Dokumentacja projektu: Gra sieciowa "Chomp"

Autor: Mateusz

1. Sformułowanie zadania

Celem projektu było stworzenie gry "Chomp" (gra w czekoladę) dla dwóch osób, wykorzystującej mechanizmy programowania współbieżnego oraz komunikację sieciową.

Szczegółowe wymagania wariantu:

1. **Plansza:** Imitacja tabliczki czekolady. Rozmiar ustalany przez pierwszego gracza (domyślnie 5x4, max 20x20).
2. **Przebieg gry:** Gracze wykonują ruchy na zmianę. Wybór kostki powoduje usunięcie jej oraz wszystkich kostek znajdujących się na prawo i poniżej niej.
3. **Warunek przegranej:** Kostka w lewym górnym rogu jest zatruta. Gracz, który jest zmuszony ją zjeść, przegrywa.
4. **Komunikacja:** Zastosowanie gniazd sieciowych (sockets). Architektura Klient-Serwer.
5. **Interfejs:** Graficzny (GUI).
6. **Koordynacja:** Poprawna synchronizacja wątków i obsługa wielu gier jednocześnie (lobby).

---

2. Schemat komunikacji

System oparty jest na architekturze **Klient-Serwer** wykorzystującej protokół **TCP/IP**.

- **Serwer:** Jest wielowątkowy. Główny wątek nasłuchuje nowych połączeń (accept()), a dla każdego podłączonego klienta tworzony jest osobny wątek obsługujący komunikację (threading.Thread). Serwer przechowuje stan wszystkich aktywnych pokoi (lobby).
- **Klient:** Posiada wątek główny odpowiedzialny za GUI (biblioteka tkinter) oraz wątek poboczny (receive_loop) odpowiedzialny za nasłuchiwanie komunikatów od serwera w tle, co zapobiega "zamrażaniu" interfejsu.

Protokół komunikacyjny

Komunikaty są przesyłane w formie tekstu kodowanego w UTF-8, zakończonego znakiem nowej linii (\n).

Format: KOMENDA parametr1 parametr2 ...

A. Komunikaty od Klienta do Serwera:

Komen	Parametry	Opis
LIST	brak	Żądanie pobrania listy dostępnych pokoi.
CREATE	nazwa wiersze k	Utworzenie nowego lobby o podanych wymiarach.
JOIN	id_lobby	Dołączenie do istniejącego pokoju.
MOVE	wiersz kolumna	Wysłanie ruchu (współrzędne wybranej kostki).
RESTAF	brak	Głosowanie za ponownym rozpoczęciem gry.
LEAVE	brak	Wyjście z gry i powrót do menu.

B. Komunikaty od Serwera do Klienta:

Komenda	Parametry	Opis
LOBBY_LIS	id:nazwa:liczba_ ...	Lista dostępnych pokoi do wyświetlenia.
JOINED	id id_gracza rov	Potwierdzenie dołączenia, przydzielenie ID i wymiarów planszy.
UPDATE	tura stan_plans	Aktualizacja stanu gry. stan_planszy to ciąg zer i jedynek.
GAMEOVE	id_zwyciezcy	Informacja o zakończeniu gry i wyniku.
VOTE_ACC	brak	Potwierdzenie przyjęcia głosu na restart.
OPPONENT	brak	Informacja, że przeciwnik opuścił grę.
ERROR	tresc_bledu	Informacja o błędzie (np. lobby pełne).

---

3. Instrukcja użytkowania

Uruchomienie

1. Należy uruchomić plik serwera: python server.py.
2. Należy uruchomić plik klienta: python client.py (można uruchomić wiele instancji klienta, aby symulować wielu graczy).

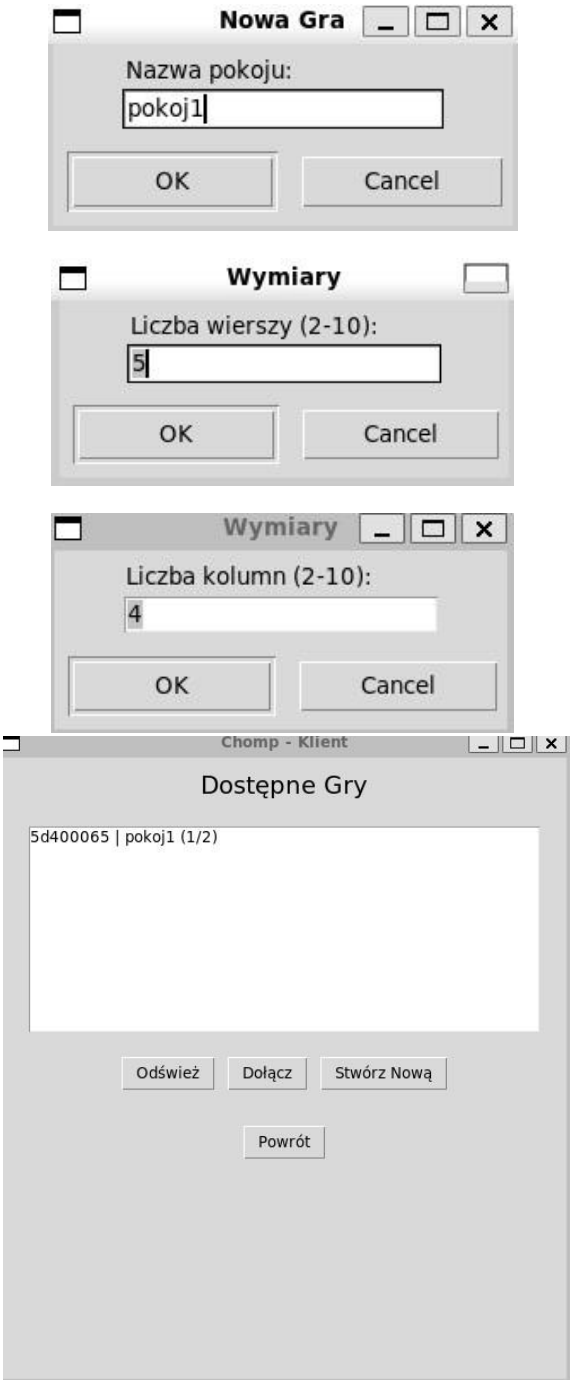
Menu Główne

Po uruchomieniu klienta widoczne jest menu główne.

Lista Lobby i Tworzenie Gry

Po kliknięciu "GRAJ", użytkownik widzi listę aktywnych pokoi.

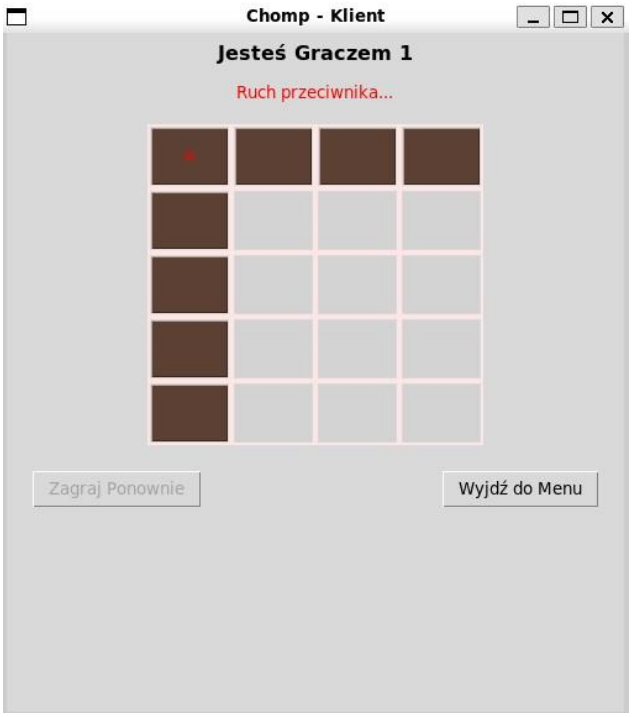
- **Stwórz Nową:** Pozwala zdefiniować nazwę pokoju oraz wymiary planszy (np. 6x5).
- **Dołącz:** Po zaznaczeniu pokoju na liście pozwala do niego wejść (jeśli jest miejsce).
- **Odśwież:** Pobiera aktualną listę pokoi z serwera.



Rozgrywka

Gra rozpoczyna się automatycznie po dołączeniu drugiego gracza.

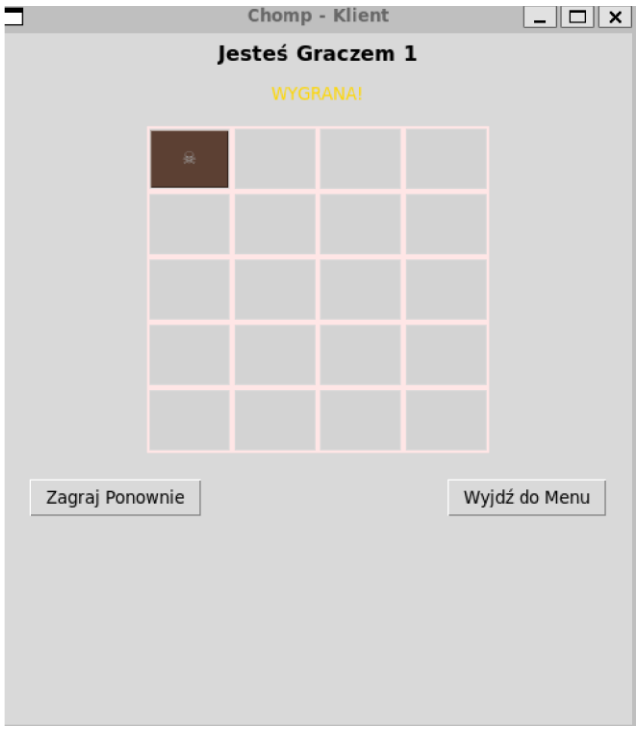
- **Plansza:** Brązowe pola to czekolada, szare to pola zjedzone. Pole z ikoną ☠ to trutka.
- **Ruch:** Gracz klika na wybraną kostkę. Jeśli jest jego tura, kostka i obszar "na prawo i w dół" znikają.
- **Status:** Nad planszą widoczna jest informacja "TWOJA TURA" (na zielono) lub "Ruch przeciwnika..." (na czerwono).



Zakończenie gry

Gdy jeden z graczy zje trutkę, pojawia się komunikat o wygranej/przegranej.

- **Zagraj ponownie:** Obaj gracze muszą kliknąć ten przycisk, aby zrestartować planszę.
- **Wyjdź do Menu:** Powrót do listy serwerów.



Obsługa sytuacji błędnych

Program został zabezpieczony przed typowymi błędami:

- **Zerwanie połączenia:** Jeśli jeden gracz wyjdzie z gry (zamknie okno lub kliknie "Wyjdź"), drugi gracz otrzyma komunikat "Przeciwnik wyszedł z gry" i zostanie przeniesiony do lobby. Serwer automatycznie usunie puste lobby.
- **Błędne wymiary:** Próba stworzenia planszy mniejszej niż 2x2 lub większej niż 10x10 zakończy się komunikatem błędu.
- **Lobby pełne:** Próba dołączenia do gry, w której jest już 2 graczy, zostanie odrzucona przez serwer.

---

4. Fragmenty kodu (Implementacja)

Poniżej przedstawiono kluczowe fragmenty kodu realizujące wymagania projektowe.

1. Serwer - Obsługa wielowątkowości i gniazd

Funkcja start_server nasłuchuje połączeń i dla każdego klienta uruchamia nowy wątek handle_client. Python def start_server():

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server.bind((HOST, PORT)) server.listen() print(f"--- SERWER CHOMP ---")

while True:

conn, addr = server.accept()

Utworzenie nowego wątku dla klienta thread = threading.Thread(target=handle_client, args=(conn, addr))

thread.start()

2. Serwer - Logika gry (algorytm zjadania kostek)

Serwer przetwarza ruch gracza, aktualizuje macierz planszy i rozsyła nowy stan do klientów.

Python

Fragment funkcji handle_client obsługujący ruch if player_id == lobby.turn_idx and not lobby.game_over:

if r == 0 and c == 0:

```
    # Zjedzenie trutki - koniec gry
lobby.game_over = True          winner = 1 -
player_id       lobby.broadcast(f"GAMEOVER
{winner}")
  else:
    # Logika "Chomp" - usuwanie prostokąta
changed = False              for i in range(r,
lobby.rows):
        for j in range(c, lobby.cols):
if lobby.board[i][j] == 1:
lobby.board[i][j] = 0
changed = True

    if changed:
      lobby.turn_idx = 1 - lobby.turn_idx                    # Wysłanie
zaktualizowanego stanu planszy                lobby.broadcast(f"UPDATE
{lobby.turn_idx} {lobby.get_state_str()}")
```

3. Klient - Odbiór danych w tle

Aby GUI nie zacinało się podczas oczekiwania na ruch przeciwnika, odbiór danych odbywa się w osobnym wątku. Python def receive_loop(self): buffer = "" while self.connected:

```
    try:
      data = self.sock.recv(BUF_SIZE).decode()
if not data: break          buffer += data
      # Obsługa przychodzących komend linia po linii
while "\n" in buffer:
          line, buffer = buffer.split("\n", 1)
self.process_server_message(line)
except:          break
```

4. Klient - Interfejs graficzny (Tkinter)

Tworzenie siatki przycisków reprezentujących czekoladę.

```
Python                          def
show_game_interface(self):
  # ... (inicjalizacja ramki)
self.buttons = []     for r in
range(self.rows):
      row_btns = []        for c
in range(self.cols):
        # Każdy przycisk wysyła swoje współrzędne po kliknięciu          btn =
tk.Button(self.board_frame, width=4, height=2, bg="#5C4033",
command=lambda x=r, y=c: self.send_move(x, y))          btn.grid(row=r,
column=c, padx=2, pady=2)          row_btns.append(btn)
self.buttons.append(row_btns)

  #              Oznaczenie              "zatrutej"              kostki
self.buttons[0][0].config(text="☠", fg="red")
```