Proceduralno generiranje 3D svijeta u Unity3D

Brazdil, Vedran

Undergraduate thesis / Završni rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:874427

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

PROCEDURALNO GENERIRANJE 3D SVIJETA U UNITY3D

Završni rad

Vedran Brazdil

Osijek, 2016.

SADRŽAJ

1. UVOD 1
1.1. Zadatak završnog rada 1
2. UNITY 3D 2
2.1. Grafika 3
2.1.1. Osvjetljenje 3
2.1.2. Sjene 6
2.1.3. Kamere 7
2.2.Stvaranje terena (površine) 9
3. PROGRAMSKI JEZIK C# 13
3.1. Ciljevi nastanka 13
3.2. Sintaksa 14
4. IZRADA IGRE U UNITY 3D 15
4.1. Scene 15
4.2. Kamere 16
4.3. Objekti 19
4.4. Igrač i kontrole 22
4.5. Izbornik (menu) i canvas 27
5. PROCEDURALNO GENERIRANJE SVIJETA 32
5.1. PearlinNoise funkcija 32
5.2. Drveće 34
5.3. Rudnici 37
5.4. Oblaci i nevidljivi zidovi 40
6. ZAKLJUČAK 43
LITERATURA 44
SAŽETAK 45
ABSTRACT 46
ŽIVOTOPIS 47

1. UVOD

Tema završnog rada je "Proceduralno generiranje svijeta u Unity3D". U teorijskom dijelu opisan je program Unity3D pomoću kojeg je izrađena 3D igra. Također je opisan programski jezik koji je korišten, to jest C#. Postupak izrade 3D igre koja je izrađena, a koja je ujedno i praktični dio zadatka, biti će napisan u seminaru. U seminaru potrebno je teorijski obuhvatiti jednu od metoda proceduralnog generiranja svijeta u Unityju te ju upotrijebiti u igri. Praktični dio završnog rada napravljen je koristeći osobno računalo s Windows 7 operacijskim sustavom. Osim Unity3D-a od pomoćnih programa korišten je GIMP za uređivanje izgleda igre te MonoDevelop za pisanje programskog koda.

1.1. Zadatak završnog rada

U teorijskom dijelu rada potrebno je opisati osnove 3D grafike u računalnim igrama. Potrebno je opisati najčešće korištene pristupe proceduralnom generiranju 3D svijeta u računalnim igrama. Odabrati jedan pristup i detaljno ga opisati. U praktičnom dijelu potrebno je izraditi 3D igru u Unity 3D razvojnom okruženju koja će koristiti odabrani pristup za proceduralno generirani svijet. Generirani svijet mora sadržavati elemente slučajnosti čime će pri svakom pokretanju igre biti drugačiji.

Tehnologije: C#, Unity3D.

2. UNITY 3D

Unity [17][1] (hr. jedinstvo) je cross-platforma za izradu igara koju je razvio Unity Technologies i koristi se za izradu igara za računala (PC), konzole, mobilne uređaje te web stranice. Prvi put predstavljen je samo za OS X na Appleovoj svjetskoj konferenciji za programere 2005. godine. Od tada se proširio na još nekoliko platformi. Unity je osnovan softverski sadržaj za izradu igrica za platformu Wii U. Od tada, izašlo je pet bitno velikih verzija Unitya. 2006. godine na WWDC-u, Apple Inc. je prozvao Unity za najbolje korištenje Mac OS X grafike. Trenutno najnovija verzija Unitya, Unity 5, ujedno i jedna od najboljih razvojnih platformi za stvaranje 2D i 3D igara te interaktivnih iskustava. Unity 5 donosi novu umjetničku moć. Poboljšana učinkovitost čini teški posao glatkim i zanimljivim te uz višeplatformsku podršku ima više korisnika nego ikada prije.

Uz naglasak na prenosivost, razvoj cilja na sljedeće API-je: Direct3D na Windowsu i Xbox 360; OpenGL na Macu i Windowsu; OpenGL ES na Androidu i iOS-u; te vlasničke API-je na video igračkim konzolama. Unity dozvoljava specifikaciju kompresije i rezolucijske postavke za svaku platformu koju Unity podupire te pruža podršku za „bump“ mapiranje, refleksijsko mapiranje, „parallax“ mapiranje, prostorno ekranski prostor ambijentalne okluzije (SSAO), dinamično zasjenjivanje korištenjem mapa sjena, „render-to-texture“ i puni zaslon poslije procesnih efekta. Unityev grafički mehanizam platformske raznolikosti može pružiti sjenčanje s raznolikim varijantama. Time dozvoljava Unityju da detektira najbolju varijantu za trenutni video hardver te ako ni jedan nije dobar, dozvoljava vraćanje na alternativno sjenčanje koje može žrtvovati buduće performanse.

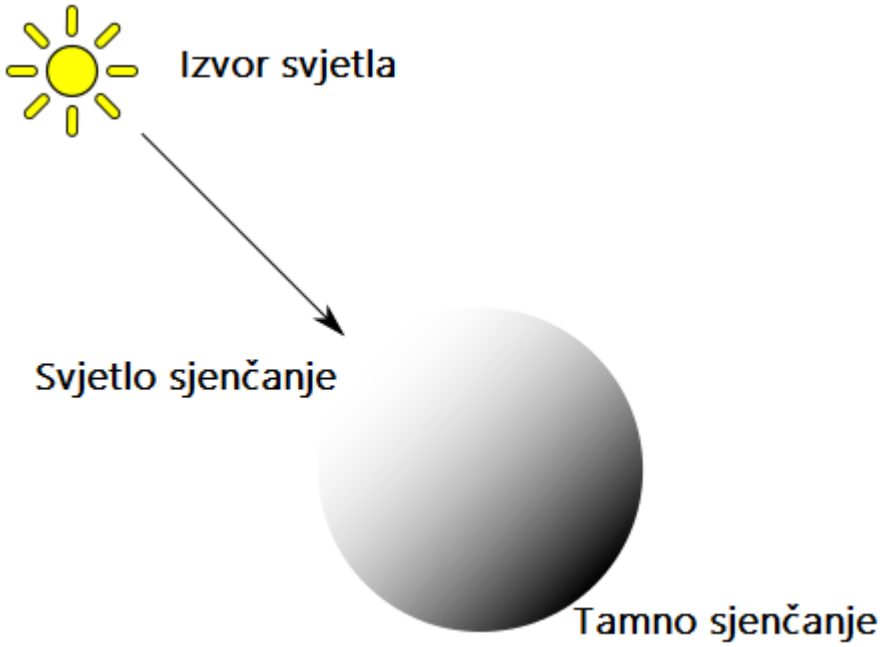
Unity je poznat po svojoj mogućnosti da razvija igre na više platforma. Unutar projekta, programeri imaju kontrolu nad isporukom na mobilne uređaje, web preglednike, stolna računala i konzole. Podržane platforme uključuju i Android, Apple TV, BlackBerry 10, iOS, Linux, Nintendo 3DS line, OS X, PlayStation 3, PlayStation 4, PlayStation Vita, Unity Web Player (koji uključuje Facebook), Wii, Wii U, Windows Phone 8, Windows, Xbox 360 i Xbox One. Uključuje i poslužitelja vlasništva i Nvidia PhysX, grafički program. Unity Web Player je preglednikov (browserov) dodatak kojeg podržavaju samo Windows i OS X. Korištenje Unity Web Playera obustavljeno je u korist WebGLa. Unity je osnovna oprema za razvoj programa na Nintendovoj Wii U igraćoj konzoli. Nintendo uz svaku dozvolu za programiranje daje i besplatnu kopiju samog programa.

2.1. Grafika

Unity nudi iznenađujuću vizualnu točnost, snagu generiranja i ambijent. S obzirom na visoko razvijenu tehnologiju grafike koju koristi, Unity omogućava programerima da igre naprave baš onako kako su to i zamislili. Razumijevanje grafike [4] ključ je za pravilno dodavanje elementa određene dubine u svoju igru.

2.1.1. Osvjetljenje

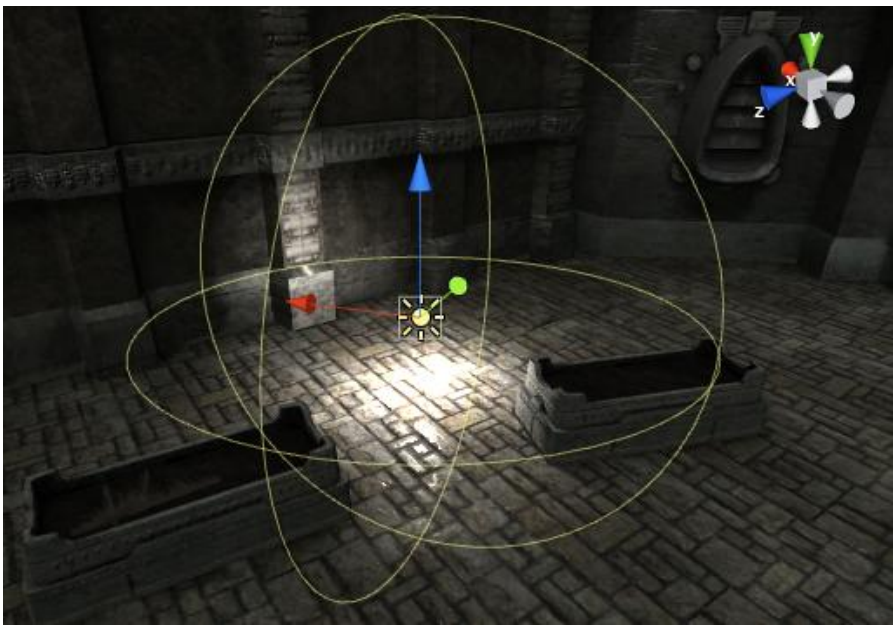
Jedan od najbitnijih dijelova grafike u Unityju je osvjetljenje [5] koje daje ambijent igrama. Da bi Unity izračunao nijansiranje 3D objekta, treba znati intenzitet, smjer i boju svjetlosti koja pada na taj objekt. Izvor svjetlosti se dobiva tako da u scenu ubacimo objekt svjetlost.



Sl. 2.1.1.1 Osvjetljenje [5]

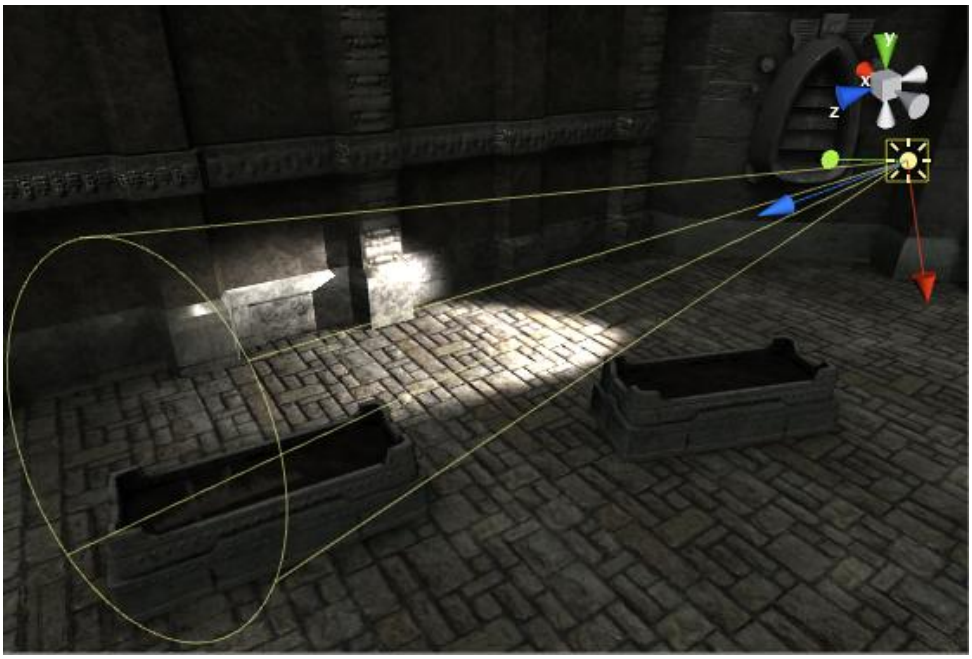
Početna boja i intenzitet na objektu svjetlost namješteni su identično za sve vrste svjetlosti, ali smjer ovisi o vrsti svjetlosti koju koristimo. Naravno, svjetlost se može smanjivati s obzirom na udaljenost od izvora. Postoje 4 vrste svjetlosnih izvora koja se mogu koristiti u Unityju.

Prva vrsta osvjetljenja su point lights (točkasta svjetla). Točka izvora ove vrste svjetlosti je locirana na jednoj točki u prostoru i iz te točke šalje svjetlost na sve strane jednako. Smjer svjetla koje pada na površinu je crta od točke dodira i natrag u centar svjetlosnog objekta. Intenzitet se smanjuje s obzirom na udaljenost od izvora koje dolazi do iznosa 0 na točno određenoj udaljenosti. Ova vrsta izvora svjetlosti korisna je za simuliranje svjetlosti lampe, baklje ili sličnih vrsta svjetlosti u sceni. Mogu se koristiti tako da na tren osvijetle scenu prilikom eksplozija ili udara groma.



Sl. 2.1.1.2 Točkasta svjetla unutar scene [5]

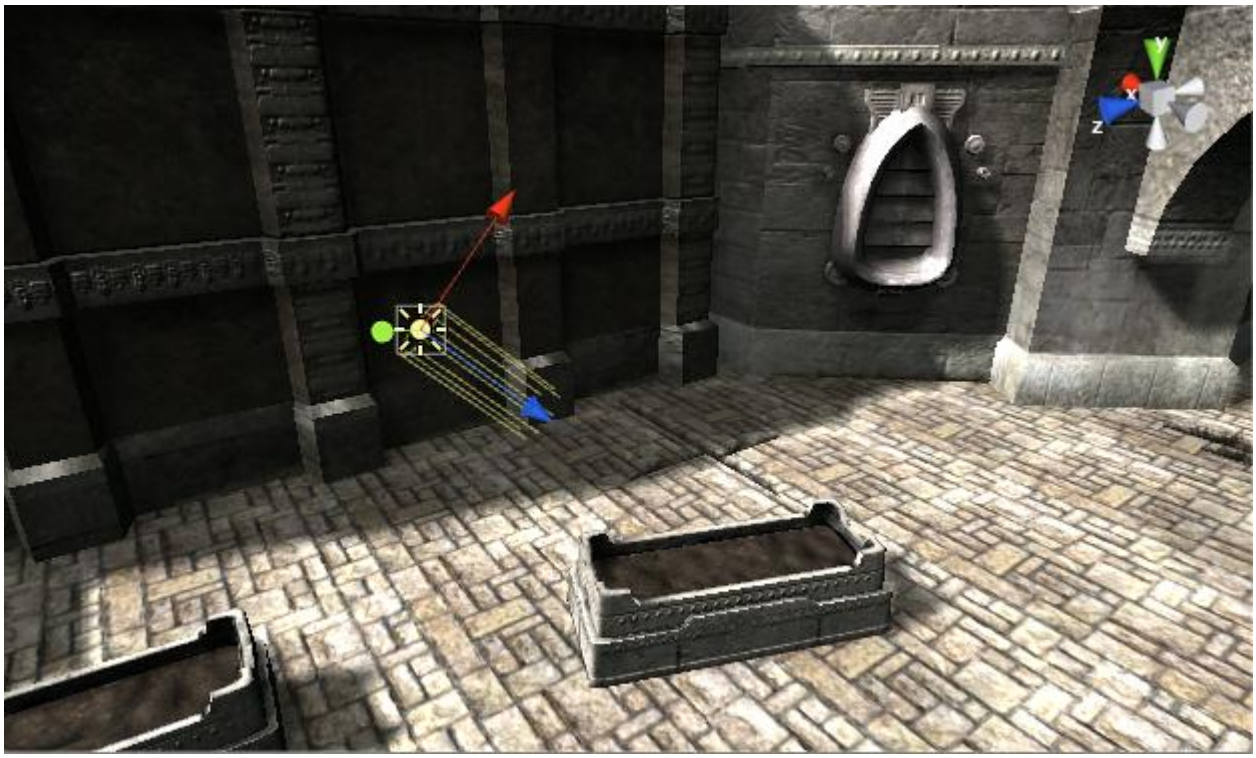
Druga vrsta osvjetljenja su spot lights (reflektori). Slično kao kod točkastih svjetla, reflektori imaju specifičnu lokaciju i domet svjetlosti pri kojoj se intenzitet postupno smanjuje. Međutim, reflektori su ograničeni na jedan kut što rezultira stožasto područje osvjetljavanja. Reflektori su generalno korišteni za umjetne izvore svjetlosti kao što su ručne svjetiljke, svjetla na autu ili žmigavci. S obzirom na to da se na smjer kontrolira kodom ili animacijom, pomicanjem reflektora osvijetlit će samo mali dio scene i tako stvoriti dramatično okruženje.



Sl. 2.1.1.3 Reflektorska svjetla unutar scene [5]

Treća vrsta osvjetljenja su directional lights (usmjerena svjetla). Usmjerena svjetla nemaju neko točno određeno mjesto odakle izvire svjetlost, pa se zato ovakav objekt svjetlosti može postaviti bilo gdje na sceni. Svi objekti na sceni su osvijetljeni kao da svjetlost dolazi uvijek iz istog smjera. Udaljenost između objekta i izvora svjetlosti se ne može izračunati pa se ni svjetlost ne smanjuje s obzirom na udaljenost.

Usmjerena svjetla predstavljaju nekakav veliki izvor svjetlosti koji se nalazi izvan postojeće mape. Dok se u realnoj sceni mogu koristiti da bi simulirali sunce ili mjesec, u svijetu apstraktne igre mogu se koristiti kao lagan način da bismo ubacili sjene u igru te objektima dali realan izgled, pritom ne brinući se o tome odakle svjetlost dolazi.



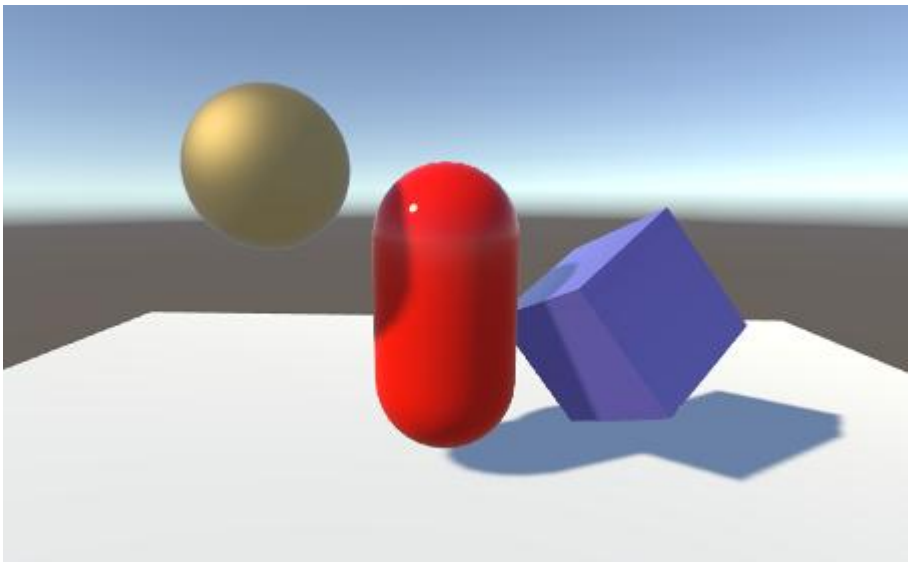
Sl. 2.1.1.4 Usmjerena svjetla unutar scene [5]

Četvrta vrsta osvjetljenja su area lights (svjetla prostora). Svjetla prostora određena su pravokutnikom u prostoru. Svjetlo se emitira u svim smjerovima, ali samo iz jedne strane pravokutnika. Svjetlina opada u određenom dometu. S obzirom na to da je računanje svjetline prilično zahtjevan proces, svjetla prostora nisu omogućena pri nekim pokretanjima igre.

S obzirom na to da iz svjetla prostora svjetlina izlazi u svim smjerovima, sjenčanje je pretežito mekše i suptilnije nego u ostalim metodama osvjetljavanja. Može se koristiti da bi se stvorila realno osvijetljena ulica s uličnim svjetlima ili hrpa svjetla koja se nalaze vrlo blizu igraču. Mala svjetla prostora mogu simulirati male izvore svjetlosti kao što su svjetla unutar kuće te im dati puno realniji osjećaj nego s točkastim izvorima svjetla.

2.1.2. Sjene

Unityeva svjetla mogu stvarati sjene koje s objekta padaju na drugi dio tog objekta ili na neki drugi objekt u prostoru. Sjene daju određenu dubinu i realnost sceni zbog toga što iznose veličinu i položaj objekta koji bi inače izgledao ravno.



Sl. 2.1.2.1 Scena gdje objekti imaju sjene [5]

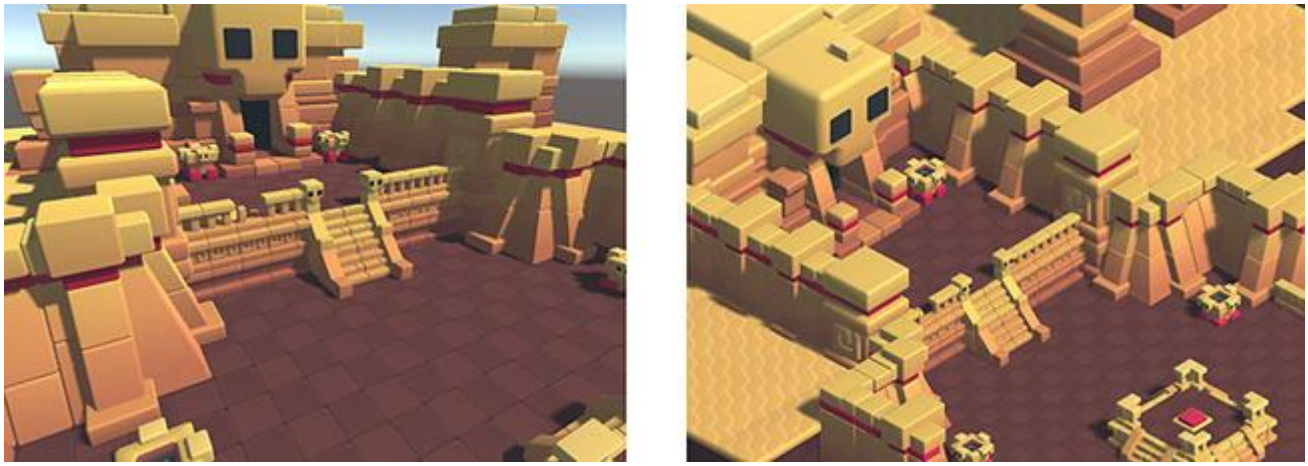
Ako uzmemo za primjer scenu s jednim objektom svjetlosti. Zrake svjetla putuju u ravnim crtama od izvora i mogu eventualno sudariti s objektom u sceni. Jednom kad zraka udari objekt, ne može nastaviti putovati i ne može osvijetliti ništa drugo. Sjene koje objekt baca su jednostavno mjesta na koja te zrake ne mogu doći.

2.1.3 Kamere

Unity scena stvorena je slaganjem i premetanjem objekta u trodimenzionalnom prostoru. S obzirom na to da je ekran korisnika dvodimenzionalan, mora postojati način da bi se taj prizor uhvatio i „izravnao“ za prikaz na ekranu. To uspijevamo postići uz pomoć objekta kamera [6].

Kamera je objekt koji definira pogled na prostor scene. Pozicija objekta u prostoru određuje točku gledišta, dok osi objekta naprijed (Z) i gore (Y) određuju dubinu i domet gledanja, odnosno vrh ekrana. Komponenta kamere isto tako definira veličinu i oblik regije koja se nalazi unutar vidika. Kada namjestimo sve ove parametre, kamera može prikazivati što ona trenutno „vidi“ na naš ekran. Ako se objekt kamere pomiče i rotira, pomicat će se i pogled kamere na našem ekranu isto kao što se pomiče i objekt.

Kamere mogu biti postavljene na različite načine i time znatno promijeniti igru. Jedni od takvih primjera su perspektivne i ortografske kamere.



Sl. 2.1.3.1 Ista scena sa perspektivnom kamerom (lijevo) i ortografskom kamerom (desno) [6]

Kamera u stvarnosti, naravno i samo ljudsko oko, vidi svijet tako da su predmeti u daljini manji nego predmeti u blizini. Tehnika stvaranja takvog pogleda u igri je već poznata te se koristi u umjetnosti i grafici te je vrlo važna za stvaranje realnih scena u igrama. Naravno da Unity podržava perspektivne kamere, ali za neke svrhe bolje je koristiti pogled bez ovog efekta.

Na primjer, možda želite stvoriti mapu ili prikaz informacija koji ne bi trebao izgledati kao objekt u stvarnosti. Kamera koja ne smanjuje objekte s obzirom na njihovu udaljenost zove se ortografska kamera. Unityeve kamere imaju opciju za tu vrstu kamera. Perspektivni i ortografski modovi pogleda na scenu nazivaju se još i projekcijama kamera.

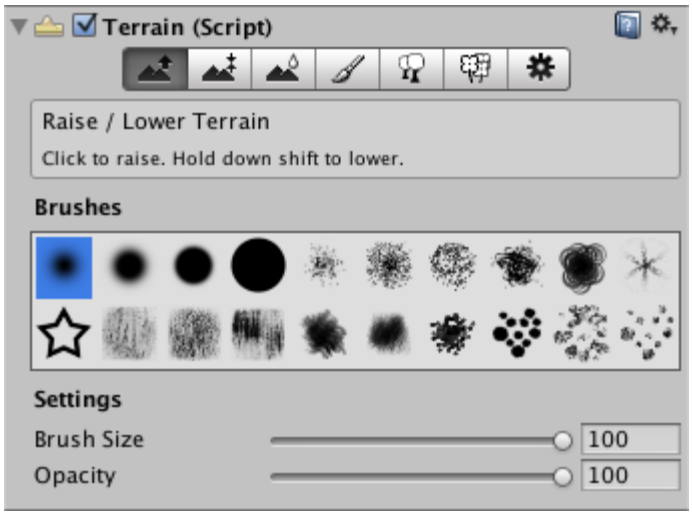
2.2 Stvaranje terena (površine)

Unityev sistem za stvaranje terena [7] omogućuje korisnicima da jednostavno dodaju velike krajolike u igru. Za vrijeme izvođenja, stvaranje terena vrlo je dobro optimizirano i pruža učinkovito renderiranje dok smo u izborniku. Postoji veliki izbor alata koji omogućuju vrlo lagano i brzo stvaranje terena za igru.



Sl. 2.2.1 Primjer scena gdje se koristi stvaranje površine [7]

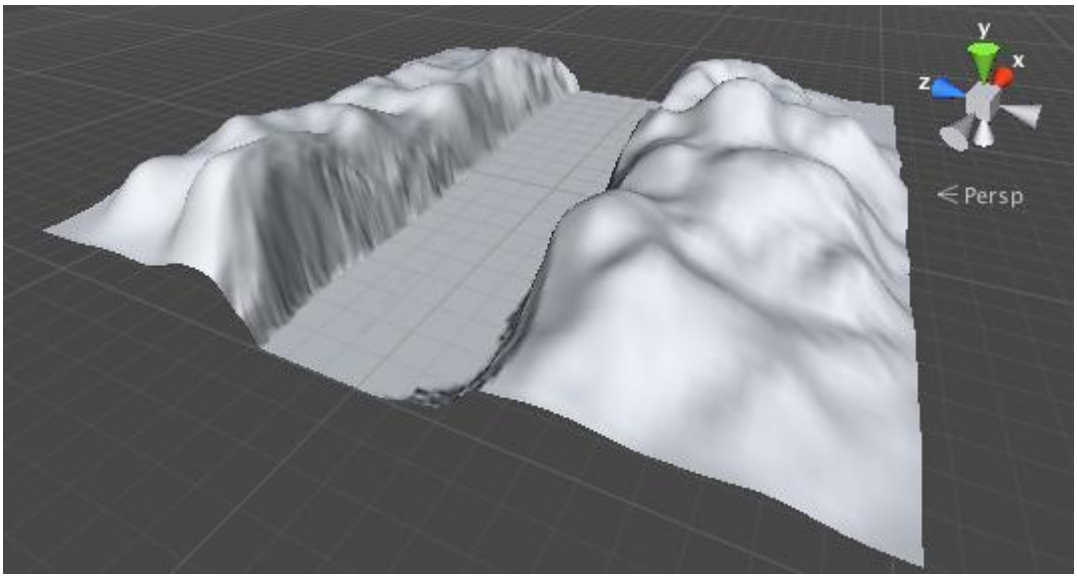
Stvaranje terena u sceni se postiže jednostavno klikom na GameObject > 3D > Object > Terrain iz izbornika. Naravno, ovaj teren će na početku biti samo jedna obična ravna ploča. Klikom na stvoren teren, otvara se izbornik koji daje velik raspon opcija kako možemo promijeniti izgled tog terena. Mijenjanje opcija u izborniku terena nam daje mogućnost da dobijemo željeni izgled.



Sl. 2.2.2 Izbornik stvorenog terena

S izuzetkom za alat stvaranja drveća, svi alati na alatnoj traci pružaju set „četkica" i opcije za njihovu veličinu i gustoću. Nije slučajnost da ove četkice sliče na alate iz programa za obradu slika. Razlog tomu je baš to što se na takav isti način i postižu detalji na terenu, to jest „crtanjem" detalja na krajolik. Odabirom alata za crtanje na izborniku, kursor miša se pretvara u svjetlosnu točku te nam omogućuje crtanje po terenu. Korištenjem alata iz ovog izbornika može se bojati, pomicati teren gore i dolje te mijenjati njegov oblik. Veličina i gustoća četkice znatno mijenjaju način bojanja i daju određeni efekt krajoliku.

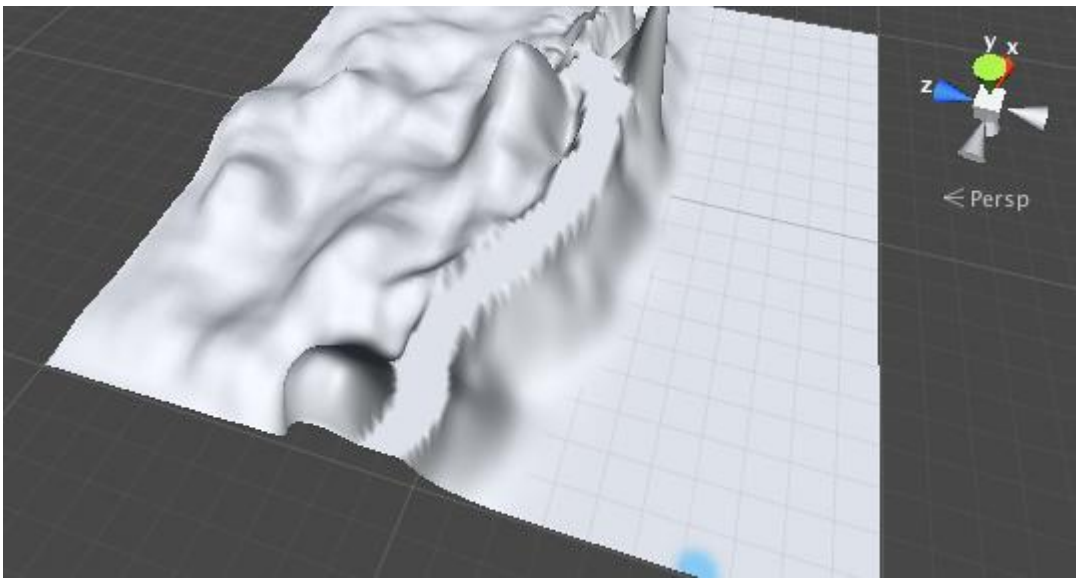
Prva tri alata na izborniku za stvaranje terena koriste se za određivanje, odnosno mijenjanje visine terena. Počevši s lijeva, prva tipka aktivira izbornik za mijenjanje visine klikom miša na određeni dio terena kojeg želimo povisiti ili držeći shift tipku da bismo ga snizili. Radi na principu sličnom kao i četkice za crtanje, odnosno bojanje terena. Mijenjanjem četkica dok mijenjamo visinu možemo dobiti različite oblike terena. Na primjer, brda i doline dobivamo koristeći mekše četkice i mijenjanje površine gore i dolje, dok planine i stijene dobivamo koristeći oštre četkice i podizanjem površine prema gore.



Sl. 2.2.3 Slika primjera stvaranja brda i dolina [7]

Drugi alat s lijeve naziva se bojanje visina. Sličan je prvom alatu, ali je razlika u tome što ima opciju za namještanjem ciljane visine. Klikom na objekt terena smanjuje se dio terena koji je iznad zadane visine, dok se dio terena koji je niži od zadane visine podiže. Tražena visina može se upisati u izborniku ili se može odabrati uzorak s terena koristeći shift tipku i klik miša.

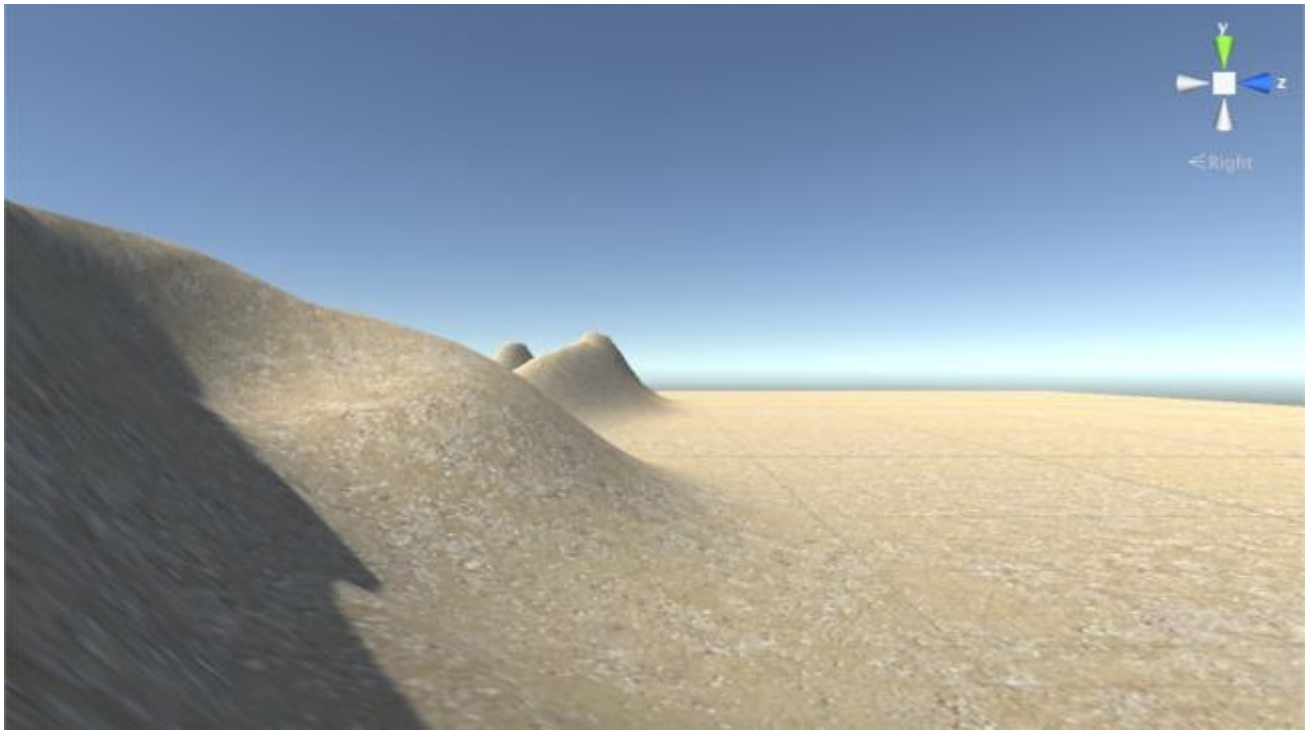
Ovaj alat daje i mogućnost sravnjivanja cijelog terena na određenu visinu jednim klikom. Zbog toga se ova opcija koristi za postaviti početne visinu terena, ovisno o tome želimo li samo planine, doline ili oboje. Ovaj se alat još koristi i za stvaranje ravnog terena poput puteljaka, staza, cesta ili slično.



Sl. 2.2.3 Slika primjera poravnavanja terena [7]

Treći alat s lijeve, izglađivanje visina ne mijenja visinu terena nego jednostavno uklapa njegovu visinu s obližnjim visinama terena. Ovaj alat omekšava površinu te joj daje realniji izgled i lakšu prohodnost unutar igre. Sličan je kao alat za zamućivanje u programima za obradu slika. Ovaj alat najbolje je koristiti nakon rada s četkicama koje su preoštre, odnosno koje ostavljaju previše hrapavu površinu.

Moguće je dodavati teksture na površine terena da bismo dobili željenu sliku krajolika. S obzirom na to da su tereni velike površine, poznato je da se stalnim ponavljanjem iste slike može dobiti približno realna površina terena. Takvo ponavljanje teško se primjećuje unutar igre. Samo jedna slika je dovoljna da služi kao tekstura pozadine kroz cijeli krajolik, no korištenjem više različitih slika dobivamo realniji i zanimljiviji krajolik. To su na primjer teksture trave, pijeska ili snijega. Mijenjanjem transparentnosti i gustoće dobivamo određene nijanse na određenim mjestima terena.



Sl. 2.2.4 Slika primjera dodavanja tekstura terenu [7]

3. PROGRAMSKI JEZIK C#

C# [10] je više-paradigmatski programski jezik koji obuhvaća jako tipkane, imperativne, deklarativne, funkcionalne, generičke, objektno-orijentirane (bazirano na klasama) i komponentno-orijentirane programske discipline. Microsoft je razvio C# u okviru svoje .NET inicijative i kasnije odobren kao standard od strane Ecma (ECMA-334) i ISO (ISO/IEC 23270:2006). C# je jedan od programskih jezika dizajniran za uobičajen jezik infrastrukture (Common Language Infrastructure).

Objektno-orijentiran programski jezik C# opće je namjene. Tim koji razvija ovaj jezik pod vodstvom je Anders Hejlsberga. Najnovija trenutna verzija je C# 6.0, koja je objavljena 20.07.2015. godine.

3.1. Ciljevi nastanka

ECMA standard napravio je listu ovih ciljeva za C#:

- C# jezik je namijenjen da bude jednostavan, moderan, opće namjene, objektno orijentiran programski jezik.
- Sam jezik i njegova implementacija treba osigurati potporu načelima programerskih inženjera kao što su jake provjere tipa, provjere granica array-a, detekcija pokušaja korištenja prethodno ne određenih varijabli te automatsko prikupljanje otpada. Programska robusnost, izdržljivost i programerska produktivnost su vrlo bitne.
- Jezik je namijenjen za korištenje u razvoju softverskih komponenata prikladnih za primjenu u distribuiranim okruženjima.
- Prenosivost je vrlo bitna za izvorni kod i programere, posebno za one koji su već upoznati s programskim jezicima C i C++.
- Podrška za internacionalizaciju je vrlo bitna.
- C# je namijenjen da bude pogodan za pisanje aplikacija za vlastiti i tuđi sistem u rasponu od vrlo velikog, koji koristi sofisticiran operacijski sustav, sve do vrlo malog s nekim posebnim funkcijama.

- Iako su C# aplikacije namijenjene da budu ekonomične u pogledu memorije i zahtijevane procesorske snage, jezik nije namijenjen da se natječe direktno izvedbom i veličinom s C ili nekim asemblerskim jezikom.

3.2. Sinktaksa

Jezgra sintakse C# programskog jezika [11] slična je ostalim sintaksama C stilskih programskih jezika kao što su C, C++ i Java.

Neke od sličnih stvari u sintaksi ovih jezika su:

- Točka sa zarezom („ ; “) koristi se za označavanjem kraja izjave, odnosno naredbe.
- Vitičaste zagrade („ { “ i „ } “) koriste se za grupiranje naredba. Naredbe se tako grupiraju u funkcije (metode), funkcije se tako grupiraju u klase, a klase se tako grupiraju u namespace (imenski prostor).
- Varijable se dodjeljuju koristeći znak jednakosti („ = “), ali uspoređuju se korištenjem dva uzastopna znaka jednakosti („ == “).
- Kockaste zagrade („ [“ i „] “) koriste se s matricama i poljima da bi ih deklarirali i da bi im dali vrijednost pod određenim indeksom.

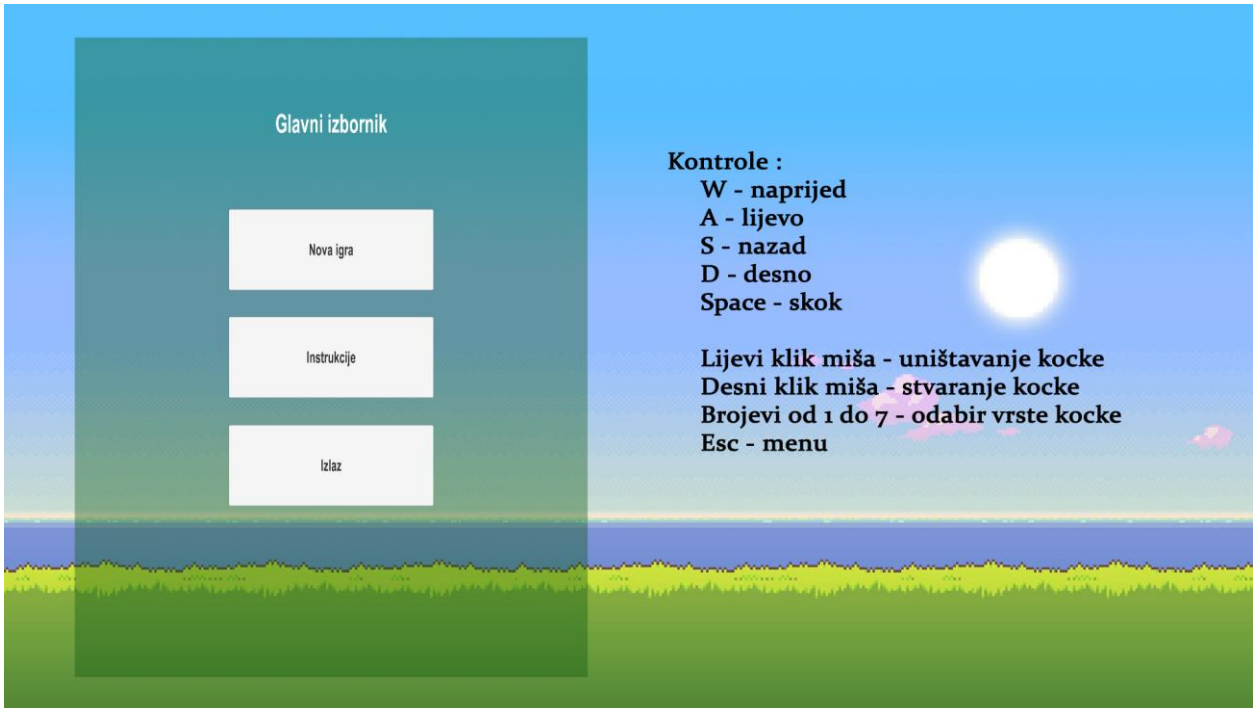
4. IZRADA IGRE U UNITY 3D

U ovom poglavlju bit će objašnjeno sve što je potrebno da bi se napravila 3D igra koristeći program Unity 3D. U igri će se svijet nasumično i proceduralno generirati svaki puta pri pokretanju igre te će igra biti namijenjena za samo jednog igrača.

4.1. Scene

Scene sadrže sve objekte igre. Mogu se koristiti da bi se stvorio glavni izbornik, samostalni nivo ili bilo što slično. Svaku scenu zasebno možemo gledati kao samostalni nivo. Onaj koji radi igru u svakoj sceni stvara njeno okruženje, predmete, prepreke, dekoraciju, odnosno stvara igru uz pomoć manjih dijelova složenih na sceni da čine jednu sliku.

U ovoj igri koriste se dvije scene. Početna scena s indeksom „0“ sastoji se od glavnog izbornika (glavni menu) koji sadrži 3 tipke.



Sl. 4.1.1 Početna scena

Pritiskom tipke „Nova igra“ pokreče se druga scena s indeksom „1“. Druga scena je scena igre.

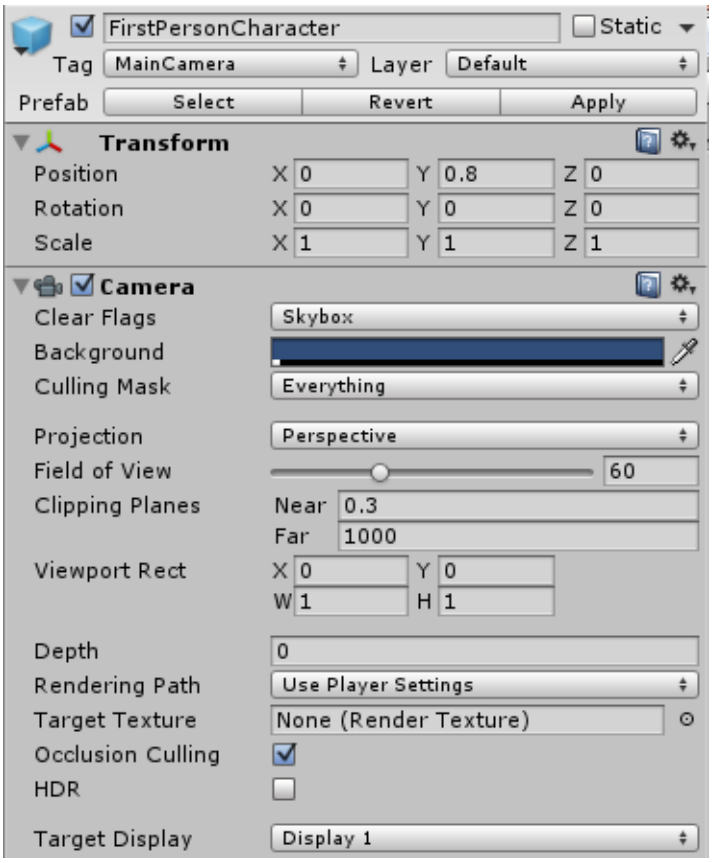


Sl. 4.1.2 Scena igre

4.2. Kamere

Unity scena stvorena je željenim slaganjem i premetanjem objekta u trodimenzionalnom prostoru. S obzirom na to da je ekran korisnika dvodimenzionalan, mora postojati način da bi se taj prizor uhvatio i „izravnao“ za prikaz na ekranu. To se uspijeva uz pomoć objekta kamera [6]. Kamera je objekt koji definira pogled na prostor scene. Kamere pojednostavljuju kretanje igrača te znatno mijenjaju način i vrstu igranja igre.

U ovoj igri koriste se dvije kamere. Prva kamera je kamera koja prikazuje što vidi igrač. Ta je kamera spojena na igrača te se kreće kako se kreće igrač i omogućava igranje igre iz prvog lica.



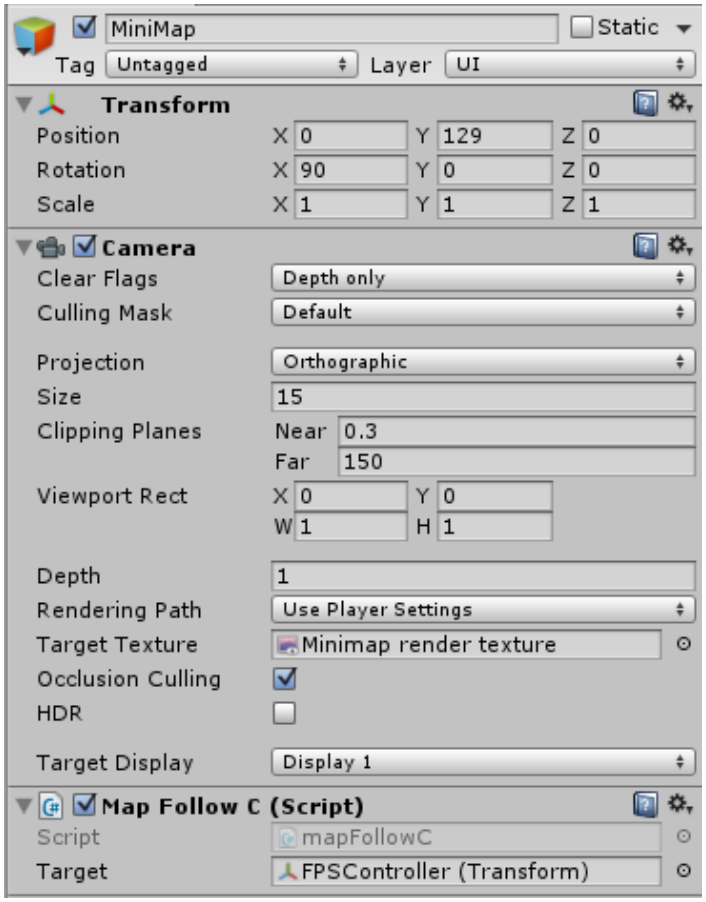
Sl. 4.2.1 Postavke kamere igrača



Sl. 4.2.2 Kamera igrača unutar igre

Dok je prva kamera ujedno i glavna kamera te bez nje igra ne bi mogla normalno funkcionirati, u igru je ubačena još jedna kamera. Druga kamera je sporedna kamera koja služi

kao „mini mapa“ („mala mapa“). Ona pomaže igraču da se snalazi na mapi. Isto kao i prva kamera spojena je s igračem te prati njegovo kretanje. Velika razlika je ta što druga kamera snima mapu odozgo i njen prikaz je ortografski što znači da prikazuje sliku u 2D obliku gledajući samo na duljinu i širinu, dok zanemaruje visinu. Prikaz je umanjen kako ne bi zauzimao previše prostora na ekranu.



Sl. 4.2.3 Postavke kamere „mini mape“



Sl. 4.2.4 Kamera „mini mape“ unutar igre

Okvir mini mape dodan je naknadno kako se slika mini mape ne bi miješala sa slikom kamere igrača. Crvena točka u sredini označava položaj igrača na mapi.

```
4 public class mapFollowC : MonoBehaviour {
5
6     public Transform Target;
7
8
9     void LateUpdate () {
10        transform.position = new Vector3 (Target.position.x, transform.position.y, Target.position.z);
11    }
12 }
```

Sl. 4.2.5 Kod za praćenje igraća kamerom „mini mape“

4.3. Objekti

Svaki objekt [16] u igri se smatra objektom igre („GameObject“). Međutim, objekti igre ne rade ništa sami od sebe. Oni trebaju imati namještene posebne opcije prije nego što postanu okolina, lik, posebni efekt ili slično. Svaki od ovih objekata radi posebne i različite stvari.

Objekti koji su korišteni da bi se napravila ova igra su pretežito 3D modeli kocaka, objekt kamere, objekt igrača te objekt svjetlosti. Sve 3D kocke su iste dimenzije, ali se razlikuju po materijalu koji upotpunjava njihovo značenje.

```
6  public class Block
7  {
8      public int type;
9      public bool vis;
10     public GameObject block;
11
12     public Block(int t, bool v, GameObject b)
13     {
14         type = t;
15         vis = v;
16         block = b;
17     }
18 }
```

Sl. 4.3.1 Kod za određivanje klase kocka („Block“)

```
31     public GameObject grassBlock;
32     public GameObject sandBlock;
33     public GameObject snowBlock;
34     public GameObject cloudBlock;
35     public GameObject diamondBlock;
36     public GameObject dirtBlock;
37     public GameObject goldBlock;
38     public GameObject stoneBlock;
39     public GameObject coalBlock;
40     public GameObject woodBlock;
41     public GameObject leavesBlock;
42     public GameObject invisibleBlock;
```

Sl. 4.3.2 Kod za najavljivanje objekta igre (kocaka)

Ideja za izgled kocaka, odnosno materijal kojim su one obojene dolazi iz stvarnog svijeta. Ti materijali omogućuju da scena stvorenog svijeta izgledala realnije, a to su trava, zemlja, pijesak, snijeg, oblak, kamen, drvo, lišće, ugljen, zlato, dijamant te nevidljiva kocka koja služi za blokiranje prolaza igrača izvan mape. Dimenzija svih kocaka je ista i iznosi 1x1x1.



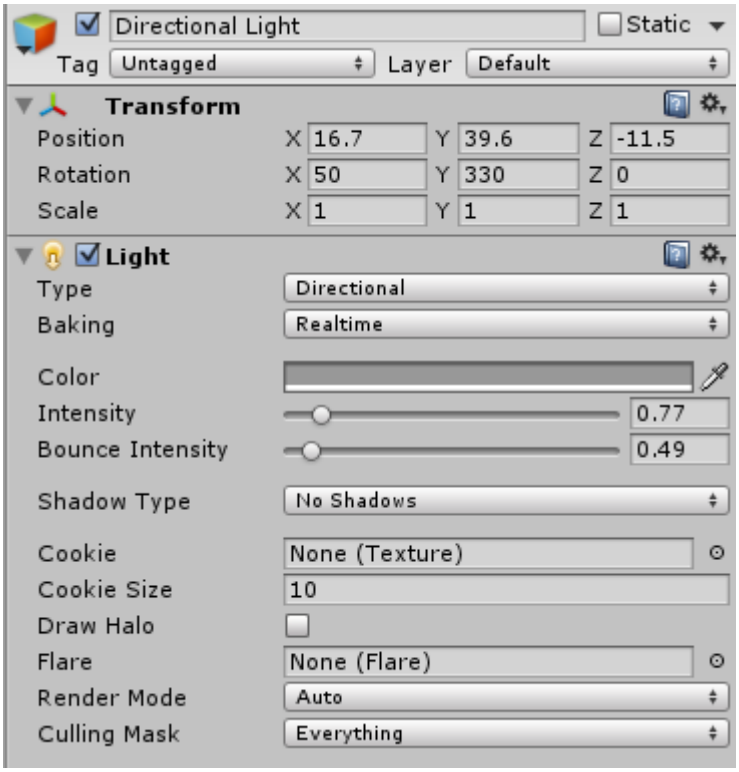
Sl. 4.3.3 Kocke spremljene kao objekti

```
476     void DrawBlock(Vector3 blockPos)
477     {
478         //ako je izvan mape:
479         if (blockPos.x < 0 || blockPos.x > width - 1 ||
480             blockPos.y < 0 || blockPos.y > height - 1 ||
481             blockPos.z < 0 || blockPos.z > depth - 1)
482             return;
483 
484         if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] == null)
485             return;
486 
487         if (!worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].vis) {
488             GameObject newBlock = null;
489             worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].vis = true;
490             if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 1)
491                 newBlock = (GameObject)Instantiate (snowBlock, blockPos, Quaternion.identity);
492             else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 2)
493                 newBlock = (GameObject)Instantiate (grassBlock, blockPos, Quaternion.identity);
494             else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 3)
495                 newBlock = (GameObject)Instantiate (sandBlock, blockPos, Quaternion.identity);
496             else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 5)
497                 newBlock = (GameObject)Instantiate (diamondBlock, blockPos, Quaternion.identity);
498             else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 6)
499                 newBlock = (GameObject)Instantiate (dirtBlock, blockPos, Quaternion.identity);
500             else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 7)
501                 newBlock = (GameObject)Instantiate (goldBlock, blockPos, Quaternion.identity);
502             else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 8)
503                 newBlock = (GameObject)Instantiate (stoneBlock, blockPos, Quaternion.identity);
504             else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 9)
505                 newBlock = (GameObject)Instantiate (coalBlock, blockPos, Quaternion.identity);
506             else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 10)
507                 newBlock = (GameObject)Instantiate (woodBlock, blockPos, Quaternion.identity);
508             else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 11)
509                 newBlock = (GameObject)Instantiate (leavesBlock, blockPos, Quaternion.identity);
510             else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 12)
511                 newBlock = (GameObject)Instantiate (invisibleBlock, blockPos, Quaternion.identity);
512             else
513                 worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].vis = false;
514 
515             if (newBlock != null)
516                 worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].block = newBlock;
517         }
518 
519     }
```

Sl. 4.3.4 Kod za određivanje vrste kocke pri njenom stvaranju

Igrač kao objekt i njegovo kontroliranje bit će objašnjeno u sljedećem dijelu poglavlja.

Osvjetljenje koje je korišteno, zbog svoje jednostavnosti, jedno je od mnogobrojnih prednosti pri korištenju Unity 3D programa. U Unityju osvjetljenje se lagano namjesti uz to da je potrebno odrediti vrstu svijetla, mjesto odakle će svijetliti i smjer prema kojemu će svijetliti. Mijenjanjem opcije jačine i intenziteta možemo dobiti svjetlost različitog doba dana. U igri se koristi jako svijetlo kako bi dobili izgled popodnevnih sati u danu.



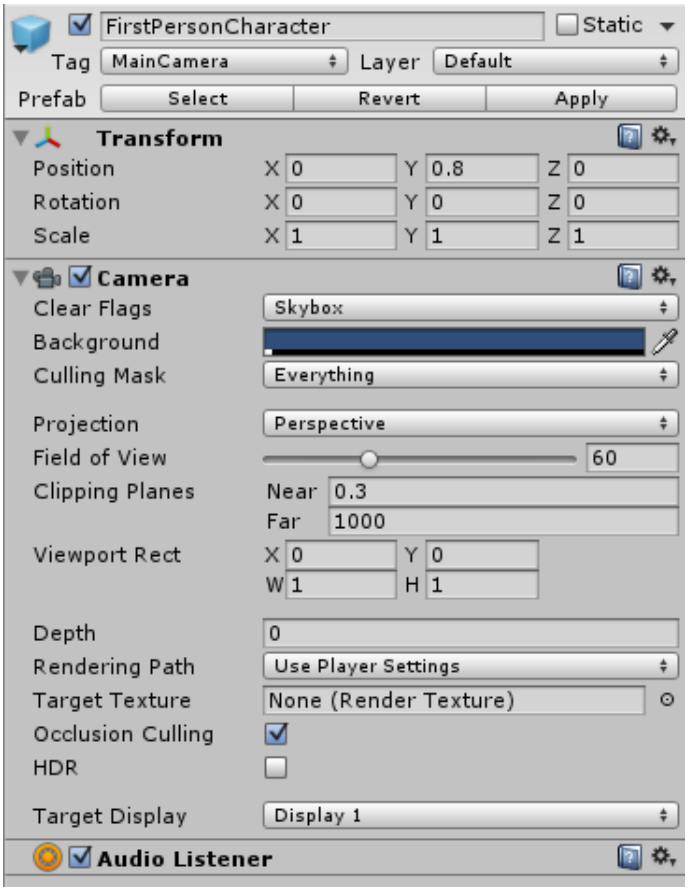
Sl. 4.3.5 Opcije osvjetljenja

4.4. Igrač i kontrole

Igrač je poseban objekt koji ne stoji na sceni samo da bi upotpunjavao okolinu nego ima mogućnost kretanja i izvođenja naredbi. Igra može sadržavati jednog ili više igrača. Da bi osoba koja igra igru lakše kontrolirala objekt igrača, igraču pridodajemo kameru koju spajamo na sami objekt igrača. Ta ga kamera prati i prikazuje scenu iz njegove perspektive.

Igrački objekt je dobiven koristeći već postojeći igrači objekt kojeg nudi sam program Unity. On se sastoji od određenih dimenzija u prostoru, kontrola za lagano i jednostavno kretanje po sceni, spojene kamere koja prikazuje scenu iz prvog lica, namještenih osnovnih opcija, mikrofona koji predstavlja „uši“ igrača te zvuka kojeg igrač proizvodi svojim kretanjem i ponašanjem.

Ponuđeni igrači objekt se sastoji od dva dijela. Prvi dio je „FPSController“ koji omogućuje kretanje, osnovne naredbe i zvukove te drugi dio „FirstPersonCharacter“ koji omogućava mijenjanje kamere igrača i mikrofon kojim može „slušati“ zvukove i prenositi ih na vanjske zvučnike.



Sl. 4.4.1 Izbornik opcija „FirstPersonCharacter“

Naredbe za kretanje i igranje su jednostavne i slične kao u većini igara. Kontrole se mogu pročitati unutar igre klikom na tipku „Instrukcije“ unutar glavnog izbornika, a to su:

Osnovne:

W – Naprijed

A – Lijevo

S – Nazad

D – Desno

Space – Skok

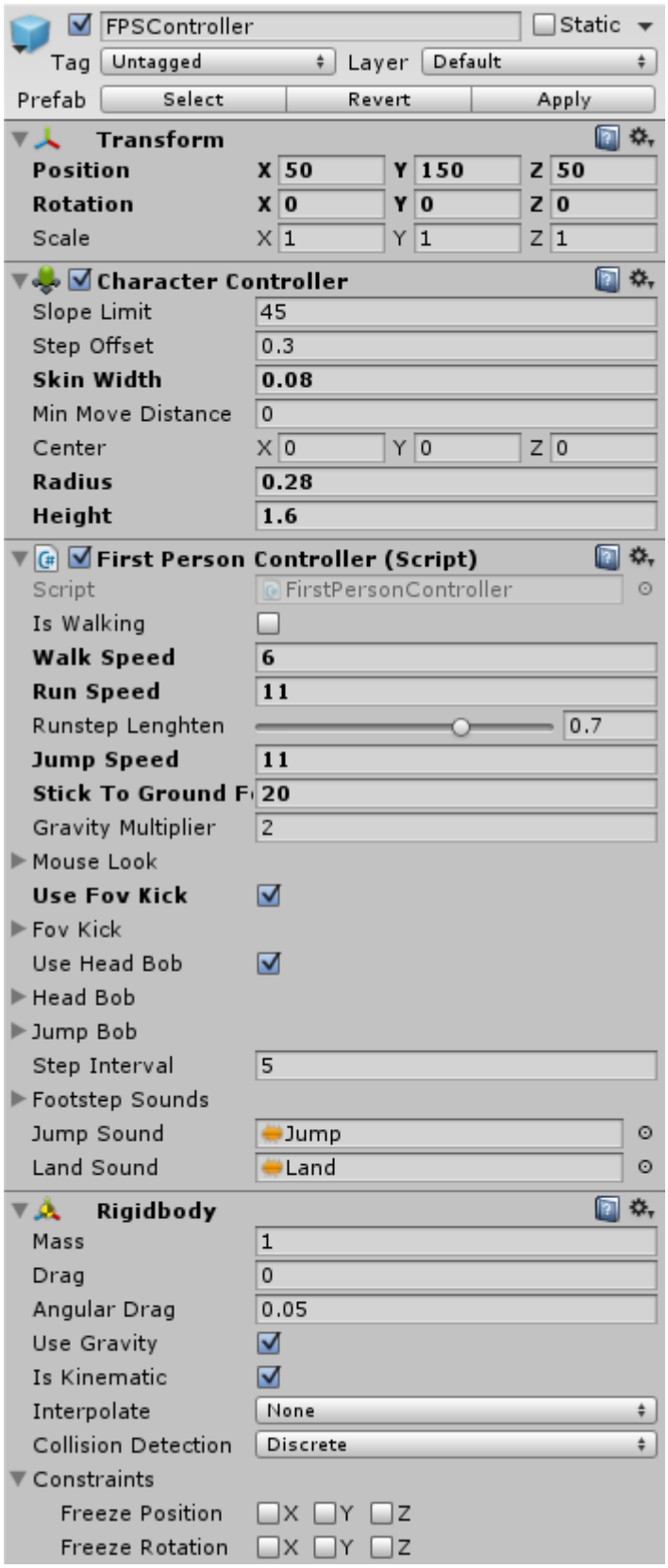
Dodane:

Lijevi klik miša – Uništavanje/ skupljanje kocke

Desni klik miša – Stvaranje kocke

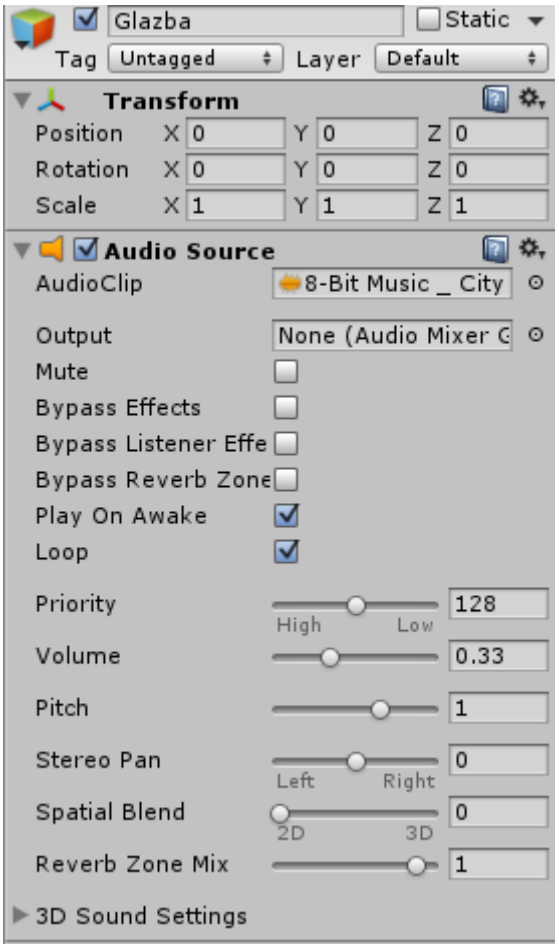
Brojevi od 1 do 7 – Mijenjanje vrste kocaka koju stvaramo desnim klikom miša

Esc – Pauza/ otvaranje izbornika unutar igre



Sl. 4.4.2 Izbornik opcija „FPSController“

Kako bi igra bila zanimljivija na igrački objekt dodana je glazba. Glazba je retro (8-bit) ambijentalnog stila te izvrsno upotpunjava pozadinu igre. U opcijama je namješten loop što znači da ako pjesma dođe do kraja, ponavlja se opet iz početka.



Sl. 4.4.3 Opcije glazbe

Kodovi za dodatne naredbe napisani su unutar funkcije „Update“ koja se poziva svaki frame što bi značilo više puta u sekundi, ovisno o namještenom fps-u. Svaki puta kada se pokrene ta funkcija, zbog napisanog koda, provjerava se ako je korisnik pritisnuo bilo koju od tipki na tipkovnici koja se koristi u igri. U slučaju da je, aktivira se zasebna funkcija koja dalje govori što se treba dogoditi.

```
578         if(Input.GetKey ("1")){
579             atmblock = 1;
580             Rsnow.enabled = true;
581             Rgrass.enabled = false;
582             Rdirt.enabled = false;
583             Rsand.enabled = false;
584             Rstone.enabled = false;
585             Rwood.enabled = false;
586             Rleaves.enabled = false;
587         }
```

Sl. 4.4.4 Kod za tipku „1“

Ovaj kod omogućava korisniku da mijenja vrstu kocke koju želi postaviti u igru. Slika 4.4.4 prikazuje kod za tipku „1“, ali kod je sličan za sve vrste kocaka, uz male promjene.

```
651         if (Input.GetMouseButtonDown (0)) {
652             RaycastHit hit;
653             Ray ray = Camera.main.ScreenPointToRay (new Vector3 (Screen.width / 2.0f, Screen.height / 2.0f, 0));
654             if (Physics.Raycast (ray, out hit, 5.0f)) {
655                 Vector3 blockPos = hit.transform.position;
656                 //ne unistavati naj doljnju kocku---
657                 if ((int)blockPos.y == 0)
658                     return;
659 
660                 //brojaci kocaka za inventorij
661                 if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 1) {
662                     bcsnow = bcsnow + 1;
663                     tbcsnow.text = bcsnow.ToString ();
664                 }
665                 else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 2) {
666                     bcgrass = bcgrass + 1;
667                     tbcgrass.text = bcgrass.ToString ();
668                 }
669                 else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 3) {
670                     bcsand = bcsand + 1;
671                     tbcsand.text = bcsand.ToString ();
672                 }
673                 else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 6) {
674                     bcdirt = bcdirt + 1;
675                     tbcdirt.text = bcdirt.ToString ();
676                 }
677                 else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 8) {
678                     bcstone = bcstone + 1;
679                     tbcstone.text = bcstone.ToString ();
680                 }
681                 else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 10) {
682                     bcwood = bcwood + 1;
683                     tbcwood.text = bcwood.ToString ();
684                 }
685                 else if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].type == 11) {
686                     bcleaves = bcleaves + 1;
687                     tbcleaves.text = bcleaves.ToString ();
688                 }
689 
690 
691 
692                 worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = null;
693                 Destroy (hit.transform.gameObject);
694 
695                 for (int x = -1; x <= 1; x++)
696                     for (int y = -1; y <= 1; y++)
697                         for (int z = -1; z <= 1; z++) {
698                             if (!(x == 0 && y == 0 && z == 0)) {
699                                 Vector3 neighbour = new Vector3 (blockPos.x + x, blockPos.y + y, blockPos.z + z);
700                                 DrawBlock (neighbour);
701                             }
702                         }
```

Sl. 4.4.5 Kod za lijevi klik miša

Ovaj kod omogućava uništavanje kocaka lijevim klikom miša. Uništene se kocke, s obzirom na njihovu vrstu, pridodaju spremniku koji dalje omogućuje njihovo ponovno stvaranje. Zadnji dio ovog koda omogućava stvaranje kocaka koje do prije uništenja određene kocke nisu postojale, odnosno nisu bile vidljive igraču. Ovime sprječavamo nastanak rupa na mapi i propadanjem igrača iz mape. Zbog ove funkcije ne moramo stvarati veliki broj kocaka istovremeno nego samo određeni broj kocaka dovoljan da bi se igrač mogao slobodno kretati. Stvaranje velikog broja kocaka istovremeno moglo bi uzrokovati preopterećenje na slabijim računalima.

```
717         else if (Input.GetMouseButtonDown (1)) {
718 
719                 RaycastHit hit;
720                 Ray ray = Camera.main.ScreenPointToRay (new Vector3 (Screen.width / 2.0f, Screen.height / 2.0f, 0));
721                 if (Physics.Raycast (ray, out hit, 5.0f)) {
722                     Vector3 blockPos = hit.transform.position;
723                     Vector3 hitVector = blockPos - hit.point;
724 
725                     hitVector.x = Mathf.Abs (hitVector.x);
726                     hitVector.y = Mathf.Abs (hitVector.y);
727                     hitVector.z = Mathf.Abs (hitVector.z);
728 
729                     if (hitVector.x > hitVector.z && hitVector.x > hitVector.y)
730                         blockPos.x -= (int)Mathf.RoundToInt (ray.direction.x);
731                     else if (hitVector.y > hitVector.x && hitVector.y > hitVector.z)
732                         blockPos.y -= (int)Mathf.RoundToInt (ray.direction.y);
733                     else
734                         blockPos.z -= (int)Mathf.RoundToInt (ray.direction.z);
735 
736                     CreateBlockPlayer ((int)blockPos.y, blockPos, true);
737 
738 
739                     CheckObscuredNeighbours(blockPos);
740 
741                 }
742 
743         }
```

Sl. 4.4.6 Kod za desni klik miša

Ovaj kod omogućuje igraču stvaranje kocke s obzirom na vrstu kocke odabranu prije stvaranje. Prilikom stvaranja određene kocke, broj te vrste kocaka u spremniku se smanjuje. Ovim kodom sprječavamo beskonačno stvaranje kocaka, odnosno omogućujemo stvaranje samo onih kocaka koje su prethodno bile uništene.

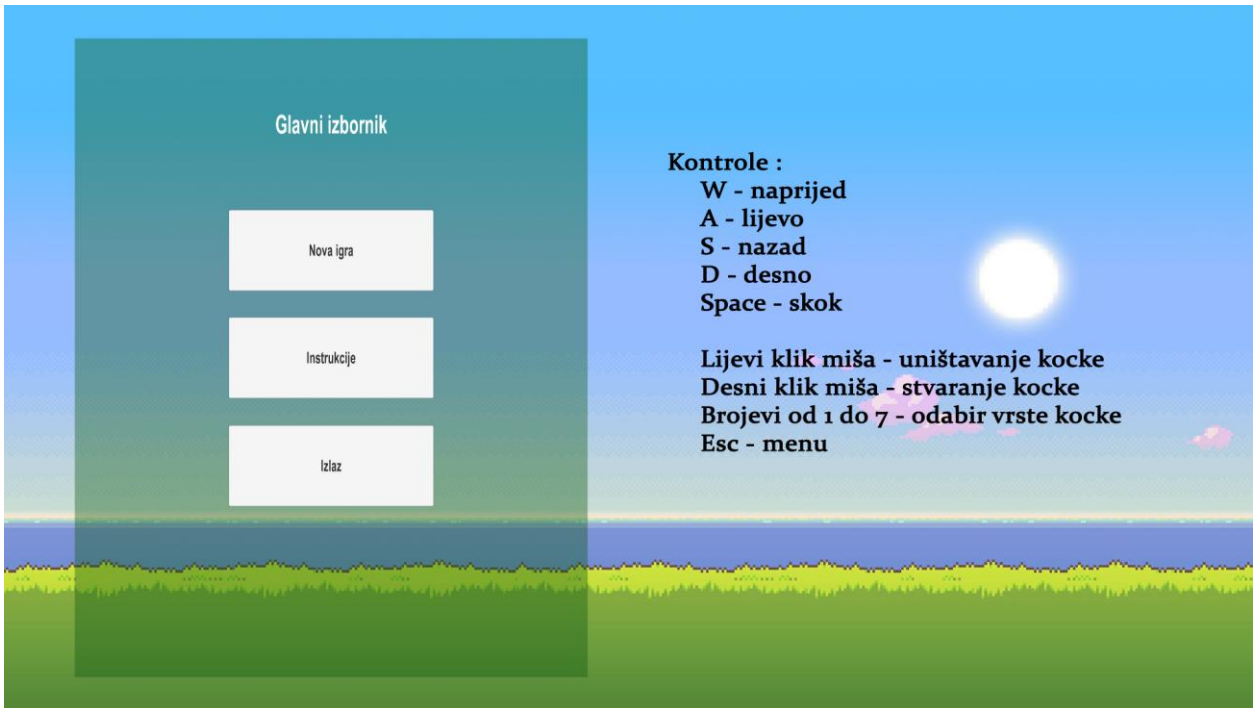
Funkcija CreateBlockPlayer stvara kocku dok funkcija CheckObscuredNeighbours provjerava ako postoji kocka na mapi koja je „zatrpana“ ostalim kockama te ju korisnik ne vidi. Ako takva kocka postoji funkcija ju briše kako ne bi zauzimala prostor.

4.5. Izbornik (menu) i canvas

UI („user interface“) [9] sistem omogućuje stvaranje korisničkog sučelja brzo i intuitivno. Canvas (hr. platno) je područje u kojem se nalaze svi elementi UI sistema, odnosno trebali bi se nalaziti. Canvas je isto jedan od objekata igre sa svim platnenim komponentama koje se nalaze unutar njega. Svi UI elementi moraju biti djeca („children“) tog canvasa. Stvarajući novi UI element, na primjer sliku koristeći izbornik: GameObject > UI > Image, program automatski stvara canvas u izborniku. U slučaju da je canvas već postojao neće se stvoriti još jedan. UI element je stvoren kao dijete tog canvasa.

UI elementi iz canvasa crtaju se istim redom kako su poredani u hijerarhiji. Prvo dijete je nacrtano prvo, sljedeće je nacrtano drugo, i tako do kraja liste. U slučaju da se dva UI elementa preklapaju slikom, onaj koji je na listi u hijerarhiji poslije biti će nacrtan preko onoga koji je prije.

U ovoj igri canvas je različit za svaku scenu. Prva scena, odnosno scena glavnog izbornika, sastoji se od jednostavnog canvasa koji se sastoji se od pozadinske slike, 3 tipke i teksta. Korištenjem canvasa dobivamo željeni izgled glavnom izbornika.



Sl. 4.5.1 Prikaz canvasa unutar scene glavnog izbornika

Tipke koje se nalaze na glavnom izborniku su:

Nova igra – pokreće scenu igre

Instrukcije – pokreće tekst instrukcija igre, odnosno kontrole

Izlaz – zatvara igru

Klikom na tipku Instrukcije otvara se tekst Kontrola kao što je prikazano na slici 4.5.1. Kontrole su objašnjene u prethodnom poglavlju pa nije potrebno ponovo isto pisati.

Druga scena, odnosno scena igre ispunjena je jednostavnim slikama koje pojednostavljuju igranje. To su slike za: pokazivanje vrste kocke koja je trenutno odabrana za korištenje, tekst koji prikazuje brojeve kocaka u spremniku, okvir mini mape i crvena točka koja označava poziciju igrača na mini mapi, pijuk, centar slike koji ujedno označava nišan za uništavanje i stvaranje kocaka te izbornik koji se otvara pritiskom na tipku Esc.

Slika pijuka dodana je kako bi upotpunila prazninu ekrana. Prilikom pritiska lijevog klika na mišu slika pijuka mijenja položaj te daje dojam kopanja kocaka pijukom.



Sl. 4.5.2 Prikaz canvasa unutar igre

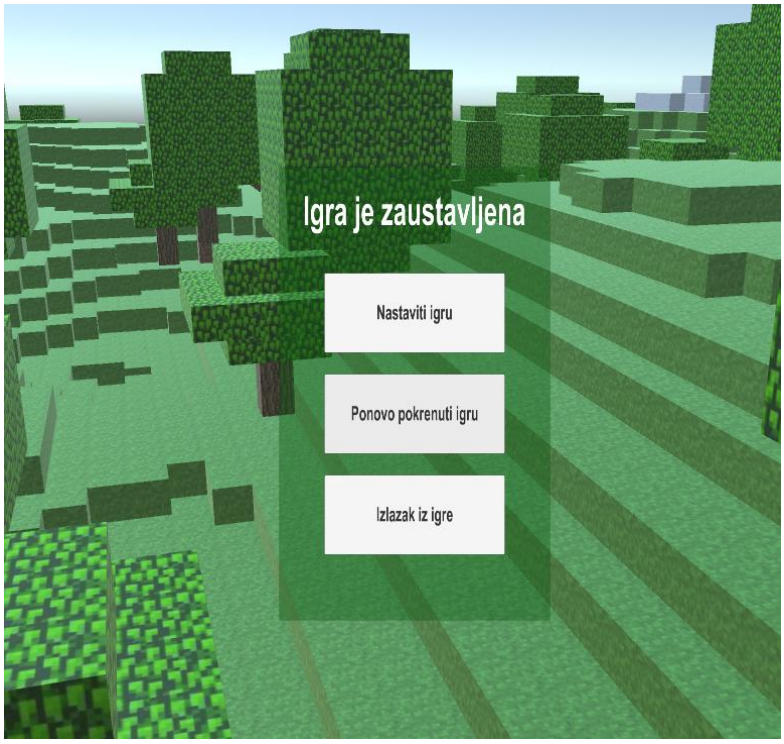
Izbornik unutar druge scene igre otvara se pritiskom na Esc tipku s tipkovnice i sadrži tri tipke na sebi. Ponovim pritiskom na tipku Esc izbornik se zatvara i igra se nastavlja.

Tipke izbornika unutar igre su:

Nastaviti igru – nastavlja se igra kao što je bila prije pritiska tipke

Ponovo pokrenuti igru – igra se ponovno pokreće i stvara se sasvim novi svijet

Izlazak iz igre – zatvara igru



Sl. 4.5.3 Izbornik unutar igre

```
18      void Update () {
19          if (isPaused) {
20              PauseGame (true);
21          } else {
22              PauseGame (false);
23          }
24
25          if (Input.GetButtonDown ("Cancel")) {
26              SwitchPause ();
27          }
28      }
29
30      void PauseGame (bool state){
31          if (state) {//pauzirano
32              Time.timeScale = 0.0f;
33          } else {//odpauirano
34              Time.timeScale = 1.0f;
35          }
36          pausePanel.SetActive (state);
37      }
38
39      public void SwitchPause(){
40          if (isPaused){
41              isPaused = false; //mijenja vrijednost
42          }
43          else {
44              isPaused = true;
45          }
46      }
```

Sl. 4.5.4 Kod za tipku „Nastaviti igru“

```
52      public void Restart(){
53          int scene = SceneManager.GetActiveScene ().buildIndex;
54          SceneManager.LoadScene (scene, LoadSceneMode.Single);
55      }
```

Sl. 4.5.5 Kod za tipku „Ponovo pokreni igru“

```
48      public void Quit(){
49          Application.Quit ();
50      }
```

Sl. 4.5.6 Kod za tipku „Izlazak iz igre“

5. PROCEDURALNO GENERIRANJE SVIJETA

Proceduralno generiranje svijeta znači stvaranje svijeta uz pomoć nekog određenog algoritma svaki puta pri pokretanju igre. Ovaj proces moguće je izvesti na više načina. On služi za uštedu na memoriji tvrdog diska, CD-a ili DVD-a te je baš zbog toga idealna je za igre namijenjene pohrani na vanjsku memoriju. Umjesto stvaranja cijele scene samostalno i spremanja te scene na tvrdi disk, ovom metodom scena ostaje prazna te ona ne zauzima memoriju onoliko koliko bi zauzimala u slučaju da je popunjena objekt po objekt. Prilikom pokretanja igre scena se proceduralno generira po napisanom algoritmu i tek onda počne zauzimati dio memorije.

5.1. PerlinNoise funkcija

Metoda koju je korištena pri izradi ove igre, odnosno proceduralnog generiranja svijeta, temelji se na funkciji Mathf.PerlinNoise [12]. Perlin noise je pseudo-nasumičan uzorak float vrijednosti generiran kroz 2D ravninu. Ova se tehnika može koristiti i na 3D ili nekoj drugoj dimenziji, no to još uvijek nije implementirano u Unity.

Šum (eng. noise) ne sadrži nasumične vrijednosti na svakom dijelu nego sadrži točno određene „valove“ čije se vrijednosti postepeno smanjuju ili rastu kroz cijelu ravninu. Šum se može koristiti za neke osnovne stvari kao što su teksture, animacije, generiranje terena (svijeta), mapa za visinu te još puno drugih stvari.



Sl. 5.1.1 Primjer PearlinNoise raspona od 0 do 10

PerlinNoise kao parametre uzima koordinate x i y, odnosno u ovom slučaju x i z, dok je y visina koja je rezultat ove funkcije. S obzirom na to da je poznata veličina svijeta koji je stvoren, uz pomoć dvije for petlje, parametri x i z jednostavno su iskorišteni da bi PearlinNoise funkcijom dobili vrijednost y (visine). Vrijednost dobivena tom funkcijom pretvorena je iz float u int zato što su veličine kocaka vrijednosti 1x1x1, pa je stoga lakše raditi s cijelim brojevima.

Korištenjem ove funkcije ne dobivamo nasumični rezultat. Taj problem riješen je funkcijom Network.time koja svaki puta pri pokretanju igre uzme trenutno vrijeme i time zapravo svaki puta uzima drugačiji uzorak iz PearlinNoise funkcije. Tim načinom dobiveno je vrlo jednostavno nasumično generiranje brojeva koje se koristi za parametar y.

```
79      void Start () {
80          int seed = (int) Network.time * 10;
81          for (int z = 1; z < depth-1; z++) {
82              for (int x = 1; x < width-1; x++) {
83                  int y = (int)(Mathf.PerlinNoise ((x+seed) / detailScale,
84                      (z+seed) / detailScale) * heightScale) + heightOffset;
85                  Vector3 blockPos = new Vector3 (x, y, z);
86
87                  CreateBlock (y, blockPos, true);
88                  while (y > 0) {
89                      y--;
90                      blockPos = new Vector3 (x, y, z);
91                      CreateBlock (y, blockPos, false);
92                  }
93              }
94          }
```

Sl. 5.1.2 Kod korištenja PearlinNoise funkcije

Ovim kodom se stvaraju površinske kocke u svijetu koje je sačinjeno od brda, dolina i planina. Mijenjanjem vrijednosti varijable detailScale mijenja se gustoća PearlinNoise funkcije te mijenjanjem vrijednosti varijable heightScale mijenja se raspon visine i dubine PearlinNoise funkcije. Mijenjanje vrijednosti varijable heightOffset mijenja se visina cijelog svijeta zajedno.

Nakon određivanja pozicija svake površinske kocke treba napisati algoritam koji će odrediti njihov izgled i vrstu. Naravno, koristeći što veći izbor vrsta kocaka smanjuje se monotonija izgleda igre. Prethodno navedeno, u igri postoji 12 vrsta kocaka. Od navedenih 12, njih 7 mogu biti uništene i ponovo stvorene od strane igrača, dok 3 mogu biti samo uništene. Preostale dvije čine oblaci koji su neuništivi te nevidljive kocke koje su neuništive i služe za sprječavanje igračevog ispadanja iz mape.



Sl. 5.1.3 Primjer stvaranja svijeta uz pomoć PearlinNoise funkcije

Prilikom stvaranja svake kocke parametara x, y i z, algoritam određuje točnu vrstu te kocke. Za početak, najviše kocke na visinama y > (14 + heightOffset) postaju kocke snijega. Zatim kocke na visinama y > (2 + heightOffset) postaju kocke trave ako iznad njih nema ni jedne kocke, u suprotnom ako ima postaju kocke blata. Na istom rasponu visine postoji mala šansa za stvaranjem drveća, čiji će postupak kasnije biti opširnije objašnjen. Na visinama y > (heightOffset - 5) stvaraju se kocke pijeska. Na svim visinama ispod toga stvaraju se kocke kamena uz male mogućnosti stvaranja kocaka ugljena, zlata i dijamanta na njima zasebno određenim rasponima visine odnosno dubine.

5.2. Drveće

Drveće je objekt koji ispunjava scenu te joj daje realniji i prirodniji izgled. Stvaranje drveća jednostavan je algoritam koji puno koristi naredbu Random.Range. Funkcija Random.Range daje jedan nasumičan broj uz pomoć dva parametra. Parametri funkcije x i y označavaju raspon generiranja nasumičnog broja od x do y.

```
if (worldBlocks [(int)blockPos.x, (int)blockPos.y + 1, (int)blockPos.z] == null && y < (12 + heightOffset))
{
    if (create) {
        Vector3 drvo = new Vector3 (blockPos.x, blockPos.y + 1, blockPos.z);
        newBlock = (GameObject)Instantiate (woodBlock, drvo, Quaternion.identity);
    }
    worldBlocks [(int)blockPos.x, (int)blockPos.y+1, (int)blockPos.z] = new Block (10, create, newBlock);

    int j = 1001;
    int c = 1;
    int triger = 0;

    for(int i = 2; i <=9; i++)
    {
        if (i == 9) {
            j = 0;
        }
        if (Random.Range (0, 1000) < j && worldBlocks [(int)blockPos.x, (int)blockPos.y + i - 1, (int)blockPos.z] != null) {
            if (create) {
                Vector3 drvo = new Vector3 (blockPos.x, blockPos.y + i, blockPos.z);
                newBlock = (GameObject)Instantiate (woodBlock, drvo, Quaternion.identity);
            }
            worldBlocks [(int)blockPos.x, (int)blockPos.y + i, (int)blockPos.z] = new Block (10, create, newBlock);
            j = j - 70;
            c = c + 1;
```

Sl. 5.2.1 Kod za stvaranje stabla drveta

Stablo se stvara kocku po kocku te je vjerojatnost stvaranja sljedeće kocke sve manja i manja. Ovim postupkom dobiveno je na raznolikosti visine drveća te je ujedno i ograničena maksimalna visina drveća.

Dok je kod za stvaranje stabla vrlo jednostavan, kod za stvaranje krošnje, odnosno kocaka lišća znatno je kompliciraniji. Kod je kompliciran zato što se radi s kockama te je teško dobiti okrugao oblik krošnje kako ona otprilike izgleda u stvarnosti. Osim izgleda te krošnje bitno je znati da nisu sva stabla jednake visine, nego različite.

```
if (c == 3) {
    for (int x1 = -1; x1 <= 1; x1++)
        for (int z1 = -1; z1 <= 1; z1++) {
            if (!(x1 == 0 && z1 == 0)) {

                //da ne izlaze izvan
                if (blockPos.x + x1 < 0 || blockPos.x + x1 > width - 1 ||
                    blockPos.y + i < 0 || blockPos.y + i > height - 1 ||
                    blockPos.z + z1 < 0 || blockPos.z + z1 > depth - 1)
                    return;

                if (worldBlocks [(int)blockPos.x + x1, (int)blockPos.y + i, (int)blockPos.z + z1] == null) {
                    if (create) {
                        Vector3 lisce = new Vector3 (blockPos.x + x1, blockPos.y + i, blockPos.z + z1);
                        newBlock = (GameObject)Instantiate (leavesBlock, lisce, Quaternion.identity);
                    }
                    worldBlocks [(int)blockPos.x + x1, (int)blockPos.y + i, (int)blockPos.z + z1] = new Block (11, create, newBlock);
                }
            }
        }
}
```

Sl. 5.2.2 Kod za stvaranje prva dva sloja lišća

```
if (c > 3) {
    for (int x1 = -2; x1 <= 2; x1++)
        for (int z1 = -2; z1 <= 2; z1++) {
            if (!(x1 == 0 && z1 == 0)) {

                //da ne izlaze izvan
                if (blockPos.x + x1 < 0 || blockPos.x + x1 > width - 1 ||
                    blockPos.y + i < 0 || blockPos.y + i > height - 1 ||
                    blockPos.z + z1 < 0 || blockPos.z + z1 > depth - 1)
                    return;

                if (worldBlocks [(int)blockPos.x + x1, (int)blockPos.y + i, (int)blockPos.z + z1] == null) {
                    if (create) {
                        Vector3 lisce = new Vector3 (blockPos.x + x1, blockPos.y + i, blockPos.z + z1);
                        newBlock = (GameObject)Instantiate (leavesBlock, lisce, Quaternion.identity);
                    }
                    worldBlocks [(int)blockPos.x + x1, (int)blockPos.y + i, (int)blockPos.z + z1] = new Block (11, create, newBlock);
                }
            }
        }
}
```

Sl. 5.2.3 Kod za stvaranje srednjih slojeva lišća

```
else {
    if (c == 3) {
        if (worldBlocks [(int)blockPos.x, (int)blockPos.y + i, (int)blockPos.z] == null) {
            if (create) {
                Vector3 lisce = new Vector3 (blockPos.x, blockPos.y + i, blockPos.z);
                newBlock = (GameObject)Instantiate (leavesBlock, lisce, Quaternion.identity);
            }
            worldBlocks [(int)blockPos.x, (int)blockPos.y + i, (int)blockPos.z] = new Block (11, create, newBlock);
        }
        triger = 1;

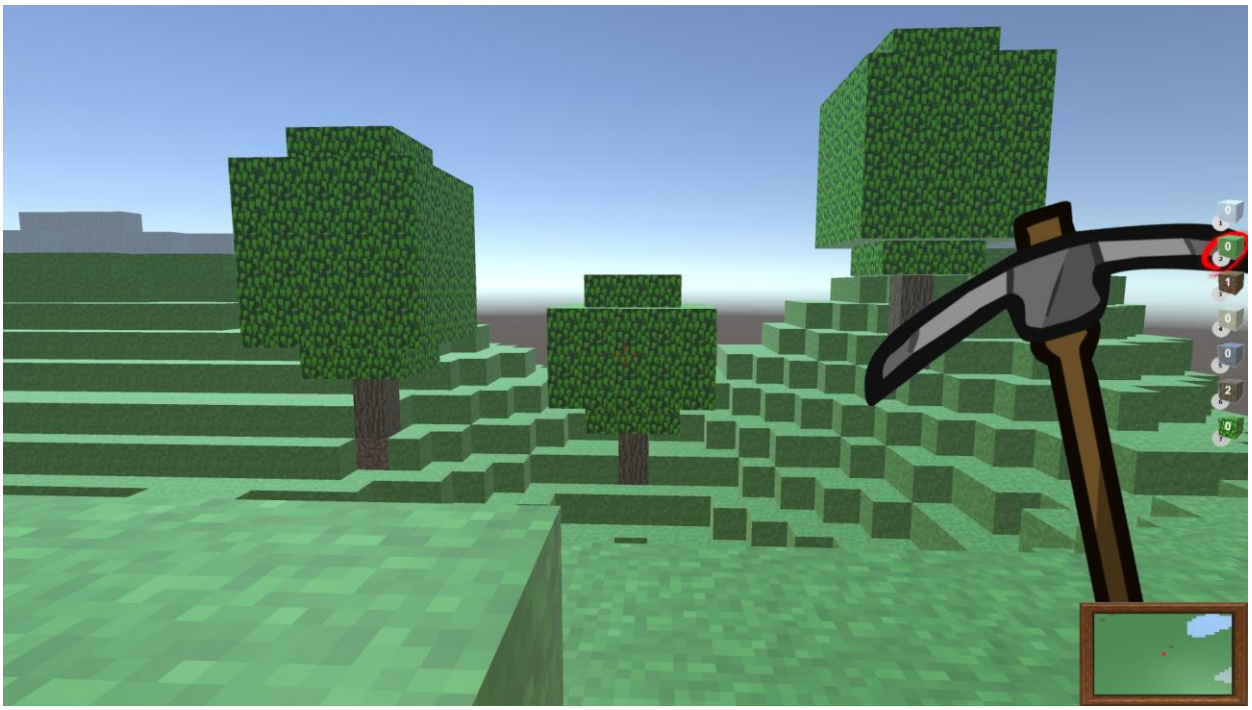
    }
    else if (c > 3 && i < 9) {
        for (int x1 = -1; x1 <= 1; x1++)
            for (int z1 = -1; z1 <= 1; z1++) {

                if (worldBlocks [(int)blockPos.x + x1, (int)blockPos.y + i, (int)blockPos.z + z1] == null) {
                    if (create) {
                        Vector3 lisce = new Vector3 (blockPos.x + x1, blockPos.y + i, blockPos.z + z1);
                        newBlock = (GameObject)Instantiate (leavesBlock, lisce, Quaternion.identity);
                    }
                    worldBlocks [(int)blockPos.x + x1, (int)blockPos.y + i, (int)blockPos.z + z1] = new Block (11, create, newBlock);
                }
            }
        triger = 1;

    }
```

Sl. 5.2.4 Kod za stvaranje zadnjih slojeva lišća

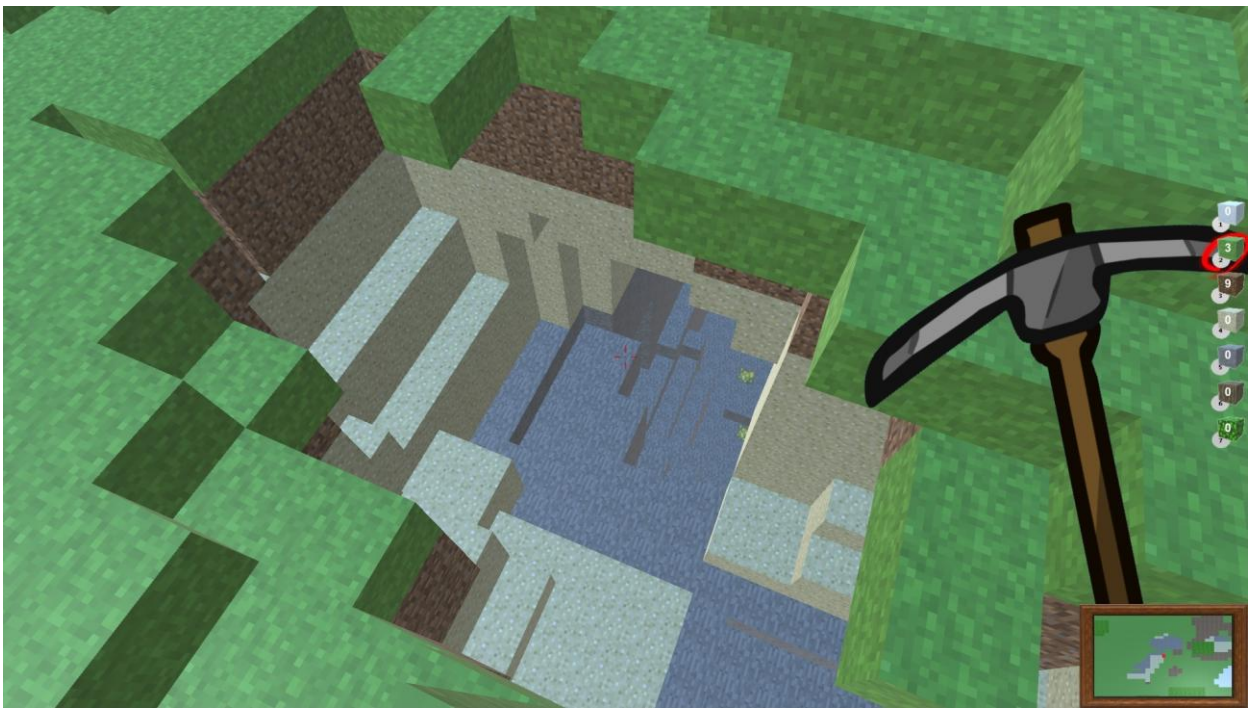
Osim ova 3 koda za stvaranje lišća potrebno je i napisati kod koji će sprječavati stvaranje lišća izvan mape ako se stablo stvorilo blizu ruba mape.



Sl. 5.2.5 Primjer izgled drveća unutar igre

5.3. Rudnici

Rudnici su rupe i tuneli u mapi stvorene nasumično na početku igre. Rudnici služe da igra ne bi bila monotona te da igrač ne mora toliko puno kopati da bi došao do određene dubine.

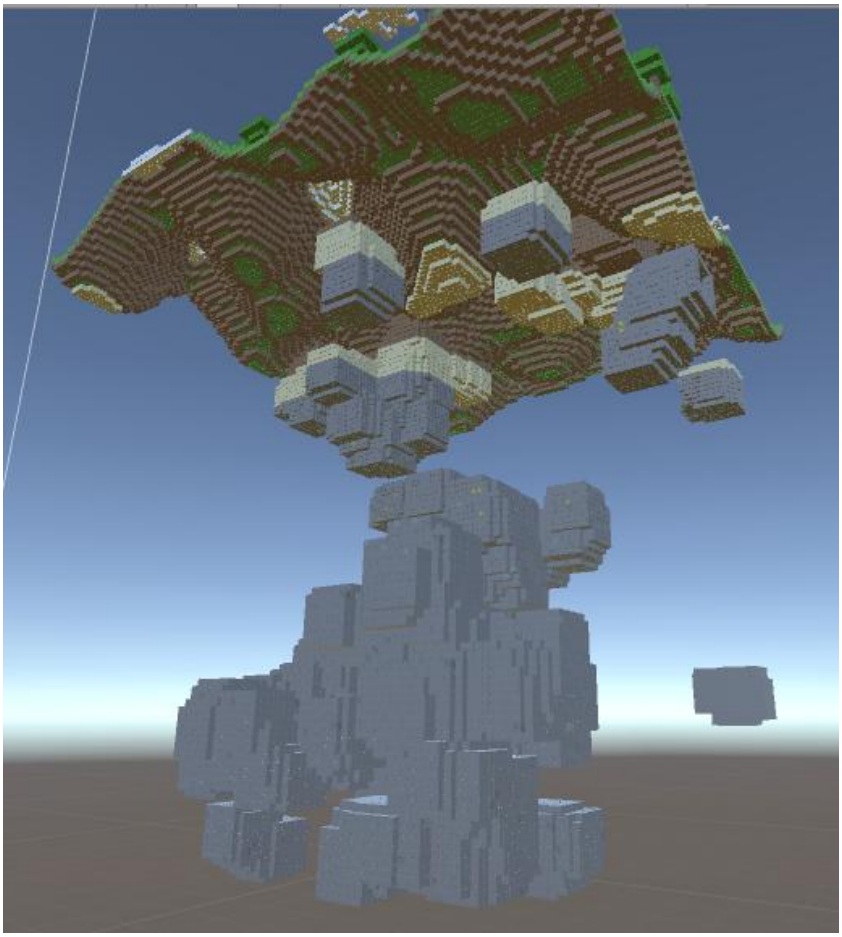


Sl. 5.3.1 Primjer rudnika unutar igre

Rudnici se stvaraju uz pomoć jedne funkcije [14] koju možemo pozvati više puta, ovisno o željenom broju i veličini rudnika. Funkcija za stvaranje rudnika uzima 2 cijela (int) broja kao parametar. Jedan parametar označava veličinu rudnika dok drugi označava broj stvaranja takvih rudnika.

U Igri funkcija se poziva 2 puta. Prvo pozivanje stvara 5 velikih rudnika od 500 kocaka veličine, a drugo stvara 12 manjih rudnika od 15 kocaka veličine.

Funkcija radi tako da nasumično odabire jednu kocku na mapi te ju uništava. Nakon uništenja stvaraju se susjedne kocke oko uništene kocke te funkcija odabire nasumično jednu od susjednih kocaka i uništava ju. Postupak se ponavlja sve dok se ne ispuni broj uništenih kocaka koji je zatražen kao parametar veličine rudnika.



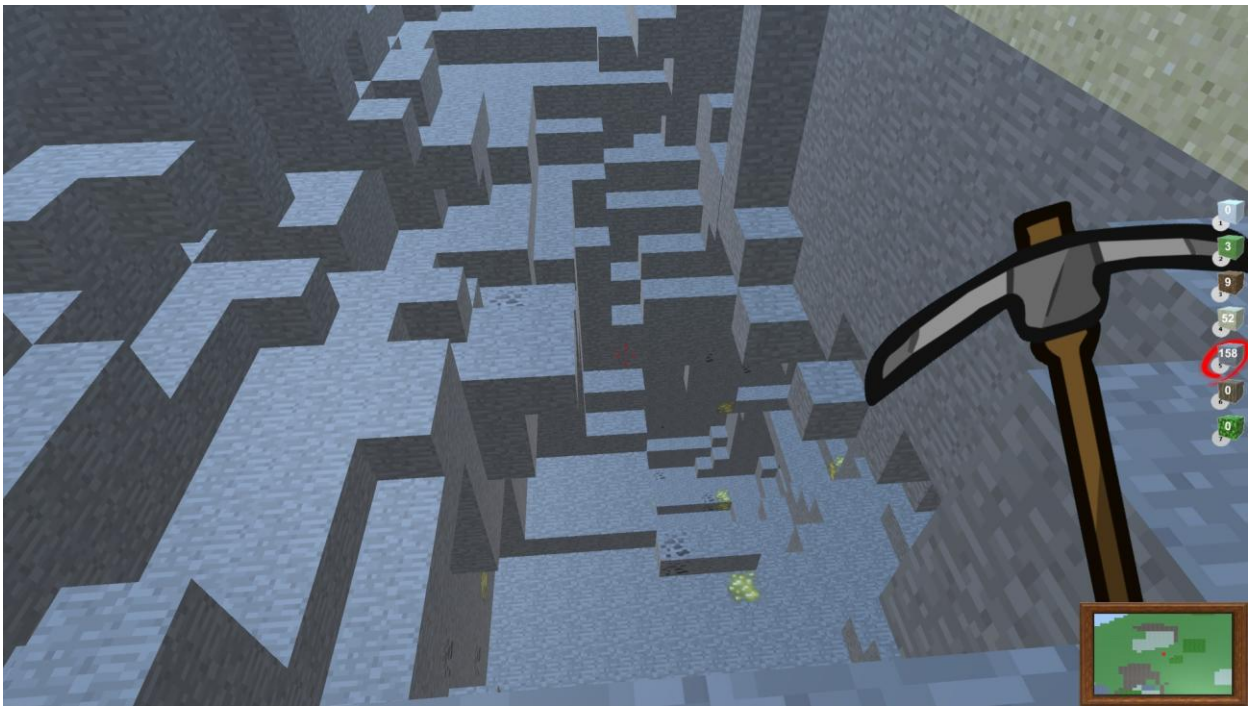
Sl. 5.3.2 Primjer izgleda rudnika pri stvaranju svijeta

```
159     void DigMines(int numMines, int mSize)
160     {
161         int holeSize = 2;
162         for(int i = 0; i < numMines; i++)
163         {
164             int xpos = Random.Range (10, width - 10);
165             int ypos = Random.Range (10, height - 17);
166             int zpos = Random.Range (10, depth - 10);
167             for(int j = 0; j < mSize; j++)
168             {
169                 for(int x = -holeSize; x <= holeSize; x++)
170                     for(int y = -holeSize; y <= holeSize; y++)
171                         for(int z = -holeSize; z <= holeSize; z++)
172                         {
173                             if(!(x == 0 && y == 0 && z == 0))
174                             {
175                                 Vector3 blockPos = new Vector3(xpos+x, ypos+y, zpos+z);
176                                 if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] != null)
177                                 if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].block != null)
178                                     Destroy (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z].block);
179                                 worldBlocks[(int)blockPos.x,(int)blockPos.y,(int)blockPos.z] = null;
180                             }
181                         }
182 
183                 xpos += Random.Range (-1, 2);
184                 ypos += Random.Range (-1, 2);
185                 zpos += Random.Range (-1, 2);
186                 if (xpos < holeSize || xpos >= width - holeSize)
187                     xpos = width / 2;
188                 if (ypos < holeSize || ypos >= height - holeSize)
189                     ypos = height / 2;
190                 if (zpos < holeSize || zpos >= depth - holeSize)
191                     zpos = depth / 2;
192             }
193         }
```

Sl. 5.3.3 Kod za stvaranje rudnika

```
195         for(int z = 1; z < depth-1; z++)
196         {
197             for(int x = 1; x < width-1; x++)
198             {
199                 for(int y = 1; y < height-1; y++)
200                 {
201                     if(worldBlocks[x,y,z] == null)
202                     {
203                         for(int x1 = -1; x1 <=1; x1++)
204                             for(int y1 = -1; y1 <=1; y1++)
205                                 for(int z1 = -1; z1 <= 1; z1++)
206                                 {
207                                     if(!(x1 == 0 && y1 == 0 && z1 == 0))
208                                     {
209                                         Vector3 neighbour = new Vector3 (x + x1, y + y1, z + z1);
210                                         DrawBlock(neighbour);
211                                     }
212                                 }
213                     }
214                 }
215             }
```

Sl. 5.3.4 Kod za stvaranje susjednih kocaka



Sl. 5.3.5 Primjer rudnika iznutra

5.4. Oblaci i nevidljivi zidovi

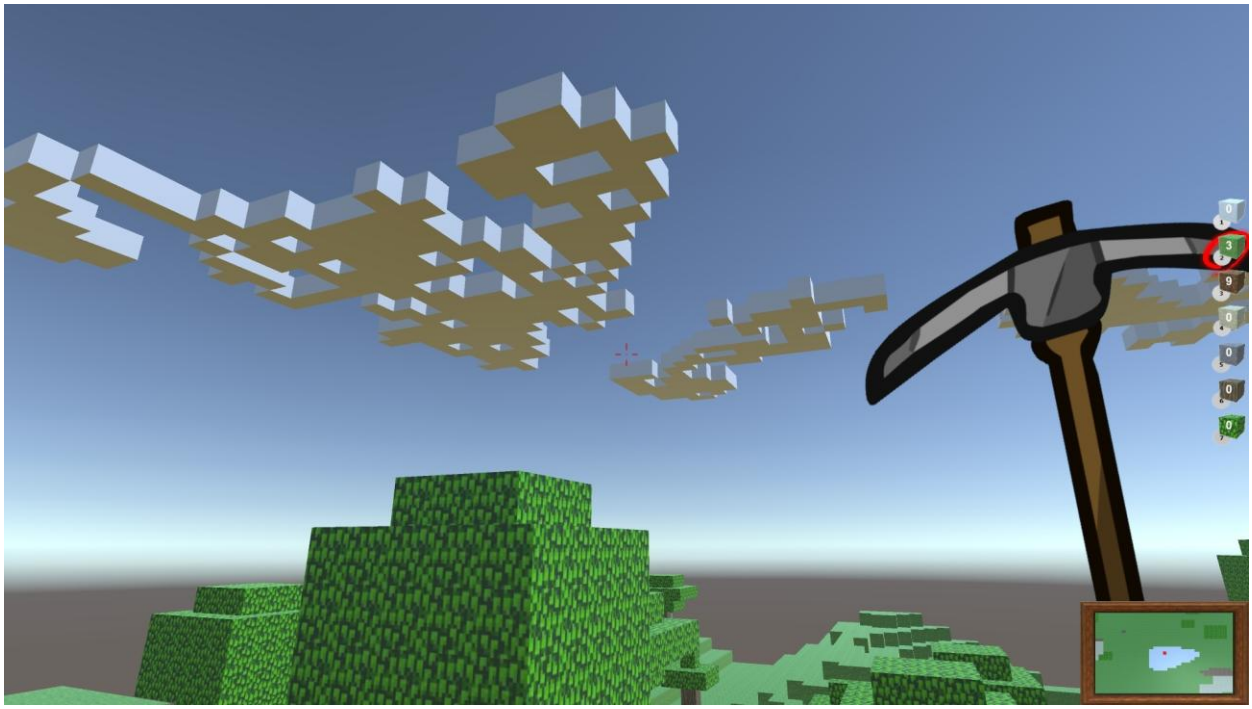
Obje vrsta kocaka korištena za oblake i za nevidljive zidove su neuništive. Razlika je što su oblaci vidljivi i kroz njih se može prolaziti dok su zidovi nevidljivi te se kroz njih ne može prolaziti.

Funkcija za stvaranje oblaka [14] radi na sličnom principu kao i funkcija za stvaranje rudnika. Ima 2 parametra, jedan označava veličinu oblaka, a drugi broj takvih oblaka. Funkcija odabire jednu nasumičnu kocku na točno određenoj visini i stvori kocku vrste oblak. Nakon stvaranja kocke oblaka, funkcija uzima nasumično jednu od susjednih kocaka na istoj visini zatim tamo stvara još jednu kocku vrste oblak. Funkcija se tako stalno ponavlja dok se ne ispuni broj kocaka zatražen unošenjem parametara funkcije.

```
114     void DrawClouds(int numClouds, int cSize)
115     {
116         for(int i = 0; i < numClouds; i++)
117         {
118             int xpos = Random.Range (0, width);
119             int zpos = Random.Range (0, depth);
120             for(int j = 0; j < cSize; j++)
121             {
122                 Vector3 blockPos = new Vector3(xpos, height-1, zpos);
123                 GameObject newBlock = (GameObject) Instantiate(cloudBlock, blockPos, Quaternion.identity);
124                 worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = new Block (4, true, newBlock);
125                 xpos += Random.Range (-1, 2);
126                 zpos += Random.Range (-1, 2);
127                 if (xpos < 0 || xpos >= width) xpos = width / 2;
128                 if (zpos < 0 || zpos >= depth) zpos = depth / 2;
129             }
130         }
131     }
```

Sl. 5.4.1 Kod za stvaranje oblaka

U igri ova funkcija poziva se samo jednom s parametrima 10 oblaka veličine 160 kocaka.



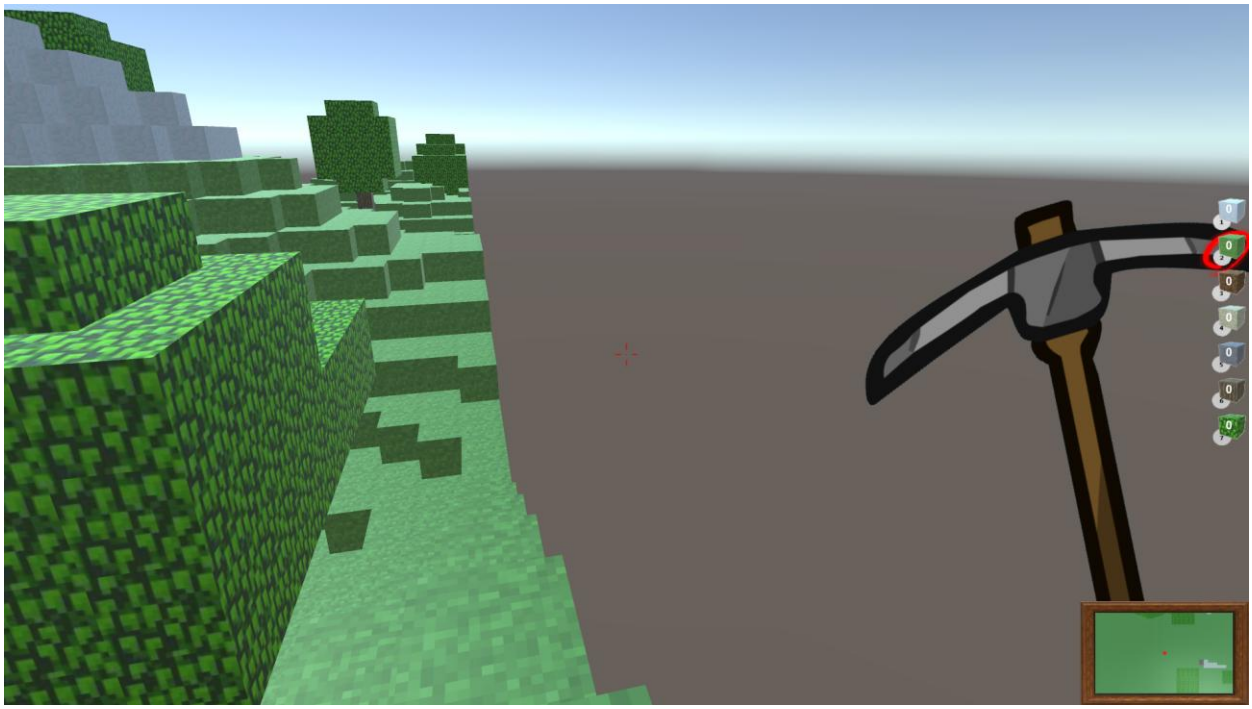
Sl. 5.4.2 Primjer oblaka unutar igre

Funkcija za stvaranje nevidljivih zidova vrlo je jednostavna zato što je dimenzije mape već poznata. Funkcija stvara niz nevidljivih kocaka od maksimalne dubine do maksimalne visine okolo same mape. Koristi se da bi spriječila ispadanje igrača iz mape u „rupu bez dna“.

```
134     void DrawWalls()
135     {
136         for (int z = 0; z < depth; z++) {
137         for (int x = 0; x < width; x++) {
138                 for (int y = 0; y < height; y++) {
139                     Vector3 blockPos = new Vector3 (x, y, z);
140                     if (worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] == null) {
141                         if (z == 0 || z == depth-1) {
142                             GameObject newBlock = (GameObject)Instantiate (invisibleBlock, blockPos, Quaternion.identity);
143                             worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = new Block (12, true, newBlock);
144                         }
145                             else if (x == 0 || x == width-1) {
146                             GameObject newBlock = (GameObject)Instantiate (invisibleBlock, blockPos, Quaternion.identity);
147                             worldBlocks [(int)blockPos.x, (int)blockPos.y, (int)blockPos.z] = new Block (12, true, newBlock);
148                         }
149                     }
150                 }
151             }
152         }
153     }
```

Sl. 5.4.3 Kod za stvaranje nevidljivih zidova

Za razliku od funkcija rudnika i oblaka koje su nasumične, ova funkcija poziva se samo jednom te ima točno određenu funkciju. Na početku igre stvara se nevidljivi zid koji se ne može promijeniti tijekom igre.



Sl. 5.4.4 Slika ruba mape i nevidljivog zida

6. ZAKLJUČAK

Unity je vrlo jednostavan i zanimljiv program za stvaranje igara, ali i za programiranje općenito. Unityeva stranica daje velik izbor informacija o korištenju samog programa zapisane u online priručniku te puno videa koji pobliže objašnjavaju postupke programiranja ili neke teže razumljive funkcije. Ovakva igra nije lagana za napraviti i programirati ako ste početnik, ali uz pomoć svih online priručnika i videa može se napraviti. Igra je namijenjena za opuštanje, lagana je i nije prezahtjevna. Način na koji se stvara svijet u igri, a koji je ujedno i tema ovog zadatka, može se koristiti i u drugim igrama koje zahtijevaju nasumično proceduralno generiranje svijeta prilikom pokretanja igre. Igru se može bezbroj puta ponovo pokrenuti i svijet će svaki puta biti drugačiji, što igranje ove igre čini još zanimljivijim.

LITERATURA

[1] Unity online stranica, https://unity3d.com/ 25.4.2016.

[2] Unity online video zapisi, https://unity3d.com/learn/tutorials 26.4.2016.

[3] Unityev online priručnik, http://docs.unity3d.com/Manual/index.html 1.5.2016.

[4] Unityev priručnik o grafici, http://docs.unity3d.com/Manual/Graphics.html 21.5.2016.

[5] Unityev priručnik o osvjetljenju, http://docs.unity3d.com/Manual/LightingOverview.html 21.5.2016.

[6] Unityev priručnik o kamerama, http://docs.unity3d.com/Manual/CamerasOverview.html 22.5.2016.

[7] Unityev priručnik o stvaranju terena, http://docs.unity3d.com/Manual/script-Terrain.html 22.5.2016.

[8] Unityev priručnik o fizici, http://docs.unity3d.com/Manual/PhysicsSection.html 23.5.2016.

[9] Untyev priručnik o canvasu, http://docs.unity3d.com/Manual/UICanvas.html 23.5.2016.

[10] Wikipedia, Programski jezik C#, https://en.wikipedia.org/wiki/C_Sharp_(programming_language) 30.5.2016.

[11] Wikipedia, C# sintaksa, https://en.wikipedia.org/wiki/C_Sharp_syntax 30.5.2016.

[12] Unityev izbornik o PearlinNoise funkciji, https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html 2.5.2016.

[13] Holistic3d, Video lekcija izrade ovakve igre, https://www.youtube.com/watch?v=4hVAzRahcdY 1.5.2016.

[14] Holistic3d, video lekcija izrade oblaka i rudnika, https://www.youtube.com/watch?v=4Qp9nqVi5KQ 1.5.2016.

[15] Google pretraživač slika, https://www.google.hr/search?q=materials 4.5.2016.

[16] Unityev priručnik o objektima, http://docs.unity3d.com/Manual/GameObjects.html 1.6.2016.

[17] Wikipedia, Unity 3D, https://en.wikipedia.org/wiki/Unity_(game_engine) 10.6.2016.

SAŽETAK

Unity 3D je jednostavan i zanimljiv program za stvaranje igara i sličnog sadržaja. Dok se nasumično proceduralno generiranje svijeta može postići na više načina, način koji je korišten omogućen je uz pomoć par funkcija od kojih je najbitnija funkcija PearlinNoise. Proceduralno generiranje svijeta koristi se zbog uštede memorije, dok nasumično stvaranje svijeta razbija samu monotoniju igre. Svijet se u igri postupno stvara korak po korak prilikom pokretanja igre te igrač kreče igrati nakon što se stvori najviši sloj terena. Postupak kojim se stvara svijet u ovoj igri može se primijeniti i na druge slične igre, naravno, uz male promjene.

Ključne riječi: Unity 3D, proceduralno generiranje, pearlinNoise, ušteda memorija

ABSTRACT

Unity 3D is a simple and interesting program for creating games and similar content. While random procedurally generated world can be achieved in several ways, the way that is used was managed with the help of a couple of functions, of which the most important function is PearlinNoise. Procedural generation of world is used to save memory, while the random creation of the world itself breaks the monotony of the game. World in the game is gradually created step by step when starting the game and the player starts to play after the highest layer of the terrain is created. A process that creates the world in this game can be applied to other similar games, of course, with small changes.

Key words: Unity 3D, procedural generation, pearlinNoise, memory saveing

ŽIVOTOPIS

Vedran Brazdil rođen je 01. Prosinca 1994. godine u Osijeku. Živi u Osijeku te se u istom mjestu obrazuje. Pohađao je Osnovnu školu Ljudevita Gaja u Osijeku od 2001. do 2009. Zatim nastavlja svoje obrazovanje u srednjoj Elektrotehničkoj i prometnoj školi Osijek, četverogodišnjeg smjera elektrotehničar. Završetkom srednje škole, uspješno upisuje preddiplomski sveučilišni studij računarstva 2013. godine te sve ispite redovno polaže.

Potpis:

_____