Gliwice, 22.05.2018 r.

Projekt z Grafiki Komputerowej

Sprawozdanie z projektu

Tytuł projektu:
Gra 2D typu Tower Defence

Prowadzący:
dr Ewa Lach

Skład sekcji:
Karol Marszałek
Błażej Moska
Kamil Janas

1. Treść zadania

Gra typu Tower Defence 2D zrealizowana z wykorzystaniem jezyka C++ oraz bibliotek graficznych umozliwiajacych tworzenie gier komputerowych w tym języku. Gra opiera się na popularnym modelu gier typu Tower Defence polegającym na obronieniu pewnego kluczowego punktu na mapie, na przykład zamku, przed nadchodzącymi przeciwnikami. Przeciwnicy poruszają się po ściśle określonej ścieżce na mapie, na pozostałych polach użytkownik może ustawiać wieże eliminujące przeciwników w drodze do zamku. Rozgrywka kończy się wygraną gracza jeżeli pokona on wszystkich przeciwników lub jego porażką jeżeli przeciwnicy zniszczą zamek.

2. Analiza zadania

Pierwotna analiza zadania:

1. Podstawy teoretyczne problemu:
2. Wykorzystane zagadnienia grafiki komputerowej:
    1. Przekształcenia afiniczne
    2. Shadery
    3. Wykrywanie kolizji
3. Wykorzystane biblioteki i narzedza programistyczne:
    1. Visual Studio jako rozbudowane narzedzie programistyczne zostanie wykorzystane w celu implementacji projektu z wykorzystaniem jezyka C++.
    2. Biblioteka SFML - biblioteka graficzna do jezyka C++ ułatwiajaca korzystanie z mechanizmów do tworzenia grafiki komputerowej w wykorzystywanym środowisku programistycznym. SFML wprowadza podejście obiektowe do programowania efektów graficznych co pozwala na lepszą integrację warstwy graficznej programu z resztą programu napisaną z wykorzystaniem podejścia programowania zorientowanego obiektowo. Dodatkowo biblioteka SFML zapewnia multiplatformowość tworzonych z jej wykorzystaniem programów.Jest ona zorientowana w celu optymalizacji grafiki dwuwymiarowej, w związku z czym nie wspiera ona mechanizmów do tworzenia grafiki trójwymiarowej. Jest to ograniczenie, ktore nie wpływa na specyfikację projektu ze względu na wybór implementacji grafiki dwuwymiarowej. Biblioteka SFML nie jest przystosowana do prostego tworzenia graficznego interfejsu użytkownika w przeciwieństwie do, na przykład, bibiotelki QT, jednak jej możliwosci pozwalają na tworzenie intuicyjnego interfejsu gry komputerowej.
4. Algorytmy, struktury danych, ograniczenia specyfikacji
    1. Struktury danych:
        1. Klasa opisujaca mapę zrealizowana wewnętrznie w postaci tablicy jednowymiarowej.
        2. Klasa umożliwiajaca przechowywanie historii wyników gracza zapisywanych na dysku twardym użytkownika, wczytywana do aplikacji przy jej uruchomieniu do listy.
        3. Klasa przechowujaca konfigurację klas graficznych, w tym zbiór tekstur wykorzystywanych przez grę w tablicy.
        4. Klasa przechowująca informacje o znajdujących sie na mapie przeciwnikach implementowana z wykorzystaniem tablicy jednowymiarowej.
        5. Klasa przechowująca informacje o znajdujących sie na mapie wieżach ustawionych przez użytkownika w postaci tablicy jednowymiarowej.

2. Algorytmy:
    1. Algorytm wyboru celu wieży na mapie na podstawie jej zasiegu oraz położenia przeciwników na mapie.
    2. Algorytm wyznaczania trajektorii pocisku.
    3. Algorytm wyznaczania trasy przejścia przeciwników po mapie.
3. Ograniczenia specyfikacji:
    1. Rozmiar planszy - minimalnie: 250x250 px, maksymalnie 500x500 px.
    2. Ograniczenie grafiki do grafiki dwuwymiarowej.
    3. Gra przeznaczona jest dla jednego uzytkownika bez rozróżnienia uzytkowników lokalnych.

Odstępstwa pierwotnych założeń od zrealizowanego zadania:

W trakcie realizacji projektu zrezygnowano z tworzenia klasy z punktu 4.1.3 przechowującej historię rozgrywki na rzecz zestawu klas pozwalającego na dynamiczne zarządzanie poziomami rozgrywki odczytywanymi z pliku w formacie XML. Pozwalają one na tworzenie konfiguracji poziomów w łatwo rozszerzalny sposób, poprzez modyfikację pliku zawierającego konfigurację rozgrywki. Rozwiązanie to łatwo jest rozszerzyć o przechowywanie wielu różnych konfiguracji w różnych plikach, jednak taka funkcjonalność nie została zrealizowana w ramach projektu. Z pominięciem nieistotnych z punktu widzenia funkcjonalności klas usprawniających implementację, nie poczyniono więcej zmian w realizacji projektu.

3. Podział pracy

1. Karol Marszałek:
    1. Przygotowanie środowiska programistycznego oraz repozytorium, stworzenie projektu odpowiedzialnego za grafikę oraz działającej pustej aplikacji graficznej.
    2. Stworzenie klasy implementującej mape gry w formie tablicy.
    3. Implementacja reguł ścieżek przechodzenia planszy przez przeciwników w przypadku rozgałęzienia ścieżek.
    4. Wykrywanie kolizji pomiedzy przeciwnikami oraz odczytanie reguł przejścia po planszy w przypadku rozgałęzień ścieżek.
    5. Narysowanie pocisków na mapie.
    6. Rozpoznanie warunków końca rozgrywki w funkcji main programu.
2. Kamil Janas:
    1. Stworzenie klasy przechowującej globalne ustawienia grafik, między innymi ścieżki tekstur.
    2. Stworzenie klasy rysującej mape na podstawie tablicy.
    3. Obsługa wież na planszy - umieszczenie, wykrywanie kolizji.
    4. Implementacja algorytmu obliczającego trajektorie pocisków.
    5. Implementacja zarządzania pieniędzmi gracza zdobywanymi za pokonywanie przeciwników

3. Błażej Moska:
    1. Osbługa przeciwników na planszy - rysowanie przeciwników oraz przejście po planszy zgodnie z regulami zdefiniowanymi na planszy.
    2. Implementacja algorytmu wyboru celu dla wiezy.
    3. Implementacja kolizji pocisku oraz przeciwnika.
    4. Implementacja usuwania pokonanych przeciwników z mapy.
    5. Implementacja parsera pliku XML oraz klas przechowujących konfigurację poziomów

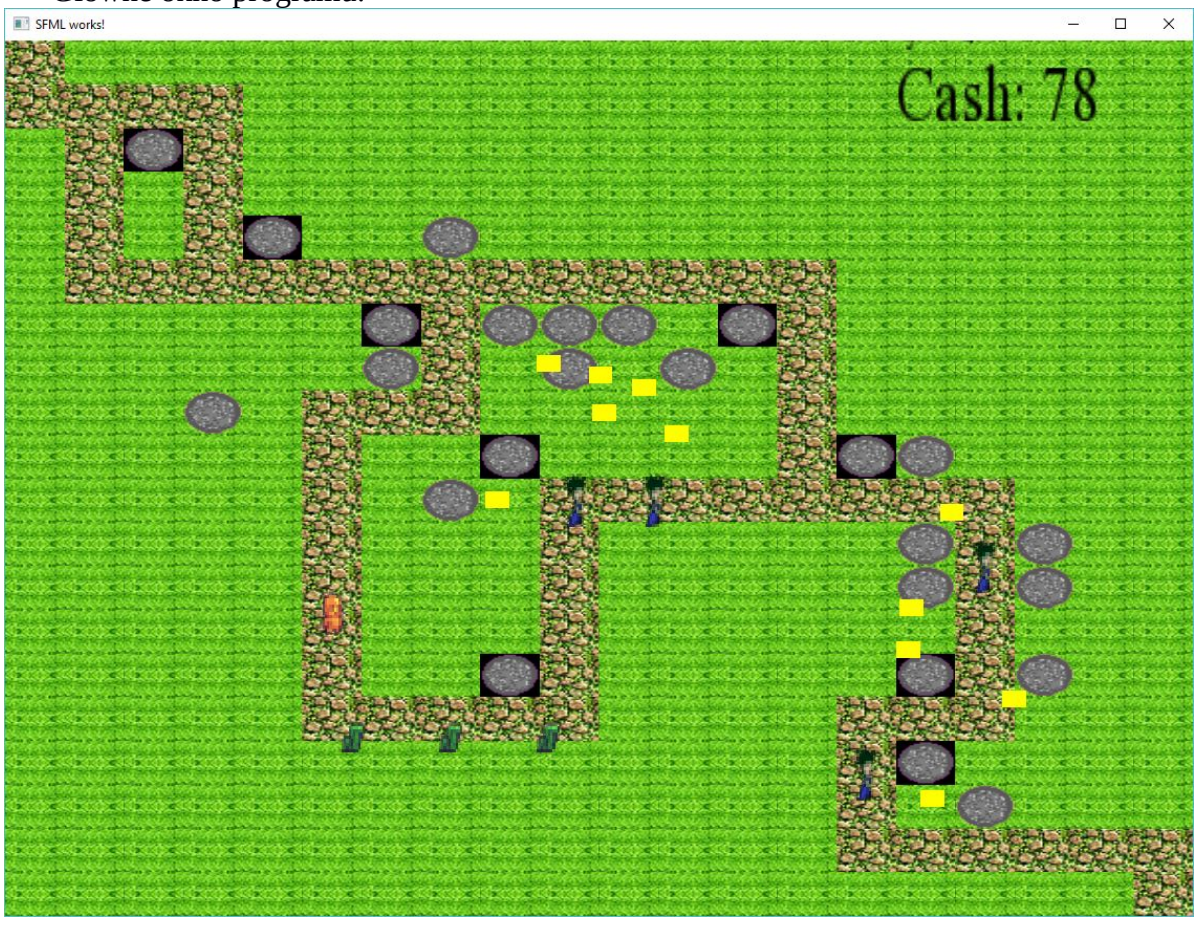
Dodatkowo każdy z członków zespołu był zaangażowany w testowanie programu w trakcie implementacji oraz finalnego testowania ukończonego programu.

4. Specyfikacja zewnętrzna

Gra posiada bardzo uproszczony, dzięki czemu intuicyjny interfejs użytkownika. Po uruchomieniu aplikacji użytkownik zostaje powiadomiony o rozpoczęciu rozgrywki poprzez wyświetlenie okna dialogowego. Po jego zamknięciu rozpoczyna się rozgrywka, otwierane jest okno z mapą, a na planszy zaczynają pojawiać się przeciwnicy. W rogu ekranu użytkownik może zobaczyć ilość posiadanych punktów życia oraz pieniędzy. Poprzez klikanie myszką po mapie gry użytkownik może umieszczać wieże na planszy rozgrywki, które zaczynają strzelać do przeciwników. Rozpoczęcie każdego poziomu oraz każdej fali w ramach poziomu sygnalizowane jest wyświetleniem okna dialogowego. Rozgrywka kończy się w momencie kiedy graczowi skończą się punkty życia lub gdy pokona on wszystkich przeciwników. Zakończenie rozgrywki, zarówno po wygranej jak i porażce gracza sygnalizowane jest oknem dialogowym.

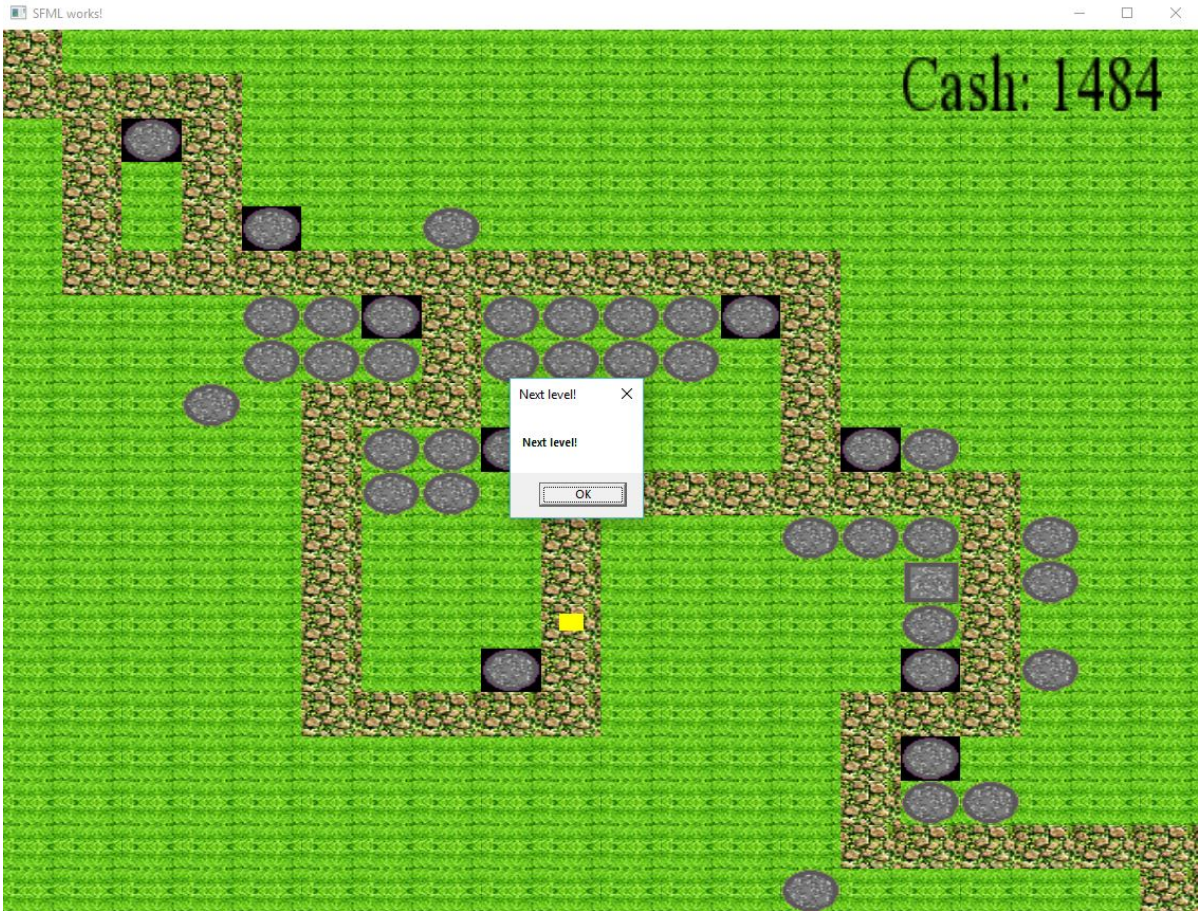
5. Przykład działania

Główne okno programu:



Informacja dla użytkownika o początku następnej fali w ramach poziomu:

Informacja dla użytkowniku o początku nstępnego poziomu:



6. Specyfikacja wewnętrzna

https://karmar1995.github.io/GK/index.html

7. Testowanie i uruchamianie

Gra została przetestowana pod kątem poprawności działania. Do danych testowych zalicza się przypadki niepoprawnych plików mapy, błędy jednak nie są sygnalizowane użytkownikowi, gdyż nie ma on bezpośredniego wpływu na pliki mapy, gdyż są to wewnętrzne dane aplikacji. Ponadto sprawdzono poprawność działania samej gry, także w przypadku zmian w pliku konfiguracyjnym XML. Gra działa poprawnie, zarówno pod kątem rozgrywki, poruszania się obiektów po scenie jak i rozpoznawania zdarzeń zakończeń rozgrywki.

8. Wnioski

Prace nad projektem przebiegły zgodnie z planem zadeklarowanym na początku pracy nad projektem. Podział pracy umożliwiał równoległe realizowanie zadań przez kolejnych członków zespołu na kolejnych etapach tworzenia programu, co wraz z zastosowaniem mechanizmów programowania zorientowanego obiektowo, takich jak polimorfizm czy podział implementacji poprzez interfejsy, pozwoliło w prosty sposób dokładać kolejne funkcjonalności do już istniejących co umożliwiło szybszą implementację oraz testowanie, dzięki czemu możliwe było wcześniejsze ukończenie projektu niż pierwotnie zakładał plan pracy.