**PROJECT DEVELOPED BY:**

* PARTH PATEL - 19DCS098
* SAUMYA SHAH - 19DCS133
* SHRUTI PATEL - 19DCS098

**MINI PROJECT SOURCE CODE:**

# THE PROJECT IS ABOUT A PERSONAL ASSISTANT THAT CAN HELP IN PARTICULAR TASKS

import pyttsx3

#pyttsx3 is a text-to-speech conversion library in Python

import speech\_recognition as sr

#Library for performing speech recognition

import webbrowser

#The webbrowser module provides a high-level interface to allow displaying web-based documents to user

from GoogleNews import GoogleNews

#Google News search for Python

from playsound import playsound

#The playsound module is a cross platform module that can play audio files

import yagmail

#yagmail is a GMAIL/SMTP client that aims to make it as simple as possible to send emails.

import pygame as pg,sys

#Pygame is a cross-platform set of Python modules designed for writing video games.

from pygame.locals import \*

import time

from math import inf as infinity

from random import choice

import platform

import time

from os import system

engine=pyttsx3.init('sapi5')

#SETTING UP THE PYTTSX3

#Microsoft Speech API (SAPI5) is the technology for voice recognition and synthesis provided by Microsoft.

#The pyttsx3 module supports two voices first is female and the second is male which is provided by “sapi5” for windows.

#SETTING UP THE VOICE FOR THE ASSISTANT

voice=engine.getProperty('voices')

engine.setProperty('voice',voice[0].id)

#INITIALIZING FOR GOOGLE NEWS

google\_news=GoogleNews()

#SETTING UP THE RECOGNIZER INSTANCE

recognizer=sr.Recognizer()

#DEFINING FUNCTION FOR ASSISTANT TO SPEAK

def ai\_voice(audio):

engine.say(audio)

engine.runAndWait()

#FUNCTION WHICH WILL PROCESS THE COMMAND FROM USER

def read\_news():

with sr.Microphone() as source:

recognizer.adjust\_for\_ambient\_noise(source,duration=1)

#TO REDUCE THE BACKGROUND NOISE

ai\_voice("Which news you want?")

#CALLING ai\_voice FUNCTION TO SPEAK

audio=recognizer.listen(source,timeout=1)

#LISTENING TO THE USER

playsound('process.wav')

#PLAYING SOUND TO ACKNOWLEDGE USER

ai\_voice("Processing your query")

#CALLING ai\_function TO SPEAK

try:

#TRY BLOCK

answer=recognizer.recognize\_google(audio,language='en-US')

#RECOGNINZES THE COMMMAND WITH THE HELP OF ENGISH(us)

answer=answer.lower()

#CONVERTING THE OBTANINED STRING TO LOWER CASE

print("YOU SAID: "+format(answer))

#DISPLAYING THE COMMAND SPOKEN BY USER

print("--------------------------------------------------------")

print()

except Exception as e:

#EXCEPT BLOCK (IF ASSISTANT DOSENT UNDERSTAND THE COMMAND)

print(e)

ai\_voice("Sorry! I cannot understand your query.")

ai\_voice("Please Try Again!")

return "None"

return answer

#RETURNING THE OBTAINED COMMAND IN TEXT FORM

def search\_news(str):

#FUNCTION WILL SEARCH THE NEWS FROM THE GOOGLE NEWS

if 'headlines' in str or 'headline' in str or 'Today news' in str:

google\_news.get\_news('Today news')

#SEARCHING NEWS WITH THE HELP OF KEYWORD

google\_news.result()

elif 'politics' in str or 'politic' in str:

google\_news.get\_news('Politics')

google\_news.result()

elif 'sports' in str or 'sport'in str:

google\_news.get\_news('Sports')

google\_news.result()

elif 'stock markets' in str or 'stock market' in str:

google\_news.get\_news('Sports')

google\_news.result()

elif 'bollywood' in str or 'movies' in str or 'movie' in str:

google\_news.get\_news('Movies')

google\_news.result()

elif 'world' in str:

google\_news.get\_news('world')

google\_news.result()

else:

google\_news.get\_news('Breaking News')

google\_news.result()

ai\_voice("Getting the news")

news=google\_news.gettext()

#GETTING THE NEWS IN LIST FORMAT

for i in range(10):

#READING THE TOP-10 NEWS OF THE DAY

print(news[i])

print()

ai\_voice(news[i])

def mail():

#FUNCTION FOR DOING THE MAIL

with sr.Microphone() as source:

recognizer.adjust\_for\_ambient\_noise(source,duration=1)

#REDUCING THE BACKGROUND NOISE

print("WAITING FOR YOUR MESSAGE")

ai\_voice("WAITING FOR YOUR MESSAGE")

recorded\_audio=recognizer.listen(source)

playsound('process.wav')

print("DONE RECORDING!!")

ai\_voice("Processing !")

try: #TRY BLOCK

print("PRINTING THE MESSAGE")

text=recognizer.recognize\_google(recorded\_audio,language='en-US')

#LISTENING TO USER

ai\_voice("Your Message")

print("YOUR MESSAGE : ")

print(format(text))

#PRINTNING THE MAIL BODY SAID BY THE USER

except Exception as e: #EXCEPTION BLOCK

print(e)

ai\_voice("Enter the Receiver's e-mail address:")

playsound('process.wav')

receiver=input("Enter the Receiver's e-mail address: ")

#TAKING THE EMAIL ADDRESS OF RECEIVER

ai\_voice("Enter the your email address")

playsound('process.wav')

send=input("Enter the your email address: ")

#TAKING THE EMAIL ADDRESS OF SENDER

message=text

sender=yagmail.SMTP(send)

#SETTING UP THE PROTOCOL

ai\_voice("Enter the subject of the mail")

playsound('process.wav')

subject\_mail=input("Enter the subject of the mail: ")

#ENTER THE SUBJECT OF THE EMAIL

sender.send(to=receiver,subject=subject\_mail,contents=message)

#SENDING THE EMAIL

ai\_voice("E-mail sent!!")

playsound('process.wav')

return

#LOGIC FOR THE TIC TAE TOE GAME USING MIN-MAX ALGORITHM

# Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory.

# It provides an optimal move for the player assuming that opponent is also playing optimally.

# Mini-Max algorithm uses recursion to search through the game-tree.

# Min-Max algorithm is mostly used for game playing in AI.

# Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game.

# This Algorithm computes the minimax decision for the current state.

# In this algorithm two players play the game, one is called MAX and other is called MIN.

# Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.

# Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.

# The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.

# The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

HUMAN = -1

AI = +1

board = [

[0, 0, 0],

[0, 0, 0],

[0, 0, 0],

]

def evaluate(state):

# Function to heuristic evaluation of state.

# :param state: the state of the current board

# :return: +1 if the AIuter wins; -1 if the human wins; 0 draw

if wins(state, AI):

score = +1

elif wins(state, HUMAN):

score = -1

else:

score = 0

return score

def wins(state, player):

# This function tests if a specific player wins. Possibilities:

# \* Three rows [X X X] or [O O O]

# \* Three cols [X X X] or [O O O]

# \* Two diagonals [X X X] or [O O O]

# :param state: the state of the current board

# :param player: a human or a AIuter

# :return: True if the player wins

win\_state = [

[state[0][0], state[0][1], state[0][2]],

[state[1][0], state[1][1], state[1][2]],

[state[2][0], state[2][1], state[2][2]],

[state[0][0], state[1][0], state[2][0]],

[state[0][1], state[1][1], state[2][1]],

[state[0][2], state[1][2], state[2][2]],

[state[0][0], state[1][1], state[2][2]],

[state[2][0], state[1][1], state[0][2]],

]

if [player, player, player] in win\_state:

return True

else:

return False

def game\_over(state):

# This function test if the human or AIuter wins

# param state: the state of the current board

# return: True if the human or AIuter wins

return wins(state, HUMAN) or wins(state, AI)

def empty\_cells(state):

# Each empty cell will be added into cells' list

# param state: the state of the current board

# return: a list of empty cells

cells = []

for x, row in enumerate(state):

for y, cell in enumerate(row):

if cell == 0:

cells.append([x, y])

return cells

def valid\_move(x, y):

# A move is valid if the chosen cell is empty

# param x: X coordinate

# param y: Y coordinate

# return: True if the board[x][y] is empty

if [x, y] in empty\_cells(board):

return True

else:

return False

def set\_move(x, y, player):

# Set the move on board, if the coordinates are valid

# param x: X coordinate

# param y: Y coordinate

# param player: the current player

if valid\_move(x, y):

board[x][y] = player

return True

else:

return False

def minimax(state, depth, player):

# AI function that choice the best move

# param state: current state of the board

# param depth: node index in the tree (0 <= depth <= 9),

# but never nine in this case (see iaturn() function)

# param player: an human or a AIuter

# return: a list with [the best row, best col, best score]

if player == AI:

best = [-1, -1, -infinity]

else:

best = [-1, -1, +infinity]

if depth == 0 or game\_over(state):

score = evaluate(state)

return [-1, -1, score]

for cell in empty\_cells(state):

x, y = cell[0], cell[1]

state[x][y] = player

score = minimax(state, depth - 1, -player)

state[x][y] = 0

score[0], score[1] = x, y

if player == AI:

if score[2] > best[2]:

best = score # max value

else:

if score[2] < best[2]:

best = score # min value

return best

def clean():

# Clears the console

os\_name = platform.system().lower()

if 'windows' in os\_name:

system('cls')

else:

system('clear')

def render(state, c\_choice, h\_choice):

# Print the board on console

# param state: current state of the board

chars = {

-1: h\_choice,

+1: c\_choice,

0: ' '

}

str\_line = '---------------'

print('\n' + str\_line)

for row in state:

for cell in row:

symbol = chars[cell]

print(f'| {symbol} |', end='')

print('\n' + str\_line)

def ai\_turn(c\_choice, h\_choice):

# It calls the minimax function if the depth < 9,

# else it choices a random coordinate.

# param c\_choice: AIuter's choice X or O

# param h\_choice: human's choice X or O

depth = len(empty\_cells(board))

if depth == 0 or game\_over(board):

return

clean()

print(f'AIuter turn [{c\_choice}]')

render(board, c\_choice, h\_choice)

if depth == 9:

x = choice([0, 1, 2])

y = choice([0, 1, 2])

else:

move = minimax(board, depth, AI)

x, y = move[0], move[1]

set\_move(x, y, AI)

time.sleep(1)

def human\_turn(c\_choice, h\_choice):

# The Human plays choosing a valid move.

# param c\_choice: AIuter's choice X or O

# param h\_choice: human's choice X or O

# return:

depth = len(empty\_cells(board))

if depth == 0 or game\_over(board):

return

# Dictionary of valid moves

move = -1

moves = {

1: [0, 0], 2: [0, 1], 3: [0, 2],

4: [1, 0], 5: [1, 1], 6: [1, 2],

7: [2, 0], 8: [2, 1], 9: [2, 2],

}

clean()

print(f'Human turn [{h\_choice}]')

render(board, c\_choice, h\_choice)

while move < 1 or move > 9:

try:

move = int(input('Use numpad (1..9): '))

coord = moves[move]

can\_move = set\_move(coord[0], coord[1], HUMAN)

if not can\_move:

print('Bad move')

move = -1

except (EOFError, KeyboardInterrupt):

print('Bye')

exit()

except (KeyError, ValueError):

print('Bad choice')

def play():

# Main function that calls all functions

clean()

h\_choice = '' # X or O

c\_choice = '' # X or O

first = '' # if human is the first

# Human chooses X or O to play

while h\_choice != 'O' and h\_choice != 'X':

try:

print('')

h\_choice = input('Choose X or O\nChosen: ').upper()

except (EOFError, KeyboardInterrupt):

print('Bye')

exit()

except (KeyError, ValueError):

print('Bad choice')

# Setting AIuter's choice

if h\_choice == 'X':

c\_choice = 'O'

else:

c\_choice = 'X'

# Human may starts first

clean()

while first != 'Y' and first != 'N':

try:

first = input('First to start?[y/n]: ').upper()

except (EOFError, KeyboardInterrupt):

print('Bye')

exit()

except (KeyError, ValueError):

print('Bad choice')

# Main loop of this game

while len(empty\_cells(board)) > 0 and not game\_over(board):

if first == 'N':

ai\_turn(c\_choice, h\_choice)

first = ''

human\_turn(c\_choice, h\_choice)

ai\_turn(c\_choice, h\_choice)

# Game over message

if wins(board, HUMAN):

clean()

print(f'Human turn [{h\_choice}]')

render(board, c\_choice, h\_choice)

print('YOU WIN!')

ai\_voice("YOU WIN!!")

ai\_voice("CONGRATS!!")

elif wins(board, AI):

clean()

print(f'AIuter turn [{c\_choice}]')

render(board, c\_choice, h\_choice)

print('YOU LOSE!')

ai\_voice("YOU LOSE!!")

else:

clean()

render(board, c\_choice, h\_choice)

print('DRAW!')

ai\_voice("GAME DRAWN!!")

exit()

status=True

while status:

print("--------------------------------------------------------")

print()

ai\_voice("WELCOME !!")

ai\_voice("Please select from the Following!")

print("--------------------------------------------------------")

print()

print("1. News Assistant")

print("2. Auto Email")

print("3. Play")

print("4. Change Voice of Assistant")

print("5. Quit/End/Good Bye")

print("6. About Me")

print("--------------------------------------------------------")

print()

with sr.Microphone() as source:

recognizer.adjust\_for\_ambient\_noise(source,duration=0.5)

ai\_voice("Please select an option from above")

audio=recognizer.listen(source,timeout=1)

playsound('process.wav')

ai\_voice("Processing your query")

answer=recognizer.recognize\_google(audio,language='en-US')

answer=answer.lower()

print("YOU SAID: "+format(answer))

print("--------------------------------------------------------")

print()

if "news" in answer:

news=read\_news()

search\_news(news)

elif "mail" in answer or "email" in answer :

mail()

elif "play" in answer or "game" in answer:

play()

elif "change" in answer or "voice" in answer:

print("--------------------------------------------------------")

print()

print("Please select from the following:")

print("1.Male\n2.Female")

gender=input()

if gender=="male":

engine.setProperty('voice',voice[0].id)

elif gender=="female":

engine.setProperty('voice',voice[1].id)

ai\_voice("HELLO! I am your new helper.")

elif "about me" in answer or "abou" in answer:

ai\_voice("Hello! I am assistant X")

ai\_voice("I AM CREATED BY PARTH PATEL AND SAUMYA SHAH AND SHRUTI PATEL")

elif "quit" in answer or "end" in answer or "goodbye" in answer:

ai\_voice("GOOD BYE!!")

status=False

else:

ai\_voice("Sorry! I cannot process your Query!!")