Universitatea Tehnică “Gheorghe Asachi”, Iași

Facultatea de Automatică și Calculatoare

Domeniul Calculatoare și Tehnologia Informației

Specializarea Calculatoare

INTELIGENȚĂ ARTIFICIALĂ

"Jocul NIM"

Algoritmul minimax cu retezarea alfa-beta aplicat jocului Nim

Coordonator,
Prof. dr. ing.
Mircea Hulea

Studenti,
Spataru Alexandru - 1405B
Versanu George-David - 1405B

An universitar 2024-2025

Cuprins

Descrierea problemei considerate
1.1. Prezentarea jocului Nim
1.2. Obiectivele implementării

Aspecte teoretice privind algoritmul
2.1. Algoritmul Minimax
2.2. Retezarea alfa-beta

Modalitatea de rezolvare
3.1. Structura soluției
3.2. Funcția de evaluare
3.3. Calcularea următoarei poziții a calculatorului
3.4. Algoritmul Minimax cu retezare alfa-beta

Exemple semnificative din implementare
4.1. Funcția pentru calculul Minimax cu retezarea alfa-beta
4.2. Funcția pentru determinarea mutării botului
4.3. Funcția pentru afișarea grămezilor
4.4. Funcția pentru actualizarea statusului jocului
4.5. Funcția bot_decide
4.6. Funcția de generare a grămezilor
4.7. Funcția de pornire a jocului
4.8. Funcția de afișare a meniului
4.9. Funcția de afișare a setărilor

Rezultatele obținute
5.1. Prezentarea interfeței grafice
5.2. Setarea mediului de joc
5.3. Startul jocului
5.4. Mutarea botului

Concluzii
6.1. Performanța algoritmului Minimax cu tăiere alfa-beta
6.2. Limitări și îmbunătățiri posibile

Bibliografie

Rolul fiecărui membru al echipei

❖ *Descrierea problemei considerate*

Jocul Nim este un joc strategic în care doi jucători iau pe rând bețe din mai multe grămezi, având libertatea de a alege câte bețe să ia dintr-o singură grămadă, într-un interval definit. Scopul este de a evita să iei ultimul băț, care semnifică pierderea jocului. Implementarea de față permite jucătorului să concureze împotriva unui bot inteligent care utilizează algoritmul Minimax cu retezarea alfa-beta pentru a lua decizii optime. De asemenea, jocul oferă o interfață grafică pentru o experiență mai plăcută utilizatorului.

❖ *Aspecte teoretice privind algoritmul*

Algoritmului minimax este un algoritm recursiv pentru a găsi cea mai bună mișcare într-o situație dată. Algoritmul minimax constă dintr-o funcție de evaluare poziționalã care măsoară bunătatea unei poziții (sau a stării de joc) și indică cât de dorit este ca jucătorul dat să atingă acea poziție; jucătorul face apoi mișcarea care minimizează valoarea celei mai bune poziții atinse de celălalt jucător.

Cei doi jucători sunt numiți maximizant și minimizant. Maximizantul încearcă să obțină cel mai mare scor posibil, în timp ce minimizantul încearcă să obțină cel mai mic scor posibil. Dacă asociem fiecărei table de joc un scor de evaluare, atunci unul din jucători încearcă să aleagă o mutare care să îi maximizeze scorul, iar celălalt alege o mutare care are un scor minim, încercând să contra-atace.

Vom considera jocul ca fiind de sumă nulă, ceea ce înseamnă că aceeași funcție de evaluare poate fi aplicată ambilor jucători.

- Dacă $f(n)>0$, poziția n este avantajoasă pentru calculator și nefavorabilă pentru om.
- Dacă $f(n)<0$, poziția n este dezavantajoasă pentru calculator și favorabilă pentru om.

Astfel, se calculează funcția de evaluare pentru frunze și se propagă evaluarea în sus, selectând minimele pe nivelul minimizant (decizia omului) și maximele pe nivelul maximizant (decizia calculatorului).

Algoritmul de tăiere alfa-beta îmbunătățește Minimax eliminând din analiză subarborii inutili. Dacă o mutare m este mai slabă decât cea mai bună mutare curentă, restul variantelor sale nu mai sunt evaluate. Parametrul alfa reprezintă cea mai bună valoare garantată pentru maximizator, iar beta pentru minimizator. Pe măsură ce arborele este parcurs, alfa și beta se actualizează, iar ramurile care nu pot îmbunătăți rezultatul sunt ignorate, economisind timp și resurse.

❖ *Modalitatea de rezolvare*

Rezolvarea jocului Nim din acest script se bazează pe implementarea algoritmului Minimax cu optimizare prin tăiere alfa-beta pentru deciziile botului. Soluția este implementată astfel:

1. **Algoritmul Minimax**
   - **Scop**: Determină mutarea optimă pentru bot, asigurând cel mai bun rezultat posibil împotriva unui adversar care joacă perfect.
   - **Funcționare**:
     - Jocul este modelat ca un arbore de decizii.
     - **Nivelurile arborelui**:
       - Nivelurile corespunzătoare jucătorului sunt de **maximizare** (botul încearcă să câștige).
       - Nivelurile corespunzătoare adversarului sunt de **minimizare** (adversarul încearcă să câștige).
     - Se evaluează toate mutările posibile, simulând jocul până la o stare terminală sau până la o adâncime prestabilită.
   - **Funcția de evaluare**:
     - Returnează:
       - `+1`: Dacă botul se află într-o poziție câștigătoare.
       - `-1`: Dacă adversarul se află într-o poziție câștigătoare.
       - `0`: Dacă poziția este neutră (nu oferă avantaje clare).
2. **Optimizarea prin Tăiere Alfa-Beta**
   - **Scop:** Reduce numărul de mutări evaluate în arborele de decizii fără a afecta rezultatul final.
   - **Mecanism:**
     - α (alfa): Cea mai bună valoare găsită pentru bot de-a lungul ramurilor analizate.
     - β (beta): Cea mai bună valoare găsită pentru adversar.
     - Dacă în timpul evaluării unei ramuri se constată că o altă ramură este mai favorabilă, evaluarea curentă este întreruptă.
3. **Strategia Botului**
   - **Euristică**: Botul decide în funcție de situația curentă și adâncimea permisă a evaluării:
     - Adâncimea este limitată pentru a menține performanța (de ex., depth = total_bete // 3).
     - Mutarea optimă este determinată de funcția minimax, care ia în considerare toate posibilitățile de joc.
4. **Structura Jocului**
   - **Generarea grămezilor:**
     - Se distribuie un număr total de bețe în mod uniform între grămezi, cu variații aleatorii pentru diversitate.
   - **Reguli de joc:**
     - Jucătorii pot lua între 1 și max_picks bețe dintr-o singură grămadă.
     - Jocul continuă până când toate grămezile sunt goale, moment în care:
       - Jucătorul care ia ultimul băț pierde.

❖ ***Părți semnificative***

Funcția pentru calculul Minimax cu retezarea alfa-beta:

```python
def minimax(depth, is_maximizing, sticks, alpha, beta, max_picks):
    if sticks == 0:
        return -1 if is_maximizing else 1

    if depth == 0:
        return 0

    if is_maximizing:
        max_eval = -math.inf
        for i in range(1, min(max_picks, sticks) + 1):
            eval = minimax(depth - 1, is_maximizing: False, sticks - i, alpha, beta, max_picks)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = math.inf
        for i in range(1, min(max_picks, sticks) + 1):
            eval = minimax(depth - 1, is_maximizing: True, sticks - i, alpha, beta, max_picks)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return min_eval
```

Aceasta funcție implementează algoritmul Minimax pentru a evalua stările jocului și a decide mutarea optimă. Ea returnează 1, -1, sau 0 în funcție de evaluarea stării jocului pentru bot sau adversar.

Funcția pentru determinarea mutării botului:

```python
def bot_move(sticks, max_picks, max_depth, factor=3):
    depth = min(max_depth, max(1, sticks // factor))
    best_move = None
    best_value = -math.inf
    for i in range(1, min(max_picks, sticks) + 1):
        value = minimax(depth, is_maximizing: False, sticks - i, -math.inf, math.inf, max_picks)
        if value > best_value:
            best_value = value
            best_move = i
    return best_move
```

Aceasta decide numărul de bețe pe care botul ar trebui să le ia, utilizând algoritmul Minimax. Returnează numărul optim de bețe pe care botul ar trebui să le ia.

Funcția pentru afișarea grămezilor :

```
def display_gramezi(gramezi, frame):
    for widget in frame.winfo_children():
        widget.destroy()

    max_per_row = 5
    for idx, sticks in enumerate(gramezi):
        row = idx // max_per_row
        col = idx % max_per_row
        pile_frame = tk.Frame(frame, bg="lightblue", bd=2, relief="groove")
        pile_frame.grid(row=row, column=col, padx=15, pady=15)

        tk.Label(
            pile_frame,
            text=f"Gramada {idx + 1}: {sticks} bete",
            font=("Arial", 12, "bold"),
            bg="lightblue"
        ).pack()

        # Afisare pe linii de câte 7 bețe
        stick_frame = tk.Frame(pile_frame, bg="lightblue")
        stick_frame.pack()
        for i in range(sticks):
            if i > 0 and i % 7 == 0:
                stick_frame = tk.Frame(pile_frame, bg="lightblue")
                stick_frame.pack()
            tk.Label(
                stick_frame,
                text="\u25A0",
                font=("Arial", 18),
                bg="green",
                width=2,
                height=1
            ).pack(side=tk.LEFT, padx=2)
```

Funcția display_gramezi primește o listă ce conține numărul de bețe din fiecare grămadă și afișează vizual grămezile și bețele rămase în interfața grafică. În final creează elemente vizuale pentru fiecare grămadă și afișează bețele ca simboluri grafice.

Funcția pentru actualizarea statusului jocului:

```
def update_game_state(gramezi, max_picks, depth, player_turn, frame, inputs_frame):
    if sum(gramezi) <= 0:
        if player_turn:
            messagebox.showinfo( title: "Joc terminat",  message: "Ai pierdut! Botul a castigat.")
        else:
            messagebox.showinfo( title: "Joc terminat",  message: "Felicitari! Ai castigat impotriva botului.")
        frame.master.destroy()
        return

    display_gramezi(gramezi, frame)

    for widget in inputs_frame.winfo_children():
        widget.destroy()

    if not player_turn:
        bot_choice, grămadă_idx = bot_decide(gramezi, max_picks, depth)
        gramezi[grămadă_idx] -= bot_choice
        messagebox.showinfo( title: "Botul a mutat",  message: f"Botul a luat {bot_choice} bete din gramada {grămadă_idx + 1}.")
        update_game_state(gramezi, max_picks, depth,  player_turn: True, frame, inputs_frame)
    else:
        tk.Label(inputs_frame, text="Introduceti gramada :", font=("Arial", 12), bg="lightblue").pack(pady=5)
        pile_input = tk.Entry(inputs_frame, font=("Arial", 12))
        pile_input.pack(pady=5)
        tk.Label(inputs_frame, text="Introduceti numarul de bete:", font=("Arial", 12), bg="lightblue").pack(
            pady=5)
        stick_input = tk.Entry(inputs_frame, font=("Arial", 12))
        stick_input.pack(pady=5)

        def on_player_move():
            try:
                grămadă_idx = int(pile_input.get()) - 1
                sticks_to_remove = int(stick_input.get())

                if 0 <= grămadă_idx < len(gramezi) and 1 <= sticks_to_remove <= min(max_picks, gramezi[grămadă_idx]):
                    gramezi[grămadă_idx] -= sticks_to_remove
                    update_game_state(gramezi, max_picks, depth,  player_turn: False, frame, inputs_frame)
                else:
                    messagebox.showerror( title: "Eroare",  message: "Introdu o gramada valida si un numar valid de bete.")
            except ValueError:
                messagebox.showerror( title: "Eroare",  message: "Introdu o gramada valida si un numar de bete.")

        tk.Button(inputs_frame, text="Mutare", font=("Arial", 14), bg="#ADD8E6", command=on_player_move).pack(pady=10)
        tk.Button(inputs_frame, text="Înapoi", font=("Arial", 14),  bg="#ADD8E6",command=lambda: [inputs_frame.pack_forget(), show_menu()]).pack(pady=10)
        tk.Button(inputs_frame, text="Exit", font=("Arial", 14),  bg="#ADD8E6",command=root.destroy).pack(pady=10)
```

Această funcție actualizează starea jocului după fiecare mutare și determină câștigătorul dacă jocul s-a terminat.

Funcția bot_decide:

```
def bot_decide(gramezi, max_picks, depth):
    for grămadă_idx, sticks in enumerate(gramezi):
        for i in range(1, min(max_picks, sticks) + 1):
            gramezi_test = gramezi[:]
            gramezi_test[grămadă_idx] -= i
            if minimax(depth,  is_maximizing: False, sum(gramezi_test), -math.inf, math.inf, max_picks) == 1:
                return i, grămadă_idx
    return 1, 0
```

Funcția alege grămada și numărul de bețe pe care botul le va lua, utilizând Minimax și va returna un tuplu cu numărul de bețe de luat și indexul grămezii alese.

Funcția de generare a gramezilor:

```python
def generate_gramezi(total_bete, num_gramezi):
    # Calculează distribuția inițială uniformă
    base_bete = total_bete // num_gramezi
    gramezi = [base_bete] * num_gramezi

    # Distribuie restul de bețe rămase
    bete_ramase = total_bete - sum(gramezi)
    for i in range(bete_ramase):
        gramezi[random.randint( a: 0, num_gramezi - 1)] += 1

    # Introduce o variație aleatorie pentru diversitate
    for i in range(num_gramezi):
        if gramezi[i] > 1:  # Asigură-te că grămada nu devine 0
            schimb = random.randint(-1,  b: 1)  # Mică variație: poate adăuga/scădea 1
            gramezi[i] += schimb
            # Ajustează altă grămadă pentru a păstra totalul constant
            alt_index = random.randint( a: 0, num_gramezi - 1)
            while alt_index == i:
                alt_index = random.randint( a: 0, num_gramezi - 1)
            gramezi[alt_index] -= schimb

    # Asigură-te că toate grămezile au cel puțin un băț
    for i in range(num_gramezi):
        if gramezi[i] <= 0:
            gramezi[i] = 1

    # Amestecă grămezile pentru diversitate suplimentară
    random.shuffle(gramezi)

    # Asigură-te că suma totală e corectă
    assert sum(gramezi) == total_bete, "Suma totală a bețelor nu este corectă!"

    return gramezi
```

Funcția de generare a gramezilor generează grămezile inițiale cu distribuție aleatorie de bețe și returnează o listă cu numărul de bețe din fiecare grămadă.

Funcția de pornire a jocului:

```
def start_game(frame, total_bete, num_gramezi, max_picks):
    if total_bete <= 0 or num_gramezi <= 0 or max_picks <= 0:
        messagebox.showerror( title: "Eroare",  message: "Toate valorile trebuie să fie mai mari ca zero!")
        return

    if total_bete < num_gramezi:
        messagebox.showerror( title: "Eroare",  message: "Numărul total de bețe trebuie să fie cel puțin egal cu numărul de grămezi!")
        return
    if max_picks <= 0 or max_picks > (total_bete - num_gramezi):
        messagebox.showerror( title: "Eroare",  message: "Numărul maxim de bețe a unei mutări este prea mare!\n "
                                       "Vă rog sa aveți maxim: (total_bețe)-(număr_grămezi)")
        return
    gramezi = generate_gramezi(total_bete, num_gramezi)
    depth = max(1, min(10, total_bete // 3))

    frame.pack_forget()
    game_frame = tk.Frame(frame.master, bg="lightblue")
    game_frame.pack(fill=tk.BOTH, expand=True)

    inputs_frame = tk.Frame(frame.master, bg="lightblue")
    inputs_frame.pack(fill=tk.BOTH, expand=True)

    update_game_state(gramezi, max_picks, depth,  player_turn: True, game_frame, inputs_frame)
```

Această funcție inițializează jocul și modulul tkinter cu setările specificate și începe interacțiunea.

Funcția de afișare a meniului:

```
def show_menu():
    def reguli():
        info_text = (
            "Acesta este jocul Nim.\n\n"
            "Reguli:\n"
            "1. Jocul începe cu un număr de grămezi, fiecare conținând un anumit număr de obiecte.\n"
            "2. Jucătorii, pe rând, pot lua unul sau mai multe obiecte dintr-o singură grămadă.\n"
            "3. Scopul jocului poate varia:\n"
            "   - În varianta standard, pierde jucătorul care ia ultimul obiect.\n"
            "   - În varianta misère, câștigă jucătorul care ia ultimul obiect.\n"
            "\nBucurați-vă de acest joc de strategie simplu dar captivant!"
        )
        messagebox.showinfo( title: "Despre", info_text)
    for widget in root.winfo_children():
        widget.destroy()

    frame = tk.Frame(root, bg="#f0f8ff")
    frame.pack(fill=tk.BOTH, expand=True)

    tk.Label(frame, text="Jocul Nim", font=("Arial", 24, "bold"), bg="#f0f8ff").pack(pady=40)

    tk.Button(frame, text="Joaca", font=("Arial", 16), bg="#87CEEB", command=lambda: show_settings(frame)).pack(pady=20)
    tk.Button(frame, text="Informatii", font=("Arial", 14), command=lambda: reguli()).pack(pady=20)
    tk.Button(frame, text="Creatori", font=("Arial", 14),bg="#87CEEB",command=lambda: messagebox.showinfo( title: "Creatori",  message: "Versanu George-David \n Alexandru Spătaru")).pack(
    tk.Button(frame, text="Exit", font=("Arial", 16), bg="#FF6347", command=root.destroy).pack(pady=20)
```

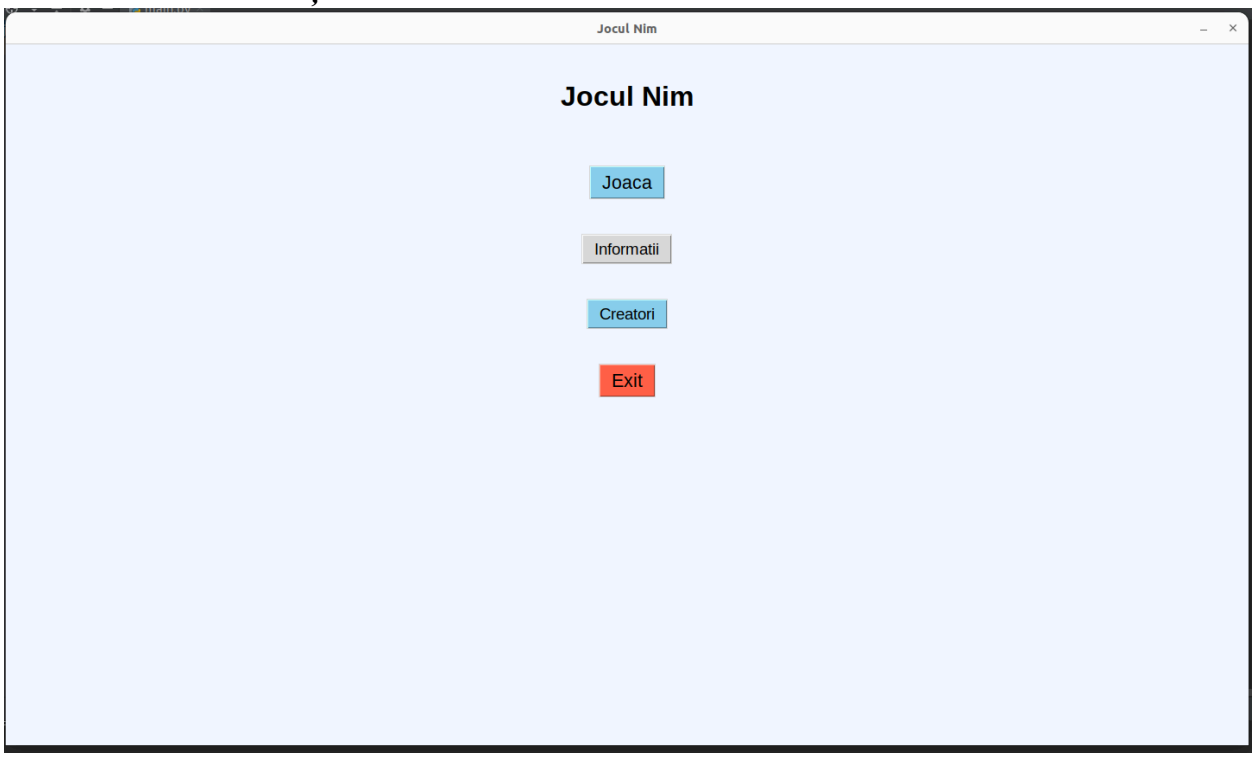
Această funcție creează și afișează meniul principal al jocului și permite navigarea între opțiuni (joacă, informații, exit).

Funcția de afișare a setărilor:

```
def show_settings(frame):
    frame.pack_forget()
    settings_frame = tk.Frame(root, bg="#f0f8ff")
    settings_frame.pack(fill=tk.BOTH, expand=True)

    tk.Label(settings_frame, text="Setari Joc", font=("Arial", 20, "bold"), bg="#f0f8ff").pack(pady=20)

    tk.Label(settings_frame, text="Numar total de bete:", font=("Arial", 14), bg="#f0f8ff").pack(pady=5)
    sticks_entry = tk.Entry(settings_frame, font=("Arial", 14))
    sticks_entry.pack()
    sticks_entry.insert( index: 0,  string: "15")

    tk.Label(settings_frame, text="Numar de gramezi:", font=("Arial", 14), bg="#f0f8ff").pack(pady=5)
    piles_entry = tk.Entry(settings_frame, font=("Arial", 14))
    piles_entry.pack()
    piles_entry.insert( index: 0,  string: "3")

    tk.Label(settings_frame, text="Numar maxim de bete per mutare:", font=("Arial", 14), bg="#f0f8ff").pack(pady=5)
    max_picks_entry = tk.Entry(settings_frame, font=("Arial", 14))
    max_picks_entry.pack()
    max_picks_entry.insert( index: 0,  string: "3")

    tk.Button(settings_frame, text="Incepe", font=("Arial", 14), bg="#32CD32",
              command=lambda: start_game(
                  settings_frame,
                  int(sticks_entry.get()),
                  int(piles_entry.get()),
                  int(max_picks_entry.get())
              )).pack(pady=20)

    tk.Button(settings_frame, text="Inapoi", font=("Arial", 14), bg="#87CEEB",
              command=lambda: [settings_frame.pack_forget(), show_menu()]).pack(pady=10)
```

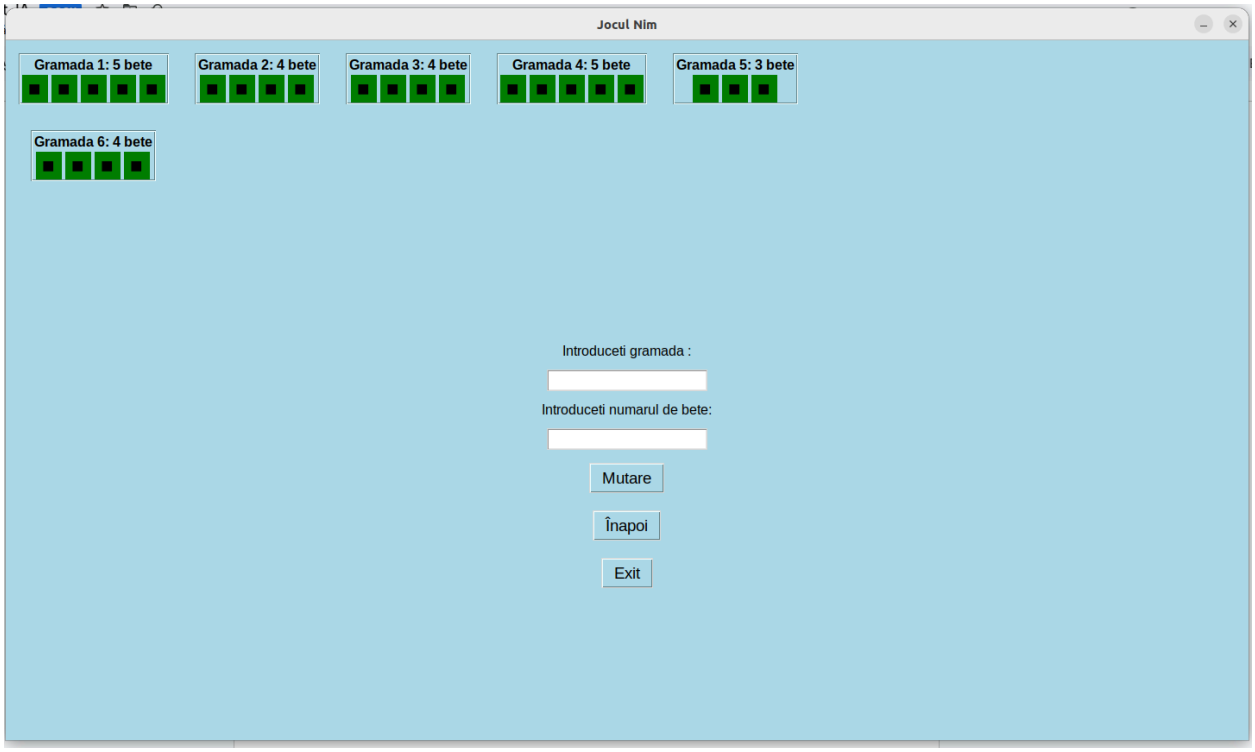
Această funcție afișează interfața pentru configurarea jocului înainte de start.

❖ ***Rezultatele obținute***

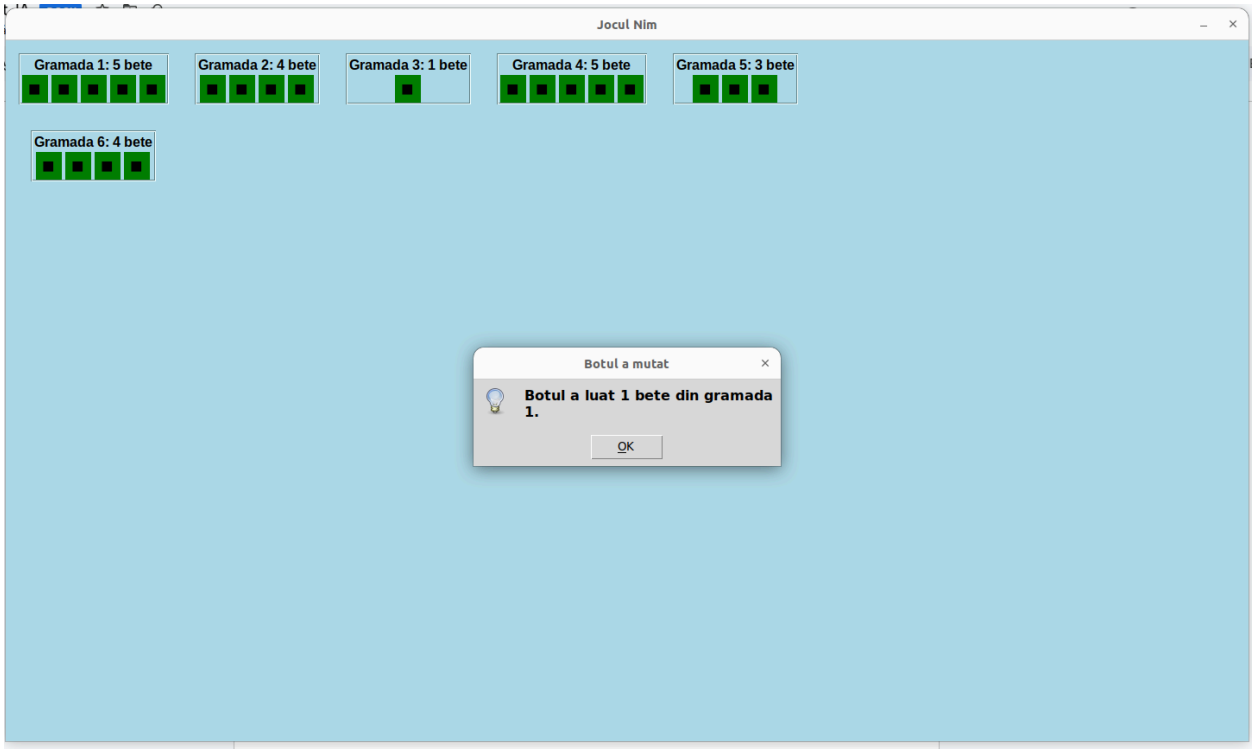


Meniul de start al jocului, cu opțiunile disponibile.

Setarea mediului de joc.



Startul jocului.



Mutarea botului.

Algoritmul Minimax cu tăiere alfa-beta a fost aplicat pentru a dezvolta un joc Nim, cu scopul de a oferi o experiență competitivă utilizatorului. Implementarea a fost orientată către maximizarea performanței botului și oferirea unei interfețe intuitive utilizatorului. Proiectarea a avut în vedere două obiective esențiale:

1. **Maximizarea performanței botului** – prin utilizarea algoritmului Minimax pentru a determina cea mai bună mutare posibilă într-un interval de timp rezonabil, luând în considerare starea curentă a jocului.
2. **Interfața prietenoasă pentru utilizator** – prin afișarea clară a informațiilor despre grămezi și utilizarea unui design simplu și interactiv pentru efectuarea mutărilor.

❖ *Concluzii*

În final, implementarea algoritmului Minimax cu tăiere Alpha-Beta în contextul jocului Nim a reprezentat o provocare interesantă și instructivă. Algoritmul a demonstrat eficiență în luarea deciziilor strategice și în gestionarea complexității deciziilor pe termen scurt. Cu toate acestea, limitările practice, precum reducerea adâncimii arborelui de căutare pentru a asigura un timp de răspuns rezonabil, au generat unele compromisuri în ceea ce privește optimizarea mutărilor botului.

Proiectul a evidențiat importanța echilibrului dintre performanță și complexitate, subliniind valoarea ajustărilor fine pentru îmbunătățirea jocului. Este clar că o abordare adaptivă ar putea oferi un nivel suplimentar de dificultate personalizată, sporind astfel experiența utilizatorului.

În concluzie, această implementare a oferit o perspectivă mai profundă asupra potențialului algoritmilor de căutare și decizie în dezvoltarea de jocuri interactive, demonstrând utilitatea lor în rezolvarea problemelor practice din domeniul inteligenței artificiale.

❖ *Bibliografie*

https://realpython.com/python-minimax-nim/
https://github.com/marcjethro/Nim-Game
https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

❖ *Contribuția fiecărui membru:*

Verșanu George-David:

- Implementarea algoritmilor
- Documentație
- Research
- Fixare și rezolvare de bug-uri

Spătaru Alexandru:

- Interfață grafică
- Documentație
- Implementarea algoritmilor
- Fixare și rezolvare de bug-uri