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

Dokumentacja końcowa

Danmaku Shooter

Skład sekcji:

Buchała Bartłomiej

Forczmański Mateusz

Motyka Marek

Wudecki Wojciech

Treść zadania

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 dodatkowy parametr, 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 znane zajście 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. Forczmański Mateusz
• 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 znajdą się poza planszą • Sprawne przechodzenie pomiędzy planszami • Zapisywanie uzyskanego wyniku do pliku • Utworzenie klasy Spellcard dla Bossa	• 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**
• 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)	• Narysowanie i napisanie ekranu powitalnego • Utworzenie klasy nadrzędnej Playfield jako miejsca, gdzie mogą być wyświetlanie 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