Dokument projektowy

Spis treści

1. Ogólny opis projektu

Projekt ma na celu utworzenie aplikacji sieciowej, która pozwoli użytkownikom na wspólną grę w Państwa-Miasta. Aplikacja jest tworzona w języku Python, a do obsługi komunikacji sieciowej zostanie użyty moduł - *socket*.

2. Spis członków

- Przemysław Sałek

https://github.com/PituchaAleksander/country-city_game/commits?author=PrzemyslawSalek

- Aleksander Pitucha

https://github.com/PituchaAleksander/country-city_game/commits?author=PituchaAleksander

- Szymon Sala

https://github.com/PituchaAleksander/country-city_game/commits?author=szymix1999

3. Instrukcja obsługi

1. Uruchomienie serwera

```
python server.py
```

2. Uruchomienie klienta

```
python app.py
```

Po wybraniu opcji „stwórz grę” staniemy się hostem nowo utworzonego pokoju do którego mogą dołączyć inni klienci. Klienci dołączają do pokoju za pomocą hasła podanego przez hosta (wybierając opcję „dołącz do gry” i podając hasło). Liczba pokoi oraz liczba graczy w pokoju nie jest ograniczona.

3. Następnie należy postępować zgodnie z poleceniami wyświetlanymi w konsoli i interfejsie graficznym klienta.

4. Przebieg gry

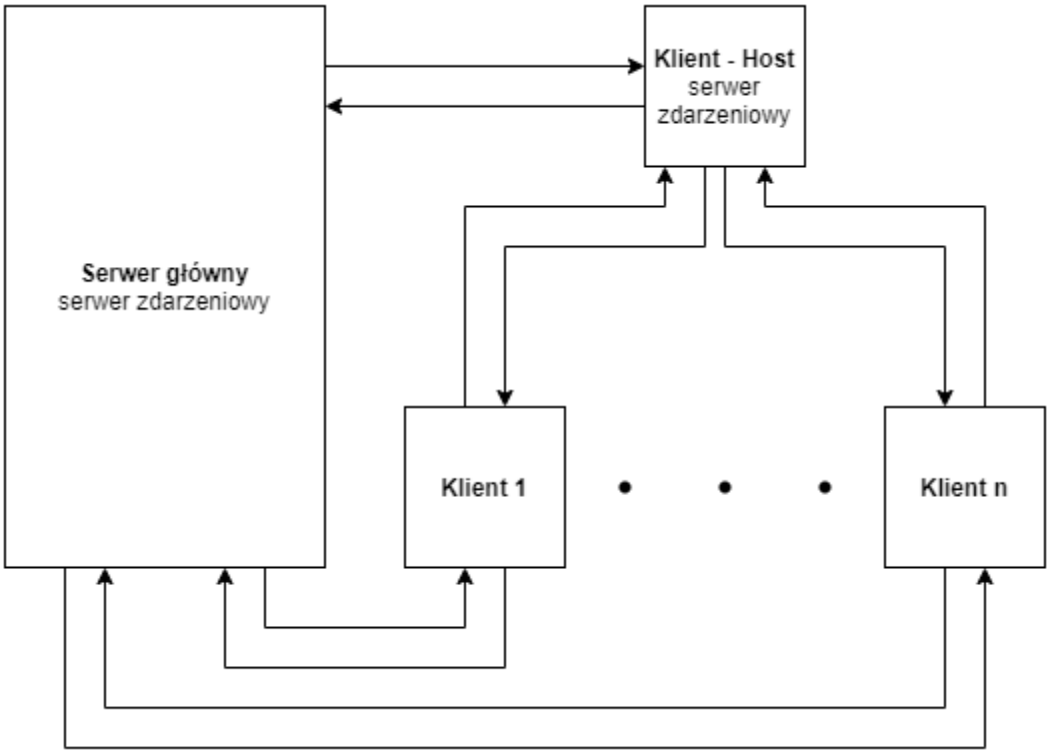
1. Wylosowanie litery przez system gry.
2. Od momentu wylosowania litery wszyscy gracze zaczynają wpisywać słowa zaczynające się na wylosowaną literę – po jednym słowie pasującym do danej kategorii. Wpisywane wyrazy nie powinny być wyrażeniami zbyt ogólnymi. Między innymi błędem jest użycie słowa dinozaur w kategorii zwierzęta. Wpisywanie słów kończy się, gdy skończy się czas przeznaczony na rundę.
3. Zliczanie punktów – wszyscy gracze otrzymują po jednym punkcie za każde poprawne słowo z danej kategorii.
4. Gra toczy się tak długo, jak tylko gracze mają na to ochotę.

5. Struktura sieci

Serwer główny – jest to serwer zdarzeniowy, który przechowuje informacje o utworzonych pokojach. Host (klient-host) wysyła mu informację o utworzeniu pokoju. Zwykły klient po wysłaniu poprawnego hasła pokoju dostaje dane hosta, dzięki którym może się z nim połączyć.

Klient-host – jest to zwykły użytkownik, który utworzył u siebie pokój gry. Jest zarządcą gry, czyli m.in. decyduje kiedy zaczyna się runda. Inni gracze mogą dołączyć do utworzonego przez niego pokoju i grać razem z nim.

Klient – jest to zwykły użytkownik, który może dołączyć do pokoju innego gracza.



6. Protokół – wiadomości

Serwer do klienta

201 CREATED password	Informacja od serwera o utworzeniu pokoju wraz z hasłem.
200 OK	Potwierdzenie przez serwer rozpoczęcia gry.
404 ERROR	Błąd. Nie można rozpocząć gry.
202 EXISTS host_address	Informacja o istnieniu pokoju wraz z adresem hosta.
404 NOT_EXISTS	Błąd. Taki pokój nie istnieje.

Klient do serwera

CREATE_ROOM	Żądanie utworzenia pokoju.
GAME_START password	Informacja o rozpoczęciu gry w danym pokoju.
JOIN password	Żądanie danych potrzebnych do dołączenia do pokoju.

Host do klienta

OK session_id nick	Informacja zwrotna na dołączenie gracza do pokoju wraz z przydzielonym mu identyfikatorem i nazwą hosta.
NOT_OK	Błąd, gracz o takiej nazwie już istnieje.
BAD_SESSION	Informacja o niezgodności identyfikatora sesji.
NEW_PLAYER nick	Informacja o dołączeniu nowego gracza.
ROUND_START letter time	Informacja o rozpoczęciu rundy wraz z literą i czasem o której ma się skończyć.
END_ROUND	Informacja o końcu rundy.
RESULTS results	Wiadomość z serializowanymi wynikami wszystkich graczy w pokoju.
END_GAME	Informacja o końcu gry. Zamknięto pokój.

Klient do hosta

CONNECT nick	Żądanie dołączenia do pokoju wraz ze swoją nazwą.
session_id ANSWERS answers	Wiadomość z własnymi serializowanymi odpowiedziami poprzedzona identyfikatorem sesji.

7. Protokół – kody wiadomości zwrotnych od serwera

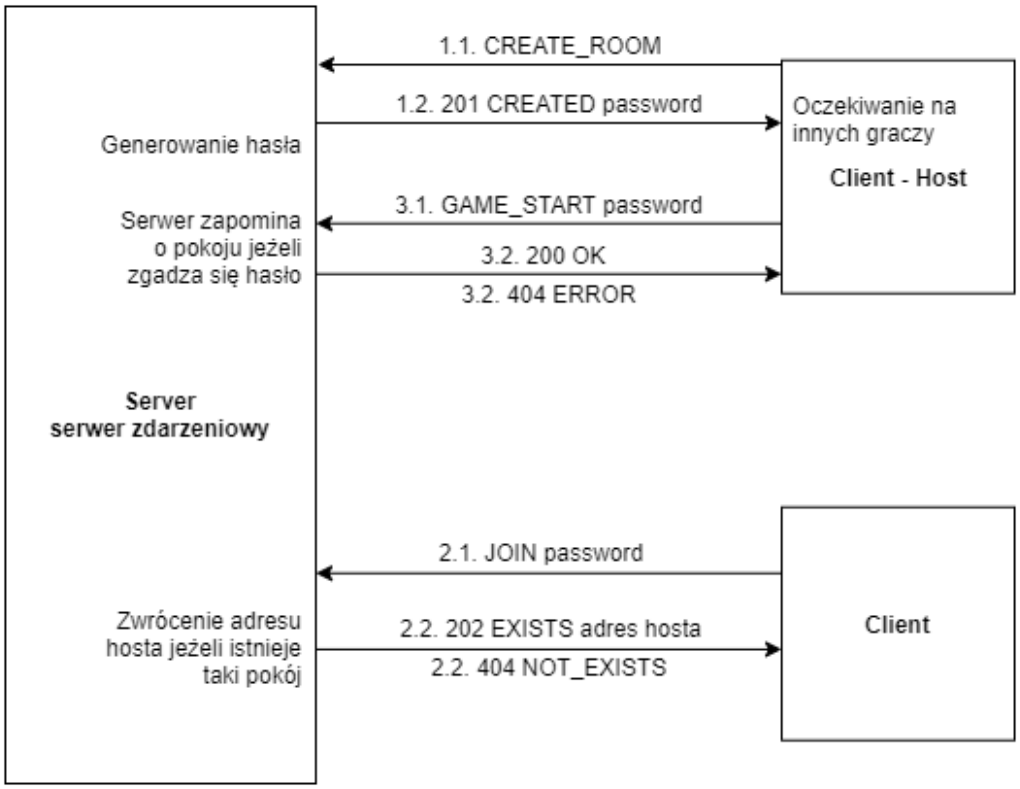
Akceptacja

200	OK
201	CREATED
202	EXISTS

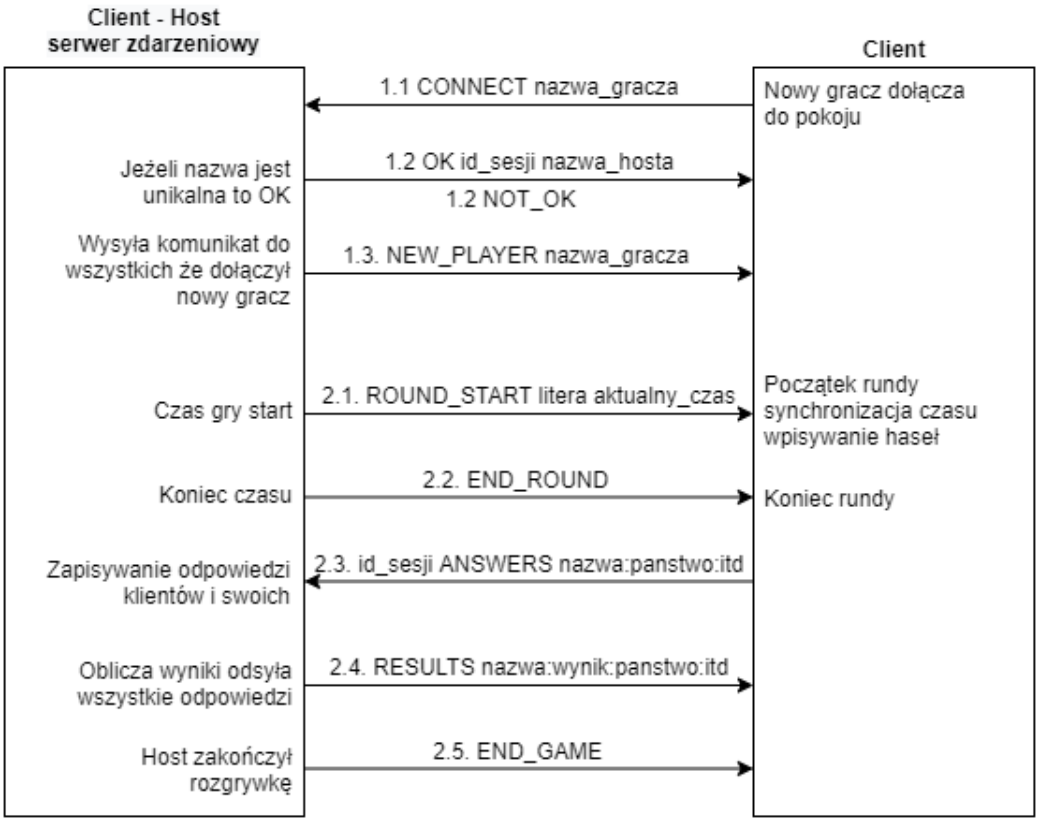
Błędy

404	NOT_EXISTS

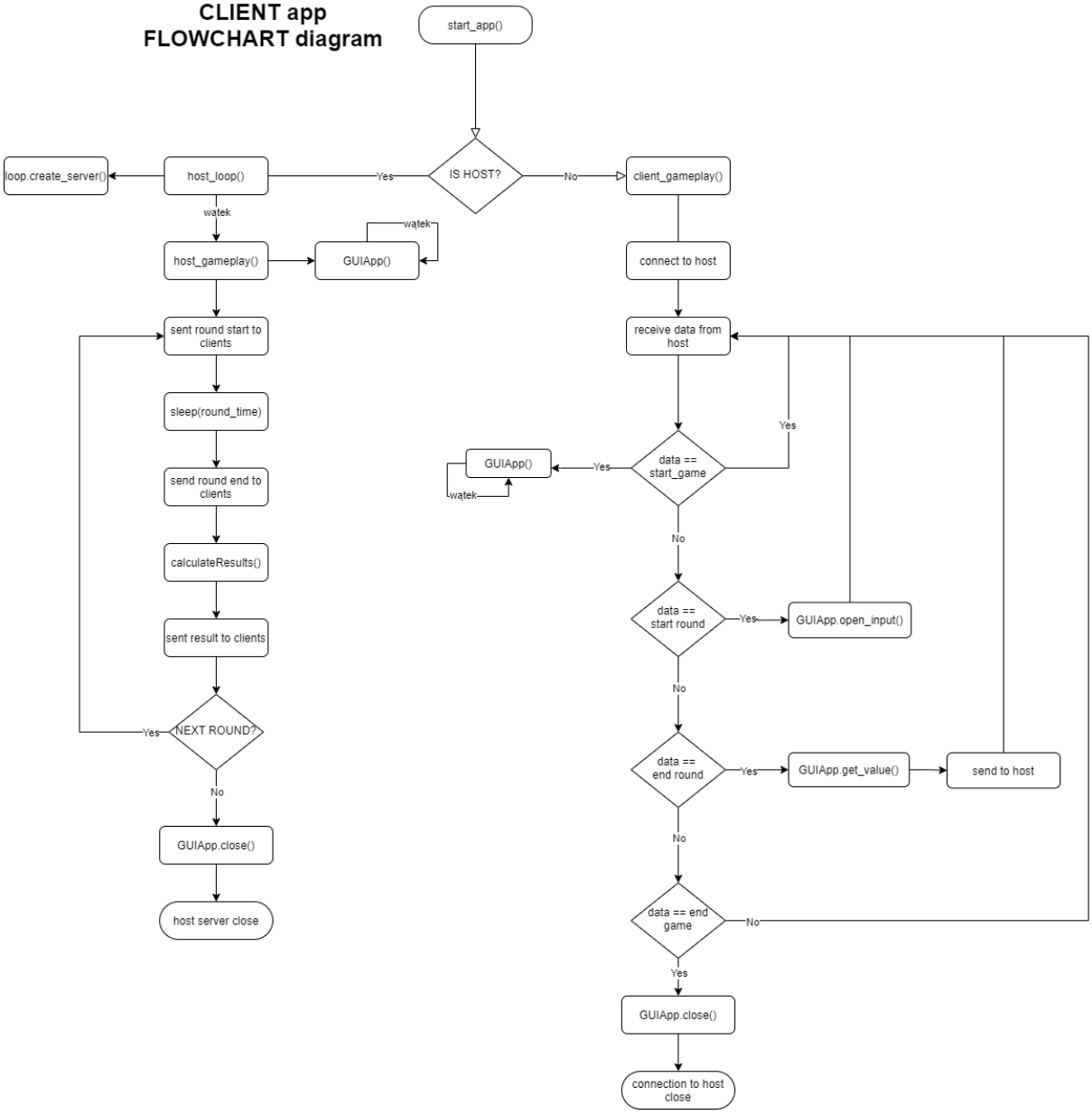
8. Komunikacja serwer – klient (diagram)



9. Komunikacja host - klient (diagram)



10. Schemat blokowy aplikacji



Link do lepszej jakości: https://github.com/PituchaAleksander/country-city_game/blob/main/docs/Schemat%20blokowy%20aplikacji.png

11. Pliki aplikacji

Source - https://github.com/PituchaAleksander/country-city_game/tree/main/source

server.py – zawiera klasę *CountryCityServerProtocol* oraz funkcje i metody odpowiedzialne za działanie serwera zdarzeniowego aplikacji, który przechowuje pokoje i zarządza informacjami o nich.

app.py – plik startowy który posiada funkcje odpowiadające za start programu klienta – gracza.

client.py – plik z funkcjami odpowiedzialnymi za zarządzanie połączeniem klienta z hostem oraz prowadzeniem gry po stronie zwykłego gracza.

host.py – plik w którym mamy serwer zdarzeniowy hosta oraz funkcję odpowiedzialną za prowadzenie gry po stronie hosta i komunikację ze wszystkimi klientami w pokoju.

GUI.py – plik zawierający klasę interfejsu graficznego działającego na własnym wątku zbudowanego z pomocą tkinter. Zawiera wszystkie metody budujące interfejs i zarządzające nim.

gameData.py – plik z klasą *GameData* odpowiedzialną za logikę gry.

categories.py – plik z klasą *Categories* przechowującą informacje o kategoriach i obliczającej wyniki.

Logs - https://github.com/PituchaAleksander/country-city_game/tree/main/logs

server_logs – plik przechowujący logi z lokalnego serwera. Zostaje utworzony wraz z pierwszym uruchomieniem serwera.

12. Struktura plików

- app:
  - Funkcje**:**
    - receive() - funkcja odbioru danych
    - start_app() - odpowiada za start aplikacji, nadanie niku i wybranie opcji (dołączenia lub stworzenia pokoju).

- client
  - Pola:
    - player_data - obiekt z danymi odnośnie rundy (class PlayerData)
    - session_id - identyfikator sesji klienta
  - Funkcje:
    - client_gameplay() - funkcja łączy clienta z hostem i odpowiada za główną pętle gry

- host
  - Pola:
    - game_data - logika gry (class GameData)
    - host_player_data - obiekt z danymi hosta odnośnie rundy (class PlayerData)
    - clients - tablica połączonych klientów
    - responses_from_client - słownik służąca do sprawdzania czy klient wysłał wiadomość
  - Funkcje:
    - notify_clients(message) - rozsyła powiadomienia do klientów
    - host_loop(s) - włącza serwer hosta i pętle gry na oddzielnych wątkach
    - host_gameplay() - pętla gry po stronie hosta
  - (class) HostServerProtocol(asyncio.Protocol):
    - Pola:
      - session_id - identyfikator dla danego połączenia
      - name - nick gracza
    - Metody:
      - data_received() - odbiera dane od klientów i je przetwarza
      - async def async_connect_client() - obsługa pierwszego połączenia klienta
      - async def async_receiving_answers(message) - walidacja sesji i obsługa odpowiedzi od klientów
      - answer_service(self, message) - funkcja wykonywana na oddzielnym wątku obsługa odpowiedzi od klientów

- server
  - Pola:
    - room_info - słownik przechowujący utworzone hasła pokoi i adres hosta
    - filepath - ścieżka do pliku zawierającego logi serwera
  - Funkcje:
    - file_write(log)
    - create_room(address) - zwraca hasło do pokoju
    - join_room(password) - sprawdza czy pokój o danym haśle istnieje i zwraca dane hosta lub None
    - game_start(password) - usuwa pokój o danym haśle z słownika, ponieważ gra hosta się rozpoczęła
  - (class) CountryCityServerProtocol(asyncio.Protocol):
    - Metody:
      - data_received(self, data: bytes)
      - async_create_room(self) -obsługa tworzenia pokoju
      - async_join_room(self, password) -obsługa dołączania do pokoju
      - async_game_start(self, password) -obsługuje komunikat o zaczętej grze od hosta

- playerData
  - (class) PlayerData:
    - Pola:
      - nick
      - score
      - categories - kategorie gry (class Categories)
    - Metody:
      - answers_to_pickle() - zwraca serializowany obiekt self (PlayerData)
      - show_answers_and_save_score(string) - wyświetla odpowiedzi i zapisuje wynik gracza do zmiennej score

- gameData
  - (class) GameData:
    - Pola:
      - password - hasło do pokoju
      - letter - aktualna litera gry
      - scoreboard - tablica z wynikami wszystkich wszystkich graczy
      - round_time - długość rundy w sekundach
    - Metody:
      - set_letter(letter)
      - calculate_results() - oblicza wyniki wszystkich graczy z scoreboard
      - score_board_to_pickle(self) - zwraca pickle z scoreboard
      - add_answers(string, nick) - dodaje odpowiedź (pickle) do scoreboard

- categories
    - (class) Categories:
        - Pola:
            - state
            - city
            - plant
            - animal
            - color
            - name
        - Metody:
            - calculate_score(character, score) - zwarca sume punktów słowo musi zaczynać się tylko na dobrą litere

- GUI
    - (class) GUIApp(threading.Thread):
        - Pola:
            - created - czy GUI zostało zbudowane prawda/fałsz
            - host - nazwa hosta pokoju
            - nick - nazwa gracza
            - round_time - czas rundy wyświetlany na zegarze
        - Metody:
            - callback(self) - niszczy obiekt Tk()
            - get_values(self) - zwraca wartości wypełnionych pól kategorii z GUI
            - clear_fields(self) - czyści pola kategorii w GUI
            - clock(self) - funkcja odpowiedzialna za działanie zegarka
            - is_created(self) - zwraca prawdę/fałsz w zależności czy GUI zostało już zbudowane
            - set_time(self, round_time) - ustawia koniec czasu rundy i uruchamia zegar
            - set_score(self, score) - ustawia wyświetlany wynik w GUI
            - set_letter(self, letter) - ustawia wyświetlaną literę w GUI
            - set_warning(self, warning, color) - ustawia wyświetlany komunikat w GUI
            - start_game(self, letter, round_time) - metoda ustawiająca wszystkie obiekty kiedy zaczyna się runda
            - build_interface(self) - buduje całe GUI
            - run(self) - metoda wywoływana przy tworzeniu obiektu klasy, odpowiedzialna za wywołanie metody budującej i rozpoczęcie pętli głównej GUI