Data wykonania :12.02.2020

Dokumentacja

Projektu wykonanego w ramach przedmiotu

Programowanie w Języku C++

Gra WOW

Mateusz Żydzik

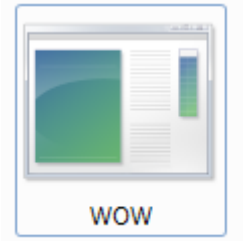
158375

|| EF-DI L02

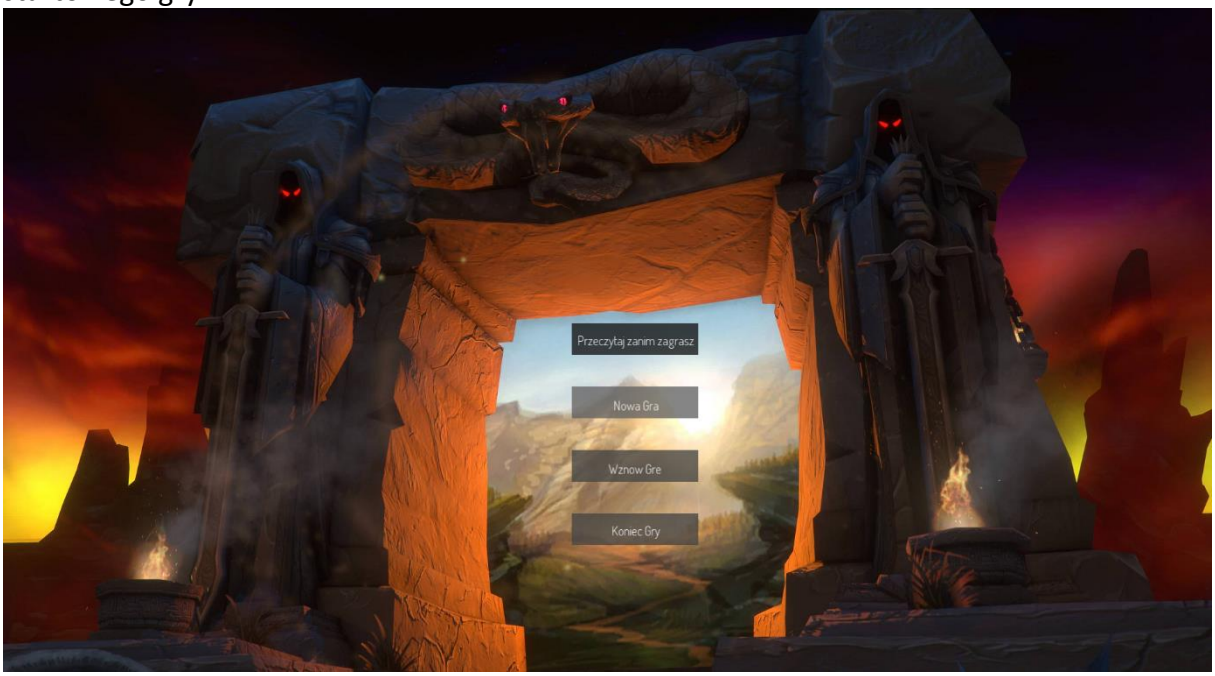
1) Wstęp

Celem projektu było stworzenie aplikacji zorientowanej na budowę obiektową przy użyciu języka C++. Takie podejście miało nauczyć rozbijania projektu na mniejsze części, co jest szczególnie przydatne w przyszłej pracy programistycznej. Dodatkowo można było użyć dowolnej biblioteki dodatkowej, która wzbogacała możliwości programistyczne o dodatkowe funkcje, takie jak rysowanie elementów graficznych oraz obsługa innych rozdzielczości. Jako projekt została obrana gra własnego pomysłu – WOW.

2) Instrukcja dla użytkownika
Pierwszym elementem, jaki należy poczynić w celu uruchomienia wyżej wspomnianej gry jest uruchomienie jej za pomocą pliku WOW.exe:

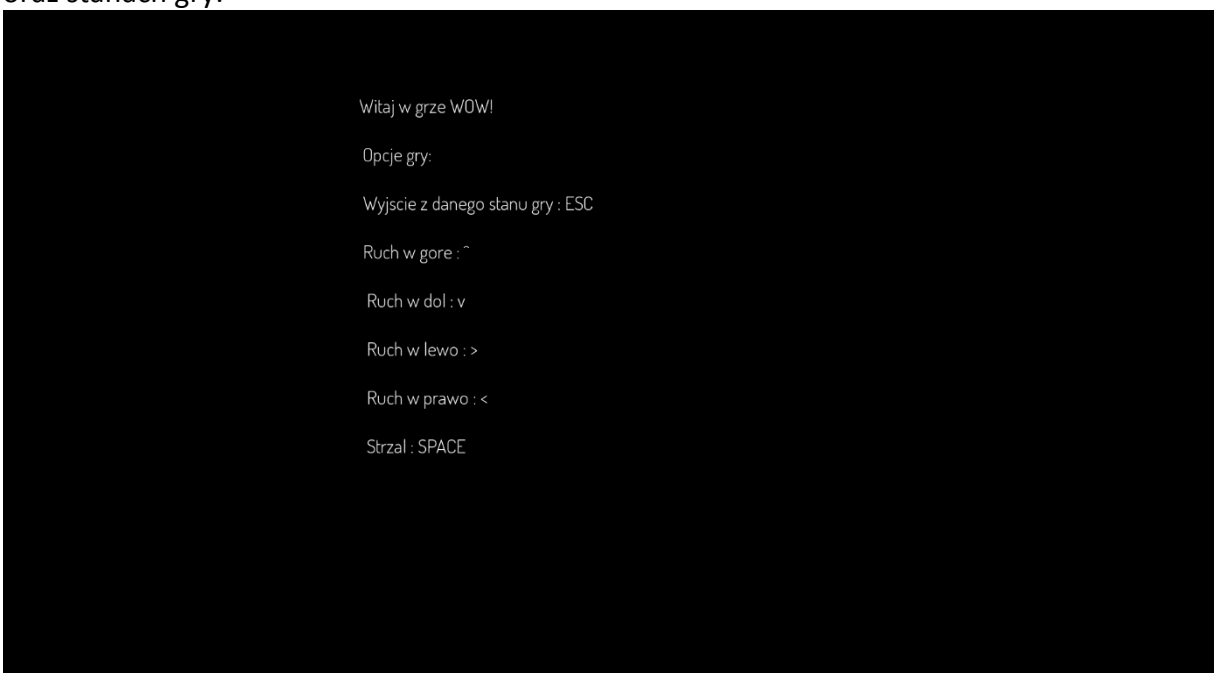


Po dwukrotnym kliknięciu lewym przyciskiem myszy, zobaczymy ekran menu startowego gry:

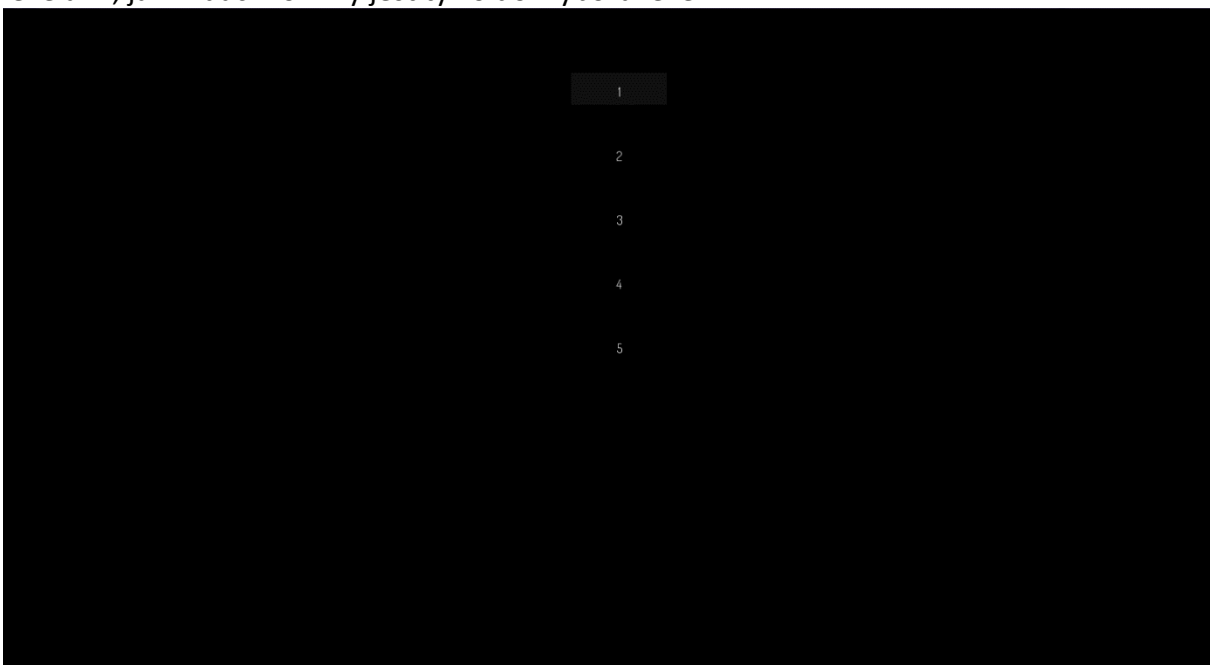


Ruch pomiędzy przyciskami wykonujemy strzałką w dół lub strzałką w górę.

Mamy do wyboru 4 przyciski związane z grą:
- Pierwszy przycisk przenosi nas to podstawowych informacji poruszania się po mapie oraz stanach gry:



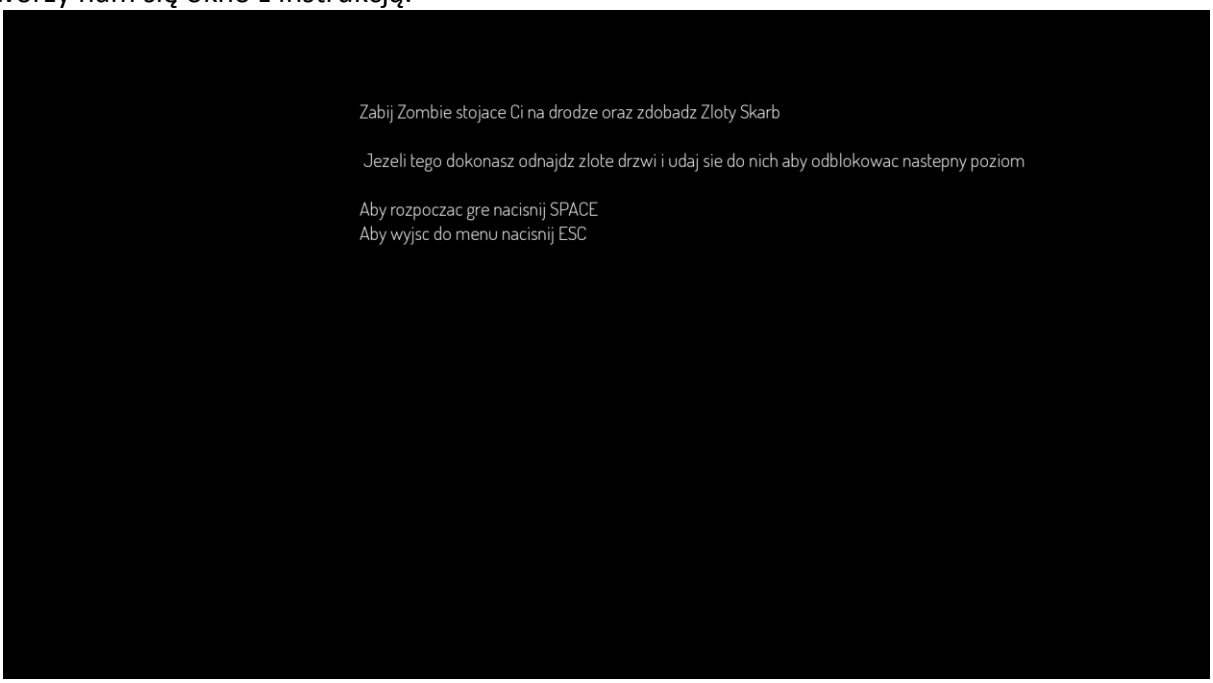
- Drugi przycisk resetuje dotychczasowe w grze oraz przenosi nas do stanu gry z levelami, jak widać możliwy jest tylko do wyboru level 1.



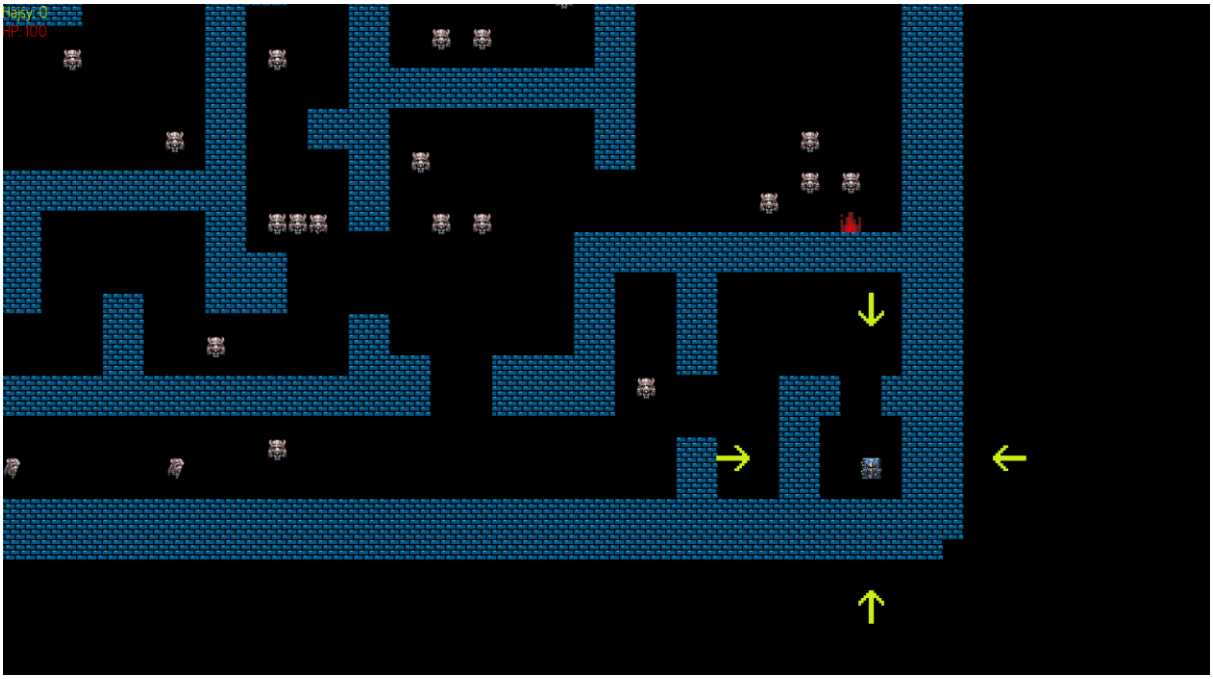
-Trzeci przycisk włącza stan z levelami gry które są dostępne.
Ruch po przyciskach leveli odbywa się tak samo jak po przyciskach w menu startowym gry. Jak łatwo można się domyśleć przykładowo, przejdziesz level 3 wtedy odblokuje się level 4 itp.

- Czwarty przycisk wyłącza gre.

Teraz sama rozgrywka:
Po wciśnięciu przycisku „Nowa gra” oraz wybraniu jedynego możliwego przycisku po nazwie „1”, otworzy nam się okno z instrukcją.



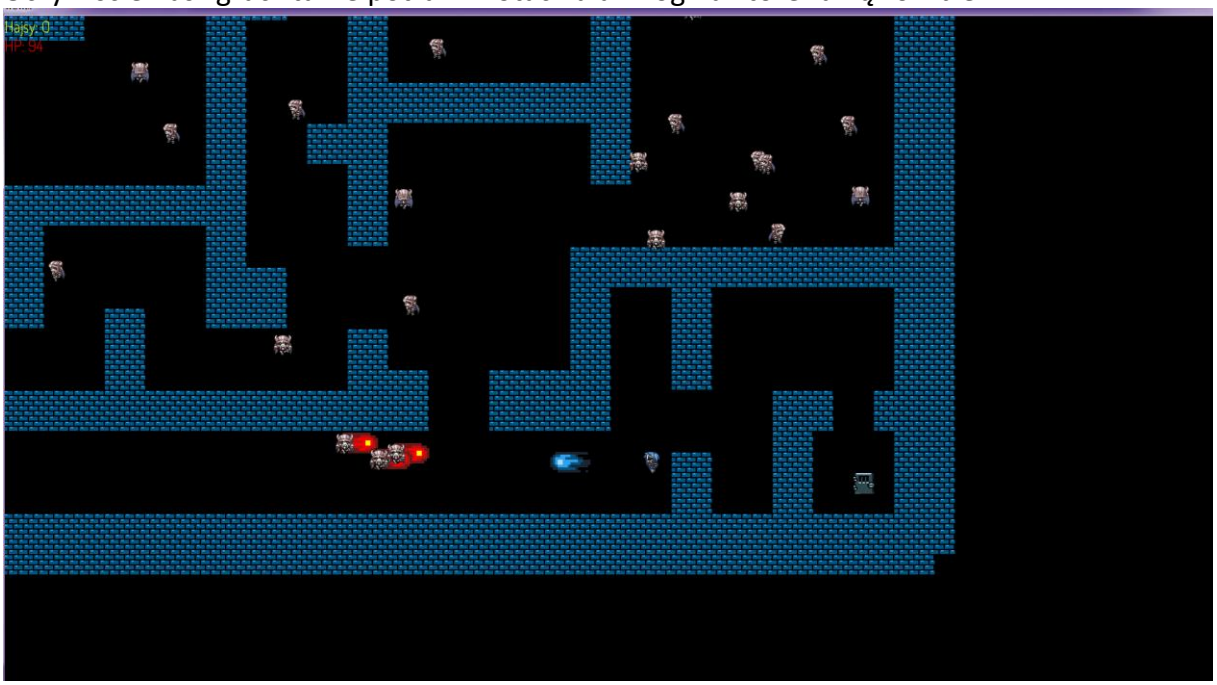
Po wciśnięciu przycisku Spacji ukaże nam się pierwszy poziom gry. Aby Wiedzieć gdzie gracz startuje ukazują nam się strzałki wskazujące na naszego bohatera.



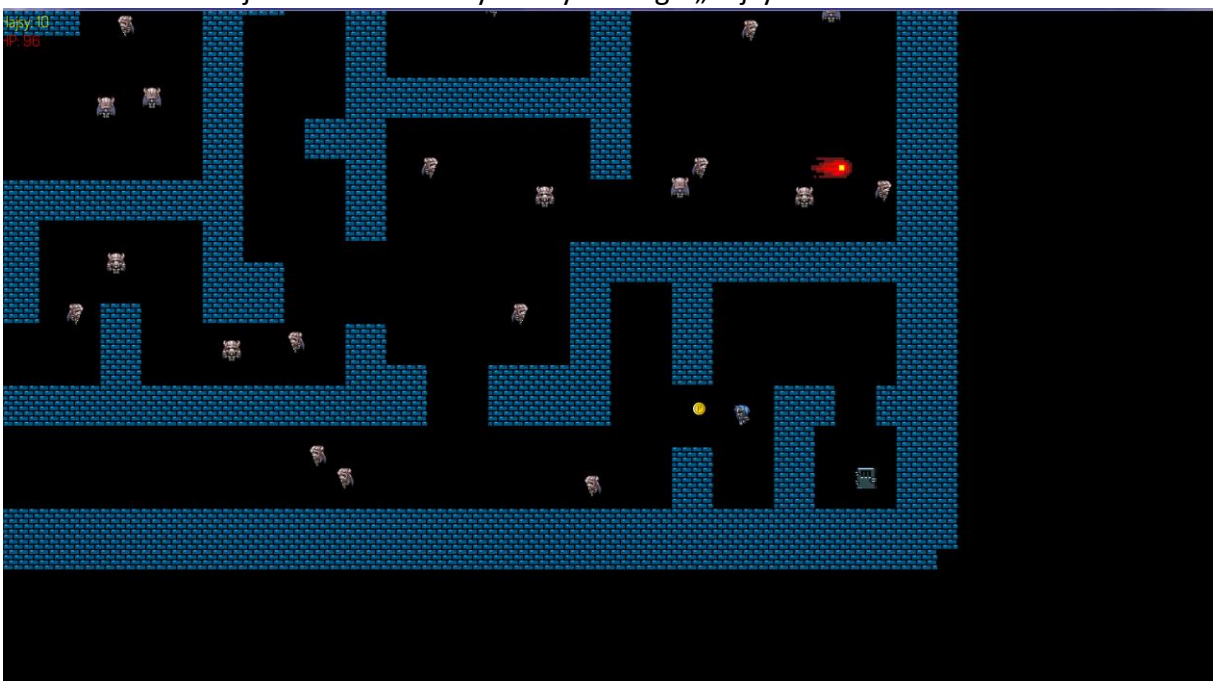
W lewym górnym rogu jest podane ile zgarnęliśmy pieniędzy oraz nasza ilość życia.

Należy uważać na wrogów- straszliwych zombie miotających kulami ognia. Jeżeli jakaś kula dosięgnie naszego bohatera odejmowana jest odpowiednia ilość życia(co widać na screenie).

Oczywiście nasz gracz także potrafi miotać kulami ognia które ranią zombie.



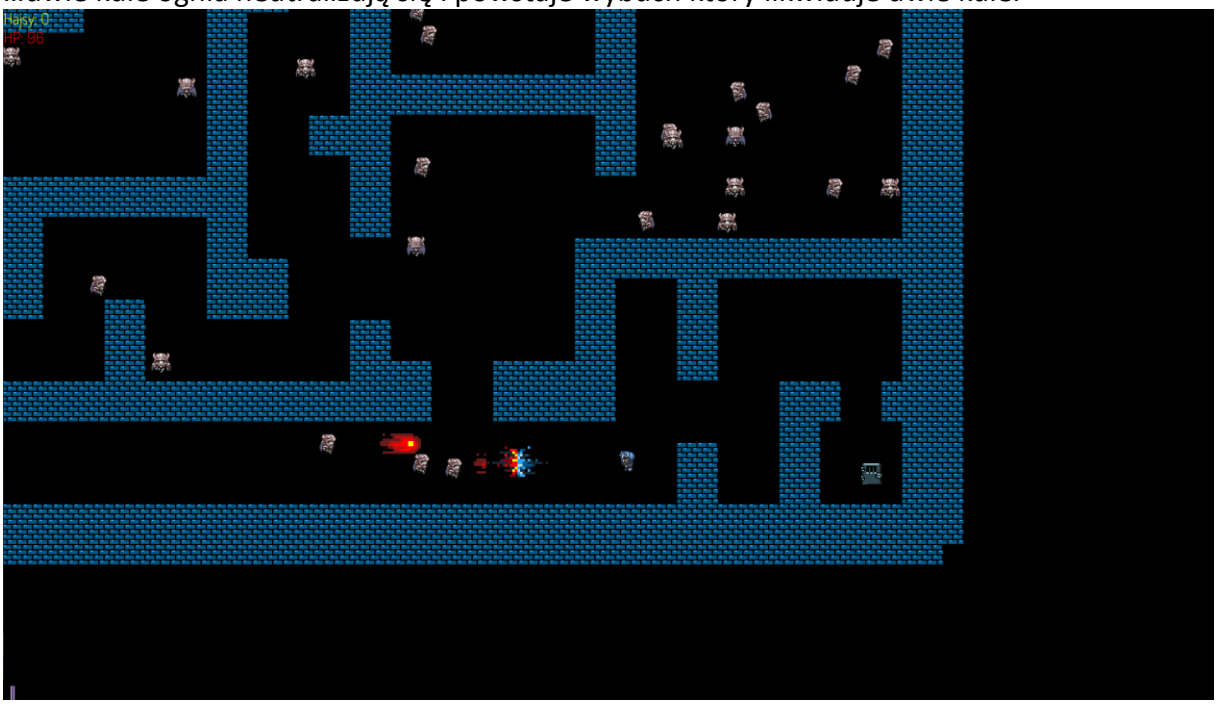
Po zabiciu zombie jest duża szansa by zdobyć z niego „hajsy”



Jak widzimy po zebraniu nasz informator o „hajsach się zwiększył”

Przy bliskim spotkaniu obu kul ognia – naszej i jakiegoś wroga …

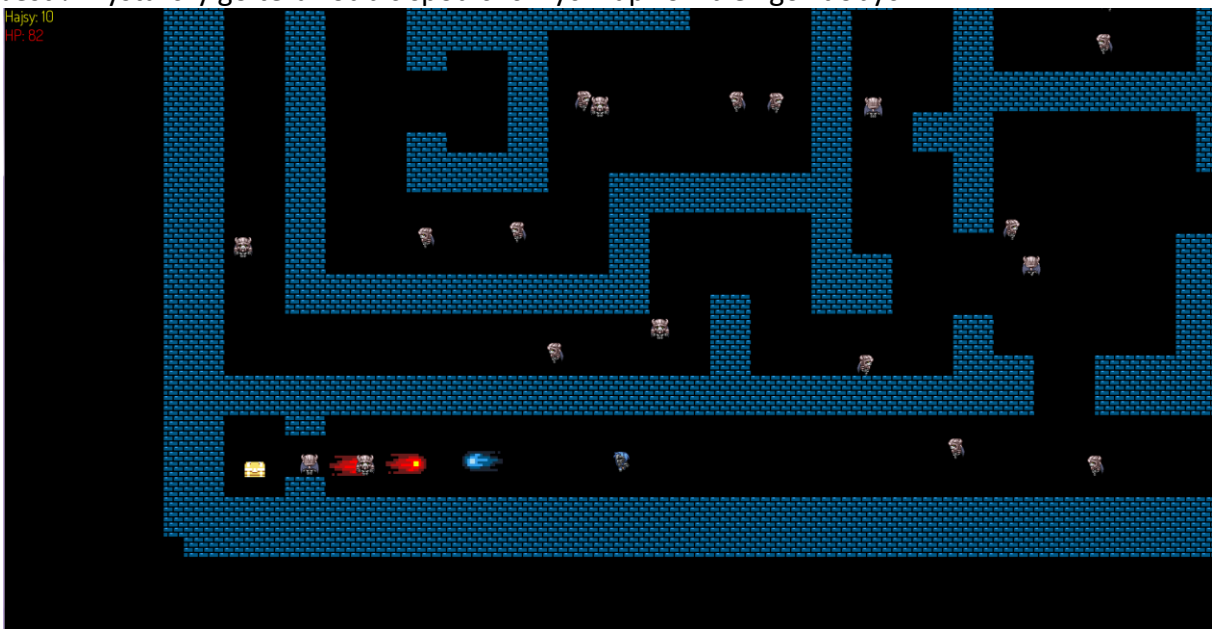
…dwie kule ognia neutralizują się i powstaje wybuch który likwiduje dwie kule.



Aby nie było za łatwo, wrogowie jeżeli „zobaczą” czyli znajdą się w podobnej części mapy, rzucają w Ciebie kulami ognia oraz zaczynają zmierzać do Ciebie.

Jeżeli nasz informator mówiący o ilości życia będzie równy 0 następuje koniec gry.

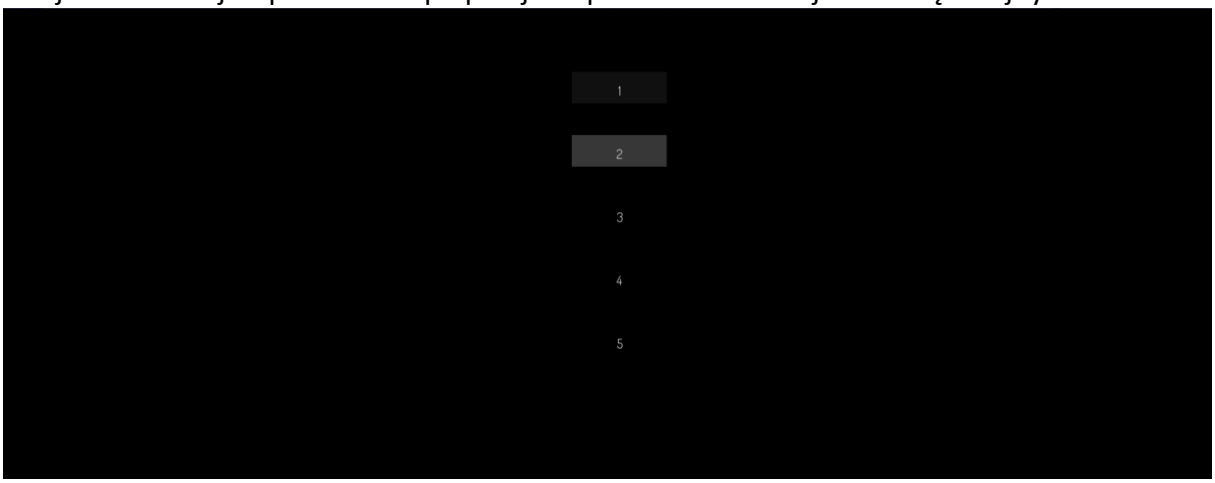
Aby przejść na następny poziom tak jak było w instrukcji należy znaleźć złoty skarb który jest schowany gdzieś na mapie.
Jest ! Wystarczy go teraz odbić spod chciwych łap zombie i go zdobyć.



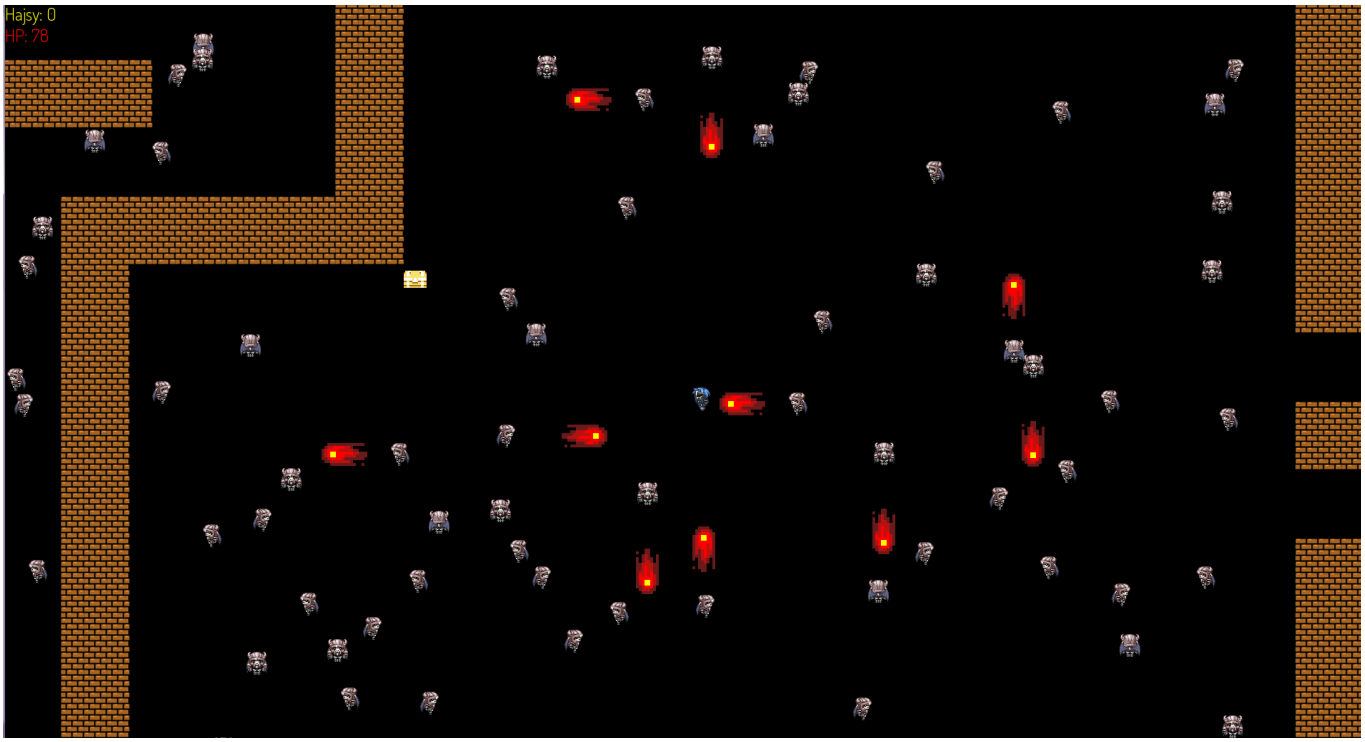
Po odbiciu skarbu i zdobyciu, ukazuję się nam napis „Odnaleziono skarb!”. Teraz pozostaje nam odnaleźć złote drzwi.

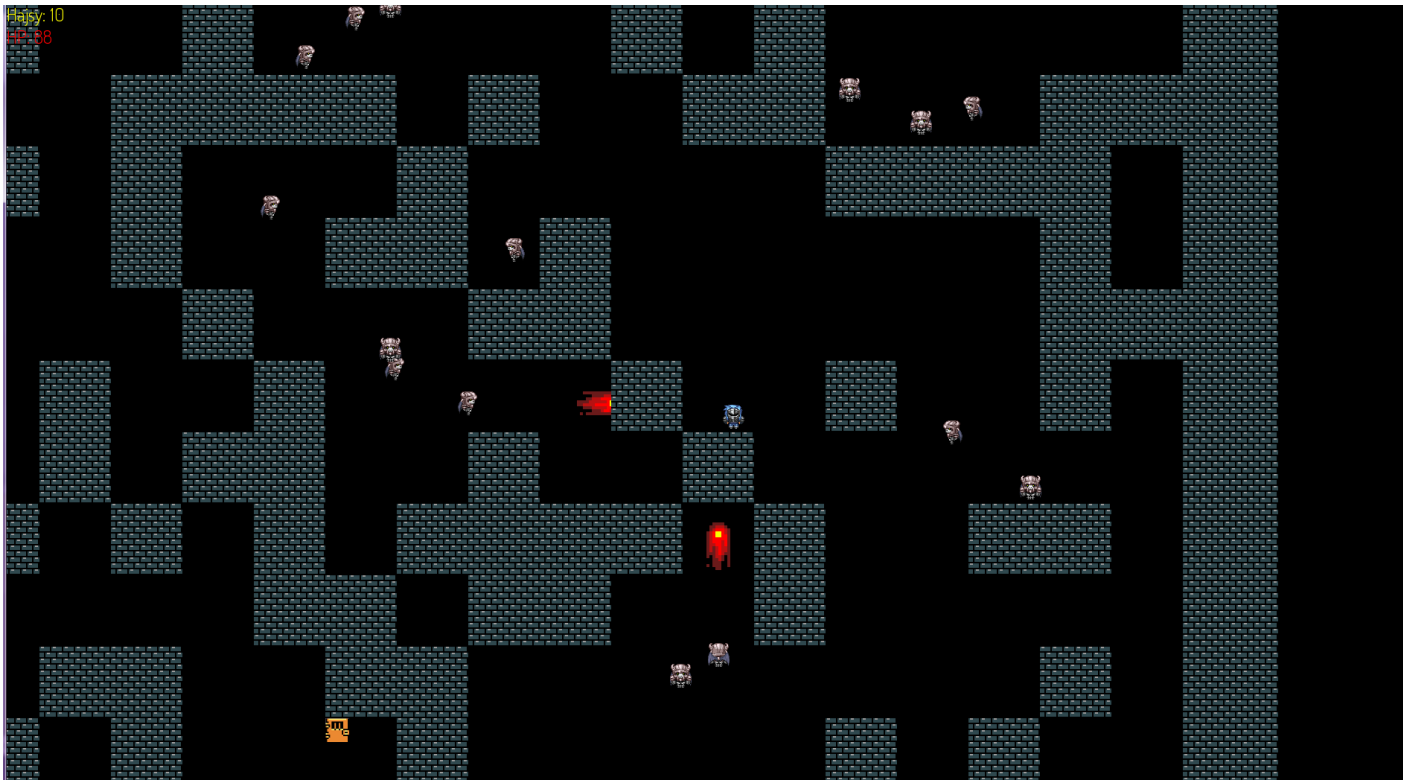
Po odnalezieniu złotych drzwi trzeba się do nich udać. Przeniesie nas to do stanu gry z poziomami ale pod warunkiem że złoty skarb został zdobyty.

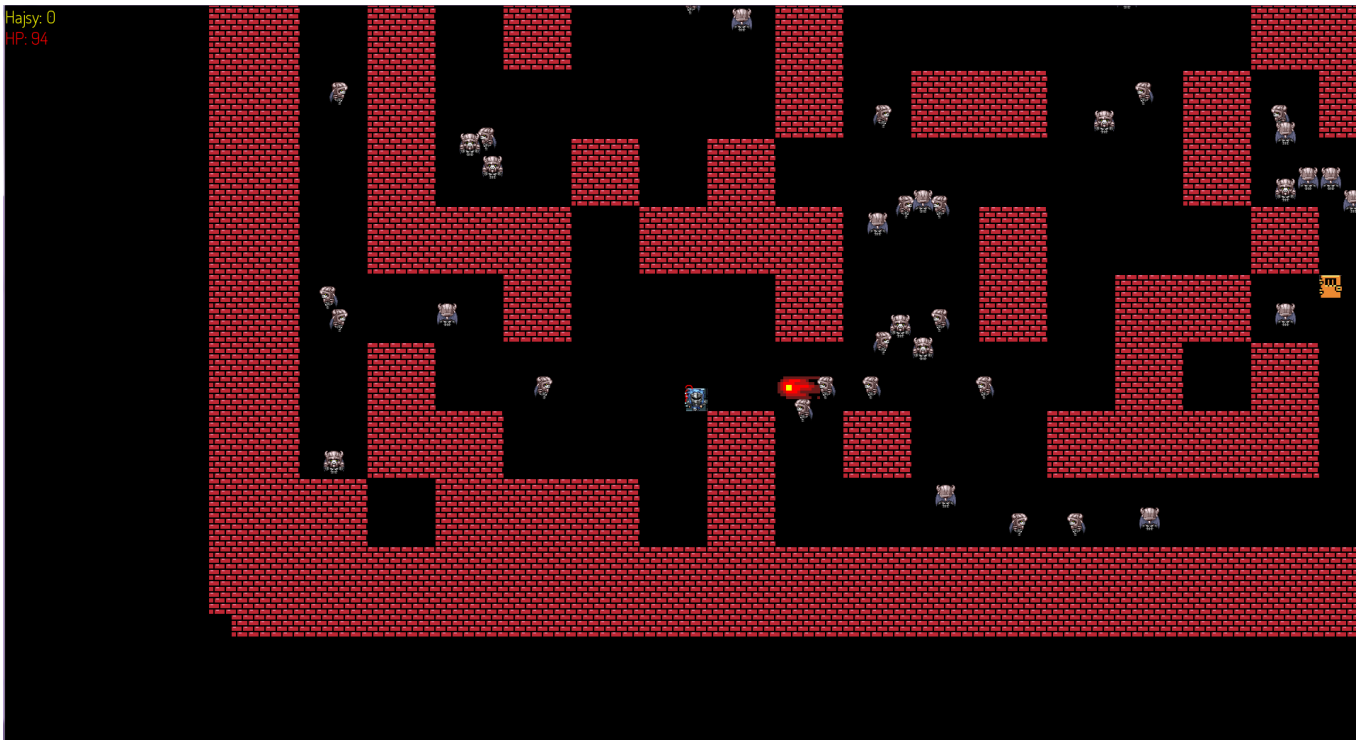
Tak jak wcześniej wspomniałem po przejściu poziomu odblokuje nam się kolejny.



Dodatkowo: wygląd innych map oraz screny z rozgrywki :




Hajsy: 10
HP: 88


Hajsy: 0
HP: 94

3) Dokumentacja kodu źródłowego

Dokumentacja będzie odbywała się od omówienia najważniejszych klas tworzących same stany gry aż po omówienie poszczególnych klas tworzących samą rozgrywkę.

a) WOW.cpp

Ten plik zawiera tylko #include ‘’Game.h” wraz z zainicjowaniem obiektu tworzącego grę Game game; oraz funkcje uruchamiania gry :game.run();

b) Game.h

```
	Game();
	Konstruktor dla klasy Gam. Inicjalizowane są tutaj stany gry oraz
samo okno gry.

	virtual ~Game();
Virtualny destruktor zwalniający pamięc w chwili wyłączenia gry.

	void updateDt();
Aktualizuje czas całej gry – odpwoiedzialny za czas.

	void updateSFMLEvents();
Aktualizuje zdarzenie.

	void update();
Aktualizuje wszystko. Odniesienie do innych metod update oraz
aktualizacja aktualnych stanów gry.


	void render();
Rendowanie czyli tworzenie poszczególnych stanów gry.


	void run();
Aktualizacja rendowanie aktualnego stanu oraz samej rozgrywki

	void initWindow();
Inicjalizacja okna Gry.

	void initVariables();
Inicjalizacja zmiennych potrzebnych do procesu gry.

	void initWindow();
Inicjalizacja okna Gry.

	void initStates();
Inicjalizacja stanów gry.
```

c) State.h

```
State(sf::RenderWindow* window,  std::stack<State*>* states);
```
Konstruktor mający za zadanie zainicjalizowanie stanu oraz okna.

```
bool getQuit();
```
Metoda która zwraca czy należy zamknąć dany stan.
```
void endStan();
```
Metoda odpowiedzialna za zamykanie danego stanu Gry.

Metody abstrakcyjne które używamy każdej inaczej w danych stanach gry. W tej klasie nie wykonujemy w nich nic.
```
virtual void updateInput(const float& dt) = 0;
virtual void update(const float& dt) = 0;
virtual void render(sf::RenderTarget* target = NULL) = 0;
```

d) MainMenuState

```
void moveUp();
```
Metoda ruchu po przyciskach w górę.
```
void moveDown();
```
Metoda ruchu po przyciskach w dół.

```
void initBackground();
```
Inicjalizacja grafiki która widnieje w tylnym położeniu menu startowego gry.
```
void initFonts();
```
Inicjalizacja czcionki potrzebnej do przycisków
```
void initButtons();
```
Inicjalizacja przycisków w naszym menu.

```
MainMenuState(sf::RenderWindow* window,  std::stack<State*>* states);
```
Konstruktor mający zadanie inicjalizacje powyższych części menu startowego.

```
virtual ~MainMenuState();
```
Virtualny destructor zwalniający pamięc

```
void updateButtons();
```
Aktualizacja poruszania się po przyciskach

```
void renderButtons(sf::RenderTarget* target = NULL);
```
Rendowanie przycisków.

```
void update(const float& dt);
```
Odniesienie do innych aktualizacji.

```
void updateInput(const float& dt);
```
Aktualizacja informacji wychodzących tzn. klikniecia przycisków/ poruszanie się po przyciskach.

```
void render(sf::RenderTarget* target = NULL);
```
Tworzenie całego menu oraz powyższych części menu.

e) LevelState.h

```
	void initFonts();
```

Inicjalizacja czcionki podobnie jak w MainState.

```
	void initLevel();
```

Inicjalizacja leveli – potrzebne do pokazania które poziomy są odblokowane.

```
	void initButtons();
```

Inicjalizacja przycisków z numerami poziomów.

```
	LevelState(sf::RenderWindow* window,  std::stack<State*>* states);
```

Konstruktor odpowiedzialny za odwołania inicjalizacji całego stanu gry.

```
	~LevelState();
```

Destruktor zwalniający pamięc.

```
	void updateInput(const float& dt);
```

Metoda aktualizująca informacje otrzymane z klawiatury, odpowiedzialna za ruch w górę/dół przycisków.

```
	void updateButtons();
```

Metoda aktualizująca przyciski odpowiedzialna za pokazanie na którym aktualnie przycisku jest kursor oraz ukryciu reszty.

```
	void update(const float& dt);
```

Metoda aktualizująca z odniesieniem do innych aktualizacji.

```
	void renderButtons(sf::RenderTarget* target = NULL);
```

Rendowanie – tworzenie przycisków.

```
	void render(sf::RenderTarget* target = NULL);
```

Rendowanie – tworzenie z odniesieniem do przycisków oraz tworzenie widoku okna.

f) InstructionState.h

```
	void initFonts();
```

Inicjalizacja Czcionki potrzebnej do całego opisu.

```
	InstructionState(sf::RenderWindow* window,  std::stack<State*>* states, int stan);
```

	Konstruktor w ktorym jest odniesienie do inicjalizacji czcionki oraz ustawienie tekstu wraz z wersją co ma być napisane(w zależności w jakim stanie gry jest używany).

```
	virtual ~InstructionState();
```

Destruktor zwalniający pamięc.

```
	void updateButtons();
```

Aktualizacja przycisków oraz wyprowadzenia z klawiatury.

```
	void update(const float& dt);
```

Aktualizacja widoku oraz odniesienie do aktualizacji przycisków.

```
	void renderButtons(sf::RenderTarget* target = NULL);
```

Rendowanie – tworzenie przycisków.

```
	void render(sf::RenderTarget* target = NULL);
```

Rendowanie – tworzenie z odniesieniem do przycisków oraz tworzenie widoku okna oraz tekstu napisanego na ekranie.

g) GameState.h – ostatni z stanów gry, lecz najbardziej rozbudowany odpowiedzialny za całokształt rozgrywki oraz wszystkie efekty

```
	void initTextures();
```

Inicjalizacja wszystkich textur które widzimy w czasie rozgrywki oraz czcionki.

```
	void initPlayers();
```

Inicjalizacja gracza który jest bohaterem gry.

```
	void initEnemy();
```

Inicjalizacja wrogów losowo rozstawionych na mapie wraz z blokadą losowania na textury muru oraz blokadą odległości od miejsca startowego gracza.

```
	void initWall();
```

Inicjalizacja muru, wczytanie z pliku txt całej mapy.

```
	void initText();
```

Inicjalizacja tekstu który będzie wyświetlany w rozgrywce.

```
	void initView();
```

Inicjalizacja widoku który przesuwa się wraz z ruchem bohatera. Ustawienie bohatera w centrum.

```
	void initArrow();
```

Inicjalizacja strzałek które wskazują początkową pozycje bohatera.

```
	void updateInput(const float& dt);
```

Metoda aktualizująca informacje wyjściowe z klawiatury w czasie gry czyli ruchy bohatera, wyjscie ze stanu czy ataki.

```
	void updateEnemy(const float& dt);
```

Metoda akualizująca wszystkich wrogów widocznych na mapie. Ataki oraz ruch wraz z kolizjami z murem.

```
	void updateCoinBum(const float& dt);
```

Metoda aktualizująca wszystkie “hajsy” – pieniązki widoczne na mapie oraz aktualizowanie wybuchów.

```
	void updateAttack(const float& dt);
```

Metoda aktualizująca ataki gracza – kolizje z murem oraz wrogami.

```
	void updateAttackE(const float& dt);
```

Metoda aktualizująca ataki wroga – kolizje z murem oraz graczem.

```
	void updateText(const float& dt);
```

Metoda aktualizująca tekst – życie oraz obrażenia.

```
	void updateView(const float& dt);
```

Metoda aktualizująca widok gracza – odpowiedzialna za przesuwanie widoku wraz z ruchem bohatera.

```
	void updateEnd(const float& dt);
```

Metoda aktualizująca zakączenie każdego z poziomów.

```
	void updateArrow(const float& dt);
```

Metoda aktualizująca począkowe strzałki wskazujące położenie.

```
public:
	GameState(sf::RenderWindow* window,  std::stack<State*>* states,int lvl);
```

Konstruktor odpowiedzialny za inicjalizacje powyższych aspektów całej rozgrywki.

virtual ~GameState();
Destruktor który zwalnia pamięć dynamicznej alokacji m.in. wszystkich wrogów czy muru.

void render(sf::RenderTarget* target);
Rendowanie – tworzenie wszystkiego co widzimy w czasie rozgrywki.

void update(const float& dt);
Metoda aktualizująca posiadająca w sobie odniesienia do innych metod aktualizujących.

h) Entity
void initVariables();
Inicjalizacja zmiennych
Entity();
Konstruktor z odniesieniem do inicjalizacji.

virtual ~Entity();
Wirutalny destruktor odpowiedzialny za zwolnienie pamięci;

void createSprite(sf::Texture* texture);
Metoda odpowiedzialna za stworzenie kształtu.
virtual void setPosition(const float x, const float y);
Metoda mająca na celu ustawianie pozycji początkowej.

virtual void update(const float& dt) = NULL;
Metoda abstrakcyjna inna dla kazdej pochodnej klasy.
virtual void render(sf::RenderTarget* target);
Rendowanie danej textury.

i) Attack.h
Attack(float x, float y, sf::Texture* texture);
Konstruktor mający na celu zainicjonowanie textury oraz zmiennych

void update(const float& dt);
Metoda służąca do aktualizacji ruchu wraz z innymi właściwościami ataku.

j) Coin.h
Coin(float x, float y, sf::Texture* texture);
Konstruktor z odniesieniem do inicjalizacji.

void update(const float& dt);
Aktualizacja coinow / wybuchów.

k) Enemy.h

Enemy(float x, float y, sf::Texture* texture);
Konstruktor odpowiedzialny za inicjalizacje zmiennych i textury

void update(const float & dt);
Metoda aktualizujaca ruchy oraz textury wrogów

l) Player.h

Player(float x, float y, sf::Texture* texture);
Konstruktor odpowiedzialny za inicjalizacje zmiennych i textury.

void update(const float& dt);
Metoda akutalizująca ruchy i textury bohatera.

m) StartArrow.h

StartArrow(float x, float y, sf::Texture* texture;
Konstruktor inicjializujący kierunek strzałek wraz z texture

void update(const float& dt);
Metoda odpowiedzialna aktualizacje ruchu strzałek.

n) TextDisplay.h

TextDisplay()
Konstruktor ustawiający czcionke pozycje oraz wielkość tekstu

void update();
Aktualizacja textu.

o) Wall.h

Wall(float x, float y, sf::Texture* texture);
Konstruktor odpwoeidzialny za ustawienie pozycji oraz załadowanie textury

void update(const float& dt);
Metoda aktualizująca- nic nie powoduje ale jest potrzebna bo zastosowany jest polimorfizm.

p) Random.h

Plik posiadający tylko funkcje która losuje liczbe.

q) Button.h

Button(float x, float y, float width, float height,
sf::Font* font, std::string text,
sf::Color idleColor, sf::Color hoverolor, sf::Color activeColor);
Konstruktor ustawiający pozycje przycisku, długosc/ szerokosc, czcionke, text oraz kolory w różnych stanach.

~Button();
Konstruktor zwalniający przydzieloną pamięć.

const bool isPressed() const;
Metoda odpowiedzialna za sprawdzenie czy przycisk jest wcisniety

void update(bool activ);
Aktualizacja przycisku.

void render(sf::RenderTarget * target);
Rendowanie- tworzenie przycisku.