SEP **SES** **TNM**

INSTITUTO TECNOLÓGICO DE CHIHUAHUA II

“Creación de un Avatar digital que aprenda a mover los labios usando Aprendizaje por Refuerzo”

TESIS PARA OBTENER EL GRADO DE

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA

LEONARDO NEVÁREZ PORRAS

DIRECTOR DE TESIS
DR. HERNÁN DE LA GARZA GUTIÉRREZ

CO-DIRECTOR DE TESIS
M.S.C. CARLOS HUMBERTO RUBIO RASCÓN

CHIHUAHUA, CHIH., A JULIO 2021

Dictamen

Chihuahua, Chih., a 02 de julio de 2021

M.C. MARÍA ELENA MARTÍNEZ CASTELLANOS
COORDINADORA DE POSGRADO E INVESTIGACIÓN
PRESENTE

Por medio de este conducto el comité tutorial revisor de la tesis para obtención de grado de Maestro en Sistemas Computacionales, que lleva por nombre "CREACIÓN DE UN AVATAR DIGITAL QUE APRENDA A MOVER LOS LABIOS USANDO APRENDIZAJE POR REFUERZO", que presenta el C. LEONARDO NEVÁREZ PORRAS, hace de su conocimiento que después de ser revisado ha dictaminado la APROBACIÓN de la misma.

Sin otro particular de momento, queda de Usted.

Atentamente
La Comisión de Revisión de Tesis.

DR. HERNÁN DE LA GARZA GUTIÉRREZ
Director

M.S.C. CARLOS HUMBERTO RUBIO RASCÓN
Co-director

M.C. ARTURO LEGARDA SÁENZ
Revisor

DRA.MARISELA IVETTE CALDERA FRANCO
Revisor

CONTENIDO

INDICE DE FIGURAS

RESUMEN

Se describe la investigación previa, planeación, desarrollo, resultados y conclusiones de la creación de un sistema de aprendizaje del movimiento de los labios a partir de un audio sin haberle dado tratamiento previo, de una persona hablando en español. Como salida se genera una animación 2D a 30 cuadros por segundo de un personaje que muestra el movimiento de los labios generado a partir de la implementación de la estrategia de Aprendizaje por Refuerzo, que incluye el uso de una red neuronal convolucional profunda, entrenada con el algoritmo de Q-Learning.

ABSTRACT

This document describes the research, planning, development, results, and discussion about a learning system for lips' movements, which takes raw human speech audio in Spanish. The system creates a 2D animation at 30 frames per second of a character that shows the movements of the lips, generated by the implementation of the Reinforcement Learning strategy, that includes a convolutional neural network trained using the Q-learning algorithm.

CAPÍTULO I. INTRODUCCIÓN

1.1 Introducción.

El presente proyecto de tesis corresponde a la creación de un sistema de aprendizaje automático que tomando como entrada un audio, manipule el movimiento de labios de un Avatar digital. También se considera como parte del proyecto la creación del Avatar digital.

Para esto se propone explorar el uso de uno de los métodos del aprendizaje automático (ML por sus siglas en inglés: machine learning) para la creación del sistema que manipulará al Avatar digital.

En la actualidad, la generación de material digital es de gran importancia y utilidad, en diferentes ámbitos, como son, en la enseñanza a distancia y en la creación de contenido de entretenimiento, además la producción de video con personajes animados requiere de herramientas especializadas y de personas que sepan usarlas. Dependiendo de la naturaleza de la animación y los movimientos de los personajes se puede requerir de una refinación de dichos movimientos para que se asemejen a los movimientos naturales de las personas de manera que se ajusten al contenido (Cabiedes et al, 2007).

A partir de este tipo de retos, surgen sistemas de automatización de la animación, algunos de ellos apuntados a generar el movimiento de los labios, ya sea a partir del movimiento original del actor de voz (Suwajanakorn et al, 2017) o mediante otras técnicas como siguiendo animaciones hechas a mano (Cabiedes et al, 2007, Aneja y Li, 2019). El sistema que se describe a continuación utiliza datos generados a partir de las posiciones del mismo locutor del audio y busca generar los movimientos de labios a partir de un audio en español utilizando técnicas de Aprendizaje por Refuerzo.

Recientemente han emergido técnicas de Aprendizaje de Máquina que permiten entrenar redes neuronales profundas utilizando el algoritmo de propagación hacia atrás (LeCun et al 2015), las cuales pueden aprender a realizar la tarea encomendada a partir de datos sin procesamiento previo, tales como imágenes, video y audio (Mnih et al 2015).

El proyecto que se presenta busca aprender a imitar el movimiento de labios a partir de un video, del cual se hace la separación de las imágenes y del audio. El audio se toma como entrada al módulo de Aprendizaje por Refuerzo para obtener una posición de los labios de salida, la cual se

compara con las posiciones reales obtenidas de las imágenes del video y como resultado de dicha comparación generar la recompensa que promueva el aprendizaje. Principalmente se utiliza una Red Neuronal Convolucional (RNC), la cual describiremos a fondo más adelante para estimar una función que entrega las expectativas de recompensa de un agente al mover unos labios basado en la señal de audio. En el contexto del trabajo actual, nos referimos a un agente como la parte del sistema que percibe el estado del ambiente en el momento y toma acciones. En la figura 1.1, se pueden apreciar estos elementos, más el intérprete, que juntos forman parte del aprendizaje por refuerzo.

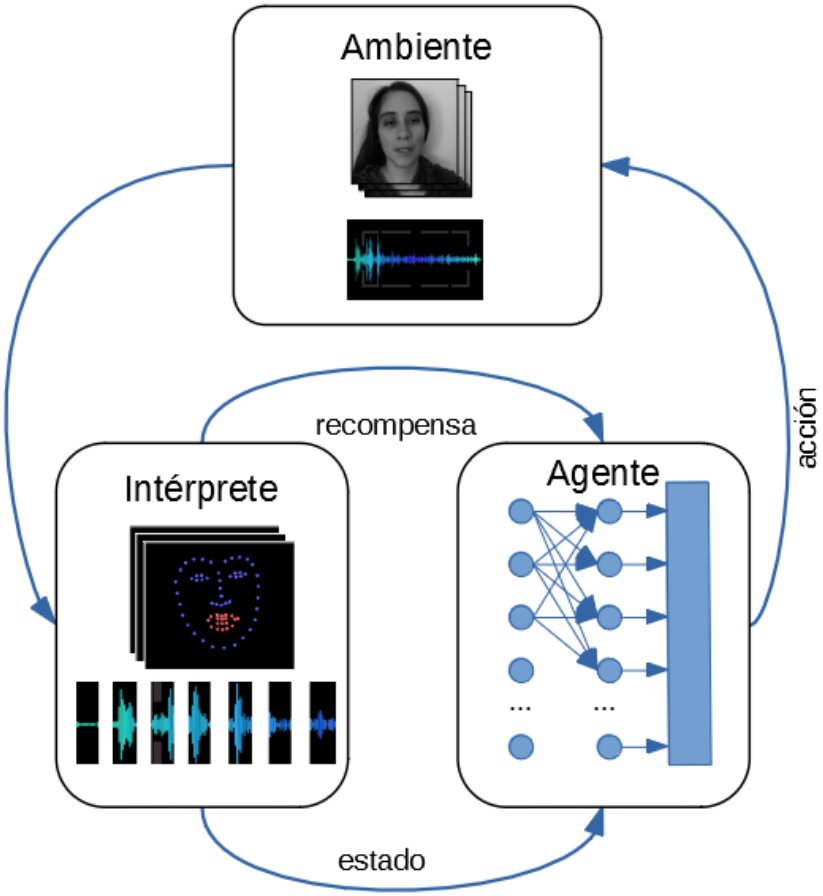

Figura 1.1 Diagrama del aprendizaje por refuerzo.

1.2. Planteamiento del problema.

Uno de los retos de la animación por computadora es el de imitar el movimiento de los personajes tomados como referencia para crear la animación, es decir, de crear movimientos naturales en la animación similares a los llevados a cabo por las personas, animales o cosas reales.

Uno de los movimientos en la animación con diálogo es el de los labios. Este tipo de movimiento presenta un reto a los animadores, ya que las personas están predispuestas a seguir el movimiento de labios y asociarlo con el audio que lo acompaña. Es por esto que el movimiento de labios mal animado distrae del contenido de la animación al crear una disonancia en el espectador entre el diálogo y el movimiento de labios.

Una técnica de animación de labios es la de capturar video de los actores de voz, con marcadores especiales adheridos a partes clave de los labios del actor, para procesar el video y audio por computadora, extraer la posición de los labios del actor para cada cuadro y así asociar las posiciones de los labios con el audio. Sin embargo, esta técnica requiere de capturar video del locutor original cada vez que se produzca una animación con audio diferente, además de procesamiento adicional para generar la animación.

1.3. Alcances y limitaciones.

Los alcances y limitaciones del proyecto son los siguientes:

1.3.1 Alcances

- Entrenar un Avatar digital a mover la boca de manera sincronizada con un determinado audio.
- Lograr que el Avatar se entrene en lenguaje español.
- Que el Avatar llegue a reproducir los movimientos de manera correcta para un audio. Se espera que este proceso sea simultáneo, es decir que a la par de que se escucha el audio, el Avatar esté moviendo la boca de manera acorde.

1.3.2 Limitaciones

- La fluidez del movimiento estará limitada por los recursos computacionales del equipo.
- El Avatar solo moverá los labios de forma correcta siguiendo un discurso en español y no otros idiomas.
- El proceso simultáneo tendrá un pequeño retraso causado por el tiempo que toma al sistema capturar el audio y generar la animación.
- El entrenamiento y generación de la animación se realizará con un mismo video.

1.4. Justificación.

En una gran cantidad de ámbitos como el educativo, las películas de animación o caricaturas, el contar con un Avatar que una vez entrenado pueda mover la boca de manera realista para que se considere que es él el que está hablando, facilitará en gran manera la generación de materiales de diferentes índoles. Pensemos en el educativo, el Avatar solo necesitará el audio y él empezará a mover la boca como si él lo estuviera diciendo.

Dentro de entornos educativos, los estudiantes que interactúan con un medio de aprendizaje con un agente pedagógico animado mostraron un crecimiento estadísticamente significativo de retención del conocimiento (Cabiedes et al, 2006).

En encuentros presenciales donde la boca del locutor no es visible un sistema capaz de generar una representación digital del movimiento de labios a partir del habla puede auxiliar a personas con dificultades de escucha a entender el mensaje.

Este escenario es común en situaciones de pandemia con el uso obligatorio de cubrebocas en espacios cerrados y áreas comunes.

Cabe mencionar que existen técnicas como las descritas en el capítulo II que tienen objetivos similares de crear animaciones automáticas a partir de audio utilizando técnicas de aprendizaje de máquina, sin embargo, el presente trabajo busca explorar la aplicación en este problema de distintas técnicas a las utilizadas en pasadas investigaciones.

1.5. Objetivos.

1.5.1. Objetivo general

Crear un sistema de software capaz de entrenar un Avatar digital para que realice el movimiento correcto de los labios siguiendo un audio, utilizando Aprendizaje por Refuerzo.

1.5.2. Objetivos específicos:

- A partir de un video, identificar las posiciones de la boca para cada cuadro (labio superior, labio inferior, comisuras) y el segmento de audio asociado a cada cuadro.
- Creación de Avatar en una plataforma de diseño y animación.
- Hacer que un Avatar digital "aprenda" a mover los labios en la secuencia correcta para que se vea como que es él quien está "hablando". El Avatar aprenderá por prueba y error usando Aprendizaje Por Refuerzo.

CAPÍTULO II. ESTADO DEL ARTE.

Se describen algunos de los trabajos publicados que tienen relación con las tres principales áreas de nuestro proyecto: Sincronización de labios, Aprendizaje de Máquina y Sistemas de visión.

2.1 Sincronización de labios

Existen trabajos que apuntan a crear Avatares digitales con sincronización de labios tomando varios enfoques:

El trabajo realizado por Cabiedes et al (2007) genera una animación en 2D basándose en un audio en español, sin embargo, la animación está estilizada como caricatura, de manera que el muestreo es bajo, genera poca credibilidad y la sencillez de la animación generada busca no causar distracciones al usuario. En el trabajo de Aneja, (Aneja y Li 2019) se utiliza un enfoque similar para audio en inglés, además de generar el Avatar 2D en tiempo real (con pequeño retraso).

En el trabajo desarrollado por Suwajanajorn (Suwajanajorn et al, 2017) se creó un sistema capaz de generar videos realistas de Barack Obama a partir de un audio del mismo Obama. El sistema se entrena primero con hasta 15 horas de video tomado de los reportes presidenciales semanales de Obama. Este sistema solo es capaz de generar contenido con la voz de un solo personaje y necesita alrededor de 15 horas de video de referencia además de procesamiento posterior adicional para producir resultados realistas.

Otros trabajos similares de síntesis de Avatares a partir de un audio utilizando aprendizaje automático (Fan et al, 2015, Karras et al, 2017, Xu et al, 2013) hacen uso de material en otros idiomas para entrenar sus modelos, por lo que son aptos para generar material en otros idiomas distintos al español, además de utilizar técnicas de Aprendizaje Supervisado. Nuestra propuesta crea un Avatar capaz de mover sus labios a partir de un audio en español utilizando técnicas de Aprendizaje por Refuerzo.

Los trabajos mencionados anteriormente también buscan crear una imitación digital de los movimientos de los labios. Sin embargo, el trabajo actual busca explorar las técnicas del aprendizaje por refuerzo para posteriormente poder compararlas con los resultados obtenidos mediante aprendizaje supervisado.

2.2 Aprendizaje de Máquina

Avances recientes en Redes Neuronales Profundas permiten entrenar estas redes para procesar datos sin un tratamiento previo, tales como audio y video (LeCun et al, 2015).

El algoritmo descrito por Volodymyr (Volodymyr et al, 2015) para entrenar a un agente a jugar al Atari 2600, permite entrenar una RNC que estima una función Q(s,a) que toma como entrada un estado s determinado por los pixeles de la pantalla actual y cuya salida representa la recompensa esperada al tomar la acción a dado el estado s. Al tomar la acción que maximice la recompensa y con suficiente exploración e iteraciones se logra obtener una aproximación lo suficientemente cercana a la función Q(s,a) real para superar retos en distintos juegos de Atari.

Para procesar audio existen enfoques como los de Aytar (Aytar et al, 2016, Wei et al, 2017) donde se utiliza una RNC Profunda, es decir, de varias capas. Este tipo de arquitecturas permiten procesar audio en forma de onda directamente y crear a partir del entrenamiento, filtros que extraen de forma secuencial características del audio cada vez más abstractas.

Otros trabajos como (Lonce 2017) utilizan representaciones del audio en espectrogramas, los cuales se alimentan a RNC similares a las utilizadas para procesar imágenes y video con convoluciones en 2D.

El sistema aquí presentado hace uso de estas técnicas para extraer características del audio sin ser procesado previamente, y generar animaciones a partir de éste.

2.3 Sistemas de visión

Para entrenar a un agente de Q-learning a que mueva los labios de manera correcta siguiendo un audio, se requiere de una señal de recompensa, el cual es un número real que funciona como una medida del rendimiento del agente al imitar el movimiento de los labios de manera similar al puntaje de un videojuego como en (Volodymyr, 2015).

Como en otros trabajos de sincronización de labios (Fan et al,2015, Karras et al, 2017), el modelo se somete a un entrenamiento durante el cual debe ser posible cuantificar la diferencia entre la posición de los labios del locutor original del audio y la posición de los labios generada por el agente a partir del mismo audio. Por esta razón, se debe poder ubicar la posición de los labios tanto originales como los generados por el agente, ya sea localizando puntos selectos en labios inferior y superior o bien, de un trazado completo de éstos.

Existen diferentes métodos para ubicar los puntos de referencia en la cara. Uno de ellos es el descrito por Kazemi y Sullivan (2014), el cual consta de un conjunto de árboles de regresión el cual se puede entrenar en una base de datos de caras etiquetadas con la ubicación de distintos puntos de referencia, incluyendo los labios.

CAPÍTULO III. MARCO TEÓRICO

Primero se presentan dos conceptos que se utilizan recurrentemente en este trabajo:

Visema: Un visema representa la posición de la cara y la boca al hablar. Es el equivalente visual de un fonema, que es la unidad acústica básica que forma una palabra. Los visemas son los componentes visuales básicos del habla.

Avatar digital o Avatar: una representación gráfica que se asocia a un usuario en particular para su identificación en un videojuego, foro de internet, etc. El Avatar puede ser una fotografía, icono, figura o dibujo artístico y puede tomar forma tridimensional, como en juegos o mundos virtuales, o bidimensional, como icono en los foros de internet y otras comunidades en línea (Lawrence, 2000).

El proyecto aquí descrito se fundamenta en distintas áreas de la ciencia y disciplinas para su implementación. A continuación, se describen los principales elementos tomados de estas áreas.

3.1 Aprendizaje Por Refuerzo

El Aprendizaje por Refuerzo (RL por sus siglas en inglés) es un área del Aprendizaje Automático que se preocupa por cómo agentes de software deben tomar acciones en un ambiente para maximizar la noción de recompensa acumulada. RL difiere del aprendizaje supervisado en que no necesita una entrada de pares de datos y etiquetas. En problemas de RL el enfoque está en encontrar un balance entre exploración (de las partes desconocidas) y explotación (del conocimiento) (Barto y Sutton, 2008).

En el modelo de RL, un agente está conectado a su ambiente a través de estados y acciones, como se muestra en la figura 1.1. En cada paso de la interacción, el agente recibe como entrada una representación del estado s del ambiente y después el agente elige una acción a. La acción modifica el estado del ambiente y el valor de esta transición de estado es comunicada al agente por medio de una señal escalar de refuerzo o recompensa r. El agente busca tomar el conjunto de acciones que tiendan a incrementar los valores de la señal de refuerzo. Esto se logra con la interacción sistemática del agente con el ambiente y existen un número de algoritmos en los cuales se puede guiar este aprendizaje (Kaelbling et al, 1996).

De manera formal, el modelo de RL consiste en los siguientes elementos:

- Un conjunto de estados del ambiente *S*.
- Un conjunto de acciones del agente *A*.
- Un conjunto de señales de recompensa.

(Barto y Sutton, 2008, Kaelbling et al, 1996)

Una política π, se describe como una relación entre estados y acciones. Con los elementos del RL antes mencionados, el agente busca encontrar una política π, la cual maximice la recompensa que el agente percibe al interactuar con el ambiente a largo plazo. Se denomina política óptima a aquella política que obtenga la mayor suma de recompensa posible a través de un episodio.

Para encontrar una política óptima existen diferentes técnicas de RL, según la naturaleza del problema como se muestra en la Figura 3.1. Al elegir una técnica de RL hay que tomar en cuenta diferentes aspectos del problema, tales como la interacción del agente con el ambiente, es decir, el espacio de acciones y estados, así como la forma y frecuencia de la señal de recompensa. A continuación, se describe la técnica de RL llamada Q-learning.

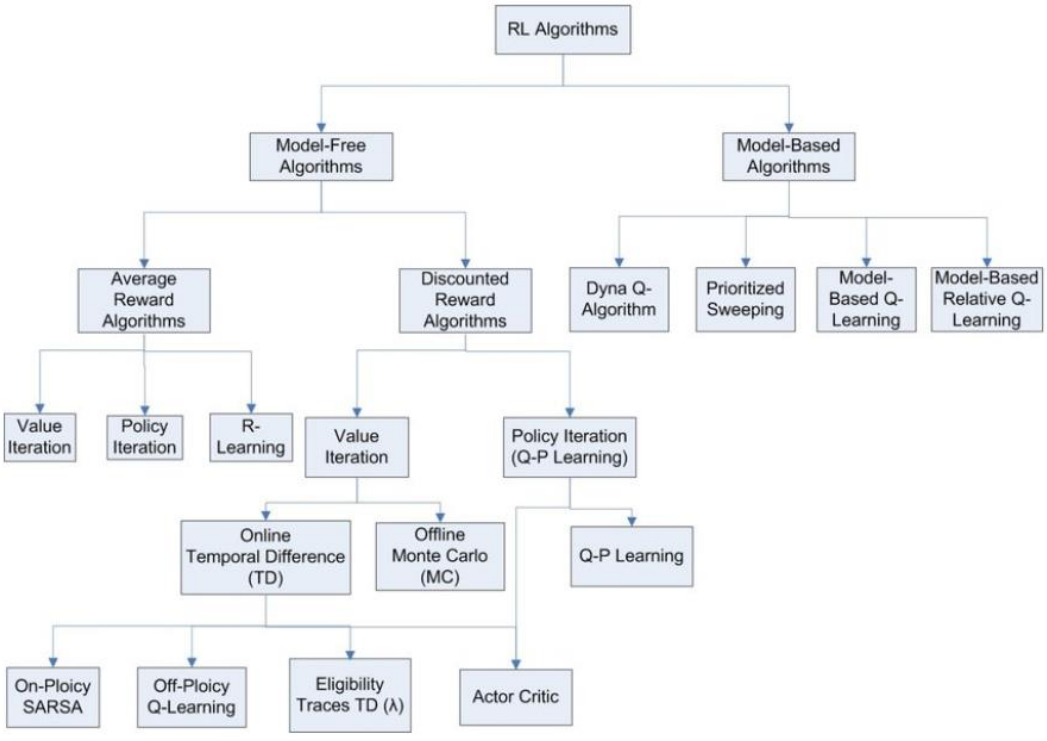

Figura 3.1 Técnicas de RL (El-Tantawy y Abdulhai, 2010)

3.1.1 Técnica de aprendizaje

En este problema de sincronización de labios, se toma de un video las posiciones de labios del locutor para compararlos con los del agente y generar la señal de recompensa, por lo que se cuenta con una señal de recompensa inmediata después de cada acción. Esto también significa que podemos plantear el problema como episódico o continuo, ya que se podría dividir un video arbitrariamente o combinar más de un video en uno solo. Esto último nos permite utilizar técnicas de RL para problemas tanto episódicos como continuos, como por ejemplo TD y MC en la Figura 3.1.

El siguiente aspecto por considerar al elegir la técnica de RL es la forma en que el agente decidirá la acción a tomar en cada estado, es decir la política.

En cada paso de interacción del agente con el ambiente, el agente recibirá una señal de audio. Esta señal debe ser reducida e interpretada por el agente para producir una acción. El principal reto es definir a partir del audio sin tratamiento previo las características que permitan decidir al agente cuál acción producirá una mayor señal de recompensa. Existen trabajos como los de Volodymyr et al (2015) y Suwajanakorn et al (2019), que utilizan Redes Neuronales Convolucionales (RNC), que toman como entrada datos sin tratamiento previo como video o audio y son capaces de extraer características que funcionan como una abstracción o lista reducida de datos que definen a la entrada y pueden ser alimentadas posteriormente a una Red Neuronal Completamente Conectada (RNCC) que produce a partir de estos datos, valores que pueden ser fácilmente utilizados para definir la mejor acción a tomar para el estado de entrada. El funcionamiento de las RNC se detalla más adelante en esta misma sección.

El algoritmo comúnmente utilizado para entrenar RNCs conectadas a RNCCs para tomar decisiones dentro del RL es el de Deep Q-Learning y las Redes Neuronales compuestas de RNC y RNCC (u otros tipos de Redes), conectadas en serie aplicadas a problemas de Deep Q-Learning son llamadas Deep Q Network o DQN.

En Q-Learning, se parte del supuesto de que las recompensas en el ambiente están dadas por una función no lineal, la cual toma como entrada el estado s y la acción a, y produce un valor que representa la recompensa potencial que el agente esperaría recibir a partir de tomar la acción a en el estado s. El valor de la función Q cuenta además con un descuento arbitrario gama γ el cual

funciona como una medida de la preferencia del agente por la recompensa inmediata contra la recompensa a futuro.

Resulta impráctico buscar una función Q que describa la recompensa en el ambiente dados *a* y *s*, debido al gran número de estados que se pueden dar en el ambiente, por lo que se utiliza un estimador de función no lineal, normalmente una RNC conectada con una RNCC (Sutton y Barto, 2014).

Las Redes Neuronales Convolucionales (RNCs) son un tipo especial de Red Neuronal Artificial y toma su nombre del operador matemático de convolución, aplicado en matrices. Las RNCs constan de varias capas de neuronas como las capas de convolución, de no linealidad, de agrupamiento y completamente conectadas. Las capas de convolución y completamente conectadas tienen parámetros como en otras redes, mientras que las de no linealidad y agrupamiento no tienen. Las RNCs tienen un excelente rendimiento en problemas de aprendizaje automático. Especialmente en las aplicaciones que tratan con datos de imágenes, (tales como Image Net), la visión por computadora, y en el procesamiento del lenguaje natural (Albawi y Mohammed, 2017), ya que normalmente se cuentan con grandes bases de datos para el entrenamiento.

Como explican Albawi y Mohammed (2017), las distintas capas de una RNC funcionan en conjunto para resaltar las características más relevantes de los datos. Comenzando con una capa de convolución, la cual tiene una serie de filtros cuya dimensionalidad depende de los datos de entrada, por ejemplo, para imágenes de un canal, el filtro podría ser de 3x3. Cuando se alimenta a la RNC con los datos de entrada, cada filtro realiza una operación de convolución que consiste en pasar secuencialmente por toda la imagen, creando una nueva imagen que es el resultado de la suma de los productos de los elementos del filtro con la imagen. Esta operación permite utilizar los mismos parámetros, agrupados en filtros, para toda la imagen lo que incrementa el rendimiento y la eficiencia en el uso de parámetros o pesos en comparación a una Red Neuronal Artificial convencional.

Las capas de agrupamiento (pooling en inglés) buscan resumir la información para reducir la complejidad de las capas siguientes. Una forma común de agrupamiento es el Maxpooling que consiste tomar el máximo valor de cada segmento a agrupar (Goodfellow et al, 2016).

Por otro lado, la capa de no linealidad se describe como una función aplicada a la salida de la capa anterior que modifica la activación de las neuronas y facilita el entrenamiento (Albawi y Mohammed, 2017). Una de las funciones más populares en trabajos recientes como el de Volodymyr et al (2005) es ReLU (Rectified Linear Unit, en inglés) y consiste en tomar el valor mayor entre 0 y x (la entrada de la función). Y la podemos escribir como:

$$f(x) = \max(0, x)$$ (What is ReLU, 2021) (1.1)

Como lo mencionan Hasselt et al (2006), existen problemas de sobreestimación y rendimiento en Q-learning al utilizar RNCs. Una solución es utilizar dos RNC: sobre la primera se toman las decisiones en cada paso mientras que la segunda se actualiza. Finalmente, los cambios de la segunda RNC se reflejan cada cierto número de pasos en la primer RNC.

Barto y Sutton (2015) dicen que otro de los retos importantes en RL es el de exploración contra explotación, que se refiere a la cantidad de exploración que debe hacer un agente antes de comenzar a dar prioridad a obtener una mayor recompensa. Para obtener una cantidad grande de recompensa, el agente debe tomar acciones que ha tomado anteriormente y han probado producir una recompensa grande. Pero para descubrir tales acciones, debe probar acciones que no ha seleccionado aún. El agente debe explotar su conocimiento sobre el ambiente, pero también debe explorar para tomar mejores acciones en el futuro.

Existen varios acercamientos para resolver este problema, algunos de ellos, como el de Optimismo, no son fáciles de aplicar al utilizar Redes Neuronales Artificiales (RNA) y no profundizaremos en éste. Por otro lado, destaca la técnica de Epsilon-greedy (ε-greedy) que consiste en definir el grado de exploración del agente mediante un parámetro ε. Este representa el porcentaje de las acciones que el agente tomará al azar (exploración) y el resto se harán tomando la acción que en ese momento prometa la máxima recompensa (explotando). En este enfoque, ε toma un valor de 0 a 1, y la acción se elige con el siguiente pseudocódigo:

```
x = valor_aleatorio()
Si x < ε:
        Tomar acción exploratoria
Si no:
        Tomar acción maximizando recompensa
```

Diferentes valores de ε-greedy resultan en diferentes ritmos de aprendizaje, por lo general, entre más alto el valor de ε, es decir, mayor exploración, más rápido el agente logra obtener una política óptima, como se ilustra en la figura 3.2 en escala logarítmica para el problema del multi-armed bandit. La desventaja de valores altos de exploración es que, al llegar a la política óptima, el agente sigue explorando, siguiendo el valor de ε y obtiene menor recompensa total comparado a valores más bajos de ε cuando el número de pasos es grande. Esto se ve demostrado en distintos trabajos de RL (Barto y Sutto, 2015, Shawl, 2020).

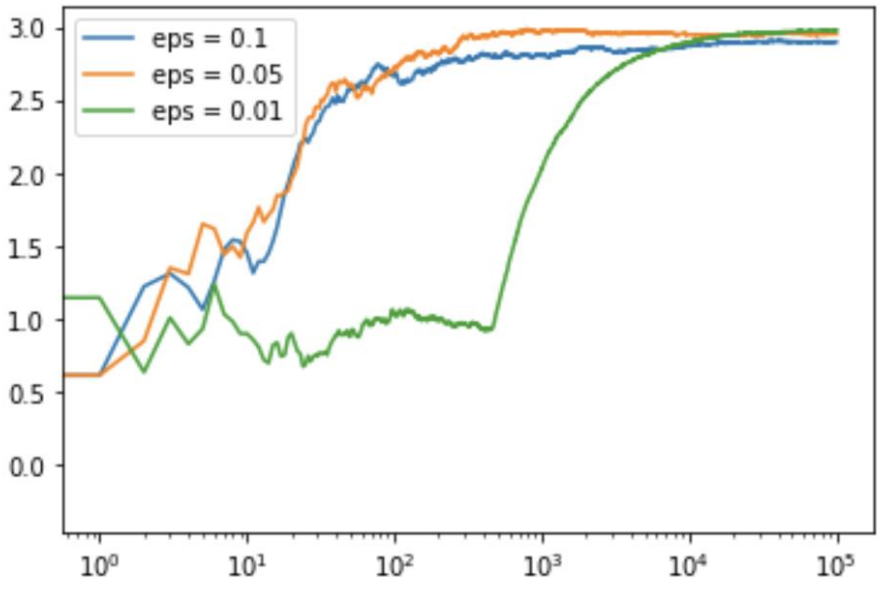

Figura 3.2 Recompensa total para distintos valores de épsilon en el problema del *multi-armed bandit* (Shawl, 2020).

Adicionalmente, se puede hacer disminuir ε con cada paso a lo largo del entrenamiento, es decir, asignar un valor grande de exploración al principio e ir disminuyendo el tiempo que el agente explora (Volodymyr, 2015).

3.2 Procesamiento de señales de audio

Para definir la tasa de muestreo mínima del audio que se alimenta al agente se parte del teorema de muestreo Nyquist-Shannon, definido por Nyquist (1928) y probado por Shannon (1949) que dice lo siguiente:

"La frecuencia de muestreo debe ser por lo menos dos veces mayor a la frecuencia más alta contenida en la señal. Esto se expresa matemáticamente como:

$$f_s \geq 2f_c \tag{3.2}$$

donde fs es la frecuencia de muestreo y fc es la frecuencia más alta que contiene la señal."

Con base a lo anterior y tomando en cuenta el rango de frecuencia de la voz humana cuyo límite superior es de 8kHz, se calcula la frecuencia óptima de muestreo como 16kHz.

El utilizar una frecuencia de muestreo más baja a 16kHz puede generar una interpretación errónea de la información del audio, conocida como *Aliasing*. Este problema puede causar que algunas de las características del audio se pierdan, principalmente aquellas en frecuencias altas.

Para almacenar el audio en la computadora existen diversos formatos. Uno de ellos es el Waveform Audio Format (WAV o WAVE por la terminación del formato), el cual almacena los datos de audio de forma secuencial en uno o dos canales y con distintas resoluciones y frecuencia de muestreo. Otra ventaja que presenta este formato es su facilidad de leer y manipular debido a que no comprime los datos del audio (Branson, 2015).

Al procesar el audio, éste puede interpretarse de distintas maneras tales como forma de onda, características manipuladas o extraídas con técnicas de aprendizaje de máquina y espectrogramas como Coeficientes Cepstrales en las Frecuencias de Mel (MFCC, por sus siglas en inglés) y sus formas derivadas (Wyse, 2017).

Entre estas formas de representación de audio, son de especial interés la forma de onda y MFCC, la primera por la sencillez de procesamiento ya que simplemente se requiere leer del almacenamiento el archivo en formato WAV, y la segunda por los avances recientes en Aprendizaje Profundo para su procesamiento y posterior uso en aplicaciones de Aprendizaje de Máquina, como lo demuestra (Aytar 2016). También existen avances que permiten utilizar la forma de onda sin tratamiento previo en RNCs.

3.3 Procesamiento de video

La posición de los labios de un locutor podría ser comparado con el trazo de otro video generado a partir de un Avatar manipulado por agente de RL y generar una función de recompensa para así entrenar al agente a imitar el movimiento de labios. Para ello, se deben identificar las posiciones de los labios del locutor original en cada uno de los cuadros del video para poder empatar la posición de cada cuadro con el audio que produce el locutor en cada momento. Es por esto que se busca una técnica de detección de puntos en la cara que sea lo suficientemente precisa para identificar distintos puntos clave de la boca y rápida, para procesar todos los cuadros de un video en un tiempo corto.

3.3.1 Detección de puntos

Hasan y Ahmed (2014) proponen un sistema de trazado de labios basado en regiones de modelado de contornos.

Otro método para la detección de labios es el descrito por Liévin et al (1999) basado en segmentación bayesiana, pero presenta limitaciones en la detección.

Kazemi V. y Sullivan J. (2014) describen otro enfoque en el trazado de labios con énfasis en la rapidez y precisión de detección utilizando un conjunto de árboles de regresión entrenado con caras etiquetadas con la ubicación de distintos puntos de referencia.

Este último enfoque es implementado por Rosebrock (2017) como una librería de Python llamada dlib, y cuenta con un modelo previamente entrenado en una base de datos de caras etiquetadas, por lo que no requiere más entrenamiento para extraer posiciones de la cara.

3.4 Ingeniería del software

El proyecto actual conlleva el desarrollo de software para alcanzar los objetivos planteados para el proyecto. A continuación, se describen las herramientas consideradas para auxiliar el desarrollo de software.

3.4.1 Manejo de versiones

El software de manejo de versiones facilita la administración del código fuente y otros documentos de un proyecto.

Git es un software de manejo de versiones para llevar registro de los cambios en el código fuente durante el desarrollo (Scopatz y Kathryn 2015). Github es una implementación de Git en la nube que permite a sus usuarios llevar el control de versión en la nube.

Esta herramienta se puede usar para tener un registro de los cambios en el proyecto a través de todo el desarrollo y tener un respaldo del código fuente, así como también mantener registro de los resultados de los experimentos y otros documentos del proyecto.

3.5 Software de animación

3.5.1 Tkinter

Para la evaluación del desempeño del agente y su depuración durante el desarrollo, se debe mostrar en pantalla la animación generada a partir de la manipulación del agente de RL sobre el Avatar de labios. Con este propósito, la librería de tkinter de Python permite crear interfaces gráficas de usuario (GUI).

Tktinter es la interfaz por defecto de Python. Y permite desarrollar GUIs para distintos sistemas operativos como Windows, Unix y Mac.

3.5.1 OpenCV

Open Source Computer Vision Library (OpenCV) es una librería de visión por computadora y Aprendizaje de Máquina. Esta librería está construida para proveer una infraestructura común para aplicaciones de visión por computadora y acelerar su uso en productos comerciales.

OpenCV usa una licencia BDS, lo cual facilita a los negocios el utilizar y modificar el código. Además, éste se encuentra disponible en su sitio para su descarga (About OpenCV, 2019).

CAPÍTULO IV. DESARROLLO

4.1 Análisis

Para efectuar el análisis inicial se parte de los objetivos, los cuales se traducen posteriormente en requerimientos. El objetivo general del proyecto es “entrenar un Avatar digital para que realice el movimiento correcto de los labios siguiendo un audio, utilizando Aprendizaje por Refuerzo”. De este objetivo, se identifican tres objetivos específicos, definidos en el capítulo I:

A partir de un video, identificar las posiciones de la boca para cada cuadro (labio superior, labio inferior, comisuras) y el segmento de audio asociado a cada cuadro.

Creación de Avatar en una plataforma de diseño y animación.

Hacer que un Avatar digital “aprenda” a mover los labios en la secuencia correcta para que se vea como que es él quien está “hablando”. El Avatar aprenderá por prueba y error usando Aprendizaje Por Refuerzo.

A continuación, se toman cada uno de estos objetivos y se profundiza en los requerimientos que debe cumplir el sistema para llevar cada uno de ellos a cabo.

4.1.1 Identificación de fonemas y visemas

Como punto inicial, se debe capturar video que servirá para crear los datos de entrenamiento. Se necesita de video donde se pueda apreciar el movimiento de los labios y el audio correspondiente.

El video debe cumplir con los siguientes requisitos:

1. Buena iluminación.
2. A color o blanco y negro.
3. Una resolución lo suficientemente alta para identificar de forma automática puntos clave de la boca, 720p en modo retrato.
4. El locutor de frente a la cámara, centrado y alineado.
5. El locutor debe hablar en español durante la duración total del video.
6. El video debe ser capturado a 30 cuadros por segundo, esto para proveer una transición nítida del movimiento de los labios.
7. El audio del video debe tener una frecuencia de muestreo de por lo menos 16kHz.

Se partirá de un video de un solo locutor de 30 minutos, lo cual aporta suficientes datos para el entrenamiento.

Como siguiente paso a la captura, hay que extraer las posiciones de los puntos clave de la boca. Para comparar la posición de la boca del locutor y del agente, primero hay que identificar cuatro puntos en la boca del locutor: centro del labio superior, centro del labio inferior, comisura izquierda y derecha. La extracción de características debe hacerse de forma automática de modo que se puedan extraer datos adicionales de nuevos videos más adelante en el proyecto.

Para cada cuadro del video, se extraen las posiciones de puntos clave de los labios en forma de coordenadas de un plano cartesiano o similar, luego, tomando como referencia la línea recta entre la base y la punta de la nariz hay que rotar la cara de forma que la línea mencionada se encuentre paralela al eje de las ***y***. Por último, se acoplan las posiciones con el segmento correspondiente de audio (de 10 a 500 milisegundos).

4.1.2 Creación del Avatar

El sistema debe contar con un módulo de animación que despliegue una secuencia de imágenes a 30 cuadros por segundo, la cual represente el movimiento de labios generado por el sistema y lo muestre al usuario. Este módulo debe cumplir con los siguientes requisitos:

1. El usuario debe poder elegir:
    a. La fuente del audio en base a la cual el sistema generará el movimiento de labios.
    b. La configuración que se utilizará para generar la animación. Se entiende como configuración a los diferentes resultados de entrenamiento generados durante las pruebas.
    c. Un video con audio para generar la animación, en cuyo caso el módulo mostrará lado a lado las posiciones del locutor con las generadas con el sistema, a fin de comparar el rendimiento del sistema.
2. El módulo debe mostrar como representación de los labios al menos las cuatro posiciones a ser utilizadas como punto de comparación, estas son; centro de labios superior e inferior, comisuras izquierda y derecha

4.1.3 Aprendizaje e imitación

Después de un entrenamiento de preparación y teniendo un audio sin procesamiento previo de una persona hablando en español como entrada, el sistema de labios debe imitar el movimiento de labios del locutor y generar una animación de los labios, como se describe en los dos segmentos anteriores. La forma en que el agente debe llevar a cabo el aprendizaje de movimiento de labios se describe a continuación:

1. El sistema debe contar con un módulo de Aprendizaje de Máquina, el cual pueda aprender las relaciones entre el audio y los movimientos que el Avatar debe ejecutar en los labios para simular el movimiento humano.
2. El aprendizaje debe llevarse a cabo utilizando una de las técnicas de Aprendizaje por Refuerzo (RL). Puede ser alguno de los mencionados en Sutton y Barto (2015) u otros.
3. La experiencia del módulo de aprendizaje debe después de ser entrenado, recibir un audio y producir una animación de sincronización de labios correspondiente al audio.

El marco del Aprendizaje por Refuerzo consta de tres partes que debe incluir el proyecto, ilustrados en la figura 1.1. Estos son:

1. Ambiente
2. Agente
3. Intérprete

A continuación, se describen a detalle cada uno de estos elementos y los requisitos que deben cumplir en el sistema.

4.1.3.1 Ambiente

El ambiente debe contener las clases, datos y librerías que juntas proveen una serie de estados al agente y las posiciones de los labios al intérprete. La entrada del ambiente es el video previamente capturado y sus salidas son una serie de datos que son alimentados de forma secuencial al agente o intérprete. La composición de estos datos, para cada cuadro del video es:

1. Las posiciones de los labios para el respectivo cuadro de video.

2. Un segmento de audio que comience de 30 a 1000ms antes de la aparición del cuadro de video y termine al mismo tiempo que aparece el cuadro de video. El sistema debe permitir la experimentación de varias duraciones de segmento, la duración mínima y máxima del segmento puede variar según los resultados de la experimentación.

4.1.3.2 Agente

El agente se compone de las clases, algoritmos y datos que le permitan tomar acciones en el ambiente tomando como entrada un segmento de audio y pueda aprender en base a las señales de recompensa que recibe después de haber tomado cada una de las acciones en el ambiente.

4.1.3.3 Intérprete

El intérprete funciona como una interfaz entre el ambiente y el agente. Éste toma datos del ambiente y los convierte en una señal de recompensa que el agente puede utilizar para aprender las relaciones entre la última acción que tomó en el ambiente y qué tan cercanamente sigue los movimientos de labios de locutor.

Durante el entrenamiento, el intérprete debe recibir las posiciones de los labios del locutor y Avatar de parte del ambiente y calcular una señal de recompensa en base a la cercanía entre las posiciones.

La forma en que se calcula la recompensa a partir de las posiciones de la boca debe ser fácilmente modificable para experimentación.

4.1.4 Desarrollo guiado por pruebas

La metodología elegida para desarrollar el proyecto es la de “desarrollo guiado por pruebas” (Test Driven Development o TDD, por sus siglas en inglés) el cual consiste en la repetición de un ciclo corto: los requerimientos se convierten en casos de prueba muy específicos, luego el código es mejorado para que pase las pruebas (Beck, 2002).

En base a la metodología de TDD, se basa en el diagrama de la figura 4.1 para describir el plan de trabajo.

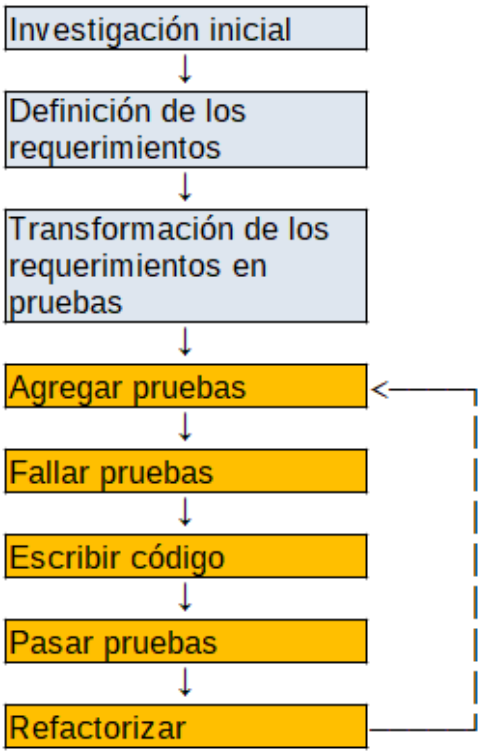

Figura 4.1 Diagrama de la metodología de trabajo.

El diagrama consiste en dos partes:

Parte inicial: en gris en la figura 4.1, donde se definen el alcance y limitaciones del proyecto, así como también los requerimientos y se realiza una investigación preliminar sobre el estado del arte y marco teórico. Esta parte corresponde a los capítulos I, II y III.

Parte final: resaltada en color naranja en la figura 4.1, donde se lleva a cabo la parte del desarrollo del software. Esta parte corresponde al ciclo de desarrollo guiado por pruebas. Los pasos se describen a continuación:

1. Agregar pruebas: Consiste en tomar uno de los requerimientos y hacerlo una prueba automática. El proceso para generar las pruebas en base a los requerimientos se describe detalladamente en la siguiente sección (4.4 – Pruebas).
2. Fallar pruebas: Una vez creada una nueva prueba, se ejecuta y verifica que falle. Si no falla, eso quiere decir que el sistema ya cumple con el requisito: no es necesario agregar código,

se pueden omitir los siguientes pasos y buscar un nuevo requerimiento. Por el contrario, si la prueba falla entonces hay que seguir con el paso 3.

3. Escribir código: Se escribe el mínimo código necesario para pasar la prueba.
4. Pasar pruebas: Se verifica que el código nuevo pase la prueba, si no lo hace, volver al paso 3.
5. Refactorizar: Aplicar los principios de código limpio para facilitar el desarrollo y mantenimiento del proyecto. Estos principios se explican en la siguiente sección.

4.1.5 Código limpio

Una de las etapas del desarrollo guiado por pruebas consiste en refactorizar el código de manera que sea legible y fácil de mantener. Se utilizarán los principios de código limpio descritos en el libro de Martin R (2008), los cuales describen:

1. Mejores prácticas para nombramiento de clases, métodos:
    a. Uso de nombres concisos y significativos que describan a las clases, métodos y variables.
    b. Congruencia en el uso de términos (no utilizar palabras distintas para describir un solo concepto).
2. Mejores prácticas para la creación y nombramiento de funciones:
    a. Nombres descriptivos.
    b. Funciones separadas por nivel de abstracción.
3. Mejores prácticas para comentarios:
    a. Minimizar el uso de comentarios.

El nombramiento de las variables, clases, funciones y otros artefactos se basa en la Guía para Código Python – PEP 8, disponible en la página oficial de Python (Van Rossum G, 2001).

La plantilla para los nombres de los principales artefactos es como se ilustra en la tabla 4.1.

Tabla 4.1. Plantilla para nombres de variables

Tipo	**Público**	**Interno**
Paquetes	lower_with_under	
Módulos	lower_with_under	_lower_with_under
Clases	CapWords	_CapWords
Excepciones	CapWords	
Funciones	lower_with_under()	_lower_with_under()
Constantes Globales/Clase	CAPS_WITH_UNDER	_CAPS_WITH_UNDER
Var Globales/Constantes	lower_with_under	_lower_with_under
Variables de Instancia	lower_with_under	_lower_with_under
Nombres de Métodos	lower_with_under()	_lower_with_under()
Función/Método	Parameters	lower_with_under
Variables Locales	lower_with_under	

Además, el código Python debe cumplir con lo siguiente:

1. La documentación de métodos y clases (docstring) deberá llevar triple comilla sencilla (’).
2. Las líneas de código no deben ser más largas de 100 caracteres.
3. El código, nombres de archivos, clases, variables, funciones y comentarios se escriben en inglés. El nombre de la carpeta y subcarpeta principales “Labios” se considera el nombre del proyecto, por lo que se mantiene en español.

4.1.6 Manejo de versiones

El manejo de versiones se llevará a cabo utilizando GitHub, el cual fue descrito en el capitulo III: Marco Teórico.

Cada cambio al código fuente será reflejado en GitHub, con comentarios y etiquetas necesarias para ubicar de manera rápida cualquier versión importante que pueda ser utilizada como punto de recuperación o referencia en etapas posteriores.

4.2 Diseño

A partir de los requerimientos definidos durante el análisis, se propone un diseño del sistema para su posterior implementación. De forma similar al análisis, se parte de un diseño general del sistema y luego se desglosa cada una de las partes del diseño.

El objetivo del proyecto es tener un sistema que aprenda a imitar el movimiento de los labios para luego generar animaciones a partir de nuevos audios. Se parte de dos módulos principales como lo ilustra la figura 4.2:

El primero de aprendizaje, que aprende a imitar el movimiento de labios.

Un segundo módulo de animación utiliza la experiencia del módulo de aprendizaje para generar animaciones a partir de un nuevo audio.

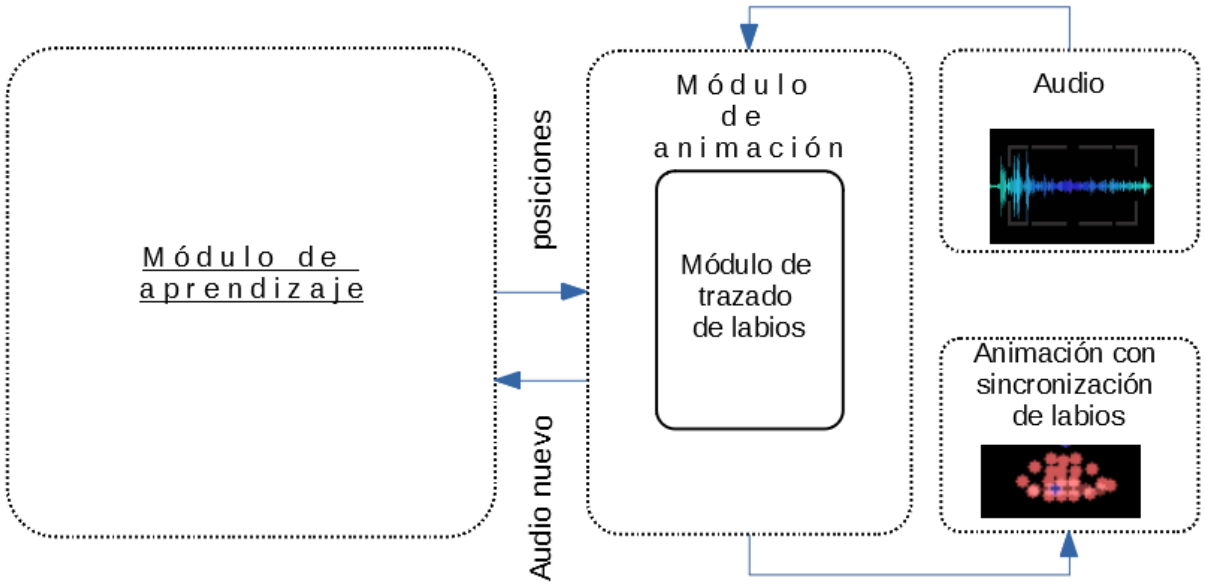

Figura 4.2 Diseño de alto nivel del sistema.

A continuación, se describen los dos módulos generales del sistema los cuales se ilustran más a detalle en la figura 4.3:

4.2.1 Módulo de aprendizaje

El Aprendizaje por Refuerzo se puede describir como un agente interactuando en un ambiente que cambia de estado a partir de las acciones tomadas por el agente y a su vez presenta una señal de recompensa la cual el agente busca maximizar (Volodymyr et al 2015).

El módulo de aprendizaje consta de los modelos y algoritmos necesarios para aprender el movimiento de labios después de un entrenamiento y se define en base al marco de Aprendizaje por Refuerzo descrito en el capítulo 3 y también según Sutton y Barto (2018) que consta de:

Una política: Es una relación de estados a acciones que le dice al agente cuál acción tomar en cada estado.

Una señal de recompensa: La cual se busca maximizar para encontrar la política óptima.

Una función de valor (de estado-acción): Que indica la recompensa que se espera recibir a partir de ese estado o estado-acción.

Al igual que en otros trabajos de Aprendizaje por Refuerzo, buscamos maximizar una señal de recompensa, la cual en este caso indica qué tan cercanamente los movimientos de los labios del agente siguen a los del locutor original. Suponemos que las recompensas en el ambiente están dadas por la función Q(s,a) la cual podemos aproximar iterativamente mediante la ecuación 4.1 como se describe en Sutton y Barto (2018).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right] \quad (4.1)$$

Donde s es el estado actual del ambiente dado por la señal de sonido, a la acción tomada por el agente en el estado s, elegida entre 16 acciones diferentes. $Q(s_t,a_t)$, que representa la recompensa que el agente espera ganar a partir de un estado s_t y tomando la acción at puede ser inicializada con un valor arbitrario para cada s_t y a_t. El valor de α es el ritmo de aprendizaje y puede ser cualquier valor entre 0 y 1; R_{t+1} es la recompensa obtenida durante la transición de estado, γ es el valor de descuento que define la importancia que se le da a la recompensa futura mientras que (a)max $Q(s_{t+1},a_t)$ - $Q(s_t,a_t)$ es el valor máximo de recompensa a partir del estado siguiente tomando la acción a.

Esta actualización iterativa aproxima directamente **q*** que es la función acción-valor óptima que maximiza la recompensa (Sutton y Barto, 2018). Sin embargo, ya que resulta impráctico el encontrar una función Q(s,a) debido a los recursos computacionales y el hecho de que se busca generalizar, es práctica común en este tipo de problemas el utilizar funciones de aproximación, estando entre ellas las Redes Neuronales de Convolución que dan muy buenos resultados para aproximar la función Q(s,a) como en Volodymyr (2015).

Sin embargo, surgen dificultades a partir de usar una Red Neuronal para aproximar la función Q(s,a), principalmente el que la Red Neuronal tiende a no converger en la función óptima, al estar iterando sobre una política cambiante que depende en las salidas de la misma red. Este problema se resuelve utilizando un mecanismo llamado *experience replay*, que toma muestras aleatorias de experiencias pasadas para entrenar la red, así como también limitando las actualizaciones a la Red Neuronal que representa a la función Q. Se utilizan dos redes, una para tomar las decisiones directamente y otra que se entrena en cada transición de estado y cuyos cambios se copian a la primera red cada 1000 pasos (Volodymyr, 2015), ilustrado en la figura 4.3 en la parte del Agente Q-learning.

Como se menciona en el trabajo de Sutton y Barto (2018), cabe destacar que el entrenamiento de la RNC que estima la función Q se puede ver como un problema de aprendizaje supervisado, ya que se cuenta con datos (el estado) y una etiqueta que es el valor de recompensa potencial hacia el cual queremos acercar las predicciones de la RNC.

La forma en que se implementa la ecuación 4.1 es en el entrenamiento de la RNC, donde se calcula el valor de salida esperado, el cual se utiliza a su vez para obtener la pérdida con la cual entrenamos a la RNC.

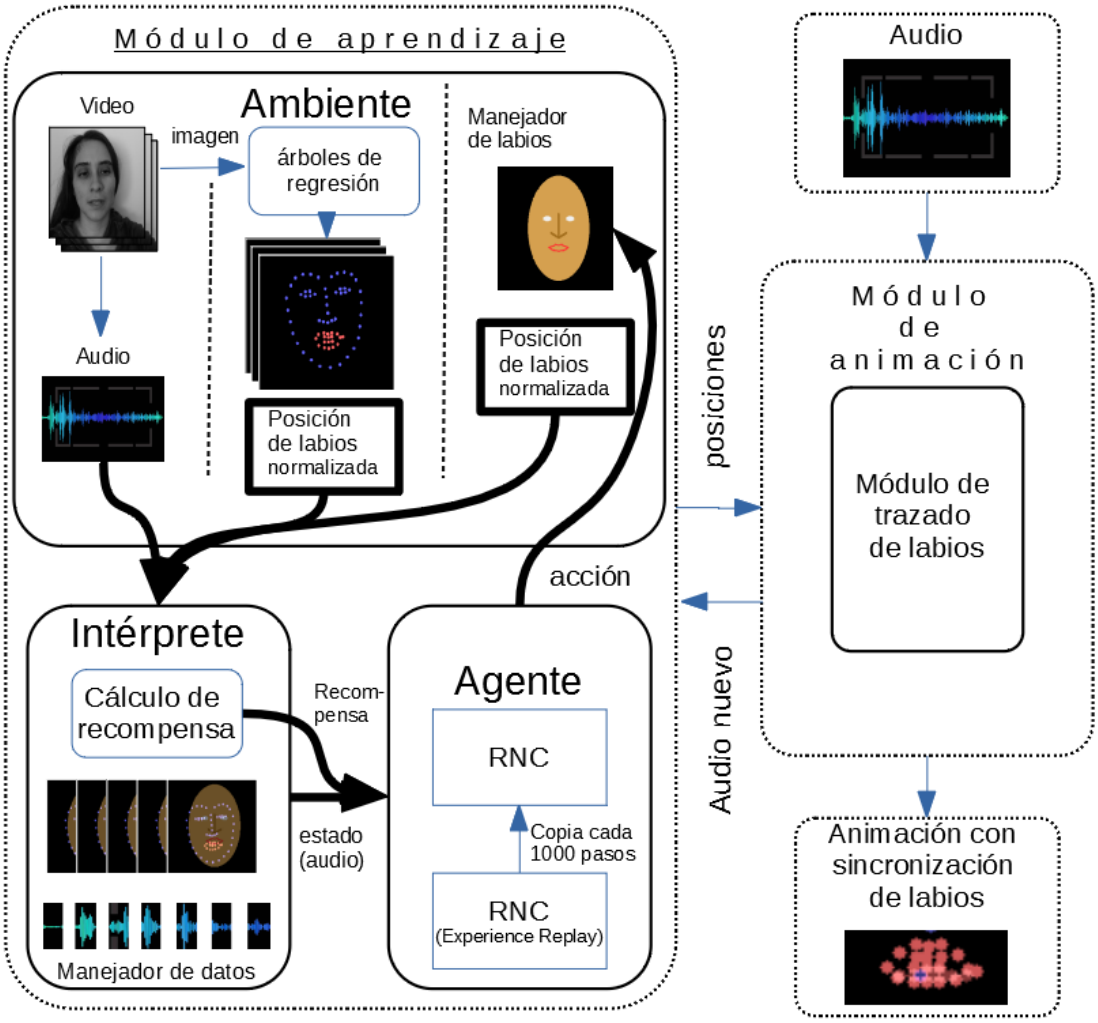

Figura 4.3 Esquema detallado de las partes del sistema.

4.2.1.1 Arquitectura de la Red Neuronal Convolucional

La RNC del sistema ilustrada en la figura 4.4, está basada en la red descrita en el trabajo de Mansar (2018) y consta de varias capas de convoluciones, agrupamiento y al final, capas completamente conectadas. A cada dos capas de convolución le siguen una de agrupación tomando los mayores valores de cada segmento (max-pooling) y después se aplica una desactivación aleatoria (dropout) del 10% para prevenir sobreajuste (overfitting), todo esto, desde las convoluciones, agrupación y desactivación aleatoria se hace cuatro veces en cadena. De esta forma se obtienen dos capas de convolución con 16 filtros cada una, dos con 32 y dos capas más con 256 filtros, cada par seguido por agrupamiento y desactivación aleatoria. La salida de las capas de convolución representa las

características del audio extraídas a partir de los filtros de convolución y esta salida se alimenta a la segunda parte de la Red Neuronal, la cual consiste en dos capas de neuronas completamente conectadas que generan un valor numérico por cada acción disponible en el ambiente. El valor de cada salida representa el estimado de la función Q(s,a) donde a y s son los datos de entrada a la Red Neuronal.

Además, se propone otra red donde se suman a la salida de la RNC los parámetros que describen a las posiciones de los labios y juntos forman la entrada de la RNCC.

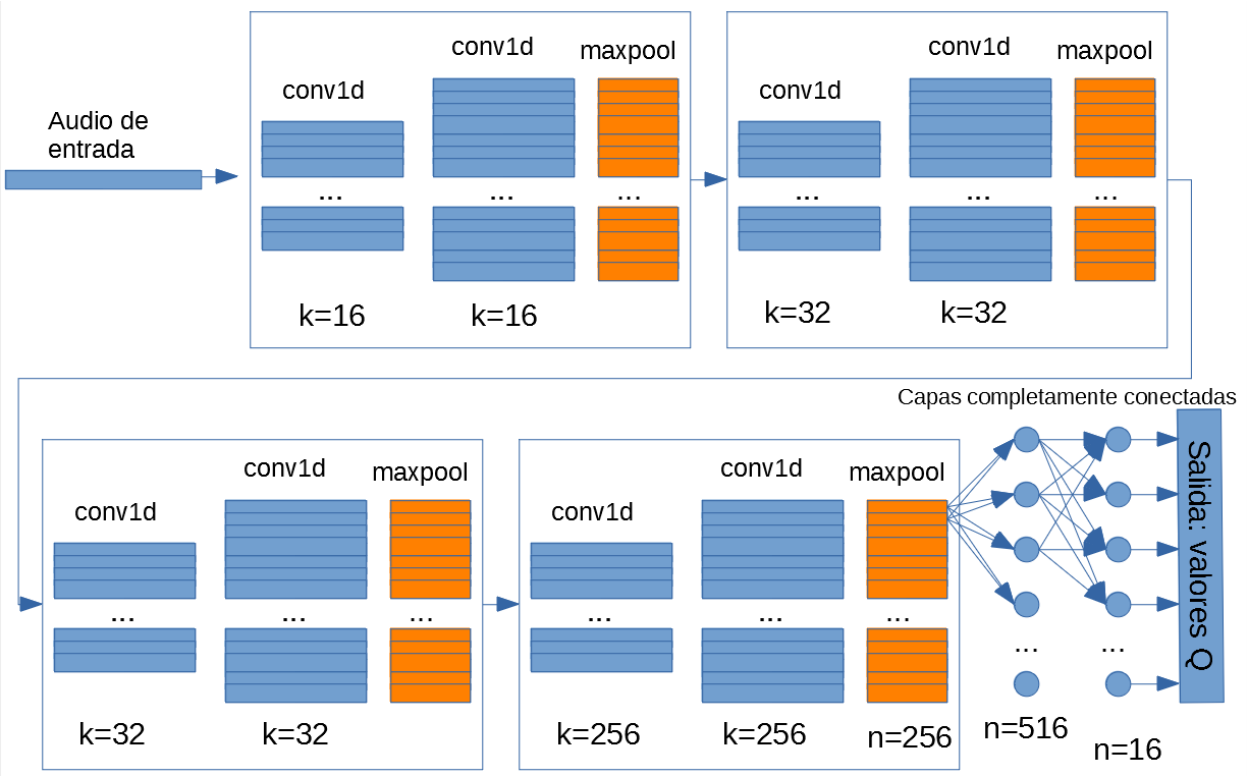

Figura 4.4 Arquitectura de la Red Neuronal Convolucional

4.2.1.2 Cálculo de la recompensa

La recompensa en el Aprendizaje por Refuerzo es un número real que le dice al agente qué tan bien se está desempeñando en el ambiente. Se busca que el agente siga los labios del locutor lo más cercanamente posible, por lo que durante el entrenamiento la recompensa debe ser una medida de qué tan cerca están siguiendo los labios controlados por el agente a los del locutor. Los labios del locutor se representan como 4 pares de valores que ubican las coordenadas de 4 diferentes

puntos de los labios en una imagen. Estos puntos son las comisuras izquierda y derecha, el centro del labio superior y centro del labio inferior. Los puntos de los labios del agente se representan de la misma manera.

La recompensa en cada paso se debe calcular con el siguiente algoritmo:

Entrada: *Posiciones de los labios del agente y el locutor.*
Para labios del agente y locutor, calcular:
distancia_labio_superior = centro_labio_superior_y – centro_boca_y
distancia _labio_inferior = centro_labio_inferior_y – centro_boca_y
distancia _comisuras = comisura_izq_x – centro_boca_x

a = agente.distancia_labio_superior - locutor.distancia_labio_superior
b = agente.distancia_labio_inferior - locutor.distancia_labio_inferior
c = agente.distancia_comisuras - locutor.distancia_ inferior

recompensa = - (a^2 + b^2 + c^2)

De esta forma, la recompensa es el negativo de la suma de los cuadrados de las distancias entre los puntos clave de los labios del locutor y el agente. Y sirve como una medida de qué tan cerca sigue a los labios del locutor conforme avanza el entrenamiento, especialmente al promediarlo para varios pasos. Se utiliza el cuadrado de las distancias para disminuir la penalización cuando las posiciones de la boca son ligeramente diferentes.

A través del entrenamiento el agente busca maximizar la recompensa, o en este caso, reducir la penalización, al aproximar la recompensa a cero.

Durante el entrenamiento, se alimenta la RNC con 30 a 100 milisegundos de audio, sin tratamiento previo, es decir en forma de onda y se busca que la RNC determine la acción que brinde un mayor valor esperado de recompensa. De esta forma se obtiene la política del agente. La duración del segmento de audio es constante a lo largo de cada experimento, pero puede variar en cada ejecución y su duración puede ser modificada por el usuario.

Después de tomar la acción con mayor valor, se compara la recompensa real con el estimado generado por la red neuronal, la diferencia se conoce como error y se utiliza para cambiar ligeramente los pesos de la red e irla entrenando.

Uno de los principales problemas del Aprendizaje por Refuerzo está en encontrar un balance entre exploración (de las partes desconocidas) y explotación (del conocimiento) (Sutton y Barto, 2018). En este caso, la exploración está representada por ε, cuyo valor indica el grado de exploración deseado, es el porcentaje de acciones que se escogerán aleatoriamente del espacio de acciones sin importar que exista una acción conocida que entregue una recompensa máxima. Para sacar máximo provecho de la exploración al inicio del experimento para luego explotar incrementalmente, el valor de ε en el sistema disminuirá conforme al número de pasos.

4.2.2 Módulo de animación

Este módulo genera una animación de labios siguiendo un audio como entrada, a esto también se le conoce como sincronización de labios. Las entradas de este módulo son:

1. Los pesos de la RNC, previamente entrenada.
2. Un archivo de audio en formato WAV a 16khz con monólogo en español.

La salida del módulo es una secuencia de imágenes (30 imágenes por cada segundo de audio de entrada) que representan el movimiento de los labios generado a partir de las acciones del agente.

A continuación, se describe la forma en que se deben generar las imágenes:

1. Inicializar el Manejador de Labios que funciona como una interfaz para que el agente manipule la posición de unos labios virtuales, la posición inicial de los labios es cerrada.
2. Cargar el audio en memoria. Inicializar variable indice_audio para apuntar al elemento del audio en la posición 16000/30, redondeando hacia abajo.
3. Tomar el segmento de audio con inicio en [Índice Audio - 16000*0.30] y final en Índice Audio y alimentar la RNC con este segmento como entrada, para obtener la acción que maximiza la recompensa.
4. Tomar la acción que maximiza la recompensa, generar una imagen con la nueva posición de labios.
5. Índice Audio <= Índice Audio + 16000*0.30
6. Ir a paso 3 hasta procesar todo el audio

Al combinar las imágenes generadas en los pasos descritos anteriormente con el audio de entrada, podemos producir un video con sincronización de labios.

4.3 Implementación

Ahora se describe la implementación desarrollo del sistema. El lenguaje de programación para llevar a cabo la mayoría del desarrollo es Python. El intérprete de Python utilizado para el desarrollo y pruebas es Anaconda versión 3 (para Python 3), y el entorno de desarrollo es Visual Studio Code, el cual soporta múltiples extensiones que ayudan en el desarrollo con Python. Las extensiones de Visual Studio Code utilizadas fueron:

- Graphviz – Generación de diagramas UML a partir de código Python
- Python –Soporte de Python para VS Code, incluye:
- Pylance – Soporte de Python para VS Code.
- Linting (sugerencias de código).
- Depuración.
- Jupyter – notebooks de Jupyter.
- Python docstring Generator – Asistente de generación de documentación para funciones Python.

Tomando en base la metodología TDD, se define la estructura de las carpetas del proyecto, ilustrado en la figura 4.5.

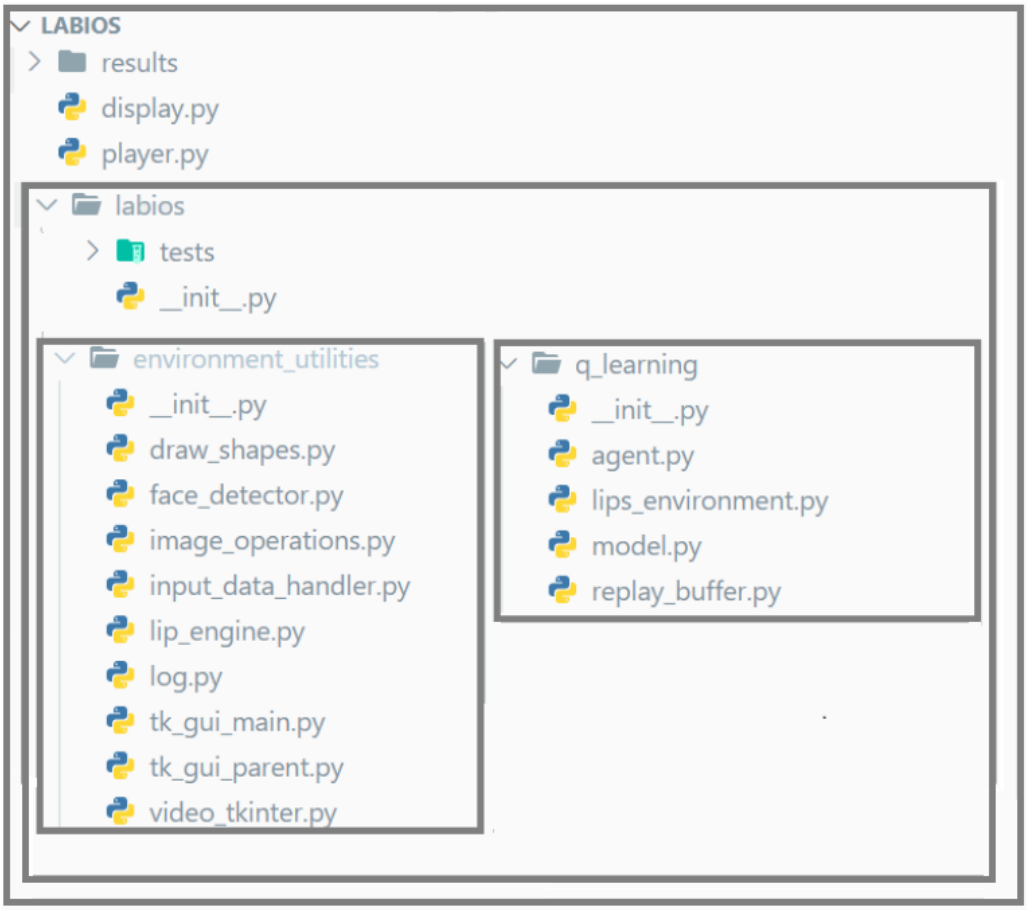

Figura 4.3 Diseño de la estructura del proyecto.

La estructura consta de una carpeta principal que contiene todos los archivos del proyecto. Ahora se detallan cada una de estas partes.

En la raíz se tiene una carpeta para guardar resultados de los experimentos. También en la raíz se encuentra el script para entrenar al agente, el cual puede ser configurado con distintos parámetros para realizar experimentos y además de otro script para generar las animaciones el cual se trata más adelante.

En la raíz está la carpeta labios que tiene tres subcarpetas:

1. **q_learning**: contiene todas las clases relacionadas con el Aprendizaje de Máquina a alto nivel, esto incluye clase de agente, ambiente, modelo y buffer de memoria.

2. **environment_utilities**: es una colección de librerías que auxilian a ambiente y otras clases del sistema.
3. **tests**: contiene las pruebas para todas las clases dentro de las otras dos subcarpetas.

La carpeta de labios en raíz tiene archivos __init__.py en todas sus subcarpetas, lo que le indica al intérprete de Python que es un paquete. Esto nos permite hacer referencias relativas entre las clases del paquete.

Los archivos de acceso que se encuentran en la raíz, train.py y display.py no se consideran paquetes, ya que son el punto de entrada al sistema y deben poder ejecutarse, lo cual no es posible para un paquete de Python.

Ahora se describe el código del sistema, en orden según la estructura de carpetas definida anteriormente. Como se ve en la figura 4.3, el sistema consta de dos módulos principales. En el más alto nivel tenemos el módulo de aprendizaje representado por el script de entrenamiento, llamado train.py, que es el punto de entrada para ejecutar los experimentos de aprendizaje. Por el otro lado, el módulo de animación se representa por el script display.py (ver figura 4.5).

4.3.1 Entrenamiento

El script de train.py, al ser llamado se encarga de entrenar al agente, creando una instancia de éste y controlando la interacción y el aprendizaje por medio de la configuración contenida en el mismo script. Primero, la inicialización de los parámetros se lleva a cabo creando una instancia de la clase *Player* pasando la configuración. Luego, secuencialmente se manda a llamar al método step de la instancia *Player* para hacer al agente tomar una acción en el ambiente y recibir retroalimentación, esto se realiza secuencialmente hasta llegar al número de pasos de entrenamiento deseado.

A continuación, en la figura 4.6 se muestra el código para el entrenamiento el cual está basado en el código de Tessera (2019) que es una implementación de Volodymyr (2015).

Archivo: train.py

```
1.     class Player():
2.         """Player class to train agent or showcase training
3.         """
4.         def __init__(
5.                     if os.path.isfile(self.check_point_file):
6.                 self.eps_start= 0.01
7.             else:
8.                 self.eps_start= 1
9.             # set default hyperparameters
10.            self.hyper_params = {
11.                "seed": 42,  # which seed to use
12.                "replay-buffer-size": int(5e3),  # replay buffer size
13.                "learning-rate": 1e-3,  # learning rate for Adam optimizer
14.                "discount-factor": 0.0, #0.99,  # discount factor
15.                # total number of steps to run the environment for
16.                "num-steps": int(1e6),
17.                "batch-size": 32,  # number of transitions to optimize at the same
time
18.                "learning-starts": 60,  # number of steps before learning starts
19.                 "learning-freq":  1,   #  number  of  iterations  between  every
optimization step
20.                "use-double-dqn": True,  # use double deep Q-learning
21.                "target-update-freq": 1000,  # number of iterations between every
target network update
22.                "eps-start": self.eps_start,  # e-greedy start threshold
23.                "eps-end": 0.01,  # e-greedy end threshold
24.                "eps-fraction": 0.1,  # fraction of num-steps
25.                "print-freq": 1,
26.                "obs-duration": 0.03, # audio observation in seconds
27.                "dqn-type": "DQNMedium"
28.                 obs_type   =   "wave+pos"   if   "pos"   in   self.hyper_params["dqn-
type"].lower() else "wave"
29.            self.env = LipsEnvironment(
30.                lip_data_source = f'{vid_source}.npy',
31.                obs_duration= self.hyper_params["obs-duration"],
32.                obs_type = obs_type
33.                )
34.            self.env.seed(self.hyper_params["seed"])
35.

36.             self.replay_buffer   =   ReplayBuffer(self.hyper_params["replay-buffer-
size"])

37.            self.agent = DQNAgent(
38.                self.env.observation_space,
39.                self.env.action_space,
40.                self.replay_buffer,
41.                use_double_dqn=self.hyper_params["use-double-dqn"],
42.                lr=self.hyper_params['learning-rate'],
43.                batch_size=self.hyper_params['batch-size'],
44.                gamma=self.hyper_params['discount-factor'],
45.                  device=torch.device("cuda"   if   torch.cuda.is_available()   else
"cpu"),
46.                dqn_type=self.hyper_params['dqn-type']#'DQNMedium'
```

```
47.             )
48.             print('agent initialized.')

49.             if os.path.isfile(self.check_point_file):
50.                 print(f"Loading a policy - { self.check_point_file } ")
51.                 # config = torch.load(self.check_point_file)
52.                 self.agent.policy_network.load_state_dict(
53.                     torch.load(self.check_point_file))

54.             self.eps_timesteps = self.hyper_params["eps-fraction"] * \
55.                 float(self.hyper_params["num-steps"])
56.             self.episode_rewards = [0.0]
57.             self.episode_q_values = [0.0]
58.             self.episode_loss = [0.0]
59.             self.step_stats = []

60.             self.state = self.env.reset()
61.             self.reward = None
62.             self.done = None
63.             self.info = None
64.             print('environment reset done. Start...')
65.             self.t = 0
66.             self.done = False

67.             self.total_steps = 0
68.             self.max_steps = self.hyper_params["num-steps"]

69.         def step(self):
70.             """ Make agent take 1 step on the environment and train the NN module
71.

72.             Returns:
73.                 boolean: True if currently in final step in episode, False otherwise
74.             """
75.             t = self.t
76.             self.t += 1
77.
78.             fraction = min(1.0, float(t) / self.eps_timesteps)
79.             eps_threshold = self.hyper_params["eps-start"] + fraction * \
80.                 (self.hyper_params["eps-end"] - self.hyper_params["eps-start"])
81.             sample = random.random()

82.             q_value = 0.0
83.             if sample > eps_threshold or not self.learn:
84.                 # Exploit
85.                 action,q_value = self.agent.act(self.state)
86.             else:
87.                 # Explore
88.                 action = self.env.action_space.sample()
89.
90.             if  "pos" in self.agent.dqn_type.lower():
91.                     next_state,     self.reward,     self.done,     self.info     =
self.env.step(action)
92.                 state_tmp = self.state["state1"]
93.                 state_pos_tmp = self.state["state2"]
```

```
94.                 next_state_tmp = self.state["state1"]
95.                 next_state_pos_tmp = self.state["state2"]
96.
97.                 self.agent.memory.add_pos(state_tmp, state_pos_tmp,
98.                      action,self.reward,next_state_tmp,next_state_pos_tmp,float(sel
f.done))
99.                 self.state = next_state
100.            else:
101.                    next_state,     self.reward,     self.done,     self.info     =
self.env.step(action)
102.                self.agent.memory.add(self.state, action, self.reward, next_state,
float(self.done))
103.            self.state = next_state
104.            if self.done:
105.                self.state = self.env.reset()
106.            step_loss = 0
107.            if self.learn:
108.                if (t > self.hyper_params["learning-starts"] and
109.                    self.t % self.hyper_params["learning-freq"] == 0):
110.                    step_loss = self.agent.optimise_td_loss()

111.                if (t > self.hyper_params["learning-starts"] and
112.                    self.t % self.hyper_params["target-update-freq"] == 0):
113.                    self.agent.update_target_network()

114.            self.episode_loss[-1] += step_loss
115.            step_stats = [self.reward,step_loss,q_value]
116.                                                self.step_stats.append(step_stats)

117.            num_episodes = len(self.episode_rewards)

118.            if (self.learn and self.done and self.hyper_params["print-freq"] is not
None and len(self.episode_rewards) % self.hyper_params["print-freq"] == 0):
119.

120.             mean_100ep_reward = round(np.mean(self.episode_rewards[-101:-1]), 1)
121.             return self.done
```

Figura 4.4 Código del script de entrenamiento

Las líneas 4 a la 68 de la figura 4.6, corresponden a la inicialización de la clase Player. Aquí se crea una instancia del agente y ambiente con los parámetros correspondiente.

Después, de la línea 69 a 121 se encuentra la función *step* que al ser llamada lleva a cabo la interacción del agente con el ambiente y según los parámetros, toma acciones exploratorias o explotadoras (líneas 83-88) y entrena al agente con el buffer de memoria (líneas 107-113).

Para comenzar a entrenar al agente, se instancia el Player y se itera en un ciclo para llamar a la función *player.step()* por el número de pasos que se desee iterar.

La clase de Player durante su inicialización crea instancias de las clases de Agente y Ambiente. La primera contiene un modelo de DQN, cuya arquitectura se ilustra en la figura 4.2 y se implementa como lo muestra la figura 4.7. El modelo consta de una RNC (front-end) que se define en las líneas 1 a 27 de la figura 4.7 y una RNCC (back-end), líneas 28 a 33 y que juntas forman el modelo de DQN.

La clase de Agente cuenta con cuatro funciones:

1. __init__ : Recibe como entrada los hiper parámetros del experimento, crea dos instancias del modelo de DQN: target la cual se utiliza para entrenar y policy en la cual se basa el sistema para la toma decisiones, e inicializa el optimizador de la DQN.
2. optimise_td_loss : Se llama durante el entrenamiento, toma muestras del buffer de memoria de replay descrito en 4.2.1 y los utiliza para entrenar a la DQN.
3. update_target_network : Refleja el entrenamiento de la DQN target en la DQN policy.
4. act : Toma como entrada un estado del ambiente y regresa la mejor acción a tomar en dicho estado en base a la predicción de la DQN.

La parte de RNC de la DQN consta de un conjunto de cuatro capas compuestas cada una de la siguiente manera:

1. Capas de convolución: reducen datos, aplicando la operación de convolución con filtros progresivamente más grandes.
2. ReLU: aplica una fnción de activación rectificada: max(0, x).
3. Segunda capa de convolución.
4. Segunda función ReLU.
5. Agrupamiento (MaxPool): reduce la información.
6. Dropout: desactiva un porcentaje de las salidas de cada capa para evitar sobre entrenamiento.

Archivo: model.py - función de inicialización del modelo

```
1.     self.conv =
2.     nn.Sequential(
3.     nn.Conv1d(in_channels=1, out_channels=16, kernel_size=9, stride=1),
4.     nn.ReLU(),
```

```
    nn.Conv1d(in_channels=16, out_channels=16, kernel_size=9, stride=1),
    nn.ReLU(),
    nn.MaxPool1d(kernel_size = 16,stride = 1,padding = 0),
    nn.Dropout(p=0.1, inplace=False),
    nn.Conv1d(in_channels=16, out_channels=32, kernel_size=3, stride=1),
    nn.ReLU(),
    nn.Conv1d(in_channels=32, out_channels=32, kernel_size=3, stride=1),
    nn.ReLU(),
    nn.MaxPool1d(kernel_size = 4,stride = 1,padding = 0),
    nn.Dropout(p=0.1, inplace=False),
    nn.Conv1d(in_channels=32, out_channels=32, kernel_size=3, stride=1),
    nn.ReLU(),
    nn.Conv1d(in_channels=32, out_channels=32, kernel_size=3, stride=1),
    nn.ReLU(),
    nn.MaxPool1d(kernel_size = 4,stride = 1,padding = 0),
    nn.Dropout(p=0.1, inplace=False),
    nn.Conv1d(in_channels=32, out_channels=256, kernel_size=3, stride=1),
    nn.ReLU(),
    nn.Conv1d(in_channels=256, out_channels=256, kernel_size=3, stride=1),
    nn.ReLU(),
    nn.MaxPool1d(1551),
    nn.Dropout(p=0.2, inplace=False)
            )
    #back end
    self.fc = nn.Sequential(
    nn.Linear(in_features=256 + 3 , out_features=512),
    nn.ReLU(),
    nn.Linear(in_features=512,                    out_features=action_space.n))
```

Figura 4.5 Implementación de la RNC

Este modelo de DQN está encapsulado en una clase CNNPos la cual es una clase hija de nn.Module, el módulo de la librería pytorch para redes neuronales. La clase CNNPos tiene además de inicialización, una única función forward (pase hacia enfrente) para generar las predicciones de la red tomando como entrada el estado observado, el código se muestra en la figura 4.8. Nótese que la función forward recibe una sola entrada que representa al estado que se observó en el ambiente y es un diccionario. Después, se divide la entrada en dos variables: la primera corresponde a la señal de audio y la segunda a la posición de labios del Avatar digital. Con esto el agente estima qué recompensa podría ganar tomando cada una de las acciones posibles con el audio y posición de labios del momento y genera un valor numérico para cada acción posible que indica la recompensa potencial para cada acción. Esto se hace a través de la función forward.

Así pues, la función forward toma la observación del ambiente (estado) y alimenta la primera parte de este que consta únicamente de la señal de audio a la RNC. La RNC produce un vector de características que describen a la señal de audio de forma reducida. Luego a este vector se le

concatena la segunda parte del estado que se forma de parámetros que describen la posición de labios del Avatar.

Este nuevo vector compuesto de las características reducidas del audio y las posiciones de los labios del Avatar se alimenta después a la RNCC la cual produce un valor para cada acción que puede tomar el agente que indica la recompensa que espera recibir al tomar dicha acción.

Archivo: model.py - función forward

```
1.          def forward(self, d):
2.              x = d["state1"]
3.              x2 = d["state2"]
4.              conv_out = self.conv(x)
5.              conv_out_flat = torch.squeeze(conv_out)
6.              if(len(conv_out_flat.shape) == 1):
7.                  conv_out_flat = conv_out_flat.unsqueeze(0)#add minibatch dim
8.              if(len(x2.shape)==1):
9.                  x2 = x2.unsqueeze(0)
10.                     combined = torch.cat((conv_out_flat,x2),dim=1)
11.             out = self.fc(combined)
12.             return out
```

Figura 4.6 Función forward del modelo de DQN.

Enseguida, en el aprendizaje tenemos la clase de Ambiente que se forma de las tres clases ilustradas por el diagrama UML en la figura 4.9. La clase LipsEnv, que es la clase principal y de mayor abstracción de las tres representa un ambiente en el contexto del RL, es decir, se encarga a alto nivel de proveer una instancia del ambiente, su estado y señales de recompensa en cada paso, dependiendo de las acciones del agente.

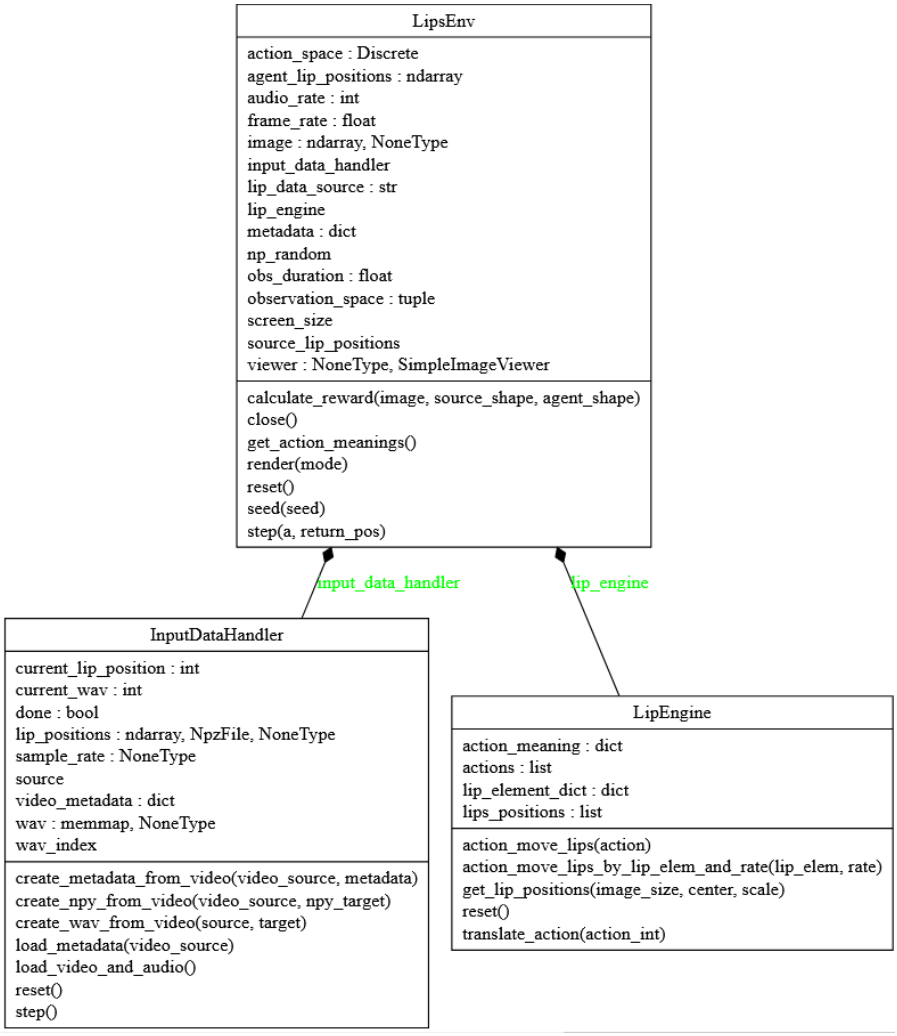

Figura 4.7 UML de las clases del ambiente

Para la implementación de la clase LipsEnv se utilizó la librería de Gym de OpenAI que provee una plataforma para la creación de diversos elementos para problemas de RL en general, LipsEnv es una clase hija de la clase de ambiente de Gym de OpenAI: gym.Env. El código de la inicialización de esta clase se ve en la figura 4.10, se puede ver que la clase Ambiente contiene instancias de las clases LipEngine e InputDataHandler, las cuales utiliza más adelante durante los experimentos, particularmente al llamar a la función LipsEnv.step. El código de las clases InputDataHandler se puede ver en el Anexo A.

Archivo lip_environment.py - LipsEnv.__init__()

```
1.  import gym
2.  from gym import error, spaces
3.  from gym.utils import seeding
4.  from ..environment_utilities import lip_engine
5.  from ..environment_utilities import input_data_handler as idh
6.  from ..environment_utilities import draw_shapes
7.  from ..environment_utilities.image_operations import center_shape_on_pivot
8.  import numpy as np
9.  class LipsEnv(gym.Env):
10.     """
11.     Lips Environment which handles the lips' position, audio, reward
12.     """
13.     metadata = {'render.modes': ['human', 'lip_pos','image']}
14.     def __init__(
15.             self,
16.             lip_data_source = 'test_vid.mp4',
17.             frame_rate = 29.94,
18.             audio_rate = 16000,
19.             repeat_action_probability = 0.,
20.             obs_type = 'wave',
21.             obs_duration = .5
22.             ):
23.         self.lip_data_source = lip_data_source
24.
      self.frame_rate = frame_rate
25.         self.viewer = None
26.         self.audio_rate = audio_rate
27.         self.obs_duration = obs_duration
28.         self.seed()
29.
30.         self.lip_engine = lip_engine.LipEngine()
31.         self._action_set = self.lip_engine.actions
32.         self.action_space = spaces.Discrete(len(self._action_set))
33.
34.         self._obs_type = obs_type
35.         self.input_data_handler = idh.InputDataHandler(lip_data_source)
36.                                   self.screen_size                                =
self.input_data_handler.video_metadata['resolution']
37.         self.image = None
38.         # Set observation space based on audio, lip positions
39.         if self._obs_type == 'wave':
40.             sample_size = self.audio_rate * self.obs_duration
41.             lip_positions_len = len(self.lip_engine.lips_positions)
42.             self.observation_space = int(sample_size)
43.         elif self._obs_type == 'wave+pos':
44.             sample_size = self.audio_rate * self.obs_duration
45.             lip_positions_len = len(self.lip_engine.lips_positions)
```

```
46.                 self.observation_space = (int(sample_size),lip_positions_len)
47.             else:
48.                     raise      error.Error('Unrecognized      observation      type:
{}'.format(self._obs_type))
49.         …
```

Figura 4.8 Función de inicialización de la clase LipsEnv.

Luego, la clase LipsEnv implementa un número de funciones de su clase padre *gym.Env*, la principal de ellas es *step* la cual es llamada por *Player* cuando hace al agente interactuar con el ambiente.

Esta función *step*, la cual se puede ver en la figura 4.11, recibe como entrada la acción elegida por el agente y regresa el nuevo estado que presenta el ambiente a partir de la acción de entrada, una señal de recompensa que indica qué tan cercanas son las posiciones de los labios de Avatar y el locutor, una bandera que indica si el episodio ha terminado y un mensaje con información adicional.

Archivo lip_environment.py - LipsEnv.step()

```
1.     def step(self, a, return_pos = False):
2.         reward = 0.0
3.         action = self._action_set[a]
4.
5.         self.lip_engine.action_move_lips(action)
6.         self.input_data_handler.step()
7.                                 self.source_lip_positions                          =
self.input_data_handler.lip_positions[self.input_data_handler.current_lip_position]
8.         self.agent_lip_positions = self.lip_engine.get_lip_positions(
9.             self.screen_size,center=None,scale=0.8)
10.
11.         if self.image is None:
12.             self.image = np.zeros(self.screen_size,np.int8)
13.                 composite_reward         =         self.calculate_reward(self.image,
self.source_lip_positions,self.agent_lip_positions)
14.         reward = 0
15.         for d in composite_reward:
16.             reward-= d
17.         obs = self._get_obs()
18.         done = self.input_data_handler.done
19.                                             info                                   =
f'comp_rwd:u{composite_reward[0]},l{composite_reward[1]},c{composite_reward[2]}'
20.         return obs, reward, done, info
```

Figura 4.9 Función step de la clase LipsEnv.

La clase InputDataHandler o IDH tiene como función principal administrar los datos del video y derivados que utiliza el ambiente para crear sus estados y señales de recompensa.

Al cargar un video, el IDH verifica que existan archivos para ese video con las posiciones de labios extraídas y en formato npy así como también un archivo de audio en formato wav extraído del video.

De no existir estos archivos, la clase IDH los crea a partir de un conjunto de funciones internas que funcionan de la siguiente manera:

1. Teniendo la ruta a un archivo de video mp4 en almacenamiento, verificar si existen archivos con formato npy y wav con el mismo nombre.
2. **Extracción de posiciones de los labios**: Si no existe un archivo npy con el nombre del video, utilizar la clase FaceDetector para extraer secuencialmente las posiciones de cada cuadro del video y guardarlas en un arreglo de numpy y un archivo con el mismo nombre del video en disco.
3. **Extracción de audio**: Si no existe un archivo wav con el mismo nombre que el video, manda a llamar a la librería ffmpeg con los parámetros para extraer el audio del video mp4 a 16kHz. Los parámetros son:
    - -i : video del cual extraer audio
    - -ab 160k : bit rate (bps)
    - -ac 1 : utilizar un solo canal de audio
    - -ar 16000 : frecuencia de muestreo
    - -vn : nuevo archivo de audio

    Por ejemplo: ffmpeg -i vid.mp4 -ab 160k -ac 1 -ar 16000 -vn audio.wav

Durante el entrenamiento, esta clase mantiene un apuntador al cuadro de video durante cada paso del episodio, LipsEnv obtiene las posiciones de los labios del locutor y el segmento de audio correspondiente a cada posición a través de las funciones de esta clase.

Otra clase del ambiente es LipEngine. Esta clase mantiene los labios virtuales del Avatar, los cuales pueden ser manipulados mediante acciones que toma el agente. La clase LipsEnv envía las acciones tomadas por el agente en cada paso a una instancia propia de LipEngine (ver figura 4.11, línea 5) el cual efectúa la acción sobre los labios y mantiene la posición resultada de la serie de acciones tomadas por el agente desde el principio de cada episodio.

Al inicializarse LipsEnv, éste crea a su vez una instancia de LipsEngine, el cual carga en memoria los parámetros de los labios virtuales con la configuración inicial (figura 4.12, líneas 5-28).

Cada episodio, LipsEnv llama a la función action_move_lips de IDH pasando el índice de la acción efectuada por el agente. Esta función busca en un diccionario interno llamado action_meaning que contiene el significado de las acciones, el cual es aplicado a los labios utilizando funciones internas (figura 4.12 - líneas 34-37, 61-73 y 38-60, respectivamente).

Cabe resaltar que la clase LipEngine solo guarda tres variables de punto flotante que describen la posición de la boca. La posición exacta de cada uno de los puntos de la boca se calcula cada vez que son necesarios, esto se aprecia en la función LipEngine.get_lip_positions en la figura 4.12, líneas 74-112.

Archivo lip_engine.py

```
1.  from decimal import Decimal
2.  import numpy as np
3.  class LipEngine:
4.  """ This module serves as an interface for the agent to move the lips """
5.  def __init__(self):
6.      """Initialize LipEngine instance, default lip position is closed"""
7.      commissure_pos = 0.7
8.      upper_lip_pos = 0.0
9.      lower_lip_pos = 0.0
10.     self.lips_positions = [commissure_pos,upper_lip_pos,lower_lip_pos]
11.     self.action_meaning = {
12.         0:  "NOOP",
13.         1:  "LO1",
14.         2:  "LC1",
15.         3:  "CO1",
16.         4: "CC1",
17.         5: "CO2-LO2",
18.         6: "CC1-LC2",
19.         7: "UO1-LO3",
20.         8: "UO1-LO3-CO1",
21.         9: "UC1-LC3-CC1"
```

```
22.         }
23.         self.actions = list(self.action_meaning.keys())
24.         self.lip_element_dict = {
25.                 'C':0,
26.                 'U':1,
27.                 'L':2
28.             }
29.     def reset(self):
30.         commissure_pos = 0.7
31.         upper_lip_pos = 0.0
32.         lower_lip_pos = 0.0
33.         self.lips_positions = [commissure_pos,upper_lip_pos,lower_lip_pos]
34.     def action_move_lips(self,action):
35.         lip_elements,rates = self.translate_action(action)
36.         for i in range(len(lip_elements)):
37.             self.action_move_lips_by_lip_elem_and_rate(lip_elements[i],rates[i])
38.     def translate_action(self,action_int = 0):
39.         """ translate action_int to the parameters necessary for
40.             action_move_lips_by_lip_elem_and_rate using action_meaning
41.             returns actions,rates
42.         """
43.         lip_elements = []
44.         rates = []
45.         if self.action_meaning[action_int] == "NOOP":
46.             return lip_elements,rates
47.         # one action_int may act on more than one lip element
48.         actions = str(self.action_meaning[action_int]).split(sep='-')
49.         for individual_action in actions:
50.             lip_elem = self.lip_element_dict[individual_action[0]]
51.             rate = Decimal(individual_action[2]) * Decimal(0.05)
52.             if individual_action[1] == 'O':
53.                 rate *= 1
54.             elif individual_action[1] == 'C':
55.                 rate *= -1
56.             else:
57.                 raise Exception("Movement direction not valid.")
58.             lip_elements.append(lip_elem)
59.             rates.append(rate)
60.         return lip_elements,rates
61.     def action_move_lips_by_lip_elem_and_rate(self,lip_elem = None,rate = None):
62.         # self.lips_positions = [commissure_pos,upper_lip_pos,lower_lip_pos]
63.         new_lips_positions = self.lips_positions
64.         if lip_elem is None:
65.             return
66.         elif rate is None:
67.             raise Exception('Error:rate is not provided')
68.           new_lips_positions[lip_elem]   =   round(new_lips_positions[lip_elem]   +
float(rate),2)
```

```
69.         if new_lips_positions[lip_elem] > 1.0:
70.             new_lips_positions[lip_elem] = 1.0
71.         elif new_lips_positions[lip_elem] < 0.0:
72.             new_lips_positions[lip_elem] = 0.0
73.         self.lips_positions[lip_elem] = new_lips_positions[lip_elem]
74.     def get_lip_positions(self,image_size,center=None,scale=1.0):
75.         shape = np.zeros((68,2))
76.         image = np.zeros(image_size)
77.         """ draw lips from 3 arguments for commissures, upper and lower lips from
lip_engine class"""
78.         lip_par = self.lips_positions
79.         if center is None:
80.             center_x = int(image.shape[1] / 2)
81.             center_y = int(image.shape[0] / 2) + 40
82.         else:
83.             center_x = center[0]
84.             center_y = center[1]
85.         max_commissures = int(60.0 * scale)
86.         max_upper = int(20.0 * scale)
87.         max_lower = int(40.0 * scale)
88.         #lip commissures calculation
89.         left_commissure_x_pos = center_x - int((lip_par[0] * max_commissures) /
2.0)
90.         right_commissure_x_pos = center_x + int((lip_par[0] * max_commissures) /
2.0)
91.         commissures_y_pos = center_y
92.         #calculate upper and lower lips limits
93.         upper_lip_y = int(center_y - (max_upper * lip_par[1]))
94.         lower_lip_y = int(center_y + (max_lower * lip_par[2]))
95.         horizontal_diameter = right_commissure_x_pos - left_commissure_x_pos
96.         vertical_diameter = lower_lip_y - upper_lip_y
97.
98.         #upper lip
99.                                         shape[49]                                   =
[left_commissure_x_pos,commissures_y_pos]                        #left commissure
100.        shape[40] = [center_x - int(horizontal_diameter * 0.2),upper_lip_y - int(1.0
* scale)]
101.        shape[51] = [center_x - int(horizontal_diameter * 0.1) ,upper_lip_y - int(3.0
* scale)]
102.             shape[52]      =      [center_x,upper_lip_y      -      int(1.0      *
scale)]                       #center upper lip position
103.        shape[53] = [center_x + int(horizontal_diameter * 0.1) ,upper_lip_y - int(3.0
* scale)]
104.        shape[54] = [center_x + int(horizontal_diameter * 0.2),upper_lip_y - int(1.0
* scale)]
105.              shape[55]        =        [right_commissure_x_pos,commissures_y_pos
]                   #right commissure
106.        #lower lip
```

```
107.        shape[56] = [center_x + int(horizontal_diameter * 0.3),lower_lip_y -
int(vertical_diameter * 0.225)]
108.        shape[57] = [center_x + int(horizontal_diameter * 0.15) ,lower_lip_y -
int(vertical_diameter * 0.0125)]
109.       shape[58] = [center_x , lower_lip_y]    #center lower lip position
110.        shape[59] = [center_x - int(horizontal_diameter * 0.15) ,lower_lip_y -
int(vertical_diameter * 0.0125)]
111.        shape[60] = [center_x - int(horizontal_diameter * 0.3),lower_lip_y -
int(vertical_diameter * 0.225)]
112.       return shape
```

Figura 4.10 Clase LipEngine

4.3.2 Animación

Como se menciona previamente, el sistema tiene dos tareas principales que se pueden resumir de la siguiente manera:

Comenzando por el primer cuadro del video, uno a uno se alimentan las posiciones de labios y audio correspondiente. Con esto, el módulo de aprendizaje hace modificaciones internas a sus propios parámetros de forma que aprende a predecir las acciones que aproximan más cercanamente la posición de labios del locutor.

Una vez llevado a cabo el aprendizaje, el módulo de animación hace uso de la experiencia del módulo de aprendizaje para generar una animación de labios con audio distinto al utilizado en el entrenamiento.

Ahora se describe la implementación de la segunda de estas tareas.

La animación de los labios se lleva a cabo por una serie de clases y librerías a cuya agrupación se le llama módulo de animación. El punto de entrada al módulo de animación es la clase PlayGUI que se encuentra en el archivo display.py en la raíz del proyecto y genera una interfaz como se muestra en la figura 4.13.

Figura 4.11 Interfaz gráfica del módulo de animación.

La clase *GUIParent* (figura 4.14) contiene funciones para dibujar en pantalla una interfaz de usuario con dos o tres cuadros sobre los cuales la clase hija puede mostrar animaciones. La interfaz gráfica es creada utilizando la librería de tkinter de Python, la cual contiene funciones para el manejo de interfaces gráficas. El diseño de la interfaz es modular con secciones delimitadas por marcos (frames), los cuales contienen paneles cuyas clases hijas insertan las imágenes a mostrar (ver figura 4.14, líneas 6-31).

Archivo tk_gui_parent.py

```
1.     import tkinter as tk
2.     import labios.environment_utilities.video_tkinter as util
3.     from labios.environment_utilities.log import log
4.     class GUIParent:
5.         """ Tkinter Graphic user interface to display source video and lip trace
"""
6.         def __init__(self, main_window):
7.             log('initializing MyGUI')
8.             self.fps = None
9.             self.wait_time = -1  # In order to get it from source video's fps, set
to any number
10.            self.agent = None
11.            self.face_detector = None
12.            self.render_pixels_width = 0
13.            self.render_pixels_height = 0
14.            self.refresh_thread = None
15.            self.source_image = None
```

```
16.             self.main_window = main_window
17.             # Setup Frames
18.             self.source_frame = tk.LabelFrame(
19.                     main_window,borderwidth=2,relief = tk.RAISED,text='Source Video
/ Render')
20.             self.render_frame = tk.LabelFrame(
21.                     main_window,borderwidth=2,relief = tk.RAISED,text='Rendered')
22.             self.compare_frame = tk.LabelFrame(
23.                     main_window,borderwidth=2,relief = tk.RAISED,text='Compare')
24.             self.controls_frame = tk.LabelFrame(
25.                     main_window,borderwidth=2,relief = tk.RAISED,text='Controls')
26.             #self.render_frame.pack(anchor = "e", fill = 'both')
27.             self.controls_frame.pack(anchor = "s", fill = 'both')
28.             self.src_panel = None
29.             self.render_panel = None
30.             self.compare_panel = None
31.             log("MyGUI parent init completed")
32.           def    draw_panel_and_pack(self,panel,parent_frame,image,panel_side    =
None,frame_side = None):
33.             """Update the frame/image of the current panel and initialize the panel
and parent frame
34.             if it has not been initialized yet, then pack both panel and frame to
given side."""
35.             image = util.convert_image_for_display(image)
36.             if panel is None:
37.                   log(str.format('initializing    panel    to    {},    frame    to
{}',panel_side,frame_side))
38.                 panel = tk.Label(parent_frame, image=image)
39.                 panel.image = image
40.                 if panel_side is None:
41.                     panel.pack(fill = 'both')
42.                 else:
43.                     panel.pack(fill = 'both', side = panel_side)
44.                 if frame_side is not None:
45.                     parent_frame.pack(side = frame_side)#fill = 'both'
46.             else:
47.                 panel.configure(image=image)
48.                 panel.image = image
49.             return panel,parent_frame
50.         def video_init(self,fps):
51.              """ get frames per second of source video and update wait_time
accordingly"""
52.             if self.wait_time is not None:
53.                 self.fps = fps
54.                 log(str.format('updated fps: {}',self.fps))
55.                 self.wait_time = int((1 / self.fps) * 1000)
56.                 self.wait_time =int(self.wait_time / 2)
57.                 log(str.format('updated wait time:{}', self.wait_time))
```

```
58.             else:
59.                 self.wait_time = 1
60.                 self.fps = 0
61.             log('playing source video....')
62.             #set image width and height for the lip render and comparison
63.             self.render_pixels_width = self.source_image.shape[1]
64.             self.render_pixels_height = self.source_image.shape[0]
```

Figura 4.12 Código de la clase GUIParent.

PlayGUI (figura 4.15) hereda las funciones y métodos de GUIParent y además implementa funciones para dibujar en los cuadros creados por su clase padre: los puntos de la cara extraídos del video original, los puntos de los labios del Avatar digital sobre los cuales actúa el agente descrito anteriormente y opcionalmente un tercer cuadro con ambas imágenes combinadas. Las animaciones son generadas con un hilo que ejecuta la función de PlayGUI refresh_video_display (figura 4.15 líneas 19-81) donde se consume el audio sobre el cual el agente previamente entrenado se basará para tomar las decisiones de movimiento de los labios. Y luego, se generarán imágenes secuencialmente para cada segmento de audio con la duración seleccionada, actualizando las imágenes en base a las acciones tomadas por el agente.

Las imágenes son generadas dibujando puntos en las coordenadas correspondientes en una imagen mediante la librería de OpenCV, utilizando la función cv2.circle.

Archivo display.py

```
1.     import tkinter as tk
2.     import threading
3.     import cv2
4.     import numpy as np
5.     import tk_gui_parent
6.     import labios.environment_utilities.draw_shapes as draw_shapes
7.     from labios.environment_utilities.log import log
8.     from player import Player
9.     VIDEO_SOURCE = r'\vids\train_vid'
10.    class PlayGUI(tk_gui_parent.GUIParent):
11.        """ Tkinter Graphic user interface to display source video and lip trace
"""
12.        def __init__(self,main_window):
13.            super().__init__(main_window)
14.            # Start source video refresh/streaming thread
15.            self.refresh_thread = threading.Thread(
16.                    target=self.refresh_video_display,
```

```
17.                     daemon = False)
18.             self.refresh_thread.start()
19.         def refresh_video_display(self):
20.             """ Top level function to process and display the frames of the video
on tkinter GUI. """
21.             log('refresh thread started','info')
22.             log('creating capture object')
23.             check_point_file = 'checkpoint.pth'
24.             hp = {
25.                 "dqn-type" : "DQNPos",
26.                 "obs-duration":0.1,
27.                 "eps-start": 0.0,  # e-greedy start threshold
28.                 "eps-end": 0.0  # e-greedy end threshold
29.                 }
30.             player = Player(
31.                 vid_source = VIDEO_SOURCE,
32.                 check_point_file=check_point_file,
33.                 hyper_params= hp)
34.             self.source_image = None
35.             video_played = False
36.             counter = 0
37.             while not player.done:
38.                 player.step()
39.                 # try:
40.                 # _, frame = cap.read()
41.                 source_image= np.zeros(
42.                           (player.env.screen_size[0],player.env.screen_size[1],3),
np.uint8)
43.                 agent_image = np.zeros(
44.                           (player.env.screen_size[0],player.env.screen_size[1],3),
np.uint8)
45.                                                 source_image                                =
draw_shapes.draw_shape(source_image,np.array(player.env.source_lip_positions,np.int16
))
46.                                                  agent_image                                =
draw_shapes.draw_shape(agent_image,np.array(player.env.agent_lip_positions,np.int16))

47.                 if not video_played:
48.                     self.source_image = np.zeros(
                    (player.env.screen_size[0],player.env.screen_size[1],3),
              np.uint8)
49.                     self.video_init(30)
50.                     video_played = True
51.                 trace_img = cv2.addWeighted(agent_image,0.5,source_image,0.8,0)
52.                 counter +=1
53.                 cv2.putText(agent_image,'Agent',(10,50), cv2.FONT_HERSHEY_SIMPLEX,
0.5,(255,255,255),1,cv2.LINE_AA)
```

```
54.                 cv2.putText(trace_img,'Compare',(10,50), cv2.FONT_HERSHEY_SIMPLEX,
1,(255,255,255),1,cv2.LINE_AA)
55.                 cv2.putText(trace_img,player.info,(10,100),cv2.FONT_HERSHEY_SIMPLE
X, 1,(255,255,255),1,cv2.LINE_AA)
56.                                         cv2.putText(source_image,'Source',(10,50),
cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255),1,cv2.LINE_AA)
57.                 self.src_panel,self.source_frame = self.draw_panel_and_pack(panel =
self.src_panel,
58.                                     parent_frame = self.source_frame,
59.                                     image = source_image,
60.                                     panel_side = 'left')
61.                 self.render_panel, self.render_frame = self.draw_panel_and_pack(
62.                     panel = self.render_panel,
63.                     parent_frame = self.source_frame,
64.                     image = agent_image,
65.                     panel_side = 'right',
66.                     frame_side = 'top')
67.                 self.compare_panel, self.compare_frame = self.draw_panel_and_pack(
68.                     panel = self.compare_panel,
69.                     parent_frame = self.compare_frame,
70.                     image = trace_img,
71.                     panel_side = 'right',
72.                     frame_side = 'bottom')
73.                 # handle frame rate
74.                 if cv2.waitKey(self.wait_time) & 0xff == ord('q'):
75.                     break
76.             if not video_played:
77.                 log('could not open video')
78.             log('total frames:' + str(counter))
79.             log('releasing video capture')
80.             # cap.release()
81.             cv2.destroyAllWindows()
```

Figura 4.13 Código de la clase PlayGUI.

El módulo de animación puede utilizarse con un video como fuente, lo que genera una animación del Avatar con los puntos del locutor original como comparación.

También se puede tomar como entrada únicamente un audio, en cuyo caso el resultado es únicamente la animación del Avatar.

4.4 Pruebas

Como se definió en la sección 4.2 - Diseño, se utilizó una metodología de desarrollo guiado por pruebas (TDD, por sus siglas en inglés: Test Driven Development). Las pruebas son fundamentales para el desarrollo, ya que primero se crean las pruebas y luego se implementa el código mínimo necesario para pasar esa prueba. Esto quiere decir que las pruebas necesitan de la robustez necesaria para asegurar que el código cumple con los requerimientos.

Por la naturaleza del proyecto se prevé que sean necesarios cambios a los algoritmos de aprendizaje y detección de labios/audio incluso en etapas avanzadas, por lo cual el tener pruebas automatizadas que midan el desempeño del proyecto durante cada cambio es importante.

Esta metodología permitirá crear pruebas automatizadas desde el comienzo del proyecto y poder realizar cambios de manera rápida sin sacrificar la calidad, así como evaluar el desempeño de manera sencilla.

Para correr las pruebas se hace uso del marco de trabajo Pytest, el cual facilita la creación y ejecución de pruebas de distintos tamaños. Este se puede instalar como librería de Python utilizando el siguiente comando en consola, teniendo Python instalado: pip install pytest.

Para ejecutar las pruebas basta con mandar a llamar el comando “pytest” en consola, desde la carpeta del proyecto. De esta forma, pytest busca automáticamente los archivos que contienen las pruebas y los ejecuta, mostrando los resultados de las pruebas en consola.

Las pruebas son contenidas en un directorio especial que refleja la estructura de los directorios del resto del sistema. Cabe recordar que la estructura del sistema se ve como muestra la figura 4.16, así que la carpeta test contiene también una carpeta labios con subcarpetas de q_learning y environment_utilities.

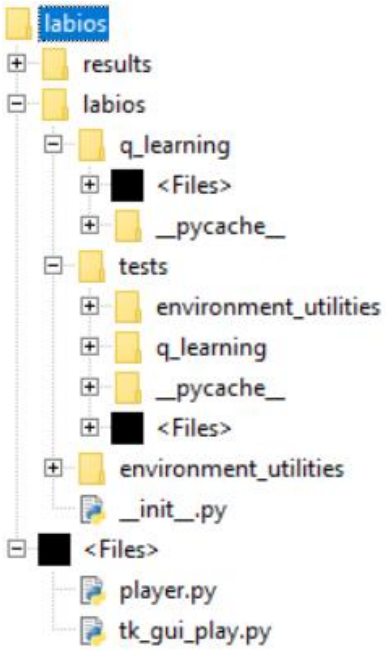

Figura 4.14 Estructura del proyecto.

En estas carpetas, existe un archivo de prueba para cada uno de los archivos python con el mismo nombre y el prefijo “test_”. Por ejemplo, el archivo environment_utilities/face_detector.py tiene un archivo de pruebas test/environment_utilities/test_face_detector.py, como lo muestra la figura 4.17, y así es también para los archivos de la subcarpeta q_learning.

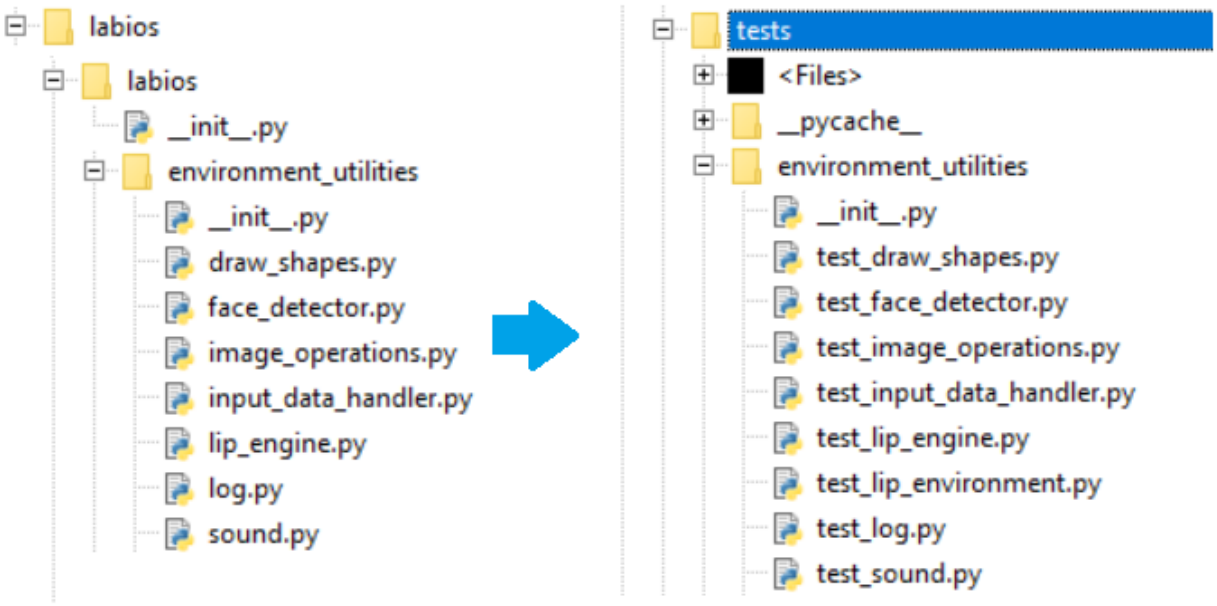

Figura 4.15 Correspondencia de archivos de prueba.

Cada uno de los archivos en la subcarpeta test, contiene las pruebas creadas el archivo correspondiente.

Ahora se describe la forma de crear las pruebas con algunos ejemplos seleccionados. No se incluye la totalidad de las pruebas del proyecto, pues son de gran extensión. Durante desarrollo del sistema, de forma iterativa se elige uno de los requerimientos definidos previamente, luego se crea una prueba en el archivo correspondiente en la subcarpeta test, tomando en cuenta el nombre del archivo donde se creará el código que deberá pasar esa prueba. Un ejemplo serían los requerimientos descritos en la sección 4.1.1, que se refieren al procesamiento de video para extracción de la posición de labios y audio, específicamente el párrafo:

"*Para cada cuadro del video, se extraen las posiciones de puntos clave de los labios en forma de coordenadas de un plano cartesiano o similar, luego, tomando como referencia la línea recta entre la base y la punta de la nariz hay que rotar la cara de forma que la línea mencionada se encuentre paralela al eje de las y. Por último, se acoplan las posiciones con el segmento correspondiente de audio (de 10 a 500 milisegundos).*" (4.1.1 Identificación de fonemas y visemas).

El requerimiento anterior dice que el sistema debe extraer las posiciones de la cara del locutor, contenidas en el video de entrada, para después rotarla con la finalidad de que quede alineada.

En base a este requerimiento, se deduce que hay que crear una función llamada center_shape_on_pivot que tome como entrada una serie de puntos que representan la posición de los elementos de la cara del locutor, un pivote y un punto de referencia, y regrese otra serie de puntos rotados en base al pivote y el punto de referencia, para que la cara se encuentre recta. El pivote será la punta de la nariz y el punto de referencia la base de la nariz.

Para lograr esto, esta función puede calcular el ángulo que se debe girar la base de la nariz para alinearse a la punta y después llamar a otra función con nombre rotate_around_point que, tomando el pivote y el ángulo de giro, mueva a cada uno de los puntos debidamente.

Posteriormente se crea un archivo en la carpeta de labios/labios/environment_utilities de nombre image_operations.py donde se implementarán esta y otras operaciones que se deban hacer sobre imágenes. Pero primero se crea la prueba y archivo contenedor: test_image_operations.py en la ruta de labios/test/environment_utilities, esta prueba verifica primero que una función llamada rotate_around_point (que aún no se implementa al momento de crear la prueba), al llamarla con las coordenadas de una combinación de puntos y un ángulo de giro, regrese los valores esperados, esta verificación se lleva a cabo con la palabra reservada de python: “assert” (ver figura 4.18).

Archivo test_image_operations.py - test_rotate_around_point

```
1.    def test_rotate_around_point():
2.        assert rotate_around_point(xy = (10,10),radians = math.radians(90),
3.            origin = (0,0)) == (10,-10)
4.
5.        assert rotate_around_point(
6.            xy = ((100,100)),radians = math.radians(180),
7.            origin = (90,90)) == (80,80)
```

Figura 4.16 Pruebas para alineación de puntos.

Como se puede ver en la figura 4.18, cada prueba se encuentra en un archivo con el mismo nombre que el archivo que prueba más el prefijo *test*. Dentro de cada archivo de pruebas se crean funciones que clasifican a las pruebas por la función a probar.

En cada una de las funciones de prueba, se agregan comprobaciones "*assert*" donde se manda a llamar al módulo o función y se compara con el resultado esperado. Se pueden agregar tantas comprobaciones como se crea necesario para cada uno de los requerimientos a probar.

Lo siguiente es crear las pruebas para la función *center_shape_on_pivot*, que consiste en generar unas posiciones de cara artificiales como se hace en la prueba de la figura 4.18 y compararlas con los valores esperados mediante una serie de comprobaciones con la palabra *assert*, de manera similar a la función *rotate_around_point*.

Otro requerimiento es mover la cara del locutor al centro de la imagen, esta prueba (ver figura 4.18) requiere como entrada las posiciones de la cara para luego centrarlas. Ya que se espera que todos los puntos se desplacen de forma similar, basta con prestar atención en un par de puntos y comprobar el desplazamiento del resto de puntos sea similar. La figura 4.19 demuestra cómo se pueden crear datos artificiales antes de utilizar las comprobaciones.

Archivo test_image_operations.py - test_center_shape_on_image

```
1.      def test_center_shape_on_image():
2.          image = np.zeros((1000,1000,3), np.uint8)
3.          nose_bridge = 27
4.          nose_tip = 30
5.          shape = np.zeros((68,2),np.int16)
6.          shape[nose_tip] = [500,500]
7.          shape[nose_bridge] = [550,450]
8.          shape[0] = [400,400]
9.          shape[1] = [400,600]
10.         shape[2] = [600,400]
11.         shape[3] = [600,600]
12.         shape = shape - 5
13.         new_shape = np.zeros(shape.shape,dtype = np.int16)
14.         i = 0
15.         for x,y in shape:
16.             new_shape[i] = rotate_around_point((x,y),10,(500,500))
17.             i += 1
18.         shape = new_shape
19.         rotated_shape = center_shape_on_image(image,shape)
20.         …
...
```

Figura 4.17 Pruebas para centrado de cara.

El resto de los archivos del programa tienen pruebas creadas utilizando estos principios.

CAPÍTULO V. RESULTADOS Y DISCUSIÓN

Los experimentos realizados consisten en el entrenamiento del modelo con un video de 32 minutos de una persona leyendo un texto de frente a la cámara. El entrenamiento consistió en 33 episodios o 1 millón de pasos para cada experimento.

Para todos los experimentos se usó el optimizador de RMSProp, disponible en la librería de pytorch. El sistema utiliza un valor de ε que disminuye conforme avanza el entrenamiento y también se experimentó con distintos valores de ε constantes. En la figura 5.1 se puede observar la recompensa promedio por episodio para 33 episodios de entrenamiento con diferentes valores de α (ritmo de aprendizaje) y un valor de ε fijo. Mientras que en la figura 5.2 se observa la pérdida correspondiente, la cual se busca disminuir con el entrenamiento.

Para la primera serie de experimentos, se toma como entrada únicamente el audio del locutor. Para esto, se parte de la premisa de que la DQN será capaz de extraer la información importante para el aprendizaje dado un segmento de audio adecuado y suficiente entrenamiento. Se buscará observar si la DQN es capaz de determinar la acción correspondiente sin esa información.

En la segunda serie de experimentos se alimenta la DQN con el audio adicionando las posiciones de los labios del Avatar en la parte de la RNCC, esperando obtener mejores resultados ya que las posiciones de los labios y el audio describen de manera más completa el estado del ambiente, lo que debería brindar una ventaja al agente.

Como punto de comparación se toma la recompensa obtenida por un agente sin movimiento que mantiene una posición de boca cerrada, a este agente se le denomina agente flojo. Con el agente flojo se obtiene un nivel base de recompensas cuando no se hacen acciones y permitirá hacer comparaciones con los modelos que sí realicen acciones. En la figura 5.1 se puede comparar la recompensa para la primera serie de experimentos sin posición de labios a través de 33 episodios con diferentes valores de α, el agente busca maximizar la recompensa a través del aprendizaje, explorando diferentes políticas, por lo que se espera que la recompensa por episodio vaya claramente en aumento con el pasar de los episodios, lo que no sucede aquí.

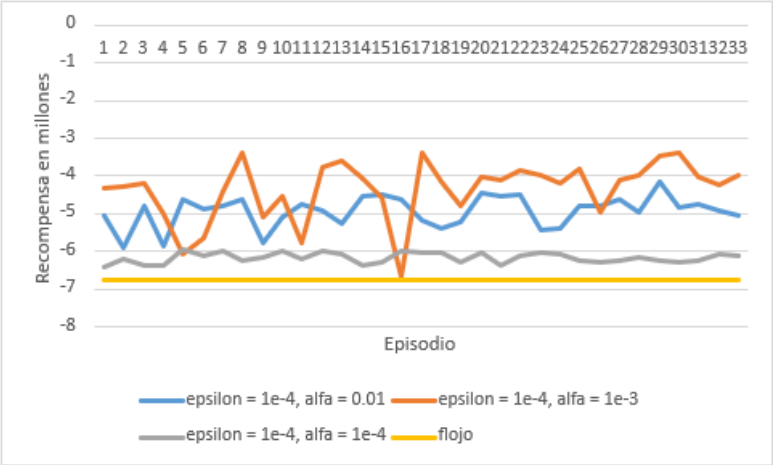

Figura 5.1 Recompensa por episodio, con ε = 1e-4 y diferentes valores de α.

Otra forma de medir el aprendizaje de la RNC es por la pérdida promedio durante un número de pasos. La pérdida se puede describir como una medida del error en las predicciones de la RNC por lo que se busca que disminuya con el entrenamiento. En la Fig. 5.2 se puede observar la pérdida por episodio del sistema para diferentes valores de α para la primera serie de experimentos con solo audio como entrada. Una pérdida que disminuye a través de los episodios indica que la red va mejorando en sus predicciones.

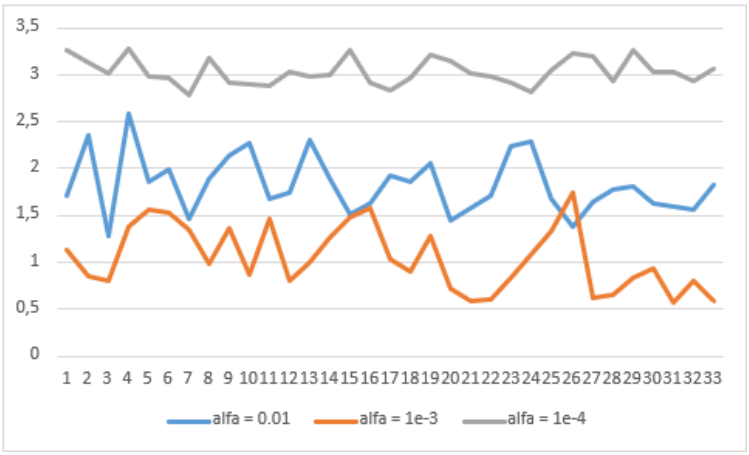

Figura 5.2 Recompensa por episodio, con ε = 1e-4 y diferentes valores de α.

Como se puede ver, la DQN que recibe como entrada únicamente el audio del locutor, no logra converger en un óptimo. Esto es debido a que no tiene suficiente información del ambiente, es decir, el audio no representa por si solo con suficiente exactitud el estado del ambiente.

Al visualizar la recompensa para los últimos pasos del primer episodio, en la figura 5.3, se puede observar que, a pesar de que el agente es capaz de estimar los cambios burdamente en la recompensa por episodio, no puede discriminar los cambios más pequeños, lo que resulta en un movimiento de labios impreciso.

Figura 5.3 Recompensa por paso, con ε = 1e-4 y diferentes valores de α.

Si bien los experimentos donde se toma como entrada únicamente el audio proveniente del locutor no presentan resultados alentadores, la segunda serie de experimentos presenta mejores resultados. Como se observa en la figura 5.4, el agente que toma como entrada el audio más las posiciones de la boca del Avatar logra un rendimiento más alto que su contraparte que toma únicamente audio. Aquí, se puede visualizar de mejor manera el aprendizaje a través de los episodios. La recompensa supera desde el primer episodio a la obtenida por la primera serie de experimentos, de -3M vs -1.4M. Cabe recordar que la recompensa indica qué tan cerca sigue el Avatar a la posición original del locutor, a mayor recompensa, o en este caso entre más se acerca a cero.

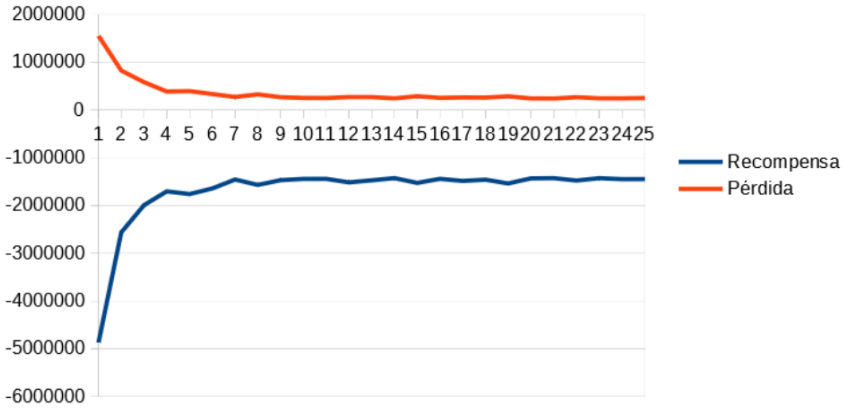

Figura 5.4 Recompensa y pérdida con audio y posiciones.

CAPÍTULO VI. CONCLUSIONES

A partir de los experimentos realizados se obtiene un movimiento de labios que se asemejan a los del locutor original, demostrando con ello que funciona el Aprendizaje por Refuerzo. A partir de la recompensa total por episodio de varios agentes y comparándola con los resultados de un agente flojo, se puede ver que el sistema de Aprendizaje por Refuerzo obtiene mayor recompensa cuando tiene como estado una composición del audio y las posiciones del agente. Se observa que la animación sigue de forma general los labios del locutor.

Se planea comparar el rendimiento del sistema con el de otros trabajos que utilizan distintas técnicas y plataformas, por el momento la métrica de comparación es la recompensa promedio por episodio, lo cual permite comparar el aprendizaje de distintos modelos que utilizan el mismo entorno.

Otro plan consiste en probar la capacidad de generalización de la técnica de aprendizaje por refuerzo aquí aplicada a la sincronización de labios en comparación con otras técnicas como las de aprendizaje supervisado.

Para este trabajo se limitó el entrenamiento a 33 episodios, pero se puede dar tanto tiempo como el agente requiera para quedar completamente entrenado.

Una variante probada limitadamente hasta el momento es la duración del audio que recibe el agente en cada paso, para este trabajo fue de 300, 100 y 30 milisegundos. Se podría experimentar con ventanas de audio con diferente duración buscando mejorar los resultados.

Los mejores resultados fueron obtenidos al utilizar la salida de la última serie de capas de convolución y de agrupamiento de la red, después sumando los parámetros que describen las posiciones de los labios del agente, para finalmente alimentar este conjunto a las capas finales de la red. Este acercamiento provee una imagen más clara del aprendizaje del agente a través de los episodios.

Adicionalmente, se sugiere crear un modelo similar que reciba el audio en forma de espectrograma, esto daría la facilidad de poder usar modelos adicionales probados en aplicaciones similares tal como en el trabajo de Wyse (2017).

CAPÍTULO VII. BIBLIOGRAFÍA

About OpenCV (2019). Obtenido de https://opencv.org/about/

Albawi, S., Mohammed A. (2017). Understanding of a Convolutional Neural Network. 10.1109/ICEngTechnol.2017.8308186.

Aneja D., Li W. (2019): Real-Time Lip Sync for Live 2D Animation. ArXiv 1910.08685v1.

Aytar, Y., Vondrick, C., & Torralba, A. (2016). SoundNet: Learning Sound Representations from Unlabeled Video. Retrieved 6 30, 2021, Obtenido de http://papers.nips.cc/paper/6146-soundnet-learning-sound-representations-from-unlabeled-video.pdf

Beck K. (2002). Test-Driven Development by Example. Vaseem: Addison Wesley. ISBN 978-0-321-14653-3.

Branson R. (2015). What Makes WAV Better than MP3. https://videoconversiontools. wordpress.com/2015/10/21/what-makes-wav-better-than-mp3/.

Cabiedes F., Pelczer I., Gamboa F., Bretón J. & Rodríguez S. (2007), Sincronización de labios: método sin visemas. Revista Iberoamericana de Educación a Distancia 10 (1), pp. 37-50.4

El-Tantawy S. & Abdulhai B. (2010). Towards multi-agent reinforcement learning for integrated network of optimal traffic controllers (MARLIN-OTC). Transportation Letters: The International Journal of Transportation Research. 2. 89-110. 10.3328/TL.2010.02.02.89-110.

Fan B., Xie L., Yang S., Wang L & Soong F. K. (2015): A deep bidirectional LSTM approach for viddeo-realistic talking head. Multimedia Tools and Applications archive 75(9), pp. 5287-5309.

Goodfellow I., Bengio Y., & Courville A. (2016). Deep Learning. MIT Press, http://www.deeplearningbook.org

Hasan T. & Ahmed S. (2014). Human Lips-Contour Recognition and Tracing. International Journal of Advanced Research in Artificial Intelligence. 3. 10.14569/IJARAI.2014.030107.

Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W. (1996). "Reinforcement Learning: A Survey". Journal of Artificial Intelligence Research. 4: 237–285. arXiv:cs/9605103. doi:10.1613/jair.301.

Kaleab Tessera (2019). Deep Q-network implementation for Pong-vo. https://github.com/KaleabTessera/DQN-Atari

Karras T., Aila T., Laine S., Herva A., Lehtinen J. (2017): Audio-driven facial animation by joint end-to-end learning of pose and emotion. ACM Transactions on Graphics, 36 (4), Artículo 94.

Kazemi V., Sullivan J. (2014): One Millisecond Face Alignment with an Ensemble of Regression Trees, IEEE Conference on Computer Vision and Pattern Recognition.

Lawrence L (2000). Code and Other Laws of Cyberspace. Basic Books. ISBN 0-465-03913-8

LeCun Y., Bengio Y. & Hinton 1G. (2015): Deep Learning, Nature 521, pp. 436-444.

Liévin M., Delmas P., Coulon P., Luthon F & Fristot V. (1999) Automatic lip tracking: Bayesian segmentation and active contours in a cooperative scheme. HAL CCSD

Mansar Y. (2018): Audio Classification: A Convolutional Neural Network Approach, https://medium.com.

Martin R. (2008). Clean Code: A Handbook of Agile Software Craftmanship

Nyquist H. (1928). Certain Topics in Telegraph Transmission Theory.

Rosebrock A. (2017) Detect eyes, nose, lips, and jaw with dlib, OpenCV, and Python. Obtenido de https://www.pyimagesearch.com/2017/04/10/detect-eyes-nose-lips-jaw-dlib-opencv-python/

Scopatz, Anthony; Huff, Kathryn D. (2015). Effective Computation in Physics. O'Reilly Media, Inc. p. 351. ISBN 9781491901595.

Shannon C. E. (1949). Communication in the presence of noise.

Shawl S. (2020). Epsilon-Greedy Algorithm in Reinforcement Learning. Obtenido de https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/

Sutton S. & Barto G. (2018): Reinforcement Learning: An Introduction, Francis Bach, pp. 3, 6-7, 223.

Suwajanakorn S., Seitz S. M. & Kemelmacher-Shlizerman I. (2017): Synthesizing Obama: Learning Lip Sync from Audio. ACM Trans. Graph. 36, 4, Articulo 95.

Van Hasselt H., Guez A. & Silver D. (2016). Deep Reinforcement Learning with Double Q-learning. Vol. 30 No. 1: Thirtieth AAAI Conference on Artificial Intelligence .

Van Rossum G., Warsaw B., Coghlan N. (2001). PEP 8 -- Style Guide for Python Code. Obtenido de https://www.python.org/dev/peps/pep-0008/

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, Demis Hassabis. (2015): Human-level control through deep reinforcement learning, Nature 518, pp. 529–533.

Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, Samarjit Das. (2017): A very deep convolutional neural networks for raw waveforms, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

What is ReLu? (2021). Obtenido de https://deepai.org/machine-learning-glossary-and-terms/relu.

Wyse L. (2017): Audio Spectrogram Representations for Processing with Convolutional Neural Networks, Proceedings of the First International Workshop on Deep Learning and Music joint with IJCNN. 1(1). pp 37-4.

Xu Y., Feng A. W., Marsella S., Shapiro A. (2013): A Practical and Configurable Lip Sync Method for Games. MIG '13 Proceedings of Motion on Games, pp 131-140.

ANEXO A

Archivo input_data_handler.py

```
1.    import pickle
2.    import os.path
4.    import subprocess
5.    import numpy as np
6.
7.    from scipy.io import wavfile
8.
12.   from . import image_operations as img_ops
15.   from .log import progress_bar,log
16.
17.   class InputDataHandler():
18.       def __init__(
19.           self,
20.           source: str):
21.           """ Load lip positions and audio file into memory
22.           Parameters:
23.               @source: source of the input data, can be any of:
24.                   mp4: process mp4 into npy and wav files
25.                   npy: process npy and wav files directly
26.                   wav: take only audio, use once agent has been trained.
27.
28.           """
29.           self.source = source
30.           # self.override = override
31.
32.           self.lip_positions = None
33.           self.wav = None
34.           self.sample_rate = None
35.


36.           self.video_metadata = {}
37.           self.load_video_and_audio()
38.           self.wav = self.wav / 256
39.           self.wav_index = self.wav.shape[0] / self.lip_positions.shape[0]
40.


41.           self.current_lip_position = 0
42.           self.current_wav = 0
43.


44.           self.done = False
45.


46.       def load_video_and_audio(self):
47.           """ Set """
48.           source_type = self.source.split('.')[-1]
49.
```

```
50.             source_npy = self.source[:-4] + '.npy'
51.             source_wav = self.source[:-4] + '.wav'
52.
53.             # Check if files exist or should be created based on parameters
54.             if not os.path.isfile(self.source):
55.                 raise Exception('Source file does not exist:'+ self.source)
56.             else:
57.                 if source_type == 'mp4':
58.                     if os.path.isfile(source_npy):
59.                         # self.lip_positions =  np.fromfile(source_npy,dtype=i
nt,count=-1,sep='',offset=0)
60.                         self.lip_positions = np.load(source_npy)
61.                         self.video_metadata = self.load_metadata(self.source)
62.                     else: #create npy from video file
63.                         self.lip_positions = self.create_npy_from_video(self.s
ource,source_npy)
64.                     if os.path.isfile(source_wav):
65.                         self.sample_rate, self.wav = wavfile.read(source_wav)
66.                     else:
67.                         self.sample_rate,self.wav = self.create_wav_from_video
(self.source,source_wav)
68.                 elif source_type == 'npy':
69.                     if not os.path.isfile(source_wav) or not os.path.isfile(so
urce_npy):
70.                         raise Exception("Error: wav/npy file does not exists,
try calling init with source='mp4'.")
71.                     # self.lip_positions = np.fromfile(source_npy)
72.                     self.lip_positions = np.load(source_npy)
73.                     self.video_metadata = self.load_metadata(self.source)
74.                     self.sample_rate,self.wav = wavfile.read(source_wav)
75.                 elif source_type == 'wav':
76.                     self.sample_rate, self.wav = wavfile.read(source_wav)
77.                     lip_len = 30 * len(self.wav) / self.sample_rate
78.                     self.lip_positions = np.zeros((lip_len,68,2))
79.                     self.video_metadata = {'resolution': (500,500)}
80.                 else:
81.                     raise Exception("Error: source type not valid.")
82.


83.         def create_wav_from_video(self, source,target):
84.             """ Create wav file from video and load it to memory"""
85.             log('creating wav file from video')
86.             log('\tSource:'+  source)
87.
88.             log('\tTarget:' + target)
89.
95.             command = 'ffmpeg -i '+source+' -ab 160k -ac 1 -ar 16000 -
vn '+ target
96.             log('\tCalling ffmpeg subprocess:'+ command)
97.
98.             subprocess.call(command)
100.            samplerate, data = wavfile.read(target)
```

```
101.            return samplerate,data
102.
103.        def create_npy_from_video(self,video_source,npy_target):
104.            from .face_detector import FaceDetector
105.            import cv2
106.            cap = cv2.VideoCapture(video_source)
107.
108.            # try:
109.            video_played = False
110.
111.            self.video_metadata['total_frames'] = cap.get(cv2.CAP_PROP_FRAME_C
OUNT)
112.
113.            log(f"Creating npy file from video. Video contains {self.video_met
adata['total_frames']} frames")
114.            lip_positions = []
115.            face_detector = FaceDetector()
116.
117.            angle = 90
118.            center = None
119.            rotation_matrix = None
120.
121.            progress_items = list(range(0, int(self.video_metadata['total_fram
es'])))
122.
123.            # while cap.isOpened():
124.            for progress_items in progress_bar(progress_items, prefix = 'Progr
ess:', suffix = 'Complete', length = 50):
125.                if not cap.isOpened():
126.                    break
127.
128.                _, source_image = cap.read()
129.                source_image = cv2.cvtColor(source_image, cv2.COLOR_BGR2GRAY)
130.                source_image = img_ops.resize_image(source_image)
131.
132.                if not video_played:
133.                    # cv2.CAP_PROP_FPS -> this is how to get the video fps
134.                    (h, w) = source_image.shape[:2]
135.                    center = (int(w /2), int(h /2))
136.                    rotation_matrix = cv2.getRotationMatrix2D(center, angle, s
cale = 1.0)
137.                    self.video_metadata['resolution'] = source_image.shape[:2]
138.                    video_played = True
139.
140.                # source_image = cv2.warpAffine(source_image, rotation_matrix,
 (w, h))
141.
142.                shape = face_detector.get_features_from_image(source_image)
143.                centered_shape = img_ops.center_shape_on_image(source_image,sh
ape)
```

```
144.
145.                 lip_positions.append(centered_shape)
146.
147.             if not video_played:
148.                 log('could not open video')
149.             #except Exception as exception:
150.                 #log('Error:'+ str(e) + ' | ' + type(e).str)
151.                 #raise exception
152.             #finally:
153.                 # Para terminar, soltar la captura
154.             self.create_metadata_from_video(video_source,self.video_metadata)
155.
156.             log('releasing video capture')
157.             cap.release()
158.
159.             np_lip_positions = np.array(lip_positions)
160.             #np_lip_positions.save(npy_target)
161.             np.save(npy_target,np_lip_positions)
162.             return np_lip_positions
163.
164.         def create_metadata_from_video(self,video_source,metadata):
165.             with open(video_source + '.pickle', 'wb') as f:
166.                 pickle.dump(metadata, f, pickle.HIGHEST_PROTOCOL)
167.
168.         def load_metadata(self,video_source):
169.             with open(video_source + '.pickle', 'rb') as f:
170.                 return pickle.load(f)
171.
172.         def reset(self):
173.             self.current_lip_position = 0
174.             self.done = False
175.
176.         def step(self):
177.             self.current_lip_position += 1
178.
180.             wav_end_index = int(self.wav_index * self.current_lip_position)
181.             # if wav_start_index >=0:
182.             self.current_wav = wav_end_index
183.             # else:
184.             #     self.currenet_wav
185.             if self.current_lip_position >= self.lip_positions.shape[0] -1:
186.                 self.done = True
187.
```