Proceduralno generiranje razine za dvodimenzionalnu videoigru žanra platformer pomoću genetskog algoritma

Horvat, Roko

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Roko Horvat

PROCEDURALNO GENERIRANJE RAZINE ZA DVODIMENZIONALNU VIDEOIGRU ŽANRA PLATFORMER POMOĆU GENETSKOG ALGORITMA

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Roko Horvat

Matični broj: 0016156729

Studij: Informacijski i poslovni sustavi

PROCEDURALNO GENERIRANJE RAZINE ZA DVODIMENZIONALNU VIDEOIGRU ŽANRA PLATFORMER POMOĆU GENETSKOG ALGORITMA

ZAVRŠNI RAD

Mentor :

Doc. dr. sc. Bogdan Okreša Đurić

Varaždin, rujan 2024.

Roko Horvat

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj rad usmjeren je na rješavanje problema proceduralnog generiranja razine za dvodimenzionalnu platformer videoigru korištenjem genetskog algoritma. Područje platformerskih videoigara posebno je odabrano zbog dinamičke mehanike koja čini igre izazovnim, dok proceduralno generiranje stvara razine koje su poprilično različite i raznolike. U ovom radu je detaljno opisana povijest i karakteristike platformerskih videoigara te moderna uloga umjetne inteligencije u videoigrama.

Kako bi se razvili dizajni razina temeljeni na unaprijed definiranim kriterijima funkcije korisnosti (eng. *Fitness function*), primijenjen je genetski algoritam koji koristi osnovne principe selekcije (eng. *Selection*), križanja (eng. *Crossover*) i mutacije (eng. *Mutation*). Za implementaciju ovog modela korišteno je Unity razvojno okruženje, gdje su definirani svi ključni elementi razine.

Evaluacije razina koje je sustav generirao temeljile su se na rezultatima testiranja i analizi iskustva igre. Rezultati su pokazali da je razvijeni model sposoban generirati razine u skladu sa postavljenim kriterijima. Razine su u početku izazovne i teške, no tijekom dužeg igranja prilagođavaju se igračkim sposobnostima, sve dok igrač ne uspije doći do cilja.

Rad završava prikazom prednosti i nedostataka razvijenog modela, te predlaže preporuke za moguće unaprjeđenje.

Ključne riječi: proceduralno generiranje, platformer videoigre, genetski algoritam, Unity, umjetna inteligencija, dizajn razina.

Sadržaj

1. Uvod

Proceduralno generiranje razina u videoigrama vrlo brzo je postalo jedan od najutjecajnijih čimbenika u današnjem svijetu razvoja igara. Žanr videoigara platformer jedan je od najpopularnijih, a u njima uživaju sve dobne skupine. Proceduralno generiranje može se koristiti za dobivanje različitih i jedinstvenih razina, a sve to bez trošenja puno vremena na ručno dizajniranje, što veoma olakšava posao razvojnim timovima [1].

Kroz povijest su platformeri imali velik utjecaj na razvoj videoigara. Igre kao što su *Super Mario Bros* [2] i *Sonic the Hedgehog* [3], stvarno su zacementirale ovaj žanr i dale život mnogim drugim naslovima koji su uslijedili nakon njihovog vremena [4]. U današnje vrijeme s napretkom tehnologije, automatsko generiranje sadržaja unutar platformera postaje sve popularnije, baš zbog toga što omogućuje igračima drugačiji izgled i novo iskustvo svaki put kada igraju. [5].

Umjetna inteligencija (eng. *Artificial Inteligence*) prevladava gotovo svim današnjim video igrama, od toga da protivnici uče šablonu kretanja igrača, do stvaranja situacija i složenog sadržaja [6]. Jedna takva tehnika za proceduralno generiranje razina je putem genetskog algoritma - metode koja na neki način oponaša evoluciju živih bića primjenom selekcije, križanja i mutacija za stvaranje optimalnih dizajna razina [7].

Ovaj rad ima za cilj istražiti mogućnosti i prednosti korištenja genetskog algoritma u proceduralnom generiranju razine za dvodimenzionalnu platformer videoigru. Rad se sastoji od osnovnih pojmova platformerskih videoigara s njihovim povijesnim kontekstom, uloge umjetne inteligencije u videoigrama te principa proceduralnog generiranja i genetskih algoritama. Drugi dio rada bavi se praktičnom implementacijom genetskog algoritma za generiranje razine te će detaljno objasniti metode i alate koji se koriste. Treći dio posvećen je evaluaciji rezultata, gdje se procjenjuje učinkovitost generiranog sadržaja testiranjem u igri.

Na kraju se daje zaključak, te se pokazuju prednosti i nedostaci, kao i moguće poboljšanje modela.

1.1. Uvod u platformer videoigre

Jedan od najprepoznatljivijih žanrova videoigara je upravo platformer. Platformer videoigre uglavnom sadrže skakanje, trčanje i izbjegavanje prepreka na različitim razinama. Ono što ih čini posebnima je dinamično kretanje i preciznost koja je prije svega potrebna da se savladaju sve prepreke i neprijatelji na putu. Igrači moraju brzo reagirati, skočiti u pravo vrijeme i paziti na svaki korak ako žele doći do cilja.

Od vrlo jednostavnih 2D platformera s pikseliziranom grafikom, do sada potpuno detaljnih 3D modernih naslova, platformeri su uvijek bili tu da testiraju vještinu i strpljenje igrača. Danas, zahvaljujući naprednoj tehnologiji, dobijaju se raznolike i kreativne razine, dizajnirane da daju potpuno novo iskustvo svaki put kada igrač igra.

Još jedan razlog zašto je platformer tako popularan žanr je njegova pristupačnost. Općenito, platformeri ne sadrže komplicirane kontrole i igrači ne trebaju odvajati veliki vremenski

period na njihovo učenje. Također, mogu biti dovoljno zanimljivi da zadrže igračevu pozornost satima. Osim toga, platformeri su često popraćeni nostalgijom, vraćajući starije igrače u jednostavna vremena prvih igara. Svaka platformer videoigra može biti i vlastiti podžanr. Tako postoje akcijski platformeri s naglaskom na brze reflekse, dok primjerice neki žanrovi uključuju elemente istraživanja i rješavanja zagonetki, dok drugi imaju vlastitu priču koja prati protagonista. [4]

1.1.1. Kratka povijest i karakteristike platformer videoigara

Platformer videoigre potječu još iz ranih 1980-ih, postavši jedan od najbržih žanrova koji je okupio toliko igrača svih dobnih kategorija. Jedna od prvih videoigara koja je popularizirala ovaj žanr bila je legendarna *Donkey Kong* [8] iz 1981. godine, gdje je lik s kojim se upravlja, *Jumpman* - sada poznat kao *Mario*, morao spasiti djevojku od ogromnog gorile koji ju je držao zatočenu. Još značajnije, ova videoigra je uvela razne mehanike koje su definirale platformere: preskakanje prepreka i izbjegavanje mnogih neprijatelja različitih vrsta.

Prava invazija platformerskih videoigara na tržište započela je 1985. godine. *Mario* i *Luigi* [2] postali su ikone igraće (eng. *gaming*) industrije, a njihova videoigra uvela je nove značajke, poput pojačanja koja su igraču davala posebne moći te različitog izgleda razina. Ova videoigra ne samo da je definirala žanr, već je postala pravi kulturni fenomen.

U 1990-ima, platformeri su se nastavili razvijati s novim likovima i svjetovima. Jedan od najpoznatijih primjera je dolazak poznate videoigre *Sonic the Hedgehog* [3] 1991. godine. S vrlo brzom igrom, koja je i bila zaštitni znak franšize, *Sonic* je bio suprotnost *Mariju* na mnogo načina, nudeći različita iskustva igračima unutar istog žanra.

Karakteristike platformerskih videoigara su kombinacija raznih elemenata koji ih čine posebnima. Postoji opcija skakanja i trčanja kroz razine, pri čemu igrači moraju kontrolirati svog lika s preciznošću kako bi izbjegli razne prepreke i neprijatelje. U većini platformerskih videoigara, razine postaju sve teže, te zahtijevaju odgovarajuće vještine igrača kako se videoigra nastavlja. Također, većina platformera ima implementiranu funkcionalnost sakupljanja raznih predmeta, kao što su novčići, zvjezdice, prstenovi ili druga pojačanja koja su korisna za pomoć igraču ili za povećanje konačnog rezultata.

Ono što ljude stvarno privlači u platformerskim videoigrama je kreativnost dizajna razina. Većina razina u ovakvim videoigrama je jedinstvena, ima svoje teme, rasporede i izazove. U većini slučajeva, kreativnost je ključna stavka koja odvaja loše platformere od dobrih. Dobar dizajn razine navodi igrače da prolaze i ponovno igraju videoigru, dok loš dizajn izaziva frustraciju i nezadovoljstvo, te samim time prestanak igranja.

Moderni platformeri vrlo često imaju priče koje vode igrača kroz igru. Priče mogu biti jednostavne poput spašavanja princeze, ili složene s dubokim pričama i razvojem likova. Priče dodaju dodatnu razinu interesa i angažmana u igri, te se kod uspješnih platformera priče produbljuju s raznim nastavcima (eng. *sequel*), pa čak i u obliku filmova ili raznih serija.
[4] [9] [10] [11] [12]

1.2. Umjetna inteligencija u videoigrama

Umjetna inteligencija danas je postala bitan dio videoigara, sa toliko širokom primjenom da se teško može zamisliti videoigra bez njezine prisutnosti. Umjetna inteligencija stvara pametne protivnike, dinamične svjetove i prilagodljiva iskustva koja se mijenjaju prema stilu igranja svakog igrača.

1.2.1. Uloga i različite primjene AI u videoigrama

Jedna od prvih primjena umjetne inteligencije u videoigrama bila je videoigra *Pac-Man* [13], gdje su duhovi slijedili određene algoritme kako bi pronašli igrače. Već su ti jednostavni AI (hrv. *Umjetna inteligencija*, eng. *Artificial Inteligence*, skraćeno *AI*) protivnici pokazali kako osnovni algoritmi mogu učiniti igru izazovnom i zabavnom.

S napretkom videoigara napredovao je i AI. Kreativnost koju današnja umjetna inteligencija može izvesti u igrama varira od simulacije realističnog ponašanja neigračkih likova (eng. *Non-player-character*, skraćeno *NPC*), do prilagođavanja težine videoigre za igrača. Na primjer, u videoigri *The Elder Scrolls V: Skyrim* [14], NPC-jevi imaju rasporede koji reagiraju na prostor oko sebe na takav način da svijet igre čini živim i vjerodostojnim.

Umjetna inteligencija nadalje je primijenjena u proceduralnom stvaranju sadržaja, kao što se vidi u slučaju razina, misija ili čak cijelih svjetova. Igrači bi, dakle, mogli imati beskonačan broj varijacija i iskustava kroz koje tijek igre svaki put postaje drugačiji. Umjetno inteligentne videoigre kao što je *No Man's Sky* [15], sposobne su stvoriti ogromne svemire s milijardama jedinstveno različitih planeta za istraživanje.

Još jedna uzbudljiva primjena umjetne inteligencije u igrama je učenje ponašanja igrača i prilagođavanje taktike protivnika u skladu s tim. U igrama poput *Middle-earth: Shadow of Mordor* [16], neprijatelji se sjećaju susreta s igračem u prošlosti, mijenjaju svoju strategiju i čak izazivaju igrače na temelju prethodnih borbi.

Umjetna inteligencija ne mora uvijek postavljati samo izazove; može i daljnje pripovijedanje. Na primjer, u seriji videoigara *The Sims* [17], AI preuzima kontrolu nad svim ponašanjima i interakcijama između likova u videoigrici kako bi stvorio neočekivane i smiješne situacije koje čine svako igranje drugačijim, te na taj način može pomoći u stvaranju jedinstvenih priča za svakog igrača [6] [18] [19] [20].

1.3. Proceduralno generiranje razina

Proceduralno generiranje razina (skraćeno *PGR*), u dizajnu videoigara, tehnika je u kojoj algoritmi automatski stvaraju sadržaj vrlo raznolike prirode za razine igre. Umjesto da ručno razvijaju svaku pojedinu razinu, programeri definiraju skup pravila i parametara prema kojima algoritam generira sadržaj. Ova tehnika daje mogućnost stvaranja velikog broja jedinstvenih razina koje igračima daju potpuno drugačije iskustvo svaki put kada se igra.

PGR je posebno prikladan za žanrove igara kao što su *roguelike* igre („Igre u kojima se razine najčešće kreiraju nasumično, a smrt lika je trajna posljedica.“ [21]), *sandbox* igre (igre koje većinom nemaju cilj, već daju igračima velik stupanj kreativnosti za interakciju) ili beskrajni trkač (eng. *Endless runner*, igre u kojima je cilj prijeći što veću udaljenost bez umiranja, često popraćene s pojačanjima koja olakšavaju igraču) gdje je varijabilnost razina ključna komponenta dugovječnosti igre. Domene primjene PGR-a kreću se od jednostavnih algoritama koji nasumično postavljaju prepreke i neprijatelje, do složenijih sustava koji stvaraju svjetove s vlastitim ekosustavom i pričom.

1.3.1. Koncepti i prednosti proceduralne generacije razina

Vjerojatno najzanimljiviji koncept u razvoju igara je proceduralno generiranje - oslanjanje na algoritme za stvaranje sadržaja igre. Sve što programeri rade je definiranje pravila, kao što su veličina razina, vrste prepreka i raspored neprijatelja, a zatim algoritam kreira razine. Ovaj pristup donosi razne ključne prednosti.

Jedna od najvećih prednosti PGR-a sigurno je njegova sposobnost generiranja beskonačnih razina, s time da je svaka različita. To znači da svaki put kad igrač pokrene igru, može imati novo iskustvo. To je razlog koji će uvijek videoigru učiniti zanimljivom, te ono što privlači igrača da joj se uvijek iznova vraća.

Uz to, štedi se na vremenu i na resursima. Dizajn razina oduzima puno vremena i obično se izrađuje ručno, što još više vrijedi za igre koje sadrže velik broj razina. PGR može brzo generirati mnogo različitih razina; dakle, ovo smanjuje potrebu za velikim timovima dizajnera i štedi puno vremena.

Budući da je svaka razina različita, algoritam uglavnom daje najneočekivanije i najkreativnije rezultate kojih se dizajneri sami ne bi mogli sjetiti, te se tako razvijaju inovativni i uzbudljivi dizajni koji obogaćuju igru.

Također, algoritam može lako promijeniti stupanj težine i složenosti u skladu s vještinom igrača. Na primjer, generirat će lakše razine za početnike i teže razine za igrače s iskustvom, pružajući tako optimalno iskustvo za svakog igrača.

Proceduralno generiranje razina nalazi se u mnogim poznatim igrama. U *Minecraftu* [22], primjerice, proceduralno generiranje svjetova omogućuje igračima da istražuju beskrajne varijacije krajolika. Ovaj primjer jasno ilustrira kako PGR može povećati broj igrača i učiniti videoigru veoma zanimljivom, do mjere gdje videoigra koja je izašla na tržište 2011. godine je i dan danas jedna od najigranijih videoigara u svijetu [1] [5] [23].

1.4. Genetski algoritmi

Genetski algoritmi su heuristička optimizacijska metoda koja se temelji na procesu prirodne selekcije i genetskim mehanizmima evolucije. Osnovni principi koji se koriste za dobivanje optimalnog ili gotovo optimalnog rješenja danog problema u ovim algoritmima su selekcija (eng. *Selection*), križanje (eng. *Crossover*) i mutacija (eng. *Mutation*). Genetski algoritmi

posebno su korisni u problemima gdje tradicionalne metode optimizacije nisu učinkovite ili su neprimjenjive zbog velike složenosti prostora pretraživanja. [7]

Genetski algoritmi fukcioniraju na temelju evolucije, odnosno populacija rješenja kandidata evoluira kroz određeno vrijeme. Svaki pojedinac u populaciji predstavlja jedno moguće rješenje, te mu se dodjeljuje vrijednost pomoću funkcije korisnosti (eng. *Fitness function*) koja je povezana s problemom. Proces evolucije se ponavlja kroz nekoliko generacija, pri čemu se odabiru najbolje jedinke za reprodukciju (križanje), a određena svojstva mutiraju kako bi se osigurala raznolikost u stvaranju potomstva kombiniranjem gena svakog roditelja. [7]

1.4.1. Osnovni principi i terminologija genetskih algoritama

Osnove i terminologija genetskih algoritama su sljedeći:

- **Populacija** - Populacija je pojam koji se dodjeljuje skupu potencijalnih rješenja koja se razvijaju kroz vrijeme. Svaki član te populacije naziva se jednika ili kromosom [7] [24].
- **Kromosom** - Kromosom je način na koji je predstavljeno moguće rješenje određenog problema. U globalu, kromosomi su prvenstveno nizovi bitova, brojeva ili drugih vrsta objekata [7] [24].
- **Gen** - Ovo je temeljna jedinica informacija koja je sadržana u kromosomu. Svaki gen prikazuje neki aspekt rješenja [7] [24] [25].
- **Funkcija korisnosti** - funkcija koja uzima svaki pojedinačni kromosom iz populacije, te za povrat vraća vrijednost koja pokazuje koliko je svaki kromosom vrijedan. Funkcija korisnosti zapravo prikazuje koliko je određeno rješenje blizu ili daleko od optimalnog rješenja [7] [24] [25].
- **Selekcija** - Ovo je proces odabira najboljih jedinki u trenutnoj populaciji te njihova reprodukcija. Često primjenjivana tehnika za selekciju je rulet selekcija (eng. *Roulette selection*). Rulet selekcija je potpuno slučajna, s time da jedinke koje imaju veću vrijednost funkcije korisnosti će zauzimati veći postotak ruleta, a samim time će imati i veću šansu da budu odabrane [7] [24] [25] [26].
- **Križanje** - proces u kojem se kombiniraju geni dvaju roditeljskih kromosoma kako bi se dobilo potomstvo. Križanje osigurava da se korisna svojstva prenose s roditelja na potomstvo [7] [25].
- **Mutacija** - Nasumična promjena vrijednosti gena kromosoma potomka. Ovo pomaže da se unese određena raznolikost u populaciju kako se ona ne bi prerano približila optimalnom rješenju [7] [25].
- **Generacija** - Termin koji se odnosi na jedan ciklus selekcije, križanja i mutacije. Ponavljanje nekoliko generacija je zapravo evolucija, a ona traje sve dok se ne pronađe zadovoljavajuće rješenje ili dok se ne postigne maksimalan broj generacija [7] [24] [25].

1.5. Ciljevi i metodologija rada

1.5.1. Kratki pregled istraživanja i ciljevi rada

Ovaj se rad bavi implementacijom genetskih algoritama u razvoju proceduralno generirane razine za dvodimenzionalnu platformer videoigru. Glavni cilj ovog rada je razviti model koji koristi genetski algoritam kako bi se došlo do razina koje su zanimljive, različite i izazovne za igranje. Ostali ciljevi ovog rada uključuju razumijevanje osnovnih principa i prednosti povezanih s generiranjem proceduralnih razina, razradu uloge i primjene umjetne inteligencije u videoigrama s genetskim algoritmima te razvoj i implementaciju genetskih algoritamskih modela za proceduralno generiranje razina u Unity razvojnom okruženju. Također, rad nastoji evaluirati generirane razine kroz igračka testiranja i analizu igračkog iskustva te identificirati prednosti i slabosti razvijenog modela te moguća poboljšanja.

Metodologija ovog rada podijeljena je u dva dijela - teorijski i praktični. Teorijski dio rada uključuje proučavanje relevantnih izvora i postojećih metoda generiranja proceduralnih razina i genetskih algoritama. U praktičnom dijelu razvijen je algoritamski model, odnosno genetski algoritam, koji je izrađen u Unity razvojnom okuženju. Testiranja za generirane razine provode se zajedno s igračima, a analize se rade kako bi se utvrdila učinkovitost i kvaliteta generiranog sadržaja.

2. Teorijski dio

U ovom dijelu rada opisuju se različite tehnike korištenja umjetne inteligencije u videoigrama.

2.1. Metode umjetne inteligencije u videoigrama

Područje primjene umjetne inteligencije u videoigrama može varirati od vrlo jednostavnih algoritama sve do složenih sustava koji oponašaju ljudsko ponašanje. Među mnoštvom tehnika umjetne inteligencije, postoje neke koje stvaraju protivnike, simuliraju okruženje ili prilagođavaju igru prema stilu igranja igrača.

2.1.1. Pronalaženje puta (eng. *Pathfinding*)

Pronalaženje puta (eng. *Pathfinding*) jedna je od najvažnijih primijenjenih tehnika umjetne inteligencije u videoigrama. Ona omogućava igraćim i ne igraćim likovima (NPC) da se inteligentno kreću virtualnim svjetovima. Vjerojatno najpoznatiji algoritam za pretraživanje je takozvani A* (eng. *A-star*, skraćeno *A**) algoritam, koji koristi heurističku funkciju za pronalaženje najkraćeg puta od početne do krajnje točke. Uzimajući u obzir stvarnu udaljenost već prijeđenog puta i procjenu preostale udaljenosti do odredišta, A* algoritam pronalazi optimalni put [27] [28]. Osim u videoigrama, A* algoritam koristi se i u raznim navigacijskim sustavima.

Osim A* algoritma, postoje i implementacije *Dijkstrinog* algoritma i *Floyd-Warshall-ovog* algoritma. Iako je *Dijkstrin* algoritam osnova za mnoge varijante algoritama za pronalaženje najkraćeg puta, on doista nastoji pronaći najkraće staze u grafovima s ne-negativnim težinama rubova [29]. Floyd-Warshall algoritam se koristi za pronalaženje najkraćih puteva između svih parova vrhova u grafu. [30]

2.1.1.1. A-star algoritam

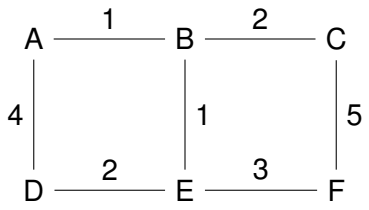
A* algoritam koristi sljedeću funkciju za procjenu ukupnog troška puta [28]:

$$f(n) = g(n) + h(n)$$

gdje je $f(n)$ ukupni procijenjeni trošak čvora n, $g(n)$ trošak puta od početnog čvora do čvora n, a $h(n)$ heuristički procijenjen trošak puta od čvora n do ciljnog čvora [28].

Heuristička funkcija $h(n)$ mora biti optimistična, što znači da nikada ne smije precijeniti stvarni trošak do cilja [27] [28].

Za bolju vizualizaciju, prikazuje se jednostavan primjer s grafom. Pretpostavimo da imamo sljedeći graf s čvorovima i troškovima:



Pretpostavimo da su heuristički troškovi do cilja F:

$$h(A) = 0$$
$$h(B) = 4$$
$$h(C) = 8$$
$$h(D) = 5$$
$$h(E) = 3$$
$$h(F) = 0$$

A* algoritam izračunava put od A do svih povezanih čvorova koristeći formulu $f(n) = g(n) + h(n)$, a zatim po dobivenim rezultatima traži optimalan put do ciljanog čvora.

Da bi algoritam bio jasniji, izračunaje se trošak puta i heuristički trošak između svih povezanih čvorova:

$$f(A) \rightarrow f(B) = 1 + 4 = 5$$
$$f(A) \rightarrow f(D) = 4 + 5 = 9$$

$$f(A) \rightarrow f(B) \rightarrow f(C) = (1 + 2) + 8 = 11$$
$$f(A) \rightarrow f(B) \rightarrow f(E) = (1 + 1) + 3 = 5$$
$$f(A) - > \rightarrow f(D) \rightarrow f(E) = (4 + 2) + 3 = 9$$

$$f(A) \rightarrow f(B) \rightarrow f(C) \rightarrow f(F) = (1 + 2 + 5) + 0 = 8$$
$$\mathbf{f(A) \rightarrow f(B) \rightarrow f(E) \rightarrow f(F) = (1 + 1 + 3) + 0 = 5}$$
$$f(A) \rightarrow f(D) \rightarrow f(E) \rightarrow f(F) = (4 + 2 + 3) + 0 = 9$$

Objašnjenje: Nakon izračuna troškova puta i svih heurističkih troškova, dolazi se do rezultata. Čvor A ima putanju do dva čvora, čvora B i D. Kao što se vidi u izračunu funkcije, veza $f(A) \rightarrow f(B) = 5$ je jeftinija solucija nego $f(A) \rightarrow f(D) = 9$, te je ovdje bolje rješenje veza $A \rightarrow B$. Da nema više veza, ovo bi bilo konačno rješenje, ali pošto je graf složeniji, algoritam nastavlja dalje u potrazi za ciljanim čvorom i optimalnim rješenjem. Sljedeće veze nastavljaju do čvorova C i E preko veza $A \rightarrow B$ i $A \rightarrow D$, te se može vidjeti da optimalni put preko čvora C iznosi 11, a preko čvora E iznosi 5, no to i dalje nije rješenje te ciljani čvor još nije pronađen. Naposlijetku, dolazi se do ciljanog čvora F. Nakon izračuna troškova, dolazi se do rezultata da je optimalni put $A \rightarrow B \rightarrow E \rightarrow F$, te on iznosi

$$f(A) \rightarrow f(B) \rightarrow f(E) \rightarrow f(F) = (1 + 1 + 3) + 0 = 5$$

te je ovo konačno rješenje.

2.1.2. Strojno učenje (eng. *Machine Learning*)

Strojno učenje omogućuje likovima u videoigri da uče iz iskustava i prilagode svoje ponašanje okolini ili igraču. Protivnici u videoigri sposobni su učiti i prilagođavati se strategijama igrača, a mogu se stvoriti baš pomoću algoritma strojnog učenja, posebice neuronskih mreža i poticanog učenja (eng. *Reinforcement learning*) [31]. Dok neuronske mreže mogu modelirati vrlo složene obrasce i ponašanja, poticano učenje omogućuje agentima učenje optimalnih strategija kroz interakciju s okolinom.

Primjerice, AI protivnik u videoigri uči iz igračevih prethodnih susreta i mijenja svoje ponašanje, taktike ili pokrete u skladu s tim. Na taj način moguće je modelirati dinamične i izazovne protivnike koji se mogu prilagoditi stilu igre svakog pojedinog igrača. S druge strane, strojno učenje može se koristiti za generiranje realistične animacije likova, simulaciju fizičkih sustava ili optimizaciju mehanike igre [32].

2.1.3. Ostale metode umjetne inteligencije (kratki pregled)

Druge tehnike umjetne inteligencije koje se koriste u videoigrama uključuju sustave temeljene na pravilima (eng. *Rule-based systems*), simulacije gomile (eng. *Crowd simulations*) te proceduralno generiranje sadržaja. Sustavi koji se temelje na pravilima funkcioniraju korištenjem unaprijed definiranih pravila pri donošenju odluka, što omogućava laku implementaciju, ali rezultira prilično predvidivim ponašanjem [33]. Simulacija gomile koristi se za postizanje realističnih ponašanja velikih grupa likova, poput simulacije gomile ljudi koji se kreću virtualnim gradom ili grupe neprijatelja koji napadaju igrača [34]. Metodologija proceduralnog generiranja sadržaja koristi algoritme za automatsko generiranje razina, likova i drugih elemenata igre. Kao što je već navedeno, pomoću ove tehnike može se generirati beskonačnost različitih razina koje igrač može istraživati, čime se povećava vrijednost ponavljanja bilo koje igre [1].

2.2. Proceduralno generiranje razina

Proceduralno generiranje razina definira se kao tehnika u videoigrama u kojoj algoritmi generiraju sadržaj, te ne koristite ručne tehnike dizajna. PGR pruža mnoge prednosti poput skraćivanja vremena razvoja i pružanja nesputanog broja beskrajno novih iskustava za igrače. Također, pristup pomoću kojeg se mogu generirati jedinstvene razine, likovi, predmeti i mnogi drugi elementi za igru važan je dio proceduralnog generiranja [1].

2.2.1. Tehnike proceduralnog generiranja razina

Postoje mnoga pravila kojih se treba pridržavati prilikom implementacije PGR-a, a svako od njih ima specifičnu prednost i područje najbolje primjene. U narednim točkama istaknute su tri metode – gramatska metoda, metoda postavljanja šablone i evolucijski algoritmi.

2.2.1.1. Gramatske metode

Gramatske metode koriste formalne gramatike za generiranje sadržaja prema zadanim pravilima. Ove metode temelje se na pravilu transkripcije, odnosno opisuju kako se različiti osnovni elementi trebaju kombinirati kako bi se napravile složenije strukture. Gramatske metode omogućuju generiranje složenih razina velike raznolikosti koristeći jednostavna pravila [35].

Jedan takav primjer su L-sustavi (eng. *Lindenmayer systems*), koji su oblik formalne gramatike za generiranje fraktalnih struktura [36] i raznih biljnih oblika [37]. U videoigrama, L-sustavi mogu se koristiti za generiranje organskih razina, poput stabala, grmova i raznih biljaka. Također, pokazali su se korisnima prilikom generacije razina za igre gdje su razine nasumično generirane tamnice ili labirinti.

2.2.1.2. Postavljanje šablona (eng. *Tile-based methods*)

Metoda postavljanja šablone tehnika je koja generira sadržaj koristeći već unaprijed dizajnirane šablone (segmente) koje se kombiniraju radi stvaranja većih razina. Ove metode su vrlo popularne jer su jednostavne za korištenje i veoma fleksibilne. Segmenti se mogu jednostavno prilagoditi i zamijeniti kako bi se povećala varijabilnost razina [38].

Često citiran primjer gdje je implementiran pristup postavljanja šablona je videoigra *Rogue*, koja koristi nasumično generirane predloške za dizajniranje razina tamnice [39].

2.2.1.3. Evolucijski algoritmi

Evolucijski algoritmi su metoda koja generira sadržaj procesom prirodne selekcije i evolucije, a njoj pripadaju i genetski algoritmi. Općenito, takvi algoritmi stvaraju populaciju mogućih rješenja te primjenjuju svojstva selekcije, križanja i mutacije u razvoju dizajna razina prema unaprijed određenim kriterijima [5].

Genetski algoritmi vrlo su korisni kada razine zahtijevaju ispunjavanje određenih uvjeta ili optimizacijskih ciljeva. Kao što je već rečeno, genetski algoritmi mogu generirati razine usmjerene prema određenoj težini ili stilu igranja igrača [7].

2.2.2. Prednosti i nedostaci proceduralnog generiranja razina

Proceduralno generiranje razina u globalu prati više prednosti nego nedostataka. Prije primjene PGR-a, uz prednosti moraju se uvažiti i nedostaci, kako bi se za razvoj videoigre odabrala najbolja metoda.

Prednosti

- **Raznolikost videoigre i ponovno igranje** - Budući da PGR može generirati beskonačan broj različitih varijanti razine, povećava se i vjerojatnost ponovnog igranja. Najvažniji dio ove prednosti je mogućnost prelaženja novih i različitih razina svaki put kada igrači zaigraju videoigru [1].
- **Ušteda vremena i resursa** – Ručni dizajn razina razvojnim timovima oduzima puno vremena i poprilično je skup. PGR uveliko smanjuje troškove te omogućava automatsko generiranje razina, čime se smanjuje količina uključenog ručnog rada i povećava brzina razvoja [5].
- **Prilagodljivost** - Proceduralno generiranje omogućuje prilagodbu razina vještinama i preferencijama igrača. Primjerice, težina razine može se automatski prilagoditi igračevom stilu igranja [40].
- **Kreativnost** – Proceduralnim generiranjem vrlo je lako generirati neočekivane i kreativne dizajne koje ručni dizajn možda nije mogao ni zamisliti [35].

Nedostaci

- **Nedostatak kontrole** - Osnovni nedostaci PGR-a su gubitak kontrole nad točnim dizajnom razine. Ručno dizajnirane razine mogu se fino podesiti prema određenim ciljevima, dok s druge strane proceduralno generirane razine mogu odstupati od namjera dizajnera [1].
- **Tehnička složenost** - Tehnička složenost implementacije sustava za proceduralno generiranje je velika i zahtijeva specifično poznavanje algoritama umjetne inteligencije i programiranja, što bi moglo predstavljati izazov manje iskusnim razvojnim timovima [35].
- **Kvaliteta sadržaja** - Proceduralno generirane razine uvijek će biti različite kvalitete. Neke razine mogu biti iznimno dobro dizajnirane, dok neke druge mogu biti neuravnotežene i dosadne [5].

2.3. Genetski algoritmi

Genetski algoritmi (GA) su tehnike inspirirane prirodnom selekcijom i evolucijom. Za proceduralno generiranje razina općenito, GA se koriste za automatsko generiranje razina prema određenim postavljenim kriterijima ili ciljevima. Ovi algoritmi stvaraju populaciju mogućih rješenja, a zatim koriste principe kao što su selekcija, križanje i mutacija za razvoj dizajna, NPC-jeva i drugih funkcionalnosti [7].

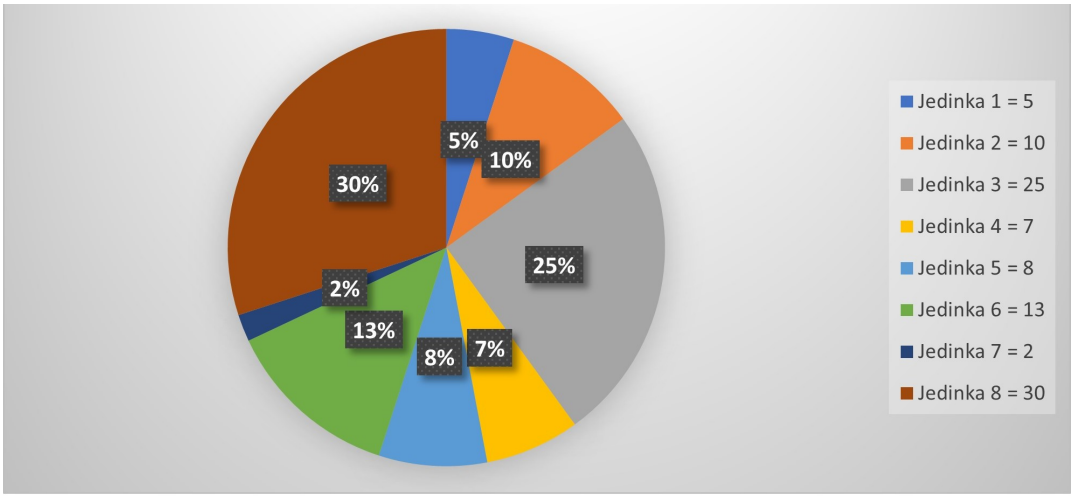
2.3.1. Temeljni koncepti genetskih algoritama

Neki od osnovnih principa koje genetski algoritmi koriste za razvoj rješenja navedeni su u nastavku.

2.3.1.1. Selekcija

Selekcija je proces kojim se jedinke u promatranoj populaciji biraju za sudjelovanje u stvaranju nove generacije. Bolji pojedinci imaju veće šanse za reprodukciju, a to omogućava rast kvalitete populacije s vremenom. Neke od najpoznatijih metoda selekcije su:

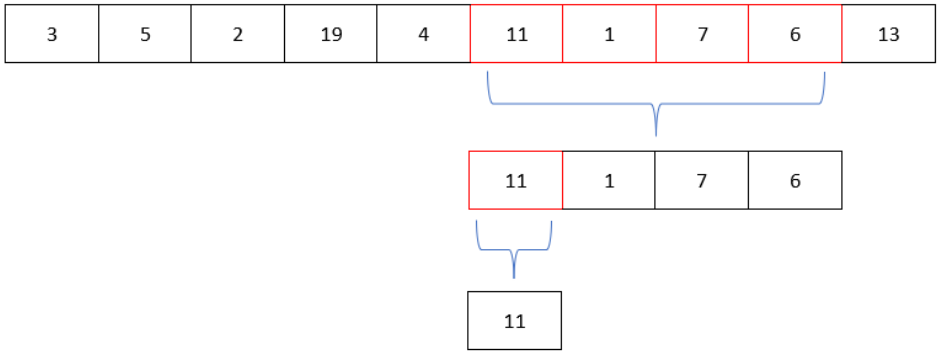
1. Rulet selekcija (eng. *Roulette selection*) – Korištenjem ove metode, oponaša se kotač ruleta, čiji su segmenti razmjerni vrijednosti funkcije korisnosti svakog pojedinaca. Što je veća vrijednost pojedinca, veći je i segment, a time i vjerojatnost da će biti odabran. Ovaj se postupak ponavlja sve dok se ne odabere potreban broj jedinki [7] [25]. Na slici 1 prikazan je primjer rulet selekcije sa osam jedinki. Vrijednost funkcije korisnosti svake jedinke navedena je s desne strane slike.



Slika 1: Primjer rulet selekcije sa osam jedinki

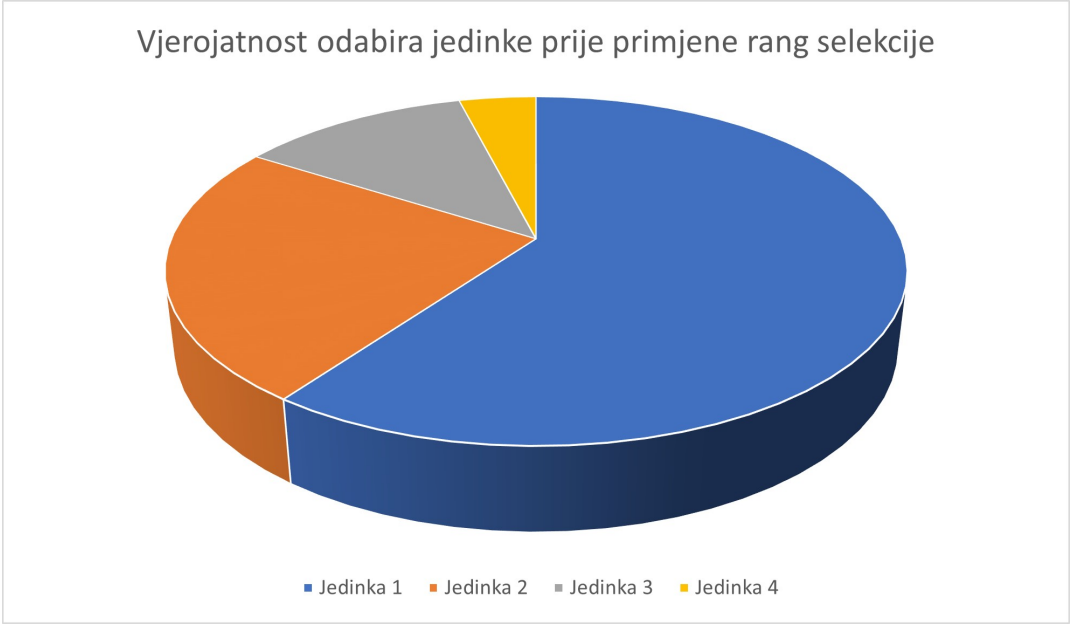
2. Turnir selekcija (eng. *Tournament selection*) – Turnir selekcija funkcionira na način da se mali skup jedinki nasumično odabere iz populacije, nakon čega se odabire najbolja

jedinka tog podskupa za reprodukciju. Veličina podskupa često može varirati, ali obično se koriste mali podskupovi koji osiguravaju kompetitivnost. Ova vrsta selekcije pokazala se dosta efikasnom i jednostavnom za implementaciju [7] [25]. Na slici 2 prikazana je turnir selekcija sa deset jedinki, zajedno sa njihovim fitness vrijednostima.

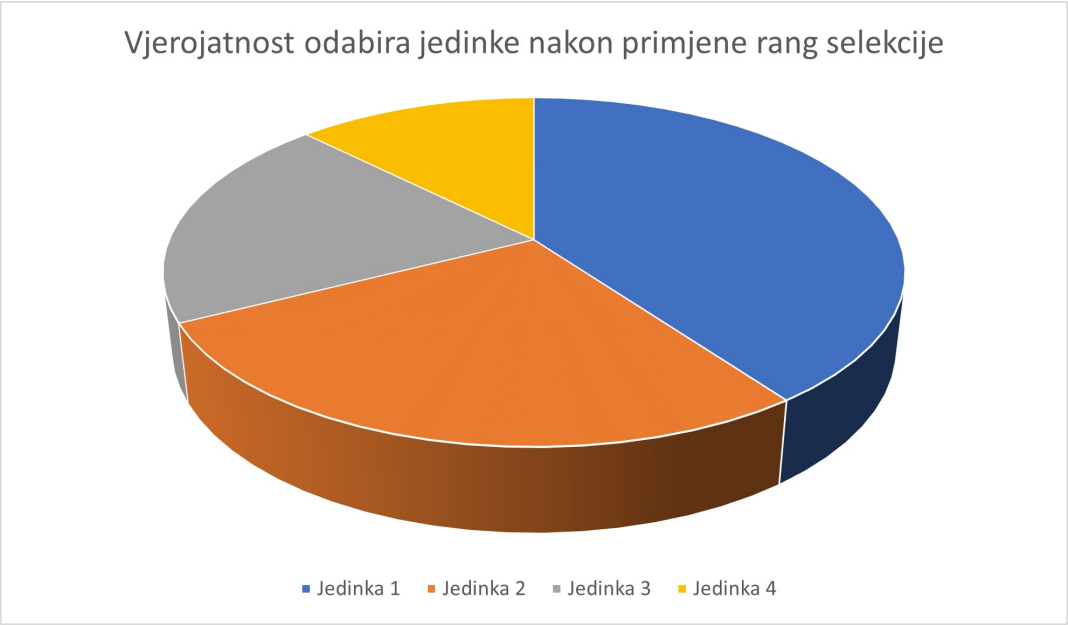


Slika 2: Primjer turnir selekcije sa deset jedinki

3. Rang selekcija (eng. *Rang selection*) - Prema ovom principu selekcije, jedinka je rangirana, a vjerojatnost da će jedinka biti odabrana za reprodukciju proporcionalna je njenom rangu, a ne apsolutnoj vrijednosti funkcije korisnosti. Ovaj pristup smanjuje vjerojatnost da jedna jedinka dominira populacijom, što je ponekad problem kod rulet selekcije [7] [25]. Na slici 3 prikazan je primjer populacije prije primjene rang selekcije, a na slici 4 nakon primjene rang selekcije.



Slika 3: Primjer populacije prije primjene rang selekcije



Slika 4: Primjer populacije nakon primjene rang selekcije

2.3.1.2. Križanje

Križanje je kombinacija genetskog materijala dviju jedinki, odnosno roditelja, kako bi se dobila nova jedinka s različitim genetskim materijalom. Cilj križanja je istražiti nova područja rješenja i povezati osobine roditelja koje su korisne. Neki od načina križanja su:

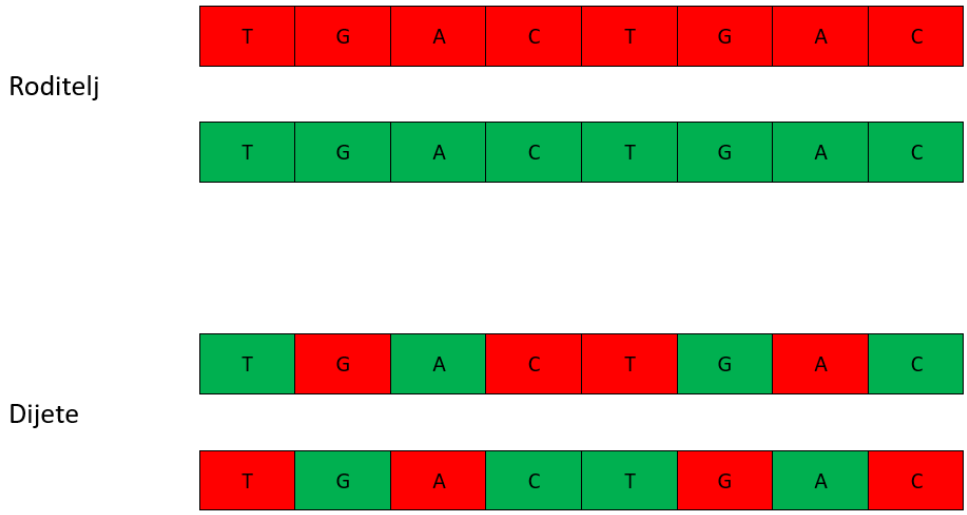
1. Križanje u jednoj točki (eng. *Single-Point Crossover*) – U ovoj metodi, točka križanja se nasumično odabire na kromosomu roditelja. Genetski materijal od početka kromosoma do točke križanja potječe od jednog roditelja, dok preostali materijal dolazi od drugog roditelja. Ovo je najjednostavnija i najčešće primjenjivana metoda u mnogim eksperimentima [25] [7]. Na slici 5 prikazan je primjer križanja u jednoj točki.

Slika 5: Križanje u jednoj točki

2. Križanje u više točaka (eng. *Multi-Point Crossover*) – Ova metoda je slična križanju s jednom točkom, samo je razlika što se koristi više od jedne točke križanja. Genetski materijal će se razmjenjivati između roditelja na svakoj od točaka križanja. Višestruko križanje može stvoriti više jedinstvenih potomaka [25] [7]. Na slici 6 prikazan je primjer križanja u više točaka.

Slika 6: Križanje u više točaka

3. Uniformno križanje (eng. *Uniform Crossover*) – Svaki gen potomaka nasumično se odabire od roditelja, neovisno o njihovom položaju. Uniformno križanje rezultirat će najvećom raznolikosti genetskog materijala [25] [7]. Na slici 7 prikazan je primjer uniformnog križanja.

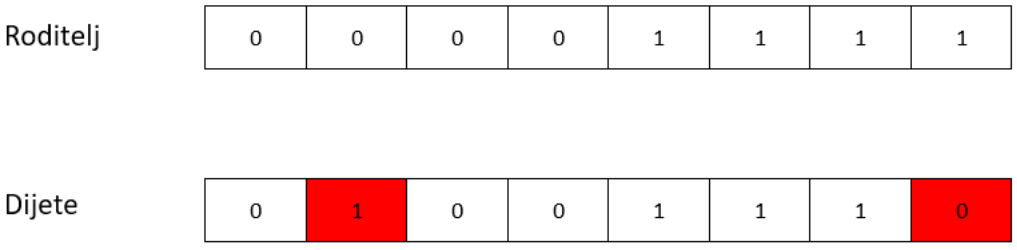


Slika 7: Uniformno križanje

2.3.1.3. Mutacija

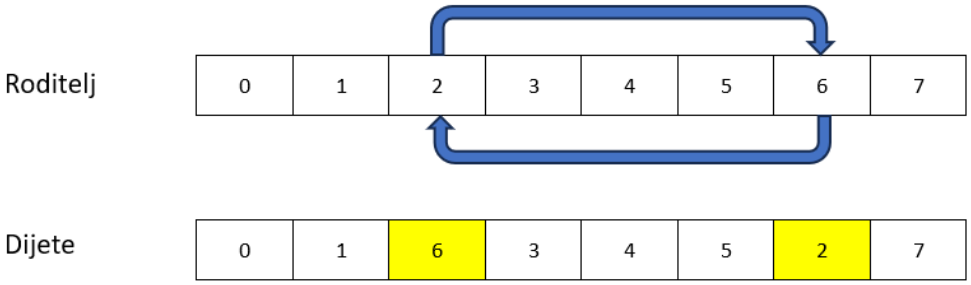
Mutacija je proces u kojem se jedan ili više gena kromosoma jedinke nasumično mijenja. Mutacija ima za cilj prenijeti varijacije unutar populacije. Neke od metoda mutacije su:

1. Bitna mutacija (eng. *Bit Flip Mutation*) - U slučaju binarnih kromosoma, bitna mutacija mijenja jedan bit iz 0 u 1 ili obrnuto. Obično je vjerojatnost mutacije niska kako bi se osigurala stabilnost populacije [7] [25]. Na slici 8 prikazan je primjer bitne mutacije.



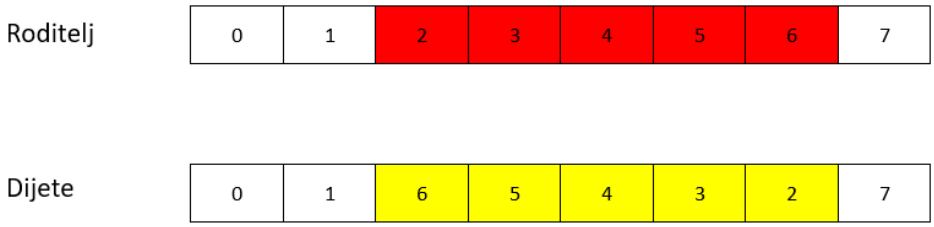
Slika 8: Bitna mutacija

2. Mutacija zamjene (eng. *Swap Mutation*) - U permutacijskim kromosomima, mutacija zamjene mijenja položaj dvaju gena. Ova metoda je osobito korisna kod problema kao što je problem trgovačkog putnika [7] [25]. Na slici 9 prikazan je primjer mutacije zamjene.



Slika 9: Mutacija zamjene

3. Inverzijska mutacija (eng. *Inversion Mutation*) – Ova mutacija uključuje nasumično biranje dviju točaka u kromosomu, a genetski materijal između te dvije točke se obrne [7] [25]. Na slici 10 prikazan je primjer inverzijske mutacije.



Slika 10: Inverzijska mutacija

2.3.2. Primjene genetskih algoritama u videoigrama

Neke od čestih uloga genetskih algoritama uključuju sve veću primjenu u dizajnu i razvoju videoigara, uglavnom u proceduralnom stvaranju sadržaja i optimizaciji. Neki od stvarnih primjera genetskog algoritma u videoigrama biti će navedeni u nastavku.

Generiranje razina – GA se često koriste za dizajniranje razina po raznim kriterijima, poput težine ili izgleda. Na primjer, videoigra *Infinite Mario Bros* koristi GA kako bi osigurala da razine nisu samo teške nego i zanimljive za igranje. Ovi algoritmi također mogu generirati razine ovisno o stilu igre i sposobnostima igrača. Razine kreirane na ovaj način ispast će jedinstvene svaki put kada se razina zaigra [41].

Ponašanje NPC-jeva - Genetski algoritmi mogu se primijeniti za optimizaciju ponaša-

nja NPC-jeva. Na primjer, u videoigri *NERO* (*Neuro-Evolving Robotic Operatives*), korištena borbena strategija razvija se pomoću GA i prilagođava se svakoj situaciji u igri. NPC-jevi uče iz izravnih interakcija s igračem te putem neizravnih interakcija, odnosno kroz učenje iz međusobnih interakcija s drugim NPC-jevima putem borbe ili suradnje. Ova uloga GA omogućuje izazovnije igranje dok se neprijatelji cijelo vrijeme prilagođavaju i razvijaju [42].

Prilagodljiva težina igre - Genetski algoritmi mogu automatski prilagoditi težinu igre kako bi odgovarala vještinama igrača. Primjerice, u videoigri *Left 4 Dead*, *AI Director* koristi genetske algoritme za prikupljanje podataka o performansama igrača i prilagođava težinu igre u stvarnom vremenu. Ova mogućnost GA može prilagoditi broj i težinu danih neprijatelja, dodijeliti resurse ili prilagoditi događaje u igri [43].

Stvaranje likova i priča - Genetski algoritmi vrlo su korisni pri stvaranju jedinstvenih likova i priča. Sjajan primjer je videoigra *Galactic Arms Race*, gdje kontinuirani rad genetskih algoritama zapravo generira jedinstvena oružja, svako prema igračevom stilu igre, kako bi igri dodali više dubine i raznolikosti. GA može smisliti vrlo različite osobnosti, pozadinske priče i motivacije za uključene likove, što dodatno povećava razinu uživljavanja igrača u videoigru. Korištenjem ove metode GA mogu se izgraditi bogatiji i složeniji svjetovi u kojima svaki lik preuzima svoju jedinstvenu ulogu i priču [44].

3. Praktični dio

U ovom poglavlju obrađena je implementacija modela za proceduralno generiranje razine - odabir platforme, definiranje elemenata razine i razvoj genetskog algoritma. Svrha ovog dijela rada je ilustrirati kako se teorijske ideje dane u prethodnim poglavljima primjenjuju u praksi.

3.1. Implementacija modela za proceduralno generiranje razina

Proceduralno generiranje razine zahtijeva korištenje odgovarajuće platforme za implementaciju. Platforma treba biti fleksibilna, podržavati napredne grafičke i fizičke sustave te imati mogućnost jednostavne integracije genetskih algoritama.

3.1.1. Izbor platforme za implementaciju

Model proceduralnog generiranja razine implementiran je pomoću razvojnog alata Unity. Kao jedan od najpopularnijih alata za razvoj videoigara, Unity je nadaleko poznat po svojoj fleksibilnosti i podršci za više platformi. Unity može jednostavno integrirati nekoliko algoritama, uključujući i genetske algoritme, s bogatim skupom alata za dvodimenzionalnu i trodimenzionalnu grafiku.

Unity je u ovom trenutku dobar jer ima veliku zajednicu i dosta dokumentacije, što može pomoći u jednostavnom rješavanju problema i integraciji naprednih značajki. Također, Unity nudi mnoge gotove elemente i dodatke koji uvelike ubrzavaju razvojni proces, kao i napredne alate za optimizaciju performansi koje koristi prilikom proceduralnog generiranja razina. Koristeći C# kao glavni jezik, Unity omogućuje učinkovitu implementaciju.

Uzimajući u obzir njegove mogućnosti, fleksibilnost i podršku zajednice, odabir Unityja kao platforme za implementaciju prilično je razumljiv. Unity kombinira snagu genetskih algoritama s naprednim alatima za dizajn videoigara koji autoru omogućuje stvaranje vrlo raznolikih razina.

3.1.2. Kratki opis cilja implementiranog modela

Cilj implementiranog modela je dobiti što povoljnije uvjete za prolazak razine, odnosno da kroz igranje svi parametri za generaciju platformi budu povoljni. Igrač se koristi kontrolama A i D za kretanje lijevo i desno te se skače pomoću razmaknice. Kretanje igrača po platformama i njegove animacije implementirane su u skripti **Kretanje.cs**.

```
public class Kretanje : MonoBehaviour {

    [SerializeField] float brzinaKretanja;
```

```
    public Rigidbody2D rb;
    public float jumpSpeed = 4f;
    bool canJump = true;
    Animator animator;

        void Update() {

        float kretanje = Input.GetAxisRaw("Horizontal");

        transform.Translate(brzinaKretanja * Time.deltaTime *
        Mathf.Abs(kretanje), 0f, 0f);

        if (kretanje != 0) {
            animator.SetBool("isRunning", true);
        } else {
            animator.SetBool("isRunning", false);
        }

        if (kretanje < 0) {
            transform.localEulerAngles = Vector3.up * 180;
        } else if (kretanje > 0) {
            transform.localEulerAngles = Vector3.zero;
        }

        if (Input.GetButtonDown("Jump") && canJump) {
            rb.AddForce(new Vector2(0, jumpSpeed), ForceMode2D.Impulse);
            canJump = false;
            animator.SetBool("IsJumping", true);
        }

        if (canJump == true) {
            animator.SetBool("IsJumping", false);
        }
    }
}
```

Svaka razina generirana je korištenjem genetskog algoritma. Cilj svake pojedine razine je doseći najvišu moguću točku, ili ako je moguće, dohvatiti dijamant koji se nalazi na samom vrhu zadnje platforme. Vrijednost funkcije korisnosti je zapravo najviša točka koju igrač dosegne u pojedinoj razini. Model pamti najbolje rezultate iz ranijih pokušaja na kraju svih razina i koristi genetski algoritam za generiranje nove generacije razina koje optimiziraju težinu i raspored platformi u sljedećoj igri. Igra se sastoji od dvije sekcije, *Overworld* sekcija i sekcija svake pojedine razine. Kada se uđe u igru, igrača se prvo smješta u *Overworld* sekciju. Na slici 11

prikazana je *Overworld* sekcija.

Slika 11: *Overworld* sekcija

Na zaslonu je moguće vidjeti natpis „Jedinke 1-5“ i pet vrata, koje označavaju ulaz u svaku pojedinu razinu. Također, u lijevom gornjem kutu zaslona moguće je vidjeti i vrijednosti pojedinih parametara vezanih za proceduralno generiranje platformi za razine, ali o tome će se pričati malo više kasnije. Uz pet pari vrata, u *Overworld* sekciji nalaze se i vrata s natpisom *New gen* (Nova generacija).

Slika 12: *New Gen* vrata

Nakon rješavanja svih pet razina te ulaskom vrata *New gen*, događa se proces selekcije, križanja i mutacije, te se stvara novih pet razina s drugačijim vrijednostima parametara. Detaljniji opis funkcionalnosti naveden je u odjeljku ispod.

3.1.3. Razvoj modela genetskog algoritma

3.1.3.1. Definiranje elemenata razine

Proceduralno generiranje razina u ovom 2D platformeru zahtijeva postojanje jasnih definicija osnovnih elemenata koji čine igru i omogućuju igraču kretanje kroz razine, postizanje ciljeva i interakciju s okolinom. Ova igra, inspirirana konceptom od dna do vrha (eng. *Bottom-to-top*), fokusira se na upravljanje igračem kako bi se popeo na najvišu platformu gdje ga čeka dijamant, koji je cilj svake razine.

3.1.3.2. Generacija platformi

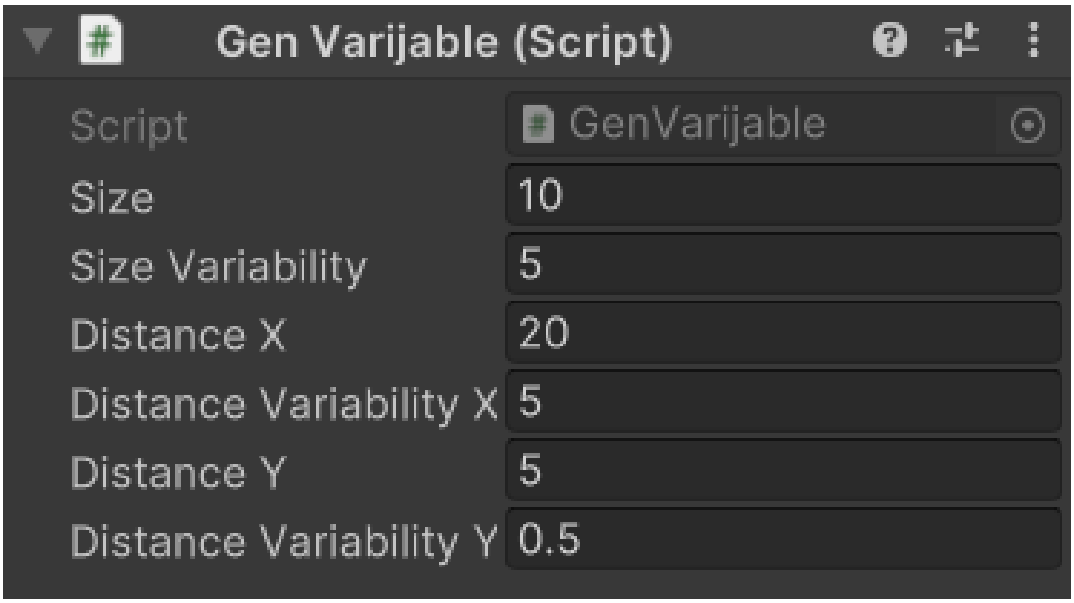
Platforme su osnovni elementi razine koje igrač koristi kako bi se popeo na sam vrh. Definirano je da svaka razina treba sadržavati 20 platformi i rasporediti ih prema parametrima njihove veličine i razmaka između njih na obje X i Y osi uz varijabilnost tih parametara. Parametri su definirani u posebnoj skripti **genVarijable.cs**, a njihove vrijednosti su zadane u Unity sučelju.

```
public class GenVarijable : MonoBehaviour
{
    public float size;
    public float sizeVariability;
    public float distanceX;
    public float distanceVariabilityX;
    public float distanceY;
    public float distanceVariabilityY;
}
```

gdje je:

Size - početna veličina, odnosno širina platforme, početna vrijednost iznosi 10.
Size variability - odstupanja od početne veličine, može povećati ili smanjiti platformu, početna vrijednost iznosi 5.
Distance X - vrijednost koja određuje razmak između svake platforme na osi X, početna vrijednost iznosi 20.
Distance X variability - vrijednost koja smanjuje ili povećava razmak između svake platforme na osi X, početna vrijednost iznosi 5.
Distance Y - vrijednost koja određuje razmak između svake platforme na osi Y, početna vrijednost iznosi 5.
Distance Y variability - vrijednost koja smanjuje ili povećava razmak između svake platforme na osi Y, početna vrijednost iznosi 0.5.

Na slici 13 dan je prikaz početnih vrijednosti zadanih parametara.



Slika 13: Početne vrijednosti zadanih parametara

Platforme se generiraju proceduralno, a definirano je da se generira točno 20 platformi. Dakle, zbog toga će njihove karakteristike biti određene nasumično unutar određenih granica koje postavlja genetski algoritam. Granice moraju biti postavljene kako bi se spriječilo preklapanje platformi, a samim time i velika udaljenost između platformi, što bi vremenski produžilo postizanje optimalnog rješenja. Uvjeti, granice i način generacije platformi dani su u skripti **GenetskiAlgoritam.cs**.

```
public class GenetskiAlgoritam : MonoBehaviour {

    public Transform[] generation = new Transform[5];
    public Transform platform;
    public Transform diamond;
    public int currLevel;
    bool overworld = true;

        public void LevelGenerator(Scene scene, LoadSceneMode mode) {

        if (overworld) {

            Vector2 lastPos = Vector2.down * 2;
            GenVarijable genv =
            generation[currLevel].GetComponent<GenVarijable>();

            for (int i = 0; i < 20; i++) {

                Transform plat = Instantiate(platform);
                plat.position = lastPos;
```

```
                float dx = UnityEngine.Random.Range(0f, 1f) < 0.5f ?
                genv.distanceX : -genv.distanceX;

                plat.position += Vector3.right * (dx +
                UnityEngine.Random.Range(-genv.distanceVariabilityX,
                genv.distanceVariabilityX))+ Vector3.up * (genv.distanceY +
                UnityEngine.Random.Range(0, genv.distanceVariabilityY));

                Vector3 size = new Vector3(genv.size +
                UnityEngine.Random.Range(0, genv.sizeVariability), 1, 1);
                plat.localScale = size;

                lastPos = plat.position;
            }

            Instantiate(diamond, lastPos + Vector2.up * 4.5f,
            transform.rotation);

        }

        overworld = !overworld;

}
}
```

Granice se primjenjuju prilikom križanja i mutacije. Razlog tome je što vrijednosti koje su početno zadane nikad neće moći prijeći granicu. Funkcija *Selection()* poziva se prilikom ulaska u *New gen* vrata koja su prethodno spomenuta. Selekcija se vrši pomoću turnir selekcije, samo je razlika što se u primjeru ovog implementiranog modela umjesto jednog uzorka populacije koristi cijela populacija, a umjesto jedne najbolje jedinke izabiru se dvije. Križanje je uniformno, što znači da se svaki gen za potomstvo nasumično odabire između dva roditelja. Mutacija je bitna, te svaka vrijednost parametara u implementiranom modelu mutira.

```
public float mutationPower;
GenVarijable genA = parentA.GetComponent<GenVarijable>();
GenVarijable genB = parentB.GetComponent<GenVarijable>();


    for (int i = 0; i < generation.Length; i++) {
        Transform child = Instantiate(parentA, transform);
        generation[i] = child;
        GenVarijable genChild = child.GetComponent<GenVarijable>();
```

```
        genChild.size = Cross(genChild.size, genA.size, genB.size, 2, 20);

        genChild.sizeVariability = Cross(genChild.sizeVariability,
        genA.sizeVariability, genB.sizeVariability, 0, 15);

        genChild.distanceX = Cross(genChild.distanceX,
        genA.distanceX, genB.distanceX, 15, 25);

        genChild.distanceVariabilityX = Cross(genChild.distanceVariabilityX,
        genA.distanceVariabilityX, genB.distanceVariabilityX, 0, 10);

        genChild.distanceY = Cross(genChild.distanceY,
        genA.distanceY, genB.distanceY, 3.4f, 6f);

        genChild.distanceVariabilityY = Cross(genChild.distanceVariabilityY,
        genA.distanceVariabilityY, genB.distanceVariabilityY, 0, 1);
    }

float Cross(float ch, float a, float b, float min, float max) {

    ch = UnityEngine.Random.Range(1, 101) > 50 ? a : b;

    float mut = (ch / 2) * Math.Max(1, mutationPower);

    ch += (float)(UnityEngine.Random.Range(-mut, mut));

    ch = Mathf.Clamp(ch, min, max);


    return ch;

}

public void Selection() {
    int maxA = 0, maxB = 0;
    for (int i  = 1; i < generation.Length; i++) {
        if (fitness[i] > fitness[maxA]) {
            maxA = i;
        }
    }
    for (int i = 0; i < generation.Length; i++) {
        if (fitness[i] > fitness[maxA] && maxA != maxB) {
            maxB = i;
```

```
        }
    }

    Transform parentA, parentB;
    parentA = generation[maxA];
    parentB = generation[maxB];
```

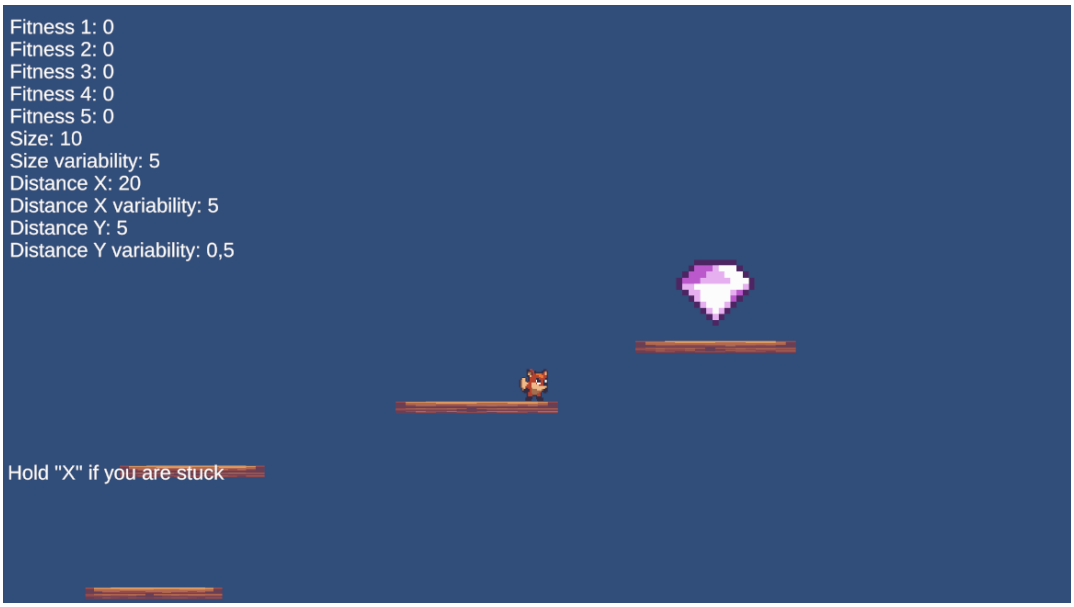
Granice su zadane u *for* petlji, gdje su zadnje dvije vrijednosti u argumentu poziva funkcije *Cross()* minimalna i maksimalna vrijednost granice. Odabir gena roditelja vrši se u funkciji *Cross()*, na način da se nasumično odabere broj u rasponu između 1 i 101. Ako je broj veći od 50, uzima se vrijednost roditelja A, a ako je broj manji od 50 uzima se vrijednost roditelja B. Ta se vrijednost potom dijeli s brojem 2 i množi s brojem 1 (vrijednost *mutationPower* iznosi 1), te se pridružuje vrijednosti *mut*, što označava vrijednost mutacije. Zatim se uzima nasumična vrijednost intervala između -*mut* i *mut*, te se zbraja s vrijednosti početnog parametra *ch*. U zadnjem dijelu koda u funkciji *Cross()*, funckija **Mathf.Clamp** ograničava vrijednost *ch* na način da ako vrijednost prijeđe maksimalnu vrijednost, ona se postavlja na maksimalnu vrijednost, te ista stvar vrijedi i za minimalnu vrijednost. Mutacija i nasumičan odabir vrše se za sve parametre u implementiranom modelu.

3.1.3.3. Cilj

Dijamant je objekt koji predstavlja cilj. Nalazi se na najvišoj platformi kao nagrada koju igrač treba prikupiti kako bi uspješno prešao razinu, čime postiže najveću vrijednost funkcije korisnosti. Njegovo ponašanje kontrolira skripta **Diamond.cs**. Ova skripta provjerava koliziju igrača s dijamantom, te kad ga se pokupi vraća igrača natrag u *Overworld* sekciju.

```
    public class Diamond : MonoBehaviour
{

    private void OnTriggerEnter2D(Collider2D collision) {

        if (collision.tag == "Player") {
            GenetskiAlgoritam algoritam =
            FindAnyObjectByType<GenetskiAlgoritam>();

            algoritam.fitness[algoritam.currLevel] =
            transform.position.y * 100;

            SceneManager.LoadScene(1);
        }

    }

}
```

Kao što se vidi u prikazu koda, kada se sakupi dijamant, vrijednost funkcije korisnosti se množi s brojem 100 kako bi se dodatno nagradilo igrača za postignuti uspijeh i samim time povećala šansa za odabir dane razine u procesu selekcije. Na slici 14 dan je prikaz izgleda dijamanta koji se nalazi na najvišoj platformi razine.



Slika 14: Dijamant

3.1.3.4. Vrata za pristup razinama

Vrata za pristup razinama su entitet koji odvodi igrača iz *Overworld* dijela igre u proceduralno generiranu razinu pritiskom na tipku "E". Postoje dvije vrste vrata; vrata za pristup razinama i vrata za stvaranje nove generacije. Vrata za stvaranje nove generacije biti će detaljnije objašnjena u točki ispod. U skipti **Kretanje.cs**, zadani su uvjeti koji razlikuju dvije vrste vrata, a u skripti **DoorCollision.cs** implementirana je funkcionalnost vrata i detekcije kolizije s vratima.

Skripta **DoorCollision.cs**:

```
public class DoorCollision : MonoBehaviour
{

    public int level;
    public bool genDoor = false;


    private void OnTriggerEnter2D(Collider2D collision) {
        if (collision.tag == "Player") {
            if (!genDoor) {
                collision.GetComponent<Kretanje>().door = transform;
```

```
            } else {
                FindAnyObjectByType<Kretanje>().nextGen = true;
            }
        }
    }

    private void OnTriggerExit2D(Collider2D collision) {
        if (collision.tag == "Player") {
            if (!genDoor) {
                collision.GetComponent<Kretanje>().door = null;
            } else {
                FindAnyObjectByType<Kretanje>().nextGen = false;
            }
        }
    }
}
```

Skripta **Kretanje.cs**:

```
public class Kretanje : MonoBehaviour {

    public Transform door;
    public bool nextGen = false;

    if (Input.GetKeyDown(KeyCode.E) && door) {

        FindAnyObjectByType<GenetskiAlgoritam>().currLevel =
        door.GetComponent<DoorCollision>().level;
        SceneManager.LoadScene(2);
    }

    else if (Input.GetKeyDown(KeyCode.E) && nextGen) {
        FindAnyObjectByType<GenetskiAlgoritam>().Selection();
        transform.position = Vector3.zero;
    }
 }
```

Na slici 15 dan je prikaz vrata za pristup razinama.

Slika 15: Vrata za pristup razinama

3.1.3.5. Vrata za stvaranje nove generacije *(New gen* vrata)

Slično kao i kod vrata za pristup razinama, pritiskom na tipku "E", igrač ulazi u vrata za stvaranje nove generacije te se izvršava proces selekcije, križanja i mutacije. Kao što je već rečeno, skripta **GenetskiAlgoritam.cs** upravlja generiranjem razina i optimizacijom uvjeta na temelju rezultata iz prethodnih igranja. Što se tiče ocjenjivanja uspjeha generiranih razina, ključna je funkcija korisnosti, koja prati koliko je visoko igrač došao na razini. Kod za ovu funkcionalnost dan je u točki iznad. Na slici 16 dan je prikaz vrata za stvaranje nove generacije.

Slika 16: Vrata za stvaranje nove generacije (*New gen*)

3.1.3.6. Implementacija funkcije korisnosti

Za procjenu kvalitete generirane razine u implementiranom modelu koristi se funkcija korisnosti ili fitness funkcija. U ovom slučaju, funkcija korisnosti je najviša vrijednost koju igrač može doseći u razini, odnosno visina koju igrač dosegne prije pada ili završetka razine. Što je veća dosegnuta visina, veća je i vrijednost funkcije korisnosti. Kada igrač završi razinu ili izađe iz nje, generira se rezultat na temelju njegovog uspjeha. Funkcija korisnosti osigurava da samo najuspješniji dizajni razina mogu ići u sljedeću generaciju. Kod za funkcionalnost funkcije korisnosti naveden je u skripti **Kretanje.cs**.

```
    if (Input.GetKey(KeyCode.X) && canExit) {

    pressedX += Time.deltaTime;

    if (pressedX > 1) {

        GenetskiAlgoritam algoritam =
        FindAnyObjectByType<GenetskiAlgoritam>();

        algoritam.fitness[algoritam.currLevel] = maxFit;
        SceneManager.LoadScene(1);
    }
}
    if (maxFit < transform.position.y) {
        maxFit = transform.position.y;
        }
```

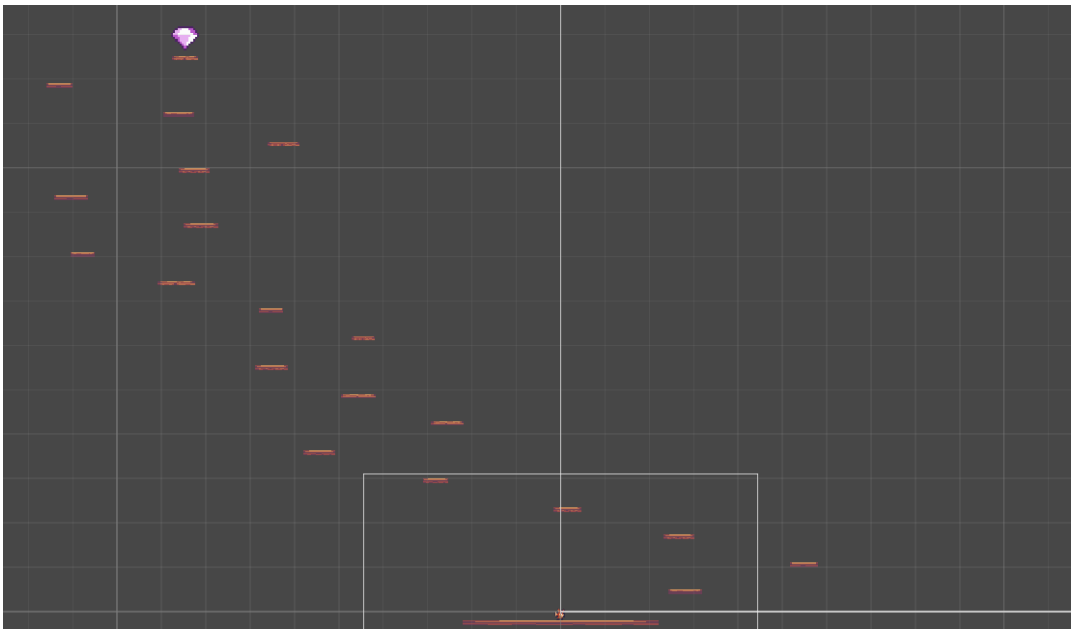
Ovaj kod implementira proces praćenja i spremanja fitness vrijednosti. Gornji dio koda prati maksimalnu visinu koju je igrač dosegao tijekom igre. Uz to, koristi se varijabla *maxFit* za pohranu y koordinate pozicije igrača. Kad god igrač dosegne novu, višu poziciju, *maxFit* se ažurira s novom vrijednošću. Ovo omogućuje sustavu praćenje najviše točke koju je igrač dosegao na razini.

Igra također detektira pritisak na tipku "X" koju igrač koristi za napuštanje razine. Držanje gumba "X" dulje od jedne sekunde pokreće niz akcija. Prvo, pronalazi se klasa **GenetskiAlgoritam** i koristi se za pohranjivanje maksimalne visine *maxFit* u element niza *fitness* unutar genetskog algoritma, koristeći indeks trenutne razine (*currLevel*). Ova vrijednost predstavlja ključni podatak na kojoj će genetski algoritam temeljiti svoju sljedeću generaciju razina kada ih napravi, optimizirajući ih na temelju postignutih rezultata.

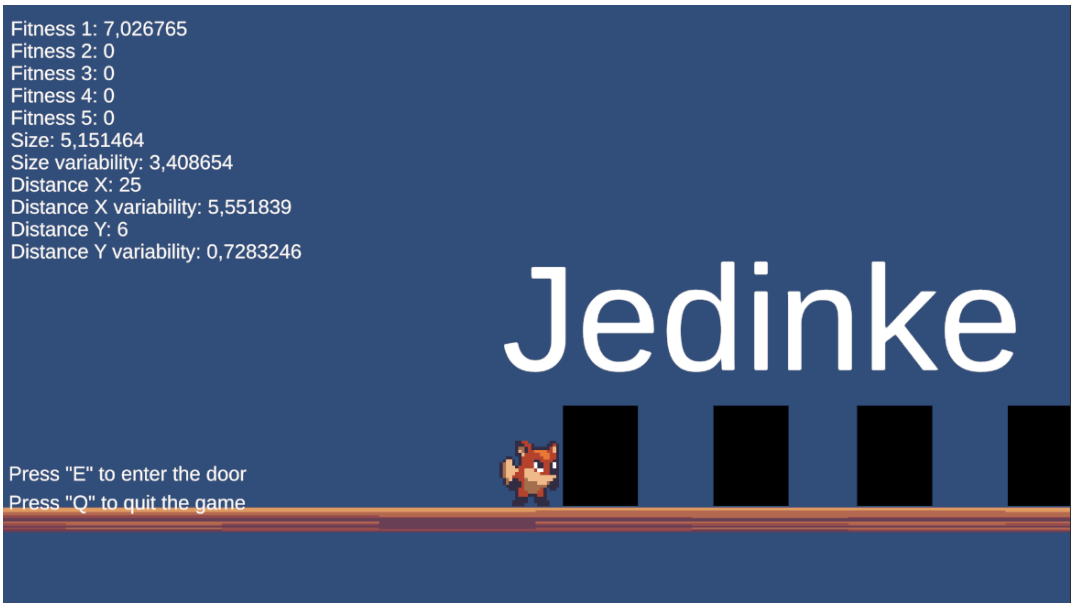
Nakon spremanja vrijednosti *fitness*, sustav vraća igrača iz razine u scenu *Overworld*, omogućujući igraču da vidi rezultate svog napretka. Kao što je već rečeno, nakon prolaska svih razina igrač pristupa vratima za stvaranje nove generacije pri čemu se vrši selekcija dvije razine s najboljim fitness vrijednostima.

3.1.4. Vizualizacija generiranih razina

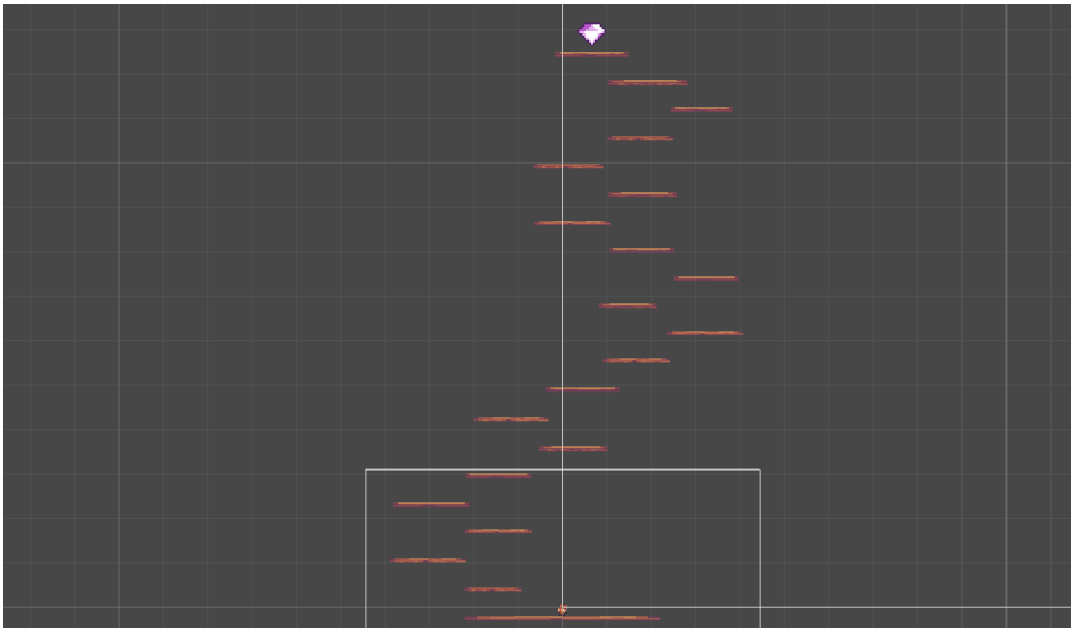
Vizualizacija generiranih razina ključna je za procjenu uspješnosti genetskog algoritma u kreiranju platformi unutar igre. Nakon stvaranja nove razine, igra automatski prikazuje sve elemente te razine. Svaka generirana razina prikazana je u stvarnom vremenu unutar igre. Elementi razine, poput platformi, generiraju se proceduralno prema parametrima koje je odredio genetski algoritam, dok su pozicije dijamanta također određene istim procesom. Igrač odmah može vidjeti rezultat proceduralnog generiranja, a vizualni prikaz omogućava mu pregled platformi koje su postavljene pred njega. Uz vizualne elemente razine, na ekranu se prikazuju i dodatne informacije koje omogućuju bolje razumijevanje procesa generacije. Na primjer, parametri generacije kao što su veličina platformi, udaljenost između njih i varijabilnost tih parametara prikazani su u lijevom gornjem kutu zaslona tijekom igre. Nakon što igrač završi razinu, prikazuju se informacije o postignutom *fitnessu*, tj. najvišoj točki koju je igrač dosegnuo, ovisno o razini koju je završio. Na slikama 17, 18, 19, i 20 biti će prikazan dizajn razina, zajedno sa prikazom promjene vrijednosti fitnessa nakon završetka pojedine razine. Na slikama 18 i 20, također su prikazane vrijednosti parametara za pojedinu razinu.



Slika 17: Prikaz razine 1



Slika 18: Promjena vrijednosti fitnessa nakon rješavanja razine 1, igrač je morao prisilno izaći jer nije mogao rješiti razinu po zadanim parametrima



Slika 19: Prikaz razine 2

Slika 20: Promjena vrijednosti fitnessa nakon rješavanja razine 2, igrač je uspješno rješio razinu i dobio najveću moguću vrijednost fitnessa

3.2. Evaluacija generiranih razina

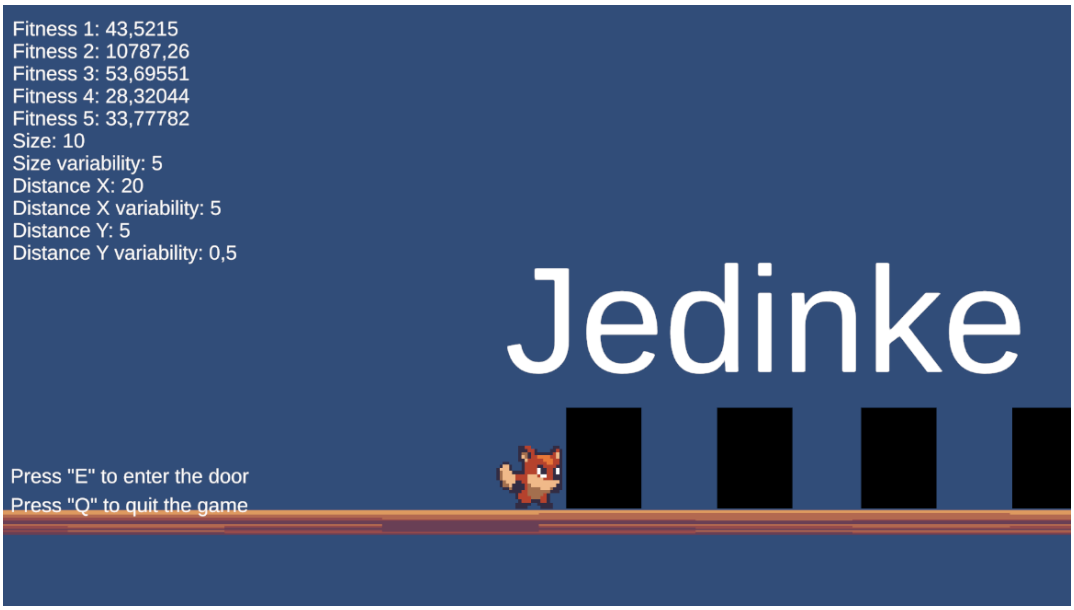
U ovom odjeljku opisane su radnje koje su izvedene u praktičnom dijelu rada za evaluaciju generiranih razina. Igračka testiranja i analiza iskustva igre provedeni su s naglaskom na težinu i zanimljivost razina.

3.2.1. Igračka testiranja

Igračka testiranja provedena su kako bi se utvrdila tehnička ispravnost i mogućnost igranja svake generirane razine. Igrači su igrali kroz generacije različitih razina kako bi dali povratne informacije o tome kako se kvaliteta i težina razina mijenjaju kroz evoluciju. Na taj način prikupljeni su podaci koji su omogućili prepoznavanje problema u dizajnu razina, bilo da se radi o neprohodnim ili predalekim platformama. U početku razvoja implementiranog modela, nije postojala donja i gornja granica za dane parametre za generaciju platformi, što je često rezultiralo razinama koje se ne mogu prijeći. Budući da igra smješta igrača na sredinu platforme pri ulasku u razinu, znalo se dogoditi preklapanje platforme i igrača, što je uzrokovalo dodatne probleme i nemogućnost kretanja igrača. Za definiranje granica, u obzir se uzela visina igraćeg lika, visina njegovog skoka u vis te brzina kretanja. Ove promjene su uvelike olakšale pronalazak idealnih razina i uštedjele na igračevom vremenu igranja, bez remećenja rada genetskog algoritma.
U nastavku je prikazano kako se prolaskom kroz generacije postepeno olakšava prelazak razina.

3.2.1.1. Prva generacija

Prva generacija započinje na način da su svi parametri razina isti. Početni parametri već su zadani u točki **3.1.3.2. Generacija platformi**. Igrač pristupa svakoj razini i pokušava je riješiti najbolje što može. Nakon rješavanja svih 5 razina prve generacije, pregledava se vrijednost fitnessa pojedinih razina.

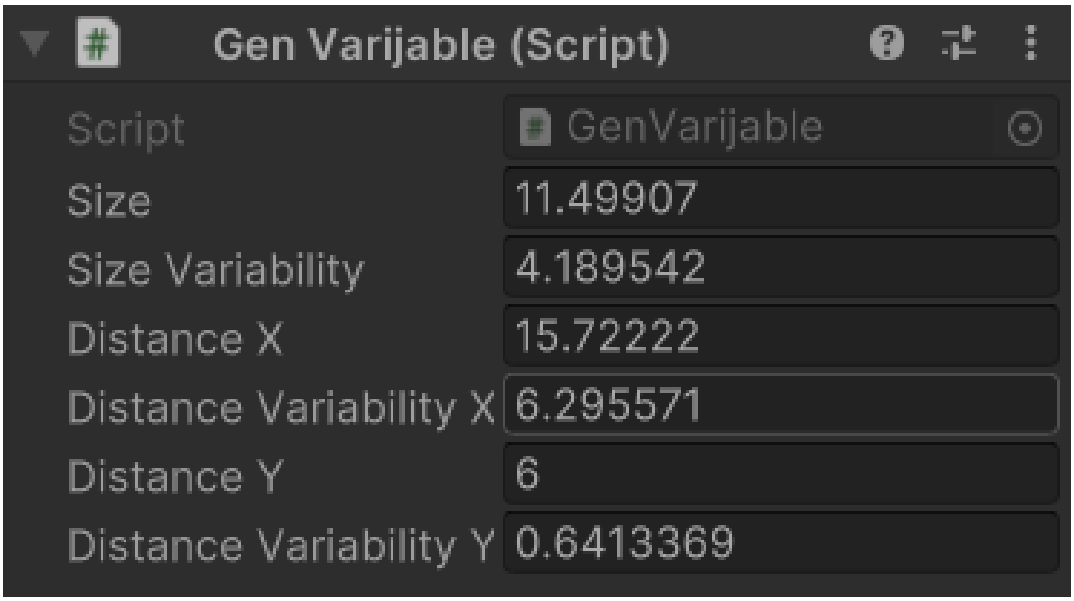


Slika 21: Vrijednosti fitnessa nakon rješavanja svih razina prve generacije

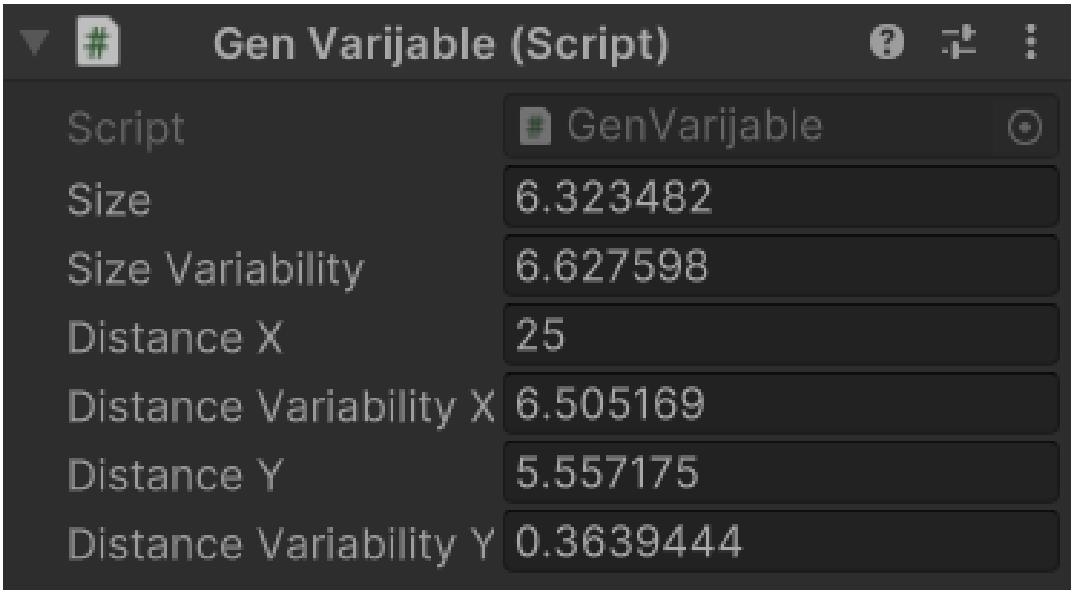
Kao što se vidi na slici, igrač je uspješno prešao samo razinu 2, dok je kod ostalih razina morao prisilno izaći. Sada igrač pristupa *New gen* vratima, gdje će se izvršiti proces selekcije, križanja i mutacije. Iz prethodnih objašnjenja već znamo da će u procesu selekcije biti odabrane razine 2 i 3, iz razloga što imaju najveću vrijednost funkcije korisnosti.

3.2.1.2. Druga generacija

Nakon selekcije, križanja i mutacije, dobivaju se potpuno različite vrijednosti parametara za pojedinu razinu. Prije pristupa razini, u Unity razvojnom okruženju, moguće je vidjeti te vrijednosti. Na slici 22 i 23 prikazan je primjer promijenjenih vrijednosti parametara razine 1 i 2.



Slika 22: Vrijednosti parametara za razinu 1



Slika 23: Vrijednosti parametara za razinu 2

Na slici 22 uočljivo je kako bi za parametar *Distance Y*, odnosno udaljenost platformi na osi Y, vrijednost prešla granicu, no to je onemogućeno pravilnom implementacijom te se zato vrijednost postavlja na maksimalnu vrijednost granice. Također, na slici 23 ista stvar vrijedi za parametar *Distance X* (udaljenost platformi na osi X).

Nakon rješavanja svih razina druge generacije, pregledavaju se fitness vrijednosti.

Slika 24: Vrijednosti fitnessa nakon rješavanja svih razina druge generacije

U drugoj generaciji, igrač je uspješno prešao 3 od 5 razina. Razina 1 imala je dosta povoljne vrijedosti parametara, dok je razina 2 imala preveliku udaljenost platformi na osi X, što je onemogućilo efikasno skakanje po lijevoj i desnoj strani. Na slici 24 može se uočiti da će roditelji za potomstvo treće generacije biti razina 1 i 5.

3.2.1.3. Treća generacija

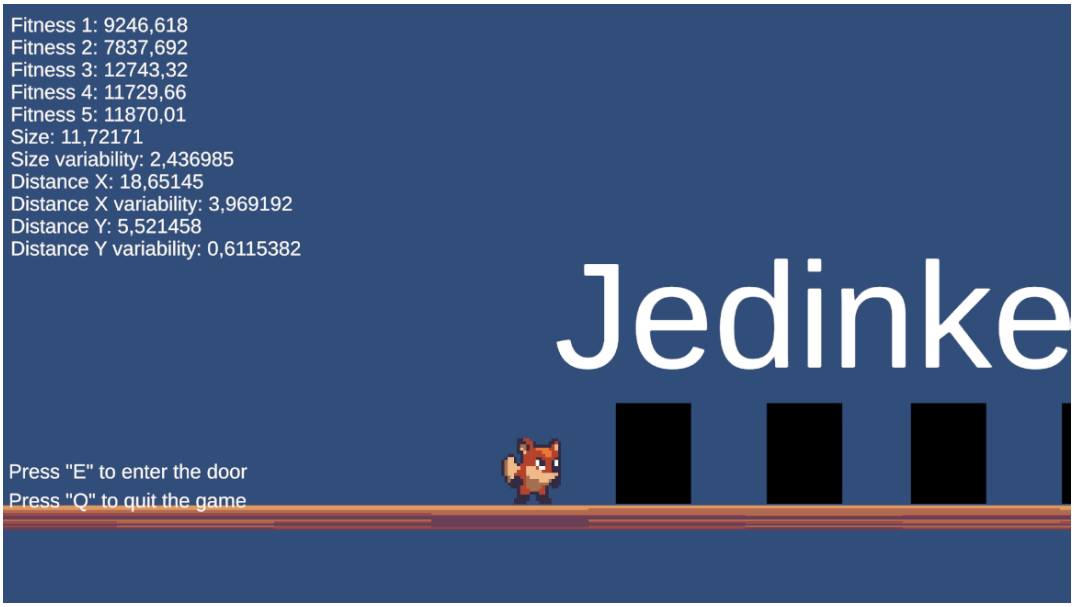
Razine treće generacije bile su pogodnije za rješavanje od razina druge generacije. Također se uspješno prešlo 3 od mogućih 5 razina, a razlog za nemogućnost prelaska preostalih dviju razina bio je utjecaj mutacije koji je smanjio veličine platformi, a povećao njihovu udaljenost. Prilikom igranja razine 4, igrač je došao do 19. platforme, te ga je jedan skok dijelio do uspješnog prelaska. Nažalost, udaljenost između zadnje dvije platforme bila je prevelika.

Slika 25: Vrijednosti fitnessa nakon rješavanja svih razina treće generacije

Roditelji za potomstvo četvrte generacije biti će razina 3 i 5.

3.2.1.4. Četvrta generacija

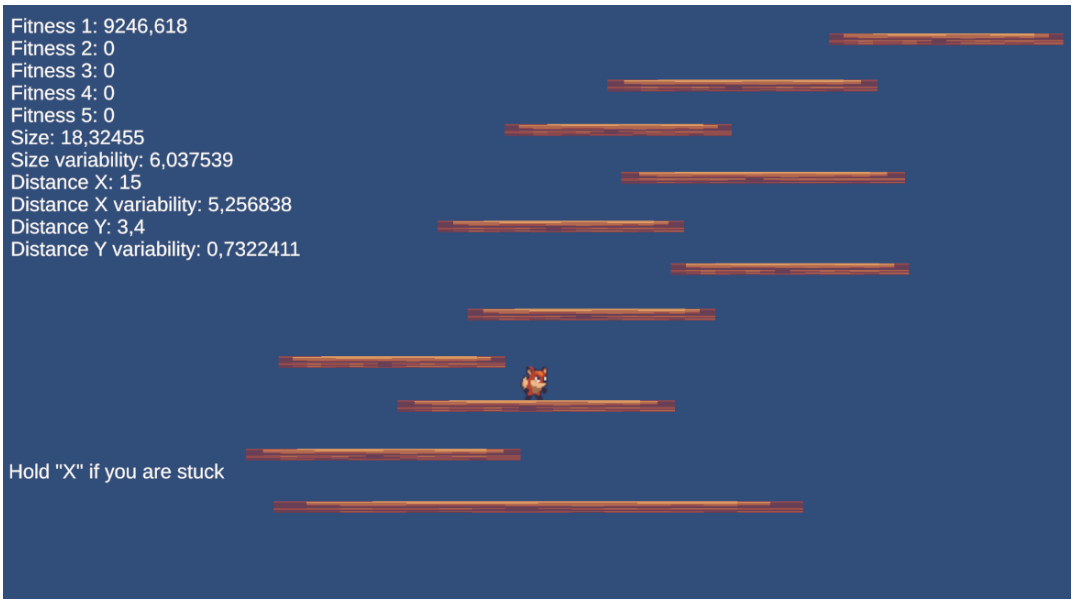
Četvrta generacija rezultirala je velikim iznenađenjem. Igrač je uspješno prešao svih 5 razina.



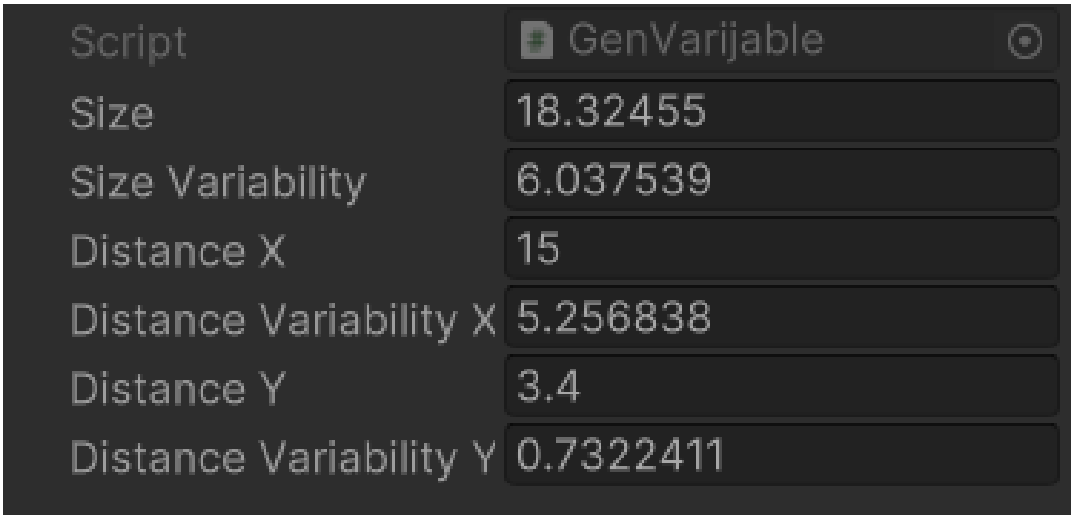
Slika 26: Vrijednosti fitnessa nakon rješavanja svih razina četvrte generacije

Razlog tome su povoljne vrijednosti roditelja treće generacije (razina 3 i 5) te male vrijednosti mutacije. Razine su se svele s preciznog i opreznog skakanja i upravljanja na razine koje su dosta lakše. Na slici 27 prikazan je izgled razine 2 četvrte generacije, a na slici 28

vrijednosti njezinih parametara.



Slika 27: Izgled razine 2 četvrte generacije



Slika 28: Vrijednosti parametara razine 2 četvrte generacije

Slične vrijednosti parametara bile su i za ostale razine, no razina 2 iznenadila je svojim neuobičajenim izgledom. Naime, prilikom igranja razine 2 nema prostora za grešku. Igrač se samo mora pozicionirati blizu ruba platforme i skočiti na drugu platformu te ako padne, može ponovno pokušati, što nije čest primjer kod proceduralnog generiranja razina za implementirani model. Na slici 28 može se primijetiti da je vrijednost parametra *Distance X* (Udaljenost između platformi na osi X) postavljena na minimum, dok je vrijednost parametra *Size* (Veličina platformi) vrlo visoka. U prosjeku, za rješavanje svih 5 razina trebalo se izmijeniti otprilike 7 generacija. U ovom primjeru to nije bio slučaj, jer su vrlo povoljne vrijednosti parametara roditelja u trećoj generaciji omogućile lako rješavanje razina u četvrtoj generaciji.

3.2.2. Analiza igračkog iskustva

Analiza igračkog iskustva provedena je s namjerom da se procijeni kako različite generacije razina utječu na težinu i zanimljivost igre. Uspješno prelaženje razina i razmišljanje igrača pruža važne informacije za procjenu učinkovitosti genetskog algoritma. Igračka iskustva dovela su do sljedećih zaključaka.

U prvoj generaciji, sve vrijednosti parametara bile su iste, što je dovelo do prevelike homogenosti. Općenito, to je činilo razine vrlo predvidivima i monotonima. Prva generacija smatra se pomalo dosadnom, s nedovoljno raznolikosti prilikom generacije platformi.

U drugoj generaciji, nakon prve primjene selekcije, križanja i mutacije, došlo je do značajne promjene parametara, što je dovelo do veće raznolikosti, ali i težih razina. Dan je komentar da su neke od razina nerješive, posebno raziname gdje su udaljenosti između platformi veće. Ova generacija je shvaćena kao teška, ali zanimljiva zbog novih izazova. Također, ova generacija pokazala je važnost uravnoteženja parametara kako bi se osigurala optimalna težina.

Treća generacija donijela je neka poboljšanja težine, ali je i dalje zadržala raznolikost. Ova generacija ocijenjena je kao malo pogodnija u usporedbi s drugom generacijom. Ipak, usprkos toj činjenici, generacija je i dalje bila izazovna. Ističe se balans raznolikosti i težine, kao i činjenica da je igranje postalo zanimljivije i uzbudljivije.

Četvrta generacija bila je veliko iznenađenje, a dobila je pohvale kao generacija koja je najugodnija za igranje. Prešle su se sve razine zahvaljujući povoljnim vrijednostima roditeljskih parametara iz prethodne generacije. Primijećeno je da su razine prohodnije, a da su i dalje dovoljno složene da ne izgube interes. Ova generacija također je pokazala kako pravilna selekcija roditelja i roditeljskih parametara može itekako utjecati na zanimljivost i težinu razina nadolazećih generacija.

Ukupno gledano, analiza igračkog iskustva pokazala je da su kasnije generacije, koje su prošle kroz nekoliko ciklusa selekcije, križanja i mutacije, rezultirale razinama koje su bile izazovnije, ali i pravednije prema igračima. Istaknuto je da je postepena promjena razina i težine kroz generacije bila motivirajuća. Također, prilagodbe genetskog algoritma temeljene na povratnim informacijama omogućile su stvaranje razina koje su bile balansirane u smislu težine i zanimljivosti, što je rezultiralo boljim ukupnim iskustvom igranja.

4. Zaključak

4.1. Kratki pregled teorijskih postavki i rezultata praktičnog rada

Ovaj rad daje pregled glavne primjene genetskih algoritama u proceduralnom generiranju za dvodimenzionalne platformer videoigre. Teorijski dio prvo opisuje povijest i razvoj platformera, te potom daje uvid kako funkcioniraju genetski algoritmi, uključujući selekciju, križanje, mutaciju i njihovu primjenu u videoigrama. Ispituju se različite proceduralne tehnike generiranja, kao što su gramatske metode i metode postavljanja šablona.

U praktičnom dijelu rada razvijen je model genetskog algoritma koristeći Unity razvojno okruženje za generiranje razina igre. Dok igrač prolazi kroz razine u mnogim generacijama, parametri razina optimizirani su pomoću genetskog algoritma kako bi se povećao stupanj izazova, kao i zanimljivost same videoigre. Testiranje razina, analiza igračkog iskustva te vizualizacija generiranih razina uvelike su pomogli pri evaluaciji performansi razvijenog modela. Rezultati pokazuju da se tijekom nekoliko generacija postiže značajno poboljšanje kvalitete razina, koje postaju bolje uravnotežene.

4.2. Prednosti i nedostatci razvijenog modela

4.2.1. Diskusija ograničenja i mogućnosti poboljšanja

Prednosti modela genetskog algoritma razvijenog za proceduralno generiranje razina su generiranje različitih i ponekad teških razina koje su zanimljive za igranje. Među ostalim prednostima modela su generiranje velikog broja razina bez ručnog dizajna i konstantna promjena izgleda razina, što održava interes igrača.

Međutim, model ima i neke nedostatke. Među glavnim nedostacima je mogućnost da razine budu preteške ili nepoštene zbog učinka mutacije ili nasumičnog odabira lošijih vrijednosti parametara, što u većini slučajeva frustrira igrače. Nekad se razina čini lagana, dok se ne dođe do jednog dijela koji je nerješiv. Takve situacije mogu posebno frustrirati, pa čak i naljutiti igrače, što dovodi do prekida igranja i manjka zainteresiranosti za videoigru.

Mogućnosti poboljšanja za navedene nedostatke uključuju implementaciju naprednijih tehnika u selekciji i križanju kako bi se smanjile šanse za stvaranje nepoštenih razina. Također, korištenjem nekih složenijih elemenata, kao što su neprijatelji ili pokretne platforme, može se povećati raznolikost i izazov u generiranim razinama.

U konačnici, implementirani model bi osim automatskog generiranja razina trebao raditi i na kontinuiranom poboljšanju iskustva igrača. Algoritam može biti izvrsno implementiran, ali ako on ne zadržava pažnju i zainteresiranost igrača, on nema svrhu i takva videoigra će se teško probiti na tržištu.

5. Popis literatura

[1] N. Shaker, J. Togelius i M. J. Nelson, ur., *Procedural Content Generation in Games*. Springer, 2016.

[2] Nintendo, *Super Mario Bros*, Nintendo Entertainment System Game, 1985.

[3] Sega, *Sonic the Hedgehog*, Sega Genesis Game, 1991.

[4] S. L. Kent, *The Ultimate History of Video Games: From Pong to Pokémon and Beyond*. Three Rivers Press, 2001.

[5] J. Togelius, G. N. Yannakakis, K. O. Stanley i C. Browne, „Search-based Procedural Content Generation: A Taxonomy and Survey," *IEEE Transactions on Computational Intelligence and AI in Games*, str. 172–186, 2011.

[6] G. N. Yannakakis i J. Togelius, *Artificial Intelligence and Games*. Springer, 2018.

[7] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[8] Nintendo, *Donkey Kong*, Arcade Game, 1981.

[9] RedBull, *The evolution of platform games in 9 steps*, Accessed: 2024-07-20, 2017. adresa: https://www.redbull.com/in-en/evolution-of-platformers.

[10] Britannica, *Super Mario bros*, Accessed: 2024-07-20, 2024. adresa: https://www.britannica.com/topic/Super-Mario-Bros.

[11] I. A. Wayback Machine, *Sonic the Hedgehog, Sonic Boom*, Accessed: 2024-07-20, 2004. adresa: https://web.archive.org/web/20040822083659/http://www.1up.com/do/feature?cId=3134008.

[12] Wikipedia, *Sonic the Hedgehog*, Accessed: 2024-07-20, 2024. adresa: https://en.wikipedia.org/wiki/Sonic_the_Hedgehog.

[13] Namco, *Pac-Man*, Arcade Game, 1980.

[14] B. G. Studios, *The Elder Scrolls V: Skyrim*, Video Game, 2011.

[15] H. Games, *No Man's Sky*, Video Game, 2016.

[16] M. Productions, *Middle-earth: Shadow of Mordor*, Video Game, 2014.

[17] Maxis, *The Sims*, Video Game, 2000.

[18] G. M. T. on Substack, *The Genius AI Behind The Sims*, Accessed: 2024-07-21, 2023. adresa: https://gmtk.substack.com/p/the-genius-ai-behind-the-sims.

[19] I. Millington i J. Funge, *Artificial Intelligence for Games*. CRC Press, 2016.

[20] Wikipedia, *Artificial intelligence in video games*, Accessed: 2024-07-21, 2024. adresa: https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games.

[21] HCL, *Rougelike*, Accessed: 2024-07-21. adresa: https://www.hcl.hr/zanr/roguelike/.

[22] M. Studios, *Minecraft*, Video Game, 2011.

[23] J. McIntosh, *Minecraft: The Unlikely Tale of Markus "Notch" Persson and the Game that Changed Everything*. Random House, 2013.

[24] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

[25] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[26] J. H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.

[27] P. E. Hart, N. J. Nilsson i B. Raphael, „A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, str. 100–107, 1968.

[28] R. A. S, *A* Algorithm Concepts and Implementation*, 2024. adresa: https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm.

[29] E. W. Dijkstra, „A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, str. 269–271, 1959.

[30] R. W. Floyd, „Algorithm 97: Shortest Path," *Communications of the ACM*, str. 345, 1962.

[31] R. S. Sutton i A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[32] N. Justesen, P. Bontrager, J. Togelius i S. Risi, „Deep Learning for Video Game Playing," *IEEE Transactions on Games*, str. 1–20, 2019.

[33] S. Rabin, *Game AI Pro 2: Collected Wisdom of Game AI Professionals*. CRC Press, 2015.

[34] S. R. Musse i D. Thalmann, „A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis," *Computer Animation and Simulation '97*, Springer, 1997., str. 39–51.

[35] G. Smith i J. Whitehead, „Analyzing the Expressive Range of a Level Generator," *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ACM, 2011., str. 1–7.

[36] F. Foundation, *What Are Fractals?* Accessed: 2024-07-22. adresa: https://www.fractalfoundation.org/resources/what-are-fractals/.

[37] A. Lindenmayer, „Mathematical Models for Cellular Interactions in Development I. Filaments with One-Sided Inputs," *Journal of Theoretical Biology*, str. 280–299, 1968.

[38] L. J. Johnson, G. N. Yannakakis i J. Togelius, „Cellular Automata for Real-Time Generation of Infinite Cave Levels," *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ACM, 2010., str. 1–4.

[39] K. Donnelly, *Rogue: Exploring the Dungeons of Doom*. Boss Fight Books, 2014.

[40] G. N. Yannakakis i J. Togelius, „Experience-Driven Procedural Content Generation," *IEEE Transactions on Affective Computing*, str. 147–161, 2011.

[41] J. Togelius, S. Karakovskiy i R. Baumgarten, „The 2009 Mario AI Competition," *Proceedings of the 6th IEEE International Conference on Computational Intelligence and Games*, IEEE, 2009., str. 1–8.

[42] K. O. Stanley, B. D. Bryant i R. Miikkulainen, „Real-Time Neuroevolution in the NERO Video Game," *IEEE Transactions on Evolutionary Computation*, str. 653–668, 2005.

[43] M. Booth, *The AI Systems of Left 4 Dead*, 2009.

[44] E. J. Hastings, R. K. Guha i K. O. Stanley, „Interactive Evolution of Particle Systems for Computer Graphics and Animation," *IEEE Transactions on Evolutionary Computation*, str. 418–432, 2009.

6. Popis slika