Politechnika Śląska w Gliwicach

Wydział Automatyki, Elektroniki i Informatyki

Programowanie Komputerów

Projekt własny

---

autor	Patryk Szulc
prowadzący	dr inż. Jacek Szedel
Rok akademicki	2017/2018
Kierunek	Informatyka
Semestr	3
Termin laboratorium	Piątek 13:00-14:30
Grupa	6
Sekcja	1
Data oddania sprawozdania	06.01.2018

Link do githuba z projektem:
https://github.com/yamaz0/Unreal/tree/master/gra/gra

1. Temat

Wybrany temat projektu to gra z animacją. Zadanie nie należy do listy zadań semestralnych.

Wymagania projektu:

- Dziedziczenie
- Polimorfizm
- Operatory
- Strumienie i pliki
- Zaawansowane elementy c++ (dodatkowo)

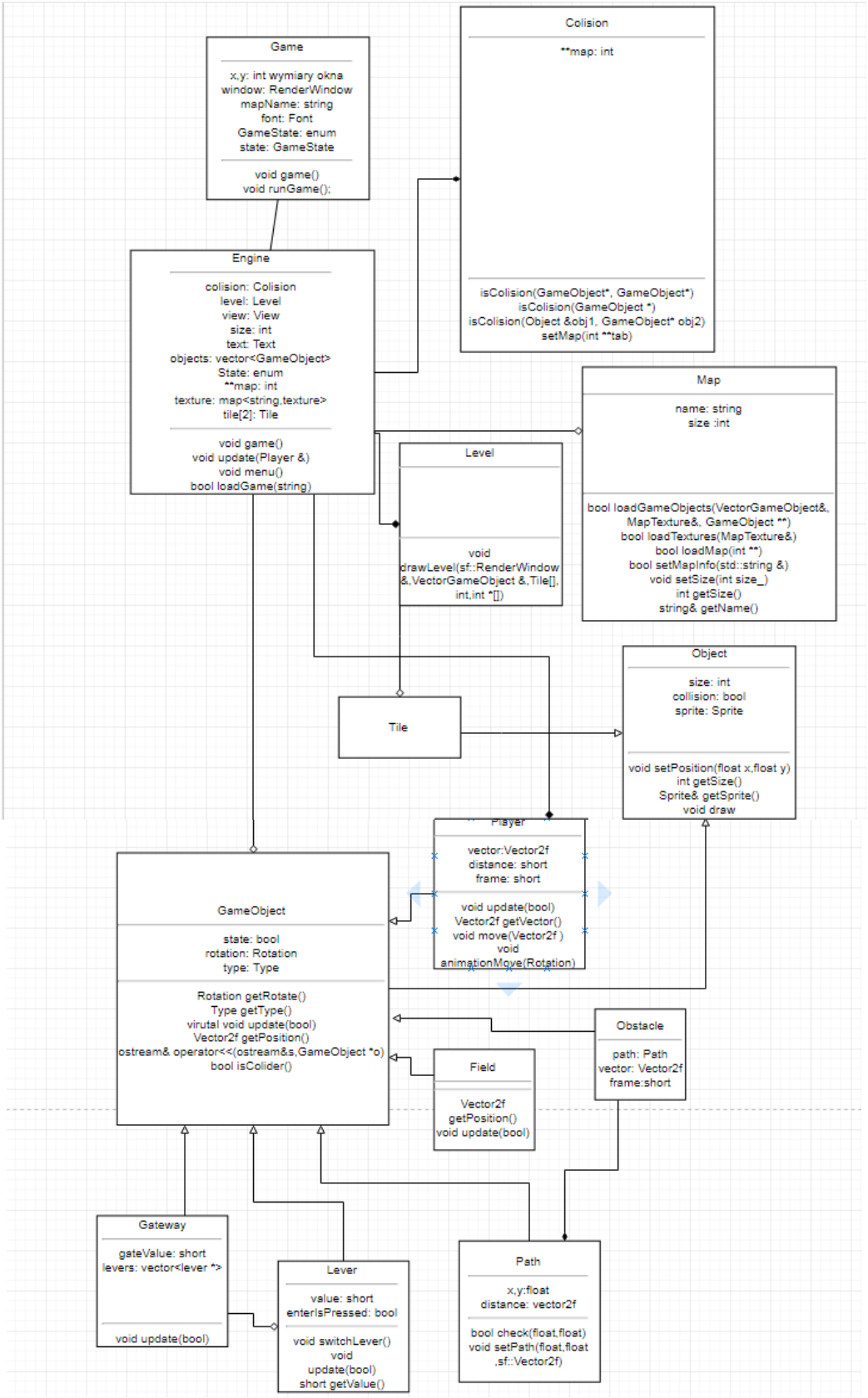
2. Analiza i projektowanie

Postęp tworzenia projektu miał być omawiany z prowadzącym laboratorium co każde zajęcia. Zaczeliśmy od specyfikacji wstępnej, drugim krokiem był diagram klas UML(zamieszczony niżej). Następnie mieliśmy zacząć deklarować klasy i przygotować propozycję metod, a na końcu to wszystko zaimplementować do projektu.

Projekt w przyszłości chcę trochę rozwinąć, dlatego pojawiają się w nim elementy, które są tam niepotrzebne lub puste(edytor map) lecz zostały umieszczone z myślą o przyszłych zmianach.

Gra jest 2D z widokiem z góry ale niektóre obiekty widzi się z boku. Istnieje bardzo uboga animacja poruszania się postaci i obiektów. Grafika w grze jest stworzona w programie GIMP i jest bardzo prymitywna, ponieważ nie jest ona ważna w projekcie.

Diagram UML


Game
x,y: int wymiary okna
window: RenderWindow
mapName: string
font: Font
GameState: enum
state: GameState
void game()
void runGame();
Colision
**map: int
isColision(GameObject*, GameObject*)
isColision(GameObject *)
isColision(Object &obj1, GameObject* obj2)
setMap(int **tab)
Engine
colision: Colision
level: Level
view: View
size: int
text: Text
objects: vector<GameObject>
State: enum
**map: int
texture: map<string,texture>
tile[2]: Tile
void game()
void update(Player &)
void menu()
bool loadGame(string)
Map
name: string
size :int
bool loadGameObjects(VectorGameObject&, MapTexture&, GameObject **)
bool loadTextures(MapTexture&)
bool loadMap(int **)
bool setMapInfo(std::string &)
void setSize(int size_)
int getSize()
string& getName()
Level
void drawLevel(sf::RenderWindow &,VectorGameObject &,Tile[], int,int *[])
Tile
Object
size: int
collision: bool
sprite: Sprite
void setPosition(float x,float y)
int getSize()
Sprite& getSprite()
void draw
Player
vector:Vector2f
distance: short
frame: short
void update(bool)
Vector2f getVector()
void move(Vector2f)
void animationMove(Rotation)
GameObject
state: bool
rotation: Rotation
type: Type
Rotation getRotate()
Type getType()
virutal void update(bool)
Vector2f getPosition()
ostream& operator<<(ostream&s,GameObject *o)
bool isColider()
Obstacle
path: Path
vector: Vector2f
frame:short
Field
Vector2f
getPosition()
void update(bool)
Gateway
gateValue: short
levers: vector<lever *>
void update(bool)
Lever
value: short
enterIsPressed: bool
void switchLever()
void
update(bool)
short getValue()
Path
x,y:float
distance: vector2f
bool check(float,float)
void setPath(float,float ,sf::Vector2f)

3. Specyfikacja zewnętrzna

Opis gry:

Gra polega na otwieraniu dźwigni, aby otworzyć przejścia, by dostać się do punktu wyjścia. Dźwignie trzeba użyć w odpowiedniej kombinacji. Przeszkodami, które należy omijać są piły, które poruszają się po określonej ścieżce tam i z powrotem. Kiedy dotkniemy przeszkody, przenosi nas do ostatniego odwiedzonego checkpointu.

Menu:

Menu główne składa się z 2 części: z menu głównego i menu wyboru map.

Po kliknięciu:

Play wyświetla się menu z wyborem mapy

Map editor nic się nie dzieje

Exit program zamyka się

W menu wyboru mapy możemy poruszać się w lewo i prawo po kolejnych stronach za pomocą strzałek na ekranie. Możemy wrócić do poprzedniego menu naciskając BACK. Po wybraniu mapy załącza się gra.

Sterowanie:

Postacią steruje się za klawiatury.

Strzałka w lewo – postać porusza się w lewo

Strzałka w prawo – postać porusza się w prawo

Strzałka w góre - postać porusza się w górę

Strzałka w dół - postać porusza się w dół.

Enter – używanie dźwigni lub jeśli naciśniemy Escape(pauza) wychodzimy do menu

R – cofa postać do checkpointu

Możliwe jest iść na skos ale przy ścianie postać zatrzymuje się i czeka na ruch w innym kierunku.

Pliki:

Istnieje możliwość dodania własnej mapy. W katalogu Assets/Maps należy dodać do pliku maps.txt nazwe mapy oraz stworzyć katalog z tą samą nazwą. W katalogu mapy trzeba dodać pliki:

Field.txt

Gateway.txt

Info.txt

Lever.txt

Obstacle.txt

map.txt

W pliku Info zapisujemy rozmiar mapy np. 20 (mapa jest kwadratowa) następnie współrzędne wyjścia i kierunek np. SOUTH

W map.txt podajemy mape w postaci 0 i 1 gdzie 0 to podłoga, a 1 to ściana. Należy robić spacje pomiędzy każdą cyfrą.

W pozostałych plikach dodajemy obiekty:

Field X Y ZWROT

Gateway X Y ZWROT ILE_DŹWIGNI ID_DŹWIGNI

Lever X Y ZWROT WARTOSC

Obstacle X Y ZWROT DYSTANS_X DYSTANS_Y PREDKOSC_X PREDKOSC_Y

Przykłady można popatrzeć w innych mapach.

Trzeba wiedzieć, że większość obiektów ma rozmiar 128x128, a np. przeszkoda ma 64x64. W katalogu Maps znajduje się arkusz kalkulacyjny, który pomagał mi przy projektowaniu poziomów.

Program nie ma zabezpieczeń przy błędnych danych.

4. Specyfikacja wewnętrzna

Różne typy i zmienne wykorzystywane w programie:

GameState	typ wyliczeniowy okreslajacy stan gry: MENU,MENU2, GAME, EDITOR, END
`vector<GameObject*>`	Wektor służacy do przechowywania wskaźników na obiekty wszystkich obiektów w grze.
State	typ enumeryczny stanu gry: GAME, MENU, END
**map	Tablica dynamiczna typu int. Przechowuje mapę w postaci 0(podłoga) i 1 (ściana)
Rotation	Typ wyliczeniowy opisujący zwrot obiektu: NORTH, SOUTH, EAST, WEST
Type	Typ wyliczeniowy opisujący typ obiektu: LEVER, GATEWAY, OBSTACLE, PLAYER, END, CHECKPOINT, ACTIVATOR

Klasy:

Game:

```
const int x = 800;
//wysokosc okna
const int y = 600;

//okno programu
sf::RenderWindow window;

//nazwa mapy(potrzebne do wczytania wybranej mapy)
std::string mapName;

//Czcionka
sf::Font font;
```

```
//typ wyliczeniowy okreslajacy stan gry
enum GameState {MENU,MENU2, GAME, EDITOR, END};

//zmienna okreslajaca stan gry
GameState state;

///Metoda odpowiadajaca za pierwsze menu
void menu();

///Metoda odpowiadajaca za drugie menu
void menu2();

///Metoda, ktora steruje gra
void runGame();

///Metoda uruchamia gre
void game();
```

Engine:

```
///główna pętla gry
void game();

///aktualizowanie klatki gry
void update(Player &);

///menu podczas gry
void menu();

///ladowanie gry
bool loadGame(std::string);

//okno programu
sf::RenderWindow &window;

//klasa do sprawdzania kolizji
Colision colision;
```

```
//klasa do wyswietlania mapy
Level level;

//kamera
sf::View view;

//wielkosc mapy
int size;

//klasa do wyswietlania napisow
Text text;

//vektor wszystkich obiektow
std::vector<GameObject*>objects;

//std::vector<GameObject*>levers;//wektor dzwigni

//typ enumeryczny stanu gry
enum State { GAME, MENU, END }state;

//numer klatki aktualnej
short frame = 0;

//pole na ktore sie teleportuje po kolizji z przeszkoda
GameObject* checkpoint;

//tablica dwuwymiarowa reprezentująca mape w postaci 0-podloga 1 - sciana
int **map = nullptr;

//mapa textur
MapTexture textures;

//Tablica kafelek(2) podłogi i ściany.
Tile tile[2] = { Tile(false,textures["background"]),Tile(true,textures["background"]) };
```

Map:

```
//nazwa mapy
std::string name;

//rozmiar mapy
int size;
```

```
///Metoda wczytująca nazwy i rozmiary map z pliku.
bool loadMapsName(VectorString &);

///wczytywanie tetxtur
 bool loadTextures(MapTexture&);

///Metoda wczytująca obiekty z pliku
bool loadGameObjects(VectorGameObject&, MapTexture&, GameObject **);

///Metoda wczytująca mape z pliku do tablicy dwuwymiarowej.
bool loadMap(int **);

///Metoda zapisuje nazwe mapy
bool setMapInfo(std::string &);

///Metoda zapisuje rozmiar mapy
void setSize(int size_) { size = size_; }

///Metoda zwracająca rozmiar mapy.
int getSize()

///Metoda zwracająca nazwę mapy.
std::string& getName()
```

Text:

```
///wyswietlanie napisu na srodku ekranu
void displayText(const std::string &,sf::Vector2f, sf::RenderWindow &);
```

Colision:

```
///Metoda sprawdza kolizje dwóch obiektów
bool isColision(GameObject*, GameObject*);

/// Metoda sprawdza kolizje miedzy player i mapa
bool isColision(GameObject *);

///Metoda ustawia tablice dwuwymiarowa reprezentujaca mape
void setMap(int **tab)

///Metoda sprawdza kolizje Object z GameObject
bool isColision(Object &obj1, GameObject* obj2);
```

Object:

```
int size=128;//128x128 domyslny rozmiar
//czy obiekt jest kolizyjny
bool collision;

//sprite obiektu
sf::Sprite sprite;

 ///metoda ustawia pozycje obiektu
 void setPosition(float , float );

 ///metoda zwraca rozmiar obiektu(jako kwadrat)
 int getSize()

 ///zwraca referencje do sprite
sf::Sprite& getSprite()

///metoda rysujaca obiekt
void draw(sf::RenderTarget& target, sf::RenderStates state) const override;
```

GameObject:

```
/// zwraca w ktorym kierunku jest zwrocony obiekt
Rotation getRotate();

/// zwraca jakiego typu jest obiekt
Type getType()

/// funkcja czysto wirtualna. Aktualizuje stan obiektu.
///Jako parametr jest podany typ bool przekazuj¹cy informacje o kolizji z graczem
virtual void update(bool) = 0;

///zwraca pozycje sprite'a
sf::Vector2f getPosition()

///operator do wyswietlania pozycji
friend std::ostream & operator<<(std::ostream &, GameObject *);

///zwraca czy obiekt jest kolizyjny
bool isColider()

//stan obiektu(aktywny/nieaktywny)
```

bool state;

//kierunek zwrotu
Rotation rotation;

//typ obiektu
const Type type;
Field:
///Metoda pobiera pozycje pola
sf::Vector2f getPosition()

///Metoda aktualizujaca pole
void update(bool);
Obstacle:
///Metoda aktualizujaca obiekt
virtual void update(bool);

///Animowanie obiektu
void animationMove();

Path path;//sciezka

sf::Vector2f vector;//wektor o ktory porusza sie obiekt
Player:
//wektor do poruszania postacia
sf::Vector2f vector;

//odleglosc o ile ma sie przesuwac postac
short distance;

///Metoda aktualizuje obiekt
virtual void update(bool);

///Metoda zwraca wektor ruchu
sf::Vector2f getVector()

/// Metoda przemieszcza obiekt o podany wektor
void move(sf::Vector2f);

/// Animacja ruchu postaci

```
void animationMove(Rotation);
```
Lever:
```
///Przelacza dzwignie, czyli zmienia jej stan
void switchLever();

///aktualizuje stan dzwigni
virtual void update(bool);

///zwraca wartosc dzwigni(potrzebne do otwarcia przejsc)
short getValue()

//id dzwigni
short id=0;

//wartosc dzwigni
short value = 1;

//zmienna pomocnicza ograniczajaca aktywacje dzwigni
bool enterIsPressed = false;
```
Gateway:
```
/wartość która jest zmieniana przez dźwignie.
//jeśli z dźwigni wyjdzi wartość to przejście
//otwiera się.
short gateValue = 0;

//vector referencji do dźwigń
VecLever levers;
```

5. Wnioski

Przed pisaniem programu trzeba najpierw zaplanować jak ma wyglądać, ponieważ nie da się pisać “z głowy” większych programów.

Biblioteka STL jest bardzo pomocna przy pisaniu programu. Ułatwia wiele problemów i jest prosta w obsłudze.