Sky Wars 1.0

Poprojektowa dokumentacja techniczna

Autorzy:
Przemysław Dobrowolski, PCC
Paweł Gorczyński, PCC
Michał Harasimowicz, RCC
Maciej Jurkowski, PCC
Piotr Komisarski, PCC

Spis treści

1. Struktura programu.

Program ma budowę modułową. Składa się z siedmiu modułów: aplikacji, lobby, menu, sceny, ustawień, dzwięku, i moduł informacji o programie (about). Cały czas w aplikacji działa przynajmniej jeden moduł, jeśli wymagane jest dzialanie większej ich ilości jeden dziala z tak zwanym fokusem, a reszta niejako w tle (np. przy rozgrywce sieciowej kiedy to moduł sceny jako glówny wyświetla na ekranie scenę 3D i prowadzi obliczenia fizyczne dzierżąc wejście z klawiatury lub innego kontrolera, a moduł lobby w tle replikuje dane symulacji i realizuje funkcje chatu).

Techniki i algorytmy użyte w najważniejszych modułach opisane są w dalszej części dokumentacji.

2. Moduł aplikacji

Nad całą synchronizacją, uruchamianiem i zamykaniem modułów zajmuje się główny moduł aplikacji. On też kieruje przekazywaniem wejścia z klawiatury i innych urządzeń sterujących między modułami, lub duplikuje owe wejście jeśli jest ono wymagane przez więcej jak jeden moduł. Sama obsługa urządzeń sterujących (w tym jojstica i kierownicy) odbywa się przy użyciu biblioteki OIS (Object oriented Input System).

To w tym module obslugiwana jest konsola, która daje możliwość zmiany wszelkich ustawień. Wywołanie jej jest zrealizowane poprzez wyłapanie wciśnięcia specjalnego klawisza '~' na klawiaturze. W tym momencie aktualny moduł jest pauzowany, wejście klawiatury "odbierane", a użytkownikowi wyświetlane jest pole konsoli. Po zakończeniu korzystania z konsoli, wejście klawiatury przekazywane jest do zapauzowanego modułu, który jest wznawiany.

3. Moduł lobby

Moduł ten odpowiada za sieć i komunikację za jej pomocą. Do obsługi sieci wykorzystana została biblioteka RakNet.

Zrealizowano tu dwie funkcjonalności. Pierwsza to chat, który zaimplementowano jako rozsyłanie pakietów do innych klientów połączonych z serwerem i przypisanych do tego samego pokoju. Druga to replikowanie danych graczy w czasie rozgrywki. Do tego zadania wykorzystano dodatkowo ReplicaManager, który automatycznie realizuje serializację danych, rozesłanie ich do wszystkich zarejestrowanych klientów i deserializację.

Rolę serwera pełni uprzywilejowany klient (tworzący rozgrywkę). Uprzywilejowanie pozwala temu klientowi na podejmowanie administracyjnych decyzji sterujących serwerem. Instancja programu u tego klienta dodatkowo wykonuje kod serwera (czyli w szczególnym przypadku, np. przy zamknięciu aplikacji przez tego klienta, komunikacja rozgrywki sieciowej zostanie przerwana jako, że serwer przestanie działać). Sam uprzywilejowany klient komunikuje się z serwerem przez ten sam interfejs co inni użytkownicy.

Moduł ten działa jako aktualny (z focusem) przy tworzeniu rozgrywki sieciowej, po jej rozpoczęciu kontynuje on swoje działanie "w tle".

4. Moduł menu

Moduł ten realizuje logikę menu głównego. Jego funkcjonalność ogranicza się do umożliwienia graczowi przejścia do obranego modułu.

5. Moduł sceny

Moduł odpowiada za inicjowanie symulacji, wykonywanie obliczeń, wyświetlanie graficznej reprezentacji, a także kończenie symulacji. Budowa tego modułu opiera się o pluginy. Wyświetlanie grafiki czy przeprowadzanie obliczeń fizycznych zaimplementowane są jako pluginy i są włączane przy inicjalizacji modułu.

Do graficznej reprezentacji świata symulacji wykorzystano bibliotekę graficzną Ogre3D, Hydrax (do wyświetlenia wody), oraz PagedGeometry (wyświetlenie drzewek). Sam samolot jest modelem z ruchomym szkieletem, wykonanym w programie 3D Studio Max. Teren zaś wczytywany jest z highmapy.

Pliki modeli samolotów pogrupowane są w katalogach wg nazwy modelu. By model został poprawnie zaimportowany, w takim katalogu powienien znaleźć się plik 3D Studio Max-a o tej samej nazwie z projektem graficznym, oraz plik xml z danymi potrzebnymi do symulacji fizycznej.

Samo wyświetlanie oddzielone jest od fizycznego modelu świata. Rysowanie samolotu odbywa sie na podstawie danych o orientacji, wychyleniu lotek i sterów, które są "pobocznym" efektem działania pluginu fizycznego.

Do implementacji fizyki użyto biblioteki ODE i wraper do niej - OgreODE. Model samolotu złożony jest z boxów o stałych masach łączonych sztywnymi połączeniami. Boxy są tak rozmieszczone żeby tworzyć sylwetkę samolotu i tym samym służyć przy wykrywaniu kolizji. Momenty bezwładności dla każdego elementu samolotu zostały ręcznie wyliczone i im przypisane. Zostało to wymuszone ograniczeniami wrapera OgreODE co do składania modeli z kilku obiektów prostych sztywno połączonych.

Dla tak przygotowanego modelu w każdej klatce symulacji wyznaczane są siły działające na części samolotu. Dokładniejszy opis liczenia sił znajduje się w dokumentacji technicznej. Rozwiązaniem równań ruchu zajmuje się już samo ODE.

W aplikacji zrealizowano również strzały. Są to wydłużone prostopadłościany o zadanej masie wystrzeliwane w kierunku osi samolotu z dużą prędkością. ODE samo z siebie zapewnia wykrycie kolizji i wywołanie odpowiedniej metody. Dla zaoszczędzenia czasu, pociski nie są tworzone przy strzale i niszczone po uderzeniu, ale znajdują się w "puli" z której są pobierane i aktywowane przy wystrzale, a po kolizji z terenem lub innym obiektem dezaktywowane i zwracane do puli, do kolejnego użycia.

W momencie trafienia w samolot (konkretnie w jedną z jego części) zmniejszany jest współczynnik "zdrowia" tej części. Współczynniki te modyfikują przykładane do części siły.

W modelu sceny realizowana jest także sztuczna inteligencja. Do bota przekazywana jest część informacji ze sceny (np. położenie i orientacja przeciwnika) na podstawie której bot wyznacza odpowiednie wychylenia sterów i lotek. Na tej podstawie niezależna od AI uproszczona metoda wyznacza położenie i orientacje bota.

Cały moduł sceny oparty jest o logikę gry. Umożliwia to stosowanie modułu przy rozgrywce sieciowej i trybie jednego gracza. Logika gry ma za zadanie odpowiednie zainicjowanie modułu i pluginów odpowiednio do rządanej logiki (gra jednoosobowa, gra wieloosobowa).

6. Moduł ustawień

Moduł przechowuje wszystkie ustawienia programu. Na początku dzialania aplikacji wczytuje on plik xml z ustawieniami i wstawia do drzewa którego węzły identyfikowane są ciągiem znaków. Odwoływanie się do wartości odbywa się poprzez odpowiednio przygotowany string złożony z

słów odpowiadających kolejnym węzłom drzewa, oddzielonych kropką (tak samo jak np. w Firefoxie). Modyfikowanie ustawień jest możliwe poprzez odpowiednie graficzne interfejsy lub przez konsolę, którą można wywołać z dowolnego modułu.

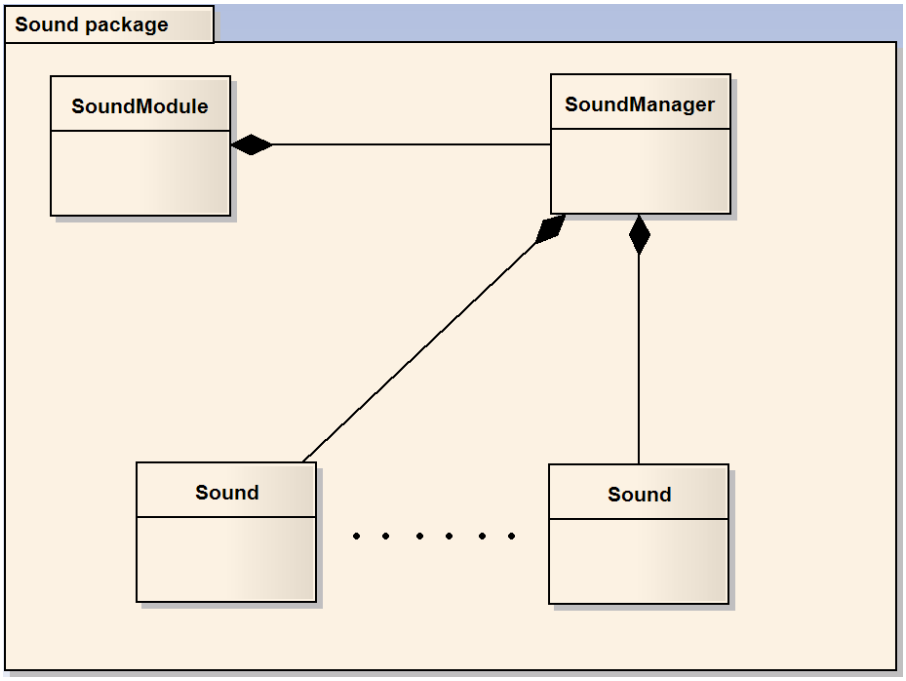
7. Moduł dźwieku

Funkcjonalność dźwięku była wykonana w technologii OpenAL.
(http://connect.creativelabs.com/openal)
Styl kodowania i konwencja OpenAL najbardziej przypomina tą z OpenGL.

OpenAL jest biblioteką dostępną na wielu platformach. Obsługuje dźwięki w wielu formatach. My używamy formatu WAV.

OpenAL udostępnia funkcje do obsługi wielu efektów, np. :

- efekt Doppler'a
- dźwięk 3D
- efekty EAX



Schemat klas modułu dźwięk

`SoundModule` – Klasa wiążąca funkcjonalność dźwięku z pozostałą architekturą. Zawiera uchwyt do modułu ustawień (`SettingsLoaderPtr`) oraz odpowiada za obsługę zdarzeń typu `OnLoad` czy `OnFrameQueued`

`SoundManager` – Manager dźwięku odpowiadający za update pozycji, prędkości i przyspieszenia słuchacza. Ładuje i zarządza poszczególnymi dźwiękami. Zawiera listę wszystkich załadowanych dźwięków w grze.

`Sound` – Klasa odpowiadająca za pojedynczy dźwięk. Ustawia pozycję, prędkość i przyspieszenie
dźwięku. Zawiera szereg przydatnych funkcji typu `Play, Stop` czy `SetVolume.`

8. Moduł "about"

Jest to moduł podobny do modułu menu. Wyświetla jedynie informacje o programie i twórcach.

9. GUI

GUI programu zostało wykonane z użyciem biblioteki CEGUI, która udostępnia do tego celu edytor. Projekty interfejsu graficznego zapisane są w deklaratwnie w pliku zewnętrznym i możliwe jest zmienianie ich bez konieczności powtórnej kompilacji projektu. Samo załadowanie GUI odbywa się w głównym module aplikacji.