

Grafika Komputerowa						
Rok akademicki	Termin	Rodzaj studiów	Kierunek	Prowadzący	Grupa	Sekcja
2014/2015	Wtorek	SSI	INF	dr Ewa Lach	GKiO3	1
	12:45 - 15:00					



Karta projektu

Danmaku Shooter

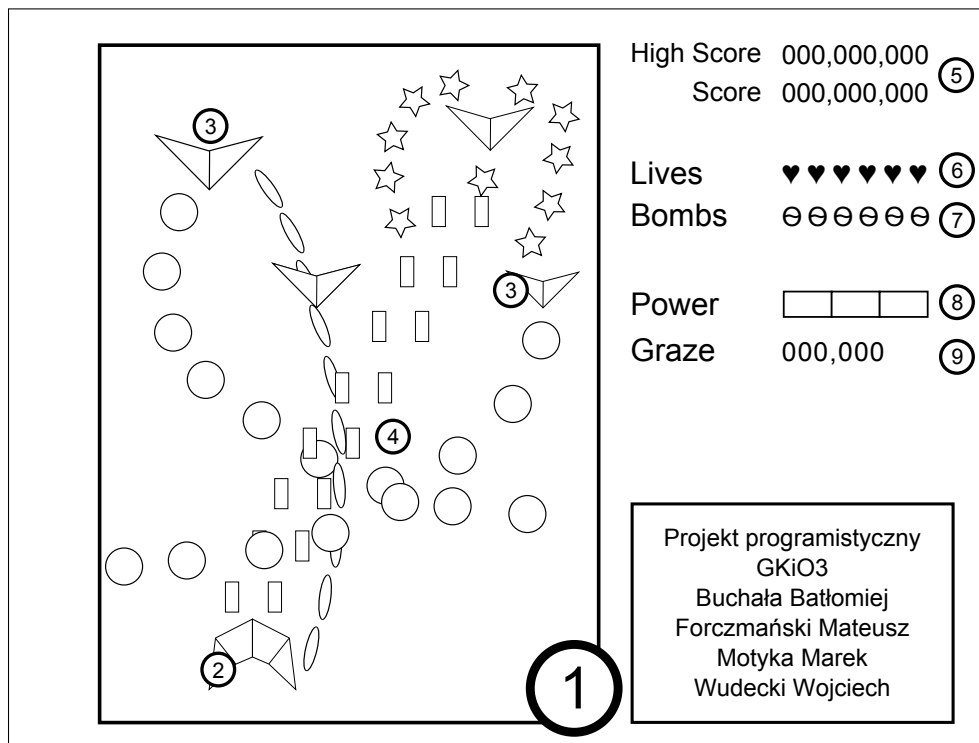
Skład sekcji:

Buchała	Bartłomiej
Forczmański	Mateusz
Motyka	Marek
Wudecki	Wojciech

Krótki opis aplikacji

Shoot' em up (w skrócie zwany shmup) jest gatunkiem gier akcji wywodzącym się w prostej linii od gier typu *Space Invaders* lub *River Raid*. Kontrolowana przez gracza postać (np. statek) w pojedynkę stawia czoło przeciwnikom, niszcząc ich za pomocą wystrzeliwanych pocisków, jednocześnie unikając ich ataków. Podgatunek shmupów, zwany *danmaku* (z jap. *ściana pocisków* lub *piekło pocisków*) kładzie większy nacisk na omijanie wrogich ataków, niż na ofensywie. Przykładowymi danmaku są np. *Ikaruga* czy większość gier z uniwersum *Touhou Project*.

Gra powstawać będzie jako projekt łączony z przedmiotów Projekt Programistyczny i Grafika Komputerowa.



1. Ekran gry właściwej. W jej obrębie znajduje się gracz, pociski oraz wszyscy wrogowie.
2. Grywalna postać, porusza się po ekranie gry, unikając pocisków oraz strzelając do wrogów.
3. Wrogowie, których należy pokonać.
4. Chmara pocisków. Jak widać na rysunku, nie wchodzi ze sobą w żadną interakcję, każdy leci swoim wyznaczonym torem. Sprajty wrogów są niewrażliwe na swoje pociski, nie występuje *friendly fire*.
5. Liczba zdobytych punktów oraz porównywanie ich z największym wynikiem.
6. Liczba pozostałych żyć. W trakcie gry można zdobywać kolejne. Utrata wszystkich kończy grę.
7. Liczba pozostałych bomb. Każda wykorzystana bomba zapewnia kilkusekundową odporność na wrogie pociski oraz umożliwia pojedynczy silniejszy atak. Można je zdobyć w trakcie gry.
8. Pasek mocy, napęlnia się w trakcie gry wraz ze zdobytymi punktami. Utrata życia skutkuje zmniejszeniem paska o 1 segment.
9. Liczba "otarć", czyli uniknięć bardzo blisko pociskowi. Aby umożliwić większe wyzwanie, ostateczny wynik pomnożony jest przez licznik Graze.

Analiza zadania

1. Podstawy teoretyczne problemu

1.1. Przestrzeń fizyczna

W naszej grze w interakcję ze sobą będzie wchodzić bardzo dużo elementów, począwszy od gracza, a na ostatnim z setki pocisków skończywszy. Bardzo ważne jest, aby ruchy i interakcje wszystkich elementów były ze sobą zsynchronizowane, a droga o długości 1 była tym samym dla każdego z nich.

W tym celu zdecydowaliśmy się na wprowadzenie do gry przestrzeni fizycznej, w której będą zachodzić wybrane przez nas zjawiska fizyczne, potrzebne dla realizacji gry.

1.2. Ruch

Każdy obiekt, który może się poruszać i zmienić swoje położenie, posiada swoją prędkość. W naszej grze wyróżniliśmy 3 rodzaje ruchu:

1.2.1. Ruch jednostajny

W jednostce czasu ciało pokonuje jednakową drogę, a przebyta droga jest proporcjonalna do czasu.

$$v = \frac{s}{t} = \text{const}$$

Gdzie v to prędkość, s - droga, a t to czas. W naszym obiekty poruszają się na ekranie, więc jednostką długości jest piksel.

1.2.2. Ruch jednostajnie przyspieszony

W jednostce czasu prędkość ciała ulega zwiększeniu o stałą wartość.

$$v(t) = v_0 + a \cdot t$$

1.2.3. Ruch jednostajnie opóźniony

W jednostce czasu prędkość ciała ulega pomniejszeniu o stałą wartość.

$$v(t) = v_0 - a \cdot t$$

1.3. Tor pocisków

Jednym z podstawowych problemów w naszym zadaniu jest tor po jakim poruszają się pociski. Obiekty wrogów będą poruszać się w prosty sposób, jednak zbiory pocisków będą układać się w skomplikowane wzory, tworzyć ze sobą specyficzny układ. W naszej grze chcemy zaimplementować pociski, które będą poruszać się m.in. po takich torach jak okrąg i elipsa.

1.3.1. Okrąg

Wybraliśmy opis w postaci równania parametrycznego, który jest wygodniejszy i bardziej efektywny w implementacji:

$$\begin{cases} x = x_0 + r \cdot \cos \alpha \\ y = y_0 + r \cdot \sin \alpha \end{cases}$$

Gdzie punkt $O(x_0, y_0)$ jest środkiem okręgu, r promieniem, a parametr $\alpha \in [0, 2\pi)$.

1.3.2. Elipsa

Podobnie jak wyżej, elipsę zdefiniowaliśmy równaniem parametrycznym:

$$\begin{cases} x = x_0 + a \cdot \cos \alpha \\ y = y_0 + b \cdot \sin \alpha \end{cases}$$

Gdzie punkt $O(x_0, y_0)$ jest środkiem elipsy, a , b długościami półosi, a parametr $\alpha \in [0, 2\pi)$.

1.3.3. Spirala

Tor spirali jest zdefiniowany równaniem parametrycznym:

$$\begin{cases} x = x_0 + a \cdot \alpha \cos \alpha \\ y = y_0 + b \cdot \alpha \sin \alpha \end{cases}$$

Gdzie punkt $O(x_0, y_0)$ jest punktem centralnym spirali, a , b długościami półosi, a parametr $\alpha \in [0, \infty)$.

2. Wykorzystywane zagadnienia grafiki komputerowej

2.1. Przekształcenia afiniczne

Postanowiliśmy wykorzystać na poznane na laboratorium przekształcenia afiniczne. Wszystkie ruchome obiekty w naszej grze, które zmieniają dynamicznie swoją pozycję, będą korzystały z trzech wybranych przekształceń:

2.1.1. Translacja

Przesunięcie punktu $P = (x, y, z)$ o wektor $d = [d_x, d_y, d_z]$, w wyniku którego powstaje obraz $P = (x', y', z')$, gdzie:

$$\begin{cases} x' = x + d_x \\ y' = y + d_y \\ z' = z + d_z \end{cases}$$

2.2. Skalowanie

W naszym programie wszystkie przekształcenia skalowania będą zachowywać proporcje, więc zdefiniować je jako: przesunięcie punktu $P = (x, y, z)$ o współczynnik skalowania s , w wyniku którego powstaje obraz $P = (x', y', z')$, gdzie:

$$\begin{cases} x' = s \cdot x \\ y' = s \cdot y \\ z' = s \cdot z \end{cases}$$

2.3. Obrót

3. Wykorzystywane biblioteki i narzędzia programistyczne

3.1. DirectX 9

Jako narzędzie do budowania obiektów graficznych zdecydowaliśmy się na DirectX w wersji 9. Jako alternatywne rozwiązanie rozważaliśmy DirectX 11, jednak nasza gra będzie budowana na scenie 2D, a DirectX 9 oferuje wygodniejsze narzędzia - wciąż operuje na takich klasach jak Sprite lub Texture, które są lepsze dla naszej gry. DirectX 11 wszystkie te klasy zastępuje interfejsem IResource, który wymaga odpowiedniej konwersji (większe nastawienie na grafikę trójwymiarową). By móc wykorzystać DirectX 11 do pracy na zwykłych sprajtach, wymagane byłyby dodatkowe zestawy narzędzi, jak np. DirectX Tool Kit.

Jako konkurencyjne rozwiązanie dla samego Directxa rozważaliśmy OpenGL, jednak nasz program będzie zorientowany obiektowo, a DirectX daje lepsze możliwości enkapsulacji.

3.2. Microsoft Visual Studio 2012

4. Algorytmy, struktury danych, ograniczenia specyfikacji

Plan pracy

Podział pracy