Grafika komputerowa

Projekt

Łazik WALL·E

L05

Agata Osuch

Mikołaj Pałka

1. Opis projektu

Realizacja projektu składała się z sześciu etapów:

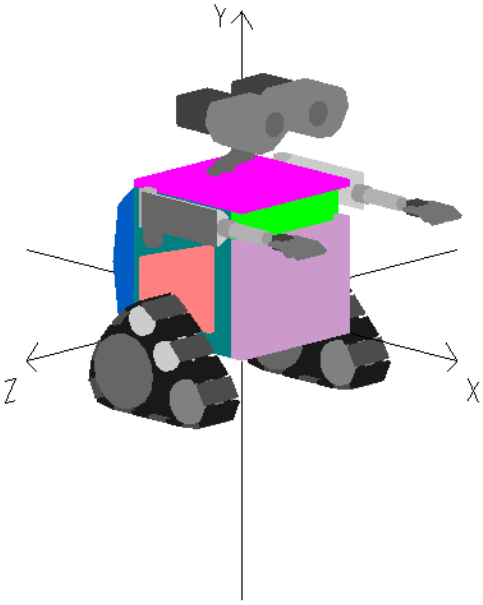
- Budowy obiektu sterowanego
- Budowy otoczenia
- Teksturowania
- Sterowania obiektem głównym
- Wykrywania kolizji
- Fabuły gry

Do stworzenia projektu wykorzystaliśmy język C++ oraz biblioteki OpenGL, Assimp i stb_image. Potrzebne modele utworzyliśmy w programie Blender.

2. Budowa obiektu sterowanego

Celem pierwszych zajęć poświęconych projektowi było stworzenie łazika wykorzystując prymitywy bazujące na trójkącie. Pomimo możliwości zaimportowania modelu stworzonego w Blenderze, obiekt został stworzony przy użyciu prymitywów biblioteki OpenGL, ponieważ obiekt, na którym się wzorowaliśmy, składa się z prostych figur.

Rysunek 1. Inspiracja zespołu projektowego



Rysunek 2. Utworzony łazik

```
2402    kola();
2403    gasienica();
2404
2405    glPushMatrix();
2406    glTranslatef(14, 0, 0);
2407    kola();
2408    gasienica();
2409    glPopMatrix();
2410
2411    tyl();
2412    przod();
2413    boki();
2414    dach();
2415    podloga();
2416    szyja();
2417
2418    ramiona();
2419
2420    glPushMatrix();
2421    glTranslatef(1.25, -8, 2.5);
2422    glScalef(-1, 1, 1);
2423    glTranslatef(-8.75, 8, -2.5);
2424    ramiona();
2425    glPopMatrix();
2426
2427    glPushMatrix();
2428    glTranslatef(0,0,-9);//3 przesuniecie z powrotem na miejsce
2429    glRotatef(obr, 1.0f, 0.0f, 0.0f);//2 obrot
2430    glTranslatef(0,0,9); //1 przesuniecie na 'poczatek' ukladu
2431    brzusio();
2432    glPopMatrix();
2433
2434    glPushMatrix();
2435    glTranslatef(7.75, 15.5, -10);
2436    glRotatef(oko1, 0.0f, 0.0f, 1.0f);
2437    glTranslatef(-7.75, -15.5, 10);
2438    oczko1(7.75, 15.5, -10);
2439    glPopMatrix();
2440    glPushMatrix();
2441    glTranslatef(2.25, 15.5, -10);
2442    glScalef(-1,1,1);
2443    glRotatef(oko1, 0.0f, 0.0f, 1.0f);
2444    glTranslatef(-2.25, -15.5, 10);
2445    oczko1(2.25, 15.5, -10);
```

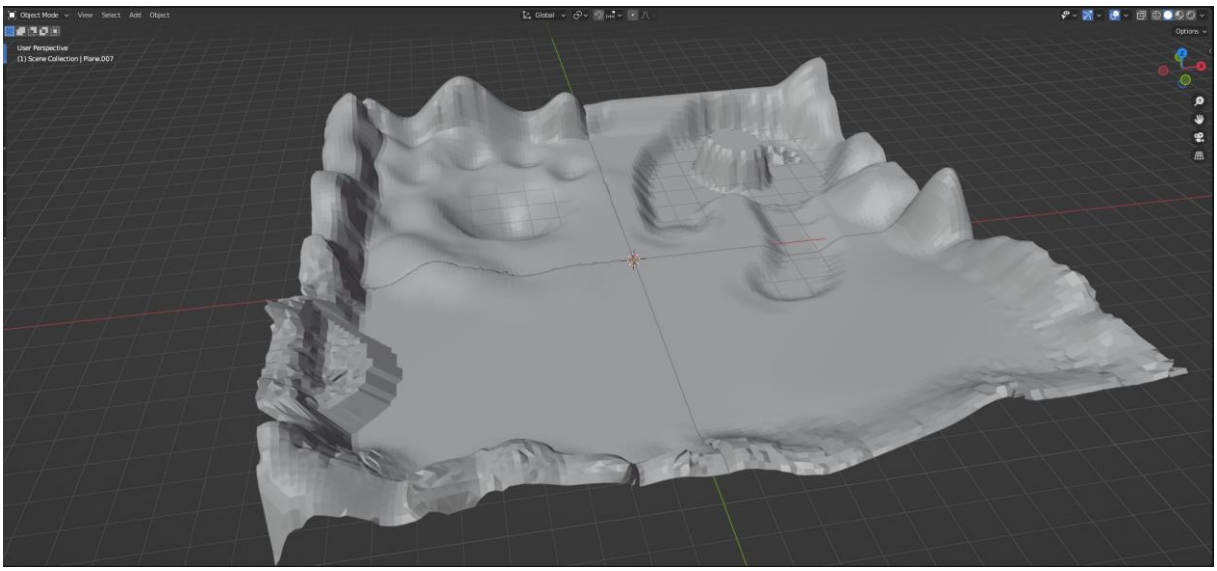
```
1522  void przod(void)
1523  {
1524      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
1525      glColor3f(0.0f, 1.0f, 0.0f);
1526      glBegin(GL_TRIANGLE_STRIP);
1527      glVertex3f(0, 8, -9);
1528      glVertex3f(0, 10, -9);
1529      glVertex3f(10, 8, -9);
1530      glVertex3f(10, 10, -9);
1531      glEnd();
1532
1533      glBegin(GL_TRIANGLE_STRIP);
1534      glVertex3f(1.5, 8, -10);
1535      glVertex3f(1.5, 10, -10);
1536      glVertex3f(8.5, 8, -10);
1537      glVertex3f(8.5, 10, -10);
1538      glEnd();
1539
1540      glBegin(GL_TRIANGLE_STRIP);
1541      glVertex3f(1.5, 8, -10);
1542      glVertex3f(1.5, 10, -10);
1543      glVertex3f(1.5, 8, -9);
1544      glVertex3f(1.5, 10, -9);
1545      glEnd();
1546
1547      glBegin(GL_TRIANGLE_STRIP);
1548      glVertex3f(8.5, 8, -10);
1549      glVertex3f(8.5, 10, -10);
1550      glVertex3f(8.5, 8, -9);
1551      glVertex3f(8.5, 10, -9);
1552      glEnd();
1553
1554      glBegin(GL_TRIANGLE_STRIP);
1555      glVertex3f(1.5, 8, -9);
1556      glVertex3f(1.5, 8, -10);
1557      glVertex3f(8.5, 8, -9);
1558      glVertex3f(8.5, 8, -10);
1559      glEnd();
1560  }
```

Rysunek 3. Funkcje tworzące łazik

W całości łazik składa się z 147 brył.

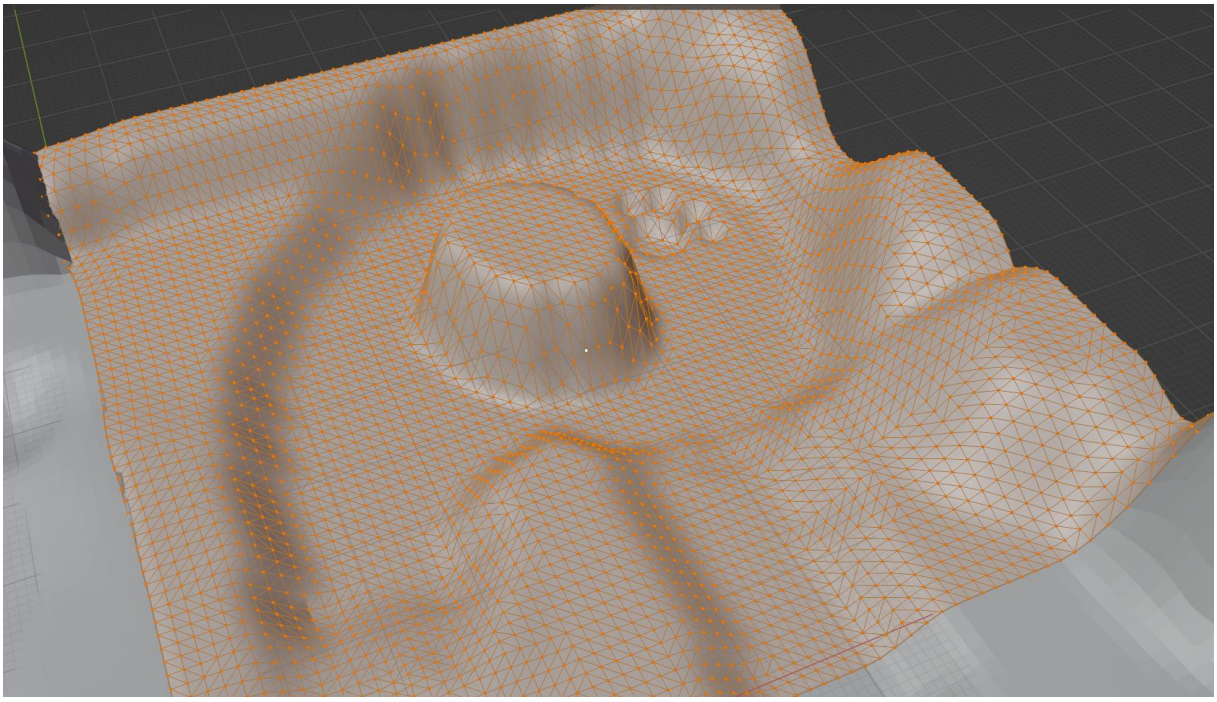
3. Budowa otoczenia

W celu stworzenia nieregularnego podłoża dla łazika zaimportowaliśmy do projektu model stworzony w Blenderze przy użyciu funkcji biblioteki Assimp.



Rysunek 4. Teren modelowany w Blenderze

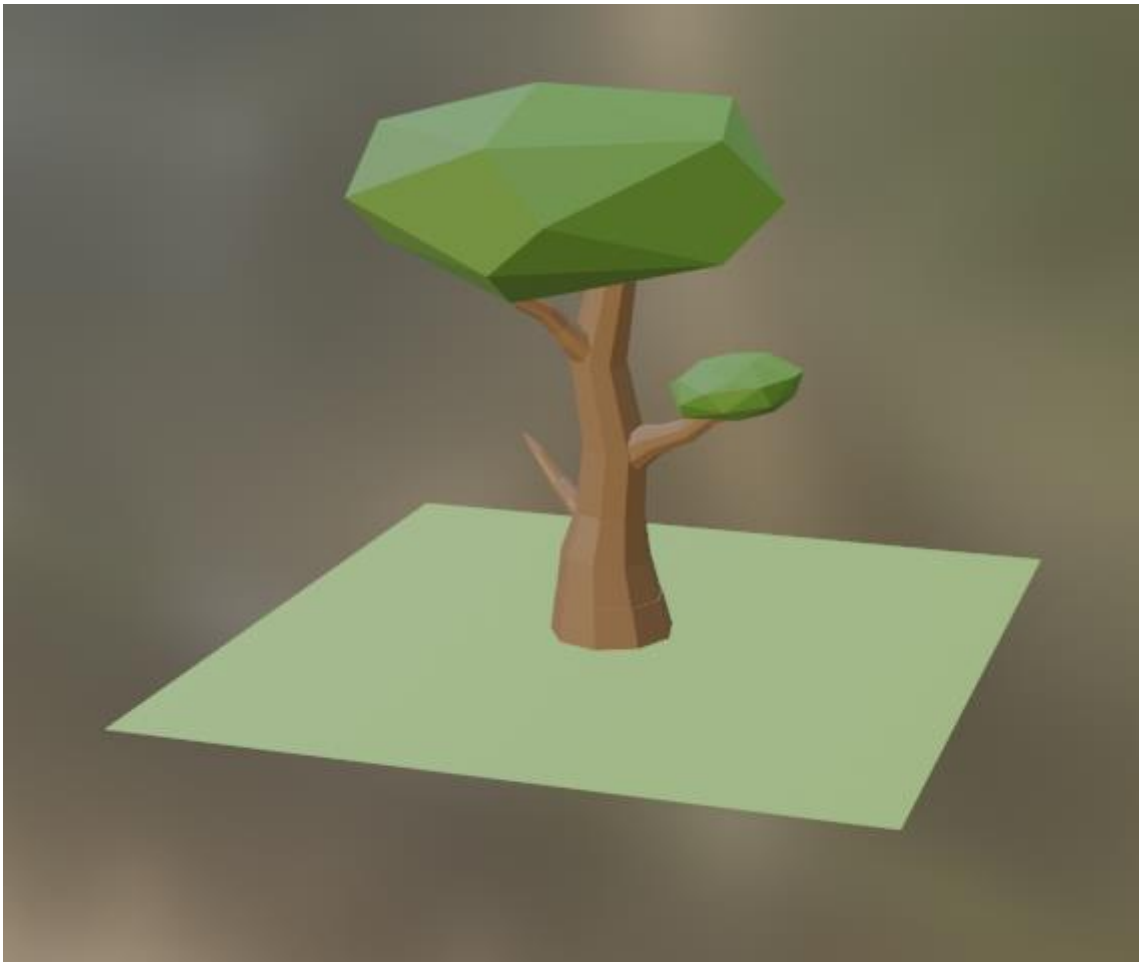
Podłoże składa się z czterech płaszczyzn podzielonych na wiele trójkątów. Wgłębienia oraz wzniesienia zostały utworzone poprzez zmianę wysokości poszczególnych wierzchołków. Bardzo przydatnym okazał się tryb proporcjonalnej edycji.



Rysunek 5. Jedna z czterech płaszczyzn tworzących podłoże

Rysunek 6. Zaimportowane podłoże

Stworzyliśmy i zaimportowaliśmy również model drzewa.



Rysunek 7. Drzewo modelowane w Blenderze

Podłoże zostało pokolorowane różnymi odcieniami zieleni, aby poprawić widoczność wgłębień w modelu.

Rysunek 8. Otoczenie dla łazika

Importowanie modeli zostało zrealizowane za pomocą biblioteki Assimp.

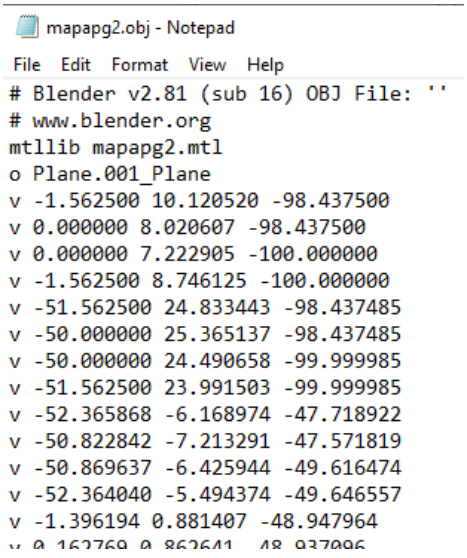
```
#include <assimp/Importer.hpp>
#include <assimp/postprocess.h>
#include <assimp/scene.h>
Assimp::Importer importer;
const aiScene* g_scene = nullptr;

bool Import3DFromFile(const std::string& pFile)
{
    g_scene = importer.ReadFile(pFile, aiProcessPreset_TargetRealtime_Quality);
    if (!g_scene)
        return false;
    return true;
}

void drawAiScene(const aiScene* scene, double skala, int teren)
{
    recursive_render(scene, scene->mRootNode, skala, teren);
}
```

```
if (!Import3DFromFile(modelpath))
{
    cleanup();
    return 0;
}
```

```
void cleanup()
{
    if (g_hWnd)
        KillGLWindow();
}
```

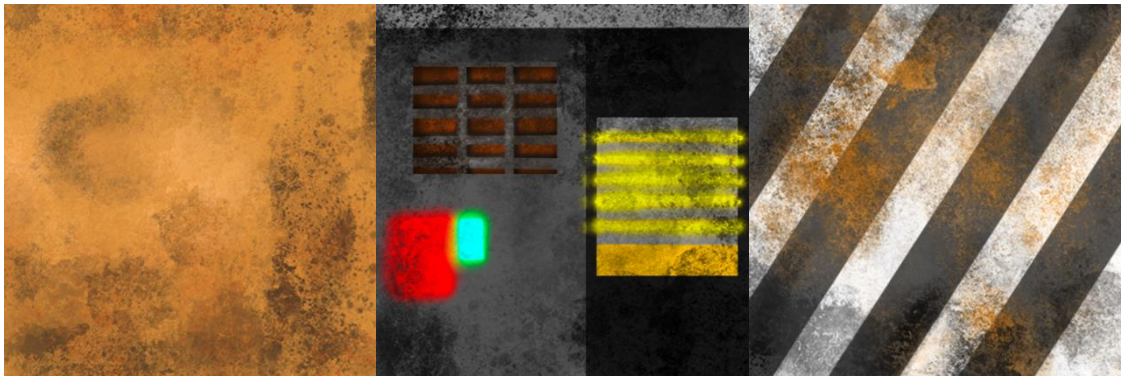
```
void recursive_render (const struct aiScene *sc, const struct aiNode* nd, float scale, int teren)
{
    unsigned int i;
    unsigned int n=0, t;
    aiMatrix4x4 m = nd->mTransformation;
    aiMatrix4x4 m2;
    aiMatrix4x4::Scaling(aiVector3D(scale, scale, scale), m2);
    m = m * m2;
    m.Transpose();
    glPushMatrix();
    glMultMatrixf((float*)&m);
    for (; n < nd->mNumMeshes; ++n)
    {
        const struct aiMesh* mesh = sc->mMeshes[nd->mMeshes[n]];
        for (t = 0; t < mesh->mNumFaces; ++t)
        {
            const struct aiFace* face = &mesh->mFaces[t];
            GLenum face_mode;
            switch(face->mNumIndices)
            {
                case 1: face_mode = GL_POINTS; break;
                case 2: face_mode = GL_LINES; break;
                case 3: face_mode = GL_TRIANGLES; break;
                default: face_mode = GL_POLYGON; break;
            }
            glBegin(face_mode);
            for(i = 0; i < face->mNumIndices; i++)
            {
                int vertexIndex = face->mIndices[i];
                if(mesh->HasTextureCoords(0))
                    glTexCoord2f(mesh->mTextureCoords[0][vertexIndex].x, 1 - mesh->mTextureCoords[0][vertexIndex].y);
                glNormal3fv(&mesh->mNormals[vertexIndex].x);
                if (mesh->mVertices[vertexIndex].y > 15.0f && teren)
                    glColor4f(0.42, 0.61, 0.345,1.0f);
                else if ((mesh->mVertices[vertexIndex].y < 15.0f && mesh->mVertices[vertexIndex].y > -2.0f)&& teren)
                    glColor4f(0.34, 0.49, 0.27,1.0f);
                else if ( mesh->mVertices[vertexIndex].y < -2.0f && teren)
                    glColor4f(0.27, 0.39, 0.22, 1.0f);
                glVertex3fv(&mesh->mVertices[vertexIndex].x);
            }
            glEnd();
        }
    }
    for (n = 0; n < nd->mNumChildren; ++n)
        recursive_render(sc, nd->mChildren[n], scale, teren);

    glPopMatrix();
}
```

Funkcja recursive_render rekurencyjnie przechodzi przez plik .obj i tworzy odpowiednie typy prymitywów na podstawie zawartości pliku. Jest tu także zrealizowane kolorowanie powierzchni terenu w zależności od wysokości wierzchołków.



4. Teksturowanie

W celu poprawienia wyglądu utworzonych obiektów nałożyliśmy na nie tekstury. Tekstury łazika zostały namalowane w programie Procreate.



Rysunek 9. Tekstury łazika – ściany, przedni panel, ramiona

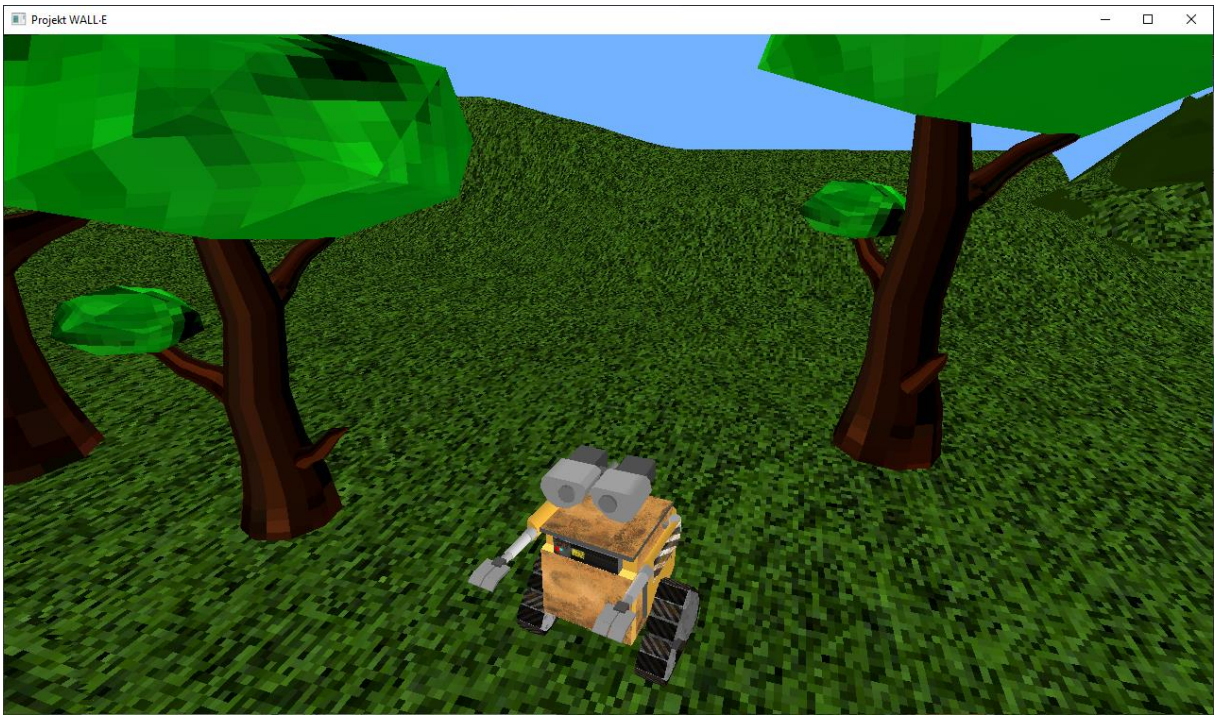
Tekstury otoczenia zostały wzięte z Internetu.

Rysunek 10. Tekstury otoczenia – drzewa, liście, woda, trawa

Do wczytywania tekstur użyto gotowej funkcji podanej w przykładowym projekcie „sześcian”.

```
#define BITMAP_ID 0x4D42
// Opis tekstury
BITMAPINFOHEADER    bitmapInfoHeader;   // nagłówek obrazu
unsigned char* bitmapData;              // dane tekstury
unsigned int        texture[10];        // obiekt tekstury

glGenTextures(10, &texture[0]);                 // tworzy obiekt tekstury

    // ładuje pierwszy obraz tekstury:
bitmapData = LoadBitmapFile("Bok.bmp", &bitmapInfoHeader);

glBindTexture(GL_TEXTURE_2D, texture[0]);       // aktywuje obiekt tekstury

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);

// tworzy obraz tekstury
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bitmapInfoHeader.biWidth,
    bitmapInfoHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

if (bitmapData)
    free(bitmapData);
```

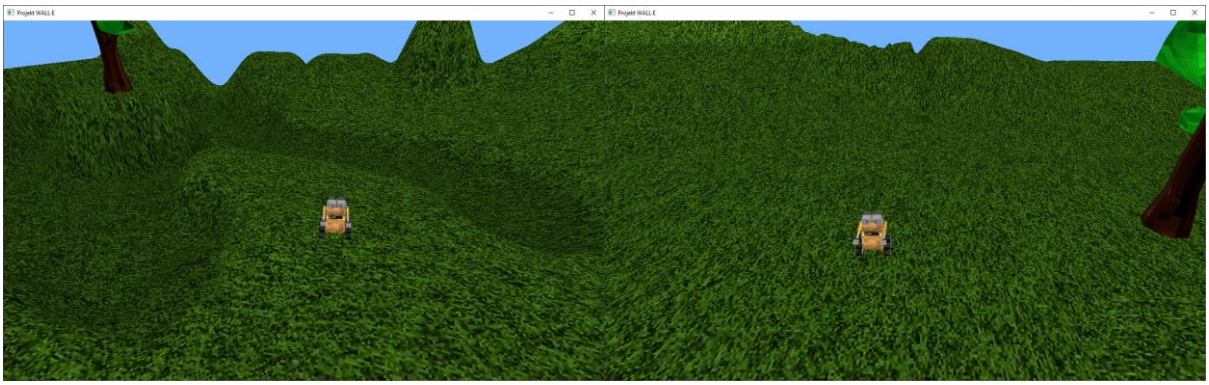
```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture[7]);
glColor3f(1,1,1);
glBegin(GL_TRIANGLE_STRIP);
glNormal3f(0.0f, 0.0f, -1.0f);
glTexCoord2d(0.0, 0.0); glVertex3f(1.5, 8, -10);
glTexCoord2d(0.0, 1.0); glVertex3f(1.5, 10, -10);
glTexCoord2d(2, 0.0); glVertex3f(8.5, 8, -10);
glTexCoord2d(2, 1.0); glVertex3f(8.5, 10, -10);
glEnd();
glDisable(GL_TEXTURE_2D);
```



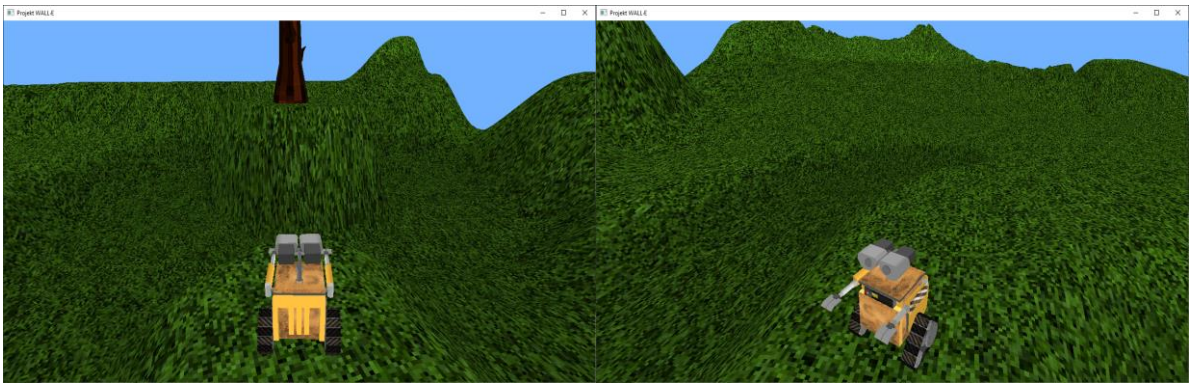
Rysunek 11. Obiekty z nałożonymi teksturami

W ramach tego etapu zrealizowane zostało także sterowanie kamerą. Za pomocą funkcji gluLookAt stworzyliśmy kamerę podążającą za łazikiem na odległości 50 jednostek. Dodaliśmy również kamerę zawieszoną 100 jednostek nad środkiem podłoża. Obiema kamerami można obracać w lewo i prawo za pomocą strzałek.

```
if (kamera)
{
    glRotatef(30, 1.0f, 0.0f, 0.0f);
    glRotatef(yRot, 0.0f, 1.0f, 0.0f);
    glTranslatef(0, -100, 0);
}
else
    gluLookAt(50 * sin(-yRot / 20.0f) + Xpos, 50 + Ypos, 50 * cos(-yRot / 20.0f) + Zpos, Xpos, Ypos + 30, Zpos, 0, 1, 0);
```



Rysunek 12. Kamera zawieszona nad środkiem obszaru gry



Rysunek 13. Kamera podążająca za łazikiem

5. Sterowanie obiektem głównym

Poruszanie się łazika zrealizowaliśmy za pomocą funkcji glTranslatef, glRotatef, glPushMatrix i glPopMatrix. Pozycja łazika przechowywana jest w zmiennych Xpos, Ypos, Zpos. Obrót łazika wokół własnej osi przy użyciu glRotatef wymaga, aby był on wyśrodkowany. Obiekt jest najpierw przesuwany o -5 jednostek w osi X i 5 jednostek w osi Z. Następnie jest obracany o kąt ustalany za pomocą klawiszy klawiatury numerycznej. Na koniec jest przesuwany na pozycję określoną przez zmienne.

```
glTranslatef(Xpos, Ypos, Zpos);
glRotatef(kat, 0, 1, 0);
glRotatef(pochylenie, 1, 0, 0);
glTranslatef(-5, 0, 5);
```

```
if (keys[VK_NUMPAD8])
{
    if(pochylenie <1)
    pochylenie += 0.125f;
    if (predkosc < 1.3)
        predkosc += 0.1;
    obrotSrubek += predkosc * 10;
    if (!kolizja(-predkosc * sin(kat * GL_PI / 180.0f), -predkosc * cos(kat * GL_PI / 180.0f)))
    {
        Xpos += -predkosc * sin(kat * GL_PI / 180.0f);
        Zpos += -predkosc * cos(kat * GL_PI / 180.0f);
    }
    kolorGasienicy += 0.2f;
    if (kolorGasienicy > 0.2f)
        kolorGasienicy = 0.0f;
}
if (keys[VK_NUMPAD5])
{
    obrotSrubek -= 0.25 * 10;
    if (!kolizja(0.25 * sin(kat * GL_PI / 180.0f), 0.25 * cos(kat * GL_PI / 180.0f)))
    {
        Xpos -= -0.25 * sin(kat * GL_PI / 180.0f);
        Zpos -= -0.25 * cos(kat * GL_PI / 180.0f);
    }
}
if (keys[VK_NUMPAD4])
{
    obrotSrubek += 5;
    kat += 1;
    if (!keys[VK_NUMPAD8] && !keys[VK_NUMPAD5])
    {
        kolorGasienicy += 0.2f;
        if (kolorGasienicy > 0.2f)
            kolorGasienicy = 0.0f;
    }
}
if (keys[VK_NUMPAD6])
{
    obrotSrubek += 5;
    kat -= 1;
    if(!keys[VK_NUMPAD8] && !keys[VK_NUMPAD5])
    {
        kolorGasienicy += 0.2f;
        if (kolorGasienicy > 0.2f)
            kolorGasienicy = 0.0f;
    }
}
```

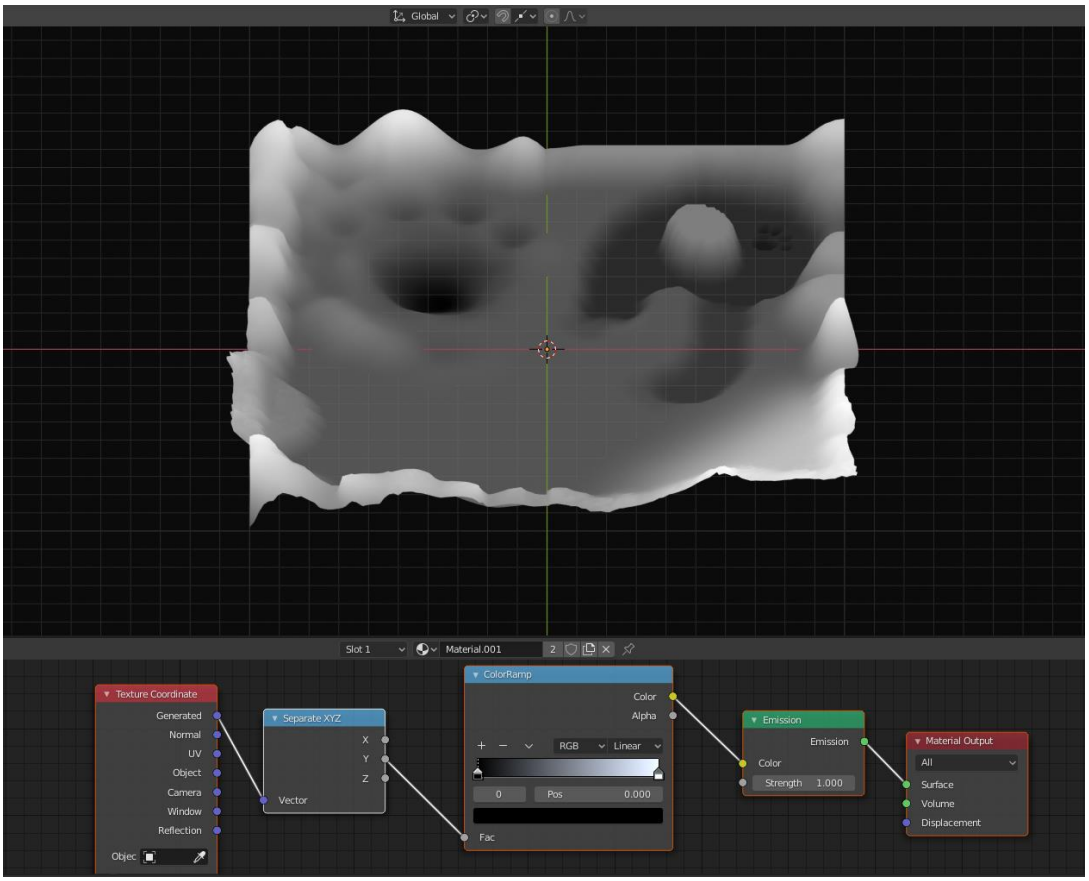
Poprzez przytrzymanie klawisza numpad8 zwiększa się prędkość łazika. Następnie dzięki funkcjom sinus i cosinus obliczana jest nowa pozycja X oraz Z obiektu. W celu wizualnego polepszenia ruchu łazika zmienia się także prędkość śrubek na kołach, kolor gąsienic i przechylenie łazika. Klawisze numpad4 i numpad6 odpowiedzialne są za obrót łazika wokół jego osi. Numpad5 sprawia, że łazik powoli jedzie do tyłu.

```
case WM_TIMER:
{
    switch (wParam)
case 1:
    {
        if (!keys[VK_NUMPAD8] && predkosc > 0.0f)
        {
            predkosc -= 0.1;
            obrotSrubek += predkosc * 10;
            if (!kolizja(-predkosc * sin(kat * GL_PI / 180.0f), -predkosc * cos(kat * GL_PI / 180.0f)))
            {
                Xpos += -predkosc * sin(kat * GL_PI / 180.0f);
                Zpos += -predkosc * cos(kat * GL_PI / 180.0f);
            }

            kolorGasienicy += 0.2f;
            if (kolorGasienicy > 0.2f)
                kolorGasienicy = 0.0f;
            pochylenie -= 0.125f;
        }
        else
            if (pochylenie < 0)
                pochylenie += 0.125f;
```

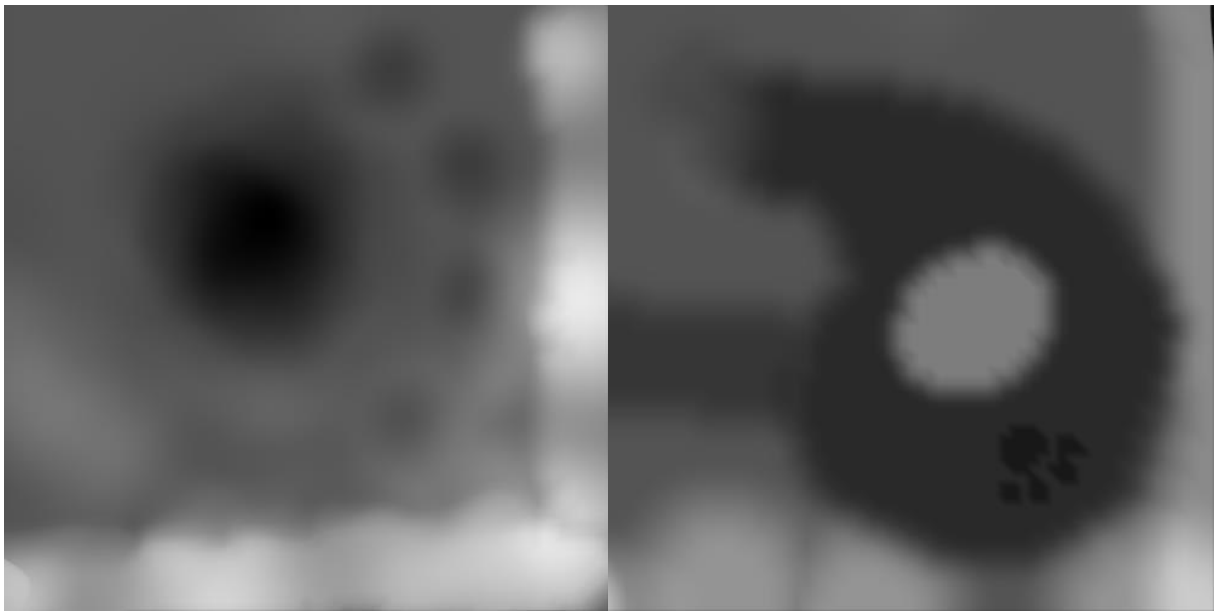
Za pomocą funkcji SetTimer ustawiliśmy wywoływanie się fragmentu kodu po upływie kilkunastu milisekund. Jest on odpowiedzialny za zmniejszenie prędkości łazika kiedy gracz puści klawisz poruszania się do przodu. Dzięki temu łazik nie zatrzymuje się w chwili puszczenia klawisza, lecz dopiero po krótkiej chwili ponieważ zachowuje swój pęd.

Rysunek 14. Ruch łazika po podłożu

Zmianę pozycji łazika w osi Y wykonaliśmy za pomocą map wysokości stworzonego wcześniej podłoża. Na model terenu nałożyliśmy materiał, który dzięki węzłom koloruje model na biało i czarno w zależności od wysokości powierzchni. Następnie za pomocą ortogonalnej kamery położonej nad podłożem wykonaliśmy cztery zdjęcia o rozmiarze 500x500 pikseli, które w projekcie są wczytywane przy użyciu biblioteki stb_image.



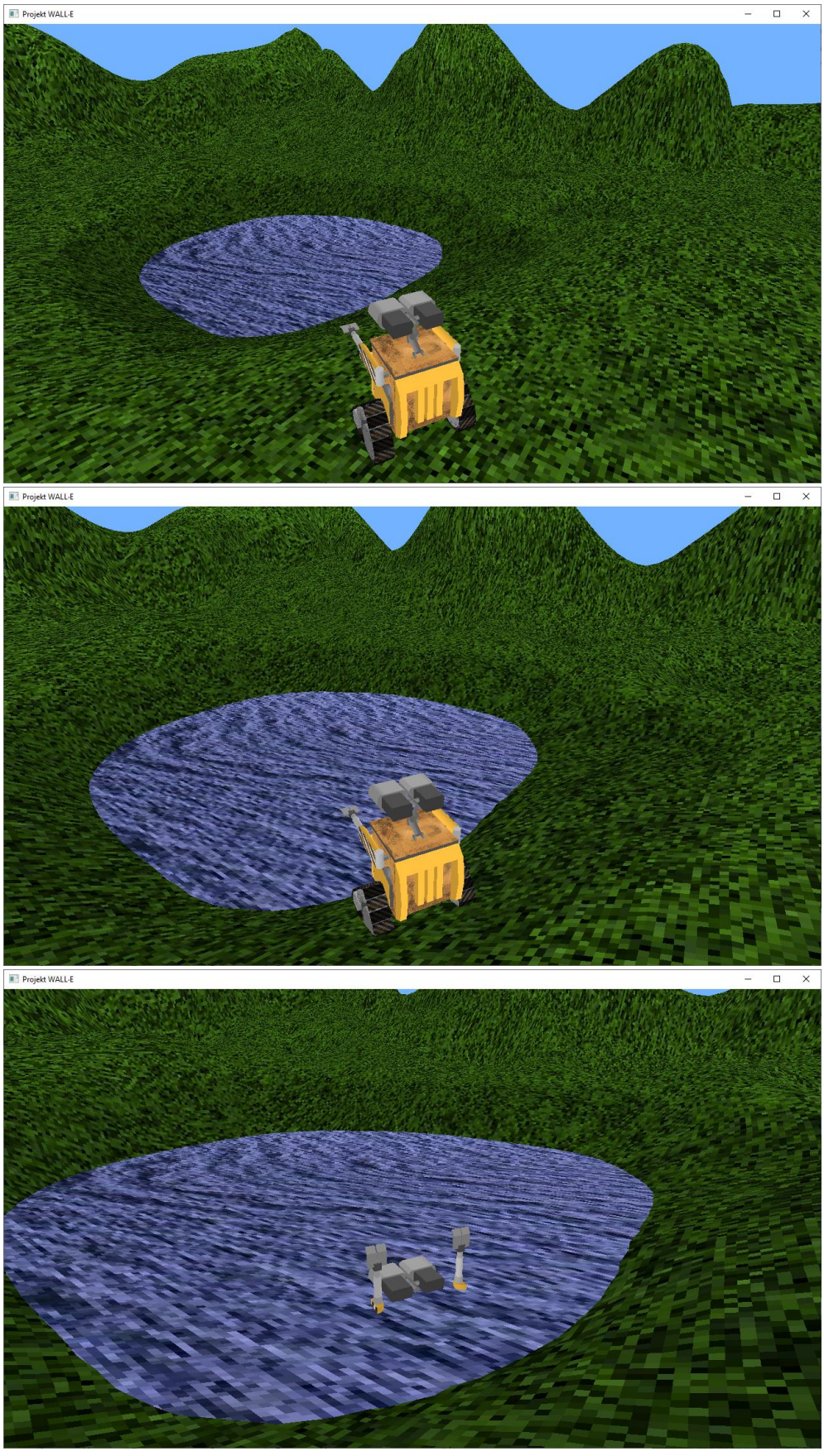
Rysunek 15. Mapa wysokości w Blenderze



Rysunek 16. Dwie z czterech części mapy wysokości podłoża

```
#define STB_IMAGE_IMPLEMENTATION
#include "contrib/stb/stb_image.h"

float wys1[500][500];
float wys2[500][500];
float wys3[500][500];
float wys4[500][500];

int w, h;
stbi_uc* mapa = stbi_load("mapapg.png", &w, &h, NULL, 1);
stbi_uc* mapalg = stbi_load("mapalg.png", &w, &h, NULL, 1);
stbi_uc* mapald = stbi_load("mapald.png", &w, &h, NULL, 1);
stbi_uc* mapapd = stbi_load("mapapd.png", &w, &h, NULL, 1);

 for (int i = 0; i < 500; i++)
 {
     for (int j = 0; j < 500; j++)
         wys1[i][j] = mapalg[w * i + j] -80.0f ;
 }
 for (int i = 0; i < 500; i++)
 {
     for (int j = 0; j < 500; j++)
         wys2[i][j] = mapa[w * i + j] - 80.0f;
 }
 for (int i = 0; i < 500; i++)
 {
     for (int j = 0; j < 500; j++)
         wys3[i][j] = mapald[w * i + j] - 80.0f;
 }
 for (int i = 0; i < 500; i++)
 {
     for (int j = 0; j < 500; j++)
         wys4[i][j] = mapapd[w * i + j] - 80.0f;
 }
if (Xpos < 0 && Zpos < 0)
    Ypos = wys1[(int)-Xpos][(int)-Zpos];
else
    if (Xpos > 0 && Zpos < 0)
        Ypos = wys2[(int)Xpos][(int)-Zpos];
else
    if (Xpos < 0 && Zpos > 0)
        Ypos = wys3[(int)-Xpos][(int)Zpos];
else
    if (Xpos > 0 && Zpos > 0)
        Ypos = wys4[(int)Xpos][(int)Zpos];
```

Cztery tablice dwuwymiarowe są wypełniane wartościami opisującymi kolor poszczególnych pikseli zdjęć map wysokości. Położenie łazika w pionie określane jest w zależności od jego pozycji w osiach X i Z.



Rysunek 17. Ruch łazika po nierównym podłożu

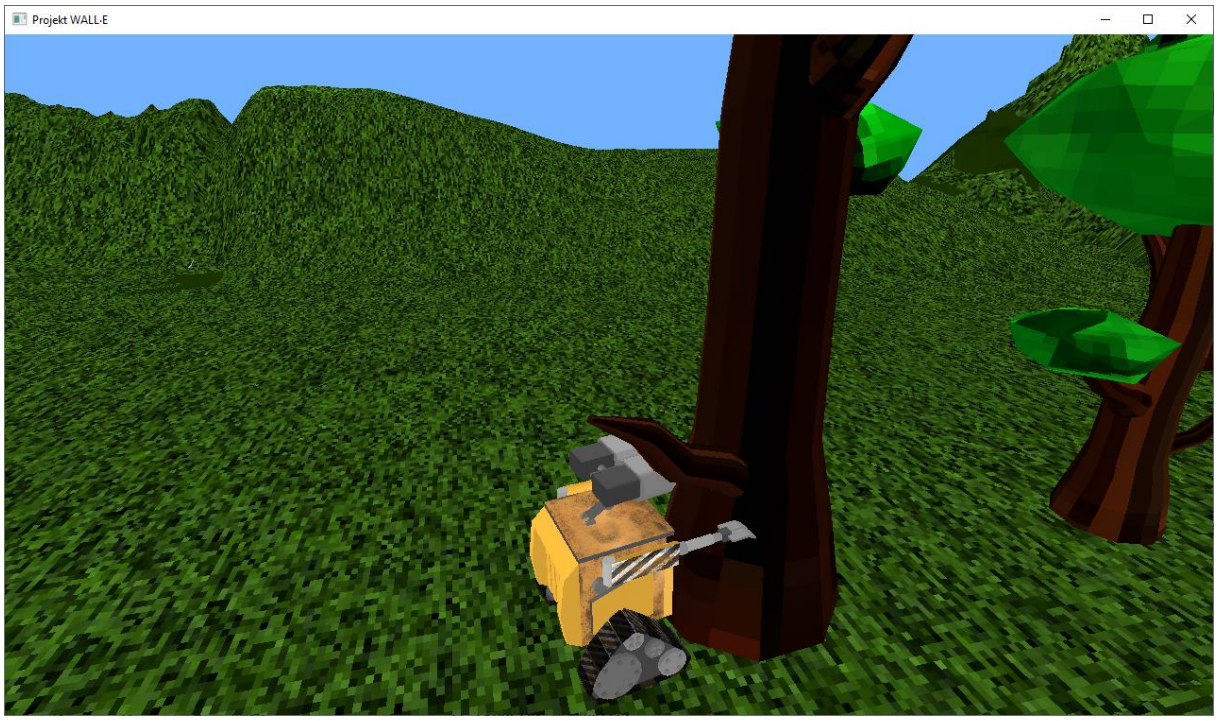
6. Wykrywanie kolizji

W celu usunięcia przenikania łazika przez obiekty otoczenia zaimplementowaliśmy prostą detekcję kolizji.

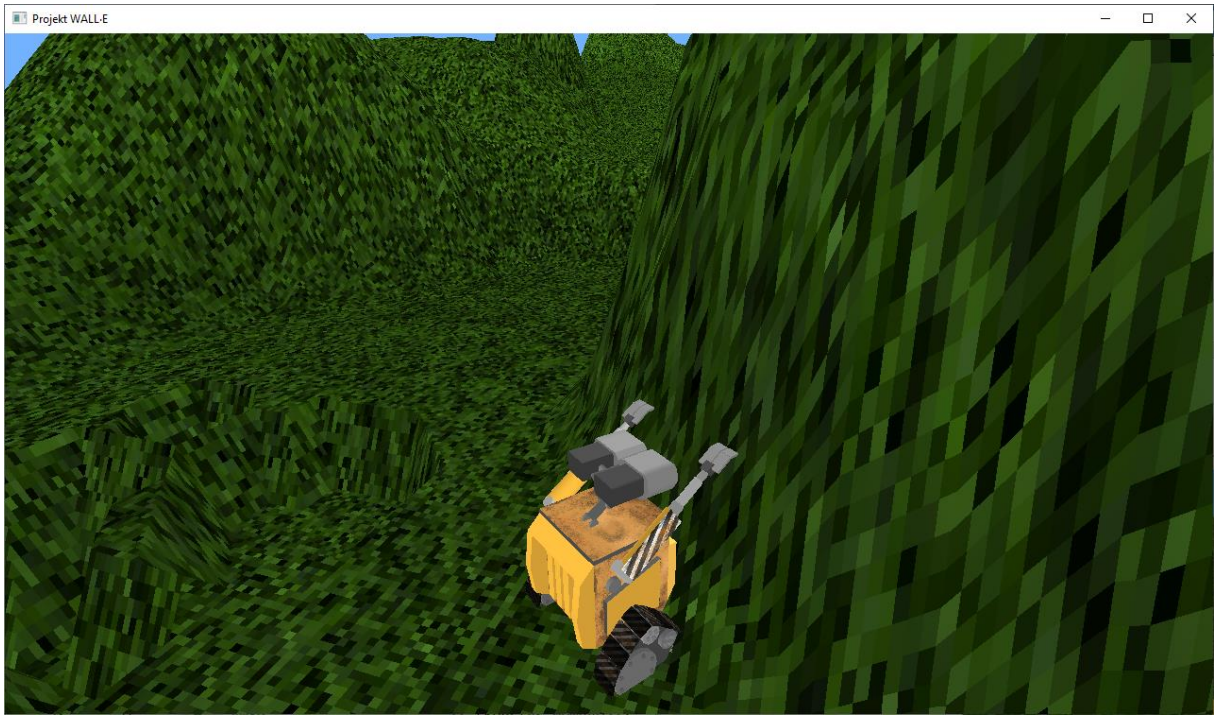
```
float odleglosc(float aX, float bX, float aZ, float bZ)
{
    return sqrt(pow((aX-bX),2)+pow(aZ-bZ,2));
}


bool kolizja(float zmianaX, float zmianaZ)
{
    if (Xpos > 0 && Zpos < 0)
    {
        if (odleglosc(Xpos + zmianaX, 260, Zpos + zmianaZ, -300) < (5 * sqrt(2) + 75))
            return true;
    }
    if (Xpos < 0 && Zpos > 0)
    {
        if (odleglosc(Xpos + zmianaX, -150, Zpos + zmianaZ, 150) < (5 * sqrt(2) + 12))
            return true;
    }
    if (odleglosc(Xpos + zmianaX, 0, Zpos + zmianaZ, -100) < (5*sqrt(2) + 10))
        return true;

    return false;
}
```

Promień obszaru kolizyjnego łazika został przybliżony do okręgu opisanego na kwadracie o rozmiarach łazika. Podczas ruchu sprawdzane jest, czy po przesunięciu łazika na nową pozycję odległość środków obiektów byłaby mniejsza od sumy ich promieni. Wykrywanie kolizji podzielono tak, aby odległości obiektów były sprawdzane wtedy, kiedy są w tej samej ćwiartce układu współrzędnych.

```
if (keys[VK_NUMPAD8])
{
    if(pochylenie <1)
    pochylenie += 0.125f;
    if (predkosc < 1.3)
        predkosc += 0.1;
    obrotSrubek += predkosc * 10;
    if (!kolizja(-predkosc * sin(kat * GL_PI / 180.0f), -predkosc * cos(kat * GL_PI / 180.0f)))
    {
        Xpos += -predkosc * sin(kat * GL_PI / 180.0f);
        Zpos += -predkosc * cos(kat * GL_PI / 180.0f);
    }
    kolorGasienicy += 0.2f;
    if (kolorGasienicy > 0.2f)
        kolorGasienicy = 0.0f;
}
```



Rysunek 18. Łazik zatrzymuje się w miejscu po zderzeniu z drzewem



Rysunek 19. Obszar kolizyjny wokół pagórka zapobiega wspinanie się łazika po zbyt stromych powierzchniach

7. Fabuła gry

Ten etap projektu poświęciliśmy na rozbudowę interakcji z łazikiem oraz otoczeniem. Dodaliśmy kilka funkcjonalności inspirowanych łazikiem, którego staraliśmy się odwzorować.

Rysunek 20. Obrót oczu łazika

Rysunek 21. Poruszanie ramionami łazika

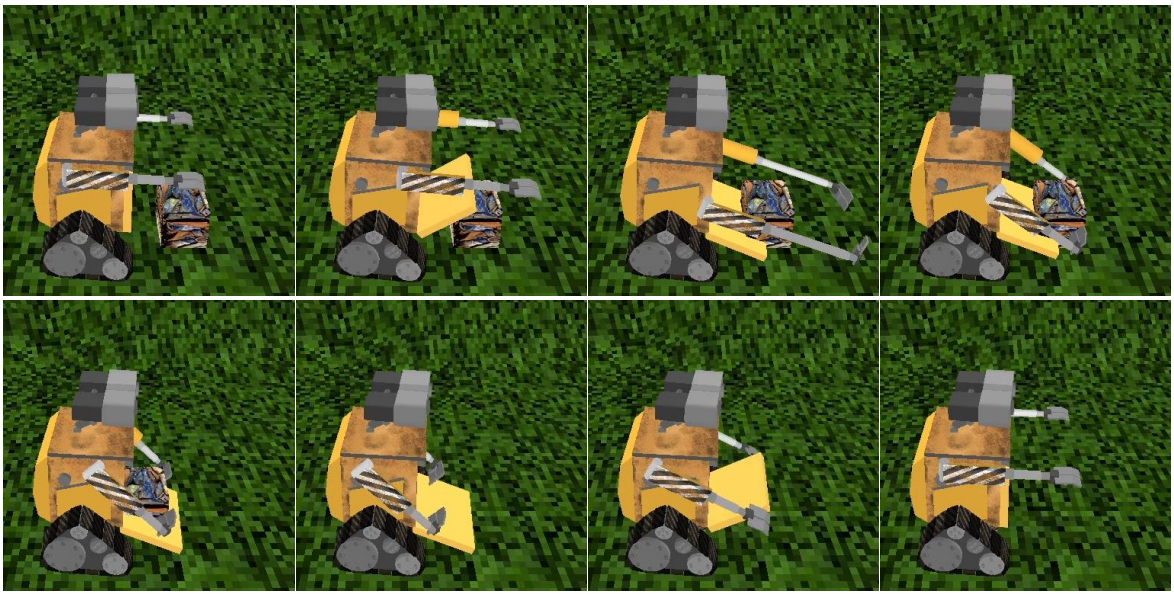
Rysunek 22. Otwieranie przedniej klapy łazika

Rysunek 23. Rozglądanie się łazika zrealizowane za pomocą funkcji SetTimer

```
if (keys[0x4F])//O opusc klape
{
    if(obr>-110)
    obr -= 3;
}
if (keys[0x50])//P podnies klape
{
    if(obr<0)
    obr += 3;
}
if (keys[0x4B])//K opusc oczka
{
    if (oko1 > 0)
        oko1 -= 1;
}
if (keys[0x4C])//L podnies oczka
{
    if (oko1 < 13)
        oko1 += 1;
}
if (keys[0x52])//R wydluz ramie
{
    if (przedluzenie < 7)
        przedluzenie += 0.5;
    if (dlonObr < 90)
        dlonObr += 10;
    if (dlonObr2 < 75)
        dlonObr2 += 5;
}
if (keys[0x54])//T skroc ramie
{
    if (przedluzenie > 0)
        przedluzenie -= 0.5;
    if (dlonObr > 0)
        dlonObr -= 10;
    if (dlonObr2 > 0)
        dlonObr2 -= 5;
}
```

```
if (keys[0x46])//F ramie w dol
{
    if (ramieZ < 5.75)
    {
        ramieZ += 0.25;
    }
    else
    {
        if (ramieY < 5.75)
        {
            ramieY += 0.25;
        }
    }

}
if (keys[0x47])//G ramie w gore
{
    if (ramieY > 0)
    {
        ramieY -= 0.25;
    }
    else
    {
        if (ramieZ > 0)
        {
            ramieZ -= 0.25;
        }
    }

}
if (keys[0x43])//C podnies ramie
{
    if(katRamion < 90)
        katRamion += 2.5;
}
if (keys[0x56])//V opusc ramie
{
    if (katRamion > -25)
        katRamion -= 2.5;
}
```

Połączenie nowo dodanych funkcjonalności poruszania indywidualnymi częściami łazika pozwoliło na zbudowanie skomplikowanej animacji podnoszenia obiektów z podłoża i chowania ich wewnątrz łazika.



Rysunek 24. Animacja wciągania kostki złomu do łazika

Animacja wykonuje się po wciśnięciu klawisza X dzięki funkcji SetTimer. Kostka jest przesuwana na środek łazika po wykonaniu wszystkich kroków animacji.

```
if (keys[0x58])//X podnies
{
    podnoszenie = 1;

}
```

```
case 3:
{
    if (podnoszenie)
    {
        if (kroki < 25)
        {
            if (obr > -110)
                obr -= 3;
            if (ramieZ < 5.75)
            {
                ramieZ += 0.25;
                if (pochylenie > -5)
                    pochylenie -= 0.2;
            }
            kroki++;
        }
        if (kroki >= 25 && kroki < 50)
        {
            if (obr > -110)
                obr -= 3;
            if (ramieY < 5.75)
                ramieY += 0.25;
            if (katRamion > -25)
                katRamion -= 1;
            if (przedluzenie < 7)
                przedluzenie += 0.5;
            if (dlonObr < 90)
                dlonObr += 10;
            if (dlonObr2 < 75)
                dlonObr2 += 3.5;
            kroki++;
        }
        if (kroki >= 50 && kroki < 100)
        {

            if (pochylenie < 0)
                pochylenie += 0.2;
            if (przedluzenie > 0)
                przedluzenie -= 0.5;
            if (katRamion > -45)
                katRamion -= 1;
            if (ramieY > 0)
                ramieY -= 0.25;
        }
        else
        {

            if (ramieZ > 0)
                ramieZ -= 0.25;
            if(ramieZ < 4)
                wpychanie = 1;
        }
        kroki++;
    }
    if (kroki >= 100 && kroki < 110)
    {
        wpychanie = 0;
        /*if (dlonObr > 0)
            dlonObr -= 10;*/
        if (dlonObr2 > 0)
            dlonObr2 -= 10;
        if (obr < 0)
            obr += 3;
        kroki++;
    }
    if (kroki >= 110 && kroki < 150)
    {
        if (dlonObr > 0)
            dlonObr -= 10;
        if (obr < 0)
            obr += 3;
        if (katRamion < 0)
            katRamion += 2.5;
        kroki++;
    }
    if (kroki == 150)
    {
        kroki = 0;
        podnoszenie = 0;
        nies = 1;
    }
}
```

```
glPushMatrix();
if (nies)
{
    catX = Xpos;
    catY = Ypos+5;
    catZ = Zpos;
}
else
{
    if (wpychanie)
    {
        if (abs(Xpos - catX) > 0)
        {
            if (Xpos > catX)
                catX++;
            else
                catX--;

        }
        if (abs(Zpos - catZ) > 0)
        {
            if (Zpos > catZ)
                catZ++;
            else
                catZ--;
        }
        if (catY < 5)
            catY++;
    }
```

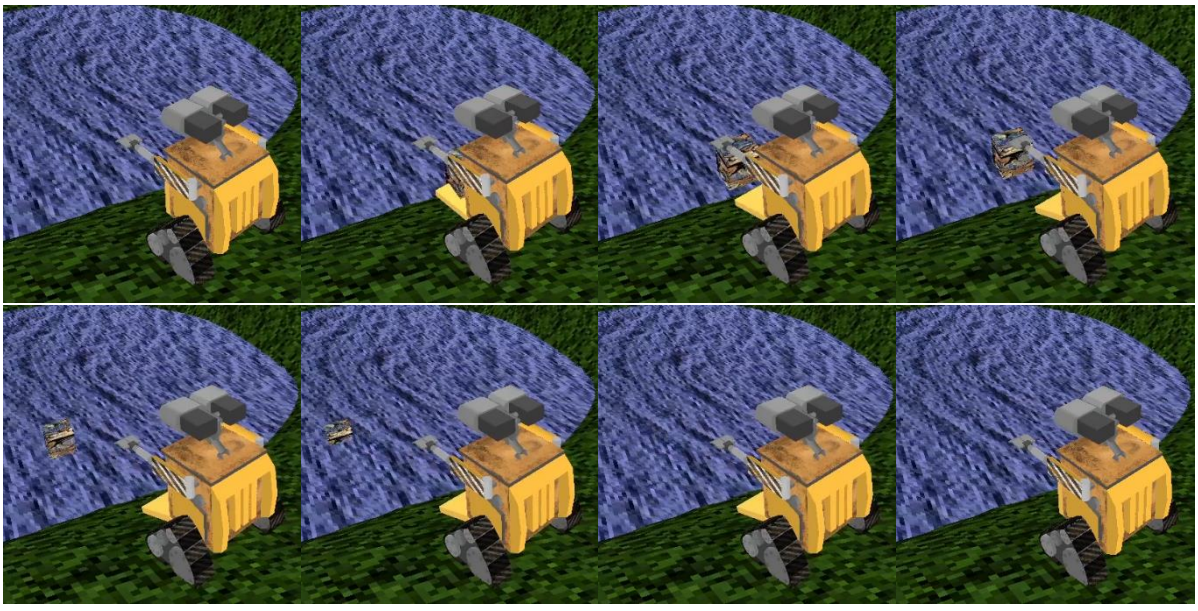
Rysunek 25. Łazik przewożący kostkę

Po podniesieniu obiektu łazik może go wystrzelić. W celu dodania realizmu kostka opada podczas lotu. W realizacji tej funkcjonalności przydatne były wcześniej stworzone mapy wysokości pozwalające na kolizję kostki z podłożem.

```
if (keys[0x42])//B opusc
{
    if (nies)
    {
        opuszczanie = 1;
        opad = 0.45;
    }
}
```

```
if (opuszczanie)
{
    if (kroki < 25)
    {
        if (obr > -110)
            obr -= 3;
        if (pochylenie > -5)
            pochylenie -= 0.2;
        kroki++;
    }
    if (kroki >= 25 && kroki < 70)
    {
        nies = 0;
        if (obr > -110)
            obr -= 3;
        catX += -(2+predkosc) * sin(kat * Gl_PI / 180.0f);
        catZ += -(2+predkosc) * cos(kat * Gl_PI / 180.0f);
        kroki++;
        opad += 0.015;
    }
    if (kroki >= 70 && kroki < 125)
    {
        if (obr < 0)
            obr += 3;
        kroki++;
    }
    if (kroki == 125)
    {
        opuszczanie = 0;
        kroki = 0;
    }
}
if (catY > minCatY)
    catY-=opad;
```

```
if (catX < 0 && catZ < 0)
    minCatY = wys1[(int)-catX][(int)-catZ];
else
    if (catX > 0 && catZ < 0)
        minCatY = wys2[(int)catX][(int)-catZ];
    else
        if (Xpos < 0 && Zpos > 0)
            minCatY = wys3[(int)-catX][(int)catZ];
        else
            if (Xpos > 0 && Zpos > 0)
                minCatY = wys4[(int)catX][(int)catZ];
}

glTranslatef(catX, catY, catZ);
if(nies)
    glRotatef(kat, 0, 1, 0);
glScalef(0.018, 0.018, 0.018);
```



Rysunek 26. Łazik wystrzeliwujący kostkę