Proceduralno generiranje planeta

Sišul, Marko

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Marko Sišul

Proceduralno generiranje planeta

Završni rad

Mentor: doc. dr. sc. Marina Ivašić Kos

Rijeka, srpanj 2018.

Sadržaj

1. Zadatak za završni rad

2. Sažetak

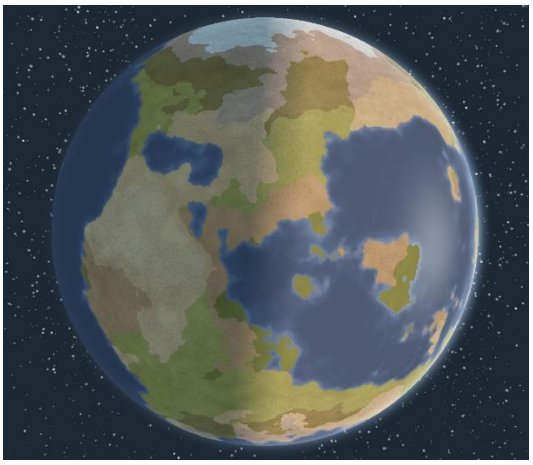
U ovom radu je prikazano proceduralno generiranje planeta korištenjem grafova. Detaljno je opisan način i proces izrade planeta te svi algoritmi korišteni u njegovoj izradi. Opisane su klase koje su potrebne u grafu za opis planeta te algortmi koji su korišteni u procesu generacije i njihov utjecaj na samu strukturu planeta.

Ključne riječi: Unity, proceduralno generiranje, planet, perlin noise, noise, graf, poligonalna mreža.

3. Inspiracija

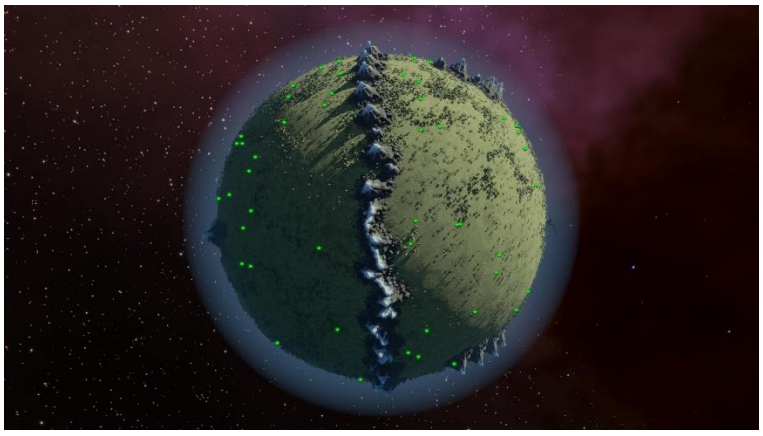
Odabir ove teme je inspiriran raznim igrama koje su koristile proceduralno generirane planete, a i onim igrama gdje bi takvo nešto unijelo nove mogućnosti pa čak i pridonijelo igrivosti same igre.

Jedna od tih igara je naravno Rimworld [1], koji je postao jedan od indie[1] [2] hitova kroz zadnjih nekoliko godina razvoja te igre. Trenutno je još uvijek nedovršena te u Beti, no najavljeno je da će njena sljedeća verzija biti i finalna verzija za izlazak igre. Na Slici 1 je prikazan proceduralno generiran planet u Rimworld-u.



Slika 1: Rimworld planet [3]

Planetary Annihilation [4] je još jedna od igara koja koristi proceduralno generirane planete, no razlika ovog načina generiranja od prethodnog je u tome što kod Rimworld-a planet ima polja, dok u Planetary Annihilation-u nema. Kao što je prikazano na Slici 2, njihov planet je naseljen različitim objektima, a i on sam nije ravan kao kod prethodnog primjera, već ima uzvisine i udubine.



Slika 2: Planetary Annihilation planet [5]

Napominje se da je preduvjet za jasnije i lakše razumijevanje ovog rada dobro poznavanje Unity-a, linearne algebre, grafova te funkcija šuma (perlin noise) [6].

[1] Indie igra je video igra koja je najčešće kreirana bez financijske potpore izdavača.

4. Cilj

Cilj ovog rada je opisati postupak izrade generatora proceduralnih planeta koji će po strukturi biti što općenitiji kako bi jednostavno mogli doći do željenog tipa planeta. Ovaj generator je zamišljen kao baza koju je vrlo lako nadograditi ili prilagoditi različitim potrebama. Modularnost i raznolikost opcija je bit generatora, ali također i čitljiva struktura samog planeta za jednostavnije buduće nadograđivanje.

Kroz ovaj rad pobliže će biti opisan postupak samog generiranja baznog planeta ali i svake od opcija koje generator nudi, kao što su: elevacija planeta, veličina planeta, crtanje po planetu, odabir prikaza planeta, spremanje i čitanje planeta.

Generator je napisan u C# programskom jeziku, u zasebnoj biblioteci zbog portabilnosti i modularnosti. Uz njega je napravljen i program u Unity-u koji služi kao sučelje za korištenje biblioteke za pregled raznih mogućnosti, međutim, to sučelje nije nužno za generiranje planeta jer se ono može odvijati automatski uz nasumične vrijednosti.

5. Proceduralno generiranje

Da bi mogli definirati proceduralno generiranje planeta, prvo moramo definirati što je zapravo proceduralno generiranje sadržaja. Proceduralno generiranje sadržaja je kreiranje sadržaja igrice algoritmima uz limitiran ili indirektan korisnikov unos [7]. Iz ove definicije se može zaključiti da proceduralno generiranje nije ručno generiranje podataka, već generiranje podataka algoritmima.

Proceduralno se mogu generirati razni sadržaji igara: teksture, 3D modeli, leveli, sela, gradovi, pa tako i planeti. U većini slučajeva proceduralno generiranje sadržaja može se izvesti na nekoliko načina, jer ne postoji jedan najbolji način za generirati neki sadržaj. Budući da svaki način ima svoje prednosti i nedostatke vrlo je važno prilikom odabira postupka generiranja objekata voditi računa o pitanjima kao što su: Čemu služi generirani sadržaj? Je li bitna brzina? Složenost? Nasumičnost?

Kako onda započeti proceduralno generirati objekt? Kao prvo treba odabrati što želimo generirati. Za primjer uzmimo proceduralno generiranje levela. Prva dva glavna pitanja koja se moramo upitati su hoće li cijeli level biti proceduralno generiran? Ili ćemo imati dijelove njega kao glavnog neprijatelja i njegovu okolinu napravljenu rukom. Dobar dizajn je nekada imati ključne dijelove napravljene rukom dok se oni povezuju sa sadržajem generiranim proceduralno. Primjer toga bi bili dungeon-i [8] u Diablo 3 [9] igri [10].

Proceduralno generiranje planeta je jedna od novijih ideja, koja prije nije bila moguća zbog kompleksnosti te vremena koje algoritmi troše. Također, algoritmi za njihovo generiranje nisu bili razvijeni, a i danas su u samim začecima.

6. Načini proceduralnog generiranja planeta

6.1. *Općenito*

U proceduralnom generiranju planeta treba uzeti u obzir dvije različite potrebe za proceduralnim planetom.

Slika 3:Planet s poljima[11]

Slika 4: Običan planet[12]

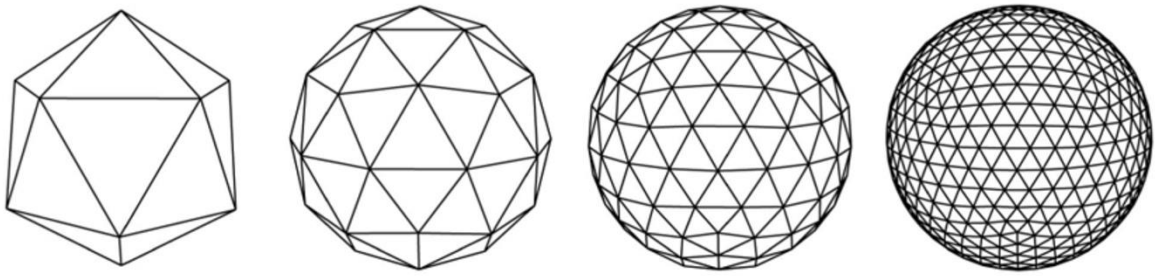
Na Slici 3 prikazan je prvi način planeta koji bi nas mogao zanimati, a to je planet koji se sastoji od polja od kojih je svako definirano s nekim svojim sadržajem.

Slika 4 prikazuje drugi način, gdje nam polja uopće nisu bitna, već nam je bitna tekstura i opcije različitih veličina planeta. Tekstura nam odlučuje o tome kako nam planet izgleda, hoće li on biti vodeni planet, plinski div ili nešto sasvim novo.

Naravno to nisu jedine potrebe za planetom, postoji mnogo drugih potreba za izgledom i strukturom planeta.

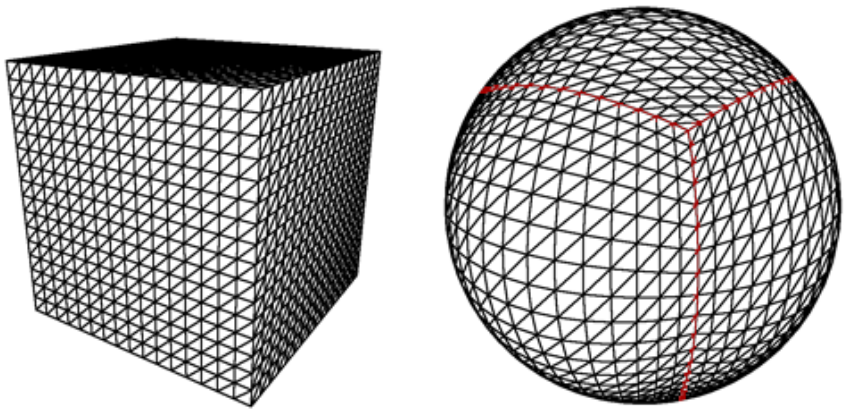
Postoji nekoliko načina za generiranje planeta, te od njih treba odabrati onog koji nam najviše odgovara za ono što želimo postići. Jedni od poznatijih načina su:

1. Ikosaedrena sfera (eng. *Icosahedron sphere*)
2. Sferična kocka (eng. *Spherified cube*)
3. Sfera zemljopisnih dužina (eng. *Longitude latitude sphere*)



Slika 5: Ikozaedrena sfera [13]

Tip generiranja sfere prikazan na Slici 5 je zahtjevan za računalo zbog velikih količina računanja, a s druge strane je odličan po pitanju strukture sfere s minimalnim distorzijama te gotovo istim trokutima na svakom dijelu površine.



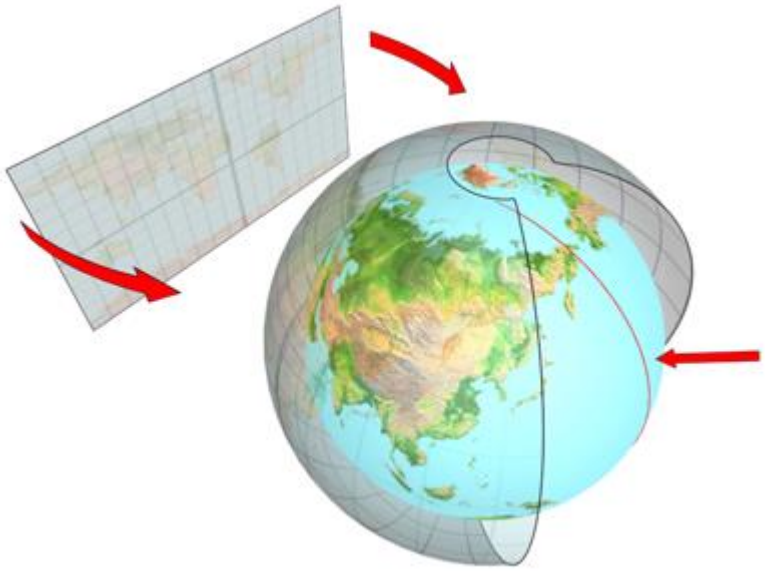
Slika 6: Sferična kocka [14]

Generacija sfere prikazana na Slici 6 nije zahtjevna po pitanju računalnih resursa, te je vrlo lagano dobiti UV koordinate[2] [15] za teksturiranje, no deformacije se mogu uočiti kod spajanja stranica kocke (označeno crveno).

[2] UV koordinate – koordinate svake točke 3D modela u rasponu od 0 do 1, koriste se za stavljanje tekstura na modele.

Slika 7: Sfera zemljopisnih dužina [16]

Sfera zemljopisnih dužina je način generiranja uzet s geografskog stajališta. Uzimaju se longitudinalne i latitudinalne vrijednosti te one služe za određivanje UV koordinata. Koliko god to zvučalo prirodno i lagano za implementirati, ovaj način zapravo donosi velike anomalije. Što bliže polovima gledamo, to su tekstura i planet više deformirani. Na Slici 8 se to može vidjeti. Sama tekstura mora kompenzirati za anomalije koje nastaju pri generaciji takvog planeta, kako bi one bile uklonjene. Naravno, ova tehnika ima prednosti kad nas na primjer, ne zanimaju polovi kao na Slici 3.



Slika 8: Razvučena tekstura [17]

6.2. Odabir

Za potrebe generatora planeta velika važnost je usmjerena na pravilnost i podobnost planeta za teksturiranje. Uz ta dva uvjeta ističu se sferična kocka te ikosaedrena sfera.

Sferična kocka je pogodna i za generirati i za teksturirati, no smetaju distorzije koje bi bile uočljive. U daljnjem promatranju, ako bi krenuli generirati elevaciju te baviti se nekakvim distorzijama površine, postoji šansa da će se to maskirati, ali isto tako i rizik da će se početne distorzije još više vidjeti.

Zbog toga je na kraju kao način generiranja planeta uzeta ikozaedrena sfera koja u teoriji nema distorzija te ima odličnu strukturu i raspored likova po površini, a to nije ograničavajuće za daljnji rad i ne stvara nikakve rizike.

7. Unity

Unity [18] je cross-platform[3] [19] game engine[4] [20] napravljen od strane Unity Technologies, prvi puta izdan 2005 godine. Od 2018 godine podržava 27 platforma te može biti korišten da se kreiraju 2D i 3D igre kao i razne simulacije na računalima, laptopima, konzolama, pametnim TV-ovima i mnogim drugim platformama.

Slika 9: Unity logo[21]

Glavni programski jezik Unity-a je C# od 2017 godine, a izbačeni su prethodno podržani jezici Javascript i Boo.

Također Unity podržava mnoge grafičke API-eve poput: Direct3D, OpenGL, OpenGL ES, WebGL, Metal i Vulkan.

Unity je vrlo popularan game engine zbog podržavanja mnogih platformi, ali isto tako i lakoći rada u njemu. Podržava i olakšava izradu 2D i 3D igara te nudi mnoge mogućnosti, ali ono zbog čega se ističe od ostalih je da igrice u njemu mogu raditi i ljudi bez mnogo iskustva u izradi igara. Njihov cilj njihovim riječima izrečen je: „Making game development universally accessible“.

Neki od poznatijih velikih naslova izrađenih u Unity-u su: Endless Space 2, Pillars of Eternity, Cities: Skylines, pa čak i Pokemon GO [22].

A neke od poznatijih indie naslova: Rimworld, Oxygen not Included, i mnogi drugi.

[3] Cross-platform – podržava više platformi
[4] Game engine – skup alata i programa koji olakšavaju izradu igara ali i pokretanje grafike, zvuka i ostalih komponenti potrebnih za rad igre

8. Implementacija

Nakon odabira načina generiranja planeta treba krenuti s njegovom implementacijom. U slučaju ikosaedrene sfere počinje se s ikosaedrom. Kako bi se prikazao lik u Unity-u treba generirati nekoliko stvari te ih zajedno ubaciti u pologinalnu mrežu (mesh) [23].

8.1. Mesh

Polygon mesh je mreža koju čine vrhovi (verteksi), rubovi i stranice poligona pravilnog geometrijskog tijela koje je opisano nekom objektu u 3D grafici.

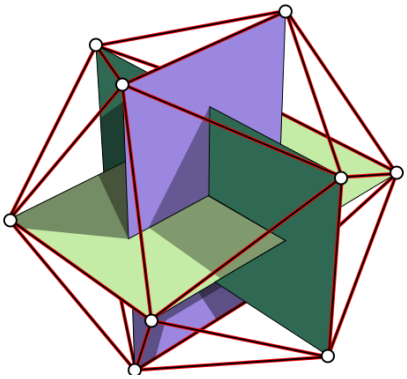
```
239         //set up
240         GameObject go = new GameObject { name = "planet" };
241         go.AddComponent<MeshFilter>();
242         go.AddComponent<MeshRenderer>();
243         Mesh mesh = go.GetComponent<MeshFilter>().mesh;
244
245         //assing mesh stuff
246         mesh.vertices = vertices;
247         mesh.normals = normals;
248         mesh.triangles = triangles;
249         mesh.colors = colors;
```

Slika 10: Mesh u Unity-u

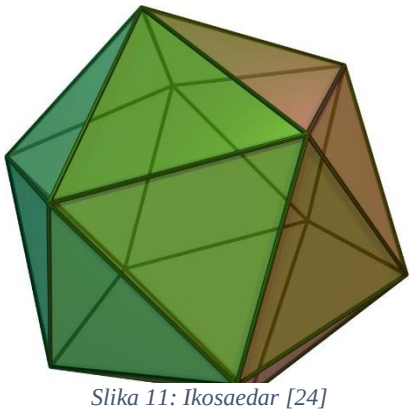
Da bi u Unity-u kodom napravili neku poligonalnu mrežu te ga prikazali u sceni trebamo mu dodijeliti vertekse, normale i trokute. To troje zajedno definira oblik koji će mesh imati kao što prikazuje Slika 10. Također treba dodati komponente MeshFilter te MeshRenderer, a mesh treba uzeti iz MeshFilter komponente.

8.2. Prvi pokušaj implementacije

Ikosaedar je pravilni geometrijski lik s 20 strana od kojih je svaka trokut kao što se vidi na Slici 12.

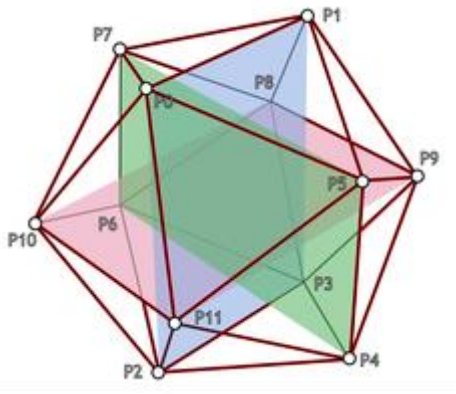


Slika 12: Sastav ikosaedra [24]



Slika 11: Ikosaedar [24]

Slika 11 pokazuje od čega ćemo krenuti graditi naš ikosaedar, a Slika 13 definira koje točke nam trebaju za izradu ikosaedra.



Slika 13: Točke ikosaedra [25]

```
33          //create the array
34          icoVertices = new Vector3[22];
35          //FACE 0: P0, P11, P5
36          icoVertices[0] = new Vector3(-1, d, 0);//P0
37          icoVertices[1] = new Vector3(-d, 0, 1);//P11
38          icoVertices[2] = new Vector3(0, 1, d);//P5
```

Slika 14: Točke u kodu

Svaka točka u 3D prostoru sastoji se od x, y i z komponente. U ovom slučaju potrebno su 22 točke jer točke na polovima treba duplicirati. To je potrebno tako da je moguće imati 5 različitih UV koordinata ovisno o tome koja je strana. Slika 14 prikazuje implementaciju točaka i niza koji čuva njihove podatke u Unity-u. Niz mora biti tipa Vector3, kao i svaka točka.

```
111         //around point 0
112         icoTriangles = new int[20 * 3];
113         icoTriangles[0] = 0; icoTriangles[1] = 1; icoTriangles[2] = 2;
114         icoTriangles[3] = 3; icoTriangles[4] = 2; icoTriangles[5] = 4;
115         icoTriangles[6] = 5; icoTriangles[7] = 4; icoTriangles[8] = 6;
116         icoTriangles[9] = 7; icoTriangles[10] = 6; icoTriangles[11] = 8;
117         icoTriangles[12] = 9; icoTriangles[13] = 8; icoTriangles[14] = 10;
```

Slika 15: Stranice u kodu

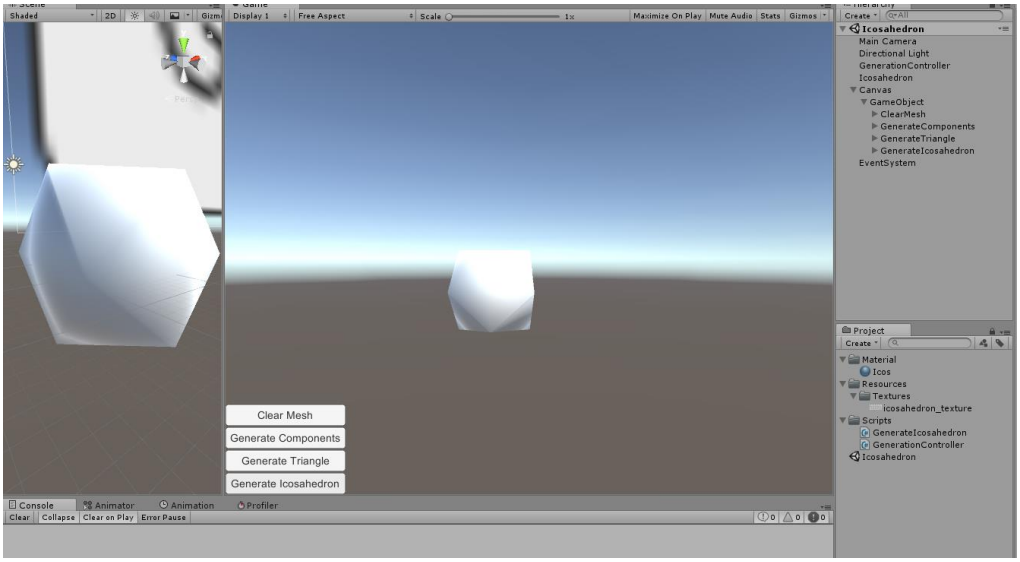
Svaki trokut se sastoji od 3 točke, a na Slici 15 se vidi način implementacije u Unity-u. Veličina niza trokuta koji sprema njihove podatke mora biti višekratnik broja 3, a svaka 3 broja definiraju 3 točke koje čine jedan trokut.

```
37          mesh.RecalculateNormals();
```

Slika 16: Funkcija za računanje normala

Normale nije potrebno ručno dodjeljivati već se mogu izračunati jednostavno uz pomoć Unity-eve funkcije.

Nakon dodjeljivanja novo stvorenog mesh-a praznom GameObject-u. Dolazimo do rezultata prikazanog na Slici 17.



Slika 17: Generirani ikosaedar

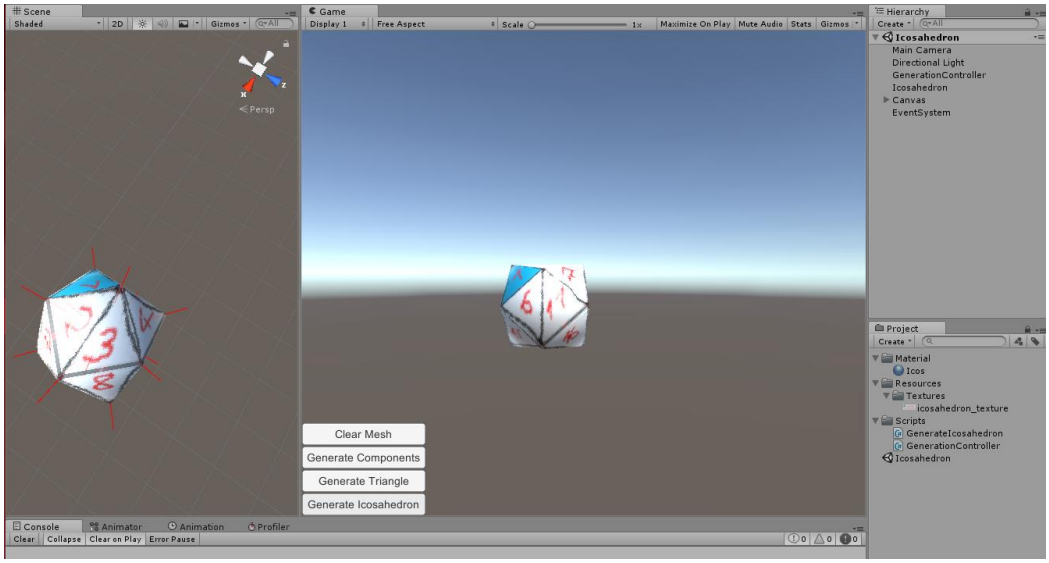
Sad kada smo zasigurno napravili ikosaedar možemo krenuti na generiranje UV koordinata tako da možemo staviti neku teksturu na naš ikosaedar.

```
83          icoUvs = new Vector2[22];
84          icoUvs[0] = new Vector2(0.0909f, 1);//P0
85          icoUvs[1] = new Vector2(0, 0.66f);//P11
86          icoUvs[2] = new Vector2(0.1818f, 0.66f);//P5
```

Slika 18: UV koordinate u kodu

Svaka UV koordinata ima 2 komponente u i v od kojih svaka mora biti u rangu od 0 do 1 gdje ti brojevi označavaju poziciju točke na teksturi. Zbog strukutre ikosaedra računanje tih vrijednosti je zapravo vrlo jednostavno. Na Slici 18 je prikaz zapisa UV koordinata u Unity-u te niz UV koordinata koji mora biti iste duljine kao i niz verteksa.

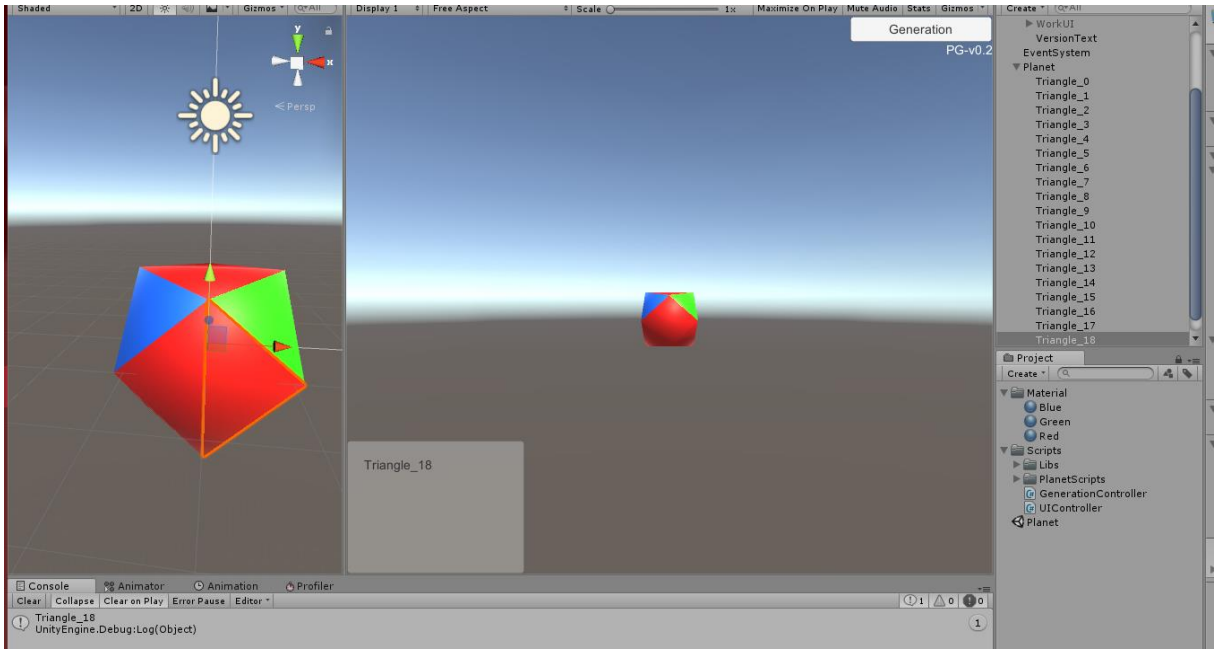
Na Slici 19 je rezultat teksturiranja sa zadanim UV koordinatama.



Slika 19: Teksturirani ikosaedar

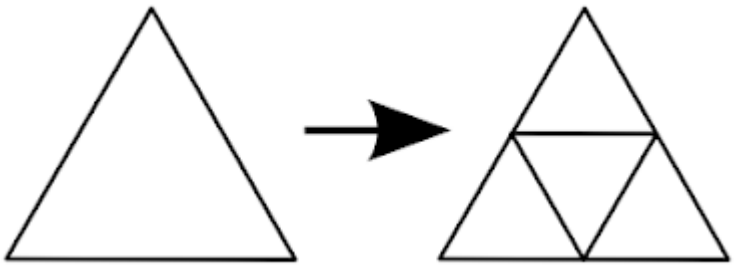
S obzirom da će nam trebati mogućnost polja na našem planetu možemo uzeti svaki trokut našeg ikosaedra i stvoriti od njega novi GameObject. Bez stvaranja GameObject-a ne možemo kliknuti i selektirati neko polje.

Na slici 20 je prikazan rezultat koji se dobije ako svaki trokut zamijenimo GameObjectom i dodamo odgovarajuće komponente kao MeshCollider da ga je moguće selektirati.



Slika 20: Selekcija polja

Kako bi iz ikosaedra dobili ikozaedrenu sferu treba primijeniti postupak subdivizije na svaku od trokuta ikosaedra. Taj postupak se ponavlja nekoliko puta kako bi se došlo do određene razine detalja.



Slika 21: Subdivizija trokuta [25]

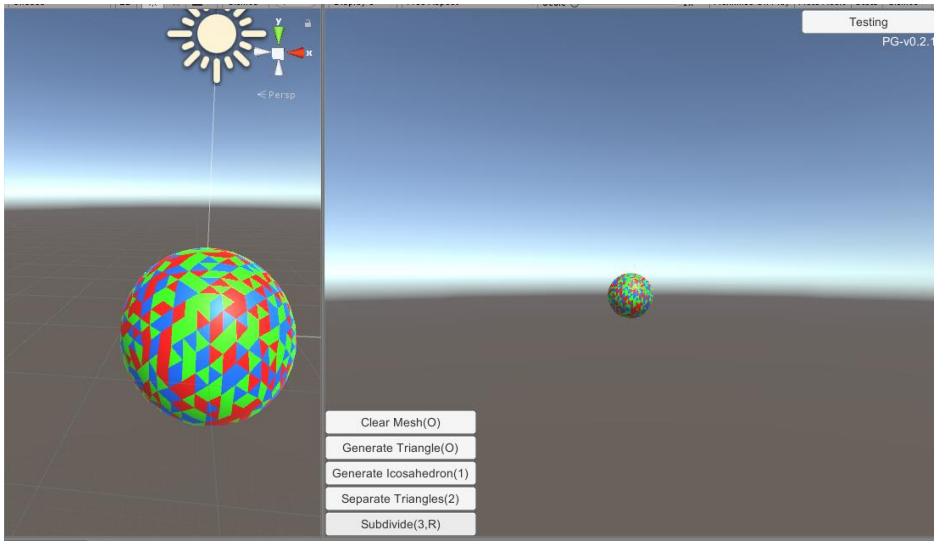
Na Slici 21 je prikazan postupak koji treba primijeniti. Od svakog trokuta treba dobiti 4 nova, no uz to treba primijeniti i zakrivljenost sfere. Zbog toga svaki verteks se treba nalaziti na istoj udaljenosti od središta kako bi dobili zakrivljenost sfere.

```
        }

        //subdivides
        public static void Subdivide(GameObject triangle, Transform parent, Material[] mats)
        {
            //get vertices
            Mesh mesh = triangle.GetComponent<MeshFilter>().mesh;
            Vector3 v1 = mesh.vertices[mesh.triangles[0]];
            Vector3 v2 = mesh.vertices[mesh.triangles[1]];
            Vector3 v3 = mesh.vertices[mesh.triangles[2]];

            //calculate middle points
            Vector3 midLeft = GetMiddlePoint(v1, v2);
            Vector3 midDown = GetMiddlePoint(v2, v3);
            Vector3 midRight = GetMiddlePoint(v3, v1);

            float length = Mathf.Sqrt(v1.x * v1.x + v1.y * v1.y + v1.z * v1.z);
            v1 = new Vector3(v1.x / length, v1.y / length, v1.z / length);
            v2 = new Vector3(v2.x / length, v2.y / length, v2.z / length);
            v3 = new Vector3(v3.x / length, v3.y / length, v3.z / length);

            //add new vertices and triangles
            mesh.vertices = new Vector3[] { v1, v2, v3, midLeft, midDown, midRight };
            mesh.triangles = new int[] { 0, 3, 5, 1, 4, 3, 2, 5, 4, 3, 4, 5 };

            //separate to GOs
            SeparateTrianglesToGOs(triangle, mesh, parent, mats, true, true);
        }

        static Vector3 GetMiddlePoint(Vector3 a, Vector3 b)
        {
            Vector3 mid = new Vector3((a.x + b.x) / 2, (a.y + b.y) / 2, (a.z + b.z) / 2);
            float length = Mathf.Sqrt(mid.x * mid.x + mid.y * mid.y + mid.z * mid.z);
            mid = new Vector3(mid.x / length, mid.y / length, mid.z / length);
            return mid;
        }
    }
}
```

Slika 22: Subdivizija trokuta

S obzirom da više nemamo jedan lik koji se sastoji od više trokuta, već je i svaki trokut za sebe GameObject možemo napraviti subdiviziju svakog trokuta za sebe, te onda opet novo dobivene trokute pretvoriti u GameObject-e. To se radi na svakom koraku tako da korisnik može sam odabrati kad mu je planet dovoljno detaljan. Sam kod načina subdivizije prikazan je na Slici 22.

Nakon nekoliko takvih operacija dobijemo ikosaedrenu sferu prikazanu na Slici 23.



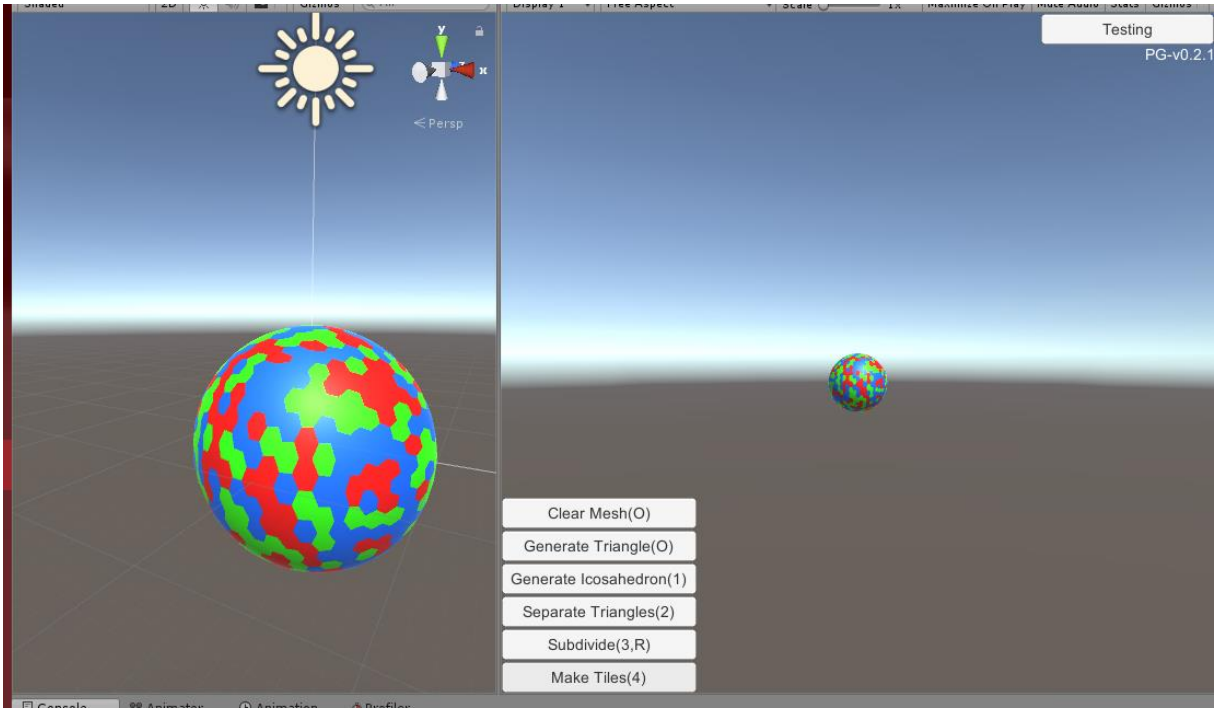
Slika 23: Ikosaedrena sfera

Sada kada napokon imamo sferu s poljima treba popraviti polja. Polja u obliku trokuta se gotovo i ne koriste zbog toga je potrebno grupirati nekoliko trokuta, preciznije 6 ili 5, u jedno polje poligonalnog oblika.

S obzirom da nije rađena nikakva struktura niti graf za praćenje podataka verteksa, rubova i trokuta, najlakši način za dobiti heksagone je bio koristiti se već postavljenim MeshCollider-ima. Odnosno napraviti OverlapSphere od Unity-a na određenim točkama te dobiti trokute i točke koje okružuju tu jednu točku.

Ti podaci su bili dovoljni da se organizira heksagon od prethodnih 6 ili 5 trokuta. Prilikom te pretvorbe uoči se da nisu sve pravilni heksagoni već da ima i nekoliko pravilnih peterokuta, a to je uzrok generiranja ikozaedrene sfere iz ikosaedra.

Tim procesom dobijemo rezultat prikazan na Slici 24.

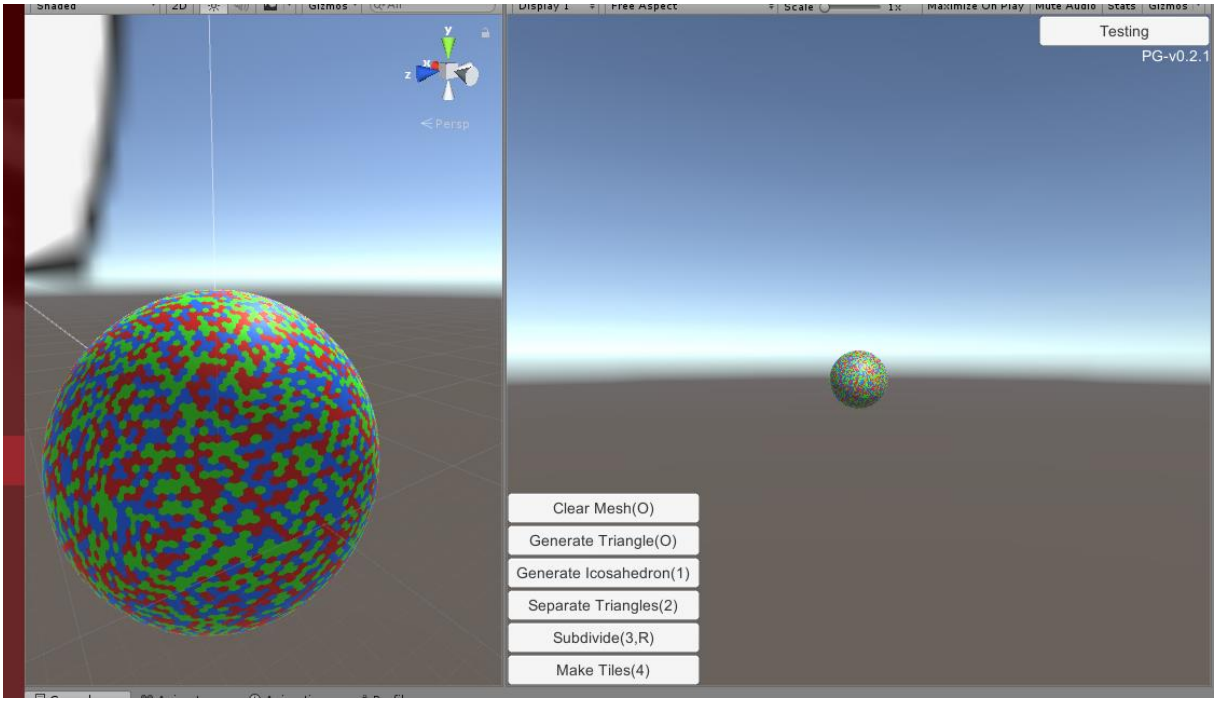


Slika 24: Heksagonalna polja

8.3. Problemi

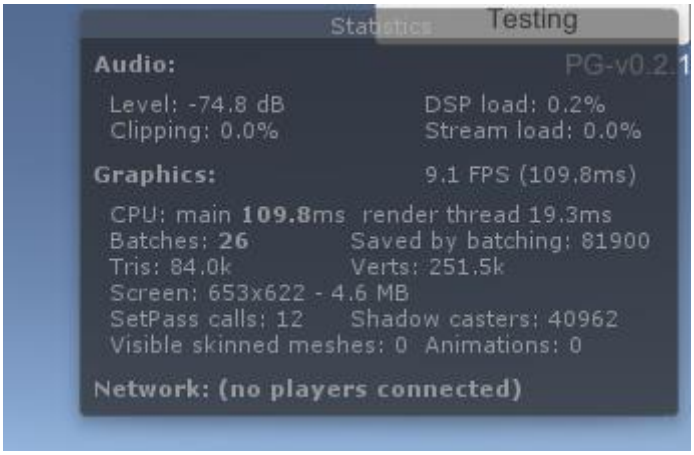
Iako su već bili uočeni i prije, ovaj postupak je počeo imati ogromne probleme već kod subdivizija trokuta. Naime, nakon druge subdivizije proces postane spor, a svakom sljedećom postane sve sporiji tako da se čeka i desetak sekundi na izvođenje subdivizije.

Izvođenje takvom brzinom je bilo jednostavno neprihvatljivo za trenutna računala, a i planet generiran u Rimworld-u nije trajao toliko, a bio je više detaljan od onog generiranog u nekom razumnom vremenu ovom metodom, prikazano na Slici 25.



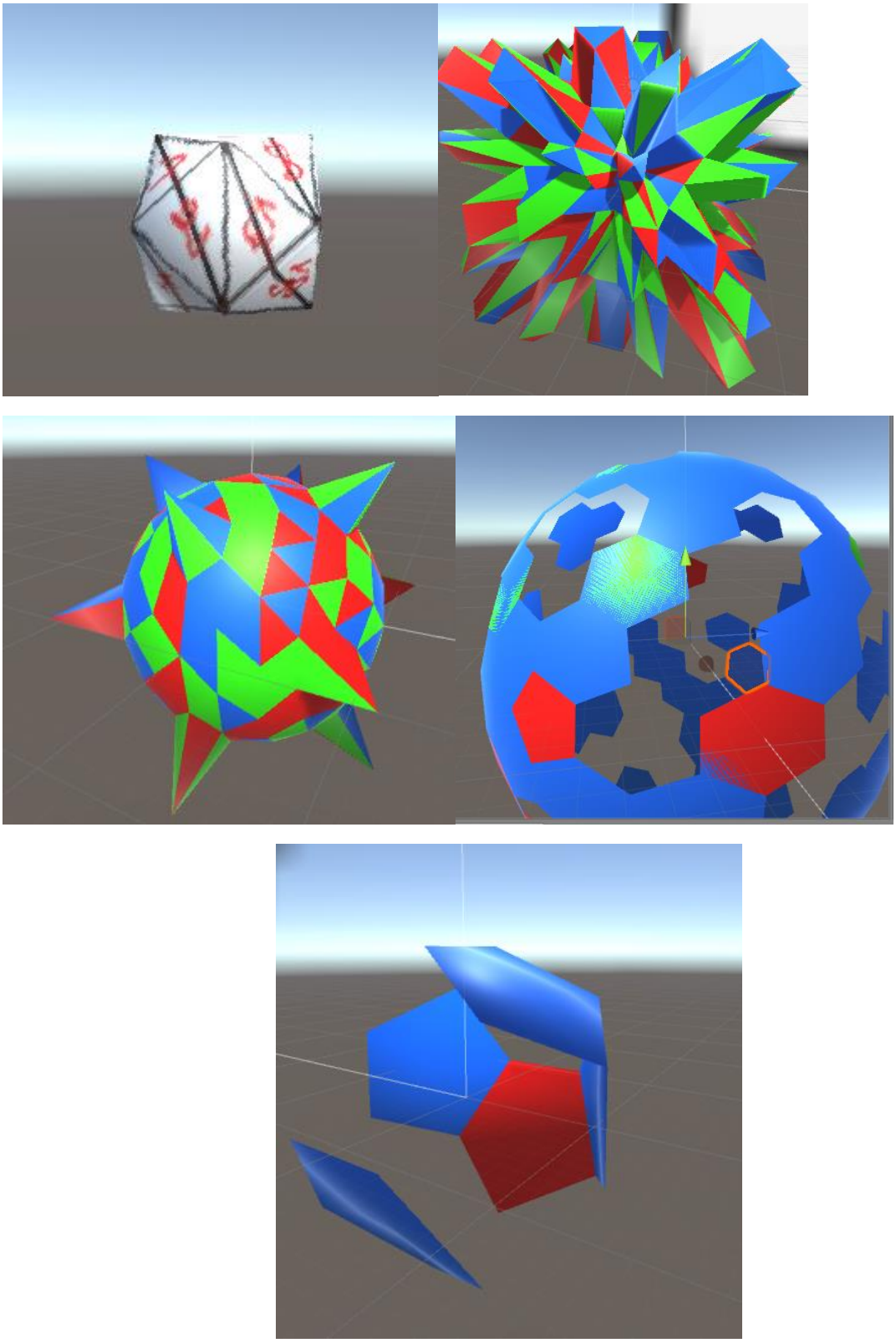
Slika 25: Maksimalni detalji

Nadalje, još veći problem bilo je i izvođenje, pri većim detaljim FPS je bio katastrofalno malen, prikazano na Slici 26.



Slika 26: FPS pri velikom broju subdivizija

Iskustvom krivog procesa generiranja te brojnim greškama tijekom izrade, prikazanih u sljedećim slikama, možemo krenuti u planiranje i izradu boljeg i efikasnijeg generatora planeta.



Slika 27: Greške tijekom izrade

8.4. Novi pokušaj

Ovaj puta umjesto oslanjanja na Unity, napravili smo graf čvorova, rubova i strana koji koristi napravljene i organizirane strukutre.

Kao što je rečeno oslanjati ćemo se na strukturu grafa, a ona će se sastojati od čvorova (eng. Nodes), rubova (eng. Edges) te stranica (eng. Faces).

Prvo definirajmo klasu čvora, on mora znati svoju lokaciju, koje rubove dodiruje i kojim stranama pripada. Implementacija je prikazana na Slici 28.

```
12  public class Node
13  {
14      //point, a point in 3D space
15      public Vector3 point;
16      //what edges is this node a part of
17      public List<int> edges;
18      //what faces is this node a part of
19      public List<int> faces;
20      //UV location
21      public Vector2 UVPoint;
22
```

Slika 28: Klasa čvora

Nadalje moramo definirati klasu rubova. Svaki rub mora biti svjestan o tome koje čvorove spaja, ali isto tako i između kojih strana se nalazi. Također kako bi olakšali subdiviziju kasnije možemo zapisati i podatke o čvorovima i rubovima koji nastaju subdivizijom na njemu. Struktura klase ruba je prikazana na Slici 29.

```
45  public class Edge
46  {
47      //what nodes is this edge connecting
48      public int[] nodes;
49      //what faces is this edge a part of
50      public List<int> faces;
51      //subdivision data
52      public List<int> subdividedNodes;
53      public List<int> subdividedEdges;
```

Slika 29: Klasa ruba

Jedino što nam nedostaje još je klasa strane. Ono što je bitno za tu klasu su čvorovi i rubovi koji pripadaju toj strani. Prikaz koda klase strane je na Slici 30.

```
83  public class Face
84  {
85      //what nodes are a part of this face
86      public List<int> nodes;
87      //what edges are a part of this face
88      public List<int> edges;
```

Slika 30: Klasa strane

Zbog jednostavnosti možemo dodati i klasu planet koja će sadržavati podatke o svim čvorovima, rubovima i stranama koje planet ima. Tako smo omogućili da se prikazuju samo

oni podaci iz grafa koje mi specificiramo, odnosno možemo pustiti neki dio grafa da postoji samo podatkovno. Implementacija je prikazana na Slici 31.

```
110     //defines planet template
111     public class Planet
112     {
113         //planet data
114         public List<Node> nodes;
115         public List<Edge> edges;
116         public List<Face> faces;
```

Slika 31: Klasa planet

Sa svim tim klasamo smo definirali komponente koje će nam u konačnici služiti u grafu da izgradimo planet.

Kao i kod prvog pokušaja, prvi korak kod generacije nam je generacija ikosaedra, već tu će biti promjene jer neće biti dupliciranih čvorova. Na Slici 32 je prikazan unos čvorova, na Slici 33 rubova, a na Slici 34 strana.

```
//set nodes
List<Node> nodes = new List<Node>() {
                   new Node(new Vector3(0, d2, d1), new List<int>(), new List<int>()),
                   new Node(new Vector3(0, d2, -d1), new List<int>(), new List<int>()),
                   new Node(new Vector3(0, -d2, d1), new List<int>(), new List<int>()),
```

Slika 32: Dodavanje čvorova

```
    //set edges
    List<Edge> edges = new List<Edge>() {
                       new Edge(new int[] { 0, 1 }, new List<int>()),
                       new Edge(new int[] { 0, 4 }, new List<int>()),
                       new Edge(new int[] { 0, 5 }, new List<int>()),
                       new Edge(new int[] { 0, 8 }, new List<int>()),
```

Slika 33: Dodavanje rubova

```
List<Face> faces = new List<Face>() {
                   new Face(new List<int>() { 0, 1, 8 }, new List<int>() { 0, 7, 3 }),
                   new Face(new List<int>() { 0, 4, 5 }, new List<int>() { 1, 18, 2 }),
                   new Face(new List<int>() { 0, 5, 10 }, new List<int>() { 2, 21, 4 }),
                   new Face(new List<int>() { 0, 8, 4 }, new List<int>() { 3, 19, 1 }),
```

Slika 34: Dodavanje strana

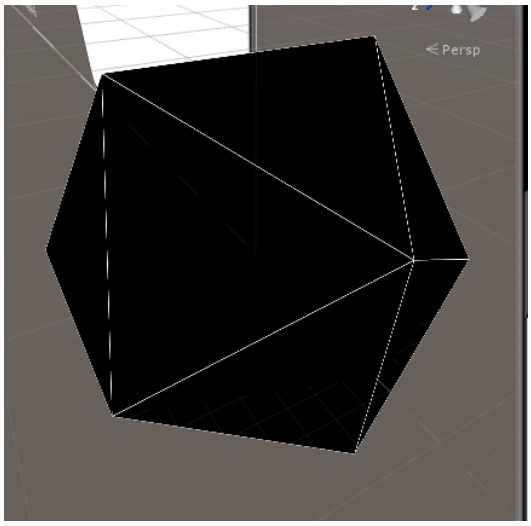
No isto tako kako smo ga stvarali ručno, njegovom formulom, u našim klasama čvorova i rubova nedostaju podaci za međusobno povezivanje. Na Slici 35 je prikazano dodavanje preostalih podataka:

- U čvorovima, koji rubovi dodiruju taj čvor
- U čvorovima, kojim stranama pripada čvor
- U rubovima, kojim stranama pripada rub

```
//fill up missing data from nodes.edges
for (int i = 0; i < edges.Count; i++)
    for (int ii = 0; ii < edges[i].nodes.Length; ii++)
        nodes[edges[i].nodes[ii]].edges.Add(i);
//fill up missing data from nodes.faces
for (int i = 0; i < faces.Count; i++)
    for (int ii = 0; ii < faces[i].nodes.Count; ii++)
        nodes[faces[i].nodes[ii]].faces.Add(i);
//fill up mising data from edges.faces
for (int i = 0; i < faces.Count; i++)
    for (int ii = 0; ii < faces[i].edges.Count; ii++)
        edges[faces[i].edges[ii]].faces.Add(i);
```

Slika 35: Preostali podaci

Korištenjem materijala VR/SpatialMapping/Wireframe, možemo vidjeti točno izgled ikosaedra s njegovim točkama, rubovima i stranama. Prikaz dobivenog lika nakon ovog koda s navedenim materijalom je prikazana na Slici 36.



Slika 36: Wireframe ikosaedar

Nastavljamo istim redosljedom, što znači da je sljedeće subdivizija ikosaedra da dobijemo ikosaedrenu sferu.

Prijašnji način subdivizije rekurzijom je imao mnogo nedostataka, uvijek se svaki rub subdivizijom rastavio na dva, što znači da mi nismo mogli dobiti ni 3, ni 5, ni 7,... rubova od originalnog. Također problem je bio to da je dolazak do određene razine subdivizije, bio uvjetovan dolaskom do svih prethodnih subdivizija.

Kada napravimo subdiviziju mi znamo točno što ćemo dobiti, što bi značilo da se to može organizirati na način da umjesto da se dižemo po razinama subdivizije do one koje želimo, mi možemo odmah izračunati finalnu subdiviziju.

To možemo jer znamo da ako želimo subdiviziju 3 odnosno da se svaki rub rastavi na tri dijela, znamo koliko je svaki dio dug, znamo i da će broj strana biti n^2, i mnoge druge podatke možemo izvuži zbog pravilnosti trokuta a i subdivizije njega.

Proces toga se odvija pronalaženjem točaka unutar trokuta.

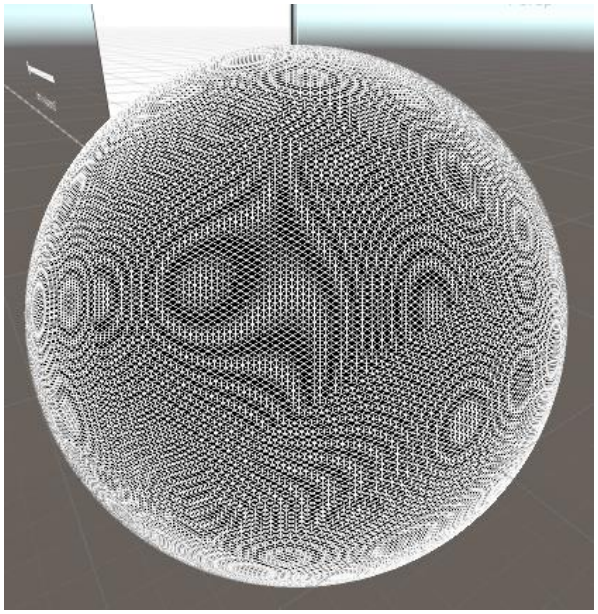
Kako znamo n i znamo da je na glavna 3 ruba trokuta, ako označimo duljinu tog ruba sa a, onda je svaka nova točka na njemu udaljena za a/n.

Ako se počnemo kretati vertikalno po trokutu kako se on sužava, svaka nova razina (definirana točkom na y osi pomještenom za a/n) ima jednu točku manje nego prethodna razina, a razmak između točaka ostaje isti.

Pošto trebamo dobiti zakrivljenost kugle, Unity nam pruža jednu funkciju koja nam točno taj problem rješava, Vector3.Slerp. Ta funkcija uzima dvije točke te postotak puta pređen između njih i automatski nam daje poziciju nove točke na sferi koja bi sadržavala te dvije točke.

Određivanje rubova i strana nakon izračuna svih točaka je samo pitanje organzacije točaka te slaganje istih u rubove i strane.

Korištenjem ove metode, generacija planeta je gotovo instantna, a i razina subdivizije može biti mnogo veća nego kod prethodne metode. Slika 37 prikazuje maksimalnu dopuštenu subdiviziju u generatoru.



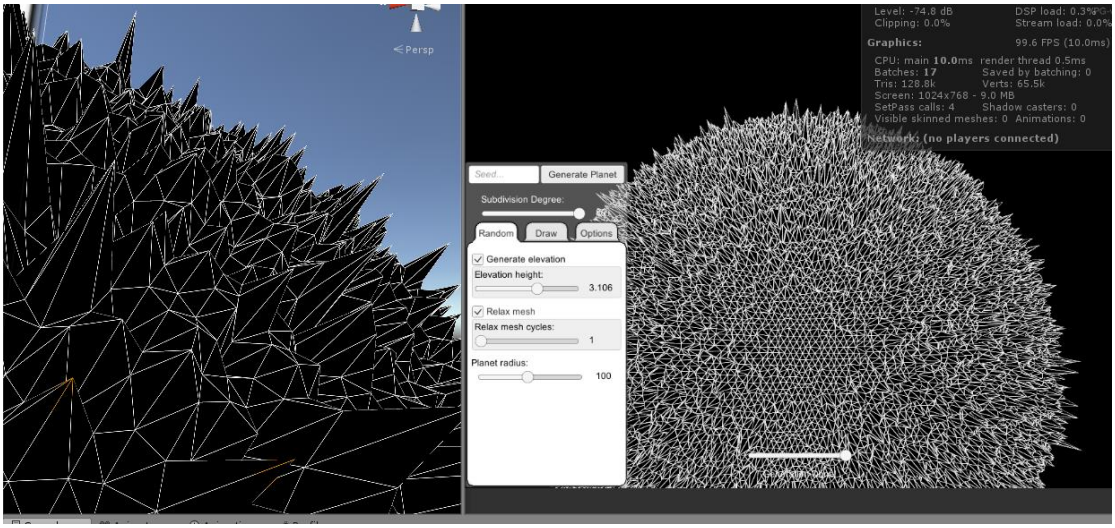
Slika 37: Maksimalna subdivizija

Za razliku od prošle metode, ovaj puta generiranje polja nije potrebno, odnosno zbog same strukture grafa, generiranje polja heksagona i pentagone više nije problematično kao i prije. Zbog toga ćemo krenuti na daljnje kreiranje detalja na planetu poput elevacije.

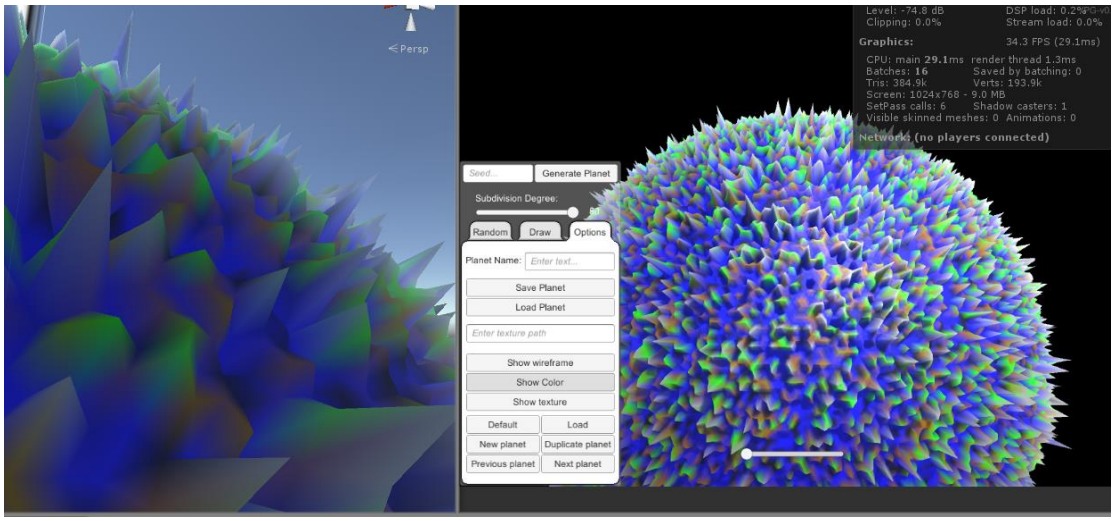
Generacija same elevacije je nešto što se već godinama proučava i radi, a došlo je do izražaja nakon izlaska Minecraft-a [26]. Naime, elevacije se većinom generiraju pseudo-random noise funkcijama poput simplex noise-a. Mi ćemo koristiti jedan od poznatijih, perlin noise.

Perlin noise je tip gradient noise-a [27] koji generira proceduralnu teksturu. Ta tekstura se koristi u vizualnim efektima kako bi se povećao realizam u kompjuterskoj grafici. Ona je pseudo-nasumična funkcija, a kod nje je važno da je lagana za kontrolirati.

Nakon implementacije perline noise-a na svaku točku naše sfere dobijemo rezultat prikazan na Slici 38. Također za svaku točku smo dodali boju ovisno o visini na planeti, čiji se prikaz može vidjeti na Slici 39.



Slika 38: Elevacija planeta



Slika 39: Boja elevacije

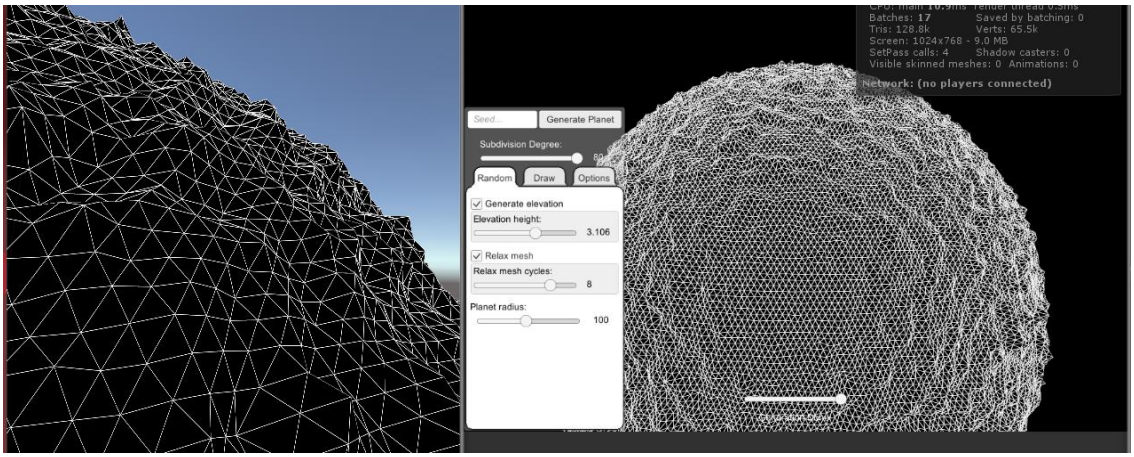
Kao što se može vidjeti planet izgleda previše radikalno, odnosno skok između točaka je prevelik. Umjesto da popravljamo i mijenjamo perlin noise, možemo implementirati tehniku zvanu relax mesh.

Ta tehnika je vrlo jednostavna, uzme se točka i njena okolica te se uzima prosjek toga kao njena nova vrijednost. Tu je jedna stvar bitna, a to je da se utjecaj svake točke može mijenjati pa tako da to bude postepena funkcija utjecaj okolnih točaka je samo 20%. Implementacija kroz broj ciklusa je prikazana na Slici 40.

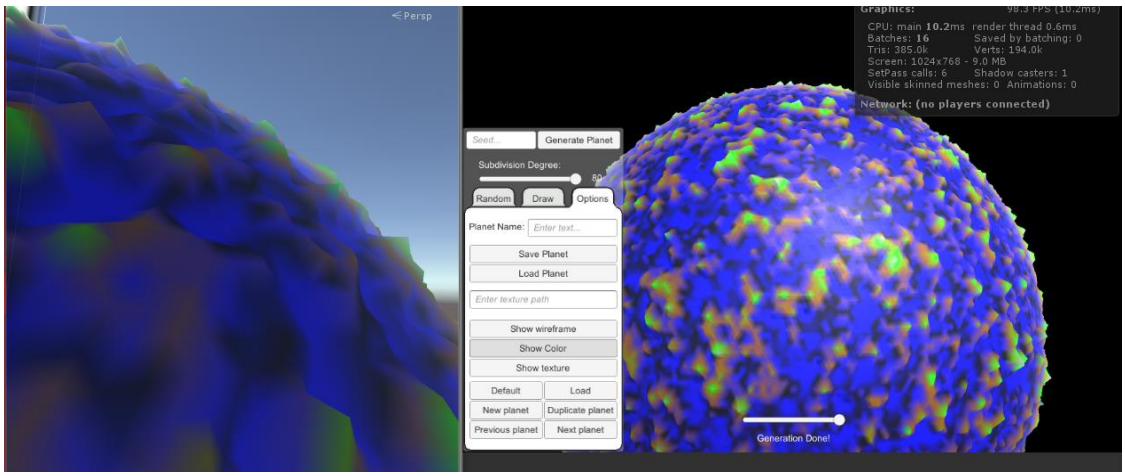
```
//start making new planet
Planet oldPlanet = p;
Planet newPlanet = p;

for(int cycles = 1; cycles <= PGHelper.relaxCycles; cycles++)
{
    //relax
    foreach (Node n in newPlanet.nodes)
    {
        Vector3 sumPoint = Vector3.zero;
        //add all points around
        foreach (int i in n.edges)
        {
            for (int ii = 0; ii < oldPlanet.edges[i].nodes.Length; ++ii)
            {
                Node other = oldPlanet.nodes[oldPlanet.edges[i].nodes[ii]];
                if (other != n)
                {
                    sumPoint += other.point * 0.2f / n.edges.Count;
                }
            }
        }
        //add myself
        sumPoint += n.point * 0.8f;
        //save
        n.point = sumPoint;
    }
    oldPlanet = newPlanet;
}
```

Slika 40: Relax mesh

Kao rezultat uz iste parametre prošlog primjera uz 8 ciklusa relax mesh procesa dobiven je planet prikazan na Slici 41. Na slici 42 je isti primjer samo su prikazane boje.

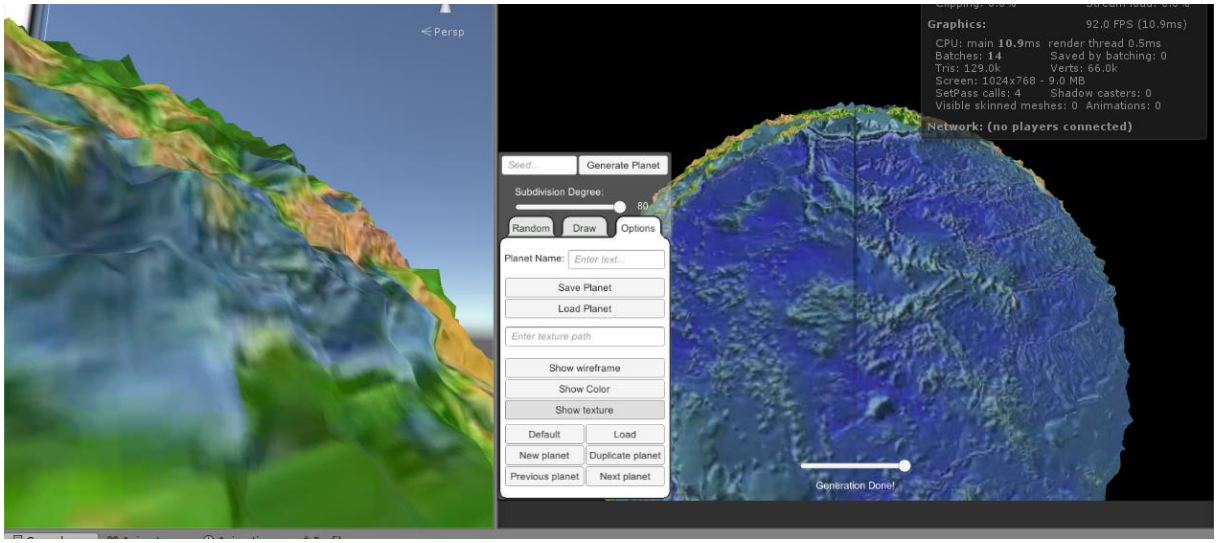


Slika 41: Planet nakon 8 ciklusa relax mesh-a



Slika 42: Obojani planet nakon 8 ciklusa relax mesh-a

Posljednju stvar koju ćemo još dodati je tekstura na planet. Koristeći Cubemap shader [28] samo teksturiranje ikozaedrene sfere, UV koordinate nije potrebno niti računati već sve prepustimo Unity-u. Na slici 43 je prikaz prijašnje generiranog planeta s teksturom zemlje.

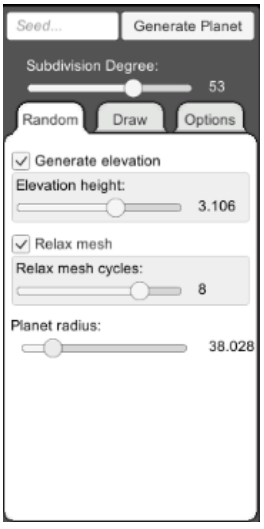


Slika 43: Prikaz teksture

8.5. Sučelje

Kako je sam generator planeta napravljen kao biblioteka u C#, za rad i prikaz mogućnosti potrebno je bilo napraviti sučelje. Kao što se moglo vidjeti u prethodnim slikama, sučelje prikazuje maksimalne mogućnosti koje je trenutno moguće dobiti bez dodavanja novih funkcija.

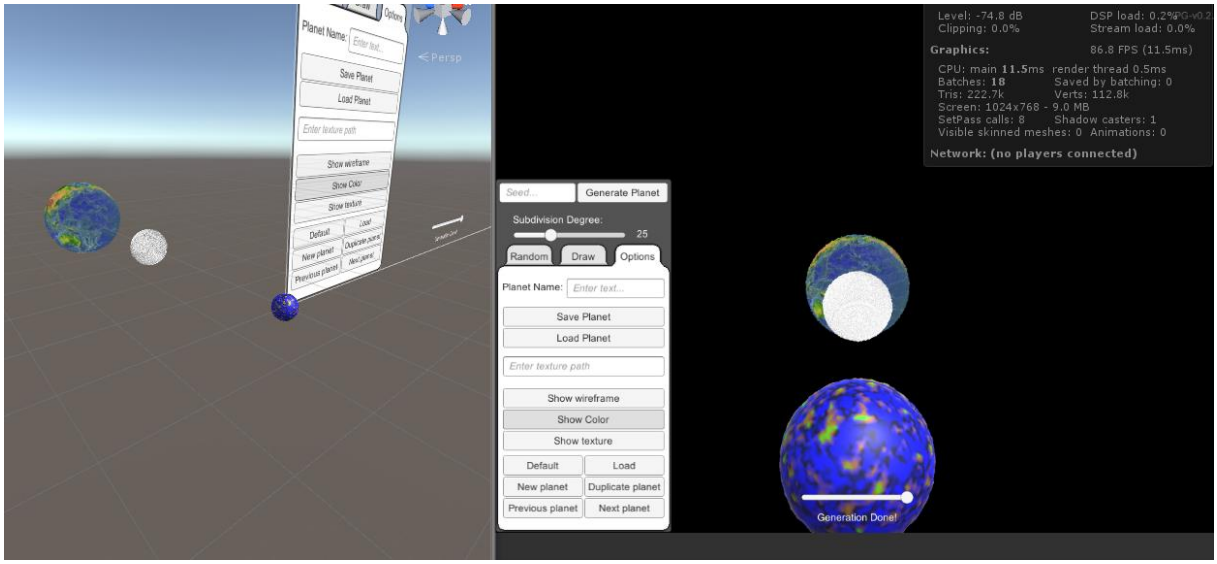
Opcije koje se nude kod same generacije su: odabir subdivision-a planeta te seed za pseudo-random stvari. Odabir generacije elevacije te visinu, odabir relax mesh-a te broj ciklusa, odabir radijusa planeta. Prikaz tih opcija je na Slici 44.



Slika 44: Random UI

Opcije koje se nude kod opcija je spremanje planeta u datoteku i čitanje iz datoteke, odabir teksture dinamički koji trenutno funkcionira samo unutar Unity Editora, odabir prikaza planeta ako će to biti wireframe, boja ili tekstura. Prikaz default-ne teksture koja je tekstura

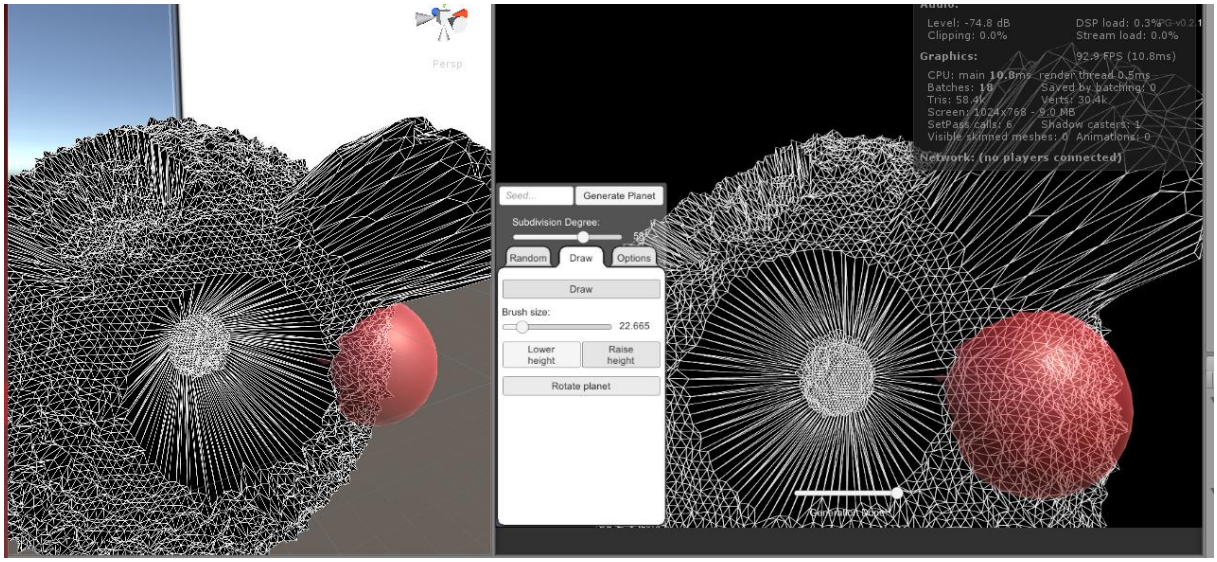
zemlje ili učitavanja neke druge. Također moguća je kreacija više planeta te ih istovremeno imati prikazanih, kao što je prikazano na Slici 45. A prikaz tih opcija je na slici 46.



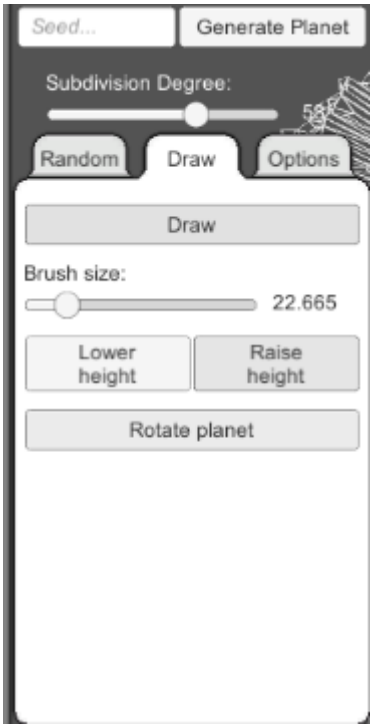
Slika 45: Prikaz više planeta s različitim opcijama

Slika 46: Prikaz opcija UI

Opcije koje se nude korisniku za daljnje definiranje planeta su crtanje po planetu tako da definira veličinu svog kista. Dohvat kista prikazan je prozirnom sferom, a nudi se smanjivanje i povećanje visine planeta. Nadalje također postoji opcija da se planet rotira. Prikaz korištenja opcija je na Slici 47, a na slici 48 je prikaz opcija.



Slika 47: Crtanje po planeti



Slika 48: Prikaz opcija crtanja

8.6. Korištenje

Za ubacivanje u već postojeći Unity projekt, odnosno neku igru, prvo što je potrebno je definirati varijablu tipa Planet koja je definirana u biblioteci, prikaz deklaracije je na Slici 50. Također je potrebno uključiti biblioteku PG, što je skraćeno od planet generator što je prikazano na Slici 49.

```
using PG;
```

Slika 49: Uključivanje biblioteke PG

```
Planet planet;
```

Slika 50: Varijabla tipa Planet

Kada smo pripremili varijablu te uključili biblioteku, dvije funkcije koje se obavezno trebaju zvati su GenerateIcosahedron i GenerateSubdividedIcosahedron. Obje funkcije se nalaze u PGGeneration unutar biblioteke PG. Druga funkcija prima dva argumenta već generirani ikosaedar iz prve funkcije te broj subdivizija koji može biti po želji. Prikaz obaveznih funkcija te njihov redoslijed zvanja prikazan je na Slici 51.

```
//init planet
planet = PGGeneration.GenerateIcosahedron();
//subdivide
planet = PGGeneration.GenerateSubdividedIcosahedron(planet, subdivisonDegree);
```

Slika 51: Obavezne funkcije

Nakon tog koraka funkcije za generaciju elevacije, relax mesh, generaciju podataka za Unity objekt, spremanje i čitanje se može zvati bilo kada i u bilo kojem redoslijedu.

Generacija elevacije se izvodi kroz dva koraka:

- Prvi korak je postavljanje vrijednosti visine planeta (Slika 52)
- Drugi korak je pozivanje funkcije za generaciju elevacije (Slika 53)

```
PGHelper.height = s.value;
```

Slika 52: Postavljanje vrijednosti visine planeta

```
planet = PGGeneration.GeneratePlanetElevation(planet);
```

Slika 53: Funkcija za elevaciju planeta

Na isti način se poziva funkcija za relax mesh-a. Prvo postavimo koliko ciklusa želimo (Slika 54), a odmah nakon zovemo funkciju (Slika 55).

```
PGHelper.relaxCycles = (int)s.value;
```

Slika 54: Postavljanje broja ciklusa

```
planet = PGTopology.RelaxMesh(planet);
```

Slika 55: Poziv funkcije za relax mesh-a

Spremanje se odvija pozivom funkcije na Slici 56 koja prima argumente: putanja i naziv datoteke te planet koji se sprema.

```
PGUtilities.SavePlanet(nameInputField.text, planet);
```

Slika 56: Spremanje planeta

Dok za čitanje planete je potrebna samo putanja do datoteke (Slika 57).

```
planet = PGUtilities.LoadPlanet(nameInputField.text);
```

Slika 57: Čitanje planeta

Nakon poziva svih funkcija generacije planeta, poziva se nekoliko funkcija koje služe kao sakupljači i oblikovatelji podataka u obliku koji to paše Unity objektima za prikaz. Na Slici 58 je prikazan poziv svih potrebnih funkcija za dohvaćanje informacija.

```
vertices = PGUtilities.GetVerticesFromPlanet(planet);
triangles = PGUtilities.GetFacesFromPlanet(planet);
normals = PGUtilities.GetNormalsFromPlanet(vertices);
colors = PGUtilities.GetColorsForVertices(vertices);
```

Slika 58: Dohvaćanje informacija o planetu

Sve te informacije samo treba preslikati u mesh novog Unity objekta kako bi se on pojavio na ekranu.

9. Zaključak

U ovom radu pokazan je jedan od načina proceduralnog generiranja planeta uz pomoć grafova i klasa. Proceduralno generiranje mijenja način razvoja igara, a i same igre njegovom mogućnošću izrade novog i drugačijeg sadržaja svakim pozivom. Mnogi sadržaji se mogu generirati proceduralno, samo je bitan odabir načina i pristupa proceduralnom generiranju ovisno o potrebi.

Ostvaren je cilj stvaranja proceduralnog generatora planeta je ostvaren, iako proces pisanja programa nije bio uspješan iz prve jer je bilo potrebno testirati i isprobavati različite funkcije za generairanje planeta. Na kraju dobiveni rezultati bolji su od onih koje smo na samom početku mogli i zamisliti.

Veliku ulogu u procesu generiranja planeta je imala matematika te njena aplikacija na 3D prostor. Organizacija podataka u klase i strukture, te korištenje grafova za stvaranje bilo je novo iskustvo koje se pokazalo kao dobar put. Zabava i zadovoljstvo kada program proradi i generira nešto, je neopisiva i odličan motiv za nove izazove povezane za proceduralno generiranje sadržaja.

Plan je nastaviti razvijati ovu ideju proceduralnog generiranja planeta temeljenu na grafovima, a prvi korak bi bio omogućiti generaciju i selekciju polja. Također, ideju korištenja grafova kod proceduralne generacije primijeniti na drugim objektima sa svojim unikatnim karakteristikama i svojstvima.

10. Popis slika

11. Popis priloga

Uz ovaj rad kao prilog priložen je CD na kojem se nalazi ovaj rad, cijeli Unity projekt, te build-ana aplikacija istog unity projekta.

12. Literatura

[1] https://rimworldgame.com/, pristupano 5.7.2018.
[2] https://en.wikipedia.org/wiki/Indie_game, pristupano 5.7.2018.
[3] https://steamcommunity.com/games/294100/announcements?p=3, pristupano 6.7.2018.
[4] http://www.uberent.com/pa/, pristupano 3.7.2018.
[5] https://forums.uberent.com/threads/wip-custom-planet-origin.65260/, pristupano 3.7.2018.
[6] https://en.wikipedia.org/wiki/Perlin_noise, pristupano 4.7.2018.
[7] http://pcgbook.com/, pristupano 4.7.2018.
[8] https://en.wikipedia.org/wiki/Dungeon_crawl, pristupano 5.7.2018.
[9] https://us.diablo3.com/en/, pristupano 8.7.2018.
[10] https://www.diablowiki.net/Randomization, pristupano 8.7.2018.
[11] http://www.necessarygames.com/reviews/sid-meiers-civilization-iv-game-commercial-mac-os-x-windows-turn-based-strategy-historical, pristupano 24.6.2018.
[12] https://www.spacesector.com/blog/2013/05/sins-of-a-solar-empire-rebellion-dlc-%E2%80%9Cforbidden-worlds%E2%80%9D/, pristupano 24.6.2018.
[13] http://clivebest.com/blog/?p=7820, pristupano 24.6.2018.
[14] https://www.gamedev.net/forums/topic/662284-spherified-cube/, pristupano 24.6.2018.
[15] https://en.wikipedia.org/wiki/UV_mapping, pristupano 5.7.2018.
[16] https://dorit.mercatodos.co/latitude-lines/, pristupano 24.6.2018.
[17] https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/3DSMax/files/GUID-78327298-4741-470C-848D-4C3618B18FCA-htm.html, pristupano 24.6.2018.
[18] https://unity3d.com/, pristupano 24.6.2018.
[19] https://en.wikipedia.org/wiki/Cross-platform, pristupano 6.7.2018.
[20] https://en.wikipedia.org/wiki/Game_engine, pristupano 6.7.2018.
[21] https://en.wikipedia.org/wiki/Unity_(game_engine), pristupano 24.6.2018.
[22] https://en.wikipedia.org/wiki/List_of_Unity_games, pristupano 6.7.2018.
[23] https://en.wikipedia.org/wiki/Polygon_mesh, pristupano 8.7.2018.
[24] https://en.wikipedia.org/wiki/Regular_icosahedron, pristupano 6.7.2018.
[25] http://blog.andreaskahler.com/2009/06/creating-icosphere-mesh-in-code.html, pristupano 25.6.2018.
[26] https://minecraft.net/en-us/, pristupano 26.6.2018.
[27] https://en.wikipedia.org/wiki/Gradient_noise, pristupano 26.6.2018.
[28] https://forum.unity.com/threads/cube-mapped-sphere-aka-quad-sphere-asset.292860/, pristupano 27.6.2018.