MONSTER HUNTER

Autorzy:	Kierunek:	Rok studiów i grupa:	Temat prezentacji:
Adam Podymniak Adrian Malawski Dominik Marek	Informatyka Techniczna	1. Rok, 3. grupa	Projekt gry Wave Survival - “Monster Hunter”

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.

Menu - Główna pętla

```
while (!WindowShouldClose() && keepRunning)
{
menuState = game.menu();
switch (menuState)
{
case 1: //Start game
    game.showLevelsList();
    game.resetGame();
    game.run();
    break;
case 2: // Map editor
    editor.open();
    break;
case 3: //Settings
    settings.open();
    settings.reload();
    break;
case 4: // exit
    keepRunning = false;
    break;
default:
    std::cout << "Menu Case switch Error" << std::endl;
}
}
```

Główna pętla menu wywołuję funkcję game.menu(), która w zależności od tego, co wybierzemy, zwraca konkretną wartość, i na tej podstawie uruchamia nam kolejne części kodu. Takie zastosowanie pozwala nam potem wrócić do menu z poziomu innych części kodu.

Menu - Wyświetlanie graficzne

```
PlayMusicStream(soundtrack);
soundtrack.looping = true;

    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();
```

Tutaj wyświetlają się przyciski, które są tworzone przy pomocy osobnej klasy Button. Obiekty klasy Button mają swoje tekstury i swoje koordynaty, a także metody obsługujące ich rysowanie i to czy są wciśnięte.

```
void Button::Draw()
{
    DrawTextureV(texture, position, WHITE);
}

bool Button::isPressed(Vector2 mousePos, bool mousePressed)
{
    Rectangle rect = { position.x, position.y,texture.width,texture.height};
    if (CheckCollisionPointRec(mousePos, rect) && mousePressed) {
        return true;
    }
    return false;
}
```

Ustawienia - Wyświetlanie

```
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();
```

Ustawienia mają swoją osobną klasę Settings, a tutaj jest przedstawiony fragment jej metody open(), tutaj są wyświetlane ustawienia. Używam tutaj także obiektów klasy Button do zapisywania i wychodzenia z ustawień.

Ustawienia - Zapisywanie i odczyt z pliku

```
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 -
        jSet["volume"] = volume;

        file << jSet.dump(4);

        file.close();
    }
}
```

Metoda save() zapisuje ustawienia do pliku .json używając słynnej biblioteki do obsługi plików .json, nlohmann/json

Metoda loadAll() najpierw sprawdza czy plik z ustawieniami istnieje i czy można go otworzyć, a następnie przypisuje odpowiednie wartości do zmiennych klasy Settings. Także używa nlohmann/json

```
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; }

}
```

Pauza - Wyświetlanie i logika

```
while (!WindowShouldClose())
{
    mousePos = GetMousePosition();
    isLPressed = IsMouseButtonPressed(MOUSE_LEFT_BUTTON);


BeginDrawing();
ClearBackground(BLACK);
returnButton.Draw();
settingsButton.Draw();
exitButton.Draw();



if (timeBlock <= 0)
{
    if (returnButton.isPressed(mousePos, isLPressed))
    {
        return false;
    }
    if (settingsButton.isPressed(mousePos, isLPressed))
    {
        settings.open();
        settings.reload();
    }
    if (exitButton.isPressed(mousePos, isLPressed))
    {
        return true;
    }

    if (IsKeyDown(KEY_ESCAPE))
    {
        return false;
    }
}
else {
    timeBlock -= 1.0 / 60.0;
}
EndDrawing();
}
```

Pauza w grze działa na identycznej zasadzie co menu, tylko zamiast zwracać wartości zwraca jedynie True, gdy trzeba wyjść z gry do menu, lub False, gdy pauza ma nas spowrotem prowadzić do gry. Jest też opcja zmiany ustawień, lecz na bieżąco można zmieniać jedynie głośność, ze względu na limity silnika. Reszta ustawień zmieni się po wyjściu i wejściu do gry.

DZIAŁANIE EDYTORA POZIOMÓW - WYBÓR OPCJI

```
enum AppState { // Przełączanie scen
    select_mode,
    load_file,
    create_file,
    editor
};
```

Przełączanie się między konkretnymi scenami w edytorze odbywa się za pomocą AppState. W zależności od tego jaką wartość przechowuje zmienna, taka scena jest renderowana.

W tej części tworzone jest menu. Po wybraniu 1 przechodzimy do ekranu z wybraniem poziomu, 2 pozwala nam utworzyć nowy dokument, a 3 powoduje powrót do 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();
```

DZIAŁANIE EDYTORA POZIOMÓW - NOWA MAPA

```
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;
}
```

W tej części kodu znajduje się obsługa wprowadzania poprawnych znaków do nazwy nowego pliku oraz zabezpieczenie się przed tworzeniem zbyt długiej nazwy. Oprócz tego jest przełączanie się między wyborem szerokości i wysokości mapy oraz ustalenie wielkości planszy od 5 do 20 kratek. W momencie naciśnięcia enter do zmiennej map są zapisywane wymiary mapy, które potem będą wygenerowane.

DZIAŁANIE EDYTORA POZIOMÓW - RENDER PLANSZY

```
// 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;
```

Wymiary mapy są zapisywane do zmiennych. Następnie pobierane są współrzędne myszki w momencie naciśnięcia lewego lub prawego przycisku. Naciśnięcie powoduje zmianę wartości z pustej kratki na ścianę lub pole na którym mogą tworzyć się przeciwnicy. Po naciśnięciu enter zmodyfikowana mapa zostaje zapisana do pliku.

DZIAŁANIE EDYTORA POZIOMÓW - RENDER PLANSZY CD.

```
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();
```

Ta część kodu odpowiada za rysowanie wszystkich kratek, które zostały pobrane z pliku. Jeżeli pole przechowuje wartość 2, to rysuje ją na zielono (tzn. ścianę), natomiast jak pole przechowuje 5, to rysuje ją na czerwono (pole na którym przeciwnicy się mogą pojawić).

LISTA POZIOMÓW - 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);
    }

}
```

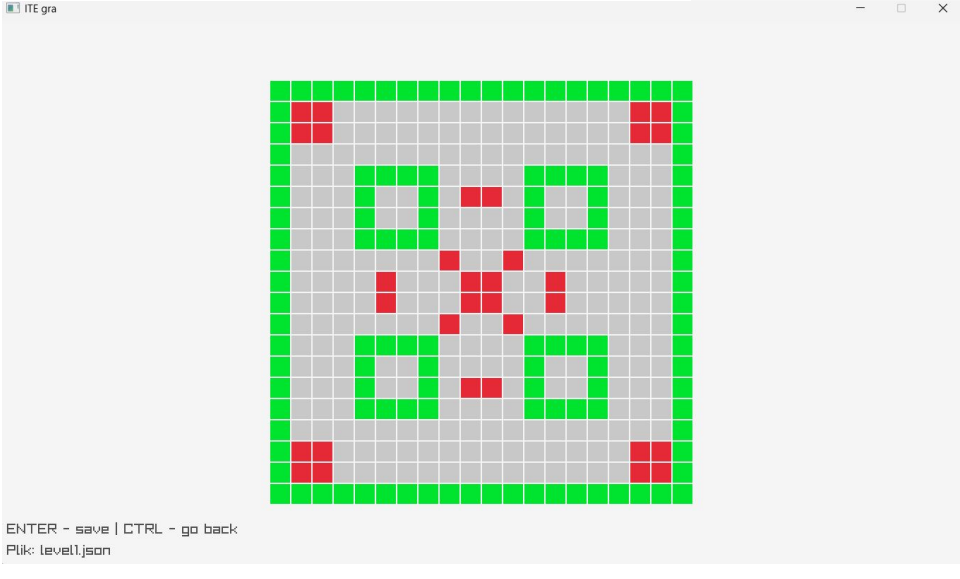
Aby móc wyświetlić listę z poziomami trzeba najpierw pobrać wszystkie pliki jakie się znajdują w pliku Levels/. Jeżeli program znajdzie jakiekolwiek inne pliki z innym rozszerzeniem niż .json, to pomija je. Następnie tworzy obiekt Level i zapisuje go do zmiennej levels.

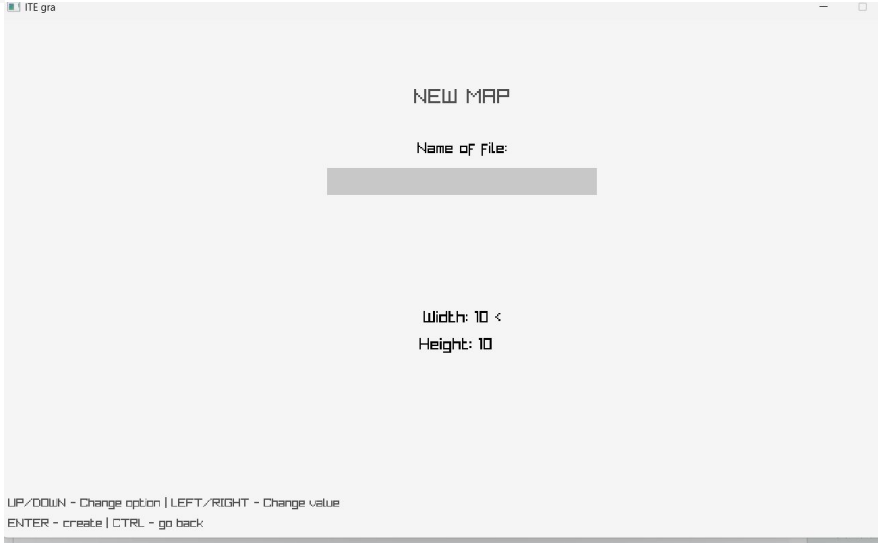
LISTA POZIOMÓW - WYŚWIETLENIE LISTY

```
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;
    }
}
```

W tej części kodu dochodzi do wyczyszczenia zmiennej levels, następnie znowu pobieramy pliki z folderu. Jest to tak zrobione żeby lista mogła się aktualizować zaraz po tym jak wychodzimy z edytora. Następnie sprawdzane jest w którym miejscu gracz nacisnął przycisk myszki i na podstawie tego jest pobrany poziom i załadowany. Dodatkowo jest dodana zmiana koloru przycisku w momencie naciśnięcia na niego.

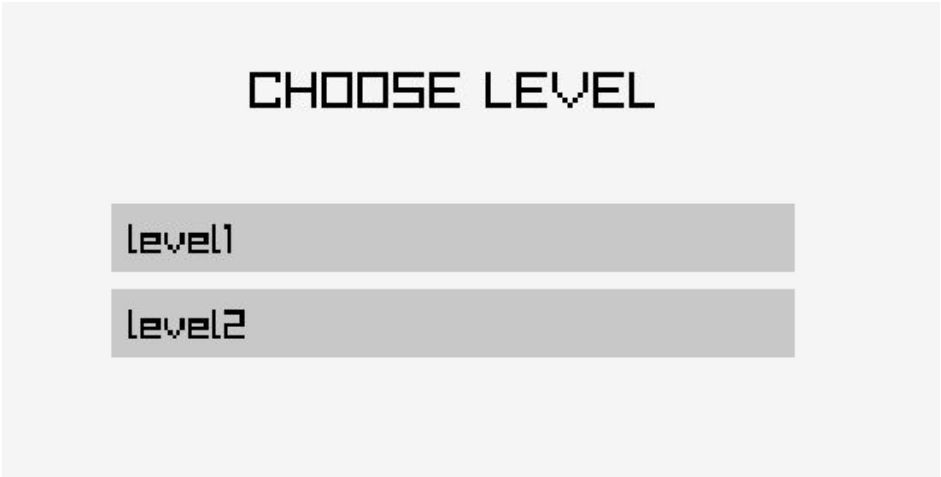
EDYTOR - ZRZUTY EKRANU



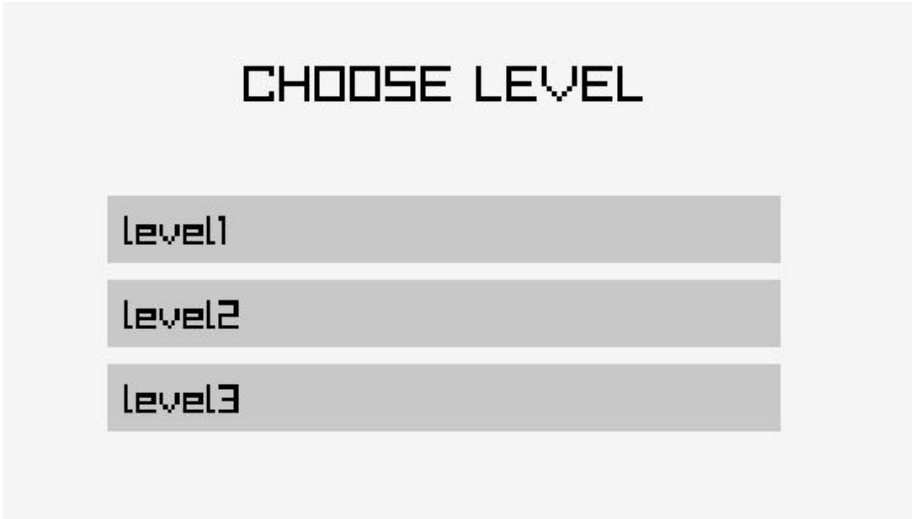


LISTA POZIOMÓW - ZRZUT EKRANU

Początkowa lista poziomów:

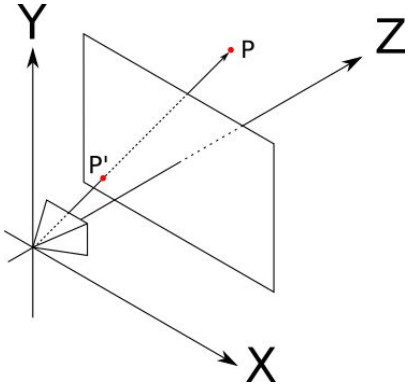


Lista poziomów po dodaniu nowego poziomu:



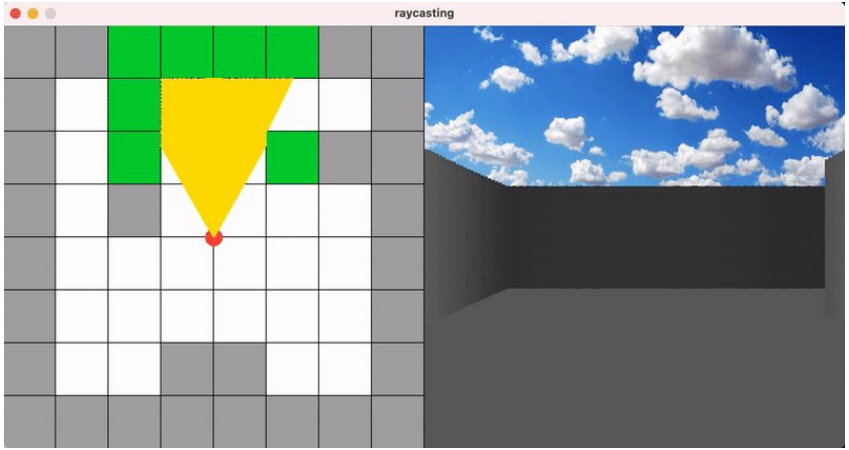
RÓŻNE RODZAJE RENDEROWANIA 3D

PROJECTION RENDERING




```
function project({x, y, z}){
    return { x: x / z, y: y / z }
}
```

RAYCASTING




```
int wallHeight = (int)(projectionHalfHeight / distance);
drawTexturedWall(rayCount, wallHeight, rayX, rayY);
```

RAYCASTING W KODZIE

```
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;
```

Ustawianie zmiennych wymaganych do obsługi głównej pętli raycastowej. Przede wszystkim tworzy się kąt naszych rayów, który potem będzie inkrementowany, by stworzyć stożek widoczny wcześniej.

RAYCASTING W KODZIE

```
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);
```

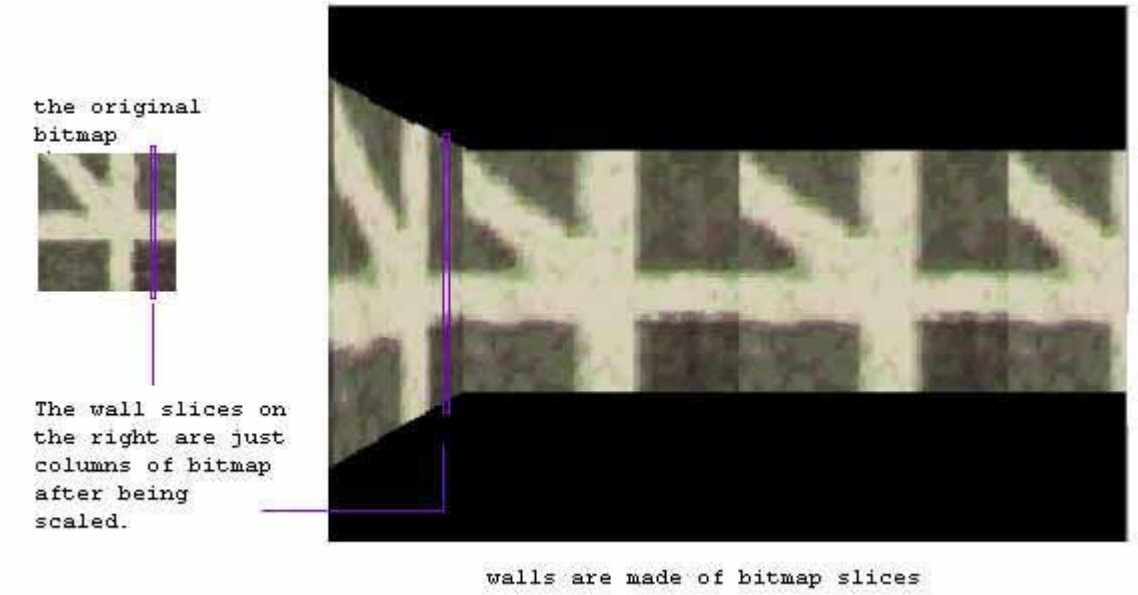
Ta część kodu odpowiada za sprawdzanie, kiedy ray dotknie najbliższej ściany

RAYCASTING W KODZIE

```
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
```

Ostatnia część odpowiada za rysowanie i teksturowanie ścian, skyboxa i podłogi, wg wcześniej wyliczonych zmiennych

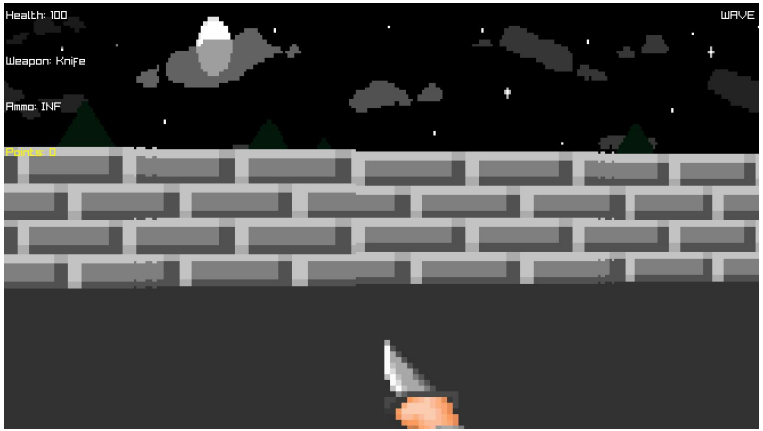
TEKSTUROWANIE

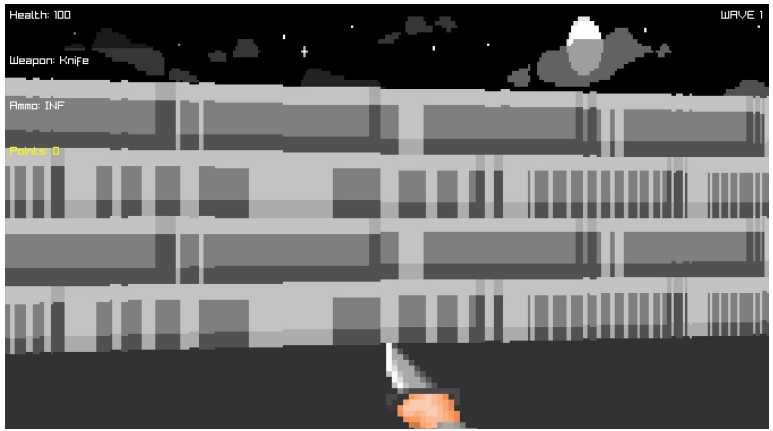


Z uwagi na to, że silnik raycastowy nie ma stałych bloków, teksturowanie działa nieco inaczej, jak w standardowych silnikach 3D

TEKSTUROWANIE - Ściany

Teksturowanie ścian polega na wielokrotnym powielaniu tej samej tekstury, tworząc iluzję jednolitości. Jednak przez wzgląd na to, że nasz silnik jest podatny na błędy precyzji, czasami ta iluzja się załamuje i tworzy się efekt wielokrotnego powielania ściany.





TEKSTRUROWANIE - Spritey

Spritey różnią się od ścian tym, że generują tylko jeden obraz. Nie działają też na bazie rayów. Obliczany jest dystans między przeciwnikiem i graczem, co ustala rozmiar spritea. Sprawdzane jest też to, czy między graczem i przeciwnikiem nie ma ściany (co korzysta już z rayów), by sprawdzić, czy sprite powinien być widoczny. W pierwszych wersjach gry pojawiał się też błąd, gdzie spritey pojawiały się jednocześnie z przodu i z tyłu, przez brak weryfikacji kierunku.

TEKSTUROWANIE W KODZIE - Ściany

```
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;
}
```

Ten kawałek kodu służy do “wycinania” kawałka tekstury, bazując na wcześniej pozyskanych danych, by potem całość ułożyła się w jednolitą ścianę. Z istniejącej tekstury wybieramy odcinek o danej długości pixeli i wszywamy go w miejsce, gdzie ray dotknął ściany.

TEKSTUROWANIE W KODZIE - Spritey

```
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
```

Ta część kodu odpowiada za sprawdzanie kiedy Sprite nie powinien się renderować.

TEKSTUROWANIE - Spritey

```
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);
```

Tutaj natomiast kalkulowana jest wysokość i szerokość, oraz renderowany jest sam sprite (w tym przypadku sprite potwora).

PRZECIWNICY - Programowanie Obiektowe

```cpp
#include <string>
#include "game_texture.h"

struct Monster {
    std::string id;
    std::string type;
    std::string skin;
    std::string audio;
    double x, y;
    int health;
    bool isDead;
    int width, height;
    GameTexture* texture;
    int damage;
    double lastAttack;
    double attackCooldown;
    double moveSpeed;

    Monster() : x(0), y(0), health(100), isDead(false), width(512), height(512),
        texture(nullptr), damage(1), lastAttack(0), attackCooldown(1.0) {
    }
};
```

```cpp
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;
```

Dzięki stworzeniu generycznej struktury przeciwnika później mogłem szybko i bezproblemowo dodawać kolejne ich rodzaje.

GAMEMASTER - Programowanie Obiektowe

```
public:
    Game();
    ~Game();

    bool initialize();
    int menu();
    void showLevelsList();
    void loadLevel(int levelIdx);
    void run();
    void cleanup();
    void resetGame();
```

```
case 1: //Start game
    game.showLevelsList();
    game.resetGame();
    game.run();
    break;
```

Dzięki funkcjom publicznym mogłem rozpoczynać, resetować i wybierać nowe poziomy do gry z poziomu głównego pliku.

PODZIAŁ OBOWIĄZKÓW W ZESPOLE:

Adrian Malawski - Menu, ustawienia, muzyka, pauza gry
Adam Podymniak - Główne działanie gry, Render mapy, przeciwników, broni, sklep, itemy, fale
Dominik Marek - Edytor map, Lista z poziomami

Dziękujemy za uwagę!