Universitatea POLITEHNICA din Bucureşti
Facultatea de electronică, Telecomunicații și Tehnologia Informației

INTERFEȚE OM-MAȘINĂ

JOC DE TIP SNAKE CONTROLAT PRIN COMENZI VOCALE

STUDENT: **ABAGERU ȘTEFAN-FLORIAN**
MĂLUROIU-VLĂDUȚ NICOLAE-RĂZVAN

GRUPA: **444A**

CUPRINS

Cuprins

Tema proiectului

Tema proiectului se bazează pe dezvoltarea în limbajul de programare Python a unui joc de tip Snake ce se poate controla prin comenzi vocale primite de la user în timp real. Se pot folosi diferite tehnologii si biblioteci, precum: Tkinter, PyQt, PySide, Kivy, wxPython, SpeechRecognition, PyAudio, Librosa.

Pentru tema individual primită, am ales să folosim bibliotecile Tkinter pentru interfața jocului si SpeechRecognition pentru procesarea comenzilor vocale.

Descrierea aplicatiei

Ce este Python?

Python este un limbaj de programare de nivel înalt, cunoscut pentru simplitatea sintaxei și versatilitatea sa. Dezvoltat pentru a fi ușor de învățat și citit, Python suportă diverse paradigme de programare și oferă o bibliotecă standard bogată, facilitând dezvoltarea rapidă a aplicațiilor. Cu o comunitate activă și extensibilitate, Python a devenit omniprezent în dezvoltarea web, analiza datelor, inteligența artificială și multe alte domenii.

Ce este Tkinter?

Tkinter, biblioteca integrată pentru interfețe grafice în Python, oferă un mod simplu și eficient de a crea aplicații cu interfețe vizuale. Cu sintaxa sa simplă și widget-urile încorporate, Tkinter facilitează dezvoltarea rapidă și portabilă a aplicațiilor cu GUI în Python. Este o alegere populară pentru dezvoltatori datorită flexibilității, extensibilității și documentației detaliate.

Ce este SpeechRecognition?

SpeechRecognition, biblioteca Python pentru recunoașterea vocală, furnizează o interfață simplă și eficientă pentru integrarea capacităților de recunoaștere vocală în proiecte Python. Cu suport pentru diverse servicii și formate audio, precum și o documentație detaliată, este alegerea ideală pentru dezvoltatorii care doresc să adauge funcționalități de recunoaștere vocală precise în aplicațiile lor.

Implementarea claselor și a funcțiilor

```
class Snake:
    def __init__(self):
        self.body_size = BODY_SIZE
        self.coordinates = []
        self.squares = []

        for i in range(0, BODY_SIZE):
            self.coordinates.append([0, 0])

        for x, y in self.coordinates:
            square = canvas.create_rectangle(
                x, y, x + SPACE_SIZE, y + SPACE_SIZE,
                fill=SNAKE, tag="snake"
            )
            self.squares.append(square)
```

Figura 1. Clasa "Snake"

Această clasă definește clasa "Snake" în Python, care reprezintă un șarpe în joc. Constructorul clasei stabilește dimensiunea corpului șarpelui, inițializează coordonatele și creează pătrate colorate pentru fiecare segment al șarpelui pe un canvas. Pătratele sunt stocate într-o listă pentru gestionarea ulterioară a șarpelui.

```python
class Food:
    def __init__(self):
        x = random.randint(0, (WIDTH / SPACE_SIZE) - 1) * SPACE_SIZE
        y = random.randint(0, (HEIGHT / SPACE_SIZE) - 1) * SPACE_SIZE
        self.coordinates = [x, y]
        canvas.create_oval(
            x, y, x + SPACE_SIZE, y + SPACE_SIZE, fill=FOOD, tag="food"
        )
```

Figura 2. Clasa "Food"

Această clasă definește clasa "Food" în Python, care reprezintă obiectul hrană în joc. Constructorul clasei generează aleator coordonate pentru hrană, asigurându-se că aceasta se află în limitele canvas-ului. Apoi, creează un oval colorat (reprezentând hrana) la acele coordonate pe canvas.

```python
def next_turn(snake, food):
    x, y = snake.coordinates[0]

    if direction == "up":
        y -= SPACE_SIZE
    elif direction == "down":
        y += SPACE_SIZE
    elif direction == "left":
        x -= SPACE_SIZE
    elif direction == "right":
        x += SPACE_SIZE

    x = x % WIDTH
    y = y % HEIGHT

    snake.coordinates.insert(0, (x, y))

    square = canvas.create_rectangle(
        x, y, x + SPACE_SIZE, y + SPACE_SIZE, fill=SNAKE
    )
    snake.squares.insert(0, square)

    if x == food.coordinates[0] and y == food.coordinates[1]:
        global score
        score += 1
        label.config(text="Points:{}".format(score))
        canvas.delete("food")
        food = Food()

    else:
        del snake.coordinates[-1]
        canvas.delete(snake.squares[-1])
        del snake.squares[-1]

    if check_collisions(snake):
        game_over()
    else:
        window.after(SPEED, next_turn, snake, food)
```

Figura 3. Funcția "next turn"

Funcția **next_turn** gestionează evoluția jocului Snake, actualizând poziția șarpelui, verificând coliziunile, gestionând hrana și menținând recursivitatea pentru continuarea jocului. Se ocupă de mișcarea șarpelui, verificarea coliziunilor, actualizarea scorului și regenerarea hranei. Este structurată într-un mod care permite evoluția continuă a jocului la intervale definite de viteză (SPEED).

```
def change_direction(new_direction):
    global direction
    if new_direction == 'left':
        if direction != 'right':
            direction = new_direction
    elif new_direction == 'right':
        if direction != 'left':
            direction = new_direction
    elif new_direction == 'up':
        if direction != 'down':
            direction = new_direction
    elif new_direction == 'down':
        if direction != 'up':
            direction = new_direction
```

Figura 4. Funcția "change_direction"

Funcția **change_direction** permite schimbarea direcției șarpelui în joc. Dacă noua direcție este validă (adică nu opusă direcției curente), variabila globală `direction` este actualizată corespunzător. Este o funcție compactă care împiedică șarpele să se întoarcă direct înapoi și să se ciocnească cu propriul său corp.

```
def check_collisions(snake):
    x, y = snake.coordinates[0]
    if x < 0 or x >= WIDTH:
        return True
    elif y < 0 or y >= HEIGHT:
        return True
    for body_part in snake.coordinates[1:]:
        if x == body_part[0] and y == body_part[1]:
            return True
    return False
```

Figura 5. Funcția "check_collisions"

Funcția **check_collisions** este responsabilă de verificarea coliziunilor șarpelui cu marginile ecranului și cu propriul său corp în joc. Returnează True dacă șarpele se ciocnește cu el însuși, altfel returnează False. Este o funcție compactă și eficientă care contribuie la gestionarea corectă a coliziunilor în joc.

```
def game_over():
    canvas.delete(ALL)
    canvas.create_text(
        canvas.winfo_width()/2,
        canvas.winfo_height()/2,
        font=('consolas', 70),
        text="GAME OVER", fill="red",
        tag="gameover"
    )
```

Figura 6. Funcția "game_over"

Funcția **game_over** are rolul de a gestiona afișarea mesajului "GAME OVER" pe canvas și eliminarea tuturor elementelor existente. Prin apelul funcției `canvas.delete(ALL)`, se elimină toate obiectele grafice de pe canvas, asigurând o curățare completă. Apoi, se adaugă un text "GAME OVER" cu dimensiuni mari și culoare roșie în mijlocul canvas-ului utilizând `canvas.create_text`.

```
def listen_for_commands():
    recognizer = sr.Recognizer()
    microphone = sr.Microphone()

    while True:
        with microphone as source:
            print("Ascultare...")
            recognizer.adjust_for_ambient_noise(source)
            try:
                audio = recognizer.listen(source, phrase_time_limit=2, timeout=3)
                command = recognizer.recognize_google(audio).lower()
                print("Ai spus:", command)
                process_voice_command(command)
            except sr.UnknownValueError:
                print("Nu s-a inteles audio-ul")
            except sr.WaitTimeoutError:
                print("Fara comanda")
                pass
            except sr.RequestError as e:
                print(f"Eroare ; {e}")
```

Figura 7. Funcția "listen_for_commands"

Funcția **listen_for_commands** utilizează SpeechRecognition pentru a asculta comenzile vocale continuu:

- Inițializează un obiect `Recognizer` și un obiect `Microphone` pentru captarea audio.
- Intră într-o buclă infinită pentru a asculta continuu comenzile vocale.
- Utilizează un bloc `with` pentru a asigura eliberarea resurselor microfonului după încheierea utilizării.
- Ajustează nivelul de zgomot ambiental și începe ascultarea. Încearcă să recunoască comanda vocală utilizând `recognizer.recognize_google(audio)`, apoi converteste comanda în litere mici.
- Afișează comanda recunoscută și apelează funcția **process_voice_command** pentru a executa acțiunile corespunzătoare comenzii.
- Gestionarea excepțiilor: **sr.UnknownValueError** pentru cazul în care nu se recunoaște comanda vocală, **sr.WaitTimeoutError** pentru cazul în care nu se primesc comenzi într-un interval de timp și **sr.RequestError** pentru orice eroare în cererea către serviciul de recunoaștere vocală.

```
def process_voice_command(command):
    command_parts = command.split()

    if command_parts:
        first_word = command_parts[0]

        if first_word[0] == "u":
            change_direction("up")
        elif first_word[0] == "d":
            change_direction("down")
        elif first_word[0] == "l":
            change_direction("left")
        elif first_word[0] == "r":
            change_direction("right")
```

Figura 8. Funcția "process_voice_command"

Funcția **process_voice_command** primește o comandă vocală și o descompune în părți:

- command_parts = command.split(): Descompune comanda vocală în părți separate;

- Verifică dacă există părți în comandă;
- Extrage prima parte a comenzii;
- Utilizează prima literă a primului cuvânt pentru a determina direcția și apoi apelează funcția **change_direction** cu direcția corespunzătoare ("up", "down", "left" sau "right");

Această funcție procesează comenzile vocale și le traduce în direcții pentru șarpele din joc. Este concepută pentru a răspunde la comenzile vocale care încep cu literele "u", "d", "l" sau "r".

```
# MAIN #
window = Tk()
window.title("SNEIK")

score = 0
direction = 'down'

label = Label(window, text="Points:{}".format(score),
              font=('consolas', 20))
label.pack()

canvas = Canvas(window, bg=BACKGROUND,
                height=HEIGHT, width=WIDTH)
canvas.pack()

window.update()

window_width = window.winfo_width()
window_height = window.winfo_height()
screen_width = window.winfo_screenwidth()
screen_height = window.winfo_screenheight()

x = int((screen_width/2) - (window_width/2))
y = int((screen_height/2) - (window_height/2))

window.geometry(f"{window_width}x{window_height}+{x}+{y}")

voice_thread = Thread(target=listen_for_commands)
voice_thread.daemon = True
voice_thread.start()

snake = Snake()
food = Food()

next_turn(snake, food)

window.mainloop()
```

Figura 9. Codul principal al jocului

Codul principal al jocului Snake utilizează Tkinter pentru interfața grafică și lansează jocul într-un thread separat pentru ascultarea comenzilor vocale:

Setup UI: Creează fereastra jocului, un canvas pentru desenarea elementelor grafice, și afișează scorul.

Dimensiuni fereastră: Centralizează fereastra jocului pe ecran, luând în considerare dimensiunile ecranului.

Inițializare Snake și Food: Creează instanțe ale claselor Snake și Food.

Începe jocul: Apelează funcția **next_turn** pentru a începe logica jocului și menține aplicația în bucla (window.mainloop()).

Datorită faptului că dorim să avem comenzi vocale în timp real cu desfașurarea jocului, pentru a nu opri jocul în momentul în care se face o înregistrare a vocii a fost nevoie de împartirea acestuia pe două threaduri care rulează concomitent (un thread pentru desfășurarea jocului și un thread pentru prelucrarea comenzilor vocale).

Desfășurarea jocului

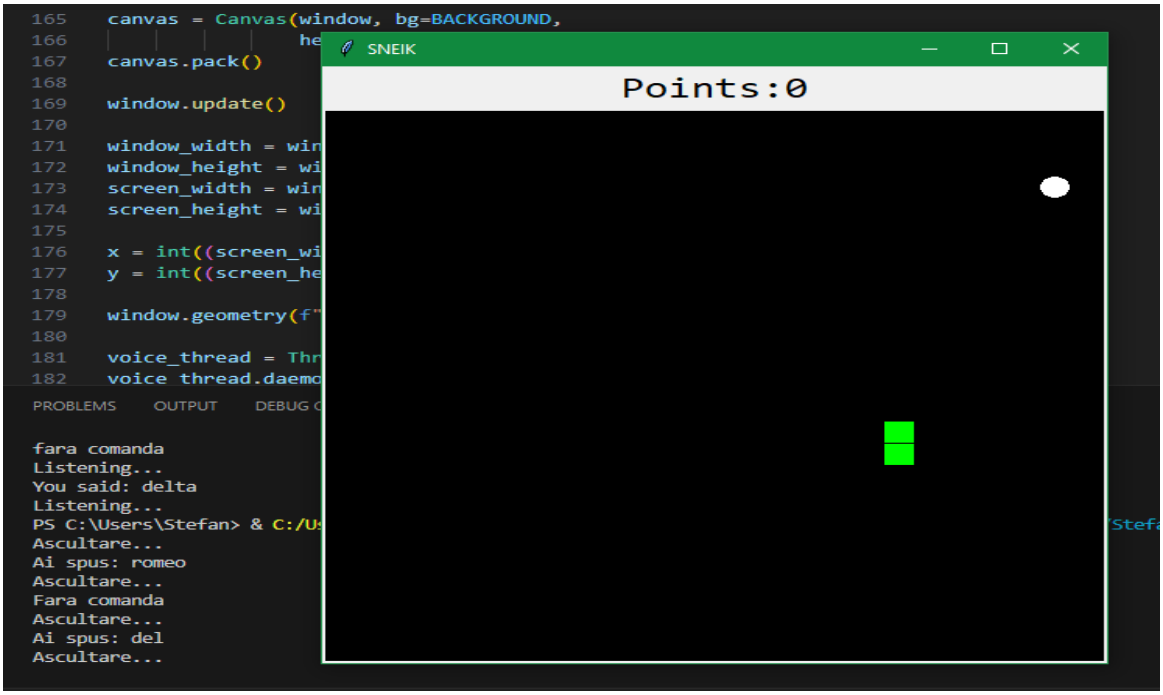

Figura 10. Fereastra jocului

Se începe cu un snake de mărime 2 care are ca scop colectarea hranei pusă aleator pe spațiul de joc.

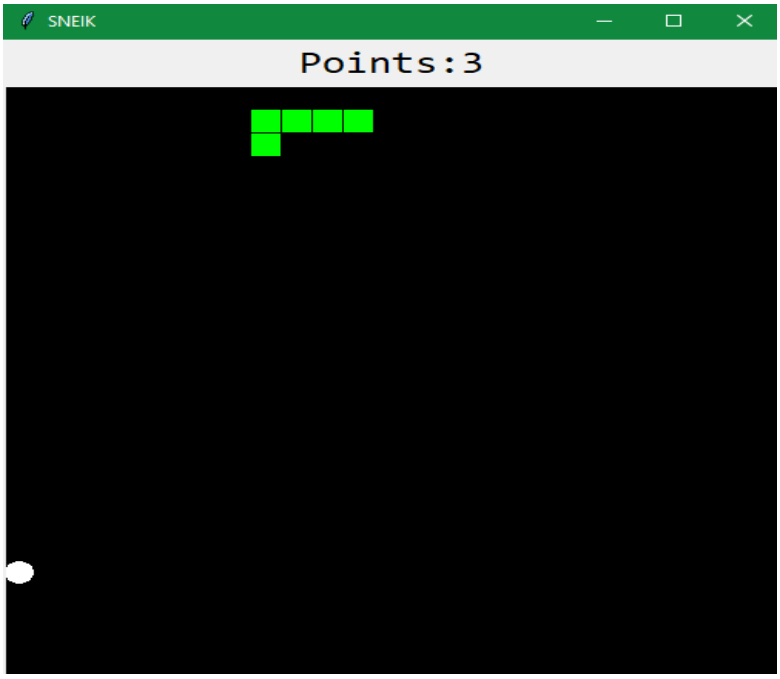

Figura 11. Captură din timpul jocului

Colectarea hranei rezultă în creșterea snake-ului și a scorului general.

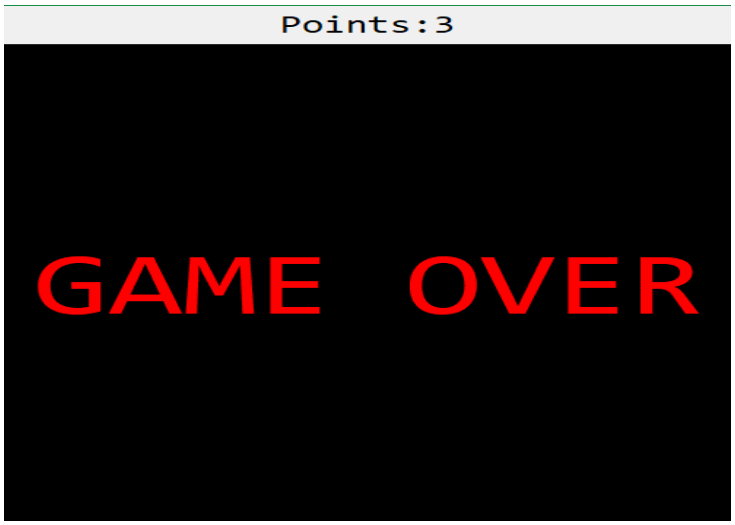

Figura 12. Sfârșitul jocului

În momentul in care snake-ul se atinge de el însuși, atunci apare fereastra de game over și jocul se termină.

Concluzii

Jocul Snake implementat în acest cod integrează cu succes interacțiunea vocală, adăugând o dimensiune captivantă și inovatoare experienței tradiționale a jocului. Acest proiect ne-a făcut să cimentăm bazele limbajului de programare Python și să căpătăm noi cunoștințe lucrând cu bibliotecile Tkinter si SpeechRecognition.

Bibliografie

https://docs.python.org/3/library/tk.html

https://www.geeksforgeeks.org/python-gui-tkinter/

https://pypi.org/project/SpeechRecognition/

https://realpython.com/python-speech-recognition/

https://www.youtube.com/

https://docs.python.org/3/library/threading.html

https://www.geeksforgeeks.org/multithreading-python-set-1/

https://en.wikipedia.org/wiki/Thread_(computing)

Lista figuri

Anexa cod

```
from tkinter import *
import random
import speech_recognition as sr
from threading import Thread

WIDTH = 500
HEIGHT = 500
SPEED = 500
SPACE_SIZE = 20
BODY_SIZE = 2
SNAKE = "#00FF00"
FOOD = "#FFFFFF"
BACKGROUND = "#000000"

class Snake:
    def __init__(self):
        self.body_size = BODY_SIZE
        self.coordinates = []
        self.squares = []

        for i in range(0, BODY_SIZE):
            self.coordinates.append([0, 0])

        for x, y in self.coordinates:
            square = canvas.create_rectangle(
                x, y, x + SPACE_SIZE, y + SPACE_SIZE,
                fill=SNAKE, tag="snake"
            )
            self.squares.append(square)

class Food:
    def __init__(self):
        x = random.randint(0, (WIDTH / SPACE_SIZE) - 1) * SPACE_SIZE
        y = random.randint(0, (HEIGHT / SPACE_SIZE) - 1) * SPACE_SIZE
        self.coordinates = [x, y]
        canvas.create_oval(
            x, y, x + SPACE_SIZE, y + SPACE_SIZE, fill=FOOD, tag="food"
        )

def next_turn(snake, food):
    x, y = snake.coordinates[0]
```

```
    if direction == "up":
        y -= SPACE_SIZE
    elif direction == "down":
        y += SPACE_SIZE
    elif direction == "left":
        x -= SPACE_SIZE
    elif direction == "right":
        x += SPACE_SIZE

    x = x % WIDTH
    y = y % HEIGHT

    snake.coordinates.insert(0, (x, y))

    square = canvas.create_rectangle(
        x, y, x + SPACE_SIZE, y + SPACE_SIZE, fill=SNAKE
    )
    snake.squares.insert(0, square)

    if x == food.coordinates[0] and y == food.coordinates[1]:
        global score
        score += 1
        label.config(text="Points:{}".format(score))
        canvas.delete("food")
        food = Food()

    else:
        del snake.coordinates[-1]
        canvas.delete(snake.squares[-1])
        del snake.squares[-1]

    if check_collisions(snake):
        game_over()
    else:
        window.after(SPEED, next_turn, snake, food)

def change_direction(new_direction):
    global direction
    if new_direction == 'left':
        if direction != 'right':
            direction = new_direction
    elif new_direction == 'right':
        if direction != 'left':
            direction = new_direction
    elif new_direction == 'up':
```

```
        if direction != 'down':
            direction = new_direction
    elif new_direction == 'down':
        if direction != 'up':
            direction = new_direction

def check_collisions(snake):
    x, y = snake.coordinates[0]
    if x < 0 or x >= WIDTH:
        return True
    elif y < 0 or y >= HEIGHT:
        return True
    for body_part in snake.coordinates[1:]:
        if x == body_part[0] and y == body_part[1]:
            return True
    return False

def game_over():
    canvas.delete(ALL)
    canvas.create_text(
        canvas.winfo_width()/2,
        canvas.winfo_height()/2,
        font=('consolas', 70),
        text="GAME OVER", fill="red",
        tag="gameover"
    )

def listen_for_commands():
    recognizer = sr.Recognizer()
    microphone = sr.Microphone()

    while True:
        with microphone as source:
            print("Ascultare...")
            recognizer.adjust_for_ambient_noise(source)
            try:
                audio = recognizer.listen(source, phrase_time_limit=2, timeout=3)
                command = recognizer.recognize_google(audio).lower()
                print("Ai spus:", command)
                process_voice_command(command)
            except sr.UnknownValueError:
                print("Nu s-a inteles audio-ul")
            except sr.WaitTimeoutError:
                print("Fara comanda")
                pass
```

```
            except sr.RequestError as e:
                print(f"Eroare ; {e}")


def process_voice_command(command):
    command_parts = command.split()

    if command_parts:
        first_word = command_parts[0]

        if first_word[0] == "u":
            change_direction("up")
        elif first_word[0] == "d":
            change_direction("down")
        elif first_word[0] == "l":
            change_direction("left")
        elif first_word[0] == "r":
            change_direction("right")


# MAIN #
window = Tk()
window.title("SNEIK")

score = 0
direction = 'down'

label = Label(window, text="Points:{}".format(score),
              font=('consolas', 20))
label.pack()

canvas = Canvas(window, bg=BACKGROUND,
                height=HEIGHT, width=WIDTH)
canvas.pack()

window.update()

window_width = window.winfo_width()
window_height = window.winfo_height()
screen_width = window.winfo_screenwidth()
screen_height = window.winfo_screenheight()

x = int((screen_width/2) - (window_width/2))
y = int((screen_height/2) - (window_height/2))
```

```
window.geometry(f"{window_width}x{window_height}+{x}+{y}")

voice_thread = Thread(target=listen_for_commands)
voice_thread.daemon = True
voice_thread.start()

snake = Snake()
food = Food()

next_turn(snake, food)

window.mainloop()
```