DESARROLLO Y ENSAMBLE DE SCRIPTS EN EL LENGUAJE C# PARA VIDEOJUEGOS DE PLATAFORMAS INDEPENDIENTES CON ENFOQUES SOCIALES MEDIANTE EL MOTOR UNITY3D

DAVID ANTONIO BARRIOS ALVARADO

UNIVERSIDAD PONTIFICIA BOLIVARIANA
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ESCUELA DE INGENIERÍAS
BUCARAMANGA
2019

DESARROLLO Y ENSAMBLE DE SCRIPTS EN EL LENGUAJE C# PARA VIDEOJUEGOS DE PLATAFORMAS INDEPENDIENTES CON ENFOQUES SOCIALES MEDIANTE EL MOTOR UNITY3D

DAVID ANTONIO BARRIOS ALVARADO

Director
CARLOS HUMBERTO CARREÑO DÍAZ

Supervisor en la organización
ARMANDO CALDERÓN GRANADOS

UNIVERSIDAD PONTIFICIA BOLIVARIANA
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ESCUELA DE INGENIERÍAS
BUCARAMANGA
2019

DEDICATORIA

Dedicado a mis padres y mi familia que son la razón de mi vida.

AGRADECIMIENTOS

En el transcurso de mi carrera he crecido en mi ámbito profesional y como persona, esto ha sido gracias a todas aquellas personas que he podido conocer a través de la misma, quiero agradecer a mis profesores por transmitirme su conocimiento e impulsarme a seguir creciendo cada día, agradezco también a las personas con las que pude trabajar realizando mis horas de becario, agradezco a María Fernanda Mantilla Silva por acompañarme a través de todo este tiempo apoyándome incondicionalmente y siendo un pilar en mi vida, y por último, quiero darles las gracias a mi padres Antonio Barrios Pabón y Bilma Alvarado Villamizar que hicieron posible todo en mi vida, ellos son la razón de mi vida, así que mi más sincero gracias.

CONTENIDO

LISTA DE ILUSTRACIONES

RESUMEN GENERAL DE TRABAJO DE GRADO

TITULO:	DESARROLLO Y ENSAMBLE DE SCRIPTS EN EL LENGUAJE C# PARA VIDEOJUEGOS DE PLATAFORMAS INDEPENDIENTES CON ENFOQUES SOCIALES MEDIANTE EL MOTOR UNITY3D
AUTOR(ES):	DAVID ANTONIO BARRIOS ALVARADO
PROGRAMA:	Facultad de Ingeniería de Sistemas e Informática
DIRECTOR(A):	CARLOS HUMBERTO CARREÑO DÍAZ

RESUMEN

La industria de los videojuegos en la última década ha presentado un crecimiento gigante en el dinero que esta industria mueve y en el número de usuarios que son parte de ella, dado a su tamaño actual esta es consumida por gran cantidad de jóvenes y adultos, "*Below the game*" es una empresa desarrolladora de videojuegos que buscan crear obras únicas en los apartados visuales, en las mecánicas del juego y dejar una marca en sus jugadores, teniendo esto en cuenta empresa tiene varios proyectos en desarrollo, en esta práctica se trabajó en dos de ellos: Aqueducts y CrisTales, en el momento del inicio de la práctica estos proyectos ya se encontraban en desarrollo y en estos se realizaron actividades tales como desarrollo de scripts en lenguaje C# para control de cámara del personaje, control de la cámara en el mundo del juego, se realizaron actividades de ensamblaje de diversos *assets* en el motor grafico Unity 3d los cuales son proporcionados por el equipo de arte, se realizaban pruebas de rendimiento de las *build* para identificar los puntos a corregir o mejorar debido a que un punto muy importante en los videojuegos es la optimización de los mismos debido a que la mayoría de los proyectos en esta empresa está enfocado a las consolas y estas cuentan con un hardware limitado.

PALABRAS CLAVE: Ensamblaje, Unity 3D, scripts, videojuegos, desarrollo, *assets.*

GENERAL SUMMARY OF WORK OF GRADE

TITLE: DEVELOPMENT AND ASSEMBLY OF SCRIPTS IN THE C # LANGUAGE FOR INDEPENDENT PLATFORM VIDEO GAMES WITH SOCIAL APPROACHES THROUGH THE UNITY3D ENGINE

AUTHOR(S): DAVID ANTONIO BARRIOS ALVARADO

FACULTY: Facultad de Ingeniería de Sistemas e Informática

DIRECTOR: CARLOS HUMBERTO CARREÑO DÍAZ

ABSTRACT

The industry of videogames in the last decade has represented a big growth in the money that this industry move and in the number of users that are part of it, given its current size this is consumed by many young people and adults, "Below the game" is a video game company that seeks to create unique works in the visual sections, in the mechanics of the game and leave a mark on its players, taking this into account the company has several projects under development, in this practice I worked in two of them: Aqueducts and CrisTales, at the time of the beginning of the practice these projects were already in development, I carried out activities such as development of scripts in C # language for character camera control, camera control in the game world, assembly activities of various assets were carried out in the Unity 3d graphic engine which are provided by the art team, I performed performance tests of the builds to identify the points to correct or improve because a very important point in videogames is their optimization because most of the projects in this company are focused on consoles and these have limited hardware.

KEYWORDS: Assembly, Unity 3D, scripts, videogames, development.

1. INTRODUCCIÓN

En el presente documento se describirán las actividades realizadas como ensamblador en los proyectos de Voxels y Cristales de la empresa Santandereana Below the game (BTG), la cual es una empresa especializada en el desarrollo de videojuegos. Esta práctica se inició trabajando en el proyecto Voxels en cual se trabajó hasta su finalización y posterior a dicha finalización se pasó a trabajar en el proyecto de Cristales, en dichos proyectos se realizaron actividades tales como ensamblaje de escenarios y personajes, creación de sistemas de partículas, desarrollo de scripts, *testing*, creación de timeline, entre otras cosas. Para un mejor entendimiento de cómo se desarrollaron dichas actividades este documento se dividió en varios capítulos tales como:

- La descripción de la empresa Below The Game, para entender el enfoque de los videojuegos que son desarrollados por la misma.
- La justificación del desarrollo mismo de la práctica.
- Los objetivos que se cumplirán en el desarrollo de la práctica.
- Las actividades que se realizaran en la práctica.
- Un marco teórico donde se explicarán los conceptos más importantes para entender lo realizado en esta práctica empresarial, desde el entendimiento de que es un videojuego, como se pueden aplicar al ámbito educativo, que son los motores gráficos, entre otras cosas.
- Una breve descripción de la practica en la cual se detalla cómo se manejó el flujo del recibimiento, realización y entrega de las actividades dentro de la empresa, y como se comunicaban los diferentes grupos de trabajo en la misma.
- Un capítulo donde se describirá el análisis y diseño que se tuvieron que realizar para poder desarrollar cada una de las actividades.
- Un capítulo donde se explicará detalladamente como fue el desarrollo de cada una de las actividades mencionadas anteriormente.
- Un capítulo en el cual se mostrarán todos los resultados obtenidos en esta práctica empresarial.
- Las conclusiones a las cuales se llegaron a través del desarrollo de toda la práctica.

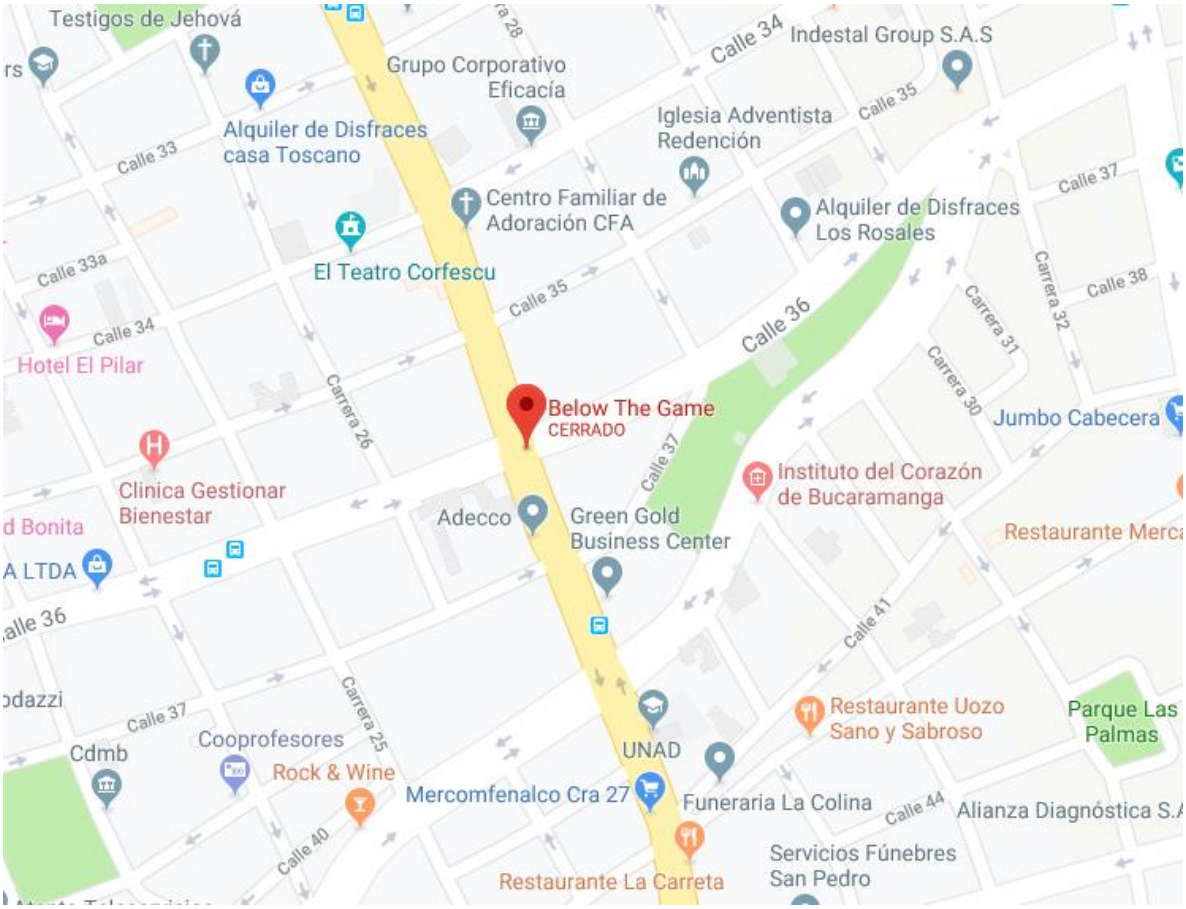
A continuación, se describirán detalladamente cada uno de los capítulos mencionados anteriormente.

2. GENERALIDADES DE LA EMPRESA

2.1. UBICACIÓN GEOGRÁFICA

La empresa de *Below The Game* -BTG- es un estudio independiente de desarrollo de videojuegos con múltiples nominaciones y premios nacionales e internacionales ubicada en la Carrera 27 N°. 36-14 en Bucaramanga, Santander. En la siguiente imagen se muestra la ubicación de la oficina por medio de la herramienta de Google Maps.

Ilustración 1. Mapa de ubicación física de Below the game



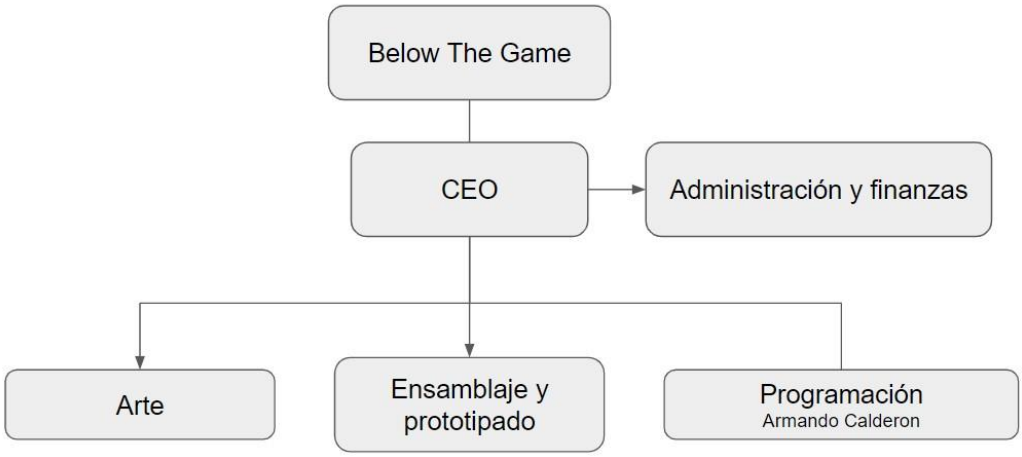
Fuente: Tomado de: https://www.google.com/maps/place/Palacio+Municipal/@6.9883208,-73.055204,15.75z/data=!4m12!1m6!3m5!1s0x8e6847afa54fe9f5:0xe884c2c4766d6412!2sPalacio+Municipal!8m2!3d6.9858291!4d-73.0501707!3m4!1s0x8e6847afa54fe9f5:0xe884c2c4766d6412!8m2!3d6.9858291!4d-73.0501707

2.2. BELOW THE GAME

Below The Game es un estudio de videojuegos independiente con sede en Colombia que crea videojuegos únicos, ya sea por la mecánica del juego o por el lado artístico de ellos. Su representante legal es **CARLOS ANDRÉS ROCHA SILVA**. Esta empresa desarrolla videojuegos tanto propios como encargados por empresas externas, siempre manejando altos estándares artísticos y visuales, como en las mecánicas dentro del juego y su rendimiento en diferentes plataformas.

2.3. ESTRUCTURA BELOW THE GAME

Ilustración 2. Estructura Below the game



Fuente: Autor

La empresa cuenta con diversos grupos de desarrollo encargados de distintas áreas como son: Grupo de desarrollo el cual cuenta con varios programadores los cuales son los encargados de la programación dentro del motor gráfico, el grupo de arte el cual es el encargado de generar todos los *assets*, animaciones, arte que serán usados dentro de los videojuegos, el grupo de ensamblaje, el cual es el encargado de juntar todo el arte y scripts creado dentro del motor gráfico, también cuenta con el grupo administrativo, donde se cuenta con el director administrativo, secretaria y el productor ejecutivo.

3. JUSTIFICACIÓN

La empresa en la cual se desarrolló la práctica empresarial que soporta este documento es un estudio de videojuegos de carácter independiente de origen bumangués llamado Below the game.

A través de la creación de videojuegos únicos ya sea por la mecánica del juego o por el lado artístico, esta empresa busca expandir su crecimiento y continuar con reconocimientos y nominaciones a nivel nacional e internacional, como por ejemplo ha trabajado en proyectos de SXSW Gamer's Voice o Nickelodeon Kids Choice Awards en Colombia. Sumado a esto, el estudio y desarrollo de los videojuegos hacen cada vez más que profesionales le apuesten a esta industria, generando así, la necesidad de desarrollar nuevos e innovadores productos como advergames, aplicaciones, animación, creación de experiencias VR, AR y entre otras.

Los proyectos en los cuales me desenvolví como profesional en ingeniería de sistemas en la práctica empresarial fueron Voxels y Cristales. El primero de género puzles tiene como fin la enseñanza de la programación básica, a través de reconocer y modificar códigos de programación.

El segundo proyecto llamado Cristales es un videojuego comercial, el cual pertenece al género RPG y será próximamente lanzado para distintas plataformas tales como PC, Ps4, Xbox one, Nintendo Switch, Steam.

Para la elaboración de proyectos como los mencionados anteriormente se requiere a un ingeniero de sistemas e informática para llevar a cabo tareas de programación, ensamblaje de escenarios, personajes y demás, desarrollo de sistemas de partículas, análisis de pruebas de desarrollo, entre otras. El aporte de un ingeniero de sistema e informática en esta área es brindar por medio de un proceso de ingeniería soluciones a situaciones o problemas que requieran análisis y desarrollo. Asimismo, haber tenido la oportunidad de hacer mi práctica empresarial en la empresa Below the Game, siendo esta mi primera experiencia laboral en el campo de los videojuegos tuvo un valor significativo tanto en mi crecimiento profesional como en mi proyección laboral encaminada hacia la creación y desarrollo de videojuegos.

4. OBJETIVO DE LA PRÁCTICA

Desarrollar scripts en el lenguaje C# y ensamblar escenas y niveles para el desarrollo de videojuegos de plataformas independientes con enfoques sociales mediante el motor UNITY3D.

5. ACTIVIDADES

1. Programación de software en lenguajes C#, C++ y HTML5 para su implementación en el proyecto.
2. Desarrollo de scripts de programación compatible con motores de desarrollo de videojuegos.
3. Importación y configuración de *scripts* y *assets* dentro del motor de desarrollo para el proyecto.
4. Construcción y estructuración de escenas y ambientes del videojuego dentro del motor de desarrollo.
5. Integración de scripts de código con *assets* visuales dentro del motor de desarrollo.
6. Aseguramiento de la calidad y pruebas de las versiones generadas para la detección de bugs/errores de programación dentro del proyecto.

6. MARCO TEÓRICO

6.1. VIDEOJUEGOS

Los videojuegos son aplicaciones de software los cuales tienen como característica principal que el usuario puede interactuar directamente con el mismo a través de dispositivos de entrada. [1] Para dicha interacción los dispositivos en los cuales se ejecutan dichos videojuegos cuentan controles o mandos, que en el caso del computador sería el teclado y el ratón, en el caso de las consolas sería el mando o control y para los dispositivos móviles como los smartphones se interactúa directamente con la pantalla del mismo dispositivo. Los primeros juegos informáticos solían hacer uso de un teclado para llevar a cabo la interacción, o bien requerían que el usuario adquiriera un joystick con un botón como mínimo, como en la consola del Atari 2600 [2]

6.2. VIDEOJUEGOS EDUCATIVOS

Un videojuego educativo es un software de entretenimiento por medio del cual se puede aprender uno o varios temas en específico. Este tipo de juegos contienen elementos propios de un videojuego común que lo hace divertido y entretenido, y un conjunto de elementos educativos que le permitan al jugador adquirir conocimientos de un tema en específico de forma implícita, es decir, que los jugadores no se percaten directamente de la adquisición de dicho conocimiento, pero que sea a su vez lo suficientemente claro lo que se pretende enseñar a través del videojuego. [3] [4]

6.3. MOTORES GRÁFICOS

Un motor gráfico es una serie de rutinas de programación que permiten el diseño, el desarrollo y el funcionamiento de un videojuego. Los motores gráficos actuales poseen diversas funcionalidades tales como la creación de modelados 3D, sistemas de físicas realistas, gestión de los recursos, animación tanto 2D como 3D, creación de scripts en diversos lenguajes de programación como C# o JavaScript, manejo de redes, manejo de sonidos, entre muchas otras cosas. Dentro de los motores gráficos se combinan todo el arte, programación, diseño, animaciones y todo lo necesario para la creación de un videojuego. [5]

6.4. MODELADO 3D

El modelado 3D es una representación matemática de cualquier objeto tridimensional creado a través de un software como Cinema 4D, AutoDesk, Maya o Zbrush los cuales son los softwares más populares para su creación actualmente. Se puede

visualizar como una imagen bidimensional mediante un proceso de renderizado 3D. Estos modelos 3D son usados en los videojuegos para representar tanto a personajes como el entorno, alcanzando gran nivel de detalle en los videojuegos AAA actuales. [6]

6.5. SISTEMA DE PARTÍCULAS

En un videojuego 3D los personajes, objetos y escenarios son representados con modelados 3D y en los videojuegos 2D se representan haciendo uso de *sprites*. Sin embargo, dentro los videojuegos también se tienen otras cosas u objetos como líquidos en movimiento, humo, nubes, explosiones, hechizos, etc., los cuales son difíciles de representar a través de modelados 3D y *sprites*, por lo tanto, se crearon los sistemas de partículas los cuales pueden representar mejor dichas cosas. [7]

6.6. ASSETS

Un *asset* es una representación de cualquier *item,* objeto o componente que puede ser utilizado en un videojuego o proyecto. Un *asset* podría venir de un archivo creado afuera de Unity, tal como un modelo 3D, un archivo de audio, una imagen, o cualquiera de los otros tipos de archivos que Unity permite. [8]

6.7. SPRITES

Un *sprite* es una imagen usada para representar un ente gráficamente. Dicho ente no tiene estar representado gráficamente por un único *sprite* sino que puede estar dividido en varios diferentes, por ejemplo un personaje como Mario Bros puede estar dividido en varios *sprites* donde uno de ellos es sus brazos, otro sus piernas y otro su cuerpo y cabeza. [9] [10]

6.8. UNITY 3D

Unity 3d es un motor grafico utilizado para la creación de videojuegos 2D, 3D, VR, entre otros, para plataformas como Windows, Linux, Mac OS, Web, Android, IOS, PlayStation, Xbox, Nintendo. Este motor grafico es uno de los más populares entre los desarrolladores indies y desarrolladores de juegos tripe A. [11]

6.9. LENGUAJE C#

El C# es un lenguaje de programación orientado a objetos y tipado, fue desarrollado por Microsoft y hace parte de la plataforma de .NET. C# tiene sus orígenes en el lenguaje C. [12]

7. DESCRIPCIÓN DE LA PRACTICA

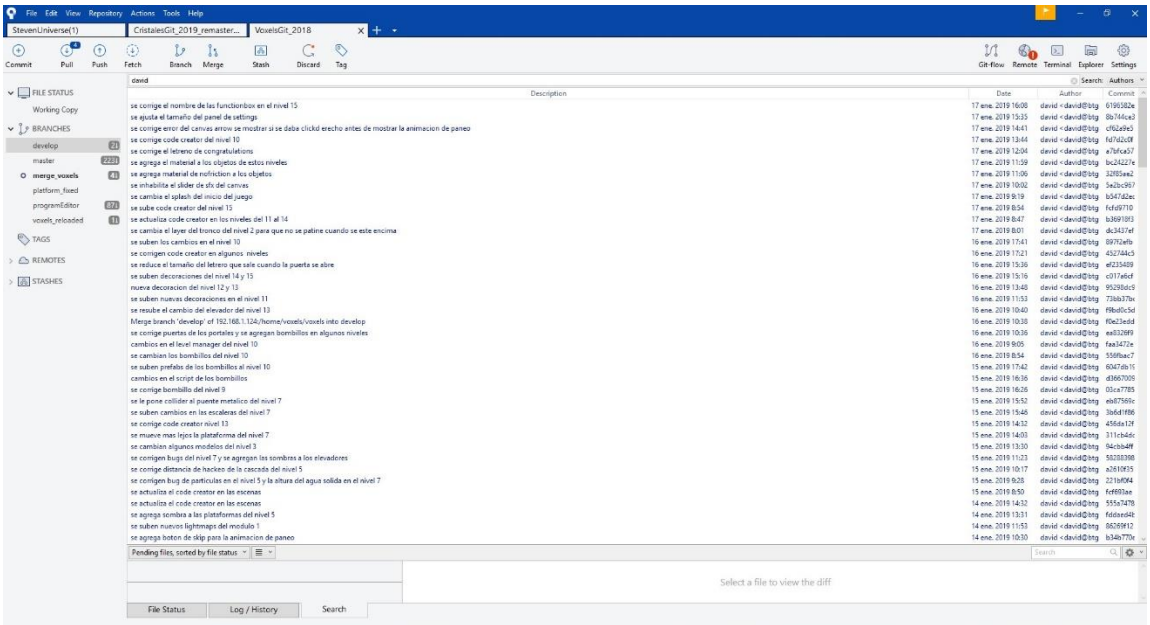
A través del tiempo en el que se realizó la práctica se participó en dos proyectos de la empresa: aqueducts(Voxels) y Cristales, para los cuales se manejaron programas y servicios web para el manejo de versiones del proyecto, asignación y cumplimiento de tareas, compartir documentos relacionados con las tareas a realizar, comunicación entre el equipo de trabajo, entre otras cosas, los programas usados para dichas actividades dentro de la empresa fueron los siguientes:

7.1. MANEJO DE VERSIONES

Debido a que el equipo de trabajo constaba de varias personas se hace necesario el uso de una herramienta como *git* para mantener el control de los cambios realizados y quien los realizo. Para un manejo más visual y organizado del *git* se hizo uso del GUI *sourceTree* el cual permite visualizar cuales son las diferencias entre las versiones de los archivos, ver quien realiza cada *commit*, crear ramas adicionales, entre otras cosas.

En los dos proyectos se manejaron la misma estructura del proyecto en el repositorio de git y las mismas ramas principales, las cuales son la rama de *master* en el cual se maneja siempre una versión estable de la cual se creaba un *build* para las entregas periódicas y la rama de *develop* en la cual todos los integrantes del grupo de desarrollo subían sus avances. Las ramas manejadas en el proyecto se pueden observar en la siguiente imagen:

Ilustración 3. Repositorio de git



Fuente: Autor

El grupo de arte no subían directamente los archivos al mismo *git*, si no que manejaban un repositorio SVN el cual les permitía mantener un control de los archivos de forma sencilla, en este repositorio se subía exclusivamente las imágenes, gifs, *sprites*, archivos de las animaciones y demás archivos gráficos usados en los proyectos.

7.2. ASIGNACIÓN DE TAREAS

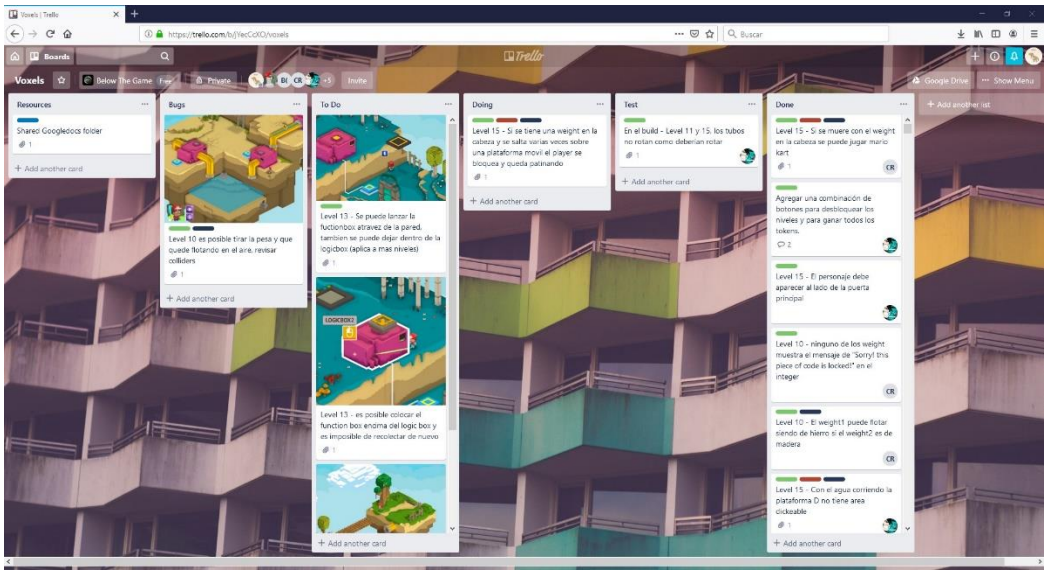
Para la organización de las tareas dentro de los proyectos se hizo uso de la herramienta Trello la cual consta de un tablero donde se tienen diferentes columnas en las cuales se ponen las actividades a realizar como tarjetas, esta herramienta permite asignar dichas tarjetas a uno o varios miembros del equipo de trabajo para así mantener control de quien realizo dicha tarea y en qué fase de desarrollo se encuentra la misma.

Las columnas dentro del tablero de Trello permiten saber en qué etapa se encuentra la tarea o actividad, las que se manejaron dentro los proyectos fueron:

- la columna de *resources* en la cual se subían archivos o textos que eran de uso general o guías para realizar ciertos procesos de forma adecuada, ahí también se encontraba la descripción general de la estructura que se debía manejar.
- La columna *bugs* en la que se encontraban las tareas que creaba el productor en la cual se reportaban comportamientos extraños o errores dentro del videojuego que no impedían la continuidad del juego pero que se debían tener en cuenta para corregirlos.
- La columna *to do* en la cual se ponían todas las nuevas tareas que se debían hacer en el proyecto como corregir errores, el ensamblaje de nuevas escenas, el desarrollo de scripts, etc.
- La columna *doing* en la cual se ponen las tarjetas que ya tenían asignadas a un desarrollador y se encuentra trabajando en dicha tarea.
- La columna Test a la cual se movían todas las tarjetas las cuales el desarrollador ya había finalizado y quedaban a espera para ser testeadas, el productor es el encargado de revisar dichas tarjetas y verificar que se desarrollaron correctamente o deben ser devueltas.
- La columna *Closed* en las cuales se encontraban todas las tarjetas que ya se han desarrollado y fueron testeadas y no presentaron errores.

La estructura del Trello del proyecto de Aqueducts se puede ver a continuación:

Ilustración 4. Estructura del tablero de Trello

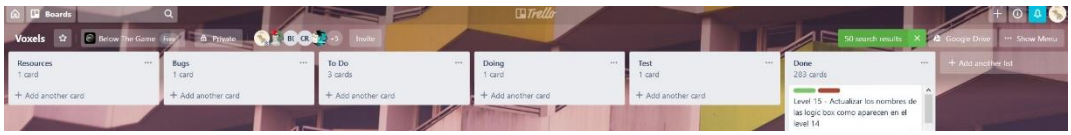


Fuente: Autor

El proceso para seguir desde la creación de la tarjeta hasta la finalización de la misma es el siguiente:

Los encargados de la creación de tarjetas son el productor y el *game designer*, los cuales deciden que actividades son necesarias para el avance del proyecto, las tareas nuevas siempre son puestas en la columna de *To do* o bugs dependiendo de qué tipo de tarea es, después de creada la tarjeta y puesta en la columna correspondiente el productor o el *game designer* añadían una etiqueta la cual especificaba que tipo de tarea era ya se de programación, ensamblaje, arte, después dichas tareas eran asignadas a un desarrollador el cual debían moverla a la columna de *Doing* cuando empezara a desarrollarla, posteriormente cuando se terminaba de desarrollar dicha tarea el desarrollador movía la tarjeta la columna de Test, el encargado de revisar dichas tarjetas que se encontraban en Test era el productor, el cual testeaba lo desarrollado para comprobar si cumplía con lo requerido en la tarea, si se encontraban errores la tarjeta era movida a la columna de *To do* con una etiqueta de volver a revisar para que el desarrollador pudiera revisar de nuevo la tarjeta y corregir los errores, cuando terminaba de corregir dichos errores se devolvía la tarjeta a la columna de *test*, si no se encontraban errores el productor movía la tarjeta a la columna de *closed*.

Ilustración 5. Columnas de Trello



Fuente: Autor

7.3. COMPARTIR DOCUMENTOS

En la empresa muchas veces se hacía necesario compartir documentos tales como la estructura de los diálogos que tendrían los personajes dentro del juego, o las secuencias que se manejarían en las escenas, el orden de la historia, libros sobre temas que se necesitan en el desarrollo, etc, para compartir dichos documentos entre los integrantes del proyecto se hacía uso de una carpeta compartida en Google Drive donde cada uno podía ingresar descargar los archivos o editarlos allí mismo.

7.4. COMUNICACIÓN ENTRE EL GRUPO DE TRABAJO

Debido a que era un grupo de trabajo de varias personas en un espacio cerrado, se podían presentar casos en los que cada persona estuviera hablando de temas diferentes y se perdiera el orden dentro de la oficina, por lo tanto la mayoría de conversaciones que no fueran muy importantes se manejaban a través de *Slack* la cual es una herramienta que permite tener conversaciones entre dos o más personas, de esta manera se podía comunicar entre integrantes del proyecto incluso cuando estos no se encontraban en la oficina, debido a que en varios casos se hacía necesario contratar personas que no se encontraban en la ciudad o el país, por lo tanto todas las conversaciones relacionadas con los proyectos se realizaban a través de *Slack*.

8. ANÁLISIS Y DISEÑO

Como se ha comentado anteriormente el proceso de desarrollo de las actividades asignadas comenzaba desde la creación de las tarjetas dentro de la herramienta de Trello, una vez se notificaba que la tarea había sido asignada se revisaba que se debía realizar para completar dicha tarea, las tareas que se realizaban generalmente en la práctica eran desarrollo de scripts, ensamblaje, creación de sistemas de partículas, desarrollo de cinemáticas de transición entre escenas, entre otras cosas, a continuación se describirá el proceso de análisis que se llevaba a cabo para cada una de las actividades:

8.1. DESARROLLO DE SCRIPTS

A lo largo de la práctica se desarrollaron varios scripts para los proyectos en los que se trabajó, algunos ejemplos de funcionalidades que se desarrollaron fueron controladores de la cámara del personaje la cual seguía al personaje a través del nivel, un controlador de la cámara para permitir al jugador recorrer el nivel para observar todo el panorama, un contador de *Frames per second*(FPS) o imágenes por segundo el cual fue usado para realizar pruebas de rendimiento en uno de los proyectos debido a que este fue desarrollado orientado a un hardware con pocos recursos, se desarrolló también scripts para el manejo de animaciones las cuales se activaban según cierto tipo de eventos dentro de los niveles, se desarrolló también scripts para el manejo del sonido del videojuego, entre otros, para el desarrollo de todos estos scripts era necesario realizar un previo análisis de lo que se requería en la actividad y que comportamiento final se esperaba de él, por lo tanto al momento de recibir la tarjeta en *Trello* se realizaba un análisis de los requerimientos que tenía, a partir de ahí si no se había desarrollado algo similar y era un comportamiento nuevo se revisaba la documentación oficial de Unity, y a partir de ahí desarrollar una solución a la tarea asignada, se decidía que variables iban a ser necesarias para el desarrollo. En Unity las variables que se crean es posible editar sus valores en tiempo de ejecución dentro del motor grafico para así depurar el comportamiento del objeto e ir ajustando el valor de la variable hasta obtener un valor que permita el comportamiento deseado, por lo tanto, desde el inicio en el análisis se debe establecer cuáles son las variables que influyen en el comportamiento del objeto y se desean modificar desde el inspector de Unity.

8.2. ENSAMBLAJE (ESCENARIOS, PERSONAJES, ANIMACIONES)

Para el desarrollo de las tareas de ensamble se manejaban diferentes procesos dependiendo de que se iba a ensamblar ya que los procesos que se manejaban eran distintos, los procesos que se manejaban dependiendo del tipo de ensamble que se quería realizar eran:

El proceso de la creación de un escenario para un videojuego empieza desde el desarrollo de un concepto de que es lo que se quiere crear, ver qué tipo de juego se está desarrollando, como es su ambientación, cual es el estilo gráfico, si es un juego 2D o 3D, que se espera que el jugador haga dentro de ese escenario, como lo podrá recorrer el jugador. El diseño de estos escenarios tiene mucho más trasfondo que simplemente luzca bien visualmente, este proceso de diseño de los niveles empieza desde el *game designer* que ese encarga de plantear un boceto de que quiere dentro del escenario teniendo en cuenta las consideraciones anteriores, el boceto se transmite al equipo de arte el cual se encarga de transformar esa idea o boceto a algo visual que se adapta a la estética del juego, todo eso se plasma en los *assets* que son las partes que conforman todo el arte del videojuego, ya pueden ser imágenes separadas del escenario o modelos 3D del mismo, y lo empaquetan en un archivo .rar el cual contendrá todo lo necesario para el ensamblaje de del nivel, dentro de dicho paquete también se incluyen algunos bocetos de cómo se espera que luzca el nivel al terminar.

En la práctica como ya se ha mencionado se trabajó en dos proyectos principalmente, los cuales manejaban estilos gráficos totalmente diferentes y la forma de ensamblar los *assets* era distinta, para el primero proyecto en el que se trabajó el cual fue Aqueducts o Voxels el cual era el nombre clave que se manejaba dentro del equipo de desarrollo, este proyecto era un videojuego 3D con una estética de cubos marcada, conocida como *voxelArt* de ahí el nombre clave, esta estética nace de lo que se denomina como pixel art el cual estilo de arte 2D basado en imágenes con poca resolución el cual eran utilizadas en las primeras generaciones de los videojuegos debido a que las consolas en esa época no contaban con un gran poder de cómputo por lo tanto las imágenes o *sprites* debían ser de tamaño pequeño, generalmente de 8 bits(8 pixeles x 8 pixeles), 16 bits, 32 bits, dependiendo de la consola, esto hacía que en la imagen se pudiera ver claramente cada pixel que conformaba dicha imagen, de ahí el nombre que tomo “*PixelArt*”.

Ilustración 6. Ejemplo de pixel Art. Super Mario Bros

Fuente: Tomado de: https://rpp.pe/tecnologia/videojuegos/super-mario-bros-videojuego-cumple-33-anos-noticia-1149562

Posteriormente, cuando los videojuegos incursionaron en los modelados 3D estos empezaron con modelados con muy bajo poligonaje debido a las restricciones de los recursos que poseían las consolas, por lo tanto muchos desarrolladores empezaron a crear la estética de sus juegos basados en cubos los cuales son figuras con pocos polígonos, por lo tanto consumían pocos recursos, de ahí nació el *VoxelArt* el cual es un estilo basado en cubos los cuales conforman los modelos 3D así como los pixeles formaban la imagen en el pixel art. Un muy reconocido juego basado en este estilo de arte es Minecraft el cual es el juego más vendido de la historia.

Ilustración 7. Ejemplo de Voxel Art. Minecraft

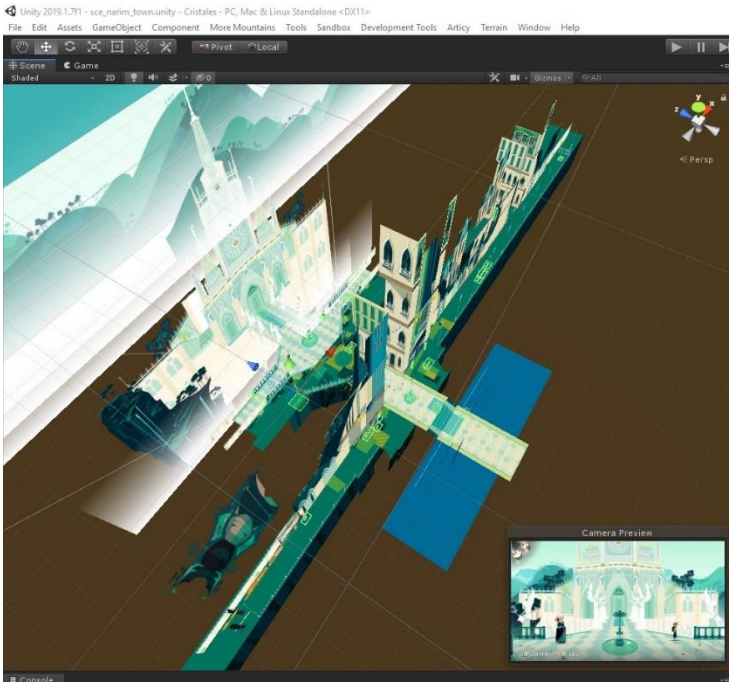
Fuente: Tomado de: https://minecraft-pocket-edition-apk-to-download.es.aptoide.com/

Debido a que el proyecto Voxels estaba destinado a ser ejecutado en computadores de especificaciones limitadas y no tan potentes, este debía consumir la menor cantidad de recursos posibles por esta razón se optó por manejar una estética VoxelArt, que como ya se había mencionado antes, los modelados 3D están compuestos por cubos los cuales son de bajo poligonaje y consumen menos recursos.

El artista que crea estos modelos 3D debe realizar cada pieza de nivel por separado (arboles, flores, puertas, postes, animales, adornos, suelo, puentes, etc.) para su posterior ensamblaje. Según el tamaño del modelado se decide si es mejor separarlo en varias piezas o no, los objetos pequeños generalmente constan de una sola pieza, en cambio objetos como el suelo del nivel que ocupa gran parte del mismo, generalmente es dividido en dos o más partes para ser más fácil de ensamblar y en caso de que se quiera realizar algún cambio al mismo, solo se tenga que modificar la sección deseada y no toda la pieza completa, ahorrando así tiempo. Para el ensamblaje de estos modelados debido a que eran cubos se trabaja cada pieza como un cubo de lego, uniendo los modelados por sus vértices y posicionándolos dentro de Unity.

Para el segundo proyecto Cristales este manejaba una estética completamente diferente, este proyecto se caracteriza por ser innovador en mecánicas, estética, etc., el arte para este proyecto es compuesto por imágenes 2D ensambladas en un escenario 3D lo cual crea un efecto de profundidad ya que el jugador puede recorrer el videojuego en el eje *x*, *y*, *z*.

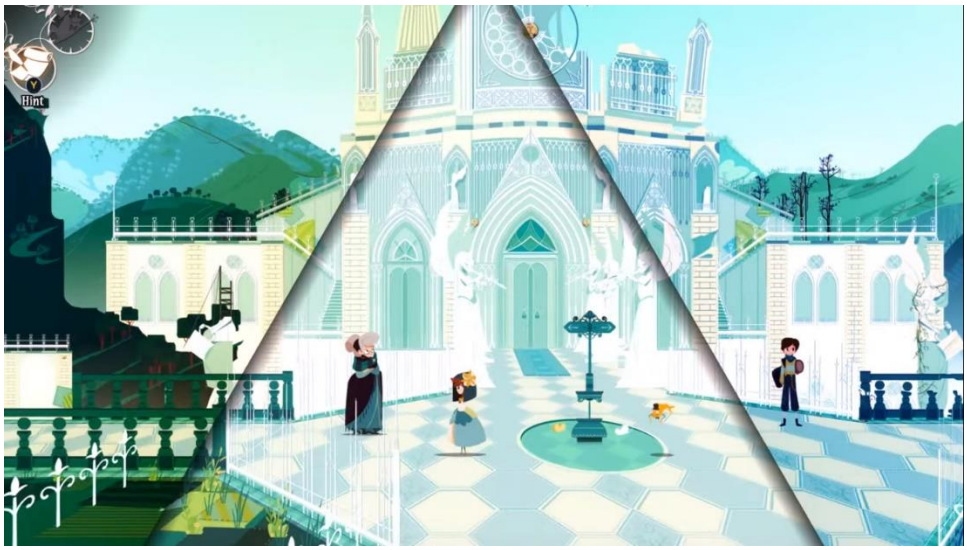
Ilustración 8. Ejemplo de cómo es un escenario de Cristales



Fuente: Autor

Una de las mecánicas del videojuego es que el jugador puede visualizar en la pantalla el pasado, el presente y el futuro además de poder interactuar con el pasado y futuro a través del acompañante del personaje principal el cual es un sapo llamado Matias.

Ilustración 9. Ejemplo de visualización de pasado, presente y futuro en Cristales



Fuente: Autor

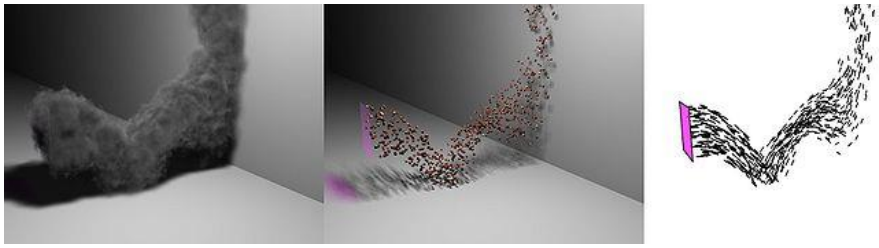
Dado esta mecánica cada escenario debe ser pensado cuidadosamente debido a que dentro de la misma escena van a estar interactuando 3 tiempos diferentes, los artistas deben crear diferentes imágenes para cada objeto, ya que este objeto puede estar en el pasado, presente y futuro. Así mismo se debe ensamblar el objeto teniendo en cuenta esta mecánica para lo cual se había creado una herramienta o plugin para Unity la cual ayudaba al ensamblaje de las imágenes de cada objeto debido a que estos debían estar los 3 siempre en la misma ubicación en el espacio en todo momento, por ejemplo si se movía de posición una imagen que correspondía a la línea temporal del pasado, las imágenes del presente y futuro debían quedar ubicadas en la misma posición a la que se movió la del pasado, esta herramienta permitía crear un objeto padre en el cual se encontraban las imágenes de cada tiempo del objeto por lo cual si se movía el padre, las imágenes hijas se moverían junto con el padre.

Debido a que a pesar de estar ensamblado en un espacio 3D todos los *assets* del proyecto eran imágenes de dos dimensiones, por lo tanto deben ser posicionados de cierta forma para que cuando el jugador este recorriendo el mundo se cree el efecto de un mundo 3D, para mantener un control de la perspectiva del mundo que tiene el jugador en el proyecto la rotación de la cámara estaba bloqueada y está dependiendo del escenario se movía por ciertos rieles creados con ese propósito o de manera libre si era requerido. Por lo tanto, al momento del ensamblaje de las imágenes estas debían ser posicionadas y rotadas de manera que al recorrer el mundo produjeran dicho efecto para ello también se debía establecer que camino tendría permitido recorrer el jugador y sobre que rieles se movería la cámara.

8.3. SISTEMAS DE PARTÍCULAS

Tanto en los videojuegos como en las películas, se hace uso de muchos efectos hechos por computadora para poder replicar el humo, fuego, explosiones, agua, etc., para poder realizar estos efectos se hace uso de un Sistemas de partículas, que el autor William Reeves el cual era un investigador de LucasFilms y trabajo en películas como Star Trek II define como:" ... una colección de muchas partículas diminutas que juntas representan un objeto difuso. Durante un periodo de tiempo, las partículas se generan en un sistema, se mueven y cambian dentro del sistema y mueren desde el sistema."

Ilustración 10. Ejemplo de un sistema de partículas



Fuente: Tomado de: https://es.wikipedia.org/wiki/Sistema_de_part%C3%ADculas_(software)

El motor grafico Unity permite la creación de sistemas de partículas de manera sencilla, tomando como base una imagen o un modelado 3D, y modificando valores como la duración de la emisión, intervalos de emisión, cantidad de partículas, etc., se pueden crear sistemas de partículas sencillos. Para este proyecto se crearon diversos sistemas de partículas para efectos de nieve, vapor, lluvia, resplandor, explosiones y emisiones de partículas, entre otros, su creación será detallada en el capítulo de desarrollo.

8.4. TESTING

Las pruebas de testeo fueron realizadas para el proyecto de Voxels, debido a que este proyecto estaba enfocado a maquinas con un poder computacional bajo, se necesitaba estar verificando constantemente que la maquina pudiera ejecutar el videojuego con unos estándares mínimos de al menos 25 FPS o superior, por la tanto, cada cierto tiempo se debían realizar pruebas de lo que se había implementado dentro del juego no impactaba al rendimiento de este. La primera tarea para realizar estas pruebas era desarrollar un contador de *fps* para poder realizar la medición de los mismos mientras el juego era ejecutado en la maquina a la que estaría destinado el juego, se desarrollaron algunas versiones hechas por el equipo de trabajo, pero al final se eligió un programa de un tercero para realizar las pruebas, para la realización de las mismas se solicitaba que se realizaran varias *builds* del proyecto con algunas variaciones entre ellas para identificar que nuevas implementación eran las que impactaban en el rendimiento del videojuego y se debía testear cada *build* registrando los valores de los fps mínimos y promediados obtenidos en un documento que sería entregado al *game designer* para que fueran evaluados.

8.5. TIMELINE

Las *timeline* permiten crear cinemáticas, secuencias de *gameplay*, audio secuencias y complejos efectos de partículas [13], estas serían usadas dentro del proyecto de Cristales para crear pequeñas escenas cinemáticas dentro del videojuego ya que era algo que no se había trabajado dentro del proyecto fue necesario indagar de su funcionamiento en la documentación oficial de Unity. Para la creación de dichas *timeline* se debía realizar un boceto de la escena basado en el guion del videojuego, en el cual se describía paso a paso que debía suceder, que personajes debían aparecer en la misma, en qué lugar se desarrollaría la escena, se debía establecer que evento dentro del videojuego iniciaría dicha escena, entre otros aspectos.

9. DESARROLLO

A continuación, se describirán el procedimiento que se realizaba para completar cada una de las actividades que se eran asignadas.

9.1. DESARROLLO DE SCRIPTS

A partir del análisis previo al desarrollo del script, estos se empezaban a desarrollar para un ejemplo se hablará del script desarrollado para el control de la cámara para realizar *zoom,* para el desarrollo de este script el requerimiento era que fuera posible a partir de un evento se realizará un acercamiento de la cámara y luego a partir de otro evento esta se alejará. Por lo tanto, como paso inicial se definían cuáles son las variables que se iban a necesitar para el script y los diferentes métodos que serían usados, y se desarrolla la lógica que sea necesaria para el cumplimiento del requerimiento, al desarrollar dicho script, su funcionamiento es testeado por el *game designer* y el productor para evaluar si se tiene el comportamiento deseado de la cámara, si no es así el que lo testeo da unas indicaciones de que se debe corregir o se da por aprobado dicho script.

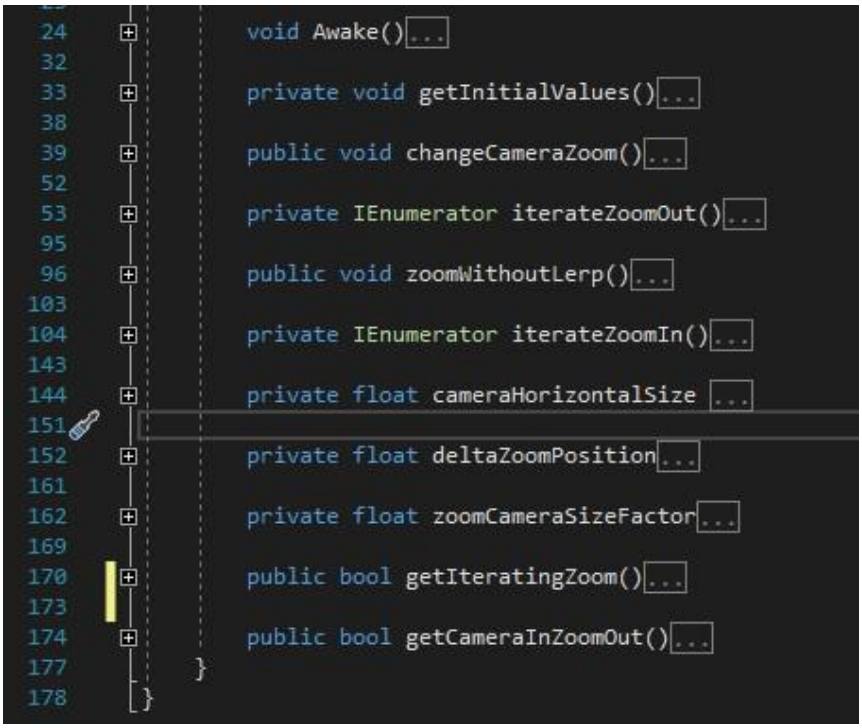
Ilustración 11. Definición de los atributos del script

```
public class CameraController : MonoBehaviour
{
    [SerializeField] private float _zoomFactor;
    [SerializeField] private float _initialCameraSize = 14.0f;
    private Vector3 _initialCameraPosition;
    private float _initialHorizontalSize;
    private bool _cameraInZoomOut = false;
    private bool _iteratingZoom = false;
    private Camera _camera;
    private float _zoomPositionFactor; // valor utilizado para calcular la nueva posicion de la camara cuando se hace zoom.
    private float _zoomCameraSizeFactor; // valor utilizado para calcular el nuevo tamaño de la camara cuando se hace zoom.

    private const float POSITION_LERP_SPEED = 5.0f;
    private const float SIZE_LERP_SPEED = 4.0f;
    private const float LERP_TOLERANCE = 0.02f;
```

Fuente: Autor

Ilustración 12. Definición de los métodos del script



Fuente: Autor

9.2. ENSAMBLAJE

Como ya se ha recalcado anteriormente debido a las diferencias entre los dos proyectos en los cuales se trabajó, la manera de ensamblar los *assets* era diferente entre ellos por lo tanto se explicará de manera separada cada uno de ellos.

9.2.1. Voxels

El proceso de ensamble en Voxels empezaba desde que el artista encargado de la creación de los modelados subía el paquete de los archivos correspondientes a Google Drive, este paquete contenía todos los modelados 3D y un boceto de como el artista y el *game desinger* había contemplado el escenario. A partir de tener todos los recursos subidos en su respectiva plataforma se descargaban, se descomprimían y se agregaban al proyecto local de Unity, en el caso del proyecto de Voxels el tipo de archivos que se manejaba de manera general era de tipo Obj ya que es un formato de archivo muy usado y de bastante compatibilidad con Unity. Si era el caso de ensamblar un escenario, después de importar los archivos al proyecto local se organizaban en las diferentes carpetas basados en la estructura del proyecto la cual es la siguiente.

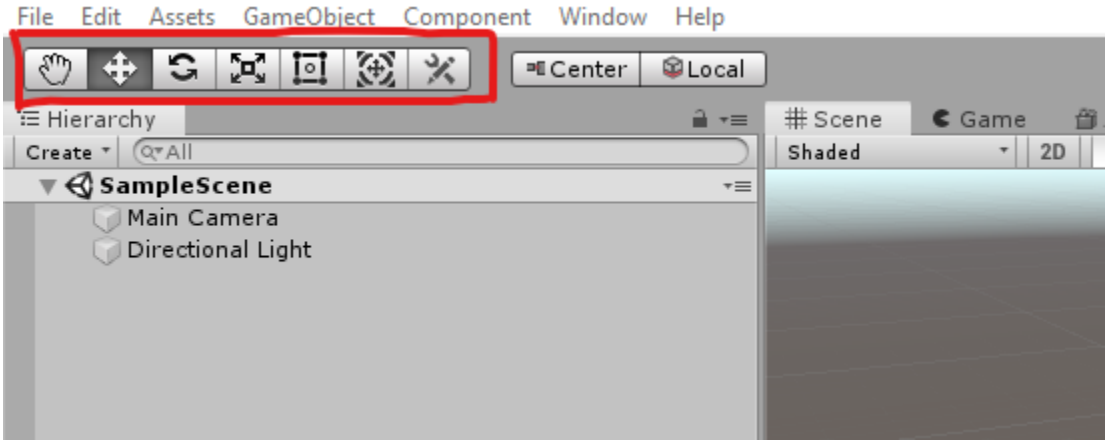
Ilustración 13. Estructura de las carpetas en el proyecto Voxels

Fuente: Autor

Los modelados 3D para Voxels se realizaron haciendo uso del programa MagicaVoxel el cual permite realizar modelados 3D a partir de un cubo el cual se puede ir esculpiendo hasta formar la figura deseada, basándose siembre en quitar o poner cubos a la figura inicial, al momento de la exportación del archivo el programa permite guardar el modelado 3d como Obj y guardar aparte la paleta de color o textura que se aplicó al mismo dentro de Magica para que al agregarlo a Unity se vea como el artista lo realizo en Magica. Por lo tanto, después de importar los archivos y organizarlos, se agregaban dentro de la escena y se verificaba que su escala y rotación fuera correcta, si no era así en primera instancia se intentaba corregir desde Unity y si no era posible se le era comunicado al artista para su corrección. Si todo estaba bien se procedía a agregarle la textura al modelo, esta textura como se había mencionado se encuentra entre los archivos que se sube el artista al repositorio y es una imagen png con los colores que se usó en modelado el cual Unity interpreta y colorea el modelado basado en lo que configuro el artista en Magica. Ya cuando

se realiza este proceso con todos los archivos del escenario, se procede a acomodar los modelos dentro del escenario basado en el mockup base que diseño el artista junto con el *game designer*, Unity posee diversas herramientas para que el usuario se pueda mover por el escenario y pueda mover los objetos dentro del mismo, algunas de esas herramientas son mover, rotar, escalar, estas son las herramientas básicas las cuales eran usadas para el ensamblaje del escenario, en la siguiente imagen se pueden observar dichas herramientas.

Ilustración 14. Herramientas de Unity



Fuente: Autor

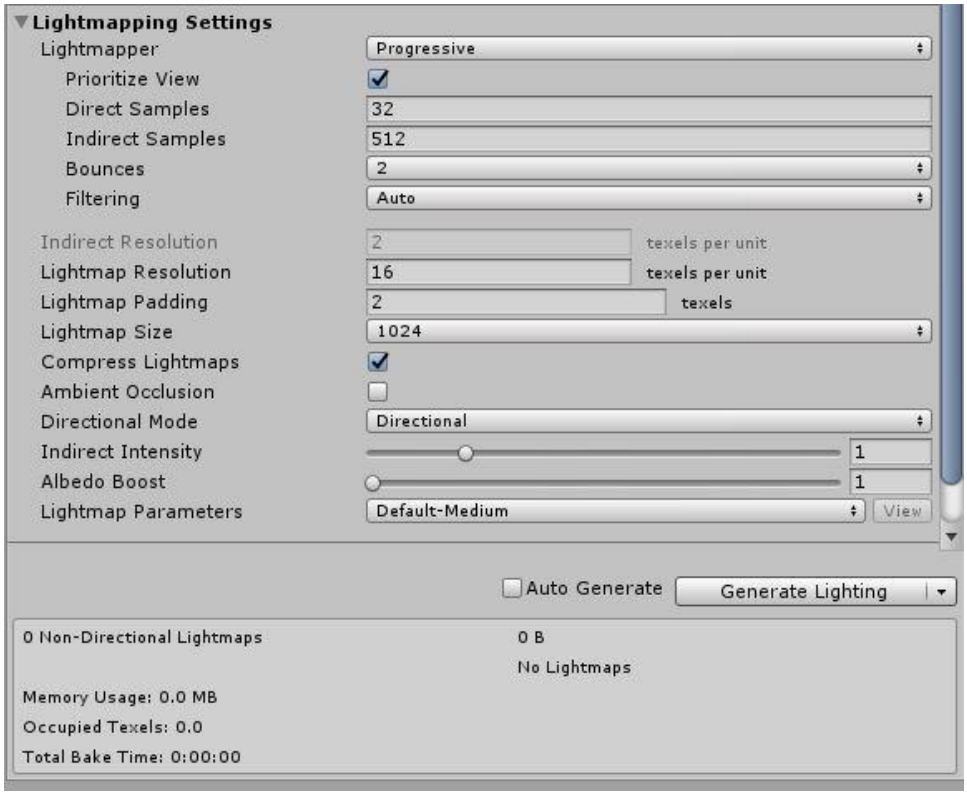
Una de las herramientas que brinda Unity permite juntar dos vértices de dos modelos diferentes, dado que el proyecto maneja cubos, esta herramienta permite posicionar de forma exacta los modelos que conforman el escenario debido a que más adelante es necesario que la superficie del escenario sea lo más definida posible y no tenga desniveles innecesarios para que cuando se realice el *lightmap* no se hagan notorias estas imperfecciones.

Unity es un motor que permite la simulación de interacciones físicas entre objetos, una de esas interacciones es la colisión, esta es manejada por el componente de colisiones el cual permite al jugador poder caminar sobre una superficie, no atravesar objetos, saber cuándo un objeto lo impacta, saber que objetos están en contacto, etc. Para que el jugador no atraviese el suelo y caiga indefinidamente, o atraviese los objetos del escenario es necesario agregar el componente de *Collider* en este caso *Box Collider* dado que es el que encaja perfectamente con la forma de los escenarios, para agregar estos *Box collider* se crea un *game object* vacío dentro del cual se irán agregando Cubes con dicho componente, y se empiezan a acomodar dichos cubes en tamaño y posición que concuerden con los modelados del terreno, para cuando es necesario que queden posicionados perfectamente a la altura del terreno para que cuando el jugador navegue sobre el no de un efecto de que estuviera flotando se hace uso de la herramienta de juntar los vértices de las dos figuras (el *Cube* con el *collider* y el modelado), cuando ya se encontraban en posición se eliminaba el

componente de *mesh* dentro de los cube para que no se hiciera visible y solo se mantenía el componente de colisión.

La iluminación en tiempo real en los videojuegos es algo bastante costoso en cuanto los recursos de la plataforma y como ya se había comentado anteriormente el proyecto de Voxels estaba dirigido a ser implementado y jugado en ordenadores con recursos limitados, por lo tanto se optó por otro método para manejar la iluminación dentro del videojuego y se hizo uso de lo que se llaman los *Lightmaps* que a diferencia de la iluminación global utilizada generalmente la cual es calculada *frame* a *frame*, los *lightmap* se crearon para evitar ese gasto de cálculo computacional de la iluminación en tiempo real, la técnica consiste que en el momento del desarrollo se precalcule la iluminación que tendrá la escena y toda esta información se condensara en una textura la cual se sobrepondrá a la textura original y dará el efecto de iluminación aunque no haya ninguna fuente de luz en la escena, dado que es una textura precargada esta será estática y la escena siempre lucirá de la misma manera. Unity posee un componente para realizar *Lightmaps* el cual es el siguiente

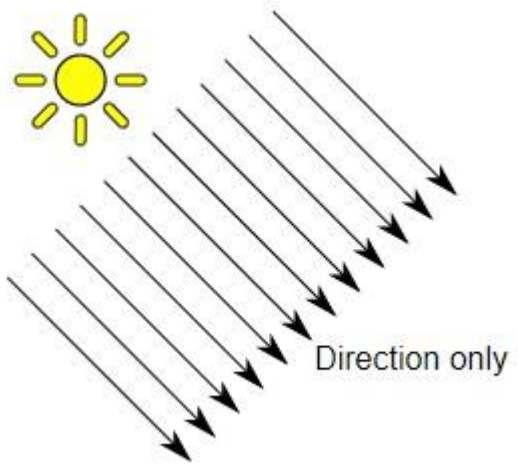
Ilustración 15. Componente de Unity para realizar los Lightmaps



Fuente: Tomado de: https://docs.unity3d.com/Manual/Lightmapping.html

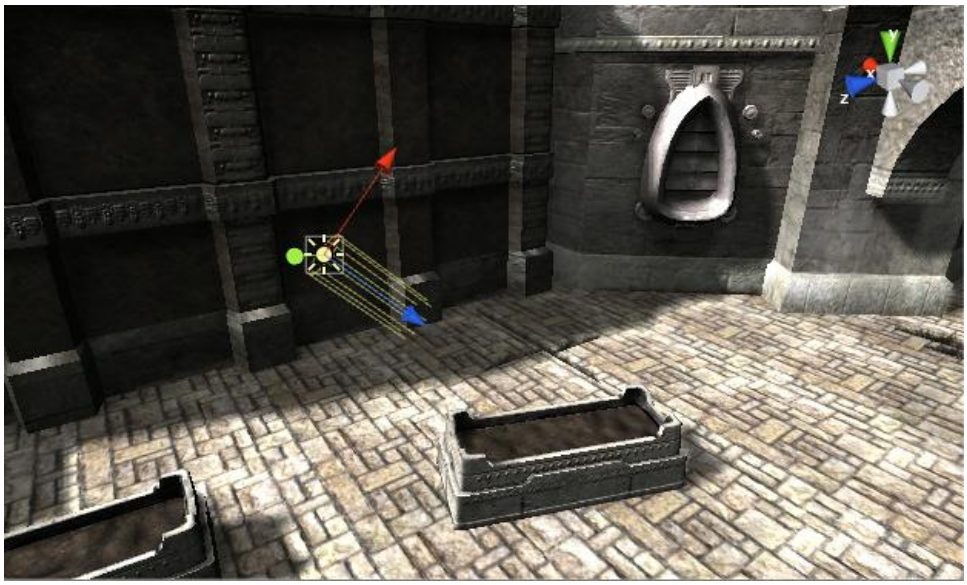
Para realizar el *lightmap* se debía preparar los componentes de la escena, primero que todo se debían agregar unas *directional light* o luz direccional, que es un *component* de Unity el cual genera luz en una sola dirección y se usa generalmente para simular la luz que viene del sol.

Ilustración 16. Ejemplo de una Directional Light



Fuente: Tomado de: https://docs.unity3d.com/Manual/Lighting.html

Ilustración 17. Ejemplo de una Directional Light en Unity



Fuente: Tomado de: https://docs.unity3d.com/Manual/Lighting.html

Se posiciona una apuntando hacia la dirección que se desea que tenga la escena y la otra se posiciona apuntado en la dirección contraria, pero con una intensidad de luz menor, esto se hace para que los objetos que giren o se muevan no sea tan notoria que la luz es estática y se noten sombras muy fuertes. Antes de realizar el *lightmap* se debían reposicionar en otro lugar de la escena todos aquellos objetos que no son completamente estáticos como plataformas que subirán y bajaran, el jugador en sí, el agua, etc., debido a que si se dejan es su posición original estos generaran unas sombras que al momento de que dichos

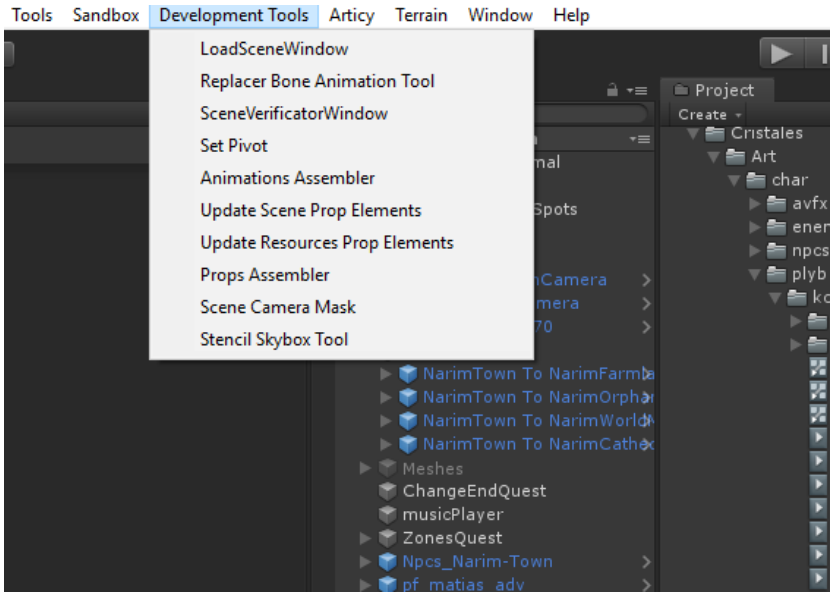
objetos se muevan la sombra quedara en dicha posición aunque el objeto no se encuentre allí, por lo tanto después de mover todos dichos elementos, se genera el *lightmap* de la escena.

9.2.2. Cristales

9.2.2.1 Escenarios

“Cris Tales es un sentido y precioso homenaje a los JRPG clásicos con una nueva perspectiva. Explora el pasado, actúa en el presente y observa como tus decisiones cambian dinámicamente el futuro” [14], en este juego el jugador puede interactuar con el pasado, presente y futuro en la misma pantalla, por lo tanto, cuando se ensamblaron los escenarios de este proyecto se debió tener en cuenta como luciría cada escenario en el pasado, en el presente y el futuro, debido a que cada punto del tiempo podía lucir diferente, por ejemplo un edificio dentro del videojuego podía tener pasado y presente pero no futuro debido a que fue destruido, para evitar tener que ensamblar 3 escenarios diferentes para cuando el escenario estaba en los tres puntos del tiempo, se desarrollaron algunas herramientas las cuales facilitaron en el ensamblaje de los niveles:

Ilustración 18. Herramientas de desarrollo para Cristales



Fuente: Autor

Una de las herramientas que fueron desarrolladas para este proyecto por BTG fue el Props Assembler, la cual permitía que a partir de seleccionar un *sprite*, esta herramienta buscaba por el nombre del archivo, que debía tener una nomenclatura ya establecida todas las coincidencias y agrupaba todos los *sprites* con el mismo nombre en un mismo objeto con las configuraciones necesarias el cual servía para los tres puntos en el tiempo, y así se evitaba tener que ensamblar el escenario tres veces ya que todos los *sprites* de los tres tiempos se movían como uno solo y siempre permanecían en la misma posición. Para

hacer uso de esta herramienta se establecieron como se manejarían los nombres de los *sprites* en el proyecto, por lo que todos los *sprites* del escenario tenían la siguiente nomenclatura o formato:

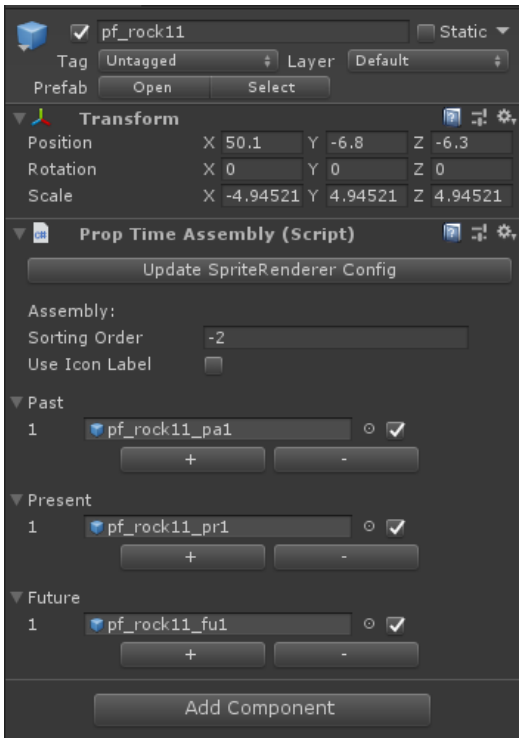
"tipoArchivo_nombreObjeto_tiempo"

Ilustración 19. Ejemplo formato de nombres de los sprites

Fuente: Autor

Como se puede ver en la ilustración anterior se tiene un objeto que tiene tres estados en el tiempo, siguiendo el formato del nombre: "s" hace referencia a que es un *sprite*, "rock11" es el nombre del objeto y la siguiente parte hace referencia a que punto en la línea del tiempo pertenece ese *sprite*, "fu" es futuro, "pa" es pasado y "pr" es presente, cabe resaltar que puede haber mas de un futuro, pasado o presente, por eso el numero al final del nombre. Después al ingresar uno de estos *sprites* a la herramienta se creaba el siguiente objeto que era con el cual se debía ensamblar los escenarios.

Ilustración 20. Ejemplo de objeto creado con la herramienta de Props Assembler

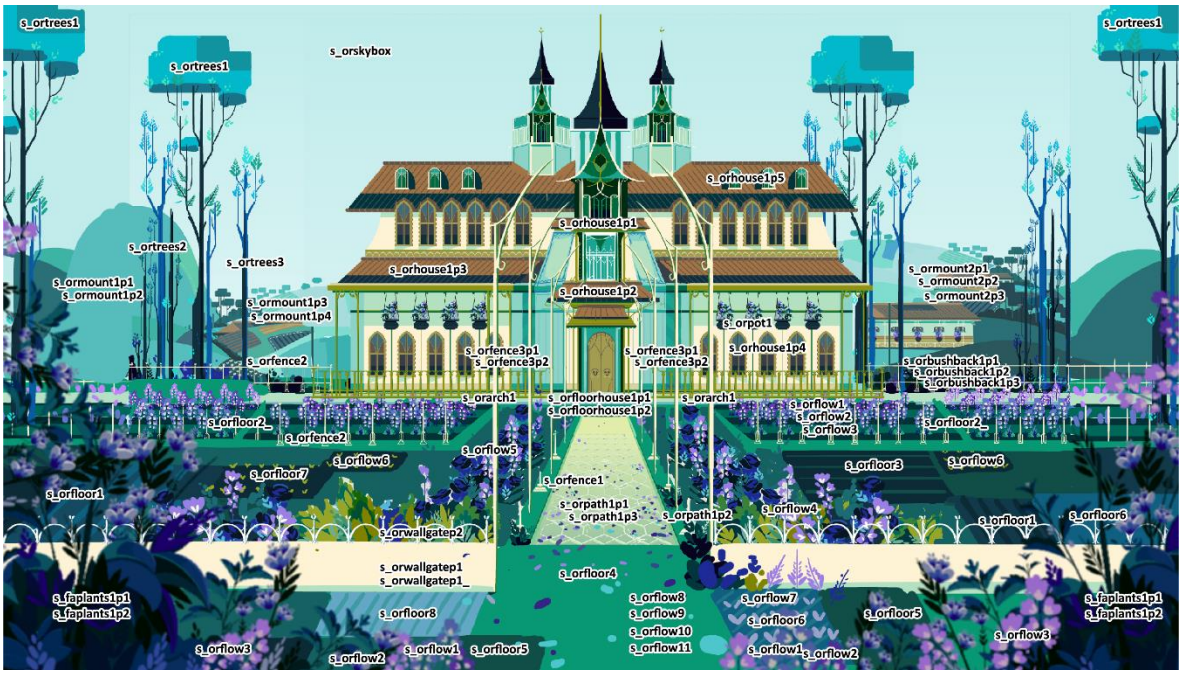


Fuente: Autor

Después de haber importado todos los archivos del escenario dentro de Unity y haber creado todos los objetos con la herramienta se podía ensamblar el escenario. Debido a que Cristales es un juego en perspectiva 3D, pero con un arte 2D el ensamblaje de los escenarios se debían hacer de forma artesanal, acomodando cada uno de los objetos por separado.

Al igual que en el proyecto de Voxels, se empezaba a ensamblar el escenario a partir de un mockup realizado por los artistas, como el que se puede apreciar en la siguiente ilustración.

Ilustración 21. Mockup de un escenario de Cristales



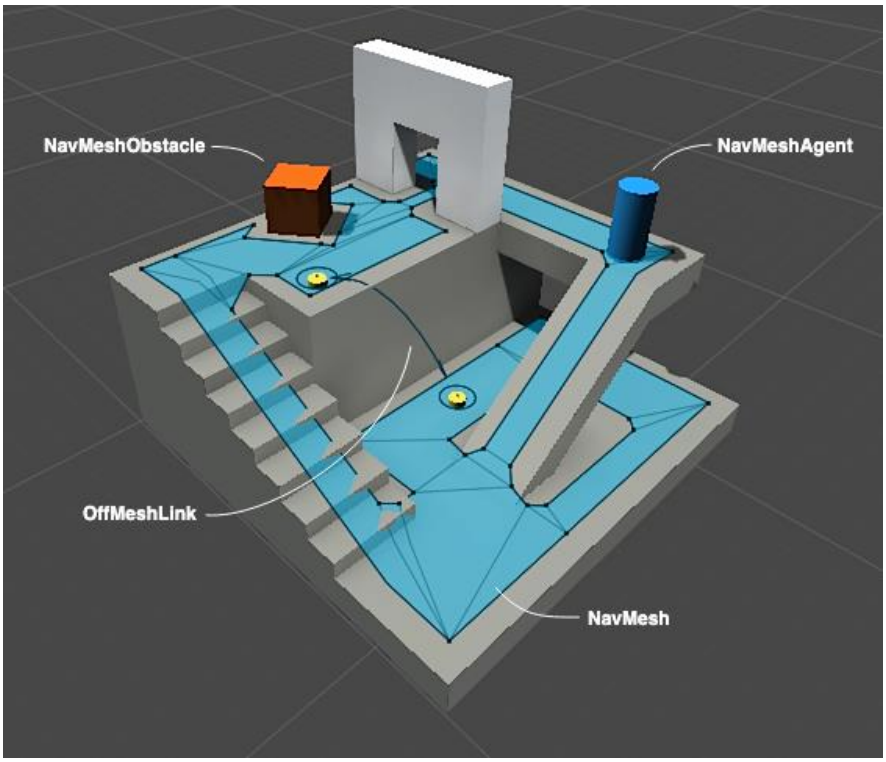
Fuente: Autor

Para crear el efecto de un mundo 3D se decidió que el jugador no podría rotar la cámara, y se estableció para cada escenario una inclinación en el eje X de los objetos para crear ilusiones visuales al momento en que el jugador recorría el escenario, en la mayoría de los casos las inclinaciones eran las siguientes, el suelo tenía una inclinación en el eje X de 90 grados y las demás estructuras de 0 grados, en algunos escenarios se manejaron inclinaciones de 90 grados en el suelo y 60 en los demás objetos, esto era establecido por los artistas en conjunto con el *game designer*.

Para el ensamblaje de estos escenarios se iniciaba con el suelo y a partir de ahí se empezaban a posicionar cada objeto uno por uno tomando como referencia un objeto principal, como por ejemplo en la ilustración 19 se tomó como referencia la casa y partiendo de ella se iba posicionando cada uno de los objetos, debido a que los mockups no eran exactos en las distancias, se debía ir armando una versión base del escenario para ver cómo se veía ya siendo recorrido por el jugador, para ello se debían agregar componentes estándar que poseían todas las escenas para que pudieran ser recorridas por el jugador, e ir verificando la calidad del escenario. A medida que se iba ensamblando la escena esta era revisada por los artistas y el *game designer* para verificar que el escenario cumpliera con los requisitos y si no era el caso corregir los errores que se tuvieran.

Al final de ensamblar cada pieza del arte era necesario la creación del *NavMesh*, el cual es una funcionalidad de Unity en la cual se puede definir las zonas y lugares por las cuales el jugador puede navegar dentro del escenario [15], en la siguiente ilustración se puede observar un ejemplo de un *navMesh* y sus diferente componentes.

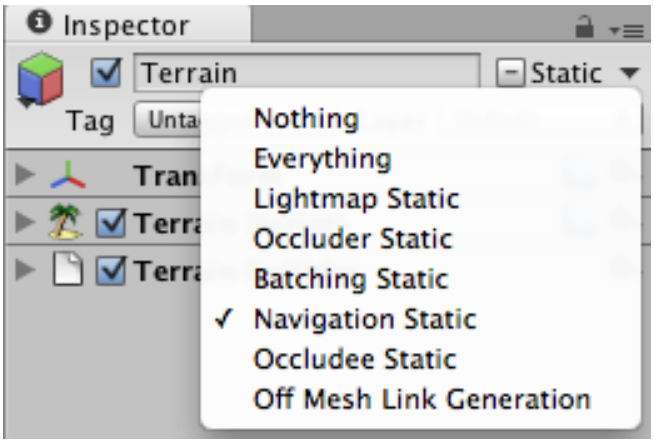
Ilustración 22. Ejemplo de navMesh



Fuente: Tomada de: https://docs.unity3d.com/es/530/Manual/nav-NavigationSystem.html

Para la creacion del NavMesh se crean unos planos(Gameobject), los cuales no seran visibles al jugador, estos planos se crean y se deben ir escalando y posicionando para que coincidan con la altura y posicion de los sprites del suelo por el cual se desea que el jugador pueda navegar, para facilitar la modificacion de estos planos el quipo de desarrollo tenia una herramienta por la cual por medio de unos puntos que se creaban estos iban formando un plano, se podian añadir mas puntos o vertices a dicho plano y estos podian ser reposicionados individualmente para formar figuras mas complejas las cuales se adaptaran mejor al diseño del arte. Cuando se finalizaba la creacion de estos planos se debia establecer que cada uno de los planos que iba a formar el suelo serian Navigation Static como se observa en la siguiente ilustración.

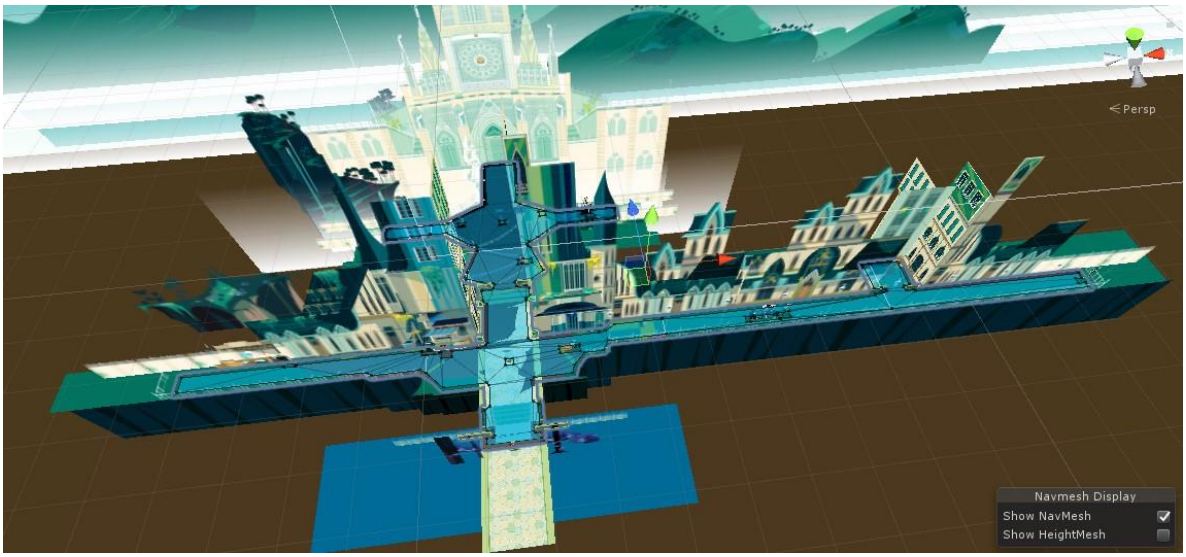
Ilustración 23. Configuración del atributo static para el navMesh



Fuente: Tomada de: https://docs.unity3d.com/es/530/Manual/nav-BuildingNavMesh.html

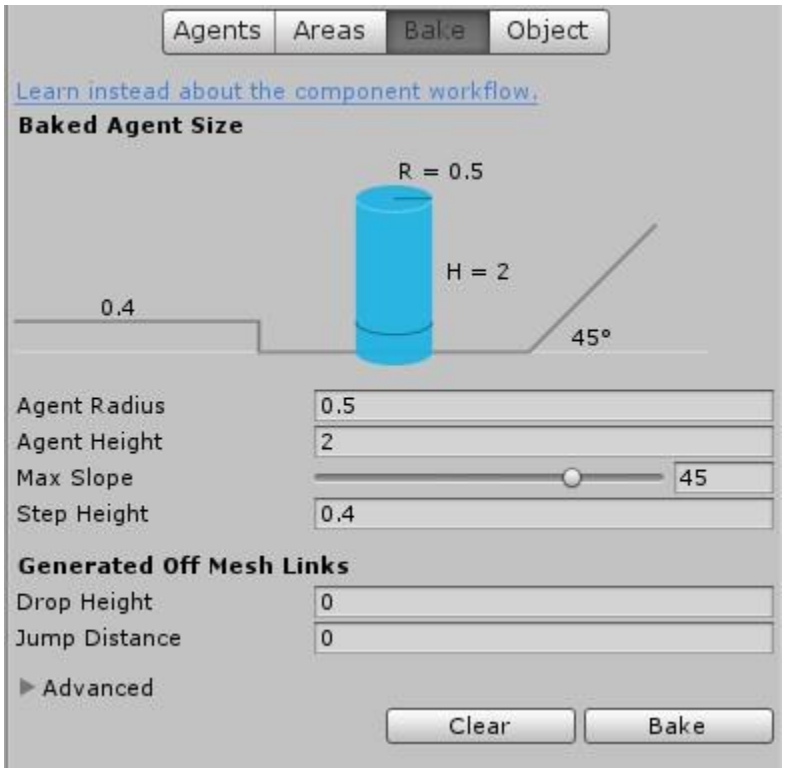
Debido a que dentro del escenario hay objetos tales como faros, cajas, etc., y el jugador no debería atravesarlos para que no se pierda la inmersión de un mundo 3D, se debían crear objetos tales como *Spheres* o *Cubes* para posicionarlos en el mismo lugar donde se encuentran los *sprites* de dichos objetos y agregarles un *Collider* para que el jugador no los atraviese. Por último, es necesario realizar el *baking* del escenario en el cual se generará el archivo del *NavMesh*, para el *baking* se pueden establecer algunos parámetros para tener en cuenta tales como la inclinación máxima de un plano por el cual el jugador podrá navegar, entre otras cosas.

Ilustración 24. Ejemplo de navmesh en Cristales



Fuente: Autor

Ilustración 25. Componente donde se realiza el baking



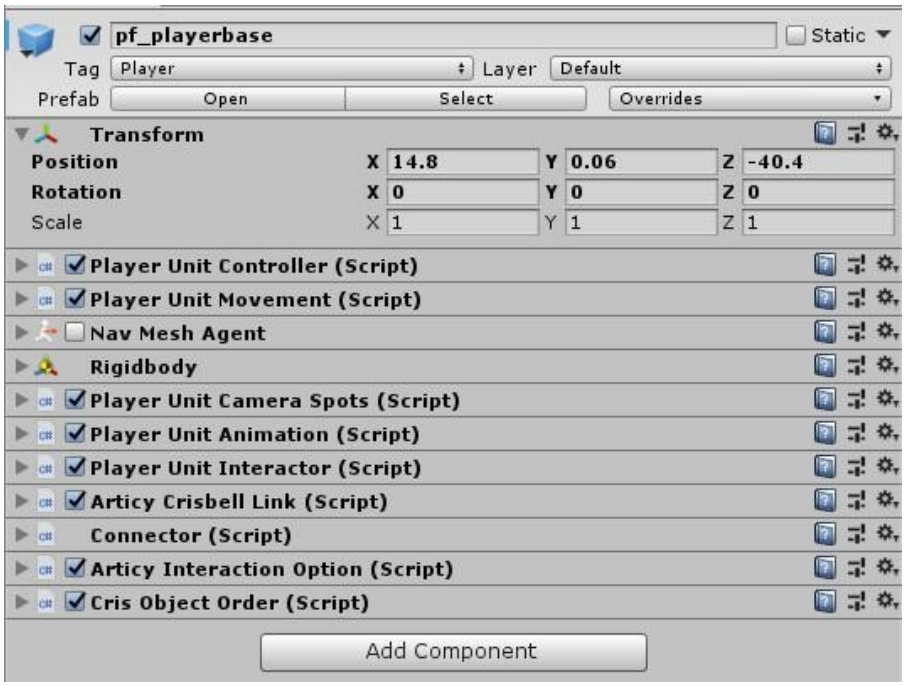
Fuente: Autor

Después de realizado el *NavMesh*, todos los objetos usados para su creación son deshabilitados para que no sean visibles cuando el jugador recorra el escenario.

9.2.2.2 Personajes

Para el ensamble de personajes se contaban con unos personajes base, los cuales contaban con los componentes básicos que se habían establecido, tales como un *animator*, *scripts* para navegación o para la lucha dependiendo si el personaje iba a ser usado para navegar o en batalla respectivamente, entre otros componentes.

Ilustración 26. Ejemplo de componentes del personaje base en Cristales

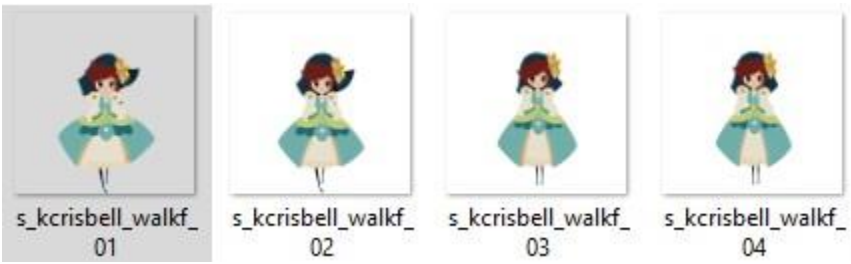


Fuente: Autor

Debido a que ya se contaba con una base, se debía añadir los componentes adicionales tales como las animaciones del personaje dentro del *animator*, modificar los *Collider*, etc. Las animaciones eran entregadas por el grupo de arte por medio del repositorio SVN como ya se había mencionado, los archivos que contenían eran las imágenes de las animaciones del personaje(*idle*, *walk*, *run*, *attacks*, *jump*, *special moves*, etc) y un archivo de texto por cada animación el cual contiene la cantidad de fotogramas que debe durar cada *frame*, todo el contenido estaba organizado en carpetas dentro del proyecto y así como los *sprites* del escenario también el nombre tenía un formato el cual sería usado en una herramienta usada para su ensamble. El formato es el siguiente.

"tipoArchivo_nombrePersonaje_nombreAnimacion_numeroImagen"

Ilustración 27. Ejemplo del formato de las animaciones



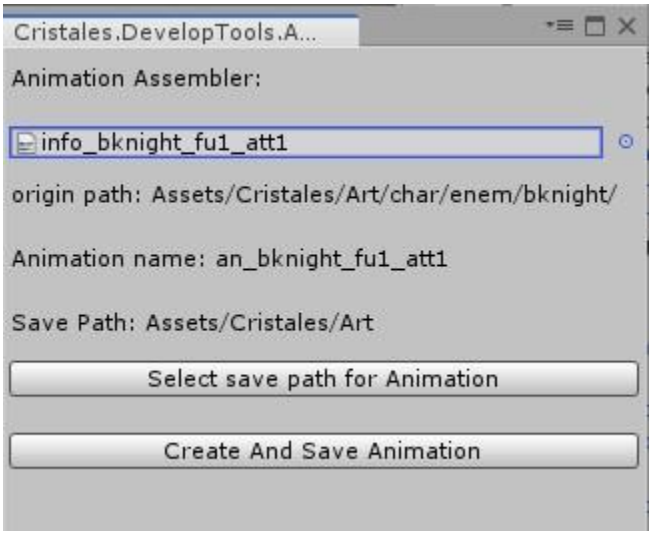
Fuente: Autor

Como se observa en la ilustración anterior el nombre de las imágenes iniciaban con el tipo de archivo, en este caso "s" porque es un *sprite*, seguido del nombre del personaje, luego el nombre de la animación en este caso "walkf" debido a que es la animación de caminar

hacia el frente y al final el numero de la imagen, en este ejemplo se muestran los primeros cuatro *frames* de una animación.

Para el ensamble de las animaciones de los personajes se contaba con una herramienta la cual permite que a partir del archivo de texto que se había mencionado este automáticamente genere el archivo de la animación, para la creación de la misma la herramienta solicita ingresar el archivo de texto con la información de la animación y un lugar donde se guardara ya creada, a partir del formato del nombre del archivo busca las imágenes que coinciden con el mismo nombre y las organiza dentro del componente *animation* para la creación de la animación y posteriormente solo es necesario modificar la cantidad de *frames* por segundo a la cual fue hecha dicha animación, este valor era establecido por el grupo de arte para que la animación se viera fluida.

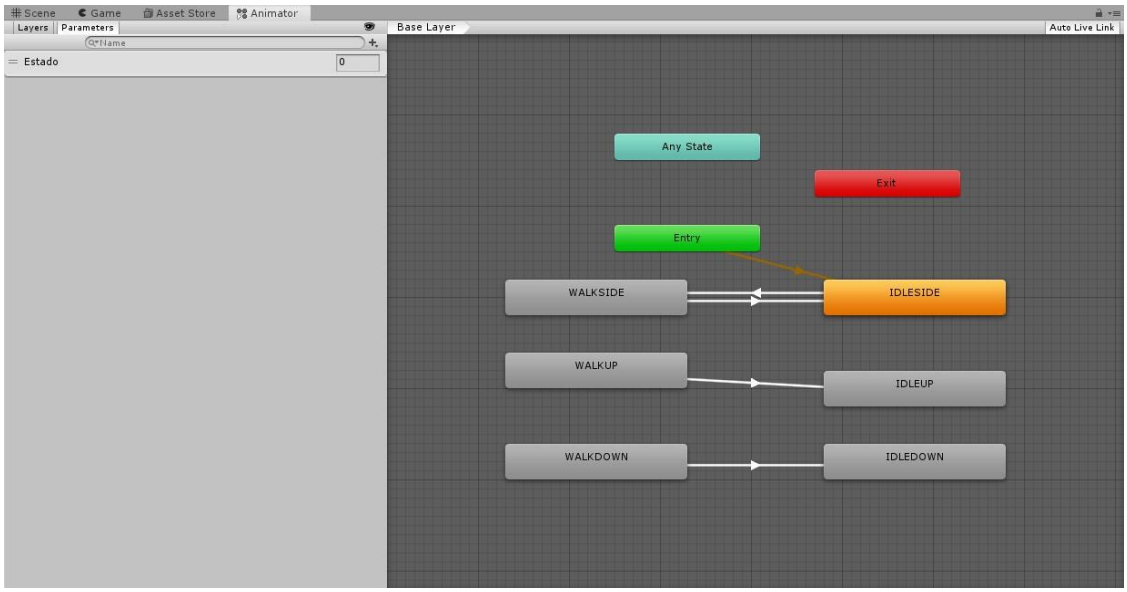
Ilustración 28. Herramienta para la creación de animaciones a partir de un archivo de texto



Fuente: Autor

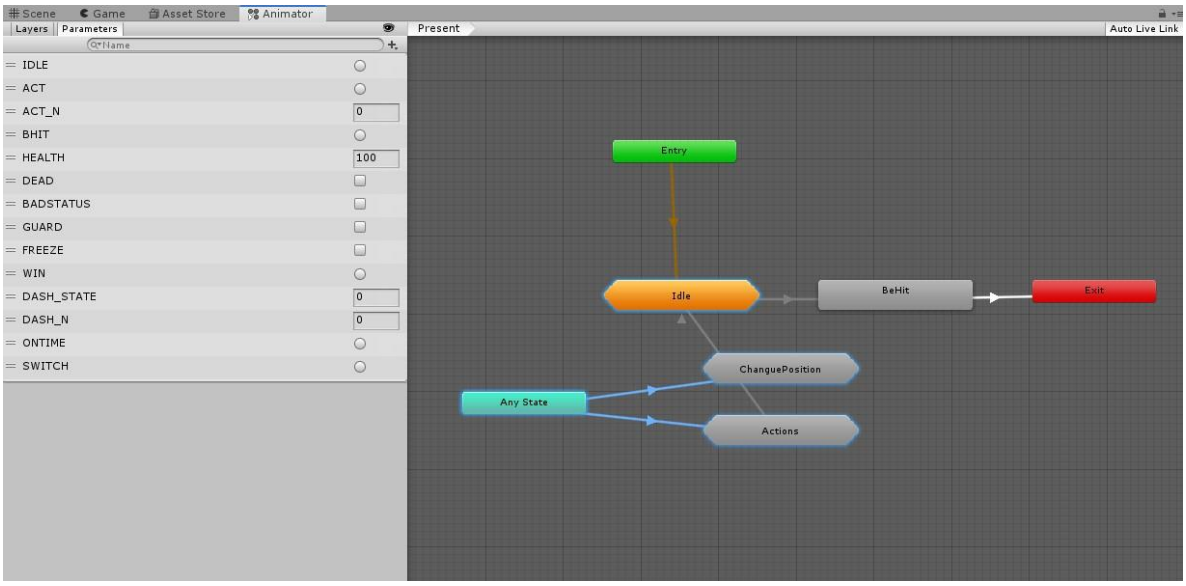
Después de generar las animaciones estas debían reemplazar a las que estaban en el *animator* del personaje base, para ello se debía ingresar a cada animación del *animator* y reemplazar la animación nueva del personaje, y después de ello se realizaba una verificación de la transición de las animaciones por medio del *animator* dado que este cuenta con los valores o atributos con los que se establece que animación debe ser reproducida en cada momento.

Ilustración 29. Ejemplo de un animator de un personaje de exploración



Fuente: Autor

Ilustración 30. Ejemplo de un animator de un personaje de combate



Fuente: Autor

Después de añadidas todas las animaciones, en el componente principal es necesario reemplazar los *sprites* que se usan para dichas animaciones debido a que para algunas de ellas el artista usa el sistema de animación de *bones* de Unity para crearlas.

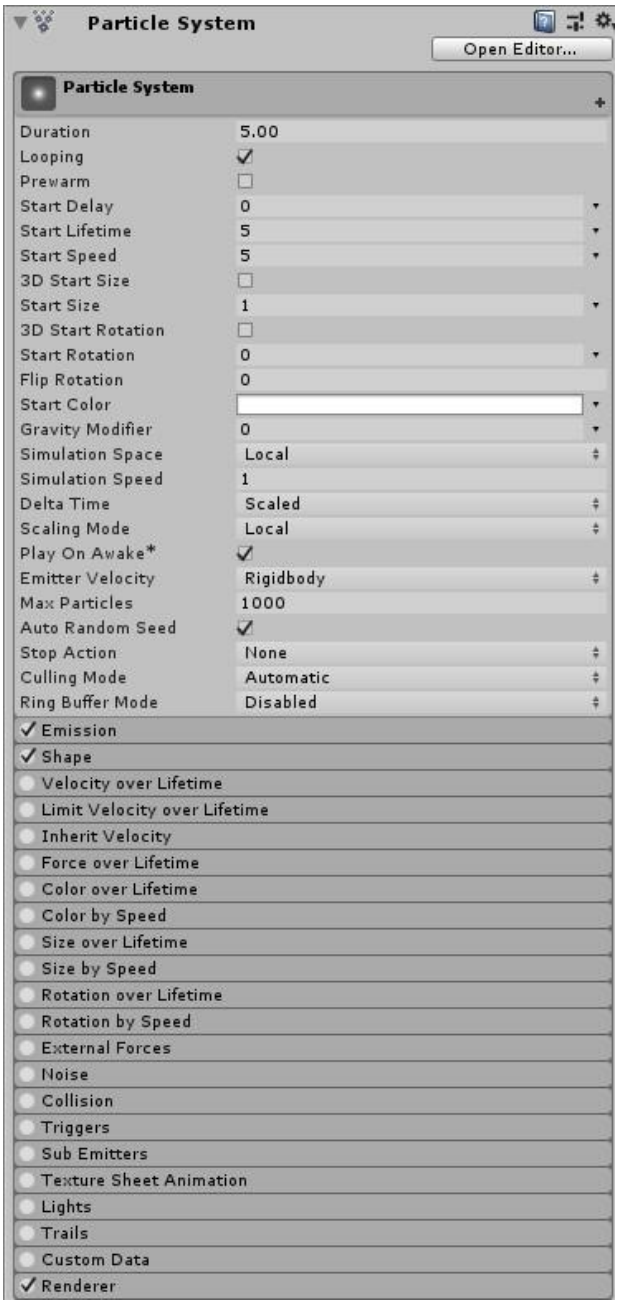
Ilustración 31. Componente de un personaje

Fuente: Autor

9.3. SISTEMAS DE PARTÍCULAS

Un sistema de partículas permite simular cosas como humo, fuego, explosiones, etc., para Voxels y Cristales se crearon algunos sistemas de partículas tales como un efecto de vapor, un efecto de explosión para la finalización de los niveles, efectos de brillo para los objetos, entre otros. Unity cuenta con un componente para la creación de los sistema de partículas, este componente cuanta con diferentes atributos los cuales se pueden modificar para crear el efecto que se desea, unos de los principales atributos usados fueron *duration* con el cual se establece la duración en tiempo del sistema de partículas , *looping* que establece si la animación se repetirá continuamente, *start speed* el cual es la velocidad inicial de las partículas , *start size* el cual es el tamaño inicial de las partículas, *shape* el cual establece de qué forma geométrica serán emitidas las partículas, *emision* que establece cada cuanto se emitirán partículas desde el origen o si será un flujo continuo de las mismas, *velocity over lifetime* el cual permite modificar la velocidad de las partículas a través del tiempo para crear efectos de ralentización, *size over lifetime* el cual permite crear el efecto de que el tamaño partícula crece o decrece a través del tiempo y *color over lifetime* el cual permite crear el efecto de desaparecer dicha partícula reduciendo el valor del alfa.

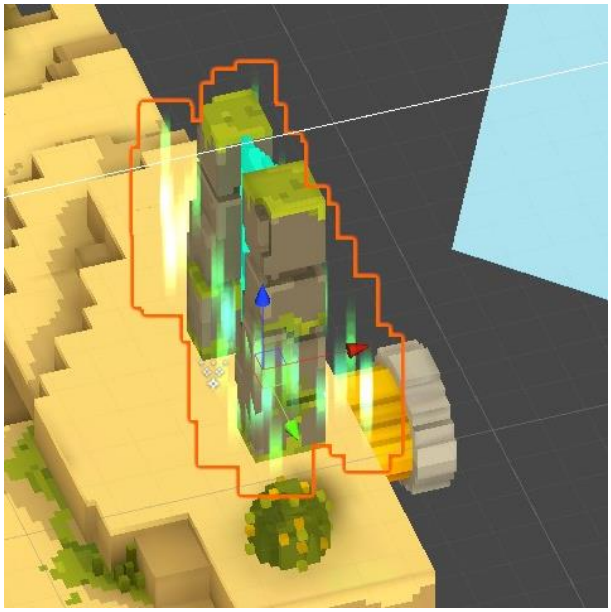
Ilustración 32. Sistema de partículas en Unity



Fuente: Autor

Una parte fundamental para la creación del sistema de partículas es contar con una imagen o *sprite* el cual es el que será emitido como partícula, por ejemplo para crear el efecto de hojas cayendo a través del escenario, el *sprite* usado como partícula es una imagen de una hoja, la cual a través de los atributos del sistema partículas de Unity mencionados anteriormente crearan dicho efecto de la caída de las hojas, por lo tanto para la creación de los sistemas de partículas también fue necesario crear diferentes imágenes en *Photoshop* para ser usadas como partículas, estas imágenes pueden ser desde un cuadrado simple, o un efecto de un brillo hasta una imagen detallada de un objeto.

Ilustración 33. Ejemplo de un sistema de partículas en el juego Voxels



Fuente: Autor

9.4. TESTING

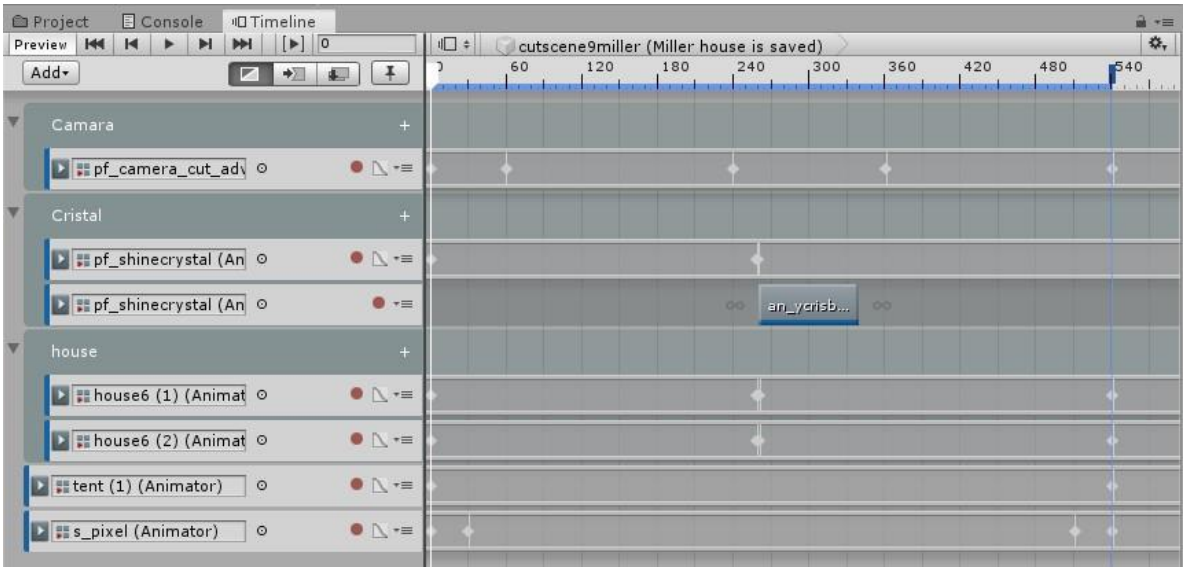
El *testing* fue realizado en el proyecto Voxels debido que a la poca potencia computacional que contaba la plataforma a la cual estaba dirigido el videojuego era necesario estar evaluando el rendimiento del juego a medida que se desarrollaban nuevas características. Dicha evaluación era realizada cuando se agregaba una característica nueva al proyecto por ejemplo cada vez que se agregaba un nuevo nivel, se debía generar un *build* con dicha versión para ser testeada directamente en el computador que el cliente brindo para el testeo del videojuego. Para la medición del rendimiento se instaló un plugin el cual brinda la información de los *fps* en tiempo real en la pantalla. Las pruebas eran realizadas en la plataforma antes mencionada y se testeaba el rendimiento del juego al menos por cinco minutos realizando diferentes acciones, los datos de las pruebas se registraban en un documento en el cual constaba de una descripción en la cual se hacía un resumen de que cambios tenía dicha versión, los *fps* mínimos, promedio y máximo obtenidos en la prueba. El documento era revisado por el *game designer* para establecer si eran unos valores aceptables para la calidad de la jugabilidad, si no era así se realizaban correcciones y se volvían a testear, hasta que se obtenía una calidad/rendimiento aceptable en los estándares del proyecto.

9.5. TIMELINE

Las *timeline* fue un componente que se introdujo al proyecto de Cristales para poder realizar pequeñas animaciones dentro del juego, para poder realizar las *time line* se debió investigar

directamente la documentación brindada por Unity debido a que este componente no había sido utilizado en el proyecto anteriormente. Las *timelin*e eran definidas por el *game designer* y escritor, se encontraban documentadas en el libreto del videojuego, en el cual se describía quienes participaban en la escena y cuál era el orden de los sucesos, a partir de ese guion eran desarrolladas. Para la creación de un time line es necesario crear un *gameObject* vacío al cual se le añadirá el componente de *Playable director*, en este componente se añaden todos los objetos los cuales interactuaran en la escena como se muestra en la siguiente ilustración.

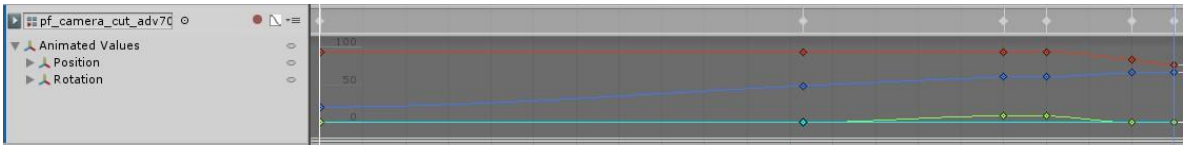
Ilustración 34. Componente del time line



Fuente: Autor

En la anterior ilustración se puede observar el ejemplo de una time en la cual se mostraría una escena de una casa siendo reconstruida después de un evento en el juego, por lo tanto fueron agregados todos los elemento que se animarían dentro de dicha time line como la cámara, el cristal que muestra los distintos estados del tiempo, la casa que sería salvada, la tienda que también se salvaría y un *sprite* de un pixel el cual sirve para realizar los *fade* de la escena, en las time line cada componente de la misma se anima por separado, por ejemplo para realizar la animación de la cámara se realiza el siguiente procedimiento, se oprime el botón rojo que se observa en cada componente para iniciar la grabación, el time line permite que cada cambio de posición que se realice en el *frame* quede guardado como un fotograma clave, entonces en el caso de querer mover la cámara de una posición X a una posición Y, se deberá posicionar el cursor en la línea de tiempo en el segundo 0 y mover la cámara a la posición X después se deberá mover el cursor en la línea del tiempo avanzando los segundos que desea que dure dicho movimiento por ejemplo en el segundo 10 y mover la cámara a la posición Y, esto el time line creara automáticamente el movimiento de la cámara desde el punto X a Y en 10 segundos, el time line permite modificar las curvas del movimiento para que la velocidad de la cámara no sea lineal si no que comience lento y después vaya acelerando.

Ilustración 35. Ejemplo de las curvas de un time line



Fuente: Autor

Para la animación de un personaje se deben de tener al menos dos líneas de animación para el mismo debido a que en una línea se animara su movimiento a través del escenario y en la otra se agregaran las animaciones tales como caminar, saltar, etc.

Ilustración 36. Time line de un personaje

Fuente: Autor

10. RESULTADOS

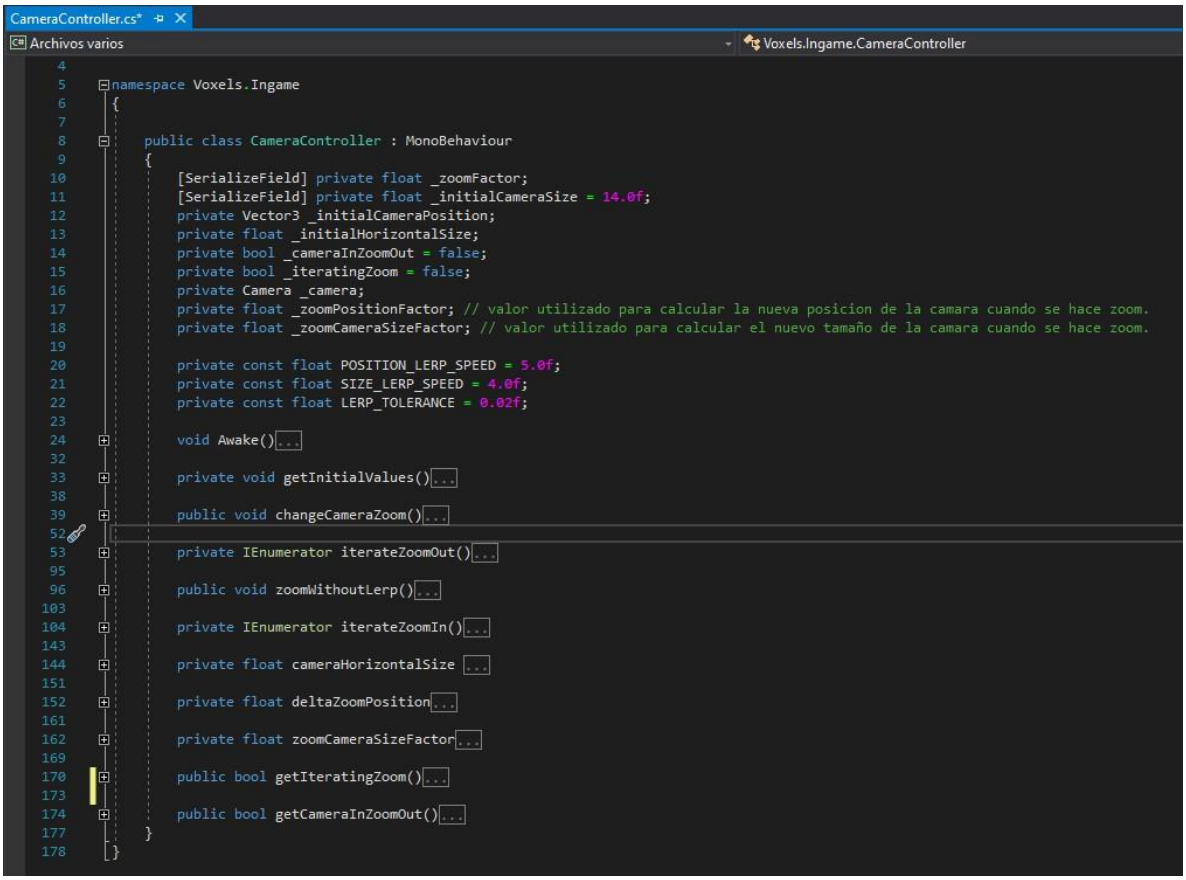
En este capítulo se mostrarán todos los resultados obtenidos en esta práctica, estos resultados pueden ser evidenciados también la versión final de Voxels y en el demo lanzado para Steam de Cristales, los cuales los podrán encontrar en los siguientes enlaces:

- Voxels (Aqueducts): “https://endless-studios.itch.io/aqueducts”
- Cristales Demo: “https://store.steampowered.com/app/1079830/Cris_Tales/”

10.1. DESARROLLO DE SCRIPTS

Se desarrollaron diversos scripts para los proyectos de Voxels y Cristales, entre ellos uno para el control del zoom de la cámara, uno para animar la cámara del videojuego cuando se cumpliera cierto objetivo, uno para habilitar e inhabilitar objetos dependiendo de ciertas condiciones, un script para el seguimiento de la cámara al jugador, uno para realizar el paneo de la cámara por el escenario cuando se oprimía una tecla, un script para controlar el volumen de la música dentro del videojuego, entre otros. Cada uno de estos scripts eran testeados por el productor y el líder de programación Armando Calderón, y algunos de ellos fueron incorporados en la versión final del videojuego, otros por cambios en las mecánicas de los juegos no fueron incluidos. Debido al contrato de confidencialidad firmado con la empresa no pueden ser mostrados a terceros.

Ilustración 37. Script de control de zoom de la cámara en Voxels

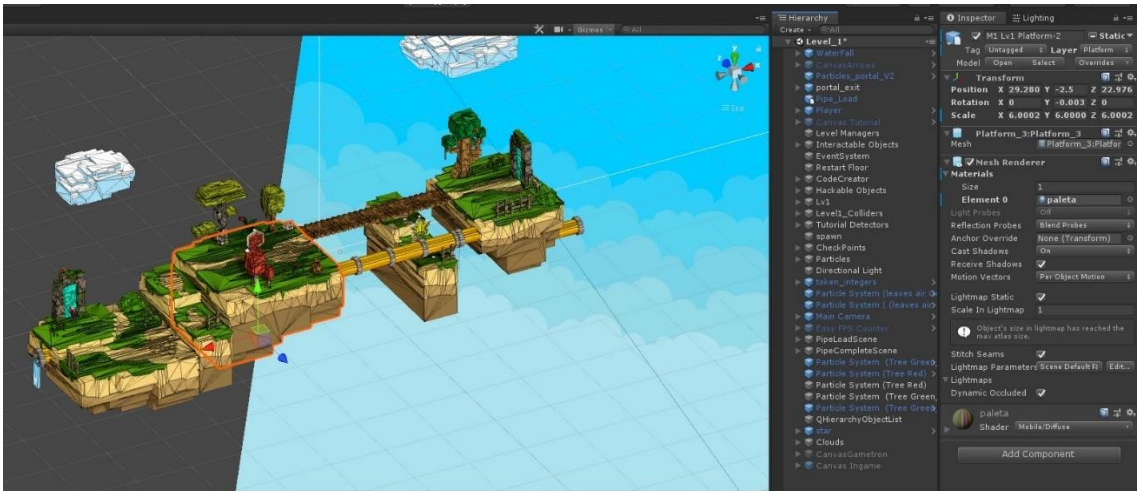


Fuente: Autor

10.2. ENSAMBLAJE

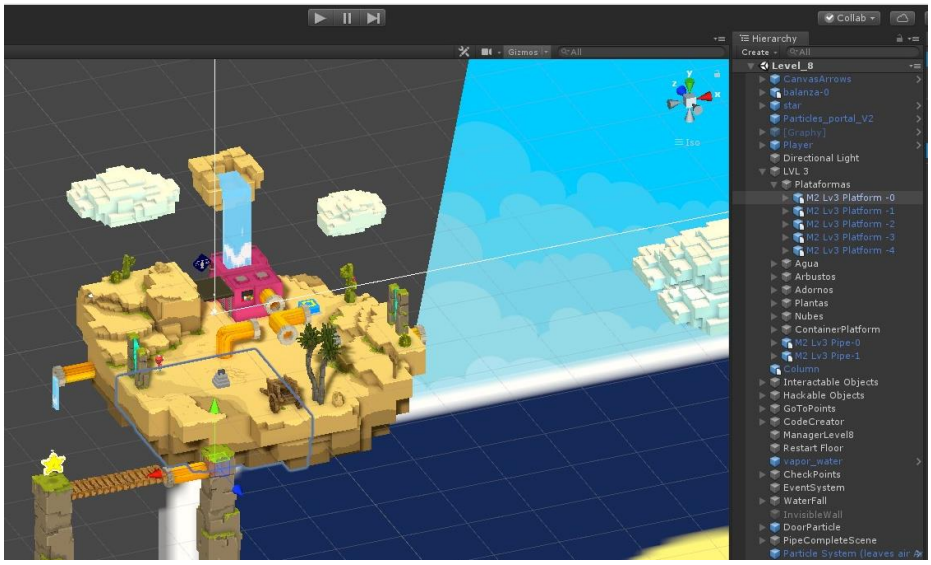
A continuación, se mostrarán algunos de los escenarios ensamblados para los proyectos de Voxels y Cristales los cuales fueron testeados y aprobados por el productor, los cuales se encuentran en la versión final de Voxels y en el desarrollo actual de Cristales.

Ilustración 38. Escenario del nivel 1 del mundo 1 de Voxels



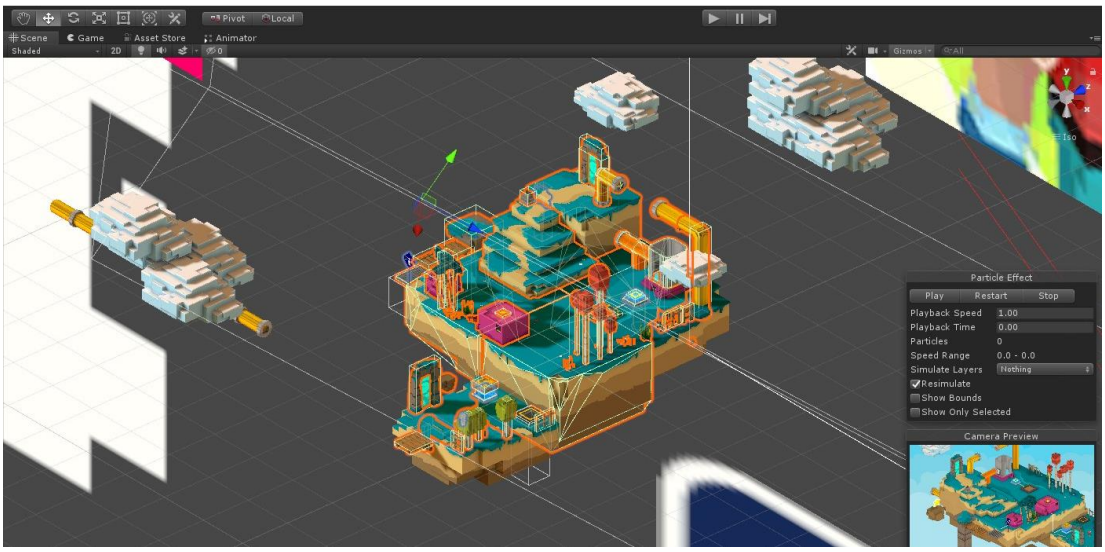
Fuente: Autor

Ilustración 39. Escenario del nivel 3 del mundo 2 de Voxels



Fuente: Autor

Ilustración 40. Escenario del nivel 3 del mundo 3 de Voxels

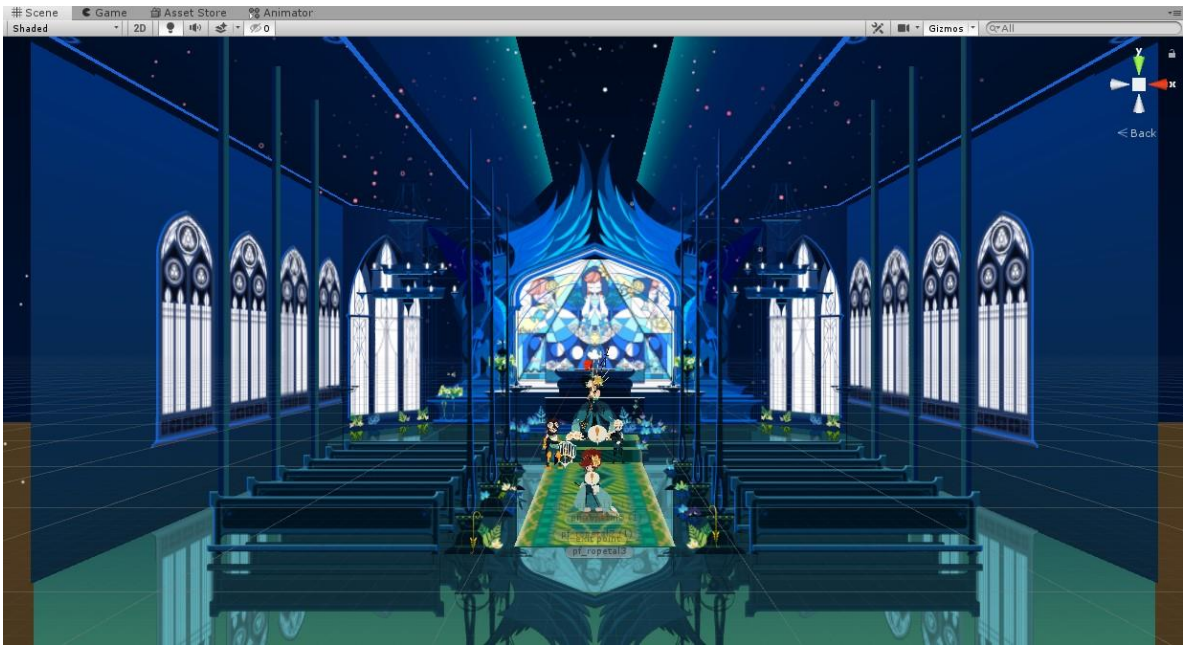


Fuente: Autor

Ilustración 41. Ciudad principal de Cristales

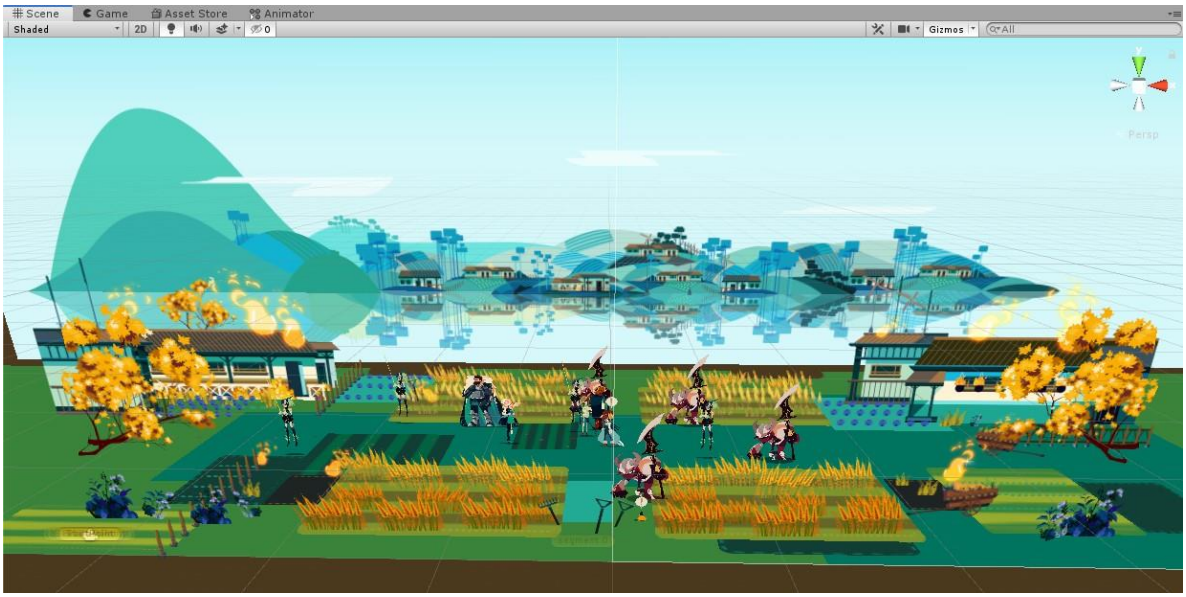
Fuente: Autor

Ilustración 42. Interior de la catedral de la ciudad principal



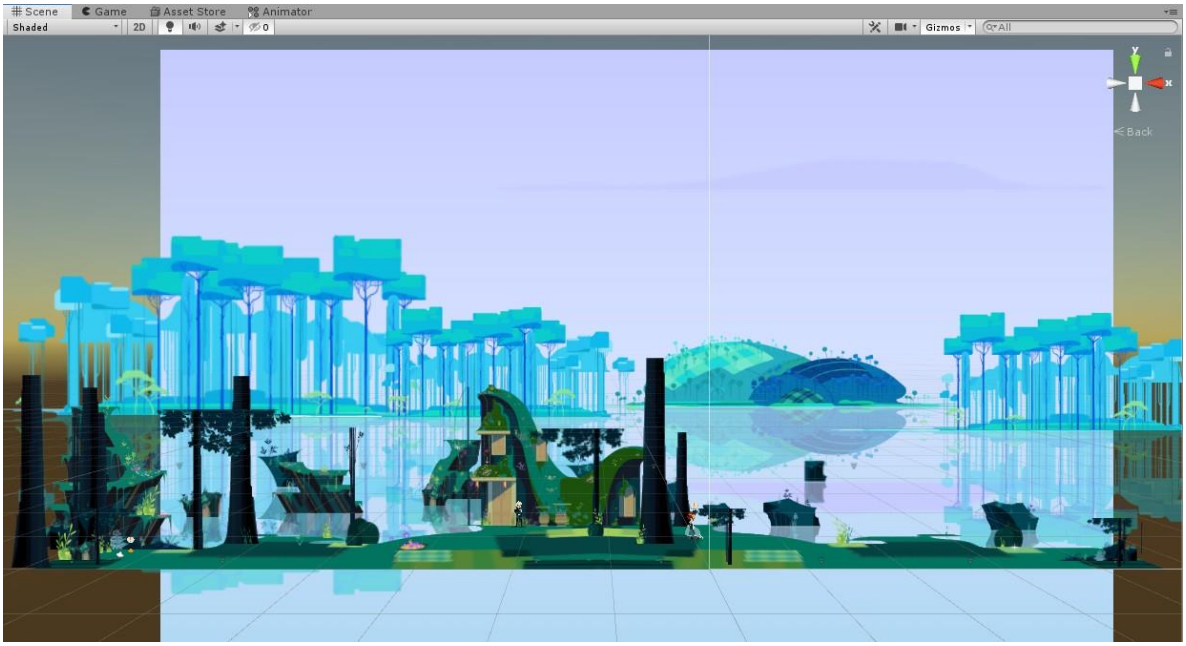
Fuente: Autor

Ilustración 43. Granja de Cristales



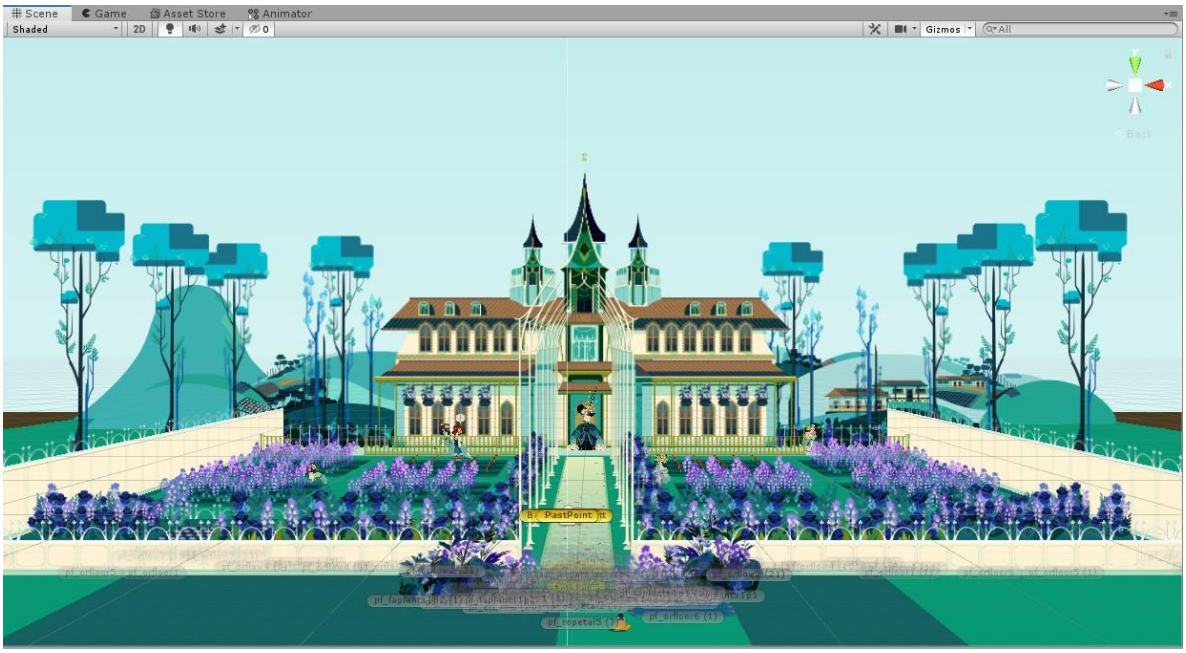
Fuente: Autor

Ilustración 44. Bosque en Cristales



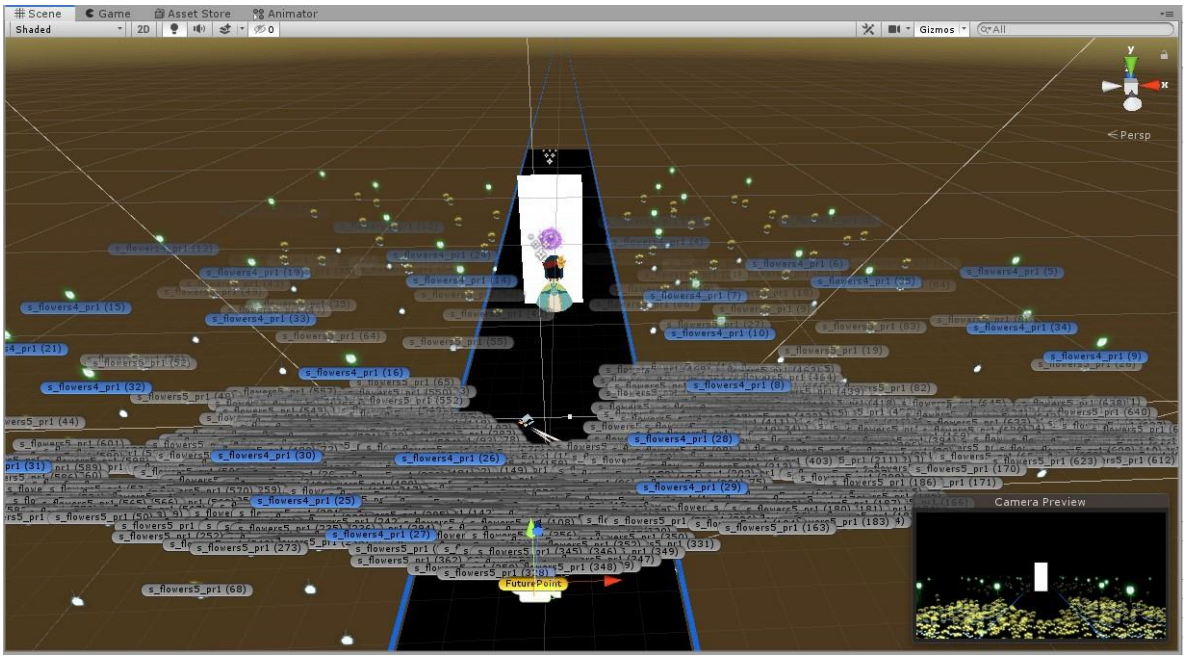
Fuente: Autor

Ilustración 45. Orfanato de Cristales



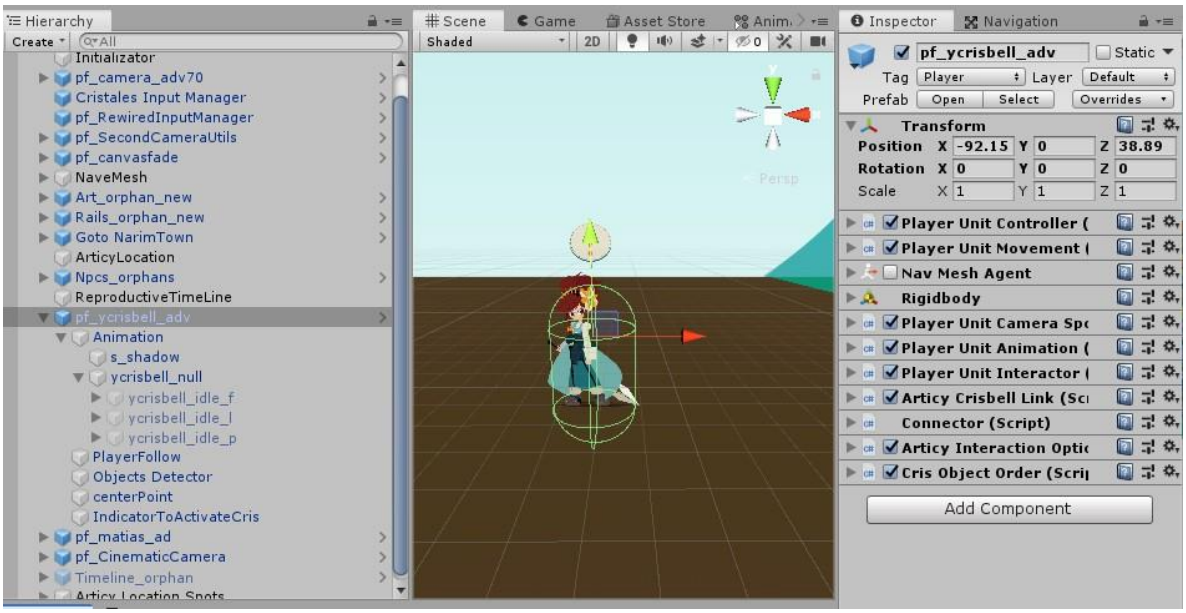
Fuente: Autor

Ilustración 46. Máquina del tiempo en Cristales



Fuente: Autor

Ilustración 47. Personaje ensamblado para Cristales



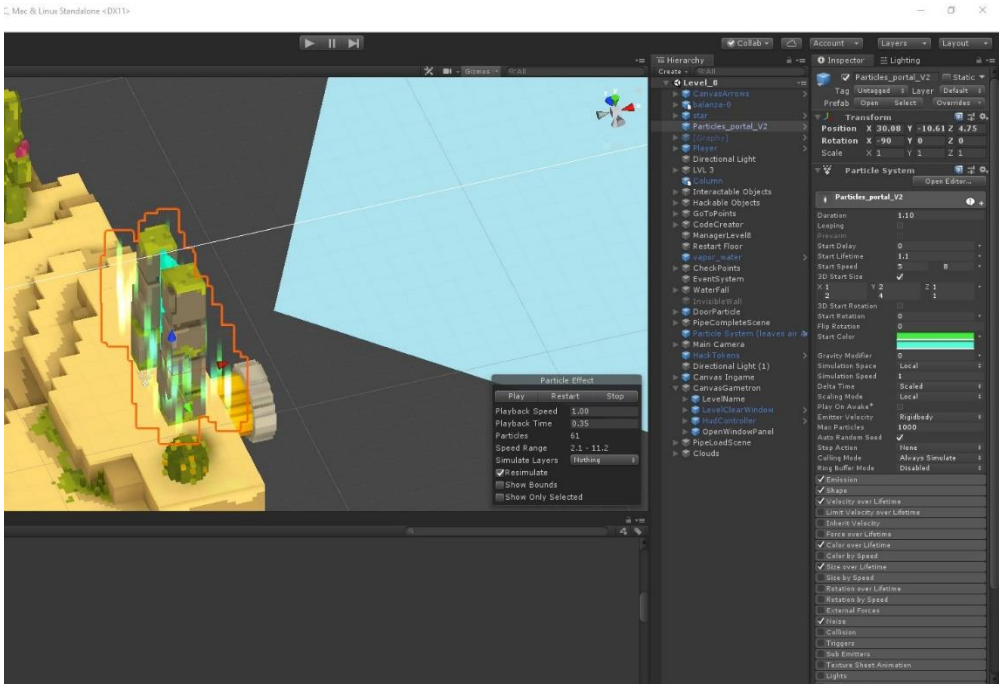
Fuente: Autor

10.3. SISTEMAS DE PARTÍCULAS

Se realizaron diversos sistemas de particulas en ambos proyectos, a continuacion se mostraran algunos de ellos, todos se encuentran en la version final de los proyectos.

En la siguiente ilustracion se observa un sistema de particulas el cual fue desarrollado para el proyecto de Voxels, este se ejecuta cuando el jugador completa el nivel y se desbloquea el portal de salida, este efecto cuenta con tres sistemas de particulas combinados para crear el efecto de explosion de energia en el suelo y de forma vertical.

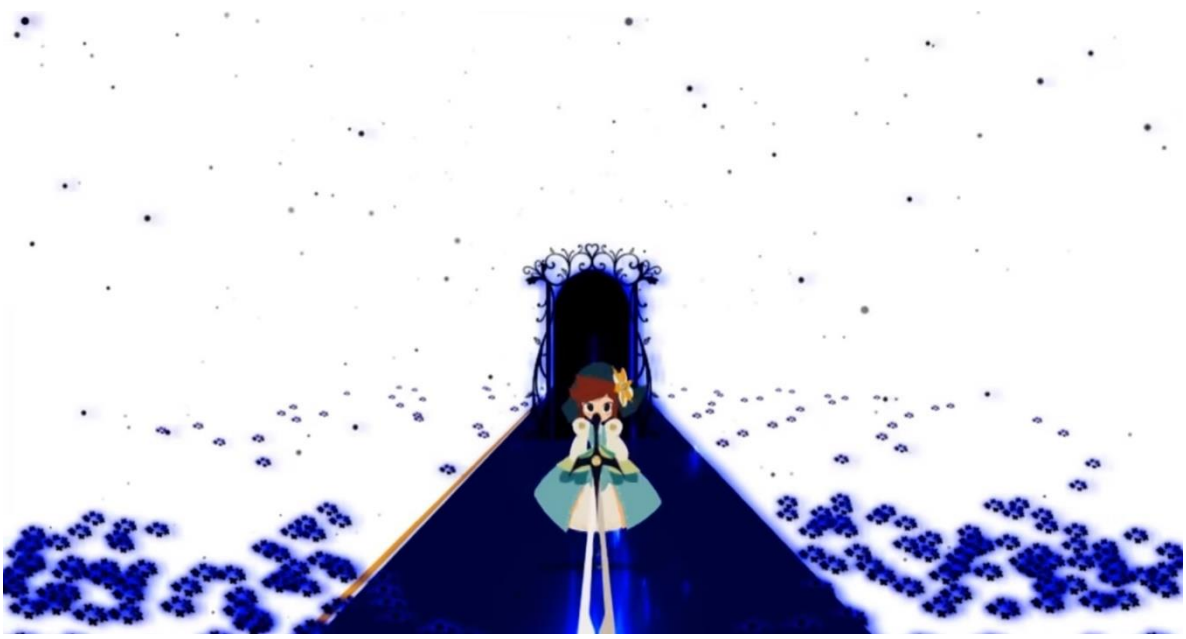
Ilustración 48. Sistema de partículas para Voxels



Fuente: Autor

En la siguiente ilustración se muestran unos sistemas de partículas los cuales fueron desarrollados para una timeline de Cristales, estos sistemas son la energía que brota de la espada y las partículas que flotan en el ambiente del escenario.

Ilustración 49. Sistema de partículas para Cristales



Fuente: Autor

10.4. TESTING

Como se describió en el capítulo de desarrollo el *testing* que se realizo fue en el proyecto de voxels en el cual se generaba el siguiente documento en base a las pruebas realizadas y los resultados obtenidos.

Ilustración 50. Documento de pruebas de rendimiento en el proyecto de Voxels

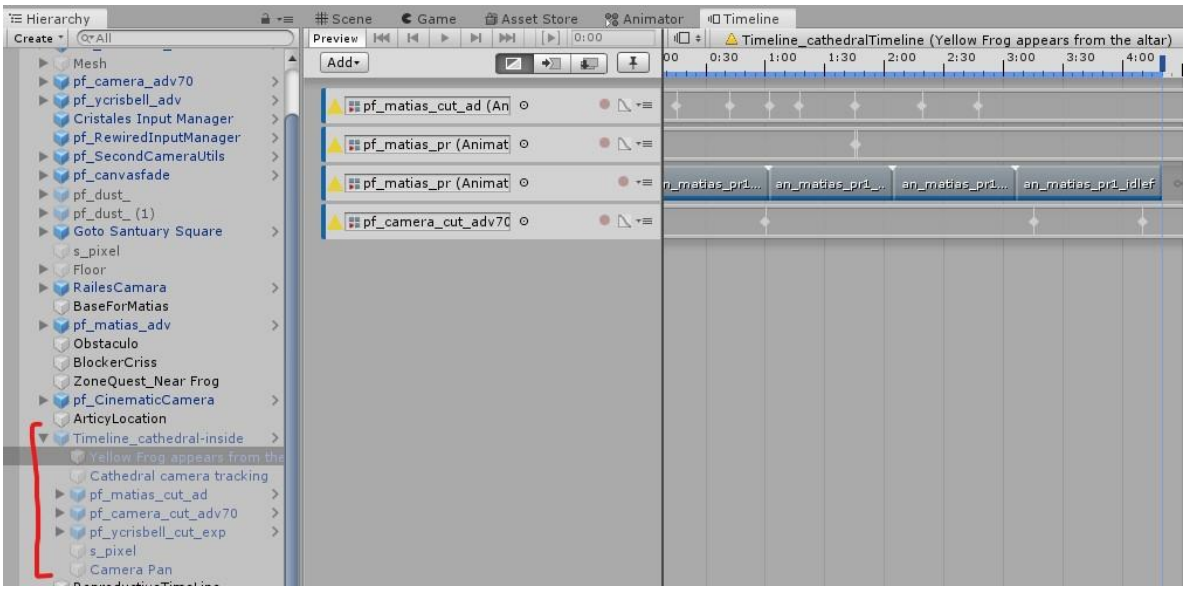
Proyecto Voxels		
Descripcion: Se trabaja con iluminacion en tiempo real		
Fps Min	Fps Promedio	Fps Max
15	20	26
Descripcion: Se reemplaza la iluminación global por los lightmaps		
Fps Min	Fps Promedio	Fps Max
34	41	62
Descripcion:Se combinan los lightmaps con una iluminacion global que solo afecte a la sombra del jugac		
Fps Min	Fps Promedio	Fps Max
21	27	35

Fuente: Autor

10.5. TIMELINE

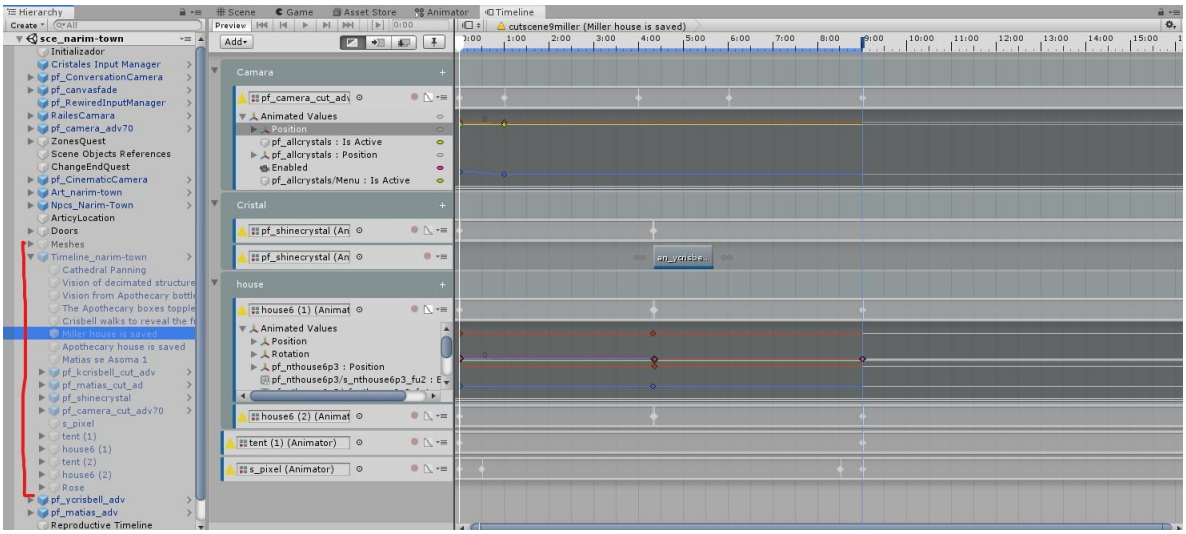
Las timeline fueron usadas en el proyecto de Cristales, y están fueron aprobadas por el *game designer* y el líder de programación, la mayoría de las timeline realizadas se encuentran dentro del proyecto y se pueden ver en el demo del videojuego el cual se encuentra en Steam. Todas las timeline de escena se organizaban dentro de un mismo componente vacío como se puede observar en las siguientes ilustraciones.

Ilustración 51. Timelines de la escena del interior de la catedral en Cristales



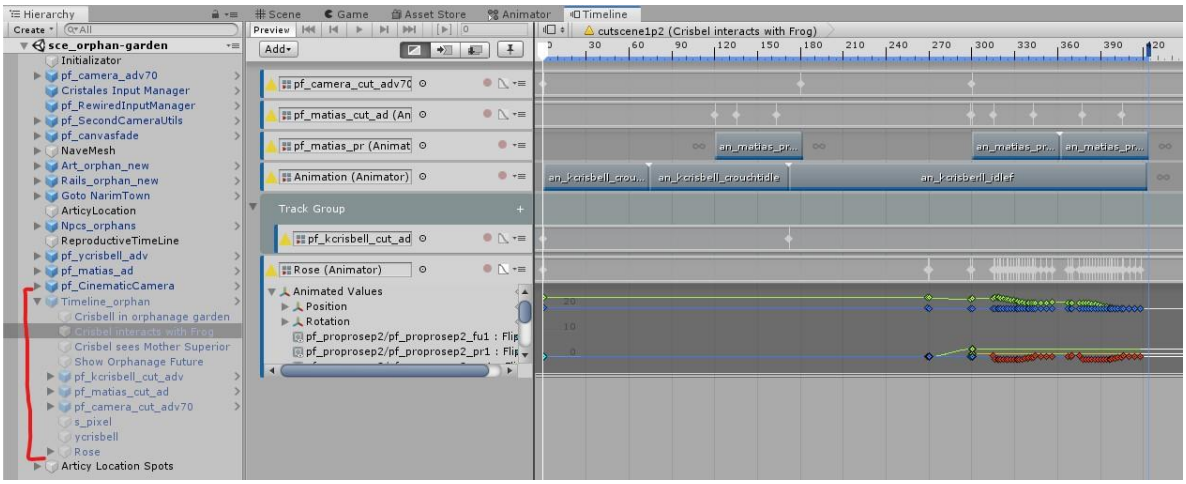
Fuente: Autor

Ilustración 52. Timelines de la escena de la ciudad principal en Cristales



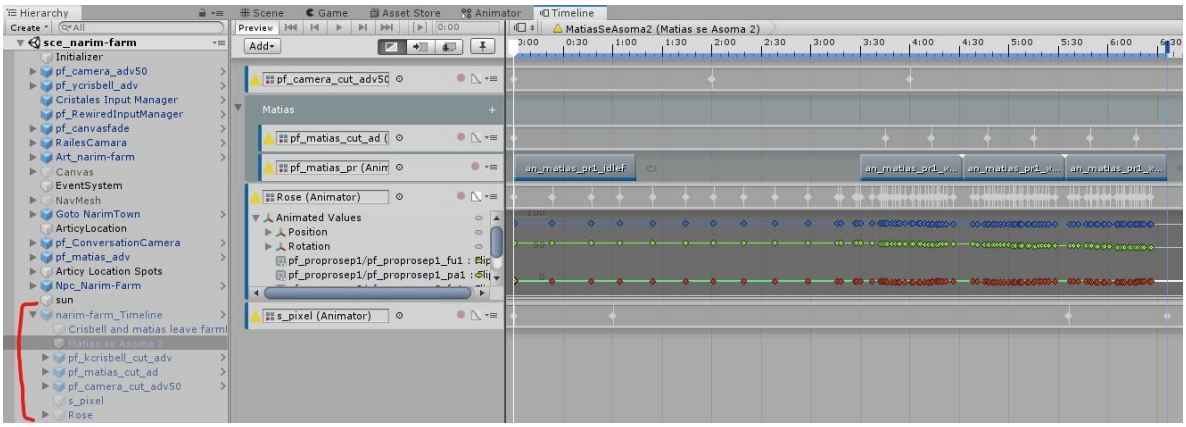
Fuente: Autor

Ilustración 53. Timelines de la escena del orfanato de Cristales



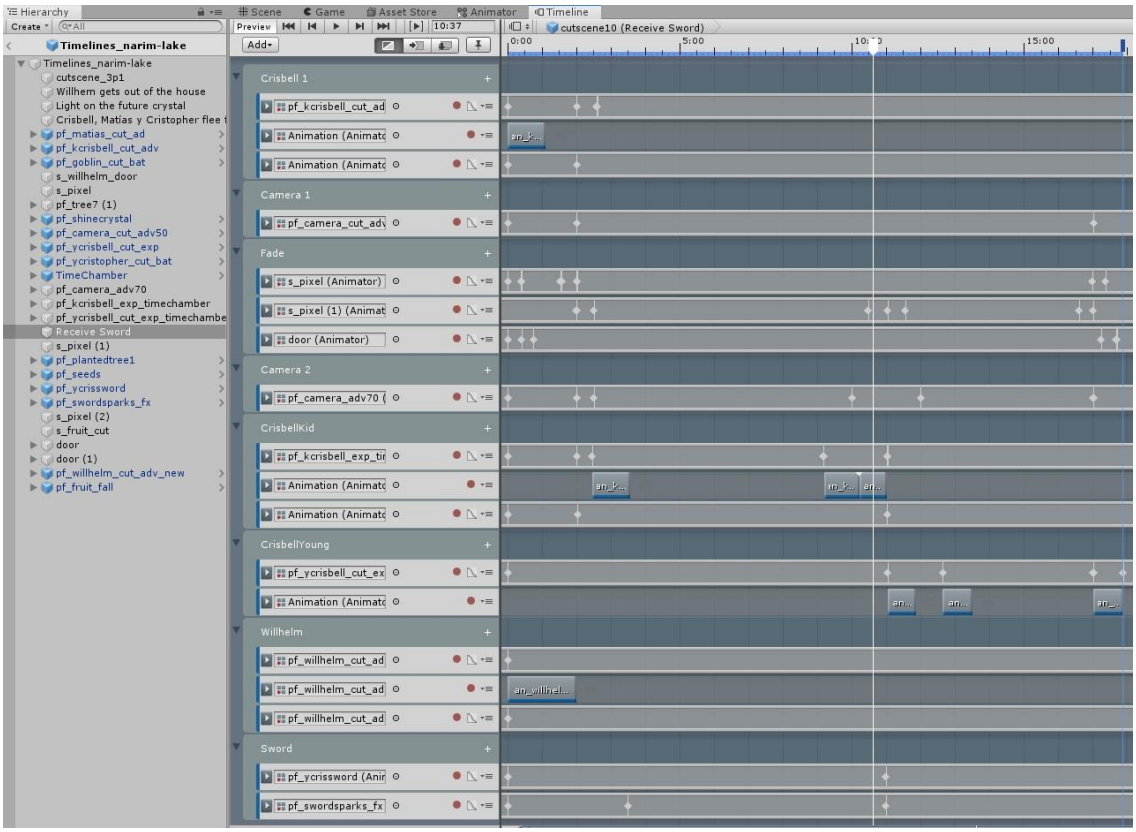
Fuente: Autor

Ilustración 54. Timelines de la escena de la granja de Cristales



Fuente: Autor

Ilustración 55. Timelines de la escena del lago en Cristales



Fuente: Autor

11. CONCLUSIONES

- Gracias al continuo estado de pruebas de las actividades desarrolladas, se pudieron detectar en una etapa temprana muchas funcionalidades innecesarias o errores de funcionamiento, lo cual permitió su corrección inmediata y el mantener siempre una calidad del proyecto dentro de los estándares establecidos para los mismos.

- En el desarrollo de Voxels a través de las pruebas de rendimiento que se realizaron que uno de los aspectos que más recursos de la maquina consume es la iluminación en tiempo real, y que al reemplazar esta iluminación por los *lightmaps* se podía mejorar el número de fps del videojuego en al menos un 30% en la mayoría de los casos.

- El uso de las *timeline* dentro del proyecto permitió ahorrar tiempo de desarrollo debido a que los artistas no tienen que realizar todas las animaciones a mano y paso a paso, las cuales solo serían usadas para dicha escena, los *timeline* también permiten reutilizar animaciones ya creadas para otras situaciones y modificar fácilmente dichas animaciones en caso de solicitar algún cambio en ellas.

- En el desarrollo de los scripts es necesario establecer que parámetros o atributos pueden ser editables en tiempo de ejecución debido a que esto permite que al momento de realizar las pruebas se puedan hallar más fácilmente los valores que se ajusten al comportamiento deseado, por ejemplo, permitir que sea editable la distancia de la cámara al jugador, para que cuando se estén ejecutando las pruebas esta distancia pueda ser editada y configurada a la distancia ideal.

12. REFERENCIAS

[1] P. Julián Pérez y G. Ana, «definicion.de,» 2013. [En línea]. Available: https://definicion.de/videojuego/.

[2] J. Newman, Videogames, second edition ed., Abingdon: Routledge, 2013.

[3] N. Padilla, C. Collazos, F. Gutiérrez y N. Medina, Ciencia e Ingeniería Neogranadina, Instituto de Investigaciones Jurídicas / UNAM, 2012.

[4] N. P. Zea, «Metodología para el diseño de videojuegos educativos sobre una arquitectura para el análisis del aprendizaje colaborativo,» 2011. [En línea]. Available: https://digibug.ugr.es/bitstream/handle/10481/19440/20058287.pdf?sequence=1&isAllowed=y. [Último acceso: 2019 12 15].

[5] A. C. Carrasco, «blogs.upm.es,» 4 Julio 2018. [En línea]. Available: https://blogs.upm.es/observatoriogate/2018/07/04/que-es-un-motor-de-videojuegos/.

[6] L. Alegsa, «Alegsa,» [En línea]. Available: http://www.alegsa.com.ar/Dic/modelo_en_3d.php. [Último acceso: 28 12 2019].

[7] Unity, «Unity docs,» 2018. [En línea]. Available: https://docs.unity3d.com/es/current/Manual/ParticleSystems.html.

[8] Unity, «Unity docs,» 2018. [En línea]. Available: https://docs.unity3d.com/es/current/Manual/AssetWorkflow.html.

[9] Fandom, «Wikijuegos,» 6 12 2018. [En línea]. Available: https://videojuegos.fandom.com/es/wiki/Sprite.

[10] A. R. Smith, «A sprite theory of image computing,» 2009-2010. [En línea]. Available: http://alvyray.com/Sprite/SpriteTheory_v4.2.pdf.

[11] Unity, «Unity,» 2020. [En línea]. Available: https://unity.com/products/core-platform. [Último acceso: 2019 12 16].

[12] Microsoft, «Microsoft docs,» 04 04 2019. [En línea]. Available: https://docs.microsoft.com/es-mx/dotnet/csharp/tour-of-csharp/index.

[13] Unity, «Unity documentation,» [En línea]. Available: https://docs.unity3d.com/Manual/TimelineSection.html. [Último acceso: 21 12 2019].

[14] D. Uncorpored, «ModusGames,» [En línea]. Available: https://modusgames.com/cris-tales/es/. [Último acceso: 25 11 2019].

[15] Unity, «Unity documentation,» [En línea]. Available: https://docs.unity3d.com/es/530/Manual/nav-NavigationSystem.html. [Último acceso: 2019 12 2].