CREATION OF A 3D FIRST PERSON ADVENTURE GAME WITH THE OCULUS RIFT POWERED BY UNITY

DOMINIK KIRNER / MAX SCHUMAYER

G.R.E.C

URBAN SALVAGE CORP

THE GAME

Hochschule Offenburg
University of Applied Sciences

IRON CLOUD

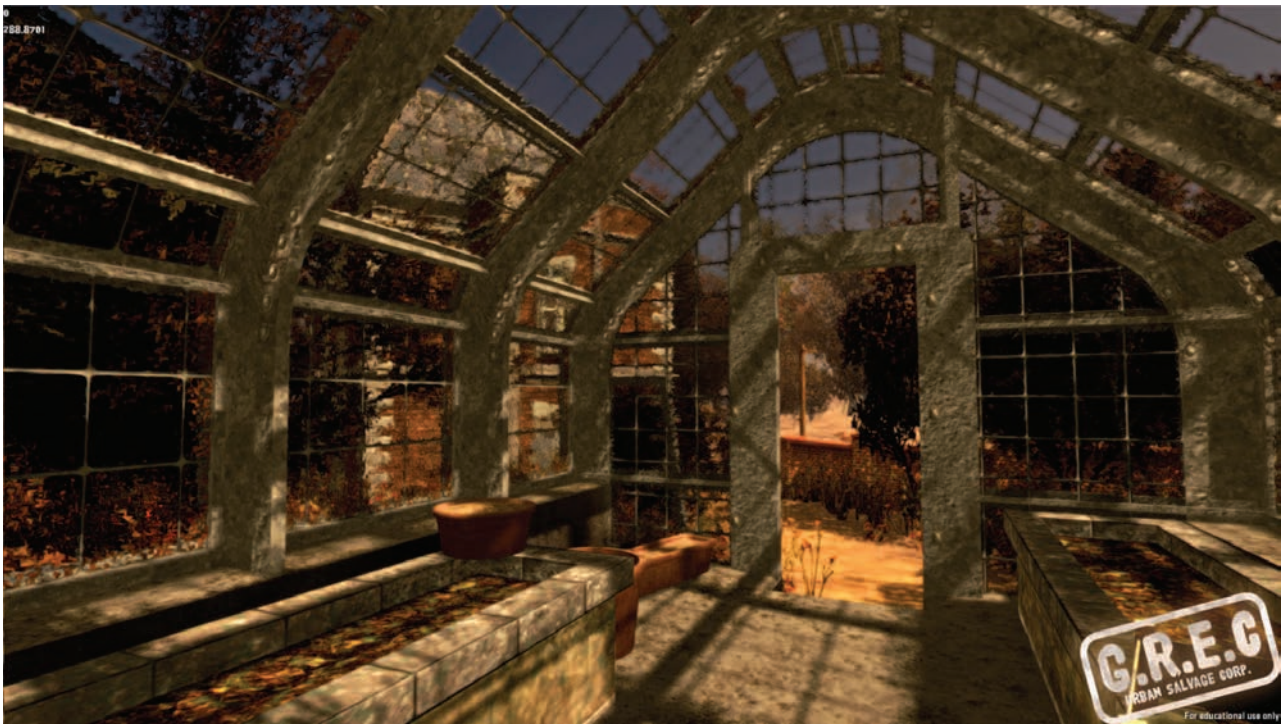
1.GREC Screenshot, the Town Hall from the tracks

2.GREC Screenshot, Vankhart from above

Statutory Declaration

We declare that we have written this thesis independently, that we have not used any other than the stated sources / resources and that we have explicitly marked all material which has been quoted either literally or by content from the used sources.

place, date, signature



3.GREC Screenshot, Greenhouse

4.GREC Screenshot, the countryside of Vankhart Valley

Acknowledgements

We express our gratitude to:

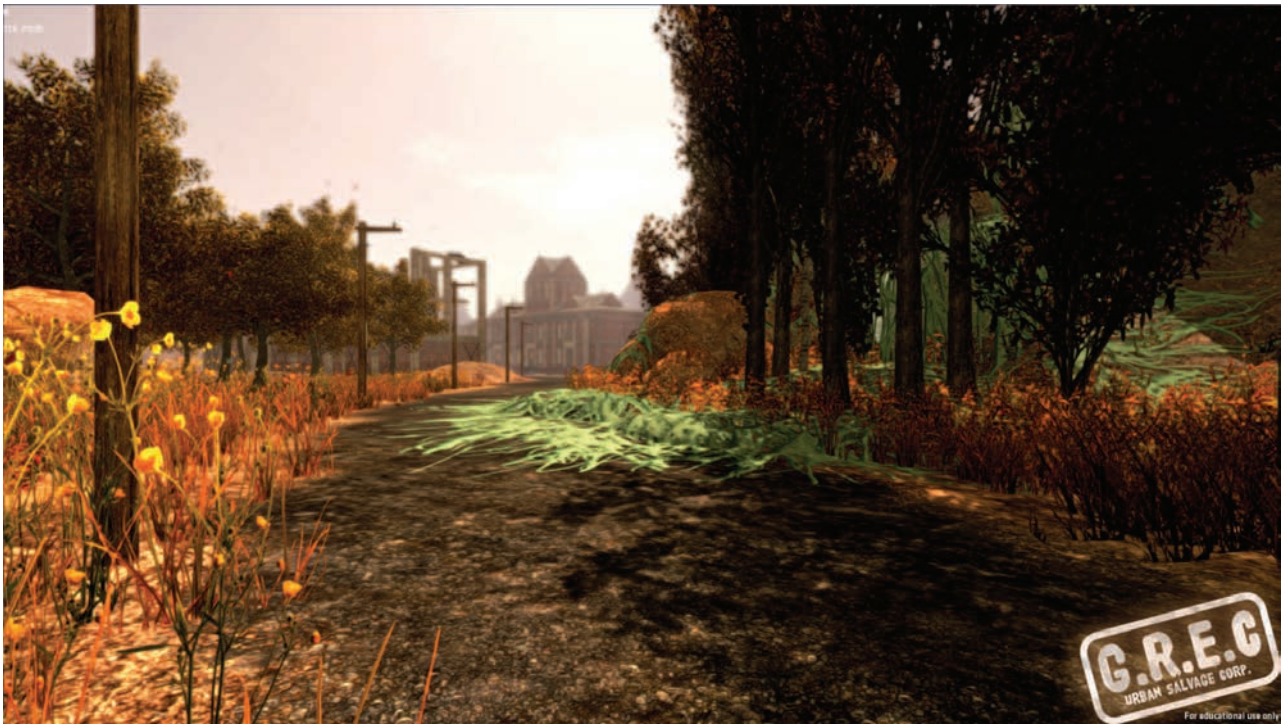
Prof. Sabine Hirtes, and Prof. Dr. Volker Sänger for the guidance and trust.

The University of Applied Science in Offenburg for financing our production process.

Jimena Castillo and Jenny Lehmann for your brilliant work.

Sincere thanks to Timo Hartan, Lee-Anne Gribbon and Catherine Smith for proofreading.

And of course: All those great people around the world who were willing to share their knowledge over the internet.



5.GREC Screenshot, lonely roads

6.GREC Screenshot, a nice view

Contents

7.GREC Screenshot, silence

8.GREC Screenshot, in the streets of Vankhart

Introduction

We could write at this point that video games are a growing part of modern culture and a fast and evolving industry. It is an industry that is known for innovation like the Oculus Rift, and has still room for new and exciting things.
Games are still not what they could be one day, and thousands of people around the world are working on that.
But that's not the reason why we made a game.

We made a game because we wanted to. We wanted to see our character jump, run, stumble, die, and succeed. We wanted to let every leaf on every tree follow our orders. We wanted to shape polygons and color to represent a reality, our reality.

While others wrote pages and pages no one will ever read again, we created something that made people smile and cheer, and kids completely ignore their mothers. We opened a whole new world of vitality and completed our personal climax of multimediality. All of our knowledge, gained over the past four years, is combined in this game. This is our humble contribution to the vast world of modern video games; We hope you will enjoy it.

G.R.E.C - Oneliner

G.R.E.C is a adventure game, set in an dystopien industrial world, where you are a scavenger for hire. Explore the village of Vankhart Valley and grab everything valuable you can get your hands on.
Your trusty old jump boots will help you avoiding the nasty and deadly spores that changed the world of G.R.E.C forever.

Get the game here:
http://www.iron-cloud.de

What is Oculus Rift

The Oculus Rift VR-Headset was originally a crowd funding project on the web platform Kickstarter, and was successfully founded in September 2012. The following pages cover the basics about the Oculus, a virtual reality head-mounted-display.[1]

This is the original description of the campaign:[2]

Step inside your favorite game.

Oculus Rift is a new virtual reality (VR) headset designed specifically for video games that will change the way you think about gaming forever. With an incredibly wide field of view, high resolution display, and ultra-low latency head tracking, the Rift provides a truly immersive experience that allows you to step inside your favorite game and explore new worlds like never before.

We're here raising money on Kickstarter to build development kits of the Rift, so we can get them into the hands of developers faster. Kickstarter has proven to be an amazing platform for accelerating big and small ideas alike. We hope you share our excitement about virtual reality, the Rift, and the future of gaming.

The Oculus Dev-kit 1 had three main features, here the original text from the official oculus web page:[3]

Low Latency 360° Head Tracking

The Rift uses custom tracking technology to provide ultra-low latency 360° head tracking, allowing you to seamlessly look around the virtual world just as you would in real life. Every subtle movement of your head is tracked in real time creating a natural and intuitive experience.

Stereoscopic 3D View

The Oculus Rift creates a stereoscopic 3D view with excellent depth, scale, and parallax. Unlike 3D on a television or in a movie, this is achieved by presenting unique and parallel images for each eye. This is the same way your eyes perceive images in the real world, creating a much more natural and comfortable experience.

Ultra Wide Field of View

The Oculus Rift provides an approximately 100° field of view, stretching the virtual world beyond your peripheral vision. Your view of the game is no longer boxed in on a screen and is only limited by what your eyes can see. The combination of the wide field of view with head-tracking and stereoscopic 3D creates an immersive virtual reality experience.

---

1 https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game/posts
2 https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game
3 http://www.oculusvr.com/rift/

And more technicality about the dev kit 1, you see that the biggest problem is the relatively low resolution of 1280x800 px.

Resolution	Developer kit: 1280×800 (640×800 per eye)
Horizontal FOV	Developer kit: > 90°
Diagonal FOV	Developer kit: > 110°
Head Tracking	Developer kit: 1000 Hz absolute 3DOF orientation (gyr/acc/mag)

Here are quotes to display the great anticipation the Oculus Rift and the idea behind it, coming from the world wide gaming industry and community:[2]

"...the effect of using the Oculus Rift as you play is mind-blowing."
G4TV - Online Blog

"What I've got now, is, I honestly think the best VR demo probably the world has ever seen."
John Carmack, id Software

"...we found ourselves entirely absorbed; a gaming experience with a level of immersion genuinely unlike anything else we have ever encountered."
CVG - online Blog

9.Oculus Rift[4]

4 http://i2.wp.com/www.roadtovr.com/wp-content/uploads/2013/04/oculus-rift-ifixit.jpg

Why the Oculus

We started researching games for the Oculus in the beginning out of pure curiosity. Our University had one acquired so we could use it and so we did. The first experience we had was the tech demo ***Tuscany***[5] made by Oculus VR themselves.

Developer	Oculus VR
Publisher	Oculus VR
Release Date	March 28, 2013
Category	Tech Demo
Genres	Casual, Simulation
Supported Rift Versions	DK1
Supported Controllers	Keyboard, Mouse, Gamepad
Game Modes	Single-Player

We enjoyed walking around in that small world and were amazed by the stereoscopic 3D view; we had never experienced this before. Compared to 3D movies this was way closer to reality.

We played many more free demos, but to that time most games were only small demos filled with gimmicks. Some bigger games had already full support, like Valves First-Person Shooter, ***Team Fortress***[6]. But logging into an online match with the Visor mounted was difficult; you could not read the UI because of the low resolution of dev kit 1. Also being the only one with the head turning based controls on a server with a bunch of mouse jockeys leads to frequent death and harsh insults from your team, luckily you can't read those.


10.Scardeep

Also worth mentioning is the level-showcase ***Scardeep***[7]. It shows what kind of graphical experience we wanted to create.

These testing rounds, with the head mounted display, left us with few conclusions and seasickness.

5 https://share.oculusvr.com/app/oculus-tuscany-demo
6 http://www.teamfortress.com
7 https://developer.oculusvr.com/forums/viewtopic.php?f=29&t=4075

Our thoughts about the brave new virtual-world

Even before we went into planning G.R.E.C we came to these conclusions about the Rift.

First a major feature is the head tracking. In first person games it is used to determine the character direction while moving. That may be a nice gimmick but when you play a game for a long time you end up most of the time in a strange sitting position, or you have to turn around if you want to walk in the opposite direction.
However in simulator games like flight simulators you only move your head in a cockpit and fly the plane with the regular controls.
So if we wanted to make a game with an objective and a game time of more than around 15 minutes we wanted to have a hybrid that would let you move your head freely but lets You control your walking direction with regular controls.
The stereoscopic 3D and the wide field of view was awesome even with the low resolution. It can generate a feeling of height and falling.
So therefore working with high structures (and as we later discovered high-jumping), can create a roller-coaster feeling.

Considering this we focused on the stereoscopy and the look of the game and kept the head-tracking as a working, but minor feature.

The Oculus is maybe not a new idea because 3D visors have been built and have failed multiple times, like the Virtual boy made by Nintendo released in 1995, but is by far the most promising for the future of entertainment and virtual reality.
So we wanted to create something new and unique and were trying to compete with other innovators.

Chapter I

Pre-Production

The Path to Game-Making

Before it was certain that we would make a game, we made four different concepts. A 3D render movie about aerial dog fights, a visual-effect movie about decay and mildew, a single Character design with a complex rig, that should control a human-Character and her robot suit as well as a showcase level for the oculus rift Visor.

All of these projects were intended to be a platform to improve our skills at the same time being small enough to be completed by one person. To cover as many of our interests as possible, we made a list of tasks we wanted to fulfill and divided that into the four key concepts. However the task of combining all of them into one project, like a game or a 3D movie, seemed to be too much for two guys without the assistance of a programmer.

Anyway here is a rough list of the task we wanted to be included:

- A female character design (because our last project included a male character)
- A single realistic rural landscape, including:
    - An airplane
    - A train-track and a steam-locomotive
    - An abandoned factory (basically less fantasy then the Scout had)
    - A realistic balance between farmland and urbanized areas
- A robot suit (basically having one character rig controlling another more mechanical rig)
- A dystopian world, but without war or radiation (the common reasons for destruction)

However, choosing would always reduce our potential result and as a result we would need to focus only on one topic. We would inevitably be forced to perfect that sole task or risk the chance of producing something of inferior quality. Having that in mind we went back to considering the option of making a game, that would combine as many ideas as possible, but still be achievable in the given three month time frame.

To find the best balance between included features and time, we had to analyze our previous projects to determine what was possible and what would be too ambitious.

Postmortem Scout Game

The best source of information was our previous game we made in the winter-semester 2013/14. We made a game as a sequel to our animated 3d movie “The Scout”. We had a team of six project members including us and one additional programmer who programmed the movement system as his own bachelor project. This is our post-mortem of the scout from which we decided the corner stones for the next project.

What went right

The steep learning curve
Personally the most important and positive aspect of our last project was the steep learning curve we experienced. We learned a lot in a very short amount of time, and found that finishing a bad project in a short amount of time was much more enlightening then finishing one, of equal quality, in a greater amount of time. Additionally the results we achieved out of game making, compared to our 3D render stuff (we had previously completed a 3D movie for a course), were much more satisfying. We came out quite content and competed of this project.

Understanding the work flows and Unity
This also applied for the pipelines we learned how all the components work together for a game, basically we realized the simplicity of adding, already created assets into a game-engine. It was possible to calculate the time-effort for an certain object created by one of us.

Creating levels went smoothly
We created several small instance-like maps for The Scout and found that creating those worked out well. The graphics were appropriate for starters and the collision was acceptable for the type of game. I wouldn't play multilayer shooter game on the maps but it served its purpose. In addition we realized unity had its strengths and weaknesses when it comes to sculpting landscapes.

What went wrong

We wanted too much
The initial ideas for “The Scout” was a sidescroller-platformer with a simple two attack battle system.
However once our team grew bigger, our ambitions did too. We wanted a survival game with a rich back story, group management, a fitting fight system as well as a threat level that would lead to random ambushes. Sound too much to handle for 7 rookies? We didn't think so.
In the end a prototype of the map with instance-function and a certain integration of the survival aspect was carried out. The back story and some dialogs were sketched however never polished to a satisfying end result. With hindsight it is now evident that our success would have been greater had we have kept everything simpler.

All assets by ourselves
We had no noticeable budget and on top of that no motivation to outsource anything. Due to these constraints everything would have to be created by ourselves, in order to prove that we can do everything we had imagined. Oh our sweet hubris.


11.ScoutGame

NPCs are an lot of work and a project for themselves
We planned to create NPCs friendly or hostile, like rocks and bushes. But where rocks and bushes are just modeled and integrated and made in one editor, NPCs drag a whole lot of problems and a big workload along. So in the end most planed Characters were concept art on a stand-up displays. You could talk to them and they were able to give an answer but they appeared quite stiff.

Additionally due to the extra work needed on the NPCS, they ended up looking very familiar to our main character, but wore different hats.

Resolution

After this we were able to calculate time and workload for features. Instead of making up more and more features that could never be realized, we learned that we should focus on a specific kind of gameplay. Once the game play is decided we would then boil it down to its components and create them as reliable as possible in order to get a good result.
We also realized we would have to make features that didn't depend on each other or make one feature too important. Separating systems would give us the chance to cut them loose if the final result didn't appeal to us. A game is the sum of its components not the product of one good idea.

Chapter II

Design

Initial Concept for G.R.E.C

G.R.E.C means basically Giant Robot Energy Core, and describes the main objective for the first more adventure like concept. The player was supposed to discover a hidden secret laboratory in an abandoned building than activates a giant robot which would destroy the complete building.
The level should be small but detailed and as graphically-appealing as possible.
To integrate the Oculus Rift head movement we created mind based button riddles on an electric circuit system. The player was supposed to monitor electric-potential with a device in the characters hand and then push the right buttons to open doors or deactivate devices.

By using a hand device we could limit the User Interface to a minimum, that was necessary because the Oculus Dev Kit 1 has a very low resolution (640x800 pixels each eye), which makes it nearly impossible to show any kind of text or small HUD elements.

Our character was supposed to be a secret agent who had to investigate the abandoned building which was using high amounts of energy. This was also the explanation for the riddles.

"One of our sources in the ministry of science stumbled across a rundown fertilizer Factory near your location witch was collecting unusually high amounts of government founding."
(Intro Line, first concept)

Design change

We started with the project around April 2014 and got into Pre-Production in May, we also made the bachelor official so the clock was ticking. Around that time the concept above was done. However we were never really comfortable with the spy secret lab idea but it did server the purpose, we could fit most of the features we wanted in into that concept, but there were always inconsistencies and all seemed very constructed.

We kept working on a small prototype with the goal of showing it to the public for the 50 years anniversary of our university which was on the 23rd May. The Oculus integration and the character controls were already working and as a small gimmick we let the character jump very high. It was fun with the Oculus and we had no other presentable features so we went with that to the public.
The prototype was called Timber Cove and contained a small garden like level on a small peninsula in a lake. We had most of the environment assets including a greenhouse and a lot of garbage.

The reception was great, the Oculus controls were quickly adopted by most of the players and the level was appealing to the testers. We got the best feedback for the high jumping, which worked great with the stereoscopic visuality of the Oculus. It led to a rollercoaster effect that made most people go “woah”. On the same day we decided to give the jumping a higher priority and after a few days we came up with a new concept based on the style of G.R.E.C but with having the focus on jumping and exploring.


12.Testlevel Timbercove. A look through the Oculus

The new World of G.R.E.C

The infested areas should represent an abandoned environment with the technological level and architecture similar to central Europe in the early 20th century. A comparable big conflict like WW1 did happen years before the game and was the cause of the infestation. So the world did move on but the invested Areas stuck in that period. We never intended to show a battlefield, the conflict should be an echo of the past.

In the initial concept of G.R.E.C the player should discover the secrets of an abandoned factory building. We kept that thought by, after the design shifted to a more jumping oriented game. As a result the buildings are massive and industrialized with chimneys, clinker walls and high ceiling, basically jump friendly.
Also we moved away from the concept of one very small detailed factory to a small city to make the scenery more open. No walls around the map corners should block the players view, the scene should blend into distance and be blocked by a deadly spore infestation.

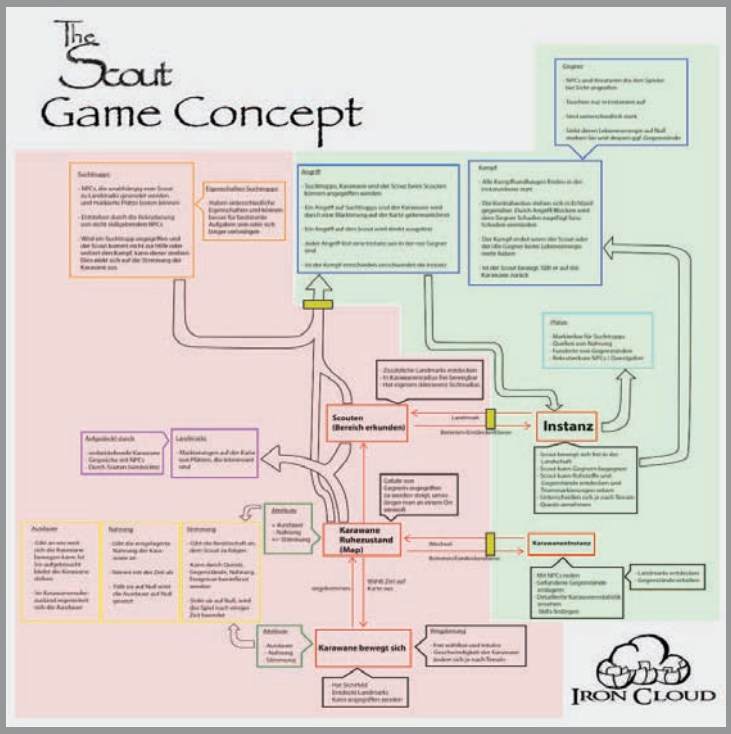
We also wanted to have a balance between nature and cities as well as indoor and outdoor. To get that result we decided to build a small village with two or three factories, a few big buildings and around eight family houses. To make it plausible that two big factories only have a few houses surrounding them, we added a train-track including a station so workers may came from other villages. The buildings should, if possible, have an interior and multiple entrances. To give the player positive feedback for exploring we added a simple looting system to score points.

The landscape was planned as a mix of abandoned fields, small dirt tracks, some streets including the train tracks, open meadows and a forest surrounding everything, of curse most of it infested.

We also choose the autumn as the time of year. Mainly because the red and brown colors are a good contrast to the green of the spores. Also most games are set in summer, even the ones with multiple climate zones generally start in summer, so autumn offered something more unusual.

All further topics are covered in the game design segment but this is basically how the project evolved, and how the world has been planned.

Game Design Overview



13.The Scout Gameconcept

These are the “Roadmaps” we made to have a overview of what is to design and what is dependent on each other. The one for G.R.E.C is way more focused then the more complex Scout Game Concept.

14.GREC Gameconcept

Basic Movement

All necessary animations

- Jump
- Walk (Fwd, Bkwd, StrafeL, StrafeR)
- Run (Fwd only)
- Idle
- Rotation on the spot (left, right)
- Boosting
- Boost Walking (Fwd)
- Boost Rotation (left, right)
- Kick
- Breathing-Gas-meter arm lifting

Walking, speed and direction with Oculus

Gamepad movement
The movement speed and the players walking direction over the map is being controlled with the left analog stick. The pressure put on the stick is responsible for the movement speed of the character. High pressure (stick is pressed as far as it will go in one direction) plays the animation with normal speed. Less pressure plays the animation with less speed.

The right analog stick turns the player on the x-axes. A gentle pressure to the left or right turns the player smoothly in the correspondent direction. With higher pressure (as far at it goes) the rotation is at its maximum. Moving the analog stick up- or downwards has no effect.

Oculus Integration
The Head movement will be transferred with the Oculus Rift onto the players' character. This means the camera rotates equivalent with the players head. A constraining of the characters head-rotation occurs through the natural constraining of the neck muscles.

This setup allows the player to look in any desired direction while moving smoothly somewhere else, this will be a much more realistic approach of controlling a simulated human body.

Keyboard and mouse movement
Controlling the character with keyboard and mouse is slightly different then playing the game with the gamepad. The mouse behaves now like the right analog-stick. Moving the mouse left or right rotates the character in the corresponding direction. Slow mouse movement rotates the player slow on the x-axis and a fast mouse-movement rotates the player fast. The y-axis is blocked and reserved to the oculus.
The keys W,A,S,D are responsible for the players movement.

W = Moving forward
A = Strafe left
S = Moving backward
D = Strafe right

In regards to the action of running, when shift is pressed, the player runs in the corresponding direction. While running The WASD-Layout stays the same, however running is only possible in the forward direction. A smooth movement speed does not exist on this layout. Only the head movement will be transferred with the Oculus Rift onto the players Character. It will offer the same level interactivity as playing with a controller.

Walking, speed and direction, First Person

Gamepad movement
The basic controls will stay the same with or without the oculus. However of course without the oculus, the left analog-stick will also control the rotation on the x and y-axis. Those rotation will be limited by min- and maximum values, due to the fact that without an oculus and a human neck attached to it, there would be no natural limits to the rotations.

Keyboard and mouse movement
Same for keyboard controls. The mouse will now have limited control over the rotation in the x- and y-axis.

Control Settings

Actions	Gamepad	Keyboard
Jump	A	Space
BoostJump (Keeping the Jumpbutton pressed while not moving)	A	Space
Loot	X	Left Mouse Button
Kick	B	Right mouse Button
Metering breathing gas	Y	E
Walk	Left Analog-stick	WASD
Rotate	Right Analog-stick	Mouse
Run	RB	Shift
Questloot status	LB	Q
Pause	Start	Escape
Return game	Start	Escape
Exit Game	Back	Enter
Reset Game	/	R

Breathing Gas

Background:
Breathing Gas is a mix of gases that, combined with oxygen, can neutralize most of the spores in the inhaled air. However this method has its limitations and is not a completely save way of countering the infestation, but it is more economic and flexible then a full body pressure suit.
If a Breathing Gas user comes in direct contact with spore-nests or is stuck in an unventilated infested-room he can still be poisoned, Breathing Gas or not.
It also is self-igniting under high pressure so it can be used for the power jump shoes.

Design

Breathing Gas-Usage
The character permanently uses Breathing Gas, so the max value decreases by one unit per second. On top on this some actions and the environment can dramatically increase the usage.

The following list contains all situations which increases the Breathing Gas usage:

Actions (deducts immediately a fix value)
- jump
- Boost-jump
- run-jump
- Running
- Infestation Zone 3 (are lethal infestation. Breathing Gas will be deducted to zero)

Infestation Zones (increases the per second amount)
- Zone 1 -> small increase
- Zone 2 -> medium increase

If the Breathing Gas reaches zero the character will die and the game will be over.

User-Feedback
The player can always check the amount of Breathing Gas in his tank by looking at a watch like device on his wrist, a button should let the character lift his arm to make it easier to look at the pressure-meter.

If the Breathing Gas value reaches a certain value the character should show that he is slowly poisoned by coughing.
Depending how dangerous it is for the characters health the coughing should differ in its intensity.

If the Breathing Gas is completely used the character should choke to death.

Values
Breathing-Gas initial Value: 1000
Jump: 3
Boost-jump: 15
Run-jump: 2
Running: Additional decrease of one per second
Zone 1: Additional decrease of 10 per second
Zone 2: Additional decrease of 20 per second

Replenish the gas
Breathing Gas easily replenished at the start point (the G.R.E.C plane), the breathing-gas will always be fully restored if the player visits the start-point.

In addition the player can find gas bottles around the map which will increase the gas value by a fixed amount. These bottles should be placed near vehicles and in factories or other industrial infrastructure.

Loot System

Background
The player is an employ for the G.R.E.C Company whose primary field of business is to recover items from the invested lands and deliver them to their clients.
Doing this is technically illegal because the invested areas are under quarantine. However it is endured by the authority as it is sometimes necessary to allow private-contractors enter the infestation even though not officially, so keep your head down.

Most items that need to be delivered have legal or private value to the customers, last will, contacts, birth- and stock-certificates, pictures of grandparents and other things that were forgotten during the evacuations.
But the player can also loot whatever item that has a practical value like jewellery, money and medicine.

Story-Loot
The primary objective for the player is to recover certain items from a specific area in the invested lands. He will receive a list and instructions via radio. These items will be called “story-loot” and are unique. If all the “story-loot” is looted the player has successfully cleared the level and can then return to the chopper.

Side-Loot
Additionally to the primary “story-loot” the player can also collect multiple items of variable value.
These items can be found on various locations in each individual level. Every looted item adds up to a “loot value counter” which determines how high the outcome will be. Individual item names are not recorded, only their fix value will be added to the counter. This makes it is easily possible to collect more than one example of each side loot item.

Types of Side Loot :
- Piles of money/change
- Cigarettes
- Gum
- Medicine/Drugs
- Old Scotch

Loot value counter
The loot value counter is the G.R.E.C variation of a High-Score. By looting side-loot objects the player can increase his score. The player can check his score in the pause menu and when the game is over.

Design

Loot objects of both kinds should be easily identifiable and stand out from the rest of the set dressing items.

Some loot objects should be more or less hidden depending on their value.

The items can be looted by the player, if he stand in a short distance from them and looks directly at them. When the player does so combined with pressing the loot button, a grab animation is played together with a “put stuff in your pocket" sound and the item then disappears.
Depending on the object the value counter or the story loot list will get updated.

The player can loot multiple items per button press.

Infestation

Infestation Story
The infestation is the direct result of massive use of weaponized mushroom spore gas during a large scale war 20 years before the game happens. Now an area 135,000 km³ is uninhabitable for humans.
The infestation is only slowed down by two massive mountain chains surrounding the lost areas.

Design

The main threat for the players' health is the infestation. The infestation manifests itself in clusters on the map. Each cluster emits toxic spores which are highly contagious and can kill humans in seconds through chemical-burning of the lungs and all other exposed tissue.

Infestation areas should contain amounts of mushrooms and spore-nests to be easily identified by the player. Every infestation area contains multiple threat levels around it. Threat levels are categorized in:

Spore Zone
Spore zones have the largest radius. This area contains a small amount of spore particle and has a minor impact on the players' health. The character should start coughing similar to being exposed to smoke from a camp fire.

Toxic Zone
The toxic zone is a small zone tight around the spore-nests which contains a high amount of spores in the air. The damage to the player is high and should force him to immediately leave the zone. The character should cough heavily and additional camera effects should make it clear that death is near.

Dead Zone
The dead zone is the zone next to big clusters of spore-nests and mushroom bodies and will kill the player instantly. Not all areas contain such a dangerous area.
The dead zone can be used as level borders and as additional danger at the bottom of jump spots.

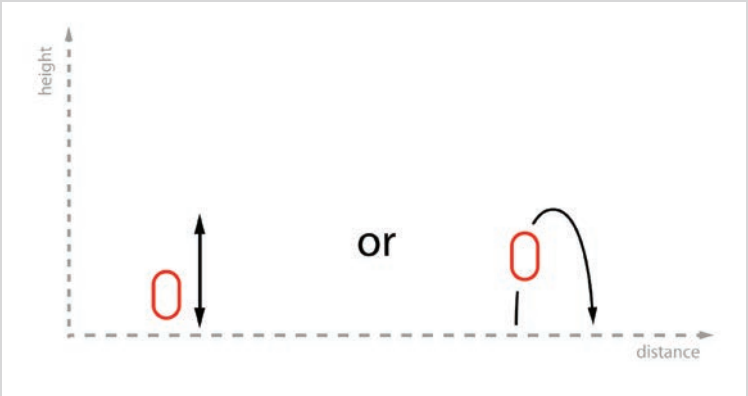
Infestation appearance in game
Infestations manifests itself in forms of big ball like bladders (pustules, fleshy sack, abscess) covered by net-like veins and small spore particles hovering around them. They appear mostly in wind protected areas and can grow on nearly every surface and in every angle and some bladders (pustules, feshy sack, abscess) can even hang from the ceiling. They glow slightly and are very brightly coloured so they are easily identifiable.

Jump Design

Jump Story
Jumping in G.R.E.C serves two purposes. First with the Oculus Rift jumping is a very straight forward thrill, this is due to the stereoscopic and the screen size which creates an intense feeling for height. In order to get the maximum effect we gave the character pyro dynamics boots which enables him to jump up to 10m.

Three different jump types are possible in G.R.E.C.

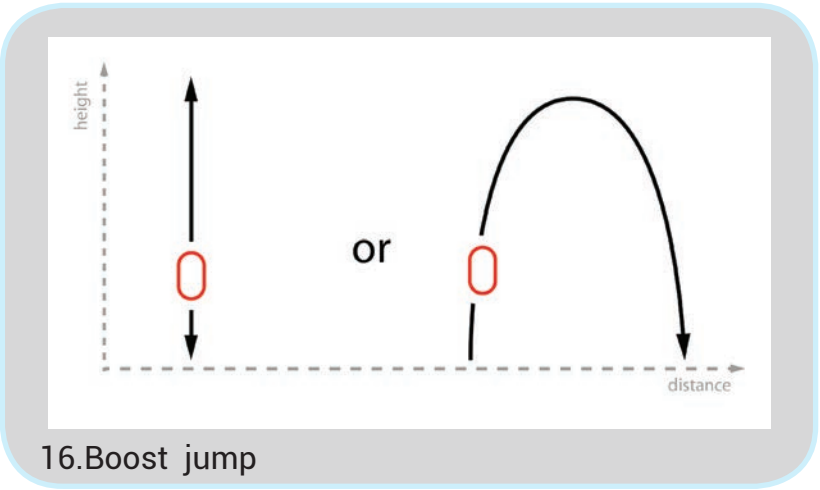
Jump
Represents a normal jump, still boosted by the characters jump-boots but basically with a slightly higher ballistic curve. The player is able to control and change the direction during the jump. This normal jump should be the most flexible and most frequently used jump.



15.Jump

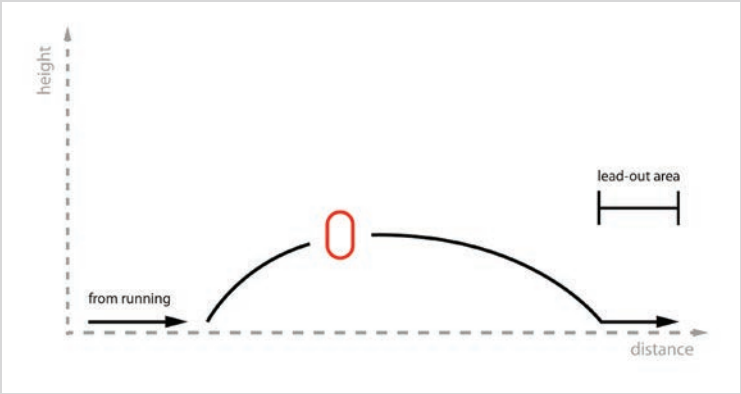
Boost jump
The boost jump represents the fully unleashed power of the jumping-boots. After a short loading time, which is represented by a pressure sound and a crouching position of the character, the player will be able to jump multiple stories straight upwards. He can slightly control his position to make it possible to jump on buildings.

It is still possible to move on the ground if the boost jump is loaded.



16.Boost jump

Long jump
Can be used during running. The character uses his momentum and jumps a far distance straight forward. The player cannot change the direction while using this form of jump.



17.Long jump

All jumps use Breathing Gas as power source and as a result deduct a certain amount of gas each time used.

Level Design - Map Comments

18.Leveldesign Overview Map

The green lighted areas are special little locations that bear a secret and are more detailed than others.

1. A house which is barricaded, we wanted to imply that some residents didn't left voluntarily.
2. The train workshop with the steam locomotive. The train was left behind because of technical difficulties.
3. An infested waterfall and our only location with unique lightning.

The red area is the dead zone and the level boarder for the player. He shall not pass the red zone!

The blue marked houses are those with planed quest loot.

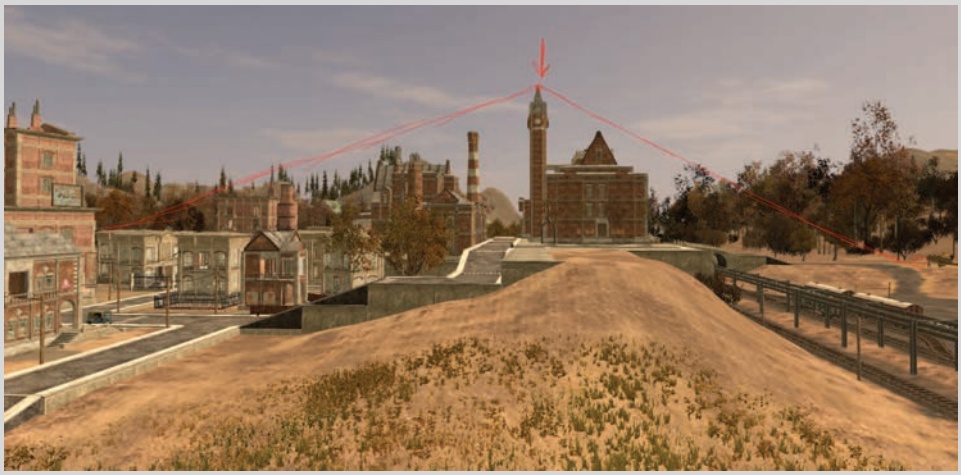
A. The city hall and archive in the attic was supposed to be a storage for city artifacts.

B. The village General store, including one of the quest loots. A Bookmaker's bock, apparently they didn't sell just groceries and tobacco.

C. A big city house with multiple flats and an assembly hall in the first floor. The player will find an old painting which is the third quest-loot.

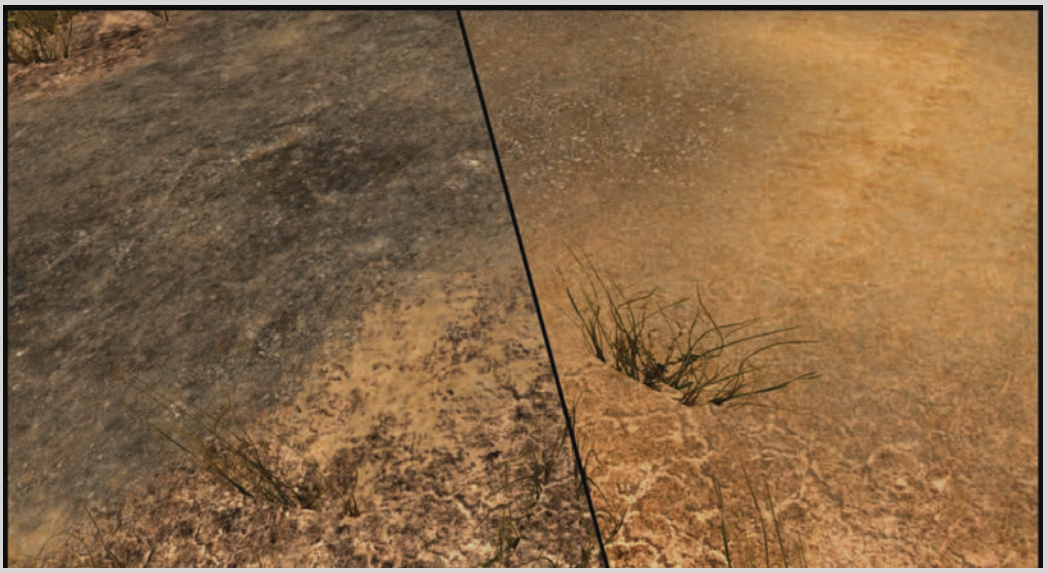
The red line follows the ways the players take when playing the game for the first time. Most of them follow the road and pass the bridge some meter the little house next to the city hall. Other players follow the road to the big city house, and then start exploring.

The trees on the mountains are the level boarders and are scaled down, so the player will see the distant mountains as bigger and further away. This seems like quite a simple trick but it´s still worth mentioning.



19.Leveldesign Silhouette

The city should have a tall center and a pyramid like silhouette. Most central European city's had a church and massive building in its center, later when factories were built around the actual village.
We tried to make a city that looks like it wasn't built in one day. Also the clock tower should serve as the focal points which should help the player with the orientation.



20.Leveldesign RTS Comparison

Here is the map in an early state, no trees have been placed, but the rough terrain colors have been applied.
The terrain shader allows eight different textures which have all functionality (parallax, advanced blending),
so we blended them into each other as often as possible and used the pure textures only on remarkable places like streets, the tracks and stone.
The fields are still missing but the general shape is nearly final.

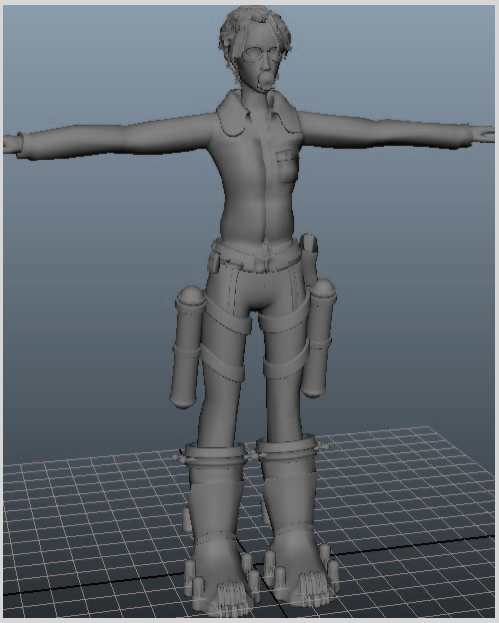
21.Leveldesign Early State

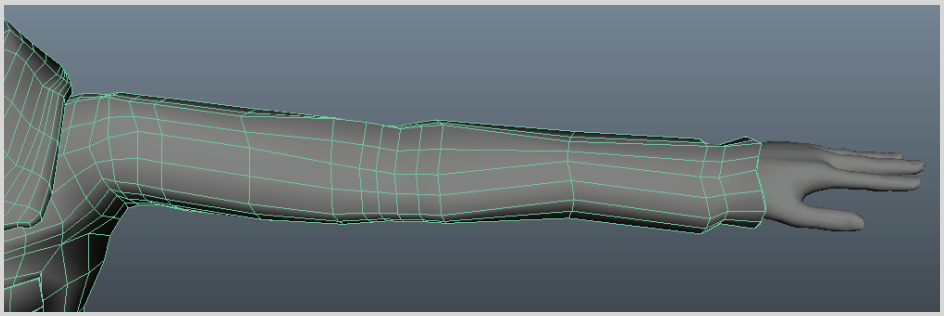
Chapter III

Production

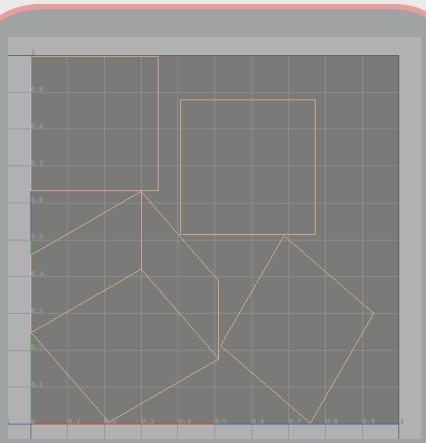
Modeling, UV-Mapping and Texturing

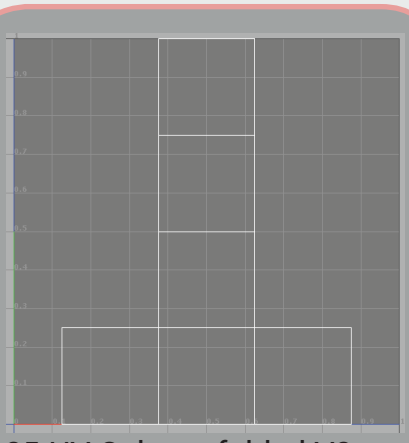
After we made thoughts about the character design it was time to start modeling the character.
Modeling is all about bringing the polygons into the desired shape, adding edges, moving vertices, extruding faces, etc. It's not necessary to build the model in one piece, but having the torso, arms, legs and head in one mesh makes it much easier to weight-paint the mesh, once rigging is completed. Having a nice edge flow on the place where the limbs are, makes the bending look smoother and avoids a clinched look. Generelly you can say, having more edges/ vertices enables the bending to look more natural.


22.Character in T-Pose


23.Edge Flow

For example: A polygon cube in a 3 dimensional space could be unfolded in different ways. The following pictures showing two ways of many, which might be possible:

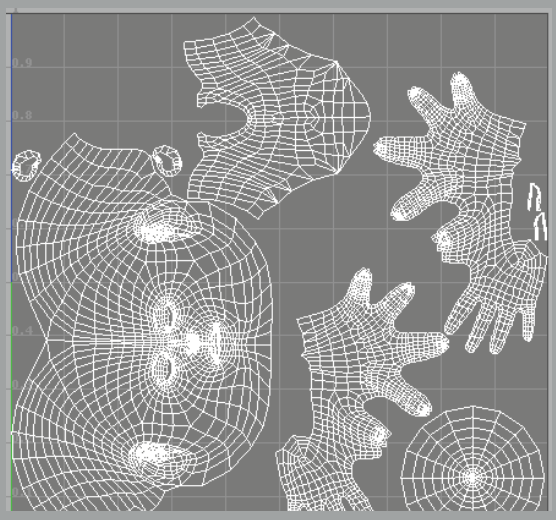

24.UV Cube unfolded V1


25.UV Cube unfolded V2

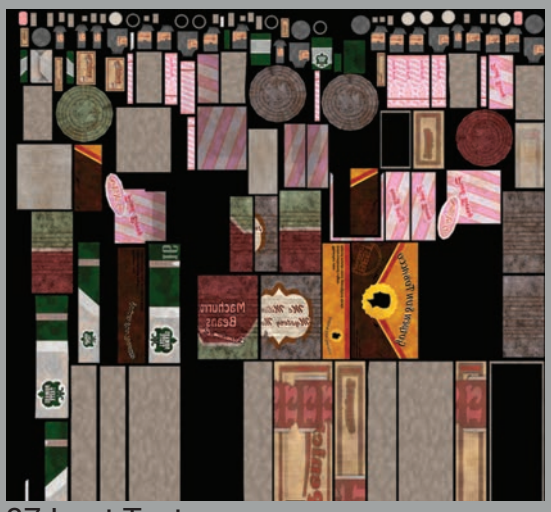
The cube in the 3 dimensional space looks the same, but the UV-Map is completely different. Imagine you have to create a texture for this cube, so on the first UV-map it's understandably difficult, nearly impossible to create a not distorted texture for it.

After UV-Mapping it was time to create textures. This was done with Photoshop, but Gimp or another raster graphics editor could also manage this process.

UV-Maps can be exported in different resolutions but they are always quadratic. It is wise, to try to fit (scaling the UVs) the whole quadratic area, so you can take most out of it.
Remember: We are working with limited resources and it is better to use one Texture-Atlas for many objects parts rather than having multiple files in your projects.
This applies especially to mobile-games, everything that is needed should be saved on one file, in order to save space. This is called resource-saving. We are also doing this for the Loot-able objects, they are all referring to one texture file.



26.UV-Map



27.Loot Texture

Rigging the Character

Rigging means the mesh will be attached to a joint system. This process must be completed, before the character can be animated.
Joints can be compared with the bones of a skeleton, so a joint system of a human character is nearly the same to the bone structure of a human body. The complexity of a rig can be quite different, but mandatory joints are those few for the legs, the arms, the spine and one for the head. These “simple” rigs will work fine, however do result in the animations you can perform being limited.
We kept in mind which requirements were needed for the character before we started rigging. This is fundamental, so the joint system was adapted to those specific needs.

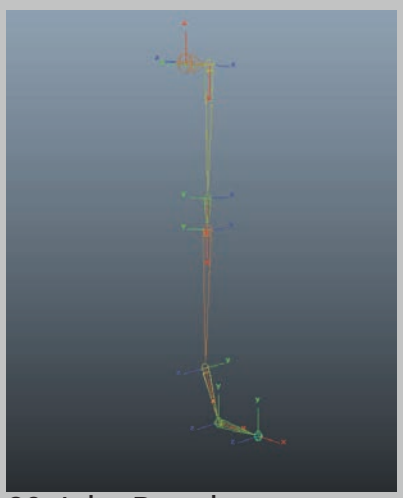
It was important to know that there are some differences between gaming rigs and rigs for 3D movies. Gaming rigs (especially rigs imported in unity with the mechanism animation type) have some restrictions. It is not possible to use cluster or deformers for the animations, or a rig with some parts of it connected with constraints. Only joint based animation is supported.

How to rig a character

Rigging is more a technical process then a creative one. The goal is to build a joint system according to the requirements of the character. Rigging is not a build in process in unity, it has to be done in a 3D program like Maya, Cinema 4D, 3ds Max, Blender, etc. In our case we used Maya for booth, modeling and rigging.


28.Rigging Start

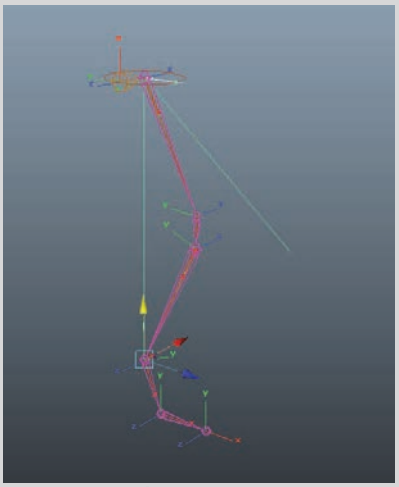

29.Rigging The Leg


30.Joint Rotations

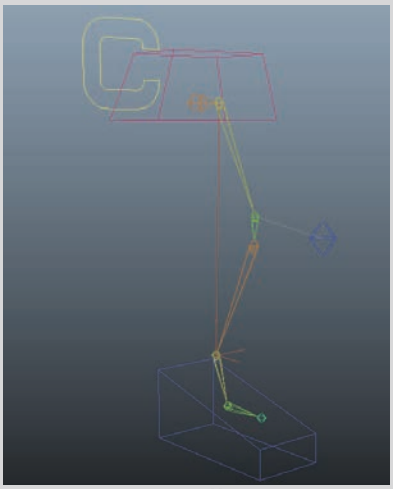
The following steps were done, to rig the entire character:
- We placed a root bone in the center of the hips. All other joints were connected to this root joint
- We placed the other joints inside the mesh. For example both legs contain joints for the thigh, knee, ankle, ball, toes and shin (not particular needed, however nice to have in order to achieve smooth bending). Also joints for the spine, shoulders, arms, wrists, fingers, head, mouth, eyes, etc. were built
- These joints were connected, but the root bone was kept as the “master bone”, to have a top down hierarchy. We had to be careful with the joint´s orientation of

each joint, because this determines the way and order how the joint chain rotates after parenting them

- For the legs and arms, we create IK-handles to move and follow the position of the attached bones. With a Pole Vector Constraint it was possible to change the bending angle
- We created control curves and constrained them to the desired joints, in order to make the joints easier to control while animating them, we locked the translations which were not needed. This also minimizes the risk of touching and manipulating them accidentally
- After coloring the control curves, to make their visual appearance more clear while animating (the left side was colored blue and the right side red. Unique curves were colored yellow) the rig was bound to the mesh. This assigned the vertices of the mesh to the bones
- The last step was to paint the vertices, of the mesh, to the right joints. At this process we arranged the weight of 1 to the needed joints



31.Leg Jointsystem



32.Leg Controller

33.Final Rig

Finally the playable character had a rig which perfectly fitted our needs. We could now move and rotate the legs, knees, arms, shoulders, wrists and the fingers. In addition the neck, head, jaw, eyes and an eye blink could be manipulated as well as the upper and lower body were now able to rotate.

Character animations

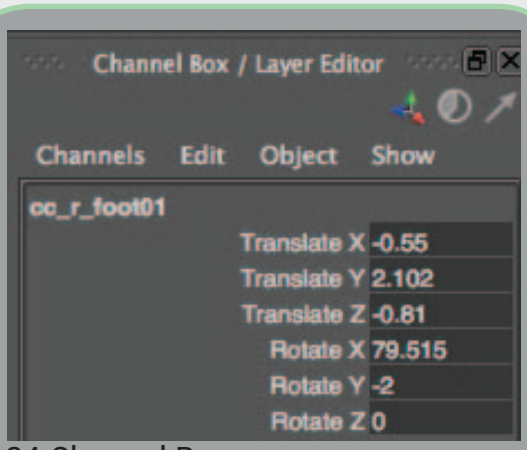
After rigging, binding and skinning the mesh to the character we began the animation process. All character animations were done in Maya and every character movement and action required a unique type of animation. The list of needed animations were defined early in the game design process. The whole list of needed animations is seen below:

Walk animations	Walk, Walk_Bkwds, StrafeL, StrafeR, TurnL, TurnR
Run animations	RunFwd
Boost animations	Boostwalk, Boosting, BoostLanding, BoostLandingFinish, Boost_TurnL, Boost_TurnR
Jump animations	Jumping, Falling
Idle animations	Idle, Lootgrab, ReadingGasUp, ReadingGasStay, ReadingGasDown, StartCough, Coughing, Dying
kicking animations	Kick
Gas animations Up	Bracelet_ 1-10, Bracelet_10-20, Bracelet_20-30, Bracelet_30-40, Bracelet_40-50, Bracelet_50-60, Bracelet_60-70, Bracelet_70-80, Bracelet_80-90, Bracelet_90-100
Gas animations Down	Bracelet_100-90, Bracelet_90-80, Bracelet_80-70, Bracelet_70-60, Bracelet_60-50, Bracelet_50-40, Bracelet_40-30, Bracelet_30-20, Bracelet_20-10, Bracelet_10-1

Animations like, Walk, Walk_Bkwds, StrafeL, StrafeR, TurnL, TurnR, Run, Boostwalk, Boost_TurnL, Boost_TurnR needed to be looped.
When looping an animation, the first and the last key frame has to be the same. After importing the animations in Unity, the frame range were adjusted one frame less to get a smooth cycle.
For example: The run animation goes from frame 1 to 33 in Maya, but in Unity it takes from 1 to 32 to avoid a little hold of one frame.

Animating in Maya

Animating is all about setting keyframes on the controllers, which were created during the rigging process, timing, looking at references. “Channel Box” shows the right foot controller. The Scale transitions and the visibility parameter were locked and hidden, because they were not needed while animating. Keyframes are displayed red in Maya.

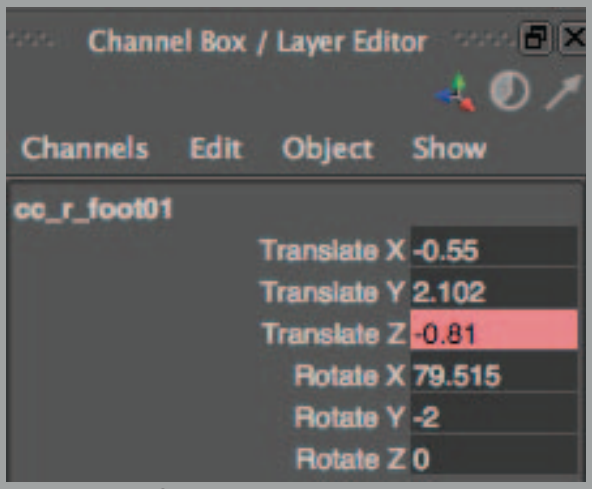


34.Channel Box

The Animating process, roughly described, contained:

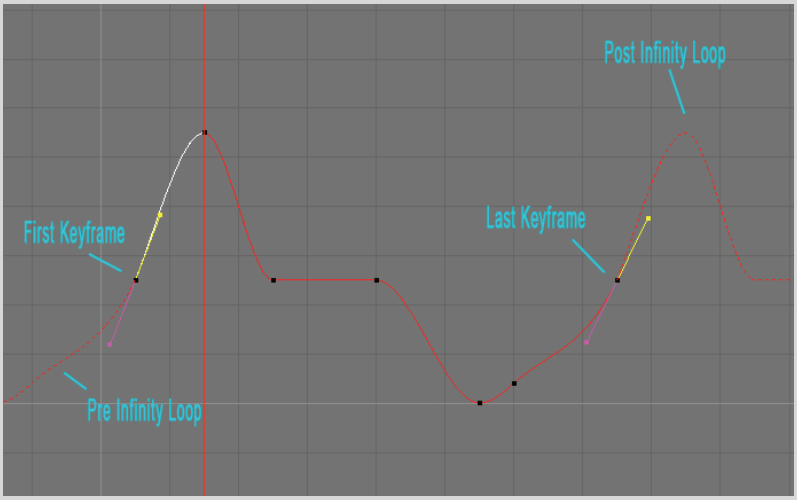
- Creating a start Keyframe (the shortcut “s” keys all channels, right clicking on the desired transition is also possible to key this transition only)
- Moving the controller in the wanted position
- Keying the channel again

Now the controller, which has keyframes on it moves between those keyframes.
As mentioned for looping animations it is important that the first and the last keyframes are the same.
We also took a look at the animation graph in the Graph Editor, because the Graph Editor displays the incoming and outcoming angle of the keyframes. Especially for looped animations it is important to smooth the angle of the first and the last key frame in order to create nice and seamless loops.



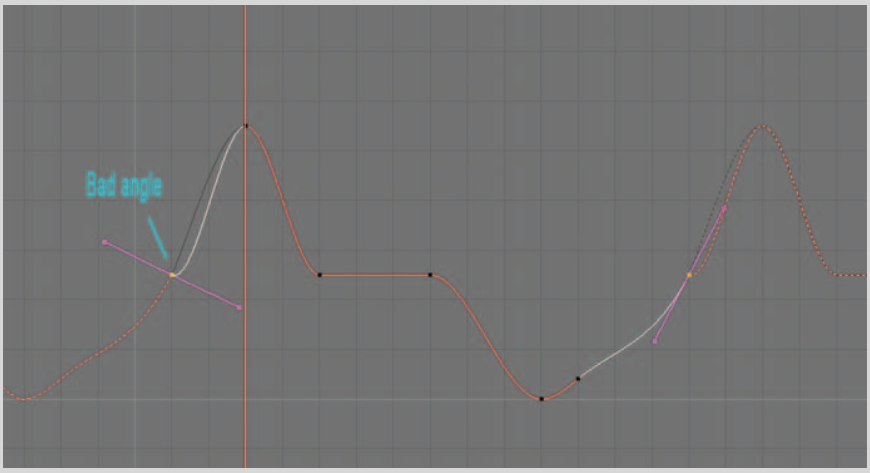
35.Set Keyframe

Now the controller, which has keyframes on it moves between those keyframes.
As mentioned for looping animations it is important that the first and the last keyframes are the same.
We also took a look at the animation graph in the Graph Editor, because the Graph Editor displays the incoming and outcoming angle of the keyframes. Especially for looped animations it was important to smooth the angle of the first and the last keyframe for creating nice and seamless loops.



36.Graph Editor

In “Graph Editor” you see a smooth angle between the last and the first keyframe



37.Graph Editor Bad Angle

In the picture “Graph Editor Bad Angle”, the outgoing angle of the first keyframe is very sharp. This results in an edged animation. It is important to avoid that.

The character was animated at one place. It is not necessary to move the character on any axis (for the walk and run animation), because the game engine will move the character, based on the programming and input real-time.
The whole file was saved in the Maya .mb format, and then imported into Unity.

Unity Animation Import Settings

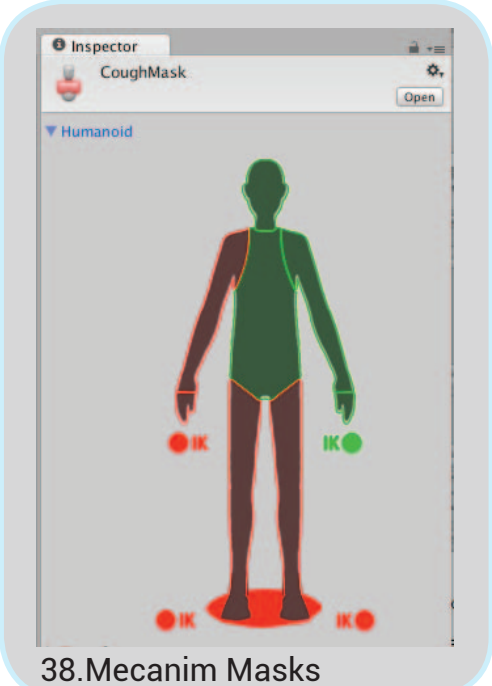
Mecanim is one of three animation types, how Unity interprets imported animations. The other two are Legacy and Generic. The animation type tells unity how to handle animations, which are stored in the Maya files. After we imported the files into our asset working folder we had to choose one of those three.

The Unity documentation describes exactly what the main differences are between Mecanim, Generic and Legacy. But in short form the difference is all about humanoid animations and non-humanoid animations.
For humanoid characters Mecanim is highly recommended. For non-humanoid you can choose between Generic and Legacy. Generic has almost the same features as Mecanim, apart from the avatar system, masks and portable animations, but it also supports animator states. Legacy can be used for simple animations, for example we used it for the breathing gas animation, which displays the actual value of the breathing gas. Legacy animations need a separate “Animation" component.

Why Mecanim

The main reason we used Mecanim for our character is the possibility to use animation masks. Masks allowed us to play animations on the top of other animations without overwriting them.

For example: With masks it is possible to play the “LootGrab" animation or the “Coughin" animation while walking (the CoughMask is seen on “Mecanim Masks"). The red parts shows the zones which are being ignored by the animation. So the coughing just affects the green parts of the body (left arm, body and head).
Masks can be created under Assets -> Create -> Avatar Mask.



38.Mecanim Masks

Another very nice feature is the Playmaker action “Set Animator Look At” which only works with Mecanim. With the combination of the Oculus Rift it is possible to combine the players head movements with the Character movements. So if the player turns his head around, the character also does.

39.Oculus Integration

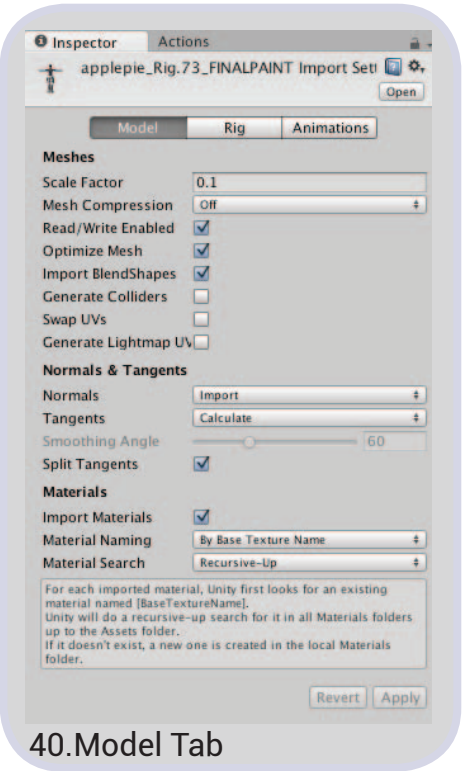
Pro:
- Supports animation masks
- Exchangeable animations with other Mecanim based rigs

Contra:
- Cluster, Deformer, Blendshapes and Constraints are not supported, Joint based animation only with Mecanim
- Human characters only
- Not every animation is possible (facial animation is reduced on eye and jaw movement)

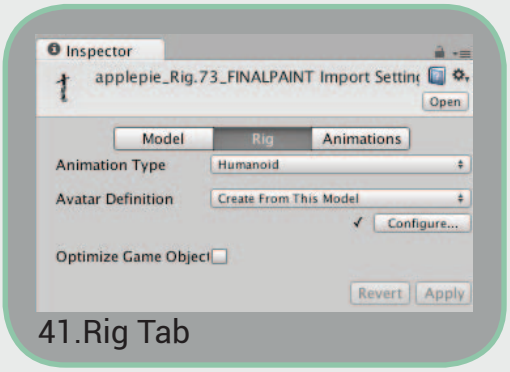
Mecanim Import settings

After we imported the maya files into the asset folder we had to adjust the import settings for each animation file:

Model Tab:
This tab directly affects the imported mesh. The scale factor is especially important, as the model scale in Maya may different to the scale in Unity. This is where we tweaked the meshes scale. It is also possible to scale the imported mesh inside the scene under the transform tab but this changes the scale values (of course). Changing the scale factor does not affect the transform scale values, as they stay at 1, which is the default value.

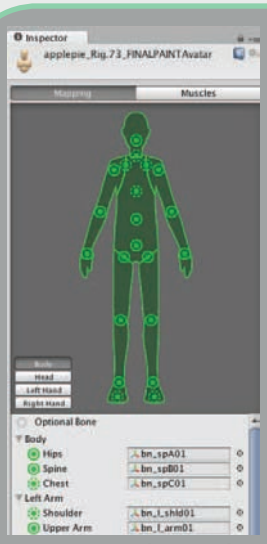


40.Model Tab

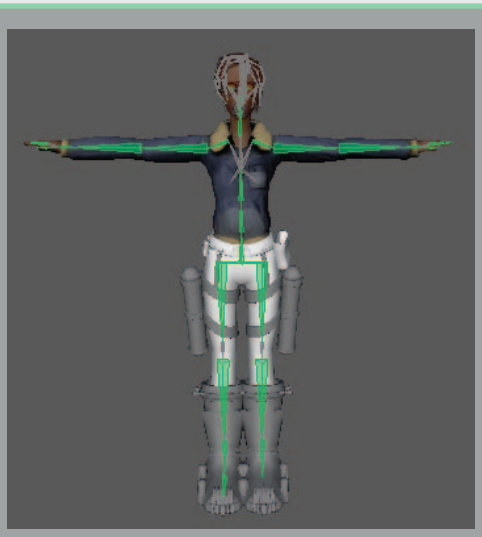


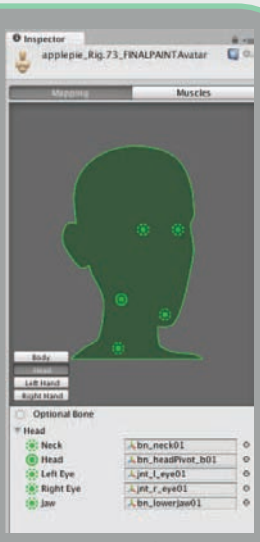
41.Rig Tab

Rig Tab:
The animation type can be chosen here. After selecting “Humanoid” we had to click on the configure button. Now we assigned the models joints to the Mecanim system. The model has to be in a T-pose, which is very important. Clicking on the “Head” button shows one the restrictions of Mecanim. Just the Neck, Head, Left Eye, Right Eye and Jaw can be assigned and will be transformed with Mecanim. These few joints are simply not enough to realize strong facial expressions. We made sure that the Mecanim joints matched exactly to the joints of the Rig, and stands in T-pose, to avoid strange behavior in the animations.



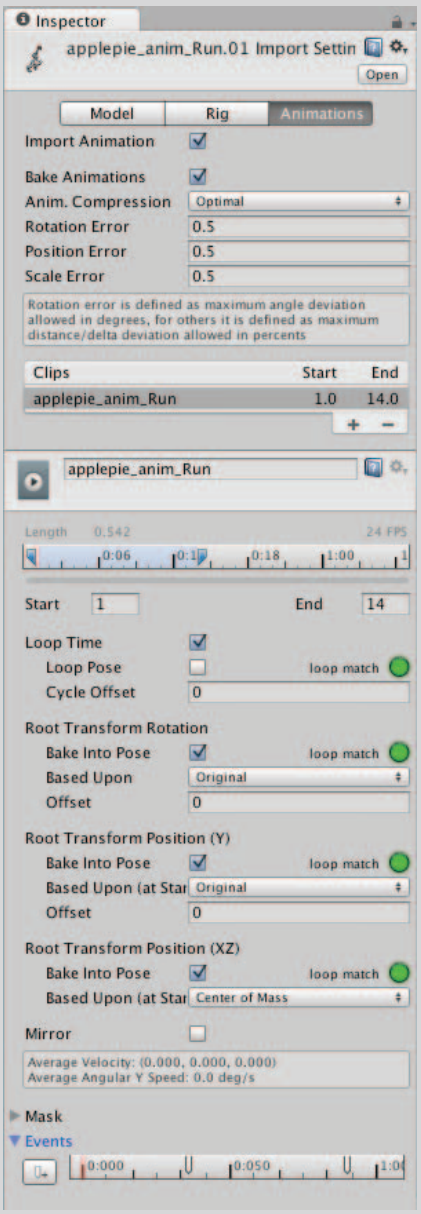
42.Mecanim Joint Assignement


43.Mecanim T-Pose



44.Mecanim Head Bones

Animation Tab:
Importing, adjusting and applying the animations can be done here. First we had to check “Import Animation”. After that it was possible to create animation clips for the animations which are written inside the maya file.
A Maya file can contain multiple animations, so it is also possible to create and adjust the frame rate for multiple animation clips from one file inside Unity.
We activated “Bake Into Pose” on all three points and also made sure “Bake Animations” was checked, to force Unity to keep the animation in the same position and rotation like it was animated in Maya. We also made sure to check “Loop Time” if the animation needed to be looped.
Under “Events” it is possible to trigger custom events with a script. We used this for playing footsteps sounds, for every time the character animation sets down a foot on the ground. Every event can trigger a customizable function (for example left foot function “PlaySound5” and for the right foot “PlaySound6”)
“Footsteps Soundscript” displays a part of the sound scripts and how the action event works. It plays the audio “Rotate” on every event if the booleans inAir, isBoosting and isDying is false. Those booleans are accessed via different Playmaker FSMs. The sound “Rotate” can be assigned inside the Inspector on each field.



45.Animation Tab

```
void playSound2(){
    if(inAir == false && isBoosting == false && isDying == false)
        audio.PlayOneShot (Rotate, 0.3F);
}
```

46.Footsteps Soundscript

Playmaker – Our Little Helper

Playmaker is the reason why this game could be realized without a programmer. Playmaker is a visual scripting tool for Unity and makes it possible to a non programmer to "code" without actual coding in a programming language. It creates **F**inite **S**tate **M**achines (FSM) and organizes behaviors into discrete states. Playmaker uses event-driven FSMs and allows simple actions to be combined to complex behaviors.
Except for the basic movement system and the animation integration all designed features were realized in Playmaker. This contains among other things:

- Normal Loot System / Quest Loot System
- Jump / Run System
- Breathing Gas System
- Dying system (Spores, etc.)
- Particle System
- Coughing System

Another great feature is the possibility to create your own action events. Action events are separate classes which can be easily implemented in Playmaker. The Playmaker community is also very active, you are able to find a lot of self written events in the forums and for our game we used a few of them, like the self written "CharacterMotorAction". With this action it's possible to control the parameters of the Character Motor script inside Playmaker.

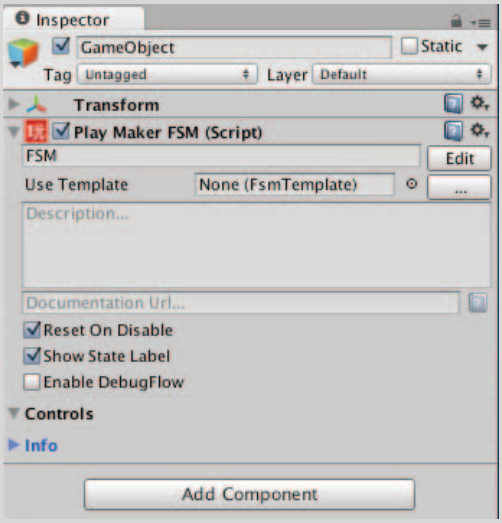
An example of an FSMs, and how they work are described later

How Playmaker works

After importing Playmaker from the Asset store, a new tab with the name "PlayMaker" appears. The PlayMaker Editor stays inside the PlayMaker window. After clicking on it, the editor pops up in a new window. That's the workplace where the magic happens.

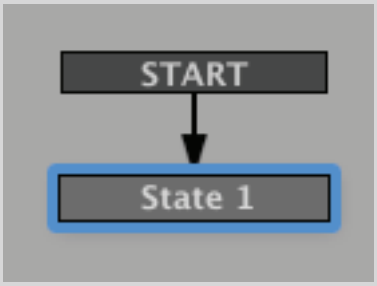
The editor window contains four tabs. FSM, State, Events and Variables. The important ones are the last three. The FSM tab just contains settings like changing the FSM name.

It is important to select a GameObject to add a FSM on it. Now, in the PlayMaker Editor it is possible to right click and select "Add FSM". After doing that, a Playmaker FSM / Script appears in the Inspector.



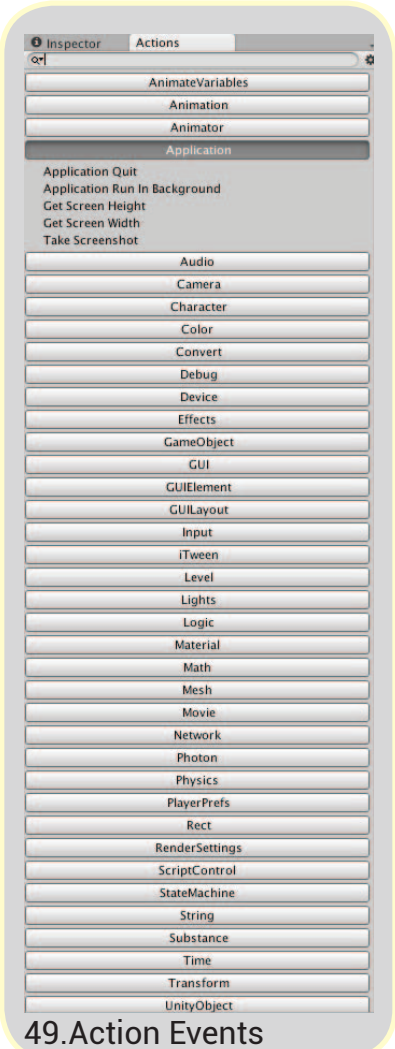
47.Playmaker Inspector

In the Playmaker Editor window there is a a start transition “START” which points directly to to the state “State 1”. START is a global transition, once the game has been started, the state it points on will be active.



48.Start State

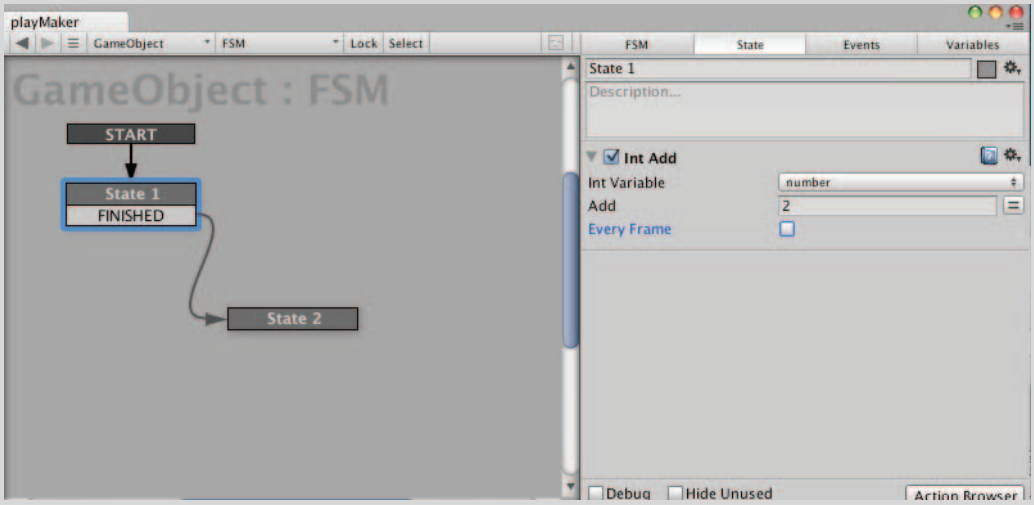
Every state can hold an infinite amount of action events. These action events can be compared with classes. For example the “Application Quit” event does the same as “Application.Quit” in Java or C# and looking deeper into the “Application Quit” script Playmaker provided it will be clear that playmaker has done the coding for you and you're basically assigning script classes together inside the editor. That's the nice thing about Playmaker: You're coding without actually needing to write anything which allows you to focus completely on the function and not on the syntax.



49.Action Events

States

States can hold different kinds of Events. With Events it's possible to switch into the next state after all Action Events are processed. An example can be seen in the next picture.



50.Playmaker States

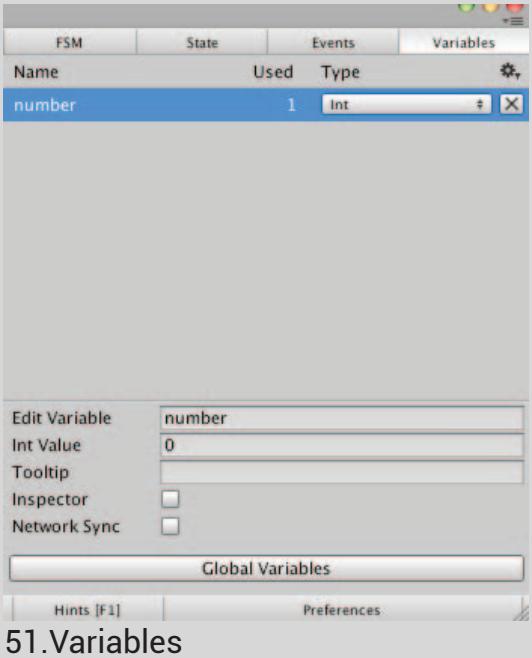
– After the game starts, State 1 is active
– The Action Event "Int Add" adds a number to a variable. After this is done the FINISHED event will be active and Playmaker switches into State 2, which could contain other Action Events

Custom Events can also be created, but Playmaker provides a long list of Global-, System- and Network events.

Variables

The "Int Add" Action needs an Int variable. Variables can be created in the Variables tab.

All kinds of variables are possible. Float, Int, Bool, String, Vector2, Vector3, etc. … also some game engine typical variables can be created: Color, Rect, Material, Texture, GameObject, Quaternion or Object are possible.
Variables can only be used in the FSM which they were created, except global variables. A global variable can be used for all FSMs, even if they are not staying on the same game object.



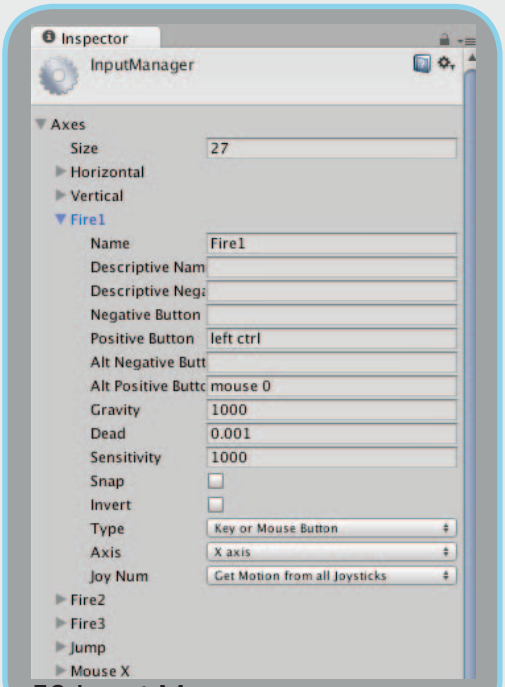
51.Variables

Unity Input Manager

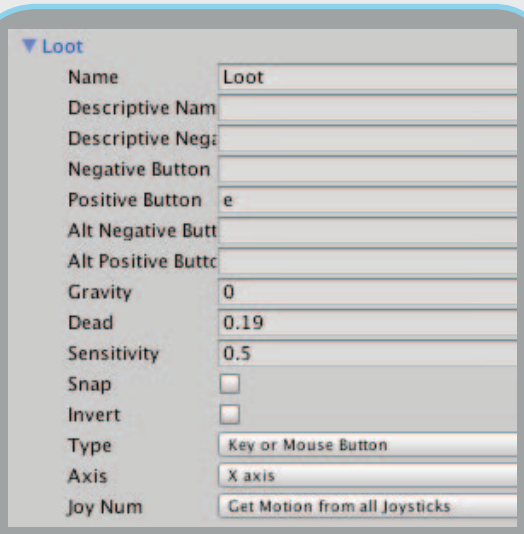
With the help of Unitys' Input Manager we defined all the different input axes and game actions for the project. It can be opened over Edit -> Project Settings -> Input.

While assigning inputs (either in C#, Java or in Playmaker) it is important to understand how to use it.
Infinite axes in the Input Manager can be designed. The most significant fields for each axis are “Name” (the name of the axis) and “Positive Button” (the button that sends a positive value to the axis).

Let's create a small example, of how to use the Input Manager:



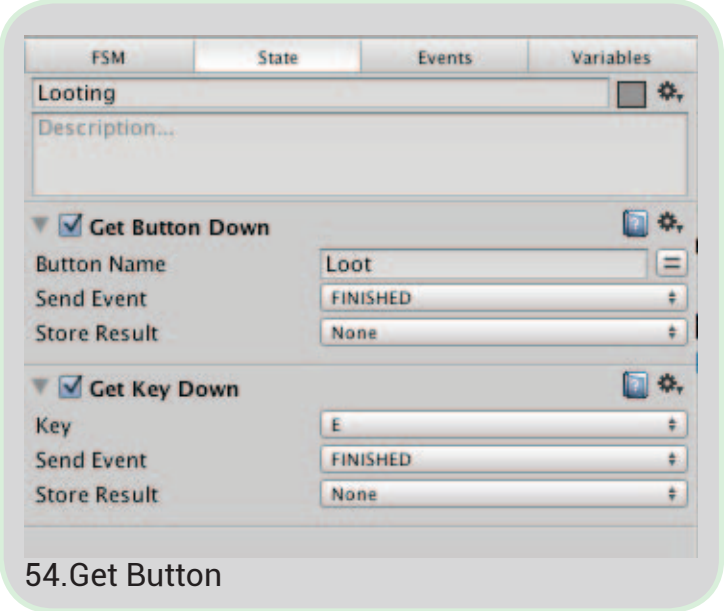
52.Input Manager



53.Loot Button

All looting objects in G.R.E.C have a small trigger area around themselves and we have to enter this area and then look straight to this object to loot it, after pressing the “Loot Button”. But which is the “Loot Button”? The loot button has to be created in the Input Manager. The name is “Loot” and the “Positive Button” is “e”, so the key “e” is the loot button.

It sounds circumstantial to design an individual loot button for the inputs. “Variables” shows another way to use the key “e”. “Get Key Down” does exactly the same as “Get Button Down” but it just reacts to the keyboard inputs. It can't be used for playing the game with a gamepad. This is the main feature of “Get Button”[8]. It's possible to design multiple inputs for one button. In our game The loot button has three assignements: One for the Keyboard, one for the gamepad used in windows and one for the gamepad used in Mac (the Xbox360 button assignments are different for each particular operating system[9]. They have to be assigned with “joystick button XX” in the “Positive Button” field).



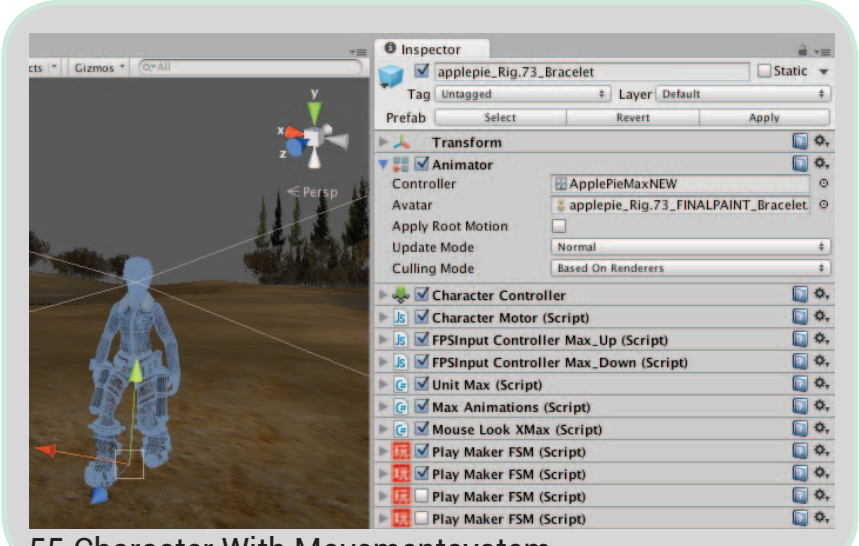
54.Get Button

Creating the Movementsystem

After applying the animations in Unity with the help of Mecanim we started building the animator states and created the scripts for the locomotion system.

This describes our workflow:

– We placed a Maya file into the scene (we used Character in the walking pose, but every other animation file will work well) . All scripts, including the Animator component, the Character Controller etc. were added on this game object



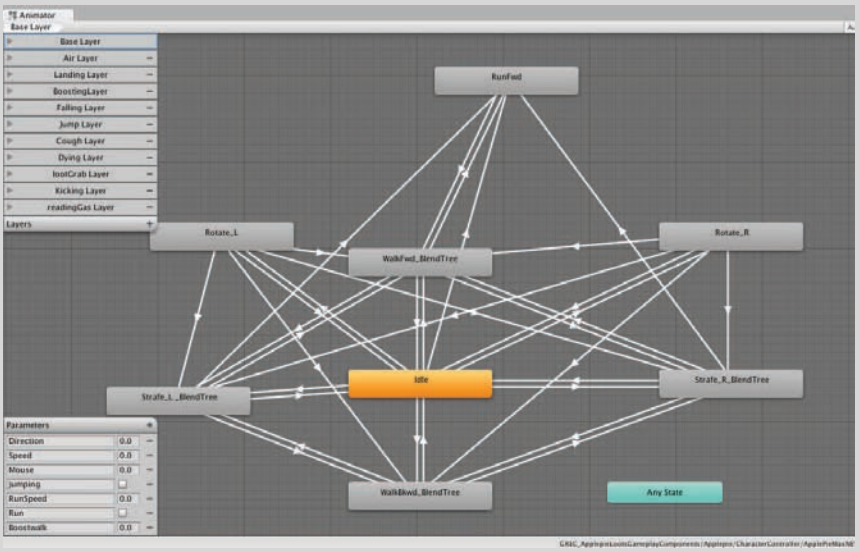
55.Character With Movementsystem

8 http://docs.unity3d.com/Manual/ConventionalGameInput.html
9 http://wiki.unity3d.com/index.php?title=Xbox360Controller

- We created an Animator Controller. The controller contains the animator states and the transitions which connects the states together
- We loaded the animations into the states and tweaked the fading between them.
- We created Parameters and used them for the states to control the conditions. Build Layers and Layer Masks for the different animations
- We connected the Animator Parameters with the control scripts. These scripts build a bridge between the users inputs and the game engine. To avoid confusing scripts we decided to split the scripts into control scripts and animation scripts. Control scripts are responsible for the users input. Animation scripts for playing the correct animation at the right time.

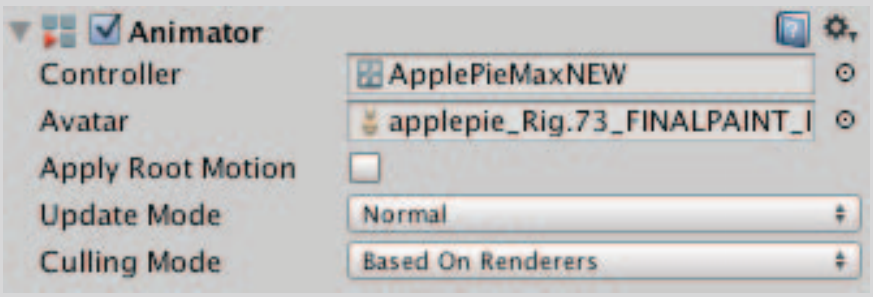
These steps don´t sound very complicated at first glance but after beginning and hooking up the locomotion system, the animator and scripts were getting more complex quickly.

A deeper look inside the Animator



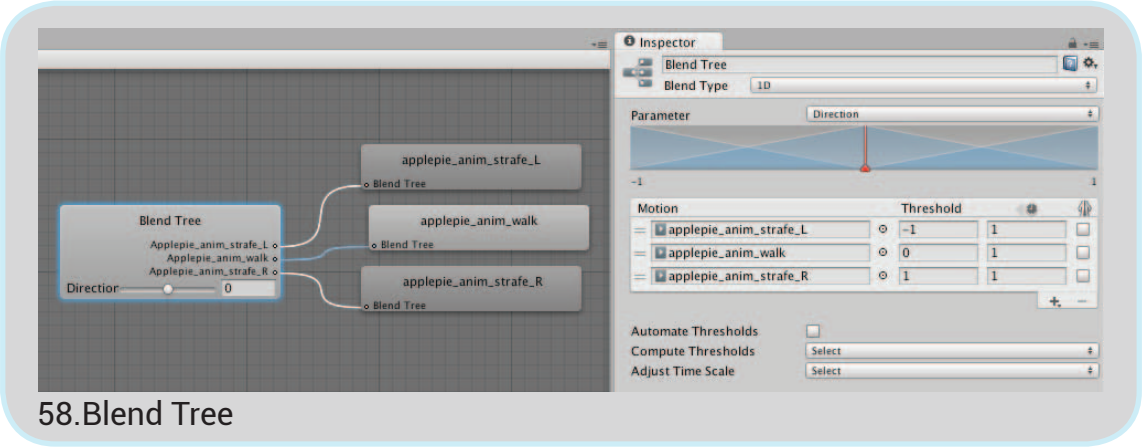
56.Character Animator

The whole animator we used for the character is seen on picture “Character Animator”. The Animator is saved inside a Animator Controller, which has to be created (Assets -> Create -> Animator Controller).
Per drag and drop we put this Controller in the “Controller” field on the Animator Component. The Animator Component was created on our Character object with a click on the “Add Component” button at the bottom of the Inspector.



57.Animator Component

The States Strafe_L, Strafe_R, WalkBkwd and WalkFwd are BlendTrees, each of these include the three animations, which are connected next to them. So for example the WalkBkwd Blend Tree includes the Animations:
Strafe_R, Strafe_L and of course walkBkwds.

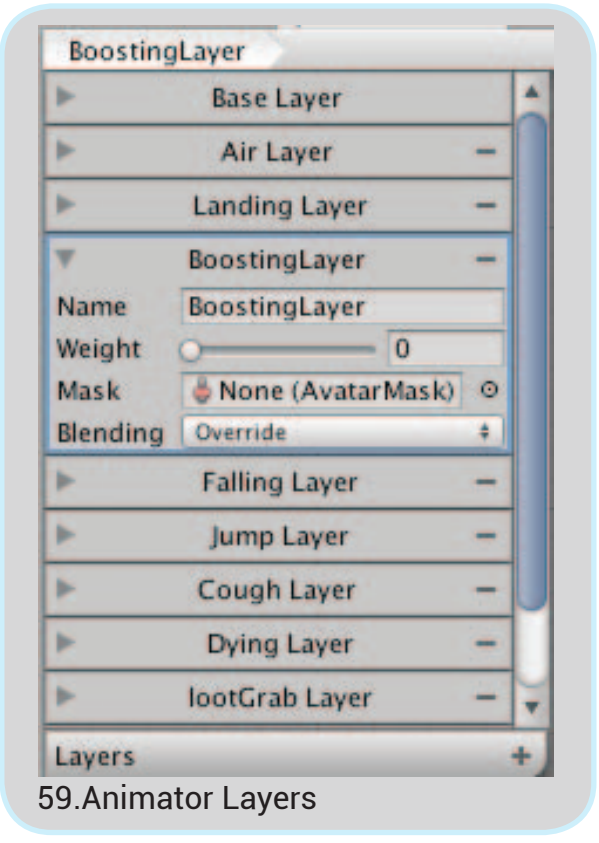


58.Blend Tree

The orange state is the start state. It's important that the animation which is imported into this state is looping, because this state handles the idle animation. An idle animation is important for the character, as it will start playing every time when the player doesn't sends any input commands to the game.

We created multiple layers in one Animator. Layers and layer masks are necessary for creating smooth and plausible animations. How we created layer masks and their use was already mentioned, however not how to implement them.

A layer can be created over the little plus icon. A layer needs a name, but the name is not really important for Unity, because accessing the layer over script or Playermaker works with addressing the layer index. The Base Layer has the index zero, the next layer one, the next layer three and so on. For the G.R.E.C Animation System up to ten layers were created. The layer weight of the base layer is zero and it's not possible to change that (and not needed). However blending the layer weight for all created layers from zero (not active) to one (fully active) is possible.



59.Animator Layers

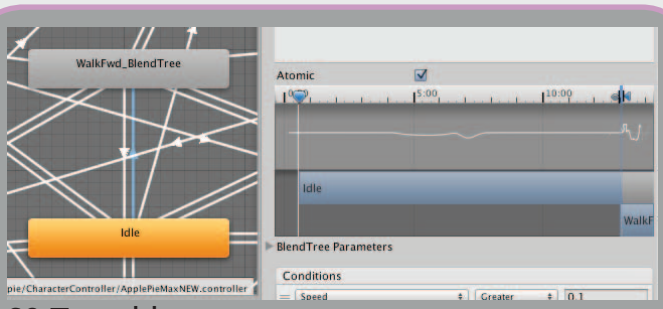
Blend Trees

Inside the Blend Trees the Parameter “Direction” is the driving force. In every Blend Tree “Direction” controls which animation will be played. When looking on picture “Blend Tree” if the value is negative one it blends to the animation Strafe_L. If it's plus one it blends to Strafe_R. If the value is zero, the walk animation will be played.
Notice that if the “Direction" is zero it doesn't mean the character is not moving.

The Blending Trees behavior can be described like this:

- The player presses the key “w” or pushes the left analog stick forward. The parameter “Speed" fades from zero to one, and the character starts walking forward
- The Animator switches to the WalkFwd Blend Tree. The fading speed is adjusted inside the transitions (the white arrows, which connects the states together)
- The Blend Trees parameter “Direction" is not touched yet. It will just get a value if the player presses another button additional, like “a" and “d" (or moving the left analog stick to the left or right) then the blending between walking forward and strafe left or right will start

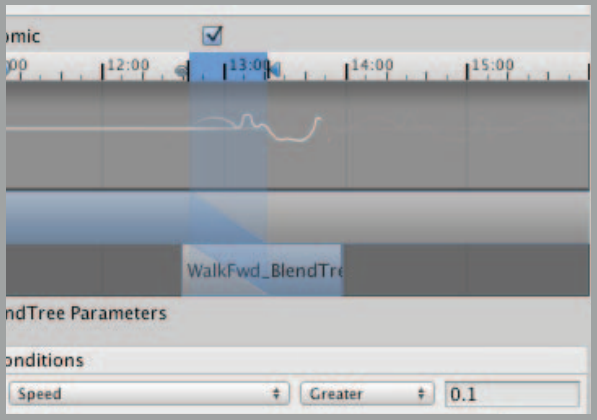
Transitions



60.Transitions

Just creating transitions between two states doesn't tell the animator which state he has to play. Every transition needs at least one condition, at least, which defines when the state starts blending into another one.

The condition can be described as a simple if – else statement. If the value of the parameter “Speed" is greater than 0.1 it changes into the connected state. It's possible to create more conditions for one transition with the little plus icon.



61.Conditions

After the transitions were created between two states, it is possible to adjust the blending time between them. The blending can easily be adjusted on the timeline on the right side by stretching or compressing them.

Animator Parameters

The Animator is controlled by script (or Playmaker), over the Parameters.
Creating a parameter happens by clicking on the little “+" icon and choosing on of the four (Int, Float, Bool, Trigger) possible variables. These parameters control the animator and the parameters and can be addressed by their names in a script.

Speed and Direction
If the character walks forward, the value of Speed is one. For walking backward the value is negative one. Walking left letting the value of Direction go to negative one and walking right positive one. These values are float variables, which makes walking and blending between states nice and smooth.

Direction (inside the Blend Trees)
This is necessary for creating smooth blends between two states next to each other, if the player is already moving. For example pressing “s" and “d" (walking diagonally right backwards), “d" and “w" (walking diagonally right forward), “w" and “a" (walking diagonally left forward) and “a" and “s" (walking diagonally left backward) on the keyboard.

Mouse
This value turns to negative while the player moves the mouse to the left. Moving the mouse to the right increases the value. Not moving the mouse horizontally allows the value to go to zero. The MouseLook script checks inside the update function every frame if the mouse is moving left or right and the animator plays the corresponding animation (rotating left for a negative value, rotating right for a positive value).

Jumping
This boolean will be true during the time the player presses the jump button. This boolean is very important for the sound script in order to check if the character is on the ground which in turn plays the correct sound.

RunSpeed
If the player presses the run button and is already walking forward, the character will start running. To run left, right or backwards is not possible, so a RunDirection parameter was not needed. Running is possible from the idle state and straight from the walk, so transitions from booth states to the running state are needed.

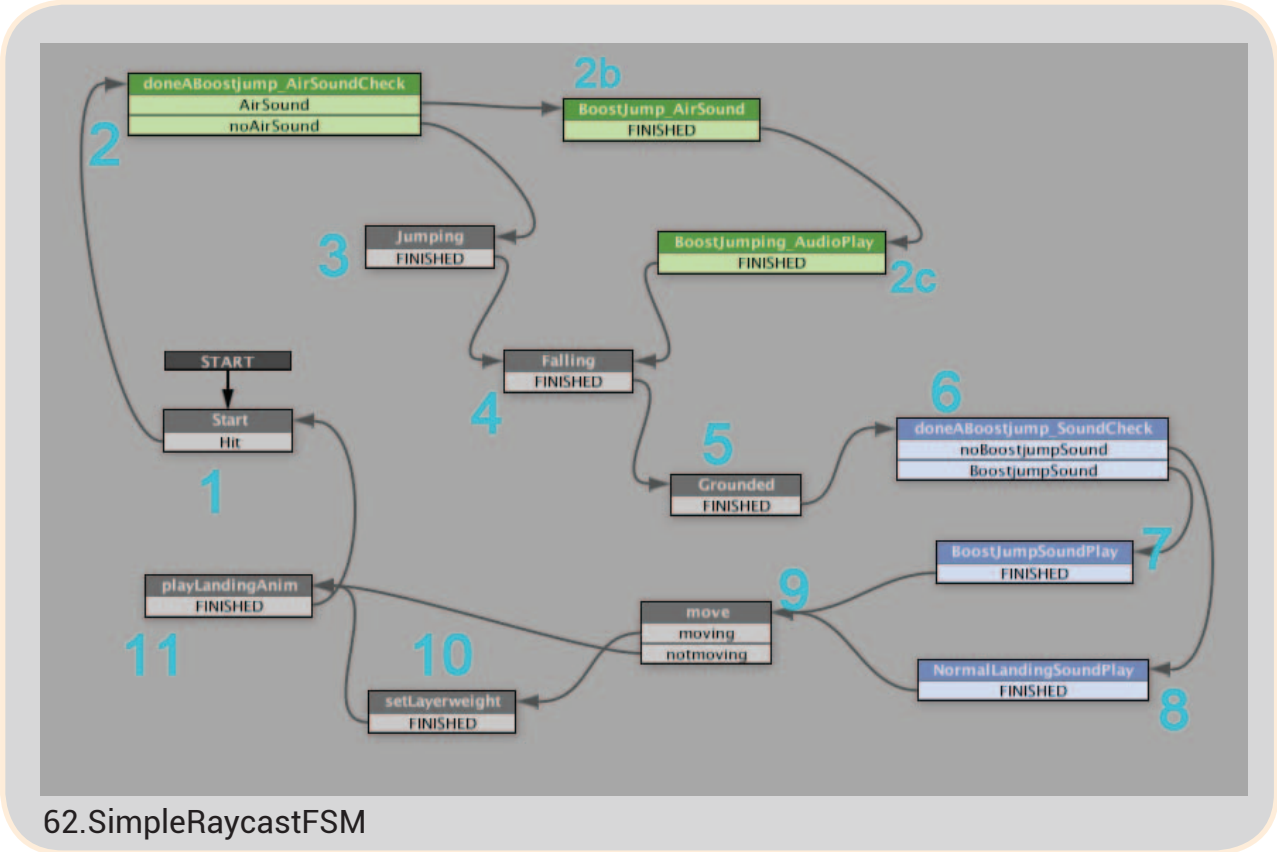
Run
This boolean is needed to check if kicking is allowed. For example: While running around it shouldn't be possible to kick.

Boostwalk
After boosting, the player changes his animation into boost walking. This walk is much slower than the normal walk, the character can just walk forward and is not able to run. While the Boostwalk is active the “Speed" and the “Direction" parameters turn to zero, to avoid playing the normal walk animations.

The Jumping System

The Jumping System is a combination of two big FSMs: SimpleRaycastFSM and ExtendedJumpingFSM. This actual version was not the first one, rather a result of previous jumping prototypes and a lot of try and error.
This is just a rough explanation about what is going on inside those FSMs. Not every Action Event is explained deeply and why every bool was switched.

SimpleRaycastFSM



62.SimpleRaycastFSM

The SimpleRaycastFSM detects when the character jumps, is in the air, is grounded and which jump has been performed (Boost jump or normal jump / run jump). It is also responsible for enabling other FSMs and changing the values of different bools in different FSMs and scripts. The SimpleRaycastFSM is part of the entire jumping system.

- If the game starts, the "Start" state (1) does a raycast down the y-axis with a length of three every frame. If the character is above the ground three units long, the raycast will detect that and switches to the next state (2)

- The State (2) checks if a boost jump or a normal jump was done and either switches to the state (2b) or (3). This check is needed for differencing which sound should play, because the AirSound (wind) should just play if the character is boost jumping. State (2b) and (2c) plays the WindBlowing sound, sets the inAir bool in the sound script to true and plays the BoostJumping sound

- State (3) plays the normal jump sound and also sets the inAir boolean in the sound script to true

- The “Falling” state (4) fades in the layer weight of the “Falling” layer inside the Animator to display the falling animation. It also starts playing the falling animation

- The “Grounded” state (5) detects if the character touches the ground with the help of a raycast. The raycast direction is also negative one on the y- position. The raycast length is one unit and stops directly under the characters feet. This makes the system very solid. The Landing bool in the ExtendedJumpingFS is also being set to true

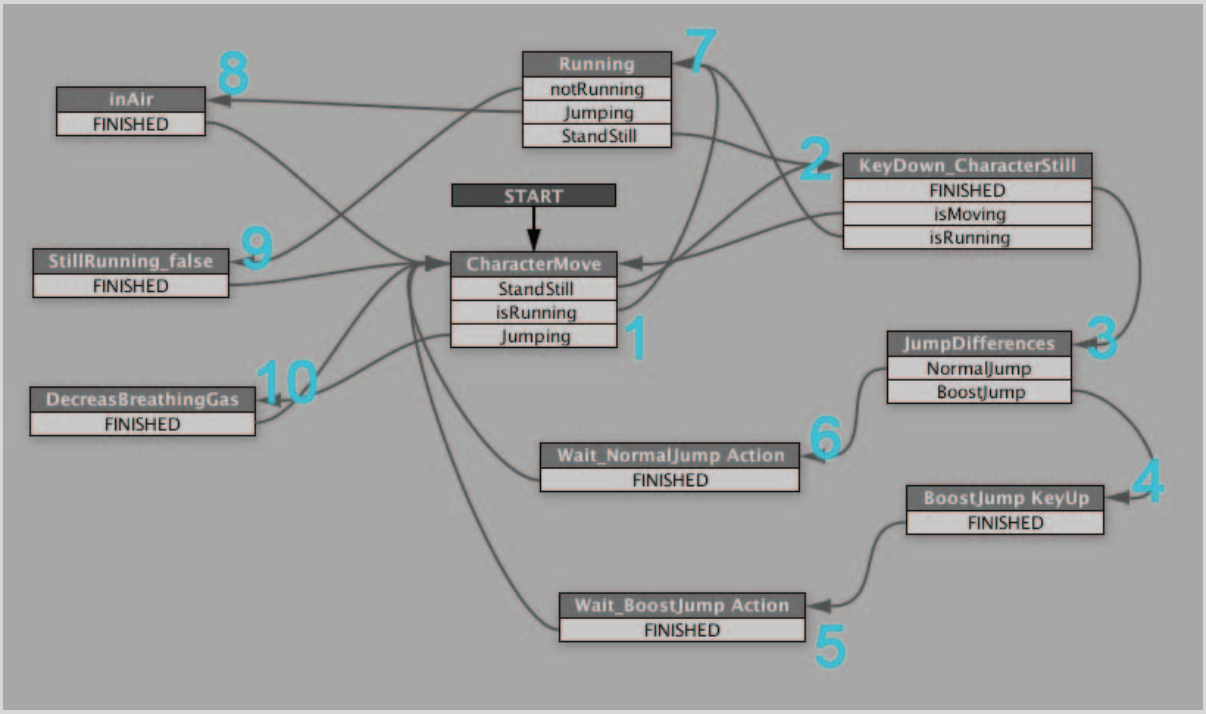
63.Falling Animation

- This state (6) checks which state should be active. The state (7) which plays the boostjump landing sound and sends an Event to a FSM which spawns strong particles. Or the state (8) which plays the normaljump landing sound and sends an event to a FSM which spawns the small particles

- The “move” state (9) sets the isJumping boolean in the UnitMax script to false. This script is responsible for the playing the running animation. While the isJumping boolean is true, it's not possible to run. This makes sense, because running in the air should not be allowed. This state also checks if the character is moving while landing and switches to the according state. Character moves -> State (10), sets the layer weight of the “Landing” layer to 0.5. Character is not moving -> State (11) Playing the Landing animation (with a Layer weight of 1)

- The State (1) is active and it casts again a Raycast to check if the character is jumping and sets different bools to true or false. Landing bool inside ExtendedJumping_FSM = true, the global doneABoostjump bool = false, inAir bool inside the Sound script = false

ExtendedJumpingFSM



64.ExtendedJumpingFSM

The ExtendedJumpingFSM controls values of the Character Motor script and switches between the two FPSInput Controller scripts. It's connected with the SimpleRaycastFSM. It also activates different sounds and decreases the Breathing Gas values on different states

- On the first start and also if the character is moving, the first state "CharacterMove" (1) is active. It changes all necessary values for walking and jumping (forward speed, sideways speed, backwards speed, Gravity, etc..) in the character Motor script, with help of the Character Motor Action. It enables the FPSController_Down script

- If the character is not moving, the state (2) will be active. The main differences between these two states is, it enables the FPSController_Up script and disables the FPSController_Down script

- Based on this state, the next state (3) will be active if the player wants to boost and presses the Jump Button, in order to execute a boost jump

- State (3) increases a float variable over time and if this float variable reaches a certain value, the boost jump is charged and the "BoostJump KeyUp" state (4) will be active. If the Player releases the jump Button before the boost jump is charged (before the float variable reaches it's maximum), state (6) will be active and the character will do a normal jump instead of a boost jump

- Running is possible out of standing and out of walking, so two transitions from State (1) and (2) to State "Running" (7) were built. In this state the character is running

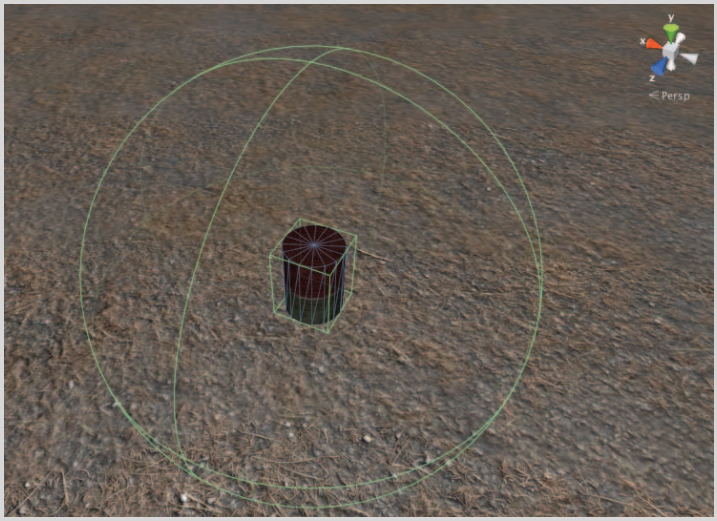
- Jumping during a Run will activate the “inAir” state (8) This state waits for the true status of a landing variable which will be set true in the SimpleRaycastFSM if the character touches the ground. If the character is still running after landing, the FSM switches directly into the state (7) again

- State (10) is needed for decreasing breathing gas, while doing a normal jump, out of the walk

- State (9) changes the boolean value “stillRunning” to false, if the player is running (and is in the “Running” state and releases the run button, to avoid changing into state (7) again (because state (1) checks if this boolean is true and if so it changes directly into state (7))

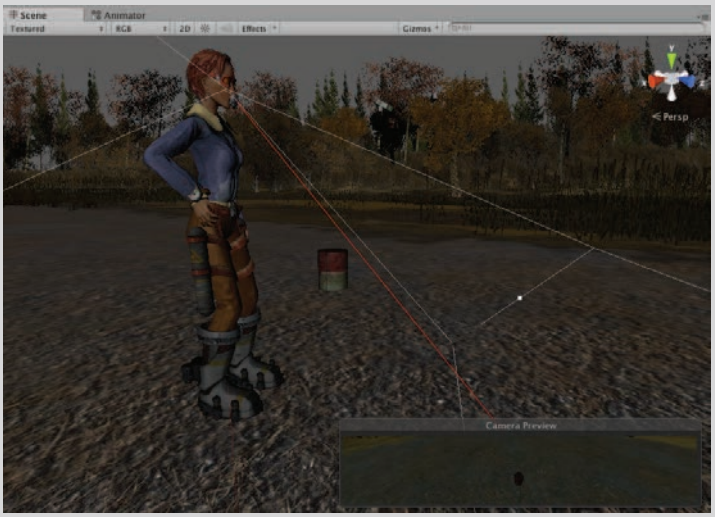
Looting System

Looting is an important feature of G.R.E.C so we needed a solid loot system which fitted our needs. For looting objects the Oculus should be used. We thought it would be nice if the player has to look at the object before he could loot it, instead of just walking next to it and pressing the loot button. We decided to create a FSM which does a raycast from the player's actual camera position to detect loot objects.
After doing several little tests we realized that it was important to create a trigger area around the item to avoid looting over a huge area. Just inside this area (which should have the radius of an arm's length) and looking at the object, looting should be possible. To make it clear to the player that an item can be looted, we decided that we needed a visual response from the system, so the material should change its color.
Finally, the interaction of three FSMs, placed on the camera and the loot object itself, and the communication among them, represents our final Looting system.



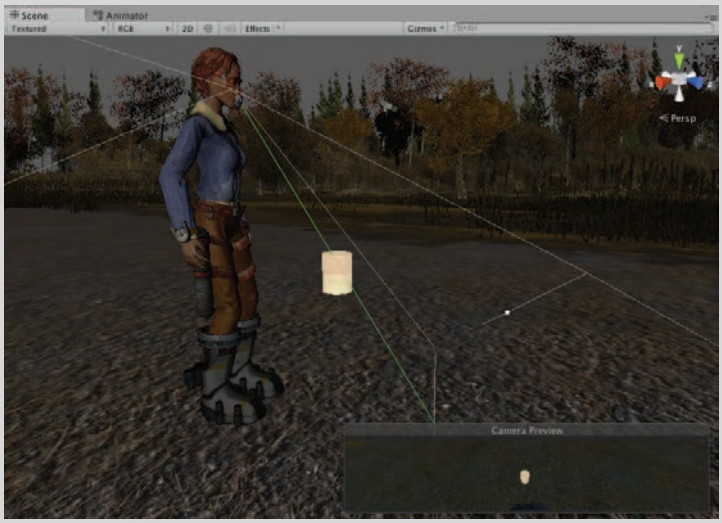
65.Lootobject With Trigger

One of many loot objects, in this case a coffee can. Every Loot object is a game object with a mesh and a shader. It has a collider (in most cases a box collider) and a triggerzone (in most cases a sphere collider).



66.Lootobject Raycast False

The lootable item is in front of the character. The raycast is red, the character doesn't look at the loot object.

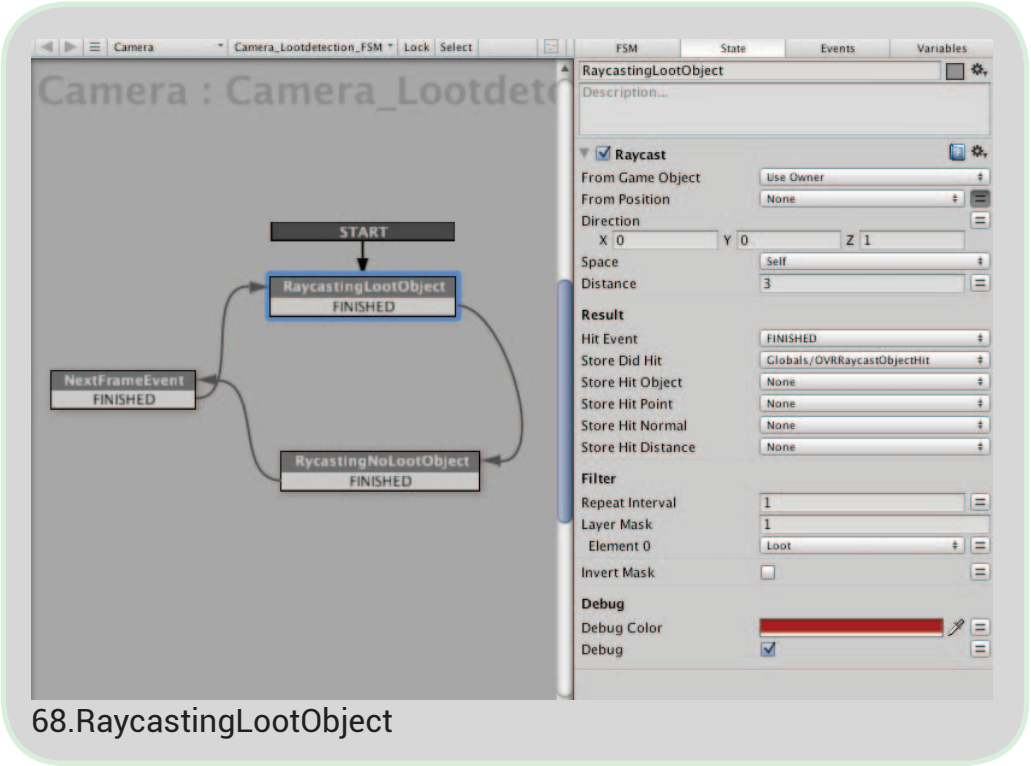


67.Lootobject Raycast True

The player looks at the lootobject and stays inside the triggerzone of the object, so the raycast is green. The item can be looted.

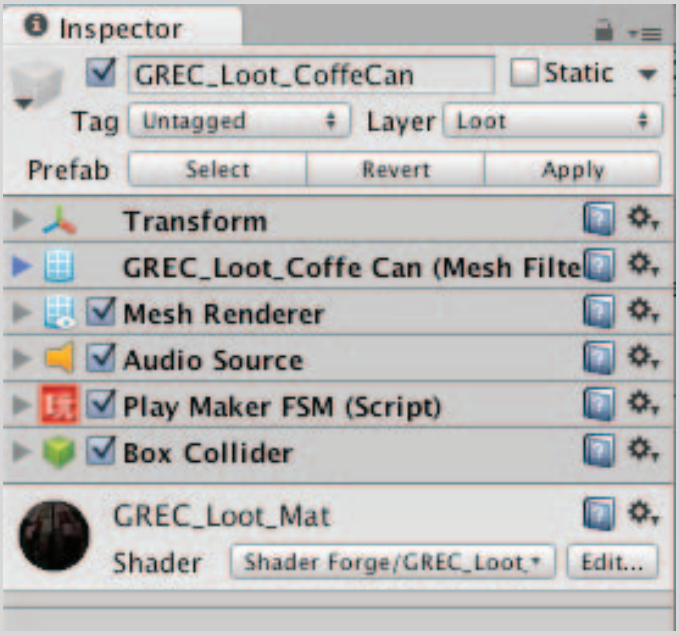
Camera_Lootdetection_FSM

This FSM is responsible for detecting the lootable items. It's placed on the Camera (first person version) or on the right Camera for the Oculus Rift version. It's an infinite loop of states, which is doing a raycast every frame. The raycast is centered in the middle of the camera and follows the camera rotation.

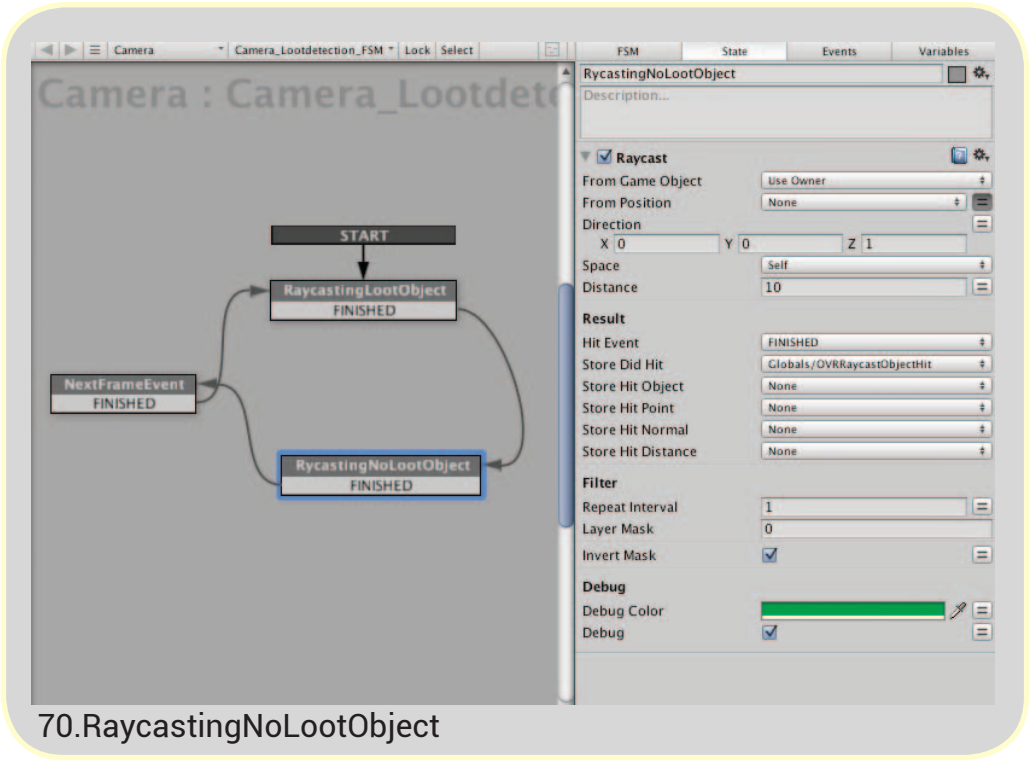


68.RaycastingLootObject

The Action Event "Raycast" placed in the "RaycastingLootObject" state does a raycast with a "Reapeat Interval" of 1. This means the raycast will be repeated every frame. The "Direction" has to be straight forward, so the Z-Axis is one. The Distance is set to 3 units, this approaches the arm length of the character.
In the "Filter" section, the Element 0 is set to "Loot". This is the layer which the raycast detects. Layers can be assigned in the layer field on the top of each game object in the inspector. The layer "Loot" is an individual layer and was created before. If a raycast detects a lootable object, it sets the global bool variable "OVRRaycastObjectHit" to true and changes into the next state.

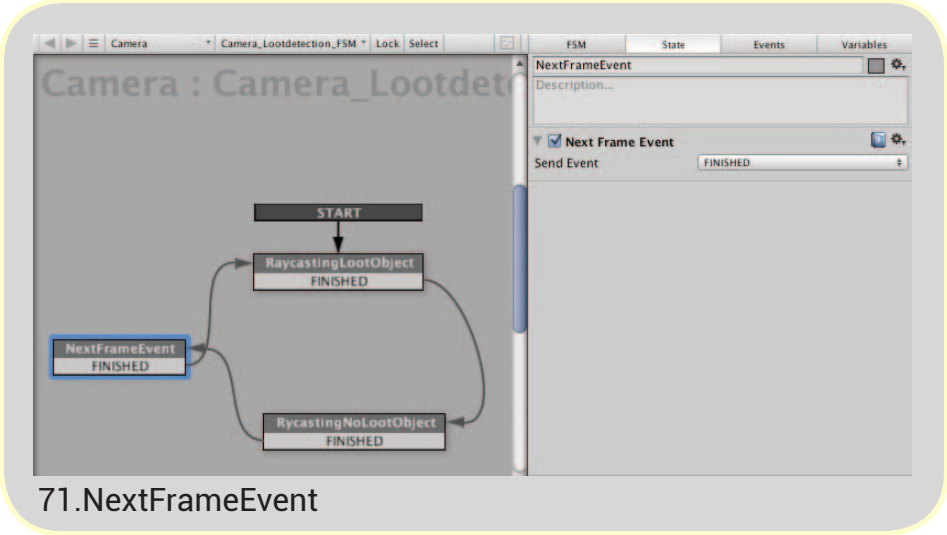


69.Objectlayer Loot



70.RaycastingNoLootObject

The state “RaycastingNoLootObject” checks every frame if the raycast is still hitting an object with the layer “Loot”, so the checkbox “Invert Mask” is checked. If it's not, the global
bool “OVRRacastingObjectHit” is set to false and the next state will be active.



71.NextFrameEvent

The “Next Frame Event” just plays the next state, but with a short delay. This avoids creating a stack overflow, which may occurs with the infinite loop of this state machine.

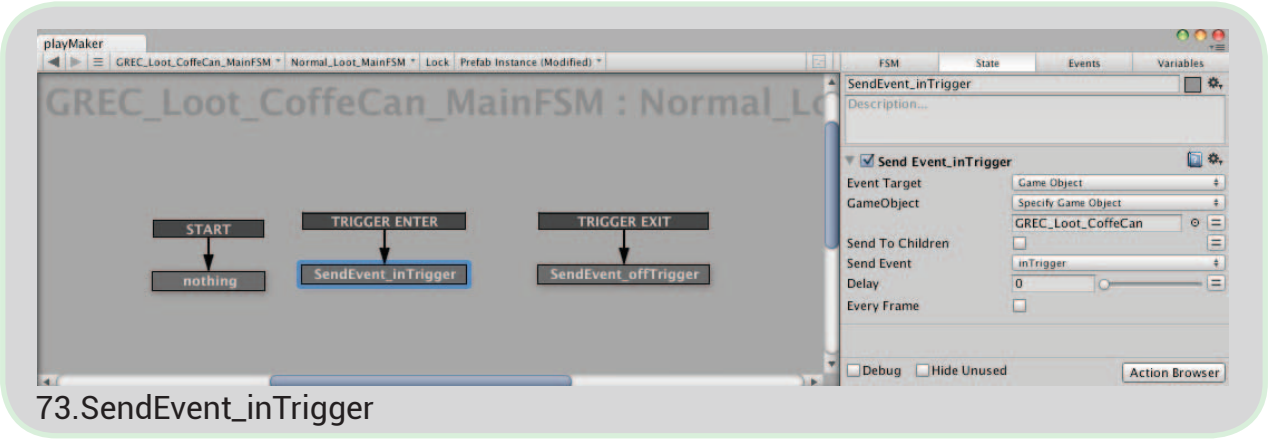
Normal_Loot_MainFSM

This FSM has just one function, sending events to the “Normal_Loot-System_OVR_FSM”, which is placed directly on the lootable object. The “Normal_Loot_MainFSM” is placed on the group, where the loot object is stored.

This group is also the holder of the sphere collider which works as a trigger. If not separating the trigger and the Lootobject, the raycast would also detect the trigger area as a Lootobject.

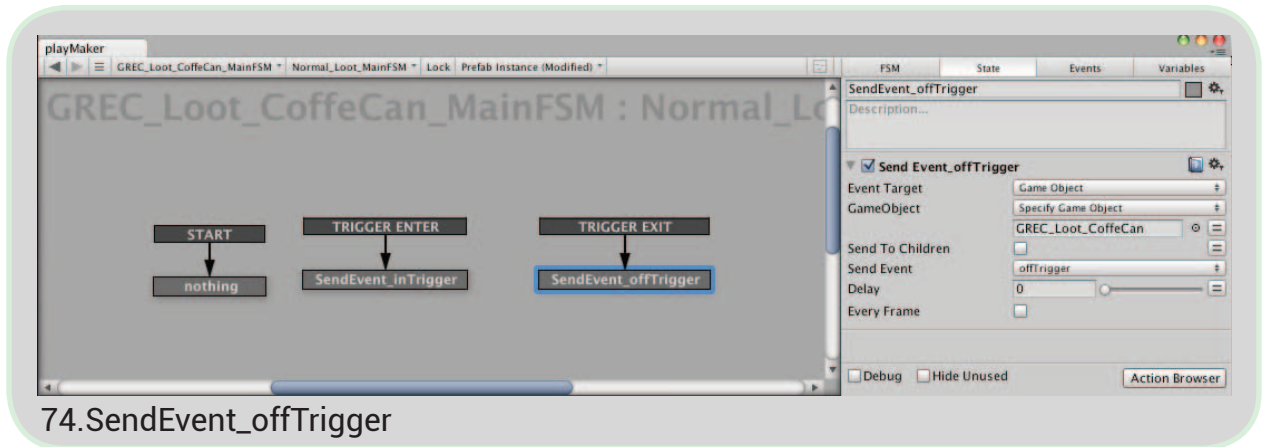
72.LootSystem Inspector

“The GREC_Loot_CoffeCan_MainFSM” game object includes the trigger area and the “Normal_Loot_MainFSM”. The “GREC_Loot_CoffeCan” (the actual lootable object) includes the “Normal_Loot-System_OVR_FSM”.



73.SendEvent_inTrigger

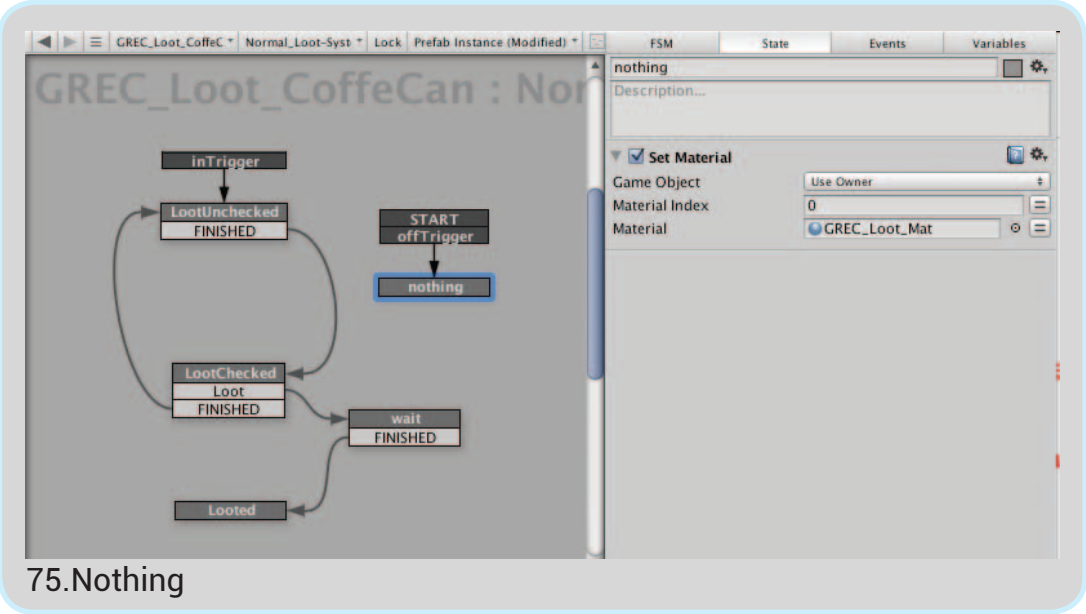
After entering the trigger area, the state “SendEvent_inTrigger” will be active. Its activating the “inTrigger” event inside the “Normal_Loot-System_OVR_FSM”. TRIGGER ENTER works as a global event.



74.SendEvent_offTrigger

The “SendEvent_offTrigger” state will be activated if the character leaves the trigger area. The Event “offTrigger” is send to the “Normal_Loot-System_OVR_FSM”. TRIGGER EXIT also works as a global event.

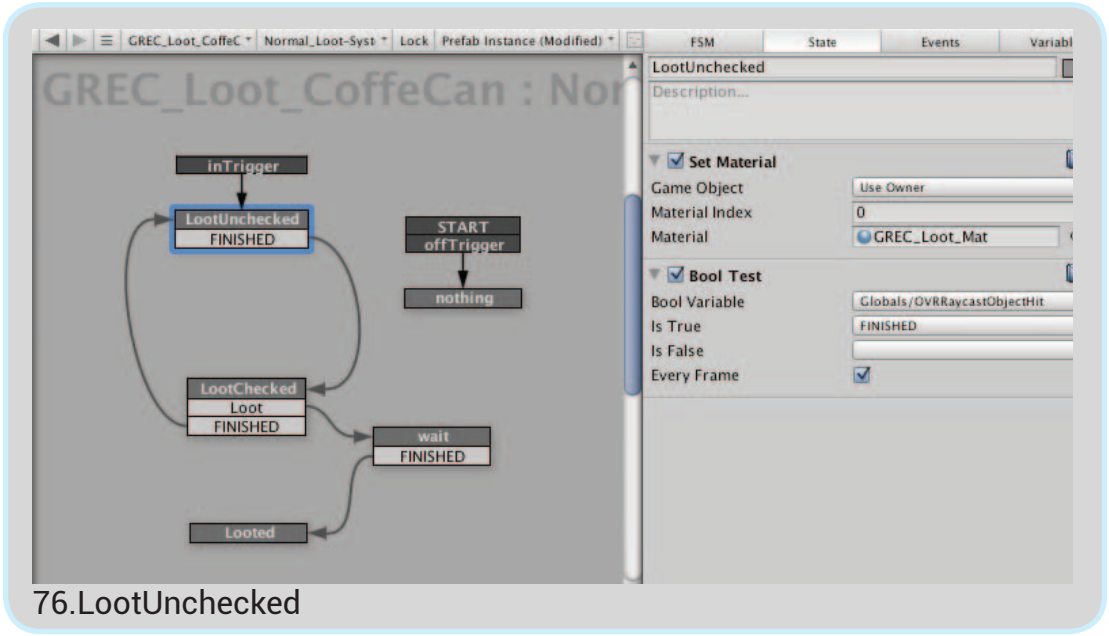
Normal_Loot-System_OVR_FSM



75.Nothing

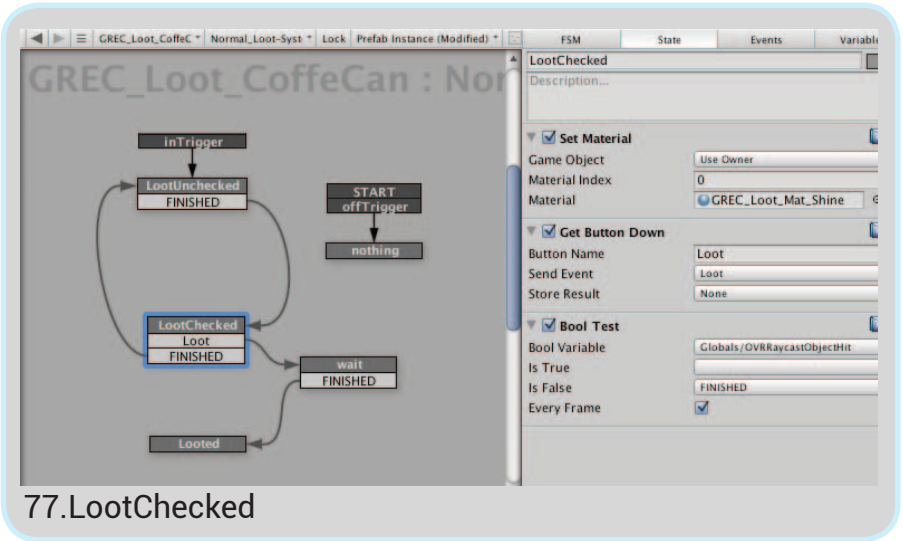
This FSM does the actual looting, plays the loot animation, adds a specific value to the loot counter, plays the looting sound and does the visual response to the player if an object can be looted. It's stored on the loot object itself.

After the game has started, the FSM switches into the “nothing” state, where the material is set to “GREC_Loot_Mat”. This material contains the texture of all lootable objects. The FSM also switches to this state, when the event “offTrigger” was sent from the “Normal_LootMainFSM” (character leaves the trigger zone).



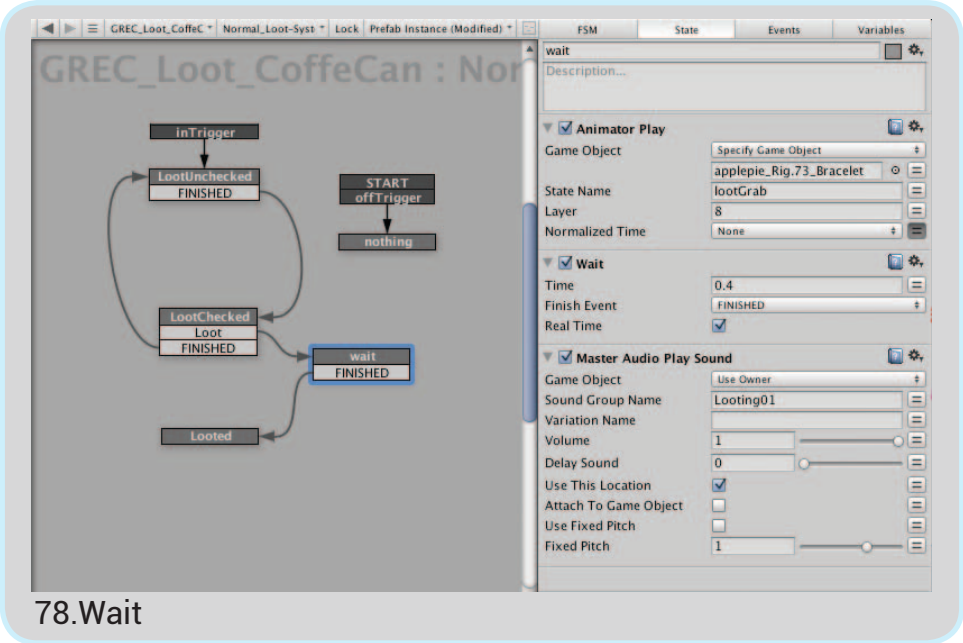
76.LootUnchecked

If the character enters the triggerzone around the lootable item, the “Normal_ Loot_MainFSM” sends the “inTrigger” global event, which directly points to the “LootUnchecked” state.
This state checks every frame if the bool “OVRRaycastObjectHit” is true (if the character is looking at the loot object). If this is given, the state “LootChecked” will be active.



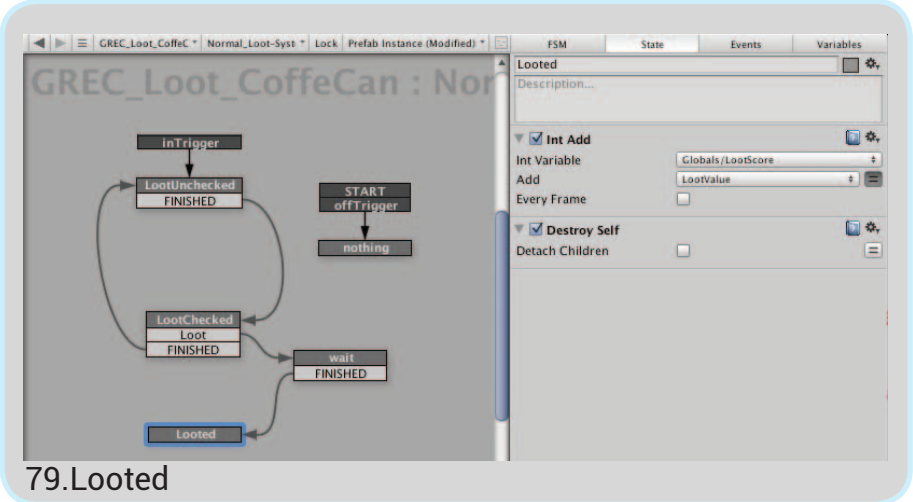
77.LootChecked

The “LootChecked” state sets the material to “GREC_Loot_Mat_Shine”, so now the player recognizes that the object can be looted. If the player isn't looking at the object anymore, the “Camera_Lootdetection_FSM” sets the bool “OVRRaycastObjectHit” to false. The state is changed to “LootUnchecked” and the the material is set to “GREC_Loot_Mat” again.
With the action event “Get Button Down” the system detects if the player presses the loot button to switch to the “wait” state. The Loot button was assigned in the Input Manager.



78.Wait

This state is responsible for playing the loot animation and the looting sound with the help of the “Animator Play” and the “Master Audio Play Sound” action events. “Animator Play” plays the animation which is stored inside the layer 8 on the state “lootGrab”. The loot sound comes from the group “Looting01” which was created in Master Audio. This group contains multiple sound files and the “Master Audio Play Sound” action event picks one file out of it.
The action event “Wait” waits a time of 0.4 seconds, which is exactly the time the animation needs to grab at the loot object.

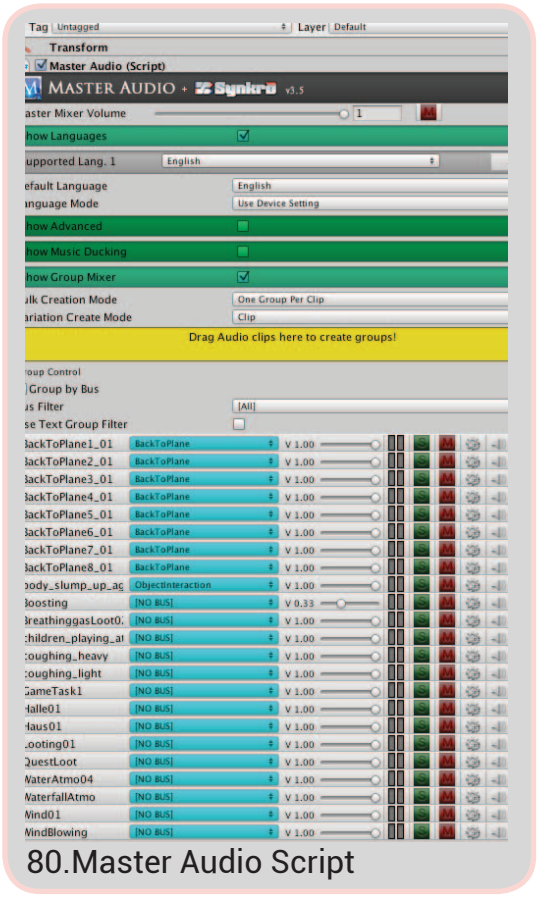


79.Looted

After the 0.4 seconds are exceeded, the state “Looted” becomes active. This state adds the value of the “LootValue” variable (this value is individual for each looting object) to the global “LootScore” variable. The loot score is displayed on the top of the left corner. The “Destroy Self” action event deletes the game object with it's FSMs on it, szo every object can be looted just one time.

Master Audio

With the bought asset Master Audio, it was possible to add sound effects to our game easily. Master Audio has a full integration into Playmaker, which makes this asset extremely powerful to control without coding.
Even without Playmaker it's nice to handle, because it features a whole list of Events and possibilities to fire of sound clips.
Master Audio can organize audio groups and put them in a pool together so it's possible to play a random sound stored in this group. We used this feature for our audio feedback which quest loots has to be found or are already found, to avoid repeating one clip all the time.
The main sound integration has been realized with Master Audio, apart from some little audio clips. These and the footsteps were directly played with the “Audio Play” Action Event which comes with Playmaker, because they were fired with the animation events and this Mecanim feature requires scripting.



80.Master Audio Script

Master Audio has a master script, which could be compared with an audio mixer. This script includes the creation audio groups, busses, playlists, custom events as well as a fader, a Master Mixer Volume control, and many other features. First of all we put all the needed sounds in this script and organized them into groups. Over Playmaker with the Action “Master Audio Fire Custom Event” we fired of the individual events we created.

To make this work we needed an additional script attached to our character. This script is called “Event Sound” and comes with Master Audio. In this script we had to apply the custom event we created as our “trigger” event. Then we had to choose which sound group we wanted to play which specific sound. This is the way we put sound together in our game. And all without coding.

81.Event Sounds

Where the budget went

Unity offers a quite interesting service: the Asset store. A built in online-share and shop platform to buy sell or exchange 3D-Models, Sounds, Editors or Particle Effects.
For G.R.E.C we made appropriate use of this possibility and purchased necessary assets directly. This assisted in making complex project like G.R.E.C manageable for two guys.

List of payed Assets:

Number	Asset	Asset Store URL
01	Autumnal Nature Pack	https://www.assetstore.unity3d.com/en/#!/content/3649
02	All Sky	https://www.assetstore.unity3d.com/en/#!/content/10109
03	Decal System Pro	https://www.assetstore.unity3d.com/en/#!/content/10608
04	Playmaker	https://www.assetstore.unity3d.com/en/#!/content/368
05	Master Audio: AAA Sound Solution	https://www.assetstore.unity3d.com/en/#!/content/5607
06	Industrial District	https://www.assetstore.unity3d.com/en/#!/content/16671
07	Jump Physics	https://www.assetstore.unity3d.com/en/#!/content/14117
08	Dynamic Bone	https://www.assetstore.unity3d.com/en/#!/content/16743
09	Relief Terrain Pack	https://www.assetstore.unity3d.com/en/#!/content/5664
10	Ultimate Rope Editor	https://www.assetstore.unity3d.com/en/#!/content/7279
11	Unity Constraints	https://www.assetstore.unity3d.com/en/#!/content/1919
12	Shader Forge	https://www.assetstore.unity3d.com/en/#!/content/14147
13	Breakable Barrel	https://www.assetstore.unity3d.com/en/#!/content/2884
14	Old Dishes	https://www.assetstore.unity3d.com/en/#!/content/5525
15	Breakable Whiskey Bottle	https://www.assetstore.unity3d.com/en/#!/content/6973
16	Freight Wagon WS02	https://www.assetstore.unity3d.com/en/#!/content/9536
17	Passenger Wagon WK06	https://www.assetstore.unity3d.com/en/#!/content/9738
18	Steam Locomotive LK 06	https://www.assetstore.unity3d.com/en/#!/content/10056
19	Dirty bathroom collection	https://www.assetstore.unity3d.com/en/#!/content/277
20	Auto Rikshaw – Tuk tuk 3d model	https://www.assetstore.unity3d.com/en/#!/content/13143
21	Dirty Room Set	https://www.assetstore.unity3d.com/en/#!/content/18703

The reasons for purchasing

01 Autumnal Nature Pack
This Pack includes trees, bushes and plants, skyboxes and a lot of different objects. Which fitted perfectly in our autumn based world design concept.

02 All Sky
We needed Skyboxes and the description in the asset store held true to its promise- “A palette of well over one hundred skies!“. Not only did this provide us with a variety of different sky options, but it also was on sale which allowed us to stay within our budget more effectively.

03 Decal System Pro
The Decal System Pro enables the user to create decals in a very comfortable way and with a descent graphical budget, however the price seemed a bit too high however it does provide the user with what they need.

04 Playmaker
Without Playmaker it wouldn't be possible to realize this game without a programmer. Playmaker is a visual scripting tool and is incredibly powerful. It makes it possible for a non-programmer to “code” without actual coding in a programming language. Playmaker was quintessential in enabling us to create G.R.E.C.

05 Master Audio: AAA Sound Solution
Master Audio allowed us to implement sound to our game without writing any code. This particular asset basically allowed us to determine when each sound is played. It also enabled us to create buses, sound groups and playlists. The full Playmaker support was just one reason to buy.

06 Industrial District
Is a package including wall parts and factory Probs, which allows the user to create large factories with a bit of steam punk design. We were quite lucky with this package because the style did fit quite well with our world design. It was possible to assemble nearly our whole city from this package.

07 Jump Physics
This script looked suitable for our jumping design, however after creating an initial jumping prototype we realized that we could not change the code to fit our specific needs. It was also not possible to integrate this script into Playmaker. This resulted in us having to use the Character Motor script instead and develop a custom solution.

08 Dynamic Bone
We thought we would use this plugin for animating the hair while walking and jumping, as the Web Demo looked promising. The problem was each wisp of hair clipped into each other every time they moved around. It became evident to us that the plugin was basically made for ponytails and not for short hair.

09 Relief Terrain Pack
As mentioned in the scout postmortem, the terrain in unity has a few weaknesses, mainly the shader. The Relief Terrain Pack is a way better solution for the terrain, it added parallax displacement to the bump and the blending from one texture to another, based on a normal map. It increased the quality tremendously.

10 Ultimate Rope Editor
Originally this plugin was intended for connecting together the breathing gas system to the character. We had planned for a flexible tube to go from the mouth piece to the breathing gas meter and from there down to the breathing gas tanks. The idea was nice, but the realization was tricky. After spending a lot of time, we found that it was not possible to avoid the tube clipping inside the characters head, arms and body. The Colliders couldn't be scaled small enough to get the desired result. So we decided to use this plugin to build the power cable for the power lines. This worked out very successfully, because if the players hits the power cable with the character, the reaction appears physically accurate. Additionally we created the hanging spores with the rope physics.

11 Unity Constraints
This asset was needed to build the first person version of G.R.E.C. We had to bind the camera to the neck joint to follow the position, but not the rotation, so it was more or less a last minute solution.

12 Shader Forge
We found that shader forge was a necessary addition to the unity editors. This Plugin allows the user to create shaders in a node based visual editor: A feature most other engines are shipped with.

13-21 Probs
Basically self-explanatory, models to set dress and for level design reasons.

List of free Assets:

Number	Asset	Asset Store URL
01	Dumpster v1.0	https://www.assetstore.unity3d.com/en/#!/content/2615
02	Large Cart v1	https://www.assetstore.unity3d.com/en/#!/content/19232
03	Medieval Gold	https://www.assetstore.unity3d.com/en/#!/content/14162
04	Piano v1.0	https://www.assetstore.unity3d.com/en/#!/content/154
05	Rocks Pack – Freebies	https://www.assetstore.unity3d.com/en/#!/content/13568
06	Rusty Tricycle v1.0	https://www.assetstore.unity3d.com/en/#!/content/19413
07	Shanty Town: Power Lines v1.0	https://www.assetstore.unity3d.com/en/#!/content/63
08	Shanty Town: Rock Formations v1.0	https://www.assetstore.unity3d.com/en/#!/content/65
09	Shanty Town: Tall Fence v1.0	https://www.assetstore.unity3d.com/en/#!/content/70
10	Shanty Town: Trees v1.0	https://www.assetstore.unity3d.com/en/#!/content/44
11	Shanty Town: Stove	https://www.assetstore.unity3d.com/en/#!/content/66
12	Small Cart v2	https://www.assetstore.unity3d.com/en/#!/content/19231
13	Shanty Town: Table Wood	https://www.assetstore.unity3d.com/en/#!/content/69
14	Terrain Assets v1.04	https://www.assetstore.unity3d.com/en/#!/content/6
15	Wood Fence v1.0	https://www.assetstore.unity3d.com/en/#!/content/2630
16	Wooden Plank Fence Construction Kit v1	https://www.assetstore.unity3d.com/en/#!/content/19237

Above is a small list of free assets available to everyone to download from the asset store. A great way to populate your level. Although a lot of the free stuff is junk, after spending some time searing through the options, we did however manage to collect a few pearls.

Chapter IV

Conclusion

Conclusion

Final thoughts

Our goal was to create a game in three months.
We didn't start completely from scratch, we had already made a game prototype. We failed at some points and zwe succeeded at other points, but we gained enough knowledge to make G.R.E.C possible.
We had a budget, the Asset Store and a plan, and we changed that plan for the better.

Finally, G.R.E.C is a game. The player has a task to fulfill, the game has a result, the player can fail, restart and can also have fun.
The Oculus Rift Integration raised our game to another level of interactivity and amazed people.
There are still bugs to fix, but for us the game is complete.

We made something we are proud of and hope people will enjoy G.R.E.C.

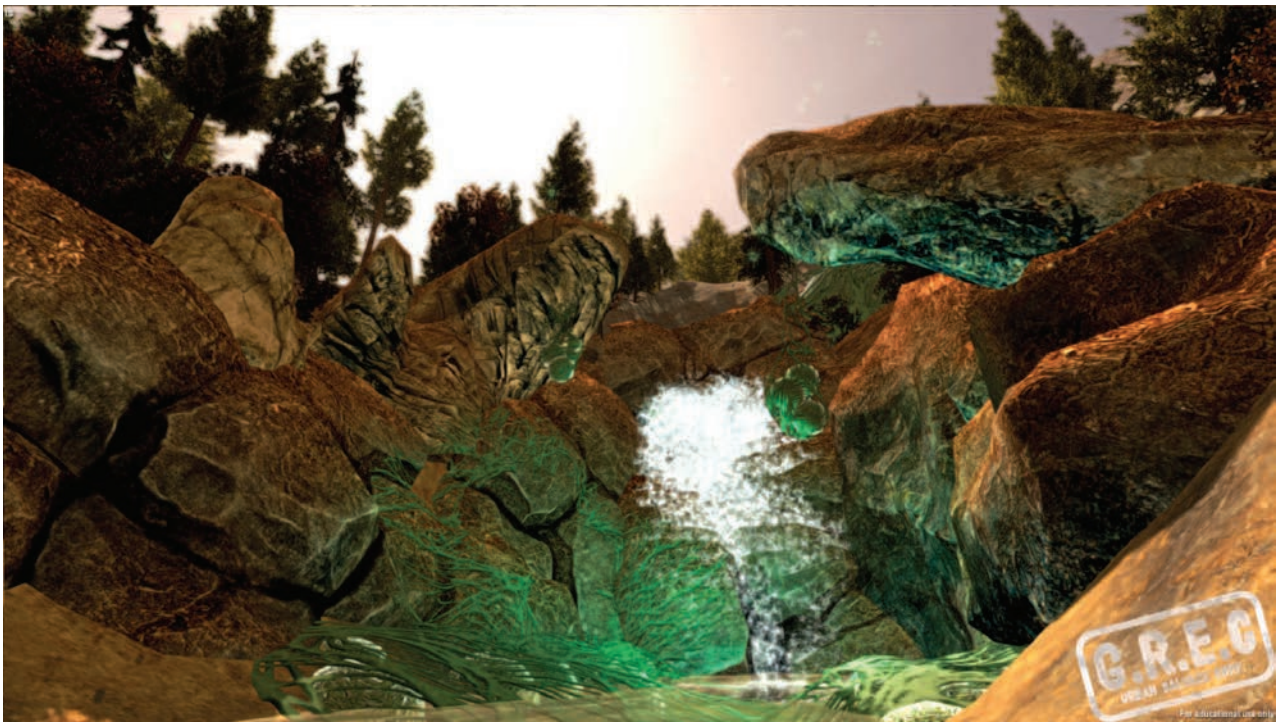
Chapter V

Appendix

Appendix

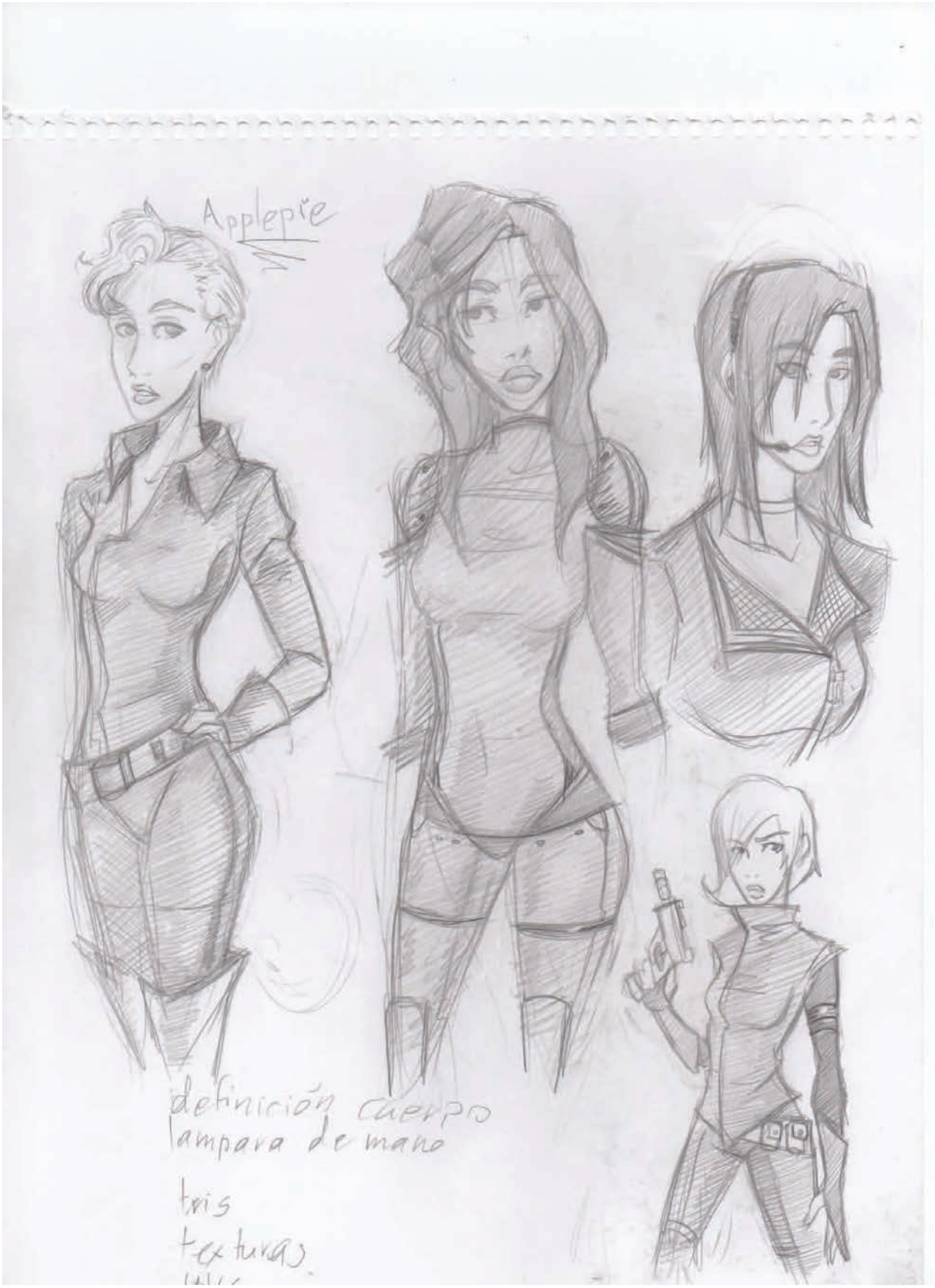
Screenshots


G.R.E.C
URBAN SALVAGE CORP.
For educational use only

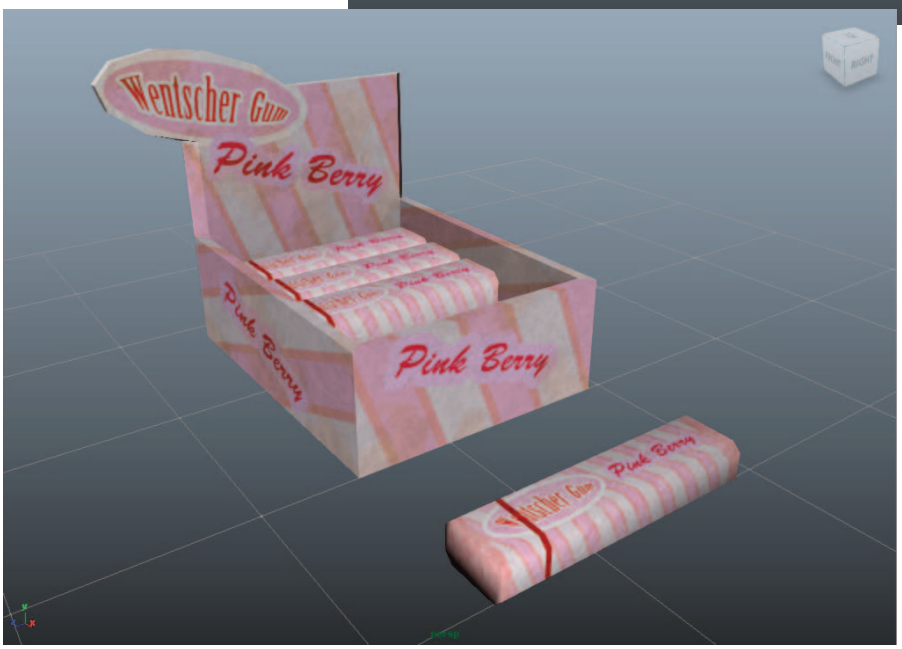

G.R.E.C


G.R.E.C


G.R.E.C


Applepie
definición cuerpo
lampara de mano
tris
texturas


Applepie
Pilot Jake


Wentscher Gum
Pink Berry
Pink Berry
Pink Berry
Pink Berry
Wentscher Gum

List of figures

List of references

This is a list of references, we gathered over the production time.

Footnotes

1. https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game/posts
2. https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game
3. http://www.oculusvr.com/rift/
4. http://i2.wp.com/www.roadtovr.com/wp-content/uploads/2013/04/oculus-rift-ifixit.jpg
5. https://share.oculusvr.com/app/oculus-tuscany-demo
6. http://www.teamfortress.com
7. https://developer.oculusvr.com/forums/viewtopic.php?f=29&t=4075

Game Design

http://www.gamasutra.com/

http://www.gamasutra.com/blogs/SebastienLambottin/20140718/220979/The_prototype_methodology_weve_used_for_the_naval_of_Assassins_creed_4.php

http://www.gamasutra.com/view/feature/2438/how_to_prototype_a_game_in_under_7_.php

Level Design

http://www.gamasutra.com/blogs/ChristopherTotten/20140711/220802/Excerpts_from_An_Architectural_Approach_to_Level_Design.php

http://www.worldofleveldesign.com/

http://www.gamasutra.com/view/feature/132714/action_adventure_level_design_.php?print=1

http://multiplayerblog.mtv.com/2009/10/13/exclusive-shadow-complex-prototype-map-revealed/

http://blog.digitaltutors.com/keeping-players-engaged-tips-great-game-level-design/

Playmaker

http://www.hutonggames.com

http://hutonggames.com/playmakerforum/

Miscellaneous

http://www.digitaltutors.com/11/index.php

http://unity3d.com

http://forum.unity3d.com

http://docs.unity3d.com/ScriptReference/