Proiect IOM

Interfață pentru un joc interactive
Documentație

Ionescu Maria-Catalina
Alecu Cedric-Ioachim
444A

Introducere

Acest proiect își propune implementarea clasicului joc de Minesweeper folosind interfețe in Python. Jocul consta in o table de 8x8 pătrățele in care se afla bombe puse la întâmplare pe care jucătorul trebuie sa le evite pentru a câștiga.

Caracteristici

- **Mod de joc Minesweeper**: o grilă 8x8 în care jucătorul dezvăluie plăci, încercând să evite bombele.
- **Clasament**: afișează primele 10 scoruri și permite utilizatorilor să-și trimită scorul după încheierea unui joc.
- **Efecte sonore**: Redă un sunet atunci când o bombă este declanșată, îmbunătățind experiența de joc.
- **Interfață de utilizare receptivă**: construită cu Tkinter pentru a crea o interfață grafică ușor de utilizat.

Biblioteci folosite

- **Tkinter**: Pentru interfața grafică cu utilizatorul (GUI).
- **Pygame**: Pentru gestionarea sunetului (sunetul exploziei bombei).
- **Random**: Pentru plasarea aleatorie a bombelor pe grilă.
- **Json**: pentru salvarea și încărcarea datelor din clasament.
- **Os**: Pentru verificarea existenței fișierelor.

Descrierea funcțiilor si clase principale

play_bomb_sound()
Redă sunetul exploziei atunci când este declanșată o bombă.

```
def play_bomb_sound():
    # Load the sound file and play it
    pygame.mixer.Sound("bomb_sound.wav").play()
```

LeaderboardManager
Această clasă se ocupă de funcționalitatea clasamentului: Încărcarea și salvarea datelor clasamentului dintr-un fișier JSON. Adăugarea de scoruri și asigurarea faptului că doar primele 10 scoruri sunt salvate.

```
class LeaderboardManager:
    def __init__(self, filename='leaderboard.json'):
        self.filename = filename
        self.leaderboard = self.load_leaderboard()

    def load_leaderboard(self):
        if os.path.exists(self.filename):
            with open(self.filename, 'r') as f:
                return json.load(f)
        return []

    def save_leaderboard(self):
        with open(self.filename, 'w') as f:
            json.dump(self.leaderboard, f)

    def add_score(self, name, score):
        # Remove oldest entries if more than 10
        if len(self.leaderboard) >= 10:
            self.leaderboard = sorted(self.leaderboard, key=lambda x: x['score'], reverse=True)[:10]

        # Add new score
        self.leaderboard.append({'name': name, 'score': score})

        # Sort and keep top 10
        self.leaderboard.sort(key=lambda x: x['score'], reverse=True)
        self.leaderboard = self.leaderboard[:10]

        self.save_leaderboard()
```

LeaderboardWindow
Această clasă creează o nouă fereastră pentru a afișa clasamentul, afișând primele 10 scoruri într-o listă derulabilă.

```
class LeaderboardWindow:
    def __init__(self, root, leaderboard_manager):
        self.window = tk.Toplevel(root)
        self.window.title("Leaderboard")
        self.window.geometry("300x400")

        # Title
        tk.Label(self.window, text="Leaderboard", font=("Arial", 16, "bold")).pack(pady=10)

        # Leaderboard entries
        leaderboard = leaderboard_manager.leaderboard
        for i, entry in enumerate(leaderboard, 1):
            entry_frame = tk.Frame(self.window)
            entry_frame.pack(fill='x', padx=20, pady=5)

            tk.Label(entry_frame, text=f"{i}. {entry['name']}", font=("Arial", 12), width=15, anchor='w').pack(side='left')
            tk.Label(entry_frame, text=str(entry['score']), font=("Arial", 12), width=10, anchor='e').pack(side='right')
```

Minesweeper
Aceasta clasa conține toate funcțiile descris mai jos:

Show_start_screen()
este responsabilă pentru afișarea ecranului de pornire al jocului Minesweeper. Aceasta curăță orice widget existent din fereastră, creează un layout centrat și adaugă butoanele pentru a începe jocul sau a accesa clasamentul (leaderboard).

```
def show_start_screen(self):
    # Clear any existing widgets
    for widget in self.root.winfo_children():
        widget.destroy()

    # Center frame
    frame = tk.Frame(self.root)
    frame.place(relx=0.5, rely=0.5, anchor='center')

    # Title
    tk.Label(frame, text="Minesweeper", font=("Arial", 20, "bold")).pack(pady=10)

    # Play Button
    play_button = tk.Button(frame, text="Play", font=("Arial", 14), command=self.start_game, width=15)
    play_button.pack(pady=5)

    # Leaderboard Button
    leaderboard_button = tk.Button(frame, text="Leaderboard", font=("Arial", 14),
                                   command=self.show_leaderboard, width=15)
    leaderboard_button.pack(pady=5)
```

Start_game()
este responsabilă pentru inițializarea și afișarea jocului Minesweeper

```
def start_game(self):
    # Clear start screen
    for widget in self.root.winfo_children():
        widget.destroy()

    # Reset game state
    self.board = [[None for _ in range(self.grid_size)] for _ in range(self.grid_size)]
    self.buttons = [[None for _ in range(self.grid_size)] for _ in range(self.grid_size)]

    # Create back button
    back_button = tk.Button(self.root, text="Back", command=self.show_start_screen)
    back_button.grid(row=self.grid_size, column=0, columnspan=2, sticky='w', pady=5)

    self.generate_board()
    self.create_buttons()

    # Reset score tracking
    self.start_time = threading.Timer(1, self.increment_score).start()
    self.score = 0
    self.safe_tiles_count = self.grid_size * self.grid_size - self.bomb_count
```

Increment_score()

folosește un timer în thread-uri pentru a crește scorul cu 1 la fiecare secundă

```
def increment_score(self):
    # Increment score
    self.score += 1
    self.start_time = threading.Timer(1, self.increment_score).start()
```

Generate_board()

Funcția generate_board generează tabla pentru jocul Minesweeper, plasând mai întâi bombele random apoi calculează pentru fiecare căsuță cate bombe sunt in vecinătate.

```
def generate_board(self):
    # Plasarea bombelor
    self.bombs = set(random.sample(range(self.grid_size * self.grid_size), self.bomb_count))
    directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]

    # Inițializare tablă cu valori implicite (0 pentru casetele goale)
    for i in range(self.grid_size):
        for j in range(self.grid_size):
            self.board[i][j] = 0

    # Plasarea bombelor
    for bomb in self.bombs:
        x, y = divmod(bomb, self.grid_size)
        self.board[x][y] = "B"

        # Actualizarea numerelor din jurul bombei
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < self.grid_size and 0 <= ny < self.grid_size and self.board[nx][ny] != "B":
                self.board[nx][ny] += 1

    # Atribuirea valorilor finale pentru casetele care nu sunt bombe
    for i in range(self.grid_size):
        for j in range(self.grid_size):
            if self.board[i][j] != "B" and self.board[i][j] == 0:
                self.board[i][j] = ""  # Lasă caseta goală dacă nu sunt bombe adiacente
            elif self.board[i][j] == 0:  # Dacă este o casetă goală, o lăsăm goală
                self.board[i][j] = ""
```

Reveal_tile()

Dezvăluie o căsuță din tabla

```
def reveal_tile(self, i, j):
    if self.board[i][j] == "B":
        self.buttons[i][j].config(text="💣", bg="red", state="disabled")
        play_bomb_sound()
        self.game_over()
    elif self.board[i][j] == "":
        self.reveal_blank(i, j)
    else:
        self.buttons[i][j].config(text=str(self.board[i][j]), bg="lightgray", state="disabled")
        self.safe_tiles_count -= 1

    # Check for win condition
    if self.safe_tiles_count == 0:
        self.game_win()
```

Reveal_blank()
Dezvăluie toate căsuțele goale alăturate

```
def reveal_blank(self, i, j):
    stack = [(i, j)]
    visited = set()

    while stack:
        ci, cj = stack.pop()
        if (ci, cj) in visited:
            continue
        visited.add((ci, cj))

        self.buttons[ci][cj].config(text="", bg="lightgray", state="disabled")
        self.safe_tiles_count -= 1

        # Add neighbors if they are blank or numbered
        directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]
        for dx, dy in directions:
            ni, nj = ci + dx, cj + dy
            if 0 <= ni < self.grid_size and 0 <= nj < self.grid_size:
                if self.board[ni][nj] == "" and (ni, nj) not in visited:
                    stack.append((ni, nj))
                elif self.board[ni][nj] != "B":
                    self.buttons[ni][nj].config(text=str(self.board[ni][nj]), bg="lightgray", state="disabled")
                    self.safe_tiles_count -= 1
```

game_win()
Oprește jocul, dezvăluie toate bombele si salvează scorul

```
def game_win(self):
    # Stop the score timer
    try:
        self.start_time.cancel()
    except:
        pass

    # Reveal all bombs
    for i in range(self.grid_size):
        for j in range(self.grid_size):
            if self.board[i][j] == "B":
                self.buttons[i][j].config(text="💣", bg="green", state="disabled")
            else:
                self.buttons[i][j].config(text=str(self.board[i][j]), state="disabled")

    # Ask for player name and save score
    self.ask_for_name(True)
```

game_over()
Oprește jocul, dezvăluie toate bombele si salvează scorul

```
def game_over(self):
    # Stop the score timer
    try:
        self.start_time.cancel()
    except:
        pass

    # Reveal all bombs
    for i in range(self.grid_size):
        for j in range(self.grid_size):
            if self.board[i][j] == "B":
                self.buttons[i][j].config(text="💣", bg="red", state="disabled")
            else:
                self.buttons[i][j].config(text=str(self.board[i][j]), state="disabled")

    # Ask for player name and save score
    self.ask_for_name(False)
```