POLITECHNIKA ŚLĄSKA

Wydział Elektryczny

# Tanks 3D FPP

## Gra turowa napisana w XNA 4.0

*Kierunek, semestr, grupa dziekańska, sekcja:*

**Informatyka, sem. VII, gr. A1, s. 4**

*Przedmiot:*

**Grafika Ruchoma**

*Prowadzący zajęcia:*

**Dr inż. Artur Pasierbek, Dr inż. Marcin Połomski**

*Zespół:*

**Korneliusz Caputa (Project Manager)**

**Piotr Matla**

**Witold Wąsowicz**

**Gliwice, styczeń 2013**

Niniejsza praca powstała w trakcie i jest rezultatem przedmiotu Grafika Ruchoma prowadzonego w Politechnice Śląskiej przez Dr inż. Artura Pasierbka oraz Dr inż. Marcina Połomskiego. Opracowanie to w całości ani w części nie może być wykorzystywane do jakichkolwiek celów bez uprzedniej zgody autorów bądź odpowiednich władz Politechniki Śląskiej.

Copyright ©2013 Politechnika Śląska w Gliwicach

Spis treści

[Tanks 3D FPP 1](#_Toc345969342)

[Gra turowa napisana w XNA 4.0 1](#_Toc345969343)

[1. Sformułowanie zadania projektowego 3](#_Toc345969344)

[1.1. Cel projektu 3](#_Toc345969345)

[2. Wymagania 4](#_Toc345969346)

[2.1. Wymagania funkcjonalne – gra z punktu widzenia użytkownika 4](#_Toc345969347)

[2.1.1. Menu 4](#_Toc345969348)

[2.1.2. Losowo generowana mapa 4](#_Toc345969349)

[2.1.3. Tura gracza 4](#_Toc345969350)

[2.2. Wymagania pozafunkcjonalne 4](#_Toc345969351)

[2.2.1. Rozszerzalność 4](#_Toc345969352)

[2.2.2. Wydajność 4](#_Toc345969353)

[3. Realizacja menu 5](#_Toc345969354)

[4. Realizacja pola bitwy 5](#_Toc345969355)

[5. Realizacja walki turowej 5](#_Toc345969356)

[5.1. Kalibracja modeli czołgu i pocisku 5](#_Toc345969357)

[5.2. Detekcja kolizji modeli między sobą oraz kompatybilność z terenem 5](#_Toc345969358)

[5.2.1. Kolizje między modelami 5](#_Toc345969359)

[5.2.2. Kolizja modeli z terenem 6](#_Toc345969360)

[5.3. Mechanika tury i listy wyników 6](#_Toc345969361)

[5.4. Kamera w trakcie wystrzału 6](#_Toc345969362)

[5.5. Wnioski z realizacji 7](#_Toc345969363)

# Sformułowanie zadania projektowego

## Cel projektu

Celem projektu jest.

# Wymagania

## Wymagania funkcjonalne – gra z punktu widzenia użytkownika

### Menu

* ???

### Losowo generowana mapa

* ???

### Tura gracza

* Jeden gracz kontroluje jeden czołg i ma do dyspozycji następujące możliwości:
  + nastawianie kąta obrotu wieżyczki oraz nachylenia działa
  + nastawianie siły wystrzału
  + wystrzelenie pocisku z zadanymi wcześniej parametrami
* W trakcie tury gracz może rozglądać się za pomocą:
  + strzałek – kamera z wieżyczki
  + myszki – po wystrzale swobodnie z punktu widzenia pozycji lecącego pocisku
* HUD informuje gracza o:
  + kącie obrotu wieżyczki
  + kącie nachylenia działa
  + nastawionej sile wylotowej pocisku
  + czyja tura odbywa się w danej chwili
  + statusie życia poszczególnych graczy
* Tura kończy się po kolizji pocisku z innym czołgiem lub terenem
* Rozgrywka kończy się, gdy na polu bitwy pozostanie tylko jeden czołg; wyświetla się wówczas:
  + lista graczy wg kolejności wyników
  + komunikat iż można zacząć kolejną rozgrywkę na nowo wygenerowanym terenie lub zakończyć grę

## Wymagania pozafunkcjonalne

### Rozszerzalność

Architektura aplikacji ma pozwalać na dodawanie nowych funkcjonalności do gry. Może zostać rozbudowana o ruch czołgów, różnorodność czołgów i amunicji. Przygotowana jest również na różne traktowanie trafienia w różne partie czołgu.

???

### Wydajność

??? coś o mapie?

# Realizacja menu

# Realizacja pola bitwy

# Realizacja walki turowej

## Kalibracja modeli czołgu i pocisku

Na początku została zaimplementowana klasa Tank, której zadaniem było radzenie sobie z obsługą modelu czołgu. Bardzo podobna do tej która realizowana była na zajęciach laboratoryjnych, chociaż stworzona została przed nimi. Zaopatrzona dodatkowo w zmienną Scale, przez którą model jest przeskalowywany. Wprowadzono ją, aby model mógł być dopasowywany do mapy, gdyby ta okazała się być trudna do powiększenia. Obecnie czołgi mają 10% swojego oryginalnego rozmiaru.

Klassa TankMissile realizuję zmianę położenia, detekcję kolizji oraz obsługę modelu pocisku. Poświęciliśmy zasoby, aby pocisk leciał parabolicznie – to było dobre posunięcie oczywiście. Poświeciliśmy również zasoby, aby pocisk ten obracał się przodem do kierunku lotu – to też było dobre, ale w końcowym efekcie okazało się absolutnie zbędne. Co z tego iż pocisk leci ładnie i mniejwięcej zgodnie z prawami fizyki, kiedy gracz chce przeprowadzić w locie zwiad terenu, a nie zachwycać się modelem pocisku. Nie byłby to jeszcze taki straszny strzał w stopę, gdyby nie to, że chcąc ułatwić sobie sprawę ustawiliśmy pivot modelu na czubku z przodu pocisku. To wymusiło na nas dodatkowe przekształcenia rotacji, które później odbiły się na nas zarówno przy ustalaniu pozycji wylotowej pocisku z lufy jak i przyczepianiu kamery do pozycji pocisku.

## Detekcja kolizji modeli między sobą oraz kompatybilność z terenem

### Kolizje między modelami

Kolizje między modelami zostały zrealizowane przy pomocy listy obiektów typu BoundingSphere. Podczas lotu pocisku kolizja z modelem czołgu sprawdzana jest co przesunięcie o zadany wektor prędkości.

W przypadku gdy BoundingSphere modelu pocisku zachodzi na BoundingSphere modelu czołgu wykrywana jest kolizja. W następstwie odejmowane są punkty życia i następuje zmiana aktualnego gracza na następnego w kolejce.

Ponieważ każdy mesh czołgu ma własny BoundingSphere, bez problemu można określić w jaką partię model czołgu dostał pociskiem. W międzyczasie zrodził się pomysł by zaimplementować jako dodatkowy smaczek „niszczenie” poszczególnych części modelu, przy czym pomimo iż zabrakło na to zasobów to w przyszłości projekt może być rozwinięty o taką funkcjonalność.

### Kolizja modeli z terenem

Klasa abstrakcyjna CollidingEntity została wprowadzona by powiązać wszystkie klasy reprezentujące jakieś byty z mapą wysokościową. Wymusza ona implementację dwóch metod:

* bool IsInFloorBounds(IHeightMap floor, Vector3 position);

Sprawdza czy wskazana pozycja nie wykracza po za mapę wysokościową. Pozwala to na uniknięcie niechcianych wyjątków odwołania do nieistniejących punktów w tablicy mapy wysokościowej.

* Vector3 OffsetToFloorHeight(IHeightMap floor, Vector3 position);

Zwraca podaną pozycję ze zmodyfikowaną wartością Y, aby sprowadzić pozycję „na ziemię”. Pozwala to zarówno na położenie czołgu na powierzchni mapy wysokościowej jak i sprawdzenie czy pocisk nie zleciał już dość nisko.

## Mechanika tury i listy wyników

Klasa TanksController odpowiada za realizację rozgrywki. Zawiera ona listę czołgów, zarządza modelami, zmianą tury i częściowo także odpowiada za obsługę klawiatury. Przechowuje i zarządza również częścią efektów dźwiękowych takich jak wybuchy i odgłosy ruchu.

W swojej turze gracz może modyfikować kąt obrotu wieżyczki, kąt nachylenia działa oraz początkową siłę wylotową pocisku z lufy. Klasa Tank zajmuje się stosownymi przekształceniami transformacji odpowiednich macierzy modelu czołgu.

Zmienna TurnToken zawiera indeks do listy czołgów graczy, który świadczy o tym czyja aktualnie jest tura. TurnToken inkrementowany jest za każdym razem, gdy nastąpi kolizja wystrzelonego pocisku z terenem lub modelem czołgu.

Lista wyników przechowująca kolejne czołgi które poległy jest wypisywana na ekran od końca, gdy na polu bitwy pozostaje tylko jeden czołg.

## Kamera w trakcie wystrzału

Gdy stworzyliśmy kamerę widoku z pierwszej osoby FPP sądziliśmy, że to już krótka droga do powiązana jej z pozycją lufy i kierunku w który ona „patrzy”. Niestety okazało się, że przez to iż chcieliśmy sobie wcześniej ułatwić (tak jak wspomniano w podrozdziale 5.1. pivot modelu został przypisany na czubku pocisku) trzeba było uwzględnić obrót modelu pocisku, oraz w którą stronę jest on skierowany. Na szczęście po uprzednich doświadczeniach nie przysporzyło to wiele problemów.

* public void AttachAndUpdate(Tank tank);

W trakcie gdy kontrolowany jest czołg pozycja kamery otrzymuje współrzędne pozycji początku lufy. Zarówno kierunek patrzenia jak i wektor UP obliczane są na podstawie kąta obrotu kamery, który realizowany jest na podstawie obrotu wieżyczki i działa danego czołgu.

* public void AttachAndUpdate(Vector3 missilePos);

Kiedy pocisk zostanie wystrzelony, pozycja kamery przenoszona jest w miejsce pocisku i gracz może swobodnie rozglądać się dookoła przy pomocy myszki i tym samym zapoznawać się z terenem.

## Wnioski z realizacji

blablablablablabla