Projekt protokołu sieciowego dla aplikacji Sokoban

- **Działanie serwera:**
  1. Serwer jest odpowiedzialny za przechowywanie oraz udostępnianie na polecenie klienta parametrów gry dotyczących: ***listy konfiguracyjnej*** (ilość żyć, poziom początkowy, poziom końcowy, liczba poziomów, parametry okna aplikacji itp.), ***wyglądu plansz***, ***wyglądu elementów graficznych*** (zastosowane awatary postaci oraz elementy znajdujące się na planszy) oraz ***listy najlepszych wyników***.
  2. Do zadań serwera należy również ***przechowywanie*** oraz ***aktualizacja listy najlepszych wyników*** uzyskanych w trybie gry serwerowej.
  3. Serwer jest aplikacją mogącą obsłużyć dowolną liczbę klientów, którzy są obsługiwani w osobnych wątkach.
- **Działanie klienta:**
  1. Po uruchomieniu aplikacji klient może podjąć wybór dotyczący preferowanego trybu gry- tryb sieciowy albo tryb lokalny.
  2. Po rozpoczęciu gry w trybie sieciowym klient wysyła żądanie do serwera o udostępnienie danych zawartych w plikach konfiguracyjnych.
  3. Po zakończeniu rozgrywki w trybie sieciowym klient wysyła żądanie o zapisanie wyniku na ***liście najlepszych wyników***. Jeśli wynik, który osiągnął użytkownik był lepszy od najgorszego wyniku z listy wyników, to serwer dopisuje rezultat gracza do listy.

- **Sposób uruchomienia:**
  W momencie uruchamiania programu należy:
  1. ***Skompilować*** i ***uruchomić serwer*** (do tych czynności mogą posłużyć specjalnie przygotowane programy zawarte w podkatalogu Serwer).
  2. Z uwagi na połączenie serwera i klienta za pomocą numeru portu oraz hosta serwer na samym początku automatycznie generuje plik, gdzie są zawarte wszystkie niezbędne informacje do zaczęcia gry poprzez serwer. ***Wygenerowany plik*** (DaneWyjscioweSerwera.txt, który znajduje się w tym samym miejscu co programy do kompilacji i uruchomienia serwera) ***należy przenieść do folderu*** Klient->src->com->sokoban->config w celu możliwości oczytania go przez serwer.
  3. ***Uruchomić klienta*** i wybrać interesującą nas opcję. Zgodnie z założeniami gra jest również możliwa w trybie offline, bez udziału serwera (wtedy w menu głównym gry należy wybrać opcję Game).

- **Schemat działania protokołu:**
  1. Wszystkie informacje wysyłane przez klienta mają postać pojedynczej linii tekstu. Ze względu na możliwość wyboru różnych wariantów w przypadku wczytywania tekstur, awatarów oraz poziomów, a także określonej odpowiedzi serwera w w/w przypadkach klient wysyła wiadomości zmodyfikowane.
     Przykłady:
     - Aby uzyskać pliki konfiguracyjne klient wysyła żądanie do serwera typu: „Load_config”.

- Aby uzyskać informację dotyczącą najlepszych wyników klient wysyła żądanie do serwera typu: „Load_ranking”.
- Aby uzyskać pliki związane z możliwością wyboru klient wysyła żądanie zmodyfikowane do serwera (nadal jest ona w postaci jednej linii tekstu), np.:
  „Load_avatars!0”- żądanie załadowania zerowego awataru z listy wszystkich awatarów,
  „Load_levels!3”- żądanie załadowania trzeciego poziomu z listy wszystkich poziomów,
  „Load_textures!1”- żądanie załadowania zerowej tekstury z listy wszystkich tekstur.
- Klient w celu zapisania listy wyników może wysłać także inną wiadomość zmodyfikowaną do serwera: „Save_ranking!Imię/Punkty”, np.: „Save_ranking/Ania/100”.
- Jak widać w powyższych przykładach wiadomości, które wysyłamy do serwera są spójne i tworzą logiczną całość. Oprócz polecenia, po znaku „!” znajdują się informacje o przekazywanych danych.
- *W powyższych przykładach każda linia tekstu jest zakończona znakiem końca linii „\n”. Zostały podane tu tylko przykłady wysyłanych wiadomości przez klienta. Szczegółowe omówienie typów znajdzie się w dalszej części tego projektu.*

2. Serwer może odpowiadać klientowi na dwa, różne sposoby. Pierwszym z nich jest odpowiedź za pomocą linii tekstu, a drugą odpowiedź za pomocą tablicy byte (w przypadku awatarów i tekstur).
   - W przypadku odpowiedzi serwera dotyczącej zapisu wyniku gracza do listy najlepszych wyników serwer przesyła wiadomość dotyczącą wyniku tej operacji (czy dany gracz jest lepszy niż najgorszy wynik na tej liście). Odpowiedź jest przesyłana za pomocą pojedynczej linii tekstu: „Huuura! Jestes najlepszy!”.
   - W przypadku żądania klienta dotyczącego udostępnienia listy najlepszych wyników z serwera, serwer również odpowiada pojedynczą linią tekstu typu: „Kasia 100/Basia 60/Kuba 10/”. Poszczególni gracze są oddzieleni znakiem „/”, a nazwy graczy od liczby punktów spacją.
   - W przypadku żądania klienta dotyczącego załadowania plików konfiguracyjnych są one również wysyłane jako pojedyncza linia tekstu typu: „400/300/0/2/1/2/”.
   - W przypadku odpowiedzi serwera dotyczących elementów graficznych są one przekazywane za pomocą obiektu zbudowanego z tablicy byte (następuje tutaj serializacja).
   - W przypadku odpowiedzi serwera dotyczącego żądania o przesłanie poziomów każde kolejne linie tekstu również są przesyłane za pomocą pojedynczej linii tekstu oddzielonej znakiem „/”, np. /######/#@$.#/######
   - Jak widać w powyższych przykładach wiadomości, które odsyłamy jako odpowiedzi do serwera są spójne i tworzą logiczną całość. Zazwyczaj mają one postać przekazywanych danych ze znakiem „/” rozdzielającym poszczególne linie.

- *W powyższych przykładach każda linia tekstu jest zakończona znakiem końca linii „\n”. Zostały podane tu tylko przykłady wysyłanych wiadomości przez serwer. Szczegółowe omówienie typów znajdzie się w dalszej części tego projektu.*

- ***Omówienie poszczególnych żądań klienta i odpowiedzi serwera.***
  Na samym początku następuje połączenie klienta z serwerem poprzez zestaw unikalnych danych: host i port.
  W przypadku wybrania opcji gry serwerowej jest nawiązywane połączenie z serwerem.

  1. **Pliki konfiguracyjne:**
     Prośba klienta o wysłanie początkowych parametrów gry: ilości żyć *(int)*, wymiarów okien *(int)*, ilość kroków na początku gry *(int)*, ilość map *(int)*, pierwsza mapa *(int)*, ostatnia wczytana mapa *(int)*.
     C: Load_config => S
     Przykładowo:
     From client: Load_config

     Serwer odpowiada na prośbę klienta przesyłając 7 parametrów:
     Ilość żyć *(int)*, wymiary okien *(int)*, ilość kroków na początku *(int)*, ilość map *(int)*, pierwsza mapa *(int)*, ostatnia mapa *(int)*. Są one rozdzielone znakiem „/”.
     S: Config => C
     Przykładowo:
     Server respond: 3(Ilość żyć na początku)/800(Szerokość okna)/600(Wysokość okna)/0(Ilość kroków na początku)/4(Ilość wczytanych map)/0(Pierwsza mapa)/3(Ostatnia wczytana mapa)/

  2. **Wczytanie rankingu:**
     Prośba klienta o udzielenie informacji dotyczącej listy najlepszych wyników przechowywanych na serwerze *(String)*. Poszczególne wyniki rozdzielone są za pomocą znaku „/”, a w ramach danego wyniku nazwa gracza od liczby punktów jest rozdzielona za pomocą znaku spacji.
     C: Load_ranking => S
     Przykładowo:
     From client: Load_ranking

     Serwer odpowiada na prośbę klienta udostępniając mu listę najlepszych wyników *(String)*.
     S: Ranking_List => C
     Przykładowo:
     Server respond: Kacper 1418/Basia 1273/Kacper 1000/Kacper 1000/Kacper 1000/Kacper 1000/

  3. **Wysłanie wyniku gracza na serwer:**
     Prośba klienta o zapisanie wyniku gracza na liście najlepszych wyników przechowywanej przez serwer. Wynik gracza od polecenia jest oddzielony znakiem „!”. Nazwa gracza *(String)* od liczby punktów *(int)* jest oddzielona za pomocą znaku „/”.

C: Save_ranking => S
Przykładowo:
From client: Save_ranking!Asia/1200

Serwer odbiera od klienta dane- komendę, nazwę gracza *(String)* i ilość punktów uzyskanych przez gracza *(int)*. Następnie serwer zarządza otrzymaną wiadomością- sortuje listę graczy i sprawdza, czy dany wynik znajduje się w liście najlepszych 6 wyników. Jeśli wynik będzie lepszy od najgorszego wyniku z listy to zapisuje go do pliku. **Zaktualizowana lista najlepszych wyników jest zwracana do klienta podczas następnego żądania wczytania rankingu.** Serwer wysyła informację do klienta czy wynik został zapisany do listy najlepszych wyników.
S: Save_score => C
Przykładowo:
Server respond: Huuura! Jestes najlepszy! (String)

4. **Awatary:**
Klient wysyła prośbę do serwera o udostępnienie mu elementów graficznych mapy- w tym przypadku awatarów postaci. Numer awataru *(int)* jest przekazywany do serwera wraz z komendą i oddzielony od niej za pomocą znaku „!”.
C: Load_avatars => S
Przykładowo:
From client: Load_avatars!1

Serwer odbiera od klienta komendę oraz numer wczytywanego awataru *(int)*. Następnie otwiera odpowiedni awatar z listy dostępnych mu awatarów oraz przetwarza obraz w formacie png na tablicę typu byte, którą odsyła do klienta. Tablica ta, jako obiekt, ulega serializacji i zostaje przekazana do klienta.
S: Load_avatar => C
Przykładowo:
Server respond: [-119, 80, 78, 71, 13, 10, 26, 10, 0, 0, +163,553 more]
Z uwagi na dużą liczbę elementów znajdującą się w tablicy typu byte nie jest możliwe przykładowe wypisanie wszystkich wartości. Następnie ta tablica jest przetwarza i jest tworzony BufferedImage.

5. **Tekstury:**
Klient wysyła prośbę do serwera o udostępnienie mu innych elementów graficznych znajdujących się na planszy, np.: wyglądu bloków, ścian, punktów docelowych dla bloków itp. Numer aktualnie wczytywanej tekstury *(int)* jest przekazywany do serwera wraz z komendą oraz oddzielony od niej za pomocą znaku „!”.
C: Load_textures => S
Przykładowo:
From client: Load_textures!0

Serwer odbiera od klienta komendę oraz numer wczytywanej tekstury *(int)*. Następnie otwiera odpowiedną teksturę z listy dostępnych mu tekstur oraz przetwarza obraz w formacie png na tablicę typu byte, którą odsyła do

klienta. Tablica ta, jako obiekt, ulega serializacji i zostaje przekazana do klienta.
S: Load_textur => C
Przykładowo:
Server respond: [-119, 80, 78, 71, 13, 10, 26, 10, 0, 0, +83,564 more]
Z uwagi na dużą liczbę elementów znajdującą się w tablicy typu byte nie jest możliwe przykładowe wypisanie wszystkich wartości. Następnie ta tablica jest przetwarza i jest z niej tworzony BufferedImage.

6. **Poziomy:**
Klient wysyła prośbę do serwera o udostępnienie mu listy poziomów wczytywanej w ramach tej gry. Numer aktualnie wczytywanej mapy *(int)* jest przekazywany do serwera wraz z komendą oraz oddzielony od niej za pomocą znaku „!”.
C: Load_levels => S
Przykładowo:
From client: Load_levels!0

Serwer odbiera od klienta komendę oraz numer wczytywanej mapy *(int)*. Następnie serwer otwiera odpowiedni plik z mapą i odsyła do klienta wiadomość dotyczącą danej mapy *(String).* Kolejne wiersze mapy są oddzielone od siebie za pomocą znaku „/”.
S: Level => C
Przykładowo:
From client: /#####/#@$.#/#####
Wiadomość odsyłana do klienta jest przez niego przetwarzana i z niej jest tworzona mapa (# to ściana, @ to punkt startowy postaci, $ to bloki do poruszenia, $ to punkt docelowy).