TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

Procedural-Mind Generación de entornos 3D a través de la interacción natural con el usuario

Curso 2020/2021

Alumno/a:

Diego Cangas Moldes

Director/es:

José Antonio Piedra Fernández

No ha sido fácil llegar hasta aquí, no han sido pocas las adversidades ni las veces que me he querido rendir a mitad de camino, como tampoco lo han sido las alegrías y experiencias que he ganado durante todo este tiempo. Si algo tengo claro al echar la vista atrás, es que este ha sido y será un viaje que jamás hubiera logrado hacer solo. Quiero agradecer a todas y cada una de las personas que han formado parte de mi vida en algún momento, desde mis padres a mis profesores, desde mis amigos a mis rivales. A los que me han allanado el camino y a los que me han puesto piedras para que me tropezase. Cada persona que ha pasado por mi vida me ha marcado a su manera y me han convertido en alguien de quien yo mismo puedo sentirme orgulloso, aun así, de entre tantas personas que me han marcado, dos son las que destacan por encima del resto.

En primer lugar, quisiera agradecerte a ti, Isabel, mi amor, toda la confianza y cariño que me has dado. Me has ayudado a ver la vida de otra manera, a centrarme más en las personas y menos en las cosas. Gracias a ti, cada día soy una mejor persona. Estoy orgulloso de tus logros y progresos y espero poder recorrer contigo ese camino al que llaman vida.

Quiero agradecerte también a ti, Álvaro, mi hermano y mi mejor amigo. Por mostrarme todo el mundo que hay ahí fuera, por convertir esos sueños que consideraba inalcanzables en reales. Es gracias a ti que siempre he podido esforzarme para alcanzar lo que en otra época hubiese considerado imposible. Gracias por inspirarme y guiarme a llegar siempre más alto. Con cariño, tu hermano mayor.

ÍNDICE GENERAL

INDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTADOS

ABREVIATURAS

ESI	Escuela Superior de Ingeniería
TFG	Trabajo fin de grado
UAL	Universidad de Almería
XML	Extensible Markup Language
FPS	Frames per Second
GPU	Graphics Processing Unit
CPU	Central Processing Unit
AMD	Advanced Micro Devices
PHP	PHP: Hypertext Preprocessor
HTML	HyperText Markup Language
JSON	Javascript Object Notation
API	Aplication Programing Interface
HTTP	Hypertext Transfer Protocol

RESUMEN Y ABSTRACT

Uno de los problemas a los que tiene que enfrentarse la industria del videojuego y del entretenimiento hoy en día es a los altos costes del desarrollo de escenarios en 3D derivado del gran tiempo que tienen que dedicar los desarrolladores a esta tarea.

Este trabajo busca agilizar esta tarea generando escenarios de forma procedural que, además, son a la medida de cada persona.

El sistema utiliza una basta base de conocimiento de objetos que podrá ser usada tanto por los desarrolladores como por personas menos puestas en la materia (artistas, directivos, clientes, etc.) para generar y recolectar a través de la voz lo que el usuario desee crear. Además, la voz no es la única fuente de datos del sistema, puesto que este también hace uso del reconocimiento de imágenes (para extraer características de los objetos que la componen) o de la geolocalización del usuario.

El programa tiene la capacidad de definir un entorno espaciotemporal para los objetos que se generan a partir de un objeto base. De esta manera en función de la geolocalización (por defecto la ubicación del usuario) y la época elegida, los objetos que se generen serán distintos, a pesar incluso de que sean las mismas peticiones. Por ejemplo, si se pide un coche y se marca como época el sigo XVIII, lo que se regenerará será un vehículo a vapor.

Finalmente, el sistema implementa algoritmos de generación de ríos, terrenos, caminos y objetos, que se adaptan a los escenarios previamente creados.

Palabras clave: Generación personalizada, extracción de datos, reconocimiento de imágenes y voz, Entornos 3D, Dijkstra, Bézier, Inteligencia Artificial.

One of the issues that the video game and entertainment industry must face nowadays is the high costs of developing 3D environments derived from the huge time that developers must spend in this task.

This work seeks to streamline this task by generating procedural environments that are also tailored to each person.

The system uses a vast knowledge base of objects that can be used by both, developers and people less knowledgeable (artists, managers, clients, etc.), to generate and collect through the voice what the user wants to create. In addition, voice is not the only source of data in the system, since it also makes use of image recognition (to extract characteristics of the objects that compose it) or of the geolocation of the user.

The program can define a space-time environment for objects that are generated from a base object. In this way, depending on the geolocation (by default the user's location) and the chosen time, the objects generated will be different, even though they are the same requests. For example, if a car is ordered and the 18th century is marked as a period, what will be regenerated will be a steam vehicle.

Finally, the system implements algorithms for the generation of rivers, terrain, roads, and objects, which are adapted to the previously created scenarios.

Keywords: Custom generation, data extraction, image and speech recognition, 3D environments, Dijkstra, Bezier, Artificial Intelligence.

1 INTRODUCCIÓN

1.1 Planteamiento del problema

Uno de los principales problemas que enfrenta la industria tecnológica hoy en día a la hora de recrear simulaciones foto-realistas, es la del elevado coste del diseño y el modelado 3D al que se enfrentan las empresas, los datos revelan que los presupuestos en videojuegos de esta parte pueden llegar a ser un 25% de los costes totales del producto o un 60% de los coses relacionados con la producción y el desarrollo [1]. Cantidad nada despreciable teniendo en cuenta que algunos videojuegos tienen unos costes de producción superiores al de muchas películas de Hollywood, por ejemplo, el videojuego 'GTA 5' (2013) tuvo unos costes de desarrollo de 137 millones de dólares [2], lo que resultaría en unos costes de diseño y modelado de aproximadamente 80 Millones de dólares. Pero el modelado 3D no solo se usa en videojuegos, series como "The Mandalorian (2020)" de Disney han hecho uso de la tecnología de vanguardia 'Stagecraft' para la renderización escenarios 3D a tiempo real sobre el rodaje haciendo uso una pantalla gigante que sirve como sustitución al croma [3], tecnología que, a nivel de coste, le ha supuesto a Disney un gasto de 15 millones de dólares por capítulo.

Todos estos números reflejan un enorme gasto en recursos humanos y tiempo a la hora de desarrollar escenarios y modelos 3D realistas, lo que aleja mucho esta tecnología de su uso para el público general. Esto puede que hará unos años fuese algo irremediable, pero en la actualidad gracias a la generación de los amplios volúmenes de datos que se generan a diario podemos saber y procesar de forma inteligente el escenario virtual que el usuario desea sin necesidad de convertir esta tarea en algo largo y tedioso. Como ejemplo empresas como Adobe ya están empezando a utilizar sistemas basados en inteligencia artificial para simplificar las tareas de diseño y edición en fotografías e imágenes en 2 dimensiones [4].

1.2 Objetivos del proyecto

El principal objetivo es desarrollar un sistema inteligente que obtenga datos e información del usuario para ser capaz de generar un entorno tridimensional acorde con los deseos y costumbres del propio usuario, realista y próximo a sus pensamientos. Para alcanzar este objetivo, el proyecto se dividirá en 3 partes independientes, cada una con sus propios objetivos:

En primer lugar, la Extracción de datos de Usuario, se encargará no solo de extraer y analizar los datos que el usuario comunica, sino que deberá también extraer más información del propio usuario. Es decir, obtendrá toda la información relevante que pueda de este, como por ejemplo su ubicación o su cultura social, esto es relevante ya que, si por ejemplo el usuario desea colocar un pueblo en el escenario, ¿debería ser de arquitectura occidental? ¿Oriental? ¿asiática tal vez?,

si el usuario pide un bosque, y vive en Almería, su definición de bosque será la de uno frondoso repleto de pinos o, se asemejará más bien, a lo que ha visto dentro de su entorno. Como se puede ver, distintas personas pueden tener una distinta imagen mental de una misma palabra. Es por ello, que el algoritmo deberá reconocer y procesar toda la información útil del usuario y comprimirla en uno o varios archivos que el sistema de generación pueda leer.

La Base de Datos de Conocimiento es otro factor relevante, debido a la enorme cantidad de información que puede haber (¿Cuántos tipos de árboles o de vehículos diferentes pueden existir por ejemplo?), es necesario contar con una o varias bases del conocimiento, las cuales pueden generarse de forma automática, semiautomática o manual, y su granularidad dependerá de los intereses del usuario. Además, estas bases de conocimiento pueden ser incluso hasta temáticas (podemos tener una base de conocimiento que busque una recreación fidedigna de la realidad, otra que genere un estilo visual más de dibujos animados u otra que haga uso de datos fantásticos que no existen en nuestro mundo), estas bases de conocimiento al final, al igual que en el caso anterior, deberán generar uno o varios archivos (con formato estándar) que puedan ser leídos por el sistema de generación. Esta arquitectura en 3 partes permite una gran modularidad y personalización del sistema de acuerdo con los deseos del usuario o de la organización a al que este pertenezca y permite la modificación de las bases del conocimiento sin tener siquiera que modificar la mayoría de las veces el algoritmo de generación.

Finalmente, el objetivo de la construcción del entorno 3D será la de recibir los datos de los otros dos sistemas y recrear de la manera más fidedigna posible los deseos del usuario de acuerdo con toda la información extraída de este y de las bases de conocimiento que se estén usando en ese momento (Figura 1.1).

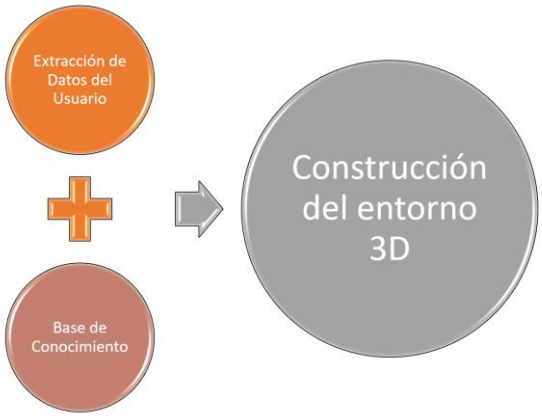


Figura 1.1. Estructura del sistema

Otro objetivo transversal de este proyecto reside en el reclamo y la concienciación de que distribución voluntaria por parte del usuario de información, a través de sistemas como las como su localización o datos personales entre otros, pueden tener un uso desconocido o incluso nocivo para este mismo, como puede ser el de procesar sus imágenes u obtener su ubicación, que, aunque en este proyecto se use para brindarle al usuario un servicio positivo, no sabemos con que fines podrían utilizarlo otras organizaciones y/o sistemas.

1.3 Contribución

Este trabajo contribuye al panorama de herramientas actuales en la utilización de datos del usuario (imágenes de referencia, geolocalización, reconocimiento de voz) para la generación del escenario. Además, al dividir el sistema en dos partes, una de extracción de datos y otra de generación permite que dos personas diferentes manejen cada una de ellas, dando como resultado que la persona que haga uso del sistema de extracción de información pueda ser, por ejemplo, un directivo o un artista que no tenga conocimientos de diseño de niveles.

El sistema de generación es también en sí mismo bastante novedoso puesto que intenta funcionar como un sistema general, a diferencia de los habituales sistemas de generación procedural que se centran en generar aspectos muy concretos, como sistemas de generación de bosques procedurales o de combinación de escenas (habitaciones) de forma procedural. El sistema además es multicapa pasando por varios procesos de generación antes de entregar el resultado final, algunas de estas capas (como la superficie del terreno) son controladas por el desarrollador, otras, sin embargo, depende de los datos generados por la herramienta de recolección de datos y finalmente otras son calculadas directamente por el sistema en base a todo lo anterior.

Otro de los aportes de este proyecto al campo, es el uso de una base de datos del conocimiento como base para generar los objetos, ya que esto permite aumentar el contenido del programa sin tan siquiera tener que tocar el código e incluso permite desarrollar aplicaciones que auto rellenen esta base de datos de manera procedural o automática, pudiendo hacerlo incluso extrayendo datos del mundo real. Sobre este último respecto, el sistema puede vincular los objetos que posee dentro de una geolocalización (cuyo esquema de coordenadas depende del desarrollador) y de una variable temporal. Lo que en la práctica se traduce en un cubo o una matriz tridimensional por la que el desarrollador puede moverse para que con un mismo archivo de datos el contenido de generación pueda cambiar drásticamente. Esto permite, por ejemplo, vincular la variable temporal a un objeto de manera, como podría ser una columna romana (con varios modelos 3D según su nivel de deterioro introducidas como variantes) de manera que, si se marca la época, en la época romana, esta columna se genere intacta en perfectas condiciones, pero que por el contrario si se marca una época posterior como la edad media, el estado de dicha columna sea un estado ruinoso. Esto dota de una gran libertad a los desarrolladores para generar sus mundos donde por supuesto podrán usar objetos de épocas y lugares distintos a los que tienen marcados siempre y cuando así lo especifiquen.

1.4 Organización del trabajo

Este trabajo está dividido en 8 partes junto con 2 anexos que guiarán al lector a lo largo y ancho de la solución propuesta. Este primer capítulo ofrece una visión general del problema que se quiere resolver, los objetivos que se quieren alcanzar con este proyecto y una visión general de las aportaciones que ofrece este trabajo al panorama actual del sector.

En el segundo capítulo, denominado Estado del Arte, se busca darle al lector una visión de la actualidad en cuanto a las tecnologías y los sistemas que se usarán en este proyecto. Asi como alternativas a las herramientas utilizadas en este proyecto. Por lo que el objetivo principal de esta sección será la de poder darle al lector toda la información necesaria para que comprenda el funcionamiento y el contenido de las secciones posteriores. Esta sección además está dividida en 4 subsecciones. La primera centrada en entender que es un modelo3D (la base del proyecto),

de que está compuesto y como se puede trabajar con ellos. La segunda sección profundiza en el nivel de abstracción para hablarnos de los motores de renderizado en tiempo real, de cómo estos funcionan y de por qué son diferentes a los motores de renderizado tradicionales, así como también ahonda en los sus distintos tipos para finalizar citándonos los más populares actualmente. La tercera subsección se centra en la generación procedural de escenarios, cuál es su finalidad y su origen, así como de las diversas alternativas que pueden encontrarse en el mercado. Finalmente, la última subsección se centra en la comunicación entre las distintas partes del proyecto y en los estándares que hay detrás de estas comunicaciones defendiendo el uso de las alternativas elegidas para este proyecto y su por qué.

El tercer capítulo, denominado Etapas del Desarrollo, ofrecerá una visión general de las etapas del proyecto dando una breve explicación de cada una y una comparación de los tiempos estimados y los tiempos reales del proyecto.

El cuarto capítulo, denominado Análisis, es el encargado de explicar en profundidad la solución escogida, y de especificar no solamente la propia solución, si no también otros aspectos transversales como es el análisis de costes estimados del proyecto. Este capítulo, al igual que el segundo, se subdivide esta vez en 5 secciones, la primera de ellas, denominada Descripción detallada de la solución, nos ofrecerá como su propio nombre indica, una visión profunda del proyecto a abordar. La segunda sección nos ofrece por otro lado de forma rápida y visualmente el diagrama de casos de uso del sistema que usará la solución, haciendo hincapié en cada uno de los susodichos casos de uso. A continuación, la tercera y cuarta sección nos introducirán en los requisitos funcionales y no funcionales respectivamente, dándonos una explicación de que son y cuales podemos encontrar en este proyecto. Finalmente, la última sección se centrará en el análisis económico y de costes del desarrollo de todo el proyecto, desde su concepción hasta su finalización y despliegue.

El quinto capitulo, denominado tecnologías y herramientas, se encargará de enumerar y explicar las distintas herramientas y tecnologías que han sido utilizadas en el proyecto, así como su utilidad para el correcto desarrollo de este. Cabe destacar que se han obviado aquellas relacionadas exclusivamente con el desarrollo de este documento y que no han tenido impacto alguno en el resto del proyecto.

El sexto capítulo, denominado Desarrollo, funciona como el eje central del trabajo y se encargará de detallar exhaustivamente cada una de las partes del trabajo realizado. Al igual que otros capítulos, este también se encuentra dividido en varias secciones, concretamente en dos, que son arquitectura e implementación. Siendo la primera una extensa explicación del funcionamiento de la arquitectura del proyecto. Finalmente, la segunda la más extensa de ambas se centra en como el proyecto ha pasado de ser un diseño a una herramienta software completamente funcional.

El séptimo capítulo y el penúltimo de ellos, denominado Despliegue, se centrará en la fase final del proyecto, en como este se transforma de un proyecto software a un archivo ejecutable que permite de instalarlo en cualquier ordenador. Además, la segunda parte de este capítulo se centra en la prueba del software con los usuarios, y los resultados obtenidos a partir de esta.

Por último, para terminar con los capítulos, encontramos el capítulo ocho, denominado conclusiones, en el cual se hará una valoración final del trabajo desarrollado, así como una

explicación de los conocimientos adquiridos al realizar este proyecto y finalmente se plantarán las semillas de todo aquel trabajo futuro que se podrá realizar a partir de los diseñado en este documento.

Una vez finalizados estos siete capítulos, el lector puede dar por concluida su lectura, pero además de estos, se ha querido añadir al final un total de 2 anexos que complementan todo lo anteriormente presentado.

El primero de ellos corresponde a los manuales de usuario de las distintas partes del proyecto que el lector podrá usar en caso de que se aventure a intentar utilizarlo. El segundo anexo corresponde al modelo de cuestionario usado para verificar la utilidad de la herramienta en profesionales y cuyos resultados pueden encontrarse en el capítulo 6.3.

2 ESTADO DEL ARTE

2.1 Modelado 3D

Se entiende por modelado 3d el proceso de desarrollo de una representación matemática de cualquier objeto tridimensional (animado o no) a través de un software especializado. Al resultado se le conoce como "Modelo 3D" y se puede visualizar como una imagen bidimensional a través de un proceso conocido como "renderizado 3D", existen tecnologías como la Realidad virtual que nos permiten ver el objeto en tres dimensiones pero que en ultima estancia siguen siendo renderizados puesto que lo que en realidad generan son 2 proyecciones bidimensionales del objeto, una para cada ojo. Los Modelos 3D no solo pueden ser creados a mano por un diseñador, también pueden ser generados automáticamente haciendo usos de algunas técnicas como la fotogrametría [5] (véase 2.1.3). Adicionalmente, los modelos 3d también se pueden convertir al mundo real a través de impresión 3D (véase figura 2.1)

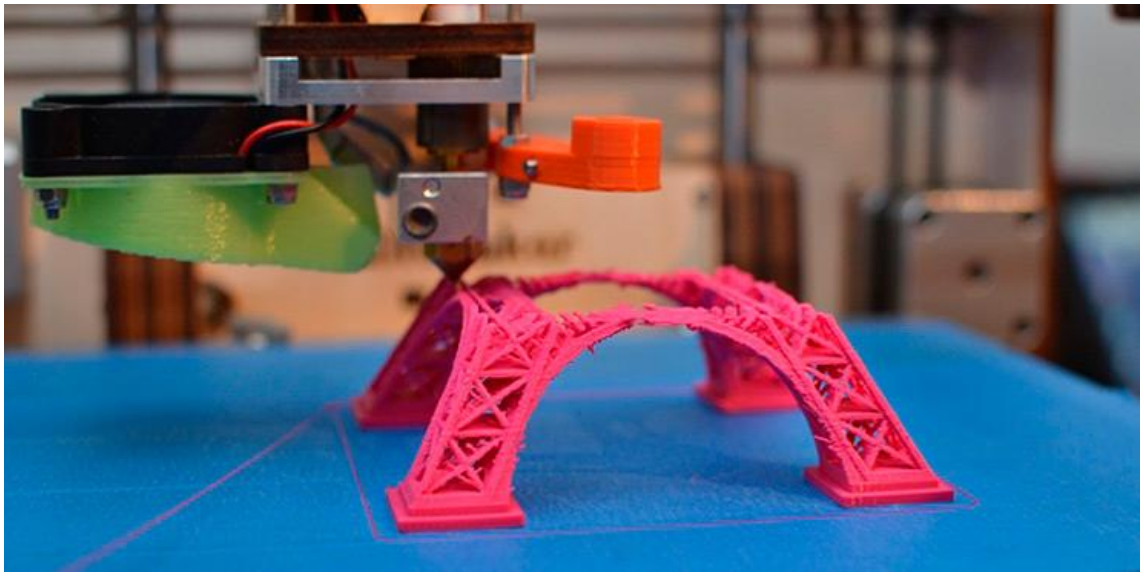


Figura 2.1. Impresión de un Modelo 3D haciendo usa de una impresora 3D cartesiana

Los modelos 3D se conforman por una serie de caras o "polígonos" (usualmente triángulos) que componen toda la estructura. Estos polígonos componen el "mesh" o maya del objeto y son sobre los que luego se aplicarán los cálculos de iluminación o profundidad durante el proceso de renderizado. Los polígonos de una figura determinan en última instancia el tiempo que se tardará en renderizar el objeto, cuanto más polígonos tenga un objeto más tiempo tardará el

ordenador en renderizarlo. Esto cobra suma importancia más adelante en los “sistemas de renderizado a tiempo real”.

Para generar los modelos 3D, en la actualidad se siguen varias técnicas que detallaremos a continuación:

2.1.1 Modelado poligonal

Esta técnica fue la primera en desarrollarse y consiste en trabajar directamente sobre los polígonos haciendo uso de operaciones básicas de geometría sobre ellos, rotación, translación, extrusión, etc. u operaciones más complejas como suavizado de contornos, subdivisión de caras, etc. Hasta hace relativamente poco tiempo dados los recursos limitados de los que disponían los ordenadores, esta era la técnica más usada con amplia diferencia, aunque hoy en día por el aumento de las prestaciones de los sistemas, la técnica de escultura de modelos está ganando algo más de popularidad. Existe una gran variedad de software que trabajan con esta técnica como pueden ser “Maya”, “3D Studio” o “Blender”.

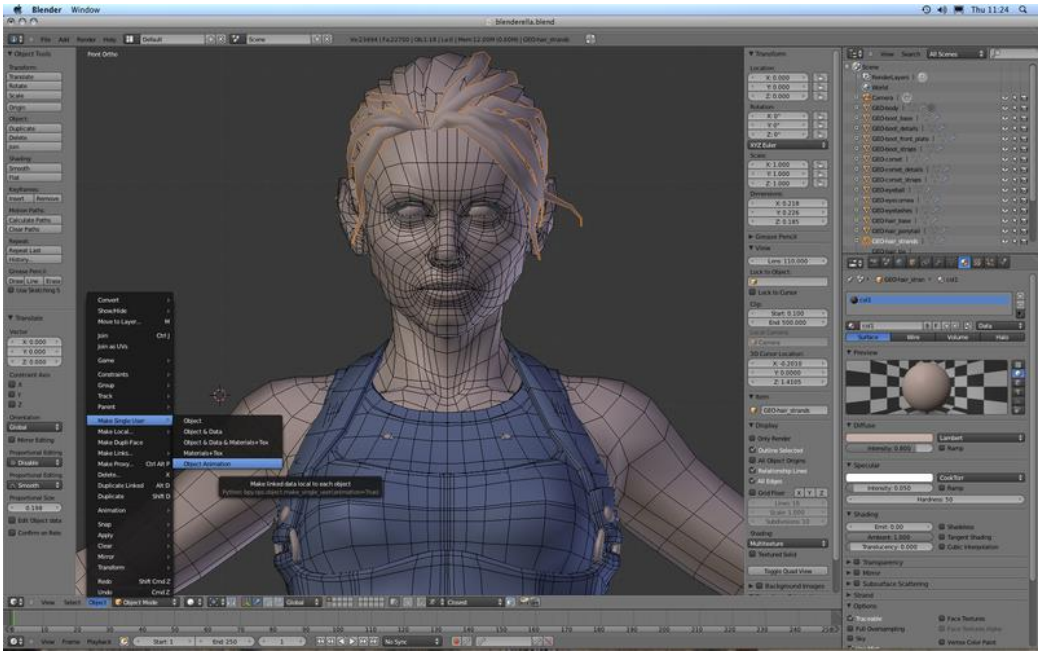


Figura 2.2. Diseño de un modelo 3D poligonal en Blender

2.1.2 Escultura de modelos

Es una técnica relativamente más nueva que la anterior y consiste en tratar al modelo 3D como si de una escultura de arcilla se tratase, aplicando sobre ella las transformaciones típicas de un escultor, como cincelado, lijado de superficies, estirado de superficies, etc. Finalmente es el propio programa el encargado de generar los polígonos de la maya.

El principal software que desarrolla esta técnica es “ZBrush” un software desarrollado por la empresa “Pixologic” y lanzado por primera vez en 1999 [6].

Figura 2.3. Diseño de un Modelo 3d en Zbrush haciendo uso de la técnica de escultura

2.1.3 Fotogrametría

La fotogrametría es una técnica consistente en fotografiar o "escanear" mediante un escáner 3D objetos del mundo real, estos se captan a través de una cámara a color o una infrarroja y después a través de un algoritmo se genera el modelo 3D del objeto deseado. Esta técnica facilita enormemente el trabajo de diseño puesto que no se necesitan apenas conocimientos para usarla, pero, así como esto es cierto, también lo es el hecho de que es una técnica que plantea varios problemas, como el hecho de la gran cantidad de polígonos que se generan a partir del objeto (normalmente más de 1M), lo que hace que los modelos basados en esta tecnología estén muy poco optimizados y sean muy lentos de renderizar. El otro principal problema es, que, al ser una técnica basada en la captación de imágenes o fotografías, tiene muchos problemas al tratar con superficies reflectantes como espejos o metales pulidos donde la luz pueda rebotar. Es por ello, que esta técnica no está especialmente extendida o se suele usar para generar un modelo base que luego el diseñador utilizará para generar el suyo propio haciendo uso de las otras técnicas mencionadas. En la figura 2.4 se puede apreciar un modelo 3D generado por fotogrametría donde cada uno de los "rectángulos" que hay alrededor del modelo son cada una de las fotografías tomadas desde distintos ángulos del objeto a generar.

Figura 2.4. Modelo 3D generado a partir de la toma de múltiples fotografías (fotogrametría)

2.1.4 Composición de objetos

Finalmente, la composición de objetos es la última técnica que se utiliza es la de composición de objetos, esta es una técnica que no puede usarse por si sola, puesto que consiste en combinar objetos 3D ya prediseñados para generar modelos o escenarios más complejos, por ejemplo, a partir de varios modelos de ramas de árbol y un modelo de una hoja, se puede combinar para generar el modelo 3D de un árbol, o usando varios modelos 3D de casas y carreteras se puede generar el modelo 3D de una ciudad. Esta técnica es la que ahondaremos y desarrollaremos en el presente trabajo

Todas estas técnicas pueden desarrollarse por separado o combinarse especialmente para la creación de modelos 3D complejos.

2.2 Motores de renderizado en tiempo real

Los motores de renderización en tiempo real, también conocidos como motores de videojuegos actualmente, son aquellas herramientas cuyo fin es ejecutar la tarea de renderización 3D en un tiempo aceptablemente bajo (por debajo de los 33 milisegundos). A cada iteración de la renderización se la conoce como “frame”, para tener una experiencia fluida se recomienda que el número de frames que se estén rendrizando por segundo sea siempre igual o superior a 60. Aun así, con una tasa de 30 FPS la experiencia ya se considera aceptable.

Para alcanzar estas cotas de velocidad, normalmente el motor de renderizado en tiempo real no suele usar las mismas técnicas que un motor de renderizado tradicional como pueden ser Blender o ZBrush ya que en un motor tradicional el tiempo de renderizado es indiferente puesto que no es a tiempo real, un ejemplo de ello es la película “El libro de la selva (2016)” de Disney donde para renderizar cada imagen (frame) se llegaron a necesitar 80 Horas. [7]

A continuación, detallaremos el proceso que sigue el motor de renderizado a tiempo real:

1. Carga de renders. El software busca todos los renders activos de una escena y los enlista para renderizarlos.

2. Eliminación. Se eliminan todos los renders que se encuentran fuera de los límites visibles de la cámara o que se encuentran en una capa de renderización distinta a la de la cámara.
3. Recopilación datos de los Renders. En este punto el motor busca por todos los archivos necesarios para renderizar cada objeto (mallas, texturas, sombreadores, variables adicionales, etc…)
4. Carga de los datos. El motor comprueba que si es la primera vez que ve alguno de los archivos y de ser así los carga en la memoria de la GPU. Muchos de estos datos una vez se cargan en la GPU se borran de la memoria de la CPU por temas de espacios.
5. Ordenado. Ahora se ordenan los objetos en orden de renderizado, por ejemplo, según distancia, para que los objetos cercanos se rendericen sobre los lejanos.
6. Configuración del Renderizado. Esta es una de las partes más interesantes del proceso, el motor le indica a la GPU los datos que son compartidos entre objetos (materiales, shaders, etc.) o entre frames (matrices de transformación de un objeto o del cámara).
7. Renderizar objetos opacos. El motor renderiza (muestra la proyección bidimensional de los objetos sobre un punto de vista (cámara)) los objetos opacos ordenadamente.
8. Renderizado Skybox. Ahora es el turno de renderizar la malla del skybox (o el cielo para simplificar) que tiene una forma de cúpula lejana.
9. Renderizar objetos transparentes. Ahora el motor renderiza al igual que antes, la lista de objetos transparentes en orden. Cabe destacar que el coste de renderización de estos objetos es superior al de los objetos sólidos.

A pesar de que la tarea de renderización es el valor central de los motores de renderizado a tiempo real, existen una gran diversidad de otro tipo de tareas que estos implementan para facilitarle el trabajo a los desarrolladores como puede ser la gestión automática de personajes humanoides, máquinas de estados para animaciones o editores gráficos de shaders.

Como se puede observar, el proceso de renderizar una escena es complejo y lento, por ello en este tipo de motores se suele hacer uso de trucos para renderizar las escenas, principalmente en la iluminación, que es una de las tareas que más recursos consume, aunque es cierto que en los últimos años se está consiguiendo una mejora exponencial en los resultados de la iluminación debido a que las tarjetas gráficas están empezando a incluir hardware (Hardware dedicado para raytracing, RT Cores en el caso de Nivdia, Ray Acelerators en el de AMD) dedicado para esta tarea, aun así el número de rayos de luz que se renderiza sigue siendo bastante menor que el que se usa en los motores de renderizado tradicionales. En la figura 2.5 puede verse la diferencia entre el uso de algoritmos tradicionales de renderizado de imagen contra el uso del raytracing para calcular la iluminación y los reflejos.

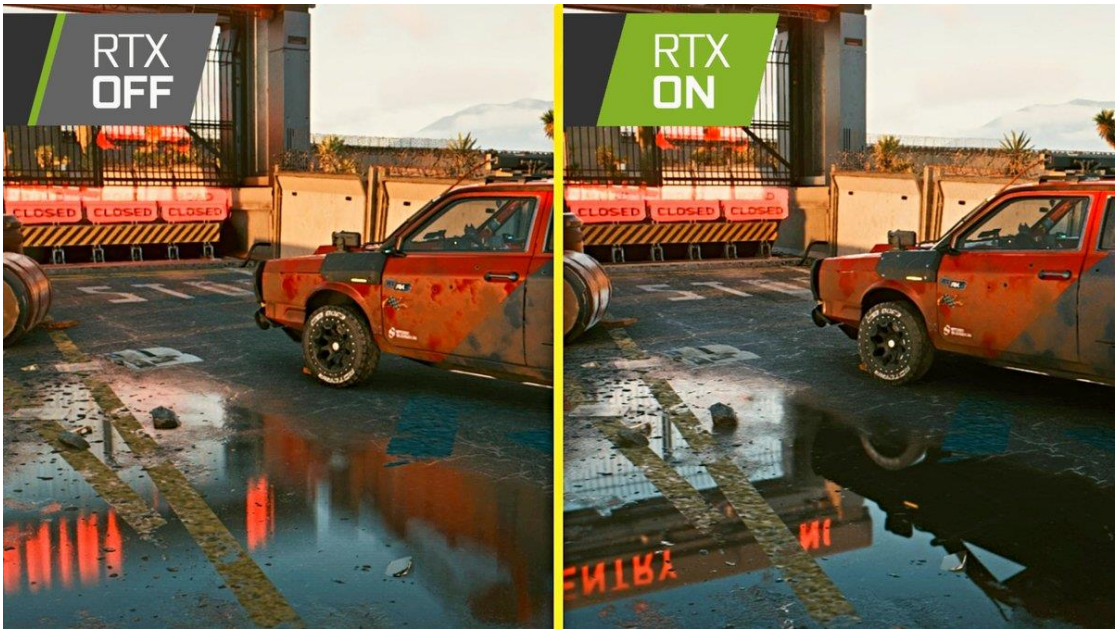


Figura 2.5. Iluminación tradicional vs raytracing

Actualmente los motores de renderizado a tiempo real se pueden clasificar en 2 grupos principalmente:

- Los motores propietarios. Pertenecen a una empresa y no pueden utilizarse fuera de estas. Son los más extendidos, ejemplos de este tipo serían el motor Frostbite (Electronic Art) o Snowdrop (Ubisoft).
- Los motores de acceso libre. Pertenecen y son desarrollados por empresas privadas como los anteriores, pero a diferencia de estos últimos, cualquier desarrollador tiene acceso a utilizar estos motores para sus proyectos, ya sea gratuitamente o pagando una cuota o una subscripción. Ejemplos de estos son Unity3D (Unity), Unreal Engine (Epic Games) o Godot (OKAM Studios), siendo además este último un motor de software libre.

Dada la naturaleza de este último tipo de motores, este trabajo estará orientado para este tipo, concretamente para Unity3D. que es actualmente junto con Unreal Engine el motor más utilizado por los desarrolladores en la actualidad. Finalmente, en la figura 2.6 se puede apreciar la diferencia de renderizar 2 imágenes, una con un motor de renderizado a tiempo real (Unity3D) y otra en un motor clásico (VRay). Como se puede apreciar, incluso siendo la misma escena (una bastante pequeña y rápida de ejecutar), el motor tradicional genera un mejor resultado, sobre todo en lo referente a la iluminación (véase por ejemplo el reflejo de la mesa).

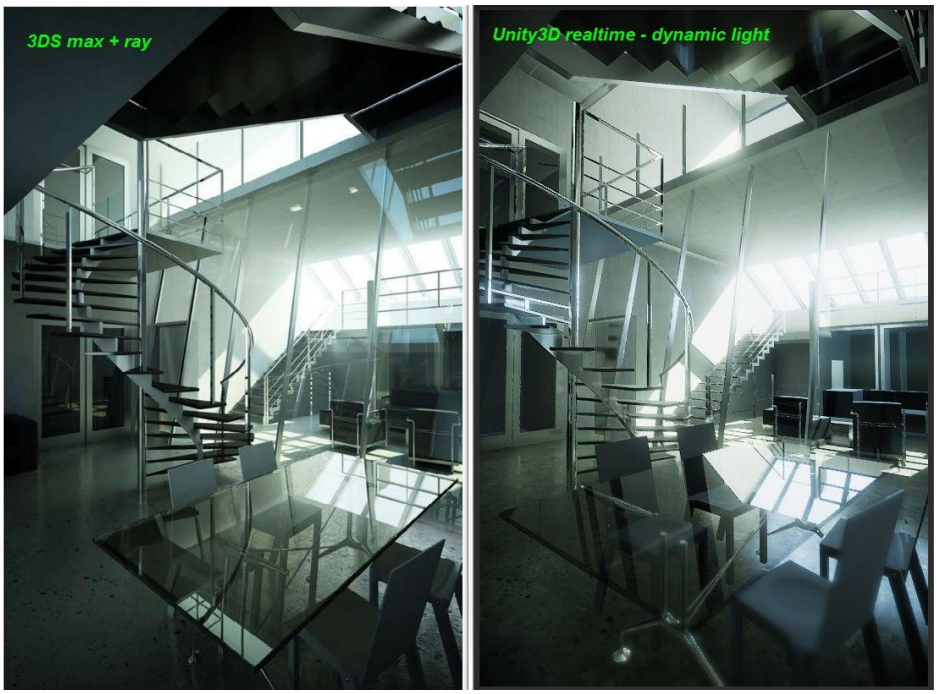


Figura 2.6. Comparativa entre motores

A pesar de que el motor tradicional sigue siendo mejor que el de tiempo real, gracias al aumento de potencia de las tarjetas gráficas, estos últimos se acercan más a los primeros. Basta con hacer una comparativa de la evolución de la potencia de cálculo entre tarjetas gráficas (Tabla 2.1) para darse cuenta de ello.

Tarjeta gráfica	Potencia de cálculo en FP32 (TFLOPS)	Año de lanzamiento
GeForce GTX 480	1.345 TFLOPS	2010
GeForce GTX 590	2.488 TFLOPS	2011
GeForce GTX 680	3.090 TFLOPS	2012
GeForce GTX 780 ti	5.046 TFLOPS	2013
GeForce GTX 980 ti	5.632 TFLOPS	2015
GeForce GTX 1080 ti	10.609 TFLOPS	2017
GeForce RTX 2080 ti	11.750 TFLOPS	2018
GeForce RTX 3090	29.284 TFLOPS	2020

Tabla 2.1. Comparativa de potencia de cálculo en FP32 entre las tarjetas tope de gama del segmento de consumo de Nivida de los últimos 10 años

Como se puede apreciar en la tabla, la capacidad de las tarjetas gráficas se ha multiplicado exponencialmente en la última década, pudiendo hacer la más moderna, la RTX 3090 21 veces más operaciones punto flotante que la primera la GTX 480. Estos datos realmente no sirven para hacer una comparación directa entre las gráficas puesto que la cantidad de cálculos de coma flotante no es lo único importante para renderizar una imagen (por ejemplo, una RTX 2080ti rinde en videojuegos parecido a una RTX 3070 que tiene un 70% más de TFLOPS), pero si nos sirve para darnos cuenta de cómo ha ido aumentado la potencia de cálculo de las tarjetas gráficas.

2.3 Sistemas procedurales e inteligencia artificial para generar escenarios

Los algoritmos procedurales llevan acompañándonos en videojuegos, la friolera ya de 40 años, fueron utilizados en videojuegos incluso antes de que estos hicieran uso de gráficos por ordenador [8]. Un ejemplo de esto es el videojuego “Beneath Apple Manor” que utilizaba la generación por procedimientos para la construcción de mazmorras para los sistemas ASCII (Figura 2.7). Esta técnica fue ampliamente utilizada durante el desarrollo de los primeros videojuegos gráficos puesto que los sistemas en los que estos se ejecutaban, carecían de suficiente memoria como para tener todos los datos guardados y estos se debían ir generando según hicieran falta.

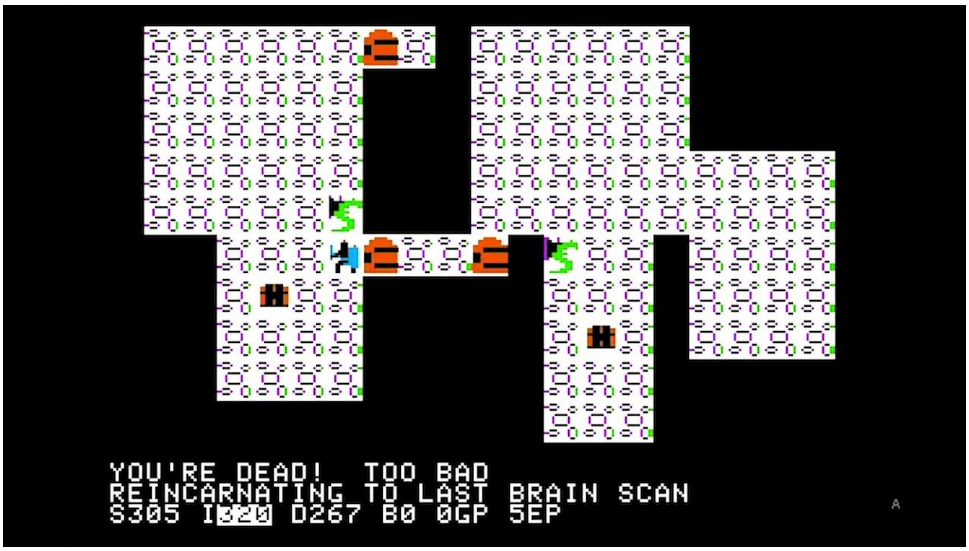


Figura 2.7 Escenario procedural del juego Beneath Apple Manor (1978)

Otro ejemplo también bastante impresionante para su época fue el de videojuego “Elite” (1984) que fue capaz de almacenar un total de 2048 mundos persistentes distintos (agrupados en 8 galaxias de 256 planetas cada uno) en 8 bits de memoria haciendo uso de lo que hoy en día conocemos como semilla o seed, un sistema de valores fijos que determinan el resultado del mundo de forma pseudoaleatoria, usando la misma semilla siempre se generará el mismo mundo.

Figura 2.8 Una pantalla del videojuego Elite

Pero no solamente son los videojuegos antiguos los que se benefician de este tipo de recursos, juegos más modernos como por ejemplo “The Division” (2016) o “No Man’s Sky” (2016) también hacen uso de este tipo de sistemas.

“The Division” un juego lanzado por Ubisoft Massive hace uso de algoritmos procedurales para generar los interiores de los edificios en lo que a colocación y distribución de mobiliario se refiere, mientas que dichos edificios están diseñados por diseñadores por fuera, sus estancias interiores se auto-rellenan de forma automática.

La continuación de este videojuego, “The división 2” (2019) va incluso más allá, presentando en una sus expansiones un nuevo modo de juego que se desarrolla en un rascacielos de 100 plantas generado de manera procedural [9].

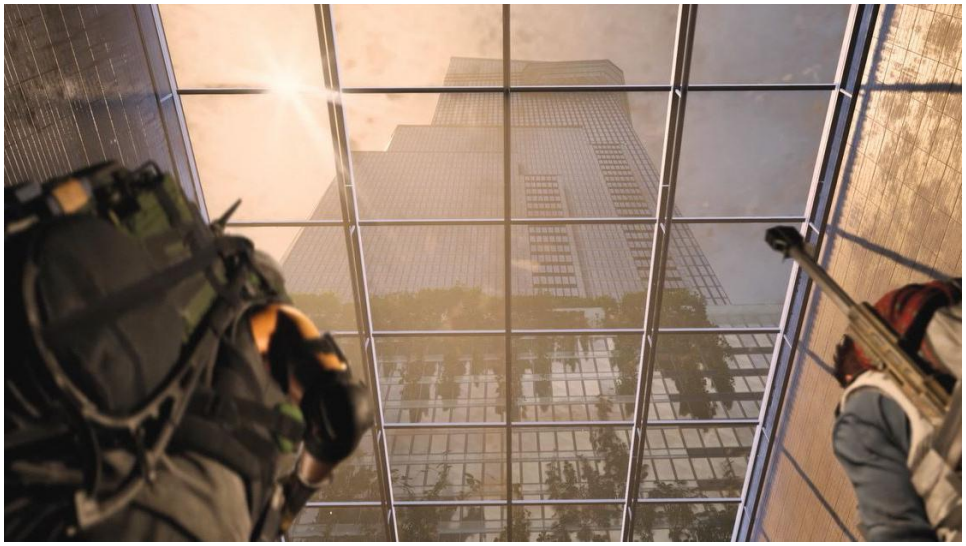


Figura 2.9. “The Division 2” (2019) Edificio de 100 plantas cuyo interior se genera de forma procedural

Otro ejemplo aún más asombroso de esta tecnología es el de “No Man’s Sky” (2016) un videojuego presentado por la empresa “Hello Games” en el año 2014 (Figura 2.10) como un juego de exploración espacial que contaba inicialmente con 18 Trillones Europeos (o 18 quintillones en el sistema de conteo americano) de planetas distintos generados automáticamente de manera procedural, incluyendo tanto su fauna como su flora.

Figura 2.10. Escenario procedural enseñado en 2014 en el juego 'No Mans's Sky'

Finalmente, el resultado del proyecto no acabó cumpliendo con las expectativas prometidas y acabo siendo un fracaso comercial. La figura 2.11 muestra el estado del juego tras su lanzamiento, aunque también cabe mencionar que la desarrolladora siguió trabajando en el juego aún después del lanzamiento (de hecho, a la fecha en la que estoy escribiendo estas líneas, se ha anunciado una nueva expansión para el juego, que aumenta su contenido) puliendo y mejorando las carencias de su lanzamiento inicial.

Figura 2.11. Estado de los escenarios procedural en el lanzamiento del juego "No Man's Sky" en 2016

El último juego a fecha de la redacción de este proyecto en implementar la tecnología de generación de escenarios procedurales es el juego "Returnal" de la desarrolladora de videojuegos "Housemarque" adquirida recientemente por la empresa "Sony PlayStation". Returnal hace uso de un avanzado sistema de generación procedural para generar los escenarios del juego, creando un nivel de realismo difícilmente visto por otros juegos que hacen uso de este mismo tipo de algoritmos. (Véase figura 2.12) Además, Returnal no hace uso de una semilla de generación del mundo (al menos no en sus primeras versiones), por lo que cada vez que el

usuario entra al juego el escenario, los enemigos, los objetos y las armas cambian de lugar, evitando así una experiencia monótona y un alto nivel de rejugabilidad, puesto que, al menos en la teoría, no existirán dos partidas del mismo juego iguales.

Figura 2.12. Escenario generado de manera procedural en el videojuego "Returnal"

2.4 Formatos Estándar para paso de mensajes entre programas

Otro de los puntos clave de este proyecto es el factor de modularidad del sistema y la independencia de cada uno de los módulos. Para lograr esto es necesario crear un sistema que permita el paso de mensajes (información) entre los módulos. Esto podría ser diseñado a medida para el proyecto, pero simplicidad y sobre todo por facilitar la compatibilidad con softwares externos, se ha buscado hacer uso de algún estándar para el paso de mensajes. En la actualidad existen dos estándares que se están por encima de los demás para estas tareas que son JSON y XML. A continuación, veremos en detalle cada uno de estos estándares y en que se diferencian.

2.4.1 JSON

JSON es el acrónimo de JavaScript Object Notation, un formato de texto sencillo de entender por personas diseñado para el intercambio de objetos. [10] Está basado en un subconjunto de la notación de objetos de JavaScript aunque dada su amplia adopción como alternativa a XML se le considera desde 2019 como un formato independiente del lenguaje. En la teoría, JSON cuenta con la ventaja sobre XML de que es más fácil escribir un analizado semántico para el. De hecho, JavaScript, el leguaje del que parte un JSON se puede analizar fácilmente con la instrucción "eval". El hecho de que sea tan fácil de integrar en JavaScript (uno de los leguajes de programación web más utilizado en el mundo), es uno de los factores claves de su popularidad. En la actualidad, debido a los problemas de seguridad que puede causar la instrucción "eval" y por el hecho de que ha habido un aumento significativo del uso de procesamiento nativo de XML, las ventajas que tenía JSON sobre este último lenguaje ya no quedan tan claras. Un ejemplo de la sintaxis de JSON puede verse en el siguiente listado:

```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customers": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

Listado 2.1 Sintaxis de JSON

2.4.2 XML

XML viene del inglés “eXtensible Markup Language” que significa lenguaje de marcas extensibles en castellano, al igual que en el caso anterior es un metalenguaje que permite el intercambio de información a través de las denominadas marcas o etiquetas (similar a como lo hace HTML). Este estándar fue desarrollado por el World Wide Web Consortium (W3C) como un sistema para almacenar datos de forma legible. A diferencia de otros leguajes similares, XML da soporte también a las bases de datos, siendo útil para conectar varias aplicaciones entre sí o para integrar datos. Un ejemplo de sintaxis de XML puede encontrarse en el capítulo 5, concretamente en el listado 5.1

3 ETAPAS DE DESARROLLO

3.1 Previsión

Para el desarrollo de este proyecto se ha seguido una metodología hibrida, se ha hecho un análisis preliminar de los requisitos del sistema extrayendo la información de las distintas partes y fases del proyecto (véase capítulo 4), una vez finalizado el análisis, se ha dividido el proyecto en distintas fases, cada una con sus propias tareas.

Debido a que los recursos y el tiempo para el desarrollo del proyecto eran limitados, se ha tratado cada una de estas fases como un subproyecto independiente de manera que aunque no fuese posible completar toda la especificación del proyecto, al menos se dispusiera de una versión utilizable, de esta manera la metodología utilizada para las distintas fases y dentro de cada una las distintas tareas es una metodología fuertemente inspirada en metodologías agiles como scrum, pero que no puede considerarse una metodología ágil en si misma puesto que carece de ciertas características de estas, como por ejemplo que no existe equipo de trabajo, ni cliente como tal (en la fase de desarrollo), puesto que todos estos roles los asume el propio estudiante. A continuación, se muestra un pequeño esquema que resume la metodología seguida para el desarrollo de este proyecto:

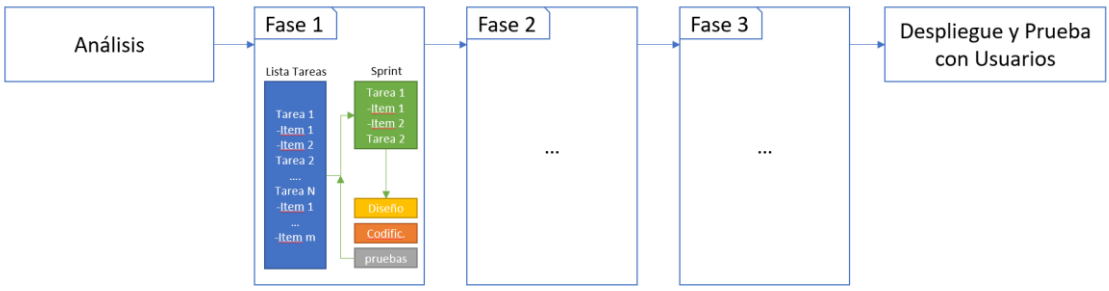


Figura 3.1 Metodología de desarrollo del proyecto

Una vez aclarado este punto, continuaremos el capítulo mostrando las fases del desarrollo de este proyecto junto con su estimación temporal (Figura 3.2. Cronograma del proyecto).

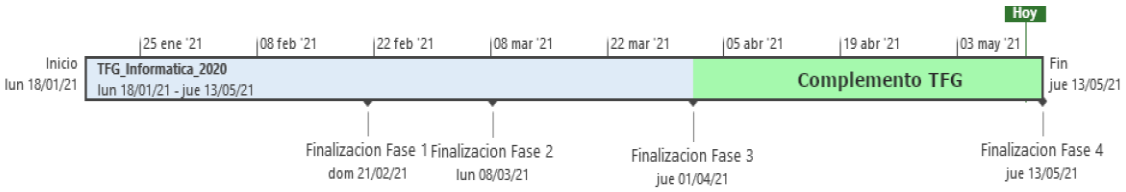


Figura 3.2: Cronograma del proyecto

A continuación, se especificarán las 3 Fases antes mencionadas y que se corresponden con el desarrollo técnico de este proyecto explicando cada una de las tareas que las componen, así como una breve descripción de estas.

Fase 1 – Diseño individual de cada parte (181 Horas)

Esta primera fase consiste en generar los distintos tipos de objetos y escenarios (simples) que usará la herramienta final, así como crear una versión funcional de todos los módulos (de manera independiente sin conectarse entre ellos).

- **Generación Procedural del terreno (28 Horas)**
  Esta tarea consiste en diseñar el algoritmo que se encargará de generar el terreno (suelo del escenario), detallando su superficie y topología.

- **Lector de archivos XML de la base de conocimiento. (24 Horas)**
  Este módulo es el encargado de leer la información de la base de conocimiento y cargarla en memoria, asi como de generar las estructuras de datos pertinentes para que pueda ser usado por el sistema.

- **Desarrollo de un Dataset de pruebas. (15 Horas)**
  Consistente en un archivo XML diseñado a mano con varios objetos que se puedan generar sobre el terreno y sus características para poder probar el sistema en su versión temprana y poder probar el buen funcionamiento de este.

- **Generación de objetos de superficie: (34 Horas)**
  Este módulo se hace cargo de la gestión de los objetos que se pueden colocar sobre la superficie del terreno como casas, árboles o piedras. Entre otras cosas se encarga de determinar donde se coloca un terreno y de asegurarse que el objeto se coloca de acuerdo con las limitaciones propias del objeto (por ejemplo, que no se coloque un árbol boca abajo).

- **Desarrollo del módulo de reconocimiento de voz. (30 Horas)**
  El objetivo de este módulo será el de captar la voz del usuario y transcribirla a texto para posteriormente poder hacer un análisis de lo que el usuario desea de cara a poder generarlo en el escenario.

- **Generación de objetos basados en líquidos. (30 Horas)**

Este módulo tratará con los líquidos, como ríos o lagos, estos objetos son especiales pues son afectados por la gravedad y pueden formar surcos en la tierra, por tanto, deben tratarse de forma distinta al resto de objetos

- **Generación de vías y caminos, mismo caso que el anterior (20 Horas)**
  Las vías y caminos son otro tipo de objetos que requieren un tratamiento especial pues dependen de la topología del terreno y por tanto requieren de un módulo especial.

Fase 2 – Unión y mejora de cada parte (124 Horas)

Esta segunda fase se centra en la conexión entre los distintos módulos del sistema a nivel de intercambio de datos, así como de mejorar lo ya desarrollado en la primera fase, añadiendo estructuras más complejas y mayor variedad de datos de entrada.

- **Exportador de Datos de la extracción de usuario en XML (20 Horas).**
  Esta tarea consiste en desarrollar el módulo que se encargará de exportar toda la información obtenida por el subsistema de captación de datos en forma de XML para poder usarlo en el de generación de escenarios.
- **Lector de archivos XML de los datos de usuario (24 Horas).**
  Este módulo se encarga de la tarea opuesta al anterior, es decir, importar los datos en XML generados por el anterior módulo al módulo de generación y generar las estructuras de datos necesarias para poder usarlas.

- **Desarrollo de un Dataset completo de la base de conocimiento, (35 Horas)**
  Este módulo consiste en generar una versión final del dataset, mucho más extendida que la anterior versión de pruebas, con muchos más objetos y más variados, además de la posibilidad de introducir variaciones de objetos como parte del dataset.

- **Generación de estructuras colectivas como bosques o ciudades. (45 Horas).**
  Finalmente, el cometido de esta tarea será la de procesar estructuras colectivas (unión de varios objetos más simples) para generarlos en el escenario en forma de bosques, aldeas, etc.

Fase 3 - Finalización del TFG (20 Horas)

- **Desarrollo de la herramienta para desarrolladores (20 Horas).**
  Este es el hito final del proyecto, consistente en la unificación de todo lo desarrollado anteriormente para generar la herramienta que posteriormente podrán usar los desarrolladores para generar sus propios escenarios.

Una vez finalizadas estas tres fases, se procederá con la etapa despliegue de la aplicación, la prueba con usuarios y la retroalimentación ofrecida por estos, toda esta información se encuentra especificada en el capítulo 6, concretamente la sección 6.3 de este documento.

3.2 Resultado final

La versión final del cronograma del proyecto ha sufrido varias variaciones respecto a la versión inicial prevista, principalmente por un retraso al intentar aplicar la API de Facebook para cargar los datos directamente de las redes sociales.

Desde el año 2019 Facebook ha cambiado la forma de conectarse a sus redes sociales, haciendo que las conexiones a una página o un perfil deban pasar primero por un proceso de "login" al cargar la página, de esta manera se vuelve muy difícil por no decir casi imposible la utilización de "scrappers" para obtener la información, obligando al desarrollador a utilizar la propia API de Facebook para obtenerla.

Pero, en contra partida a lo anterior, por motivos de seguridad la API de Facebook exige un largo, riguroso y tedioso proceso de verificación para permitirle el acceso al desarrollador a datos como las imágenes o la ubicación del usuario, ya que estos datos se encuentran protegidos al menos para las aplicaciones externas.

Finalmente, tras un arduo mes de trabajo intentando obtener acceso a esta información, se optó por desestimar esta funcionalidad en pro de poder entregar este documento en tiempo y forma. Aun así, se adaptó parcialmente la funcionalidad, obteniendo la ubicación del usuario a través de un script PHP y se optó por que el usuario pudiese cargar las imágenes directamente desde una carpeta de su equipo local. Todos estos cambios ocasionaron un retraso aproximadamente de 30 días como se puede ver en la siguiente figura:

Figura 3.3. Tiempo real de desarrollo del proyecto

4 ANÁLISIS

4.1 Descripción Detallada de la Solución

Se desarrollará una herramienta para Unity3D que pueda ser usada por los desarrolladores para generar escenarios de forma procedural y personalizada a los deseos y datos de cada usuario. Esta herramienta recibirá como input dos ficheros XML, uno consistente en la base de datos del conocimiento, que contará con información de diversos objetos, climas, etc. Así como de sus respectivos datos específicos, por ejemplo, en el caso de los objetos se reconocerán además de su correspondiente modelo 3D, sus datos de tamaño máximo y mínimo, o el clima al que pertenecen. El segundo fichero que se utilizará será otro archivo XML que contendrá los deseos de lo que el usuario quiere generar, así como información relevante acerca de este, esta parte se explicará a continuación.

Para la parte de obtención de datos del usuario, se creará otra herramienta esta vez en Visual Studio que recopilará haciendo uso de los servicios cognitivos de Microsoft Azure (reconocimiento de voz), los deseos del usuario, asi como dará la posibilidad al desarrollador de crear un perfil personalizado haciendo uso de fotografías personalizadas que permitirán al sistema a través de un sistema de reconocimiento de patrones de imágenes (Alturos Yolo) los objetos de dichas imágenes y por tanto los entornos por los que esa persona se mueve.

A lo anterior se le suma un script PHP en un servidor privado que reconocerá la ubicación del usuario, para obtener datos más precisos de la cultura y el clima de dicho usuario.

Finalmente, para que el reconocimiento de voz sea satisfactorio se hará uso de un diccionario de datos (XML) que interpretará/traducirá las palabras del usuario a un “lenguaje” (XML) entendible por el sistema.

4.2 Casos de uso del sistema

En esta sección se detallarán los casos de uso del sistema, incluyendo el diagrama de casos de uso, junto con la especificación de cada uno.

4.2.1 Diagrama de casos de uso

A continuación, en la figura 4.1 se puede observar el diagrama de casos de uso del sistema, en el que se resumen visualmente los casos de usos que están ligados a la interacción del usuario con el sistema.

Figura 4.1. Diagrama de casos de uso del sistema.

4.2.2 Actores del sistema

El sistema cuenta con dos actores, el desarrollador y el usuario, uno para cada uno de los módulos principales del sistema, recopilación de datos y generación, aunque en la práctica pueden ser la misma persona:

A01	**Desarrollador**
Requisitos Asociados	RF01, RF02, RF03, RF04, RF08, RF09, RF10, RF11, RF12, RF13, RF14, RF15
Descripción	Formado por todas las personas que harán uso de la herramienta de generación en Unity3D
Comentarios	

A02	**Usuario**
Requisitos Asociados	RF05, RF06, RF07, RF16
Descripción	Formado por aquellas personas que hagan uso del módulo de recopilación de datos
Comentarios	

4.2.3 Casos de uso del sistema

A continuación, se detallarán cada uno de los casos de uso del anterior diagrama (figura 4.1)

RF01	Importar Base Datos Conocimiento
Requisitos Asociados	
Descripción	El Desarrollador (A01) podrá importar un archivo de datos XML con los datos necesarios para generar el escenario (Base de datos del conocimiento)
Precondición	Debe existir el archivo de datos de la base del conocimiento
Postcondición	El archivo ha de tener un formato de datos adecuado
Comentarios	

RF02	Interactuar con el escenario
Requisitos Asociados	RF03
Descripción	El sistema generara un escenario que podrá interactuar con el resto de assets que pueda crear un Desarrollador (A01)
Precondición	
Postcondición	
Comentarios	

RF03	Gestión de físicas
Requisitos Asociados	RF02
Descripción	El sistema dotará de físicas a los elementos que sean necesarios para que el Desarrollador (A01) pueda interactuar con el sistema
Precondición	Deben existir elementos en el escenario que hayan de ser dotados de físicas
Postcondición	
Comentarios	

RF04	Importar Datos Personalizados
Requisitos Asociados	
Descripción	Un Desarrollador (A01) podrá importar un archivo XML con los datos personalizados recabados en el módulo de recolección de datos
Precondición	Debe existir un archivo de datos del usuario
Postcondición	El archivo ha de tener un formato de datos adecuado
Comentarios	

RF05	Reconocimiento Deseos Usuario
Requisitos Asociados	RF06
Descripción	Un usuario (A02) podrá especificar a través de la voz lo que de sea generar y sistema lo reconocerá y almacenará
Precondición	
Postcondición	
Comentarios	

RF06	Recopilar Información Usuario
Requisitos Asociados	RF05, RF07
Descripción	El sistema podrá recopilar los datos generados por un usuario (A02)
Precondición	
Postcondición	
Comentarios	

RF07	Reconocimiento de Imágenes
Requisitos Asociados	RF06
Descripción	Un usuario (A02) podrá adjuntar un conjunto de imágenes para que el sistema reconozca los objetos que hay en ellas.
Precondición	
Postcondición	
Comentarios	

RF08	Generar Escenario
Requisitos Asociados	RF09, RF11, RF12, RF13
Descripción	El sistema generará un escenario virtual de acuerdo con los archivos de datos introducidos por el desarrollador (A02)
Precondición	Deben estar cargados, la base de datos y el archivo de los datos de usuario
Postcondición	
Comentarios	

RF09	Generar Caminos
Requisitos Asociados	RF08, RF10
Descripción	El sistema calculará y generará caminos de acuerdo con los elementos que existen en la escena.
Precondición	Existen elementos en la escena que requieran de la generación de uno o varios caminos
Postcondición	
Comentarios	

RF10	Suavizar Caminos
Requisitos Asociados	RF09
Descripción	El sistema suavizará el trazado del camino generado anteriormente haciendo uso de curvas de Bezier
Precondición	El nivel de precisión del camino debe ser mayor que uno o bien este ha sido generado con un algoritmo de generación de varios niveles de granularidad
Postcondición	
Comentarios	

RF11	Generar Objetos

Requisitos Asociados	RF08, RF12
Descripción	El sistema generará objetos en el escenario de acuerdo con lo especificado en los archivos de datos
Precondición	El archivo de datos del usuario debe contener algún objeto
Postcondición	
Comentarios	

RF12	**Generar Estructura**
Requisitos Asociados	RF08, RF11
Descripción	El sistema generará estructuras (objetos colectivos) de acuerdo con lo especificado en los archivos de datos
Precondición	Deben existir estructuras en el archivo de datos del usuario
Postcondición	
Comentarios	

RF13	**Generar Terreno**
Requisitos Asociados	RF08, RF14
Descripción	El sistema generará la superficie del terreno de acuerdo con los especificado en los archivos de datos
Precondición	Deben estar cargados, la base de datos y el archivo de los datos de usuario
Postcondición	
Comentarios	

RF14	**Generar Ríos**
Requisitos Asociados	RF13, RF15
Descripción	El sistema generará los ríos de acuerdo con los especificado en los archivos de datos y el contexto del escenario.
Precondición	Deben existir ríos en el archivo de datos del usuario o bien deben ser requeridos por la topología del terreno del archivo de datos de usuario
Postcondición	
Comentarios	

RF15	**Modificar Terreno**
Requisitos Asociados	RF14
Descripción	El sistema modificará la superficie de los terrenos en función de los ríos que se hayan generado
Precondición	Deben existir ríos en el escenario
Postcondición	
Comentarios	

RF16	**Reconocimiento Datos de ubicación Usuario**
Requisitos Asociados	RF06

Descripción	El sistema reconocerá a través de la conexión con un script PHP y la IP publica de un usuario (A02) su ubicación.
Precondición	Debe haber conexión a internet
Postcondición	
Comentarios	

4.3 Requisitos funcionales

Los requisitos funcionales del proyecto coinciden con los expuestos anteriormente en el diagrama de casos de uso, y por tanto son los siguientes:

- RF01 Importar Base de Datos Conocimiento
- RF02 Interactuar con el escenario
- RF03 Gestión de físicas
- RF04 Importar Datos Personalizados
- RF05 Reconocimiento Deseos Usuario
- RF06 Recopilar Información Usuario
- RF07 Reconocimiento de Imágenes
- RF08 Generar Escenario
- RF09 Generar Caminos
- RF10 Suavizar Caminos
- RF11 Generar Objetos
- RF12 Generar Estructura
- RF13 Generar Terreno
- RF14 Generar Ríos
- RF15 Modificar Terreno
- RF16 Reconocimiento Datos de Ubicación Usuario

4.4 Requisitos no Funcionales

Por un lado, los requisitos no funcionales desde el punto de vista temporal no son relevantes ya que no estamos trabajando con un sistema a tiempo real, más allá de los requisitos temporales de la interfaz de usuario.

- El tiempo de respuesta de la interfaz de usuario debe ser menos a 0.1 segundos.

Respecto a la disponibilidad de los servicios la mayoría de ellos se ejecutan en local por lo que están siempre disponibles a excepción de los servicios de Azure y el del servidor Apache/PHP:

- El servicio de reconocimiento de voz de Azure debe estar disponible el 99.95% del tiempo
- El servidor Apache con el script PHP ha de estar disponible el 99.5% del tiempo

Finalmente, el resto de los requisitos no Funcionales son:

RNF01	Lenguaje soportado por el diccionario de reconocimiento de objetos
Descripción	El idioma predeterminado del archivo será ingles
Comentarios	Se incluirá también soporte para el idioma español como principal idioma usado por el usuario.

RNF02	Seguimiento de errores
Descripción	La aplicación tendrá sus propios códigos de errores
Comentarios	Estos serán mostrados a través de la consola de Visual Studio / Unity según corresponda

RNF04	Manual de Usuario
Descripción	El sistema deberá de contar con manuales de sistema bien estructurados.
Comentarios	Serán 2 los manuales, uno para el sistema de recolección de datos y otro para el de generación.

4.5 Análisis económico

4.5.1 Azure Speech Recognition

El precio de este servicio se calcula en función del número de horas de audio que se tengan que analizar, además existe una limitación en cuanto al número de conexiones simultaneas que el servicio puede resolver. Microsoft ofrece 3 planes, Gratuito, Estándar y Personalizado. Además de los tres mencionados también incluye un servicio de transcripción multicanal que aún se encuentra en versión preliminar.

A continuación, en la tabla 4.1 se puede ver la lista completa de precios de este servicio por cada hora de audio analizada (o las limitaciones de uso en el caso de la versión gratuita).

Precios Reconocimiento de Voz Azure		
	Gratis	5 horas de audio gratis al mes
	Estándar	0,844€
	Personalizado	1,181€ + 0,0454€/modelo hora
	Audio Multicanal	1,178€

Tabla 4.1. Precios por hora de los servicios de reconocimiento de voz de Microsoft Azure a fecha de 26-06-2021. Fuente: Microsoft Azure

Dada la envergadura actual del proyecto en fase de pruebas se ha optado por usar el servicio gratuito (puesto que se está lejos de alcanzar las 5 horas de uso mensuales), pero de cada a fases posteriores de despliegue en caso de que el volumen de datos aumentase significativamente se haría el traspaso al servicio Estándar.

4.5.2 Unity3D

Aunque unity3D es una herramienta que se puede usar de forma gratuita, a la hora de publicar el software si puede haber ciertos requerimientos de pago en caso de que se cumplan ciertas condiciones. Principalmente estas condiciones consisten en comercializar el proyecto y obtener unos ingresos superiores a ciertas cantidades. La siguiente tabla refleja los precios anuales y

mensuales de las licencias de Unity3D además de las condiciones para la obligatoriedad de contratarlas. Cabe destacar que estos precios son para ca uno de los puestos de trabajo.

Plan	Personal	Plus	Pro	Enterprise
Requisitos	Ingresar menos de 100K$ en los últimos 12 meses	Ingresar menos de 200K$ en los últimos 12 meses	Ingresar más de 200K$ en los últimos 12 meses	Ingresar más de 200K$ en los últimos 12 meses. Mínimo contratar 10 asientos
Coste por puesto de trabajo	0$	399$/año O 40$/mes	1800$/año O 150$/mes	2000$/año No ofrece tarifa mensual

Tabla 4.2. Precios por puesto de trabajo de unity3D a fecha de 26-06-2021. Fuente: Unity 3D

4.5.3 Presupuesto de realización del proyecto

A continuación, se detallará el coste económico que ha supuesto el desarrollo del presente proyecto, para ello se hará uso de las estimaciones temporales realizada en el apartado 3 "Etapas del desarrollo". A pesar de esto, no solo se considerarán los costes de tiempo de desarrollo si no también todos aquellos costes de adquisiciones y costes indirectos que se hayan tenido que hacer de cada a este proyecto.

Costes Directos

Se consideran costes directos todos aquellos que están directamente relacionados con la realización de proyecto como pueden ser el capital humano o las adquisiciones de equipos. A continuación de desgranarán estos costes:

- **Capital Humano.** Para el cálculo de este coste se ha tenido en cuenta el nuero de horas estimadas de trabajo, así como se ha usado como referencia el sueldo actual del alumno en su actual empleo que es de 14€/Hora neto como una aproximación del valor de su tiempo. A este valor habría que sumarle las retenciones por IRPF que serían en este caso del 24%. Por lo que el precio hora bruto sería de:

$$14€/h \;/\; (1-0.24)(IRPF) \;= 18.42€/Hora$$

  Finalmente, el coste en capital humano sería de:

$$300Horas \times 18{,}42€/Hora \;= 5526€$$

- **Hardware**. Para el desarrollo de este proyecto se ha hecho uso principalmente de tres dispositivos hardware, el ordenador del alumno, concretamente un ordenador personalizado, que en el momento de su adquisición supuso un coste total de 1200€, un servidor NAS de la marca Synology para ejecutar el script PHP que obtiene la localización del usuario a través de la dirección IP. Este servidor no tiene costes directos pues no se le está cobrando al alumno por su uso, pero si tiene unos costes indirectos asociados que veremos más adelante y una Raspberry pi 4B que contiene el software de Redmine utilizado para el seguimiento del proyecto y que ha tenido un coste de 40€

- **Software**. Aunque actualmente todo el software es gratuito, en caso de que el volumen de trabajo del servicio aumentase podrían aparecer costes adicionales como los costes asociados al reconocimiento de voz de Azure o los costes de suscripción de Unity.

Costes Indirectos

En este apartado se tendrán en cuesta todos aquellos gastos relacionados con el proyecto pero que no tienen una influencia directa sobre él.

- **Coste eléctrico y de amortización del Servidor Synology.** En este caso se ha calculado que el uso del servidor es de aproximadamente 40 horas al mes para este proyecto. Se ha tomado como precio medio de la luz 0.14€/KWH. La esperanza de vida del servidor y su disco duro es de 10 años, ya no se puede saber con certeza si fallará antes o después, tomaremos este valor como referencia. Finalmente, el coste del servidor (DS218j) y del disco duro (Samsung Evo 860) asciende a 300€, lo que nos da un coste por hora aproximado de 0,0034€/hora de uso lo que nos da para este proyecto un coste de 13.6 céntimos de euro al mes en concepto de amortización. A esto hay que sumarle el consumo eléctrico que es de 20w/h, lo que mensualmente nos costaría aproximadamente 11.2 céntimos. Por lo que mensualmente el coste total ascendería a:

$$0.136€ + 0.112€ = 0.248€/mes \simeq 0.25€/mes$$
$$0.25€/Mes \times 6\ Meses\ de\ Uso = 1.5€$$

- **Coste eléctrico del ordenador**. Otro coste no directo sería el coste eléctrico del ordenador (y las pantallas) del alumno para el cual para este caso se tomará como referencia las 300 Horas calculadas en al apartado 3 “Etapas del desarrollo “, así como un consumo medio de 300w (calculados con un medidor de consumo conectado a la toma eléctrica del ordenador y las pantallas). En este caso usaremos el mismo coste eléctrico de 0.14€/KWH. El coste total de este gasto quedará reflejado de la siguiente manera:

  $$300Horas \times 0.300KwH \times 0.14€/KwH = 12.6€$$

  Dentro del consumo eléctrico se ha contabilizado también el propio de la Raspberry Pi (10W)

- **Costes de desplazamiento.** A raíz de la pandemia de la Covid-19. No se ha necesitado hacer ningún desplazamiento para la realización de este TFG. Por lo que no existen costes derivados de este concepto.

Otros Gastos Indirectos y margen de maniobras para imprevistos.

Finalmente se contempla un gasto de un 20% del total del proyecto destinado al margen de maniobra para emergencias o gastos imprevistos y los gastos indirectos que no se encuentren contemplados en este documento.

Costes totales

A continuación, se muestran desglosados los costes totales del proyecto:

Tipo de Coste	Importe
Capital Humano	5526€
Hardware	1240€
Software	0€
Costes Indirectos	14.1€
Subtotal	6780.1€
Margen de maniobras y otros gastos (20%)	1356.02€
Coste Total	**8132.12€**

Tabla 4.3. Presupuesto del proyecto

5 TECNOLOGÍAS Y HERRAMIENTAS

5.1 XML

Como ya se comentó en el capítulo 2, XML es un lenguaje de comunicación de información a través de etiquetas [11], es un lenguaje fácil de comprender tanto por humanos como por las maquinas, esta es una de las principales razones de su elección ya que ciertos archivos como la base de datos del conocimiento debía ser diseñada a mano, se necesitaba de un lenguaje que también fuera entendible por un ser humano. Esto se podría resolver con un simple archivo de texto, sí, pero eso no tendría la ventaja de ser un estándar y por tanto obligaría los desarrolladores a aprenderse su sintaxis para poder utilizarlo, con XML esto último no es necesario. Este es un ejemplo de sintaxis extraído del archivo XML de la base de datos del conocimiento:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Root>
  <Objects>
    <Object name = "arbolMediterraneo" resource = "Broadleaf_Desktop_2">
      <maxHeight>60</maxHeight>
      <maxVangle>30</maxVangle>
      <maxSize>1.3</maxSize>
      <minSize>0.9</minSize>
    </Object>

    <Object name = "arbol_Alantico" resource = "Broadleaf_Desktop_3">
      <maxHeight>80</maxHeight>
      <maxVangle>30</maxVangle>
      <maxSize>0.6</maxSize>
      <minSize>0.3</minSize>
    </Object>

    <Object name = "arbol" resource = "CartoonTree">
      <maxHeight>60</maxHeight>
      <maxVangle>30</maxVangle>
      <!-- <lockYRotation>TRUE</lockYRotation> -->
      <maxSize>1.3</maxSize>
      <minSize>0.9</minSize>
      <Variant name = "arbolMediterraneo" wheather ="Mediterraneo"></Variant>
      <Variant name = "arbol_Alantico" wheather ="Atlantico"></Variant>
    </Object>
```

Listado 5.1. Estructura de XML

Como se puede apreciar la sintaxis de XML es limpia y sencilla, justo lo que se busca, además de esto tanto Visual Studio como Unity 3D tienen librerías que permiten trabajar con bastante facilidad con XML. Lo que lo hace idóneo para este proyecto.

5.2 Redmine

Redmine es una aplicación web flexible para la gestión de proyectos. Funciona bajo la licencia GNU Generic Public License v2 (GPL) lo cual es una gran ventaja a la hora de desarrollar módulos personalizados para esta herramienta. Además, una de las principales ventajas de usar esta herramienta es que al ser una herramienta web no hace falta tener instancias en cada equipo para utilizarlo y además puede trabajar con múltiples plataformas y múltiples bases de datos. Otro de los principales de Redmine es su interfaz gráfica sencilla y su curva de aprendizaje que es muy poco elevada, en la figura 5.1 se puede observar un ejemplo de dicha interfaz.

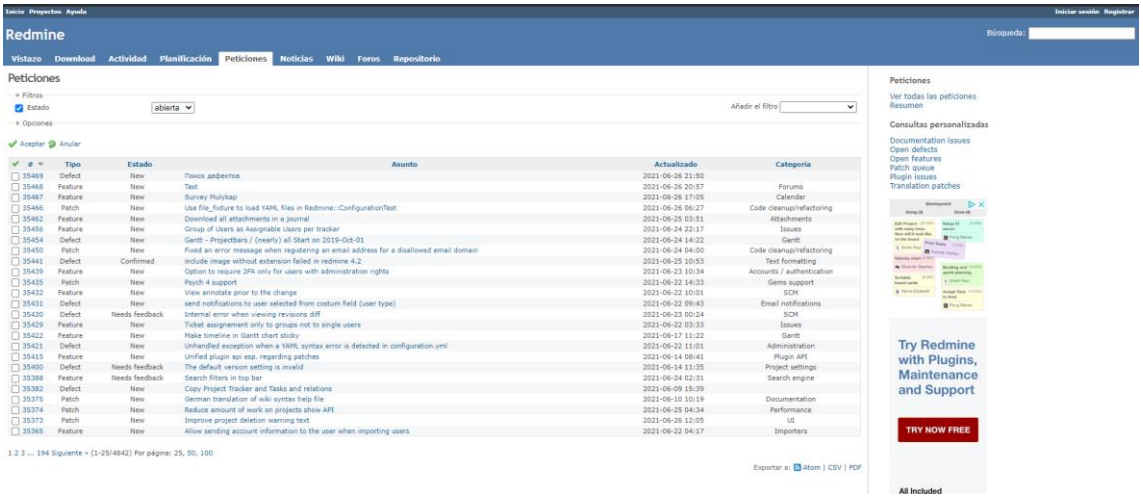


Figura 5.1. Interfaz de Redmine

La única pega que tiene Redmine es que está escrito usando el framework de “Ruby on Rails” y esto limita a la herramienta. Por ejemplo, en el caso de este proyecto, esta herramienta no se ha podido instalar en el servidor Synology ya que este no cuenta con soporte para Ruby nativo y finalmente se ha optado por obtener un dispositivo raspberryPi4 sobre el que se ha instalado la herramienta. Los problemas con Ruby son tales, que la recomendación que se suele hacer es la de ejecutarlos desde Docker y no hacer la instalación nativamente.

5.2.1 Docker

Como se comenta anteriormente, Redmine tiene bastantes problemas para instalarse de forma nativa con facilidad, eso empujó a que para su utilización se tuviera que hacer uso de un contenedor de Docker.

Un contenedor es una unidad software que empaqueta un código (junto con todas sus dependencias) que permite que la aplicación pueda ejecutarse sobre cualquier entorno informático, asegurando la compatibilidad, a pesar de que las maquinas sean distintas.

Estos contenedores se rellenan con “imágenes” que es como se le denomina a un paquete de software que tienen todos los recursos necesarios para ejecutar una aplicación por si solo.

Esta tecnología puede sonar parecido a como funcionaría una máquina virtual, y en cierta manera ambas tecnologías tienen bastante en común pero también bastante en discordia, la siguiente figura muestra las principales diferencias entre Docker y una máquina virtual:

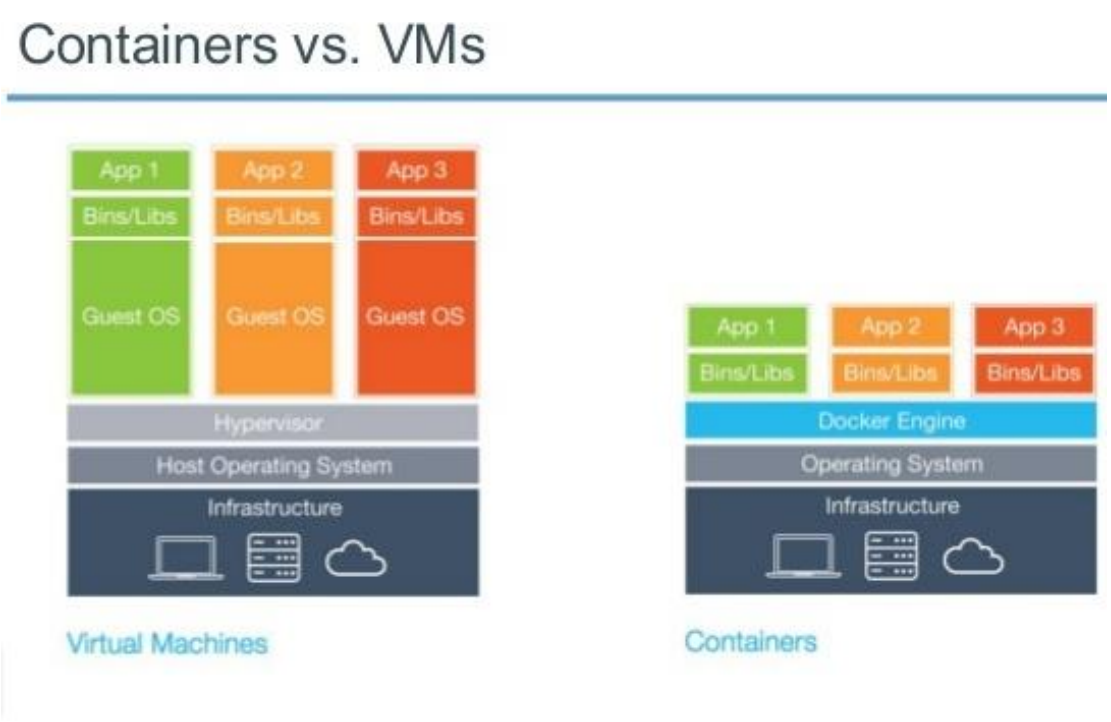


Figura 5.2. Diferencias entre Docker y Una Máquina Virtual

Como se puede ver en la imagen anterior hay varios puntos que se pueden remarcar: [12]

- Los contenedores Docker son procesos aislados y no requieres un hipervisor de hardware, lo que en la práctica se traduce en que son más ligeros y requieren menos recursos que las máquinas virtuales.
- Precisamente derivado del punto anterior, los contenedores de Docker son mucho más rápidos de iniciar y utilizar que las máquinas virtuales.
- Los contenedores de Docker se pueden compartir entre máquinas o personas sin preocuparse por la compatibilidad o por si el software funcionara de forma distinta en otro equipo.
- Al ejecutar una máquina virtual, se pueden usar instancias de Docker dentro de esa misma máquina virtual. Lo cual indica que las máquinas virtuales y Docker no son recursos antagónicos si no que pueden trabajar en conjunto.

Internamente Docker está compuesto por 3 componentes, un cliente, por medio del cual se ejecutan los distintos comandos, un servidor que es el que ejecuta las tareas y una API Rest que es la encargada de comunicar las dos partes anteriores. A continuación, se puede ver un ejemplo de esta arquitectura:

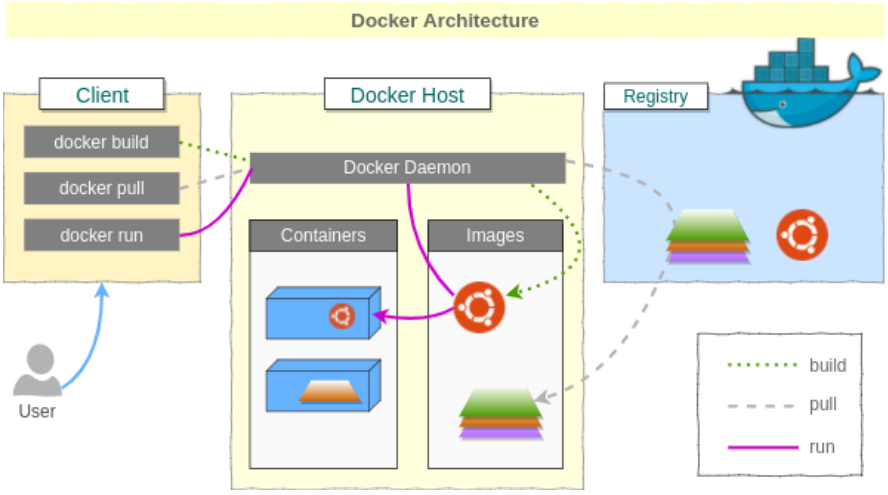


Figura 5.3 Arquitectura de Docker

5.3 Blender

Blender es una herramienta de modelado 3D multiplataforma orientada principalmente al diseño, renderizado, iluminación y animación de gráficos en 3D, además tiene opciones composición digital, edición de vídeo, escultura y pintado digital. Para este proyecto, Blender se ha utilizado para el diseño de gran parte de modelos 3D de objetos utilizados tanto para este trabajo fin de grado como para el homólogo complemento de este.

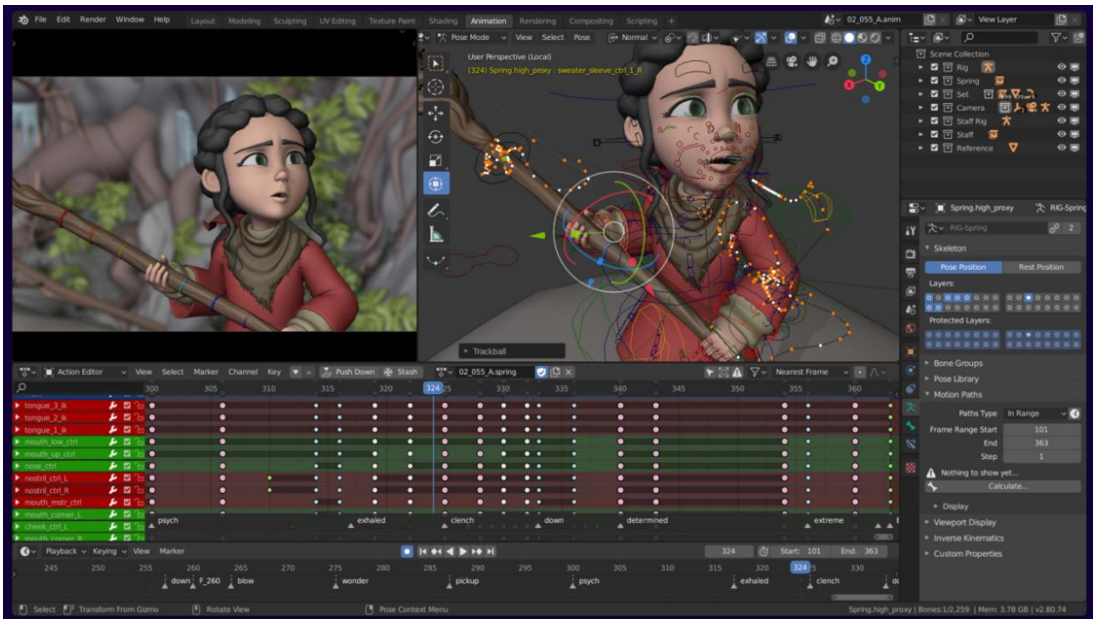


Figura 5.4. Interfaz de Blender 2.8 la penúltima versión del programa

Blender fue lanzado originalmente en 1994 de forma gratuita junto con un manual, pero sin el código libre, aunque más tarde paso a formar parte del modelo del software libre. Blender acepta una gran variedad de formatos de imagen y modelo 3d entre los que destacan:

- 3ds Max (.3ds)
- Autodesk FBX (.fbx)
- Collada (.dae)

- MDD (.mdd)
- Motion Capture (.bvh)
- Paths (.svg, .ps, .eps, .ai y .gimp)
- Raw Image File (.raw)
- Stanford PLY (.ply)
- STL (.stl)
- Wavefront OBJ (.obj)
- X3D Extensible 3d (.x3d)

Aceptar tanta variedad de formatos, dota a Blender de una gran versatilidad que lo hace útil para una gran cantidad de situaciones profesionales, desde el Diseño de películas, al modelado 3D de videojuegos, pasando por la animación de personajes, la impresión 3D o los efectos especiales. Un ejemplo de esto último puede verse en el corto desarrollador en Blender, "Tears of Steel" producido en 2013 haciendo uso de la versión 1.6.3 de Blender. En dicho corto puede verse la utilización de actores y entornos reales mezclados con elementos y escenarios virtuales, dando como resultado una composición visual francamente espectacular para la época.

Figura 5.5. Escena de la película "Tears of Steel" desarrollada en Blender 2.6.3 y que combina elementos reales y virtuales en la misma pantalla

Para alcanzar estos resultados Blender pone a la disposición de los usuarios varios motores de renderizado, los cuales se encargarán de convertir los polígonos y efectos diseñados en la herramienta en las imágenes que el usuario podrá observar como resultado final del proceso de renderización anteriormente explicado (capítulo 2):

5.3.1 Workbench (Anteriormente Blender Render)

El más básico y el primer en implementarse. Es el más rápido de los tres, pero también el más desfasado y el que da peores resultados. La siguiente figura muestra una imagen renderizada con este motor.

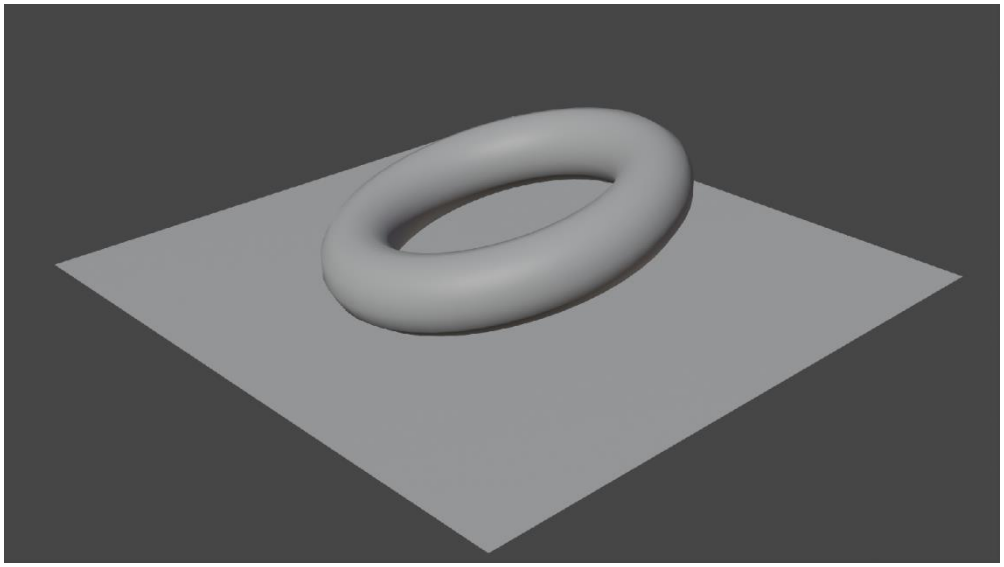


Figura 5.6. Renderizado de una simple composición en Blender con el motor Workbench

5.3.2 Cycles

Fue diseñado como el motor de alto rendimiento de Blender y es el que hasta la fecha ofrece los mejores resultados, tiene un sistema de shaders y materiales mucho más complejos que Workbench, pero es a su vez muchísimo más lento (en la tabla 5.1 puede verse una comparativa entre los tiempos de renderizado de cada motor). La siguiente figura muestra la misma composición que antes, pero renderizado con Cycles en vez de con Workbench.

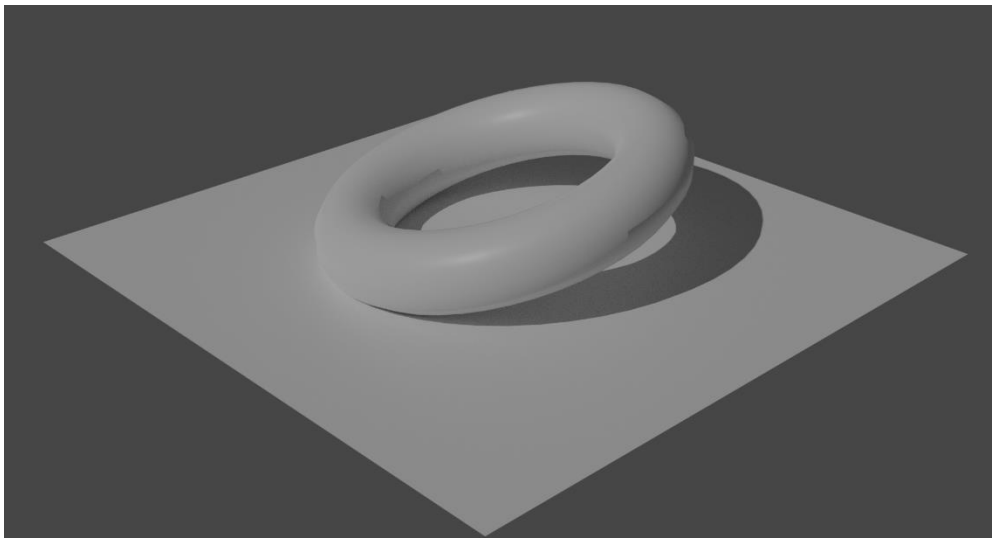


Figura 5.7. Renderizado de una simple composición en Blender con el motor Cycles

5.3.3 Eevee

Es el motor más moderno, fue incluido por primera vez en la versión 2.80 el 30 de julio de 2019. Su objetivo es el de alcanzar unas cotas de calidad muy similares a Cycles pero haciendo uso de una menor cantidad de recursos y por tanto de mucho menos tiempo de renderizado. A continuación, puede verse la misma composición que en las figuras 5.6 y 5.7 pero renderizada en esta ocasión con el motor Eevee:

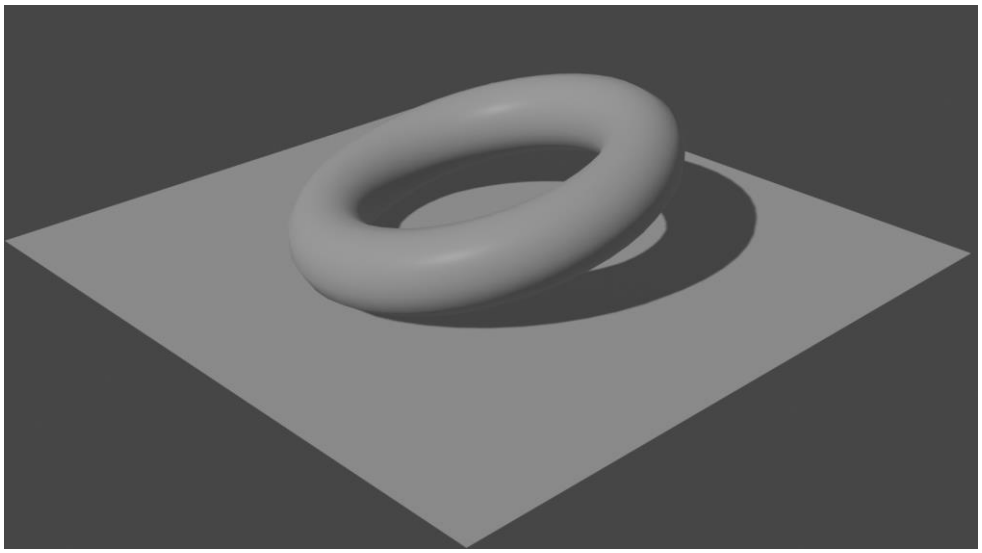


Figura 5.8 Renderizado de una simple composición en Blender con el motor Eevee

La siguiente tabla muestra una comparación de los tiempos de renderización de la composición para los siguientes motores, el tiempo mostrado es el tiempo medio extraído de renderizar cada imagen 5 veces. Cabe destacar que el tiempo de renderizado depende de la velocidad de cálculo del sistema y de la complejidad de la escena, por lo que no es extrapolable a otras escenas u sistemas informáticos.

WorkBench	Cycles	Eevee
0.11 seg	15.45 seg	0.38 seg

Tabla 5.1. Comparativa de tiempo de renderizado de los 3 motores de Blender en Blender 2.90.1 con un procesador Ryzen 7 3700X

5.4 Visual Paradigm

Visual Paradigm [13] es una herramienta de diseño de diagramas altamente utilizadas por los ingenieros de software a la hora de diseñar las funcionalidades de sus programas. Aunque, el diseño de diagramas es su función principal, también incluye herramientas complementarias como la generación automática de código y/o de documentación o funciones de ingeniería inversa entre otras. Además, otra ventaja de visual paradigma es que su interfaz es relativamente sencilla de utilizar lo que lo hace ideal para ir introduciéndose en el desde la etapa de estudiante.

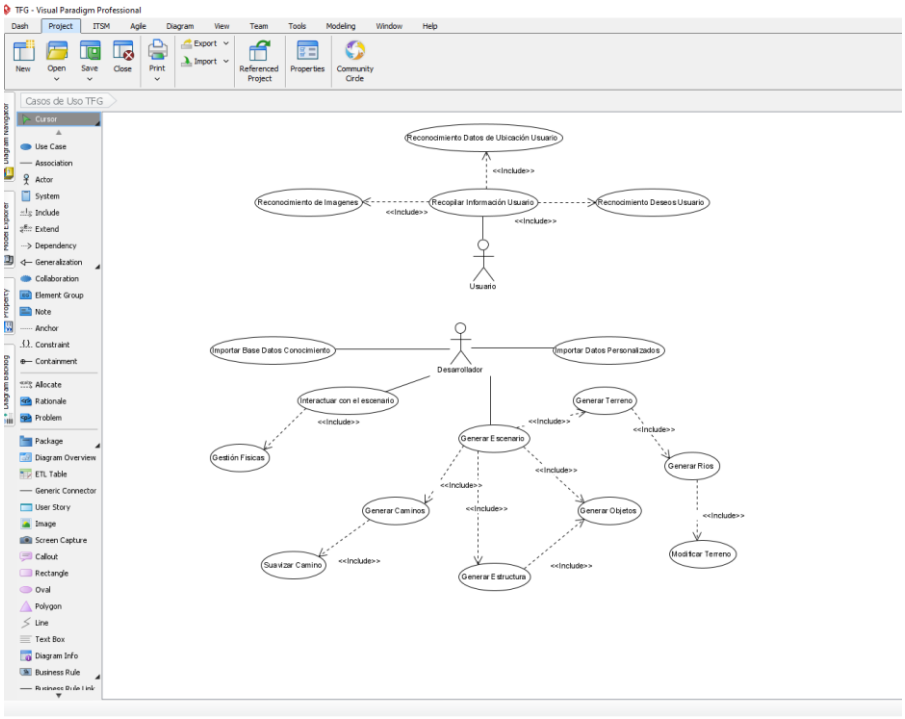


Figura 5.9 Interfaz de visual paradigm

Dentro de la edición de diagramas, Visual Paradigm da soporte a 18 categorías diferentes de diagramas desde diseño de software a diagramas de estrategia empresarial. En la figura 5.10 se puede apreciar esta variedad de tipos de diagramas:

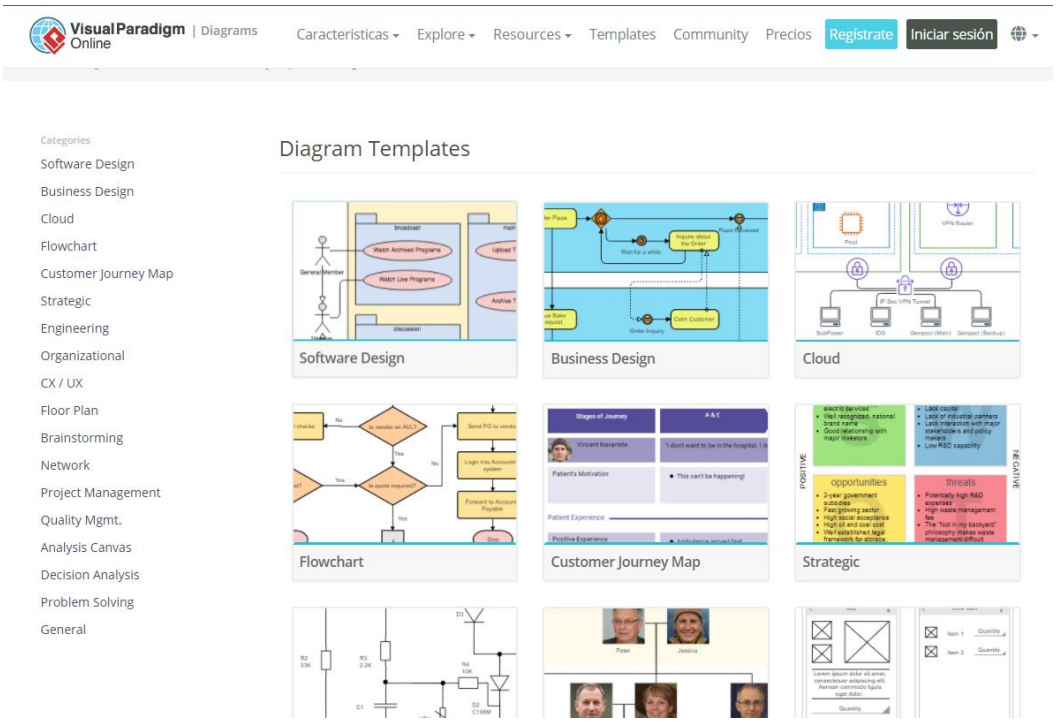


Figura 5.10. Categorías de diagramas soportados por Visual Paradigm

Dentro de los diagramas que nos brinda Visual Paradigm vamos a destacar 3 que han sido los usados para este proyecto:

5.4.1 Diagrama de Casos de Uso

Los diagramas de casos de uso son diagramas enfocados en modelar el comportamiento de un sistema. Está compuesto por Actores, los puntos de entrada a un sistema y los propios casos de uso, que son una descripción de una acción y/o actividad. Estos diagramas se encuentran en un nivel de abstracción bastante alto. En la página anterior, en la figura 5.4 puede verse un ejemplo de este tipo de diagramas.

5.4.2 Diagrama de Clases

Los diagramas de clase se encuentran en un nivel de abstracción mucho más bajo que los diagramas de casos de uso, en un nivel bastante cercano al código, puesto que nos sirven para definir la estructura de clases y métodos que tendrá nuestro programa. Se compone principalmente de Clases que se corresponden con las clases de los lenguajes de programación orientados a objetos. A continuación, la figura 5.11 muestra un ejemplo de este tipo de diagramas.

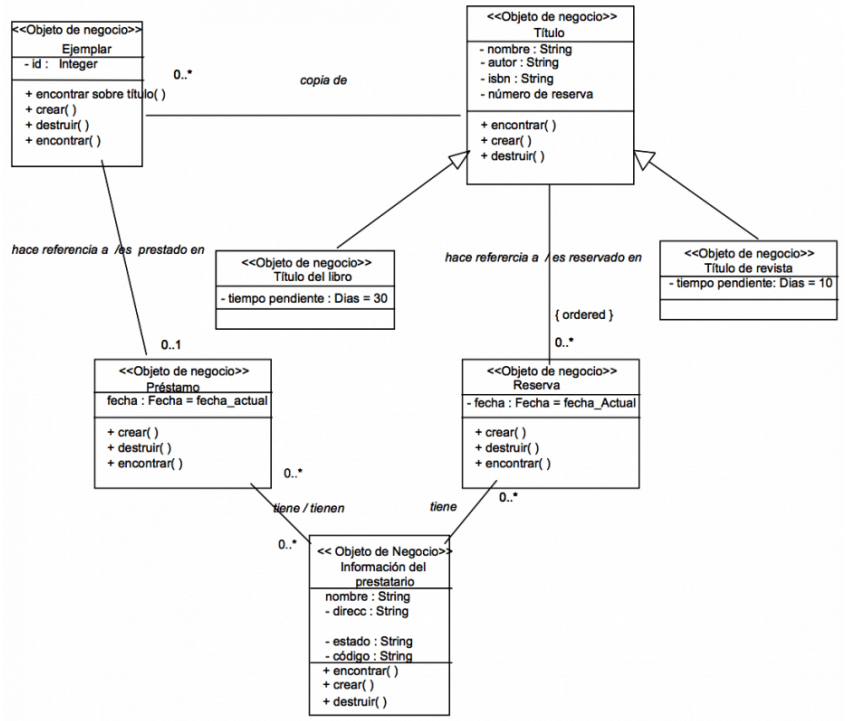


Figura 5.11. Ejemplo de un diagrama de clases

5.4.3 Diagrama de secuencias

Estos diagramas son los que tienen la menor abstracción de los tres comentados puesto que directamente representan el comportamiento del sistema a nivel de métodos y llamadas. Se utilizan para especificar como se comportará un método o función antes de que este sea implementado y sirve para poder entender su funcionamiento sin tener que adentrarse primero en el código. Un ejemplo de este tipo de diagramas puede verse a continuación en la figura 5.12.

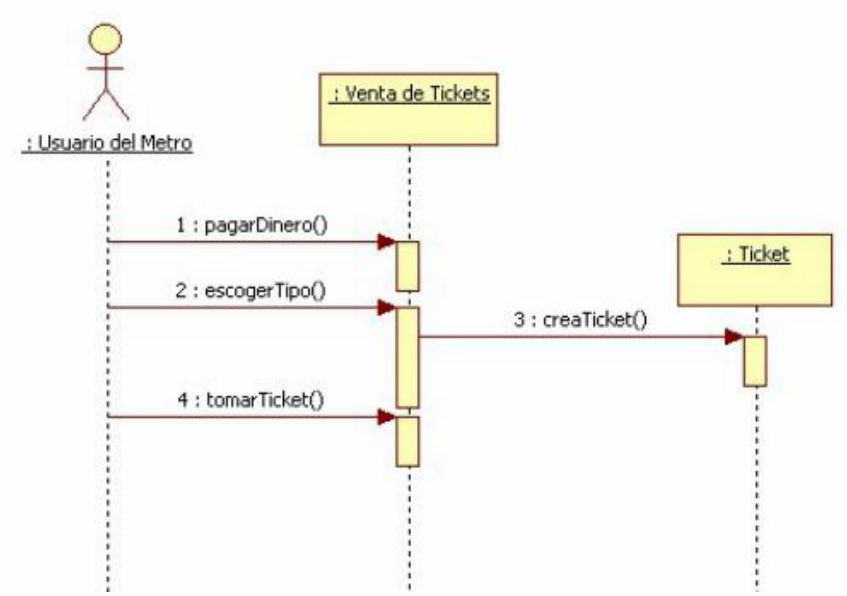


Figura 5.12 Ejemplo de un diagrama de secuencias

5.5 Alturos Yolo

Alturos Yolo es una API que funciona bajo el lenguaje de programación C# y cuya funcionalidad principal es la de detectar objetos sobre imágenes en tiempo real. Cuenta con un conjunto de dataset de distintas resoluciones y cantidades de objetos (yolo9000, yoloV2, yoloV2-tiny, yoloV3, yoloV3-tyny) que permiten al programador definir el nivel de recursos que desea gastar en la tarea de reconocimiento de imágenes. Además, el sistema también permite entrenar la red neuronal para ampliar el conocimiento de esta con nuevos objetos o tipos de objetos, aunque no es una función que se haya utilizado en este trabajo, es algo que considero remarcable de comentar.

El sistema soporta reconocimiento de imágenes por CPU o GPU, siendo este último significativamente más rápido. La siguiente tabla muestra una comparativa entre los tiempos medios de procesar la imagen bird1.png (ver figura 5.13) en tres procesadores Intel versus el tiempo de procesarlas en una tarjeta (ya algo desfasada) Nvidia Geforce GT 1030.

Dispositivo	YOLOv2-tiny	YOLOv3
Intel i7 3770	260 ms	2200 ms
Intel Xeon E5-1620 v3	207 ms	4327 ms
Intel Xeon E3-1240 v6	182 ms	3213 ms
Nvidia GeForce GT 1030	40 ms	160 ms

Tabla 5.2 Comparativa de tiempos de procesado de YOLO. Fuente: Github de Yolo

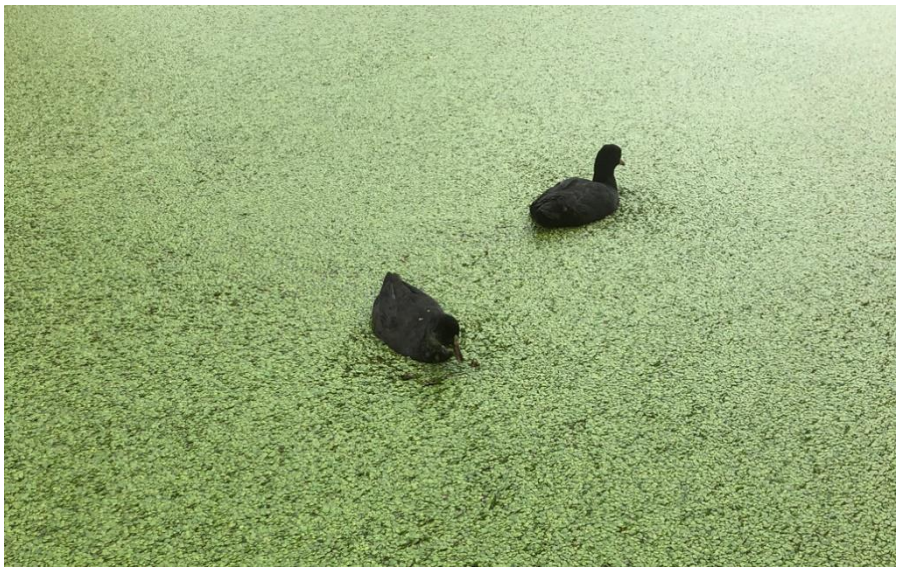


Figura 5.13 archivo bird1.png

5.6 Visual Studio

Visual Studio (véase figura 5.14) es un entorno de desarrollo integrado (IDE) para Windows y macOS desarrollado por la empresa Microsoft. Es compatible con una gran variedad de lenguajes de programación como pueden ser C#, C++, Visual Basic, PHP o Java, así como con entornos de desarrollo web como MVC o Django. En la actualidad está considerado como uno de los entornos de desarrollo más potentes. Entre sus ventajas, las que Microsoft más destaca [14] son:

- Desarrollar aplicaciones para Android, iOS, Mac, Windows, la Web y la nube desde un mismo IDE
- Escribir código con rapidez y de manera fluido
- Depurar y emitir diagnósticos con facilidad a través de la depuración avanzada
- Capacidad de realizar pruebas de código completas
- Entorno personalizable con temas, fuentes, etc.
- Capacidad para ampliarlo y extenderlo a través de complementos
- Colaboración de manera eficiente
- Integración con GIY y otros sistemas de repositorios
- Terminaciones de código inteligentes con IntelCode

Para este proyecto se ha hecho uso de este IDE junto con un proyecto de interfaz gráfica de Windows Form, para toda la parte de recolección de los datos del usuario.

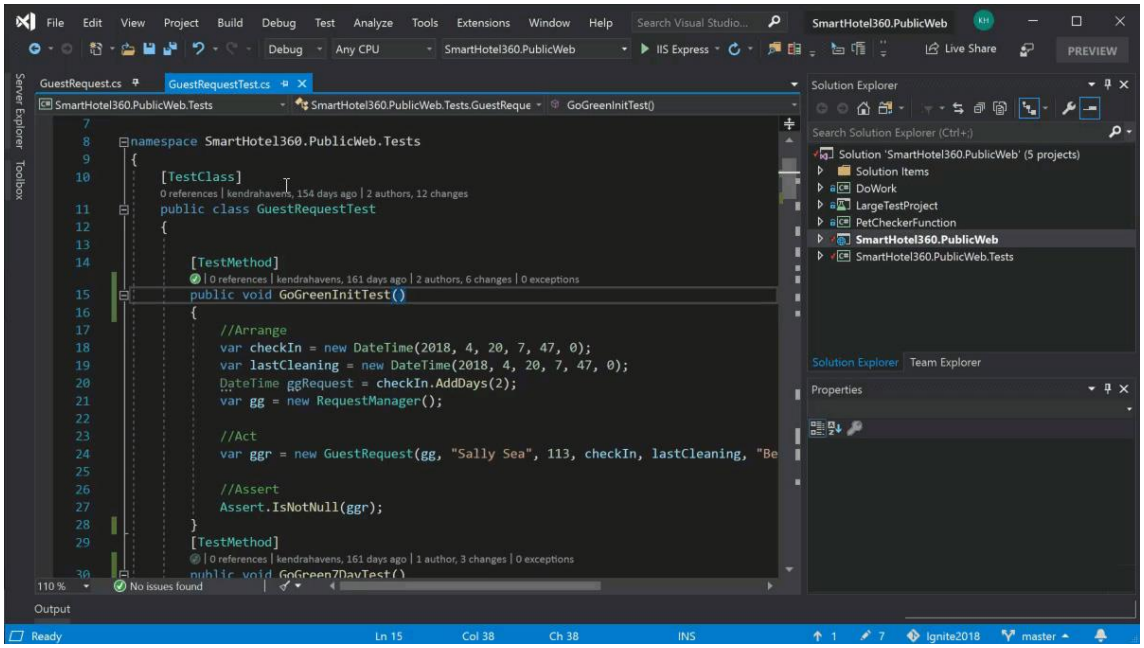


Figura 5.14. Interfaz de Visual Studio 2019

5.7 Unity 3D

Unity3D es un motor de videojuegos (motor de renderizado a tiempo real, véase capítulo 2) creado por Unity Technologies presentado por pinera vez al mundo en la conferencia mundial de desarrolladores de Apple en 2005 como una herramienta orientada a este sistema, aunque más tarde se terminaría abriendo para funcionar también bajo Microsoft Windows y bajo Linux.

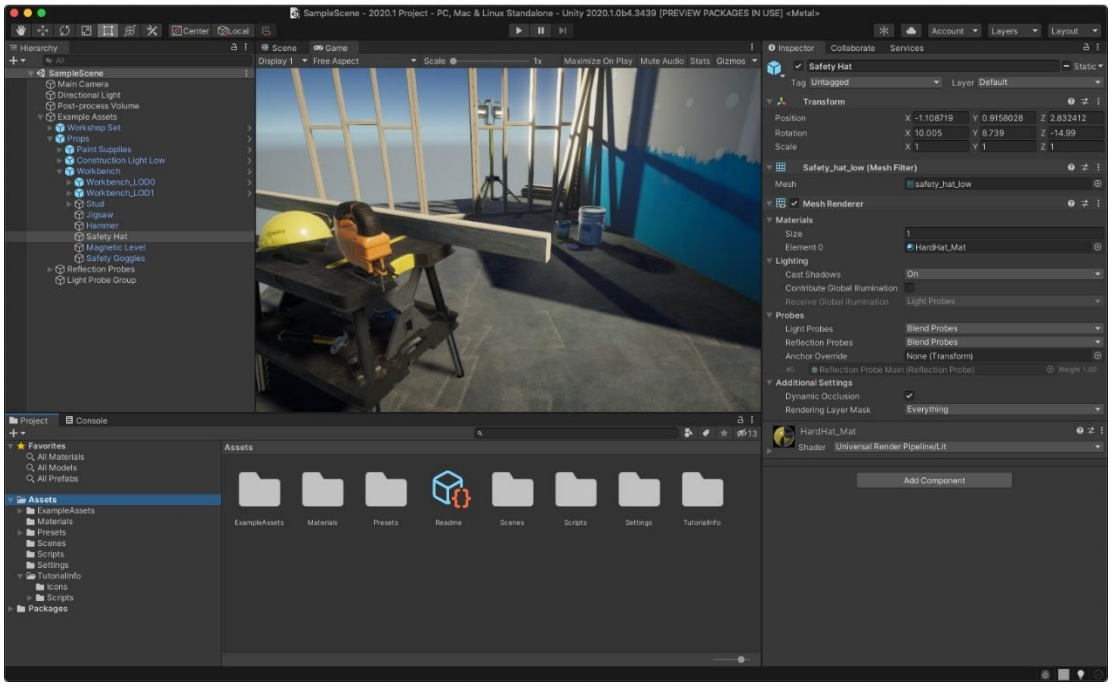


Figura 5.15. Interfaz de Unity 2020.1 Una de las últimas versiones lanzadas

Una de las principales ventajas de Unity3d es que no solo está orientado a los grandes equipos de desarrollo, si no que también lo esta a equipos más pequeños y por ello incluye varias

herramientas orientadas a equipos conformados por pocas personas y con poca diversidad de perfiles técnicos, como puede ser la herramienta de “Mecanim” (Figura 5.16) que permite traspasar animaciones 3D entre distintos modelos 3D humanoides, ahorrando así el trabajo de tener que diseñar las animaciones de cada modelo por separado y permitiendo además crear un complejo sistema de máquinas de estados para las animaciones de los personajes, O la integración básica de programación grafica por componentes, que permite a personal con baja cualificación en programación crear sus propios videojuegos con unas mecánicas simples.

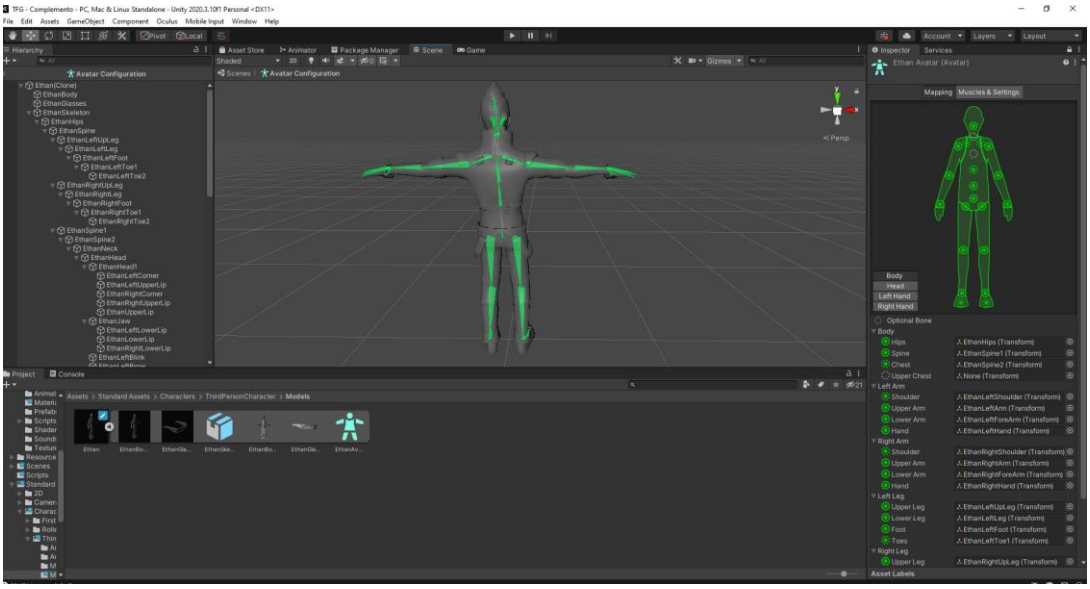


Figura 5.16. Sistema “Mecanim” para el traspaso de animaciones entre personajes humanoides

Adicionalmente, Unity cuenta con la ventaja de que sus proyectos son multiplataformas, pudiendo exportarse a fecha de escribir estas líneas a más de 25 plataformas entre las que destacan las siguientes:

Listado 5.2 Principales plataformas de compilación de Unity3D

La gran cantidad de plataformas que soporta, junto con las facilidades que da a los desarrolladores y el alto nivel visual que es capaz de alcanzar son algunos de los motivos del éxito de este motor y lo que ha hecho que algunos de los estudios más grandes de la industria hayan confiado en él para el desarrollo de alguno de sus proyectos, un ejemplo de ello, es el del archiconocido “Pokemon GO” (Figura 5.17) desarrollado por “Niantic” y financiado por Nintendo y “The Pokémon Company” que fue un éxito rotundo a nivel mundial en 2016 y que causó un auténtico furor entre aficionados y no tan aficionados al mundo Pokémon.

Figura 5.17 Imagen del archiconocido videojuego "Pokémon GO"

Otro ejemplo que nos aleja del pensamiento de que Unity solo se usa para el desarrollo de juegos móviles sería el de "Resident Evil: Umbrella Corps" un videojuego "Shooter del estilo Survival Horror" desarrollado por Capcom para las plataformas PlayStation 4 y Microsoft Windows en el año 2016.

Figura 5.18 Imagen del videojuego "Resident Evil: Umbrella Corps"

5.8 C#

C# es un lenguaje de programación desarrollado por Microsoft en el año 2000 como parte de la plataforma .NET. Su sintaxis inicial deriva de C/C++ y utiliza el modelo de objetos de .NET que es muy similar al de Java, aunque con unas cuantas mejoras respecto a este último.

C# Esta considerado como uno de los lenguajes de programación más populares entre los programadores. Según GitHub Octoverse 2020, el reporte anual de datos que publica el popular sitio GitHub en 2020, C# fue el quinto lenguaje de programación más usado, superando a gigantes de la talla de PHP, C o C++.

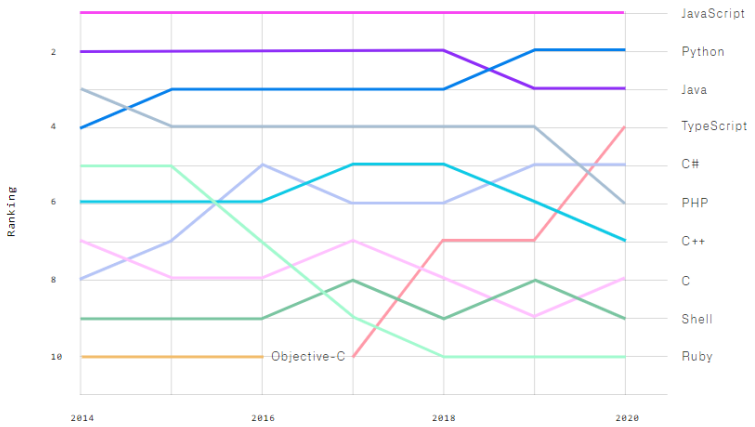


Figura 5.19 Lenguajes de programación más usados desde 2014. Fuente. GitHub Octoverse

Dentro de su estructura de datos, C# es un lenguaje bastante completo, contiene 14 categorías de tipos de datos de valor que se clasifican en 4 categorías.

Los primeros y más básicos son los usados para definir los números enteros, cuyos tipos en c# son los que siguen:

Tipo	Tamaño	Intervalo	Significado
byte	8-bit (1-byte)	0 a 255	Entero sin signo
sbyte	8-bit (1-byte)	-128 a 127	Entero con signo
short	16-bit (2-byte)	-32.768 a 32.767	Entero corto con signo
ushort	16-bit (2-byte)	0 a 65.535	Entero corto sin signo
int	32-bit (4-byte)	-2.147.483.648 a 2.147.483.647	Entero medio con signo
uint	32-bit (4-byte)	0 a 4.294.967.295	Entero medio sin signo
long	64-bit (8-byte)	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Entero largo con signo
ulong	64-bit (8-byte)	0 a 18.446.744.073.709.551.615	Entero largo sin signo

Tabla 5.3 Tipos de datos enteros en C#

A continuación, están los tipos de datos de coma flotante que sirven para representar número decimales, existe 3 tipos en esta categoría, float, double y decimal. El segundo es el más usado para cálculos matemáticos debido a su mayor cantidad de decimales respecto a float, el último, decimal, es utilizado para operaciones monetarias, ya que su objetivo es eliminar los errores de redondeo típicos que suelen tener las operaciones de números de coma flotante.

Tipo	Tamaño	Intervalo	Significado
float	32-bit (4-byte)	±1.401298E−45 a ±3.402823E+38	Coma flotante corto
double	64-bit (8-byte)	±4.94065645841246E−324 a ±1.79769313486232E+308	Coma flotante largo
decimal	128-bit (16-byte)	−7.92281625142643375935439503 35 a +7.92281625142643375935439503 35	Coma flotante monetario

Tabla 5.4 Tipos de datos de coma flotante en C#

Finalmente existen los datos de caracteres y los de tipo lógico. A diferencia de otros lenguajes de programación, en C# los caracteres no tienen un tamaño de 8 bits si no que son de 16 bits debido a que utilizan la codificación Unicode. A continuación, puede verse una tabla estos dos tipos de datos (Caracteres y datos lógicos) junto a sus propiedades.

Tipo	Tamaño	Intervalo	Significado
char	16-bit (2-byte)	'\u0000' a '\uFFFF'	Carácter unicode
bool	8-bit (1-byte)	true o false	Verdadero o falso

Tabla 5.5 Tipos de caracteres y datos lógicos en C#

Además de tipos de datos, c# cuanta con una gran variedad de instrucciones de control, métodos, etc… a continuación se muestran dos ejemplos, uno de la estructura de las instrucciones de control if-else y el segundo de la sintaxis de los métodos, estos ejemplos son interesantes para entender la sintaxis del lenguaje con el que estamos trabajando:

```
if (i == 2)
{
    // ...
}
else if (i == 3)
{
    // ...
}
else
{
    // ...
}
```

Listado 5.3 Sintaxis de if-else en C#

```
void PassRef(ref int x)
{
    if (x == 2)
    {
        x = 10;
    }
}

int z = 0;
PassRef(ref z);
```

Listado 5.4 Sintaxis de métodos en C#

Finalmente, para terminar de ejemplificar la sintaxis de C#, el siguiente listado refleja un ejemplo de un método construido en este lenguaje por el propio alumno usado para controlar las colisiones contra objetos de una flecha (tras ser disparada por un arco).

```
public class Arrow : MonoBehaviour
{
    Mensaje de Unity | 0 referencias
    private void OnCollisionEnter(Collision collision)
    {
        if(collision.transform.GetComponentInParent<Player>()) return;
        this.GetComponent<Rigidbody>().isKinematic = true;
        this.GetComponent<Collider>().enabled = false;
        this.transform.position += transform.forward * 0.5f;
        this.transform.parent = collision.transform;
    }
}
```

Listado 5.5 Método para gestionar colisiones de una flecha en C# (Unity)

5.9 Azure Services

Microsoft Azure es un servicio de computación en la nube diseñado por Microsoft para diseñar, desplegar y administrar servicios y aplicaciones a través de sus propios centros de datos ubicados alrededor de todo el mundo (Véase figura 5.20). Fue lanzado inicialmente el 1 de febrero de 2010 bajo el nombre de Windows Azure, renombrado a Microsoft Azure en 2014.

Figura 5.20 Centros de datos Azure a fecha de diciembre 2020

Los principales servicios que ofrece Azure se pueden clasificar en:

5.9.1 Software como servicio (SaaS)

El software como servicio es un modelo de distribución de software que se instala y se controla en un servidor perteneciente a una compañía (en este caso Microsoft) y al que se accede a través de un cliente. Además, el software es gestionado, mejorado y actualizado por la empresa proveedora, pudiéndose despreocupar el cliente de todo esto.

Dentro de esta categoría se encuentran productos tan conocidos como:

- Office 365. La suite de ofimática de Microsoft
- Dynamic 365. El software de inteligencia de negocio predilecto de Microsoft

Además, Azure también permite alojar software creado por el propio desarrollador para que cumpla con esta categoría de cara a sus clientes.

5.9.2 Plataforma como servicio (PaaS)

Plataforma como servicio es un entorno de desarrollo e implementación completo en la nube, haciendo uso de recursos que permiten desde el diseño de aplicaciones sencillas basadas en la nube hasta el desarrollo de complejas soluciones empresariales. La ventaja de este sistema es que suele ser de "pago por uso", es decir, el cliente solo debe de pagar cuando esté haciendo uso de los servicios que incluye la infraestructura (servidores, almacenamiento y redes). A diferencia de IaaS que comentaremos a continuación, la plataforma como servicio suele incluir también el middleware necesario para ejecutar el software, como pueden ser el sistema operativo o las herramientas de desarrollo, administración, bases de datos, etc.

En esta categoría Azure cuenta con:

- Aplicaciones webs
- Back End de aplicaciones móviles
- Aplicaciones de lógica de procesos
- Funciones
- Trabajos web

5.9.3 Infraestructura como servicio (IaaS)

Finalmente, Infraestructura como servicio es el nombre que reciben aquellos servicios ofrecidos por las empresas para que el cliente pueda hacer uso de los recursos de un servidor en remoto, pero sin tener que depender del software instalado por el proveedor como pasa con el PaaS. Un gran ejemplo de uso de este tipo de sistemas en Azure es el de las máquinas virtuales, que permiten crear y gestionar diversas maquinas con distintos sistemas operativos para desarrollar o desplegar software.

A modo de resumen visual, la siguiente figura muestra lo que se incluye dentro de cada una de estas categorías de software:

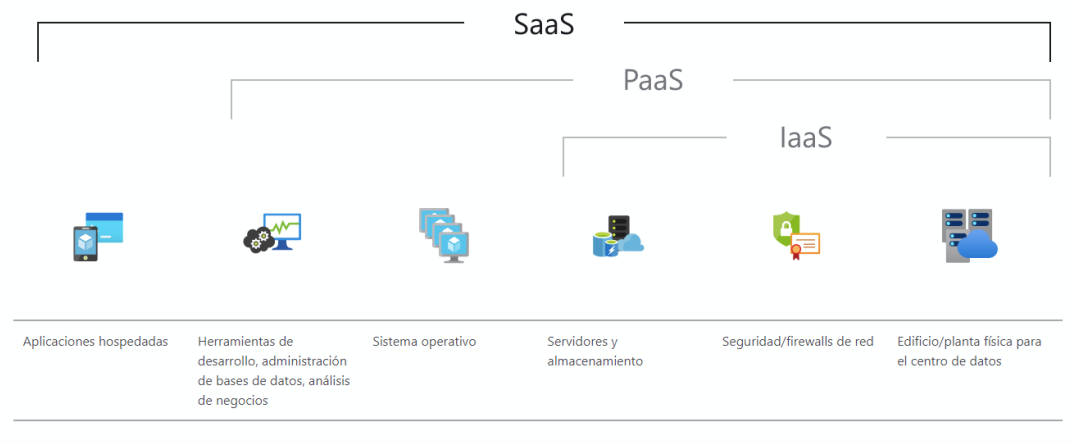


Figura 5.20 Áreas cubiertas por SaaS, PaaS e IaaS en Azure

Finalmente, el servicio utilizado en este trabajo, el de reconocimiento de voz de Azure se enmarca en el primer grupo, puesto que el funcionamiento que tienen es el de una API online, el sistema u ordenador envía una grabación de audio del usuario al servidor de Azure, este procesa el audio y devuelve la cadena de texto generada (Figura 5.21). En el caso de este proyecto, todo este proceso es gestionado por la API de Microsoft Speech Cognitive Services a través de la licencia de uso del alumno de este servicio.

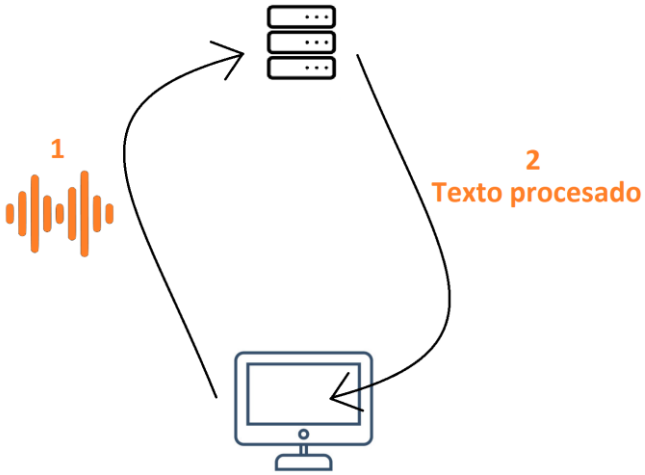


Figura 5.21 Proceso de análisis de la voz

5.10 Synology NAS

Synology es una empresa taiwanesa especializada en soluciones de NAS y servidores. Sus productos se separan en 3 ramas, DiskStation para soluciones de escritorio, FlashStation para soluciones flash y RackStation para soluciones en forma de Rack para servidores. El dispositivo utilizado para este proyecto es un NAS de la gama DiskStation, concretamente el DS218j (Figura 5.22). Pero antes de hablar de este producto, responderemos a la pregunta de ¿Qué es un NAS?

Figura 5.22 NAS Synology modelo DS218j, el mismo que el utilizado en el proyecto

Un NAS, conocido por sus siglas en inglés, Network Attacher Sotrage o almacenamiento conectado en red en castellano. Es el nombre que suelen recibir aquellos sistemas cuya principal función es la de gestionar un almacenamiento (uno o varios discos duros) y compartirlos a través de la red. Los primeros NAS no contaban con muchas funciones más la del almacenamiento, pero en los sistemas modernos un NAS funciona ya más como un pequeño servidor que como un disco duro a secas.

En el caso de Synology, sus NAS cuentan con su propio sistema operativo llamado DiskStation Manager (véase figura 5.23), al cual el usuario puede conectarse remotamente a través de la IP del dispositivo NAS y del puerto 5000. Este sistema operativo funciona sobre un Linux modificado para sistemas x86 y permite la instalación de paquetes (programas) directamente del propio servidor de Synology. A continuación, comentaremos los dos principales paquetes usados para este proyecto.

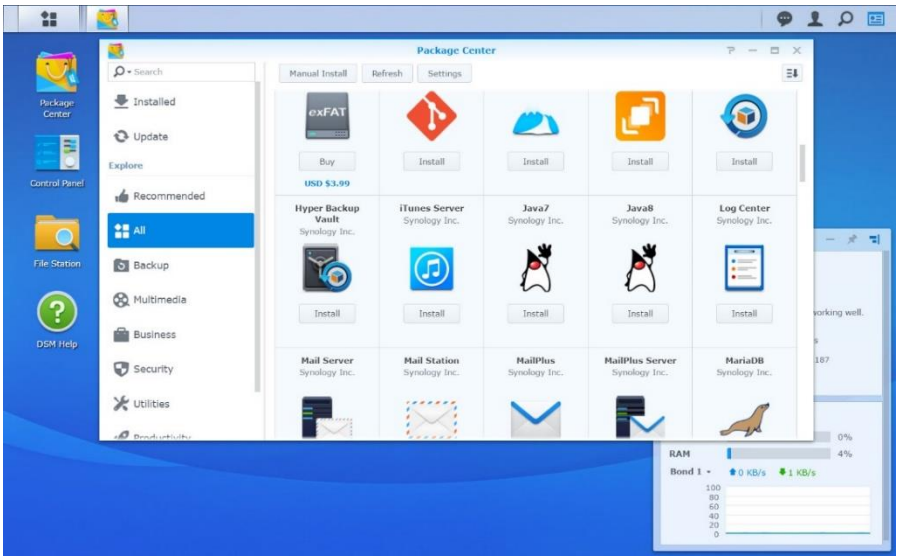


Figura 5.23 Interfaz del sistema operativo de los NAS Synology y del gestor de paquetes

5.10.1 Servidor HTTP Apache

Un servidor apache es un software servidor web HTTP de código abierto multiplataforma, que permite alojar páginas y servicios web. Apache es el software que se utiliza en el 46% de los sitios web de todo el mundo y está mantenido por la Apache Software. Apache está también considerado como uno de los servidores web más antiguos, puesto que su primera versión fue lanzada hace más de 25 años, en 1995.

5.10.2 PHP

PHP es un lenguaje de programación de uso general que se suele usar principalmente para desarrollo web. Fue creado en 1994 y en la actualidad va ya por su octava versión (PHP 8.0.7): Uno de los factores que diferencian el código PHP del de otros lenguajes como el de JavaScript es que PHP se ejecutan en el lado del servidor, es decir, no es el ordenador del cliente el que tiene que ejecutarlo, si no es el propio servidor el que se encarga de procesar este código, de manera que el script PHP se vuelve invisible para el cliente aportando seguridad al sistema. Debido a que el cliente nunca tiene acceso a modificarlo, esto lo hace especialmente útil para establecer conexiones con servicios como bases de datos o para generar APIs. En el listado siguiente puede verse un ejemplo de sintaxis de PHP. Cabe destacar PHP en su momento no fue

diseñado como un lenguaje orientado a objetos, pero en la actualidad tiene varias funciones orientadas a este tipo de lenguajes.

```php
<?php

$a = $_GET["firstname"];  //retrieve the value of First Name input
$b = $_GET["lastname"];  //retrieve the value of Last Name input

  if(substr($a, 0, strlen("<SCRIPT"))=== "<SCRIPT" ) {
    $a=htmlspecialchars($a);   }
   if(isset($b)) {
    $goonb = true;   }
else {
    $goonb = false; }
  if ($goonb ) {
    $b=htmlspecialchars ( $b );   }
    echo $a;   // sensitive s ink
  if ( $goonb ) {
       echo $b;  // sensitive s ink
}
?>
```

Listado 5.6 Ejemplo de sintaxis de un programa escrito en PHP

6 DESARROLLO

6.1 Arquitectura

El desarrollo de este proyecto ha sido diseñado con la visión de que las partes del proyecto sean independientes, e incluso puedan remplazarse según las necesidades de la empresa, el siguiente esquema refleja el funcionamiento de la arquitectura de este sistema.

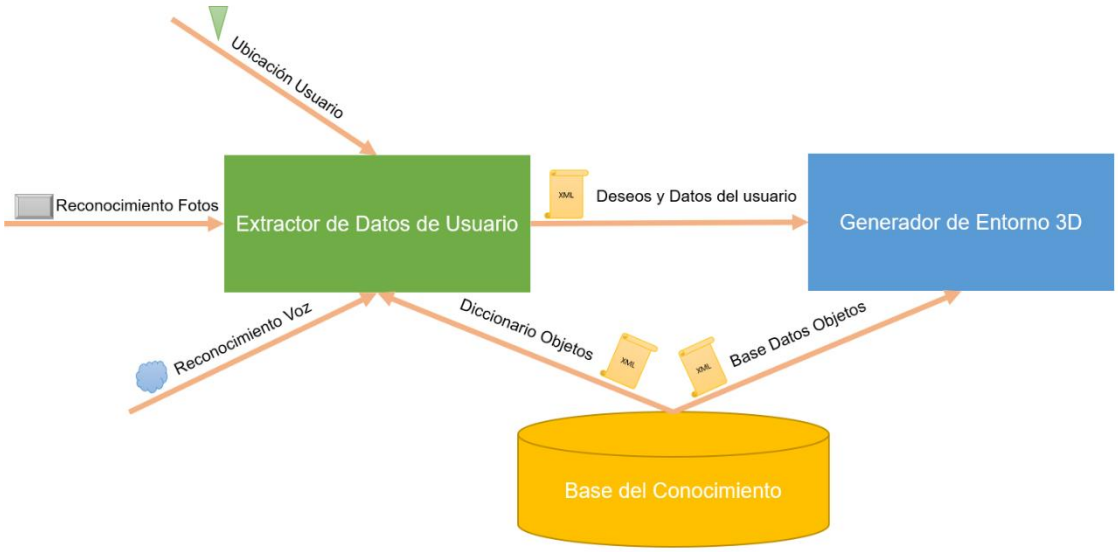


Figura 6.1 Arquitectura del sistema

El sistema se compone de 3 partes separadas (realmente dos y media, más adelante se explicará el porqué) que son:

6.1.1 Módulo de extracción de datos del usuario

Este es el primero de los módulos, su cometido es extraer toda la información que pueda de lo del usuario, esto incluye el reconocimiento de voz sobre lo que el usuario desea generar, pero también información adicional como la geolocalización del usuario o imágenes que el usuario pueda introducir al sistema para extraer información acerca del entorno del usuario.

Para entender lo que el usuario quiere generar, no basta solo con traducir a texto lo que el usuario pide, sino que también es necesario identificar los elementos de dicho texto, como los

números, las propiedades o los propios objetos que el propio sistema más tarde podrá generar, es por ello, qué, junto a los anteriores, el sistema cuenta con otra entrada más. El diccionario o (traductor) de objetos, que traducirá la sintaxis expresada por el usuario a una que el generador posteriormente pueda entender.

La salida generada por este módulo es un archivo XML con la abstracción de datos que el sistema ha podido extraer del usuario.

6.1.2 Moduló del generador de entorno 3D

El módulo del generador de entornos 3D, tiene como objetivo como su propio nombre indica, hacer uso de toda la información previamente extraída para generar un escenario de acorde a los deseos y entorno del usuario.

Recibe como entrada tanto el archivo XML antes generado como otro archivo XML con los datos de la base de conocimiento, que contiene entre otras cosas datos como los modelos 3D que se pueden cargar, las restricciones de generación de dichos modelos o los climas y zonas del mundo en los que estos pueden aparecer.

La salida que este módulo genera es el propio escenario 3D generado dentro de Unity3D.

6.1.3 Base de datos del conocimiento

La base de conocimiento es el último de los 3 módulos del sistema, aunque para el estado actual del sistema debería considerarse más bien “medio módulo” puesto que este sistema está compuesto por dos archivos, el Diccionario de objetos y la base de datos de objetos, pero ambos archivos han sido creados manualmente, no de forma automática. Uno de los principales cambios de cara al trabajo futuro que se comentará más adelante (capítulo 7.3) será el de automatizar la extracción de datos, pudiendo por ejemplo hacer uso de un sistema de inteligencia artificial que extraiga la información de los objetos directamente de internet para luego generar estos dos ficheros.

6.1.4 Conclusiones

Estos tres módulos son los que componen el sistema, siendo capaces cada uno de ellos de funcionar de manera independiente y de ser remplazados de forma separada para adaptarse a las necesidades que los requieran, debido a esto en el siguiente punto, “6.2 Implementación”, se tratarán de manera independiente la construcción de estos sistemas, sobre todo los referentes a los capítulos 6.1.1 y 6.1.2 de este trabajo.

6.2 Implementación

Esta parte cubrirá el proceso de la puesta en marcha de cada uno de los subsistemas del proyecto. Para facilitarle al lector la comprensión de esta parte, esta última ha sido subdividida en 4 apartados, Extractor de datos, que cubrirá el módulo de la extracción de datos, Generador de entorno 3D que cubrirá el sistema de generación en Unity3D, Base del conocimiento, que cubrirá el funcionamiento y la estructura de los archivos XML de la base de conocimiento y del diccionario de datos, y finalmente, Escenario de pruebas, que cubrirá la última parte del proyecto consistente en el desarrollo del banco de pruebas para probar el buen funcionamiento de este.

6.2.1 Extractor de datos

Este es el primero de los módulos por desarrollar, a diferencia del generador de entornos que se ha desarrollado en Unity3D, este se ha desarrollado haciendo uso de Visual Studio, concretamente de un proyecto de Windows Forms. ¿Podría haberse desarrollado en Unity junto con el otro módulo? Si, pero la principal ventaja de este sistema es que ambos módulos son independientes y se pueden remplazar sin tener que acceder siquiera al otro. Además de esta manera también se diferencian los roles de ambos proyectos, quedando el generador en mano de los desarrolladores y pudiéndose usar el extractor de datos no solo por estos, si no por personas de otros perfiles, como puedan ser administrativos o directivos, sin que tengan que aprender el funcionamiento de una herramienta de desarrollo como pueda ser Unity3D.

Antes de empezar a diseñar nada, es conveniente importar las librerías que necesitaremos para este proyecto que concretamente son 3:

- **Microsoft Cognitive Services Speech.** La librería encargada de comunicarse con los servicios de reconocimiento de voz de Azure en la nube.
- **Alturos Yolo.** Librería que gestiona el reconocimiento de objetos en las imágenes
- **Alturos YoloV3Data**. Modelo de datos pree entrenado que utiliza la red neuronal de la librería anterior. Alturos cuenta con 5 modelos distintos cuya principal diferencia es el peso del modelo, así como la resolución de las imágenes con las que ha sido entrenado. (La tabla 6.1 compara los datos de estos modelos). De estos modelos, se usará para este proyecto YOLOv3 que es el más moderno y amplio de los cinco (aunque también el más lento).

Modelo	Resolución	Peso del modelo
YOLOv3	608x608 px	236 Mb
YOLOv3-tiny	416x416 px	34 Mb
YOLOv2	608x608 px	195 Mb
YOLOv2-tiny	416x416 px	42 Mb
Yolo9000	448x448 px	186 Mb

Tabla 6.1 Diferencias entre los distintos modelos de Yolo

Para importar estas librerías lo haremos a través del gestor de paquetes de Nuget De VisualStudio, Microsoft Cognitive Services se puede descargar desde la interfaz gráfica, Alturos Yolo parece ya no tiene servicio y hay que descargarlo desde la línea de comandos de Nuget. Finalmente descargaremos la versión de YOLOv2-tiny para hacer las pruebas, aunque posteriormente la sustituiremos por la versión V3 completa. A continuación, se puede apreciar como deberían quedarnos los paquetes una vez instalados:

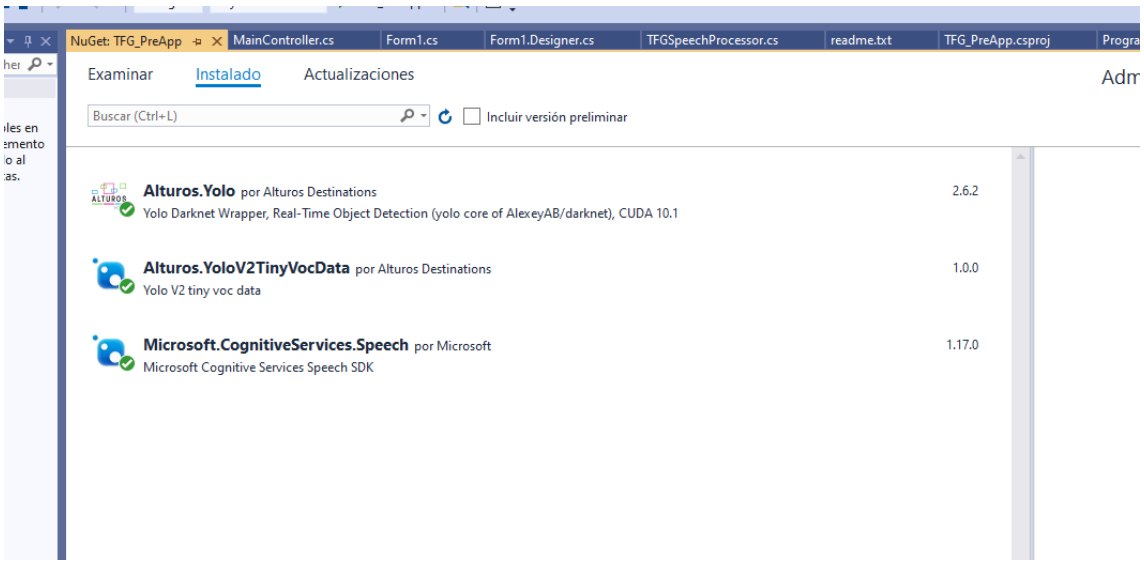


Figura 6.2 Paquetes/Librerías usadas para el módulo de extracción de datos

Con las librerías ya instaladas, ya podemos comenzar a diseñar el proyecto y lo primero que diseñaremos para este módulo será su interfaz gráfica, para la cual haremos uso de las herramientas que nos brinda Windows Form, al crear el proyecto, por defecto se nos habrá creado un archivo llamado Form1.cs que renombremos a "VentanaPrincipal" y será el archivo sobre el que trabajaremos para generar la interfaz gráfica de este proyecto, por defecto usaremos los siguientes elementos gráficos:

- **Button:** Genera un botón que activa un evento cuando es pulsado
- **TextBox:** Genera un cuadro de texto en el que el usuario puede escribir datos
- **PictureBox:** Genera un cuadro en el que se pueden insertar y visualizar imágenes
- **DataGridView:** Permite ver a modo de tabla una lista de objetos y sus propiedades
- **Lavel:** Este componente consistente únicamente en un texto con el que no se puede interactuar, pero que se puede editar a nivel de código para mostrar distintos mensajes.

La pantalla se dividirá en varias áreas que son las que siguen:

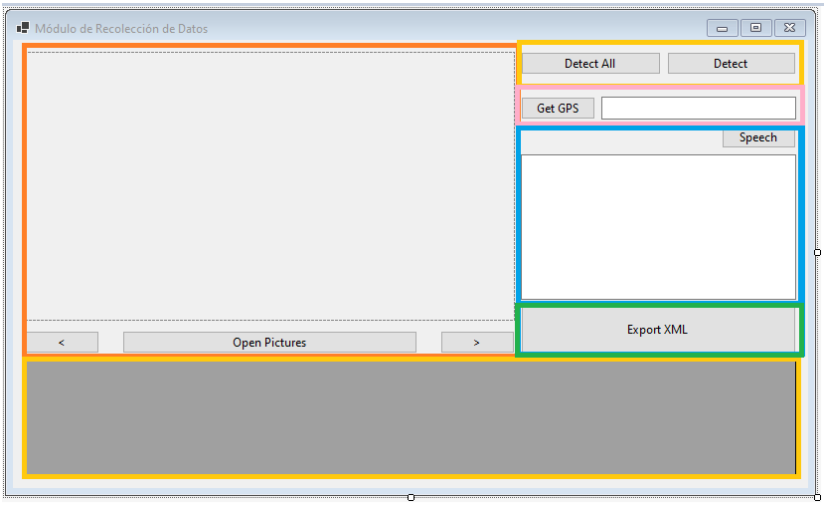


Figura 6.3 Partes de la Interfaz del extractor de datos

La zona naranja se corresponde la apertura y gestión de las imágenes. La amarilla superior con el procesado de estas mientras que la inferior es la diseñada para mostrar el resultado de dicho análisis. La parte rosa comprende la obtención de la geolocalización del usuario. La azul se encarga de gestionar el reconocimiento de voz. Finalmente, la verde se encarga de la exportación de los datos generados a XML.

Pero vayamos por pasos, empezaremos con la importación de imágenes, que es un pilar necesario para varios de los otros módulos. Lo primero que deberemos hacer antes ponernos a programar, es darles un nombre a los distintos elementos de la interfaz gráfica, para poder así luego "llamarlos" e interactuar con ellos desde el código. Esto se puede hacer sobre el panel de propiedades del objeto que queramos modificar.

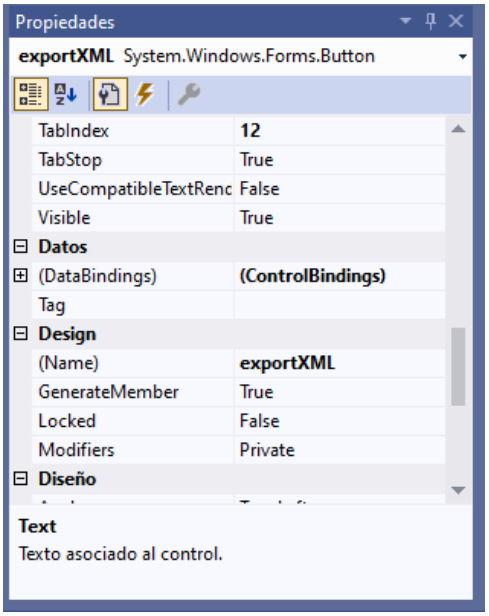


Figura 6.4 Panel de propiedades de los elementos de la interfaz gráfica de Windows Forms

Una vez dados todos los nombres ya podemos empezar a programar. Inicialmente esta parte del proyecto se pensó como una librería de código, aunque finalmente acabo por convertirse en una aplicación independiente, es por ello por lo que en el diseño de la propia aplicación existe un método MainController que funcionaría como punto de entrada al Backend de la aplicación, mientras que VentanaPrincipalDesigner se encarga principalmente de controlar el "FrontEnd".

Una vez hecho el anterior paréntesis, comenzaremos ahora sí con el código, haremos doble clic sobre el botón "Open Pictures" que nos abrirá el script "VentanaPrincipalDesigner.cs" y concretamente el método que se lanza al activarse el evento de hacer clic en el botón en tiempo de ejecución. Su funcionamiento es el siguiente:

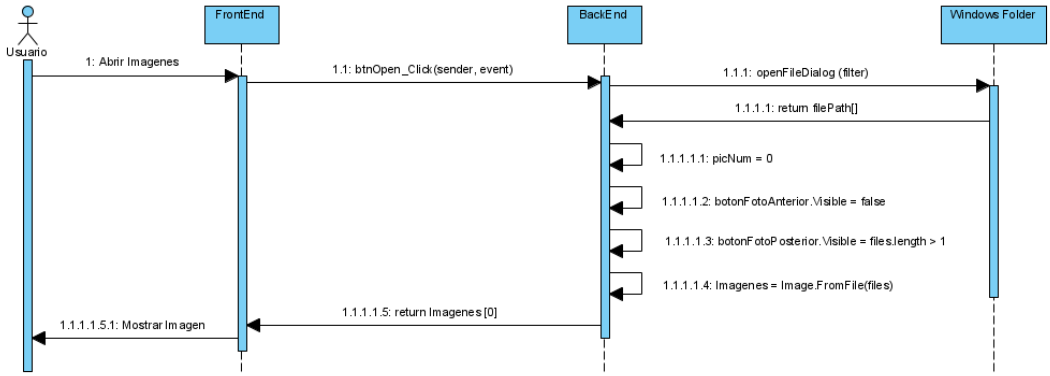


Figura 6.5 Diagrama de secuencias de la carga de imágenes

Como se puede ver, en este caso se permite la carga múltiple de imágenes (pulsando control + clic sobre las imágenes a seleccionar), de esta manera se pueden procesar y reconocer varias imágenes simultáneamente, pero por contrapartida aumentará la cantidad de datos en memoria y hará que debamos tener un sistema para ir pasando entre las distintas fotos. Esto se ha solucionado guardando las imágenes en un array y colocando dos botones, de avance y retroceso que permitan ir pasando de unas imágenes a otras.

El siguiente módulo que desarrollar es el del reconocimiento de imágenes con YOLO, para esta parte se usarán 2 botones, “detectar” y “detectar todo” en castellano, que analizarán una imagen o todas (a través de un bucle analizándolas una a una) respectivamente. Esto se ha diseñado así por una cuestión de tiempo de procesamiento, puesto que el tiempo de procesamiento de cada imagen es de media unos 4 segundos, por lo que el tiempo de procesado para 5 o más imágenes empieza a ser un tiempo de relativamente largo para un usuario cualquiera, por lo cual, permitirle al usuario ir analizando las imágenes una a una, puede amenizar la sensación de espera del propio usuario, sobre todo cuando el número de imágenes es relativamente pequeño.

Internamente el funcionamiento del método es bastante sencillo pues toda la gestión la hace el propio plugin de YOLO por lo que simplemente se deberá inicializar un nuevo “YoloWrapper” con los datos del modelo que se quiere usar (YOLOv3 en este caso), y se analiza el array de bytes que componen la imagen que se quiere utilizar. Como resultado, obtenemos un “IEnumerator” (que convertiremos a lista) de Yolo Ítems, una estructura de datos de Yolo cuyos parámetros pueden verse en la tabla 6.2. a nivel de código está sería la sencilla implementación que deberemos utilizar para usar este sistema:

```
2 referencias
public static List<Alturos.Yolo.Model.YoloItem> ImageDetector(byte[] data) {
    using (var yoloWrapper = new YoloWrapper("yolov3.cfg", "yolov3.weights", "coco.names"))
        return yoloWrapper.Detect(data).ToList();
}
```

Listado 6.1 Llamada a la API de YOLO para procesar las imágenes

Yolo ítem	Resultado del análisis de una imagen, representa los datos de cada uno de los objetos identificados en una imagen	
Dato	**Tipo**	**Descripción**
Type	String	El tipo de objeto detectado (Coche, Camión, etc.)
Confidence	Double	El grado de seguridad de la red de haber detectado un objeto correctamente, tiene un valor 1 a 0 (1- totalmente seguro, 0 nada seguro)
X	Int	Posición en X del vértice Inferior-Izquierdo del rectángulo que contiene al objeto detectado
Y	Int	Posición en Y del vértice Inferior-Izquierdo del rectángulo que contiene al objeto detectado
Width	Int	Altura en píxeles del rectángulo que contiene al objeto detectado
Height	Int	Anchura en píxeles del rectángulo que contiene al objeto detectado
Center ()	Point	Punto que representa el centro del cuadrado a partir de los datos anteriores

Tabla 6.2 Estructura del Objeto Yolo Ítem, resultante del análisis de una Imagen con YOLO

Pero para este proyecto no trabajaremos directamente con Yolo Ítem si no como una extensión de este (una clase heredada), "Extended Yolo ítem", que contendrá además de los parámetros que ya tiene Yolo Item, con uno nuevo llamado color y un método llamado MainColor.

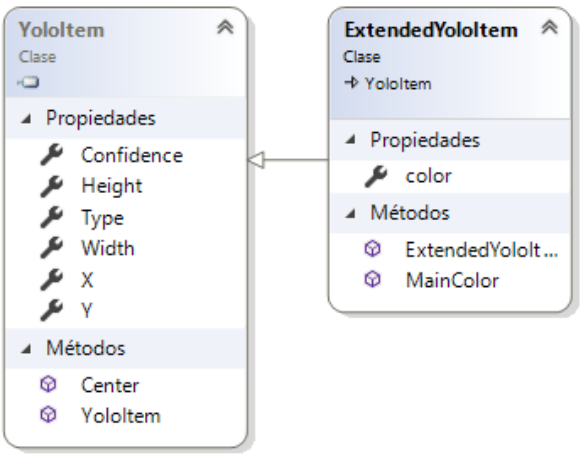


Figura 6.6 Clases YoloItem y ExtendedYoloItem

Esta lista de objetos Yolo Ítems la recogemos y convertiremos a "Extended Yolo ítem" para almacenarla en un array de listas donde cada posición del array representará la imagen analizada para ese objeto y la lista interior corresponderá con cada uno de los objetos identificados para las imágenes analizadas. Por desgracia la cantidad de objetos que puede detectar el algoritmo de YOLO preentrenado por defecto (sin entrenarlo nosotros con nuevos objetos) es limitado. Y solo puede reconocer los siguientes tipos de objetos:

person	*tie*	*donut*	*dog*	*knife*
bicycle	*suitcase*	*cake*	*horse*	*spoon*
car	*frisbee*	*chair*	*sheep*	*bowl*
motorbike	*skis*	*sofa*	*cow*	*banana*
aeroplane	*snowboard*	*pottedplant*	*elephant*	*apple*
bus	*sports ball*	*bed*	*bear*	*sandwich*
train	*kite*	*diningtable*	*zebra*	*orange*
truck	*baseball bat*	*toilet*	*giraffe*	*broccoli*
boat	*baseball glove*	*tvmonitor*	*backpack*	*carrot*
traffic light	*skateboard*	*laptop*	*umbrella*	*hot dog*
fire hydrant	*surfboard*	*mouse*	*handbag*	*pizza*
stop sign	*tennis racket*	*remote*	*clock*	*toaster*
parking meter	*bottle*	*keyboard*	*vase*	*sink*
bench	*wine glass*	*cell phone*	*scissors*	*refrigerator*
bird	*cup*	*microwave*	*teddy bear*	*book*
cat	*fork*	*oven*	*hair drier*	*toothbrush*

Listado 6.2 Objetos Reconocibles por YOLO

Finalmente, esta información la mostraremos en el DataGridView que se encuentra en la parte inferior de la pantalla (Ver figura 6.3) para cada una de las imágenes (solo se mostrarán los objetos de la imagen actual, no todos), y se dibujará el rectángulo (en color verde) de cada objeto sobre la imagen analizada como se puede ver a continuación:

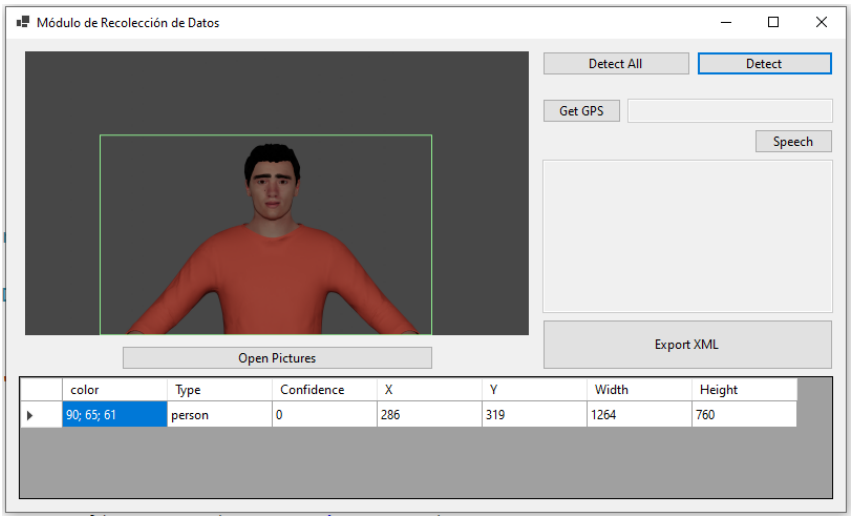


Figura 6.7 Resultado del análisis de una imagen usando YOLO

Una vez finalizado el proceso de Análisis de cada objeto, se usarán los datos del tipo de objeto y su ubicación, para extraer de cada uno el color medio de dicho objeto para guardarlo en la variable color anteriormente mencionada, para ello se ha diseñado una función en C# (la ya mencionada MainColor) que hace una implementación del siguiente pseudocódigo:

```
Color MainColor (imagen, yoloItem)
        bitMap = bitMap (imagen)
        r,g,b = 0
        for (x = yoloItem.X; x < Min(yoloItem.X+yoloItem.Width, imagen.width); x++)
                for (y = yoloItem.Y; y < Min(yoloItem.Y+yoloItem.Height, imagen.height); y++)
                        pixelColor = bitMap.GetPixel(x,y)
                        r += pixelColor.R/totalPixeles
                        g += pixelColor.G/totalPixeles
                        b += pixelColor.B/totalPixeles
        return Color.FromRGB (r, g, b)
```

Listado 6.3 Función para obtener el color medio de un objeto

El color obtenido por esta función se utilizará posteriormente para adaptar el color de los objetos cuando se generen dentro de Unity3D más adelante.

El tercer módulo que compone este sistema es el módulo de reconocimiento de Geolocalización, este es quizás uno de los más especiales del proyecto puesto que supone la implementación de código en un nuevo lenguaje de programación, respecto a lo escrito hasta ahora, que es PHP, cierto es que Visual Studio tiene su propio método para extraer la localización del usuario, pero es una implementación antigua que da muchos problemas en las últimas versiones. Por lo que finalmente se optó por este método que consiste en un archivo PHP alojado en los servidores de la empresa del alumno, que identifica la dirección IP de la conexión y la manda a una API web (ipinfo.io), que devuelve como resultado un archivo JSON que rápidamente es tratado por PHP para obtener como resultado información acerca los datos asociados a la IP del usuario como pueden ser la geolocalización, el país, la ciudad o el proveedor de los servicios de internet del usuario, aunque de estos datos, solo haremos uso del primero. El siguiente listado muestra el código del script PHP y lo fácil que puede ser obtener los datos de ubicación de un usuario a partir solamente su dirección IP (el identificador de su rúter wifi en internet). Muchos estamos ya acostumbrados a este hecho, por ejemplo, en los típicos anuncios del estilo de “Millonarios de Almería (o de cuál sea la ciudad desde la que se conecte) no quieren que se entere de…”, pero, aun así, y sobre todo para los lectores menos versados en los aspectos técnicos de la informática, les sorprenderá saber que con solo 3 líneas de código ya se puede obtener toda esta información:

```php
<?php
$ip = $_SERVER['REMOTE_ADDR']; //Obtiene la dirección IP del cliente
$details = json_decode(file_get_contents("http://ipinfo.io/{$ip}/json")); //Obtiene los datos de esa dirección IP
echo $details->loc; //Imprime la geolocalización del usuario
?>
```

Listado 6.3 Código PHP para obtener la ubicación de un usuario a partir de su dirección IP pública

Una vez generado este archivo, el próximo paso será el de subirlo al servidor para poder ejecutarlo. Lo primero será conectarse al servidor de la empresa e iniciar sesión en el portal de acceso.

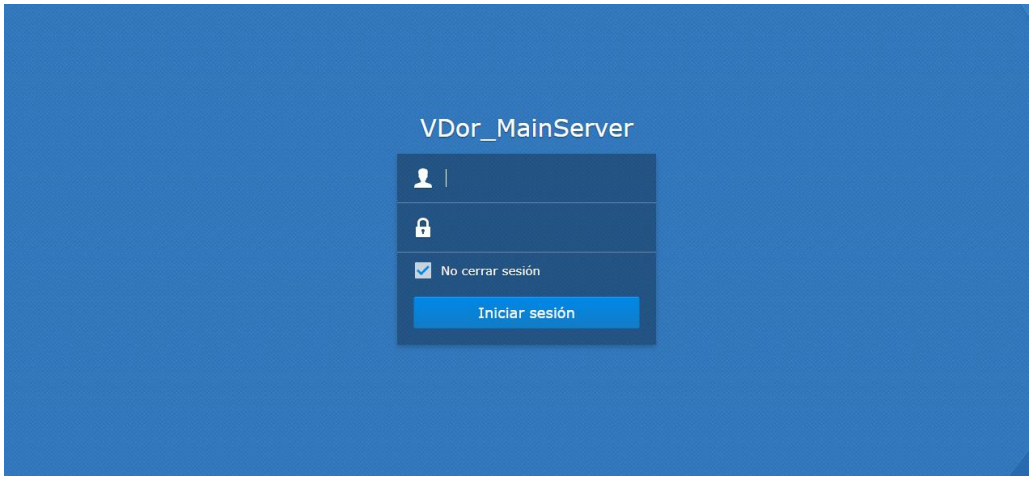


Figura 6.8 Portal de acceso al servidor de la empresa

Una vez iniciamos sesión con las claves, accederemos a la interfaz gráfica de la administración del servidor, nos dirigiremos al sistema de ficheros y desde ahí abriremos la carpeta “web” donde se alojan los datos del servidor web apache, una vez dentro crearemos una carpeta llamada TFG y ahí dentro subiremos nuestro archivo:

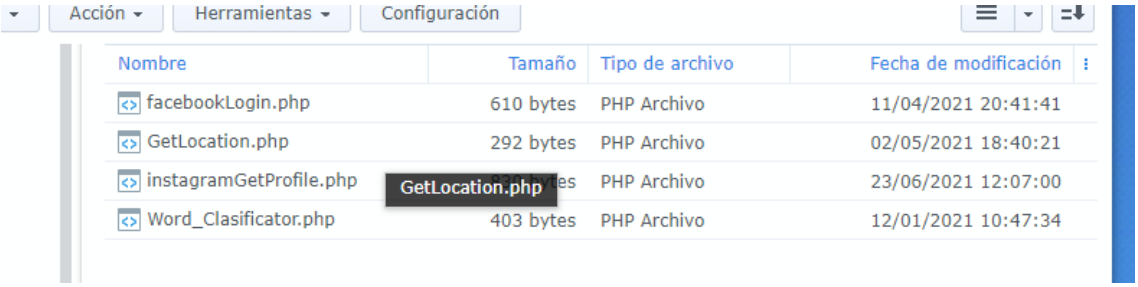


Figura 6.9 Archivo PHP subido al servidor

Como se puede observar existen también otros archivos PHP para la gestión de los datos de Instagram y Facebook (matriz) (funcionalidades diseñadas pero que no se llegaron a aplicar (Léase capítulo 1)) y el sistema original de clasificación de palabras del reconocimiento de voz que estaba basado en una base de datos SQL pero que se acabó sustituyendo por el actual sistema (esto se explicará más adelante). Con el archivo ya copiado ya podemos probar la conexión para saber si el sistema esta funcionando. Y como veremos a continuación, así es.

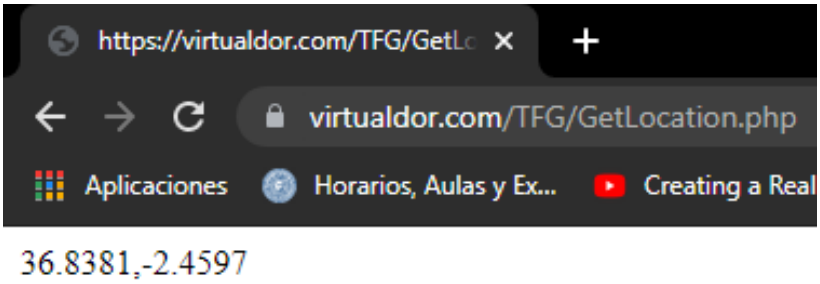


Figura 6.10 Sistema de obtención de la Geolocalización del usuario

Antes de continuar, quisiera hacer una aclaración, y es que para que el anterior script funcione, la conexión debe ser segura y del tipo (HTTPS) y por tanto requiere de un certificado de autentificación del servidor de válido. El servidor usado inicialmente no implementaba HTTPS sino HTTP, por lo que para hacer este script funcionar se debieron modificar varias configuraciones del servidor y generar un certificado válido generado por una entidad externa para este (Ver figura 6.11). Por motivos de seguridad y privacidad este proceso no puede ser mostrado en el presente trabajo, pero consideraba relevante comentarlo.

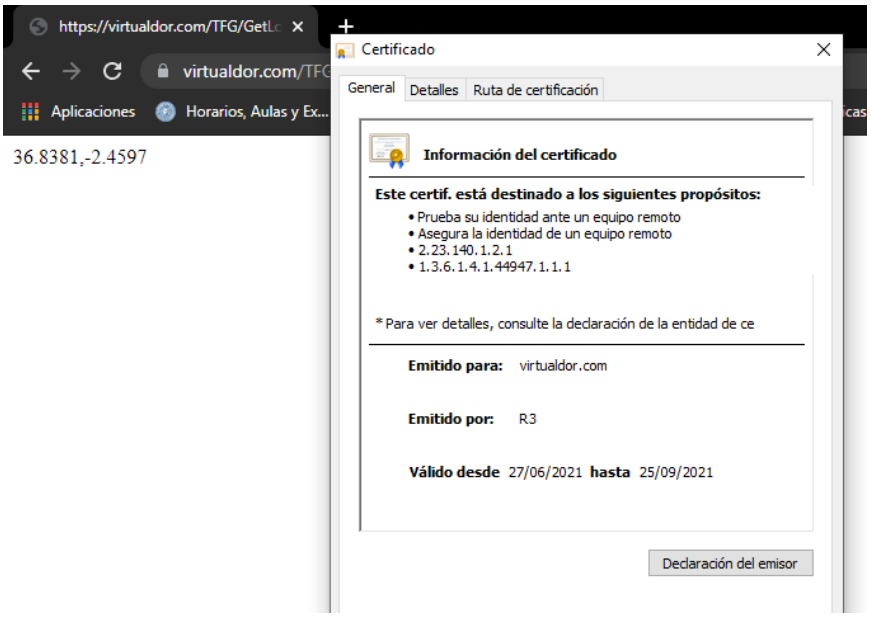


Figura 6.11 Certificado que acredita que la conexión con el servidor es segura

Una vez hechos todos los pasos anteriores ya solo nos queda recuperar esta información desde nuestra herramienta en C#, para ello se hará uso del método asíncrono "GetGeoPosition", que forma parte de este tipo de métodos debido a que la respuesta del servidor no es inmediata, si no que tarda un tiempo en procesar la respuesta antes de devolverla, por lo que nuestro método deberá esperar a esa respuesta antes de poder continuar. A modo de resumen de esta parte desde que el usuario pulsa el botón "Get GPS" hasta que recibe una respuesta, la interacción es la siguiente:

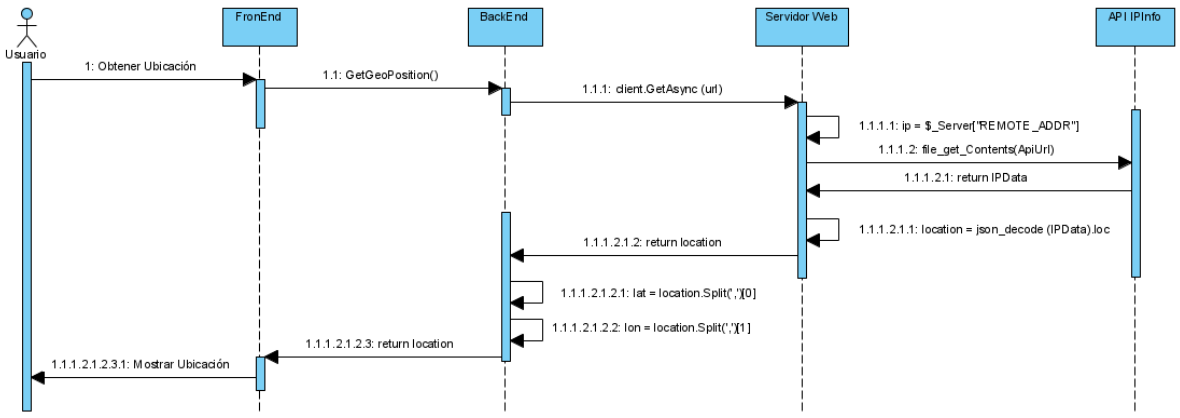


Figura 6.12 Diagrama de secuencias del proceso de obtener la Geolocalización del usuario

Finalmente, como se indica arriba, tras recibir la respuesta del servidor, los valores de la localización se guardan en sus propias variables y se le muestran al usuario a través de una caja de texto (desactivada para que el usuario no pueda escribir sobre ella). Con esto quedaría concluido este módulo y podemos dar ya paso al siguiente.

El siguiente módulo, y ya el penúltimo de esta parte es el del reconocimiento de voz de Azure, este al igual que el anterior es algo especial debido a que requiere de sistemas en la nube (Concretamente la de Azure) para funcionar, por lo que, para usar este módulo, el primer paso (Excluyendo la importación de librerías que hicimos al principio del capítulo) será el de dar de alta este servicio en la nube de Azure.

Para entrar en el portal de Azure podemos hacerlo a través de la siguiente dirección web: https://azure.microsoft.com/es-es/features/azure-portal/ Una vez dentro deberemos pulsar en Iniciar Sesión y la iniciaremos con nuestra cuenta o en caso de no tener una, podemos crearla. Para este proyecto usaremos la cuenta vinculada al correo electrónico que nos proporciona la UAL al entrar en la universidad de la misma manera que lo hicimos para la asignatura de "Herramientas y Métodos de Ingeniería del Software". Una vez dentro del portal veremos una interfaz como esta:

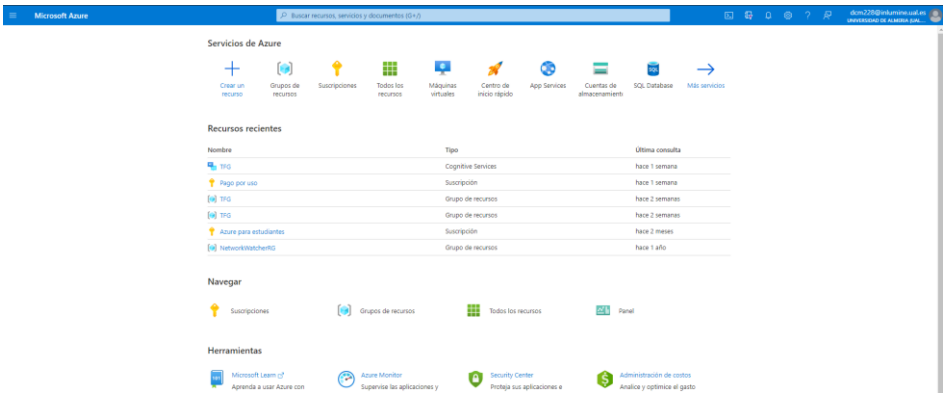


Figura 6.13 Interfaz web de la plataforma de Azure

Una vez dentro deberemos de generar un nuevo recurso pulsando sobre el botón de "Crear un Recurso" que encontramos en la esquina superior-izquierda de la interfaz. Se nos abrirá un menú con los distintos elementos que podemos utilizar. En nuestro caso haremos uso de la búsqueda que integra la ventana para buscar "Speech" y seleccionaremos el recurso que nos aparece el primero, que será el siguiente:

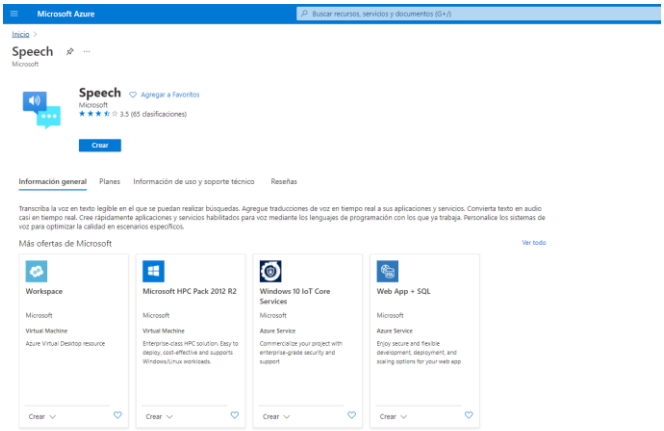


Figura 6.14 Recurso de reconocimiento de Voz de Microsoft Azure

Una vez en esta ventana pulsaremos sobre crear, le daremos un nombre, elegiremos una subscripción (en este caso la suscripción de pago por uso, puesto que la de Azure para

estudiantes está caducada), elegiremos la ubicación del servidor (lo más recomendable suele ser elegir el servidor que se encuentre geográficamente más cerca de nosotros, para reducir al máximo la latencia de las comunicaciones) y el plan de tarifa, que en este caso será F0, el gratuito. En último lugar deberemos elegir el grupo de recursos al que queremos asignar este nuevo recurso, si no tenemos uno, podremos crearlo en esta misma ventana, pinchando sobre "Crear Nuevo".

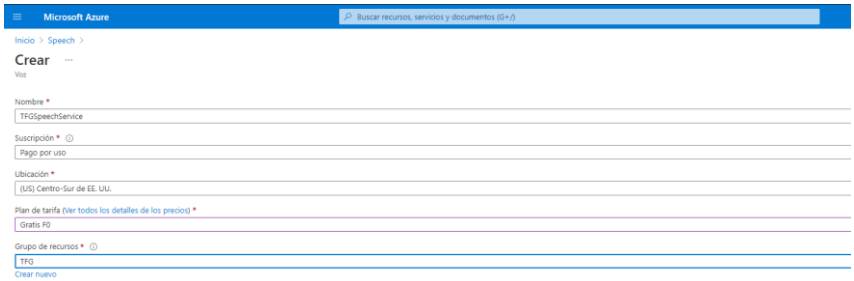


Figura 6.15 Ventana de creación de recursos

Finalmente pulsamos sobre el botón y Azure empezará a generarnos ese recurso, lo que tardará un rato. Antes de cerrar el navegador, hay dos cosas que nos convienen mirar, en primer lugar, la clave de nuestro recurso de reconocimiento de voz y en segundo el servidor en el que se encuentra (el que hemos especificado anteriormente).

Para obtener estos datos, deberemos esperar a que el recurso termine de crearse y una vez hecho, pulsar en el y acto seguido entrar en la sección Claves y puntos de conexión. De aquí deberemos copiar la "Clave 1" y la Ubicación ya que estos son los datos que nos requerirá la API en Visual Studio para poder funcionar correctamente.

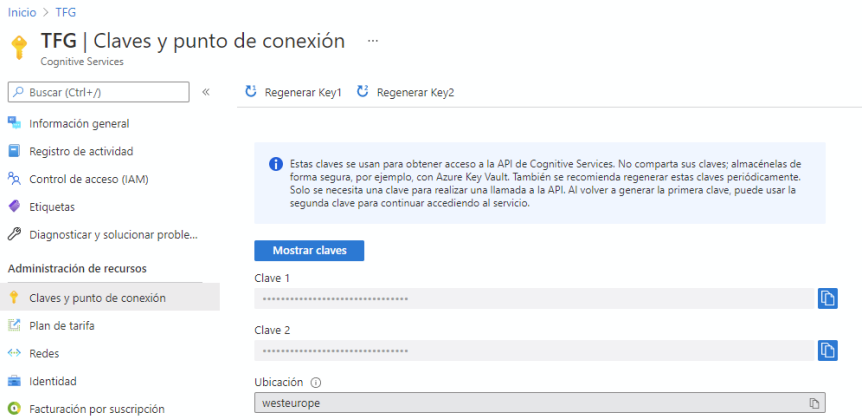


Figura 6.16 Claves de acceso del recurso de reconocimiento de voz de Azure

Una vez copiados estos datos, ahora sí ya podemos cerrar la ventana del navegador y volver a Visual Studio. Una vez de vuelta en visual Studio ya podemos empezar a "picar" el código para que el sistema reconozca nuestra voz.

Para usar el reconocimiento de voz desde visual Studio lo primero que tendremos que crear será la configuración de este a través de una variable de tipo SpeechConfig que será la que almacenará datos de nuestra configuración del reconocimiento de voz tales como los datos de la subscripción (las anteriormente nombradas claves del recurso y la ubicación de los servidores)

o el idioma del reconocimiento entre otros, que en este caso será el Castellano (español de España). Además de la configuración del reconocimiento de voz, también deberemos generar la configuración del audio y del micrófono, pero este último es más sencillo puesto que podremos decirle que use la configuración por defecto y despreocuparnos de los demás. Una vez que tenemos ambas configuraciones, el resto es tarea sencilla, puesto que ya solo deberemos crear la variable "recognizer" que será el propio reconocedor de voz y llamar a su método RecognizeOnceAsync que reconocerá de forma asíncrona la voz del usuario y no parará hasta que el usuario termine de hablar o se produzca un silencio de varios segundos. Tras este tiempo, se mandará el audio a los servidores de Azure, se procesará y se devolverá en forma de texto. Una vez que obtengamos este texto lo filtraremos un poco remplazando los números escritos con texto (básicamente uno, dos y tres, el resto los devuelve en número) por sus equivalentes símbolos matemáticos (los números en sí) y también eliminaremos caracteres problemáticos como los acentos y las tildes. Este nuevo texto ya estará listo para procesar, pero eso lo haremos en la siguiente fase, por ahora nos limitaremos a mostrarlo por pantalla en el recuadro de texto que tenemos bajo el botón de reconocer texto.

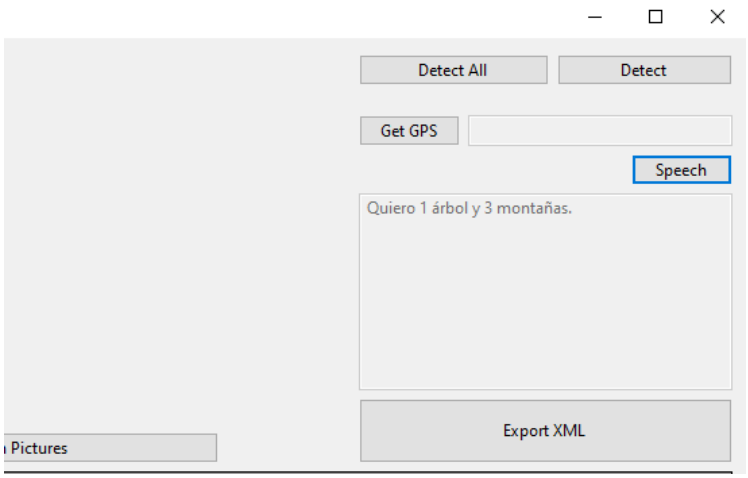


Figura 6.17 Ejemplo de reconocimiento de texto generado usando el reconocimiento de voz de Azure

Con esto ya habríamos completado todos los módulos necesarios para utilizar el módulo final, la exportación de los datos obtenidos a XML. Este módulo tiene varios subpuntos a tratar, puesto que más allá de la generación del archivo XML es el encargado de pretratar algunos de los datos antes de exportarlos.

Principalmente los datos que este módulo trata son 2, los datos generados por el reconocimiento de imágenes y principalmente los datos del reconocimiento de texto. Empezaremos por los primeros ya que son los más sencillo y servirán para introducirnos en algunos de los aspectos relevantes de los segundos. Pero antes responderemos a la pregunta ¿Por qué deben pretratarse los datos antes exportarlos? La respuesta a esa pregunta se encuentra en el concepto de sintaxis. Los ordenares de forma tradicional están acostumbrados a trabajar con datos binarios, que en mayores niveles de abstracción son número, letras, cadenas de caracteres, etc. Pero de forma predeterminada estos datos no tienen ningún sentido para el ordenador. Por ejemplo, la palabra "Coche" para los seres humanos tienen un significado claro, estoy seguro qué todo lector al leer esa palabra ha podido vislumbrar en su cabeza lo que un coche significa para él. Bien pues creásemos un String cuyo valor fuera "Coche" para el

ordenador esa cadena no tendría un significado persé. Puesto que para el ordenador solo sería una cadena de los caracteres `C`, ‘o’, ‘c’, ’h’, ’e’. que estos a su vez en codificados (para este caso supongamos que en ASCII). Sería una cadena conformada por los valores en base decimal 67 111 99 104 101, que a su vez en codificación binaría sería:

<<01000011 01101111 01100011 01101000 01100101 >>

O dicho a “Grosso modo” un conjunto de ceros y unos colocados de una manera específica para poder interpretarlos como una palabra, pero poco más. Para poder dotar al sistema de una sintaxis tenemos que crear un lenguaje que el ordenador pueda entender. Cierto es que este podría ser un lenguaje complejo como el castellano o el inglés, pero esto sería mucho más dificil de procesar para un ordenador que por ejemplo un metalenguaje como XML. Y esto es precisamente lo que vamos a hacer, vamos a “traducir” toda la información que el usuario ha generado con la herramienta, a un metalenguaje con el que luego sea fácil trabajar y con el que podamos decirle al ordenador que cada vez que detecte la cadena “Coche” o más bien la cadena binaria expresada antes, bajo ciertas circunstancias, lo que esperamos de el es que cargue al escenario un modelo 3D que vincularemos con dicha cadena. De esta manera estaremos creando un modelo de caja blanca que procese la información dada por el usuario en un metalenguaje más sencillo que luego el generador pueda leer e interpretar con muchísima más facilidad que leyendo la información en bruto, de diferentes fuentes, y con diferentes sintaxis que supondría importar directamente todos los datos generados por el usuario sin pretratar.

Una vez hecha respondida la pregunta de por qué es necesario este proceso, ya podemos entrar de lleno a explicar cómo se ha diseñado este sistema. En primer lugar, se ha diseñado un archivo que funciona como diccionario o traductor para traducir las palabras de un texto, en castellano por ejemplo a las palabras que usaremos para el metalenguaje que comentamos anteriormente. En la estructura profunda de este archivo entraremos más adelante, concretamente en el capítulo 6.2.3 cuando hablemos de la base del conocimiento y los sistemas que la componen, pero por ahora lo relevante es saber que las palabras de nuestro sistema las clasificaremos gracias al archivo de traducción en 3 categorías, nombres, adjetivos y números. Para obtener las palabras que pertenecen a esta categoría, se hace uso de los métodos de la clase XML_Manager cuya estructura puede verse a continuación:

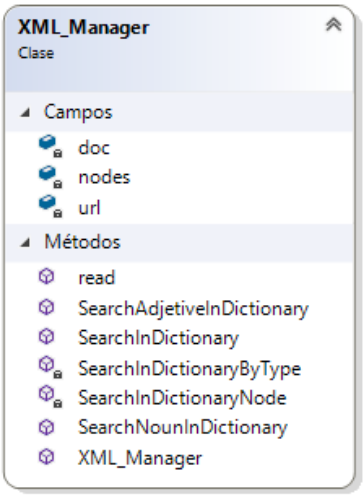


Figura 6.18 Clase XML_Manager

Concretamente se usan los métodos de búsqueda "SearchNounInDictionay" y "SearchAdgetiveInDictionary" que, como se puede suponer, son casos específicos de la función "SearchInDictionayByType" que a su vez implementa SearchInDictionaryNode, que busca en el archivo XML objetos que contengan un nombre dado. Gracias a estos métodos, podemos reconocer palabras y buscar las "traducciones" a otras palabras generadas por el sistema (como es en este caso, el de las palabras generadas por el reconocimiento de imágenes) o generadas por el usuario (para el reconocimiento de voz).

En el caso que nos acomete, los datos extraídos por el algoritmo de reconocimiento de imágenes, se hace uso del método "SearchNounInDictionay" para convertir los objetos detectados por la red neuronal (su tipo) en objetos entendibles más adelante por el generador mediante una simple conversión antes de exportarlo.

A continuación, vendría el tratamiento de los datos generados por el reconocimiento de voz, este es más interesante que el anterior puesto que ahora ya no es solo cambiar una palabra por otra, si no que tendremos que enseñar al sistema a interpretar el contenido de las frases.

Para ello el algoritmo que se ha seguido es el siguiente:

- **Paso 1.** Eliminar datos que obstaculicen la lectura de las palabras como los puntos o las comas.
- **Paso 2**. Separar el texto en una lista numerada de palabras
- **Paso 3.** Se recorre la lista y se clasifican los datos en nombres, adjetivos o números y se guardan sus posiciones en el array de palabras en sus respectivos arrays.

Hasta aquí la parte fácil, ahora se complica un poco más

- Paso 4. Se genera una lista de Objetos (WordNodes) que guardarán los datos de los objetos del texto
- **Paso 5.** Se recorren los nombres, y se crean objetos (WordNodes) para esos nombres que se añaden a esa lista. A excepción de unos en concreto, los pronombres, que el traductor ha debido traducir como "[LastName]" y por tanto cuando aparezcan el sistema debe de poner crear un objeto nuevo usando el nombre anterior al pronombre. Excepto en 2 casos, primero que el pronombre aparezca en el texto antes que cualquier nombre (en cuyo caso se crea un objeto con el nombre "???") o que la siguiente palabra sea un nombre, en cuyo caso no se habría añadido en el paso 3.
- **Paso 6.** Se recorren las listas de adjetivos y números anteriormente generadas y se busca el sustantivo que se encuentre a menor distancia (en número de palabras) del adjetivo o número a través de la función NumeroMasCercano(). En el caso de los números, estos se añaden a una variable propia llamada "number", los adjetivos en cambio se añaden a una lista llamada "Atributes" en ambos casos en el objeto WordNode al que pertenezca el sustantivo.
- **Paso 7.** El método devuelve la lista de objetos.

Estos serían los pasos seguidos para procesar el texto dictado por el usuario. Respecto a la función número más cercano, se ha diseñado como una función más genérica por si se necesitase reutilizar más adelante por lo que a partir de una lista valores numéricos ordenados y un número calcula cuál de los números de la lista es que está más próximo a este número y devuelve su posición en la lista. El código de este método puede verse a continuación:

```
2 referencias
static int NumeroMasCercano(List<int> lista, int numero)
{
    if (lista.Count == 0) Debug.WriteLine("Error N02: la lista tiene longitud 0");
    if (numero < lista[0]) return 0;
    if (numero > lista[lista.Count-1]) return lista.Count - 1;

    for (int i = 0; i <= lista.Count - 2; i++)
    {
        //Debug.WriteLine("number: " + numero + " list: " + lista[i]);
        if (numero < lista[i]) continue;
        if (lista[i + 1] - numero < numero - lista[i])
            return i + 1;
        else return i;
    }
    Debug.WriteLine("Error N01: el número buscado se encuentra fuera de los límites: " + numero);
    return -1;
}
```

Listado 6.4 Código de la función NumeroMásCercano

Con esto terminaríamos la parte de pretratamiento de los datos y ya podríamos pasar a la parte de la exportación propiamente dicha. En este caso el archivo se escribirá como un texto procedural por lo que no hará falta usar funciones especializadas. Solamente una lista de Strings que serán las diferentes líneas del archivo y el método File.WriteAllLines() al que deberemos pasarle como parámetros la lista anterior y la dirección en la que guardaremos el archivo y su nombre, estos últimos datos los obtenemos a partir de la clase SaveFileDialog que le mostrará al usuario la interfaz del explorado de Windows (Figura 6.19) para que pueda elegir donde guardar el archivo y el nombre que quiere darle.

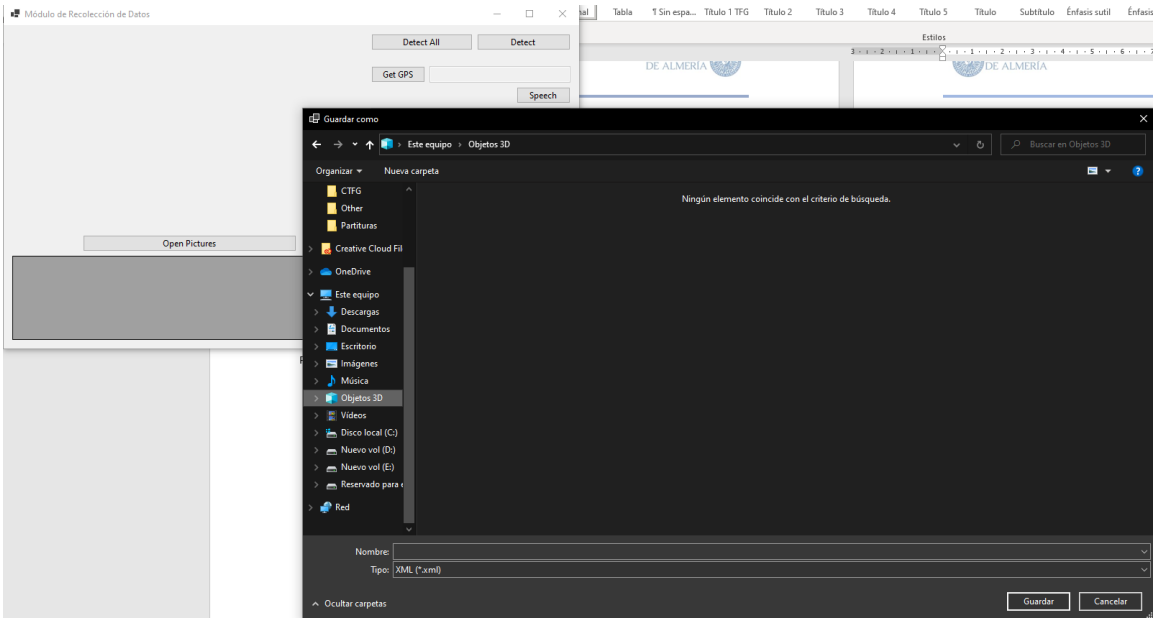


Figura 6.19 Interfaz del explorador de Windows usada para guardar el fichero XML

Esta sería la parte del guardado del archivo, pero hablemos ahora del formato, el archivo XML esta formado por un nodo inicial llamado UserDataFile en el que se guardan dentro todos los datos. El siguiente listado (Listado 6.5) muestra el formato de guardado de este archivo XML.

prueba.xml ● prueba2.xml ×

C: > Users > diego > Desktop > prueba2.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<UserDataFile>
    <originalText text = "Quiero 3 casas con 1 arbol rojo y 2 coches grandes."></originalText>
    <pos lat = "36.8381" long = "-2.4597"></pos>
    <element name = "Casa" number = "3" atributes = ""></element>
    <element name = "Arbol" number = "1" atributes = "#FF0000"></element>
    <element name = "Coche" number = "2" atributes = "Big"></element>
    <imageItem name = "Coche" color = "#3F6083"></imageItem>
</UserDataFile>
```

Listado 6.5 Formato del archivo XML de salida del extractor de datos

Como se puede ver en el listado anterior, se guardan 4 tipos de datos diferentes:

- **originalText**, es el texto originar dictado por el usuario, no se usa, pero se considera interesante guardarlo para poder comparar, el texto de origen y el resultado y por si en el futuro se quiere hacer uso de él.
- **pos**, almacena la geolocalización del usuario, lat hace referencia a la latitud y long a la longitud.
- **element**, almacena cada uno de los objetos detectados en el reconocimiento de voz, almacena su nombre, la cantidad y los atributos (adjetivos) del objeto, como el tamaño o el color.
- **imageItem**, guarda el resultado del reconocimiento de imágenes, es decir los estilos para los distintos objetos, cuando a un objeto no se le ha dado un color a través de la voz, se utilizan estos estilos para generar el color del objeto en el generador

Estos son todos los elementos que el sistema almacena de los extraído del usuario y es lo que luego se usará en el generador para personalizar la experiencia de este además de saber lo que desea generar. Con esto queda concluida esta parte referente a la obtención de los datos del usuario y su transformación a un metalenguaje que pueda entender la siguiente parte.

Finalmente, como último añadido a esta sección, a continuación, se puede ver el conjunto de clases que conforman este sistema junto con sus respectivas herencias (Figura 6.20):

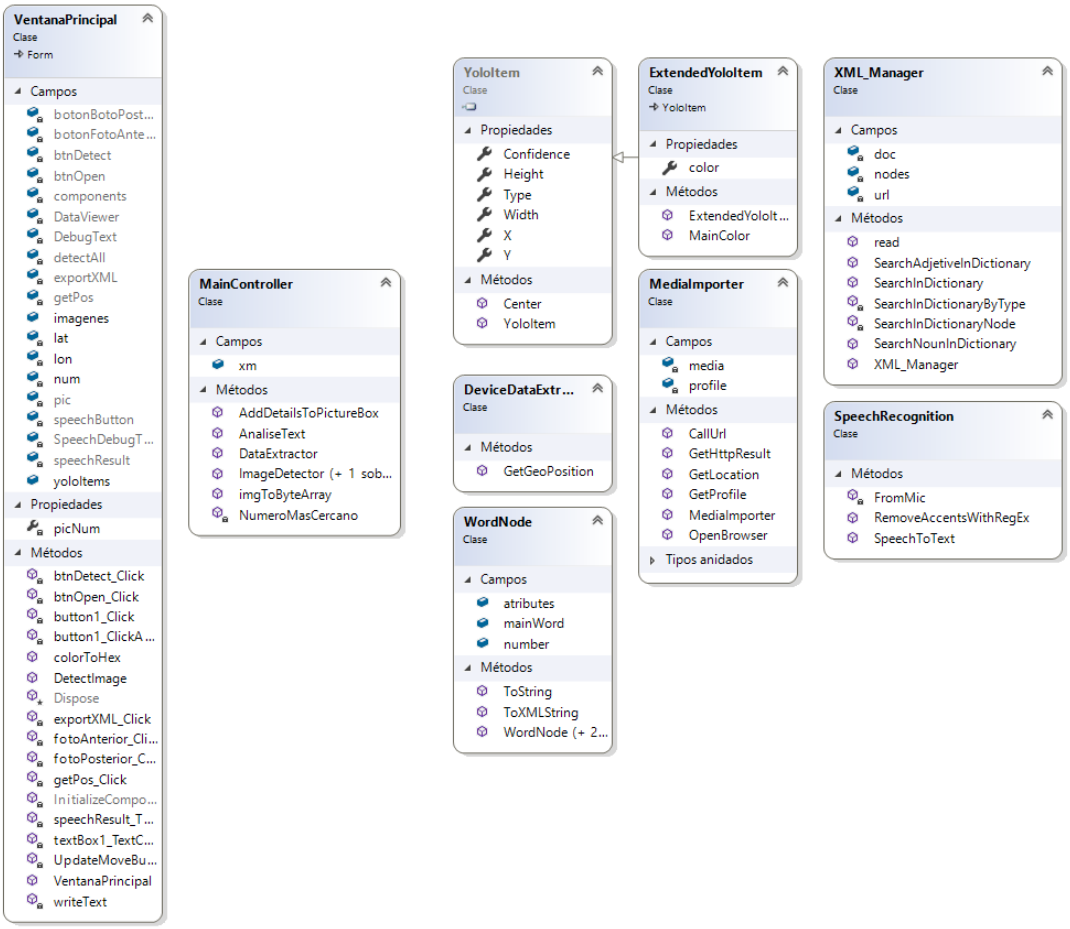


Figura 6. 20 conjunto de clases que componen este módulo

6.2.2 Generador de entorno 3D

Una vez terminada la parte de recolección de datos, nos toca la parte de usar estos datos para generar los escenarios, por lo que ahora saltaremos de herramienta y entraremos de lleno en la parte de desarrollo en Unity3D.

Esta parte se ha diseñado para contar con una clase central, “TerrainGenerator” que será la encargada de interactuar con el usuario y de controlar el resto del sistema, esto quiere decir que será esta clase la que se encargue de todas las cargas de archivos, así como de la generación de objetos. Dado que la generación del entorno se ejecuta por fases y esta clase es sumamente extensa, la dividiremos en varios puntos a partir de ahora. El primero que detallaremos será la parte de importación de archivos, puesto que esta debe ser previa a la generación.

Esta parte hace uso de dos archivos diferentes, la base de datos de objetos, y la salida del anterior módulo, en ambos casos, se ha generado una clase para cada una de ellas, con sus respectivas estructuras de datos y operaciones.

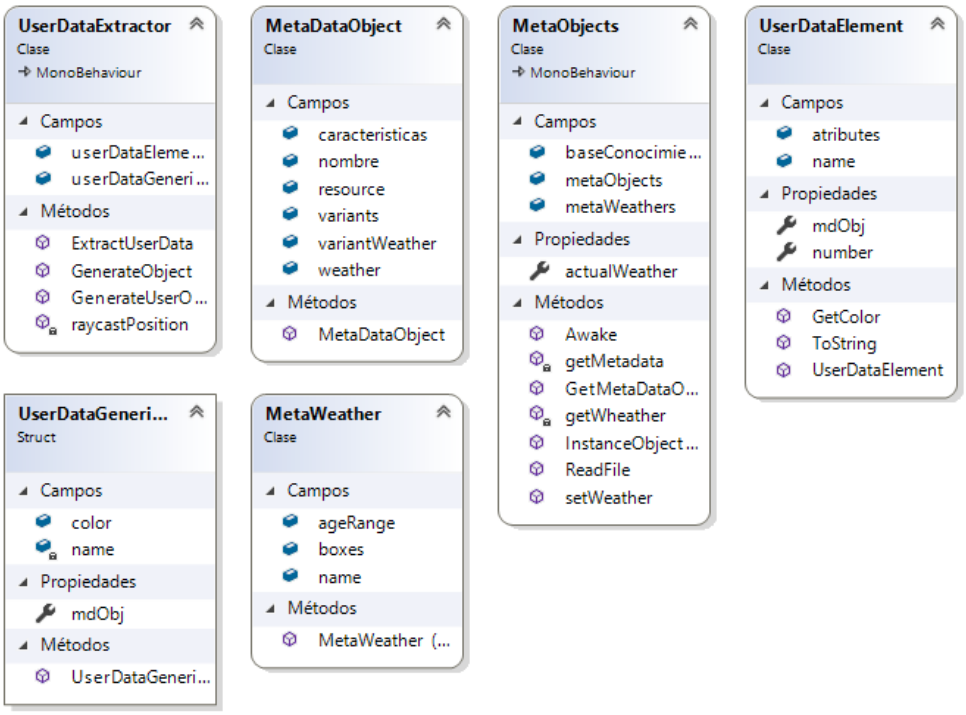


Figura 6.21 Clases para la importación de datos

La figura anterior (6.21) muestra estas dos clases maestras, "UserDataExtractor y MetaObjects", para guardar los datos se han usado 3 clases y dos estructuras. MetaDataObject, que guarda un objeto de la base de datos de objetos, MetaWeather que hace lo propio, pero con los distintos climas, Structures que se encarga de generar las estructuras (objetos compuestos por otros objetos) y, por otro lado, desde el archivo de las preferencias de usuario están, UserDataElement que guarda un objeto dicho por el usuario y UserDataGenericElement que guarda los datos (color) de un objeto reconocido por el reconocimiento de imágenes.

Como ambos archivos son "XML", se hace uso de la clase XMLMagement para ectraer sus datos, a través de los nodos XML. Un ejemplo de esto puede verse en el siguiente fragmento de código (Listado 6.6):

```
System.Globalization.CultureInfo.CurrentCulture = new System.Globalization.CultureInfo("en-US");
XmlDocument doc = new XmlDocument();
doc.Load(path);

XmlNodeList nodes = doc.DocumentElement.SelectNodes("/UserDataFile/element");
foreach (XmlNode element in nodes)
{
    string name = element.Attributes["name"].Value;
    int number = 1;
    try
    {
        number = int.Parse(element.Attributes["number"].Value);
    }
    catch (Exception)
    {
        Debug.LogError("Error: El archivo de datos de usuario contiene un valor number no numérico, se tomará el valor 1");
    }
    UserDataElement ude = new UserDataElement(name, number);
    if (element.Attributes["atributes"].Value.Length > 0)
    {
        print(element.Attributes["atributes"].Value);
        foreach (string s in element.Attributes["atributes"].Value.Split(','))
        {
            print(s);
```

Listado 6.6 Fragmento del método ExtractUserData

El anterior listado comprende un fragmento del código usado para extraer la información de los datos del usuario. Como se puede ver se carga el archivo XML (a través de un Path) y se accede a sus nodos para extraer el contenido de sus atributos y de sus nodos hijos. El proceso para este y el resto de los elementos de los dos archivos se lleva a cabo de la misma manera, pero cambiando los parámetros a los que se accede o las estructuras de datos que se rellenan.

Una vez terminada la parte de la importación, ya tenemos a nuestro alcance todos los recursos que necesitamos para trabajar, por lo que ya podemos empezar con la programación del sistema. Lo primero que se desarrollará será la parte de la modificación del terreno en Unity.

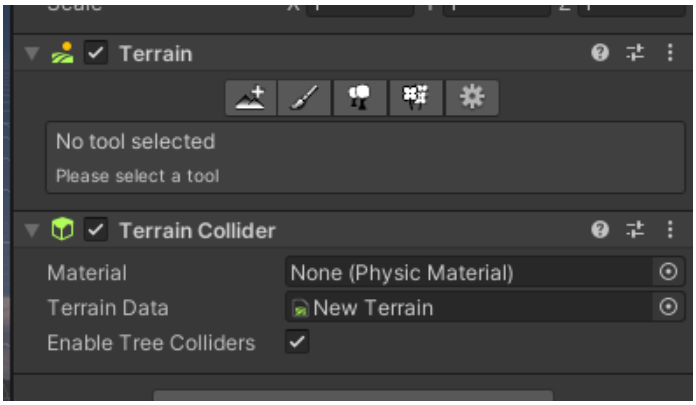


Figura 6.22 Componente terreno en Unity3D

El terreno en unity está controlado por el componente "Terrain" (figura 6.22) de este componente nos interesan saber 3 cosas, las dimensiones del terreno (principalmente la altura máxima), el tamaño de la matriz de alturas y la propia matriz de alturas que es del tipo "float[,]" es decir un array bidimensional de número decimales de 0 a 1. Para modificar esta matriz introduciremos el concepto de los "shapes" o formas de superficie, esto son matrices más pequeñas con valores de 0 a 1 que se pueden sumar, restar, multiplicar o dividir respecto a la matriz del terreno anterior. Pero, hacer que la persona introduzca los valores de estas matrices manualmente, sería como pedirle a un programador que escribiera el código de un programa en binario, algo largo y tedioso, es por eso por lo que en vez de introducir una matriz haremos uso de una imagen en blanco y negro, también conocido como mapa de alturas, donde la claridad u obscuridad de los pixeles representarán la altura del punto, siendo blanco, 1 y negro, 0.

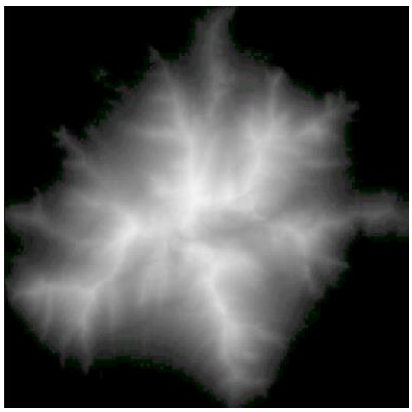


Figura 6.23 mapa de altura de una montaña

La figura anterior (6.23) representa el mapa de altura de una montaña donde los puntos más blancos (el centro) son a su vez los puntos más altos de la montaña, mientras que los más oscuros son los más bajos.

Para guardar estos mapas de altura usaremos una estructura personalizada llamada "TerrainTopology" (Listado 6.7)

```
public struct TerrainTopology {
    public string name;
    public Texture2D topology;
    public bool inverseDirection;
}
```

Listado 6.7 Estructura "TerrainTopology"

Esta estructura almacenará el nombre de la superficie que queremos generar, su mapa de alturas y una variable booleana (verdadero o falso) sobre si el mapa de altura es inverso, es decir, las alturas en realidad son número negativos y la superficie debería generarse hacia abajo y no hacia arriba (por ejemplo, un lago). Una vez creada esta estructura, guardaremos una lista de estas donde el desarrollador podrá ir añadiendo las distintas superficies que desee. Además, en la clase "TerrainGeneratorEditor" que es una extensión de la clase TerrainGenerator para la personalización de la interfaz gráfica de Unity, añadiremos una lista dinámica (esto es que se actualiza cada vez que se hace un cambio en la clase) con los nombres de los elementos de esta lista de manera que así el desarrollador pueda elegir que topología quiere aplicar en cualquier momento.

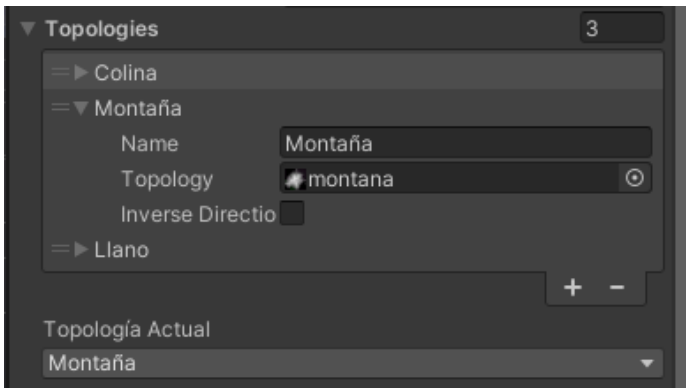


Figura 6.24 Gestión de topologías del terreno

La figura anterior (6.24) muestra el resultado gráfico de lo anteriormente diseñado. A continuación, el último paso que nos queda de esta parte es la de calcular las alturas del terreno. Para ello simplemente crearemos una función que recorra píxel a píxel la imagen y dándole una posición de inicio (X, Y) sume (o reste) al terreno los valores de la superficie en ese punto. Además sumaremos una altura fija al terreno (60m) para poder crear superficies cóncavas luego, como ríos o lagos. Tras hacer eso, y ejecutar el método, este (Figura 6.25) sería uno de los posibles resultados que podríamos obtener (6.25):

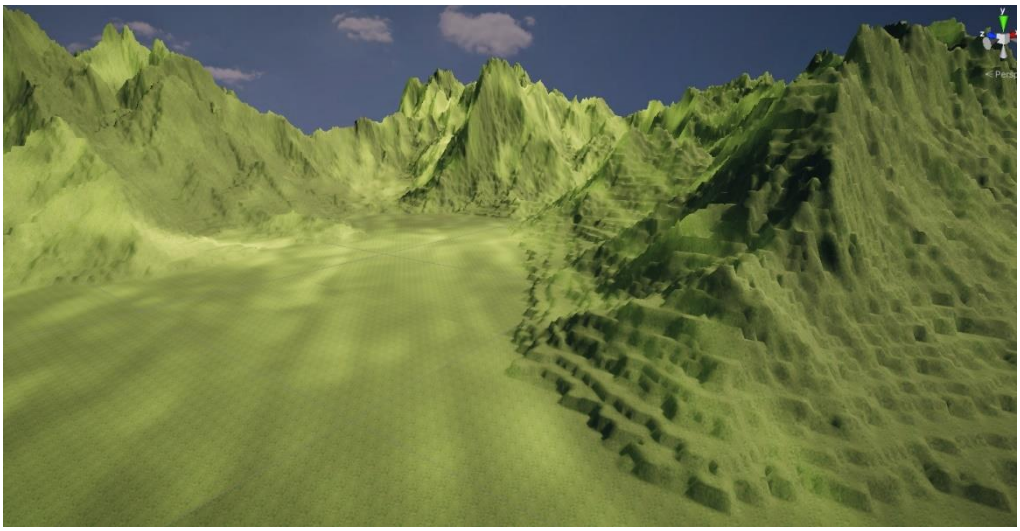


Figura 6.25 Resultado de un terreno generado automáticamente.

El problema, aquí residiría en que al utilizar superficies fijas (como la montaña) e ir repitiéndola, aunque se puede obtener cierta sensación de variedad, al final hay muchas partes del terreno que pueden verse excesivamente similares. Por lo que, para solucionar este punto, se hará uso de una función que le otorgue cierta variación casi al azar a la superficie del terreno, para así eliminar o al menos reducir este efecto de similitud (Esta opción es opcional y se puede desactivar desde la interfaz de Unity3D). La siguiente figura (6.26) muestra como sería el escenario tras aplicar este efecto:

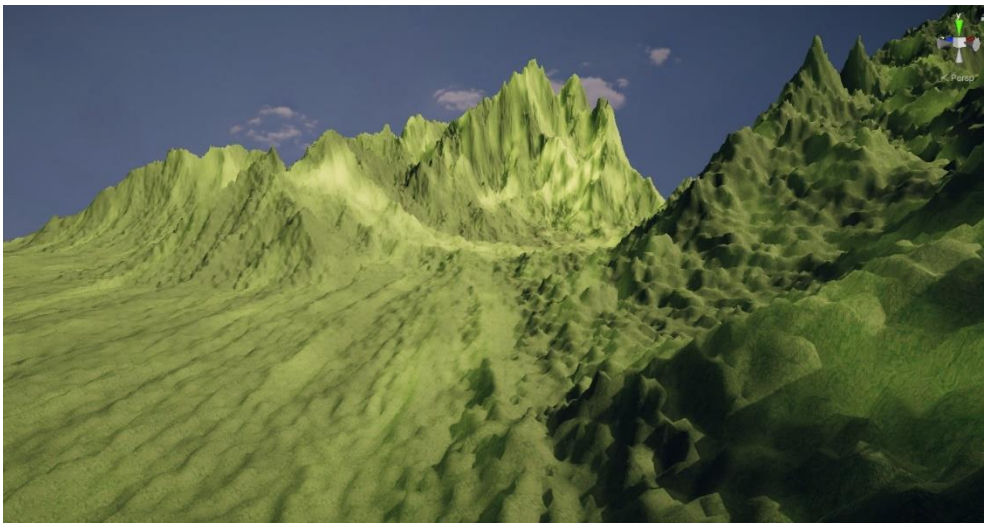


Figura 6.26 Resultado del terreno tras añadir ruido aleatorio

Como se puede ver en ambas figuras anteriores, el terreno de ambas figuras se ve como estuviera compuesto de cubos, esto es debido a que la textura usada tiene un número limitado de tonos de grises, lo que acaba conllevando que ocurra este efecto, para arreglarlo, lo que se deberá aplicar a continuación es un filtrado o suavizado que reduzca estos picos. El siguiente listado (6.8) muestra el código utilizado para hacer este suavizado:

```
1 referencia
public float GetNeighbourValuesMean(float[,] f, int i, int j) {
    List<float> f2 = new List<float>();
    int[] lista = new int[] { i - 1, j, i, j - 1, i - 1, j - 1, i + 1, j, i, j + 1, i + 1, j + 1, i - 1, j + 1, i + 1, j - 1 };
    for (int index = 0; index < lista.Length - 1; index += 2)
    {
        try { f2.Add(f[lista[index], lista[index + 1]]); } catch (System.IndexOutOfRangeException) { continue; };
    }

    return mean(f2);
}
1 referencia
public void SmoothTerrainMatrix(float[,] f, int times = 1) {
    for (int k = 0; k < times; k++)
    {
        for (int i = 0; i < f.GetLength(0); i++)
            for (int j = 0; j < f.GetLength(1); j++)
            {
                f[i, j] = (GetNeighbourValuesMean(f,i,j) + f[i, j]) / 2;
            }
    }
}
```

Listado 6.8 Código para suavizar el terreno

Como se puede ver en la anterior figura, el código se compone de 2 funciones, GetNeighbourValuesMean que obtiene la altura media de todos los puntos colindantes a uno dado. Por otro lado, SmoothTerrainMatrix hace uso de este valor para calcular la media con el punto actual y usar este valor como la nueva altura del punto.

La siguiente figura (6.27) muestra el resultado tras aplicar este filtro:

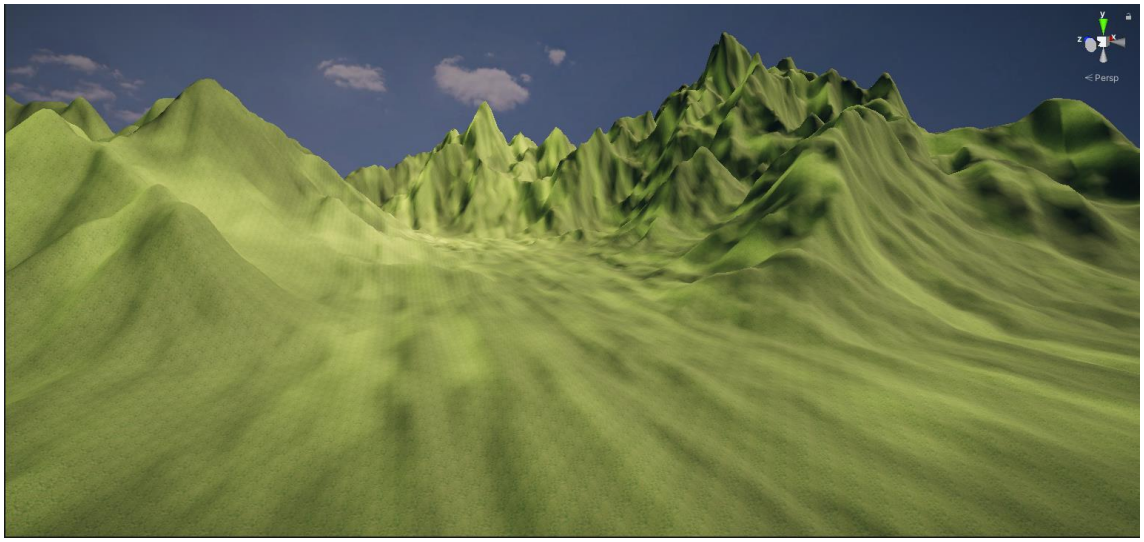


Figura 6.27 Resultado del terreno tras el filtrado

Con esto ya tendríamos lista la superficie del terreno que posteriormente usaremos como base para colocar los objetos y demás añadidos.

Una vez que tenemos la superficie del terreno generada, es hora de añadir algunos cambios, en este caso hablamos de los ríos. Los ríos son un tipo de objeto especial pues están especialmente afectados por las físicas (el agua siempre busca el mejor camino para moverse) y además tienen la capacidad de modificar el terreno, por lo que requieren de un tratamiento especial. Pero antes de implementar los ríos deberíamos de pensar en cuando deberemos utilizarlos, me explico, los rios diseñados para este proyecto son ríos que se mueven por gravedad desde un punto dado, por tanto los grandes ríos sobre terrenos llanos no están comprendidos aquí puesto que el sistema no hace uso de fuerzas iniciales (cuando el escenario genera solo una sección de un rio,

el agua de este entra al mapa con un empuje que la empuja hacia delante y eso no está integrado aquí), por lo que partir de un lugar elevando o con al menos algo de inclinación es un requisito indispensable aquí. Es por ello por lo que primero deberemos calcular si el terreno es apto para contender algún rio, para calcular esto mediremos la diferencia de distancia entre el punto más alto y el más bajo del mapa y a partir de ahí decidiremos. No es el método más certero, quizás calcular la diferencia de altura entre todos los puntos sea un mejor método (también está implementado, pero no se utiliza), pero a su vez consume bastantes más recursos. Por lo que considero que el primero es un mejor método heurísticos en cuestión de resultado / coste de recursos. Esto nos da como resultado la siguiente fórmula para calcular el número de ríos a generar:

$$\big(TerrainMax(h) - TerrainMin(h)\big) * RIVER\_MULTIPLICATOR$$

Como se comentó anteriormente, la altura de un terreno dentro de su matriz de altura es un número que va de 0 a 1, por lo que el resultado de restar a la altura máxima la mínima, siempre dará un número entre 0 y 1. Finalmente este número se multiplicado por una constante, RIVER_MULTIPLICATOR cuyo valor por defecto es 4, este será el número total de ríos que generaremos.

Ahora que ya hemos respondido a la pregunta ¿Cuántos? Nos falta responder a la pregunta ¿Dónde? Para resolver en esto haremos uso de una función “getPercentHeight” que ordenará los puntos del terreno según su altura y luego tomará el punto que se encuentre en el percentil indicado, en este caso uno al azar entre el 85 y el 92.

Ahora que ya sabemos cuántos ríos queremos y donde vamos a colocarlos, es hora de programar los propios ríos, esto se hace básicamente desde la clase “River”, (figura 6.28) creada específicamente para esta función:

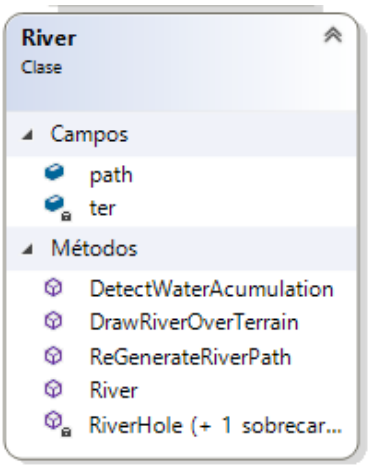


Figura 6.28 Clase “River” para la generación de ríos

Como puede verse en la figura anterior, la clase “River” cuenta con 2 variables y 5 métodos distintos. A continuación, detallaremos estos datos:

Variables

- **path:** la lista de vectores que contienen el “recorrido” del río
- **ter:** es el terreno sobre el que se va a generar el río

Métodos

- **River:** Es el constructor de la clase, asigna el terreno y llama a la función ReGeneratRiverPath para generar los valores de la variable path, es decir la lista de vectores que conformarán el río.
- **ReGenerateRiverPath**: El el método encargado de generar los puntos por los que pasará el río. Para calcular estos puntos se sigue el siguiente procedimiento:
    - **1.** Se proyecta el punto inicial (P0) hacia el suelo hasta colisionar con el terreno y se crea una variable fuerza a 0.
    - **2.** A partir de este nuevo punto (primer círculo rojo de la figura 5.29), al vector fuerza se le suma el vector normal del terreno en ese punto.
    - **3.** Se añade ese punto a la lista de la solución
    - **4.** Desde ese punto se crea un nuevo punto en la dirección de la fuerza y a una distancia equivalente a la definida por el método (distBeetweenPoint) por defecto 1.
    - **5.** Se repite el proceso desde el paso 1 hasta que ocurra una de las 3 siguientes condiciones.
        - Condición 1. Se han alcanzado todos los puntos solicitados para el río
        - Condición 2. Haya al menos 10 puntos generados y el vector fuerza sea menor a distBeetweenPoint / 10
        - Condición 3: Exista al menos un punto y el vector fuerza, y el vector fuerza del punto anterior tengan un ángulo comprendido entre 165 y 195 (el río está girando 18 grados y volviendo sobre sí mismo).

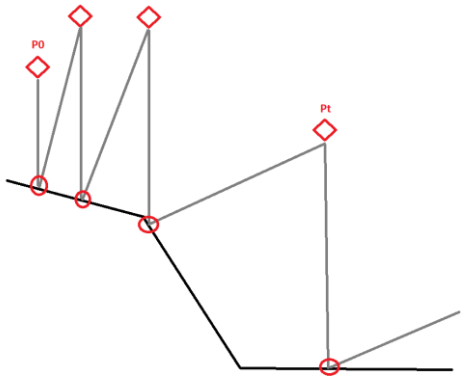


Figura 6.29 Funcionamiento de la generación de ríos

La figura 2.29 representa una simulación del cálculo de las posiciones del rio, siendo los círculos redondos los puntos resultantes y los robos los puntos generados a partir de un punto más el vector fuerza, finalmente las líneas grises, representa el desplazamiento que va haciendo el sistema para calcular los puntos redondos.

- **DetectWaterAcumulation:** Esta función detecta las acumulaciones de agua (los lugares donde hay acumulados una gran cantidad de los puntos del río), esta funcionalidad se diseño originalmente para producir lagos y estructuras similares, pero, aunque es funcional, no es utilizada por la versión actual del software. Básicamente recorre la lista de vértices dos veces y calcula la cantidad de puntos cercanos a un determinado punto para luego devolver este elemento, también calcula el tamaño del lago en función de

cuantos puntos se ubiquen cerca, a mayor numero de puntos cercanos, mayor será el tamaño del lago resultante.

- **DrawRiverOverTerrain:** Este método junto con el siguiente son los encargados de dibujar el rio sobre el terreno y de modificar este último para que se adapte a la forma del río. Para hacer esto, se deben hacer dos ejecuciones al mismo tiempo, por un lado, hay que generar primero la maya que compondrá la superficie del rio, y por otro, se debe de crear una hendidura en el terreno, esta última parte se cubre en el siguiente método, por lo que aquí nos centraremos en la generación de la maya del rio. Para generar esta maya necesitamos definir, la posición, el vector de la arista y el ancho de la arista para cada punto. El primer dato ya lo tenemos, el segundo lo podemos calcular con la siguiente fórmula:

$$VNormal = (path[i+1] - path[i])\ X\ Vector3.Up$$

Como muestra la formula anterior, la dirección del camino lo calcularemos obteniendo el vector Path[i]Path[i+1] donde Path[i] es el punto actual que estamos generando y path[i+1] el siguiente de la lista, después multiplicamos el vector resultante por el Vector3.Up (x,y,z: 0,1,0) y de esta manera obtendremos el vector perpendicular a la vertical y vector del movimiento. Finalmente, los rios no suelen tener un ancho fijo a lo largo de toda su vida, por lo que el ancho del río también tendremos que calcularlo, esta es su fórmula:

$$temporalSize = size * 0.9 * \sin\frac{i * \frac{\pi}{2}}{pathLength} + size * 0.1$$

Donde size es el tamaño máximo que alcanzará el rio, i el índice del punto actual en la lista de puntos, y path.length la longitud de dicha lista. (la función seno en Unity opera con radiales, no con grados, por i se multiplica por phi/2 y no por 360). La formula dará siempre como resultado un valor entre Size/10 y Size y la forma que tendrá el ancho del río sigue una forma sinuidal, que da unos mejores resultados que una lineal.

- **RiverHole:** Finalmente la función RiverHole es la que se encarga de generar la hendidura en el terreno por donde pasa el río. Para ello como parámetros recibe de la función anterior, la matriz del terreno, la posición de este a modificar, el tamaño del punto a modificar y finalmente nueva altura que se deberá fijar. El siguiente listado (Listado 6.9) muestra el funcionamiento de este método:

```
Vector2 origin = new Vector2(x, y);
float distance = 0;
for (int i = Mathf.Max(0, x - dist); i < Mathf.Min(f.GetLength(0), x + dist); i++)
    for (int j = Mathf.Max(0, y - dist); j < Mathf.Min(f.GetLength(1), y + dist); j++)
    {
        distance = Vector2.Distance(new Vector2(i, j), origin);
        if (distance > dist) continue;
        f[i, j] = newPos + (newPos - f[i, j]) * (distance / dist);
    }
```

Listado 6.9 Método para generar la hendidura de los ríos sobre el terreno

Esto sería todo lo referente a los ríos, por lo que ahora procederemos a pasar al siguiente elemento de la lista, que ahora sí es la generación de los objetos generados por el usuario. Esto se hace a través del método GenerateUserObjects que recibe por parámetro un vector de posición que actuará como centro del punto de generación. A partir de este punto, los objetos

se generarán en un radio de 30 metros. Para generar los objetos se hace unos del método InstanceObject de la clase MetaObjects pero para poder usar esta clase primero deberemos obtener el MetaDataObject que corresponde, esto es bastante fácil, puesto que este objeto ya lo tenemos en una variable de la estructura de datos de "UserDataElement" (se generó en el constructor de esta última clase).

El método InstanceObject irá comprobando que el objeto se puede generar en el punto asignado ya que no hay impedimentos para ello, primero proyectará desde la posición especificada un rayo vertical hacia abajo que choque con el terreno para obtener el punto exacto donde el objeto debería generarse, luego comprueba que este punto cumpla con todas las restricciones del objeto, altura máxima, tamaño máximo, etc., en caso afirmativo genera el objeto. Una vez generado, falta hacer las comprobaciones propias del usuario, esto es, cambiar el color por un lado (ver listado 6.10) y aplicar las propiedades de tamaño definidas por el usuario al objeto, sobre esta última cuestión, si el usuario desea un objeto "grande" se cambia el tamaño del objeto por su tamaño máximo, si desea uno "pequeño" se cambia por su tamaño mínimo multiplicado por 1.2 y si lo desea muy pequeño se cambia por su tamaño mínimo, en los 3 casos, el tamaño mínimo viene definido por el archivo de la base de datos (ver listado 6.11).

```
public bool GetColor(out Color co)
{
    foreach (string at in atributes)
        if (at.StartsWith("#") && at.Trim().Length == 7)
        {
            co = Common.HexToColor(at.Replace('#', ' ').Trim());
            return true;
        }
    List<Color> c = new List<Color>();
    foreach (UserDataGenericElement udge in UserDataExtractor.userDataGenericElements)
        if (udge.mdObj.Equals(this.mdObj))
        {
            c.Add(udge.color);

        }
    if (c.Count > 0) {
        co = c[UnityEngine.Random.Range(0, c.Count)];
        return true;
    }
    co = Color.white;
    return false;
}
```

Listado 6.10 Código para obtener el color de un "UserDataElement"

Como se puede ver en la figura anterior, para obtener los colores, lo primero que se hace es convertir el color hexadecimal del objeto (si lo hubiera), para después devolver este color por parámetro, en caso de que este objeto no tuviera definido ningún color, este se buscaría en sus respectivos objetos genéricos (los analizados por el reconocimiento de imágenes de la fase anterior), en caso de que existieran este tipo de objetos para el objeto actual, sus colores se añadirían en una lista, para al final coger uno al azar.

```
<Object name = "arbol" resource = "CartoonTree">
  <maxHeight>60</maxHeight>
  <maxVangle>30</maxVangle>
  <!-- <lockYRotation>TRUE</lockYRotation> -->
  <maxSize>1.3</maxSize>
  <minSize>0.9</minSize>
  <Variant name = "arbolMediterraneo" wheather ="Mediterraneo"></Variant>
  <Variant name = "arbol_Alantico" wheather ="Atlantico"></Variant>
</Object>
```

Listado 6.11 restricciones y propiedades de un objeto en la base de datos de objetos

Antes de terminar con la generación de objetos me gustaría mencionar como se recuperan los objetos MetaDataObject, estos se almacenan en un diccionario en su respectiva clase, donde la clave es el nombre del objeto y el valor del diccionario es el propio objeto. Finalmente, para concluir esta parte la siguiente figura (6.30) muestra el resultado de generar varios de estos objetos a partir de un archivo de extracción de datos de usuario.

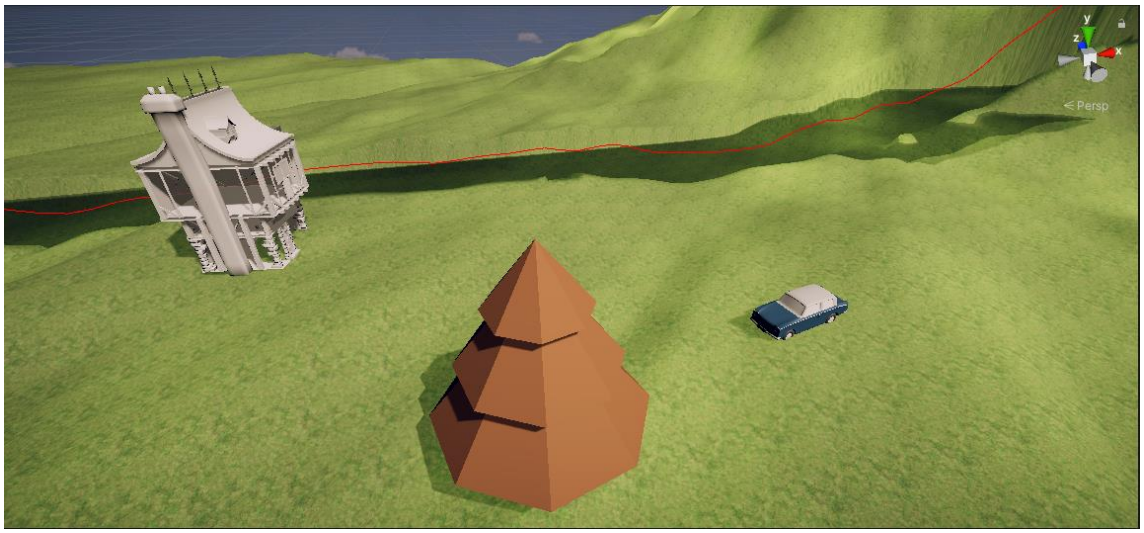


Figura 6.30 Objetos del usuario generados en el escenario de manera procedural.

El siguiente punto para generar, son los caminos, este es quizás uno de los puntos más interesante puesto que hace uso de una versión especial del algoritmo de Dijkstra para generar el camino y también hace uso de curvas de Bézier para suavizarlo posteriormente. A diferencia de los ríos donde comenzamos primero ubicándolos y luego calculándolo, en este caso empezaremos primero hablando de los algoritmos y luego pasaremos a cómo usarlos. El primer paso para obtener el camino que une dos puntos con el menos coste es definir ambos puntos, que en esta ocasión no serán los puntos reales (Vector3) si no los puntos de la matriz del terreno desde los que queremos partir y a los que queremos llegar. Para resolver este problema, empezaremos aportando una solución inicial, que posteriormente mejoraremos, que en este caso será aplicar el algoritmo de Dijkstra. Cuya implementación es la misma que encontramos en la litera de la asignatura “Estructura de datos y Algoritmos I” en segundo de carrera, solo que adaptada al lenguaje de programación C#, y tomando como función de coste entre los nodos, la diferencia de altura (pendiente) entre ambos puntos. El problema que tenemos con este algoritmo es que al estar trabajando con una matriz de 512x512, el Dijkstra debe operar con más de un cuarto de millón de nodos. Esto provoca que la ejecución sea tan lenta que puede llegar a tardar casi 15 o 20 minutos en resolverse con el ordenador actual del alumno. Por lo que para solucionar o al por lo menos hacer mas liviano este problema. Introduciremos los siguientes cambios en el algoritmo de Dijkstra:

- En primer lugar, no tiene sentido que para calcular la distancia entre dos puntos cercanos (por ejemplo, en una esquina del terreno) necesitemos hacer uso de todos los puntos de la matriz, por lo que lo primero que cambiaremos, será el tomar un subconjunto de esta del tamaño del rectángulo (por defecto 0) que conforman los dos

puntos más un margen en porcentaje a las dimensiones de este rectángulo que podrá definir el programador.

- En segundo lugar, otro de los cambios que podemos hacer es reducir la precisión del sistema, es decir, en vez de tomar como nodo cada uno de los puntos de la matriz, podemos tomar como nodo, uno de cada 3 puntos o uno de cada 10 por poner dos ejemplos, de manera que el número de puntos se reduciría en 3 y 10 veces respectivamente.
- Finalmente intentaremos obtener una aproximación de la solución aplicando el algoritmo de Dijkstra de forma anidada, aumentando en cada paso el el número de nodos con los que trabajará en algoritmo. El funcionamiento detrás de este algoritmo puede verse en la figura que sigue (6.31):

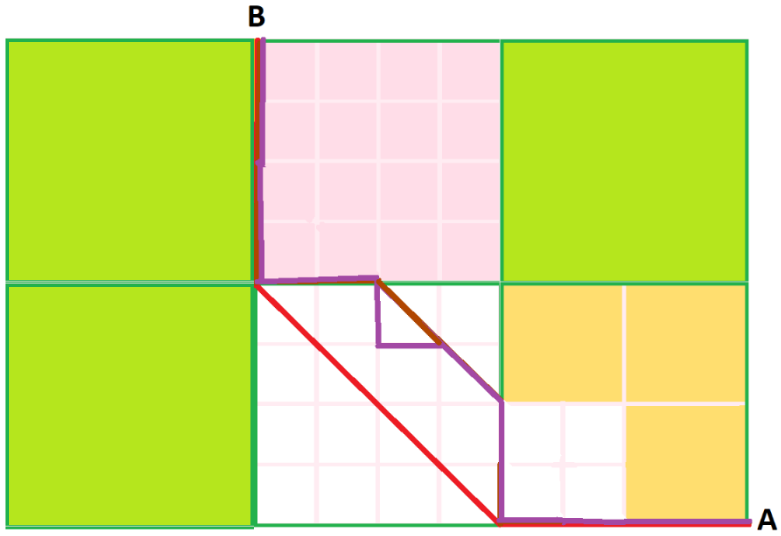


Figura 6.31 Esquema funcionamiento Dijkstra Anidado

La figura anterior es un esquema del funcionamiento del algoritmo, en este caso los valores que toma el algoritmo serían como valor inicial de búsqueda 16 unidades (cuadrados verdes), valor de reducción en cada paso, 2 (2 por cada eje, por lo que en realidad serían 4), tamaño de búsqueda mínimo 1 unidad (cuadrados rosas). En este caso el sistema empieza calculando Dijkstra cogiendo como nodos, cuadrados del tamaño de los verdes, dando como solución la línea roja, una vez calculada esta línea se aumenta la profundidad hasta 4 unidades (Cuadrados amarillos) y se calcula de nuevo Dijkstra para entre los puntos del cálculo anterior, y esta operación se repite hasta alcanzar la profundidad mínima (1 unidad en este ejemplo) dando como solución la línea violeta. Puede que para este ejemplo la diferencia entre las líneas no sean muy notable puesto la reducción se va haciendo de 2 en dos, si se aumenta este valor, en unidades más grandes como 16 o 32, si se aprecia una gran diferencia entre los distintos niveles del algoritmo.

Ahora la cuestión es, ¿Merece la pena aplicar este algoritmo? Parece que los tiempos dicen que sí, a continuación (Figura 6.32) se muestra la comparativa entre los dos algoritmos, Dijkstra Mejorado (con las dos primeras mejoras) y Dijkstra Anidado (el mismo de antes pero multinivel):

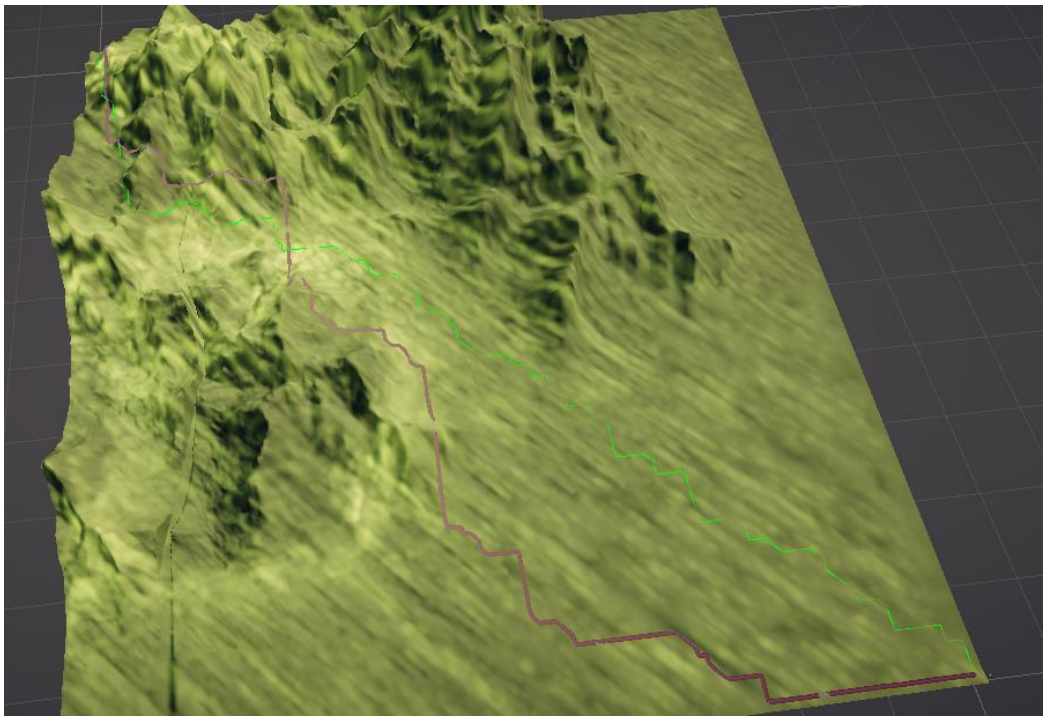


Figura 6.32 Dijkstra Mejorado (Verde) versus Dijkstra Anidado (Morado)

Para esta simulación se han hecho uso de los siguientes parámetros:

- **Dijkstra Mejorado:** Precisión 9 unidades cuadradas, tiempo en generarse **2.83 Segundos** (media)
- **Dijkstra Anidado:** Precisión inicial 81 unidades cuadradas, reducción por cada paso 3 unidades cuadradas, precisión mínima 9 unidades cuadradas. Tiempo en generarse **0.002 segundos.**

Como podemos ver, aunque los resultados de Dijkstra Anidado son inferiores, el tiempo de generación de este último es de 1400 veces inferior al primero, aún teniendo la misma precisión final. Esto significa que podríamos profundizar más en la precisión del algoritmo anidado o ampliar su zona de cálculo, teniendo incluso menos coste que el algoritmo sin anidar.

Una vez tenemos ya calculados los caminos, nos quedan dos cosas por hacer, en primer lugar, suavizar estos caminos (para que tengan un trayecto algo más suave) e integrarlos con el editor para que el desarrollador pueda definirlos.

Par lo primero haremos uso de las conocidas como curvas de Bézier, estas además como se muestran a continuación (Listado 6.10) son muy fáciles de calcular:

```
0 referencias
public Vector2 BezierCurve(Vector2 a, Vector2 b, Vector2 c, Vector2 d, float t) {
    return new Vector2(BezierCurve(a.x,b.x,c.x,d.x,t), BezierCurve(a.y, b.y, c.y, d.y,t));
}
3 referencias
public Vector3 BezierCurve(Vector3 a, Vector3 b, Vector3 c, Vector3 d, float t)
{
    return new Vector3(BezierCurve(a.x, b.x, c.x, d.x, t), BezierCurve(a.y, b.y, c.y, d.y, t),BezierCurve(a.z, b.z, c.z, d.z, t));
}
5 referencias
float BezierCurve(float a, float b, float c, float d, float t) {
    return a * Mathf.Pow((1 - t), 3) +
      b * 3 * t * Mathf.Pow((1 - t), 2) +
      c * 3 * Mathf.Pow(t, 2) * (1 - t) +
      d * Mathf.Pow(t, 3);
}
```

Listado 6.11 Implementación de las curvas de Bézier en Unity3D y C#

Las curvas de Bézier necesitan de cuatro puntos para calcularse, el origen o punto inicial, el destino o punto final y los puntos b y c que determinan la inclinación de la curva y de un valor (de 0 a 1) t que representa el punto de la curva que queremos tomar. Para este caso iremos tomando los puntos de los caminos de dos en dos, siendo el que se "salta" el que definirá la curvatura de la susodicha curva. De manera que para un punto dado se calculará:

$$Num.\,(1\ a\ 10): Dibujar\ (BezierCurve(punto\ [i-1], punto\ [i], punto\ [i], punto\ [i+1], \frac{Num}{10}))$$

La siguiente figura (6.33) muestra el resultado de aplicar curvas de Bézier a través de la formula anterior a un camino generado con Dijkstra anidado.

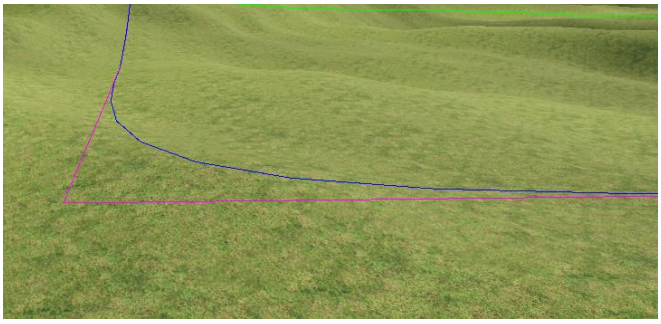


Figura 6.33 Esquina de un camino generado por Dijkstra, lía rosa, sin suavizar, línea azul tras aplicar Bézier

Ahora que ya tenemos detallado el lugar por el que pasará el camino, tan solo nos queda dibujarlo, para ello haremos uso de una función modificada muy similar a la usada para generar los ríos, solo que aquí no hará falta tamaños variables, ni muchas complicaciones, este método mencionado es el que sigue (listado 6.12):

```
public bool DibujarCamino(Terrain t, float[,] f, float size, float bezierSubidivisions = 10) {
    if (positions == null || positions.Count == 0) return false;
    MeshLineRender mr = new MeshLineRender("Path(" + positions[0] + "," + positions[positions.Count-1] + ")");
    Vector3 dir = Vector3.zero;
    Vector3 lastPos = TerrainGenerator.GetAbsolutePositionOverTerrain(t, f, positions[0]);

    for (int i = 1; i < positions.Count-2; i+=2)
    {
        for (float j = 0; j < 1; j+=1.0f/bezierSubidivisions) {
            Vector3[] realPositions = new Vector3[] { TerrainGenerator.GetAbsolutePositionOverTerrain(t, f, positions[i - 1]), TerrainGenerator.GetAbsolu
            Vector3 pos = BezierCurve(realPositions[0], realPositions[1], realPositions[1], realPositions[2], j);

            dir = pos - lastPos;
            mr.AddPoint(pos+Vector3.up, Vector3.Cross(dir, Vector3.up).normalized, size);
            lastPos = pos;
        }
        //dir = positions[i + 1] - positions[i-1];


    }
    mr.SetMeshCollider(true);
    mr.trail.material = Resources.Load("RoadMaterial") as Material;
    return true;
}
```

Listado 6.12 método para dibujar los caminos

Por motivos de espacio, la imagen no ha captado una línea completa, pero el resto de la línea es básicamente cargar las posiciones i-1, i, i e i+1 de la lista de vértices,

Finalmente, el resultado que obtenemos tras esta función es el que sigue (figura 6.34):

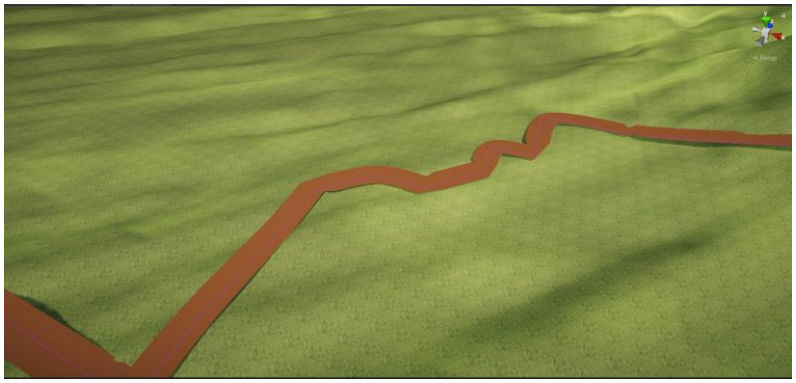


Figura 6.34 Camino generado de manera procedural mediante las técnicas detalladas anteriormente

Ahora ya tenemos los caminos diseñados y solo nos falta que sea el propio usuario el que los defina, para esto crearemos una estructura auxiliar llamada Camino (Ver figura 6.35) y una lista de estos, finalmente hacemos un método que recorra la lista y vaya generando los distintos caminos, convirtiendo previamente, las posiciones de origen y destino es posiciones 2D sobre el terreno.

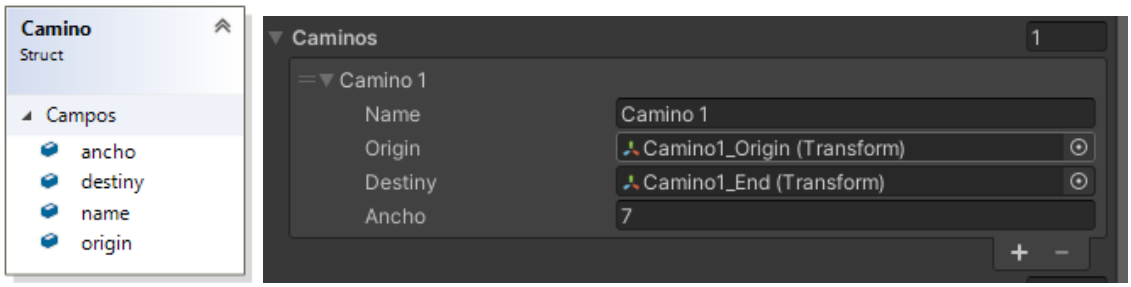


Figura 6.35 Estructura "Camíno" (izuquerda) lista de caminos en el editor de Unity (Derecha)

Como se puede ver, los datos de origen y destino de la estructura no son vectores, sino que son objetos de tipo "Transform", esto quiere decir que son objetos que se pueden mover por el escenario, lo que hace más fácil el trabajo de ubicarlos que ir calculando manualmente las posiciones que deberían tener estos vectores. Finalmente, la figura 6.36 muestra el botón para generar estos caminos junto con el vector para modificar los cálculos del algoritmo de Dijkstra.

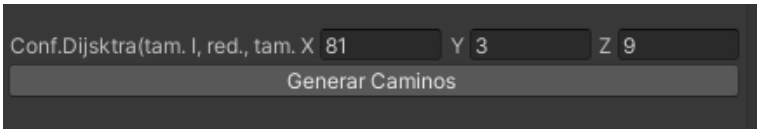


Figura 6.36 Botón para generar los caminos y parámetros de Dijkstra Anidado

Con esto daríamos por concluida la parte de caminos para adentrarnos en la última parte de este subcapítulo 6.2.3 que sería el completar el mundo. Ya que por lo general el número de elementos que va a pedir el usuario es limitado, (no importa si llega a 40 o 50 objetos, sigue siendo insuficiente para rellenar un mundo de 2500m2). Por lo que en esta ocasión haremos uso

de las estructuras, (objetos compuestos a partir de otros objetos). El siguiente listado (6.13) muestra un ejemplo de estas estructuras:

```
<Structures>
  <Structure name ="forest">
    <Object name ="arbol2"></Object>
  </Structure>
</Structures>
```

Listado 6.13 Estructuras en el archivo XML de la base de conocimiento

Esta última parte es relativamente sencilla puesto que lo unico que deberemos hacer es recorrer estas estructuras y generar tantas veces los objetos de dentro como el desarrollador haya especificado. Los métodos para todo esto ya están diseñados anteriormente (salvo los de recorrer la estructura, pero al final no es mucho más que un bucle for). Por lo que podremos saltar directamente a la parte de la interfaz, la cual también es bastante sencilla puesto que además del botón para generar los objetos, simplemente deberemos añadir un desplegable que nos permita seleccionar la estructura que queremos usar y un campo de texto donde podremos insertar cuantos elementos queremos. La figura 6.36 refleja estos aspectos de la interfaz gráfica:

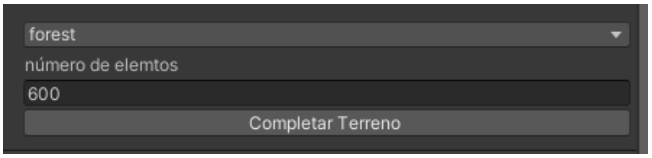


Figura 6.36 Interfaz gráfica del completado del terreno

Al igual que antes para colocar los objetos se proyecta un rayo contra el suelo que colisiona con el terreno para generar el objeto, si el rayo colisionase con otro tipo de objeto o no colisionase, no se generaría, de esta manera se evita que se mezclen objetos o que se generen sobre las carreteras, ríos u objetos definidos por el usuario. Finamente, este (figura 6.37) sería el resultado obtenido de manera procedural (sin necesidad de que el diseñador haga nada) de un escenario creado con esta herramienta:

Figura 6.37 Resultado Final de la herramienta

Como no podía ser de otra manera, finalizaremos este subcapítulo (6.2.2) mostrando en la siguiente figura (6.38) el conjunto de todas las clases, variables y métodos desarrollados para esta parte del proyecto:

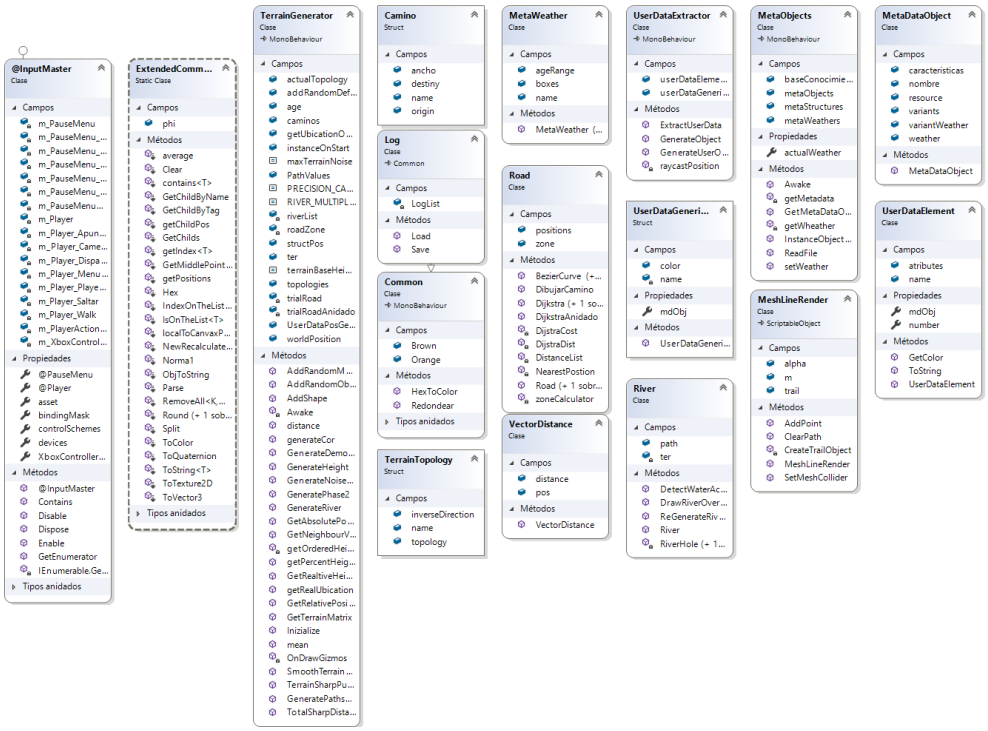


Figura 6.38 Listado de clases, atributos y métodos desarrolladas para esta parte del proyecto

6.2.3 Base del conocimiento

La base del conocimiento como se comentaba al principio del capítulo no puede considerarse un módulo en sí mismo porque no tiene ni una sola línea de código en la actualidad, pero es lo suficientemente interesante y relevante para el proyecto como para ser tratada como tal. La base de datos de conocimiento en su versión está compuesta por dos archivos, la base de datos de objetos y el traductor a metalenguaje. Se podría pensar que ambos archivos son independientes, pero realmente esto no es así, el traductor necesita conocer los objetos de la base de conocimiento para saber el nombre que debe darle a las cosas. Empezaremos detallando este último por el mero hecho de seguir la dinámica de este documento de ir de lo básico y sencillo a lo enrevesado y más complejo. El siguiente listado (6.14) muestra un ejemplo del archivo de traducción:

```
<?xml version="1.0" encoding="utf-8"?>
<Root>
    <noun name = "Arbol" otherNames = "Arbolito, Tree, arboles"></noun>
    <noun name = "Rio" otherNames = "rio, rios, river"></noun>
    <noun name = "Casa" otherNames = "casa, house, hogar, casas"></noun>
    <noun name = "Coche" otherNames = "car, coche, auto, automovil, coches, autos, automoviles, cars"></noun>
    <noun name = "Montaña" otherNames = "mountain, montañas, mountains"></noun>
    <noun name = "Persona" otherNames = "person, humano, human, personas, humanos, people, humans"></noun>
    <noun name = "Corbata" otherNames = "tie, ties, corbatas"></noun>
    <noun name = "Camion" otherNames = "truck, camiones, trucks"></noun>
    <noun name = "Bicicleta" otherNames = "bici, cycle, bicycle, bicis, biciletas, cycles, bicycles"></noun>
    <noun name = "[LastName]" otherNames = "otro, otros"></noun>
    <adjetive name = "Big" otherNames = "Grande, Big, gran, grandes"></adjetive>
    <adjetive name = "Small" otherNames = "Pequeño, Pequeños, Pequeña, Pequeñas"></adjetive>
    <adjetive name = "VerySmall" otherNames = "Enano, Diminuto, Enana, Diminuta, Diminutos, Diminutas, Enanos, Enanas"></adjetive>
    <adjetive name = "#FF0000" otherNames = "Rojo, Magenta"></adjetive>
    <adjetive name = "#00FF00" otherNames = "Verde"></adjetive>
    <adjetive name = "#0000FF" otherNames = "Azul"></adjetive>
    <adjetive name = "#FFFF00" otherNames = "Amarillo"></adjetive>
    <adjetive name = "#CD00D4" otherNames = "Violeta, Morado"></adjetive>
    <adjetive name = "#F8B0FF" otherNames = "Rosa, Lila"></adjetive>
    <adjetive name = "#612E0F" otherNames = "Marron"></adjetive>
</Root>
```

Listado 6.14 Estructura del archivo de traducción

Este archivo es relativamente sencillo pues solo tiene tres partes, dos de ellas con la misma estructura, la primera es como siempre el contenedor de los datos, que este caso se le ha dado el nombre de “Root” este nodo es el que contiene a los demás y la base de la jerarquía del archivo. Un nivel por debajo de la raíz encontramos los nodos “noun” y los nodos “adjetive” ambas con la misma estructura de propiedades, “name” el nombre del objeto que sobrescribirá el texto de salida del extractor de datos y se usará para que el módulo de generación de datos sepa que generar, y “otherNames”, todos aquellos nombres por los que se pueden identificar el objeto, que serán los que se sustituyan por el anterior (“name”) en los textos que se analicen.

Esta sería la estructura de este archivo, para ver más en profundidad el funcionamiento y la utilidad de este archivo el lector puede remitirse al final del capítulo 6.2.1 donde se habla larga y extensamente del uso de este fichero.

El segundo y último fichero que en la actualidad compone la base de datos de usuarios, es, en este caso la base de datos de objetos. Este fichero es un poco más complejo que el anterior, pues incluye conceptos tales como la herencia (estructural no de código) entre objetos los a generar. Un ejemplo de estructura de este archivo sería la que sigue (Listado 6.15):

metadataObjects.xml | GestureRecognition.cs | RobotitoIntro.cs | SpeechRecognition.cs | SaveLoadGame.cs | salvableObject.cs | HandPonter.cs

```
<?xml version="1.0" encoding="utf-8"?>
<Root>
  <Objects>
    <Object name = "arbolMediterraneo" resource = "Broadleaf_Desktop_2">
      <maxHeight>60</maxHeight>
      <maxVangle>30</maxVangle>
      <maxSize>1.3</maxSize>
      <minSize>0.9</minSize>
    </Object>
    <Object name = "arbol" resource = "CartoonTree">
      <maxHeight>60</maxHeight>
      <maxVangle>30</maxVangle>
      <!-- <lockYRotation>TRUE</lockYRotation> -->
      <maxSize>1.3</maxSize>
      <minSize>0.9</minSize>
      <Variant name = "arbolMediterraneo" wheather ="Mediterraneo"></Variant>
      <Variant name = "arbol_Alantico" wheather ="Atlantico"></Variant>
    </Object>
  <Wheathers>
    <Wheather name ="Mediterraneo">
      <Box x = '-20' y ='30' width='100' height='50'></Box>
    </Wheather>
    <Wheather name ="Atlantico">
      <Box x = '-167' y ='-60' width='140' height='140'></Box>
    </Wheather>
  </Wheathers>
</Root>
```

Listado 6.15 Estructura del archivo XML de la base de datos de objetos

Como se puede observar en el archivo este archivo contiene una mayor variedad de datos que el anterior. La jerarquía del archivo (Listado 6.16) es la siguiente:

- Root
  - Objets
    - Objeto
      - Propiedades
      - Variantes
  - Structures
    - Structure
      - Object
  - Wheathers
    - Wheather
      - Box

Listado 6.16 Jerarquía del archivo XML de la base de datos de objetos

El nodo raíz para este archivo tiene el mismo nombre que en el anterior, root, su función es almacenar el resto de los nodos del sistema. En el siguiente nivel tenemos las categorías de objetos, climas y estructuras. Objets que contiene la lista de todos los objetos que puede generar el sistema, Structures que almacena las estructuras que están compuestas por conjuntos de objetos y Wheather que contiene la lista de climas en los que se pueden clasificar los distintos elementos, aunque como ya vimos en el capítulo 6.2.2, no tienen por qué ser climas, sino que se pueden hacer una gran cantidad de subdivisiones, dentro de los climas encontramos las boxes o cajas, que son el conjunto de rectángulos que componen un clima, las dimensiones y posiciones de estos rectángulos dependen del mapa que se defina en el editor de Unity, además los climas también pueden poseer un intervalo histórico lo que dota al sistema de un tercer eje convirtiendo las cajas antes mencionadas, realmente en cubos tridimensionales, generando así una matriz de 6 vértices completamente cargada de posibilidades. Volviendo a los objetos, dentro de la categoría de objetos, encontramos los objetos que se componen básicamente de un nombre y un recurso (Modelo 3D normalmente, aunque también pueden ser “Prefabs” de Unity) y dentro de estos podemos encontrar sus propiedades como la altura máxima o el tamaño relativo máximo del objeto respecto a su modelo 3D. Al mismo nivel también encontraremos las variantes que son variaciones de un objeto para un clima dado, en el listado 6.?? Puede verse como el objeto árbol tiene dos variantes, una para un árbol mediterráneo y otra para uno atlántico, cada uno con sus propias propiedades.

6.2.4 Escenario de pruebas

Para la realización de este proyecto, además de lo detallado anteriormente, se ha diseñado un escenario de pruebas para probar el funcionamiento del sistema en un entorno real. Para comprobar si por ejemplo si el sistema de físicas funciona correctamente o de si la interacción con el sistema es correcta.

Lo primero para esta parte será desarrollar los Assets (Modelo 3D) que necesitaremos para realizar las pruebas, en este caso un humanoide. Este personaje ha sido desarrollado con Blender y dotado de un esqueleto completo, de manera que puede hacer uso del sistema Mecanim (ver capítulo 5) para el paso de animaciones entre humanoides. El proceso creativo de un personaje con Blender no será explicado en este documento, pero si el lector está interesado en dichos temas, ha de saber que este tema sí que está desarrollado en el complemento de este

trabajo fin de grado, bajo el capítulo 6.1, “Diseño de los recursos”. A continuación, se muestra el resultado del modelado del personaje en Blender (Figura 6.39).

Figura 6.39 Personaje generado en Blender

A diferencia del personaje del complemento, a este se le han diseñado también apartados como puede ser el mapeado de las texturas o las texturas en sí también, un ejemplo de esto puede verse en la siguiente figura, mostrando estas las botas del personaje, que han sido diseñadas, modeladas, mapeadas y pintadas a mano directamente por el alumno (Figura 6.40).

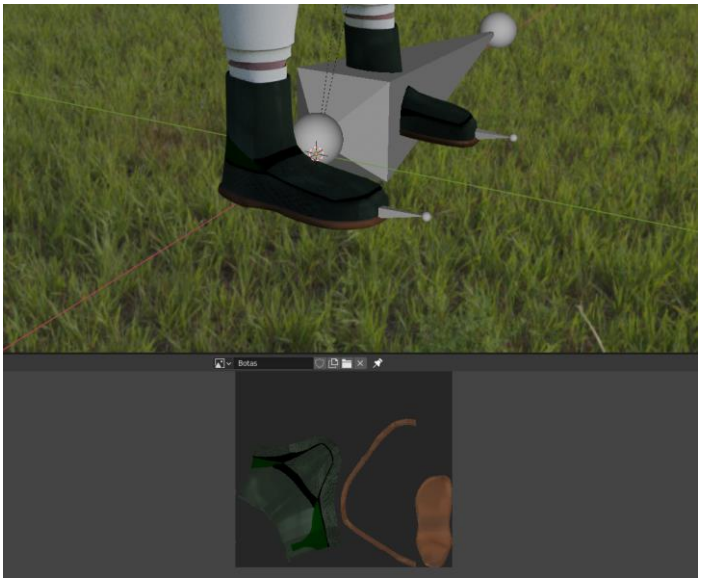


Figura 6.40 Botas del personaje de pruebas, arriba resultado del objeto 3D abajo texturas hechas a mano para el personaje

Además del anterior también se han creado modelos 3D de un arco y una flecha para poder interaccionar con el sistema que se vincularán finalmente con el usuario una vez ya dentro de Unity. Donde además se han diseñado y adaptado sus materiales para que puedan funcionar con este motor. Algunos de los modelos aquí diseñados como por ejemplo la camiseta del personaje, hacen uso de sistemas avanzando de cálculo de iluminación como la utilización de mapas de normales para darle más profundidad al objeto (compárese la siguiente figura con la figura 6.41 Y se podrá apreciar como la camiseta gana pliegues y detalles en la última gracias a la implementación entre otras cosas de mapas de normales).

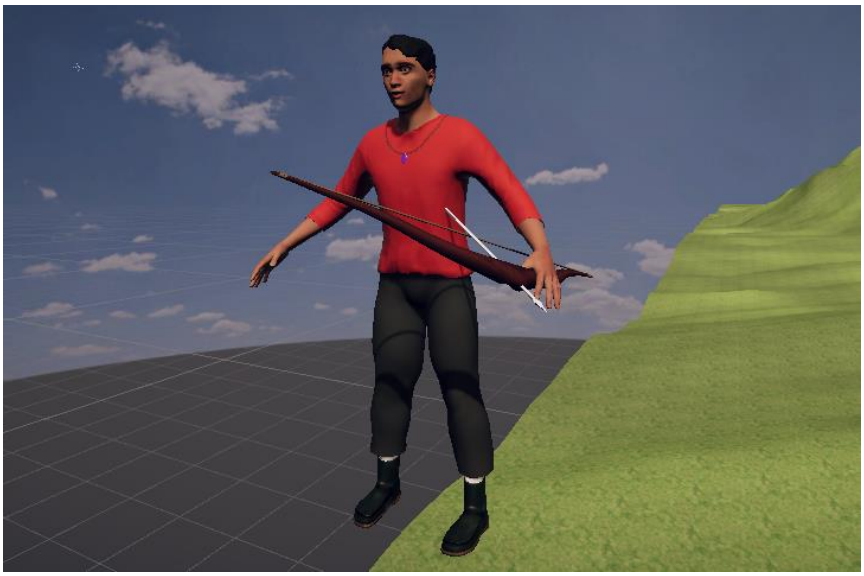


Figura 6.41 Personaje integrado en unity3D junto con la creación de sus materiales

Una vez diseñado el personaje, hay que dotarle de vida, hay muchas maneras para hacer esto, pero lo habitual suele ser combinar pequeñas acciones que en su conjunto hagan dar al personaje una respuesta más realista a las interacciones con el jugador y el entorno. Uno de estos ejemplos es el de las animaciones para la que en este caso se hace uso de una máquina de estados multinivel como se puede ver a continuación (Figura 6.42):

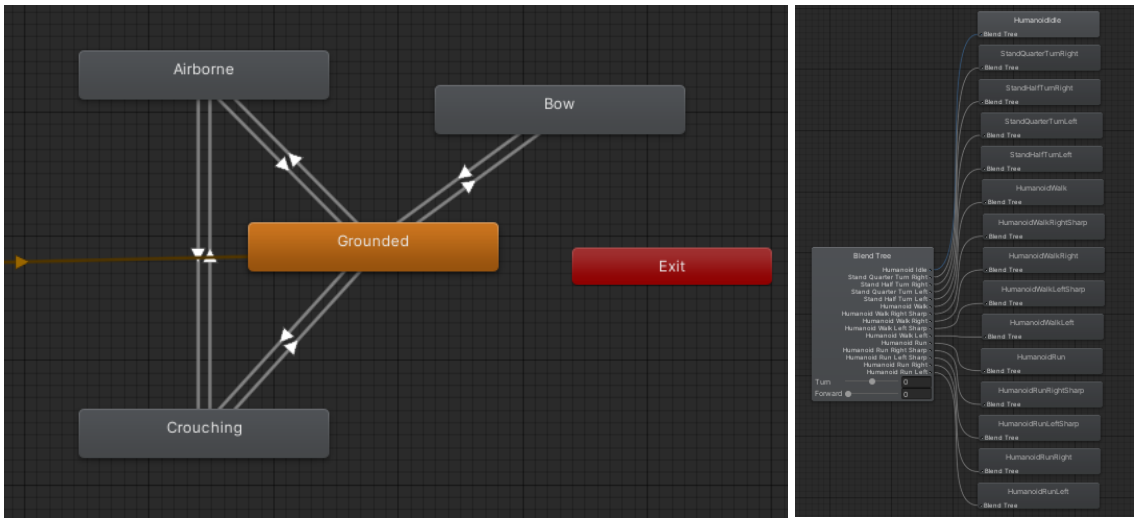


Figura 6.42 Maquina de estados del personaje princial, izquierda nivel 0 de la jerarquía, derecha nivel 1 de profundidad para el estado "Grounded" del nuvel 0

Esta máquina de estados controla los movimientos generales del personaje como caminar, correr o saltar, pero por si sola, está muy lejos de crear una sensación de realismo, puesto que hay muchas funciones como los gestos de la cara que este sistema no controla.

Respecto a este último se ha diseñado un sistema que permite que el jugador gire la cabeza y los ojos para mirar en una dirección (en este caso la cámara cuando se posiciona frente a él) y que cambie su animación de la cara para sonreír, y se ha añadido también un sistema de parpadeo que hace que le personaje parpadee de vez en cuando, dotando de mayor expresionismo y realidad a sus animaciones.

Otro de los puntos que se han desarrollado es la conocida como IK [15] o Cinemática Inversa. Cuando pensamos en desplazar un objeto articulado como puede ser el brazo robot de la siguiente figura podemos que ver que para llevar el brazo a otro sitio tendremos que ir ajustando las rotaciones de q1, q2, 13, y q4 en ese orden hasta llevar el agarre del brazo a donde queramos, este es el sistema usual que utilizan las animaciones y es un sistema que funciona francamente bien pero que tiene sus limitaciones cuando lo que queremos no es ejecutar un gesto o un movimiento para una animación si no que lo que queremos es llevar por ejemplo ese agarre un punto concreto, dándonos un poco igual la forma que tomen los puntos (q1-q4) anteriormente mencionados. Aquí es donde entra en juego la cinemática inversa, donde nosotros definiremos el logar en el que queramos colocar el extremo de una articulación como podría ser una mano o un brazo y el sistema nos calculará de manera automática la posición más natural del brazo o la pierna respectivamente para llevar esa mano o pie al punto indicado. La siguiente figura (6.43) muestra esquemáticamente este funcionamiento.

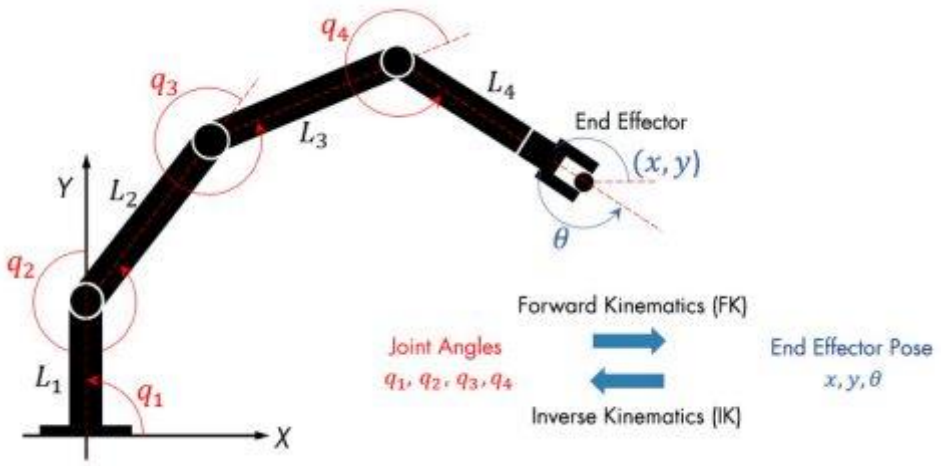


Figura 6.43 Cinemática Directa e Inversa. Fuente mathworks.com

¿Qué utilidad tiene esto? Bueno en este trabajo se le han encontrado dos, la primera con los brazos para sujetar el arco y las flechas, queremos que el jugador tense el arco y tire de él, por lo que sabemos exactamente dónde queremos colocar la mano del personaje (sujetando el arco y su cuerda), pero si tuviéramos que calcularlo a partir de la posición del hombro, luego del brazo

superior, brazo inferior y mano (Figura 6.44), sería una tarea bastante tediosa, por eso se han utilizado las cinemáticas inversas para resolver este problema.

Figura 6.44 Animación hecha a mano de tiro con arco, donde las posiciones de las manos y los pies se calculan con cinemáticas inversas.

El otro lugar donde hemos hecho uso de esta tecnología como ya nos adelanta la figura de arriba es en los pies, normalmente las animaciones de un personaje suelen ser fijas, no dependen de su entorno, una animación de correr, por ejemplo, suele diseñarse para un personaje corriendo sobre un suelo plano, ¿pero qué ocurre cuando este mismo personaje intenta corree por una colina?, pues que su animación sigue siendo como si corriese sobre un plano y por tanto al final sus pies acabarán o flotando o atravesando el terreno, este problema es un poco más complejo que el anterior puesto que lo que queremos no es que los pies del personaje se coloquen la tierra todo el tiempo y los vaya arrastrando por el escenario, sino que se fije al suelo en función de la distancia con este y el estado de la animación actual.

```
private void OnAnimatorIK(int layerIndex)
{
    if (!animator) return;
    if (ikActive) {
        float dist = checkDistance - upDistance;
        for (int i = 0; i < foots.Length; i++) {
            RaycastHit rh;
            AvatarIKGoal goal = foots[i].gameObject.name.ToLower().Contains("_l") ? AvatarIKGoal.LeftFoot : AvatarIKGoal.RightFoot;

            if (Physics.Raycast(foots[i].position + Vector3.up * upDistance, Vector3.down, out rh, checkDistance, layers))
            {
                float weight = Mathf.Min(Mathf.Max(0, 1 - Vector3.Distance(foots[i].position, rh.point) / dist),1);
                animator.SetIKPositionWeight(goal, weight);
                animator.SetIKRotationWeight(goal, weight);
                animator.SetIKPosition(goal, rh.point + addDistance);

                animator.SetIKRotation(goal, Quaternion.FromToRotation(transform.up, rh.normal) * transform.rotation);
            }
            else
            {
                    animator.SetIKPositionWeight(goal, 0);
                    animator.SetIKRotationWeight(goal, 0);
            }
        }
    }
}
```

Listado 6.17 Método de control de lass cinemáticas inversas de los pies

El anterior método (Listado 6.17) detalla el funcionamiento de las cinemáticas inversas a nivel de código para los pies del jugador mientras que la siguiente figura muestra, los distintos scripts

diseñados para el control de estas cinemáticas dentro de Unity3D. La interfaz gráfica de estos scripts puede verse en la figura (6.45) que viene a continuación:

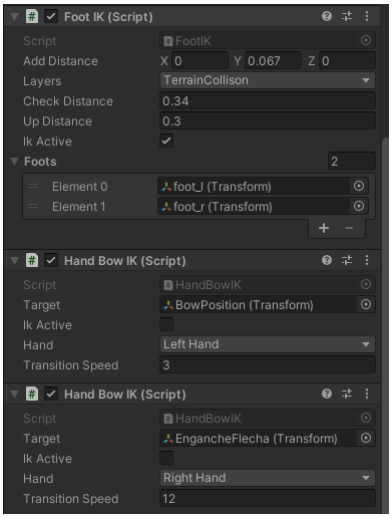


Figura 6.45 Scripts para el control de las cinemáticas inversas

Otro punto interesante de los desarrollados para este sistema de pruebas es la que ya se nos ha ido introduciendo con anterioridad, el sistema de disparo de flechas, este sistema es interesante por dos motivos, el primero de ellos es que las flechas están dotadas de físicas al lanzarse por lo que son un muy buen ejemplo para poder interactuar con los sistemas de colisiones y de objetos físicos de Unity y nuestro sistema. El segundo motivo, mucho menos relevante que el anterior, es para probar el sistema multicámara y la respuesta de los controles a este tipo de interacciones. Como ya se ha visto en la figura 6.44 Y como se verá en la siguiente (6.46), el sistema de arcos y flechas se integra con facilidad con la estética del juego y es un sistema, que a pesar de haber llevado muchas horas de trabajo y de pruebas (ajustando parámetros, posiciones, fuerzas, etc.), ha sabido dar una respuesta positiva a lo que se esperaba de él.

Figura 6.46 Cambios de cámara y animaciones del uso del tiro con arco

Finalmente, concluiremos esta parte, por un lado, haciendo mención de que también se ha diseñado un sistema de menú de pausa interactivo a través del mando de juego que detiene y reanuda la configuración del juego y que tiene un par de opciones. Y por otro hablando, de la interacción entre el usuario y el sistema que en este caso se ha diseñado a través del Nuevo sistema de inputs de Unity y que permite la utilización de mandos de Xbox Series X | S (Figura 6.47)y Xbox One | One S | One X para manejar el sistema. Inicialmente la interacción se planifico con el sistema antiguo de Unity en el cual se capturaban directamente las pulsaciones de los botones del mando en el código.

Figura 6.47 Controlador de Xbox Series S, el utilizado para controlar al jugador en la escena de pruebas

El nuevo sistema de Unity permite hacer uso de varias capas de abstracción de manera que el desarrollador no tiene que trabajar directamente a nivel de código con los botones del mando, si no con unas abstracciones de estos en forma de Acciones, que pueden ser desde simples entradas como pulsar un botón o mover un Joystick a opciones más complejas como pulsar una combinación de teclas o botones. Una de las mayores ventajas de este sistema es que el desarrollador no tiene que estar pensando en los mandos o sus configuraciones, de manera que cuando quiera añadir un mando nuevo solo tendrá que añadirlo al sistema de entradas y asignarle las acciones necesarias para que funcione, tal y como se muestra en la siguiente figura (6.48), despreocupándose de todo lo demás.

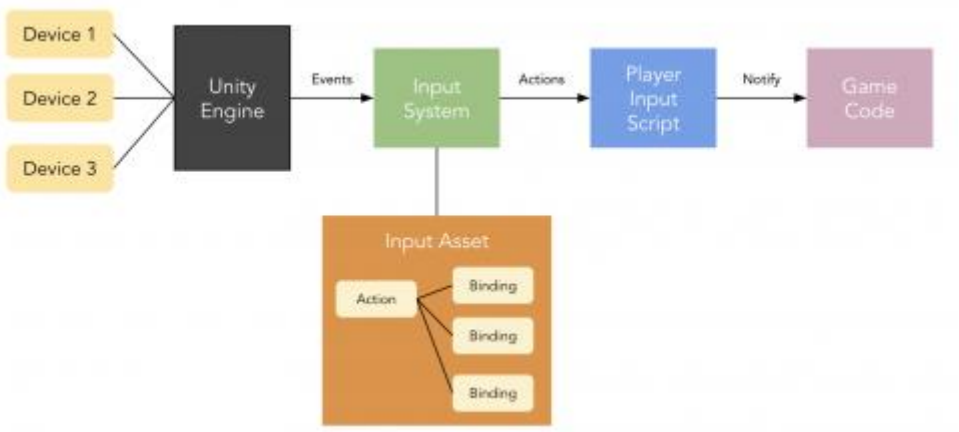


Figura 6.48 Funcionamiento del nuevo sistema de entradas de Unity [16]

Finalmente, al igual que en las secciones anteriores, terminaremos esta sección mostrando el sistema de clases que han compuesto esta parte del proyecto (Figura 6.49):

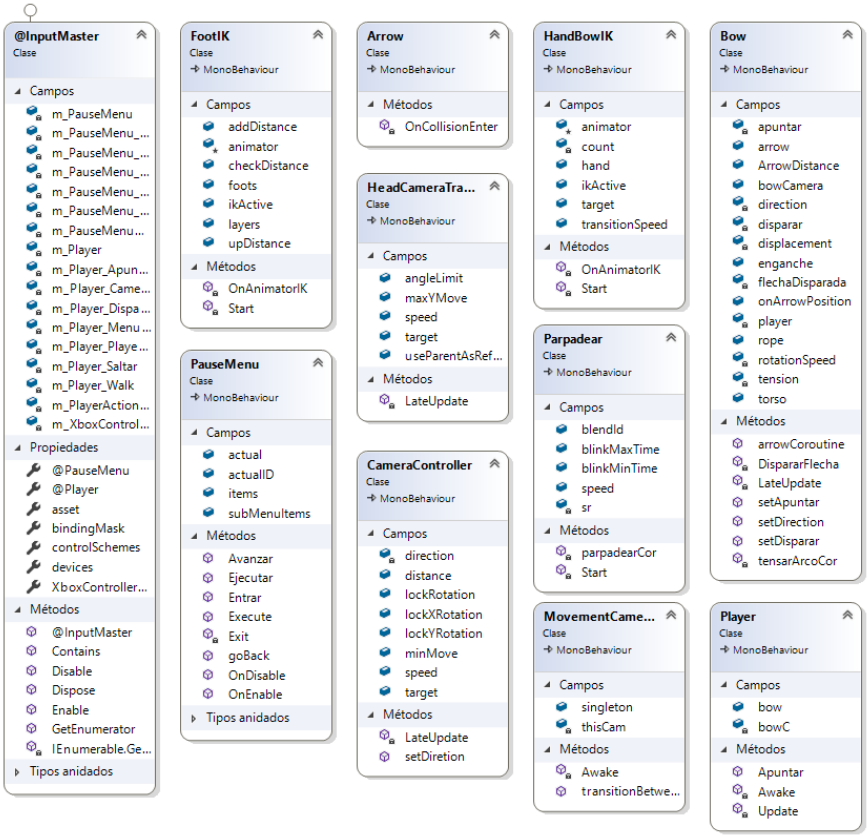


Figura 6.49 Clases generadas para la escena de pruebas

7 DESPLIEGUE

Esta sección cubre el proceso de despliegue de la aplicación, es decir, el proceso mediante el cual la aplicación pasa de ser un software de desarrollo a un producto final que pueden usar los consumidores. Pasando en el último punto de este capítulo por las pruebas y el análisis con consumidores reales.

7.1 Despliegue de la App de captación de datos

La aplicación de captación de datos está diseñada como una aplicación de escritorio de Windows Forms por lo que su trabajo de despliegue es más bien escaso, lo primero que deberemos hacer es compilar la aplicación a través del menú "Compilar → Compilar Solución" aunque este paso no es realmente necesario si previamente hemos ejecutado la aplicación con anterioridad, puesto que Visual Studio la compila antes de ejecutarla. Aun así, siempre es recomendable hacer este paso para asegurarnos que la versión compilada disponible, será siempre la última. Una vez compilada la aplicación podremos acceder a ella mediante la ruta del proyecto / bin / Debug / netcoreapp3.1 (Figura 7.1)

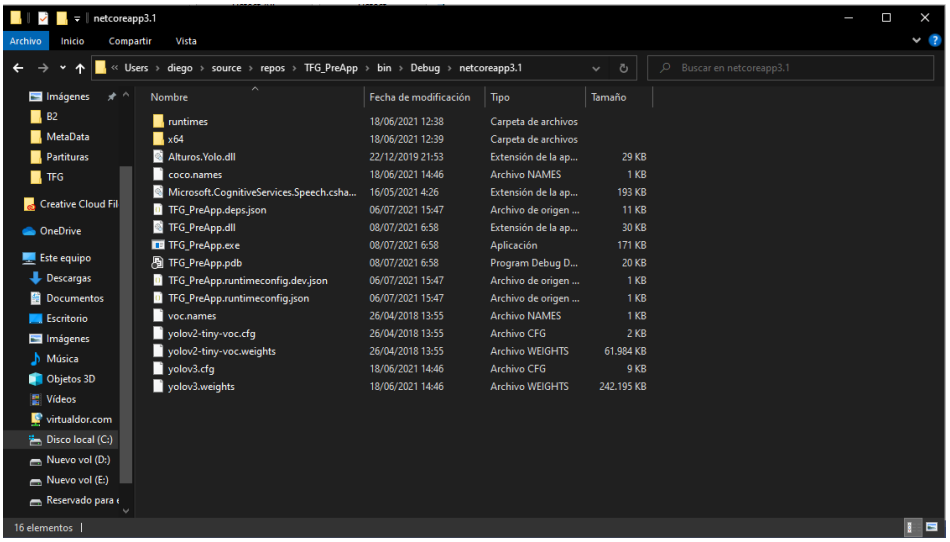


Figura 7.1 Carpeta de los archivos binarios del programa una vez compilado

En esta carpeta encontraremos todos los archivos necesarios para hacer uso del proyecto, por lo que ya podríamos copiarla a un pendrive o un disco duro y pasárselo a quien quisiéramos, pero en este caso, al igual que haremos en el Complemento de este trabajo, vamos a comprimirla, pero a diferencia del complemento, que generaba un archivo .EXE en este caso debido al sistema que usaremos para compartir la información generaremos un archivo .RAR, hacer esto es muy sencillo puesto que solo necesitaremos tener el programa WinRAR instalado en nuestro ordenador y después hacer clic derecho en la capeta "netcoreapp3.1" y pulsar sobre la opción de añadir al archivo, al hacerlo se nos abrirá una ventana con varias opciones como el formato de compresión deseado o el método de compresión. En nuestro caso lo dejaremos como se muestra a continuación (Figura 7.2):

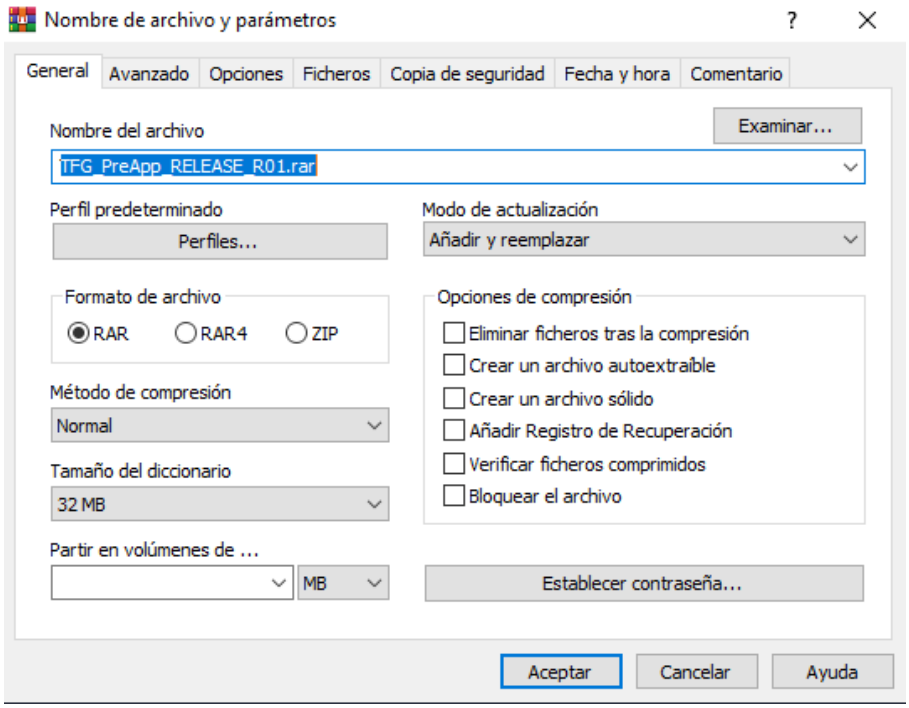


Figura 7.2 Opciones de compresión de WinRAR

Con el archivo ya comprimido es más fácil compartirlo a otras personas, como en este caso para la utilización de esta parte del software no se requiere de ningún dispositivo adicional, el software se compartirá a través de un enlace de Google Drive para que cualquier persona con el enlace pueda hacer uso de este programa.

Por lo que accederos directamente a este servicio (Google Drive) (Figura 7.3) a través de la cuenta de Google y crearemos una carpeta nueva llamada TFG a la que arrastraremos el archivo rar que hemos creado y esperaremos a que termine de subirse.

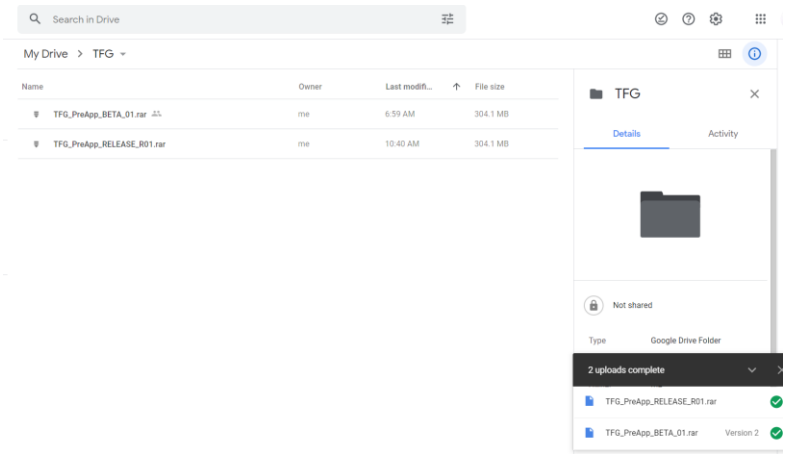


Figura 7.3 Archivo rar de la parte de extracción de datos en GoogleDrive

Una vez subido el archivo ya está listo para compartirse, pero por defecto todos los archivos subidos a drive se consideran privados (solo tiene acceso a ellos el propietario de la cuenta) siempre que el usuario no diga lo contrario. Por lo que, si queremos compartirlo, deberemos cambiar esta política, para ello hacemos clic derecho en el archivo que hemos subido y pulsaremos sobre la pestaña compartir (el botón “share” en inglés).

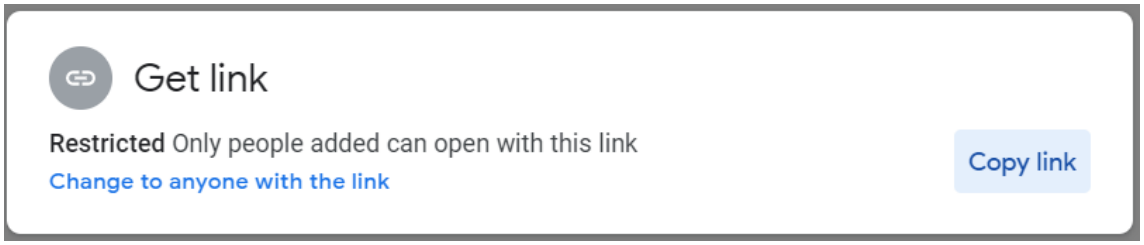


Figura 7.4 política de privacidad por defecto de Google Drive

Nos aparecerá una nueva ventana con un mensaje como el que aparece arriba (Figura 7.4), para desprivatizar el acceso al archivo simplemente deberemos pulsar sombre el enlace/botón “Change to anyone with the link” para que de esa manera cualquier persona con el enlace pueda acceder al archivo. Una vez hecho esto, la ventana habrá cambiado (Ver figura 7.5) y ya podremos copiar el enlace para acceder al archivo, donde ya finalmente solo nos quedará pulsar en aceptar (done) para guardar los cambios y salir.

Figura 7.5 Archivo .RAR del proyecto compartido correctamente por Google Drive

7.2 Despliegue de la herramienta de generación de escenarios

Si la primera parte de este capítulo se centraba en el despliegue de la herramienta de la extracción de datos, esta segunda se centra en lo propio con la herramienta de generación de entornos, que de nuevo al igual que los otros dos despliegues (el anterior y el del complemento de este trabajo) la metodología seguida para desplegar esta herramienta es algo diferente a las anteriores, pues en esta ocasión tenemos un producto orientado a los desarrolladores y por tanto no podemos generar los archivos binarios (.EXE) y mandárselos, puesto que en lo que los desarrolladores están interesados no es en la versión compilada del proyecto si no en el código de esta. Por lo tanto, lo compartiremos en esta ocasión será todo el proyecto de Unity, junto con sus Assets y sus sistemas de pruebas (lo que incluye también el código del presente complemento del trabajo fin de grado, como una prueba de las capacidades de este sistema).

En esta ocasión nos encontramos con el mismo problema y la misma solución que los casos anteriores, puesto que los archivos que queremos compartir son muchos y tienen un peso bastante alto, el mayor de todos hasta ahora de hecho (casi 4 Gibibytes), por lo que de nuevo deberemos recurrir a los sistemas de compresión de información concretamente WinRAR para generar agrupar todo el proyecto en un único archivo y que con suerte su peso sea inferior al que ya tenemos (Figura 7.6)

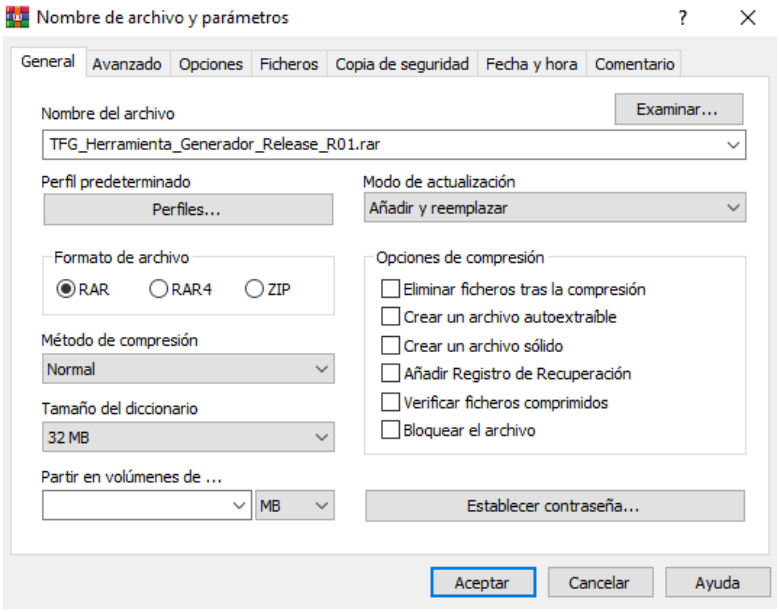


Figura 7.6 Compresión de la herramienta para desarrolladores en un archivo RAR

En esta ocasión podríamos generar un archivo .EXE en vez de un .RAR ya que los archivos no se van a subir a Drive en primera instancia, pero al ser un producto de desarrollo y no una aplicación para el usuario final, se ha considerado más conveniente guardarlo todo en un archivo rar. Además, en este caso, de hecho, y dado al alto peso que podría tener el archivo, podríamos usar un sistema de compresión más agresivo para reducir aún más el espacio que ocupa, pero por contraposición esto supondría más tiempos de compresión y descompresión de archivos lo cual no es interesante y menos para grandes tamaños de datos (Aunque 4 Gibibytes ya no lo son tanto en los tiempos que corren) por lo cual hemos preferido tomar como se muestra en la figura superior los parámetros por defecto para la compresión del archivo .RAR que sorpresivamente, sin ser muy agresivos con la compresión, ha visto reducido su peso hasta las 1.5Gb haciendo

posible el mandar este archivo a través de sistemas como WeTransfer, Dropbox u otros. Ya que este archivo tiene un peso superior a los demás, y sumando a esto se desea poseer cierto control sobre él, la opción escogida para traspasarlo ha sido principalmente en físico, vía Pendrive USB, asegurándonos así que no se hagan copias ilegítimas de este a través de terceros.

7.3 Pruebas con usuarios

Finalmente, esta es la última parte del capítulo, una vez compilados los proyectos necesarios y comprimidos los demás es hora de probar cómo se comporta el software dentro de entornos reales, tanto con profesionales (la parte de generación y la de captación de datos) como usuarios menos experimentados (solo captación de datos). Se han planificado sesiones de uso de aproximadamente 15-20 minutos, tras las cuales a los usuarios se les ha pasado un cuestionario, o dos (Véase Anexo I) en caso de haber probado ambas partes del programa, con el fin de saber su opinión y valoración de software, a continuación, se detallará el análisis hecho sobre estos cuestionarios.

7.3.1 Herramienta de Captación de datos

Esta encuesta ha sido pasada a un total de 12 personas con edades comprendidas desde los 16 a los 54 años. De las cuales el 67% se identificaron como hombres, el 25% como mujeres y el 8% como otro.

En la titulación académica la mayoría de los encuestados poseía un título de grado (6/12), seguida por los títulos de doctorado (2/12) y bachillerato (2/12), además, una persona declaró tener un grado medio y otra un graduado escolar.

Respecto a la experiencia con los ordenadores, las respuestas han sido variadas, indicando un 42% (5/12) Tener un nivel experto en el manejo de ordenadores, mientras que un 33% (4/12) declaró hacer un uso avanzado del ordenador, finalmente un 25% (3/12) declaró poseer un nivel básico de manejo de ordenadores.

Una vez definidos estos datos para identificar el tipo de usuarios que compondrían la muestra, a estos se les pidió responder con valoración de 1 a 5 donde 5 es muy de acuerdo y 1 muy en desacuerdo, su grado de acuerdo con un total de 15 afirmaciones repartidas en 5 categorías distintas que son: Presentación, Interactividad y manejo, contenido, funcionamiento y eficacia y finalmente, compromiso. A continuación, la siguiente tabla (7.1) muestran los valores medios obtenido para cada una de estas afirmaciones:

Presentación			Interactividad y Manejo				Contenido		Funcionamiento y Eficacia		Compromiso		
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
4,08	4,58	4,67	4,42	4,42	4,58	4,75	4,58	4,67	5,00	4,50	4,92	4,75	4,92

Tabla 7.1 Valoración media por afirmación en el cuestionario del módulo de recolección de datos

Como se puede observar, la valoración de todas las afirmaciones ha sido bastante alta en general, no bajando ninguna de una nota media de un 4 sobre 5. La afirmación con la que de media los usuarios han estado más de acuerdo ha sido la décima “No han surgido errores que han obligado a detener la ejecución el programa”, con una puntación perfecta. Demostrando así que el programa se ha ejecutado sin ningún tipo de problema. También ha obtenido una valoración muy alta la afirmación número 14, “Volvería a utilizar el programa si tuviera oportunidad”. Poniendo de manifiesto el interés que han tenido los usuarios por este software.

Por otro lado, la afirmación con la que los usuarios han estado de media menos de acuerdo es la primera, “La estética general del programa es adecuada” con una valoración media de 4,08/5 poniendo de manifiesto que se debe trabajar más el apartado visual de esta aplicación. Como ultimo punto a destacar de esta parte, la valoración general media de los usuarios ha sido de 8.92 puntos.

Finalmente, en la sección de comentarios abiertos, el 50% de los usuarios han hecho alguna valoración o comentario, mientras que el otro 50% no ha respondido nada.

La pregunta de “que ha sido lo más interesante del programa” ha recibido una gran variedad de respuestas, desde la generación de datos a partir del reconocimiento de voz, a la velocidad de reconocimiento, la sencillez de uso o el reconocimiento de imágenes.

En lo menos interesante, dos valoraciones han coincidido en la exportación a fichero XML y una ha valorado la detección de la ubicación del usuario como la menos interesante.

Por otro lado, lo que más se ha echado en falta por parte de los usuarios ha sido, mas variedad en la extracción de datos de las imágenes, el reconocimiento de vídeos y no solo de imágenes estáticas y la posibilidad de guardar y cargar los datos y finalmente una barra de carga para los procesos que duren mucho.

Como sugerencias se ha solicitado principalmente una mejora en la interfaz y la adaptación al castellano de los menús y botones de la aplicación.

Finalmente, para acabar esta parte, la siguiente figura, (7.7) Resume visualmente las puntaciones medias dadas a cada categoría, asi como la nota media final de este módulo del sistema:

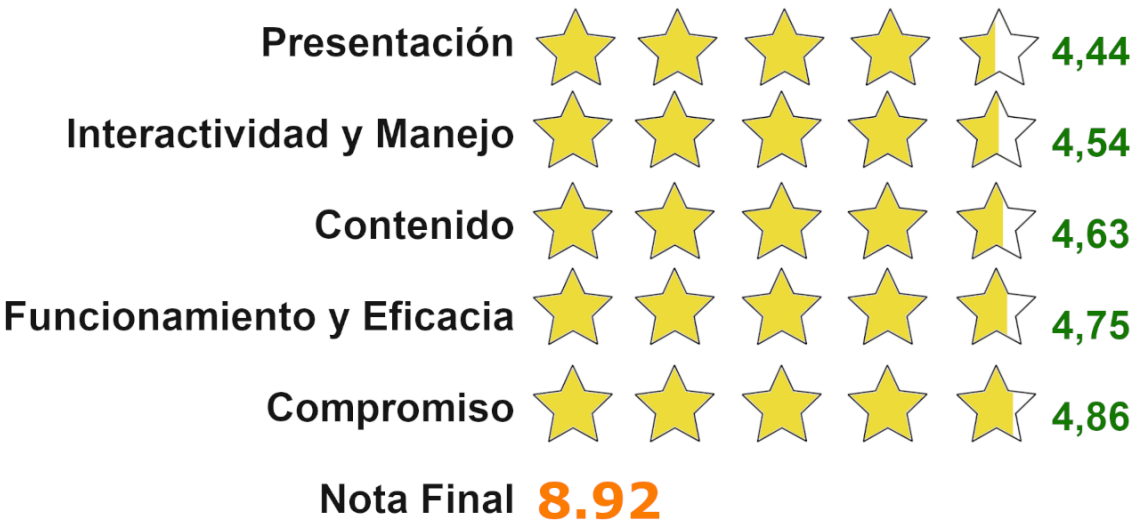


Figura 7.7 Valoración final de los usuarios por categoría al módulo de recolección de datos

7.3.2 Módulo de generación de escenarios

Este análisis es algo distinto al anterior, puesto que ya no va enfocado a un usuario normal, si no hacia diseñadores y programadores que trabajen con Unity3D, es por ello por lo que la cantidad de encuestados se ha visto reducida de 12 a 4, en los cuales se ha intentado buscar cierta variedad de perfiles.

Respecto a la experiencia con ordenadores todos han valorado su nivel como experto. Por otro lado, en la experiencia (en años) con Unity3D, donde se ha buscado la variedad de perfiles, uno de lo encuestados posee más de 10 años de experiencia, otro posee entre 5 y 10 años, otro entre 2 y 5 años y finalmente el último tiene menos de dos años. Por lo que se ha cubierto el espectro completo de esta pregunta.

Dado que el número de encuestados en esta parte ha sido bajo, no considero que el estado de acuerdo individual de cada pregunta tome un peso significativo en este análisis, por lo que me centraré en expresar las valoraciones medias por categoría que al igual que en la sección anterior, se mostrarán de forma gráfica (Figura 7.8):

Figura 7.8 Valoración final de la herramienta de generación según Expertos

De lo anteriores puntos se puedes destacar varios puntos, en primer lugar las valoraciones por categoría, principalmente Interactividad y Manejo, han bajado de suavemente de puntación respecto al otro módulo, esto principalmente puede deberse a que al estar esta parte valorada por expertos, estos son más críticos que los usuarios normales dado que están acostumbrados a trabajar con este tipo de sistemas y no les sorprende tanto, por otro lado, el dato que considero más positivo, es el compromiso y la nota final, que son muy altos, (en el caso del compromiso, ha obtenido una calificación perfecta). Esto denota, un alto interés por los expertos en utilizar este tipo de tecnologías una vez estén más maduras. Aunque dado los resultados obtenidos, es probable que el grado de madurez del programa aún sea relativamente bajo, si parece que, de seguir trabajando en él, puedan cosecharse unos muy buenos resultados de cara a un futuro no tan lejano.

8 CONCLUSIONES

Procedural-MIND ha sido un trabajo arduo, difícil, desafiante para mí que me ha hecho dar un 110% constante para poder estar hoy aquí escribiendo estas líneas. Este proyecto me supuesto el uso de muchas tecnologías que nunca había utilizado, me ha hecho buscar soluciones complejas y creativas a los problemas que han ido surgiendo, aun así, Procedural-MIND ha alcanzado su objetivo de generar escenarios de manera procedural, adaptada a los deseos y datos de los usuarios.

Se ha podido desarrollar en dos herramientas como se planeó desde un principio, siendo una, la de recolección de datos siendo posible de utilizar por un usuario sin rol técnico y la segunda una herramienta para desarrolladores integrada dentro de Unity3D.

Se ha generado un sistema también capaz de recolectar la voz del usuario, los datos de su geolocalización y aunque al final no se han podido cargar fotos directamente de su perfil de redes sociales, esto no ha sido un impedimento, puesto que lejos de desechar la funcionalidad, se le ha dado una vuelta de tuerca, y se ha generado un subsistema que permite al usuario cargar las fotografías que desee para luego analizarlas para obtener objetos personalizados con los datos de las fotografías (el color del objeto).

Se ha diseñado una interfaz personalizada para la herramienta de unity3D dotándola de nuevos elementos visuales con los que no cuenta la herramienta como la opción de un mapa sobre el que se puede hacer clic para obtener las coordenadas relativas del ratón sobre el mapa. Se le ha dotado de conectividad con servicios externos para poder por ejemplo obtener la geolocalización del usuario en tiempo real a través de la IP sin que este tenga la necesidad de contar con ningún módulo de GPS o similares.

Se creado un sistema de climas y variantes que permite que los objetos que genere el usuario puedan estar adaptados a su tierra, a su entorno y cultura, dotando así al sistema de mayor personalidad y capacidad creativa. Y es más se ha llevado este sistema más allá de lo inicialmente planeado, añadiendo soporte a que los climas puedan ser dependientes de una barra de tiempo de manera que se pueden tener varios climas solapados espacialmente pero que pertenezcan a distintas épocas, como por ejemplo la Europa moderna, Europa renacentista, Europa medieval, etc., donde dependiendo del “clima” un objeto “coche”, pueda ser un deportivo, un coche de los años 50 o un carruaje de caballos.

8.1 Detalle específico de lo aprendido

Este TFG comprende el final de una etapa de 5 largos años de mi vida hacia el ansiado “pin del ingeniero”, o bromas aparte, del título de ingeniero Informático. Aun así, este trabajo supone también la puerta hacía un nuevo mundo de posibilidades y nuevas metas, encabezadas por el máster y más allá aún, el doctorado.

No son pocas las cosas que he aprendido durante mi periodo de estudios que han terminado culminando como parte del desarrollo de este proyecto, y más siendo este el primer desafío real a mi camino como futuro investigador. De entre todas ellas, además de las ya nombradas anteriormente hay algunas para este trabajo que si me gustaría nombrar:

- **Planificar y gestionar un proyecto software desde 0.** Este trabajo me ha ayudado a comprender mejor el ciclo de vida de un proyecto, desde su concepción como idea, hasta su despliegue como producto, pasando por supuesto por las distintas etapas de análisis o codificación. Ahora ya me veo capacitado para afrontar nuevos retos de este o mayor nivel en el futuro.
- **Análisis económico de un proyecto.** Otro de los puntos en los que considero que me ha ayudado el proyecto es a aprender a analizar el coste real de un proyecto, a aprender a gestionar los retrasos y a tomar las decisiones difíciles de tener que perder funcionalidad a cambio de tipo o de transformar esas funcionalidades que no están llegando a ninguna parte en otras funcionalidades nuevas e interesantes.

Desde un marco más técnico y definitivamente más orientado a la programación creo que este proyecto me ha sido de gran utilidad, para este proyecto he querido salirme de mi zona de confort, trabajar con herramientas, programas y métodos distintos a los que me había que tenido que enfrentar anteriormente. Y creo que eso es algo que me ha ayudado a crecer como programador y como futuro ingeniero informático. Aun así, dentro de este punto me gustaría remarcar los siguientes puntos que he aprendido a lo largo de este proyecto.

- **Trabajar con cadenas de texto y analizarlas.** Ya había trabajado anteriormente en analizar alguna cadena de texto, pero no al nivel de intentar interpretar y traducir, de la mejor manera posible (acorde a mis conocimientos) sintaxis tan complejas como las del propio español de manera que pudieran ser entendidas por un sistema informático. Creo que es un paso que me ayuda a comprender mejor la sintaxis humana y como nos comunicamos, y creo que es una habilidad que me ayudará a diseñar IAs y Autómatas más potentes e interesantes en el futuro, que puedan comprender mejor a las personas y que puedan en algún momento comunicarse con estas de manera más fluida y natural.
- **Lenguajes de paso de mensajes XML.** Otro de los puntos fuertes de mi aprendizaje en este proyecto ha sido el de trabajar con sistemas de pasos de mensajes para la comunicación entre programas / procesos. Suelo estar acostumbrado a gestionarlo todo desde un mismo programa, pero este punto de vista de trabajar por componentes que se comunican entre sí me ha abierto los ojos a muchas posibilidades y creo que es una faceta a la que le sacaré provecho en un futuro no tan lejano.
- **Trabajar con APIs y servicios en la nube.** Otro de los platos fuertes del proyecto ha sido trabajar con APIs y, en definitiva, software de terceros y tener que gestionar la comunicación con estos, especialmente cuando hablamos de sistemas en la nube como

pueden ser Microsoft Azure o IPInfo. Sin duda, otra de las habilidades que pienso aprovechar en el futuro.

- **Trabajar con sistemas de reconocimiento de imagen.** Otro de los elementos en la misma línea que las anteriores, es la del reconocimiento de objetos en imágenes una de las líneas de investigación para mi más interesantes del panorama actual, puesto que es clave para que sistemas como los que controlan un robot puedan tener constancia del sistema que los rodea. Esta es la primera vez que hago uso correcto de una API y para mí es algo sumamente satisfactorio, puesto que es algo que he llevado varios años intentando sin mucho éxito.
- **Trabajar con datos de geolocalización.** Este punto quizás más flojo que los anteriores, pero igualmente relevante, me ha hecho ver lo fácil que es extraer información de un usuario de cosas tan simples como una dirección IP. Lo cual me ha hecho darme cuenta de la necesidad de implementar sistemas de seguridad y leyes que velen por la seguridad y privacidad de los ciudadanos en internet.

Otros aspectos quizás más de mi campo, pero no por ello irrelevantes son:

- **Editor de Unity personalizado.** Hasta ahora siempre he trabajado con el editor por defecto que brinda Unity a los desarrolladores y no conocía la abismal cantidad de posibilidades que brindan Unity para editar la interfaz gráfica de sus clases en el motor, desde crear variables gráficas calculadas o colocar anotaciones a crear mis propios componentes gráficos (como el mapa) que serían imposibles de crear sin hacer uso de estos sistemas.
- **Generación procedural de objetos.** ¡Oh generación procedural! Que más podría decir de una de las tecnologías (si es que se le puede llamar así) que siempre más me han interesado, que el sistema pueda autoconstruirse solo, no es casualidad que ese sea uno de los pilares fundamentales de este proyecto, este es hasta la fecha el proyecto de generación procedural más complejo que he enfrentado y también uno de los más satisfactorios hasta la fecha.
- **Edición de terreno 3D procedural.** Otra de esas funciones que sospechaba que existían pero que tras probarlo he quedado fascinado de como con un par de funciones, un par de número y una o dos texturas de profundidad puedo crear terrenos aún más realistas que los que yo mismo diseño a mano, sinceramente es una tecnología que pienso aprovechar para mis futuros proyectos.
- **Algoritmo de Dijkstra y mejoras.** Dijkstra, Dijkstra, amigo y enemigo desde segundo de carrera, pero cierto es que no es la primera ni la segunda vez que lo implemento, lo que sí es diferente es el cómo lo implemento, no para una red de nodos, si no para ¡una matriz!, una matriz de alturas, también llamada terreno y no solo Dijkstra en sí, ya que su uso para matrices de 500x500 (250.000 nodos) es demasiado lenta, si con distintas iteraciones sobre su código para por ejemplo tener un grado de precisión o quízas el más interesante, Dijkstra anidado consistente en ir subdividiendo el terreno en matrices una y otra vez como si de un fractal matemático se tratase, e ir calculando Dijkstra para cada uno de estos tableros.
- **Trabajar Curvas de Bezier**. Otro de los puntos incesantes del proyecto ha sido el de usar las ya conocidas curvar de bezier para suavizar los caminos y obtener unos giros más suaves y desde luego menos bruscos.

8.2 trabajo Futuro

Finalmente, no se puede terminar este trabajo sin hablar previamente del trabajo de futuro del proyecto, de las cosas que se pueden mejorar y las nuevas funciones que serían bastante interesantes de ver en futuras versiones de este proyecto.

La primera de estas sería la de mejorar la integración de la herramienta de recolección de datos con las redes sociales de los usuarios, pudiendo descargarse las fotos de una cuenta y analizarlas en busca de patrones que puedan permitir al sistema conocer mejor al usuario, sus gustos, los lugares que visita, de manera que (con su autorización previa, por supuesto) el sistema pueda recopilar esta información para brindarle un mejor soporte al usuario a la hora de generar los mundos que imagina o recuerda.

Otro de los aspectos a mejorar del sistema es el reconocimiento de palabras. El sistema actual que escucha lo que dice el usuario y lo reconvierte en un metalenguaje que luego el generador pueda entender, hace un análisis palabra por palabra en vez de intentar analizar el contexto de la frase, además el sistema tampoco es capaz de reconocer el texto de voz por lo que tampoco puede distinguir el estado de ánimo del usuario, ni mucho menos si está haciendo uso de sarcasmos por ejemplos, creo que esto es un problema que se podría solucionar aplicando algún modelo de red neuronal, pero es algo que habrá que probar con el tiempo.

El tercero de los puntos que considero interesante a mejorar es el sistema de ríos y caminos para que tengan en cuenta el resto de los ríos o caminos respectivamente antes de generarse de forma que si dos ríos se juntan se conviertan en uno más grande.

El cuarto de los aspectos que creo que sería más interesante de añadir a este proyecto sería el de que la base de datos del conocimiento pudiese generarse ella sola (a través de un script) y no solo que fuese capaz de generar ambos ficheros, sino que también fuese capaz de navegar por internet (a través de un webScrapper) para encontrar y modelar nuevos elementos que añadir al dataset, de manera que pudiera funcionar prácticamente de manera autónoma.

Enlazado con el punto anterior, otra de las futuras mejoras a realizar es la de ampliar la capacidad de reconocimiento de imágenes para que sea capaz de reconocer una mayor variedad de objetos, así como de poder incluso clasificar las lases de objetos en subclases más definidas como por ejemplo “Coche→Coche Deportivo→ Porsche 911” de esta manera, se multiplicarían exponencialmente las posibilidades de este sistema a la hora de recabar la información mucho más allá de los colores y estilos de determinados tipos de objetos.

Estos son algunos ejemplos concretos de aspectos del sistema que pueden puliste y mejorarse, pero no solo de mejoras va la cosa, también se pretende (cuando la situación epidemiológica lo permita) presentar este proyecto en congresos y eventos (Como la noche Europea de los Investigadores en Almería 2021) y redactar un artículo que expanda y profundice en varios de los temas expuestos en este documento y que bien profundizados podrían llegar a ser interesantes a nivel científico. Lo que está claro es que la línea de trabajo iniciada en este proyecto no es una que vaya a desaparecer en el corto plazo. Quizás no todas las partes del proyecto puedan recibir el mismo soporte, pero a mi parecer el proyecto tiene suficientes puntos interesantes como para justificar su soporte y mejora en un medio y largo plazo.

BIBLIOGRAFÍA

[1] 20minutos.es (25/03/2021) *¿Por qué son tan caros los videojuegos?* https://www.20minutos.es/videojuegos/noticia/videojuegos-caros-precio-185723/0/ (Visitado 25/03/2020).

[2] Videojuegosworld.com. (25/03/2021) *¿Cuánto costó hacer gta 5?* https://videojuegosworld.com/cuanto-costo-hacer-gta-5/

[3] Víctor López G. (26/03/2021) *'The Mandalorian': así funciona Stagecraft, la revolucionaria tecnología con la que se ha rodado la serie de Disney+* https://www.espinof.com/series-de-ficcion/the-mandalorian-asi-funciona-stagecraft-revolucionaria-tecnologia-que-se-ha-rodado-serie-disney

[4] Intelligent. (26/03/2021) *Adobe crea nuevas funciones importantes de Inteligencia Artificial en Photoshop.* https://itelligent.es/es/adobe-crea-nuevas-funciones-importantes-inteligencia-artificial-photoshop/

[5] QuasarDynamics (25/06/2021) Fotogrametría 3D. https://quasardynamics.com/fotogrametria-3d/

[6] Pixologic (30/06/2021) ZBrush at a Glance http://pixologic.com/features/about-zbrush.php

[7] BideoSare (25/06/2021) *El Libro de la Selva: 80 horas para renderizar un frame.* https://bideosare.com/libro-la-selva-80-horas-renderizacion-frame/

[8] Isra Fernández (01/07/2021) *Procedurally Generated Content: la revolución de los videojuegos es ahora* *https://www.xataka.com/videojuegos/procedurally-generated-content-la-revolucion-de-los-videojuegos-es-ahora-aunque-llevamos-40-anos-creandola*

[9] Mario Gómez (01/07/2021) *Un rascacielos procedural de 100 plantas: The Division 2 presenta su nuevo modo de juego* https://www.3djuegos.com/noticias-ver/207558/un-rascacielos-procedural-de-100-plantas-the-division-2/

[10] Json Org (26/06/2021) *Introducing JSON.* https://www.json.org/json-en.html

[11] Ivan de Souza (28/06/2021) *XML: ¿qué es y para qué sirve este lenguaje de marcado?* https://rockcontent.com/es/blog/que-es-xml/

[12] Aquasec (26/06/2021) Docker Containes vs. Virtual Machines https://www.aquasec.com/cloud-native-academy/docker-container/docker-containers-vs-virtual-machines/

[13] Visual Paradigm. (28/06/2021) Visual Paradigm Online. https://online.visual-paradigm.com/es/

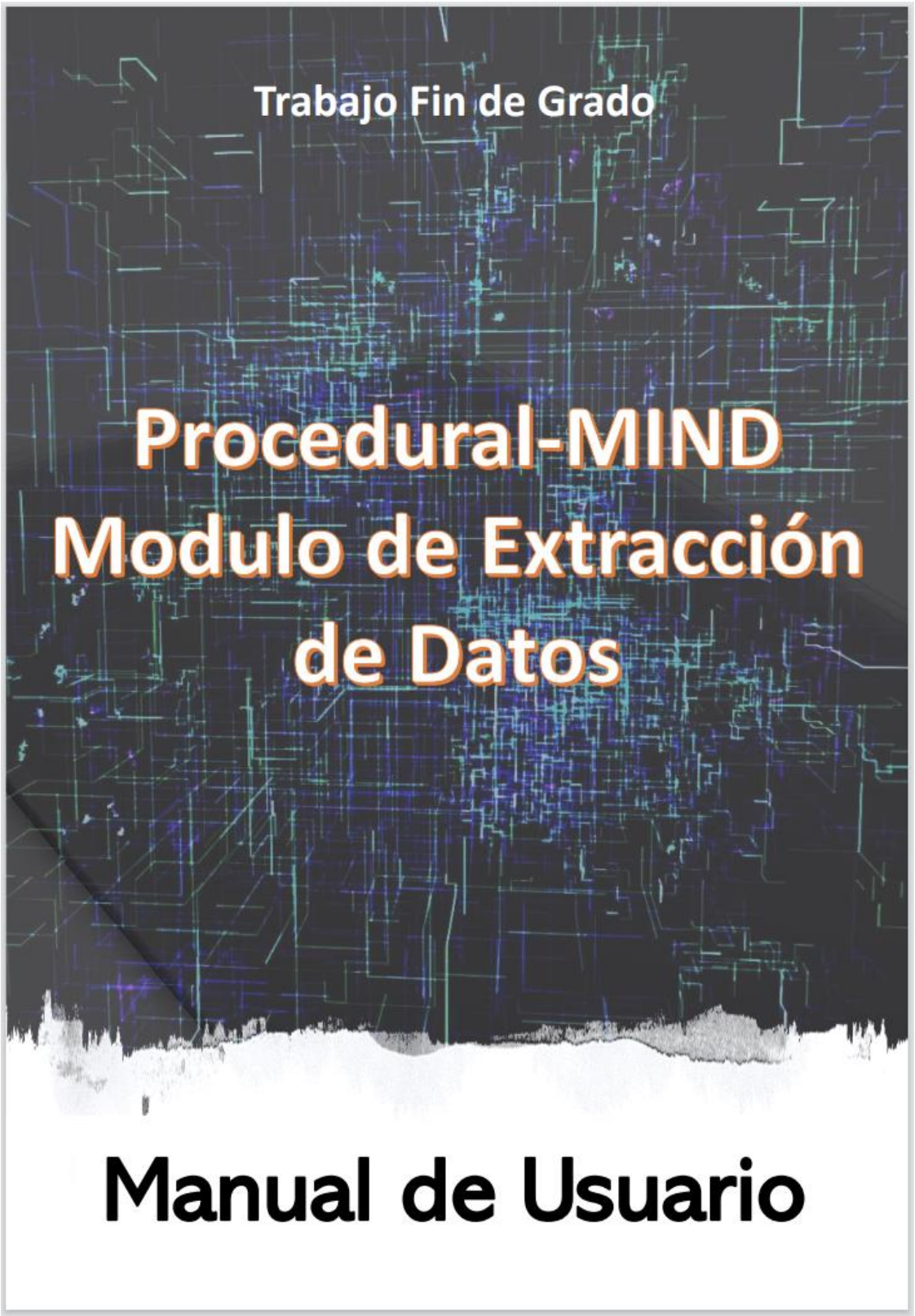
[14] Microsoft. (28/06/2021) *Visual Studio 2019.* https://visualstudio.microsoft.com/es/vs/

[15] MathWorks (08/07/2021) *Inverse kinematics (IK) algorithm design with MATLAB and Simulink* https://www.mathworks.com/discovery/inverse-kinematics.html

[16] Ken Lee (08/07/2021) *New Unity Input System: Getting Started* https://www.raywenderlich.com/9671886-new-unity-input-system-getting-started

ANEXO I. MANUALES DE USUARIO

Anexo I.1. Manual de usuario de la extracción de datos




Módulos del Sistema
Detect All
Detect
Get GPS
Speech
Export XML
Open Pictures
Importación de Imágenes
• Carga las fotos que se quieran analizar
Tratamiento de Imágenes
• Analiza las fotos en busca de objetos reconocibles
Localización GPS
• Obtiene la geolocalización actual del usuario
Reconocimiento de Voz
• Reconoce y muestra las palabras que dice el usuario
Exportar Datos
• Exporta los datos generados a un formato XML


Uso de la Herramienta
IMPORTA UNA O VARIAS IMÁGENES A TRAVÉS DEL BOTÓN "OPEN PICTURES"
ANALIZA LAS IMÁGENES UNA A UNA USANDO "DETECT" O TODAS JUNTAS CON "DETECT ALL"
OBTÉN TU GEOLOCALIZACIÓN USANDO EL BOTÓN "GET GPS"
PULSA EL BOTÓN "SPEECH" Y DEJA QUE EL SISTEMA CAPTURE TU VOZ. PARA FINALIZAR LA GRABACIÓN DEJA DE HABLAR.
XML
EXPORTA LOS DATOS XML CON EL BOTÓN EXPORTAR DATOS

Anexo I.2. Manual de usuario del módulo de generación (Unity3D)


Componentes del programa
Terrain Generator (Script)
Script
TerrainGenerator
Ter
Terrain (Terrain)
Instance On Start
Get Ubication On Start
Add Random Deformati
User Data Pos Generat
UserGenerationPos (Transform
Caminos
1
Camino 1
Topologies
4
Colina
Montaña
Llano
MontanaInversa
Topología Actual
Montaña
Variables de Crono-Localizacion
Año: 1766 DC
1766
worldTex
Obtener Ubiación Real
dimensiones
X -180
Y 90
Z 360
W -180
Longitud -54.85714°, Latitud -4.285714°
Cargar Fichero de Datos
Generar Superficie del Terreno
Generar Objetos
Generar Rios
Conf.Dijsktra(tam. l, rec X 81
Y 3
Z 9
Generar Caminos
forest
número de elemtos
600
Completar Terreno
Meta Objects (Script)
Add Component
Terreno a editar
Lugar de generación de los objetos de usuario
Lista de caminos
Lista de topologías
Topología actual
Variable de tiempo
Variable de localización
Cargar Fichero Datos Usuario
Generar Superficie del terreno
Generar Objetos del Usuario
Generar Ríos
Generar Caminos y Conf. Dijkstra
Completar terreno y num. Objetos


Pasos
Para la Utilización del Programa
1. Crear una escena y añadir un terreno
2. Crear un objeto vacío y añadir el componente TerrainGenerator
3. Asignar el terreno al componente
4. Crear / Importar la lista de formas del terreno y definir donde se generarán los datos de usuario
5. Cargar los datos generados por el usuario
6. Ajustar geolocalización y tiempo
7. Elegir una forma del paso 4 y generar el terreno
8. Generar los datos del usuario
9. Generar ríos del terrenos
10. Definir los caminos que se van a generar
11. Ajustar el algoritmo de Dijkstra y Generar los caminos del terreno
12. Seleccionar una estructura y completar la escena

ANEXO II. MODELO DE CUESTIONARIOS REALIZADO

Cuestionario sobre la usabilidad y la calidad técnica del software del TFG – Módulo de Recolección de Datos

Estimado usuario del programa, con el objetivo de obtener información relevante para el estudio del programa, le solicitamos 5 minutos de su tiempo para contestar, con la mayor sinceridad posible, el siguiente cuestionario anónimo.

Marque con una X la casilla correspondiente.

Sexo Hombre ☐ Mujer ☐ Otro ☐

Edad ☐

Titulación académica que posee			
Sin Titulación		Graduado Escolar	
Formación Profesional o Ciclo de Grado Medio		Bachillerato	
Ciclo de Grado Superior		Diplomatura	
Licenciatura o grado		Doctorado	
Otro (Indicar):			

Indique su experiencia en el manejo de ordenadores			
Experiencia Nula		Conocimientos básicos	
Nivel Avanzado		Nivel Experto	

Califique las siguientes afirmaciones del 1 al 5 según su grado de acuerdo con las mismas, siendo 1 "muy en desacuerdo", 2 "algo en desacuerdo", 3 "ni en acuerdo ni en desacuerdo" 4, "algo de acuerdo" y 5 "muy de acuerdo".

Presentación	**1**	**2**	**3**	**4**	**5**
La estética general del programa es adecuada					
La interfaz es clara y resulta sencilla de entender					
El manual es fácil de entender y suficientemente comprensible					

Interactividad y Manejo					
El programa ha resultado fácil de manejar					
El programa es intuitivo					
El sistema responde de forma adecuada a mis interacciones					
Los botones han respondido en tiempo y forma adecuadamente					

Contenido					
La cantidad y variedad de objetos que el sistema ha podido reconocer ha sido suficiente					
Los datos obtenidos del programa han sido suficientemente precisos					

Funcionamiento y eficacia					
No han surgido errores que han obligado a detener la ejecución el programa					
La velocidad de carga de los elementos del programa ha sido suficientemente rápida					

Compromiso					
El programa ha respondido a sus expectativas					
Recomendaría el programa a otros usuarios					
Volvería a utilizar el programa si tuviera oportunidad					

Valoración general del programa	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Comentarios Abiertos

Indique lo que considerarías más interesante del programa

Indique lo que considerarías menos interesante del programa

¿Qué contenidos has echado en falta en el programa?

Indique cualquier sugerencia cambio o modificación que desearía que se llevase a cabo en el programa

Cualquier comentario que no haya tenido cabida en las preguntas anteriores

Cuestionario sobre la usabilidad y la calidad técnica del software del TFG – Módulo de Generación de Objetos

Estimado usuario del programa, con el objetivo de obtener información relevante para el estudio del programa, le solicitamos 5 minutos de su tiempo para contestar, con la mayor sinceridad posible, el siguiente cuestionario anónimo.

Marque con una X la casilla correspondiente.

Sexo Hombre ☐ Mujer ☐ Otro ☐

Edad ☐

Titulación académica que posee			
Sin Titulación		Graduado Escolar	
Formación Profesional o Ciclo de Grado Medio		Bachillerato	
Ciclo de Grado Superior		Diplomatura	
Licenciatura o grado		Doctorado	
Otro (Indicar):			

Indique su experiencia en el manejo de ordenadores			
Experiencia Nula		Conocimientos básicos	
Nivel Avanzado		Nivel Experto	

¿Cuántos años de experiencia tiene usando Unity3D?			
Menos de 2 Años		Entre 2 y 5 Años	
Entre 5 y 10 Años		Más de 10 Años	

Califique las siguientes afirmaciones del 1 al 5 según su grado de acuerdo con las mismas, siendo 1 "muy en desacuerdo", 2 "algo en desacuerdo", 3 "ni en acuerdo ni en desacuerdo" 4, "algo de acuerdo" y 5 "muy de acuerdo".

Presentación	**1**	**2**	**3**	**4**	**5**
La estética general del módulo es adecuada					
La interfaz es clara y resulta sencilla de entender					
El manual es fácil de entender y suficientemente comprensible					
La estética del programa se integra correctamente en la Unity3D					

Interactividad y Manejo					
El programa ha resultado fácil de manejar					
El programa es intuitivo					
El sistema responde de forma adecuada a mis interacciones					
Los botones han respondido en tiempo y forma adecuadamente					

Contenido					
Los objetos que se deben generar se han colocado adecuadamente en el escenario					
(En caso de haberlos) El programa ha generado los ríos y caminos de una forma adecuada					
La generación de objetos según ubicación funciona correctamente					

Funcionamiento y eficacia

No han surgido errores que han obligado a detener la ejecución el programa					
La velocidad de carga de los elementos del programa ha sido suficientemente rápida					
Una vez generados los objetos he podido interactuar con ellos de la manera adecuada					
Una vez generados los objetos he podido editarlos de manera adecuada					

Compromiso

El programa ha respondido a sus expectativas					
Recomendaría el programa a otros usuarios					
Volvería a utilizar el programa si tuviera oportunidad					

Valoración general del programa **1 2 3 4 5 6 7 8 9 10**

Comentarios Abiertos

Indique lo que considerarías más interesante del programa

Indique lo que considerarías menos interesante del programa

¿Qué contenidos has echado en falta en el programa?

Indique cualquier sugerencia cambio o modificación que desearía que se llevase a cabo en el programa

Cualquier comentario que no haya tenido cabida en las preguntas anteriores

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

Uno de los problemas a los que tiene que enfrentarse la industria del videojuego y del entretenimiento hoy en día es a los altos costes del desarrollo de escenarios en 3D derivado del gran tiempo que tienen que dedicar los desarrolladores a esta tarea. Este trabajo busca agilizar esta tarea generando escenarios de forma procedural que, además, son a la medida de cada persona. El sistema utiliza una basta base de conocimiento de objetos que podrá ser usada tanto por los desarrolladores como por personas menos puestas en la materia (artistas, directivos, clientes, etc.) para generar y recolectar a través de la voz lo que el usuario desee crear. Además, la voz no es la única fuente de datos del sistema, puesto que este también hace uso del reconocimiento de imágenes (para extraer características de los objetos que la componen) o de la geolocalización del usuario. El programa tiene la capacidad de definir un entorno espaciotemporal para los objetos que se generan a partir de un objeto base. De esta manera en función de la geolocalización (por defecto la ubicación del usuario) y la época elegida, los objetos que se generen serán distintos, a pesar incluso de que sean las mismas peticiones. Por ejemplo, si se pide un coche y se marca como época el sigo XVIII, lo que se regenerará será un vehículo a vapor. Finalmente, el sistema implementa también algoritmos de generación de ríos, terrenos, caminos y objetos.

Conceptos clave: Generación personalizada, extracción de datos, reconocimiento de imágenes y voz, Entornos 3D, Dijkstra, Bézier, Inteligencia Artificial.

One of the issues that the video game and entertainment industry must face nowadays is the high costs of developing 3D environments derived from the huge time that developers must spend in this task. This work seeks to streamline this task by generating procedural environments that are also tailored to each person. The system uses a vast knowledge base of objects that can be used by both, developers and people less knowledgeable (artists, managers, clients, etc.), to generate and collect through the voice what the user wants to create. In addition, voice is not the only source of data in the system, since it also makes use of image recognition (to extract characteristics of the objects that compose it) or of the geolocation of the user. The program can define a space-time environment for objects that are generated from a base object. In this way, depending on the geolocation (by default the user's location) and the chosen time, the objects generated will be different, even though they are the same requests. For example, if a car is ordered and the 18th century is marked as a period, what will be regenerated will be a steam vehicle. Finally, the system implements also algorithms for the generation of rivers, terrain, roads, and objects..

Keywords: Custom generation, data extraction, image and speech recognition, 3D environments, Dijkstra, Bezier, Artificial Intelligence.