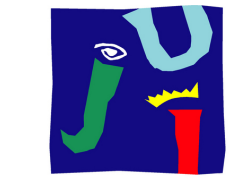


Animating feelings: Comprehending and replicating character psyche through Animation

Pau Martínez Fortuny

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

July 24, 2023

Supervised by: Diego José Diaz García

To Maribel, thank you for always being there

Acknowledgments

First of all, I would like to thank my Final Degree Work supervisor,Diego José Díaz Garcia, for lending me his help when I needed it.

I would also like to thank Pierrick Picaut for his amazing effort and dedication to teach animation.

Finally, I would like to thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this report.

Abstract

This document presents the project report of the Final Project for the Video Game Design and Development Degree by Pau Martínez Fortuny. It includes a comprehensive introduction and theoretical approach to evoking personality and emotions in 3D animations. Additionally, the project puts that theory into practice by animating humanoid characters for video games. Besides, the project also aims to create a Unity Add-on for a user friendly interface that allows to organise and visualize humanoid animations in order to see if they fit a certain character.

Important consideration: In order to visualize the animations within this PDF file, it is imperative to open it via a PDF reader such as Adobe Acrobat Reader, and not a browser.

Keywords: Animation, Blender, Unity, Character personality

Contents

CHAPTER 1

INTRODUCTION

Contents

This chapter must clearly reflect what is going to be done during the development of the work. Although the fundamental point is to state the objectives of the presented work, it is also interesting to comment on the need, idea, etc., that motivates it, and the state from which it was started.

1.1 Work Motivation

Growing up, I have always been fascinated by animation movies, as any kid my age would. I found animated movies to be particularly impressive in their ability to tell captivating stories that kept me thoroughly engaged. Films such as *Kung Fu Panda*, *The Incredibles*, and *Shrek* are just a few examples of the films that sparked my interest in animation. In retrospect, and after many years I have come to the conclusion that what kept me at the edge of my seat whilst watching these movies was thanks to an extraordinary narrative and technical effort of a team of talented people. Due to this, my fascination with animation has grown exponentially in the last couple of years, and I aspire to be able to one day convey the same sense of wonder and excitement that I experienced as a kid to other people.

As of today, the field of 3D animation has experienced a tremendous amount of growth, with many advancements in technology, software and resources allowing for more realistic and lifelike animations in all media. However, despite all this, I find

that on many occasions, video game developers cut corners in character animation, especially when it comes to big projects (which is totally understandable given their tight schedules). They normally struggle to convey the personality and emotions characters portrayed in their worlds, leading to characters that feel lifeless and flat, which takes a lot out of the immersion of said world.This is particularly true for humanoid characters, where the challenge is to make them appear human-like while still maintaining the unique characteristics that make them distinct.

Personally, working on a subject that has not been thoroughly expanded upon in the degree adds an additional level of excitement, as the opportunity to learn and work with a whole new subject does not present itself often. Furthermore, this project holds a lot of importance to me, as it relates closely to what I want to work in the industry once I finish the degree. Therefore, I think of it as not only a project I would love to do, but an opportunity to finally develop my skills in this subject and to create a portfolio not just as an animator, but a technical animator.

1.2 Objectives

This section is essential so that the reader can understand what has to be accomplished by the work. Everything that was intended to be achieved must be described with objectivity and precision.

Also, it should be indicated in this section if the work is part of a larger job, if it is complemented by others, etc.

In short, the readers of this report, after reading the work objectives, should clearly know what the author aims to achieve on his own.

As for the objectives of the project, there are many. Some of them are purely academic related, whereas the other ones have more to do with personal development, such as acquiring new skills. These are the main objectives that this project aims to achieve:

- To research and understand the industry's standard when it comes to developing character animations.
- To animate a character taking into account the principles of animation.
- To explore techniques and approaches to making humanoid 3D animations
- To better comprehend how to convey personality to a character via movement.
- To develop a user friendly add-on for setting animations into characters based on their personality or other factors in Unity.
- Research further uses or implementations that may also be tied to this project.

1.3 Environment and Initial State

As previously stated, we will explore techniques and approaches to making humanoid 3D animations feel alive and portray the personality of the characters they represent. This will be done by analysing existing literature and implementing what we learn onto our own animations, as we aim to contribute to the ongoing conversation surrounding the creation of realistic and engaging 3D animations. I'm not starting from ground zero, as having passed both vj1226-character design and animation and vj1223 - video game art I already have the basic knowledge of both technical animation and animation principles respectively. On top of that, I also watched many videos breaking down famous animated scenes.

In order to orderly find information on the subject, I was faced with a sea of results. A quick search of "3D character animation" resulted in more than 352 million results, and trying to narrow it down by putting "learning 3d character animation" resulted in 118 million results. Seeing these results was very daunting at first, as most of the results were either focused on a very specific subject or hidden behind a paywall. Most of the academic studies on the matter were also mostly extremely technical or didn't shed any light on the inner workings of the animation's industry. I certainly was puzzled, there were just so many techniques and tools currently being used in the industry for me to make a comprehensive analysis.

Therefore, I decided on making a subjective and fair comparison of the most widely used animation techniques in the video game industry as of today which are Skeletal animation (making use of rigging) and Motion Capture. I also hope that this document may be of help for those who might find themselves in the same situation as I have.

But aside from the more theoretical aspect of the project, there is also the practical aspect. And this part aims to tackle a problem that I myself have encountered during the degree, that is the lack of a way to orderly and consistently organise and and assign your animations to your characters in the Unity engine, as I usually just pick any animation (usually from an external site like Mixamo) and manually configure it for each and every character. Now this might seem like a minor drawback, but in games with many agents it is really a pain.

Again, searching for solutions to the problem both inside of Unity and in external add-ons in the Unity Store brings up no feasible solution or a piece of software hidden behind another paywall. Therefore I have decided to create my own add-on to help in this matter. The idea is to make it so one can organise its animations by character traits so it's easier to find an animation that suits a certain character and assign it to them. A sample scene demonstrating the use of the add-on should also be necessary in order to properly show how it works.

Chapter 2

Planning and resources evaluation

Contents

In order to keep on schedule, it is imperative to pay special attention to the planning. This section will show how the different parts of the project are divided, and the time they took to be realised. To accurately show them, I will keep track of what has been done and how much time it took.

2.1 Planning

- Task 1 (30 hours): Read about relevant literature on character animation and personality portrayal.

- Task 2 (60 hours): Study about theory of emotion, animation history and principles,and compare different animation techniques within the industry, portraying their individual advantages and disadvantages, taking into consideration portrayal of a character's feelings/personality as the main objective.

- Task 3 (20 hours):Learning about how professionals in the video game industry generally work. Animation software and the different stages between idealising an action and having it ready to implement in a video game.

- Task 4 (70 hours):Taking into account everything learnt, create a set of several humanoid animations to the best of my ability. At the very least there should be two or more styles for each type of animation (idle, walking, jumping).
- Task 5 (60 hours):Learning and developing a personalised add-on within Unity so it's easier to add and preview animations to your characters, sorting available animations via character traits.
- Task 6 (5 hours):Analysis and design document.
- Task 7 (60 hours): Work on the memory of the project and presentation. Total: 305 hours.

2.2 Resource Evaluation

In this section I'll list all hardware and software necessary for the realisation of this project, as well as other information such as human and equipment costs. Not all these are a must have for the development of the project, but they serve as a way to speed things up along the way.

- **Desktop personal computerWindows 10 (Intel(R) Core(TM) i7-7700K CPU, 16,0GB RAM, NVIDIA GeForce GTX 1050 Ti GPU) (around 1000€)**: This is the device I will be mainly focusing the development of this game on.
- **Unity 3D 2021.3.10f1 (free for educational purposes):** The game engine of choice to develop the add-on and create a sample scene.
- **Blender (free):** This will be the software I will be mainly using to develop my character animations.
- **Visual Studio (free):** This tool, paired with unity will allow me to code and debug the inner workings of the add-on.
- **Trello (free):** A website for task managing and time control. I will also keep track of time spent on each time in this workspace.
- **Unity asset store (free):** I will be using the asset store mainly for research purposes and to search for assets that may be useful.
- **GitHub (free):** This program will be very useful, as its code hosting and version control will make it so I can't accidentally lose much progress on the project.
- **Labour (2,158€):** This is the estimated cost of human labour, taking into account the average junior salary in the field of animation and the hours worked.

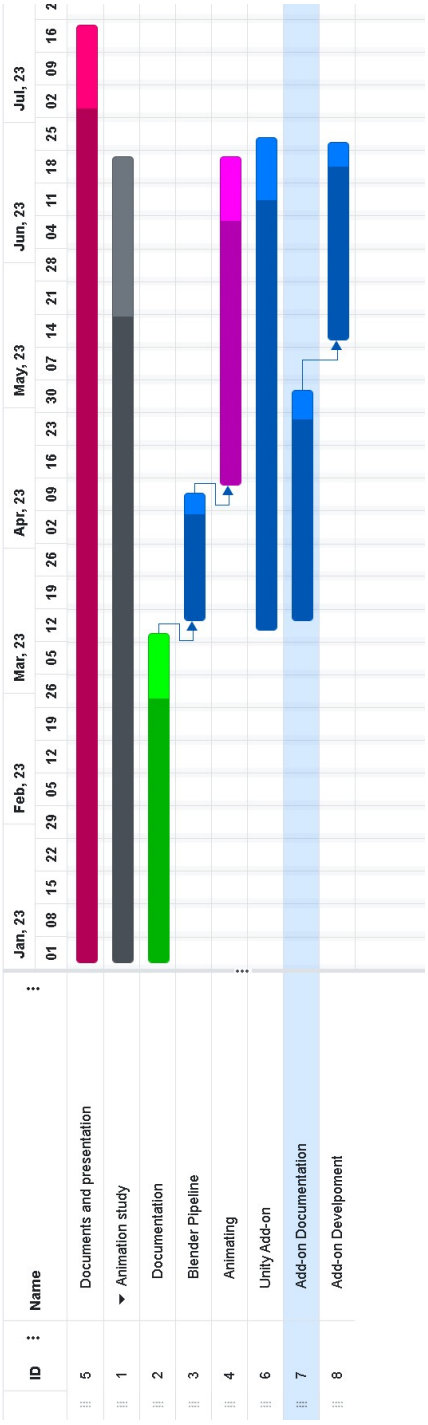


Figure 2.1: Gantt chart for the project (made with online Gantt)

2.3 Time verification

In this section, I will compare the initial times proposed on the 2.1 section with the real time that has been dedicated onto every task. Justifying the possible deviations that can derive from what was initially estimated.

Task list		
Task Name	Initial est. time	final time
Task 1	30	45
Task 2	60	60
Task 3	20	15
Task 4	70	70
Task 5	60	65
Task 6	5	5
Task 7	60	70
TOTAL	305	330

Although generally I have stayed within a certain margin of the estimated time, I have overshoot it a bit, mainly due to an unexpected time increase when documenting and learning about the process of animating and Unity documentation.

CHAPTER 3

Personality in Animation: Theoretical Analysis

Contents

3.1 Animation Basics

3.1.1 Historical Overview on the evolution of emotional expression in animation.

I believe that studying the evolution of emotional expression throughout the history of animated media is a key factor in order to understand which aspects of this field have changed over the years and which have remained unchanged, therefore constituting what we can consider to be the most essential in this field. It also helps in knowing the theoretical and technological aspects that have helped to influence said development. This can provide us with crucial contextual information for this study, and establish a solid foundation on which to build this research. The next subsections constitute a rough approach to differentiate the stages of this said evolution, starting from the early forms of traditional animated media for the masses, in the 1920s and 30s, to contemporary trends in the field:

Early Beginnings: Although the concept of animating images has existed for hundreds of years, usually by the means of rapidly and successively going through a set of concurrent images, just like a flipbook, we are more interested in the form of traditional animation, which originated in the early 20th century as a novelty way to tell stories and transmit emotions. One of these early examples would be Winsor McCay's "*Gertie the Dinosaur*" (1914), which utilised innovative keyframe animation to convey personality and emotion in a dinosaur character. As this technique evolved, many animators, such as Walt Disney, began to push the boundaries of emotional expression in animation, with the creation of characters such as Mickey Mouse, or Snow White. For the first time, the world was introduced to animated characters that were able to portray a wide range of emotions, from joy and love to fear and sadness.

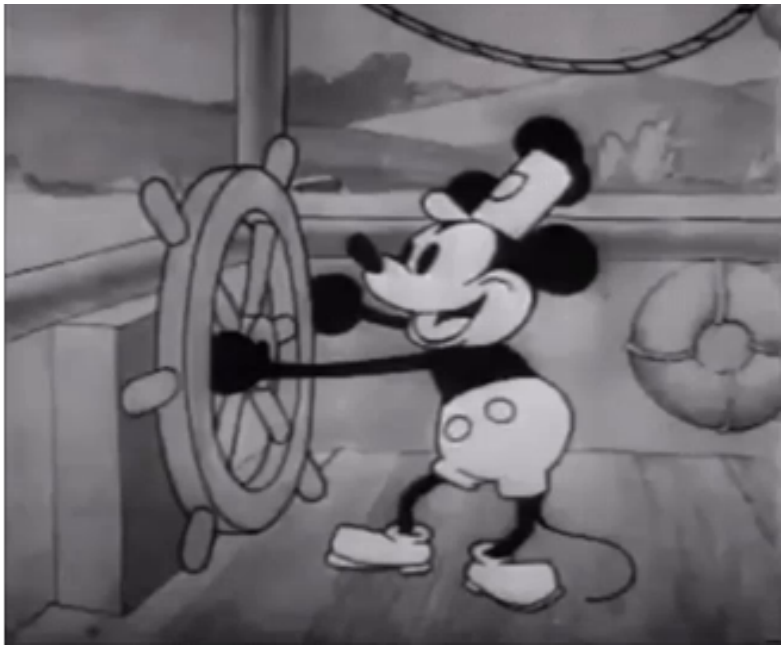


Figure 3.1: "Steamboat Willie" 1928 Disney animated film.

Golden Age of Animation: This period that took place in the mid-20th century and led Warner Bros and Disney, is usually referred to as "*The Golden age of Animation*" because of the increase in popularity that this medium received. It marked a distinct and significant shift in the way characters were portrayed, often opting to introduce characters with a rich emotional background that resonated more with the audience. An example of this type of character would be Bugs Bunny which often went over the "one dimension" trope of a comedic character in order to convey other emotions or to even evoke political or social messages to the audience.

Modern Animation and Technological Advancements: The introduction of computer animation between the late 20th century and the early 2000s, was a revolutionary event in the field, opening up many possibilities for digital techniques and effects,

not achievable otherwise. Computer animation allowed for an automated interpolation between different frames or poses, thus 3D models were introduced to feature films. This new technique also cleared the way for many new and exciting possibilities for emotional expression. Pixar's "*Toy Story*"(1995) is a prime example of this phenomenon. Being the first feature-length film made entirely in CGI, not only it introduces many new digital techniques, such as the Catmull-Clark subdivision surface for their animated characters but also portrays nuanced emotions via character movement and facial expressions in a whole different way than how traditional animation archived it, demonstrating the potential of this new technology for evoking deep emotional responses.

Contemporary trends: The aforementioned technological evolution of the field of animation continued throughout the end of the last century up until today, and it continually pushed the boundaries of what was possible in the realm of emotional expression. Over the last few decades, 3D animation has become the default form of this art expression in the mainstream media, leading to a certain point in which although the technical aspects of making these animations is constantly evolving, its foundations remain unchanged. That is until, in the last few years, a new wave of animation that challenges what is established is changing the environment of the industry once more. This renaissance of stylistic experimentation is mainly due to the desire of many animators that want to push the boundaries of modern animation in order to create new visual experiences. The most iconic example of this, and probably the movie that kicked off this movement, is "*Spider-Man: Into the Spider-Verse*" (2018). It challenged the status quo by introducing alternative visual styles, such as a comic style, and blending them seamlessly with traditional CGI. This unique animation style combined 2D and 3D animation in a manner that was never done before, and resonated both visually and emotionally with the audience. Its great success and reviews were what signalled the start of this new stylistic era within the animation industry.

Another very recent trend in the field of animation worth mentioning and that has come with the recent popularisation of AI technology is AI animation. Although there are multiple ways that AI technology is being added to animation workflow, the technique that has seen the most improvement over the last few months and is being predominantly used in the internet is a technique that mimics the effects of rotoscopy animation, by layering AI generated images (via Stable Diffusion or any other method) over video footage, this tool has been greatly polished by the team at Corridor Crew in late February 2023. This technology has the potential to greatly shape the future of the industry, but as of today it has not been yet implemented as an industry standard or in its usual workflow.

Moving forward, it will be very interesting to see the way these trends are continuing to evolve and shape the future of animation styles and how they will affect emotional expressions, as the possibilities seem endless. Some fascinating books on animation and it's history are [7] [3]

Figure 3.2: "Spiderman:Across the Spiderverse" 2023 by Sony Pictures

3.1.2 Literature Review

There are many books on the topic of animation, as it is a broad subject. Nonetheless, some of them are regarded as indispensable by professionals all around the world. These are some of the best recommendations I've gathered, some being of an introductory level and others more focused on the topic that this thesis aims to cover.

- **"*The Animator's Survival Kit*" by Richard Williams [16]:** This is definitely the "go to" book for every aspiring animator of any kind, from traditional to digital. It consists of a comprehensive guide on the principles and techniques from traditional hand drawn-animation. It is nonetheless a very complete guide, as it talks about must know topics that can translate to all kinds of media outside traditional animation, such as the principles of animation (which we will talk extensively about in this document), analysing and breaking down movement, character design or acting/posing of the characters.

- **"*Character Animation Fundamentals*:** Developing Skills for 2D and 3D Character Animation" by Steve Roberts : In order to further develop your base knowledge of the essentials, as well as introducing you to the world of 3D animation, this book is very useful. It also introduces the concept of character personality and ways to show emotion within an animation which is a nice addition. It is however a bit too vague sometimes as it tries to focus on so many topics at once. Nonetheless it is still a good read.

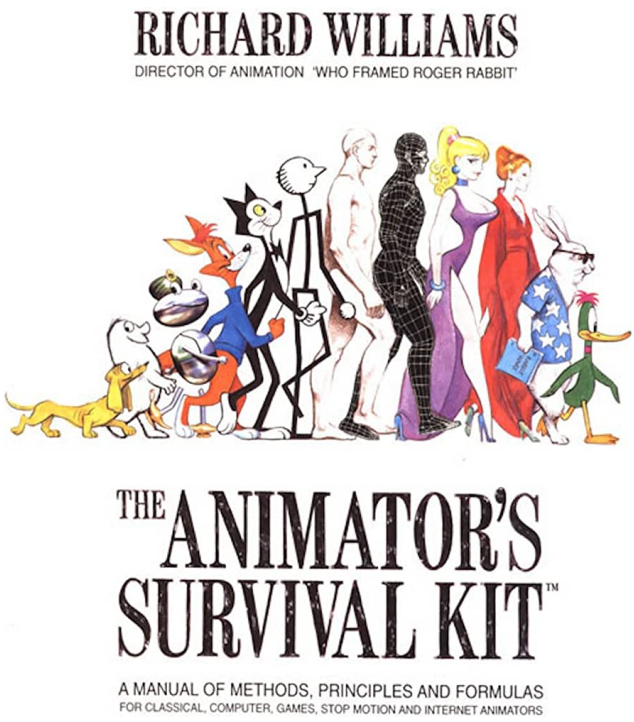


Figure 3.3: "Animator's Survival Kit"by Richard Williams

- **Designing Creatures and Characters:** How to Build an Artist's Portfolio for Video Games, Film, Animation and More" by Marc Taro Holmes: In this book, the focus is shifted from motion to design. It provides powerful and useful tips for an artist to design compelling and distinct characters. There's even a section where it explains how personality traits affect a character's design. I find it really easy to extrapolate concepts from this book to that of animation, which is really useful.
- **"*Acting and Performance for Animation*" by Derek Hayes [5]:** This might be the book that's most in-line with this thesis, as it mainly focuses on creating lively and expressive character performances, taking into account that character's personality. Some of the most important features highlighted are the importance of body language, acting techniques and pose breakdowns, as well as emphasising the importance of referencing as much as possible. Having many exercises in order to strengthen those teachings and having interviews with professionals in the field to share their insight on creating believable performances is the cherry on top.
- **"*Blender Character Animation Cookbook*" by Virgilio Vasconcelos:** When it comes to books on specific software, especially those that are focused on one theme, scarcity is the norm. For teaching the user the basics of animation techniques on Blender in a formal manner, this is a good introduction. Despite that, when it comes to software, I believe that these kinds of books get outdated really

fast, and this is especially the case for an open sourced program that receives constant updates like Blender. Maybe looking up recent iterations of how the latest version works online might be more useful.

- **Online authors**: What I have come to find as one of the most useful resources when it comes to learning animation, is to simply follow professionals online, on YouTube or even their social media. Most of them usually provide a lot of incredible insight, up to date content, and even bring up relevant breakthroughs I would've never discovered if not for them. To name some of them, Pierrick Picaut [9], Michael Cuevas [4]and Alan Becker [2] would be amongst the ones that have been the most helpful and inspirational to me.

3.2 Portraying personality and Emotion

Evoking emotion is an integral part in the art of animation. To convey a wide array of emotions through the subtle manipulation of movement is crucial for the audience to connect with any character or narrative. Therefore, as an animator one must go through the process of researching both motion and emotions, and understanding how they relate to each other.

3.2.1 Theory of Emotion in Visual Arts and Animation

Psychological Perspectives and motion: In discussing theory of emotion in animation, it is essential to understand the psychological basis for emotions. Theories such as *James-Lange theory*, *Cannon-Bard theory*, or the *Schachter-Singer theory* explore how our physical and psychological responses are intertwined between themselves and environmental stimuli, and how together they create emotional experiences. In the field of animation, these theories translate into an understanding of how motion can induce an emotional response and vice-versa. For instance, fast-paced, erratic motion might induce a feeling of excitement or anxiety, whilst low, fluid movements might instil a sense of calm or sadness. In essence, the manner in which a character moves can significantly influence the emotional tone of a scene. References:[1] [6]

Kinesthetic Empathy: The concept of kinesthetic empathy, although fairly modern, plays a crucial role in many fields, but particularly in the arts, such as acting or animation. This idea basically says that an audience emotionally and subconsciously respond to and mirror the movements they see on screen (or in real life). When characters move in ways that convey emotion, we subconsciously recognise those movement patterns, thus feeling or empathising with said emotions to a point. One good example of this would be when someone you are engaging with yawns. More often than not, you are more likely to yawn yourself, even if you are not sleepy. Reference: [10]

Animation Techniques and Emotional Expression: Having discussed theory of emotion, we can then extrapolate them onto animation, by making use of different animation techniques that empathise or communicate certain emotions. These techniques make use of a set of rules that help add personality(thus, emotion) to any animation,

they are referred to as the principles of animation, which we will now discuss in more detail, as they are the backbone of conveying personality, emotion and overall quality in animation.

3.2.2 The 12 principles of animation

As previously stated, the principles of animation are the bread and butter for anyone working in the field of animation, as they must be always thinking about when and how to apply them. They consist of 12 basic principles, originally established in their 1981 book "*The Illusion of Life: Disney Animation*" by Disney animators Frank Thomas and Ollie Johnston [11] in their quest to produce realistic an grounded animation, and it is based on the work produced in the studio from 1930s onward, yet they are to this day applicable to any medium. Although there could be a whole thesis based on them, I'll roughly explain them and how they relate to the topic at hand, that is conveying emotion and portraying personality:

1.Squash and Stretch: This principle is all about giving weight and flexibility to an object, as it refers to how an object can be squashed and stretched by applying motion or a force to it, giving that object a distinct behaviour. The degree of this effect may vary, and it can also be used to convey emotion, for a character might stretch with surprise, or squash/shrink with fear or disappointment. The most basic example that Frank and Ollie give, as well as many other animators, is the bouncing ball 3.4, as one can see very clearly how the ball feels different (heavier and rigid or light and bouncy) when changing these properties.

2.Anticipation: Anticipation is when a character prepares for an action in order to provide the audience with a clue as to what is happening next, as well as to make that action appear more realistic. One great example of this would be the jump: in order to jump high, before leaping into the air a character has to prepare for the action by crouching to build up the energy needed, just like a spring that coils right before releasing. By visually foreshadowing actions, we can also help evoke an emotional response, such as suspense or excitement.

3.Staging: Staging basically refers to the presentation of any idea so that it is portrayed in a manner that is completely and unmistakably clear. This definition can be a bit vague, as it refers to many areas of animation, such as acting, timing or setting. That means that it can be applied in order to control where the audience is looking at any given point, but also be used to convey ideas. If a character is sad you want to make sure that the viewer knows that and feels that, and that can be archived throughout many ways, like with the character's performance. This ties up with the previous points talking about psychological perspectives and motion and kinesthetic empathy, and one can go over the top to ensure giving the message or be subtle about it.

4.Straight ahead and pose to pose: This principle has little to do with the topic at hand, as it relates to the method of animation used. The first method, "Straight ahead" is when one draws the first drawing, then the next and then the third and so on,

basically animating as you go. The second method “pose to pose” is where you draw the beginning and the end of each and every main pose, and later fill in the blank spaces in between. This last process is generally called ‘inbetweening’. In character animation, and specially when working digitally, the method used is “pose to pose”. This makes sense, as computers are able to interpolate character poses, thus making that inbetweening much easier.

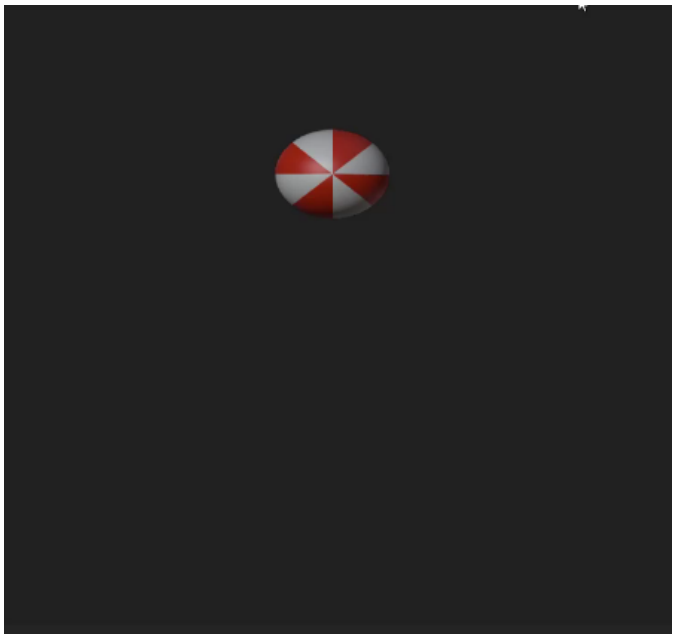


Figure 3.4: Bouncing ball, timing and squash and stretch

5.Follow Through and Overlapping Action: This is the technique of having body parts or appendages being dragged behind the rest of the main body, and continue to move as the body stops, like the movement of a tail. They serve to add an extra oomph of realism onto a character through subtle movement.

6.Slow In and Slow Out: This principle talks about the speeding up and slowing down of the action that occurs naturally and archives a lifelike motion. Without it, motion at a constant speed feels more mechanical-like. In 3D software, adding this slow in and slow out is a matter of just changing the motion paths from linear to a spline curve.

7.Arcs: Most natural actions tend to follow an arched trajectory, and this principle states that by following certain natural “arcs”, a greater realism is archived. In characters it is always applied and taken into consideration when rotating a limb by its joint and on inanimate objects, parabolic trajectories are also very common because of gravity. The shape of these arcs can also influence the viewer’s perception of motion within an animation. While smooth, wide arcs might convey a relaxed, joyful or prideful emotion, tighter, more jagged arcs can achieve the opposite effect.

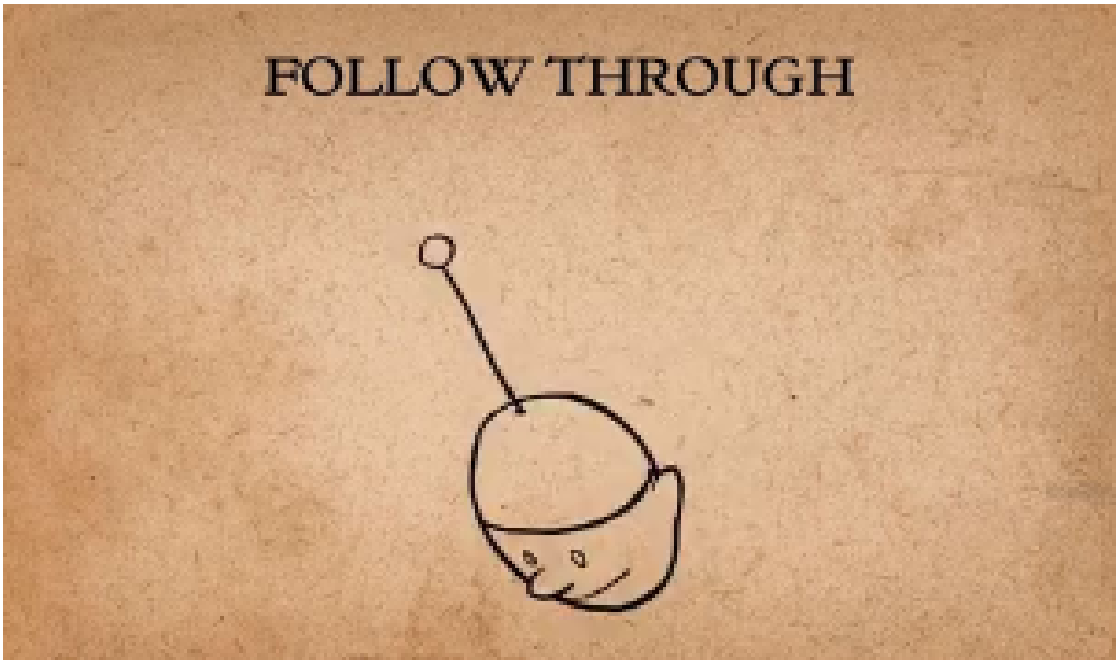


Figure 3.5: Follow through, by Alan Becker

8.Secondary Action: This principle, often associated with overlapping action, refers to all of the actions or gestures that help support or emphasise the main action in order to add an extra dimension to a character's animation. Reinforcing an action that conveys any emotion via the use of this secondary motion also reinforces said emotion, even if this secondary motion is very subtle, like a nervous tic on an agitated character. As in every art form, the devil is in the details.

9.Timing: Timing states that the personality and nature of an animation is greatly affected by the number of frames inserted between each main action. In other words, the more frames inserted between two main poses the slower the animation will result, therefore timing refers to the speed of the action. Careful timing can also influence the perception of an action. For example, a walking cycle can evoke a more sad, heavy walk, if the timing is slow or a more lively mood with a quicker timing.

10.Exaggeration: As the name suggests, exaggeration means to take every action, pose, and expression or even a personality trait to the next level, in order to increase the amount of impact to the viewer. Basically, if a character is sad, to make him sadder, if he is wild, make it wilder and so on. It is important to note that exaggeration does not mean more distorted but more convincing instead, although in quick motions, if exaggerated motion is to be applied it needs to be more extreme in order to be noticed, resulting in frames that might seem too extreme if the animation is paused.

(a) Anticipation Principle, by Alan Becker

(b) Exaggeration Principle, by Alan Becker

11.Solid Drawing: This principle ties intimately with maintaining the laws of physics of the world that one is working in, unless it is imperative to deviate from them. This means that one has to keep in mind the weight, balance and three-dimensionality of animated objects. Traditional animation is more involved with this principle, as 3D

software already makes it so that one does not have to worry so much about keeping consistent three-dimensionality, but it applies to other aspects in 3D animation, such as the so called “twinning”. That is when, for example, paired features of a character such as their legs are doing the exact same thing, disbalancing the weight distribution and thus making that action seen off.

12.Appeal: Last but not least, appeal tells us that the characters that one animates should be somewhat pleasing to the eye, either by their unique personality, distinctive or out of the box movement, or a rich psychological background. Anything that makes a certain character interesting. This one is a highly subjective topic, as everyone has a different standard as to what is or is not appealing . However, certain traits like a dynamic design or emphasising an aspect of a character's personality through motion can greatly improve the outlook of a character.

To summarise, these 12 principles of animation are essential in order to provide a framework in which animators can create more realistic, appealing, and emotional animations. By understanding these principles and applying them correctly, animators can better express emotions through their work.

3.3 Case Study

In order to take these theoretical concepts into perspective, it would be appropriate to take a look at how established and mainstream pieces of media make use of them, analysing the way they use motion in order to convey emotions or describe character personality. I will talk about *Kung Fu Panda*(2008), and character animation for *League of Legends* (2009-2023), as not only they remain relevant to this day, but also can point out many of the things already discussed, from different perspectives, as video game animation and feature film animation have different objectives.

3.3.1 Background

Kung Fu Panda is a globally acclaimed computer-animated film produced by DreamWorks Animation. The film premiered in 2008 and has since been followed by multiple sequels due to its widespread success. The narrative centers on Po, an endearing and somewhat clumsy panda who adores Kung Fu, as he embarks on a journey of self-discovery and growth. The film's animation style masterfully blends traditional Chinese art elements with cutting-edge animation techniques to create an engaging, humorous, and often touching narrative that resonates with viewers of all ages. *League of Legends* is an immensely popular multiplayer online battle arena (MOBA) video game developed and published by Riot Games. Since its release in 2009, it has become a standout title in the world of esports and online gaming, boasting millions of players worldwide. Players choose from a roster of "champions," each with unique abilities, characteristics, and backstories. The rich lore, competitive gameplay, and diverse character design, coupled with the detailed and unique animations for each champion, create a dynamic and immersive gaming experience. The animation in League of Legends is essential, not only

for the functionality of the game but also for expressing the distinct personalities and emotional responses of the characters.

3.3.2 Theory Application

Let's start by taking a look at Kung Fu Panda. In order to properly contextualise what its animation archives in terms of evoking personality and emotion, we must take into consideration that because it is a feature film produced by DreamWorks, a lot of time and money was put into it, with around 4.5 years of production and a budget of 130 million dollars. Furthermore, as an animation of this kind of scale, a polished and well rounded animation is given for granted. Keeping this in mind, we can take a look at how the motion and behaviour of some characters in specific scenes influence the emotional responses of the audience. We can see throughout the course of the movie, how the main character expresses his feelings through his actions and how his way of acting evolves by the end of the movie. At the beginning of the movie, when he is awakened by his father to work in the noodle shop, we can see how the low ceiling of the shop that his father walks so confidently in is just not able for Po to comfortably stand, forcing him to crouch a little and thus cementing the idea that he is uncomfortable with his current situation. Even though this idea is also enforced by the dialog and overall narrative, this display of character motion helps to round up the scene.

Another particularly good scene to analyse could be Po's training montage with Master Shifu. This scene serves to show the change in Po's demeanour as his initially awkward and amateur movements that make him seem like a novice, gradually evolve into more fluid and confident ones as his skills and self esteem grow, mirroring the change that he makes throughout the whole movie. This sequence elicits an emotional response in the audience that correlates with Po's evolution, ranging from sympathy and frustration to hope and triumph, as they emphasise with the main character. One last very interesting element to take from this movie is the way in which character motion matches both personality and design. For example master Oogway, being a tortoise moves in a calm and soothing manner without unnecessary movements and yet as he moves, one can see that he is shivering, probably from old age. This shiver is a perfect example of adding secondary action in an animation, giving a new layer that contributes to make it better, even if only in a subtle manner. Another great example of personality and design is how the martial arts team known as "the furious 5" is shown, as each one of their members portray a very distinct form of martial arts, and that alone already differentiates them from each other and gives them their own appeal. For example, Mantis moves very quickly from one point to another, jumping and stretching itself to a point that it looks like a green blur on the screen whereas Tigress makes more decisive and powerful strikes that serve to show her prowess. Interestingly, these fighting styles are actually based in traditional martial arts stances that share the animal's names, and I find it very poetic how the animators have managed to translate these martial arts styles onto animal characters, as initially they were conceived by humans taking inspiration from those animals in nature.

Figure 3.7: "*Kung fu panda*":Body Language in the theory of Emotion

Figure 3.8: "*Kung fu panda*":Body Language in the theory of Emotion

Now let's take a look at League Of Legends character animations, and how they differ from that of a movie. First of all, it is important to know that, even if video game animation and animated movies have a lot in common, as they share their principles, they each serve very different purposes. On one hand, animation in movies is a key factor, as many aspects of the movie relies on a well rounded animation and therefore revolve around it, from camera movement to even the script in some cases. That's why staging takes a specially important role in this type of animated media. On the other hand, animation in video games, although it is an important part of the game, is one of the aspects of the game that work together in order to archive the final product, like gameplay, character design or the narrative. You can say that the animation serves the game, and not the other way around. That is why in video games it is crucial to develop animations that work with what the game is trying to archive and its style. Now, given the nature of League Of Legends, a MOBA, in which there is a catalogue of various characters to choose from, each one with its distinct background and style, it is imperative for the animation of said character to differentiate itself from the others, whilst maintaining its personality and adhering to the overall style of the game. That is the main reason why this game is interesting from the perspective of portraying character

personality and emotion in video games. Let's look at some examples: For instance, let's consider the champion Fiddlesticks, known as "The ancient Fear". He is depicted as a ghastly, ancient demon that has taken over the body of a scarecrow and it is clear by both his character design and animations that he is meant to be this scary monster that haunts its enemies. Fiddlestick's animations help convey this premise by using this completely unnatural jittery or shaking that makes it look so that something is off about the character. This is obviously made on purpose, and pairing it with all the clawing and swinging of its multiple arms makes an effective impression. I personally have been jumpscared by Fiddlesticks more than once. One character of the game that showcases the principles of animation is also one of the characters that players get to play first. The champion is known as Garen "The Might of Demacia". He is a tall humanoid man that wears a heavy armour and carries a seemingly heavy sword. Most of his animations emphasise this "heaviness" aspect of his, often portraying weight by adding little to no squash and stretch, and timing the animation so that each step of his seem powerful, thus making the player feel he's caring a tank-like character. His animations (more specifically his shoulder plates) also often make use of an advanced animation technique called space switching, which is intimately correlated with the principle of follow through and overlapping action. It is noteworthy to add that, given that this game has an aerial perspective, its animations are made so that they are the most recognizable from said perspective, as they would otherwise confuse the players. And readable animations are something primordial in any game, but mostly in a MOBA just like this example, as players often have to rely on animations in order to see what their characters and others are about to do. Because of that, many animations are often very exaggerated and have a recognisable or flashy start in order to cue the plates that an specific action is about to happen.

Figure 3.9: Fiddlesticks animation, by Riot Games, form teemo.gg

Figure 3.10: Garen's running animation, by Riot Games, form teemo.gg

3.3.3 Conclusion

This section has served to show three main concepts: How all animations are principled in common concepts such as the principles of animation, how they depend on the context

in which they are made, having different functionalities within different pieces of media and and how the theory of emotion is applied through the use of motion and behaviour to evoke specific emotional responses and create a deeper connection with the characters

3.4 Digital 3D Animation Types

As an industry that's constantly evolving, especially throughout the last couple decades, the 3D animated space has seen many improvements that have radically changed the workflow and manner that professionals approach animation. Nowadays, many tools and techniques have been developed in order to solve specific scenarios, as is the case with procedural animation. However when it comes to character animation, specifically for video games, we find that mostly two methods are being used, those being Keyframe animation and Motion Capture. In this section we will discuss how they work as well as compare their advantages and disadvantages when it comes to character animation portrayal

3.4.1 Keyframe Animation

This widely used technique consists in changing the position, rotation and scale within a 3D model in a timeline. The animator manually creates certain frames, called 'keyframes' being those that define the most important poses of a given motion, these poses will later be interpolated by the computer in order to create the desired effect. As 3D characters are usually complex, with many moving parts as are humanoid characters, a system must be put in place in order to easily convey the motion of the joints to properly manipulate and pose the model. That system is called a 'rig', and it sometimes is as complex of a procedure as animating so it's no surprise that rigging is in of itself a whole profession. When it comes to creating an animation with this method, there are many considerations to be made. First and foremost is the fact that this is generally a really time consuming process and it requires a considerable amount of experience to pull off a decent result. Therefore having a good grasp of animation principles such as timing, spacing, anticipation or squash and stretch is essential. The animator has basically absolute control over the model, allowing for both really stylized and beautiful animations and the complete opposite. Keyframe animation is widely used, especially in feature films, TV shows, and video games.

3.4.2 Motion Capture

On the other hand, Motion capture (or 'mocap' for short) is a technique in which an actor has to wear special clothing with sensors that allow for tracking the movement of their bodies via cameras or other methods. That movement that the actor realises is then converted into data that can be applied into a 3D character model, conveying that actor's movements. This is great for some things, as it allows for a realistic and grounded movement with much more ease than with keyframe animation alone. It is also quite time efficient, as with mocap animation there is no rigging to be made, as

the sensors on the suit do that job already. However it is also necessary to take into account that the expertise of the actor's plays a very important role if you want to make engaging animation, and that there are many nuances of an animation that just can't be achieved by using this kind of method.

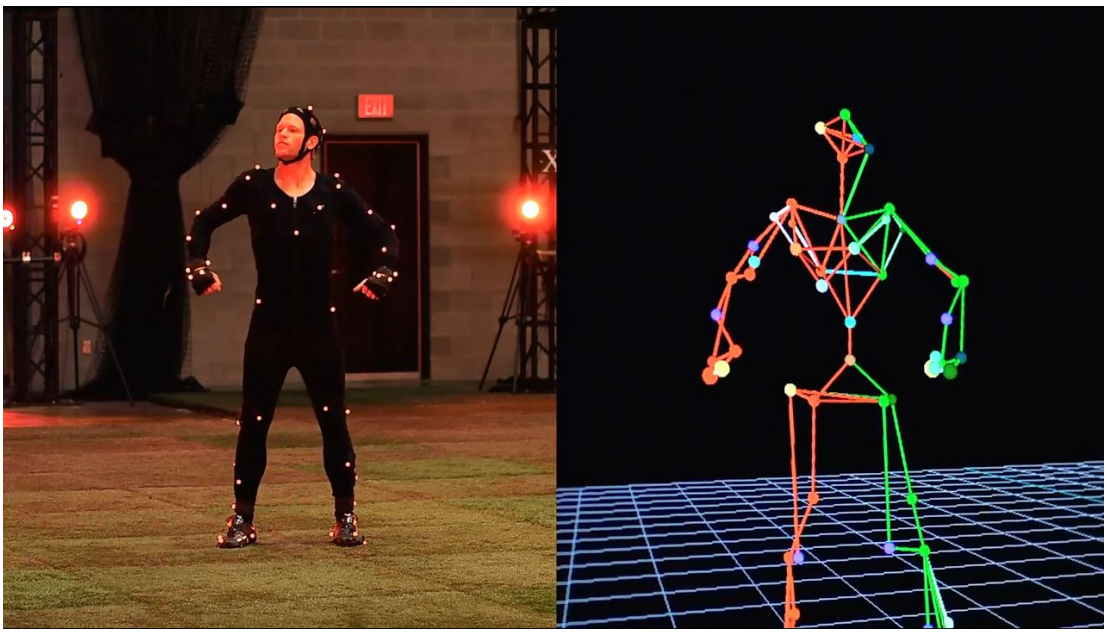


Figure 3.11: Motion capture technique

3.4.3 Technique comparison

As previously mentioned, both of these systems have their own strengths and weaknesses, and choosing the right option may depend on various factors, such as your initial resources or the target result. While it's true that keyframe animation is widely used and easily available, there's a really high 'skill ceiling' in which one has to dedicate a lot of time to get good. It is also a very time consuming method, as every aspect of the model has to be taken into account, as well as animation principles, physics or the character personality, among others. In the other part, motion capture allows for a quicker result with a high level of realism, as every subtle movement of the actor can be traced. That is why it is mostly used when working in large-scale projects or when realism of the movement is primordial, to give an example EA sports uses this technology when animating sportsmen, as is the case in their FIFA games. However, not everyone can afford a professional mocap setup as it is usually very expensive, and its results are sometimes a little glitchy so they require post-processing. Overall, both of these methods are very valid depending on the objective. Such is the case that many studios work with both types at the same time, using mocap for the base movement and then polishing and exaggerating that movement via keyframe animation. In my case, I

will be using keyframe animation, as my final objective is to create more of a stylized type of animation that should be very hard to achieve with mocap. There is also the problem of finding specialised equipment, which I couldn't possibly solve.

3.5 3D Animation software

In this section we are going to take a look into the current software for making animations, overviewing their capabilities, user friendly-ness and prices. We are going to focus specifically on 3D oriented software, as there are many other programs that allow for 2D animation.

Maya

When it comes to animation, Maya is the leading piece of software, making the industry-standard. Although it boasts of many features, including modeling, simulation and rendering, its strongest part is undoubtedly animation, and thus it is the go-to software for most animation and video game companies. Its comprehensive set of tools and features allow for a very efficient working pipeline. These are some of its main features:

- Native motion capture plugin
- Animation Bookmarks
- Arnold GPU/CPU

This piece of software can be quite complex and daunting for beginners, due to its extensive toolset. It usually requires significant time in order to get used to its layout and workflow, and much more time to actually master it. Price: As of today, although Maya is free if you are enrolled in a 3D related course, and although it has a limited free trial, the price to acquire the software is subscription based, and you can choose from from 235€ a month, 1,875€ yearly, or 5,625€ every three years.

Blender

Blender is a free, open source 3D animation suite that is able to support the whole animation pipeline (modeling, rigging, animation, simulation rendering...). It boasts of a very active community that works to improve on the software, resulting in many patches throughout the year that improve the program's efficiency and toolset. These are some of the features that have been added to blender in the last few years

- 2D animation (Grease pencil)
- Geometry nodes for animation purposes

- Storyboarding tools

When it comes to user-friendliness, Blender has different opinions. Some people think that the sheer number of tools are overwhelming and that the interface is confusing. Others say that being a free program it is a very powerful tool that opens the door to the 3D industry to everyone. I think that this last statement holds true, and that with recent and upcoming changes, it has the potential to break into the industry standard, because as of today it is not usually a part of production pipelines in studios, and usually leans more towards independent creator's works

3Ds Max

Even though in terms of animation, Blender and Maya are the most widely used programs, Autodesk 3Ds Max is not much behind. 3Ds Max is able to model, animate, rig and add particle effects among others, although it excels in terms of modeling and rendering. Many animation tools are available within the program, just like Maya, as they both are part of the Autodesk package. The main difference between the two is the process flow, ease of use (it is harder to animate characters in comparison to Maya), and number of tools available. Price: As of today, 3Ds Max is a subscription based software with a free trial and an exemption for students. The price options you can choose are as follow: 279€ monthly, 2,245€ yearly, or 6,734€ every three years.

Cinema 4D

Cinema 4D is a professional 3D modeling, animation, simulation and rendering software, widely known for its strong capabilities in motion graphics creation. Its main strengths are that it is one of the most user-friendly 3D animation software on the market, with an intuitive interface while still maintaining a fair amount of features, and its quick workflow. It also offers robust features for creating high-quality 3D images and animations and integrates well with After Effects, making it particularly appealing for motion graphics. Price: There are three plans, that include a full copy of the program to choose from, teams, individual and perpetual:

- Individual: Monthly: 59.91€ per month Annually: 719€ per year
- Teams: Annually: 949€ per year
- Perpetual: 3495€ as a one-time payment

Figure 3.12: Most used animation software logos

3.5.1 Why Blender?

Being a Student at UJI, I would have been able to gain free access to Autodesk software in order to develop this project, so why did I choose Blender instead? Well, there are some reasons for that. First of all, familiarity has played a huge role in deciding which software to work with. The learning curve for Autodesk products is really steep and I presumed that learning just the basics would take many hours. Furthermore, I already had some experience with Blender, having worked with it on *vj1212 - expresión gráfica*, and on standalone projects. Although it is true that nowadays Maya is the Industry standard in this field, I believe that in recent years, Blender has gone through an immense development, without seemingly stopping, and every day I see more and more animators (specially freelancers or on indie studios) having Blender as their preferred animation software. Therefore, I believe that if this trend holds, Blender can one day become Industry-standard, alongside Maya.

Chapter 4

Developing animations for characters on Blender

Contents

This chapter is meant to show the pipeline of making humanoid animations within blender, showing my process whilst explaining which steps animators usually follow in a professional environment and showcasing Blenders tools and interface for animation.

4.1 Preparation

There are a few steps we must take before we start animating our characters, these are finding a model to animate, rigging said model and developing a keying set for it.

Getting a model: When developing animations for humanoid characters, although the animation data can be transferred to one another (as they share their skeletal/bone hierarchy), we must first find a 3D model that can fit said hierarchy to animate from. There are several ways to get a model like this, and one can either model it themselves (although it requires extensive knowledge in 3D modeling), buy it from someone online (many freelancers will develop your custom character) or download a humanoid character from a specialised web. As I didn't need any specific character, and was only looking for the resulting animation data, I decided to download a humanoid testing character from Mixamo. Mixamo is a 3D computer graphics company with online services that provide

downloadable 3D humanoid models and small interchangeable animation sequences for them, made via motion capture. I downloaded a test character, Y-bot without any animation. This model came with its own materials and its bone hierarchy although it was not yet rigged.

Getting a rig: The process of rigging can be described as the process of creating a bone structure onto a 3D model in order to be able to deform and animate it. Now, we already have a "rig" as moving the bones of the model we've downloaded already deforms it, but this is highly inefficient if our intention is to animate complex motion. Therefore, a better, more intuitive rig is needed. Luckily, there exists an add-on for Blender that automatically creates an effective rig from the Mixamo model that has many advantages when animating, such as using inverse kinematics on the extremities, providing intuitive controls or 'handles' to easily pose your character, and adjusting/correcting the bone hierarchy. This rig is referred to as 'Control Rig'.

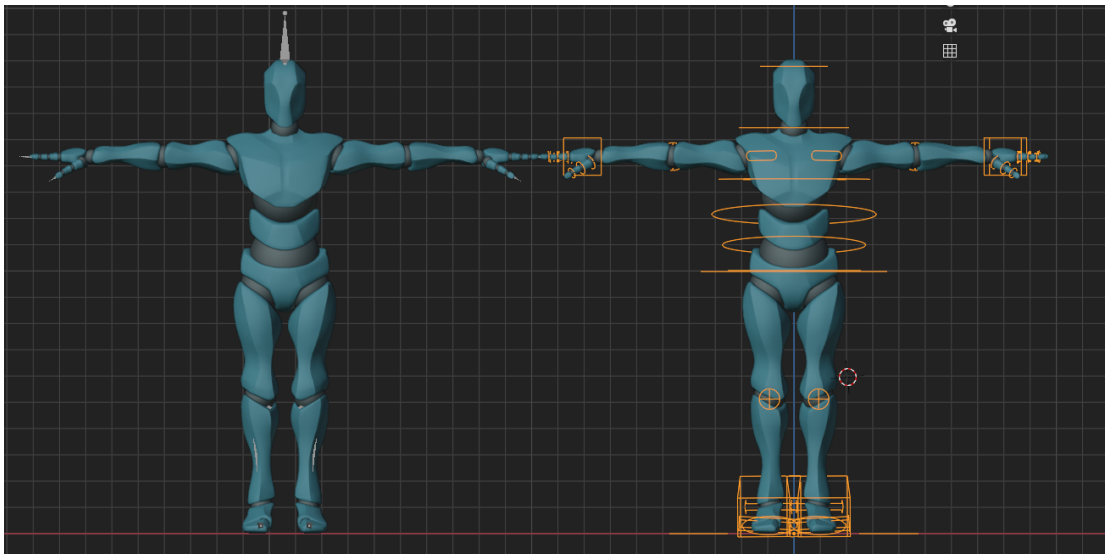


Figure 4.1: Character with bones (left) vs character with control Rig (right)

Developing a keying set: Once we have the Control Rig ready, we could already start animating, but right now every bone on every keyframe will have its x,y and z values of their position, rotation and scale registered, even though they might never need to use them. For example, the model's head can be rotated in order to look around and maybe scaled if one wants to archive squash and stretch at some point, but it will never separate from the shoulders and therefore its position won't change. We can apply this logic to every bone in the rig, locking the parameters that you know for sure that won't change, thus making the keying set. What this archive is, when creating or modifying a keyframe, locked values won't show on the various editors, thus resulting in a more clear and readable interface.

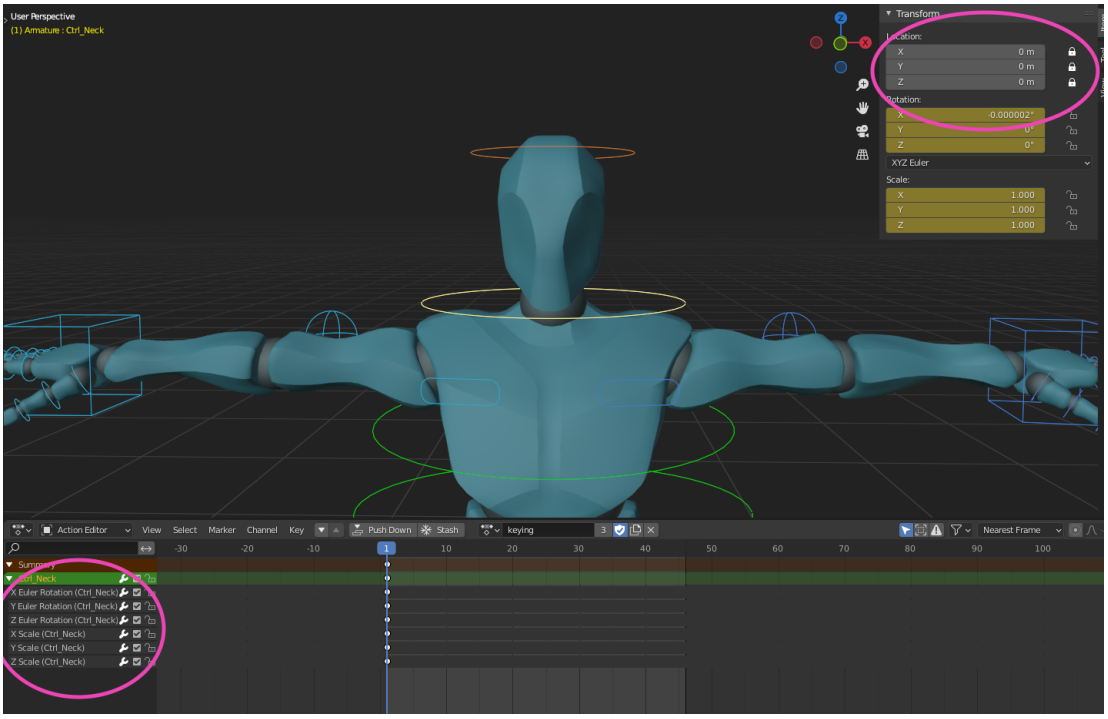


Figure 4.2: Keying set for the neck bone (position locked)

4.2 Blocking

Once all the preparations are done, we can finally start animating. Generally animators separate this process in two sub-processes, in the first one they build the basics of the animation, getting down the main poses. This stage is referred to as “blocking”. The next step is to fill in the gaps between those main poses in order to finish the animation, and that is referred to as “inbetweening” in traditional animation or “polishing” in 3D.

I like to make the comparison between animation and drawing, as it makes it easier for me to understand the blocking stage. When making a drawing, let’s say a pose, artists start by laying out a rough sketch of the final picture, getting down the proportions, shape and outline; while often using some kind of reference. The blocking stage has the same idea. The goal here is to lay down the most important poses,called ‘extremes’ or ‘extreme keyframes’ taking into account the timing of the overall animation. This is vital, as changing the overall timing further on will only get more difficult as the animation gets polished.

When starting to work on an animation, nearly all animators will tell you to work your poses from a reference. You can either find a reference online of what you are looking for, or just record yourself making that desired action. That’s the reason you see animators in their studios jumping, punching the air and making outright weird stuff. It may sound weird, but this might just be the single most useful piece of advice that I have come across, as I personally went to the park and took some references, which

ended up making those animations much easier.

Let's now take a look at the most important aspects of Blender's Interface related to the Blocking stage. We will generally work within three different windows: the 3D Viewport, the Action Editor, and the Timeline Editor. In the Viewport, we will modify our rig in order to create the desired poses, to then insert a keyframe onto the timeline/action editor. The timeline is used for animation playback, setting the frame count of what will be animated and simple keyframe management whilst the action editor will be in charge of visualising and managing all keyframes for an object or a selected group of objects. You can edit, duplicate or scale a set of keyframes of an action, among other usages. You can also switch between different actions or animations for one object (one character might have an action for running, another for jumping and so on).

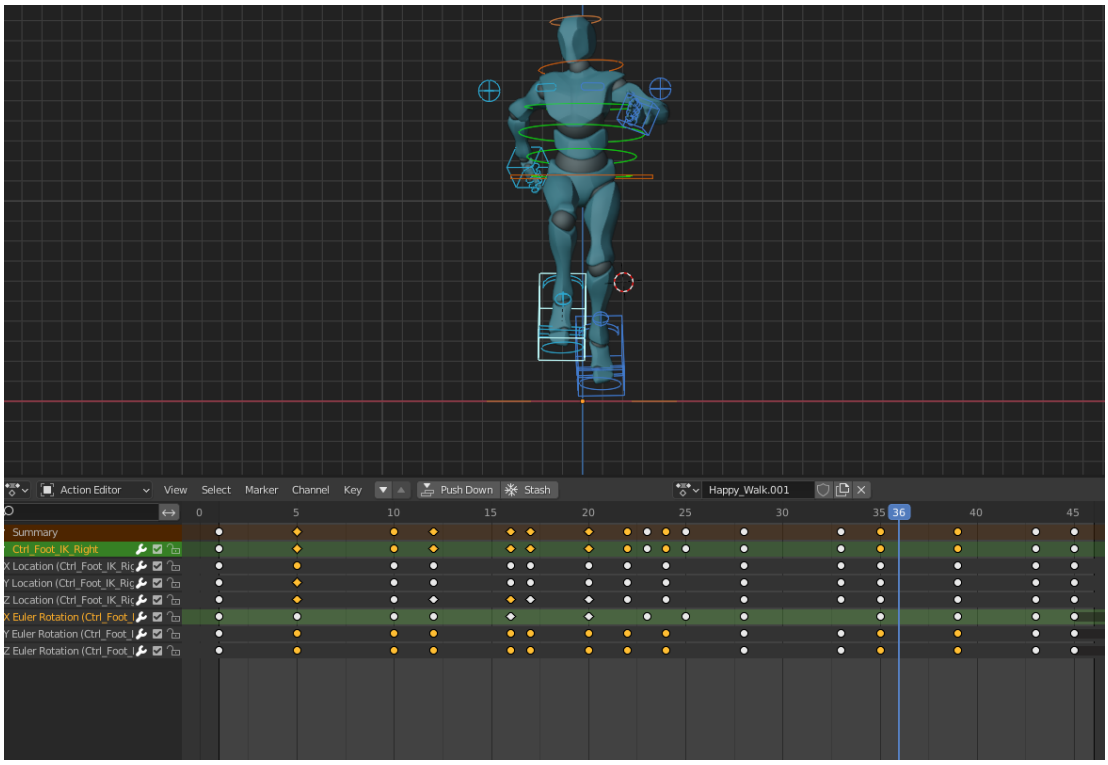


Figure 4.3: Interface for the Action Editor (bottom)

These interfaces contain many shortcuts that the user can do, here I'll put some of the most important and time saving:

- **Viewport**(pose mode)
  - Alt + G/R/S: Reset position rotation and scale on an object.
  - Ctrl + C / Ctrl + Shift + V: Copies the pose of a bone or a group of bones, and paste that pose flipped on the X axis. Useful for mirroring poses or extremities.

  - I: Insert keyframe menu.

- **Action Editor**
  - Click selection, box selection.
  - Shift +D: Duplicate keyframe.
  - X: Delete keyframe.

- **Timeline**
  - Right/Left arrows: Go forward/backward one frame.
  - Up/Down arrow: Go to next/previous keyframe.
  - P + selection: Preview the selected part of the timeline.

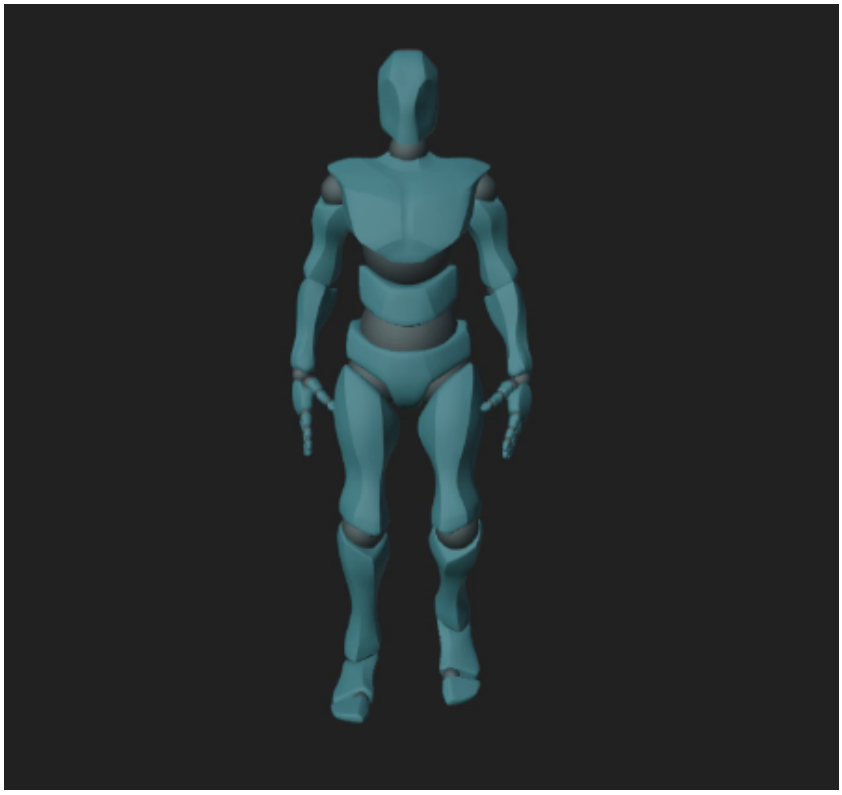


Figure 4.4: Blocking stage of a normal walk

4.3 Polishing

If the Blocking stage is the backbone of an animation, the polishing stage is its soul. The purpose of it is to connect the key poses made in the blocking stage, while adding depth

and personality to the animation. Going back to the drawing analogy, the polishing stage would be equivalent to giving volume, adding details or shading. Similar to drawing, there is no "end" to the polishing stage, as the animation is done only when the animator is satisfied, and usually the more you look at the screen the more mistakes you will find. That is why many animators state that there is no perfect animation.

The first step to the polishing process is the "splining". This term stands for the type of interpolation between the frames, as in the blocking stage the interpolation curve is either linear or constant. Having a spline-like curve adds an ease in and ease out into the key frames, but you have to be careful, as not all movement is meant to behave like this. For example, a ball falling and hitting the ground will not need an ease in, as that would result in an unnatural movement.

It is on this stage that one has to keep in mind the principles of animation, as many apply here, for example giving details through secondary action, checking the arcs in the movement, applying slow in and slow out though the animation curves or adding appeal onto the action.

Let's talk about which windows of Blender's interface are related to the polishing stage. Here we find two main windows, the Viewport and the Graph Editor. The Viewport will be used to manually tweak keyframes on occasion, but it will mostly serve as a visual aid. It is in the Graph Editor that most of the work will be done, by modifying the animation curves displayed in it.

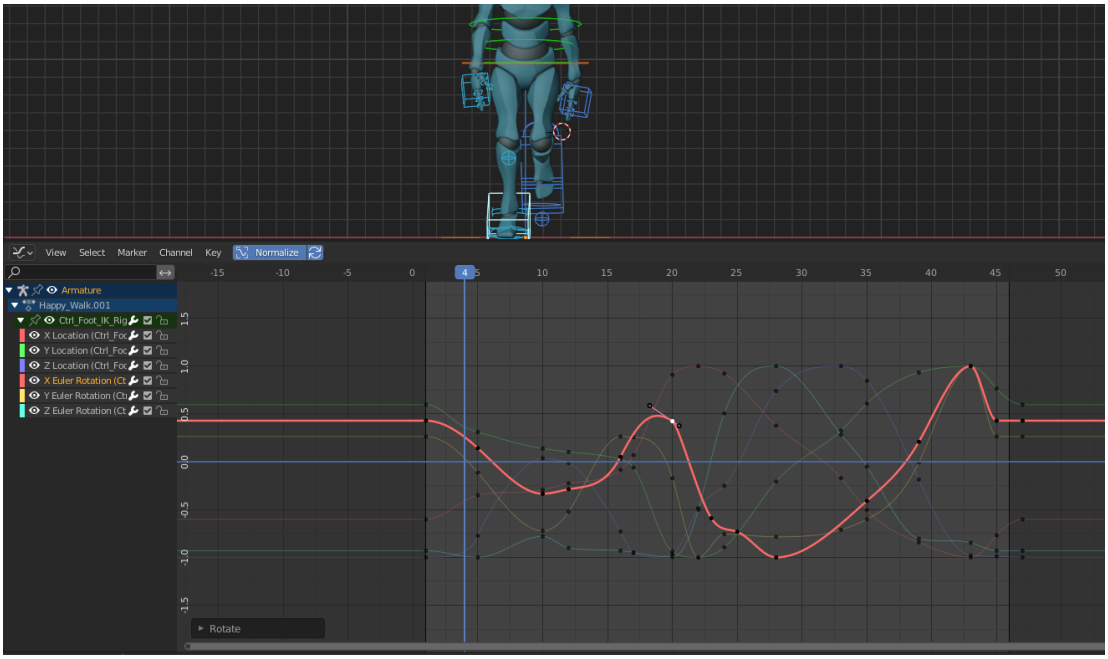
Figure 4.5: Splining of the same normal walk

The Graph Editor and Viewport also contain interesting shortcuts as well as powerful visualisation tools, here are some of them:

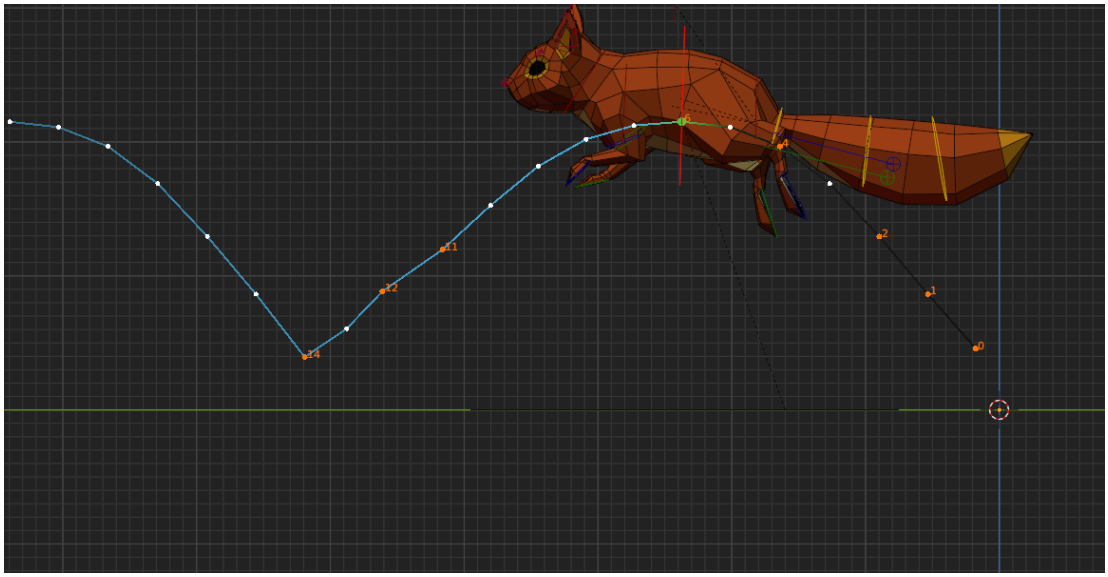
- Motion Paths: This tool allows the user to visualise the path of a point or a bone throughout a series of frames. It is extremely useful for seeing the arc that a join does, as well as its timing.
- Graph Editor shortcuts:
  - T : Change interpolation mode from selected objects
  - V: Modify the handle type of the curves
  - Shift +E: Add an extrapolation mode. Very useful for looping animations indefinitely.
  - Shift + Ctrl + M: Adds an f-curve modifier. Useful to change the shape of the curve, like adding noise to it.

4.4 Exporting to Unity

When the animation is done, we can extract its data by exporting a .fbx file of the character that holds said animation. It is important to set the values in a specific way so the world coordinates of Blender and Unity align, as they are different. For the purpose of this project, I have also made sure that the root motion is disabled, as my intention was to show the animations in place. Once imported in Unity, you can extract the animation data, and use it interchangeably with any humanoid Unity model. For a more thorough dive on this topic, refer to A



(a) Interface for the Graph Editor



(b) A squirrel model's body motion path

CHAPTER 5

System Analysis and Design

Contents

This chapter presents the requirements analysis, design and architecture of a Unity Add-on that provides an interface that allows for a quick visualization of animations in humanoid characters organizing those animations via folders, each one of them representing certain type of personality.

5.1 Requirement Analysis

In order to carry out this task, it is of utmost importance to preventively analyze its functional and non-functional requirements that must be carried out.

5.1.1 Functional Requirements

Here I will list the functional requirements, which are those that will define a function of the Add-on that is going to be developed. These are the main functional requirements for this project:

R1: Animation Management. The Add-on needs to provide an efficient system for properly visualising humanoid animations within the Unity Editor, in a manner that the user is allowed to add, remove and edit animations.

R2: Automatic animation file sorting and recognition. This tool should also automatically detect and sort the animation clips present in the project, and be able to organise them in the way that the user desires.

R3: Animation Playback. It should provide the options for playing, stopping and controlling a given animation.

R4: Inspector integration. It also should be integrated within Unity's inspector in order to provide a user-friendly interface for managing animation parameters and clip selections.

R5: Compatibility and Exportation. Ensure that the Add-on is compatible with various Unity versions and that it can be easily exported and used in different projects.

R6: Character interchangeability: The Add-on should be able to work with any character that has a humanoid structure and it should allow for an agile change in the target character.

Input:	Choosing an animation to preview
Output:	Displayed Animation
When selecting an animation clip from the Add-on interface, there needs to be an automatic update on the animation displayed on screen. Changing the selected animation clip with another of the same or different category will have the same effect.	

Table 5.1: Functional requirement «R1. Animation Management»

Input:	The user adds or removes an animation clip or a folder category
Output:	The resulting structure is updated
When the user modifies the folder structure within the Unity asset system, by adding/removing animation clips or category folders, the Add-on mus register it and show the correct undated structure. If there is any sort of error in this process, a warning mus be shown.	

Table 5.2: Functional requirement «R2. File sorting»

Input:	The user modifies a characteristic of a clip
Output:	The clip with the changed characteristic is displayed
When the user modifies one aspect of the currently playing animation clip, via the options that the interface provides, it should update said characteristic of the animation on run time. The aspects of an animation that can be modified are, pausing/resuming the clip, changing the clip's speed and enabling/disabling root motion.	

Table 5.3: Functional requirement «R3. Animation Playback »

Input:	The user wants to previsualize a clip
Output:	The Add-on lets him easily do so
By integrating the add-on onto Unity's inspector system, while also providing drop downs with your animation's folders, must make for an easy to use and intuitive tool for quickly previsualizing you animations onto different humanoid characters.	

Table 5.4: Functional requirement «R4. Adequate inspector integration»

Input:	The user wants to use the Add-on on different Unity projects
Output:	The user is capable to do it without major issues
The Add-on must be made in a way that ensures that it functions well in many Unity versions so that its usability is worth, and that it is easy to export and set up onto other projects.	

Table 5.5: Functional requirement «R5. Compatibility and Exportation»

Input:	The user wants to try the clips with another character
Output:	The character is properly added
It must be ensured that any Unity character that has a humanoid skeleton is able to properly showcase the animation clips that the Add-on manages, as well as to ensure that adding those characters onto the tool is done in a quick and easy manner.	

Table 5.6: Functional requirement «R6. Character interchangeability»

5.1.2 Non-functional Requirements

Non-functional requirements are the requirements that impose restrictions on design or implementation such as restrictions on design or quality standards. These are properties that my Add-on must have:

R7: Performance. I must ensure an efficient memory usage and optimized performance during the animation playback and management, as well as minimize any impact on the overall performance of the Unity Editor.

R8: User Experience. It is primordial to create a user-friendly and intuitive interface that is easy to navigate and understand.

R9: Scalability. It is important to design the Add-on in a modular and extensible manner to allow for future enhancements or additions.

R10: Reliability. I must ensure that the Add-on functions in a reliable manner, without major issues or bugs that may cause crashes or unexpected behaviour.

R11: Maintainability. The code in which the Add-on runs must be clean and well documented in order to enhance maintainability and help to develop further updates or modifications.

R12: Documentation. The project must be properly documented, with clear and easy instructions for its installation, setup, usage instructions and troubleshooting advice.

5.2 System Design

This section must present the (logical or operational) design of the system to be carried out. In order to portray the operational design that the Add-on will follow, I'll use a conceptual diagram 5.1 that showcase its functionality.

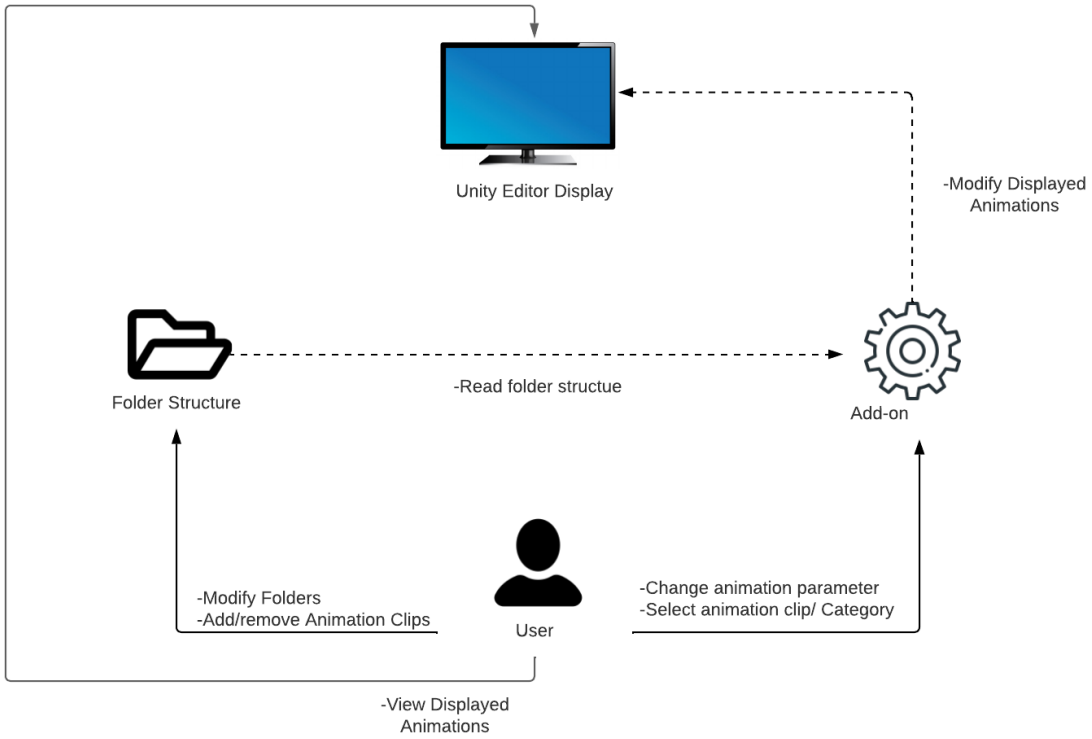


Figure 5.1: Conceptual diagram for the functionality

5.3 System Architecture

This section should describe the architecture of the projected system. In this case, the Add-on has been developed with Unity3D, specifically with the 2021.3.10f1 version. According to Unity's documentation [13], the minimum system requirements in order to properly use this engine's version are as follow:

- Operating System:
  - Windows: Windows 7 SP1+
  - mac OS: High Sierra10.13, Big Sur 11.0, +
  - Linux: Ubuntu 18.04+
- CPU: X64 architecture with SSE2
- GPU: DX10, DX11, and DX12- capable GPUs

Although these are the official requirements, Unity Documentation states that a system that satisfies all of these requirements might still not work on rare occasions. In my case, the computer used to develop the Add-on has these requirements:

- Operating System: Microsoft Windows 10 Home
- CPU: Intel(R) Core(TM) i7-7700K
- GPU: NVIDIA GeForce GTX 1050 1050 Ti
- RAM: 16GB

5.4 Interface Design

While very simple, this tool's interface design must be made in a way that is coherent and easy to understand and use, displaying all of the most relevant options either integrated onto the unity editor or in a floating window. In the Display screen, although there is a simple scenery to provide a relaxed ambient, the focus is made in the characters , displayed in the foreground. Here I'll show the design, made via the Editor class and GUIStyles 5.2 5.3.

Figure 5.2: Display Screen Layout

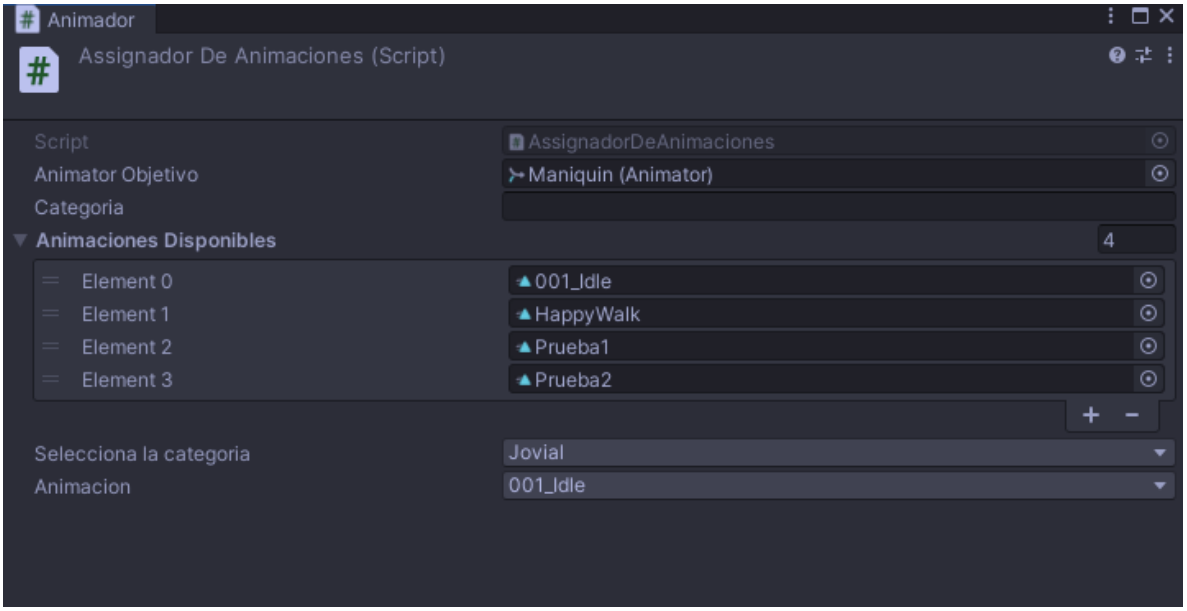


Figure 5.3: Tool interface

CHAPTER 6

Work Development and Results

Contents

Once the main idea of the Add-on has been laid out, with its requirements and functionalities that it must have, the time has come to develop said tool. In this chapter of the project, it's development will be shown, showcasing intermediate milestones and deviations from the initial plan, problems that have appeared along the way and the results obtained.

6.1 Work Development

In this section, the most relevant aspects of the proposed work will be explained ,differentiating the most relevant parts of it, along with the decisions that have been taken into account in order to contribute to the overall result. Most of the information that will be talked about in this section is available in the Unity documentation and also on YouTube videos.

6.1.1 Animator Assigner Script

The backbone of this tool, containing the functionality needed in order to manipulate the user's animations, is the *AsignadorDeAnimaciones* script. It essentially serves as a bridge between the animation in the user's asset folder and the Animator component in the scene.

```
29 referencias
public Animator AnimatorObjetivo//animator del personaje objetivo, solo coge avatares, o personajes humanoides
{
    get { return _animatorObjetivo; }
    set
    {
        // Para que el animator solo funcione con personajes humanoides
        if (value != null && value.avatar != null && value.avatar.isHuman)
        {
            _animatorObjetivo = value;
        }
    }
}
public Animator _animatorObjetivo;
```

Figure 6.1: AnimatorObjetivo property

Some variables are needed in order to store the information that we'll use later. First and foremost, we must reference the Animator component. This component is used for controlling the animations of a GameObject , and it works in conjunction with an Animation Controller (which defines a state machine of animations), and an Avatar, which defines how said animations are mapped onto a character model. Later, in the unity Editor, this property will allow the user to chose from an animator in the scene to start previsualizing the animations. In my particular case, I've coded it so the tool only recognizes an animator as valid if its avatar holds the 'isHuman' property, in order to make sure that we are working with a humanoid character, as seen in 6.1. Documentation for the inner workings of these properties can be found in [14] [15]. Other variables that are needed for the tool to function correctly, are a list of Animation Clips to store the animations stored in the Asset folders, transform variables to store the position of the characters and a string variable to store the selected category.

Among the script's core functions, these are the most important:

- **obtenerCategorias()**: This method, only used in the Editor, obtains all the directories within the Animations folder, and returns their names as a list of strings. Each of those names represents a category of animation. In order to be able to

```
1 referencia
public string[] obtenerCategorias()
{

    // Obtenemos la lista de categorias segun las carpetas del poryecto

    string assetFolderPath = "Assets";//carpeta assets
    string animFolderPath = "Assets/Animaciones";//carpeta animaciones

    //lista de directorios dentro de la carpeta animaciones
    string[] directorios = Directory.GetDirectories(animFolderPath, "*", SearchOption.TopDirectoryOnly);

    string[] categorias = directorios.Select(directory =>
    {
        string relativePath = directory.Replace(assetFolderPath, "").TrimStart('/');
        string nombreCategoria = Path.GetFileName(relativePath);
        return nombreCategoria;
    }).ToArray();

    return categorias;

}
```

Figure 6.2: obtener Categorias method

access the directory paths and transfer that information into the category list, it is necessary to use the System.LINQ namespace, which provides a set of query operators for working with data. For more information about this library, you can check out its guide [8].

- **GetAnimationsByCategory(string categoria)**: This method, also used in the Editor, finds all files within a category that end with '.anim' which correspond with animation clips, and loads those clips in a list.

```
//coge los clips por categoria de tu carpeta de Animaciones
2 referencias
public List<AnimationClip> GetAnimationsByCategory(string categoria)
{
    //Aqui obtendremos una lista de las animpaciones existentes embase a una categoria

    string animFolderPath = "Assets/Animaciones";//directorio carpeta animaciones
    string categoryFolderPath = Path.Combine(animFolderPath, categoria);

    //obtiene los archivos .anim de la carpeta de categoria seleccionada
    string[] animFiles = Directory.GetFiles(categoryFolderPath, "*.anim", SearchOption.TopDirectoryOnly);

    List<AnimationClip> animaciones = new List<AnimationClip>();
    foreach(string animFile in animFiles)
    {
        //convertimos la ruta absoluta del archivo de animacion en una ruta relativa al archivo Assets del proyecto
        string relativePath = animFile.Replace("\\", "/").Replace(Application.dataPath, "Assets");

        //Cargamos el archivo AnimationClip
        AnimationClip animationClip = AssetDatabase.LoadAssetAtPath<AnimationClip>(relativePath);

        if(animationClip != null)
        {
            animaciones.Add(animationClip);
        }
    }

    return animaciones;
}
```

Figure 6.3: Animations By Category Function

- **AsignarAnimacion(AnimationClip animationClip)/Crear Animator**: This method is an important part of the script, as it is the one responsible for creating an animator for managing when the user wants to change an animation. The first version was made using the 'AnimatorOverrideController' property on the AnimatorObjetivo's AnimatorController. and assigns that override controller back onto the 'AnimatorObjetivo'. This 'AnimatorOverrideController' is a type of 'RuntimeAnimatorController' that allows you to replace an existing Animator Controller's animations with new ones on the fly at runtime. What this achieves is to dynamically and efficiently change the animation of the 'AnimatorObjetivo' at runtime, based on the selected animation Clip. This function has been substituted by the **CrearAnimator** method in a new version of the add-on on which a new instance of an animator controller was updated every time that an animation Clip was selected, as it simplifies and provides a cleaner code without losing a lot of resources.

```
0 referencias
public void AsignarAnimacion(AnimationClip animationClip)
{
    // Crea un nuevo AnimatorOverrideController basado en el AnimatorController del objetivo
    AnimatorOverrideController overrideController = new AnimatorOverrideController(AnimatorObjetivo.runtimeAnimatorController);

    // Asigna la animación seleccionada al overrideController
    overrideController["NombreDelEstadoDeAnimacion"] = animationClip;

    // Aplica el overrideController al objetivo
    AnimatorObjetivo.runtimeAnimatorController = overrideController;
}
```

Figure 6.4: Version 1

```
3 referencias
private RuntimeAnimatorController CrearAnimator(AnimationClip clip)
{
    // Crear un Animator Controller en tiempo de ejecución con una sola animación
    AnimatorController controller = new AnimatorController();
    controller.AddLayer("Default");

    AnimatorStateMachine sm = controller.layers[0].stateMachine;
    AnimatorState state = sm.AddState(clip.name);
    state.motion = clip;

    return controller;
}
```

Figure 6.5: Final Version

6.1.2 Editor Script

The other script we will be taking a look at will be the editor script, which I have named *AsignadorDeAnimacionesEditor*. This script will contain the visual part of the add-on(its interface) and will also manage the possible interactions from the user, while inheriting the functionalities of the main script. Let's now take a look into its main functionalities. The script relies on the OnInspectorGUI() method, which is the function needed to create a custom editor within Unity Editor. It essentially modifies the editor inspector window for the objects on the scene with the aforementioned script. For more information on the functionalities of this method, go to [12]. I n order to create the elements that the user will be able to interact with, there are elements that can be created via the function EditorGUILayout, like popups, buttons and sliders. When the user interacts with said elements, if we reference the *AsignadorDeAnimaciones* script, we can the use its functions in order to provide the desired outcome.

```
//referencia a asignadordeanimaciones
AssignadorDeAnimaciones assignadorDeAnimaciones = (AssignadorDeAnimaciones)target;
```

Figure 6.6: Reference to the other Script

Lets take a look at how the layout is created and the reference script's functions are applied:

```
//lista desplegable de seleccionador de categorias
string[] categorias = assignadorDeAnimaciones.obtenerCategorias();
int nuevoIndiceCategoriaSeleccionada = EditorGUILayout.Popup("Selecciona la categoria",indiceCategoriaSeleccionada,categorias);


if (nuevoIndiceCategoriaSeleccionada != indiceCategoriaSeleccionada)
{
    indiceCategoriaSeleccionada = nuevoIndiceCategoriaSeleccionada;
    indiceAnimSeleccionada = 0;


}
//animaciones de categoria seleccionada
assignadorDeAnimaciones.animacionesDisponibles = assignadorDeAnimaciones.GetAnimationsByCategory(categorias[indiceCategoriaSeleccionada]);

//seleccion animacion
string[] nombesAnim = assignadorDeAnimaciones.animacionesDisponibles.ConvertAll(a => a.name).ToArray();
indiceAnimSeleccionada = EditorGUILayout.Popup("Animacion", indiceAnimSeleccionada, nombesAnim);

//asignar animacion seleccionada al animator
if((previousSelectedAnimationIndex != indiceAnimSeleccionada || previousSelectedCategoryIndex!= indiceCategoriaSeleccionada)&& assignadorDeAnimaciones.animacionesDisponibles != null && assignadorDeAnimaciones.animacionesDisponibles.Count > 0)
{
    //assignadorDeAnimaciones.AsignarAnimacion(assignadorDeAnimaciones.animacionesDisponibles[indiceAnimSeleccionada]);

    assignadorDeAnimaciones.AnimatorObjetivo.runtimeAnimatorController = CrearAnimator(assignadorDeAnimaciones.animacionesDisponibles[indiceAnimSeleccionada]);
    previousSelectedAnimationIndex = indiceAnimSeleccionada;
    previousSelectedCategoryIndex = indiceCategoriaSeleccionada;

    assignadorDeAnimaciones.SetInitialPositionAndRotation(initialPositions[instanceID], initialRotations[instanceID]);
    assignadorDeAnimaciones.ResetPositionAndRotation();
}
```

Figure 6.7: Caption

In figure 6.7, we can see how to create elements in in the interface, as well as how to properly callback the other script's functions to function with said interface elements. In this code, the list of categories in the assets is stored to then create a popup with the elements of said list. Later on, is saves the index of the currently selected category to display its animations, with another popup. Finally, when selecting an animation clip, it creates a replaces the current instance of the Animator with another with the new animation clip, so that the selected animation matches with the one displayed on screen. This constitutes the foundation of the add-on, but from this we can further expand its functionalities.

6.1.3 Adding Functionality

Once the basic functions are correctly working, it is time to start adding functionalities, with the final objective being to have a useful, yet simple tool which is easy to learn and use. The thing that I think would help while using the add-on and visualizing animations are, as stated in the previous chapter:

1. Buttons to the previous/next animation in a category.
2. Velocity control
3. Root motion activation/Deactivation
4. Reset position Button
5. Pause Button

Luckily, the first functionality is simple, as we can use the already existing data structures, like the category lists or the animation lists to easily increase or decrease the index of the animation displayed, making sure to not go out of the bounds of the lists.

In the case of both Velocity Control and Pausing the animator, I've thought of modifying the Animator's inherent *speed* property, setting in to 0 in order to pause the animation, and implementing slider to change this *speed* value.

A similar case appears when wanting to toggle on and off the root motion of an Animator, as there exists a property that applies or deactivates said root motion.

For resetting the position, the initial position and rotation must be registered , so I have saved the animator's value on the Awake function, so that it always keeps its initial value, and not a modified value that can appear if the root motion is activated.

The code for these implementations can be found in the Appendix C, which will contain the source code of the add-on's main scripts.

6.1.4 Stylizing

Right now, from a functionality point of view, the add-on is complete, although the interface is not very appealing, as it is very homogeneous.This can get in the user's way when using this tool, so I feel we should address the issue. Luckily, we can fix this via the UIToolkit package from Unity, which allows us to modify the default inspector by adding colour, space and form.

In here I'll show a basic usage of this Toolkit as the ultimate goal is to have a simple Interface. We can see the difference between the bare interface between 6.8 and 6.9, and already it is easy to see that the latter would be way more convenient for the user.

Figure 6.8: Simple Layout

Figure 6.9: Stylized Layout

6.2 Results

In this section I will describe my perceived results of the work, taking into account the proposed milestones and the initial objectives that I put on section 1.2. Based on those proposed objectives, I can firmly say that they are all satisfactorily completed, although there is are a few considerations to be made.

First of all, the Add-on has shed light on a part of Unity that I believe is very useful to game developers, as it provides a custom pipeline that can save a lot of time when developing games, specially on a bigger scale. I would strongly recommend for the degree to include the creation of custom tools in the curriculum, as at first I did not have a clue on how it worked, and spent a lot of time researching.

As for the Animation development section, I am very happy with its results as I believe that I learnt many things along the way, more specifically I got a basis from which to start developing my skills as an animator. On the other side, as I set foot into the world of animation, I realized it is far more complex from what I initially anticipated, which leave a loot of room to for me to grow.

As for the applications of this document, as well as its project, I hope that it can shed some light onto aspiring animators and people curious about the subject. When it comes to the project, I intend to leave the project as it is, and freely accessible, as well as a demonstration video in this link:

Drive link

For Documentation about using the tool, as well as instructions for setting up humanoid characters and animations, refer to B. Also, For instruction about visualizing animations in Blender, and exporting setup, refer to A.

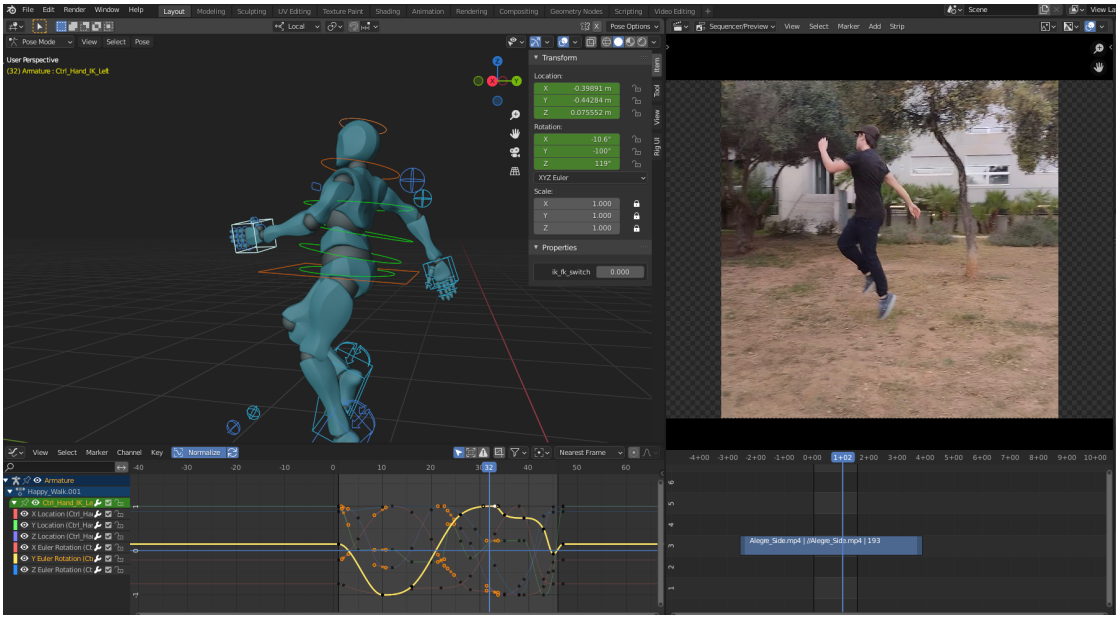


Figure 6.10: Blender Project Screen

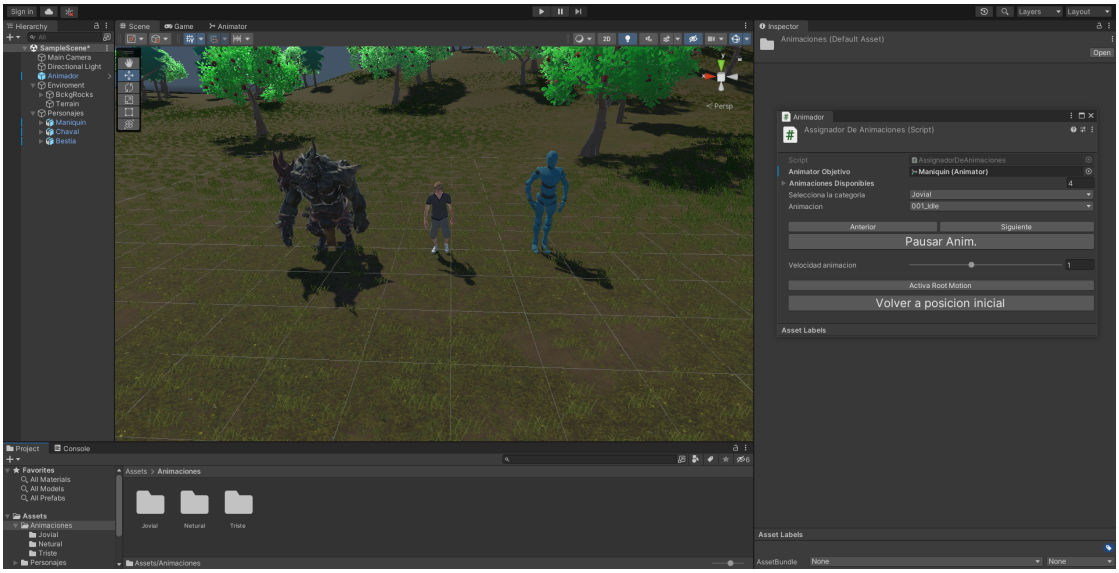


Figure 6.11: Unity Project Screen

CHAPTER 7

Conclusions and Future Work

Contents

7.1 Conclusions

In retrospective, I must say that I have enjoyed the development of this project (specially when animating characters) quite a lot. I have come to learn many things about animation and its industry that I did not see in my coursed degree, so much so that I could not fit everything here. I might have also experienced the so called "dunning-Kruger" effect, because it dawned on me that there is so much more to learn, as this field is much more extensive that I initially though. Nevertheless, I am overall happy with my result on both animation work and the development of the Unity Add-on, while at the same time I feel I have a long way to go if I want to pursue this field professionally.

7.2 Future work

As for future work, I believe that I will continue developing my skills an animator and technical artist, because I feel that the work that I did laid the ground works for further professional development, as well as being a starting point in my portfolio as an animator and a technical artist. I do not think I will modify the project itself, but I will certainly use it in order to use it as reference or to refresh concepts.

Bibliography

[1] Rudolf Arnheim. *Art and visual perception: a psychology of the creative eye.* University of California Press, 1954.

[2] Alan Becker. Alan becker's youtube channel. youtube.com/alanbecker. Accessed: 2023- 17-04.

[3] Howard Beckerman. *Animation: The Whole Story.* Allworth, 2003.

[4] Michael Cuevas. Michael cueva's twitter pofile. twitter.com/michaelCuevas27. Accessed: 2023- 17-02.

[5] Derek Hayes. *Acting and performance for Animation.* Routledge, 2013.

[6] Brian Massumi. *Parables for the Virtual, Movement, Affect, Sensation.* Duke University Press, 2002.

[7] Dan McLaughlin. A rather incomplete but still fascinating history of animation. https://silo.tips/download/a-rather-incomplete-but-still-fascinating-history-of-animation-by-dan-mclaughlin. Accessed: 2023- 17-04.

[8] Microsoft. System.linq library. learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/. Accessed: 2023- 05-14.

[9] Pierrick Picaut. Pierrick picaut's youtube channel. youtube.com/PierrickPicaut$_P$2$DESIGN. Accessed : 2023 - 17 - 02.$

[10] Dee Reinolds. *Kinesthetic Empathy in Creative and Cultural Practices.* 2012.

[11] Frank Thomas and Ollie Johnston. *The illusion of life: Disney animation.* Disney Editions, 1995.

[12] Unity. Oninspectorgui. docs.unity3d.com/ScriptReference/Editor.OnInspectorGUI.html. Accessed: 2023- 05-14.

[13] Unity. System requirements for unity 2021 lts. docs.unity3d.com/Manual/system-requirements.html. Accessed: 2023- 06-04.

[14] Unity. Unity animator. docs.unity3d.com/ScriptReference/Animator.html. Accessed: 2023- 05-02.

[15] Unity. Unity animatorcontroller. docs.unity3d.com/Manual/class-AnimatorController.html. Accessed: 2023- 05-02.

[16] Richard Williams. *The animator's survival kit.* Faber and Faber, 2001.

APPENDIX A

Visualizing and exporting humanoid animations in Blender

This appendix serves to shed some light on the process of properly exporting blender characters and animation in order to use them in Unity projects. It also includes instructions to find animations of characters within a Blender project.

A.1 Visualizing animations

In Blender, one character may contain several Animations even though it only displays one at a time. In order to get to these animations, with the armature of the character or object selected, one must follow these steps:

1. Make sure to have a display window and and extra window, if there is only the default 3D viewport, you can add another window by left-click dragging from the bottom left of the screen up. Then, on the extra window, you have to choose the Dope Sheet window, as seen in A.1 .

2. In the Dope Sheet window, one can choose between several modes or sub-windows, we are interested in the Action Editor window A.2.

3. In the Action Editor, above the timeline, a drop down will be accessible, containing all the animations for the selected character A.3.

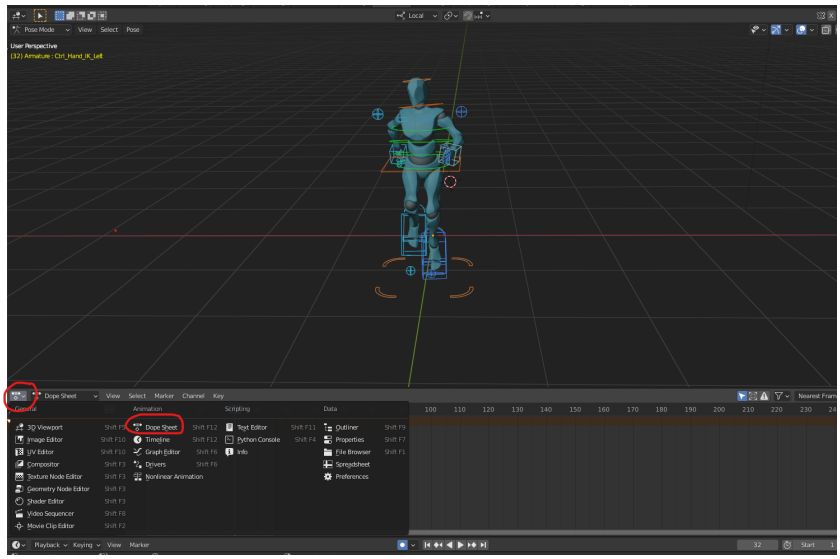


Figure A.1: Window setting

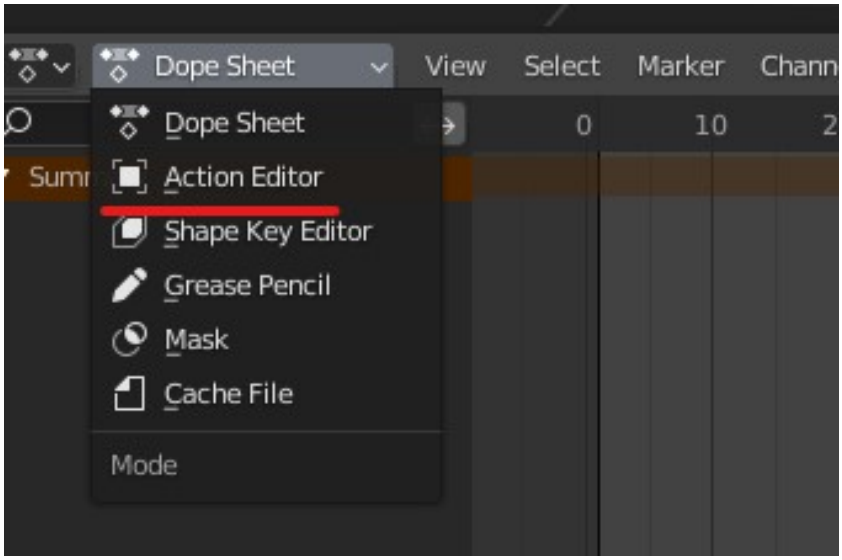


Figure A.2: Action Editor

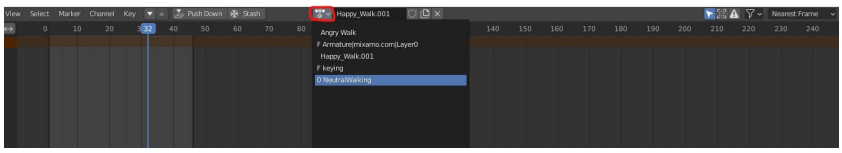


Figure A.3: Animations drop down

A.2 Exporting setup

Let's now take a look onto how to set up the exportation of characters and animations onto Unity.

First of all, we will access the export menu, selecting the option for exporting an .fbx file as shown here A.4.

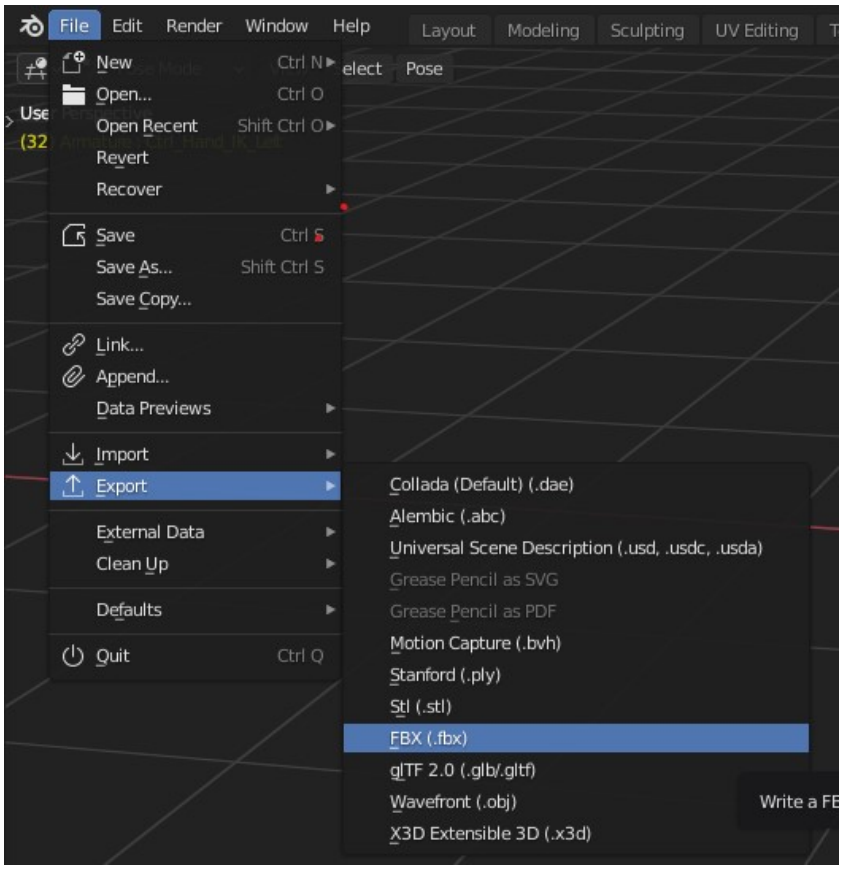


Figure A.4: Export menu

In the exporting menu, we have to take special attention to the Include, Transform and Armature sections of the properties in the right column.

1. In the include section, we select the object to export, we are only interested in the armature and Mesh options.
2. The transform section hold special importance, as the world orientation in Blender and Unity are different. The parameters must be set as shown in A.5.
3. In the Armature section, only the leaf bones checkbox must be checked, as checking the deform bones results in unexpected behaviour.

Now we are ready to Import the resulting .fbx file onto our Unity project.

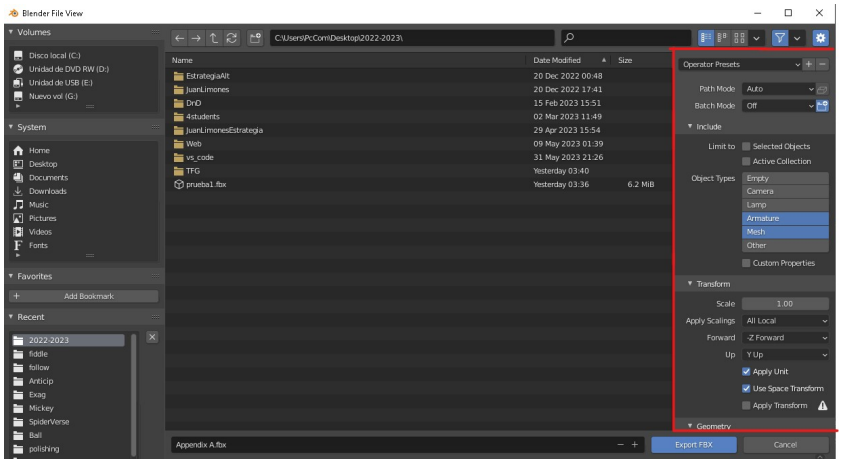


Figure A.5: Exporting parameters

APPENDIX B

Add-on Documentation

B.1 Setting up the Add-on and importing characters

Setting up the Add-on onto a unity project is as easy as either adding the *AssignadorDeAnimaciones.cs* script onto an empty object in your Unity scene, or adding the Animator prefab provided in the project.

Now, when importing a character it is important to make sure of some things. First and foremost, the character must be sure that the bone structure of your character follows that of a humanoid character. When adding a character model onto the project, in the rig section, its animation type is set to generic by default. It must be changed to humanoid for the tool to work , otherwise an error message would appear and the tool would not work. It is also vital to ensure that when the character is added onto the

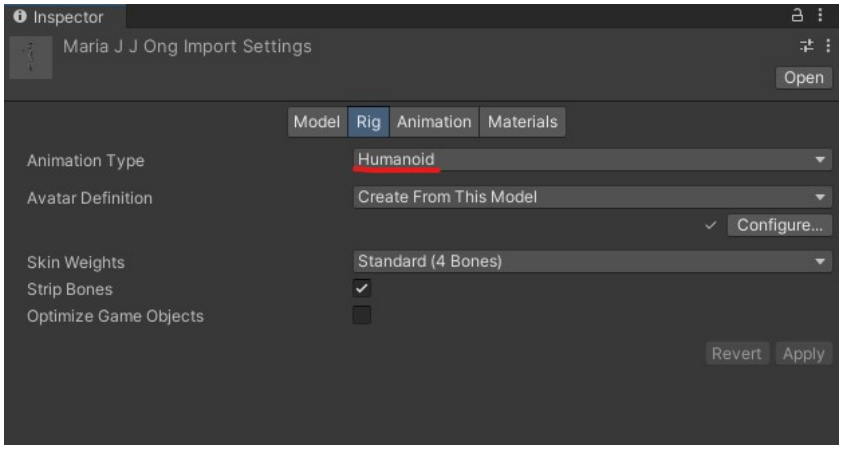


Figure B.1: Humanoid animation type

scene , it has an animator property. It is not necessary to add an animation controller onto it, as it will be rewritten when the the tool is being used.

When in play mode in the Unity Editor, the user will be able to use the tool's functionalities. In order to select different characters within your scene, the user with have to access the property containing the Animators on scene called Animador Objetivo. From there, the user will be able to access the animation clips and modify its behaviours in order to check if the animation suits the character.

Figure B.2: Tool interface

B.2 Setting up and importing animation clips

Now that the tool is in place and the characters to test are set up, we must add our animation clips onto the Unity Project, ensuring that they work correctly with the rigs. When importing an animation clip from a humanoid character onto blender, certain properties, such as making sure that it is looping or baking the root motion into pose based on the original pose so that when enabling and disabling the animation so that no unexpected behaviour occur. Lastly, the way the tool is set up, The animation Clips are searched within a folder called 'Animaciones', so having a folder with this exact name is necessary. Inside it, the user can create any number of sub folders that he wants, and their names will be the categories displayed in the tool. Inside those sub folders, the animation clips must be stored.

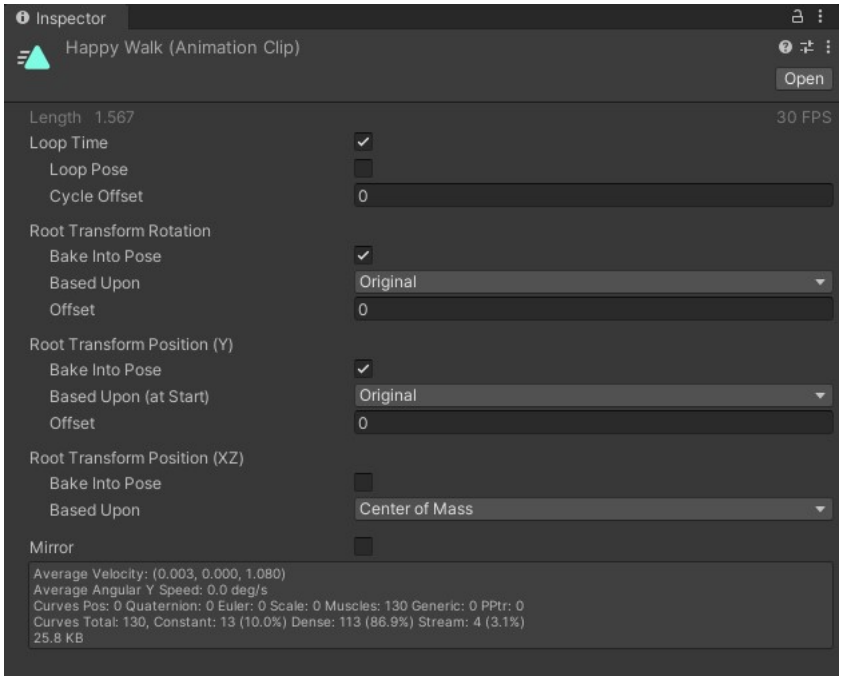


Figure B.3: Animation Clip Settings

Figure B.4: File structure

Appendix C

Source code

This is the source code for the scripts that make up the Unity tool for visualizing animations:

C.1 AsignadorDeAnimaciones

```
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
#if UNITY_EDITOR
using UnityEditor;
#endif
using System.Linq;



public class AssignadorDeAnimaciones : MonoBehaviour
{

    public Animator AnimatorObjetivo//animator del personaje objetivo, solo coge avatares, o personajes humanoides
    {
        get { return _animatorObjetivo; }
        set
        {
            // Para que el animator solo funcione con personajes humanoides
            if (value != null && value.avatar != null && value.avatar.isHuman)
            {
                _animatorObjetivo = value;
            }
        }
    }
    public Animator _animatorObjetivo;

    private string categoria; //Categoria de la animacion seleccionada
    public List<AnimationClip> animacionesDisponibles; //Lista de animaciones disponibles
```

```
31      private Vector3 initialPosition;
32      private Quaternion initialRotation;
33      private void Awake()
34      {
35          if (AnimatorObjetivo != null)
36          {
37              initialPosition = AnimatorObjetivo.transform.position;
38              initialRotation = AnimatorObjetivo.transform.rotation;
39          }
40      }
41      //TODO: acoplar la funcionalidad overrideController
42      public void AsignarAnimacion(AnimationClip animationClip)
43      {
44          // Crea un nuevo AnimatorOverrideController basado en el AnimatorController del objetivo
45          AnimatorOverrideController overrideController = new AnimatorOverrideController(AnimatorObjetivo.runtimeAnimatorController);
46
47          // Asigna la animacion seleccionada al overrideController
48          overrideController["NombreDelEstadoDeAnimacion"] = animationClip;
49
50          // Aplica el overrideController al objetivo
51          AnimatorObjetivo.runtimeAnimatorController = overrideController;
52      }
53
54      public void CargaAnimacionesPorCategoria()
55      {
56          // Carga las animaciones de la categoria seleccionada en availableAnimations
57
58          animacionesDisponibles = GetAnimationsByCategory(categoria);
59      }
60
61  #if UNITY_EDITOR
62
63      //coge los clips por categoria de tu carpeta de Animaciones
64      public List<AnimationClip> GetAnimationsByCategory(string categoria)
65      {
66          //Aqui obtendremos una lista de las animpaciones existentes embase a una categoria
67
68          string animFolderPath = "Assets/Animaciones";//directorio carpeta animaciones
69          string categoryFolderPath = Path.Combine(animFolderPath, categoria);
70
71          //obtiene los archivos .anim de la carpeta de categoria seleccionada
72          string[] animFiles = Directory.GetFiles(categoryFolderPath, "*.anim", SearchOption.TopDirectoryOnly);
73
74          List<AnimationClip> animaciones = new List<AnimationClip>();
75          foreach(string animFile in animFiles)
76          {
77              //convertimos la ruta absoluta del archivo de animacion en una ruta relativa al archivo Assets del proyecto
78              string relativePath = animFile.Replace("\\", "/").Replace(Application.dataPath, "Assets");
79
80              //Cargamos el archivo AnimationClip
81              AnimationClip animationClip = AssetDatabase.LoadAssetAtPath<AnimationClip>(relativePath);
82
83              if(animationClip != null)
84              {
85                  animaciones.Add(animationClip);
86              }
87          }
88
89          return animaciones;
90      }
91  #endif
92      public string[] obtenerCategorias()
93      {
94
95          // Obtenemos la lista de categorias segun las carpetas del poryecto
96
```

```
97          string assetFolderPath = "Assets";//carpeta assets
98          string animFolderPath = "Assets/Animaciones";//carpeta animaciones
99
100         //lista de directorios dentro de la carpeta animaciones
101         string[] directorios = Directory.GetDirectories(animFolderPath, "*", SearchOption.TopDirectoryOnly);
102
103         string[] categorias = directorios.Select(directory =>
104         {
105             string relativePath = directory.Replace(assetFolderPath, "").TrimStart('/');
106             string nombreCategoria = Path.GetFileName(relativePath);
107             return nombreCategoria;
108         }).ToArray();
109
110         return categorias;
111
112     }
113
114     public void ResetPositionAndRotation()
115     {
116         if (AnimatorObjetivo != null)
117         {
118             AnimatorObjetivo.transform.position = initialPosition;
119             AnimatorObjetivo.transform.rotation = initialRotation;
120         }
121     }
122
123     public void SetInitialPositionAndRotation(Vector3 position, Quaternion rotation)
124     {
125         initialPosition = position;
126         initialRotation = rotation;
127     }
128
129     public void PauseAnim()
130     {
131         if(AnimatorObjetivo != null)
132         {
133             AnimatorObjetivo.speed = 0;
134         }
135     }
136
137     public void ResumeAnim()
138     {
139         if (AnimatorObjetivo != null)
140         {
141             AnimatorObjetivo.speed = 1;
142         }
143     }
144
145     public void SetAnimationSpeed(float speed)
146     {
147         if(AnimatorObjetivo != null)
148         {
149             AnimatorObjetivo.speed = speed;
150         }
151     }
152
153     public void ToggleRootMotion()
154     {
155         if (AnimatorObjetivo != null)
156         {
157             AnimatorObjetivo.applyRootMotion = !AnimatorObjetivo.applyRootMotion;
158         }
159     }
160 }
```

C.2 AsignadorDeAnimacionesEditor

```
using System.Collections.Generic;
using UnityEditor;
using UnityEditor.Animations;
using UnityEngine;

[CustomEditor(typeof(AssignadorDeAnimaciones))]
public class AssignadorDeAnimacionesEditor : Editor
{


    private int indiceCategoriaSeleccionada;
    private int indiceAnimSeleccionada;
    private bool pause = false;
    private int previousSelectedAnimationIndex = -1;
    private int previousSelectedCategoryIndex = -1;

    private Dictionary<int, Vector3> initialPositions = new Dictionary<int, Vector3>();
    private Dictionary<int, Quaternion> initialRotations = new Dictionary<int, Quaternion>();



    public override void OnInspectorGUI()
    {


        //Estilo para botones
        GUIStyle estiloBoton = new GUIStyle(GUI.skin.button);
        estiloBoton.fontSize = 20;


        //inspector
        DrawDefaultInspector();

        //referencia a asignadordeanimaciones
        AssignadorDeAnimaciones assignadorDeAnimaciones = (AssignadorDeAnimaciones)target;

        if (assignadorDeAnimaciones.AnimatorObjetivo != null &&
        assignadorDeAnimaciones.AnimatorObjetivo.avatar != null &&
        !assignadorDeAnimaciones.AnimatorObjetivo.avatar.isHuman)
        {
            EditorGUILayout.HelpBox("El personaje no es humanoide!", MessageType.Warning);
        }



        //Almacenar posicion y rotacion inicial de todos los animators
        int instanceID = assignadorDeAnimaciones.AnimatorObjetivo.GetInstanceID();
        if (!initialPositions.ContainsKey(instanceID))
        {
            initialPositions[instanceID] = assignadorDeAnimaciones.AnimatorObjetivo.transform.position;
            initialRotations[instanceID] = assignadorDeAnimaciones.AnimatorObjetivo.transform.rotation;
        }


        //lista desplegable de seleccionador de categorias
        string[] categorias = assignadorDeAnimaciones.obtenerCategorias();
        int nuevoIndiceCategoriaSeleccionada = EditorGUILayout.Popup("Selecciona la categoria",indiceCategoriaSeleccionada,categorias);


        if (nuevoIndiceCategoriaSeleccionada != indiceCategoriaSeleccionada)
        {
            indiceCategoriaSeleccionada = nuevoIndiceCategoriaSeleccionada;
            indiceAnimSeleccionada = 0;
```

```
64
65
66          }
67          //animaciones de categoria seleccionada
68          assignadorDeAnimaciones.animacionesDisponibles = assignadorDeAnimaciones.GetAnimationsByCategory(categorias[indiceCategoriaSeleccionada]);
69
70          //seleccion animacion
71          string[] nombesAnim = assignadorDeAnimaciones.animacionesDisponibles.ConvertAll(a => a.name).ToArray();
72          indiceAnimSeleccionada = EditorGUILayout.Popup("Animacion", indiceAnimSeleccionada, nombesAnim);
73
74          //asignar animacion seleccionada al animator
75          if((previousSelectedAnimationIndex != indiceAnimSeleccionada || previousSelectedCategoryIndex!= indiceCategoriaSeleccionada)
76              && assignadorDeAnimaciones.animacionesDisponibles != null && assignadorDeAnimaciones.animacionesDisponibles.Count > 0)
77          {
78              //assignadorDeAnimaciones.AsignarAnimacion(assignadorDeAnimaciones.animacionesDisponibles[indiceAnimSeleccionada]);
79
80              assignadorDeAnimaciones.AnimatorObjetivo.runtimeAnimatorController =
81                  CrearAnimator(assignadorDeAnimaciones.animacionesDisponibles[indiceAnimSeleccionada]);
82              previousSelectedAnimationIndex = indiceAnimSeleccionada;
83              previousSelectedCategoryIndex = indiceCategoriaSeleccionada;
84
85              assignadorDeAnimaciones.SetInitialPositionAndRotation(initialPositions[instanceID], initialRotations[instanceID]);
86              assignadorDeAnimaciones.ResetPositionAndRotation();
87          }
88
89          GUILayout.Space(15);
90
91
92          EditorGUILayout.BeginHorizontal();
93          if (GUILayout.Button("Anterior"))
94          {
95              indiceAnimSeleccionada = Mathf.Max(0, indiceAnimSeleccionada - 1);
96              //assignadorDeAnimaciones.AsignarAnimacion(assignadorDeAnimaciones.animacionesDisponibles[indiceAnimSeleccionada]);
97
98              assignadorDeAnimaciones.AnimatorObjetivo.runtimeAnimatorController =
99                  CrearAnimator(assignadorDeAnimaciones.animacionesDisponibles[indiceAnimSeleccionada]);
100             assignadorDeAnimaciones.SetInitialPositionAndRotation(initialPositions[instanceID], initialRotations[instanceID]);
101             assignadorDeAnimaciones.ResetPositionAndRotation();
102         }
103         if (GUILayout.Button("Siguiente"))
104         {
105             indiceAnimSeleccionada = Mathf.Min(assignadorDeAnimaciones.animacionesDisponibles.Count - 1, indiceAnimSeleccionada + 1);
106             //assignadorDeAnimaciones.AsignarAnimacion(assignadorDeAnimaciones.animacionesDisponibles[indiceAnimSeleccionada]);
107
108             assignadorDeAnimaciones.AnimatorObjetivo.runtimeAnimatorController =
109                 CrearAnimator(assignadorDeAnimaciones.animacionesDisponibles[indiceAnimSeleccionada]);
110             assignadorDeAnimaciones.SetInitialPositionAndRotation(initialPositions[instanceID], initialRotations[instanceID]);
111             assignadorDeAnimaciones.ResetPositionAndRotation();
112         }
113         EditorGUILayout.EndHorizontal();
114         //boton Pausar/Reanudar
115         string buttonText = pause ? "Reanudar Anim." : "Pausar Anim.";
116
117         if (GUILayout.Button(buttonText, estiloBoton))
118         {
119             if (pause)
120             {
121                 assignadorDeAnimaciones.ResumeAnim();
122
123             }
124             else
125             {
126                 assignadorDeAnimaciones.PauseAnim();
127
128             }
129         }
```

```
130
131         pause = assignadorDeAnimaciones.AnimatorObjetivo.speed == 0;
132
133 GUILayout.Space(15);
134
135         float newVel = EditorGUILayout.Slider("Velocidad animacion", assignadorDeAnimaciones.AnimatorObjetivo.speed, 0, 2.5f);
136         assignadorDeAnimaciones.SetAnimationSpeed(newVel);
137
138
139 GUILayout.Space(15);
140
141
142         string RootMotiionTexto = assignadorDeAnimaciones.AnimatorObjetivo.applyRootMotion ? "Desactiva Root Motion" : "Activa Root Motion";
143
144
145         if (GUILayout.Button(RootMotiionTexto))
146         {
147             assignadorDeAnimaciones.ToggleRootMotion();
148         }
149
150
151 GUILayout.Space(5);
152
153         if (GUILayout.Button("Volver a posicion inicial",estiloBoton))
154         {
155             assignadorDeAnimaciones.ResetPositionAndRotation();
156         }
157
158     }
159
160
161     // Create a 2D texture
162     /*private Texture2D MakeTex(int width, int height, Color col)
163     {
164         Color[] pix = new Color[width * height];
165
166         for (int i = 0; i < pix.Length; i++)
167         {
168             pix[i] = col;
169         }
170
171         Texture2D result = new Texture2D(width, height);
172         result.SetPixels(pix);
173         result.Apply();
174
175         return result;
176     }*/
177
178     //crea un animator para cada clip
179
180     private RuntimeAnimatorController CrearAnimator(AnimationClip clip)
181     {
182         // Crear un Animator Controller en tiempo de ejecucion con una sola animacion
183         AnimatorController controller = new AnimatorController();
184         controller.AddLayer("Default");
185
186         AnimatorStateMachine sm = controller.layers[0].stateMachine;
187         AnimatorState state = sm.AddState(clip.name);
188         state.motion = clip;
189
190         return controller;
191     }
192
193 }
```