Gra w Statki

Sieciowa gra dla dwóch graczy

Programowanie Współbieżne 2025 – Zadanie 11

Kajetan Lach

9 stycznia 2026

Spis treści

1 Sformułowanie zadania **2**
1.1 Opis ogólny . . . 2
1.2 Zasady gry . . . 2
1.2.1 Plansza . . . 2
1.2.2 Flota . . . 2
1.2.3 Rozmieszczenie statków . . . 2
1.2.4 Przebieg rozgrywki . . . 2

2 Architektura systemu **3**
2.1 Model klient-serwer . . . 3
2.2 Komponenty systemu . . . 3
2.3 Współbieżność . . . 3

3 Schemat komunikacji **3**
3.1 Protokół transportowy . . . 3
3.2 Format wiadomości . . . 3
3.3 Typy wiadomości . . . 4
3.4 Diagram sekwencji – rozgrywka . . . 4

4 Instrukcja użytkowania **5**
4.1 Wymagania systemowe . . . 5
4.2 Uruchomienie serwera . . . 5
4.3 Uruchomienie klienta . . . 5
4.4 Interfejs graficzny . . . 5
4.5 Legenda kolorów . . . 6

5 Obsługa sytuacji błędnych **6**
5.1 Błędy połączenia . . . 6
5.2 Błędy w trakcie gry . . . 6
5.3 Zamknięcie aplikacji . . . 6

6 Testowanie **6**

1 Sformułowanie zadania

1.1 Opis ogólny

Celem zadania jest zaimplementowanie sieciowej gry w statki (Battleship) dla dwóch graczy z zachowaniem zasad programowania współbieżnego. Rozwiązanie umożliwia rozgrywkę użytkownikom działającym na osobnych komputerach lub w osobnych procesach na jednej maszynie z wykorzystaniem gniazd sieciowych (sockets).

1.2 Zasady gry

1.2.1 Plansza

Gra odbywa się na dwóch kwadratowych planszach o rozmiarze 10×10 pól. Każdy gracz widzi:

- **Własną planszę** – z rozmieszczonymi statkami
- **Planszę strzałów** – widok pola przeciwnika z oznaczonymi trafieniami i pudłami

1.2.2 Flota

Każdy gracz dysponuje następującą flotą:

Typ statku	**Rozmiar (pola)**	**Ilość**
Jednomasztowiec	1	4
Dwumasztowiec	2	3
Trzymasztowiec	3	2
Czteromasztowiec	4	1
Razem	**20 pól**	**10 statków**

1.2.3 Rozmieszczenie statków

- Statki mogą być ustawione w pionie lub poziomie
- Statki **nie mogą się stykać** bokami ani rogami
- Pozycje statków są generowane losowo na początku gry

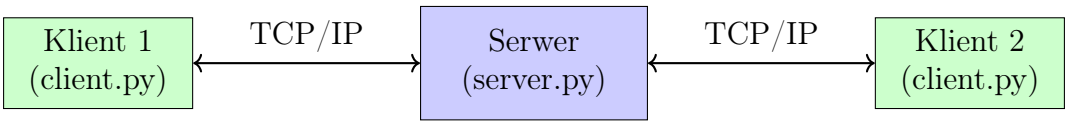
1.2.4 Przebieg rozgrywki

1. **Start:** Grę rozpoczyna losowo wybrany gracz
2. **Tura:** Gracze wykonują ruchy naprzemiennie
3. **Strzał:** Gracz wybiera pole na planszy strzałów
4. **Wynik strzału:**
   - *Trafienie* – gracz kontynuuje turę
   - *Pudło* – tura przechodzi do przeciwnika
   - *Zatopienie* – wszystkie pola statku zostały trafione
5. **Wygrana:** Wygrywa gracz, który pierwszy zatopi wszystkie statki przeciwnika

2 Architektura systemu

2.1 Model klient-serwer

Aplikacja wykorzystuje architekturę klient-serwer:



2.2 Komponenty systemu

server.py Serwer gry – zarządza połączeniami, koordynuje rozgrywkę, przechowuje stan gry

client.py Klient z GUI – interfejs graficzny Tkinter, obsługa interakcji użytkownika

protocol.py Protokół komunikacji – definicje wiadomości, kodowanie/dekodowanie JSON

game_logic.py Logika gry – plansze, statki, zasady, sprawdzanie warunków wygranej

2.3 Współbieżność

System wykorzystuje wielowątkowość do obsługi równoczesnych operacji:

- **Serwer:** Każdy klient obsługiwany w osobnym wątku
- **Klient:** Wątek główny (GUI) + wątek odbierania wiadomości
- **Synchronizacja:** Blokada (lock) chroni dostęp do współdzielonego stanu gry

3 Schemat komunikacji

3.1 Protokół transportowy

Komunikacja odbywa się przez protokół TCP/IP z wykorzystaniem gniazd (sockets):

- **Port domyślny:** 5000
- **Format danych:** JSON z nagłówkiem długości
- **Kodowanie:** UTF-8

3.2 Format wiadomości

Każda wiadomość składa się z:

1. **Nagłówka** (8 bajtów) – długość danych JSON
2. **Danych** – obiekt JSON z polami `type` i `data`

```
{
    "type": "shoot",        # Typ wiadomosci
    "data": {               # Dane zalezne od typu
        "row": 5,
        "col": 3
    }
}
```

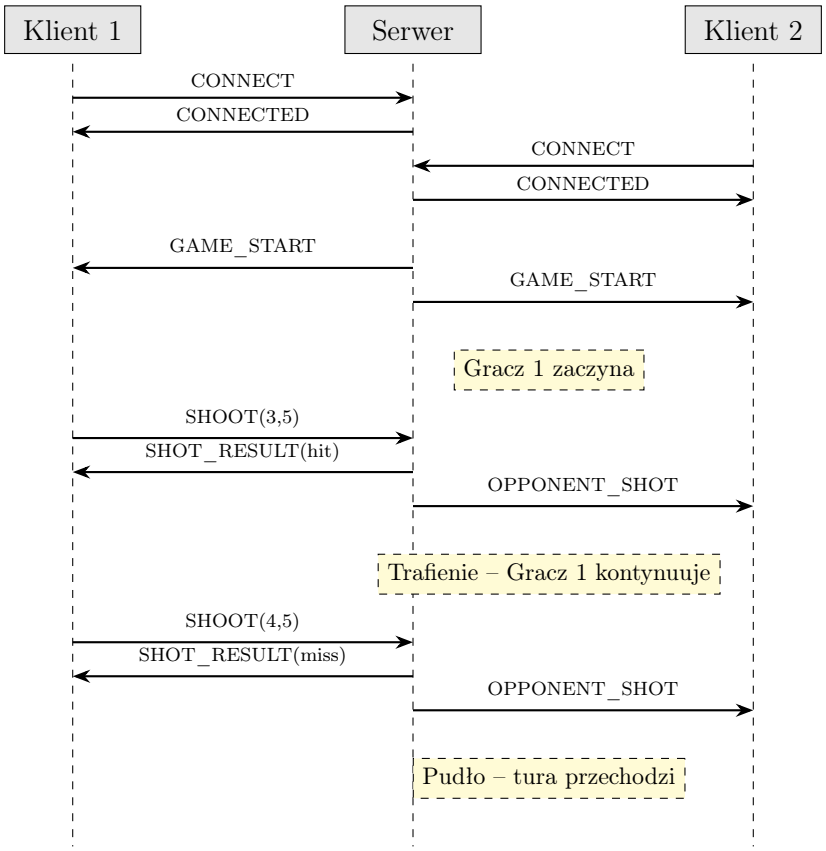
Listing 1: Struktura wiadomości

3.3 Typy wiadomości

Typ	Kierunek	Opis
CONNECTED	S → K	Potwierdzenie połączenia z ID gracza i planszą
GAME_START	S → K	Rozpoczęcie gry, informacja o turze
SHOOT	K → S	Strzał gracza (row, col)
SHOT_RESULT	S → K	Wynik własnego strzału
OPPONENT_SHOT	S → K	Informacja o strzale przeciwnika
GAME_OVER	S → K	Koniec gry z wynikiem
PLAY_AGAIN	K → S	Żądanie rematchu
DISCONNECT	K ↔ S	Rozłączenie
OPPONENT_DISCONNECTED	S → K	Przeciwnik opuścił grę

Legenda: S – Serwer, K – Klient

3.4 Diagram sekwencji – rozgrywka



4 Instrukcja użytkowania

4.1 Wymagania systemowe

- Python 3.8 lub nowszy
- Biblioteka Tkinter (standardowo wbudowana w Python)
- Dostęp do sieci (dla gry między komputerami)

4.2 Uruchomienie serwera

```
cd zad11
python server.py
```

Listing 2: Uruchomienie serwera

Opcjonalne parametry:

- `-H / -host` – adres nasłuchiwania (domyślnie: 0.0.0.0)
- `-p / -port` – port (domyślnie: 5000)

Przykład z niestandardowym portem:

```
python server.py -H 192.168.1.100 -p 8080
```

4.3 Uruchomienie klienta

```
python client.py
```

Listing 3: Uruchomienie klienta

W oknie aplikacji:

1. Wpisz adres serwera (np. `localhost` lub IP serwera)
2. Wpisz port (np. `5000`)
3. Kliknij przycisk **Połącz**
4. Poczekaj na drugiego gracza
5. Gdy gra się rozpocznie, klikaj na pola planszy strzałów

4.4 Interfejs graficzny

Element	Opis
Lewa plansza	Twoja flota (widoczne statki)
Prawa plansza	Plansza strzałów (pole przeciwnika)
Wskaźnik tury	Informuje czyja jest tura
Pasek statusu	Komunikaty o strzałach i zdarzeniach

4.5 Legenda kolorów

Kolor	Znaczenie
Niebieski	Woda (nieodkryte pole)
Stalowy	Statek (widoczny tylko na własnej planszy)
Czerwony	Trafienie
Szary	Pudło
Fioletowy	Zatopiony statek

5 Obsługa sytuacji błędnych

5.1 Błędy połączenia

Sytuacja	Reakcja systemu
Serwer niedostępny	Komunikat: *„Nie można połączyć z serwerem"*
Timeout połączenia	Komunikat po 5 sekundach oczekiwania
Gra pełna (2 graczy)	Komunikat: *„Gra jest pełna"*

5.2 Błędy w trakcie gry

Sytuacja	Reakcja systemu
Strzał w zajęte pole	Komunikat: *„To pole było już ostrzeliwane"*
Strzał nie w swojej turze	Blokada interakcji z planszą strzałów
Rozłączenie przeciwnika	Komunikat + oczekiwanie na nowego gracza
Utrata połączenia z serwerem	Automatyczne rozłączenie + komunikat

5.3 Zamknięcie aplikacji

- Zamknięcie okna klienta powoduje bezpieczne rozłączenie z serwerem
- Serwer informuje pozostałego gracza o rozłączeniu przeciwnika
- Serwer można zatrzymać przez Ctrl+C (sygnał SIGINT)

6 Testowanie

Projekt zawiera automatyczne testy weryfikujące poprawność implementacji:

```
python test_game.py
```

Listing 4: Uruchomienie testów

Testy sprawdzają:

- Generowanie planszy i rozmieszczanie statków
- Protokół komunikacji (serializacja/deserializacja)
- Połączenie z serwerem i wymianę wiadomości
- Prawidłowość zmiany tur