Imię i Nazwisko: Malawski Adrian Podymniak Adam Marek Dominik	Kierunek: Informatyka Techniczna	Rok i grupa studiów: Rok pierwszy/ Grupa 3
Nazwa projektu: Monster Hunter		

Cel i Opis projektu:

Monster Hunter to gra typu Wave Survival. Projekt ten miał za cel nauczyć nas współpracy w grupie z wykorzystaniem kontroli wersji git oraz wzbogacenie naszych umiejętności programistycznych. Wymagało to od nas ustalenia kto odpowiada za dany fragment gry, wymiana pomysłami oraz wspólne rozwiązywanie problemów. Każdy z nas musiał poświęcić czas na naukę w jaki sposób zaimplementować daną funkcję rozgrywki, poszerzając tym wiedzę o różne dziedziny programowania, np. poznanie narzędzi jakie nam oferuje biblioteka graficzna, obsługa sterowania, podział kodu na poszczególne klasy, pobieranie i zapisywanie danych, praca z plikami z rozszerzeniem .json, z grafikami oraz z muzyką.

Zastosowane techniki programistyczne:

1. Menu, pauza i ustawienia:

Podczas tworzenia menu, używałem głównie programowania obiektowego, opartego o klasy. Działałem także na pliku .json, gdzie zapisują się i

są pobierane ustawienia wprowadzane przez użytkownika. Używałem też funkcji raylib do wczytywania i wyświetlania tekstur, a także do wczytywania i odtwarzania muzyki w menu. Textury przycisków były generowane przez AI a następnie je obrabiałem aby pasowały do projektu. Muzyka to darmowa copyright free muzyka z Youtube. Linki:

menu- https://www.youtube.com/watch?v=luGJQCJfNLI
settings - https://www.youtube.com/watch?v=p5cWMxzzMdA

2. Generator i wczytywanie map:

Edytor poziomów stworzyłem w osobnej klasie odpowiedzialnej za pobieranie i zapisywanie map oraz tworzenie obrazu. Mapy są zapisane w rozszerzeniu .json. Przełączanie się między scenami odbywa się za pomocą przypisywania do zmiennej typu enum konkretnych wartości. Jest blokada przed wprowadzaniem zbyt długich tekstów jako nazwa pliku. Gracz może sobie wybrać wielkość mapy, a następnie ta mapa jest renderowana. W

zależności od tego jaką wartość przechowuje dane pole mapy, taki kolor się wyświetla. Edytor pozwala na stawianie ścian, wskazywanie w którym miejscu ma się przeciwnik pojawić oraz kasowanie tych pól. Wczytywanie mapy odbywa się poprzez otwarcie folderu, który przechowuje poziomy, następnie pobiera wszystkie pliki, które są z rozszerzeniem .json, a na końcu wszystkie je wyświetla w postaci przycisków do wyboru. Wybranie konkretnego poziomu powoduje załadowanie gry na wybranym poziomie.

3. **Główna gra:**

   Programowanie obiektowe, Silnik Raycastowy (opisane poniżej).

Wykorzystane biblioteki:

-Raylib

-Nlohmann/json

Wybrane Funkcjonalności:

1. **Menu, pauza i ustawienia:**

   Główną funkcjonalnością w tej części kodu jest zapisywanie i wczytywanie ustawień z pliku .json i sprawdzanie przed rozpoczęciem gry, czy taki plik istnieje.

```
bool Settings::checkForFile()
    {
```

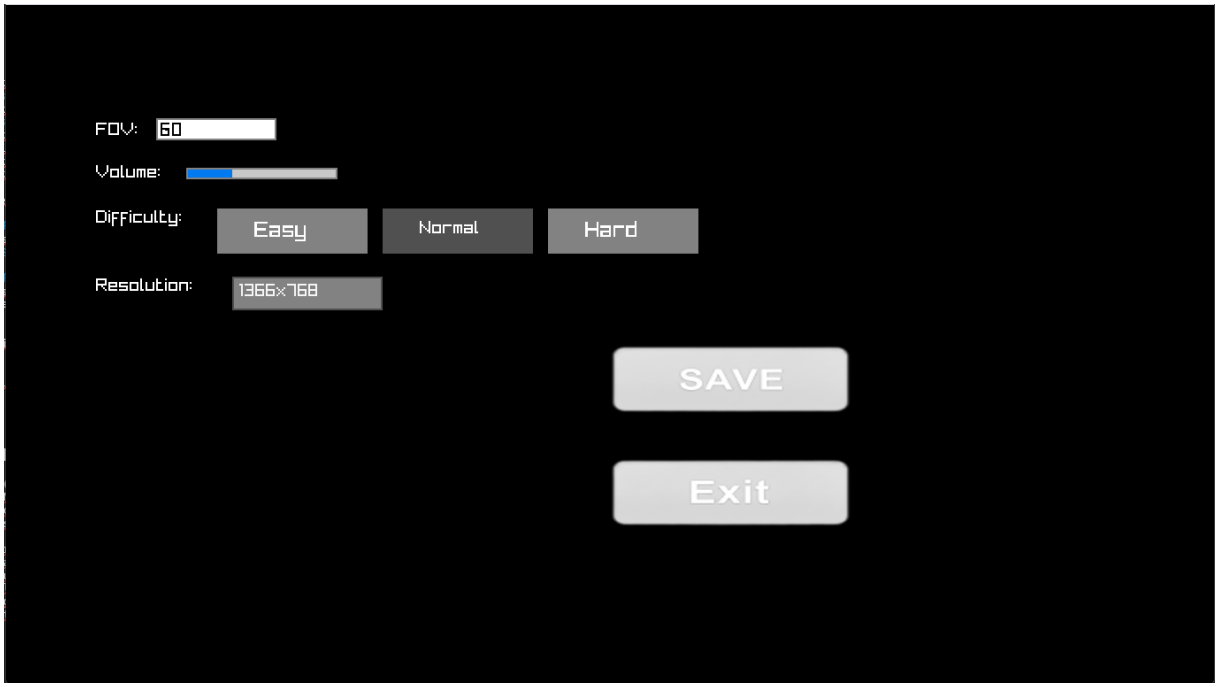
```
	if (!std::filesystem::exists(path))
	{
		std::ofstream file(path, std::ios::trunc);

		if (file.is_open())
		{
			jSet["sWidth"] = 800;
			jSet["sHeight"] = 600;
			jSet["fov"] = 60;
			jSet["difficulty"] = 1; // 0 - easy, 1 - normal 2 - hard
			jSet["volume"] = 1.0;

			file << jSet.dump(4);
			file.close();
			return true;

		} else { return false; }
	}
	return true;
}


void Settings::loadAll()
{
	Settings::checkForFile();
	if (std::filesystem::exists(path))
	{
		std::ifstream file(path);
		if (file.is_open())
		{
			file >> jSet;

			fov = jSet["fov"];
			volume = jSet["volume"];
			difficulty = jSet["difficulty"];
			resolution = { jSet["sWidth"], jSet["sHeight"] };
			scale.x = jSet["sWidth"] / 800.0;
			scale.y = jSet["sHeight"] / 600.0;
			file.close();

		}else { std::cout << "Cannot open settings files" << std::endl; }
	}
	else { std::cout << "Settings file doesn't exist" << std::endl; }

}

void Settings::save() {

	std::ofstream file(path, std::ios::trunc);

	if (file.is_open())
	{
		jSet["sWidth"] = resolution.x;
		jSet["sHeight"] = resolution.y;
		jSet["fov"] = fov==0 ? 60 : fov;
		jSet["difficulty"] = difficulty; // 0 - easy, 1 - normal 2 - hard
		jSet["volume"] = volume;

		file << jSet.dump(4);
```

```
            file.close();
}
      }
```

Kolejnymi ważnymi funkcjami są te, które wyświetlają menu i ustawienia:

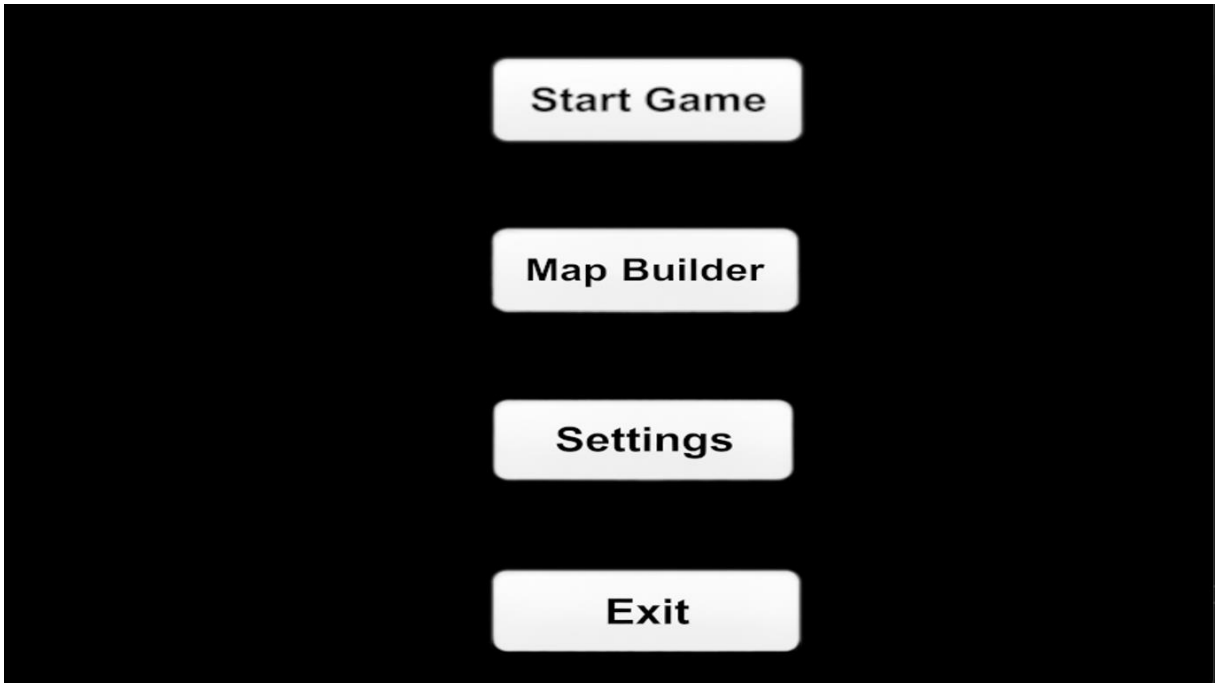
Ustawienia:

```
while (!WindowShouldClose() && exit != true)
{
UpdateMusicStream(soundtrack);
BeginDrawing();
ClearBackground(BLACK);
DrawText("FOV: ", 60*scale.x,103*scale.y,16*scale.y,WHITE);
Settings::inputText(80*scale.x, 20*scale.y, 100*scale.x, 100*scale.y, 3);
DrawText("Volume: ", 60*scale.x, 140*scale.y, 16*scale.y, WHITE);
Settings::volumeSlider(120*scale.x, 144*scale.y);
DrawText("Difficulty: ", 60*scale.x, 180*scale.y, 16*scale.y, WHITE);
Settings::diffChoose(140*scale.x,180*scale.y);
DrawText("Resolution: ", 60*scale.x , 240*scale.y, 16*scale.y, WHITE);
resolutionChoose(150*scale.x,240*scale.y);
saveButton.Draw();
exitButton.Draw();

if (timeBlock <= 0)
{
            if (saveButton.isPressed(GetMousePosition(), IsMouseButtonReleased(MOUSE_BUTTON_LEFT)))
            {Settings::save();}

            if(exitButton.isPressed(GetMousePosition(), IsMouseButtonReleased(MOUSE_BUTTON_LEFT)))
            {exit = true;}
}
else {
            timeBlock -= 1.0 / 60.0;
}

EndDrawing();
}

}
```



Menu:

```
while (!WindowShouldClose()) {
UpdateMusicStream(soundtrack);
mousePos = GetMousePosition();
isLPress = IsMouseButtonReleased(MOUSE_BUTTON_LEFT);

BeginDrawing();
ClearBackground(BLACK);
startButton.Draw();
mapBuildButton.Draw();
settingsButton.Draw();
exitButton.Draw();
   if (startButton.isPressed(mousePos,isLPress))
   {
      return 1;
   }
   if (mapBuildButton.isPressed(mousePos, isLPress))
   {
      return 2;
   }
   if (settingsButton.isPressed(mousePos, isLPress))
   {
      return 3;
   }
   if (exitButton.isPressed(mousePos, isLPress))
   {
      return 4;
   }

EndDrawing();
}
return 0;

      }
```



2. Generator i wczytywanie map:

Enum odpowiadający za przełączanie scen:

```
enum AppState { // Przełączanie scen

   select_mode,
      load_file,
      create_file,
   editor

};
```

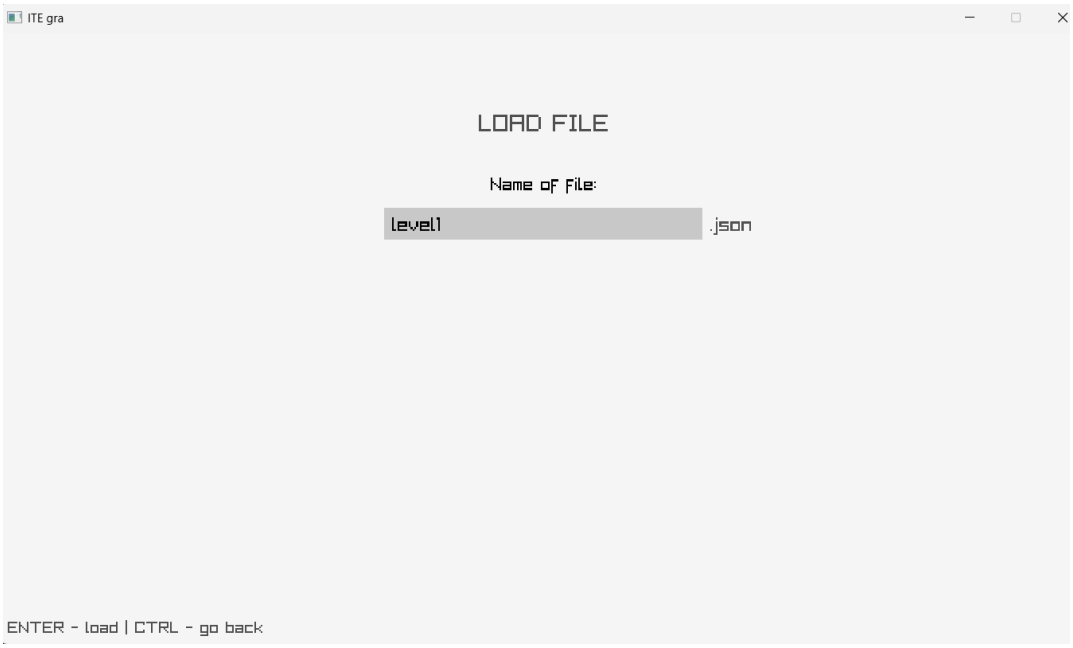
Kod odpowiedzialny za tworzenie menu

```
// Tworzenie menu

if (state == select_mode) {
   if (IsKeyPressed(KEY_ONE)) {
         filenameInput.clear();
         errorMessage.clear();
         state = load_file
if (IsKeyPressed(KEY_TWO)){
    filenameInput.clear();
      editSize = true;
  state = create_file;
      }
  if (IsKeyPressed(KEY_THREE)) {
        isActive = false;
   }
BeginDrawing();
ClearBackground(RAYWHITE);
DrawText("MAP EDITOR", (GetScreenWidth()-MeasureText("MAP EDITOR",32))/2, 100, 32,
DARKGRAY);
```

```
DrawText("1 - Load map", (GetScreenWidth() - MeasureText("1 - Load map", 24))/2, 220, 24, BLACK);
DrawText("2 - New map", (GetScreenWidth() - MeasureText("2 - New map", 24))/2, 260, 24, BLACK);
DrawText("3 - Menu", (GetScreenWidth() - MeasureText("3 - Menu", 24))/2, 300, 24, BLACK);
 EndDrawing();
 continue;
}
```

Kod odpowiedzialny za tworzenie wczytywania poziomu:

```
if (state == load_file) {
 int key = GetCharPressed();
 while (key > 0) {
 if (key >= 32 && key <= 125 && filenameInput.size() < 24)
 filenameInput += (char)key;
 key = GetCharPressed();
 }

 if (IsKeyPressed(KEY_BACKSPACE) && !filenameInput.empty())
 filenameInput.pop_back();

 if (IsKeyPressed(KEY_ENTER)) {
 std::ifstream file(basePath + filenameInput + ".json");
 if (!file.is_open()) {
 errorMessage = "This file doesn't exist!";
 }
 else {
 file >> j;
 map = j["data"];
 loadedFilename = filenameInput;
 state = editor;
 }
 }

 if (IsKeyPressed(KEY_LEFT_CONTROL))
 state = select_mode;

 BeginDrawing();
 ClearBackground(RAYWHITE);
 DrawText("LOAD FILE", (GetScreenWidth()-MeasureText("LOAD FILE", 30))/2, 100, 30, DARKGRAY);
 DrawText("Name of file:", (GetScreenWidth() - MeasureText("Name of file:", 22))/2, 180, 22, BLACK);
 DrawRectangle((GetScreenWidth()/2) - 200, 220, 400, 40, LIGHTGRAY);
   DrawText(filenameInput.c_str(), (GetScreenWidth() / 2)-190, 230, 24, BLACK);
   DrawText(".json", (GetScreenWidth() / 2)+210, 230, 24, DARKGRAY);

   if (!errorMessage.empty())
      DrawText(errorMessage.c_str(), 300, 470, 22, RED);

   DrawText("ENTER - load | CTRL - go back", 10, GetScreenHeight()-30, 20, DARKGRAY);
  EndDrawing();
   continue;
}
```



Kod odpowiedzialny za tworzenie nowego poziomu:

```
if (state == create_file) {


   int key = GetCharPressed();
   while (key > 0) { // Zabezpieczenie się na poprawność nazwy dokumentu
      if (key >= 32 && key <= 125 && filenameInput.size() < 24)
         filenameInput += (char)key;
      key = GetCharPressed();
   }

   if (IsKeyPressed(KEY_BACKSPACE) && !filenameInput.empty()) { // Wykasowanie tekstu
```

```
            filenameInput.pop_back();
        }

        // Edycja szerokości lub wysokości nowego pola
        if (IsKeyPressed(KEY_UP) || IsKeyPressed(KEY_DOWN)) {
            editSize = !editSize;
        }

        // Nadanie maksymalnych wymiarów mapy
        if (IsKeyPressed(KEY_RIGHT)) {
            if (editSize && newWidth < 20) newWidth++;
            if (!editSize && newHeight < 20) newHeight++;
        }

        // Nadanie minimalnych wymiarów mapy
        if (IsKeyPressed(KEY_LEFT)) {
            if (editSize && newWidth > 5) newWidth--;
            if (!editSize && newHeight > 5) newHeight--;
        }

        if (IsKeyPressed(KEY_ENTER) && !filenameInput.empty()) {
            map.assign(newHeight, std::vector<int>(newWidth, 0));
            loadedFilename = filenameInput;
            state = editor;
        }

        if (IsKeyPressed(KEY_LEFT_CONTROL)) {
            state = select_mode;
        }
        BeginDrawing();
        ClearBackground(RAYWHITE);
        DrawText("NEW MAP", (GetScreenWidth()-MeasureText("NEW MAP", 30))/2, 100, 30,
DARKGRAY
        DrawText("Name of file:", (GetScreenWidth()-MeasureText("Name of file:", 22))/2, 180, 22,
BLACK);
        DrawRectangle((GetScreenWidth()/2)-200, 220, 400, 40, LIGHTGRAY);
        DrawText(filenameInput.c_str(), (GetScreenWidth()/2)-190, 230, 24, BLACK);
        DrawText(TextFormat("Width: %d %s", newWidth, editSize ? "<" : ""), (GetScreenWidth()-
MeasureText("Width: 10 <",24))/2, 430, 24, BLACK);
        DrawText(TextFormat("Height: %d %s", newHeight, !editSize ? "<" : ""), (GetScreenWidth() -
MeasureText("Height: 10 <", 24)) / 2, 470, 24, BLACK);
        DrawText("UP/DOWN - Change option | LEFT/RIGHT - Change value", 10, GetScreenHeight()-60,
18, DARKGRAY);
        DrawText("ENTER - create | CTRL - go back", 10, GetScreenHeight()-30, 18, DARKGRAY);
        EndDrawing();
        continue;
    }
```

Kod odpowiedzialny za wygląd edytora:

```
        // Tworzenie edytora
        int rows = map.size();
        int cols = map[0].size();
```

```
    int mapWidthPx = cols * cellSize;
    int mapHeightPx = rows * cellSize;

    int offsetX = (GetScreenWidth() - mapWidthPx) / 2;
    int offsetY = (GetScreenHeight() - mapHeightPx) / 2;

    Vector2 mouse = GetMousePosition();

    if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON) ||
        IsMouseButtonPressed(MOUSE_RIGHT_BUTTON)) {

        int col = (mouse.x - offsetX) / cellSize;
        int row = (mouse.y - offsetY) / cellSize;

        if (row >= 0 && row < rows && col >= 0 && col < cols) { // Zamiania wartości pola po
naciśnięciu na nią
            if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON))
                map[row][col] = (map[row][col] == 2) ? 0 : 2;
            if (IsMouseButtonPressed(MOUSE_RIGHT_BUTTON))
                map[row][col] = (map[row][col] == 5) ? 0 : 5;
        }
    }

    if (IsKeyPressed(KEY_ENTER)) { // Zapisywanie mapy
        json out;
        out["data"] = map;
        std::ofstream file(basePath + loadedFilename + ".json");
        file << out.dump(4);
    }

    if (IsKeyPressed(KEY_LEFT_CONTROL))
        state = select_mode;

    BeginDrawing();
    ClearBackground(RAYWHITE);

    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            Color col = LIGHTGRAY;
            if (map[r][c] == 5) col = RED;
            if (map[r][c] == 2) col = GREEN;

            DrawRectangle(offsetX + c * cellSize,
                offsetY + r * cellSize,
                cellSize – 2,
                cellSize – 2,
                col);
        }
    }

    DrawText("ENTER - save | CTRL - go back", 10, GetScreenHeight()-60, 20, DARKGRAY);
    DrawText(TextFormat("Plik: %s.json", loadedFilename.c_str()), 10, GetScreenHeight()-30, 20,
DARKGRAY);

    EndDrawing();
}
```

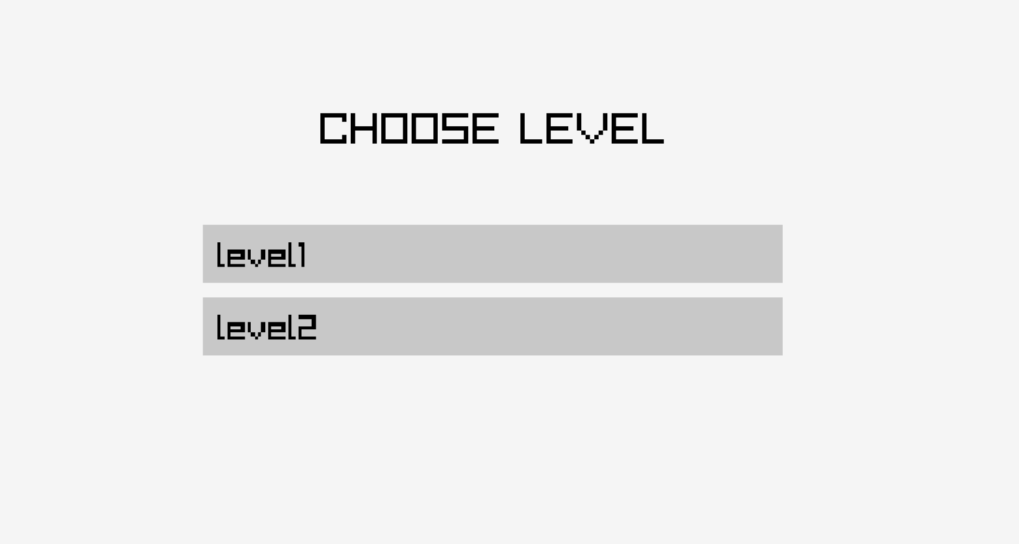
Pobranie poziomów:

```
void Game::loadFiles() {

for (const auto& entry std::filesystem::directory_iterator("Resources/Levels
 {
 if (!entry.is_regular_file() && (entry.path().extension() != ".json")) {
 continue;
 }

 std::ifstream file(entry.path());
 if (!file.is_open())
 continue;

 json j;
      file >> j;

      if (!j.contains("data") || !j["data"].is_array()) {
         continue;
      }

      Level level;
      level.name = entry.path().stem().string();
      level.map = j["data"];

      levels.push_back(level);
   }

}
```

Wyświetlanie listy z poziomami:

```
void Game::showLevelsList() {
   levels.clear();
   loadFiles();
   while (!WindowShouldClose()) {

      Vector2 mouse = GetMousePosition();
      bool isLevelPicked = false;

      BeginDrawing();

      ClearBackground(RAYWHITE);
      for (int m = 0; m < levels.size(); m++) {
         Rectangle button = { (float)(GetScreenWidth() / 2) - 200, 180.0f + (m * 50), 400.0f, 40.0f };
         bool hovered = CheckCollisionPointRec(mouse, button);
         if (hovered && IsMouseButtonPressed(MOUSE_LEFT_BUTTON
{
            selectedLevel = m;
            DrawRectangleRec(button, DARKGRAY);
            isLevelPicked = true;
         }
         else {
            DrawRectangleRec(button, LIGHTGRAY);
         }
         DrawText(levels[m].name.c_str(), button.x + 10, button.y + 10, 24, BLACK);
```

```
        }
        DrawText("CHOOSE LEVEL", (GetScreenWidth() - MeasureText("CHOOSE LEVEL", 30)) / 2, 100,
30, BLACK);
        EndDrawing();
        if (isLevelPicked) break;
    }
}
```



3. Główna gra:

W tej sekcji wytłumaczę działanie najważniejszych funkcji odpowiedzialnych za silnik Raycastingowy, oraz za wszystkie mechaniki gry wchodzące w podstawowy Game Loop, takie jak: system fal, przeciwników, strzelania, poruszania się, punktów i upgradeów.

1)Raycasting

Silnik Raycastowy polega na rysowaniu z pozycji kamery linii, sprawdzających jak daleko od nich znajduje się ściana. Dystans ten następnie zamieniany jest na cieniowanie i na wysokość tekstury ściany, czy innych spriteów w świecie. Tworzy to iluzję świata 3D w środowisku 2D. Technikę tę stosowały wszystkie najstarsze gry komputerowe, takie jak Wolfenstein 3D, czy DOOM. Technika ta jest bardzo mało wymagająca, dlatego nie potrzebowała akceleracji hardwareowej, jak inne gry używające 3D (takie jak Quake). Rozwój technologii uczynił jednak ten sposób renderowania 3D bezużytecznym, gdyż każdy komputer jest w stanie udźwignąć renderowanie przez poligony. Współczesne karty graficzne są w stanie wywoływać miliardy prostych operacji na tych geometrycznych trójkątach w jednym momencie. Jednak warto jest spojrzeć na stare sposoby renderowania 3D przed wyruszeniem na trudniejsze projekty, by zrozumieć ograniczenia i problemy, jakie niesie za sobą ten proces.

Największym problemem jaki napotkałem przy kodowaniu tej funkcji był tzw Fish Eye Effect, czyli efekt zakrzywiania się ścian przy podejściu do nich blisko (pewna wersja tego błędu nadal znajduje się w grze). Tak samo renderowanie tekstury podłogi okazało się ogromnym wyzwaniem, dlatego w tym projekcie podłogę stanowi szary kolor, bez tekstury.

Kod:

```
void Game::rayCastingFunc() {
    const std::vector<std::vector<int>>& currentMap =
levels[currentLevel].map; // wybieranie obecnej mapy
    int rayPrecision = rayCasting.precision; //precyzja (im większa, tym
wolniej działa, ale jest dokładniejsza)
    int projectionHalfHeight = projection.halfHeight; // środek ekranu
(przydatny potem w kodzie)

    double rayAngle = player.angle - player.halfFov; // obsługa kątów rayów
(by odchaczyć renderowanie całego FOV)
    double angleIncrement = rayCasting.incrementAngle;

    double playerX = player.x; // pozycja gracza
    double playerY = player.y;

    for (int rayCount = 0; rayCount < projection.width; rayCount++) { //
rayCount -> koordynat x
        double rayAngleRad = degreeToRadians(rayAngle); // przesuwanie raya o
mały kawałek, by pokryć całe FOV
        double rayCos = cos(rayAngleRad) / rayPrecision;
        double raySin = sin(rayAngleRad) / rayPrecision;

        double rayX = playerX; // ray zaczyna się w pozycji gracza
        double rayY = playerY;

        int wall;
        do { // powolne przesuwanie raya, aż nie dotrze do ściany
            rayX += rayCos;
            rayY += raySin;
            int mapX = (int)rayX;
            int mapY = (int)rayY;

            if (mapY >= 0 && mapY < (int)currentMap.size() && // wyjście poza
mapę -> dotknięcie ściany (by nie crashowało)
                mapX >= 0 && mapX < (int)currentMap[mapY].size()) {
                wall = currentMap[mapY][mapX];
            }
            else {
                wall = 2;
            }
        } while (wall != 2);

        double dx = rayX - playerX;
        double dy = rayY - playerY;
```

```
        double distance = sqrt(dx * dx + dy * dy) *
cos(degreeToRadians(rayAngle - player.angle)); // naprawianie efektu Fish Eye
(częściowo działa).

        int wallHeight = (int)(projectionHalfHeight / distance); // obliczanie
wysokości ściany wg dystansu od gracza

        for (int y = 0; y < projectionHalfHeight - wallHeight; y++) { //
rysowanie skyboxa
            int skyX = (int)((rayAngle / player.fov) * skyTexture->width) %
skyTexture->width;
            int skyY = (int)((float)y / projection.halfHeight * skyTexture-
>height);
            drawPixel(rayCount, y, skyTexture->pixels[skyY * skyTexture->width
+ skyX]);
        }

        drawTexturedWall(rayCount, wallHeight, rayX, rayY); // teksturowanie
ścian

        for (int y = projectionHalfHeight + wallHeight; y < projection.height;
y++) { // rysowanie podłogi
            drawPixel(rayCount, y, Color{ 50, 50, 50, 255 });
        }

        rayAngle += angleIncrement; // inkrementacja raya
    }
}
```

2)Tekstury

Sam silnik jest w stanie wygenerować proste, jednokolorowe bloki, jednak do pełnej obsługi tekstur trzeba napisać własną funkcję. Teksturowanie dzieje się na dwa sposoby: teksturowanie ścian i renderowanie spriteów. Pierwszy proces polega na wycinaniu z pliku graficznego kawałka o odpowiedniej długości i wklejania go na odpowiadający jej kawałek ściany, co widać w zamieszczonym kodzie:

```
void Game::drawTexturedWall(int rayCount, int wallHeight, double rayX, double
rayY) {
    if (!wallTexture || !wallTexture->pixels) return;

    double hitX = rayX - floor(rayX);
    double hitY = rayY - floor(rayY);

    int texX;
    if (hitX < 0.01 || hitX > 0.99)
```

```
        texX = (int)(hitY * wallTexture->width);
    else
        texX = (int)(hitX * wallTexture->width);

    texX = std::clamp(texX, 0, wallTexture->width - 1);

    float texStep = (float)wallTexture->height / (wallHeight * 2);
    float texPos = 0;

    int startY = projection.halfHeight - wallHeight;
    if (startY < 0) startY = 0;
    int endY = projection.halfHeight + wallHeight;
    if (endY > projection.height) endY = projection.height;

    for (int y = startY; y < endY; y++) {
        int texY = (int)texPos & (wallTexture->height - 1);
        int index = texY * wallTexture->width + texX;

        drawPixel(rayCount, y, wallTexture->pixels[index]);
        texPos += texStep;
    }
}
```

Renderowanie Spriteów natomiast polega na wyświetlaniu całego obrazu z pliku graficznego, jeżeli tylko odpowiednie kryteria są spełnione. Obraz skalowany jest względem dystansu od gracza. Kod sprawdza, czy Sprite nie jest za ścianą, czy nie jest za blisko, czy jest z przodu, czy z tyłu gracza. Oto kod (w tym przypadku kod na Spritey potworów, ponieważ do nich wymagane były dodatkowe funkcje. Jednakowy mechanizm działa przy renderowaniu pickupów i kul):

```
void Game::drawSpriteInWorld(const Monster& monster) {
    if (!monster.texture || monster.texture->texture.id == 0 ||
monster.isDead) return; // sprawdzanie, czy sprite istnieje i czy potwór żyje

    if (!isVisibleToPlayer(monster)) return; // raycastowanie, by sprawdzić,
czy nie ma ścian

    double relX = monster.x - player.x; // pozycja relatywna od gracza
    double relY = monster.y - player.y;

    double distance = sqrt(relX * relX + relY * relY); // obliczanie
odległości (do skalowania)
    if (distance < 0.5) return; // sprawdzanie, czy jest za blicko

    double angleToMonster = atan2(relY, relX);
    double angleDiff = angleToMonster - degreeToRadians(player.angle);
```

```
    while (angleDiff > PI) angleDiff -= 2.0 * PI;
    while (angleDiff < -PI) angleDiff += 2.0 * PI;

    double halfFovRad = degreeToRadians(player.halfFov);
    if (angleDiff < -halfFovRad || angleDiff > halfFovRad) return; //
sprawdzanie, czy poza FOV

    double screenXProj = (angleDiff + halfFovRad) / (2.0 * halfFovRad) *
projection.width; // zmiana przedziału do obliczeń

    double screenX = screenXProj * screen.scale;
    double screenY = projection.halfHeight * screen.scale;
    //obliczanie wysokosci i szerokości
    int monsterHeight = (int)(projection.halfHeight / distance);
    int monsterWidth = (int)((double)monster.texture->width / monster.texture-
>height * monsterHeight);

    monsterHeight *= screen.scale;
    monsterWidth *= screen.scale;

    Rectangle src = {
        0, 0,
        (float)monster.texture->width,
        (float)monster.texture->height
    };

    Rectangle dst = {
        (float)(screenX - monsterWidth / 2),
        (float)(screenY - monsterHeight / 2),
        (float)monsterWidth,
        (float)monsterHeight
    };
    // rysowanie tekstury
    DrawTexturePro(monster.texture->texture, src, dst, { 0, 0 }, 0, WHITE);
}
```

Największym problemem w tej sekcji okazało się renderowanie pocisków. Do teraz nie mam pewności co do tego, jaka linijka kodu uniemożliwia wyświetlanie oteksturowanych kul, zamiast jednokolorowych prostokątów.

3)System fal

System fal jest jednym z najprostszych algorytmów w całym kodzie gry. Ma on dwie główne funkcje: przelosować pola spawnu przeciwników i wylosować przeciwników do danej fali. Z każdą falą zwiększa się trudność przeciwników, jak i ich ilość. Rośnie też szansa na

pojawienie się trudniejszych przeciwników w większych ilościach. Cała implementacja nie sprawiła mi jakichkolwiek problemów. Oto kod:

```
void Game::spawnWave()
{
    monsters.clear(); //czyszczenie przeciwników, jakby przez przypadek jakiś został
    monsterDefeated = 0;
    monsterTotal = 0;

    int spawnCount = enemiesPerWave + currentWave; //wyznaczanie trudności fali

    //losowanie tilei, tak, by przeciwnicy nie spawnowali się zawsze w tym samym miejscu
    std::vector<int> spawnIndices(enemySpawnPoints.size());
    std::iota(spawnIndices.begin(), spawnIndices.end(), 0);

    for (int i = spawnIndices.size() - 1; i > 0; --i)
    {
        int j = rand() % (i + 1);
        std::swap(spawnIndices[i], spawnIndices[j]);
    }

    int usedTiles = 0;

    //wybieranie jacy przeciwnicy zrespią się w danej fali
    for (int i = 0; i < spawnCount && usedTiles < spawnIndices.size();)
    {
        int roll = rand() % 3;
        int spawnIndex = spawnIndices[usedTiles];

        if (roll == 0 && spawnCount - i >= 3 && usedTiles + 3 <= spawnIndices.size())
        {
            // kreowanie przeciwnika ze struktury (dla mojej wygody i by opcjonalnie dało się ich dodawać więcej)
            Monster wolf;
            wolf.id = "monster_" + std::to_string(i);
            wolf.type = "wolf";
            wolf.skin = "wolf";
            wolf.audio = "bigcat";
            wolf.x = enemySpawnPoints[spawnIndex].x;
            wolf.y = enemySpawnPoints[spawnIndex].y;
            wolf.health = 10;
            wolf.isDead = false;
            wolf.width = 512;
```

```
            wolf.height = 512;
            wolf.damage = 10 + currentWave;
            wolf.attackCooldown = 0.5;
            wolf.moveSpeed = 0.06;
            wolf.texture = wolfTexture;

            monsters.push_back(wolf);
            usedTiles += 3;
            i += 3;
        }
        else if (roll == 1 && spawnCount - i >= 5 && usedTiles + 5 <=
spawnIndices.size())
        {
            /*Tworzenie Demona*/
        }
        else
        {
            /*Tworzenie szkieleta*/
        }

        monsterTotal++;
    }

    waveActive = true;
}
```

4)Movement gracza

Movement gracza przedstawię wyłącznie na wycinku kodu, gdyż jest on powtarzalny i zająłby niepotrzebnie dużo miejsca w sprawozdaniu:

```
if (keys["up"].active) {
        double playerCos = cos(degreeToRadians(player.angle)) *
player.speed.movement;
        double playerSin = sin(degreeToRadians(player.angle)) *
player.speed.movement;
        double newX = player.x + playerCos;
        double newY = player.y + playerSin;

        int checkX = (int)(newX + playerCos * player.radius);
        int checkY = (int)(newY + playerSin * player.radius);
        int mathfloorX = (int)player.x;
        int mathfloorY = (int)player.y;

        if (checkY >= 0 && checkY < mapHeight && mathfloorX >= 0 && mathfloorX
< mapWidth &&
```

```
            map[checkY][mathfloorX] != 2) {
            player.y = newY;
        }
        if (mathfloorY >= 0 && mathfloorY < mapHeight && checkX >= 0 && checkX
< mapWidth &&
            map[mathfloorY][checkX] != 2) {
            player.x = newX;
        }
    }
```

Oblicza on na podstawie kątu gracza w jaki sposób można poruszać się w danym kierunku. Ogranicza go przy ścianach, nadaje prędkość. Największym problemem było dla mnie zachowanie jednakowego sterowania pod różnymi kontami. Czasami strzałka do przodu przesuwała mnie do przodu, a czasami w tył. Wyliczanie tego przez sinusi i cosinusy umożliwiło mi ominięcie tego problemu i stworzenie w pełni działającego poruszania się.

5)Strzelanie

Jest to fragment funkcji UpdateGameObject(). Kod iteruje przez wszystkich przeciwników i przez wszystkie pociski, sprawdzając, czy weszły ze sobą w kolizję. Jeżeli tak, to przeciwnik dostaje obrażenia zależne od typu pocisku. Jeżeli uderzyła go rakieta, aktywuje się też funkcja eksplozji, która sprawdza, czy inni przeciwnicy blisko niej też nie powinni dostać obrażeń. Odgrywane są też dźwięki bólu i śmierci. Oto kod:

```
if (projectile.owner == "player") {
            for (size_t j = 0; j < monsters.size(); j++) {
                Monster& monster = monsters[j];
                if (!monster.isDead) {
                    double dx = monster.x - projectile.x;
                    double dy = monster.y - projectile.y;
                    double distanceSq = dx * dx + dy * dy;

                    if (distanceSq < bulletHitboxRadius * bulletHitboxRadius)
{
                        if (projectile.type == "rocket") {
                            explodeRocket(projectile);
                            projectilesToRemove.push_back(i);
                        }
                        else if (projectile.type == "pistolBullet") {
                            monster.health -= 50;
                        }
                        else {
                            monster.health -= 25;
                        }

                        if (monster.health <= 0) {
                            monster.isDead = true;
```

```
                        if(projectile.type != "rocket") monsterDefeated++;
                        awardPoints(monster);
                        playSound(monster.audio + "-death");
                        int roll = rand() % 4;
                        if (roll == 0 || roll == 1) {
                            Sprite ammoSprite("ammo", monster.x,
monster.y, 100, 81);
                            ammoSprite.texture = ammoTexture;
                            sprites.push_back(ammoSprite);
                        }
                        else if (roll == 2) {
                            Sprite rocketSprite("rocketammo", monster.x,
monster.y, 35, 18);
                            rocketSprite.texture = rocketammoTexture;
                            sprites.push_back(rocketSprite);
                        }
                    }
                    else {
                        int rnd = rand() % 3;
                        if (rnd == 0) playSound(monster.audio + "-pain-
1");
                        else if (rnd == 1) playSound(monster.audio + "-
pain-2");
                        else playSound(monster.audio + "-pain-3");
                    }

                    projectilesToRemove.push_back(i);
                    break;
                }
            }
```

6)Punkty i Upgradey

Same funkcje odpowiadające na zbieranie punktów i za kupywanie upgradeów też są proste. Opis ich działania wydaje mi się zbędny, gdyż sam kod jest banalny w zrozumieniu. Oto on:

```
void Game::awardPoints(const Monster& monster) {
    if (monster.type == "skeleton") points += 10;
    else if (monster.type == "wolf") points += 20;
    else if (monster.type == "demon") points += 50;
}

void Game::openUpgradeMenu() {
    while (!WindowShouldClose()) {
        BeginDrawing();
        ClearBackground(BLACK);
```

```
        DrawText("UPGRADES", screen.width / 2 - 80, 50, 30, WHITE);

        DrawText("1 - Heal +50 HP (30 pts)", 100, 150, 20, WHITE);

        if (upgradeRocketBought) {
            DrawCrossedText("2 - Unlock Rocket Launcher (100 pts)", 100, 180,
20, GRAY);
        }
        else {
            DrawText("2 - Unlock Rocket Launcher (100 pts)", 100, 180, 20,
WHITE);
        }

        if (upgradeSpeedBought) {
            DrawCrossedText("3 - Increase Speed (50 pts)", 100, 210, 20,
GRAY);
        }
        else {
            DrawText("3 - Increase Speed (50 pts)", 100, 210, 20, WHITE);
        }

        if (upgradeDashBought) {
            DrawCrossedText("4 - Unlock Dash (75 pts)", 100, 240, 20, GRAY);
        }
        else {
            DrawText("4 - Unlock Dash (75 pts)", 100, 240, 20, WHITE);
        }



        if (IsKeyPressed(KEY_ONE) && points >= 30) {
            player.health = std::min(player.health + 50, player.maxHealth);
            points -= 30;
            playSound("pickup");
        }

        if (IsKeyPressed(KEY_TWO) && !upgradeRocketBought && points >= 100) {
            rocketUnlocked = true;
            upgradeRocketBought = true;
            points -= 100;
            playSound("pickup");
        }

        if (IsKeyPressed(KEY_THREE) && !upgradeSpeedBought && points >= 50) {
            player.speed.movement += 0.02;
            upgradeSpeedBought = true;
            points -= 50;
            playSound("pickup");
```

```
        }

        if (IsKeyPressed(KEY_FOUR) && !upgradeDashBought && points >= 75) {
            dashUnlocked = true;
            upgradeDashBought = true;
            points -= 75;
            playSound("pickup");
        }
        if (IsKeyPressed(KEY_ESCAPE)) break;

        EndDrawing();
    }
}
```

Link do GitHub i ReadMe:

https://github.com/AdamPodymniak1/ITE-gra.git