```
import tkinter as tk
from tkinter import filedialog,
messagebox, scrolledtext
import sounddevice as sd
import scipy.io.wavfile as wavfile
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.animation as
animation
import pyttsx3  # For text-to-
speech
import json  # For saving and
loading learning data
from scipy.fft import fft  # For
```

```
Fourier Transform
from sklearn.cluster import
KMeans  # For clustering
from OpenGL.GL import *
from OpenGL.GLU import *
import pygame  # For handling
3D rendering

# Define constants
SAMPLE_RATE = 44100
DURATION = 3.33
PHI = (1 + np.sqrt(5)) / 2  #
Golden Ratio
SPEED_OF_LIGHT = 299792458
# Speed of light in m/s
G = 6.67430e-11  # Gravitational
```

```
constant in m^3 kg^-1 s^-2
rho_d = 0.3e-26  # Density of
dark matter in kg/m^3 (example
value)

# Load or initialize the learning
data (frequency-symbol
mappings)
try:
    with open('learning_data.json',
'r') as f:
        LEARNING_DATA =
json.load(f)
except FileNotFoundError:
    LEARNING_DATA = {}
```

```
# Light language dictionary (expandable)
LIGHT_LANGUAGE_DICT = {
    # Example mappings
    'A': 'Activation', 'G': 'Gateway', 'S': 'Spiritual', 'a': 'Awakening',
    'l': 'Love', 'j': 'Joy', 'g': 'Gratitude', 'D': 'Divine guidance'
}

# Function to decode light language
def decode_light_language(message):
    return '
```

```
'.join(LIGHT_LANGUAGE_DICT.get(char, '') for char in message).strip()

# Function to calculate energy based on mass using E = mc^2
def mass_energy_equivalence(mass):
    return mass * (SPEED_OF_LIGHT ** 2)

# Function to calculate dark matter potential energy
def dark_matter_energy(mass, distance):
```

```
    """Calculates potential energy
contribution from dark matter."""
    if distance <= 0:
        raise ValueError("Distance
must be positive")
    return -G * mass * rho_d /
distance  # Simplified for
demonstration

# Total energy calculation
considering dark matter
def
mass_energy_with_dark(mass,
distance):
    """Calculates total energy
considering dark matter
```

```
influence."""
    energy =
mass_energy_equivalence(mass)
    dark_energy =
dark_matter_energy(mass,
distance)
    return energy + dark_energy

# Frequency generation
def generate_wave(frequency,
duration):
    """Generates a sine wave
signal for a given frequency and
duration."""
    t = np.linspace(0, duration,
int(SAMPLE_RATE * duration),
```

```
endpoint=False)
    return 0.5 * np.sin(2 * np.pi *
frequency * t)

# Learning and updating the
database
def
update_learning_data(frequency,
symbol):
    """Updates the learning data
with a new frequency-symbol
mapping."""
    mass = 1  # Example mass
    distance = 1  # Example
distance for dark matter
contribution
```

```
    total_energy =
mass_energy_with_dark(mass,
distance)

LEARNING_DATA[str(frequency)]
= {
        "symbol": symbol,
        "energy": total_energy
    }
    save_learning_data()

def save_learning_data():
    """Saves the learning data to a
file."""
    with open('learning_data.json',
'w') as f:
```

```
json.dump(LEARNING_DATA, f)

# Learning from video frames
def process_video_file(notepad):
    video_file =
filedialog.askopenfilename(filetypes=[("Video Files", "*.mp4")])
    if not video_file:
        return
    cap =
cv2.VideoCapture(video_file)
    if not cap.isOpened():
        raise ValueError("Failed to
open video file")
```

```
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        gray = cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY)
        brightest_pixel =
np.max(gray)
        frequency = 100 +
(brightest_pixel / 255) * 2000  #
Map brightness to frequency
        symbol = chr(65 +
len(LEARNING_DATA))  #
Generate new symbol

update_learning_data(frequency,
```

```
symbol)
        notepad.insert(tk.END,
f"Learned: Frequency
{frequency}, Symbol: {symbol}\n")

    cap.release()

# 3D Flight Simulation
def draw_spaceship():
    """Draw a simple spaceship
model."""
    glBegin(GL_TRIANGLES)
    glColor3f(1, 0, 0)  # Red color
    glVertex3f(0, 1, 0)
    glVertex3f(-1, -1, 1)
    glVertex3f(1, -1, 1)
```

```
    glVertex3f(0, 1, 0)
    glVertex3f(1, -1, 1)
    glVertex3f(1, -1, -1)

    glVertex3f(0, 1, 0)
    glVertex3f(1, -1, -1)
    glVertex3f(-1, -1, -1)

    glVertex3f(0, 1, 0)
    glVertex3f(-1, -1, -1)
    glVertex3f(-1, -1, 1)
    glEnd()

def
draw_frequencies(frequencies):
```

```
    """Draw frequency symbols in
the environment."""
    for freq in frequencies:
        glPushMatrix()
        glTranslatef(freq, 0, 0)  #
Position based on frequency
        glBegin(GL_QUADS)
        glColor3f(0, 1, 0)  # Green
color for frequency symbols
        glVertex3f(-0.1, 0, -0.1)
        glVertex3f(0.1, 0, -0.1)
        glVertex3f(0.1, 0, 0.1)
        glVertex3f(-0.1, 0, 0.1)
        glEnd()
        glPopMatrix()
```

```
def main_flight_simulation():
    """Main function for the flight
simulation."""
    pygame.init()
    display = (800, 600)

pygame.display.set_mode(displa
y, DOUBLEBUF | OPENGL)
    gluPerspective(45, (display[0]
/ display[1]), 0.1, 50.0)
    glTranslatef(0.0, 0.0, -5)

    x_move = y_move = z_move = 0
    clock = pygame.time.Clock()

    while True:
```

```
        for event in
pygame.event.get():
            if event.type ==
pygame.QUIT:
                pygame.quit()
                return

        keys =
pygame.key.get_pressed()
        if keys[pygame.K_a]:
            x_move -= 0.1
        if keys[pygame.K_d]:
            x_move += 0.1
        if keys[pygame.K_w]:
            z_move += 0.1
        if keys[pygame.K_s]:
```

```
            z_move -= 0.1
        if keys[pygame.K_SPACE]:
            y_move += 0.1
        if keys[pygame.K_LSHIFT]:
            y_move -= 0.1

        glTranslatef(x_move,
y_move, z_move)

glClear(GL_COLOR_BUFFER_BIT
| GL_DEPTH_BUFFER_BIT)

        draw_spaceship()  # Draw
the spaceship

draw_frequencies(list(LEARNING
```

```
_DATA.keys()))  # Draw
frequency symbols

        pygame.display.flip()
        clock.tick(60)

# GUI setup
def create_gui():
    root = tk.Tk()
    root.title("Infinite Frequency
Database")

    notepad_frame =
tk.Frame(root)
    notepad_frame.pack(pady=10)
    tk.Label(notepad_frame,
```

```
text="Learned Symbols
Notepad:").pack()

    notepad =
scrolledtext.ScrolledText(notepa
d_frame, width=40, height=10)
    notepad.pack()

    tk.Button(root, text="Process
Video File", command=lambda:
process_video_file(notepad)).pac
k(pady=10)
    tk.Button(root, text="Simulate
Flight",
command=main_flight_simulatio
n).pack(pady=10)
```

```
    root.mainloop()

# Entry point
if __name__ == "__main__":
    create_gui()
```