Filip Witkowicz 156169

Sprawozdanie sk2

Sieciowa turowa gra logiczna - Gomoku

1. Opis projektu (0.5 strony)

Projektem jest sieciowa turowa gra logiczna – Gomoku. Gomoku to prosta gra polegająca na stawianiu pionków swojego koloru (są dwa kolory, jeden gracz ma białe, drugi czarne) na planszy NxN (tradycyjnie 15x15 – taka jest też moja implementacja). Zaczyna gracz biały, a następnie wszystkie pionki stawiane są na zmianę. Cel gry jest prosty – wygrywa gracz, który ustawi 5 pionków w rzędzie, kolumnie lub po skosie (zasady gry podobne jak w kółko i krzyżyk). Gra kończy się remisem, gdy zapełnione są wszystkie pola planszy, a żaden gracz nie spełnił celu gry.
Założenia:
- Gracz, który podłączy się pierwszy (czeka na przeciwnika) gra białymi – czyli zaczyna.
- Gdy któryś z graczy rozłączy się, drugi gracz zostanie poinformowany o zwycięstwie – niezależnie jaki był stan planszy w momencie rozłączenia.
- Gdy rozłączy się serwer wszyscy gracze zostają poinformowani o zwycięstwie niezależnie od stanu planszy – w ramach pocieszenia.

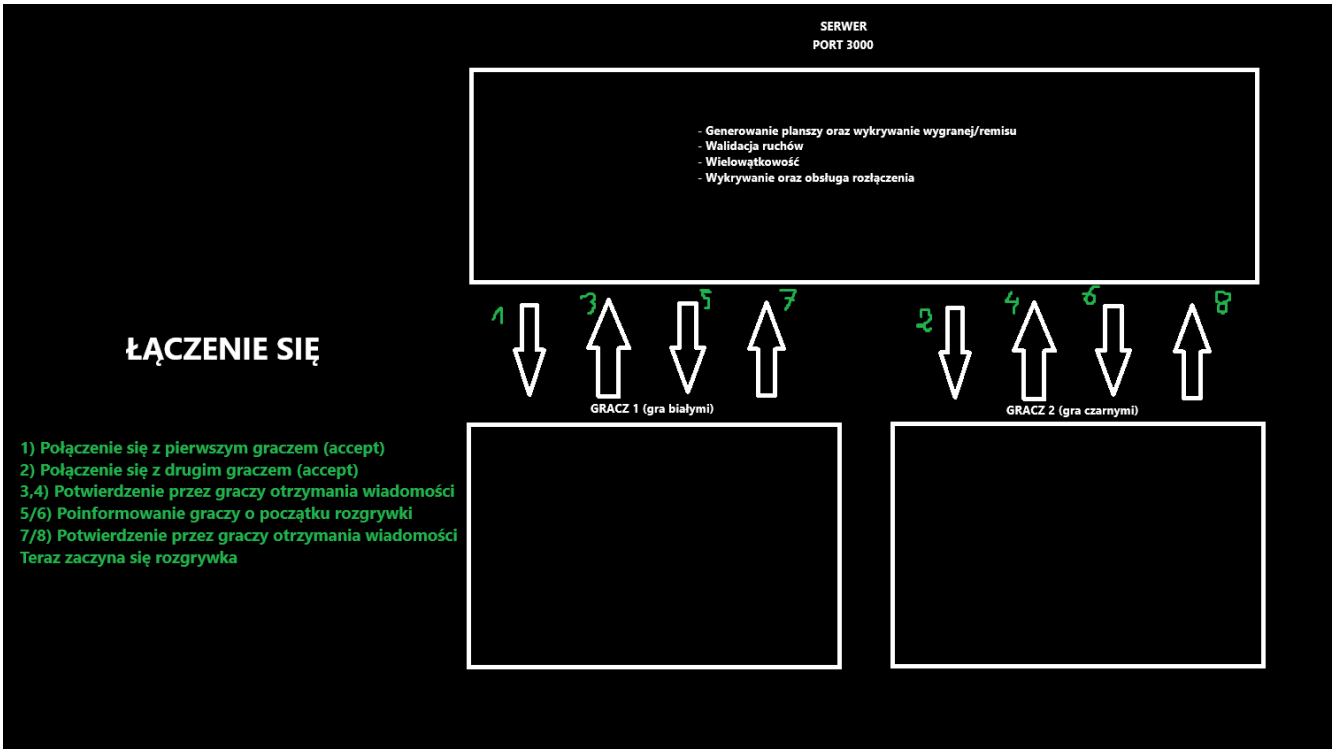
Serwer oraz klient zostały napisane w C (kod kompiluje się przy użyciu kompilatora gcc), sockety oraz wielowątkowość są te same z których korzystaliśmy w trakcie zajęć. Oto lista bibliotek z których korzystam:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include <string.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <unistd.h>

#include <pthread.h>
```

2. Opis komunikacji pomiędzy serwerem i klientem (0.5 strony, może być schemat/rysunek)



Po udanym połączeniu gracze są odsyłani do nowego wątku serwera. Następnie wysyłają ruchy w postaci ‘KOLUMNAWIERSZ’ np. ‘A02’ lub ‘B12’ zaczynając od białego na zmianę. Gracz musi powtarzać swój ruch jeśli ten nie przejdzie walidacji (np. podanie pola już zajętego albo wpisanie niepoprawnego stringa). Na koniec każdej tury, niezależnie od tego kogo był ruch, serwer wysyła informację o tym czy gra toczy się dalej czy któraś ze stron wygrała lub czy nastąpił remis.

3. Podsumowanie (0.5-1 strona)

Komendy kompilacji:

Serwer:
gcc server.c shared.c -o game && ./game

Klient:
gcc player.c shared.c -o player && ./player 127.0.0.1 3000

Projekt składa się z trzech plików .c (player, server, shared) oraz pliku nagłówkowego shared.h. Player zawiera kod klienta. Server kod serwera i gry. Shared zawiera funkcje z których korzystają server i klient (klient korzysta jedynie z funkcji rysowania planszy). Funkcje i define pliku nagłówkowego:

```
// rozmiar tablicy
#define n 15

// Deklaracje funkcji
char checkWinner(char board[n][n]);
void makeBoard(char board[n][n]);
int moveValidation(char board[n][n], int row, int column, int turn);
int sendMsg(int socket, const char* message);
int handleTurn(int playing, int waiting, char board[n][n], int turn);
int checkDraw(char board[n][n]);
```

Gracz zostaje poinformowany o zwycięstwie z powodu rozłączenia, gdy jego socket zostanie zamknięty przed zakończeniem gry. To założenie ma swój minus, ponieważ gdy to serwer się rozłączy to gracz również dostanie informację o zwycięstwie, mimo że drugi gracz nie opuścił rozgrywki z własnej woli.

Główną trudnością napotkaną podczas realizacji projektu był problem natury data race. W moim projekcie wysyłałem kilka wiadomości z serwera do jednego klienta z rzędu np. w momencie łączenia się, przez co czasami 2 sendy wykonywały się przez jednym recv co doprowadzało do niepoprawnego odczytania wiadomości. Problem rozwiązałem wprowadzając acknowledgementy po stronie klienta, wysyłane po odebraniu wiadomości od serwera. Był to jedyny poważniejszy problem napotkany podczas rea