SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 731

PROCEDURALNO GENERIRANJE NIZA POVEZANIH PROSTORIJA

Ana Marija Devčić

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 731

PROCEDURALNO GENERIRANJE NIZA POVEZANIH PROSTORIJA

Ana Marija Devčić

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 731

Pristupnica:	**Ana Marija Devčić (0036524526)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Proceduralno generiranje niza povezanih prostorija**

Opis zadatka:

Proučiti tehnike proceduralnog generiranja objekata. Razraditi problematiku proceduralnog generiranja prostorija. Realizirati metodu za proceduralno generiranje i povezivanje niza prostorija na proizvoljno velikim prostorima. Načiniti testiranje na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Koristiti grafički pogon Unity i programski jezik C#. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 10. lipnja 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 731

Proceduralno generiranje niza povezanih prostorija

Ana Marija Devčić

Zagreb, srpanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 731

Pristupnica:	**Ana Marija Devčić (0036524526)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Željka Mihajlović

Zadatak: **Proceduralno generiranje niza povezanih prostorija**

Opis zadatka:

Proučiti tehnike proceduralnog generiranja objekata. Razraditi problematiku proceduralnog generiranja prostorija. Realizirati metodu za proceduralno generiranje i povezivanje niza prostorija na proizvoljno velikim prostorima. Načiniti testiranje na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Koristiti grafički pogon Unity i programski jezik C#. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 10. lipnja 2022.

SADRŽAJ

1. Uvod

Proceduralna generacija se često primjenjuje u razvoju video igara zbog svoje nepredvidivosti. Jedna od njenih primjena je u generiranju nivoa, to jest prostorija i terena u kojima se igra odvija. Proceduralno generirani nivoi su česti u *roguelike* žanru igara, koji je karakterističan po tome da se može igrati beskonačno puno puta i igra je svaki put drugačija. Tradicionalno se u igrama nivoi stvaraju ručno, bili oni ručno nacrtani ili 3D modelirani. Ručno stvaranje nivoa ipak ima neke prednosti prema proceduralnoj generaciji te se stoga još uvijek koristi.

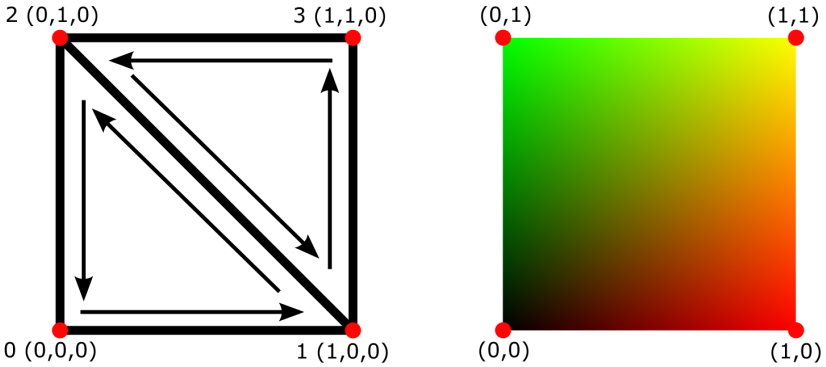
Za nivoe je bitno da ne budu monotoni kako igraču ne bi bili dosadni. Pri ručnom izrađivanju nivoa dizajner zna kako ga učiniti zanimljivim. Proceduralna generacija, s druge strane, često dovodi do monotonije i toga da sve izgleda nasumično razbacano. Potrebno je naći neki balans između efikasnosti i ponovne uporabljivosti proceduralne generacije te prirodnosti i umjetničke vrijednosti ručno dizajniranih nivoa. Kako bi se postiglo prirodno generiranje nivoa potrebno je proučiti postojeće tehnike proceduralne generacije te ih primjeniti na vlastito rješenje. Za testiranje će biti potrebno implementirati nekoliko parametara koji utječu na generaciju levela kako bi se mogli usporediti, te kako bi se moglo procjeniti što daje najbolje rezultate.

2. Metode proceduralne generacije

2.1. Proceduralno generiranje objekata

Proceduralna generacija je stvaranje virtualnih objekata programski umjesto ručno. Jedna primjena proceduralne generacije je stvaranje trodimenzionalnih okruženja, što se u ovom radu proučava. [10]

Generiranje objekata u trodimenzionalnom prostoru zahtjeva programsko stvaranje 3D modela. Ovo se najčešće radi definiranjem niza točki u prostoru, te trokuta koji te točke čine. Koordinate točaka se računaju programski na osnovu željenih parametara, te se zatim one povezuju u trokute koje čine plašt 3D modela. Za svaku točku je često potrebno generirati ne samo njene koordinate u prostoru već i UV koordinate koje se koriste pri teksturiranju objekta. Točke se povezuju u poligone tako da se u datoteci modela definira skup od tri ili više točaka za svaki pojedini poligon. [2] Najčešće se upotrebljavaju trokuti jer su najjednostavniji. Na slici 2.1 grafički je prikazana struktura 3D modela na jednostavnom primjeru kvadrata sastavljenog od dva trokuta. U modelu bi nizovi (0,1,2) i (1,3,2) činili dva trokuta. Redoslijed točaka u trokutu je bitan kako bi se znalo u kojem smjeru je on okrenut.

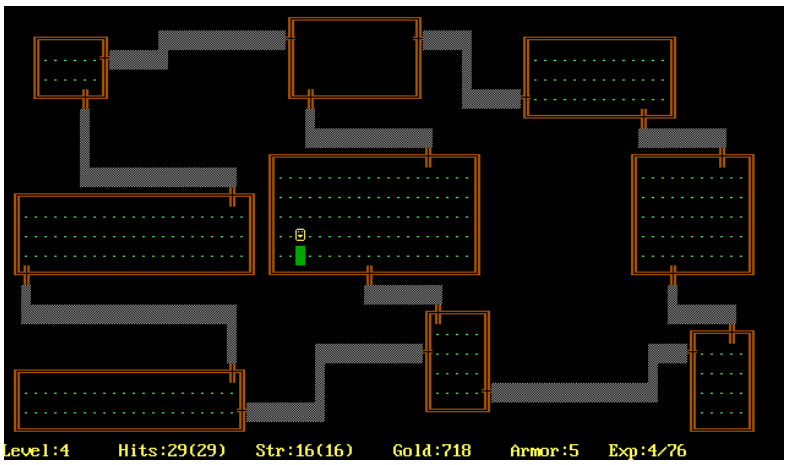


Slika 2.1: Dijagram točaka i UV koordinata kvadrata

U primjeru su točke definirane u smjeru suprotnom od kazaljke na satu no u stvarnosti to ovisi o implementaciji. Na desnoj strani dijagrama vidi se primjer teksture i četiri točke na teksturi koje se pridodjeljuju točkama na modelu.

2.2. Pojedine prostorije

Jedna od metoda proceduralnog generiranja prostorija je da sve moguće prostorije imaju zadane dimenzije i izgled, te je samo njihov raspored proceduralno generiran. Također je često da su sve prostorije istih dimenzija. Ovakav pristup je najlakši za implementirati ali dovodi do prostora koji je predvidiv i monoton, no ovisno o dizajnu igre može biti i poželjan. Alternativa tome je da se oblik prostorije generira proceduralno. Ovako se dobivaju raznolikije prostorije, no teže je za implementirati i u praksi je često neisplativo. [8]



Slika 2.2: Primjer proceduralne generacije u igri *Rogue* (1980.)

Korištenje proceduralne generacije prostorija je često u dvodimenzionalnim igrama. Na slici 2.2 se može vidjeti jedan od najranijih primjera proceduralnih prostorija u igrama iz računalne igre *Rogue* [9]. U tri dimenzije je proceduralna generacija teža za implementirati i stoga manje česta. Za predefinirane prostorije je potrebno napraviti 3D model prostorije umjesto da je se samo nacrta ili da se slaže od prethodno nacrtanih pločica kao u 2D, a ako se same prostorije proceduralno generiraju potrebno je proceduralno napraviti i 3D model.

2.3. Raspored prostorija

Jedan od načina raspoređivanja prostorija je da se prvo generira apstraktna reprezentacija slijeda prostorija, primjerice u obliku grafa, koji se zatim koristi kako bi se redom generirale prostorije. Još jedan način je da se prvo generiraju prostorije u 3D prostoru te se zatim povežu hodnicima ili vratima. Među ostalim metodama postoje i genetski algoritmi i generativne gramatike. [7]

Pri generiranju rasporeda potrebno je voditi računa o tome da se prostorije međusobno ne sijeku. Jedan način da se ovo izvede je da se u nekoj strukturi podataka, na primjer dvodimenzionalnoj matrici, vodi računa o pozicijama na kojima se prostorije nalaze te onima koje su prazne. Ovaj pristup je ograničen po tome da se može generirati samo po rešetki određenih dimenzija, te su prostorije najčešće samo pravokutne. Drugi način je da se koristi simulacija fizike kako bi se provjeravale kolizije između prostorija. Ovaj način je jednostavniji za implementirati i fleksibilniji po pitanju oblika prostorija ali mu je potrebno više vremena.

Osim generiranja samog rasporeda potrebno je prostorije napuniti interaktivnim i dekorativnim objektima kako ne bi bile potpuno prazne. Ovo se najčešće ne radi potpuno nasumično nego se na osnovu nekih parametara određuje gdje će se objekti staviti. Na primjer, neki objekti se stavljaju u kutevima a neki na zidove, te ih je pri generaciji potrebno ispravno pozicionirati. [5]

3. Korištene tehnologije i alati

3.1. Razvojni pogon *Unity3D*

Unity3D je razvojni pogon za izradu igara, simulacija i drugih interaktivnih medija. Razvija ga tvrtka Unity Technologies kako bi razvoj igara učinili pristupačnijim većem broju ljudi. Danas se vrlo često koristi za izradu igara.

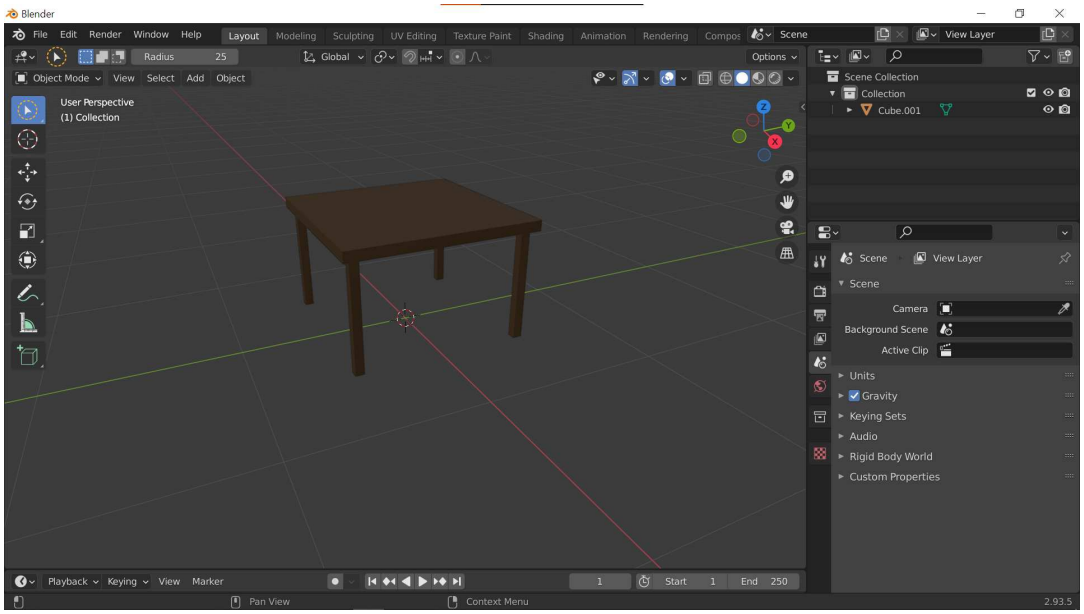
Potrebno je definirati neke osnovne pojmove koji se koriste u radu. Hijerarhija scene u Unity-u sastoji se od objekata zvanih *GameObjects*. Objekti mogu biti djeca scene ili drugih objekata. Svaki *GameObject* sastoji se od komponenti ili *Components*, koje definiraju izgled i ponašanje *GameObject*-a. Primjerice, svaki objekt mora imati *Transform* komponentu koja definira njegovu poziciju u prostoru, rotaciju te skaliranje. Neke od komponenti koje objekt može, ali ne mora imati su *MeshRenderer* za iscrtavanje 3D modela, *Collider* za detekciju kolizije, *Particle System* za simulaciju čestica i mnogo drugih. Korisnici mogu definirati vlastite komponente kao skripte napisane u programskom jeziku C#, te se u istima najčešće nalazi sama logika igre ili aplikacije. One se pridružuju *GameObject*-ima na isti način kao i Unity-eve zadane komponente.

Od bilo kojeg *GameObject*-a moguće je stvoriti predložak ili *Prefab*. Oni se mogu proizvoljno instancirati u sceni, bilo to ručno u grafičkom sučelju ili programski unutar neke skripte.

Osim komponenti, skriptama se mogu definirati i takozvani *ScriptableObjects*. Njihova svrha je da sadrže podatke koji nisu nužno vezani ni za koji *GameObject*, nego postoje sami za sebe.

3.2. Alat za 3D modeliranje *Blender*

Blender je besplatni program za 3D modeliranje te se koristi u brojnim industrijama uključujući razvoj igara. Ovdje je korišten za izradu objekata koji bi se postavili u prostorije. Sadrži brojne alate za izradu i manipulaciju 3D objekata, primjerice modeli-



Slika 3.1: Blender-ovo korisničko sučelje

ranje, osvjetljenje, teksturiranje, simulacija fizike i animacija, no u svrhu rada korišteni su samo osnovni alati za modeliranje. [4]

3D modeli napravljeni u Blender-u mogu se pretvoriti u .fbx format koji se zatim može koristiti u Unity projektu. Na slici 3.1 može se vidjeti korisničko sučelje te primjer jednog od modela napravljenih za programsko rješenje.

4. Programsko rješenje

Cilj je bio stvoriti proceduralni generator prostorija s određenim parametrima. Izgledom liči na unutrašnjost dvorca, a prostorije se granaju poput labirinta. Pojedine prostorije se generiraju proceduralno.

Prostorija u sceni definirana je pomoću *Room* komponente, za koju je napisana skripta Room.cs. U njoj su definirani atributi prostorije kao što su širina, duljina itd. Generator prostorija definirana je klasa *RoomGenerator* koja instancira objekte s komponentom *Room*.

3D modeli korišteni u rješenju su napravljeni ručno u *Blender*-u, a teksture su preuzete s interneta [1]. Rezultat je javno dostupan na GitHub-u. [3]

4.1. Definicija prostorija

Za definiciju pojedine vrste prostorije napravljen je *ScriptableObject* zvan *RoomData*. Pojedine vrste prostorija moguće je definirati stvaranjem *RoomData* objekta i postavljanjem njegovih parametara direktno u Unity-evom grafičkom sučelju. Za svaku prostoriju je moguće kao prvo definirati minimalnu širinu i duljinu te maksimalnu širinu i duljinu, zatim materijale koji se koriste na zidovima i podu, objekti koji se postavljaju u prostoriju, te maksimalni broj novih prostorija koji se iz nje granaju. Ovim pristupom moguće je definirati velik broj različitih prostorija tako da konačan prostor nije monoton. Na slici 4.1 može se vidjeti jedan primjer ovako definirane prostorije. Unutar *RoomGenerator* klase postoji lista prostorija koje je moguće generirati. Unutar Unity-evog inspektora svi željeni *RoomData* objekti se mogu dodati u generator.

4.2. Generiranje pojedinih prostorija

Kako bi se mogle stvarati prostorije proizvoljnih dimenzija potrebno ih je programski generirati. Jedan od načina na koji bi se ovo moglo izvesti je da se instanciraju modeli

Slika 4.1: Primjer vrste prostorije definirane pomoću RoomData

dijelova zidova u obliku sobe. Međutim, takav pristup može dovesti do loših performansi jer se radi o velikom broju objekata u memoriji računala. Drugi način, koji će se ovdje koristiti, je da se generira kvadar u obliku sobe željenih dimenzija.

Modeli se generiraju unutar Unity-eve *Mesh* klase te se stvoreni Mesh dodjeljuje *Mesh Renderer* komponenti.

4.2.1. Generiranje modela

U Unity-u se 3D modeli definiraju pomoću klase *Mesh* koja sadrži metode za rad s modelima. Imaju nekoliko određenih polja kojima je moguće pridijeliti vrijednosti. Potrebno je definirati točke u prostoru, indekse točaka koje čine trokute, te njihove UV koordinate na teksturi. [6] Za generiranje modela prostorija napisana je klasa *RoomMeshGenerator*. Ona u sebi sadrži tri liste: lista trodimenzionalnih vektora koje predstavljaju točke, lista cijelih brojeva kao indekse, te listu dvodimenzionalnih vektora kao UV koordinate. Također su joj zadani parametri o prostoriji koju treba generirati: njenu duljinu, širinu, visinu, te debljinu zida.

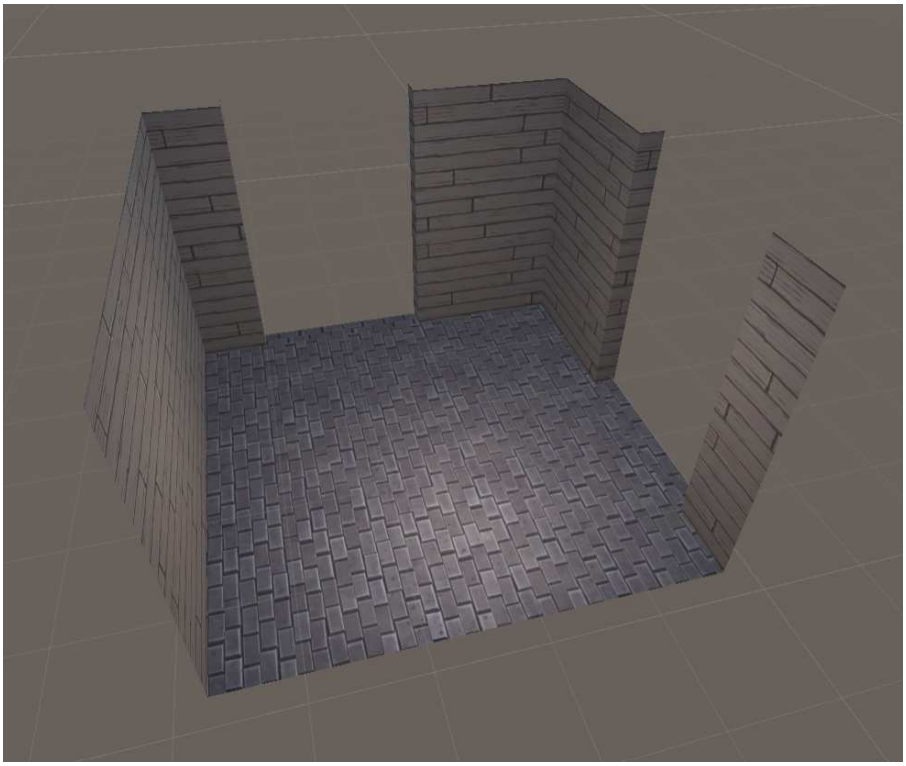
Temeljna metoda koju koriste sve ostale metode je *AddQuad*, koja prima četiri vrha kao argumente te stvara četverokutnu plohu između tih zadanih vrhova. To čini tako da prvo doda zadane vrhove u popis vrhova, zatim za te vrhove izračuna UV

koordinate i doda ih u popis UV koordinata, te napokon poveže vrhove u dva trokuta tako da ih redom u smjeru kazaljke na satu doda u popis indeksa. U isječku koda 4.1 se može vidjeti cijela metoda. Pri zvanju metode bitno je imati na umu da bi normale plohi trebale biti okrenute prema unutrašnjosti tijela, ne prema van kao što se to u 3D modeliranju tipično radi, budući da se igrač nalazi unutar prostorije te se ne bi trebalo vidjeti izvana.

```
1  private void AddQuad(
2     Vector3 tl, Vector3 tr, Vector3 bl, Vector3 br,
3     List<Vector3> verts, List<int> indices, List<Vector2> uvs)
4  {
5     int n = verts.Count;
6     verts.Add(tl);
7     verts.Add(tr);
8     verts.Add(bl);
9     verts.Add(br);
10
11    float w = (br-bl).magnitude*textureScaleFactor,
12        h = (tl-bl).magnitude*textureScaleFactor;
13
14    uvs.Add(new Vector2(0, 0)); // bottom left
15    uvs.Add(new Vector2(w, 0)); // top left
16    uvs.Add(new Vector2(0, h)); // bottom right
17    uvs.Add(new Vector2(w, h)); // top right
18
19
20    indices.Add(n);
21    indices.Add(n+1);
22    indices.Add(n+2);
23    indices.Add(n+1);
24    indices.Add(n+3);
25    indices.Add(n+2);
26 }
```

Listing 4.1: Metoda *AddQuad* u skripti *RoomMeshGenerator.cs*

Za izgradnju samih soba postoje metode *CreateFloorAndCeiling*, *CreateWall* te *CreateDoorway*. *CreateFloorAndCeiling* stvara dvije plohe zadanih dimenzija kao pod i strop. Kako bi pod imao drugačiji materijal od zidova, potrebno je indekse poda dodati u novu listu indeksa, koji će se pri iscrtavanju modela gledati kao zasebni model



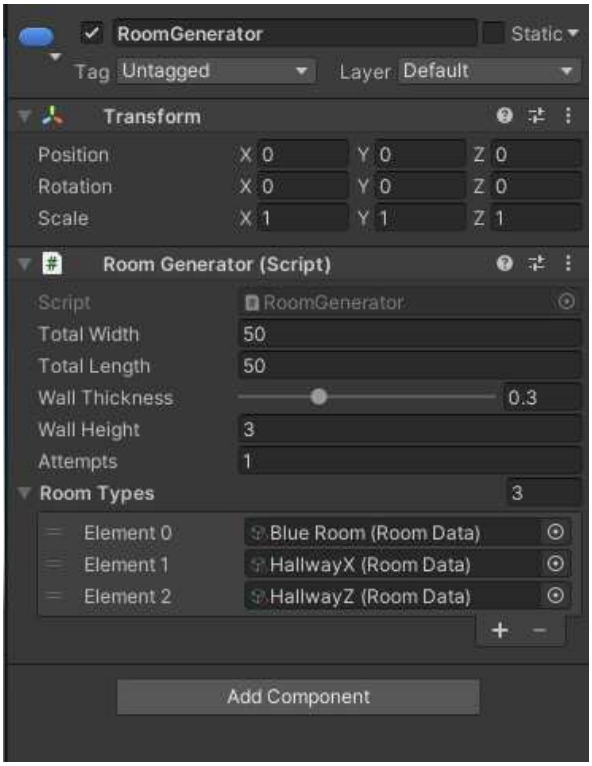
Slika 4.2: Primjer jedne prostorije s vratima

sa zasebnim materijalom. *AddWall* stvara zid određene orijentacije, duljine i debljine. To čini tako da stvara plohe od početka do kraja zida, ostavljajući rupe za sva vrata definirana na tom zidu. Kako bi se dobio dojam debljine zida, plohe su pomaknute prema unutra za određenu malu udaljenost. Za svaka vrata na zidu poziva se metoda *AddDoorway* koja stvara obrub zadane debljine kako ne bi ostala praznina između prostorija. Na slici 4.2 može se vidjeti primjer jedne prostorije generirane i teksturirane ovom metodom.

4.3. Raspored prostorija

Sami generator rasporeda ima nekoliko varijabli koje utječu na proces generacije. Prvi su ukupna širina i visina generiranog prostora. Idući su visina i debljina zidova, koji ne utječu na raspored prostorija nego samo na njihov izgled. Zatim je određen broj pokušaja kojim se svaka pojedina soba generira. Na kraju se zadaje je li generacija ide prvo u širinu ili u dubinu. Na slici 4.3 vidi se primjer konfiguracije generatora u Unity-evom grafičkom sučelju.

Za početak se na poziciji (0, 0) instancira prazni objekt kojem se dodaje komponenta *Room*. Parametri te prostorije se zatim inicijaliziraju na osnovu nasumično



Slika 4.3: Primjer konfiguracije generatora

odabranog *RoomData* objekta iz kolekcije. Na osnovu njegovog faktora grananja generator stvori zadani broj vrata na nasumičnim pozicijama i poziva metodu *AddDoor* kako bi toj prostoriji dodao vrata na toj poziciji.

Nakon što je prva prostorija stvorena redom se dodaju nove prostorije dok se ne popune zadane dimenzije. Opet se instancira nova nasumična prostorija i povezuje je s prvom pomoću metode *LineUpWith*. Metoda *LineUpWith* prvo stvara vrata u prostoriji na zidu suprotnom zadanim vratima, primjerice ako je roditelj na desnom zidu prve sobe vrata će se dodati na lijevom zidu druge. Zatim mijenja poziciju prostorije kako bi stara vrata i nova vrata bila povezana. U isječku koda 4.2 može se vidjeti cijela metoda. Program zatim čeka idući *FixedUpdate* kako bi se mogle provjeriti kolizije. Kolizije se provjeravaju automatski u Unity-u pomoću *Box Collider* komponente postavljene na prostoriju. Ako se nova soba ne sječe s bilo kojom drugom, i ako dodavanjem sobe ukupna veličina ne bi premašila maksimum, ona se dodaje u listu prostorija, granice se prošire kako bi obuhvatile novu prostoriju, i dodaje joj se *maxBranches* broj vrata, te se sva nova vrata dodaju u red. U suprotnom, to jest ako se ne može uspješno dodati, briše prostoriju i generira novu dok se uspješno ne doda ili do određenog broja pokušaja. Zatim se iz reda uzimaju iduća vrata i postupak se ponavlja.

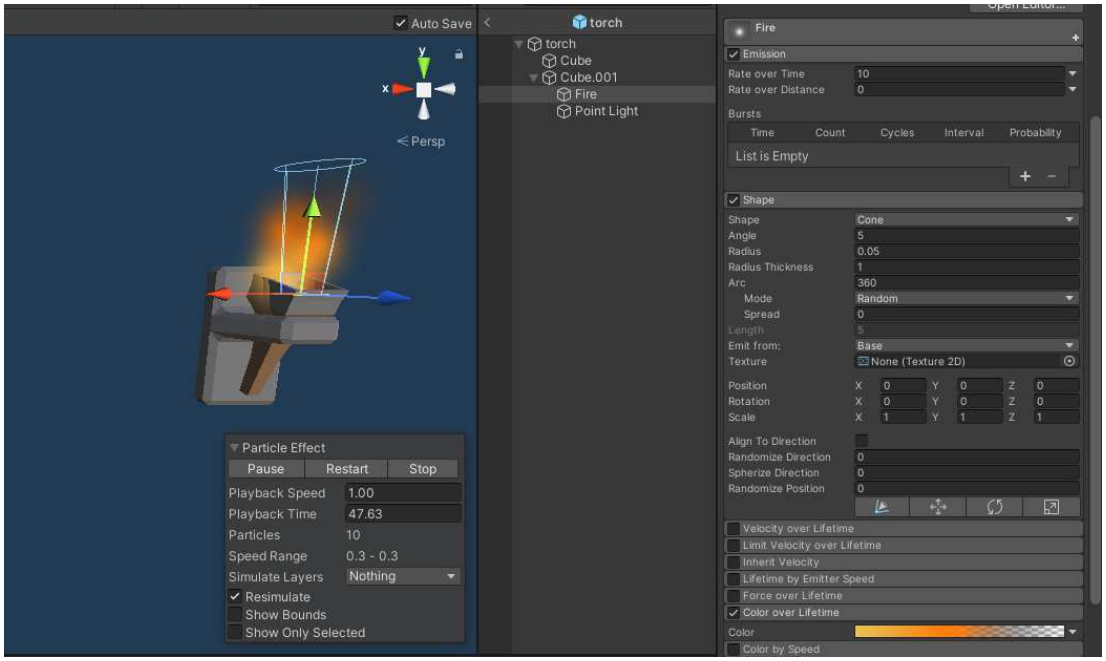
```
1 public void LineUpWith(Door door) {
```

```
    Room parent = door.parent;
    RoomMeshGenerator.Wall opposite =
         meshGenerator.GetOppositeWall(door.wall);
    Door newDoor =
        AddDoor(Random.Range(1,
        meshGenerator.GetWallWidth(opposite)-doorWidth),
         opposite);
    transform.position =
        parent.transform.position + parent.GetVectorToDoor(door)
         - GetVectorToDoor(newDoor) -
          meshGenerator.getWallDir(newDoor.wall)*doorWidth;
}
```

Listing 4.2: Metoda *LineUpWith* u skripti *Room.cs*

Nakon što se red isprazni i prestanu se generirati nove prostorije, generator još jednom prođe sve sobe u listi rooms i na njima zove metodu Populate kako bi ih popunio objektima. Nakon ovog je postupak gotov i dobiveni prostor se može pregledati. Primjer gotovog prostora iz perspektive igrača vidi se na slici 4.4.

Slika 4.4: Primjer generirane sobe iz perspektive igrača



Slika 4.5: Prefab baklje sa prikazom parametara sustava čestica

4.4. Populiranje prostorija objektima

Jednom kad je oblik prostorije određen, potrebno ju je popuniti predmetima. Za to se koriste *Prefab* objekti koji se u inspektoru mogu dodati u *RoomData* objekt. Relacija između objekta i prostorije definirana je pomoću *ScriptableObject* klase *RoomItem*. Klasa *RoomData* sadrži listu *RoomItem* objekata koji definiraju vrste objekata u toj prostoriji. Ovaj pristup je vrlo fleksibilan jer se na *Prefab* objekt može stavili bilo što: skripte, izvori svjetla ili zvuka, čestice itd.

U klasi *RoomItem* definiran je enumerator *Position*. Njegova svrha je da za pojedini *RoomItem* definira gdje će biti postavljen. Ima četiri vrijednosti: *Corners*, *Center*, *Walls* i *Random*. Sami *RoomItem* ima *Position* varijablu u kojoj je određeno gdje je smješten. Zatim ima float varijablu offset koja određuje njegovu udaljenost od zidova ili poda. Ima boolean varijablu *yStretch* koja, ako je istinita, označava da se objekt proteže od poda do stropa. Float varijabla *frequency* utječe na količinu generiranih objekata gdje je primjenjivo. Napokon, ima definiran i sami *Prefab* koji predstavlja objekt.

4.4.1. Modeliranje objekata

Objekti su modelirani u programu Blender, pa su zatim pretvoreni u *Prefabs* u Unity-u. Jedan od napravljenih objekata je baklja koja stoji na zidu. Nakon što je njen model

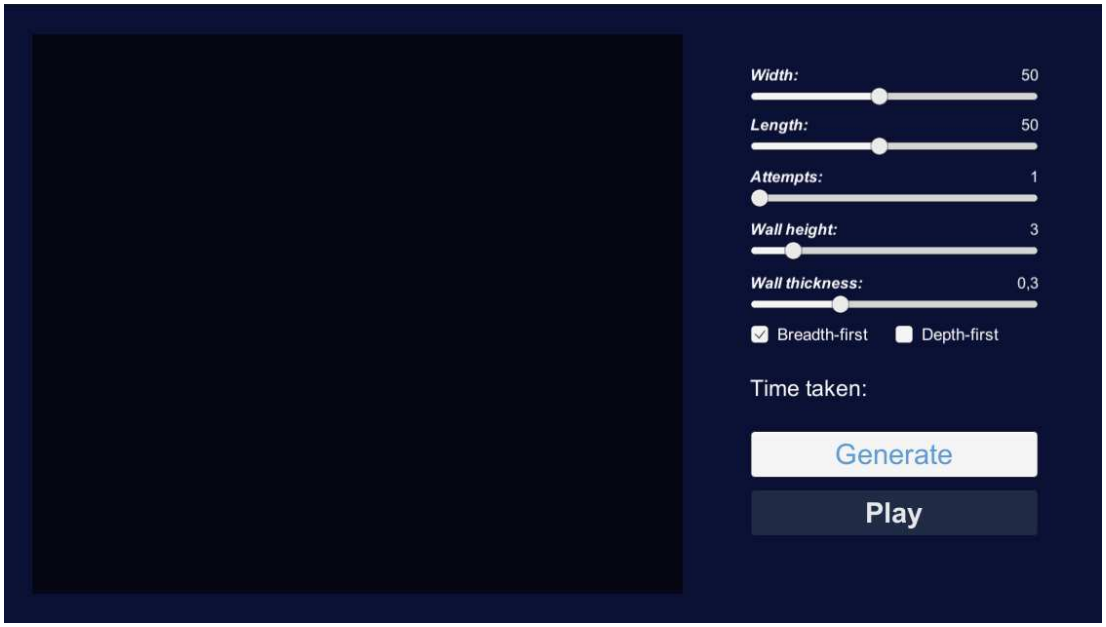
stavljen u *Prefab*, na njega su dodani *Particle System* koji stvara efekt vatre te izvor svjetla kojeg baklja emitira. Na slici 4.5 vide se postavke sustava čestica za baklju. Također su napravljeni stup i stol za koje je bilo dovoljno samo napraviti model.

4.4.2. Postavljanje objekata

U skripti *Room.cs* definirana je metoda *Populate*. Ona redom gleda *RoomItem* objekte pridružene prostoriji te ih instancira u prostoru zvanjem metode *Instantiate*. Najprije, ako je *yStretch* varijabla istinita, objekt se skalira kako bi mu visina bila jednaka visini zida. Zatim se gleda način na koji je objekt smješten. Ako je vrijednost *pos* varijable objekta jednaka *Position.Corners*, metodom *GetCorners* se dohvaćaju kutevi prostorije sa zadanom udaljenošću te se u svakom kutu objekt instancira. Ako joj je vrijednost *Position.Center* onda se metodom *GetCenter* dobiju koordinate sredine prostorije, te se objekt instancira na tim koordinatama. Ako je vrijednost *Position.Random* onda se biraju nasumične koordinate bilo gdje u prostoriji. Ako je vrijednost *Position.Walls* potrebno je instancirati objekte duž svaki zid prostorije. Petljom se prođe kroz sva četiri zida, te se još jednom petljom prolazi po pojedinom zidu. Za svaku dobivenu poziciju se metodom *CheckIfPositionValid* gleda je li se na toj poziciji smije stvoriti objekt, to jest je li se na toj poziciji nalazi zid ili su na njoj već postavljena vrata. Ako je pozicija validna onda se objekt na njoj instancira. Također je potrebno voditi računa da početna orijentacija objekta bude okomita na zid.

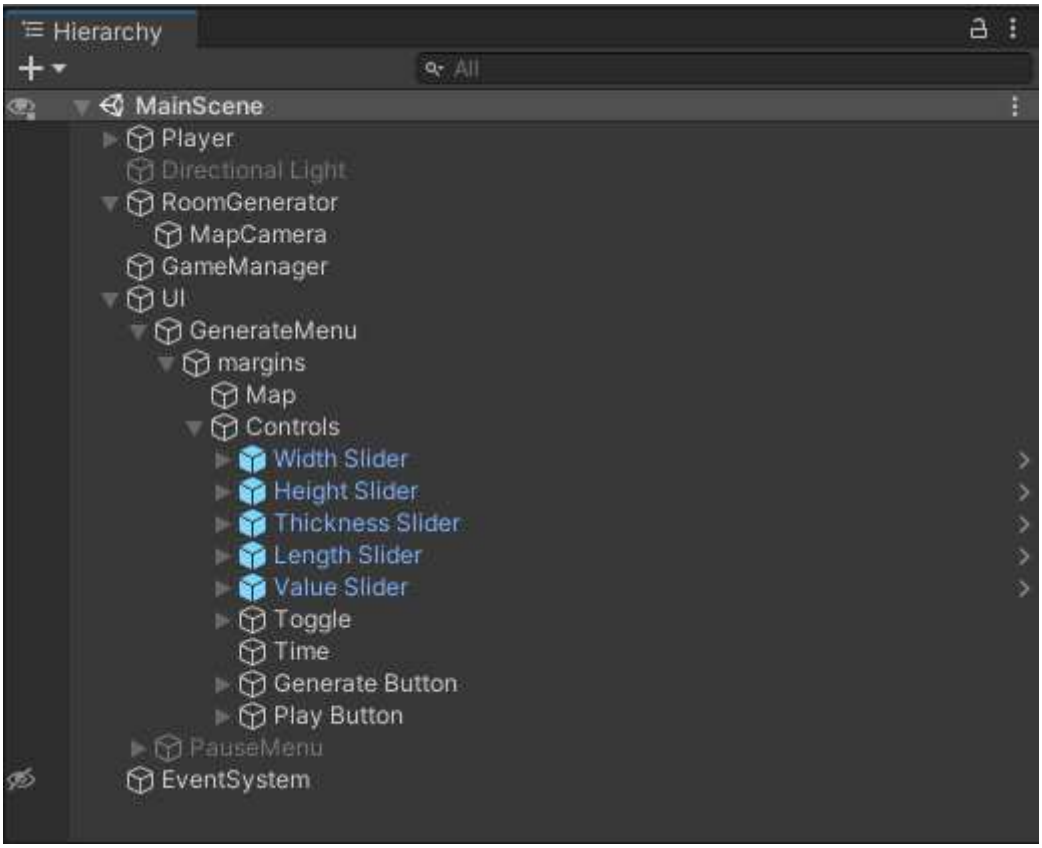
5. Korisničko sučelje

Pri pokretanju programa korisnik se susreće sa sučeljem koje se sastoji od prozora za pretpregled te opcija sa strane. Može se vidjeti na slici 5.1. Pomicanjem klizača korisnik može mijenjati razne parametre generacije: ukupnu širinu i duljinu prostora, broj pokušaja, visinu i debljinu zidova te redoslijed generiranja. Nakon postavljanja željenih parametara, pritiskom na gumb "*Generate*" se pokreće proceduralna generacija, te se konačni rezultat može vidjeti u prozoru na lijevoj strani. Iznad gumba se prikazuje koliko je vremena u sekundama trajalo generiranje. Ovaj postupak se može ponoviti proizvoljno mnogo puta dok korisnik ne dobije rezultat s kojim je zadovoljan. Zatim se pritiskom na gumb "*Play*" ulazi u interaktivni pregled prostora u kojem se korisnik može slobodno kretati i istraživati.



Slika 5.1: Izgled korisničkog sučelja

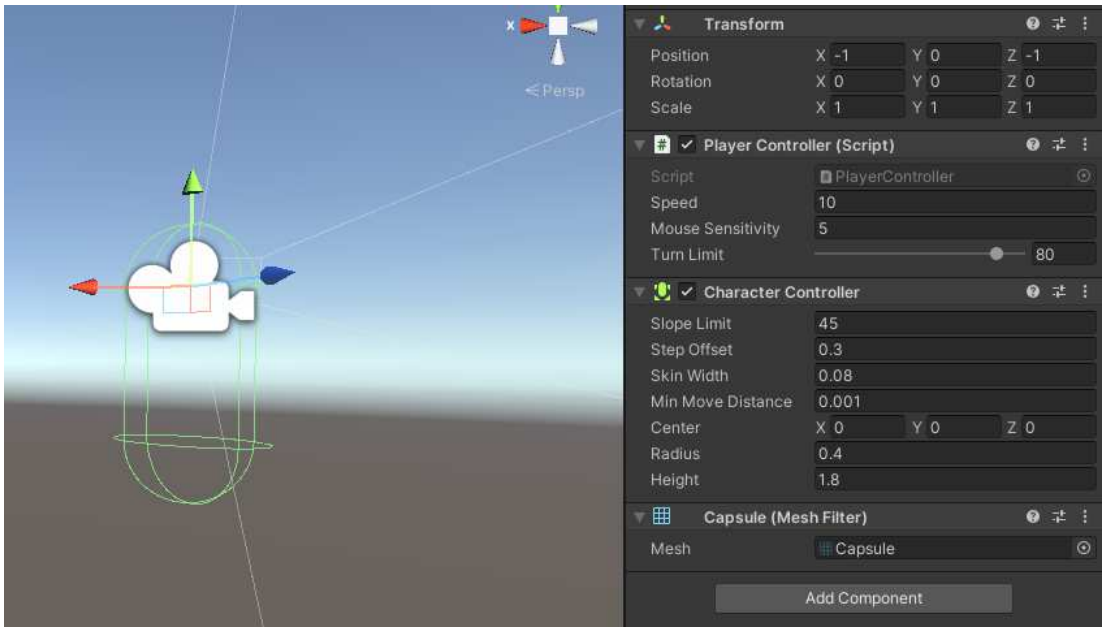
Za interaktivni pregled napisana je skripta *PlayerController.cs* koja pomiče igrača po prostoru. Postavke igrača mogu se vidjeti na slici 5.3. Igrač se kreće tipkama *WASD* ili strelicama te može gledati okolo pomicanjem miša. Unutar interaktivnog pregleda



Slika 5.2: Prikaz hijerarhije scene

korisnik može pritisnuti tipku *Escape* na tipkovnici kako bi pauzirao program, te mu se prikažu opcije za nastavak programa ili povratak u glavni izbornik. Također se može vratiti u interaktivni pregled ponovnim pritiskom tipke *Escape*.

Cijelo korisničko sučelje napravljeno je unutar Unity-evog grafičkog sučelja. Stablo scene se može vidjeti na slici 5.2. Za pretpregled generiranog prostora korištena je ortografska kamera postavljena iznad generiranih soba. Njen pogled se mapira na *RenderTexture* koji se koristi kao tekstura na *Panel* komponenti. Za klizače je napravljena skripta *Slidertext.cs* koja vrijednost klizača prikazuje tekstom, za bolju čitljivost.



Slika 5.3: Postavke igrača u sceni

6. Upotrebljivost rezultata i moguća proširenja

6.1. Ispitivanje programa

Nakon što je napravljen proceduralni generator koji prima različite parametre, potrebno je ispitati koliko dobro radi i s kojim parametrima. Na glavnom izborniku se nakon generiranja prikazuje koliko je sekundi trajalo generiranje, dok se direktno u Unity-u može vidjeti koliki je *framerate* ili broj iscrtanih slika u sekundi.

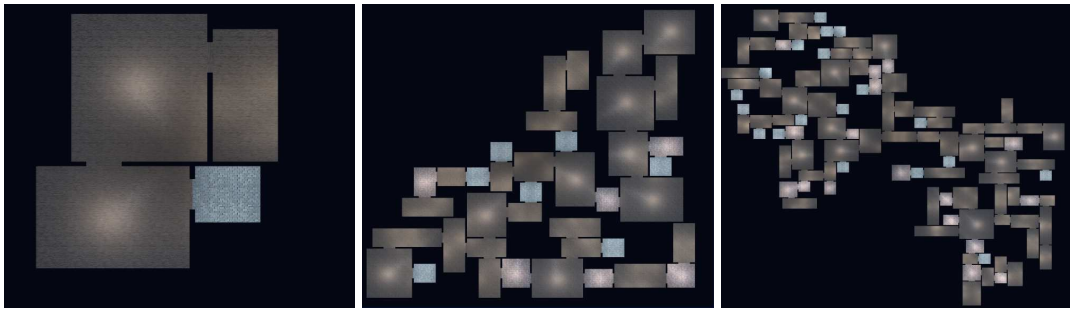
U svrhu testiranja stvoreno je nekoliko *RoomData* objekata koje će generator stvarati. Unutar Unity-evog grafičkog sučelja su dodani u RoomGenerator. Njihovi parametri su prikazani u tablici 6.1. Svaki od njih također ima definirane teksture i objekte ali u svrhu testiranja rasporeda ih nije potrebno navesti.

RoomData	maxBranches	minSize	maxSize
BasicRoom	3	(5,5)	(10,10)
HallwayX	3	(5,4)	(15,4)
HallwayZ	3	(4,5)	(4,15)
LargeRoom	6	(7,7)	(12,12)
TableRoom	0	(4,4)	(4,4)
WoodRoom	4	(4,4)	(7,7)

Tablica 6.1: Popis vrsta prostorija korištenih u testiranju

6.1.1. Dimenzije prostora

Prvo je testiran parametar veličine prostora. Generiranje većih prostora općenito traje dulje od manjih, što je logično jer ima više prostorija za stvoriti. Za najmanje moguće dimenzije, 10 metara u širinu i duljinu, obično traje svega nekoliko stotinki sekunde.



Slika 6.1: Primjeri rezultata za širinu i visinu 10, 50 i 100

Može se dogoditi da stvori samo jednu prostoriju te da mu odmah ponestane mjesta. Na srednje velikom prostoru širine i duljine 50 metara generiranje traje jednu do dvije sekunde, a na najvećim mogućim dimenzijama, 100 metara u širinu i duljinu, traje između tri i četiri sekunde. S veličinom prostorija se također smanjuje *framerate*. Na najmanjim prostorima je u prosjeku 450 sličica u sekundi, na srednjim u prosjeku 200, a na najvećim u prosjeku 100. U svim ovim primjerima je broj pokušaja postavljen na jedan. Primjeri rezultata mogu se vidjeti na slici 6.1.

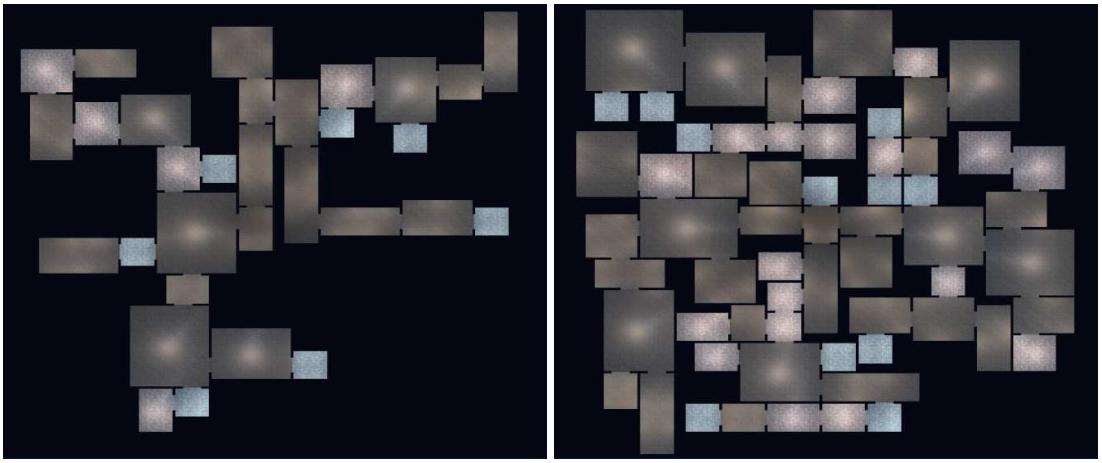
Prostorije širine i visine između 30 i 60 se čine dovoljno velike da budu zanimljive u igri, a da se još uvijek generiraju prihvatljivom brzinom i da imaju dobre performanse.

6.1.2. Broj pokušaja

Broj pokušaja se ovdje odnosi na broj puta koliko će generator probati stvoriti sobu ako prvi put ne uspije, bilo zbog kolizija ili prekoračenja maksimalnih dimenzija. Kad ima vrijednost 1 često se dogodi da generiranje prestane prije nego što se uopće popune zadane dimenzije. Očekuje se da će s većim brojem pokušaja prostor biti bolje popunjen, no da će generiranje biti sporije. Na srednje velikom prostoru s brojem pokušaja 1, kao što je već pokazano, generiranje traje jednu do dvije sekunde. S brojem pokušaja 5 prostor je vidljivo bolje popunjen ali generiranje traje tri do četiri sekunde. S brojem pokušaja jednakim 10 prostor je opet popunjeniji, i rijetko se pojavljuju prostori nedovoljnih dimenzija, no traje 8 do 10 sekundi što je znatno veće. Primjeri rezultata vidljivi su na slici 6.2. Vrijednosti između 3 i 5 općenito daje najbolje rezultate s prihvatljivim performansama.

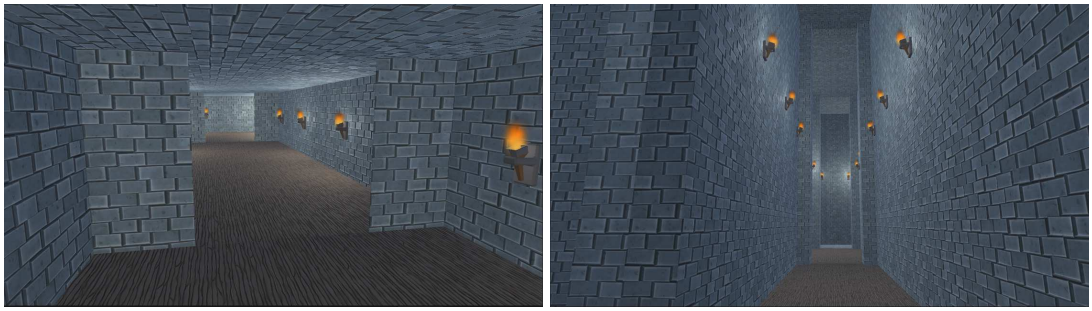
6.1.3. Debljina i visina zida

Na ovim parametrima nije potrebno mjeriti performanse budući da oni na njih ne utječu. Svejedno se mogu isprobati različite vrijednosti kako bi se vidio njihov utjecaj



Slika 6.2: Primjeri s brojem pokušaja jednakim 1 (lijevo) i 10 (desno)

na izgled prostora. Jako tanki i jako debeli zidovi ne izgledaju prirodno te stoga nisu vrlo upotrebljivi, dok su vrijednosti između 0.4 i 0.6 najviše nalik stvarnim zidovima. Najmanja moguća visina je 2 što izgleda neprirodno i zakrčeno, vrijednosti od 3 do 4 su nalik stvarnim prostorijama, dok veće vrijednosti izgledaju neobično ali bi stilistički takav izgled mogao biti poželjan. Rezultati se mogu vidjeti na slici 6.3.

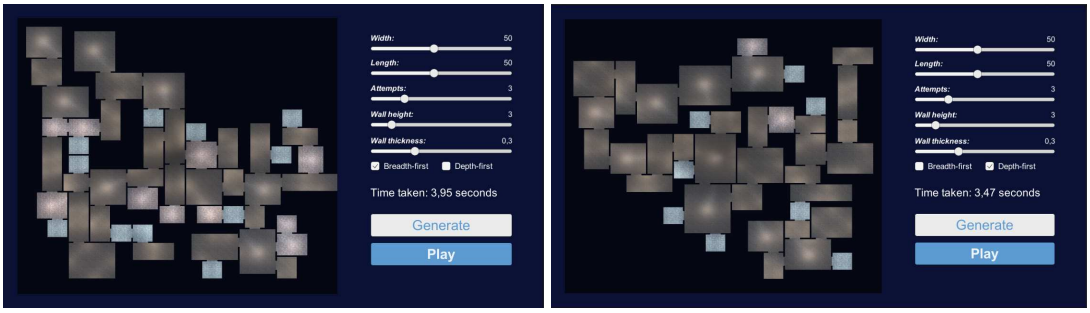


Slika 6.3: Primjer s niskim i tankim zidovima, te s visokim i debelim zidovima

6.1.4. Redoslijed generiranja

Redoslijed generiranja se odnosi na način kojim generator odlučuje gdje će stvoriti iduću prostoriju. Može biti u širinu, tako da se prostorije granaju odmah kad su stvorene, ili u dubinu, tako da se prvo generira dugi niz prostorija pa zatim grane. Obje vrste redoslijeda imaju približno iste performanse i jednako dobro popunjavaju prostor, stoga su razlike opet većinom subjektivne i izbor ovisi o željenom stilu. Generiranje u širinu daje prostor nalik labirintu, s puno različitih puteva i bez očitog početka ili kraja. Generiranje u dubinu daje više linearan slijed prostorija, s vidljivim glavnim putevima iz kojih se granaju manji putevi ili pojedine prostorije. Primjeri rezultata se

mogu vidjeti na slici 6.4.



Slika 6.4: Primjer s generiranjem u širinu (lijevo) i u dubinu (desno)

6.2. Moguća proširenja

Budući da se detekcija kolizija odvija unutar Unity-a a ne pomoću vlastite implementacije bilo bi moguće proširiti rješenje tako da se mogu koristiti prostorije bilo kakvog oblika, ne samo pravokutne. Također bi se ovako mogle dodati sobe skroz predefiniranog izgleda, s određenim pozicijama vrata kako bi se na njih moglo nadovezivati. Implementirani algoritam stvara samo sobe koje se granaju, no mogao bi se promijeniti tako da stvara petlje koje bi činile prostor zanimljivijim. Bilo bi moguće proširiti prostor na više od jednog kata tako da se dodaju prostorije sa stepenicama koje vode na kat iznad ili ispod, te se cijeli algoritam ponavlja na novom katu.

7. Zaključak

Cilj rada bio je ostvariti proceduralni generator prostorija koji prima niz parametara. Implementirana je metoda generiranja prostorija i interaktivni pregled stvorenog prostora. Generiranje prostorija ostvareno je tako da je napravljen proceduralni generator 3D modela koji na osnovu nekoliko parametara stvara prostorije željenih dimenzija. Ostvareno je povezivanje prostorija u kompletno okruženje na koje je također moguće utjecati različitim parametrima. Interaktivni pregled je ostvaren kao igrač koji može u prvom licu gledati oko sebe te se kretati kroz prostor. Na kraju je izvedeno testiranje generatora s različitim kombinacijama parametara.

Dobiveni proces generacije prilično je nepredvidiv te bi se u tom smislu mogao poboljšati. U izvršnoj datoteci je definiran jedan određeni skup prostorija no unutar Unity-a se one mogu definirati po želji. Dobiveno rješenje stoga nije samo jedan primjer generiranja prostora nego i alat kojim bi se mogle definirati puno različitih vrsti prostorija. Moguće proširenje bi bilo da se prostorije mogu definirati direktno unutar programa umjesto u Unity projektu. U stvarnoj primjeni, primjerice u igri, u prostorije bi se mogli dodati interaktivni elementi poput neprijatelja, ciljeva i mnogih drugih, no cilj ovog rada je bio samo razraditi generiranje prostorija a ne stvaranje potpune igre. Također bi, koristeći veći broj profesionalno napravljenih 3D modela, rezultat mogao izgledati estetski privlačnije i zanimljivije nego što je u svom trenutnom stanju.

LITERATURA

[1] Lumo-Art 3D. FREE Stylized PBR Textures Pack. https://assetstore.unity.com/packages/2d/textures-materials/free-stylized-pbr-textures-pack-111778.

[2] Catlike Coding. Creating a Mesh. https://catlikecoding.com/unity/tutorials/procedural-meshes/creating-a-mesh/, 2021.

[3] Ana Marija Devčić. Github repozitorij. https://github.com/amdevcic/procedural-generation-thesis, 2022.

[4] Blender Foundation. Features. https://www.blender.org/features/.

[5] Darius Kazemi. How to effectively use procedural generation in games. *Game Developer*. URL https://www.gamedeveloper.com/design/how-to-effectively-use-procedural-generation-in-games.

[6] Unity Technologies. Scripting API: Mesh. https://docs.unity3d.com/ScriptReference/Mesh.html.

[7] Roland van der Linden, Ricardo Lopes, i Rafael Bidarra. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6:78–89, 2014.

[8] Breno M. F. Viana i Selan Rodrigues dos Santos. A survey of procedural dungeon generation. *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, stranice 29–38, 2019.

[9] Wikipedia contributors. Rogue (video game) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Rogue_

(video_game)&oldid=1088295220, 2022. [Online; accessed 2-July-2022].

[10] Wikipedia contributors. Procedural generation — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Procedural_generation&oldid=1088517001, 2022. [Online; accessed 9-June-2022].

Proceduralno generiranje niza povezanih prostorija

Sažetak

Rad prati ostvarivanje proceduralnog generatora interaktivnog virtualnog okruženja. Predstavljene su različite metode generiranja prostorija. Opisan je proces generiranja 3D modela i algoritam povezivanja prostorija. Objašnjen je način postavljanja objekata u prostor. Dobiveni rezultat je testiran sa različitim parametrima te su na temelju toga opisani načini na koje bi se algoritam mogao poboljšati. Programsko rješenje je napravljeno koristeći Unity i C# te je javno dostupno na internetu.

Ključne riječi: Proceduralna generacija, razvoj igara, virtualna okruženja, Unity.

Procedural dungeon generation

Abstract

This paper follows the development of a procedural generator for an interactive virtual environment. Different methods for dungeon generation are presented. The process of mesh generation and the algorithm for connecting rooms are described. The method for placing objects in the environment is explained. The obtained result was tested with different parameters and based on that, different ways in which the algorithm could be improved were described. The software solution was created using Unity and C# and is publicly available online.

Keywords: Procedural generation, game development, virtual environments, Unity.