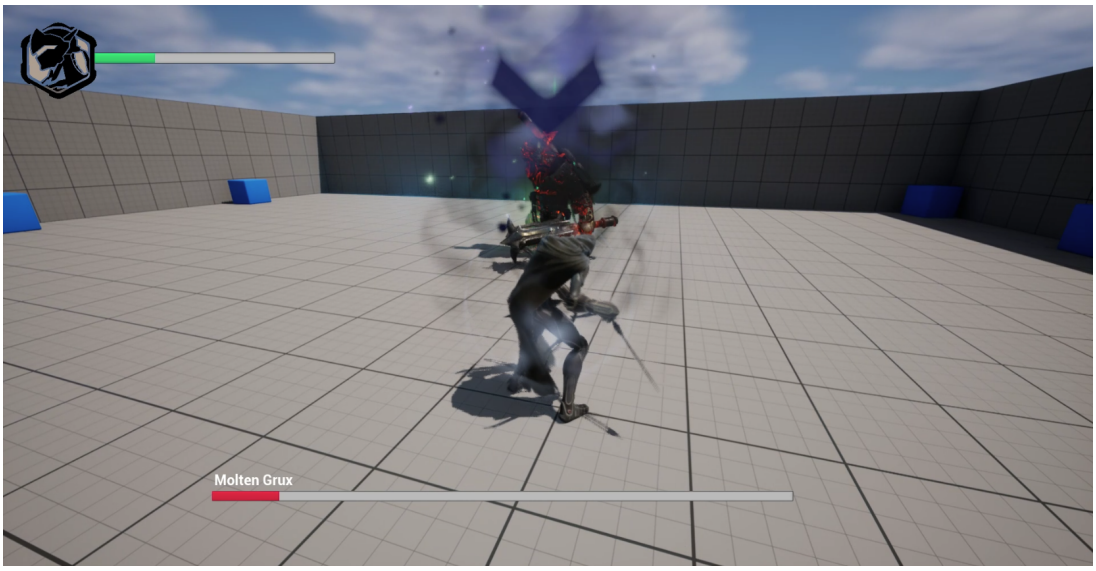

Figure 105: Screenshot 1

Figure 106: Screenshot 2

Figure 107: Screenshot 3

Figure 108: Screenshot 4

Figure 109: Screenshot 5

Figure 110: Screenshot 6

Figure 111: Screenshot 7

2. Attachment 2 - Used Assets

- Player Icon Placeholder - https://www.pngitem.com/middle/homRhm_overwatch-logo-genji-overwatch-genji-logo-hd-png/s
- Player Character Mesh, Animations, Sound Effects - https://marketplace-website-node-launcher-prod.ol.epicgames.com/ue/marketplace/en-US/product/paragon-kallari
- AI Character Mesh, Animations, Sound Effects - https://marketplace-website-node-launcher-prod.ol.epicgames.com/ue/marketplace/en-US/product/paragon-grux
- Visual Effects and Additional Sound Effects - https://marketplace-website-node-launcher-prod.ol.epicgames.com/ue/marketplace/en-US/product/anime-stylized-vfx
- Additional Animations - https://marketplace-website-node-launcher-prod.ol.epicgames.com/ue/marketplace/en-US/product/close-combat-animset

3. Attachment 3 - Example Projects

- example project for describing agent characteristics - https://github.com/TheRealHuzy/LoL-Overlay-System/wiki