

# NAČINI KREIRANJA RAZINA U RAČUNALNIM IGRAMA

---

**Crnogorac, Nikola**

**Undergraduate thesis / Završni rad**

**2023**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **Algebra University College / Visoko učilište Algebra**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:225:814262>

*Rights / Prava:* [In copyright](#)/[Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2024-12-22**



*Repository / Repozitorij:*

[Algebra University - Repository of Algebra University](#)



**VISOKO UČILIŠTE ALGEBRA**

ZAVRŠNI RAD

**NAČINI KREIRANJA RAZINA U  
RAČUNALNIM IGRAMA**

Nikola Crnogorac

Zagreb, ožujak 2023.

*„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.*

# Predgovor

Ovim putem želim zahvaliti svojoj obitelji koja me podržavala tokom pisanja ovog rada. Također se želim zahvaliti svojem mentoru Ivanu Porkolabu koji me vodio kroz ovaj rad i pomogao da bude najviše moguće kvalitete.

## Sažetak

Cilj ovog diplomskog rada je prikazati načine kreiranja razina u računalnim igrama kroz praktičan primjer dvodimenzionalne igre izrađene pomoću Unity alata za kreiranje računalnih igara. U radu se obrađuju tri načina kreiranja razina: dinamičko, nasumično i proceduralno generiranje. Kako je igračko iskustvo ključna komponenta suvremenih razvojnih timova, sve više se vremena i resursa ulaže se u analizu igračkih iskustava, što omogućuje developerskim timovima i samostalnim developerima kreiranje računalnih igrica s najboljim praksama. Ovaj rad pruža pregled najboljih načina i praksi generiranja razina, koji će pomoći developerima i developerskim timovima u stvaranju što kvalitetnijih igara.

# Summary

The aim of this thesis is to showcase the ways of creating levels in computer games through a practical example of a two-dimensional game made using the Unity game development tool. The thesis covers three ways of creating levels: dynamic, random, and procedural level generation. As the gaming experience is a crucial component of modern development teams, more and more time and resources are being invested in analyzing player experiences, allowing development teams and independent developers to create computer games with the best practices. This paper provides an overview of the best ways and practices for level generation, which will help developers and development teams to create higher quality games.

# Sadržaj

1. Uvod .....	1
2. Dokument specifikacije .....	2
2.1. Sažetak .....	2
2.2. Priča .....	3
2.3. Mehanika računalne igre .....	3
3. Način generiranja razina .....	5
3.1. Dinamičko generiranje .....	6
3.2. Nasumično generiranje .....	7
3.3. Proceduralno generiranje .....	8
4. Izrada računalne igre .....	9
4.1. Izrada bazne računalne igre .....	10
4.1.1. Osnovni koncept i ideja .....	10
4.1.2. Osnovni elementi Unity platforme .....	11
4.1.3. Objekti .....	14
4.1.4. Grafika .....	14
4.1.5. Glavni lik .....	15
4.1.6. Neprijatelji .....	19
4.1.7. Kamera .....	22
4.1.8. Oružja .....	23
4.1.9. Interaktivni objekti .....	25
4.1.10. Mapa .....	27
4.1.11. Unity analytics .....	28
4.2. Implementacija različitih načina generiranja .....	29

4.2.1.	Proceduralno .....	30
4.2.2.	Dinamičko .....	34
4.2.3.	Nasumično .....	36
4.3.	Utjecaj na igračko iskustvo.....	37
5.	Analitika .....	39
5.1.	Metrike i praćenje utjecaja .....	39
5.2.	Analiza rezultata .....	42
	Zaključak .....	47
	Popis kratica .....	49
	Popis slika.....	50
	Popis kôdova .....	51
	Literatura .....	52



# 1. Uvod

Računalne igre postale su jedan od najpopularnijih načina zabave i njihova popularnost raste iz dana u dan (Blossom et Dunning, 2015). Jedan od ključnih faktora koji utječu na uspjeh računalnih igara je njihova kvaliteta. Kako bi računalna igra imala što veću kvalitetu inženjeri moraju znati kako je isprogramirati i koje su naj bolje prakse koje čine računalnu igru što zabavnijom i interaktivnijom (Short, 2017). Generiranje razina je ključni proces u razvoju računalnih igara, a u ovom diplomskom radu istražiti ćemo tri različite metode generiranja razina: nasumično, dinamičko i proceduralno generiranje razina. Alat u kojemu se ovaj diplomski rad napravio se zove Unity alat za kreiranje računalnih igri te je jedan od naj popularnijih alata za kreiranje računalnih igri gdje je po njihovim statistika 50% računalnih igara napravljeno u tom alatu. (Fuad, 2022)

Nasumično generiranje razina je najjednostavnija metoda generiranja razina. Kao što samo ime predlaže nasumično generiranje razina je nepredvidivo. Zbog toga metoda generiranja razina može biti vrlo uzbudljiva za igrače, ali nije uvijek učinkovita u stvaranju izazova i kvalitetne igračke dinamike. Dinamičko generiranje razina može biti učinkovitije u stvaranju izazova za igrače, ali je zahtjevnije za programiranje. Proceduralno generiranje razina, s druge strane, nudi najveću fleksibilnost i kontrolu nad kreiranjem razina, ali dolazi s manom što je složeno i zahtjevno za implementaciju. U ovom radu će također biti obrađeni primjeri poznatih igara koje koriste ove metode generiranja razina, a istražiti će se i kako su te igre postigle svoj uspjeh i što je doprinijelo njihovoj popularnosti među igračima.

Svrha ovog rada je istražiti kako ove tri metode generiranja razina utječu na igračko iskustvo i kvalitetu igre. Uz to, analizirat ćemo podatke o igračkom iskustvu kroz Unity analytics servis, kako bismo dobili bolji uvid u to kako igrači doživljavaju različite razine u igri te kako se različite metode generiranja razina odražavaju na igračko iskustvo. Unity analytics je servis unutar Unitya koji pomaže pri prikupljanju podataka i njihovog ispisa. (Fuad, 2022) Na kraju, cilj ovog rada je pružiti preporuke za razvojne timove i samostalne programere o tome koje metode generiranja razina trebaju koristiti kako bi stvorili najbolje iskustvo igračima.

## 2. Dokument specifikacije

GDD je dokument specifikacije (engl. *Game Design Document*) igre koji opisuje sve aspekte igre, uključujući priču, likove, svijet, mehanike igre, vizualni stil, zvuk i druge elemente. Dokument služi kao plan za razvoj igre i kao temelj za timove koji sudjeluju u razvoju igre. GDD je važan alat u procesu razvoja igre jer omogućuje timovima da razumiju i jasno definiraju svoje zadatke i ciljeve, kao i da se usklade oko zajedničke vizije i ciljeva za igru. GDD može biti vrlo detaljan i opsežan, a često se ažurira tijekom procesa razvoja igre kako bi se osiguralo da igra ostane na pravom putu.

### 2.1. Sažetak

Računalna igra "Machine Menace" je dvodimenzionalna igra iz ptičje perspektive koja pripada rogulike žanru računalnih igrica. Radnja se temelji na robotu koji je zarobljen u gnijezdu vanzemaljaca i mora prijeći prepreke te se boriti s vanzemalcima kako bi pobjegao. Na kraju svake razine, igrač se mora suočiti s najmoćnijim vanzemalcem, koji je čuvar gnijezda i pobijediti ga kako bi pobjegao.

Radnja cijele igre odvija se u malim kockastim sobama koje su generirane računalno, a u kojima se nalaze vanzemaljci i interaktivni objekti. Igrač se koristi mišem i tipkovnicom kako bi se kretao u dvodimenzionalnom svijetu. Pritiskom na tipke w, a, s, d igrač se može kretati gore, lijevo, dolje i desno, a tipka razmak omogućava mu brzo kretanje s jednog mjesta na drugo. Miš se koristi za rotiranje i ciljanje neprijatelja, a lijevim klikom miša igrač može pucati na njih.

Kako bi igraču bilo zanimljivo i intuitivno igrati igricu, omogućene su mnoge mogućnosti prilagođavanja igre njegovom stilu igranja. Igra ima tri načina generiranja razina: proceduralni, dinamički i nasumični, koje igrač može odabrati na početku igre. Proceduralni način igranja je jednostavan za razumijevanje i nudi dobar omjer učenja i zabave. Dinamičko generiranje olakšava ili otežava igru, ovisno o sposobnostima igrača. Nasumično generiranje pruža kaos, zabavu i adrenalin jer igrač ne zna što ga čeka u sljedećoj sobi.

Tijekom igranja, igraču pomaže mala mapa smještena u donjem desnom kutu ekrana koja prikazuje njegovu poziciju i izgled svih generiranih prostorija. Ovo mu daje jasan smjer kretanja i pomaže mu da se orijentira u igri.

## 2.2. Priča

Machine Menace je priča smještena u futuristički svijet gdje ljudi, suočeni s brutalnim ratom protiv napadačkih vanzemaljaca, donose odluku da zamijene svoje tijela s robotskim dijelovima. Ova odluka je bila nužna kako bi ljudi mogli preživjeti u svijetu u kojem su fizička tijela postala smetnja.

U tom svijetu, igrači se poistovjećuju s robotom koji se našao napušten u gnijezdu vanzemaljaca nakon što je odbačen u bitci. Robot sada mora naći svoj put van kroz mnoge protivnike, pazeći na razne zamke i opasnosti koje su postavili vanzemaljci kako bi zaštitili svoje gnijezdo.

Tijekom njegove avanture, robot nailazi na razne vanzemaljce koji ga pokušavaju uništiti iz blizine i iz daleka. Kako bi preživio, robot se mora boriti i sakupljati oružja koja su ostala iza drugih robota koji su nekada bili tamo. Međutim, na putu će se pojaviti i klopke poput mina koje su postavili drugi roboti kako bi uništili vanzemaljce, no koje sada smetaju igraču kako bi došao do svoga cilja.

Konačno, robot će se naći licem u lice s najjačim i najmoćnijim vanzemaljskim čuvarom koji pazi na izlaz iz gnijezda. U bitci koja slijedi, igrač će morati koristiti svoje vještine, oružja i inteligenciju kako bi nadmudrio vanzemaljca i pobijedio u posljednjem koraku svog puta do slobode.

Ova priča je zamišljena kao akcijska avantura za igrače koji vole izazove, taktičke odluke i napete borbe protiv raznih protivnika. Svojom mračnom atmosferom, zanimljivim likovima i uzbudljivom pričom, Machine Menace bi pružila igračima jedinstveno iskustvo u rogulike žanru računalnih igrica.

## 2.3. Mehanika računalne igre

Igra započinje u sobi gdje igrač može naučiti osnovne mehanike igre. Igrač ima neuništivog klona vanzemaljca i pristup svim trima vrstama oružja kako bi mogao testirati svoje preferencije. Nakon što se igrač upozna s mehanikama, kreće njegov prvi izbor - odabir stila igranja. Igrač može birati između tri različita načina generiranja razina, a odabrana razina otvara novu scenu gdje se prava igra započinje.

Prva soba u kojoj se igrač pojavljuje ima četiri otvorena izlaza - gore, dolje, lijevo i desno - te je u početku prazna. Igrač treba istražiti svijet kako bi pronašao izlaz. Međutim, na putu će naići na brojne prepreke - od objekata koji otežavaju kretanje, poput mina, do neprijatelja koji će pokušati uhvatiti ili gađati igrača. Srećom, postoje i pozitivni objekti, poput kutija koje sadrže oružja koja je igrač testirao u početnoj sobi.

Svaka generirana razina sadrži sobu s glavnim protivnikom. Ovi protivnici imaju više napada i izazov su za igrača. Postoje tri glavna protivnika, a svaki je novo iskustvo za igrača. Nakon što igrač pobijedi glavnog protivnika, otvara se izlaz koji vodi do ekrana zahvale na igranju. Igrač je zatim vraćen u početnu sobu, gdje može započeti svoju avanturu ponovno s istim ili drugačijim načinom generiranja razina.

### 3. Način generiranja razina

Generiranje razina u računalnim igrama igra ključnu ulogu u stvaranju iskustva igrača. Izrada različitih razina koja će izazvati igračevu pažnju, pružiti mu zabavu i izazov, predstavlja izazov i za programera. U sklopu ovog diplomskog rada odabrana su tri načina generiranja razina u računalnim igrama – nasumično generiranje razina, dinamičko generiranje razina i proceduralno generiranje razina.

Kada se radi o odabiru pravog načina generiranja razina za određenu računalnu igru, programeri igara moraju uzeti u obzir brojne čimbenike, uključujući vrstu igre, ciljanu publiku, kompleksnost igre, vremenska ograničenja i mogućnosti koje pruža razvojna platforma. Osim toga, programeri igara mogu koristiti kombinaciju različitih metoda generiranja razina kako bi stvorili što bolje iskustvo igračima.

Dinamičko generiranje razina se koristi u računalnoj igri Diablo III RPG koju je razvio i objavio studio Blizzard Entertainment. Dinamičko generiranje razina koristi se kako bi se omogućilo igračima da uvijek doživljavaju drugačije izazove i iskustva tijekom igranja. Ovisno o razini težine koju igrač odabere, računalna igra će generirati različite skupine neprijatelja, čudovišta i izazova na temelju igračevog iskustva i sposobnosti.

Proceduralno generiranje razina se koristi u računalnoj igri No Man's Sky koju je razvio i objavio studio Hello Games. Igra se sastoji od beskonačnog svemira koji igrači istražuju i otkrivaju. Svaki planet, zvijezda i galaksija u računalnoj igri su proceduralno generirani uz pomoć matematičkih algoritama i parametara koji utječu na fiziku, geografiju i ekologiju pojedinog svemirskog tijela.

Nasumično generiranje razina se koristi u računalnoj igri Binding of Isaac RPG koju su razvili nezavisni programeri Edmund McMillen i Florian Himsl. Nasumično generiranje razina u ovoj igrici je odabir developera jer je cilj igre pružiti igračima nepredvidljivo iskustvo, a nasumično generiranje razina je idealan način da se to postigne. Razine se generiraju nasumično, što znači da svaki put kada računalna igra počne iznova, igrač će doživjeti drugačiju verziju računalne igre s različitim izazovima i neprijateljima. To poboljšava igrivost računalne igre jer igrači ne mogu memorirati poziciju neprijatelja i izazova. Nasumično generirane razine također produžuju trajanje igre i pružaju neprekidno iskustvo za igrače.

Uz pomoć analitičkih alata, kao što je Unity Analytics, programeri igara mogu prikupljati podatke o igračevom iskustvu tijekom igre, uključujući vrijeme provedeno u igri, razine do kojih su igrači došli i koliko su puta prešli svaku razinu. Ovi podaci mogu pomoći programerima da bolje razumiju kako igrači doživljavaju različite razine u igri, kako se različite metode generiranja razina odražavaju na igračko iskustvo i kakve su sklonosti igrača kada su u pitanju različiti načini generiranja razina. Konačni cilj ovog rada je pružiti preporuke za razvojne timove i samostalne programere o tome koje metode generiranja razina trebaju koristiti kako bi stvorili najbolje iskustvo igračima.

### **3.1. Dinamičko generiranje**

Dinamičko generiranje razina u računalnim igrama je proces koji omogućava stvaranje razina igre na temelju igračevog ponašanja i sposobnosti, a može se primijeniti u različitim žanrovima igara, kao što su pucačine, platforme, RPG i slično. Ovaj proces može poboljšati korisničko iskustvo tako što će se igra prilagoditi sposobnostima igrača, a istovremeno i učiniti igru izazovnijom.

Dinamičko generiranje razina se sastoji od nekoliko faza. Prva faza je prikupljanje podataka o igraču, kao što su njegova sposobnost, iskustvo i prethodne pobjede u igri. Zatim se analiziraju podaci kako bi se procijenilo koje su razine igre pogodne za igrača. Na temelju analize, algoritam generira novu razinu igre, koja odgovara sposobnostima igrača, a istovremeno mu pruža izazov.

Algoritmi za generiranje dinamičkih razina u igrama obično koriste tehnike strojnog učenja i umjetne inteligencije kako bi se analizirali podaci o igraču. Na primjer, algoritmi mogu koristiti neuronske mreže ili genetske algoritme za generiranje novih razina.

Prednosti dinamičkog generiranja razina u računalnim igrama su mnogobrojne. Prvo, omogućava igri da se prilagodi različitim vrstama igrača. Drugo, stvara izazovniju igru i održava igru zanimljivom tijekom vremena. Treće, smanjuje vrijeme koje razvojni timovi moraju potrošiti na stvaranje novih razina igre, jer algoritmi mogu stvoriti velik broj različitih razina u kratkom vremenskom periodu.

Iako dinamičko generiranje razina u računalnim igrama ima mnoge prednosti, postoje i neke mane s kojima se developeri moraju nositi. Na primjer, algoritmi se moraju pažljivo podešavati kako bi se osiguralo da generirane razine budu izazovne, ali ne i previše teške za

igrače. Također, važno je osigurati da se generirane razine igre ne ponavljaju previše često, jer to bi moglo dovesti do gubitka interesa igrača.

## 3.2. Nasumično generiranje

Nasumično generiranje razina u računalnim igrama je proces koji koristi algoritme za stvaranje različitih razina igre iz nasumično odabranog skupa elemenata. Ova metoda se često koristi u igrama koje zahtijevaju veliki broj različitih razina kako bi se osigurao dugotrajni izazov za igrače.

Jedna od prednosti nasumičnog generiranja razina je ta da se mogu stvoriti velike količine različitih razina igre u kratkom vremenskom periodu. To omogućava razvojnim timovima da uštede vrijeme koje bi inače potrošili na ručno stvaranje različitih razina igre. Također, ova metoda stvara iznenađenja za igrače, koji ne znaju što ih očekuje u sljedećoj razini, što dodatno poboljšava iskustvo igranja.

Nasumično generiranje razina u igrama može se primijeniti u različitim žanrovima igara, kao što su pucačine, platforme, RPG i drugo. Algoritmi za generiranje nasumičnih razina mogu koristiti različite metode, kao što su generiranje slučajnih brojeva, slučajne matrice ili algoritme za traženje puta, kako bi stvorili različite kombinacije elemenata u razini igre.

Međutim, postoji i nekoliko izazova koji se moraju riješiti pri korištenju nasumičnog generiranja razina u igrama. Prvo, generirane razine igre mogu biti preteške ili prelagane za igrače, što može dovesti do frustracije ili dosade. Stoga je važno osigurati da algoritmi koji se koriste za generiranje razina stvaraju razinu koja je izazovna, ali ne i preteška za igrače.

Drugo, nasumično generirane razine igre mogu biti manje strukturirane i manje dobro osmišljene od ručno stvorenih razina igre. Stoga je važno osigurati da se nasumično generirane razine igre pažljivo testiraju kako bi se osiguralo da se igrači neće izgubiti ili se suočiti s nedostatkom izazova.

### 3.3. Proceduralno generiranje

Proceduralno generiranje razina u računalnim igrama je proces koji koristi algoritme kako bi stvorio različite razine igre. Ova metoda je slična nasumičnom generiranju razina, no umjesto da se koristi nasumični skup elemenata, koriste se pravila i uvjeti kako bi se stvorile strukturirane i dobro osmišljene razine.

Jedna od prednosti proceduralnog generiranja razina je ta da se može stvoriti velik broj različitih razina igre koje su dobro osmišljene i izazovne za igrače. Ova metoda također može biti korisna u igrama koje imaju otvoreni svijet, jer omogućava igračima da istražuju i otkrivaju nove dijelove igre bez obzira na redoslijed.

Proceduralno generiranje razina može se koristiti u različitim žanrovima igara, kao što su RPG, pucačine, platforme i drugo. Algoritmi za generiranje razina mogu koristiti različite metode, kao što su evolucijski algoritmi, algoritmi za traženje puta i drugo.

Međutim, postoji i nekoliko izazova koji se moraju riješiti pri korištenju proceduralnog generiranja razina u igrama. Prvo, postoji opasnost da će stvorene razine biti manje strukturirane i manje osmišljene od ručno stvorenih razina igre. Stoga je važno osigurati da algoritmi koji se koriste za generiranje razina igre imaju dovoljno pravila i uvjeta kako bi se stvorile dobro strukturirane razine.

Drugo, proceduralno generirane razine igre mogu biti previše teške ili prelagane za igrače, što može dovesti do frustracije ili dosade. Stoga je važno osigurati da se algoritmi za generiranje razina testiraju kako bi se osiguralo da su izazovne, ali ne i preteške za igrače.



## 4. Izrada računalne igre

Za izradu računalne igre u ovom diplomskom radu odabran je Unity. Unity je jedna od najpopularnijih platformi za izradu računalnih igara, a koristi se za izradu računalnih igara na različitim platformama, uključujući mobilne uređaje, računala i konzole. Unity olakšava razvoj računalnih igara. (Scolastici, 2015) U ovom diplomskom radu, opisat ćemo proces kreiranja 2D računalne igre u Unity platformi. Za izradu računalne igre koristio se C# programski jezik.

Prvi korak u izradi igre u Unityju je planiranje igre. U ovom koraku se razmišlja o tome koju računalnu igru stvoriti i koje funkcije će biti potrebne za izradu te računalne igre. Ovo uključuje razmišljanje o vrsti računalne igre, mehanikama računalne igre, vizualnom stilu računalne igre i drugim ključnim elementima.

U sklopu diplomskog rada izrađena je dvodimenzionalna računalna igra, koja će služiti kao podloga za prikaz različitih načina generiranja razina u računalnim igrama kao što su: proceduralno generiranje, dinamičko generiranje i nasumično generiranje.

Za tip igre u ovom diplomski rad odabran je roguelike žanr računalnih igara koji je poznat po neograničenoj ponovnoj igrivosti koja se postiže generiranjem razina i neprijatelja na različite načine kako igraču nikada ne bi bilo dosadno (Watkins, 2016). Postoje mnoge roguelike računalne igre te svaka od njih koristi drugačiji način generiranja razina i ne postoji točan odgovor koji način kreiranja razina je naj zanimljiv ili naj zanosniji za igrača, te su zbog toga u ovom diplomskom odabrana tri naj poznatija i naj korištenija načina generiranja razina kako bi se odgovorilo na to pitanje.

Nakon što se napravi plan računalne igre, sljedeći korak je kreiranje projekta u Unityju. U ovom koraku se stvara novi projekt i biraju željene postavke, kao što su platforme za objavu računalne igre i željene postavke grafike. Nakon toga započinje proces kreiranja računalne igre.

U nastavku poglavlju ćemo detaljno obraditi proces kreiranja računalne igre u Unity platformi. Kreiranje igre je složen i zahtjevan proces koji zahtijeva mnogo vještina i znanja, a mi ćemo se posvetiti svakom koraku procesa.

Prvo ćemo istražiti proces smišljanja i planiranja igre. To uključuje definiranje ciljeva igre, razmatranje ciljne publike i uspostavljanje temeljnih koncepata igre. Nakon što smo

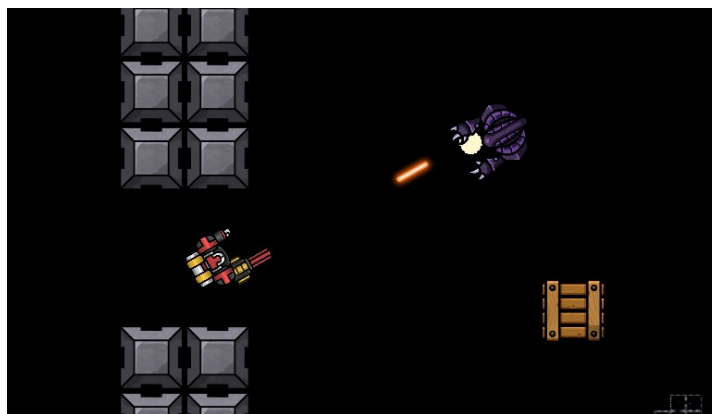
definirali te koncepte, prelazimo na stvaranje osnovnih elemenata igre, kao što su karakteri, oružja, predmeti, okruženja i neprijatelji. Ovi elementi su ključni za stvaranje jedinstvenog i zanimljivog iskustva igranja. Svi koraci bit će popraćeni programskim kodom i detaljnim objašnjenjem koji će pojasniti funkcioniranje pojedinog djela računalne igrice.

## **4.1. Izrada bazne računalne igre**

### **4.1.1. Osnovni koncept i ideja**

Računalna igra (radni naslov *Machine Menace*) je dvodimenzionalna računalna igra u kojoj igrač preuzima ulogu robota i istražuje nasumično generirane sobe u nastojanju da pronađe izlaz iz gnijezda vanzemaljaca. Dok istražuje, igrač se suočava s nizom neprijatelja (vanzemaljaca) koji ga ometaju u pronalasku puta do izlaza. Prije početka bijega iz tamnice, igrač ima izbor načina generiranja puta kojim će pokušati pobjeći. Uz to, igrač će imati priliku naići na kutije koje sadrže moćnija oružja, čime će mu biti olakšan bijeg. Računalna igrice je napravljena s ciljem istraživanja različitih utjecaja generiranja razina na igračko iskustvo te kako bi se dobili uvidi u različite preference igrača.

Osnovne mogućnosti igrača u ovoj igri su slobodno kretanje u dvije dimenzije te korištenje oružja za borbu s neprijateljima ili uništavanje generiranih objekata poput kutija. Kao što se može vidjeti na slici Slika 4.1 Slika igrice, na početku igre igrač se nalazi u startnoj zoni i od tamo odabire i pristupa određenom generiranom putu po svom izboru. Svaka razina (engl. *level*) sadrži neprijatelje, objekte koji se mogu razbiti i zamke. Ako igrač uspješno stigne do izlaza, vraća se ponovno u startnu zonu, odakle može započeti novi bijeg i odabrati način na koji će ga pokušati ostvariti.



Slika 4.1 Slika igrice

Iako postoji velik broj igara u roguelike žanru koje koriste različita generiranja razina kako bi igraču omogućile jedinstveno igračko iskustvo, ova igra nudi više različitih načina generiranja razina, što omogućuje igračima da odaberu svoj omiljeni način generiranja i osiguraju najbolje igračko iskustvo za sebe.

#### 4.1.2. Osnovni elementi Unity platforme

Osnovni prozori unutar Unity platforme su (Tuliper et Blacman, 2016; Scolastici, 2015):

- Ekran za prikaz scene
- Ekran za prikaz igre
- Hijerarhijski pregled objekata
- Prikaz foldera
- Prikaz konzole
- Inspektor

Ekran za prikaz scene (engl. *Scene view window*) predstavlja jedan od osnovnih alata koje Unity pruža za vizualizaciju i manipulaciju scene u realnom vremenu. Ovaj prozor je iznimno važan za developere jer omogućava interaktivno uređivanje scene u stvarnom vremenu, a time olakšava i ubrzava proces razvoja igara. Ovaj prozor omogućava developerima da stvore, uredi i nadgledaju scene u 2D i 3D prostoru. Ekran za prikaz scene pruža mogućnost dodavanja, uklanjanja i transformacije objekata, kao i manipuliranje svim drugim elementima scene, uključujući osvjetljenje, teksturu, animaciju i kameru. Ekran za prikaz scene također pruža mogućnost postavljanja scenarija i animacija unutar igre.

Ekran za prikaz igre (engl. *Game view window*) je jedna od ključnih značajki ove razvojne platforme koja programerima igara pruža mogućnost da stvore i testiraju svoju igru u realnom vremenu. Jedna od osnovnih funkcija ekran za prikaz igre je pružanje programerima igara mogućnost da vide kako će njihova igra izgledati i funkcionirati u konačnom izdanju, dok je još uvijek u procesu razvoja. Korištenjem ekrana za prikaz igre, programeri mogu testirati različite dijelove igre, kao što su kretanje likova, ponašanje neprijatelja, interakcija igrača s okolinom i tako dalje. Kao takav, ekran za prikaz igre omogućava programerima da se fokusiraju na detalje u igri, uključujući i sitne grafičke i mehaničke elemente, te ih testirati u stvarnom vremenu. Ekran za prikaz igre radi tako da vraća informacije od kamere koja je jedna od naj bitnijih elemenata u Unityu.

Kamera je jedan od najvažnijih elemenata u izradi igre u Unity platformi. Ona omogućuje igraču da vidi igru iz perspektive određene pozicije i orijentacije (Nugent et Buttfield-Addison et Manning, 2016).. Kamera može biti postavljena u različitim modovima, uključujući perspektivni (engl. *perspective*), ortografski (engl. *orthographic*) i sferni (engl. *spherical*) način. Svaki od ovih modova ima svoje prednosti i mane te se odabir kamere ovisi o vrsti igre i željenom efektu. U kombinaciji s ekranom za prikaz igre, kamera omogućuje igraču da doživi igru.

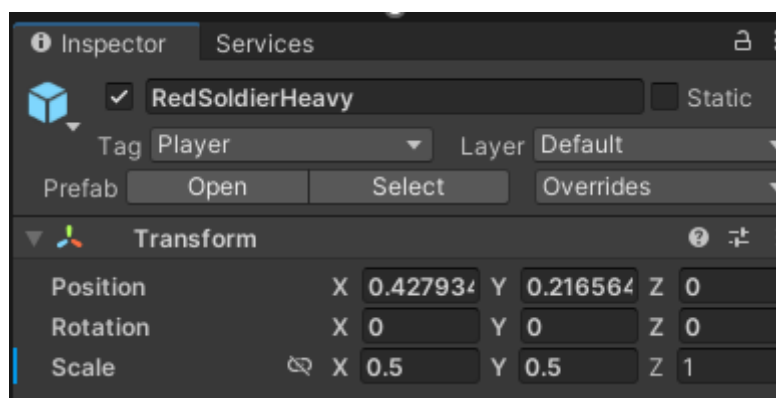
Hijerarhijski pregled objekata pruža programerima i dizajnerima uvid u organizaciju objekata u igri i omogućava lakše upravljanje s njima. Jedan od ključnih razloga zašto je hijerarhijski pregled bitan je taj što omogućava lako organiziranje i manipulaciju objektima u igri (Nugent et Buttfield-Addison et Manning, 2016). Korištenjem hijerarhijskog pregleda, programer ili dizajner može jednostavno organizirati objekte po različitim kategorijama, kao što su likovi, oružja, neprijatelji, zgrade i drugo. Na primjer, ako programer želi napraviti animaciju za lika u igri, on može jednostavno pronaći lika u hijerarhijskom pregledu i odabrati ga za animaciju.

Svi elementi unutar hijerarhijskog prikaza organizirani su prema roditelj-dijete vezi (eng. *Parent-child-relationship*). Roditelj-dijete arhitektura sastoji se od dvije vrste igračkih objekata roditeljskih objekata i njihove djece objekata. Svaki objekt može biti roditelj i dijete u isto vrijeme, osim ako nije postavljen kao korijen hijerarhije objekata. Roditelj igračkog objekta može imati jedan ili više djece objekata, dok dijete objekta može biti dijete drugih objekata ili može biti krajnji list u hijerarhiji objekata.

Prikaz foldera u Unityju je element koji igra važnu ulogu u organizaciji i upravljanju projektom. Unity omogućuje korisnicima da organiziraju sve elemente projekta, kao što su skripte, teksture, modele, zvukovi i ostali materijali, u različite foldere (Nugent et Buttfield-Addison et Manning, 2016).. Kada se projekt postaje složeniji, pregledavanje i pronalaženje određenih elemenata postaje teže i može potrajati dugo vremena. Ovdje nastupa prikaz foldera u Unityju, koji olakšava pronalaženje i upravljanje različitim elementima projekta. Jedna od ključnih prednosti prikaza foldera u Unityju je jednostavno organiziranje svih elemenata projekta u hijerarhiju foldera. Ova hijerarhija pomaže korisnicima da lako pristupe određenim elementima projekta i poboljšaju svoju produktivnost.

Konzola (eng. console) je alat u Unityu koji omogućuje programerima i razvojnim timovima da prate stanje igre, dijagnostičke informacije, upozorenja i pogreške koje se javljaju tijekom izvođenja igre. Konzola je korisna za razvoj igre, testiranje, rješavanje problema i optimizaciju performansi. Programeri mogu koristiti konzolu kako bi testirali određene dijelove koda i provjerili da li rade kako treba.

Inspektor (eng. Inspector) je jedan od alata u koji igra važnu ulogu u razvoju igara. Inspektor se koristi za pregled i manipuliranje svojstvima i komponentama igračkog objekta u igri. Svaki igrački objekt u igri ima svojstva, kao što su pozicija, rotacija i skala, koje se mogu uređivati u inspektoru kao na slici Slika 4.2 Slika inspektora. Inspektor se prikazuje na desnoj strani Unity sučelja, a sadrži popis svih komponenti i svojstava koji se mogu prilagoditi. Omogućava programerima i dizajnerima da pregledaju i uredе svojstva objekta u realnom vremenu, što im daje potpunu kontrolu nad igrom i njenim funkcionalnostima. Osim toga, inspektor omogućuje i uvid u vrijednosti varijabli koje se koriste u kodu, što olakšava pronalaženje grešaka u kodu.



Slika 4.2 Slika inspektora

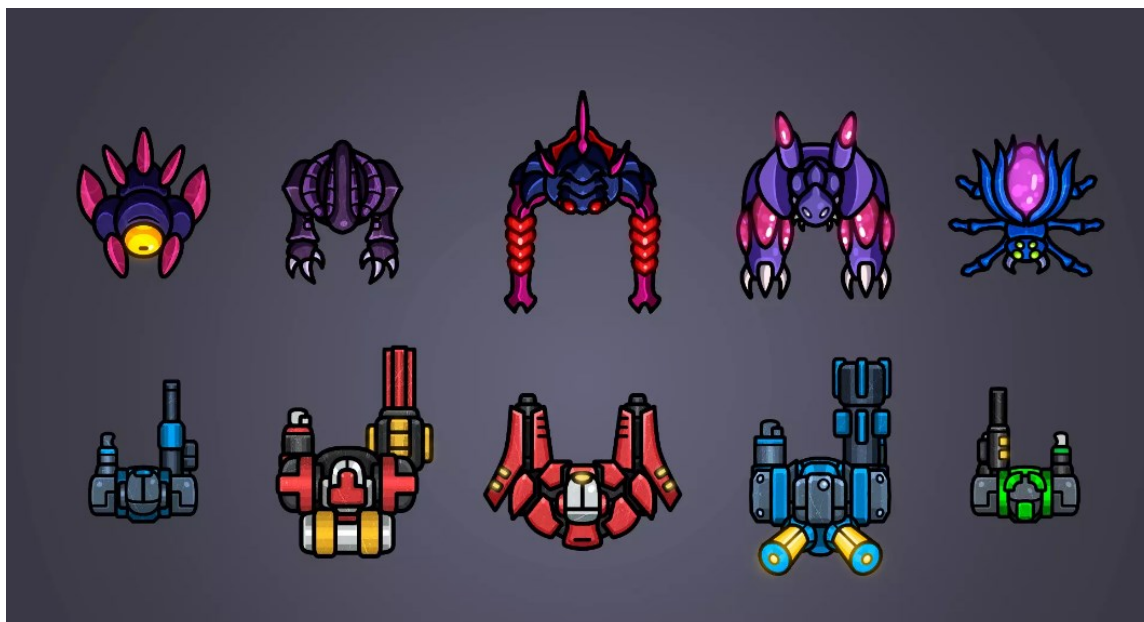
### 4.1.3. Objekti

Prvi korak pri kreiranju igre je kreiranje objekata koji će se koristiti u scenama. Takvi objekti zovu se (engl. *Prefab*). Prefab je predložak objekta koji se može ponovno koristiti u scenama (Scolastici, 2015). Prefab se može stvoriti od bilo kojeg objekta u sceni, a zatim se taj prefab može koristiti za stvaranje mnogih kopija objekta u različitim scenama. Korištenjem prefaba može se uštedjeti vrijeme i olakšati rad, jer se ne mora ručno ponavljati stvaranje istih objekata i postavki za svaku scenu. Uz to, promjene koje se naprave na prefabu automatski će se primijeniti na sve instance tog prefaba u scenama.

### 4.1.4. Grafika

Prije početka izrade elemenata potrebno je pripremiti gotove grafičke komponente koje će određivati izgled igrice. Sva grafika korištena u ovoj računalnoj igri je dobivena korištenjem jednog od potencijalno naj jačih alata u unityu. (Tuliper, 2014) Korištenjem gotovih grafika iz trgovine dodatcima unutar Unitya (engl. *Unity asset store*) pod nazivom Futuristic pack kao na slici Slika 4.3 Prikaz likova iz Futuristic packa. sa sljedeće lokacije –

<https://assetstore.unity.com/packages/2d/characters/top-down-assets-futuristic-pack-133669>

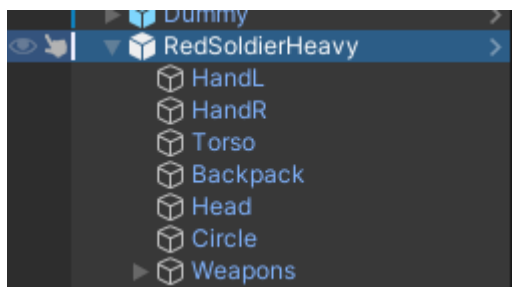


Slika 4.3 Prikaz likova iz Futuristic packa

Futuristic Pack je jedan od popularnijih paketa unutar trgovine dodatcima, koji nudi sve potrebne elemente za stvaranje futurističkog svijeta. Ovaj paket sadrži preko 200 unaprijed izrađenih modela, tekstura i materijala koje se mogu koristiti za izgradnju svemirskih brodova, futurističkih građevina, okoliša i drugih elemenata u igrici.

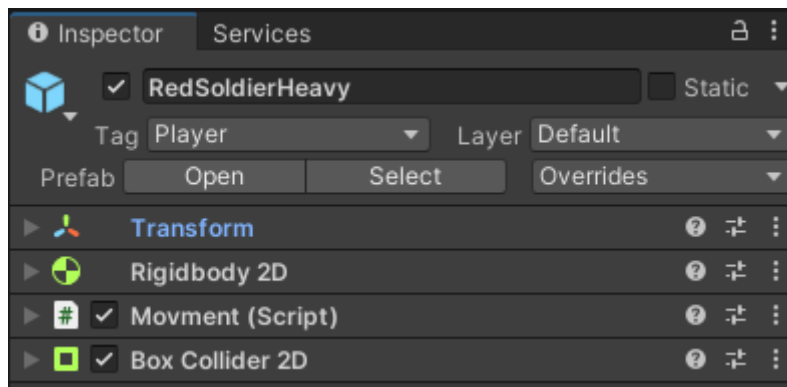
#### 4.1.5. Glavni lik

Glavni lik u igrici je objekt koji je najviše fokusiran u igri, a kojim igrač upravlja i koji najviše utječe na napredak igre. Glavni lik može biti različit za različite igre, ovisno o vrsti igre, njenom žanru i priči. Jedna od naj bitnijih stavki je definiranje grafičkog izgleda igrača, a on se može vidjeti na slici Slika 4.1 Slika igrice. Igrač je glavni objekt koji se sastoji od više manjih objekata po parent-child principu. Objekti od kojih se igrač sastoji su: tijelo, lijeva i desna ruka, ruksak, glava i oružje kao na slici Slika 4.4 Slika objekta igrača



Slika 4.4 Slika objekta igrača

Bitno dodati sve potrebne komponente unutar inspektora kako bi igrač imao sve potrebne predispozicije kako bi radio kako je to zamišljeno. Nakon što se definira grafički izgled igrača, važno je dodati sve potrebne komponente unutar inspektora kako bi igrač imao sve potrebne predispozicije kako bi radio kako je to zamišljeno. Komponente u inspektoru mogu uključivati skripte koje upravljaju kretanjem lika, njegovom interakcijom s objektima u igri, njegovim sposobnostima i drugim aspektima lika i njegove okoline. Na glavnom liku u igrici su dodane komponente BoxCollider 2D komponenta, Rigidbody 2D komponenta i skripta za micanje kao na slici Slika 4.5 Prikaz komponenti na objektu igrača.



Slika 4.5 Prikaz komponenti na objektu igrača

Rigidbody 2D komponenta u Unityu je dio sistema fizike igre koji simulira dinamičko ponašanje objekata unutar igre. Ova komponenta se koristi za simuliranje ponašanja tijela u dvodimenzionalnom prostoru, kao što su kretanje, sudari, gravitacija, rotacija i interakcije s drugim objektima u igri. Ukratko Rigidbody 2D komponenta služi za određivanje fizičkih svojstva objekta u 2D prostoru.

Box Collider 2D komponenta u Unityu je dio sistema kolizija igre koji definira 2D geometriju objekata u igri. Ova komponenta se koristi za definiranje okvirnog oblika objekata unutar igre kako bi se omogućilo detektiranje sudara između objekata. Na objektu glavnog lika ova komponenta služi kako bi detektirala napade od neprijatelja, određene dodire s objektima i odredila dužinu i duljinu doticaja igrača.

```

Vector2 movment;
public Rigidbody2D rb;
public float moveSpeed = 5f;
void Update() {
    movment.x = Input.GetAxisRaw("Horizontal");
    movment.y = Input.GetAxisRaw("Vertical");
}
void FixedUpdate() {
    rb.MovePosition(rb.position +
        movment *
        moveSpeed *
        Time.fixedDeltaTime);
}

```

Kôd 4.1 Funkcija za pomicanje igrača

Ovaj isječak kodaKôd 4.1 Funkcija za pomicanje igrača pokazuje primjenu Vector2 varijable s nazivom movement. Ona se koristi za pohranjivanje vrijednosti x i y osi ovisno



o pritisku gumba igrača za micanje u tim smjerovima. Kako bi se igrač mogao glatko micati, poziva se instanca `Rigidbody2D` komponente unutar `FixedUpdate` metode, koja omogućuje korištenje `MovePosition` metode za glatki prijelaz objekta i stvaranje osjećaja micanja. Ovdje se koristi `FixedUpdate` metoda umjesto `Update` metode za micanje igrača jer nije optimalno da se micanje igrača ovisi o frame rate-u. `FixedUpdate` metoda se koristi kako bi se osiguralo da se fizički procesi u igri izvršavaju u redovitim vremenskim intervalima, neovisno o tome koliko je puta pozvana u sekundi. To osigurava stabilnost i predvidljivost ponašanja igre.

```

Vector2 mouse;
public Camera cam;
void Update() {
    mouse = cam.ScreenToWorldPoint(Input.mousePosition);
}
void FixedUpdate() {
    Vector2 lookDir = mouse - rb.position;
    float angle = Mathf.Atan2(lookDir.y, lookDir.x) *
    Mathf.Rad2Deg - 90f;
    rb.rotation = angle + 0.5f;
}

```

#### Kôd 4.2 Program za ciljanje

Ovaj isječak koda Kôd 4.2 Program za ciljanje se koristi za ciljanje objekata u igri. U `Update()` metodi, kod prima poziciju miša na ekranu i pretvara je u koordinate svijeta koristeći funkciju `ScreenToWorldPoint()` kamere koja je postavljena u varijablu `cam`. Rezultat se pohranjuje u `Vector2 mouse`. U `FixedUpdate()` metodi, vektor `lookDir` računa smjer od pozicije objekta (pohranjeno u varijabli `rb.position`) prema poziciji miša. Zatim se računa kut između vektora i x-osi pomoću metode `Atan2`, što će biti orijentacija objekta. Taj se kut pohranjuje u varijablu `angle`. Konačno, ta vrijednost kuta se postavlja kao rotacija objekta pomoću `rb.rotation`.

```

void Update() {
    var dash = Input.GetButtonDown("Jump");
    if (dash && canDash) {
        isDashing = true;
        canDash = false;
        tr.emitting = true;
        dashDirection = new Vector2(movement.x, movement.y);
    }
}

```

```

        StartCoroutine(StopDashing());
    }
    if (isDashing){
        rb.AddForce(dashDirection.normalized * dashSpeed, 0f);
        return;
    }
    private IEnumerator StopDashing(){
        yield return new WaitForSeconds(dashTime);
        tr.emitting = false;
        isDashing = false;
        canDash = true;
    }

```

#### Kôd 4.3 Funkcija za brzo kretanje

Prethodni isječak koda Kôd 4.3 Funkcija za brzo kretanje prikazuje kod za brzi pomak igrača tijekom pritiska gumba razmaka. Ako je gumb pritisnut i ako igrač zadovoljava uvjet da se brzo pomakne, varijabla `canDash` se mijenja na vrijednost koja ne bi omogućila ponovno pokretanje brzog pomaka (engl. *false*). Varijable `isDashing` i `tr.emitting` postavljaju se na pozitivnu vrijednost kako bi se igraču putem varijable `isDashing` mogla dodati sila koja bi ga gurala, a putem `tr.emitting` bi mu se dao vizualni efekt čestica koji bi ukazivao na brzo kretanje igrača. Efekti su vrlo bitan dio igrice zato što se pomoću njih igraču može dočarati osjećaj stvarnost nekog objekta (Roberts, 2011)

Nakon što se sve postavi, pokreće se korutina koja nakon određenog vremena zaustavlja brzo kretanje igrača, vraća mu normalnu brzinu i postavlja efekt čestica na negativnu vrijednost, a `canDash` varijablu na pozitivnu vrijednost tako da se igrač može ponovno brzo kretati nakon prethodnog brzog kretanja.

```

public Slider slider;
private void Start(){
    slider.value = (float)health/100f;
}
public void TakeDamage(int damage){
    health -= damage;
    slider.value = (float)health / 100f;
}

```

#### Kôd 4.4 Funkcija za zdravlje igrača

Prikazani dio koda Kôd 4.4 Funkcija za zdravlje igrača opisuje kako radi zdravlje igrača u računalnoj igri. Prvo se stvara referenca na klizač (engl. *Slider*) element u igri. Ta referenca služi kako bi se popunio vizualni prikaz zdravlja igrača kao na slici Slika 4.6 Prikaz promjene igračevog zdravlja U metodi `Start()`, koja se poziva jednom kada se scena pokrene, vrijednost klizača se postavlja na temelju početne vrijednosti zdravlja igrača. Ako je `health` varijabla postavljena na 100, klizač će biti postavljen na vrijednost 1 (jer je skala klizača od 0 do 1).



Slika 4.6 Prikaz promjene igračevog zdravlja

Funkcija `TakeDamage()` uzima ulazni parametar `damage`, koji predstavlja količinu oštećenja koju treba oduzeti od igračevog zdravlja (`health` varijable). Ta funkcija također pomiče klizač koji je vizualni prikaz igračevog trenutnog stanja zdravlja kako bi bio u skladu s njihovim pravim zdravljem. Primjer bi bio da igrač ima 100 bodova za zdravlje i primi 50 bodova oštećenja, to bi dovelo ažuriranja klizača da rezultata izgleda kao na slici Slika 4.6 Prikaz promjene igračevog zdravlja.

#### 4.1.6. Neprijatelji

U Unityju, neprijatelji su objekti ili entiteti u igri koji su dizajnirani da djeluju kao prepreke ili protivnici igrača, a njihov cilj je ometati ili oštetiti igrača. Neprijatelji mogu imati različite oblike, veličine, ponašanja i sposobnosti, a mogu biti predstavljeni u igri kao 2D ili 3D modeli, čestice ili drugi vizualni elementi. Mogu se kretati po različitim putanjama ili ciljati igrača, a igrač ih može napadati ili izbjegavati ovisno o igri. Neprijatelji se često koriste u igrama kao način za povećanje izazova i napetosti, te za davanje igračima dodatnih ciljeva ili izazova koje treba svladati. U ovoj igrici postoje dva tipa neprijatelja, a to su: daleko dometni i kratko dometni.

##### 4.1.6.1 Kratko dometni

Kratko dometni neprijatelji su objekti u igrici koji služe kako bi stjerali igrača u kut i natjerali ga da koristi sve moguće resurse i strategiju kako bi ih izbjegavao ili kako bi ih uništio da slobodno može doći do cilja.

```

void Start () {
    player =
        GameObject.FindGameObjectWithTag("Player").transform
}
void Update () {
    else if (Vector2.Distance(transform.position,
                               player.position) < follow) {
        transform.rotation = Quaternion.AngleAxis(angle,
        Vector3.forward);
        transform.position =
        Vector2.MoveTowards(transform.position,
                             player.position,
                             movement * Time.deltaTime);
    }
}

```

#### Kôd 4.5 Funkcija za praćenje igrača

Isječak koda Kôd 4.5 Funkcija za praćenje igrača prikazuje funkciju za praćenje igrača koji koriste neprijateljski objekti. U funkciji `Start()`, prvo se pronalazi igračev objekt u sceni s oznakom taga igrač (engl. *Player*) i dohvaća se njegova transformacija. Transformacija objekta uključuje podatke o poziciji, rotaciji i skaliranju objekta. Pomoću igračeve transformacije možemo natjerati objekt neprijatelja da ga prati to jest napada. U funkciji `Update()`, provjerava se udaljenost između pozicije objekta na kojem se nalazi skripta i igračevog objekta. Ako je udaljenost manja od prethodno definirane vrijednosti `follow`, tada se objekt rotira prema igračevom objektu, a zatim se koristi funkcija `MoveTowards` da se objekt pomakne prema igračevom objektu s predefiniranom brzinom `movement` te se pomnoži s `Time.deltaTime`. `Time.deltaTime` predstavlja vremenski interval protekli od posljednjeg frejma, što se koristi za glatko kretanje objekta neovisno o tome koliko često se ažurira `frame rate` u igri.

#### 4.1.6.2 Daleko dometni

Daleko dometni neprijatelji su objekti koji služe kako bi igraču dali izazov. Daleko dometni neprijatelji su vrlo agresivni i služe kako bi igraču otežali tako da ne mora samo izbjegavati kratkodometne neprijatelje nego da se također mora i brinuti za projekte koje izbacuju

daleko dometni neprijatelji dok se bori. Što dovodi do potrebe za učenjem i analiziranjem same igrice od strane igrača.

```
void Update() {
    var direction = player.position - transform.position;
    var angle = Mathf.Atan2(direction.y, direction.x) *
        Mathf.Rad2Deg - 90;
    if (isShooter && Vector2.Distance
        (transform.position, player.position) < follow){
        if (Vector2.Distance
            (transform.position, player.position) > stop){
            transform.position =
                Vector2.MoveTowards(transform.position,
                    player.position,
                    movment * Time.deltaTime);
            transform.rotation =
                Quaternion.AngleAxis(angle,
                    Vector3.forward);
        }
    }
    else if (Vector2.Distance(transform.position,
        player.position) < stop &&
        Vector2.Distance(transform.position,
        player.position) > retreat){
        transform.rotation =
            Quaternion.AngleAxis(angle, Vector3.forward);
        transform.position = this.transform.position;
    }
    else if (Vector2.Distance(transform.position,
        player.position) < retreat){
        transform.rotation = Quaternion.AngleAxis(angle,
            Vector3.forward);
        transform.position =
            Vector2.MoveTowards(transform.position,
                player.position,
                - movment * Time.deltaTime);
    }
}}
```

#### Kôd 4.6 Funkcija za praćenje i pucanje neprijatelja

Ovaj kod prikazuje ponašanje neprijatelja u igri koji pucaju na igrača kada se igrač približi njima unutar određene udaljenosti. Kôd 4.6 Funkcija za praćenje i pucanje se izvršava u

svakom ciklusu ažuriranja igre, što znači da se kontinuirano provjerava i ažurira u svakom trenutku igre.

Varijabla `direction` računa razliku pozicije igrača i pozicije neprijatelja kako bi se odredio smjer u kojem se treba kretati. Nakon toga, računa se kut prema igraču iz korijena koordinatnog sustava neprijatelja, a zatim se okreće prema tom kutu.

Ako je neprijatelj tip koji puca, a igrač se nalazi unutar zadanog radijusa, neprijatelj će se kretati prema igraču pomoću funkcije `Vector2.MoveTowards` koja će omogućiti postupno približavanje igraču s određenom brzinom `movment` definiranom u kodu. Također, neprijatelj će se okretati prema igraču pomoću `transform.rotation` funkcije kako bi se uvijek usmjerio prema igraču.

Ako je igrač unutar određene udaljenosti, ali preblizu neprijatelju (unutar granica definiranih sa `stop` varijablom), neprijatelj se neće kretati, već će samo slijediti igrača rotirajući se prema njemu.

Ako je igrač preblizu (unutar granica definiranih s `retreat` varijablom), neprijatelj će se kretati unatrag s određenom brzinom koja je također definirana u kodu kako bi izbjegao direktni sukob s igračem.

Sve navedene radnje se stalno izvršavaju u svakom ciklusu ažuriranja igre, tako da neprijatelj neprestano prati igrača i reagira na njegovo približavanje na različite načine, ovisno o udaljenosti između njih.

#### 4.1.7. Kamera

Kamera je jedan od naj bitnijih objekata u Unityu. Ona služi kako bi omogućila fokus na igračev objekt (lika) i to Unity čini vrlo lakim za implementaciju (Watkins, 2016). Svrha kamere je također da developer ima uvid kako će igrice izgledati kada će biti u igračevim rukama te kako neki objekti utječu na igračevo iskustvo i igračev vid u igrici.

```
void Update() {  
    transform.position = new Vector3(target.position.x,  
                                    target.position.y,  
                                    target.position.z - distance);  
}
```

Kôd 4.7 Funkcija za praćenje igrača

Ovaj kodKôd 4.7 Funkcija za praćenje igrača predstavlja jednostavnu implementaciju kamere koja prati igrača u 2D prostoru. U ovom slučaju, kamera se pomiče tako da joj je uvijek pozicija jednaka poziciji igrača, ali se udaljenost kamere od igrača kontrolira varijablom `distance`.

Konkretno, kod se nalazi u `Update` metodi, što znači da se poziva svaki frame. U sklopu ove metode se izvršava jedna naredba, a to je postavljanje pozicije transform kamere na novu poziciju `Vector3`. Nova pozicija se izračunava tako da se uzme pozicija igrača `target.position`, a zatim se na tu poziciju dodaje vektor koji će dati željenu udaljenost kamere od igrača u z-osi. U ovom slučaju se oduzima vrijednost varijable `distance` od pozicije igrača u z-osi kako bi kamera bila uvijek iza igrača.

#### 4.1.8. Oružja

Oružja su bitan element igre pošto igraču pomažu kako bi došao do cilja. Kroz cijelu igru igrač može naletjeti na 3 različite vrste oružja koje su: minigun, snajper i zolja. Sva oružja služe različitim svrhama kako bi igrač dobio što veći raspon mogućnosti igranja te kako bi se zadovoljila što veća populacija igrača.

```
void Update() {
    if (Input.GetButton("Fire1")) {
        if (canShoot) {
            Shoot();
            StartCoroutine(NextBullet());
        }
    }
}

public IEnumerator NextBullet() {
    canShoot = false;
    yield return new WaitForSeconds(cooldown);
    canShoot = true;
}

void Shoot() {
    var helper = new Vector2(firePoint.position.x,
                            firePoint.position.y);
    GameObject bullet = Instantiate(bulletPrefab,
                                    helper,
                                    firePoint.rotation);
    Rigidbody2D rb = bullet.GetComponent<Rigidbody2D>();
    rb.AddForce(firePoint.up * bulletSpeed,
                ForceMode2D.Impulse);
}
```

```
}
```

#### Kôd 4.8 Funkcija za pucanje

Ovaj kod Kôd 4.8 Funkcija za pucanje opisuje pucanje iz puške u 2D Unity igrici. Ako igrač pritisne lijevu tipku miša (`Input.GetButton("Fire1")`), onda se provjerava da li igrač može pucati preko varijable `canShoot`. Ako igrač može pucati, funkcija `Shoot()` se poziva, a nakon toga se pokreće korutina `NextBullet()` pomoću `StartCoroutine` metode.

`NextBullet()` je korutina koja postavlja `canShoot` varijablu na `false`, čime se igraču onemogućuje pucanje dok ne prođe određeni vremenski period definiran s `cooldown` varijablom. Kada vrijeme kašnjenja istekne, `canShoot` varijabla se postavlja na `true`, što omogućuje igraču da ponovno puca.

`Shoot()` funkcija stvara novi projektil `bullet` kloniranjem prefaba `bulletPrefab`. Kao prvi argument u funkciju `Instantiate()` prosljeđuje se pozicija definirane točke oružja `firePoint.position` i kao drugi argument prosljeđuje se rotacija oružja `firePoint.rotation`. Ovo se koristi kako bi se novi klon stvorio na poziciji i orijentaciji oružja.

Zatim se dodaje sila projektilu pomoću `Rigidbody2D` funkcije `AddForce()`. Smjer i brzina projektila određuje se množenjem vektora `firePoint.up` sa skalarnom veličinom `bulletSpeed`, a kao drugi argument u funkciju `AddForce()` prosljeđuje se `ForceMode2D.Impulse` kako bi se postigao trenutačni učinak i kako bi metak bio ispaljen s fizičkom silom koja ga gura prema naprijed.

Mogućnost stvaranja različitih oružja s minimalnim naporom čini ovaj kod Kôd 4.8 Funkcija za pucanje vrlo korisnim za raznovrsnost oružja u igrici. Jednostavno je potrebno postaviti različite vrijednosti za parametre kao što su brzina pucanja, izgled metka i veličina eksplozije, što omogućuje kreiranje oružja koja se osjećaju kao potpuno nova i različita. Korištenjem ovog koda Kôd 4.8 Funkcija za pucanje, razvojni timovi mogu brzo i jednostavno stvoriti veliki broj oružja, prilagoditi ih igračevim potrebama i osigurati dugotrajnu i raznovrsnu igrivost igre.

```
private void OnCollisionEnter2D(Collision2D collision){
    if (collision.gameObject.tag == "Enemy"){
        collision.gameObject.GetComponent<Enemy>().
```



```
        TakeDamage (bulletDamage) ;  
    }}
```

#### Kôd 4.9 Funkcija za koliziju između metka i neprijatelja

Ovih nekoliko linja koda Kôd 4.9 Funkcija za koliziju između metka i neprijatelja predstavlja detekciju sudara između metka i objekta s oznakom neprijatelj (engl. *Enemy*) u 2D prostoru. Kada se metak sudari s objektom s ovom oznakom, metak poziva funkciju `TakeDamage` u skripti `Enemy` objekta s kojim se sudario. Argument `bulletDamage` predstavlja količinu oštećenja koju će metak nanijeti neprijatelju.

### 4.1.9. Interaktivni objekti

Interaktivni objekti u igrici su kutija, eksplozivna bačva i mina, kako je prikazano na slici Slika 4.7 Slika interaktivnih objekata.

Kutija ima mogućnost da baci jedno od tri moguća oružja kada se raznese. Mina eksplodira pri prilasku i nanosi štetu igraču, a također može ozlijediti i neprijatelje ako su dovoljno blizu. Eksplozivna bačva eksplodira pri razbijanju i nanosi štetu svima u blizini, bilo da su igrači ili neprijatelji.



Slika 4.7 Slika interaktivnih objekata

Svi ovi interaktivni objekti su vrlo važni za igrača jer mogu pružiti korisne prednosti u borbi protiv neprijatelja ili pak nanijeti štetu igraču ako se ne koriste na pravi način. Osim toga, različiti interaktivni objekti u igri pomažu u održavanju dinamičnosti igre, čineći je manje predvidljivom i stvarajući novi izazov za igrače.

```
Destroy (gameObject) ;  
var randomNumber = Random.Range (0, 10) ;  
if (gun.Length <= randomNumber) {
```

```

        Instantiate (gun[randomNumber],
                    transform.position,
                    Quaternion.identity);
    }

```

#### Kôd 4.10 Kod za kutije

Kod Kôd 4.10 Kod za kutije iznad služi za uništavanje kutije i davanje igraču nasumično odabranog oružja. Nakon što igrač razbije kutiju, poziva se funkcija `Destroy` koja uništava objekt kutije. Nakon toga, generira se slučajan broj između 0 i 10. Ako duljina polja `gun` nije veća od slučajno generiranog broja, tada se stvara novi oružje na poziciji kutije, koristeći `Instantiate` funkciju. Parametar `Quaternion.identity` označava da ne želimo rotirati objekt, već želimo da ostane uspravan. Dakle, ovaj kod osigurava da igrač dobije nasumično odabrano oružje kada uništi kutiju.

```

    Destroy (gameObject);
    GameObject effect = Instantiate (explosion,
    transform.position, Quaternion.identity);
    effect.transform.localScale = new Vector3 (explosionSize,
    explosionSize, 0);
    Destroy (effect, 0.1f);

```

#### Kôd 4.11 Kod za eksplozivnu bačvu

Gornji kod Kôd 4.11 Kod za eksplozivnu bačvu opisuje eksplozivnu bačvu. Kada bačva primi određenu količinu štete bačva će biti uništena pozivom funkcije `Destroy (gameObject)`, a zatim će se stvoriti vizualni efekt eksplozije u toj poziciji pozivom funkcije `Instantiate (explosion, transform.position, Quaternion.identity)`. Nakon toga, efekt eksplozije će biti skaliran na odgovarajuću veličinu (koja je prethodno definirana u igri) pozivom `effect.transform.localScale = new Vector3 (explosionSize, explosionSize, 0)`. Konačno, pozivom funkcije `Destroy (effect, 0.1f)` će se uništiti efekt eksplozije nakon 0.1 sekunde.

```

    if (collision.gameObject.tag == "Player"){
        Destroy (gameObject);
        GameObject effect = Instantiate (explosion,
        transform.position,
        Quaternion.identity);
        effect.transform.localScale =

```

```
        new Vector3(explosionSize, explosionSize, 0);
        Destroy(effect, 0.1f);
    }
```

#### Kôd 4.12 Kod za eksploziju mine

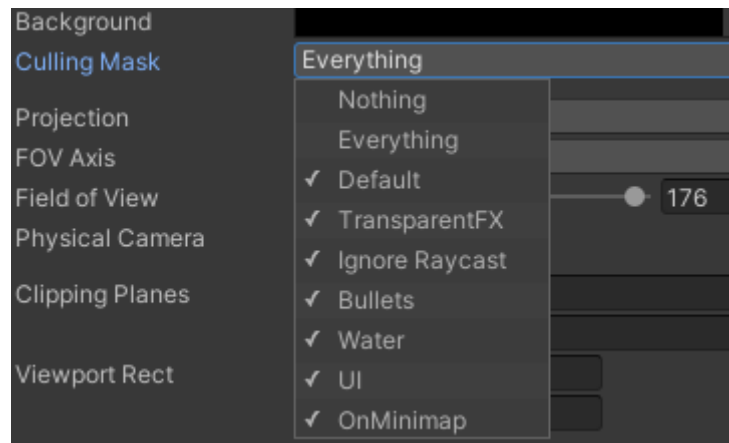
Ovaj kod Kôd 4.12 Kod za eksploziju mine provjerava sudar igrača s minom. Ako se igrač previše približi mini, ona će eksplodirati i uništiti se. Nakon toga, stvara se eksplozije u obliku objekta `explosion` koji se instancira na poziciji mine i skalira na veličinu koju određuje varijabla `explosionSize`. Učinak eksplozije nestaje pomoću funkcije `Destroy` nakon 0.1 sekunde.

### 4.1.10. Mapa

U igrama u kojima igrač mora proći kroz nekoliko razina, može biti korisno prikazati malu mapu koja prikazuje generirani put za igrača da dođe do izlaza. Jedan način da se to postigne u Unityju je korištenjem platno (engl. *Canvas*) i druge kamere koja se postavlja u igru samo za prikaz male mape.

Prvo je potrebno stvoriti novo platno element i dodati mu objekt slike (engl. *Raw Image*). Slika se koristi za prikaz onoga što vidi kamera. Nakon što se objekt slike doda, potrebno je kreirati novu kameru koja će se koristiti za prikaz male mape. Ta kamera će se koristiti samo za prikaz male mape, pa se postavlja na poseban layer.

Nakon što se kamera i objekt slike postave, potrebno je podesiti poziciju i veličinu kamere da se fokusira na područje koje se želi prikazati na maloj mapi. Također, potrebno je postaviti da kamera prikazuje samo objekte koji se nalaze na layeru za malu mapu. To se može postići korištenjem maskiranja kao na slici Slika 4.8 Prikaz maskiranja na objektu kamere. Te se na kraju dobije izgled kao na slici Slika 4.9 Slika male mape unutar igrice.



Slika 4.8 Prikaz maskiranja na objektu kamere



Slika 4.9 Slika male mape unutar igrice

#### 4.1.11. Unity analytics

Unity Analytics je servis unutar Unity platforma koji omogućava programerima da prikupe podatke o igračkom iskustvu tijekom igranja igre. Ova usluga pruža programerima detaljne analize o ponašanju igrača, poput vremena koje igrači provode u igri, razine na kojima igrači najčešće odustaju, najpopularnijih ruta kroz igru i drugih statističkih podataka. U sklopu ovog diplomskog rada, korišten je Unity Analytics servis za prikupljanje podataka o igračkom iskustvu. Analizirani su broj smrti igrača, broj ponovljenih pokušaja, broj pobjeda igrača, vrijeme provedeno za svakog posebnog igrača i svaku posebnu vrstu generiranja razina i zadovoljstvo igrača kada prvi put dovrši određenu razinu generiranja. Cilj ovog pristupa bio je dobiti uvid u to koje metode generiranja razina najbolje funkcioniraju u kontekstu računalnih igara te kako utječu na korisničko iskustvo.

```
Dictionary<string, object> parameters =
    new Dictionary<string, object>()
    {
        { "BP", 0 },
        { "Deaths", 1 },
        { "PP", 1 },
        { "Timee",
```

```

        GameObject.FindGameObjectWithTag("Timer")
            .GetComponent<Counter>().getTime()});
    try{
        AnalyticsService.Instance
            .CustomData("RandomInformation", parameters);
    }
    catch{
        Debug.Log("error while sending");
    }
    AnalyticsService.Instance.Flush();

```

Kôd 4.13 Kod za slanje informacija u Unity analytics servis

Ovaj kod Kôd 4.13 Kod za slanje informacija u Unity analytics servis prikazuje slanje prilagođenih podataka u Unity Analytics servis. Slanje se obavlja putem stvaranja rječnika (engl. *Dictionary*) koji sadrži ključne vrijednosti podataka koji se žele poslati te poziva metode koja će poslati te podatke Unity Analytics servis.

U konkretnom primjeru, stvaraju se četiri ključne vrijednosti podataka: BP (broj pokušaja), Deaths (broj smrti), PP (broj pobjeda) i Timee (vrijeme provedeno u igri). Te vrijednosti se pohranjuju u rječnik i nazivaju `RandomInformation` (u ovom slučaju šalju se informacije o igranju nasumično generirane razine). Zatim se poziva metoda `CustomData` koja prima naziv eventa u analytics servisu i sam rječnik s vrijednostima koje se šalju.

U slučaju neuspjeha slanja podataka, tj. ako se dogodi iznimka, kod će ispisati grešku u konzoli. Na kraju se poziva metoda `Flush` kako bi se osiguralo da su podaci poslani.

## 4.2. Implementacija različitih načina generiranja

U idućem poglavlju, bit će riječi o implementaciji različitih načina generiranja razina u računalnoj igrici. Korištenjem različitih pristupa generiranja razina, moguće je postići različite razine dinamike i izazova u igri, što zauzvrat može povećati njezinu ponovljivost i uzbudljivost. Proceduralno, dinamičko i nasumično generiranje razina su tehnike koje su odabrane za detaljnije istraživanje u sljedećem poglavlju, s naglaskom na njihove prednosti, mane i način implementacije.

## 4.2.1. Proceduralno

Proceduralno generiranje je tehnika u računalnom programiranju koja omogućuje stvaranje sadržaja na temelju algoritama i matematičkih formula (Bycer, 2021). Zadaća proceduralnog algoritma je kreiranje automatsko generiranih razina uz pomoć elemenata igre opisanih u prijašnjim poglavljima (Bycer, 2021). To uključuje raspoređivanje soba, kreiranje protivnika i interaktivnih objekata i glavnih neprijatelja koji omogućuju pobjedu razine.

Ova tehnika se sve više koristi u razvoju računalnih igara kako bi se stvorile dinamičke i nepredvidive razine koje igrači mogu istraživati. Proceduralno generiranje omogućuje izgradnju složenih i raznolikih svjetova koji se prilagođavaju igračevom iskustvu i pružaju neprestano izazovno okruženje. Proceduralni algoritam za generiranje razina ima zadatak stvaranja početka i kraja, izgleda razine, protivnika i interaktivnih objekata u igrici.

```
private void Start() {  
    roomTemplate = GameObject.FindGameObjectWithTag("Rooms")  
        .GetComponent<Rooms>();  
    Invoke("Spawner", 0.1f);  
}  
  
private void Spawner() {  
    if (spawn == false) {  
        switch (spawnerDirection) {  
            case 1: roomNumber = Random.Range(0,  
                roomTemplate.topRoom.Length);  
            Instantiate(roomTemplate.topRoom[roomNumber],  
                transform.position,  
                roomTemplate.topRoom[roomNumber].transform.rotation);  
            case 2: roomNumber = Random.Range(0, roomTemplate.bottomRoom.Length);  
            Instantiate(roomTemplate.bottomRoom[roomNumber],  
                transform.position,  
                roomTemplate.bottomRoom[roomNumber].transform.rotation);  
            case 3: roomNumber = Random.Range(0, roomTemplate.rightRoom.Length);  
            Instantiate(roomTemplate.rightRoom[roomNumber],
```

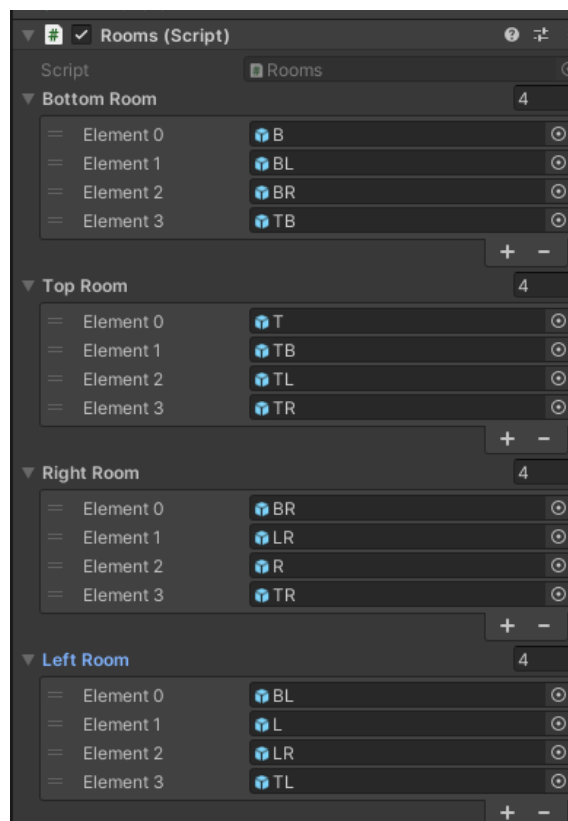
```

        transform.position,
        roomTemplate.rightRoom[roomNumber].transform.rotation);
    case 4:roomNumber = Random.Range(0, roomTemplate.leftRoom.Length);
    Instantiate(roomTemplate.leftRoom[roomNumber],
        transform.position,
        roomTemplate.leftRoom[roomNumber].transform.rotation);
    spawn = true;
}}}

```

#### Kôd 4.14 Fukcija za generiranje razina

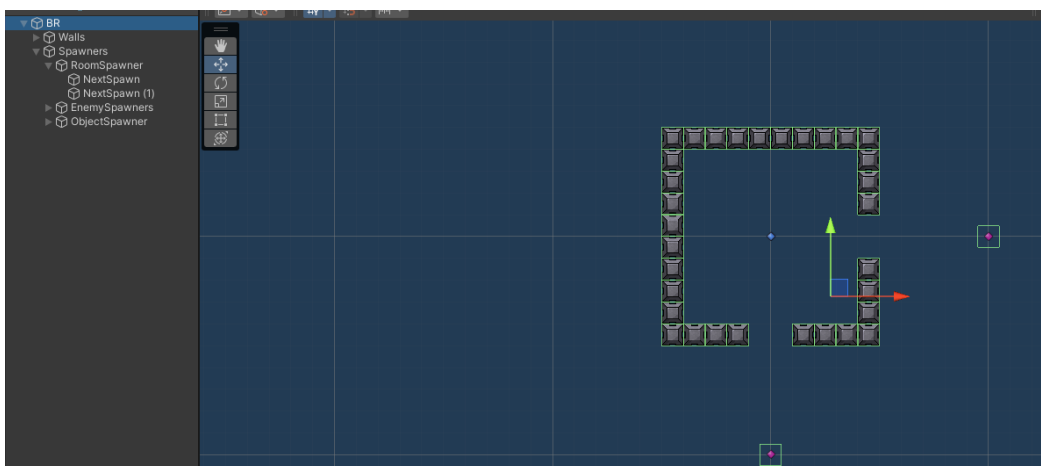
U prethodnom isječku koda Kôd 4.14 Fukcija za generiranje razina prikazan je način proceduralnog generiranja u računalnoj igri. U metodi `Start`, najprije se inicijalizira varijabla `roomTemplate` s mogućim objektima soba za generiranje kao na slici Slika 4.10 Prikaz Rooms skripte, nakon čega se poziva metoda `Spawn` uz pomoć `Invoke` metode koja daje vremensku odgodu pri zvanju metoda.



Slika 4.10 Prikaz Rooms skripte

Kada se pozove metoda `Spawn`, u slučaju da je varijabla `spawn` pozitivna, ništa se neće dogoditi. No, ako je varijabla `spawn` negativna, sobe će se početi generirati prema smjeru označenom varijablom `spawnerDirection`, koja označava prema kojoj strani je soba otvorena. Svaka soba ima određeni broj otvorenih površina, gornju, donju, lijevu ili desnu, koje mogu biti izmiješane na sve moguće načine.

Otvor u sobama ima prazan objekt koji na sebi drži ovu skriptu te se na njemu inicijalizira vrijednost varijable `spawnerDirection` ovisno o strani njegovog otvora. Na primjer, ako je lijeva strana sobe otvorena, na nju će se nadovezati soba koja ima otvor na desnoj strani. Skripta gleda `spawnerDirection` varijablu kako bi odredila koje sobe se trebaju generirati, primjerice, ako je `spawnerDirection` varijabla jednaka broju 1, to znači da je soba otvorena prema dolje i na nju se moraju nadovezati sobe koje imaju otvor prema gore. Nakon generiranja sobe, varijabla `spawn` postavlja se na pozitivnu vrijednost kako se ne bi desilo više generiranja soba odjednom. Na kraju se dobije objekt kao na slici Slika 4.11 Slika objekta sobe gdje `NextSpawn` objekt u sebi sadrži skriptu s gore objašnjenim kodom.



Slika 4.11 Slika objekta sobe

Da bismo osigurali prisutnost neprijatelja i interaktivnih objekata u sobama, koristimo objekte nazvane `EnemySpawner` i `ObjectSpawner` kao na slici Slika 4.11 Slika objekta sobe. Oni na sebi imaju komponente skripte koje služe kako bi bilo osigurano da sobe ne budu prazne.

```
var randomSpawnX = Random.Range(randomPositionOne,  
                                randomPositionTwo);  
var randomSpawnY = Random.Range(randomPositionOne,  
                                randomPositionTwo);  
var randomEnemy = Random.Range(0,
```



```

        enemy.minionsLevelOne.Length);
Instantiate(enemy.minionsLevelOne[randomEnemy],
            new Vector3(transform.position.x + randomSpawnX,
                       transform.position.y + randomSpawnY),
            Quaternion.identity);

```

#### Kôd 4.15 Kod za kreiranje neprijatelja

Ovaj kod Kôd 4.15 Kod za kreiranje neprijatelja služi za stvaranje tri nasumično postavljene neprijatelja unutar sobe. Varijable `randomSpawnX` i `randomSpawnY` određuju nasumične koordinate unutar sobe gdje će se neprijatelji stvoriti, dok varijabla `randomEnemy` odabire nasumičnog neprijatelja iz liste svih mogućih neprijatelja. Konačno, korištenjem metode `Instantiate`, stvaramo neprijatelja na odabranoj lokaciji.

```

var randomSpawnX = Random.Range(randomPositionOne,
                                randomPositionTwo);
var randomSpawnY = Random.Range(randomPositionOne,
                                randomPositionTwo);
var randomObject = Random.Range(0, objects.objects.Length);
Instantiate(objects.objects[randomObject],
            new Vector3(transform.position.x + randomSpawnX,
                       transform.position.y + randomSpawnY),
            Quaternion.identity);

```

#### Kôd 4.16 Kod za kreiranje interaktivnih objekata

Kôd iznad Kôd 4.16 Kod za kreiranje interaktivnih objekata ima funkciju stvaranja dva objekta nasumično odabranih iz liste mogućih. Varijable `randomSpawnX` i `randomSpawnY` služe za određivanje nasumičnih koordinata unutar sobe gdje će se objekt stvoriti, dok varijabla `randomEnemy` služi za odabir nasumičnog objekta iz liste svih mogućih objekata. Konačno, pozivom metode `Instantiate` stvaramo objekt na odabranoj lokaciji. U prvih pola sekunde objekt na sebi ima entitet sudarača (engl. *collider*) kako bi se uništio u slučaju da se stvori na drugom objektu poput neprijatelja.

Jedna od najvećih prednosti proceduralnog generiranja je da omogućava stvaranje ogromne količine sadržaja s relativno malo truda. To je posebno korisno u igrama s otvorenim svijetom ili RPG igrama gdje igrači mogu putovati kroz različite dijelove svijeta i otkrivati nove teritorije. Bez proceduralnog generiranja, stvaranje toliko mnogo sadržaja ručno bi bilo neizvedivo i zahtijevalo bi ogromne količine ljudskog rada. Zbog toga je proceduralno generiranje idealno za manje timove i samostalne developere.

Druga velika prednost proceduralnog generiranja je da svaki igrač dobiva jedinstveno iskustvo. Sadržaj se generira algoritamski. Igrači mogu igrati igru nekoliko puta, a svaki put dobivaju različito iskustvo. To znači da igra može postati znatno dugovječnija i da će igrači imati razloga igrati igru više puta iznova.

Proceduralno generiranje također omogućava razvojnim timovima da lako ažuriraju igru. Ako tim otkrije da je neki dio igre pretežak ili prelagan, može jednostavno izmijeniti algoritam koji generira taj dio igre. To znači da se igra može lako prilagoditi i poboljšati nakon izlaska.

Unatoč tim prednostima, proceduralno generiranje ima svoje mane. Jedna od glavnih mana je nedostatak ljudskog elementa. Algoritmi mogu stvoriti određene vrste sadržaja, ali ponekad je potrebna ljudska kreativnost kako bi se stvorio doista jedinstven i upečatljiv svijet koji će zapanjiti. Ponekad igre generirane proceduralno mogu postati dosadne i predvidljive jer se koriste slični elementi za stvaranje razina, objekata i neprijatelja.

Još jedna mana je da algoritmi mogu stvoriti probleme s izbalansiranošću i težinom igre. Bez ljudskog nadzora, teško je osigurati da će svi dijelovi igre biti izazovni, ali ne preteški za igrača. Ponekad proceduralno generirane igre mogu biti prelagane ili preteške, što može frustrirati igrača i smanjiti ukupnu kvalitetu igre.

## 4.2.2. Dinamičko

Dinamičko generiranje je tehnika koja se koristi u razvoju računalnih igara kako bi se osiguralo da se igrači susreću s novim i izazovnijim izazovima svaki put kad igraju igru. Ova tehnika uključuje generiranje sadržaja u stvarnom vremenu tijekom igre, što igračima omogućuje da se suoče s različitim scenarijima koji se prilagođavaju njihovom napretku i igračkim odlukama. Dinamičko generiranje se sve više koristi u modernim igrama kako bi se osiguralo da igrači imaju jedinstveno iskustvo igranja igre i da se ne dosade nakon više igranja.

Dinamičko generiranje soba se odvija na sličan način kao i proceduralno generiranje, no s jednom bitnom razlikom - kod za kreiranje neprijatelja i interaktivnih objekata je prilagođen kako bi se igraču pružio izazov ili olakšanje ovisno o njegovim sposobnostima.

```
if (PlayerPrefs.GetInt("timesPlayed") /  
    PlayerPrefs.GetInt("Deaths") >= 1)  
if (PlayerPrefs.GetInt("timesPlayed") /
```

```
PlayerPrefs.GetInt("Deaths") >= 1.5)
if (PlayerPrefs.GetInt("timesPlayed") /
PlayerPrefs.GetInt("Deaths") >= 2)
```

#### Kôd 4.17 Kod za dinamičko generiranje

Kod iznad Kôd 4.17 Kod za dinamičko generiranje za kreiranje neprijatelja se razlikuje po tome što u sebi sadrži if upite koji pozivaju spremljene podatke o broju smrti i broju ponovnih pokušaja igre igrača. Po tome se može pružiti izazov ili olakšanje soba ovisno o omjeru smrti i ponovnih pokušaja igrača. Ostatak koda za kreiranje neprijatelja je isti kao i u gornjem kodu Kôd 4.15 Kod za kreiranje neprijatelja. Kod za kreiranje interaktivnih objekata također koristi iste uvjete kao kod iznad te su veće šanse da se kreiraju dobri ili neutralni interaktivni objekti (kutije i eksplozivne bačve), nego negativni interaktivni objekti (mine) ovisno o omjeru smrti i broja pokušaja igrača.

Jedna od najvećih prednosti dinamičkog generiranja razina je nepredvidivost. Svaka igra koja koristi ovaj pristup ima potencijal za stvaranje beskonačnog broja razina, što igračima pruža beskrajno mnogo izazova i avantura. Također, dinamičko generiranje omogućuje igrama da se prilagođavaju igračevom stilu igre, na primjer, kroz povećavanje težine igre ako se igrač dobro snalazi ili kroz smanjenje težine ako se igrač muči s određenim aspektima igre.

Druga prednost dinamičkog generiranja je smanjenje potrebe za ljudskom intervencijom u kreiranju razina. Razvojni timovi ne moraju ručno dizajnirati i izrađivati svaku razinu, što može biti vrlo vremenski i financijski zahtjevno. Dinamičko generiranje može smanjiti ove troškove i ubrzati proces razvoja igre.

Ipak, dinamičko generiranje razina također ima svoje mane. Najveća mana je nedostatak unaprijed osmišljenog dizajna koji bi igri mogao dati jedinstvenu estetsku vrijednost. Dinamički generirane razine su često manje sofisticirane nego one koje su pažljivo dizajnirane i izrađene ručno. Osim toga, proces dinamičkog generiranja može biti nepredvidiv, što može dovesti do razina koje su previše teške ili previše jednostavne, ili imaju drugih nedostataka koji utječu na kvalitetu iskustva igrača.

Još jedan problem je učinkovitost igre. Dinamičko generiranje može zahtijevati velike količine procesorske snage, što može dovesti do loših performansa igre ili čak do nemogućnosti igranja na nekim uređajima.

Konačno, dinamičko generiranje razina u računalnim igrama ima prednosti i mane. Prednosti uključuju nepredvidivost i smanjenje potrebe za ljudskim intervencijom u kreiranju razina, dok su mane nedostatak estetske vrijednosti, nepredvidljivost i problemi s performansama igre. Developerski timovi moraju pažljivo razmotriti prednosti i mane dinamičkog generiranja te donijeti odluku o tome da li je to naj bolji način generiranja za njih i igricu koju razvijaju.

### 4.2.3. Nasumično

Nasumično generiranje u igrama je tehnika koja se koristi za generiranje različitih varijacija sadržaja u igri, poput različitih mapa, neprijatelja ili oružja uz pomoć pre definiranih objekata unutar igrice svaki put kada se ta igra igra (Bycer, 2021). U nasumičnom generiranju, elementi su odabrani nasumično iz zadanog skupa i postavljeni na razini prema određenim pravilima ili algoritmima. Cilj je stvoriti igru koja će uvijek biti drugačija i izazovna svaki put kada se igra. Ovo je koncept koji se često koristi u igrama s otvorenim svijetom, poput Minecrafta, gdje su igrači u mogućnosti istraživati i graditi u svijetu koji je potpuno slučajno generiran. Iako ovo može stvoriti jedinstvenu igračku avanturu, nasumično generiranje ima i svoje prednosti i mane koje će se dotaknuti kasnije.

Proces nasumičnog generiranja soba odvija se na sličan način kao i kod proceduralnog generiranja, no postoji jedna značajna razlika - kod za kreiranje neprijatelja i interaktivnih objekata prilagođen je kako bi se igraču pružila različita i nepredvidiva iskustva.

Kod za kreiranje neprijatelja Kôd 4.15 Kod za kreiranje neprijatelja i interaktivnih objekata Kôd 4.16 Kod za kreiranje interaktivnih objekata funkcionira na sličan način pri stvaranju tih elemenata. Međutim, za razliku od proceduralnog ili dinamičkog načina, nasumično generiranje ima postavljene granice za najveći i najmanji broj mogućih neprijatelja i objekata u sobi. Stoga se sve stvara na temelju slučajnog odabira između postavljenih granica.

Jedna od prednosti nasumičnog generiranja je što igrači nikada neće doživjeti dvije potpuno iste razine, što može stvoriti beskrajno ponavljajuće igračko iskustvo. Ovo je posebno važno u igrama gdje je ponavljanje sadržaja ključno, poput igara s visokim stupnjem ponavljanja poput roguelike igara.

Druga prednost je što nasumično generiranje može stvoriti izazovnije razine. Budući da nema unaprijed postavljenog plana ili strukture, razine se mogu stvoriti s elementima koji

igrača prisiljavaju na improvizaciju i brzo donošenje odluka. Ovo može stvoriti napetu igračku iskustvo za igrače koji uživaju u takvom izazovu.

Međutim, nasumično generiranje također ima svoje mane. Jedan od glavnih problema je nedostatak koherentnosti i smisla u razinama. Budući da su razine generirane potpuno slučajno, moguće je da se dogode situacije u kojima su razine preteške ili nemaju smisla, što može frustrirati igrače.

Također, nasumično generirane razine mogu biti vrlo nepredvidive i neizvjesne. Igrači koji traže strukturu ili cilj u igri možda će se osjećati izgubljeno u svijetu koji se stalno mijenja i nema jasno definirane ciljeve. Ovo može biti posebno problematično u igrama koje imaju priču ili zadatke koji se moraju rješavati, jer se takve elemente može teško integrirati u svijet koji je nasumično generiran.

U konačnici, nasumično generiranje je koncept koji se najbolje koristi u određenim vrstama igara i nije uvijek prikladan za sve vrste igara. Igrači trebaju biti svjesni prednosti i mana nasumično generiranih razina prije nego što se upuste u igru koja se oslanja na ovaj koncept generiranja sadržaja.

### **4.3. Utjecaj na igračko iskustvo**

Proceduralno, dinamičko i nasumično generiranje razina mogu značajno utjecati na igračev doživljaj u igrici.

Proceduralno generiranje razina se koristi kako bi se automatiziralo stvaranje velikog broja različitih razina, što omogućuje igračima jedinstveno iskustvo u svakoj igri. Ovaj način generiranja razina može doprinijeti većem izazovu za igrače, jer se ne mogu osloniti na naučene taktike i strategije, nego moraju biti spremni na nepredvidljivost. U ovom radu je utvrđeno da se igrači pozitivno odazivaju na korištenje proceduralnog generiranja razina u igrici, te da je njegova sposobnost stvaranja različitih izazova u svakoj igri dovoljno privlačna i zanimljiva da zadrži igrače zainteresiranima i motiviranima da pokušaju pobijediti razinu više puta.

Dinamičko generiranje razina se fokusira na prilagodbu razine igre igračevim postignućima, prilagođavajući razinu izazova u skladu s njihovim sposobnostima. Ovo može pomoći igračima da se osjećaju uspješno i motiviraju ih da nastave igrati. Međutim, neki igrači mogu se osjećati razočarani ako se igra čini prelaka ili ako nisu suočeni s dovoljno izazova.

Dinamičko generiranje najzanimljivije igračima, koji su u njemu proveli znatno više vremena, nego u ostalim načinima generiranja razina, a istovremeno je postiglo najbolji omjer pobjeda i poraza igrača u usporedbi s drugim oblicima generiranja razina.

Nasumično generiranje razina se odnosi na stvaranje različitih razina igre bez ikakvog unaprijed određenog obrasca. Ovaj način generiranja razina omogućuje igračima izuzetno nepredvidljiv doživljaj, s novim izazovima u svakoj igri. Međutim, može biti i frustrirajuće ako igrači ne mogu naučiti iz svojih pogrešaka i primijeniti ih u budućnosti. U ovom diplomskom, nasumično generiranje nije bilo popularno među igračima, jer je nepredvidivo i nije bilo u mogućnosti osigurati ujednačenu razinu izazova. Mnogi igrači su odustali od nasumičnog generiranja zbog generiranja prelaganih ili preteških razina, što je umanjilo njihovo zadovoljstvo igrom.

Stoga, različiti načini generiranja razina u igrici mogu utjecati nepredvidivo na igračevo iskustvo i važno je pažljivo odabrati pravi način generiranja za postizanje što boljeg igračkog iskustva.

## 5. Analitika

### 5.1. Metrike i praćenje utjecaja

U ovom diplomskom radu proučavamo kako različite metode generiranja razina u računalnim igrama utječu na igračko iskustvo i uspješnost igre te koristimo Unity Analytics za prikupljanje relevantnih metrika.

Kako bismo utvrdili utjecaj generiranja razina, fokusiramo se na tri načina generiranja razina: nasumično, dinamičko i proceduralno generiranje razina. Nasumično generiranje razina, iako jednostavno za implementaciju, može biti manje učinkovito u stvaranju izazova i kvalitetne igračke dinamike. S druge strane, dinamičko generiranje razina pruža više izazova, ali je zahtjevnije za programiranje. Proceduralno generiranje razina nudi najveću fleksibilnost i kontrolu, no može biti složeno i zahtjevno za implementaciju.

Za prikupljanje metrika, koristimo Unity Analytics servis, koji pruža informacije o broju smrti igrača, broju ponovljenih pokušaja, broju pobjeda igrača, vremenu provedenom za svakog posebnog igrača i svaku posebnu vrstu generiranja razina i zadovoljstvo igrača na kraju pobjede svake pojedine razine. Te informacije omogućuju nam da dobijemo uvid u to kakav utjecaj na igračko iskustvo imaju različiti načini generiranja razina.

Uz to, prikupljanje podataka pomaže nam u razumijevanju različitih problema koji se mogu pojaviti u procesu generiranja razina. Na primjer, može se dogoditi da određena metoda generiranja razina uzrokuje frustraciju kod igrača, što može dovesti do manje uspješne igre. Identificiranje tih problema pomaže nam da razvijemo rješenja koja će poboljšati igračko iskustvo i poboljšati uspješnost igre.

Nakon prikupljanja podataka, analiziramo ih u sljedećem poglavlju kako bismo utvrdili koji način generiranja razina najbolje funkcionira u kontekstu računalnih igara i kako se odražava na korisničko iskustvo. Također proučavamo sklonosti igrača prema određenim metodama generiranja razina.

Za prikupljanje podataka odabrane su metrike:

- Vrijeme provedeno u određenoj razini generiranja
- Broj smrti igrača
- Broj pobjeda igrača

- Broj ponovnih pokušaja
- Ocjena pobjede pojedine razine sa strane igrača

Vrijeme koje igrač provede u određenoj vrsti generiranja razina je jedna od najvažnijih metrika za praćenje u našem istraživanju. To nam pruža dragocjeni uvid u preference igrača i stilove igre koje preferiraju. Na primjer, ako igrač provodi puno vremena igrajući igru s proceduralno generiranim razinama, to bi moglo ukazivati na to da mu se sviđa predvidljivost koju takva vrsta generiranja nudi. Takvi igrači često uživaju u tome što mogu stvoriti strategiju i naučiti kako pobijediti svaku razinu koju igraju, bez nepotrebnih iznenađenja.

S druge strane, ako igrač preferira dinamičko generiranje razina, vjerojatno uživa u tome što igra prilagođava njegovoj razini znanja i vještine. Ova vrsta generiranja omogućuje igraču da uvijek ima novo i raznoliko iskustvo igranja igre, a da pritom ne gubi izazov i uzbuđenje. Takvi igrači vjerojatno će provesti više vremena igrajući igru jer uvijek postoji nešto novo za istražiti i naučiti.

Konačno, igrači koji preferiraju nasumično generiranje razina često vole avanturu i izazov nepoznatog. Oni vole izazov da se suoče s nepredvidivim, a svaka nova razina predstavlja nova uzbuđenja i izazove. Stoga, takvi igrači vjerojatno će provesti puno vremena igrajući igru, jer nikada ne znaju što ih čeka iza sljedećeg kuta.

Osim toga, vrijeme koje igrači provode u određenoj vrsti generiranja razina može nam također pružiti uvid u to koliko su uspješne različite razine generiranja kod igrača. Što s dovoljno velikom količinom ljudi može ukazati na neke od preferenci generiranja razina kod većine populacije.

Broj smrti igrača je važna metrika koja nam omogućuje da pratimo igračko iskustvo i težinu igre. Ovaj podatak nam daje uvid u koliko često igrači umiru tijekom igre, što može ukazivati na preveliku težinu ili loše balansiranje igre. Međutim, broj smrti igrača ne mora uvijek biti negativan pokazatelj - u nekim igrama, poput Geometry dash serijala, smrti su uobičajene i često su dio igračkog iskustva.

Kako bi dobili bolji uvid u igračko iskustvo, pratimo i broj pobjeda igrača. Ovaj podatak nam omogućuje da vidimo koliko je uspješan igrač u savladavanju igre i postizanju ciljeva. Uz to, uspoređujući broj smrti i pobjeda igrača, možemo vidjeti postoji li korelacija između uspješnosti igre i vremena provedenog u njoj.



Važno je napomenuti da broj smrti i pobjeda igrača ne mora uvijek biti jedini pokazatelj igračkog iskustva. Ostale metrike poput vremena provedenog u igri, broja pokušaja, uspješnosti u završavanju posebnih zadataka i zadovoljstva igrača mogu pružiti dodatni uvid u igračko iskustvo.

Broj ponovnih pokušaja je važna metrika koja se koristi kako bi se analizirao utjecaja različitih načina generiranja razina na igračko iskustvo. Ova metrika nam omogućuje da razumijemo koliko je igrača spremno ponoviti određenu razinu kako bi je završili i postigli pobjedu.

Ovo je važno zato što pomaže u procjeni koliko je igračima stalo do napretka u igri i koliko su motivirani da istraju u prelaženju određene razine. Ako se primijeti da igrači često odustaju nakon nekoliko pokušaja, to može ukazivati na neugodan izazov ili pretjeranu težinu igre. S druge strane, ako se primijeti da igrači uporno pokušavaju riješiti razinu, to može ukazivati na zanimljivu igračku dinamiku i izazov koji ih drži motiviranima.

Uz to, analizirajući broj ponovnih pokušaja, možemo utvrditi razlike u preferencijama igrača prema načinu generiranja razina. Na primjer, igrači koji preferiraju nasumično generiranje razina mogu biti manje skloni ponavljanju određene razine, jer žele iskusiti raznolikost koju nudi nasumičnost. S druge strane, igrači koji više vole proceduralno generiranje razina mogu biti skloniji ponavljanju razina, jer žele poboljšati svoje znanje o algoritmima generiranja. Dok igrači koji ponavljaju dinamičko generiranje mogu biti skloni napretku ili ležernijem načinu igranja zbog konstante prilagodbe njihovom načinu igranja.

Na posljetku, ocjena pobjede nasumično generirane razine može biti jedan od najvažnijih pokazatelja svidanja te razine igraču. Ova metrika služi kao izravno pitanje igraču - sviđa li vam se ovaj način generiranja razine ili ne. Ocjena pobjede je često ključna u odlučivanju hoće li igrač nastaviti igrati igru ili će odustati od nje. Ako igrač ima loš dojam nakon završetka razine, to može utjecati na njihovu motivaciju da igraju dalje ili da se vrate u igru u budućnosti.

Ocjena pobjede također može biti korisna za procjenu kvalitete samog generiranja razina. Ako igrači često daju negativne ocjene za proceduralno, dinamičko i nasumično generirane razine, to može ukazivati na probleme u procesu generiranja. Na primjer, možda se razine ne generiraju na način koji pruža zanimljive izazove ili se igračima ne daje dovoljno informacija o tome što se od njih očekuje u toj razini.

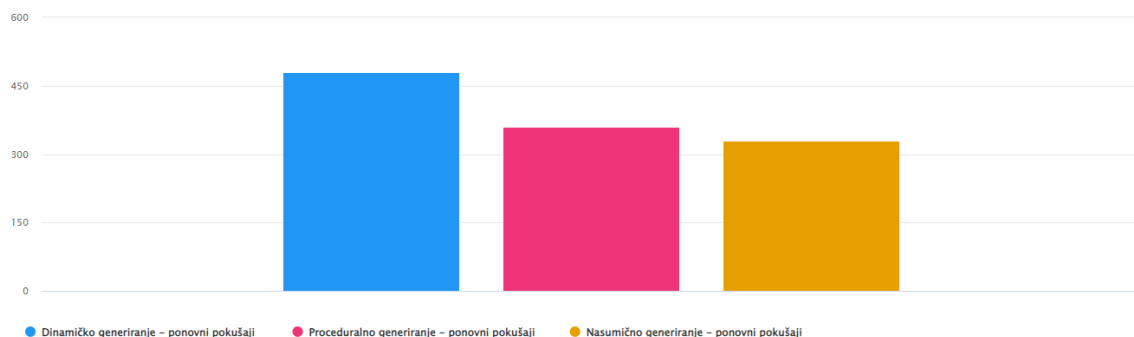
## 5.2. Analiza rezultata

Analiza rezultata igrača ključna je za razvoj i poboljšanje bilo koje igre. Kroz pohranu podataka o smrti, vremenu provedenom, pobjedama, ponovljenim pokušajima igrača putem i zadovoljstvu igrača nakon prve pobjede pojedine razine Unity Analyticsa kao prilagođena polja i event, moguće je dublje razumjeti kako igrači igraju igru i gdje se mogu pojaviti problemi ili prepreke koje ih sprječavaju da uživaju u igri na najbolji način. Ova vrsta analize može biti od velike pomoći u procesu dizajniranja igre, optimizaciji performansi, balansiranju igre i poboljšanju korisničkog iskustva. U ovom dijelu analizirat će se ovi ključni podaci prikupljeni putem Unity Analytics servisa kako bi se dobila detaljna slika o tome kako različite vrste generiranja razina utječu na igračko iskustvo.

Ukupan broj igrača koji su igrali računalnu igru tijekom posljednjih dva mjeseca iznosi 92. To je ključna informacija koju smo dobili kroz praćenje pomoću Unity Analytics servisa. Ovaj servis nam omogućuje da pratimo ponašanje igrača i prikupimo važne podatke o njihovom iskustvu dok igraju igru.

Kroz Unity Analytics, prikupili smo razne statistike i analitike o igračima tijekom igre, kao što su broj smrti, broj ponovljenih pokušaja, broj pobjeda i vrijeme koje su igrači proveli igrajući igru. Te statistike omogućavaju nam da bolje razumijemo ponašanje igrača i da stvorimo bolje igračko iskustvo.

U daljnjem dijelu ovog poglavlja analizirat ćemo ove podatke kako bismo vidjeli kako se različite vrste generiranja razina (nasumična, dinamička i proceduralna) odražavaju na igrače. Također, razmotrit ćemo kako možemo iskoristiti ove informacije da dođemo do zaključka.



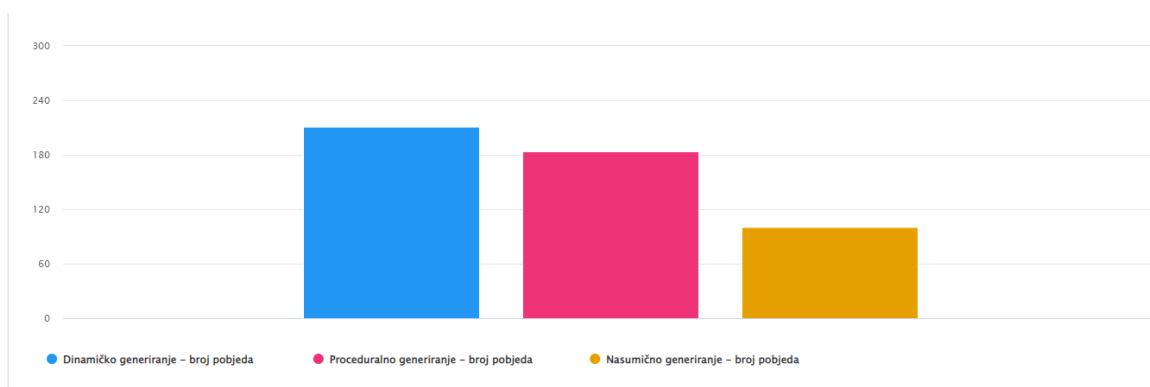
Slika 5.1 Slika grafa ponovnih pokušaja igrača

Slika Slika 5.1 Slika grafa ponovnih pokušaja igrača iznad prikazuje graf koliko su puta igrači pokušali pobijediti određeno generiranu razinu ovisno o njenom načinu generiranja. Po grafu se može vidjeti da su igrači sve tri razine igrali sve ukupno 1197 puta. To jest dinamičko generiranje 492 puta, nasumično generiranje 332 puta i proceduralno generiranje 373 puta. Vremenski duljina igranja svih triju razina je 53 sati (zaokruženo zbog ne relevantnosti minuta i sekunda) što daje prosječnu duljinu jedne razine na 2 minute i 39 sekundi. No kako bi bili što precizniji točna vrijednost utrošenog vremena igrača na pojedinoj razini je:

- Dinamičko: 24h utrošenog vremena
- Proceduralno: 16h utrošenog vremena
- Nasumično: 13h utrošenog vremena

Broj pokušaja pobjede igrača je jedan od najboljih pokazatelja kvalitete dizajna igre i osjećaja koje igrač doživljava dok igra. Analiza podataka pokazuje da se dinamičko generiranje razina istaknulo kao najbolja opcija kada je riječ o poboljšanju ponovne igrivosti, dok su se proceduralno generiranje i nasumično generiranje razina pokazali nešto manje popularni u tom pogledu.

Iako je broj pokušaja igrača važan pokazatelj svidanja razine igračima, trebamo uzeti u obzir i druge metrike kako bismo bolje razumjeli zašto se igračima sviđa određeni način generiranja razina i zašto provode najviše vremena u njemu. Također, važno je saznati zašto im se ne sviđa neki način generiranja razina te zašto provode najmanje vremena u njemu. Odgovori na ta ključna pitanja su za dobivanje cjelovitog uvida u igračko razmišljanje i bolje razumijevanje različitih načina generiranja razina.



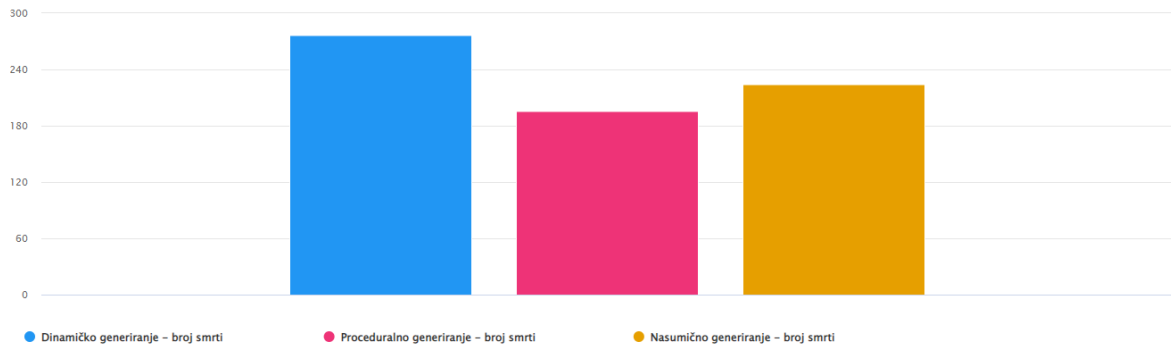
Slika 5.2 Slika grafa broja pobjeda igrača

Prijašnja slika Slika 5.2 Slika grafa broja pobjeda igrača prikazuje graf koji nam daje informaciju o broju pobjeda igrača ovisno o tipu generiranja razina koji su oni odabrali (dinamičko, nasumično i proceduralno generiranje). Po grafu može se vidjeti da su igrači najviše uspjeha imali s dinamičkim generiranjem, dok su imali srednji uspjeha s proceduralnim generiranjem i nisu imali uspjeha na nasumičnom generiranju. Broj pobjeda na pojedinom načinu generiranja razine je:

- Dinamičko: 214 pobjeda
- Nasumično: 104 pobjeda
- Proceduralno: 182 pobjeda

Broj pobjeda igrača je ključni faktor u procjeni igračkog iskustva te nam pruža važan uvid u omjer pobjeda i poraza te utjecaj tog omjera na zadovoljstvo igrača. Analizom grafa vidljivo je da se najveći broj pobjeda dogodio u dinamičkom generiranju razina, dok je drugi najveći broj pobjeda zabilježen u proceduralnom načinu generiranja, dok je najmanji broj pobjeda bio u nasumičnom generiranju razina. Važno je naglasiti da je broj pobjeda bio usko povezan s vremenom utrošenim u određeni način generiranja razine, što ukazuje na to da su igrači ostvarivali više pobjeda u onim načinima generiranja koje su igrali dulje vrijeme.

Međutim, važno je napomenuti da visok broj pobjeda ne garantira i visoku razinu zadovoljstva igrača, te da igrači mogu imati različite preference i ukuse u odabiru načina kreiranja razina. Stoga je potrebno daljnje istraživanje kako bi se saznalo koje su točno karakteristike dinamičkog i proceduralnog generiranja razine koje igračima donose veću razinu zadovoljstva u igri. Također, potrebno je proučiti razloge za manji broj pobjeda u nasumičnom generiranju razine te utvrditi što ga čini manje privlačnim igračima. Samo cjelovito istraživanje može nam pružiti sveobuhvatan uvid u igračko iskustvo i preference u odabiru načina kreiranja razina.



Slika 5.3 Slika grafa broja smrti igrača

Prijašnja slika Slika 5.3 Slika grafa broja smrti igrača nam daje uvid u broj smrti igrača tijekom igranja ovisno o odabranom tipu generiranja razina. Kada se graf iščita možemo vidjeti da su igrači imali naj veći ne uspjeh s nasumičnim generiranjem. Proceduralno i nasumično generiranje su bili podjednaki sa sve ukupno manje od 50 smrti razlike. Broj smrti na pojedinim načinima generiranja razina su:

- Dinamičko: 278 smrti
- Nasumično: 228 smrti
- Proceduralno: 191 smrti

Omjer smrti i pobjeda je važan pokazatelj koji nam pomaže da shvatimo razinu frustracije igrača u igri. Ako je taj omjer loš, igrači se mogu osjećati preopterećeno i izgubljeno te mogu izgubiti interes za igru. Stoga, pri procjeni učinkovitosti različitih načina generiranja razina u računalnim igrama, potrebno je uzeti u obzir i omjer smrti i pobjeda igrača.

Rezultati analize pokazuju da je omjer smrti i pobjeda najbolji za proceduralno generirane razina. Iako je vremenski zaostajalo za dinamičkim generiranjem, igrači su se osjećali zadovoljnijima igrajući ovaj način generiranja. To sugerira da je ravnoteža između izazova i nagrada bila dobro postavljena u ovoj vrsti generiranja. S druge strane, nasumično generiranje razina pokazalo se kao najlošiji izbor s obzirom na to da je omjer smrti i pobjeda bio najgori. Ovi rezultati ukazuju na to da je izazov u igrici važan, ali da igrači ne žele da bude prevelik i frustrirajući.

Kada je riječ o ocjenjivanju razina, igračima je na raspolaganju bio širok raspon opcija. Naime, nakon prve pobjede u svakoj razini, igrači su mogli ocijeniti razinu na skali od 1 do 5, kako bi dobili ne samo analitički, već i igrački povrat. Analiza dobivenih ocjena ukazala je na to da igrači zaista vole ono što igraju, pri čemu su dinamičke razine bile najbolje

ocijenjene. Proceduralne i nasumične razine su se pokazale nešto manje popularnima, pri čemu je prosječna ocjena proceduralnih razina bila za 0.4 bolja od dinamičkih. Ponekad, stvaranje upita je najbolji način za dobivanje preciznijih i korisnijih odgovora na neka pitanja.

Na kraju, uz pomoć grafova, možemo utvrditi da je dinamičko generiranje naj bolje za savladati i da su igrači najviše uživali u njemu zbog uvida u vrijeme te pozitivnih recenzija. Proceduralno generiranje je bilo drugo najzanimljivije i najzabavnije igračima prema utrošenom vremenu. Iako je prosjek smrti i pobjeda bio nešto bolji od dinamičkog generiranja, igrači su i dalje bolje reagirali i pozitivnije ocijenili dinamičko generiranje. Nasumično generiranje je bilo najmanje uspješno s najgorim vremenom utrošenim na igranje, najlošijim omjerom pobjeda i smrti igrača i naj lošijom ocjenom zadovoljstva od igrača.

Ova analiza naglašava važnost kvalitetnog dizajna generiranja razina u računalnim igricama. Takav dizajn ima izrazito velik utjecaj na igračko iskustvo i na način na koji igrači percipiraju igru. Rezultati analize su bili očekivani, gdje je dinamičko generiranje pokazalo najbolje rezultate prilagođavajući se igračevim mogućnostima i postignućima. Proceduralno generiranje, s druge strane, usredotočilo se na opće karakteristike igrača kako bi stvorilo razine na najpravedniji način mogući, što je bilo drugo najbolje rješenje. Naposljetku, nasumično generiranje je stvorilo kaos u kojem igrači nisu mogli imati pozitivno igračko iskustvo jer je bilo ili prejednostavno ili preteško.

Važno je napomenuti da to ne znači da je nasumično generiranje loše te da nije vrijedno implementacije u druge računalne igre. Međutim, ova analiza pokazuje da postoje područja u kojima se može napredovati i da igrači općenito više vole predvidljivost u igrama. Različiti žanrovi igara zahtijevaju različite načine generiranja razina, a nasumično generiranje svakako ima svoje mjesto u nekim igrama. U konačnici, ova analiza nam daje uvid u to kako dizajniranje generiranja razina u računalnim igricama može biti ključno za pružanje najboljeg mogućeg igračkog iskustva.

# Zaključak

Zaključak ovog diplomskog rada je da igre s proceduralno generiranim razinama, dinamičkim i nasumičnim elementima mogu biti vrlo zabavne i zanimljive za igrače. Istraživanjem različitih razina u igri te analizom njihovih karakteristika, mogu se identificirati prednosti i mane svake razine, te se na temelju toga donijeti zaključak o tome koja je razina bolja.

Unity Analytics je bio vrlo koristan alat za prikupljanje podataka o ponašanju igrača u različitim razinama, što je omogućilo detaljnu analizu i razumijevanje njihovih preferencija i reakcija. Analizom podataka prikupljenih kroz Unity Analytics, mogu se identificirati uzorci i trendovi u ponašanju igrača, što može biti korisno za razvoj budućih igara.

Kroz testiranje i evaluaciju ovih različitih načina generiranja uz pomoć Unity Analytics servisa, utvrđeno je da igrači najviše uživaju u igrama s dinamički generiranim razinama. Dinamičko generiranje je bilo najduže i najčešće igrano te je omjer pobjeda i poraza igrača bio u najboljem omjeru, te su korisnici i naj bolje ocijenili tu vrstu generiranja. Proceduralno generiranje je također bilo dobro prihvaćeno među igračima, s nešto manjom omjeru pobjeda i poraza od dinamičkog generiranja i poprilično dobrom ocjenom. Nasumično generiranje je bilo najgori izbor, s najmanjim brojem pokušaja i vremena utrošenog u igranje te s najgorim omjerom smrti i pobjeda i naj lošijom ocjenom od igrača.

Zaključak je da se preporučuje dinamičko generiranje u računalnim igricama kako bi se poboljšalo iskustvo igrača i povećala njihova motivacija za igranje. Uz to, dinamičko generiranje je jedno od lakših generiranja za implementirati i usavršiti. Budući da igrači vole promjene u razinama, preporučuje se korištenje različitih načina generiranja kako bi se osigurala raznolikost i ponovna igrivost. Proceduralno i nasumično generiranje iako drugačijeg rezultata nego dinamičko nisu loši načini generiranja. Naj veći problem kod proceduralnog i nasumičnog generiranja je potreba za usavršavanjem težine razina. Igraču se treba postaviti razina koja mu nije pre laka niti pre teška što dovodi do jako teškog cilja pošto svi igrači nisu isti. Iz tog razloga je dinamičko generiranje uzelo pobjedu jer se adaptira igračevom načinu igranja te mu pomaže kroz cijelo igračko iskustvo tako da mu učini iskustvo težim ili lakšim ovisno o njemu i njegovim potrebama.

Konačno, važno je napomenuti da se igračka iskustva mogu razlikovati ovisno o željama i preferencijama svakog igrača, pa bi daljnje istraživanje o ovoj temi bilo korisno za bolje razumijevanje igračkih preferencija i unaprjeđenje njihovog iskustva u igrama.



## Popis kratica

RPG *Role playing game*

2D *dvodimenzionalno*

3D *trodimenzionalno*

igra s igranjem uloga

dvije dimenzije

tri dimenzije

## Popis slika

Slika 4.1 Slika igrice.....	11
Slika 4.2 Slika inspektora .....	13
Slika 4.3 Prikaz likova iz Futuristic packa .....	14
Slika 4.4 Slika objekta igrača .....	15
Slika 4.5 Prikaz komponenti na objektu igrača .....	16
Slika 4.6 Prikaz promjene igračevog zdravlja .....	19
Slika 4.7 Slika interaktivnih objekata.....	25
Slika 4.8 Prikaz maskiranja na objektu kamere.....	28
Slika 4.9 Slika male mape unutar igrice .....	28
Slika 4.10 Prikaz Rooms skripte .....	31
Slika 4.11 Slika objekta sobe.....	32
Slika 5.1 Slika grafa ponovnih pokušaja igrača .....	42
Slika 5.2 Slika grafa broja pobjeda igrača .....	43
Slika 5.3 Slika grafa broja smrti igrača .....	45

## Popis kôdova

Kôd 4.1 Program za pomicanje igrača.....	16
Kôd 4.2 Program za ciljanje .....	17
Kôd 4.3 Program za brzo kretanje .....	18
Kôd 4.4 Program za zdravlje igrača .....	18
Kôd 4.5 Funkcija za praćenje igrača .....	20
Kôd 4.6 Funkcija za praćenje i pucanje neprijatelja.....	21
Kôd 4.7 Program za praćenje igrača.....	22
Kôd 4.8 Program za pucanje.....	24
Kôd 4.9 Funkcija za koliziju između metka i neprijatelja.....	25
Kôd 4.10 Kod za kutije.....	26
Kôd 4.11 Kod za eksplozivnu bačvu .....	26
Kôd 4.12 Program za eksploziju mine .....	27
Kôd 4.13 Program za slanje informacija u Unity analytics servis.....	29
Kôd 4.14 Program za generiranje razina .....	31
Kôd 4.15 Program za kreiranje neprijatelja.....	33
Kôd 4.16 Program za kreiranje interaktivnih objekata.....	33

## Literatura

- [1] ADAM TULIPER, Unity: Developing your first game with unity and c#, 2014.
- [2] KHONDAKER ZAHIN FUAD, Unity game development guidelines: Build Unity game, 2022
- [3] TANYA X. SHORT, Procedural generation in game design, 2017
- [4] RYAN WATKINS, Procedural content generation for Unity game design, 2016
- [5] STEVE ROBERTS, Character animation fundamentals: developing skills for 2d and 3d character animation, 2011
- [6] ANNY BOSSOM, BEN DUNNING, Video games: An interduction to the industry, 2015
- [7] CLAUDIO SCOLASTICI, Unity 2D game development cookbook, 2015
- [8] JOSHUA BYCER, Game design deep dive: roguelikes, 2021
- [9] PARIS BUTTFIELD-ADDISON, JON MANNING, TIM NUGENT, Unity game development cookbook: essentials for every game, 2021
- [10] SUE BLACMAN, ADAM TULIPER, Learn Unity for windows 10 game development, 2016