Kamil Grzymkowski 151908

Sprawozdanie sk2

Sieciowa gra szachy

1. Opis projektu

W ramach mojego projektu stworzyłem serwer i klient sieciowej gry w szachy. Zastosowałem klasowy model architektury kodu, w którym każdy bierek odpowiada za walidację własnego ruchu (oprócz przypadków kiedy sprawdzamy czy jest mat czy też gracz próbuje ruszyć się bierką nie swojego koloru). Do zbudowania użyłem g++ oraz make. Kompilacja jest ustawiona na poziom optymalizacji -O3.
Polecenie make wykonane w folderze serwera tworzy plik wykonywalny main który jest serwerem.

serwer zaimplementowany w C++ z użyciem biblioteki <pthread.h>
klient zaimplementowany w dart (język frameworka flutter)
Polecenie: flutter start w folderze klienta uruchamia go

2. Opis komunikacji pomiędzy serwerem i klientem

Serwer pasywnie oczekuje na wiadomości od klientów i komunikuje się z nimi tylko po otrzymaniu jakiejś wiadomości. Socket ma ustawianą flagę TCP_NODELAY więc wiadomości są wysyłane jak najszybciej.

Klient komunikuje się z serwerem za pomocą krótkich pojedynczych wiadomości.

Wszystkie wiadomości wysyłane przez klienta albo przez serwer zaczynają się od bajtu sygnalizującego typ wysyłanej wiadomości, reszta zależy od kontekstu. Poszczególne bajty i wiadomości które kodują:

```
51  enum ServerMessageType : char
52  {
53      LIST_OF_GAMES_INFO = 0,
54      GAME_JOINED,
55      GAME_NOT_JOINED,
56      PLAYER_JOINED_YOUR_GAME,
57      PLAYER_LEFT_YOUR_GAME,
58      GAME_STARTED,
59      GAME_ENDED,
60      GAME_CREATED,
61      GAME_NOT_CREATED,
62      ENEMY_PIECE_MOVE,
63      PIECE_MOVE_CONFIRM,
64      PIECE_MOVE_REJECT,
65      PLAYER_ID,
66  };
```

```
50  enum ClientMessageType : char
51  {
52      GET_LIST_OF_GAMES = 0,
53      SET_PLAYER_NAME,
54      JOIN_GAME,
55      CREATE_NEW_GAME,
56      MOVE_PIECE,
57      LEAVE_GAME
58  };
```

ClientMessageType::GET_LIST_OF_GAMES - wysyła prośbę o przesłanie aktualnej listy gier na serwerze. Serwer w odpowiedzi wysyła wiadomość typu →
ServerMessageType::LIST_OF_GAMES_INFO – informacja o jednej grze zawiera id(unsigned int) gry; id, nazwy graczy(32 bajty) i ich kolory(po bajcie) oraz kto wykonuje aktualnie ruch(1 bajt).

ClientMessageType::SET_PLAYER_NAME - wysyła 32 bajty z nazwą gracza →
ServerMessageType::PLAYER_ID - wysyła 32 bity z nadanym id gracza

ClientMessageType::JOIN_GAME - wysyła 32 bity z id gry do której chcemy dołączyć →
ServerMessageType::GAME_JOINED – odsyła 32 bity z id gry oraz 1 bajt który jest kolorem gracza
ServerMessageType::GAME_NOT_JOINED – jest tylko informacją, że nie udało się dołączyć

ClientMessageType::CREATE_NEW_GAME – prosi o stworzenie nowej gry dla gracza →
ServerMessageType::GAME_CREATED – przesyła 32 bity z id gry i aktualny kolor gracza
ServerMessageType::GAME_NOT_CREATED – jest tylko informacją, że nie udało się stworzyć gry

ClientMessageType::MOVE_PIECE – wysyła request ruszenia się bierką (2 pierwsze bajty pozycja z której się ruszamy, 2 następne pozycja docelowa) →
ServerMessageType::PIECE_MOVE_CONFIRM – informacja zwrotna
ServerMessageType::PIECE_MOVE_REJECT – informacja zwrotna

ClientMessageType::LEAVE_GAME – wysyła request rozłączenia z grą

Poza tymi wiadomościami w odpowiedzi na wiadomość od klienta, serwer może wysłać następujące:
ServerMessageType::PLAYER_LEFT_YOUR_GAME – 1 bajtowa informacja
ServerMessageType::PLAYER_JOINED_YOUR_GAME – wysyła 32 bity id gracza, 32 bajty nazwa gracza i 1 bajt kolor gracza
ServerMessageType::ENEMY_PIECE_MOVE - (2 pierwsze bajty pozycja z której oponent się rusza, 2 następne jego pozycja docelowa)

3. Podsumowanie

Z uwagi na użycie RawSocket we flutterze aplikacja nie działa w przeglądarce (javascript nie obsługuje bsd sockets). Do umożliwienia działania na platformach innych niż linux wymagane byłoby zarządzanie dostępem do pamięci wewnętrznej.

Największymi trudnościami w stworzeniu aplikacji było debugowanie błędów i implementacja silnika szachowego na serwerze.