Bartłomiej Czerwiński
Rafał Majewski

20.01.2016r.

Projekt ZPR
Dokumentacja końcowa

Prowadzący projekt: dr inż. Rafał Biedrzycki

Projekt realizowany w ramach zajęć z przedmiotu Zaawansowane Programowanie w C++ na wydziale EiTI PW rok 2015/2016 semestr zimowy.

1. Temat projektu

Gra zainspirowana klasycznym tytułem “Asteroids” z możliwością uczestniczenia w rozgrywce jednego lub dwóch graczy.

2. Ogólna idea

Gra ma umożliwiać grę na pojedynczej maszynie na jednego lub dwóch graczy. Ekran gry składa się z nieruchomej planszy, na której pojawiają się ruchome obiekty - rakieta, asteroidy i pociski, a także nieruchome power-upy. Celem gracza sterującego rakietą jest uzyskanie jak najwyższego wyniku punktów, które zdobywa poprzez niszczenie asteroid za pomocą działka zamontowanego z przodu rakiety. Zadaniem drugiego gracza (bądź, w przypadku gry jednoosobowej, algorytmu generowania asteroid) jest zniszczenie rakiety przy pomocy asteroid wystrzeliwanych od krawędzi ekranu w zadanym kierunku i o zadanej prędkości. Rozgrywka toczy się do momentu utraty przez gracza sterującego rakietą wszystkich dostępnych żyć.

3. Funkcjonalność

Uzyskana funkcjonalność:

1) Podstawowa mechanika rozgrywki, obsługa jednego bądź dwóch graczy, urozmaicenia w postaci power-upów.
2) Konsola obsługująca skrypty w języku Python, wywoływana w dowolnym momencie rozgrywki za pomocą klawisza “~”.
3) Obsługa skryptów zapisanych w specjalnym pliku - łatwa możliwość modyfikacji działania gry.
4) Menu gry umożliwiające wybrać ilość graczy oraz zmienić rozdzielczość, w jakiej działa gra (im wyższa rozdzielczość, tym większa powierzchnia planszy gry).

5) Komunikat o zakończeniu rozgrywki w momencie utraty wszystkich żyć, po którym można rozpocząć nową rozgrywkę z menu głównego.

Funkcjonalność, na którą zabrakło czasu:

1) Tabela z wynikami uzyskanymi przez graczy z podziałem na tryb gry.
2) Animowane tła ekranów.
3) Dodatkowe zagrożenia dla rakiety, takie jak czarne dziury.
4) Opcja pełnego ekranu, zmiany głośności muzyki/odgłosów w grze.

4. Użyte technologie

Aplikacja

Użyte biblioteki, kompatybilne z systemami Windows oraz Linux, to biblioteka Allegro 5 odpowiedzialna za obsługę ekranu i urządzeń wejścia, Box2D do symulacji fizycznej obiektów na planszy oraz boost wraz z boost::python.

System kontroli wersji

Podczas tworzenia aplikacji został użyty system kontroli wersji *git* w serwisie github pod adresem https://github.com/defacto2k15/pwAsteroids

5. Architektura aplikacji

Na najwyższym poziomie abstrakcji postanowiliśmy nasz program podzielić na trzy części zgodnie ze wzorcem MVC. Miało to na celu umożliwienie pracy nad poszczególnymi modułami w stosunkowo dużym odseparowaniu of pozostałem części programu. Co więcej, liczyliśmy że luźniej połączone elementy prościej będzie testować

5A. Model

Kod obsługujący logikę gry, łączący się z Box2d czy językiem python znajduje się w Modelu. Architekturę tej części programu wzorowaliśmy na "*Entity component model",* wzorcu często używanym przy tworzeniu gier komputerowych. Wyszliśmy od założenia, że obiekty w modelu działać będą w pętli, a w każdym jej obrocie model jako całość będzie reagował na dane wejściowe (Takie jak informacje o naciśnięciu klawiszy) generowaniem danych wyjściowych, np współrzędnymi bitmap do wyświetlenia, po czym silnik graficzny używając tych danych wyrenderuje obraz. Aby lepiej wspierać działanie programu w pętli podzieliliśmy klasy na kilka podgrup, w których dziedziczono po wspólnym typie bazowym. Wyróżniliśmy następujące podgrupy: *Aktorów,* reprezentujących pojedyncze byty w grze (zwykle wyświetlane na ekranie, np. rakieta). Aktorzy posiadali grupę *Komponentów:* obiektów wykonujących jedną czynność na rzecz swojego aktora, np. wysyłających obrazek go reprezentujący na wyjście modelu. Komponenty można było dodawać i usuwać z aktora podczas wykonywania programu, co więcej komponenty tego samego aktora w prosty sposób mogły się ze sobą

komunikować. Istniały także klasy - *serwisy* posiadające zwykle jedną instancje i wykonujące operacje “ponad” część gry kontrolowanej przez graczy, takie jak symulowanie zderzeń czy zliczanie czasu. Wszystkie klasy z podgrup posiadały trzy metody wirtualne: *OnStart* wywoływane raz, po rozpoczęciu działania modelu, *OnUpdate* wywoływane co obrót pętli (*Co klatkę)* oraz *OnStop,* wywoływane raz przed usunięciem obiektu. Dodanie dwóch skrajnych metod podyktowane było potrzebą lepszej kontroli tworzenia obiektów. Konstruktor używany był jedynie do inicjalizacji pól i wykonania prostych akcji. *OnStart* na obiekcie wywoływano wtedy, gdy skonstruowano już całe jego “otoczenie” z którym będzie komunikował się podczas działania.

5B. Kontroler

Klasy z grupy kontrolera stanowią spoiwo między dosyć abstrakcyjnym modelem a niskopoziomowym widokiem. Kontroler przede wszystkim zajmuje się tworzeniem i utrzymaniem pętli wydarzeń generowanych przez silnik allegro. Wydarzenia te są interpretowane, po czym przekazywane są do aktualnie aktywnego Interpretera wydarzeń (*EventInterpreter).* Do każdego *Ekranu* widoku przypisany jest jeden interpreter który zajmuje się reagowaniem na akcje takie jak wciśnięcie klawisza. Wywołuje on odpowiednie metody na swoim ekranie, by zmienić treść na nim wyświetlaną.

5C. Widok

Widok zajmuje się wyświetlaniem obrazów używając silnika allegro, i tworzy warstwe abstrakcji nad tym silnikiem dla Kontrolera. W widoku operujemy na *DrawableObjects* które obudowują bitmapy zawierając wiadomości o ich położeniu, obrocie czy wielkości. Obiekty tej klasy grupujemy w *sceny*, a kilka takich scen(np scena tła i scena obiektów ruchomych) tworzy osobny *ekran,* który to jest już bezpośrednio zarządzany przez kontroler.

6. Rozgrywka

Uruchomienie rozgrywki następuje po wybraniu trybu gry z menu głównego. Gracz sterujący rakietą używa klawiszy WSAD bez “S” do poruszania rakietą - “W” odpowiada za ciąg nadający prędkość do przodu, a klawisze “A” i “D” za skręcanie rakietą, również w miejscu. Proces hamowania, który ze względu na komfort rozgrywki jest szybszy niż przyspieszanie do przodu, odbywa się poprzez użycie ciągu silnika w stronę przeciwną do ruchu rakiety. Dodatkowo gracz może strzelać z działka umieszczonego na przodzie rakiety, które pozwala niszczyć asteroidy. Asteroidy występują w kilku rozmiarach - zniszczenie większej powoduje pojawienie się w jej miejsce kilku mniejszych, dopiero zniszczenie najmniejszych powoduje ich ostateczne unicestwienie.

Drugi z graczy do sterowania używa strzałek lewo i prawo oraz myszki, która pełni rolę wskaźnika. Za pomocą strzałek przesuwa on punkt na krawędzi

ekranu, który wyznacza miejsce pojawienia się asteroidy, a przy pomocy myszy wskazuje on miejsce, w które poleci asteroida (wystrzelenie odbywa się za pomocą lewego klawisza myszy). Rozmiar asteroidy oraz jej prędkość zależy od odległości celownika od miejsca wystrzelenia - im jest ona mniejsza, tym asteroida będzie większa, ale za to wolniejsza.

Utrata życia przez gracza sterującego rakietą następuje po zbyt mocnym zderzeniu z asteroidą - lekkie otarcia nie powodują jego straty. Dodatkowym ułatwieniem dla gracza sterującego rakietą są tzw. power-upy, czyli wzmocnienia, które pojawiają się w sposób losowy na planszy. Dostępne są dwa power-upy - dodatkowe życie oraz “super strzał”, który przez kilka sekund pozwala rakiecie strzelać trzema pociskami naraz.

Istotnym aspektem mechaniki gry jest fakt, że krawędź ekranu nie stanowi w grze przeszkody - niezniszczone asteroidy i chybione pociski po wyleceniu z planszy znikają, ale za to rakieta pojawia się z drugiej strony ekranu. Przejście jest płynne - w czasie przechodzenia część rakiety jest widoczna w jednym miejscu, a pozostała część w innym.

7. Uruchamianie i kompilacja

Linux

Windows

Do poprawnej kompilacji aplikacji konieczne będzie zainstalowanie i zbudowanie kilku bibliotek. Instrukcje dostępne są w pliku “README.txt”.

8. Testy

Testowanie jest tą częścią projektu gdzie osiągnęliśmy najmniej satysfakcjonujące wyniki. W początkowych pracy nad projektem zakładaliśmy, że uda nam się tworzyć bogaty pakiet testów który powstawałby razem z kodem programu. Planowaliśmy do tego celu używać biblioteki *google test* wraz z biblioteką *google mock* do tworzenia atrap obiektów. Planowaliśmy używać też możliwości jakie daje nam język Python do bezpośredniego sterowania modelem. W toku pracy nie udało się nam osiągnąć tych celów. Tworzenie *mock objects* w języku bez maszyny wirtualnej, takim jak c++ było utrudnione i wymagało generowania kodu który musiał być utrzymywany w zgodności w kodem klas “atrapowanych” Co więcej przez tworzenie testów wydłużał się czas kompilacji, który na dosyć przeciętnym laptopie mógł dochodzić do siedmiu minut. Co więcej przeszacowaliśmy nasze możliwości w zakresie szybkiego tworzenia działającego kodu, przez co w późnym czasie projektu testowanie zarzucono z powodu braku czasu.Należy jednak zauważyć, że z tych testów które zostały napisane można wyciągnąć interesujące wnioski. Przede wszystkim testowanie modelu w odosobnieniu od reszty programu może dawać dobre efekty. Wspieranie języka Python wraz z prostym interfejsem wyściowym modelu sprawia że możliwe staje się tworzenie prostych “eksperymentów” i

sprawdzanie ich wyników (plik *AsteroidCollisionEndToEndTest.cpp).* Co więcej tworzenie prostych testów, bez atrap zdaje się sprawdzać przy niewielkich klasach nie przechowujących odnośników do większych klas.

9. Napotkane problemy i trudności

Pierwszym i jednym z największych problemów było niedoszacowanie czasu którego poświęcenia wymagał projekt. Jak widać z porównania czasu przewidywanego z rzeczywistym, wykonanie zadania zajeło nam ponad dwa razy więcej czasu niż się spodziewaliśmy. Ponadto dosyć trudne okazało się regularne pisanie testów. Dużym problemem był czasami bardzo długi czas kompilacji, który nie pozwalał na szybkie naprawianie błędów w kodzie. Bardzo poważnych trudności dostarczało nam też takie pisanie programu by działał tak na systemie Windows jak i linux. Obecnie na niektórych komputerach z systemem Eindows gra nie działa prawidłowo (złe wykrywanie kolizji), ponadto nie zawsze udawało się nam uruchomić na tej platformie część gry obsługującej język Python

10. Lista zadań wraz z czasami

Zadanie	Przewidywany czas wykonania	Faktyczny czas wykonania
Architektura aplikacji ● model z logiką gry ● połączenie z warstwą wyświetlającą grafikę	= 6h	= 20h
Mechanika gry: ● rakieta - system poruszania, obsługi ● asteroidy - system poruszania, zniszczeń ● system kolizji ● działko pozwalające niszczyć asteroidy ● system ulepszeń (powerupy) ● generator asteroid dla gry jednoosobowej ● system zliczania punktów	= 12h	= 32h
System sterowania ● programem (menu) ● rakietą ● wystrzeliwaniem asteroid	= 3h	= 5h
Grafika	= 7h	= 18h

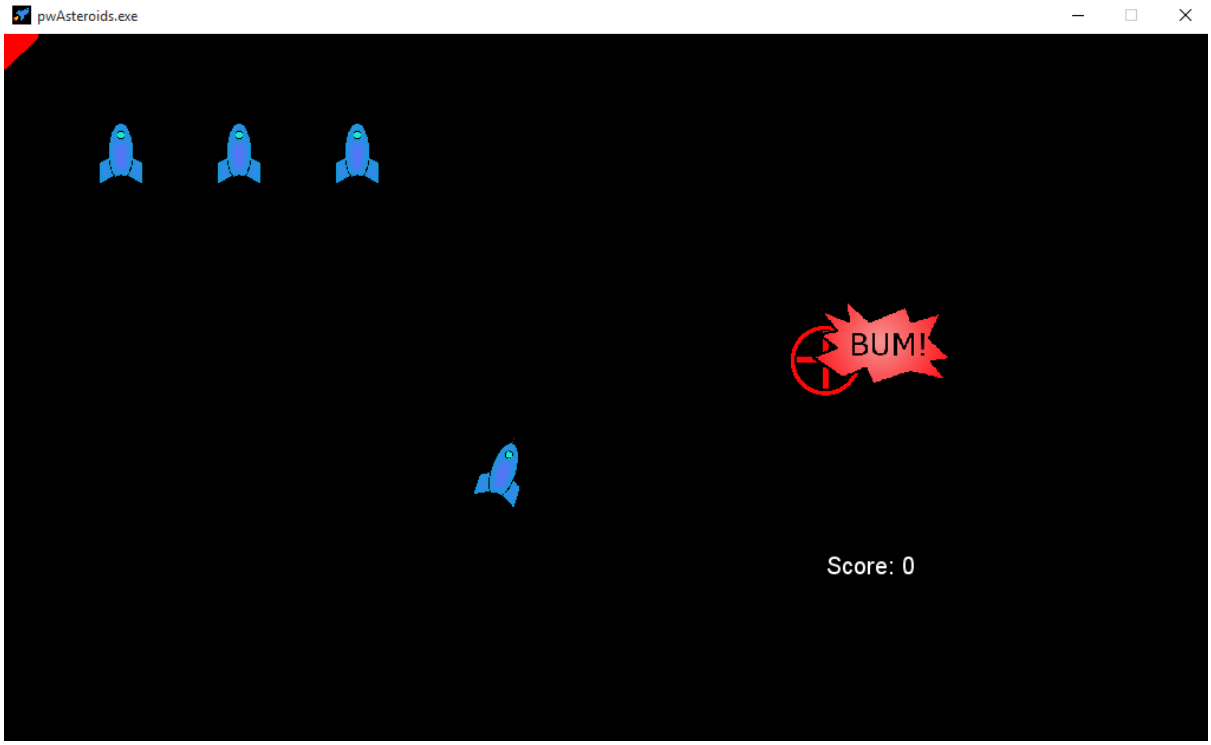
• implementacja zarządzania ekranem • zaprojektowanie i wykonanie menu oraz GUI • oprawa graficzna		
Dźwięk • podstawowa oprawa dźwiękowa	= 3h	= 1h
Obsługa skryptów • konsola obsługująca pythonowe skrypty	= 6h	= 12h
Dokumentacja końcowa	= 2h	= 4h
	Razem: ~39h	Razem: ~92h

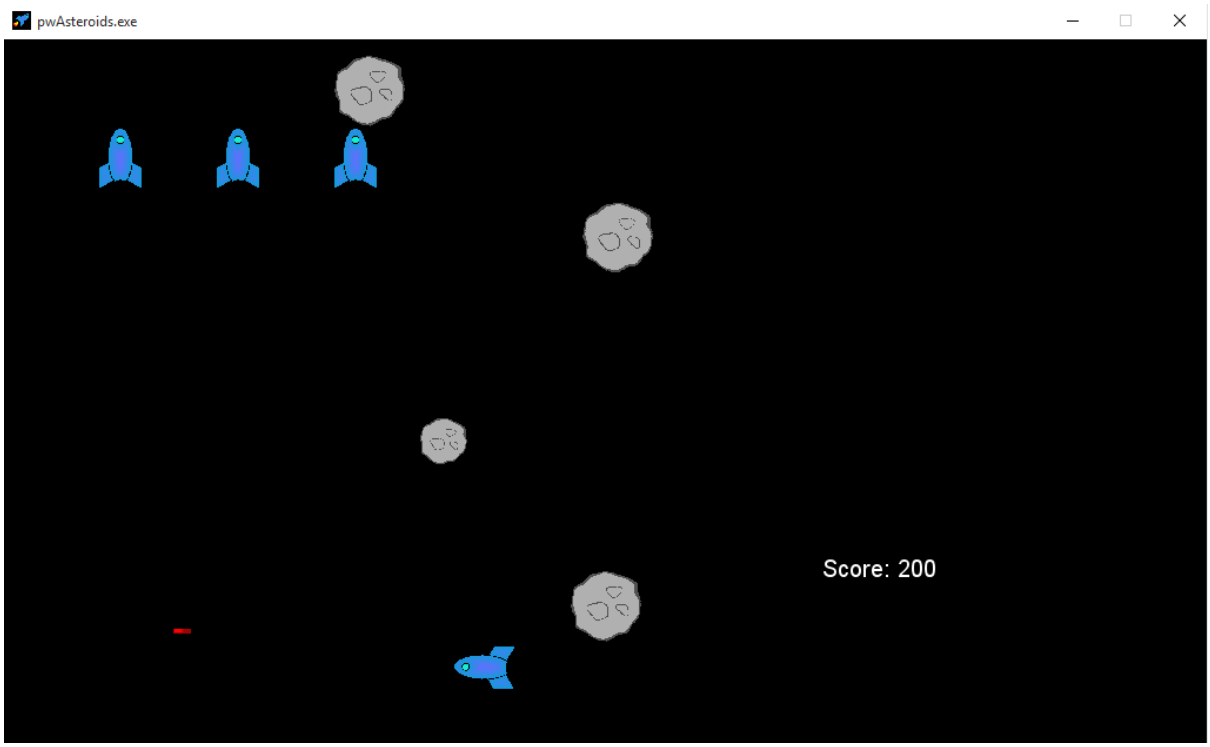
11. Ilość linii kodu

- Pliki nagłówkowe: **6009** (z czego niepustych: **4451**)
- Pliki .cpp: **5535** (z czego niepustych: **4555**)

Razem: **11544** (z czego niepustych: **9006**)

12. Zdjęcia z gry




pwAsteroids.exe
Score: 200