Dokument projektowy

## **Spis treści**

[1. Ogólny opis projektu 2](#_Toc74665137)

[2. Spis członków 2](#_Toc74665138)

[3. Instrukcja obsługi 2](#_Toc74665139)

[4. Przebieg gry 2](#_Toc74665140)

[5. Struktura sieci 3](#_Toc74665141)

[6. Protokół – akcje (wiadomości) 3](#_Toc74665142)

[7. Protokół – kody wiadomości zwrotnych od serwera 4](#_Toc74665143)

[8. Komunikacja serwer - klient (diagram) 5](#_Toc74665144)

[9. Komunikacja host - klient (diagram) 5](#_Toc74665145)

[10. Schemat blokowy aplikacji 6](#_Toc74665146)

[11. Pliki aplikacji 7](#_Toc74665147)

[12. Struktura plików 8](#_Toc74665148)

# 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*.

# Spis członków

* Przemysław Sałek

[https://github.com/PituchaAleksander/country-city\_game/commits?author=PrzemyslawSalek](https://github.com/PituchaAleksander/country-city_game/commits?author=PituchaAleksander)

* Aleksander Pitucha

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

* Szymon Sala

[https://github.com/PituchaAleksander/country-city\_game/commits?author=szymix1999](https://github.com/PituchaAleksander/country-city_game/commits?author=PituchaAleksander)

# Instrukcja obsługi

1. Uruchomienie serwera

python server.py

1. 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.*

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

# Przebieg gry

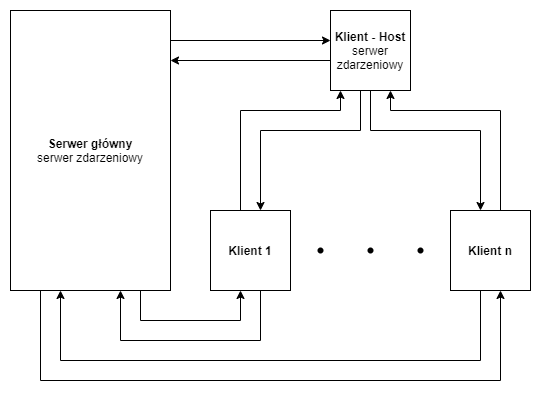
1. Wylosowanie litery przez system gry.
2. Od momentu wylosowania litery wszyscy gracze zaczynają wpisywać słowa zaczynające się na wylosowaną literę – po jednym słowie pasującym do danej kategorii. Wpisywane wyrazy nie powinny być wyrażeniami zbyt ogólnymi. Między innymi błędem jest użycie słowa dinozaur 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.

# Struktura sieci

Serwer główny – jest to serwer zdarzeniowy, który przechowuje informacje o utworzonych pokojach. 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 dołączyć do pokoju innego gracza.



# Protokół – akcje (wiadomości)

Serwer do klienta

|  |  |
| --- | --- |
| CREATED password | Informacja od serwera o utworzeniu pokoju wraz z hasłem. |
| OK | Potwierdzenie przez serwer rozpoczęcia gry. |
| ERROR | Błąd. Nie można rozpocząć gry. |
| EXISTS host\_address | Informacja o istnieniu pokoju wraz z adresem hosta. |
| 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. |
| 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 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 odpowiedziami. |

# Protokół – kody wiadomości zwrotnych od serwera

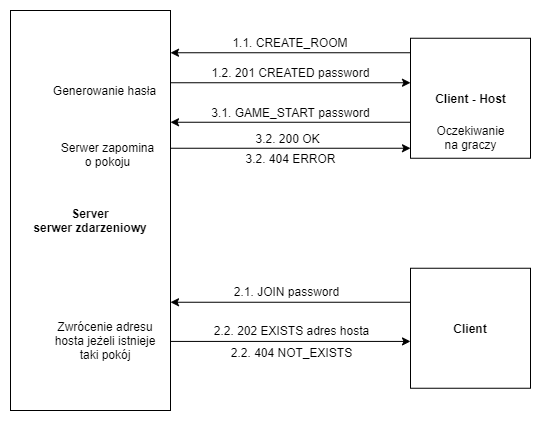
Akceptacja

|  |  |
| --- | --- |
| 200 | OK |
| 201 | CREATED |
| 202 | EXISTS |

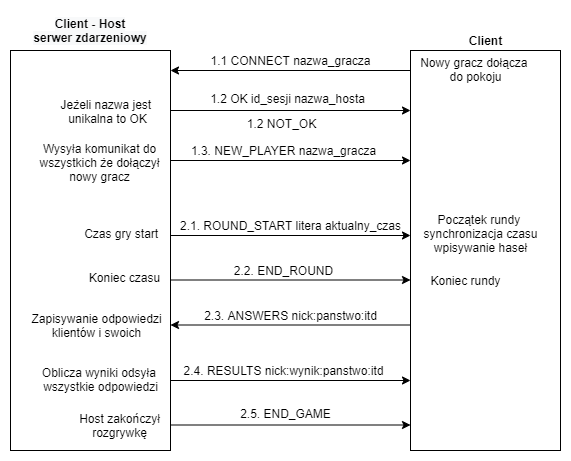
Błędy

|  |  |
| --- | --- |
| 404 | NOT\_EXISTS |

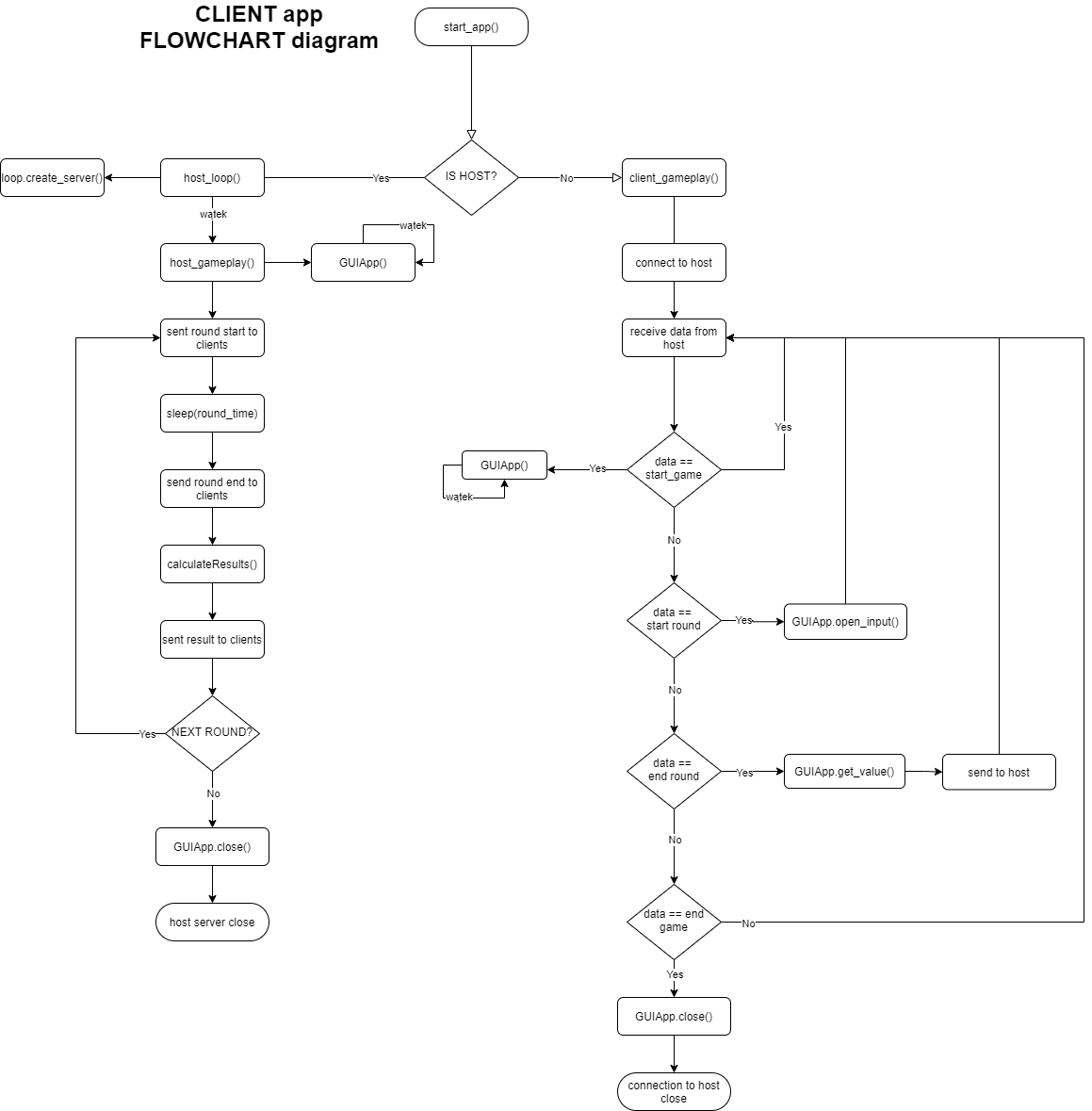
# Komunikacja serwer - klient (diagram)



# Komunikacja host - klient (diagram)



# Schemat blokowy aplikacji



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

# 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.

# 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