Izrada klona videoigre Minecraft

Mušica, Marko

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marko Mušica

Izrada klona videoigre Minecraft

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marko Mušica

JMBAG: 0016142667

Studij: Informacijski sustavi

Izrada klona videoigre Minecraft

ZAVRŠNI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Mario Konecki

Varaždin, kolovoz 2022.

Marko Mušica

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovome radu bit će opisana izrada Klona igre Minecraft, koja pripada sandbox žanru video igara. „Minecraft“ je najpoznatija sandbox igra danas, a i zadnjih 10 godina pa će se ovaj rad opisati izradu klona u programskom alatu Unity. Prvo će biti opisani alati pomoću kojih je igra napravljena, a onda opis Minecrafta i sličnih igara istog žanra. Nakon toga slijedi opis izrade praktičkog djela koji će biti podijeljen na opise glavnih funkcionalnosti. Glavne funkcionalnosti bit će detaljno opisane i objašnjene uz primjer kodova za ostvarivanje istih.

Ključne riječi: sandbox igre, programiranje , Unity, računalne igre, algoritmi, generacija svijeta, Minecraft

Sadržaj

1. Uvod

S razvojem računalnih komponenata dolaze sve naprednije video igre poput „Red Dead Redemption 2“, „Sea of Thieves“, „Stray“, „Elden Ring“ i mnoge druge. Iz dana u dana izrađuju se igre s boljima grafikama, kompliciranijim pričama, većim svjetovima, a među najpopularnijim igrama još uvijek ostaje „Minecraft“. Mislim da je ova igra najbolji primjer kako ljudi mogu uživati u jednostavnim igrama gdje mogu izraziti svoju kreativnost.

Vrlo je važno objasniti što su točno glavne funkcionalnost Minecrafta i njegovog žanra kako bi se mogle shvatiti glavne funkcionalnosti koje bi klon trebao imati. Gore spomenute igre pripadaju različitim žanrovima. Npr. „Red Dead Redemption 2“ je pucačina otvorenog svijeta. „Sea of Thieves“ spada u igre istraživanja otvorenog svijeta, „Elden Ring“ je isto tako igrica u kojoj igrač može istraživati otvoreni svijet i poraziti različite neprijatelje u njemu. Za razliku od ovih igra, „Minecraft“ nema čitav svijet detaljno izrađen od strane cijelih timova za „map design“, njegov svijet kreira algoritam. Algoritam koji samo stvara nove terene i objekte ovisno o kretanju igrača u svijetu. No, sve ove igre i „Minecraft“ imaju isti cilj. Omogućiti igraču kontroliranje svojeg lika koji će istraživati novi svijet i u slučaju Minecrafta omogućiti igraču izražavanje svoje kreativnosti izgradnjom različitih objekata.

Možemo vidjeti da su u ovim igrama jako bitni koncepti otvorenog svijeta za istraživanje, mogućnost kontroliranje lika kojim će se svijet istražiti, te najbitnije mogućnost kreiranja svojih građevina i izražavanja svoje kreativnosti. U ovom radu bit će opisane neke od najpopularnijih igara koje su ovo ostvarile najbolje. Također, bit će opisana kreacija klona koji sadržava sve ove bitne koncepte, kratko objašnjenje korištenih alata i primjeri koda kojim ostvarili ranije spomenuti koncepti.

2. Metode i tehnike rada

Glavni alat za izradu igre je *Unity game engine*. Pomoćni alati su Visual Studio za razvoj koda u programskom jeziku C#, te različiti dodaci u *Unity game engine* alatu poput *Sprite editing tool-a* koji omogućuje uređivanje ikona koje su se koristile za prikaz nekih objekata u korisničkom sučelju.

2.1. Unity

Unity game engine, osnovan od poduzeća Unity Technologies, dobro je poznat kao jedan od vrhunskih pokretača igara. Zbog laganog korištenja i primamljivog dizajna, Unity je ubrzo postao jedan od najpopularniji game enginea na tržištu (Dave, 2022). Unity uključuje AR, VR, 2D i 3D alate za kreaciju igre koje game developeri mogu implementirati na više platformi, kao što su mobilni uređaji, računala, konzole i čak web igre (Dave, 2022).

Osim game enginea, Unity je i IDE – „integrated development environment“. Što znači da je sučelje koje daje pristup svim alatima potrebnim za game development. Unity software ima editor koji omogućuje game developerima jednostavno korištenja objekata i elemenata pomoću drag&drop funkcionalnosti (Sinicki, 2021). Ovim objektima mogu se mijenjati postavke i svojstva, a primjer toga bit će opisan kasnije u radu.

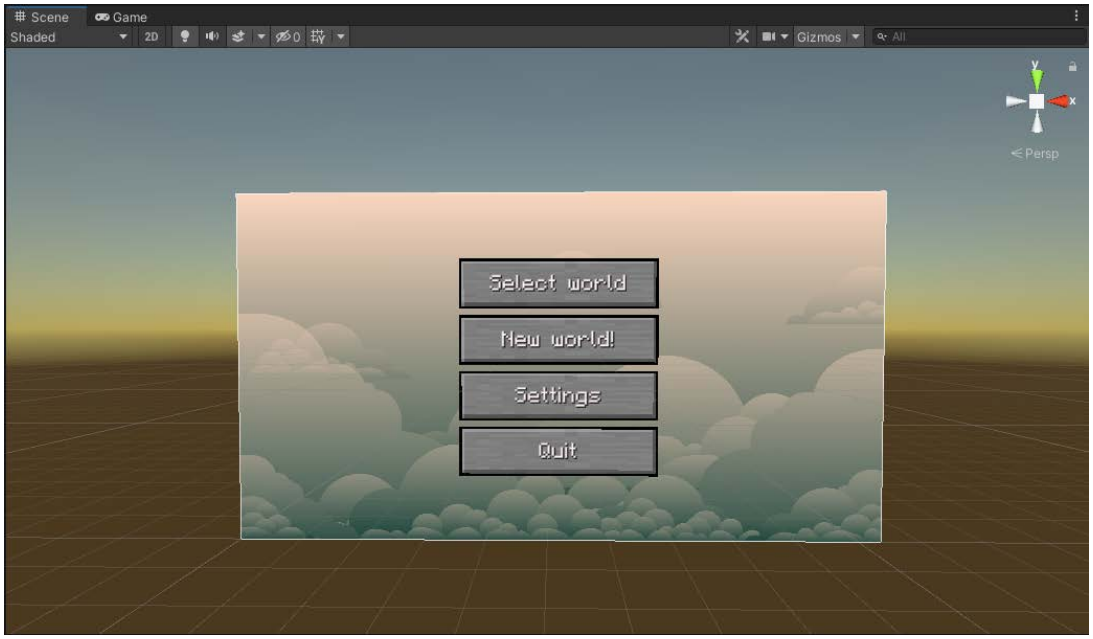
Najveća prednosti koju Unity ima nad drugim game engine alatima je lakoća pristupa i korištenja. Naime, Unity je besplatan game engine koji mogu koristiti svi studenti ili početnici u game developmentu. Velika prednost je ta što se Unity u pozadini pobrine za sve kompleksne izračune i fiziku izrade igre u 3D ili 2D-u (Dealessandri, 16).

Veliki nedostatak Unity-a je neprikladnost za velike projekte. Ne omogućuje detaljniju razradu nekih njegovih značajki što je jako dobro za početnike u game developmentu, ali za iskusnije programere koji se žele baviti složenijim projektima koji moraju biti optimizirani nije toliko dobar. Ovaj nedostatak znatno je vidljiv u igrama otvorenog svijeta kod kojih se moraju provoditi velike optimizacije. (Dealessandri, 16)

2.1.1. Unity korisničko sučelje

2.1.1.1. Scene prozor

Otvaranjem Unitya dobivamo početni prozor koji uz druge bitne informacije prikazuje i scenu. Na ovoj sceni možemo vidjeti izgled naše igre i postavljati te mijenjati kreirane objekte.



Slika 1. Unity - Scena

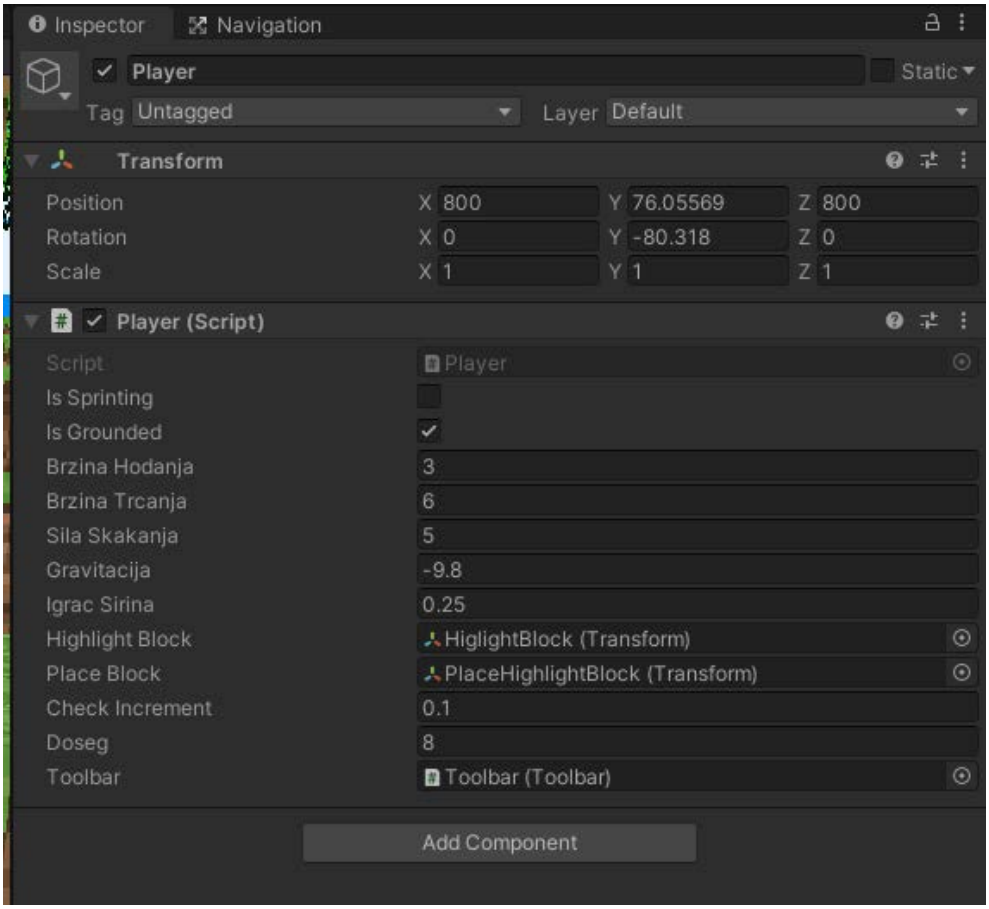
U gornjem desnom kutu možemo vidjeti „Scene Gizmo“ koji pokazuje orijentaciju kamere te omogućuje mijenjanje kuta iz kojeg gledamo scenu. Imamo X, Y, Z smjerove na koje možemo kliknuti pa se ovisno o njima mijenja naš pogled na scenu. Alati u lijevom kutu mijenjaju osvjetljenje, prikaz iz 2D-a u 3D, prikaz mreže itd.

2.1.1.2. Hijerarhija

U Unity-u, hijerarhija sadržava sve objekte koji su u sceni. Hijerarhija može sadržavati više scena i svaka scena će sadržavati svoje posebne objekte. Unity koristi koncept roditelj-dijete kod kojeg jedan objekt (roditelj) može sadržavati druge objekte (djecu) koji nasljeđuju njegova svojstva. Npr. poziciju, rotaciju i skalu. Ovime Unity omogućuje rad nad više objekata istovremeno bez potrebe mijenjanja svakog specifičnog objekta posebno.

2.1.1.3. Inspektor

Alat u kojemu se mogu pregledati i mijenjati sve postavke Unity objekata, materijala i različitih „asseta“. Unutar alata možemo dodavati i pregledavati komponente. Jedna od osnovnih komponenata je „Transform“ kojim možemo mijenjati poziciju, rotaciju i skalu objekata. Još jedna jako bitna komponenta koja se dodaje objektima je skripta. Skripta je kod napisan od programera koji utječe na ponašanje objekta u igri (Unity Technologies, 2022).



Slika 2. Inspektor Igrača

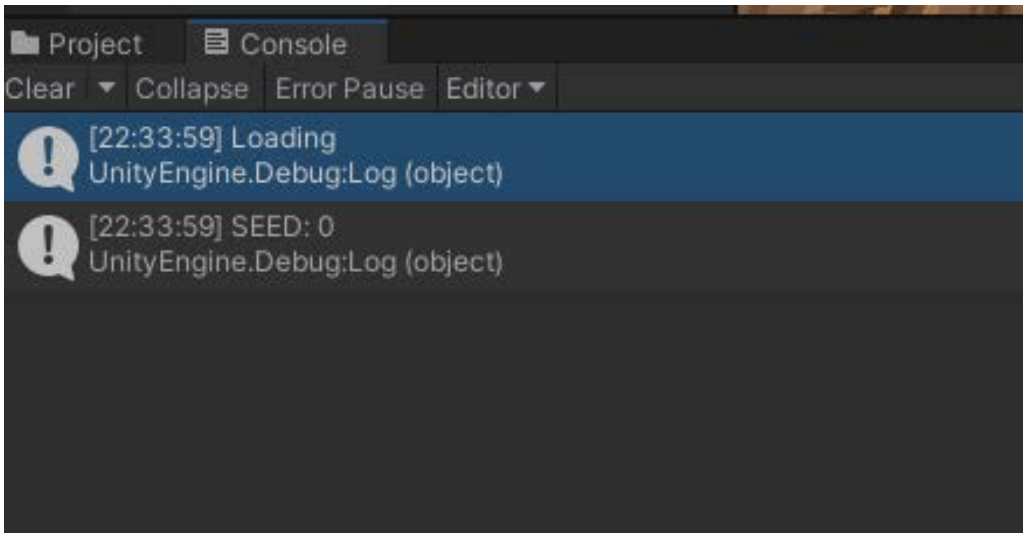
U priloženoj slici možemo vidjeti skriptu i „Transform“ komponentu za igrača u Minecraft klonu. Skripta sadržava različite atribute koji će biti jasnije objašnjeni u kasnijim poglavljima, ali možemo vidjeti da utječe na ponašanje objekta igrača. Npr. atribut „Brzina Trcanja“ utječe na brzinu igrača u svijetu itd.

2.1.1.4. Projektni prozor

Ovaj prozor nam prikazuje sve assete i skripte koje smo kreirali u projektu. Lijevi panel prozora pokazuje struktura mapa u projektu kao hijerarhijsku listu. Kod odabira neke mape, Unity pokaže sve njegove sadržaje na desnom panelu. Ovo su individualni asseti, npr. materijali, skripte ili slično. (Unity Technologies, 2022)

2.1.1.5. Konzola

Jako jednostavan i dosta važan alat u Unity-u koji nam prikazuje poruke, upozorenja i greške. Svaka pojedina poruka može se onesposobiti i bitno je reći da se u konzolu može pisati u našim skriptama koristeći funkciju *Debug.Log().*



Slika 3. Prikaz konzole u Unity-u

2.1.1.6. Game prozor

Game prozor nam prikazuje perspektivu kamere u našoj sceni kada pokrenemo igru. Dosta je bitan za rana testiranja igre pa se jako često koristi. Konzola i game prozor su alati koji se najčešće koriste za testiranje rada igrice prema ranije postavljenim očekivanjima.

2.2. Visual studio

Najbolji sveobuhvatni IDE za .NET i C++ programere na Windowsima. Integrirano je razvojno okruženje za pisanje računalnih programa, web aplikacija i web usluga. Uređivač za kod i program za ispravljanje grešaka (Hope, 2019).

Osim .NET (C#, F#) i C++ jezika još podržava Fossil, M, HTML, XHTML, CSS, JavaScript, Python. Novim alatom IntelliCode automatski nadopunjuje kod prema ranije napisanim linijama. Zbog toga odjednom može dovršiti cijeli redak s imenima varijabli, imenima funkcija i nekom programskom logikom što pomaže programeru s bržim i sigurnijim programiranjem (Microsoft).

Visual Studio 2022 ima ugrađenu podršku za Git kloniranje, stvaranje i otvaranje vlastitih repozitorija. Git alat ima sve što je potrebno za preuzimanje (commit) i izdavanje (push) promjena u kodu, upravljanje grana i rješavanje sukoba pri spajanju. Može se GitHub računom upravljati svim repozitorijima izravno unutar Visual Studia (Microsoft).

3. Sandbox žanr

Sandbox je stil igre u kojem su igraču postavljena minimalna ograničenja, dopuštajući igraču da luta i mijenja virtualni svijet po želji. Za razliku od igre u stilu napredovanja, igra u sandboxu naglašava lutanje i omogućuje igraču odabir zadataka za maksimalno izražavanje svoje kreativnosti.

3.1. Minecraft

Minecraft je igra u kojoj igrači stvaraju i razbijaju razne vrste blokova u trodimenzionalnim svjetovima. Dva glavna načina igre su Survival i Creative. U igri Survival, igrači moraju sami pronaći zalihe za gradnju i hranu. Također imaju interakciju s različitim stvorenjima. U igri Creative, igrači dobivaju sve materijale u igri i izgubi se cijeli koncept preživljavanja. Također mogu odmah razbiti sve vrste blokova.

Ova igra može se igrati sama ili online s drugim igračima. Igrači se mogu povezati na online servere gdje mogu graditi i boriti se s drugim igračima te igrati mnoge zasebne igre nastale posebno iz zabavnih funkcionalnosti i mehanika Minecrafta. (Washington Post)

Praktični dio ovog rada temeljit će se više na Creative načinu igre s fokusom na proceduralnu generaciju svijeta i različitih terena u svijetu. Neki od terena će biti Pustinje, Šume itd. Svi će biti objašnjeni kasnije u radu. Uz kreaciju novih terena također će se kreirati neke varijacije u kreiranju jedne velike cjeline svijeta – npr. u pustinji vidjet ćemo kaktuse, dok u Forest terenu drveća i šume. Ovakve varijacije vidjet će se i u dubini svijeta s pojavom različitih vrsta blokova, npr. minerala koje igrači mogu pronaći istraživanjem.

3.2. Terraria

Terraria je proceduralno generirana igra izgrađena oko istraživanja i borbe. Igrači počinju s vrlo osnovnim alatima i oružjem s ciljem poboljšanja svoje snage za osvajanje novih područja. Ima vrlo detaljan sustav bioma (terena), koji uključuje određene zle zone uključujući sloj pakla. Za razliku od drugih igara sandbox žanra, Terraria ima desetke borbi protiv „bossova“ koje su namijenjene kao izazovi igraču (Williams, 2022).

Zbog svog kockastog umjetničkog stila, napredovanja temeljenog na rudarenju i sličnog datuma izdavanja, Terraria se često uspoređuje s Minecraftom. Iako obje dijele isti temeljni koncept, Terraria se mnogo više oslanja na elemente igre uloga sa složenijim sustavom borbe.

Igra ima puno borbi protiv „bossova“ koje uzrokuju drastične promjene u svijetu i napredovanju igrača prema kraju igre. Nasumične nagrade također su velike razlike s Minecraftom. U Minecraftu nagrade su dosta dosljedne i puno ih je manje. Terraria također ima napredovanje s opremom koje je puno bolje razvijeno nego što je u Minecraftu (Williams, 2022).

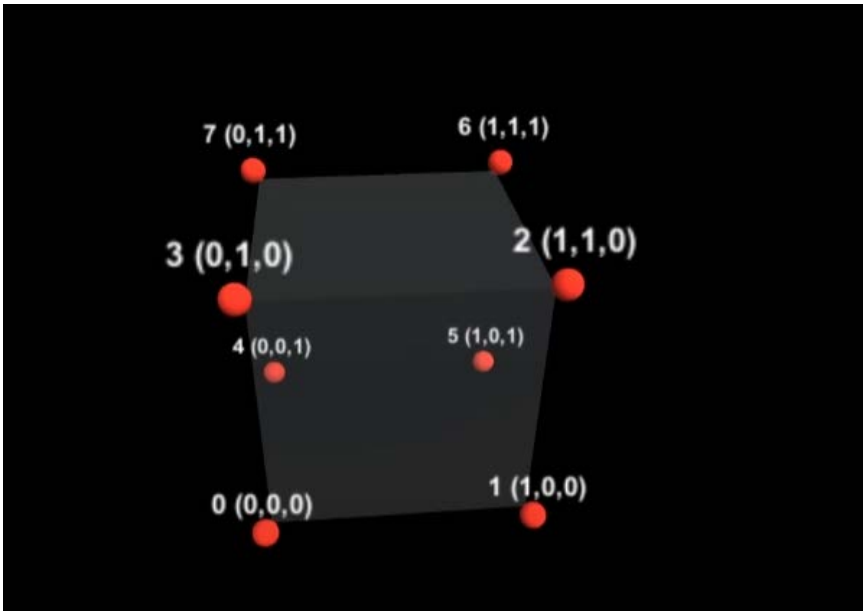
4. Igra

Cilj projekta je napraviti klon Minecraft igre s proceduralno generiranim svijetom koji se još dodatno mora optimizirati. Igraču mora biti omogućeno kretanje, postavljanje i uništavanje blokova te mora postojati mogućnost spremanje trenutnog stanja svijeta da se spremi sve što je izgrađeno.

Proceduralno generiranje svijeta naporno je za računalo, pogotovo s game objektima prije kreiranima u Unity-u. Iz ovog razloga ne možemo koristiti normalne „Cube“ objekte koje Unity već nudi nego moramo kreirati neke svoje objekte koje ćemo kasnije grupirati. Najmanji ovakav objekt naziva se *voxel*, a grupirat ćemo ga u *chunkove*. Dakle, *voxel* će biti jedan Minecraft blok, a skup voxela bit će *chunk*.

4.1. Voxel

Voxel je jedinica grafičke informacije koja definira točku u trodimenzionalnom prostoru. Budući da piksel (element slike) definira točku u dvodimenzionalnom prostoru sa svojim X i Y koordinatama, potrebna je treća Z koordinata. Najjednostavnije rečeno, *voxel* je kocka čije su točke izražene X,Y koordinatama, a Z koordinata predstavlja prednju ili stražnju stranu kocke.



Slika 4. Voxel i njegovi vrhovi

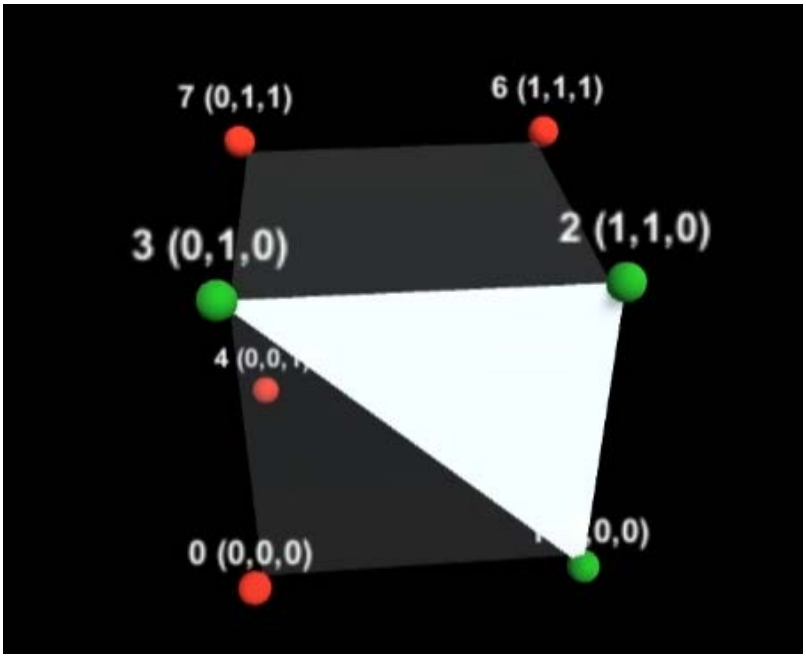
Na ovoj slici možemo vidjeti važnosti X,Y,Z koordinata i kako funkcioniraju. Naime, prva točka u donjem lijevom kutu ima X,Y,Z vrijednosti od (0,0,0). Ako idemo desno na X osi onda

imamo vrijednost (1,0,0). Kada se pomaknemo prema gore to ćemo raditi na Y osi i dobit ćemo vrijednost (1,1,0). Naposljetku se pomičemo na stražnju stranu kocke pomoću Z osi i dobivamo vrijednost (1,1,1).

Ovako to izgleda u kodu:

```
public static readonly Vector3[] VoxelVerts = new Vector3[8]
 {
     new Vector3(0.0f, 0.0f, 0.0f),
     new Vector3(1.0f, 0.0f, 0.0f),
     new Vector3(1.0f, 1.0f, 0.0f),
     new Vector3(0.0f, 1.0f, 0.0f),
     new Vector3(0.0f, 0.0f, 1.0f),
     new Vector3(1.0f, 0.0f, 1.0f),
     new Vector3(1.0f, 1.0f, 1.0f),
     new Vector3(0.0f, 1.0f, 1.0f),
 };
```

Polje *VoxelVerts* predstavlja vrhove kocke a ovi vrhovi su samo jednostavne Vector3 varijable koje sadržavaju svoje X,Y,Z vrijednosti. Bitno je napomenuti da računala svaki mnogokut crtaju uz pomoć manjih trokuta pa tako crtaju i kvadrate u ovoj kocki.



Slika 5. Trokut unutar Voxela

Kod crtanja trokuta jako je bitno da biramo točke u smjeru kazaljke na satu zato što se tako trokut prikaže kada smo ispred njega. U slučaju da smo crtali trokut obrnuto smjeru

kazaljke na satu, prikazivao bi se jedino u slučaju gledanja unutar kocke što baš i nema smisla za našu igru. U gornjoj slici možemo vidjeti redom odabir vrhova 3, 2 i 1 u donjem desnom kutu koji nam formira trokut koji je vidljiv s prednje strane.

```
public static readonly int[,] VoxelTrokut = new int[6, 4] {

    {0,3,1,2}, // stražnja strana kocke
    {5,6,4,7}, // prednja strana kocke
    {3,7,2,6}, // gornja strana kocke
    {1,5,0,4}, // donja strana kocke
    {4,7,0,3}, // lijeva strana kocke
    {1,2,5,6} // desna strana kocke

};
```

Za crtanje ovakvih trokuta treba nam dvodimenzionalno polje koje se zove *VoxelTrokut*. U njega spremamo polja koja nam predstavljaju vrhove koce. Svako polje predstavlja jednu stranu kocke. Primjer može biti gornja strane kocke. Kod Slike 5. možemo vidjeti vrhove kojima crtamo dva trokuta u jednom kvadratu kod gornje stranice. Trokut (3,7,2) i trokut (2,7,6) – skupa su (3,7,2,2,7,6). Ovaj način može se još više pojednostaviti na (3,7,2,6) jer se vrhovi 2 i 7 ponavljaju pa ih je nepotrebno više puta spremati.

Sada kada znamo što je *vertex* i kako ćemo ga koristiti, moramo ga znati nacrtati u Unity-u. Za ovo je zaslužan *Mesh.* Mesh se sastoji od trokuta raspoređenih u 3D prostoru kako bi se stvorio dojam čvrstog objekta (Unity Technologies, 2022). Trokut je definiran sa svoja tri vrha koje smo spomenuli ranije.

```
List<Vector3> vertices = new List<Vector3>();
List<int> triangles = new List<int>();
List<Vector2> uvs = new List<Vector2>();
void UpdateMeshData (Vector3 pos)
{

    for (int p = 0; p < 6; p++)
    {

        if (!CheckVoxel(pos + VoxelData. provjeraVidljivostiStrana
[p]))
        {

            vertices.Add(pos +
VoxelData.VoxelVerts[VoxelData.VoxelTrokut[p, 0]]);
```

```
                vertices.Add(pos +
VoxelData.VoxelVerts[VoxelData.VoxelTrokut[p, 1]]);
                vertices.Add(pos +
VoxelData.VoxelVerts[VoxelData.VoxelTrokut[p, 2]]);
                vertices.Add(pos +
VoxelData.voxelVerts[VoxelData.VoxelTrokut[p, 3]]);
                uvs.Add(VoxelData.voxelUvs[0]);
                uvs.Add(VoxelData.voxelUvs[1]);
                uvs.Add(VoxelData.voxelUvs[2]);
                uvs.Add(VoxelData.voxelUvs[3]);
                triangles.Add(vertexIndex);
                triangles.Add(vertexIndex + 1);
                triangles.Add(vertexIndex + 2);
                triangles.Add(vertexIndex + 2);
                triangles.Add(vertexIndex + 1);
                triangles.Add(vertexIndex + 3);
                vertexIndex += 4;

            }
        }

    }
```

S ovom gore funkcijom prolazimo kroz sve strane kocke, provjerimo postojanje voxela s *CheckVoxel* funkcijom koja jednostavno vraća *true* ako voxel postoji na proslijeđenoj poziciji ili *false* ako ne postoji. U našem slučaju tražimo *voxele* koji ne postoje, a provjere vidljivosti strana nam služe za crtanje voxela koji su površinski nama izloženi. Naime, da bi poboljšali performansu igre, ne moramo crtati unutarnje teksture jednog voxela jer ih ne vidimo pa crtamo samo one vanjske koje trenutno gledamo. Ako voxel ne postoji onda moramo dodati vrhove, trokute, i uv koordinate da bi Unity mogao nacrtati jedan voxel. Jako je bitno da kod dodavanje vrhova moramo koristiti poziciju jer bi se inače svaki voxel kreirao na istoj poziciji u svijetu. Ova gore funkcija samo je primjer inicijalnog postavljanja voxela koji se kasnije dodatno komplicira i bit će objašnjen u kasnijim poglavljima.

```
    public void KreirajMesh()
    {
        Mesh mesh = new Mesh();
        mesh.vertices = Vertices.ToArray();

        mesh.subMeshCount = 2;
        mesh.SetTriangles(triangles.ToArray(),0);
        mesh.SetTriangles(transparentTriangles.ToArray(), 1);
```

```
            mesh.uv = uvs.ToArray();
            mesh.colors = colors.ToArray();
            mesh.normals = normals.ToArray();
            meshFilter.mesh = mesh;
        }
```

Ova gore funkcija popunjene strukture vrhova, trokuta i uv koordinata stavlja u Mesheve koje zatim Unity može crtati.

4.2. Chunk

Kao što je ranije spomenuto, jedan *chunk* je skup više *voxela*. Za popunjavanje chunkova koristimo sljedeću funkciju.

```
    public VoxelState[,,] map = new VoxelState[VoxelData.ChunkSirina,
VoxelData.ChunkVisina, VoxelData.ChunkSirina];
public void Populate()
    {

        for (int y = 0; y < VoxelData.ChunkVisina; y++)
        {
            for (int x = 0; x < VoxelData.ChunkSirina; x++)
            {
                for (int z = 0; z < VoxelData.ChunkSirina; z++)
                {

                    map[x, y, z] = new
VoxelState(World.Instance.DobiVoxel(new Vector3(x + position.x, y, z +
position.y)));

                }
            }
        }
        World.Instance.worldData.AddToModifiedChunkList(this);
    }

}
```

Gledamo ranije unesene vrijednosti za visinu i širinu chunka kojeg želimo popuniti. Naš chunk je 3D objekt pa ga moramo gledati X,Y,Z vrijednosti. X i Z vrijednosti ići će do njegove

širine, a Y vrijednost do njegove visine. U trodimenzionalnom polju „map“ spremamo stanje jednog voxela u chunku. Sada možemo kreirati svaki voxel u chunku zasebno s gornjom funkcijom *Populate* i uz pomoć funkcije *DobiVoxel* koja nam samo vraća stanje voxela na određenoj poziciji. Ovakav kreirani chunk koji sadrži mapu svojih voxela se zatim proslijedi u listu modificiranih chunkova koja se kasnije generira u Unity-u.

4.3. Teksture i tip bloka

Nakon kreiranja voxela, moramo im dati neku teksturu. Za ovo i još neka dodatna svojstva napravljena je klasa *TipBloka* .

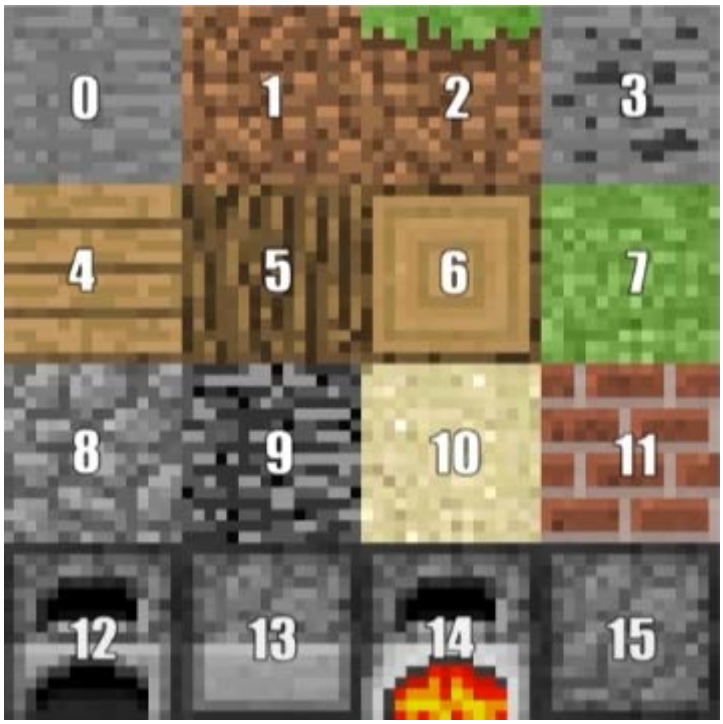
```
public class TipBloka
{
    public string ImeBloka;
    public bool isSolid;
    public bool renderNeighbourFaces;
    public float transparency;
    public Sprite Ikona;


    [Header("Texture Values")]
    public int straznjeLiceTexture;
    public int prednjeLiceTexture;
    public int gornjeLiceTexture;
    public int donjeLiceTexture;
    public int lijevoLiceTexture;
    public int desnoLiceTexture;



    public int GetTextureID(int faceIndex)
    {

        switch (faceIndex)
        {

            case 0:
                return straznjeLiceTexture;
            case 1:
                return prednjeLiceTexture;
```

```
                case 2:
                    return gornjeLiceTexture;
                case 3:
                    return donjeLiceTexture;
                case 4:
                    return lijevoLiceTexture;
                case 5:
                    return desnoLiceTexture;
                default:
                    Debug.Log("Error in GetTextureID; invalid face index");
                    return 0;


            }

        }

    }
```

Ova klasa sadržava atribute koji dodatno opisuju jedan blok. *ImeBloka* u kojoj će se pohraniti ime stvorenog bloka. *isSolid* koji definira čvrstoću bloka. Ako je postavljen u *true*, igrač će moći hodati po tim blokovima, a ako je postavljen u *false*, igrač se kroz takve blokove može kretati. Postavljanjem atributa *renderNeighbourFaces* u *true* govorimo Unity-u da želimo crtati susjedne blokove koji su u doticaju s trenutno odabranim blokom. *Transparency* predstavlja svojstvo gledanja kroz blok. Ako *transparency* iznosi vrijednost 0 onda se kroz taj blok ne može vidjeti, a ako iznosi vrijednost 1 onda se kroz takav blok može vidjeti.

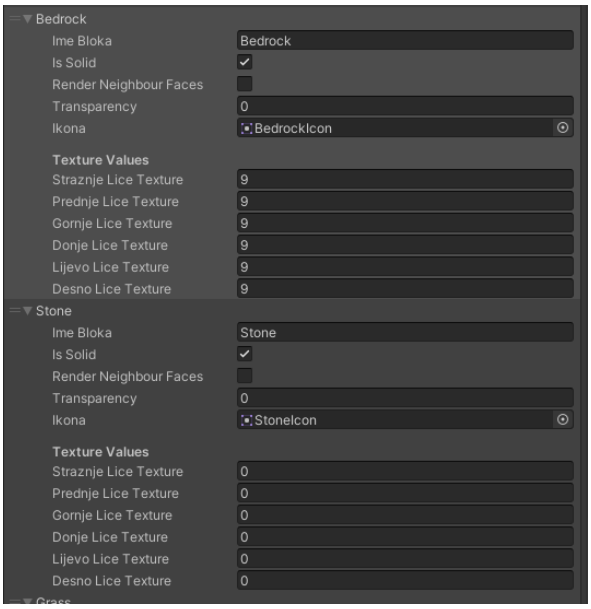
Metoda *GetTextureID* prima indeks strane koju želimo prikazati, te ovisno o njemu vraća tu stranu kocke. Npr. u slučaju da proslijedimo broj 4 kao indeks, dobili bi lijevu stranu kocke.

Da dobijemo teksture različitih blokova koristit ćemo sliku koja u sebi sadržava izgled više blokova i onda ćemo iz nje uzeti blok koji nam treba. Uzimanje blokova odredit ćemo prema indeksima na kojima se nalaze. Početkom u gornjem lijevom kutu s indeksom 0 sve do donjeg desnog kuta teksture.



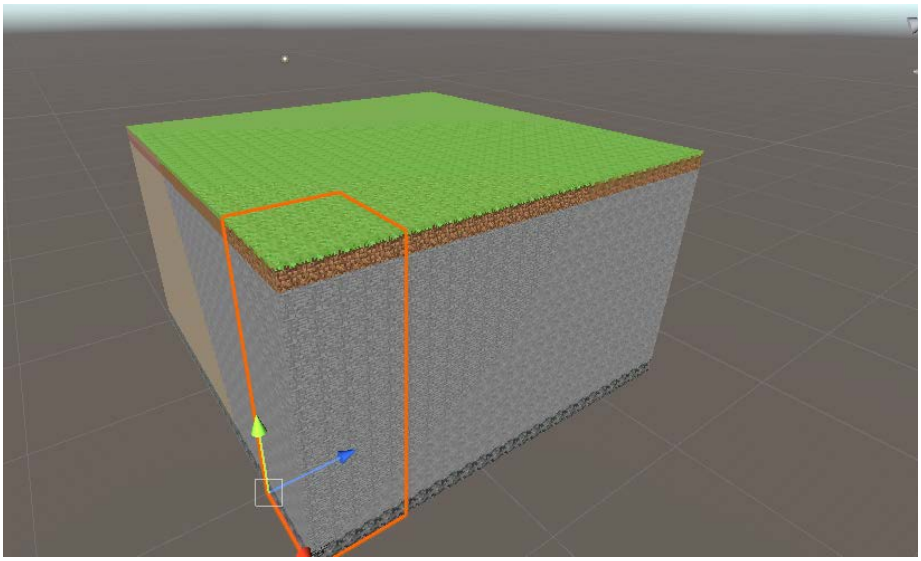
Slika 6. Teksture i njihovi indeksi

Ova slika koristi samo za objašnjenje, u igri je korištena druga slika koja sadržava malo više tekstura. Na slici možemo vidjeti da nam indeks 0 predstavlja teksturu kamena, dok nam npr. indeks 10 predstavlja pijesak.



Slika 7. Prikaz Tipa bloka u Unity-u

U gornjoj slici možemo vidjeti da nam Unity omogućuje dodavanje blokova u polje koje je kreirano u skripti. Kod tekstura moramo samo staviti indeks teksture koje želimo dobiti i tako dobivamo izgled željenog bloka. Nakon uvođenja ovakve logike, nije problem kreirati chunk s različitim blokovima u sebi samo s mijenjanjem indeksa teksture. Naknadno pristupamo polju blokova koje smo kreirali s određenim teksturama.



Slika 8. Chunkovi s različitim blokovima

4.4. Generacija Svijeta

Zamisao iza generacije svijeta je takva da se chunkovi generiraju od određene udaljenosti igrača. Kako igrač hoda svijetom, chunkovi iza njega prestaju biti aktivni i generiraju se novi chunkovi koji se zatim spremaju da se ne bi morali generirati više puta. Ovakvi chunkovi se ovisno o kretanju igrača postavljaju u aktivne, prikazane ili neaktivne, skrivene chunkove. Ovakav način rada potreban je za optimizaciju igre. U slučaju da bi svaki chunk ostao prikazan, igra bi za svakim novokreiranim chunkom bila sve sporija.

Da bismo olakšali njihovo kreiranje, kreirana je klasa *ChunkCoord* koja služi za postavljanje koordinata na kojima želimo kreirati chunk. Ona prima vrijednosti X,Z koje se kasnije proslijede chunku.

```
public class ChunkCoord
{
    public int x;
    public int z;

    public ChunkCoord(int xCord, int zCord)
    {
        x = xCord;
        z = zCord;
    }
}
```

Kod za generaciju svijeta je malo složeniji i izgleda ovako:

```
    Chunk[,] chunks = new Chunk[VoxelData.VelicinaSvijetaChunk,
VoxelData.VelicinaSvijetaChunk];
void GenerateWorld()
    {

        for (int x = (VoxelData.VelicinaSvijetaChunk / 2) -
settings.loadDistance;
            x < (VoxelData.VelicinaSvijetaChunk / 2) +
settings.loadDistance; x++)
        {
            for (int z = (VoxelData.VelicinaSvijetaChunk / 2) -
settings.loadDistance;
                z < (VoxelData.VelicinaSvijetaChunk / 2) +
settings.loadDistance; z++)
            {
                ChunkCoord newChunk = new ChunkCoord(x, z);
                chunks[x, z] = new Chunk(newChunk);
```

```
                chunksToCreate.Add(newChunk);
            }
        }

        igrac.position = spawnPozicija;
        ProvjeriUdaljenostGledanja();

    }
```

Ovdje je bitno spomenuti da imamo dvodimenzionalno polje *chunks* u koje spremamo sve chunkove s koordinatama koje nam ove dvije petlje generiraju. Naime, petlje počinju na polovici svijeta što je zapravo sredina svijeta i kreću se od sredine prema van točno toliko chunkova koliko nam dopušta *loadDistance.* Znači učitavamo toliko chunkova koliko želimo s *loadDistance* varijablom. Petlje generiraju X,Z koordinate. Y koordinatu nemamo jer chunkove ne moramo posebno postavljati na toj koordinati, ovime bi nastale planine (povišeni chunkovi), a one će biti kreirane na drugi način. S novo dobivenim koordinatama kreiramo chunk, spremamo ga u ranije spomenuto polje *chunks* i dodajemo ga u listu chunkova *chunksToCreate* kroz koju iteriramo za kreaciju chunkova. Na kraju funkcije postavljamo poziciju igrača u svijetu i provjeravamo udaljenost gledanja.

```
    private void ProvjeriUdaljenostGledanja()
    {

        ChunkCoord coord = GetChunkCoord(igrac.position);
        igracZadnjiChunk = tempIgracChunkCoord;
        List<ChunkCoord> prosloAktivniChunkovi = new
List<ChunkCoord>(aktivniChunkovi);
        aktivniChunkovi.Clear();

        for (int x = coord.x - settings.udaljenostGledanjaChunks; x <
coord.x + settings.udaljenostGledanjaChunks; x++)
        {
            for (int z = coord.z - settings.udaljenostGledanjaChunks; z
< coord.z + settings.udaljenostGledanjaChunks; z++)
            {
                ChunkCoord ovajChunk= new ChunkCoord(x, z);
                if (isChunkInWorld(new ChunkCoord(x,z)))
                {
                    if (chunks[x,z] == null)
                    {
                        chunks[x, z] = new Chunk(ovajChunk);
                        chunksToCreate.Add(ovajChunk);
```

```
                }
                else if (!chunks[x,z].isActive == true)
                {
                    chunks[x, z].isActive = true;
                }

                chunks[x, z].isActive = true;
                aktivniChunkovi.Add(new ChunkCoord(x, z));
            }


            for (int i = 0; i < prosloAktivniChunkovi.Count; i++)
            {
                if (prosloAktivniChunkovi[i].Jednako(ovajChunk))
                {
                    prosloAktivniChunkovi.RemoveAt(i);
                }
            }
        }
    }
    foreach (ChunkCoord item in prosloAktivniChunkovi)
    {
        chunks[item.x, item.z].isActive = false;
    }
}
```

ProvjeriUdaljenostGledanja najbitnija je funkcija generacije svijeta koja zapravo razlikuje aktivne i neaktivne chunkove. Petlje uzimaju X,Z koordinate chunka na kojem se igrač nalazi. Pomoću varijable *udaljenostGledanjaChunks* iterira se po X,Z osima i provjerava se postojanje chunka s ovim X,Z koordinatama u svijetu. Ako postoji, onda se gleda je li kreiran. Ako je u dvodimenzionalnom polju *chunks* koordinata chunka postavljena u „null“ onda nije kreiran i mora se kreirati. Ako je već kreiran, a nije aktivan onda postaje aktivan i dodaje u listu aktivnih chunkova jer je sada unutar udaljenosti gledanja. Lista *prosloAktivniChunkovi* je kopija liste *aktivniChunkovi,* a na kraju funkcije se ove dvije liste uspoređuju. Ako u listi *prosloAktivniChunkovi* postoji chunk na kojem smo trenutno, znači chunk koji se nalazi unutar udaljenosti gledanja, onda se takav izbriše iz te liste. Zadnji korak je samo svaki chunk unutar liste *prosloAktivniChunkovi* postaviti u neaktivno stanje koje Unity neće prikazati.

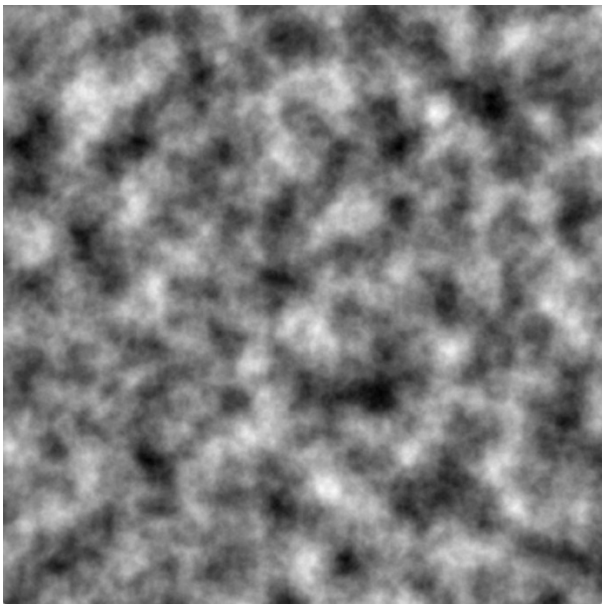
Funkcija *ProvjeriUdaljenostGledanja* se osim u *GenerateWorld* funkciji priziva kod Unity *Update* funkcije koja se poziva svaki frame.

```
private void Update()
  {
      tempIgracChunkCoord = GetChunkCoord(igrac.position);

      if (!tempIgracChunkCoord.Jednako(igracZadnjiChunk))
      {
          ProvjeriUdaljenostGledanja();
      }
  }
```

Ovdje možemo vidjeti da se *ProvjeriUdaljenostGledanja* poziva svaki put kada je trenutni chunk na kojem se igrač nalazi različit od onog na kojem se igrač prijašnje nalazio i time smo postigli optimizirano generiranje svijeta.

4.4.1. Perlinov šum i različiti tereni

Sada kada postoji generacija svijeta treba se i dodatno napraviti varijacija u svijetu. Ove varijacije postat će brda, planine i različite vrste terena. Sve ovo će se postići koristeći Perlinov šum za koji već postoje ugrađene funkcije. Na sljedećoj slici možemo vidjeti kako Perlinov šum izgleda računalno nacrtan.



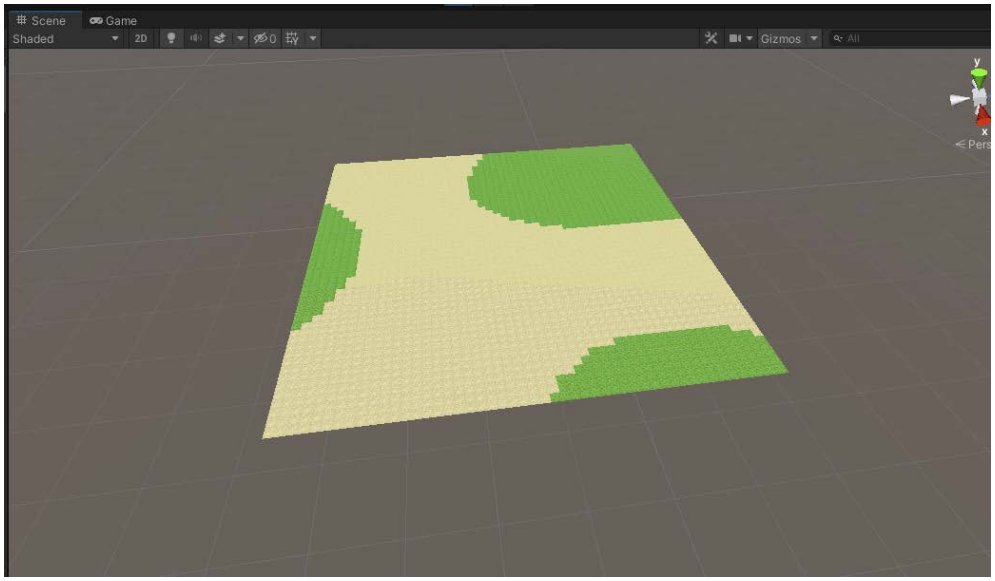
Slika 9. Perlinov šum

(preuzeto s https://medium.com/@yvanscher/playing-with-perlin-noise-generating-realistic-archipelagos-b59f004d8401)

Funkcija za korištenje Perlinovog šuma izgleda ovako:

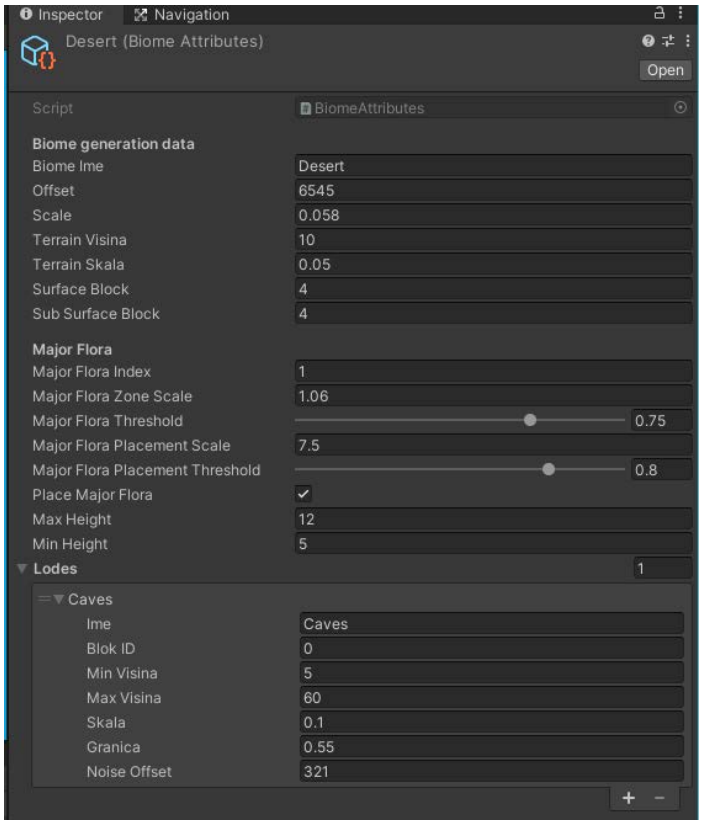
```
public static float Get2DPerlin(Vector2 pozicija, float offset, float
skala )
    {

        pozicija.x += offset + VoxelData.seed + 0.1f;
        pozicija.y += offset + VoxelData.seed + 0.1f;
        return Mathf.PerlinNoise(pozicija.x / VoxelData.ChunkSirina *
skala, pozicija.y / VoxelData.ChunkSirina * skala);
    }
```

VoxelData.seed varijabla određuje naš položaj na „slici“ Perlinovog šuma. Seed kao i u Minecraftu određuje gdje u svijetu ćemo se stvoriti i to radi zajedno sa proslijeđenim offsetom. Skala određuje veličinu Perlinovog šuma. Na primjeru planine, manja skala bi dala višu planinu, a veća skala nižu. Funkcija nam vraća vrijednost između 0,0 i 1,0 koju koristimo za generaciju voxela na različitim visinama time stvarajući brda. Na ovaj način možemo i kreirati različite vrste blokova na naš chunk.



Slika 10. Perlinov šum na chunku

Ove vrijednosti nepotrebno je unositi ručno pa je kreiran *Unity Asset* nazvan *Biome Attributes* koji možemo vidjeti na sljedećoj slici.



Slika 11. Biome Attributes

Jedan ovakav *asset* sadržava svoje *offsetove* i *skale* po kojima se izvršava Perlinov šum. *Terrain Visina* i *Terrain Skala* služe za postavljanje voxela po visini i time kreiramo brda. *Surface Block* i *Sub Surface Block* sadrže ID-eve tekstura koje želimo imati za površinske i ispod površinske blokove u chunku. U ovom slučaju ID predstavlja pijesak što i ima smisla jer se radi o *Desert* terenu.

„Major Flora“ dio ovog *asseta* je zaslužan za kreiranje drveća i kaktusa. *Major Flora Indeks* je postavljen u 1, a u kodu nam to predstavlja kaktuse koji se zatim po maksimalnoj i minimalnoj visini postavljaju u svijet. *Major Flora Zone Scale i Threshold* služe za određivanje zone u kojoj će se postavljati drva ili kaktusi. *Major Flora Placement Scale i Major Flora Placement Threshold* služe za pronalazak bloka na kojem će biti kreirano drvo ili kaktus unutar te zone. Manje *Treshold* vrijednosti nam daju veću mogućnost kreiranja npr. šuma dok veće rade suprotno.

„Lodes“ (mother lode jer služi za generaciju minerala) dio ovog *asseta* predstavlja generaciju voxela unutar chunka pomoću 3D Perlinovog šuma. Blok ID je blok koji će se generirati, a *min i max visine* su visine ispod i iznad kojih se taj blok neće pojaviti. Znači blok određenog ID-a na sličan način generirat će se između zadane min i max visine. Blok ID na

primjeru je zrak i njime generiramo špilje. Mogli smo zadati npr. dijamante i time bi igraču omogućili pronalazak dijamanata kopanjem što je jedna velika značajka Minecrafta.

4.5. Igrač

U ovom poglavlju osvrnuti ću se na izradu kontrola za igrača. Igrač se mora moći kretati, uništavati i stavljati blokove, pristupiti inventaru i ugasiti igru. Bitno je napomenuti da se igračev input traži svaki „frame“ u ranije spomenutoj Unity *Update* funkciji što znači da se ova funkcija poziva više puta u sekundi.

4.5.1. Kretanje igrača

U sljedećem kodu možemo vidjeti bitne atribute koje igrač ima. *isSprinting, isGrounded, zahtjevSkakanja* predstavljaju stanje igrača. Ako su postavljene na *true*, igrač trči ili je prizemljen ili želi skočiti. Pomoću ovih atributa i brzinama hodanja i trčanja možemo kreirati igrača koji se kreće po svijetu. Uz gravitaciju i silu skakanja možemo kreirati skakanje igrača.

```
public class Player : MonoBehaviour
{
    public bool isSprinting;
    public bool isGrounded;
    public float brzinaHodanja = 3f;
    public float brzinaTrcanja = 6f;
    public float silaSkakanja = 5f;
    public float gravitacija = -9.8f;
    public float igracSirina = 0.15f;
    private float verticalMomentum = 0;
    private bool zahtjevSkakanja;
    private float vertical;
    private  float horizontal;
    Vector2 mouseTurn;
    private World svijet;
    private Transform kamera;
    private Vector3 brzina;
    public float doseg = 8f;
}
```

4.5.1.1. Hodanje, trčanje i skakanje

Ovaj isječak funkcije predstavlja kretanje igrača i njegove pokrete mišem.

```
    private void DobiPlayerInput()
     {
         horizontal = Input.GetAxis("Horizontal");
         vertical = Input.GetAxis("Vertical");
         mouseTurn.x += Input.GetAxis("Mouse X") *
svijet.settings.mouseSensitivity;
         mouseTurn.y += Input.GetAxis("Mouse Y") *
svijet.settings.mouseSensitivity;
         if (Input.GetButtonDown("Sprint"))
         {
             isSprinting = true;
         }
         if (Input.GetButtonUp("Sprint"))
         {
             isSprinting = false;
         }
         if (isGrounded && Input.GetButtonDown("Jump"))
         {
             zahtjevSkakanja = true;
         }
      }
```

Prvi dio funkcije dobiva X,Y kretanje miša i množi ga s osjetljivošću koji igrač može odabrati. Zatim se gleda „Sprint“ gumb koji je u Unity-u postavljen u „Shift“ i omogućuje se trčanje igrača. Isto tako gleda se „Jump“ gumb koji je postavljen u „Space“ pa uz uvjet da je igrač prizemljen on može skočiti.

4.5.1.2. Uništavanje i postavljanje blokova

Sljedeći isječak funkcije predstavlja uništenje bloka pritiskom na lijevi klik miša i postavljanje bloka pritiskom na desni.

```
    private void DobiPlayerInput()
     {
         if (HighlightBlock.gameObject.activeSelf)
         {
             if (Input.GetMouseButtonDown(0))
             {
svijet.GetChunkFromV3(HighlightBlock.position).EditVoxel(HighlightBlock.
position, 0);
             }
```

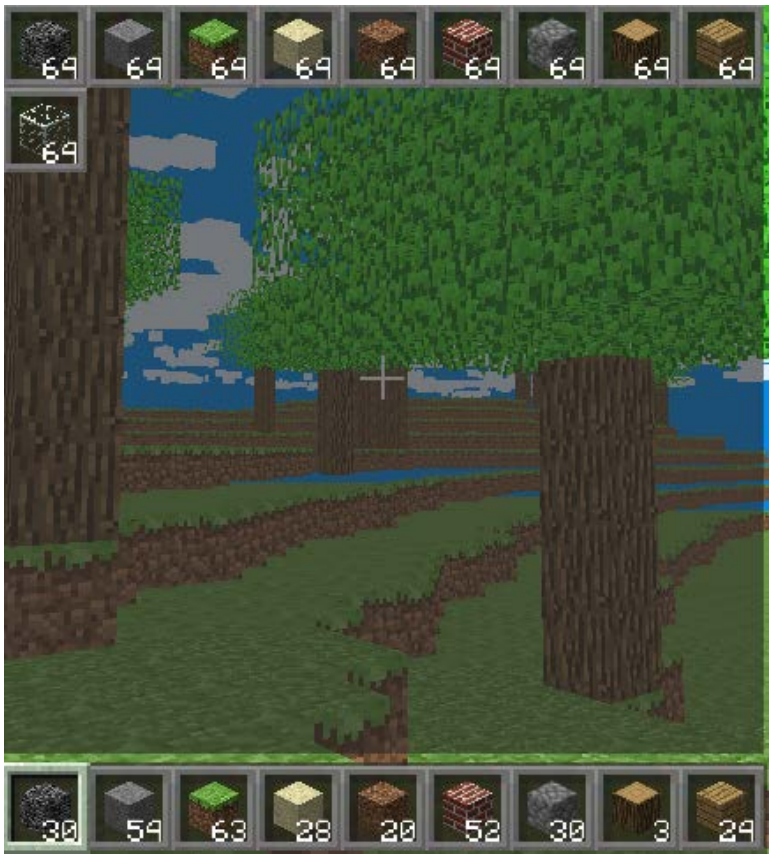
```
            if (Input.GetMouseButtonDown(1))
                {
                    if (toolbar.slots[toolbar.slotIndex].HasItem)
                    {
svijet.GetChunkFromV3(PlaceBlock.position).EditVoxel(PlaceBlock.position
, toolbar.slots[toolbar.slotIndex].itemSlot.stack.id);
                        toolbar.slots[toolbar.slotIndex].itemSlot.Take(1);
                    }
                }
            }
        }
```

Funkcija jednostavno gleda je li neki blok označen i ako je onda pronađe njegov *chunk* pa u slučaju lijevog klika briše taj *voxel*. U slučaju desnog klika pregleda igračev „toolbar“ s blokovima, provjeri ima li igrač neki blok i taj blok postavi na označenu lokaciju.

4.5.2. Pristupanje inventaru i gašenje igre

U funkciji *Update* na pritisak gumba „I“ otvara se ili zatvara inventory. Ako je inventory prije bio otvoren onda se zatvara i suprotno. Pritiskom gumba „Escape“ igra se ugasi zvanjem funkcije *Application.Quit.*

```
    private void Update()
      {
          if (Input.GetKeyDown(KeyCode.I))
          {
              svijet.inUI = !svijet.inUI;
          }
      }
    private void DobiPlayerInput()
      {
          if (Input.GetKeyDown(KeyCode.Escape))
          {
              Application.Quit();
          }
      }
```

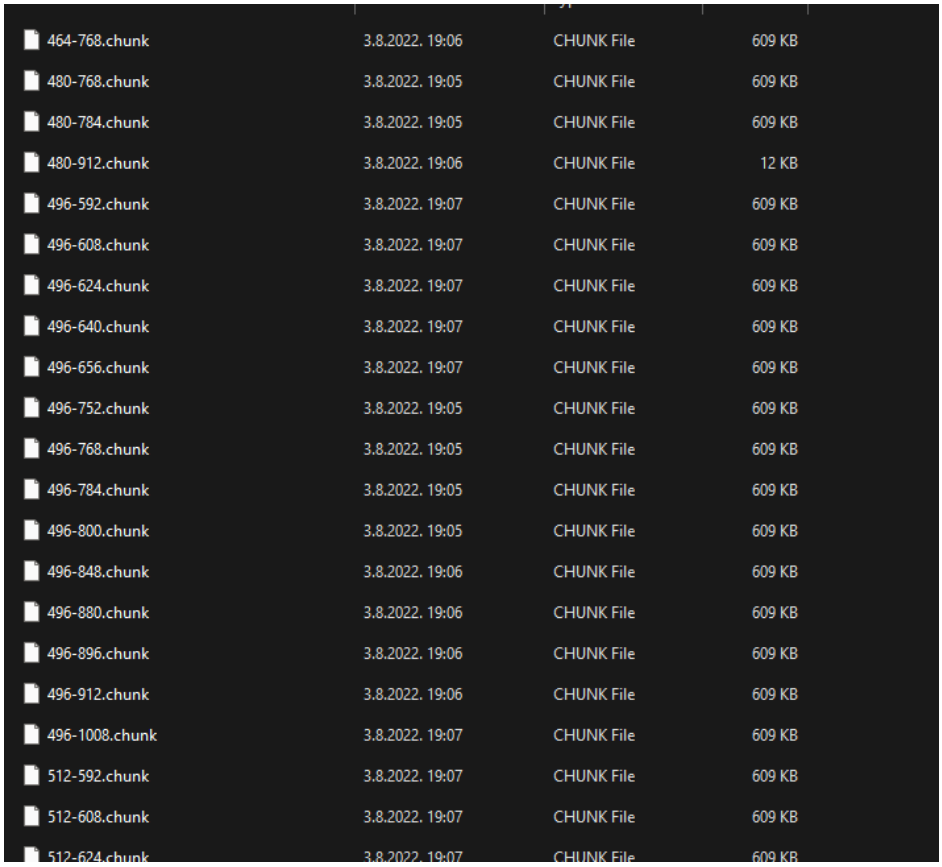


Slika 12. Inventar i Toolbar
(Izrađeni u Unity alatu)

Inventar učitava ikone iz slike i postavlja ih kao ikone blokova. Ovi blokovi se mogu pomoću *drag&drop* funkcionalnosti dovući do *toolbara* gdje ih igrač može odabrati i onda postavljati. Prilikom postavljanja se broj blokova smanjuje, a nakon postavljanja zadnjeg bloka se ikona tog bloka miče iz *toolbara*.

4.6. Spremanje svijeta

Naravno da igraču moramo omogućiti spremanje svijeta kakvog ga je napravio. Za ovo ćemo pohraniti stanje svijeta i svakog chunka u ranije spomenutim strukturama na računalo igrača. Kod učitavanja igre samo ćemo provjeriti postajanje svijeta s imenom kojeg želimo učitati i onda učitati spremljene chunkove. Za svijet će se generirati .world datoteka, a za chunk .chunk datoteka koja ima poziciju chunka u svijetu.



Slika 13. Spremljeni Chunkovi

Ovako izgleda funkcija za učitavanje chunkova:

```
public static ChunkData LoadChunk(string worldName, Vector2Int position)
    {
        string chunkName = position.x + "-" + position.y;
        string loadPath = World.Instance.appPath + "/saves/" + worldName
+ "/chunk/" + chunkName + ".chunk";
        if (File.Exists(loadPath))
        {
            BinaryFormatter formatter = new BinaryFormatter();
```

```
            FileStream stream = new FileStream(loadPath, FileMode.Open);

            ChunkData chunkData = formatter.Deserialize(stream) as
ChunkData;

            stream.Close();
            return chunkData;
        }
        else
        {
            return null;
        }
    }
```

Funkcija dobiva poziciju chunka i ime svijeta pa ako postoji takva datoteka, u njoj traži chunk s istim imenom i vrati ga. Ako ne postoji onda vrati *null*. Vidimo da varijabla *chunkName* dobiva ime *x i y* koordinatom *Vector2Int* varijable *position.* Bitno je spriječiti zabunu kod korištenja *y* koordinate varijable *position*, to je zapravo *z* koordinata chunka jer su svi chunkovi na istoj *y* poziciji kao što je objašnjeno ranije.

4.7. Početni zaslon (Main menu)

Svaka igra mora imati početni zaslon kod kojeg igrač može postaviti neke opcije, odabrati i kreirati svjetove. Ovaj zaslon napravljen je kao posebna scena u alatu Unity na kojoj su dodavani objekti nad kojima se mogu izvršiti neke radnje. Primarno je to ovdje gumb čijim klikom se izvršava skripta u kojoj se sakriju objekti koji se ne koriste i učitaju novi željeni objekti. Npr. kod *Select world* gumba će se sakriti početni zaslon i otvoriti objekti kojima igrač može odabrati svijet. Opcija *New World* istom logikom omogućuje kreaciju novih svietova, a *Settings* otvara postavke koje igrač može mijenjati.

Slika 14. Početni zaslon

Slika 15. New World zaslon

Igrač u *New World* zaslonu može unijeti ime svijeta i seed od kojeg će se svijet generirati. Ovi podaci spremaju se kao što je objašnjeno ranije. Pritiskom na gumb *Main Menu* vraća se na početni zaslon. Spremanje naziva svijeta i seeda bitno nam je za *Select world* zaslon jer on koristi ove podatke za učitavanje svijeta kojeg igrač odabere.

Slika 16. Select world zaslon

U *Select World* zaslonu, igrač pomoću drop down menija može odabrati svoje spremljene svjetove i početi igrati igru. Ostale opcije kao *Main Menu* i *Quit* ranije objašnjenim korištenjem skripti nakon klika vraćaju igrača na *Main Menu* ili ugase igru.

Slika 17. Settings zaslon

U *Settings* zaslonu igrač može mijenjati opcije koje se kasnije učitaju u igru. Neke od ovih opcija su oblaci koji mogu biti „Fast“,“Fancy“ ili „Clouds Off“ čime utječe na različite načine prikaza oblaka u igri. „Enable Threading“ opcija omogućava dretve s kojima se igrica puno brže učitava. „Limit Fps“ opcija ograniči broj frameova koji se prikazuju na ekranu. Igrač također može mijenjati brzinu miša opcijom „Mouse sensitivity“. „Chunk Animations“ opcija omogućuje posebnu animaciju kod učitavanja chunkova.

5. Zaključak

Proces kreiranja igara u Unity alatu dugotrajan je i podosta zahtjevan posao koji uključuje znanja iz različitih područja. Neka od tih područja su programiranje, dizajniranje tekstura za igre, razumijevanje računalne grafike i dizajniranje korisničkog sučelja. Da bi ovaj posao bio čim lakši potrebno je dobro razumijevanje alata u kojima radimo pa se zato prvi dio ovog rada oslanjao na objašnjavanje bitnih značajki Unity alata. Učenje nam znatno olakšava velik broj uputa na Unity forumima i njegovom priručniku.

Za ovaj komplicirani proces nije dovoljno dobro poznavanje alata nego i optimizacija koje se mogu postići u istom. U ovom radu detaljno je objašnjena optimizacija kreiranja svijeta i općenita optimizacija koda tamo gdje je to bilo moguće.

Iako je izrada bilo koje igre zahtjevan proces, izrada igre sandbox žanra sa svojom proceduralnom generacijom svijeta je jedan od najkompliciranijih projekata za početnika u razvijanju igara. Svakom promjenom algoritma za kreaciju svijeta, igra se može „rušiti“ i zbog toga čak velika poduzeća s ogromnim brojem zaposlenika kreiraju detaljnije verzije ovakvih igara godinama te godinama nakon izdavanja još uvijek na njima rade kako bi se uklonile greške.

Popis literature

Dave, A. (6. 4 2022). *Why Choose Unity 3D for Your Next Game Development Project?* Preuzeto 7. 7 2022 iz MindInventory: https://www.mindinventory.com/blog/unity-3d-game-development/

Dealessandri, M. (2020. 1 16). *What is the best game engine: is Unity right for you?* Preuzeto 7. 7 2022 iz Game Industry.biz: https://www.gamesindustry.biz/what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you

Hope, C. (7. 6 2019). *Computer Hope.* Preuzeto 7. 7 2022 iz https://www.computerhope.com/jargon/v/visual-studio.htm

Microsoft. (n.d.). *Visual Studio.* Preuzeto 7. 7 2022 iz Microsoft: https://visualstudio.microsoft.com/vs/

Sinicki, A. (3. 20 2021). *What is Unity? Everything you need to know.* Preuzeto 2022. 7 7 iz Android Authority: https://www.androidauthority.com/what-is-unity-1131558/

Unity Technologies. (30. 7 2022). *Anatomy of a Mesh.* Preuzeto 2. 8 2022 iz Unity Manual: https://docs.unity.cn/560/Documentation/Manual/AnatomyofaMesh.html

Unity Technologies. (30. 7 2022). *The Inspector window.* Preuzeto 2. 8 2022 iz Unity Manual: https://docs.unity3d.com/Manual/UsingTheInspector.html

Unity Technologies. (30. 7 2022). *The Project window.* Preuzeto 2. 8 2022 iz Unity Manual: https://docs.unity3d.com/Manual/ProjectView.html

Washington Post. (n.d.). *What is Minecraft.* Preuzeto 7. 7 2022 iz https://www.washingtonpost.com/lifestyle/kidspost/what-is-minecraft/2013/03/14/98c54514-8a57-11e2-a051-6810d606108d_story.html

Williams, K. (12. 2 2022). *Why you should play Terraria in 2022.* Preuzeto 7. 7 2022 iz WIN.gg: https://win.gg/news/here-is-why-you-should-play-terraria-in-2022/

Popis slika

Prilozi

Build igre (.exe) dostupan je na:

https://drive.google.com/drive/folders/1KIj8skdgp_reNMUEHefWWZuxz3B5LdPt?usp=sharing.