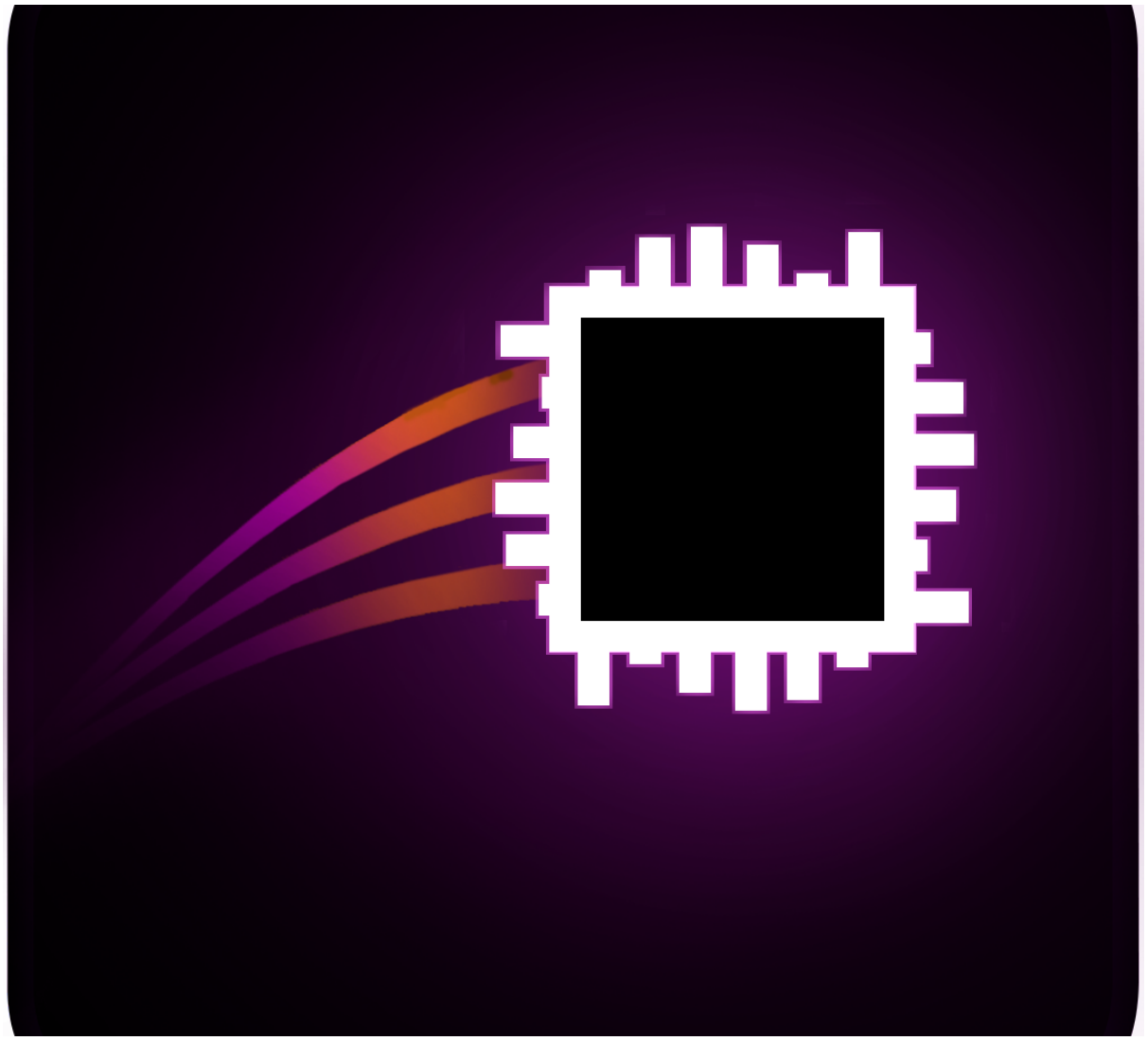
Neon Beats
Technical Design Document



Modification

Index

Overview

Game Summary

Neon Beats is a platformer game with a geometrical neon visual aspect with the particularity, that the Level Design responds to the music.
This game contains classic platformer mechanics (move, jump, wall-jump), and a lot different LD elements giving a contextual challenge.
Any player is supposed to be able to play one-handed and every sign should be comprehensible for color-blind people.

System Requirements

Computer running Windows 7+
Keyboard / Controller
Graphic Card 1080+

IOS 8 or Above
Android Platform

Technical Risks / Challenges

As in this game all the level design should be affected by music, beat detection is really important.
It's essential to be able to detect every different sample of musics quite easily and to choose on which one synchronize level design elements or even visual elements.

Being a platformer, Neon Beats' physics should not spoil the user experience.
The character controller is supposed to be easy and enjoyable to control especially if the game is supposed to be playable one-handed.

One of the constraint lead the game to be playable on two differents type of controller, in this idea the game shouldn't lose any ergonomics by playing with one or the other, which can lead to change some behaviour depending on the controller.

Finally, the game is mostly rendered technically using Unity components like line or trail renderer that will have to have specifics behaviours.

Tools

Engine Used

Unity 2018.2.6f1

3rd Party Tools / Assets

- Dotween
- Cinemachine
- InControl

Versioning

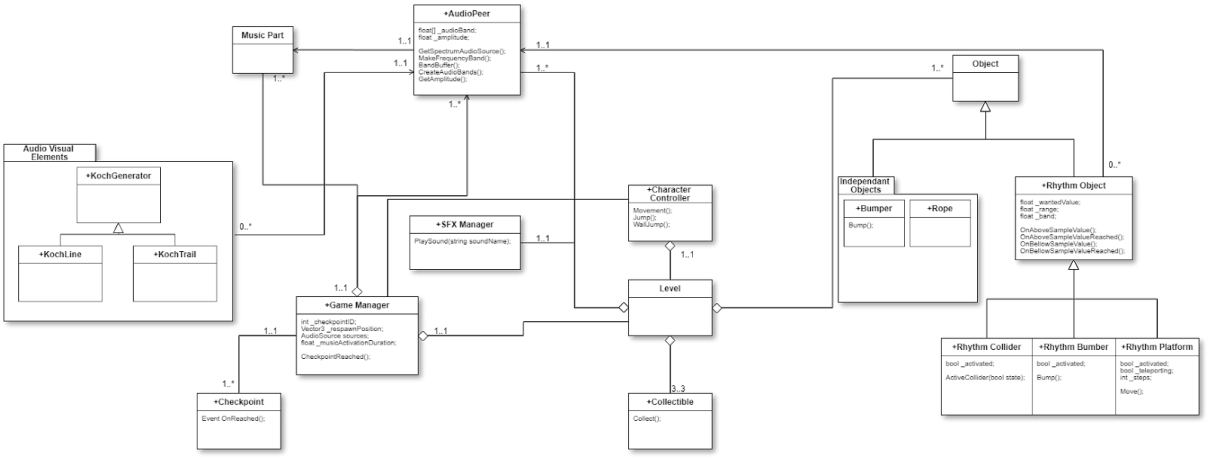
Using Unity Collaborate Service

Code Overview

General Architecture

The game is split in severals **Levels** that contains different elements.
First of all, several **Music** parts that will be analyzed by distinct **Audiopeers**.
The music frequency and amplitude are used by **Rhythm Object** to synchronize themselves on the music.
Still talking about sound, a **SFX Manager** is used to trigger all additional sound effects needed.
A **Game Manager** is there to progressively activate music parts depending on the player level completion checked by **Checkpoints**.
Some elements are not synchronized to the music but will allow the player to interact directly with themselves, they're called **Sound Independent Objects**.
Collectibles are also used to add some more challenge to the player, they act as an additional objective.
Finally, the **Character Controller** is there to let the player move and play all along the level.

Entities Diagram



Link : https://drive.google.com/file/d/1fpzOrNFicBt4WfSOtFgXukB6rlw1HQmM/view?usp=sharing

Entities Description

Character Controller : Contains all the main character mechanics (movement, jump, wall jump) and some physics adjustment.

Game Handler : Manage and save data related to player progression and settings.

Game Manager : Mainly used to control the character progression with **checkpoints**, and to trigger musics depending on it.

Checkpoints : Will change the respawn position of the player, and/or trigger musics when reached.

Timed Drop : Trigger audio loop on it's start time.

Audiopeer : Analyze the music and get usable values to sync elements with it.
It contains values like average amplitude, actual amplitude for each frequency bands.

Rhythm Object : Object which will use **Audiopeer**'s values to have a specific behaviour.
It will execute it when the average amplitude or a frequency band amplitude will reach a certain value.

Independent Object : Object which will have a specific behaviour not related to the music.

Input Manager : Manage player inputs.

SFX Manager : contains method to trigger Sound effects.

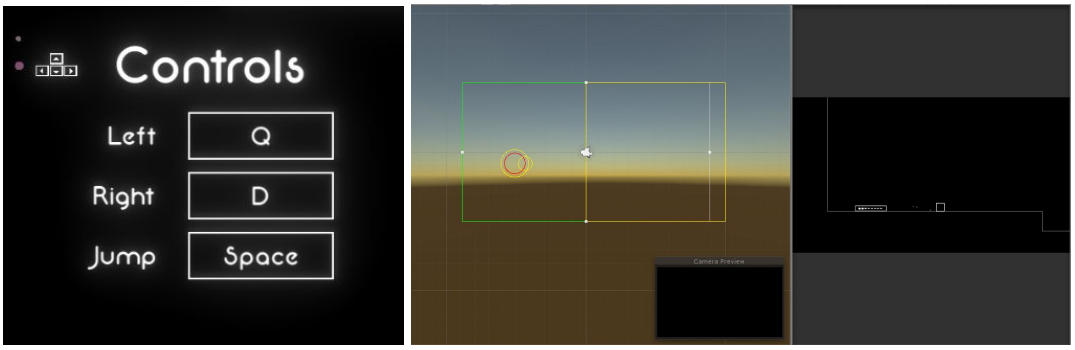
Collectible : additional objective for the player, will trigger an additional music part going with the level.

Koch Generator : generate complex shape based on two simple shapes.
This complex shape will be inherited by **Koch Line** to animate line renderers on music and by **Koch Trail** to animate trail renderers on music.

Technical Features

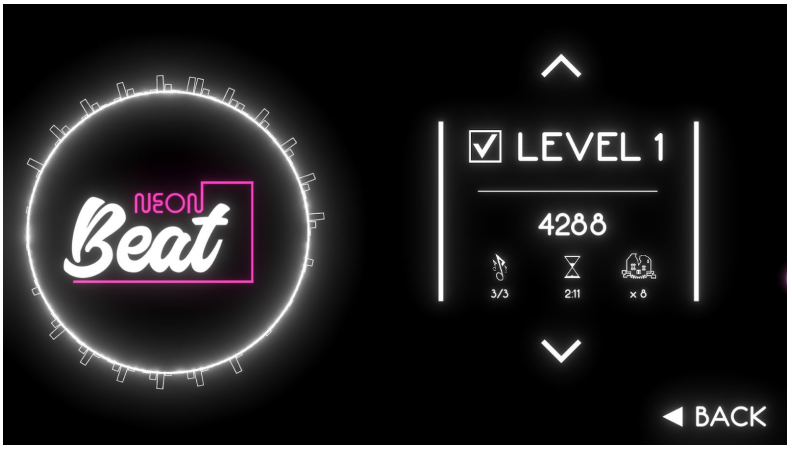
Inputs System

Inputs are managed using inControl to set several inputs to actions and rebinding.
For mobile inputs, a touch button control and touch button stick are used to handle jump and movement.



Save System

To save the player progression and settings, a Game Handler class manage all the data like level score, death, time, bindings etc …
This class convert a storage class into json file on save and read it on load.
In the case the file doesn't exist, it will create it and set all progression and settings value to default.



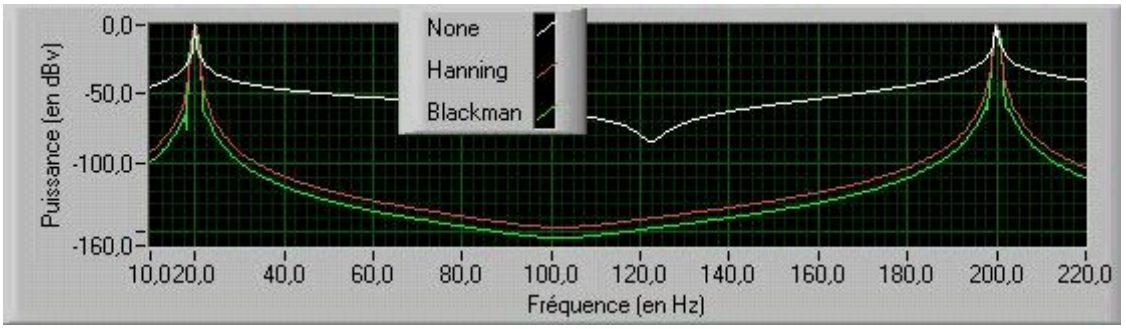
Animation

To have very specific behaviour, almost every animation in the game are made using Dotween.

Audio

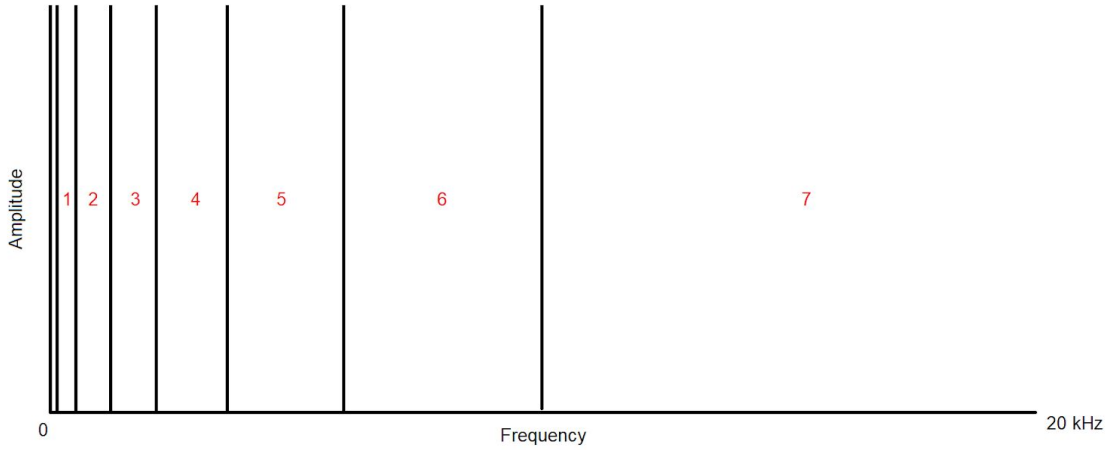
Music is a really strong pillar in this game, a lot of elements have to react to it, and so we used beat detections to trigger the animations of the differents elements.

First, the audio spectrum is analyzed using a Fast Fourier Transform of Blackman window type which is quite good to get precisely the amplitude due to a larger Roll-Of-Rate, and so be more precise on when a sound will play, it's commonly the most used for standard audio uses.



To get an easy amplitude value to use the actual amplitude is divided by the highest amplitude of the track, so we know the actual amplitude proportionally to the amplitude of the track.

Basically, the frequency human ears can hear is going from 0 to 20k Hz.
To ease the use of this values, we will convert the audio spectrum data from 20k Hz to eight frequency band.



To smooth the behaviour of frequency bands values, a buffer method is made to decrease progressively the frequency band value instead of getting it instantly to his target value.

Each music will be split in several part to activate them progressively along the player progression in the level, and it also allow us a more precise analysis of the music.

Rhythm elements behaviours are based on two values :

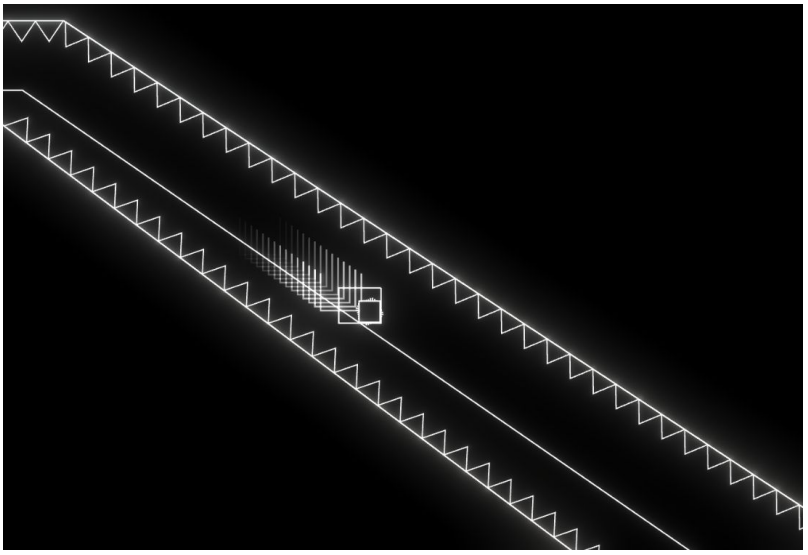
- the amplitude of a specific audio band
- the average amplitude

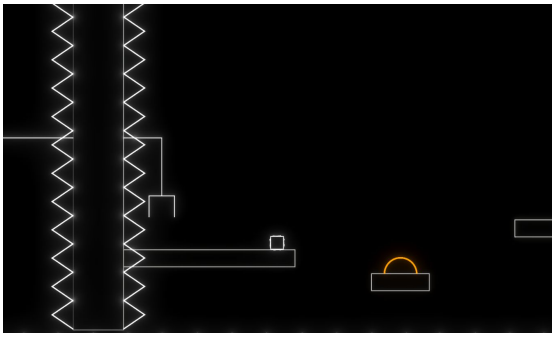
There is two kind of rhythm element, some will always follow the frequency of and audioband, and the others will wait until the frequency of the audioband reach a wanted value to activate themselves.

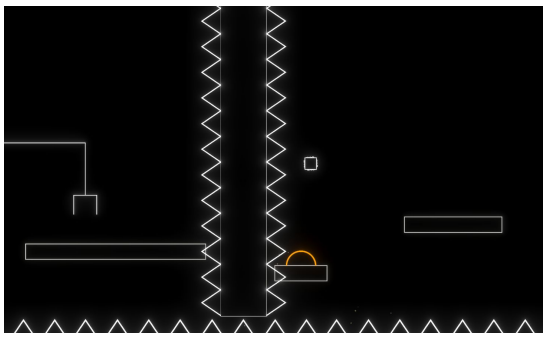
As the game music is split in multiple audio loop, we have to trigger them in specifics condition by maintaining them synchronized.
To do so, all Audiosources are a multiple of a define value, are playing on awake and looping with volume of 0.
Checkpoints are then used to set different audio loop volume to 0 or 1 to enable or disable them.

In some cases like music drop, we have to use Timed Drop script to trigger the audio loop at it's start time using another loop time to detect when it should start.

Finally, in situations in which the player need to be in rhythm with the music at a specific point, some prefabs are used like a wagon with a travel time depending on the music or a wall forcing the player to go further in the level.







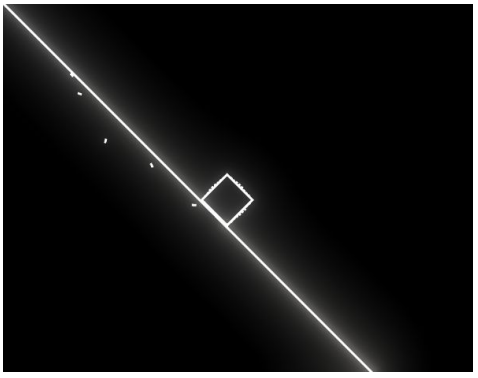
Physics

Has the jump is the main mechanic, it should be easy to use, responsive and have a good feeling.
When the character jump, the fall should be quite fast by still letting the player use the air control to replace himself.
To do that some tweaks were made on the character's rigidbody to increase the gravity scale, and also to his jump force.

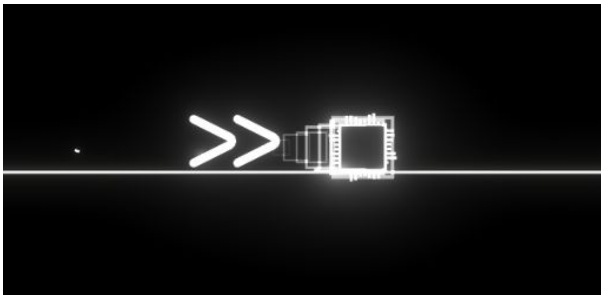
To check if the character is grounded, we used several raycast going down which will also give us the angle between the player and the ground.
6 frames of tolerance are provided to let the player jump on grounded exit.

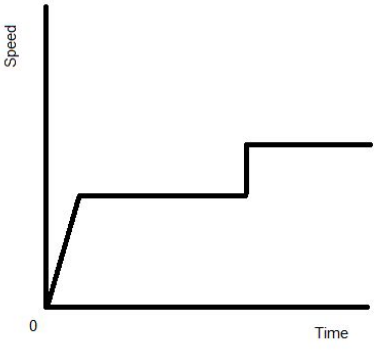
This angle is after used to snap the player rotation to the ground, this way the player rotation is simulated and the rigidbody rotation is frozen to have a more precise collision calculation.

To have a smooth feeling of movement in curves, the character movement direction is calculated not only using the player horizontal input, but also with a vector generated by the slope angle, in this idea, the character will always follow the ground direction.

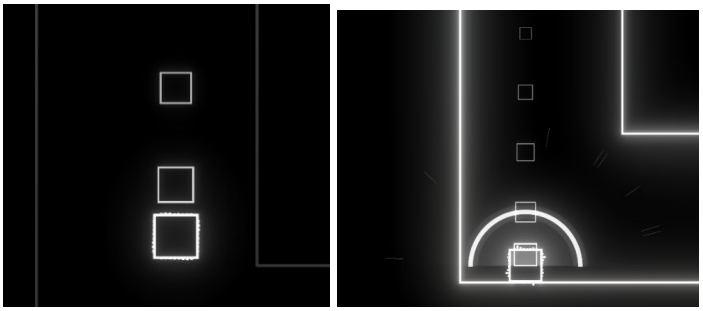


When the player touch a speed boost, his speed will change to it's high speed value.

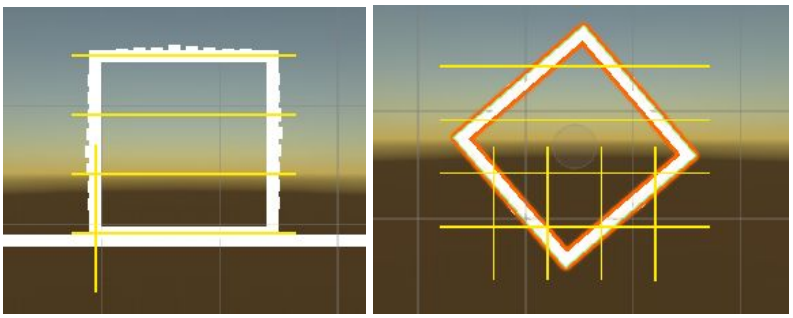




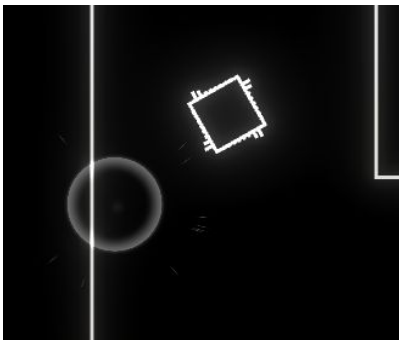
If the player reach a certains velocity on the y axis on his fall, it will make it make a “Super Landing” which will lock his movement input for .5s.



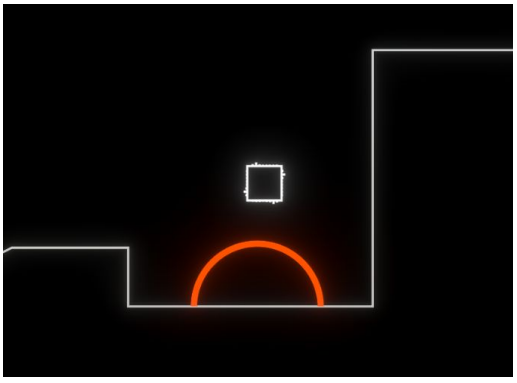
A raycast is also checking if the player is wall sliding, in that case a more important linear drag is applied and it allows the player to wall jump in the wall opposite direction.
The player jump animation being a z axis rotation, the distance of the raycast as to be recalculated depending on the player angle to fit the max cube length.



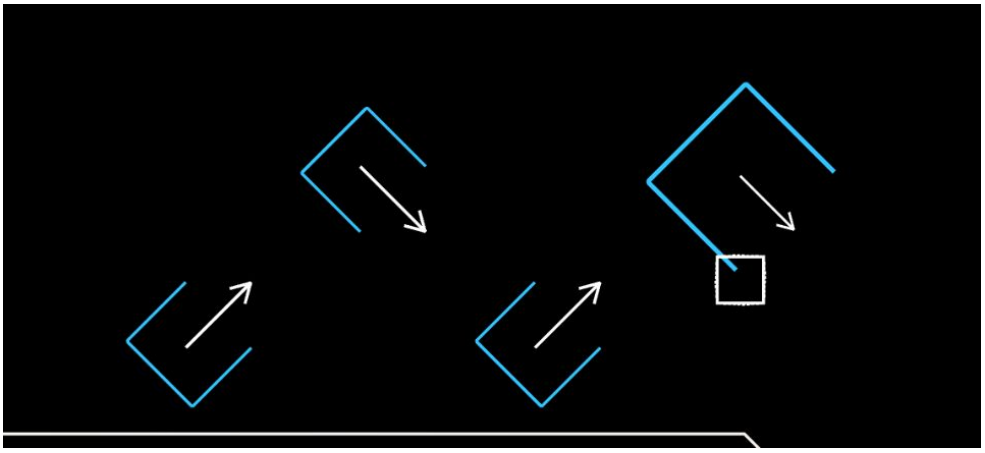
To facilitate the player wall jump, we decided that it should be possible to do it just by the jump input, that’s why the player is not forced to move in the wall direction to wall slide.
In the same way, a little input lock (0.14s) is used to prevent the player to not stick to the wrong direction input is pushed.
Both, using directional button or not is possible with the wall jump.



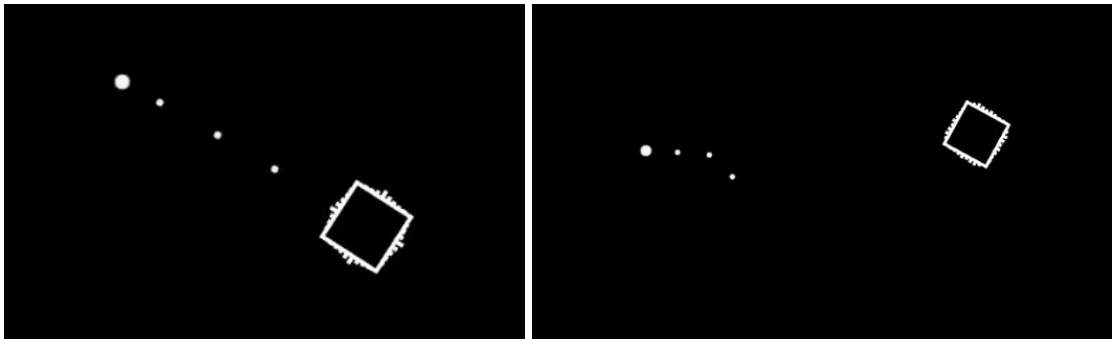
In the case where the player is jumping on the top half of a bumper, it will not add him a “realistic force” but only a force on the y axis to ease the control on his direction.



Expulser will always add a force on the player in the same direction depending on it’s rotation.
It can also add an input lock and nullify the gravity scale so the player will exactly follow the direction (mainly used if it leads to another expulser)

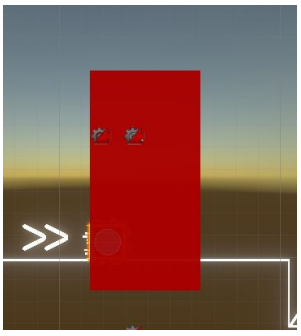


Rope are made using Hinge Joint 2D unity components, several of them are linked to give this swinging aspect.
The player can Roll Jump adding a force depending on his input and velocity.

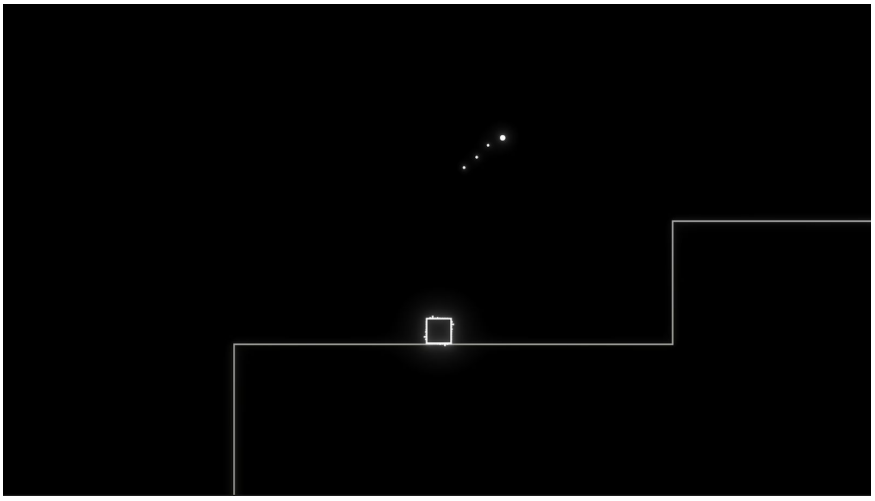


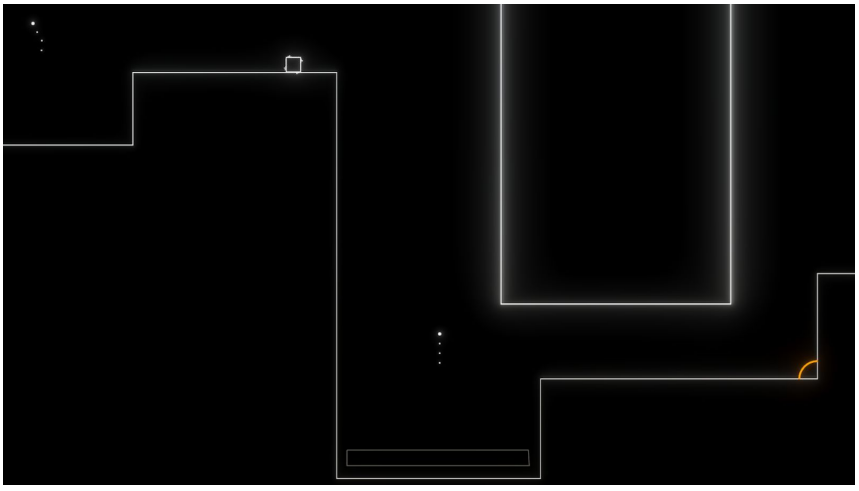
Camera

The normal player camera is following the player using a focus area, when the player collider reach it, the camera move with some LookAHead on x axis depending on a smooth time.



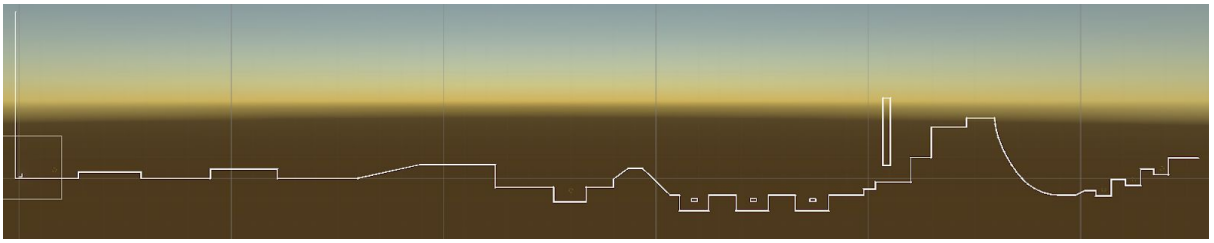
Using cinemachine, multiple virtual camera are used to change smoothly the camera behaviour, orthographic size and position.
A static class Camera Manager contains method to set the camera to the wanted camera with different blend setups.





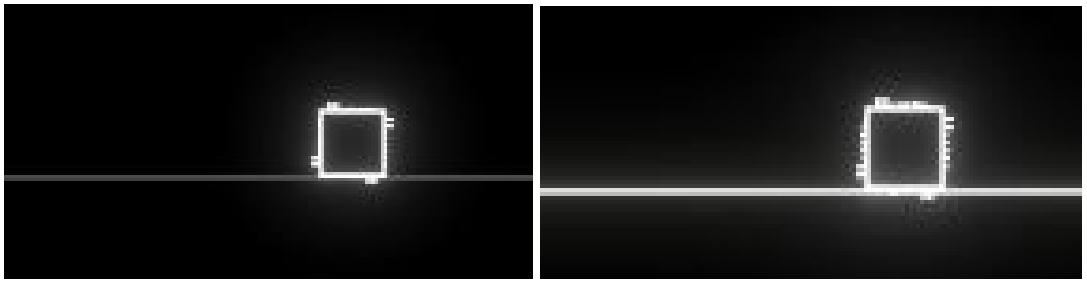
Graphics

As the game visual is mostly line, we decided to use the Line Renderer Unity Component a lot.
For example, the ground is drawn by a one long line renderer.



And Editor were made to ease the level building by placing points directly from the unity viewport.

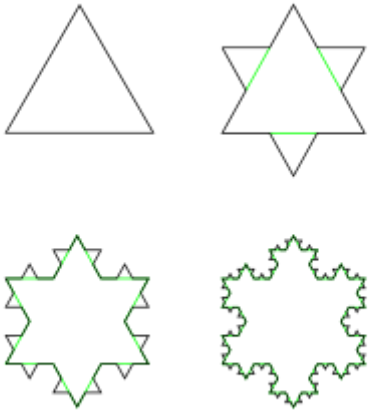
Moreover, to give a more important neon aspect getting along with the music, several line renderers material emission value is synchronized on the music.



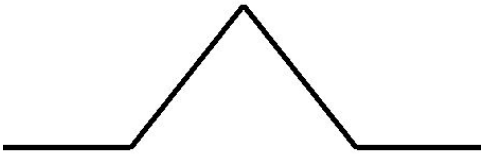
The square you can see above is the player character visual, also made by one line renderer and several sprites to make little bands reacting to the music.
Some particles system are also used for differents purpose like wall jump, or a trail when the player is moving.

Finally, a huge part of the technical visual work were on two audio visualisation components.
Both of them are using a rewriting system called L-System or LindenMeyer System basically used in plant or bacteria proliferation.
The basic principle of rewriting system is to achieve to get a complex object by editing the properties of a simple start object.
The most common example for graphical objects is the Koch Snowflakes created in 1904 by Helge Von Koch.

Basically this system is using 2 shapes, an initiator which will be our simple start polygon shape, and a generator which will replace each side of the initiator.

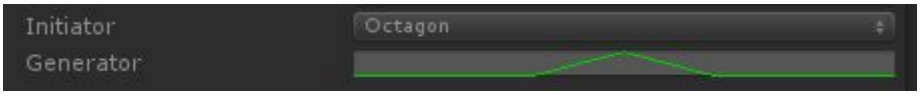


In this example, the initiator is an equilateral triangle, and the generator would look like this :

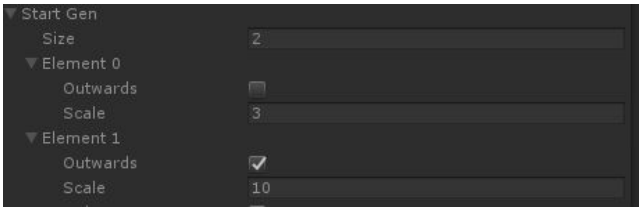


depending on the number of generation, we will achieve a more complex shape.

Both audio visualization components in the game will inherit from a **Koch Generator**, which will apply the rules we saw above on a chosen initiator shape based on another chosen generator shapes.

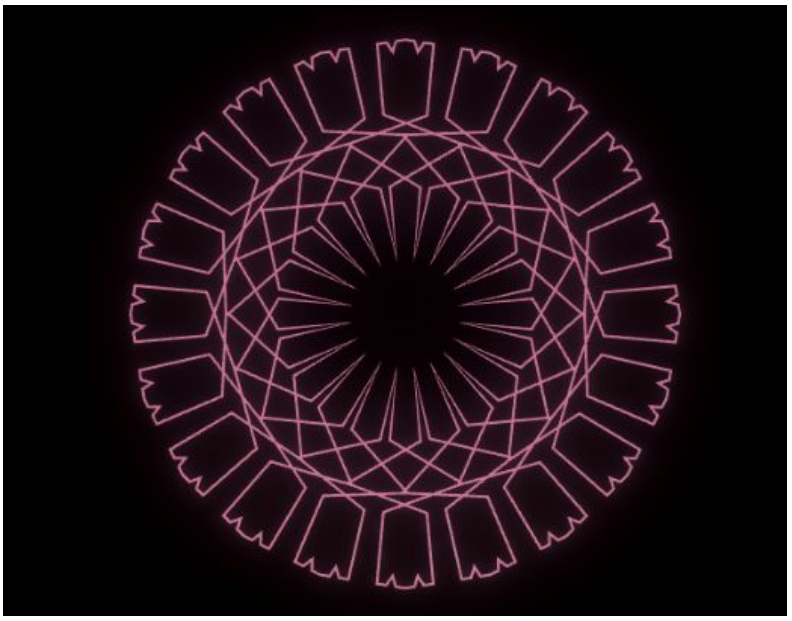


We can then choose the number of generation we want and assign them a direction(outwards, inwards) and a scale.



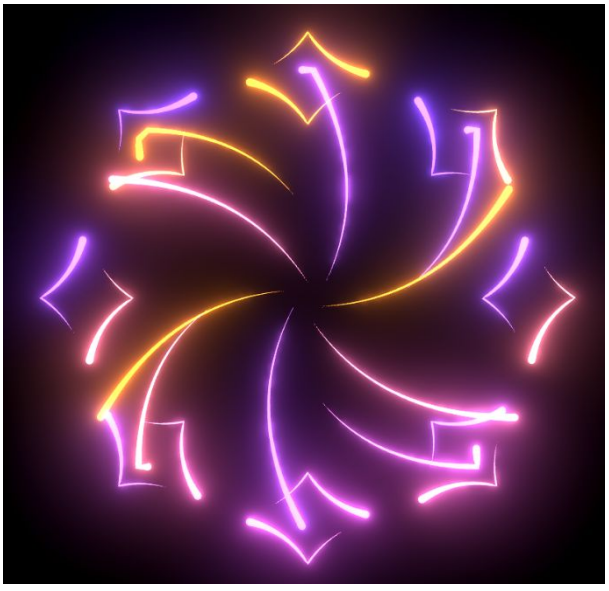
finally we can choose to use or not bezier curves to get less sharps shapes.

The first audio visualisation using this system are **Koch Lines.**
It's the application of the Koch Snowflakes on the Line Renderer components.



Koch Lines will generate all the necessary line renderer points and move them with a chosen direction and scale/strength depending on the amplitude of the chosen frequency band.
The material emissive color is also animated using the music.

The second are **Koch Trails**, working the same way but for trail renderer components.



This time the Koch Snowflakes will be used to generate the path of the different trail renderer generated. Their emission, lifetime and width are based on the amplitude of the chosen frequency band when the speed is based on the audio average amplitude.