Studenckie RPG - Dokumentacja Projektu

Antoni Przybylik
PROI 23L
GRUPA 103

Zoja Hordyńska
PROI 23L
GRUPA 103

10 kwietnia 2023

Cel i opis projektu

Głównym celem projektu jest zademonstrowanie umiejętności z zakresu programowania obiektowego, znajomości języka C++, jego standardowej biblioteki, umiejętności inżynierskich związanych z projektowaniem złożonych aplikacji i umiejętności społecznych związanych z pracą w zespole zdobytych na przedmiocie „Podstawy Informatyki i Programowania" (PROI), a także pogłębienie i utrwalenie tej wiedzy i tych umiejętności. Dodatkowo projekt może pomóc jego uczestnikom w lepszym zrozumieniu zasady działania gier komputerowych i zdobyciu wiedzy na temat budowania oprogramowania.

Przedmiotem projektu jest gra „Studenckie RPG". Jest to RPG do którego stworzenia inspiracją jest gra Diablo.

Podczas rozgrywki, gracz wcielający się w studenta przechodzi przez mapę Politechniki, zbierając przedmioty i walcząc z potworami godnymi studiów. Każdy gracz posiada pewne umiejętności, zdrowie i manę. Podczas walki, gracz rzuca umiejętności, które zadają losowe obrażenia z danego przedziału (obrażenia mogą być zwiększone przez niektóre przedmioty i umiejętności). Celem gry jest zabicie wszystkich bossów i tym samym przeżycie studiów.

Będzie to gra 2D z widokiem z boku. Świat będzie zbudowany z bloków (tak jak Minecrafcie). Podobne mechanizmy gry możemy znaleźć w grach takich jak Paper Minecraft, albo Epic Ninja na Scratchu.

Projekt ma zostać zrealizowany w języku C++ z użyciem jego zaawansowanych mechanizmów i ze szczególnym uwzględnieniem paradygmatu obiektowego. Planowane jest stworzenie dedykowanego silnika gry zaimplementowanego przy użyciu biblioteki SFML, który będzie tworzył warstwę abstrakcji między wysokopoziomowym kodem właściwej gry operującym na „duszkach", a interfejsem biblioteki SFML pozwalającym na proste operacje takie jak wyświetlanie obrazów.

Architektura

Projekt jest podzielony na cztery moduły:

- Silnik gry
- Ciało gry
- Launcher
- Aplikacja gry

Silnik gry dostarcza interfejs zaprojektowany w paradygmacie obiektowym. Jest on oparty o model „Sprite'owy". Użytkownik tworzy „duszki" (ang. sprite) i określa ich interakcje z otoczeniem. Następnie dodaje duszki do silnika i uruchamia silnik. Silnik symuluje zachowanie duszków podane przez użytkownika. Programowanie obiektowe pozwala na przeładowanie klasy Sprite (duszek) i umieszczenie w nim kodu który zostanie wykonany na odpowiednich zdarzeniach takich jak zderzenie dwóch duszków lub kodu który jest wykonywany w każdym tyknięciu zegara.

```
        /* Utworzenie duszka. */
        skin = new SpriteSkin(*tiles, 3, 500, 1);
        sprite = new Sprite(skin,
                            Rect(0, 0, win_x, win_y),
                            BA_BOUNCE);
        sprite->set_position(Rect(100, 100, 75, 75));
        sprite->set_velocity(Vector(120, 250));

        /* Silnik. */
        engine = new Engine();
        engine->bind_window(window);
        engine->add_sprite(sprite);

        engine->exec();
```

Listing 1: Interfejs silnika - kod poglądowy

Działanie tego kodu jest zademonstrowane na filmie.

Ciało gry to kod implementujący właściwą rozgrywkę. To w nim znajdą się klasy takie jak: Player, Enemy, Spell, czy Item.

Klasa Player przechowuje stan gracza: liczbę punktów zdrowia, manę, moc i ekwipunek. Metody klasy będą pozwalać na zarządzanie ekwipunkiem i modyfikację parametrów gracza. TODO: Dziedziczy po Sprite.

Klasa Enemy jest klasą bazową dla wrogów. Jej interfejs powinien pozwalać na walkę niezależnie od tego, z jakim przeciwnikiem mamy do czynienia.

Klasa Spell jest pomniejszą klasą bazową dostarczająca interfejs czaru. Wszelkie parametry czaru są wirtualne, a interfejs ma być taki sam dla wszystkich klas pochodnych.

Item jest pomniejszą klasą bazową implementującą interfejs przedmiotu.

```
class Sword : public Item {
public:
        virtual Sword(void);
        virtual ~Sword(void);

        virtual void action(void);
};
```

Listing 2: Klasa „Sword” - kod poglądowy

Launcher jest programem służącym do uruchomienia gry. Można w nim modyfikować ustawienia rozgrywki. Zakładane możliwości dostosowania ustawień rozgrywki obejmują: zmianę mapy, wczytanie stanu gry z pliku. <TODO>

Aplikacja gry jest programem który służy do grania w grę. W oknie aplikacji gry jest renderowana symulacja gry. Aplikacja gry powinna dodatkowo pozwalać na zatrzymanie gry, zmianę ustawień i wyjście z gry. <TODO: Napisz coś o aplikacji gry.>

Wymagania

Gra wymaga zainstalowanej biblioteki SFML. Założone jest działanie gry na systemach operacyjnych Windows i Linux. Opcjonalnie gra mogłaby wspierać system MacOS. Jednak nie mam tego systemu operacyjnego i nie zobowiązuję się że gra będzie na nim działać.

Instrukcja użycia

Należy uruchomić launcher. W launcherze wybrać preferowane ustawienia i rozpocząć grę przyciskiem START. Wtedy pojawi się nowe okno aplikacji gry.

Biblioteki i Narzędzia

Projekt w całości ma zostać zrealizowany w języku C++, w standardzie C++20. Dopuszczalne jest używanie rozszerzeń GCC.

Skrypty pozwalające na zbudowanie projektu mają być kompatybilne z programem make i do zbudowania projektu ma zostać użyty toolchain GCC. Do kompilacji gry na Windowsa zostanie użyty MinGW.

Projekt ma zostać zrealizowany przy użyciu biblioteki SFML, biblioteki standardowej języka C++ (libstdc++), biblioteki standardowej języka C (libc) i biblioteki matematycznej (libm). Poza nimi, nie jest planowane użycie żadnych dodatkowych bibliotek.

W projekcie nie jest wymagane ścisłe trzymanie się kanonicznej wersji tych bibliotek. Dopuszczalne jest użycie rozszerzeń GLIBC.

Podział Pracy

Antoni

- Silnik
- Wybrane części gry

Zoja

- Menu
- Interfejs gry
- Tekstury i obrazki
- Ciało gry; Klasy: Player, Enemy, Spell, Item