# Dokumentacja Projektu

## Gra Sieciowa „Labirynt”

Implementacja w języku Python z użyciem gniazd TCP

**Przedmiot:**
Programowanie Współbieżne

**Zadanie:**
Wariant: Labirynt, Sockets, GUI

**Autor:**
Damian Mitros
Nr albumu: 292586

10 stycznia 2026

# 1 Sformułowanie zadania

Zgodnie z wylosowanym wariantem, celem projektu było zaprojektowanie i zaimplementowanie gry sieciowej dla dwóch osób pod tytułem **„Labirynt”**.

## 1.1 Zasady rozgrywki

Gra jest wyścigiem rozgrywanym na planszach o rozmiarze $10 \times 10$. Mechanika gry opiera się na asymetrycznej widoczności:

- **Aktywność gracza (Atak):** Gracz steruje swoim pionkiem wyłącznie na planszy przeciwnika (Sektor Ataku). Jego celem jest odnalezienie drogi do ukrytego skarbu. Plansza ta jest początkowo zakryta i odkrywana w miarę eksploracji.
- **Podgląd zagrożenia (Obrona):** Równocześnie gracz posiada podgląd na własny labirynt, po którym porusza się przeciwnik. Jest to widok pasywny – gracz widzi postępy rywala, ale nie wykonuje na tej planszy żadnych akcji.
- **Generowanie mapy:** Labirynty są generowane losowo. Algorytm gwarantuje powstanie ścieżki o długości 30 pól.
- **Tury:** W każdej turze gracz dysponuje limitem **5 punktów energii**. Ruch kończy się po wyczerpaniu limitu lub po uderzeniu w ścianę.

# 2 Architektura i Schemat Komunikacji

System został zrealizowany w architekturze **Klient-Serwer**, wykorzystując protokół TCP/IP. Zastosowano bibliotekę `socket` do niskopoziomowej obsługi połączeń oraz `threading` do współbieżnej obsługi wielu klientów.

## 2.1 Struktura pakietów (Protokół JSON)

Wymiana danych odbywa się bezstanowo. Każdy komunikat jest obiektem JSON zawierającym pole `type` oraz opcjonalny `payload`. Poniżej przedstawiono przykłady kluczowych komunikatów:

**1. Żądanie ruchu (Klient → Serwer):**

```
{
  "type": "MOVE",
  "direction": "UP"  // Dopuszczalne: UP, DOWN, LEFT, RIGHT
}
```

**2. Wynik ruchu (Serwer → Klient):**

```
{
  "type": "MOVE_RESULT",
  "payload": {
    "status": "MOVED", // Statusy: MOVED, WALL_HIT, TREASURE_FOUND
    "x": 4, "y": 5,    // Nowe współrzędne gracza
    "next_turn": 0     // ID gracza, którego jest teraz kolej
  }
}
```

**3. Synchronizacja gry (Serwer → Klient):**

Wiadomości takie jak `GAME_START` zawierają pełną macierz planszy ($10 \times 10$) oraz identyfikator gracza rozpoczynającego.

### 2.2 Bezpieczeństwo wątkowe

Serwer implementuje mechanizmy synchronizacji, aby zapobiec hazardom (*Race Conditions*):

- Każdy klient obsługiwany jest przez dedykowany wątek (`Thread`).
- Dostęp do współdzielonego obiektu `GameState` (plansze, pozycje graczy) jest chroniony blokadą `threading.Lock()`.
- Wysyłanie wiadomości (Broadcast) odbywa się w sekcji krytycznej, aby zachować kolejność zdarzeń.

## 3 Instrukcja Użytkownika

### 3.1 Uruchomienie i Lobby

Program nie wymaga instalacji zewnętrznych bibliotek. Należy uruchomić kolejno:

1. `python server/server.py` – Serwer nasłuchuje na porcie 65432.
2. `python client/gui.py` (x2) – Dwóch graczy łączy się z serwerem.

Po podłączeniu pierwszego gracza, widzi on ekran oczekiwania (Lobby). Gra startuje automatycznie po dołączeniu drugiego uczestnika.

### 3.2 Obsługa interfejsu

Główne okno aplikacji podzielone jest na trzy sekcje:

- **Panel Lewy (Status Obrony):** Widok pasywny własnego labiryntu. Gracz widzi tutaj fioletowy pionek przeciwnika zbliżający się do skarbu. Służy to wyłącznie do oceny, jak blisko wygranej jest rywal.
- **Panel Środkowy (Obszar Gry):** Główny obszar aktywności. Gracz steruje tutaj swoim (pomarańczowym) pionkiem, odkrywając labirynt przeciwnika. Pola nieznane są dla gracza całkowicie czarne, uderzone ściany oznaczane są kolorem czerwonym, a przebyte ścieżki fioletowym.
- **Panel Prawy (Logi):** Historia ruchów, czat oraz status połączenia.

### 3.3 Sterowanie

Gra obsługiwana jest z klawiatury:

- **Strzałki:** Wybór kierunku (pojawia się przerywany obrys planowanego ruchu).
- **SPACJA:** Zatwierdzenie ruchu.
- **ENTER:** Wysłanie wiadomości na czacie.

### 3.4 Obsługa sytuacji błędnych

Program jest odporny na typowe błędy sieciowe i logiczne:

1. **Zerwanie połączenia (Walkower):** Jeśli jeden z graczy zamknie okno lub utraci połączenie sieciowe, serwer wykrywa wyjątek `ConnectionResetError`, kończy rozgrywkę i ogłasza zwycięstwo drugiego gracza.
2. **Niedozwolony ruch:** Próba wykonania ruchu poza swoją kolejnością lub po wyczerpaniu energii jest blokowana na poziomie klienta (brak reakcji GUI) oraz weryfikowana przez serwer (odrzucenie pakietu).
3. **Brak serwera:** Uruchomienie klienta bez aktywnego serwera skutkuje wyświetleniem komunikatu błędu i bezpiecznym zamknięciem aplikacji.