DEGREE IN DESIGN AND DEVELOPMENT OF VIDEOGAMES.

VJ 1241 - BACHELOR'S THESIS

3D to 2D transformation of environment and characters in videogames

Author:

Francisco Miguel Morales Sánchez

Tutor of the project:

José Vicente Martí Avilés

07/07/2020

Acknowledgments

First of all, I would like to thank my Final Degree Project supervisor, José Vicente Martí Avilés, for his guidance in creating this memory, I am not good at creating this type of document, but he has given me his advice to improve it increasingly.

Also, I would like to thank him for some of the talks that we had in his office about what I could do with the TFG, because at the beginning I couldn't decide on any idea and when this occurred to me, he supported me from the beginning.

I would also like to thank him for the LaTex template recommendation that I have relied on to create this document.

I would also like to thank all the teachers I have had, that although there are some with whom I have not had much affinity. They all ended up teaching me useful things that I can use in my professional and personal life.

INDEX

1 - Introduction

1.1 - Work Motivation

There are two types of video games, 2D and 3D games, each one has its own visual and playable characteristics and both can become famous games, however, both skills are very important in the video game industry, but there are people who have more facility creating 3D modeling, the question and motivation that arose with this thought was

What if there was some way to convert 3D modeling to 2D?

It is an idea that could bring great benefits to indie studios:

1- Allow looking at the character or scenario from any angle / perspective

2- Easily change animations that we don't like without having to do everything from the beginning

This idea was also inspired by the video game "Dead cells" where only one person was in charge of the entire artistic section, where this person created 2D art from 3D art (Figures 1 and 2).

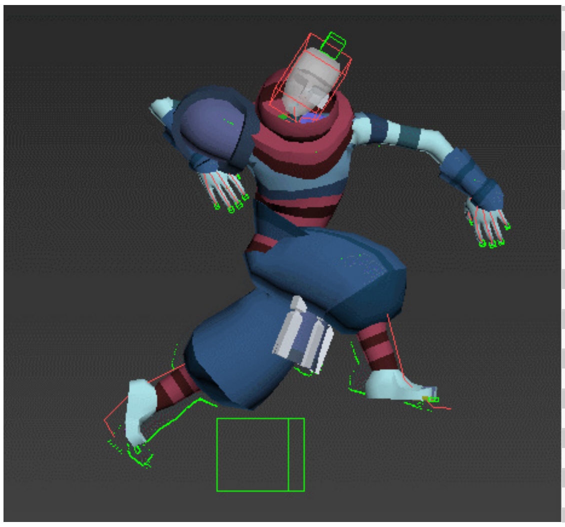


Figure 1: Dead Cells videogame pipeline image, 3D character modeling.

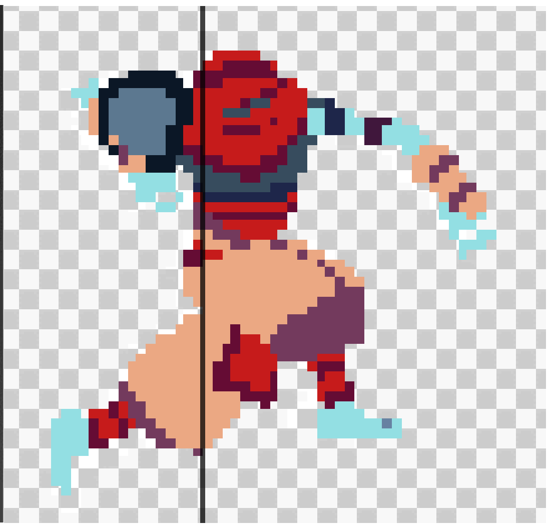


Figure 2: Dead Cells videogame pipeline image, 2D visual transformation performed by the head of the artistic section.

1.2 - Objectives

- To use different programs and techniques to convert 3D character and scene modeling to 2D character and scenario modeling
- To create several Demos that implement this technique with different uses, to observe its potential and possible errors

1.3 Environment and Initial State

There are several options in the modeling environments, "3ds Max", "Blender", "Maya", "ZBrush", etc.

All these programs are very powerful, any of these would help us to achieve the objective of this project, however, we are going to focus on "3ds Max" and "Blender", because both have been taught throughout the university.

Nowadays, these two programs are the most widely used modeling environments worldwide and have been taught throughout the university, making any student familiar with them.
Both are good options, but only one program will be used for the development of the project.
"Blender" has been chosen because:

- It is for free ("3ds max" is a paid program)
- It is a program that since its last update 2.8., is increasingly booming.
- Any student is acquainted with it, because it is the most used in GameJam.

With video game engines there are also several options, "Unity", "UnrealEngine", "GameMaker", etc.
Of all these engines, the one that interests us most is Unity, the reasons are several:

- It is the videogame engine taught throughout the university.
- It is used in GameJam's, so any student is familiar with it.
- It is also for free.
- It is a very powerful game engine, it is possible to do almost anything with it.

In summary:

As development environment, Unity (version: 2018.4.12f1) has been chosen as game engine and Blender as 3D environment.

1.4 Concepts

- **Sprite:**

It is a two-dimensional bitmap that is integrated into a larger scene, most often in a 2D video game.

- **SpriteSheet:**

It is an image that consists of several smaller images (sprites) and/or animations. Combining the small images in one big image improves the game performance, reduces the memory usage and speeds up the startup time of the game.

- **Rigging:**

It is needed for animation and physics simulations in Blender. Rigging gives a solid model articulation at certain points in certain ways, usually to simulate a biological skeleton of some kind. These articulations allow to create the model animation (manually or with Mixamo)

- **FBX and OBJ:**

File format. These formats are used to provide interoperability between digital content creation applications.

- **GameObject:**

GameObjects are the fundamental objects in Unity that represent characters, props and scenery. They do not accomplish much in themselves but they act as containers for Components, which implement the real functionality.

- **Instance:**

In programming, an instance is usually created as a copy of some template. If you are cloning a GameObject then you can also optionally specify its position and rotation (these default to the original GameObject's position and rotation otherwise). If you are cloning a Component then the GameObject it is attached to will also be cloned, again with an optional position and rotation.

- **Prefab:**

Unity's Prefab system allows to create, configure, and store a GameObject complete with all its components, property values, and child.
The Prefab Asset acts as a template from which to create new Prefab instances in the Scene.

- **Collider / Collision:**

A collider is invisible, does not need to be the exact same shape as the GameObject's mesh. When collisions occur (between two or more GameObjects), the physics engine calls functions with specific names on any scripts attached to the objects involved.

2 Planning and Resources Evaluation

2.1 Initial Planning

- 10h to get all the 3D models
- 25h to add the necessary animations in the 3D models
  - 10h to learn how to add the animations
  - 15h to add all the animations in the 3D models
- 50h to convert 3D models to 2D sprite sheet
  - 30h to learn how to do the different types of conversion, because there are at least 3 different methods to do that
  - 20h to do all conversions once the method is selected
- 25h to obtain the "normal maps" that will be used for dynamic light using the "sprite sheet"
  - 10h to learn how to obtain the "normal maps"
  - 15h to obtain all the normal maps
- 10h make the scene
- 20h to create the dynamic lighting of the scene
  - 10h to learn how to make the dynamic lighting
  - 10h to make the dynamic lighting
- 80h to programming playable mechanics
- 50h for the realization of memory
- 20h to the presentation
- 10h for unforeseen

Figures 3 and 4 of Section "2.2" shows the planning followed. However, there are no significant changes in the amount of time spent.

The only difference is that each part of the project has been interspersed with others on different days, but at the beginning of the project it was believed that each part of the project would be carried out independently

2.2 Planning followed

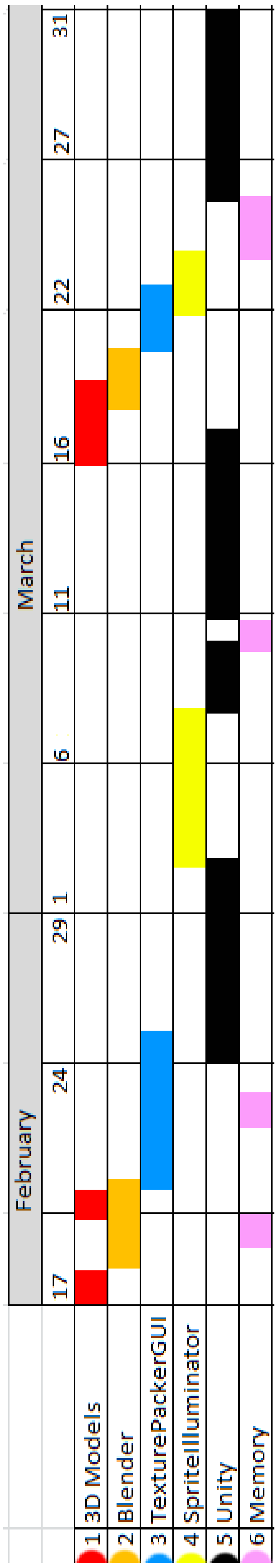


Figure 3: First part of the Gantt chart

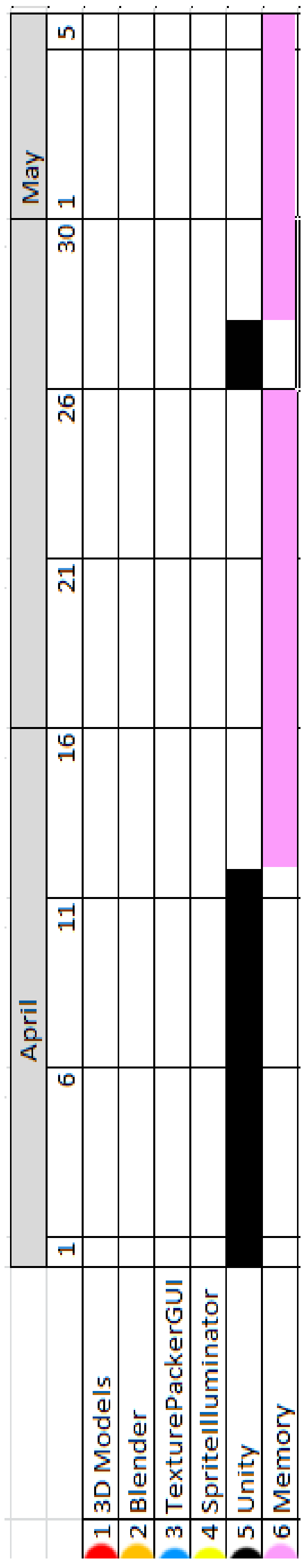


Figure 4: Second part of the Gantt chart

2.3 Resources Evaluation

Hardware:

The computer used for this project has had the following characteristics:

- Intel Core i7-9700K
- Graphic Card gtx 1660 Ti
- 32GB RAM

Although a powerful computer has been used, as long as the computer is capable of using "Unity", there should be no problem. Also, keep in mind that "Blender" on a low budget computer will take longer to render, but it is still possible to be used this program.

Software:

- Unity:

It is a cross-platform game engine developed by "Unity Technologies".
The engine can be used to create 3D, 2D, VR, and AR games, as well as simulations and other experiences.

- Blender:

It is a free and open-source 3D computer graphics software toolset with multiple purposes (modelling, rigging, visual effects, animating, etc).

- TexturePackerGUI:

It is a program developed by "Codeandweb", designed for the use of 2D images, creating SpriteSheets automatically and easily. This program allows compatibility with other programs such as "Unity", "Photoshop", etc.

- Sprite Illuminator:

It is a program developed by "Codeandweb", designed for creating 2D normal maps.

- Mixamo:

It is a 3D computer graphics technology, use machine learning methods to automate the steps of the character animation process, including 3D modeling to rigging and 3D animation.

- Photoshop:

It is a popular image changing software package. It is widely used by photographers for photo editing (fixing colors, reducing noise, adding effects, etc). The software is made by the company "Adobe Systems".

- Github:

It is an online repository that offers free accounts that are commonly used to host open source projects.

- Image Magick:

It is a free and open-source software suite for displaying, creating, converting, modifying, and editing raster images.

2.4 Economic cost

Once the used hardware and software have been seen, it may be interesting to know an approximate price required.

Hardware:

On the Internet, the requirements vary depending on the website visited, but to get a rough idea:

Minimum requirements:

SO: Windows 7 / 8 / 10

Processor: Intel Core i3

RAM: 4 GB

Graphic card: Nvidia 560 ti

Nowadays, any computer exceeds these requirements, any PC that fits the budget would be worth it, so an exact price will not be specified.

Software:

- Blender, Mixamo, Github y Image Magick are for free.
- Unity3D offers free licenses for freelancers and videogame development companies that bill less than $ 100,000 in a year.
- TexturePackerGUI costs 39.99 and SpriteIlluminator also costs 39.99, although there is an offer to get both for 59.99
- Photoshop has 3 types of plans:
  - Annual plan, monthly payment - € 24.19 / month
  - Annual plan, prepaid - € 290.17 / year
  - Monthly plan - € 36.29 / month

3. Tools

These tools and processes have been used in all the demos (except for Demo4), instead of seeing this procedure in each demo, a summary will be made here.
Before using "Unity", It is necessary to create the 2D transformation. Different tools have been used. The order described in the memory is the same order in which these tools were used in the project.

3.1 - Blender

To put the modeling in "Blender", it is first necessary to get them, for this we will use various pages:

https://clara.io/
https://free3d.com/
https://sketchfab.com/feed
https://www.turbosquid.com/
https://www.cgtrader.com/free-3d-models

3.1.1 - Modeling preparation

Once the modeling is obtained, it is necessary the following three modifications need to be made before rigging.

Add Texture

Generally all the models (free or paid) are contained in a "WinRAR", separated between "modeling" and "texture". Therefore, once obtained, it is necessary to add this texture, because although it is given with the model, it is located in a different folder so that there are no problems or errors when loading the model in the desired 3D program (Blender, 3Ds Max, etc). These modifications are necessary, since it is necessary to "link it" and then, when loading it into "Mixamo" the reference will not be lost.

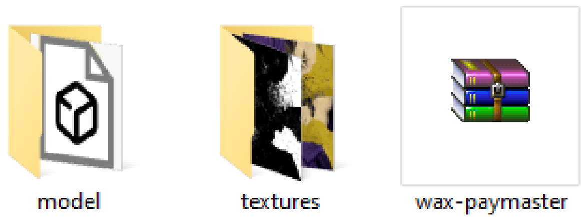


Figure 5: Folders containing the textures and 3D modeling

1- Folder containing only modeling (Figures 5 and 6)

Figure 6: Modeling ".fbx" file

2- Folder containing the texture of the modeling, normals map and emission maps (Figure 7)

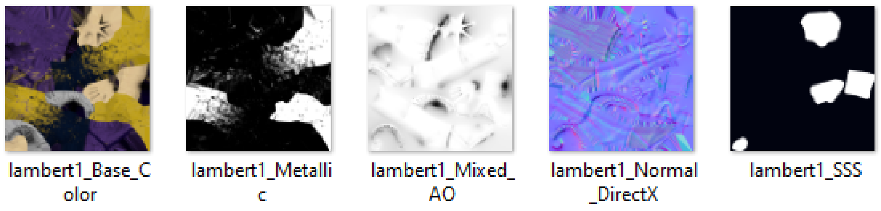


Figure 7: The different “texture maps” of 3D modeling

Although these models contain all the different necessary maps to achieve a correct 3D visualization, due to the nature of this project, only the texture is necessary (Figure 9).
This generates the first problem. There are many models without texture (Figure 8), this should be taken into account when downloading models, there may be some that cannot be used because of this, which eliminates many models that can be useful, but the textures of the maps must be generated separately.

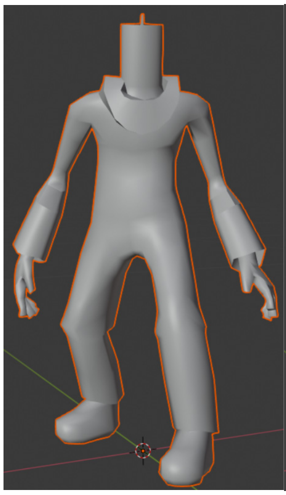


Figure 8: Blender modeling without texture

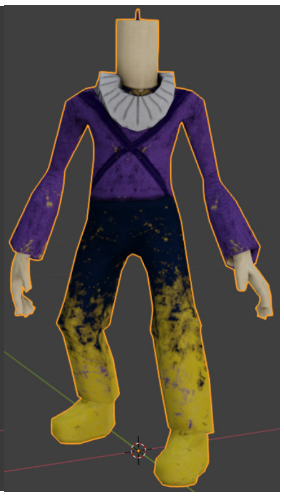


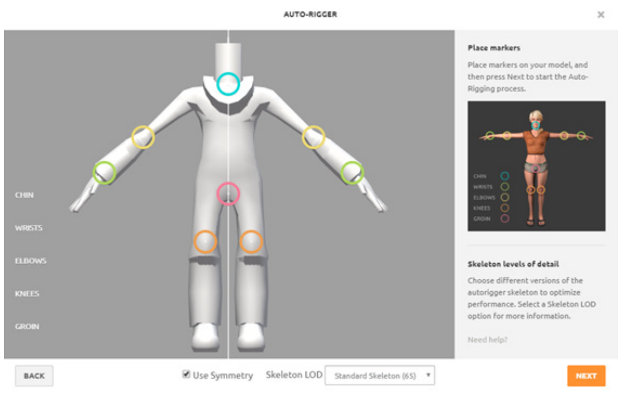
Figure 9: Blender modeling with texture

These Models must be saved as FBX, so all the information is saved avoiding future problems

Rig in Mixamo

www.mixamo.com

“Mixamo” is a public domain page used to obtain animations for modeling (Figure 10). It is also possible to obtain some of the models shown as an example on this page, to see the result of the animations. In addition, it is possible to rig characters on this page, allowing to add any animation on the page. This is the main feature that interests us.



Place markers

Place markers on your model, and then press Next to start the Auto-Rigging process.

Skeleton levels of detail

Choose different versions of the autorigger skeleton to optimize performance. Select a Skeleton LOD option for more information.

Figure 10: Rigging in Mixamo

Once rigged, It can be observed in real time the chosen animation merged with the modeling in the form of "Preview" before downloading it (Figure 11), to use it in "Blender". the download will be done the first time with "with skin" so that the rigged modeling is downloaded and the following times "without skin" since having a rigged one, it is enough to substitute later in "Blender" the animation of your "armature".

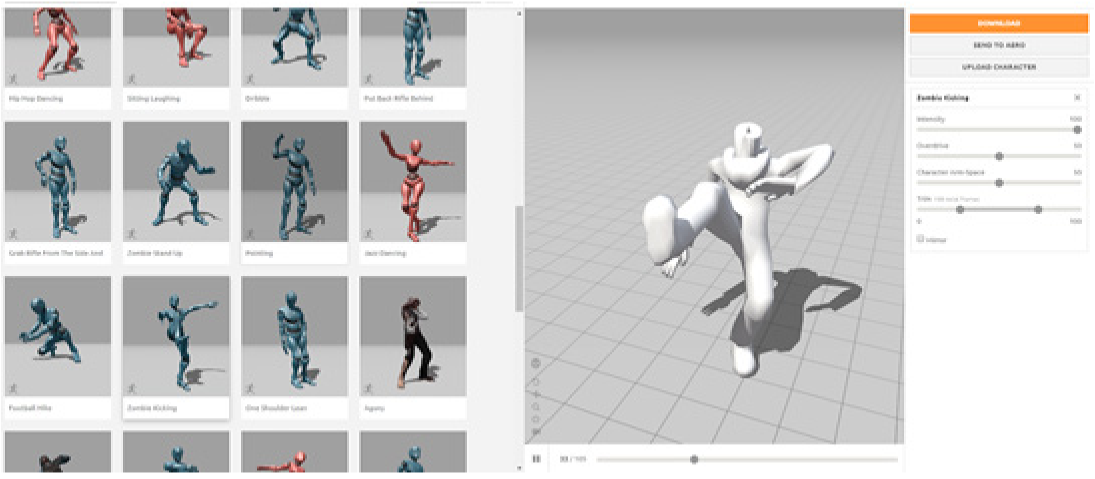


Figure 11: Add animation in Mixamo

The problem produced by this method: when downloading it again from "Mixamo", if the format is ".OBJ" it only saves the geometry, but references to texture and the geometry that were saved in parts are lost. So it is necessary to do it in ".FBX" to save these references.

Re-add texture

Sometimes "Blender" can generate errors when adding the texture at this point, so it is necessary to do step 2.1.1 again, this way it does not lose the references and it is easy to add the textures now (Figure 12).
The same process is done, from the "Blender Shading".

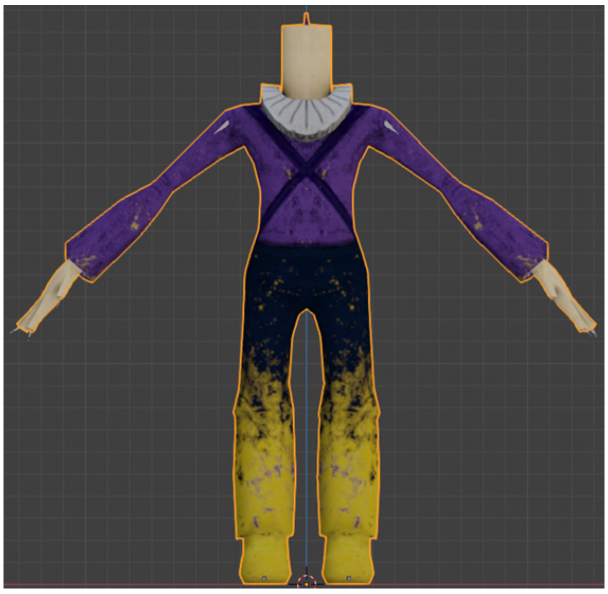


Figure 12: Re-add 3D modeling in Blender with animations

3.1.2 - Add Animations

Once the character has been rigged, the animation must be added (Figure 13), animation that is achieved from the Mixamo itself (Figure 11). It must be in ".FBX" (to avoid loss of information) and "without skin" (to obtain only the animation)

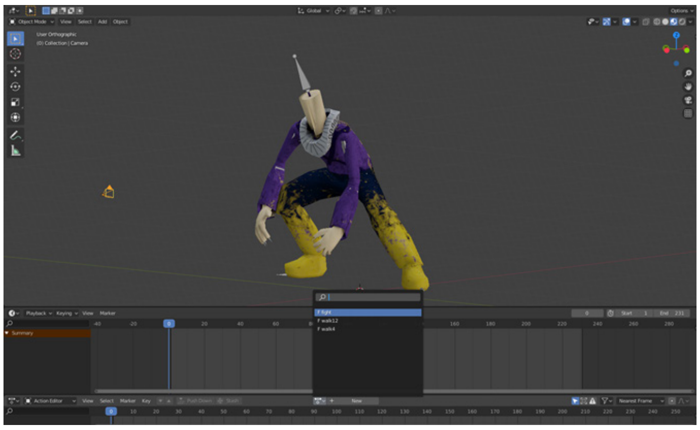
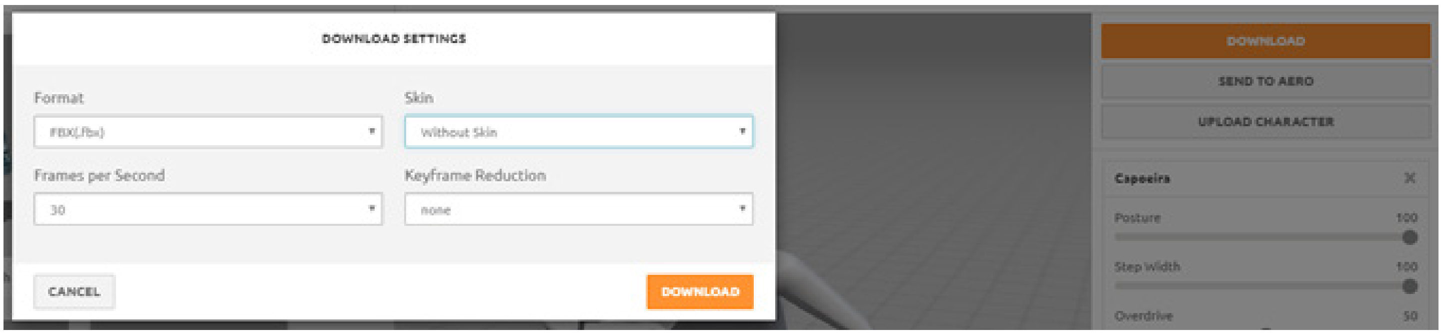


Figure 13: Select animation in Blender.


DOWNLOAD SETTINGS

Format: FBX(.fbx)
Skin: Without Skin
Frames per Second: 30
Keyframe Reduction: none

Capoeira
Posture: 100
Step Width: 100
Overdrive: 50

Figure 14: Download the animation in Mixamo

Depending on the animation, the camera needs to be fixed or follow the character.
Generally the camera looks at a fixed site but there are animations with which this does not work (Figure 15).

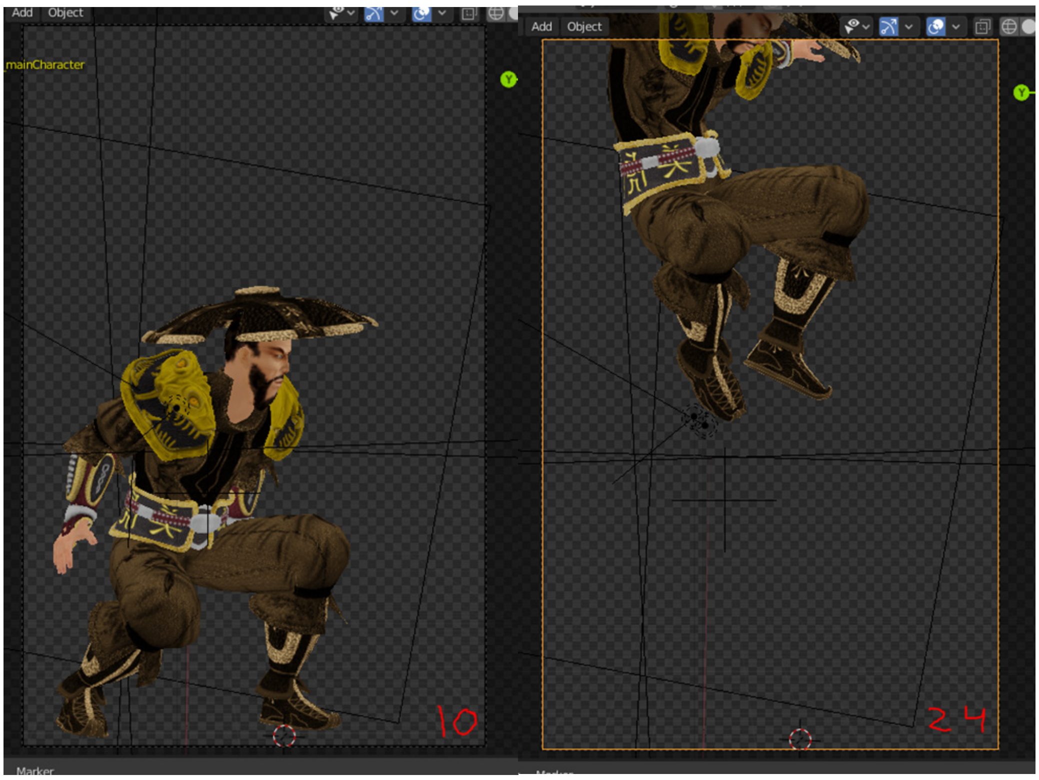


Figure 15: Camera that does not follow the character's jump

3.1.3 - 2D Sensation

Once the animation has been added to the 3D modeling (Figure 16), it is time to extract the spritesheets. To do it the first thing is to place a camera perpendicular to the modeling, this way when doing the animation it gives the 2D sensation

Next it is necessary to add the pixelation, there are 3 ways:

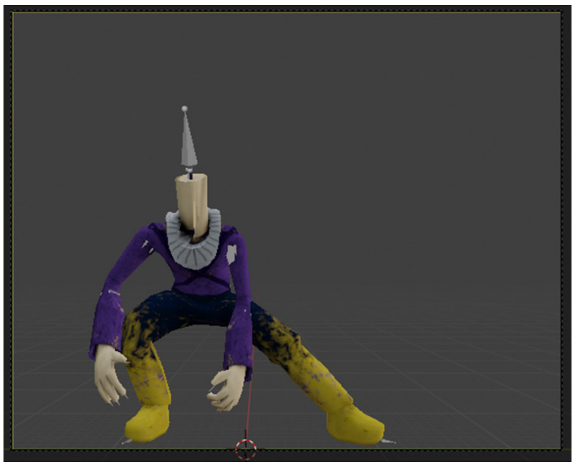
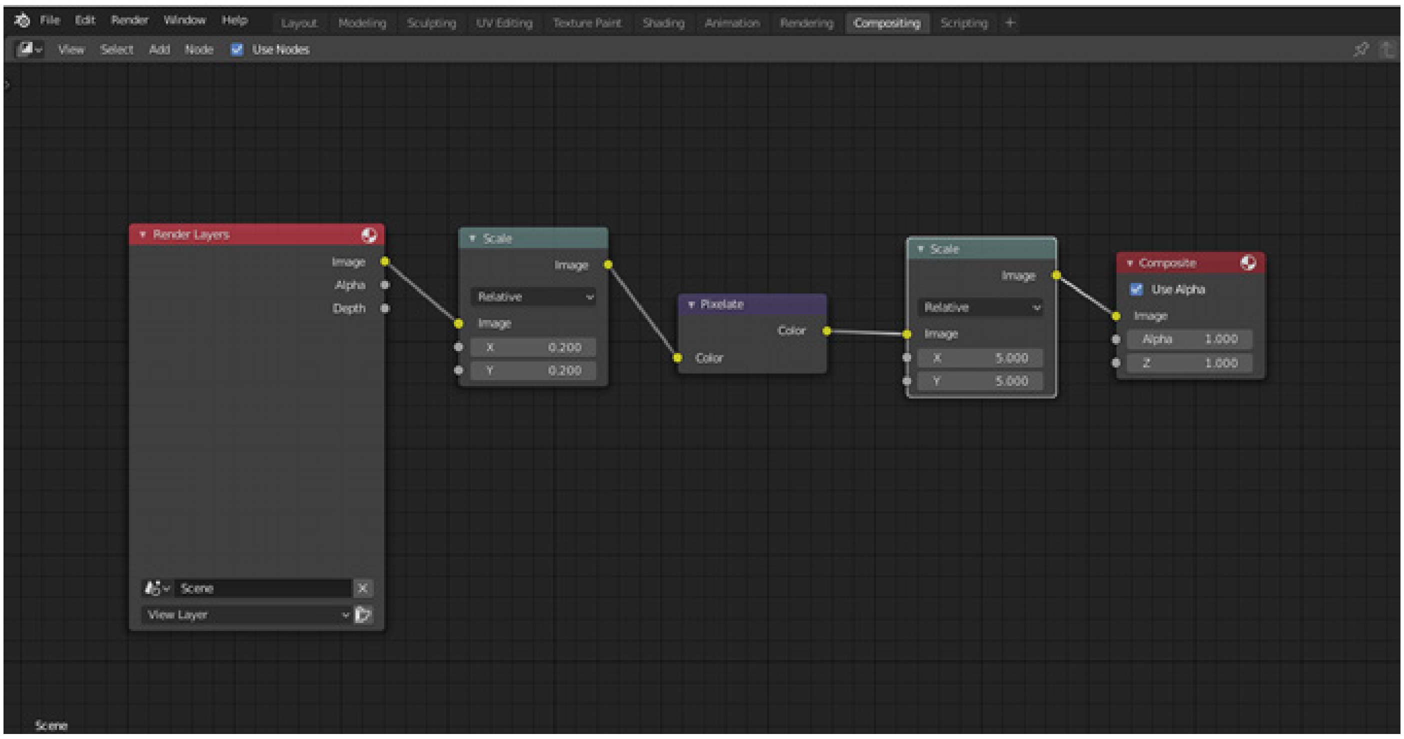


Figure 16: Example of character animation.

Compositing

"Compositing" is a tool that allows globally control the visual part of the "Blender" scene, adding nodes that affect rendering (Figure 17).

For pixelation, two scale nodes and one pixelate node are needed:


File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
View Select Add Node Use Nodes
Render Layers
Image
Alpha
Depth
Scene
View Layer
Scale
Image
Pixelate
Color
Color
Scale
Image
Composite
Use Alpha
Image
Scene

Scale: Relative, Image, X 0.200, Y 0.200

Scale: Relative, Image, X 5.000, Y 5.000

Composite: Use Alpha, Alpha 1.000, Z 1.000

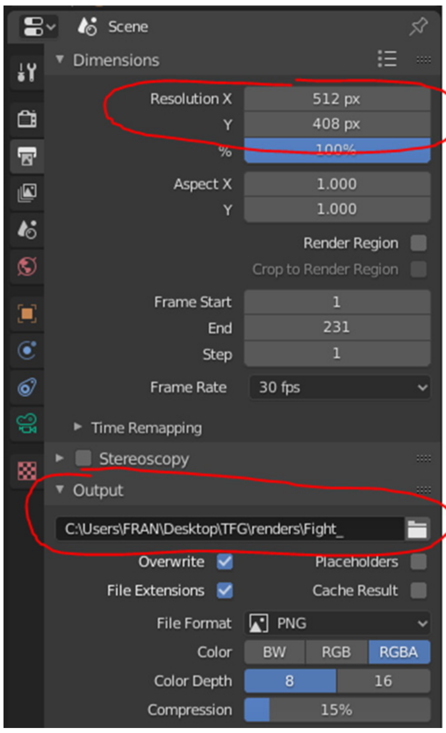
Figure 17: Example of "composition", performs the main pixelation

- **First Scale:** the global scene is scaled to a low resolution, achieving the visual pixelation
- **Pixelate:** if then it is simply rescaled to the original resolution the pixelation is removed, then it is necessary to save the information in this node
- **Second Scale:** in this node is rescaled to the size it was originally but pixelated, because the pixelated information was saved in the previous one

Output properties

In the “output properties” (Figure 18) the resolution is lowered in height and width so that it can be used as a sprite and that it also serves for pixelating.

It also can specify in the “output” the destination of the containing folder.



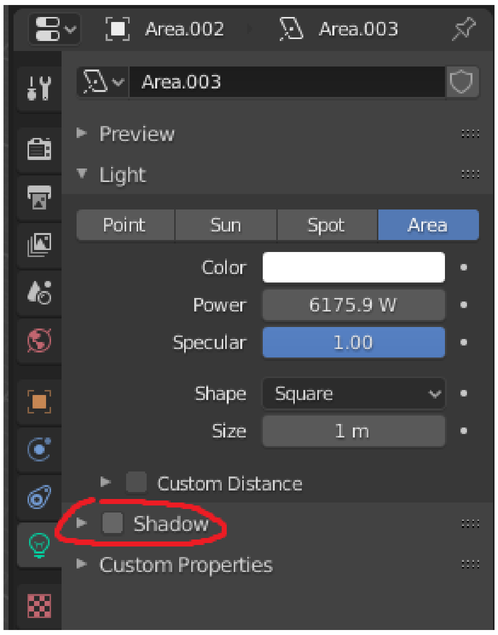
Scene
Dimensions
Resolution X 512 px
Y 408 px
% 100%
Aspect X 1.000
Y 1.000
Render Region
Crop to Render Region
Frame Start 1
End 231
Step 1
Frame Rate 30 fps
Time Remapping
Stereoscopy
Output
C:\Users\FRAN\Desktop\TFG\renders\Fight_
Overwrite ✓ Placeholders
File Extensions ✓ Cache Result
File Format PNG
Color BW RGB [✓ RGBA]
Color Depth [✓ 8] 16
Compression 15%

Figure 18: “Resolution” and “Output” properties of the Output Properties

Object Data Properties (Shadows)

The “Object Data Properties” (Figure 19) appear when a light is selected in “Blender”. There is a “shadow” check box which is enabled by default and should be disabled.

This is because although the lights are placed in all directions to illuminate the character, shadows are always generated (even if there are few) and we are not interested in these shadows when generating the “normal map” and removing this check box, these shadows are not generate (Figure 20).



Area.002 Area.003
Area.003
Preview
Light
Point Sun Spot [✓ Area]
Color
Power 6175.9 W
Specular 1.00
Shape Square
Size 1 m
Custom Distance
Shadow
Custom Properties

Figure 19: “Shadow” property of the Object Data Properties

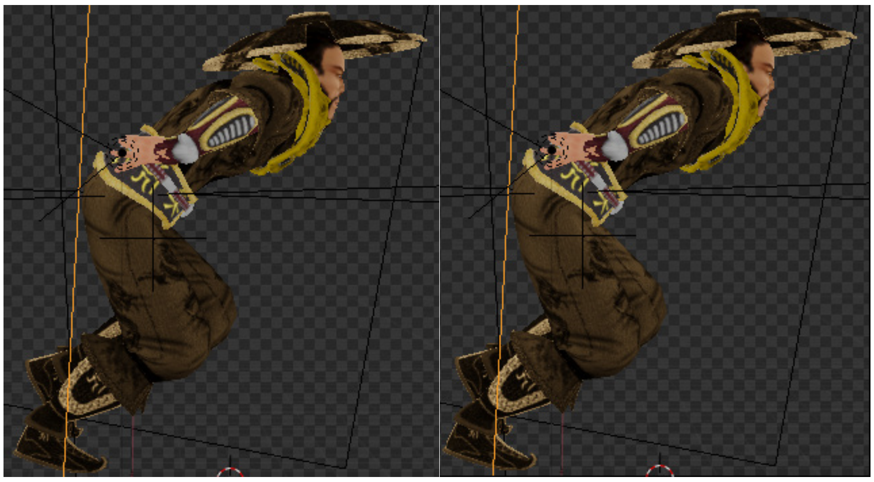


Figure 20: Shadow comparison by activating the “Shadow” option

Render properties

It is possible to change the "filter size" (Figure 21). The result of the render will be more accurate the lower the number and the higher the more blurred it is. Depending on the quality of the modeling, some values or others will be necessary.

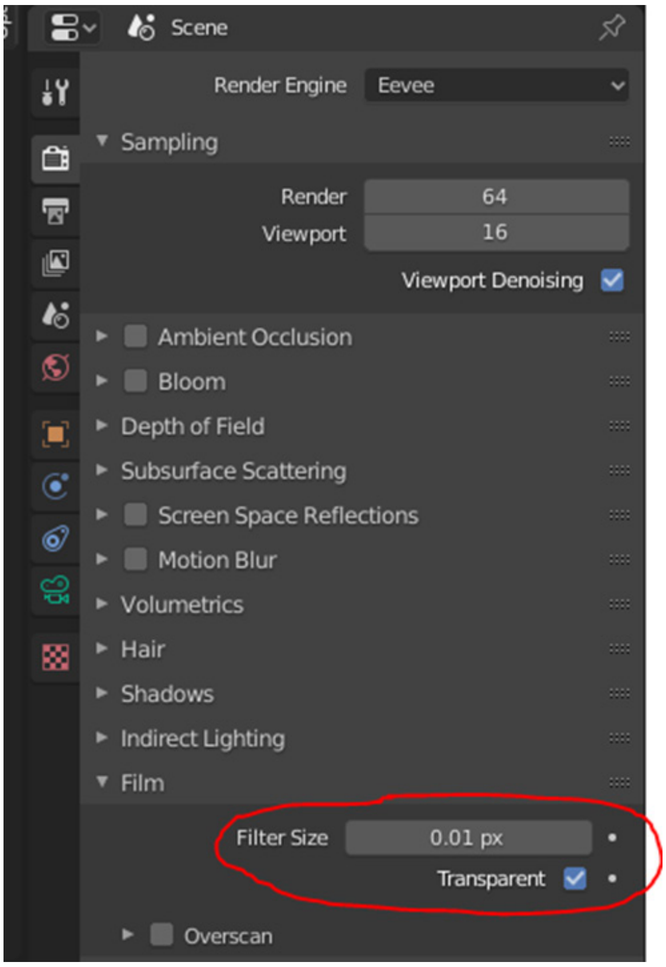

Scene
Render Engine: Eevee
Sampling
Render: 64
Viewport: 16
Viewport Denoising ✓
Ambient Occlusion
Bloom
Depth of Field
Subsurface Scattering
Screen Space Reflections
Motion Blur
Volumetrics
Hair
Shadows
Indirect Lighting
Film
Filter Size: 0.01 px
Transparent ✓
Overscan

Figure 21: "filter size" property of the Render Properties

The "Transparent" check box must also be activated so that in the rendering it removes the background, creating the PNG (Figures 22 and 23).

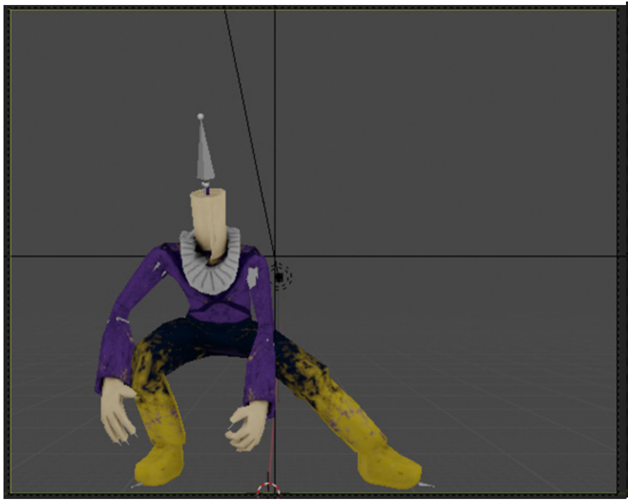


Figure 22: 3D modeling character before applying Blender properties

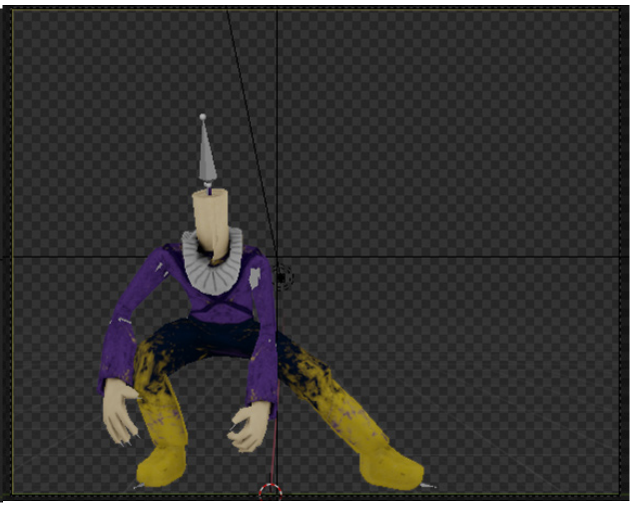


Figure 23: 3D modeling character after applying Blender properties

Each of these three pixelated shapes gives a different result, therefore, combining them gives a very natural result (Figures 18, 19, 21 and 23)

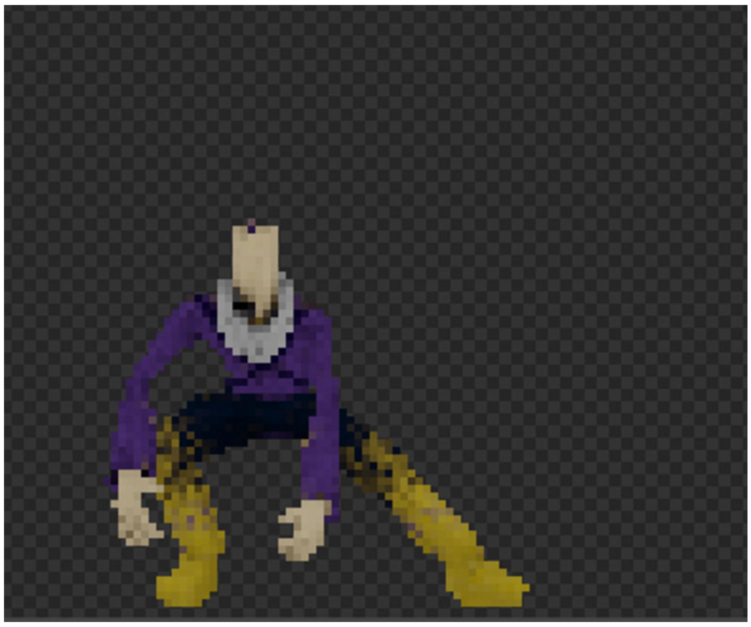


Figure 24: 3D modeling character after applying Blender properties. Version 2

Bordered (Shaded)

It is possible to create a border on the character (Figures 25 and 26). This technique was discovered in the pixelation documentation but in this project it has not been used, although it has been considered important to document it.

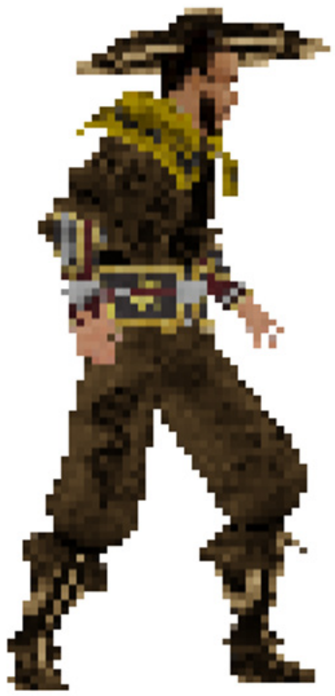


Figure 25: Character with shading

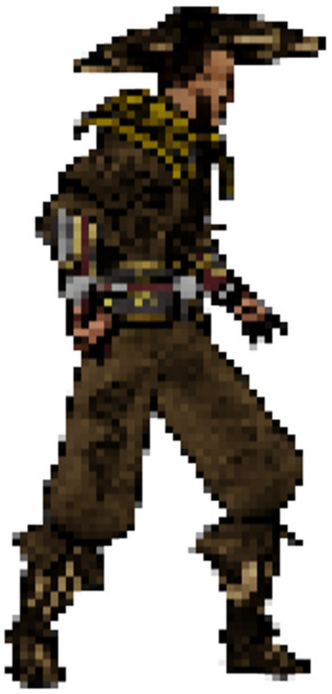


Figure 26: Character without shading

To do it, just activate the "freestyle" (Figure 27)

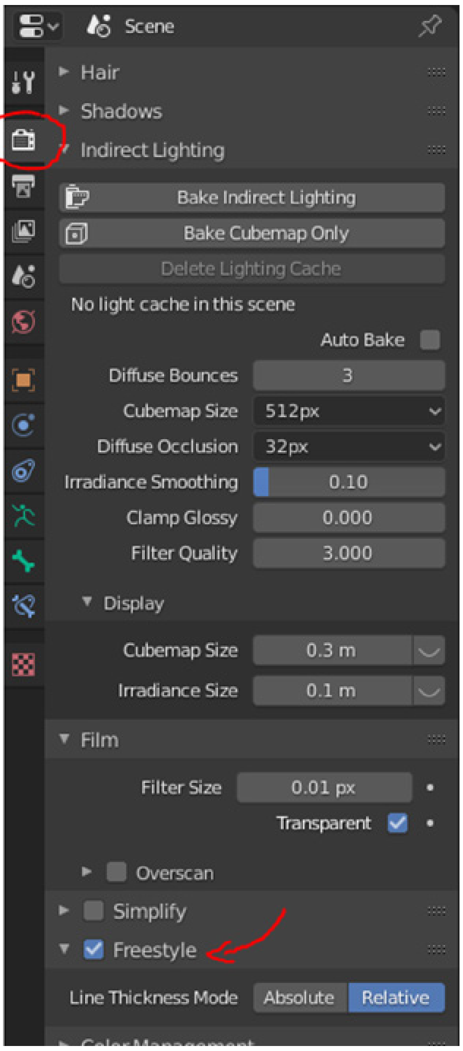

Scene
Hair
Shadows
Indirect Lighting
Bake Indirect Lighting
Bake Cubemap Only
Delete Lighting Cache
No light cache in this scene
Auto Bake ☐
Diffuse Bounces 3
Cubemap Size 512px
Diffuse Occlusion 32px
Irradiance Smoothing 0.10
Clamp Glossy 0.000
Filter Quality 3.000
Display
Cubemap Size 0.3 m
Irradiance Size 0.1 m
Film
Filter Size 0.01 px
Transparent ✓
☐ Overscan
☐ Simplify
✓ Freestyle
Line Thickness Mode Absolute [✓ Relative]
Color Management

Figure 27: "Freestyle" property of the Render Properties

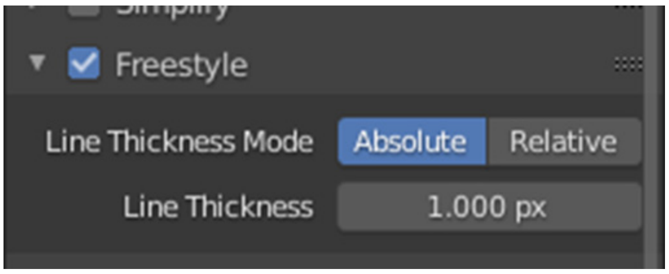

Simplify
✓ Freestyle
Line Thickness Mode [✓ Absolute] Relative
Line Thickness 1.000 px

Figure 28: "Freestyle" property of the Render Properties

In "relative" it puts a border automatically, but in "absolute" it can be adjusted (Figure 28), 1px is equivalent to the "relative" and then it is possible to adjust it by putting more or less.

It is interesting to know how this shading can be created since there are world famous games that use this technique, such as "Megaman" (Figure 29).

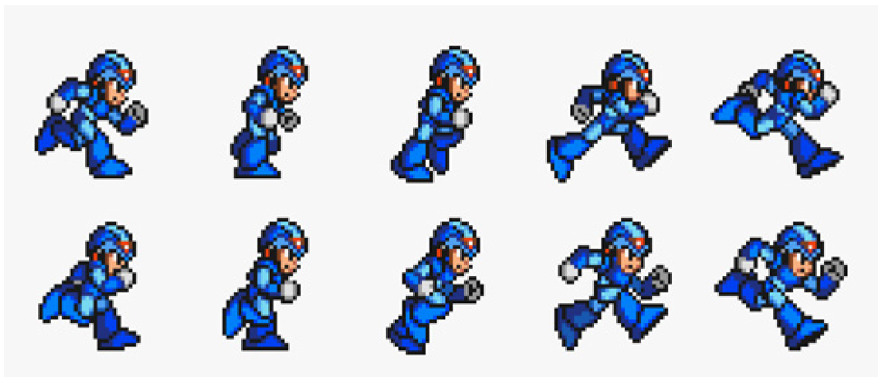


Figure 29: SpriteSheet with shading (Megaman)

But it could also not be applied. There are other games that do not use it (Figure 30), like all video game techniques depending on the artistic style, one or the other may be more useful.

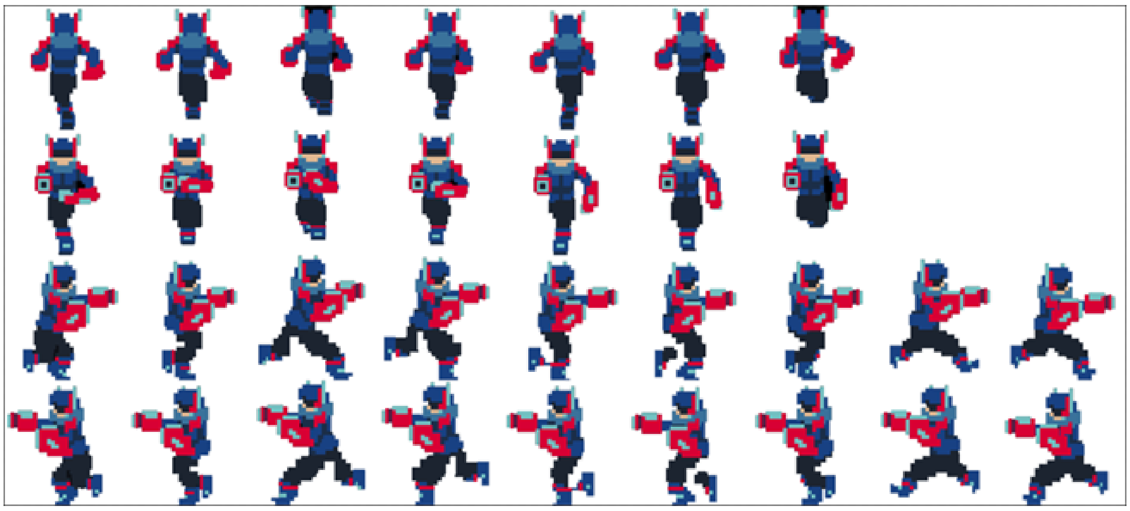


Figure 30: SpriteSheet without shading

3.1.4 - Illumination

Because "normal maps" will be generated later for real-time lighting, it is now necessary to illuminate it from all angles (up, down, right, left, front and back) to avoid unwanted shadows (Figure 31) and then normal maps are generated without problems (Figure 32).

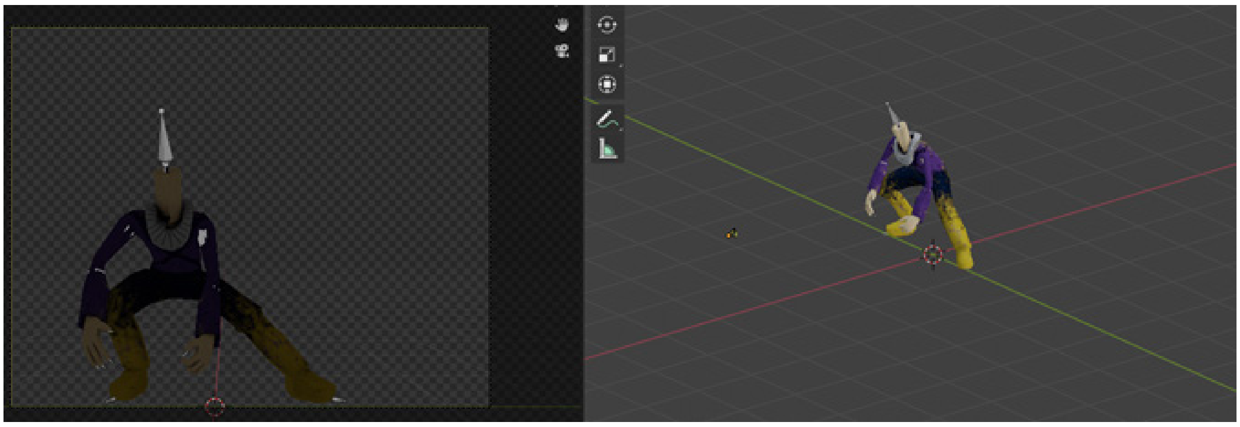


Figure 31: Blender scene without lighting

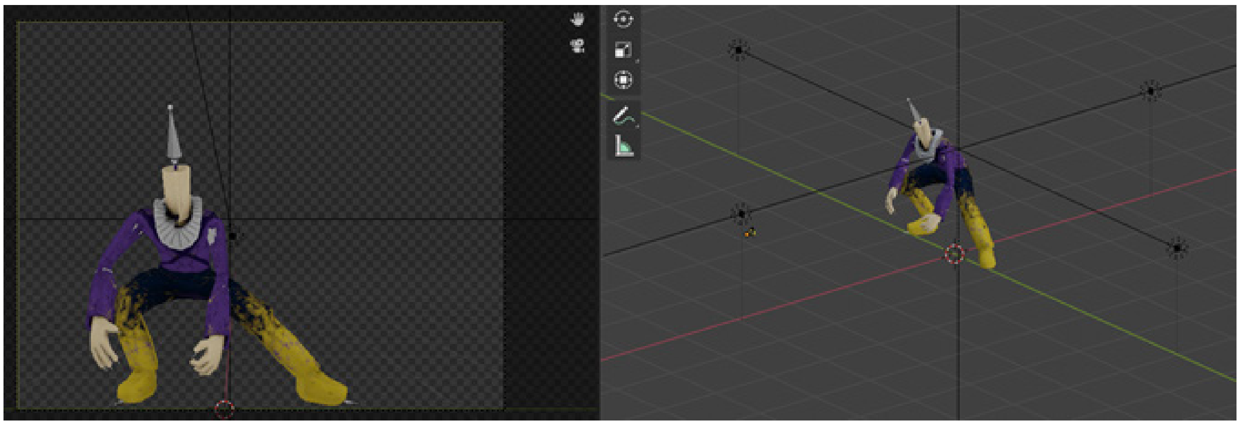


Figure 32: Blender scene with lighting

3.1.5 - Rendering

"Blender" provides an automated rendering, which allows to obtain all the "photos" of each movement (Figure 33). All these images need to be converted into spritesheets to be able to use them in the video game engine.

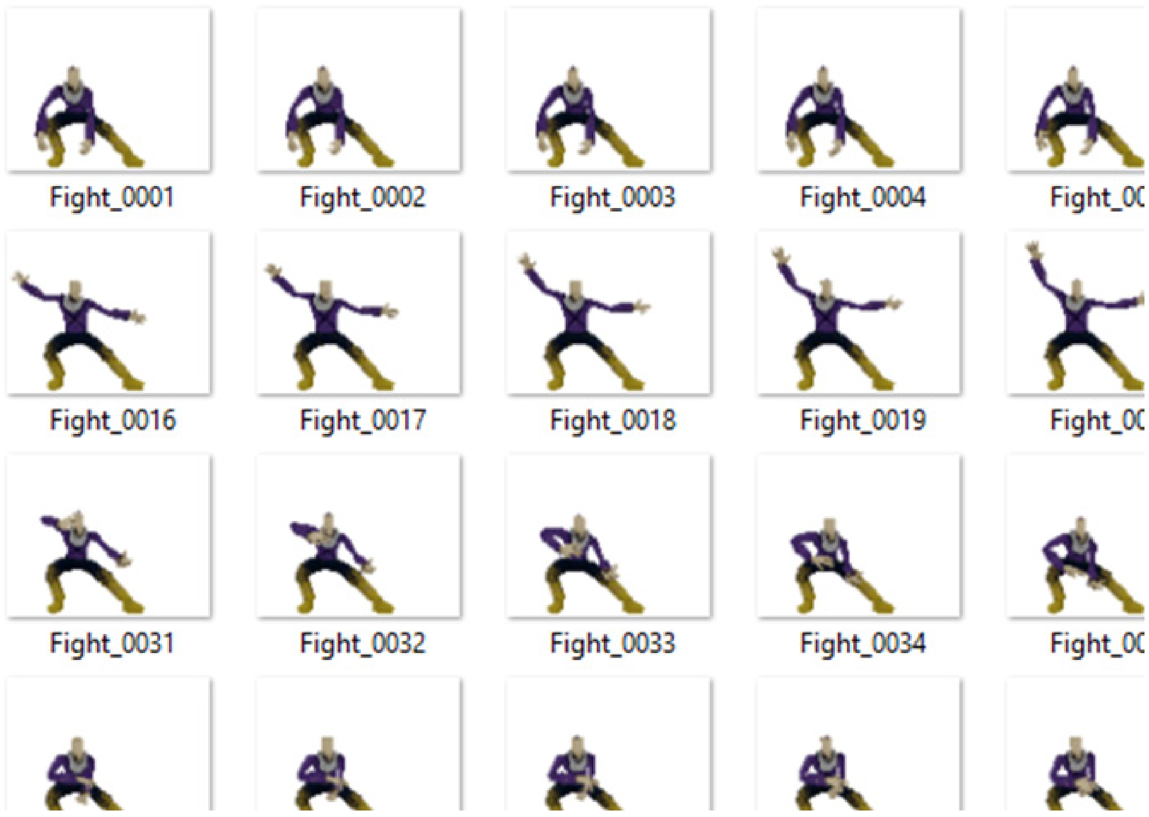


Figure 33: Images generated by Blender (the SpriteSheet has not been generated yet)

3.2 - Texture Packer GUI

This tool allows to create Spritesheets in a very automated way and adapted for “Unity” (video game engine used for this project). Therefore, it is the tool that has been decided to use.

Figure 34: TexturePackerGUI icon

Generate the SpriteSheet simply by “dragging” all the images into the program (Figure 35).

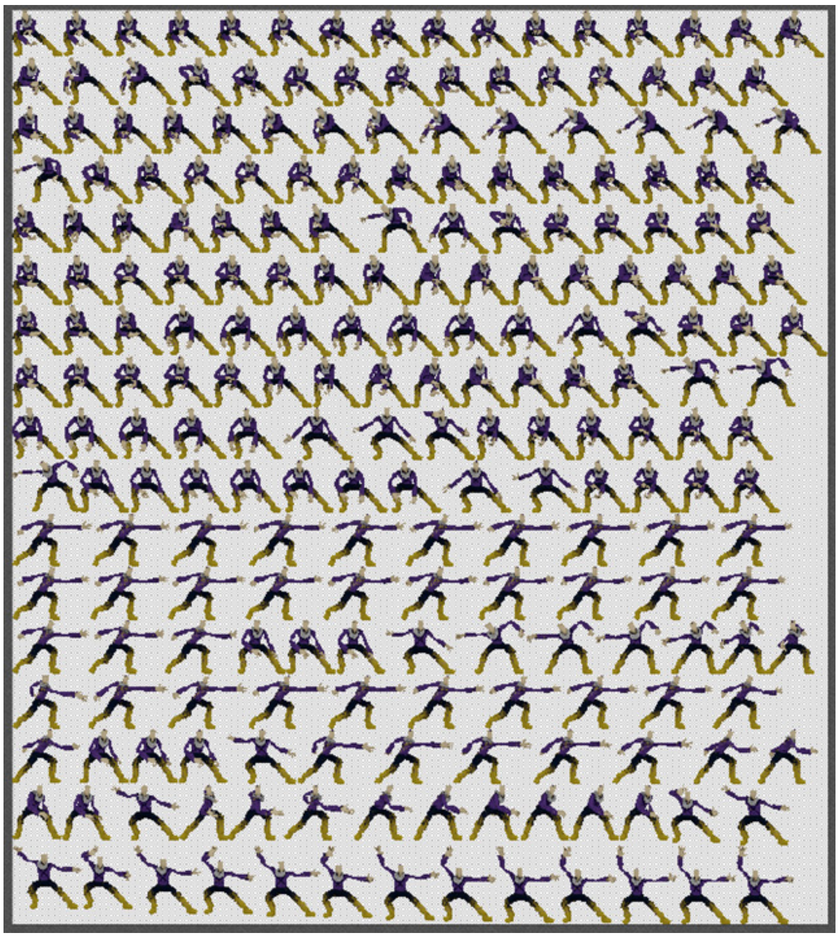
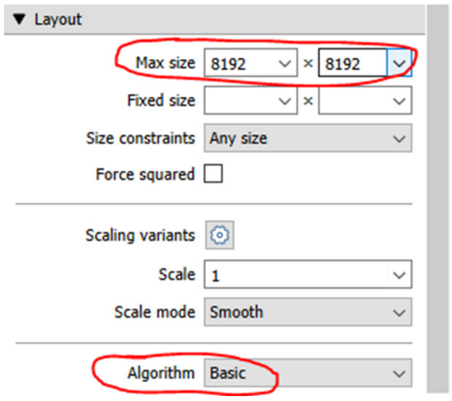


Figure 35: SpriteSheet generated in the TexturePackerGUI

Depending on the amount of images obtained, some parameters or others will have to be configu-red (Figure 36), which will then have to be reflected in “Unity”.



▼ Layout

Max size	8192 × 8192
Fixed size	×
Size constraints	Any size
Force squared	☐
Scaling variants	
Scale	1
Scale mode	Smooth
Algorithm	Basic

Figure 36: TexturePackerGUI Properties

3.3 - Useful frames

At first several intermediate frames were going to be eliminated to generate the feeling of "hand drawn SpriteSheets" (Figure 37).

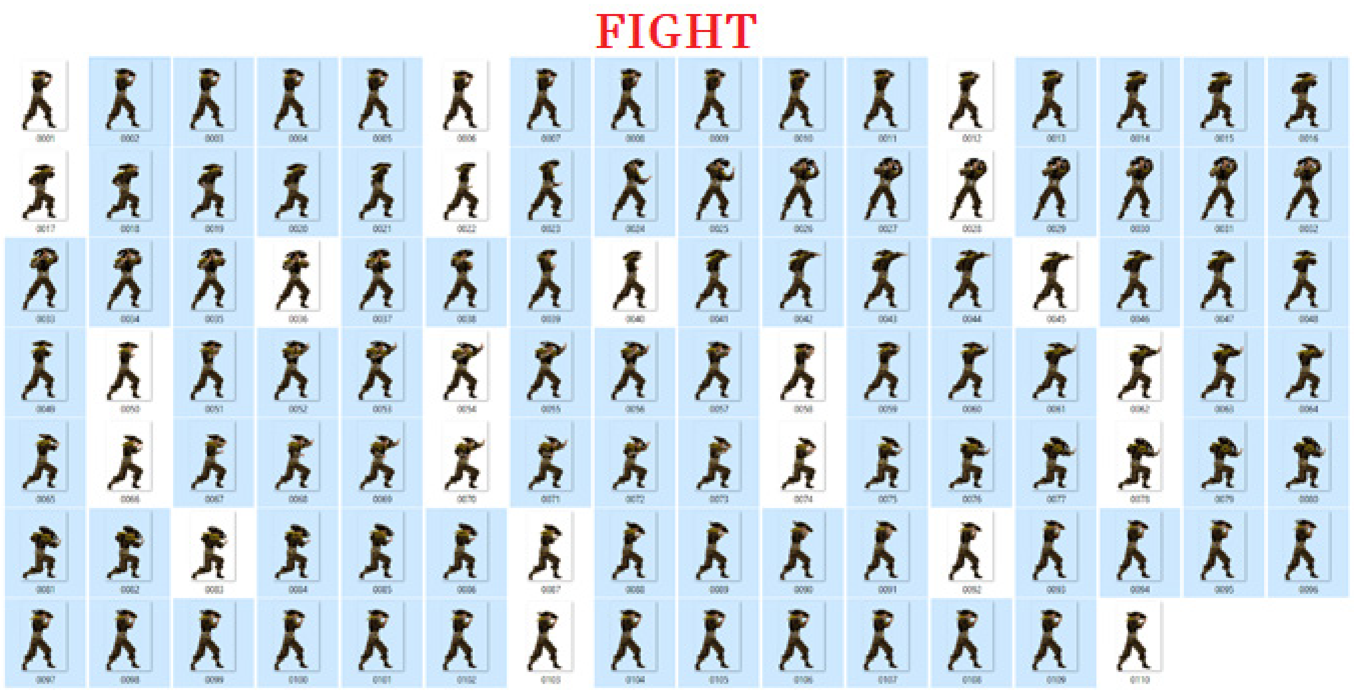


Figure 37: All images generated from an animation. The images selected at the beginning were not necessary for the final result.

All the selected images were to be deleted, leaving only the unselected ones, which would make up the animation, leaving a result like this:

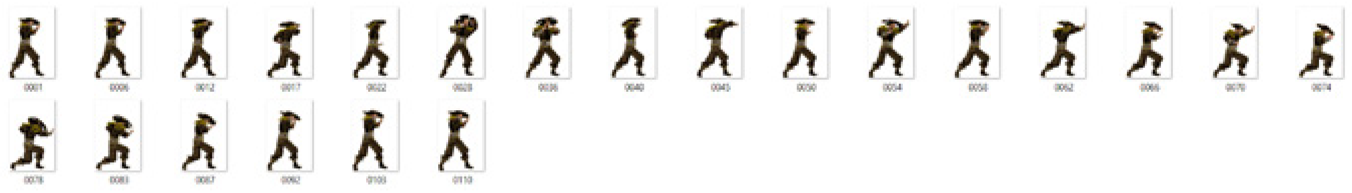


Figure 38: Sequence of images after deleting selected images

Visually approaches the traditional SpriteSheet style:

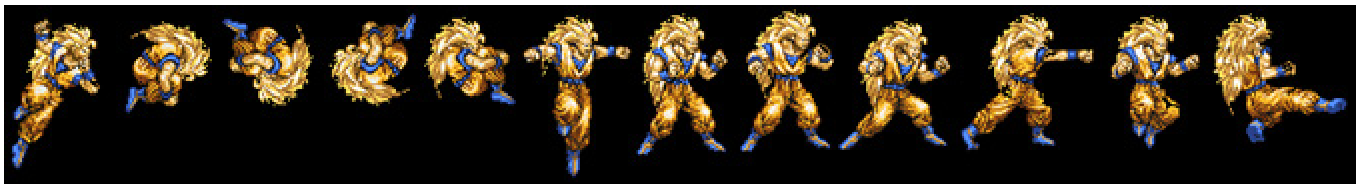
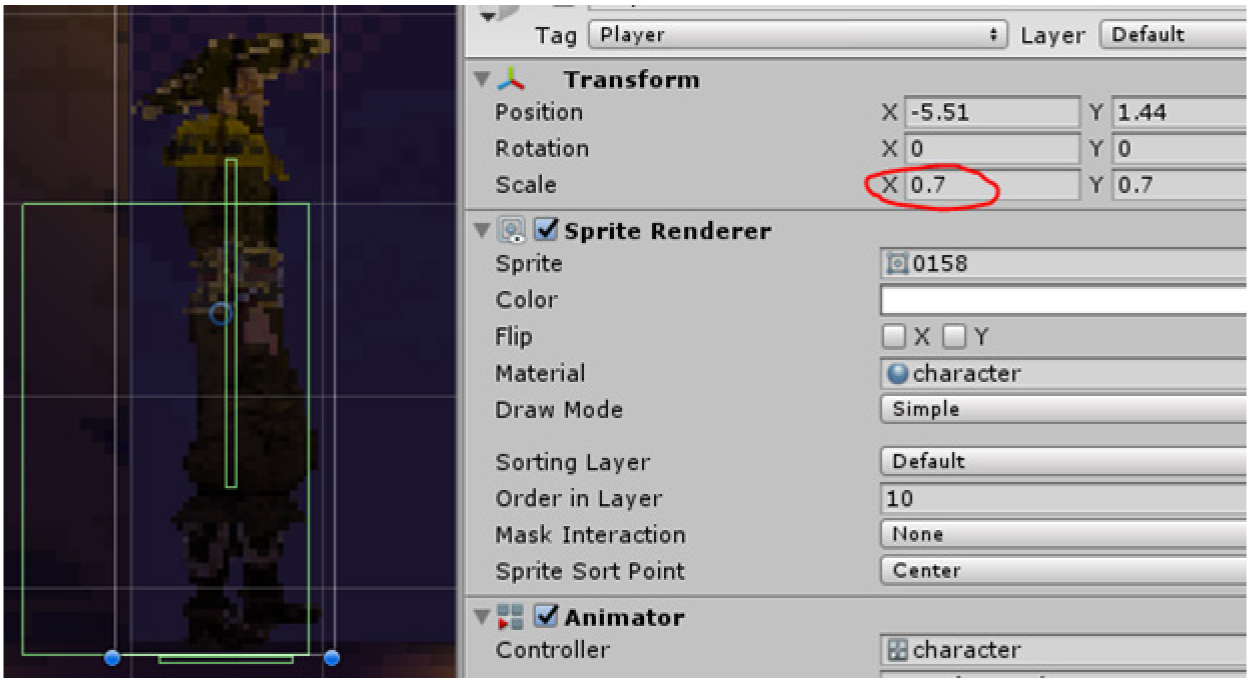


Figure 39: Example of a classic SpriteSheet

But after several tests the decision taken was to use all the frames, because that desired feeling of "hand drawn SpriteSheets" is achieved in the same way, and leaving all the frames gives a more natural animation for the human eye.

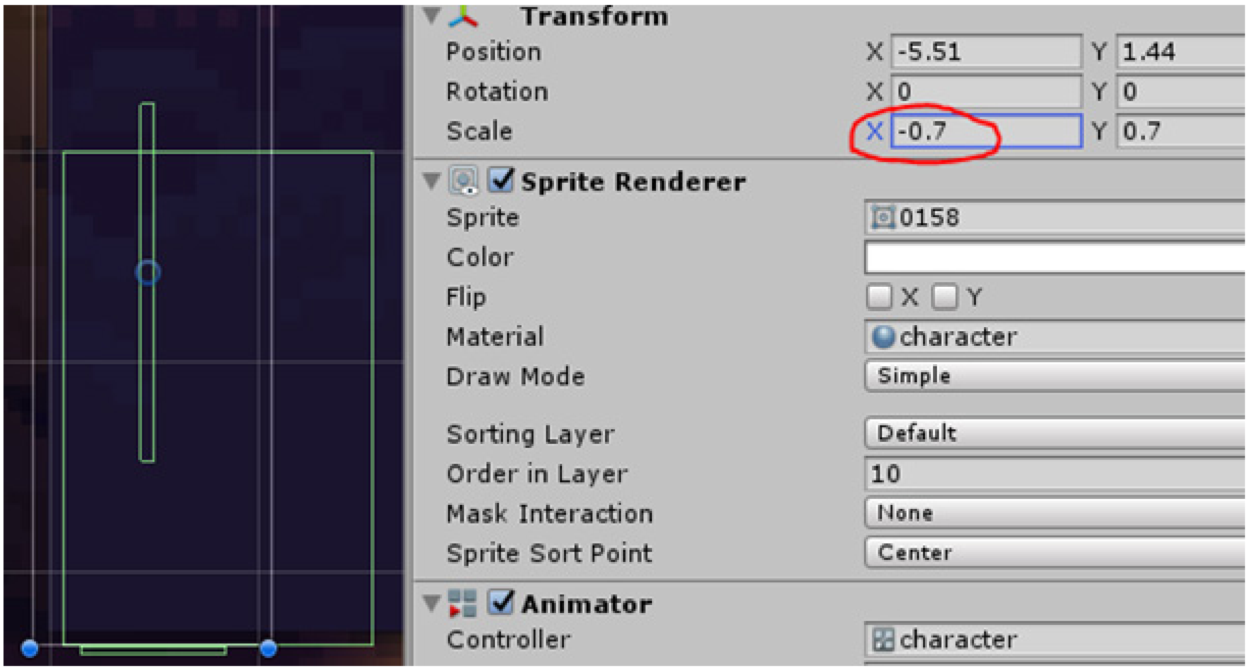
3.4 - Image Magick

Once the images have been obtained to create the sprite sheets, it is necessary to make a copy and flip them vertically to have the “right” and “left” direction because when putting negative values on the X scale in “Unity” to flip the character disappears . (Figure 41)



```
Tag  Player                      Layer  Default
Transform
Position          X -5.51   Y 1.44
Rotation          X 0       Y 0
Scale             X 0.7     Y 0.7
Sprite Renderer
Sprite            0158
Color
Flip              X  Y
Material          character
Draw Mode         Simple
Sorting Layer     Default
Order in Layer    10
Mask Interaction  None
Sprite Sort Point Center
Animator
Controller        character
```

Figure 40: Character in Unity3D on positive X axis



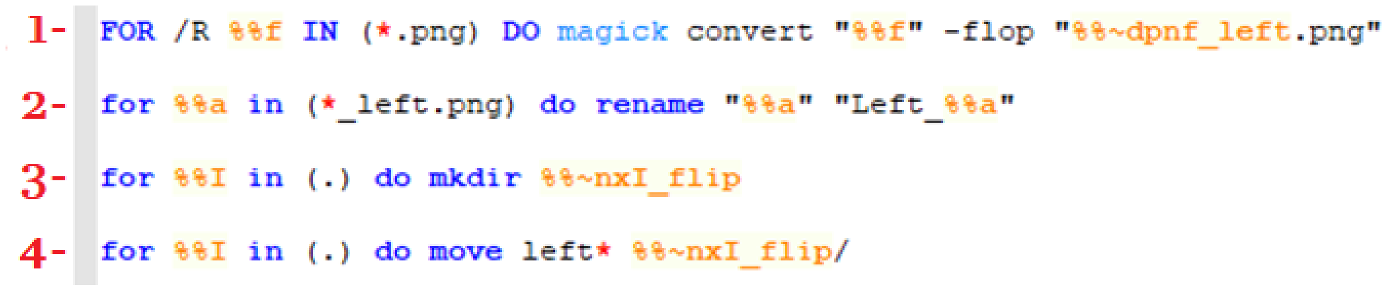
```
Transform
Position          X -5.51   Y 1.44
Rotation          X 0       Y 0
Scale             X -0.7    Y 0.7
Sprite Renderer
Sprite            0158
Color
Flip              X  Y
Material          character
Draw Mode         Simple
Sorting Layer     Default
Order in Layer    10
Mask Interaction  None
Sprite Sort Point Center
Animator
Controller        character
```

Figure 41: Character in Unity3D on negative X axis

It would be possible to easily do it in programs like Photoshop, but if this were done with each image generated, it will take too long. So this process must be automated. In Photoshop it cannot be done, so it is necessary to find another method.

The program found which is going to help us in this process is the "Image Magick" program which has several functions, including the option to flip the image vertically, this program also allows us to do it with the cmd with the command "flop". Then came the idea of creating an automated ".bat" program with this command.

The program has these four lines of code (Figure 42):



```
FOR /R %%f IN (*.png) DO magick convert "%%f" -flop "%%~dpnf_left.png"

for %%a in (*_left.png) do rename "%%a" "Left_%%a"

for %%I in (.) do mkdir %%~nxI_flip

for %%I in (.) do move left* %%~nxI_flip/
```

Figure 42: Lines of code from the ".bat" program created

For example, having a folder with the program (Figure 43), executing it for each image will do the following:

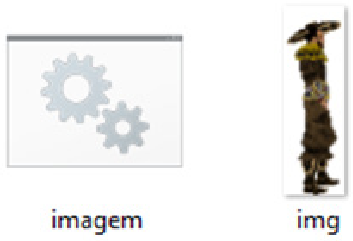


Figure 43: Before running the program

1- For each object in this folder which ends in ".png" it flips vertically and receives the same name as the original image, but ends in "_left.png"

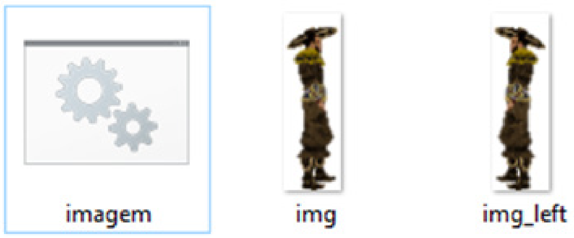


Figure 44: Create a copy of the (flipped) image

2- For each existing object in this folder ending in "_left.png" it is renamed by taking the name it had before and putting the "Left_" first

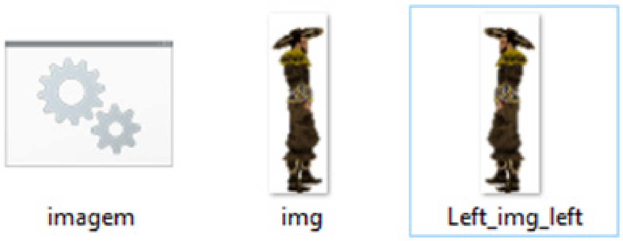


Figure 45: Rename the flipped image

3- Another folder is created in the current folder taking the name of the current folder, adding "_flip" at the end

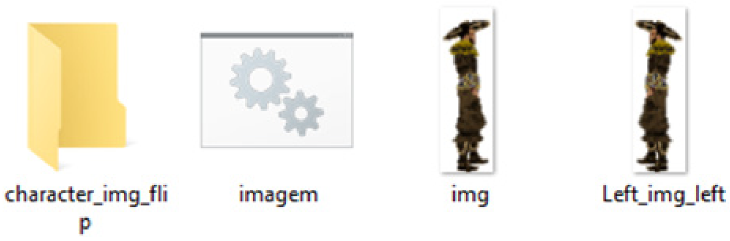


Figure 46: Create a folder to save all flipped images

4- All items starting with "left" are moved to the newly created folder.

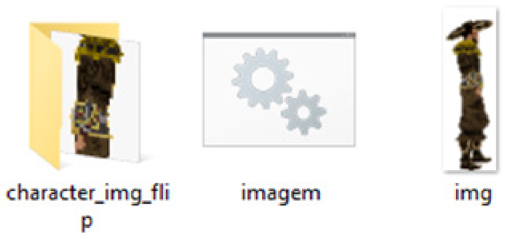


Figure 47: Move all images to created folder

4 - Work Development and Results

4.1 - Work Development

4.1.1 - Demo1

The first demo created was “Demo1”. In this Demo it was wanted to test the normal maps applied to the 2D spritesheet character and see how light can affected it in real time. To do this, a 3D character has been downloaded, converted to 2D, the sprite sheets and normal maps required have been generated.
A 2D scenario from the Asset Store has also been used to see if there is any visual problem when putting together a real 2D element and a 2D element created with these characteristics.

Main character

This is the main character used for this demo:

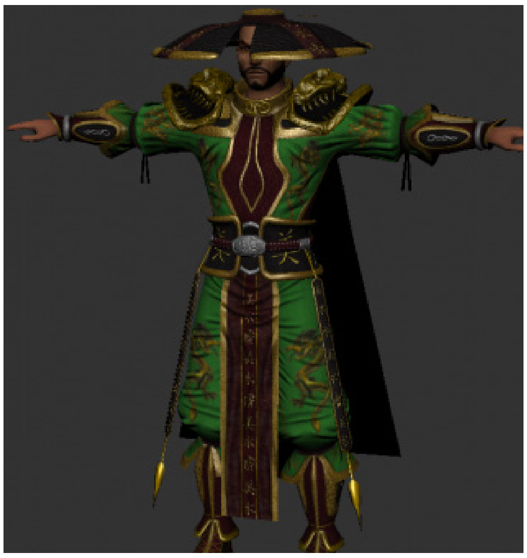


Figure 48: Demo1 main character without modifications.

The idea was to use this character in the downloaded “.blend” format, but when animations had to be created in “Blender”, the clothes that the character has as his front and back cape need the use of “physics”, which due to the current lack of knowledge about the physics applied to “Blender”, it was decided to remove this clothing and give the character another visual appearance to make it more pleasing to the eye.

After removing these clothes and several tests in the Blender “Shading” to modify their color, this was the result:

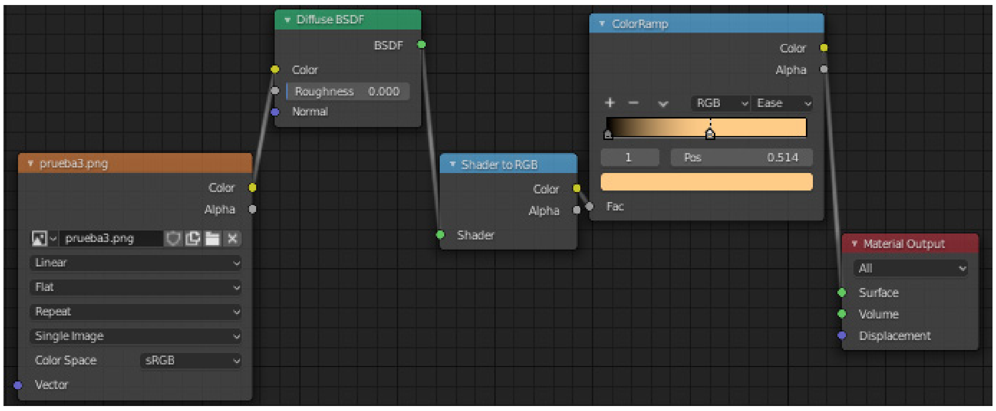


Figure 49: Character modifications in “Shading”

The result obtained has been:

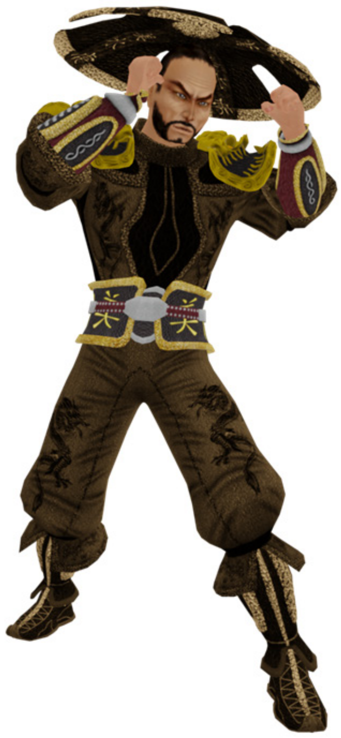


Figure 50: Main character after applying the visual changes.

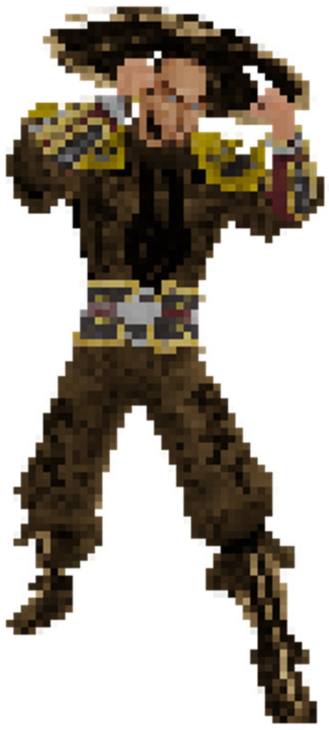


Figure 51: Main character after applying 2D transformation techniques

Figure 50: It is the character of Blender after applying the visual changes
Figure 51: It is the character when 2D transformation techniques have been applied.

SpriteSheets:

Now it was necessary to create the animations. These animations have been downloaded from "Mixamo" and applied to the character in "Blender". In "renders" the animation is generated. The problem with this method is that for each movement of the animation it creates a "photo" but not a "spriteSheet". To create it we need to use the "TexturePackerGUI" program that does it automatically.

Jump: Simple jumps are the best visually in 2D games, because making complex jump animation attractive takes a lot of work, so the choice has been the "classic solution", a fixed jump.

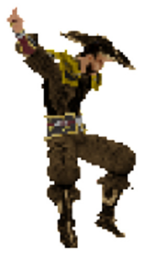


Figure 52: Jump of the main character

Idle: A long animation for this state was sought.

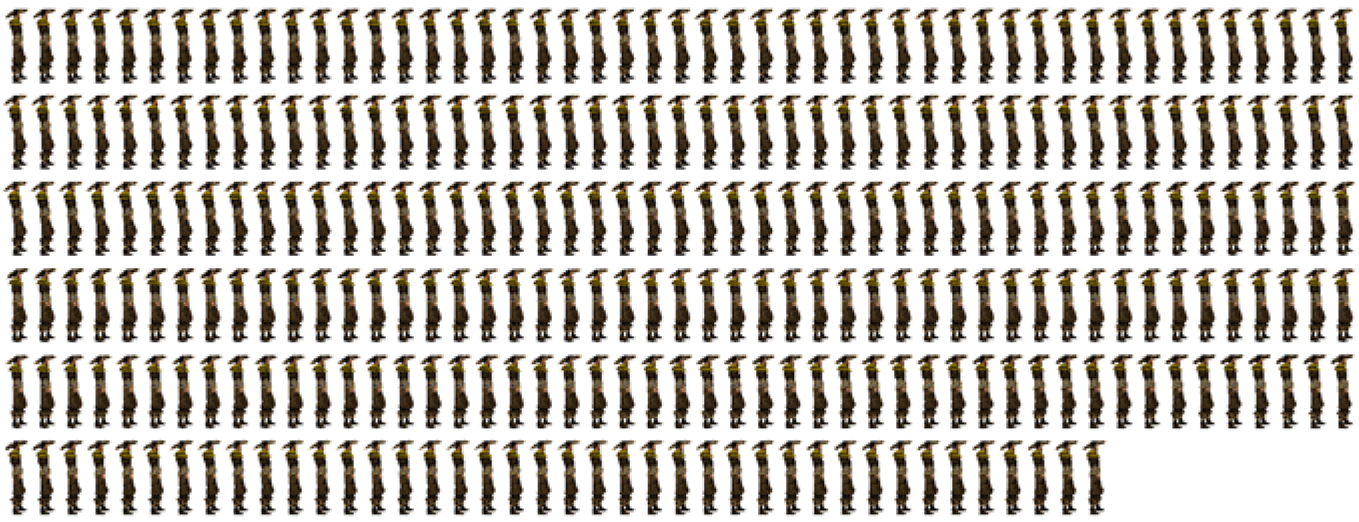


Figure 53: Idle of the main character

Fight: Although there are no enemies for this demo, an animation with a certain visual dynamism was sought

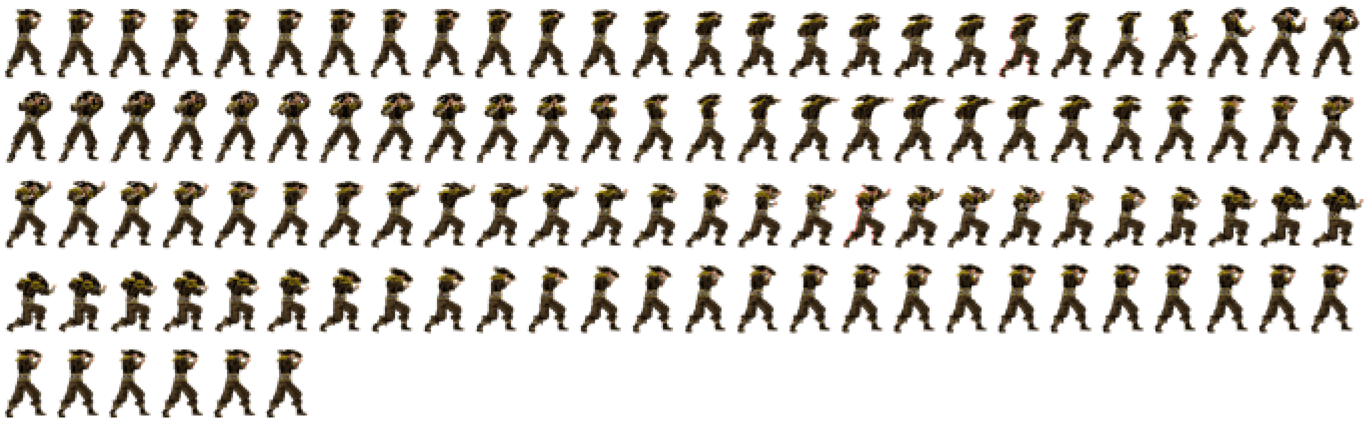


Figure 54: Fight of the main character

Run: Character movement

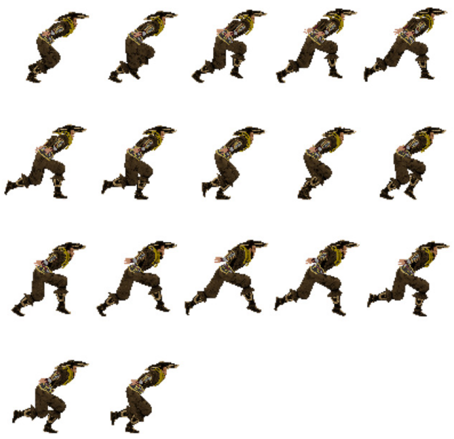


Figure 55: Run of the main character

Background

The chosen background for this Demo has been the "GothicVania Town" of the "AssetStore", a 2D background.

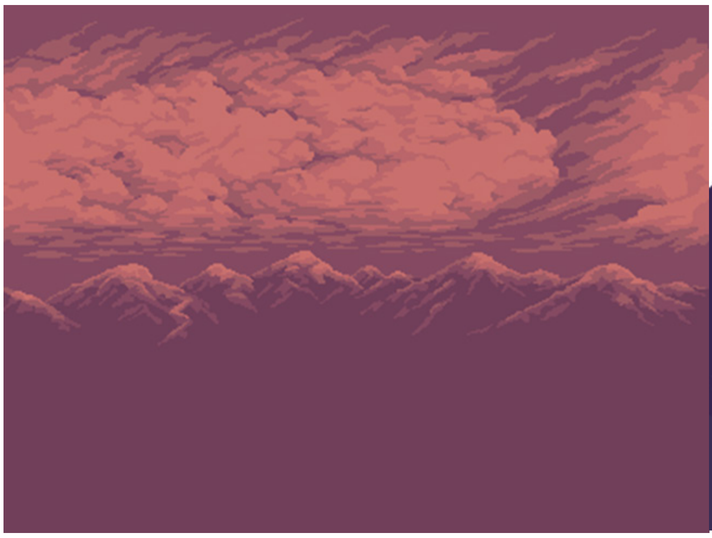


Figure 56: Background1 of Demo1

Figure 57: Background2 of Demo1

Tile Map:

Unity has a function called "Tile Map" (Figure 59), it allows creating 2D scenes in "Unity" with an elements palette.

The chosen Asset has elements to fill the scene, these elements have been used in the "Tile Map"

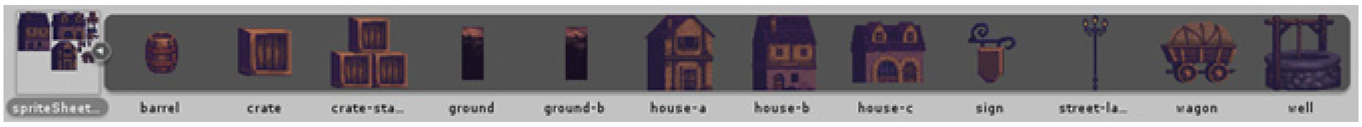


Figure 58: Image containing elements of the scene in Unity3D

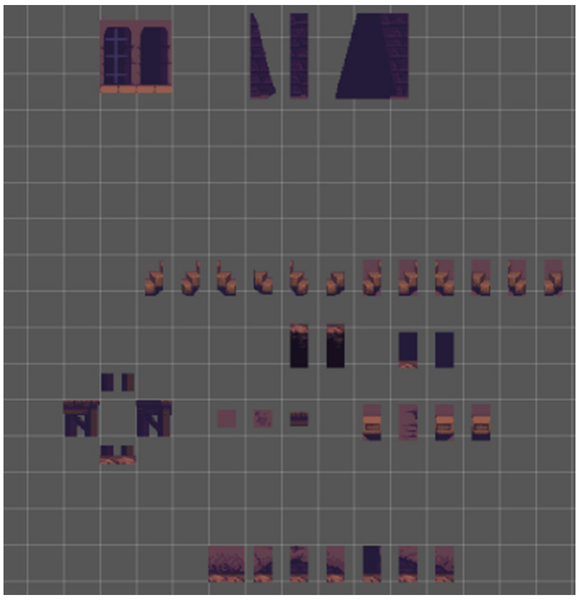


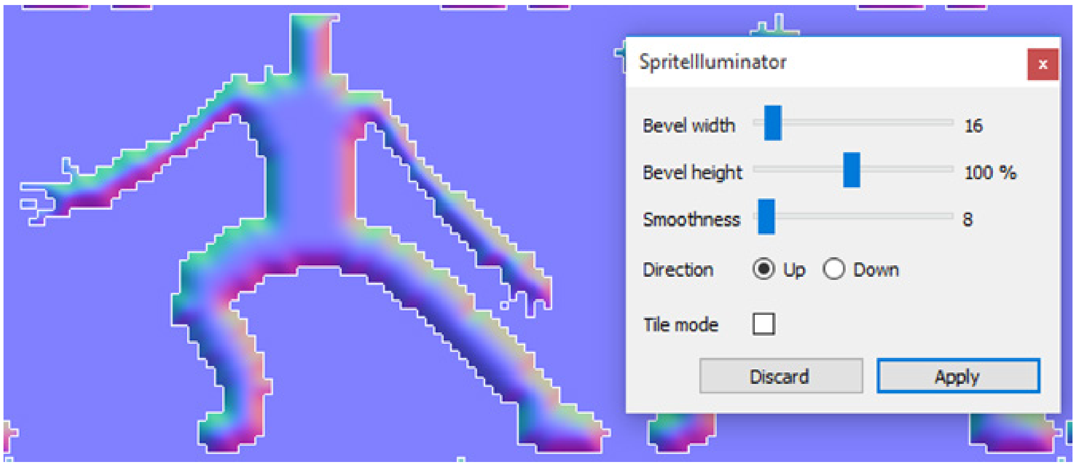
Figure 59: Scene elements placed in the TileMap

Normal Map

Normal maps need to be created for real time light to work in Unity, to do this, the "SpriteIlluminator" has been used

Figure 60: SpriteIlluminator Icon

The creators of "TexturePackerGUI" have more programs, among them the "SpriteIlluminator", which in a similar way to "TexturePackerGUI", generates normal maps in an automated way.


SpriteIlluminator

Bevel width 16
Bevel height 100 %
Smoothness 8
Direction [✓ Up] Down
Tile mode ☐

Figure 61: Example of using SpriteIlluminator

It is possible to modify the three parameters that control the quality of the normal map (Figure 61).
Although it is also possible to manually make the normals map, painting them individually.

<u>Scenario</u>: SpriteIlluminator allows to create the normal maps automatically (objects to the right of Figure 62) or manually (objects to the left of Figure 62), houses give better results with flat colors, that's why it has been done manually.

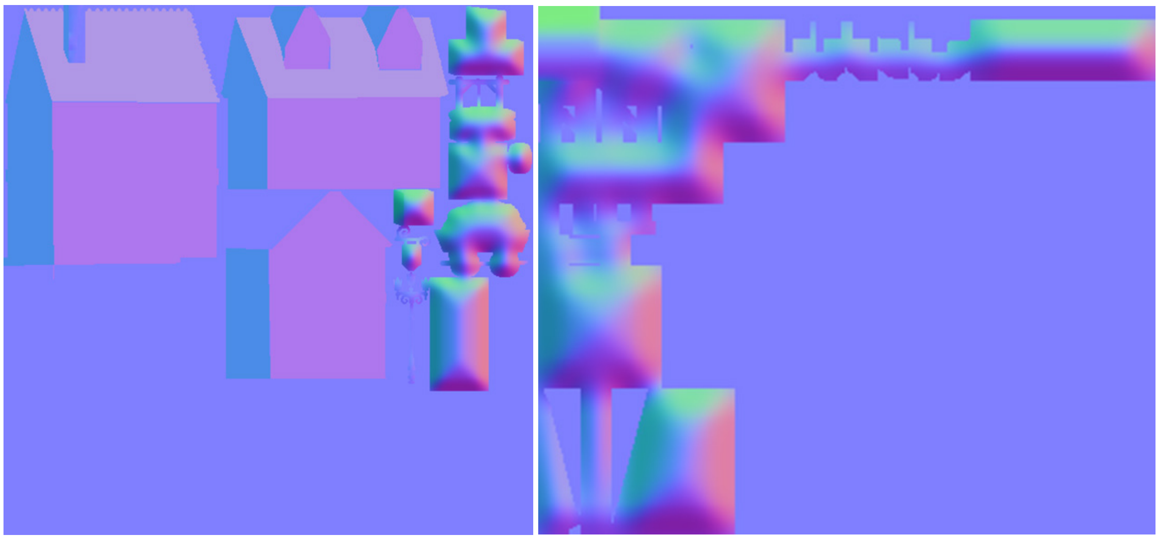


Figure 62: Normal map results in the scene elements

Main Character: When creating a normal map of a character, all the images are together (although there are some that appear mixed, it does not matter, Unity detects which animation it belongs to), unlike the “spriteSheets” that need a “strip” for each animation

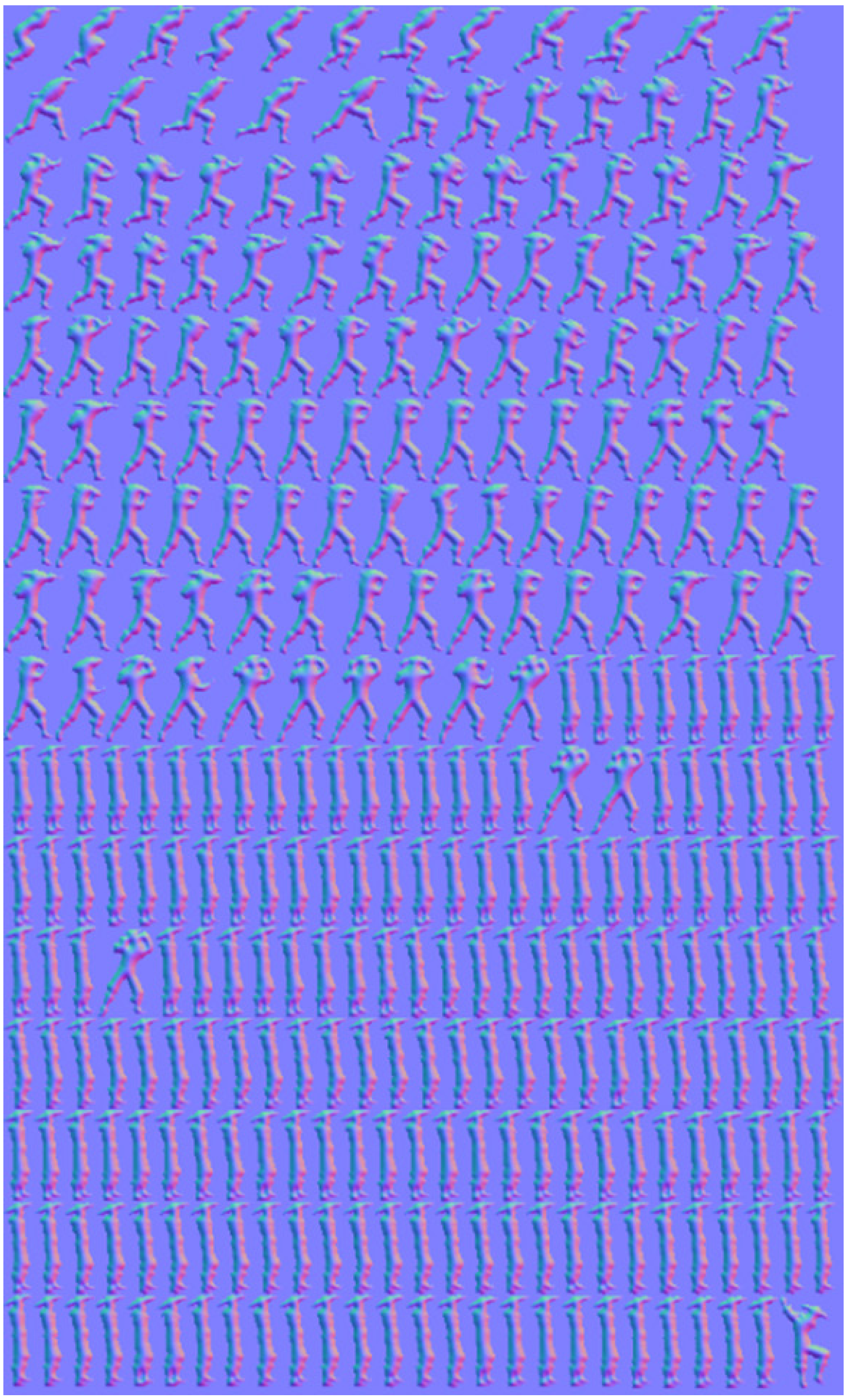


Figure 63: Results of the “normal map” in the main character

Emmision map:

To put points of light it is necessary to use "emission maps", which allows creating them in different places (Figure 65):

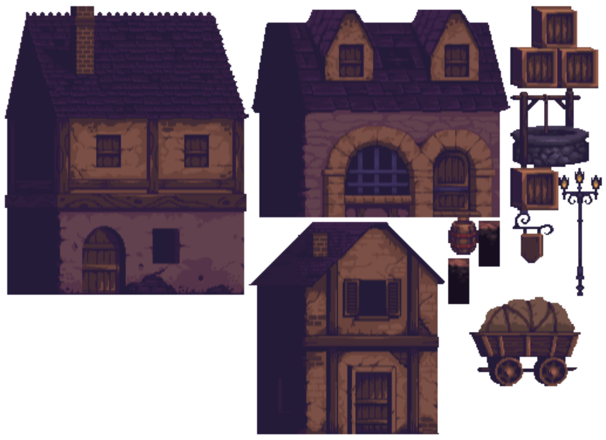


Figure 64: Image containing elements of the scene

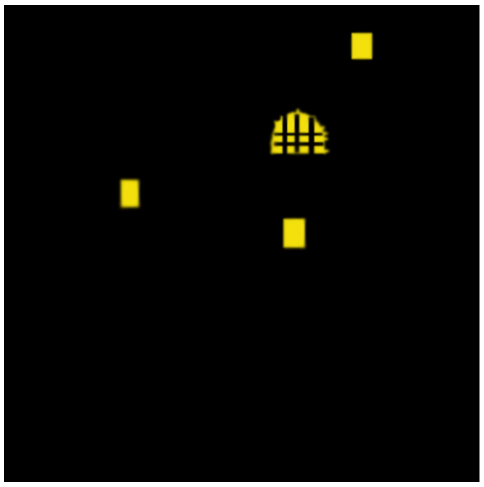


Figure 65: Emission map to put points in houses

If the next two figures were aligned, the light spots could be seen in the windows:

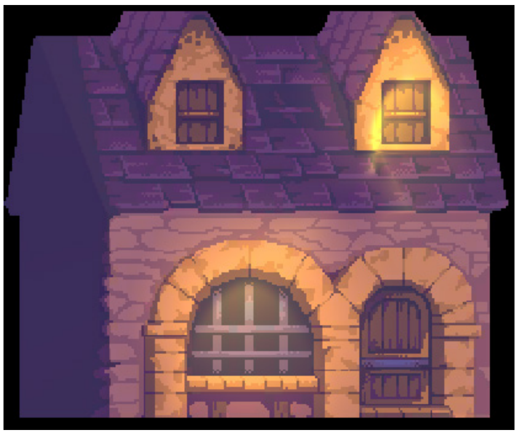


Figure 66: House in Unity3D with scene lighting but without the emissions map.

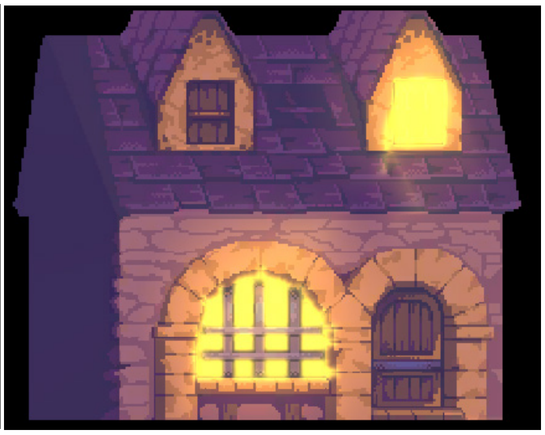
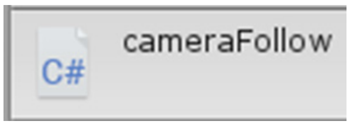


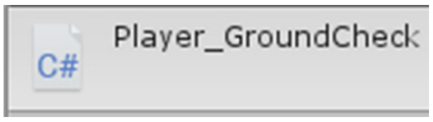
Figure 67: House in Unity3D with scene lighting and emission map.

Figures 66 and 67 have a saturated brightness because it seemed too dark for this document. In the Demo it is not really like that, but in this way we can appreciate the change.

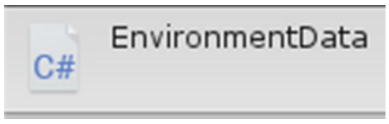
Scripts:


C# cameraFollow

The camera follows the player


C# Player_GroundCheck

Detects the surface of the character's feet to see if the player can jump


C# EnvironmentData

In addition to "Player_GroundCheck" it detects the character jump, but this Script is used in "InteractableObjects" to allow sliding vertically with the "S"

C# movement

This Script aside from the basic movement controls the animator and the references to the other scripts to control the functionalities of these scripts (such as the jump or the interactions with the environment)

C# InteractableObjects

This Script works as a platform detector to know if the player are in the cart or not and to get off with “S” as in traditional platform games.

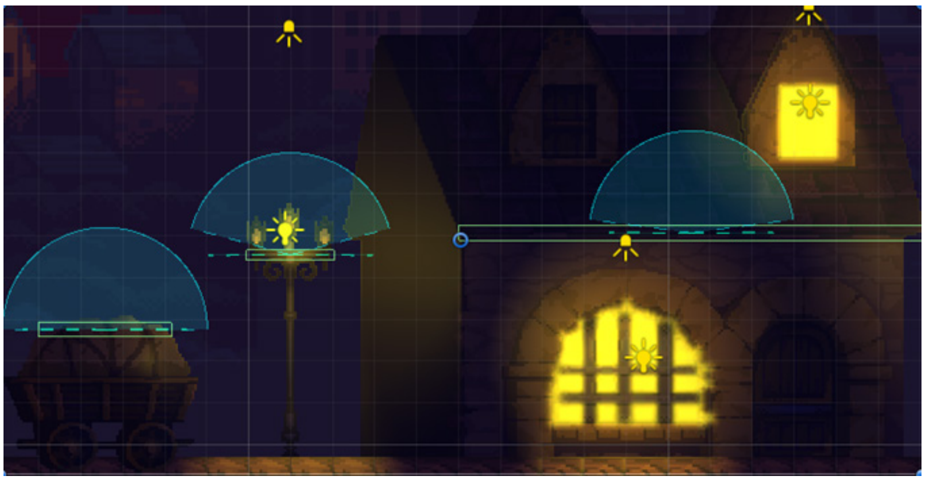


Figure 68: Objects in unity with the script “InteractableObjects”

C# CarretaMove

Detects if the player is on the cart, as long as the player does not get out of the cart, the player can use it as a “means of transport” with greater speed.

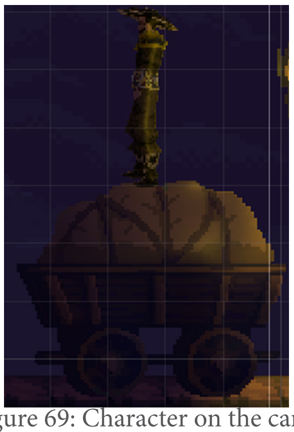


Figure 69: Character on the cart

C# BoxScript

Detects if the player is within range of the box, if he is in range and the key “Q” is pressed, the box will be picked up and transported with the player, matching the speed of the box to transport with the player, walking or getting on the cart.

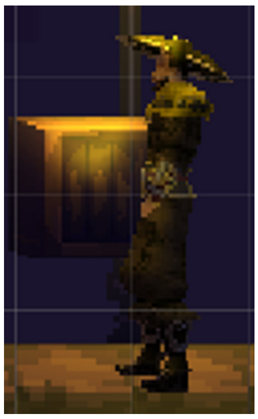


Figure 70: Character picking up a box

4.1.2 - Demo2

For Demo2 a “Space Shooter” has been created. This demo is the most similar to a video game. While the other demos have been created with the purpose of showing the objective that was sought, in this demo, in addition to that, we wanted to show something more like a video game. For this demo we have explored more changes that could be made in “Blender” before importing it into “Unity”.

Instead of adding an animation to the characters, creating the animation by positioning the camera in different ways in Blender. Also, use simple elements that Blender can offer and polish them until satisfactory results are obtained, for example, the “bullet” that the characters are going to shoot.

Main character:

The character in figure 71 is the given asset and the one in figure 72 is the character with all the pixelation effects previously discussed

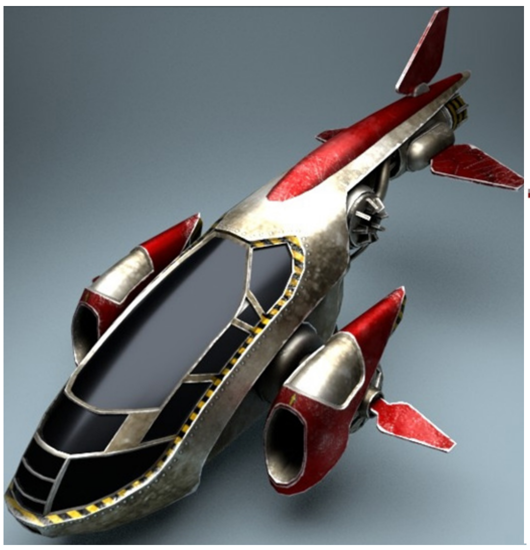


Figure 71: Demo2 main character without modifications.

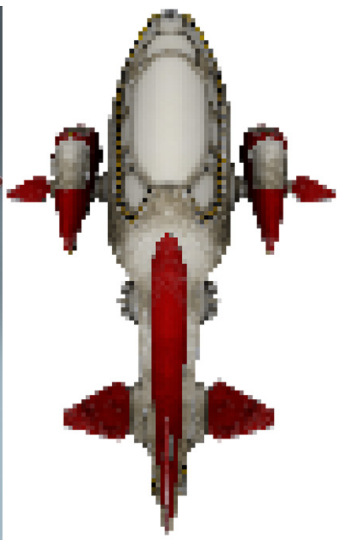


Figure 72: Main character after applying 2D transformation techniques

One of the capabilities of having a modeling in Blender is that the camera can be placed at any angle of the scene, which can generate different animations, in the case of Figure 73 the movement of the character.

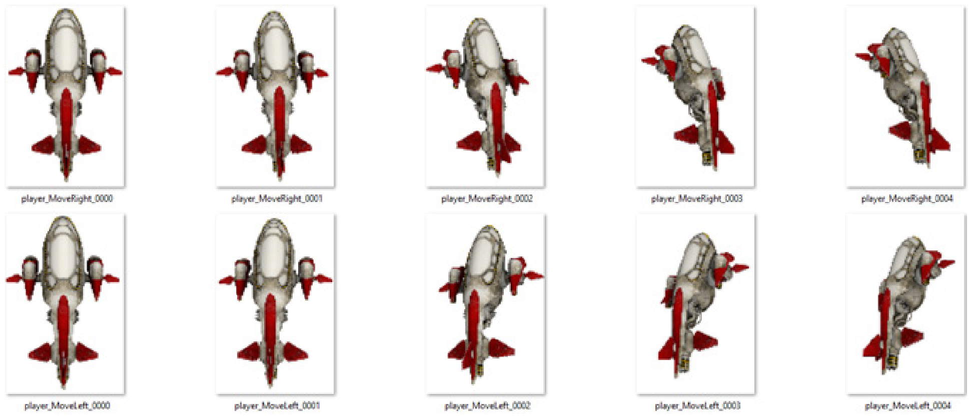
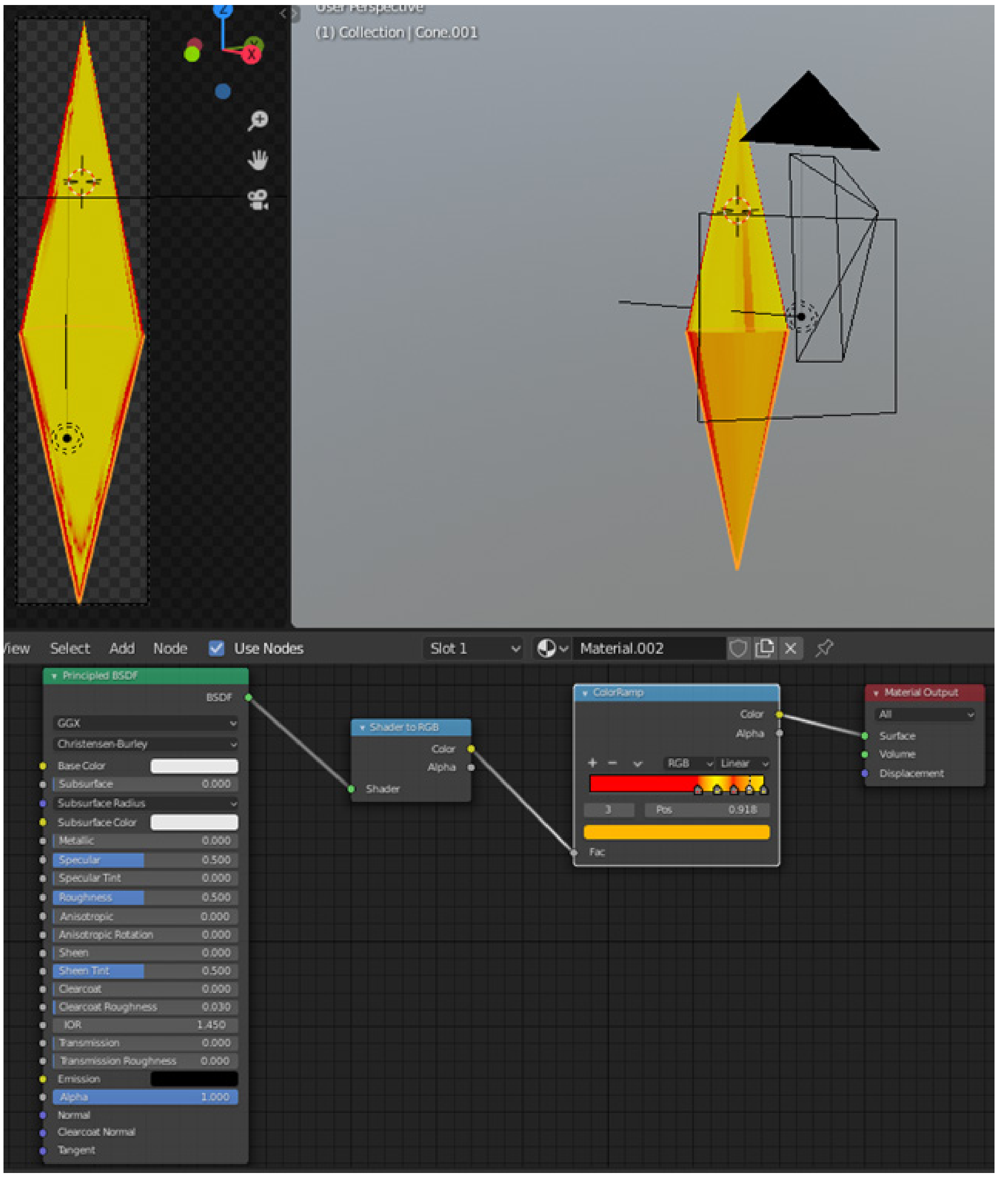


Figure 73: Movement of the character using the turns of the Blender camera

Bullet of the Main Character: At first the bullet was going to be a sphere

There are two cones placed in mirror mode, with some modifiers to obtain a cartoon color



Principled BSDF
BSDF
GGX
Christensen-Burley
Base Color
Subsurface 0.000
Subsurface Radius
Subsurface Color
Metallic 0.000
Specular 0.500
Specular Tint 0.000
Roughness 0.500
Anisotropic 0.000
Anisotropic Rotation 0.000
Sheen 0.000
Sheen Tint 0.500
Clearcoat 0.000
Clearcoat Roughness 0.030
IOR 1.450
Transmission 0.000
Transmission Roughness 0.000
Emission
Alpha 1.000
Normal
Clearcoat Normal
Tangent

Shader to RGB
Color
Alpha
Shader

ColorRamp
Color
Alpha
RGB Linear
3 Pos 0.918
Fac

Material Output
All
Surface
Volume
Displacement

Result:

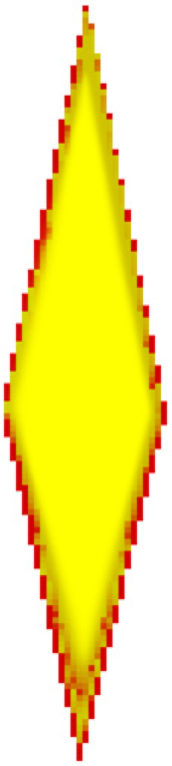


Figure 74: “Shading” of the main character’s bullet

The enemy:

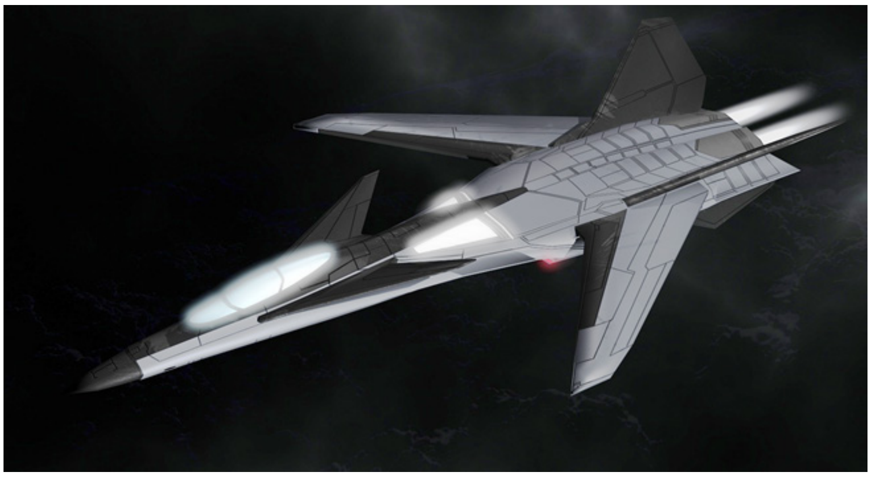


Figure 75: Enemy without modifications.

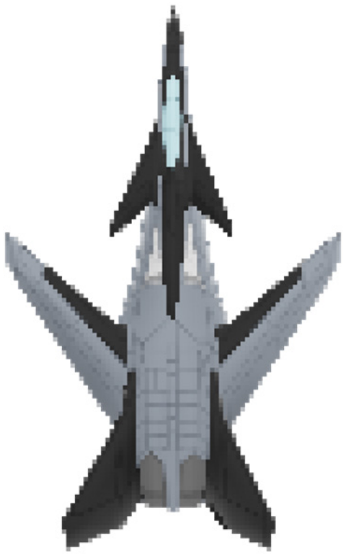


Figure 76: Enemy after applying 2D transformation techniques

The enemy bullet:

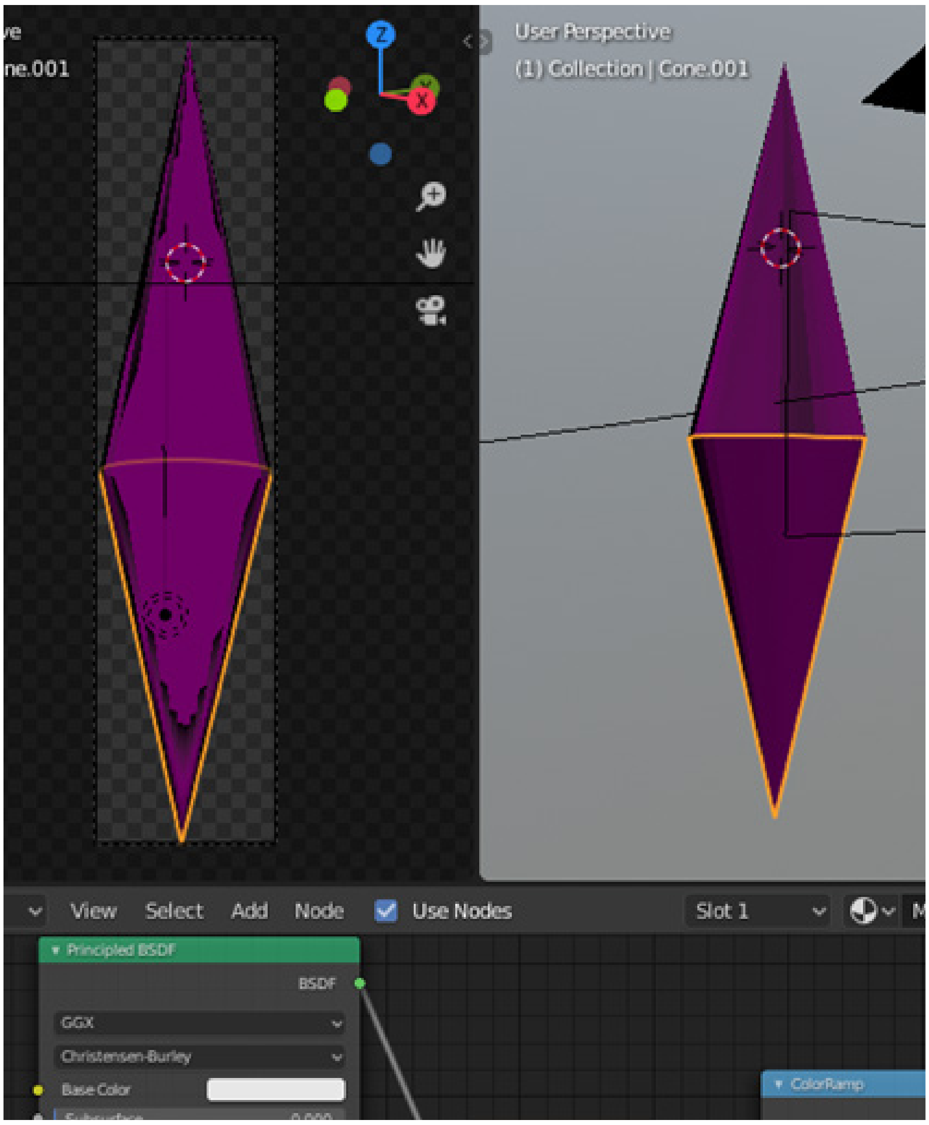
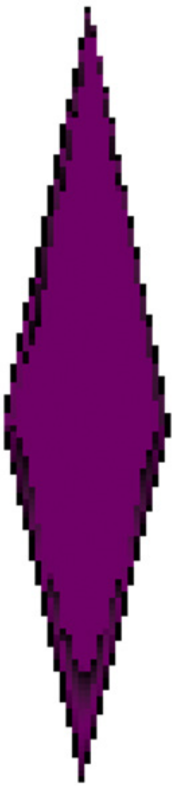


Figure 77: "Shading" of the enemy's bullet

Result:



Enemy explosion: Asset taken from "opengameart" to give feedback to the player when an enemy has been eliminated

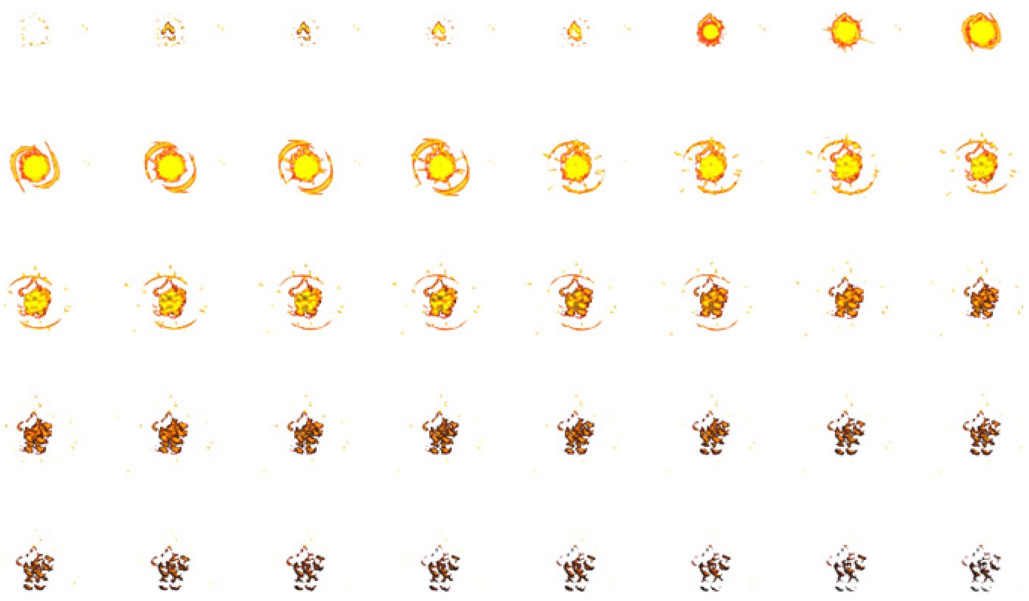


Figure 78: Enemy explosion

Background: Asset taken from "opengameart", this demo is a space Shooter, so we have searched for a background that would give a feeling of space

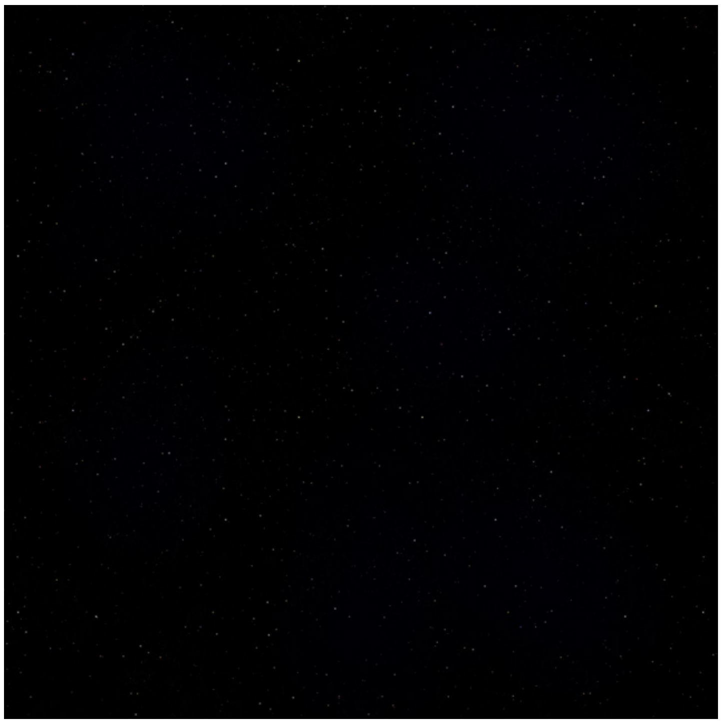
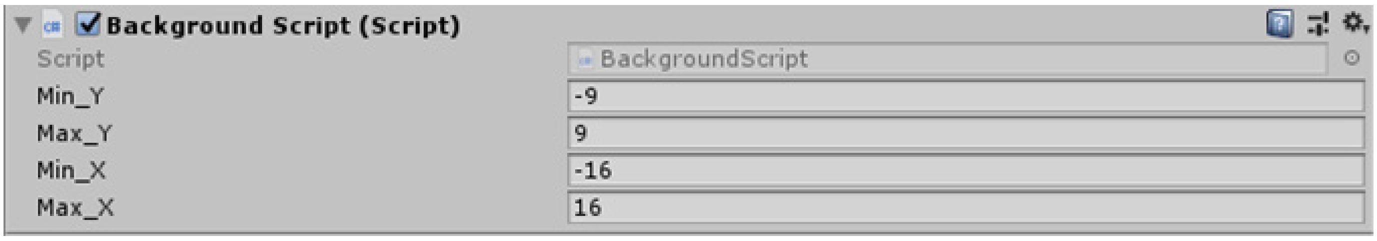


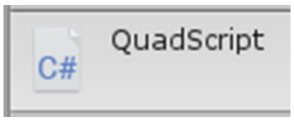
Figure 79: Demo2 background

Scripts:

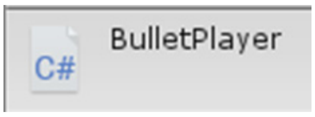
This script saves the minimum and maximum values of the vertical and horizontal position to prevent the player from leaving the scene and control that the enemies are born and disappear out of scene so that it is natural visually


Background Script (Script)

Script	BackgroundScript
Min_Y	-9
Max_Y	9
Min_X	-16
Max_X	16

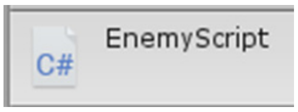


This script generates a visual movement on the scene, makes the background texture scroll infinitely horizontally by repetition even though the object does not physically move



Script that will be in the prefab of the player's bullet, its speed will be controlled, where it is instantiated and when it disappears (when it collides with an enemy or leaves the scene)

It acts similarly to the "BulletPlayer" script, but changes its speed to be negative (so the bullet will go towards the player) and collisions (to affect the player).



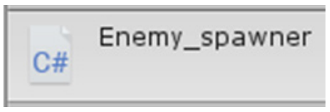
This script controls the enemy speed, creates a different "zig zag" pattern for each one using random and the sound and animation of its destruction.

Controller_Player

This script has two functions:

<u>MovePlayer ()</u>: Movement of the character, is performed using the "Input.GetAxis ()" to compare vertical and horizontal movement. The "BackGround_Script" is also used to obtain the limits of the screen and that the character does not visually leave the scene

<u>Attack()</u>: A bullet is instantiated and the delay between shots is controlled by a "timer"


Enemy_spawner

Through the prefab of the enemy, it generates them with a "timer" that acts as a delay

Score_script

Each enemy eliminated will add 10 points to the total score

Level_script

The game starts with level 1, and every X number of enemies eliminated adds 1 level, this number of enemies is decided by a minimum and maximum Random, level 10 is the maximum level, with each level the difficulty will be greater.

EndGame_Demo2

When the player dies the death screen comes out, putting the score and level in the center of the screen and a button to restart the level.

Restart_Demo2

It restart the level, it is connected with the script "EndGame"

4.1.3 - Demo3

For Demo3, 3 different types of enemies have been created, a "golem", "human" and "wolf". It has been wanted to do in this way, to work with the interaction of a human, a non-human and an animal. The interesting about this demo is to observe the interaction with the enemies, this is the reason why the protagonist has been created not to die and different attack patterns have been given to the enemies.

The first thing that was needed for this Demo was the protagonist and an attack to defeat the enemies that would be added. The choice was to reuse the character from demo1, but without applying the normal maps (figure 80).

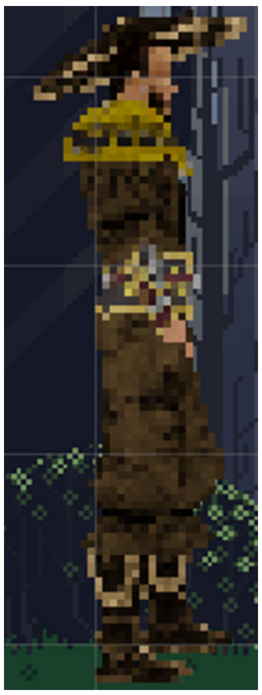


Figure 80: Demo3 main character

The attack that has been added is a slash, to give visual feedback

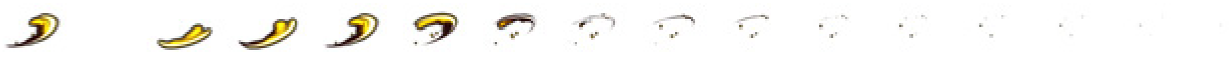


Figure 81: Slash of the main character

Enemies:

After searching for several enemies 3 have been selected

- Soldier
- Golem
- Wolf

Soldier: Modeling in human form, this character, in addition to being human, was the first option because it had many polygons, it was interesting to observe if Blender could support expensive modeling and make the corresponding sprite sheets. This character attacks closely (figure 82).

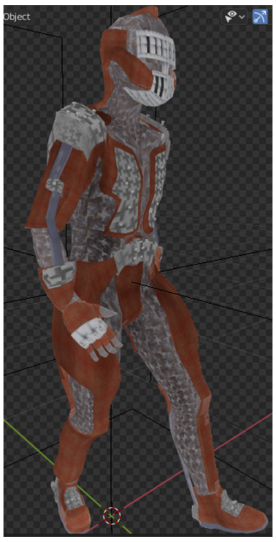


Figure 82: Demo3 Enemy Soldier

<u>Idle</u>: Character's resting state, does not chase the player

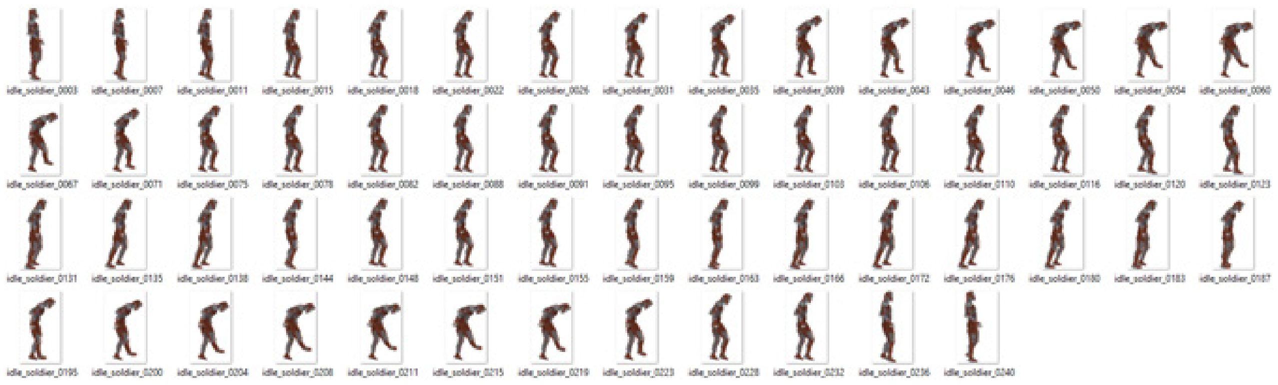


Figure 83: Idle of the Soldier

<u>Attack</u>: When the player approaches, an attack is made at close range.

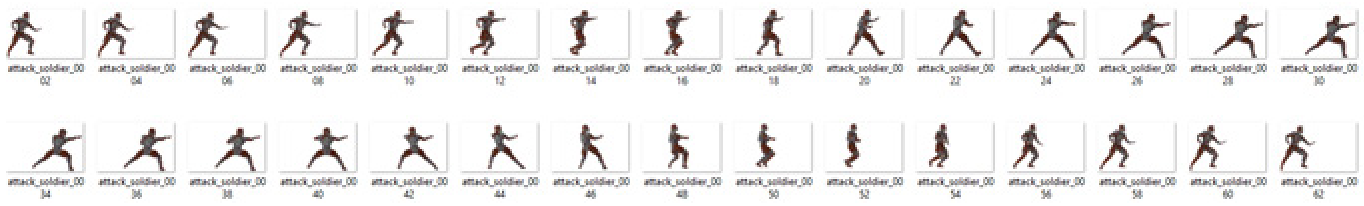


Figure 84: Attack of the Soldier

Walk: Chase the player when he enters his search range.

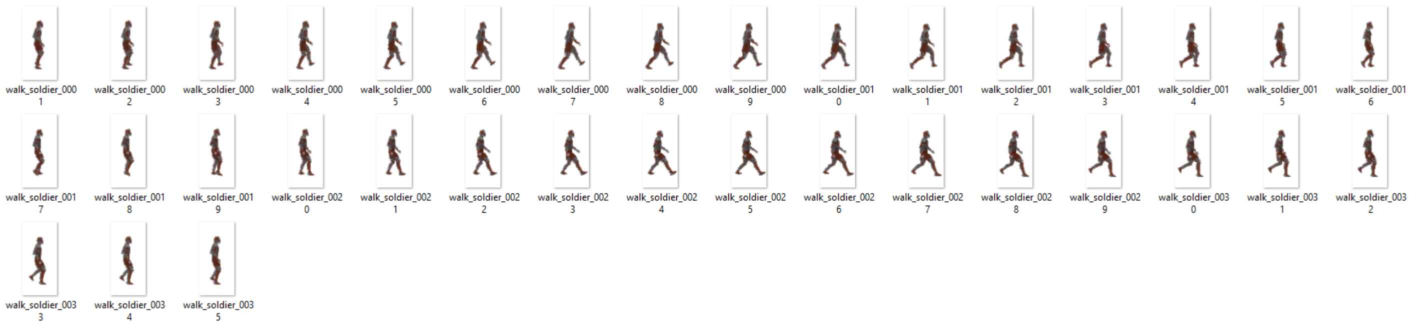


Figure 85: Walk of the Soldier

Death: Death of the character

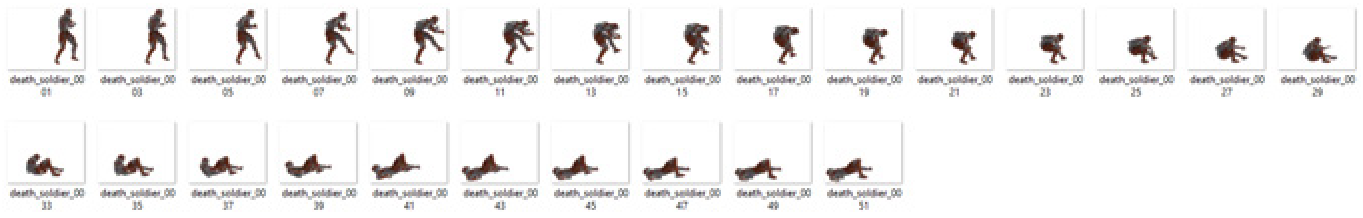


Figure 86: Death of the Soldier

Golem: Modeling in non-human form, this character was interesting to test with non-human characters, for example, the "Death" spriteSheet where only uses his head to create the animation or his "Attack" where instances of the hands are created. This character attacks from afar (figure 87).

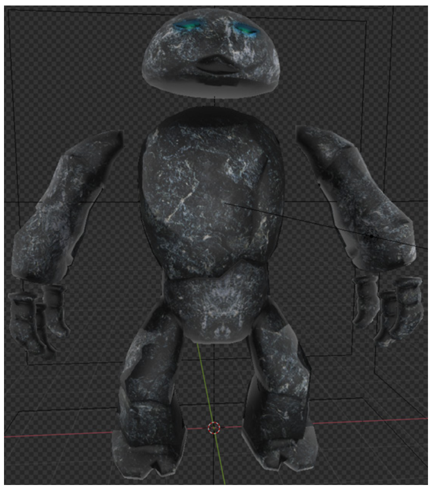


Figure 87: Demo3 Enemy Golem

Walk: Chase the player when he enters his search range.

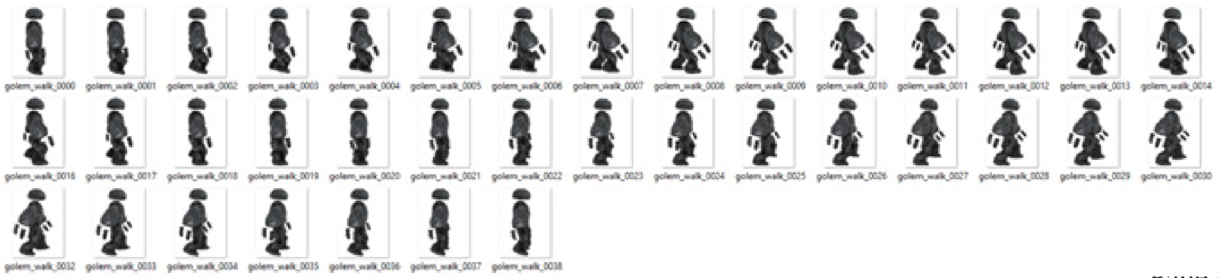


Figure 88: Walk of the Golem

Idle: Character's resting state, does not chase the player

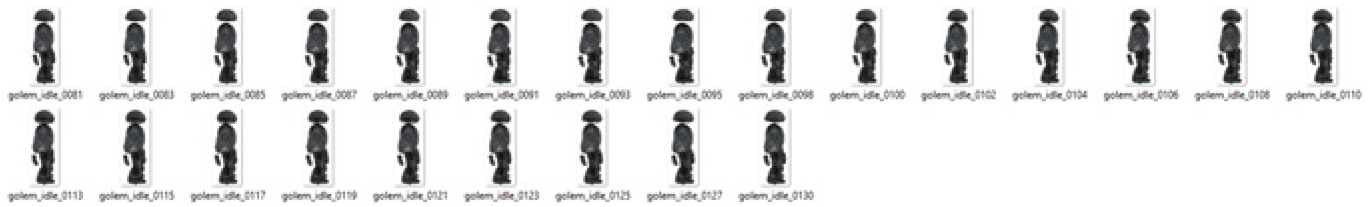


Figure 89: Idle of the Golem

Attack: When the player enters his attack range, he makes a long distance attack

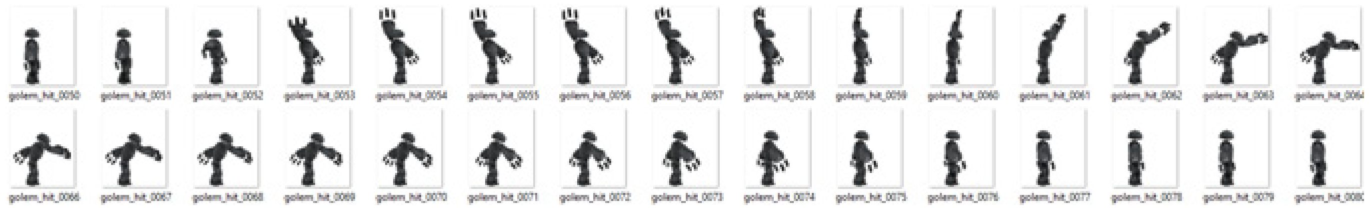


Figure 90: Attack of the Golem

Death: Death of the character, this animation has been created manually, using the character's head from the IDLE state

Figure 91: Golem head (first frame of the IDLE animation)

In Photoshop, using the first image from the IDLE spriteSheet, the body was removed, using only the head that has been manually placed in different positions, creating the animation.

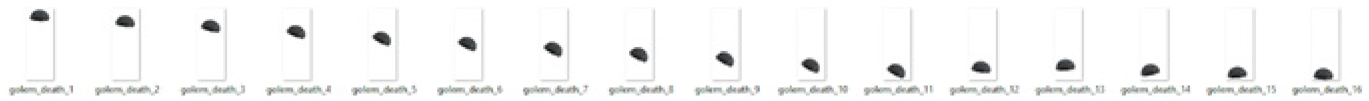


Figure 92: Death of the Golem

This animation could also have been created in Blender, removing the body and creating the animation frames. However, because it was a simple animation, Photoshop was preferred.

A problem found when implementing this character in Unity, is that in Blender when extracting the images (spriteSheets), the animation of "attack" and "walk" were made at different resolutions as can be seen in figure 93, although this has been tried to fix through scripts when doing the change between these two animations, for one second it makes a visual failure, noting that bug.

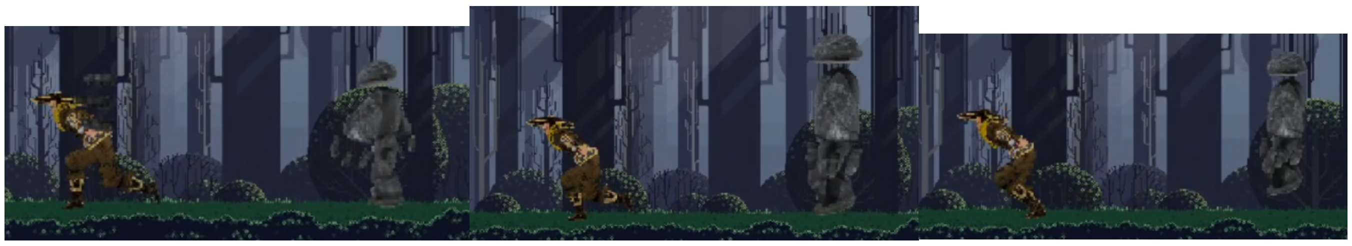


Figure 93: Visual frame error with Golem between "attack" and "walk" animation

This same error occurred to the character when changing from "idle" to "walk", but in this case it was fixed. It is easy to fix, just have to do the pixelation process taking care that it is at the same resolution.
Instead, this bug has been left to be seen in the demo.

Wolf: Modeling in animal form, an animal character was sought to use a non-biped character and observate if this could create problems. According to the obtained result, it does not give problems, but Mixamo does not work for these characters, to animate a non-biped character, the animation must be created manually. This character uses his "Run" animation as attack (figure 94).

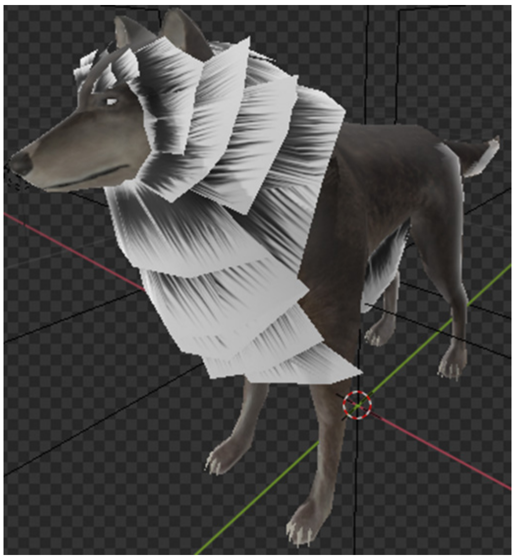


Figure 94: Demo3 Enemy Wolf

<u>Walk</u>: Rest state of the character, unlike the other two characters, this one does not have IDLE, it is walking between two zones established by code, in this state it does not chase the player

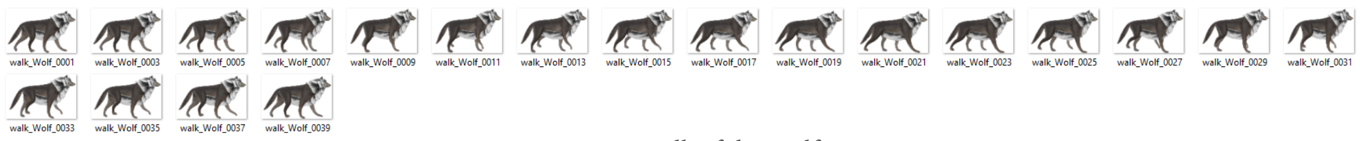


Figure 95: Walk of the Wolf

<u>Run</u>: Stop patrolling the assigned zones and start chasing the player. This state will also be used to attack the player, being a wolf does not have an "attack" animation, it will be physical contact with the player.

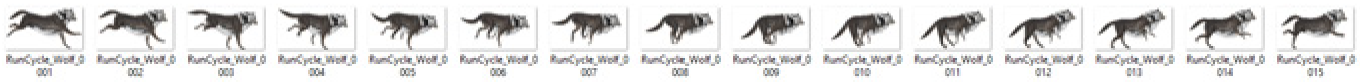


Figure 96: Run of the Wolf

<u>Death</u>: Death of the character

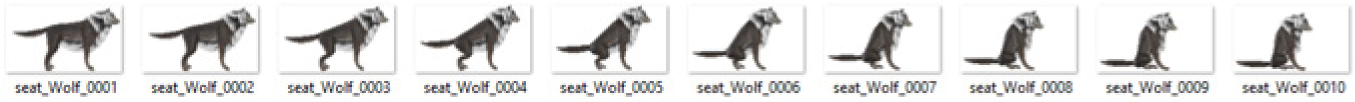


Figure 97: Death of the Wolf

Background: The added Asset has been “free pixel art forest” from the assetStore. After seeing the characters used in this Demo, it was thought that the setting that could best be left would be one of the “forest”, so we looked for one that would give those sensations.

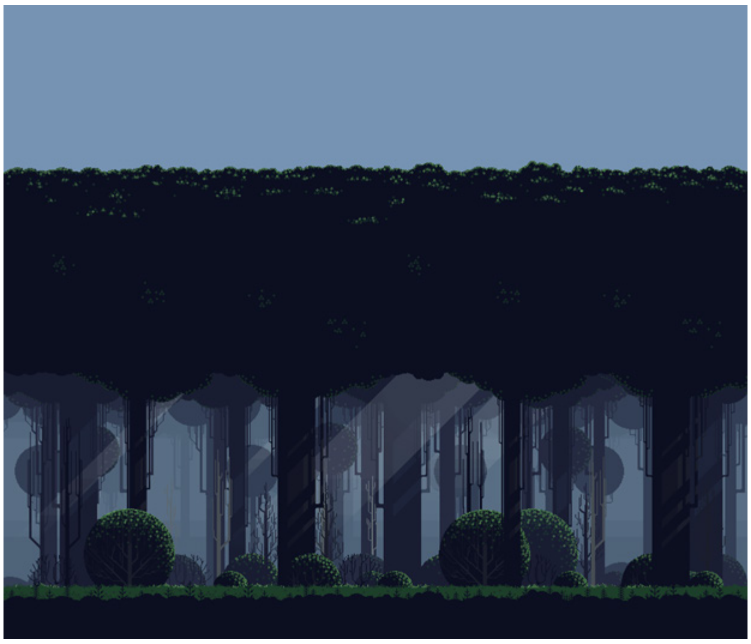


Figure 98: Demo3 background

Scripts:

C# Player_groundCheck_demo3

In the same way that Demo1’s “Player_GroundCheck” script worked, it will control if the player can jump, it was necessary to create another similar script instead of reusing the one that already existed because it uses references to other different scripts and therefore a different one was necessary

C# AttackPlayer_Demo3

This script controls the player’s attack that can be done to the enemies, it has a timer that controls the time between each attack

C# PlayerMovement_demo3

The same way Demo1’s “movement” script worked, it will control the basic movement, character animator and references to the other scripts to control the functionality of these scripts.

C# SoldierScript

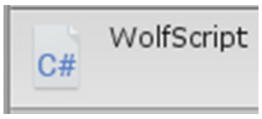
This Script controls the movement and death of the Soldier and has two defined ranges:

<u>The first range</u>: If the player enters, the Soldier will start chasing him and the second rank that is closer

<u>The second range</u>: If the player enters he will start attacking, also has a timer that will act between each attack

C# GolemScript

This Script has the same components as the Soldier Script but its two ranges have different values.



C# WolfScript

This enemy does not wait statically. It has two points that it patrols constantly walking and it has only one range, where the wolf will stop patrolling and will start chasing the player changing from "walk" to "run" (its speed will be increased too)

4.1.4 - Demo4

This Demo is focused on observing the pixelation techniques that are in the AssetStore and why these techniques are not valid. All the options offered by the assetStore are similar, The so-called "pixelation" has been chosen. It only has a Script that added to the camera, it is possible to control the pixelation.

Scene:

For this Demo a 3D environment has been applied, the Asset "Voxel Functional Furniture FREE" from the AssetStore

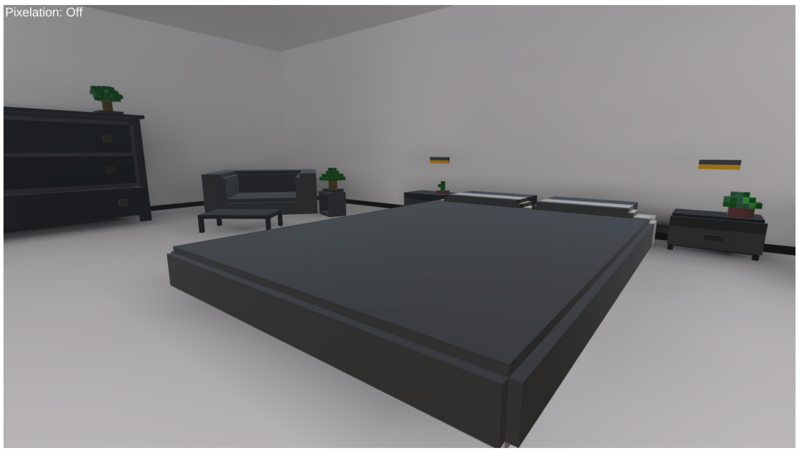


Figure 99: Demo4 scene

Once we have the environment and the pixelation script added to the camera, when using them, we observe that the level of pixelation can be very large (figure 100) or practically non-existent (figure 101). In addition to visually, this will be shown to the user through the "Slider" of the interface located in the upper left.

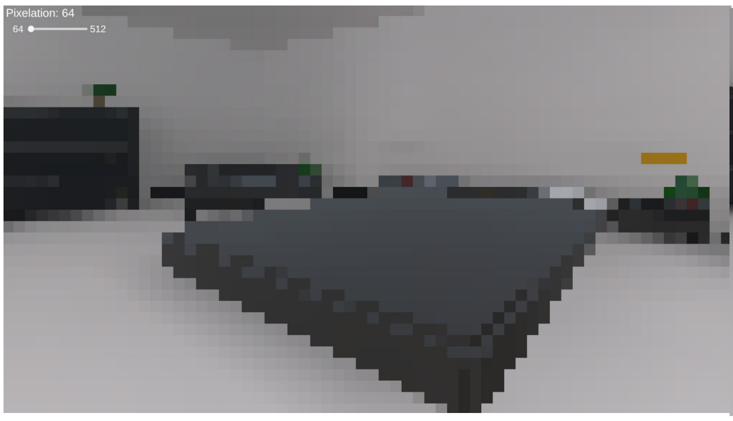


Figure 100: Scene with pixelation at 64 (minimum)

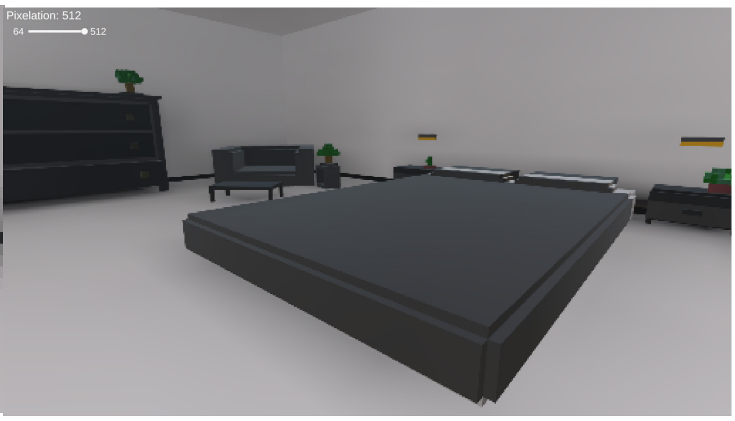


Figure 101: Scene with pixelation at 512 (maxi-

This is a great advantage because it is possible to change the grade of pixelation that interests us through the editor or the Script itself. However, it is not a pixelation of the object, it is a pixelation of the camera, this means that with movement, the pixelation of the scene also changes, a visual change can be seen in Figures 102 and 103.

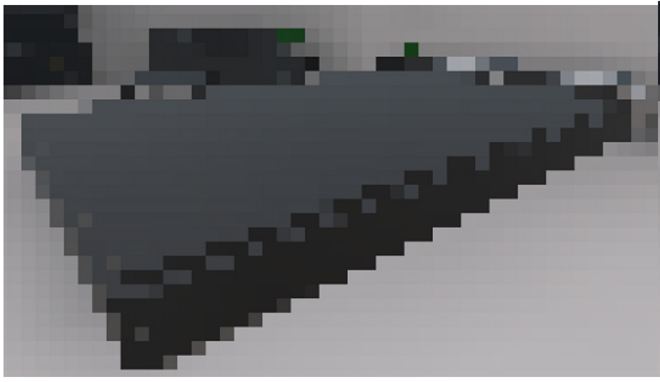


Figure 102: Pixelation example 1

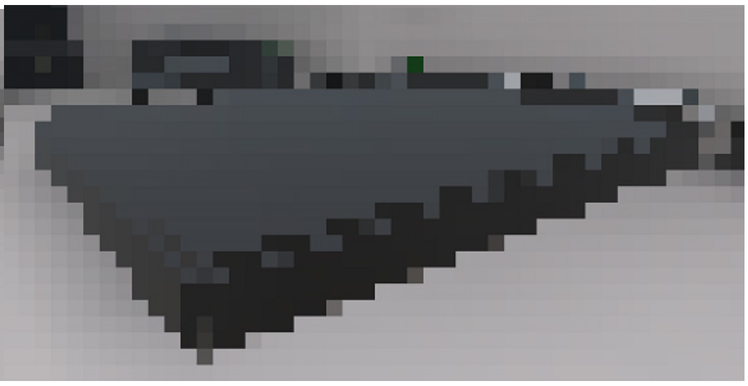
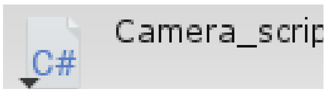


Figure 103: Pixelation example 2

Scripts:



This script is a modification of the "Pixelation" Asset. A pixelation has been added that is activated or deactivated with the "Q" key, increases the pixelation with the "E" key and decreases with the "R" key.

This script creates the basic movement of the character (W, A, S, D)

4.1.5 - Menu

Once the demos are finished, in order to access the demos in a simple way, a "menu" Scene has been created to quickly access the different Demos.

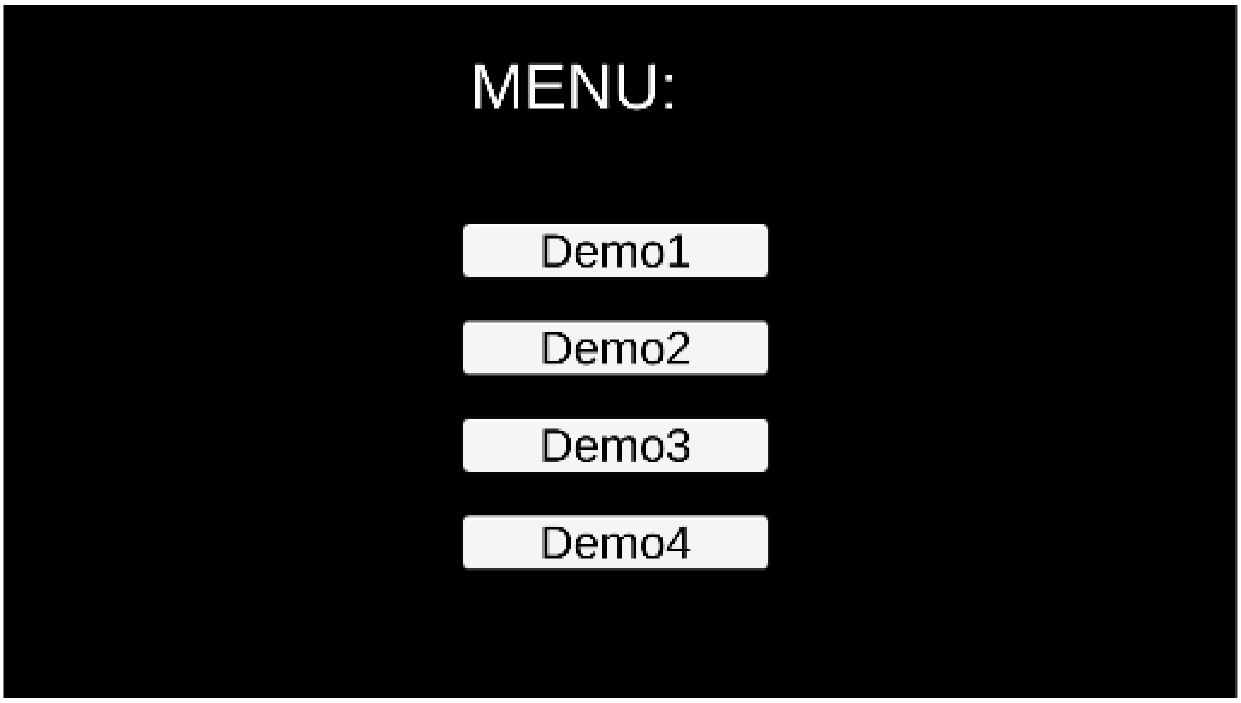


Figure 104: Menu

4.2 Results

The results observed in the "playable demos" have been very satisfactory. Applying the techniques in an adequate way leaves very good results.

Example Demo1:

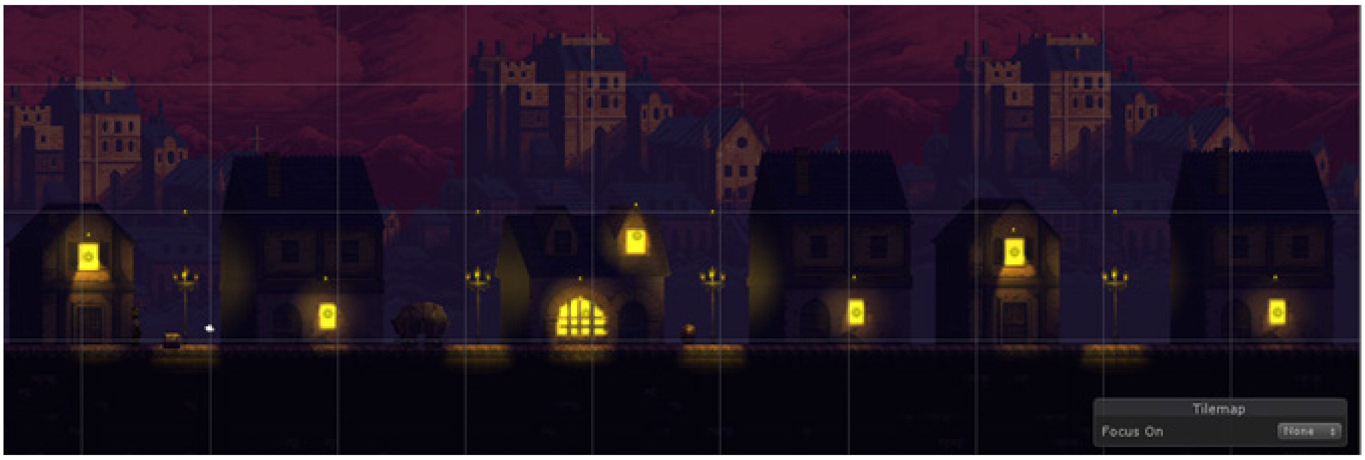


Figure 105: Demo1 scene

Example Demo2:

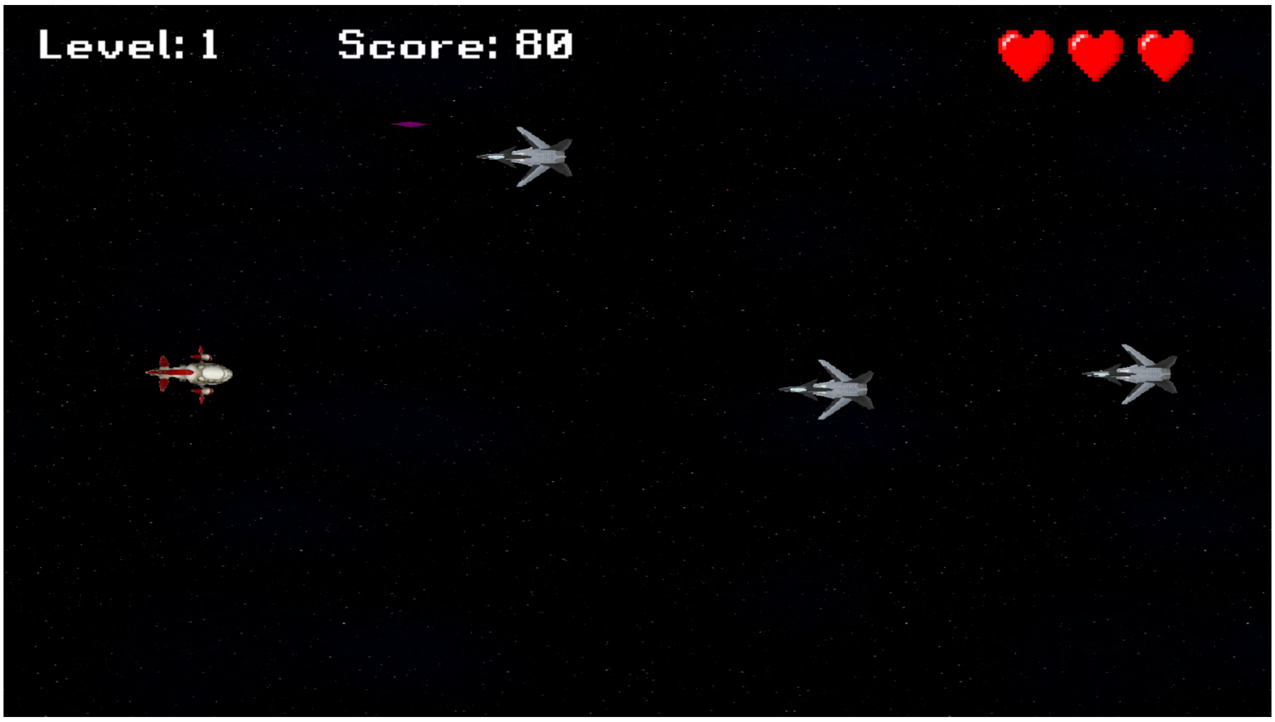


Figure 106: Demo2 scene

Example Demo3:

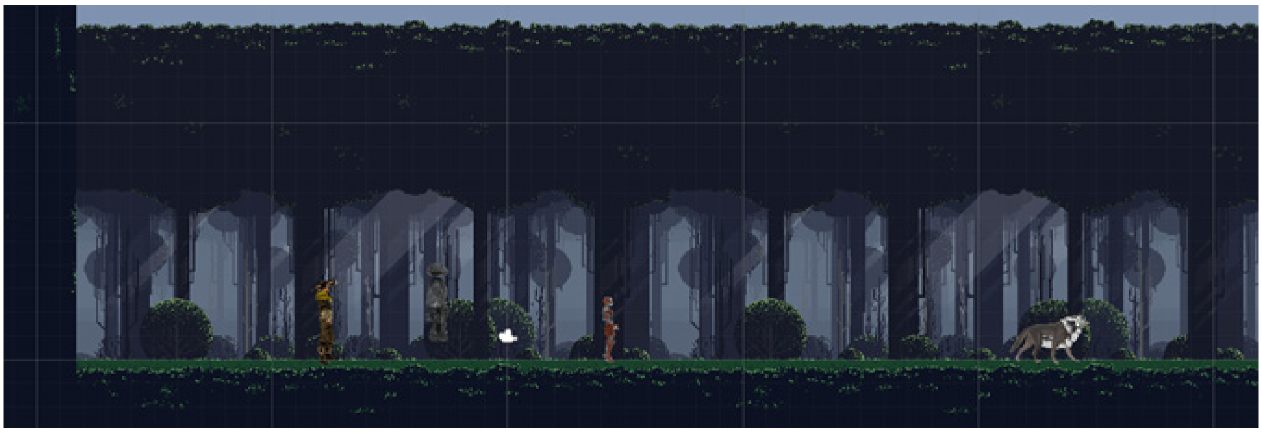


Figure 107: Demo3 scene

However, there have been two problems that must be taken into account when performing this technique.

4.2.1- Scenarios

In this project we wanted to make a transformation from 3D to 2D, which has been achieved with characters but has not been achieved with scenarios, this is because 2D backgrounds are essentially different from 3D backgrounds. for example:

2D Background

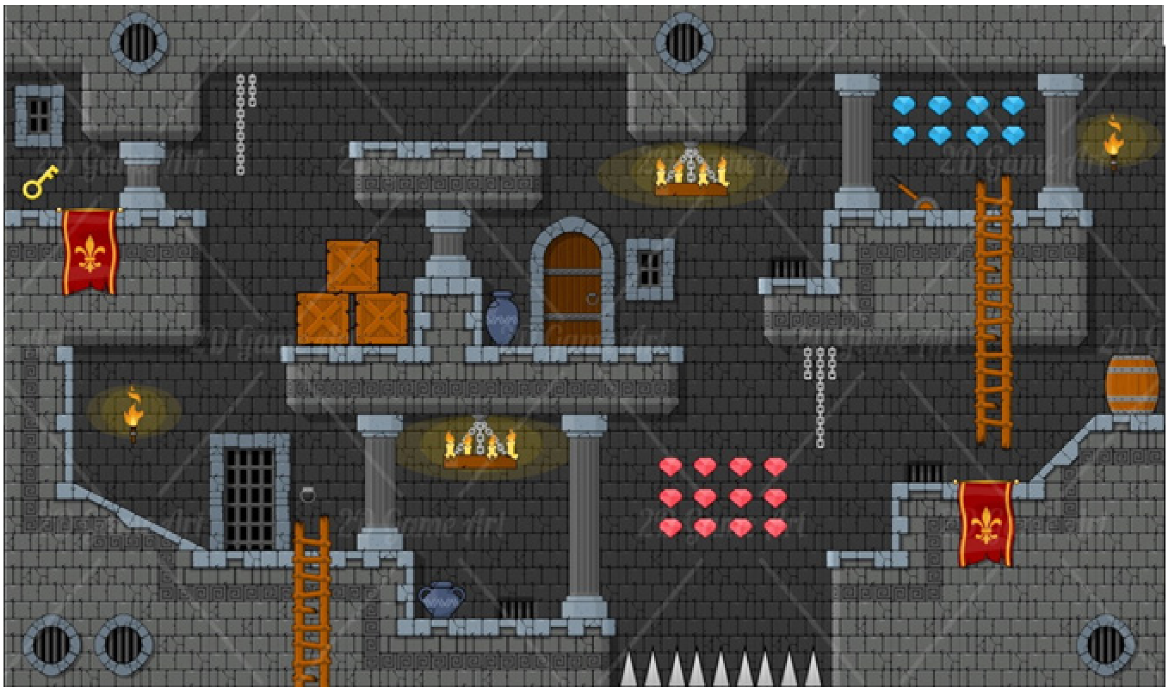


Figure 108: 2D Background Example

3D Background

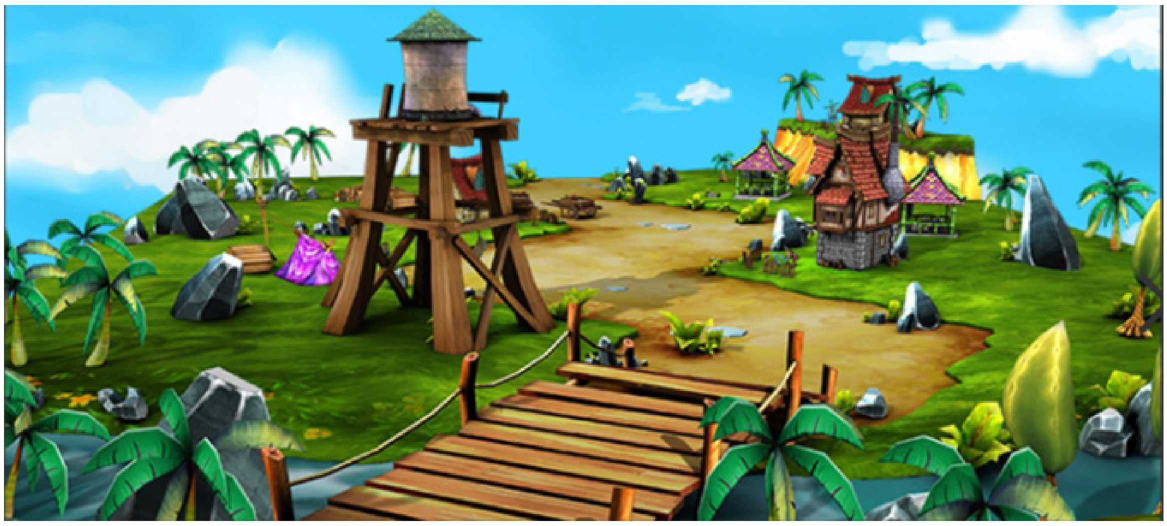
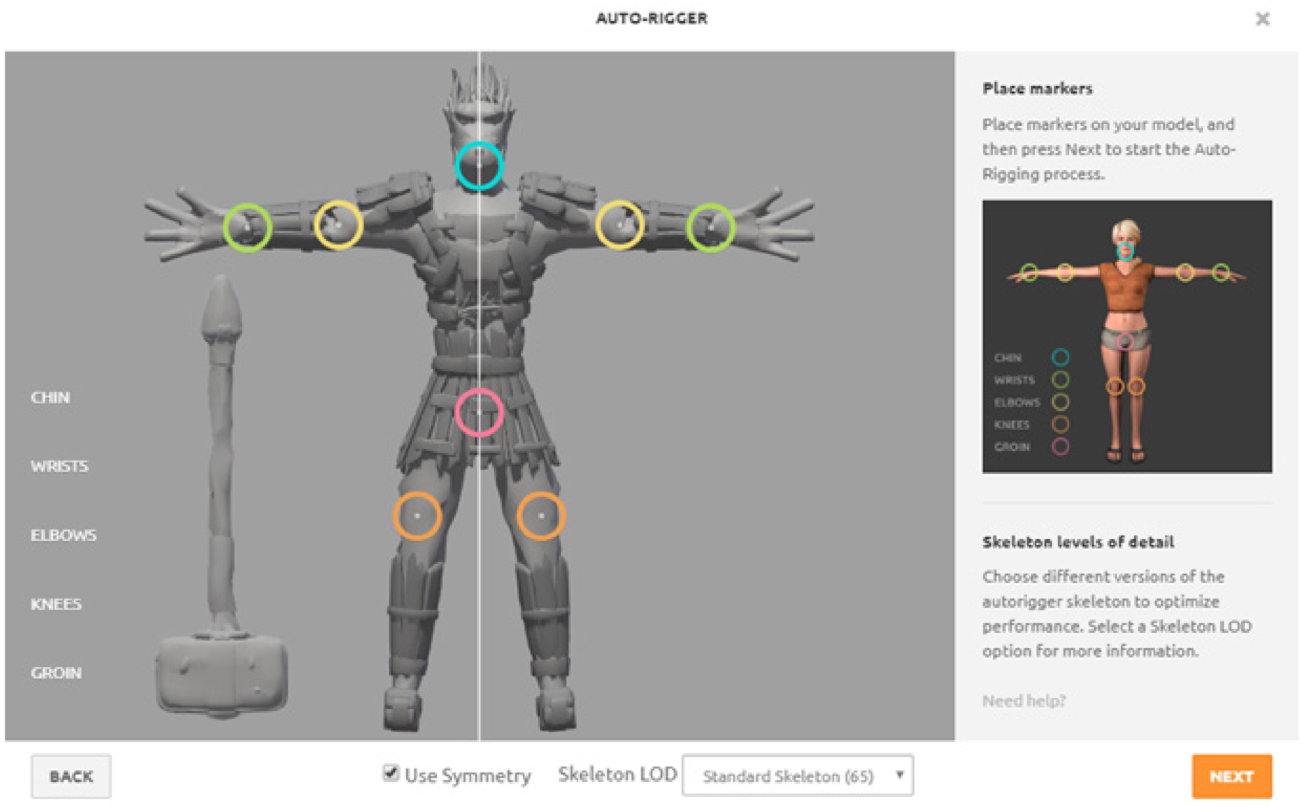


Figure 109: 3D Background Example

The problem encountered is that a 2D background (Figure 108) is prepared to move on the X and Y axes, while 3D background (Figure 109) have depth, being able to move on the Z axis, this implies that a 2D background cannot be created from a 3D background.

4.2.2- Non-human characters

An idea thought for a demo of this project was to make a level only of non-human enemies, but the rigging problem was found, because Mixamo only admits human characters.



Place markers

Place markers on your model, and then press Next to start the Auto-Rigging process.

Skeleton levels of detail

Choose different versions of the autorigger skeleton to optimize performance. Select a Skeleton LOD option for more information.

[✓] Use Symmetry Skeleton LOD Standard Skeleton (65)

Figure 110: An example to observe that Mixamo's articulation points are only for humans

The points that can be placed in Mixamo to rigge the character are designed for biped characters (figure 110).

So the only thing that can be done to animate this, is to obtain a modeling on the internet already rigged or do it ourselves in 3D modeling programs (such as Blender or 3ds Max).

5 - Conclusions and future work

5.1 - Conclusions

The best part of doing this project has been being able to learn this transformation, it is a skill that if it had not been for the TFG, surely I would never have studied on my own, that's why I am grateful to have done it for this TFG

It is a very powerful technique. Using practically any "human" modeling we can create any perspective or angle that we want, so if we want to make a 2D video game and our skills specialize in 3D, it is a powerful technique. This was one of the reasons why the idea about this project started. Also, with enough imagination, you can apply different uses that are probably not explored in this TFG.

It is also applicable to 3D games that are created to be 3D, but you want to remake it (re-create the game) in 2D, in this way you can reuse what has already been created to give another gaming experience.

It is true that it is not a technique that will be used in all the games that a video game developer will create, but as we have seen in this project, it is possible to create a video game based on this technique, as did "Dead cells" (game visually based on this technique).
It is also possible that if a 2D video game is being developed, even if spriteSheets are used, some objects or enemies will be easier to do with this technique and with the same result.

A problem found has been time, there are 4 demos, the most developed has been Demo2 (and still wanted to expand more), this is because with this technique, different uses can be investigated and created. The idea of this project was to test some of these. This is a problem considering the time, because you cannot create an entire video game for everything you want to try. However, this same reason has served to use this TFG as an instrument to learn these different options

5.2 - Future Work

As the project progressed, new ideas emerged to do in each "Demo" and how to apply it to complete video games. It is exciting to think about this and want to apply it. In the future, video games with these polished characteristics will undoubtedly be created, to give the best possible game feeling.

Bibliography:

Normal maps light
https://www.youtube.com/watch?v=VDcWSEs856k
https://www.youtube.com/watch?v=mcTnoGTWkm4

Rig models with Mixamo
https://www.youtube.com/watch?v=0N4gtmDANKo

Import in Blender the Mixamo animation
https://www.youtube.com/watch?v=3ZeoZl5pvq0

Multiple animations from Mixamo
https://www.youtube.com/watch?v=sFkGcHxRsqg

TILEMAPS in Unity
https://www.youtube.com/watch?v=ryISV_nH8qw

ColorMap in Shading
https://www.youtube.com/watch?v=semEVhy5t6U

Use compositing
https://www.youtube.com/watch?v=cLnyKRfBFsQ

3D to 2D pixel in-depth Blender Tutorial
https://www.youtube.com/watch?v=vIKJ1AqfI0Q

Lighting in Blender
https://www.youtube.com/watch?v=Ycz1wQY_7KI

Use of TexturePacker
https://www.youtube.com/watch?v=5cP91980VEE

Shake Effect in Unity
https://www.youtube.com/watch?v=kzHHAdvVkto

Modeling Pages:

https://clara.io/
https://free3d.com/
https://sketchfab.com/feed
https://www.turbosquid.com/
https://www.cgtrader.com/free-3d-models

DEMO1:

Character asset:
https://free3d.com/3d-model/martial-arts-man-34523.html

Background asset:
https://assetstore.unity.com/packages/2d/characters/gothicvania-town-101407

DEMO2:

Character asset:
https://www.turbosquid.com/es/3d-models/free-sample-fighter-3d-model/798902

Enemy asset:
https://www.turbosquid.com/es/3d-models/free-fa-59a-mako-3d-model/1078468

Explosion asset:
https://opengameart.org/content/2d-explosion-animations-frame-by-frame

Background asset:
https://opengameart.org/content/space-backgrounds-0

Músic asset:
https://assetstore.unity.com/packages/audio/music/loop-music-free-111896

Explosions sound enemy

https://assetstore.unity.com/packages/audio/sound-fx/retro-sound-effects-22153
Enemy sound: “explosion_05”

Shoot player sound
https://assetstore.unity.com/packages/audio/sound-fx/weapons/ultra-sci-fi-game-audio-weapons-pack-vol-1-113047
"WPN_SCI-FI_FIRE_06.wav"

Shoot enemy sound:
https://www.freesfx.co.uk/sfx/-space-sci-fi-science-fiction-ship-craft-vessel-fighter-weapons-lasers-laser-shooting-fire-firing-battle-war-ambience-background

DEMO3:

The asset:
https://assetstore.unity.com/packages/2d/textures-materials/free-pixel-art-forest-133112

Golem asset:
https://www.turbosquid.com/es/3d-models/rigged-animation-3d-1488804

Soldier asset:
https://www.turbosquid.com/es/3d-models/free-3ds-model-rigged/1087579

wolf asset:
https://free3d.com/3d-model/wolf-rigged-and-game-ready-42808.html

DEMO4:

Camera pixelation:
https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/pixelation-65554

Scene:
https://assetstore.unity.com/packages/3d/props/furniture/voxel-functional-furniture-free-134903

Results - Scenarios:

Background 2D
https://www.gameart2d.com/platformer-game-tileset-9.html

Background 3D
https://www.artstation.com/artwork/ZZg20