Gliwice, 29.06.2014

Sprawozdanie z projektu

Programowanie Komputerów 4

Temat: Dwuwymiarowa gra platformowa w języku C++

Marcin Okularczyk
Informatyka sem. IV
gr. 5, sekcja1

1. Analiza tematu

   Tematem projektu było stworzenie dwuwymiarowej gry platformowej w języku C++. Sterowanie odbywa się za pomocą klawiatury i myszki. Zadaniem gracza jest pokierowanie postacią przez wszystkie poziomy, zbierając przy tym rozmieszczone na mapach przedmioty oraz radząc sobie z przeciwnikami.

   Do napisania programu wykorzystałem bibliotekę SFML w wersji 2.1.

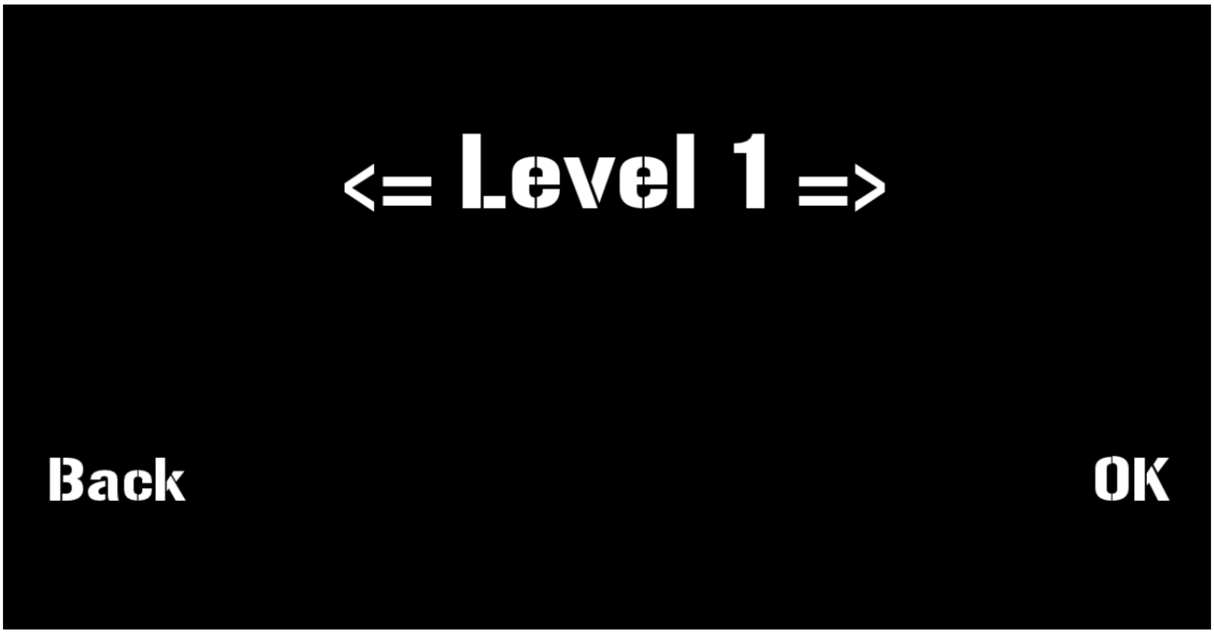
2. Specyfikacja zewnętrzna

Po uruchomieniu gry wyświetlone zostaje główne menu programu:

Użytkownik ma do wyboru następujące opcje:

New Game – rozpoczęcie nowej rozgrywki od pierwszego poziomu
Select Level – rozpoczęcie rozgrywki od wybranego poziomu
Exit – wyjście z gry

Menu wyboru poziomu:



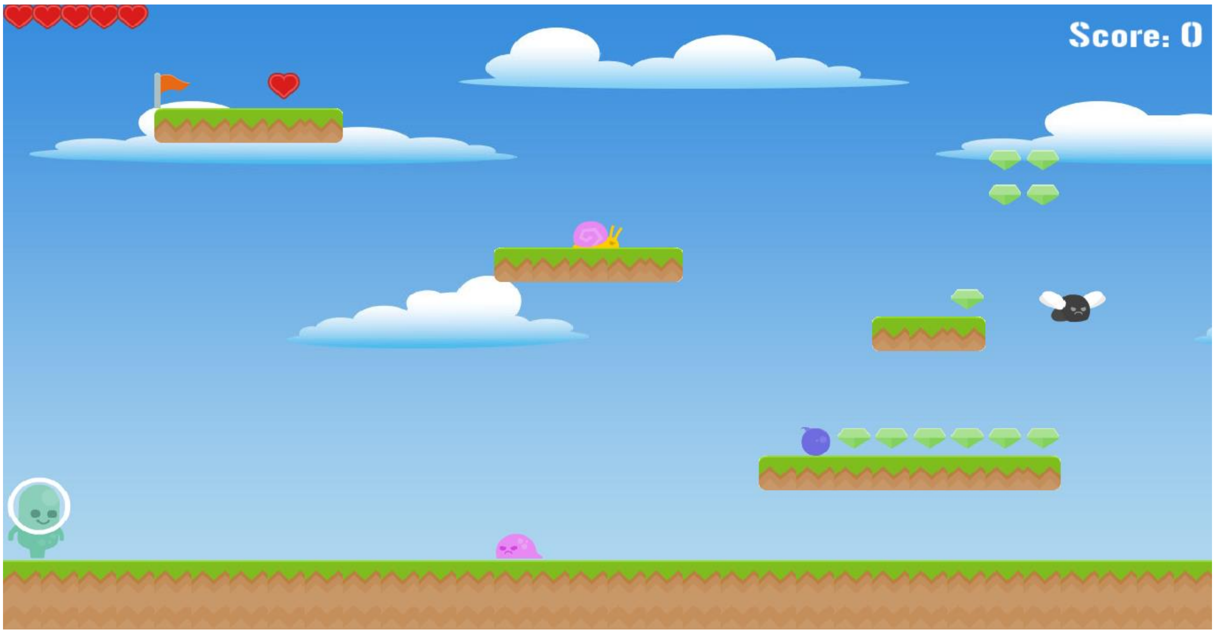
Strzałki – przewijanie dostępnych poziomów
OK – rozpoczyna grę od wybranego poziomu
Back – powrót do głównego menu

W trakcie rozgrywki menu dostępne jest pod klawiszem „Esc” i oferuje następujące opcje:

Continue – powrót do gry

Exit to main – kończy rozgrywkę i wyświetla główne menu programu

Ekran gry:



W lewym dolnym rogu widzimy postać, którą steruje gracz. Sterowanie odbywa się przy pomocy klawiatury: strzałka w prawo – ruch w prawo, strzałka w lewo – ruch w lewo, strzałka w górę – skok, Esc – wywołanie menu.

Dodatkowo na ekranie wyświetlani są wrogowie, których gracz może ominąć lub wyeliminować poprzez „wskoczenie” na wroga. Inny kontakt z przeciwnikiem powoduje zmniejszenie poziomu zdrowia gracza, który widoczny jest w lewym górnym rogu ekranu. W przypadku zmniejszenia poziomu zdrowia do zera, wyświetlony zostaje ekran końca gry:

Game Over

Your score is: 700

Wciśnięcie dowolnego przycisku klawiatury spowoduje przejście do menu głównego gry.

Na każdym poziomach rozmieszczone są również przedmioty o różnym działaniu:

Flaga – przejście do kolejnego poziomu,

Serce – zwiększenie poziomu zdrowia o 1,

Bomba – zmniejszenie poziomu życia, odepchnięcie gracza,

Diament – zwiększenie wyniku,

Po pomyślnym przejściu wszystkich poziomów wyświetlany jest ekran końcowy:

YOU WIN!

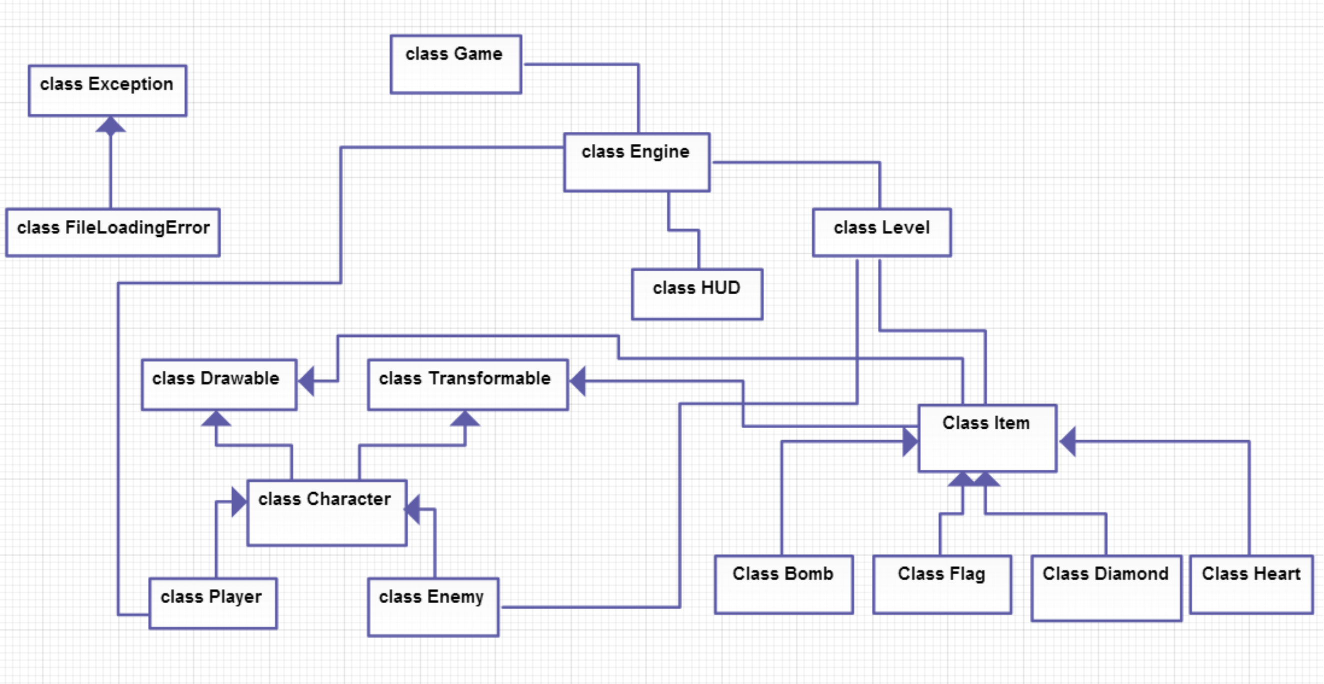
Your score: 2200

Press any key to continue

Naciśnięcie dowolnego przycisku klawiatury spowoduje przejście do głównego menu programu.

3. Specyfikacja wewnętrzna

Struktura programu przedstawiona została na poniższym diagramie klas:



klasa Game: główna klasa gry. Odpowiada za stan programu (gra, menu, wybór poziomu). W konstruktorze ładowana jest czcionka menu gry. Klasa przechowuje obiekt okna gry typu *RenderWindow* oraz iterator numeru poziomu typu int. Zawiera funkcje: *void menum(),* odpowiadającą za wyświetlanie i obsługę menu głównego, void single() uruchamiającą nową grę oraz *void load(),* która odpowiada za wczytanie poziomu, na który wskazuje iterator poziomów.

klasa Engine: najbardziej rozbudowana klasa programu. Odpowiada za rysowanie i wyświetlanie okna gry, a także za prostą fizykę w grze (grawitacja, kolizja obiektów). Główną metodą jest tutaj metoda *void RunEngine(),* która obsługuje silnik gry.

klasa Character: klasa abstrakcyjna, zawierająca metody wspólne dla dla klasy Player oraz klasy Enemy. Wykorzystuje dziedziczenie wielobazowe, dziedzicząc klasy *Transformable* oraz *Drawable*, należące do wykorzystanej biblioteki SFML. Dzięki temu możliwe jest wykonanie operacji *draw(obiekt*). Zawiera m.in. typ wyliczeniowy opisujący aktualny stan postaci, metody odpowiadające za pobranie danych takich jak pozycja obiektu, szybkość poruszania oraz metody wirtualne odpowiadające za poruszanie i aktualizację animacji postaci.

klasa Enemy: dziedziczy po klasie *Character*. Zawiera m.in. tablicę tekstur, pole opisujące długość drogi jaką poruszają się przeciwnicy, metody odpowiadające za poruszanie się oraz wczytywanie konkretnych typów przeciwników, różniących się teksturami, prędkością poruszania, zadawanymi obrażeniami. Tekstury postaci wczytywane są w konstruktorze.

klasa Player: dziedziczy po klasie *Character.* Zawiera rozszerzoną względem klasy Enemy tablicę tekstur, pole *int health* opisujące ilość zdrowia gracza, a także metody obsługujące ruch postaci gracza. Tekstury postaci wczytywane są w konstruktorze.

klasa HUD: zawiera pola i metody do obsługi interfejsu gracza (pasek zdrowia, informacja o wyniku) a także zmienną *int changeflag* informującą o tym, czy nastąpiła zmiana poziomu. Potrzebne tekstury oraz czcionka wczytywane są w konstruktorze.

klasa Level: przechowuje dane poziomów. Zawiera m.in. tekstury elementów mapy, które wczytywane są w konstruktorze, tablicę przechowującą informacje o fragmentach ekranu, a także wektory wskaźników na klasy Item oraz Enemy, przechowujące informacje o przedmiotach oraz przeciwnikach na danym poziomie. Zawiera również metody wczytujące w plików tekstowych dane o budowie poziomu, rozmieszczeniu przedmiotów oraz przeciwników (*bool LoadFromFile(string filename), bool LoadEnemyList(string filename), bool LoadItemList(string filename)*).

klasa Item: klasa abstrakcyjna, dziedzicząca po klasach *Drawable* i *Transformable*. Zawiera metodę *void PlaceItem(int x, int y)* ustawiająca obiekt na pozycji (x,y), metodę *void Contact(Player &player*) sprawdzającą, czy nastąpił kontakt między graczem a przedmiotem. Dodatkowo zawiera metodę wirtualną *virtual void Effect(HUD &hud, int &it)* przyjmującą jako argumenty referencje do HUD'a oraz iteratora poziomów, obsługującą efekt danego przedmiotu.

klasa Flag: dziedziczy po klasie *Item*. W konstruktorze wczytywana jest tekstura flagi. Wykonanie metody *Effect* powoduje zwiększenie iteratora poziomu i w efekcie przejście do kolejnego poziomu.

klasa Bomb: dziedziczy po klasie *Item*. W konstruktorze wczytywana jest tekstura bomby. Zawiera dodatkowe pole przechowujące ilość obrażeń, jakie przedmiot wyrządza graczowi w razie kontaktu. Wykonanie metody *Effect* powoduje zadanie graczowi obrażeń i odepchnięcie go.

klasa Diamond: dziedziczy po klasie *Item*. W konstruktorze wczytywana jest tekstura diamentu. Zawiera dodatkowe pole przechowujące wartość dodawaną do wyniku w razie zebrania diamentu. Wykonanie metody *Effect* powoduje dodanie tej wartości do wyniku gracza.

klasa Heart: dziedziczy po klasie *item*. W konstruktorze wczytywana jest tekstura serca. Zawiera dodatkowe pole przechowujące ilość zdrowia dodanego graczowi w razie kontaktu. Wykonanie metody *Effect* powoduje dodanie tej wartości do zdrowia gracza.

klasa Exception: klasa obsługująca wyjątki

klasa FileLoadingError: klasa obsługująca wyjątek błędu wczytywania pliku

Poziomy wczytywane są w klasie Level z plików tekstowych o następującej budowie:

LVL1.txt

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 2 1 1 1 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 1 1 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 3 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 1 1 1 1 1 3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

Plik ten zawiera informacje o budowie poziomu. Numery 1-4 w jednoznaczny sposób odpowiadają różnym typom pól: 0 – brak platformy 1 – środek platformy, 2 – lewa krawędź platformy, 3 – prawa krawędź platformy, 4 – podłoże.

W podobny sposób skonstruowane są pliki zawierające informację o przeciwnikach (dla poziomu 1 E_LVL1.txt) oraz przedmiotach (dla poziomu 1 I_LVL1.txt).

4. Testowanie i uruchamianie

Testowanie kodu odbywało się na bieżąco podczas tworzenia programu. Po napisaniu małych fragmentów od razu starałem się sprawdzić działanie tych fragmentów kodu. Było to możliwe również dzięki kolejności, w jakiej tworzyłem kolejne części projektu, a kiedy nie było to możliwe starałem się tworzyć zastępcze, uproszczone moduły dla sprawdzenia działania kodu.
Podczas tworzenia projektu napotkałem na kilka problemów. Jednym z większych było poprawne wyświetlenie animacji postaci. Pierwotnie zmiana tekstury wykonywała była z każdym obiegiem głównej pętli przez co animacja była o wiele za szybka. Problem ten rozwiązałem tworząc metodę odpowiedzialną za aktualizację animacji, która wywoływana jest co pewien regularny odstęp czasu.

Kolejnym problemem było poprawne wyświetlenie mapy. Problemem było przypadkowe zamienienie iteratorów pętli, co było trudne do zauważenia.

Również wykrywanie kolizji okazało się problematyczne, jednak udało mi się ją zaimplementować tworząc niewielkie prostokąty na krawędziach sprite'a, których nakładanie się z innymi obiektami można łatwo wykryć. W tym przypadku największy problem tkwił w samym opracowaniu metody.

5. Wnioski

Projekt ten był moim pierwszym podejściem do bardziej rozbudowanych projektów. Podczas tworzenia programu poznałem dość rozbudowaną bibliotekę – SFML, którą najprawdopodobniej wykorzystam ponownie w przyszłości.
Praca nad tym projektem spowodowała, że nauczyłem się nowych rzeczy, a wygląd całego projektu znacznie się zmieniał, choć główne założenia pozostały te same.

Przy pisaniu projektu wykorzystałem następujące techniki poznane na zajęciach laboratoryjnych:
- dziedziczenie
- dziedziczenie wielobazowe (klasy Character, Item)
- metody wirtualne (klasa Item)
- wyjątki, strumienie (wykorzystane przy wczytywaniu danych z plików)

Pliki projektu można pobrać pod adresem:
https://www.dropbox.com/s/pzh78fd5ggzt4i7/PROJEKT.rar