Krzysztof Garsztka 155954
Wojciech Kaczmarek 155994

Sprawozdanie sk2

Snake Multiplayer

1. Opis projektu (0.5 strony)

- Projekt to gra komputerowa w snake typu multiplayer. Obsługuje od 2 do 4 graczy.
- Gra ma 5 okienek:
- - okienko Start - strona tylko z przyciskiem start,
- - okienko Connect - strona w której należy wpisać adres ip serwera gry, port TCP serwera gry oraz nick,
- - okienko Waiting - strona z aktywnym stanem Lobby (jacy inni gracze się połączyli) oraz z przyciskiem Cancel - pozwala wyjść z Lobby,
- - okienko Game - okienko w którym odbywa się gra. Każdy gracz porusza się strzałkami, ruchy są widoczne na ekranach wszystkich osób połączonych. Węże nie rosną poprzez zjadanie jabłek, natomiast rosną co jakiś czas.
- - okienko End - okno podsumowujące grę.

Aby stworzyć GUI klienta wykorzystano biblioteki SDL2. Czcionka wykorzystana przez grę pochodzi z listy darmowej czcionek udostępnianej przez Google.
Dodatkowo, aby serwer miał możliwość obsługiwana wiele gier naraz dodano program “Launcher” który czeka na ip serwerów i ustalonym porcie. Klient wysyła żądanie do Launchera pod tym samym adresem, który próbuje stworzyć serwer pod podanym portem TCP. Żądanie jest typu UDP i zawiera port TCP, na którym ma powstać serwer.

.

2. Opis komunikacji pomiędzy serwerem i klientem (0.5 strony, może być schemat/rysunek)

Klient włącza program gry. Klika start, dostaje okienko w którym ma wpisać ip, port TCP serwera oraz nazwę użytkownika. Klient wysyła prośbę do Launchera o uruchomienie w tym miejscu serwera.

Jeśli użytkownik kliknie Connect a serwer pod danym adresem i portem TCP nie zostanie znaleziony zostanie wysłana prośba do Launchera o włączenie serwera. Nie ważne, czy Launcher osiągnie sukces czy nie wszystkie dane, które wpisał klient zostaną usunięte.

W przypadku, gdy połączenie osiągnęło sukces klient wyśle do serwera swój nick. W odpowiedzi dostanie nicki wszystkich innych użytkowników, którzy już się połączyli oraz na końcu swój nick.

W momencie przejścia na okienko “Waiting” użytkownik tworzy dodatkowy wątek, którego jedyną funkcją jest obserwowanie, czy kliknięty został przycisk “Cancel”. Jeśli został, klient przechodzi do ekranu startowego, a do serwera wysyłana jest wiadomość “@Disconnect”. Wiadomość

tą, wraz z nickiem klienta, który się odłączył, dostają wszyscy inny podłączeni klienci. Usuwają oni następnie ten nick ze swojej listy połączonych graczy.

W przypadku, gdy połączyła się odpowiednia ilość graczy (zdefiniowana w "#define MAX_PLAYERS) wszyscy klienci otrzymują po kolei "@Start" oraz strukturę "mój_numer.portUDPserwera". Struktura "mój_numer" pozwala na ustalenie własnej pozycji startowej (przy użyciu kodu Greya) oraz przekazywanie ruchów. Użytkownicy, kiedy chcą wykonać ruch, wysyłają do serwera komunikat "mój_numer.nowy_kierunek". Mogą tak robić przez 300 ms. Po 300 ms do wszystkich klientów (przy użyciu TCP, który jest nieblokujący w kliencie na tym etapie) wysyłana jest struktura "numer_gracza.jego_kierunek.numer_gracza … ". Klienci wykorzystują te dane, aby wiedzieć gdzie każdy gracz powinien się ruszyć. Jeśli dany gracz jest martwy, to miejsce "jego_kierunek" dla niego zastąpione jest liczbą 5 (ruchy są od 0 do 3 włącznie). Cykl się powtarza aż zostaje tylko jeden żywy gracz. Wszyscy gracze dostają wtedy komunikat "@End".

3. Podsumowanie (0.5-1 strona)

Program spełnia wszystkie wymogi ze strony prowadzącego w temacie "Wielosobowej gry czasu rzeczywistego". Potrafi obsłużyć od 2 do 4 graczy, obsługuje rozłączenia oraz wiele gier równolegle.

- Aby spełnić wymogi obsługi równoległych serwerów musieliśmy dorobić program "Launcher". Spowodowane jest to tym, że początkowo nie braliśmy pod uwagę możliwości obsługi równoległych gier. Dodatkowo obsługa funkcjonalności "@Disconnect" wymagała stworzenia dodatkowego wątku który pełni tylko tą funkcję. Wynika to z tego, że odczytywanie TCP jest blokujące (odczytywanie nieblokujące powodowało praktycznie losowe errory), przez co kliknięcie przycisku nie byłoby wykrywane gdyby nie osobny wątek.

Pewne trudności wynikały poprawnej synchronizacji klienta z serwerem. Pojawiały się błędy nielogiczne (jak na przykład write, który niby się wykonywał (zwracał odpowiednią liczbę bitów), ale nigdy nie przesyłał żadnych danych (mimo to, że każdy write po nim już działał poprawnie) co wymagało restrukturyzacji kodu dla tej funkcji.
Dodatkowo pojawiały się błędy związane z tym, że serwer wysyłał dane do bufora szybciej, niż klient je odczytywał. Wymagało to dodania małych opóźnień w kodzie.

Niewątpliwie jednak największe problemy pojawiły się z wyborem, kiedy użyć UDP a kiedy TCP oraz z tworzeniem samego GUI. Sama biblioteka do tworzenia GUI użytkownika zmieniana była 2 razy, z uwagi na niskiej jakości poradniki do ich użycia (czy w jednym przypadku wręcz płatne poradniki). Końcowo wybrana biblioteka (SDL2) okazała się być najlepsza spośród rozpatrywanych opcji (dla naszego użytku lepsza nawet niż SDL3).