CI601 PROCEDURAL ANIMATION SHOWCASE

Gabriel Kelly

Student Number – 20812043

Digital Games Development

Supervisor - Almas Baimagambetov

Second Reader - Karina Rodriguez Echavarria

Software Artifact GitHub Link - https://github.com/Gabe16Kelly/ProceduralAnimationShowcase.git

Table of contents

Section 1: Introduction

1.1 Abstract

Procedural Animation is a concept widely known in the games industry while only being used subtly to create dynamic interactions in the game's world such as a character looking towards an object or ragdoll physics. This paper will discuss the study and implementation of rigging and animating a 3D character locomotion system using a Procedural Animation Algorithm. The goal is to see if Procedural Animation can be successfully implemented onto a rigged object preventing the need for traditional keyframe Animations.

1.2 Aims and Objectives

It is widely shown that the graphics of modern games are now at a very high standard while the animation is lacking in certain ways decreasing the world's high levels of immersion. Although some games have been able to develop high-quality animations, such as God of War PS4, this still uses a lot of high-end resources. To create an immersive and dynamic environment you need characters to live in them that believably respond to the world. In most games, animators try to create a believable locomotion system for their characters, however this can be tricky. For continuous movement animators create a walking/running cyclic animation using keyframes or motion-capture, which can be repetitive. However, using an algorithm framework to set up, a procedurally animated character easily adjusts their movement with the environment during play.

By using an algorithm for procedural animations, it is possible to create a diverse and believable animation while keeping the expenses and time of setting up the animations smaller than it would be doing it by hand (framework/motion capture).

This will be done by implementing a procedural animation framework onto characters that identify certain functions of the body for the character to do in real-time while the game is running to create a more believable experience.

The main aims are to create a software artefact in Unity that consists of multiple characters, which have been rigged and animated using an algorithm/framework with the language C#.

A demonstration game will also be created to show the Procedural Animation, however this will not be the focus of the project but will help in demonstrating and test the capabilities of the algorithm.

Objectives to get to this aim:

1. Research the main principles/rules of Procedural animation and how others have been able to use it.
2. Research more thoroughly the inner principles such as raytracing and Inverse Kinematics.
3. Plan/design what characters will be created and how they might move around using Procedural Animation
4. Create/collect 3D models of characters and rig them.
5. Implement the code onto the limbs of the characters so they can procedurally walk around.
6. Compare how using Forward or Inverse kinematics can affect a character's movement.
7. Test how the algorithm affects the characters and adjust to better fit.
8. Refine any problems found when testing.

1.3 Requirements

Functional Requirements:

Source Code

- The creature will be made of limbs that are connected by joints.
- Inverse Kinematics are used on the legs, so they can be controlled by the algorithm.
- The character’s body parts will be rigged together using a hierarchy of bones.
- When the character moves faster the legs should move faster as well
- Target Points are ray casted on the ground from the body, so the feet know when and where to step next.
- Each joint of the character should have a separate script attached so they can change independently.

Non-functional requirements:

Source Code

- It should work for straight, backwards, side to side and curved movements.
- Feet of the character should be fixed on the ground.
- The feet should only move when the opposite feet are grounded.
- Legs should adapt dynamically to different heights and slopes of the terrain.
- Inverse Kinematics are used on the character to create a more natural movement.
- The character’s joints should have minimum and maximum angle restrictions to prevent the joints from moving into each other.
- It should be able to animate dynamically depending on the speed of the character.

1.4 Deliverables

For my project to be successful, I need to complete certain deliverables, I would like to have a software artefact that shows the usage and development of Procedural Animation in games, a report showing my design and implementation of the project, and finally, a presentation to visually show off the Software Artefact, which has been created.

1.4.1 Scope of Deliverables

The Software Artefact will be created to develop and implement Procedurally Animated characters. Multiple unique characters will be developed to show a broad overview of what Procedural Animation has to offer. In the end, I would like to have a better understanding of Procedural Animation where I can develop this style even more over multiple iterations of characters. The design and implementation of the Software Artefact will be completed first with an expected due date of completion being February 2023.

The report will allow me to show what I have created from my software artefact in a more detailed approach. The Report will consist of the research and design, which went into creating the Procedural Animated characters with a detailed look into the implementation of the code and 3D models. Finally, with a critical review of what I have produced in further areas, I could investigate to improve my project in the future. Alongside creating the software artefact, I will be writing up the report allowing me to easily communicate across the implementation, the report will be completed in May 2023.

Section 2: Methodology

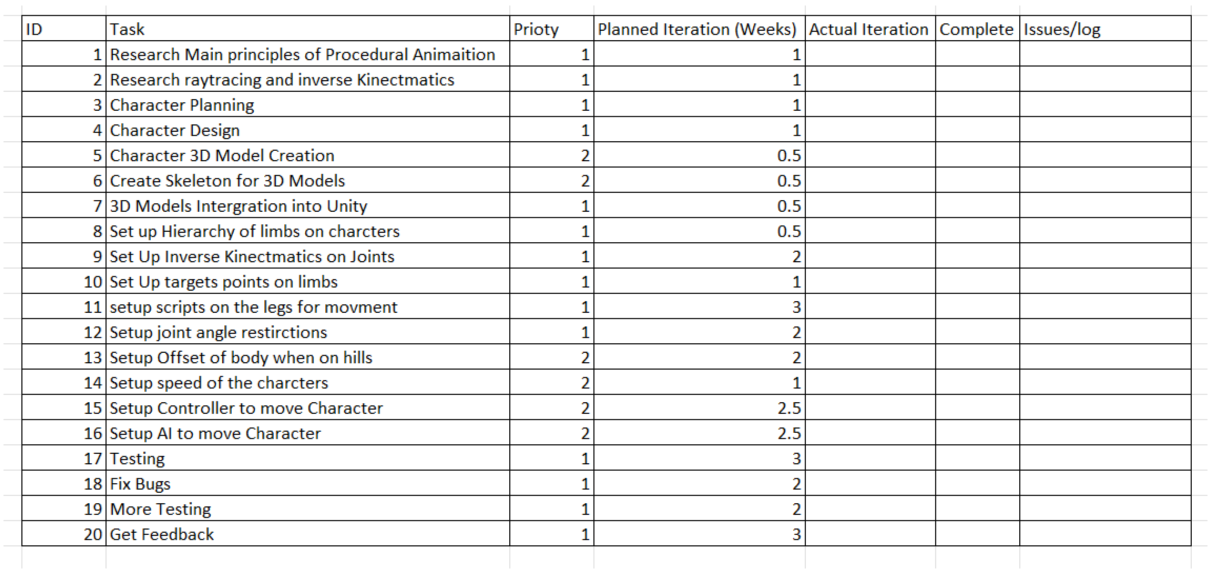
Explanation of choice and use of relevant methodologies such as project management, surveys, development tools/environments, testing

2.1 Project Management

Using a Project Management Methodology is crucial to develop a strong project. This will help with efficiency and productivity by helping with visualising the tasks at hand and time allocated. Below I will discuss the planning and development tools I have chosen to help with the development of this project.

2.1.1 Product Backlog

By creating a long list of tasks I can see exactly what I need to do in one place. However, as the project evolves the list can become very complicated and disorganised, so instead of a list I will be creating a product backlog to be able to track the priorities, planned iterations, what the actual iteration was, planned and actual effort put in, completion and a quick log of any issues. Shown in figure 5.1 is my Product Backlog this is the first version as I am sure there will be a lot more I will add during the creation of the project. Although the Product Backlog I have created in excel is very manageable, it is not well organised, which is why I have chosen to use a Trello Board over the excel sheet.

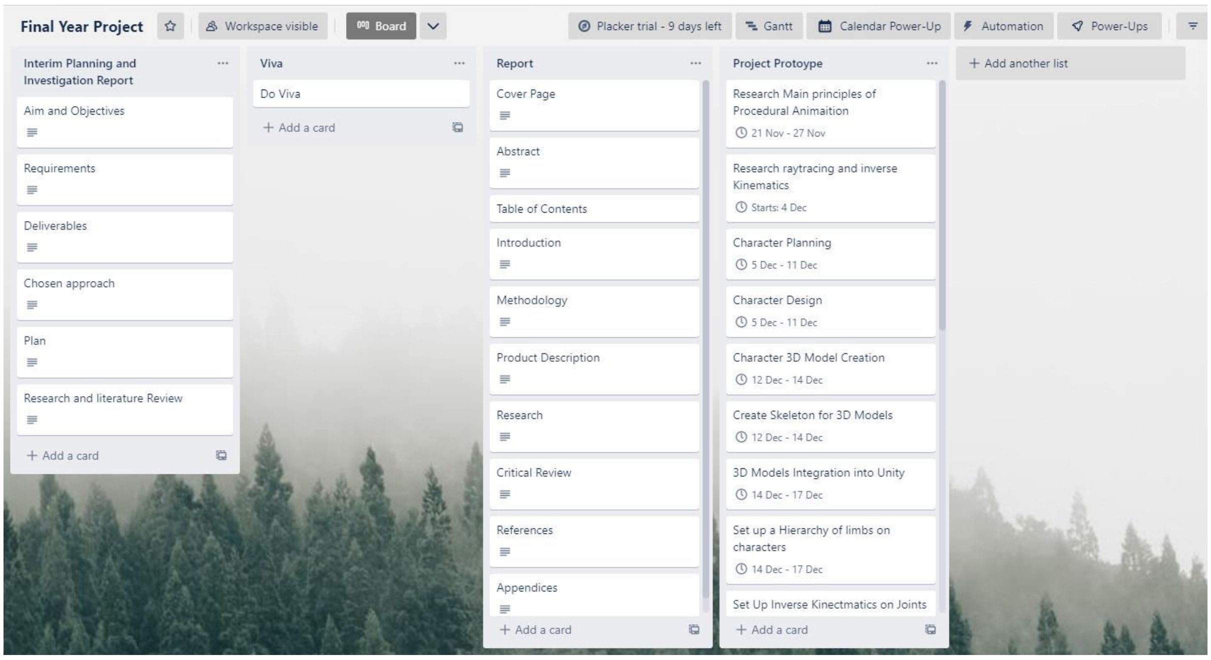


ID	Task	Prioty	Planned Iteration (Weeks)	Actual Iteration	Complete	Issues/log
1	Research Main principles of Procedural Animaition	1	1			
2	Research raytracing and inverse Kinectmatics	1	1			
3	Character Planning	1	1			
4	Character Design	1	1			
5	Character 3D Model Creation	2	0.5			
6	Create Skeleton for 3D Models	2	0.5			
7	3D Models Intergration into Unity	1	0.5			
8	Set up Hierarchy of limbs on charcters	1	0.5			
9	Set Up Inverse Kinectmatics on Joints	1	2			
10	Set Up targets points on limbs	1	1			
11	setup scripts on the legs for movment	1	3			
12	Setup joint angle restirctions	1	2			
13	Setup Offset of body when on hills	2	2			
14	Setup speed of the charcters	2	1			
15	Setup Controller to move Character	2	2.5			
16	Setup AI to move Character	2	2.5			
17	Testing	1	3			
18	Fix Bugs	1	2			
19	More Testing	1	2			
20	Get Feedback	1	3			

Figure 2.1 – Product backlog created in Excel

2.1.2 Trello Board

I started with the product backlog on excel first, as I wanted to get everything I needed to do down somewhere that is easy to read and copy. However, a Trello board is a much better way to organise my work due to its card system where you can easily add them onto your boards with easy to read titles and descriptions within, so I can add even more information and they can be dragged around the board depending on what stage that card is at.



Interim Planning and Investigation Report
- Aim and Objectives
- Requirements
- Deliverables
- Chosen approach
- Plan
- Research and literature Review
- + Add a card

Viva
- Do Viva
- + Add a card

Report
- Cover Page
- Abstract
- Table of Contents
- Introduction
- Methodology
- Product Description
- Research
- Critical Review
- References
- Appendices
- + Add a card

Project Protoype
- Research Main principles of Procedural Animaition — 21 Nov - 27 Nov
- Research raytracing and inverse Kinematics — Starts: 4 Dec
- Character Planning — 5 Dec - 11 Dec
- Character Design — 5 Dec - 11 Dec
- Character 3D Model Creation — 12 Dec - 14 Dec
- Create Skeleton for 3D Models — 12 Dec - 14 Dec
- 3D Models Integration into Unity — 14 Dec - 17 Dec
- Set up a Hierarchy of limbs on characters — 14 Dec - 17 Dec
- Set Up Inverse Kinectmatics on Joints
- + Add a card

+ Add another list

Figure 2.2 – Trello Board

2.2 Development tools

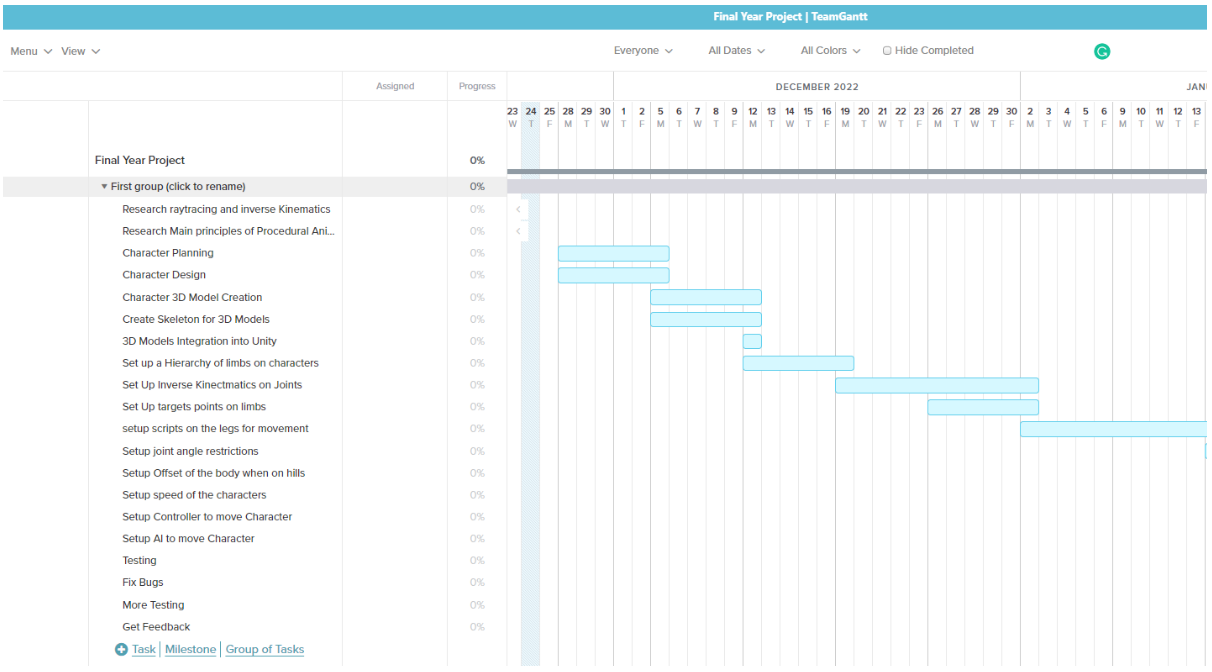
Choosing the right software to create my project was very tricky but after trialling several options, I narrowed it down to Unity or Unreal Engine. The main difference between the software is their native programming language. Unity uses C#, which is a language I am more comfortable with, while Unreal uses C++ that I am still learning. However, Unreal has its blueprints feature unique to the engine, which I am more comfortable using than C# but I would prefer to use C# in my project to advance my knowledge in that specific language to make myself more employable.

Both Engines have a wide community of developers and designers, each having thousands of pages of documentation with 24/7 support, so I am sure I will be able to find a solution for any problems for both. The asset store is something I will be using to obtain some models for my project, both are stocked with lots of free assets. However, I think Unreal has more higher quality assets than Unity.

Overall, Unity is the best choice for me as it uses a very familiar language, so I can easily jump into it and I do not need to spend time learning the alternative engine. Even though Unity is a bit slower to render and process information than Unreal, it takes up less processing power on my computer preventing the chance of lag and crashes.

2.2.3 Gantt Chart

I am using a Gantt Chart, so I can easily see the length of time I have to complete a certain task in my project. As you can see in Figure 2.3 this is a section of the Gantt Chart I have created. I have carefully spread out how long I believe each task will take for me to complete by adding some extra time in case problems appear and it takes me longer to do than predicted. As you can see multiple tasks overlap each other, this is because they can be done together efficiently, for example, character planning and character design are close, because designing characters is a part of planning. The same applies with character model creation and character skeleton creation, for me to create a 3D model of a character part of that is adding the skeleton to it, which will tick off 2 tasks at once.



Task	Progress
Final Year Project	0%
First group (click to rename)	0%
Research raytracing and inverse Kinematics	0%
Research Main principles of Procedural Ani...	0%
Character Planning	0%
Character Design	0%
Character 3D Model Creation	0%
Create Skeleton for 3D Models	0%
3D Models Integration into Unity	0%
Set up a Hierarchy of limbs on characters	0%
Set Up Inverse Kinectmatics on Joints	0%
Set Up targets points on limbs	0%
setup scripts on the legs for movement	0%
Setup joint angle restrictions	0%
Setup Offset of the body when on hills	0%
Setup speed of the characters	0%
Setup Controller to move Character	0%
Setup AI to move Character	0%
Testing	0%
Fix Bugs	0%
More Testing	0%
Get Feedback	0%

Figure 2.3 – section of Gantt Chart

2.3 – Risk Analysis

This project can have many risks involved, which could slow down or terminate project productivity, this Risk Analysis table will allow me to see some of the risks that have a chance of happening, allowing me time to come up with ways to prevent them.

Risks	Likelihood (1 Unlikely – 5 Likely)	Consequences	Prevention
The project will take longer than the allocated timeline	2	An incomplete project is delivered on the final due date	Keep backups of your work and keep in schedule with the Gantt Chart
Hardware/Software Failure	1	Work is lost, causing a delay	Create regular backups to GitHub, and prevent using software in beta updates prevents chance of crashes.
Corrupted Files	2	Files being lost, causes delay	Create regular backups, prevent moving around files too much to prevent data loss
Burnout due to stress/ mental health	3	Productivity is reduced	Create buffer zones around work deadlines to allow time for less productivity
Scope Creep	3	More work might be needed to successfully complete the project increasing delay.	Create clear parameters from the start.
Supervisor Change	1	Need to take the time to find a new supervisor and get them up to date with the work that is being produced	Cannot predict this
External Hazards	1	Fire, earthquake, floods, vandalism etc, could destroy all hardware providing loss of data	Cannot be predicted but news can be monitored to see possibility of hazards appearing
Illness	2	Delay in work due to inability to work	Create a buffer zone before the deadline to allow for time loss

Section 3: Research

3.1 Introduction

Procedural Animation in its basic concept enables an object to be animated using an algorithm, which automatically updates the transforms of the bones depending on specific rules (Pêcheux, 2022). I wanted to explore Procedural Animation in my project to come up with a way to create a more efficient development cycle for creating animations. As traditionally in games when the character moves, an artist has hand-crafted each individual movement. Each animation is a predefined animation (Zucconi, 2017), so when a character wants to perform a different movement, an artist needs to create another animation.

In this section, I would like to explore how Procedural Animation can be created and used more effectively than traditional animation and how this has influenced my design choices for the project.

3.2 Benefits of Animating Procedurally

Procedural Animation automatically generates animation in real time that allows for more diverse actions, which otherwise would be done using predefined keyframe animations. Procedural Animation is used in particle systems to create smoke or fire, cloth animations, hair/fur dynamics and character animation (Team Houdini, 2019).

It is most used in video games for more simple features such as characters turning their head to look at an object and ragdoll physics for death of characters, allowing for the body to realistically fall onto the floor.

Figure 3.1 – Ragdoll Physics

When creating a keyframe animation, an animator is allocated to manually create the behaviour of the model by manipulating each bone. The animator has control over all the bones' positions and movements. While using Procedural Animation, the animator adds conditions onto the bones using parameters such as force and torques that also help control the bones' positions and movements.

The main benefits of using Procedural Animation are its dynamic movements, which mimics the randomness of nature. It is difficult to create fluid and organic animations when using keyframes that also take a lot longer to do by comparison.

In my project, I am creating a Procedural Animation Locomotion System to produce a dynamic animation applied to a character. Similarly using the same character, I will then implement Keyframe Animation and compare the two.

3.2 Fundamentals of Procedural Animation

Characters being animated through Procedural Animation are guided dynamically by the code rather than traditional Keyframe Animation, saying I want this character to be in this position now then move to another position. The code can communicate with the bones effectively, for example, if the player looks down, the code will tell the spine bones rotation of the character to be negative 40. This allows us to generate any animation using code.

However, the more specific and complex you want the animation to be, the more lines of code are required, and this is why Procedural Animation is normally used to animate a few bones at a time. The biggest benefit of using Procedural Animation is you are no longer needed to create the animations yourself. The animation is determined by the model's bone physics enabling you to create some dynamic poses (Zucconi, 2017).

Through my research, I understood I needed to start small with only animating a few bones at a time, which led me to create a locomotion system for a character. The main bones, which will be animated are the thigh, ankle and foot bones. A locomotion system uses a walk cycle and there are 4 key poses in a walk cycle for a humanoid character; Left Foot Contact Pose, Right Foot Passing Pose, Right Foot Contact Pose and Left Foot Passing Pose (Polygon Treehouse, n.d.).

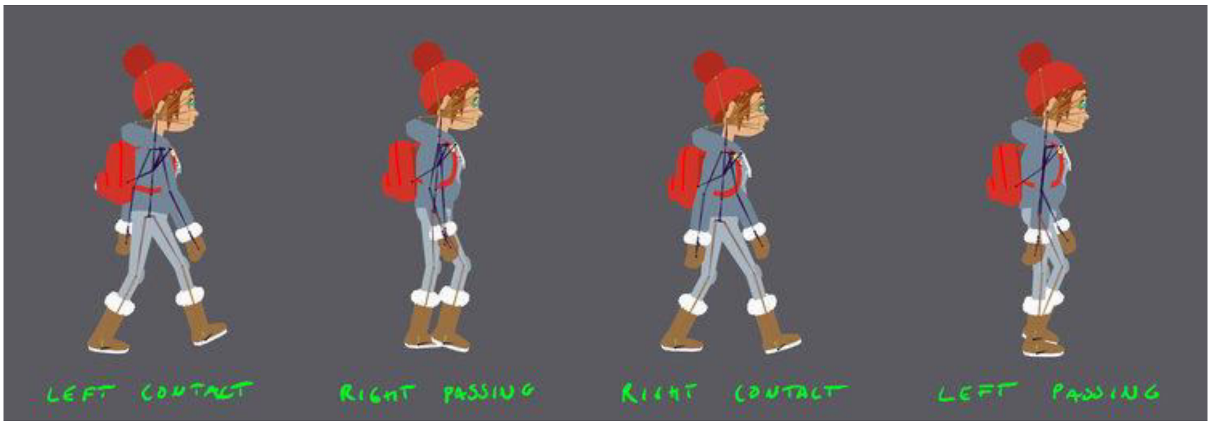
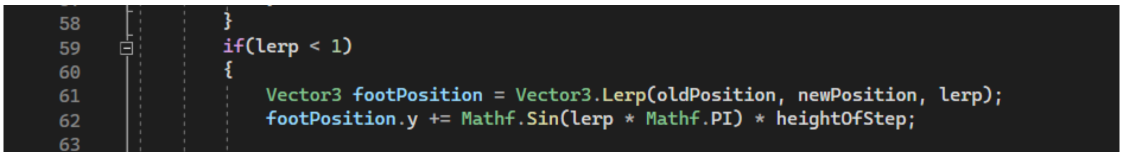


Figure 3.2 – Walking Cycle

Through the Walking Cycle, I was able to come up with some rules, which the animation needed to follow:

- The foot should flow in an arc-type movement to the next step.
- The opposite foot should be grounded when the current foot is moving.
- The opposite foot should start moving when the current foot is grounded.

Most of these rules have transitioned into non-functional requirements, such as the foot flowing in an arc-type movement, which influenced my project in using a sin curve shown in figure 3.3. This code is attached to each one of the character's feet and when the foot is about to take a step, by using the Mathf.sin function this allows me to make the curve like movement to create the illusion that the foot is walking.



```
        }
        if(lerp < 1)
        {
            Vector3 footPosition = Vector3.Lerp(oldPosition, newPosition, lerp);
            footPosition.y += Mathf.Sin(lerp * Mathf.PI) * heightOfStep;
```

Figure 3.3 – Sin curve function used for foot movement

3.3 Successful Procedural Animation Projects

Codeer on YouTube has been able to successfully use Procedural Animation for the creatures in their game. He has been a very good inspiration for me as I was able to see the capabilities of Procedural Animation in Unity and understand some of the rules, he applied to make this happen.

Figure 3.4 – Preview of Codeers game using Procedural Animation

Codeer has successfully used ray casts on the characters to scan the environment. This real-time calculation understands where the character is able to place their next foot. I was able to apply this to my project by creating 2 ray casts for each foot. One to place the current character's foot position onto the ground and another to calculate the next foot position. As shown in figure 3.5, above the spider there are 4 target points each corresponding to one foot. These target points are attached to the body of the spider and each point is ray casted down scanning the ground. Because the feet are stuck to the ground they will not move with the body so when the target points get too far away from the current position of the feet, it will move the feet to the current position of the ray casted target points.

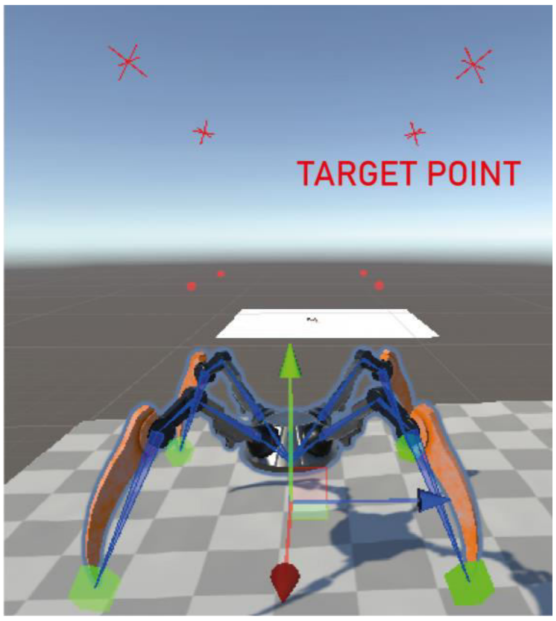


Figure 3.5 -Target Points used for ray casting the next step

Section 4: Product Description

4.1 Implementation of the Product

4.1.1 Setting Up the Rig

Animation is traditionally implemented by creating a character rig. The rig is made up of joints and bones in its own hierarchy that can manipulate the mesh. The bones use a pivot point allowing them to rotate, which creates the joints at these points. The bones can translate, rotate and scale around their corresponding pivot point (Petty, 2018).

The bones are setup in a hierarchy, for example in a rigged leg the thigh bone would be the parent of the ankle bone, which is separated by the knee joint. When the parent thigh bone is moved the children will automatically move with it down the chain of hierarchy. Moving parents makes its children's bones move around in the world space, however their transformation always stays the same.

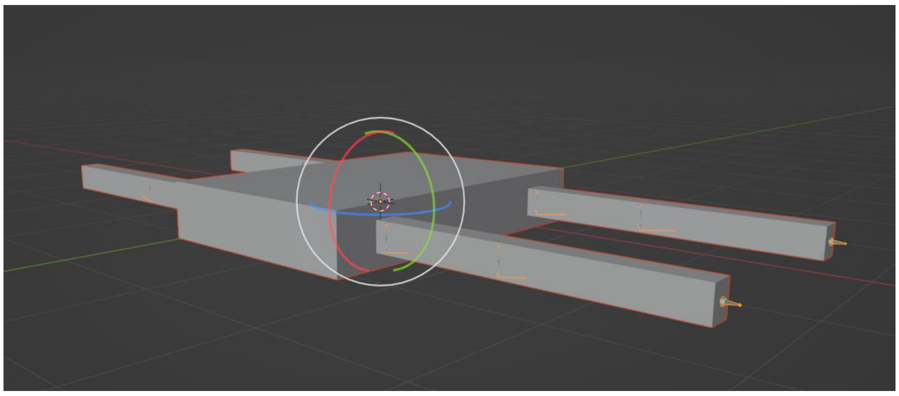


Figure 4.1 – Model created for code implementation.

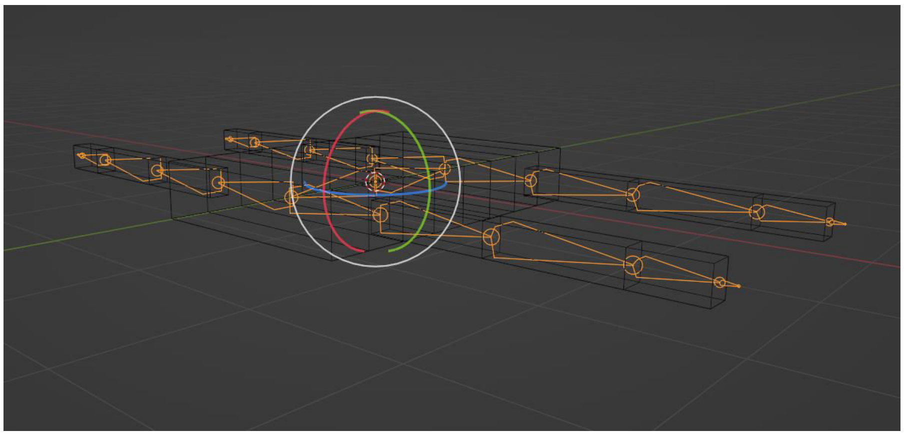
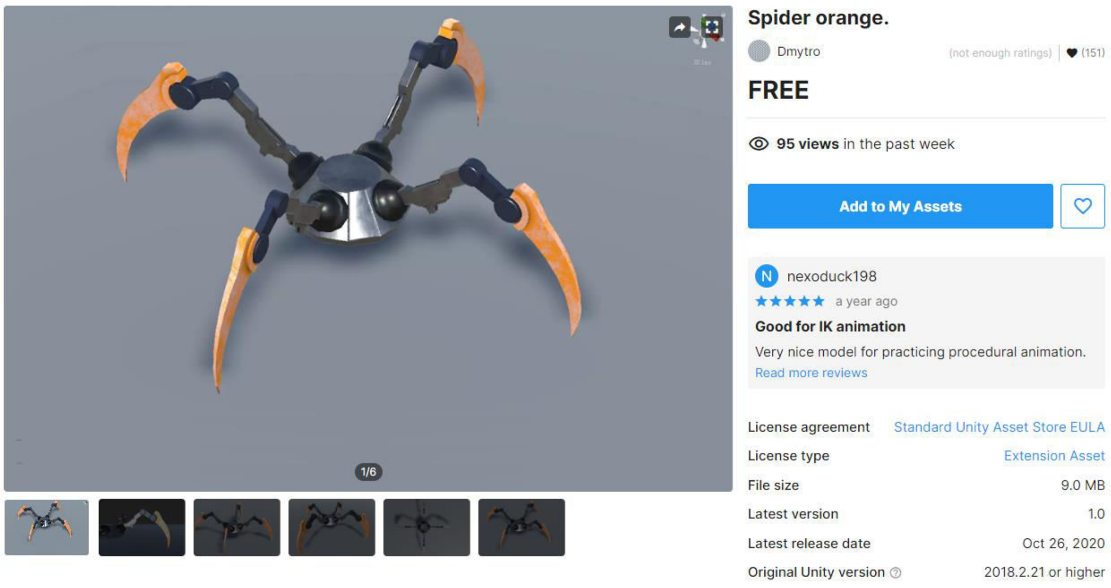


Figure 4.2 – Rigged model created for code implementation

For my project, I have used simple joints as I only need them to rotate. I started off with creating my own simple rigged mesh to experiment on, shown in figure 4.1. It would be a simple quadriplegic character using 3 joints to make up the legs, which will be the main part I will be adding the code onto. However, I found using this model quite delicate to manipulate due to its simple form and I knew when I finished the implementation of the code it would not be satisfying to look at.

This made me come to the decision to scrap the idea of creating my own models and use rigged assets from the Unity Store instead to experiment with, shown in figure 4.3. I was recommended this spider due to its simplistic form with a very professional rigged body. Also the reviews all agree that this spider is very good for practising Procedural Animation, as shown in figure 4.4.



Spider orange.

Dmytro (not enough ratings) | ♥ (151)

FREE

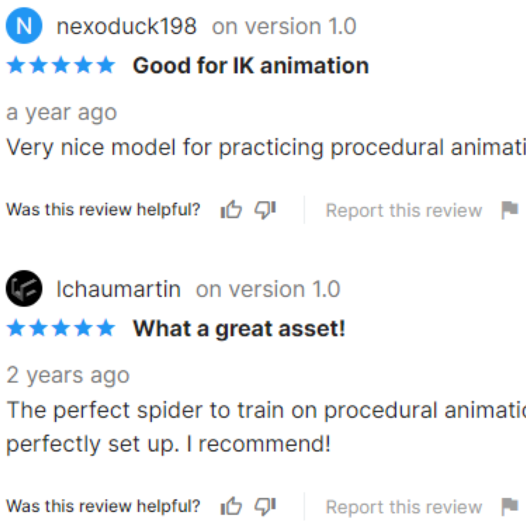
95 views in the past week

Add to My Assets

nexoduck198
★★★★★ a year ago
Good for IK animation
Very nice model for practicing procedural animation.
Read more reviews

License agreement	Standard Unity Asset Store EULA
License type	Extension Asset
File size	9.0 MB
Latest version	1.0
Latest release date	Oct 26, 2020
Original Unity version	2018.2.21 or higher

Figure 4.3 – Orange Spider Asset from Unity store

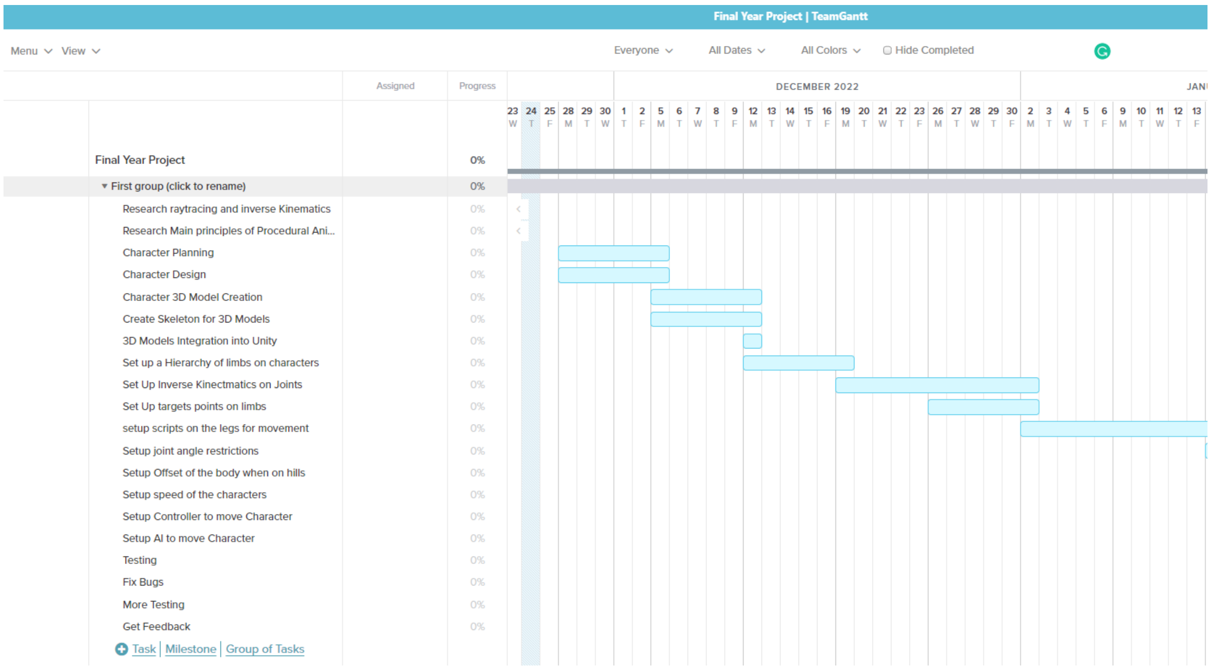


nexoduck198 on version 1.0
★★★★★ Good for IK animation
a year ago
Very nice model for practicing procedural animation.
Was this review helpful? Report this review

lchaumartin on version 1.0
★★★★★ What a great asset!
2 years ago
The perfect spider to train on procedural animation. Looks really nice, and the skeleton is perfectly set up. I recommend!
Was this review helpful? Report this review

Figure 4.4 – Reviews for the Orange Spider Asset

Now that I was not starting with planning and creating my own models this put me 2 weeks ahead of schedule in the Gantt Chart shown in figure 4.5. So instead, I started off with setting up the limbs and implemented Inverse Kinematics while moving the planning and creating of the 3D models to the end of the timeline, as now I know these are not a necessary task to complete for my project to work successfully, but are a task if I wanted to experiment with different unique types of characters.

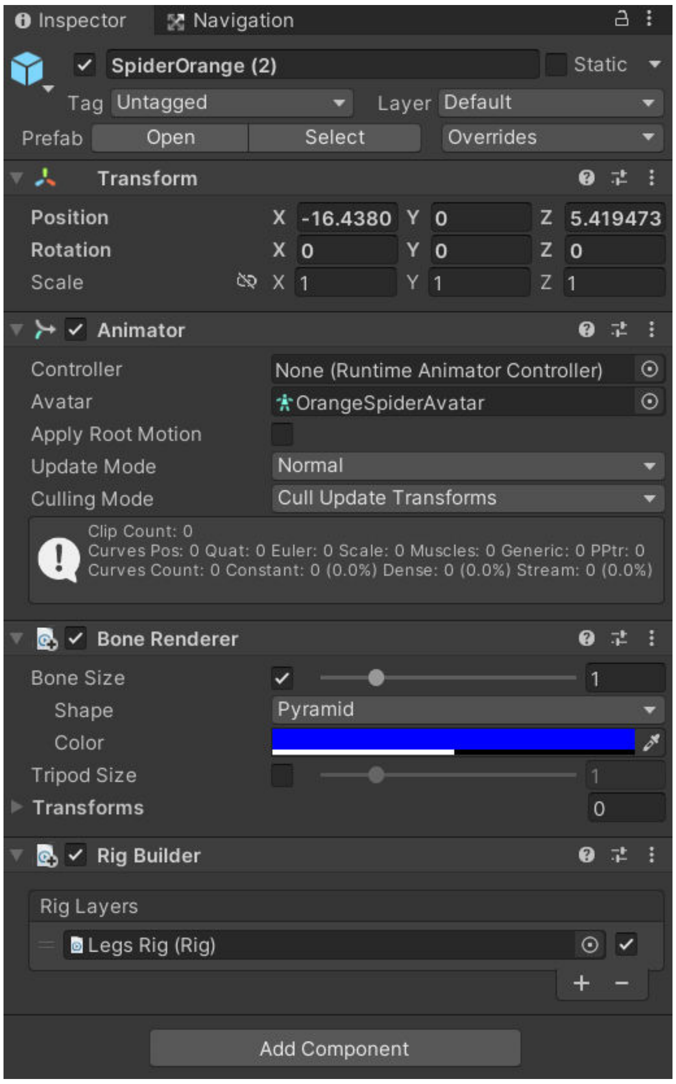


	Assigned	Progress
Final Year Project		0%
▾ First group (click to rename)		0%
Research raytracing and inverse Kinematics		0%
Research Main principles of Procedural Ani...		0%
Character Planning		0%
Character Design		0%
Character 3D Model Creation		0%
Create Skeleton for 3D Models		0%
3D Models Integration into Unity		0%
Set up a Hierarchy of limbs on characters		0%
Set Up Inverse Kinectmatics on Joints		0%
Set Up targets points on limbs		0%
setup scripts on the legs for movement		0%
Setup joint angle restrictions		0%
Setup Offset of the body when on hills		0%
Setup speed of the characters		0%
Setup Controller to move Character		0%
Setup AI to move Character		0%
Testing		0%
Fix Bugs		0%
More Testing		0%
Get Feedback		0%

Task | Milestone | Group of Tasks

Figure 4.5 – Gantt Chart showing 2 weeks ahead on schedule.

For the limbs of the spider to move realistically I needed to implement a rig onto the spider using Inverse Kinematic Handles. Traditionally, when setting up the rig the animator will implement this using 3D software where they can animate and export the model to a game's engine. When you do export a rigged object to a games engine the rig does not export with it, so this needs to be set up in Unity.



SpiderOrange (2) Static
Tag Untagged Layer Default
Prefab Open Select Overrides

Transform
Position X -16.4380 Y 0 Z 5.419473
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Animator
Controller None (Runtime Animator Controller)
Avatar OrangeSpiderAvatar
Apply Root Motion
Update Mode Normal
Culling Mode Cull Update Transforms
Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0 (0.0%)

Bone Renderer
Bone Size ✓ 1
Shape Pyramid
Color
Tripod Size 1
Transforms 0

Rig Builder
Rig Layers
Legs Rig (Rig) ✓

Figure 4.6 – Orange Spider Inspector

I can create the rig by implementing the Unity Rigging and Animation Plugin, which allows me to place different constraints on the limbs, so we are able to manipulate them realistically. For my Orange Spider, I first needed to add a rig builder and bone renderer shown in figure 4.6. The rig builder is the master control for organising the rig because my character is simple and only the legs are being moved and I only need to add 1 rig called "Legs Rig" into the rig builder. However, if my character is more complex and I wanted to also move its arms I would add another rig into the rig builder called "Arms Rig". The bone renderer component is not a necessary part for the implementation to work, however it visually renders all the bones of the object shown in figure 4.7, which allows me to easily see what bones I need when it comes to the implementation of the IK Handles.

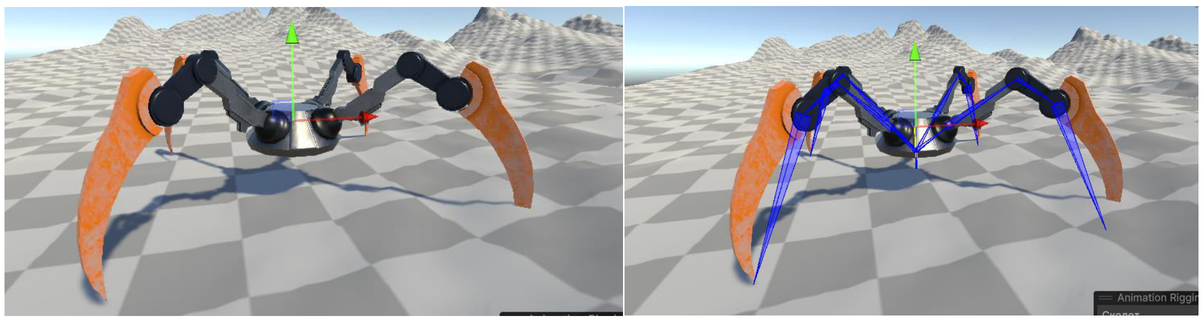
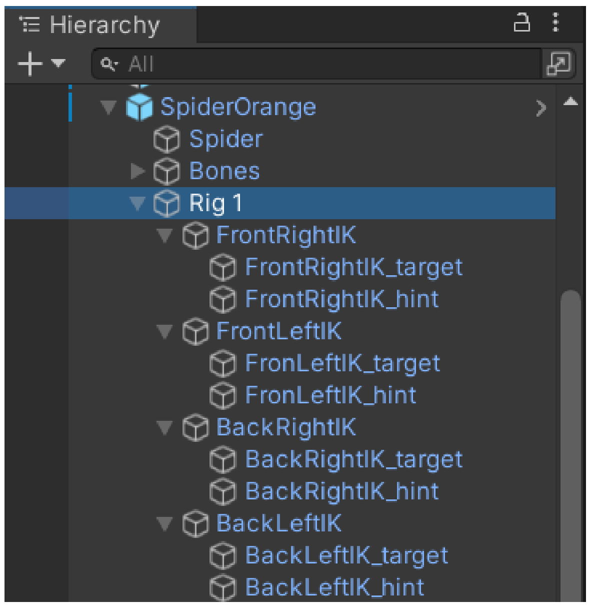


Figure 4.7 – Orange Spider with Bone Renderer Activated and De-activated

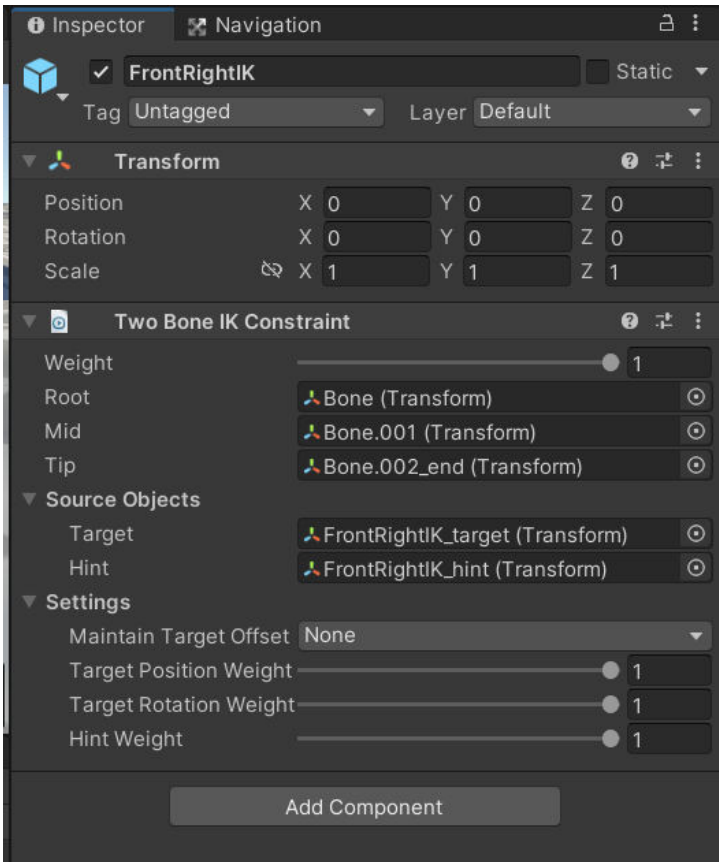
When the rig builder component is created, it also creates a rig section embedded in the Spiders Hierarchy shown in figure 4.8. In the Hierarchy I have created 4 empty objects for each of the spiders' legs where the Inverse Kinematics will be handled.



```
SpiderOrange
    Spider
    Bones
    Rig 1
        FrontRightIK
            FrontRightIK_target
            FrontRightIK_hint
        FrontLeftIK
            FronLeftIK_target
            FronLeftIK_hint
        BackRightIK
            BackRightIK_target
            BackRightIK_hint
        BackLeftIK
            BackLeftIK_target
            BackLeftIK_hint
```

Figure 4.8 – Rig Section inside the Hierarchy

By using the Rigging and Animation Unity plugin, I can choose from a few embedded scripts to setup the IK Constraints. The 2 main scripts, which I can use are the Two Bone IK Constraint and Chain IK Constraint. The Two Bone Constraint shown in figure 4.9 allows me to choose from 3 bones to add the IK handles to the tip of the hierarchy, a middle bone and the root, which is the starting bone in the hierarchy. This works really well for a limb with around 3 to 5 bones in its hierarchy, however, I do not recommend using this script, if the hierarchy has more than 5 bones. For example, the bones imbetween the main hierarchy points (Root, Mid and Tip) deactivates their rotation preventing them from "bending", which creates a very unrealistic look.



Inspector | Navigation

FrontRightIK — Static

Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

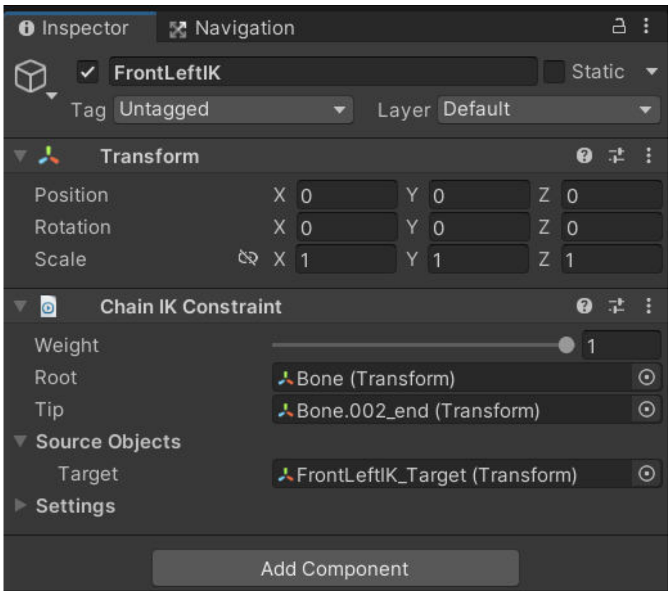
Two Bone IK Constraint

- Weight: 1
- Root: Bone (Transform)
- Mid: Bone.001 (Transform)
- Tip: Bone.002_end (Transform)
- Source Objects
  - Target: FrontRightIK_target (Transform)
  - Hint: FrontRightIK_hint (Transform)
- Settings
  - Maintain Target Offset: None
  - Target Position Weight: 1
  - Target Rotation Weight: 1
  - Hint Weight: 1

Add Component

Figure 4.9 - Two bone IK constraint Implementation

Instead, the Chain IK constraint shown in figure 4.10 is a lot better for a long hierarchy of bones. Implementing the Chain IK Constraint is a lot easier than the Two Bone Constraint as all you need to choose from are 2 bones, the root where the hierarchy of bones start and the tip being the last bone in the hierarchy. This is very different from the 2-bone constraint, which will not lock the rotation of the bones in between the chosen bones, so all the joints will follow the root when it rotates, relative to its parent.



Inspector | Navigation

FrontLeftIK — Static

Tag: Untagged — Layer: Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

Chain IK Constraint

- Weight: 1
- Root: Bone (Transform)
- Tip: Bone.002_end (Transform)
- Source Objects
  - Target: FrontLeftIK_Target (Transform)
- Settings

Add Component

Figure 4.10 – Chain IK Constraint Implementation

For my project I have decided to use the 2 Bone IK Constraint as the legs on the spider use only a few bones allowing for a much easier implementation. Figure 4.11 shows bones, which are allocated to the tip to create the Constraint. Between the mid and tip bone there is another bone, which has not been allocated. This has been done on purpose to prevent any rotation happening, which creates the illusion that the leg is strong and in control of the body.

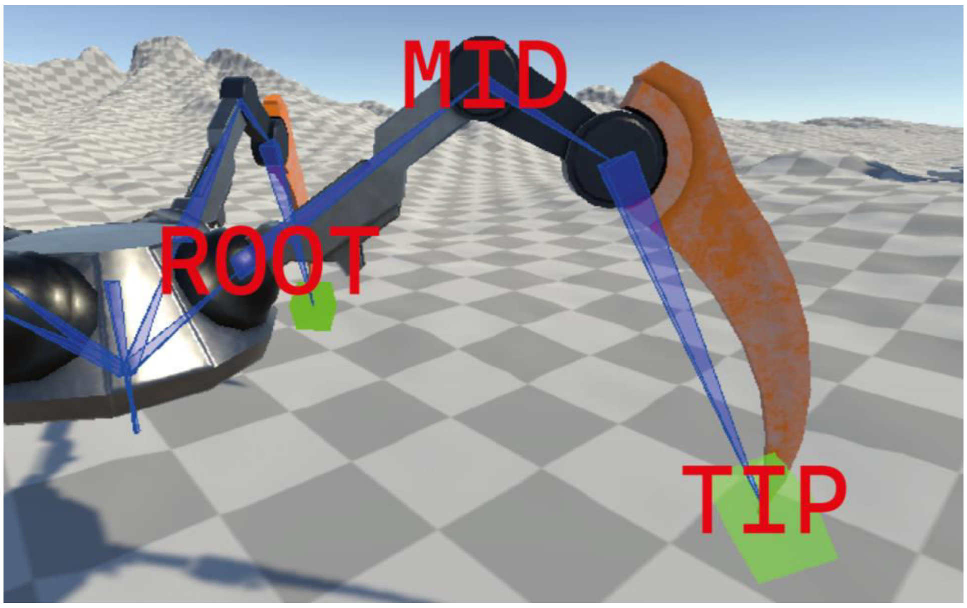
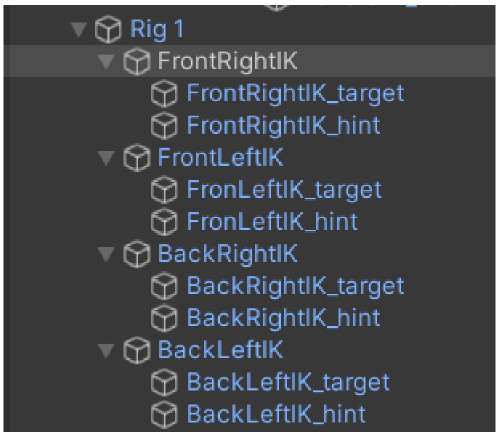


Figure 4.11 – Choosing the Tip, Mid and Root for Two bone IK Constraint

When the bones are all allocated, you are able to automatically setup the Tip Transform, which creates a Target Empty and a Hint Empty shown in figure 4.12. When the world is in Play mode the Tip bone that has been allocated will always equal the position of the Target. So, to make sure the spider starts in a natural position the Target node should be moved to the Tip's position shown in figure 4.13. The hint communicates with the Mid bone and depending on where the hint is, the bone will rotate to that position. To make the spider appear natural at the start the hint should be located above the mid-bone also shown in figure 4.13.



```
▼ Rig 1
  ▼ FrontRightIK
      FrontRightIK_target
      FrontRightIK_hint
  ▼ FrontLeftIK
      FronLeftIK_target
      FronLeftIK_hint
  ▼ BackRightIK
      BackRightIK_target
      BackRightIK_hint
  ▼ BackLeftIK
      BackLeftIK_target
      BackLeftIK_hint
```

Figure 4.12 – Hierarchy of the rig with Targets and Hints

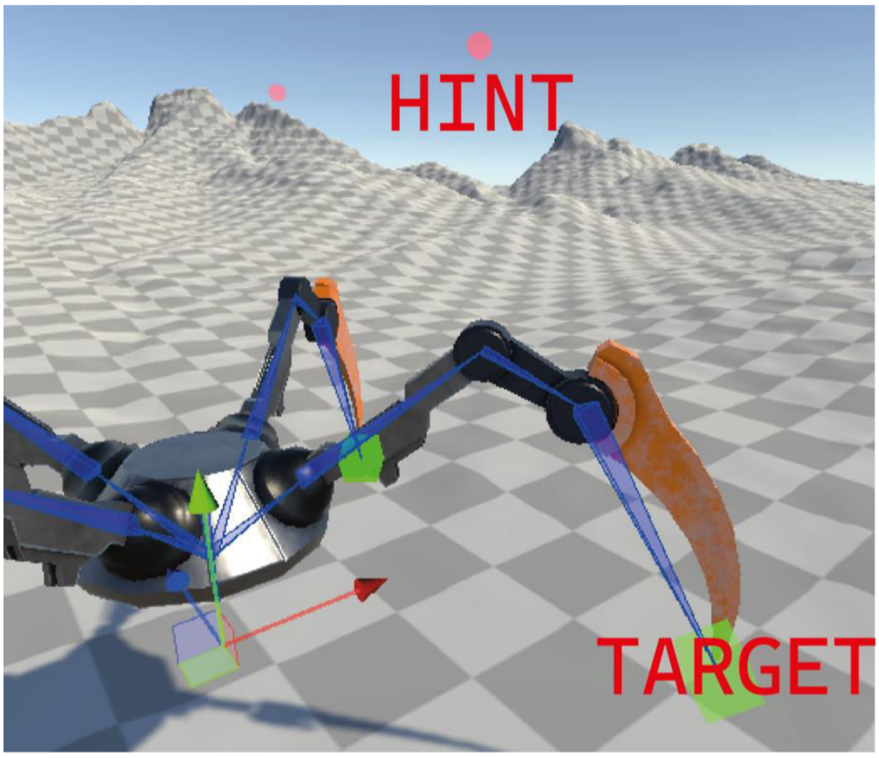


Figure 4.13 – Screenshot to show where the Target and Hint are located

I found positioning the hint very tricky to create a natural position as first. As I did not realise that it is allocated to the position of the mid bone. This creates problems by making the spider's limbs very unrealistic as shown in figure 4.14. This was easily fixed by moving the hints into the correct positions above the mid-bone of the allocated legs.

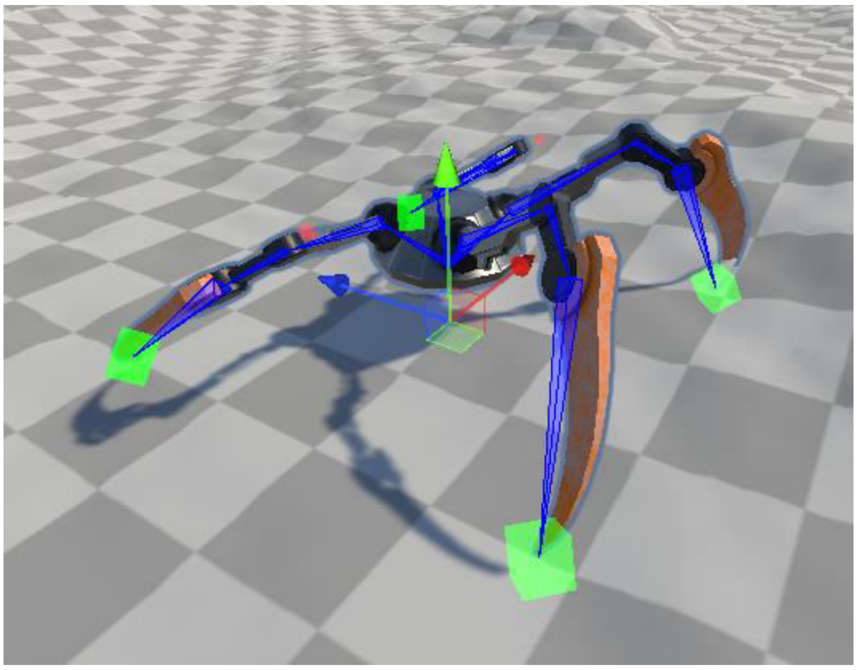


Figure 3.14 – Incorrect Implementation of hint nodes

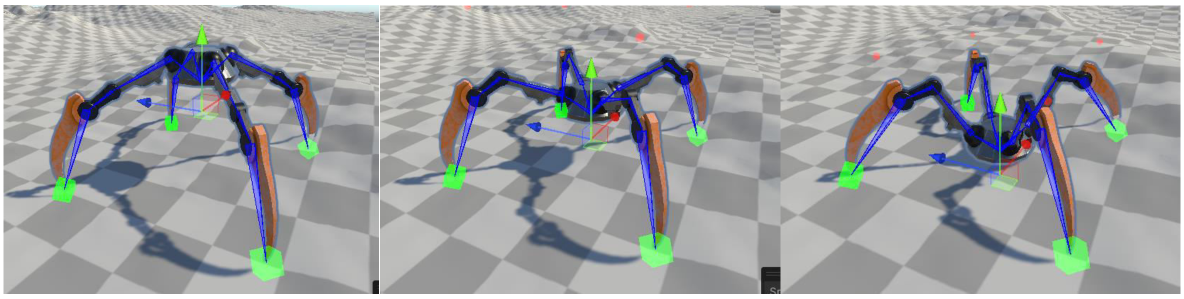


Figure 4.15 – showing Different stages of the Constraints working successfully

The implementation of the Inverse Kinematics can get quite tedious after a while especially if you have a character with multiple legs. As the model is already created with the skeleton you are not able to copy across the rig and IK constraints on each leg. Implementing the rig is not a quick job but needs to be done. However, it is a lot more efficient and easier to make this rig than having to do it traditionally in 3D software. In Unity when you create the constraints it automatically creates the hints and targets. These constraints can be easily positioned to the correct bones by using the transform position function enabled by the plug in. While in the 3D software you need to create the constraints, then create new targets and hints that need to be placed in the right position. Then you need to attach these to the correct IK, which takes a lot longer and can mess up quite easily.

Now that the constraints are implemented successfully, I can move the body of the spider around and the legs will move with it in a realistic manner, as they are constrained to the position of the targets. You can see in figure 4.15 this works in the correct manner.

4.1.2 Sticking the feet to the ground

For the creature to dynamically walk across an environment, the feet need to understand where it can place its foot. The code implemented is attached to each Target object shown in Figure 4.17. it uses a ray cast from the target's position. This is worked out by finding the "skeletonMain.position", which is a public transform with the first parent in the hierarchy of the skeleton shown in Figure 4.16. Then moving the position of the ray across the "footSpacing" that is the position between the Target and the Skeleton Main. Finally using a simple Vector3.down to point the ray towards the floor.

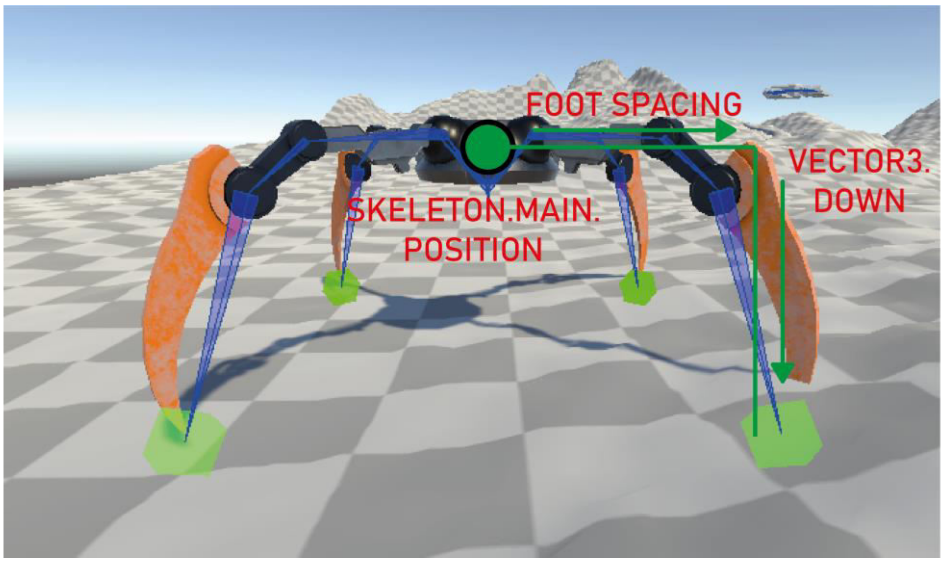
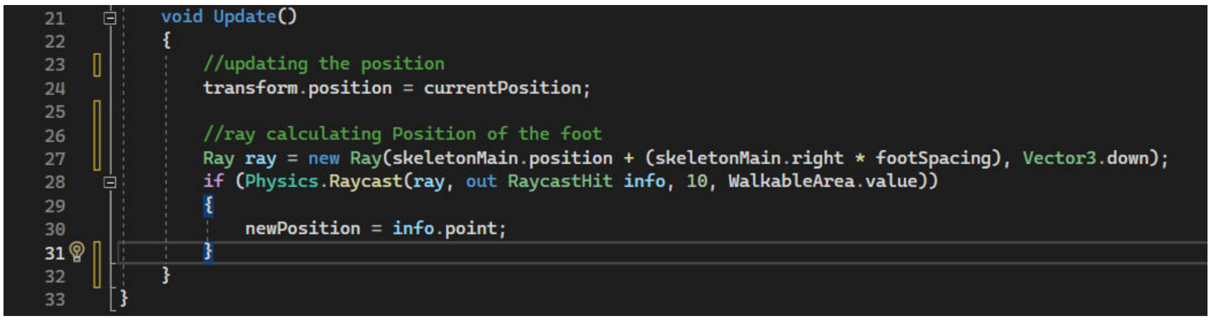


Figure 4.16 – Diagram to show how the Ray paths are created



```
    void Update()
    {
        //updating the position
        transform.position = currentPosition;

        //ray calculating Position of the foot
        Ray ray = new Ray(skeletonMain.position + (skeletonMain.right * footSpacing), Vector3.down);
        if (Physics.Raycast(ray, out RaycastHit info, 10, WalkableArea.value))
        {
            newPosition = info.point;
        }
    }
}
```

Figure 4.17 – Sticking feet to ground Script attached to Target Objects

Now that the ray is created, a simple if statement is used to find out if the ray hits the "WalkableArea", which is the simple terrain I have created. This outputs the "RaycastHit info" that is the position of where the ray has hit the terrain. When the if statement is active it takes the "RayCastHit info" position and makes that equal to the new Position, which is the Target's Position making the target always fit perfectly onto the ground, even when the body of the spider is moved shown in figure 4.18.

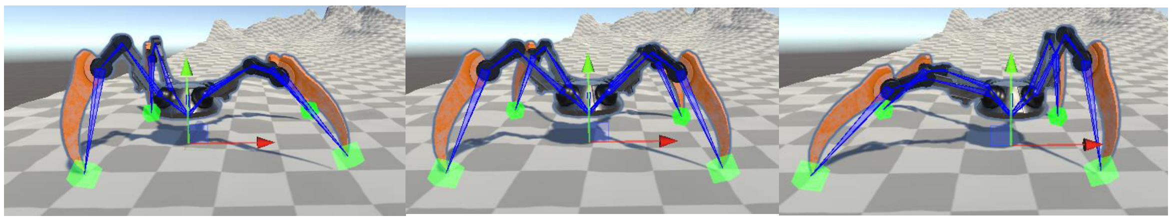


Figure 4.18 – Showing the feet Stick to the ground while the body is moved

4.1.3 – Calculating the Spider’s Next Step Target

Now that the feet are fixed to the ground, we need to calculate the new position of its next step. First, I have created a new empty for each foot, which is parented to the body of the position so when the body moves the Target Points move with it, as shown in figure 4.19.

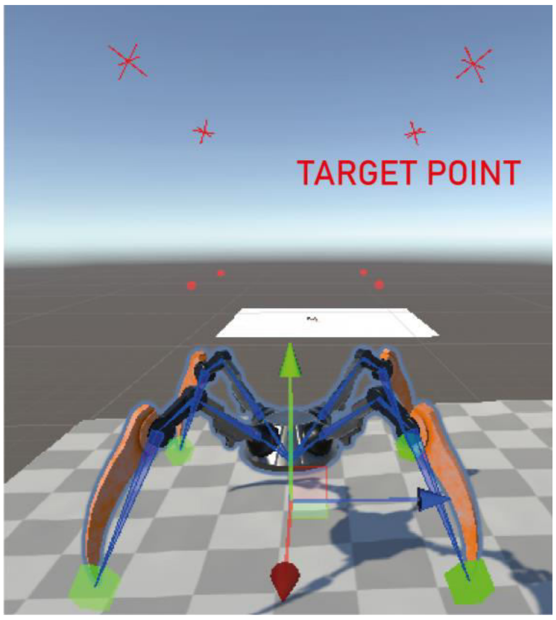
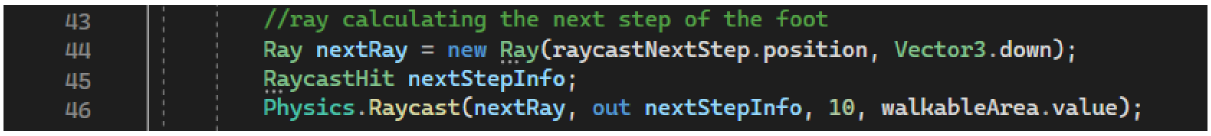


Figure 4.19 – Showing the Next Step Targets Points

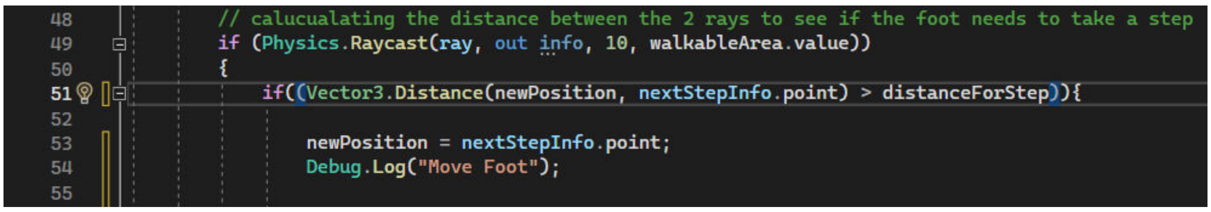
A ray is cast from each Next Step Target point shown in figure 4.20 and the point in the Terrain it touches becomes the new position where a step should be taken.



```
//ray calculating the next step of the foot
Ray nextRay = new Ray(raycastNextStep.position, Vector3.down);
RaycastHit nextStepInfo;
Physics.Raycast(nextRay, out nextStepInfo, 10, walkableArea.value);
```

Figure 4.20 – Ray cast calculating the next step

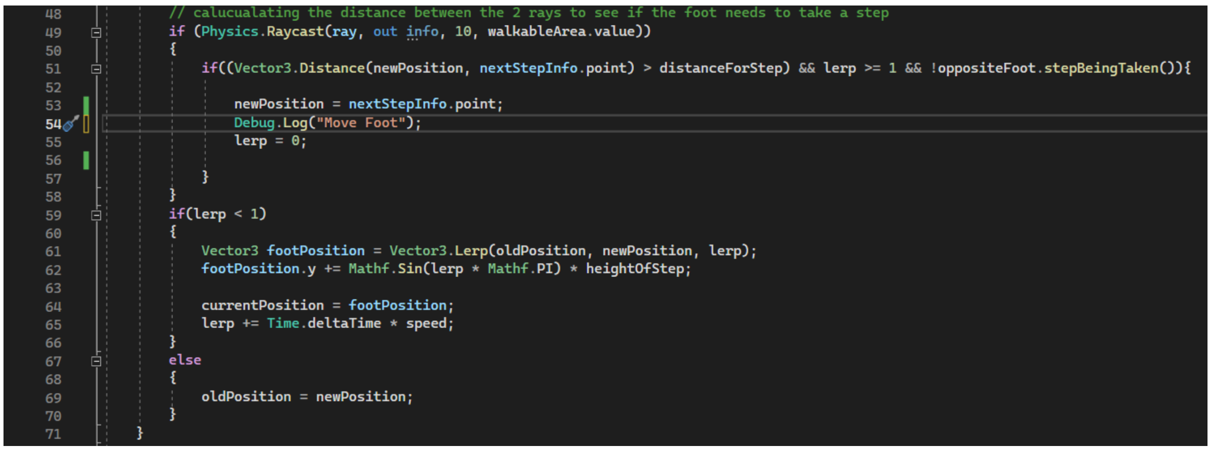
Now that we have the foot’s current position and next step position, we can work out the distance between them by using the float Vector3.Distance shown in figure 4.20. We input the current foot’s position and the next step ray casted point, and we see if that is more than the “DistanceForStep”, which is a public float that I can change for each foot to customise how often the spider steps. If the distance between the 2 points is more than the “DistanseForStep” it makes both points equal, which creates the illusion that the spider is taking a new step.



```
// calucualating the distance between the 2 rays to see if the foot needs to take a step
if (Physics.Raycast(ray, out info, 10, walkableArea.value))
{
    if((Vector3.Distance(newPosition, nextStepInfo.point) > distanceForStep)){

        newPosition = nextStepInfo.point;
        Debug.Log("Move Foot");

```

Figure 4.21 – if Statement to know when to take a new step

Now when the body of the spider moves it also moves the Target points of each foot, shown in figure 4.19 and when they get too far away from the current foot position, the current foot position moves/teleports to the new position located under the Target Points. This works perfectly, however there is no transition between the steps and this is where I use a lerp shown in figure 4.22.



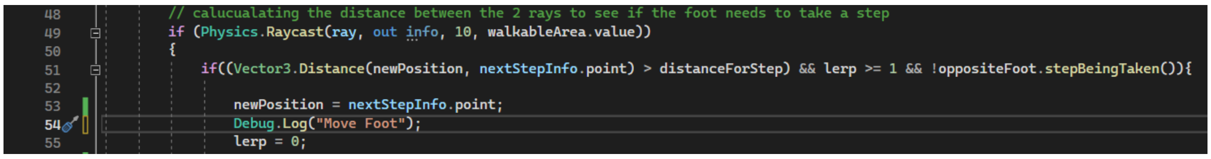
```
        // calucualating the distance between the 2 rays to see if the foot needs to take a step
        if (Physics.Raycast(ray, out info, 10, walkableArea.value))
        {
            if((Vector3.Distance(newPosition, nextStepInfo.point) > distanceForStep) && lerp >= 1 && !oppositeFoot.stepBeingTaken()){

                newPosition = nextStepInfo.point;
                Debug.Log("Move Foot");
                lerp = 0;

            }
        }
        if(lerp < 1)
        {
            Vector3 footPosition = Vector3.Lerp(oldPosition, newPosition, lerp);
            footPosition.y += Mathf.Sin(lerp * Mathf.PI) * heightOfStep;

            currentPosition = footPosition;
            lerp += Time.deltaTime * speed;
        }
        else
        {
            oldPosition = newPosition;
        }
    }
```

Figure 4.22 – Transition of the step being taken

When a next step is needed to activate it sets the lerp to 0, which activates a new if Statement when the lerp is less than 1 that means the leg is ready to be moved. The lerp is a Vector 3 value, which interpolates between two points (Technologies, n.d.), the lerp transitions from the old position of the foot to the new position at the rate of “lerp” that goes from 0 – 1 by using time.deltaTime multiplied by speed, which is a public float that I can customise myself, allowing me to decide the transition speed of each foot individually.

To create an arc-like motion of a foot stepping we take the foots y position and increment a Sin function, which takes the lerp value and multiplies it by PI that makes it transition perfectly in time with the step. Then multiply by the heightOfStep, which is another public float that can be customised to the user’s needs.

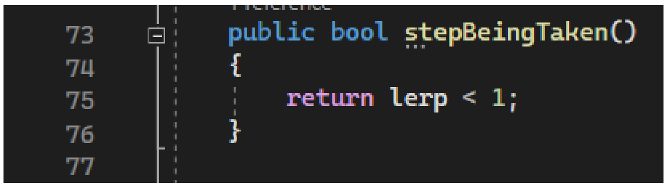
The use of the Sin curve creates a very satisfying step. However, if the spider was moving too fast it would take a step while it was already stepping, which is not a natural walking pattern. I was able to fix this by adding new conditions for when it needs to take a new step. Alongside the foot checking, if it is too far away from the next step position, I am also checking if the lerp is more than or equal to 1 because if the lerp variable was less than one that would mean it is walking.



```
        // calucualating the distance between the 2 rays to see if the foot needs to take a step
        if (Physics.Raycast(ray, out info, 10, walkableArea.value))
        {
            if((Vector3.Distance(newPosition, nextStepInfo.point) > distanceForStep) && lerp >= 1 && !oppositeFoot.stepBeingTaken()){

                newPosition = nextStepInfo.point;
                Debug.Log("Move Foot");
                lerp = 0;
```

Figure 4.23 – Checking to see if foot can take the next step

I am also checking to see if the “oppositeFoot” is not taking a step, because in a normal walking pattern your legs move opposite to each other. At first, all the legs moved simultaneously, when I wanted the front left leg and the back right leg to move first, then the other legs to move and vice versa. oppositeFoot is a public script that takes the code from the other (opposite) foot and checks the public bool shown in figure 4.24, which is only going to return true if lerp is less than 0, when it returns false this allows the foot to take a step.



```
public bool stepBeingTaken()
{
    return lerp < 1;
}
```

Figure 4.24 – Boolean to check if a step is being taken.

4.2 – Finished Walk Cycle

Below are links to my demonstrations:

Procedural Animation Test 1 – https://youtu.be/zTg5eEIQZ3Y

Procedural Animation Walk Cycle - https://youtu.be/IrigbqzcJuw

With the implementation complete, I was successfully able to create a Procedural Animation Locomotion System. However, during testing in the Procedural Animation Test 1 video I noticed the feet were not synced correctly. The legs were not walking in the correct walk cycle, instead the feet on the left side were stepping then both feet on the right were walking, which created an unusual movement.

I was able to fix this by changing the starting positions of each foot by moving the Target points so instead of the character starting in a default standing position, it starts in a position as though already walking. This helped sync up the legs, but after a while, I noticed that when the spider has been walking for a few minutes it goes right back to the incorrect walking cycle. However, this was a very easy fix because I did not match up the opposite legs' scripts correctly. To fix it, I made the front 2 legs opposites. So, the back left leg behind the front left leg were opposites and the same was applied to the right side. This would also work if you had a creature with 20 legs where you made the opposite leg equal the leg in front, but made sure that the front 2 legs are always opposite.

By making these fixes I was able to implement a successful walking Cycle shown in the Procedural Animation Walk Cycle Video.

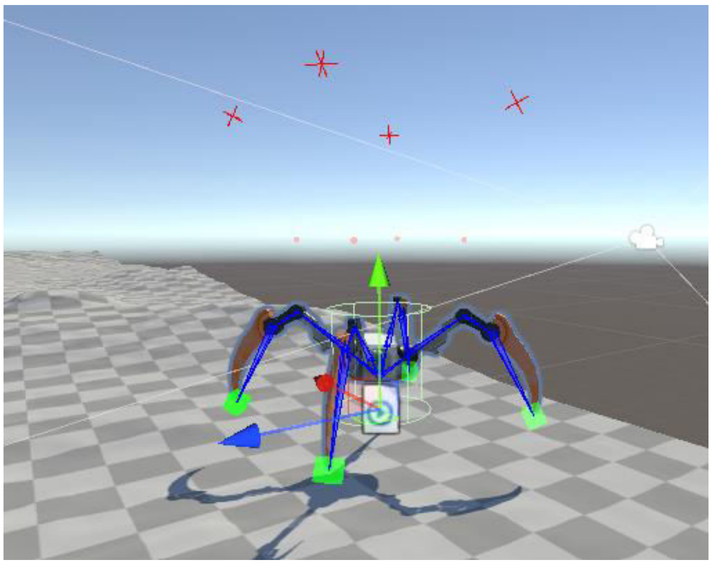


Figure 4.25 – Finished Spider with all gizmos shown

4.3 Testing

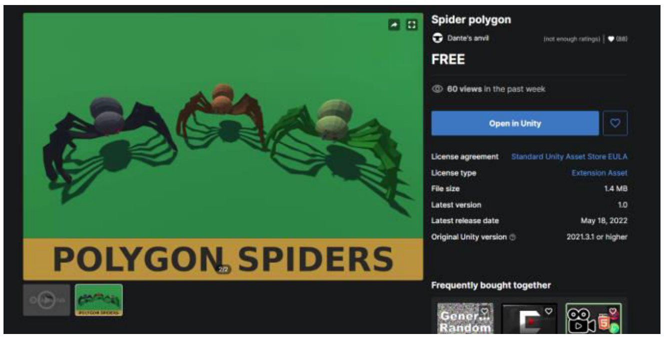
4.3.1 Testing Against Requirements

What Should Happen?	What Did Happen?	Was it Successful	Further Comments
The feet of the character should be fixed on the ground when standing still	The feet of the spider is always fixed onto the ground when it is standing still	This was very successful however when the player mainly pulls the spider's body up into the air the feet get unstuck from the ground. Nevertheless, when the body gets placed back onto the ground the feet become fixed again.	This bug does not break the project, however, in the future it might be feasible to add a constraint to stop the player from lifting the creature up into the air when play mode is enabled.
The animation should work for straight, backward, side to side and curved movements.	The legs animate when the body moves backwards, forwards, side to side and curved movements.	The legs do animate successfully, however, if the characters suddenly turn or move too quickly the legs do get left behind but slowly re-join the body	This is not a bug but shows how the character might adapt with very unrealistic movements. Overall, this shows how successful the project is because of how quickly the rig and animation can get fixed right back together after rapid unnatural movements.
Legs should adapt dynamically to different heights and slopes of the terrain.	The legs dynamically adapt and can step onto different heights and slopes of the terrain	The legs can dynamically change height depending on the terrain, however, if the body is on a high ledge and the legs cannot find a place to land, they will magically start flying backwards.	The legs cannot find a place to land because the body is constantly sitting straight, the legs will be able to fit on the floor all the time if the body rotated in the direction of the hill it is standing on to let the feet down.
Inverse Kinematics is used on the character to create a more natural movement.	Inverse Kinematics has been applied to the character's limbs to create natural movement	I have applied a two-bone IK constraint to each one of the legs that allow the end foot bone to move the entire legs, which help create make the movement of natural	This works perfectly and I have also got the choice of using the Chain IK Constraint as well if my legs had many bones in a limb
The character's joint should have Minimum and maximum angle	No, the Character's joints do not have minimum and	When I started this project, I thought I needed to constrain	The limbs can still move into each other however because the

restrictions to prevent the joints from moving into each other.	maximum angle restrictions so the limbs can move into each other	how much the angles could move to prevent any irregular movement but when creating my project, I realised that I did not need to implement this for my project to be successful.	limbs are being animated to move out rather than closer to the body the limbs should never intercept or prevent any irregular movement.
It should be able to animate dynamically depending on the speed of the character.	When the character speeds up the animation also speeds and when the character is slowed the animation is slowed.	Because the character scans the floor to find out when to make the next step it can do this at most speeds. However, when the character moves too fast the animation cannot catch up with it preventing proper steps from being taken.	The Character can move as fast as the animation takes. If the character moves faster than the step animation the feet will fall behind but slowly catch up. This bug can be fixed by working out your character's personality, if the character was a small insect it would move quickly so the animation would need to be sped up and if the character was a 100ft tall giant the animation will need to be slowed down.

4.4 Key Frame Animation vs Procedural Animation

To be able to see the quality of the Procedural Animation I have created I wanted to compare what I have created with another animation which uses Key Frames. for this experiment I have used the same model of a spider shown in figure 4.26, one of the spiders' uses its predefined animations built into it and the other uses the implementation of Procedural Animation which I have created. Both spiders use a simple nav mesh agent with movement code which randomises where they move around the terrain. The terrain consists of a flat plain with a few elevated surfaces to see how they might react.



Spider polygon
Dante's anvil (not enough ratings) | (88)
FREE
60 views in the past week
License agreement Standard Unity Asset Store EULA
License type Extension Asset
File size 1.4 MB
Latest version 1.0
Latest release date May 18, 2022
Original Unity version 2021.3.1 or higher
Frequently bought together

Figure 4.26 – Spider models used for experiment

Key Frame Animation vs Procedural Animation Video - https://youtu.be/cboMBkfTsPc

The main differences in the models and animations are how they interact with the environment when the key frame animation walks near the elevated floor its legs clip through the blocks, which makes it very unbelievable this is because the key frame animation is not checking to see where it legs can be placed as it is repeating the same defined animation. However, on the procedurally animated spider when it gets near an elevated block instead of the legs clipping through, they get placed perfectly onto the top, this looks a lot more realistic then the keyframe as it shows the spider can effectively interact with the game world.

Both of the spiders move around the game world using the same code, however they both appear to walk very differently, because of the keyframes repeating animation the spider looks like it is in a trance and ice skating around on the other hand because the procedural animation uses inverse kinematics this adds a bit of randomness to how the bones might be effected creating diverse set of leg steps depending on if the feet are on elevated platform or when its turning. When the procedural animated spider turns its can successfully place its fit in a realistic manner when the body of the spider is rotating however as the key frame spider uses the same animation when it's turning it still using the same walking animation decreasing the realism as you wouldn't be walking in place while turning.

Overall I feel that the procedurally animated spider is a greater improvement to the model compared to the keyframe which clips through objects, repeats animation and fails to interact with the game world effectively while the Procedural Animation doesn't clip through objects but interacts with the environment successfully its animation never repats due to the use of Inverse Kinematics creating dynamic movements and finally the procedural spider feels apart and comfortable with the game world as it can collide and move around effectively and realistically.

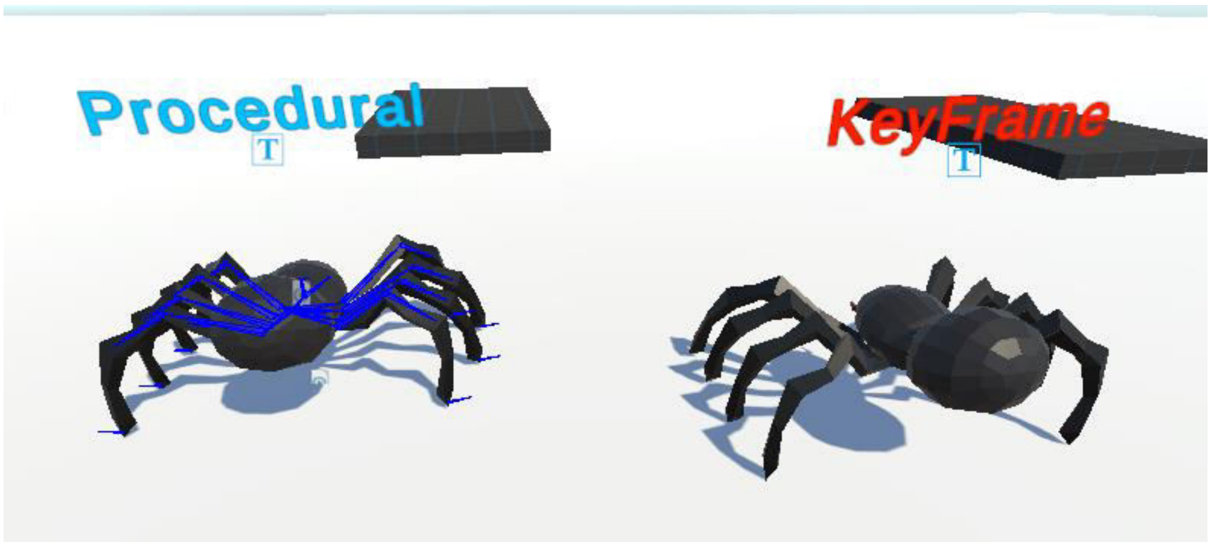


Figure 4.27 – Both spiders 1 implement with key frame animation the other with Procedural animation

Section 5: Critical Review

5.1 The Good

5.1.1 Implementation of the Inverse Kinematics

The skeleton and Inverse Kinematics were a very important part of the project, which needed to be implemented to a very high standard because if the limb constraints were not implemented correctly this will completely change how the character would walk and create a very unnatural look. I believe the Inverse Kinematics on the character have been implemented very well. The use of targets allows me to move the legs in a lifelike manner and the hints being used as struts to prevent the legs rotating unnaturally. The only downfall to this is for each leg I would need to create a separate constraint, which becomes quite tedious if your bones are not named properly or if the character has hundreds of legs, which need to be animated. This cannot be prevented, however can become easier if you have a centipede-like creature shown in figure 5.1 where only one pair of legs need to be implemented with IK then you are able to copy and paste each module backwards and join up the bones in the hierarchy.

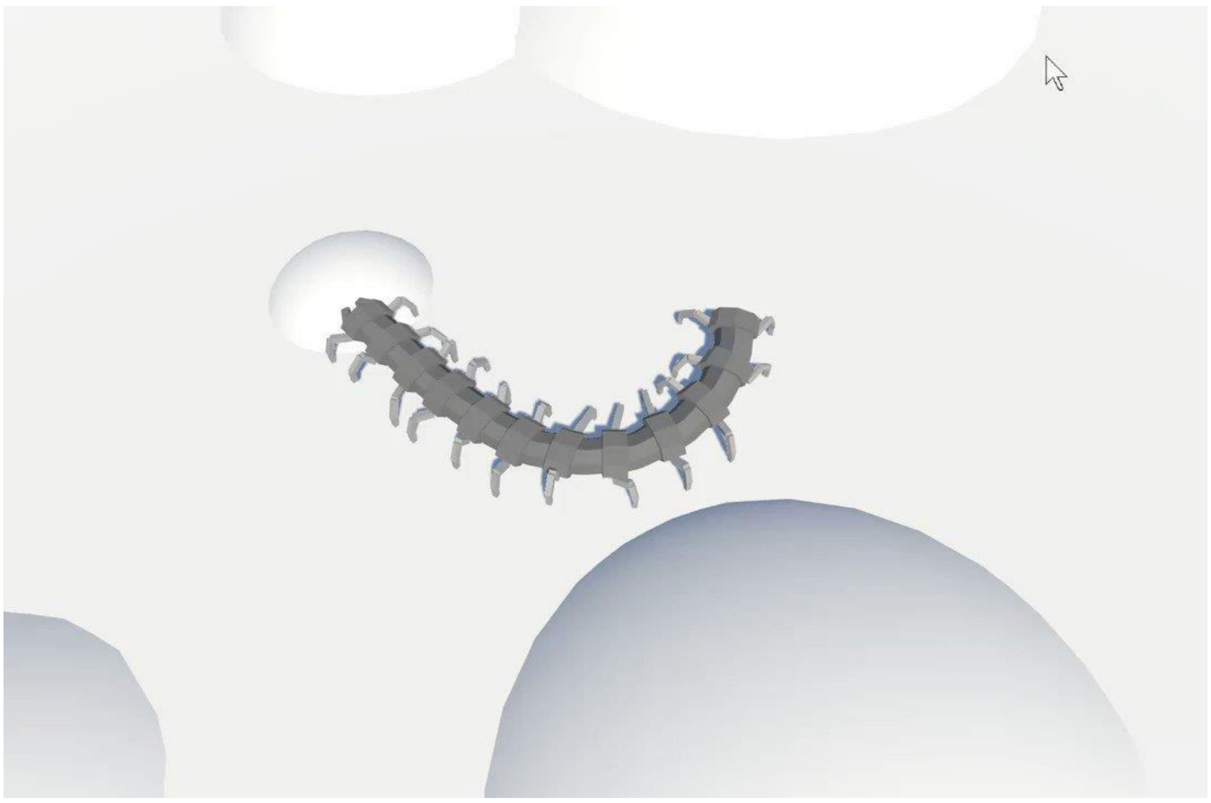


Figure 5.1 – Centipede Creature to show how IK can be implemented

5.1.2 How the feet dynamically move around the environment.

One of the main features of Procedural Animation over traditional animation is the added benefit of the character being able to scan and adapt to the environment. I have been able to implement this very successfully by using ray casts. The rays are being shot down at the environment allowing the feet to know the height of the ground, making its feet sit correctly on top. However, if the ground is too far away from the feet being placed onto it, these glitch the feet, making them fly in the air. This can be fixed by rotating the body down in the direction of where the feet should be fixed allowing them to be placed.

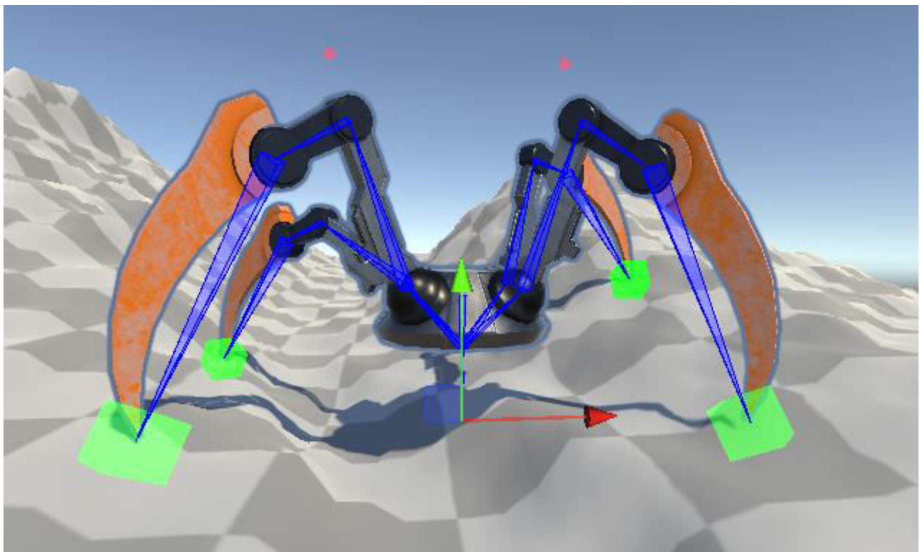


Figure 5.2 – Spiders feet fitting perfectly on different elevated ground

5.1.3 Codes Compatibility with different models of characters

The way the code has been handled allows it to be used on multiple different models and still work effectively. For example, to get the feet to sit on the ground it takes the main skeleton and calculates the spacing between the feet, which gets the feet position, allowing it to be placed correctly so it works out the calculations depending on the different models making it universally accessible. The overall fundamentals of the implementation and the code works correctly in Unity, however I am not sure how it will differ in other game engines. I believe the overall core rules will still apply but as I was using the Unity Rigging and Animation plugin this will differ per engine and some might not have the ability to create automatic IK handles, resulting in more work, as this would all need to be coded.

5.1.4 Efficiency of Implementation over Traditional Key Frame Animation

Traditional Keyframe Animation is a long and tedious process, which I have successfully cut in half by creating a locomotion system using Procedural Animation. I was able to create a more dynamic locomotion system where the feet scan the environment to understand the height of the ground, which is a very big improvement over Traditional keyframe animation, which uses a hand-crafted animation that plays on repeat, creating an unrealistic movement with the chance of the characters model clipping through the ground. The Process of Procedural Animation cuts out the implementation of creating a rig for the characters, creating keyframes, exporting the animations needed and setting up which actions correspond to which animation.

5.2 The Improvements

5.2.1 Create a foot manager

On each foot you can change the step speed, height and distance to take a step. This works perfectly fine for what I have created, however if the character had 10+ limbs, for each limb the user would have to type in the correct attributes, which becomes a very tedious job. One way this could be fixed is by creating a property manager where it takes all the public values, which the user is able to change and puts them all into one object and script, so when the user changes the attributes on the manager it will change it all for each individual foot. This will help prevent the user inputting incorrect values onto feet and is able to test different values with ease.

5.2.2 Rotate and add bounce

The main components of the characters being animated are the legs, which get dragged along by the body. The body stays in a fixed location no matter the location of the legs. What I would like to happen is for the body to rotate in relation to the position of the feet, as this will help with the realism and allows the feet to always touch the ground. This could be implemented by finding the difference between the left legs and right legs and applying this to the body's location. This might be difficult to implement at the moment due to each foot having an individual script, so either a new script that creates the Procedural Animation for all the feet in one object is created to store the feet location or another script is implemented just for the body's rotation, which takes all the separate feet scripts and uses their location to apply the rotation.

5.2.3 Placement of Next Step Target Points

Shown in figure 5.3 are the target points, which are floating in the air and attached to the spider's body. These points are ray casted down to indicate the foots next step. This works well on a plain bumpy terrain, however because the target points are real objects in the world space they can get pushed through walls and other places where the foot cannot be placed, making the feet fly in the air as they have no place to fit. This can be fixed by adding colliders to the target points so when they collide with the walls they do not go through, allowing the foot to be placed next to the wall. However, when the target points collide this could push them out of alignment, so by adding an idles position for the feet and target point, when the character stands still for a few seconds this will allow the feet and target points to be aligned perfectly.

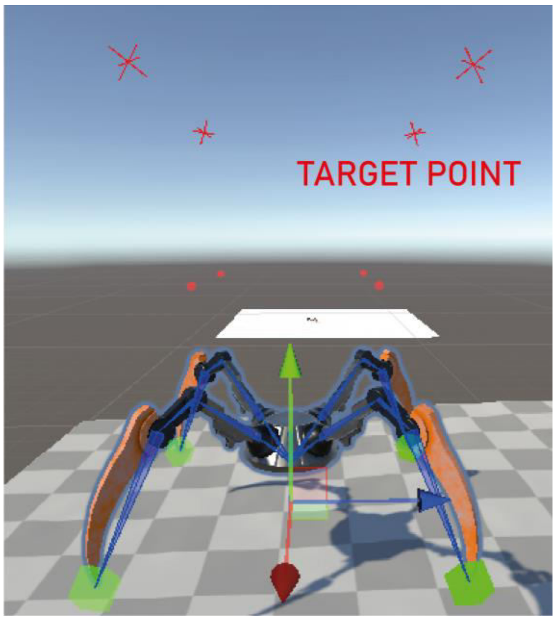


Figure 5.3 – Target Points on spider to work out next step

5.2.4 Adding Sound and Particle Effects

By adding sound and particle effects, this will help improve the dynamic aspect of the character. When a leg is moving, I could add a mechanical movement sound to indicate the legs are operational and made of metal and wires, this will help improve the realism that the character is moving. However, it might get quite annoying if the sound plays every time a leg moves, so this sound should be subtle and/or fades to avoid repetitiveness. Particle effects will also help with the realism, when the foot is placed onto the ground, dust particles could fly around on the foot's location to indicate the leg is heavy and help show the player that the character is moving around.

Section 6: Conclusion

The implementation of Procedural Animation has helped me achieve a more dynamic and realistic animated character compared to traditional Keyframe Animation. This approach allows for a more efficient and customisable way to create a locomotion system, which lessens the need for an animator's skill set. Through the spider model the animation has been implemented very successfully with the spider able to scan the environment to know where and when to place its feet. It can be customised individually to change the character's behaviour through step height, step length and speed.

Although the animation I have created is quite simple it creates a very immersive creature that feels comfortable and fluid in the game world. I believe the implementation I have used to create a procedurally animated character can be improved significantly with multiple different animations happening on a multitude of different types of character, at the same time such as:

- Picking up objects
- Using Objects (Swords, keys etc)
- Head turning towards objects
- Facial expressions changing
- Rolling animation
- Crouching
- Swimming

While comparing the key frame animation with the procedurally generated animation I can see that procedural animation is a much more successful form of animation which is able to interact with the game world very realistically, preventing any object clipping which appears quite often with key frame animation. Although procedural animation is very successful and can replace some parts of the key frame implementation and I believe it would be very difficult to do on a very complex model with many moving parts. I feel there could be too many parameters and rules needed to be implemented to create a realistic and dynamic model. However, using both procedural and keyframe in the same model would be a much better way to get very good results. For example, you could use procedural animation to animate the legs of a model as this is the main feature which interacts with the game environment. Then use keyframe animation to animate the upper half of the model which could be animated to attack enemies, pickup objects or interact with objects without effecting the Procedural animation creating a realistic and exciting animation.

Overall procedural animation can be implemented and used very successfully to allow a model to interact very dynamically and realistically around the game world while also creating a very effective and satisfying animation, which can be a much better improvement then using keyframe animation. While procedural animation is best used in combination with the key frame animation allowing for the model to dynamically interact with the environment while also having a multitude of animations which can work together flawlessly.

References

Websites

Petty, J. (2018). *What is 3D Rigging For Animation & Character Design?* [online] Concept Art Empire. Available at: https://conceptartempire.com/what-is-rigging/.

Technologies, U. (n.d.). *Unity - Scripting API: Vector3.Lerp*. [online] docs.unity3d.com. Available at: https://docs.unity3d.com/ScriptReference/Vector3.Lerp.html.

Pêcheux, M. (2022). *Creating procedural animations in Unity/C#*. [online] CodeX. Available at: https://medium.com/codex/creating-procedural-animations-in-unity-c-8c5c2394739d [Accessed 25 Mar. 2023].

Zucconi, A. (2017). *An Introduction to Procedural Animations*. [online] Alan Zucconi. Available at: https://www.alanzucconi.com/2017/04/17/procedural-animations/.

Polygon Treehouse. (n.d.). *Anatomy of a Walk Cycle - Part 1*. [online] Available at: https://www.polygon-treehouse.com/blog/2019/1/30/anatomy-of-a-walk-cycle.

www.youtube.com. (n.d.). EPIC Boss battles in UNITY! [Procedural Animations]. [online] Available at: https://www.youtube.com/watch?v=S36CH4cbW1A&ab_channel=SpeedTutor [Accessed 6 Nov. 2022].

Bhatti, Z., Ismaili, I.A., Zardari, S., Malik, H.A.M. and Karbasi, M. (2016). Procedural Animation of 3D Humanoid Characters Using Trigonometric Expressions. Bahria University Journal of Information & Communication Technologies (BUJICT), [online] 9(2). Available at: http://ojs.bahria.edu.pk/index.php/bujict/article/view/51/43 [Accessed 6 Nov. 2022].

Horswill, I.D. (2009). Lightweight Procedural Animation With Believable Physical Interactions. IEEE Transactions on Computational Intelligence and AI in Games, 1(1), pp.39–49. doi:10.1109/tciaig.2009.2019631.

Antti Ruuskanen Inverse Kinematics in Game Character Animation Bachelor of Business Ad- ministration Information Technology. (2018). [online] Available at: https://www.theseus.fi/bitstream/handle/10024/156571/Ruuskanen_Antti.pdf?sequence=1&isAllowed=y#:~:text=Inverse%20kinematics%20works%20in%20the [Accessed 6 Nov. 2022].

Unity Technologies (2019). Unity - Scripting API: Physics.Raycast. [online] Unity3d.com. Available at: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html.

Technologies, U. (n.d.). Unity - Manual: Inverse Kinematics. [online] docs.unity3d.com. Available

at:https://docs.unity3d.com/Manual/InverseKinematics.html.

GraftPublisherMay 13, K. and 2019 (2019). The 12 principles of animation in video games. [online]

Game Developer. Available at: https://www.gamedeveloper.com/production/the-12-principles-of-animation-in-video-games.

Team Houdini. (2019). *What are the benefits of animating procedurally?* [online] Available at: https://teamhoudini.wordpress.com/2019/02/01/what-are-the-benefits-of-animating-procedurally/.

docs.unity3d.com. (n.d.). *Animation Rigging | Animation Rigging | 1.0.3*. [online] Available at: https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.0/manual/index.html.

Coron, T. (2022). *Understand Disney's 12 principles of animation*. [online] Creative Bloq. Available at: https://www.creativebloq.com/advice/understand-the-12-principles-of-animation.

A. D'Souza, S. Vijayakumar and S. Schaal, "Learning inverse kinematics," Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180), Maui, HI, USA, 2001, pp. 298-303 vol.1, doi: 10.1109/IROS.2001.973374.

G. Salazar, X. Luo, A. A. Navarro Newball, C. Zúñiga and C. Lozano-Garzón, "Multiple Character Motion Adaptation in Virtual Cities Using Procedural Animation," 2019 International Conference on Virtual Reality and Visualization (ICVRV), Hong Kong, China, 2019, pp. 223-226, doi: 10.1109/ICVRV47840.2019.00053.

C. -H. Liang, P. -C. Tao and T. -Y. Li, "IMHAP – An Experimental Platform for Humanoid Procedural Animation," Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2007), Kaohsiung, Taiwan, 2007, pp. 345-348, doi: 10.1109/IIHMSP.2007.4457560.

Z. Bhati, A. Shah, A. Waqas and H. A. M. Malik, "Template Based Procedural Rigging of Quadrupeds with Custom Manipulators," 2013 International Conference on Advanced Computer Science Applications and Technologies, Kuching, Malaysia, 2013, pp. 259-264, doi: 10.1109/ACSAT.2013.58.

Guerraz, S., Perbet, F., Raulo, D., Faure, F. and Cani, M.P. (2003). *A procedural approach to animate interactive natural sceneries*. [online] IEEE Xplore. doi:https://doi.org/10.1109/CASA.2003.1199306.

Yang, P.-C., Funabashi, S., Al-Sada, M. and Ogata, T. (2022). Generating Humanoid Robot Motions based on a Procedural Animation IK Rig Method. *2022 IEEE/SICE International Symposium on System Integration (SII)*. doi:https://doi.org/10.1109/sii52469.2022.9708820.

Appendices

Date of Meeting	What was Discussed
21/10/2022	- What the module is about - How to write an interim report - Due dates for all deliverables
12/12/2022	- Discussed project with supervisor and second reader. - Got feedback on the interim report. - Discussed the next steps
28/02/2023	- Showed supervisor the progress made with project. - Told to start work on the report. - Given a due date for report to be completed by end of March. - Discussed to create game to show of project
15/03/2023	- Discussed if the final year report needed a project management section
28/03/2023	- Showed supervisor my first draft - Gave feedback to: - Check speeling of Gantt Chart - Fix white space and layout of report - Move research section to before the production description - Link the research and product description together and how this influenced my work - Add more to the critical review section - Add a conclusion to state what I learned through the process
2/5/2023	- Showed supervisor my final draft - Gave feedback to: - Fix white space - Add more references - Add emails to supervision log - Delete the individual assessment criteria in report