

Grafika Komputerowa						
Rok akademicki	Termin	Rodzaj studiów	Kierunek	Prowadzący	Grupa	Sekcja
2014/2015	Piątek	SSI	INF	dr Ewa Lach	GKiO3	1
	09:30 - 11:00					



Dokumentacja końcowa

Danmaku Shooter

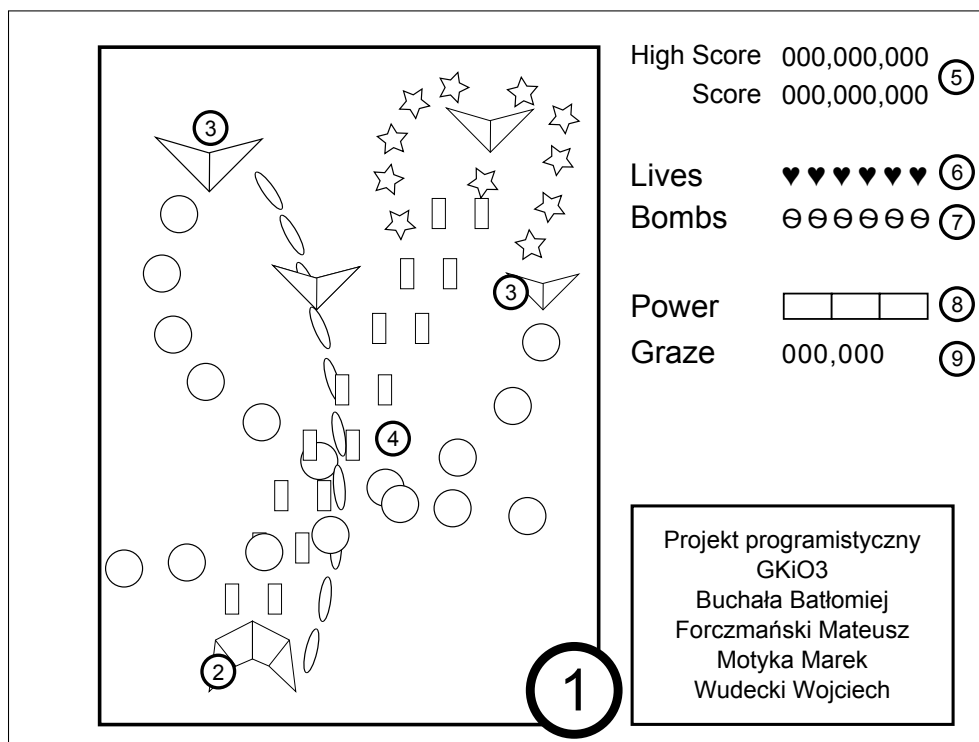
Skład sekcji:

Buchała	Bartłomiej
Forczmański	Mateusz
Motyka	Marek
Wudecki	Wojciech

Treść zadania

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 powstała jako projekt łączony z przedmiotów *Projekt Programistyczny* i *Grafika Komputerowa* oraz miała spełniać następujące założenia:



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. Chmura 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. Sprawdzanie kolizji - okrąg czy elipsa?

W trakcie naszej pracy nad programem początkowo wykorzystywaliśmy okrąg do sprawdzania kolizji. Wysokość i szerokość wszystkich sprajtów była w przybliżeniu równa, więc punkty środkowe okręgu i sprajta były równe, a promieniem był krótszy z boków. Jednak po czasie pojawiły się kształty (np. bomba), dla których obsługa kolizji poprzez okrąg wyglądałaby nieatrakcyjnie. Wówczas postanowiliśmy wykorzystać elipsę. Okrąg jest jedynie specjalnym przypadkiem elipsy o równych półosiach. Gdy obiekt miał mieć hitbox w kształcie elipsy, dłuższy bok definiował długość jednej półosi, a krótszy drugiej.

Rozwiązanie to okazało się bardzo mało efektywne - do wykrywania kolizji potrzebny były dodatkowe parametry, kąt pomiędzy obiektami. Dla jednego testu należało obliczyć ten kąt, następnie obliczyć długość promienia elipsy dla tego kąta u obu obiektów i dopiero wtedy była znana zbieżność kolizji. Dla hitboxów o równych osiach powodowało to mnóstwo niepotrzebnych obliczeń. Nawet pominięcie ich w przypadku równych półosi wymagało obliczania i przekazania kąta, który dla każdej elipsy był niezbędny.

W takiej sytuacji mieliśmy dwie możliwości:

1. *Zmiana elipsy wieloma okręgami* - eliminuje wykorzystanie elipsy, jednak dla każdego obiektu jest potrzebne zdefiniowanie ile okręgów potrzebuje, w którym miejscu i o jakiej wielkości, tak, by dopasować je do kształtu sprajta. Projektowanie ciała okręgów byłoby niezwykle niewygodne bez wsparcia designerskiego, którego nasza gra nie miała.
2. *Osobne klasy dla okręgu i elipsy* - okręgu potrzebowałyby jak najmniejszej liczby danych i obliczeń, a elipsa pozostałaby obsługiwana. Wadą jest naruszenie zasad polimorfizmu - przy każdym teście okręgi nie potrzebują kąta między obiektami, więc przekazywanie ich jest zbędne i prowadzi do niepotrzebnych operacji.

2. Sprawdzanie kolizji - zastrzyk zależności

Rozwiązaniem problemu opisanego w powyższym rozdziale okazał się wzorzec projektowy, zastrzyk zależności (*dependency injection*). Realizujemy go w ten sposób, że przy każdym teście kolizji, do jednego hitboxa wstrzykujemy drugi - pierwszy zna swoje metody działania, rzutuje drugi na odpowiedni typ i sprawdza kolizję zwracając true lub false.

Rozwiązanie jest skuteczne, ponieważ jeżeli w teście kolizji znajduje się elipsa, to pobiera one dane jakiej jej potrzeba. Jeżeli nie, test wykonywany jest najmniejszym możliwym kosztem. Sprawdzaniem, z którym typem hitboxa mamy do czynienia zajmuje się mechanizm RTTI.

Podział pracy

1. Buchała Bartłomiej	2. Forcmański Mateusz
<ul style="list-style-type: none"> • Interfejs graficzny i wyświetlanie danych • Ruch gracza po planszy • Strzelanie pociskami przez gracza • Implementacja wzorców pocisków gracza • Wystrzelenie bomby • Narysowanie sprajtów pocisków gracza i bomby • Umożliwienie pociskom ruchu po wybranych torach • Usuwanie pocisków z pamięci gdy znajdują się poza planszą • Sprawne przechodzenie pomiędzy planszami • Zapisywanie uzyskanego wyniku do pliku • Utworzenie klasy Spellcard dla Bossa 	<ul style="list-style-type: none"> • Inicjalizacja urządzenia graficznego Direct3D 9 • Trajektorie obiektów i wyliczanie przesunięcia • Przekształcenia afiniczne sprajtów i trajektorii • Synchronizacja sprajtów z obiektami gry • Wzory pocisków wrogów: kształt linii, elipsy i spirali • Wczytywanie planszy gry z pliku XML • Zarządzenie zasobami sprajtów • Utworzenie fabryk obiektów • Utworzenie parserów XML • Obsługa wadliwego formatu plików wejściowych • Obsługa zakończenia gry po pokonaniu bossa
3. Motyka Marek	4. Wudecki Wojciech
<ul style="list-style-type: none"> • Utworzenie hitboxa, jego typów oraz kształtów • Obsługa kolizji między hitboxami • Możliwość wykrywania kolizji z elipsą • Wykrywanie oraz zwiększanie otarć między obiektami • Utworzenie i obsługa wszystkich bonusów • Realizacja zmian po zderzeniu z bonusem • Przyciąganie bonusów ku graczowi • Usuwanie bonusów z pamięci • Implementacja DirectInput i reakcji na wciskanie klawiszy • Możliwość definiowania własnych klawiszy • Wyświetlanie napisów w grze (klasa Font) 	<ul style="list-style-type: none"> • Narysowanie i napisanie ekranu powitalnego • Utworzenie klasy nadrzędnej Playfield jako miejsca, gdzie mogą być wyświetlane elementy gry • Utworzenie wrogów, ich klasy i sprajtów • Realizacja zależności pomiędzy wrogami, a ich wzorami pocisków • Strzelanie pociskami przez wrogów • Generowanie bonusów przez wrogów • Narysowanie tła • Wyświetlanie wyników w podmenu Scores • Możliwość zmiany ustawień w podmenu Options • Resetowanie ustawień do wartości domyślnych

Specyfikacja zewnętrzna

Przykład działania

Specyfikacja wewnętrzna

Testowanie i uruchamianie

Wnioski