Autorzy:

Kacper Kuras

Jakub Wieczorek

Wieloosobowa gra czasu rzeczywistego.

1. Opis projektu

Gra bulletHell.io jest wieloosobową grą czasu rzeczywistego z gatunku shooter-battleRoyal pozwalającą na jednoczesną grę do 4 graczy (aczkolwiek jest to ograniczenie miękkie, które można łatwo zwiększyć) na pojedynczej instancji (które mogą być uruchomione równolegle).

Gracz, aby wygrać - musi zostać ostatnim [graczem] na planszy – poprzez eliminację innych przeciwników. Przy rozpoczęciu rozgrywki i połączeniu się z serwerem gracz ma dostęp do edycji swojego awatara i przejścia do kolejki oczekujących na rozgrywkę.

Gdy zostanie zebrana odpowiednia ilość oczekujących graczy [3], serwer odlicza czas do rozpoczęcia rozgrywki [30s] i uruchamia grę. Każdy z graczy zostaje rozmieszony w losowym miejscu na ograniczonej planszy i jego zadaniem jest jednorazowe trafienie przeciwnika – wtedy też przeciwnik zostaje wyeliminowany i usunięty z rozgrywki. Gra jest kontynuowana do ostatniego gracza. Jeżeli w trakcie już istniejącej gry – do serwera podłączyłby się nowy gracz – otwierana jest nowa poczekalnia.

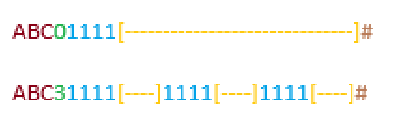
* użyta technologia

- BSD socket na systemie linux – użyte do komunikacji klient-serwer (linux-linux)

- raylib/raygui – silnik graficzny będący wrapperem opengl z dostępem do kreatora GUI i opcją manipulowania kolizji napisana w C

2. Opis komunikacji pomiędzy serwerem i klientem

Komunikacja klient-serwer odbywa się po protokole SCTP przez ręcznie przygotowane „wiadomości” (nazwanej również pakietem) przez funkcje serializującą w ciele programu. Każdy wiadomość podlega następującej strukturze:



ABC jest nagłówkiem wiadomości odpowiadającym strukturze wiadomości

1. wartość przekazywania liczby elementów dla struktury wektorowej – dla 0 przekazywany jest jeden element

1111 rozmiar najbliższej wysyłanej struktury

[----] wartość przekazywanej struktury

# znak końca wiadomości

Klient i serwer w swoim programie posiadają te same definicje struktur wykorzystanych do przesyłu. Każda ze struktur posiada 3-literowy nagłówek wiadomości pozwalający na przekazanie pakietu do odpowiedniej funkcji zarządzającej danymi. Jeżeli przekazywany jest wektor struktur. Bajt wektora ustawiany jest na liczbę przekazywanych struktur, co pozwala na odczytanie jednego pakietu ciągiem jako całość wiadomości. Rozmiar najbliższej struktury jest informacją - ile bajtów należy odczytać do wczytania całej wiadomości. Bajt końca będący ‘#’ używany jest do sprawdzenia poprawnego odczytania pakietu.

3. Podsumowanie

* Serializacja informacji

- Do transformacji struktury wysyłanej do formatu pakietu wykorzystuje się funkcje szablonowe, które za

argument przyjmują dowolną „wspieraną” klasę (czyli taką posiadającą wbudowany nagłówek jako jedną z funkcji szablonowych), oraz deskryptory, na który ma zostać wysłana wiadomość. Cała transformacja wykorzystuje mechanizm ‘castowania’ do zamiany ciągu bitów na inny – w przypadku BSD sockets na ciąg znaków char, która odbywa się w sposób nieblokujący. Odbiór odbywa się na tej samej zasadzie przy zapisie struktury jako referencji.

* Działanie serwera

- Przy prawidłowym połączeniu się z serwerem i wybraniu awatara - gracz trafia do tzw. poczekalni, gdzie

serwer oczekuje na graczy do rozpoczęcia gry. W poczekalni co około 0.5 s rozsyłana jest informacja do wszystkich graczy w poczekalni o znajdujących się w niej graczach (nazwa oraz kolor) a także o pozostałym czasie. Gdy liczba graczy w poczekalni jest równa 3 – rozpoczyna się okres przygotowania do gry, gdzie po okresie 30 sekund zostanie uruchomiona rozgrywka. Gracze którzy połączą się z serwerem w trakcie okresu przygotowania - również zostają przeniesieni do rozgrywki. Dla potencjalnych następnych graczy tworzona jest nowa poczekalnia i proces się powtarza.

- Gra w sposób ciągły przesyła informację o pozycji wszystkich graczy, którzy to pokazują się na ekranie. Po trafieniu któregoś z graczy przez pocisk – po stronie serwera wysyłany jest sygnał „terminacji”

trafionego gracza, który to „przegrywa” i zostaje usunięty z gry. Gra jest kontynuowana do ostatniego gracza.

- Jeżeli gracz utraci połączenie z serwerem zostaje uznany za

wyeliminowanego i usuniętego z rozgrywki. Jako że długość jednej rozgrywki nie powinna być

dłuższa, niż minuta – rozwiązanie jest uczciwe dla pozostałych graczy (zakładając, że gracz z wolnym łączem – tak naprawdę byłby oszustem chowającym przed eliminacją)

* Co sprawiło trudność
  + Transmisja danych
    - Początkowo nasze gra miała być wieloplatformowa – bazująca na komunikacji windows/linux(klient) – linux(serwer) gdzie za pomocą dyrektyw kompilatora miały być dobierane odpowiednie biblioteki. Jednak problemem okazały się różne kompilatory msvc,clang,g++ które każde zapisywały informację w różnym formacie - przez co serializacja danych nie była możliwa - pomimo zastosowania typów wieloplatformowych. Rozwiązaniem problemu było porzucenie wirtualnych interfejsów dziedziczących na rzecz abstrakcyjnych szablonów – które ignorowały wszelkie potencjalne błędy każdego kompilatorów. Jednakże porzuciliśmy wieloplatofrmowość z powodu błędów kompilacji.
  + Synchronizacja wątków oraz przesyłu informacji
    - Do zarządzania każdej z gier chcieliśmy wykorzystać wątki do każdej z części rozgrywki (co zostało prawidłowo wykonane, jednak sprawiło problem). Każdy oddzielny wątek przypada na: poczekalnie, rozgrywkę, obsługę gniazda dla każdego z graczy. Ponadto początkowo klient wraz z serwerem musiały pracować w identycznym tempie inaczej gracze byli coraz bardziej „rozsynchronizowani” przez pobieranie starszych pakietów. W ostatecznej wersji klient odbiera w jednej klatce wszystkie pakiety dotyczące pozycji znajdujące się obecnie w buforze, co sprawia iż może on działać wolniej od serwera, jednak prędkości obydwu programów zostały ograniczone do 60 kl./s/