VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
DEPARTMENT OF COMPUTATIONAL AND DATA MODELING

Final Bachelor Thesis

Streamlining Character Animation Workflow for Unreal Engine 4

Done by:

Justinas Lekavičius   signature

Supervisor:

lekt. Linas Būtėnas

Vilnius
2021

Contents

Abstract

This final bachelor thesis project is the development of a plugin for the Unreal Engine 4 game engine and an animation blending testing tool. Game engines such as Unity and Unreal Engine 4 are increasing in popularity, becoming simpler for newcomers to develop computer games with. However, developers do not always have access to needed game assets such as character animations, or the means to produce them, and have to spend time looking for assets on the Internet, while in some cases a wide variety of animations is needed. Therefore, the goal of this final bachelor thesis project was to streamline the character animation workflow in Unreal Engine 4. Research was done on how the process of developing animations can be simplified by creating a plugin that downloads animation assets from a server, which are specifically made for the default Unreal Engine 4 Mannequin skeleton, and how animation blending can be used to produce new ones. The results are a developed plugin for downloading Unreal Engine 4 animation assets, a tool for experimenting with character animation blending, and the conclusion that the animation workflow can be streamlined by giving developers easier access to game assets, and by producing new variations of character motions in-engine with a minimal amount of existing assets.

Santrauka

Personažų animacijų kūrimo proceso Unreal Engine 4 žaidimų varikliui supaprastinimas

Šis bakalauro baigiamojo darbo projektas yra įskiepio Unreal Engine 4 žaidimų varikliui bei animacijų maišymo testavimo įrankio kūrimas. Žaidimų varikliai, tokie kaip Unity ir Unreal Engine 4, populiarėja, kadangi naujokams tampa vis lengviau kurti žaidimus. Vis dėlto, žaidimų kūrėjai ne visada turi prieigą prie žaidimų kūrimui reikalingų resursų, tokių kaip personažų animacijos, ar priemonių jas sukurti, todėl reikia praleisti laiko ieškant šių resursų internete, o tam tikrais atvejais reikalinga plati įvairovė animacijų. Taigi, šio bakalauro baigiamojo darbo tikslas – supaprastinti personažų animacijų kūrimo procesą Unreal Engine 4 žaidimų variklyje. Ištirta kaip galima supaprastinti animacijų kūrimo procesą sukuriant įskiepį, kuris atsiunčia animacijų resursus iš serverio, specialiai pritaikytus standartiniam Unreal Engine 4 personažo skeletui, ir kaip, naudojant animacijų maišymą, galima paruošti naujas. Darbo rezultatai yra sukurtas įskiepis Unreal Engine 4 žaidimų variklio animacijų atsiuntimui, įrankis, skirtas eksperimentavimui su personažo animacijų maišymu, bei išvada, jog personažų animacijų kūrimo procesą galima supaprastinti suteikiant kūrėjams lengvesnę prieigą prie reikiamų žaidimų resursų ir sukuriant naujas personažų judesių įvairoves žaidimų variklyje, turint minimalų egzistuojančių animacijų kiekį.

Introduction

Computer game development is becoming increasingly popular and accessible in this day and age, not only due to overall growing interest in game development, but also thanks to easier access to game development tools and growing developer communities. One such engine, the Unreal Engine 4, first publicly launched in 2014 and developed by Epic Games, has become popular in recent years. It is especially accessible to novice developers due to its Blueprint Visual Scripting System which can be used to create game systems without writing a single line of code. It is a game engine widely used by major video game companies producing high-quality and critically acclaimed games such as Star Wars Jedi: Fallen Order, Gears 5, and also Epic Games' own Fortnite. Nevertheless, it is also used by smaller game developer studios to produce games such as Hello Neighbor and ABZU. Unreal Engine 4 is also known for its possibilities of producing photorealistic visuals and can even be used in film making, one example being the produced virtual sets for the TV series The Mandalorian. Unreal Engine 4 also features the Marketplace, a platform that offers developers a selection of free and paid assets, such as music, 2D and 3D assets, animations, textures, and plugins for the game engine. Plugins for the Unreal Engine 4 can provide various new functionality by introducing new gameplay systems that can be integrated into game projects or even provide additional development nodes for the Unreal Engine 4 Blueprint Visual Scripting System. Some plugins on the Marketplace are free and there are also numerous paid ones, however, it is not necessary to use the Marketplace to get plugins, as third-party plugins may also be easily downloaded from the Internet and imported into a game project.

Game developers usually can produce their own assets for their computer games, such as character models, sounds, or textures. However, smaller game developer studios or people just recently introduced to development for Unreal Engine 4 game engine do not always have access to some game assets because of either lack of skills, equipment, or workforce needed to produce the assets. When it comes to character animation assets, developers that do not possess sufficient animating skills, or do not own the hardware or software to produce high-quality animations have to rely on pre-made animation assets, available on either the Unreal Engine 4 Marketplace or provided by other online services. The animation assets are commonly provided as Autodesk Filmbox (FBX) format files, which then can be imported into a game project. However, acquiring the animation assets can be problematic, as in certain instances the animation assets are produced for character skeletons that differ from the Unreal Engine 4 default Mannequin skeleton, thus proving to be difficult to set up or impossible to use with the character skeleton that is provided in a default game project template. Furthermore, problems occur when there is a need for a wide variety of unique character animations, for example, when creating a realistic simulation with hundreds of non-playable characters. In this situation, the characters would need to have unique movement behaviors, and using the same animation set for all characters would have to be avoided. However, it may prove to be time consuming and inconvenient to download large amounts of animation assets from various online services that provide the assets, and even having the hardware and software needed to produce character animations it can be a tedious and expensive process to produce a large number of various character movements. Therefore, animation blending techniques, i.e. combining one or several animations for one character can be used to achieve new character motion variations. Having an interest in working with Unreal Engine 4 and exploring possible solutions for the mentioned issues I decided to develop a solution for simplifying the process of character animation development for Unreal Engine 4.

The main goal of this final bachelor thesis is streamline character animation workflow for Unreal Engine 4 by developing a plugin for Unreal Engine 4 and an animation blending testing tool. The plugin allows game developers to download animation assets for their game project directly from a server, and the animation blending testing tool helps analyse how animation blending can be used to generate new various character behaviors with a limited amount of owned animation assets.

The tasks of this final bachelor thesis are the following:

- Analysis of other systems that offer similar functionality to developed plugin

- Selection of the technologies used for the development

- Design of the plugin (theoretical part of the final bachelor thesis)

- Development of the plugin

- Development of the animation blending testing tool

The result of the final bachelor thesis is a developed plugin for Unreal Engine 4 game engine that usable in game projects and a tool for testing animation blending in the game engine. The result can help discover how the character animation workflow can be streamlined by providing game developers easy and direct access to game assets without having to rely on third-party services, and by using animation blending to produce new character motions on the fly, using the existing animation assets.

1 Analysis of Similar Systems

In this section, the analysis of systems similar to the final bachelor thesis project is performed. The similar systems help solve the problem of obtaining character animation assets for game projects, making overall game development easier. The systems can be inspected and compared in order to make better design decisions for the developed project.

1.1 3D Character Animation Asset Sharing and Editing Systems

When it comes to solving the problem of obtaining character animation assets, there are various services that provide access to them, either free of charge or requiring payment. Some services only have the option to download animation assets, and some provide the ability to edit them before downloading. Therefore, it is evident that such services have their own advantages and disadvantages.

1.1.1 Mixamo

Mixamo is an online 3D character animation system by Adobe that can be used for games and film making. It is available for free and does not require any purchases to download character models or their respective animations. Mixamo offers a wide selection of 3D character models and various high-quality animation assets that made using motion capture. The animations can be applied to different characters and downloaded as an FBX (Autodesk Filmbox) format file, used by 3ds Max, Blender, Unity game engine, and the Unreal Engine 4 game engine. The animations can also be modified using the website user interface before downloading, for example, controlling arm space or height by sliding respective sliders or selecting the needed duration of the animation. The animations can be downloaded individually, or in pre-defined packs, e.g. "Breakdance Pack", which includes character dancing animations, or "Locomotion Pack", which features character movement animations. Furthermore, Mixamo allows the user to upload their own character model and apply any of the animations for that specific model. This is extremely helpful for developers that have their custom made 3D character models with no animations for them. Nevertheless, the uploaded character model has to be processed before can be used with the Mixamo system. The uploaded character model has to be rigged with Mixamo's Auto-Rigger tool, which involves placing markers on the 3D model's chin, wrists, elbows, knees, and groin to generate a new skeleton for the character. The Mixamo animations can then be applied for the newly rigged character. Sadly, some problems with the animations can arise if the markers are set incorrectly during the auto-rigging process, such as model deformations or animation artefacts. Also, Mixamo uses its own skeleton for 3D characters and animations, meaning that the animations cannot be used with the default Unreal Engine 4 Mannequin skeleton out-of-the-box. In that case, the user would have to rely on the Unreal Engine 4 Animation Retargeting system, to retarget animations from the Mixamo skeleton to the Unreal Engine 4 default skeleton.

Mixamo's main advantages are the offering of animation assets free of charge, the ability to modify the animations before actually downloading them, and the option of applying the animations to an uploaded character model. Despite that, its lack of straightforward support for the Unreal Engine 4 Mannequin skeleton and possible problems during the rigging of an uploaded character may be a nuisance for some and a big disadvantage for others.

1.1.2 MoCap Online

MoCap Online is an online service that provides 3D character animations. Users that visit the Mo-Cap Online website may browse a vast selection of various character animations, professionally created using motion capture. The animations are offered as bundles, similar to Mixamo's animation packs. The animations can be downloaded in FBX file format, therefore they are compatible with Unreal Engine 4. MoCap Online offers animation packs specifically made for certain game engines such as Unity or Unreal Engine 4. The animation packs for Unreal Engine 4 are delivered as FBX files and are made for the default Unreal Engine 4 Mannequin skeleton. Therefore, the animations can be easily used with the default Unreal Engine 4 skeleton in a game project with little to no additional tweaking needed. In terms of convenience, MoCap Online has an advantage. However, animations are only offered as packs and there is no possibility to download individual animations. Furthermore, all of the Unreal Engine 4 character animation packs are paid, at least at the time of writing. Interestingly, MoCap Online also offers some animation packs on the Unreal Engine Marketplace. Only one animation pack "MCO Mocap Basics" by MoCap Online is available on Unreal Engine Marketplace and is free of charge, containing 20 various idle and movement animations.

1.1.3 Unreal Engine Marketplace

The Unreal Engine Marketplace is a platform for buying and selling game assets such as textures, music, characters, and animations. Assets on the Unreal Engine Marketplace can be purchased or downloaded free of charge, depending on the asset. Once purchased or downloaded, the assets can then be added to a game project via Epic Games Launcher's Vault section. Unreal Engine Marketplace is one of the most convenient ways for game developers to get animation assets for their game projects, as the Marketplace is integrated into Unreal Engine 4, allowing quick access to downloaded assets. Furthermore, most animation packs on the Unreal Engine Marketplace are made with the default Mannequin skeleton in mind. The overwhelming majority of animation assets on the Unreal Engine Marketplace is also paid, with only some animation packs being free. One of them is the already mentioned MoCap Online's "MCO Mocap Basics" pack and Mixamo's own "Mixamo Animation Pack". Some of the animation assets that can be downloaded on both Mixamo and MoCap Online systems are also available on the Unreal Engine Marketplace.

1.2 Comparison of Similar Systems

System name	Are the animations free?	Are the animations compatible with the UE4 default skeleton?	Can the animations be downloaded individually?
Mixamo	Free	No	Yes
MoCap Online	Paid	Yes	No
Unreal Engine Marketplace	Some paid, some free	Yes	No

Table 1. A comparison of Mixamo, MoCap Online and Unreal Engine Marketplace.

The analysed systems all provide the same functionality, i.e. give users the option to download animation assets and use them for their game projects, however, the systems have some key differences. Mixamo, in contrast to MoCap Online and Unreal Engine Marketplace, offers all of its animation assets free of charge and allows users to download them individually, or in animation packs. Mixamo is also the only system out of all three that offers the option to modify the animation assets before downloading them. However, the animations are not immediately ready to use with the Unreal Engine 4 default skeleton, therefore they would have to be retargeted to the different Unreal Engine 4 Mannequin skeleton. Another alternative would be to upload the 3D model of the Unreal Engine 4 Mannequin character, use the auto-rigger function to map a new skeleton to the character model and then select needed animations for download. However, the uploaded character would have the Mixamo skeleton applied, and the animations made for the Unreal Engine 4 Mannequin skeleton would be unusable without retargeting. Therefore, neither approach is convenient. MoCap Online and Unreal Engine Marketplace offer animation packs that are available for the Unreal Engine 4 Mannequin character out-of-the-box, however, the animations are only offered in bundles. Also, MoCap Online animation packs are paid (excluding MoCap Online's own "MCO Mocap Basics" pack which is available on the Unreal Engine Marketplace), and Unreal Engine Marketplace animations are mostly all paid, with the exception being the already mentioned MoCap Online and Mixamo animation packs that are available for free.

1.3   Conclusion

The three analysed systems have some differences in terms of convenience and availability of animation assets, however, they all have one common goal, which is to provide game developers and learners the assets that they would not be able to make themselves, for their game development, learning, and other purposes. Analysing the systems served a key role in making better choices for the design of the developed plugin. Taking into account the convenience and functionality of the analysed systems, functional and non-functional requirements were set, and used technologies were chosen in an attempt to improve on their disadvantages and also take an example from the advantages. Therefore, the decision was made to develop a plugin that would be integrated into a game project as seamlessly as possible. For convenience, all of the animation assets have been produced specifically for the Unreal Engine 4 Mannequin skeleton, using a motion capture solution for the best possible quality.

2   Research of Unreal Engine 4 Game Engine

The following section provides information that has been gathered while researching Unreal Engine 4, essential to the process of developing the final bachelor thesis project. That includes the analysis of the Unreal Engine 4 game engine modules such as the plugin system and Blueprints Visual Scripting system, as well as the analysis of animation file formats supported by Unreal Engine 4 and the overall process of developing animation for an Unreal Engine 4 game project. The analysis was performed with the intention to explore the advantages and disadvantages of Unreal Engine 4, and how the former and latter can be taken into account when developing the project.

2.1 Unreal Engine 4 Blueprints Visual Scripting System

The Unreal Engine 4 game engine features the Blueprints Visual Scripting system that is based on the concept of building Blueprints made up of interconnected nodes. This system allows game developers to create game scripts without actually writing a single line of code. This system is appealing due to its visual representation of game scripting and providing the ability for game developers to create various systems without necessarily having C++ or other programming skills. There are various types of Blueprints, including but not limited to Level Blueprint, Animation Blueprints, Character Blueprints, etc. Even though the Blueprint system does not require writing code, the system can be expanded by developing new nodes that can be used along with other nodes to build more complex systems. The nodes can be written in C++, compiled, and then used in Blueprints, along with other already existing systems. Some plugins found on the Unreal Engine Marketplace or online, when enabled offer various new nodes that can be added to Blueprints and connected to make new systems.

Even though the Blueprints Visual Scripting system is prominent in Unreal Engine 4, it is not necessarily the only way to write game logic. C++ classes can be used alongside the Blueprint system, and in some cases, C++ classes even have advantages over Blueprints, such as faster runtime performance and more explicit design. However, the Blueprint system is also very flexible and has such advantages as more flexible editing and faster access [1]. It is argued whether the better practice to write game logic is purely with C++ or by using Blueprints. In certain scenarios, a case is made against heavily relying on C++ programming, as it is noted that Unreal Engine 4 documentation is not as well written as, for example, Unity game engine documentation, therefore raising difficulties in working with the Unreal Engine C++ framework, not only due to lack of documentation but also due to overall steep learning curve [2] [3]. However, a developed Blueprint may be hard to read if the amount of connected nodes is considerably large and if the nodes are connected in a disorderly fashion. Therefore, because the Blueprint system was created with the intention of easier prototyping and better visualisation, and because C++ provides the ability to implement custom code for the game engine, a combination of both C++ and Blueprints Visual Scripting system ought to be considered when creating game systems, without relying too heavily on the former or the latter.

2.2 Developing Plugins for Unreal Engine 4

The Unreal Engine 4 game engine features a plugin system that allows users to enable or disable various modules that add additional functionality to developed games or the engine itself [4]. The plugins that provide new functionality are written in C++. Unreal Engine 4 has a variety of built-in plugins that can be enabled or disabled on demand, and additional plugins can be downloaded from the Unreal Engine Marketplace or elsewhere online. Plugins can either be of Engine or Game type. The main difference is that the Engine type plugins are located in the Unreal Engine 4 installation directory, under "Engine/Plugins/", while Game type plugins are located in the game project directory's "Plugins" folder. Therefore, Game plugins are exclusive to the game projects they are copied to, while Engine plugins can be used throughout all created game projects, without having to copy the plugin files to each game project separately. Plugins also do not necessarily contain code, rather containing additional game content such as game models, sounds, textures, etc. Such plugins have the game assets stored in the Content folder, and the plugins with C++ code have a

Source folder that contains the source code for the plugin.

The plugins usually contain the "Binaries", "Config", "Content", "Intermediate", "Resources" and "Source" folders, as well as the Plugin Descriptor which is a .uplugin format file. The Plugin Descriptor contains such information as version name, plugin title and description, category, etc. The Binaries folder contains the compiled plugin code and the Intermediate folder contains temporary build product files [4]. The plugin code can only be compiled when a C++ game project is compiled. When creating a new game project the user may select to create either a C++ or a Blueprint project. The main difference is that a C++ game project adds a "Compile" button to the Unreal Editor that allows compiling C++ code in the game project, while also allowing the user to build various Blueprint systems, whereas a Blueprint project omits the ability to compile project C++ source code and the user is only able to use the Blueprints Visual Scripting system for building any game systems. The Binaries folder can be deleted, for example, to save storage space when needed, however in that case the plugin would have to be recompiled. The Config folder contains a single file FilterPlugin.ini, which is created by default and is entirely optional, specifying which additional files ought to be packaged with the plugin. The Content folder contains additional content of the plugin (such as models, sounds, textures, etc.). The Resources folder usually contains the single file Icon128.png, which is simply an image that is displayed next to the plugin details in the Unreal Editor's Plugins section. The Source folder, as the name implies contains the source code for the plugin in .cpp and .h C++ classes, in "Private" and "Public" sub-folders accordingly.

The plugins in Unreal Engine 4 have the advantage of being easily reusable – the installation is simple, consisting of simply moving the plugin folder either into the appropriate game engine directory, or the game project folder. However, a big disadvantage of the plugins in Unreal Engine 4 is the lack of out-of-the-box backward-compatibility or forward-compatibility. This means that a plugin developed in one version of the Unreal Engine is not compatible with newer or older versions of the game engine, e.g. a plugin developed for Unreal Engine 4.26 is not compatible with Unreal Engine version 4.25.4, and vice-versa. The only solution to this problem is to manually recompile the plugin code for the used Unreal Engine version, via the Unreal Editor. Otherwise, if the plugin was downloaded via Unreal Engine Marketplace, the user may wait for the plugin creator to publish an update of the plugin that makes it compatible with a newer version of the engine.

2.3   Animation Blending in Unreal Engine 4

Animation blending can be defined as the process of making a smooth transition between two or more animations on one character that are played in succession [5]. The intent is to eliminate artefacts in character motion when the animation is changed depending on the situation, i.e. from walking to running animation and display a "crossfade" that appears natural. This may also be used for more advanced systems such as character locomotion when it is necessary to realistically simulate character foot placement on uneven terrains in dynamic environments [6]. It can also be defined as generating new character motions by using one or mode additional animations in additive layers that have appropriate blend weights set to them [7]. This helps to create new character motions during runtime without the need to create brand new animations for each possible scenario, which is likely to be extremely difficult or even impossible. For example, applying an animation of a character waving their hand to an in-game character that is walking on a path. In this

case it is only needed to apply the wave animation as an additive layer to the upper part of the character skeleton, or at the very least the right or left arm (depending on which arm is waving), and by applying a different blend weight to the additive layer the animation can either be more or less intensive. For example, having a blend weight of 1.0 would play the waving animation for the top part of the body normally, and 0.5 blend weight would display the waving motion less prominently. It should be noted that in this case the animation blending is used with already existing animation assets, and should not be confused with procedural animation. Procedural animation may be used to generate new animations using, for example, trigonometric expressions, to create dynamic animations without using example motion capture or keyframe data [8], or enhance character skeleton interaction with the environment during runtime, using inverse kinematics (to ensure that feet step correctly on the ground) [5].

Animation blending in Unreal Engine 4 is accomplished in the character's Animation Blueprint AnimGraph. The AnimGraph is used to manipulate or blend character poses for a Skeletal Mesh into the final output pose, i.e. the character animation that is displayed in-game [9]. Using the "Layered blend per bone" node, two poses can be blended into one. The first input pin for Base Pose, for example, can be the Default Animation State Machine, i.e. the collection of all animation States and Transitions, and the second one (Blend Poses 0) can be the animation that we want to apply. The Blend Weights 0 variable of float type determines the weight of the layer, i.e. the Blend Pose, and ranges from 0.0 to 1.0. A value of 0.0 gives full weighting to the Base Pose (no blending applied), and a value of 1.0 gives full weighting to the Blend Pose 0, i.e. overriding the original animation. If the Blend Weight is set to, for example, 0.6, 40% of the original animation will be retained and 60% of the Blend Pose will be used. "Layered blend per bone" node can be used to apply blending to the whole skeleton, or certain parts of it. For example, an animation can be added only to the upper part of the skeleton, e.g. an animation of carrying an object. "Blend Multi" node may also be used to blend multiple poses into one animation, using a single node. This blueprint node can only be used to blend the animations for the entire skeleton, and cannot be used for specific parts of the skeleton. With this node, many animations can be blended into one, and each pose can have its own desired alpha, i.e. the weighting of the pose, ranging from 0.0 to 1.0. The percentage of each animation used for the final output pose depends on the number of animation poses used, and their alphas, with their sum being 100% (the output animation). Animation blending can also be used, for example, only during certain scenarios for transitioning from one pose to another. For that, the "Blend poses by bool" blueprint node can be used. When the Active Value (variable of bool type) is true, the pose that is connected to the True Pose will be played, and vice-versa. This can be used to transition from one pose to another, for example, from one Animation State Machine to another State Machine. An example use case: if the "isCharacterHurt" variable of bool type has a "true" value, the animation set for character acting like they were hurt would be used, otherwise, the standard animation set would be used. The True Blend Time and False Blend Time define the amount of time needed to blend into true and false poses respectively.

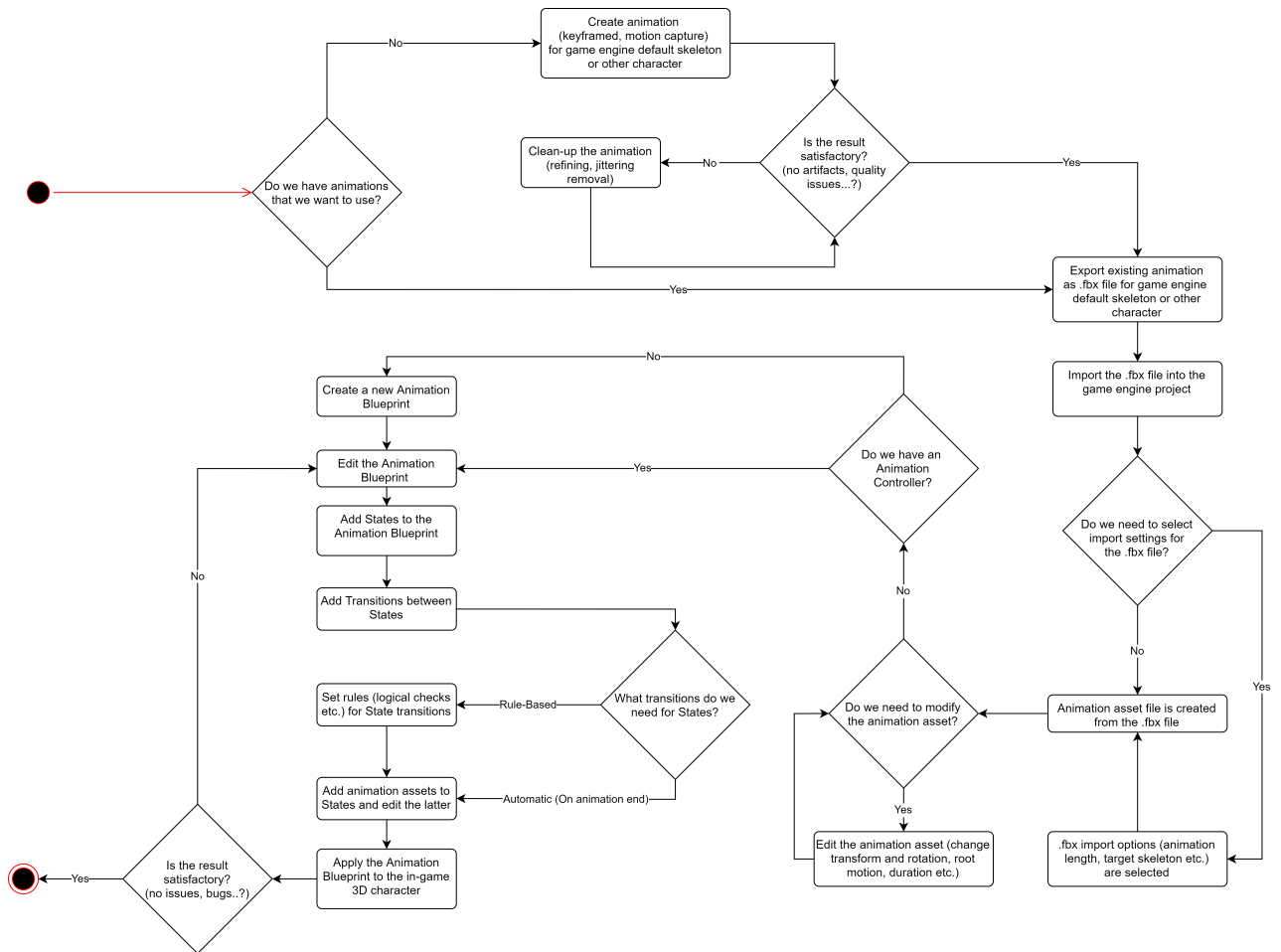## 2.4 Workflow of Unreal Engine 4 Character Animation Development

Figure 1. Diagram representing the workflow of developing animations for an Unreal Engine 4 game project, applying them to an in-game character.

In order to determine the best possible approach to making the process of using animations for game projects as simple as possible, it is important to examine the workflow of the whole process, from actually developing the animation to finally using it for an in-game 3D character. The Unreal Engine 4 built-in animation system can be used for the majority of the character animation development process, especially if the needed animation assets already exist and the production of new ones is not needed [10]. The workflow diagram was drawn taking into account the likely procedure for transferring animation to Unreal Engine 4 game engine and based on Unreal Engine 4 documentation, as well as own trial and error. For the sake of simplicity, an assumption is made that a 3D character model is already available, therefore the examination of 3D character model sculpting, rigging, weight painting, etc. process is omitted.

If there is existing animation that can be used, it can be exported as an FBX file for the game engine default character skeleton, or other character skeletons. Otherwise, the starting point is preparing the animation itself, either using specific animation software (such as Autodesk Maya, MotionBuilder, etc.) or using motion capture solutions. Once the animation is done, its quality can be assessed. If the quality is unsatisfactory or there are other artifacts, the animation can be cleaned up by refining and removing jittering, i.e. involuntary shaking of the character. Once the

animation is of acceptable quality, it can finally be exported. The animation file is then imported into the Unreal Engine 4 game engine project. During the importing process, there may be some options that need to be set before the animation is imported, such as the target skeleton, animation length, etc. Afterwards the animation is imported into the project by creating a separate asset file for the animation by using the original FBX file as a reference. The original FBX file is not directly modified or used by the engine, only the newly made UASSET file. UASSET file is a proprietary file format used by Unreal Engine 4 for its game assets. The newly created animation asset can then be modified (enable looping, mirroring, root motion, etc.) until it is suitable for further use. Then the animation asset is then ready to be used with an Animation Blueprint.

The Animation Blueprint can be created if it does not exist, otherwise, it can be edited to include various States, State Machines, Transitions between States, as well as setting rules for the Transitions. For the States to be useful, animation assets need to be added to them. Once we have our Animation Blueprint set up, it can be applied to the in-game character. When starting the simulation of the game, we can examine if everything is to our liking. If there are any problems with the animation (for example, the animations transition too soon or don't transition at all, or there are any other technical issues), we can go back to the Animation Blueprint and make additional adjustments. Alternatively, if everything is satisfactory the animation transferring to the game engine is complete. It is important to note that this workflow of transferring animation for an in-game character is not exclusive to Unreal Engine 4. It could also be applied to other game engines such as Unity. The workflows of importing animation for Unreal Engine 4 and Unity are very similar, with only minor differences in terminology and some functionality. For instance, in the Unity game engine, Animation Controllers act very similarly to Animation Blueprints of Unreal Engine 4, with the former also having States and Transitions between them. Moreover, just like Unreal Engine 4, the Unity game engine does not directly use the original FBX file for animation. Rather, it creates a copy of the FBX animation file with a metafile that contains the information of changes to the imported animation.

Looking at the diagram it is possible to determine which steps can be simplified or overall omitted when developing the plugin. Normally the starting point of the whole process would be the question if there are animations ready for use with the game project. Having a plugin that downloads animation assets directly to the game project eliminates this question, as there is no need to create animations (using either motion capture or keyframes) or export existing animations to FBX format. Furthermore, the downloaded animation assets are in UASSET format, i.e. animation assets are already created from their respective FBX animation files, thus there is no need for the user to import the FBX animation files, select the Skeleton Asset for each respective animation and then confirm their selection. That is because every downloaded UASSET file is already applied to the Unreal Engine 4 default Mannequin skeleton and ready to use, consequently no additional setup is needed unless the user wishes to modify the animation assets. The downside to using UASSET files, however, is that they are not backward-compatible, meaning that they may not work with a game project that was created with an older version of the Unreal Engine. The usage possibilities of animation assets are numerous, either using them individually or as a collection of States in Animation Blueprints, and that would be the choice of the game developer. Hence, the developed plugin can help simplify a significant part of the workflow of character animation transfer to the Unreal Engine 4 game project.

3 Theoretical Model of the Developed Plugin

In this section, the theoretical model of the developed plugin will be described. That includes the system requirements for the developed plugin, the configuration, the performed activities, and the possible use cases.

3.1 System Requirements

The functional and non-functional system requirements of the developed plugin are presented down below.

Functional requirements:

- Downloading character animations from a server

- Fetch animations based on categories selected by the user

- Download new assets whenever they are added to the server

Non-functional requirements:

- Simple to use the plugin

- As less additional needed input from the user as possible

The plugin is made for Unreal Engine version 4.26 (the latest version at the time of writing), and it requires version 4.26 to work. The plugin will not work with older versions of the game engine, such as 4.25. The recommended hardware for the Unreal Engine 4 is a computer running on Windows 10 64-bit operating system, with 8 gigabytes of RAM, quad-core Intel or AMD, 2.5 GHz or faster CPU, and a video card compatible with DirectX 11 or 12. For developing with the engine (writing and compiling C++ code) Visual Studio 2017 or 2019 is needed, depending on the version of the engine. For Unreal Engine version 4.26 specifically, Visual Studio 2019 is required to compile C++ code. Unfortunately, there may be issues with Unreal Engine 4 detecting an installation of Visual Studio, even though a compatible version is installed. In such a case, Unreal Engine 4 has to be launched with administrator rights, using the "Run as Administrator" option. To download the animation assets from the server to a game project, an Internet connection is needed.
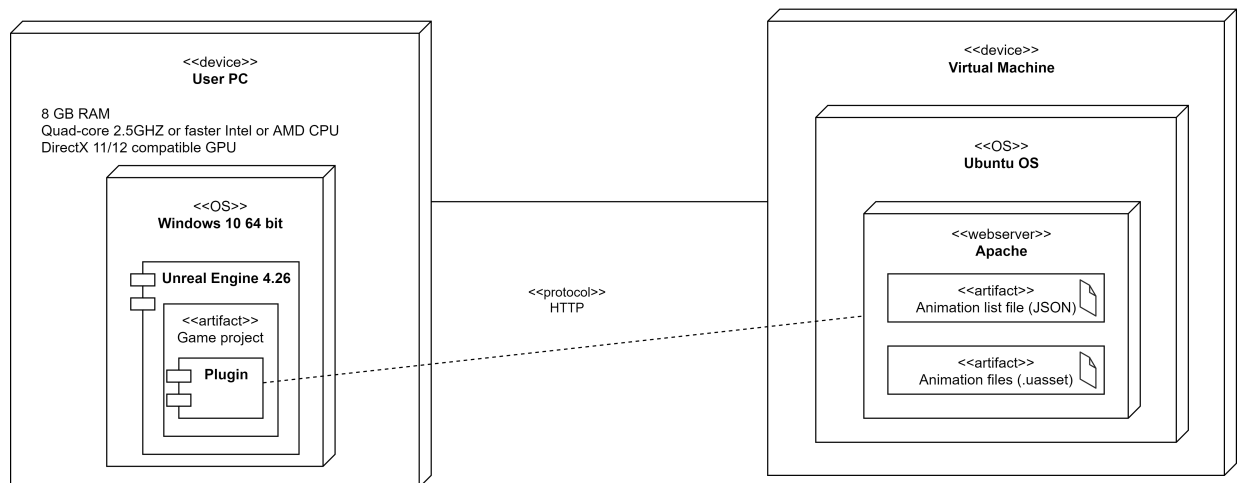
3.2 UML Deployment Diagram

Figure 2. UML Deployment diagram of the developed plugin.

The configuration of the developed plugin is represented in the UML Deployment diagram. An assumption is made that the developed plugin is installed as a Game plugin, i.e. added to a single Unreal Engine 4 game project. On the left is the plugin user PC and on the right is the virtual machine that acts as a means of storage for animation assets, as well as providing them to the client machine using a webserver. Since the recommended PC specifications for Unreal Engine 4 are Windows 10 64-bit with 8 GB of RAM, quad-core Intel or AMD, 2.5 GHz or faster CPU, and a video card compatible with DirectX 11 or 12, an assumption is made that the user's PC is of the mentioned specifications, along with an Internet connection which is needed to download animation assets from a server. Furthermore, Unreal Engine version 4.26 has to be installed on the user PC, as the plugin is only compatible with Unreal Engine 4.26. A virtual machine running Ubuntu operating system can be used to run the Apache webserver that can be connected to by HTTP protocol. To be able to get responses from the virtual machine using a public IP, port 80 has to be opened. The HTTP protocol is used by plugin code to send GET requests to the webserver. Mainly two types of GET requests are sent – for fetching the contents of the animation list JSON file and for fetching a single animation asset file. The animation list JSON file contains information for an array of objects that represent the animation files stored on the virtual machine and contains info such as file name, animation category, and file type. The file acts as the list of animation assets that are available to download, therefore with each newly added animation asset to the virtual machine, the animation list JSON file also needs to be appended to reflect the new additions.
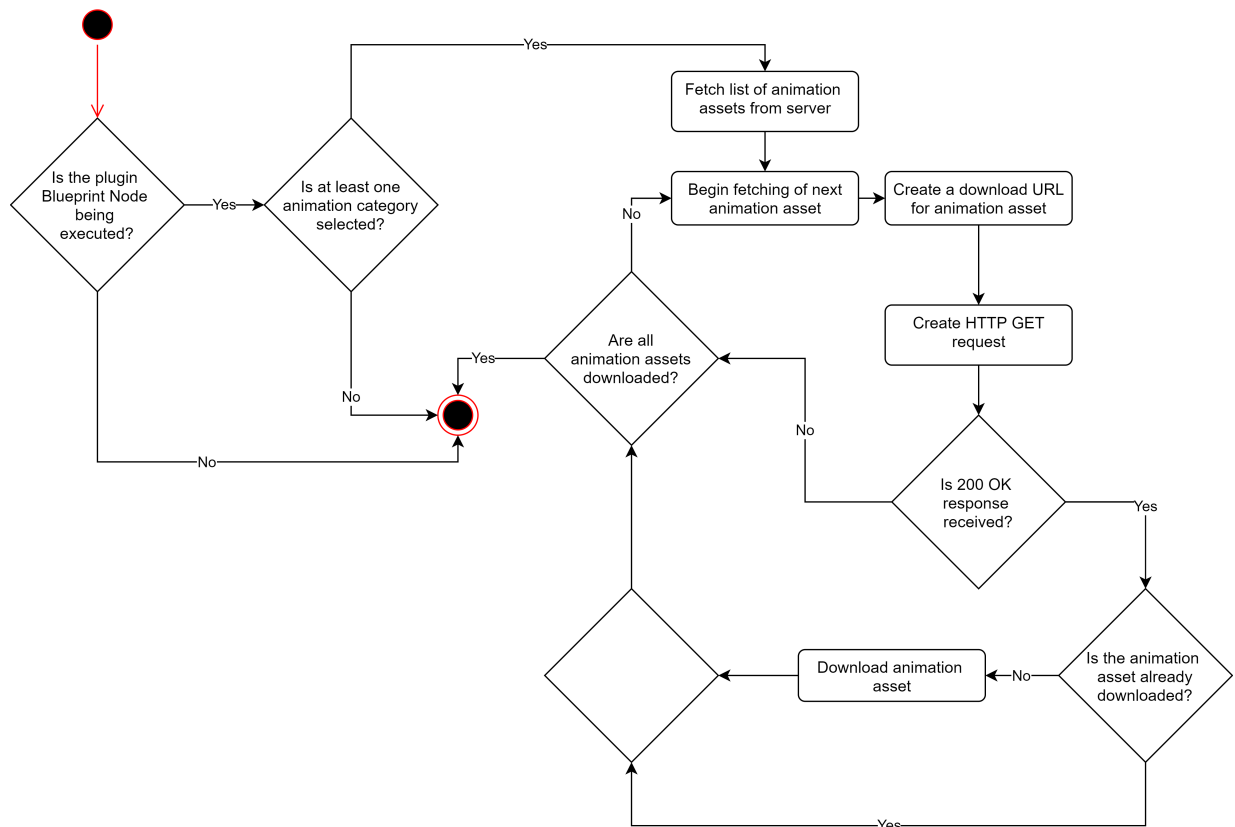
3.3 UML Activity Diagram

Figure 3. UML Use Case diagram of the developed plugin.

The UML activity diagram represents the activities that the developed plugin performs. The plugin functionality is executed whenever the plugin Blueprint Node is added to a Blueprint and is connected to another active node by the Execution Pin. There are numerous possibilities of Blueprint construction and various ways to execute nodes, therefore for the sake of simplicity, an assumption will be made that the plugin Blueprint Node is added to the Level Blueprint and connected to the Event BeginPlay node. In this case, the node is executed every time the Play button is pressed in the Unreal Engine 4 Editor. If the plugin Blueprint Node is being executed and if at least one animation category checkbox is checked for the Blueprint Node, the process of downloading animation assets is initiated. Otherwise, nothing happens. At first, the list of animation assets is fetched from the server. This list in this case is an array of objects that represent the animation assets on the server. The objects have their attributes such as the file name, animation category, and file type. Once that is done, a for loop is run, iterating through the list of animation assets. For each animation asset, a download URL is created. The download URL is built by using the animation asset's category, file name, and file type. For example, if an animation asset called Wave, belonging to the Gestures category and of "uasset" file type (the file type for Unreal Engine 4 game assets), the download link would be http://<virtual machine public IP>/<main folder>/Gestures/Wave.uasset. Then, an HTTP GET request is created for the download URL. If an OK response (code 200) is received, the animation file is accessible and can be downloaded. Otherwise, it is possible that the animation asset described in the animation list does not exist (has been moved, removed or renamed). In that case, the next animation asset is fetched. If the animation asset already exists in the game project, it is not downloaded again. Otherwise, the animation asset is downloaded to the game project.

If all animation assets are downloaded, the plugin functionality is done. If not, other remaining animation assets are downloaded, following the same procedure.
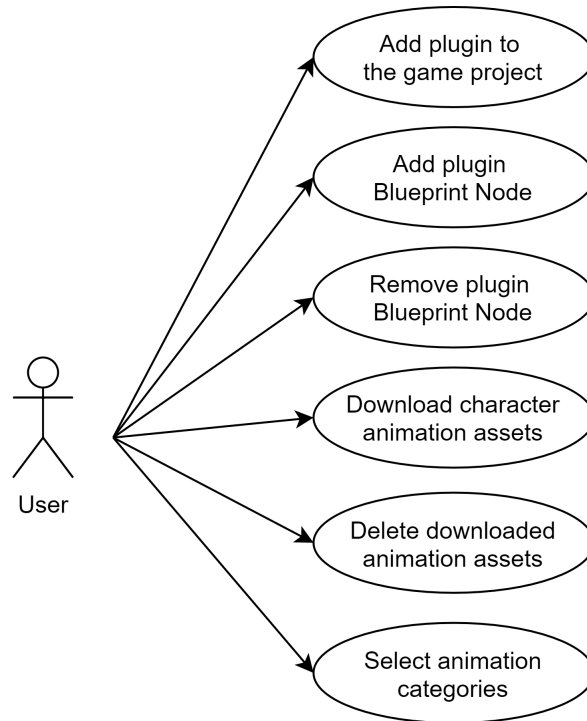
3.4   UML Use Cases Diagram

Figure 4. UML Use Case diagram of the developed plugin.

The functions that the user can perform when using the plugin are displayed in the UML use cases diagram. The user can start using the plugin by adding it to their game project. That can be done by copying the Plugins folder to the game project root folder. The plugin can also be copied to the Unreal Engine 4 installation directory's Plugins folder and used across all existing game projects. The plugin ought to be automatically enabled, otherwise, the user may have to navigate to Edit -> Plugins in Unreal Engine 4 Editor, find the plugin named Download Animation Assets, and check the Enabled checkbox if not already checked. Once that is done, the plugin can be used. The plugin's functionality is represented as a Blueprint Node that can be added to the Level Blueprint or any other Blueprint, however, using the Level Blueprint is recommended. The plugin node can also be removed from the Blueprint when not needed, or simply disconnected and isolated from other nodes. Once the node is connected by the Execution Pin to another node, such as, for example, Event BeginPlay, the user can press the Play button in the Unreal Engine 4 Editor and the plugin will instantly start working, downloading the character animation assets. The assets can then be used individually or added to the character Animation Blueprint, based on the user needs. The animation assets can be deleted from the project whenever they are not needed and downloaded again on demand. If the user wants to download only some animations, they can select an appropriate category of animations by checking a box in the plugin Blueprint Node. Depending on which checkboxes are checked only some, all, or none of the animations will be downloaded.

4 Implementation of the Developed Plugin

In this section, the practical implementation of the developed plugin will be described in detail. It is made up of selecting suitable hardware and software for development and ultimately, the development process of the plugin.

4.1 Used Hardware and Software

This subsection contains the details of used hardware and software for development. Some information about the tools is provided, along with arguments for why the specific hardware and software solutions were selected.

4.1.1 OpenNebula MIF Cloud Service

The OpenNebula MIF Cloud service is used for creating a virtual machine on the VU MIF network. There are many templates for creating virtual machines, each differing in the selected operating system, storage, amount of RAM, etc. In this case, a virtual machine is created using the "ubuntu-20.04" template, with 1GB of RAM and 5GB of storage. Normally the default amount of storage is set to 1GB, however, due to the possibility of storing a large number of game asset files, more storage was selected. As the template name suggests, the virtual machine is running on the Ubuntu operating system version 20.04. This operating system was selected simply because of familiarity with it. Port 80 has been opened so that the virtual machine could be accessed using the external, i.e. public IP.

4.1.2 Unreal Engine 4

The game engine chosen for the development of the plugin and creation of the animation blending testing tool is Unreal Engine 4, developed by Epic Games. Originally, during the course of writing this final bachelor thesis, Unreal Engine version 4.25.4 was used. However, version 4.26 was released on December 3rd, 2020 and it was decided to switch to the newer version of the engine. The new version did not introduce any new functionalities that could aid the development of the plugin, however. One of the reasons for the switch was made the desire not to use the older version of the engine. Version 4.25.4 is a hotfix release for version 4.25, whereas the newly released version 4.26 is a major release. The other reason is to avoid possible difficulties that could arise when transferring the plugin from an older version of the engine to a newer one. The game engine is also used to prepare the character animation assets by importing FBX files into a game project, applying the Unreal Engine 4 default Mannequin skeleton for each one, and then exporting the generated UASSET files for storage on the virtual machine.

4.1.3 Microsoft Kinect

Microsoft Kinect is a game motion controller, originally developed by Microsoft for the XBOX 360 game console and released in 2010. Although the controller is intended for use with video games that support motion controls on the XBOX 360 game console, it can also be used with a personal computer running Windows 10, having Microsoft's own Kinect for Windows SDK installed. In this case, Kinect is used to make body motion capture animations. The main reason for using this peripheral is the potential to create quality character animations easily and quickly, thus being able

to have a collection of self-made animation assets used for the final bachelor thesis project without relying on any other external services. Although using a motion controller designed for video games may not be as practical as using motion capture systems specifically designed for producing quality character animations, its affordability and usability are still enough for the production of quality game assets.

4.1.4 Microsoft Visual Studio 2019

Microsoft Visual Studio 2019 is an integrated development environment that can be used to write code in C#, C++, Java, and other programming languages. This software was chosen because it is specifically required by Unreal Engine 4 to edit and compile C++ code. Specifically, Visual Studio 2019 is required when compiling Unreal Engine 4.26 C++ code. The required version of Visual Studio varies by Unreal Engine version. For example, Unreal Engine version 4.20.3 requires Visual Studio 2017 to be installed.

4.1.5 iPi Mocap Studio and iPi Recorder

iPi Mocap Studio and iPi Recorder are a combination of motion capture and video capture programs, developed by iPi Soft LLC. iPi Recorder is used to record a video of a person moving with a camera such as the Microsoft Kinect depth sensor or PlayStation Eye cameras. The motion capture solution is markerless, meaning that the person does not need to wear any special equipment during the motion capture session. The person in the foreground is separated from the background by performing a background analysis. This is done by firstly recording a video of the room for up to ten seconds. Then, a motion capture session is filmed. The camera is used to film a video of the person performing various actions that will be transformed into animation. Lastly, the previously filmed background is omitted from the final result and only the person's movements are processed into information. The information is then passed on to iPi Mocap Studio, the motion capture solution. iPi Mocap Studio finally transforms the motion capture data into animation by using a skeleton mesh for tracking of capture data. The tracked data is then used for a target character. The target character can be any character model that we want to prepare the animations for. iPi Mocap Studio conveniently has an option of exporting the animation for the standard Unreal Engine 4 Mannequin skeleton. After the animation is refined, cleaned up, and of satisfying quality, it can be exported to an Autodesk FBX format file and used for an Unreal Engine 4 game project.

This combination of software was selected because it supports the export of animation specifically for the default Unreal Engine 4 character skeleton, thus making the process of preparing animation assets easier. Furthermore, for the purposes of this final bachelor thesis, it is a great advantage and a possibility to quickly produce a large number of self-made animation assets that can be used with the developed project.

4.2 Developed Plugin

The details of plugin development are presented in this subsection, starting with the deployment of a virtual machine for storing animation assets that will be downloaded to game projects using the plugin, followed by the structure of the plugin, examples of how the developed plugin and downloaded animation assets can be used in a game project, and the details of technical implementation.

4.2.1 Deployed Virtual Machine

Figure 5. A screenshot of the contents of the "bakalauras2020" folder in the virtual machine.

For this final bachelor thesis project, a single OpenNebula virtual machine has been deployed. The virtual machine is running on Ubuntu 20.04 and Apache version 2.4.41 is used for the webserver. The files needed for the developed plugin on the server-side (animation assets and the animation list) are located in a separate folder. The animation list is the AnimationList.json file and the animation files are in their respective category (Emotions, Fighting, Gestures, Movement, OldMan, and Zombie) folders.

Because the virtual machine is running on the OpenNebula platform and the virtual machine does not have a graphical user interface, some difficulties arose when files had to be uploaded to the machine. To combat this issue a GitLab project called Bakalauras2020 was created, with the repository containing the file AnimationUASSETPayload.zip. The repository was then cloned on the virtual machine using the terminal "git clone" command, and the zip file was extracted to the bakalauras2020 folder. The AnimationUASSETPayload.zip file contains all the animation asset files and the animation list file, therefore the files simply only need to be extracted to the target folder. If there were a need to upload more animation assets with a modified animation list file, an updated zip file would have to be uploaded to the Gitlab project repository and then cloned on the virtual machine, deleting the old contents beforehand.

4.2.2 Structure and Usage of the Plugin

The plugin was developed using the Unreal Engine 4 Blueprint Library template, which is used to create blueprint nodes. The plugin was created using the placeholder name "FetchAnimations", therefore the root folder of the plugin and some of the source files have the FetchAnimations name as well. The plugin contains the "Binaries", "Config", "Content", "Intermediate", "Resources", "Source" folders and FetchAnimations.uplugin file which is the Plugin Descriptor.

The compiled C++ code and temporary build files are located in the Binaries and Intermediate folders respectively. The "Config" folder, as usual, contains the FilterPlugin.ini which is created by default and has not been modified. The "Content" folder contains no files as no additional content is offered with the plugin. The "Resources" folder contains an altered Icon128.png image file that

is simply a logo that is displayed next to the plugin details in the Plugins section. The "Source" folder contains the source code for the plugin, which is the FetchAnimations.cpp and FetchAnimations.h classes in Private and Public sub-folders respectively. The two source code files with their respective folders were created by default when selecting the Blueprint Library template with some minimal template source code, and all of the remaining source code for the developed plugin has been written in the FetchAnimations.cpp and FetchAnimations.h classes. The FetchAnimations.Build.cs file in the Source folder is responsible for how the plugin is built and is also created by default. The PublicDependencyModuleNames.AddRange function by default only contains the "Core" string in the string array. "HTTP", "Json" and "JsonUtilities" dependencies were added to the function string array in order to be able to use some of the functions needed for the development of the plugin, such as sending HTTP requests or working with JSON arrays. Finally, the FetchAnimations.uplugin file contains the plugin descriptor data such as the author name, category name, description, version name, etc. The final name for the plugin is "Download Animation Assets", and the plugin can be found in the Unreal Engine 4 Plugins Configuration using the specified name.

The source code of the plugin has a structure called FAnimationFile and a class called UFetchAnimations, with USTRUCT and UCLASS tags respectively. The USTRUCT tag defines a data structure and UCLASS defines an object class for the Unreal Engine. The FAnimation structure contains FString properties AnimationName, AnimationCategory, and FileType, and the structure is used for creating animation asset information objects from the animation list JSON file.
The UFetchAnimations class contains the following functions:

- static void type DownloadAnimationAssets – the main function, defined with Blueprint-Callable and Category function specifiers. with UFUNCTION tag. BlueprintCallable function specifier allows the function to be executed in a Blueprint and is represented as a Blueprint Node. The Category function specifier is used to define the category of the function, in this case, it is set to "Download Animation Assets" so that it can be found easier in the Blueprint editing interface. The function has EmotionsCategory, FightingCategory, GesturesCategory, MovementCategory, OldManCategory, and ZombieCategory parameters of bool type which are represented as checkboxes in the Blueprint Node for each respective category.

- void type GetAnimationList — responsible for creating and processing the HTTP GET request for animation list JSON file.

- void type GetAnimationAsset (with URL parameter of FString type) – responsible for creating and processing the HTTP GET request for an animation asset with the provided download URL.

- void type SetAnimationCategorySelections (with Emotions, Fighting, Gestures, Movement, OldMan and Zombie parameters of bool type) – responsible for setting the true or false values of the category variables of bool type, depending on the value of the Blueprint Node checkbox selections (if checked – true, else – false).

- void type FetchAnimationAsset – responsible for saving the received response content as a file, i.e. download an animation asset.

- void type FetchAnimationList – responsible for saving the received response content to an array (TArray) of FAnimationFile objects, therefore saving the list of animation assets to a variable that is iterated through when downloading each asset in the list.

- bool type getter functions for animation categories (GetEmotionsCategorySelection, Get-FightingCategorySelection, GetGesturesCategorySelection, GetMovementCategorySelection, GetOldManCategorySelection, GetZombieCategorySelection)
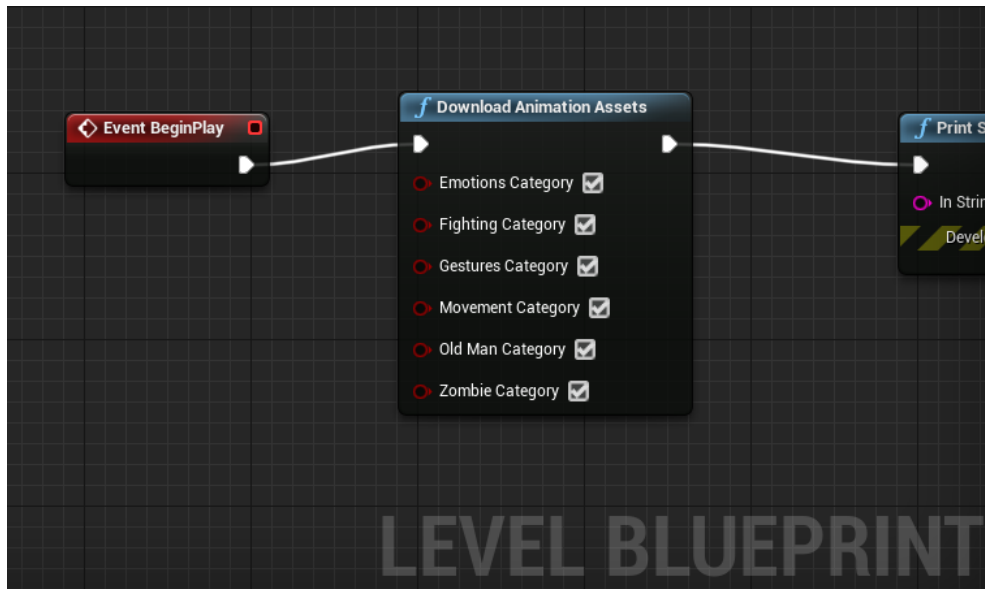

Figure 6. Example of using the plugin Blueprint Node by connecting it to the BeginPlay node in Level Blueprint.

The node is called "Download Animation Assets" and it can be added to any blueprint. The recommended way to use the node is to add it to the Level Blueprint and connect it to the Event BeginPlay node by the Execution Pin, like in Figure 6. In this case, the plugin is executed every time the Play button is pressed in the Unreal Engine Editor. The Blueprints Visual Scripting system is vast and can be used in a variety of ways, thus the plugin node can also be connected in different ways. For example, using the Event Tick node the plugin node can be executed multiple times during runtime, i.e. every frame of gameplay. Nonetheless, it may not be optimal to run the plugin functionality for every frame of gameplay. Another example would be to use the Set Timer by Event node. This node can be looped and fire an event every five seconds or another set amount of time. A custom Event node can be added to the Blueprint, connected to the plugin node by the Execution Pin, and "Set Timer by Event" node by the Event pin. In this scenario, after the Play button is pressed the plugin is run once the looped timer runs down. During runtime, any new animation assets added to the server, with their information appended to the animation asset list would also be downloaded to the game project. The plugin node has checkboxes for each of the six animation asset categories: Emotions, Fighting, Gestures, Movement, Old Man, and Zombie. Checking a check box will download all animation assets of the respective category once the plugin node is executed. If no checkboxes are selected, no animations are downloaded.

4.2.3 Fetching Animation Asset List

Before any animation assets can be downloaded, the animation asset list has to be fetched from the server. That is done using the GetAnimationList function. A GET HTTP request is created for the http://<virtual machine public IP>/bakalauras2020/AnimationList.json link, which eventually saves the contents of the animation list JSON file to an array variable, using the FetchAnimationList function. The JSON file contents are returned as a string and the JSON array objects are added to the AnimationList array of FAnimationFile structure using the FJsonObject-Converter::JSonArrayStringToUStruct function of the Unreal Engine 4 JsonUtilities module. The JSON file contains an array of objects that embody the information of animation assets on the server. The objects have string type attributes AnimationName, AnimationCategory, and FileType. All of the attributes are used to create a download URL for each asset, and the AnimationCategory attribute is used to identify each asset's category. Although all animation assets are of UASSET file type, the FileType attribute was still retained, even though all objects in the array have the same "uasset" value. The reason for that is because originally it was planned to also include the original FBX animation files that the UASSET animation asset files were derived of. One of the reasons for their absence is to avoid redundancy, as it is sufficient to only have the UASSET file for working with animations. The other reason because FBX files have to be imported into the game project by selecting a character skeleton for each animation file. If, for example, fifty-four FBX animation files were downloaded at once, the user would have to import each file and select a skeleton for it fifty four times, making the whole process irritatingly long. Hence, only UASSET files are downloaded, which already have the Unreal Engine 4 default skeleton applied and ready to use in any project without needing any additional settings.

4.2.4 Downloading and Using Animation Assets

GetAnimationAsset and FetchAnimationAsset functions are used to fetch an animation asset from the server and download it to the game project. After the animation list is fetched and stored as an array of objects, the array is iterated through, and for every animation asset, a download URL is created using the virtual machine address http://<virtual machine public IP>/bakalauras2020/, appending it with the animation asset category, file name, and file type. Once the download URL is created, it is passed on to the GetAnimationAsset function as an FString variable. The procedure is similar to fetching the animation asset list – A HTTP GET request is created for the provided URL and the response data is saved. However, whereas the animation list data is saved to a local variable, the animation asset data is saved to a file in the game project. This is done for every object in the AnimationList array of FAnimationFile structure objects. The assets are downloaded to the "DownloadedAnimations" folder under the game project's Content directory.

Before the process of downloading a single animation asset is started, a check is performed whether a file with the current iterated object file name and file type exists. If the animation asset already exists, it is not downloaded again, so as not to waste Internet bandwidth and needlessly overwrite the same existing file.

Figure 7. A screenshot of an Unreal Engine 4 game project, with a couple of newly downloaded animation assets added to the level environment. The downloaded animation assets are seen on the right.

For this project, a total of fifty-four animation assets over six categories (Emotions, Fighting, Gestures, Movement, Old Man, and Zombie) were produced. The animation asset files are in UASSET file format instead of FBX, therefore the user is not prompted to select import options for each asset, which would be the case with FBX format files. The downloaded animation assets are instantly ready to use without any additional setup needed. All fifty-four UASSET files take up around 8.49 MB of storage. In comparison, the original fifty-four FBX files that the UASSET files were derived from by Unreal Engine 4 take up around 37.6 MB of storage. If required, the animation assets can still be modified, e.g. looping interpolation can be added for smoother looping, compression can be adjusted or the animation assets may even be exported to an FBX format file. The animation assets were produced for the Unreal Engine 4 Mannequin skeleton, therefore if the user wanted to use the animations with another game character model, for example, one downloaded from Mixamo, animation retargeting would have to be applied, using the Unreal Engine Retarget Manager.

5   Implementation of Animation Blending Testing Tool

This section contains the details of implementing an animation blending testing tool as an Unreal Engine 4 game project. It includes creating a game project with some functionality, i.e. controlling a character and providing the option to perform a couple of actions, as well as developing an interface for easy preview of the animation blending prototype. Also, a prototype of character

25

blend weight randomisation is presented as a proof-of-concept for adding multiple characters to the environment, each having some uniqueness in motion.
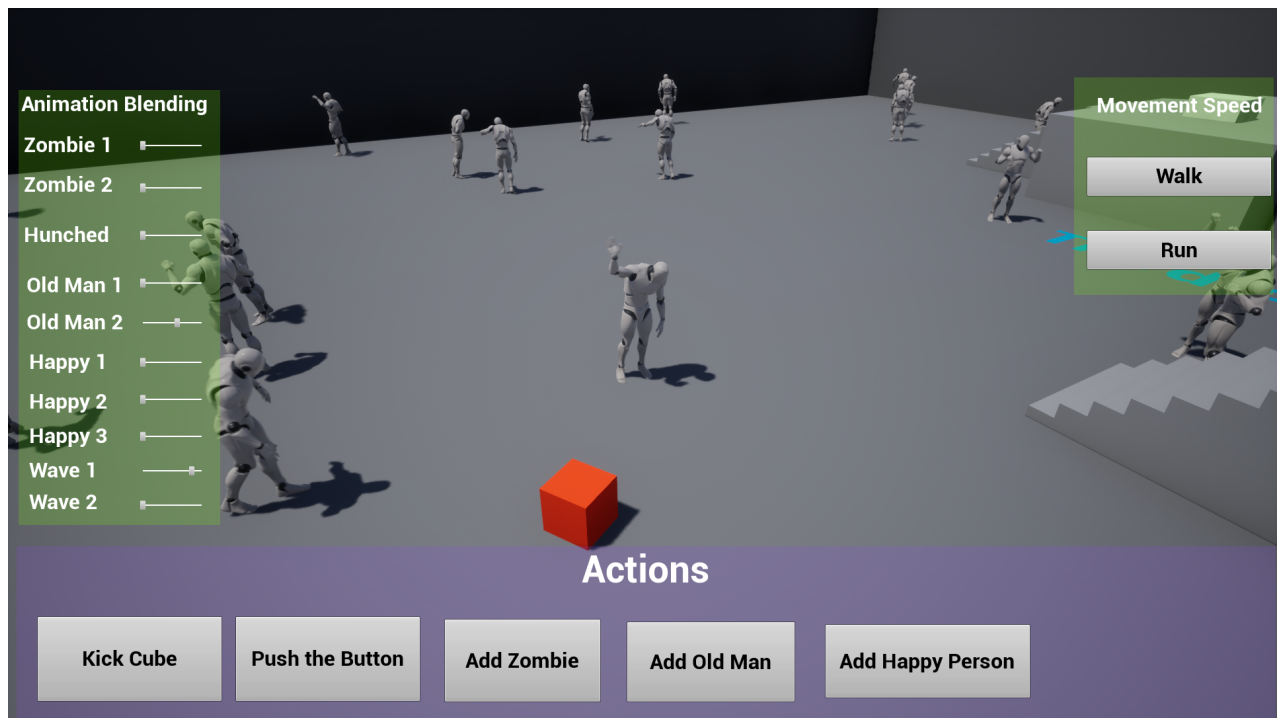

Figure 8. A screenshot of the Unreal Engine 4 animation blending testing tool. Two additive animations (Old Man 2 and Wave 1) are added to the center character animation set at different blend weights, creating new character motions.

For purposes of prototyping the implementation of animation blending, a game project was created using the "Third Person" template. A small 3D environment was built with some slopes, sets of stairs, a button, and a cube. The on-screen 3D character is using a modified Animation Blueprint that contains some additional Animation States which are used for tasks ranging from walking and running, to kicking an in-game cube or pushing a button. The tasks of kicking a cube or pressing a button are performed whenever the user presses either the "Kick Cube" or "Push The Button" buttons respectively in the user interface. The in-game character also has some simple built-in artificial intelligence, and can be moved around the environment by pressing the Left Mouse Button somewhere in the environment – the "AI MoveTo" Level Blueprint node is executed for the character as it walks to the destination provided by the mouse click. The user interface also contains buttons for adding characters that have characteristics of a zombie, an old person, and a happy person. When either the "Add Zombie", "Add Old Man" or "Add Happy Person" button is clicked, a function in the Level Blueprint is called that adds a character to the environment in a random spot, and instructs them to walk to a random spot. The characters have one or several additional animations applied to them, either of a zombie, an old man, or a joyful person, and the additive animations have random blend weights that range from 0.1 to 0.9. The sliders on the left side of the screen represent the various additive animations that can be blended with an existing animation set that is applied to the character. Sliding any of the sliders from left to right controls the blend weight (leftmost setting of the slider representing 0.0 and rightmost setting of the slider meaning 1.0) of the respective animation, thus making it more prominent and altering the motions of the character. All of the game logic was developed in the Level Blueprint, and the character animation logic was developed in the default Animation Blueprint ThirdPerson_AnimBP_C which

was modified with new logic and character animation assets.

The purpose of the developed testing tool is to display the possibilities of animation blending in Unreal Engine 4, i.e. visualise how new character behaviors can be produced by blending one or more additional animation assets to the whole animation set of an in-game character. It is intended to help prototype such systems as, for instance simulating a large environment with dozens of non-playable characters that should have different characteristics for a more natural presentation.

5.1 Blending Animations for the Character Skeleton

The in-game character animation blending is performed using the Animation Blueprint's Anim-Graph. Firstly, the AnimGraph contains the Default State Machine, which is the animation set for the character, made up of Transitions between States such as "Idle/Run", "KickCube", "PushButtonStart", etc. Each Animation State has its own logic, for example, the "Idle/Run" state contains the "ThirdPerson_IdleRun_2D" Blendspace Player node which is responsible for blending between walking and running animations based on the value of the "Speed" variable. Secondly, the AnimGraph contains numerous "Layered blend per bone" nodes, each having inputs for Base Pose, Blend Pose, and Blend Weight, and output for the result Animation Pose. Ten "Layered blend per bone" nodes are used in total, with "Zombie1", "Zombie2", "Hunched", "OldMan1", "OldMan2", "Happy1", "Happy2", "Happy3", "Wave1" and "Wave2" animation assets as Blend Pose inputs, and their respective Blend Weight variables as Blend Weight inputs for each node. For the first "Layered blend per bone" node the Default State Machine is used as an input for Base Pose, as the intention is to apply the layered blending for the entire animation set. The output of the first "Layered blend per bone" node (the result Animation Pose) is used as input for the second node's Base Pose, the second node's output is used for the third node's Base Pose and so on, until finally the tenth node's output is used for the "Output Pose" node, i.e. the final character animation. The result can be defined as the original animation set but with applied ten layers of additive animations that are blended based on the defined Blend Weights.

In this case, the Unreal Engine 4 Animation Blueprint "Layered blend per bone" nodes are used instead of "Blend Multi" or "Blend" because for some animations, the blending needs to be applied only for a part of the skeleton. The "Layered blend per bone" node provides more control over character skeleton blending than, allowing omitting certain bone chains from the blending process. For example, the additive "Wave 1" animation is blended only for the "clavicle_r" bone (and its child bones) of the character skeleton, applying the waving animation only for the right arm and nothing else. Otherwise, if the blending were applied for the whole skeleton, animation artefacts may arise such as foot sliding. As the "Wave 1" animation consists of only the character waving their right arm with the rest of the body remaining static, it would not make sense to blend the whole animation for the existing animation set. There may be scenarios when the "Blend Multi" node may be used for more convenient blending of several animations for the whole skeleton, however, "Layered blend per bone" proved to be the superior choice for combining character animations.

Problems may occur when animation blending unintentionally overrides some animations of the original animation set. This can impact character interaction with their environment. For example, when the "Wave 1" animation blend weight is set to 1.0, the character skeleton right arm bone motions are overridden by the waving motion for all animations in the Animation State Ma-

chine. This means that for instance performing the "Push The Button" action may not be possible as the right hand, which is usually used for pushing the button in the environment, is used for waving motions. Another example is applying the blending of "Zombie 2" at 1.0 blend weight and attempting to perform the character action of kicking a cube in the environment. Because the original "Zombie 2" animation has the actor moving their legs in a zombie-like fashion, part of that animation information is applied for the character set animations such as walking, running, and kicking. Therefore, the kicking motion may be less prominent with the "Zombie 2" animation applied for blending, resulting in the character kicking the cube with less force than normal. This issue was deliberately retained in the blending testing tool as a means to visualise what happens when animation blending for certain animations overrides other motions. The issue can be solved by using the "Blend Poses by bool" node and specifying the conditions for when the animation blending should be used, and when it should be disabled. Animation artefacts may also appear when the blend weight for a specific additive animation is set too low. Taking the same "Wave 1" animation as an example, setting the blend weight to 0.5 results in the right arm being raised only halfway, resulting in an unnatural motion. That is because 50% of the original animation is being used, e.g. standing still with arms relaxed, and 50% of the waving animation is used, and blended into one pose, resulting in the arm not being raised fully. Setting the blend weight to something in the range from 0.8 to 1.0, however, results in the right arm motion resembling the waving gesture. Therefore, the range of minimum and maximum blend weight has to be taken into consideration when working with certain animations, possibly finding the best result through trial and error.

In theory, a single additive animation with differently applied blend weights applied to the entire character animation set is enough to implement uniqueness and variety in character motions. For example, during a motion capture recording section a single animation can be recorded, such as the actor standing in one place and acting hurt, hunched up, laid back, etc. That single animation asset, once exported may be applied to the character default Animation State Machine, ensuring that the motions of single animation are blended with all animations. It is up to the developer to decide when the animation blending ought to not be applied, etc. when the character is performing complex motions that the blended animation may override, or which character skeleton parts should be blended, etc. when to blend animations only for the character arms, or the upper part of the body. When even more character behavior variety is needed, more animations can be recorded and blended to the character animation set by layers.

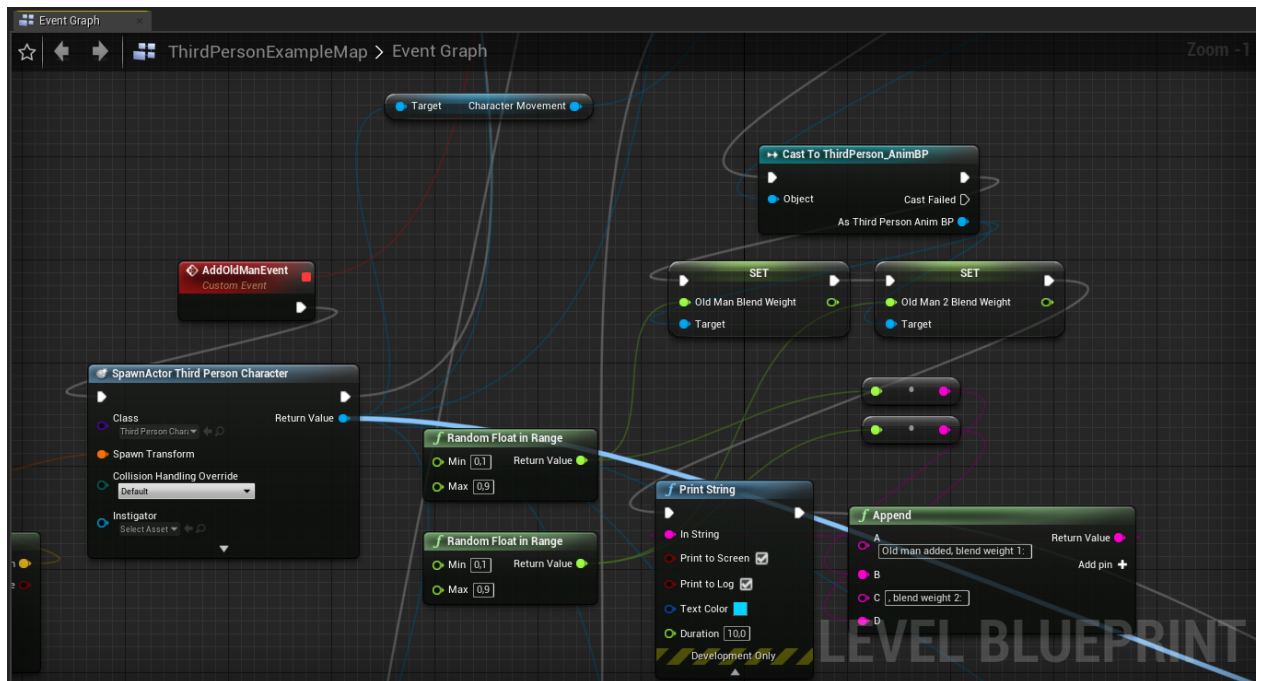## 5.2   Randomizing the Character Animation Blend Weights

Figure 9. A snippet of the Unreal Engine 4 game project Level Blueprint's Event Graph. Part of the function for spawning an in-game character with characteristics of an old person (using old man animation assets), assigning random blend weights for each new character.

One of the risks of populating the simulation or game environment with high amounts of non-playable characters is that the characters may display the same characteristics. For example, their walking patterns or body motions may be extremely similar and indistinguishable, resulting in an almost robotic and unnatural display of character crowds. Even with animation blending this may be an issue of all of the characters are using the same blend weights, resulting in all characters being blended the same way. Even though a developer can define different blend weights for each added character, it may be time-consuming, especially if the game world needs to be populated with hundreds of characters. To solve this problem, blend weight randomisation can be used.

The animation blend weights are expressed as variables of float type that may range from 0.0 to 1.0. For each new additive animation asset in the Animation Blueprint, a public float type variable can be created, which is set as the animation blend weight that can be modified in the Level Blueprint. If, for example, we needed to add ten, twenty, or even a hundred in-game characters that would have to be blended differently to achieve variety in their behaviors, the blend weights would have to be randomized. One way to achieve this is by using the "Random Float in Range" node that returns a random float value in the specified range. In this case, the range can be from 0.0 to 1.0. Once either the "Add Zombie", "Add Old Man" or "Add Happy Person" button is clicked in the interface, a new non-playable character. i.e. the actor is spawned. For each spawned actor the random float values in the specified ranges are generated by "Random Float in Range" nodes and can be set as the values of blend weights. However, in order not to spawn characters that have no blending applied for certain layers, or too much of it applied (which can override some animations in the existing animation set, or cause animation artefacts), we can set the range from 0.1 to 0.9. The minimal value of the range may be increased to escalate the likelihood of stronger animation

blending. For example, setting the range from 0.7 to 0.9 for "Old Man 1" animation (which is the animation of a character hunched with their left arm behind their back) increases the chance that all spawned characters of "Old Man" type will have their left arm closer to their back. In contrast, setting the range to, for example, from 0.1 to 0.5 will spawn characters that are only slightly hunched up.

Setting the random blend weight values in specified ranges solves the problem of user intervention, i.e. the developer does not have to manually set blend weights for each newly added in-game character. Therefore, this can be used for automation of adding non-playable characters in a 3D environment, with unique animation properties. Nevertheless, experimenting with randomly generated values may still be needed when developing an automated character population system. For example, problems may occur when the difference between the minimum and maximum values of the range for the "Random Float in Range" node is low, e.g. from 0.4 to 0.5. In this case, the randomly generated value applied to blend weights will not be very dissimilar among the added characters, resulting in less variety.

Conclusions and Recommendations

The result of this final bachelor thesis project is the developed Unreal Engine 4 game engine plugin that provides character animation assets that are already applied to the Unreal Engine 4 default character skeleton and instantly ready to use, and an animation blending testing tool that helps visualise how animation blending can be used to mix several existing animations for one skeleton, producing new character motions. Nevertheless, the project can still be improved in some ways. Even though over fifty animation assets have been produced for the project, a broader variety of animations and categories would be beneficial to the usability of the plugin. Also, there is no user interface for easily uploading new animation assets to the virtual machine and the project would greatly benefit from having one, for example via a dedicated website. Furthermore, the plugin lacks some visual feedback, such as previews of downloadable animation assets or even descriptions, therefore the user can only select an animation category and download animation assets without having an option to view them before downloading. This can be attributed to the visual limitations of the Unreal Engine 4 Blueprints Visual Scripting system. As for the animation blending testing tool, it can be improved by adding more animation blending scenarios with visual explanations. Nonetheless, the conclusion can be made that it is indeed possible to streamline the character animation workflow in Unreal Engine 4 by providing more straightforward access to game assets, i.e. downloading them directly from a server with a plugin, and by using animation blending to dynamically produce new, unique character behaviors with a minimal amount of existing animation assets. In the case of the blending testing tool specifically, there are possibilities of evolving it from a visual prototype to a more sophisticated system, usable for generating high amounts of distinctive non-playable characters in 3D environments, not only for Unreal Engine 4 but for other game engines as well.

References

[1] Epic Games. Balancing blueprint and c++. https://docs.unrealengine.com/en-US/Resources/SampleGames/ARPG/BalancingBlueprintAndCPP/index.html, 2020.

[2] Antonín Šmíd. Comparison of unity and unreal engine. *Czech Technical University in Prague*, 2017.

[3] Luis Delicado Alcántara. Synthesising character animation for real time crowd simulation systems in unreal engine. Master's thesis, Universitat Politècnica de Catalunya, 2018.

[4] Epic Games. Plugins. https://docs.unrealengine.com/en-US/ProductionPipelines/Plugins/index.html, 2020.

[5] Rune Skovbo Johansen. Automated semi-procedural animation for character locomotion. *Aarhus Universitet, Institut for Informations Medievidenskab*, 2009.

[6] Stéphane Ménardais, Franck Multon, Richard Kulpa, and Bruno Arnaldi. Motion blending for real-time animation while accounting for the environment. In *Proceedings Computer Graphics International, 2004.*, pages 156–159. IEEE, 2004.

[7] Andrew W Feng, Yuyu Xu, and Ari Shapiro. An example-based motion synthesis technique for locomotion and object manipulation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 95–102, 2012.

[8] Zeeshan Bhatti, Imdad Ali Ismaili, Shehnila Zardari, Hafiz Abid Mahmood Malik, and Mostafa Karbasi. Procedural animation of 3d humanoid characters using trigonometric expressions. *Bahria University Journal of Information & Communication Technologies (BU-JICT)*, 9(2), 2016.

[9] Epic Games. Animation blueprints. https://docs.unrealengine.com/en-US/Engine/Animation/AnimBlueprints/index.html, 2020.

[10] Janessa Aulén. Animation state machines in unreal engine. Bachelor's Thesis, 2020.