Gliwice, 28.09.2019r.

Laboratorium Programowania Komputerów 2

Temat: Gra Copter Game

Autor: Wojciech Pietrzak
Kierunek: Informatyka
sem VI
grupa dziekańska BDIS
Prowadzący: mgr inż. Grzegorz Wojciech Kwiatkowski

1. Temat

Moim zadaniem było wykonanie programu z wykorzystaniem biblioteki OpenGL w postaci gry: Copter Game. Program został napisany w języku C.

2. Analiza, projektowanie

2.1 struktury danych

Już przy projektowaniu założyłem, że struktury jakie mi będą potrzebne to:

```c
typedef struct HELICOPTER {
	int x;
	int y;
	int radius;
	int life;
	int startingLife;
}Helicopter;

typedef struct OBSTACLE {
	int verticies[8];
}Obstacle;

typedef struct BULLET
{
	int x;
	int y;
	int radius;
	float colorR;
	float colorG;
	float colorB;
	int speed;
	int running;
}Bullet;
```

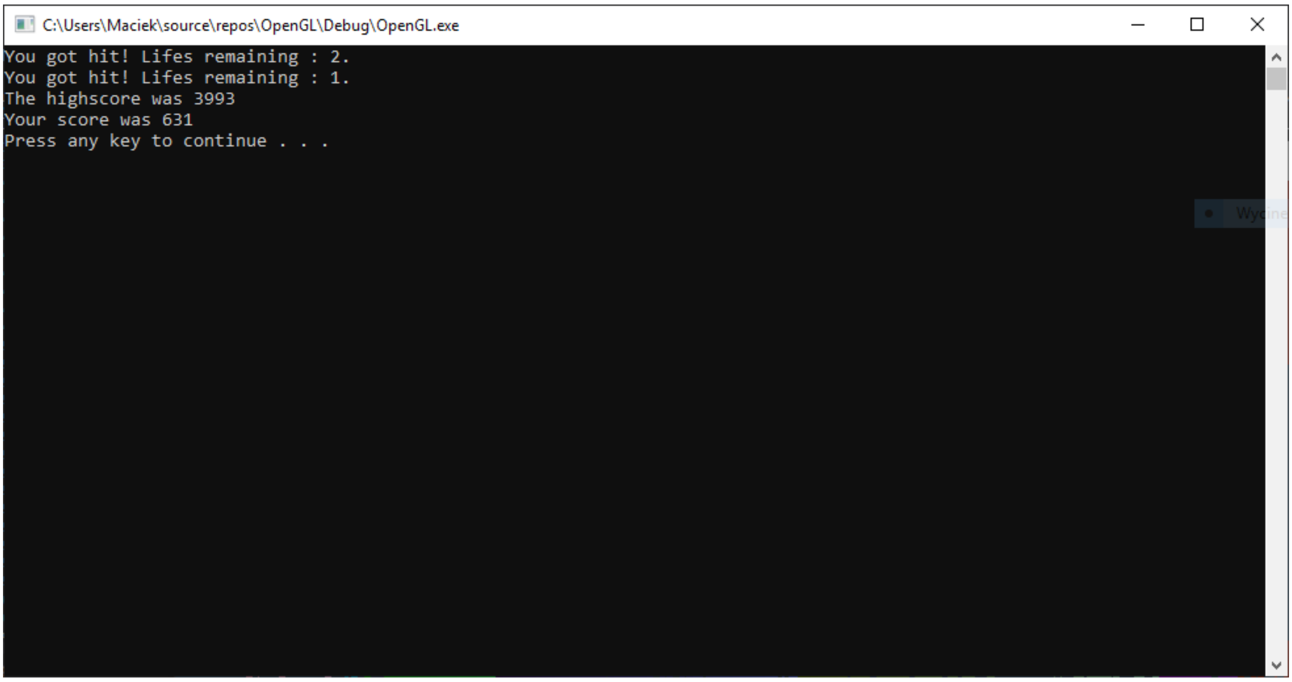
2.2. Analiza problemu, podstawy teoretyczne

Do obliczenia odległości między punktami kolizji (Obstacles) oraz strzałami (Bullet) oraz playerem (Helicopter) wykorzystałem twierdzenie Pitagorasa:

```c
float getDistance(int x1, int x2, int y1, int y2)
{
	int dx = x2 - x1;
	int dy = y2 - y1;
	int ddx = dx * dx;
	int ddy = dy * dy;
	return sqrt(ddy + ddx);
}
```

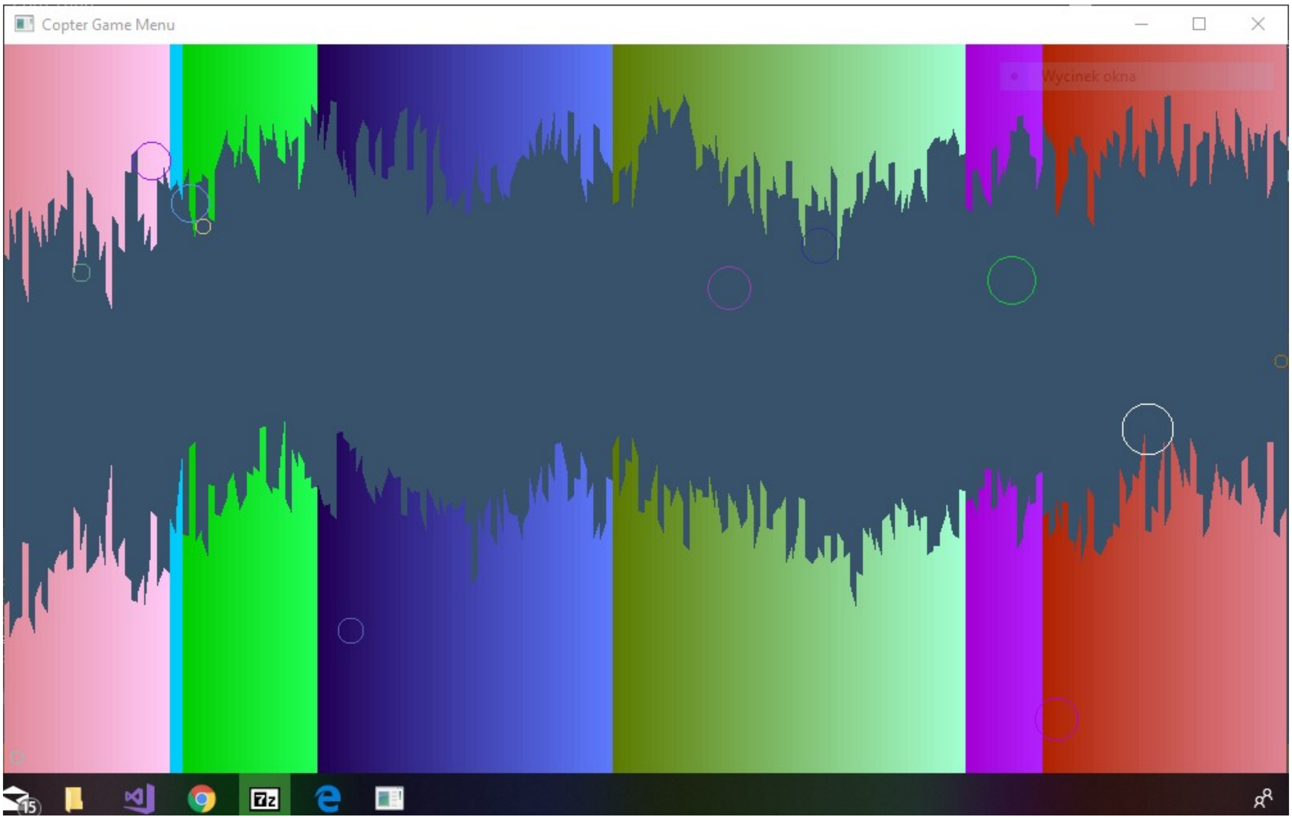
3. Specyfikacja zewnętrzna - Instrukcja obsługi oraz zachowanie programu.

Po uruchomieniu pliku OpenGL.exe włącza nam się okno graficzne ze światem gry oraz w drugim oknie konsola, w której zapisywana jest ilość zdobytych punktów po zakończeniu gry, najlepszy wynik oraz na bierząco wyskakuje infornacja o przegranej czy stracinym życiu. Jest ich trzy. Za trzecim zderzeniem z przeszkodą, gra się kończy, a informacja jest wyświetlana na konsoli. Przy pierwszym i drugim zderzeniu player zmienia na moment swój kolor domyślny biały na czewony (przy utracie życia). To również jest wyświetlane na konsoli. (ilość pozostałych żyć). Wygląda to w następujścy sposób:



```
You got hit! Lifes remaining : 2.
You got hit! Lifes remaining : 1.
The highscore was 3993
Your score was 631
Press any key to continue . . .
```

A tak wygląda świat gry:



Gracz oznaczony jest kolorem białym, natomiast pozostałe kule są strzałami.

Po przejściu graczem z wykorzystaniem strzałek (góra, dół), pojawia nam się nowy ekran, kule pojawiają się częściej i łatwiej jest stacić życie. Strzałka w lewo "hamuje" gracza na moment, aby uniknąć zderzenia.

4. Specyfikacja wewnętrzna

Użyte zmienne:

```
#define WINDOW_WIDTH 1000
#define WINDOW_HEIGHT 600
#define TERRAIN_WIDTH 200

// Constants
const GLint WIDTH = WINDOW_WIDTH;
const GLint HEIGHT = WINDOW_HEIGHT;
const GLint NUMBER_OF_OBSTICLES = TERRAIN_WIDTH;
const GLfloat OBSTICLE_WIDTH = WINDOW_WIDTH / TERRAIN_WIDTH;

// Game variables
int HIGHSCORE = 0;
int SCORE = 0;
int COLLISION_TIMEOUT = 0; //czas jaki Player jest nietykalny po utracie życia
int HARDNESS = 3000;

// Color variables
float colorR = 0.01;
float colorG = 0.01;
float colorB = 0.01;

// Player
int speed = 3; //prędkość z jaką porusza się Player
```

Użyte funkcje:

```
// Setup
void Setup();
void SetupOpenGL();
Obstacle GenerateUpperObstacle(int X, int Y); // zwraca element górnej krawędzi
Obstacle GenerateBottomObstacle(int X, int Y); // zwraca element dolnej krawędzi
void GenerateTerrain();

// Move functions
void MoveUp();
void MoveDown();
void MoveLeft();
void MoveRight();

// Detect input
void GetUserInput();
void KeyInput();

// Highscore manipulation
void LoadScore();
void OverwriteScore();

// Drawing
void DrawTerrain();
void DrawObstacle(Obstacle obstacle);
void DrawPlayer();
void DrawCircle(float cx, float cy, float r, int num_segments);
void DrawBullets();

// MainLoop
void MainLoop();
void EndGame();

// Collisions
void CheckCollisions();
void CheckPlayer_TerrainCollisions();
void CheckPlayer_BulletCollisions();
void CollisionOccured();

// Next level
void CheckIfNextLevel();
void NextLevel();

// Terrain movement
void MoveTerrain();

// Bullets
Bullet MakeBullet(int x, int y); //zwraca strukturę bullet
void GenerateBullets();
void WakeBullets();
void MoveBullets();

// Others
int getRandom(int bottomBound, int upperBound);
float getDistance(int x1, int x2, int y1, int y2);
```

4. Kod źródłowy programu

```
// Disable warnings
#define _CRT_SECURE_NO_WARNINGS

// Includes
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

// GLEW
#define GLEW_STATIC
#include <GL/glew.h>

// GLFW
#include <GLFW/glfw3.h>

#define WINDOW_WIDTH 1000
#define WINDOW_HEIGHT 600
#define TERRAIN_WIDTH 200

// Constants
const GLint WIDTH = WINDOW_WIDTH;
const GLint HEIGHT = WINDOW_HEIGHT;
const GLint NUMBER_OF_OBSTICLES = TERRAIN_WIDTH;
const GLfloat OBSTICLE_WIDTH = WINDOW_WIDTH / TERRAIN_WIDTH;

// Game variables
int HIGHSCORE = 0;
int SCORE = 0;
int COLLISION_TIMEOUT = 0;
int HARDNESS = 3000;

// Color variables
float colorR = 0.01;
float colorG = 0.01;
float colorB = 0.01;

// Structure
enum STATE { TITLE, PLAYING, GAMEOVER };
enum MYKEYS { KEY_UP, KEY_DOWN };

typedef struct HELICOPTER {
        int x;
        int y;
        int radius;
        int life;
        int startingLife;
}Helicopter;

typedef struct OBSTACLE {
        int verticies[8];
}Obstacle;

typedef struct BULLET
{
        int x;
        int y;
        int radius;
        float colorR;
        float colorG;
        float colorB;
        int speed;
        int running;
}Bullet;

// Game structures
Obstacle UpperTerrain[TERRAIN_WIDTH];
Obstacle BottomTerrain[TERRAIN_WIDTH];
Bullet Bullets[TERRAIN_WIDTH];

// Player
Helicopter Player;
int speed = 3;
void PrintPlayerPosition()
{
        printf("X : %d Y : %d\n", Player.x, Player.y);
}

// Window structures
GLFWwindow * MenuWindow;

// Setup
void Setup();
void SetupOpenGL();
Obstacle GenerateUpperObstacle(int X, int Y);
Obstacle GenerateBottomObstacle(int X, int Y);
void GenerateTerrain();

// Move functions
void MoveUp();
void MoveDown();
void MoveLeft();
void MoveRight();

// Detect input
void GetUserInput();
void KeyInput();

// Highscore manipulation
void LoadScore();
void OverwriteScore();

// Drawing
void DrawTerrain();
void DrawObstacle(Obstacle obstacle);
void DrawPlayer();
void DrawCircle(float cx, float cy, float r, int num_segments);
void DrawBullets();

// MainLoop
void MainLoop();
void EndGame();

// Collisions
void CheckCollisions();
void CheckPlayer_TerrainCollisions();
void CheckPlayer_BulletCollisions();
void CollisionOccured();

// Next level
void CheckIfNextLevel();
void NextLevel();

// Terrain movement
void MoveTerrain();

// Bullets
Bullet MakeBullet(int x, int y);
void GenerateBullets();
void WakeBullets();
void MoveBullets();

// Others
int getRandom(int bottomBound, int upperBound);
float getDistance(int x1, int x2, int y1, int y2);

int main()
{
        Setup();
        SetupOpenGL();
        GenerateTerrain();
        GenerateBullets();
        MainLoop();
}

// Setup
void Setup()
{
        srand(time(NULL));
        Player.x = 0;
        Player.y = HEIGHT / 2;
        Player.radius = 20;
        Player.startingLife = 3;
        Player.life = Player.startingLife;
}
void SetupOpenGL()
{
        //Setting up GLFW.
        glfwInit();

        //Creating the main menu window.
        MenuWindow = glfwCreateWindow(WIDTH, HEIGHT, "Copter Game Menu", NULL, NULL);

        //Check successful window creation.
        if (MenuWindow == NULL)
        {
                glfwTerminate();
                return EXIT_FAILURE;
        }

        //Setting focus.
        glfwMakeContextCurrent(MenuWindow);

        //Enabling GLEW new functionality.
        glewExperimental = GL_TRUE;

        //Initializing GLEW.
        if (glewInit() != GLEW_OK)
        {
                glfwTerminate();
                return EXIT_FAILURE;
        }

        //Data initialization.
        {
                int x; int y;
                glfwGetFramebufferSize(MenuWindow, &x, &y);
        }

        //Some other OpenGL options.
        glOrtho(0.f, WINDOW_WIDTH, WINDOW_HEIGHT, 0.f, 0.f, 1.f);
        glfwSetInputMode(MenuWindow, GLFW_CURSOR, GLFW_CURSOR_HIDDEN);
}
Obstacle GenerateUpperObstacle(int X, int Y)
{
        Obstacle obstacle;
        obstacle.verticies[0] = X - OBSTICLE_WIDTH / 2;
        obstacle.verticies[1] = Y;
        obstacle.verticies[2] = X + OBSTICLE_WIDTH / 2;
        obstacle.verticies[3] = Y;
        obstacle.verticies[4] = X + OBSTICLE_WIDTH / 2;
        obstacle.verticies[5] = 0;
        obstacle.verticies[6] = X - OBSTICLE_WIDTH / 2;
        obstacle.verticies[7] = 0;
        return obstacle;
}
Obstacle GenerateBottomObstacle(int X, int Y)
{
        Obstacle obstacle;
        obstacle.verticies[0] = X - OBSTICLE_WIDTH / 2;
        obstacle.verticies[1] = Y;
        obstacle.verticies[2] = X + OBSTICLE_WIDTH / 2;
        obstacle.verticies[3] = Y;
        obstacle.verticies[4] = X + OBSTICLE_WIDTH / 2;
        obstacle.verticies[5] = HEIGHT;
        obstacle.verticies[6] = X - OBSTICLE_WIDTH / 2;
        obstacle.verticies[7] = HEIGHT;
        return obstacle;
}
void GenerateTerrain()
{
        int index;
        int lastUpperValue = 100;
        int lastBottomValue = HEIGHT - 100;
        int random;
        for (index = 0; index < NUMBER_OF_OBSTICLES; index++)
        {

                float XPosition = (index + 0.5) * OBSTICLE_WIDTH;
                UpperTerrain[index] = GenerateUpperObstacle(XPosition, lastUpperValue);
                BottomTerrain[index] = GenerateBottomObstacle(XPosition, lastBottomValue);
                random = getRandom(-10, 10);
                lastBottomValue += random;
                lastUpperValue += random;
                if (lastBottomValue > HEIGHT)
                {
                        lastBottomValue -= 2 * random;
                        lastUpperValue -= 2 * random;
                }
                if (lastUpperValue < 0)
                {
                        lastBottomValue -= 2 * random;
                        lastUpperValue -= 2 * random;
                }
        }
}

// Move functions
void MoveUp()
{
        Player.y -= speed;
}
void MoveDown()
{
        Player.y += speed;
}
void MoveLeft()
{
        Player.x -= speed;
}
void MoveRight()
```

```
{
	Player.x += speed;
}

// Detect input
void GetUserInput()
{
	// Move up
	if (glfwGetKey(MenuWindow, GLFW_KEY_UP) == GLFW_PRESS) {
		MoveUp();
	}
	// Move down
	if (glfwGetKey(MenuWindow, GLFW_KEY_DOWN) == GLFW_PRESS) {
		MoveDown();
	}
	// Strafe right
	if (glfwGetKey(MenuWindow, GLFW_KEY_RIGHT) == GLFW_PRESS) {
		Player.x++;
	}
	// Strafe left
	if (glfwGetKey(MenuWindow, GLFW_KEY_LEFT) == GLFW_PRESS) {
		Player.x-=2;
	}
}
void KeyInput()
{

}

// Highscore manipulation
void LoadScore()
{
	FILE *filescore;
	int tempscore;
	filescore = fopen("highscore.dat", "r");
	if (filescore == NULL)
		HIGHSCORE = 0;
	else
	{
		fscanf(filescore, "%d", &tempscore);
		HIGHSCORE = tempscore;
		fclose(filescore);
	}
	return;
}
void OverwriteScore()
{
	FILE *filescore;
	filescore = fopen("highscore.dat", "w");
	if (SCORE > HIGHSCORE)
		fprintf(filescore, "%d", SCORE);
	else
		fprintf(filescore, "%d", HIGHSCORE);
	fclose(filescore);
	return;
}

// Drawing
void DrawTerrain()
{
	int index;
	for (index = 0; index < NUMBER_OF_OBSTICLES; index++)
	{
		glColor3f(colorR, colorG, colorB);
		colorR += 1.0f / NUMBER_OF_OBSTICLES;
		colorG += 2.0f / NUMBER_OF_OBSTICLES;
		colorB += 3.0f / NUMBER_OF_OBSTICLES;
		if (colorR > 1.0f)
			colorR = 0;
		if (colorG > 1.0f)
			colorG = 0;
		if (colorB > 1.0f)
			colorB = 0;
		DrawObstacle(UpperTerrain[index]);
		DrawObstacle(BottomTerrain[index]);
	}
}
```

```
void DrawObstacle(Obstacle obstacle)
{
        glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(2, GL_INT, 0, obstacle.verticies);
        glDrawArrays(GL_POLYGON, 0, 4);
        glDisableClientState(GL_VERTEX_ARRAY);
}
void DrawPlayer()
{
        if (COLLISION_TIMEOUT > 0 && COLLISION_TIMEOUT % 10 < 5)
        {
                glColor3f(0.9, 0.3, 0.3);
                DrawCircle(Player.x, Player.y, Player.radius, 30);
        }
        else
        {
                glColor3f(1.0, 1.0, 1.0);
                DrawCircle(Player.x, Player.y, Player.radius, 30);
        }
}
void DrawCircle(float cx, float cy, float r, int num_segments)
{
        glBegin(GL_LINE_LOOP);
        for (int ii = 0; ii < num_segments; ii++) {
                float theta = 2.0f * 3.1415926f * ii / num_segments;//get the current angle
                float x = r * cosf(theta);//calculate the x component
                float y = r * sinf(theta);//calculate the y component
                glVertex2f(x + cx, y + cy);//output vertex
        }
        glEnd();
}
void DrawBullets()
{
        int index;
        for (index = 0; index < NUMBER_OF_OBSTICLES; index++)
        {
                if (Bullets[index].running == 1)
                {
                        glColor3f(Bullets[index].colorR, Bullets[index].colorG, Bullets[index].colorB);
                        DrawCircle(Bullets[index].x, Bullets[index].y, Bullets[index].radius, 12);
                }
        }
}

// MainLoop
void MainLoop()
{
        while (!glfwWindowShouldClose(MenuWindow))
        {
                //Clearing the event pool.
                glfwPollEvents();

                //Clearing the view.
                {
                        glClearColor(0.2, 0.3, 0.4, 1.0);
                        glClear(GL_COLOR_BUFFER_BIT);
                }

                //Draw OpenGL stuff.
                {
                        DrawTerrain();
                        DrawPlayer();
                }

                //Player.
                {
                        MoveRight();
                        GetUserInput();
                }

                //Terrain
                {

                }

                //Bullets
                WakeBullets();
                MoveBullets();
                DrawBullets();

                //Swap front and back buffers.
                glfwSwapBuffers(MenuWindow);

                //Check for player colissions.
                if (COLLISION_TIMEOUT == 0)
                {
                        CheckCollisions();
                }
                else
                {

                }

                //Next level
                CheckIfNextLevel();

                if (COLLISION_TIMEOUT > 0)
                        COLLISION_TIMEOUT--;

                MoveTerrain();


                SCORE++;
                //PrintPlayerPosition();
        }
}
void EndGame()
{
        LoadScore();
        printf("The highscore was %d\n", HIGHSCORE);
        printf("Your score was %d\n", SCORE);
        OverwriteScore();
        system("pause");
        exit(0);
}

// Collisions
void CheckCollisions()
{
        CheckPlayer_TerrainCollisions();
        CheckPlayer_BulletCollisions();
}
void CheckPlayer_TerrainCollisions()
{
        int index;
        for (index = 0; index < NUMBER_OF_OBSTICLES; index++)
        {
                int LeftCornerX = UpperTerrain[index].verticies[0];
                int LeftCornerY = UpperTerrain[index].verticies[1];

                int RightCornerX = UpperTerrain[index].verticies[2];
                int RightCornerY = UpperTerrain[index].verticies[1];

                int tmp1 = getDistance(LeftCornerX, Player.x, LeftCornerY, Player.y);
                if (tmp1 < Player.radius)
                {
                        CollisionOccured();
                        return;
                }
                else
                        if ((getDistance(RightCornerX, Player.x, RightCornerY, Player.y) < Player.radius))
                        {
                                CollisionOccured();
                                return;
                        }

                LeftCornerX = BottomTerrain[index].verticies[0];
                LeftCornerY = BottomTerrain[index].verticies[1];

                RightCornerX = BottomTerrain[index].verticies[2];
                RightCornerY = BottomTerrain[index].verticies[1];

                tmp1 = getDistance(LeftCornerX, Player.x, LeftCornerY, Player.y);
                if (tmp1 < Player.radius)
                {
                        CollisionOccured();
                        return;
                }
                else
                        if ((getDistance(RightCornerX, Player.x, RightCornerY, Player.y) < Player.radius))
                        {
                                CollisionOccured();
                                return;
                        }
        }
}
void CheckPlayer_BulletCollisions()
{
        int index;
        for (index = 0; index < NUMBER_OF_OBSTICLES; index++)
        {
                if (getDistance(Player.x, Bullets[index].x, Player.y, Bullets[index].y) < Player.radius + 
Bullets[index].radius)
                        CollisionOccured();
        }
}
void CollisionOccured()
{
        Player.life--;
        if (Player.life == 0)
        {
                EndGame();
        }
        else
        {
                COLLISION_TIMEOUT = 70;
                printf("You got hit! Lifes remaining : %d.\n", Player.life);
        }
}

// Next level
void CheckIfNextLevel()
{
        if (Player.x >= WIDTH)
        {
                NextLevel();
        }
}
void NextLevel()
{
        GenerateTerrain();
        Player.x = 0;
        COLLISION_TIMEOUT = 20;
        int index;
        for (index = 0; index < NUMBER_OF_OBSTICLES; index++)
        {
                Bullets[index].running = 0;
                Bullets[index].y = -20;
        }
        HARDNESS -= 100;
}

// Terrain movement
void MoveTerrain()
{
        int index;
        for (index = 0; index < NUMBER_OF_OBSTICLES; index++)
        {
                if (rand() % 20 == 0)
                {
                        UpperTerrain[index].verticies[1] += rand() % 10;
                        UpperTerrain[index].verticies[3] += rand() % 10;

                        BottomTerrain[index].verticies[1] -= rand() % 10;
                        BottomTerrain[index].verticies[3] -= rand() % 10;
                }
        }
}

// Bullets
Bullet MakeBullet(int x, int y)
```

```
{
        Bullet newBullet;
        newBullet.colorR = (rand() % 255) / 255.f;
        newBullet.colorG = (rand() % 255) / 255.f;
        newBullet.colorB = (rand() % 255) / 255.f;
        newBullet.x = x;
        newBullet.y = y;
        newBullet.speed = rand() % 20;
        newBullet.radius = rand() % 15 + 5;
        newBullet.running = 0;
        return newBullet;
}
void GenerateBullets()
{
        int index;
        for (index = 0; index < NUMBER_OF_OBSTICLES; index++)
        {
                Bullets[index] = MakeBullet(index * OBSTICLE_WIDTH, -20);
        }
}
void WakeBullets()
{
        int index;
        for (index = 0; index < NUMBER_OF_OBSTICLES; index++)
        {
                if (Bullets[index].running == 0)
                {
                        if (rand() % HARDNESS == 0)
                        {
                                Bullets[index].running = 1;
                                Bullets[index].speed = rand() % 20;
                        }
                }
        }
}
void MoveBullets()
{
        int index;
        for (index = 0; index < NUMBER_OF_OBSTICLES; index++)
        {
                if (Bullets[index].running == 1)
                {
                        Bullets[index].y += speed;
                }
        }
}

// Others
int getRandom(int bottomBound, int upperBound)
{
        return rand() % (upperBound - bottomBound) + bottomBound;
}
float getDistance(int x1, int x2, int y1, int y2)
{
        int dx = x2 - x1;
        int dy = y2 - y1;
        int ddx = dx * dx;
        int ddy = dy * dy;
        return sqrt(ddy + ddx);
}
```

5. Wnioski

Wykorzystanie biblioteki OpenGL w języku C nie było łatwym zadaniem. OpenGL domyślnie pracuje w układzie, gdzie współrzędne punktu określa się w przedziale <-1, 1>, a środek renderowanego okna odpowiada koordynatom (0,0). Konieczne było użycie funkcji

```
glOrtho(0.f, WINDOW_WIDTH, WINDOW_HEIGHT, 0.f, 0.f, 1.f);
```

Uważam, że gra spełnia założenia przyjęte na etapie planowania.