



Universidad  
Carlos III de Madrid

# TRABAJO FIN DE GRADO

MARCO A. PAJARES 2012/2013

GRADO ING. INFORMÁTICA

TUTOR: YAGO SÁEZ

*Desarrollo de un videojuego completo con todas sus etapas, ubicado en el campus de la Universidad Carlos III en Leganés.*

# TABLA DE CONTENIDO

## AGRADECIMIENTOS

*En primer lugar quiero agradecer a mi familia todo el apoyo que me han dado, a mis padres por la posibilidad de estudiar y por el apoyo que han sido todos estos años*

*A mi primo Javier por su ayuda con todo el entorno UDK*

*A mis compañeros de grupo de la asignatura “informática gráfica” Carlos, Jaime, Ángel y Álvaro ya que sin ellos, este proyecto no hubiera sido posible*

*A mi tutor, Yago, por hacer esto real y vivir con la misma intensidad que yo cada novedad*

*A mis amigos y compañeros Oscar y Sergio, por tomarse la molestia de revisar y comentar una y otra vez este documento*

*Y por último a Miriam, Álvaro, Miguel y Elisabeth, por apoyarme en los momentos más duros, por cuidar de mí, y por estar a mi lado en cada etapa de este proyecto.*

# TABLA DE CONTENIDO

## RESUMEN

El objetivo de este proyecto es analizar, comprender y realizar cada etapa del proceso de creación de un videojuego usando el motor *UDK*, el producto resultante “*Uc3mPaint*” es un juego en primera persona que ubica en el campus de Leganés, una partida de *paintball*. El juego tiene una finalidad exclusivamente lúdica, si bien puede servir de base para futuras ampliaciones u otros tipos de juego que se deseen desarrollar aprovechando el escenario.

Se ha realizado un análisis previo de las características del entorno de desarrollo, así como de todas las herramientas y procesos involucrados en el desarrollo de un videojuego.

El juego ha sido desarrollado en un entorno multiplataforma, lo que permite muchos canales de difusión y un número elevado de potenciales usuarios.

# TABLA DE CONTENIDO

## Contenido

|  |    |
|--|----|
| Índice de tablas                         | 6  |
| Índice de ilustraciones                  | 8  |
| 1. Introducción y objetivos              | 12 |
| Introducción                             | 12 |
| Objetivos y motivación                   | 12 |
| Estructura de la memoria                 | 13 |
| 2. Estado de la arte                     | 14 |
| Introducción                             | 14 |
| Planteamiento del problema               | 14 |
| Análisis del estado del arte             | 14 |
| Diseño 3d                                | 14 |
| Modelado                                 | 15 |
| Los videojuegos                          | 19 |
| Motor de juego                           | 20 |
| Creación de menús                        | 25 |
| Edición de video                         | 25 |
| Tratamiento de imágenes                  | 25 |
| Requisitos y restricciones               | 25 |
| Casos de uso                             | 26 |
| Requisitos software                      | 32 |
| Marco regulador                          | 37 |
| 3. Diseño y desarrollo de la solución    | 38 |
| Introducción                             | 38 |
| Diseño de la solución inicial            | 38 |
| Limitaciones iniciales                   | 38 |
| Ideas básicas: ¿Cómo será el videojuego? | 38 |
| Arquitectura de la aplicación            | 39 |
| Software, técnicas y métodos planteados  | 40 |

# TABLA DE CONTENIDO

|   |    |
|---|----|
| Desarrollo de alternativas de diseño                          | 44 |
| Implementación  | 46 |
| Gestión de datos  | 46 |
| Creación de modelos arquitectónicos                           | 46 |
| Creación de personajes  | 64 |
| Creación de arma  | 67 |
| Generación de cinemáticas                                     | 68 |
| Generación de menús interactivos                              | 69 |
| Unreal development kit: entorno de desarrollo                 | 72 |
| 4. Pruebas, resultados y evaluación                           | 87 |
| Introducción  | 87 |
| Explicación del método de evaluación                          | 87 |
| análisis de resultados  | 89 |
| Mejora de la propuesta respecto a otras posibles alternativas | 90 |
| 5. Planificación y presupuesto                                | 91 |
| Introducción  | 91 |
| Presupuesto   | 91 |
| Personal  | 91 |
| Material  | 91 |
| Costes directos   | 92 |
| Totales   | 92 |
| Planificación   | 93 |
| Estimada  | 93 |
| Real  | 96 |
| 6. Conclusiones   | 98 |
| Introducción  | 98 |
| Objetivos cumplidos   | 98 |
| Problemas encontrados   | 98 |
| Líneas futuras de trabajo                                     | 98 |
| Conclusiones  | 98 |

## TABLA DE CONTENIDO

Bibliografía \_\_\_\_\_ 99

# TABLA DE CONTENIDO

## Índice de tablas

|  |    |
|--|----|
| Tabla 1: principales herramientas de modelaje .....    | 16 |
| Tabla 2: Plantilla Casos de Uso .....                  | 26 |
| Tabla 3:CU01 .....                                     | 27 |
| Tabla 4:CU02 .....                                     | 27 |
| Tabla 5:CU03 .....                                     | 28 |
| Tabla 6:CU04 .....                                     | 28 |
| Tabla 7:CU05 .....                                     | 28 |
| Tabla 8:CU06 .....                                     | 29 |
| Tabla 9:CU07 .....                                     | 30 |
| Tabla 10:CU08.....                                     | 30 |
| Tabla 11:CU09.....                                     | 30 |
| Tabla 12:CU10.....                                     | 30 |
| Tabla 13:CU11.....                                     | 31 |
| Tabla 14:CU12.....                                     | 31 |
| Tabla 15:CU13.....                                     | 31 |
| Tabla 16:CU14.....                                     | 32 |
| Tabla 17:CU15.....                                     | 32 |
| Tabla 18: Plantilla de requisitos.....                 | 32 |
| Tabla 19: RSF-01.....                                  | 33 |
| Tabla 20: RSF-02.....                                  | 33 |
| Tabla 21: RSF-03.....                                  | 34 |
| Tabla 22: RSF-04.....                                  | 34 |
| Tabla 23: RSF-05.....                                  | 34 |
| Tabla 24: RSF-06.....                                  | 35 |
| Tabla 25: RSF-07.....                                  | 35 |
| Tabla 26: RSF-08.....                                  | 35 |
| Tabla 27: RSF-09.....                                  | 35 |
| Tabla 28: RSF-10.....                                  | 36 |
| Tabla 29: RSNF-01 .....                                | 36 |
| Tabla 30: RSNF-02 .....                                | 36 |
| Tabla 31: RSNF-03 .....                                | 37 |
| Tabla 32: Comparativa <i>UDK</i> vs <i>Unity</i> ..... | 43 |
| Tabla 33: cuestionario de evaluación .....             | 88 |

# TABLA DE CONTENIDO

|  |    |
|--|----|
| Tabla 34: Matriz de resultados .....         | 89 |
| Tabla 35: porcentaje de resultados.....      | 90 |
| Tabla 36: presupuesto de personal .....      | 91 |
| Tabla 37: presupuesto de hardware.....       | 92 |
| Tabla 38: presupuesto de software.....       | 92 |
| Tabla 39: costes directos.....               | 92 |
| Tabla 40: costes totales .....               | 93 |
| Tabla 41: tareas planificación inicial ..... | 94 |
| Tabla 42: Planificación estimada .....       | 95 |
| Tabla 43: tareas planificación inicial ..... | 96 |
| Tabla 44: planificación real .....           | 97 |

# TABLA DE CONTENIDO

## Índice de ilustraciones

|   |    |
|---|----|
| Ilustración 1: representación del <i>mapping</i> .....  | 15 |
| Ilustración 2: representación del texturizado .....   | 16 |
| Ilustración 3: logotipo de Blender .....  | 16 |
| Ilustración 4: logotipo de Autodesk .....   | 17 |
| Ilustración 5: interfaz de 3ds Max.....   | 17 |
| Ilustración 6: logotipo de SketchUp .....   | 17 |
| Ilustración 7: interfaz SketchUp.....   | 18 |
| Ilustración 8: logo de Mixamo.....  | 18 |
| Ilustración 9: componentes de un entorno de un motor gráfico.....                             | 21 |
| Ilustración 10: logo de UDK .....   | 22 |
| Ilustración 11: interfaz del entorno UDK.....   | 23 |
| Ilustración 12: logo CryEngine .....  | 23 |
| Ilustración 13: logo Unity.....   | 24 |
| Ilustración 14: logo ScaleForm .....  | 25 |
| Ilustración 15: logo Windows Movie Maker .....  | 25 |
| Ilustración 16: Casos de Uso del menú principal.....  | 27 |
| Ilustración 17: Casos de Uso del menú opciones.....   | 28 |
| Ilustración 18: Casos de Uso durante la partida .....   | 29 |
| Ilustración 19: Casos de Uso durante el menú de pausa.....                                    | 31 |
| Ilustración 20: Código sistema PEGI.....  | 37 |
| Ilustración 21: Imagen del GTA .....  | 38 |
| Ilustración 22: imagen del Counter Strike .....   | 38 |
| Ilustración 23: <i>mockups</i> del diseño .....   | 39 |
| Ilustración 24: Esquema de la arquitectura .....  | 39 |
| Ilustración 25: Esquema de aplicaciones .....   | 41 |
| Ilustración 26: Distribución del mercado global.....  | 42 |
| Ilustración 27: conjunto de imágenes recopiladas para el diseño .....                         | 44 |
| Ilustración 28: mapas del campus de Leganés.....  | 44 |
| Ilustración 29: Directorio del proyecto .....   | 46 |
| Ilustración 30: algunos modelos extraídos de la biblioteca online de <i>SketchUp</i> .....    | 47 |
| Ilustración 31: modelo biblioteca exterior e interior .....                                   | 48 |
| Ilustración 32: comparación mapa <i>Google Maps</i> vs modelo hecho con <i>SketchUp</i> ..... | 48 |
| Ilustración 33: edificio mixto (1 y 4) en <i>SketchUp</i> .....                               | 49 |

# TABLA DE CONTENIDO

|   |    |
|---|----|
| Ilustración 34: imagen del edicio Betancourt .....  | 50 |
| Ilustración 35: Diseño fachada del edificio Betancourt.....                                 | 50 |
| Ilustración 36 Imagen del edifico Juan Benet.....   | 50 |
| Ilustración 37: Diseño del edificio Juan Benet .....  | 51 |
| Ilustración 38: Diseño del edificio Betancourt (II) .....                                   | 51 |
| Ilustración 39: Modelo edificio Betancourt.....   | 52 |
| Ilustración 40: Aplicación UVW Mapping.....   | 52 |
| Ilustración 41: Aplicación de texturas al edificio Betancourt.....                          | 53 |
| Ilustración 42: Modelo edificio Sabatini .....  | 53 |
| Ilustración 43: Modelo edificio Sabatini (II) .....   | 54 |
| Ilustración 45: Módulos del edificio Sabatini .....   | 54 |
| Ilustración 46: Fachada del edificio Sabatini .....   | 55 |
| Ilustración 47: Comparación de ventanas de la fachada principal vs original .....           | 55 |
| Ilustración 48: Comparación esquinas vs original .....                                      | 56 |
| Ilustración 49: Comparación de la entrada y el arco superior .....                          | 57 |
| Ilustración 50: Comparación del patio interior del edificio Sabatini .....                  | 58 |
| Ilustración 51: Patio del modelo del edificio Sabatini .....                                | 58 |
| Ilustración 52: Comparación de las estatuas del patio del edificio Sabatini .....           | 59 |
| Ilustración 53: vista aérea de la zona verde del campus.....                                | 59 |
| Ilustración 54: modelos de la ciudad de Leganés de la biblioteca online de SketchUp.....    | 60 |
| Ilustración 55: diversos elementos contenidos en el paquete descargable <i>citi</i> .....   | 60 |
| Ilustración 56: Eliminación de elementos sobrantes en SketchUp .....                        | 61 |
| Ilustración 57: fallo en la importación.....  | 61 |
| Ilustración 58: modelo importado a Blender .....  | 61 |
| Ilustración 59: modelo con el fallo solucionado.....  | 62 |
| Ilustración 60: interfaz ImRe.....  | 62 |
| Ilustración 61: conversión de formato y dimensiones.....                                    | 62 |
| Ilustración 62: directorio del paquete uc3mPaint .....                                      | 63 |
| Ilustración 63: creación de material .....  | 63 |
| Ilustración 64: asignación de materiales al modelo .....                                    | 64 |
| Ilustración 65: editor de modelos 3D Mixamo.....  | 65 |
| Ilustración 66: disociación del modelo de UT3 y posterior fusión con el modelo creado ..... | 65 |
| Ilustración 67: modelo mal riggeado y bien riggeado .....                                   | 66 |
| Ilustración 68: diversos movimientos que puede realizar el personaje .....                  | 67 |
| Ilustración 69: modelos de los enemigos .....   | 67 |
| Ilustración 70: arma del juego .....  | 68 |

# TABLA DE CONTENIDO

|   |    |
|---|----|
| Ilustración 71: editor de partículas .....                                    | 68 |
| Ilustración 72: editor de secuencias Matinée .....                            | 68 |
| Ilustración 73: secuencia de introducción .....                               | 69 |
| Ilustración 74: logo UC3M.....  | 70 |
| Ilustración 75: identificadores menú principal y menú pausa.....              | 70 |
| Ilustración 76: botón de acción normal y con <i>hover</i> .....               | 70 |
| Ilustración 77: posicion del <i>HUD</i> durante la partida.....               | 71 |
| Ilustración 78: elementos del <i>HUD</i> .....                                | 71 |
| Ilustración 79: asociación de eventos en UDK .....                            | 72 |
| Ilustración 80: código en <i>UDK usando Kismet</i> .....                      | 72 |
| Ilustración 81: menú con sol, lluvia y de noche.....                          | 72 |
| Ilustración 82: directorio <i>UDK</i> .....                                   | 73 |
| Ilustración 83: subdirectorio <i>/src/</i> .....                              | 73 |
| Ilustración 84: nuevas carpetas creadas.....                                  | 73 |
| Ilustración 85: edicion del archivo Defaultengine.ini .....                   | 74 |
| Ilustración 86: herencia de clases del proyecto.....                          | 75 |
| Ilustración 87: herencia de clases .....                                      | 75 |
| Ilustración 88: interfaz y visualización por defecto .....                    | 76 |
| Ilustración 89: asignación de tipo de juego al mapa .....                     | 76 |
| Ilustración 90: propiedades por defecto en uc3mPaint_1p_Pawn .....            | 77 |
| Ilustración 91: cambio cámara .....   | 78 |
| Ilustración 92: propiedades por defecto en el archivo uc3mPaint_1p_Game ..... | 78 |
| Ilustración 93: definicion del estado GoToFinalTarget .....                   | 79 |
| Ilustración 94: definición del spawner .....                                  | 79 |
| Ilustración 95: propiedades de la clase paintgun .....                        | 80 |
| Ilustración 96: codigo de asociacion del menu flash.....                      | 81 |
| Ilustración 97: Kismet del mapa .....   | 82 |
| Ilustración 98: Kismet de la parte del menú flash.....                        | 82 |
| Ilustración 99: kismet de la parte de municion .....                          | 83 |
| Ilustración 100: accionador spawner y spawner .....                           | 83 |
| Ilustración 101: kismet spawner .....   | 83 |
| Ilustración 102: matine y kismet de la transicion dia/noche .....             | 84 |
| Ilustración 103: imágenes de la transición día/noche.....                     | 84 |
| Ilustración 104: efectos meteorológicos.....                                  | 85 |
| Ilustración 105: Kismet dia/noche y meteorología.....                         | 85 |
| Ilustración 106 vista en wireframe de los cilindros de lluvia.....            | 86 |

## TABLA DE CONTENIDO

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## 1. Introducción y objetivos

### INTRODUCCIÓN

El desarrollo de videojuegos es un campo que engloba múltiples aspectos de las artes tanto digitales como no digitales.

Por un lado tenemos la parte no-digital como es la historia, los primeros bocetos de los personajes y objetos, selección de música, y de herramientas de diseño

Por otra parte, se encuentra la parte puramente informática, que engloba:

- diseño de los elementos del juego (personajes, armas, objetos, edificios, etc.)
- diseño de la música y de los distintos sonidos que reproduce el juego.
- Info-arquitectura, elaboración de mapas para establecer la posición de los elementos en el entorno.
- Cinemáticas del videojuego
- Movimientos de los distintos personajes y objetos y sus interacciones.
- IA de los personajes para determinar su comportamiento.
- Elaboración de texturas que se aplicaran a los elementos del juego
- Código necesario para crear la arquitectura base del motor de juego y establecer conexiones.

A lo largo del presente documento se hará un recorrido más en profundidad de cada uno de estos puntos y como se han enfocado en el proyecto.

### OBJETIVOS Y MOTIVACIÓN

Llevo desde los 14 años participando en el desarrollo de MODs (modificaciones no comerciales a nivel de mapas, personajes, armas e historia de videojuegos comerciales para extender su funcionalidad y/o jugabilidad) asique **realizar un videojuego con todas sus etapas**, era una meta que siempre había querido afrontar y mientras cursaba la asignatura “Informática gráfica” surgió la posibilidad de hacer de este objetivo personal, el trabajo de fin de grado y no lo dudé.

El objetivo del proyecto es la realización de un videojuego ubicado en el campus de Leganés de la Universidad Carlos III de Madrid. El proyecto es un fin en sí mismo ya que el producto final es un videojuego que puede ser utilizado para fines diversos. Al ser concebido como un “sandbox” permite que el juego pueda usarse como:

- publicidad para la propia universidad y para la ciudad de Leganés.
- Entretenimiento para los alumnos
- Base para aplicaciones de geo localización interactiva

# DESARROLLO DE UN VIDEOJUEGO CON UDK

- Base para futuros juegos ubicados en Leganés.

Los objetivos que se persiguen con la elaboración de este trabajo fin de grado son:

- Estudiar las etapas de desarrollo de un videojuego y las posibilidades existentes ahora mismo en el mercado para la creación de un videojuego completo.
- **Desarrollar un juego desde cero, ubicado en el campus de la universidad Carlos III de Leganés.**
- La aplicación desarrollada tendrá un fin lúdico, que sirva como entretenimiento a estudiantes
- Tiene un fin, experimental, como base para futuros proyectos que deseen ampliar y explotar todas las posibilidades que el motor escogido ofrece.
- El último objetivo es que el producto resultante, sea un producto final, sin necesidad de otros elementos.

## ESTRUCTURA DE LA MEMORIA

A continuación se enumeran los capítulos contenidos en el documento como una breve descripción de cada uno de ellos.

### 2-Estado del arte

Explicación del problema que se quiere tratar, indicando trabajos ya realizados anteriormente, estado actual de la tecnología, requisitos y restricciones del proyecto y las normativas que se han seguido y/o aplicado durante su desarrollo.

### 3-Diseño y desarrollo de la solución

Explicación detallada y justificada de cada una de las decisiones que se han tomado, así como una descripción concisa de la implementación.

### 4-Pruebas, resultados y evaluación

Descripción de cada una de las pruebas a las que se someterá el producto y posterior análisis de los resultados.

### 5-Planificación y presupuesto

Detalle de las distintas fases de las que consta el proyecto, identificando las tareas y temporización.

### 6-Conclusiones

Exposición de la consecución o no de los objetivos planteados y su validación durante el proceso de evaluación.

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## 2. Estado de la arte

### INTRODUCCIÓN

En esta sección se realiza un breve resumen de la industria del videojuego así como un análisis de su situación actual, investigar su marco tecnológico para ayudar al lector a valorar el trabajo realizado y apreciar claramente los aspectos más destacables del proyecto.

### PLANTEAMIENTO DEL PROBLEMA

El objetivo del proyecto es la realización de un videojuego teniendo en cuenta cada una de sus etapas, este proyecto ya se ha abordado con anterioridad en innumerables ocasiones [1], [2] o [3] pero siempre en escenarios ya creados, o entornos abiertos. Esta vez el propósito es crear un juego de paintball ubicado en un entorno urbano. Los puntos que diferenciaran este proyecto del resto serán:

- Ubicado en un entorno urbano: los distintos videos y proyectos que actualmente existen o se están desarrollando, hacen uso de mapas existentes o mapas abiertos sin edificios ni tramas
- Modelado del campus de Leganés: no existe ningún juego en el que aparezca ni la Universidad Carlos III ni la ciudad de Leganés.
- Clima y sonidos: el juego permite cambios climatológicos que afectan a la visibilidad del jugador
- Soporte multi-jugador: como línea futura, se contempla el uso de implementar un multijugador
- Kinect/Wiimote: permite controlar el personaje haciendo uso de este tipo de mandos, usando scripts externos al entorno de desarrollo.
- Base para distintos de juego: al crear el mapa permite ser utilizado en un futuro para desarrollar distintos tipos de juego.

### ANÁLISIS DEL ESTADO DEL ARTE

#### DISEÑO 3D

El diseño 3D es el campo de la informática dedicado a la creación de modelos tridimensionales generados por ordenador. Es uno de los campos con mayor evolución en los últimos años, a la par que evoluciona la tecnología informática, el diseño 3D alcanza mayor definición y capacidad de procesamiento, lo que da lugar a resultados cada vez más realistas.

El diseño tridimensional cada vez se aplica en más ámbitos, y ya no está únicamente restringido a videojuegos y cine. La medicina lo usa para recrear modelos de huesos que ayuden en las operaciones, en sistemas de vigilancia se utiliza para crear gráficos en tiempo real que analizan actividades forestales, marinas, o urbanas, en el campo del derecho, es una prueba válida para recreaciones de delitos etc.

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## MODELADO

El modelado es la fase en la que se construye y genera el modelo en sí, representa una de las fases vitales del desarrollo de videojuegos y la que más tiempo y recursos requiere.

Para modelar existen multitud de herramientas en el mercado, aunque la que domina el mercado es, sin lugar a dudas, Autodesk *3ds Max*, una solución completa de modelado, animación, renderización y composición en 3D para creadores de juegos, cine y gráficos en movimiento.

La finalidad del modelo que se busca generar no requiere un nivel técnico muy elevado, ya que para el motor de juegos, cuanta mayor definición tenga el modelo, el rendimiento será menor y mucho nivel de detalle normalmente pasa desapercibido para el jugador habitual.

Uno de los aspectos importantes a considerar es el del texturizado, fase del modelado que dota al objeto de una apariencia más realista mediante técnicas como puede ser la oclusión ambiental, el índice de refracción o la rugosidad de la superficie. La técnica de texturizado se divide en dos etapas: la primera, el mapeado, se crea una malla que envuelve el objeto tridimensional, para después en la siguiente etapa, el texturizado, aplicar la textura que desea proyectarse sobre el objeto.

### Técnicas de modelado: mapeado

El mapeado o *mapping* es la técnica del modelado que permite envolver al objeto tridimensional con una figura geométrica sobre la que se aplica una textura que es proyectada sobre el objeto 3D en la etapa de texturizado

La técnica más común y simple en términos computacionales, es la llamada *UVW Mapping* la cual, una vez seleccionado el objeto tridimensional, permite crear una representación bidimensional de la superficie del objeto tridimensional para posteriormente aplicar la textura.

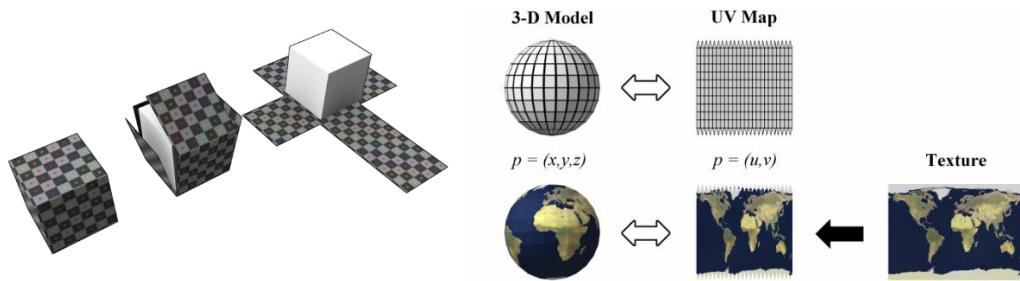


Ilustración 1: representación del *mapping*

### Técnicas de modelado: texturizado

Una vez realizado el mapeado hay que aplicar la textura sobre la representación bidimensional del objeto tridimensional. Es la parte que dota de realidad al objeto, y aunque las posibilidades y opciones de configuración pueden ser ilimitadas, hay que tener en cuenta que esta configuración puede suponer mucho trabajo computacional y afectar gravemente al rendimiento del juego.

TRABAJO FIN DE GRADO

## DESARROLLO DE UN VIDEOJUEGO CON UDK

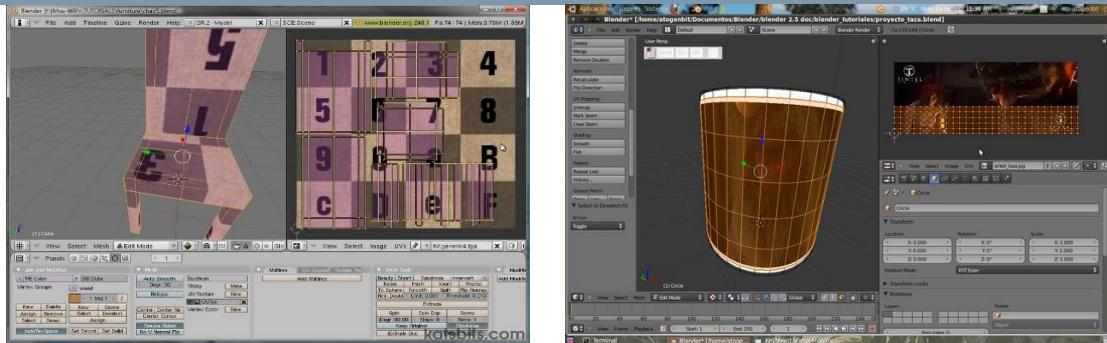


Ilustración 2: representación del texturizado

A continuación se muestra un listado de las herramientas de modelado 3D consideradas. Existen multitud de ellas que cumplen las condiciones para el proyecto planteado, pero aquí solo se va a proceder a listar las que más se ajustan por uso, popularidad, recursos en la red, ejemplos, comunidad, resultados, y curva de aprendizaje.

| Herramienta       | Desarrollador     | Página Web  |
|-------------------|-------------------|---|
| <b>3ds Max</b>    | Autodesk          | <a href="http://www.autodesk.es/products/autodesk-3ds-max/overview">http://www.autodesk.es/products/autodesk-3ds-max/overview</a> |
| <b>Blender 3D</b> | Fundación Blender | <a href="http://www.Blender.org/">http://www.Blender.org/</a>   |
| <b>SketchUp</b>   | Trimble           | <a href="http://SketchUp.com/">http://SketchUp.com/</a>   |

Tabla 1: principales herramientas de modelaje

La elección entre una u otra en principio no es una decisión crítica, ya que ambas poseen formatos de exportación/importación más o menos compatibles entre sí y todas ellas permiten la aplicación de mapeado y texturizado de manera nativa. Ambas herramientas ofrecen, al nivel que vamos a darle uso, la misma funcionalidad.

De hecho, para la realización del proyecto, se han tenido que usar las tres herramientas, debido a incompatibilidades, fallos y falta de conocimientos en profundidad.

Una vez analizadas ambas herramientas, se puede concluir que son compatibles con los requisitos de importación del motor de juego (*UDK*), si bien en algunos casos, será necesario el uso de plugins externos para asegurar la compatibilidad entre aplicaciones.

### Modelado: BLENDER



Ilustración 3: logotipo de Blender

La principal ventaja de *Blender* frente a su competidora *3ds Max* es que es gratuito, bajo licencia *Open Source* y con una comunidad inmensa de desarrolladores y multitud de ejemplos y recursos en línea.

TRABAJO FIN DE GRADO

# DESARROLLO DE UN VIDEOJUEGO CON UDK

La aplicación se encuentra disponible en su página web para las tres principales plataformas (*Windows*, Mac OS y Linux) y al estar en constante evolución, sale una nueva versión cada 3-4 meses.

Sin duda los resultados que se obtienen con esta herramienta son espectaculares, si bien en ocasiones tiene fallos al no reconocer bien, texturas o no aplicarlas en el sitio correcto, lo que ha obligado a utilizar *3ds Max* para retocar este tipo de fallos.

## Modelado: Autodesk 3ds Max



Ilustración 4: logotipo de Autodesk

Sin duda alguna, *3ds Max* constituye el estándar de facto para la realización de modelos tridimensionales, sus muchos años de experiencia y su uso en películas, anuncios, juegos y diseños hace que sea el ejemplo a seguir para el resto de competidores, la curva de aprendizaje de este software es mucho más elevada que la del programa anterior, si bien sus resultados son muy superiores.

En cuanto a licencia, *3ds* tiene una versión de prueba de 30 días, una versión libre para estudiantes para uso no comercial y para el resto es necesario una licencia de pago de mínimo 3.900€

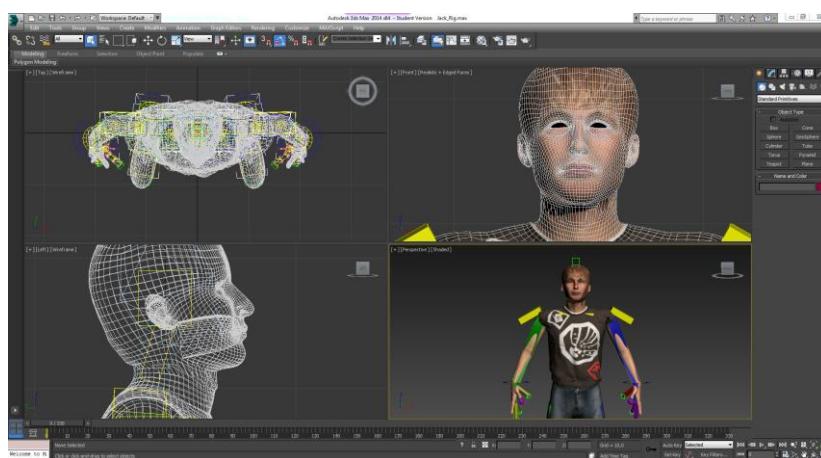


Ilustración 5: interfaz de 3ds Max

## Modelado 3d: SketchUp



Ilustración 6: logotipo de SketchUp

Es un programa de diseño gráfico y modelado 3D basado en caras. Es el que obtiene los resultados menos realistas de los tres, pero tiene a su favor que la curva de aprendizaje es prácticamente inexistente, está orientado a la info-arquitectura, diseño industrial e ingeniería civil. Su total integración con

*Google Maps*, permite a partir de estos mapas, crear edificios usando las fotos de *Google*, lo que dota de mayor realismo los objetos creados y permiten exportarse para mejorarlas en cualquiera de las aplicaciones anteriormente mencionadas.

TRABAJO FIN DE GRADO

## DESARROLLO DE UN VIDEOJUEGO CON UDK



Ilustración 7: interfaz SketchUp

### Modelado 3d: personajes Mixamo



Ilustración 8: logo de Mixamo

Para modelar el personaje, existen soluciones web que pueden proporcionar un modelo básico sin perder mucho tiempo diseñándolo, una de ellas por ejemplo es Mixamo, una página web que gracias a la tecnología *Flash* permite diseñar de manera rápida, un personaje en 3d, descargarlo y retocarlo en cualquiera de los programas mencionados anteriormente.

### Modelado 3d: resumen

En primera instancia se seleccionó *Blender* para realizar todo el proceso de modelaje del proyecto, pero debido a incompatibilidades y fallos a la hora de aplicar ciertos modelos y texturas, se decidió hacer uso de las tres herramientas, por motivos distintos:

- **Blender** para la creación de modelos, objetos, armas y aplicar texturas.
- **3ds Max** para los problemas de exportación y formatos que a veces provocaba *Blender*, aunque el motivo principal fue el haber trabajado anteriormente con él para “riggear” los personajes, darles un esqueleto y crear animaciones.
- **SketchUp** para aprovechar la potencia de los mapas de *Google* de Leganés y crear la base de edificios que más tarde serían ampliados y retocados en *Blender*. Si bien este trabajo podría haberse realizado completamente en *Blender*, la facilidad de uso a la hora de aplicar los mapas como textura no la tiene el *Blender*.
- **Mixamo** para crear personajes de manera sencilla y rápida para posteriormente aplicarles esqueleto en *3ds Max*

## DESARROLLO DE UN VIDEOJUEGO CON UDK

En mi caso, tenía experiencia previa en *3ds Max* para tratar con personajes y dotarles de esqueleto, y de *Blender* a la hora de realizar el diseño del edificio Sabatini durante el curso de una asignatura de la carrera.

Si no tuviera conocimiento previo, recomendaría el combo *SketchUp + Blender*, ya que con el primero se obtiene la base de la manera más fácil y sencilla posible y con el segundo además de ser gratuito, ofrece todas las posibilidades que el proyecto necesita.

### LOS VIDEOJUEGOS

La historia de los videojuegos comienza en la década de 1940, tras finalizar la Segunda Guerra Mundial, se construyeron las primeras computadoras programables. Los primeros intentos por implementar programas de carácter lúdico no tardaron en aparecer. Los primeros videojuegos modernos surgen en la década de los sesenta, como el mítico *pong*, y desde entonces el mundo de los videojuegos no ha dejado de crecer y desarrollarse con el único límite que impone la tecnología. La popularidad que los videojuegos han alcanzado en solo medio siglo de vida es asombrosa, un mercado en constante evolución que ya se sitúa por encima del cine y de la música en cuanto a beneficios que se incrementan año a año. El impacto tan notorio que han supuesto los videojuegos es tema de estudio para los investigadores sociales, nació fruto de un experimento en el ámbito académico, se destinó al mercado infantil, y ha evolucionado hasta convertirse en un novedoso arte audiovisual para todo tipo de públicos.

Hoy en día, los videojuegos constituyen un mercado muy importante y en constante expansión, ayudado por el avance imparable de la tecnología, la aparición de los procesadores gráficos (GPU) y la rivalidad de grandes empresas (*Nvidia vs AMD*, *Sony vs. Microsoft* vs *Nintendo*) permite que cada vez aparezcan juegos con un nivel de detalle y una jugabilidad impensable hace apenas unos años.

#### *Dificultades a la hora de desarrollar un videojuego*

El principal inconveniente es el coste de partir de cero. Hoy en día, la industria del videojuego genera más dinero que la del cine y la música juntas, y desarrollar un videojuego son resultado de un gran número de personas colaborando durante años, lo cual condiciona a que exista una gran inversión inicial, y una ingente cantidad de ventas para que el producto obtenga beneficios y sea rentable.

Desarrollar un videojuego desde cero, sin motor de juego ya desarrollado, resulta una tarea casi impensable e inviable para un solo desarrollador o pequeña empresa, para ello, se hace uso de herramientas de creación de videojuegos ya existentes como *Unreal Engine*, *Blender Game Engine* o *Unity 3D*, que ya proveen el entorno de desarrollo necesario para dotar al videojuego de efectos impensables hace apenas unos años, y sin problemas con la interacción entre personaje-entorno-efectos-movimientos-cinematográficas que resulta fluida y natural al usar el mismo entorno de desarrollo.

En el tercer apartado del presente documento se describirán los motores de juego tanto sus versiones de pago como las gratuitas, que facilitan la creación de un videojuego permitiendo usar motores muy avanzados, de gran calidad, pudiendo obtener resultados muy satisfactorios.

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## MOTOR DE JUEGO

El aspecto más importante a la hora de desarrollar un videojuego es sin duda la elección del motor gráfico.

El motor gráfico es la base sobre la que se crea todo el videojuego, reúne un conjunto de componentes y herramientas que juntas, permiten construir, diseñar y representar un videojuego. Como si de un motor de coche se tratase, es el conjunto de estos componentes lo que hace funcionar la máquina, componentes tales como el editor de sonido, el sistema de importado de modelos, o el editor de animaciones de personajes, por si solos no hacen un videojuego, pero cuando todas las piezas encajan para crear una base, empezamos a hablar de un videojuego con todas sus características.

La función básica de un videojuego es mostrar gráficos en la pantalla (el término apropiado es renderizar), adicionalmente y a medida que tanto el hardware como el software ha ido evolucionando, se han ido incorporando componentes que ahora también resultan esenciales para desarrollar un videojuego, como puede ser el motor de físicas , el sistema de iluminación dinámica, la inteligencia artificial de los enemigos...Además de proporcionar un IDE gráfico que permita comunicarnos con el motor (engine) de manera simple e intuitiva.

Hoy en día, cuando se desea abordar el tema del motor gráfico existen tres alternativas

- La primera, supone crear el motor gráfico de cero, aunque si se desea desarrollar un juego competente y a la par con los juegos actuales, a no ser que se disponga de un gran equipo y años de plazo, esto no es viable para la gran mayoría de proyectos
- La otra alternativa, es usar algún motor gráfico ya existente, en el mercado existen multitud de motores gráficos, tanto gratuitos como de pago, cada uno tiene su propia curva de aprendizaje y no todos están orientados al mismo tipo de juegos, por lo que es muy importante saber qué motor elegir, por lo general, los motores con una curva de aprendizaje elevada son los que ofrecen mejores resultados visuales. También hay que considerar que la mayoría de estos motores llevan muchos años de desarrollo por compañías "top" de la industria de los videojuegos y, por tanto, ofrecen técnicas y herramientas muy desarrolladas y optimizadas.
- La tercera alternativa, sería la combinación de las dos anteriores, pudiendo acceder a un motor ya existente, y modificar su código para hacerlo funcionar a nuestro gusto o mezclarlo con un motor específico en colisiones o en físicas.

Todos los juegos desarrollados en la actualidad, hacen uso de librerías específicas para el tratamiento de gráficos, las más utilizadas por los desarrolladores son OpenGL y DirectX, que son las encargadas de comunicarse con la GPU del ordenador y ejecutar los shader para que compute polígonos.

Los motores de juego o *game engine* son un conjunto de componentes reutilizables que se desarrollan para poder comunicarse con estas librerías y ofrecer al desarrollador una serie de componentes para editar el juego a su gusto con interfaces intuitivas y sencillas.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

Un motor de juegos actual, normalmente tiene los siguientes componentes:

- Motor gráfico: Realiza funciones de renderizado 2D/3D. Mapea texturas, aplica efectos antialiasing, genera mallas 3D, gestiona el número de polígonos de la escena y los manipula.
- Grafo de escena: es un directorio representativo de la escena, donde el nodo raíz lo forma el mapa principal y de él descienden tanto los sub-mapas, como las áreas de buffer, como los objetos y secuencias ligados a cada uno de ellos.
- Motor de física: una de los componentes más importantes del motor de juegos, es el encargado de calcular y aplicar todas las leyes físicas del mundo real al mundo virtual, aspectos tales como la velocidad, gravedad, masa, fuerza, etc. No serían posibles sin este motor.
- Detector de colisiones: a modo de GPS, el sistema está constantemente actualizando la posición del personaje, de manera que este posee un área de colisión, que le permite como si fuera "hundir la flota" si el área de colisión de un personaje se encuentra en la misma zona en la que se ubica un objeto, el personaje se "choca".
- Motor de inteligencia artificial: El otro modulo importante, permite diseñar y programar toda la inteligencia de los personajes, es el encargado de que nuestro personaje reaccione, tenga estímulos, ataque y esquive, entre otras muchas acciones.
- Motor de sonido: Reproduce, gestiona y controla todos los sonidos, músicas y efectos que se interactúan en el juego, el motor de sonido es el aspecto que menos ha evolucionado en la historia de los videojuegos, se programa de la misma manera que hace veinte años.
- Componente de redes: encargado de la sincronización en línea.

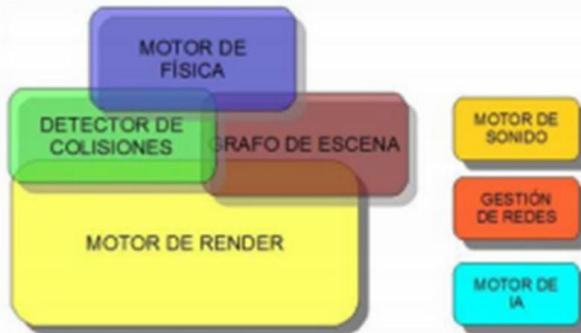


Ilustración 9: componentes de un entorno de un motor gráfico

Hoy en día existen numerosas alternativas para la creación de juegos. Por suerte, y debido a cuestiones de publicidad, tenemos a nuestra disposición motores gráficos usados en los juegos más punteros y actuales del mercado a coste cero. De este modo, la desarrolladora del motor gráfico te permite superar esa curva de aprendizaje sin haber gastado dinero, y solo se requerirá cuando quieras comercializar el producto ya concluido.

reparten el mercado de desarrollo de juegos:

- *Unreal engine*
- CryEngine
- Unity 3D

Actualmente existen tres motores que se

## DESARROLLO DE UN VIDEOJUEGO CON UDK

Tanto *Unreal* Engine, como CryEngine son los motores más usados para el desarrollo de juegos triple A, que son los que mayor presupuesto, pero gracias a la metodología de marketing comentada en el párrafo anterior, los desarrolladores de estos motores lo ofertan sin coste alguno (hasta su comercialización)

A lo largo de los siguientes apartados, se hará una breve introducción a cada uno de estos motores

### *Motores de juego: Unreal Engine*



Ilustración 10: logo de UDK

*Unreal* Engine es un motor de juego para PC, móviles y consolas desarrollado por la compañía *Epic Games* creadores del famoso videojuego "*Gears of War*". Actualmente es el motor gráfico más popular y con mayor uso entre las grandes empresas del sector, *EA*, *Capcom*, *2K Games*, *Square Enix*, *Warner Bros*, *Bethesda*, *Glu*, *Microsoft*, *Sony*, *Atari*, *Namco*, *Activision*, *Eidos*, lo que supone que la práctica totalidad de las empresas fuertes del sector de los videojuegos han hecho uso de este motor de gráficos.

*UDK, Unreal Development Kit*, es la herramienta de creación de juegos que incluye el motor *Unreal Engine 3*. Epic Games dispone de tres tipos distintos de comercialización:

- Los usuarios de *UDK*, pueden publicar sus juegos de manera gratuita únicamente si se realiza de forma no comercial.
- Epic Games ofrece licencias para permitir que el juego desarrollado con *UDK* pueda ser comercializado, sin poder acceder al código del motor de juego:
  - Para comercializar el producto es necesario adquirir una licencia que cuesta 99\$
  - Una vez comercializado, mientras los ingresos sean inferiores a 50,000\$, no es necesario pagar nada.
  - Si los beneficios por la comercialización del juego superan los 50,000\$, *Epic Games* se queda con el 25% de los beneficios futuros. Suponiendo que un juego actual cuesta en torno a unos 20\$ si es *indie* (estudio independiente), sería necesario vender 2,500 copias para pagar el 25% de los beneficios a *Epic* (una cifra muy por debajo de la primera tirada de una distribución actual, en torno a 10,000 copias)
- Por último, existe la licencia para usar y manipular el motor de juego, con un coste de 1,000,000\$

## DESARROLLO DE UN VIDEOJUEGO CON UDK

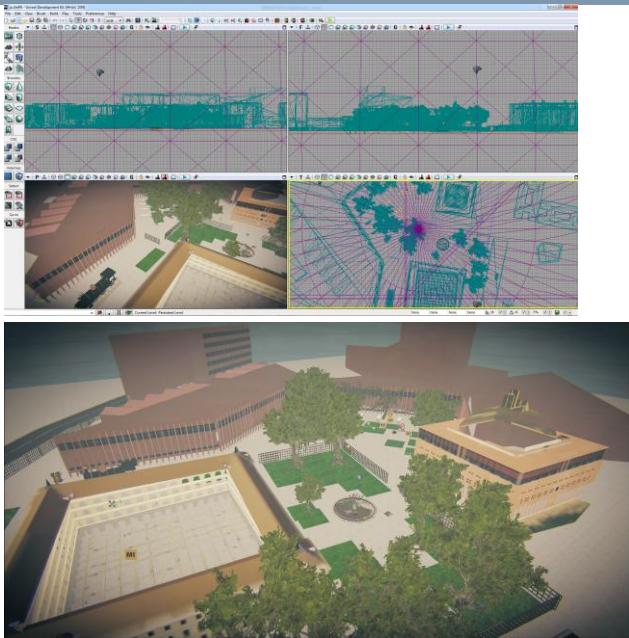


Ilustración 11: interfaz del entorno UDK

Las principales características técnicas de este motor de juegos son:

- Set de edición de desarrollo de videojuegos integrado
- Sistema de renderizado multihilo, con sistema de sombreado, iluminación, oclusión y gestor de sombras dinámicas (*Unreal Lightmass*)
- Tratamiento de partículas que aprovecha tecnologías como *PhysX*, tecnología que compró *Nvidia* y solo accesible si posees una gráfica de la misma compañía
- *Unreal Kismet* permite crear scripts de manera gráfica, a modo de elementos en un plano que se interrelacionan entre sí. Aplicado solo al nivel al que están asociados, permiten descargar parte del trabajo de código y hacerlo mucho más interactivo y fácil.
- *AnimSetViewer*, para el tratamiento de personajes, animaciones asociadas y cambios de los personajes del juego
- *AnimTree*, editor completo de la vegetación del mapa, permite crear todo tipo de plantas y árboles
- *Unreal Matinee*, editor de cinemáticas muy completo y profesional
- *Unreal Script*, lenguaje de programación de alto nivel orientado a objetos con una sintaxis estructura similar a Java y un tratamiento de funciones similar a C.

### Motores de juego: Cryengine



Ilustración 12: logo CryEngine

El motor de juegos del gran éxito de ventas “*Crysis*”, desarrollado por la empresa *Crytek* es realmente espectacular, ya que es el único de los motores analizados que se compila en

## DESARROLLO DE UN VIDEOJUEGO CON *UDK*

tiempo real, sin embargo tiene en su contra que es muy reciente y no tiene tantos recursos online como el motor *Unreal*, en constante evolución, con unos resultados más potentes y espectaculares que los que ofrece el *UDK*, requiere también de un equipo mucho más avanzado.

Al igual que el anteriormente analizado, la licencia para uso comercial, requiere un pago previo. Para desarrolladores independientes, al igual que *UDK*, es totalmente gratuito hasta su comercialización, en donde es necesario pagar el 20% de los beneficios. Comprando la licencia para editar el código núcleo del motor de juego (con un coste similar al millón de dólares del anterior) no es necesario pagar por el beneficio generado

Las especificaciones técnicas son muy similares a las del *UDK*, listare solamente las principales diferencias:

- Actualización en tiempo real en el editor *sandbox*
- Soporte multinucleo
- Iluminación dinámica en tiempo real
- Adaptación de retina y HDR
- Editor de animación facial
- Mayor velocidad de renderizado y procesado
- Motor de físicas más avanzado que el *UDK*

### *Motores de juego: Unity*



Este motor ha sido desarrollado por la empresa *Unity Technologies* que con su última versión y su gran capacidad de plataformas a las que da soporte, está teniendo mucho éxito.

Se distingue de las anteriores en tres puntos

Ilustración 13: logo Unity

- La versión gratuita no da acceso a todas las herramientas, para acceder a ellas o bien se accede a la versión Pro o se adquieren licencias para módulos extra
- La versión de pago cuesta 1,500\$ sin ningún tipo de porcentaje sobre el beneficio
- Es la que posee mayor comunidad en internet y la interfaz más amigable e intuitiva

Este motor gráfico no se encuentra en la misma liga que los dos anteriores, orientado a un público con presupuesto mucho más ajustado, cuenta con una comunidad enorme, además de soportar la publicación para muchas más plataformas que sus competidores, prácticamente todas las del mercado. Por último, su factor más importante es la flexibilidad de lenguajes que es capaz de manejar (*UDK*, utiliza *UnrealScript; CryEngine, LUA/C++*).

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## CREACIÓN DE MENÚS



Ilustración 14: logo ScaleForm

Para crear los menús del juego, se ha optado por usar la herramienta “adobe Flash Professional” ya que el formato flash es aceptado por el motor *UDK* gracias a una extensión (*plugin*) llamado Scalefrom GFx que permite usar gráficos dinámicos creados en Adobe Flash Professional como interfaces de usuario y *HUDs (Head-Up Displays, información presente durante el juego)* y permite ser renderizado directamente en la pantalla o renderizarse como textura/material para aplicarse a una superficie o modelo,

## EDICIÓN DE VIDEO



Ilustración 15: logo  
Windows Movie Maker

Todo videojuego necesita una presentación, un video introductorio, para ello, se han utilizado dos videos ya existentes, previo permiso del propietario y se ha creado un video introductorio utilizando la herramienta Windows Movie Maker.

Video Paintball: [http://www.youtube.com/watch?v=KvT6fr\\_MykY](http://www.youtube.com/watch?v=KvT6fr_MykY)

Video Uc3m: <http://www.youtube.com/watch?v=qn-J8nFDNII>

## TRATAMIENTO DE IMÁGENES

Para el tratamiento de imágenes se han usado dos programas:

- **Gimp:** programa bajo licencia GPL que permite editar imágenes digitales libre, gratuito e intuitivo.
- **ImRe:** Programa de conversión y redimensión de múltiples imágenes de forma simultánea. Permite adaptar las imágenes a las condiciones solicitadas por el entorno *UDK*

## REQUISITOS Y RESTRICCIONES

La fase de análisis, permite identificar las necesidades del proyecto gracias a la obtención y definición de los requisitos funcionales y no funcionales que ha de cumplir nuestro producto final. Para ello, comenzaremos el proceso identificando los casos de uso y a concluirá con la definición de los requisitos de software

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## CASOS DE USO

Los casos de uso es el modo de identificar las distintas interacciones que un usuario tendrá con la aplicación.

El objetivo principal del proyecto, consiste en un videojuego FPS con una finalidad únicamente lúdica.

### Actores

El actor principal y único que tiene el proyecto se trata del ***jugador***, para desempeñar este rol vale cualquier tipo de persona, genero, sexo, edad ya que el juego no presenta ningún tipo de contenido para adultos ni posee vocabulario obsceno, si bien en principio, está destinado a miembros de la propia comunidad universitaria o a jóvenes como medio propagandístico.

### Casos de uso: Especificación

Para clasificar los casos de uso se usara una plantilla estándar, para que sea más fácil la identificación de cada uno de sus campos

| Identificador   |
|-----------------|
| Nombre          |
| Descripción     |
| Actor           |
| Precondiciones  |
| Postcondiciones |
| Escenario       |

Tabla 2: Plantilla Casos de Uso

Donde cada elemento significa:

- ***Identificador***: Número único que identifica cada caso de uso.
- ***Nombre***: título descriptivo del caso de uso.
- ***Descripción***: explicación clara y concisa del caso de uso.
- ***Actor***: tipo de usuario (en este caso siempre será *jugador*)
- ***Precondiciones***: condiciones que se deben cumplir antes de ejecutarse el caso de uso.
- ***Postcondiciones***: condiciones que deben cumplir el sistema al concluir el caso de uso.
- ***Escenario***: etapas del caso de uso.

### Casos de uso: JUGADOR

La siguiente ilustración muestra los distintos casos de uso que el actor *jugador* puede realizar al comienzo del videojuego.

Como todos los videojuegos, el menú principal contiene las siguientes opciones:

- ***Play***: Inicia una nueva partida.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

- **Options:** permite configurar los parámetros del juego (modo de pantalla y sonido)
- **Exit:** permite abandonar la partida.

Debido al modo de juego, no existe la opción de guardar la partida, ya que no existe ningún sistema de niveles aun incorporado (a consultar en el apartado de líneas futuras)

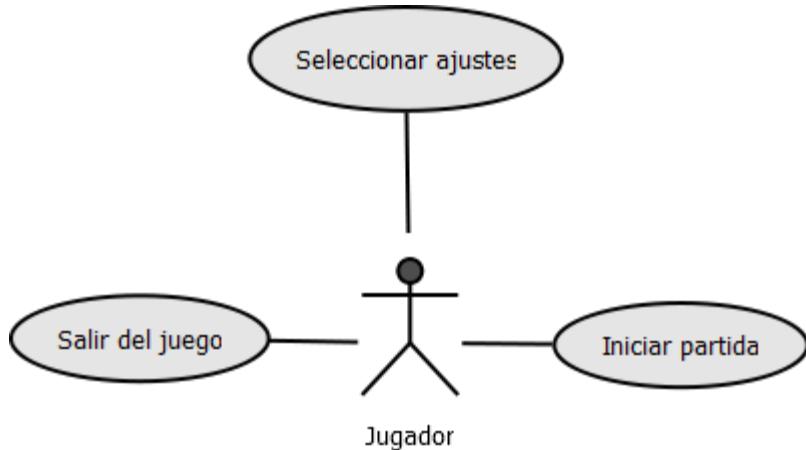


Ilustración 16: Casos de Uso del menú principal

| Identificador   | CU01  |
|-----------------|---|
| Nombre          | Iniciar partida   |
| Descripción     | El jugador tiene como primera opción iniciar una nueva partida  |
| Actor           | Jugador   |
| Precondiciones  | Ninguna   |
| Postcondiciones | Inicia una nueva partida  |
| Escenario       | El Jugador inicia la aplicación y tras la presentación, accede al menú principal, donde, tras elegir la opción de <i>play</i> dará comienzo una partida nueva |

Tabla 3:CU01

| Identificador   | CU02   |
|-----------------|--|
| Nombre          | Seleccionar ajustes  |
| Descripción     | El jugador tiene como segunda opción modificar dos parámetros del juego  |
| Actor           | Jugador  |
| Precondiciones  | Ninguna  |
| Postcondiciones | Ejecuta el menú de parámetros del juego  |
| Escenario       | El Jugador inicia la aplicación y tras la presentación, accede al menú principal, donde, tras elegir la opción de <i>options</i> ejecutara el menú de opciones |

Tabla 4:CU02

## DESARROLLO DE UN VIDEOJUEGO CON UDK

|                        |  |
|------------------------|--|
| <b>Identificador</b>   | CU03   |
| <b>Nombre</b>          | Salir de la aplicación   |
| <b>Descripción</b>     | El jugador tiene como tercera opción salir de la aplicación  |
| <b>Actor</b>           | Jugador  |
| <b>Precondiciones</b>  | La aplicación ha de estar iniciada   |
| <b>Postcondiciones</b> | Cierra la aplicación   |
| <b>Escenario</b>       | El Jugador inicia la aplicación y tras la presentación, accede al menú principal, donde, tras elegir la opción de <i>exit</i> cierra la aplicación |

Tabla 5:CU03

Si se selecciona el menú de opciones, el jugador puede realizar las siguientes acciones:

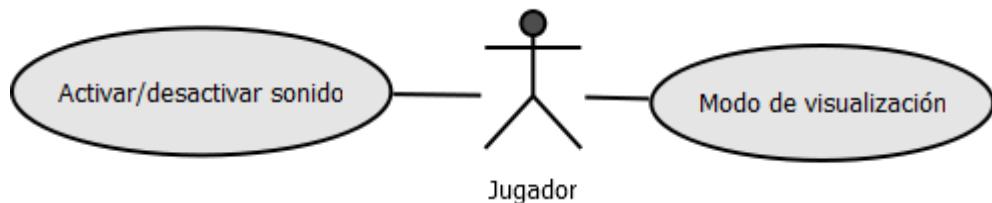


Ilustración 17: Casos de Uso del menú opciones

|                        |   |
|------------------------|---|
| <b>Identificador</b>   | CU04  |
| <b>Nombre</b>          | Activar/Desactivar sonido   |
| <b>Descripción</b>     | El jugador tiene la posibilidad de activar/desactivar sonido  |
| <b>Actor</b>           | Jugador   |
| <b>Precondiciones</b>  | La aplicación ha de estar iniciada y seleccionado el menú de opciones en el menú principal  |
| <b>Postcondiciones</b> | Activa/desactiva el sonido  |
| <b>Escenario</b>       | El Jugador tras acceder al menú de opciones desde el menú principal, tiene la posibilidad de activar/desactivar el sonido mediante un <i>toggle</i> |

Tabla 6:CU04

|                        |   |
|------------------------|---|
| <b>Identificador</b>   | CU05  |
| <b>Nombre</b>          | Modo de visualización   |
| <b>Descripción</b>     | El jugador tiene la posibilidad de cambiar la resolución del juego  |
| <b>Actor</b>           | Jugador   |
| <b>Precondiciones</b>  | La aplicación ha de estar iniciada y seleccionado el menú de opciones en el menú principal  |
| <b>Postcondiciones</b> | Modifica la resolución de la pantalla del juego   |
| <b>Escenario</b>       | El Jugador tras acceder al menú de opciones desde el menú principal, tiene la posibilidad de seleccionar entre tres resoluciones, dos de ellas en modo <i>ventana</i> y una en modo <i>fullscreen/pantalla completa</i> . |

Tabla 7:CU05

## DESARROLLO DE UN VIDEOJUEGO CON UDK

La siguiente ilustración muestra las distintas posibilidades que tiene el actor *Jugador* cuando se encuentra en el juego, al ser un juego del estilo *sandbox* puede moverse libremente, como disparar a enemigos y mover cámara, cambiar tipo de disparo, cambiar arma, etc.

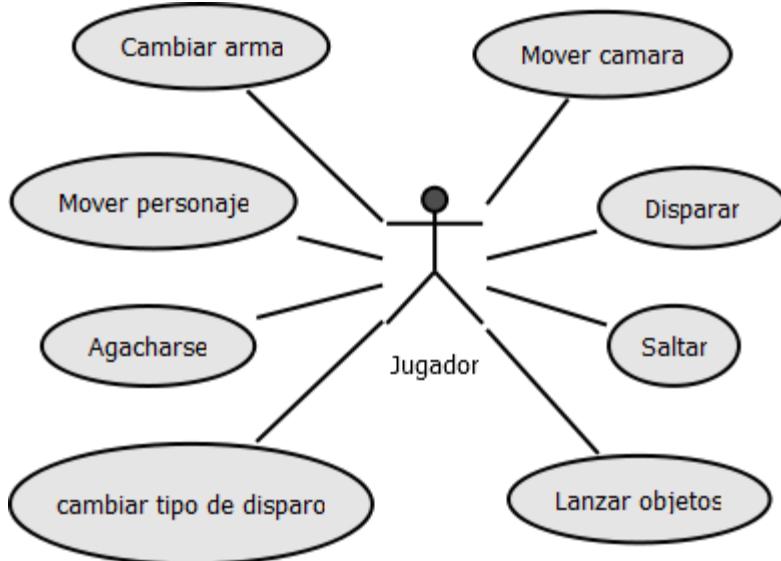


Ilustración 18: Casos de Uso durante la partida

| Identificador   | CU06   |
|-----------------|--|
| Nombre          | Cambiar arma   |
| Descripción     | El jugador tiene la posibilidad de cambiar el arma moviendo la rueda central del ratón o bien pulsando las teclas numéricas. En caso de usar un <i>gamepad</i> , se realizará pulsando los botones superiores del mando. |
| Actor           | Jugador  |
| Precondiciones  | La aplicación ha de estar iniciada e iniciada la partida   |
| Postcondiciones | Equipa al personaje la siguiente arma del inventario   |
| Escenario       | El Jugador tras acceder al juego desde el menú principal, tiene la posibilidad de seleccionar entre un número variable de armas disponibles, cada una con propiedades únicas   |

Tabla 8: CU06

| Identificador   | CU07  |
|-----------------|---|
| Nombre          | Mover cámara  |
| Descripción     | El jugador tiene la posibilidad de mover la cámara moviendo el ratón o el joystick principal si usa un <i>Gamepad</i> |
| Actor           | Jugador   |
| Precondiciones  | La aplicación ha de estar iniciada e iniciada la partida  |
| Postcondiciones | Mueve la cámara por el escenario  |

## DESARROLLO DE UN VIDEOJUEGO CON UDK

|                  |  |
|------------------|--|
| <b>Escenario</b> | El Jugador tras acceder al juego desde el menú principal, tiene la posibilidad de modificar la posición de la cámara |
|------------------|--|

Tabla 9:CU07

| <b>Identificador</b>   | <b>CU08</b>  |
|------------------------|--|
| <b>Nombre</b>          | Mover personaje  |
| <b>Descripción</b>     | El jugador tiene la posibilidad de mover al personaje mediante el uso de las teclas de dirección (W,S,A,D) o con la cruceta principal si usa un <i>Gamepad</i> |
| <b>Actor</b>           | Jugador  |
| <b>Precondiciones</b>  | La aplicación ha de estar iniciada e iniciada la partida   |
| <b>Postcondiciones</b> | Mueve la cámara por el escenario   |
| <b>Escenario</b>       | El Jugador tras acceder al juego desde el menú principal, tiene la posibilidad de modificar la posición de la cámara   |

Tabla 10:CU08

| <b>Identificador</b>   | <b>CU09</b>   |
|------------------------|---|
| <b>Nombre</b>          | Disparar  |
| <b>Descripción</b>     | El jugador tiene la posibilidad de disparar la pistola de paintball mediante el uso del clic derecho del ratón o con el botón determinado si usa un <i>Gamepad</i> (generalmente el gatillo superior trasero derecho) |
| <b>Actor</b>           | Jugador   |
| <b>Precondiciones</b>  | La aplicación ha de estar iniciada e iniciada la partida  |
| <b>Postcondiciones</b> | Dispara el arma   |
| <b>Escenario</b>       | El Jugador tras acceder al juego desde el menú principal, tiene la posibilidad de disparar el arma  |

Tabla 11:CU09

| <b>Identificador</b>   | <b>CU10</b>   |
|------------------------|---|
| <b>Nombre</b>          | Agacharse   |
| <b>Descripción</b>     | El jugador tiene la posibilidad de hacer que el personaje se agache mediante el uso de la tecla shift/ctrl del teclado o con el botón determinado si usa un <i>Gamepad</i> (generalmente el botón X/[]) |
| <b>Actor</b>           | Jugador   |
| <b>Precondiciones</b>  | La aplicación ha de estar iniciada e iniciada la partida  |
| <b>Postcondiciones</b> | El personaje ejecuta la acción de agacharse   |
| <b>Escenario</b>       | El Jugador tras acceder al juego desde el menú principal, tiene la posibilidad de hacer que el personaje se agache  |

Tabla 12:CU10

| <b>Identificador</b> | <b>CU11</b>   |
|----------------------|---|
| <b>Nombre</b>        | Saltar  |
| <b>Descripción</b>   | El jugador tiene la posibilidad de hacer que el personaje se agache mediante el uso de la tecla espacio del teclado o con el botón determinado si usa un <i>Gamepad</i> (generalmente el botón A/X) |
| <b>Actor</b>         | Jugador   |

## DESARROLLO DE UN VIDEOJUEGO CON UDK

|                        |  |
|------------------------|--|
| <b>Precondiciones</b>  | La aplicación ha de estar iniciada e iniciada la partida   |
| <b>Postcondiciones</b> | El personaje ejecuta la acción de saltar   |
| <b>Escenario</b>       | El Jugador tras acceder al juego desde el menú principal, tiene la posibilidad de hacer que el personaje salte |

Tabla 13:CU11

| Identificador          | CU12  |
|------------------------|---|
| <b>Nombre</b>          | Cambiar tipo de disparo   |
| <b>Descripción</b>     | El jugador tiene la posibilidad de modificar el tipo de disparo (balas o proyectiles) mediante el uso del clic izquierdo del ratón o con el botón determinado si usa un <i>Gamepad</i> (generalmente el gatillo delantero superior derecho) |
| <b>Actor</b>           | Jugador   |
| <b>Precondiciones</b>  | La aplicación ha de estar iniciada e iniciada la partida  |
| <b>Postcondiciones</b> | Cambia el tipo de disparo   |
| <b>Escenario</b>       | El Jugador tras acceder al juego desde el menú principal, tiene la posibilidad de cambiar el tipo de disparo  |

Tabla 14:CU12

| Identificador          | CU13  |
|------------------------|---|
| <b>Nombre</b>          | Lanzar objetos  |
| <b>Descripción</b>     | El jugador tiene la posibilidad de disparar la pistola de paintball mediante el uso de la tecla G del teclado o con el botón determinado si usa un <i>Gamepad</i> (generalmente el botón Y/Δ) |
| <b>Actor</b>           | Jugador   |
| <b>Precondiciones</b>  | La aplicación ha de estar iniciada e iniciada la partida  |
| <b>Postcondiciones</b> | Dispara el arma   |
| <b>Escenario</b>       | El Jugador tras acceder al juego desde el menú principal, tiene la posibilidad de disparar el arma  |

Tabla 15:CU13

Del mismo modo, el menú de pausa durante el juego presenta las siguientes opciones:

- **Resume:** reanuda la partida pausada.
- **Quit:** cierra la partida.

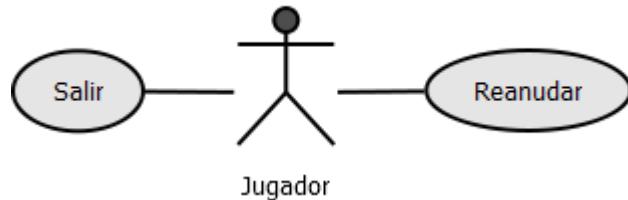


Ilustración 19: Casos de Uso durante el menú de pausa

## DESARROLLO DE UN VIDEOJUEGO CON UDK

| Identificador   | CU14   |
|-----------------|--|
| Nombre          | Reanudar   |
| Descripción     | El jugador tiene la posibilidad, una vez accede al menú de pausa durante la partida, a reanudar el juego |
| Actor           | Jugador  |
| Precondiciones  | La aplicación ha de estar abierta, iniciada la partida y en el menú de pausa                             |
| Postcondiciones | Reanuda el juego   |
| Escenario       | El Jugador tras acceder al menú de pausa desde el juego, tiene la posibilidad de volver a la partida     |

Tabla 16:CU14

| Identificador   | CU15   |
|-----------------|--|
| Nombre          | Salir  |
| Descripción     | El jugador tiene la posibilidad, una vez accede al menú de pausa durante la partida, a salir del juego |
| Actor           | Jugador  |
| Precondiciones  | La aplicación ha de estar abierta, iniciada la partida y en el menú de pausa                           |
| Postcondiciones | Cierra la aplicación   |
| Escenario       | El Jugador tras acceder al menú de pausa desde el juego, tiene la posibilidad de cerrar el videojuego  |

Tabla 17:CU15

## REQUISITOS SOFTWARE

En este apartado se listaran los requisitos software de la aplicación a desarrollar, dichos requisitos se han concretado en las reuniones concertadas con los tutores del proyecto. Al igual que en el apartado anterior, se facilita la descripción de los campos de la plantilla que se va a usar para la clasificación de los requisitos

| Identificador   | RSF/NF-01 |
|-----------------|-----------|
| Nombre          |           |
| Descripción     |           |
| Prioridad       |           |
| Claridad        |           |
| Estabilidad     |           |
| Verificabilidad |           |
| Necesidad       |           |
| Fuente          |           |

Tabla 18: Plantilla de requisitos

Donde cada campo significa:

- **Identificador:** Código numérico único que identifica cada requisito
- **Nombre:** nombre descriptivo del requisito

## DESARROLLO DE UN VIDEOJUEGO CON UDK

- **Descripción:** explicación breve, clara y concisa del requisito.
- **Prioridad:** nivel de importancia o urgencia de un requisito, a mayor prioridad, mayor necesidad de cumplirlo cuanto antes.
- **Estabilidad:** probabilidad de que el requisito cambie. A mayor estabilidad, la probabilidad de que el requisito se vea modificado es menor.
- **Claridad:** a mayor claridad, menor ambigüedad tendrá el requisito y menos posibilidades de malinterpretación.
- **Verificabilidad:** facilidad que posee el requisito de comprobar su correcta aplicación.
- **Necesidad:** nivel de importancia para el cliente que tiene el requisito.
- **Fuente:** origen del requisito, persona (cliente) o documentación.

Para la organización de los requisitos se van a dividir en dos grupos:

- **Requisito de software funcional:** especifica el propósito del software.
- **Requisitos de software no funcional:** especifica el modo de realizar las tareas y sus requisitos.

### *Requisitos funcionales*

A continuación se listan los requisitos del videojuego a desarrollar:

| Identificador   | RSF-01   |
|-----------------|--|
| Nombre          | Iniciar nueva partida  |
| Descripción     | El sistema deberá permitir iniciar una partida siempre que el usuario lo desee |
| Prioridad       | Alta   |
| Claridad        | Alta   |
| Estabilidad     | Alta   |
| Verificabilidad | Alta   |
| Necesidad       | Esencial   |
| Fuente          | Tutor  |

Tabla 19: RSF-01

| Identificador   | RSF-02  |
|-----------------|---|
| Nombre          | Mostrar escenario con campus de Leganés   |
| Descripción     | El mapa principal del videojuego debe localizarse e inspirarse en el campus que la Universidad Carlos III de Leganés tiene en Leganés |
| Prioridad       | Alta  |
| Claridad        | Alta  |
| Estabilidad     | Alta  |
| Verificabilidad | Alta  |
| Necesidad       | Esencial  |
| Fuente          | Tutor   |

Tabla 20: RSF-02

## DESARROLLO DE UN VIDEOJUEGO CON UDK

|                        |   |
|------------------------|---|
| <b>Identificador</b>   | RSF-03  |
| <b>Nombre</b>          | Sistema de juego FPS-Paintball  |
| <b>Descripción</b>     | El tipo de videojuego a desarrollar ha de ser un juego del tipo FPS, First person shooter (juego de disparos en primera persona) usando armas de paintball, nunca balas ni elementos que puedan resultar sensibles. |
| <b>Prioridad</b>       | Alta  |
| <b>Claridad</b>        | Alta  |
| <b>Estabilidad</b>     | Alta  |
| <b>Verificabilidad</b> | Alta  |
| <b>Necesidad</b>       | Esencial  |
| <b>Fuente</b>          | Tutor   |

Tabla 21: RSF-03

|                        |   |
|------------------------|---|
| <b>Identificador</b>   | RSF-04  |
| <b>Nombre</b>          | Desplazar personaje   |
| <b>Descripción</b>     | El usuario podrá desplazar el personaje que controlara mediante un joystick o las teclas del mando. |
| <b>Prioridad</b>       | Alta  |
| <b>Claridad</b>        | Alta  |
| <b>Estabilidad</b>     | Alta  |
| <b>Verificabilidad</b> | Alta  |
| <b>Necesidad</b>       | Esencial  |
| <b>Fuente</b>          | Tutor   |

Tabla 22: RSF-04

|                        |  |
|------------------------|--|
| <b>Identificador</b>   | RSF-05   |
| <b>Nombre</b>          | Mejora Desplazamiento personaje  |
| <b>Descripción</b>     | El usuario podrá desplazar el personaje haciendo uso de la tecnología Kinect/wiimote |
| <b>Prioridad</b>       | Media  |
| <b>Claridad</b>        | Alta   |
| <b>Estabilidad</b>     | Alta   |
| <b>Verificabilidad</b> | Alta   |
| <b>Necesidad</b>       | Deseable   |
| <b>Fuente</b>          | Tutor  |

Tabla 23: RSF-05

|                      |  |
|----------------------|--|
| <b>Identificador</b> | RSF-06   |
| <b>Nombre</b>        | Salir de la aplicación   |
| <b>Descripción</b>   | La aplicación ofrecerá la posibilidad al usuario de cerrar y salir de la aplicación en el momento que el jugador lo deseé. |
| <b>Prioridad</b>     | Alta   |
| <b>Claridad</b>      | Alta   |
| <b>Estabilidad</b>   | Alta   |

## DESARROLLO DE UN VIDEOJUEGO CON UDK

|                        |          |
|------------------------|----------|
| <b>Verificabilidad</b> | Alta     |
| <b>Necesidad</b>       | Esencial |
| <b>Fuente</b>          | Tutor    |

Tabla 24: RSF-06

| Identificador          | RSF-07  |
|------------------------|---|
| <b>Nombre</b>          | Menú principal  |
| <b>Descripción</b>     | La aplicación deberá tener un menú principal desde el que se pueda iniciar una nueva partida, modificar parámetros o abandonar el juego |
| <b>Prioridad</b>       | Alta  |
| <b>Claridad</b>        | Alta  |
| <b>Estabilidad</b>     | Alta  |
| <b>Verificabilidad</b> | Alta  |
| <b>Necesidad</b>       | Esencial  |
| <b>Fuente</b>          | Tutor   |

Tabla 25: RSF-07

| Identificador          | RSF-08  |
|------------------------|---|
| <b>Nombre</b>          | Menú pausa  |
| <b>Descripción</b>     | La aplicación deberá tener un menú pausa disponible en todo momento del juego, desde el que se pueda reanudar la partida o abandonar el juego |
| <b>Prioridad</b>       | Alta  |
| <b>Claridad</b>        | Alta  |
| <b>Estabilidad</b>     | Alta  |
| <b>Verificabilidad</b> | Alta  |
| <b>Necesidad</b>       | Deseable  |
| <b>Fuente</b>          | Tutor   |

Tabla 26: RSF-08

| Identificador          | RSF-09   |
|------------------------|--|
| <b>Nombre</b>          | Reproducir música y efectos de sonido                              |
| <b>Descripción</b>     | El sistema deberá utilizar e incorporar música y efectos de sonido |
| <b>Prioridad</b>       | Alta   |
| <b>Claridad</b>        | Alta   |
| <b>Estabilidad</b>     | Alta   |
| <b>Verificabilidad</b> | Alta   |
| <b>Necesidad</b>       | Esencial   |
| <b>Fuente</b>          | Desarrollador  |

Tabla 27: RSF-09

| Identificador      | RSF-10   |
|--------------------|--|
| <b>Nombre</b>      | Animaciones y cinemáticas                              |
| <b>Descripción</b> | El juego debería tener alguna cinemática y/o animación |
| <b>Prioridad</b>   | Alta   |

## DESARROLLO DE UN VIDEOJUEGO CON UDK

|                        |               |
|------------------------|---------------|
| <b>Claridad</b>        | Alta          |
| <b>Estabilidad</b>     | Alta          |
| <b>Verificabilidad</b> | Alta          |
| <b>Necesidad</b>       | Deseable      |
| <b>Fuente</b>          | Desarrollador |

Tabla 28: RSF-10

### *Requisitos no funcionales*

En este apartado se listan los requisitos no funcionales de la aplicación

| Identificador          | RSNF-01  |
|------------------------|--|
| <b>Nombre</b>          | Multiplataforma  |
| <b>Descripción</b>     | La aplicación deberá, en la medida de lo posible, ser compatible con distintas plataformas |
| <b>Prioridad</b>       | Alta   |
| <b>Claridad</b>        | Alta   |
| <b>Estabilidad</b>     | Alta   |
| <b>Verificabilidad</b> | Alta   |
| <b>Necesidad</b>       | Esencial   |
| <b>Fuente</b>          | Desarrollador  |

Tabla 29: RSNF-01

| Identificador          | RSNF-02   |
|------------------------|---|
| <b>Nombre</b>          | Compatibilidad con resoluciones   |
| <b>Descripción</b>     | La aplicación deberá, en la medida de lo posible, ser compatible con distintas resoluciones de pantalla |
| <b>Prioridad</b>       | Alta  |
| <b>Claridad</b>        | Alta  |
| <b>Estabilidad</b>     | Alta  |
| <b>Verificabilidad</b> | Alta  |
| <b>Necesidad</b>       | Esencial  |
| <b>Fuente</b>          | Desarrollador   |

Tabla 30: RSNF-02

| Identificador          | RSNF-03  |
|------------------------|--|
| <b>Nombre</b>          | Fichero Ejecutable   |
| <b>Descripción</b>     | La aplicación deberá ser un único instalador, para hacer fácil y sencilla su instalación |
| <b>Prioridad</b>       | Alta   |
| <b>Claridad</b>        | Alta   |
| <b>Estabilidad</b>     | Alta   |
| <b>Verificabilidad</b> | Alta   |
| <b>Necesidad</b>       | Esencial   |

# DESARROLLO DE UN VIDEOJUEGO CON UDK

Fuente

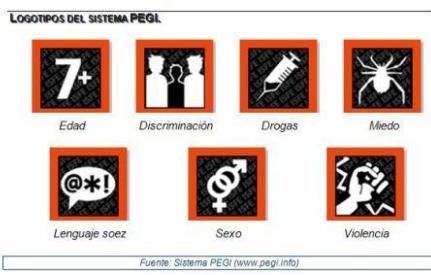
Desarrollador

Tabla 31: RSNF-03

## MARCO REGULADOR

Como toda industria en alza que mueve grandes cantidades de dinero, la industria del videojuego y la administración central han establecido un sistema de criterios y normas que permiten regular el mercado de los videojuegos en España.

- **Código nacional de autorregulación:** Elaborado por la aDeSe en el año 2001, establece las pautas de actuación en el desarrollo de software interactivo en aspectos como la clasificación, etiquetado, promoción y publicidad



- **Código europeo de autorregulación:** Auspiciado por la Comisión Europea, publicado por la *Interactive Software Federation of Europe* y suscrito por aDeSe en España, establece el estándar PEGI (*Pan Europeana Game Information*) la cual ofrece a los compradores, educadores y consumidores información concreta y garantizada sobre el contenido explícito e implícito de un videojuego y su calificación dentro de un rango de edades.

Ilustración 20: Código sistema PEGI

## DESARROLLO DE UN VIDEOJUEGO CON UDK

### 3. Diseño y desarrollo de la solución

#### INTRODUCCIÓN

En el presente capítulo se explicara de manera concisa, detallada y minuciosa el diseño del proyecto a desarrollar, componentes y arquitectura del conjunto.

#### DISEÑO DE LA SOLUCIÓN INICIAL

El proyecto consiste en un videojuego del tipo *sandbox* o mundo abierto, donde el jugador puede recorrer un mapa sin seguir un camino, a medida que recorra el mapa, el jugador puede encontrarse con objetos, armas o enemigos a batir.

#### LIMITACIONES INICIALES

Las limitaciones iniciales del proyecto están bastante claras

- El objetivo es terminar el proyecto antes del 24 de septiembre de 2013, por lo que se estima un plazo de planificación de 4 meses
- El desarrollo del proyecto depende únicamente de una persona, la cual asume todos los roles del proyecto: diseñador, programador, modelador, animador, etc...
- Para algunas de las tecnologías que se van a usar, es necesario superar el proceso de aprendizaje, lo que conlleva un tiempo que no se ve reflejado en el resultado del proyecto

#### IDEAS BÁSICAS: ¿CÓMO SERÁ EL VIDEOJUEGO?



Ilustración 21: Imagen del GTA

Para la realización del videojuego, se han tomado como modelos dos videojuegos:

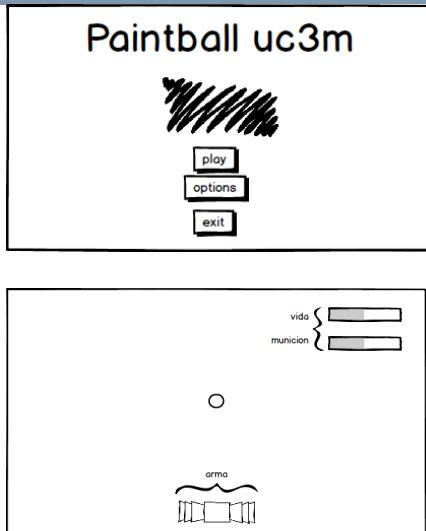
- El videojuego “*Grand Theft Auto*” que propone un *sandbox*, o *juego de mundo abierto* donde no existe una serie de acciones lineales que lleven al jugador por un camino único, sino que se le ofrece la posibilidad de explorar el escenario o realizar múltiples acciones a elección del jugador
- La serie de juegos en primera persona *Counter Strike*, en el que dos facciones se enfrentan en un escenario común



Ilustración 22: imagen del Counter Strike

Ambos juegos son tomados como referencia en su estilo, y la idea del proyecto es tomar lo mejor de ambos, proporcionar un mundo abierto donde puedas realizar múltiples acciones y a la vez enfrentarte contra otro equipo.

## DESARROLLO DE UN VIDEOJUEGO CON UDK



Los primeros wireframes, o bocetos que se hicieron de la aplicación mantenían el mismo estilo que las referencias anteriormente mencionadas, en la pantalla siempre ha de aparecer información acerca de la vida y la munición restante, el puntero y el catálogo de armas disponibles. El menú principal ofrecerá la posibilidad de iniciar una nueva partida, ajustar para metros de la misma y salir del juego.

El resto de menus tienen que seguir la estética minimalista del boceto, y el *HUD*, o menú *in-game* no debe variar en ningún momento de la partida.

Ilustración 23: mockups del diseño

### ARQUITECTURA DE LA APLICACIÓN

La arquitectura de la aplicación permite obtener un diseño a alto nivel del sistema, pudiendo identificar claramente los módulos que lo forman y la relación existente entre ellos. Para la elaboración de este diseño, se ha utilizado el modelo de arquitectura en tres capas. El diseño tiene como prioridad minimizar el grado de dependencia de los componentes.

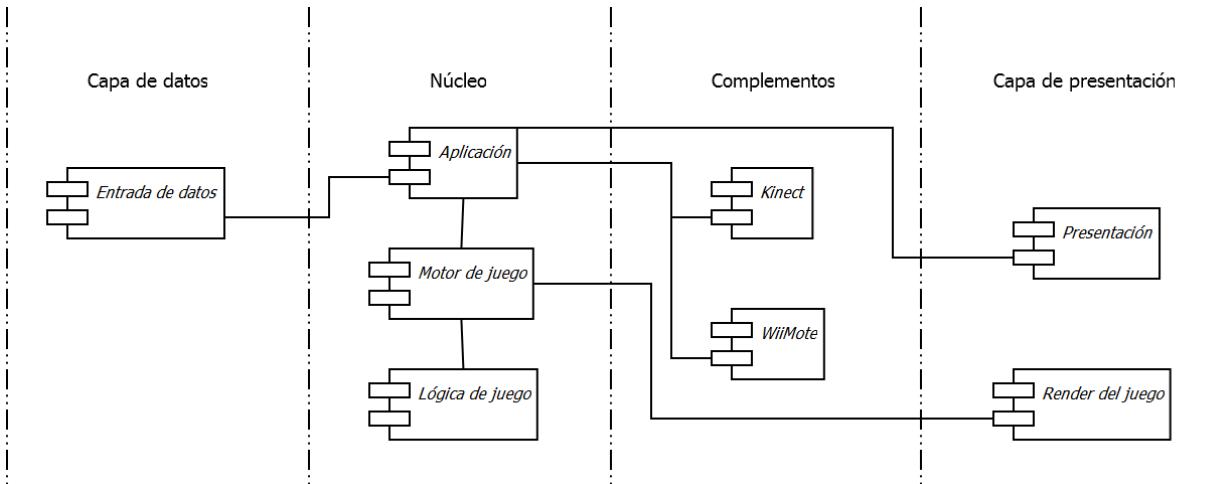


Ilustración 24: Esquema de la arquitectura

- *Módulo de entrada de datos:* Conforma la biblioteca de elementos del componente Aplicación, en él se almacenarán todos los modelos creados.
- *Módulo de aplicación:* componente principal del proyecto, encargado de aunar los componentes de mayor relevancia y ejecutar la aplicación, controla el flujo de datos, el motor de juego, la lógica, la comunicación entre el motor y la biblioteca de elementos.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

- *Módulo de motor de juego:* Controla cada uno de los aspectos del juego, iluminación, movimiento del personaje, físicas, entorno, sonidos, animaciones, control de datos, renderizado de elementos, sistema de partículas, sistema de colisiones, y el encargado de realizar la salida por pantalla a través del render.
- *Módulo de lógica de juego:* código escrito en lenguaje de programación que establece las relaciones entre los elementos del juego y el comportamiento de los personajes.
- *Módulo Kinect:* proporciona el código necesario para interpretar los datos que capta el dispositivo Kinect y los traduce en movimientos para el juego
- *Módulo Wii Mote:* proporciona el código necesario para interpretar los datos que captura el dispositivo Wii mote y los traduce en movimientos del juego.
- *Módulo de presentación:* Encargado de la salida por pantalla, proporciona al usuario una interfaz visual de la aplicación.

El modelo planteado, permite la separación de la lógica de negocios de la lógica de diseño, la capa de datos, es la encargada de obtener los modelos *.FBX/.ASE* externos a la lógica del juego, el núcleo constituye la parte más importante del videojuego, ya que engloba la lógica, el motor y la aplicación, por lo tanto, la modificación de cualquiera de estos componentes afecta directamente al producto final. La capa de complementos, situada entre el núcleo y la capa de presentación, permite ampliar la interfaz entre el jugador y la aplicación, siendo capaz de elegir entre cuatro dispositivos de entrada (*teclado, mando, Kinect y Wii mote*), por último la capa de presentación solo se encarga de mostrar al jugador el renderizado del juego y demás interfaces de la aplicación.

## SOFTWARE, TÉCNICAS Y MÉTODOS PLANTEADOS

A continuación se va a proceder a describir las tecnologías, técnicas y herramientas que se han utilizado para el desarrollo de la aplicación.

### Tecnologías utilizadas

Tras haber estudiado las distintas alternativas existentes a la hora de diseñar un videojuego, se realizó la selección de las más adecuadas para el proyecto. En la imagen siguiente se puede observar a modo de esquema la relación existente entre las aplicaciones seleccionadas para usarse a modo de guía.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

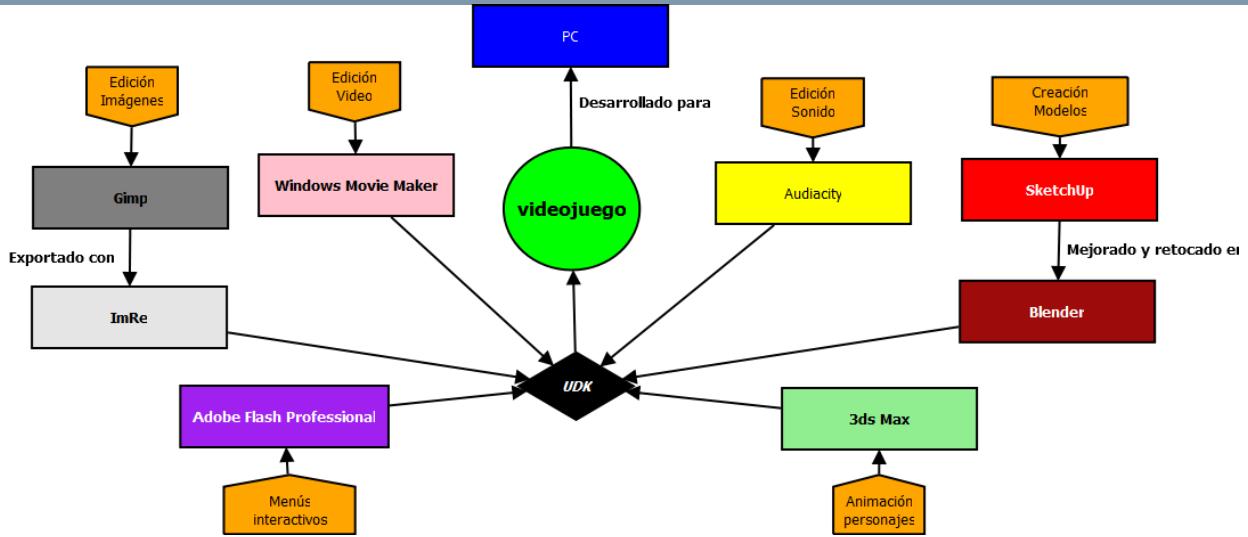


Ilustración 25: Esquema de aplicaciones

- **GIMP:** El editor de imágenes bajo licencia de uso gratuito, se ha utilizado para retoque de texturas e imágenes, creación de patrones y mejora de visualización.
- **ImRe:** Usado para la redimensión y conversión entre distintos formatos, para adaptarlos al formato que reconoce UDK.
- **Windows Live Movie Maker:** Edición de videos de presentación e introducción del juego.
- **Adobe flash profesional:** Creación de menús interactivos usando la tecnología *Flash*
- **UDK:** Entorno de desarrollo que reúne todas las herramientas necesarias para desarrollar un videojuego utilizando el motor *Unreal Engine*.
- **Audacity:** Editor de sonido con el que se han retocado música y efectos sonoros del videojuego.
- **SketchUp:** Programa de diseño gráfico y modelado 3D que incorpora integración con *Google Maps*, de interfaz muy sencilla simple e intuitiva
- **Blender:** suite de diseño gráfico bajo licencia de uso gratuita, usado para creación de edificios.
- **3ds Max:** suite de diseño y animación gráfica, usado para la creación de personajes, y proporcionarles un esqueleto.
- **\*Mixamo:** herramienta *on-line* que permite diseñar personajes en 3D

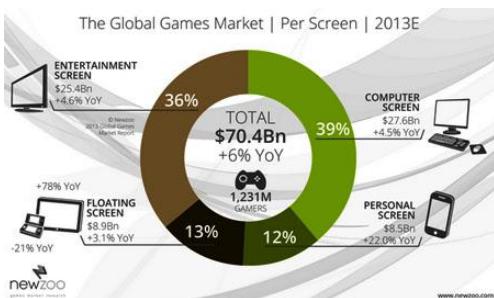
### Plataforma

La plataforma sobre la que desarrollar el proyecto era un aspecto fundamental hace unos años, ya que para cada plataforma, era necesario usar un IDE específico. Afortunadamente, hoy en día, desarrollar un juego multiplataforma es un proceso bastante sencillo e indispensable si se quiere asegurar un gran número de ventas.

La gran mayoría de los IDE actuales permiten exportar el juego a un sinfín de plataformas, tanto PC, como consolas y dispositivos móviles.

# DESARROLLO DE UN VIDEOJUEGO CON UDK

El editor escogido, *UDK*, permite desarrollar desde PC (*Windows*) y exportar a cualquier otra plataforma, previo pago de la licencia adecuada.



En cuanto a la plataforma a la que exportar, el gráfico adjunto muestra que hoy en día el sector que más usuarios posee, es el de las videoconsolas de sobremesa, para llegar al mayor número de potenciales usuarios, se recomienda exportar para las plataformas : *Xbox 360, PS3, Wii + PC*

Ilustración 26: Distribución del mercado global

## Motor de juego

Tras el estudio previo realizado, se decidió descartar *CryEngine* por los escasos recursos online que posee y realizar una comparativa en profundidad entre los dos motores restantes:

| Motor de juego                          | UDK  | Unity3D  |
|---|--|--|
| Gráficos                                | Sistema de iluminación completo<br>Sistema de partículas avanzado<br>Iluminación dinámica<br>Sombras en tiempo real<br>Mapeado con mapas de normales | Iluminación por mapeado solo con versión pro<br>Sombras en tiempo real solo con versión pro  |
| Físicas                                 | Sistema de físicas muy avanzado<br>Sistema avanzado de colisiones<br>Sistema de <i>fracturing</i><br>Basado en el objeto (Mesh)                      | Sistema de físicas por mejorar<br>No posee <i>fracturing</i><br>Sistema basado en la textura |
| Extensiones admitidas                   | .UPK, .FBX, .ASE, .OBJ   | Todas las extensiones  |
| Plataformas soportadas                  | Adobe flash, Android, Windows, Mac, iOS, Xbox 360, PS3 y PSVita  | Web, adobe flash, Mac, iOS, android, Windows, Wii, xbox360, PS3.                             |
| SS.OO soportados                        | Windows  | Winsows,Mac  |
| Lenguajes aceptados                     | UnrealScript, Kismet   | Boo, C#, JavaScript  |
| Extensiones                             | Kismet,Matinee,Scaleform, etc  | External plugins   |
| Curva de aprendizaje                    | Elevada  | Media  |
| Experiencia personal previa             | Si   | No   |
| Precio                                  | Licencia 99\$/25% beneficios si se obtienen gnaancias > 50.000\$   | iOS 400\$, Android 400\$, pro license 1500\$   |
| Juegos notables realizados con el motor | Batman Arkham City (AAA), Borderlands 2 (AAA), Mass Effect(AAA), Gears of War (AAA)  | BeGone(indie), Cowboy Guns(indie), Ghost Recon Network (A), Angry Birds (indie)              |

# DESARROLLO DE UN VIDEOJUEGO CON *UDK*

Tabla 32: Comparativa *UDK* vs *Unity*

Cada uno de los motores tiene sus puntos Fuertes, si bien me he decantado por el *UDK* por dar los mejores resultados en el aspecto gráfico al que le di mayor importancia, también otro aspecto muy influyente ha sido la experiencia previa personal con el motor *UDK*, por lo que no tenía que aprender a usarlo y me permitía ahorrar ese tiempo para dedicárselo al proyecto.

## *Formatos de importación utilizados*

Se han utilizado los siguientes formatos de importación/exportación

- **FBX:** es un formato de archivo 3D que proporciona acceso al contenido creado en cualquier paquete de software
- **ASE:** *ASCII Scene Exporter*, es el formato más utilizado para insertar modelos en *UDK* hasta la aparición del formato *.FBX* en los casos en los que el primer formato ha dado problemas de sombras o polígonos, se ha usado este formato
- **PNG:** *Portable Network Graphics*, es un formato grafico basado en un algoritmo de compresión sin pérdida para bitmaps. Es el formato estándar para texturas que reconoce *UDK*
- **UPK:** extensión genérica que proporciona *UDK* a colecciones de elementos.
- **WAV:** *Waveform Audio File Format*, es un formato de audio desarrollado por *IBM/Microsoft*
- **WMV:** nombre genérico que se otorga a un conjunto de algoritmos de compresión ubicados en el set propietario de tecnologías de video desarrolladas por *Microsoft*, normalmente el archivo se encuentra empaquetado en el contenedor multimedia AVI
- **BIK:** desarrollado por *RAD Game Tools* es un formato de video propietario (privado) utilizado en la industria del videojuego para importar secuencias de video ya que el propio códec implementa los demás códecs necesarios para que se reproduzca correctamente sobre cualquier plataforma
- **SWF:** formato de archivo de gráficos vectoriales creado por Macromedia (actual Adobe)
- **BLEND:** formato que utiliza el software *Blender* para guardar proyectos realizados con la herramienta.

## *Obtención de recursos*

El primer paso comienza por obtener información acerca del lugar que vamos a diseñar, obtención de fotos para usarlas de referencia a la hora de aplicar referencias para texturas a la hora de generar los edificios, fotos de cada uno de los edificios del campus y medidas para tener una idea de sus proporción

## DESARROLLO DE UN VIDEOJUEGO CON UDK



Ilustración 27: conjunto de imágenes recopiladas para el diseño

Otra vía de obtención de información para el mapa han sido los mapas de *Google*, ya que la vista aérea es la que nos proporciona una idea más clara sobre la colocación proporciones y distancias de cada uno de los edificios

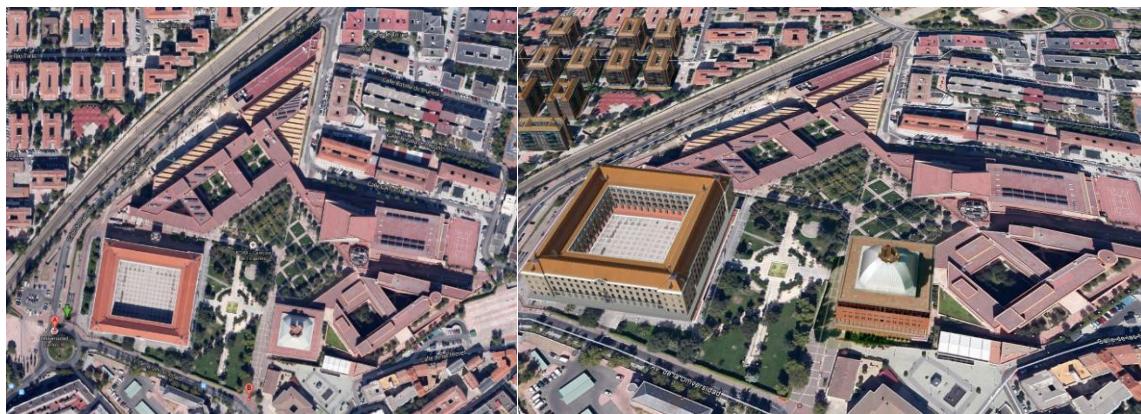


Ilustración 28: mapas del campus de Leganés

## DESARROLLO DE ALTERNATIVAS DE DISEÑO

Como alternativa de diseño al tener el edificio Sabatini ya creado en *Blender* y la realización de todos los modelos en el mismo software, se propuso utilizar el propio motor de juego que ofrece *Blender*, el *Blender Game Engine* que ofrece un entorno de desarrollo no tan avanzado como los anteriormente mencionados pero que ahorra la exportación/importación de modelos de un software a otro al tener todo en una misma herramienta.

Los resultados obtenidos, tanto en términos de calidad como en jugabilidad, no son los esperados, una tasa de imágenes por segundo (*FPS*) muy baja que hace que la jugabilidad sea muy difícil, un sistema inestable sujeto a cierres inesperados de la aplicación

MARCO ANTONIO PAJARES  
2012/2013

TRABAJO FIN DE GRADO

# DESARROLLO DE UN VIDEOJUEGO CON *UDK*

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## IMPLEMENTACIÓN

### GESTIÓN DE DATOS

A continuación se va a detallar cual ha sido la solución tomada para la gestión de datos que hacen uso cada una de las herramientas del proyecto. Al utilizar el entorno *UDK* todos los elementos se almacenan en paquetes con extensión *.UPK* en el directorio por defecto asignado.

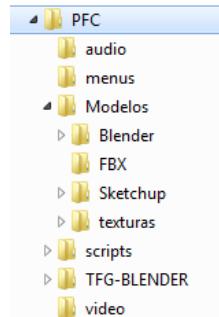


Ilustración 29: Directorio del proyecto

- Carpeta **audio**: La música que requiere ser editada y los sonido a importar en *UDK*
- Carpeta **menús**: Los distintos menús del juego realizados en *Flash* para importar al entorno *UDK*
- Carpeta **modelos**: aquí se almacenan los modelos generados tanto en *SketchUp* como *Blender*, el directorio *FBX* contiene los modelos preparados para importar en *UDK* y la carpeta texturas almacena todas las texturas de los modelos creados.
- Carpeta **scripts**: el código en *UnrealScript* necesario para hacer funcionar el proyecto. Esta carpeta se copia íntegramente en el directorio de *UDK*: *Development/src/uc3mPaint\_1p/Classes*
- Carpeta **TFG-Blender**: contiene la alternativa del proyecto, usando íntegramente el motor contenido en *Blender*
- Carpeta **video**: videos tanto de la presentación como del juego listos para importar o editar

### CREACIÓN DE MODELOS ARQUITECTÓNICOS

Para empezar modelar los edificios se hace uso de dos herramientas

- **Blender**: utilizado para la creación de modelos desde cero y para ajustar, recortar y aplicar texturas a modelos ya existentes en la biblioteca de *SketchUp*.
- **SketchUp**: Usado para descargar los modelos de la biblioteca online y eliminar las partes no deseadas para luego ser retocados en *Blender*

Tras realizar la investigación previa, se observó que en *Google SketchUp* ya existían modelos de varias ubicaciones de Leganés, algunas ya se encontraban hechas pero por problemas de rendimiento debido a

## DESARROLLO DE UN VIDEOJUEGO CON UDK

su complejidad y nivel de detalle no pudieron ser insertadas en el juego, ya que impedía que el juego se ejecutara de manera fluida.



Ilustración 30: algunos modelos extraídos de la biblioteca online de *SketchUp*

### Biblioteca

El edificio de la biblioteca Rey Pastor de Leganés ya existía en la galería de *SketchUp*, por lo que para su implantación se descargó el modelo, se exportó en formato .FBX y se retocó en *Blender* eliminando caras y reasignando texturas al exportarse estas en .JPEG y UDK solo admite el formato .PNG y con unas dimensiones recomendables de potencia de base 2.

El principal problema al importar este modelo ha sido que no ha reconocido las texturas del techo, por lo que al mover el personaje existen ciertos ángulos que al no posicionarse bien la forma, se calcula mal el color y no proyecta el material. Para solucionar parcialmente el problema, se sustituyó el material de la parte superior por un oro metálico, el modelo original tenía el interior sólido, y en *Blender* se retocó para vaciarlo y usarlo como almacén de armas y munición. En la imagen se observa el interior con paredes transparentes, esto es debido a que solo se ha aplicado material a la cara externa del modelo, la cara interna no posee material aun.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

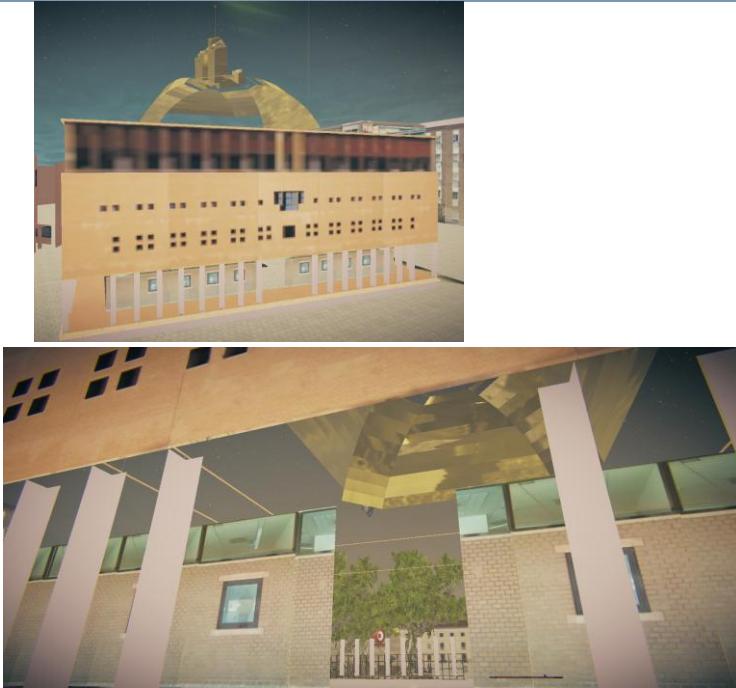


Ilustración 31: modelo biblioteca exterior e interior

### Betancourt

Este edificio se ha realizado completamente desde cero usando *SketchUp* y *Blender* ya que no existía el modelo en la biblioteca online de *SketchUp*

Para comenzar se importó una imagen de base extraída de *Google Maps* (ver Ilustración 28: mapas del campus de Leganés) y con la herramienta de lápiz creamos el contorno del edificio.

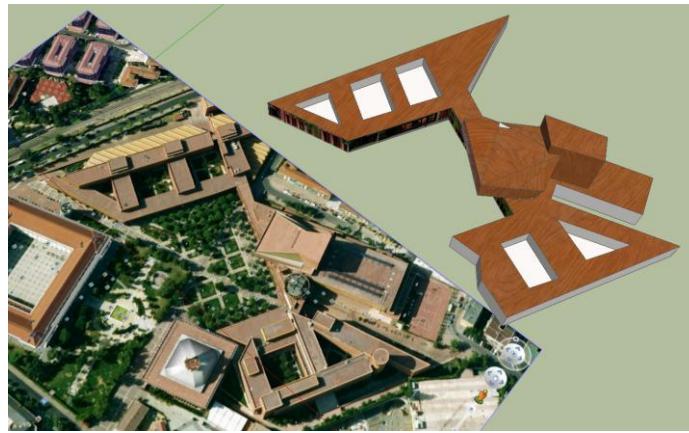


Ilustración 32: comparación mapa *Google Maps* vs modelo hecho con *SketchUp*

## DESARROLLO DE UN VIDEOJUEGO CON UDK

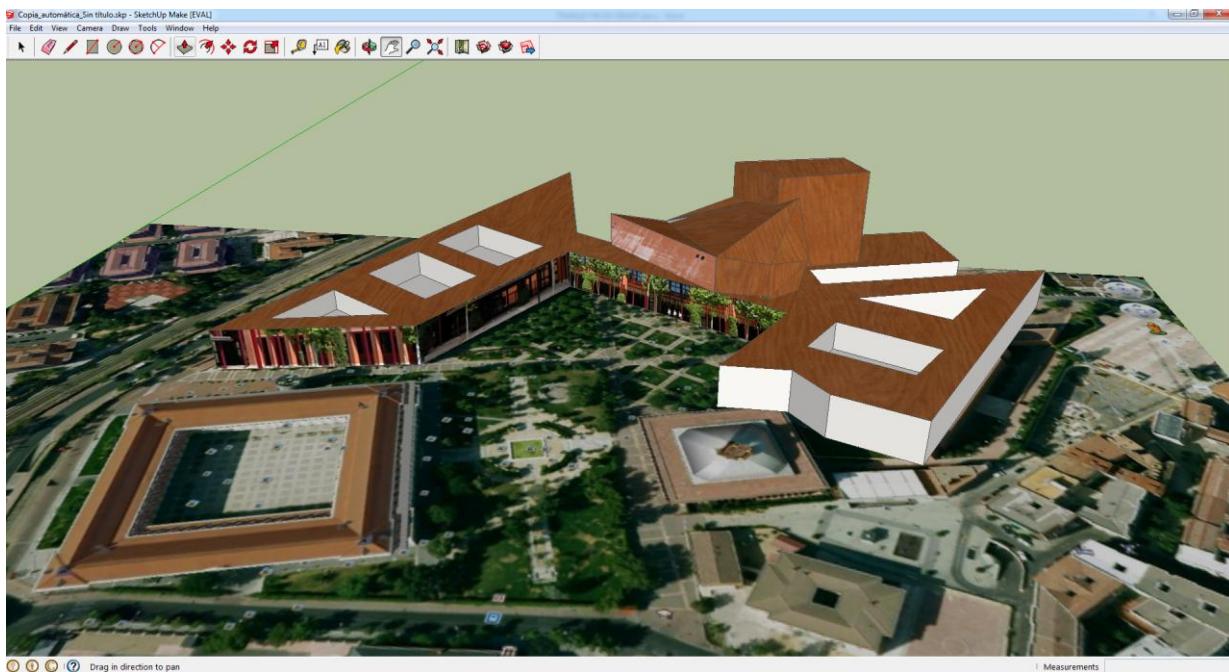


Ilustración 33: edificio mixto (1 y 4) en *SketchUp*

Como se puede apreciar en la imagen, el modelo ha intentado en lo más posible parecerse al original, con ayuda de la herramienta *push/pull* se hicieron los patios “empujándolos” hasta la base. Del mismo modo, se consiguió realizar la forma del auditorio *Padre Soler*.

En la imagen también podemos apreciar la aplicación de fotos hechas con el móvil y aplicadas sobre la propia pared del edificio, si bien no son las definitivas, podía utilizarse como guía para la edición y mejora en *Blender*.

Una vez importado en *Blender* era necesario realizar las siguientes mejoras:

- Crear y añadir la fachada del edificio *Betancourt + Torres Quevedo*
- Crear y añadir el edificio 7 (*Juan Benet*)
- Mejoras al modelo inicial
- Aplicación de texturas

## DESARROLLO DE UN VIDEOJUEGO CON UDK

Crear y añadir la fachada del edificio Betancourt + Torres Quevedo



Ilustración 34: imagen del edificio Betancourt

Tal y como se puede apreciar en la imagen, la fachada el edificio Betancourt está compuesta por una serie de cilindros verticales con un panel en la parte superior, que cubren una fachada de ladrillo visto acabado en columnas.

Para realizar el proceso, se ha creado una fachada independiente que pueda repetirse a lo largo de toda la parte frontal del edificio, esta fachada se compone de un array de cilindros y columnas pegados a un rectángulo que se apoya en un plano vertical donde se aplicara la textura y cubrirá la fachada del edificio.

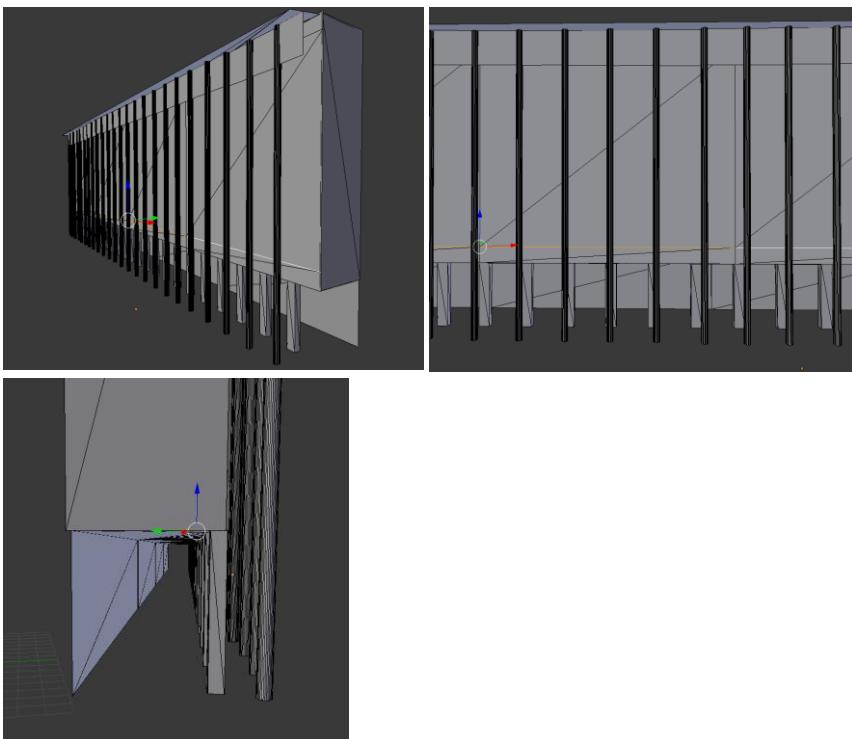


Ilustración 35: Diseño fachada del edificio Betancourt

Crear y añadir el edificio 7 (Juan Benet)



Ilustración 36 Imagen del edificio Juan Benet

El edificio 7, junto con su reciente ampliación, era necesario introducirlo en el modelo, afortunadamente este edificio es un rectángulo en el que la fachada sigue un patrón muy simple. Se utilizaron dos rectángulos uno seguido de otro y se extendió la parte superior de uno de ellos, para crear la sensación de unión entre los tejados. Para la puerta, se extruyó la parte delantera inferior del edificio consiguiendo un efecto de columnas laterales.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

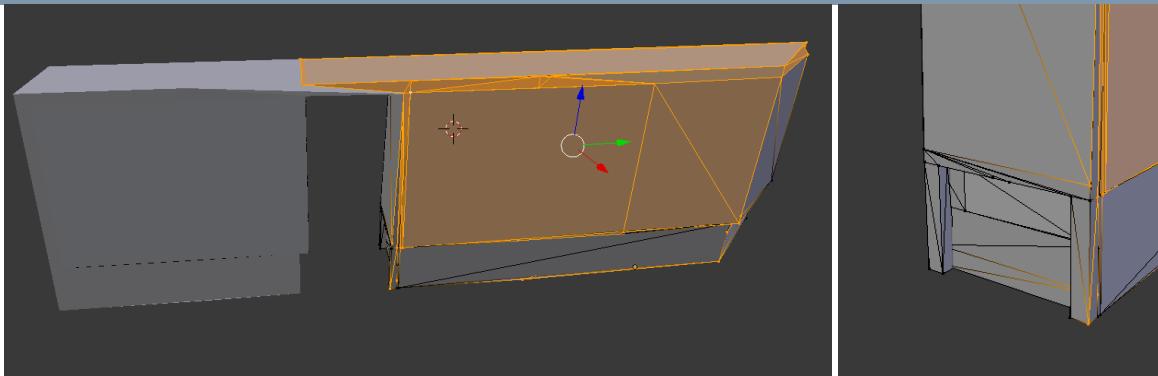


Ilustración 37: Diseño del edificio Juan Benet

### Mejoras al modelo inicial



Al modelo inicial se le ha añadido la sección lateral correspondiente a los talleres del edificio *Betancourt*, para ello, se ha añadido un rectángulo que cubra la parte lateral del edificio y se le ha colocado encima un triángulo extruido hacia fuera y con un vértice desplazado hacia el interior para asemejarse más el modelo real. Esta sección ha sido realmente fácil de implementar una vez realizado el modelo principal.

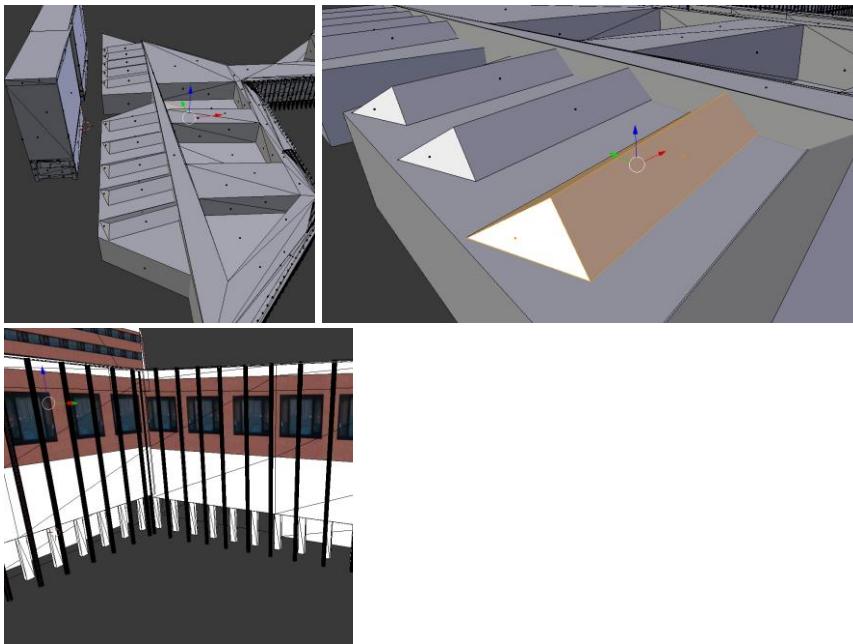


Ilustración 38: Diseño del edificio Betancourt (II)

En la imagen superior se puede apreciar el desplazamiento hacia el interior del borde superior del triángulo y como coincide con la fotografía superior de la imagen real, tambien es posible apreciar

## DESARROLLO DE UN VIDEOJUEGO CON UDK

como se ha eliminado el trozo de la esquina izquierda del patio que permite atravesar el edificio por debajo.

### Aplicación de texturas

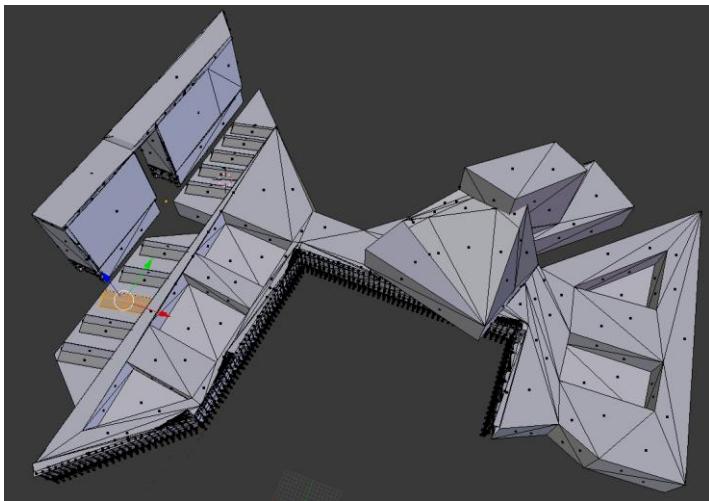


Ilustración 39: Modelo edificio Betancourt

Para la aplicación de texturas se utiliza el método UV *unwrap* el cual como se comentó en el segundo apartado, permite aplicar un imagen de dos dimensiones sobre una superficie 2D del modelo tridimensional.

En la imagen lateral podemos ver el modelo ya con las mejoras aplicadas y con los edificios añadidos, los puntos indican el centro de cada una de las caras que existen en el modelo. También es posible apreciar cómo se ha aplicado la fachada a todo el frontal del edificio, logrando el efecto deseado de las columnas y el pasillo cubierto.

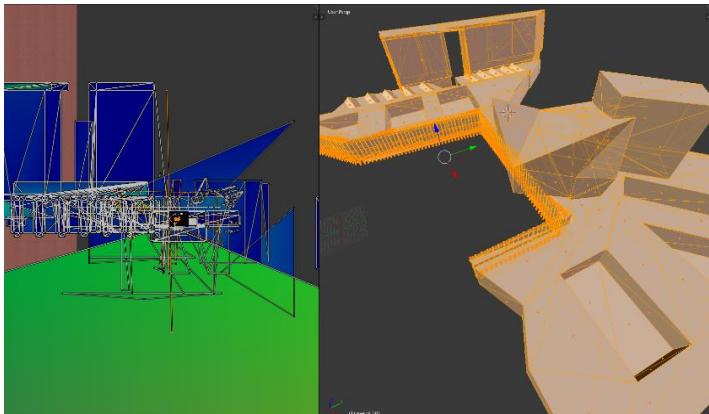


Ilustración 40: Aplicación UVW Mapping

En la parte izquierda de la imagen lateral, se ve el proceso de *mapping*, ya que es la representación bidimensional del modelo tridimensional situado a la derecha. Las secciones azules comparten más de una textura, las verdes no poseen textura o no la encuentra y la sección que se encuentra sobre la textura de ladrillo, es a la que se aplica dicha textura únicamente.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

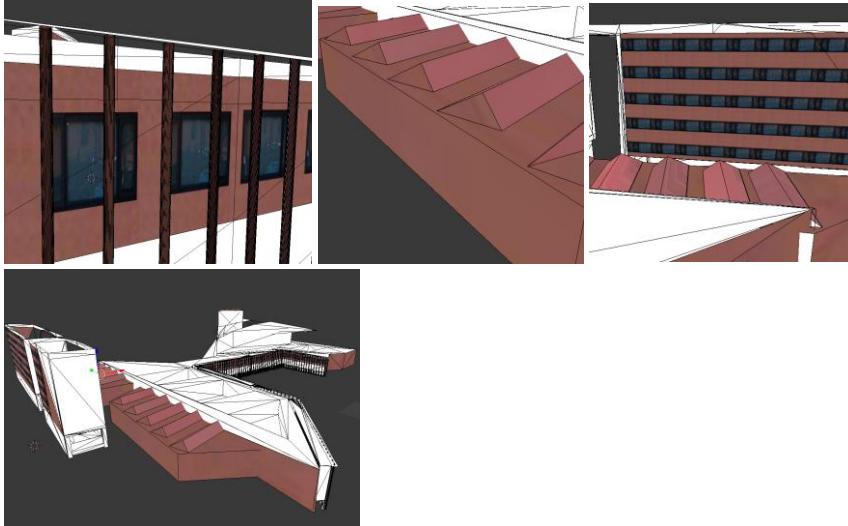


Ilustración 41: Aplicación de texturas al edificio Betancourt

### Sabatini

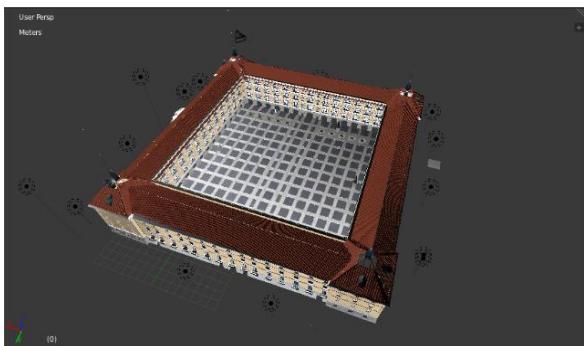


Ilustración 42: Modelo edificio Sabatini

Para la creación de este edificio se partió del modelo que previamente se había realizado para la asignatura de informática gráfica. El problema de este modelo es que era tan detallado, que el motor de juego no podía importarlo de manera correcta al tener demasiados polígonos, así que la primera opción fue usar la herramienta de simplificación para reducir el número de polígonos, gracias a la eliminación de caras que son visibles y se consiguió aligerar el peso, pero seguía siendo un número muy elevado para que el sistema lo aceptara, ya que

acepta un máximo de 65550 vértices y el modelo tenía 1.145.878 por lo que no era posible reducirlo más sin que ello afectara a la visualización, ya que el proceso de simplificado deforma el objeto al eliminar vértices y caras, así que se optó por una combinación del modelo de *SketchUp* y el modelo creado, dotando de mucha mayor profundidad al modelo de *SketchUp* y no afectando al rendimiento.

El modelo de *SketchUp* (ver ilustración 44) no tiene la calidad que presentaba el modelo anterior, al igual que la biblioteca es un rectángulo al que se le han insertado imágenes de Google Maps como textura plana sin ningún tipo de profundidad.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

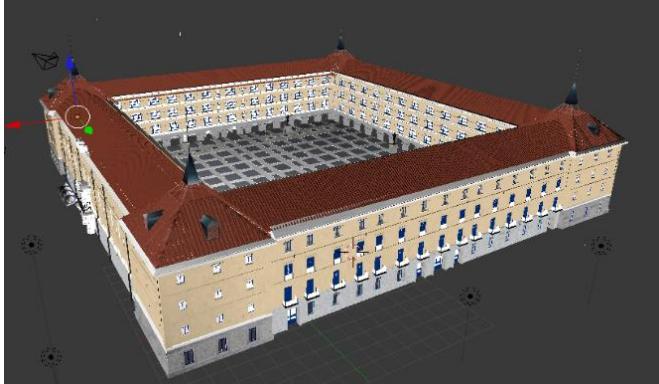


Ilustración 43: Modelo edificio Sabatini (II)

A lo largo de esta sección se explicara el proceso de creación del edificio Sabatini, aunque en el videojuego se insertara una mezcla entre el modelo de *SketchUp* y el creado en *Blender*, para mejorar la jugabilidad y que el número de polígonos no afectara al rendimiento de la aplicación

El edificio Sabatini se dividió en tres zonas:

- Fachada
- Fachada interna y patio

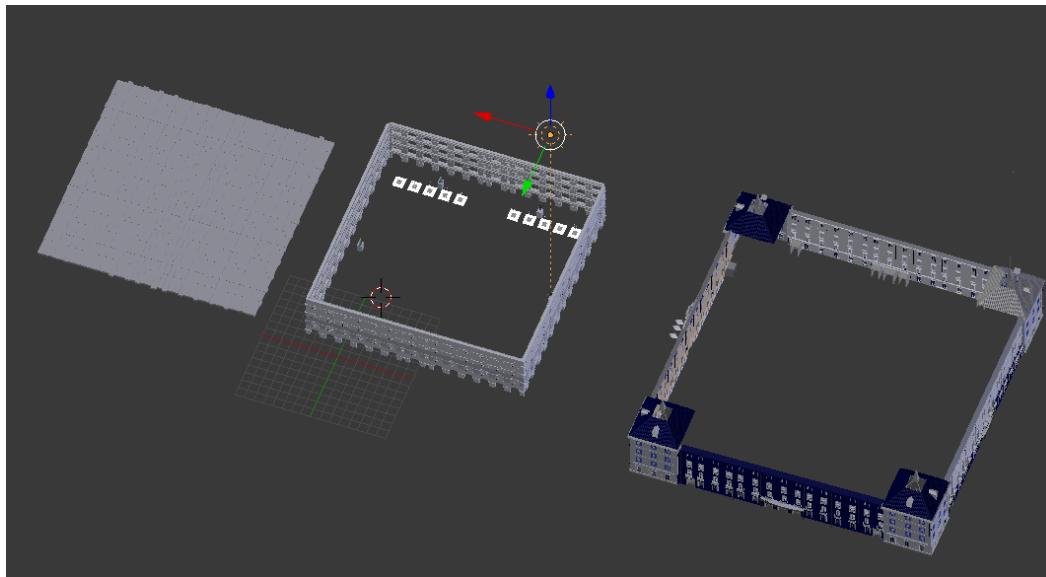


Ilustración 44: Módulos del edificio Sabatini

Estas tres partes incorporan elementos muy personalizados y claramente distinguibles, a continuación, se explicará en profundidad el desarrollo de cada una de las partes.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

Fachada

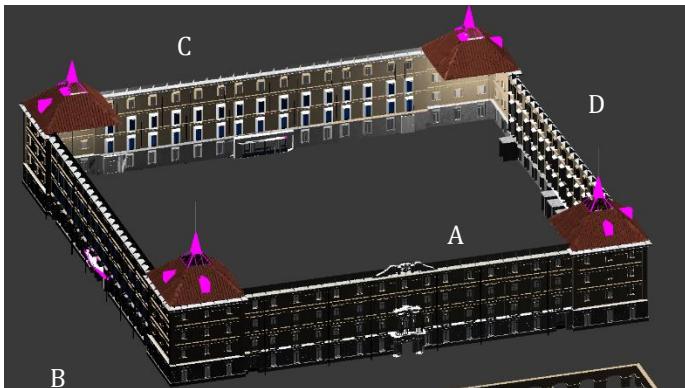


Ilustración 45: Fachada del edificio Sabatini

La fachada del edificio Sabatini supuso un reto para el equipo debido a la complejidad de la parte frontal de uno de sus lados, tiene forma rectangular, debido a la similitud de sus lados, las secciones B y D son idénticas, al igual que las secciones A y C.

La sección lateral del edificio está compuesto por cuatro series de distintos tipos de ventanas correspondientes al edificio original.



Ilustración 46: Comparación de ventanas de la fachada principal vs original

En la imagen anterior se puede apreciar:

- la colocación de la tuberías en los laterales
- la colocación exacta de las lámparas adjuntas a las puertas
- la similitud entre las ventanas y el original
- la cubierta acristalada de las puertas
- La separación entre los distintos niveles

Las esquinas eran todas iguales, lo que simplificó el proceso y permitió prestar más atención a los detalles

## DESARROLLO DE UN VIDEOJUEGO CON UDK

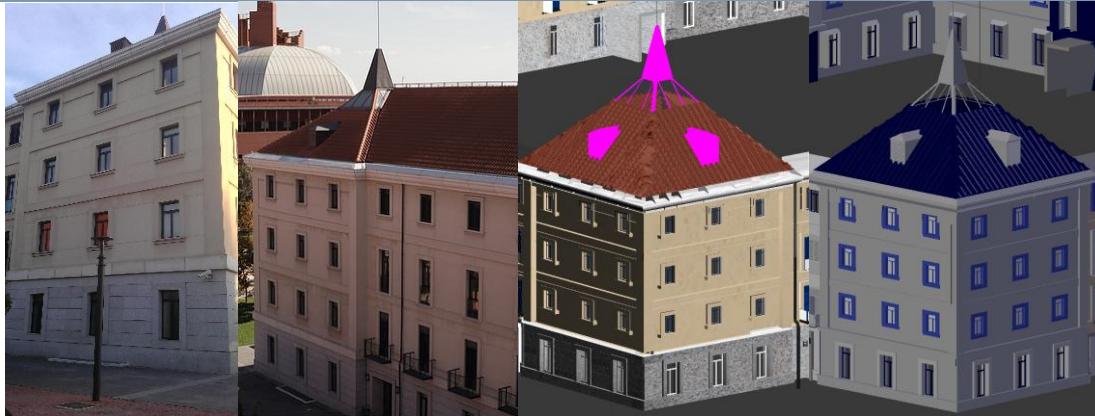


Ilustración 47: Comparación esquinas vs original

El tejado es la parte del modelo que conlleva mayor gasto computacional, lo que provocó que para acabar el diseño se tuvo que eliminar el tejado de las fachadas conservando solo el de las esquinas porque consumía toda la memoria disponible.

La fachada principal supuso un gran esfuerzo por las formas irregulares que tiene tanto en el marco de la puerta principal, como en el balcón superior y en la parte superior del tejado



## DESARROLLO DE UN VIDEOJUEGO CON UDK



Ilustración 48: Comparación de la entrada y el arco superior

Para las texturas el edificio posee muy pocas variaciones de color, y la fachada tiene 7 colores diferentes claramente distinguibles en las fotos.

### Fachada interior y patio

La fachada interior posee, al igual que la fachada externa, iguales dos a dos los laterales, la única complejidad fue añadir la estructura de la ventana y hacerlo encajar con el modelo real.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

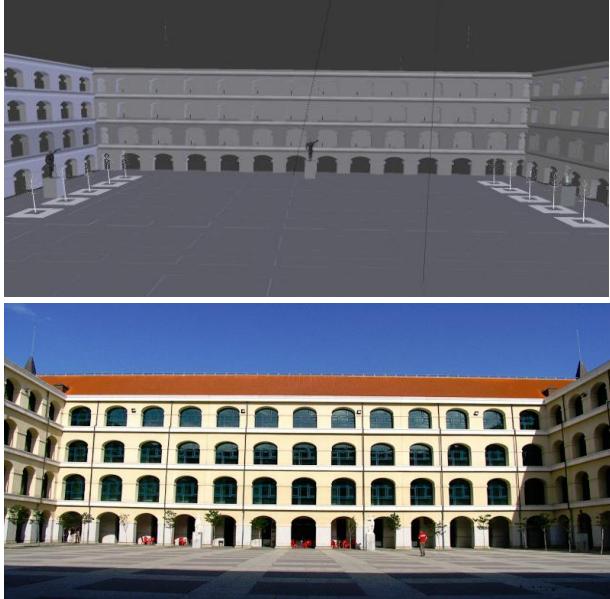


Ilustración 49: Comparación del patio interior del edificio Sabatini

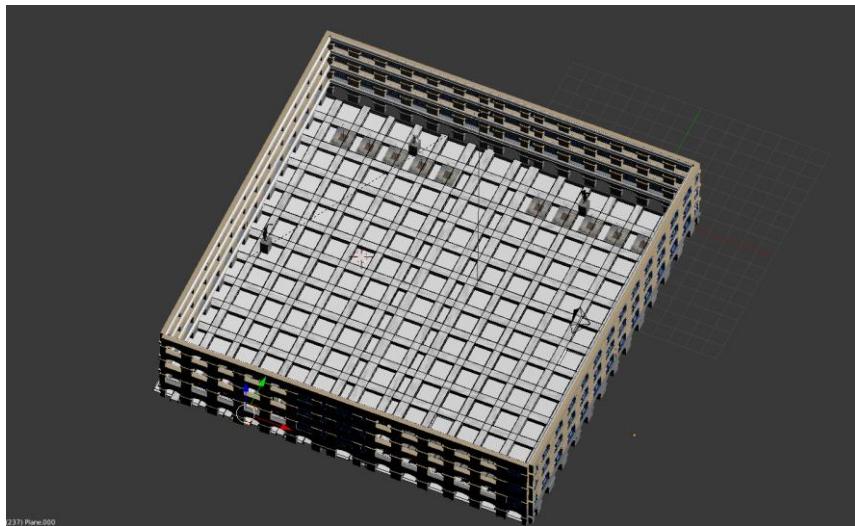


Ilustración 50: Patio del modelo del edificio Sabatini

Requiere mención especial a las esculturas realizadas.

## DESARROLLO DE UN VIDEOJUEGO CON UDK



Ilustración 51: Comparación de las estatuas del patio del edificio Sabatini

### Campus de Leganés zona verde



Para el diseño del patio se utilizó el mismo sistema que para el edificio Betancourt, con ayuda de *Google Maps* se creó un modelo de vegetación y más tarde se cubrió con una textura similar a la hierba, pero por problemas de rendimiento, esta tuvo que reemplazarse por hierba texturizada.



En la imagen se aprecia la presencia de árboles y vegetación intentando mantener la ubicación de los elementos, también es posible apreciar:

- Los bancos que rodean el patio de la cafetería
- La locomotora entre el edificio Sabatini y el edificio Betancourt
- El centro de servidores entre el edificio Sabatini y el edificio Betancourt
- La verja que rodea al parque junto a edificio Sabatini.
- La fuente en el centro del parque junto al edificio Sabatini

Ilustración 52: vista aérea de la zona verde del campus

## DESARROLLO DE UN VIDEOJUEGO CON UDK

### *Alrededores del campus*

Para los elementos restantes de los alrededores del campus, se usó tanto modelos de *SketchUp* previamente creados por A.J. Álvarez (el permiso previo aparece en el Anexo a este documento), como edificios creados mediante la combinación de *SketchUp* para obtener imágenes y *Blender* para retocarlos



Ilustración 53: modelos de la ciudad de Leganés de la biblioteca online de SketchUp

Elementos tales como vehículos, calzadas, semáforos, etc. se han obtenido de los recursos que ofrece el propio motor de juego, en el paquete gratuito y descargable (.upk) llamado *citi*

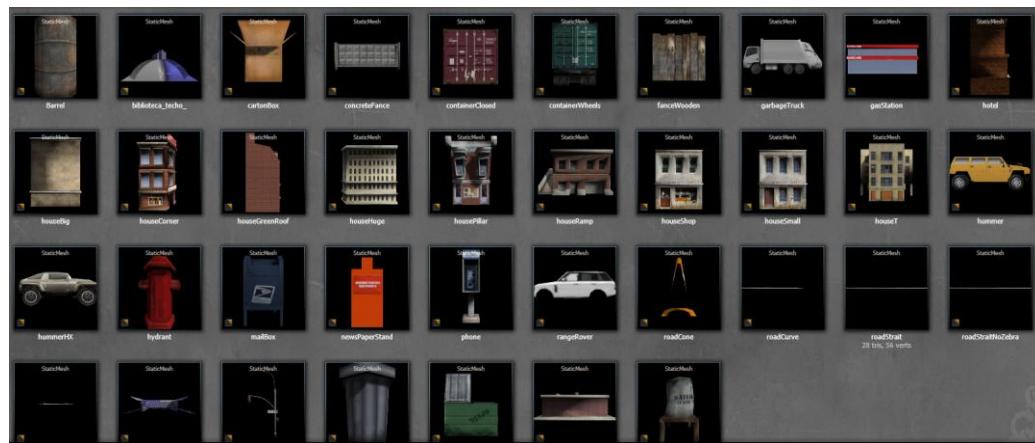


Ilustración 54: diversos elementos contenidos en el paquete descargable *citi*

### *Importación de modelos*

Para todos los modelos ya sean creados o descargados desde la librería online de *SketchUp* se ha seguido el mismo proceso:

- \*Eliminación de elementos innecesarios en *SketchUp* (solo en los modelos descargados)
- Importación y retoque en *Blender*
- Conversión de texturas
- Importación de modelo en *UDK*
- Re-asignación de texturas en *UDK*

## DESARROLLO DE UN VIDEOJUEGO CON UDK

### Eliminación de elementos innecesarios en SketchUp

Una vez descargado, con el programa *SketchUp* y se eliminan los elementos que sobren del modelo (Base, elementos sueltos, superficies ocultas)



Ilustración 55: Eliminación de elementos sobrantes en SketchUp

Tras eliminar los elementos sobrantes, se exporta en formato .3ds para importarlo a *Blender*.

\*Se podrían utilizar otras extensiones como *Collada (.dae)* o *Object (.obj)*, pero solo el formato .3ds mantiene la ruta completa de las texturas y no es necesario buscarlas y re-importarlas en *Blender*

### Importación y retoque en *Blender*

Tras importarlo a *Blender* es necesario inspeccionarlo en busca de fallos gráficos (*glitch*) o de texturas mal aplicadas.



Ilustración 57: modelo importado a Blender



Ilustración 56: fallo en la importación

A primera vista parece correcto, sin embargo, un examen más minucioso revela que tiene sectores nulos

Estos espacios han de ser llenados y solucionados antes de exportar el modelo al motor de juego, a la hora de convertir a otro formato, en ocasiones se pierden polígonos, vértices o se inviertan normales de texturas, lo que provoca que no se muestre correctamente, y tenga que ser re-editado.

TRABAJO FIN DE GRADO

# DESARROLLO DE UN VIDEOJUEGO CON UDK

En el modelo anterior, se unen los vértices del hueco para generar una cara, y con el método *UVW unwrap* se le asigna la textura del modelo inicial

Tras arreglar todos los fallos que pueden aparecer en el modelo, es necesario exportarlo en formato *.FBX* para que sea insertado en el juego.



Ilustración 58: modelo con el fallo solucionado

## Conversión de texturas

Antes de abrir el editor *UDK* es necesario importar las texturas en un formato compatible con el programa, en este caso, se eligió el formato *.PNG* y *UDK* recomienda un tamaño potencia de base 2. Estas transformaciones se realizan de manera sencilla usando el programa *ImRe*.

Gracias a este programa, convertir multitud de imágenes en otros formatos y con unas dimensiones determinadas es muy rápido y sencillo.

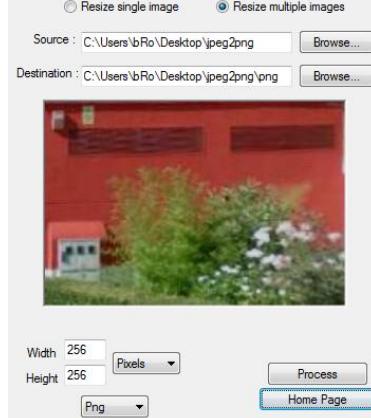


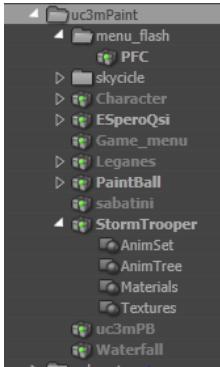
Ilustración 59: interfaz ImRe



Ilustración 60: conversión de formato y dimensiones

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## Importación de modelo en UDK



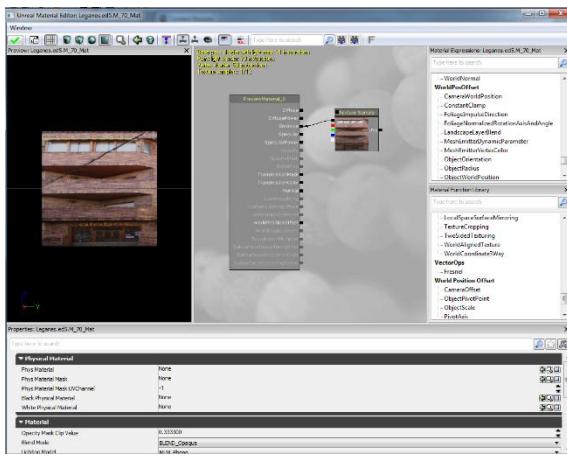
**Ilustración 61:**  
directorio del paquete uc3mPaint

Para importar modelos, se hace uso del *Content Browser*, que sirve como biblioteca de elementos para el entorno *UDK*, el procedimiento consiste en seleccionar (o crear) un paquete .upk donde se guardara el modelo importado, importar el fichero y guardar el paquete, que en este caso, lleva el nombre de *uc3mPaint*, este paquete contiene todos los elementos digitales que utiliza el juego para funcionar, por lo que se comprime en forma de paquete .upk y se almacena en el subdirectorio *UDKGame/Content/uc3mPaint/* cada subdirectorio, corresponde a secciones distintas del juego como menús flash, ciclo dia-noche, personajes, texturas, modelos, armas, paisajes, etc. Los originales pueden ser eliminados, ya que todo se almacena dentro del paquete al guardarse.

## Re-asignación de texturas en UDK

Debido a que los modelos originales creados en *SketchUp* utilizan texturas en formato *JPEG* y *UDK* no lo reconoce, es necesario re-aplicar las texturas a los modelos importados, ya que ahora las texturas están en *.PNG* y en el código asociado al modelo, asigna texturas con extensión *.JPEG*

Una vez importadas las texturas, es necesario crear el material a partir de ellas, para ello se utiliza el editor que viene en *UDK*



**Ilustración 62:** creación de material

Primero se crea un nuevo material al que asociamos la textura con la propiedad *emmissive*, lo que permite que emita la textura seleccionada. Este editor es muy potente, y la combinación de normales, con colores, gammas, filtros y uso de las distintas propiedades permite que las posibilidades sean infinitas y con resultados muy realistas.

Tras haber creado los materiales (uno por cada textura), es necesario asignar cada material a la zona correspondiente del modelo.

TRABAJO FIN DE GRADO

## DESARROLLO DE UN VIDEOJUEGO CON UDK

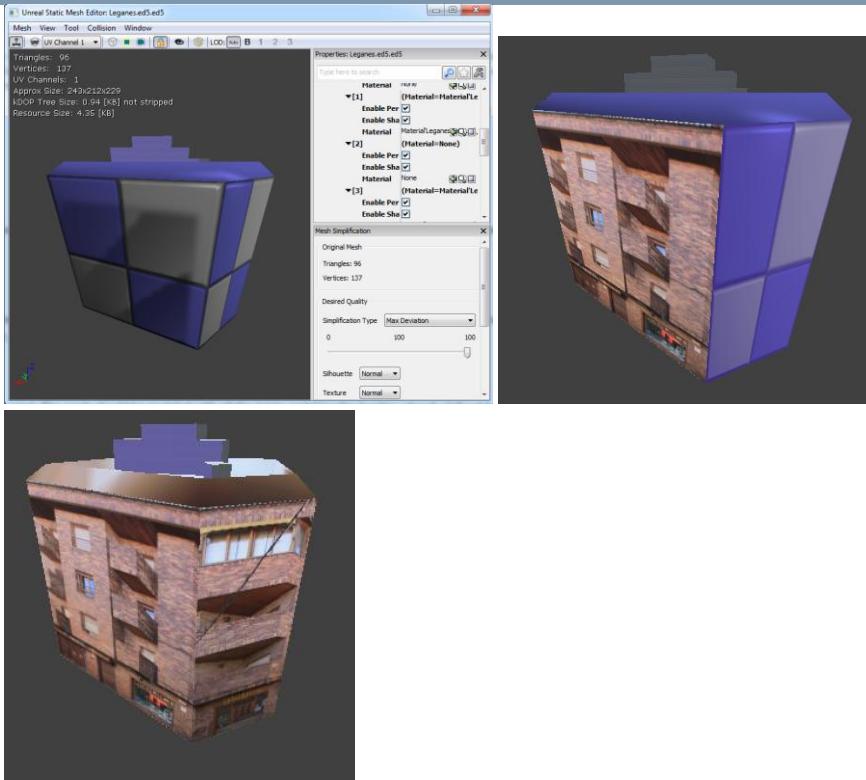


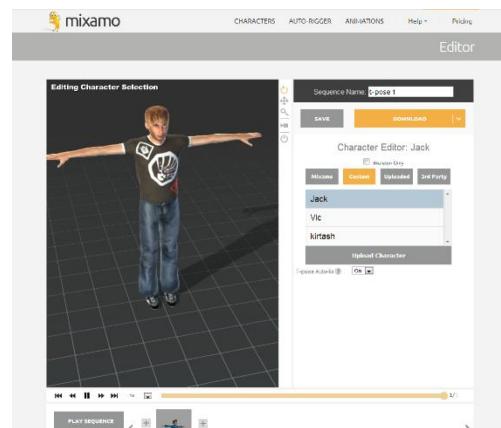
Ilustración 63: asignación de materiales al modelo

## CREACIÓN DE PERSONAJES

Para la creación del personaje se usó la idea de un estudiante común con vaqueros y camiseta, el diseño se ha realizado en *Mixamo* por tener más recursos en línea y por tener ya experiencia previa utilizando este editor online.

Utilizando un modelo base, se seleccionó entre distintos modelos de cabello, camiseta, color de piel, vaqueros, calzado dando como resultado el personaje que se aprecia en la imagen.

Desde la web, es posible descargarse el modelo en formato *.FBX* lo que permite importarlo en *3ds Max* para poder insertarle un esqueleto y aplicar las animaciones nativas del motor de juego.



# DESARROLLO DE UN VIDEOJUEGO CON UDK

Ilustración 64: editor de modelos 3D Mixamo

## Rigging de personaje

No basta con tener modelado un personaje, para poder aplicarle animaciones y que el personaje se mueva, es necesario aplicarle un esqueleto que permita asociar el modelo con unos huesos, esta técnica recibe el nombre de *rigging* y para consta de los siguientes pasos:

- Descarga del modelo de la página oficial de UT para aplicarla al personaje
- Aplicación del esqueleto sobre el modelo
- Asociación de los huesos con la parte del modelo afectada

### Descarga del modelo de la página oficial de UT para aplicarla al personaje

Download the master skeleton max files here:

- [UT3\\_Male.max](#) (3D Studio Max 9)
- [UT3\\_Female.max](#) (3D Studio Max 9)
- [UT3\\_Krall.max](#) (3D Studio Max 9)
- [UT3\\_Corrupt.max](#) (3D Studio Max 2009)

Lo primero es descargar el esqueleto oficial que ofrece de manera gratuita *Epic Games* en su página web (<http://udn.epicgames.com/Three/UT3CustomCharacters.html>)

Es posible descargar distintos tipos de esqueleto en función del género o si es humano u otro tipo de ser.

### Aplicación del esqueleto sobre el modelo

El archivo contiene el modelo del personaje principal de *Gears of War* con el esqueleto asociado, así pues, es necesario eliminar el modelo y separar el esqueleto para poder aplicarlo al personaje creado.

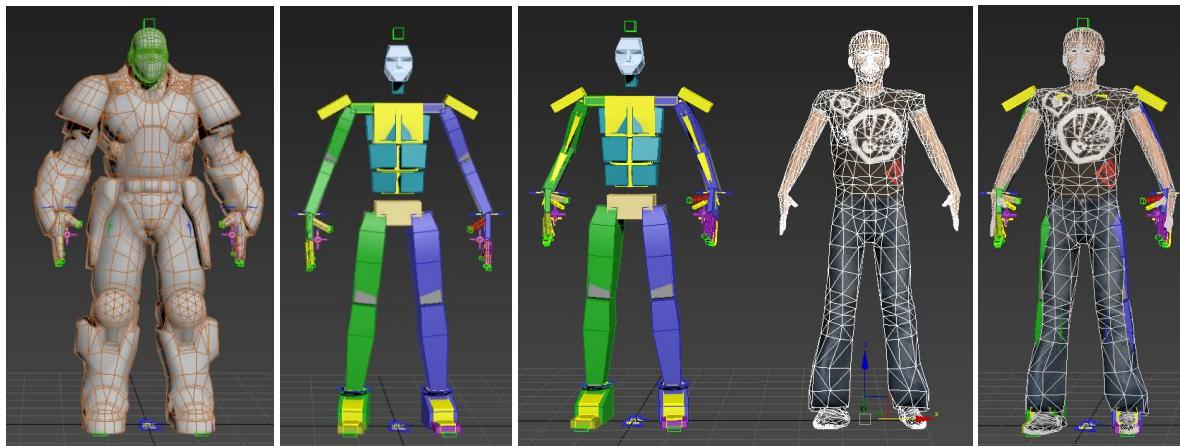


Ilustración 65: disociación del modelo de UT3 y posterior fusión con el modelo creado

### Asociación esqueleto-modo

Una vez importado el esqueleto al modelo y adaptadas las proporciones y la posición de las extremidades, concluimos asignando que partes del modelo se mueven con el movimiento de cada hueso. Este paso es el más delicado ya que si no se hace bien el movimiento de un brazo puede significar que también se mueva

## DESARROLLO DE UN VIDEOJUEGO CON UDK

parte de una pierna, lo que provoca resultados inhumanos, realizar correctamente este apartado conlleva horas de atención si se quiere obtener un resultado óptimo, fluido y natural



Ilustración 66: modelo mal riggeado y bien riggeado

Como se puede observar en la imagen superior, el modelo de la izquierda, no tiene bien definida el área de la mano ni ajustada la pierna, lo que provoca distorsiones en el movimiento, el modelo de la derecha si tiene correctamente asignadas las relaciones esqueleto-modelo.

### *Asignación de movimientos*

Una vez importado el modelo (con el esqueleto ya asociado) en *UDK*, es necesario comprobar si el modelo se mueve con las animaciones del motor *UDK*, para ello, *UDK* incorpora un editor de animaciones y personajes donde es posible editar los parámetros asociados a un esqueleto

## TRABAJO FIN DE GRADO

# DESARROLLO DE UN VIDEOJUEGO CON UDK



Ilustración 67: diversos movimientos que puede realizar el personaje

### Creación de enemigo

Al igual que en el apartado anterior, se diseñó el enemigo, en este caso y para que fuera más rápido el proceso, se han reutilizado dos modelos de proyectos anteriores, ya riggeados

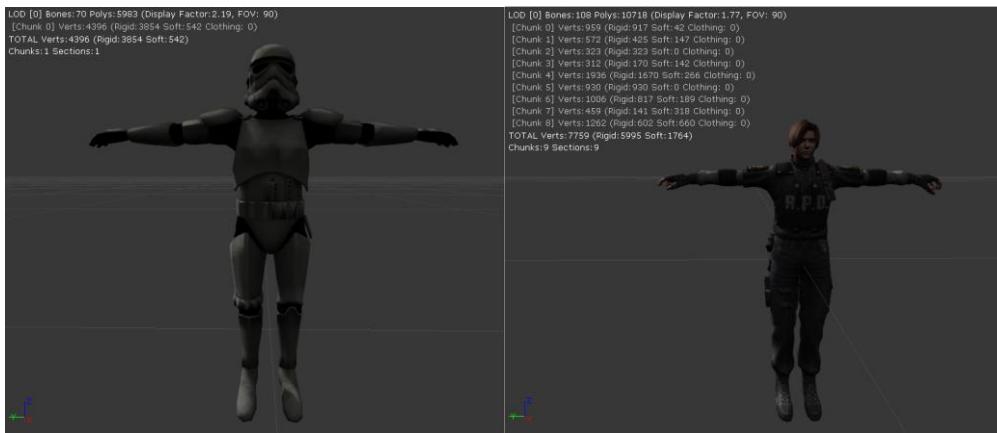


Ilustración 68: modelos de los enemigos

## CREACIÓN DE ARMA

El arma se creó en *Blender*, se utilizó como inspiración un arma real de paintball a la que se ha añadido el cargador superior con forma ovalada y una terminación que emula bomba de presión e aire que contienen las pistolas de paintball.

TRABAJO FIN DE GRADO

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## Riggeo del arma

Al igual que con el personaje, el arma ha de ser riggeada para que, en caso de que se añada alguna animación al arma, también pueda ser aplicada en el juego. En este caso, al no tener ningún tipo de animación, se le ha colocado un socket en el gatillo para que el jugador a la hora de coger el arma, se posicione correctamente en su mano.



Ilustración 69: arma del juego

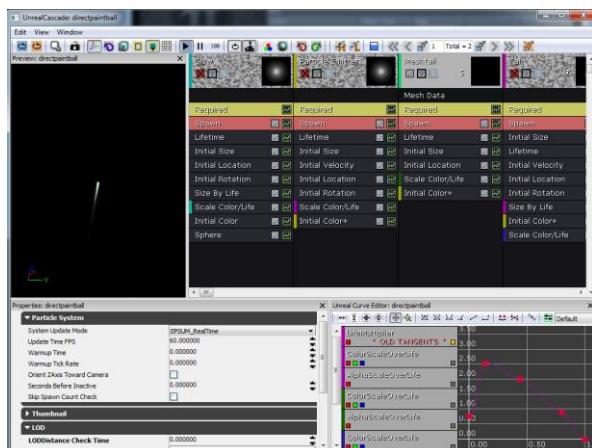


Ilustración 70: editor de partículas

## GENERACIÓN DE CINEMÁTICAS

Para la generación de cinemáticas se ha usado el sistema que incorpora el motor UDK, *Unreal Matinée*, se ha realizado una cinemática de introducción donde la cámara realiza un pequeño recorrido por una calle y se eleva hasta enfocar el edificio *Sabatini* desde una vista casi vertical.

Cada triángulo rojo, supone la posición de la cámara en un instante determinado de tiempo, para crear una posición, basta con mover la cámara por el espacio e indicar a *Matinée* en qué posición debe situarse en el instante  $t'$ .

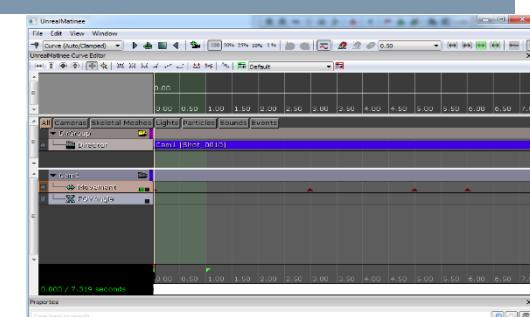


Ilustración 71: editor de secuencias Matinée

TRABAJO FIN DE GRADO

## DESARROLLO DE UN VIDEOJUEGO CON UDK

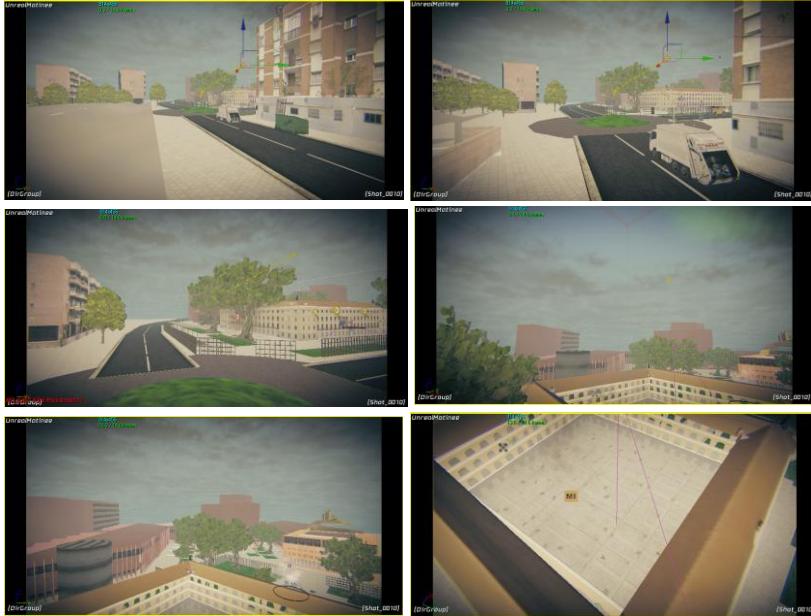


Ilustración 72: secuencia de introducción

### GENERACIÓN DE MENÚS INTERACTIVOS

Para crear los menús se ha utilizado la tecnología *Flash* usando el software Adobe Flash, se compone de las siguientes etapas:

- Creación de la interfaz
- Creación del código

#### *Creación de la interfaz*

Tal y como se vio en el apartado de diseño, los menús se componen de

- Un logo
- Identificador de menú
- Acciones
- \*HUD (*Head-Up Display*)
- Cursor

#### Logo

Para crear el logo se diseñó usando el software *GIMP* utilizando como base, el logo de la universidad, al que se le añadió disparos de pintura

## DESARROLLO DE UN VIDEOJUEGO CON UDK



Universidad  
Carlos III de Madrid



Ilustración 73: logo UC3M

### Identificador de menú

El menú principal se diferencia del menú de pausa en la palabra que aparece bajo el logo, "Paintball" indica menú principal, el que aparece al iniciar la aplicación, el menú de pausa solo aparece si el jugador durante el transcurso de la partida pulsa la tecla "P"



Ilustración 74: identificadores menú principal y menú pausa

### Acciones

Todas las acciones (*Play, options, exit, resume y quit*) tienen el mismo formato, un botón en azul con fondo rojo que al detectar una posición del ratón sobre él (*hover*) se transforma en azul

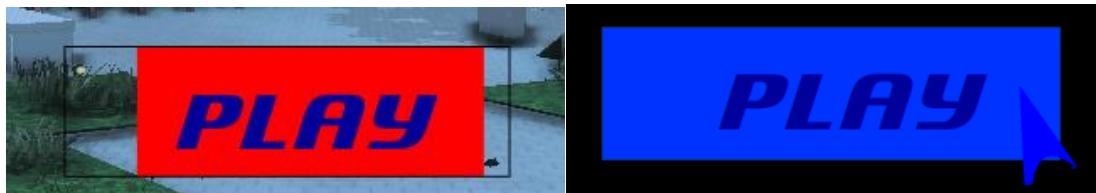


Ilustración 75: botón de acción normal y con hover

### \*HUD (Head-Up Display)

El *HUD* ha de mostrarse siempre durante la partida, ya que muestra información de importancia para el jugador, como la munición, el mapa, las armas o la salud.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

En este proyecto, el *HUD* se ha creado desde cero, permitiendo aligerar la versión que viene por defecto en el *UDK*



Ilustración 76: posición del *HUD* durante la partida

El *HUD* se sitúa en la parte superior derecha de la pantalla, e informa al jugador sobre la salud del personaje y la munición restante



Ilustración 77: elementos del *HUD*

### Creación del código

Para que los botones puedan realizar acciones es necesario dotarlos de código interactivo, este código debe realizarse tanto en la parte asociada a la animación, como en la parte de *UDK*.

- Código *flash*
- Código *UDK*

#### Código *flash*

Para el código *flash* se realiza en la propia aplicación, este código sirve para señalar qué elementos del menú, pueden ser indexados para realizar acciones.

```
onClipEvent(enterFrame) {
    _x = _root._xmouse;
    _y = _root._ymouse;
}
on (press) {
    fscommand("loadmap");
}
```

Por ejemplo, el código de la imagen, sirve para señalar a UDK que el elemento *cursor* del menú se mueve usando la posición del ratón.

O el código que indica que al presionar el elemento *play* del menú principal, carga la acción *loadmap*

#### Código *UDK*

Una vez importado en el entorno de videojuegos, es necesario utilizar la herramienta *Kismet* que permite generar código, asociado a un mapa en concreto, de una forma visual.

En el apartado siguiente se entrará más en profundidad en el funcionamiento de *Kismet* ahora solo se explicara la parte referida a la correcta ejecución del menú

TRABAJO FIN DE GRADO

## DESARROLLO DE UN VIDEOJUEGO CON UDK

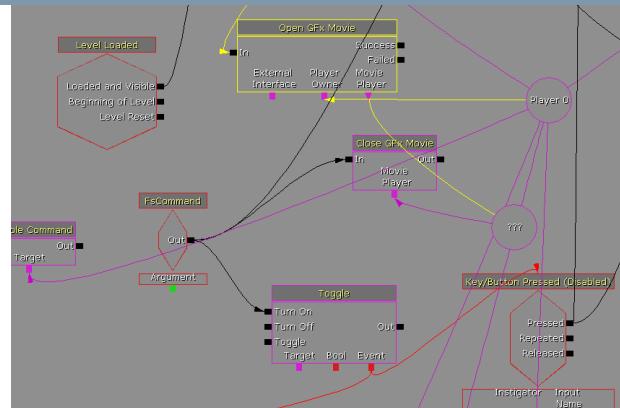


Ilustración 79: código en UDK usando Kismet

Para el código en *UDK*, primero se usa el evento *Open GFx Movie* para cargar la animación *flash*, con el evento *FsCommand* señalamos al sistema que la acción *loadmap* cierra el menú y empieza la cinematica que da comienzo a la partida

| ▼ GFx Event FSCommand |                         |
|-----------------------|-------------------------|
| Movie                 | SwfMovie'PFC.UC3MPB-MM' |
| <b>FSCommand</b>      | <b>loadmap</b>          |

Ilustración 78: asociación de eventos en UDK

La animación no tiene fondo, por lo que se usa el propio entorno del videojuego, lo que le da un toque de interacción y creatividad pudiendo tener un menú con distintos fondos animados en función del momento de la partida



Ilustración 80: menú con sol, lluvia y de noche

### UNREAL DEVELOPMENT KIT: ENTORNO DE DESARROLLO

El directorio de instalación básico de *UDK* está compuesto de 4 carpetas:

## DESARROLLO DE UN VIDEOJUEGO CON UDK



Ilustración 81: directorio *UDK*

- **Binaries:** Contiene los ejecutables del juego y programas adicionales, solo se utilizara para realizar una vista previa del juego entero, que corresponde a la última etapa de desarrollo
- **Development:** Contiene el código fuente del motor *UnrealScript* y es donde se incluirá el código que ejecutara el juego
- **Engine:** Ficheros del motor, usado en la fase de exportación, este directorio no debe ser modificado.
- **UDKGame:** contiene la biblioteca de elementos que utiliza el juego (modelos, personajes, armas, animaciones, menús, etc.)

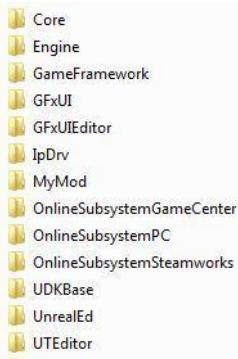


Ilustración 82: subdirectorio  
/src/

El entorno *UDK* ofrece diversos tipos de juego, sujetos a un sistema de clases, subclases y herencias que permite realizar o no diversas acciones. Esta estructura tiene la ventaja de que es muy sencilla de entender, pero también tiene la complicación de que existen funciones muy útiles en subclases que no son accesibles desde otras ramas.

Lo primero que se ha de hacer, es crear un tipo de juego propio, ya que la idea del proyecto es usar la herramienta para crear un juego sobre su base y no modificar algo ya existente.

Para crear un tipo de juego nuevo, es necesario crear un directorio en la ruta */Development/src*, y configurar el IDE para que indexe los directorios nuevos.

Cada uno de estos directorios conforma paquetes de código que ejecuta el motor, algunos de ellos, son definiciones de tipos de juego, otros son módulos que implementan el sistema multijugador, módulos del núcleo, o interacción con menús flash. Por tanto, se procede a crear dos directorios, uno de ellos implementara el juego en tercera persona y el otro en primera (lo que conlleva una implementación radicalmente distinta)

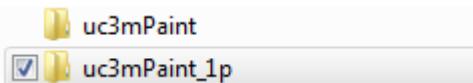
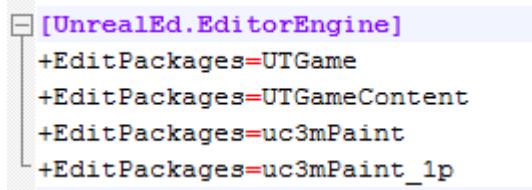


Ilustración 83: nuevas carpetas creadas

## DESARROLLO DE UN VIDEOJUEGO CON UDK

Estos directorios contienen una carpeta llamada */Classes* que es donde se almacena todo el código que caracteriza cada tipo de juego, aspectos como la interfaz, el personaje, el arma, la posición de la cámara, la salud, la munición, el tipo de daño, son definidas en este directorio.

Tal y como se explicó al comienzo del tema, en el archivo */UDKGame/Config/DefaultEngine.ini* se declara la ubicación de los elementos externos que utilizará el motor y la configuración básica del entorno de desarrollo (mapeado de teclas, configuración gráfica, etc.) En la sección donde se especifica los paquetes a indexar, se inserta la ruta de los nuevos directorios creados



```
[UnrealEd.EditorEngine]
+EditPackages=UTGame
+EditPackages=UTGameContent
+EditPackages=uc3mPaint
+EditPackages=uc3mPaint_1p
```

Ilustración 84: edición del archivo Defaultengine.ini

Tras reiniciar la aplicación, ya indexa el código que generaremos dentro de esos directorios.

A continuación, se explicará en detalle cada clase que conforma el código del proyecto, explicando dependencias y relaciones, en la siguiente figura, se explican las relaciones entre las distintas clases.

# DESARROLLO DE UN VIDEOJUEGO CON UDK

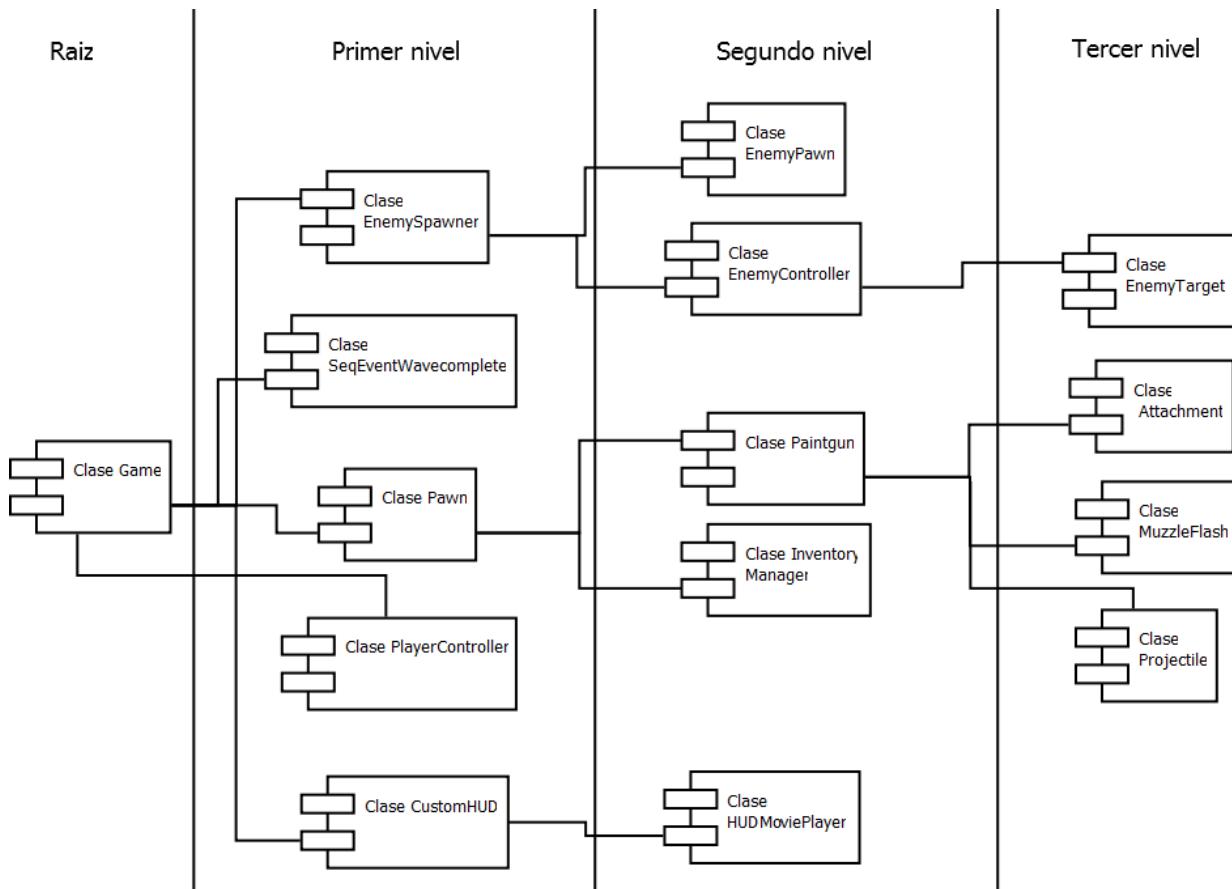


Ilustración 85: herencia de clases del proyecto

## Cambiando el tipo de juego

Para que reconozca el nuevo tipo de juego, es necesario generar un nuevo fichero en el directorio nuevo *Classes/uc3mPaint\_1p* que extienda de *UDKGame* o de alguna de sus subclases, al reiniciar el programa, busca en los subdirectorios especificados en *UDKGame/Config/DefaultEngine.ini* todas las clases que extiendan de la superclase *UDKGame* y los asigna como tipos de juego.

```

DefaultEngine.ini uc3mPaint_1p_Game.uc
1 //class uc3mPaint_Game extends UTDeathMatch;
2 class uc3mPaint_1p_Game extends UDKGame;
  
```

Ilustración 86: herencia de clases

El problema de tener el mismo juego en primera o tercera persona, es que heredar de la subclase *UTDeathMatch*, UDK elimina la posibilidad de usar tu propia interfaz, incorpora la que viene por defecto en la subclase, e impide la modificación del sistema de juego al no aceptar condicionales.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

Si se desea un juego en tercera persona, es necesario heredar de la clase *UDKGame* e implementar desde cero, la interfaz, el personaje y la cámara.



Ilustración 87: interfaz y visualización por defecto

Para probar el *GameType* es necesario recomilar los scripts, por suerte, UDK cada vez que se inicia comprueba si se han efectuado cambios y en caso afirmativo, recompila los scripts. Para probar que el sistema ha reconocido correctamente el tipo de juego (*GameType*) al abrir el mapa por defecto, en la opción *view -> World Properties* aparece nuestro tipo de juego

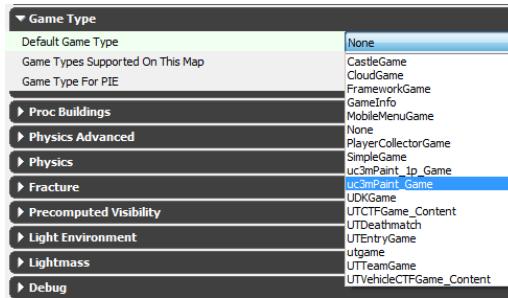


Ilustración 88: asignación de tipo de juego al mapa

### Personaje principal

Para comenzar con el código, es necesario indicarle el modelo que se va a usar, así como el controlador que maneja al personaje, las armas, tipo de disparo, etcétera. Para ello es necesario explicar dos de las clases más importantes de juego: *Pawn* y *PlayerController*

- **Pawn:** Es la clase que representa al personaje principal (peón) el muñeco que manejará el jugador, la clase *PlayerController* se encarga del movimiento del personaje y acciones de juego mientras que la clase *Pawn* controla la cámara, asocia el modelo, animaciones entorno, colisiones y propiedades de locomoción (como le afecta la gravedad, velocidad a la que puede correr, control aérea, etc.)

## DESARROLLO DE UN VIDEOJUEGO CON UDK

- **PlayerController:** es la clase responsable de trasladar las acciones del jugador en acciones dentro del juego, moviendo su personaje, la cámara asociada o capturar eventos durante el juego y sus acciones.

Ambas clases son necesarias e imprescindibles para cualquier juego desarrollado en *UDK*, y al igual que se hizo con la clase *uc3mPaint\_1p\_Game*, se creara una versión propia que extienda de estas clases para adaptarlas al proyecto.

### Cambiando el personaje

La clase por defecto a heredar es *UTPawn*, que utiliza el personaje por defecto de *Unreal Tournament* por tanto es necesario que herede de *UDKPawn* que es el padre de la anterior y más genérica.

```
defaultproperties
{
    // Iluminación de entorno del Pawn

    // Componente SkeletalMesh para el robot
    Begin Object Class=SkeletalMeshComponent Name=SkeletalMeshComponentRobot
        SkeletalMesh=SkeletalMesh 'StormTrooper.estudianteMEN'
        //AnimSets(0)=AnimSet 'CH_AnimHuman.Anims.K_AnimHuman_BaseMale'
        //AnimTreeTemplate=AnimTree 'CH_AnimHuman_Tree.AT_CH_Human'
        //SkeletalMesh=SkeletalMesh 'StormTrooper.StormTrooper'
        AnimSets(0)=AnimSet 'StormTrooper.AnimSet.Storm_animset'
        AnimTreeTemplate=AnimTree 'StormTrooper.AnimTree.Storm_tree'
```

Ilustración 89: propiedades por defecto en uc3mPaint\_1p\_Pawn

La ruta que aparece entre comillas corresponde al paquete *upk* del *Content Browser* donde se encuentra el objeto a asociar.

- *SkeletalMesh*: asocia el modelo+esqueleto creado anteriormente
- *AnimSet*: permite decirle al juego que el nombre de los huesos presentes en el modelo se puede asociar a los establecidos por defecto en el UT
- *AnimTree*: asocia el conjunto de animaciones por defecto de UT al *SkeletalMesh*

### Cambio de cámara a tercera persona

En el diseño del juego se especificó que debía ser en primera/tercera persona, para aplicar la tercera persona hay que modificar la cámara por defecto, ya que incorpora una cámara subjetiva donde solo se visualizan los brazos del personaje y el arma.

La clase *uc3mPaint\_1p\_Pawn.uc* sobrescribirá la función de la cámara en primera persona por defecto y la reemplaza por la vista en tercera persona

## DESARROLLO DE UN VIDEOJUEGO CON UDK

```
// Sobreescrive una función para que se vea al pawn por defecto
// La llama la Camara cuando este actor se convierte en su ViewTarget
simulated event BecomeViewTarget( PlayerController PC )
{
    local UTPlayerController UTPC;

    // Por defecto esta llamada pone los brazos al Pawn y el arma
    Super.BecomeViewTarget(PC);

    if (LocalPlayer(PC.Player) != None)
    {
        UTPC = UTPlayerController(PC);
        if (UTPC != None)
        {
            // Activa la vista trasera (Poner la camara en tercera
            UTPC.SetBehindView(true);
            // Esto no es necesario
            //SetMeshVisibility(UTPC.bBehindView);
        }
    }

    defaultproperties
    {
        //Determina el 'zoom' de la cámara
        CameraScale=40.0

        // Se sobreescribe el valor del CamOffset por defecto CamOffset
        CamOffset=(X=4.0,Y=0.0,Z=-13.0)
    }
}
```

Ilustración 90: cambio cámara

Por último, es necesario modificar el archivo *uc3mPaint\_1p\_Game.uc* para avisar al sistema de que vamos a hacer uso de las clases propias que se han creado.

```
defaultproperties
{
    TotalEnemiesAlive=0;
    //acronym = "UC3M1"
    //Se especifica el Pawn por defecto
    DefaultPawnClass=class'uc3mPaint_1p.uc3mPaint_1p_Pawn'
    //Se especifica el controlador
    PlayerControllerClass=class'uc3mPaint_1p.uc3mPaint_1p_PlayerController'
```

Ilustración 91: propiedades por defecto en el archivo *uc3mPaint\_1p\_Game*

### *Creando el enemigo*

Para crear el enemigo, se realiza un proceso similar al del personaje principal, para ello se hace uso de las clases **EnemyPawn**, **EnemyTarget** y **EnemyController**

Las principales diferencias de estas clases respecto a las anteriormente explicadas son:

- En la clase **EnemyPawn** se introduce el atributo *placeable* que permite colocar un objeto del tipo enemigo en cualquier parte del escenario.
- *UnrealScript* integra en el propio lenguaje las máquinas de estados, lo que permite definir estados en pocas líneas. En la clase **EnemyController** se definen varios estados:
  - *Init*: detecta el personaje principal y si entra en su rango de visión, se dirige a él.
  - *GoToFinalTarget*: si el mapa presenta una estructura compleja, el enemigo reconoce el terreno y detecta la ruta más corta para llegar hasta el personaje principal

## DESARROLLO DE UN VIDEOJUEGO CON UDK

- *AtFinalTarget*: al colisionar con el personaje principal, en función del enemigo se realizan acciones distintas, un tipo de enemigo estalla al colisionar y otro tipo le ataca.

```
state GoToFinalTarget
{
    // Calcula el camino al objetivo
    function bool FindNavMeshPath()
    {
        // Clear cache and constraints (ignore recycling for the moment)
        NavigationHandle.PathConstraintList = none;
        NavigationHandle.PathGoalList = none;

        // Create constraints
        class'NavMeshPath_Toward'.static.TowardGoal( NavigationHandle, FinalTarget );
        class'NavMeshGoal_At'.static.AtActor( NavigationHandle, FinalTarget, 32 );

        // Find path
        return NavigationHandle.FindPath();
    }

    Begin:
    if( NavigationHandle.ActorReachable(FinalTarget) )
    {
        // Si se puede llegar directamente hacerlo:
        MoveTo(FinalTarget.Location, FinalTarget, 32);
        GoToState('AtFinalTarget');
    }
    else if (FindNavMeshPath())
    {
        NavigationHandle.SetFinalDestination(FinalTarget.Location);

        if(NavigationHandle.GetNextMoveLocation(NextDest, Pawn.GetCollisionRadius()*5))
        {
            // Ir al siguiente punto del camino
            MoveTo(NextDest,none,128.0);
        }
        else
        {
            // No hay punto siguiente
            MoveToward(FinalTarget);
        }
    }
}
```

Ilustración 92: definicion del estado GoToFinalTarget

- La clase **EnemyTarget** define el radio de colisión del objetivo.

### Generador de enemigos

El juego contiene un generador de enemigos, que permite al jugador cuando quiere enfrentarse a estos enemigos. Para ello, es necesario crear un Actor en el que se define un área con un nombre específico, cuya función será la de generar un número definido de un tipo de enemigos, este actor es la clase **EnemySpawner**

```
class uc3mPaint_1p_EnemySpawner extends Actor
placeable;

// Cilindro en el que se hará el spawn de los enemigos, editable en el editor
var() editconst const CylinderComponent CylinderComponent;

// Enemigos que quedan por "spawnear"
var int enemiesLeft;

// Tiempo entre Spawns
var float timeInterval;
```

Ilustración 93: definición del spawner

El estado principal de esta clase es *Spawning* que constantemente calcula si queda algún enemigo más por crear y llama al método *calculateNewSpawnlocation* que permite seleccionar un área aleatoria no ocupada por otro objeto del cilindro para generar un enemigo. La función *Spawnuc3mPaint\_1p\_Enemy* asocia el modelo y sus propiedades al tipo de objeto que debe aparecer en el spawner, lo que permite reutilizar esta clase para crear un generador de cualquier tipo de objeto: municiones, viales de salud, armas, etcétera.

La subclase **EnemyTarget** permite definir otro tipo de cilindro para el generador de enemigos.

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## *Inventario del personaje*

UDK incorpora un sistema de inventario para almacenar y gestionar los objetos del personaje principal, para ello, provee una clase llamada **InventoryManager** que permite acceder a los métodos para gestionar estos objetos.

Para el caso de las armas, en el videojuego existen tres tipos de armas con distintos tipos de disparo y efectos, dos de ellas son propias de *UT3* por lo que no necesitan ser gestionadas, solo indicar su uso. Para la última, creada desde cero, es necesario hacer uso de la clase *Paintgun* que extiende de la clase *UTWeapon*.

## *Arma principal*

Para señalar los dos tipos de disparo que tiene el arma se hace uso del método *playfireeffects* que permite tener el disparo básico que simula una bola de pintura, y el disparo automático

```
defaultproperties
{
    // Weapon SkeletalMesh
    Begin Object class=AnimNodeSequence Name=MeshSequenceA
        bCauseActorAnimEnd=true
    End Object

    // Weapon SkeletalMesh
    Begin Object Name=FirstPersonMesh
        SkeletalMesh=SkeletalMesh'PaintBall.Mesh.PaintBall_1p'
        //Rotation=(Roll=16300)
        scale=0.8
        //FOV=60.0
    End Object

    AttachmentClass=class'uc3mPaint_1p_Attachment'

    WeaponFireSnd[0]=SoundCue'PaintBall.Sound.PaintBall_Shoot_Cue'
    MaxDesireability=0.65
    AmmoCount=1000
    LockerAmmoCount=1000
    MaxAmmoCount=1000
    MuzzleFlashSocket=MF
    MuzzleFlashPSCTemplate=ParticleSystem'VH_Manta.Effects.PS_Manta_
        MuzzleFlashColor=(R=200,G=0,B=0,A=255)
    MuzzleFlashDuration=0.25
    MuzzleFlashLightClass=class'uc3mPaint_1p_MuzzleFlash'
    CrossHairCoordinates=(U=256,V=0,UL=-64,VL=64)
    LockerRotation=(Pitch=32768,Roll=16384)
    IconCoordinates=(U=728,V=382,UL=162,VL=45)

    FiringStatesArray(0)=WeaponFiring
    WeaponFireTypes(0)=EWFT_Projectile
    WeaponProjectiles(0)=class'uc3mPaint_1p_Projectile'
    FireInterval(0)=0.1

    InventoryGroup=11
    GroupWeight=0.5
}
```

Ilustración 94: propiedades de la clase paintgun

parámetros del disparo secundario que utiliza proyectiles.

Cada uno de estos atributos llama a las clases encargadas de definir sus propiedades.

En la sección de propiedades de esta clase se asocia el modelo del arma, así como el resto de propiedades:

*Attachment* sirve para indicar en qué lugar del personaje principal ha de situarse el objeto, ya que no necesariamente debe ser un arma de mano, para su colocación es necesario señalar el *socket*, que es la zona del personaje principal donde se sitúa este arma.

*Soundcue* establece la ruta donde se encuentra el sonido asociado al disparo

Las siguientes propiedades indican el tamaño, el número de disparos, capacidad del cargador, etcétera.

*Muzzleflash* indica el sistema de partículas que utiliza el arma, tanto para el rastro que produce al disparar, como el efecto posterior

Por último, *projectile* indica al sistema la clase encargada de ajustar los

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## Creación de HUDs

La clase *customHUD* que extiende de *UTHUDBase* invoca al método *Destroyed()* que cierra la animación si no la encuentra y al método *PostBeginPlay()* que lanza la animación flash del menú principal.

Para hacer uso de los botones definidos en la animación flash, es necesario que esta clase extienda de *GFxMoviePlayer* que permite hacer uso del plugin que reconoce los identificadores de la animación y los enlaza con *UDK*.

```
//Called from STHUD's PostBeginPlay()
function Init(optional LocalPlayer PC)
{
    //Start and load the SWF Movie
    Start();
    Advance(0.f);

    //Set the cache value so that it will get updated on t
    LastHealthpc = -201;
    LastAmmoCount = -201;
    LastWeapon = none;

    //Load the references with pointers to the movieClips
    HealthMC = GetVariableObject("_root.Health");
    HealthBarMC = GetVariableObject("_root.Health.Bar");
    HealthTF = GetVariableObject("_root.Health.Value");

    AmmoMC = GetVariableObject("_root.Ammo");
    AmmoBarMC = GetVariableObject("_root.Ammo.Bar");
    AmmoTF = GetVariableObject("_root.Ammo.Value");

    WeaponMC = GetVariableObject("_root.Weapon");
}

super.Init(PC);
}

//If the cached value for Health percentage isn't equal to the current one
if (LastHealthpc != getpercentage(UTP.Health, UTP.HealthMax))
{
    //... Make it so...
    LastHealthpc = getpercentage(UTP.Health, UTP.HealthMax);
    //... Update the bar's xscale (but don't let it go over 100)...
    HealthBarMC.SetFloat("_xscale", (LastHealthpc > 100) ? 100.0f : LastHealthpc);
    //... and update the text field
    HealthTF.SetString("text", roundNum(LastHealthpc)+"%");

    if (LastAmmoCount != UTWeapon(UTP.Weapon).AmmoCount)
    {
        //... Make it so...
        LastAmmoCount = UTWeapon(UTP.Weapon).AmmoCount;

        AmmoTF.SetString("text", string(LastAmmoCount));
        AmmoBarMC.GotoAndstopI((LastAmmoCount > 10) ? 10 : LastAmmoCount);
    }

    if (LastWeapon != UTWeapon(UTP.Weapon))
    {
        LastWeapon = UTWeapon(UTP.Weapon);
        switch(LastWeapon.InventoryGroup)
        {
            case 1:
                WeaponMC.GotoAndstopI(1);
                break;
            case 8:
                WeaponMC.GotoAndstopI(2);
                break;
        }
    }
}
```

Ilustración 95: código de asociación del menú flash

La función *GetVariableObject* permite obtener el elemento de la animación y las funciones *setString()* y *setFloat()* le adjudican el valor que se refresca cada décima de segundo.

## Kismet

Kismet es un sistema de “scripting” visual, basa su funcionamiento en enlaces entre cajas que pueden ser funciones, eventos, triggers, etc.

Tiene la ventaja que su programación para *UDK* ofrece por defecto muchas opciones, como desventaja, solo se asocia a los mapas y no a un tipo de juego.

# DESARROLLO DE UN VIDEOJUEGO CON UDK

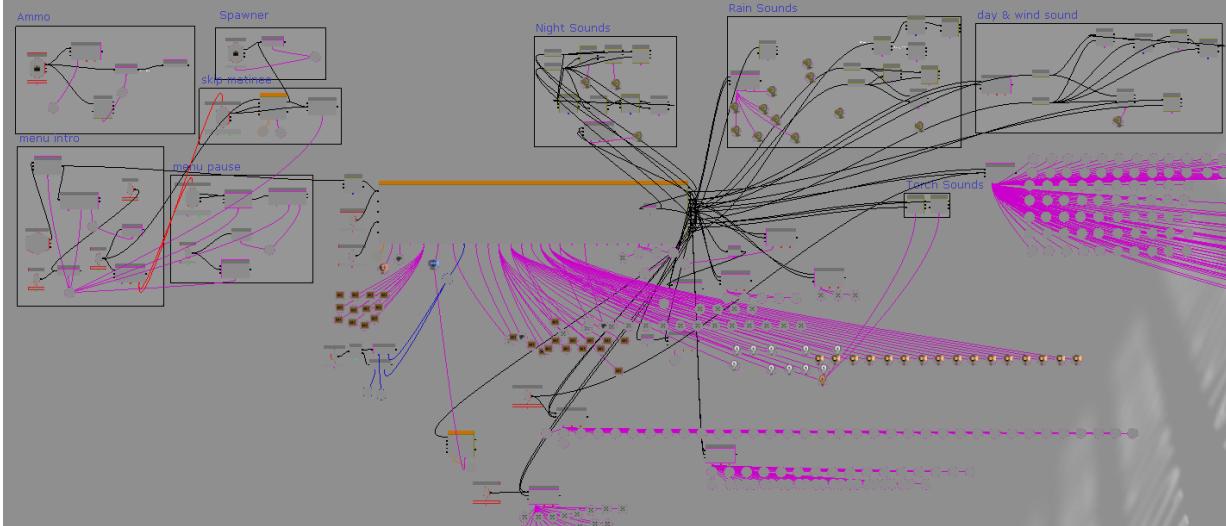


Ilustración 96: Kismet del mapa

El Kismet asociado al mapa principal del proyecto está compuesto de varios módulos, en la siguiente sección se irán describiendo cada uno de ellos.

## Menús

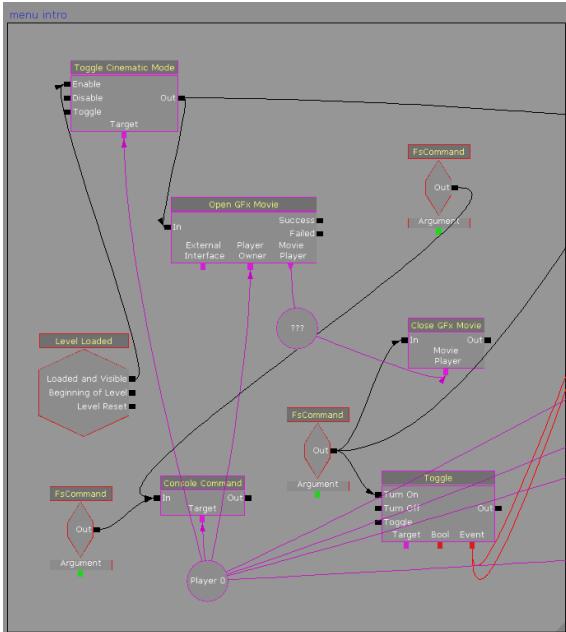


Ilustración 97: Kismet de la parte del menú flash

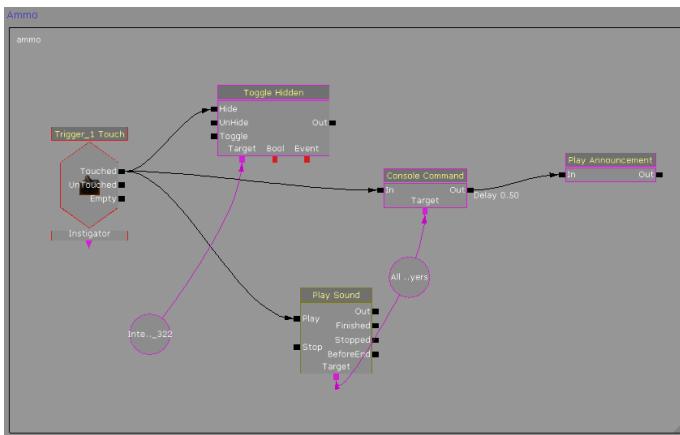
El módulo *Level loaded* permite generar o asociar eventos tras finalizar la carga del juego y empezar el nivel, se le asocia un *Toggle cinematic mode* para insertar el menú principal del juego. Los módulos *FsCommand* detectan la acción realizada en la animación y la asocian con un evento *Console Command* como salir del juego (*exit, quit*) o cargar un mapa (*open sabatini*).

El menú de pausa tiene la misma estructura que el menú intro asociando otras acciones como pasar al menú de opciones o cambiar resolución.

El menú de pausa se activa presionando la tecla 'P' del teclado y se cierra pulsando el botón *resume* o presionando la tecla 'ESC' del teclado.

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## Munición



Existe un objeto en el mapa que provee munición al personaje, para accionar este evento se crea un cilindro de colisión alrededor de este objeto y se activa si detecta movimiento dentro de él. Una vez activado, reproduce un sonido de la biblioteca y ejecuta en la consola el comando *"fullammo"* que recarga la munición del personaje

Ilustración 98: kismet de la parte de municion

## Generador de enemigos

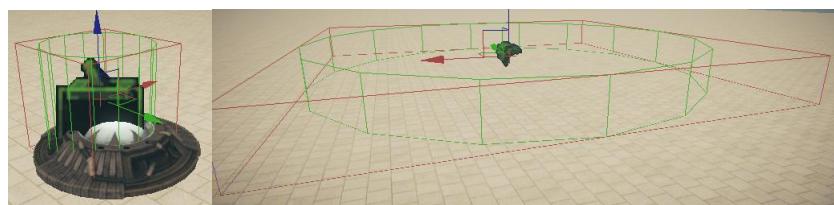
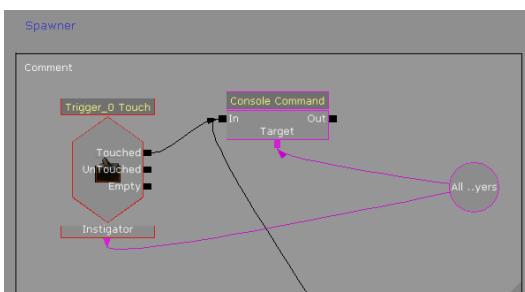


Ilustración 99: accionador spawner y spawner



El generador de enemigos es el código asociado a un cilindro que se sitúa en el mapa y permite crear un número determinado de enemigos al pasar por encima de un objeto situado en el suelo.

El objeto rectangular dentro del cilindro verde indica que es lanzador (*trigger*) que se activa con la colisión.

La cabeza de dinosaurio indica un objeto, del tipo que sea (niebla, personaje, enemigo, ítem, etcétera)

Ilustración 100: kismet spawner

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## Transición día/noche

Existen dos versiones de esta transición, la primera versión, más simple, es la que se usará para explicar el proceso, la versión más avanzada, tan solo incorpora mejoras en la luz, efectos sonoros y visuales que no afectan a la funcionalidad del Kismet.

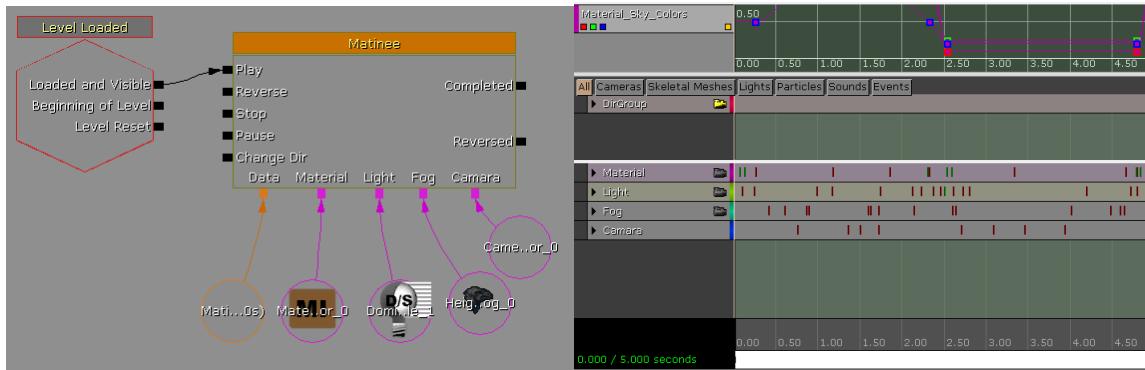


Ilustración 101: matinee y kismet de la transición día/noche

Al cargar el nivel, llama a la secuencia de matinée llamada *day&night* que establece que cada 50 segundos una fuente de luz que simula al sol, rota alrededor del mapa, y provoca un cambio de texturas e iluminación cada ciclo.

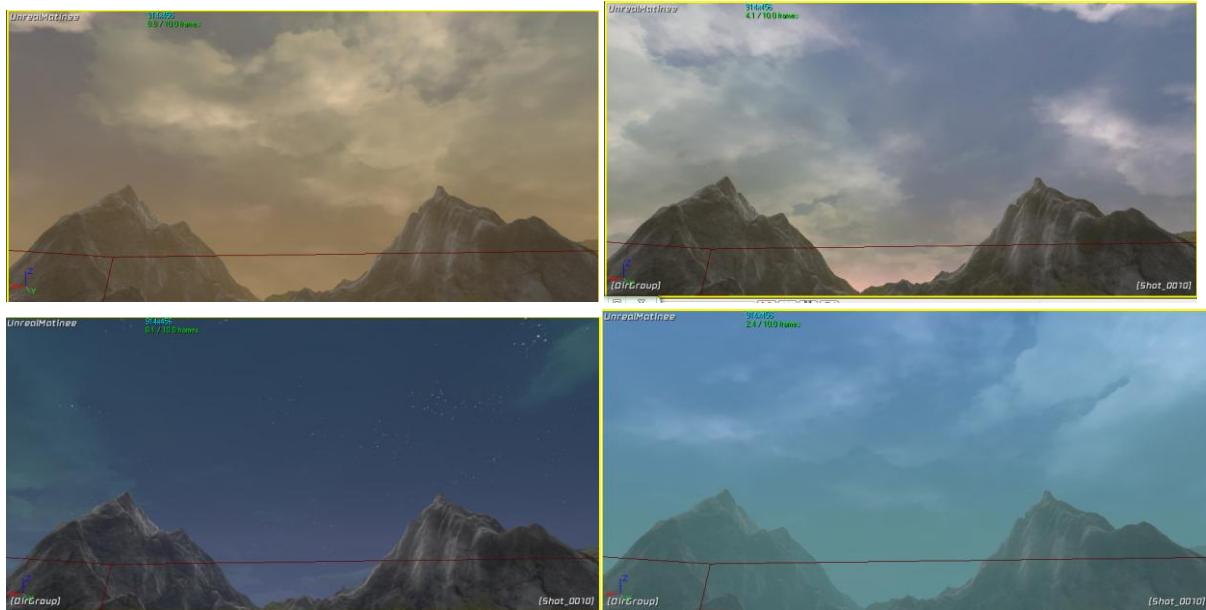


Ilustración 102: imágenes de la transición día/noche

## Efectos meteorológicos

Para la versión más avanzada, se introdujeron cambios en el clima, tales como el viento, el atardecer y la lluvia, que se incorporan todos en la misma matinée. La transición final queda del siguiente modo:

## DESARROLLO DE UN VIDEOJUEGO CON UDK

Amanecer- atardecer-viento-anochecer-lluvia



Ilustración 103: efectos meteorológicos



Para el viento se hace uso de efectos de partículas que producen un desplazamiento en el sentido deseado. Se encuentran repartidos por todo el mapa de juego y aparecen en el kismet

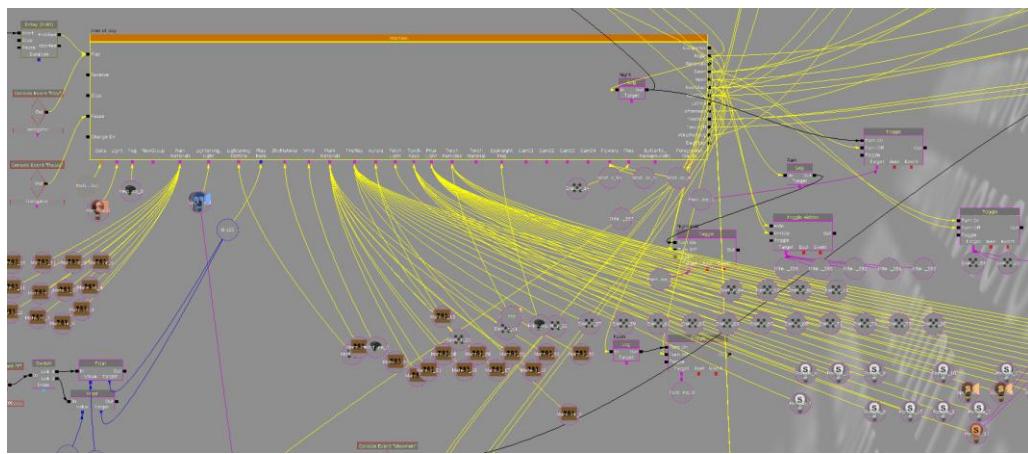
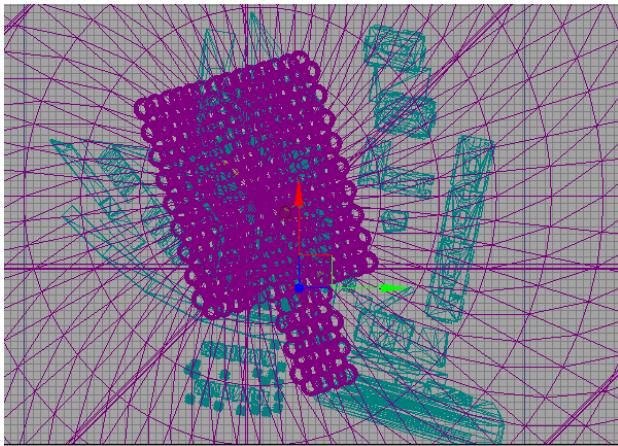


Ilustración 104: Kismet dia/noche y meteorología

## DESARROLLO DE UN VIDEOJUEGO CON UDK

Es muy significativo el cambio entre el primer Kismet y el segundo debido al número de efectos sonoros, de viento, lumínicos, efectos de partículas, animales y lluvia que se han añadido.



En concreto para la lluvia se ha diseñado un conjunto de 270 cilindros repartidos por el mapa que permiten recrear efectos de lluvia, un solo cilindro dejaba mucho espacio entre las verticales por las que se desplaza el agua, y daba un aspecto irreal, por tanto era necesario repetir el proceso a pequeña escala para producir un efecto realista.

Ilustración 105 vista en wireframe de los cilindros de lluvia

### ***Generación del ejecutable***

Para la generar el ejecutable, se usa la herramienta *Unreal FrontEnd* que permite, a partir de los elementos de UDK, seleccionar que mapas se desea exportar, y el propio programa agrupa y comprime mapas modelos y dependencias en un ejecutable.

El ejecutable final es de licencia pública, pesa 510MB y se encuentra disponible en la siguiente dirección:

<https://www.dropbox.com/s/eore09epbaoegl/UDKInstall-UC3MPB.exe>

## DESARROLLO DE UN VIDEOJUEGO CON UDK

### 4. Pruebas, resultados y evaluación

#### INTRODUCCIÓN

Una vez explicado el diseño y la implementación, es necesario verificar si el resultado funciona correctamente y cumple con los requisitos establecidos con el tutor.

Para ello, a continuación describiremos las pruebas necesarias para determinar que el sistema cumple las especificaciones requeridas.

#### EXPLICACIÓN DEL MÉTODO DE EVALUACIÓN

Para evaluar el producto, se creó un formulario con preguntas relacionadas con el cumplimiento de los requisitos y se distribuyó a 20 personas para cumplimentarlo.

| Pregunta   | Respuesta |
|--|-----------|
| ¿El ejecutable se instala correctamente?   | Si/No     |
| ¿Es necesario alguna instalación adicional?  | Si/No     |
| ¿El juego inicia correctamente?  | Si/No     |
| ¿La visualización de la introducción es correcta?  | Si/No     |
| ¿Es posible saltar el video de introducción?   | Si/No     |
| ¿El menú principal contiene las opciones de <i>play</i> , <i>options</i> y <i>exit</i> ?       | Si/No     |
| Mientras se elige opción en el menú ¿se producen cambios climatológicos en el fondo del menú?  | Si/No     |
| En el menú principal ¿se mueve la cámara mientras se elige opción?                             | Si/No     |
| Si pulsa la opción de <i>play</i> ¿inicia correctamente el juego?                              | Si/No     |
| Si pulsa la opción <i>exit</i> ¿sale de la aplicación?   | Si/No     |
| Al iniciar el juego ¿aparece una animación introductoria?                                      | Si/No     |
| Si al iniciar el juego pulsa alguna de las teclas 'W','A','S' o 'D' ¿Se desplaza el personaje? | Si/No     |

## DESARROLLO DE UN VIDEOJUEGO CON UDK

|   |       |
|---|-------|
| <i>¿Posee el juego efectos sonoros?</i>   | Si/No |
| <i>¿Reconoce el campus de Leganés de la Universidad Carlos III?</i>   | Si/No |
| <i>¿Observa algún lag o retraso durante la ejecución del juego?</i>   | Si/No |
| <i>Si se dirige a la biblioteca y obtiene el arma de paintball ¿dispara correctamente bolas de pintura?</i>                       | Si/No |
| <i>Al disparar ¿observa la mancha de pintura en el suelo?</i>   | Si/No |
| <i>Si se desplaza al exterior del edificio uno y pasa por encima del pulsador ¿observa cómo se genera una oleada de enemigos?</i> | Si/No |
| <i>Durante la partida ¿visualiza correctamente el HUD?</i>  | Si/No |
| <i>Durante la partida, al disparar u obtener munición ¿observa cómo se modifica el HUD?</i>                                       | Si/No |
| <i>Durante la partida ¿observa los cambios de día/noche?</i>  | Si/No |
| <i>Durante la partida ¿observa los cambios meteorológicos?</i>  | Si/No |
| <i>Si dispara a los soldados ¿desaparecen?</i>  | Si/No |
| <i>Si durante la partida presiona la tecla 'P' ¿visualiza el menú de opciones?</i>  | Si/No |
| <i>Si en el menú de opciones pulsa la opción resume ¿vuelve a la partida?</i>   | Si/No |
| <i>Si en el menú de opciones pulsa la opción exit ¿sale de la aplicación?</i>   | Si/No |
| <i>Si posee un mando de XBOX/PS3 conectado al ordenador, puede jugar correctamente?</i>   | Si/No |

Tabla 33: cuestionario de evaluación

## DESARROLLO DE UN VIDEOJUEGO CON UDK

### ANÁLISIS DE RESULTADOS

|    | <u>S</u> | <u>N</u> | <u>S</u> | <u>S</u> | <u>S</u> | <u>S</u> | <u>S</u> | <u>N</u> | <u>S</u> | <u>N</u> | <u>S</u> |
|----|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 1  | S        | N        | S        | <b>N</b> | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | <b>N</b> |          |          |          |          |
| 2  | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |
| 3  | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | *        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | <b>N</b> | S        | S        | S        | S        |          |          |          |          |          |
| 4  | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | <b>N</b> | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 5  | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 6  | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 7  | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 8  | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 9  | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | <b>N</b> | S        | S        | S        | S        |          |          |          |          |          |
| 10 | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 11 | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 12 | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 13 | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 14 | S        | N        | S        | <b>N</b> | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 15 | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 16 | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 17 | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 18 | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 19 | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | *        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |
| 20 | S        | N        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | N        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        | S        |          |          |          |          |          |

Tabla 34: Matriz de resultados

\*las especificaciones de estos ordenadores son inferiores a los requisitos mínimos que requiere el juego para funcionar.

Tal y como se puede apreciar en la tabla, el porcentaje de usuarios a los que la aplicación les funciona correctamente es de **75%**, aquellos usuarios que reportan 1 problema en concreto es de **15%** y solo existe un usuario que presenta dos problemas a la vez.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

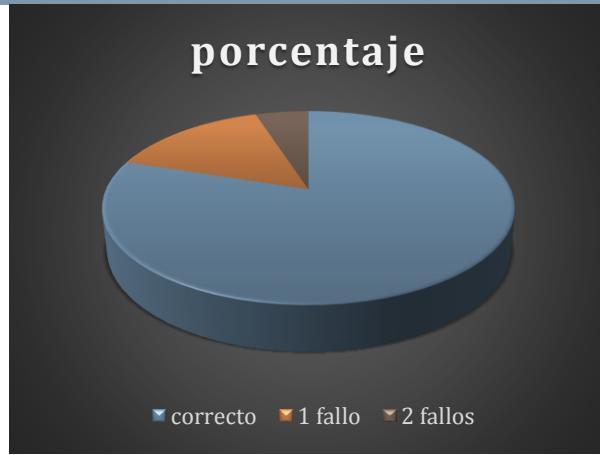


Tabla 35: porcentaje de resultados

Los problemas más comunes son:

- El menú de pausa no desaparece hasta pasado un periodo de tiempo tras pulsar el botón
- La visualización en pantalla completa a veces es errónea
- El mando presenta problemas en los menús

### MEJORA DE LA PROPUESTA RESPECTO A OTRAS POSIBLES ALTERNATIVAS

La alternativa propuesta fue el desarrollo de una aplicación usando el motor de juego incluido dentro del propio Blender

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## 5. Planificación y presupuesto

### INTRODUCCIÓN

Uno de los puntos clave del proyecto es el coste y la planificación del mismo, primero se comparará la planificación inicial con la real y a continuación se presentará el presupuesto en secciones

### PRESUPUESTO

En el siguiente apartado se detalla el presupuesto del proyecto. Gastos de personal, software, hardware y costes indirectos que el proyecto ha generado.

Las horas invertidas al proyecto han sido un total de 416 horas, que encaja con el valor total de la planificación real del diagrama de Gantt. Estas horas se descomponen de la siguiente manera:

- Análisis del problema: 7 días \* 2 horas = 14 horas
- Estudio inicial: 15 días \* 2 horas = 30 horas
- Análisis de la aplicación: 8 días \* 8.5 horas = 68 horas
- Diseño de la aplicación: 10 días \* 8 horas = 80 horas
- Implementación de la aplicación: 54 días \* 4 horas = 216 horas
- Fase de pruebas y validación: 2 días \* 3 horas = 6 horas

A continuación se puede observar un desglose del presupuesto estimado para el proyecto.

### PERSONAL

Para el cálculo del coste por hora, se ha tomado como referencia la media de la mayor y menor tarifa de dichos puestos en el portal *infojobs*

| Puesto           | Dedicación (meses/rol) | Coste/mes | Total        |
|------------------|------------------------|-----------|--------------|
| Jefe de proyecto | 0.2 (6 días)           | 3289€     | 657€         |
| Analista         | 1                      | 2160€     | 2160€        |
| Diseñador        | 1                      | 2200€     | 2200€        |
| Programador      | 1                      | 2500€     | 2500€        |
| Tester           | 0.8 (24 días)          | 1000€     | 800€         |
| <b>TOTAL</b>     | <b>4 meses</b>         |           | <b>8317€</b> |

Tabla 36: presupuesto de personal

### MATERIAL

Para la elaboración del proyecto ha sido necesario disponer de un hardware de gama alta ya que el diseño de videojuegos requiere altas prestaciones en cuanto a potencia de procesador y gráficos y licencias de software para las aplicaciones usadas.

## DESARROLLO DE UN VIDEOJUEGO CON UDK

### *Hardware*

| Descripción        | Coste | % de uso | Dedicación | Depreciación | Coste imputable |
|--------------------|-------|----------|------------|--------------|-----------------|
| <b>PC Intel i7</b> | 1748  | 75%      | 3          | 60           | 65.55           |
| <b>Wiimote</b>     | 22    | 100%     | 1          | 60           | 0.37            |
| <b>Kinect</b>      | 140   | 100%     | 1          | 60           | 2.33            |
| <b>TOTAL</b>       |       |          | <b>4</b>   |              | <b>68.25€</b>   |

Tabla 37: presupuesto de hardware

### *Software*

| Descripción            | Coste | % de uso | Dedicación | Depreciación | Coste imputable |
|------------------------|-------|----------|------------|--------------|-----------------|
| <b>Adobe Flash CS5</b> | 845   | 100%     | 3          | 12           | 211.25          |
| <b>3ds Max</b>         | 4719* | 100%     | 1          | 12           | 275.28          |
| <b>Licencia UDK</b>    | 99**  | 100%     | 1          | 12           | 8.25            |
| <b>Windows 7</b>       | 79    | 70%      | 3          | 12           | 13.825          |
| <b>TOTAL</b>           |       |          | <b>4</b>   |              | <b>508.60€</b>  |

Tabla 38: presupuesto de software

\*los programas usados han sido con licencia gratuita para estudiantes que impide su comercialización

\*\*si se supera el rango de beneficios por ventas totales en más de 50000€, hay que pagar el 25% de los beneficios a *Epic Games*

## COSTES DIRECTOS

Costes generados por el uso del ordenador o el uso de internet para acceder a recursos.

| Descripción  | Coste |
|--------------|-------|
| <b>Luz</b>   | 57    |
| <b>Adsl</b>  | 80    |
| <b>TOTAL</b> | 137€  |

Tabla 39: costes directos

## TOTALES

Para llevar a cabo el proyecto es necesario incluir el porcentaje de riesgo, el de beneficio y el IVA a las cifras ya calculadas

| Descripción     | Coste    |
|-----------------|----------|
| Personal        | 8317.80€ |
| Amortización    | 576.85€  |
| Costes directos | 137€     |

## DESARROLLO DE UN VIDEOJUEGO CON UDK

|                              |                  |
|------------------------------|------------------|
| <b>TOTAL costes directos</b> | <b>9031.65€</b>  |
| Riesgo (10%)                 | 903.17€          |
| Beneficio (10%)              | 907.17€          |
| <b>TOTAL sin IVA</b>         | <b>10837.98€</b> |
| IVA (21%)                    | 2275.98€         |
| <b>TOTAL</b>                 | <b>13113.96€</b> |

Tabla 40: costes totales

El presupuesto total del proyecto asciende a la cantidad de:

**13.113,96€**

*Trece mil ciento trece euros con noventa y seis céntimos.*

*Leganés, a 24 de Septiembre de 2013*



*Fdo. Marco Antonio Pajares Silva*

## PLANIFICACIÓN

### ESTIMADA

Al asignarse el proyecto a finales de enero, la planificación original fue pensada para entregarse en junio, más tarde, debido a problemas de trabajo, a prácticas y diversos problemas familiares, tuvo que posponerse la entrega a septiembre ya que no había tiempo para poder dedicarle al proyecto el tiempo que este necesitaba.

La planificación original tenía las siguientes tareas

| <b>Id</b> | <b>Actividad</b>                            | <b>Duración estimada de la tarea</b> | <b>Tiempo estimado dedicado</b> | <b>Inicio estimado</b> | <b>Fin estimado</b> |
|-----------|---|--------------------------------------|---------------------------------|------------------------|---------------------|
| <b>1</b>  | Viabilidad y estado del arte planteado      | 4 días                               | 4 días                          | 01/02/13               | 05/02/13            |
| <b>2</b>  | Análisis de juegos similares                | 3 días                               | 2 días                          | 06/02/13               | 09/02/13            |
| <b>3</b>  | Listado de objetivos                        | 3 días                               | 2 días                          | 09/02/13               | 11/02/13            |
| <b>4</b>  | Estudio de técnicas y herramientas graficas | 5 días                               | 5 días                          | 15/02/13               | 20/02/13            |
| <b>5</b>  | Definición de pruebas                       | 4 días                               | 3 días                          | 23/02/13               | 27/02/13            |
| <b>6</b>  | Requisitos y casos de uso                   | 6 días                               | 4 días                          | 01/03/13               | 07/03/13            |

## DESARROLLO DE UN VIDEOJUEGO CON UDK

|           |  |                 |                |                 |                 |
|-----------|--|-----------------|----------------|-----------------|-----------------|
| <b>7</b>  | Estrategia de diseño                             | 4 días          | 3 días         | 10/03/13        | 14/03/13        |
| <b>8</b>  | Diseño de personajes                             | 6 días          | 6 días         | 14/03/13        | 20/03/13        |
| <b>9</b>  | Diseño de modelos arquitectónicos                | 10 días         | 10 días        | 21/03/13        | 31/03/13        |
| <b>10</b> | Código UDK                                       | 11 días         | 10 días        | 01/04/13        | 12/04/13        |
| <b>11</b> | Importación al motor de juego                    | 10 días         | 9 días         | 14/04/13        | 24/04/13        |
| <b>12</b> | Experimentación y pruebas (1 <sup>a</sup> ronda) | 5 días          | 5 días         | 25/04/13        | 30/04/13        |
| <b>13</b> | Revisión y refactorización de código             | 8 días          | 6 días         | 03/05/13        | 11/05/13        |
| <b>14</b> | Experimentación y pruebas (2 <sup>a</sup> ronda) | 6 días          | 3 días         | 12/05/13        | 18/05/13        |
| <b>15</b> | Análisis de resultados                           | 4 días          | 2 días         | 20/05/13        | 24/05/13        |
| <b>16</b> | Revisión   | 9 días          | 8 días         | 01/06/13        | 10/06/13        |
| <b>17</b> | Documentación                                    | 10 días         | 9 días         | 10/06/13        | 20/06/13        |
| <b>T</b>  | <b>TOTAL</b>                                     | <b>103 días</b> | <b>93 días</b> | <b>01/02/13</b> | <b>20/06/13</b> |

Tabla 41: tareas planificación inicial

\*Estimando una media de cuatro horas por día.

### TRABAJO FIN DE GRADO

## DESARROLLO DE UN VIDEOJUEGO CON UDK

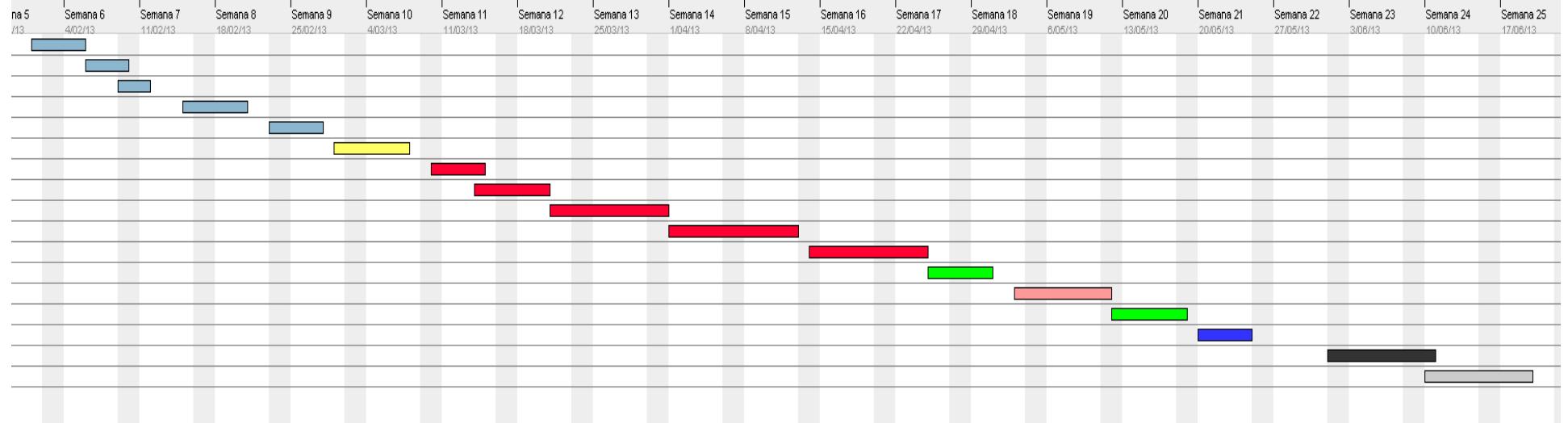


Tabla 42: Planificación estimada

- **Gris:** Planteamiento, estudio y análisis
- **Amarillo:** requisitos
- **Rojo:** diseño e implementación
- **Verde:** Experimentación
- **Rosa:** refactorización
- **Azul:** pruebas
- **Negro:** revisión
- **Gris:** documentación

## DESARROLLO DE UN VIDEOJUEGO CON UDK

### REAL

Al asignarse el proyecto a finales de enero, la planificación original fue pensada para entregarse en junio, más tarde, debido a problemas de trabajo, a prácticas y diversos problemas familiares, tuvo que posponerse la entrega a septiembre ya que no había tiempo para poder dedicarle al proyecto el tiempo que este necesitaba.

La planificación original tenía las siguientes tareas

| <b>Id</b> | <b>Actividad</b>                                 | <b>Duración real de la tarea</b> | <b>Tiempo real dedicado</b> | <b>Inicio real</b> | <b>Fin real</b> |
|-----------|--|----------------------------------|-----------------------------|--------------------|-----------------|
| <b>1</b>  | Viabilidad y estado del arte planteado           | 4 días                           | 4 días                      | 03/06/13           | 07/06/13        |
| <b>2</b>  | Análisis de juegos similares                     | 3 días                           | 2 días                      | 08/06/13           | 11/06/13        |
| <b>3</b>  | Listado de objetivos                             | 3 días                           | 2 días                      | 12/06/13           | 15/06/13        |
| <b>4</b>  | Estudio de técnicas y herramientas graficas      | 2 días                           | 2 días                      | 15/06/13           | 17/06/13        |
| <b>5</b>  | Definición de pruebas                            | 1 día                            | 1 día                       | 18/06/13           | 19/06/13        |
| <b>6</b>  | Requisitos y casos de uso                        | 4 días                           | 3 días                      | 20/06/13           | 24/06/13        |
| <b>7</b>  | Estrategia de diseño                             | 4 días                           | 4 días                      | 25/06/13           | 29/06/13        |
| <b>8</b>  | Diseño de personajes                             | 10 días                          | 6 días                      | 01/07/13           | 10/07/13        |
| <b>9</b>  | Diseño de modelos arquitectónicos                | 24 días                          | 22 días                     | 11/07/13           | 3/08/13         |
| <b>10</b> | Código UDK                                       | 21 días                          | 18 días                     | 04/08/13           | 25/08/13        |
| <b>11</b> | Importación al motor de juego                    | 4 días                           | 4 días                      | 26/08/13           | 30/08/13        |
| <b>12</b> | Experimentación y pruebas (1 <sup>a</sup> ronda) | 1 día                            | 1 día                       | 31/08/13           | 31/08/13        |
| <b>13</b> | Revisión y refactorización de código             | 2 días                           | 1 días                      | 31/08/13           | 01/09/13        |
| <b>14</b> | Experimentación y pruebas (2 <sup>a</sup> ronda) | 1 día                            | 0.5 día                     | 01/09/13           | 02/09/13        |
| <b>15</b> | Análisis de resultados                           | 2 días                           | 1 días                      | 02/09/13           | 02/09/13        |
| <b>16</b> | Revisión   | 2 días                           | 1 días                      | 03/09/13           | 05/09/13        |
| <b>17</b> | Documentación                                    | 19 días                          | 19 días                     | 05/09/13           | 24/09/13        |
| <b>T</b>  | <b>TOTAL</b>                                     | <b>107 días</b>                  | <b>90.5 días</b>            | <b>03/06/13</b>    | <b>24/09/13</b> |

Tabla 43: tareas planificación inicial

\*Estimando una media de cuatro horas por día.

TRABAJO FIN DE GRADO

# DESARROLLO DE UN VIDEOJUEGO CON UDK



Tabla 44: planificación real

- **Gris:** Planteamiento, estudio y análisis
- **Amarillo:** requisitos
- **Rojo:** diseño e implementación
- **Verde:** Experimentación
- **Rosa:** refactorización
- **Azul:** pruebas
- **Negro:** revisión
- **Gris:** documentación

# DESARROLLO DE UN VIDEOJUEGO CON UDK

## 6. Conclusiones

INTRODUCCIÓN

OBJETIVOS CUMPLIDOS

PROBLEMAS ENCONTRADOS

LÍNEAS FUTURAS DE TRABAJO

CONCLUSIONES

## DESARROLLO DE UN VIDEOJUEGO CON UDK

### Bibliografía

- aDeSe, A. E. (s.f.). <http://www.adese.es/docs/documentacion/convenios>. Obtenido de <http://www.adese.es/docs/documentacion/convenios>
- Adobe, S. (s.f.). Obtenido de [http://help.adobe.com/es\\_ES/flash/cs/using/WSE38D6111-9693-43ba-B25B-37FD63958F7Ba.html](http://help.adobe.com/es_ES/flash/cs/using/WSE38D6111-9693-43ba-B25B-37FD63958F7Ba.html)
- API, E. G. (s.f.). Obtenido de <http://udn.epicgames.com/Three/UE3Basics.html>
- BIK, B. (s.f.). Obtenido de <http://www.radgametools.com/bnkmain.htm>
- Character, U. (s.f.). Obtenido de <http://udn.epicgames.com/Three/UT3CustomCharacters.html>
- Diez, M. (s.f.). *El Blog de Marcos*. Obtenido de <http://marcosdiez.wordpress.com/pfm/>
- FBX, A. (s.f.). Obtenido de <http://latinoamerica.autodesk.com/adsk/servlet/index?siteID=7411870&id=11278653>
- Fernandez, M. D. (Julio de 2013). Obtenido de <https://dl.dropboxusercontent.com/u/13952762/PFM%20-%20RONIN/Proyecto%20Fin%20de%20M%C3%A1ster%20-%20Dise%C3%B1o%20y%20desarrollo%20de%20un%20videojuego%20en%20UDK%20-%20Marcos%20D%C3%A9z%20Fern%C3%A1ndez.pdf>
- Mixamo. (s.f.). Obtenido de <http://www.mixamo.com>
- PNG. (s.f.). Obtenido de <http://www.libpng.org/pub/png/>
- Scaleform, E. G. (s.f.). Obtenido de <http://udn.epicgames.com/Three/Scaleform.html>
- SMPTE. (s.f.). Obtenido de <https://www.smpte.org/standards>
- UPK. (s.f.). Obtenido de <http://udn.epicgames.com/Three/UnrealPackages.html#Package Types>
- WAV. (s.f.). Obtenido de <http://tools.ietf.org/html/rfc2361>
- Wikipedia. (s.f.). *Wikipedia*. Obtenido de [http://es.wikipedia.org/wiki/Industria\\_de\\_los\\_videojuegos#Principales\\_consolas\\_en\\_el\\_mercado](http://es.wikipedia.org/wiki/Industria_de_los_videojuegos#Principales_consolas_en_el_mercado)