Dokumentacja Opisowa

Protokół:

Informacje:

100	Waiting for opponent
101	Game begins
102	Game keeps going
103	Player 1 won
104	Player 2 won
105	Draw
106	Game ended
108	Opponent chosen field
110	You are Player 1. Player 1 begins
111	You are Player 2. Player 1 begins
115	Rules: Write letter {A, B, C} and number {1, 2, 3} to define field, e. g. a2
180	Select 0 for sign up or 1 for sign in
181	Type username
182	Type password
194	Opponent turn!
195	Your turn!
196	Return to game
197	Reconnection timeout
198	Opponent reconnection established
199	Opponent disconnected, waiting for connection...

Akceptacja

200	OK
201	Correct field
202	User successfully registered
203	User’s credentials successfully checked

Żądania

300	Change player
310	Play Again
320	Disconnect
331	Register
332	Log in

Błędy

400	Bad field name
401	Field has been already used
402	Username already exists
403	Incorrect credentials
404	Such user is already signed in

Klasy

Game

Atrybuty

- players
    - Utrzymuje relacje obiekt player i socket
    - słownik [klucz: socket] { wartość: obiekt player }
- p_players
    - Przechowuje aktualnie grających graczy
    - słownik [klucz: player.id] { wartość: obiekt player }
- r_players
    - Przechowuje graczy, którzy się odłączyli
    - słownik [klucz: player.id] { wartość: obiekt player }
- w_players
    - Przechowuje graczy oczekujących na grę
    - Kolejka obiektów player
- rooms
    - słownik [klucz: room.id] { wartość: obiekt room }
- rooms_max
    - stała, określa maksymalne możliwe id pokoju
- rooms_min
    - stała, określa minimalne możliwe id pokoju
- MAX_TIME
    - stała, określa czas oczekiwania na ponowne połączenie

Metody

- konstruktor
    - Rozpoczyna wątki dla poniższych metod
- match_maker()
    - Tworzy pokój.
    - Przypisuje do niego oczekujących dwóch graczy
- reconnect()
    - Sprawdza, czy gracz się rozłączył. Następnie oczekuje MAX_TIME na jego powrót, jeżeli nie nastąpi to pozostałego gracza wrzuca do kolejki oczekujących.

Room

Atrybuty

- id
    - Przydzielony identyfikator pokoju
- table
    - Plansza gry słownik [klucz: pole] {wartość: obiekt gracza}
    - Jeżeli pole jest puste, przechowuje None
- player1
    - Obiekt pierwszego gracza
- player2
    - Obiekt drugiego gracza
- current
    - Obiekt gracza, który aktualnie ma ruch
- other
    - Obiekt gracza, który czeka na ruch oponenta

Metody

- konstruktor
    - PARAMETRY: player1, player2, room_id
    - Przypisanie atrybutów i utworzenie planszy
    - Rozpoczyna nowy wątek dla metody start_game()
- start_game
    - PARAMETRY: obiekt self
    - Wysyła reguły gry graczom
    - Pobiera strzelane pola walidując ich poprawność, wysyła komunikaty graczom
    - Przypisuje odpowiedniego gracza w pole planszy
    - Zmienia tury graczom
    - Decyduje o zwycięstwie gracza
- check_field
    - PARAMETRY: room_id, table, field
    - Sprawdza czy pokój nie jest zamknięty
    - Sprawdza, czy dane pole na danej planszy jest puste
    - Zwraca krotkę
        - odpowiednie komunikaty
        - czy udało się strzelić w pole
- check_end_game
    - PARAMETRY: table
    - Sprawdza plansze (poziomo, pionowo, skos)
    - Zwraca krotkę z rezultatem gry oraz graczem, który wygrał - lub null.
        - False - gramy dalej ; True - koniec gry
        - Obiekt zwycięzcy ; None
- end_game
    - PARAMETRY: player1, player2
    - Wyrzuca graczy z pokoju
    - Usuwa graczy ze słownika przechowującego grających graczy

Player

Atrybuty

- id
    - identyfikator gracza
- addr
- socket
- room
    - obiekt pokoju
- opponent
    - obiekt player
- disconnect_time
    - przechowuje pobierany z systemu czas
    - używana do mierzenia czasu reconnect

Metody

- Konstruktor
    - PARAMETRY: socket gracza i jego adres
    - uruchamia wątek dla metody authorize()
- authorize
    - Loguje użytkownika
    - Waliduje wprowadzone dane z danymi z bazy danych
    - Sprawdza, czy zalogowany gracz opuścił poprzednią rozgrywkę
- set_room
- get_room
- set_opponent
- get_opponent
- set_disconnect
- get_disconnect

Baza Danych

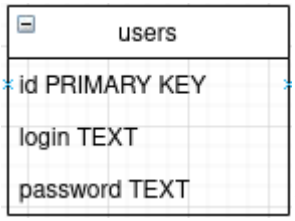
Wykorzystujemy bibliotekę sqlLite3

Atrybuty

- databaseName
    - Nazwa pliku z bazą danych
- con
    - Połączenie z sqlLite
- q_create_table
    - Stała string - zapytanie w języku sql tworzące tabelę
- q_remove_table
    - Stała string - zapytanie usuwające tabelę

Metody

- connect
    - Utworzenie połączenia z bazą danych ; ustawienie atrybutu con
- commit
    - zatwierdzenie zmian w bazie
- close
    - zamknięcie połączenia z bazą
- createTable
    - Ustawia kursor bazy
    - Tworzy tabelę korzystając z zapytania q_create_table
- removeTable
    - Ustawia kursor
    - Usuwa tabelę korzystając z zapytania q_remove_table
- register
    - PARAMETRY: login, password
    - Ustawia kursor
    - Sprawdza czy podany login istnieje w bazie (checkUser)
    - Wykonuje zapytanie
    - zatwierdza zmiany
- checkUser
    - PARAMETRY: login
    - Sprawdza czy podany login istnieje w bazie
    - Wywołuje zapytanie
- login
    - Wyszukuje podanego loginu w bazie
    - Zwraca hasło z bazy, jeżeli użytkownik istnieje



W bazie istnieje jedna tabela: users

SERWER

- Deklaracja context dla szyfrowanego połączenia (podanie kluczy i certyfikatu)
    - serwer posiada swój certyfikat oraz swój klucz prywatny, oprócz tego publiczny certyfikat klienta.
    - Szyfrowanie asymetryczne
    - W momencie kiedy serwer odbiera zdarzenie wejściowe (nadchodzący socket klienta) tworzymy wrap na danym sockecie (linia 567)
- Serwer zdarzeniowy:
    - Wydarzenia wejściowe:
        - Klient chce się połączyć
        - Inne (przysłane dane)
    - Wydarzenia wyjściowe
        - Serwer wysyła dane do konkretnego klienta
    - ATRYBUTY:
        - Tablica zdarzeń wejściowych (sockety)
        - Tablica zdarzeń wyjściowych (sockety)
        - Słownik [klucz: id gracza] { wartość: dane do odebrania }
        - Słownik [klucz: id gracza] { wartość: dane do wysłania }

Krótki opis:
UWAGA: Poniżej formuła słownik dotyczy tylko struktur przedstawionych w serwerze.

Serwer zdarzeniowy posiada dwie tablice socketów:

1. Sockety, które chcą się połączyć z serwerem
2. Sockety, do których serwer chce wysłać dane

Są jednak dodatkowe struktury danych przechowujące dane gracza w relacji z socketem gracza. Stąd ustawiamy socket w tablicy danych do wysłania, następnie dzięki socketowi pozyskujemy id gracza i ustawiamy mu wiadomość do wysłania.

Metody

- send_to_player()
    - PARAMETRY: wiadomość, socket
    - Socketów może być wiele! Ta sama informacja do wielu klientów
    - Wrzucamy do tablicy zdarzeń wyjściowych socket
    - Wrzucamy do słownika dane do wysłania.
- send_to_player_direct()
    - PARAMETRY: wiadomość, socket
    - Socketów może być wiele!
    - Wiadomość jest wysyłana bezpośrednio do klienta, omijając logikę serwera zdarzeniowego.
- rcv()
    - PARAMETRY: id gracza
    - Zwraca dane zawarte słowniku, adekwatne do id
- rcv_direct()
    - PARAMETRY: socket gracza
    - Wiadomość wysyłana bezpośrednio do gracza

KLIENT

Metody

- rcv()
    - PARAMETRY: socket
    - Odbieramy dane
    - Rozważamy
        - odłączenie innego gracza w trakcie gry
        - powrót gracza
        - decyzja czy gramy dalej czy nie
- authorize()
    - PARAMETRY: socket
    - Odbieranie informacji od serwera na etapie logowania
        - walidacja danych
    - Przyjmowanie danych od użytkownika i wysyłanie ich na serwer
- new_game()
    - PARAMETRY: socket
    - Obsługa powrotu do gry
    - Odbieranie informacji o aktualnej turze
    - Ogłoszenie wyników
- endgame()
    - PARAMETRY: socket
    - Wybór, czy użytkownik gra dalej czy kończy rozgrywkę
- disconnect()
    - PARAMETRY: socket
    - Przed zamknięciem klienta, wysyłany jest komunikat o rozłączeniu
- start_game()
    - PARAMETRY: socket
    - wywołanie metody new_game()

Na początku, obudowujemy socket klienta w context SSLSocket dla zabezpieczonego połączenia.

- Certyfikat i klucz prywatny klienta
- Publiczny certyfikat serwera

Następnie przechodzimy do logowania, uruchamiamy pętle gry i obsługujemy rozłączenie.