
CyberBeam

Development of a computer video game about beam
instrumentation for the CERN Open Days 2019

Written by

Adrián Navarro Pérez, Game Developer Intern at CERN

Summer Student Programme 2019

Supervised by

Irene Degl’Innocenti, Doctoral Student at CERN

Prévessin-Moëns, France

8/26/19



Table of contents

Table of contents	2
Table of figures	3
Disclaimer	5
1. Abstract	5
2. Introduction	5
3. Accelerated Physics and BI	6
4. Game Design	7
5. Gameplay	10
6. Used developer tools	10
6.1 Unreal Engine 4 [7]	11
6.2 Blender [10]	12
6.3 GIMP [11]	13
6.4 GarageBand [12]	13
6.5 Audacity [13]	14
6.6 GitHub [14]	15
7. Game implementation	16
7.1 Music, sound effects, voice effects, and cues	16
7.2 Fonts	18
7.3 Images, textures, materials, and material instances	19
7.4 Meshes	23
7.5 Particle systems	25
7.6 Blueprints	30
7.7 Widget Blueprints	32
7.8 Maps	34
8. Conclusion	35
Annex 1: BPM Simulator	36
Introduction	36
Development	36
Conclusion	37
Annex 2: Open Days 3D Tent	38
Introduction	38
Twinmotion	38
Development	38
Conclusion	41
Bibliography	42



Table of figures

Figure 1: The production and consumption of game artifacts. [4]	8
Figure 2: Consumption of games broken into their distinct components. [4]	8
Figure 3: Consumption of games' distinct components design counterparts. [4]	8
Figure 4: The different perspective between game designers and players. [4]	9
Figure 5: How to play to CyberBeam	10
Figure 6: CyberBeam running in Unreal Engine 4.22.3.	11
Figure 7: Example of the use of Blueprints in Unreal Engine 4. This Blueprint corresponds to the bunch the player controls in CyberBeam: BP_Beam	12
Figure 8: LHC accelerator's pipe model in Blender.	12
Figure 9: CyberBeam logo being created using GIMP.	13
Figure 10: CyberBeam main theme under production in GarageBand	14
Figure 11: Improving the successful_collision_new_record voice effect audio file using Audacity.	14
Figure 12: CyberBeam online repository at GitHub	15
Figure 13: List of the most recent Git commits to the CyberBeam repository.	15
Figure 14: Optima Regular font sample.	18
Figure 15: Designer view of the Main Menu Widget Blueprint, where the Optima font can be seen in use everywhere within the green canvas but in the CERN OpenDays logo	19
Figure 16: List of all the images, made using GIMP, which appear in the game.	19
Figure 17: All the textures, made using GIMP, which are used in the game.	20
Figure 18: List of all the materials used in CyberBeam.	20
Figure 19: All the material instances from M_Electrons_2	20
Figure 20: Drawing T_Electrons2 in GIMP	21
Figure 21: Creating M_Electrons_2 using Unreal Engine 4	21
Figure 22: Creating a material instance of M_Electrons_2 with an EmissiveBrightness value of 21.	22
Figure 23: Screenshot showing a material instance of M_Electrons_2 applied to the bunch's particle system with the maximum possible value for EmissiveBrightness: 30.	22
Figure 24: Screenshot showing a material instance of M_Electrons_2 applied to the bunch's particle system with a very low value for EmissiveBrightness	23
Figure 25: Comparison between the reality and CyberBeam.	23
Figure 26: List of all the static meshes used in the game.	24
Figure 27: Accelerator Blueprint's viewport. A set of assembled BPMs and the BTV are shown.	24
Figure 28: Wire Scanner inside of the LHC's pipe	25
Figure 29: PS_Beam1_2 - It represents the bunch the player control in the game.	26
Figure 30: PS_Beam2_2 - It represents a bunch going in the opposite direction during a game session.	26
Figure 31: PS_BeamMenu1_2 is the blue bunch which appears in the main menu	27
Figure 32: PS_BeamMenu2_2 is the red bunch which appears in the main menu.	27
Figure 33: When the player goes through the Wire Scanner, this particle system, PS_Radiation, is spawned representing the radiation emitted during this procedure in real life	28
Figure 34: PS_Crash is used when the bunch is lost in any way	28
Figure 35: PS_Collision is the particle system generated when there is a successful collision at the end of a game session	29
Figure 36: Successful collision achieved with maximum beam intensity	29
Figure 37: Successful collision achieved with very low beam intensity	30
Figure 38: BP_MainMenuManager showing its components, functions, variables, details, viewport, and event graph.	31



Figure 39: BP_Accelerator showing its components, functions, variables, details, viewport, and event graph.	31
Figure 40: BP_Beam showing its components, functions, variables, details, viewport, and event graph.	32
Figure 41: BP_Beam2 showing its components, functions, variables, details, viewport, and event graph.	32
Figure 42: WPB_mainMenu Blueprint Designer Mode.	33
Figure 43: WBP_mainMenu Graph Editing Mode.	33
Figure 44: WBP_HUD Blueprint Designer Mode.	34
Figure 45: WBP_HUD Graph Editing Mode.	34
Figure 46: Initial state of the MainMenu map.	35
Figure 47: Initial state of the SinglePlayer map.	35
Figure 48: BPM Simulator with high intensity.	36
Figure 49: BPM Simulator with low intensity.	37
Figure 50: BPM Simulator with the lowest intensity.	37
Figure 51: Twinmotion's working environment.	38
Figure 52: The original 2D distribution of the tent.	39
Figure 53: Top view of the 3D schematic tent.	39
Figure 54: Top view of the 3D tent.	40
Figure 55: Top-side view of the 3D tent.	40
Figure 56: View from the tent's entrance.	41
Figure 57: Adrián explaining some visitors how to play to CyberBeam.	41



Disclaimer

This article is kept as simple as possible to address any kind of reader, regardless of their academic background. Therefore, no previous knowledge of Game Development or Physics is required to fully grasp it.

Furthermore, some of the topics discussed below are quite subjective and the most proficient readers could differ from the point of view expressed here, and, in particular, in the 2. Game Design section.

1. Abstract

This paper focuses on the development process of CyberBeam, a 3D serious video game¹ developed at the European Organization for Nuclear Research (CERN) in the Beam Instrumentation (BI) group, through the months of June, July, and August as a project for the CERN Summer Student Programme 2019.

The goal behind the development of this video game is to illustrate in a simple, interactive, and entertaining way, the role and the importance of beam monitoring instruments in a particle collider as the Large Hadron Collider (LHC).

The game is going to be available for everyone, at the BI Group stand, during the CERN Open Days 2019, which are being hold on the 14th and 15th September.

Keywords: Accelerator physics, Audacity, Beam Instrumentation, Blender, Computer game, Design, Game, Game design, Game development, GarageBand, GIMP, GitHub, Learning experience, Open Days, Serious game, Simulator, Twinmotion, Unreal Engine 4, Video game.

2. Introduction

Usually, at events like CERN Open Days, a lot of photos, videos, simulations, and much more multimedia content is prepared to create a learning experience for all guests. Nevertheless, given the amount of information that is provided to the visitors in a short amount of time, they might have difficulties in contextualizing and linking the concepts explained and the components displayed.

At this point, the idea of creating an interactive experience to support the event's learning experience was starting to be born within the BI group.

The BI group is the main responsible for creating, optimizing and maintaining up to date all the instruments which make possible the observation and measurement of particle beams throughout their entire lifetime. [1]

¹ It is understood as a serious game as the one which its main purpose is to provide players with an effective and specific learning experience.



Irene Degl’Innocenti, doctoral student part of the BI group, came up with the idea to develop a simple video game together with Adrián Navarro Pérez, summer student part of the same group, with the intention to reach as much audience as possible at the upcoming CERN Open Days.

This way, visitors approaching the BI group stand at the event will be able to experiment this interactive and educative experience. At the same time, they will consolidate all the acquired knowledge during the event, specifically that directly related with the activities carried out by the BI group.

Thus, the game was developed from the following general concept by Irene Degl’Innocenti:

“A BASIC VIDEO GAME SHOWING THE TRIP OF A PARTICLE BEAM INSIDE A CIRCULAR COLLIDER AND THE INTERACTION WITH THE INSTRUMENTS GIVING FEEDBACKS IN ORDER TO REACH THE SUCCESSFUL FINAL COLLISION.”

In this document, the reader will follow CyberBeam’s development process, from its very first beginning, until its very end.

3. Accelerator physics and BI

CyberBeam pursues reproducing the trip of a bunch of particles inside a circular collider as the LHC.

Particles are grouped in bunches, which form a beam. At the LHC, two beams going in opposite directions collide between them in one of the four collision points, also known as big experiments, which are called ATLAS, ALICE, CMS, and LHCb.

These collisions are carried out in order to study the properties of particles and answer to fundamental questions as those concerning the origin of the Universe.

In the game, the player is exposed to several fundamental parameters of accelerated particle beams and to some of the instruments used to monitor them.

First of all, the LHC is an accelerator, so the energy of every single particle in a bunch increases during the run.

After the injection in the collider from the previous accelerator, the beam is at phase and stops when the beam reaches maximum energy and it is said to be at flat top.

The beam parameters to monitor, which are presented in the game, in order to operate the accelerator are the following: [2]

- Beam intensity: How many particles are part of a single beam.
- Position: Where the center of mass of the beam is located in the transverse plane.

- Beam tune: How many oscillations in both vertical and horizontal planes the particles do per turn in the machine.
- Beam profile: How the distribution of the particles in space looks.
- Beam loss: Beam losing particles and where.

As for the instruments which appear in-game, are:

- Beam Position Monitor (BPM): As the beam propagates in the pipe, its position is measured by detecting the electro-magnetic field surrounding the particles, or by measuring the image current induced on the metallic pipe.
- Beam TV (BTV): For the measurement of the beam size and shape, a movable screen is inserted into the beam trajectory. The interaction of the particles with the screen, generates light which is captured through an optical system and a camera.
- Beam Wire Scanner (BWS): To measure the transverse beam profile, a very thin carbon wire passes through the beam. The secondary particle shower generated is detected and transformed into an electrical current, from which the beam profile is reconstructed.

4. Game Design

Before starting to develop further this first section and stage of game development, a definition of what is understood as a game must be given first:

“A GAME IS AN ARTEFACT WHICH SUPPORTS A VOLUNTARY INTERACTION CARRIED OUT, WITHIN A FORMAL INDEPENDENT TRANSMEDIAL SYSTEM, BETWEEN ONE OR MORE USERS AND THE SYSTEM ITSELF PERFORMING A FINITE NUMBER OF DIFFERENT TYPES OF ACTIONS IN ORDER TO PRODUCE A MEANINGFUL PLEASANT PHYSICAL OR PSYCHOLOGICAL REWARD SUCH AS KNOWLEDGE, EXPERIENCE, HEALTH, WEALTH OR SOCIAL BONDS WHICH EVENTUALLY LEAD TO USER’S ENTERTAINMENT, SATISFACTION, AND ULTIMATELY, HAPPINESS.” [2]

Therefore, the role of a game designer, based in Zimmerman and Salen’s work in *Rules of Play* [3] together with the previous definition, remains as the following:

A GAME DESIGNER CONSTRUCTS AN INTERACTIVE, ENTERTAINING, AND SATISFACTORY GAME EXPERIENCE ENSURING THAT PLAYERS WILL BE WILLING TO PARTICIPATE ON IT WITH NO RESTRICTIONS. [2]

With the purpose of designing CyberBeam, keeping in mind the general concept defined by Irene Degl’Innocenti, and the educative goals the game should feature, a framework called Mechanics-Dynamics-Aesthetics (MDA) described by Robin Hunicke, Marc LeBlanc, and Robert Zubeck in *MDA: A Formal Approach to Game Design and Game Research* [4], has been used.

As any other consumable good, games are created by game designers and teams of developers, consumed by players, and finally, left behind when they have no use for them anymore.



Figure 1: The production and consumption of game artifacts. [4]

The MDA framework breaks down this type of consumption applied to games in three components: Rules, System, and “Fun”.



Figure 2: Consumption of games broken into their distinct components. [4]

These three components can be associated with three design concepts which are linked in the same way as the previous ones: Mechanics, Dynamics, and Aesthetics.



Figure 3: Consumption of games' distinct components design counterparts. [4]

- Game Mechanics, according to Miguel Sicart in *Defining Game Mechanics* [5], are methods invoked by agents, designed for interaction with the game state.
- Game Dynamics can also be understood as the definition given to Game Design Patterns by Staffan Björk, Sus Lundgren, and Jussi Holopainen in *Game Design Patterns* [6], which are defined as semi-formal interdependent descriptions of commonly reoccurring parts of the design of a game that concern gameplay².
- Game Aesthetics, according to Robin Hunicke, Marc LeBlanc, and Robert Zubeck in *MDA: A Formal Approach to Game Design and Game Research* [4], describe the desirable emotional responses evoked in the player, when they interact with the game system.

² Gameplay refers to the features of a game which define the way it is played regardless of its media, platform, or even its own content.

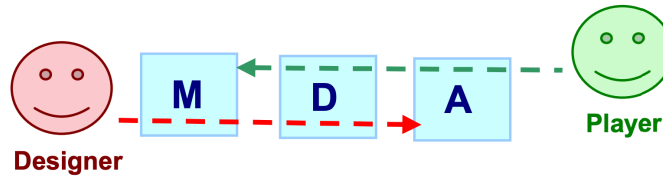


Figure 4: The different perspective between game designers and players. [4]

However, this is not a unidirectional but a bidirectional graph. Some games, as the one featuring a famous Italian plumber, are built entirely around a specific mechanic, such as jumping, and, from that point on, they decide which game dynamics can be created from the defined game mechanics to achieve determined aesthetics. On the other hand, game designers might want to build a game to make the player feel in a certain way through specific aesthetics, and from that point on, they will think on which dynamics can support them and which mechanics will be required to create such dynamics.

In CyberBeam, all the game has been developed around a specific aesthetic, **Discovery**: Players enjoy the game as an uncharted territory they have to explore. This uncharted territory, yet to discover, is learning about the use of beam instrumentation for accelerator physics.

Now, skipping right to the final stage of CyberBeam’s design, readers can understand better all the concepts just defined, see by themselves how all the MDA framework has been applied to the early stages of the game, and how supportive it can be.

The following text summarizes CyberBeam’s gameplay. Game mechanics are written in **bold** and game dynamics are underlined, both to ease the job of the reader.

In CyberBeam, you control a bunch of particles from a particle beam which has just been injected into the LHC at CERN, and it is about to collide. Achieve the perfect collision by **adjusting its trajectory** and staying in the center of the LHC’s pipe so any particle gets lost. **Adjust its tune** and keep it at a good value so your bunch does not experience resonances that might lead to instabilities and particle loss. Get extra points by performing well when going through the center of sets of BPMs and a BTV, and keeping a good tune when going through a BWS. After a short while the required energy to collide is reached and you can then aim for the other beam going in the opposite direction to finally collide with it and complete the game.

Basically, players can **adjust the bunch trajectory** and **its tune**, which are the main mechanics of the game. By using these, players can stay in the center of the pipe, keep a good tune and aim for the other beam, which are some of CyberBeam’s game dynamics.

As the conclusion of this first section regarding CyberBeam's game design, it is important to understand that, after a good design of the game, all the following steps to complete its development became easier. All the elements, such as the rules of the game or their entities, are clearly defined now as well as how they should be implemented and how they support the goal of the game designer, to create the appropriate learning experience, which is different than the actual goal of the game, which is to achieve the final collision.

5. Gameplay

Before diving into the implementation of CyberBeam and which tools have been used to make it happen, the next image summarizes the whole content of the game and how to play to it, so the reader can get a general idea with just a quick look:



Figure 5: How to play to CyberBeam.

6. Used developer tools

It is important to first understand which software tools have been used all through CyberBeam's development and the roles they portrayed on it.

Additionally, all these programs are free-open source³, source-available⁴ or free to use⁵.

6.1 Unreal Engine 4 [7]

Unreal Engine 4, also known as UE4, is a source-available game engine developed by Epic Games and released in 2014. The first Unreal Engine version was first introduced in May 1998 together with a first-person shooter known as *Unreal*. [8]

A game engine is an integrated development environment (IDE) mainly used for game development. Game engines bring together all the functionalities that any game developer might need during the development of any game: a renderer engine, a physics engine, sounds integration, scripting, animation, artificial intelligence, and support for localization services or cinematics. This way, a game engine can be used to develop different video games in an efficient and economical way since it brings all these functionalities, and much more, altogether.

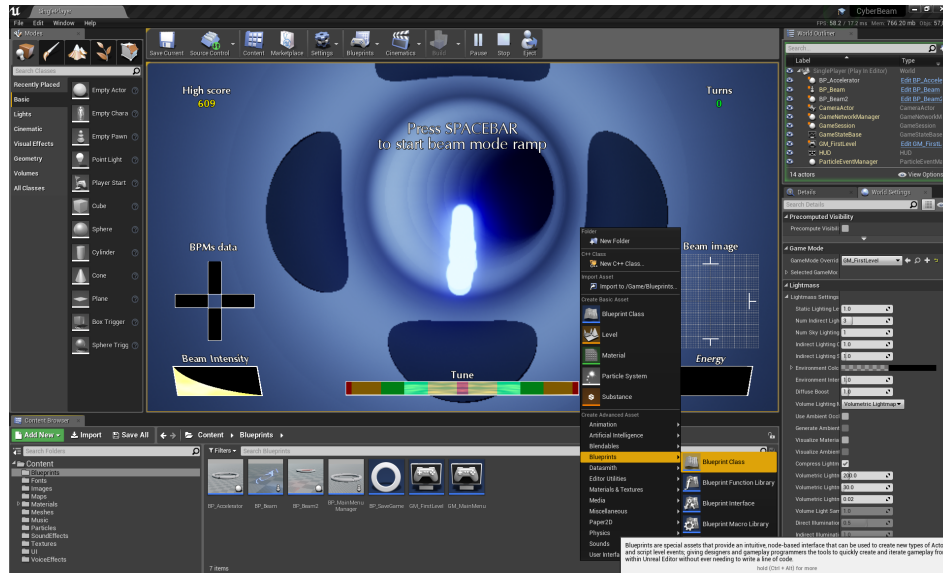


Figure 6: CyberBeam running in Unreal Engine 4.22.3.

Another advantage of making use of Unreal Engine 4, is the possibility to program all the gameplay logic of the game by using the Blueprints Visual Scripting system⁶ which can replace writing code. This system is very friendly for game

³ Understood as an open-source software which is free to use by anyone. It is encouraged to copy and modify the software in any possible way according to each one's needs, as well as improving it when possible.

⁴ Software's source is open to everyone. However, there might be some limitations on the use of the program due how it is licensed, so it cannot be called free-open source but source-available.

⁵ Unlike the other two, here the code is not open to anybody, but the software is free to use with no restrictions.

⁶ The Blueprints Visual Scripting system featured in Unreal Engine 4 is a gameplay scripting system based on making use of a node-based interface in order to create any kind of gameplay elements. The objects created using this system are also known as Blueprints.

6. Used developer tools CyberBeam

developers who are familiar with programming, but they do not feel as comfortable and proficient as a former programmer. [9]

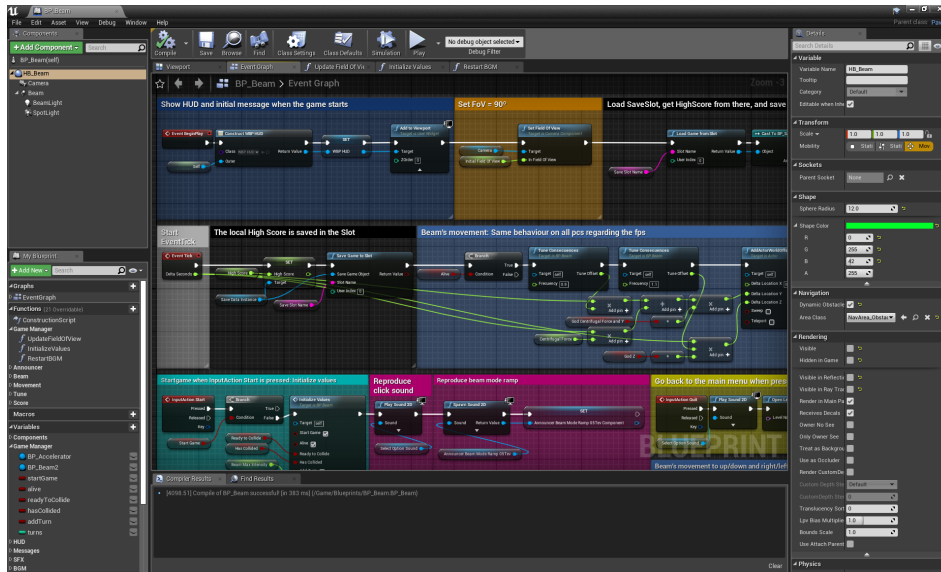


Figure 7: Example of the use of Blueprints in Unreal Engine 4. This Blueprint corresponds to the bunch the player controls in CyberBeam: BP_Beam.

6.2 Blender [10]

Blender is a free-open source 3D creation suite which has been used to create the models of all the static meshes⁷ of CyberBeam, including the LHC accelerator's pipe, the BPMs, the BTV, and the BWS.

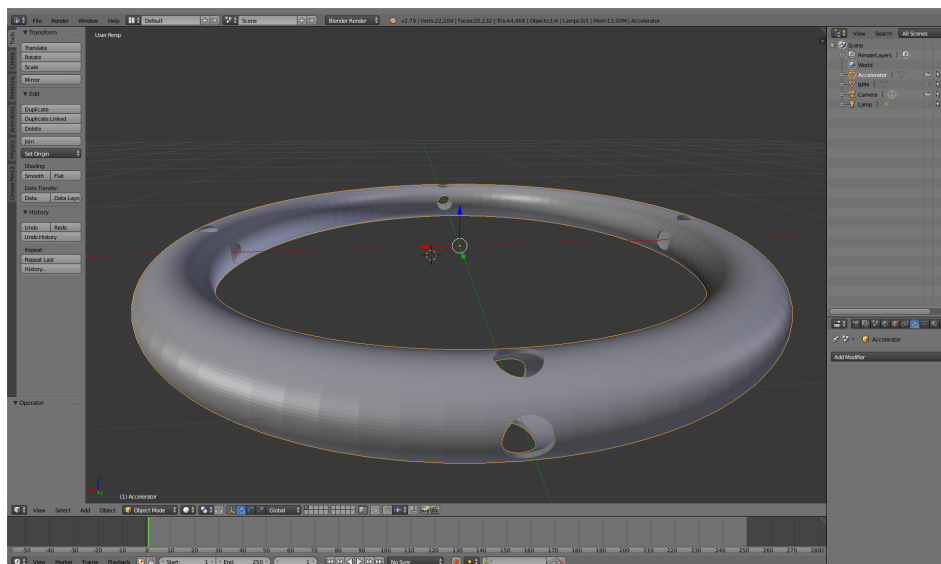


Figure 8: LHC accelerator's pipe model in Blender.

⁷ The collection of vertices, edges, and faces which describe the shape of a 3D object is considered a mesh.



This program provides support for all the 3D pipeline, which includes modeling, rigging, animation, simulation, rendering, compositing, and motion tracking, it even includes video editing and game creation.

6.3 GIMP [11]

The GNU Image Manipulation Program (GIMP) is a free open-source cross platform image editor.

Despite all the great amount of features GIMP offers, this program has only been used to create CyberBeam's logo, some simple textures⁸, and perform small tweaks in certain images, such as adding to them a transparent layer when needed.

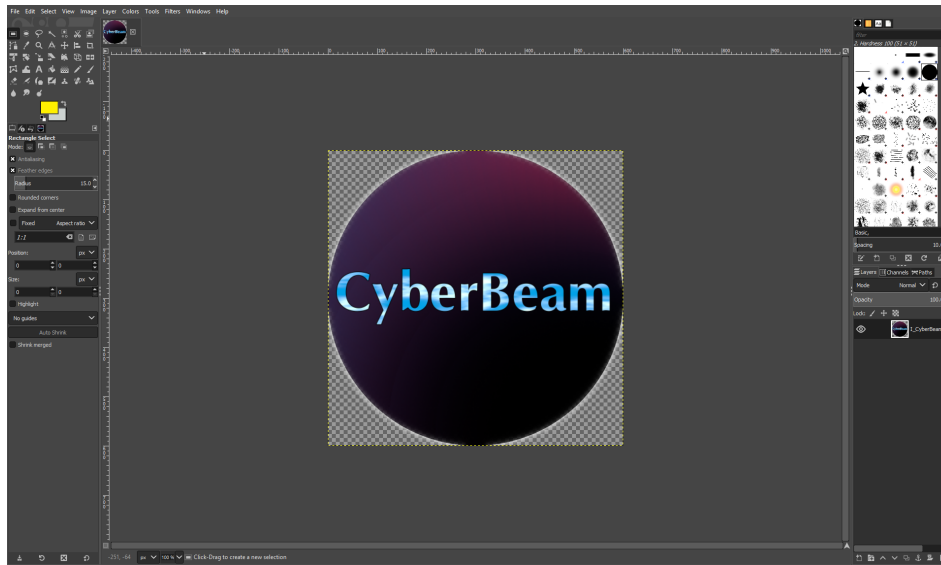


Figure 9: CyberBeam logo being created using GIMP.

6.4 GarageBand [12]

GarageBand is a free digital audio workstation for Mac and iOS with a complete library that includes instruments, presets, and a selection of session drummers and percussionists. It also includes a good number of sounds and loops to add to any composition.

As every digital audio workstation, GarageBand makes possible the creation of sound effects and music as well as their modification.

In CyberBeam, this software has been used to create almost all of the sound effects of the game as well as all its music from scratch, making use of the beforementioned GarageBand features.

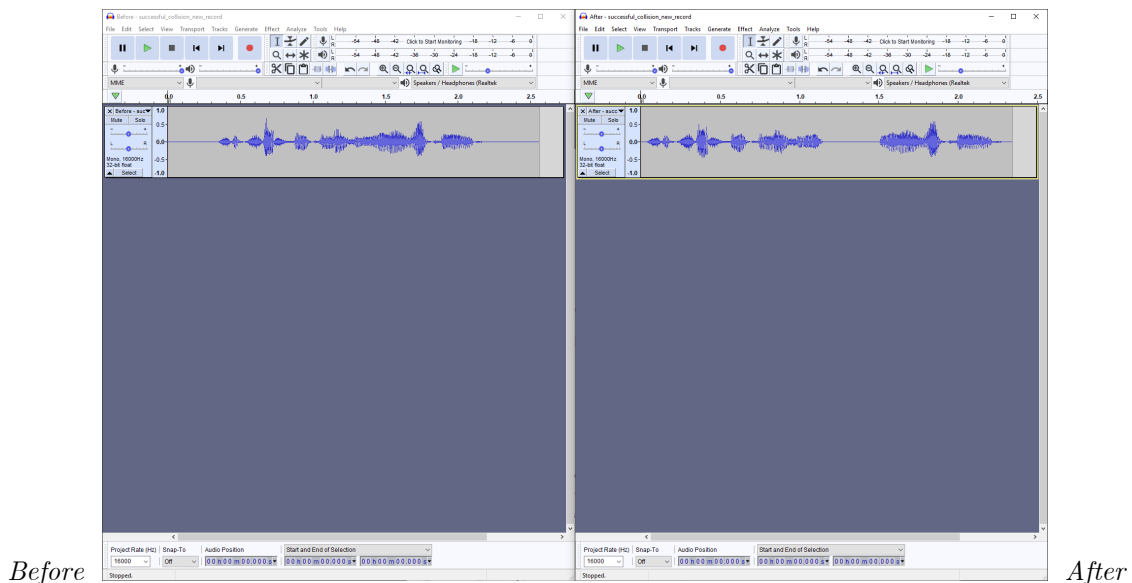
⁸ Textures are images, often wrapped, or patterns which are applied to a 2D or 3D surface.



Figure 10: CyberBeam main theme under production in GarageBand.

6.5 Audacity [13]

Audacity is a free open-source cross platform digital audio editor and recording software. It also supports post-processing of all types of audio, which is the main function that has been used for CyberBeam's sound effects and voices.

Figure 11: Improving the `successful_collision_new_record` voice effect audio file using Audacity.

One example of how Audacity was used is shown on the picture above: The duration of one of the CyberBeam's announcer voices clips was not matching the actual length of its voice lines. The sentences `successful collision` and `new record` were said too quickly. This is why its timing had to be adjusted. Moreover, any imperfection of the audio file was fixed at this point too.

6.6 GitHub [14]

GitHub is an online development platform service which provides hosting for software development control version using Git⁹.

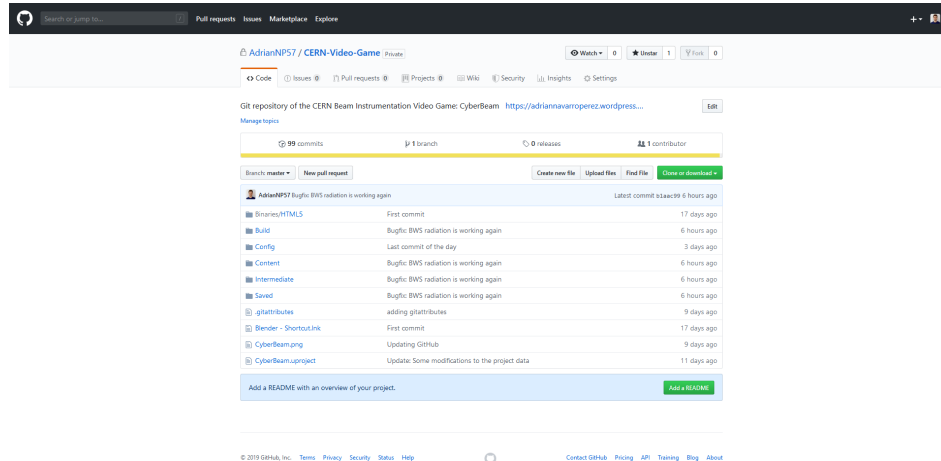


Figure 12: CyberBeam online repository at GitHub.

A proper software development control version is essential when creating a project of the extension of CyberBeam, and even more when several developers are part of the same project, although this has not been the case for CyberBeam.

Thanks to GitHub, it has been possible to keep track of all the changes the game has gone through as well as having the possibility to revert any of them at any point given for any reason, as for example, due malfunctioning code.

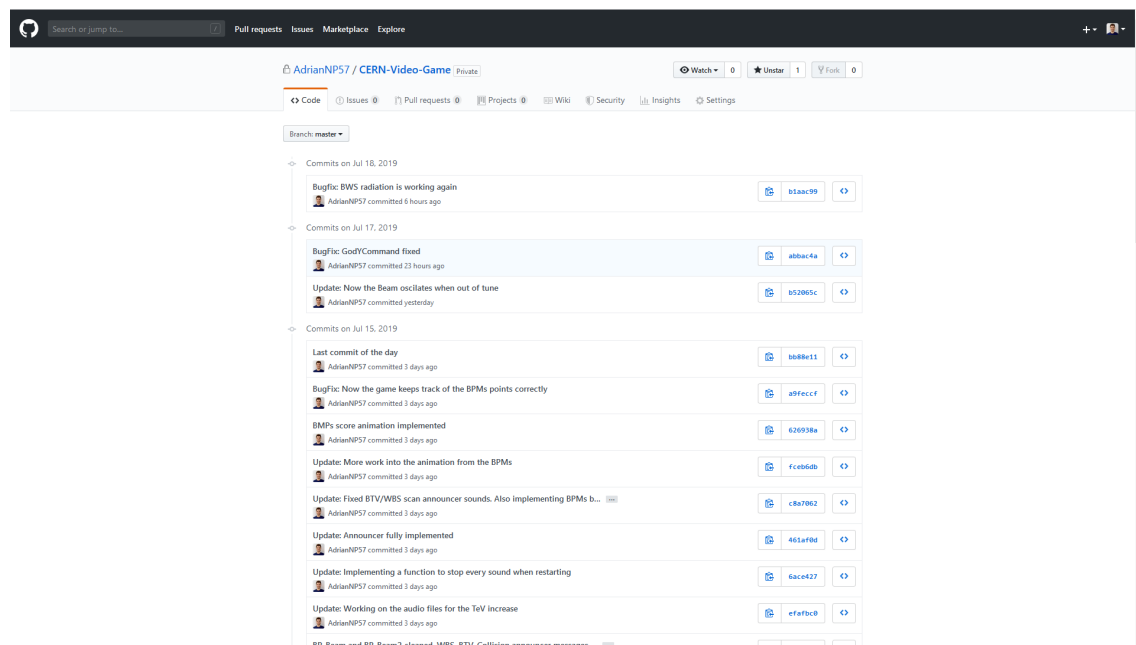


Figure 13: List of the most recent Git commits to the CyberBeam repository.

⁹ Git is a free open-source distributed control version system designed to handle any kind of project regardless of their size.



7. Game implementation

A video game is made of several elements which need to be implemented altogether. On this paper, in order to describe, as simple as possible, the whole development of CyberBeam, it has been decided to categorize all the content depending on its main functionality and use.

7.1 Music, sound effects, voice effects, and cues

A video game makes use of music, sound effects, and voice lines to improve users' experience when playing.

In CyberBeam, a total of 4 music themes have been composed, 6 sound effects have been recorded, 4 have been used from the Unreal Engine 4 Content available sound effects, and 14 voice lines have been generated by Stephen Page, the person in charge of the LHC Announcer [15], making use of the Festival speech synthesis system [16].

The music and sound effects have been produced using GarageBand and all media has been post-processed using Audacity when needed.

In Unreal Engine 4, from each audio file, another type of file, called sound cue, can be generated. Sound cues are instances of an audio file which can be modified according to the game's needs.

The best example to show the potential of these cues, can be found in the cue generated from the BGM_MainTheme.wav file. While playing the game, as the energy increases, a Continuous Modulator¹⁰ implemented in CUE_MainTheme, is modified so the pitch of the song increases proportionally to the beam's energy gain, which increases the immersion of the player when playing the game. Furthermore, as it is supposed to be background music, it has also been configured so it infinitely loops, unlike the cues generated from sound effects, which only play once every time they are triggered.

Here is the list of all the audio files and when they are used as cues in the game: Background Music (BGM):

- BGM_MainMenuTheme: This is the theme that can be listened to at CyberBeam's main menu.
- BGM_MainTheme: This is the main theme of the game which the player hears most of the time while playing.
- BGM_VictoryTheme: When the player achieves a successful collision at the end of a game session, this song starts to play.

¹⁰ A Continuous Modulator allows manipulation of pitch and volume real-time thanks to a modifiable parameter.



- BGM_GameOverTheme: When the player lets beam intensity drop to 0, the game session is over, and this song starts playing.

Recorded Sound Effects (SFX):

- SFX_Collision: When the player makes a successful collision, this sound effect is reproduced.
- SFX_Crash: When the bunch crashes on LHC's pipe or any instrument, this sound is then triggered.
- SFX_MoveOption2: This sound is used when enabling and disabling testing options within the game. Therefore, this sound effect cannot be heard by playing a regular game session.
- SFX_NewHighScore: When a higher score is achieved, the player listens to this sound effect.
- SFX_SelectOption: When the player starts a game session, this sound is reproduced.
- SFX_SelectOption2: When the player proceeds from the main menu to the main level of the game, this audio can be heard.

Unreal Engine 4 Sound Effects (SFX):

- Camera_Shutter.wav renamed as SFX_Camera_Shutter: When the bunch goes through the BTV, this camera shutter effect is triggered.
- Click_on_Button.wav renamed as SFX_BPMsActivation: Each time the bunch goes through a set of BPMs this sound can be listened to.
- Invalid_Action.wav renamed as SFX_Wire.wav: When the wire from the Wire Scanner goes through the player's bunch, this sound is reproduced.
- VR_confirm renamed as SFX_Turn.wav: Each time the bunch makes a whole turn in the collider, this audio file is then triggered.

Voice lines:

- VFX_beam_mode_ramp+energy_0_5.ogg: When the player starts a game session, this voice line is said.
- VFX_btv_scan.ogg: When the bunch goes through the BTV, the announcer confirms to the player the beam has successfully been scanned.
- VFX_energy_X_X_tev_and_rising.ogg: This set of audio files are reproduced continuously as the energy of the bunch is increasing, letting the player know its actual value real-time.
- VFX_energy_stable_ay_3_25_tev+flat_top.ogg: When the bunch is ready to collide, this audio is played.
- VFX_injection.ogg: Every time the game is ready to start a new game session, this voice line is heard.
- VFX_successful_collision_good_job.ogg: When the player manages to get a successful collision without beating the current high score, this sound is triggered.

- VFX_successful_collision_new_record.ogg: When the player manages to get a successful collision and they achieve a new high score, this sentence is said instead of VFX_successful_collision_good_job.ogg.
- VFX_unstable_beam_beam_lost.ogg: If the bunch loses all its intensity, this audio file is played.
- VFX_wire_scan: This sound confirms players that the Wire Scanner has scanned the bunch successfully.

7.2 Fonts

The font that has been used for all the text shown in CyberBeam is Optima, originally designed by Hermann Zapf [17].

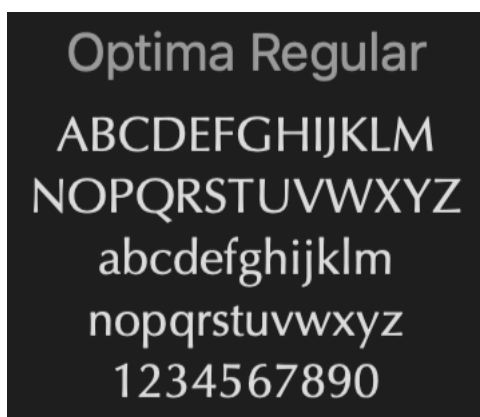


Figure 14: Optima Regular font sample.

The reason behind making use of this font is because it is the font that should be used at CERN whenever is possible, following CERN's design guidelines [18], as it is the same one used for the CERN word in CERN's logo at the moment of writing this report.

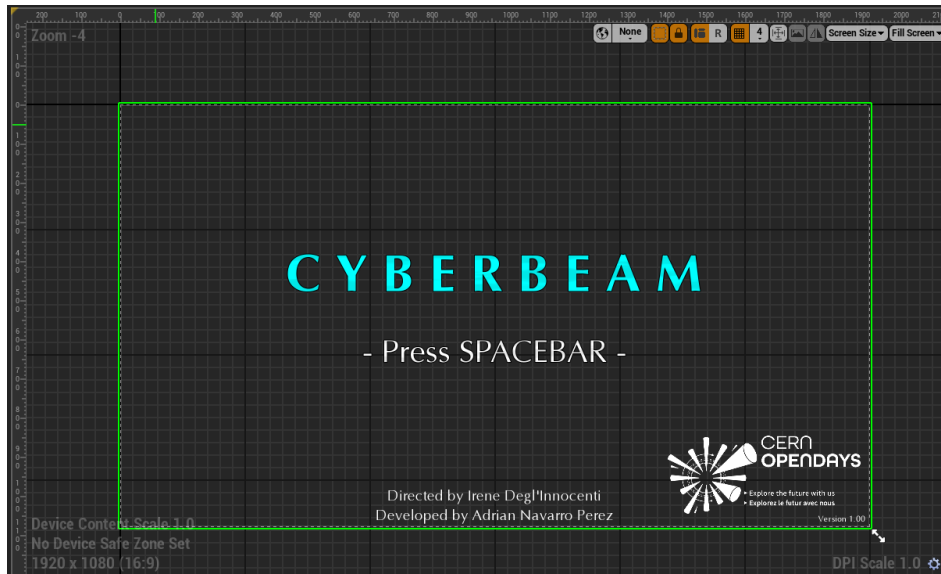


Figure 15: Designer view of the Main Menu Widget Blueprint¹¹, where the Optima font can be seen in use everywhere within the green canvas but in the CERN OpenDays logo.

7.3 Images, textures, materials, and material instances

Images and textures are mainly used in a video game as part of the user interface or as the principal component to create any material.

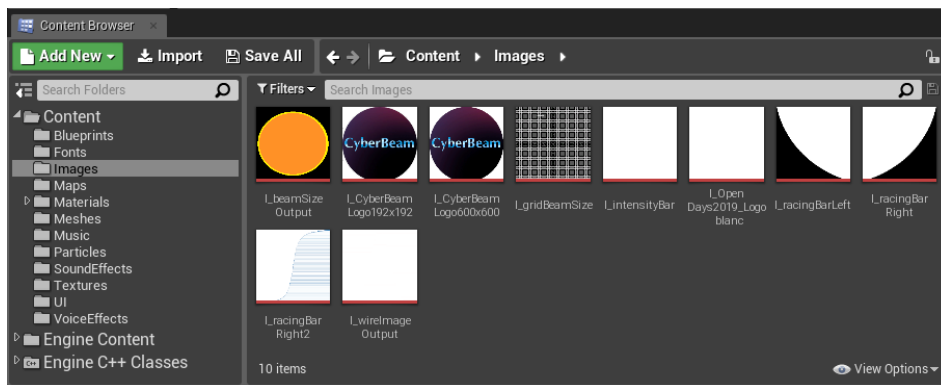


Figure 16: List of all the images, made using GIMP, which appear in the game.

¹¹ A Widget Blueprint is a specific type of Blueprint which provides better tools and services to build user interfaces (UIs) or Head-Up Displays (HUDs) than a regular Blueprint.

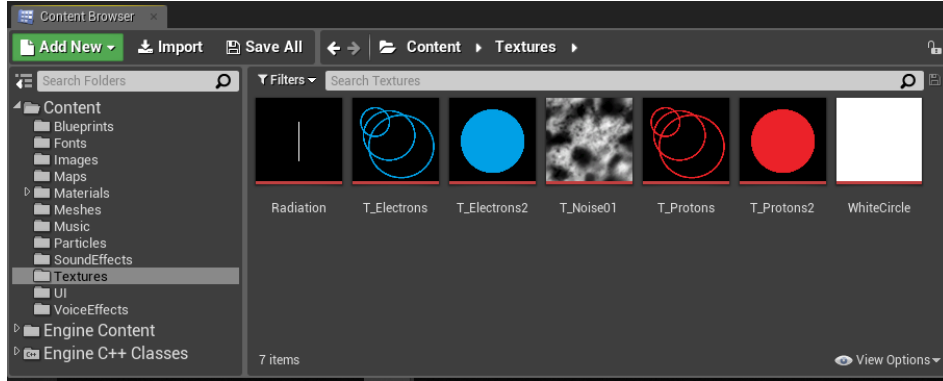


Figure 17: All the textures, made using GIMP¹², which are used in the game.

On the other hand, materials are assets that can be applied to a mesh to control the visual look of the game's scene [19]. As for material instances, the name is self-explanatory: Material instances are variations of a material which share all the properties with its parent except for the ones the programmer wants to vary.

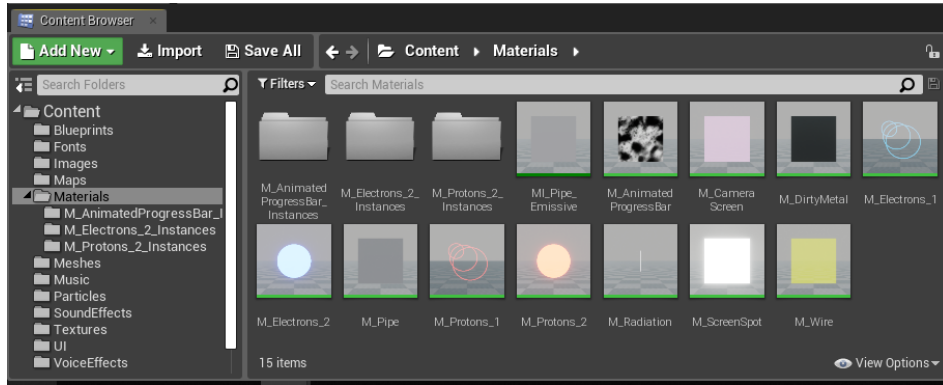


Figure 18: List of all the materials used in CyberBeam.¹³

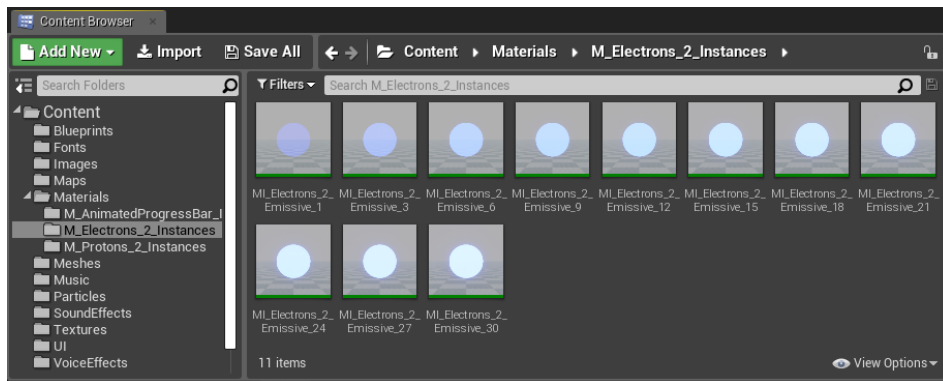


Figure 19: All the material instances from M_Electrons_2.

As an example of how the materials have been used in the game, on the next paragraph it is described the use of the T_Electrons2 texture in order to create

¹² The texture T_Noise01 shown above is part of the Unreal Engine 4 content so it is not made by CyberBeam's developer. Furthermore, the textures T_Electrons and T_Protons are early versions of T_Electrons2 and T_Protons2, which are not used in the game anymore.

¹³ M_Pipe, M_Pipe_Emissive, and M_DirtyMetal are modifications from existing materials of the Unreal Engine 4 content.

the `M_Electrons_2` material and its respective set of material instances. All of these material instances have been used to create the bunch of particles as a particle system, an important component to create visual effects in video games. Particle systems are described more in detail in the 5.5 Particle system section.

First of all, the texture had to be drawn from scratch using GIMP as usual.

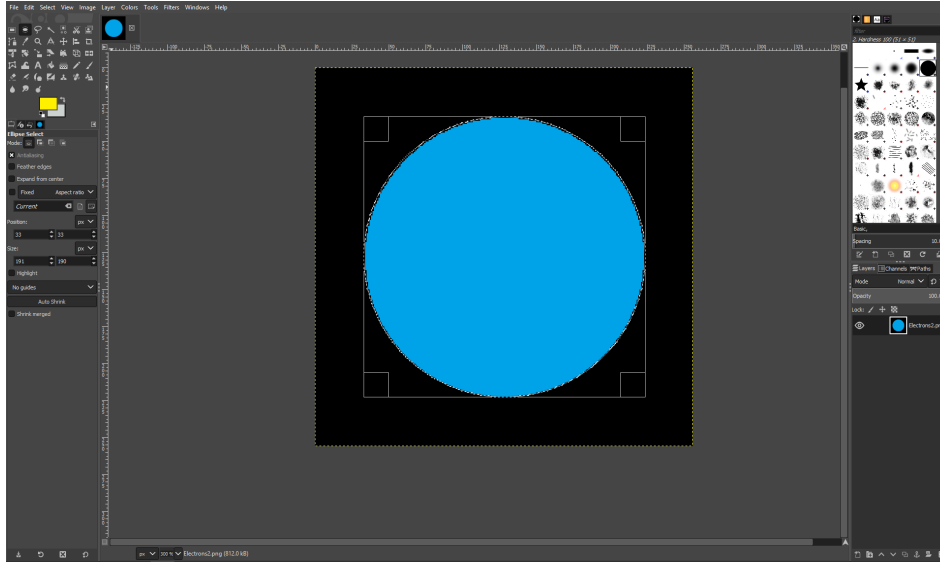


Figure 20: Drawing `T_Electrons2` in GIMP.

Secondly, once the texture was imported in the Unreal Engine 4 CyberBeam's project, a material making use of the `T_Electrons2` texture and with specific characteristics had to be created. The main characteristic implemented when creating this material, is the possibility to change its Emissive Color when creating a material instance from it through a parameter, named as `EmissiveBrightness`, so it can be reduced proportionally to the beam's intensity loss in-game.

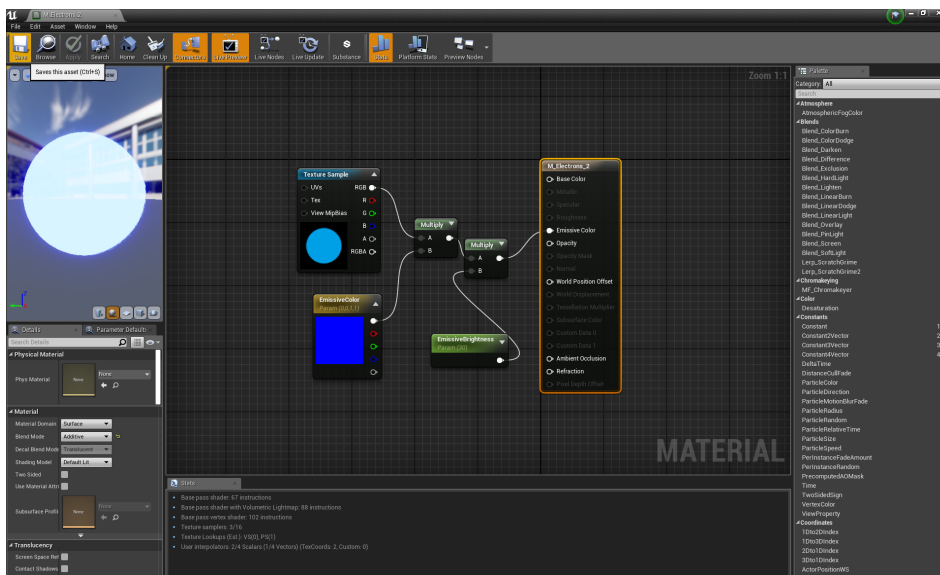


Figure 21: Creating `M_Electrons_2` using Unreal Engine 4.

It was decided then to create a total of 11 material instances from `M_Electrons_2` with different values for their `EmissiveBrightness` parameter, from 1 to 30 including the number 1 and all the multiples of 3.

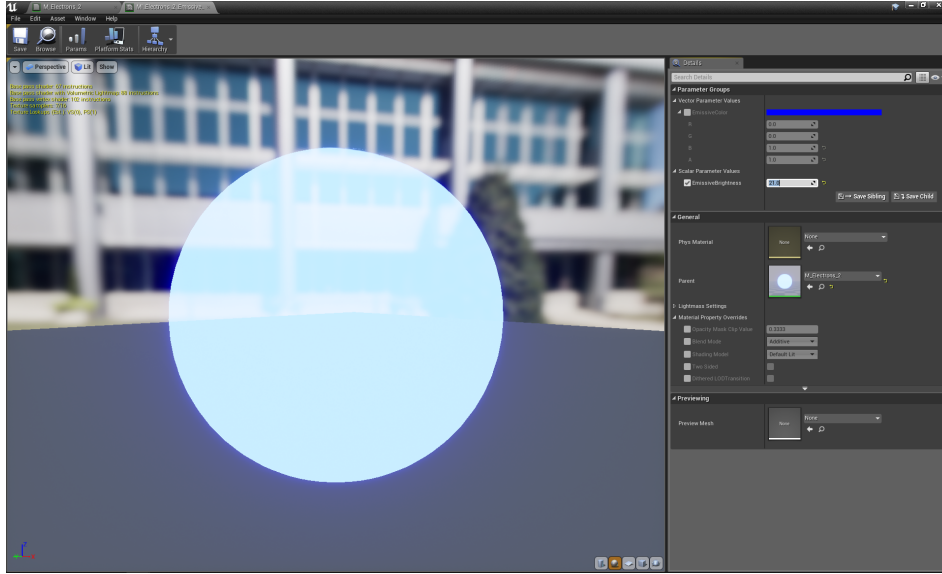


Figure 22: Creating a material instance of `M_Electrons_2` with an `EmissiveBrightness` value of 21.

Finally, the material instance of the bunch's particle system changes according to the current beam intensity after implementing this behavior into the beam's Blueprint, described more in detail in the 5.6 Blueprints section.

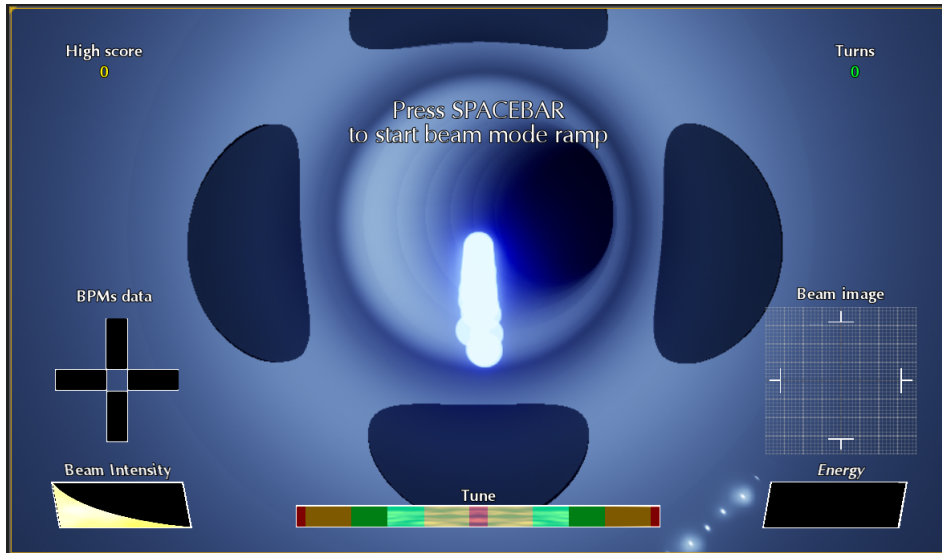


Figure 23: Screenshot showing a material instance of `M_Electrons_2` applied to the bunch's particle system with the maximum possible value for `EmissiveBrightness`: 30.

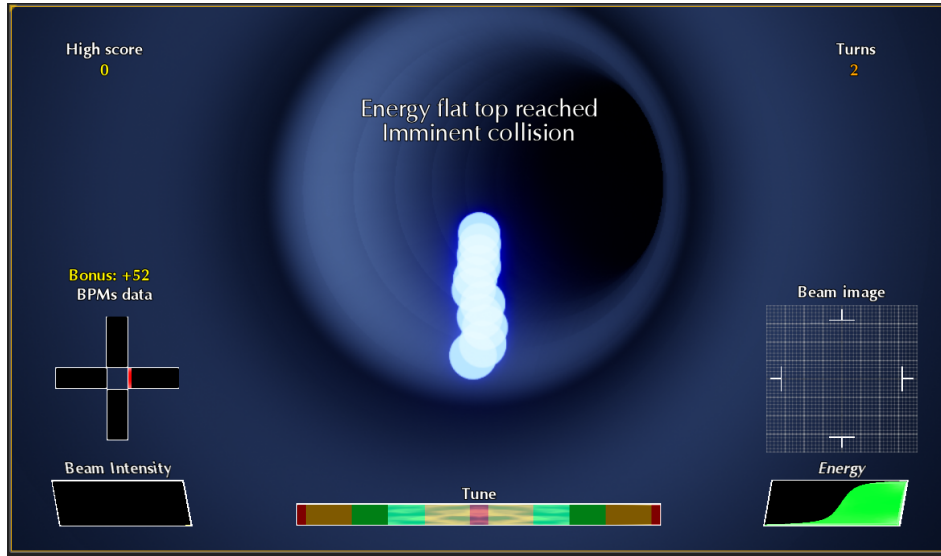


Figure 24: Screenshot showing a material instance of M_Electrons_2 applied to the bunch's particle system with a very low value for EmissiveBrightness.

7.4 Meshes

Once a model is imported to Unreal Engine 4, a static mesh is generated. After the static mesh is ready, it can be adjusted according to the needs of the game. Modifying its materials, describing its physics behavior, or adding a hitbox collision component to detect interactions between other game entities, are some of the things that can be done. After these adjustments, static meshes can then be added into the game's scene directly or added as components of a Blueprint which will be used in the game.

The models used to create the static meshes were inspired by the real instruments:

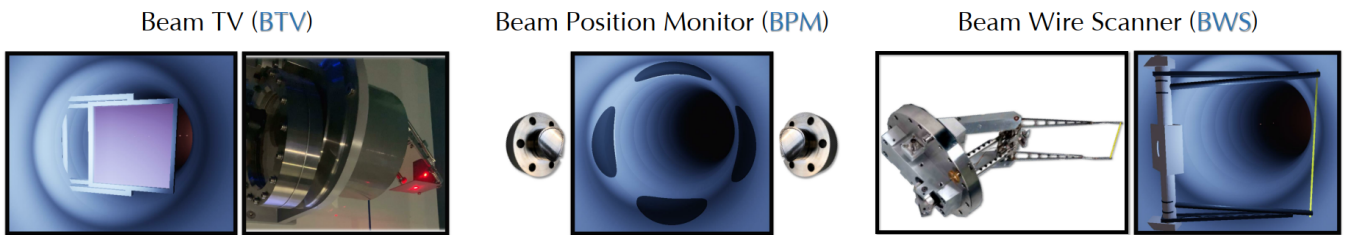


Figure 25: Comparison between the reality and CyberBeam.

A total amount of 7 static meshes have been used in CyberBeam, 1 of them for the LHC's pipe, 1 of them for the BPMs, 2 of them for the BTV, and the last 3 of them for the Wire Scanner.

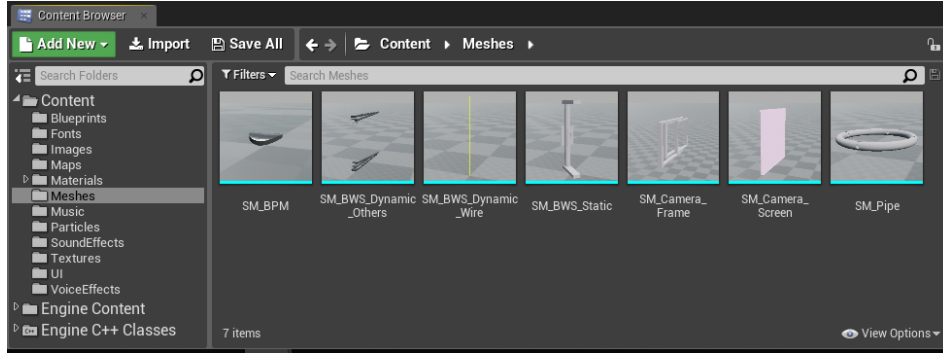


Figure 26: List of all the static meshes used in the game.

The reason behind having more than one static mesh for the BTV and the Wire Scanner, is because they needed to have different textures and behavior, even though it will form the same element at the end. For example, the bunch should be able to go through the screen of the BTV but not through its frame. Furthermore, the event that should be triggered when going through its screen is to take a picture of the beam, but the event that has to be triggered when colliding with the frame is a game over status since the bunch is lost due such collision.

In the following pictures taken from the accelerator's Blueprint viewport [20], a window showing the world you create, the final result of putting all the created static meshes together, can be seen.

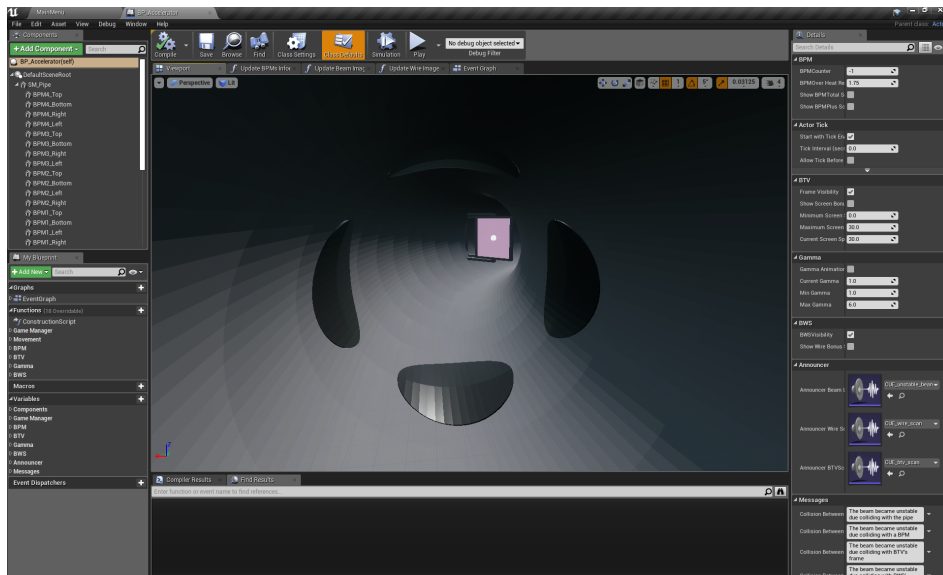


Figure 27: Accelerator Blueprint's viewport. A set of assembled BPMs and the BTV are shown.

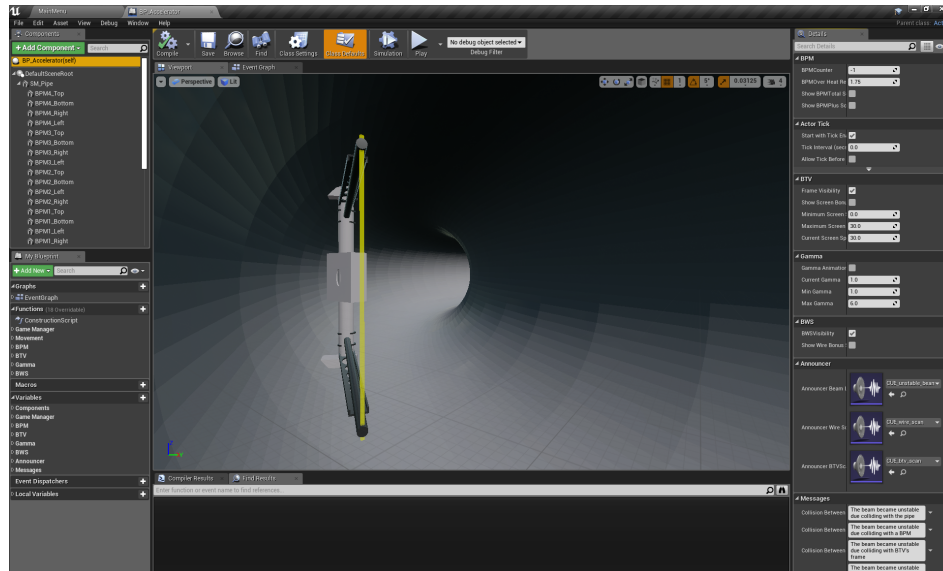


Figure 28: Wire Scanner inside of the LHC's pipe.

7.5 Particle systems

Although some hints have been given of what is a particle system before in this paper, now it will be defined more in depth.

As one might guess by its name, a particle system is a system made of a group of emitters which spawn particles at a certain point in space. At the same time, these emitters are divided by modules which define the behavior of each of the particles generated at that certain emitter. The particles that are generated can be 2D assets, 3D models... [21]

Thus, this system is inheritability perfect to create a bunch of particles close to the reality, hence, it has been used to define the bunches that appear in the game: the one the player controls and the other one going the other way around. However, these are not the only two particle systems that have been used.

This is the complete list of the particle systems that appear in CyberBeam:

7. Game implementation

CyberBeam

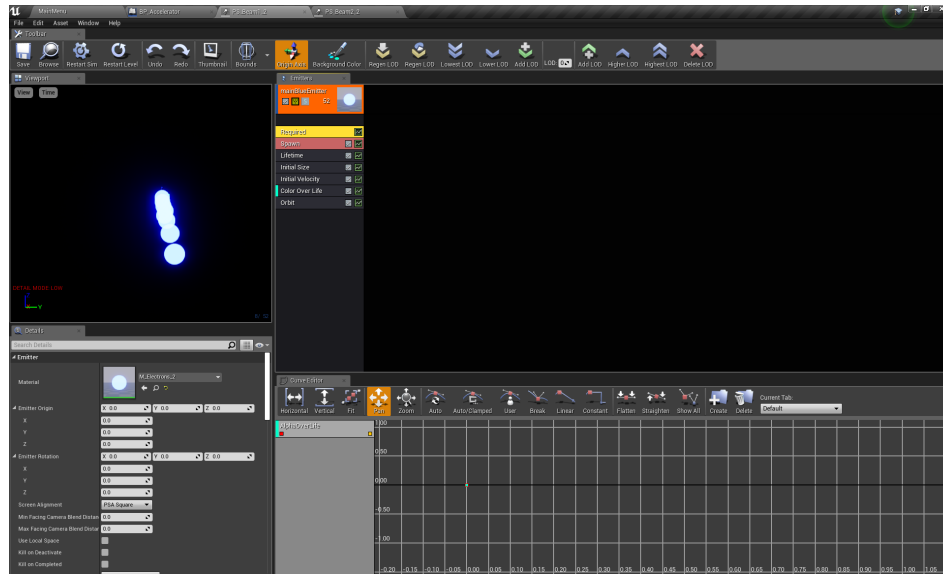


Figure 29: PS_Beam1_2 - It represents the bunch the player control in the game.

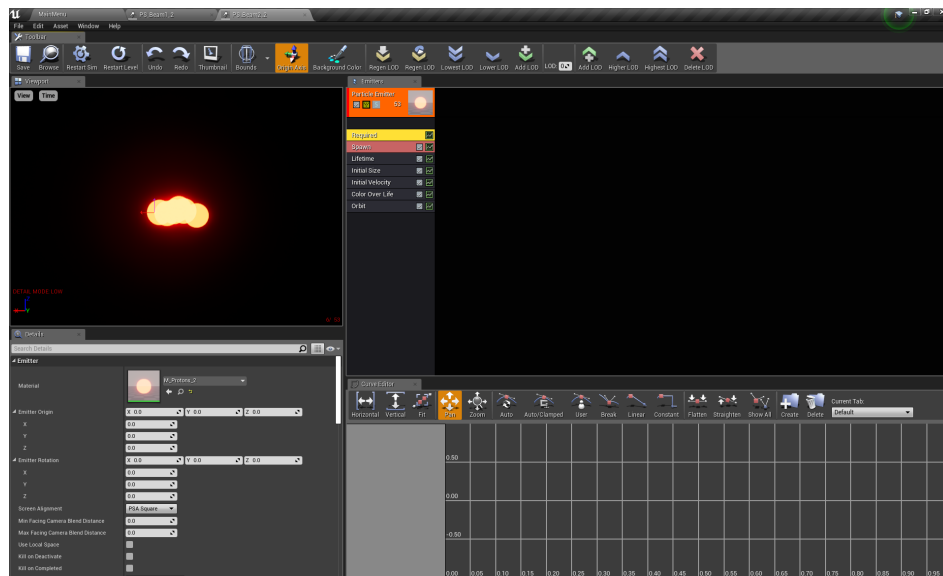


Figure 30: PS_Beam2_2 - It represents a bunch going in the opposite direction during a game session.

7. Game implementation

CyberBeam

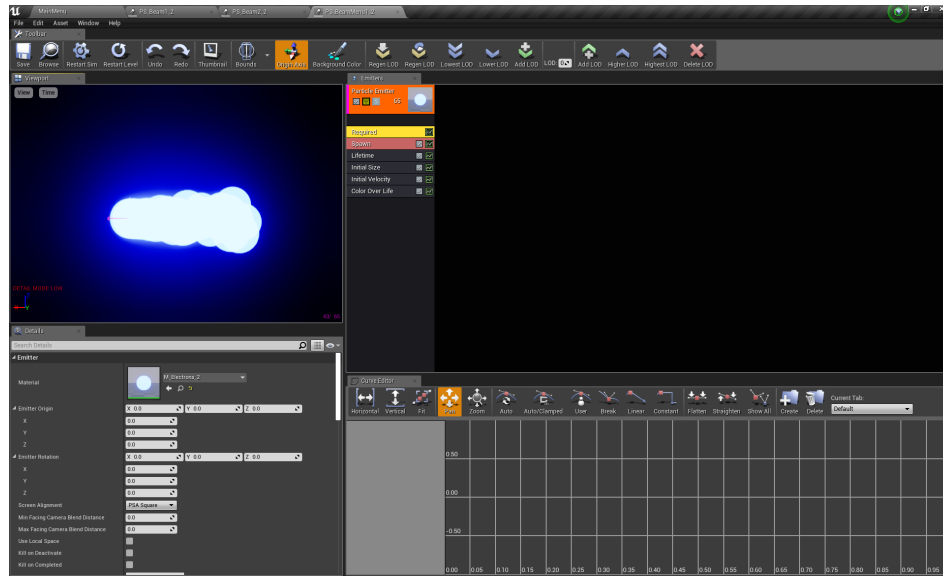


Figure 31: PS_BeamMenu1_2 is the blue bunch which appears in the main menu.

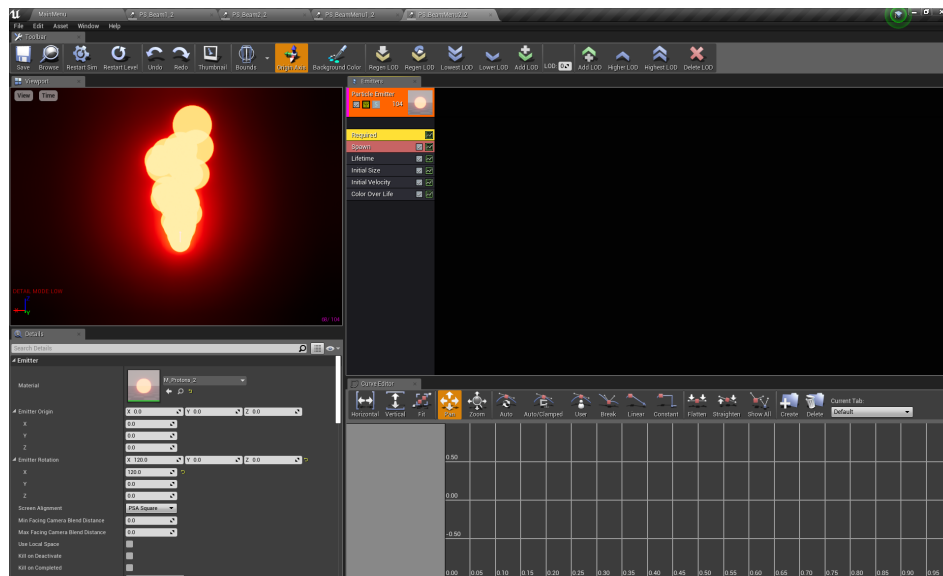


Figure 32: PS_BeamMenu2_2 is the red bunch which appears in the main menu.

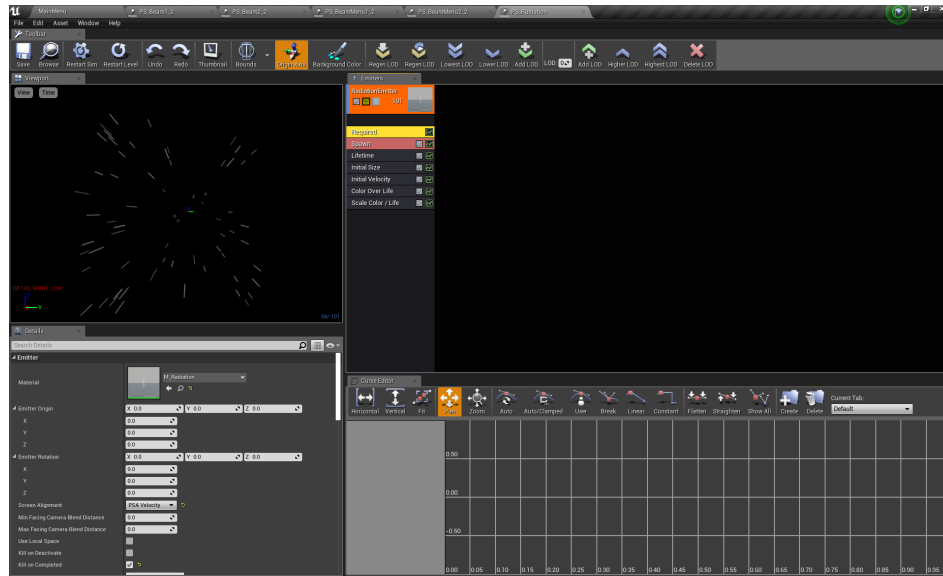


Figure 33: When the player goes through the Wire Scanner, this particle system, PS_Radiation, is spawned representing the radiation emitted during this procedure in real life.

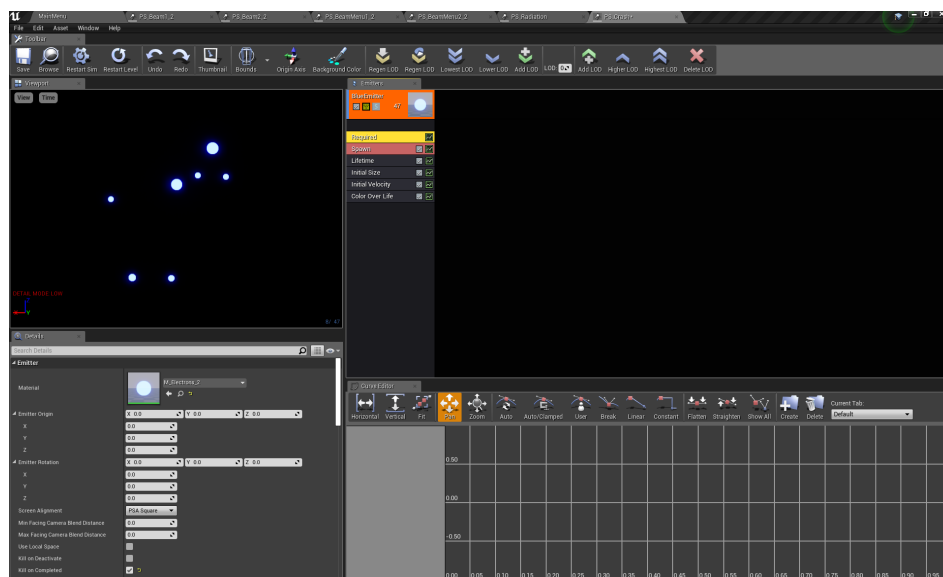


Figure 34: PS_Crash is used when the bunch is lost in any way.

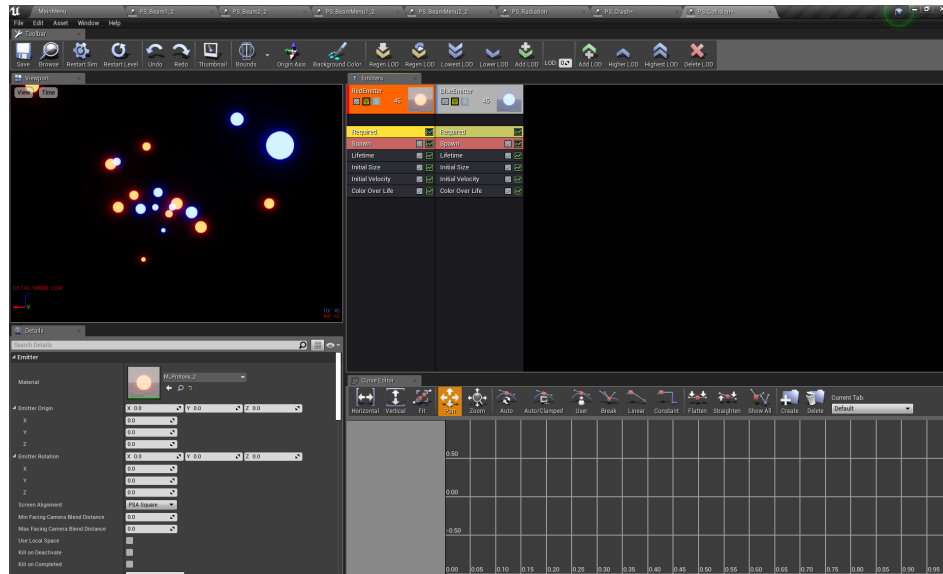


Figure 35: PS_Collision is the particle system generated when there is a successful collision at the end of a game session.

Finally, together with the different material instances applied to all the particle systems, as described in 5.3 Images, textures, materials, and material instances, the number of particles generated on their emitters is directly proportional to the current beam intensity, as it can be compared between the following two images:

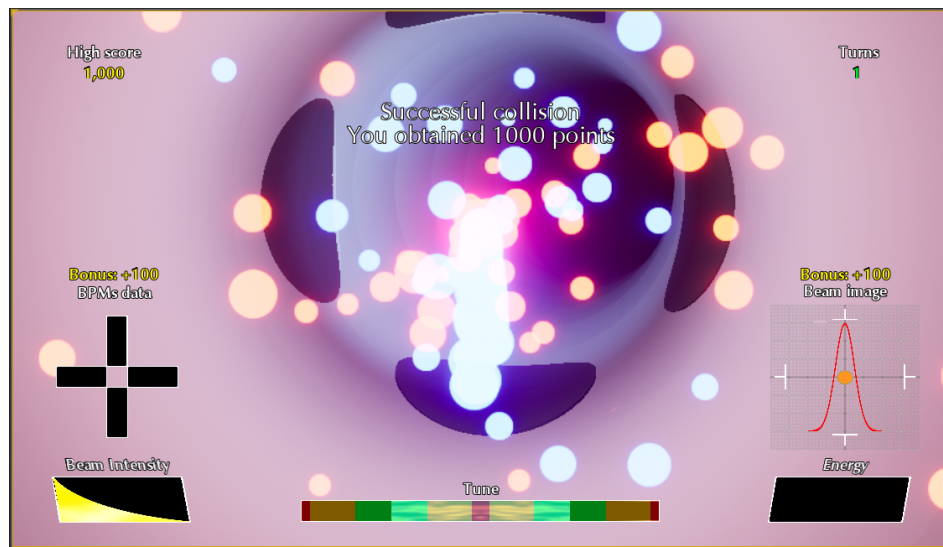


Figure 36: Successful collision achieved with maximum beam intensity.

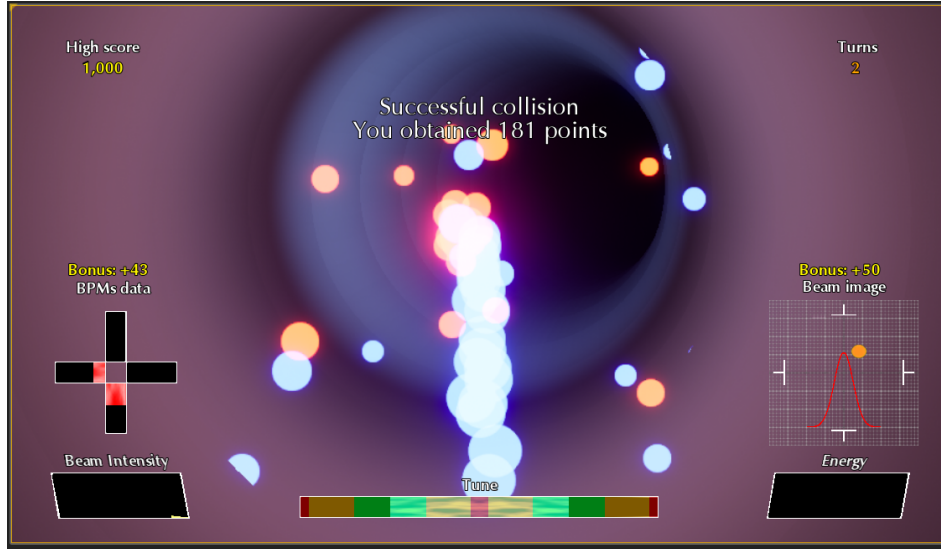


Figure 37: Successful collision achieved with very low beam intensity.

7.6 Blueprints

Blueprints could be considered as the most important elements of this game, as they gather and make use of everything that has been described up to now in this paper.

In CyberBeam there are five different Blueprints¹⁴:

- BP_SaveGame: It manages and saves the high score, so the value is not lost when closing the game.
- BP_MainMenuManager: This Blueprint manages everything that happens in the main menu. As soon as the main menu is launched, the camera is prepared, the scene is rendered with both particle systems, PS_BeamMenu1_2 and PS_BeamMenu2_2, a static mesh SM_Pipe is placed, the music is turned on, and the user interface is built. It also manages the player's input to start the game.

¹⁴ The explanation of each Blueprint has been considerably simplified, as these files have a lot of programming content behind them which are out of the scope of this paper.



Figure 38: BP_MainMenuManager showing its components, functions, variables, details, viewport, and event graph.

- BP_Accelerator: This Blueprint can be considered as the game world. It contains the static mesh for the LHC's pipe, the BPMs, the BTV, and the Wire Scanner. It also manages directly every event related to these elements, such as for example, a slow-motion effect when going through the BTV and Wire Scanner or a flashing effect when going through the BTV screen and the beam's picture is taken.

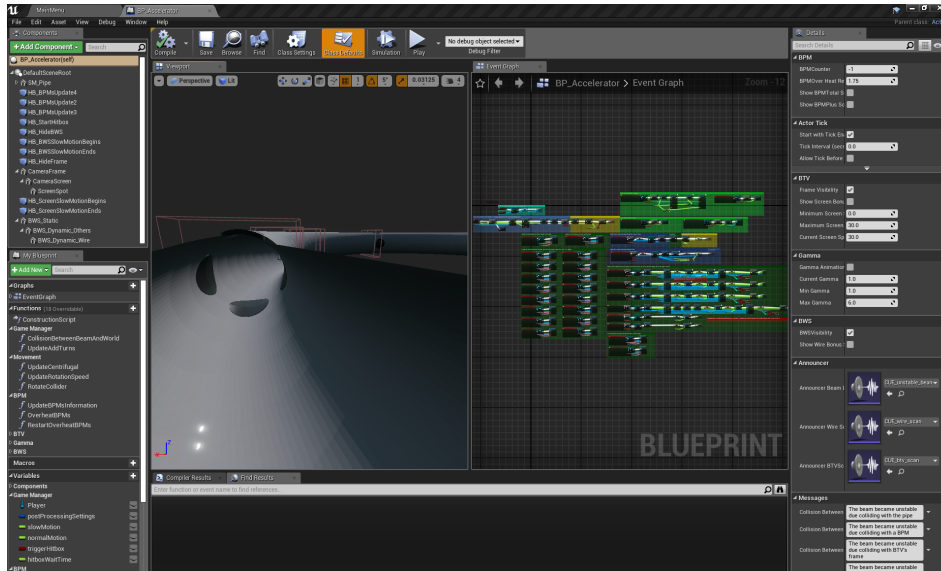


Figure 39: BP_Accelerator showing its components, functions, variables, details, viewport, and event graph.

- BP_Beam: This Blueprint represents the bunch that is controlled by the player all time during a game session. Therefore, all the inputs of the player are managed here, as other gameplay elements such as the tune or the score.

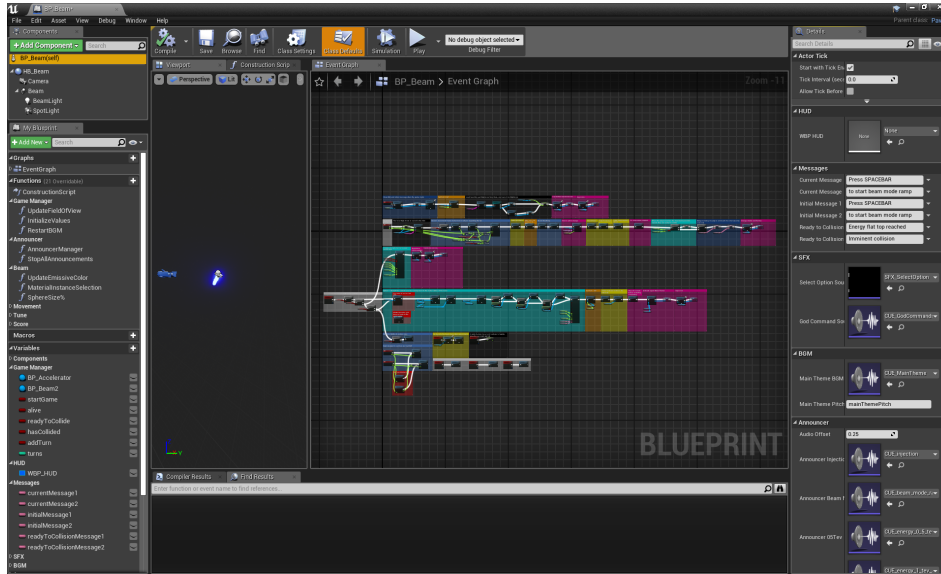


Figure 40: BP_Beam showing its components, functions, variables, details, viewport, and event graph.

- BP_Beam2: This last Blueprint represents the bunch that is going in the opposite direction of BP_Beam. It ensures players can collide with it once the flat top energy has been reached on the bunch they control, as well as the resulting output from such successful collision, like spawning the particle system PS_Collision.

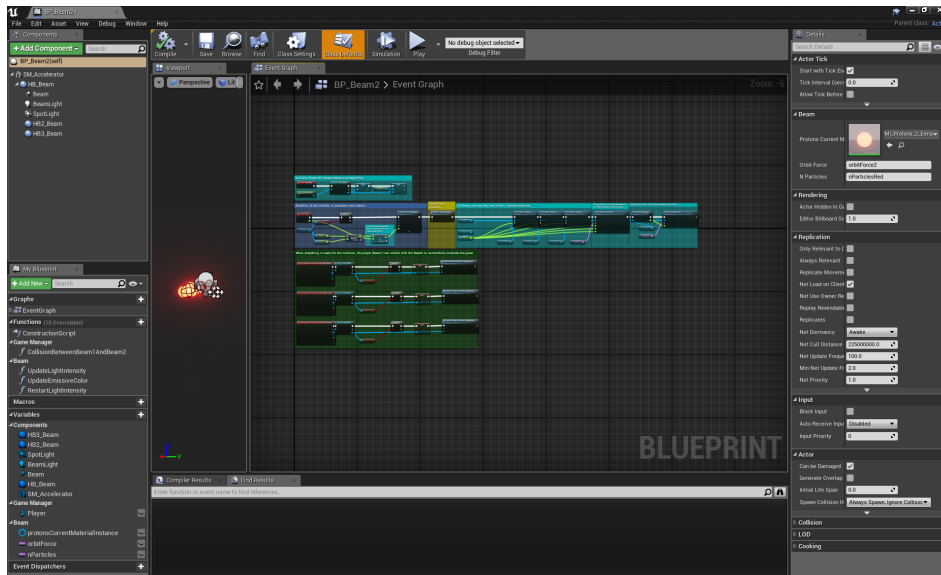


Figure 41: BP_Beam2 showing its components, functions, variables, details, viewport, and event graph.

7.7 Widget Blueprints

This type of Blueprints provides better tools and services to build user interfaces (UIs) or Head-Up Displays (HUDs) than the type of Blueprints defined in the previous section.

7. Game implementation CyberBeam

Widget Blueprints are divided in two main modes, the Blueprint Designer Mode and the Graph Editing Mode. The Blueprint Designer Mode features a canvas where several UI elements can be placed and animated freely. On the other hand, the Graph Editing Mode allows a Widget Blueprint to have all the functionality you can think of as any other regular programmed Blueprint. Furthermore, it can have its own methods and attributes which can be used together with the UI elements that have been placed, such as text boxes or progress bars.

For CyberBeam two Widget Blueprints have been programmed: One for the main menu and one for the game session.

- **WBP_mainMenu:** This Widget Blueprint represents the overlay of the screen when the player is in the main menu.

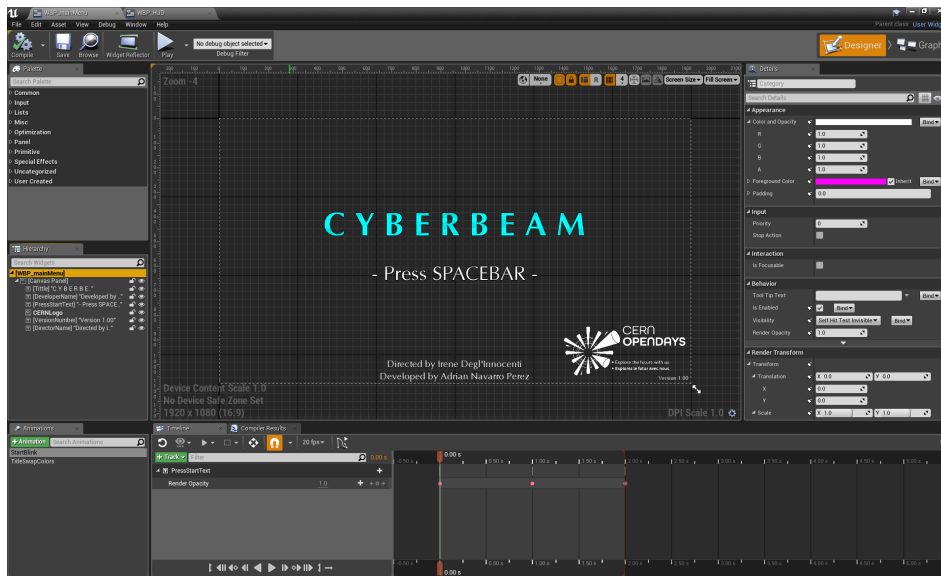


Figure 42: WBP_mainMenu Blueprint Designer Mode.

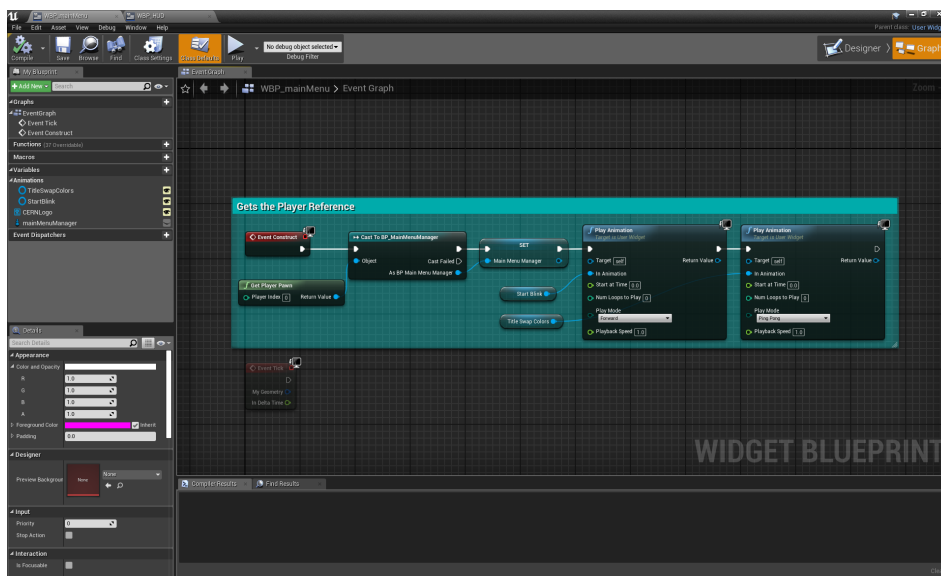


Figure 43: WBP_mainMenu Graph Editing Mode.

7. Game implementation CyberBeam

- **WBP_HUD:** This HUD is visible all the time while in the middle of a game session. It shows information about the score, turns, BPMs, BTV, Wire Scanner, their respective bonus points, beam intensity, tune, energy, and even messages which pop up on the screen as the game progresses.

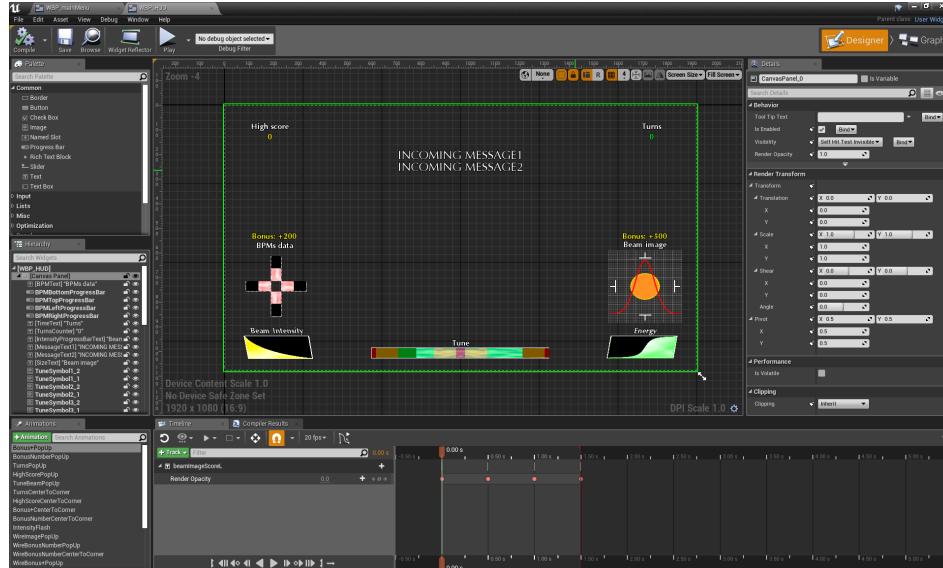


Figure 44: WBP_HUD Blueprint Designer Mode.

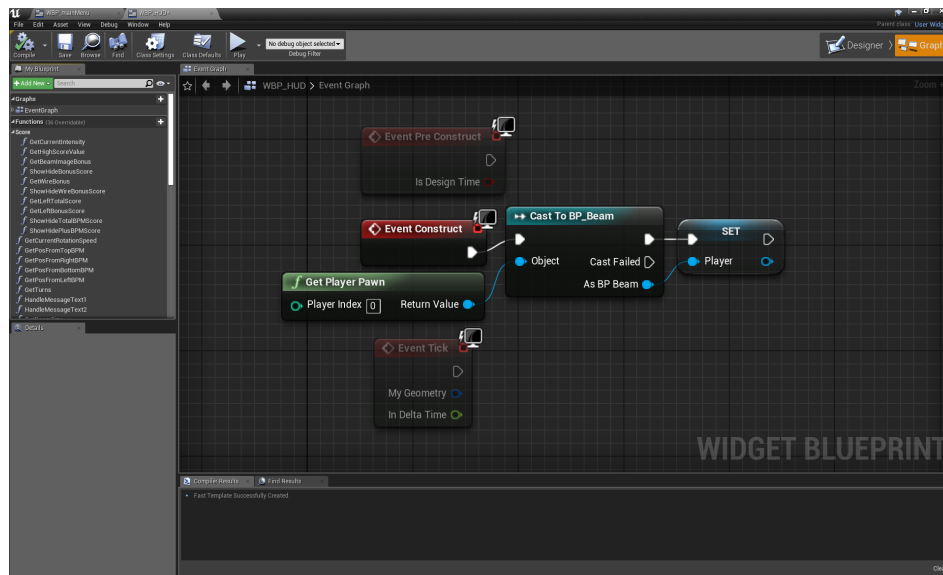


Figure 45: WBP_HUD Graph Editing Mode.

7.8 Maps.

Maps, often called levels, are the places where all the components described above are finally placed. For example, if a main character wants to be placed into a game, its specific Blueprint will be placed in one level, otherwise, the character would never appear.

CyberBeam features only two maps: MainMenu and SinglePlayer, also known as game session in this paper:



Figure 46: Initial state of the MainMenu map.

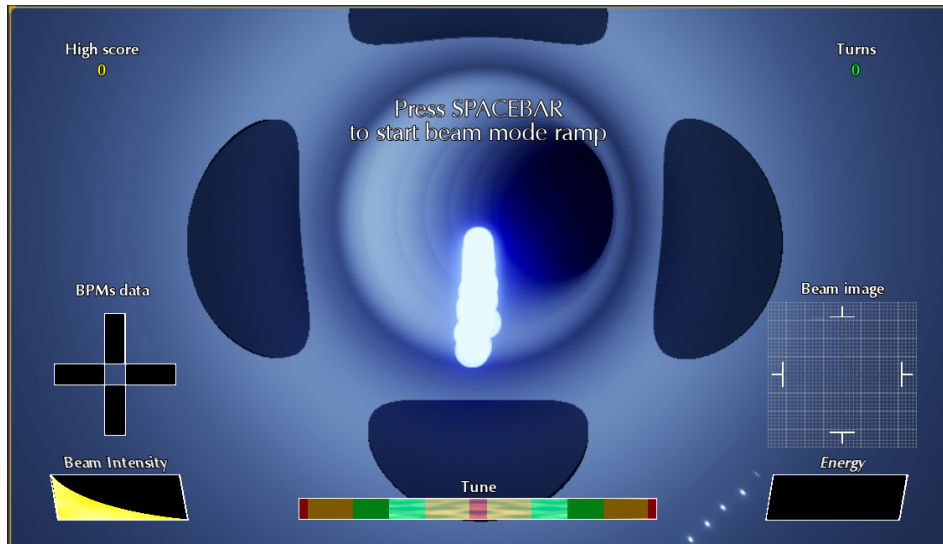


Figure 47: Initial state of the SinglePlayer map.

8. Conclusion

CyberBeam has been broadly described since its birth until its completion.

It has also been demonstrated that the goals which were set up before the start of the project have been accomplished: A serious game has been successfully developed for the CERN Open Days 2019 hold 14th and 15th September. However, at the time of writing the report, CyberBeam results remain uncertain as the CERN Open Days 2019 have not been held yet.

Annex 1: BPM Simulator

Introduction

As a parallel project to CyberBeam, a small BPM Simulator has been developed for the BPM stand at CERN Open Days 2019.

Development

Unlike the main game, in this little simulator, the attendees can change the position of the beam as well as its intensity in order to see how the signals from the four electrodes of a BPMs behave according to these parameters. The simulator shows also how the position information can be in principle calculated comparing the amplitude of the signals from two opposite electrodes (A and B) for each plane (horizontal and vertical). Thus, everyone will be able to see with their own eyes how complicated is to know where the beam is inside of the LHC when its intensity is very low.

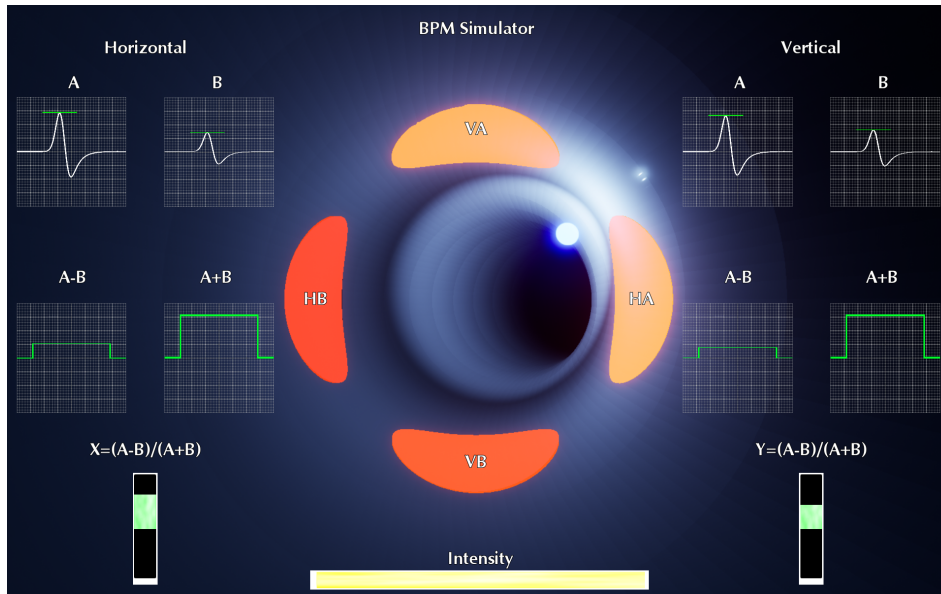


Figure 48: BPM Simulator with high intensity.

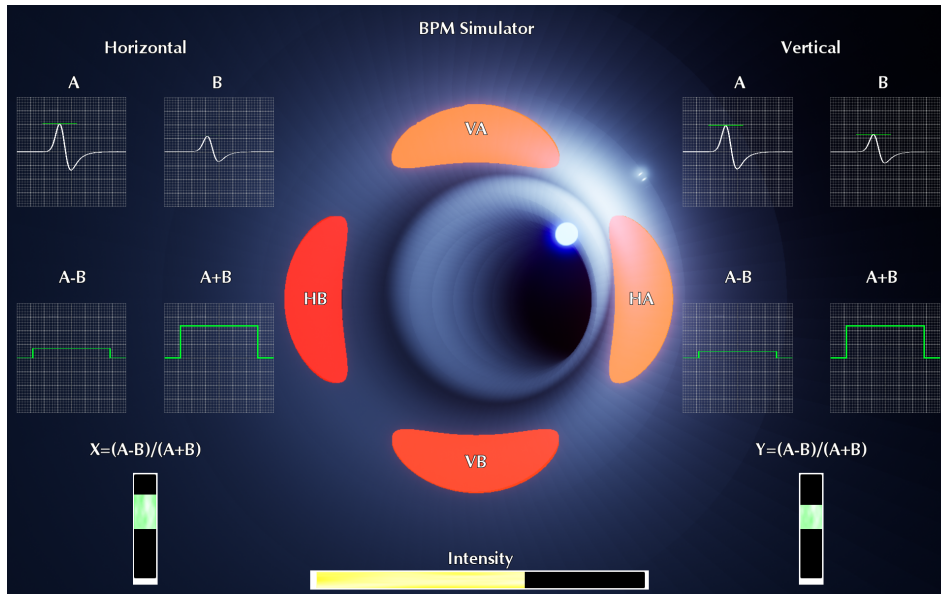


Figure 49: BPM Simulator with low intensity.

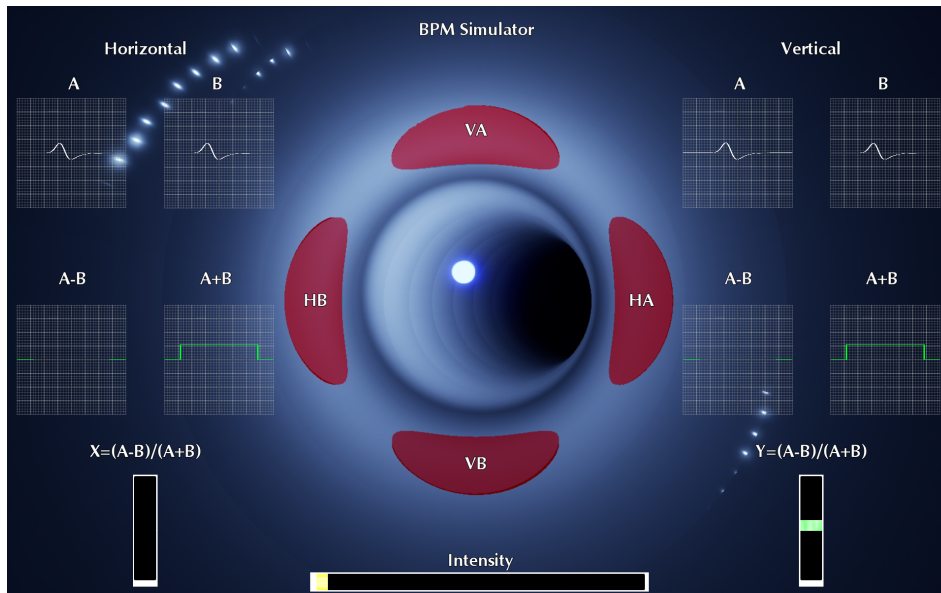


Figure 50: BPM Simulator with the lowest intensity.

Conclusion

In a similar way as the conclusion of CyberBeam, an interactive experience has been successfully created in order to let people learn by themselves through their interaction with this simulator.

Annex 2: Open Days 3D Tent

Introduction

As a small task right after CyberBeam and the BPM Simulator were finished, a 3D recreation of the CERN Open Days 2019 Beam Instrumentation tent has been done. The tent will be located at the Building 866's front parking lot inside CERN's Prévessin site.

In order to develop such 3D scaled environment, Twinmotion, has been used.

Twinmotion

Twinmotion is a source-available real-time immersive 3D architectural visualization software developed by Epic Games. It supports the production of high-quality images, panoramas, and standard or 360° VR videos. For architecture, construction, urban planning, and landscaping professionals, Twinmotion combines an icon-driven interface powered by Unreal Engine. [22]



Figure 51: Twinmotion's working environment.

Development

The development of the 3D environment was easier than one might expect.

First of all, it had to be received the 2D distribution of all its elements

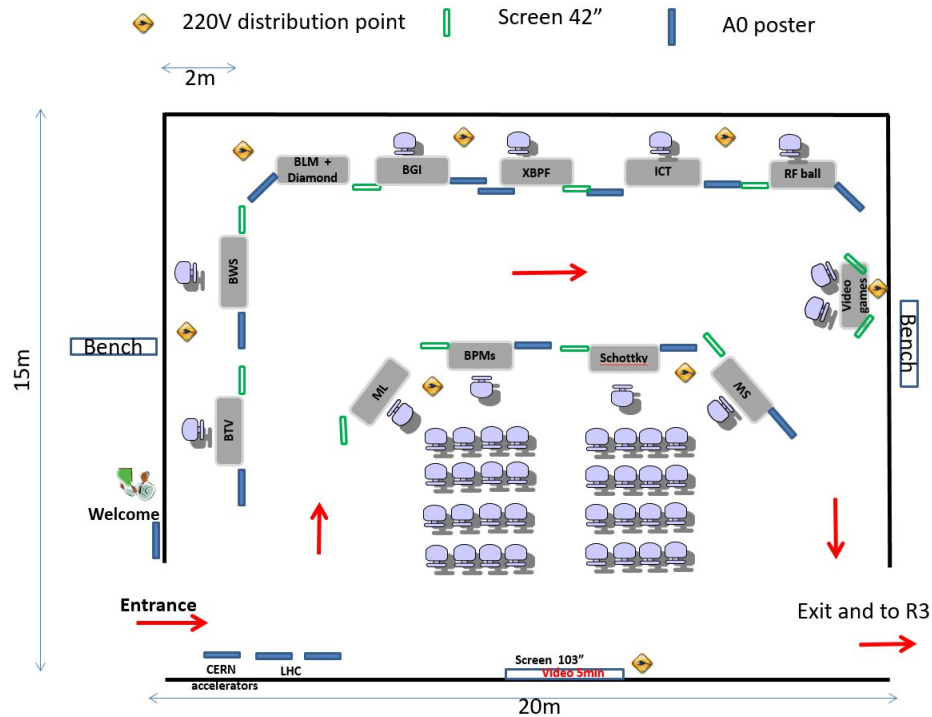


Figure 52: The original 2D distribution of the tent.

After identifying the different type of items, such as the tent itself, posters, tables, chairs... they had to be scaled according to the real measures they will have at the CERN Open Days 2019.

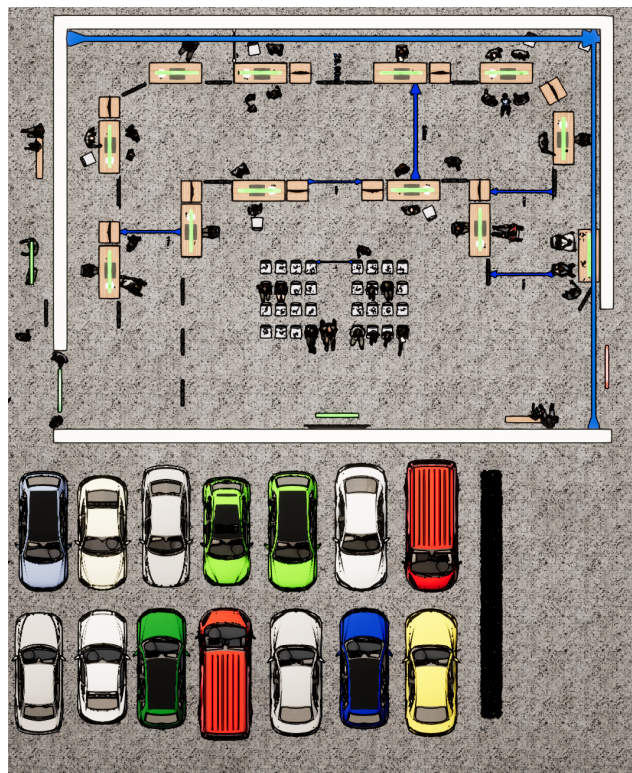


Figure 53: Top view of the 3D schematic tent.



40

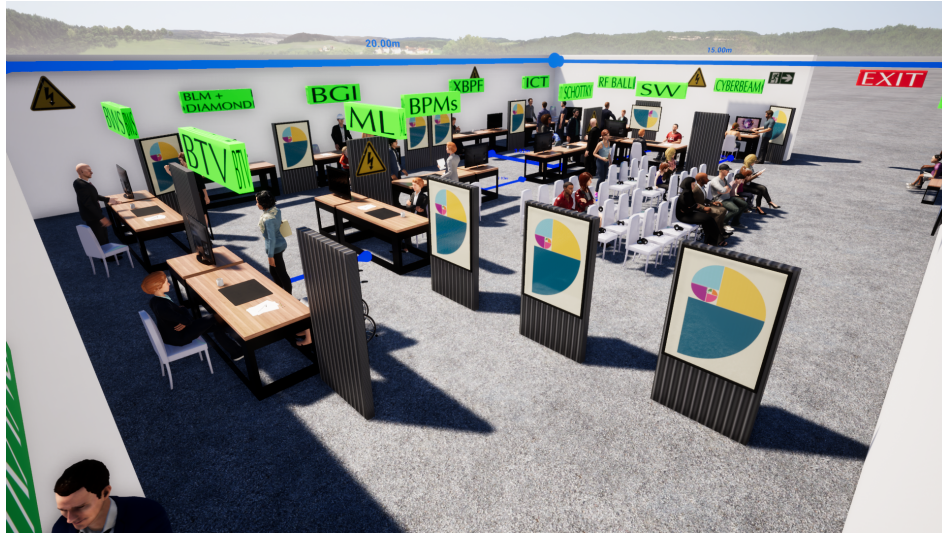


Figure 56: View from the tent's entrance.



Figure 57: Adrián explaining some visitors how to play to CyberBeam.

Conclusion

Thanks to this 3D layout, it was easier for everyone involved in the distribution of the CERN Open Days 2019 Beam Instrumentation tent how to organize all the items inside due the fact that a 3D environment provides more information than a 2D blueprint, if done properly.



Bibliography

- [1] CERN Beam Instrumentation Group, "Beam Instrumentation," CERN, [Online]. Available: <https://be-dep-bi.web.cern.ch/en>. [Accessed 12th July 2019].
- [2] Irene and I. Degl'Innocenti, "FPGA Design of the digital acquisition chain to test and implement ALPS, the new Beam Position Monitor for the Super Proton Synchrotron at CERN," 2016-2017. [Online]. Available: <https://cds.cern.ch/record/2291351/files/CERN-THESIS-2017-208.pdf>. [Accessed 26th August 2019].
- [3] Adrián Navarro Pérez, "What is a Game?," 20th November 2018. [Online]. Available: <https://adriannavarroperez.files.wordpress.com/2018/11/whatisagame.pdf>. [Accessed 16th July 2019].
- [4] E. Zimmerman and K. Salen, Rules of Play, 2003.
- [5] R. Hunicke, M. LeBlanc and R. Zubeck, "MDA: A Formal Approach to Game Design and Game Research," [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.4561&rep=rep1&type=pdf>. [Accessed 16th July 2019].
- [6] M. Sicart, "Defining Game Mechanics," 2008. [Online]. Available: <http://gamestudies.org/0802/articles/sicart>. [Accessed 16th July 2019].
- [7] S. Björk, S. Lundgren and J. Holopainen, "Game Design Patterns," 2004. [Online]. Available: <http://www.digra.org/wp-content/uploads/digital-library/05163.15303.pdf>. [Accessed 16th July 2019].
- [8] "Unreal Engine 4," Epic Games, [Online]. Available: <https://www.unrealengine.com/en-US/>. [Accessed 18th July 2019].
- [9] S. Horvat, "The imagination engine: Why next-gen videogames will rock your world," 17th May 2012. [Online]. Available: <https://www.wired.com/2012/05/ff-unreal4/>. [Accessed 26th August 2019].
- [10] "Blueprints Visual Scripting," Epic Games, [Online]. Available: <https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html>. [Accessed 18th July 2019].
- [11] "Blender," The Blender Foundation, [Online]. Available: <https://www.blender.org/>. [Accessed 18th July 2019].



- [12] "GIMP," The GIMP Team, [Online]. Available: <https://www.gimp.org/>. [Accessed 18th July 2019].
- [13] "GarageBand," Apple, [Online]. Available: <https://www.apple.com/lae/mac/garageband/>. [Accessed 18th July 2019].
- [14] "Audacity," Audacity, [Online]. Available: <https://www.audacityteam.org/>. [Accessed 18th July 2019].
- [15] "GitHub," GitHub, Inc., [Online]. Available: <https://github.com>. [Accessed 18th July 2019].
- [16] S. Page, "LHC Announcer," CERN, [Online]. Available: <https://announcer.web.cern.ch/announcer/>. [Accessed 22nd July 2019].
- [17] The Centre for Speech Technology Research, "The Festival Speech Synthesis System," [Online]. Available: www.cstr.ed.ac.uk/projects/festival/. [Accessed 22nd July 2019].
- [18] S. Heller, "One Hundred Years of Zapf," [Online]. Available: <https://www.printmag.com/steven-heller/one-hundred-years-hermann-zapf/>. [Accessed 22nd July 2019].
- [19] "CERN design guidelines - Typography," CERN, [Online]. Available: <https://design-guidelines.web.cern.ch/typography>. [Accessed 22nd July 2019].
- [20] "Materials," Epic Games, [Online]. Available: <https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/index.html>. [Accessed 22nd July 2019].
- [21] "Viewport Basics," Epic Games, [Online]. Available: <https://docs.unrealengine.com/en-US/Engine/UI/LevelEditor/Viewports/Basics/index.html>. [Accessed 23rd July 2019].
- [22] T. Tran, "Unreal Engine 4 Particle Systems Tutorial," raywenderlich.com, [Online]. Available: <https://www.raywenderlich.com/270-unreal-engine-4-particle-systems-tutorial#toc-anchor-002>. [Accessed 23rd July 2019].
- [23] "Twinmotion," Epic Games, [Online]. Available: <https://www.unrealengine.com/en-US/twinmotion>. [Accessed 14th August 2019].