Projekt Šibenik
Tehnička dokumentacija
Verzija <1.0 >

Studentski tim: Teon Banek
Damir Ciganović Janković
Marko Đomlija
Ivan Hakštok
Ana Marija Komar
Tomislav Ljubej
Niko Mikuličić
Nikolina Očić

Nastavnik: Prof. dr. sc. Željka Mihajlović

Sadržaj

Tehnička dokumentacija

1. Opis razvijenog proizvoda

Rezultat ovog projekta je program, odnosno računalna igra u kojoj se glavni lik kreće po modelu grada Šibenika i omogućena mu je određena interaktivnost s okolinom. Igra se prikazuje u 3 dimenzije i ima intuitivno sučelje.

Osim hodanja, trčanja i skakanja po gradu glavni lik rješava podigre. Kretanjem po gradu saznaje činjenice o gradu te ima zadatak pronaći ružu koja će mu pomoći da završi svoju pustolovinu u gradu.

Cilj projekta je da se igrač kroz igru, uz to što sazna neke činjenice o gradu, zainteresira za to mjesto te odluči jednog dana zaista posjetiti Šibenik.

Za razvoj igre korišten je alat za izradu igara *Unity*, program za 3D modeliranje, animiranje, uređivanje, kreiranje i pregledavanje raznih grafičkih i video datoteka *Blender,* jednostavan program za izradu 3D objekata *Google SketchUp,* besplatan program za generiranje ljudskih formi i oblika *MakeHuman™* te besplatni 3D modeli sa stranica navedenih u literaturi.

2. Tehničke značajke

Rad na projektu podijeljen je na dva osnovna dijela programiranje i modeliranje. Programski kod je napisan u jeziku C# koristeći skripte *Unity*-a. Bilo je potrebno implementirati kretanje kamere, kretanje glavnog lika te korisničko sučelje.

Modeli su napravljeni u *Blenderu*. Model grada je bilo dosta zahtjevno napraviti kako bi odgovarao stvarnom izgledu grada, a ostali modeli koji su bili potrebni većinom su preuzeti iz programskih alata s kojima smo radili ili sa stranica sa 3D modelima, dok je glavni lik napravljen pomoću programa *MakeHuman™*.

2.1 Modeliranje

Prije samog procesa modeliranja potrebno je odrediti važne objekte. Kako se igra odvija u gradu Šibeniku, jedan od najbitnijih modela je model grada. Sljedeći bitan model je model glavnog lika. Također kao bitniji modeli javljaju se katedrala Sv. Jakova[1] te tvrđava Sv. Nikole[1].

2.1.1 Model glavnog lika

Model glavnog lika računalne igre napravljen je pomoću besplatnog alata *MakeHuman™* kojemu je osnovna funkcija generirati 3D model čovjeka. Pri izradi čovjeka u *MakeHuman*-u, moguće je svom modelu osim boje očiju, kose i građe (spol, starost, muskulatura), izabrati i neke od ponuđenih modela odjeće te druge mogućnosti.

[1]Zbog kratkog roka za izradu projekta, korišteni su gotovi modeli sa stranica:

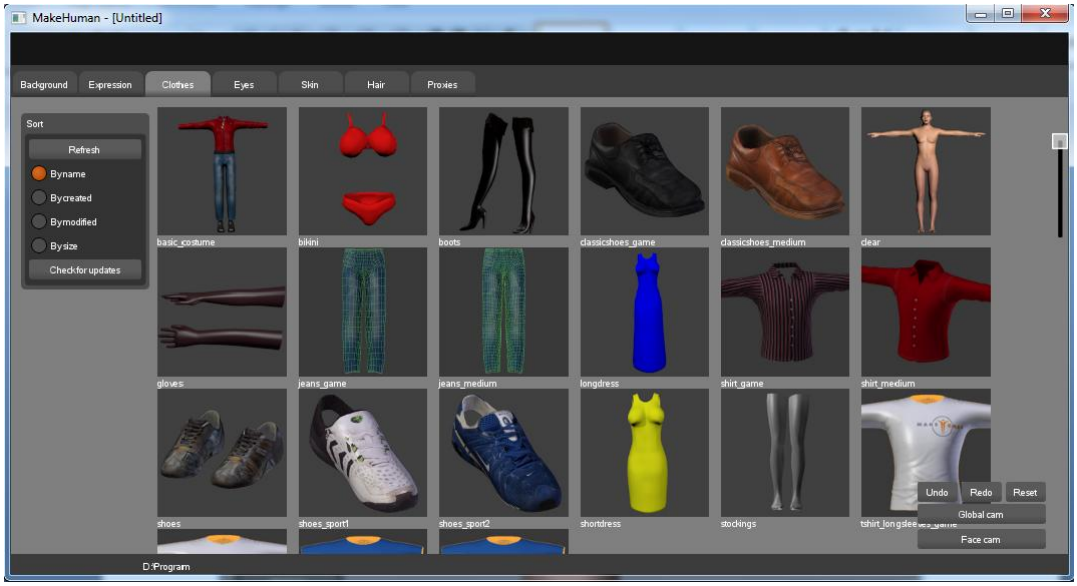
www.exchange3d.com

3dexport.com

archive3d.net

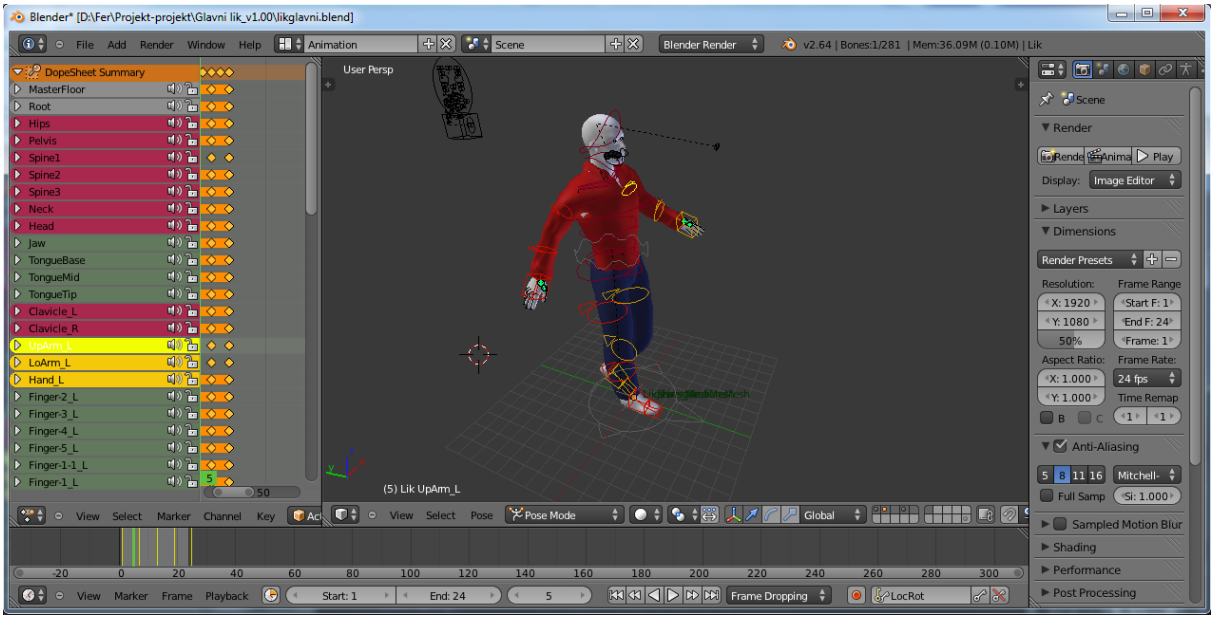
Slika 2.1 Osnovne postavke pri izradi 3D modela čovjeka

Svojstvo alata da može izvesti (*export*) modele u *.obj* ili *.mhx* formatu bilo je korisno pošto smo za modeliranje i dodavanje animacija našim modelima koristili program *Blender* koji takve formate može učitati. Format koji smo koristili je *.mhx* jer uz sam model čovjeka taj format sadrži i odgovarajuću armaturu, odnosno kostur koji podržava naš model i koji nam uvelike koristi pri izradi animacija.



Slika 2.2 Odabir odjeće i obuće

Program *Blender* ima mnoštvo funkcija i mogućnosti za rad s 3D modelima, no kako bi dovršili glavni lik, koristili smo ga samo pri izradi animacija jer smo model i armaturu učitali iz *MakeHuman*™ alata, kao što je već rečeno. Proces izrade animacija modela svodi se na postavljanje modela u određeni položaj te spremanje svakog položaja posebno kako bi dobili tečnu animaciju. Svaka spremljena slika (*položaj*) određuje jedan okvir (*frame*). Pogodnost *Blender* alata je u tome što je dovoljno postaviti lik u dva različita položaja na dva nesusjedna okvira i alat će sam nadopuniti ostale okvire odgovarajućim pokretima.



Slika 2.3 Izrada animacija glavnog lika

Model glavnog lika zajedno s njegovim animacijama ubacili smo u program *Unity* tako što smo model izvezli (*export*) iz Blendera u *.fbx* formatu koji *Unity* može pročitati.

Neki problemi:

Kada se model čovjeka u *.mhx* formatu učita u program *Blender*, on ga neće prikazivati na isti način kao *MakeHuman*™ alat, odnosno prikazivati će ga bez tekstura, ili s neodgovarajućim teksturama. No, ukoliko pokrenemo funkciju *Render*, lik će se pojaviti u osnovnom obliku. Ipak, problem nastaje pri uvozu formata *.fbx* u *Unity* jer se teksture ne prenesu s modelom. Stoga je potrebno napraviti direktorij *Textures* koji *Unity* prepoznaje te u njega ubaciti sve korištene teksture te zatim prilijepiti pojedinu teksturu odgovarajućem dijelu tijela ili odjeće.

Kroz neke dijelove odjeće se može uočiti model kože lika, iako tu ne bi trebao biti. Odjeća lika ili nije prikazana u cijelosti ili model kože prolazi kroz odjeću prilikom animacije. Odjeća se može nadopuniti u *Sculpt mode* načinu rada u *Blenderu*, no to još uvijek ne rješava problem probijanja kroz odjeću pri animaciji lika.

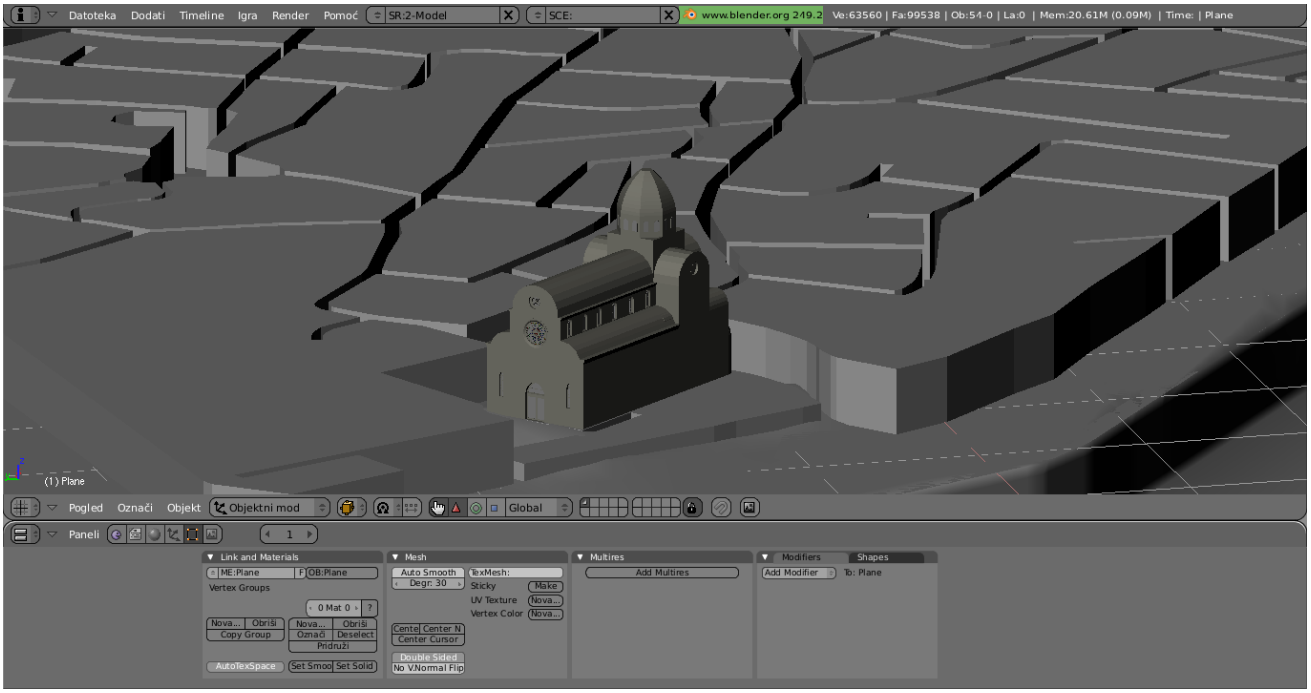
2.1.2 Model grada

Model grada Šibenika napravljen je pomoću alata *Google SketchUp* i *Blender*. Najprije smo učitali satelitski prikaz dijela grada koji smo modelirali u *SketchUp*. Na tako uvezenoj fotografiji terena moguće je dodavati linije, što smo iskoristili da modeliramo tlocrt ulica i građevina.

Slika 2.4 Tlocrt modela grada u programu *SketchUp*

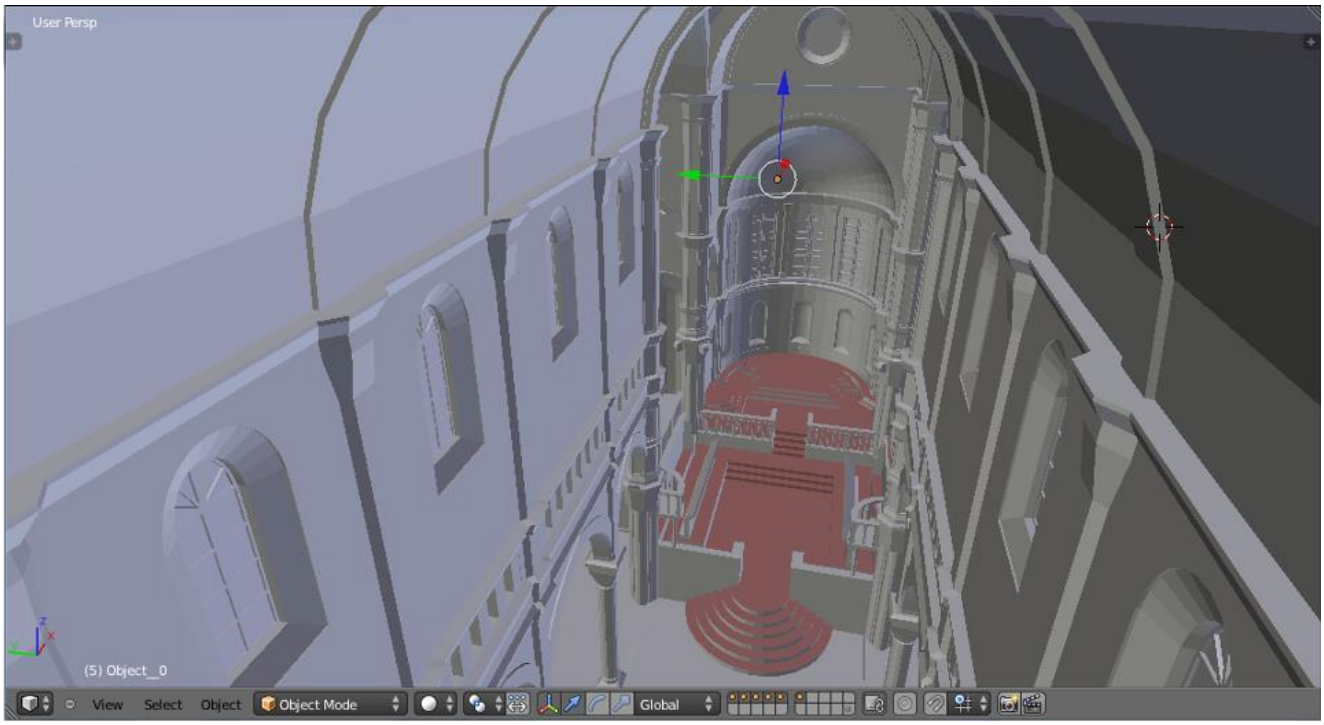
Nakon što smo završili tlocrt modela, taj isti model trebalo je uvesti u program *Blender* koji je pogodniji za detaljnije modeliranje. Problem je bio što *SketchUp* nema izravan izvoz u formate koje podržava *Blender*. Međutim, *SketchUp* ima mogućnost izvoza u datoteku *.kmz* formatu, koju je pomoću programa *WinRar* moguće raspakirati u neki direktorij, a među tako nastalim datotekama nalazi se i datoteka formata *.dae*, koju je moguće uvesti u *Blender*.

Unutar programa Blender smo od tlocrta napravili građevine. Modelirani su samo zidovi građevina, jer zbog površine dijela grada na kojem se igra odvija i broja građevina koje se na njoj nalaze, detaljno modeliranje svake zgrade bi nam oduzelo previše vremena. Na kraju je u model grada uvezen i model Katedrale svetog Jakova.

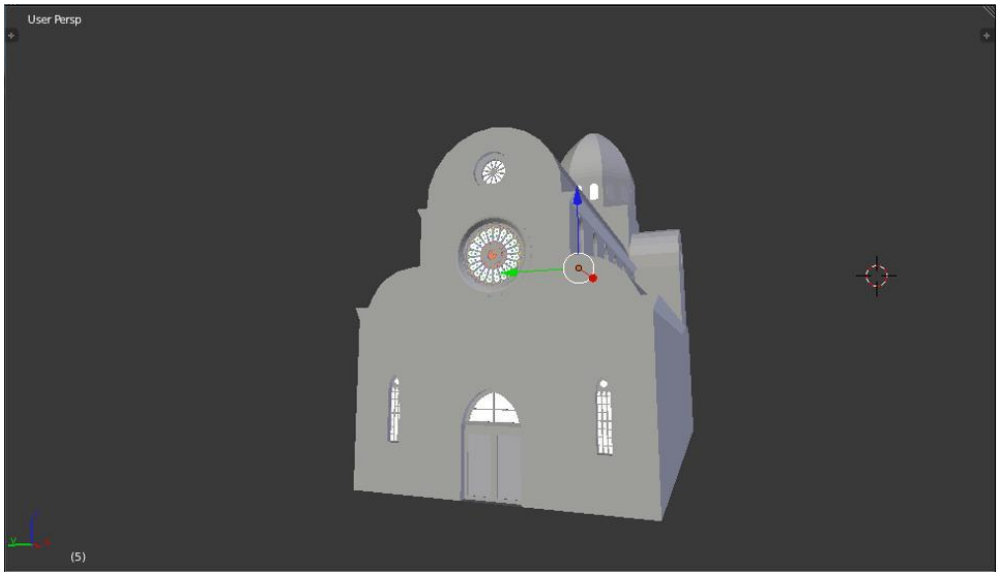


Slika 2.5 Model grada s modelom Katedrale sv. Jakova

Budući da je za dio igre bitna i unutrašnjost katedrale preuzeli smo i taj model.



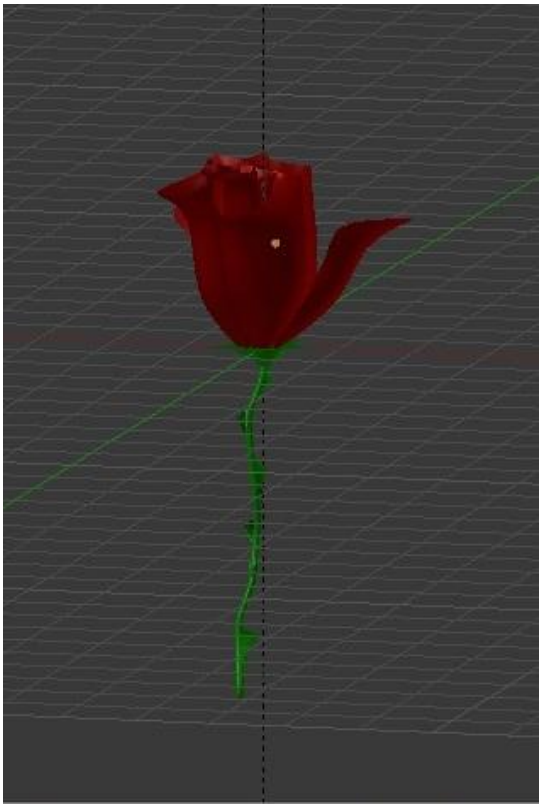
Slika 2.6 Prikaz modela unutrašnjosti Katedrale svetog Jakova



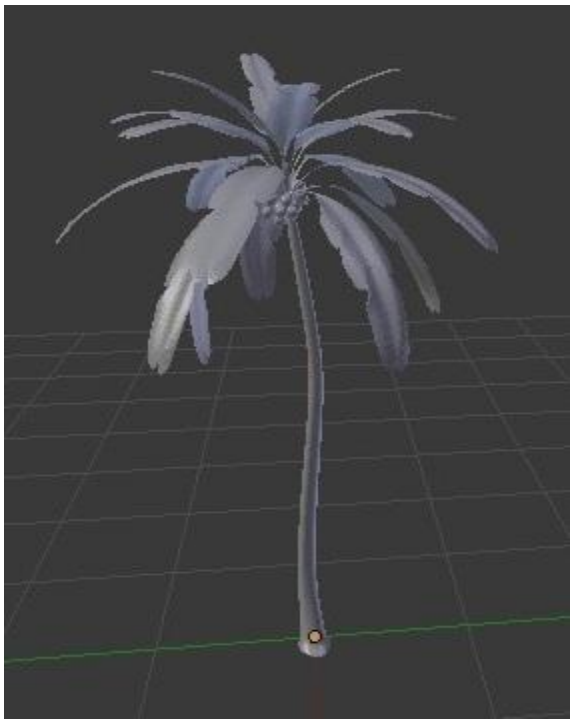
Slika 2.7 Model Katedrale svetog Jakova

2.1.3 Ostali modeli

Kao sporedni modeli javljaju se prateći objekti kao što su ulična rasvjeta[1], klupe u parkovima[1], razne ukrasne biljke te objekti važni za pojedine faze igre kao škrinja s blagom[1], ruža[1] te čamac s veslima[1].

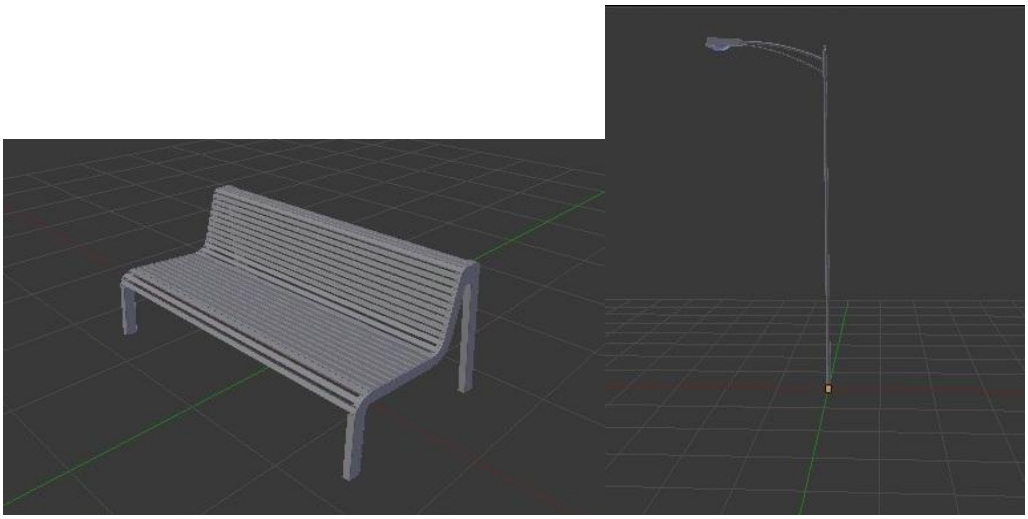


Slika 2.8 Model ruže



Slika 2.9 Model palme

Kako bi uljepšali prostor kojim se lik kreće upotpunili smo teren pratećim objektima.



Slika 2.10 Modeli klupe i ulične rasvjete

2.2 Kamera

Kamera koja prati glavnog lika je iz trećeg lica. Položaj kamere je kontroliran mišem te je omogućeno potpuno horizontalno rotiranje oko lika, dok je vertikalno rotiranje ograničeno. Ovakav način izvedbe je popularan u mnogim video igrama na tržištu.
Implementacija kamere je sadržana u razredu *MouseAimCamera* te sadrži javno dostupne vrijednosti:

```
public Transform target;
public float distance = 10;
public float xSpeed = 250;
public float ySpeed = 120;
public float yMinLimit = -20;
public float yMaxLimit = 80;
```

Objekt kojeg želimo pratiti referencira varijabla *target*. *Distance* sadrži željenu udaljenost kamere od objekta. Brzinu rotiranja određuju *xSpeed* i *ySpeed*. Ograničenje kuta u stupnjevima je zapisano u *yMinLimit* i *yMaxLimit*.

Jedan od problema upravljanja kamerom su prepreke između objekta kojeg promatramo i same kamere. Rotiranjem kamere iza nekog zida gubimo pogled na objekt. Jedno od mogućih i ovdje implementiranih rješenja je približavanje kamere do promatranog objekta kako između ne bi bilo prepreka.

Glavna funkcija ovako implementirane kamere je:

```
void LateUpdate()
{
    if (target)
    {
        x += Input.GetAxis("Camera X") * xSpeed * 0.02f;
        y -= Input.GetAxis("Camera Y") * ySpeed * 0.02f;
        y = ClampAngle(y, yMinLimit, yMaxLimit);
        Quaternion rotation = Quaternion.Euler(y, x, 0);
        float desiredDistance = distance;
        Vector3 desiredPosition = rotation * (new Vector3(0, 0, -desiredDistance))
                                    + target.position;
        int layerMask = 1;
        RaycastHit hit;
        if (Physics.Linecast(target.position, desiredPosition, out hit, layerMask))
        {
            desiredDistance = 0.8f * hit.distance;
        }
        Vector3 position = rotation * (new Vector3(0, 0, -desiredDistance))
                             + target.position;
        transform.rotation = rotation;
        transform.position = position;
    }
}
```

Horizontalni i vertikalni kut se izračunaju kretanjem korisničkih kontrola te pomnože brzinom kretanja kamere.

Funkcija *ClampAngle* provjerava nalazi li se novi vertikalni kut u granicama koje su zadane. Vektorski se izračuna nova željena pozicija kamere, ali prije samog pomaka kamere moramo provjeriti nalazi li se nešto između nje i promatranog objekta.

Funkcijom *Physics.Linecast* povlačimo liniju između kamere i objekta te spremamo pogođene prepreke u varijablu *hit*. Udaljenost smanjimo na dio udaljenosti od objekta do prve prepreke ako postoji.
Sada možemo postaviti kameru na dobru poziciju pridruživanjem vrijednosti *rotation* i *position* objektu *transform* same kamere.

2.3 Glavni lik

Glavni lik je centralan u interakciji s video igrom te mu je omogućeno više načina kretanja. Hodanje u svim smjerovima moguće tipkovnicom, kao i trčanje te skakanje.

Implementacija se nalazi u razredu *PlayerController*, a sadrži i ove pretpostavljene vrijednosti:

```
public float walkSpeed = 3;
public float runSpeed = 6;
public float gravity = 20;
public bool canJump = true;
```

Brzine hodanja i trčanja određuju varijable *walkSpeed* i *runSpeed*, a utjecaj gravitacije na glavni lik vrijednost *gravity*. Zastavica *canJump* služi kao oznaka može li glavni lik skočiti.
Funkcija *Update* koja se poziva u svakoj sličici izvođenja se nalazi ovdje:

```
void Update () {
    if (Input.GetButtonDown("Jump"))
    {
        lastJumpButtonTime = Time.time;
    }

    UpdateSmoothedMovementDirection();

    ApplyGravity();

    ApplyJumping();

    // Calculate actual motion.
    Vector3 movement = moveDirection * moveSpeed + new Vector3(0, verticalSpeed, 0);
    movement *= Time.deltaTime;

    // Move the controller.
    CharacterController controller =
(CharacterController)GetComponent("CharacterController");
    collisionFlags = controller.Move(movement);

    // Set rotation to move direction.
    transform.rotation = Quaternion.LookRotation(moveDirection);

    // Landed
    if (isGrounded() && jumping)
    {
        jumping = false;
    }
}
```

Provjerom da li je pritisnuta tipka za skok postavlja se trenutno vrijeme skoka, zatim se poziva metoda *UpdateSmoothedMovementDirection*. U njoj se izračunaju brzine kretanja ovisno o tome koje tipke su pritisnute.

Zatim se pozivaju metode *ApplyGravity* i *ApplyJumping* koje mijenjaju vertikalnu brzinu glavnog lika ovisno o gravitaciji i o tome da li je glavni lik skočio.

Nakon toga se računa konačan pomak lika i zapisuje u vektor *movement*. Izvršenje pomaka je ostvareno pozivom *controller.Move* metode.

Na kraju se provjerava da li je glavni lik sletio, ako je bio u skoku, te se postavlja zastavica *jumping* na vrijednost 'false' kako bi se označio kraj skoka.

Sva kretanja lika je potrebno povezati sa njegovim animacijama. Kontrola animacija je implementirana u razredu *PlayerAnimation*. Taj razred sadrži vlastitu *Update* metodu:

```
void Update () {
    PlayerController playerController = (PlayerController)
GetComponent("PlayerController");
    float currentSpeed = playerController.getSpeed();

    // Fade in run
    if (currentSpeed > playerController.walkSpeed + 0.1f)
    {
        animation.CrossFade("Run");
    }
    // Fade in walk
    else if (currentSpeed > 0.1)
    {
        animation.CrossFade("Walk_001");
    }
    else
    {
        animation.Blend("Walk_001", 0.0f, 0.1f);
        animation.Blend("Run", 0.0f, 0.1f);
        animation.Blend("Run", 0.0f, 0.1f);
    }

    animation["Run"].normalizedSpeed = runSpeedScale;
    animation["Walk_001"].normalizedSpeed = walkSpeedScale;

    if (playerController.isJumping())
    {
        animation.CrossFade("Jumpair", 0.2f);
    }
    else
    {
        animation.Stop("Jumpair");
    }
}
```

Status glavnog lika je zapisan u primjeru, u razredu *PlayerController* čija se referenca sprema u *playerController*. Zatim se izvršavaju provjere o načinu kretanja glavnog lika te pokreću pripadajuće animacije metodama *animation.CrossFade* i *animation.Blend* koje uzimaju kao argument nazive animacije koju treba pokrenuti.

Skok je poseban način kretanja te se on provjerava na kraju gdje se pokreće ili zaustavlja animacija skoka. Ovime je implementirano osnovno kretanje lika, a posebne interakcije s okolišem se mogu ostvariti nadopunjavanjem postojećih razreda ili dodavanjem novih.

2.4 Programiranje korisničkog sučelja

Korisničko sučelje u programskom okruženju *Unity* se ostvaruje implementacijom funkcije `void OnGUI()` u skripti koja je pridružena nekom objektu igre. Funkcija `void OnGUI()` se izvršava svaki okvir. U toj funkciji definiramo položaj i funkcionalnost različitih elemenata sučelja. Na primjer, gumb na ekranu deklariramo te implementiramo što se dogodi kada se klikne na njega na sljedeći način:

```
if (GUI.Button(new Rect(200,200,200,60),"Pokreni igru")) {
    Application.LoadLevel(1);
}
```

Funkcija *GUI.Button* stvara novi gumb i vraća 'true' kada je gumb pritisnut. Ako je vrijednost te funkcije 'true' izvršava se blok naredbi između '{' i '}'. U ovom slučaju pritiskom na gumb „Pokreni igru“ pokreće se scena u kojoj je smještena igra.

Uz gumbe, na ekranu možemo prikazivati i labele koje mogu sadržavati tekst i slike. Također, mogu se kreirati i okviri koji služe vizualnom grupiranju elemenata sučelja ili jednostavno kao podloga na kojoj će se iscrtavati ostali elementi. Način kreiranja labela i okvira je sljedeći:

```
GUI.Label(new Rect(0,0,Screen.width,Screen.height), background);

GUI.Box(new Rect(150,150,350,250),"Projekt Sibenik");
```

Funkcijom *GUI.Label()* se kreira nova labela, a funkcijom *GUI.Box()* novi okvir. Kao argumente primaju instancu klase *Rect* koja služi za definiciju položaja i veličine komponente, a drugi argument je sadržaj.

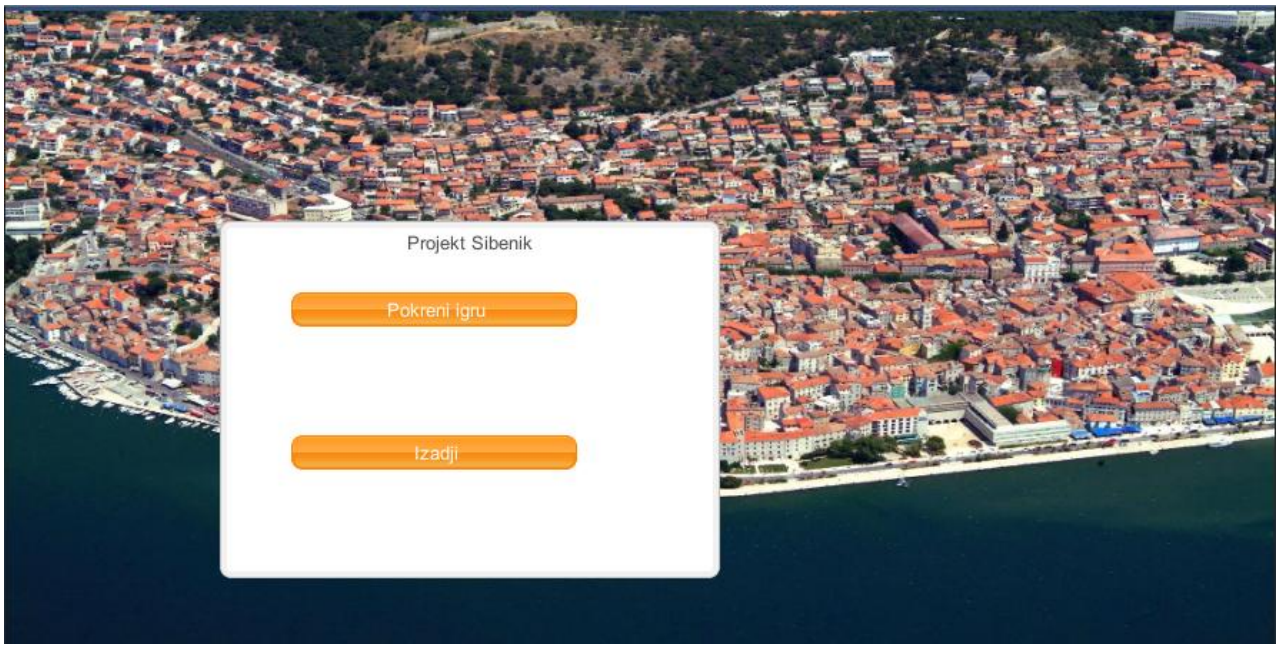
Valja primijetiti da je u ovom slučaju sadržaj labele varijabla *background* koja je tipa *Texture2D* što je zapravo slika.

Sam izgled komponenti sučelja se definira u posebnoj datoteci ekstenzije *.guiskin* koja sadrži informacije o tipu fonta, boji fonta te samoj boji i izgledu svih komponenti.

U našoj igri glavni izbornik je definiran slijedećom skriptom:

```
using UnityEngine;
using System.Collections;

public class MainMenuGUI : MonoBehaviour {

    public GUISkin skin;
    public Texture2D background;

    void OnGUI() {
        GUI.skin=skin;
        GUI.BeginGroup(new Rect(0,0,Screen.width,Screen.height));

        GUI.Label(new Rect(0,0,Screen.width,Screen.height), background);

        GUI.Box(new Rect(150,150,350,250),"Projekt Sibenik");

        if (GUI.Button(new Rect(200,200,200,60),"Pokreni igru")) {
            Application.LoadLevel(1);
        }

        if (GUI.Button(new Rect(200,300,200,60),"Izadji")) {
            Application.Quit();
        }

        GUI.EndGroup();
    }
}
```

Sadržaje varijabli *skin* te *background* definiramo u sučelju *Unity*-a.



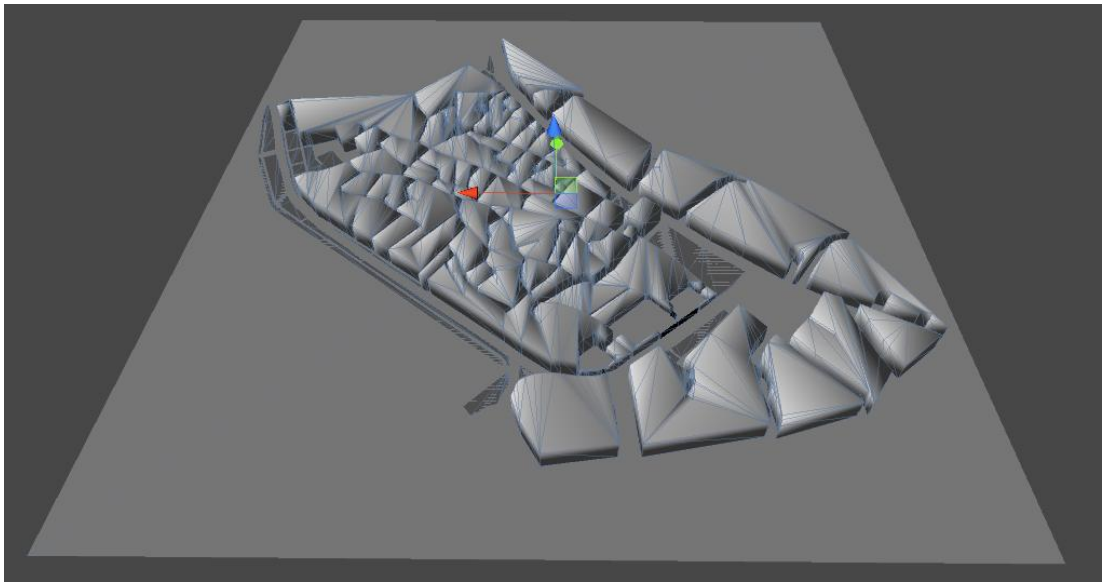
Slika 2.11 Glavni izbornik igre

2.5 Teren

Prethodno modelirani prototip grada Šibenika trebalo je učitati u *Unity*. Budući da je model izrađen u programu *Blender*, najpogodniji format za prijenos modela u *Unity* pokazao se *.fbx*.

Da bi učitani model mogao funkcionirati kao realni teren kojim bi se moglo trčati i koji bi se mogao istraživati, svakom objektu kojeg teren sadrži dodana je komponenta *Mesh Collider* koja omogućava prepoznavanje sudara među objektima i ne dozvoljavanje da se kroz njih prolazi.

Slika 2.12 prikazuje učitani prototip grada Šibenika.



Slika 2.12 Prototip grada Šibenika učitan u *Unity*

Radi realnijeg okruženja učitanom prototipu dodana je pozadina, popularno *Skybox*. Slika 2.13 prikazuje mrežu kocke iz koje se, spajanjem bridova, stvorilo 3D okruženje, dok na sljedećoj slici (Slika 2.14) možemo vidjeti i kako to izgleda zajedno s terenom, učitano u *Unity*.

Slika 2.13 Mreža *Skybox*-a

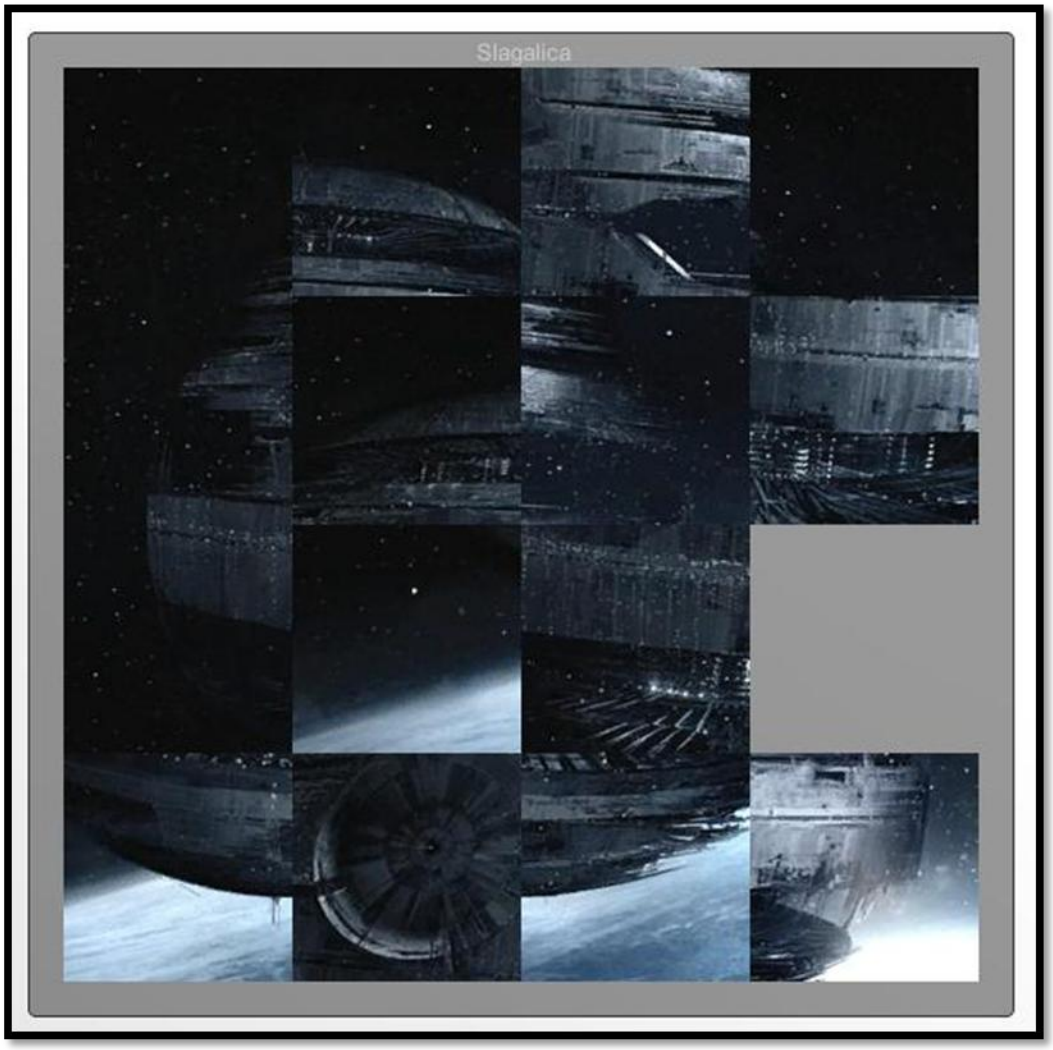
Slika 2.14 Teren sa stvorenom 3D pozadinom

2.6 Slagalica

„Slagalica“ je ime jedne od podigara unutar projekta (eng. minigame). Radi se o jednostavnoj pravokutnoj slikovnoj slagalici koja se sastoji od 8 ili 15 komadića (3x3 ili 4x4 uz to da jedan komadić „fali“ kao i u pravim slagalicama).

Slagalica je riješena kada se svi komadići slike poslože na svoje mjesto.

Na slici 2.15 prikazana je slagalica čiji su dijelovi pomiješani. Slagalicu je moguće presložiti tako da uzastopnim pomicanjem komadića složimo početnu sliku i na taj način riješimo podigru.



Slika 2.15 Slagalica prije slaganja

Sivi kvadratić predstavlja prazninu i na tu poziciju je moguće pomaknuti bilo koji od susjednih komadića. Na primjer, za zadanu situaciju pritiskom strelice prema dolje fragment koji se nalazi iznad praznine spušta se prema dolje. Cilj igre je nizom uzastopnih poteza dobiti originalnu sliku prikazanu na slici 2.16.

Slika 1.16: Složena slagalica[2]

U slagalicu je moguće umetnuti bilo koju sliku pravokutnih dimenzija bez potrebe da se samostalno raspodjeljuju dijelovi.

Program sam gradi ploču sa zadanim brojem dijelova i razbacuje elemente na način da generira nekoliko slučajnih poteza koji „kvare“ sliku.

Igrač tada za zadatak ima složiti originalnu sliku.

Evo još jednog primjera slagalice.

[2] http://24.media.tumblr.com/tumblr_m41o9wUSPW1ruox2ao1_1280.jpg

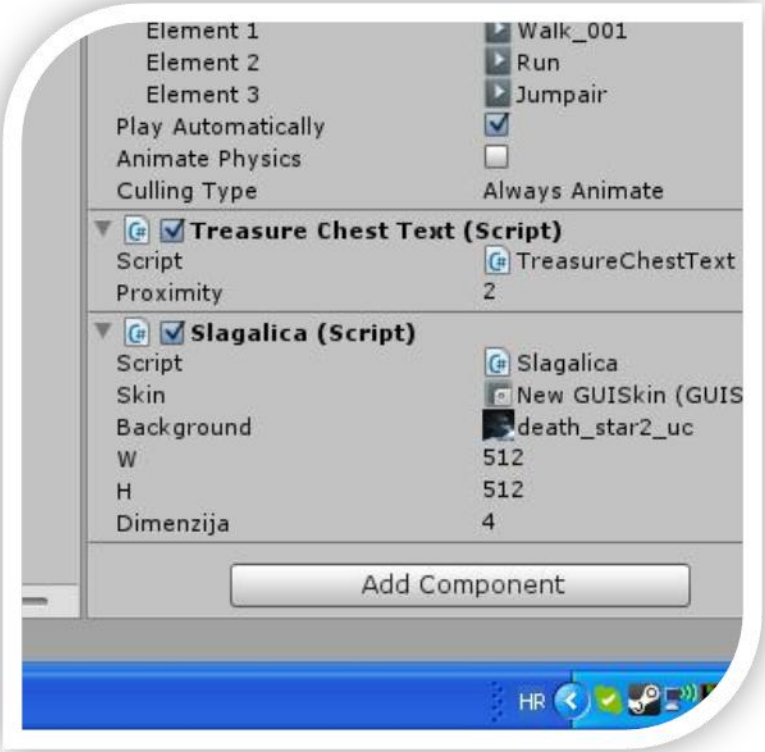
Slika 2.17 Slagalica prije slaganja – Grb grada

Slika 2.18 Složena slagalica – Grb grada

Slagalicu je moguće podesiti unutar *UnityEngine*-a.

Potrebno je postaviti dimenzije prozora slagalice (W – širina, H – visina), sliku slagalice (*Background*) i dimenziju slagalice (3 za 3x3, 4 za 4x4).

Prilikom pokretanja igre skripta inicijalizira sve potrebne parametre i kreira nasumično razbacanu slagalicu.



Slika 2.19 Podešavanje slagalice

OPASKA: Prilikom pisanja skripte ustanovljen je problem unutar *UnityEngine*-a. Prilikom pokretanja skripte konstantno dojavljuje pogrešku *DivisionByZero* na određenom mjestu u kodu, iako zapravo sve normalno funkcionira i nema dijeljenja s nulom. Kako ta „greška“ ne utječe na izvođenje sktipte i budući da se najvjerojatnije radi nekim neslaganjima unutar samog alata, nije bilo potrebno ništa ispraviti.

3. Upute za korištenje

3.1 Instalacija

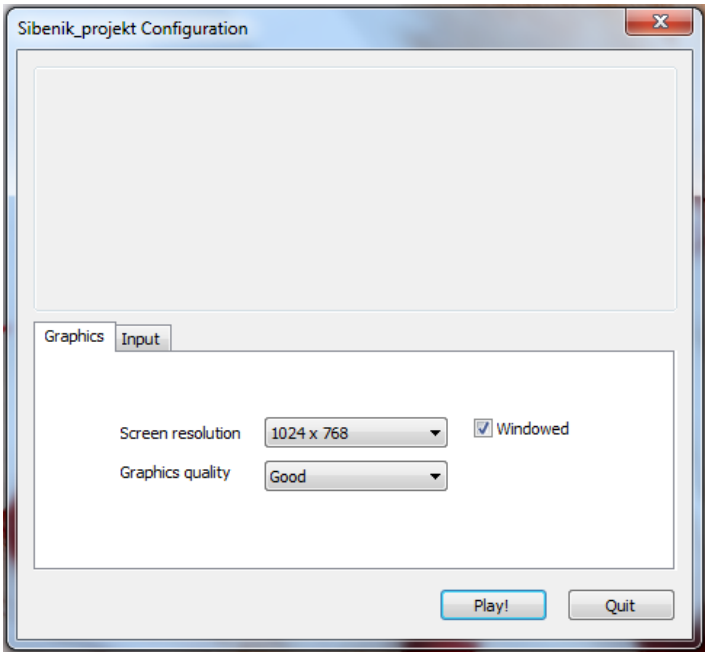
Za pokretanje igre potrebno je skinuti zip datoteku u neki direktorij na računalu te raspakirati njen sadržaj.

Slika 3.1 Ikona za pokretanje igre

3.2 Postavke

Ikona kojom se pokreće igra je prikazana na slici 3.1. Kod klika na ikonu otvara se sljedeći izbornik sa slike 3.2. Na njemu su prikazane preporučene postavke za pokretanje.

Klikom na gumb *Play!* Otvara se glavni izbornik igre ranije prikazan na slici 2.11.



Slika 3.2 Postavke igre

3.3 Kontrole

Glavni meni pruža opcije 'Pokreni igru' i 'Izadji'.

Igru pokrećemo klikom na gumb „Pokreni igru“ na glavnom izborniku igre.

Igrom upravljamo pomoću tipkovnice i miša. Unutar same igre koriste se sljedeće kontrole:

- Pomicanje miša – upravljanje kamerom
- 'W' – pomicanje prema naprijed
- 'S' – pomicanje prema nazad
- 'A' – pomicanje lijevo
- 'D' – pomicanje desno
- 'G' – interakcija sa objektima u igri
- 'Spacebar' – skakanje
- 'Esc' – vraćanje na glavni izbornik

4. Literatura

[1] http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro

[2] http://unity3d.com/

[3] http://www.exchange3d.com/

[4] http://3dexport.com/

[5] http://archive3d.net/

[6] http://24.media.tumblr.com/tumblr_m41o9wUSPW1ruox2ao1_1280.jpg

[7] http://en.wikipedia.org/wiki/MakeHuman

[8] http://www.blender.org/education-help/tutorials/