Izrada 3D RPG igre sa proceduralnim generiranjem svijeta u Unity razvojnom okruženju

Grubeša, Luka

Sveučilište Jurja Dobrile u Puli
Tehnički fakultet u Puli

LUKA GRUBEŠA

Izrada 3D RPG igre sa proceduralnim generiranjem svijeta u Unity razvojnom okruženju

Završni rad

Pula, rujan, 2022.

Sveučilište Jurja Dobrile u Puli
Tehnički fakultet u Puli

LUKA GRUBEŠA

Izrada 3D RPG igre sa proceduralnim generiranjem svijeta u Unity razvojnom okruženju

Završni rad

JMB: 030309343, redoviti student

Studijski smjer: Sveučilišni preddiplomski studij računarstvo

Predmet: Programiranje

Znanstveno područje: Tehničke znanosti

Znanstveno polje: Računarstvo

Znanstvena grana: Programsko inženjerstvo

Mentor: Izv.prof.dr.sc Tihomir Orehovački

Pula, rujan, 2022.

IZJAVA

o akademskoj čestitosti

Ja, dolje potpisan, Luka Grubeša, kandidat za prvostupnika računarstva ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu 3iterature kao što to pokazuju korištene bilješke I bibliografija. Izjavljujem da ni jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

IZJAVA

o korištenju autorskog djela

Ja, Luka Grubeša, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom:

„Izrada 3D RPG igre sa proceduralnim generiranjem svijeta u Unity okruženju“ koristi tako da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne I sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu I drugim srodnim pravima I dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

Sadržaj

1. Uvod 1

2. Analiza igara koji su služili kao inspiracija 2

3. Unity razvojno okruženje 4

3.1 Opis alata 4

3.2 Unity uređivač 4

3.3 Animator 9

3.4 Oblikovanje terena 10

3.5. Borba 12

3.6. AI neprijatelja 12

3.7. Sistem oružja 16

3.8 Sustav čestica 18

3.9 Stvaranje neprijatelja 18

4. Proceduralno generiranje terena u Unity-u 20

4.1 Perlinova funkcija šuma 21

4.2 Mesh 22

4.3 Beskonačna generacija terena 23

4.4 Dodavanje tekstura 25

5. Dodavanje glavnog izbornika i UI komponente 27

5.1 Glavni izbornik 27

5.2 Korisničko sučelje 27

5.3 Kraj igre 28

5.4 Glazba 29

6. Tijek igre 30

7. Zaključak 32

8. Literatura 33

Popis slika 34

Sažetak 35

Abstract 36

1. Uvod

Izrada računalnih igara postao je nevjerojatno skup i dugotrajan proces u kojem velike specijalizirane tvrtke poput Ubisofta, Square Enixa, Nintenda i Sege stvaraju jedinstvena interaktivna, vizualna i auditivna iskustva koja su igrana od strane milijuna igrača, no to ne znači da za izraditi računalnu igru je potreban tim developera i masovni budžet. Mnoge tvrtke napravili su programe za izradu igrica kako bi olakšali taj proces te upoznali mnoge ljude s alatima da svoje ideje pretvore u konkretne oblike.

Tema ovog završnog rada jest izrada 3D RPG igrice sa proceduralnim generiranjem svijeta u Unity razvojnom okruženju. RPG (eng. Role-Playing Game), širok je termin kojem se obično prepisuju mehanike jačanja igrača kako igra napreduje, u obliku „levela“, varijabla koja pokazuje koliko je igrač snažan, te mehanika odabira i korištenja dostupnih oružja, gdje svako oružje ima druge atribute koji ih čini posebnima.

RPG Runner se sastoji od dvije glavne scene; jedna u kojoj se obavlja odabir oružja, tj. klase, te druga u kojoj je cilj proći što veću udaljenost u beskonačno generiranom svijetu dok igrača napadaju neprijatelji. Igrač ima na raspolaganju odabrano oružje te dobiva dodatne bodove za ubijanje neprijatelja koji postaju sve jači što je igrač dalje od početne pozicije. Pri ubijanju neprijatelja igrač dobiva levele, koji ga čine jačima te ga i izliječe. Glavna inspiracija za izradu ove igre je bio Minecraft, videoigra koja je poznata po svom proceduralno generiranim svijetom, dok u druge inspiracije ubrajamo Temple Run, Subway Surfers i druge igre u kojem je svrha preživjeti što duže, te JRPG („Japanese role-playing game“) igre poput Tales of Berseria. Glavna ideja ove igre je bila da se napravi beskonačan svijet u kojem igrač treba preživjeti što dulje dok ima opciju slobodnog kretanja u 3D prostoru i dinamičkih borba koje pronalazimo u mnogim JRPG igrama, dok se bori ili izbjegava neprijatelje koji postaju jači i različitiji što je dalji od početne točke. Neki neprijatelji imaju više života ili daju veću štetu igraču, pa je na igraču da odabere prave neprijatelje u pravom trenutku.

Igra je napravljena u okruženju Unity, besplatnom programu u kojem su napravljene mnoge igre, malih i velikih budžeta, iako ga većinom koriste nezavisni razvijači igara, te početnici radi njegovog intuitivnog rasporeda programa, mnoštvo dodataka i YouTube videa na kojem novi programeri se mogu naučiti koristiti programom. Unity također pruža uslugu prodaje i kupnje sredstava za izradu igara pomoću Unity Asset Store-a, od kojih postoji mnoštvo besplatnih sredstava koji su uvedeni u RPG Runner, poput modela neprijatelja, kuća, biljaka i efekata koji su napravljeni u Unity-ovom popularnom sustavu kreiranja čestica.

2. Analiza igara koji su služili kao inspiracija

Najveća inspiracija za izradu ove igre je Minecraft, to jest njezin aspekt beskonačno generiranog terena, te način na koji se proceduralno generiraju neprijatelji. Ono što čini igru tako posebnom jest mogućnost igrača da rudari i izradi nove objekte, oružja itd. Iako igra ima sistem borbe, ona nije glavni fokus, nego preživljavanje. RPG Runner je iskoristio koncept preživljavanja u beskonačnom svijetu kao glavni mehanizam igre, no dodaje posebne efekte i napade, te sistem resursa u obliku levela i mane, gdje je igračeva dužnost držati balans između borbe i preživljavanja. Korisničko sučelje inspirirano je RPG igricama poput „The Elder Scrolls V: Skyrim“, specifično pozicija i animacija oružja koje igrač koristi, vidljivo na slici 1. U čemu se ove dvije igre razlikuju jest u prikazivanju glavnih informacija o igraču. „Skyrim“ koristi minimalističan pristup korisničkom sučelju, jedino prikazujući pojedine glavne podatke kada im se promjeni vrijednost, dok RPG Runner ima ih prikazane u svakom trenutku, zbog same brzine igre, dok je „Skyrim“ uglavnom atmosferična igra koja želi igraču prikazati veličinu svijeta oslobađajući igrača od bilo kakvih nepotrebnih informacija u tom trenutku, da bi se moglo prikazati više samog svijeta.

Pri dizajniranju igre trebao se odrediti cilj za igrača, budući da Minecraft nema pravi kraj igre, dodali smo „runner“ mehaniku u igru, gdje je glavni cilj igre preživjeti što dulje moguće te pri završetku igre, tj. kada igraču ponestane života, sistem izračuna krajnje bodove koji je igrač dobio. Takav sistem bio je inspiriran igricama poput „Temple Run“ i „Subway Surfers“.

Izgled igre inspiriran je minimalističkim dizajnom svijeta pojedinih JRPG igara, podskupinom RPG igara koji su napravljeni u Japanu, poput „Tales of Berseria“ i „Hyperdimensional Neptunia“, gdje se fokus igre orijentira uglavnom na borbu i priču, dok dizajn okruženja ponekad zna biti manji prioritet.

Slika 1. Prikaz sučelja igre “The Elder Scrolls V: Skyrim”

RPG Runner inspiriran je mnogim žanrovima igara, pogotovo RPG igrama, no glavna razlika među njima jest njezini sistem održavanja resursa. U većini RPG igara igrač ima mnoštvo resursa koje može iskoristiti, a kada mu ponestanu u većini slučajeva postoje nekakva sigurna mjesta gdje ih može ponovno nabaviti, dok u RPG Runneru igrač jedino može vratiti energiju ubijajući neprijatelje ili pronalaskom posebnih sfera koji mu povećavaju život. Tako igra daje igraču koncept „High-risk, high-reward“ sistema, gdje rizičniji potezi dovest će prosječnog igrača dalje u igri nego pasivniji, budući da neprijatelji postaju jači što veću udaljenost igrač pređe.

3. Unity razvojno okruženje

3.1 Opis alata

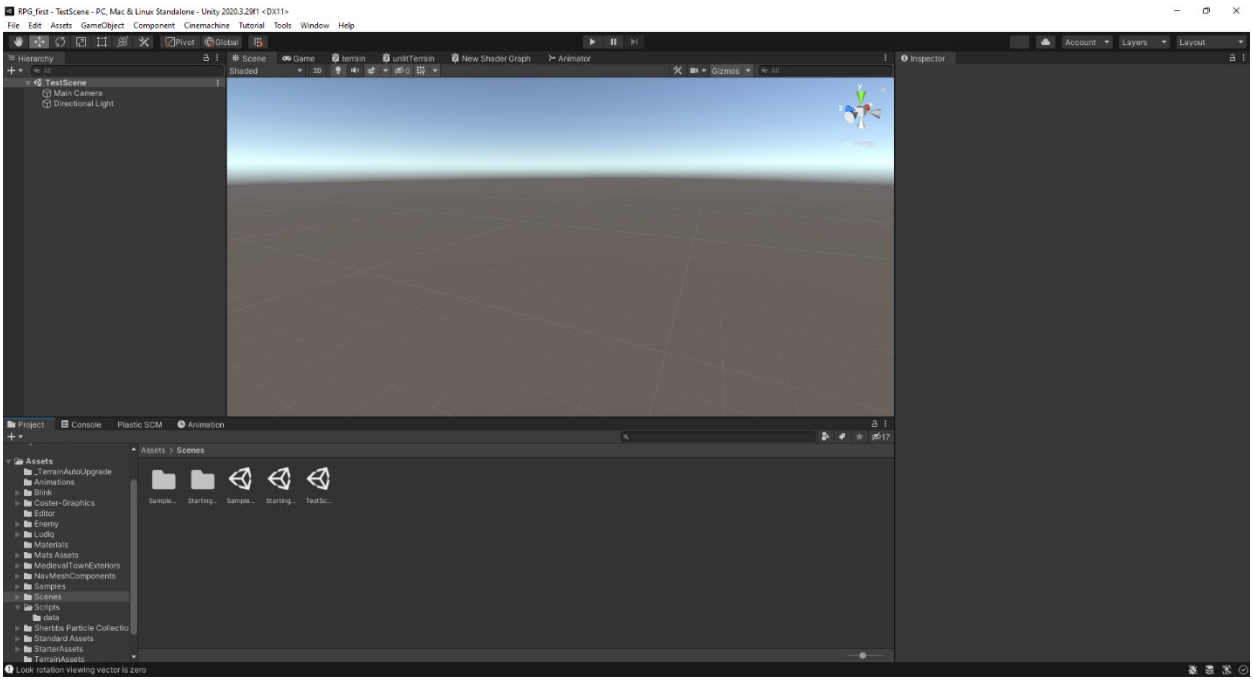
Unity je jedan od najpopularnijih alata za izradu igara, ponajprije zbog njegove lakoće korištenja, a i radi mnoštvo informativnih videa koji su dostupni na različitim platformama, kao na YouTube-u, u kojem Unity aktivno sponzorira i pomaže kreatorima da proizvedu informativne i kreativne videe.

U ostale prednosti Unity-a uključujemo [9]:

- Brzo stvaranje i prototipiranje igara
- Unity asset store gdje se mogu nabaviti mnoštvo besplatnih i kvalitetnih 3d modela, efekata, alata te gotovih scena.
- Prikazivanje i mijenjanje pojedinih vrijednosti varijabla nekog objekta pomoću play mode-a.
- Podržavanje mnoštvo platformi za pokretanje igrice (PC, Android, iOS, PS, XBOX, VR)

3.2 Unity uređivač

Pri otvaranju novog projekta vidljiva je prazna scena, kao što je prikazano na slici 2. Na njoj možemo vidjeti prozor za pregled igre gdje korisnik može uređivati i dodavati objekte u scenu, ili pokrenuti igru u tzv. Play mode-u, gdje korisnik može upravljati igrom kao u završenoj verziji. Play mode posebno je važan radi pronalaska vizualnih i skriptnih pogrešaka koje prolaze bez greške kada kompiliramo kod, ali ne daju željene rezultate u samoj igri.

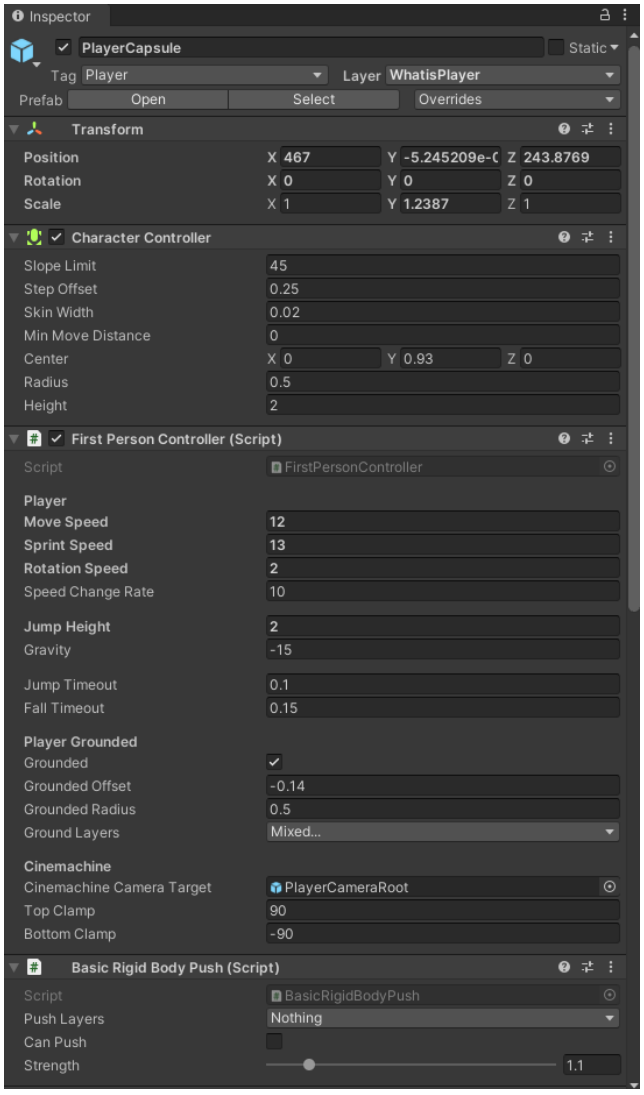


Slika 2. Prikaz prazne scene u Unity-u.

U donjem dijelu ekrana nalazimo ponajprije karticu s mapom projekta gdje se nalaze svi objekti, skripte, zvučne datoteke, animacije i slično. Primjer jedne scene je prikazan na slici 3. Jedna od prednosti Unitya jest njegova modularnost; svaku kartica i prozor možemo premještati i preuveličavati radi bolje preglednosti u određenom trenutku, što je posebice korisno kada želimo pregledati probleme sa modelima ili animacijama u posebnom prozoru dok je igrica u Play mode-u. Kao primjer, možemo pomaknuti kartice scene (eng. Scene) i igre (eng. Game) u isti prikaz, gdje bi obično bile odvojene, kako bi igrač mogao ispraviti sve probleme koji nisu vidljivi u prikazu igre, ali su vidljivi u prikazu scene.

Slika 3. Primjer scene sa mapom projekta

Na desnom dijelu ekrana nalazi se inspektor, vidljiv na slici 4, u kojem možemo pregledati i mijenjati postavke za sve varijable pozicije i ponašanja objekta u sceni, uključujući svojstva dodanih komponenta, skripta, materijala i dr.



Slika 4. Primjer inspektora za objekt Igrača.

Prikaz hijerarhije objekta u sceni nalazi se na lijevoj strani ekrana gdje prikazuje svaki objekt koji je trenutačno u sceni, npr. Kamera, igrač, prefab[1] kuće itd...Ti objekti mogu sadržavati druge objekte u njima pod relacijom parent-child[2]. Te objekte možemo detaljnije pregledati pomoću inspektora ili manipulirati njihovim redoslijedom u hijerarhiji. Primjer hijerarhije objekata jedne scene vidljiv je na slici 5.

[1] Prefab je bilo koji objekt sa svim svojim komponentama, vrijednostima i podređenim objektima koji se koristi kao ponavljajuće sredstvo.

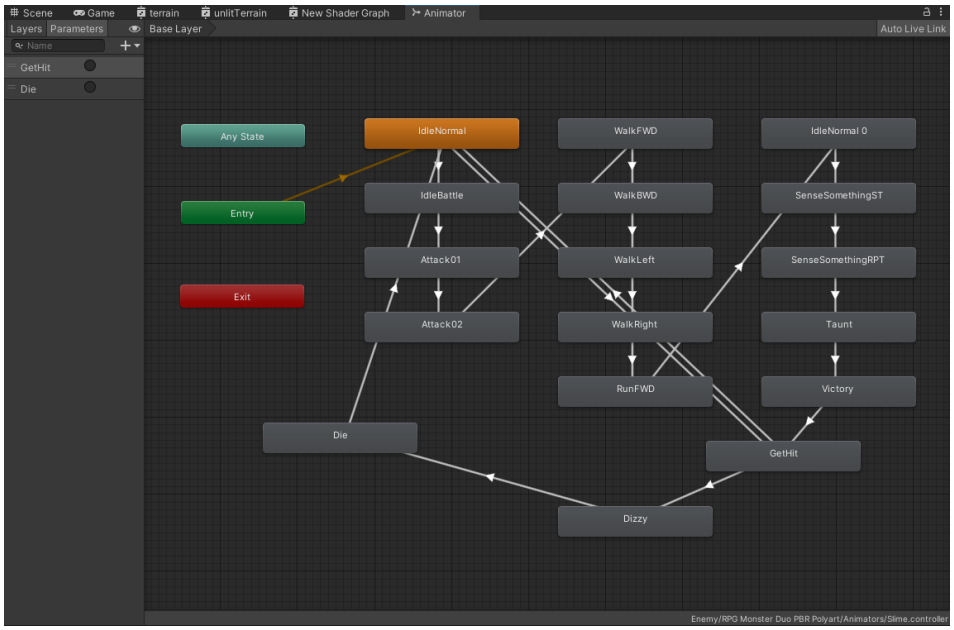
[2] Nadređeni-podređena relacija; Jedan objekt može imati više podređenih objekta.

Slika 5. Primjer hijerarhije objekata u sceni.

U druge korisne komponente koje nalazimo na glavnom ekranu uključujemo alatnu traku, gdje možemo pomicati, mijenjati veličinu i rotaciju objekta, te karticu animacija.

3.3 Animator

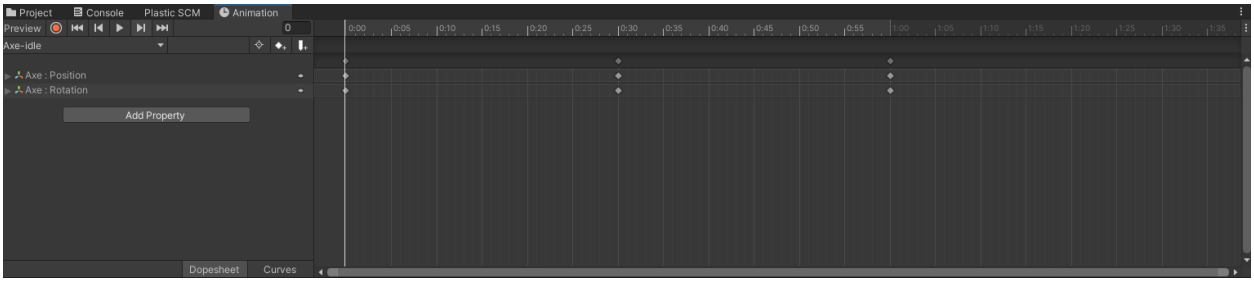
Kako bi igrici dali „života“, pojedine objekte, kao npr. igrača, njegove napade i objekte koje napada trebali bismo animirati, radi lakše vizualne predodžbe što se događa u sceni, a i radi dodavanja dinamičnosti same igre. To možemo lagano napraviti pomoću Animatora, vidljiv na slici 6, alata u Unity-u s kim lagano možemo mapirati pojedine akcije poput napada ili dobivanja štete sa vizualnim odgovorom. Za snimanje objekta možemo uzeti objekt koji želimo animirati, i staviti ga u više različitih pozicija u tzv. timeline-u, a Unity će se pobrinuti da tranzicije pomicanja tog objekta budu što glađe moguće. Zatim tu animaciju možemo implementirati pomoću kartice animatora koji pokazuje kako se animacije povezuju sa drugima, te trajanje i vremenski razmak između animacija. Zatim te animacije možemo pokrenuti pomoću skripte.



Slika 6. Primjer Animatora jednog objekta.

Same animacije možemo pojednostaviti kao pokreti određenih komponenti od kojih su građeni objekti u nekom vremenskom razdoblju. To razdoblje vizualno predočujemo pomoću vremenske crte na kartici animacije na nekom objektu. Pomoću nje možemo snimiti pokrete objekta u vremenu, a Unity će tranziciju napraviti automatski iz jedne točke

u drugu, što nam omogućava potpunu kontrolu nad brzinom i duljinom animacije. Primjer jedne jako jednostavne animacije vidljiv je na slici 7.

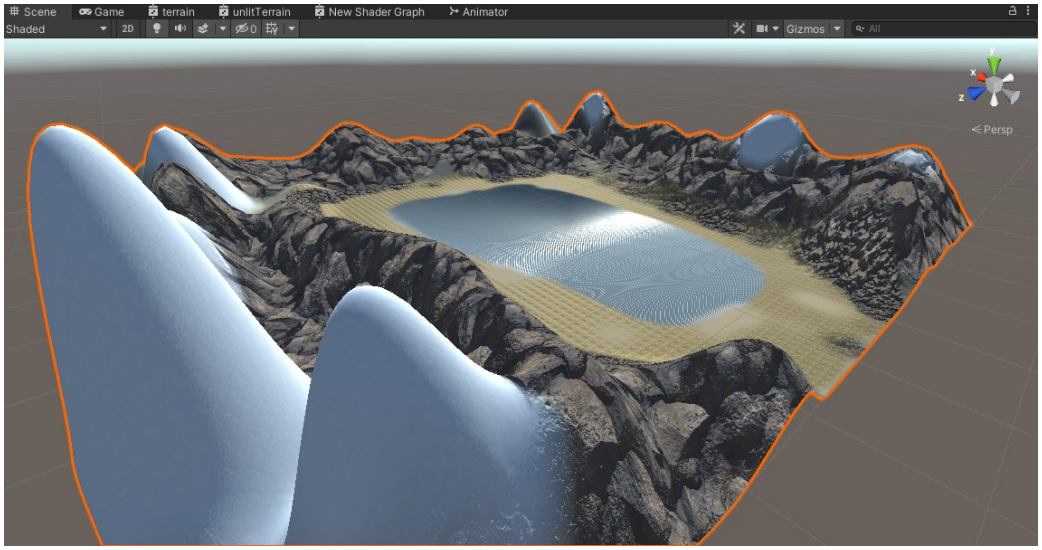


Slika 7. Primjer animacijske vremenske crte

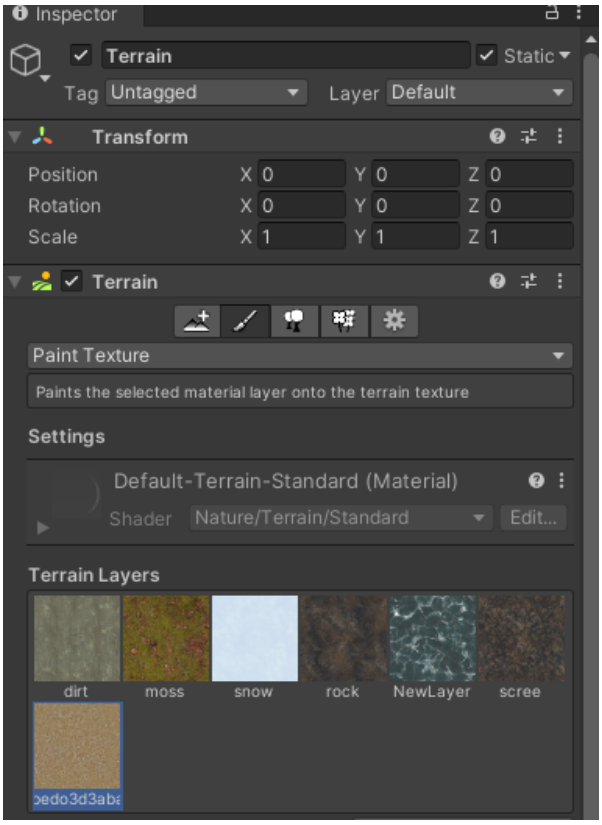
3.4 Oblikovanje terena

Jedan od veoma korisnih alata koji je nedavno ugrađen u Unity jest oblikovanje terena pomoću alata za oblikovanje terena koji uz pomoć kistova, boja i drugih alata daje mogućnost oblikovanja terena po korisnikovoj želji. Primjer takvog objekta vidljiv je na slici 8.

Korisnik ima opciju podizanja i spuštanja tla, bojanje tekstura, dodavanje prefabova kistom, zaglađivanje oštrih dijelova terena, simuliranja djelovanja erozije vode, vjetra i drugih opcija vidljivih na slici 9. Završni rezultat bio bi objekt koji bi mnogo teže bio napravljen tradicionalnim metodama.



Slika 8. Primjer izgrađenog terena



Slika 9. Alat za oblikovanje terena u inspektoru

3.5. Borba

Sistem borbe može se izvesti na bezbroj načina. Sistem borbe ovog projekta osnivat će se na RPG elementima. Svaki korisnik ima level, snagu i životne bodove koji se povećavaju ovisno o levelu igrača, vidljivo na gornjem desnom kutu na slici 10. Igrač povisuje svoj level kada porazi nekog neprijatelja, gdje mu pri svakom levelu raste snaga i životni bodovi. Igrač će moći na početku igre izabrati oružje koje ima svoju vrijednost napada i vrijeme između napada. Konačni napad se izračunava dodavanjem snage na napad oružja te pri svakom kontaktu s neprijateljem njihov život se smanjuje za tu vrijednost.

Slika 10. Primjer izgleda kamere igrača i neprijatelja.

3.6. AI neprijatelja

Neprijatelji imaju u suštini dvije skripte: Enemy AI, koji sadrži njihov život, koliko štete rade igraču pri napadu i koliko bodova daju za sljedeći level prilikom smrti, te posebnu skriptu za svaki tip neprijatelja, koji sadrži njihove napade, ponašanja za vrijeme napadanja itd. Kao primjer navest će se jedan od jačih neprijatelja, koji je u obliku zmaja. On ima skriptu

EnemyAi koja sadrži opće informacije, te RedController, u kojem se nalaze funkcije u kojima navodimo ponašanje kada susretne igrača, napadi koji se odabiru ovisno o veličini određenog nasumičnog broja u skripti i drugim obilježjima poput njegovog preostalog broja života ili da li je u letećoj poziciji. Napad se vrši tako da se odradi animacija željenog napada, a u slučaju da collider[3] komponenta neprijatelja dotakne collider objekta pod tag-om igrača, igrač gubi život ovisno o napadu zmaja. Primjer dijela koda gdje neprijatelj napada prikazan je na slici 11.

[3] U ovom slučaju koristimo box collider, tj. collider koji je u obliku kvadrata.

```
if (!alreadyAttacked && isFlying == false)
{
    chillCount = true;
    alreadyAttacked = true;
    int rand = Random.Range(0, 10);
    Debug.Log("random number:" + rand);
    //Flame attack
    if (rand > 8)
    {
        //
        breath.SetActive(true);
        alreadyAttacked = true;
        //GameObject effect = Instantiate(breath, breathContainer.position, Quaternion.identity);
        //breath.transform.rotation = transform.rotation;
        Debug.Log("Flame attack initiated at:" + rand);
        GetComponent<Animator>().SetTrigger("FlameAttack");



    }
    //claw attack
    else if (rand <= 8 && rand >= 5)
    {
        alreadyAttacked = true;
        Debug.Log("Claw attack initiated at:" + rand);

        GetComponent<Animator>().SetTrigger("ClawAttack");
    }
    //basic attack
    else if (rand <= 4 && rand >1)
    {
        alreadyAttacked = true;
        Debug.Log("attack initiated at:" + rand);

        GetComponent<Animator>().SetTrigger("Attack");


    }
    else if (rand <=1)
    {
        FlyScript();
    }
    chillCount = false;
    alreadyAttacked = true;
    Invoke(nameof(ResetAttack), timeBetweenAttacks);
}

//flying special
else if(!alreadyAttacked && isFlying == true)
    {
    chillCount = true;
    alreadyAttacked = true;

    Debug.Log("Flame attack initiated");
        Animator anim = GetComponent<Animator>();
```

Slika 11. Dio koda za napad zmaja

Svi neprijatelji imaju u skripti funkcije za kretanje u slučaju da se nalaze u određenoj daljini od igrača. Funkcionira na principu da se skripta sastoji od tri dijela; stanje ophodnje, stanje lova na igrača, te stanje napada, koji se pokreće u „Update“ funkciji, vidljivoj na slici 12.

```
0 references
private void Update()
{
    CheckForGround();
    distance = Vector3.Distance(transform.position, player.transform.position);
    playerInSightRange = Physics.CheckSphere(transform.position, sightRange, whatIsPlayer);
    playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, whatIsPlayer);
    //playerInRangedRange = Physics.CheckSphere(transform.position, rangedRange, whatIsPlayer);




    if (!playerInSightRange && !playerInAttackRange) Patroling();
    if (playerInSightRange && !playerInAttackRange) ChasePlayer();
    if (playerInSightRange && playerInAttackRange) AttackPlayer();

}
1 reference
private void Patroling()
{
    if (!walkPointSet) SearchWalkPoint();

    if (walkPointSet)
    {
        chillCount = false;
        Animator anim = GetComponent<Animator>();
        anim.SetTrigger("Walk");
        //agent.SetDestination(walkPoint);

        Vector3 distanceToWalkPoint = transform.position - walkPoint;

        //anim.SetTrigger("Walk");

        if (distanceToWalkPoint.magnitude < 1f)
            walkPointSet = false;
        if(isFlying == false) anim.SetBool("IsWalking", true);
        if (isFlying == true) anim.SetTrigger("FlyForward");

    }
}
1 reference
private void ChasePlayer()
{
    chillCount = false;
    Animator anim = GetComponent<Animator>();
    anim.SetBool("isWalking", true);
    transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(player.position - transform.position), 3);
    if (distance > stop)
    {
        transform.position += transform.forward * moveSpeed * Time.deltaTime;
    }
}
```

Slika 12. Dio koda koji odlučuje kako će se neprijatelj ponašati u određenim daljinama od igrača

Granice daljine se određuju dvjema sferama, jedna za stanje lova, a jedna za granicu napada, te je podesiva za svakog neprijatelja. Stanje ophodnje, gdje neprijatelj stoji na mjestu, stanje lova, gdje neprijatelj prepoznaje igrača te se kreče prema njemu brzinom koju smo mu zadali, te stanje napada, gdje nakon što igrač dođe u određenu daljinu, neprijatelj počinje napadati. Dok lovi igrača, skripta prati visinu neprijatelja i šalje Raycast-

ove, nevidljive linije, koje u slučaju promijene visine podesi visinu objekta kako ne bi prošao kroz zidove ili brežuljke.

3.7. Sistem oružja

Za ovu igru pripremili smo tri oružja; koplje, sjekiru i čarobni štap, svaki od njih ima jedan normalan napad i dva alternativna napada, koji troše resurs „mana“. Oružja je moguće na početnoj sceni pokupiti i baciti na pod pomoću posebne skripte, vidljive na slici 13, koja još naknadno gleda udaljenost igrača i oružja da odluči dali igrač može ili ne može pokupiti to oružje.

```
1 reference
private void Pickup()
{
    equipped = true;

    GetComponent<Animator>().enabled = true;
    Debug.Log("tried picking up");

    slotFull = true;

    transform.parent.SetParent(weaponHolder);
    transform.parent.localPosition = Vector3.zero;
    transform.parent.localRotation = Quaternion.Euler(Vector3.zero);
    //transform.localScale = Vector3.one;
    weaponController.isEnabled = true;

    rb.isKinematic = true;
    coll.isTrigger = true;


}
1 reference
private void Drop()
{
    GetComponent<Animator>().enabled = false;
    equipped = false;
    slotFull = false;


    weaponHolder.DetachChildren();

    transform.position = player.position;
    Debug.Log(transform.position);
    transform.GetComponent<Rigidbody>().isKinematic = false;

    rb.position = transform.position;


    coll.isTrigger = false;

    rb.velocity = player.GetComponent<Rigidbody>().velocity;

    rb.AddForce(cameraPlayer.forward * dropForwardForce, ForceMode.Impulse);
    rb.AddForce(cameraPlayer.up * dropUpwardForce, ForceMode.Impulse);
    float random = Random.Range(-1f, 1f);
    rb.AddTorque(new Vector3(random, random, random) * 10);
    weaponController.isEnabled = false;
}
```

Slika 13. Metode za sistem hvatanja i bacanja oružja

Kao primjer oružja uzet ćemo koplje. Sistem za svako oružje sastoji se od tri glavne skripte: „Weapon controller“, kojim se može odrediti inicijalna snaga oružja, vrijeme između napada, te se vrši provjera dali oružje trenutačno napada, „Collision detection“ koji gleda dali se, u slučaju kada igrač napada, collider oružja preklapa sa colliderom neprijatelja gdje, ako se desi kolizija, skripta daje štetu navedenom neprijatelju. Zadnja skripta se odnosi na specijalne napade oružja. Prvi specijalni napad koplja služi za teleportaciju igrača na lokaciju ispred igrača, vidljivo na slici 14. Budući da teleportacija ne gleda fiziku, igrač bi bio u mogućnosti teleportirati se van mape tako da izbjegne collider terena. Da bi to spriječili funkcija je uvedena, koja provjerava dali bi igrač prošao kroz teren te ga vrača na točku u terenu kroz koju bi inače prošao. Ista provjera se događa kako igrač ne bi mogao proći kroz neprijatelje.

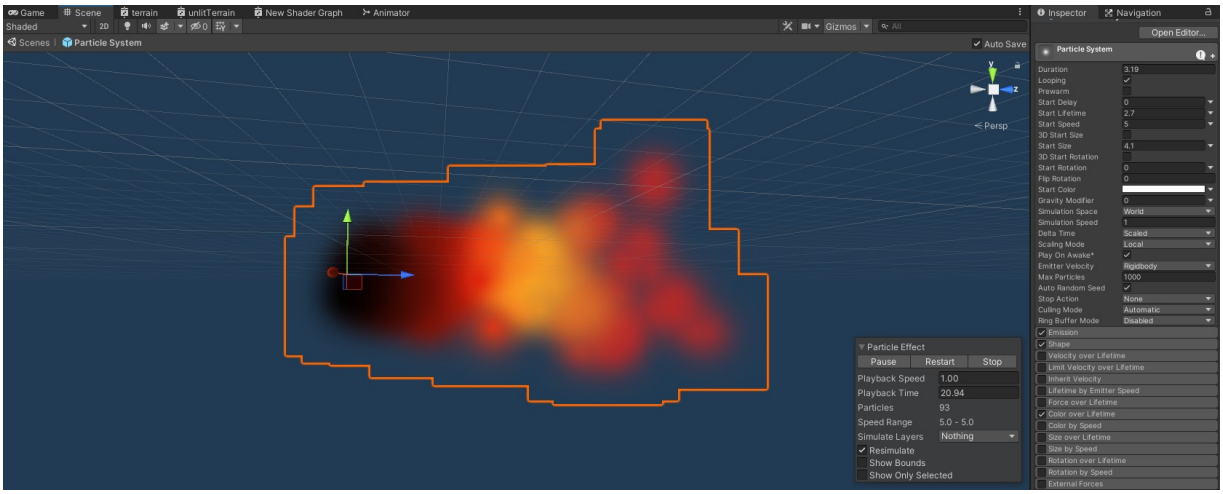
```
private void Dash()
{
    wc.IsAttacking = true;
    float temp = wc.dmg;
    wc.dmg += 0.3f * temp;


    RaycastHit hit;
    Ray dashRay = new Ray(transform.position, player.forward);
    Debug.DrawRay(transform.position, player.forward,Color.red);
    Animator anim = wc.Axe.GetComponent<Animator>();
    anim.SetTrigger("Dash");
    if (Physics.Raycast(dashRay, out hit, 15f))
    {
        if (hit.collider.tag == "Enemy")
        {
            Debug.Log("hitsomething");
            player.position = hit.point;
            player.position -= player.forward;

        }

        else if (hit.collider.tag == "Enviroment")
        {

            player.position = hit.point;
        }
        else player.position += player.forward * pushforce;
    } else player.position += player.forward * pushforce;

   // rb = player.GetComponent<Rigidbody>();

    GameObject effect = Instantiate(dashEffect, playercam.position, dashEffect.transform.rotation);
    effect.transform.parent = playercam.transform;
    effect.transform.LookAt(transform);
    //transform.parent.parent.parent.GetComponent<Rigidbody>().AddRelativeForce(Vector3.forward * pu
   // slash.SetActive(false);
    Debug.Log("added force to", rb);
    wc.dmg -= 0.3f * temp;
```

Slika 14. Kod za teleportaciju i provjeru kolizije

Drugi napad je jača verzija glavnog napada sa drugom animacijom, efektima i većim iznosom štete. Važno je napomenuti kako svaki napad, da li od strane igrača ili neprijatelja je popraćen posebnom animacijom i nekada efektima za koje koristimo Unity-ov sustav čestica tj. particle system.

3.8 Sustav čestica

Jedna od glavnih značajki Unity-a koji ga posebice čini primamljivim za početnike jest njegov sustav čestica, koji omogućava lagano i intuitivno stvaranje vlastitih vizualnih efekata. Na slici 15 prikazana je čestica koja se upotrebljava kao vatreni napad jednog od neprijatelja.



Slika 15. Primjer kreirane čestice

Čestice se, kao i neprijatelji, objekti i dr. mogu lagano instancirati pomoću skripta. U ovom primjeru se čestica instancira u istom odjeljku koda kao i za pokretanje animacije napada neprijatelja.

3.9 Stvaranje neprijatelja

Budući da nam je teren na kojem igramo beskonačan, moramo uvesti sistem koji će dinamički stvarati neprijatelje u svijet ovisno o našoj poziciji te ih maknuti nakon što smo prešli određenu udaljenost od njih. Za ovo koristimo posebni objekt koji je povezan sa

samim igračem koji u sebi ima skriptu za praćenje, stvaranje i brisanje neprijatelja. Za samo instanciranje neprijatelja koristimo listu zadanih objekata za neprijatelje i šefove, tj. veće neprijatelje. Svaki neprijatelj ima određeni postotak šanse da se instancira pomoću nasumičnih brojeva i drugih uvjeta, poput pređene udaljenosti ili ukupnom broju već pobijeđenih neprijatelja. Svaki stvoreni objekt ide u posebnu listu gdje se prati njihov broj, te se stvaranje vrši dok se ne stvore maksimalni broj neprijatelja. Kao što je vidljivo na slici 16, pri svakom ažuriranju funkcija prati udaljenost stvorenih objekata od igrača te briše neprijatelje koji su prešli maksimalnu granicu udaljenosti, te u sljedećom ažuriranju uklanja sve prazne objekte iz liste da se mogu stvoriti novi.

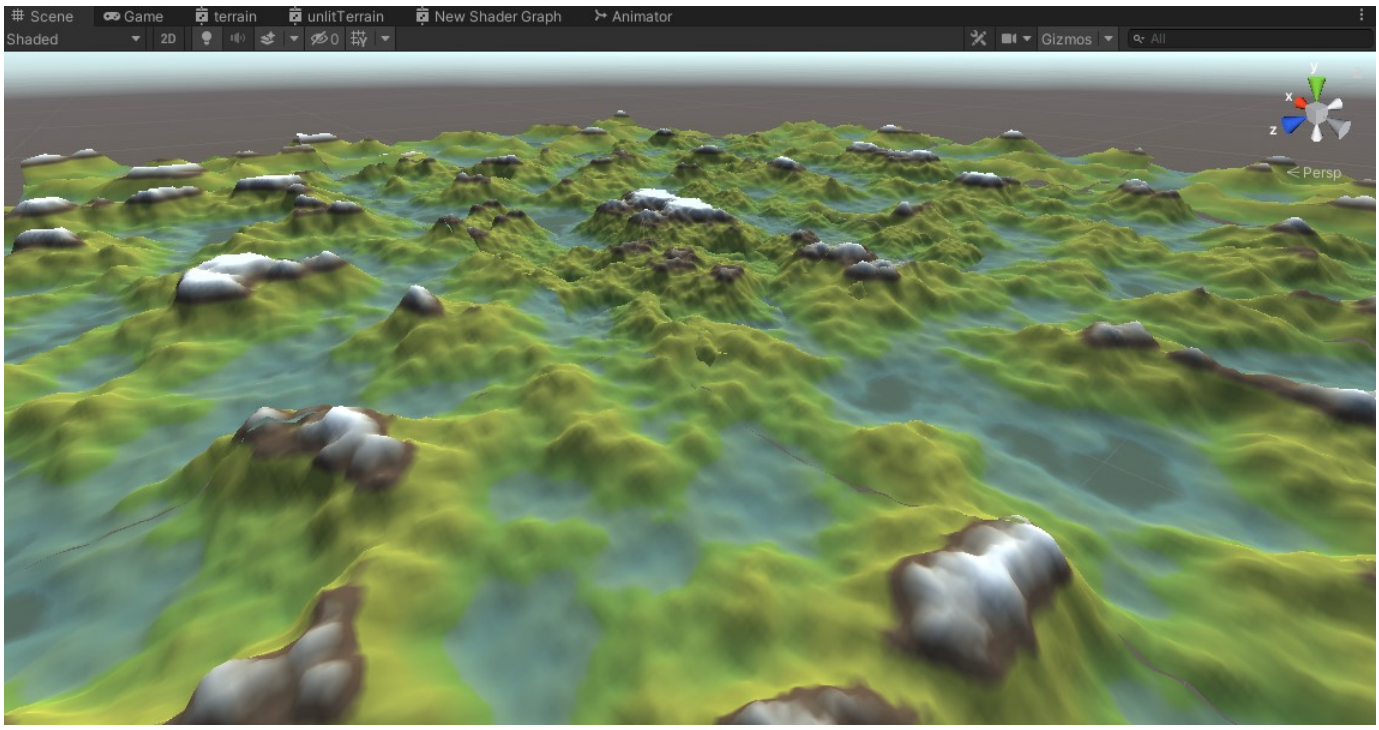
```
while (enemyCount <= 50)
{

        getRidOfEmptyList();
        if (Random.value > 0.5)
        {
        yPos = player.position.y + 2;
        randomX = Random.Range(-50, 50);
            randomZ = Random.Range(10, 50);
            GameObject enemy = Instantiate(enemies[0].enemy_prefab, new Vector3(player.position.x + randomX, yPos, player.position.z + randomZ), Quaternion.identity);
            instantiatedEnemies.Add(enemy);
            CheckForGround(enemy);
            enemyCount += 1;
            killCount += 1;
        }
        if (Random.value > 0.6)
        {
        yPos = player.position.y + 2;
        randomX = Random.Range(-50, 50);
            randomZ = Random.Range(10, 50);
            GameObject enemy = Instantiate(enemies[1].enemy_prefab, new Vector3(player.position.x + randomX, yPos, player.position.z + randomZ), Quaternion.identity);
            instantiatedEnemies.Add(enemy);
            CheckForGround(enemy);
            enemyCount += 1;
            killCount += 1;
        }
        if (Random.value > 0.9)
        {
        yPos = player.position.y + 2;
        randomX = Random.Range(-50, 50);
            randomZ = Random.Range(1, 50);
            GameObject enemy = Instantiate(enemies[2].enemy_prefab, new Vector3(player.position.x + randomX, yPos, player.position.z + randomZ), Quaternion.identity);
            instantiatedEnemies.Add(enemy);
            CheckForGround(enemy);
            enemyCount += 1;
            killCount += 1;
        }
        if (distanceTraveled > 2000 && Random.value > 0.9 && killCount > 100)
        {

            randomX = Random.Range(-50, 50);
            randomZ = Random.Range(1, 50);
            GameObject enemy = Instantiate(bosses[0].boss_prefab, new Vector3(player.position.x + randomX, yPos, player.position.z + randomZ), Quaternion.identity);
            instantiatedEnemies.Add(enemy);
            CheckForGround(enemy);
            enemyCount += 1;
            killCount += 1;
        }
        if (distanceTraveled > 1000 && Random.value > 0.8 && killCount > 20)
        {

            randomX = Random.Range(-50, 50);
            randomZ = Random.Range(1, 50);
            GameObject enemy = Instantiate(bosses[0].boss_prefab, new Vector3(player.position.x + randomX, yPos, player.position.z + randomZ), Quaternion.identity);
            instantiatedEnemies.Add(enemy);
            CheckForGround(enemy);
            enemyCount += 1;
            killCount += 1;
        }



        //despawn
        if (enemyCount >= 1)
        {

            for (int i = instantiatedEnemies.Count - 1; i >= 0; i--)
            {
                //foreach (GameObject enemy in instantiatedEnemies)

                if (instantiatedEnemies[i] != null)
                {
                    float distance = Vector3.Distance(instantiatedEnemies[i].transform.position, player.position);
                if ((int)Vector3.Distance(instantiatedEnemies[i].transform.position, player.position) > 100)
                {
                    Debug.Log("distance:" + Vector3.Distance(instantiatedEnemies[i].transform.position, player.position));
                    //element.enemy_prefab.SetActive(false);
                    Debug.Log("tried destroying " + instantiatedEnemies[i]);
                    //instantiatedEnemies.RemoveAt(i);

                    Destroy(instantiatedEnemies[i], 0.1f);

                    enemyCount -= 1;
                }
            }
```

Slika 16. Stvaranje i brisanje objekata

4. Proceduralno generiranje terena u Unity-u

Ideja za ovaj projekt je bila da stvorimo beskonačni svijet u kojem se igrač može kretati. Za to koristimo proceduralno generiranje podataka, gdje iz Perlinove funkcije šuma izvlačimo podatke o visini terena od 0 prema 1, koje onda pretvaramo u mesh, liniju

trokuta u kojoj je vrijednost Perlinovog šuma prevedena u visinu terena. Zatim tom meshu dodajemo teksture ovisno o visini terena, npr. voda će imati najmanju visinu, dok će snijeg biti samo na najvišim visinama. Primjer generiranog terena prikazan je na slici 17. Kao referencu koda koristili smo video seriju Sebastian Lague-a [3] koji prolazi kroz većinu glavnih aspekata proceduralnog generiranja terena i njenog prilagođavanja po potrebi korisnika.

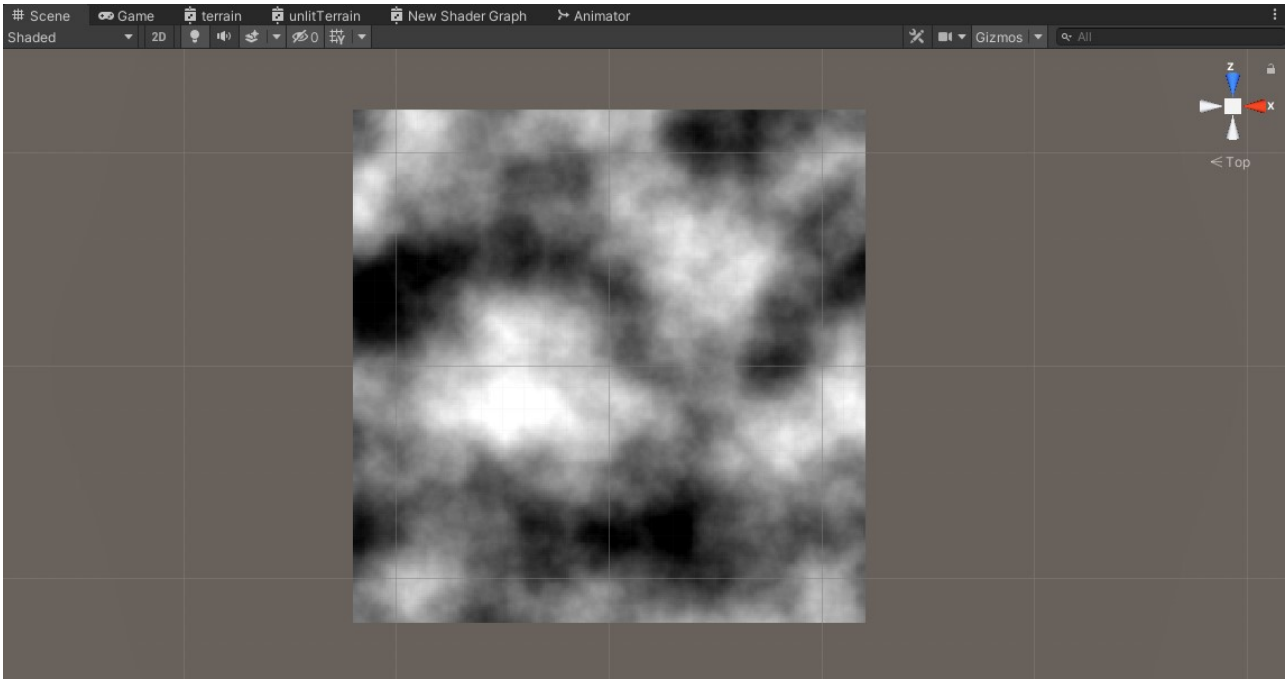


Slika 17. Primjer konačnog proceduralno generiranog terena

4.1 Perlinova funkcija šuma

Ken Perlin je 1983. godine izmislio ovaj algoritam zbog frustracije na tadašnji izgled CGI, tj. kompjuterski generiranih slika [8]. To je proceduralna višedimenzionalna struktura u kojoj se definira mreža nasumičnih vektora koja prolaskom kroz transformacije završava kao mreža pseudo nasumičnih vrijednosti gdje svjetlije nijanse predstavljaju više vrijednosti funkcije, dok tamnije nijanse predstavljaju niže vrijednosti.

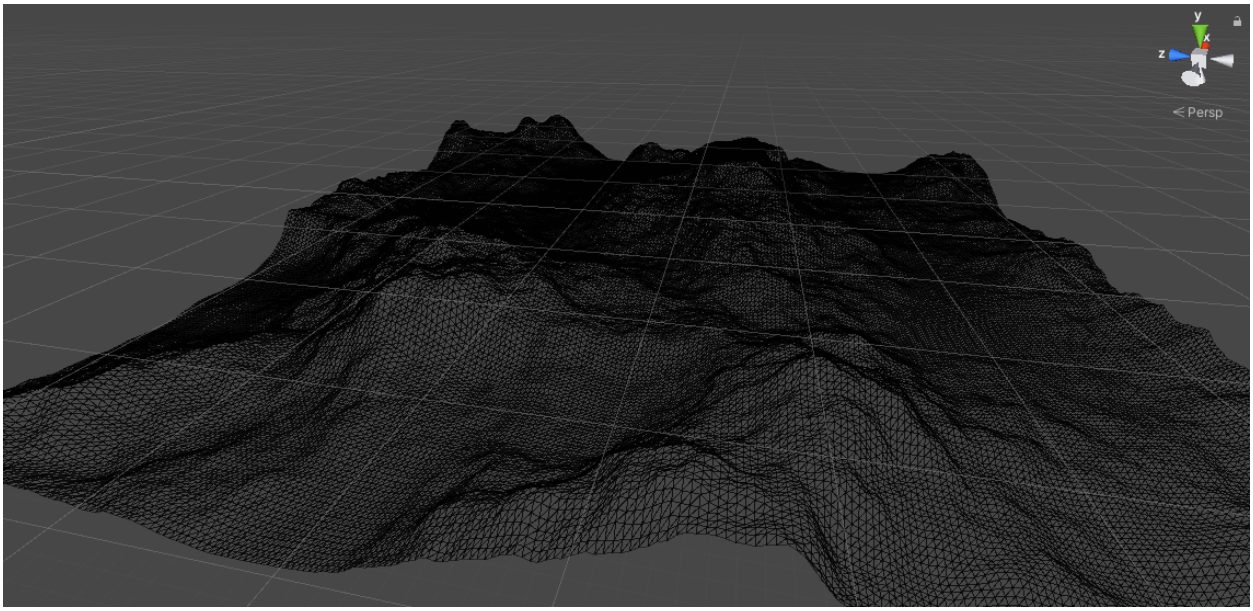
Unity u sebi ima ugrađenu funkciju Matf.PerlinNoise(float x, float y), koji uzima x i y koordinate i vraća vrijednost između 0 i 1 za svaki x i y. Na slici 18 prikazuje se primjer generiranog šuma primijenjen kao materijal na kvadratnom objektu.



Slika 18. Primjer generiranog objekta koristeći Perlinovu funkciju šuma

4.2 Mesh

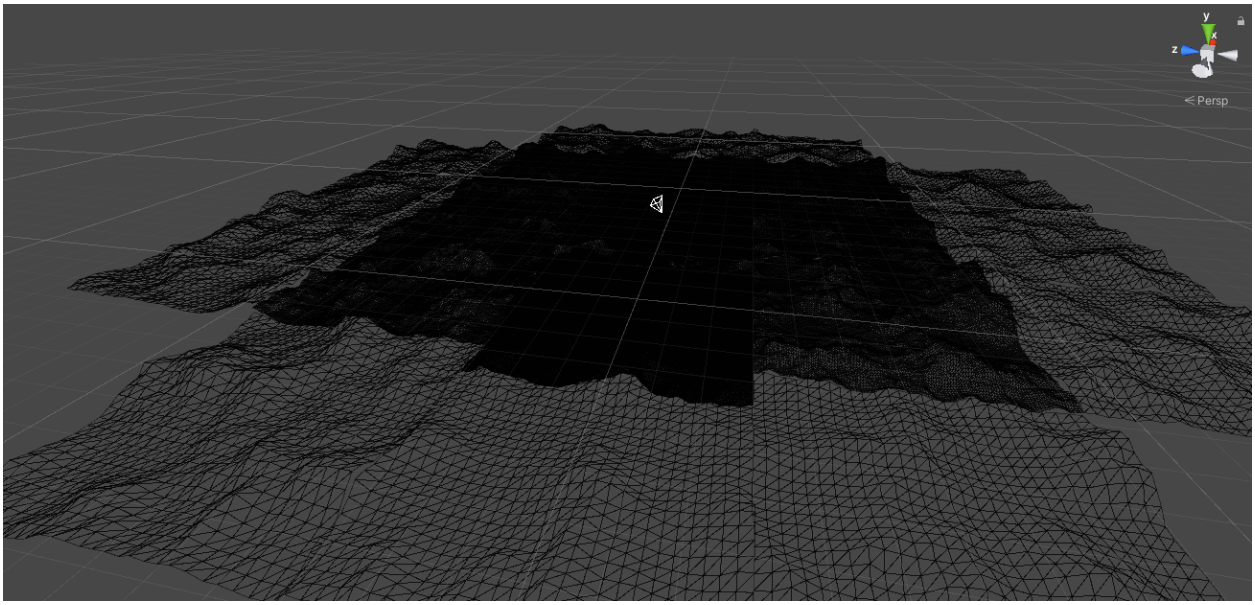
Mesh je skup točaka (eng. vertices) i linija (eng. edges) koji kreiraju poligone u kojem je osnovni oblik trokut. Kreiramo mesh tako da prvo definiramo točke i linije te stvaramo osnovne trokute koje ćemo kasnije pomoću Unity-eve podrške pretvoriti u gotovi mesh koji možemo dizati ovisno o varijabli multiplikacije visine pomoću krivine koja drugačije djeluje na visinu ovisno o vrijednosti perlinove vrijednosti šuma (npr. niske vrijednosti se neće podizati s povećanjem varijable, dok će se visoke vrijednosti jače povećavati). Završni rezultat, vidljiv na slici 19, bila bi završna mreža trokuta koje čine teren.



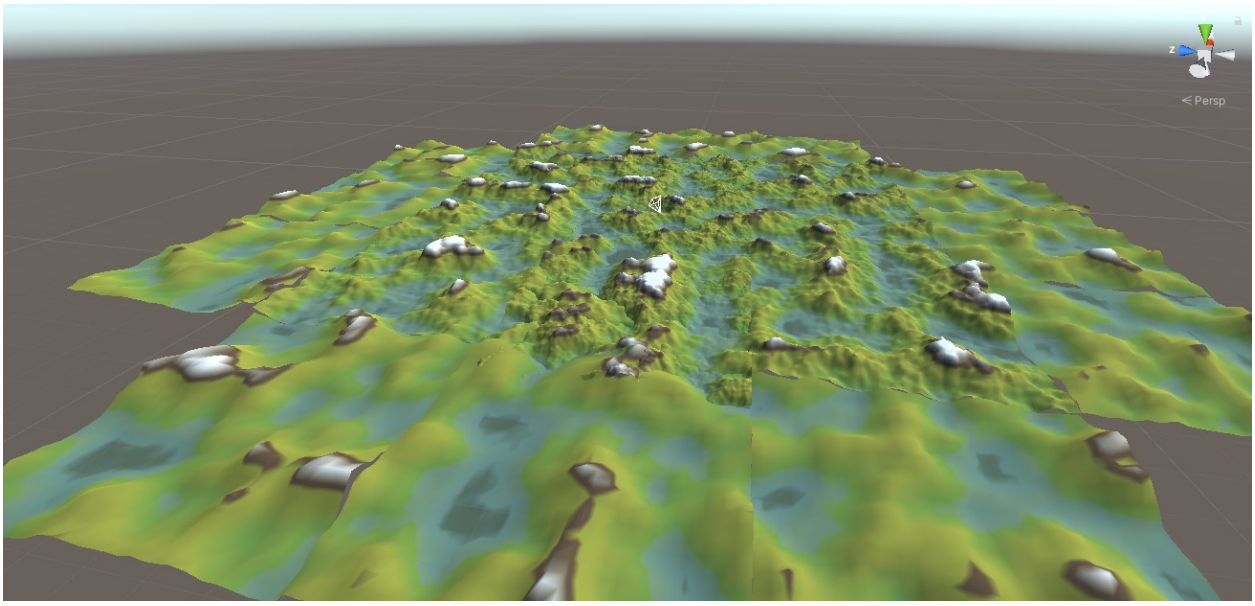
Slika 19. Primjer kreiranog mesh-a

4.3 Beskonačna generacija terena

Unity podržava mesheve do veličine 65535 točaka u jednom meshu[7]. Za beskonačno generiranje svijeta naravno ne generiramo sve u jedno vrijeme jer bi to negativno utjecalo na performanse do razine neigrivosti. Zato u ovom projektu podijelili smo teren na chunkove, tj. više terena koji imaju svoj mesh, vrijednost visine, širine i dužine, te ih povezali s pozicijom igrača, da se samo dinamički generiraju chunkovi do neke udaljenosti od igrača. Te chunkove smo spremili u memoriju i brišu se s ekrana kada se dovoljno udaljimo od njega. Implementirana je također varijabla detalja koja se može koristiti da bi smanjili, tj. povećali broj trokuta generiranih u jednom meshu. U slici 20 igrač se nalazi u sredini chunkova gdje, ovisno o udaljenosti od igrača, broj trokuta u meshu se smanjuje. Tu funkcionalnost koristimo kako bi smanjili level detalja u udaljenim chunkovima radi boljih performansa. Slika 21 prikazuje razliku u kvaliteti ovisno o udaljenosti kada se dodaju teksture. Važno je napomenuti da, iako je level kvalitete mesha chunkova na većoj udaljenosti od igrača slabiji, igrač ih neće vidjeti u potpunosti, a kada dođe dovoljno blizu, kvaliteta će se povećati.



Slika 20. Primjer proceduralno generiranog terena. Tamniji chunkovi predstavljaju veću razinu detalja, svjetliji predstavljaju nižu razinu detalja.

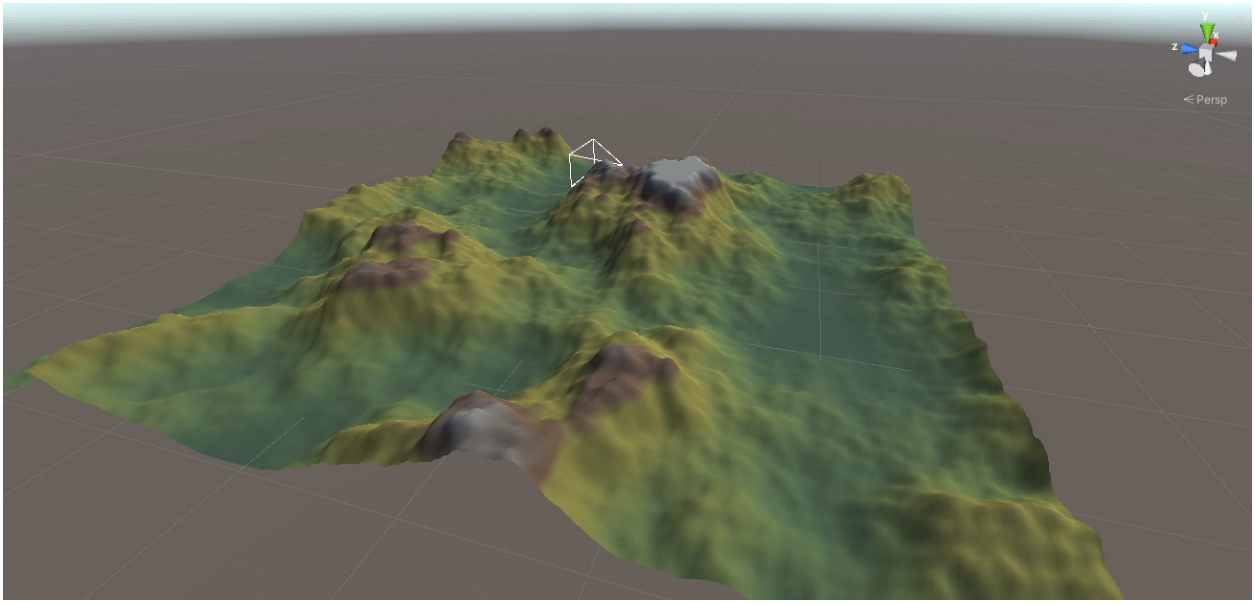


Slika 21. Prikaz razine detalja ovisno o poziciji igrača sa dodanin teksturama.

4.4 Dodavanje tekstura

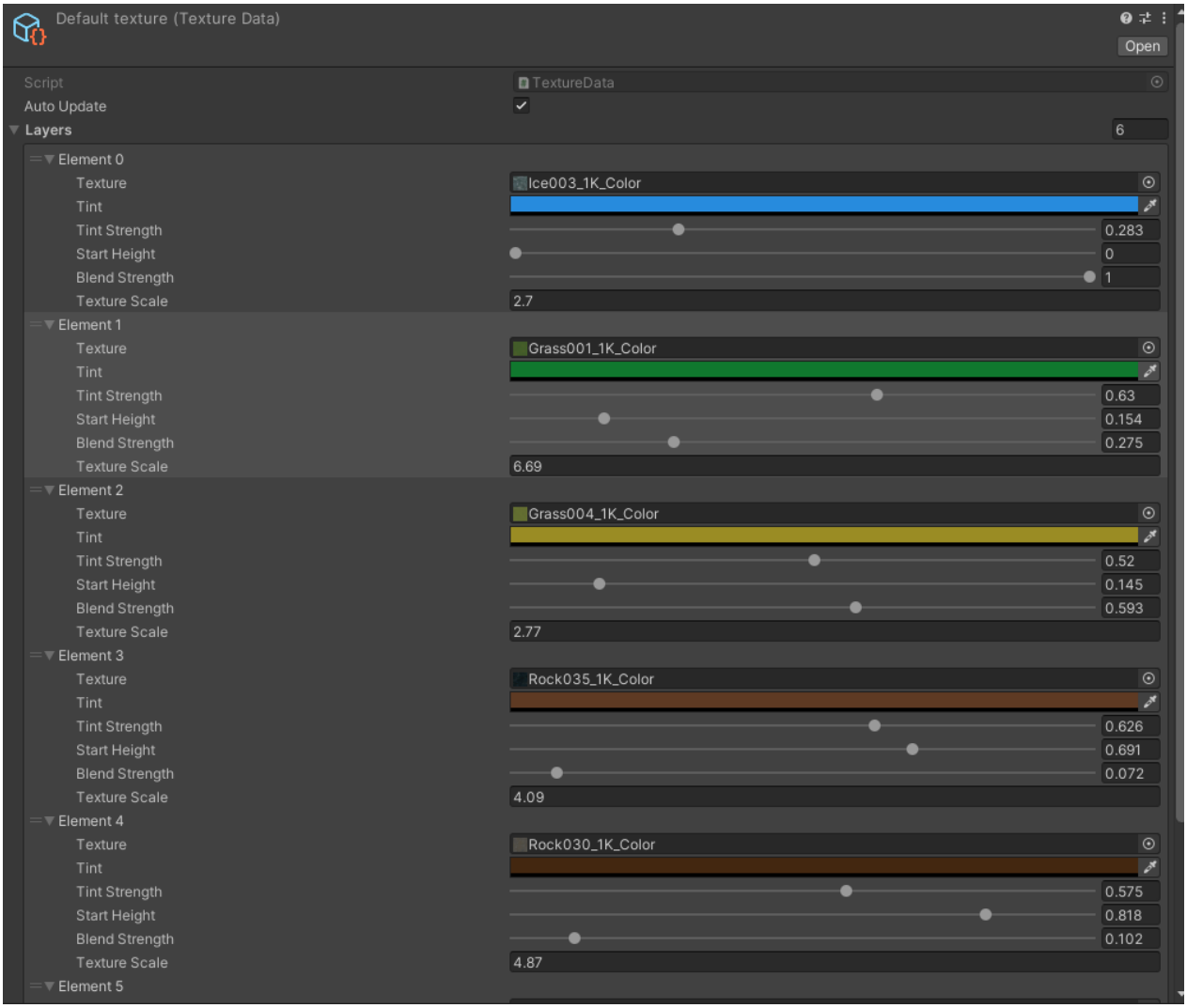
Prije dodavanja tekstura izradili smo novu shader skriptu koju ćemo koristiti kao materijal za naš mesh. Shader skripte sastoje se od algoritama i pravila o tome kako će se računati boja ili tekstura za svaki piksel u nekom objektu. Za ovaj shader cilj nam je bio izgladiti tranziciju između boja na različitim visinama tako da uzima granične vrijednosti između dvije teksture i miješa ih. Zatim taj shader dodamo novom materijalu koji će biti materijal za naš mesh, na kojem određujemo na kojoj visini će se pojavljivati koja tekstura, kao što je vidljivo na slici 22.

Sve teksture bile su preuzete kao free assets na Unity Asset Store-u.



Slika 22. Mesh sa dodanim teksturama.

Koje teksture želimo dodati na kojim visinama možemo odrediti dodajući ih u listu objekata koje sadrže teksturu kojim možemo mijenjati vrijednosti kao na slici 23, kao na primjer na kojoj visini će se pojavljivati tekstura, te također postoje druge opcije za tranziciju između različitih tekstura, da izgledaju fluidnije zajedno.

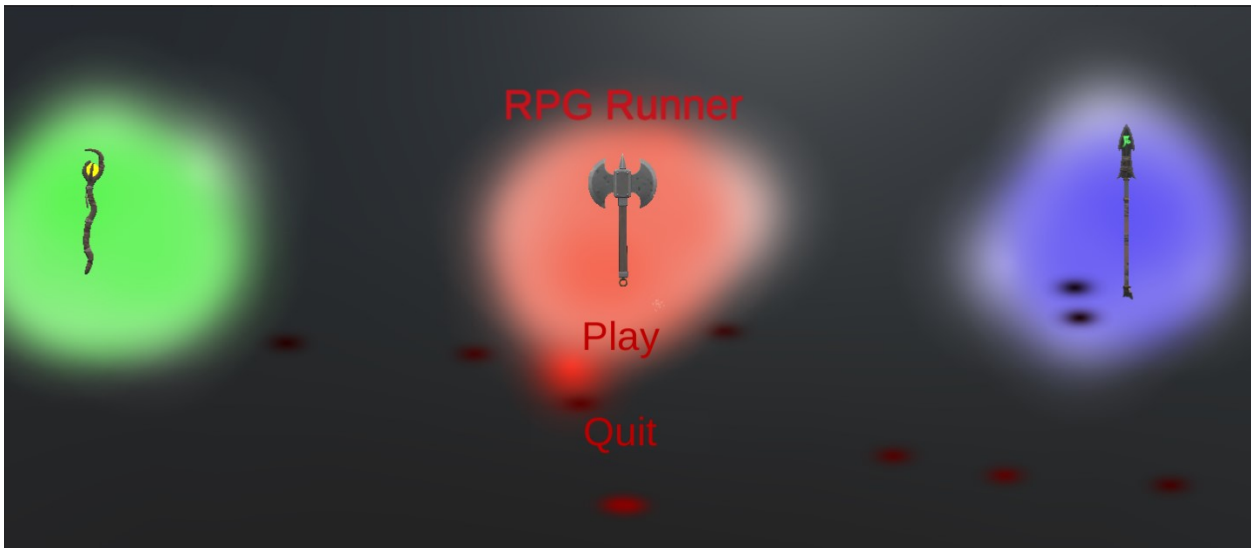


Slika 23. dodavanje tekstura ovisno o visini mesh-a na svakoj točci.

5. Dodavanje glavnog izbornika i UI komponente

5.1 Glavni izbornik

Glavni izbornik sastavljen je od pozadine, gumbova i tekst okvira, vidljivo na slici 24. Klikom na jedan od gumbova pokreće se funkcija iz određene skripte. Gumbom „Play“ mijenjamo scenu na scenu odabira oružja, dok s gumbom „Quit“ izlazimo iz programa.



Slika 24. Prikaz glavnog izbornika

5.2 Korisničko sučelje

Pri dizajnu korisničkog sučelja cilj je bio minimizirati broj objekta na ekranu. Igrač uvijek ima vidljivo na kameri svoj preostali život i manu te trenutačni nivo (eng. level) na kojem je, kao i stopa popunjenosti levela kojim vizualiziramo koliko je bodova ostalo do sljedećeg levela, vidljivo na slici 25. Pri sakupljanju dovoljno bodova igraču se poveća level, njegova „mana“ i život vrate se na 100%, te se život poveća za 15%, a mana za 10% [13].

```
public void UpdateXpUI()
{
    float xpFraction = currentXp / xpToLvlUp;
    float FXP = frontXpBar.fillAmount;
    if(FXP<xpToLvlUp)
    {
        delayTimer += Time.deltaTime;
        backXpBar.fillAmount = xpFraction;
        if(delayTimer<3)
        {
            lerp += Time.deltaTime;
            float percentComplete = lerp / 4;
            frontXpBar.fillAmount = Mathf.Lerp(FXP, backXpBar.fillAmount, percentComplete);
        }
    }
}
6 references
public void GainExperienceFlatRate(float xpGained)
{
    currentXp += xpGained;
    lerp = 0f;
    delayTimer = 0f;
}
1 reference
public void LevelUp()
{
    level++;
    playerLevelText.text = "LVL " +level.ToString();
    frontXpBar.fillAmount = 0f;
    backXpBar.fillAmount = 0f;
    currentXp = Mathf.RoundToInt(currentXp - xpToLvlUp);
    player.health = 100 + level * 15;
    player.mana = 100 + level * 10;
    xpToLvlUp = CalcReqXp();
}
```

string int.ToString() (+ 3 overloads)

Slika 25. dio skripte koja prati level igrača

Također je uvedeno praćenje prijeđene daljine od početne točke koje se pojavljuje na vrhu ekrana, koja nakon svakih 100 metara poveća i smanji veličinu fonta, dajući efekt dinamike.

5.3 Kraj igre

Nakon što igrač primi štetu, vrši se provjera u Player Manager skripti da li je preostali život veći od nule. Ako je život manji ili jednak nuli, skripta isključuje objekt igrača i prikazuje na ekranu završne poene, koji se izračunavaju ovisno o pređenoj udaljenosti, broju ubijenih neprijatelja, te levelu na kojem je igrač bio prije nego li je izgubio sav život. Igrač također

ima izbor da li želi probati opet, gdje se vraća na scenu gdje bira oružje, ili izaći iz igrice, vidljivo na slici 26.

Slika 26. GameOver ekran

5.4 Glazba

Za pozadinsku pjesmu odabrana je pjesma „Journey“ samostalne izvedbe i produkcije. Glazba je dodana pomoću jednostavne skripte, vidljive na slici 27, koja pokreće audio source komponentu u objektu te nakon promjene scena Unity zadržava objekt koji nastavlja svirati audio datoteku.

```
public static MusicControlScript instance;
0 references
private void Awake()
{
    DontDestroyOnLoad(gameObject);
    if (instance == null)
    {
        instance = this;
    }
    else
    {
        Destroy(gameObject);
    }
}
```

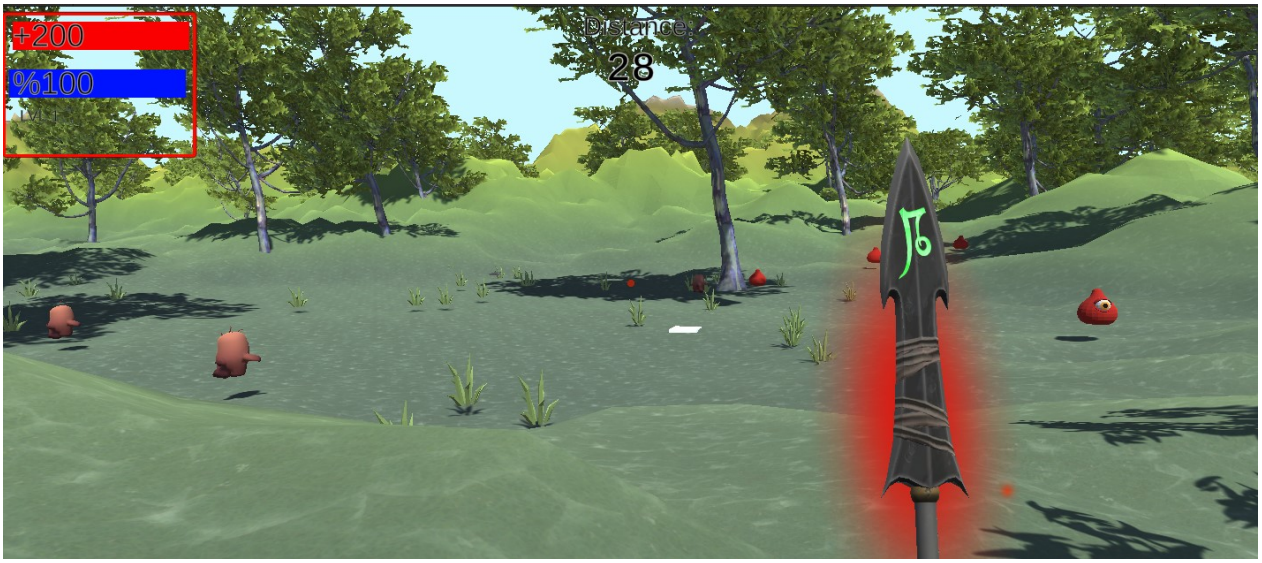
Slika 27. Skripta za zadržavanje glazbe prilikom promjene scene

6. Tijek igre

Budući da je igra bazirana na sistemu ostvarivanja što više bodova u određenom životu, igrač je potaknut da razmišlja o mogućim kombinacijama strategija prolaska kroz igru, te s kojim oružjem će to pokušati ostvariti. Nakon svake smrti, ako igrač odluči izabrati opciju „Try again”, vraća se na scenu izbora oružja, koja je prikazana na slici 28, gdje može isprobati nova oružja na jednom probnom neprijatelju bez straha da umre, te napokon izabrati oružje pritiskom tipke „F” na tipkovnici.

Slika 28. Mjesto gdje igrač može odabrati oružje koje će koristiti.

Igrač je prebačen na novu scenu gdje je odmah okružen neprijateljima te ima opciju kretanja bilo gdje na beskonačnoj mapi. Igri je također dodano generiranje stabala i drugih objekata, poput kamenja i trave. Skripta za generaciju funkcionira slično kao skripta za generiranje neprijatelja, osim što se stabala generiraju ovisno o daljini koju igrač pređe. Na primjer, svakih 150 metara stvorit će se mješavina stabala i kamenja u određenom rasponu od igrača. Početna lokacija na kojoj se igrač uvijek stvori prikazana je na slici 29.



Slika 29. Mjesto početne lokacije na kojoj igrač započinje igru.

Običan napad oružja dovoljan je da pobjedi najslabije neprijatelje, dok kod težih neprijatelja ili veće grupe lakših neprijatelja preporuća se korištenje jednog od dva posebna napada, koji daju veću štetu te imaju veći domet u širini i dužini, ali zato i troše resurs mane, kao što je prikazano na slici 30.

Slika 30. Posebni napad oružja čarobnog štapa koji koristi resurs mana.

7. Zaključak

Proceduralno generiranje svijeta u RPG Runneru proces je izrade mesha na bazi Perlinove funkcije šuma, koje nadalje generiramo beskonačno u obliku chunkova koji se stvaraju i deaktiviraju ovisno o poziciji igrača u svijetu.

Izrada igrice koja se preseže na tri dimenzije mnogo je kompliciranija od igre sa samo dvije dimenzije, prvenstveno radi samog popunjavanja tog 3D prostora naspram 2D igre. Mora se uzet u obzir također da 3D igrice zahtijevaju znatno veću procesorsku snagu računala. Samo donošenje proceduralnog generiranja terena, neprijatelja te struktura negativno utječu na optimizaciju igrice, budući da se stalno događaju kalkulacije udaljenosti između igrača i drugih objekata, uz druge skripte.

U glavne prednosti igre brojimo dinamičnu mehaniku borbe gdje igrač ima mogućnost slobodnog kretanja i teleportacije pomoću specifičnih oružja. Sistem oružja koji je inspiriran RPG igrama je fleksibilan i relativno lagano se mogu dodati dodatna oružja i napadi za nova ili već postojeća oružja. Također se lagano mogu podesiti varijable štete koju igrač zadaje i dobiva, njegov maksimalan život, brzina kretanja i sl. Iste varijable možemo podesiti na novim i već postojećim neprijateljima. Sama generacija terena i oblik terena se također može podesiti podešavanjem funkcije šuma i tekstura, na kojim visinama se teksture mijenjaju u svijetu, i dr. Pri generaciji neprijatelja pomaže poseban sistem na kojem se može podesiti za svakog neprijatelja kolike su šanse da se stvori, koliko će se daleko od igrača stvoriti, te koji se uvjeti moraju ispuniti da bi se stvorio. Međutim u igri se također primjećuju pojedini nedostatci, poput praznine terena na kojem igrač preživljava ili u problemima u skriptama, gdje neprijatelji znaju napraviti nepredviđene poteze ili nasumično ne odraditi pojedine funkcije.

Razvoj igara u Unity razvojnom okruženju složen je proces koji zahtijeva znanje mnogo različitih aspekta igre, mnoge na koje se sami igrači ne obaziru, od dizajna terena na kojem igrač stoji, ponašanja neprijatelja u određenim situacijama do stalnih provjera i promjena stanja igre koje se vrše u svakoj sekundi.

8. Literatura

[1] Unity Technologies, (2021). User manual. https://docs.unity3d.com/Manual/index.html datum pristupa 28.8.2022.

[2] Unity Technologies, (2021). Perlin noise. https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html datum pristupa 20.8.2022.

[3] S. Lague. (2019). GItHub – SebLague/Procedural-Landmass-Generation: Procedural Landmass Generation in Unity. GitHub, Procedural Terrain Generation, https://github.com/SebLague/Procedural-Landmass-Generation datum pristupa 10.6.2022.

[4] Harrigan, Pat, (2007). *Second Person: Roleplaying and Story in Playable Media*. MIT University Press. ISBN 9780262514187 datum pristupa 29.8.2022.

[5] Unity Technologies, (2021). Terrain scupting. https://docs.unity3d.com/Manual/terrain-Tools.html datum pristupa 29.8.2022.

[6] Unity Technologies, (2021), Animation https://docs.unity3d.com/Manual/AnimationSection.html datum pristupa 14.7.2022.

[7] Unity Technologies (2021.), Mesh, https://docs.unity3d.com/ScriptReference/Mesh.html datum pristupa 10.7.2022.

[8] Perlin,K. (2007.), Noisemachine, , https://web.archive.org/web/20071008162217/http://www.noisemachine.com/talk1/4.html datum pristupa 14.7.2022.

[9] Ankit, D. (4.4.2022.), Why choose Unity 3D for Your Next Game Development Project?, https://www.mindinventory.com/blog/unity-3d-game-development datum pristupa 29.8.2022.

Popis slika

Slika 1. Prikaz sučelja igre “The Elder Scrolls V: Skyrim” 3
Slika 2. Prikaz prazne scene u Unity-u. 5
Slika 3. Primjer scene sa mapom projekta 6
Slika 4. Primjer inspektora za objekt Igrača. 7
Slika 5. Primjer hijerarhije objekata u sceni. 8
Slika 6. Primjer Animatora jednog objekta. 9
Slika 7. Primjer animacijske vremenske crte 10
Slika 8. Primjer izgrađenog terena 11
Slika 9. Alat za oblikovanje terena u inspektoru 11
Slika 10. Primjer izgleda kamere igrača i neprijatelja. 12
Slika 11. Dio koda za napad zmaja 14
Slika 12. Dio koda koji odlučuje kako će se neprijatelj ponašati u određenim daljinama od igrača 15
Slika 13. Metode za sistem hvatanja i bacanja oružja 16
Slika 14. Kod za teleportaciju i provjeru kolizije 17
Slika 15. Primjer kreirane čestice 18
Slika 16. Stvaranje i brisanje objekata 20
Slika 17. Primjer konačnog proceduralno generiranog terena 21
Slika 18. Primjer generiranog objekta koristeći Perlinovu funkciju šuma 22
Slika 19. Primjer kreiranog mesh-a 23
Slika 20. Primjer proceduralno generiranog terena. Tamniji chunkovi predstavljaju veću razinu detalja, svjetliji predstavljaju nižu razinu detalja. 24

Slika 21. Prikaz razine detalja ovisno o poziciji igrača sa dodanin teksturama. 24
Slika 22. Mesh sa dodanim teksturama. 25
Slika 23. dodavanje tekstura ovisno o visini mesh-a na svakoj točci. 26
Slika 24. Prikaz glavnog izbornika 27
Slika 25. dio skripte koja prati level igrača 28
Slika 26. GameOver ekran 29
Slika 27. Skripta za zadržavanje glazbe prilikom promjene scene 29
Slika 28. Mjesto gdje igrač može odabrati oružje koje će koristiti. 30
Slika 29. Mjesto početne lokacije na kojoj igrač započinje igru. 31
Slika 30. Posebni napad oružja čarobnog štapa koji koristi resurs mana. 31

Sažetak

Cilj završnog rada bio je upoznati se sa svim alatima koji se koriste u stvaranju igre u Unity razvojnom okruženju. RPG Runner trodimenzionalna je igra sa RPG elementima, poput sistema magije i sistem odabira oružja sa posebnim efektima, dok implementira „Runner“ mehaniku u proceduralnom generiranom svijetu, gdje je cilj preći što veću udaljenost i pobijediti što više neprijatelja prije nego igrač izgubi život. U glavne mehanike igre ubrajamo beskonačno generirani teren, način na koji se proceduralno generiraju neprijatelji, kojima se povećava napad i brzina što dalje je igrač od početne točke. U glavnu prednost igre također brojimo dinamičnu mehaniku borbe gdje igrač ima mogućnost slobodnog kretanja i teleportacije pomoću specifičnih oružja, sistem levela koji iscjeljuje igrača te ga tjera da bira koje neprijatelje će napasti ako planira preživjeti.

Ključne riječi: Unity, RPG, Proceduralno generiranje terena, 3D.

Abstract

The goal of this Bachelor's thesis was to get acquainted with all the tools used in creating a game in the Unity development environment. RPG Runner is a three-dimensional game with RPG elements, such as a magic system and a weapon select system where each weapon has special effects, while implementing "Runner" mechanics in a procedurally generated world, where the goal is to cover as much distance as possible and defeat as many enemies as possible before the player loses his life. The main mechanics of the game include the infinitely generated terrain, the way in which enemies are procedurally generated, which increase their attack and speed the further the player is from the starting point. The main advantage of the game is also its dynamic combat mechanics where the player has the ability to move freely and teleport using specific weapons, and a level system that heals the player and forces him to choose which enemies to attack if he plans to survive.

Keywords: Unity, RPG, Procedural terrain generation, 3D.