Politechnika Śląska w Gliwicach

Wydział Automatyki, Elektroniki i Informatyki

Programowanie Komputerów

Projekt semestralny

---

autor	Patryk Szulc
prowadzący	Dr inż. Roman Starosolski
Rok akademicki	2017/2018
Kierunek	Informatyka
Semestr	4
Termin laboratorium	Wtorek 13:45-15:15
Grupa	6
Sekcja	2
Data oddania sprawozdania	11.06.2018

1. Temat

Wybrany temat projektu to gra zręcznościowa. Gra polega na przechodzeniu labiryntu pełnego przeszkód i pułapek przy pomocy różnych dźwigni bądź przycisków służących do otwierania przejść.

2. Analiza tematu

Poruszając się korytarzami labiryntu, unikamy zagrożenia i otwieramy przejścia za pomocą dźwigni lub przycisków czasowych. Przeszkodami, które należy omijać są piły, które poruszają się po określonej ścieżce tam i z powrotem oraz kolce, które działają czasowo. Kiedy dotkniemy przeszkody, przenosi nas do ostatniego odwiedzonego checkpointu. Gdy dojdziemy do końca gra wchodzi do menu głównego.

Gra jest stworzona z użyciem biblioteki SFML w języku C++. W programie są użyte klasy vector, map, fstream oraz iostream.

- W vectorze przechowywane są wszystkie obiekty gry.
- Map użyłem do konwersji string na enum oraz do textur.
- Fstream do odczytu z plików.
- Iostream, aby wyświetlać informacje na konsole.

Do projektu zastosowałem tematy **RTTI**, **kontenery STL**, **algorytmy i iteratory STL**.

3. Specyfikacja zewnętrzna

Menu:

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

Po kliknięciu:

Play wyświetla się menu z wyborem mapy

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, zatwierdzanie.

ESC - pauza

R – cofa postać do checkpointu

Pliki:

Tworzenie mapy(poziomu) wygląda następująco. 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

Button.txt

Trap.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

Button X Y ZWROT CZAS

Trap X Y ZWROT CZAS

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.

Przejścia:

Aby otworzyć przejście należy użyć wszystkich dźwigni, które sa przypisane do drzwi lub nacisnąć przycisk, który po jakimś czasie deaktywuje się i przejście się zamyka.

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 W przypadku obiektów aktywny/nieaktywny
**map	Tablica dynamiczna typu int. Przechowuje mapę w postaci 0(podłoga) i 1 (ściana)
Rotation	Typ wyliczeniowy opisujący zwrot obiektu: NORTH, SOUTH, EAST, WEST

Klasy:

Game:

```
//okno program
 sf::RenderWindow window;

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

//typ wyliczeniowy okreslajacy stan gry
enum GameState {MENU,MENU2, GAME,  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;

//klasa do wyswietlania napisow
Text text;

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

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

//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 (podłoga i ściana).
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 środku 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; //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();

/// funkcja czysto wirtualna. Aktualizuje stan obiektu.

///Jako parametr jest podany typ bool przekazujący 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 dezaktywuje pole

void disable();

Obstacle:

///Animowanie obiektu

void animationMove();

//sciezka

Path path

//wektor o ktory porusza sie obiekt
sf::Vector2f vector

Player:
//wektor do poruszania postacia
sf::Vector2f vector;

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

///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();

//zmienna pomocnicza ograniczajaca aktywacje dzwigni
bool enterIsPressed = false;

Gateway:
//vector referencji do dźwigń
VecLever levers;

Buton:
//Licznik
sf::Clock clock;
//długość trwania aktywacji
int time = 10;

Trap:

```
//co ile ma się zmieniać stan
int time = 1;

//Licznik
sf::Clock clock;
```

Path:

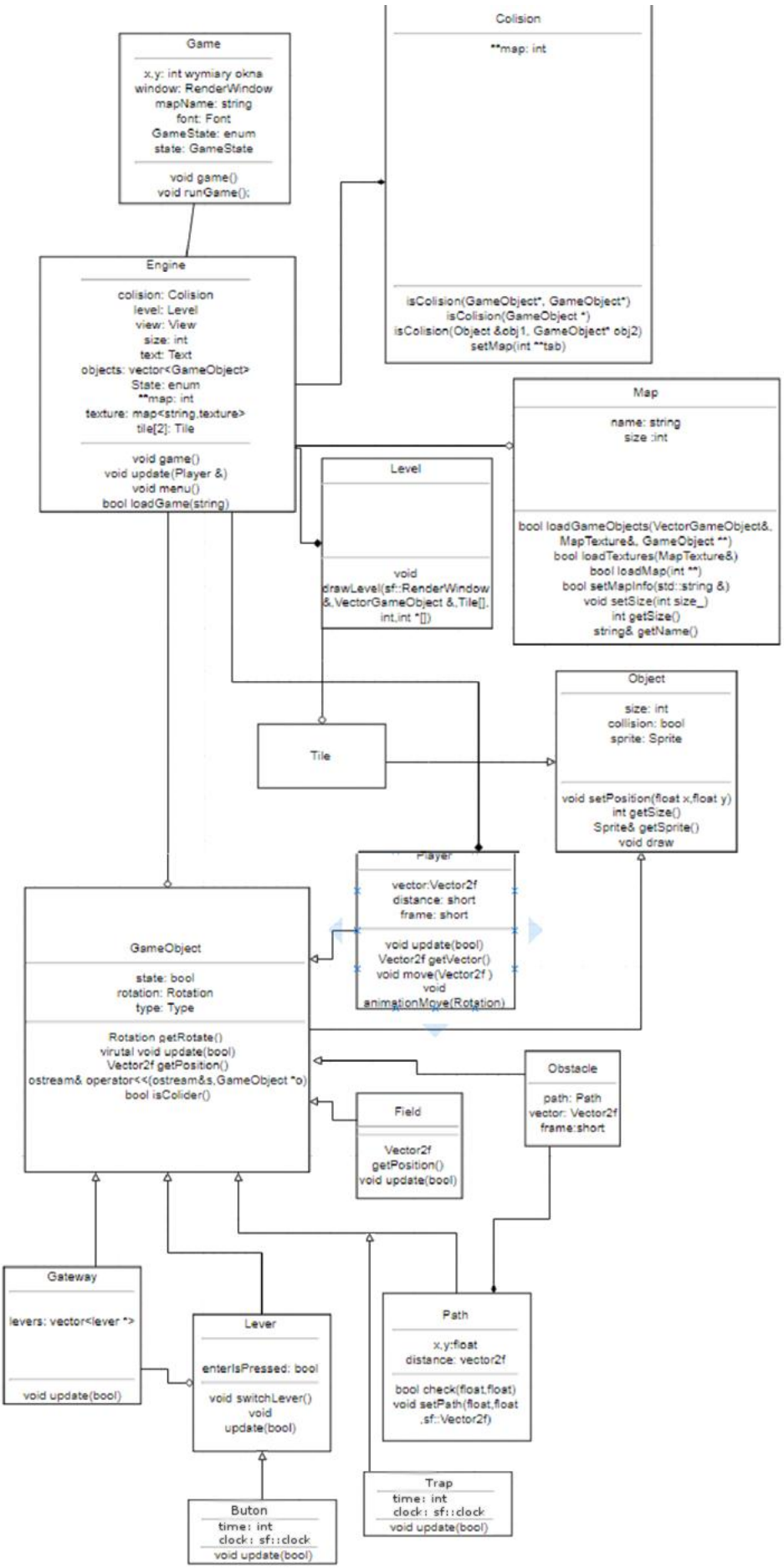
```
//punkt startowy
float x, y;

//odleglosc od punktu startu do konca
sf::Vector2f distance;

///sprawdza czy obiekt nie wyszedl poza sciezke
///jeżeli tak to zmienia kierunek ruchu na przeciwny
bool check(float,float);

///Parametry: x ,y distance
///ustawia sciezke.
void setPath(float,float,sf::Vector2f);
```

UML DIAGRAM


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()
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
levers: vector<lever *>
void update(bool)
Lever
enterIsPressed: bool
void switchLever()
void
update(bool)
Path
x,y:float
distance: vector2f
bool check(float,float)
void setPath(float,float
,sf::Vector2f)
Trap
time: int
clock: sf::clock
void update(bool)
Buton
time: int
clock: sf::clock
void update(bool)

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.