Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i informatyki

Programowanie Komputerów

Gra RPG

Autor	Rafał Grzelec
Prowadzący	mgr inż. Piotr Pecka
Rok akademicki	2015/2016
Kierunek	informatyka
Rodzaj studiów	SSI
Semestr	IV
Termin laboratorium	wtorek, 15:30 – 17:00
Grupa	3
Data oddania sprawozdania	14.06.2016r

1 Treść zadania

Napisać program umożliwiający strategiczną grę oraz tworzenie i zarządzanie postaciami.

2 Analiza zadania

Przedstawione zagadnienie opisuje problem przechowywania struktur opisujących postacie ich odczyt, zapis, wykorzystanie w praktyce oraz tworzenie GUI aplikacji. W programie została wykorzystana biblioteka graficzna SFML-2.3.2.

2.1 Struktury danych

W programie wykorzystano kontenery pochodzące z biblioteki stl. Do przechowywania wszelkiego rodzaju ekwipunku, grupowania drużyny i przycisków tworzących graficzny interfejs użyto wektorów. Mapa został wykorzystana do przypisania wskaźnika na konkretną część ubioru z jej typem (głowa, ręce, nogi …). Taka struktura danych ułatwia późniejszą prace na nich, oraz nie ogranicza programu w ilości dodawanych przedmiotów.

2.2 Algorytmy

Działanie programu opiera się na wykonywaniu odpowiednich czynności zadanych przez użytkownika. Gracz może podjąć decyzje o dalszym działaniu programu po przez wciśnięcie odpowiedniego przycisku, który wywoła odpowiadający mu event.

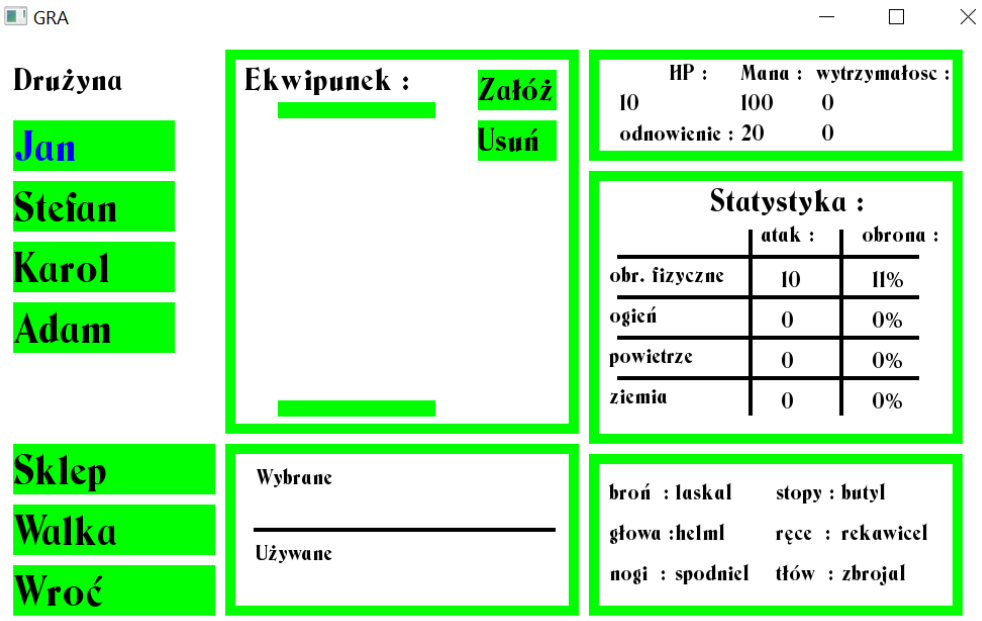
3 Specyfikacja zewnętrzna

Program po uruchomieniu wyświetla menu główne

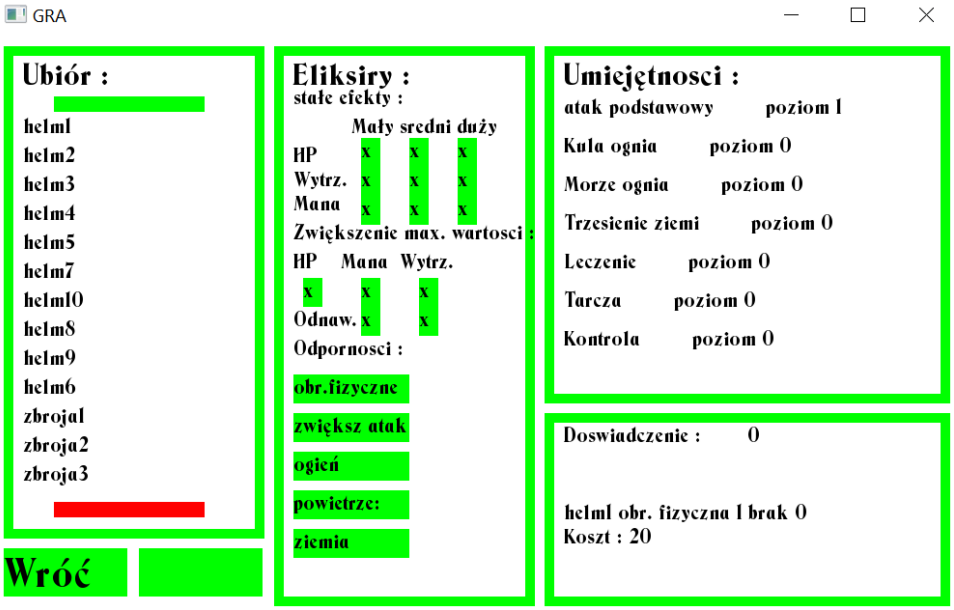
W przypadku nie odnalezienia plików zapisu stanu gry przycisk kontynuuj jest nieaktywny.

Menu „Nowa gra” umożliwia stworzenie swojej drużyny. Dostępne opcje to wybór klasy oraz imienia postaci.

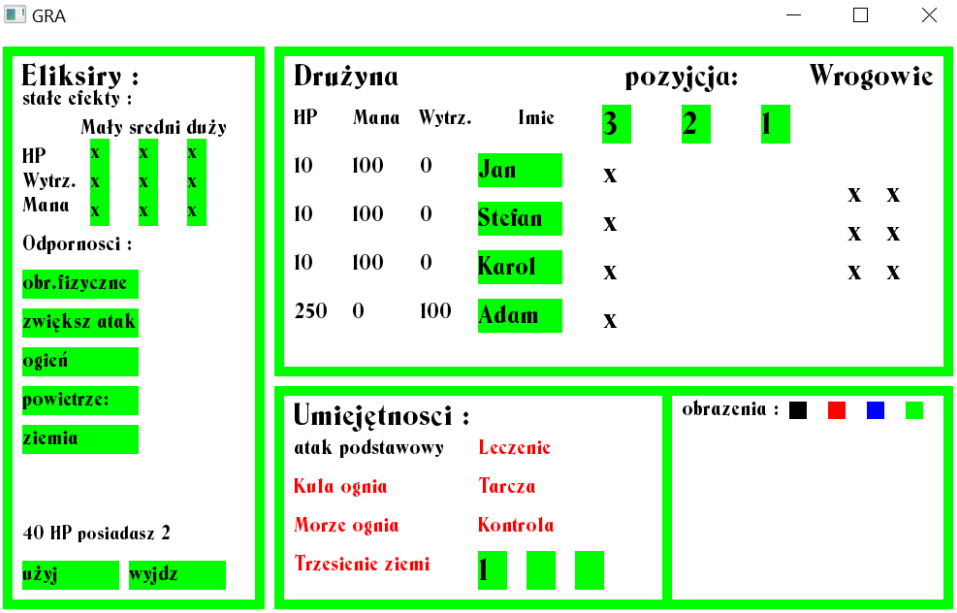
Menu kontynuuj pozwala zarządzać już utworzoną drużyną. Dostępne opcje to : wybór ekwipunku, porównanie wybranego przedmiotu z już założonym, podgląd statystyk wybranej postaci. W ramach danego menu istnieje możliwość przejścia do menu Sklepu bądź Walki, jak również powrotu do poprzedniego menu.



Menu sklep pozwala awansować postać , bądź zaopatrzyć ją w nowy ekwipunek.



Menu walki



Rozgrywka polega na turowym zadawaniu obrażeń przeciwnej drużynie, do momentu w którym któraś z drużyn nie polegnie. Aby zaatakować należy wybrać przeciwnika, umiejętność w momencie wyboru poziomu następuje atak. Istnieje możliwość zmiany pozycji postaci, powoduje ona w zależności od odległości od przeciwnika zmniejszenie otrzymanych obrażeń. Niestety sprawia ona że ataki oddalonej postaci są słabsze.

4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z wymaganiami podanymi przez prowadzącego laboratoria. Rozdzielono również interfejs od logiki programu.

4.1 Typy zdefiniowane w programie

W programie zdefiniowano następujące typy/klasy :

```
enum typ_przycisku
{
	nawigacja,
	osoba,
	przedmiot,
	napis,
	slot,
	pole_tekstowe,
	ramka,
	lista,
	zarzadzanie_ekwipunkiem,
	mikstura,
	umiejet,
	pozycja,
	wrog,
	staty,
	przycisk_pozyc,
	atakuj
};

Klasa tworząca przycisk bez obramowania
class buton
Klasa tworząca przycisk z obramowaniem
class button_border
Klasy tworzące GUI, oraz zarządzająca pewnymi jego aspektami nie wymagającymi
ingerencji użytkownika.
class menu
class nowa_gra
class menu_postaci
class sklep
class walka
Klasa wykorzystywana do przechowująca informacje o zadanych, przyjętych obrażeniach
oraz statystyki postaci
class obrażenia
Klasa reprezentująca efekt nadany postaci
class efekt
Klasy reprezentujące przedmioty w grze
class przedmioty
class ubior
class eliksir
Klasa przechowująca informacje o poziomie i koszcie umiejętności
class umiejetnosc
Klasy reprezentujące postacie, posiadające całą paletę funkcji pozwalających nimi
zarządzać
class postac
class Mag
class Wojownik
class Lucznik
class Kaplan
class Lotr
```

class Druid

4.2 Ogólna struktura programu

Przed rozpoczęciem właściwej pracy algorytmu wyczytywany jest zapis postępów z pliku postepy.sav zapisanego na dysku. Ogólna logika programu polega na wywołaniu funkcji wyświetlającej menu oraz zwracającej przycisk wybranej opcji. Pozwala to przejść do dalszej części wykonywanych czynności wybranych przez użytkownika. Tuz przed zakończeniem pracy programu rozpoczyna się zapis postępów do pliku, oraz wyczyszczenie pamięci dynamicznej, zaalokowanej w trakcie pracy algorytmu.

4.3 Szczegółowy opis implementacji funkcji

```
Funkcja zapełniająca bazę przedmiotów i eliksirów
void inicjalizacja(vector<ubior> & baza_przedmiotow, vector<eliksir> & baza_eliksirow)

Funkcja pozwalająca użyć eliksiru na konkretnej postaci
Pozycja – indeks przedmiotu w wektorze który ma być użyty
Na_kim – wskaźnik do postaci na której ma być użyty eliksir
Baza_eliksirow – vektor z eliksirami
string uzyj(int pozycja, postac * na_kim, vector<eliksir> & baza_eliksirow)

Funkcja zapisująca do pliku dane o graczach i poziomie drużyny
bool zapis_do_pliku(int poziom_druzyny, vector<postac*> druzyna, vector<eliksir>
baza_eliksirow)

Funkcja odczytujaca dane z pliku
bool odczyt_z_pliku(int & poziom_druzyny, vector<postac*> &druzyna, vector<ubior>
baza_przedmiotow, vector<eliksir> & baza_eliksirow)

Funkcja sprawdzająca czy pozycja myszy o współrzędnych x, y nie
należy do któregoś z przycisków umieszczonych w wektorze typu podanego w klasie
szablonu i zwracająca wskaźnik na najechany przycisk
template <class T>
T * znajdz_przycisk(int x, int y, vector<T*> przyciski)

Funkcja ładująca listę ze środowiska graficznego
void zaladuj_ekwipunek(vector<ubior> & ekwipunek, vector<button_border*> & lista, int
poczatek, int koniec, int pierwszy_element)


Zwolnienie pamięci każdej z postaci w wektorze
void czysc_postac(vector<postac*> & druzyna)

Zwalnianie pamięci przycisków o klasie podanej w szablonie, w menu (klasa menu podana
jako drugi argument szablonu
template<class T, class T2>
void czysc_menu(T & menu)

Sprawdzenie czy wszystkie postacie wykonały już swój ruch
bool koniec_tury(vector<postac*> osoby)

Sprawdzenie działających efektów każdej osoby z drużyny
void spr_efekty(vector<postac*> osoby)

Aktywacja postaci z danej drużyny
void aktywuj_postacie(vector<postac*> osoby)

Przyznanie doświadczenia wszystkim postaciom z listy
```

```
void nagroda(vector<postac*> osoby)
```
Metody różnych klas menu:
Funkcja dodająca przyciski danego menu do okna podanego jako argument.
```
void draw(sf::RenderWindow & okno)
```
Funkcja wykonująca działania nie wymagająca bezpośredniej ingerencji użytkownika (reakcji po wciśnięcia przycisku) takich jak wypełnienie statystyk, list z ekwipunkiem itd.
```
void zaladuj_postac(vector<postac*> & druzyna, vector<postac*> & wrogowie,
vector<ubior> & baza_przedmiotow, vector<eliksir> & baza_eliksirow)
```

Metody klasy obrażenia:
Dodaje obrażenia danego typu
```
void dodaj(string gdzie, double ile)
```
Zwraca łączną ilość zadanych obrażeń
```
double laczny()
```

Operator mnożenia (każdy typ obrażeń pomnożony o daną zmienną
```
obrazenia operator*(double czynnik)
```

Metody klasy Umiejetnosc:
Inkrementuje wartość zmiennej poziom, w przypadku osiągnięcia wartości większej niż 3 brak działania.
```
void podnies_poziom()
```

Zwraca koszt użycia danej umiejętności
```
int getKoszt(int poziom_)
```

Metody klas postaci:

Założenie konkretnej części ekwipunku, o podanym indeksie
```
string zaluz(int)
```

Usunięcie rzeczy z ekwipunku, o podanym indeksie
```
string usun(int)
```

Sprawdzenie wektora działających efektów i podjęcie odpowiednich działań dla każdego ze znalezionych efektów
```
void sprawdz_efekty()
```

Odebranie obrażeń pomniejszonych o posiadana odporność i zwrócenie otrzymanych w ten sposób obrażeń
```
obrazenia odbierz_obrazenia(obrazenia zadane)
```

Funkcja pozwalająca użyć wybranej umiejętności
```
virtual vector<string> uzyj_umiejetnosci(int numer_umiejetnosci, vector<postac*>
sojusznicy, vector<postac*> wrogowie, double poziom, int na_kim)
```

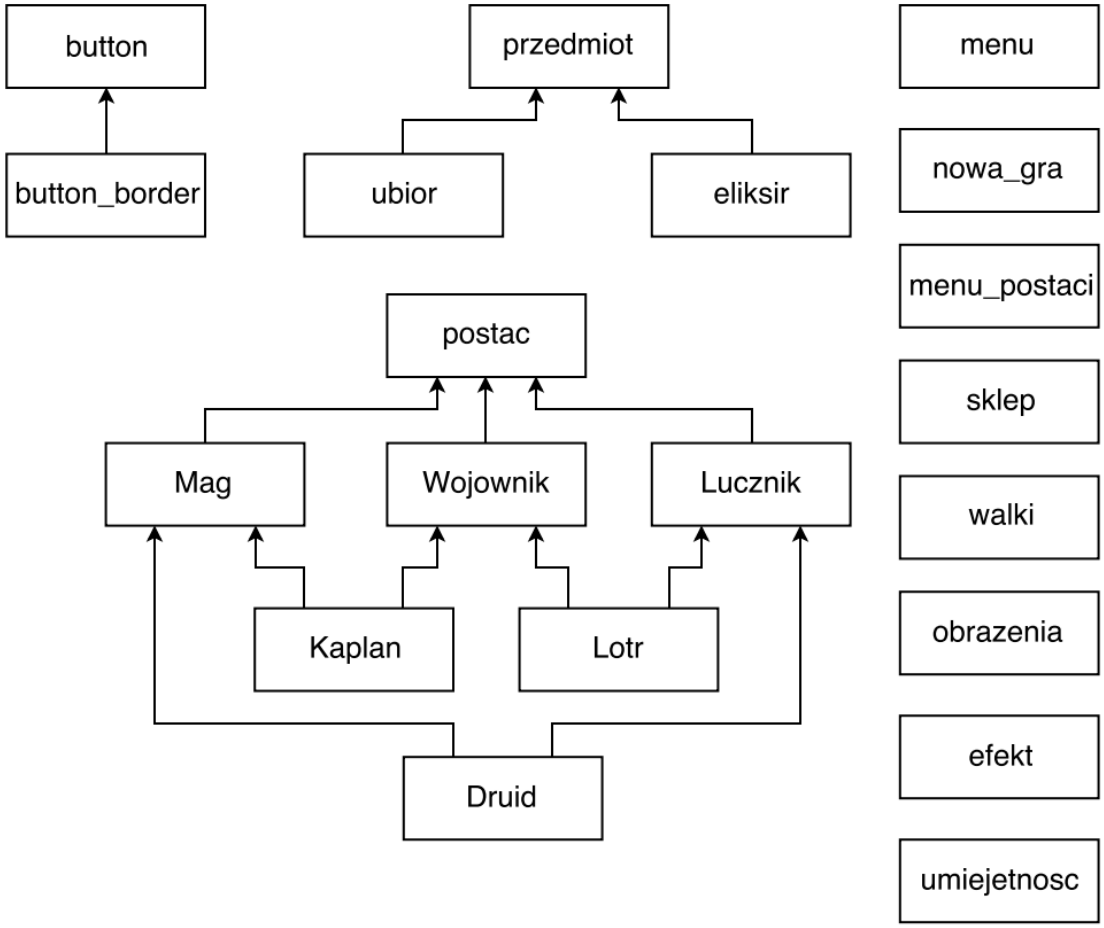
Podstawowa funkcja do zadawania obrażeń, i zwracająca napis z informacjami o zadanych obrażeniach
```
string atak(postac * przeciwnik, obrazenia przeprowadzony_atak)
```

Funkcja testowa
```
void wypisz()
```

Każda z klas postaci posiada zestaw funkcji reprezentujących konkretne umiejętności ataku, są one używane przez funkcje „uzyj_umiejetnosci”.

Można zauważyć w wielu klasach metodę `string` wypisz() zwracają one napis z wykonaną przez nich czynnością.

4.4 Diagram klas



5 Testowanie

Program został przetestowany pod względem dodawania i zarządzania postaciami, jak również samą walką. Program prawidłowo zapisuje i odczytuje dany z dysku (w przypadku braku pliku postepy.sav funkcja odczyt_z_pliku nie wykonuje żadnej czynności natomiast funkcja zapis_do_pliku przed zapisem tworzy nowy plik o nazwie postepy.sav

6 Wnioski

Program RPG jest skomplikowany. Największą trudność w jego wykonaniu było poprawne przeprowadzanie ataku oraz zapisanie i odczytanie danych z pliku. Powodem tego była wybrana struktura przechowywania danych w pamięci (występowały niepoprawne odczyty spowodowane różną wielkością obiektów klas dziedziczących), jak również niepoprawne sposób odczytu efektów działających na daną postać.