  
Izobraževalni program: tehnik računalništva

Seminarska naloga: četrti predmet poklicne mature 2012

**Spopad titanov**

Mentorica: **prof. Nataša Makarovič** Avtor: **Luka Prijatelj, R 4. A**

**Ljubljana, april 13**

*(prazna stran)*

# Zahvala

Pri izdelavi 3D igrice Spopad Titanov, sem naletel na številne težave in ovire. Premagovanje ovir je bilo včasih zelo naporno, vendar mi je kljub vsemu na koncu le uspelo dokončati maturitetni izdelek tako, kot sem si ga zamisli. Takih ovir pa ne bi nikoli zmogel premagati, če mi ne bi pomagala profesorica ga. Nataša Makarovič. Ves čas izdelovanja mi je nudila strokovno pomoč iz programskih jezikov. Bila je tudi ključna pri izdelku, saj me je skozi celoten čas izdelave igrice spodbujala ter občasno tudi kritizirala.

Posebna zahvala pa gre tudi profesorici ga. Eriki Pleši, ki mi je nudila strokovno pomoč pri reševanju matematičnih zapletov. Brez njene pomoči in razlage bi bila izdelava tega izdelka močno otežena ali pa morda celo nemogoča.

Zahvaljujem pa se tudi vsem, ki so mi kdajkoli ponudili nasvet pri izdelavi igrice.

# Povzetek

V seminarski nalogi Vam bom na kratko predstavil izdelavo mojega maturitetnega izdelka. Za temo sem si izbral računalniško igrico, ki nosi naslov Spopad titanov. Na naslednjih straneh seminarske naloge Vam bom predstavil potek izdelave računalniške igrice. Predstavil vam bom tudi posamezne dele programske kode, ki so ključni deli za pravilno delovanje računalniške igrice. V seminarski nalogi so prisotne tudi slike, ki prikazujejo izdelovanje računalniške igrice.

Na koncu seminarske naloge sem poskušal opisati probleme in ovire, ki so se pojavile pri izdelovanju računalniške igrice. Navedel sem tri najpogostejše ovire, na katere sem velikokrat naletel. Poskušal sem jih čim bolje opisati in predstaviti njihovo rešitev. Probleme bi lahko rešil tudi na drugačne načine, a sem se odločil za tiste, ki so najbolj učinkovite.

**Ključne besede:**

- Spopad titanov

- računalniška igrica

- dvoboj robotov

- C# igrica

- modeliranje objektov

- boksarska arena

# Summary

In the documentation of my game project I am going to describe the process of creating my 3D game. This computer game was named by the movie Clash of the Titans.

In this documentataion I tried to introduce You to my computer game creation. Throught my documentation I will be explaining programing code. You will be able to find many pictures that were taken during game creation. I also tried to write down and explain the problems I came across in my game creation. Even though there were many problems, I only tried to explain the ones I came across very often. There are also many diffrent solutions to my problems, but I decided to choose the solutions that were most effective for my game.

**Key words:**

- Clash of the Titans

- XNA Game Studio

- Fighting robots

- C# game

- Modeling 3D objects

- Boxing arena

Vsebina

[Zahvala 2](#_Toc353991543)

[Povzetek 3](#_Toc353991544)

[Summary 4](#_Toc353991545)

[Uvod 0](#_Toc353991546)

[Uporabljena programska oprema 1](#_Toc353991547)

[ Integrirano razvojno okolje 1](#_Toc353991548)

[ Program za oblikovanje 3D modelov 2](#_Toc353991549)

[ Program za oblikovanje slik 3](#_Toc353991550)

[Oblikovanje 3D-igrice 4](#_Toc353991551)

[Osnovna sestava XNA GS 4. 0 grafične knjižnice 5](#_Toc353991552)

[Nasveti za igranje 6](#_Toc353991553)

[Opis igralnega lika 7](#_Toc353991554)

[Projekt 1: Izdelava menija 8](#_Toc353991555)

[I. Grafično oblikovanje menija 8](#_Toc353991556)

[II. Programska koda (opis) 9](#_Toc353991557)

[Projekt 2: Izdelava glavne igrice 19](#_Toc353991558)

[I. Grafično oblikovanje igrice 19](#_Toc353991559)

[II. Programska koda (opis) 20](#_Toc353991560)

[Problemi med izdelovanjem igrice 32](#_Toc353991561)

[ Matematični problemi 32](#_Toc353991562)

[ Grafični problemi 34](#_Toc353991563)

[ Problemi pri animaciji objekta 35](#_Toc353991564)

[Kazalo slik in tabel 36](#_Toc353991565)

[Viri in literatura 37](#_Toc353991566)

[Priloge 38](#_Toc353991567)

[ Slike vseh menijev 38](#_Toc353991568)

[ Namestitveni CD za igrico Spopad titanov 41](#_Toc353991569)

[ Sklep o odobritvi naslova maturitetnega izdelka 41](#_Toc353991570)

# Uvod

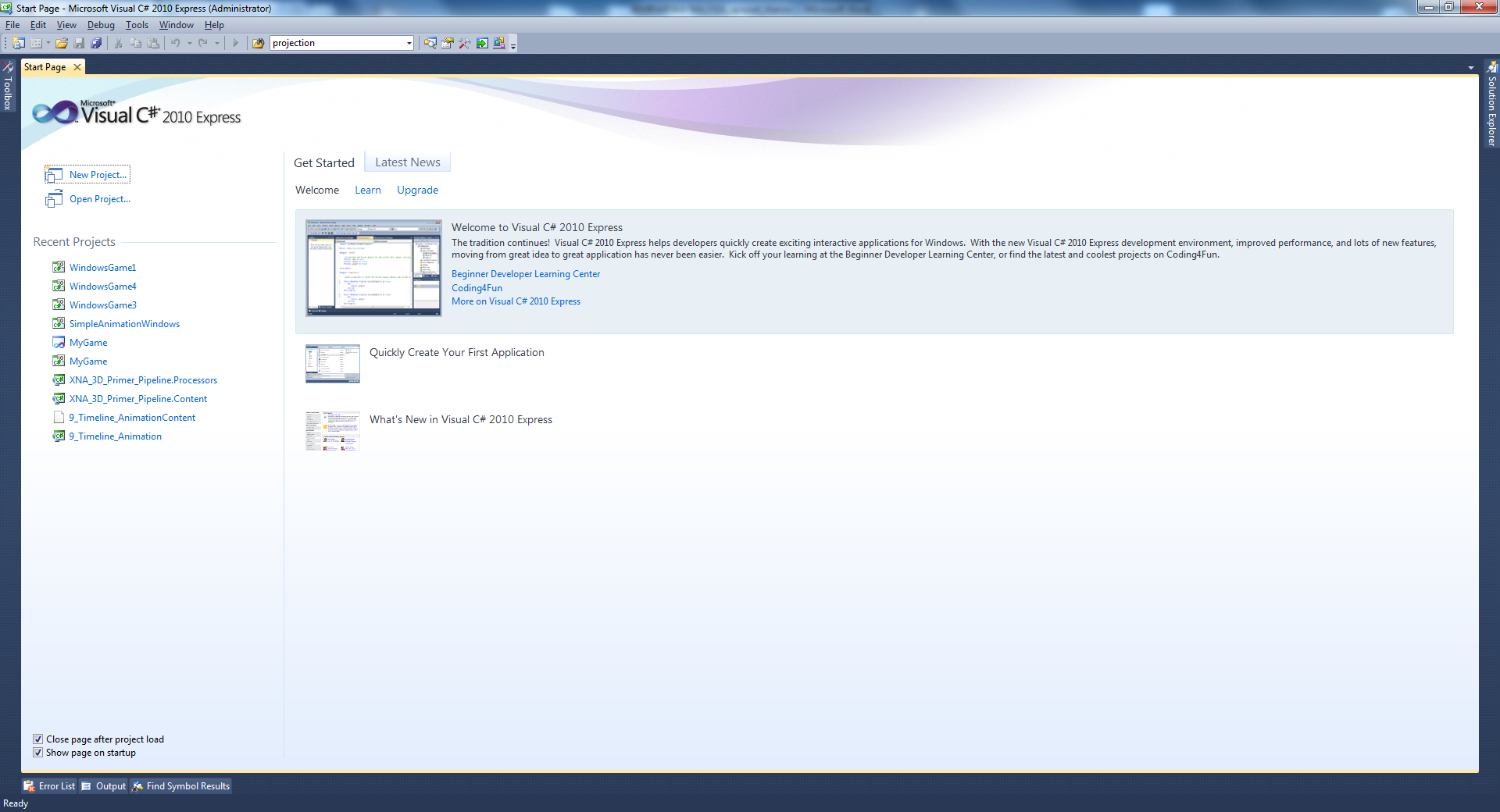
V seminarski nalogi Vam bom na kratko opisal, kako deluje moj maturitetni izdelek in kako sem ga ustvaril. Ta seminarska naloga je predvsem dokumentacija za mojo 3D-igrico. Igrica nosi naslov Spopad titanov. Zamisel za tako vrsto igrice sem dobil med gledanjem filma »jeklena moč« (angl. *Real steel*). Problem je bil le to, da se mi naslov ni zdel preveč udaren. Odločil sem se, da bom uporabil naslov od drugega filma, ki pa v samem pomenu nima nikakršne povezave z mojo 3D-igrico. Naslov, ki sem ga vzel, pa sem dobil iz filma Spopad titanov (angl. *Clash of the Titans*).

# Uporabljena programska oprema

## Integrirano razvojno okolje

Za izdelovanje računalniške igrice sem se odločil, da bom za integrirano razvojno okolje (angl. IDE) uporabil Microsoft Visual Studio Express 2010. Ker sem se odločil, da bom 3D-igrico naredil v programskem jeziku C#, se integrirano razvojno okolje na kratko imenuje kar C# Express 2010. Za grafični vmesnik sem se odločil, da bom uporabil XNA game studio 4.0.

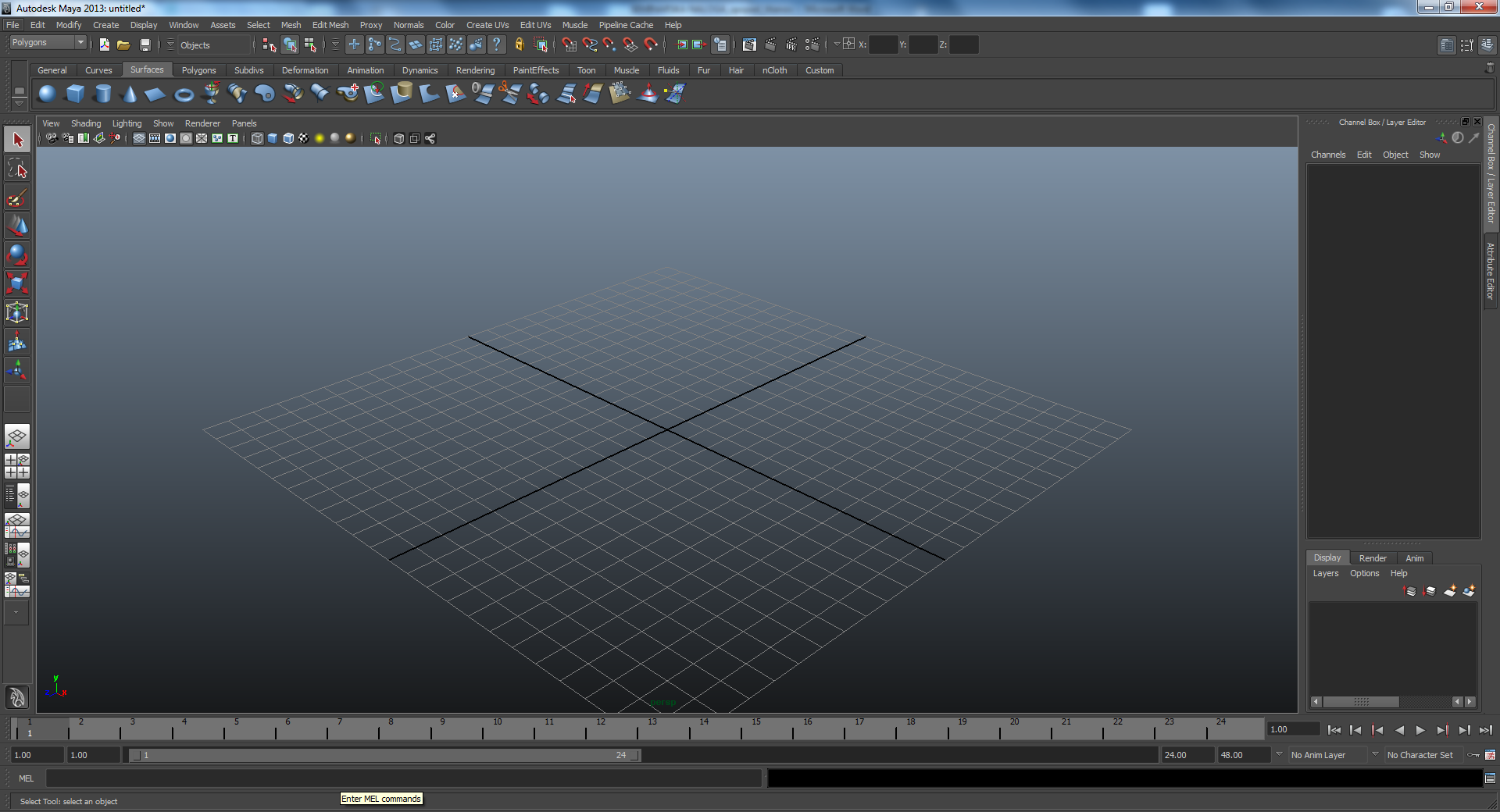
Slika 1: Microsoft Visual Studio 2010



## Program za oblikovanje 3D modelov

Za izdelavo igrice pa sem potreboval tudi programsko opremo za risanje 3D-modelov. Odločil sem se za program Autodesk Maya 2013. Program ima veliko uporabnih funkcij za 3D-modeliranje, ter podpira izvoz datotek formata [ .fbx ] in [ .x ]. Datoteke s takšnim formatom, se lahko uvozijo v XNA game studio, brez kakršnih koli dodatnih vmesnikov. Autodesk Maya pa poleg mnogih uporabnih funkcij za oblikovanje 3D-modelov, ponuja tudi možnost izdelave animacije.

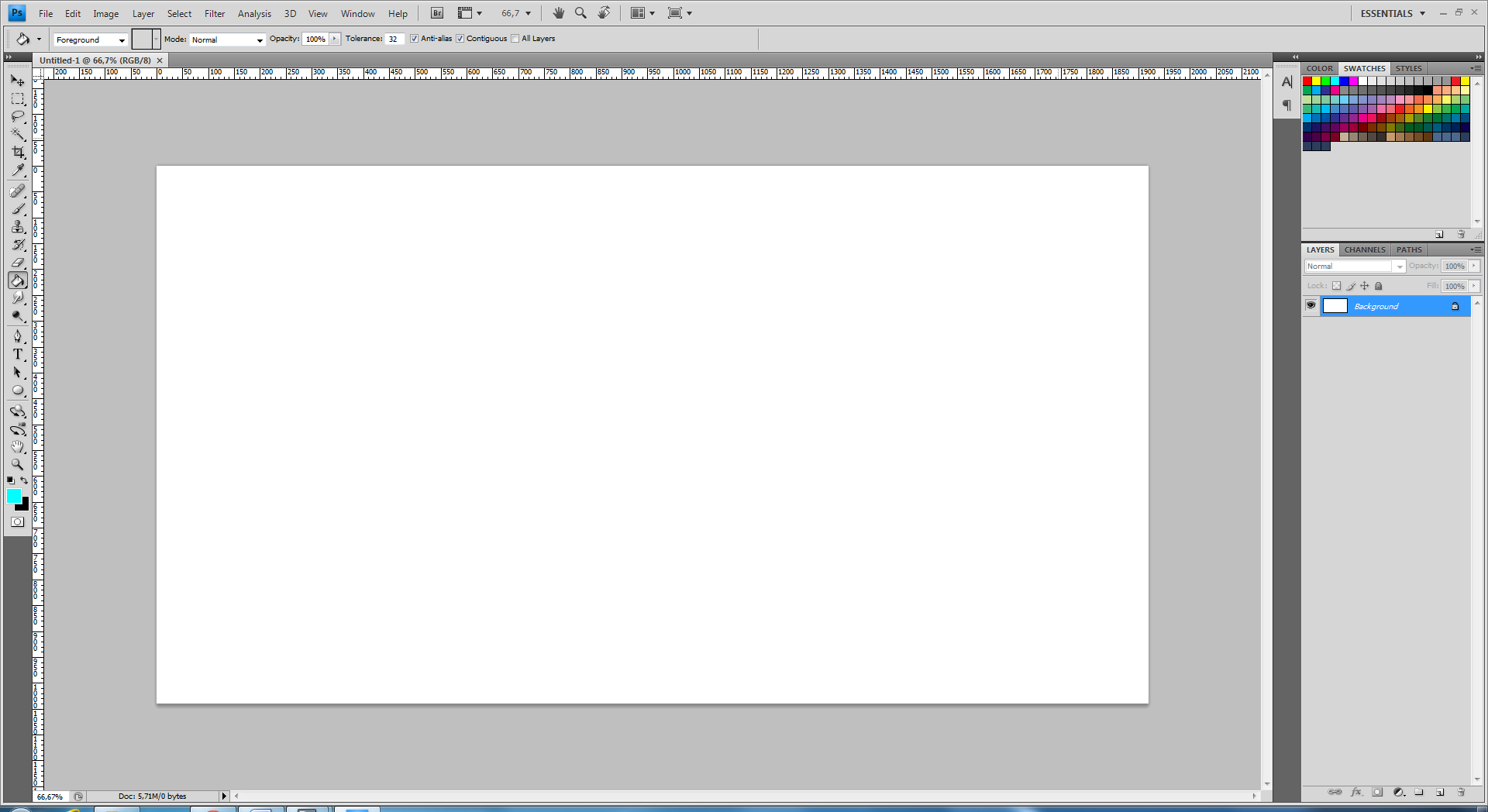
Slika 2: Autodesk Maya 2013



## Program za oblikovanje slik

Čeprav bi bila za osnovno igro dovolj le zgoraj navedena programa, pa sem za izdelavo menija potreboval program, s katerim lahko ustvarjam, preoblikujem in izvažam slike v različne formate. Ker sem bil z enim izmed takih programov že seznanjen, sem se odločil, da ga bom uporabil tudi v tem primeru. Program se imenuje Adobe Photoshop CS4. Program Photoshop je zelo priljubljen program in tudi sam lahko povem, da sem s programom zelo zadovoljen. Celoten meni v igrici je narejen s pomočjo Photoshop-a (ozadje, gumbi, slike, itd.).

Slika 3: Adobe Photoshop CS4



# Oblikovanje 3D-igrice

Na začetku izdelovanja 3D-igrice sem si zamisli, kako naj bi igrica izgledala. Poskušal sem narediti takšen meni, kot sem si ga zamisli. Prenesti zamisel, kako naj bi meni izgledal, v program Adobe Photoshop pa ni bila lahka naloga. Vedno, ko sem hotel oblikovati meni, mi je nastala nova, boljša ideja o izgledu. Idej je bilo toliko, da sem se moral odločiti za tisto, za katero mislim, da bo najbolj ustrezala 3D-igrici z naslovom Spopad titanov.

Ko sem končal z oblikovanjem menija, sem hitro ugotovil, da bi bilo še najboljše narediti iz **enega projekta v dva projekta**. Razlog za to je bila predvsem urejenost podatkov oziroma programske kode. Odločil sem se, da bo moj prvi projek izdelava menija in drugi projekt izdelava 3D-igrice. Tako sem si lahko priskrbel večjo urejenost programske kode, kot tudi urejenost posameznih datotek (slik, modelov, pisav, itd.) v različne direktorije.

Ko sem dokončal oblikovanje menija, sem začel z oblikovanjem 3D-modela (model glavnega igralca) in oblikovanjem okolja v katerem se bo igranje izvajalo (arena). Model glavnega igralca je oblikovan kot mehanični robot. Okolje, v katerem pa bo lahko glavni lik igral, pa je oblikovano kot boksarska arena.

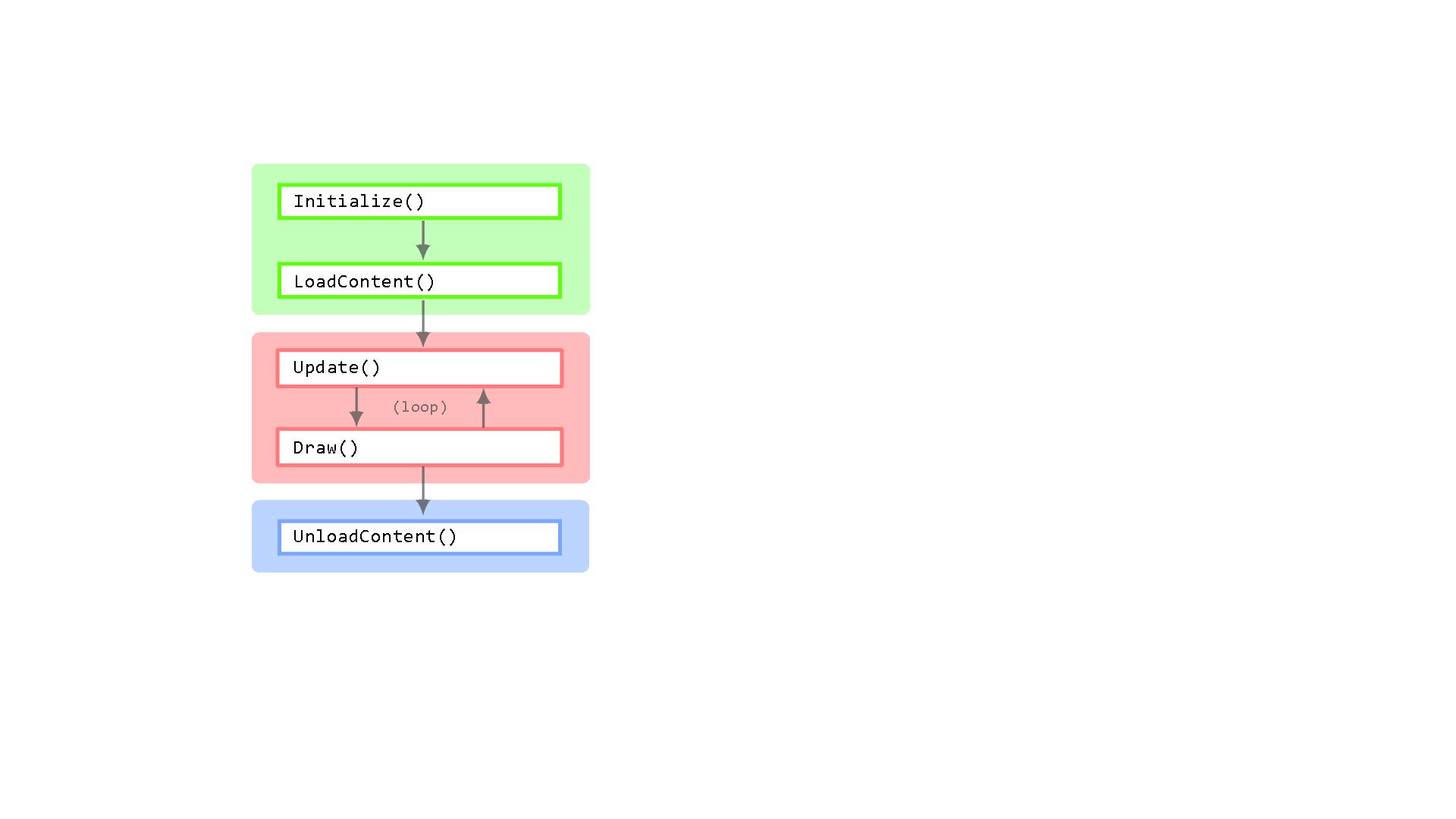
# Osnovna sestava XNA GS 4. 0 grafične knjižnice

XNA grafična knjižnica je sestavljena iz 5-ih glavih funkcij, ki se kličejo ob zagonu programa. Te funkcije poskrbijo za inicializacijo spremenljivk, prednaložitev objektov, posodabljanje spremenljivk, izrisovanje grafike na zaslon in odstranjevanje objektov (sprostitev delovnega pomnilnika).

Ob zagonu igrice, se prva kliče funkcija za inicializacijo spremenljivk. V tej funkciji lahko določimo spremenljivkam začetno vrednost. Ko program izvrši celotno programsko kodo v tej funkciji, se nato izvrši funkcija za prednaložitev objektov. Ta funkcija poskrbi, da lahko v program naložimo slike, filme in druge datoteke. Ko program konča z funkcijo za prednaložitev objektov, se premakne v funkcijo za posodabljanje.

Funkcija posodabljanja ima v celotnem programu največjo prioriteto saj skrbi zato, da: spremlja pritisnjene tipke na tipkovnici, povečuje vrednosti spremenljivkam ter skrbi za matematične operacije. Po posodabljanju je na vrsti funkcija za izrisovanje. Funkcija izriše na zaslon vso našo grafčno podobo. Po izrisovanju pa program ne nadaljuje v funkcijo odstranjevanje, ampak se vrne v funkcijo posodabljanje. Ta zanka poskrbi, da se igrica ne prekine ampak deluje vse dokler se ne pokliče funkcije za odstranjevanje. Le ta pa bo poskrbela za odstranitev naloženih objektov in nato končala igrico.

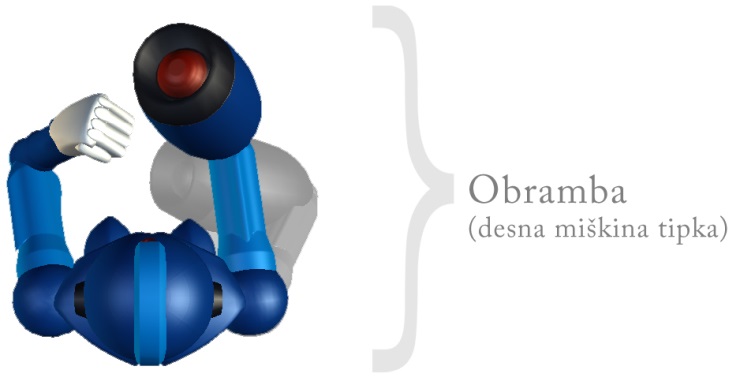
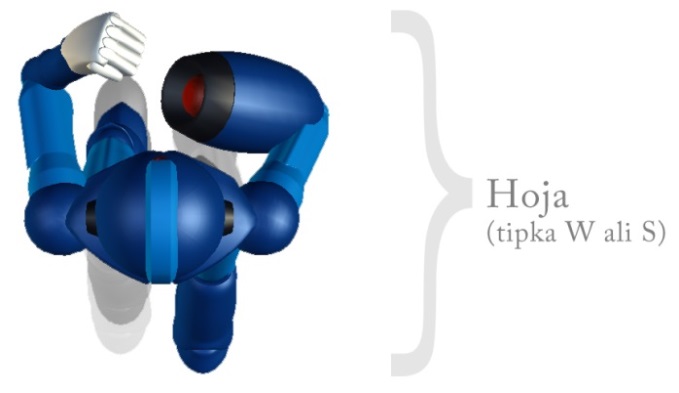
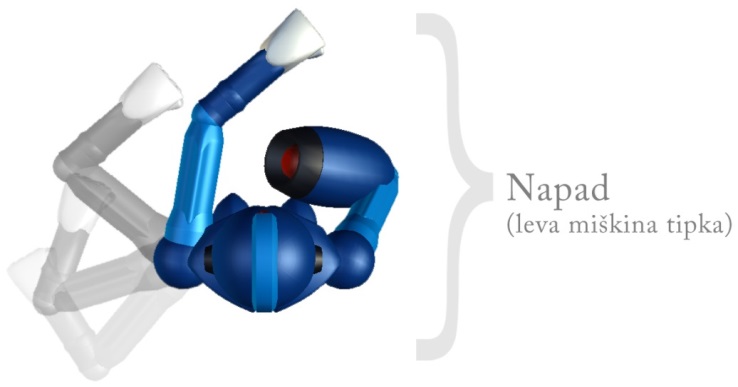
Slika 4: Struktura programa



# Nasveti za igranje

Za igranje 3D-igrice sem se odločil, da bom uporabil tipke na tipkovnici in miškine tipke. Za napad, je določena leva miškina tipka in za obrambo desna miškina tipka. Za hojo naprej sem izbral tipko W in za hojo nazaj tipko S. Za obračanje modela se uporabljata tipki A in D. V igrici je tudi poseben napad, ki se izvede ob pritisku na gumb Space.

Slika 5: Hoja, obramba, napad in posebni udarec

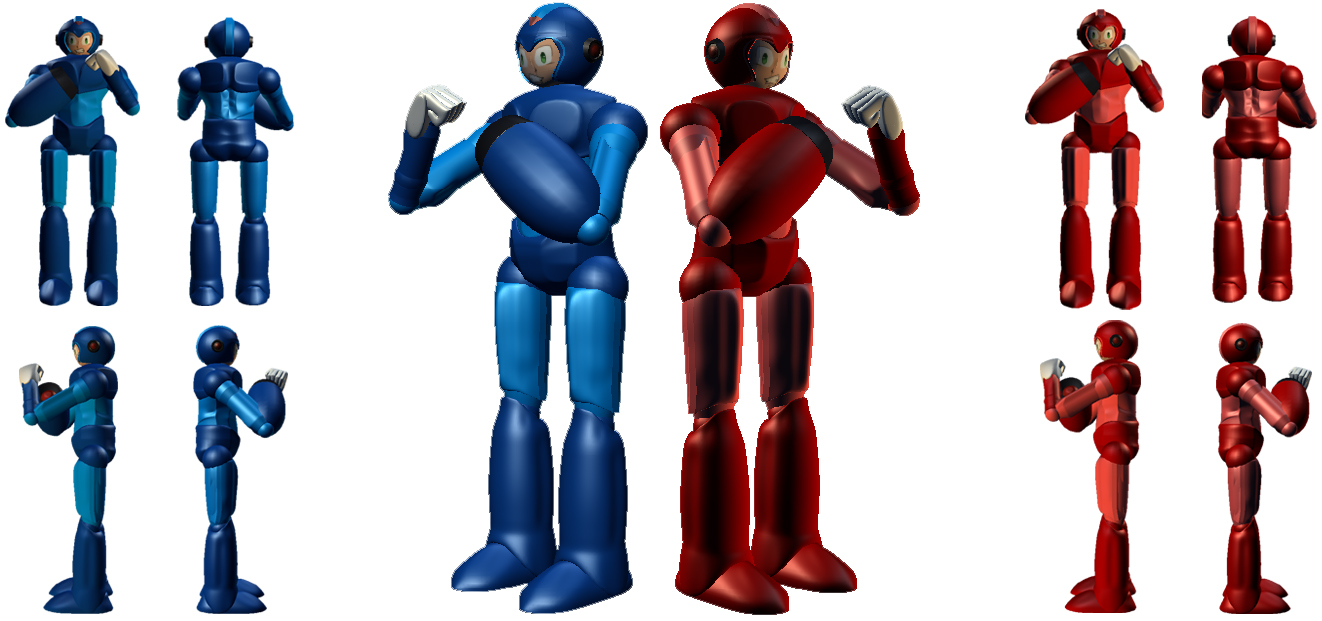


# Opis igralnega lika

Za 3D-igrico Spopad titanov, sem ustvaril 2 robota. Po obliki sta povsem enaka, le po barvah se razlikujeta. Modrega robot lahko igra igralec, rdečega robota pa lahko igra le računalnik.

Na spodnji sliki lahko opazimo prikaz obeh robotov. Prikazana sta z stranskega pogleda, pogleda od spredaj, pogleda od zadaj in pogled iz perspektive.

Slika 6: Prikaz obeh igralniških robotov



# Projekt 1: Izdelava menija

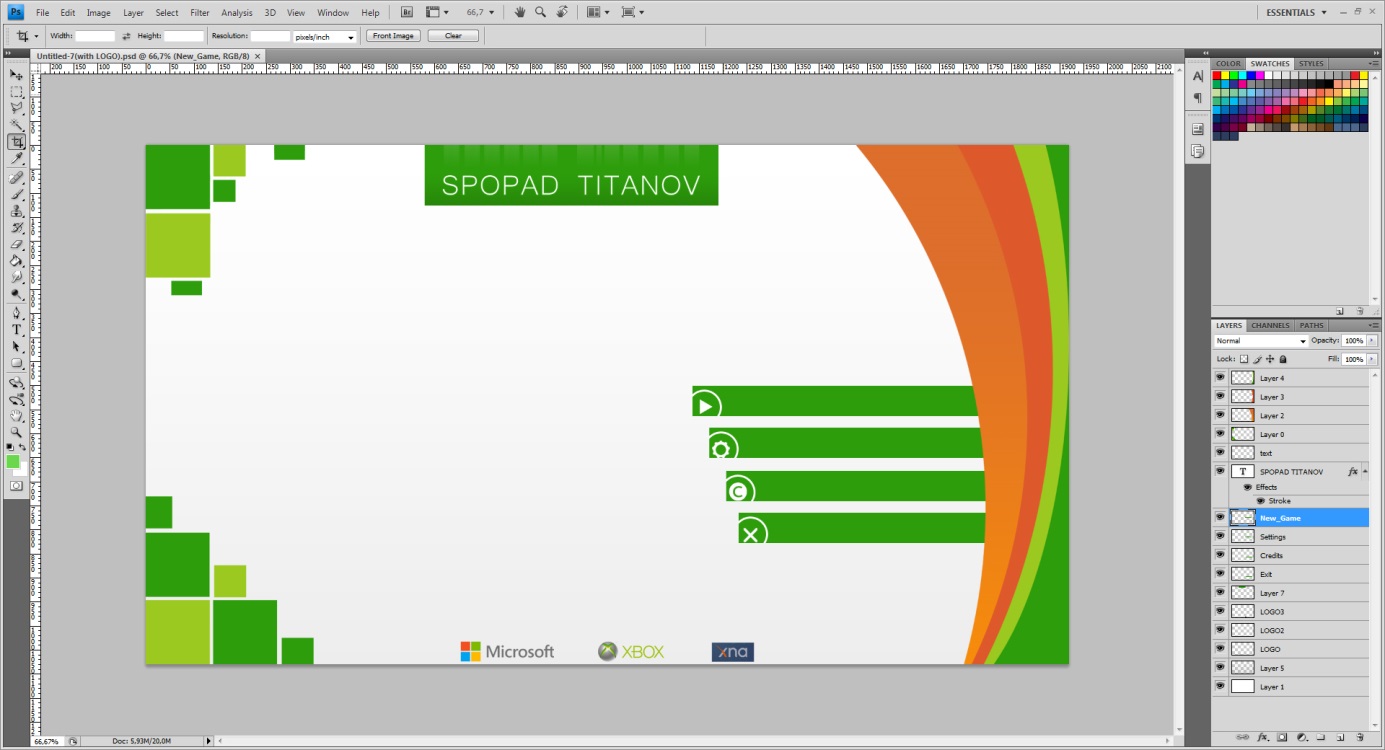
## Grafično oblikovanje menija

Idejo za grafični izgled menija sem dobil, ko sem med pregledovanjem spletnih strani na temo »C# and XNA«. Pojavila se je stran XBOX Live indie games (URL: http://xbox.create.msdn.com). Na vrhu zgoraj navedene spletne strani je bila pozdravna slika, pri kateri sem dobil idejo o tem, kako naj bi izgledal meni. Dodal sem še različne vrste gumbov in podmenije, v katere lahko vstopimo ob pritisku na določen gumb.

V glavnem meniju imamo na izbiro 4 gumbe:

* Ob pritisku na gumb **Nova igra**, se pokaže podmeni, kjer lahko izbiramo med igranjem in prikazom rezultatov (angl. s*core*). Ob kliku na gumb **Igraj** se zažene projekt 2 in začnemo z igranjem. Ob kliku na gumb **Rezultati** pa se odpre podmeni z prikazom 10-ih najboljših rezultatov.
* Ob pritisku na gumb **Nastavitve**, se nam odpre podmeni, kjer lahko spreminjamo nastavitve za 3D-igrico (resolucija, barvna globina, celozaslonski način, zvok)
* Ob pritisku na gumb **Vizitka** lahko preberemo, kako je bila igrica oblikovana
* Ob pritisku na gumb **Izhod** se igrica zapre

Slika 7: Oblikovanje glavnega menija



## Programska koda (opis)

Na začetku programa sem moral deklarirati vse spremenljivke potrebne za pravilno delovanje in prikazovanje menija.

GraphicsDeviceManager graphics;

SpriteBatch spriteBatch;

/\* SPREMENLJIVKE ZA SLIKE \*/

Texture2D front;

Texture2D back;

Texture2D front\_settings;

Texture2D front\_credits;

Texture2D[] menuButtons = new Texture2D[4];

Texture2D regularButton;

Texture2D on\_left, off\_left;

Texture2D on\_right, off\_right;

Texture2D right, left;

/\* SPREMENLJIVKE ZA RAČUNALNIŠKO MIŠKO \*/

MouseState current\_mouse;

int mouseX, mouseY;

/\* SPREMENLJIVKE ZA TIPKOVNICO \*/

KeyboardState newState = Keyboard.GetState();

/\* SPREMENLJIVKE ZA PISAVO \*/

SpriteFont font\_menu;

SpriteFont font\_settings;

/\* SPREMENLJIVKE ZA VIDEO \*/

Video vid;

VideoPlayer vidPlayer;

Rectangle vidRectangle;

Texture2D vidTexture;

/\* SPREMENLJIVKE ZA NASTAVITVE RESOLUCIJE IN ZVOKA \*/

int height;

int width;

int bpp;

bool sound;

bool fullScreen;

/\* DRUGE SPREMENLJIVKE \*/

Color myGreen = new Color(45, 157, 11); // RGB format

bool tmp1, tmp2;

int tmp3, tmp4;

Vector2 scale;

float scaleFactor;

int[,] choose\_res = new int[4, 2];

int moved = 60;

Color colorDefault = Color.White;

SpriteEffects spriteEffectDefault = SpriteEffects.None;

int[,] meni\_positions = new int[4, 2];

int menuButtons0x = 1200;

int menuButtons0y = 500;

string text;

/\* USTVARIMO IZBIRO MED ZASTAVICAMI ZA PREKLOP MED MENIJI \*/

enum gameState

{

Menu,

NewGame\_menu,

Settings\_menu,

Credits\_menu,

Game,

Intro,

}

gameState game\_state = gameState.Menu;

Ko sem deklariral vse spremenljivke, sem moral nastaviti tudi nekatere privzete nastavitve (angl. *Default value*). Vrednosti sem jim določil kar v konstruktorju razreda **Game1**. Med temi nastavitvami so bile predvsem nastavitve za grafično kartico.

public Game1()

{

graphics = new GraphicsDeviceManager(this);

setResolution();

tmp1 = sound;

tmp2 = fullScreen;

tmp3 = bpp;

choose\_res[3, 0] = 1920;

choose\_res[3, 1] = 1080;

choose\_res[2, 0] = 1600;

choose\_res[2, 1] = 900;

choose\_res[1, 0] = 1366;

choose\_res[1, 1] = 768;

choose\_res[0, 0] = 1280;

choose\_res[0, 1] = 720;

graphics.SynchronizeWithVerticalRetrace = false;

// Uporabi funkcijo za izboljševanje robov

graphics.PreferMultiSampling = true;

// Nastavi privzet direktorij za nalaganje datotek v program

Content.RootDirectory = "Content";

}

Čeprav sem določil privzete vrednosti za spremenljivke pa sem moral le te spremenljivke nastaviti oziroma potrditi (angl. *apply*) grafični kartici. Napisal sem novo funkcijo **setResolution**, ki poskrbi, da se nastavitve preberejo iz datoteke v spremenljivke. Nato funkcija spremenljivke nastavi grafični kartici.

void setResolution()

{

// Preberi iz XML datoteke

readFromXML(ref height, ref width, ref bpp, ref sound, ref fullScreen);

graphics.PreferredBackBufferHeight = height;

graphics.PreferredBackBufferWidth = width;

if(bpp == 32)

graphics.PreferredBackBufferFormat = SurfaceFormat.Color;

else

graphics.PreferredBackBufferFormat = SurfaceFormat.Bgr565;

graphics.IsFullScreen = fullScreen;

switch (width)

{

case 1920: tmp4 = 3; break;

case 1600: tmp4 = 2; break;

case 1366: tmp4 = 1; break;

case 1280: tmp4 = 0; break;

}

// Vse nastavitve imajo ratio 16:4

scaleFactor = (1920 / (float)width);

scaleFactor = (float)(1 / scaleFactor);

moved \*= Convert.ToInt32(scaleFactor);

scale.X = scaleFactor;

scale.Y = scaleFactor;

graphics.ApplyChanges();

}

Inicializiral sem tudi vse potrebne spremenljivke, za katere bom uporabil slike. Te slike se nahajajo v privzetem direktoriju. Prav tako sem inicializiral spremenljivke, ki potrebuje datoteke z pisavami. Prav tako kot slike, se tudi te datoteke nahajajo v privzetem direktoriju. Znak @ pomeni, da ni posembnih znakov v besedilu, ki bi povzročali težave (\n, \b, \a ). Prav tako lahko namesto znaka @ uporabimo dve poševni črti \\.

protected override void Initialize()

{

/\* INICIALIZACIJA SPREMENLJIVK ZA MENU \*/

front = Content.Load<Texture2D>(@"Pictures\front\_layer");

back = Content.Load<Texture2D>(@"Pictures\back\_layer");

front\_settings = Content.Load<Texture2D>(@"Pictures\front\_layer\_settings");

front\_credits = Content.Load<Texture2D>(@"Pictures\front\_layer\_credits");

regularButton = Content.Load<Texture2D>(@"Pictures\Button");

on\_left = Content.Load<Texture2D>(@"Pictures\on\_left");

on\_right = Content.Load<Texture2D>(@"Pictures\on\_right");

off\_left = Content.Load<Texture2D>(@"Pictures\off\_left");

off\_right = Content.Load<Texture2D>(@"Pictures\off\_right");

right = Content.Load<Texture2D>(@"Pictures\right");

left = Content.Load<Texture2D>(@"Pictures\left");

menuButtons[0] = Content.Load<Texture2D>(@"Pictures\\New\_game");

menuButtons[1] = Content.Load<Texture2D>(@"Pictures\\Settings");

menuButtons[2] = Content.Load<Texture2D>(@"Pictures\\Credits");

menuButtons[3] = Content.Load<Texture2D>(@"Pictures\\Exit");

//nastavitve za video najvišja ločljivost 1280×720

vidPlayer = new VideoPlayer();

vid = Content.Load<Video>("Video\\xna");

vidRectangle = new Rectangle( GraphicsDevice.Viewport.X,

GraphicsDevice.Viewport.Y, GraphicsDevice.Viewport.Width,

GraphicsDevice.Viewport.Height );

//nastavitve za pisave

font\_menu = Content.Load<SpriteFont>("Fonts\\SpriteFont\_main");

font\_settings = Content.Load<SpriteFont>("Fonts\\SpriteFont\_settings");

//prikaz kazalca

this.IsMouseVisible = true;

base.Initialize();

}

protected override void LoadContent()

{

// Ustavri nov SpriteBatch za risanje 2DTexture

spriteBatch = new SpriteBatch(GraphicsDevice);

}

Ker ločljivost grafike beremo iz XML datoteke sem moral poskrbeti, da se v datoteki nahajajo podatki (datoteka ne sme biti prazna). Preden sem lahko preizkusil ali celoten program deluje, sem napisal funkcijo **writeToXML**. Funkcija je poskrbela, da so se podatki zapisali v datoteko, kot tudi vsi komentarji, ki spadajo k podatkom.

static void writeToXML(int h, int w, int b, bool s, bool f)

{

// Ustvari novo XML datoteko, null pomeni privzeto kodiranje (utf-8)

XmlTextWriter textWriter = new XmlTextWriter(@"Content/myXmFile.xml", null);

// XML elementi naj bodo poravnani

textWriter.Formatting = Formatting.Indented;

// Odpre datoteko in čaka na zapisovanje

textWriter.WriteStartDocument();

// Zapiše komentar

textWriter.WriteComment("DEFAULT NASTAVITVE ZA IGRICO");

// Zapiše prvi element (angl. Root element)

textWriter.WriteStartElement("Settings");

/\* ZAPIŠI NASLEDNJE ELEMENTE \*/

textWriter.WriteStartElement("Width");

textWriter.WriteValue(w);

textWriter.WriteEndElement();

textWriter.WriteStartElement("Height");

textWriter.WriteValue(h);

textWriter.WriteEndElement();

textWriter.WriteStartElement("BPP");

textWriter.WriteValue(b);

textWriter.WriteEndElement();

textWriter.WriteStartElement("Full-screen");

if(f == true)

textWriter.WriteString("yes");

else

textWriter.WriteString("no");

textWriter.WriteEndElement();

textWriter.WriteStartElement("Sound");

if (s == true)

textWriter.WriteString("yes");

else

textWriter.WriteString("no");

textWriter.WriteEndElement();

// konča z zapisovanjem

textWriter.WriteEndDocument();

// zapre zapisovalnik

textWriter.Close();

}

Ko so bili podatki zapisani v datoteko, so postali uporabni za aplikacijo. Da pa podatke lahko uporabimo za grafično kartico, sem moral napisati funkcijo za branje podatkov iz XML datoteke. Funkcijo sem poimenoval **readFromXML**. Kot vhodne parametre sem v funkcijo poslal spremenljivke za višino, širino, bpp, zvok in celozaslonski način.

static void readFromXML(ref int h, ref int w, ref int b, ref bool s, ref bool f)

{

XmlTextReader textReader = new XmlTextReader(@"Content/myXmFile.xml");

textReader.Read();

while (textReader.Read())

{

textReader.MoveToElement();

switch (textReader.Name) {

case "Height":

h = textReader.ReadElementContentAsInt();

break;

case "Width":

w = textReader.ReadElementContentAsInt();

break;

case "BPP":

b = textReader.ReadElementContentAsInt();

break;

case "Sound":

if (textReader.ReadElementContentAsString() == "yes")

s = true;

else

s = false;

break;

case "Full-screen":

if (textReader.ReadElementContentAsString() == "no")

f = false;

else

f = true;

break;

}

textReader.MoveToNextAttribute();

}

textReader.Close();

}

Naslednje funkcije v programu predstavljajo predvsem preglednejšo programsko kodo in pa optimizacijo za izrisovanje 2D slik ter besedila na zaslon. Funkcije majo možnost tudi takojšnjega spreminjanja slike, v razmerju z velikostjo aplikacije (ločljivostjo).

void apply\_PictureAndText(Texture2D picture, Vector2 pic\_position, SpriteFont font,

Vector2 text\_position, Color text\_color, string text=null)

{

pic\_position.X \*= scaleFactor;

pic\_position.Y \*= scaleFactor;

text\_position.X \*= scaleFactor;

text\_position.Y \*= scaleFactor;

spriteBatch.Draw(picture, pic\_position, null, colorDefault, 0f, Vector2.Zero,

scale, spriteEffectDefault, 0f);

if (text != null)

spriteBatch.DrawString(font, text, text\_position, text\_color, 0f,

Vector2.Zero, scale, spriteEffectDefault, 0f);

}

void apply\_Picture(Texture2D picture, Vector2 pic\_position)

{

pic\_position.X \*= scaleFactor;

pic\_position.Y \*= scaleFactor;

spriteBatch.Draw(picture, pic\_position, null, colorDefault, 0f, Vector2.Zero,

scale, spriteEffectDefault, 0f);

}

void apply\_Text(SpriteFont font, Vector2 text\_position, Color text\_color, string

text = null)

{

text\_position.X \*= scaleFactor;

text\_position.Y \*= scaleFactor;

if (text == null)

text = " ";

spriteBatch.DrawString(font, text, text\_position, text\_color, 0f,

Vector2.Zero, scale, spriteEffectDefault, 0f);

}

Naslednja programska koda združuje vse zgoraj navedene funkcije in skupaj prikazuje meni kot celoto. V programski kodi so tudi pogojni stavki, ki preverjajo ali je kazalec prekril gumb ali ne. V primeru, da je kazalec prekril gumb se mora izvesti določena »animacija«. Tukaj je programska koda za glavni meni. Za ostale podmenije je koda skoraj popolnoma enaka.

public void Meni(GameTime gameTime)

{

spriteBatch.Begin();

apply\_Picture(back, Vector2.Zero);

if (mouseX > (menuButtons0x \* scaleFactor) && mouseX < ((menuButtons0x + 535)

\* scaleFactor) && mouseY > (menuButtons0y \* scaleFactor) && mouseY <

((menuButtons0y + 60) \* scaleFactor))

{

apply\_PictureAndText(menuButtons[0], new Vector2((menuButtons0x - 50),

(menuButtons0y)), font\_menu, new Vector2((menuButtons0x + 100 - moved),

(menuButtons0y)), colorDefault, "NOVA IGRA");

if (current\_mouse.LeftButton == ButtonState.Pressed)

game\_state = gameState.NewGame\_menu;

}

else

apply\_PictureAndText(menuButtons[0], new Vector2((menuButtons0x),

(menuButtons0y)), font\_menu, new Vector2((menuButtons0x + 100),

(menuButtons0y)), colorDefault, "NOVA IGRA");

if (mouseX > ((menuButtons0x + 50) \* scaleFactor) && mouseX < (((menuButtons0x

+ 50) \* scaleFactor) + (495 \* scaleFactor)) && mouseY > ((menuButtons0y +

80) \* scaleFactor) && mouseY < ((menuButtons0y + 80 + 60) \* scaleFactor))

{

apply\_PictureAndText(menuButtons[1], new Vector2((menuButtons0x + 50 –

moved), (menuButtons0y + 80)), font\_menu, new Vector2((menuButtons0x +

150 - moved), (menuButtons0y + 80)), colorDefault, "NASTAVITVE");

if (current\_mouse.LeftButton == ButtonState.Pressed)

game\_state = gameState.Settings\_menu;

}

else

apply\_PictureAndText(menuButtons[1], new Vector2((menuButtons0x + 50),

(menuButtons0y + 80)), font\_menu, new Vector2((menuButtons0x + 150),

(menuButtons0y + 80)), colorDefault, "NASTAVITVE");

if (mouseX > ((menuButtons0x + 70) \* scaleFactor) && mouseX < (((menuButtons0x

+ 70) \* scaleFactor) + (480 \* scaleFactor)) && mouseY > ((menuButtons0y +

160) \* scaleFactor) && mouseY < ((menuButtons0y + 160 + 60) \*

scaleFactor))

{

apply\_PictureAndText(menuButtons[2], new Vector2((menuButtons0x + 70 –

moved), (menuButtons0y + 160)), font\_menu, new Vector2((menuButtons0x +

170 - moved), (menuButtons0y + 160)), colorDefault, "VIZITKA");

if (current\_mouse.LeftButton == ButtonState.Pressed)

game\_state = gameState.Credits\_menu;

}

else

apply\_PictureAndText(menuButtons[2], new Vector2((menuButtons0x + 70),

(menuButtons0y + 160)), font\_menu, new Vector2((menuButtons0x + 170),

(menuButtons0y + 160)), colorDefault, "VIZITKA");

if (mouseX > ((menuButtons0x + 90) \* scaleFactor) && mouseX < (((menuButtons0x

+ 90) \* scaleFactor) + (460 \* scaleFactor)) && mouseY > ((menuButtons0y +

240) \* scaleFactor) && mouseY < ((menuButtons0y + 240 + 60) \* scaleFactor))

{

apply\_PictureAndText(menuButtons[3], new Vector2((menuButtons0x + 90 –

moved), (menuButtons0y + 240)), font\_menu, new Vector2((menuButtons0x +

190 - moved), (menuButtons0y + 240)), colorDefault, "IZHOD");

if (current\_mouse.LeftButton == ButtonState.Pressed)

{

UnloadContent();

Exit();

}

}

else

apply\_PictureAndText(menuButtons[3], new Vector2((menuButtons0x + 90),

(menuButtons0y + 240)), font\_menu, new Vector2((menuButtons0x + 190),

(menuButtons0y + 240)), colorDefault, "IZHOD");

apply\_Picture(front, Vector2.Zero);

spriteBatch.End();

}

void Settings\_menu(){. . . }

void Credits\_menu(){. . . }

void new\_Game\_menu(){. . . }

void Scores\_menu(){. . . }

Ker se miškin kazalec nenehno premika (ob premiku miške), se prav tako spreminjata x in y koordinati kazalca. Zato je potrebno zagotoviti preverjanje x in y koordinat, ali prekrivata gumb ali ne. Spodnji dve funkciji skrbita za zgoraj navedeno preverjanje.

protected override void Update(GameTime gameTime)

{

// Preveri ali je bila pritisnjena tipka ESC

if (Keyboard.GetState().IsKeyDown(Keys.Escape))

this.Exit();

// Posodobi x in y koordinati miškinega kazalca

updateMouse();

base.Update(gameTime);

}

void updateMouse()

{

current\_mouse = Mouse.GetState();

mouseX = current\_mouse.X;

mouseY = current\_mouse.Y;

}

In na vrsti je še zadnja pomembna funkcija iz projekta menu. Ta funkcija se imenuje **Draw**. Skrbi predvsem za izrisovanje objektov na zaslon, ter skrbi tdi za preklapljanje med različnimi meniji. Predno pa začne z izrisovanjem pa poskrbi, da grafična kartica izbriše vse predhodne elemente.

protected override void Draw(GameTime gameTime)

{

graphics.GraphicsDevice.Clear(Color.White);

switch (game\_state)

{

case gameState.Menu:

Meni(gameTime); break;

case gameState.NewGame\_menu:

new\_Game\_menu();

break;

case gameState.Game:

Exit();

break;

case gameState.Settings\_menu:

Settings\_menu(); break;

case gameState.Credits\_menu:

Credits\_menu(); break;

case gameState.Intro:

introVideo(); break;

case gameState.Scores:

Scores\_menu(); break;

}

base.Draw(gameTime);

}

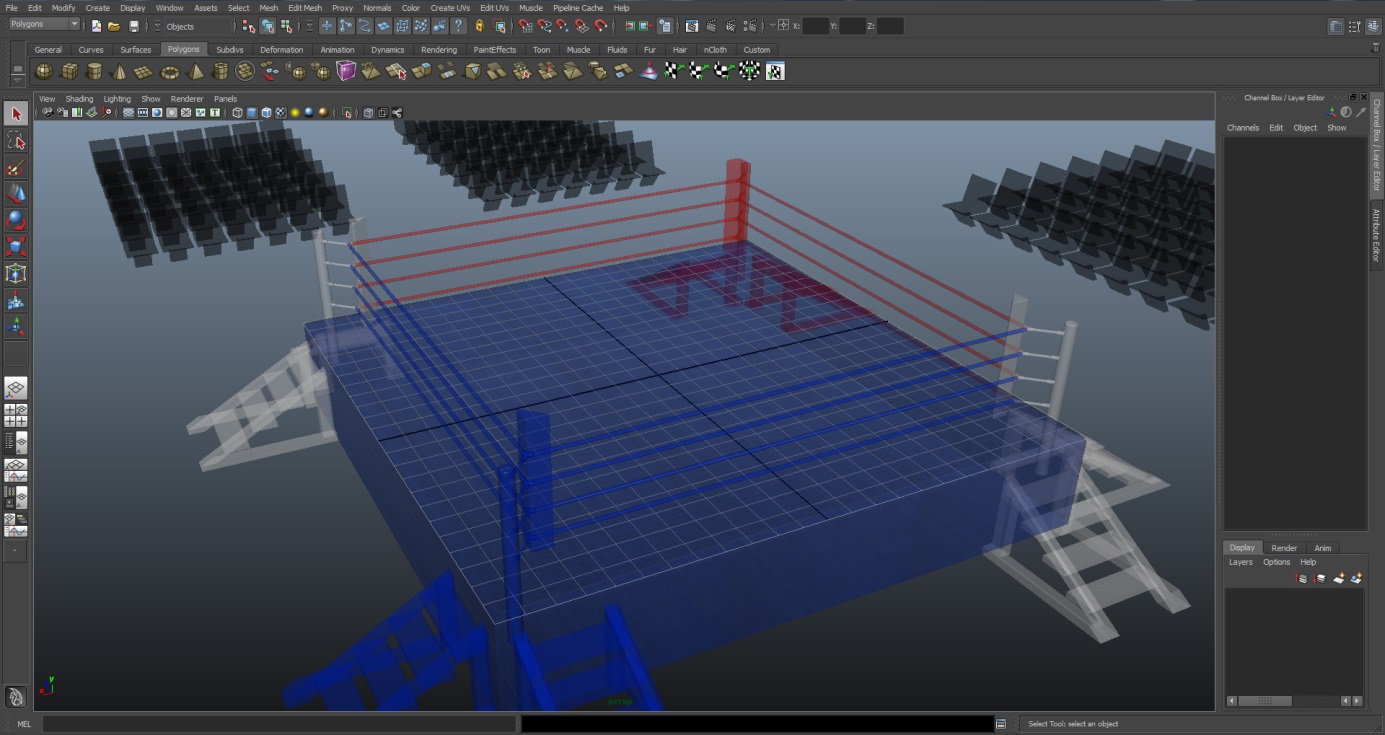
# Projekt 2: Izdelava glavne igrice

## Grafično oblikovanje igrice

Izdelovanje 3D igrice se ne začne samo z pisanjem programske kode, vendar je potrebno ustvariti tudi 3D objekte, ki bodo prisotni v igrici. Za oblikovanje 3D objektov sem si izbral program podjetja Autodesk. Ta program se imenuje Autodesk Maya. Ker sem imel nekaj izkušenj s tem programom že pred izdelovanjem maturitetnega izdelka, sem se odločil, da ga bom uporabil tudi v tem primeru.

Na začetku sem v programu Autodesk Maya izdelal ring in nato še stole, ki ga obkrožajo. Nato sem se lotil še izdelave stadiona oziroma prostora, kjer bo potekal spopad. Ko je bila izdelava okolja končana, sem se lotil še izdelave igralcev. Izdelal sem dva igralca (Rdečega in modrega).

Slika 8: Izdelovanje 3D objektov



## Programska koda (opis)

Tako kot sem pri projektu 1 deklariral vse potrebne spremenljivke, sem enako storil tudi pri projektu 2. Za razliko od projekta 1, pa sem tukaj uporabljal predvsem spremenljivke za 3D objekte in animacije.

/\* Objekti in igralci \*/

CModel player, computer;

CModel stadion, ring, statue;

/\* Za resolucijo \*/

Vector2 scale;

int width, height;

float scaleFactor;

/\* Animacije za objekte \*/

ObjectAnimation head;

ObjectAnimation left\_arm, left\_fist;

ObjectAnimation right\_arm, right\_fist;

ObjectAnimation left\_leg, left\_knee, left\_foot;

ObjectAnimation right\_leg, right\_knee, right\_foot;

/\* Position za objekte posameznega robota \*/

Vector3 headV, headR;

Vector3 left\_armV, left\_armR;

Vector3 left\_fistV, left\_fistR;

Vector3 right\_armV, right\_armR;

Vector3 right\_fistV, right\_fistR;

Vector3 right\_legV, right\_legR;

Vector3 right\_kneeV, right\_kneeR;

Vector3 right\_footV, right\_footR;

Vector3 left\_legV, left\_legR;

Vector3 left\_kneeV, left\_kneeR;

Vector3 left\_footV, left\_footR;

List<ObjectAnimation> transform = new List<ObjectAnimation>();

List<ObjectAnimationFrame> punch\_left\_arm = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> punch\_left\_fist = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> punch\_right\_arm = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> punch\_right\_fist = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> special\_left\_arm = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> special\_left\_fist = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_right\_legF = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_right\_kneeF = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_right\_footF = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_left\_legF = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_left\_kneeF = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_left\_footF = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_right\_legB = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_right\_kneeB = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_right\_footB = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_left\_legB = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_left\_kneeB = new List<ObjectAnimationFrame>();

List<ObjectAnimationFrame> walk\_left\_footB = new List<ObjectAnimationFrame>();

KeyframedObjectAnimation[] punch\_leftP = new KeyframedObjectAnimation[2];

KeyframedObjectAnimation[] punch\_rightP = new KeyframedObjectAnimation[2];

KeyframedObjectAnimation[] special\_punchP = new KeyframedObjectAnimation[2];

KeyframedObjectAnimation[] walkFP = new KeyframedObjectAnimation[6];

KeyframedObjectAnimation[] walkBP = new KeyframedObjectAnimation[6];

KeyframedObjectAnimation[] punch\_leftC = new KeyframedObjectAnimation[2];

KeyframedObjectAnimation[] punch\_rightC = new KeyframedObjectAnimation[2];

KeyframedObjectAnimation[] special\_punchC = new KeyframedObjectAnimation[2];

KeyframedObjectAnimation[] walkFC = new KeyframedObjectAnimation[6];

KeyframedObjectAnimation[] walkBC = new KeyframedObjectAnimation[6];

AnimPlayer animPlayer1 = new AnimPlayer();

AnimPlayer animPlayer2 = new AnimPlayer();

AnimPlayer animPlayer3 = new AnimPlayer();

AnimPlayer animPlayer4 = new AnimPlayer();

AnimPlayer animPlayer5 = new AnimPlayer();

AnimPlayer animPlayer6 = new AnimPlayer();

AnimPlayer animPlayer7 = new AnimPlayer();

AnimPlayer animPlayer8 = new AnimPlayer();

/\* Spremenljivke za robota \*/

string[] Objects = { "Head", "Left\_arm", "Left\_fist", "Right\_arm", "Right\_fist",

"Right\_leg", "Right\_knee", "Right\_foot", "Left\_leg", "Left\_knee", "Left\_foot" };

int healthP, healthC;

/\* Za funkcijo UPDATE() \*/

BoundingSphere player\_head, computer\_head, player\_punch\_sphere,

computer\_punch\_sphere, player\_defend\_sphere, computer\_defend\_sphere,

player\_special\_punch\_sphere, computer\_special\_punch\_sphere;

TimeSpan timer\_one, timer\_two, timer\_three, timer\_four, timer\_five, timer\_six;

SoundEffect soundMissed, soundPunched;

Vector3 rotChange\_one, rotChange\_two;

Vector3 player\_punch\_sphereV;

Song soundACDC;

V funkciji za inicializacijo sem inicializiral spremenljivke za oba igralca. Te spremenljivke so določale koordinate za vsak model v objektu.

protected override void Initialize()

{

spriteBatch = new SpriteBatch(GraphicsDevice);

spriteEffectDefault = SpriteEffects.None;

// Preberi animacijo za posamezne objekte, iz XML datoteke

readXML\_ForAnimations();

// Postavi glavo, roke in noge spremenljivke na začetne animacije

head = applyObjectDefaults(headV, headV, headR, headR, 0, true);

left\_arm = applyObjectDefaults(left\_armV, left\_armV, left\_armR, left\_armR, 0, true);

left\_fist = applyObjectDefaults(left\_fistV, left\_fistV, left\_fistR, left\_fistR, 0, true);

right\_arm = applyObjectDefaults(right\_armV, right\_armV, right\_armR, right\_armR, 0, true);

right\_fist = applyObjectDefaults(right\_fistV, right\_fistV, right\_fistR, right\_fistR, 0, true);

left\_leg = applyObjectDefaults(left\_legV, left\_legV, left\_legR, left\_legR, 0, true);

left\_knee = applyObjectDefaults(left\_kneeV, left\_kneeV, left\_kneeR, left\_kneeR, 0, true);

left\_foot = applyObjectDefaults(left\_footV, left\_footV, left\_footR, left\_footR, 0, true);

right\_leg = applyObjectDefaults(right\_legV, right\_legV, right\_legR, right\_legR, 0, true);

right\_knee = applyObjectDefaults(right\_kneeV, right\_kneeV, right\_kneeR, right\_kneeR, 0, true);

right\_foot = applyObjectDefaults(right\_footV, right\_footV, right\_footR, right\_footR, 0, true);

// Objekti spravi v seznam

transform.Add(head);

transform.Add(left\_arm);

transform.Add(left\_fist);

transform.Add(right\_arm);

transform.Add(right\_fist);

transform.Add(right\_leg);

transform.Add(right\_knee);

transform.Add(right\_foot);

transform.Add(left\_leg);

transform.Add(left\_knee);

transform.Add(left\_foot);

MediaPlayer.IsRepeating = true;

this.IsMouseVisible = true;

base.Initialize();

}

Nato je sledila funkcija za nalaganje objektov. Ta funkcija je poskrbela, da so se naložili vsi objekti (igralca, stadion in ring) ter poskrbela za položaj kamer.

protected override void LoadContent()

{

// Naloži vse potrebne 3D modele

player = newCModel("player\_blue", new Vector3(0, 400, 0), Vector3.Zero, new

Vector3(50f), GraphicsDevice);

computer = newCModel("player\_red", new Vector3(0, 400, 0), Vector3.Zero, new

Vector3(50f), GraphicsDevice);

ring = newCModel("ring", Vector3.Zero, Vector3.Zero, Vector3.One, GraphicsDevice);

stadion = newCModel("stadion", Vector3.Zero, Vector3.Zero, Vector3.One,

GraphicsDevice);

statue = newCModel("statue", Vector3.Zero, Vector3.Zero, Vector3.One,

GraphicsDevice);

// Nastavitve za sprednjo in zadnjo kamero

camera\_backward = new ChaseCamera(new Vector3(0, 1500, 2000), new

Vector3(0,900,0), new Vector3(0, 0, 0), GraphicsDevice);

camera\_menu = new ChaseCamera(new Vector3(100, 30, 0), new Vector3(0, 0, 0), new

Vector3(0, 0, 0), GraphicsDevice);

camera\_first = new ChaseCamera(new Vector3(0, 1150, -15), new Vector3(0, 1100, -

600), new Vector3(0, 0, 0), GraphicsDevice);

((ChaseCamera)camera\_first).distance=false;

/\* Nastavitve, za trk (angl. collision) \*/

player\_punch\_sphere.Radius = 50;

computer\_punch\_sphere.Radius = player\_punch\_sphere.Radius;

player\_special\_punch\_sphere.Radius = player\_punch\_sphere.Radius;

computer\_special\_punch\_sphere.Radius = player\_punch\_sphere.Radius;

player\_head.Center = player.Position;

player\_head.Radius = 130;

computer\_head.Center = computer.Position;

computer\_head.Radius = player\_head.Radius;

player\_defend\_sphere.Radius = 120;

player\_defend\_sphere.Center = player.Position;

computer\_defend\_sphere.Radius = player\_defend\_sphere.Radius;

computer\_defend\_sphere.Center = computer.Position;

// Začetne pozicije za igralca in nasprotnika

player.Position = new Vector3(1500, 0, -1500);

player.Rotation = new Vector3(0, MathHelper.ToRadians(135), 0);

computer.Position = new Vector3(-1500, 0, 1500);

computer.Rotation = new Vector3(0, -MathHelper.ToRadians(45), 0);

statue.Position = new Vector3(0, -20, 28);

statue.Scale = new Vector3(1.5f, 1.5f, 1.5f);

// Za kamero

directionOfCamera = player.Position;

directionOfCamera.Y = 900;

}

Slednja funkcija, igra v celotnem projektu zelo važno vlogo. Skrbi za posodabljanje, matematične operacije in najpomembnejše umetno inteligenco. Spodnja funkcija **Update** predstavlja najverjetneje celotno logiko igrice.

protected override void Update(GameTime gameTime)

{

if (game\_state == gameState.Game)

{

if (healthC > 0 && healthP > 0)

{

collision2 = false;

collision1 = false;

player.Rotation = new Vector3(0, player.Rotation.Y, 0);

/\* FORMULA ZA OBRAČANJE DRUGEGA IGRALCA \*/

{

Z1 = computer.Position.X;

X1 = computer.Position.Z;

Z2 = player.Position.X;

X2 = player.Position.Z;

k = (Z2 - Z1) / (X2 - X1);

if (X2 > X1)

angle = (float)(-MathHelper.ToRadians(180) + Math.Atan(k));

else

angle = (float)Math.Atan(k);

computer.Rotation = new Vector3(0, angle, 0);

}

if (Mouse.GetState().LeftButton == ButtonState.Pressed)

punchP = true;

if (Mouse.GetState().RightButton == ButtonState.Pressed)

defendP = true;

if (Keyboard.GetState().IsKeyDown(Keys.Space))

punchSP = true;

/\* Vnaprej izračunani premiki igralca in nasprotnika \*/

Vector3 tmp\_front = player.Position;

{

Matrix rotation = Matrix.CreateFromYawPitchRoll(player.Rotation.Y,

player.Rotation.X, player.Rotation.Z);

tmp\_front += Vector3.Transform(Vector3.Forward, rotation) \*

(float)gameTime.ElapsedGameTime.TotalMilliseconds \* 1.8f;

}

Vector3 tmp\_back = player.Position;

{

Matrix rotation = Matrix.CreateFromYawPitchRoll(player.Rotation.Y,

player.Rotation.X, player.Rotation.Z);

tmp\_back += Vector3.Transform(Vector3.Backward, rotation) \*

(float)gameTime.ElapsedGameTime.TotalMilliseconds \* 1.8f;

}

Vector3 tmp2\_front = computer.Position;

{

Matrix rotation = Matrix.CreateFromYawPitchRoll(computer.Rotation.Y,

computer.Rotation.X, computer.Rotation.Z);

tmp2\_front += Vector3.Transform(Vector3.Forward, rotation) \*

(float)gameTime.ElapsedGameTime.TotalMilliseconds \* 1.8f;

}

Vector3 tmp2\_back = computer.Position;

{

Matrix rotation = Matrix.CreateFromYawPitchRoll(computer.Rotation.Y,

computer.Rotation.X, computer.Rotation.Z);

tmp2\_back += Vector3.Transform(Vector3.Backward, rotation) \*

(float)gameTime.ElapsedGameTime.TotalMilliseconds \* 1.8f;

}

computer\_head.Center = new Vector3(0, 20, 0) \* player.Scale +

computer.Position;

player\_head.Center = new Vector3(0, 20, 0) \* player.Scale + player.Position;

/\* ---------> UMETNA INTELIGENCA ZA NASPROTNIKA <---------\*/

rotChange\_two = Vector3.Zero;

if (Keyboard.GetState().IsKeyDown(Keys.Left))

rotChange\_two += new Vector3(0, 1, 0);

if (Keyboard.GetState().IsKeyDown(Keys.Right))

rotChange\_two += new Vector3(0, -1, 0);

computer.Rotation += rotChange\_two \* 0.06f;

/\* UGOTAVLJANJE, ali bo nasprotnik napadal, ali se bo umikal \*/

{

timer\_three += gameTime.ElapsedGameTime;

if (timer\_three > TimeSpan.FromSeconds(1))

{

walk\_state = 0;

timer\_three = TimeSpan.Zero;

}

}

/\* NASPROTNIK - hoja naprej \*/

if (walk\_state == 0 && Vector3.Distance(tmp\_front, tmp2\_front) > 500 &&

!attack)

{

animPlayer4.Play(5, 0, walkFC, computer, gameTime);

animPlayer4.Play(6, 1, walkFC, computer, gameTime);

animPlayer4.Play(7, 2, walkFC, computer, gameTime);

animPlayer4.Play(8, 3, walkFC, computer, gameTime);

animPlayer4.Play(9, 4, walkFC, computer, gameTime);

animPlayer4.Play(10, 5, walkFC, computer, gameTime);

if ((tmp2\_front.X < 1780) && (tmp2\_front.Z < 1780) && (tmp2\_front.X > -

1780) && (tmp2\_front.Z > -1780))

computer.Position = tmp2\_front;

}

/\* NASPROTNIK - hoja nazaj \*/

else if (walk\_state == 2 && Vector3.Distance(tmp\_back, tmp2\_back) > 500 &&

!attack)

{

animPlayer4.Play(5, 0, walkBC, computer, gameTime);

animPlayer4.Play(6, 1, walkBC, computer, gameTime);

animPlayer4.Play(7, 2, walkBC, computer, gameTime);

animPlayer4.Play(8, 3, walkBC, computer, gameTime);

animPlayer4.Play(9, 4, walkBC, computer, gameTime);

animPlayer4.Play(10, 5, walkBC, computer, gameTime);

if ((tmp2\_back.X < 1780) && (tmp2\_back.Z < 1780) && (tmp2\_back.X > -1780)

&& (tmp2\_back.Z > -1780))

computer.Position = tmp2\_back;

}

else

{

animPlayer4.ResetPlayer();

int st1 = 0;

foreach (string obj in Objects)

{

transformBones(computer, obj, transform[st1]);

st1++;

}

}

/\* UGOTAVLJANJE, ali bo nasprotnik udaril, ali se bo branil \*/

{

timer\_four += gameTime.ElapsedGameTime;

timer\_six += gameTime.ElapsedGameTime;

if (Vector3.Distance(tmp\_front, tmp2\_front) <= 500)

{

if (timer\_four > TimeSpan.FromSeconds(1))

{

punchC = true;

defendC = GetRandomBoolean();

timer\_four = TimeSpan.Zero;

}

if (timer\_six > TimeSpan.FromSeconds(2) && !punchC)

{

punchSC = true;

timer\_six = TimeSpan.Zero;

}

}

else

defendC = false;

}

/\* NASPROTNIK - Udarec z levo roko \*/

if (punchSC)

{

animPlayer8.Play(1, 0, special\_punchC, computer, gameTime);

animPlayer8.Play(2, 1, special\_punchC, computer, gameTime);

if (animPlayer8.clip == 1)

for (int i = 1; i < 3; i++)

{

float x = special\_left\_arm\_radius \*

(float)Math.Cos(computer.Rotation.Y - MathHelper.ToRadians(-

100));

float z = special\_left\_arm\_radius \* -

(float)Math.Sin(computer.Rotation.Y - MathHelper.ToRadians(-

100));

player\_punch\_sphereV = new Vector3(x, 21.2f \* i, z) \*

computer.Scale + computer.Position;

computer\_special\_punch\_sphere.Center = player\_punch\_sphereV;

if (computer\_special\_punch\_sphere.Intersects(player\_defend\_sphere)

&& defendP)

{

special\_punchC[0].resetAnimation();

special\_punchC[1].resetAnimation();

animPlayer8.ResetPlayer();

punchSC = false;

if (sound)

soundMissed.Play();

}

}

if (animPlayer8.endPlaying)

{

for (int i = 1; i < 3; i++)

{

float x = special\_left\_arm\_radius \*

(float)Math.Cos(computer.Rotation.Y - MathHelper.ToRadians(-

100));

float z = special\_left\_arm\_radius \* -

(float)Math.Sin(computer.Rotation.Y - MathHelper.ToRadians(-

100));

player\_punch\_sphereV = new Vector3(x, 21.2f \* i, z) \*

computer.Scale + computer.Position;

computer\_special\_punch\_sphere.Center = player\_punch\_sphereV;

if (computer\_special\_punch\_sphere.Intersects(player\_head))

{

collision2 = true;

timer\_two = gameTime.ElapsedGameTime;

}

}

animPlayer8.ResetPlayer();

punchSC = false;

}

}

if (punchC)

{

animPlayer5.Play(1, 0, punch\_leftC, computer, gameTime);

animPlayer5.Play(2, 1, punch\_leftC, computer, gameTime);

if (animPlayer5.clip == 2)

for (int i = 150; i > 85; i--)

{

float x = left\_arm\_radius \* (float)Math.Cos(computer.Rotation.Y –

MathHelper.ToRadians(-i));

float z = left\_arm\_radius \* -(float)Math.Sin(computer.Rotation.Y –

MathHelper.ToRadians(-i));

player\_punch\_sphereV = new Vector3(x, 21.2f, z) \* computer.Scale +

computer.Position;

computer\_punch\_sphere.Center = player\_punch\_sphereV;

if (computer\_punch\_sphere.Intersects(player\_defend\_sphere) &&

defendP)

{

punch\_leftC[0].resetAnimation();

punch\_leftC[1].resetAnimation();

animPlayer5.ResetPlayer();

punchC = false;

if(sound)

soundMissed.Play();

}

}

if (animPlayer5.endPlaying)

{

for (int i = 150; i > 85; i--)

{

float x = left\_arm\_radius \* (float)Math.Cos(computer.Rotation.Y –

MathHelper.ToRadians(-i));

float z = left\_arm\_radius \* -(float)Math.Sin(computer.Rotation.Y –

MathHelper.ToRadians(-i));

player\_punch\_sphereV = new Vector3(x, 21.2f, z) \* computer.Scale +

computer.Position;

computer\_punch\_sphere.Center = player\_punch\_sphereV;

if (computer\_punch\_sphere.Intersects(player\_head))

{

collision2 = true;

timer\_two = gameTime.ElapsedGameTime;

}

}

animPlayer5.ResetPlayer();

punchC = false;

}

}

/\* NASPROTNIK - Obramba z desno roko \*/

if (defendC)

{

animPlayer6.Play(3, 0, punch\_rightC, computer, gameTime);

animPlayer6.Play(4, 1, punch\_rightC, computer, gameTime);

if (animPlayer6.endPlaying)

{

if (defendC)

{

animPlayer6.freez = true;

float x = right\_arm\_radius \* (float)Math.Cos(computer.Rotation.Y –

MathHelper.ToRadians(-70));

float z = right\_arm\_radius \* -(float)Math.Sin(computer.Rotation.Y

- MathHelper.ToRadians(-70));

player\_punch\_sphereV = new Vector3(x, 21.2f, z) \* computer.Scale

+ computer.Position;

computer\_defend\_sphere.Center = player\_punch\_sphereV;

}

else

{

animPlayer6.freez = false;

defendC = false;

}

animPlayer6.ResetPlayer();

animPlayer6.Play(3, 0, punch\_rightC, computer, gameTime);

animPlayer6.Play(4, 1, punch\_rightC, computer, gameTime);

}

}

Premikanje glavnega igralca deluje skoraj na popolnoma enak način kot deluje umetna inteligenca nasprotnika. Naslednja programska koda preverja, ali je prišlo do trka ali ne (ali je bil nasprotnik uspešno udarjen.)

/\* Ali je bil robot uspešno udarjen ali je udarec ubranil \*/

{

if (collision1)

{

if(sound)

soundPunched.Play();

attack = true;

healthC -= rnd.Next(5, 11);

}

if (collision2)

{

if(sound)

soundPunched.Play();

defence = true;

healthP -= rnd.Next(5, 11);

}

if (attack)

{

if (timer\_one < gameTime.ElapsedGameTime + TimeSpan.FromSeconds(0.2f))

{

computer.Rotation = new Vector3((float)MathHelper.ToRadians(10),

computer.Rotation.Y, computer.Rotation.Z);

timer\_one += gameTime.ElapsedGameTime;

}

else

{

attack = false;

walk\_state = 2;

}

}

if (defence)

{

if (timer\_two < gameTime.ElapsedGameTime + TimeSpan.FromSeconds(0.2f))

{

player.Rotation = new Vector3((float)MathHelper.ToRadians(10),

player.Rotation.Y, player.Rotation.Z);

timer\_two += gameTime.ElapsedGameTime;

}

else

defence = false;

}

}

updateCamera1(gameTime);

updateCamera2(gameTime);

}

}

else

updateMouse();

updateCamera\_menu(gameTime);

// Posodobi čas

base.Update(gameTime);

}

Poskrbeti sem moral tudi za primere, ko bi uporabnik želel večkrat igrati, pa nebi želel končati igrico. V tem primeru sem moral ponovno nastaviti življenja igralcev na 100 in postaviti čas igranja na vrednost 0. Postaviti sem moral tudi oba igralca na začetne pozicije. Funkcija, ki skrbi za ponovno nastavitev spremenljivk sem poimenoval **resetForNewGame**.

void resetForNewGame()

{

healthP = 100;

healthC = 100;

punchP = false;

defendP = false;

punchC = false;

defendC = false;

change\_camera = false;

attack = false;

defence = false;

name = null;

time = 0;

walk\_state = 4;

timer\_one = TimeSpan.Zero;

timer\_two = TimeSpan.Zero;

timer\_three = TimeSpan.Zero;

timer\_four = TimeSpan.Zero;

timer\_five = TimeSpan.Zero;

timer\_six = TimeSpan.Zero;

rotChange\_one = Vector3.Zero;

rotChange\_two = Vector3.Zero;

this.IsMouseVisible = false;

}

Ostali sta še dve funkciji, ki igrata pomembno vlogo pri predvajanju animiranih likov. Funkciji se imenujeta applyObjectDefaults in transformBones. Prvo funkcijo sem uporabil za določanje začetek animacije. Ta del animacije je v igri prisoten, tudi če igralec ne pritisne nobene izmed tipk.

Druga funkcija je bila narejena predvsem zaradi optimizacije in boljšega pregleda nad programsko kodo. Ta funkcija namreč skrbi za premikanje posameznih modelov.

ObjectAnimation applyObjectDefaults(Vector3 startPosition, Vector3 endPosition, Vector3 startRotation, Vector3 endRotation, float duration, bool loop)

{

ObjectAnimation temp;

temp = new ObjectAnimation(startPosition, endPosition, startRotation, endRotation, TimeSpan.FromSeconds(duration), loop);

return temp;

}

public void transformBones(CModel robot, string Object, ObjectAnimation anim)

{

robot.Model.Meshes[Object].ParentBone.Transform =

Matrix.CreateRotationY(anim.Rotation.Y) \*

Matrix.CreateRotationX(anim.Rotation.X) \*

Matrix.CreateRotationZ(anim.Rotation.Z) \*

Matrix.CreateTranslation(anim.Position);

}

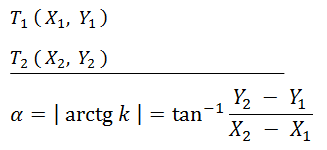
# Problemi med izdelovanjem igrice

## Matematični problemi

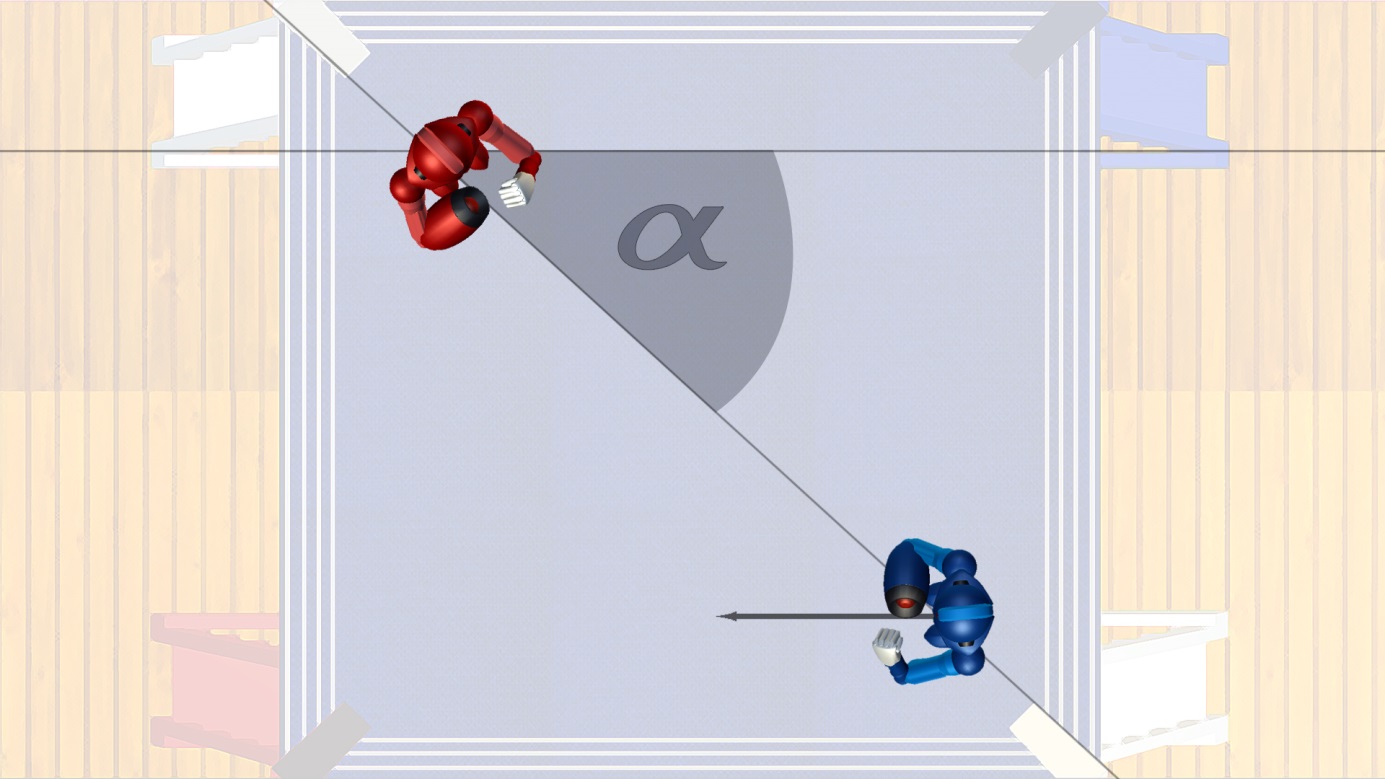
Ob pisanju programske kode sem naletel na težavo. Nisem znal izračunati, za koliko se mora nasprotnik zavrteti, da bo ves čas obrnjen proti igralcu. Pri račnanju kota z uporabo pitagorovega izreka je računanje mnogo težje, saj sta dve točki (T1 in T2) lahko v različnih kvadrantih.

Ker nisem imel nobene ideje, kako drugače izračunati kot, sem za pomoč prosil profesorico za matematiko ga. Eriko Plešo. Povedala mi je, da lahko kot izračunam tudi z uporabo tangente. Vse, kar potrebujem, sta točki T1 (pozicija igralca) in T2 (pozicija nasprotnika). S pomočjo teh dveh točk pa lahko izračunam tangento k oziroma kot med tema dvema točkama.

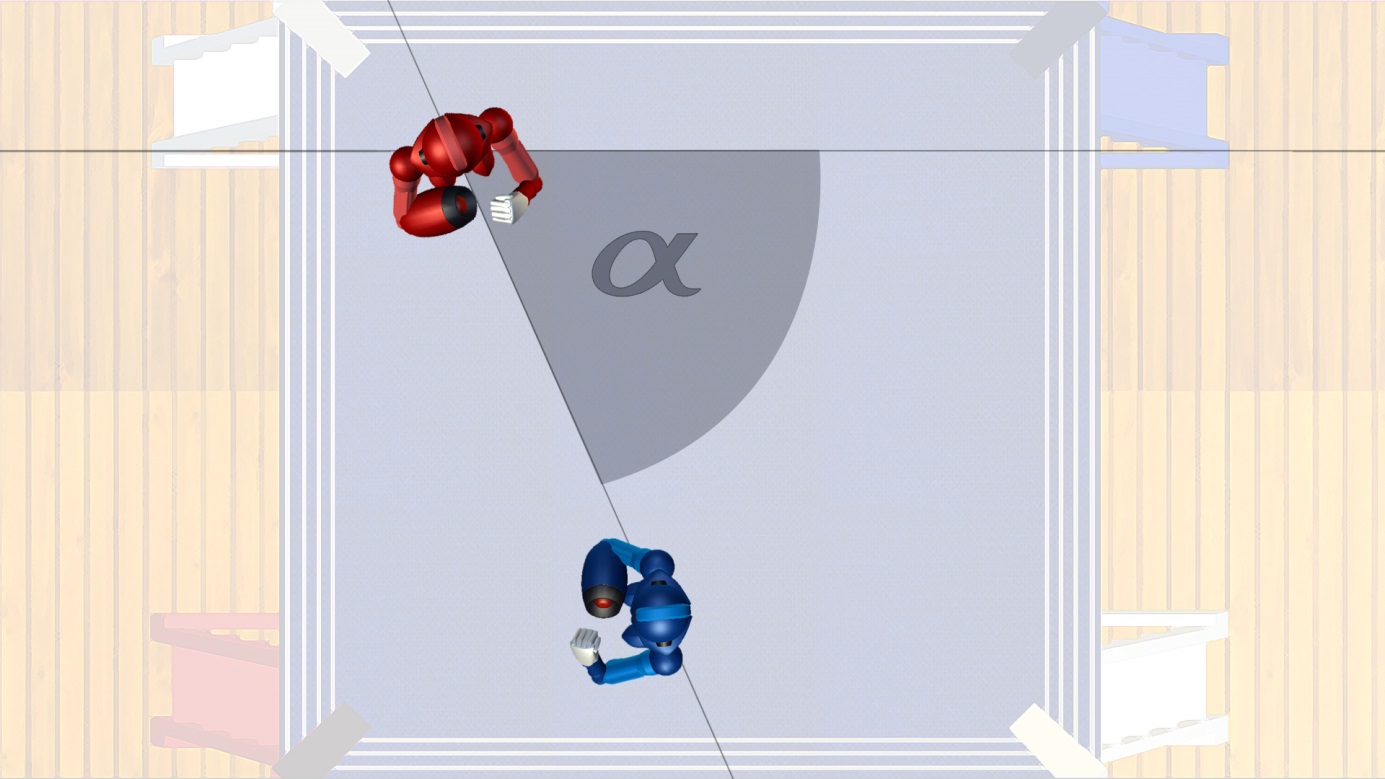
Rešitev za moj problem je naslednja enačba:



Slika 9: Velikost kota v začetni poziciji igralca



Slika 10: Velikost kota v premaknjeni poziciji igralca



## Grafični problemi

Poleg zapletov z matematičnimi formulami, pa so se pri izdelovanju igrice pojavljale težave tudi pri izrisovanju 3D objektov. Po dolgem iskanju rešitve, sem naposled le našel spletno stran, kjer so imeli podobne težave kot jaz. Njihova rešitev so bile 3 funkcije, ki skrbijo, da se grafična kartica ob vsakem izrisovanju vrne na začetno pozicijo. Ker so se zgoraj omenjene funkcije potrdile za pravilno rešitev tudi pri meni sem se odločil, da jih bom obdržal v programski kodi. Vir (URL: http://stackoverflow.com/questions/13342400/fbx-model-not-displaying-correctly-in-xna-4-0).

Rešitev za moj problem so bile naslednje 3 funkcije:

public void Clear()

{

GraphicsDevice.Clear(Color.CornflowerBlue);

}

public void ResetGraphic()

{

GraphicsDevice.BlendState = BlendState.AlphaBlend;

GraphicsDevice.DepthStencilState = DepthStencilState.None;

GraphicsDevice.RasterizerState = RasterizerState.CullCounterClockwise;

GraphicsDevice.SamplerStates[0] = SamplerState.AnisotropicWrap;

}

public void BeginRender3D()

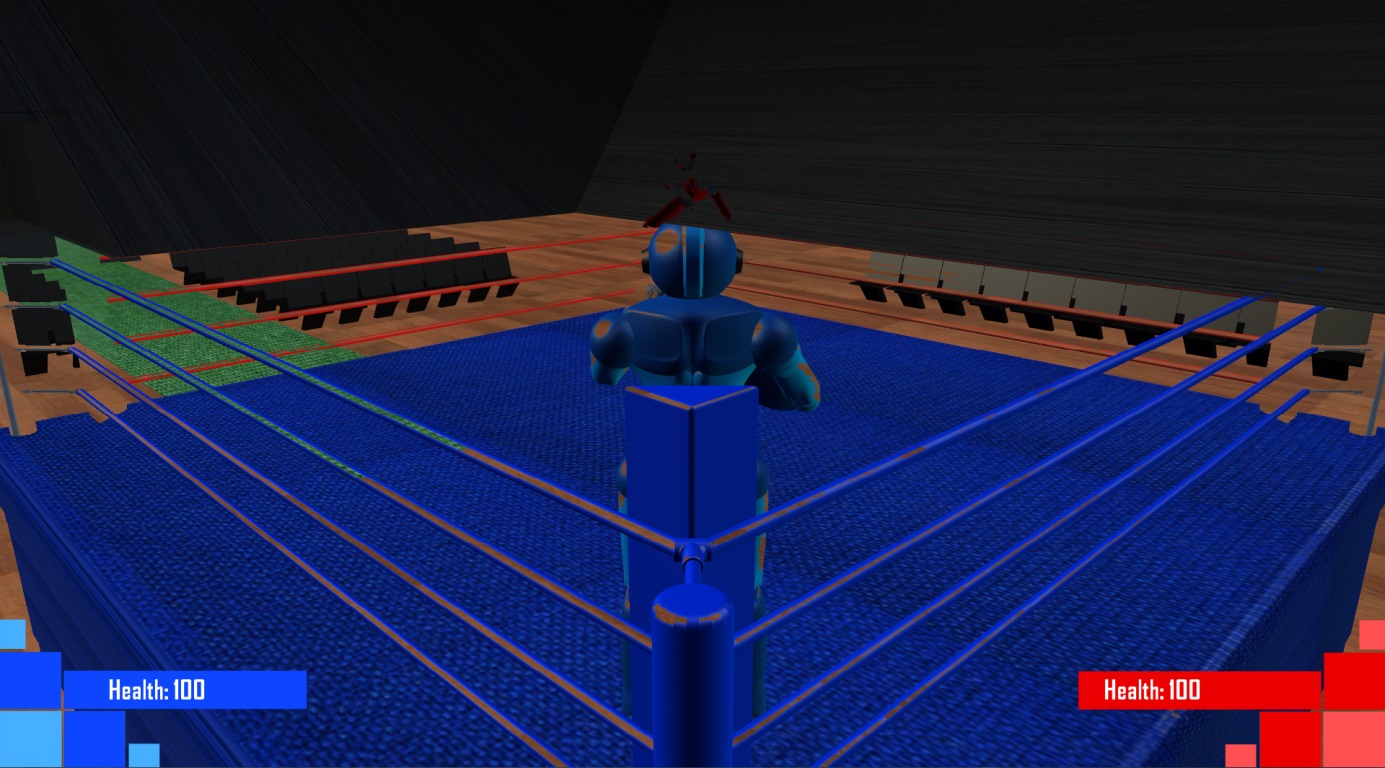
{

GraphicsDevice.BlendState = BlendState.Opaque;

GraphicsDevice.DepthStencilState = DepthStencilState.Default;

}

Slika 11: Problemi pri izrisovanju 3D objektov



## Problemi pri animaciji objekta

Ko sem že mislil, da bo problemov konec, pa sem naletel še na eno težavo. Sedaj ni bil ne matematičen, ne grafičen problem, temveč problem z predvajanjem animacij. Pri sestavljanju animacij namreč nisem bil pozoren na različne vrste animacij. Te animacije se delijo predvsem na sestavo animiranega objekta. Objekt lahko namreč vsebuje okostje (angl. *bones*) in je sestavljen z enega modela ali pa je objekt sestavljen iz več modelov (angl. *objects*).

V mojem primeru je bil objekt sestavljen iz več modelov. To je pomenilo, da bo potrebno premikati vsak model posamezno. Po pisanju programske kode sem opazil, da bo vpisovanje pozicije in obračanje posameznega modela, le potrata časa in prostora. Odločil sem se, da bom podatke bral iz XML datoteke, saj bom s tem izboljšal preglednost programa ter zmanjšal število vrstic v programu.

Rešitev v tem primeru je bila XML datoteka:

<? xml version="1.0" ?>

<!-- ANIMACIJA ZA IGRICO -->

<frame>

<!-- left\_arm -->

<default> 1 </default>

<!-- obračanje okoli x osi -->

<rotx> 0 </rotx>

<!-- obračanje okoli y osi -->

<roty> 0 </roty>

<!-- obračanje okoli z osi -->

<rotz> 0 </rotz>

<!-- čas, ki mora preteči od začetka anmacije, do določenega okvirja -->

<time> 0 </time>

<!-- id, ki določa posamezen model -->

<listid> 1 </listid>

<default> 0 </default>

<rotx> 10 </rotx>

<roty> 140 </roty>

<rotz> 0 </rotz>

<time> 0.2 </time>

<listid> 1 </listid>

<default> 0 </default>

<rotx> 10 </rotx>

<roty> 140 </roty>

<rotz> 0 </rotz>

<time> 0.25 </time>

<listid> 1 </listid>

# Zaključek

Izdelovanje računalniške igrice Spopad titanov, je bila po mojih izkušnjah zahtevna naloga. Razlog za to je, da je to moja prva osebna izkušnja izdelovanja 3D-igrice. Na začetku izdelovanja sem veliko časa potreboval za učenje programskega jezika C# in grafičnega vmesnika XNA. Ko sem se naučil najpomembnejših osnov, sem šele nato lahko začel z programiranjem igrice. Velikokrat sem zašel iz svojih načrtov in šele nato ugotovil, da se moram vrniti nazaj. V večini primerov je to pomenilo brisanje celotne programske kode. Ko pa sem ponovno začel z pisanjem programske kode, sem vsakič znova prišel korak bližje koncu. Če bi imel to izkušnjo, kot jo imam sedaj z programiranjem 3D-igrice, na začetku leta, bi verjetno celotno nalogo naredil na drugačen način. Namesto, da bi le pisal programsko kodo, bi več časa posvetil načrtovanju.

Le v tem primeru bi imel natančno določene cilje, ki bi jih želel doseči.

# Kazalo slik

[Slika 1: Microsoft Visual Studio 2010 2](#_Toc353996316)

[Slika 2: Autodesk Maya 2013 3](#_Toc353996317)

[Slika 3: Adobe Photoshop CS4 4](#_Toc353996318)

[Slika 4: Struktura programa 6](#_Toc353996319)

[Slika 5: Hoja, obramba, napad in posebni udarec 7](#_Toc353996320)

[Slika 6: Prikaz obeh igralniških robotov 8](#_Toc353996321)

[Slika 7: Oblikovanje glavnega menija 9](#_Toc353996322)

[Slika 8: Izdelovanje 3D objektov 20](#_Toc353996323)

[Slika 9: Velikost kota v začetni poziciji igralca 34](#_Toc353996324)

[Slika 10: Velikost kota v premaknjeni poziciji igralca 34](#_Toc353996325)

[Slika 11: Problemi pri izrisovanju 3D objektov 35](#_Toc353996326)

[Slika 12: Začetni meni 39](#_Toc353996327)

[Slika 13: Meni z nastavitvami 39](#_Toc353996328)

[Slika 14: Meni z informacijami 40](#_Toc353996329)

[Slika 15: Meni za novo igro 40](#_Toc353996330)

[Slika 16: Meni z rezultati 41](#_Toc353996331)

# Viri in literatura

* Aaron, Reed. 2011. ***Learning XNA 4.0*** O'Reilly media
* C# Corner: ***How to use SpriteFont in XNA*** Dostopno prek: http://www.c-sharpcorner.com/uploadfile/iersoy/how-to-use-spritefont-in-xna/ (1. 3. 2013)
* Chad, Carter. 2008. ***Microsoft XNA Unleashed***Sams Publishing
* Eric, Keller. 2010. ***Mastering Autodesk Maya 2011*** Wiley Publishing
* Jeff Ferguson, Brian Patterson, Jason Beres, Pierre Boutquin in Meeta Gupta. 2002. ***C# Bible*** Wiley Publishing
* GameDev.net: ***XNA animation without using skinned model example?*** Dostopno prek: http://www.gamedev.net/topic/640370-xna-animation-without-using-skinned-model-example/ **(10. 4. 2013)**
* James, Sean. 2010. ***3D Graphics with XNA Game Studio 4.0*** Packt publishing
* Kurt, Jaegers. 2012. ***XNA 4 3D Game Development by Example Beginner's Guide***Packt publishing
* Luke, Drumm. 2012. ***Microsoft XNA 4.0 Game Development Cookbook*** Packt publishing
* Miles, Rob. 2011. ***Microsoft XNA Game Studio 4.0: Learn Programming Now!*** Microsoft Press
* Stackoverflow: ***FBX Model not displaying correctly in XNA 4.0*** Dostopno prek: http://stackoverflow.com/questions/13342400/fbx-model-not-displaying-correctly-in-xna-4-0 **(14. 4. 2013)**
* Sharky's Blog: ***XNA Collision Detection for 3D models*** Dostopno prek: http://sharky.bluecog.co.nz/?p=108 **(5. 4. 2013)**

# Priloge

## Slike menijev

Slika 12: Začetni meni



Slika 13: Meni z nastavitvami



Slika 14: Meni z informacijami



Slika 15: Meni za novo igro



Slika 16: Meni z rezultati



## Namestitveni CD za igrico Spopad titanov

*(CD je priložen v posebnem ovitku)*