Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Piotr Truszkowski
Nr albumu: 209223

Sieciowa gra „Flatline”

Praca licencjacka
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
mgr Krzysztofa Ciebiery
Instytut Informatyki

Czerwiec 2006

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

Praca omawia projekt programistyczny „Flatline” będący sieciową grą akcji. W zamyśle nasz zespół chciał stworzyć dynamiczną grę komputerową dla wielu graczy (co najmniej 4) z możliwością gry zarówno przez sieć lokalną jak i internet. Z punktu widzenia wszystkich członków zespołu, projekt zakończył się pełnym sukcesem.

Słowa kluczowe

rozrywka, zabawa, współzawodnictwo, gra komputerowa, gra sieciowa, Flatline

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

K. Computing Milieux
K.8. Personal Computing
K.8.0. General

Tytuł pracy w języku angielskim

Net game „Flatline”

Spis treści

Wprowadzenie

W skład zespołu pracującego nad grą sieciową Flatline wchodziły cztery osoby (w kolejności alfabetycznej): Marcin Barczyński, Wojciech Kazana, Piotr Truszkowski, Robert Źrałek. Program ten miał być i ostatecznie został napisany w ramach zajęć z Zespołowego Projektu Programistycznego w roku akademickim 2005/2006.

Zespół dość szczegółowo podzielił się pracą nad poszczególnymi częściami (modułami) gry. I tak oto:

- Marcin Barczyński był odpowiedzialny za implementację algorytmów umożliwiających zarządzanie rozgrywką i badanie kolizji pomiędzy obiektami na planszy
- Wojciech Kazana był odpowiedzialny za komunikację sieciową pomiędzy poszczególnymi instancjami gry i serwerem
- Piotr Truszkowski odpowiadał za trójwymiarowe modele obiektów, zarządzanie rozgrywką oraz tworzenie części plansz
- Robert Źrałek (szef zespołu) odpowiadał za silnik graficzny, tworzenie części plansz oraz synchronizację prac poszczególnych członków zespołu.

> „I gdy przemówiła do nas, Jej głos zdawał się rozbrzmiewać wewnątrz umysłów naszych, miast przez uszy przechodzić. A postać jej skąpana była w czerwonym świetle zachodzącego słońca. Tak oto bogini wszelkiego zła po raz pierwszy ukazała nam Swe oblicze(...)"
>
> *treść starożytnej inskrypcji uważana za pierwszą wzmiankę o Anabey Avruck - starożytnej bogini śmierci, która nie jedną linię życia zmieniła w prostą rzeczywistą...*

Rozdział 1

Gra

Flatline od samego początku miała być (i jest) grą akcji. Z dużym naciskiem na grą - program ten jest przeznaczony do rozrywki i ewentualne walory edukacyjne w założeniach znajdowały się na dalszym planie.

„(...)
- Ucz się ucz uczniu mój drogi!
- Ależ kiedy ja bym na wojnę wolał! Pozabijać kogo tam się uda...
- Ot prosił, prosił i uprosił (...)”

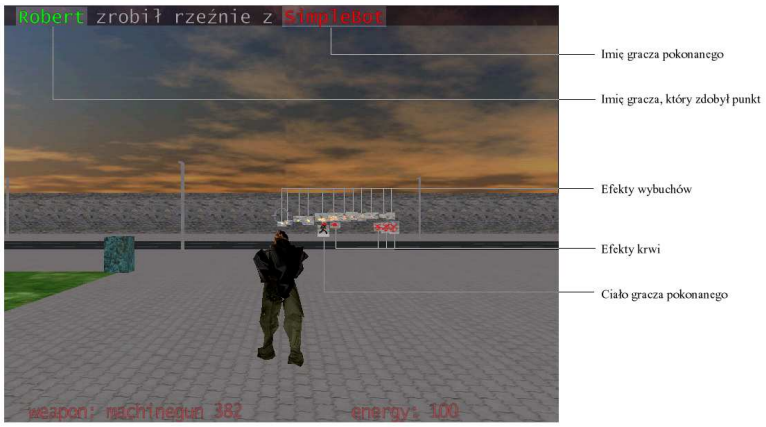
„Historia tego, którego nie było” Piks Wokzsurt

1.1. O co tu chodzi?

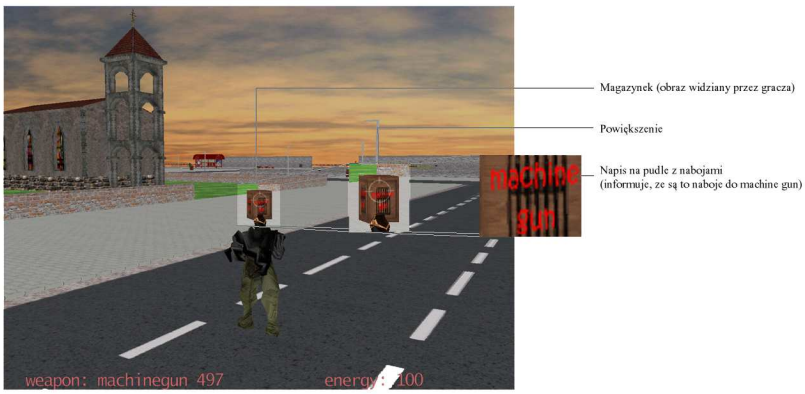
W grze Flatline sterujemy postacią żołnierza. Ten z różnych przyczyn znalazł się w miejscu dla niego nieprzyjaznym, w którym musi walczyć o przetrwanie.

Każdy ruchomy obiekt jest potencjalnym zagrożeniem. A nasza postać jest tu obca i nie ma jeszcze wiedzy, jak postępować w lokalnym otoczeniu. Dlatego najlepiej wybrać rozwiązanie - choć moralnie na pewno bardzo trudne i wieloznaczne, w praktyce pewne i skuteczne -

wyeliminować zagrożenie wszelkimi dostępnymi środkami. A wszystkie te, pech chciał, sprowadzają się do jednego: ZABIĆ ICH WSZYSTKICH.

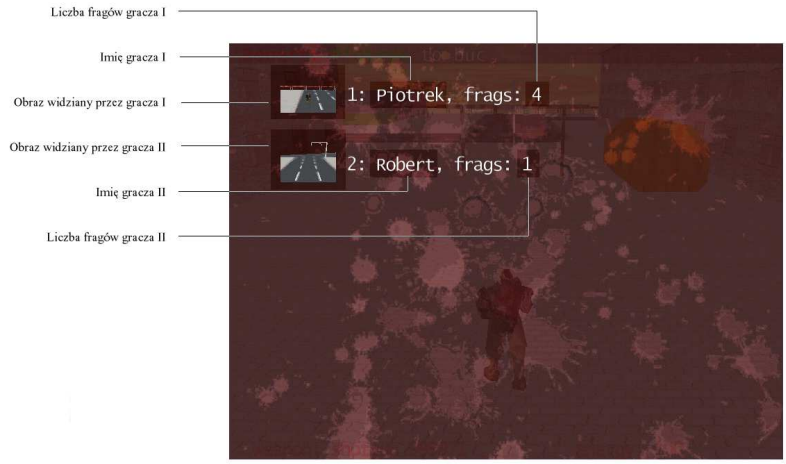


Za każdego zgładzonego przeciwnika dostajemy punkt (tzw. frag). Za zabicie samych siebie niestety go tracimy, więc trzeba uważać... Na planszy dostępne są liczne dodatkowe bronie, które mogą nam tylko pomóc. Z ewentualnym brakiem amunicji skutecznie możemy walczyć łapiąc porozrzucane tu i ówdzie skrzyneczki z pociskami.



Apteczki wirujące w wielu różnych lokacjach pozwalają przedłużyć żywot gracza o te kilka chwil niezbędnych do zadania jeszcze jednego ostatecznego ciosu! Ale zapewne i na nas kiedyś przyjdzie czas...

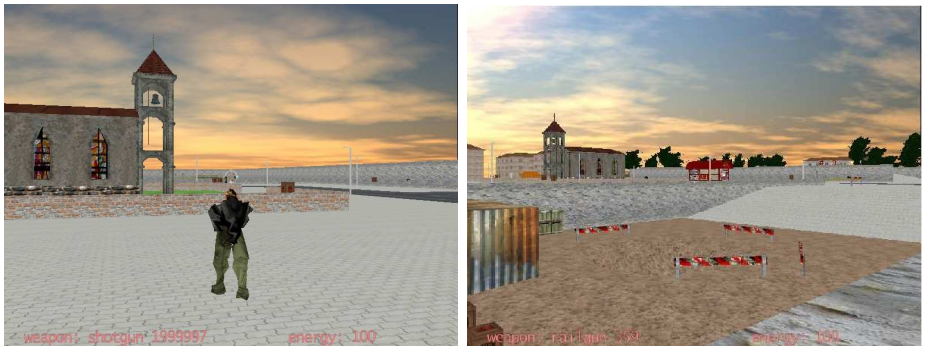
Wtedy na ekranie pojawia się informacja o aktualnym wyniku każdego z graczy i na kilkanaście sekund musimy opuścić plac boju.



Potem tylko „strzał” i wracamy. Wracamy zwarci i gotowi, by zabijać!! Ale może po kolei:

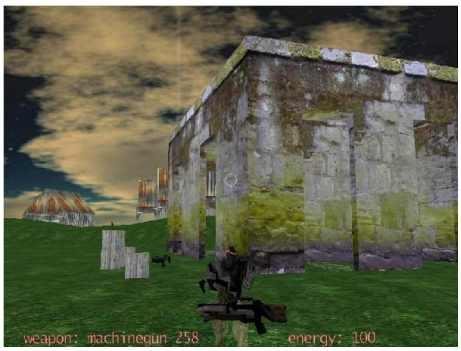
1.2. Świat gry

Gdy już z menu wybierzemy odpowiednie dla siebie opcje, gdy zdecydujemy się grać na odpowiedniej mapie bądź też podłączyć do odpowiedniego serwera ukaże się nam świat gry.



Gracz ma na początku gry jedną broń - shotgun, wraz z amunicją do niej. Zaleca się jednak szukanie innych broni, są one przede wszystkim dużo skuteczniejsze w walce. W ich przypadku jednak należy nie zapominać o amunicji. Wprawdzie zbierając broń dostajemy ją z załadowanym magazynkiem odpowiednich naboi, lecz w odróżnieniu od shotguna szybko się kończą. Na mapach znajdują się miejsca gdzie można znaleźć amunicję, każdy rodzaj broni ma odpowiadający jej magazynek. Gracz nie może zebrać dwóch takich samych broni. Nie tyczy się to magazynków, tutaj gracz może zbierać je do woli, niezależnie od ich rodzaju.
Jak rozpoznać na mapie miejsce gdzie leży broń czy magazynek? Otóż bronie czekające na zebranie wirując wokół własnej osi. Magazynki znajdują się w drewnianych skrzynkach(również wirujących). Na każdej skrzynce jest napisane do jakiej broni są. Bronie można rozróżniać po ich wyglądzie.

Rodzaj broni decyduje o tym z jaką szybkością są wystrzeliwane pociski, z jaką częstością oraz jak duże obrażenia wywołują u przeciwników. Są w grze zatem bronie o dużej sile rażenia (blaster, rocket) jak i szybkostrzelne (supershotgun, railgun). Złotym środkiem dla broni wydaje się być machinegun ale to jak zwykle bywa w grach wszystko zależy od upodobań

gracza. Podobnie z magazynkami, tak jak wcześniej była mowa każda broń ma swoją amunicję tak samo więc w różnych typach magazynków są różne ilości pocisków(magazynki tego samego typu mają tę samą liczbę naboi). Gracz może w czasie gry przełączać bronie. Gdy zabraknie pocisków w aktywnej broni, automatycznie przełączana jest następna broń(o ile każda z posiadanych broni nie wyczerpała już magazynków).

Z bronią i pełnym magazynkiem możemy ruszyć do boju.

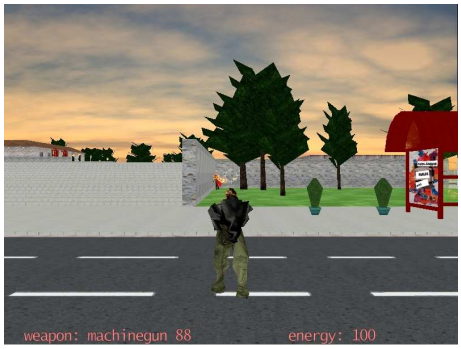
Bezpośrednie starcie z przeciwnikiem może się przyczynić do znacznej utraty energii. Ku regeneracji sił możemy pośpieszyć do najbliższej apteczki. Każda apteczka dodaje nam 33 punkty energii. Jeśli mamy 100 punktów energii zebranie apteczki nic nie zmieni. Apteczki po pewnym czasie się odradzają i można je ponownie zbierać.

Można atakować z ukrycia, urzywając zoomu.

Walczyć oko w oko, ostrzeliwać przeciwnika z góry...

W końcu, my albo przeciwnik, giniemy. Nalicza się wtedy frag temu kto zabił. Zastrzelony gracz odradza się w nowym, losowym miejscu mapy. Traci on wszystkie bronie poza shotgunem (stracone bronie odrodzą się po pewnym czasie gdzieś na mapie). Magazynki jednak gracz zachowuje. Przy śmierci gracz widzi splashscreen z najlepszymi wynikami i swoją pozycją. Widzi także w małych okienkach zbliżenia grających postaci. Po wciśnięciu „FIRE" wraca do gry.

Po odrodzeniu gracza wracamy do gry. Znów szukamy broni, amunicji a co najważniejsze ofiar. Pozostaje już tylko rozkoszowania się grą...



1.3. Menu gry

Po uruchomieniu gry pojawi się następujące pytanie:

W większości przypadków zaleca się tryb pełnoekranowy. Program osiąga w nim lepszą wydajność a przyjemność z grania jest nieporównywalnie większa.

Po wybraniu trybu pracy wyświetli się główne menu gry.

Mamy do wyboru pięć opcji:

- Resume game - pozwala na wznawianie rozpoczętej wcześniej gry a w przypadku gdy takiej nie było ma takie samo znaczenie jak następna z opcji
- Single player - uruchamia grę w trybie pojedynczego gracza; tryb ten pełni funkcję treningową: pozwala na zaznajomienie się z mapą oraz walkę z umieszczonymi na niej botami
- Multi player - uruchamia grę w trybie wielu graczy; wymaga uruchomionego wcześniej serwera gry i dobrze skonfigurowanych ustawień sieciowych; jest to główny tryb pracy programu
- Setup - pozwala na ustawienie wielu różnych opcji takich jak np. wybór imienia oraz postaci, konfiguracja klawiszy oraz ustawień sieciowych itp.
- Exit - wychodzi z programu

Wybór opcji odbywa się poprzez najechanie na nią za pomocą strzałek góra/dół oraz naciśnięcie klawisza Enter. W przypadku wybrania jednej z pierwszych trzech opcji ukaże się informacja o ładowaniu mapy (w naszym przypadku wybrana została gra w trybie pojedynczego gracza).

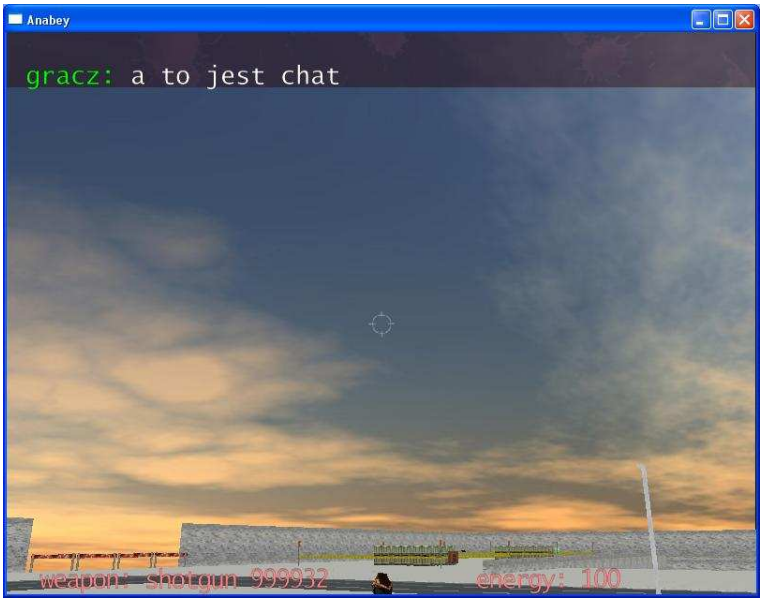
Po załadowaniu mapy ukaże nam się świat gry.

Na dole ekranu wyświetla się aktualnie posiadana broń wraz z ilością amunicji oraz ilość energii jaką posiadamy. Gdy energia spada do zera nasz bohater umiera i pojawia się okno z listą graczy oraz ilością punktów jakie zdobyli.

Po chwili okienko znika i naciśnięcie klawisza Fire powoduje odrodzenie się w nowym miejscu. Dodatkowo w dowolnym momencie gry możemy nacisnąć klawisz Escape i wrócimy do menu głównego gry z którego aby wrócić do gry należy wybrać opcję Resume Game.

Naciskając Fire odrodzimy się w nowym miejscu na planszy i będziemy znowu mogli uczestniczyć w rozgrywce. Podczas gry dostępny jest również czat pozwalający graczom komunikować się między sobą. Czat uruchomiamy klawiszem ˜(tylda) i na górze ekranu pojawia się zaciemniony obszar w którym wyświetla się pisana wiadomość.

Wysłanie wiadomości następuje po naciśnięciu klawisza Enter. Wysłanie powoduje ukazanie się wiadomości wszystkim graczom na górze ekranu.



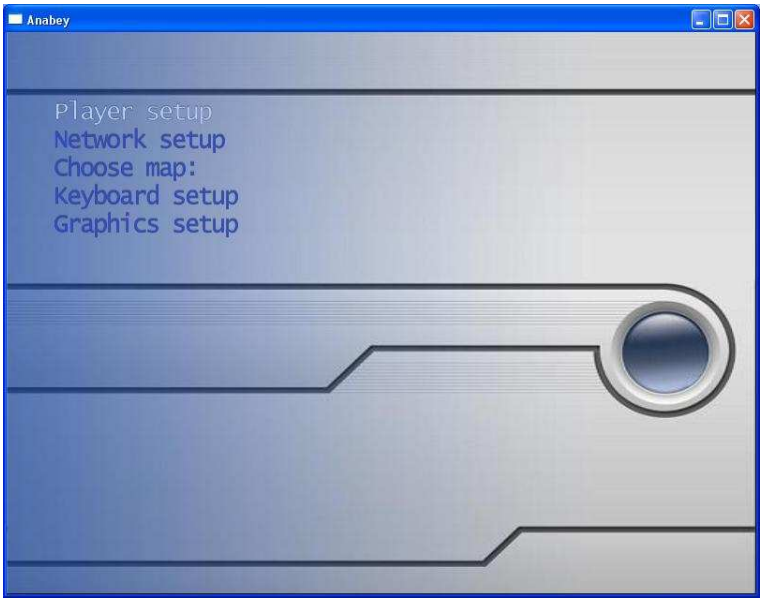
Ukryte opcje chata:

Wprowadzając niedrukowalną kombinację klawiszy ˜R, można zmienić kolor pisania w trybie chata na kolor czerwony. Analogicznie można zmieniać kolor pisania na zielony, niebieski, żółty, biały. Wystarczy wprowadzić odpowiednią kombinację klawiszy "˜" + pierwsza litera koloru w języku angielskim.

1.4. Menu setup

„(...)
- Porządek wszak być musi!!
- Co zatem czynić mamy, gdy bałagan, brud i syf straszny ze wszech stron nas opasali?
- Za mną idźcie, a ja już wam drogę wskażę!! (...)"

„Rządź innymi, bo przyjemne to jest ponad miarę" Trebor Z̃.



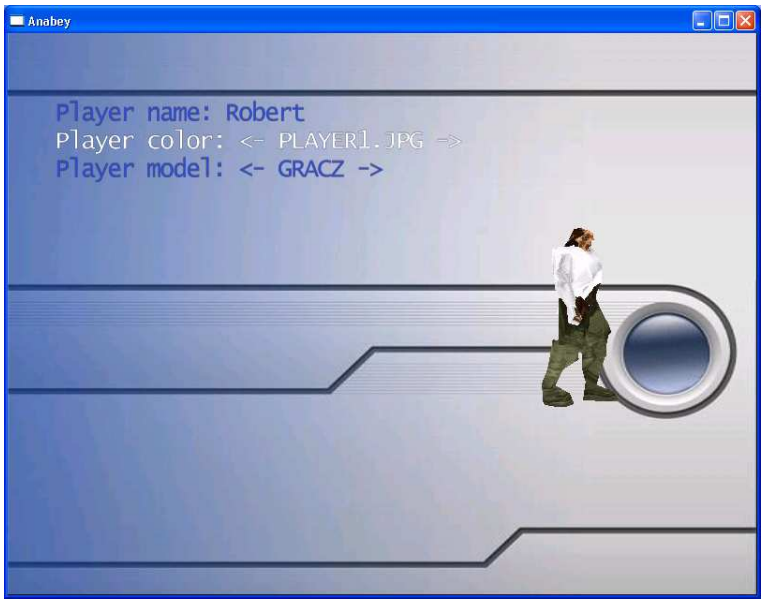
Mamy do wyboru pięć rodzajów opcji: Player Setup, Network setup, Choose map, Keyboard setup oraz Graphics Setup.

1.4.1. Player setup

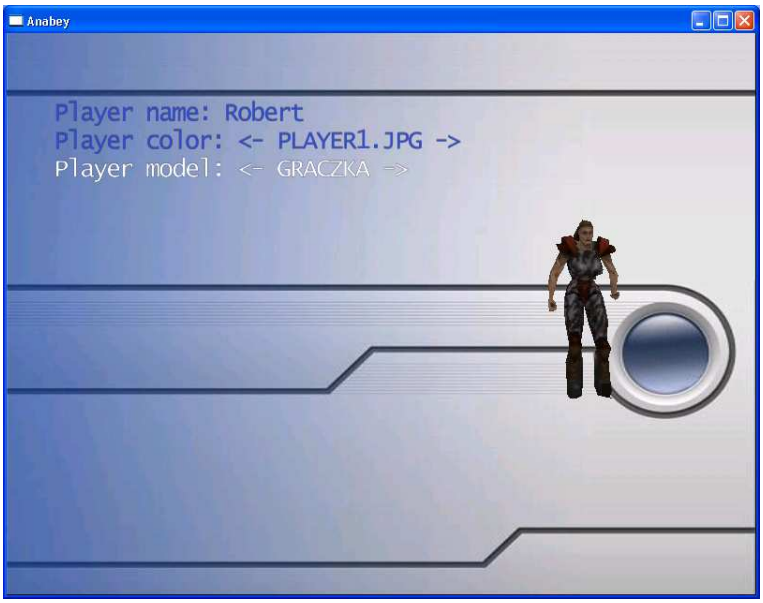
W tym menu możemy konfigurować imię gracza oraz jego skórkę i kolor.

Aby zmienić imię należy wybrać opcję Player name i wpisać żądane imię. Zatwierdzenie odbywa się po naciśnięciu klawisza Enter.

Aby zmienić skórkę lub kolor gracza należy wybrać opcję Player color i za pomocą strzałek dokonać wyboru.

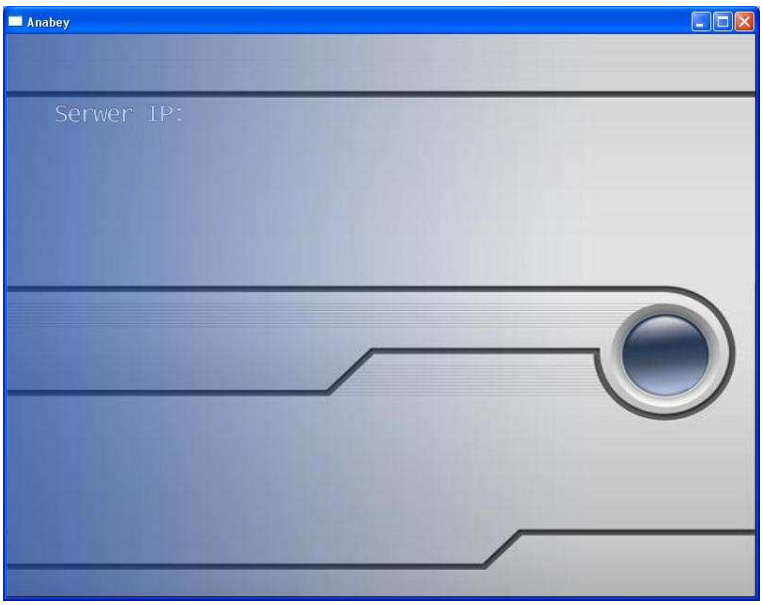


Aby zmienić model gracza należy wybrać opcję Player model i za pomocą strzałek dokonać wyboru.

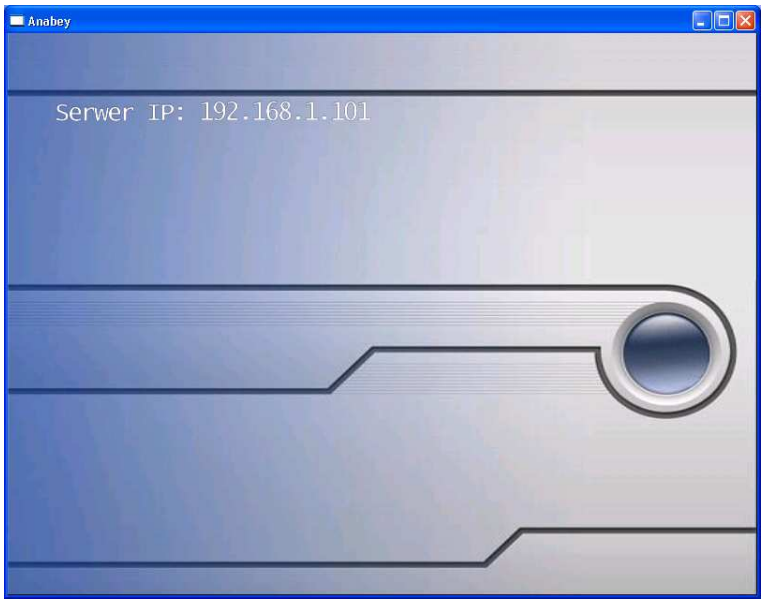


1.4.2. Network setup

Tutaj należy dokonać ustawień sieciowych niezbędnych do gry w trybie wielu graczy.

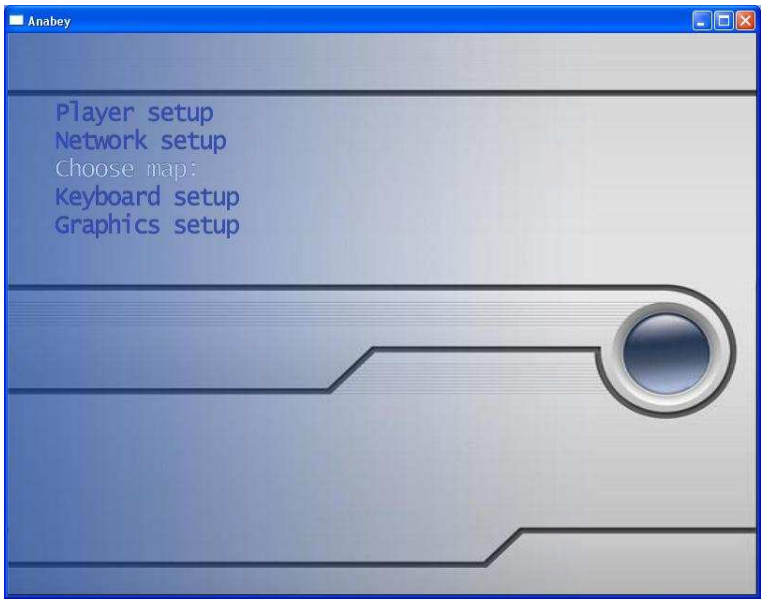


Jedyną opcją do ustawienia jest adres IP komputera, na którym uruchomiony został serwer gry. Adres ten należy wpisać w Serwer IP.

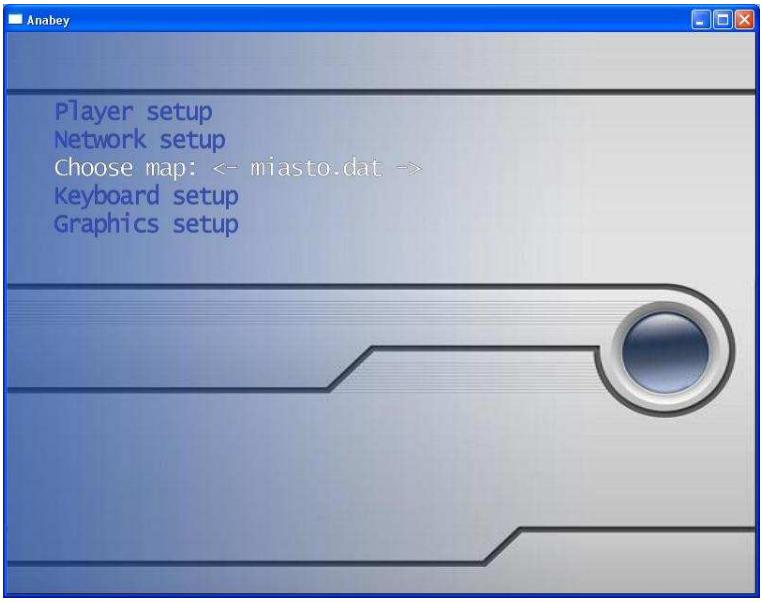


1.4.3. Choose map

Ta opcja pozwala na wybór mapy, na jakiej odbywać się będą potyczki. W przypadku gry wieloosobowej wybór nie ma znaczenia, gdyż dokonuje go serwer.



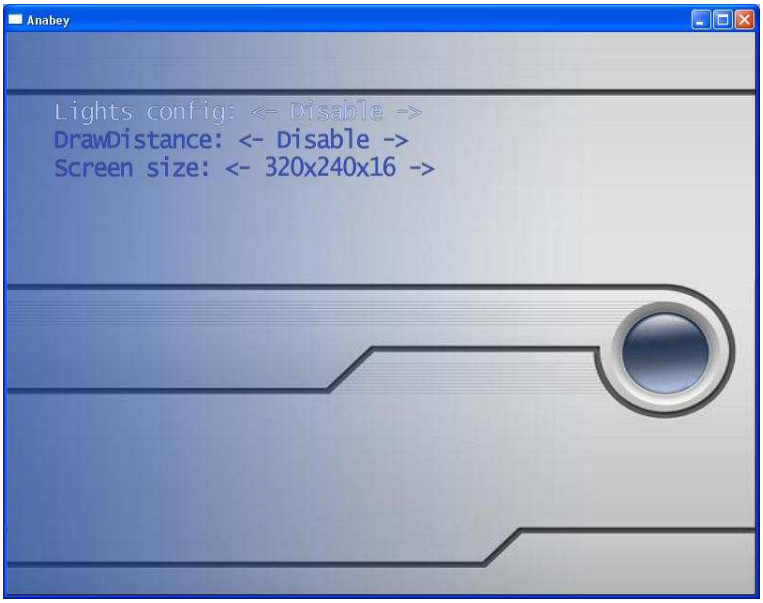
Domyślnie nie jest wybrana żadna mapa co oznacza, że gra załaduje mapę domyślną, czyli „miasto”.



Można jednak za pomocą strzałek wybrać dowolną inną z dostępnych map.

1.4.4. Graphics setup

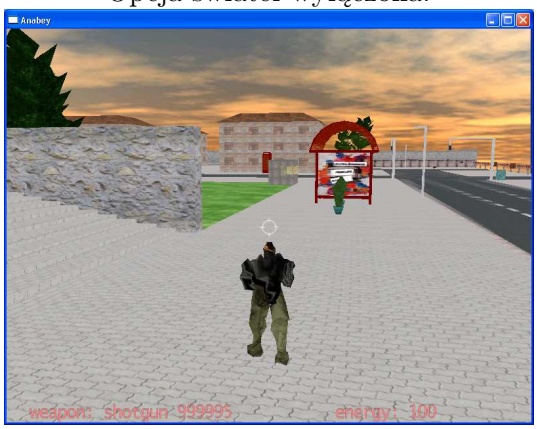
W tym menu możemy konfigurować ustawienia grafiki.



Opcja "Lights config" służy do włączania i wyłączania świateł w grze.

Opcja świateł włączona:

Opcja świateł wyłączona:



Niestety większość map nie została wzbogacona w efekty w postaci świateł. Główne ustawienia z jakimi ładowane są mapy to ustawienie cieniowania i jednego światła lub dwóch znajdujących się wysoko i oświetlających znaczną część mapy. Konfiguracja map z większą ilością świateł nie jest zalecana przez autorów gry, gdyż spowalnia proces wyświetlania grafiki a co za tym idzie jakość i płynność rozgrywki.

Opcja "DrawDistance" służy do włączania i wyłączania trybu rysowania obiektów znajdujących się tylko blisko gracza.

Opcja "Draw Distance" włączona:

Opcja "Draw Distance" wyłączona:

Opcja "Draw Distance" znacząco poprawia szybkość generowania grafiki w przypadku dużych i bardzo dużych map. W przypadku małych i średnich map przyrost wydajności procesu renderowania jest mniejszy, lecz także zauważalny. Opcja ta przeznaczona jest raczej dla graczy posiadających słabsze komputery, lub tych którzy chcą się cieszyć płynną animacją w bardzo dużych rozdzielczościach, przy dużej głębi kolorów z dużą liczbą detali.

Opcja "Screen size" służy do zmiany rozdzielczości ekranu i głębi kolorów.

Dostępne są następujące tryby wyświetlania grafiki:

- 320x240x16
- 640x480x16
- 800x600x16
- 1024x768x16
- 1280x768x16
- 1280x1024x16
- 320x240x24
- 640x480x24
- 800x600x24
- 1024x768x24
- 1280x768x24
- 1280x1024x24
- 320x240x32
- 640x480x32
- 800x600x32

- 1024x768x32
- 1280x768x32
- 1280x1024x32

Pierwsza liczba przy trybie wyświetlania grafiki oznacza szerokość ekranu przy jakim będzie wyświetlana obraz podczas gry. Druga liczba oznacza wysokość ekranu, a trzecia głębię kolorów.

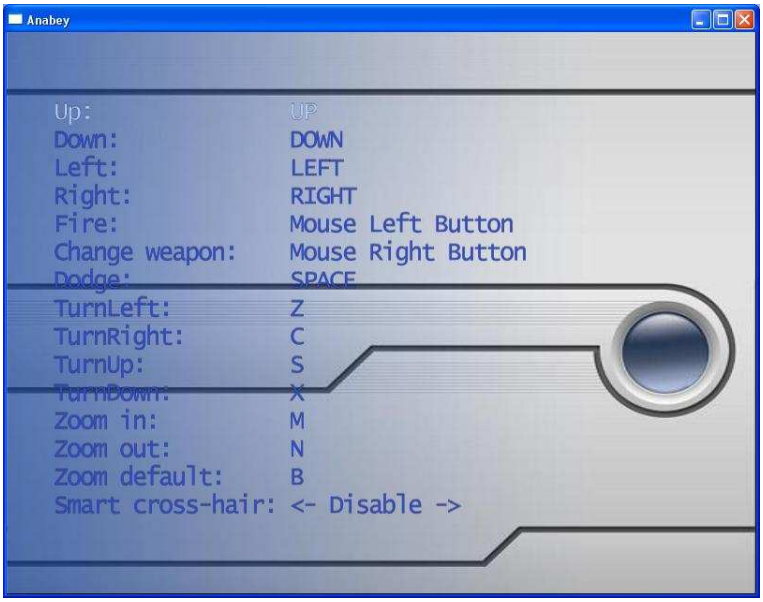
Powyższa lista jest listą standardowych rozdzielczości wraz z odpowiadającymi im standardowymi głębiami kolorów. Możliwe jest uruchomienie gry w dowolnej rozdzielczości i głębi kolorów akceptowanej przez kartę graficzną i monitor. Jest to możliwe poprzez edycję pliku konfiguracyjnego. Plik konfiguracyjny służy do edycji większej liczby opcji niż te, które znajdują się w menu. Szczegóły w dziale odpowiadającym za konfiguracje gry w pliku.

1.4.5. Keyboard setup

Używając opcji konfiguracji klawiatury z menu gry (z podmenu Setup) możemy dostosować funkcje poszczególnych klawiszy według osobistych preferencji. Opcje z menu Keyboard setup umożliwiają konfiguracje większości klawiszy potrzebnych do gry. Wszystkie klawisze (poza "Esc", który służy do wyjścia z gry i "˜", który otwiera linię chata) mogą zostać dowolnie przeprogramowane.

Po lewej stronie menu konfiguracji klawiatury znajduje się zdarzenie jakiemu odpowiadać będzie naciśnięcie klawisza znajdującego się po prawej stronie tabeli na tej samej wysokości.

Standardowo klawisz "Up" znajdujący się grupie klawiszy sterujących przypisany jest do zdarzenia któremu odpowiada poruszenie zawodnikiem do przodu. Analogicznie klawisze "Down", "Left", Right" służą do poruszania graczem do tyłu, w lewo i w prawo. Lewy klawisz myszki odpowiedzialny jest za strzelanie przez gracza z broni. Prawy klawisz myszki odpowiada za zmianę broni na kolejny model jeśli gracz posiada więcej niż jedną broń. Wybór broni na kolejne modele odbywa się cyklicznie.



Klawisz spacji służy do wykonania uniku. Gracz w wyniku tego zdarzenia wykona efektowny przewrót w przód.

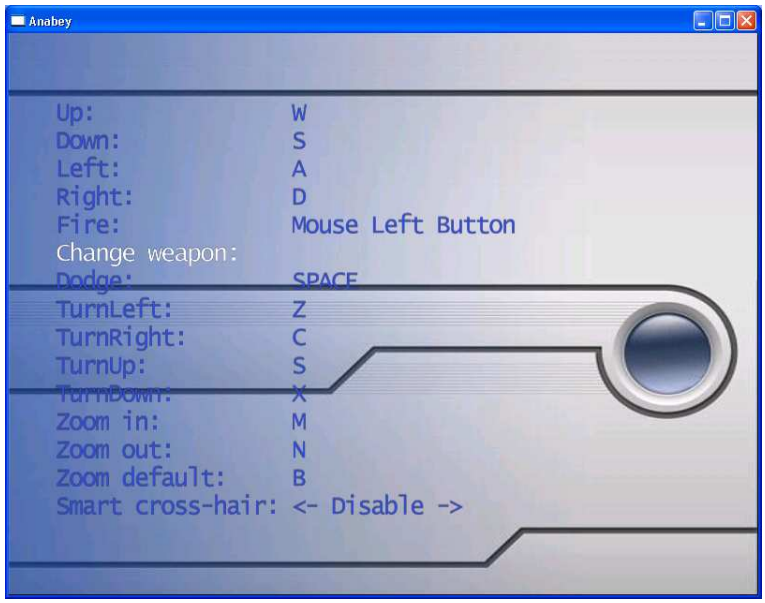
Łącząc ze sobą klawisze w lewo i klawisz odpowiedzialny z unik można wykonać przewrót w bok. Analogicznie podczas gry można łączyć ze sobą inne klawisze wykonując przewrót pod skosem w prawo i do przodu. Wszystko zależy od sytuacji i od decyzji gracza.

Klawisze odpowiedzialne za zdarzenia "TurnLeft", "TurnRight", "TurnUp", "TurnDown" służą do obrotu graczem odpowiednio w lewo, prawo, w dół i w górę. Opcja ta przeznaczona jest głównie dla graczy, którzy nie przepadają za sterowaniem postacią za pomocą myszki. Głównie są to osoby, które będą grały za pomocą różnych manipulatorów do gier typu Gamepady, Joysticki.

Opcje "Zoom in", "Zoom out", "Zoom default", służą do zmiany ustawienia teleobiektywu w broniach typu karabin snajperski. Opcja "Zoom default" służy do przywrócenia widoku z normalnej perspektywy.

Zmiana przypisania klawiszy do poszczególnych funkcji jest możliwa z poziomu menu konfiguracji klawiatury. Aby zmienić klawisz odpowiedzialny za daną funkcję należy ustawić kursor nad wybraną funkcją i nacisnąć "Enter". Następnie należy wprowadzić klawisz,

którego chcemy używać przy danej funkcji. Przykład obok pokazuje zmienione klawisze odpowiedzialne za poruszaniem postacią oraz ustawiony kursor na opcji zmiany broni wraz z trybem zmiany przycisku odpowiedzialnego za to zdarzenie.

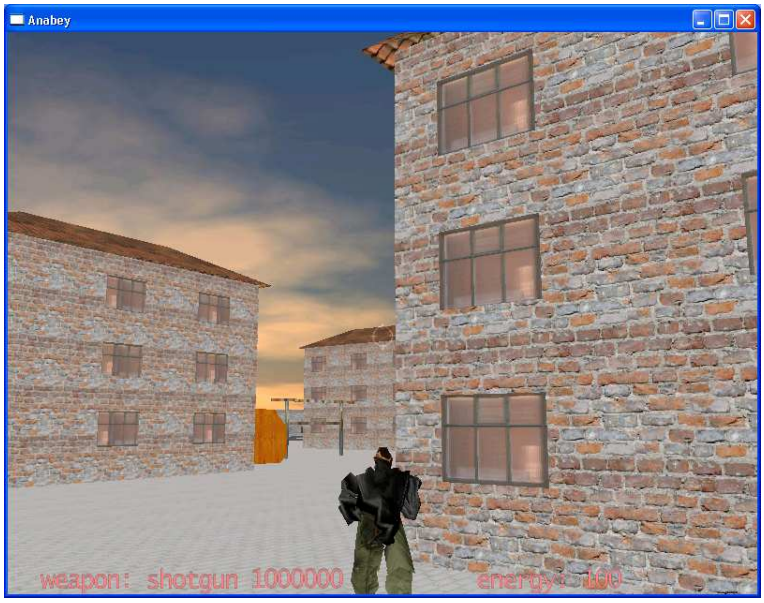


Ostatnia opcja "Smart cross-hair" przeznaczona jest raczej dla początkujących graczy. Jest to w większym stopniu opcja konfiguracji sterowania niż opcja ściśle klawiatury. Inteligentny celownik służy jako rozwinięcie procesu sterowania celownikiem podczas wymiany ognia. Inteligentny celownik naprowadza się na cel i dostosowuje swoje położenie zgodnie z tym, w co dokładnie celuje gracz. Objawia się to tym, że celownik minimalnie wędruje po ekranie i dostosowuje swoje położenie przewidując zachowanie gracza.

Jeśli gracz trzyma myszkę w takim położeniu, że celownik znajduje się niemal poziomo, to celownik ustawia się tak, że gracz będzie strzelać tak, że pociski będą lecieć dokładnie poziomo.

Jeśli gracz biegnie przy ścianie lub przy obiektach o różnych kształtach to celownik ustawia się tak, by pociski uderzyły idealnie w miejsce na które zostały naprowadzone. W tym przypadku ścianę lub beczki.

Opcja inteligentnego celownika, jak już wspomniałem, przeznaczona jest raczej dla początkujących graczy, gdyż z jednej strony poprawia celność przy mniej wymagających przeciwnikach. Lecz gracze z większym doświadczeniem przewidują ruchy przeciwników i strzelają z wyprzedzeniem w miejsce, w którym zawodnik będzie się znajdował za ułamek sekundy. Dla nich inteligentny celownik jest utrudnieniem, gdyż uniemożliwia im celowanie minimalnie obok gracza.



1.5. Konfiguracja z pliku .cfg

W podkatalogu data/conf/ znajdują się pliki konfiguracyjne gry. Plik konfiguracyjny gry jest rozszerzeniem do opcji dostępnych z poziomu menu gry. Konfiguracja opcji z pliku konfigu-

racyjnego, podobnie jak w przypadku konfiguracji z poziomu menu, nie jest potrzebna do prawidłowego funkcjonowania gry.

Plik .cfg zwany później plikiem konfiguracyjnym jest zbiorem argumentów z jakimi gra zostanie uruchomiona i które zostaną zaaplikowane przed właściwym uruchomieniem programu.

Przykład pliku konfiguracyjnego przedstawia poniższy listing.

```
SECTION NETWORK
192.168.1.100
SECTION GRAPHIC
DRAW_DISTANCE 0
LIGHTS 0
SCREEN_WIDTH 800
SCREEN_HEIGHT 600
SCREEN_DEPTH 16
MOUSE_SUSCEPTIBILITY 200.0
SECTION CHAT
~R$LOOSER: ~G$MASTER~W to buc
~G$MASTER~W zatłukł był ~R$LOOSER
~G$MASTER~W zrobił rzeźnię z ~R$LOOSER
~WMy tu gadu-gadu a ~G$MASTER~W zakatrupił wprost potwornie ~R$LOOSER
~G$MASTER: ~W Oj zabiłem ja ~R$LOOSER ~W w tej chwili. Bądźcie dla mnie mili...
SECTION PLAYER
NAME Robert
SKIN player0
MODEL gracz
```

1.5.1. SECTION NETWORK:

W sekcji network istnieje możliwość ustawienia adresu IP serwera.

Podobnie jak w przypadku konfiguracji serwera z poziomu menu, podczas ładowania gry gra będzie starała się połączyć z serwerem zarówno w celu pobrania informacji o mapie, na której będzie się odbywać rozgrywka, jak i wszystkich potrzebnych informacji do prowadzenia gry z sieciowymi przeciwnikami.

1.5.2. SECTION GRAPHIC:

W sekcji graphic istnieje możliwość ustawienia większej liczby opcji niż jest to w przypadku menu.

Ustawianie zmiennej SCREEN_WIDTH na wartość VAL powoduje uruchomienie gry przy zadanej szerokości ekranu.

Podobnie opcje SCREEN_HEIGHT i SCREEN_DEPTH służą do ustawienia wysokości ekranu i głębi kolorów.

MOUSE_SUSCEPTIBILITY służy do ustawienia czułości myszki. Jest to opcja niedostępna z poziomu menu. Większa czułość zwiększa celność podczas gry, lecz wymaga również większych ruchów, gdy chce się wykonać gwałtowny obrót.

FRAMERATE_LIMITER służy do ustawienia górnej granicy liczby klatek na sekundę. Podczas gry grafika będzie wyświetlana maksymalnie określoną liczbę razy na sekundę.

DRAW_DISTANCE służy do ustawienia opcji rysowania tylko obiektów, które znajdują się blisko.

LIGHTS opcja ta służy do ustawienia trybu wyświetlania grafiki z efektami świetlnymi.

CAPTURE_QUALITY opcja ta służy do ustawienia jakości przechwytywania obrazu wideo podczas gry. Gdy ustawiona jest opcja CAPTURE_QUALITY to każda klatka podczas gry zostanie przechwycona i zapisana na dysk. Opcja ta umożliwia nagrywanie filmów, a wartość CAPTURE_QUALITY określa jakość obrazu video. Wartość 100 oznacza obraz najlepszej jakości, ale także zajmujący najwięcej miejsca na dysku twardym. Im mniejsza wartość tym obraz gorszej jakości.

ZDALNIE. Opcja ta służy do odgrywania wcześniej nagranej rozgrywki z pliku z savem. Wartość jaka przypisana jest zmiennej ZDALNIE określa prędkość odtwarzania rozgrywki. Domyślną wartością jest liczba 50, co jest przyjętym przez nas przelicznikiem prędkości gry.

1.5.3. SECTION CHAT:

W sekcji przeznaczonej na chat można wpisywać teksty jakie mają się pojawiać, gdy ktoś nas zabije podczas gry. Teksty te podczas rozgrywki multiplayer są wysyłane do wszystkich graczy.

Schemat napisów przedstawia poniższy przykład:

```
^R$LOOSER: ^G$MASTER^W to buc
```

Napisy zostają wysłane jako zwykłe wiadomości chat, więc dotyczą ich takie same zasady tworzenia i wypisywania. Szczegółowy opis znaczeń symboli znajduje się w dziale opisującym chat.

1.5.4. SECTION PLAYER:

NAME zmienna ta określa jakie imię ma gracz rozpoczynający rozgrywkę. Imię to zostanie ustawione jako imię gracza i będzie widoczne w menu i podczas całej rozgrywki.

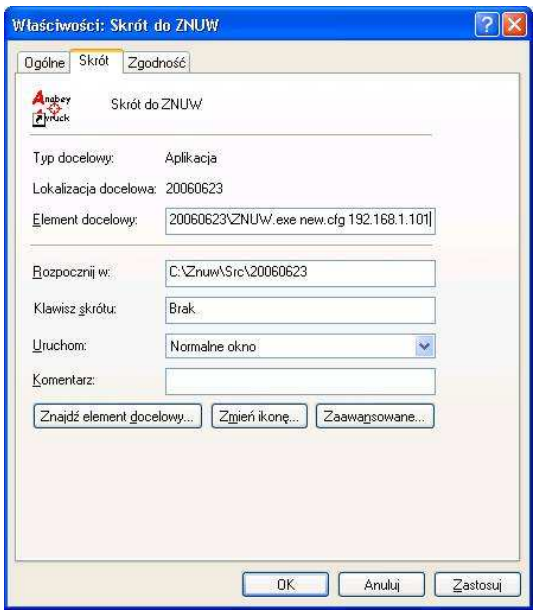
SKIN zmienna ta określa skórkę jaką będzie miał gracz podczas gry.

MODEL zmienna ta określa jakiego modelu będzie używał.

Ustawienie jakiejkolwiek z wyżej wymienionych opcji konfiguracyjnych powoduje zaaplikowanie jej do gry. Jedyną możliwą zmianą tych opcji jest późniejsza ich modyfikacja z poziomu menu.

1.6. Konfiguracja gry z linii poleceń

W grze istnieje możliwość konfiguracji z poziomu linii poleceń. Ilustruje to poniższy przykład:



Argumenty jakie zostają przekazane do programu mają ściśle określone znaczenia, które są opisane w kolejnych podrozdziałach.

1.6.1. Pliki .cfg:

Po nazwie pliku wykonywalnego istnieje możliwość podania nazw plików konfiguracyjnych. Opis znaczenia opcji w pliku konfiguracyjnym został zamieszczony w dziale (”Konfiguracja z pliku .cfg”). Jeżeli w linii poleceń zostaną podane pliki konfiguracyjne to zostaną one zaaplikowane do gry. Wszystkie po kolei.

Umożliwia to tworzenie wielu plików konfiguracyjnych zawierających ściśle określone funkcje przeznaczone do szczególnych trybów rozgrywki lub do pojedynczych plansz. Jednym z zastosowań jest tworzenie osobnych plików konfiguracyjnych do konfiguracji chata, i innych do konfiguracji samych ustawień gry.

1.6.2. Adres IP:

Możliwe jest podanie również jako opcji z linii poleceń adresu IP serwera, na którym odbywać się będzie partia.

Rozdział 2

Rozgrywka i postać gracza

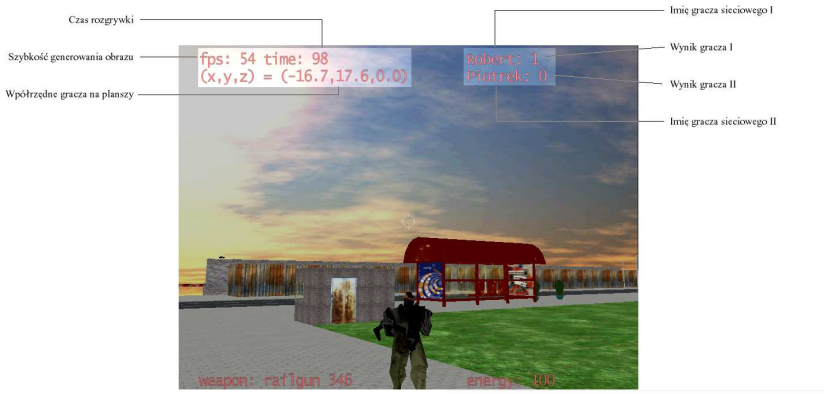
Do wirtualnego świata zanurzamy najpierw wszelkiego rodzaju obiekty. Grunt wpierw, fale na morzu też, budynki potem, pociąg i okręt oraz wszelki inny dobrobyt, a kwantu siódmego wśród nich umieszczamy graczy.

„(...)
- Nie będę celował ręką, gdyż kto nią celuje, ten zapomniał oblicza swego ojca.
- Będę celował okiem.
- Nie będę strzelał ręką, gdyż kto nią strzela, ten zapomniał oblicza swego ojca.
- Będę strzelał umysłem.
- Nie będę zabijał bronią, gdyż kto nią zabija, ten zapomniał oblicza swego ojca.
- Będę zabijał sercem.(...)”

„Rewot Krad, Sztuka Walki” Gnik Nephets (Piks Wokzsurt przełożył ze starożytnego języka Shilgne)

2.1. Rozgrywka

Gra jest typu TPP (Third Person Perspective), więc postać gracza jest cały czas widoczna, tuż obok zdarzeń dziejących przed nią.



Obiekt gracza jest jednym z wielu obiektów w hierarchii. Bo przecież w grze widzimy i obiekty statyczne(budynki, grunt...) oraz ruchome(gracze, pociski...). I tak gracz jest pewnym szcze-

gólnym przypadkiem obiektu w grze. Podobnie rzecz się ma z botem(szczególny przypadek gracza). Dzięki takiej hierarchii dla modelu samej gry nie jest istotne czy gracz jest kierowany z klawiatury i myszki czy za pomocą pewnych algorytmów. Wobec czego łatwo o umieszczanie na planszy botów, i to różnych, bo tak jak bot jest szczególnym przypadkiem gracza tak można sobie zdefiniować różne typy botów. Ale o botach już będzie osobny rozdział.

Flatline jest grą czasu rzeczywistego, więc cały ruch, zachowanie się obiektów musi symulować dobrze rzeczywistość oraz dawać graczowi poczucie dynamizmu akcji. I tak w ciągu sekundy jest przetwarzanych ileś klatek, nazwijmy to „kwantami" czasu. Taki kwant to niepodzielny ciąg instrukcji potrzebny do właściwego zarządzania. Każdy kwant składa się na:

rysowanie obiektów – rysowane są tylko te obiekty, które główny gracz akurat widzi - niebo, grunt, budynki, gracze, pociski...

zebranie zdarzeń – odczyt danych z klawiatury i myszki(ruch gracza głównego), sprawdzenie jakie ruchy chcą wykonać boty,

badanie kolizji – przekazanie ruchomych obiektów do sprawdzania kolizji,

wysłanie danych do sieci – przekazanie odpowiednich danych do serwera.

Powyższe instrukcje są wykonywane w pętli, aż do momentu zakończenia gry przez użytkownika. Kwanty czasu mogą trwać różne okresy czasu, czasami potrzeba rozrysować mniej bądź więcej obiektów, innym razem potrzeba rozpatrzeć bardzo dużo kolizji a innym razem prawie w ogóle. Ponadto do sieci nie musimy wysyłać informacji za każdym razem (dokładniej o tym w rozdziale poświęconym sieci). Wobec czego musi istnieć pewien przelicznik między czasem rzeczywistym(jaki chcemy widzieć w grze a prawdziwym, związany z szybkością komputera oraz faktem, iż każdy kwant może mieć inny czas trwania). Z tego powodu co pewną liczbę klatek sprawdzane jest ile one zajęły czasu. Potem porównywane jest z wzorcowym czasem i wtedy przelicznik jest odpowiednio zwiększany lub zmniejszany. Od tego przelicznika jest uzależnione wszystko co się rusza – pociski, gracze, animacje... Bo konieczne jest aby np pocisk przelatywał odpowiednią odległość zależnie od prawdziwego czasu a nie od ilości kwantów w sekundzie.

2.2. Gracz, jego animacja i płynność ruchu.

Postać gracza posiada szereg animacji zależnych od rodzaju ruchu. Gracz może się poruszać przód-tył na boki oraz wykonać unik, do tego dochodzą analogiczne animacje z bronią. Ruch gracza odbywa się za pomocą zdarzeń – ruch do przodu, ruch do tyłu, ruch w bok, unik, strzał. Wywołanie każdego z tych zdarzeń powoduje, jeśli jest taka potrzeba, zmianę trybu animacji(na przykład z animacji biegu z bronią na animacje strzału). Co ważne, klatki animacji gracza są wartościami rzeczywistymi. Co to oznacza? Jeśli klatka ma wartość np 5.7 to jest przeliczana klatka „pośrednia" miedzy 5 klatka a 6 zachowując odpowiednie proporcje, pozycja każdego wierzchołka jest uśredniana wg proporcji 0.7(dla klatki 5) do 0.3(dla klatki 6). Dzięki wspomnianemu wcześniej przelicznikowi szybkości komputera cała animacja jest płynna, bo nawet jeśli nagle gra będzie obrabiać mniej(więcej) wywołań na „kwant" gry to odstępy między kolejnymi klatkami animacji będą malały(rosły) a to jest możliwe bo przecież numery klatek są wartościami rzeczywistymi.
Animacja postaci gracza musi również uwzględniać dokańczanie wykonywanych ruchów. Jeśli

gracz biegnie to nie powinien się on zatrzymać w rozkroku. Oczekuje się aby gracz sam dokończył ruch. I jest to realizowane przez wprowadzenie czasu oczekiwania na zakończenie ruchu, jeśli ten czas mija i klatka animacji nie jest ustawiana w przybliżeniu na 0 to uruchamiane jest samo-zakończenie ruchu, które może być przerwane jedynie przez chęć wykonania jakiegoś ruchu przez gracza. Samo-zakończenie ruchu odbywa się do momentu uzyskania klatki bliskiej 0. Współczynnik jaki określa bliskość od zera jest związany z przelicznikiem szybkości komputera stąd nie ma możliwości aby samo-zakończenie nie mogło się wpasować w 0.

2.3. Boty jako szczególni gracze

W początkowym naszym zamyśle gry, nie miała występować żadna sztuczna inteligencja, gra miała być tylko grą przez sieć. Jednak gdy jeszcze brakowało obsługi sieciowej, trzeba było w jakiś sposób testować to co już jest w grze zrobione. Wtedy był rozwijany tryb „singleplayer". Można więc było do woli testować wygład obiektów, animacje, wszystko to co nie ma bezpośredniego związku z grą po sieci. Co jednak zrobić ze strzelaniem, ginięciem kolizjami samych graczy, kolizjami gracz - pocisk? W między czasie rodziła się komunikacja sieciowa, która pozwalała na interakcję wielu graczy. Mimo tego potrzebna była wygoda sprawdzania poprawności stosowanych algorytmów. Stąd, narodził się pomysł przemycenia do gry postaci botów, – przedefiniowanej postaci gracza która zamiast działać z klawiatury działałaby według pewnego algorytmu. Dzięki temu, że stworzenie postaci bota ograniczało się jedynie do paru operacji „Copy and paste" oraz rozpisania algorytmu, sama gra „nie wie", że ta dodatkowa postać do właśnie bot, żaden z graczy nie jest pod żadnym elementem istotnie inny. Oczywiście oprócz głównej postaci, która się steruje, ale i tu system po prostu musi jedynie wiedzieć kto jest tą postacią i nic więcej, równie dobrze główną postacią mógłby być bot. I tak tryb „singleplayer" stał się przy okazji dobrym treningiem, wprowadzeniem, przed właściwą rozgrywką. Z tego powodu zdecydowaliśmy się pozostawić tę postać w grze. I tak w finalnej wersji obok trybu „multiplayer" jest dostępny tryb „singleplayer", który jest swoistym zapoznaniem się z grą, z jej wirtualnym światem i jego zasadami. I dopiero później uruchomienie trybu „multiplayer" jest wejściem do prawdziwej rozgrywki(niczym wejście do „Matrixa"). Standardowo w grze występują dwa typy botów. Bot piechur – czyli bot nieagresywny, jedynie chodzi po planszy. Bot ochroniarz – pilnuje wybranego punktu na planszy, jeśli ktoś się do niego zbliży, przypuszcza atak. Możemy je łatwo umieścić na planszy podając w pliku mapa.dat w sekcji OTHER:

```
... SECTION OTHER SIMPLEBOT data/model/gracz/gracz.3ds -5.0 -10.0
0.0 0.0 1.0 <- piechur(postać mężczyzny) na (-5,-10,0) obrót o 0st.
energia 1punkt SECUREBOT data/model/graczka/gracz.3ds -5.0 15.0 0.0
0.0 1.0 <- ochroniarz(postać kobiety) na (-5,15,0) obrót o 0st.
energia 1punkt ...
```

W przyszłości, kolejne wersje mogłyby zawierać dodatkowe rodzaje botów, operujące na innych algorytmach. I tak – patrolujący bot, czy bot szukający zaczepki – idący do najbliższego gracza, oraz być może jakiś znacznie bardziej zaawansowany, o nazwie roboczej Smith.

2.4. Mapa świata

Standardowo udostępnione są trzy mapy: „miasto.dat", „teren.dat" i „okret.dat". Ale każdy może stworzyć własną mapę w oparciu nawet o zupełnie inne modele obiektów, inne budynki a nawet inne postacie graczy!

2.4.1. Pliki .dat

Plik „mapa” jest podzielony na sekcje(grunt, statyczne, ruchome, inne, startowe punkty). Każdy obiekt musi mieć przypisane współrzędne oraz plik 3ds(wygląd 3D). Poniżej przykładowy plik:

```
SECTION GROUND <- sekcja gruntu
-200.0 -200.0 0.0 400.0 400.0
```

Współrzędne lewego dolnego rogu świata (x,y,z) oraz rozmiar (x,y)

```
data/model/ground/ 0.0 0.0 0.0 0.0
```

katalogu z plikiem .3ds z gruntem oraz wsp. (x,y,z) i rotacja

```
SECTION STATIC <- obiekty statyczne { katalog, (x,y,z), rotacja,
energia }
data/model/hangar/ -59.0 -26.5 0.0 0.0 1.0 <- hangar
data/model/oil/ -60.5 -7.5 0.0 0.0 1.0 <- butla oleju

SECTION MOVABLE <- sekcja na przyszłość, nieużywana
SECTION OTHER <- obiekty nie standardowe
{ Rodzaj, plik 3ds, (x,y,z), rotacja, energia }
SPIN data/model/radar/model.3ds 30.0 10.0 43.3 0.0 1.0 <- obiekt obracający,
się tutaj jest to radar
WEAPON supershotgun -30.0 -10.0 1.37 0.0 1.0
```

dla broni nie trzeba podawać pliku 3ds, wystarczy podać typ.

```
MISSILEBOX railgun -139.5 1.5 0.0 0.0 1.0 <- magazynek dla railguna
HEALTHBOX data/model/health/model.3ds 97.0 -2.5 0.0 0.0 1.0  <- apteczka
SIMPLEBOT data/model/graczka/gracz.3ds -5.0 -10.0 0.0 0.0 1.0 <- bot piechur

SECTION START <- tam się gracze odradzają po śmierci
0.0 0.0 0.0
-12.0 -13.0 0.0
200.0 10.0 0.0
```

2.4.2. Samodzielne tworzenie map

Potrzebne będzie do tego jedynie edytor tekstowy oraz jeśli chcemy tworzyć własne modele 3D – program mogący zapisać obiekt 3D do formatu 3ds, na przykład 3D studio max. Oraz co najważniejsze – zamysł.

Grunt

Na początku potrzebna jest deklaracja jak duży jest świat gry, następnie umieszcza się obiekty odpowiadające za grunt w grze.

Obiekty statyczne

Tutaj jest największa dowolność doboru obiektów. Wszystkie standardowe obiekty znajdują się w katalogu „data/model/" a tekstury do nich w „data/texture".

Bronie, magazynki, apteczki

Dostępne standardowo bronie to:

shotgun – każdy z graczy go ma,

supershotgun – unowocześniona wersja shotguna, lepszy mechanizm odrzutu wysyła pociski częściej i z większą prędkością,

blaster – broń o dużej sile rażenia ale wymagająca długiego czasu przeładowania,

rocket – duża siła rażenia, szybkie pociski,

railgun – nie wyróżnia się wprawdzie, ale uśrednione wartości parametrów tworzą tę broń bardzo niebezpieczną,

machinegun – marzenie każdego gracza, majstersztyk wirtualnych inżynierów, krótki czas przeładowania i szybkie pociski.

Gracz może zebrać tylko jedną broń tego samego typu. Dlatego na planszy można rozmieszczać magazynki. Magazynki są różne dla różnych rodzajów broni, można je zbierać nawet jak nie posiada się odpowiedniej broni. Ponadto gracz może podreperować swoje zdrowie apteczkami.

Inne

Na planszy można również umieszczać inne obiekty, samemu przez siebie zdefiniowane:

```
SECTION OTHER
TRAIN data/model/train/model.3ds -137.5 100.0 4.35 180.0 1.0 <- pociąg
CLOCK data/model/clock/model.3ds -105.0 111.0 5.8 90.0 1.0 <- zegar
SIMPLEBOT data/model/gracz/gracz.3ds -5.0 -10.0 0.0 0.0 1.0 <- piechur

SECUREBOT data/model/graczka/gracz.3ds -5.0 15.0 0.0 0.0 1.0 <- ochroniarz
```

Standardowo w grze są takie odmienne obiekty: pociąg, zegar, obiekt obrotowy oraz fale morskie. Można na planszy umieścić również postaci botów(dokładniej w rozdziale o botach).

Pozycje startowe

Pozycje odradzania się graczy gdy zginą. W czasie gry dobierane są tak aby nie powstały kolizje. Jednak jeśli tworzy się nową mapę trzeba samemu zadbać o to, żeby punkt nie znajdował się wewnątrz budynku, do którego nie ma wejścia.

Pliki .3ds

Pliki 3ds przedstawiają modele obiektów 3D. Każdemu obiektowi odpowiadają dwa pliki: model.3ds i model0.3ds (w pewnych przypadkach więcej, jeśli obiekt jest animowany jak gracz, bądź składa się z wielu obiektów). Pliki model.3ds to obiekty takie jakie chcemy by były widziane, z teksturami i odpowiednio ukształtowane. Z kolei pliki model0.3ds to obiekty takie jakie chcemy żeby widziała je gra, zwłaszcza jest to istotne dla badania kolizji, czyli obiekty bez tekstur, bez zbędnych szczegółów.

Rozdział 3

Silnik graficzny.

> (...) Sztuka przemocy i barbarzyństwa nie znosi! Ale czy też Tytani nie byli panami przepięknego zniszczenia?? Dlatego właśnie siła w malunku i innych dziełach musi być widoczna! Bez żalu... Przecież to tylko kawałek materiału pokryty różnego rodzaju plamkami (...)
>
> *„Kropki i kreski na malunek się składające" Orjeh Ceicjow*

3.1. Ogólnie słów kilka

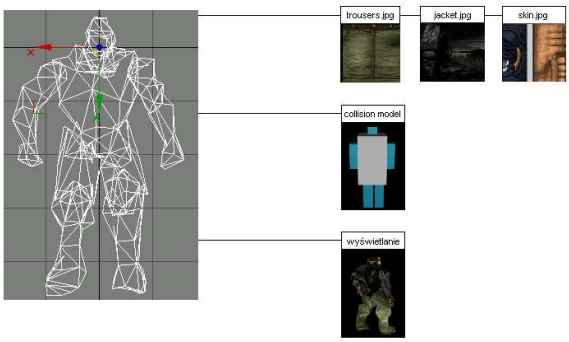
Silnik graficzny zajmuje się grafiką i różnymi potrzebnymi funkcjami. Najważniejszym elementem silnika gry są funkcje wyświetlania grafiki (2D i 3D). Główne zadanie jakie spoczywa na silniku graficznym to wyświetlanie płynnej grafiki trójwymiarowej. W związku z tym przy projektowaniu silnika graficznego wykorzystywane są złożone metody przyspieszania procesu wyświetlania obrazu. Stanowią one jedne z najważniejszych algorytmów silnika.

Sam proces wyświetlania spoczywa na karcie graficznej. Silnik komunikuje się z nią poprzez zbiór podstawowych poleceń. Do wyświetlania grafiki użyliśmy poleceń biblioteki OpenGL. W wielu algorytmach przyspieszających wyświetlanie grafiki skorzystaliśmy z gotowych rozwiązań dostarczonych przez bibliotekę OpenGL. Były to na przykład: mipmaping oraz wsparcie przy wyświetlaniu przezroczystych obiektów.

3.2. Ładowanie obiektów 3D.

Obiekty 3D ładowane są z pliku z rozszerzeniem 3ds. Plik taki zawiera wszystkie potrzebne informacje do wyświetlania na ekranie. Algorytmy odpowiedzialne za ładowanie obiektów 3ds pobierają informacje o:

- liście wierzchołków
- nazwach użytych tekstur
- kolejności wyświetlania wierzchołków
- współrzędne nanoszenia tekstury

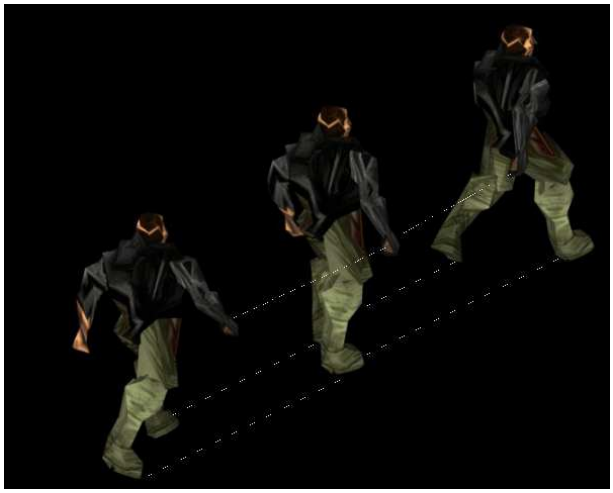


Na podstawie tych informacji tworzone są Indeksowane tablice wierzchołków, które przyspieszają proces wyświetlania obrazu 3D, oraz lepiej zarządzają pamięcią karty graficznej.

Tworzona jest również lista plików graficznych, które należy załadować. Na podstawie listy wierzchołków tworzone są uproszczone modele, które pozwalają szybciej sprawdzać kolizje. Modele kolizji są również wykorzystywanie w algorytmach usuwania niewidocznych obiektów.

3.3. Wyświetlanie grafiki 3D.

3.3.1. Wyświetlanie animacji



Tworzenie animacji postaci i innych animacji użytych w grze (m.in. animacje wybuchów i krwi) było jednym z najtrudniejszych problemów z jakimi natknęliśmy się podczas tworzenia grafiki. Problem polegał na tym, że należy użyć lub stworzyć taki format zapisu, który będzie spełniał wszystkie wymagania i będzie działać z wcześniej zaprojektowanym kodem.

W związku z tym stworzyliśmy własne rozwiązanie oparte o pliki 3ds, których algorytm ładujący wcześniej został zaprojektowany.

Pomysł polegał na tym, aby animacja składała się z kilku do kilkunastu klatek kluczowych, a algorytm będzie na ich podstawie generował płynne przejście między nimi.

Bierzemy dwie kolejne klatki i płynne przejście między nimi otrzymujemy przybliżając liniowo (aproksymacja liniowa).

alfa - współczynnik skalowania miedzy kolejnymi klatkami:

wynik_skalowania = (1 - alfa) * pierwsza + alfa * druga

Ta prosta zasada pozwoliła nam łatwo tworzyć bardziej skomplikowane animacje. Na przykład animację wybuchu. W efekcie tym ogień płynnie zmienia swoje kształty. Wystarczy zaprogramować jaki efekt chcemy otrzymać, a ogień sam do niego płynnie się dostosuje.

3.3.2. Przezroczystość

Zarówno podczas tworzenia grafiki 2D jak i 3D wyświetlane są obiekty przezroczyste i półprzezroczyste. Pełnią one bardzo ważną rolę. Bez nich niemożliwe byłoby tworzenie wielu efektów takich jak ogień czy szyba, ale również wyświetlanie prostych obiektów 2D takich jak napis na ekranie.

Aby stworzyć przezroczysty obraz należy odpowiednią teksturę załadować z dysku. W związku z tym silnik graficzny musi mieć możliwość odczytu różnych formatów zapisu tekstur. Między innymi przezroczystych tekstur takich jak pliki *.tga i *.raw.

Napisy są szczególnym typem grafiki 2D. Posiadają one całkowicie nieprzezroczyste ciało litery i całkowicie przezroczystą resztę.

Przykładem półprzezroczystej grafiki w obrazach 3D jest szyba.

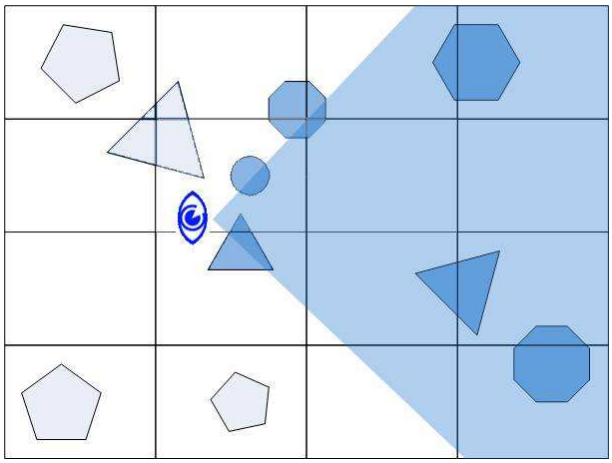
Szyba jest przykładem obiektu o różnej przezroczystości w różnych miejscach. Począwszy od prawie całkowicie przezroczystych obszarów do niemal całkowicie nieprzepuszczalnych dla światła.

Efekt przezroczystości otrzymaliśmy wykorzystując odpowiednie polecenia biblioteki OpenGL i integrując je w silnik.

3.4. Usuwanie niewidocznych obiektów.

W celu poprawienia szybkości generowania grafiki 3D stosowałem różne techniki. Kluczową z niech jest algorytm usuwania niewidocznych obiektów. Algorytm działa w oparciu o bardzo prostą zasadę: Jeżeli obiekt nie jest widoczny to nie trzeba go wyświetlać na ekranie.

Poniższy rysunek prezentuje uproszczoną zasadę działania algorytmu. Jeżeli w obszarze widzenia gracza znajduje się jakiś obiekt to prawdopodobnie będzie trzeba go narysować. Natomiast jeśli obiekt znajduje się za graczem, lub w obszarze martwym po jego bokach to obiektu na pewno nie trzeba będzie rysować.

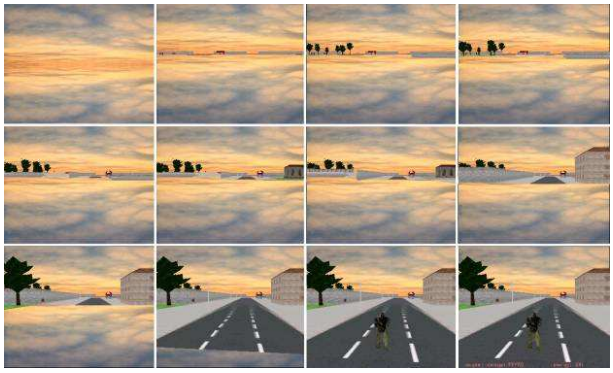


Na tym wstępnym etapie zostaje odrzuconych około 75 procent obiektów, których nie będziemy wyświetlać.

3.5. Potok wyświetlania.

Potok wyświetlania pełni bardzo ważną funkcję. Stworzenie pełnej klatki animacji wymaga wielu przejść między trybem wyświetlania grafiki 2D i 3D. Łączy on ze sobą wszystkie algorytmy wielokrotnie odwołując się do prostych operacji rysowania trójkąta.

- Proces tworzenia kolejnej klatki animacji rozpoczyna się od wyczyszczenia z ekranu efektów renderingu poprzedniej sceny.
- Następnie rysowane są obiekty począwszy do położonych najdalej do tych, które znajdują się najbliżej.
- Najpierw rysowane jest niebo, bo zakładamy, że znajduje się najdalej i wszystkie obiekty będą się znajdować przed nim. Niebo jest obiektem 2D.



- Następnie nanoszone są obiekty znajdujące się najdalej. Potem kolejne przesłaniając te które znajdują się za daleko. Kolejne obiekty znajdują się coraz bliżej i zajmują coraz większy obszar ekranu.
- Ostatnim elementem 3D jest gracz.
- A na samym końcu nanoszone są napisy 2D.

Rozdział 4

Kolizje

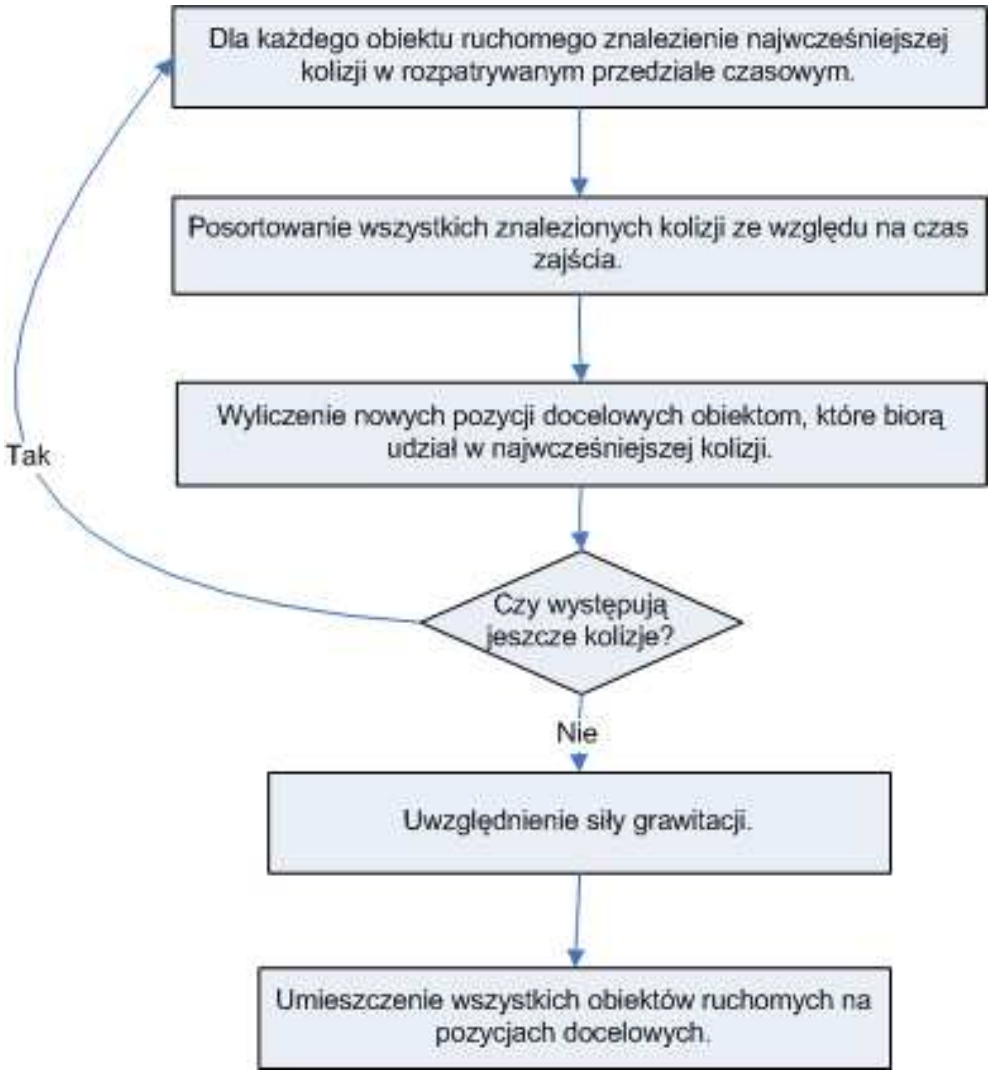
Ten rozdział opisuje system wykrywania kolizji zastosowany w grze. Zostanie zaprezentowany ogólny schemat postępowania oraz szkicowo przedstawione algorytmy wykorzystywane na poszczególnych etapach detekcji zderzeń.

(...)
- I wtedy wpadli na siebie.
- Dokładniej proszę! Który z nich wpadł na tego drugiego?
- Czy to ważne Wysoki Sądzie? Ważne jest, co nastąpiło później (...)

„Początek końca" B. Nicram

4.1. Ogólny schemat

System sprawdzania kolizji jest uruchamiany pomiędzy wyświetleniem dwóch kolejnych klatek na ekranie. W tym czasie wiele obiektów chce zmienić swoją pozycję. Zazwyczaj część graczy naciśnie którąś ze strzałek na klawiaturze a dodatkowo prawie zawsze w powietrzu mkną wystrzelone pociski. Każde takie żądanie przemieszczenia składa się z obecnej oraz żądanej docelowej pozycji obiektu (w przypadku braku ruchu w danej chwili oba powyższe położenia stanowią jeden punkt). Jednak w momencie powstania żądania program nie wie czy jest ono dozwolone - przykładowo gracz stojąc przed ścianą może chcieć iść do przodu. To właśnie zadaniem systemu detekcji zderzeń jest skorygowanie żądanych docelowych pozycji, tak aby całość była zgodna z podstawowymi prawami fizyki. Ogólny schemat działania modułu przedstawia poniższy rysunek.



Na początku dla każdego obiektu ruchomego znajdowana jest jego najwcześniejsza kolizja lub stwierdza się że obiekt z niczym i nikim się nie zderza. Przedział czasowy poszukiwań obejmuje okres pomiędzy wyświetleniem dwóch kolejnych klatek. Następnie wszystkie znalezione kolizje są sortowane po czasie wystąpienia. Pierwsze a więc globalnie najwcześniejsze zderzenie jest aplikowane do świata gry: zmieniane są docelowe położenia obiektów biorących udział w nim udział. Aktualne pozycje pozostają jednak niezmienione - nikt się na razie nigdzie nie rusza. Wyliczone położenia mają zapobiec kolizji (np. zatrzymać gracza przed ścianą). Jednak po uaktualnieniu nadal mogą występować kolizje - np. pomiędzy innym graczem a pociskiem. Wtedy wykonywana jest kolejna iteracja. W przeciwnym wypadku uwzględniana jest siła grawitacji i wszystkie obiekty ruchome (oczywiście poza pociskami) umieszczane są na gruncie. Na samym końcu aktualne pozycje zmieniane są na docelowe.

Główną i najbardziej złożoną częścią przedstawionego algorytmu jest oczywiście znalezienie pierwszej kolizji dla każdego obiektu. Opisuje ją kolejny podrozdział.

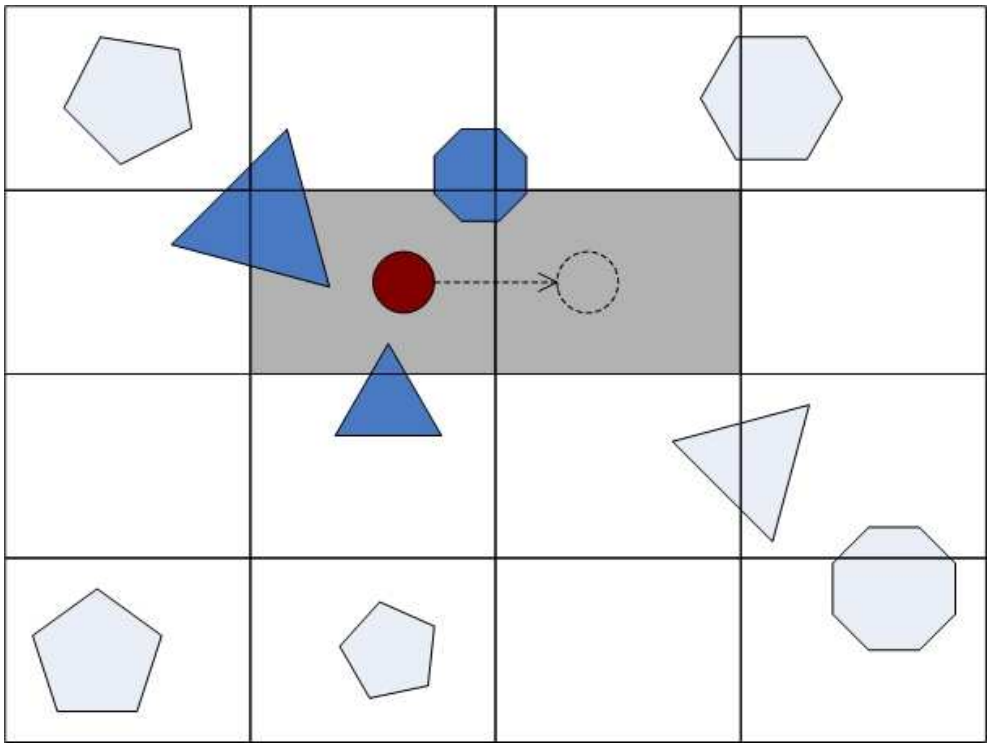
4.2. Wykrywanie kolizji

Dobry system wykrywania kolizji musi spełniać dwa wymagania. Z jednej strony musi być dokładny, czyli wykrywać wszystkie zachodzące kolizje. Większa precyzja przekłada się na

wzrost realizmu gry. Z drugiej zaś strony powinien działać szybko, aby nie okupić zwiększonego realizmu spowolnieniem całej gry. Strzelanka, aby była grywalna, musi być przede wszystkim dynamiczna. Niestety powyższe wymogi bardzo często są ze sobą sprzeczne. Im precyzyjniej opisany gracz, tym wykrywanie jego interakcji z otoczeniem staje się bardziej złożone. Dlatego też praktycznie we wszystkich grach rozgrywających się w świecie trójwymiarowym do sprawdzania kolizji stosuje się oddzielne, uproszczone modele bądź struktury. Nadal jednak pozostaje jeden problem do rozwiązania: ilość możliwych zderzeń. Przy kilku poruszających się graczach, kilkunastu pociskach w powietrzu oraz setkach nieruchomych obiektów liczba potencjalnych kolizji może sięgać kilkunastu tysięcy na pojedynczą klatkę. Na szczęście wiele zderzeń można w prosty sposób wykluczyć. Dla przykładu gracz znajdujący się na środku planszy nie zderzy się z drzewem znajdującym się w jej rogu. Powszechnie stosowanym rozwiązaniem jest podział wykrywania kolizji na dwie fazy. Pierwsza z nich pełni rolę sita, które przepuszcza dalej tylko obiekty znajdujące się blisko gracza. W drugiej natomiast wykonywane jest dokładniejsze sprawdzenie kolizji z obiektami, które nie zostały odrzucone. W naszej grze również występuje taki podział. Z każdym obiektem związane są dwa uproszczone modele: oddzielny do każdego etapu badania zderzeń. Dalsze podrozdziały opisują bardziej szczegółowo kolejne fazy.

4.2.1. Faza odrzucenia dalekich obiektów

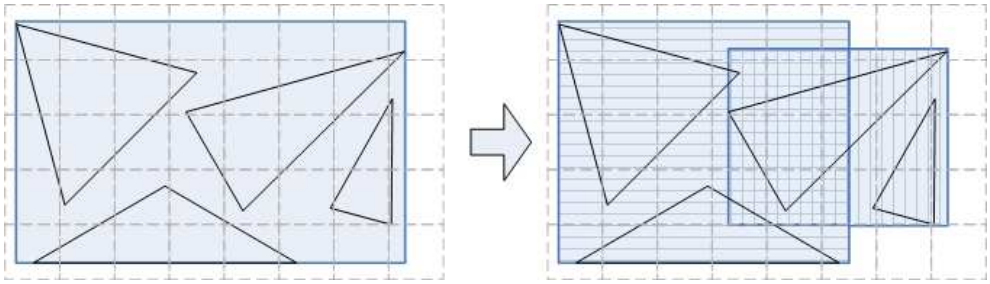
Podczas tego etapu wybierane są obiekty, które znajdują się blisko rozpatrywanego obiektu, tak że kolizja z nimi jest możliwa. Istnieje wiele metod realizacji wspomnianego celu. W naszej grze podzieliliśmy mapę na prostokątne obszary o jednakowym rozmiarze. Każdy obiekt posiada również własny model, będący najmniejszym otaczającym go prostokątem o krawędziach równoległych do osi układu współrzędnych wyznaczonego przez mapę. Każdy obiekt statyczny, taki jak budynek, ogrodzenie itp., ma na stałe wyznaczony otaczający go prostokąt. Natomiast model obiektu ruchomego uwzględnia jego bieżącą oraz docelową pozycję: znajdowany jest najmniejszy prostokąt zawierający oba położenia obiektu. W drugiej fazie algorytmu rozpatrywane są tylko i wyłącznie obiekty znajdujące się w obszarach, z którymi przecina się model obiektu ruchomego. Ideę ukazuje rysunek poniżej. W celu zwiększenia przejrzystości prostokątne modele obiektów nie zostały narysowane.



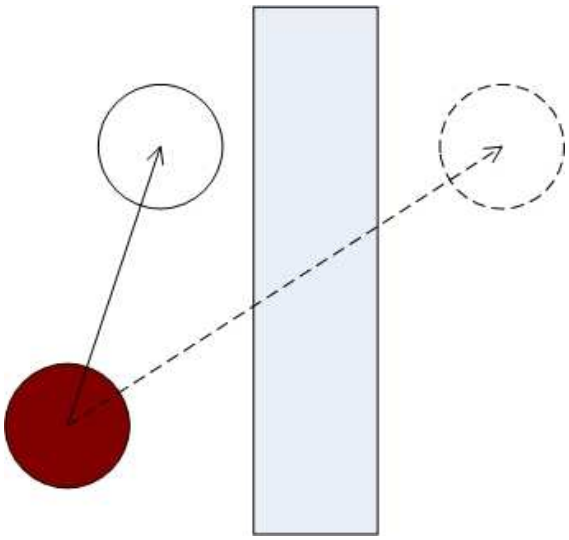
Czerwone koło reprezentuje ruchomy obiekt, którego docelowa pozycja została zaznaczona przerywanym okręgiem. Jego model przecina się z dwoma zaciemnionymi obszarami mapy. Tylko trzy obiekty przecinają się z tymi obszarami. Zostały one pokolorowane na ciemnoniebiesko. Dzięki uproszczeniu modeli do prostokątów o krawędziach równoległych do osi wszelkie obliczenia wykonują się bardzo szybko, a znalezienie bliskich obiektów zawsze zajmuje czas stały.

4.2.2. Faza dokładnego sprawdzania bliskich obiektów

W tej fazie wykonywane jest dokładniejsze sprawdzanie kolizji. W tym celu używane są inne modele, różne dla każdego rodzaju obiektów. Model dla postaci gracza to otaczająca go kula. Pocisk reprezentowany jest przez punkt. Kształty te zostały wybrane ze względu na prostotę obliczeń a dzięki temu szybkość. Obiekty statyczne natomiast są reprezentowane przez trójkąty nań się składające z nałożonym drzewem AABB (Axis Aligned Bounding Box - otaczający prostopadłościan z krawędziami równoległymi do osi układu współrzędnych). Struktura ta przyspiesza wykrywanie zderzenia dwóch obiektów. Chociaż sprawdzanie kolizji trójkąta z przemieszczającą się kulą jest dość szybkie, to przy setkach lub tysiącach trójkątów okazuje się zbyt kosztowne. Idea drzewa AABB, sprowadza się do wyliczenia najmniejszych prostopadłościanów z krawędziami równoległymi do osi układu współrzędnych, które otaczają dany zbiór trójkątów. Prostopadłościan w korzeniu drzewa zawiera wszystkie trójkąty składające się na dany obiekt. Potem następuje podział trójkątów na dwa zbiory względem najdłuższej krawędzi prostopadłościanu: do pierwszego trafiają trójkąty, których środek ciężkości znajduje się nad środkiem najdłuższej krawędzi, a do drugiego wszystkie pozostałe. Następnie w ten sam sposób rekurencyjnie liczone są poddrzewa. Proces jest przerywany w momencie, gdy liczba trójkątów zmniejszy się do ustalonej wcześniej wartości. Poniższy rysunek ilustruje przykładowe drzewo AABB dla dwóch wymiarów (w grze wykorzystywane są trójwymiarowe).



Po lewej stronie mamy prostokąt znajdujący się w korzeniu - zawiera on cały zbiór trójkątów. Po prawej stronie pokazani są jego synowie: jeden z nich reprezentowany jest przez prostokąt z poziomymi, a drugi z pionowymi liniami. Prostokąty te nie są rozłączne, gdyż to środek ciężkości trójkąta określa jego przynależność do danego syna. Łatwo zauważyć, że jeśli jakiś obiekt nie przecina się z prostopadłościanem, to nie będzie przecinał się z żadnym z trójkątów zawartych w tym prostopadłościanie - dzięki temu zyskujemy na szybkości: zamiast sprawdzać zderzenie z każdym trójkątem wystarczyło z otaczającym go prostopadłościanem. Dla każdego obiektu ruchomego rozpatrywane są wszystkie znajdujące się blisko niego obiekty i dla każdego sprawdza się, czy w którymś momencie ruchu kula przecina się z modelem innego obiektu. Jeśli tak to zachodzi kolizja i zapamiętywany jest czas jej zajścia oraz wyznaczone nowe punkty docelowe dla obiektów (lub obiektu jeśli zderzenie nastąpiło z nieruchomym przedmiotem), które zapobiegają jej powstaniu. Należy jednak pamiętać że nie jest ona w tym momencie uwzględniana w świecie gry. Poniższy rysunek pokazuje sposób wyznaczania nowego punktu docelowego dla obiektu ruchomego zderzającego się ze ścianą.



Czerwone koło to bieżąca pozycja obiektu. Przerywany okrąg oraz linia to pierwotne docelowe położenie oraz wektor przesunięcia. Nowe wyliczone wartości reprezentowane są przez ciągły okrąg oraz linię. Uwzględniają one zaimplementowane w grze "ślizganie się" przy ścianie.

Rozdział 5

Sieć

"Flatline" jest SIECIOWĄ strzelanką, warto więc opowiedzieć trochę o komunikacji sieciowej. O tym, w jaki sposób wszystkie te komputery komunikują się ze sobą i każdy gracz ma te same informacje o świecie gry co pozostali.

> „Ryby łowić można na sposoby różne. Chińczycy, bywa i tak, kormorana przywiązują na końcu kija i w wodę go wsadzają, szyję lekko mu spętawszy. Zdobycz tak pozyskana, po z gardła ptaszyska wyjęciu i wyszykowaniu, zjadaną jest ze smakiem wielkim. Ty jednak, uczniu mój krnąbrny, sieci zastawnej używaj! Czemuż to nie pytaj, tylko przynieś mi posiłek suty, bom zgłodniał wielce"
>
> *„Brednie i kłamstwa, acz z prawdy lekkim posmakiem" W. Anazak*

5.1. Ogólny zarys

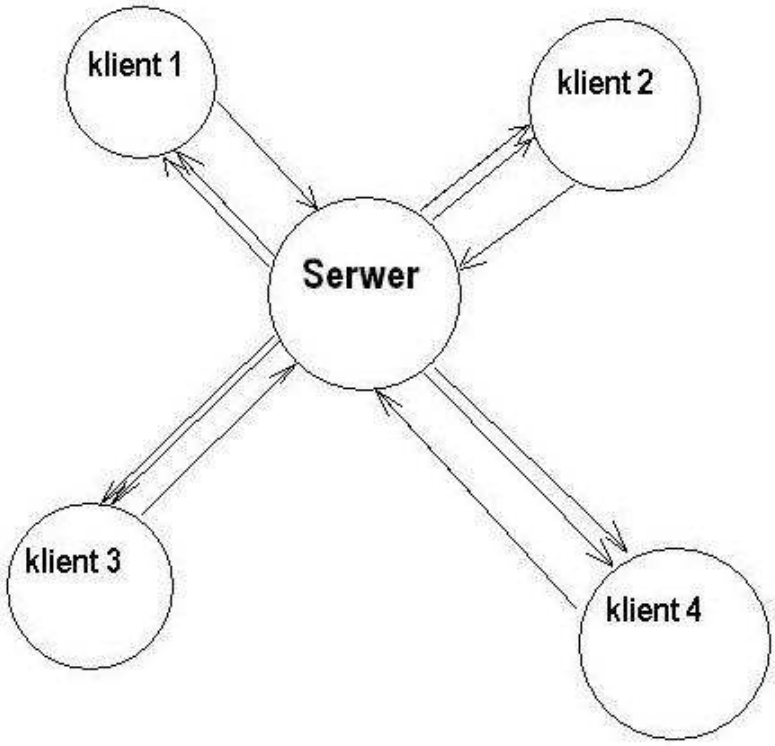
Podstawowym pytaniem, jakie postawił sobie nasz zespół na samym początku, był wybór ogólnej koncepcji komunikacji sieciowej. Czy gracze mają porozumiewać się ze sobą bezpośrednio w systemie "każdy mówi do każdego"? A może narzucić pewną hierarchię na poszczególne instancje programu i wykorzystać ten podział do zmniejszenia ilości połączeń komunikacyjnych kosztem nasilenia przepływu danych na tych pozostałych? Głosy w dyskusji były różne i przedstawiały zalety i wady każdej z opcji. Ostatecznie stanęło na:

5.2. Komunikacja klient-serwer

Oddzielny program (wchodzący w skład pakietu "Flatline") nazwany po prostu "Serwerem" zarządza wymianą danych pomiędzy graczami. Co pewien czas odbiera on dane od graczy biorących udział w rozgrywce, po czym rozsyła wszelkie informacje niezbędne do zachowania pełnej synchronizacji na każdym z komputerów. Jest to z kilku powodów bardzo wygodne:

- To oddzielny program dba o to, aby żaden z graczy nie został "pokrzywdzony" (czyli aby dostawał wszystkie informacje i aby jego działania zostały odpowiednio odnotowane przez przeciwników).
- Do zapewnienia komunikacji wystarczy podać w każdej instancji klienta (tak będziemy nazywać program grający) tylko i wyłącznie dane serwera, a nie dane każdego gracza z osobna.

- Mając oddzielny program, możemy łatwo ingerować we wszystkie instancje klienckie jednocześnie (i dzięki temu np. zsyłać dodatkowe bronie lub apteczki bez obaw, że na którymś z komputerów nie będzie to widoczne).



Ale niestety nie jest to pozbawione wad. Wad, z którymi musieliśmy walczyć (na szczęście skutecznie), aby zapewnić odpowiedni komfort rozgrywki i niezbędną dynamikę.

5.3. Serwer = wąskie gardło?

Skoro jeden program odbiera dane od poszczególnych graczy i potem je propaguje (ewentualnie dodając coś od siebie lub wykonując niezbędne sprawdzenia i testy), to komunikacja z nim nie może zawodzić w żadną ze stron. Nie mogliśmy sobie pozwolić na zbyt wolne działanie serwera, bo strzelanka jest przecież grą akcji. Grą, w której dynamika i związana z nią konieczność szybkiego podejmowania decyzji przez gracza jest niezbędna. Dlatego właśnie należało odpowiednio ograniczyć ilość (oraz częstość) przesyłanych informacji tak, aby sieć nie wywoływała odczuwalnych spowolnień rozgrywki.

5.4. Jak? No i jak często?

Początkowe stadium rozwoju gry pozwalało na komunikację pomiędzy graczami i serwerem w każdej klatce. W grze, w której nie było jeszcze możliwości strzelania ani ingerencji z planszą (zbierania apteczek, broni, magazynków, itp.) ilość przesyłanych informacji była znikoma. Protokół TCP/IP w zupełności pokrywał nasze zapotrzebowanie na szybką i pewną łączność pomiędzy klientami. I to trochę uśpiło czujność członków zespołu.

Po dodaniu możliwości strzelania stanął przed nami pierwszy poważny problem: w jaki sposób umożliwić przekazywanie informacji, aby każdy z graczy widział to samo i jednocześnie nie stracić na szybkości?! Dotychczasowe podejście nie dawało wielkiego pola manewru. I o ile jeszcze sieć lokalna na kilku komputerach dawała sobie radę, to gra np. przez Internet była już niemożliwa. Przepustowość kabli nie była barierą, z którą ktoś z nas chciał walczyć, więc należało pomyśleć.

Pierwszy pomysł - przepisanie wszystkiego na UDP - okazał się zupełnym niewypałem. Problemy z ginięciem pakietów i przede wszystkim synchronizacją mnożyły się na każdym kroku. A wizja dodania w przyszłości większych możliwości grze (wspomniane już wcześniej np. zbieranie broni) skutecznie odstraszyła nas od prób kontynuowania tej nierównej walki. Jak się później okazało, szybkie odejście od UDP było bardzo dobrym posunięciem.

Skoro pozostajemy przy TCP, a to nie wystarcza, to może nie wysyłać danych tak często? Może zamiast np. co 1 klatkę, robić to co 3 czy też 5? Ten pomysł, przez pewien czas obecny w jednej z wersji przejściowych, doczekał się w końcu rozwinięcia. W "Flatline" cała komunikacja odbywa się asynchronicznie. Serwer zbiera dane od graczy, którzy są mu je w stanie dostarczyć i dba o wysłanie informacji do klientów. Ci, zamiast bezczynnie czekać co jakiś czas na dane z sieci, sprawdzają tylko, czy przypadkiem "coś na nich nie czeka". Jeśli tak, mają wtedy pewność, że żadne zbędne czekanie nie będzie miało miejsca. Tylko odbiór i kolejne wysyłanie danych. Nic więcej. A zapewnienie, aby każdy gracz odpowiednio często dostawał informacje od innych spoczęło już na barkach serwera. Skoro programy graczy i tak w każdej klatce poświęcają czas na badanie kolizji i przechwytywanie informacji z klawiatury i myszki, nie było sensu dodatkowo ich obciążać.

Brak synchronizacji przy zapewnieniu docierania pakietów (czyli zyski z zastosowania TCP) okazały się dla nas zbawienne. Szybkostrzelne bronie, na których tak bardzo nam zależało, przestały być problemem. Przy grze nawet na 6-8 graczy nie widać żadnych spowolnień, a przecież gracze rzadko bywają spokojni. Zwykle strzelają tak długo, póki ciało ich własne lub też przeciwnika nie uderzy wreszcie o ziemię! No chyba że ten ostatni (Tak tak! Każdy zawsze to sobie powtarza!) raz znów zapomną spojrzeć na licznik amunicji, który po raz kolejny sięgnął zera.

To był już niemal koniec zabawy (bo przecież nie można tu mówić o problemach!!) z siecią. Pozostały jeszcze detale. Drobne szczególiki. W których, jak to zwykle bywa, i tym razem skrył się przysłowiowy diabeł.

5.5. Co właściwie przesyłać? I jak to wszystko od siebie odróżnić?

Chcemy przesyłać przez sieć mnóstwo informacji. O położeniu naszego gracza, o kierunku i ilości wystrzelonych pocisków, o zebranych przedmiotach z planszy, wreszcie o naszych trafieniach! I jak to wszystko od siebie odróżnić? I co z tego tak naprawdę musimy wysyłać? No i jak często?

Wiele pytań, ale na każde z nich udało nam się uzyskać zadowalającą odpowiedź. Po pierwsze informacje o graczu: gdzie się znajduje, jaką ma broń przy sobie, jaka klatka odpowiada jego aktualnemu wyglądowi, itp. To wysyłać musimy. I wysyłamy - przy każdej komunikacji z serwerem wysyłamy szczegółowe informacje o naszej postaci i odbieramy dane o pozostałych.

Apteczki, amunicja, broń? Tutaj na szczęście nie ma wiele problemów. Wystarczy przesłać informację o danym wydarzeniu (np. właśnie zebraniu KONKRETNEJ apteczki) i pozostałe programy już powinny sobie z tym poradzić. To samo dotyczy czata obecnego w grze. Po

prostu komunikat (jakaś wypowiedź) jest wysyłany do pozostałych.

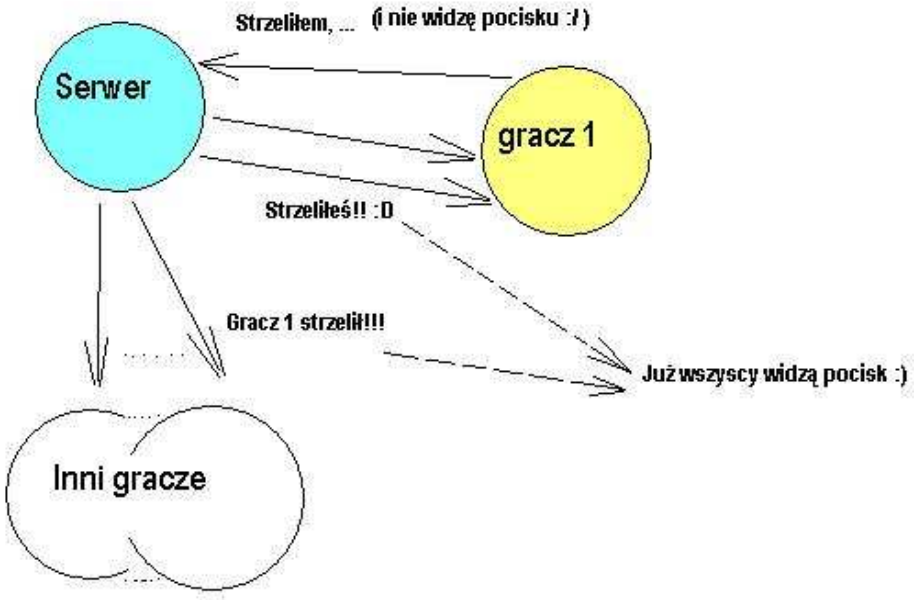
Co jednak zrobić z pociskami? Przecież one nie znikają od razu z planszy, tylko pozostają na niej przez dobre kilkadziesiąt klatek. Kto ma nimi zarządzać? Czy może serwer co pewien czas rozsyłać dane o aktualnym położeniu pocisku? A może gracz, który go wystrzelił? Obie te opcje okazują się być zupełnie błędne. Przy dużej liczbie graczy i szybkostrzelności broni, na planszy w jednej chwili może być nawet kilkaset pocisków. Wysyłanie tak ogromnych porcji danych za każdym razem jest zwykłą niemożliwością. Ale przecież wystarczy tylko raz "zdefiniować" pocisk - tuż po wystrzale! Dać mu informacje o "właścicielu" (graczowi, który strzelał), położeniu początkowemu, kierunkowi i zwrotowi oraz w jakiś sposób określić jego prędkość oraz siłę i zasięg rażenia. To ostatnie w dość sprytny sposób zmniejszamy do malutkiej informacji: z jakiej broni wystrzelono ten pocisk. I to w zupełności wystarcza. Wyliczenie pozycji pocisku w następnej klatce i zbadanie jego kolizji jest dużo szybsze niż przesyłanie całego "stanu" za każdym razem!

Dodatkowo wysyłanie informacji o np. zebraniu apteczki, wiadomości na czacie czy zebraniu broni jest napisane w "obiektowy sposób". Stworzyliśmy bardzo wygodny system przesyłania do serwera tak zwanych "wydarzeń", aby nie trzeba było za każdym razem, gdy tylko zechcemy coś do gry dodać lub jakiś szczegół w niej zmienić, przepisywać żmudnie dużych fragmentów kodu.

To wszystko brzmi ładnie i przyjemnie, ale cały czas pozostaje jeden problem. Skoro komunikacja jest asynchroniczna, to jak to się dzieje, że gra się nie rozspójnia?? Że na wszystkich komputerach graczy w tym samym momencie są oddawane strzały i że u jednego wynik nie wskazuje np. na 7 fragów, podczas gdy inni widzą ich tylko 6??

5.6. Decyduj sam za siebie i nigdy się nie spiesz.

Już wiemy co i jak często mamy wysyłać. Więc zróbmy jeszcze jedną rzecz. Clue programu, które pozwoliło nam zachować spójność rozgrywki bez uszczerbku dla dynamiki i innych cech, na których nam zależało. Skoro i tak komunikujemy się z serwerem, i tak odbieramy od niego informacje o działaniach przeciwników, to może odbierzmy też informacje o nas!! Może na pierwszy rzut oka wydaje się to dziwne czy wręcz paradoksalne, ale bynajmniej tak nie jest!! Strzelamy? No to prześlijmy informację na serwer i dopiero gdy ten "zatwierdzi" naszą akcję, umieśćmy ją w świecie Flatline. Chcemy zebrać apteczkę? Nic prostszego - prośba do serwera, a ten na pewno się zgodzi. A ta chwila oczekiwania w ogóle nie wpływa na płynność rozgrywki! Zyski zaś, jak widać, są niebagatelne.



A co się dzieje z naliczaniem fragów? No cóż - pociski mają przecież swoich "właścicieli". Po co więc badać kolizje na każdym komputerze z osobna i liczyć na brak "rozspójnienia"? My po prostu ufamy graczom! To każdy zawodnik sam stwierdza, czy i przez kogo został zabity i rozgłasza tę informację "w świat"!! A jako autorzy programy mamy pełne prawo wierzyć, że nikt nas nie oszuka :)

5.7. Krótkie podsumowanie.

Kawa na ławę, czyli w jakiej kolejności wykonujemy opisane powyżej czynności.

1. Najpierw serwer wysyła do każdego z graczy przynależny mu punkt startowy.
2. Serwer przechodzi w stan "nasłuchu"
3. Każdy z graczy wysyła do serwera informacje o swoim położeniu i o wystrzelonych pociskach
4. W każdej turze gracz sprawdza (w sposób nieblokujący) czy uzyskał jakieś dane od serwera. Jeśli tak, to je odbiera. Jeśli nie, to gra dalej (i wszelkie informacje o wystrzelonych pociskach, zebranych broniach czy apteczkach są przechowywane, ALE NIE SĄ UWZGLĘDNIANE NA PLANSZY).
5. Serwer odebrał już dane od wszystkich graczy i zaczyna je rozsyłać (każdy dostaje też informacje, że wykonane przez niego czynności są już PEŁNOPRAWNE).
6. Gdy gracz może odebrać już dane od serwera, po prostu to robi. I od razu w tej samej klatce wysyła informację o sobie (wcześniej skumulowane dane i aktualne położenie).
7. Wracamy do nasłuchiwania przez serwer (aby odebrać dane od graczy) i dalszego wysyłania...

Rozdział 6

Mój wkład w przedsięwzięcie

”
- I naprawdę pragniesz zostać strażakiem, drogi Panie Wojtku?
- Ależ bynajmniej. Tak mnie się bez przypadek jeno powiedziało... ”

„O głupotach słów kilka w przekładzie cudacznym” Orjeh Ceicjow

Projekt „Flatline” powstawał w ramach ZESPOŁOWEGO Projektu programistycznego, więc duża cześć naszej pracy była wynikiem grupowych przemyśleń i burzy mózgów. Musze jednak zaznaczyć, ze od pewnego momentu, przynajmniej w aspekcie kodowania wymyślonych wcześniej rozwiązań, wkradła się w naszych działaniach spora modularyzacja.

W projekcie zajmowałem się grafiką komputerową. Opracowywałem obiekty 3D, tekstury, zachowania i wkomponowywałem w mapę gry. W przypadku obiektów ruchomych dołączałem też animacje ruchu. Dla map z naszej gry opracowałem własny format zapisu.

Tworzenie projektu takiego jak gra komputerowa wymagało wielu godzin testowania aplikacji. Sprawdzania istniejących algorytmów w warunkach gry. Szukanie, a to nowych pomysłów, a to sposobów na ulepszenie gotowej już aplikacji. Inspiracje dla rozwijania naszej gry czerpaliśmy z... samej gry w nią, na Początku na komputerach wydziałowych potem również przez internet.
Praca w zespole jako grafik pozwoliła mi poznać wiele ciekawych aplikacji do grafiki 3D.

Przy tworzeniu dokumentacji gry współpracowałem przy powstawaniu dokumentu BUC (Business Use Case) oraz pracy licencjackiej. Utworzyłem dla projektu również stronę internetową.

Rozdział 7

Opis zawartości dołączonej płyty CD

doc Katalog z dokumentacją.

- **licencjat.pdf** Praca licencjacka.

src Katalog ze źródłem aplikacji *Flatline*.

- **flatline.zip** Źródło programu klienckiego *Flatline*.
- **serwer.zip** Źródło serwera *Flatline*.

Flatline.exe Program instalacyjny aplikacji *Flatline*.

Bibliografia

[Sieci] Anthony Jones, Jim Ohlund *Programowanie sieciowe Microsoft Windows*

[Algo] Thomas H. cormen, Charles E. Leiserson, Ronald L. Rivest *Wprowadzenie do algorytmów*

[C++] Bjarne Stroustruop *Język C++*

[Graf] Richard S. Wright jr, Michael Sweet *OpenGL. Księga eksperta*