Generación de Pixel Art mediante inteligencia artificial y creación de NFTs en la blockchain de Oasis Network.

Ricardo Martin Dos Santos

Resumen - En este documento se presenta el desarrollo de una aplicación para crear pixel arts a partir de cualquier imagen bidimensional aplicando diferentes técnicas de procesamiento de imagen basadas en Inteligencia Artificial, incluyendo Computer Vision, Machine Learning y Deep Learning. Para los respectivos pixel arts haremos el proceso de convertir un archivo digital a un token no fungible (NFT) y prepararemos un entorno para que la acuñación de estos sea posible.

El primer objetivo es construir una versión inicial de esta aplicación en un entorno local para que en un futuro se pueda llegar a escalar a un entorno cloud y así crear una herramienta autogestionada de creación de avatares a través de una página web. Una vez tenemos la aplicación, mediante el uso de esta crearemos una colección de imagenes pixel art, y con estas imágenes crearemos NFTs en la blockchain de Oasis Network utilizando el Emerald ParaTime para que así podamos tener cada NFT en una dirección Ethereum-compatible. Cuando tengamos la colección en la blockchain crearemos una página web con el objetivo de emular la acuñación de cada NFT, en la interfaz de esta podremos enlazar una billetera de criptomonedas de tipo software como por ejemplo Metamask para interactuar con esta blockchain de una manera accesible e intuitiva.

Palabras clave— Inteligencia Artificial, Pixel Art, Procesamiento de imagen, Blockchain, NFT.

Abstract— This document presents the development of an application to create pixel arts from any two-dimensional image applying different image processing techniques based on Artificial Intelligence, including Computer Vision, Machine Learning and Deep Learning. For the respective pixel arts we will do the process of converting a digital file to a non-fungible token (NFT) and we will prepare an environment so that the minting of these is possible.

The first objective is to build an initial version of this platform in a local environment so that in the future it can be scaled to a cloud environment and thus create a self-managed tool for creating avatars through a web page. Once we have the platform, by using it we will create a collection of pixel art images, and with these images we will create NFTs on the Oasis Network blockchain using the Emerald ParaTime so that we can have each NFT in an Ethereum-compatible address. When we have the collection in the blockchain we will create a web page with the aim of emulating the mint of each NFT, in its interface we will be able to link a software-type cryptocurrency wallet such as Metamask to interact with this blockchain in an accessible and intuitive way.

Index Terms— Artificial Intelligence, Pixel Art, Image Processing, Blockchain, NFT.

1 Introducción - Contexto del trabajo

Actualmente vivimos en una era en la que tanto la computación en general cómo la inteligencia artificial en específico está en constante crecimiento, podemos ver cómo esta es capaz de crear arte como nunca se había hecho antes. También vemos como en los últimos años se han desarrollado tecnologías cómo la blockchain que nos sirven para tener un registro seguro, descentralizado, sincronizado y distribuido de operaciones digitales sin necesidad de la intermediación de terceras entidades financieras. Viendo el aumento radical del interés en estos tres campos no es difícil ver grandes cantidades de proyectos que tratan de fusionarlos.

En este trabajo de final de grado trataremos de generar arte utilizando diferentes técnicas de inteligencia artificial y algoritmos de image processing y alojar todo ese arte en la blockchain de Oasis Network [1] en forma de token no fungible (NFT). Este último punto del proyecto es altamente útil ya que cómo sabemos, en la actualidad el Reglamento general de protección de datos (RGPD) es muy estricto, por eso preservar la privacidad de los datos es un problema cada vez más común para las empresas y están empezando a utilizar la tecnología blockchain para conseguirlo de la manera más óptima y eficiente. Concretamente elegimos alojar todos los datos que generemos en la Oasis Network ya que está habilitada para preservar la privacidad mediante el uso de un nuevo tipo de activo digital llamado Tokenized Data [2], es altamente escalable, muy versátil y nos permite hacer transacciones seguras, rápidas y con comisiones mínimas.

Además en esta blockchain podemos utilizar el Emerald ParaTime [3], un entorno de contrato inteligente (smart contract) que nos permite que podamos tener cada NFT en una dirección Ethereum-compatible. Esto es perfecto ya que juntamos todas las ventajas de la red de Oasis y a la vez nuestro activo es compatible con la blockchain más utilizada y con mayor popularidad en lo que a NFTs se refiere.

- *E-mail de contacto: ricardo.martind@autonoma.cat*
- *Trabajo tutorizado por: Francesc Xavier Roca Marva (Departamento de Ciencias de la Computación)*
- *Curso 2021/22*

Julio de 2022, Escola d'Enginyeria (UAB)

2 Estado del arte

En referencia al estado del arte es difícil encontrar aplicaciones muy parecidas ya que estamos ante algo muy innovador y bastante específico. En relación a la herramienta de segmentación nos inspiramos en un código de una herramienta de anotación de imágenes gráficas a partir de polígonos [4]. Para preparar cada elemento de los prototipos de avatares pixel art nos basamos en una colección [5] de un diseñador gráfico y en general en cientos de diseños diferentes. Para el apartado de la inteligencia artificial tenemos diferentes datasets a gran escala como puede ser COCO (Common Objects in Context) [6] con más de doscientas mil imágenes etiquetadas que nos servirá para hacer un modelo de aprendizaje supervisado para detectar personas. También se han estudiado diferentes datasets para la detección de ropa solo encontrando uno que se ajuste a nuestras especificaciones [7] con un total de ochocientos una mil prendas de ropa respectivamente etiquetadas, a parte de este no se ha encontrado ninguno lo suficientemente grande y con las etiquetas puestas óptimamente. En relación a la página web que queremos crear será un diseño sencillo e intuitivo [8] para que el proceso de adquirir el NFT sea lo más ameno posible.
Para enlazar un botón a la página web veremos un tutorial en udemy dónde nos proporciona una opción para hacerlo usando diferentes herramientas [9].

3 Objetivos

A continuación planteamos los distintos objetivos que se proponen a la hora de realizar el proyecto:

1. Creación de una aplicación que permita la segmentación y anotación de imágenes con el objetivo de poder segmentar y etiquetar manualmente distintos componentes de la imagen.
2. Creación del pixel art gracias a cada segmentación hecha previamente y la aplicación de las distintas técnicas de procesamiento de imagen sobre estas.
3. Entrenar e inferir sobre modelos de inteligencia artificial para la detección de personas y los elementos que nos interesa segmentar de la imagen.
4. Fusionar la aplicación de segmentación y anotación con la inteligencia artificial para visualizar el resultado de las segmentaciones de esta y poder editarlo manualmente.
5. Acuñar todos los archivos digitales (imágenes) en la blockchain de Oasis Network utilizando el Emerald ParaTime para así crear los NFT (token no fungible).
6. Creación de una página web con su respectiva interfaz en la que podamos enlazar una billetera de criptomonedas de software como por ejemplo Metamask para así poder acuñar estos NFT en el momento en el que se realiza la transacción.

La finalidad del proyecto es conseguir cumplir todos estos objetivos con el fin de luego poder escalar el sistema completo a un entorno remoto. El usuario final podrá entrar a la web, cargar una imagen, inferir en los modelos de IA que pueden estar cargados en una máquina remota, modificar los resultados de con la aplicación de segmentación y obtener el resultado de la imagen final (pixel art) con la que se podrá acuñar el NFT relacionado con en esta misma imagen.

4 Metodología y planificación

En esta sección hablaremos de la metodología y planificación que hemos seguido para realizar este proyecto.
Se ha seguido un modelo en el que se ha dividido el proyecto en 8 fases:

- **Fase de diseño general:** primera fase donde analizamos todos los requisitos de la aplicación, el tipo de pixel art que queremos generar, y qué técnicas hemos de utilizar para generarlos.
- **Fase de implementación del segmentador y la creación de avatares:** programamos la aplicación para segmentar los elementos en Python, el código para procesar cada elemento individualmente, la clase que gestiona cada avatar y la que gestiona el fondo de la imagen.
- **Fase de diseño de plantillas:** diseñaremos las plantillas usadas para la formación del avatar, así como diferentes tipos de objetos.
- **Fase de implementación de la IA:** implementaremos la inteligencia artificial para la detección y segmentación de personas y regiones de interés de la imagen como pueden ser las prendas de ropa.
- **Fase de pruebas:** cuarta fase para realizar las pruebas convenientes para los posibles distintos escenarios.

- **Fase de acuñación en blockchain:** quinta fase donde acuñarán cada uno de los archivos digitales creados en la blockchain de Oasis Network.
- **Fase de construcción de página web:** sexta fase para la creación de la página web con su respectiva interfaz en la que podamos enlazar una billetera de criptomonedas de software.
- **Fase de cierre:** séptima y última fase para repasar la memoria hecha hasta el momento y realizar la presentación del proyecto.

5 Segmentador interactivo

Empezamos a definir la aplicación para segmentar y anotar imágenes mediante el lenguaje de programación Python [10], concretamente la versión de Python 3.8 a partir de los siguientes requisitos:

Se debe de poder cargar cualquier imagen y poder segmentar y etiquetar tantos elementos como queramos, asociar a esta segmentación una etiqueta con el identificador del avatar a la que pertenecen y otra etiqueta con el nombre del objeto que se ha segmentado.
Para segmentar utilizaremos polígonos, es decir, a partir de nodos y aristas crearemos un polígono que siempre será cerrado. También necesitamos una interfaz para modificar o eliminar tanto las segmentaciones cómo sus etiquetas. Para hacer esto modificaremos un segmentador básico [4] para adaptarlo a nuestras necesidades. Usaremos decenas de librerías pero la más importante es PYQT5 [11] para la gestión de la interfaz gráfica. Primero hacemos diferentes tipos de widgets.
Un widget que nos permita gestionar la visualización de las etiquetas de cada segmentación y modificarlas de forma simple e intuitiva [FIG 1].

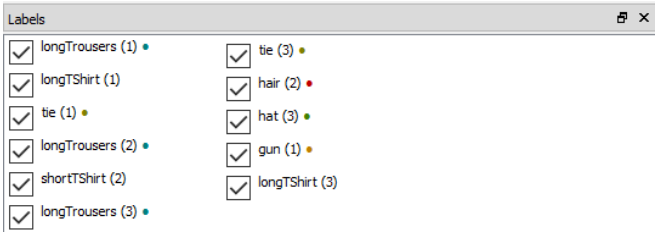


Fig. 1. Widget para visualizar las segmentaciones y sus detalles.

Otro widget para especificar, modificar y visualizar diferentes características del avatar .Unas de estas características son el género y el color de piel utilizando un widget de la librería para escoger el color en una paleta RGB. También implementamos la segmentación del pelo, añadir accesorios sin necesidad de segmentarlos y segmentar las prendas de ropa.

Para acabar hacemos que todas las segmentaciones y anotaciones de una imagen como el género, el color de piel, las etiquetas de ropa, accesorios etc. se guarden en un fichero JSON (JavaScript Object Notation) con una estructura definida que será inmutable . Para que a la hora de cargar el fichero el sistema sepa dónde se encuentra cada segmentación y la pueda representar se guardan las coordenadas de todos los puntos de cada segmentación.
A la vez, implementamos que este fichero JSON también pueda ser cargado en la aplicación y a la vez que se cargue la imagen automáticamente, para esto dentro de la estructura del mismo incluimos la ruta de la imagen. Esto nos servirá para luego a la hora de implementar la inteligencia artificial ya saber en qué estructura de datos guardar los resultados de las inferencias de los modelos, así podremos visualizar los resultados de una manera rápida fácil e intuitiva y a la vez modificar estos resultados de la forma que deseemos.

6 Inteligencia artificial

Se han implementado 4 modelos diferentes de inteligencia artificial, cada uno para segmentar diferentes partes necesarias para formar el avatar.

El primer modelo con el que se infiere a partir de una imagen es el modelo que detecta personas y ciertos animales entrenado con el dataset COCO [12], esto nos sirve para detectar cuántas personas hay en la imagen y dónde están ubicadas estas personas en la imagen, también lo configuramos para que nos segmente perros y gatos. Este modelo nos servirá para saber a qué persona pertenecen todos los elementos detectados en las siguientes inferencias de los otros modelos. En resumen, con este modelo buscamos las ROI (regiones de interés) y creamos unas bounding boxes (rectángulos imaginarios que nos sirven para delimitar el perímetro de cada persona en la imagen) que usaremos más tarde.

El segundo modelo que se infiere es el modelo para segmentar piezas de ropa en la imagen entrenado con el dataset DeepFashion2 [13].
En este modelo nos damos cuenta de que tenemos una tarea más complicada de la que pensábamos, el proceso en sí de inferir diferentes imágenes con diferentes modelos individualmente es sencillo, lo complicado es el relacionar todos los resultados de todos los modelos que no es para nada trivial. Es decir, al inferir por separado en cada modelo tenemos que relacionar a qué persona pertenece cada elemento resultado que nos devuelve el modelo, un ejemplo para entender el concepto sería que nuestro segundo modelo es capaz de detectar y segmentar una prenda de ropa, pero no es capaz de decirnos a qué persona pertenece esta prenda de ropa, esto pasa en todos los resultados de los modelos, por lo tanto hemos de tener un mecanismo que nos indique a quién pertenece cada segmentación de los diferentes modelos.
En este mecanismo poligonizamos cada máscara generada por los modelos con una librería llamada shapely [14] para así luego poder comparar el área de cada mascara de cada modelo con el primer modelo, por

ejemplo si tenemos una detección de una camiseta en la imagen, usaremos su máscara para hacer un polígono en la imagen input y como anteriormente también hemos generado la bounding box (polígono en forma rectangular) para todas las personas de la imagen lo que hacemos a continuación es comparar el porcentaje de área de la prenda que está dentro de cada polígono de cada persona, una vez hecho esto sabremos que la prenda pertenece a la persona donde la variable del porcentaje es mayor. Con esto ya tenemos la forma de relacionar cada resultado de la inferencia de los modelos con la persona de la imagen a la que pertenecen.
También se detectó un problema relacionado con este segundo modelo, cómo el modelo no tiene en cuenta la persona a la que pertenece la prenda ni las áreas de sus detecciones, puede ocurrir que detecte una prenda de ropa dos veces en una misma persona, de hecho es algo bastante frecuente. Para arreglar esto lo que hacemos es, una vez tenemos asignadas todas las prendas que pertenecen a una misma persona vemos si se repiten el tipo de prenda, comparamos sus scores (parámetro que tiene cada detección que nos dice cuánto de acertado el modelo cree que ha estado en su detección) y nos quedamos únicamente con la prenda que mayor score tiene. En este caso separamos las prendas en prendas superiores y prendas inferiores, es decir, al final de la comparación únicamente nos quedaremos con una prenda superior (p. ej. camiseta) y una prenda inferior (p. ej. pantalón).
A la hora de visualizar los resultados nos damos cuenta de un tercer error, cuándo visualizamos los polígonos en la aplicación las coordenadas no están en orden, esto provoca que ciertos polígonos se representen de una forma errónea , para arreglar esto lo que hacemos es implementar un convex hull, es decir, enlazamos todos los puntos del polígono dónde la intersección de todos los conjuntos convexos contienen un subconjunto dado de un espacio euclidiano [FIG 2].

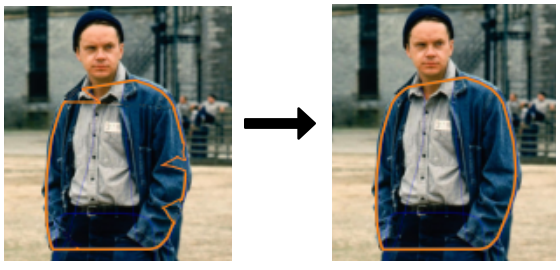


Fig. 2. Ilustración de aplicar un convex hull a un polígono.

Con el tercer modelo llamado DeepFace [15] segmentamos la cara de cada persona de la imagen , este modelo nos sirve para guardar la imagen en memoria con el fin de usarla como input a la hora de inferir con el cuarto modelo, el que nos segmentará el pelo de cada persona [16]. Actualmente no está pensado el utilizar la cara de la persona para formar la cara del avatar ya que la cantidad de pixeles de la imagen del avatar no nos permite dibujar tantas características cómo para representar una cara a partir de una imagen real, de todos modos puede que en un futuro nos sea útil.
Para finalizar con este apartado, debemos crear cómo output únicamente un fichero con la misma estructura que acepta nuestra aplicación de segmentación interactiva a la hora de cargar, como se ha dicho antes este fichero será de tipo JSON que contendrá todos los campos necesarios en un orden ya determinado para visualizar los resultados, también aprovechamos que ya hemos poligonizado anteriormente las segmentaciones de la imagen, por lo que podemos extraer las coordenadas de todos los polígonos y escribirlos en este fichero, a partir de ahora lo llamaremos fichero de features (características).

Cabe destacar que para mantener un orden en el código e inferir más rápidamente en el conjunto de imágenes lo que hacemos es gestionar varios scripts individualmente, (ver Apéndice A.1, carpeta code/AI). Tenemos un script con las funciones necesarias para generar el fichero de features (functionsCreateJson.py) y otro para generarlo (createJson.py).

Para cargar los modelos de IA utilizamos un script llamado 'modelConfigAndLoad.py' y inferimos en estos modelos que con el script 'inference.py', con esto evitamos tener que cargar los 4 modelos cada vez que hacemos la inferencia de una imagen.

La conclusión de este apartado es que nos sirve como ayuda a la hora de automatizar y así ser más rápidos a la hora de crear un avatar, en general los resultados de las detecciones y segmentaciones de los modelos son bastantes buenas y de todos modos tenemos la aplicación para visualizar y modificar estas detecciones y segmentaciones si vemos que en alguna ocasión la IA lo ha hecho erróneamente.

7 Construcción de la imagen final

Para gestionar la construcción del avatar crearemos tres clases distintas, una clase para procesar la información del fichero features creado anteriormente por la IA y modificado con la herramienta de segmentación, otra clase que recibirá como input los diccionarios de cada avatar de la imagen y sus respectivas segmentaciones con las etiquetas a la que pertenecen con el fin de construir el avatar y una tercera clase para gestionar el fondo de la imagen, la posición de cada avatar y la medida de estos.

7.1 Procesamiento de la información de entrada

Inicializamos esta clase llamada ProcessFeatures con un atributo, el nombre sin extensión de el archivo features que utilicemos para crear el avatar. Tanto el nombre sin extensión de este fichero cómo el nombre de la imagen sin extensión será el mismo.

Para empezar creamos un método para cargar la imagen, otro para cargar el fichero features y otro más para calcular el tanto el número de avatares como el número de animales detectados en la imagen. El último método y más importante se encargará de crear un diccionario para cada avatar, en este diccionario tendremos las características del avatar como por ejemplo el género o el color de piel, los accesorios y también tendremos las segmentaciones que hemos hecho anteriormente, dónde las llaves del diccionario serán el nombre de las etiquetas y los valores de cada clave será la imagen segmentada guardada como un objeto.
Para conseguir esta imagen segmentada debemos utilizar las coordenadas de cada segmentación que previamente han sido guardados en el fichero features y crear de nuevo un polígono por cada segmentación en la imagen, a partir de este polígono creamos una máscara, dónde todo lo que esté dentro de nuestro polígono será la imagen original y el fondo será de color nero, despúes de hacer esto nos quedamos con los cuatro puntos con las coordenadas más pequeñas y más grandes de los dos ejes de coordenadas x e y, así la imagen que nos guardaremos será la de el objeto [FIG 3].

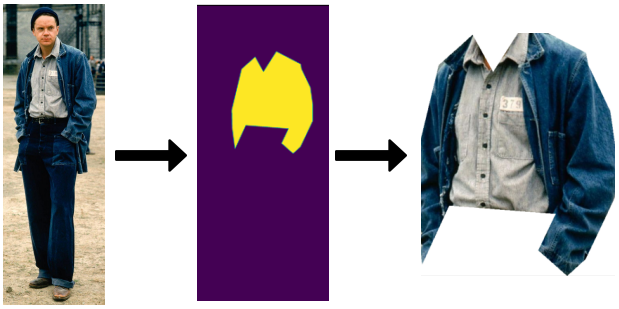


Fig. 3. Ilustración de la imagen final del objeto o prenda.

En esta clase también crearemos un diccionario con todos los animales que hayamos segmentado anteriormente.

7.2 Construcción del avatar

Inicializamos esta clase llamada CreateAvatar con dos atributos, el diccionario de características y segmentaciones que hemos construido en la anterior clase ProcessFeatures y la ruta del directorio raíz de plantillas o templates que utilizaremos para crear el avatar.

Estos avatares los crearemos haciendo uso de plantillas creadas anteriormente con cualquier programa de edición que permita crear y editar imágenes en formato PNG de 40x40 píxeles, creamos estas plantillas de una forma que luego podamos distinguir entre ciertos píxeles que queramos, es decir, para diseñarlas usamos pixeles totalmente negros (valor 0 en los 3 canales RGB), pixeles totalmente blancos (valor 255 en los 3 canales RGB) y cualquier otro color. Esto lo hacemos porque hay algunas plantillas en las que nos interesa que al traspasar las segmentaciones a la plantilla, en ciertos píxeles se pueda crear un efecto de sombra o un efecto de iluminación. Para crear estos efectos lo que haremos será crear condiciones dentro de los métodos que se dedican a traspasar los colores de la segmentación a la plantilla, por ejemplo, la condición para los pixeles negros es que cuando en los píxeles de una plantilla sus 3 canales RGB sean 0 y su canal A (alpha) sea 255 traspasaremos estos mismos pixeles de la segmentación pero a la vez oscureceremos cada pixel, esto lo hacemos multiplicando cada uno de ellos por el porcentaje que le especifiquemos, en el caso actual un 65% (0.65). En el caso de los píxeles blancos para crear iluminación creamos la condición de que cuando los tres canales RGB sean 255 y el canal A sea 255 traspasaremos los píxeles de la segmentación pero a la vez iluminaremos cada pixel aplicando un threshold y sumándoselo a los 3 canales RGB, en este caso el valor del threshold es de 130.
Si el color del pixel no es ni blanco ni negro el traspaso de color de la segmentación a la plantilla será del mismo color que tenga la segmentación.

Estas plantillas están estructuradas con un sistema de directorios previamente definido e inmutable, este directorio se llama templates (ver Apéndice A.1).
Empezamos diseñando lo que llamamos plantillas base, estas plantillas són las que siempre aparecerán cuando se crea un avatar en la misma posición y que no dependen del diccionario de características y segmentaciones, por ejemplo serían las de la cara, ojos y nariz.

Creamos un método para gestionar todo lo relacionado con este tipo de plantillas, las plantillas del cuerpo y la cara siempre serán las mismas por defecto. La de los ojos estarán siempre en la misma posición pero serán de colores aleatorios en cada ejecución, para la de la nariz también estará en la misma posición y lo que haremos será coger el color de piel del diccionario de características y segmentaciones y oscurecerlo a partir de un cierto valor que nos sirve como threshold.
Para las plantillas en las que necesitamos que se le traspase el color de las imágenes segmentadas hacemos un método llamado transferColor, los parámetros que pasamos a este método són la imagen de la segmentación y la imagen de la plantilla, ambas en formato array de NumPy [17]. Lo primero que haremos es llamar a un método que nos retorna cuanto mide la plantilla únicamente contando los píxeles no transparentes, es decir, busca la primera y última fila y columna en la cual hay un pixel pintado, con esto sabemos cuanto mide el objeto dibujado en la plantilla.

A continuación llamamos a un método que nos devuelva un array con las coordenadas de todos los píxeles no transparentes de una imagen, en este caso de la plantilla, esto nos servirá más tarde para saber a qué pixeles hemos de traspasar el color únicamente. A continuación llamamos a los métodos que nos hará el redimensionamiento (resize) de las imágenes segmentadas a las medidas de la plantilla que hemos calculado en el primer método y nos pintará los píxeles que són transparentes porque no coinciden con la segmentación, esto lo hacemos llamando a un método que llamamos expandSegmentation el cual asigna al pixel

transparente el color de su vecino más cercano, con esto nos aseguramos asegurarnos que modificaremos todos los píxeles de la plantilla y no dejaremos ninguno vacío [FIG 4].

Fig. 4. Redimensionamiento de la segmentación.

Después de esto utilizaremos el array que hemos generado anteriormente en el segundo método que hemos llamado para recorrer todos los píxeles de la plantilla,
dentro de este bucle lo que haremos es traspasar el color de cada píxel de la imagen segmentada a la misma coordenada de la plantilla. A la vez tenemos las dos condiciones comentadas anteriormente, una para saber si el pixel es completamente negro y así traspasar el color y oscurecerlo y otra para saber si el pixel es completamente blanco y así traspasar el color y iluminarlo [FIG 5].

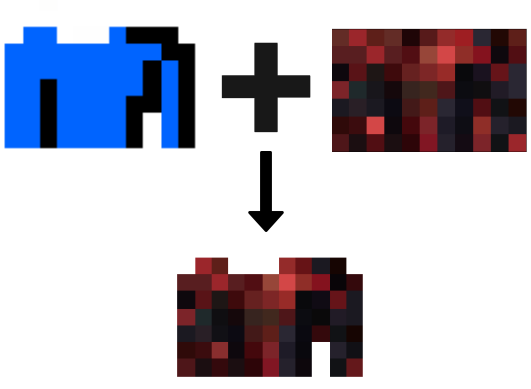


Fig. 5. Traslado de pixeles a la plantilla

También creamos una serie de métodos para crear patrones aleatorios con colores aleatorios para el caso en el que no existiera la segmentación de una plantilla en el diccionario de características y segmentaciones, esto nos permite que no haya la posibilidad en la que un avatar se cree sin una prenda que debería de tener[FIG 6].

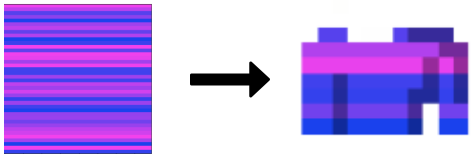


Fig. 6. Patrón de plantilla generado aleatoriamente.

También creamos un método para pseudo-aleatorizar la generación de estos patrones y colores, este método escoge un color aleatorio de una lista previamente definida y para cada pixel que va a traspasar cambia los valores de los 3 canales RGB en un rango que está entre dos valores, uno negativo y otro positivo, en este caso hemos predefinido un rango de entre [-30, 30], este método lo utilizamos para traspasar el color a el calzado ya que este no lo segmentamos y por lo tanto no estará en el diccionario de características y segmentaciones.

Para escoger las diferentes plantillas también creamos diferentes métodos dependiendo del nivel en la estructura de carpetas en el que nos encontremos.
Para las plantillas de las partes del cuerpo utilizaremos el método chooseTemplateBody, pasaremos como parámetro la ruta del directorio de las partes del cuerpo específicas y de forma opcional un array de probabilidades para controlar la aleatoriedad de la elección de la plantilla, es decir, le podemos decir el porcentaje de probabilidad de coger cada plantilla de la carpeta. Lo que nos retornará será la imagen en formato array de la parte del cuerpo y el número de la plantilla que ha escogido (este número se especifica en el nombre de la plantilla), este número nos servirá para a la hora de escoger la prenda para esa determinada parte del cuerpo saber con qué plantilla se ha empezado a construir el avatar.
También creamos un método para elegir una plantilla de forma totalmente aleatoria dada una ruta a la carpeta dónde se encuentran esas plantillas, otro que recibe como parámetro una etiqueta y que comprueba si esta etiqueta existe o no en el diccionario de características y segmentaciones para así saber si añadirla, por ejemplo para ver si existe la segmentación de el pelo o el avatar no tendrá pelo. A la vez este método nos sirve para comprobar si existen accesorios en el este mismo diccionario, esto lo hacemos de esta forma ya que en el código hard-codeamos los nombre de los accesorios para los que que hemos generado plantillas y llamamos a la función para comprobar si ese avatar debe aparecer con ellos.
A la vez tenemos otro método para elegir la plantilla de la ropa, necesitamos crearlo ya que debemos pasar como parámetro el número de plantilla que ha elegido anteriormente el método chooseTemplateBody, así nos aseguraremos de cual es la carpeta que debemos entrar, una vez nos encontremos en esa carpeta elegiremos la plantilla de forma aleatoria.

El hecho de escoger aleatoriamente plantillas como hacemos con las piernas, brazos, ciertos accesorios etc. y cambiar de colores ciertas plantillas aleatoriamente nos sirve para conseguir más diversidad entre los avatares generados. Cada vez que aplicamos estos métodos y elegimos las diferentes plantillas y les hacemos las transformaciones que les pertoquen para cada parte del avatar, lo que hacemos es utilizar una función para pegar (paste) una imagen sobre otra, por lo tanto también tenemos que tener en cuenta el orden en el que se pegan las partes del avatar.

Todas las plantillas se pegan en una imagen base (imagen totalmente transparente de 40x40 píxeles).
Siempre pegamos en esta imagen base las partes del avatar que ya han sido previamente modificadas llamando a los diferentes métodos explicados anteriormente.
Para empezar pegamos las plantillas base (cara, ojos y nariz), a continuación la parte inferior del tronco del cuerpo, es decir, las piernas, la prenda de ropa inferior, el calzado y finalmente los accesorios (p. ej. cinturón). Para finalizar pegamos la parte superior del tronco del cuerpo, que serían los brazos, la prenda de ropa superior, el pelo y los accesorios de esta parte del cuerpo (p. ej. corbata, reloj, gafas, espada, arma, bastón, sombrero, gorra, etc).

7.3 Construcción del fondo

Inicializamos esta clase llamada Background con tres atributos, una lista con el total de avatares que aparecerán en la imagen (máximo 2), el diccionario de animales que hemos generado en la clase ProcessFeature y el nombre de la imagen input, este nombre nos servirá para guardar los outputs con ese nombre.

Esta clase nos permite crear el fondo de la imagen y a la vez gestionar la posición de cada avatar y la medida de estos así como añadir los animales que hayamos segmentado. Para generar el fondo de la imagen tenemos unos métodos con los que generamos diferentes tipos de fondos a partir de colores, pueden ser fondos de un solo color o bien gradientes. Con un método encontramos los colores dominantes y complementarios de los avatares, esto fue implementado ya que al superponer un color dominante a uno complementario llama la atención al ojo humano. Así podríamos generar el gradiente por ejemplo de los dos colores complementarios de los dos avatares .
También nos permite generar el gradiente con dos colores aleatorios o pseudoaleatorios. También creamos un método para que dada una probabilidad añadiese distintas plantillas de diferentes elementos de forma aleatoria.

Cabe destacar que la resolución de la imagen final será de 400 x 400 píxeles, por lo tanto la imagen que hemos generado del fondo también tendrá esa resolución. Esto es importante porque a la hora de pegar las imágenes de cada avatar a la imagen del fondo, estas ya deben de estar con la misma resolución. Aprovechamos este redimensionamiento que debemos hacer de 40x40 píxeles a 400 x 400 píxeles para tener la libertad de cambiar el tamaño del avatar, como queremos que este redimensionamiento sea aleatorio hemos de tener un rango definido para que no se convierta en un avatar demasiado pequeño o demasiado grande.

Respecto a pegar el avatar en la nueva imagen fondo tenemos dos posibilidades.
En el caso de que haya un avatar solamente en la imagen lo que hacemos es cambiarle el tamaño (redimensionar) aleatoriamente dentro de un rango definido entre 750 y 850 píxeles tanto de altura como de anchura, y cambiarle la posición también en un rango definido para evitar que se salga de la imagen.
Para construir esta imagen final con dos avatares también debemos tener un mecanismo para gestionar la posición de cada avatar dentro de la imagen, para así evitar que se toquen o que se salgan de la imagen. Este mecanismo se basa en que dependiendo de cuál de los dos avatares sea se moverá a un lado u a otro a partir del centro de la imagen.
Primero redimensionamos cada avatar aleatoriamente dentro de un rango definido como hacemos en el caso de que solo haya un avatar pero ahora será con números menores ya que nos conviene que los avatares sean más pequeños, el rango será de entre 710 y 750 píxeles tanto de altura como de anchura.
A la vez debemos saber las coordenadas x e y del primer y último píxel de cada avatar, así evitaremos que se salgan de la imagen. Una vez tenemos estas coordenadas ya sabemos hasta dónde podemos mover cada avatar y a partir de un rango establecido podemos hacer que estos avatares se muevan en el eje x y en el eje y aleatoriamente sin que corran el riesgo de salirse de la imagen o tocarse entre sí.

Finalmente comprobamos si existe algún elemento dentro del diccionario de animales con el que llamamos a la clase y añadimos la plantilla de este animal en unas coordenadas específicas si es que existe.

7.4 Construcción de GIFs

Mientras se escribía el código para generar estos avatares, se vió una buena opción el crear archivos GIF para poder visualizar como se ha construido el avatar.
Para cada imagen final hemos creado un gif de como se construye pixel a pixel y se han subido a una web para poder visualizarlos fácilmente [18], cómo cambiamos los píxeles uno a uno dentro de un bucle y generamos el fondo también pixel a pixel esto nos permite que en cada cambio de pixel llamemos a una función de PIL para generar una imagen en cada cambio de píxel y así al final poder generar un GIF de cientos de imágenes con todo el proceso. De hecho el NFT que crearemos podría ser este GIF en vez de la imagen, solo cambiaría el hecho de que en vez de subir una imagen con extensión PNG subiremos un archivo con extensión GIF.

8 Creación de NFTs

Empezamos definiendo el tipo de token y la blockchain donde queremos alojar estos tokens. La blockchain será la Oasis Network, una blockchain descentralizada de Capa 1 diseñada para ser altamente escalable, versátil y conocida por preservar la privacidad de los datos. Una de las ventajas de esta blockchain es que tiene un ParaTime llamado Emerald, este nos proporciona un entorno de contrato inteligente con compatibilidad total con EVM

(Ethereum VIrtual Machine) [19]. Esto nos permite ejecutar contratos inteligentes de ethereum en la blockchain de Oasis con comisiones 99% más bajas que Ethereum. Una vez tenemos claro dónde queremos alojar nuestros NFTs decidimos que el tipo de token sería el ERC-721, este token se ha creado para la red Ethereum bajo los estándares de sus contratos inteligentes. Este estándar fue diseñado con el objetivo de crear tokens intercambiables pero con la particularidad de ser únicos y no fungibles. Es decir, cada token es único en toda su existencia y no puede deteriorarse o destruirse. El objetivo tras esto, es desarrollar tokens únicos.

Una vez sepamos esto y tengamos las creadas todas las imágenes en una misma carpeta nos iremos a Pinata [20]. Pinata es un servicio web que nos permite cargar y administrar archivos en IPFS (InterPlanetary File System), la red peer-to-peer para almacenar y compartir datos en un sistema de archivos distribuido que usaremos para crear estos NFTs. Nos identificamos en la web de Pinata y subimos esta carpeta, una vez subamos esta carpeta a Pinata se subirá automáticamente a IPFS, ahora lo que necesitamos es el CID (identificador de contenido) de esta carpeta, que nos sirve para ubicar y identificar nuestra carpeta en la red IPFS ya que es único. Ahora haremos los metadatos de cada NFT, estos metadatos son un archivo JSON con todos los valores y atributos que pertenecen al NFT y que no se almacenan en la cadena de bloques. Para hacer los metadatos nos basamos en la estructura estándar de metadatos para el token ERC-721 [21] , en cada fichero JSON de metadatos guardaremos el nombre que le queramos dar al NFT, una descripción, la ruta de la imagen en IPFS y unos atributos, en nuestro caso dejaremos el campo atributos vacío.
Una vez tengamos todos los ficheros JSON creados los guardamos en la carpeta y los subimos a Pinata. Ahora cogeremos el CID que se ha creado para esta carpeta con los metadatos de cada NFT.

Entramos a Ethereum Remix, un entorno de desarrollo de Ethereum que nos ofrece la posibilidad de compilar y desplegar contratos inteligentes. Escribiremos nuestro contrato inteligente usando el lenguaje de programación Solidity [22] con la versión 0.8.2 y lo compilamos con la versión 0.8.14+commit.80d49f37 (ver Apéndice A.2).
Para el contrato inteligente implementamos el código necesario, en el constructor especificamos el nombre de la colección y el símbolo, después creamos una función que con el nombre mint que nos servirá para especificar que NFT se acuñará en cada llamada al contrato, para eso tenemos un valor que incrementa en 1 cada vez que se llame al contrato y se cree el token en la blockchain, así nos aseguramos no se repitan para que todos sean únicos, también controlamos que la cantidad de llamadas al contrato nunca sean más de los tokens que existen, es decir, de la cantidad de imágenes que hemos subido a Pinata anteriormente. Creamos otra función que se llame tokenURI, esta es la que se encargará de gestionar a que token pertenece qué metadatos y que imágenes, en esta función tenemos que poner la ruta a los metadatos que hemos subido a Pinata anteriormente y su CID.
Una vez tengamos esto podemos compilar el contrato y desplegarlo (deploy), con el botón de mint que tenemos en Ethereum Remix podemos comprobar que el contrato inteligente funciona y que cada vez que pulsamos el botón se crea un NFT con los metadatos y imagen que deben ser.

9 Acuñación de NFTs

Una vez comprobado el correcto funcionamiento del contrato haremos la web y crearemos un botón desde donde se podrán acuñar estos NFTs.
La creación de esta web es algo sencillo gracias a plataformas online como puede ser WIX que ofrece una serie de herramientas para la creación de sitios web profesionales de forma rápida y sencilla. Lo realmente complicado es enlazar ese botón a una cartera software de criptomonedas para que cualquier persona pueda hacer la acuñación de este NFT. Para esto usaremos la cartera MetaMask [23] ya que tiene una extensión para el navegador. Para empezar a enlazar este botón a la cartera usamos bubble.io [24] y el plugin Web3 & Metamask [25], crearemos botones para controlar la cantidad de NFTs a acuñar, texto que muestra el número de la cantidad y que decrementa o incrementa el número según el botón que pulsemos y un botón para finalmente acuñar esa cantidad. Cuando tengamos todos los elementos creados controlaremos el flujo de todos estos para su correcto funcionamiento.
Una vez hecho esto usamos el plugin para enlazar MetaMask y creamos otro flujo en el botón de mint. El primer paso cuando se pulsa este botón es abrir la extensión de MetaMask y aceptar permisos para que la web pueda interactuar con la cartera si no se ha hecho anteriormente. A continuación deberemos de configurar el plugin para enlazarlo a nuestro contratos inteligente, para eso debemos especificar ciertas variables que las encontramos en Ethereum Remix una vez desplegamos el contrato, estas variables son la dirección del contrato en la blockchain, el ABI (Interfaz Binaria de Aplicación), es el modo estándar de interactuar con contratos en el ecosistema Ethereum, los datos se codifican acorde a esta especificación, la función a la que llamaremos dentro del contrato, que en nuestro caso es la función mint y le pasaremos un parámetro a esta función, el número de tokens a acuñar, que será el texto del elemento de la web dónde anteriormente hemos especificado cuántos NFTs queríamos acuñar y finalmente el valor de la transacción, en nuestro caso es 0, pero en un caso real debería de ser la cantidad de ROSEs (la criptomoneda que usa Oasis Network para sus transacciones y comisiones) multiplicado por el número de NFTs que queríamos acuñar.
Una vez hecho todo esto desplegamos un frame que acabamos de crear con todas sus funcionalidades, a continuación con el link de este nuevo frame vamos a un frame generator que nos genera el código de ese frame, con esto podemos embeber el frame en la página web que hemos creado [26] utilizando WIX y con ello todas las funcionalidades.

Al presionar el botón de mint una vez hayamos aceptado los permisos, se visualizan los detalles de la transacción y en este momento se puede elegir el rechazar o aceptar esta transacción [FIG 7], cabe destacar que en esta red el precio por transacción es prácticamente nulo (< 0,00001$).

Fig. 7. Detalles de la transacción.

Cuando aceptamos la transacción podemos visualizar todos los datos de esta misma en el explorador de Emerald junto con los metadatos e imagen que se le ha enlazado a él mismo, hacemos un ejemplo de una transacción para acuñar un NFT [27] y un ejemplo donde en una transacción podemos acuñar más de un NFT, se podrán acuñar tantos como el número de estos que tengamos en la colección, por ejemplo hacemos una transacción para acuñar 2 NFTs a la vez [28]. Todas las transacciones o interacciones así como los NFTs generados con nuestro smart contract también son registrados en el Emerald explorer por lo que tenemos una trazabilidad completa[29]. Dentro de cada transacción si hacemos click en el número del NFT podremos ver la información relacionada con este, incluida la imagen, en nuestro caso podemos ver para el primer NFT acuñado con la primera transacción [30] y los otros dos generados con la segunda transacción[31][32]. El ID nos indica que número de la colección es y en los metadatos podemos ver el nombre de este, el enlace a la imagen real, la descripción y los atributos si los hubiésemos generado.

10 Conclusiones

A modo de cierre de este trabajo, podemos verificar que realmente se han cumplido todos los objetivos que se propusieron al iniciar el proyecto. Al formular la propuesta no pude encontrar nada similar a lo que quería hacer y de hecho sigo sin encontrarlo, por esto la planificación del proyecto ha sido sumamente modificada durante las primeras entregas ya que realmente sabía lo que quería hacer pero no el cómo ni cuánto iba a tardar. A la vez, me he podido dar cuenta a lo largo de las fases del proyecto de que al no tener referencias no había contemplado la serie de dificultades que me podía encontrar. Se han planteado muchas y me he encontrado con problemas que me han sido difíciles de solucionar y me han llevado más tiempo del que tenía previsto en la planificación. El único problema que no he podido solucionar es el diseñar un algoritmo para comparar las diferentes segmentaciones con las diferentes plantillas y así ver cual es la más similar.

A la vez, hay ciertos puntos que me gustaría mejorar como por ejemplo reentrenar los modelos de IA utilizados con un dataset más amplio y así mejorar los resultados de estos. También me hubiera gustado implementar más modelos de IA, por ejemplo alguno que detecte el género de la persona segmentada.

En mi opinión ha sido un trabajo muy completo y con mucho potencial a futuro ya que las dos tecnologías principales de este (IA y blockchain) están en pleno auge y darán mucho de qué hablar durante los próximos años. Por esto creo que la mejor parte de este trabajo de fin de grado es que todo lo que he hecho me sirve como una base para empezar a mejorar y construir una plataforma con estas herramientas y desplegarlas en otros entornos.

En conclusión, hemos sido capaces de diseñar, crear y testear los distintos programas y aplicaciones que forman nuestra cadena de procesos o pipeline en un entorno local y que nos generan estos pixel art dada una imagen mediante el uso de inteligencia artificial y diferentes métodos de image processing y la subida de estos a la blockchain en forma de NFT.

11 Bibliografia

[1] The Oasis Network privacy-enabled blockchain platform for open finance and a responsible data economy. Retrieved 18 March 2022, from https://oasisprotocol.org/

[2] Oasis Network Primer. Retrieved 18 March 2022, from https://docs.oasis.dev/oasis-network-primer/

[3] Oasis Emerald — EVM ParaTime is live on Mainnet. Retrieved 18 March 2022, from https://medium.com/oasis-protocol-project/oasis-emerald-evm-paratime-is-live-on-mainnet-13afe953a4c9

[4] Image Polygonal Annotation with Python. Retrieved 18 March 2022, from https://github.com/wkentaro/labelme

[5] Daily pixel art collection . Retrieved 18 March 2022, from https://www.deviantart.com/jevi93/gallery/60742075/daily-pixel-art

[6] Large-scale object detection, segmentation, and captioning dataset. Retrieved 18 March 2022, from from https://cocodataset.org/

[7] Comprehensive fashion dataset. Retrieved 18 March 2022, from https://github.com/switchablenorms/DeepFashion2

[8] Dao Emeralds project. Retrieved 18 March 2022, from https://daoemeralds.com/#roadmap

[9] Udemy course for embedding crypto wallet functionalities. Retrieved 12 June 2022, from https://www.udemy.com/course/design-a-wix-nft-website-with-crypto-wallet-functionality/

[10] What is Python. Retrieved 10 April 2022, from https://es.wikipedia.org/wiki/Python

[11] PyQT5. Retrieved 10 April 2022, from https://pypi.org/project/PyQt5/

[12] Large-scale object detection, segmentation, and captioning dataset. Retrieved 18 March 2022, from from https://cocodataset.org/

[13] Comprehensive fashion dataset. Retrieved 18 March 2022, from https://github.com/switchablenorms/DeepFashion2

[14] Shapely library to work with polygons. Retrieved 12 June 2022, from https://shapely.readthedocs.io/

[15] Deepface project. Retrieved 10 April 2022, from https://pypi.org/project/deepface/

[16] Hair segmentation model. Retrieved 10 April 2022, from https://github.com/Papich23691/Hair-Detection

[17] Numpy library. Retrieved 10 April 2022, https://numpy.org/

[18] Our webpage with some generated GIFs of the avatars. Retrieved 12 June 2022 from https://riicardomartin17.wixsite.com/my-site/gifs

[19] Oasis EVM compatible Emerald ParaTime. Retrieved 12 June 2022 from https://docs.oasis.dev/general/developer-resources/emerald-paratime/

[20] Pinata online web. Retrieved 12 June 2022 from https://www.pinata.cloud/

[21] ERC-721 token contract and metadata standard. Retrieved 12 June 2022 from https://docs.openzeppelin.com/contracts/2.x/erc721

[22] Solidity language docs . Retrieved 12 June 2022 from https://docs.soliditylang.org/en/v0.8.14/

[23] Metamask webpage. Retrieved 12 June 2022 from https://metamask.io/

[24] Bubble.io webpage. Retrieved 12 June 2022 from https://bubble.io/

[25] Web3 & Metamask plugin features. Retrieved 12 June 2022 from https://bubble.io/plugin/web3--metamask-1612784921335x464807902025875460

[26] Our website created using wix. Retrieved 12 June 2022 from https://riicardomartin17.wixsite.com/my-site

[27] Details of the transaction for minting 1 NFT in Emerald explorer. Retrieved 27 June 2022 from https://explorer.emerald.oasis.dev/tx/0x2d6334f064ab80501b6d83c84013274f5d05b98a0737869fab37e827e79774ac/token-transfers

[28] Details of the transaction for minting 2 NFTs in Emerald explorer. Retrieved 27 June 2022 from https://explorer.emerald.oasis.dev/tx/0x7d0106c35d6ce4af4bf280fb610940ef117559b10cc3e5ba335369225e206431/token-transfers

[29] Details of all the transaction of our Smart contract in Emerald explorer. Retrieved 27 June 2022 from https://explorer.emerald.oasis.dev/address/0x13C014Cc4670d3838b10A794de7dB330B837b57c/transactions

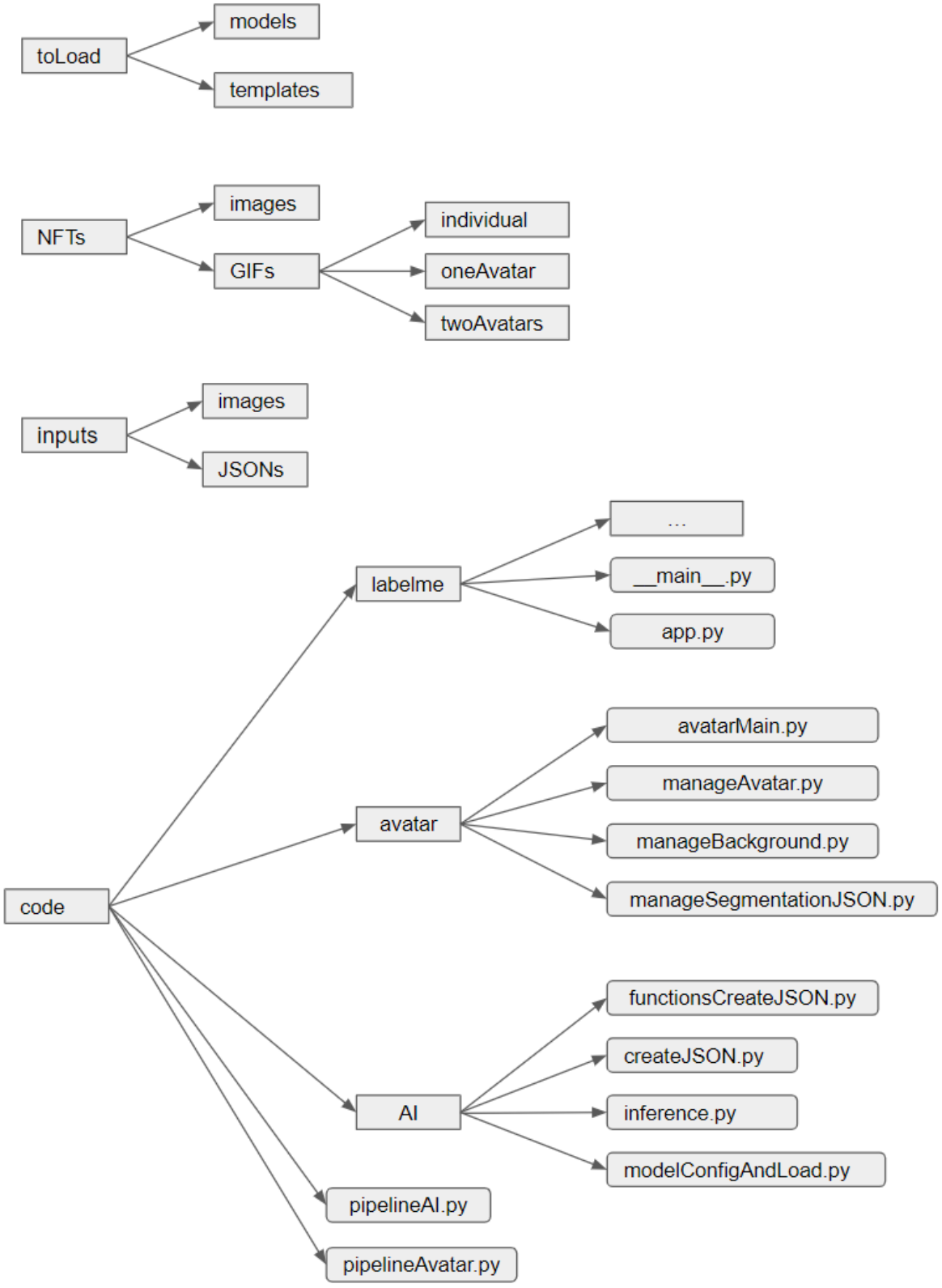
[30] Details and metadata of the NFT minted in the first transaction of our Smart contract in Emerald explorer. Retrieved 27 June 2022 from https://explorer.emerald.oasis.dev/token/0x13C014Cc4670d3838b10A794de7dB330B837b57c/instance/1/metadata

[31] Details and metadata of the second NFT minted in the second transaction of our Smart contract in Emerald explorer. Retrieved 27 June 2022 from https://explorer.emerald.oasis.dev/token/0x13C014Cc4670d3838b10A794de7dB330B837b57c/instance/2/metadata

[33] Details and metadata of the third NFT minted in the second transaction of our Smart contract in Emerald explorer. Retrieved 27 June 2022 from https://explorer.emerald.oasis.dev/token/0x13C014Cc4670d3838b10A794de7dB330B837b57c/instance/3/metadata

Apéndice

A.1 Estructura de las carpetas



A.2 Código del smart contract en solidity

```
pragma solidity ^0.8.2;

//import Open Zeppelin contracts
import
"https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/
ERC721.sol";
import
"https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Strings
.sol";

contract NFT is ERC721 {
    uint256 private _tokenIds;
    uint256 public totalMaxSupply = 30;
    constructor() ERC721("OasisTFG", "TFG") {}

//use the mint function to create an NFT
    function mint(uint256 _mintAmount)
    public
    returns (uint256)
    {
        //conditions
        //mint minimum 1 NFT
        require(_mintAmount > 0);
        //mint amount plus the number of the actual token must be lower than the total
          supply to avoid generate infinite NFTs
        require(_tokenIds + _mintAmount <= totalMaxSupply);
        //loop to generate more than one NFT per transaction
        for (uint256 i = 1; i <= _mintAmount; ++i) {
            _tokenIds += 1;
            _mint(msg.sender, _tokenIds);
        }
        return _tokenIds;
    }

//Include the CID of the JSON folder on IPFS, for every mint the NFT ID will be the next
    function tokenURI(uint256 _tokenId) override public pure returns(string memory) {
        return string(
            abi.encodePacked(
                "https://ipfs.io/ipfs/Qmcn6VKYzssaBu7whKUVz7FDn6A2N3rJ7mjjoLvcgYSNir/",
                Strings.toString(_tokenId),
                ".json"
            )
        );
    }
}
```