

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2268

PROCEDURALNO GENERIRANJE GRADA U GODOTU

Matea Maračić

Zagreb, lipanj 2026.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2268

PROCEDURALNO GENERIRANJE GRADA U GODOTU

Matea Maračić

Zagreb, lipanj 2026.

ZAVRŠNI ZADATAK br. 2268

Pristupnica: **Matea Maračić (0036555939)**
Studij: Elektrotehnika i informacijska tehnologija i Računarstvo
Modul: Računarstvo
Mentorica: prof. dr. sc. Željka Mihajlović

Zadatak: **Proceduralno generiranje grada u Godotu**

Opis zadatka:

Proceduralno generiranja sadržaja predstavlja jedan od temeljnih izazova u modernoj računalnoj grafici. Cilj rada je istražiti mogućnosti programskog alata Godot, s posebnim naglaskom na razvoj sustava za proceduralno generiranje gradova. Zasebno razmotriti generiranja osnove, odnosno prometnica i pločnika, zatim generiranja građevina koje smještaju u blokove i na posljétku drugih sadržaja. U razmatranje uzeti u obzir smislenost i povezanost postavljenih komponenti. Razraditi implementaciju koja omogućuje interaktivno upravljanje ključnim parametrima za generiranje proceduralnih sadržaja. Analizirati utjecaj raznih parametara na kvalitetu ostvarenih rezultata, te dati kritički osvrt. Usporediti postignute rezultate sa sličnim rješenjima. Diskutirati upotrebljivost rješenja i dati smjernice za buduća poboljšanja. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 12. lipnja 2026.

*Zahvaljujem se mentorici Željki Mihajlović na pruženoj pomoći i savjetima pri izradi
rada koji su bili od velike pomoći.*

Sažetak

Proceduralno generiranje grada u Godotu

Matea Maračić

Razvoj trodimenzionalnih videoigara kompleksan je i vremenski zahtjevan proces koji uključuje izradu velikog broja elemenata virtualnog okruženja. Jedno od mogućih rješenja za ubrzanje razvoja jest primjena proceduralnog generiranja sadržaja, kojim se određeni elementi razine stvaraju automatski prema unaprijed definiranim pravilima.

U sklopu ovog rada istražuju se mogućnosti proceduralnog generiranja urbanih okruženja u programskom alatu Godot (v4.5.1) kroz razvoj sustava u sklopu videoigre Dron.co. Implementira se sustav za generiranje cestovne mreže, gradskih područja, građevina, parkova te drugih elemenata okruženja poput drveća i ulične rasvjete. Generirani sadržaj temelji se na skupu pravila kojima se nastoji osigurati smislen i povezan raspored elemenata unutar grada.

Cilj ovog rada jest implementirati i prikazati proceduralni sustav generiranja grada te analizirati utjecaj različitih parametara generiranja na izgled i kvalitetu dobivenih rezultata. Nadalje, razmatraju se mogućnosti daljnjeg razvoja sustava i usporedba s postojećim rješenjima. Svrha rada jest ubrzati proizvodnju i dizajn razina videoigre te istražiti mogućnosti primjene proceduralnog generiranja sadržaja.

Ključne riječi: proceduralno generiranje; računalna grafika; Godot Engine; razvoj igara; dizajn razine;

Abstract

Procedural city generation in Godot

Matea Maračić

The development of three-dimensional video games is a complex and time-consuming process that involves the creation of a large number of elements within a virtual environment. One possible approach to accelerating development is the use of procedural content generation, where certain elements of a game level are created automatically according to predefined rules.

This thesis explores the possibilities of procedural generation of urban environments using Godot Engine (v4.5.1) through the development of levels within the video game Dron.co. The system includes the generation of roads, city areas, buildings, parks and other environmental elements such as trees and street lights. The generated content is based on a set of rules designed to ensure a coherent and complex layout of the urban environment.

The goal of this project is to accelerate the design and development of game levels by implementing a procedural city generation system in Godot Engine. Additionally, the impact of different generation parameters on the appearance and quality of the generated results is analyzed, along with possibilities for further development of the system.

Keywords: procedural generation; computer graphics; Godot Engine; game development; level design;

Sadržaj

Sažetak	1
Abstract	2
1. Uvod	5
2. Teorijska osnova	6
2.1. Proceduralno generiranje	6
2.2. Proceduralno generiranje razina	8
2.3. Metode proceduralnog generiranja urbanih područja	9
3. Korištene tehnologije	11
3.1. Godot Engine	11
4. Implementacija rješenja	13
4.1. Pravila generiranja	15
4.2. Generiranje prometnica i pločnika	17
4.3. Generiranje parkova	22
4.4. Generiranje građevina	23
4.5. Postavljanje elemenata	25
4.6. Drveće i ulična rasvjeta	26
4.7. Izvorni kod programa	26
5. Rezultati, analiza i usporedba	28
5.1. Interaktivni parametri	28
5.1.1. Polazna vrijednost generatora	28
5.1.2. Utjecaj gustoće građevina	29

5.1.3. Utjecaj gustoće cestovne mreže	30
5.1.4. Utjecaj ostalih parametara	30
5.1.5. Kritički osvrt	32
5.2. Računalna zahtjevnost	33
5.3. Usporedba postojećih rješenja	33
5.4. Budući razvoj	34
6. Zaključak	36
Literatura	37
Izjava o korištenju umjetne inteligencije	39
A: Izvorni kod	40

1. Uvod

Razvoj trodimenzionalnih videoigara koje napreduju kroz razine predstavlja složen i vremenski zahtjevan proces, osobito zbog potrebe za ručnim osmišljavanjem i izradom pojedine razine. U mnogim igrama napredovanje uključuje postupno povećanje težine razine, primjerice kroz promjenu parametra brzine, broja neprijatelja i slično, što dodatno otežava ručno stvaranje razina. Zbog toga se kao jedno rješenje uvodi proceduralno generiranje sadržaja, kojim se automatski stvaraju nove razine prema unaprijed definiranim pravilima. Bitan dio problema predstavlja i dizajn odgovarajućeg seta pravila te algoritam koji omogućuje stvaranje smislenog i složenog rasporeda virtualnog okruženja.

Cilj ovog rada je istražiti mogućnosti programskog alata Godot[1] (v4.5.1) i implementirati jedno od mogućih rješenja generiranja rasporeda u sklopu igre Dron.co, razvijene kao grupni završni projekt[2]. Osim opisa implementiranog rješenja, analizirani su dobiveni rezultati te su razmotrena ograničenja i mogućnosti daljnjeg razvoja.

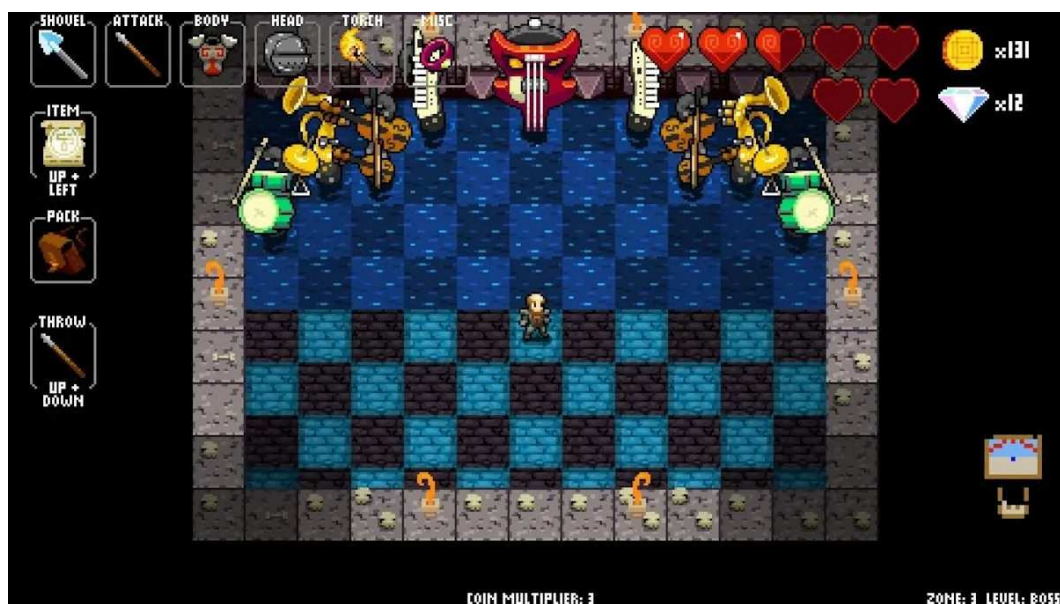
U radu se najprije obrađuje sam pojam proceduralnog generiranja te njegova primjena i važnost u razvoju igara. Daljnja poglavlja posvećena su pravilima, algoritmu te implementaciji proceduralnog generiranja u koracima. Sustav je podijeljen na generiranje cestovne mreže, formiranje gradskih područja, smještaj građevina i parkova uz dodatne detalje okoline.

2. Teorijska osnova

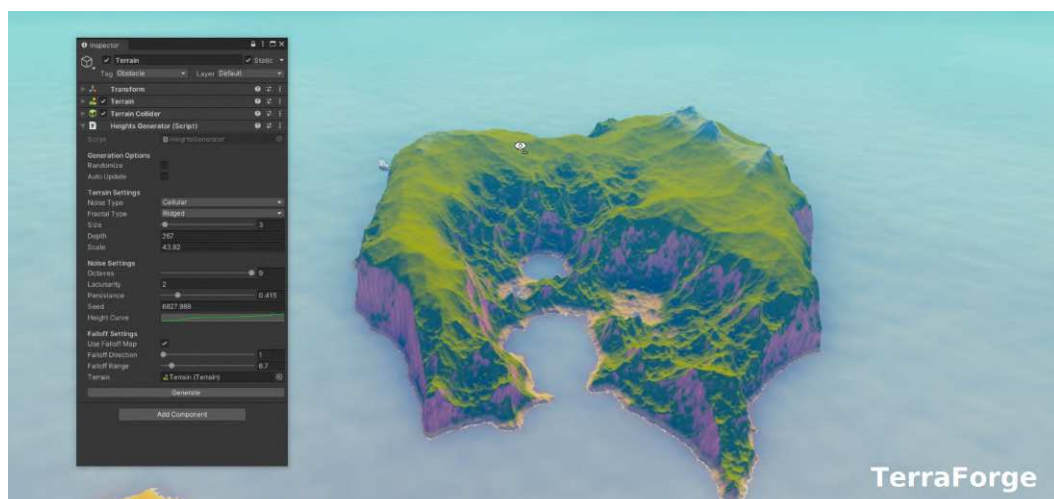
2.1. Proceduralno generiranje

Proceduralno generiranje tehnika je koja koristi algoritme i postavljena pravila kao temelj za izgradnju elemenata računalne grafike, poput terena, likova, rasporeda razina i slično. Ručno stvaranje terena za pozadinu videoigre može predstavljati mukotrpan i ponavljajući posao koji stvara problem efikasnosti i brzine razvoja igara. Primjerice, postavljanje cesta, pločnika, zgrada i ostalih detalja u dizajnu razine za videoigru Dron.co proces je koji traje nekoliko dana. Naglasak se stavlja i na videoigre otvorenog svijeta koje se oslanjaju na generiranje dovoljno velikog ili beskrajnog svijeta kako bi korisnik mogao imati različita iskustva tijekom igranja.

Proceduralno se generiranje također koristi za generiranje prirodnih oblika i tekstura, a razvojem računalne tehnologije počelo se primjenjivati i na složenije virtualne prostore. Proceduralno generiranje često se koristi za generiranje 2D videoigara. Primjer generirane 2D videoigre koja koristi raspored tamnica za dizajn razine prikazan je na slici 2.1. Primjena proceduralnog generiranja u 3D videoigramama dodatno povećava složenost postupka jer je potrebno uzeti u obzir prostorne odnose između većeg broja elemenata.



Slika 2.1. Primjer proceduralne generacije u igri Crypt of the NecroDancer [3]



Slika 2.2. Primjer proceduralne generacije terena [4]

2.2. Proceduralno generiranje razina

Proceduralno generiranje razina predstavlja primjenu proceduralnog generiranja na stvaranje prostora i elemenata unutar razine videoigre. Umjesto ručnog određivanja položaja svakog objekta, njihov se raspored određuje algoritmom prema unaprijed definiranim pravilima. Na taj se način iz istog skupa elemenata može dobiti veći broj različitih rasporeda razine.

Kod generiranja razina potrebno je uzeti u obzir funkcionalnost dobivenog rasporeda. Potpuno nasumično generiranje može proizvesti neželjen rezultat poput gradova s nepovezanim cestama, preklapajućim elementima, besmislenim položajima građevina te neprohodnim područjima. Zbog takvih posljedica potrebno je postaviti pravila o međudnosima pojedinih elemenata mape.

Proceduralno generiranje posebno je korisno kod razina koje se sastoje od većeg broja ponavljajućih elemenata ili u ovom slučaju, ponavljajućih pločica. Na slici 2.3. prikazan je primjer korištenja 3D pločica za sastavljanje većeg prostora. Sličan princip korišten je i u ovom radu, gdje se unaprijed izrađeni 3D elementi kombiniraju u alatu GridMap kako bi se formiralo urbano okruženje. Umjesto ručne izrade svakog novog rasporeda, sustav koristi skup elemenata koje kombinira i spaja u jednu cjelinu. U urbanom okruženju takvi elementi mogu biti prometnice, pločnici, parkovi, građevine i brojni detalji.



Slika 2.3. Prikaz 3D pločica terena preuzet iz završnog rada Dominika Scholza [5]

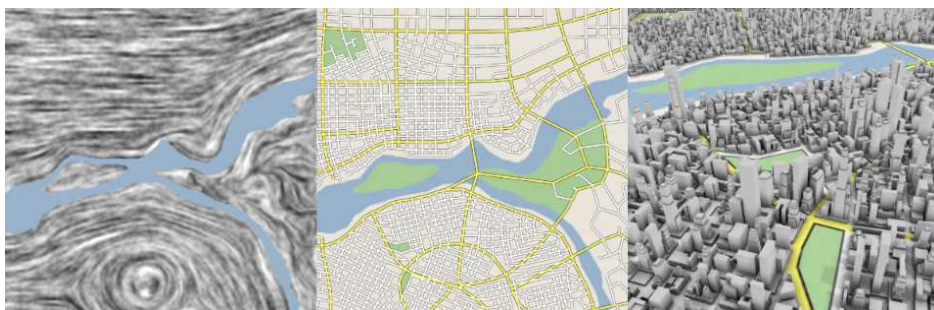
2.3. Metode proceduralnog generiranja urbanih područja

Proceduralno generiranje urbanih područja i gradskih četvrti obuhvaća skup računalnih metoda i algoritama koji omogućuju stvaranje složenijih struktura gradova, uključujući ulične mreže, građevine i druge elemente urbanog područja. Primjena proceduralnog generiranja omogućuje stvaranje većih i varijabilnih okruženja uz smanjene potrebe ručnog oblikovanja. Takvi se postupci, između ostalog, koriste u industriji videoigara, animacija, urbanističkom planiranju i geografskim sustavima.

Proceduralno generiranje urbanih područja moguće je ostvariti na temelju različitih pristupa. Među tim pristupima mogu se izdvojiti gramatike temeljene na pravilima (engl. *rule-based grammars*)[6], pristupi temeljeni na grafovima te matematički modeli poput Lindenmayerovog sustava (engl. *L-systems*)[7] i tenzorskih polja (engl. *tensor fields*)[8]. Ovisno o odabranom pristupu, moguće je generirati različite oblike urbanih struktura, od pravilnih ortogonalnih ulica do organskih mreža.

Lindenmayerovi sustavi (L-sustavi) temelje se na skupu pravila kojim se početna struktura postupno širi kroz više iteracija. Takav se sustav može koristiti za generiranje grananja i razvoja cestovnih mreža. Tenzorska polja, s druge strane, definiraju smjerove kojima se određuju dominantni smjerovi prometnica i oblikovanje strukture grada.

U ovom radu odabran je pristup temeljen na diskretnoj mreži ćelija i definiranim pravilima generiranja. Takav pristup omogućuje jednostavno određivanje odnosa između susjednih ćelija i elemenata.



Slika 2.4. Generiranje grada na temelju tenzorskih polja[8]

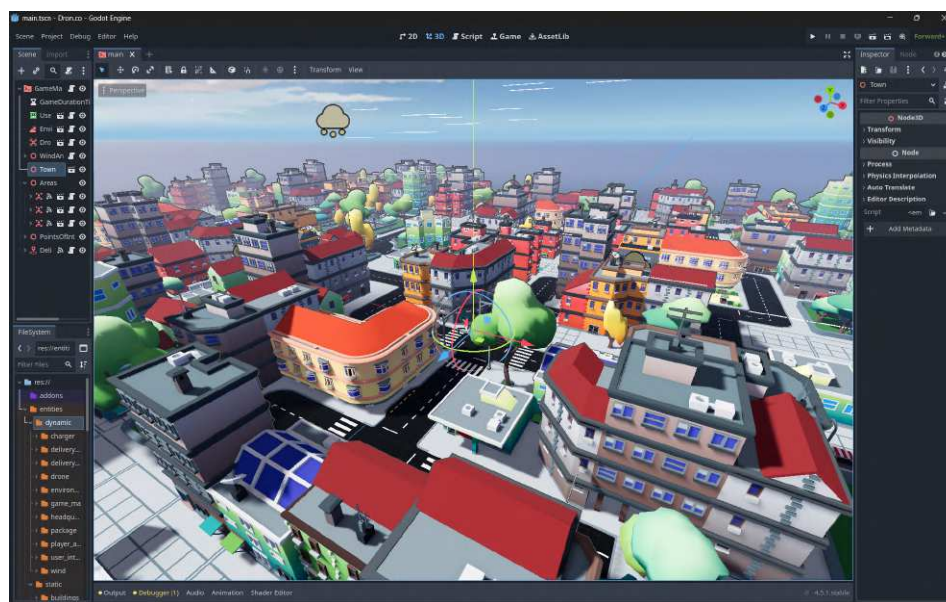


Slika 2.5. Usporedba generiranog grada koristeći Lindenmayerov sustav (gornje slike) s pravom kartom Manhattana[7] (donje)

3. Korištene tehnologije

3.1. Godot Engine

Za razvoj i implementaciju korišten je Godot (v4.5.1)[1], javno dostupan i besplatan *game engine* namijenjen razvoju 2D i 3D videoigara. Godot pruža alate potrebne za izradu i testiranje različitih komponenata videoigara, uključujući grafiku, fiziku, fiziku čestica, korisničko sučelje, pregled scena i pisanje skripti. Na slici 3.1. može se vidjeti korisničko sučelje 3D uređivača.



Slika 3.1. Prikaz zaslona uređivača u Godotu za igru Dron.co [2]

U ovom radu Godot je korišten kao razvojno okruženje za implementaciju proceduralnog sustava generiranja urbanog okruženja unutar postojeće videoigre Dron.co.

Organizacija sadržaja u Godotu temelji se na scenama i čvorovima (engl. *nodes*). Scena se sastoji od jednog ili više čvorova koji zajedno čine jednu cjelinu. Svaki čvor predstavlja pojedini objekt ili funkcionalni dio scene. Hijerarhija čvorova omogućuje

organizaciju složenijih scena te olakšava njihovo uređivanje i upravljanje.

Za raspoređivanje 3D elemenata grada korišten je Godotov alat GridMap, koji omogućuje postavljanje 3D elemenata na dvodimenzionalnu pravilno definiranu mrežu. Elementi koji se mogu postaviti u mrežu definirani su u resursu MeshLibrary, koji sadrži modele i podatke potrebne za njihovo korištenje u GridMapu.

U ovom su radu to unaprijed napravljene pločice dijelova gradskog okruženja. Pločice se mogu međusobno kombinirati na mreži GridMapa, pri čemu njihov položaj i orijentaciju određuje program.



Slika 3.2. Prikaz dijela grada [2] u GridMapu



Slika 3.3. Dio početnog grada videoigre Dron.co [2]

4. Implementacija rješenja

Problem je riješen implementacijom proceduralnog sustava za generiranje urbanog prostora koji se sastoji od nekoliko uzastopnih koraka. Ovisno o početnoj vrijednosti generatora slučajnih brojeva, sustav generira različite rasporede grada koristeći elemente definirane u *MeshLibrary* resursima. U prvom koraku, određuje se cestovna mreža, nakon čega se određuju gradske površine, parkovi i generiraju građevine. U završnom koraku na travnate se površine generira drveće, dok se na pločnike postavlja ulična rasvjeta. Svaki element se postavlja uz određenu vjerojatnost.

Za potrebe proceduralnog generiranja grad se reprezentira pomoću dvodimenzionalne mreže ćelija. Svaka ćelija mreže predstavlja određenu poziciju u prostoru grada te se identificira pomoću cjelobrojnih koordinata ($p = (x, z)$). Na taj se način složeni trodimenzionalni prostor svodi na prostor nad kojim je moguće jednostavno provoditi provjere susjedstva i postupke generiranja.

Na početku skripte, definirana su moguća stanja ćelija gradskog prostora. Za njihovo se označavanje koristi struktura *enum* koja ćelije dijeli u šest kategorija: praznu ćeliju izvan grada, praznu ćeliju grada, cestu, pločnik, travnatu ćeliju i građevine.

```
enum Cell { VOID , EMPTY , ROAD , SIDEWALK , GRASS , BUILDING }
```

Slika 4.1. Kategorije pločica

Funkcija `generate_city` predstavlja glavni tijek generiranja grada. Nakon inicijalizacije generatora slučajnih brojeva i mreže pozivaju se funkcije pojedinih faza generiranja definiranim redoslijedom.

```

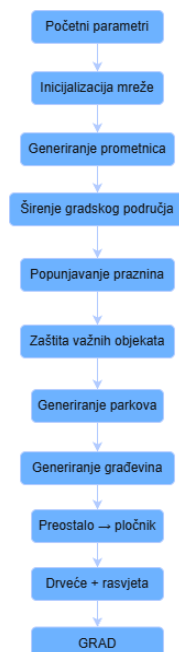
func generate_city(use_seed: int, keep_out: Array = []) ->
void:
#postavi nasumicnu pocetnu vrijednost
rng.seed = use_seed
extra_points = keep_out
setup_grid()

#ocisti prijasnji grad
floor_grid_map.clear()
building_grid_map.clear()
prop_grid_map.clear()

#postupak generiranja
grow_roads()
spread_city()
fill_gaps()
protect_cells()
make_parks()
plan_lots()
draw_floor()
draw_props()

```

Slika 4.2. Glavna funkcija programa



Slika 4.3. Dijagram procesa nastajanja grada

4.1. Pravila generiranja

Odabir pločica implementiran je bitovnim maskiranjem susjedstva (engl. *bitmask auto-tiling*), pristupom srodnim konceptu Wangovih pločica [9], kod kojeg se pločice smiju spajati samo ako se njihovi dodirni rubovi podudaraju.

Pri određivanju izgleda pločice, promatraju se četiri susjedne ćelije. Svakom od četiri smjera, sjeveru, istoku, jugu i zapadu, dodijeljena je vlastita vrijednost. Njihove vrijednosti redom iznose: 1, 2, 4 i 8, a njihovim zbrojem dobiva se ukupna vrijednost koja označava kombinaciju prometnica.

```
func road_sides(p: Vector2i) -> int:
    var sides := 0
    if is_road(p + Vector2i(0, -1)):
        sides |= N #sjever
    if is_road(p + Vector2i(1, 0)):
        sides |= E #istok
    if is_road(p + Vector2i(0, 1)):
        sides |= S #jug
    if is_road(p + Vector2i(-1, 0)):
        sides |= W #zapad
    return sides
```

Slika 4.4. Određivanje bitovne maske susjednih prometnica

Promatra se postoji li prometnica na bilo kojoj strani te se na temelju dobivene kombinacije određuje izgled i rotacija pločice. Na primjer, ukupna vrijednost 5 označava povezanost prema sjeveru i jugu, a vrijednost 15 označava povezanost u sva četiri smjera. S obzirom na to da se pločice mogu rotirati za 0° , 90° , 180° i 270° , ista se pločica može koristiti za nekoliko konfiguracija susjednih ćelija. Osnovna se konfiguracija pločice određuje za njezinu početnu, nerotiranu orijentaciju, dok se potrebna rotacija određuje na temelju dobivene bitovne maske.

Tablica 4.1. Opis vrsta pločica i njihovih svojstava

Bitovna maska	Povezane strane	Pločica	Opis elementa
1, 2, 4, 8	jedna	32_road2	slijepa ulica
5, 10	dvije, nasuprotne	01_road	ravna cesta
3, 6, 9, 12	dvije, pod pravim kutom	04_road	zavoj
7, 11, 13, 14	tri	05_road	T-raskrižje
15	četiri	06_road	raskrižje



Slika 4.5. Prikaz sustava bitmaske u modelu grada

Osim pravila vezanih za stvaranje cestovne mreže, tijekom generiranja potrebno je zaštititi određene ćelije na kojima se nalaze posebni objekti. Koristi se struktura podataka u kojoj se nalaze dinamički objekti scene, kao što su *headquarters*, *chargers*, i *delivery points*, čije su pozicije unaprijed definirane te se prosljeđuju generatoru prije početka generiranja grada.

Funkcija `_generate_city()` u klasi `GameMaster` dohvaća njihove globalne koordinate te ih prosljeđuje generatoru grada. Generator zatim pretvara dobivene svjetske koordinate u ćelije svoje mreže. Za svaku tako određenu ćeliju funkcija `protect_cells()` označava okolne ćelije kao zaštićene. Broj zaštićenih ćelija određen je parametrom `keep_clear`.

Važno je naglasiti da funkcija `_generate_city()` ne generira lokacije zaštićenih objekata, već samo prenosi njihove pozicije generatoru. Zaštićene se ćelije tijekom kasnijeg generiranja provjeravaju prije postavljanja građevina i parkova kako bi se spriječilo njihovo preklapanje s navedenim važnim objektima igre. Ovim se načinom omogućava siguran pristup točkama dostave paketa te sprječava postavljanje zgrada na početnu poziciju drona.

```
func protect_cells() -> void:
    for p in required:
        ...
        if not in_map(p):
            continue
        if grid[p.x][p.y] == Cell.VOID:
            grid[p.x][p.y] = Cell.EMPTY
        protected[p] = true
```

Slika 4.6. Mehanizam rezervacije prostora za dinamičke objekte

4.2. Generiranje prometnica i pločnika

Postupak generiranja cestovne mreže ima nekoliko uzastopnih koraka: ponavlja se pronalaženje mjesta unutar područja grada koje nema prometnicu u blizini, nakon čega se prema tom mjestu produžuje postojeća cesta, sve dok takvih mjesta više nema. Uvjet zaustavljanja time nije unaprijed zadan broj koraka, već se program zaustavlja nakon ostvarene pokrivenosti područja cestovnom mrežom.

Postupak se oslanja na tri glavne funkcije. Funkcija `find_gaps()` vraća sve ćelije unutar područja grada koje nemaju prometnicu unutar udaljenosti zadane parametrom `road_branching`. Uvjetom 4.1 prikazuje se provjera pokrivenosti ćelije cestovnom mrežom korištenjem Manhattanove udaljenosti. Ćelije čija je Manhattanova udaljenost od promatrane cestovne ćelije veća od vrijednosti $r = \text{road_branching}$ ne označuju se kao pokrivene.

$$|dx| + |dz| > r \quad (4.1)$$

Funkcija `best_start()` pronalazi ćeliju postojeće mreže iz koje je dopušteno grananje, odnosno iz koje je dopušteno započeti novu dionicu, pri čemu su uvjeti napisani u tablici 4.2. Pri određivanju početne točke za izgradnju nove prometnice također se koristi Manhattanova udaljenost između postojeće cestovne ćelije i ciljne ćelije. Za dvije ćelije $p = (x_1, z_1)$ i $q = (x_2, z_2)$, Manhattanova udaljenost definirana je kao jednadžba 4.2

$$d(p, q) = |x_1 - x_2| + |z_1 - z_2|. \quad (4.2)$$

Ovakav način mjerenja udaljenosti odgovara mreži koja se koristi za reprezentaciju grada, budući da se kretanje između susjednih ćelija odvija u četiri osnovna smjera. U funkciji `best_start()` udaljenost se koristi za odabir postojeće cestovne ćelije koja je najbliža ciljnom području. Time se nova dionica prometnice usmjerava prema dijelu grada koji još nije dovoljno pokriven prometnom mrežom, umjesto da se početna točka odabire potpuno slučajno.

Manhattanova udaljenost pritom ne predstavlja stvarnu geometrijsku udaljenost između točaka u prostoru, već mjeru udaljenosti prilagođenu mreži ćelija. Konačna putanja prometnice nije nužno jednaka najkraćoj Manhattanovoj putanji jer na njezin smjer dodatno utječe i parametar `bend_chance`.

Funkcija `road_towards()` od odabrane početne ćelije započinje hod prema cilju, odnosno prema ćeliji unutar grada koja je dovoljno udaljena od ceste. U većini se koraka odabire ćelija izravno prema cilju, osim u slučajevima određenim zadanim parametrom `bend_chance` koji cestu okreće postrance, odnosno izabire zavoj. S obzirom na to da je mjesto na kojem se nova dionica odvaja od postojeće mreže po definiciji raskrižje, a nikada zavoj, bez parametra `bend_chance` bi sve prometnice vodile izravno prema svojim

ciljevima.

Tablica 4.2. Pravila generiranja

Pravilo	Sprječava	Rezultat
Nova cesta mora započeti na staroj	stvaranje otoka	povezanost cestovne mreže
Raskrižja ne smiju biti spojena	zapetljana križanja	realističan promet
Nijedan kvadrat 2x2 ne smije biti samo cesta	usporedne ulice koje srastu	odvojene ceste

Tijekom stvaranja cestovne mreže prate se prijašnje spomenuta pravila koja se odnose na način spajanja. Prvo pravilo nije potrebno naknadno provjeravati jer je ugrađeno u sam način generiranja prometnica. Svaka nova dionica započinje iz ćelije koja je već dio postojeće mreže, čime se osigurava povezanost s prethodno generiranim dijelom mreže. Nakon privremenog postavljanja nove dionice provjerava se zadovoljava li ona definirana pravila. Ako se utvrdi kršenje nekog pravila, funkcija `keep_road()` poziva pomoćnu funkciju `undo_road()` čime se privremeno postavljene ćelije vraćaju u početno stanje. Na taj se način nepravilne konfiguracije uklanjaju odmah tijekom generiranja, bez potrebe za fazom naknadnog popravljivanja cestovne mreže.

```

func keep_road(cells: Array) -> bool:
    ...

    for p in cells:
        #provjera susjedstva predložene dionice ceste
        for dx in range(-1, 2):
            for dz in range(-1, 2):
                p = p + Vector2i(dx, dz)
                if not is_road(q):
                    continue

                #sprječavanje spajanja susjednih raskrizja
                if is_junction(q) and next_to_junction(q):
                    return undo_road(cells)

                #sprječavanje stvaranja bloka 2x2 od
                prometnica
                if (is_road(q + Vector2i(1, 0)) and
                    is_road(q + Vector2i(0, 1))
                    and is_road(q + Vector2i(1, 1))):
                    return undo_road(cells)

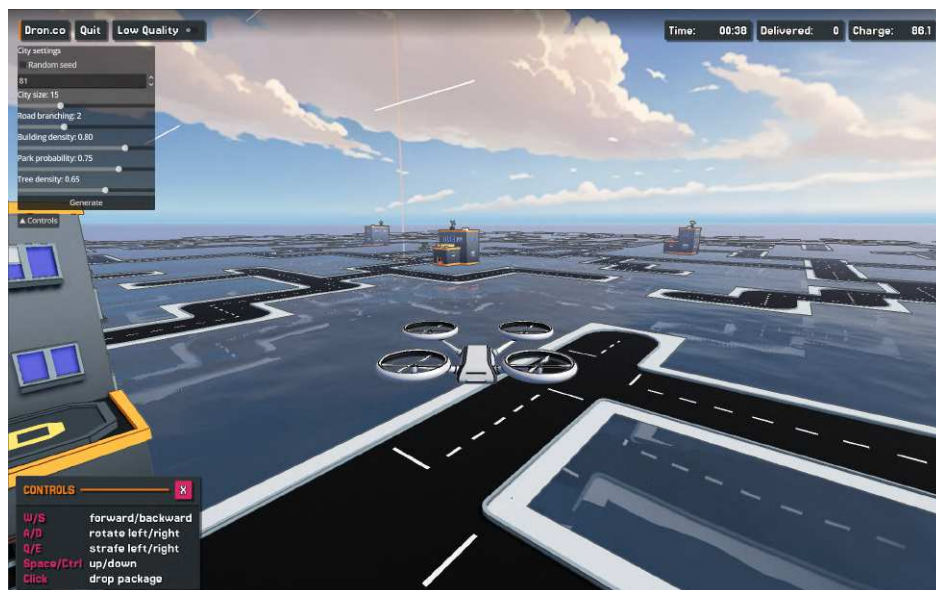
            ...

func undo_road(cells: Array) -> bool:
    for p in cells:
        grid[p.x][p.y] = Cell.VOID
    return false

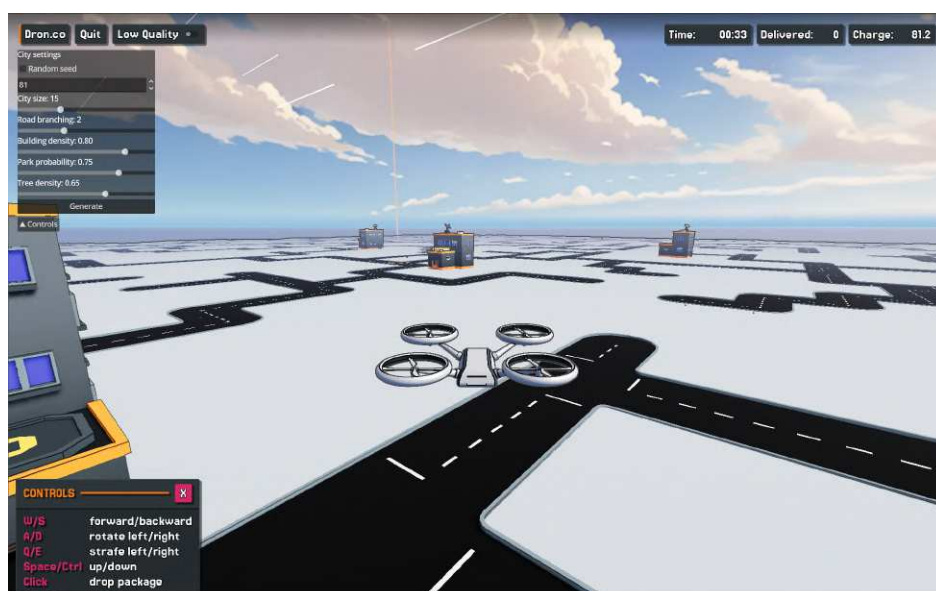
```

Slika 4.7. Provjera pravila prometne mreže i uklanjanje neispravne dionice

Za razliku od cestovnih mreža, pločnik se ne generira zasebnim postupkom. Funkcija `spread_city()` unutar područja grada označava svaku ćeliju koja se nalazi unutar zadane udaljenosti od neke prometnice, čime obris grada proizlazi iz same cestovne mreže umjesto da se određuje neovisno o njoj. Takav pristup omogućuje dvije bitne karakteristike generiranog područja. Grad dobiva nepravilan obris bez implementiranja dodatnog pravila, a s obzirom na to da je svaka ćelija grada unutar zadane udaljenosti od prometnice, grad je povezan.



Slika 4.8. Izgled grada nakon generiranja cesta



Slika 4.9. Izgled grada nakon generiranja pločnika

4.3. Generiranje parkova

Pojedini oblici parkova definirani su dvodimenzionalnim matricama u kojima svaki element predstavlja jednu travnatu pločicu i njezinu rotaciju. Definirani poredak pločica mora biti ostvaren kako bi sve pločice bile pravilno orijentirane i pratile odgovarajući redoslijed. Na taj se način mogu unaprijed definirati različiti oblici parkova koji se zatim postavljaju kao cjelina. Ovakav način reprezentacije omogućuje da se oblik parka definira podacima, a ne zasebnim algoritmom za svaki oblik.

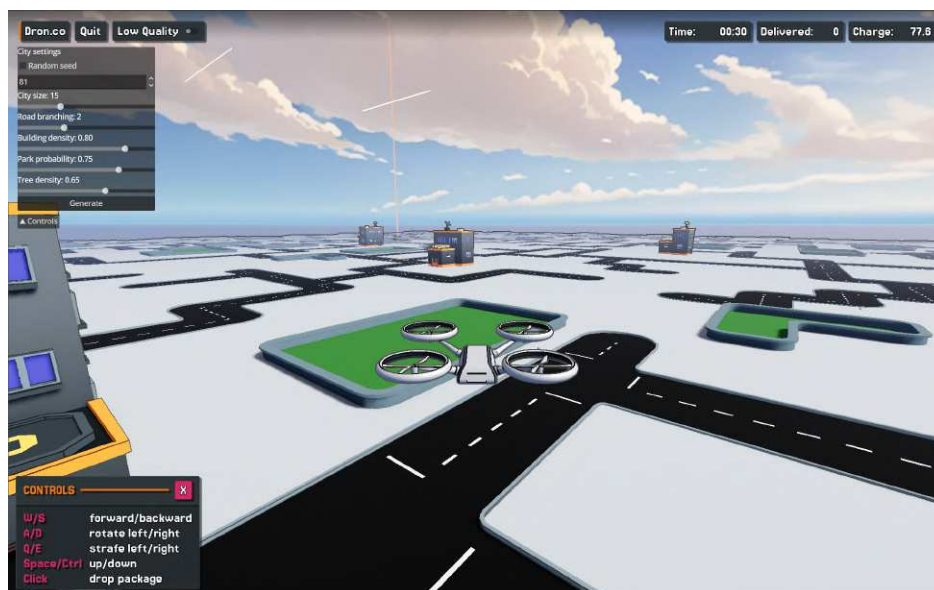
```
const PARKS := {
  "small": [
    [{"n": "10_grass", "r": 270}, {"n": "10_grass", "r": 180}],
    [{"n": "10_grass", "r": 0}, {"n": "10_grass", "r": 90}],
    ...
  ]
}
```

Slika 4.10. Primjer definicije parka u kodu

Uvođenjem preostalih postojećih travnatih pločica u sustav mogu se ostvariti mnogi oblici i veličine parkova. Međutim, u sklopu ovog rada određena su četiri oblika parkova. Generator pokušava postaviti veće oblike prije manjih kako bi se raspoloživi prostor što bolje iskoristio. Prije postavljanja provjerava se mogu li sve ćelije odabranog oblika biti zauzete bez preklapanja s postojećim elementima.

```
func make_parks() -> void:
  ...
  for i in tries:
    ...
    for shape_name in ["big", "medium", "right_corner", "small"]:
      var shape: Array = PARKS[shape_name]
      if park_fits(shape):
        stamp_park(shape)
        break
```

Slika 4.11. Generiranje parkova



Slika 4.12. Izgled grada nakon postavljanja parkova

4.4. Generiranje građevina

Nakon određivanja cestovne mreže i gradskih površina preostale ćelije grada mogu se iskoristiti za smještaj građevina. Položaj građevine ne određuje se unaprijed definiranim rasporedom, već se za svaku odgovarajuću ćeliju provjeravaju uvjeti mogućeg postavljanja zgrade.

Prije postavljanja građevine provjerava se pripada li promatrana ćelija području grada te nalazi li se među zaštićenim ćelijama. U slučaju da se ćelija nalazi među zaštićenim ćelijama, ona će se preskočiti kako bi se osigurala dostupnost posebnih objekata važnih za igru. Dodatno se koristi parametar `building_density` koji određuje vjerojatnost postavljanja građevine na odabranu ćeliju. Veća vrijednost parametra rezultira gusto naseljenim gradom, dok manja vrijednost ostavlja veći broj praznih površina.

Za imenovanje i organizaciju dostupnih modela građevina koriste se konstante. Nazivi svih modela građevina nalaze se u listi elemenata iz koje se tijekom generiranja nasumično biraju građevine koje će se postaviti. Na taj se način omogućava raznolik izgled grada te se izbjegava ponavljanje istog izgleda građevina.

Postavljanje građevina odvojeno je od generiranja cestovne mreže, ali i dalje ovisi o njoj. Za određivanje orijentacije postavljane građevine koristi se bitovna maska susjednih prometnica. Za promatranu ćeliju ispituje se nalazi li se prometnica s njene sjeverne,

istočne, južne ili zapadne strane. Dobivena kombinacija koristi se kao orijentacija, odnosno određuje u kojem će smjeru biti okrenuta referentna strana građevine. Ovom se provjerom smanjuje broj slučajeva u kojima je zgrada nasumično okrenuta prema unutrašnjosti naseljenog bloka.

Na ovaj se način građevine generiraju na temelju određene gradske cestovne mreže, a ne neovisno o njoj. Uz generiranu cestovnu mrežu, uzimajući u obzir orijentaciju građevina ovisno o prisutnim prometnicama ostvaruje se smislenost i povezanost generiranih elemenata grada. Konačni izgled građevina ipak ovisi o slučajnom odabiru modela i parametru `building_density`, zbog čega se za isti raspored prometnica mogu dobiti različiti rezultati.

```
func plan_lots() -> void:
...
for x in size:
    for z in size:
        var p := Vector2i(x, z)
        #provjera zastice i vjerojatnosti
        if grid[x][z] != Cell.EMPTY or protected.has(p):
            continue
        if rng.randf() > building_density:
            continue
        ...
        #odabir modela građevine
        var tile_name: String = BUILDING_TILES[rng.
            randi() % BUILDING_TILES.size()]
        ...
        #bitmaska
        var sides := road_sides(p)
        ...
        grid[p.x][p.y] = Cell.BUILDING
```

Slika 4.13. Odabir položaja građevina



Slika 4.14. Izgled grada nakon generiranja građevina

4.5. Postavljanje elemenata

Prethodno opisani sustav i funkcije koriste se za određivanje pozicija pojedinih elemenata u ćelijama GridMapa. Tek nakon što su određene sve pozicije pločica, slijedi njihovo postavljanje u GridMap. Za postavljanje elemenata koristi se funkcija `draw_floor()` koja prolazi kroz sve ćelije te postavlja pločicu iz MeshLibraryja s odgovarajućim imenom i rotacijom. Na ovaj se način postavljaju elementi cestovne mreže, parkova i pločnika, dok se građevine postavljaju u prethodnoj fazi generiranja. Preostale ćelije označene kao EMPTY popunjavaju se pločicama pločnika.

```

func draw_floor() -> void:
    ...
    match grid[x][z]:
        Cell.VOID:
            pass
        Cell.ROAD:
            var tile := road_tile(road_sides(p))
            put_floor(p, tile["n"], -float(tile["r"]
            ))
        Cell.GRASS:
            var g: Dictionary = park_tiles.get(p,
            {"n": GRASS_TILE, "r": 0})
            put_floor(p, g["n"], float(g["r"]))
        _:
            put_floor(p, SIDEWALK_TILE, 0.0)

```

Slika 4.15. Postavljanje elemenata u GridMap

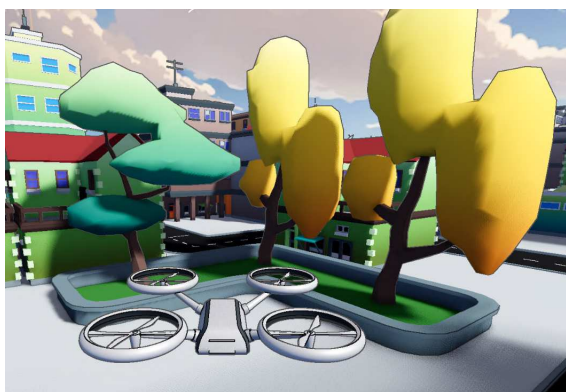
4.6. Drveće i ulična rasvjeta

Posljednji korak sustava proceduralnog generiranja jest postavljanje detalja. Za detalje je odabrano dvanaest vrsta drveća drugačijih boja i veličina te tri vrste uličnih lampi. Detalji drveća postavljaju se isključivo na pločice koje čine parkove i travnate površine, dok se ulična rasvjeta postavlja samo na pločice pločnika. Prilikom odabira vrste drveća i ulične rasvjete, program nasumično odabire oblik i postavlja ga na odabrano mjesto. Tim se pristupom jednostavno omogućuje raznolikost detalja uz određeni stupanj vjerojatnosti `tree_density` i `lamp_chance`.

4.7. Izvorni kod programa

Rezultati i izvorni kod programa dostupni su na GitHub repozitoriju:

<https://github.com/mateamaracic/Dron.co>



Slika 4.16. Prikaz generiranog drveća i ulične rasvjete

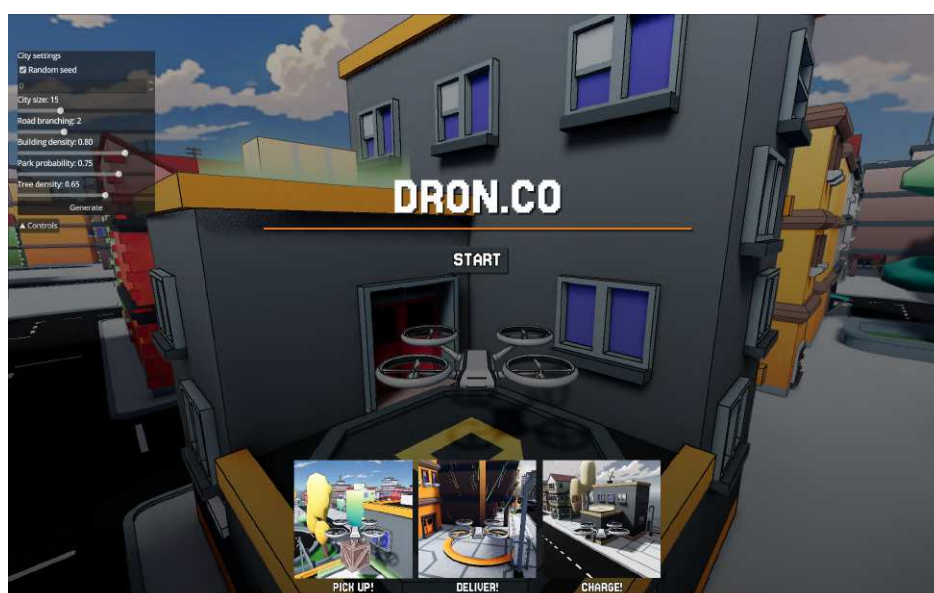


Slika 4.17. Završni zgled grada nakon generiranja

5. Rezultati, analiza i usporedba

5.1. Interaktivni parametri

Korisničko sučelje sastoji se od jednostavnog kvadratnog elementa s implementiranim klizačima i gumbima pomoću kojih se mijenjaju eksperimentalni parametri. Kontrolni element nalazi se na lijevoj strani slike 3.3.



Slika 5.1. Korisničko sučelje za upravljanje parametrima

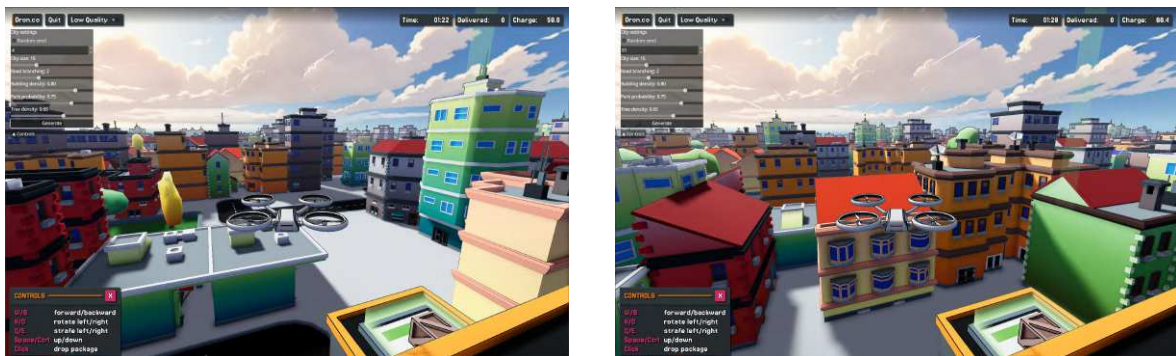
5.1.1. Polazna vrijednost generatora

Za određenu početnu vrijednost generatora slučajnih brojeva (engl. *seed*) generira se određeni raspored grada. Ista početna vrijednost rezultira istim rasporedom, dok promjena početne vrijednosti omogućuje generiranje drugačijeg rasporeda. Ručno mijenjanje početne vrijednosti ne omogućuje kontrolu korisnika nad rasporedom grada, već samo pruža novu verziju razine.

Ovaj sustav čini pojedini raspored grada reproducibilnim, odnosno za istu početnu

vrijednost ponovno će se generirati isti raspored grada. Reproducibilnost grada bitno je svojstvo prilikom testiranja i mijenjanja algoritma generiranja te omogućuje promatranje promjena parametara na istom generiranom rasporedu.

Na slici 5.2. prikazana su dva grada generirana različitim početnim vrijednostima. Iako su generirana s istim postavkama parametara, vidljivo imaju drugačiji raspored cestovnih mreža, parkova i građevina. Time se pokazuje da slučajni odabir omogućuje varijabilnost rasporeda grada bez mijenjanja osnovnih pravila generiranja.



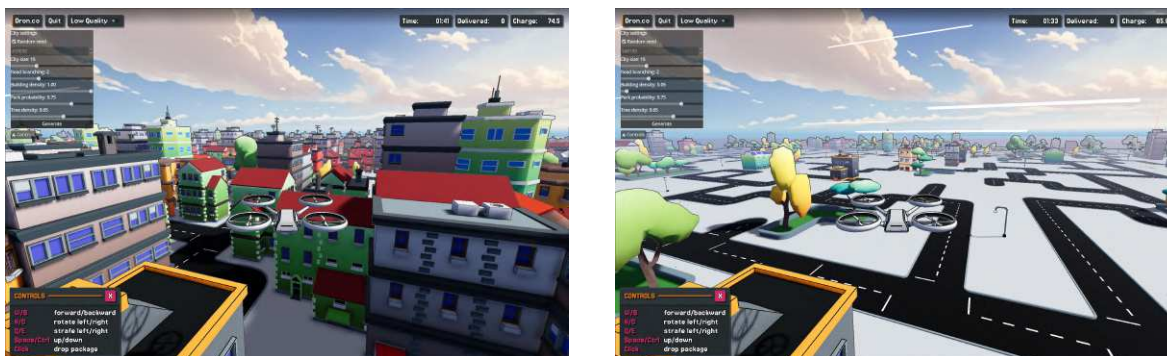
Slika 5.2. Usporedba generiranog grada s različitim početnim vrijednostima

5.1.2. Utjecaj gustoće građevina

Promjena parametra `building_density` direktno utječe na količinu naseljenog prostora, ali njezin utjecaj ne odnosi se samo na broj postavljenih građevina. Pri višim vrijednostima ovog parametra, većina gradskih površina prekrivena je građevinama te čini grad gušće naseljenim. Istovremeno se smanjuje količina prostora između građevina što otežava raspoznavanje dijelova grada te čini kretanje drona kroz grad težim.

S druge strane, pri manjim vrijednostima `building_density` povećava se količina praznih prostora te grad izgleda pusto, gotovo nedovršeno, a sastoji se od praznih pločnika, cestovne mreže i parkova.

Na slici 5.3. vidljivo je da mijenjanje parametra `building_density` uvelike utječe na cjelokupni izgled i dojam razine videoigre. Ovaj parametar pokazuje da je moguće drastično promijeniti gustoću naseljenih dijelova grada, bez potrebe za mijenjanjem algoritma i logike programa.



Slika 5.3. Maksimalna i minimalna vrijednost parametra *build_density*

5.1.3. Utjecaj gustoće cestovne mreže

Promjenom parametra *road_branching* korisnik izravno utječe na gustoću cestovne mreže, ali i na sam oblik grada, budući da grad čini svaka ćelija udaljena za *road_branching* od bilo koje ceste. Povećanjem njegove vrijednosti povećava se područje koje pojedina cesta može pokriti, zbog čega je za pokrivanje istog prostora potrebno manje prometnica. Rezultat je rjeđa cestovna mreža s većim površinama između prometnica.

Na slici 5.4. vidljiva je razlika između veće i manje vrijednosti parametra. Grad generiran s manjom vrijednošću parametra generira gušće prometnice i manje slobodnih međuprostora, dok veća vrijednost omogućuje da pojedinačna prometnica pokrije veće područje, zbog čega se generira rjeđa cestovna mreža. Nadalje, u slučaju velike gustoće ulica, parametar utječe i na mogućnost generiranja velikih parkova. Veliki park zauzima površinu od dvanaest ćelija pa pri gustoći uličnoj mreži između prometnica jednostavno nema dovoljno prostora za njegovo postavljanje. Također, smanjenje ovog parametra negativno utječe na brzinu izvođenja programa.

Ovaj primjer pokazuje da parametri generatora nisu međusobno neovisni. Iako parametar *road_branching* izravno određuje svojstva gustoće cestovne mreže, njegova promjena utječe i na mogućnost generiranja parkova i građevina. Zbog toga optimalnu vrijednost parametra nije moguće odrediti promatranjem samo jednog elementa grada, već je potrebno uzeti u obzir konačni izgled cijelog generiranog prostora.

5.1.4. Utjecaj ostalih parametara

Uz navedene parametre postoji još nekoliko parametara koje je moguće promijeniti. Veličina mape nije početno određena već se određuje u programu, ovisno o zaštićenim će-



Slika 5.4. Maksimalna i minimalna vrijednost parametra *road_branching*

lijama koje se moraju nalaziti unutar područja grada. Računa se tako da grad mora obuhvatiti sve zaštićene točke uz dodatan *padding* koji je zadan parametrom *city_size*.

Parametar *park_probability* utječe na vjerojatnost odabira travnate pločice, odnosno postavljanje parkova. Usko povezan s njim jest parametar *tree_density*, vjerojatnost da će se na određenoj travnatoj površini generirati stablo. S obzirom na to da se drveće može generirati samo na površine parkova, parametar *tree_density* utjecat će samo na one dijelove grada koji su već određeni kao površine parkova.

Tablica 5.1. Završna usporedba utjecaja promjene parametra

Parametar	Povećanje vrijednosti	Smanjenje vrijednosti
<i>building_density</i>	veća gustoća građevina	više slobodnih površina
<i>park_probability</i>	više parkova	više pločnika
<i>tree_density</i>	više drveća	manje drveća
<i>road_branching</i>	rjeđa cestovna mreža	gušće ulice

Pri prvom pokretanju videoigre, parametri su postavljeni na osobno izabranu "optimalnu" vrijednost koja ne određuje stvarnu optimalnu vrijednost već je izabrana višebrojnim testiranjem i mijenjanjem parametara. Ispitivanje je ukazalo da vrijednosti parametara prikazane na slici 5.5. rezultiraju verzijom grada koja pruža dovoljnu naseljenost i gustoću cesta za postavljanje parkova i čini razinu dovoljno zahtjevnom kako bi bila prikladna za potrebe igre.



Slika 5.5. Izabrana početna vrijednost interaktivnih parametara

5.1.5. Kritički osvrt

Provedena ispitivanja pokazuju da implementirani parametri omogućuju korisniku da promijeni raspored grada na nekoliko drugačijih načina. Korisnik ne mora ručno određivati položaj pojedinih elemenata, već osnovni izgled grada može mijenjati promjenom nekoliko parametara. Parametri na svoj način utječu na konačni raspored grada, ali pokazano je da njihov utjecaj nije uvijek neovisan. Reproducibilnost pomoću početne vrijednosti dodatno omogućuje ponavljanje i uspoređivanje rezultata.

S druge strane, ograničen broj pravila smanjuje raznolikost mogućih rasporeda grada. Cestovna mreža ograničena je na četiri osnovna smjera, prometnice nemaju hijerarhiju, a odabir građevina je slučajan. Zbog toga generirani grad može biti funkcionalan za potrebe razine videoigre, ali ne postiže razinu prostorne i vizualne raznolikosti karakterističnu za stvarne gradove ili složenije proceduralne sustave. Sustav ubrzava i automatizira ponavljajući dio izrade razine, dok bi se daljnja složenost mogla ostvariti proširenjem skupa pravila i dostupnih pločica.

5.2. Računalna zahtjevnost

Proceduralna generacija grada provodila se na računalu s procesorom AMD Ryzen 5 8645HS w/ Radeon 760M Graphics (4.30 GHz), grafičkom karticom NVIDIA GeForce RTX 3050 6GB Laptop GPU (6 GB) AMD Radeon(TM) 760M (2 GB) i 16 GB radne memorije.

Za procjenu zahtjevnosti generiranja odrađena su mjerenja ovisno o parametru `city_size` koji utječe na veličinu grada. Ovisno o promjeni navedenog parametra provedena su tri mjerenja po promijeni veličine te je uzeta srednja vrijednost tih mjerenja kako bi se osigurala konzistencija. U tablici 5.2. prikazani su rezultati mjerenja brzine izvođenja programa. Na temelju tih rezultata moguće je zaključiti da se porastom veličine grada i broja ćelija, znatno povećava vrijeme izvođenja. Razlog tome jest veći broj ćelija koji se generira te proces određivanja rasporeda elemenata postaje složeniji.

Za parametar `city_size = 15` generiranje je prema provedenim mjerenjima još bilo prihvatljivo za izvođenje u stvarnom vremenu. Daljnjim povećanjem parametra, generiranje postaje zahtjevnije i utječe na kvalitetu igre.

Tablica 5.2. Mjerenja trajanja generiranja grada

<code>city_size</code>	Srednja vrijednost
10	4.08 sekundi
15	6.36 sekundi
25	11.86 sekundi
35	18.49 sekundi
40	25.83 sekundi

5.3. Usporedba postojećih rješenja

U potpoglavlju 2.3. spomenute su različite tehnike proceduralnog generiranja. Tehnika koja je korištena u ovom radu na nekoliko se načina razlikuje od spomenutih.

Prostorna reprezentacija: L-sustavi i pristupi temeljeni na tenzorskim poljima omogućuju generiranje složenijih ulica poput dijagonalnih prometnica. U ovom radu koristi

se dvodimenzionalna mreža ćelija, zbog čega su prometnice ograničene na četiri osnovna smjera. S druge strane, korištenje mreže ćelija odgovara alatu GridMap te čini pretvorbu odabranog rasporeda grada u gotove pločice jednostavnom.

Hijerarhija prometnica: Parish i Müller [7] razlikuju glavne i sporedne ulice te njihov rast usmjeravaju pomoću karte gustoće naseljenosti. U ovom radu sve prometnice imaju jednaku širinu i jednaku ulogu, čime se smanjuje složenost implementacije, ali istodobno gubi mogućnost modeliranja hijerarhije prometne mreže. Grad je cijelim svojim područjem jednako naseljen te trenutno nema potrebe za uvođenjem gustoće naseljenosti kao faktor grananja cestovne mreže.

Kontrola dizajna: Guoning Chen i njegovi suradnici u svom radu *Interactive Procedural Street Modeling*[8] omogućuju dizajneru da uređuje i utječe na izgled generiranog grada izravnim dizajniranjem tenzorskog polja. Dizajniranjem tenzorskog polja korisnik oblikuje uzorak ulica. U ovom se radu dizajn grada svodi na nekoliko parametara i generiranu cestovnu mrežu što čini ovaj pristup jednostavnim za korištenje, ali pruža manju slobodu i kontrolu dizajna.

Primjena: Opisani se pristupi često primjenjuju u svrhu modeliranja složenih urbanih područja te zahtijevaju mogućnost kontroliranja dizajna. Proceduralno generiranje u ovom se radu koristi prvenstveno kao automatizacija stvaranja razina za videoigru te kontrola i personalizacija nisu nužne. Igrivost i prohodnost razine važniji su od vizualnog podudaranja sa stvarnim gradovima.

5.4. Budući razvoj

Tijekom izrade rada osviještena su ograničenja generatora koja bi se u daljnjem razvoju mogla ispraviti. Potencijal budućeg razvoja ovog projekta uključuje veliki broj nadogradnji i optimizacija, a neke od njih su:

1. Proširenje skupa gradskih pločica: Za sada nedostaju pješački prijelazi, autobusne stanice, parking, kružni tok, složeniji park, ograda oko ruba grada i slično. Njihovo uključivanje ostvarilo bi još realističniju sliku grada i bolji doživljaj igrača. S obzirom

na to da postojeći skup pločica već sadrži potrebne modele, za njih bi se mogla definirati dodatna pravila, slična pravilima za pločice koje su korištene prilikom izrade ovog rada.

2. Prostorno grupiranje građevina: Izmjenom algoritama za odabir građevina može se ostvariti grupiranje zgrada u monokromne gradske četvrti. Razlog zašto je u ovom radu implementiran drugačiji pristup jest želja za ostvarivanjem vizualno raznolikog grada.

3. Proširenje generiranja na teren i dinamičke elemente: Implementacija terena koristeći nove pločice dizajnirane po oblicima prirode. Grad okružen planinama, zemljom, pijeskom u drugačijim početnim vrijednostima. Također, generiranje populacije te automobila činilo bi novu prepreku u igri koja istovremeno grad čini realističnijim.

4. Generiranje i provjera zračnih putanja: Provjera postoji li putanja na određenoj visini od tla omogućila bi lakše kretanje drona po zraku, ali bi igru mogla potencijalno olakšati. Generatori inače provjeravaju put na tlu što čini ovaj razvoj još zanimljivijim i vrijednim istraživanja.

5. Naprednija kontrola dizajna: Uvođenjem matematičkih metoda u proces proceduralne generacije moguće je ostvariti veću slobodu kod dizajniranja razine. Implementacijom L-sustava ili sličnih metoda korisniku se pruža više mogućnosti personalizacije.

6. Ubrzanje i optimizacija: Kroz daljnji razvoj projekta moguće je ostvariti veću brzinu generiranja razine kako korisnik ne bi imao loša iskustva. Trenutačno je algoritam dovoljno brz da ne utječe na iskustvo igre.

6. Zaključak

U ovom radu prikazan je sustav za proceduralno generiranje urbanog okruženja u programskom alatu Godot, razvijen u sklopu videoigre Dron.co. Sustav generira cestovnu mrežu, pločnike, građevine, parkove te detalje poput drveća i ulične rasvjete. Raspored elemenata određuje se skupom pravila umjesto ručnim postavljanjem.

Za odabir pločica korištene su bitovne maske susjedstva, kojima se šesnaest mogućih rasporeda susjednih prometnica svodi na šest modela. Ostali se modeli mogu izraziti korištenjem početnih šest modela uz određenu rotaciju. Ista je tehnika primijenjena i pri određivanju orijentacije građevina, čime je postupak bitovnih maski iskorišten kao rješenje za dva različita problema.

Sustav omogućuje interaktivno upravljanje parametrima generiranja, čime je omogućena eksperimentacija nad rasporedom i personalizacija razine. Analiza parametara pokazala je da se izgled grada može značajno mijenjati promjenom nekoliko ključnih vrijednosti, pri čemu parametri nisu međusobno potpuno neovisni. Time se potvrđuje da implementirani sustav omogućuje relativno jednostavno prilagođavanje generirane razine različitim potrebama.

Implementirano rješenje, uz svoje prednosti, ima i nekoliko ograničenja. Prikaz mrežom dopušta isključivo prave kutove, te zbog toga nema dijagonalnih ulica. Cestovna mreža nema hijerarhiju te su sve prometnice jednake važnosti. Teren je ravan, a građevine se odabiru nasumično.

Unatoč navedenim ograničenjima, sustav ispunjava postavljeni cilj. Raspored razine čija je ručna izrada trajala nekoliko dana generira se u nekoliko sekundi, čime je znatno ubrzan razvoj novih razina igre.

Literatura

- [1] Godot Engine Contributors, “Godot engine – free and open source 2d and 3d game engine”, 2024., online; pristupljeno: lipanj 2026. [Mrežno]. Adresa: <https://godotengine.org>
- [2] J. Ramljak i suradnici, “Dron.co – objavljeni resursi projekta (inačica v1.0)”, <https://github.com/j-ramljak/Dron.co/releases/tag/v1.0>, 2026., [Online; pristupljeno: lipanj 2026.].
- [3] Future US, “Prikaz računalno generirane grafike”, <https://cdn.mos.cms.futurecdn.net/BehFF7GvN9w2geYVrtSmQH-1200-80.jpg.webp>, 2026., [Online; pristupljeno: lipanj 2026.].
- [4] Reddit korisnik, “Endless procedural terrain generation - what do you think?” https://www.reddit.com/r/Unity3D/comments/1822hzk/endless_procedural_terrain_generation_what_do_you/, 2023., [Online; pristupljeno: lipanj 2026.].
- [5] A. Scholz, “Procedural generation of terrains with erosion”, Bachelor’s thesis, TU Wien, Faculty of Informatics, Vienna, Austria, 2017.
- [6] E. Nickerson. (2018., 1) Using grammars to define procedural generation. Online; pristupljeno: srpanj 2026. [Mrežno]. Adresa: <https://bekwnn.github.io/blog/2018/01/04/using-grammars-to-define-procedural-generation.html>
- [7] Y. I. H. Parish i P. Müller, “Procedural modeling of cities”, u *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. New York, NY, USA: Association for Computing Machinery, 2001., str. 301–308. <https://doi.org/10.1145/383259.383292>

- [8] G. Chen, G. Esch, P. Wonka, P. Müller, i E. Zhang, “Interactive procedural street modeling”, *ACM Transactions on Graphics (TOG)*, sv. 27, br. 3, str. 103:1–103:10, 2008. <https://doi.org/10.1145/1360612.1360102>
- [9] H. Wang, “Proving theorems by pattern recognition—ii”, *Bell System Technical Journal*, sv. 40, br. 1, str. 1–41, 1961.

Izjava o korištenju umjetne inteligencije

Osvrt na korištenje umjetne inteligencije: Tijekom izrade rada koristila sam Claude Sonnet 5 kao pomoć pri provjeri ideja vezanih uz rad te kao pomoć pri implementaciji određenih dijelova sustava. Generirane prijedloge koristila sam kao pomoćno sredstvo, pri čemu sam rješenja provjeravala i prilagođavala vlastitim potrebama. Pritom se događalo da model predloži rješenja koja nisu odgovarala strukturi rada ili pogrešno protumači način na koji je određeni dio sustava implementiran. Zbog toga sam samostalno odlučivala o primjeni predloženih rješenja.

Izjava o korištenju umjetne inteligencije: Potvrđujem da sam tijekom izrade ovog završnog rada koristila umjetnu inteligenciju u skladu s Politikom primjerenog korištenja umjetne inteligencije na Fakultetu elektrotehnike i računarstva, te da umjetna inteligencija nije zamijenila moje kritičko razmišljanje niti moj doprinos.

Privitak A: Izvorni kod

Projekt Dron.co javno je dostupan na GitHub repozitoriju: <https://github.com/j-ramljak/Dron.co>

Izvorni kod rada dostupan je na GitHub repozitoriju: <https://github.com/mateamaracic/Dron.co>