

Patryk Szela
DRAGON ASCII SLAYER
Projekt zaliczeniowy - Prosta gra komputerowa
Języki Programowania Obiektowego

Wstęp: Dokument dotyczy projektu z przedmiotu Języki Programowania Obiektowego, którym w moim przypadku jest prosta gra komputerowa. Grę nazwałem „Dragon ASCII Slayer”, z powodu rodzaju gry i głównego zadania, które należy wykonać. Dokumentacja została podzielona na 4 części – są to kolejno: założenia projektu, opis samej gry, diagram klas, oraz przyszłe możliwości rozwoju gry. Dokumentacja projektu to nie tylko ten plik, w samym kodzie znajdziemy również masę komentarzy czy dopisów, które również są swego rodzaju opisem.

1. Wymagania sprzętowe i podstawowe założenia projektu

1.1. Temat projektu

Jako temat projektu z języka programowania obiektowego C++ wybrałem prostą grę komputerową. Jest to gra typu Roguelike, czyli rodzaj gry komputerowej, która z założenia ma być podobna do Rogue (Fig.1). Zdecydowałem się wybrać taki temat, ponieważ chciałem napisać grę komputerową, w której będzie istniała możliwość sterowania jakąś postacią w grze, oraz możliwość interakcji z graczem i przeciwnikami. Nieskomplikowana gra, w której rozgrywka toczy się zwykle w trybie turowym, a prezentacja świata i przedmiotów, w postaci znaków ASCII w trybie tekstowym, wydała mi się idealnym tematem.



Fig. 1 – Zdjęcie z gry Rogue

1.2. Zakres projektu

Głównym założeniem projektu było stworzenie tekstowej gry komputerowej, z wykorzystaniem poznanych na laboratorium technik projektowania i programowania obiektowego. Do napisania gry wykorzystałem klasy i standardową bibliotekę szablonów języka C++ (STL). Projekt obejmuje wczytanie świata i bestiariusza z plików tekstowych, wyświetlanie tablicy znaków na ekranie, w której zapisany jest nasz świat, oraz różnego rodzaju funkcje potrzebne do poruszania się naszej postaci i walki z przeciwnikami. Każda funkcja w projekcie została opisana i wiadomo za co jest odpowiedzialna, również wewnątrz niektórych funkcji starałem się dodać własne komentarza, ułatwiające zrozumienie ich działania.

1.3. Opis platformy testowej

System operacyjny platformy, na której przeprowadzane były testy to Windows 8.1
Microsoft Visual Studio Community 2017 Wersja 15.4.4

2. Opis gry i jej działania

2.1. Opis działania aplikacji

Napisana aplikacja, po włączeniu wyświetla menu gry (Fig.2). Stworzony w ten sposób interfejs użytkownika, pozwala na zagłębienie do instrukcji, czy bestiariusza w celu lepszego przygotowania się do rozgrywki, lub rozpoczęcie gry. Po wybraniu opcji „Start Game” możemy wybrać świat, w którym chcemy zacząć swoją rozgrywkę. Każdy świat ma swój poziom trudności (Fig.3). Po wybraniu miejsca, zobaczymy krótkie wprowadzenie do gry, po czym zostanie wczytany dany świat i będziemy mogli zacząć rozgrywkę (Fig. 4). Po pokonaniu wszystkich przeciwników, bądź też w przypadku naszej śmierci, zobaczymy napisy końcowe i gra zakończy się.

```
DRAGON ASCII SLAYER
1. Start Game
2. Instruction
3. Bestiary
4. Exit
```

Fig.2 – wygląd menu gry

```
Choose level:
1. Forest - Easy
2. City - Medium
3. Castle - Hard
4. Dragon's cave - Impossible
```

Fig.3 – możliwe światy do wyboru i ich poziomy trudności

```
#####
#C..#...g.....B.....g.#
#...#...g###.....#...###
#...g...g...#BB#...#...B..#
#g...g...#g.....g...###..#
#####.g...#
#D.#S..#g...B...#...###
#...###..#...g...#...#...g.#
#...B...B...#...S...#
#####
End game <q>, Player's stats <i>, Enter a move command <w/s/a/d>:
```

Fig.4 – wygląd jednego ze światów

2.3. Opis realizacji

Najtrudniejszym zadaniem w całym projekcie, było zdecydowanie, stworzenie świata i przetwarzanie go, oraz połączenie tego z poruszaniem się naszej postaci i walką z przeciwnikami. Widać to szczególnie po rozmiarze klasy Level i po ilości i złożoności funkcji należących do tej klasy. Jest to najważniejsza część projektu. Zawiera w sobie rozwiązania na wszystkie te najtrudniejsze i wyżej wymienione elementy gry. Pozostała część projektu odpowiedzialna jest za stworzenie i inicjalizację naszej postaci, czy też przeciwników, oraz pobieranie tekstu z pliku. W projekcie znajdziemy również klasę GameSystem. Jest to klasa zestawiająca całą grę. Zbiera ona najważniejsze funkcję z innych klas, które odpowiednio ze sobą połączone, składają się na takie rzeczy jak ruch postaci czy interfejs użytkownika. Jest to też pewnego rodzaju, zabieg estetyczny, dzięki dodatkowej klasie mamy porządek w całym projekcie, a w jego głównym pliku mamy tylko jeden konstruktor.

2.4. Scenariusz

W średniowiecznym królestwie pojawia się smok. Nad całą krainą zapanował mrok, smok jest powodem całego zła i źródłem potworów, które co dzień straszą mieszkańców. Nasza postać jest rycerzem, która dostaje od króla zadanie – zabić smoka. Jako człowiek honoru i bohater w oczach ludzi zgadzamy się i ruszamy w długą podróż do jaskini smoka. Legendy mówią, że smok posiada masę skarbów z poprzednich, zniszczonych królestw, oraz że po jego śmierci, zabójca smoka otrzyma całą jego moc. Na swojej drodze nasza postać musi pokonać wszystkie potwory, odpowiednio się wzmocnić i stanąć do walki ze smokiem.

2.5. Podręcznik użytkownika

Sama rozgrywka nie jest skomplikowana. Za ruch naszej postaci odpowiadają 4 klawisze: 'w' – ruch w górę, 's' – ruch w dół, 'd' – ruch w prawo, 'a' – ruch w lewo. Ponadto, możemy wybrać z klawiatury klawisz 'i', by wyświetlić aktualną siłę i poziom życia naszej postaci. W celu przerwania gry, możemy wcisnąć 'q' i potwierdzając swój wybór, zakończyć rozgrywkę. Do instrukcji mamy również dostęp podczas gry. W momencie uruchomienia aplikacji, z poziomu menu, możemy wybrać „Instruction” i przeczytać o sterowaniu w grze.

2.6. Oznaczenia

Każdy postać w grze została opisana inną literką. Częścią świata są też ściany i im także został przypisany konkretny znak. Lista oznaczeń dla naszej gry znajduje się poniżej:

- @ - Gracz
- g - Goblin
- B - Bandyta
- S - Wąż
- O - Ogr
- D – Smok
- # - ściana – nie można przez nią przechodzić

3. Projekt techniczny – diagram klas

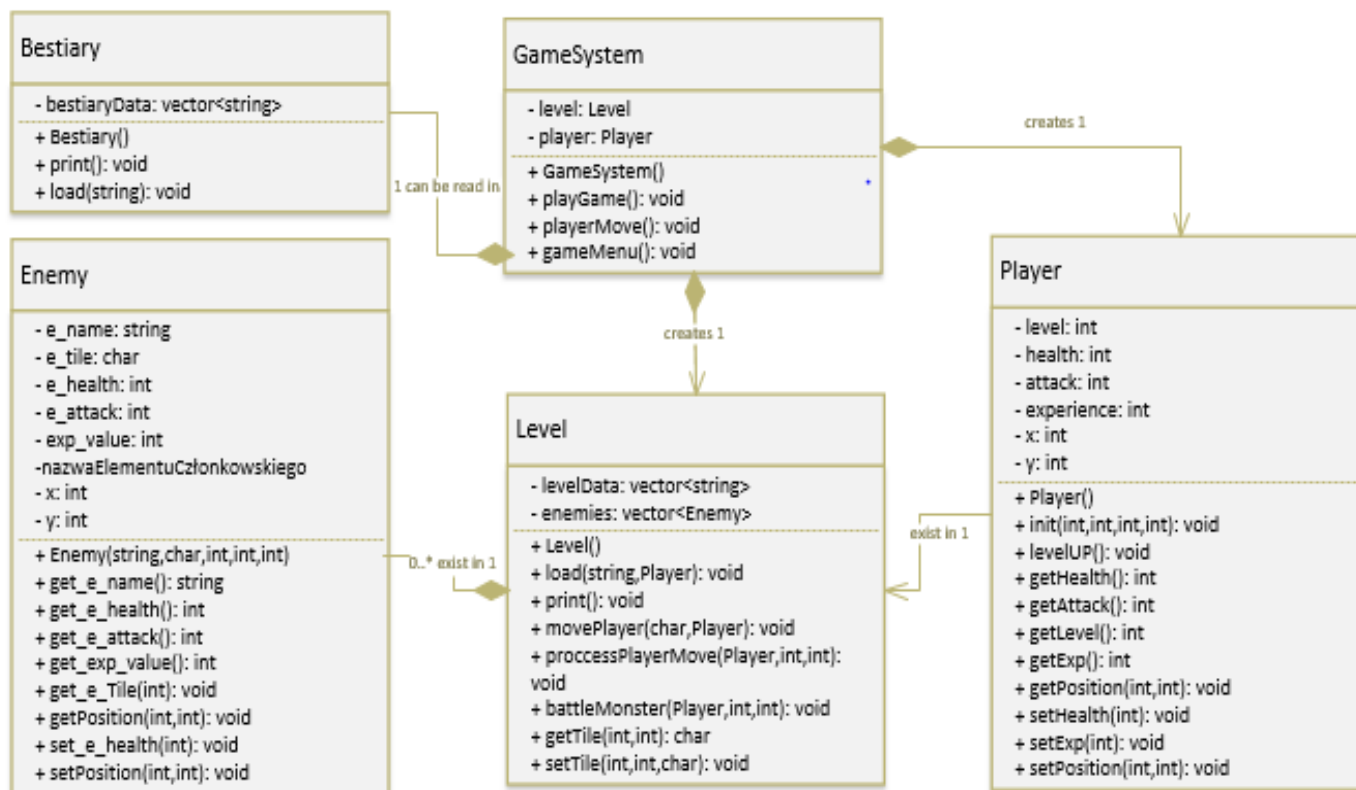


Fig.5 – diagram klas całego projektu

Jak wspomniałem wcześniej, klasą mającą największe znaczenie w projekcie, jest klasa Level. Jednakże, to klasa GameSystem odpowiada za zestawienie całej gry. Bez klasy GameSystem nie zostałby stworzony żaden obiekt, nie istniałyby reszta klas, oraz nie moglibyśmy wykorzystać funkcji w nich się znajdujących. Bez tej klasy, nasza gra nie byłaby w stanie działać. Taką zależność nazywamy kompozycją.

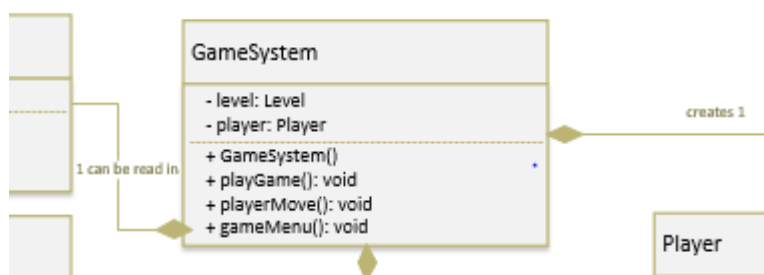
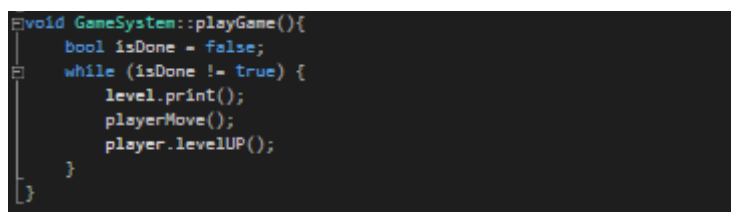


Fig.6 – połączenia dochodzące do klasy GameSystem

Kompozycji używamy wtedy, gdy obiekt „dziecko” nie może istnieć bez obiektu „rodzica”, części należą do jednej całości, a ich okres życia jest wspólny – razem z całością niszczone są również części. W naszym przypadku, całością jest klasa GameSystem, a częściami – klasy Level, Player i Bestiary. Na diagramie klas kompozycję możemy rozpoznać przez linie mające na swoich końcach wypełniony romb (Fig.6).

Tak samo jest w przypadku obiektów klasy Level i Enemy. W kodzie klasy Level, korzystamy z funkcji load(), wczytujemy nią świat z pliku i później go przetwarzamy. W przypadku natrafienia w pliku na znak odpowiedzialny za potwora, wywoływany jest konstruktor parametryczny klasy Enemy i zostaje stworzony jej obiekt. Jak widać, to dzięki klasie Level zostają skonstruowane obiekty klasy Enemy, w żadnym innym miejscu w grze świat nie jest przetwarzany, w żadnym innym miejscu w grze nie tworzymy obiektów klasy Enemy. Jest to kolejny przykład kompozycji i dlatego te klasy również połączone są ze sobą linią z wypełnionym na końcu rombem.

Kolejną zależnością jaką powiązane są ze sobą wybrane klasy, to asocjacja. Asocjacja wskazuje na trwałe połączenie pomiędzy obiektami danych klas i na diagramie oznaczona jest poprzez linię zakończoną strzałką. Taką relacją połączone są klasy Player i Level, oraz tworząca je klasa GameSystem. Po uruchomieniu aplikacji i stworzeniu obiektów wyżej wymienionych klas, są one ze sobą ściśle powiązane aż do samego końca rozgrywki. Przykładowo, funkcja playGame() z klasy GameSystem przez cały czas trwania programu, korzysta z funkcji print() z klasy Level oraz z funkcji levelUP() z klasy Player, co można zobaczyć na zdjęciu fig. 7.



```
void GameSystem::playGame(){
    bool isDone = false;
    while (isDone != true) {
        level.print();
        playerMove();
        player.levelUP();
    }
}
```

Fig.7 – wykorzystanie funkcji innych klas przez funkcję klasy GameSystem

Jest to dosyć oczywista zależność. W klasie, która odpowiada za zestawienie gry muszą pojawić się funkcje odpowiedzialne za ruch postaci, wyświetlanie świata i aktualizowanie go, czy też ciągle sprawdzanie, czy zebrane dotychczas przez naszą postać doświadczenie nie pozwala na zdobycie wyższego poziomu, dodatkowego życia i siły ataku. Podobnie ma się związek funkcji klasy Player z funkcjami klasy Level. Metody odpowiedzialne za ruch postaci, czy też walkę z przeciwnikami, potrzebują wiedzieć w jakim miejscu znajdują się nasza postać, gdyż w swoim działaniu wykorzystują jej aktualne położenie.

4. Dalsze możliwe kierunki rozwoju projektu

Jedną z możliwości rozwinięcia projektu jest na pewno stworzenie „inteligentnych” przeciwników. Nasze potwory mogłyby poruszać się w losowych kierunkach, a po zbliżeniu się do naszej postaci i po zauważeniu jej, mogłyby one zacząć ją ścigać i atakować. W aktualnym projekcie, nasza postać zawsze atakuje pierwsza, więc na pewno można by było dokonać zmiany systemu walki. Ponadto, w grę można stworzyć dodatkowe bronie i przedmioty do zbierania, które później można by było wykorzystać w sklepie.