Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Grafika Komputerowa

Dokumentacja projektowa

Gra 3D oparta na własnych modułach silnika.

Grupa: GKiO1
Mateusz Przybyłek
Adrian Skupień

Prowadzący:
Mariusz Szynalik

Gliwice 2011

1. Treść zadania

Celem naszego zadania będzie stworzenie niżej wymienionych elementarnych mechanizmów niezbędnych do stworzenia gry komputerowej. Zdecydowaliśmy się na taką interpretację:

- uproszczony parser plików XML bądź SQLLite do szybkiego i prostego dostępu do danych,
- własny silnik graficzny (oparty o bibliotekę OpenGL) lub Ogre3D – jeszcze nie zdecydowaliśmy się na wybór,
- moduł zarządzania zasobami (obiektami i elementami) w grze,
- moduł obsługi wejścia (klawiatura, mysz),
- moduł obsługi dźwięku.

Przy implementacji zamierzamy stosować środowisko deweloperskie Microsoft Visual Studio 2010, a jako język zamierzamy używać C++ z uwagi na szerokie zastosowanie w produkcji gier.

Prezentacja mechanizmów

Poprawna praca wyżej wymienionych modułów zostanie zaprezentowana w postaci prostej gry trójwymiarowej. Na razie nie podjęliśmy decyzji co do funkcjonalności gry będzie to zależało od stopnia rozwoju silnika.

2. Analiza zadania:

Elementy związane z przedmiotem:

- grafika 3D (biblioteka OpenGL)
- przekształcenia afiniczne
- proste mechanizmy animacji
- eliminacja niewidocznych powierzchni przy pomocy kontrolowania mechanizmu bufora Z udostępnianego przez standardowa implementacje OpenGL
- wykrywanie kolizji
- wykorzystujemy shadery

Użyte algorytmy:

- detekcja kolizji przy pomocy przybliżenia obiektów do sfer

Użyte wzorce projektowe:

- singleton
- interfejs
- wrapper danych

Specyfika izolacji mechanizmów:

- mechanizm graficzny odczytuje dane wprost z mechanizmu logiki i przerzuca niektóre elementy do swojego własnego formatu danych przy pomocy wrapperow

- mechanizm logiki nie jest świadomy istnienia mechanizmu grafiki, posiada za to odpowiednie flagi używane do kontroli sterowania silnika graficznego

Kamienie milowe:

1. Struktury danych.
2. Zaprojektowanie mechaniki grafiki oraz logiki.
3. Pierwsze obiekty.
4. Implementacja uproszczonej mechaniki oraz grafiki w trybie wyświetlania ortogonalnego (2D).
5. Umożliwienie gry.
6. Zaimplementowanie obsługi zderzeń.
7. Przejdzie z widoku ortogonalnego na widok perspektywiczny (3D).
8. Zakończenie testów.

Zmiany w porównaniu z dokumentem projektowym:

Implementacja własnego silnika graficznego napisanego przy pomocy API OpenGla i bibliotek pomocniczych.

Zaimplementowaliśmy także wewnątrz silnika moduły zarzadzania zasobami, pozwalający na efektywne kasowania niepotrzebnych kontenerów zawierających struktury graficzne OpenGLa.

Przy obsłudze urządzeń wejścia, zdecydowaliśmy się na zaimplementowanie zdarzeń klawiatury, jako że były one dla nas niezbędne do sterowania postacią.

Zdecydowaliśmy się na rezygnacje z parsera XMLa (czy też lokalnej bazy danych SQLLite) ze względu na to iż wybrany przez nas typ gry nie wymaga przechowywania dużej ilości danych, a wszelkie potrzebne informacje generowane są podczas działania programu.

System audio nie został zaimplementowany głownie ze względu na brak czasu, oraz małą wiedzę na dany temat, nie bez wątpienia pozostaje fakt iż taki moduł nie był dla nas sprawą priorytetową.

3. Specyfikacja zewnętrzna

Gra polega na likwidowaniu pojawiających się losowo przeciwników i zdobyciu jak największej ilości punktów.

Grę rozpoczynamy starując na środku kwadratowej planszy, na której krańcach w losowych miejscach (daleko od przeciwnika) pojawiają się przeciwnicy. Na starcie posiada się 5 żyć.

Postacią sterujemy za pomocą strzałek, praco, lewo, jest skręcanie, przód, tył, odpowiednio przesuwamy postać do przodu oraz do tyłu.

Strzał oddajemy naciskając klawisz spacji, możemy oddać maksymalnie 10 strzałów na sekundę.

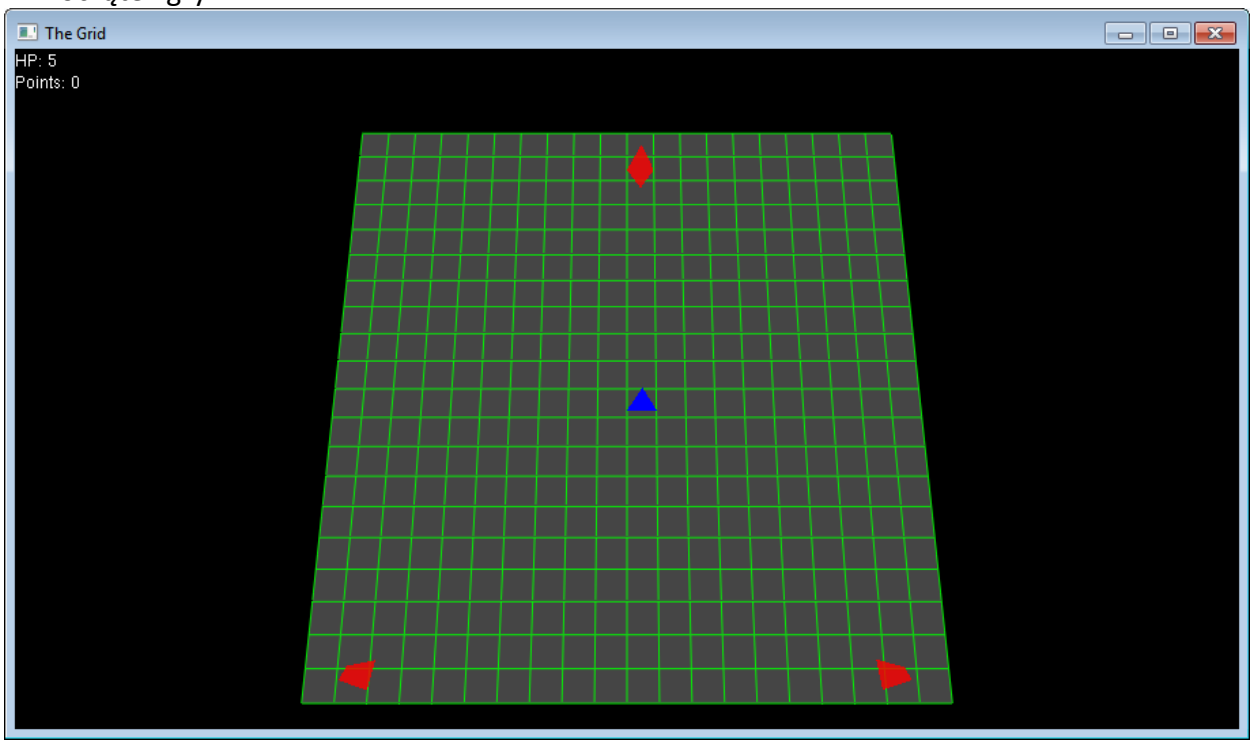
W górnym lewym roku ekranu mam informacje o posiadanej ilości życia oraz ilości zebranych punktów. Za każdego zabitego przeciwnika dostajemy 1 punkt.

Jeśli przeciwnik podejście zbyt blisko do nas, wybucha i zabiera nam jedno życie. Jeśli poziom życia zejdzie do zera gra się zakończy.

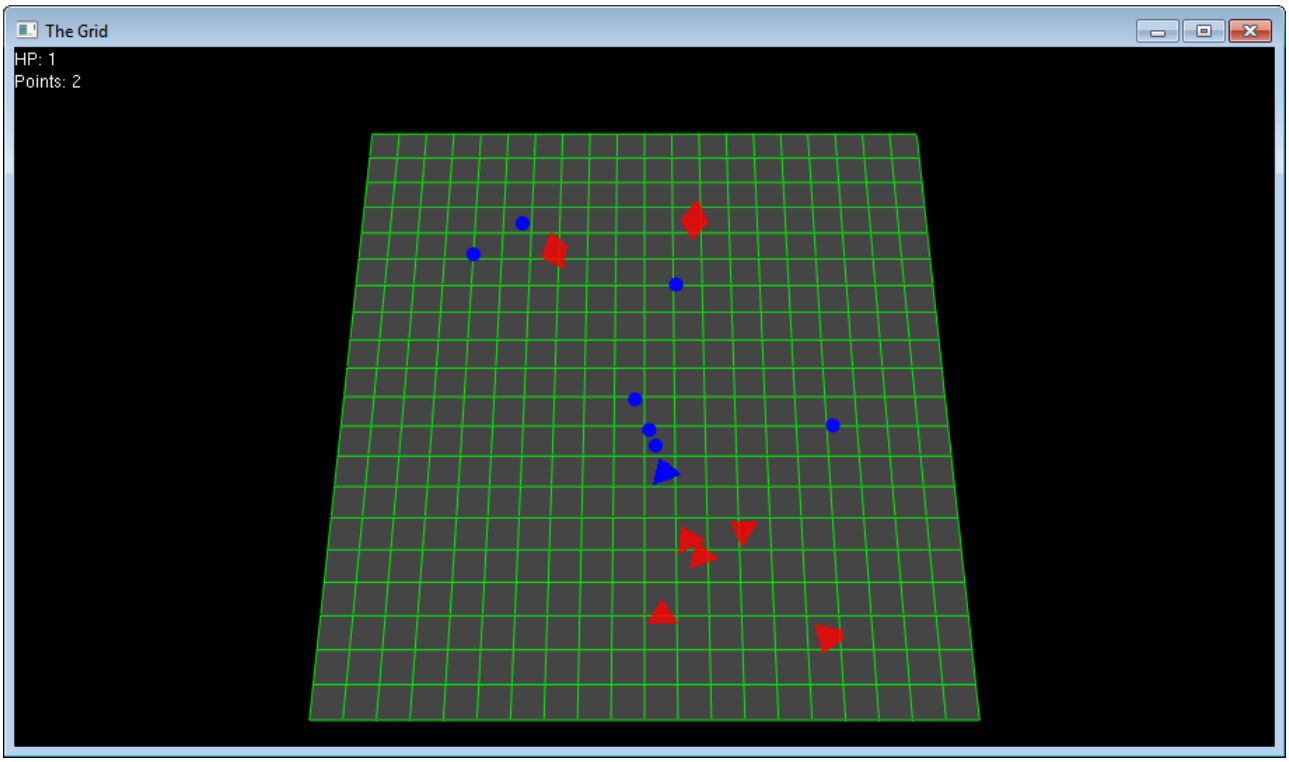
Podczas gry możemy uruchomić wyświetlanie informacji o ilości fpsów klawiszem ‘F’.

4. Przykłady z działania gry:

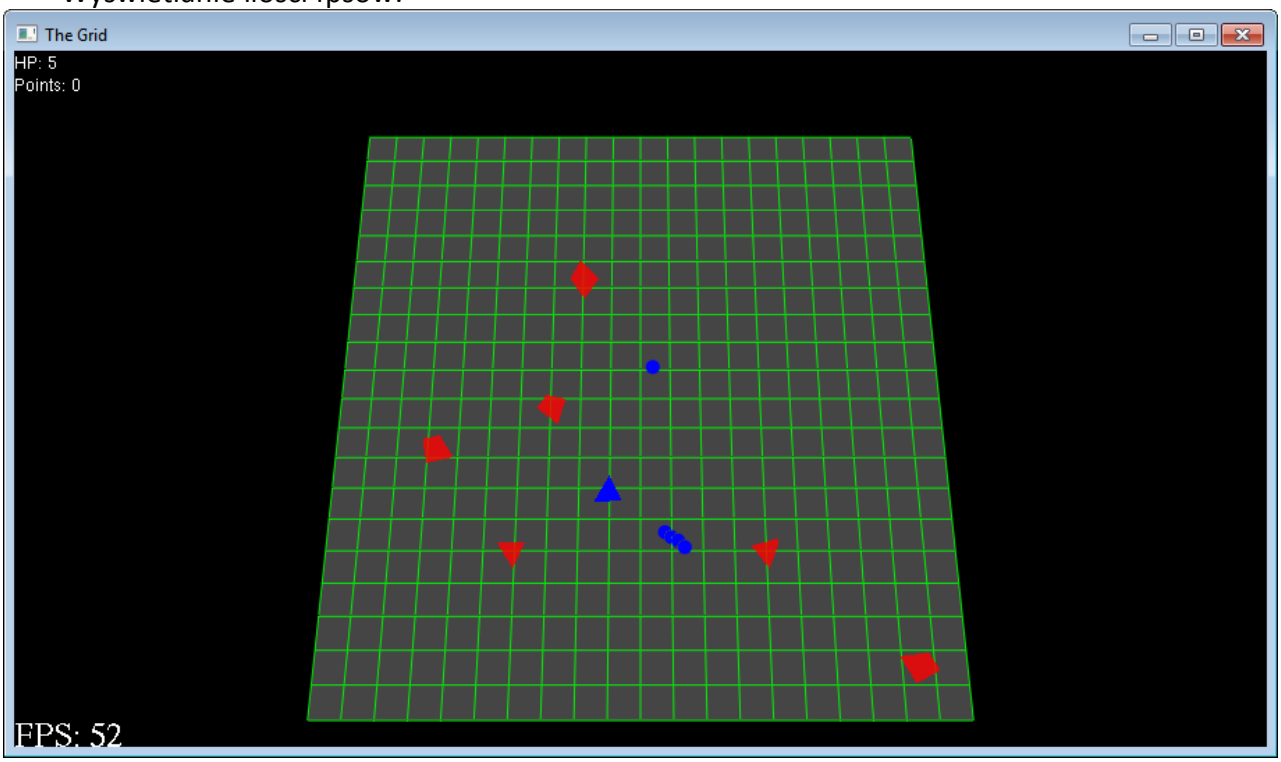
Początek gry:



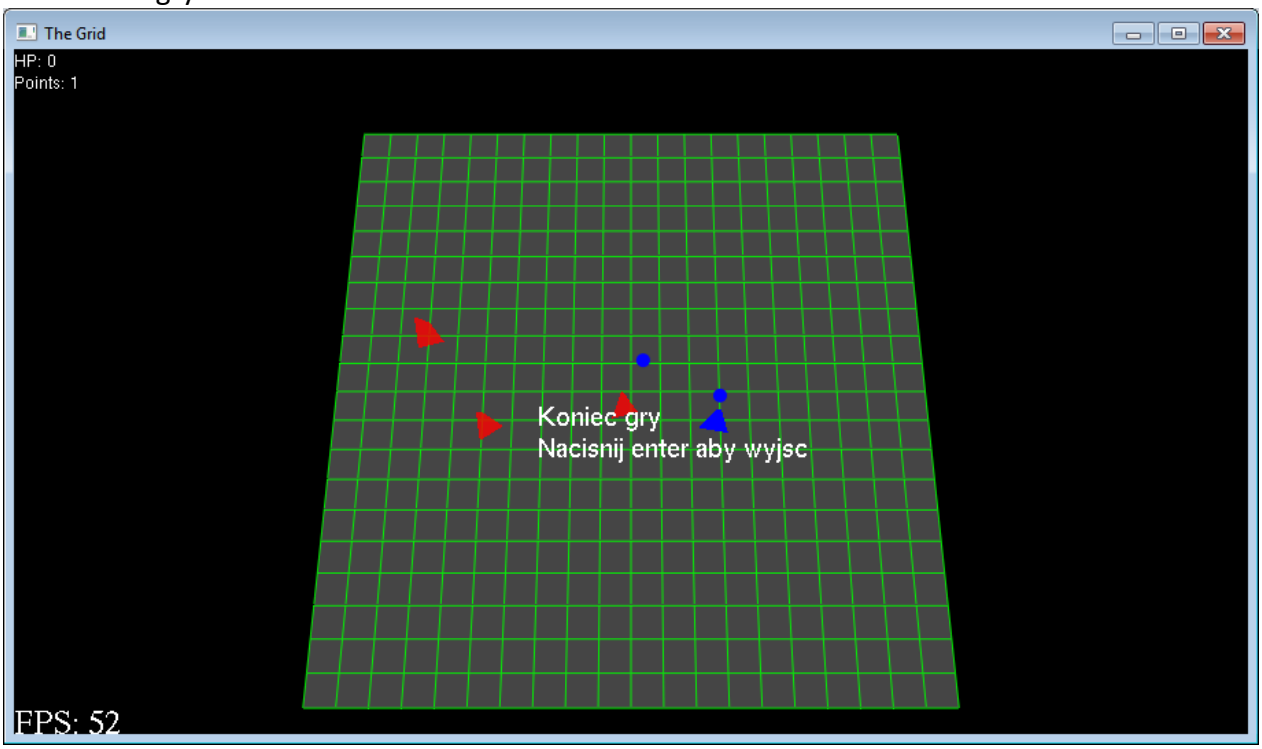
Strzelanie:



Wyświetlanie ilości fpsów:



Koniec gry:

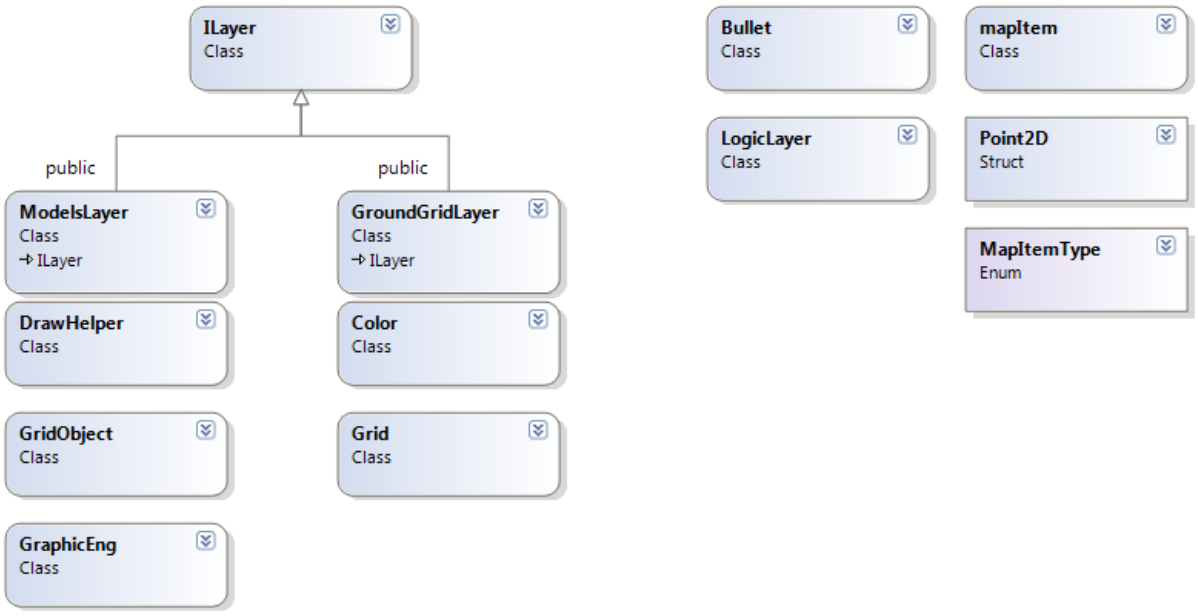


5. Specyfikacja wewnętrza:

Gra została zaimplementowania z wykorzystaniem pętli silnika gry, z czego, jako że mamy 2 odrębne systemy zarządzania, logiką oraz grafiką (logika całkowicie niezależna od grafiki), ustawiliśmy odświeżanie pętli grafiki na 50fpsów (zmiany niewidoczne dla oka), natomiast pętla logiki działa z częstotliwością 10Hz.

Szczegółową specyfikacje wewnętrzną plików klas, metod oraz parametrów, można znaleźć w katalogu „specyfikacja_wewnetrzna” w folderze z dokumentem.

Schemat klas:



6. Testowanie i uruchamianie:

W ramach testów (wykonywanych na bieżąco w przypadku testowania nowej funkcjonalności) próbowaliśmy wykonywać przypadki dla krytycznych sytuacji (jak np. wychodzenie poza plansze, zderzenia). Przykładem wykrytych problemów był błąd zarządcy danych w momencie 'zabicia' przeciwnika przy pomocy "ostatniego" z pocisków w tablicy, gra wyrzucała bardzo nieprzyjemny wyjątek, a jako że działo się to niejako losowo, problemem było odpowiednie do zdebudowanie. Dodatkowo, jako że pracowaliśmy w dwuosobowym zespole, w celu przetestowania gry często korzystaliśmy z techniki cross-testing, która pozwoliła wyeliminować drobne błędy w silniku graficznym oraz w systemie lotu pocisków.

7. Wnioski:

Implementacja własnego silnika gry przy pomocy API OpenGLa oraz niskopoziomowych funkcji pomocniczych pozwoliła nam posiąść wiedze na temat bardzo pożądanej przez pracodawców biblioteki programistycznej.

Proces debugowania aplikacji działającej na powtarzalnej pętli np. gry) jest znacznie bardziej trudniejszy niż aplikacji zdarzeniowych (eventowych, jak winapi), ze względu na swoistą losowość różnych sytuacji.