Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Animaciones programáticas con Manim como soporte a la docencia universitaria

ESCUELA POLITECNICA SUPERIOR

Autor: Álvaro Mesón Martín

Tutor: Óscar García Población

2023

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

Animaciones programáticas con Manim como soporte a la docencia universitaria

Autor: Álvaro Mesón Martín

Tutor/es: Óscar García Población

TRIBUNAL:

Presidente: María del Mar Lendínez Chica

Vocal 1º: Elena Campo Montalvo

Vocal 2º: Óscar García Población

Agradecimientos

Con este trabajo pongo el final a 4 años de trabajo, esfuerzo, constancia y, sobre todo, sacrificio.

No puedo comenzar mis agradecimientos sin mencionar a mis padres y a mi hermano. Gracias a su apoyo total y confianza absoluta, no solo en esta etapa universitaria sino durante todo mi proceso de aprendizaje. Gracias por ser ese pilar en el que ampararse cuando las condiciones no están de cara y los momentos se complican. Gracias.

Tampoco podían faltar todos mis compañeros de la carrera con los que he compartido este camino y me llevo grandes lecciones de ellos. Tengo que destacar entre ellos a Eneko y Mario ya que han sido siempre unos maravillosos compañeros y ante eso amigos.

Para concluir, tengo que nombrar al "Club Náutico" con los que he compartido mucho tiempo y me han ayudado a conseguir esa desconexión tan importante para todas las personas. De aquí me gustaría distinguir a Ana que, aparte de ser amiga en el pleno sentido de la palabra, es la persona que siempre está, no importa cuál sea la situación.

Resumen

En este Trabajo de Fin de Grado voy a desarrollar una serie de animaciones programáticas utilizando la herramienta `Manim`. Gracias al avance de la tecnología se pueden crear ficheros multimedia sin necesidad de usar los métodos convencionales para hacer animaciones. Para mostrar el potencial de `Manim`, frente a la edición de vídeo usual, haré tres animaciones haciendo uso de técnicas diferentes. Para la primera de ellas haré uso de primitivas previamente creadas para crear el vídeo final. La segunda se basa en una simulación real y gracias a la emisión y suscripción de eventos podré obtener la animación. Por último, usaré un guion que describe un proceso, para ir haciendo pequeñas animaciones que compondrán el vídeo final.

Palabras clave: animaciones programáticas, Latex, escenas, primitivas, simulación.

Abstract

In this Final Degree Project, a series of programmatic animations will be developed using the `Manim` tool. Thanks to the advancement of technology, multimedia files can be created without the need to use conventional methods to make animations. To show the potential of `Manim`, compared to the usual video editing, we will make three animations using different techniques. For the first one I will use previously created primitives to create the final video. The second one, is based on a real simulation and thanks to the broadcast and subscription of events I will be able to obtain the animation. Finally, I will use a script that describes a process, to make small animations that will make up the final video.

Keywords: programmatic animations, Latex, scenes, primitives, simulation.

Resumen extendido

La complejidad de producir animaciones es muy elevada debido a las cotas tan altas que hay hoy en día. El coste que tienen, tanto económico como personal, es muy cuantioso. El primero de ellos es evidente por el desarrollo brutal de la tecnología en este campo. Cada vez hay mejores tarjetas gráficas, mejores equipos, pero el precio también va a más. Por otro lado, el coste personal implica el gasto que supone una persona especializada en esta disciplina, pero, además, se tiene en cuenta el proceso que se necesita para dominar esta tarea, que ciertamente no es reducido.

En favor de esto, surgen las animaciones programáticas que proponen otro paradigma a la hora de llevar a cabo la producción de estas. El principal exponente de esto es la herramienta `Manim`. `Manim` es una herramienta de animación creada por Grant Sanderson, matemático de Stanford y dueño del canal de YouTube 3Blue1Brown.

Esta herramienta está especializada para crear animaciones de carácter matemático. `Manim` está compuesta por cinco partes: la primera y más importante es Python, que podríamos decir que es como su corazón. La segunda parte es `LaTeX` [1], el programa con el que se escriben los textos y fórmulas. La tercera parte es Cairo [2], el programa que hace las figuras. La cuarta es FFmpeg, la que une cada uno de los cuadros para formar el vídeo y, por último, SoX[3], un programa ligero de audio. Aunque `Manim` está compuesta por todas estas partes, solamente requerimos escribir los comandos en Python para crear las animaciones.

Las ventajas que presenta son las siguientes: en primer lugar, es software libre, esto quiere decir que se puede compartir y editar de manera gratuita y legal, además de que se puede personalizar a tu gusto. También funciona en computadoras de bajos recursos. Funciona perfectamente en Windows, Mac y Linux, además de que se pueden compartir los proyectos entre ellos. Los vídeos que exporta son de alta calidad y de bajo peso. Por último, cuenta con una amplia documentación sostenida por toda la comunidad.

Contenido

RESUMEN **7**

ABSTRACT **9**

RESUMEN EXTENDIDO **11**

1.INTRODUCCIÓN **19**

1.1 OBJETIVOS DE ESTE PROYECTO 20
1.2 ORGANIZACIÓN DEL TRABAJO 21

2.BIBLIOTECA MANIM **23**

2.1 CÓMO SE USA 23
2.2 CÓMO SE DESARROLLA CON LA HERRAMIENTA: ENTORNO DE DESARROLLO 27
2.3 GENERACIÓN DE LOS VIDEOS 28
2.4 SISTEMA DE OBJETOS 29

3.DISEÑO DE LAS ANIMACIONES **31**

3.1 PLANIFICACIÓN DE CPU 32
3.2 COLAS MM1 37
3.3 PAGINACIÓN 43
3.3.1 Escena 1: Estructura de la CPU 44
3.3.2 Escena 2: Organización de la memoria 45
3.3.3 Escena 3: Producción de direcciones físicas 46
3.3.4 Escena 4: Ejemplo de programa 47
3.3.5 Escena 5: Proceso de traducción de una dirección 48
3.3.6 Escena 6: Producción de direcciones virtuales 49
3.3.7 Escena 7: Producción de direcciones virtuales II 49
3.3.8 Escena 8: Organización de la tabla de páginas 50
3.3.9 Escena 9: Organización de la tabla de páginas II 51
3.3.10 Escena 10: Búsqueda del marco 52
3.3.11 Escena 11: Proceso de traducción de una dirección II 53
3.3.12 Escena 12: Presentación de la fórmula de la tabla de páginas 54

4.RESULTADOS PUBLICADOS **59**

5.CONCLUSIONES Y TRABAJOS FUTUROS **61**

6.BIBLIOGRAFÍA **63**

Lista de Figuras

FIGURA 1. INFORMACIÓN DE LOS REPOSITORIOS 24
FIGURA 2. OVERVIEW DE 3B1B Y MANIMCOMMUNITY 24
FIGURA 3. COLABORADORES DE LAS VERSIONES 25
FIGURA 4. ESCENARIO PRINCIPAL DE LA PLANIFICACIÓN DE UNA CPU 32
FIGURA 5. MODELO DE TEORÍA DE COLAS 37
FIGURA 6. ESCENARIO DE UNA SIMULACIÓN DE COLAS M/M/5 38
FIGURA 7. ESQUEMA DE LA PAGINACIÓN 43
FIGURA 8. ELEMENTOS DE LA CPU 44
FIGURA 9. ESTRUCTURA DE LA MEMORIA 45
FIGURA 10. GENERACIÓN DE DIRECCIONES 46
FIGURA 11. ESTRUCTURA DE UN PROGRAMA ALMACENADO EN DISCO DURO 47
FIGURA 12. PROCESO DE TRADUCCIÓN DE DIRECCIONES 48
FIGURA 13. REPRESENTACIÓN DE UNA PÁGINA 49
FIGURA 14, ESTRUCTURA DE LA TABLA DE PÁGINAS 50
FIGURA 15. CAMPOS DE LA TABLA PÁGINAS 51
FIGURA 16. TRADUCCIÓN DE LA PÁGINA AL MARCO 52
FIGURA 17. PROCESO DE TRADUCCIÓN DESDE CPU HASTA MEMORIA 53
FIGURA 18. FÓRMULA PARA EL TAMAÑO DE LA TABLA DE PÁGINAS 54
FIGURA 19. INTERFAZ PRINCIPAL DE DAVINCI RESOLVE 56
FIGURA 20. ORGANIZACIÓN DE CARPETAS EN DAVINCI RESOLVE 57

Lista de Tablas

TABLA 1. DESFASES DE LAS ESCENAS 55

1.Introducción

En este Trabajo de Fin de Grado voy a desarrollar distintas animaciones programáticas utilizando `Manim`, con el propósito de dar apoyo a la docencia universitaria.

A diferencia de las animaciones ordinarias, realizadas mediante edición de video, las animaciones programáticas se obtienen por medio de código, esto las permite basarse en simulaciones reales.

La motivación de este proyecto viene por el papel que tienen los documentos gráficos, en especial las animaciones, en el proceso de enseñanza del estudiante.

Según un estudio de *Educational Review Psychology* [4], se demuestra la efectividad de las animaciones en estudiantes, ya sea a la hora de almacenar información, de memorizar o de recuperar dicha información. En dicho estudio se examina el papel de la animación en el aprendizaje multimedia, se presenta una teoría cognitiva del aprendizaje multimedia y se resume el programa de investigación, que ha producido siete principios para el uso de la animación en la instrucción multimedia.

Estos incluyen el principio multimedia, el principio de contigüidad espacial, el principio de contigüidad temporal, el principio de coherencia, el principio de modalidad, el principio de redundancia, y el principio de personalización. Por último, el estudio dice que la animación puede promover la comprensión del alumno cuando se usa de manera consistente con la teoría cognitiva del aprendizaje multimedia.

La conclusión es evidente, el aprendizaje se mejora con las animaciones y más si estas van acompañadas de un discurso informal. Apoyándonos en los principios mencionados en el estudio, se optimiza el aprendizaje cuando se presentan palabras en estilo conversacional en lugar de formal. Por otro lado, tiene un gran beneficio presentar los textos junto con la animación en vez de la narración sola.

Uno de los casos más destacados es el de `3Blue1Brown` [5], nombre del canal de YouTube sobre videos explicativos de matemáticas. Grant Sanderson es la persona que está detrás de todo. Grant comenzó el repositorio como un proyecto personal, teniendo como propósito animar sus videos. Finalmente, se ha convertido en un potente motor para animaciones programáticas.

La clave de este trabajo reside en las propias animaciones programáticas como se ha comentado con anterioridad. Estas se basan en simulaciones reales, por lo que es un aspecto muy interesante para este campo, la informática. Esto se debe a que realizar una simulación sobre algún proceso computacional o procedimiento informático es razonablemente sencillo. `Manim`, a través de sus herramientas, facilita la transición desde una simple simulación a una animación potente desde el punto de vista visual.

1.1 Objetivos de este proyecto

Como se ha dicho con anterioridad, el objetivo de este trabajo es producir animaciones orientadas a la docencia. Como se verá durante este trabajo, las posibilidades que nos brinda esta herramienta para generar animaciones en video son bastante diversas.

En este proyecto se van a desarrollar tres técnicas diferentes del uso de la herramienta con el fin último de obtener esas animaciones que ayuden al docente.

La primera de las técnicas consistirá en el uso de primitivas. Con esta metodología se facilita al docente la creación de vídeos ya que simplemente tendrá que hacer uso, a su antojo, de las primitivas previamente creadas. Una de las ventajas que tiene esta técnica es que solamente hay que hacer las primitivas que, al ser porciones de código pequeñas, tampoco tienen una dificultad excesiva.

Las primitivas que se usan con esta técnica, pertenecen al dominio de la aplicación, es decir, que aplican terminología propia del ámbito de la especialidad. En mi caso, al realizar una animación que gira en torno a la planificación de uso de una CPU, se encontrarán primitivas como *sleep*, *dispatch*, *wake up* ya que se está dentro del ámbito de los sistemas operativos.

Con relación a la segunda, utilizaré la herramienta en consonancia con una simulación real. Esta técnica permite hacer uso de la herramienta junto con simulaciones reales de procesos. La manera en la que crearán un vínculo la herramienta y la simulación es gracias a la emisión de eventos y la posterior suscripción a estos.

Para la tercera y última de las técnicas, me acercaré mucho a la producción de animaciones de manera tradicional, ya que a partir de un guion crearemos un video. Esta técnica tiene utilidad cuando se quiere representar el comportamiento o la naturaleza de un proceso en concreto.

1.2 Organización del trabajo

Para realizar este trabajo he seguido una estructura, la cual muestro a continuación:

- Introducción: en este apartado presentaré el caso de estudio al mismo tiempo que el contexto del tema, además de determinar los objetivos del proyecto.

- Biblioteca Manim: en esta sección haré una visión general de la biblioteca que utilizaré, desde cómo es su uso hasta la composición de sus herramientas.

- Diseño de las animaciones: en este capítulo explicaré con detalle el proceso de desarrollo seguido para cada una de las animaciones.

- Resultados publicados: en este apartado se mostrarán los códigos QR que enlazan con los videos de los resultados publicados en la plataforma YouTube.

- Conclusiones y trabajos futuros: en este punto comentaré los resultados y las conclusiones a las que he llegado tras realizar este trabajo. También, mencionaré posibles vías que puedan dar continuidad al proyecto.

En el cuerpo de este trabajo se observará todo el desarrollo que se ha ido realizando, explicando con detalle cada proceso mediante imágenes y porciones de código ilustrativas.

El resultado de todo el trabajo efectuado se podrá observar en la sección de ‘Resultados publicados’, donde se encuentran los frutos de dicho desarrollo. Estos se encuentran almacenados en un canal de YouTube para su libre visualización.

2.Biblioteca `Manim`

En este capítulo explicaré la herramienta que usaré para realizar las animaciones. En primer lugar, definiré el proceso que hay que seguir para poder hacer uso de `Manim`, desde las distintas versiones que se ofrecen hasta la adquisición de todos requerimientos necesarios para estar en disposición de trabajar con la herramienta.

Seguidamente pasaré a hablar del entorno de desarrollo, necesario para realizar las tareas de implementación, junto con su organización.

Finalmente, expondré como se generan los videos con `Manim` y cuáles son las distintas configuraciones ya que, al fin y al cabo, los videos son el resultado final que se busca. Además, concluiré desarrollando el sistema de objetos que tiene la herramienta.

2.1 Cómo se usa

`Manim` es un motor de animaciones, creado inicialmente para videos de matemáticas, aunque el abanico de campos en los que se puede aplicar es muy amplio.

`Manim` cuenta con dos versiones:

- 3b1b/manim [6]: esta versión está mantenida por Grant Sanderson, creador del canal de YouTube `3Blue1Brown`. Tiene una mayor eficiencia, una velocidad de renderización más rápida y da soporte a una interacción en tiempo real.

- ManimCommunity/manim [7]: esta otra versión se mantiene gracias al Manim Community Dev Team. Para esta versión, existe una mayor cantidad de documentación gracias a la amplia contribución de la comunidad.

Para dar un poco de perspectiva a estas dos versiones, en las páginas de cada uno de los repositorios, existe una pequeña información general, pero que nos da una primera noción de la magnitud de las versiones. Esta, muestra las estrellas recibidas, los usuarios siguiendo el repositorio y la cantidad de copias del repositorio en la cuenta del usuario (forks). En la imagen siguiente se puede observar la versión de Grant a la izquierda y la de la comunidad a la derecha.

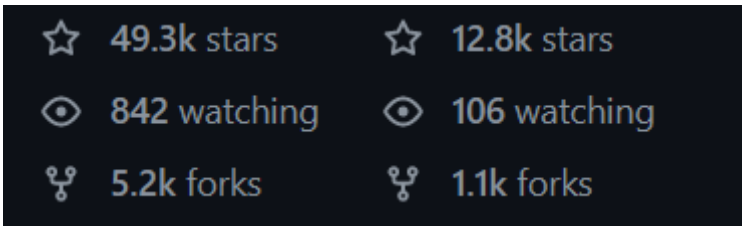

Figura 1. Información de los repositorios

Para profundizar más en la información se tiene la vista de *insights* la cual arroja información acerca del proyecto. En primera instancia, nos muestran una visión general con las solicitudes para realizar *pull* y los problemas activos en la última semana. En la Figura 2 se ve, arriba la versión de 3b1b y abajo la otra versión.

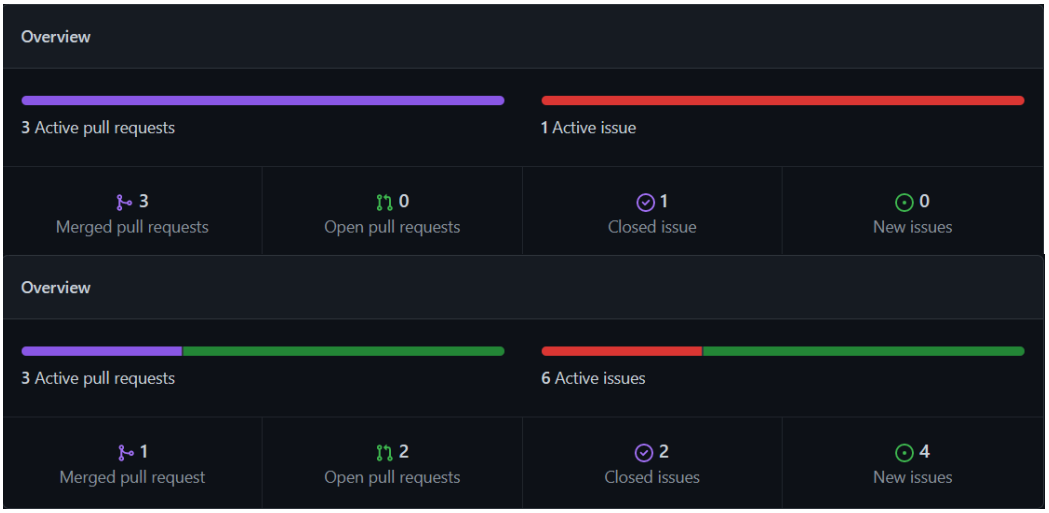

Figura 2. Overview de 3b1b y ManimCommunity

Si me dirijo a la pestaña de *contributors* puedo comparar el volumen de colaboradores entre la versión de Grant y la de la comunidad. Esta comparativa abarca desde marzo de 2015 hasta enero de 2023, en el día que se realiza este proyecto. Las gráficas muestran los colaboradores excluyendo las cuentas de *bots* y los *merge commits*. La que se encuentra en la parte superior corresponde con la versión 3b1b, mientras que la que se dispone en la parte inferior corresponde a la versión de la comunidad.

Figura 3. Colaboradores de las versiones

Tras esta comparación entre versiones, en mi caso, voy a hacer uso de la segunda versión, ya que su gran documentación me va a ayudar al desarrollo de las animaciones.

Cabe destacar que existe una versión interactiva online, en el caso de que se quiera probar la herramienta antes de instalarla, Manim online [8]. Este es un entorno de prueba temporal en el que se puede desarrollar con `Manim` sin necesidad de instalarlo localmente.

Sin embargo, hay que tener en cuenta que es un entorno temporal, lo que conlleva que los cambios no se guardarán y no se podrán compartir con otras personas. Para guardar el trabajo que se realice, se deberá descargar el archivo del cuaderno ("Archivo → Descargar como → Cuaderno (.ipynb)"). Este entorno trabaja con Jupiter Notebooks y basta con escribir el código de nuestra animación en los bloques y ejecutarlo.

Una vez elegida la versión [9], se pasa a ver su instalación. `Manim` es una librería de Python, por lo que puede ser instalada vía `pip` (sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python). Sin embargo, para un correcto funcionamiento de `Manim`, se necesitan instalar algunas dependencias primero. Estas dependencias pueden ser requeridas u opcionales. Uno de los requisitos obligatorios para la utilización de `Manim`, en el momento que se escribe esta memoria, es que requiere de la versión de Python 3.7 o superior.

Dependiendo del sistema operativo que se posea, se deberán seguir unas instrucciones u otras. En el caso que nos ocupa, llevaré a cabo una instalación en Windows.

En primer lugar, voy a instalar las dependencias requeridas. Como he mencionado antes, lo primero es tener Python en una versión razonablemente reciente (3.7 -3.10). Para ello, me dirijo a Python [10], a la vista de downloads, y elijo el installer. Sigo las instrucciones y de esta forma tendríamos Python instalado en nuestro equipo.

La siguiente dependencia requerida será FFmpeg. FFmpeg es una colección de software libre que puede grabar, convertir y hacer streaming de audio y vídeo. Manim necesita de este programa al igual que de Python para poder funcionar, este último por su evidente implicación en la generación de las animaciones.

Para obtenerla me dirijo a FFmpeg [11] donde habrá una versión precompilada. Se descomprime la carpeta que se descarga y se modifica la variable de entorno `Path`. Para conseguirlo, voy a la siguiente ruta: Editar las variables de entorno del sistema > Variables de entorno, y edito la variable `Path` añadiendo la ruta completa del directorio `Bin`. Para terminar, guardo los cambios.

```
C:\Program Files\ffmpeg-4.4.1-essentials_build\bin
```

Para comprobar que tengo FFmpeg correctamente instalado y en funcionamiento, se puede abrir un command line prompt (símbolo del sistema). Ejecuto el comando `ffmpeg` y debería de reconocerse.

Tras esto, tendré todas las dependencias instaladas y se podrá obtener `Manim` a través del siguiente comando:

```
python -m pip install manim
```

Como se ha mencionado antes, hay dependencias cuya instalación es opcional, pero que aportan funcionalidades extra. En mi caso, voy a hacer uso de la interfaz de `Manim` para `LaTeX`, por lo que `LaTeX` tiene que estar instalado también.

Esta interfaz necesita por lo tanto una distribución de LaTeX para Windows, esta es MiKTeX [12] y para obtenerla, realizo una instalación mediante el instalador de la página. Esta distribución es necesaria para llevar a cabo este proyecto, ya que es la encargada de renderizar desde ecuaciones hasta texto estilizado con `TeX`.

En este punto, ya estoy preparado para empezar a desarrollar nuestras animaciones en Manim.

2.2 Cómo se desarrolla con la herramienta: entorno de desarrollo

Para comenzar a realizar las animaciones se necesita un entorno de desarrollo. En mi caso, he elegido Visual Studio Code debido a su gran potencia como editor y a sus extensiones, ya que aportan funcionalidades adicionales de forma aislada y modular. Se puede conseguir por medio de su página Visual Studio Code [13].

Visual tiene extensiones para diversos *frameworks*, pero en mi caso necesitaré las relacionadas con Python. La primera y esencial sería 'Python', debido a que nos proporciona soporte para este lenguaje. Otra extensión interesante es 'Python Indent', la cual sirve para corregir la sangría de Python en Visual Studio Code. Por último, 'Python Snippets' sirve para hacer más productivo el trabajo con Python porque proporciona opciones de finalización junto con todos los argumentos.

Con el entorno preparado, se establecerá nuestra organización de directorios para una metodología de trabajo más ordenada y eficiente.

La jerarquía será un directorio raíz en el cual se crearán 3 carpetas, una por cada animación. En cada una de estas carpetas se encontrarán tanto los archivos Python con el código fuente como una carpeta llamada *Media* que es generada por `Manim` al momento de renderizar las animaciones.

La carpeta *Media* contiene en su interior el contenido audiovisual, que a su vez se organiza en tres carpetas distintas que contienen las imágenes, los textos y los videos. Estas carpetas están nombradas como *Images*, *Tex* y *Videos* respectivamente.

La primera carpeta, *Images*, está pensada para depositar las imágenes que se utilizarán en las animaciones. En la carpeta *Tex* se guardarán los ficheros que contienen la información de las características de los textos utilizados en las animaciones. Y, por último, la carpeta *Videos* es donde se guardarán por defecto los videos que renderizaré con la herramienta.

En cuanto al desarrollo propiamente dicho, se realizará en el lenguaje Python como he comentado. Para poder hacer uso de las funcionalidades que da `Manim` hay que importarlo mediante el siguiente comando al inicio del fichero de código:

```
from manim import *
```

De esta forma, tengo a mi disposición toda la biblioteca de Manim.

Aunque hablaré del diseño de las animaciones en profundidad más adelante, vamos a ver una visión global del desarrollo.

Un primer nivel de desagregación serían las clases que se pueden distinguir en dos tipos: clases normales y clases de tipo escena.

El primer tipo, son clases normales que proporciona el lenguaje y las uso para definir la estructura de los objetos que confeccionan la animación, mediante sus atributos. Su sintaxis es la siguiente:

```
class Nombre_de_la_clase:
      …
```

Las clases proveen una forma de empaquetar los datos, atributos y funcionalidades juntos. De esta forma, al crear una clase, se crea un nuevo 'tipo de objeto' pudiendo crear nuevas instancias de este tipo.

Por otro lado, las clases de tipo escena son nuestro lienzo donde plasmar las animaciones. Su sintaxis es la siguiente:

```
class Nombre_de_la_clase(Scene):
      …
```

En un segundo nivel, tendríamos los métodos y funciones. Con ellos se puede hacer que los diferentes objetos ejecuten bloques de código en concreto.

Utilizando un símil para comprender la organización del código, se puede establecer: Una persona (clase), tiene sus cualidades (atributos) y puede realizar diferentes acciones con su entorno (métodos y funciones).

2.3 Generación de los videos

Para generar los videos, que es el objetivo final, se deben tener en cuenta varios aspectos: si se tienen varias escenas en un mismo fichero, la resolución del video...

Para renderizar el video se usa el siguiente comando, el cual se modifica según las necesidades:

```
manim -[flags] fichero.py escena
```

Voy a explicar el comando a fondo. Primeramente, encontramos la sentencia 'manim' que se ejecutará sobre archivo 'fichero.py' que es quien contiene el código de la animación. Además, el comando especifica qué escena tiene que ser renderizada, en este caso 'escena', aunque este parámetro es opcional.

En cuanto a las flags, hay varias:

- `-p`: indica a manim que tiene que reproducir el video una vez que la ha renderizado
- `-q`: la calidad con la que renderizará, `-ql` se usará para calidad baja (low quality), mientras que `-qh` será para calidades altas (high quality).

Estas son las más importantes y usadas, sin embargo, hay muchas más flags que se pueden incluir en el comando, como, por ejemplo: `-s`, `-a`. La primera de ellas arrojará el último frame de la escena, la flag `-a`, servirá para renderizar todas las escenas contenidas en el fichero de input [14].

2.4 Sistema de objetos

Un aspecto a destacar es el sistema de objetos que tiene `Manim`. `Manim` tiene la estructura básica Mobject de la que extienden todas las demás colecciones. De esta colección raíz cuelgan las subcolecciones básicas, como pueden ser los marcos, geometría, gráficos, matrices, etc.

Desde cada una de estas subcolecciones se accede a los módulos. Los módulos son tipos de objetos más particulares de la subcolección. Como ejemplo se tiene la subcolección ‘frame’ que contiene los módulos para representar marcos a pantalla completa o módulo para representar marcos en secciones concretas de la pantalla.

Dentro de estos módulos, se encuentran tanto las clases como los métodos que contiene cada módulo y atributos. Además, incorpora ejemplos de código de casos de uso, con su correspondiente resultado visual. Esto es gracias a la buena labor que hacen en la versión de la comunidad con la documentación, como se ha comentado al inicio.

Volviendo a la colección raiz, Mobject tiene múltiples subcolecciones, pero sin duda cabe destacar algunas que serán las que más uso se les dé:

- Geometry: proporciona tanto todo tipo de figuras geométricas como líneas, flechas u otro tipo de variaciones de las líneas.
- Text: facilita todas las funcionalidades relacionadas con la representación de texto usando Pango o `LaTeX`.
- Value_tracker: ofrece las utilidades para guardar y actualizar un valor.
- Mobject: clase base para los objetos que pueden ser representados.

`Manim` también da la posibilidad de organizar los mobjects en grupos, aquí nacen los 'VGroup'. Estos son un grupo de mobjects con el fin de poder escalarlos, moverlos, animarlos, ... todos juntos.

Esta organización me será de gran utilidad a la hora de efectuar animaciones sobre objetos que se componen de varias figuras geométricas, trackers e incluso texto. Asimismo, estos grupos permiten añadir al conjunto y eliminar de él durante la producción del código, permitiendo realizar movimientos y animaciones con partes separadas de estos grupos.

3.Diseño de las animaciones

A continuación, voy a analizar y explicar en profundidad cada una de las animaciones realizadas y cómo se han aplicado las funcionalidades que nos otorga `Manim`.

En este trabajo he diseñado tres animaciones utilizando técnicas distintas para cada una de ellas.

La primera animación se basa en primitivas, esto es, unidades de código elementales que realizan funciones concretas. Las primitivas permiten reproducir diferentes escenarios, con objeto de asistir en la elaboración de material docente sobre cualquier temática. El objetivo de esta animación es representar el comportamiento de los elementos implicados al realizar una planificación de CPU.

Para la segunda animación, se cambia de técnica y se elabora una animación basada en una simulación pura. En este caso, tengo un sistema de simulación independiente a la propia animación, el cual emitirá eventos. Tras esto, la animación se generará como consecuencia de la suscripción de esta a los eventos. La temática de esta animación gira en torno a las colas M/M/s.

Por último, la tercera animación adoptará una técnica de guion, es decir, esta seguirá las directrices de una secuencia previamente definida. Se puede asemejar a la composición de una película donde se tiene un guion predefinido y posteriormente se graban las escenas. La animación escenificará el proceso de paginación paginada.

3.1 Planificación de CPU

Como he comentado anteriormente, esta animación representará la planificación de CPU. En primer lugar, voy a definir el escenario, en otras palabras, los elementos que componen la animación.

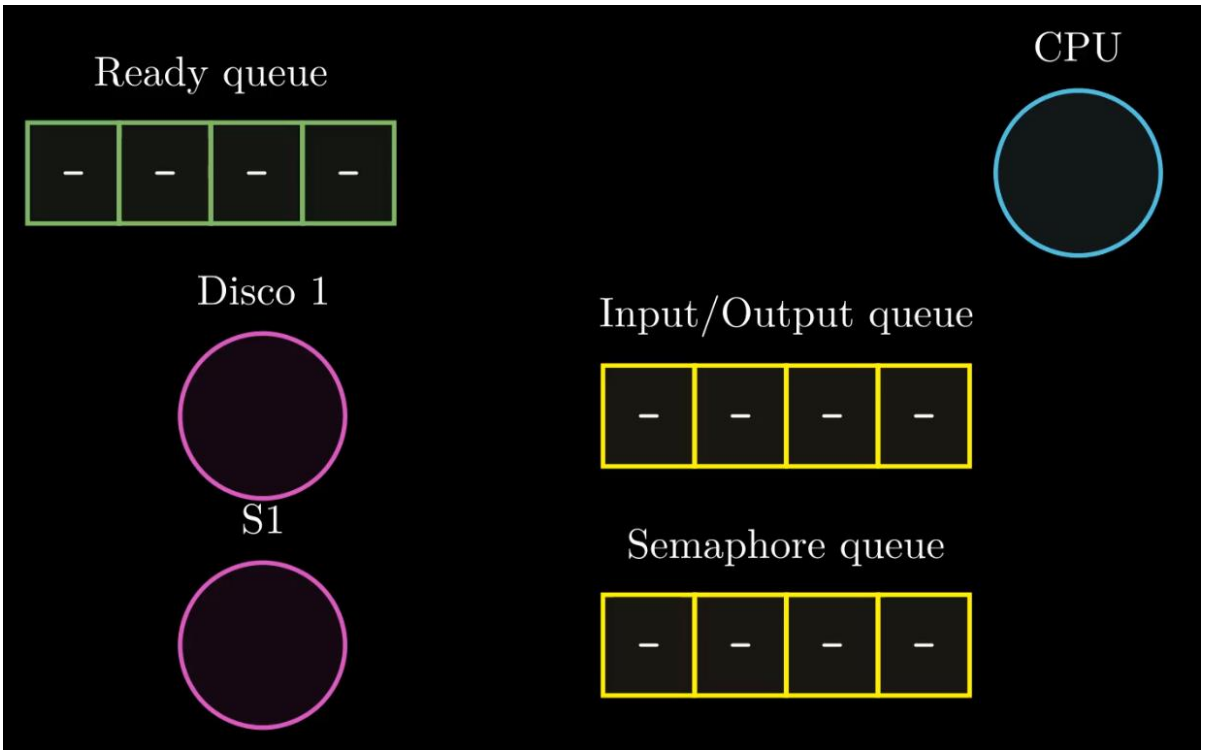

Figura 4. Escenario principal de la planificación de una CPU

En la parte superior izquierda, se tiene la cola de tareas preparadas. Esta representa una cola FIFO a la que llegarán las tareas que estén listas para ser enviadas a la CPU. Las colas FIFO (First in First out) se caracterizan por seguir el principio de 'primero en entrar, primero en salir'.

En la parte superior derecha, se encuentra la CPU. En ella entrarán las tareas para ser ejecutadas.

Justo debajo de la cola de tareas, están los elementos de entrada/salida. Estos, son elementos que permiten ingresar información al sistema informático. En mi caso, se han representado un disco y un semáforo.

Para finalizar la composición de los elementos iniciales, en la esquina inferior derecha, se localizan las colas de ambos elementos de entrada/salida. Estas colas, al igual que la de tareas preparadas, son de tipo FIFO.

En cuanto a la estructura de datos, he seguido la misma dinámica para estos seis elementos. Un elemento es el conjunto de dos estructuras, una visual y otra lógica. Para diseccionar esto, cogeré como ejemplo la CPU.

La CPU tiene como estructura visual el círculo que he explicado antes. Esta es la parte visible. Lo consigo con los VGroup que me brinda Manim. Por otro lado, tiene como parte lógica la estructura de datos de cola que nos ofrece Python. Esta no es visible y la creo gracias a la clase Queue. Con ella controlaré la posibilidad de entrar en la CPU, las salidas, etc, mientras que, con la otra, controlaré la parte visual.

```
self.cpu = Queue(maxsize=1)
self.cpu_figure = VGroup()
```

Cabe destacar que todos los elementos, en su estructura lógica, se han modelado como colas FIFO y simplemente se modifica la longitud de esta cola. Para su estructura visual, se han modelado con VGroup, los cuales contendrán las figuras necesarias, para simbolizar cada elemento y el texto asociado.

Como parte del escenario principal, hay un título, situado en la esquina superior izquierda, y un indicador de las unidades del tiempo en la esquina superior derecha.

Para llevar un control del tiempo que transcurre al realizar las tareas de la animación, usaré un 'ValueTracker'. Esta funcionalidad de `Manim` me permite establecer y actualizar el tiempo según sucedan las acciones.

```
self.tiempo = ValueTracker(0)

time_text = Tex("Time", font_size=40)
time_number = DecimalNumber(self.tiempo.get_value(),num_decimal_places=0)
```

Este escenario principal, lo crearé mediante la función 'start_animation', en la que juntaré los objetos creados antes (VGroup) con los textos identificativos de cada uno. Los textos los crearé gracias a Tex, el cual me permitirá compilar cadenas de texto con `LaTeX`.

Para la creación usaré el método 'play' que tiene la escena. Dentro de él se pueden utilizar diferentes formas para hacer aparecer los objetos.

```
self.scene.play(
            FadeIn(an_text),
            FadeIn(VGroup(time_number, time_text)),
            Create(self.rq_figure), FadeIn(rq_text),
            Create(self.cpu_figure), FadeIn(cpu_text),
            Create(self.i_o_queue_figure), FadeIn(io_text),
            Create(self.i_o_disco_figure), FadeIn(i_o_disco_text),
            Create(self.semaphore_queue_figure), FadeIn(semaphore_text),
            Create(self.i_o_semaphore_figure), FadeIn(i_o_semaphore_text)
        )
```

En este caso, 'FadeIn', que realiza un fundido, y 'Create', que crea el objeto.

Una vez situados todos los elementos iniciales, se verá el caso que escenificaré. Para ello voy a usar el siguiente fichero JSON:

```
{
  "t1": {
    "activation_time": 0,
    "behaviour": [
      {
        "CPU": 100
      },
      {
        "IO": {
          "device": "disco1",
          "duration": 30
        }
      },
      {
        "CPU": 40
      },
      {
        "P": {
          "semaphore": "S1",
          "duration": 30
        }
      }
    ]
  },
  "t2": {
    "activation_time": 10,
    "behaviour": [
      {
        "CPU": 80
      },
      {
        "IO": {
          "device": "disco1",
          "duration": 20
        }
      },
      {
        "CPU": 20
      },
      {
        "P": {
          "semaphore": "S1",
          "duration": 30
        }
      }
    ]
  }
}
```

Este archivo muestra las tareas que entrarán en acción, con su comportamiento. Cada tarea tiene 2 campos, el 'activation_time', que indica el instante de tiempo en el que se crea la tarea y el 'behaviour', que contiene tanto los elementos donde tiene que ir como el tiempo que pasará en estos.

En esta escena voy a usar la técnica de las primitivas. Las primitivas son funciones creadas para hacer una tarea en específico. La ventaja de trabajar con ellas es que puedo recrear cualquier escenario, simplemente escribiendo las primitivas sobre los objetos que se deseen.

Las primitivas que he creado son las siguientes:

- Actualizar tiempo: esta primitiva recibe por parámetro el instante de tiempo que se quiere actualizar, es decir, el instante que se quiere establecer.

- Poner mensaje: recibe una cadena de caracteres, la cual se mostrará en la parte superior central de la escena.

- Crear task: esta primitiva crea una tarea que la posiciona al lado de la cola de tareas preparadas y a la que asigna el nombre que se le pase por parámetro.

Las siguientes primitivas son las utilizadas para realizar los movimientos entre elementos. El funcionamiento de estas se puede dividir en dos fases, la lógica y la visual. En la primera, se realiza el movimiento en las estructuras lógicas que se han comentado antes. Estos movimientos no son visibles. Una vez que se han realizado estos movimientos, se realizarán los visuales, gracias a la directriz 'play'. Cabe destacar que todas estas primitivas hacen una comprobación previa, para estar seguro de que se pueda mover la tarea al elemento destino.

- Start: mueve la tarea a la cola de tareas preparadas.

- Dispatch: realiza el movimiento de la tarea desde la cola de preparados a la CPU.

- Requisa: representa el movimiento desde la CPU hasta la cola de tareas preparadas.

- Sleep disco: mueve la tarea desde la CPU hasta la cola del disco.

- Mover disco: se encarga de desplazar la tarea desde la cola del disco al disco.

- Wakeup disco: esta primitiva traslada desde el disco hasta la cola de tareas preparadas a la tarea.

- Sleep semaphore: se encarga de mover la tarea hasta la cola del semáforo desde la CPU.

- Mover semaphore: realiza el desplazamiento desde la cola del semáforo hasta el propio semáforo.

- End: elimina la tarea del escenario.

Una vez bien definidas todas las primitivas, se verá el corazón de la animación. A través de la función 'simulación' se podrá establecer la secuencia de acciones de la animación. En esta función iré situando las primitivas según mis necesidades.

```
def simulacion(self):

        task1 = self.crear_task('t1')
        self.poner_mensaje("Aparece t1")
        self.start('t1',task1)                          #APARECE T1
        self.poner_mensaje("Entra T1 en CPU")
        self.dispatch(task1)                            #ENTRA EN CPU T1


        self.actualizar_tiempo(10)

        .
        .
        .
```

En cuanto a la producción de la animación, se tienen que seguir los siguientes pasos. En primer lugar, en la consola que me proporciona el entorno de desarrollo, me situaré en la ruta donde tengo guardado el fichero Python. Tras esto, ejecutaré el siguiente comando:

```
manim -p planificador.py
```

Después de ejecutarlo, se observa que se crea el vídeo final en la carpeta 'video' que contiene el directorio 'media'.

3.2 Colas MM1

Esta segunda animación, representará el comportamiento de las colas M/M/s. La teoría de colas es el estudio matemático del comportamiento de líneas de espera. Esta se presenta cuando los 'clientes' llegan a un 'lugar' demandando un servicio a un 'servidor' el cual tiene una cierta capacidad de atención. Si el servidor no está disponible inmediatamente y el cliente decide esperar, entonces se forma la línea de espera [15].

Un modelo de sistema de colas debe especificar la distribución de probabilidad de los tiempos de servicio para cada servidor.

La distribución más usada para los tiempos de servicio es la exponencial, aunque es común encontrar la distribución degenerada o determinística (tiempos de servicio constantes) o la distribución Erlang (Gamma). Por convención, los modelos que se trabajan en teoría de colas se etiquetan de la siguiente forma:

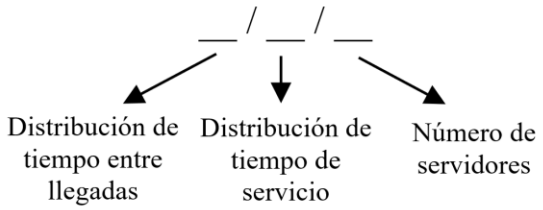

Figura 5. Modelo de teoría de colas

Las distribuciones que se utilizan son:

- **M**: Distribución exponencial (markoviana)
- **D**: Distribución degenerada (tiempos constantes)
- **E k**: Distribución Erlang
- **G**: Distribución general

Por lo que la siguiente notación, M/M/s, representa un modelo donde tanto los tiempos entre llegada como los tiempo de servicio son exponenciales y se tienen s servidores.

Con esta visión general de la teoría de colas, se puede explicar la animación. Esta, se compone de dos elementos, por un lado, está la simulación y, por otro, la creación de la animación a partir de la suscripción a eventos.

La simulación se consigue gracias a varios ficheros Python que crean un ejemplo de comportamiento de las colas M/M/s. Para el funcionamiento de la simulación, se necesitará el número de servidores y dos parámetros, tanto la tasa de llegadas como la de salidas.

La generación de la animación se consigue gracias a la suscripción a eventos, los cuales indican la acción que se debe realizar.

La conexión entre la simulación y la animación se realiza de la siguiente forma. La simulación emite eventos a lo largo de su ejecución para que posteriormente la animación se cree por la suscripción a estos.

Para explicar el diseño de esta animación voy a separarla. Por un lado, desarrollaré el fichero que contiene la animación en el que se usa efectivamente `Manim`, y, por otro lado, explicaré los ficheros que hacen posible la simulación.

En el fichero que contiene el comportamiento de la animación se encuentra, primeramente, la definición del escenario principal, al igual que en el caso anterior. Se ha creado la función 'start_animation' la cual pone el título en la parte superior central, crea la 'waiting area' y el 'service node'.

Cabe mencionar sobre estos elementos que el área de espera la he modelado como una cola FIFO de longitud diez. Y el nodo de servidores como una serie de círculos dependiendo del número de servidores que tenga la simulación.

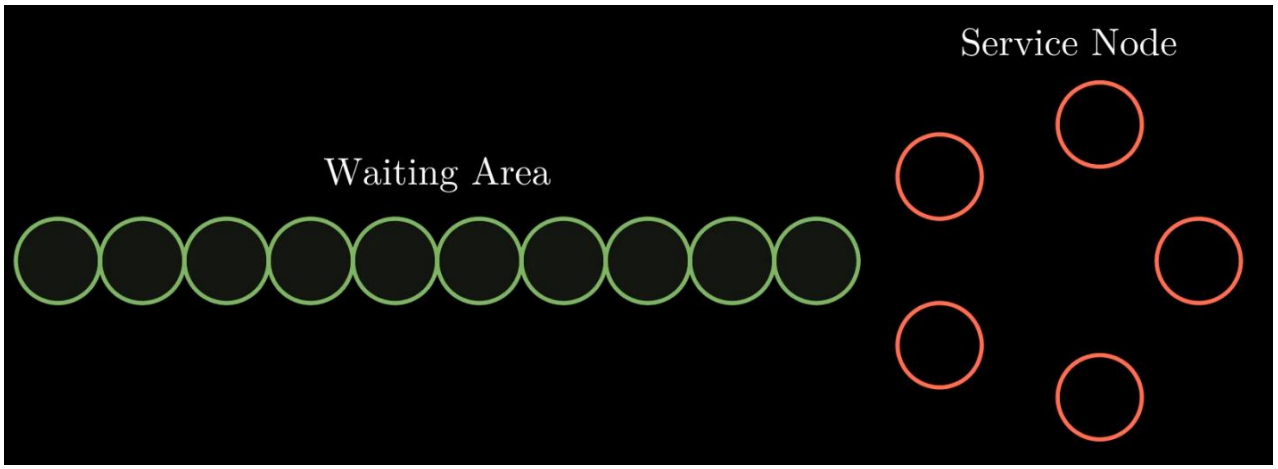

Figura 6. Escenario de una simulación de colas M/M/5

Aparte de la definición del escenario principal, en este fichero tengo la implementación visual de los eventos que se emitirán. Como comenté antes, el nexo entre la simulación y la animación son los eventos y aquí se tendrá lo que tiene que hacer `Manim` en el momento en el que reciba uno de los eventos que hay. Los eventos son:

- ArrivalDispatch: este evento maneja el caso en el que se asigna un servidor a una llegada.
- ArrivalEnqueue: cuando una llegada no tiene un servidor libre, por lo que se tiene que encolar.
- Departure: representa la salida y el establecimiento de un servidor libre.
- DepartureDispatch: este evento es la liberación de un servidor junto con una salida que va hacia un servidor libre desde el área de espera.

En cuanto al primer evento, su funcionamiento es el siguiente: gracias a la simulación obtengo el servidor que está libre y la primera posición liberada de la cola. Creo con `Manim` un círculo y lo sitúo a la izquierda de la cola. Tras esto, y ayudándome de la directriz 'play', animo el elemento para que se posicione en el primer espacio libre de la cola. Una vez finalizado este proceso, animo este mismo elemento para que realice el movimiento hasta el servidor correspondiente utilizando el índice obtenido anteriormente.

Para el segundo evento, como se encolará el elemento, necesito el índice de la cola con la primera posición libre. Después, animo nuevamente el elemento desde la parte izquierda de la cola hasta el lugar correspondiente.

El evento de Departure simplemente consigue de la simulación el servidor donde está el elemento saliente y realiza el desplazamiento hacia la derecha del nodo de servidores dejando liberado dicho servidor.

Para finalizar, el cuarto evento obtiene el servidor que será liberado. Posteriormente, se efectúa la salida del elemento del servidor y realiza el traslado del primer elemento de la cola al servidor libre.

Estos 4 eventos tienen el enlace con los ficheros de simulación mediante el siguiente código:

```
self.simulator.event_handler.link(self._on_arrival_dispatch, 'onArrivalDispatch')

self.simulator.event_handler.link(self._on_arrival_enqueue, 'onArrivalEnqueue')

self.simulator.event_handler.link(self._on_departure, 'onDeparture')

self.simulator.event_handler.link(self._on_departure_dispatch, 'onDepartureDispatch')
```

Con el método 'link' consigo establecer la relación entre las funciones previamente definidas y los eventos que emitirá la simulación.

A continuación, se verán en profundidad los ficheros que hacen posible esta simulación. El script principal es 'mmnsimulator', en él tengo todos los atributos de la simulación, como son el número de servidores, la tasa de llegadas, la tasa de salidas o el tiempo, además de la definición de los eventos.

```
def __init__(self, num_servers=1, la=1, ld=1, simulation_end_time=0):
        self.simulation_end_time = simulation_end_time
        self.num_servers = int(num_servers)
        self.num_sources = 1
        self.la = la
        self.ld = ld
        self.simulation_time = 0
        self.servers = [None] * self.num_servers
        self.waiting_queue = deque()
        self.event_handler = EventHandler('onDeparture', 'onDepartureDispatch',
'onArrivalDispatch', 'onArrivalEnqueue', 'onError')

        np.random.seed(2)
```

En cuanto a funciones se refiere, la principal, 'simulate()', se encarga de ir emitiendo los eventos según corresponda. El funcionamiento interno es el siguiente:

Primero comprueba de qué tipo es el evento en curso. Este puede ser de tipo llegada o de tipo salida. En caso de ser una llegada, intenta asignar esta llegada a un servidor libre. Si es satisfactoria esta asignación emite el evento 'ArrivalDispatch'. En caso contrario, es decir, que todos los servidores estén ocupados, la llegada se encolará en el área de espera y se emite el evento 'ArrivalEnqueue'.

```
if event_type == self.ARRIVAL:
      try:
            empty_server_index = self.servers.index(None)
            self.event_handler.fire('onArrivalDispatch', event_time)
            self.servers[empty_server_index] = self.simulation_time +
            self._random_departure()
     except ValueError:
            self.waiting_queue.append(event_time)
            self.event_handler.fire('onArrivalEnqueue', event_time)

     finally:
            continue
```

Si es una salida, se comprueba en primer lugar si hay algún proceso en la cola esperando. Si es el caso, se deja libre el servidor y se asigna el primer proceso de la cola a un servidor, luego se emite el evento 'DepartureDispatch'. Por último, si no hay procesos en espera, simplemente se realiza la salida y la emisión del evento 'Departure'.

```
if event_type == self.DEPARTURE:
      if len(self.waiting_queue):
            self.waiting_queue.pop()
            self.servers[event_source] = self.simulation_time +
            self._random_departure()
            self.event_handler.fire('onDepartureDispatch', event_time, event_source)
      else:
            self.event_handler.fire('onDeparture', event_time, event_source)
            self.servers[event_source] = None
      continue
```

La emisión de eventos la hace gracias al comando 'fire', el cual es un disparador que pone en marcha la función que tiene enlazada.

Para que la simulación funcione como se desea, hay que recolectar los datos que se pasarán en el momento de ejecutarla, que se explicará posteriormente, para poder dárselos a este script. De esto se encarga otro fichero, 'simcommand'. En este script defino el manejo de los atributos que tendré en el comando.

El comando para la generación de la animación, al contrario que antes, no se utiliza la directriz 'manim' si no que será la siguiente:

```
python app.py sim "M/M/s --la=n1 --ld=n2
```

En este caso, utilizo la palabra reservada 'python' ya que la renderización se hace desde dentro del fichero 'simcommand'. Seguidamente, se encuentra el archivo de Python que quiero ejecutar y, tras esto, el comando que habrá que *parsear*.

El primer campo será la disciplina con el formato M/M/s, donde se sustituirá el parámetro 's' por el número de servidores que se quieran en nuestra simulación. El segundo, 'n1', será la tasa de llegadas y el tercero, 'n2', la de salidas, ambos establecidos en salidas/llegadas por unidad de tiempo.

Una vez que se ha comprendido el funcionamiento de la llamada, vuelvo al script 'simcommand'. Tras haber *parseado* estos campos que he comentado, se crea una instancia del simulador pasándole el número de servidores, la tasa de llegadas y la tasa de salidas. Después, comienza la simulación.

```
mmn = MmnSimulator(
            num_servers,
            la=float(self.option("la")),
            ld=float(self.option("ld")),
            simulation_end_time=10
        )
```

Finalmente, como se ha comentado antes, para esta animación no se utiliza la directriz manim en el momento de generar el vídeo. Esto se debe a que la renderización se hace ahora con el siguiente fragmento de código:

```
animacion = ManimAnimation(mmn)
animacion.render()
```

Con el método 'render' consigo el mismo resultado que invocando por consola al comando 'manim'.

Para terminar con los ficheros de la simulación, tengo el script 'app'. Este es el fichero que se ejecuta al escribir el comando en la consola. Su función es, simple y llanamente, iniciar el manejador de comandos y la propia simulación.

Aunque ya he dado los detalles anteriormente, si se quiere realizar una producción de un video, se haría de la siguiente manera. Pongamos que voy a crear un vídeo con los siguientes parámetros de simulación:

- Servidores = 2
- Tasa de llegadas = 1
- Tasa de salidas = 2

Solamente ejecuto en la consola:

```
python app.py sim "M/M/2" --la=1 --ld=2
```

Y tendré el vídeo listo.

3.3 Paginación

En este último caso, voy a utilizar un guion como base para realizar la animación. Con esto, creo las escenas correspondientes a la narrativa para, finalmente, desarrollar la composición de un vídeo.

El guion cuenta el proceso que se lleva a cabo al emplear la técnica de paginación. Para comprender mejor esta última animación, voy a explicar previamente qué es la paginación [16].

Si se habla de memoria física en un computador es muy importante organizarla de manera correcta. Una buena estrategia es la paginación de memoria. Es importante elegir estrategias o modelos de organización que aprovechen al 100% la memoria física. Esto es debido a que esta cumple con un papel vital en el correcto funcionamiento del sistema operativo.

La paginación es un modelo de organización de memoria física en el que se divide toda la memoria física en porciones de igual tamaño. Estas porciones que menciono reciben el nombre de páginas.

La funcionalidad principal de esta técnica es la de transformar una dirección virtual a física. Para ello, los programas se dividen en unidades lógicas llamadas 'páginas' que se utilizarán como unidad de almacenamiento de información. La memoria física se divide en marcos, que son bloques del mismo tamaño. De esta manera, los marcos y las páginas tienen un tamaño fijo. Gracias a la ayuda de la tabla de páginas se transforman las direcciones virtuales o lógicas a físicas. La tabla de páginas guarda la dirección del marco en la que está almacenada cada página. Existe una tabla por proceso y con tantas entradas como páginas tiene éste [17].

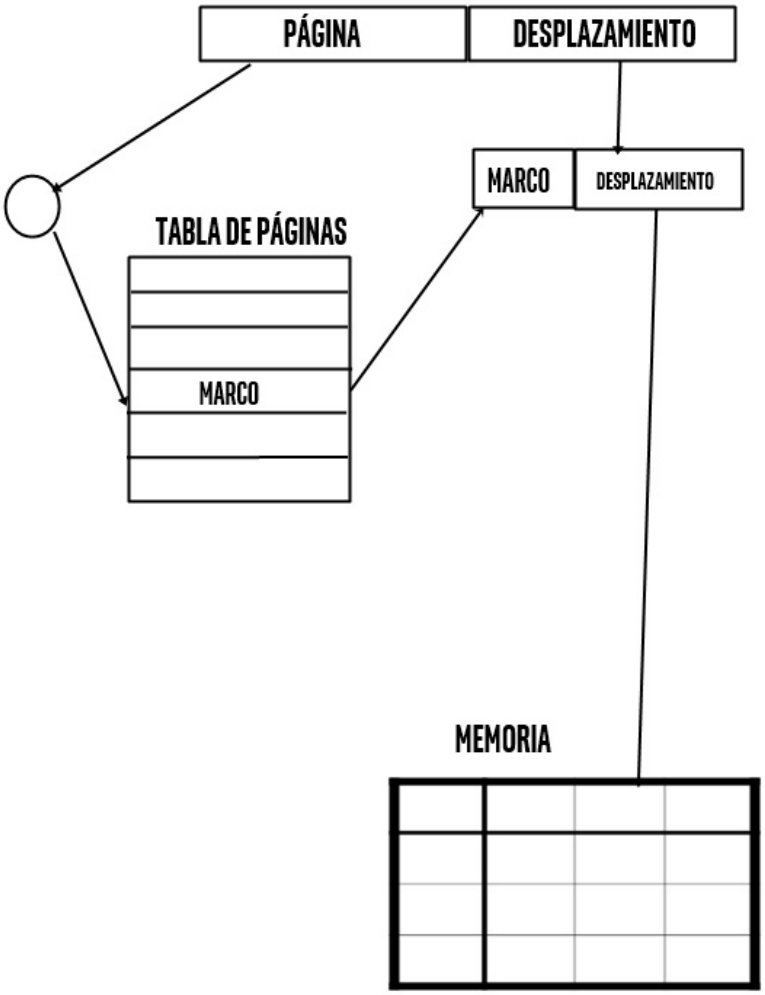

Figura 7. Esquema de la paginación

Una vez explicado el concepto de paginación, se verá el diseño de la animación. Como se ha mencionado antes, en primer lugar, se van a producir por separado todas las escenas para, posteriormente, realizar la composición del vídeo con las escenas y el audio.

En relación con las escenas, el vídeo final está compuesto por 12 de ellas, las cuales se van a detallar a continuación:

3.3.1 Escena 1: Estructura de la CPU

En esta primera escena presento nuestro caso de estudio por lo que representaré una CPU. Junto a ella tendré un bus de datos y un bus de direcciones. La animación consistirá en hacer ver cómo de grande es el bus de direcciones, es decir, cuántas direcciones es capaz de generar.

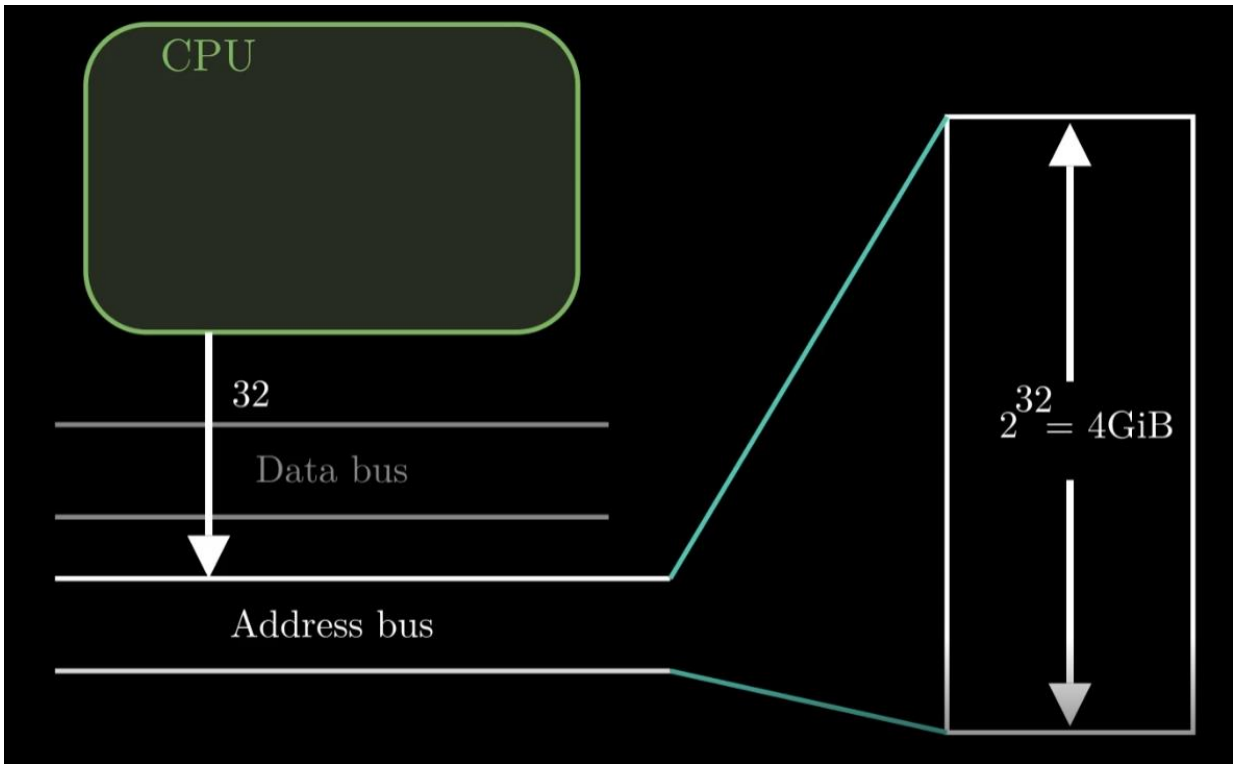

Figura 8. Elementos de la CPU

En cuanto a nivel visual, todos los textos han sido manejados mediante 'Tex', para la CPU se ha hecho uso de 'RoundedRectangle' y para los buses y flechas 'Line' y 'Arrow' respectivamente. Para la propia animación se ha utilizado un cuadrado ('Square') junto con una llave ('Brace') el cual se traslada y transforma en un rectángulo. Finalmente, la potencia que representa el tamaño y su incremento se consigue gracias a 'ValueTracker' y un updater. De esta forma se alcanza esa transición entre el primer valor y el dato final.

3.3.2 Escena 2: Organización de la memoria

Para esta segunda escena, tengo que escenificar cómo se divide la memoria en marcos y cual es realmente el tamaño de estos.

En la parte derecha de la escena tendré la memoria, la cual se irá dividiendo en los diferentes marcos. A la izquierda de la memoria se encontrarán el tamaño y el número de marcos, que irán ascendiendo según avanza la animación. En el plano inferior, tendré la fórmula que hace estos cálculos, animada para salir al final.

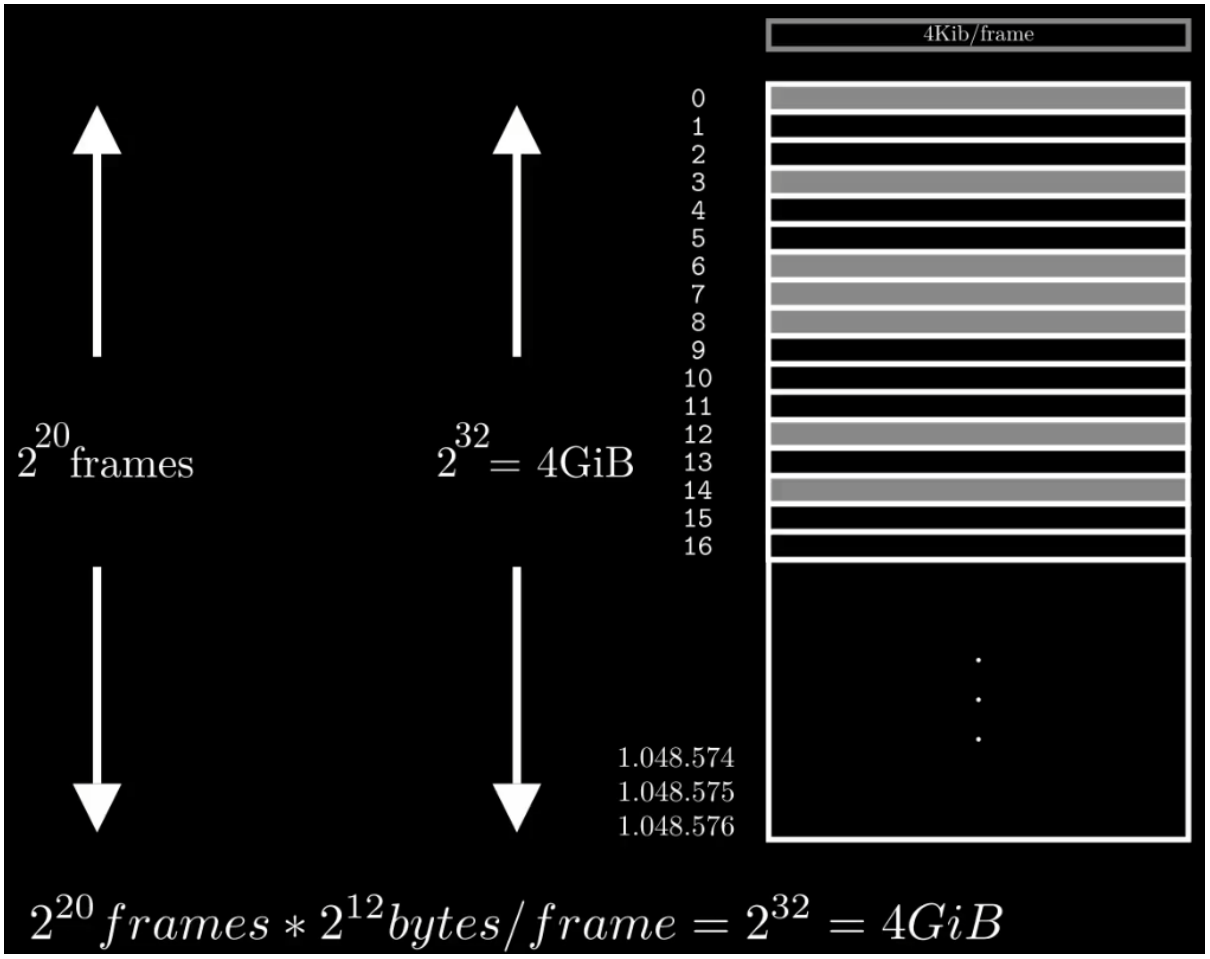

Figura 9. Estructura de la memoria

La fórmula la creo mediante 'MathTex' mientras que la memoria es un conjunto de 'Rectangle' formando un VGroup. Además, agrego color a algunos de los marcos; de esta forma se representan marcos vacíos y marcos ocupados. Tanto para el tamaño de la memoria como para el número de marcos utilizaré la misma técnica de flechas y 'ValueTracker' que para la escena anterior.

3.3.3 Escena 3: Producción de direcciones físicas

Esta escena muestra cómo genera la CPU una dirección en el bus de direcciones, su organización y el paso entre cambios de marcos y desplazamientos.

La escena se compone de una representación de una dirección situada en la parte superior, acompañada por el marco y offset correspondiente a su derecha.

La animación muestra dos cambios tanto de offset como de marco en la parte inferior.

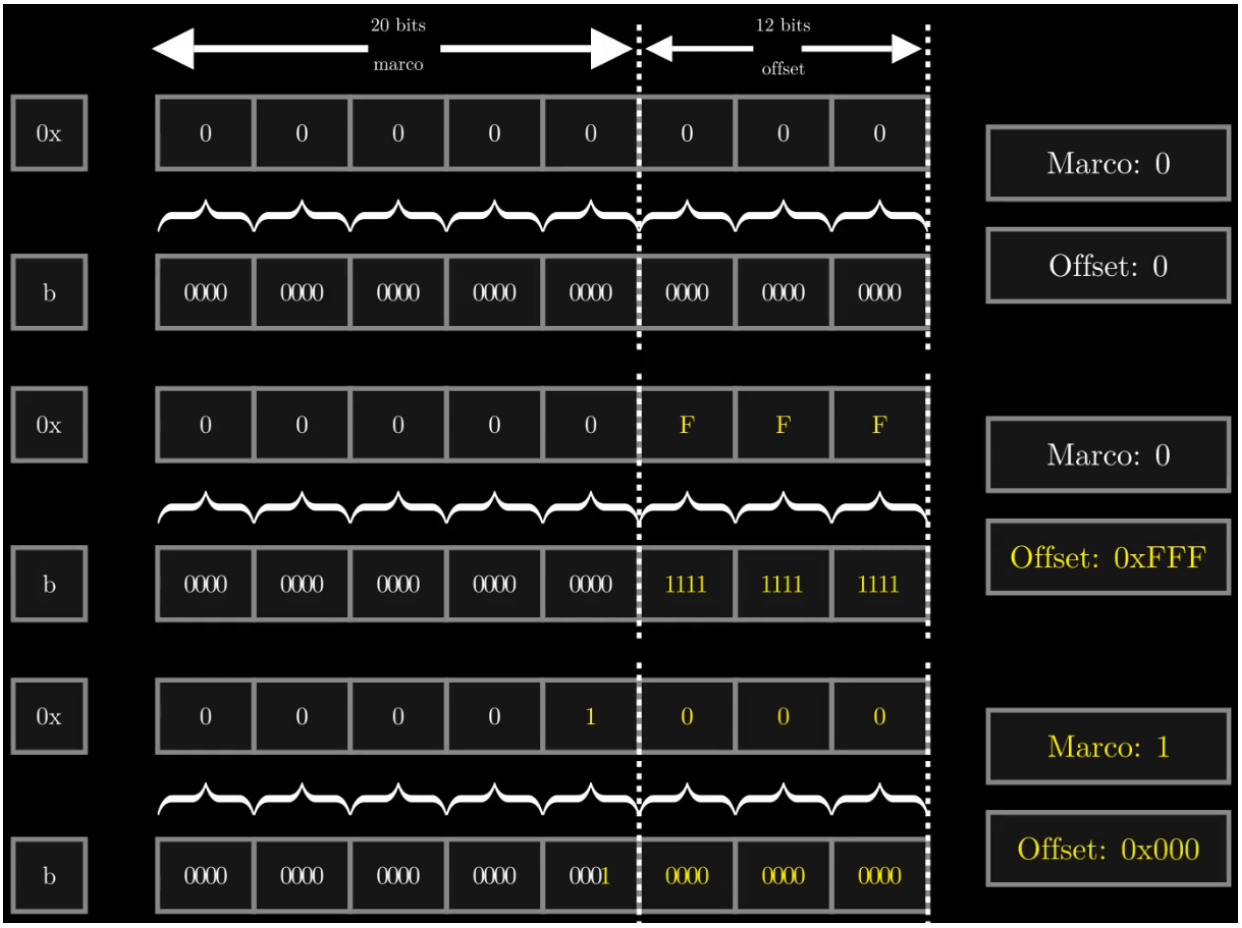

Figura 10. Generación de direcciones

Esta escena, a nivel visual, es de las más complejas ya que cada uno de los buses es un VGroup formado por 'Rectangle' y dentro de cada uno de estos tengo 'DecimalNumber'. Además, todas las llaves se añaden al VGroup.

La animación realiza tres pasos: en el primero, presenta la estructura mostrando una dirección inicial con marco y offset en 0. El segundo paso anima el cambio al offset FFF para que, en el tercer paso, se vaya al offset 0 de nuevo pero del siguiente marco.

3.3.4 Escena 4: Ejemplo de programa

Llego a la cuarta escena donde se ve el caso en el que se quiere cargar un programa en memoria para ejecutarlo y dicho programa está almacenado en el disco duro del computador. Dispongo la representación del disco en el centro de la imagen y dentro de él posiciono el programa con su bloque de código y su bloque de datos.

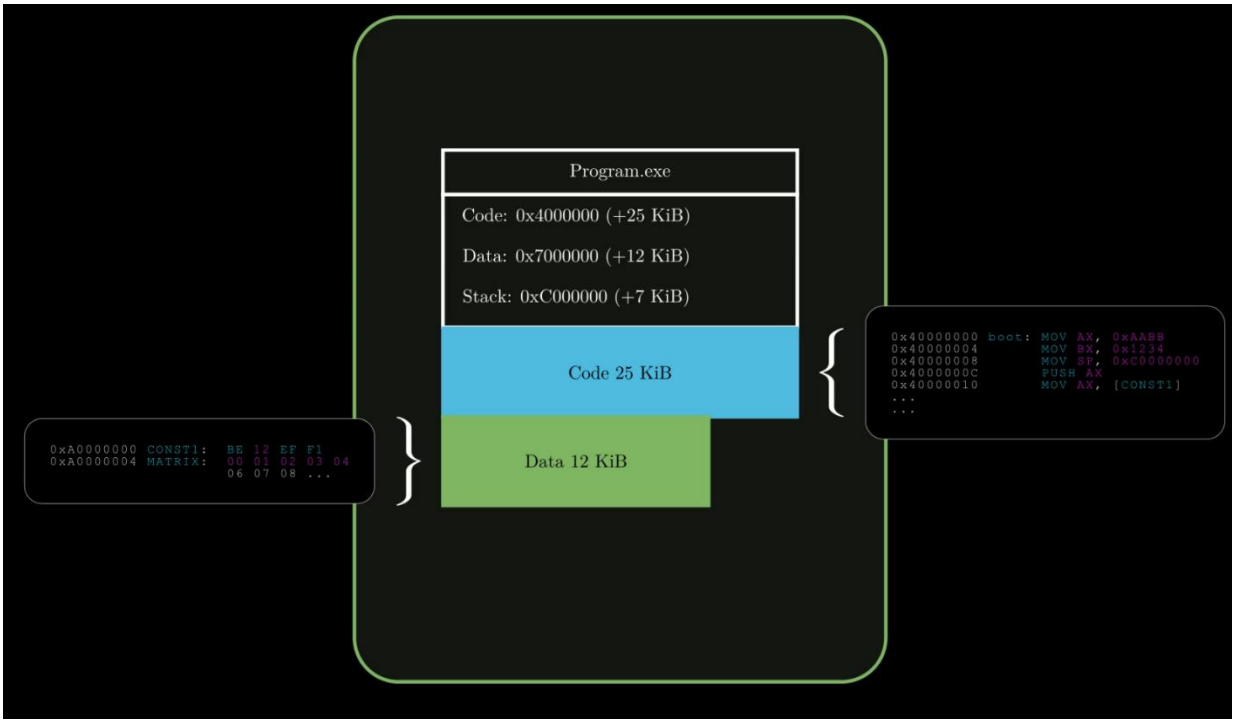

Figura 11. Estructura de un programa almacenado en disco duro

La animación consiste en ir mostrando sucesivamente los elementos de esta escena. Primero el disco, luego el programa, para continuar con los bloques de código y datos y finalizar con el contenido de estos.

Cabe destacar en esta animación el uso de 'Code' que me permite animar trozos de código indicando qué lenguaje es.

3.3.5 Escena 5: Proceso de traducción de una dirección

La escena número cinco tiene una importancia relevante en el vídeo ya que es la que muestra cómo se traducen direcciones entre la CPU y memoria. La disposición es la que sigue: en la esquina superior izquierda está la CPU, en la derecha, la memoria y, en la zona baja, los buses que las interconectan. La MMU se coloca en la parte inferior también, entre los buses.

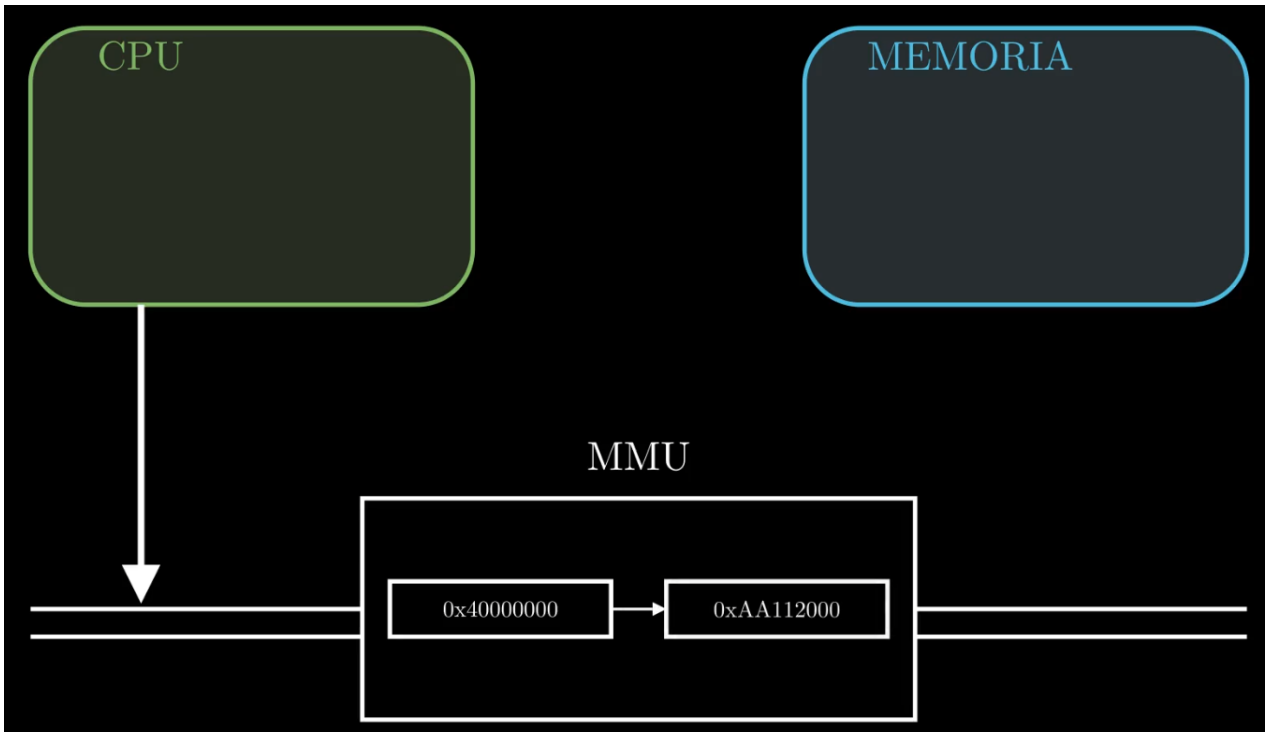

Figura 12. Proceso de traducción de direcciones

Esta escena presenta en primera instancia los elementos de CPU, memoria y buses para luego ir mostrando la MMU y cómo queda intercalada entre CPU y memoria. Una vez establecido este escenario, se anima una dirección en la CPU y esta procede a moverse hacia la parte izquierda de la MMU. Aquí se producirá la traducción, que se consigue desvaneciendo la dirección original y revelando la nueva. Finalmente, esta nueva dirección se desplaza hasta la memoria.

3.3.6 Escena 6: Producción de direcciones virtuales

En este momento, lo que representaré es cómo se organiza cada dirección virtual. Es una escena idéntica a la escena número 3 pero representando las páginas y no los marcos.

3.3.7 Escena 7: Producción de direcciones virtuales II

La escena siete es la continuación de la anterior, pero ejemplificando el caso en el que la CPU busca una dirección en concreto. Como elementos adicionales tengo una llave en la parte inferior correspondiente a los bits de la página junto con un recuadro indicando que página es.

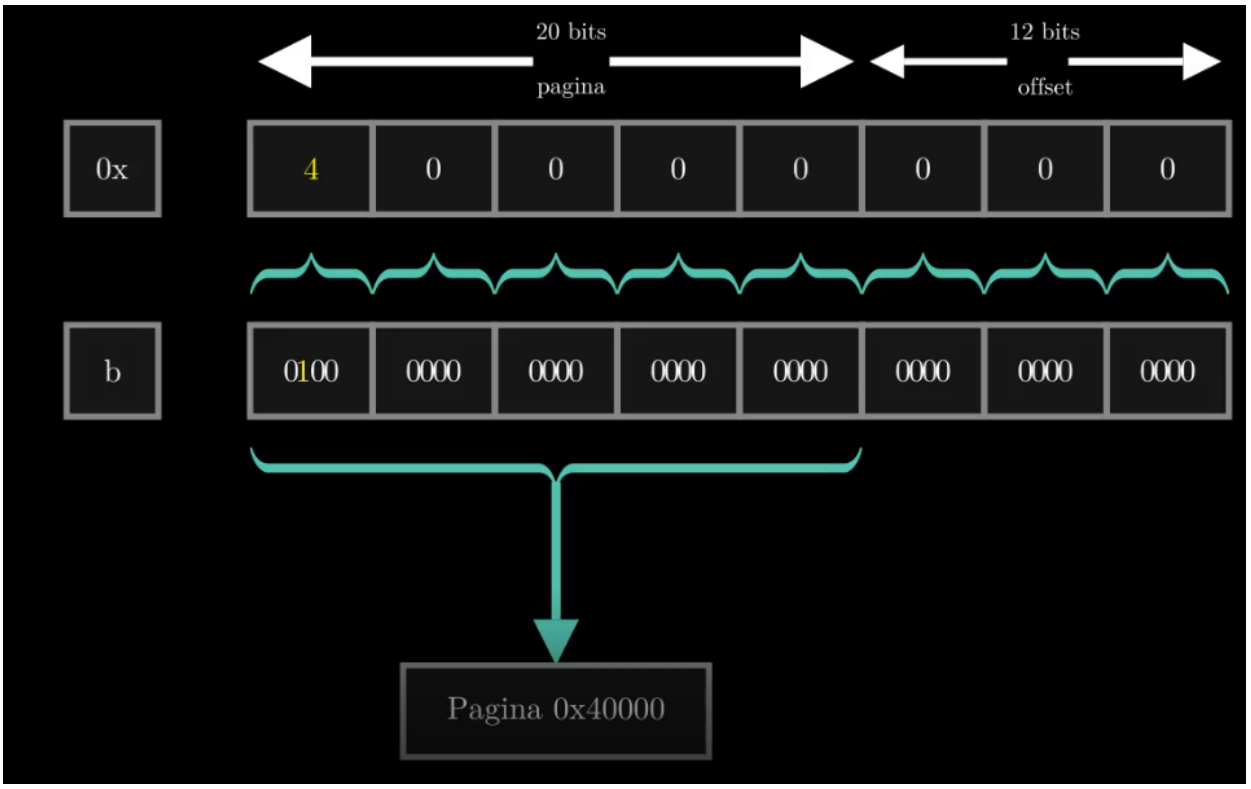

Figura 13. Representación de una página

La animación consiste en representar en los bits la página 4 para posteriormente dibujarla en un recuadro en la parte inferior.

3.3.8 Escena 8: Organización de la tabla de páginas

Con la siguiente escena represento a la tabla de páginas. Esta se dispondrá a la derecha mientras que a su izquierda estará colocado el tamaño de dicha tabla.

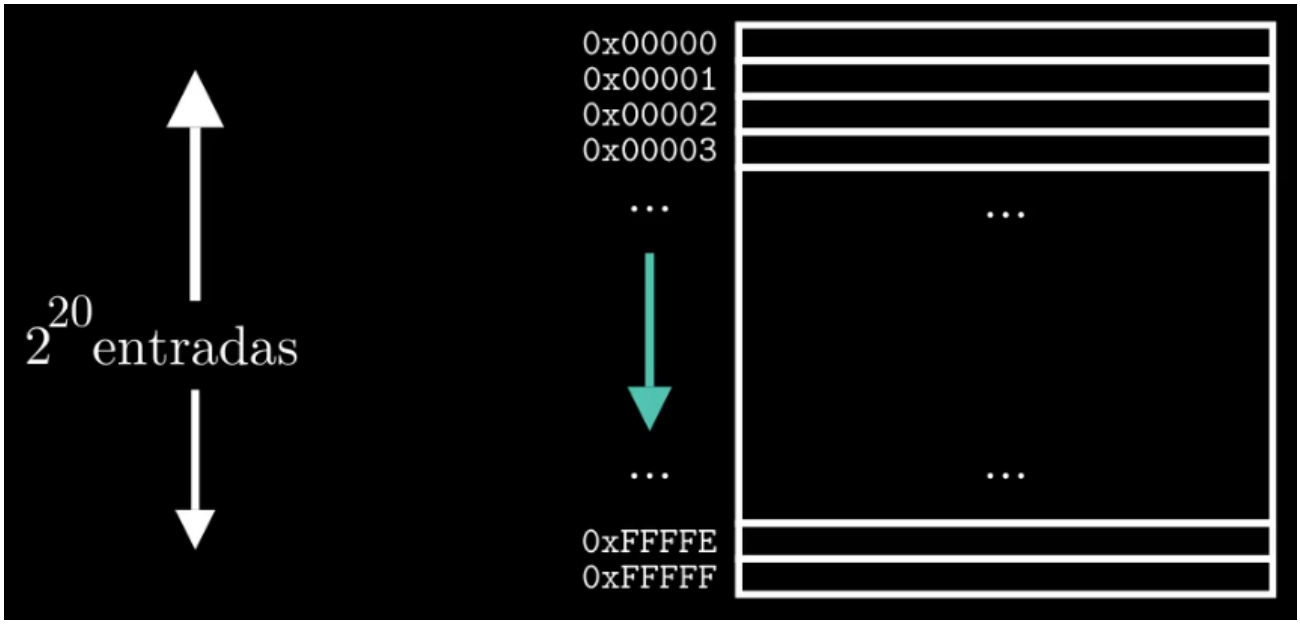

Figura 14, Estructura de la tabla de páginas

La tabla irá apareciendo en imagen progresivamente, entrada a entrada. Para finalizar la animación se muestra el tamaño, con los updaters utilizados anteriormente.

3.3.9 Escena 9: Organización de la tabla de páginas II

Con esta escena, añado aún más información sobre la tabla de páginas a la escena anterior. Partiendo de la base de la escena 8, se agrega lo que debe almacenar cada entrada y qué tamaño tendrán.

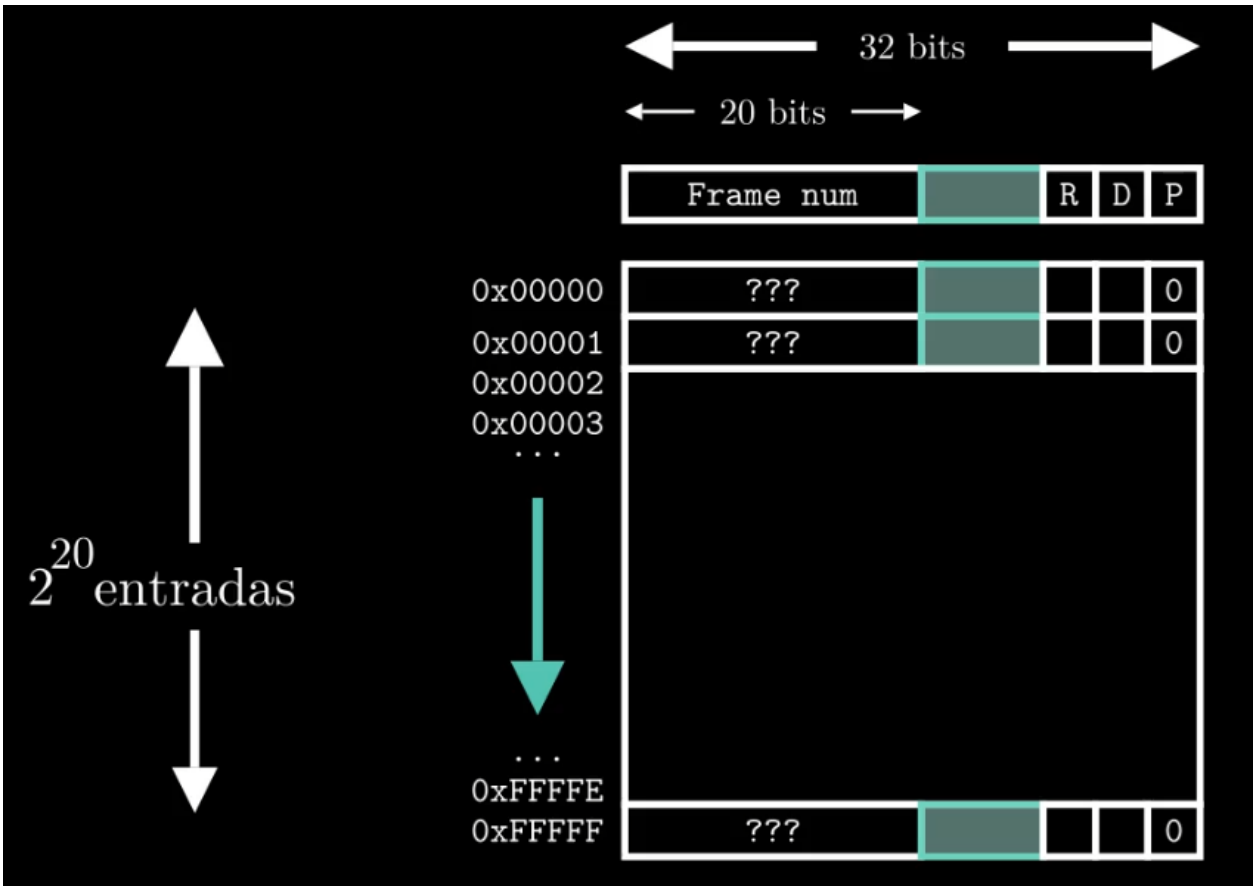

Figura 15. Campos de la tabla páginas

Cabe destacar sobre esta animación que la entrada de ejemplo que se sitúa sobre la propia tabla de páginas aparece en última instancia y los bits que contiene en su interior van mostrándose progresivamente.

3.3.10 Escena 10: Búsqueda del marco

Al igual que la escena número 5, la 10 tiene una importancia grande dentro de la narrativa. Esta escena es la finalización de las dos anteriores, donde se muestra de qué manera se hace la traducción efectiva.

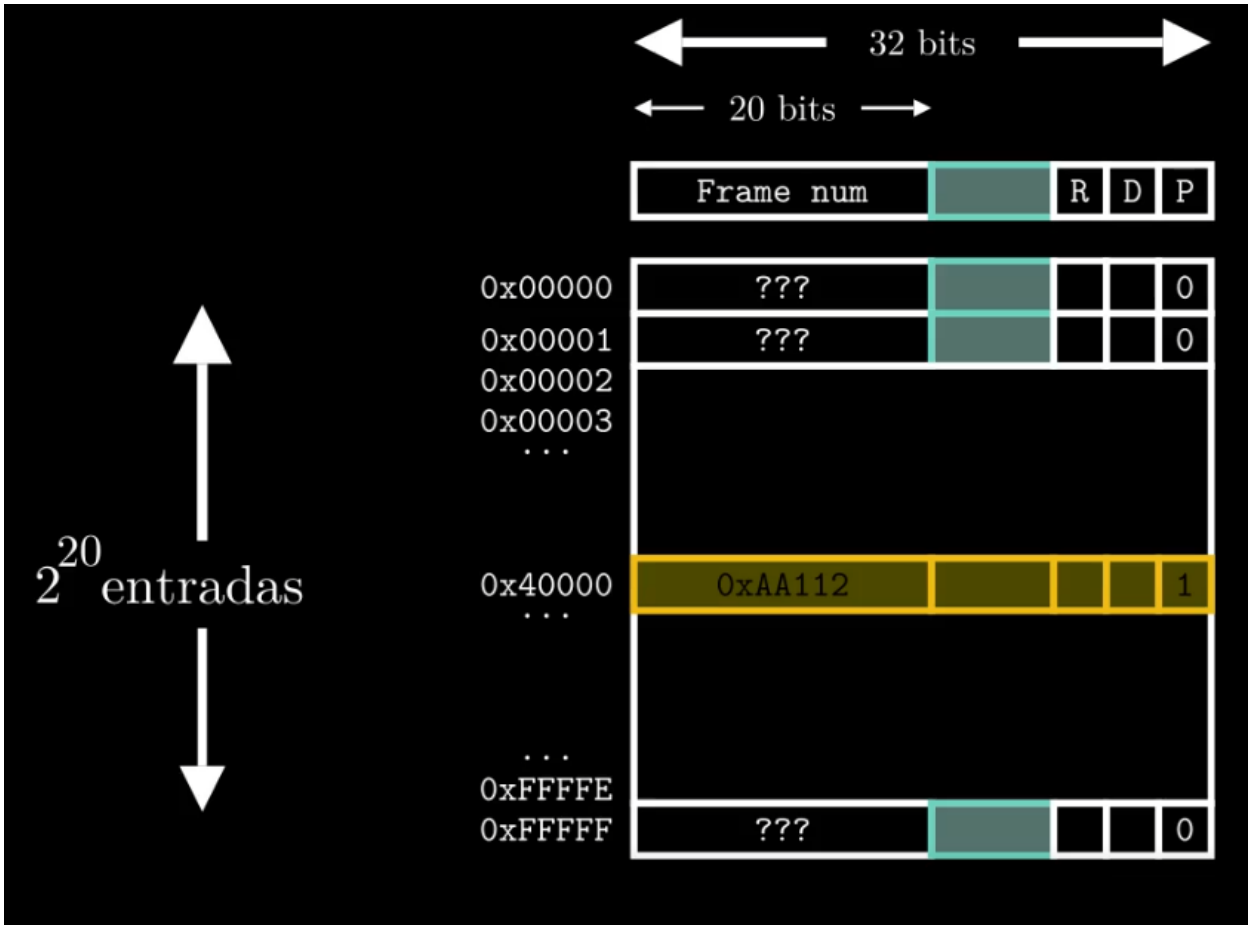

Figura 16. Traducción de la página al marco

A nivel visual, se ve cómo aparece la página junto a la entrada que tiene el número de marco correspondiente.

3.3.11 Escena 11: Proceso de traducción de una dirección II

Con esta escena se junta el proceso que se ha explicado en la escena anterior con la escena 5 para ver el proceso completo.

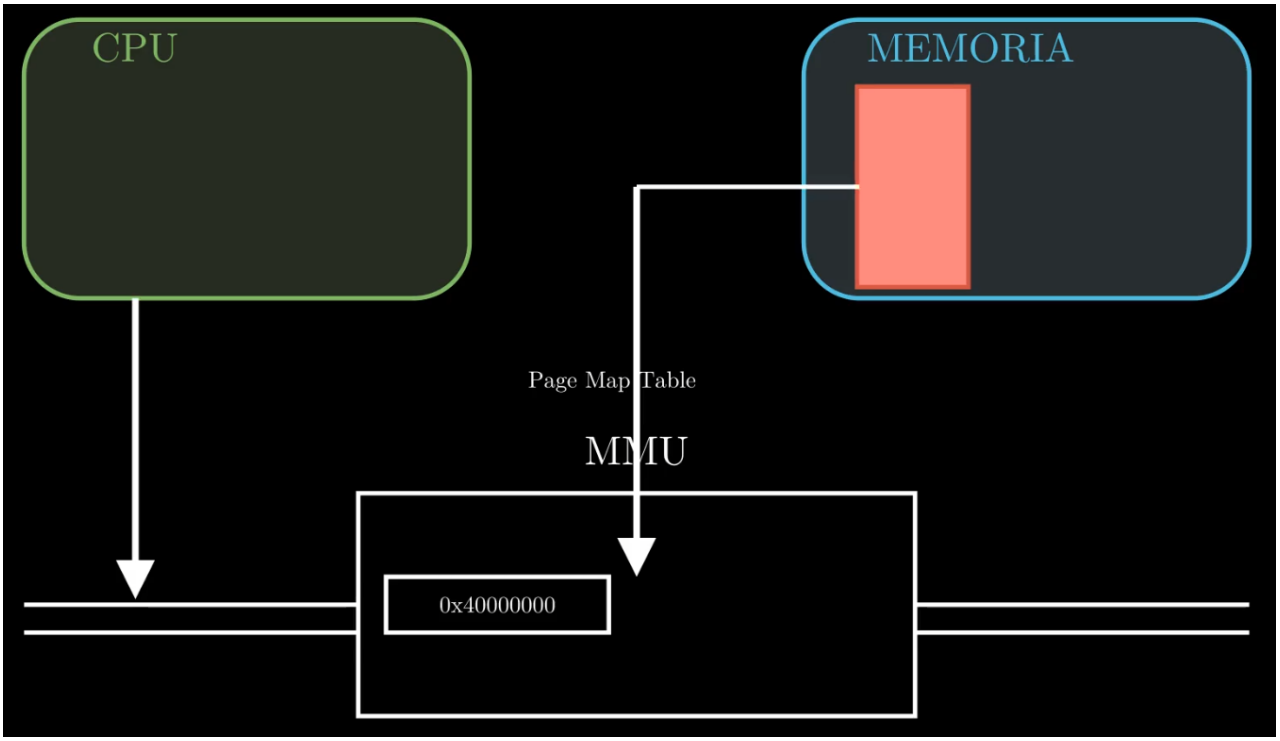

Figura 17. Proceso de traducción desde CPU hasta memoria

Aquí, se encuentran las animaciones de las direcciones y, además, unas transiciones de flechas para mostrar cómo obtiene la MMU la tabla de páginas que reside en memoria.

3.3.12 Escena 12: Presentación de la fórmula de la tabla de páginas

Para finalizar, tengo una imagen final de la fórmula que me dice cuánto ocupa la tabla de página ya que es uno de los principales problemas de esta técnica.

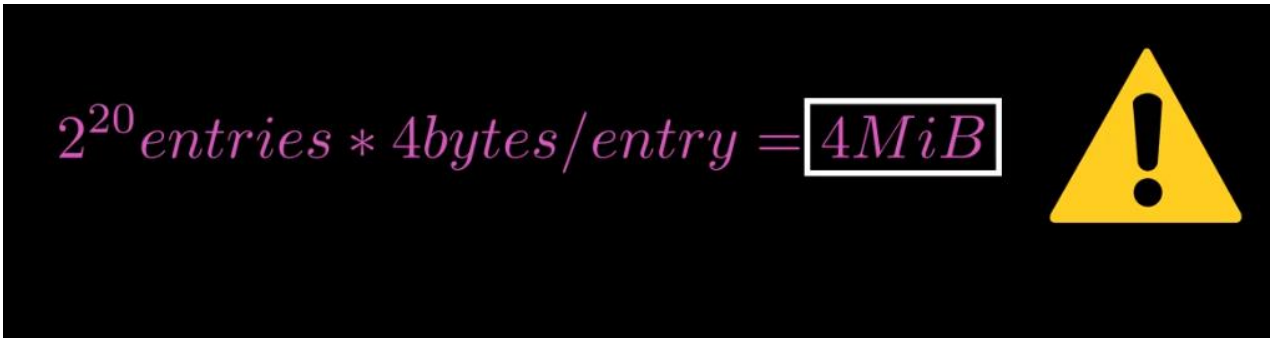

Figura 18. Fórmula para el tamaño de la tabla de páginas

Tras haber visto en detalle las escenas que componen la parte visual, quedaría por ver la parte del audio.

Este se ha realizado por escenas, es decir, la narrativa estaba dividida concordando con cada una de las escenas. Sin embargo, al efectuar esta tarea surge el siguiente problema: el audio por escena no está alineado con el tiempo de la animación por lo que hace falta un ajuste. En la siguiente tabla se muestran los desfases por escena:

	Tiempo Escena	Tiempo Audio	Corrección
Escena 1	7s	30s	23s
Escena 2	8s	37s	29s
Escena 3	30s	30s	0s
Escena 4	4s	74s	70s
Escena 5	8s	74s	68s
Escena 6	3s	33s	30s
Escena 7	5s	13s	8s
Escena 8	7s	28s	21s
Escena 9	10s	47s	37s
Escena 10	9s	20s	11s
Escena 11	11s	29s	18s
Escena 12	3s	33s	30s

Tabla 1. Desfases de las escenas

Para llevar a cabo esta corrección de timing, me voy a ayudar de dos funcionalidades que me brinda `Manim`. La primera es utilizar el método 'wait()', el cual recibe un número que será el tiempo en segundos que quiero que se detenga la animación. La otra, es el parámetro 'run_time', que se incluye dentro del método 'play' para indicarle cuánto tiempo tiene que durar dicha animación.

Por último, como ya tengo las escenas bien ajustadas y el audio, tengo que hacer la composición del video. Para ello voy a utilizar un editor de vídeo. En mi caso, he elegido DaVinci Resolve, que es un software de edición de vídeo no lineal y gratuito desarrollado por *Blackmagic Design*. DaVinci Resolve combina herramientas profesionales para la edición, talonaje, composición de efectos visuales, creación de animaciones gráficas y mezcla de audio.

De cara a poder utilizarlo, tengo que dirigirme a DaVinci [18] y descargarme el programa para el sistema operativo adecuado.

Un vez descargado e instalado el editor, voy a crear un nuevo proyecto. Al hacerlo me encuentro la siguiente interfaz:

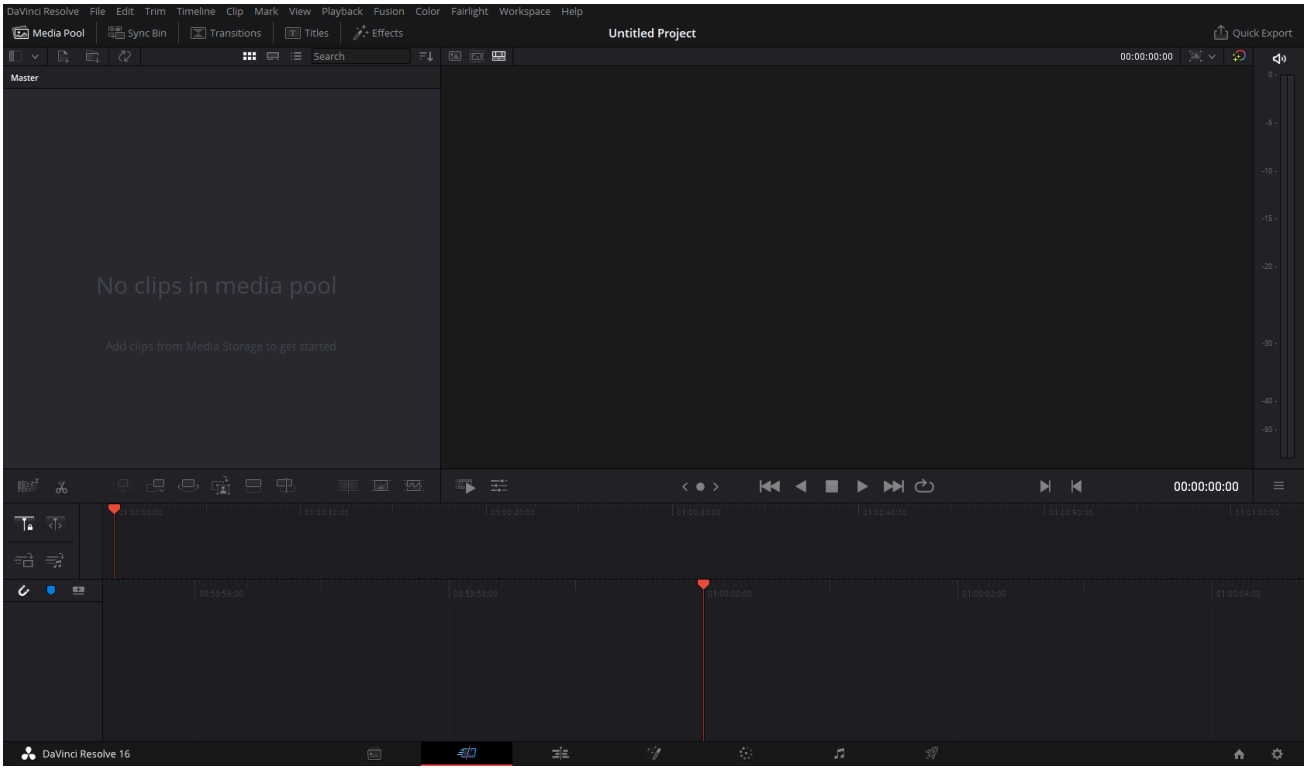

Figura 19. Interfaz principal de DaVinci Resolve

Antes de comenzar, voy a crear el espacio de trabajo. Para ello, creo dos carpetas, una para almacenar los ficheros de audio y otra para los de vídeo. En ellas introduzco todos los archivos multimedia que tengo.

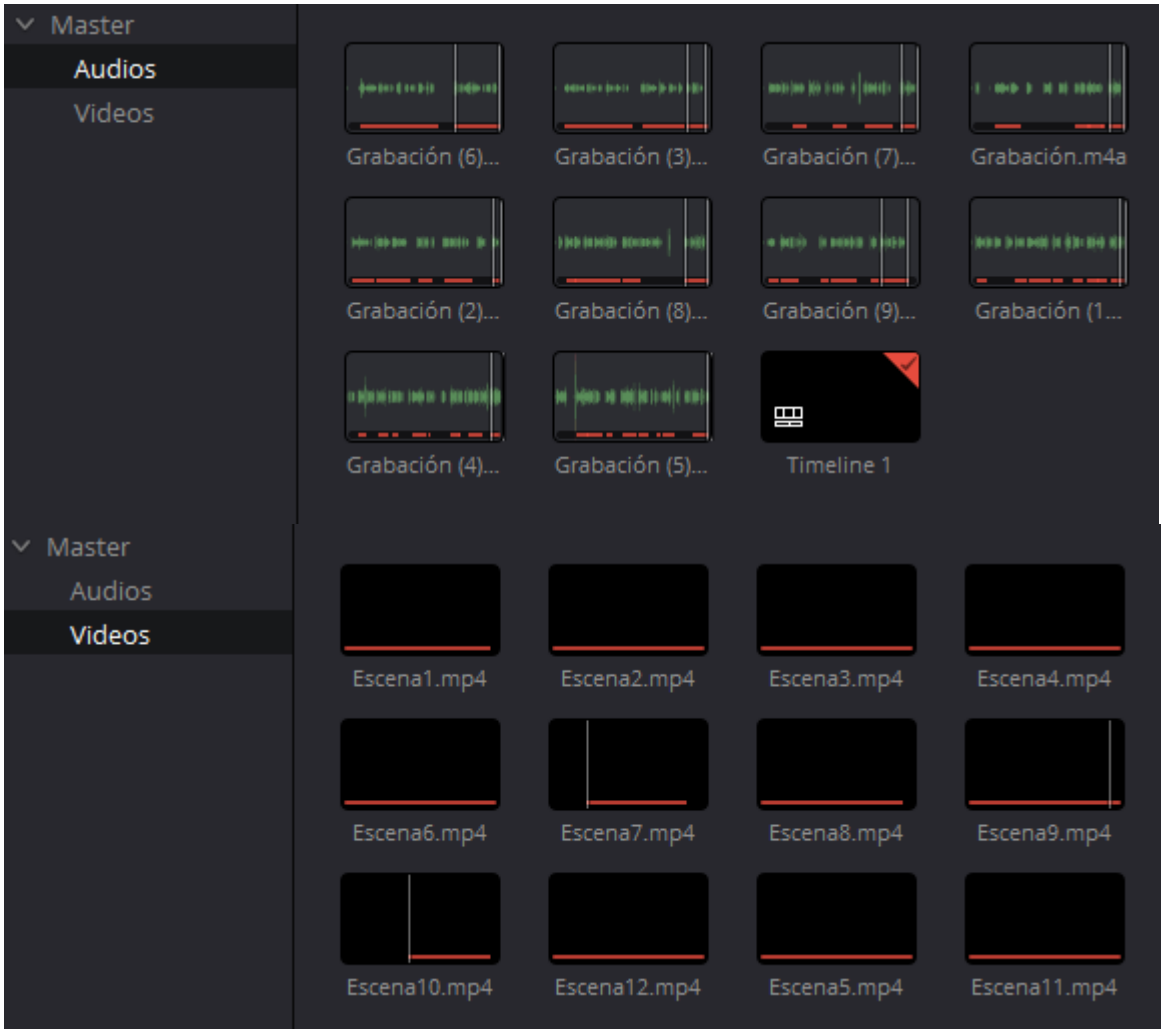

Figura 20. Organización de carpetas en DaVinci Resolve

A continuación, me desplazo a la tercera pestaña, la de edición, que es donde haré el montaje del video final.

Con el fin de ir secuenciando las pistas, voy arrastrando los archivos hasta la línea del tiempo que tengo en la parte inferior en el orden correcto. Habrá dos canales, uno para las pistas de vídeo y otro para las de audio.

Después de haber ubicado todas las escenas, voy a la última pestaña, la de renderización. Aquí le doy un nombre al fichero salida, un formato y lo añado a la cola de renderización. Tras terminar el proceso ya tengo el vídeo final en la ubicación especificada.

4.Resultados publicados

Planificación de CPU

Colas M/M/s

Paginación

5.Conclusiones y trabajos futuros

Tras realizar este proyecto, saco diversas conclusiones. En primer lugar, he comprendido la importancia de los docentes en el desarrollo personal y académico de los alumnos ya que gracias a estos es posible el progreso en cuanto a conocimientos y aptitudes del alumnado. Por este motivo, facilitar esta tarea al personal docente tiene un reflejo en la capacitación de los alumnos. Mediante `Manim` se logra esa asistencia tan útil como necesaria.

Por otro lado, gracias a herramientas como `Manim` es posible acercar a aquellos que tienen un apego al mundo de la edición, producción, al fin y al cabo, al ámbito audiovisual, pero que no tienen los suficientes recursos. Gracias a esta herramienta es posible realizar animaciones de altas calidades sin necesidad de tener unos recursos tecnológicos costosos ni unos conocimientos elevados sobre esta materia.

Tras haber visto y experimentado el proceso de creación que hay detrás de cada una de las animaciones he podido crear un juicio sobre la ayuda que pueden dotar las animaciones al docente. Debido a su forma tan sencilla de utilizar, los docentes pueden asistir, en gran medida, al aprendizaje del alumno ya que las animaciones son una forma muy asequible y visual de mostrar el contenido impartido por estos.

Este traspaso de conocimiento es mucho más eficaz con el apoyo de la multimedia como bien decía el estudio citado al inicio de este trabajo. Las animaciones favorecen que el alumno entienda y comprenda los conceptos que serían más complicados en caso de simplemente contarlos o escribiéndolos en una pizarra.

Además, desde un punto de vista de la sencillez, el docente puede optar por hacer uso de la primera técnica. Esto es debido a su gran polivalencia a la hora de crear primitivas y a su bajo requerimiento en conocimientos informáticos. Por otra parte, si lo que se busca es la completitud, la segunda técnica es la idónea. Esta es adecuada por basarse en simulaciones reales, de manera que es extensible a muchos más campos.

Este proyecto lo he enfocado en la materia de Sistemas Operativos, sin embargo, se podría ampliar el alcance de este trabajo. Esto es porque las animaciones se pueden aplicar a numerosos campos de la ciencia y, principalmente, en el ámbito tecnológico.

6.Bibliografía

[1] LaTeX, [Online]. Available: https://es.wikipedia.org/wiki/LaTeX

[2] Cario. [Online]. Available: https://www.cairographics.org/

[3] SoX, [Online]. Available: https://sox.sourceforge.net/

[4] Mayer, R.E., Moreno, R. Animation as an Aid to Multimedia Learning. Educational Psychology Review 14, 87–99 (2002).

[5] 3Blue1Brown, [Online]. Available: https://www.3blue1brown.com/

[6] Repositorio 3Blue1Brown, [Online]. Available: https://github.com/3b1b/manim

[7] Repositorio Comunidad, [Online]. Available: https://github.com/ManimCommunity/manim

[8] Manim online, [Online]. Available: https://try.manim.community/

[9] Manim Community, [Online]. Available: https://docs.manim.community/en/stable/

[10] Python, [Online]. Available: https://www.python.org

[11] FFmpeg, [Online]. Available: https://ffmpeg.org/download.html#build-windows

[12] MiKTeX, [Online]. Available: https://miktex.org/download

[13] Visual Studio Code, [Online]. Available: https://code.visualstudio.com

[14] Configuración Manim, [Online]. Available: https://docs.manim.community/en/stable/guides/configuration.html

[15] Teoría de colas, [Online]. Available: https://es.wikipedia.org/wiki/Teoría_de_colas

[16] Paginación, [Online]. Available: https://es.wikipedia.org/wiki/Paginación_de_memoria

[17] Funcionamiento de la paginación, [Online]. Available: https://www.profesionalreview.com/2020/06/10/como-funciona-la-paginacion-de-memoria/

[18] DaVinci [Online]. Available: https://www.blackmagicdesign.com/es/products/davinciresolve

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR