Piotr Nowacki 155906

Sprawozdanie sk2

Saper – gra sieciowa

1. Opis projektu

- Saper to gra sieciowa oparta na architekturze klient-serwer, w której dwóch graczy rywalizuje w odkrywaniu pól na planszy. (Program wspiera wiele równoległych rozgrywek między parami graczy.)
- Opis gry:
  - Gra działa na planszy o rozmiarze 10x10.
  - Gracze na zmianę wykonują swoje ruchy, odkrywając pola.
  - Celem gry jest unikanie min i odkrycie wszystkich bezpiecznych pól.
  - Możliwe wyniki gry:
    - WIN – jeśli gracz wygra.
    - LOSE – jeśli gracz przegra, odkrywając minę.
    - DRAW – w przypadku remisu, jeżeli cała plansza zostanie odkryta i żaden z graczy nie odkrył miny.
- Użyte biblioteki: <locale>, <iostream>, <cstring>, <vector>, <cstdlib>, <ctime>, <sys/socket.h>, <netinet/in.h>, <unistd.h>, <thread>, <mutex>, <queue>, <condition_variable>
- Używałem również napisanej przeze mnie klasy obsługującej logikę gry (pliki Saper.hpp i Saper.cpp)

2. Opis komunikacji pomiędzy serwerem i klientem

W tym projekcie komunikacja pomiędzy serwerem a klientem opiera się na protokole TCP/IP, gdzie serwer nasłuchuje na określonym porcie, a klienci łączą się z nim, aby przeprowadzić rozgrywkę. Komunikacja jest zorganizowana w sposób synchroniczny, wykorzystując gniazda (sockets), które umożliwiają wymianę danych pomiędzy procesami na różnych maszynach w sieci.

Po stronie serwera, program nasłuchuje na porcie 8080, oczekując na połączenie od klientów. Kiedy klient łączy się z serwerem, serwer akceptuje połączenie i przypisuje odpowiednie gniazdo do klienta. Następnie serwer przechodzi do głównej pętli gry, w której dwóch graczy rywalizuje ze sobą. Każdy klient otrzymuje początkowy stan gry w formie mapy. Gra jest rozgrywana w turach, w których aktywny gracz wykonuje ruch (np. literę w, a, s, d), a serwer przetwarza ten ruch, aktualizując stan gry. Następnie serwer wysyła zaktualizowaną mapę do obu graczy.

Po każdym ruchu serwer sprawdza, czy gra nie została zakończona (np. jeden z graczy odkrył minę, co oznacza przegraną). Gdy gra kończy się (wygrana, przegrana lub remis), serwer wysyła odpowiedni komunikat do obu graczy (np. "WIN", "LOSE", "DRAW"), po czym zamyka połączenie.

Wymiana danych pomiędzy serwerem a klientami odbywa się za pomocą gniazd TCP, gdzie serwer wysyła informacje o stanie gry oraz oczekuje na ruchy od graczy. Gracze wysyłają swoje ruchy do serwera, który przetwarza je i aktualizuje stan gry. Po zakończeniu gry połączenie z klientami jest zamykane.

3. Podsumowanie

- Co sprawiło trudność
  - Jednym z głównych wyzwań w tym projekcie było odpowiednie zsynchronizowanie komunikacji pomiędzy serwerem a klientem. Musiałem zadbać o to, aby serwer poprawnie zarządzał dwoma klientami w jednej grze, obsługując ich na przemian, jednocześnie aktualizując stan gry i zapewniając płynność działania aplikacji. Implementacja wielowątkowości w celu obsługi wielu połączeń była kluczowa, ponieważ każdy klient musiał działać niezależnie od innych.
  - Dużym problemem napotkanym po drodze było to, że serwer wysyłał momentami dwie wiadomości do klienta pod rząd (np. informacje o tym że to jego tura oraz stan planszy obecny), przez co klient odczytywał od razu dwie wiadomości jako jedną całość i gra przestawała działać – wiadomość nie była ani komunikatem o turze ani planszą byłą połączeniem obu i nie nadawała się do niczego. Stworzyłem więc specjalne funkcję sendString i recvString, których używam za każdym razem gdy chce przesłać jakąś wiadomość. Naprawiają one mój problem w następujący sposób – przed wysłaniem wiadomości jako stringa wysyłają na ten socket int zawierający liczbę będąca długością stringa który zostanie za chwile wysłany. Adekwatnie przy odbiorze – najpierw odbieramy zmienną int (zawsze 4 bajty) a następnie długość stringa, którą odczytaliśmy z tego inta. W ten sposób nigdy nie odczytamy dwóch wiadomości połączonych w jedno bo zatrzymamy się przy czytaniu po dokładniej takiej liczbie bajtów jak powinniśmy.
  - Aby klikanie odbywało się bez potrzeby naciskania entera po każdym ruchu musiałem wyłączyć tryb kanoniczny oraz wyłączyć echoing – dzięki temu również użytkownik nie widzi wpisywanych przez siebie znaków co daję większe złudzenie płynności poruszania się po planszy a mniejsze wpisywania literek w wiersz poleceń.
  - Wyświetlanie planszy w konsoli aby było trochę bardziej czytelne zastosowałem kolorowanie pól (szczególnie pola na którym obecnie znajduje się gracz). Kolory są definiowane przy pomocy sekwencji ANSI (np. \x1B[31m dla czerwonego), a \033[0m resetuje stylowanie.