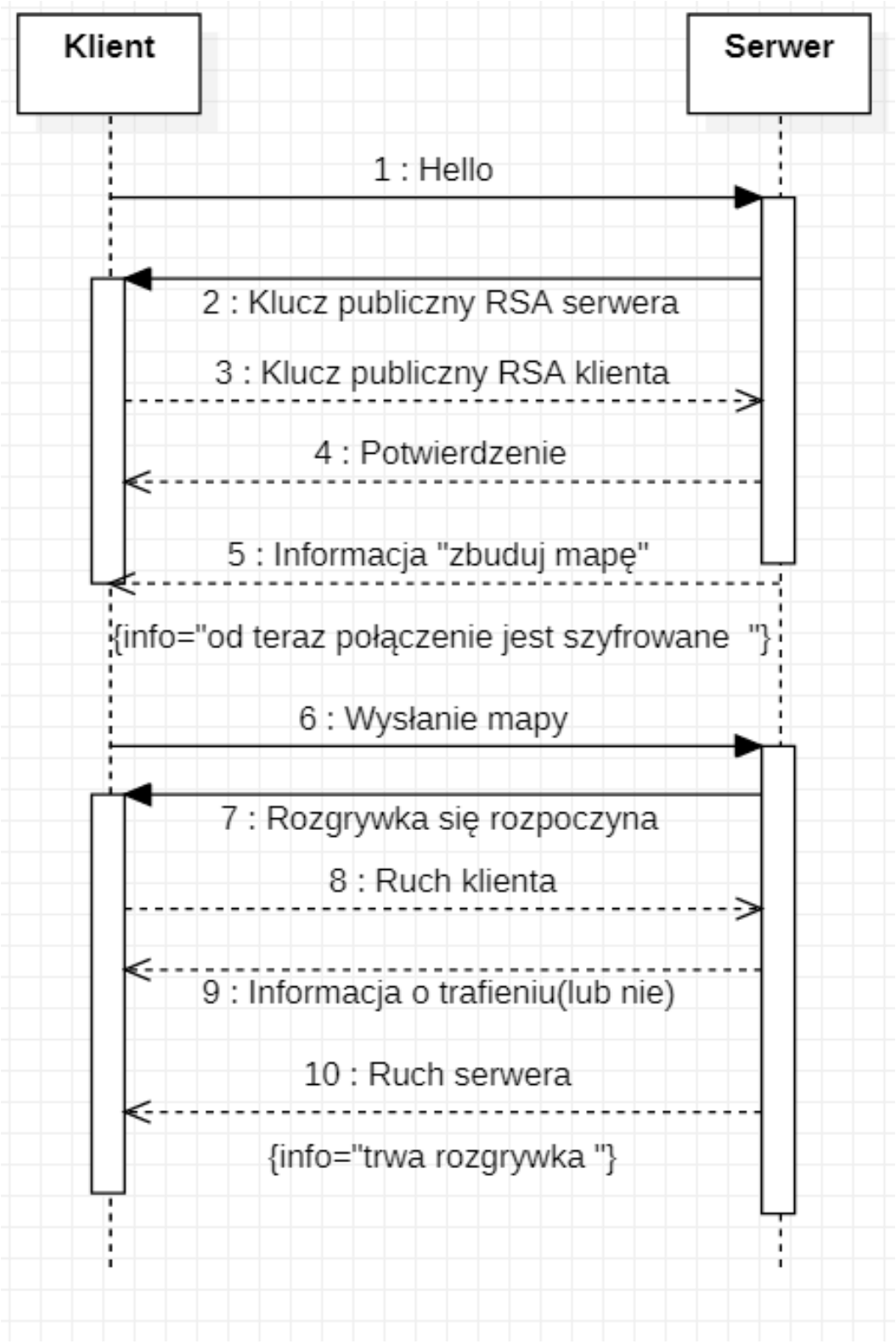
Dokumentacja aplikacji „Gra w statki”

Implementacja gry w statki w języku Python 3 z wykorzystaniem architektury klient-serwer obsługującej wielu graczy jednocześnie z użyciem szyfrowanych połączeń.

Damian Książek,
Kacper Derlatka

Protokół

Struktura wszystkich typów wiadomości jest podobna – nagłówki oddzielane są znakami ‘\r\n’, a koniec wiadomości sygnalizuje ‘\r\n\r\n’



Rysunek 1. Ogólny diagram sekwencyjny aplikacji

Klient

Po stronie klienta są wysyłane następujące typy wiadomości do serwera (numeracja odpowiada liczbom na diagramie – Rysunek 1):

1. *201\r\n\Hello\r\n\r\n*

3. *212\r\n\Message=My key\r\nKey-len=<długość klucza>\r\nKey=<klucz RSA>\r\n\r\n*

6. *400\r\n<obiekt>\r\n\r\n*

8. *405\r\nx:<int>\r\ny:<int>\r\n\r\n*

Opis:

1. Wiadomość powitalna
3. Przekazanie serwerowi klucza publicznego RSA
6. Przesłanie mapy w postaci zserializowanej tablicy dwuwymiarowej
1. Informacja o „strzale” wykonanynm przez gracza w postaci współrzędnych.

Serwer

Wiadomości przesyłane przez serwer (numeracja zgodna z diagramem na Rysunku 1):

2. *212\r\n\Message=Hello, give me your key.\r\nKey-len=<długość klucza>\r\nKey=<klucz RSA>\r\n\r\n*

4. *210\r\nMessage=All messages are now encrypted\r\n\r\n*

5. *101\r\nInfo:Build a map\r\n\r\n*

7. *401\r\nLet the game begin!\r\n\r\n*

9. *411\r\nYou: hit!\r\n\r\n* lub *412\r\nYou: Hit and sink!\r\n\r\n* lub *413\r\nYou: Miss!\r\n\r\n*

10. *421\r\nServer: I shoot <int>, <int>. I hit!\r\n\r\n* lub *422\r\nServer: I shoot <int>, <int>. Your battleship sinks!\r\n\r\n*

Opis:

2. Wiadomość odpowiedzi na powitanie klienta. Serwer tu przesyła też swój klucz publiczny RSA.
3. Wiadomość o pozytywnym przetworzyniu klucza publicznego i informacja o szyfrowaniu wiadomości.
4. Informacja
7. Informacja wysyłana po otrzymaniu przez serwer mapy od klienta
8. Informacja o „strzale” dokonanym przez klienta
9. Informacja o „strzale” przez serwer

Kody

Serwer i klient wysyłają różne komunikaty, a ich treść jest rozpoznawana między innymi przez kody wysyłane na samym początku każdej wiadomości.

Kody 1xx (kody informacyjne):

101 – informacja o tym by klient zbudował mapę

Kody 2xx (kody wiadomości powitania)

201 – wiadomość przywitania

210 – wiadomość o pozytywej wymianie kluczy

211 – klucz RSA serwera

212 – klucz RSA klienta

Kody 3xx (kody błędów):

350 – złe koordynaty podane przez klienta

370 – wiadomość bez kodu

371 – niepoprawny kod

396 – zły klucz RSA (niezaakceptowany przez funkcje z biblioteki Cypher)

397 – zły format klucza publicznego RSA

398 – niepoprawny format wiadomości przy odbiorze klucza

399 – niepoprawny format wiadomości przy powitaniu

Kody 4xx (kody rozgrywki):

400 – mapa klienta

401 – wiadomość o rozpoczęciu rozgrywki

405 – koordynaty strzału klienta

411 – trafienie przez klienta

412 –trafienie i zatopienie statku przez klienta

413 – nie trafienie przez klienta

421 – trafienie przez serwer

422 – trafienie i zatopienie statku przez serwer

423 – nie trafienie przez serwer

431 – zwycięstwo klienta

432 – zwycięstwo serwera

Klasy

Klient wykorzystuje klasy:

1. **EncryptorDecryptor** – Klasa odpowiedzialna za szyfrowanie oraz deszyfrowanie wiadomości przy użyciu kluczy RSA o długości 2048 bitów. Do poprawnego działania wykorzystuje bibliotekę pycryptodome.
2. **Map** – Klasa odpowiedzialna za proces tworzenia poprawnej mapy dla klienta oraz obsługę menu podczas tego tworzenia. Umożliwia również automatyczne wygenerowanie rozstawienia statków.
3. **Game_client** – Klasa obsługuje rozgrywkę po stronie klienta. Jest odpowiedzialna za poprawne parsowanie przyjmowanych wiadomości od serwera podczas rozgrywki oraz wysyłanie odpowiednich komunikatów do serwera. Dodatkowo wypisuje na bieżąco informacje dotyczące bieżącej gry.

Serwer wykorzystuje klasy:

1. **EncryptorDecryptor** – Klasa odpowiedzialna za szyfrowanie oraz deszyfrowanie wiadomości przy użyciu kluczy RSA o długości 2048 bitów. Do poprawnego działania wykorzystuje bibliotekę pycryptodome.
2. **Map** – Klasa odpowiedzialna za proces automatycznego stworzenia mapy dla serwera.
3. **Game_server** – Klasa, która ma za zadanie obsłużyć rozgrywkę pomiędzy serwerem, a graczem, decyduje o wysyłanych komunikatach w stronę gracza oraz sprawdza poprawność wiadomości otrzymywanych podczas rozgrywki od klienta, a także decyduje o ruchach podejmowanych przez stronę serwera podczas rozgrywki.

Wymagane biblioteki

1. Do poprawnego działania serwera wymagane są biblioteki:
    a. Asyncio
    b. Time
    c. Pickle
    d. Concurrent
    e. Pycryptodome

2. Do poprawnego działania klienta wymagane są biblioteki:

    a. Numpy
    b. Pickle
    c. Pycryptodome

Instrukcja uruchomienia

By móc rozpocząć rozgrywkę należy mieć zainstalowane wymagane biblioteki oraz uruchomić plik server.py. Następnie należy uruchomić plik klient.py i kolejno wykonywać polecenia wyświetlające się na konsoli, takie jak stworzenie swojej mapy oraz wybieranie koordynatów strzału.