Proceduralno generiranje razina za računalnu igru korištenjem metoda umjetne inteligencije

Krmpotić, Andrija

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Andrija Krmpotić

PROCEDURALNO GENERIRANJE RAZINA ZA RAČUNALNU IGRU KORIŠTENJEM METODA UMJETNE INTELIGENCIJE

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Andrija Krmpotić

Matični broj: 0016127095

Studij: Informacijski sustavi

PROCEDURALNO GENERIRANJE RAZINA ZA RAČUNALNU IGRU KORIŠTENJEM METODA UMJETNE INTELIGENCIJE

ZAVRŠNI RAD

Mentor:

dr. sc. Bogdan Okreša Đurić

Varaždin, rujan 2021.

Andrija Krmpotić

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

U ovom završnom radu na detaljan se način prolazi kroz metode umjetne inteligencije u kreiranju razina te kako su se one razvijale kroz povijest u zadnjem desetljeću, to jest u razdoblju od 2010. do 2020. godine. Dotiče se pitanja: "Što je to razina videoigre Super Mario?". Objašnjava se teorija strojnog učenja. Objašnjene su metode poput učenja iz videozapisa, generiranje pomoću uzorka dizajna razine, generiranje iz memorije uz pomoć Markovljevog lanca te na kraju generiranje uz pomoć generativnih suprotstavljajućih mreža (eng. *General Adversarial Networks* - GAN). U praktičnom djelu rada govori se o implementaciji GAN-a koja je trenirana realnim razinama igre koje su izrađene od strane ljudi te je objašnjeno na koji način sam kod radi prilikom kreiranja razine videoigre Super Mario.

Ključne riječi: generiranje novih razina; umjetna inteligencija; Metode proceduralnog generiranja; SuperMario; računalne igre;

Sadržaj

1. Uvod

Proceduralno generiranje sadržaja (engl. *Procedural Content Generation* - PCG) u videoigrama počinje dobivati na značaju još u ranim 1980-im godinama. Ta kategorija pokriva kreiranje sadržaja sa ili bez ljudske interakcije, odnosno s pomoći ljudskog dizajnera ili bez nje. U ovom radu govori se o metodama kreacije sadržaja bez pomoći ljudskog dizajnera razina, konkretno, kreaciji razina za videoigru Super Mario. Super Mario razina ima puno parametara koji se mogu mjeriti i potom kreirati nove razine koje su drugačije od svih ostalih, a ujedno i bazirane na stvarnim razinama iz originalne Super Mario Bros. igre.

Razvijanje automatizirane kreacije razina Super Mario video-igre počelo je 2010. godine kada je Nintendo objavio natjecanje u automatskoj kreaciji razina. U natjecanju je sudjelovalo mnoštvo sudionika, ali samo natjecanje bilo je tek početak razvoja mnogih metoda, algoritama i znanstvenih radova na temu automatizacije izgradnje razina kako za video-igru Super Mario, tako i za ostale video-igre. 2012. godine metoda generiranja iz uzorka dizajna objavljena od strane Stevea Dahlskoga i Juliana Togeliusa. Zatim, 2016. godine objavljene su dvije metode. Metoda kreiranja sadržaja iz memorije putem Markovljevog lanca i Metoda kreiranja sadržaja iz videozapisa bazirana na mapiranju objekata u svakoj sličici videozapisa. Na poslijetku, najsuvremenija metoda jest metoda generiranja sadržaja putem korištenja generativnih suparničkih mreža (engl. *Generative Adversarial Networks* - GAN) o kojoj će se govoriti i u praktičnom dijelu rada.

2. Što su to video-igre?

U današnjem svijetu video-igre su izvor zabave za sve uzraste. Video-igre su jedna od vrsta ljudske interakcije s računalima ili konzolama koje su kreirane isključivo kako bi mogle pokretati razne video-igre. Iako velika većina svjetske populacije zna što su to video-igre, rijetko kada će se dobiti sama definicija video-igara. Sljedeću definiciju predložio je Esposito, a glasi ovako: "Video-igra je igra koju igramo zahvaljujući audiovizualnom aparatu te se može bazirati na priči. " [1] .

Danas postoje mnogi žanrovi igara poput strategija, igara preživljavanja (engl. *Survival games*), utrka i mnogih drugih. Ono što igru može činiti zanimljivom za široke mase jest njena priča, dizajn same igre, grafičke animacije unutar video-igre ili razni elementi fantazije koji se mogu pronaći u velikom broju video-igara. Svaka video-igra je dizajnirana kao oblik zabave te kao takva pobuđuje interes kod igrača da barem probaju zaigrati tu video-igru. Međutim, jedna igra je pobudila toliko velik interes da su se i znanstvenici osvrnuli na nju te je počeli proučavati. Ta video-igra danas nije samo jedna instanca, već cijeli serijal video-igara skupno nazvanih Super Mario. Super Mario je platformer tip igre, u kojoj igrači prelaze razine na način da se kreću, penju ili skaču po platformama postavljenim u razini, koja je svojim jednostavnim mehanikama i dizajnom pobudila velik svjetski interes. Isto tako, zbog tih jednostavnih mehanika, instanca video-igre Super Mario, nazvana Super Mario Bros. je video-igra podobna automatizaciji. Automatizacija ove video-igre može se raščlaniti na dva dijela:

1. **Automatizacija prolaženja kroz video-igru** - Kreiranje algoritama koji sam prolazi kroz razine video-igre Super Mario
2. **Kreiranje novih razina** - Kreiranje algoritama koji su sposobni rekonstruirati razine video-igre Super Mario uz obogaćivanje razina novim elementima i tako stvarati potpuno nove razine unutar video-igre SuperMario

Fokus ovog rada je drugi dio automatizacije, odnosno kreiranje novih razina. U nastavnim poglavljima detaljno su opisane metode kreiranja novih razina.

2.1. Što je to razina Super Mario video-igre?

Kroz mnoge računalne igre razlikuju se mnoge vrste razina, ali svaka razina ima početak, kraj te središnji dio koji je uvjetovan pravilima igre odnosno mehanikama igre kojima se igrač mora koristiti kako bi došao do završetka razine. Međutim, postavlja se pitanje koje su specifične karakteristike razine za video-igru Super Mario? Na to pitanje može se odgovoriti korištenjem MDA okvira [2], [3] (eng. *Mechanics-Dynamics-Aesthetics Framework*) kako bi se pobliže opisale specifikacije Super Mario razine.

2.1.1. Mehanike igre

U svakoj razini bilo koje igre, igrač koristi mehanike kako bi mogao prijeći razinu. Mehanike se mogu podijeliti na tri kategorije[2], [3]: Kretanje avatara igrača, sudari sa raznim objektima i stanja terminacije.

U slučaju Super Mario razine [2], [3] avatar igrača može se kretati u dvodimenzionalnom svijetu kreiranom po horizontalnoj osi X i vertikalnoj osi Y. Horizontalno kretanje ostvaruje se pomicanjem udesno ili ulijevo za točno određeni broj piksela za hodanje ili trčanje, dok se kretanje po vertikalnoj osi ostvaruje skakanjem koje je u ovakvim razinama ograničeno silama gravitacije. [2], [3]

Sljedeća kategorija mehanike igre su sudari sa raznim ostalim objektima postavljenim kroz razinu [2], [3], a ti sudari se također mogu podijeliti na tri kategorije: sudari s kockama, sudari sa NPC-jevima (engl. *Non-Playable Character*) te sudari sa raznim predmetima koje igrač može pokupiti. U slučaju sudara kocke i avatara igrača, avatar neće primiti štetu te se kocka neće promijeniti. Isto tako postoji mogućnost da igrač udari kocku s donje strane objekta i pritom promjeni njezino stanje ili ju trajno uništi. S druge strane, ako se igračev avatar sudari sa neprijateljem (NPC) tada igrač prima štetu ukoliko ne udari u objekt sa gornje strane što rezultira uništavanjem NPC-a osim u slučaju ako je taj NPC dizajniran tako da kontrira taj potez, primjer toga su kornjače koje imaju bodlje na leđima. Treća kategorija, odnosno predmeti koje igrač može pokupiti je dizajnirana tako da promjeni samog avatara igrača kada avatar dođe u koliziju sa njima, na primjer avatar može pokupiti SuperGljivu i time se povećati, što mu daje sposobnost da uništava blokove.

Zadnja kategorija mehanika igre jesu stanja u kojima igrač zaustavlja igru i događaju se određena negativna i pozitivna stanja. Negativna stanja [2], [3] su ona u kojima igrač gubi život te je postavljen nazad na početak razine ili je postavljen nazad na kontrolnu točku razine (eng. *Checkpoint*). Takva stanja događaju se kada avatar igrača dosegne preddefiniranu vrijednost vertikalne osi ili kada avatar u osnovnom stanju dođe u koliziju sa neprijateljima, odnosno bez poboljšanja. Pozitivno stanje jest ono u kojem igrač dolazi do objekta koji označava kraj razine te se razina smatra prijeđenom.

2.1.2. Dinamike igre

Dinamike igre su mogućnosti kojima se razini daje dubina igračkog iskustva, igraču se daju dodatne mogućnosti kako bi razinu učinili zanimljivijom ili kako bi se navelo igrača da radi određene stvari kako bi dobio određene nagrade ili ne bi izgubio nešto što je bitno za prijelaz same razine. Osnovne dinamike koje definiraju razinu Super Maria su sljedeće [2], [3]: Sakupljanje novčića, ubijanje neprijatelja, povezano skakanje s objekta na objekt. Sakupljanjem novčića igrač može ostvariti pogodnosti kao što su dodatni životi koji dozvoljavaju još jedan pokušaj ako igrač završi u negativnom stanju. Ukoliko igrač uspije u jednom skoku ubiti više neprijatelja, tada je nagrađen sa više bodova ili čak dodatnim životima ako je broj neprijatelja dovoljno velik. Ova dinamika ne mora se nužno odviti tako da igrač skače na neprijatelje već s određenim predmetima ili poboljšanjima može postići isti efekt. Isto tako dinamika igre de-

finirana kao povezano skakanje sa objekta na objekt znači da igrač može skakati od zidova, odbijati se od raznih ostalih objekata, što je posebno istaknuto u novijim instancama igre, te je ovom dinamikom dodana dubina samoj razini i na taj način igrač se može lakše povezati, odnosno "uroniti" u samu video-igru.

2.1.3. Estetika igre

Kada se govori o estetici Super Mario razine, tada se ona može podijeliti u 4 kategorije [2], [3], a to su osjećaj, elementi fantazije, izazov i otkrivanje. Razine su uglavnom napravljene tako da izgledaju ugodno oku, koristeći šarolik set boja te jasno izražene objekte s njihovim, lako prepoznatljivim, karakteristikama. Uz to se može napomenuti kako postoji opcija kod izrade razina da se uz svaku akciju doda određeni zvuk ili promjeni pozadinska muzika ako igrač ispuni određene uvjete.

Elementi fantazije lako su prepoznatljivi u ovakvim video-igrama jer je vrlo lako zaključiti kako avatar igrača Super Mario računalne igre ima nadljudske sposobnosti u osnovnom stanju avatara, a dodatni aspekt fantazije dolazi do izražaja kada avatar igrača pokupi određena poboljšanja. Isto tako mogu se zamijetiti elementi fantazije u samom dizajnu neprijatelja kao što su na primjer biljke koje ispaljuju vatrene kugle ili kornjače koje bacaju čekiće i tome slično.

Kako postoji mnogo razina, tako postoji i mnogo izazova koji se skrivaju u raznim razinama Super Mario igre. Od rješavanja logičkih zagonetki do utrke s ostalim igračima ili čak borbe s velikom količinom neprijatelja, igrač će se naći u mnoštvu različitih izazova koji video-oigru čine zanimljivijom. U istom svijetlu može se opisati i aspekt otkrivanja razine [2], [3]. Kreator razine unutar same razine može sakriti niz skrivenih prolaza ili dodatnih poboljšanja koja će biti dostupna samo igračima koji otkrivaju svaki kutak razine.

Slika 1: Početni zaslon video-igre Super Mario Bros.

3. Osnove strojnog učenja

Strojno učenje (engl. *Machine learning*)[4] može se opisati kao skup komputacijskih metoda koje koriste iskustvo kako bi predvidjele određene situacije ili poboljšale performanse u određenim situacijama. Iskustvo se može opisati kao skup podataka koji su se prikupili u prošlosti koje se raznim metodama mogu analizirati i potom prema njima predvidjeti buduće događaje. Strojno učenje [5] može se klasificirati kao kategorija umjetne inteligencije (engl. *Artificial intelligence*) koja omogućuje algoritmima predviđanje budućih događaja koji nisu specifično pretprogramirani u samom programu od strane čovjeka. Strojno učenje sastoji se od kreiranja preciznih i učinkovitih algoritama čija kvaliteta se mjeri u njihovim vremenskim i prostornim složenostima [6]. Uz navecedno koristi se i kompleksnost uzorka kako bi se moglo mjeriti koju veličinu uzorka moramo uzeti kako bi se dobilo precizne rezultate iz algoritama strojnog učenja za dane situacije. Generalno, tehnike strojnog učenja [4] su metode vođene podatcima koje kombiniraju osnovne koncepte računarstva s idejama iz vjerojatnosti, statistike i optimizacije.

Neki od problema koje se mogu riješavati strojnim učenjem su sljedeći [4, str. 2]:

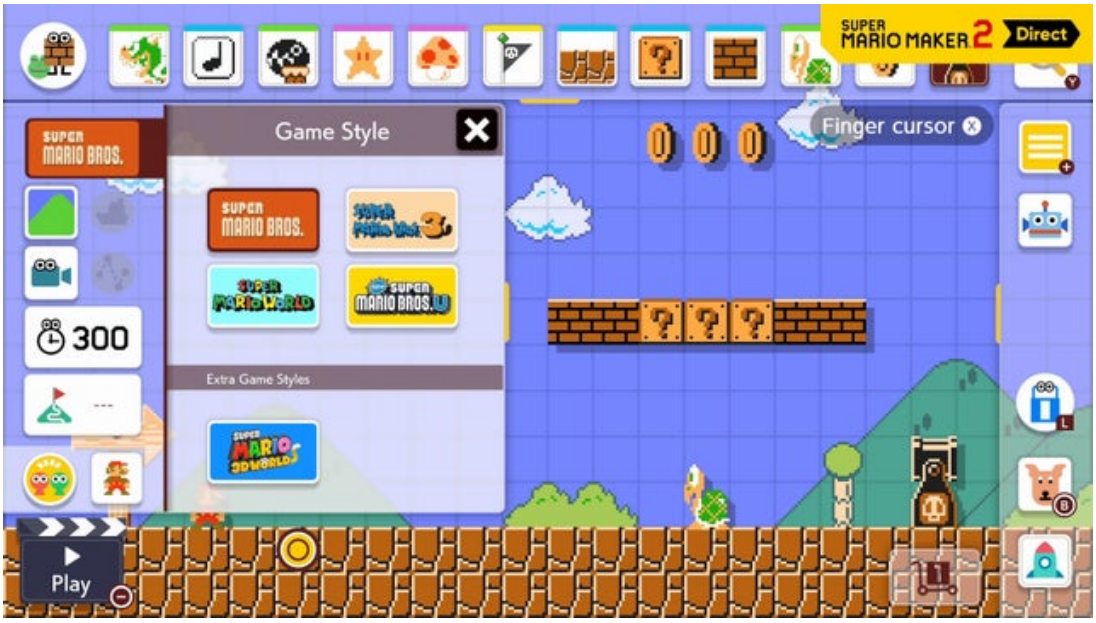
- **Obrada prirodnog jezika (engl. *Natural language processing* - NLP)** - Procesiranje jezika, završavanje rečenica, prepoznavanje i prevođenje riječi i sl.
- **Procesiranje govora** - Prepoznavanje govora iz zvučnih zapisa, glasovna provjera, identifikacija govornika i sl.
- **Računalni vid (engl. *Computer vision*)** - Prepoznavanje objekata, prepoznavanje lica, optičko prepoznavanje znakova (engl. *Optical character recognition*)
- **Računanje bioloških potreba** - Predviđanje razina vitamina i ostalih substanci neophodnih za život prema prošlom stanju organizma i sl.

Ova lista problema koji se mogu riješavati strojnim učenjem nema kraja. Postoji niz situacija koje se uz dovoljan broj faktora mogu predvidjeti strojnim učenjem u slučaju ako se za njih napravi dovoljno dobar model s dovoljno dobrim uzorcima.

Naravno, ovakav pristup može se koristiti i za trenutke zabave. Strojnim učenjem može se automatizirati procese poput dizajniranja razina unutar neke videoigre, osmišljavati izazove za društvene igre, kreirati pitanja za igre s više ljudi, a sve to može se osigurati tako da ne postoji ljudska interakcija i utrošak vremena i resursa za rješavanje problema ili kreiranje izazova.

4. Uvod u generiranje razina

Automatsko generiranje razina oduvijek je privlačilo mnoge kreatore videoigara iz razloga što je nemoguće napraviti neograničen broj razina koje su pretprogramirane od strane čovjeka prije nego što video-igra izađe na tržište, a konstantno dodavanje novih razina putem ažuriranja je skupa opcija i za kreatora, a i za potrošače koji tu videoigru kupuju i igraju. Neki kreatori video-igara su to riješili tako da su omogućili igračima da sami kreiraju razine te ih podijele s drugim igračima. Jedan takav primjer igre je Super Mario Maker 2 u kojoj igrači kreiraju razine sa svim postojećim blokovima iz svih prethodnim instanci igre Super Mario.



Slika 2: Zaslon vidljiv prilikom kreiranja nove razine unutar igre Super Marko Maker 2, preuzeto sa [7].

5. Metoda generiranja iz uzoraka dizajna

Uzorak dizajna, u kontekstu proceduralnog generiranja razina, skup je objekata poslo-ženih u točno određenom redoslijedu kako bi igrač morao napraviti određene kretnje kako bi prošao kroz tu sekciju odnosno uzorak danih objekata. U raznim igrama uzorci dizajna mogu se poistovjetiti sa rješenjima koja odgovaraju na probleme postavljene od strane mehanika igara. Isto tako može ih se gledati kao probleme, odnosno izazove koje igrač mora riješiti kako bi prešao razinu i na kraju završio videoigru. Sama ideja automatizacije stvaranja problema, odnosno razina, iz danih uzoraka je proces u kojem se koriste ograničenja provjere razine. Navedena ograničenja moraju se definirati kako se ne bi kreirala sekcija razine koju avatar igrača nije u mogućnosti prijeći. Time se omogućava da se razine proizvode same uz zadani set pravila koja definiraju kako slagati različite uzorke dizajna kako bi se dobila smislena cjelina koja se naziva igriva razina videoigre. [8]

Provođenje algoritma za proceduralno generiranje sadržaja nad definiranim uzorcima dizajna originalnih razina Super Maria je najdirektniji način kreiranja razina. Generatori kompozicijskog sadržaja svaki pojedini uzorak dizajna vide kao prostorno dizajnirani element koji ima mogućnost kombiniranja s ostalim uzorcima dizajna kako bi se dobila smislena cjelina odnosno jedna sekcija razine. Bitno je napomenuti kako se taj proces može raditi sa statičkim elementima, a može ga se i parametrizirati ukoliko je to potrebno.

Uzorci mogu biti spojeni [8] sekvencijalno u jednoj dimenziji, dvije dimenzije ili čak tri dimenzije. Procesom slaganja jednog uzorka u drugi uzorak, odnosno preslikavanjem jednog uzorka preko drugog, dobivaju se modulirani uzorci sastavljeni od jednog ili više uzoraka koji se tada mogu koristiti u specifičnim situacijama. Uzorci kreirani na taj način moraju biti smješteni u razinu tako da odgovaraju ranije postavljenim uzorcima kako bi razina izgledala što realnije.

Drugi način za slaganje uzoraka u proceduralnoj izradi razine jest da se na uzorke gleda kao na dijelove kompatibilne ili nekompatibilne s ostalim uzorcima. Sagledale bi se osobine svakog uzorka te bi se odlučilo sa kojim se ostalim uzorcima on slaže ili ne slaže te se tako stvara mapa poželjnih sekvenci uzoraka i nepoželjnih sekvenci uzoraka. Ovakav pristup vodi do većeg broja varijacija nego ranije spomenuti pristup koji koristi generatore kompozicijskog sadržaja, ali isto tako mora se uzeti u obzir da je taj proces teži za predviđanje novih sekcija i kompliciraniji računalu za izračunati.

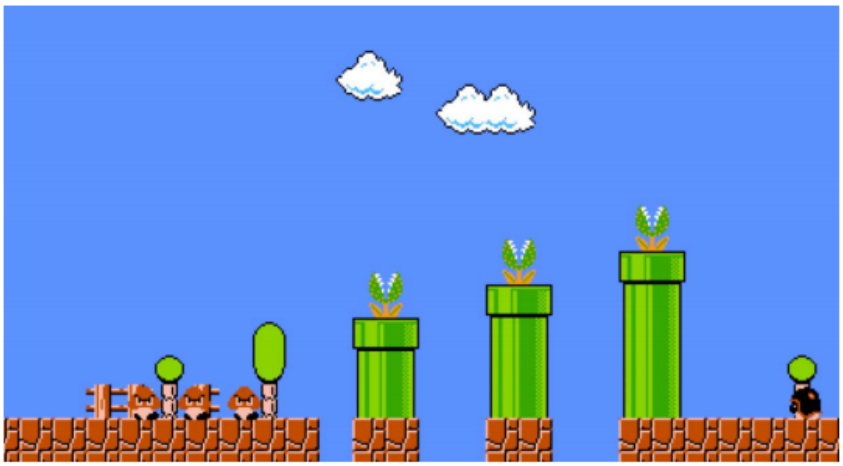
5.1. Tipovi otkrivenih uzoraka

U slučaju Super Mario igara uzorci se mogu podijeliti na pet većih kategorija uzoraka koje dijelimo na [8]: neprijatelje, praznine, doline, višestruke puteve i stepenice. Svaka od navedenih kategorija u sebi sadrži specifične uzorke koji su objašnjeni u nastavku.

U kategoriji neprijatelja postoji 5 različitih uzoraka u kojima se neprijatelji pojavljuju. Neprijatelj može kao uzorak doći sam, a isto tako postoje uzorci koji definiraju "horde" neprijatelja te je najviša uzastopna brojka u kojoj se mogu poaviti - četiri, odnosno sljedećim imenima *2-*

Horde, 3-Horde, 4-Horde. Zadnji uzorak kod skupine neprijatelja jest krovni neprijatelj. Takav uzorak postavljen je ispod viseće platforme u razini.

Sljedeća veća kategorija su praznine. Praznine se jasno mogu prepoznati po njihovim karakteristikama koje igrača jasno vode u negativno stanje tako što ga dovode ispod dozvoljene granice vertikalne osi razine. Jednostavnije rečeno, to su rupe u razini koje igraču predstavljaju izazov preskakanja, a samim time svrstavaju videoigru Super Mario u kategoriju *platformer* video-igara. Uzorci koji se mogu svrstati u ovu skupinu su [8]: rupa, višestruka rupa, varijabilna rupa, rupa sa neprijateljima i rupa sa stupom. Uzorak "rupa" se karakterizira jednim praznim blokom u podu razine, dok se višestruka rupa karakterizira uzastopnim rupama veličine jednog bloka, a između njih se nalazi jedan puni blok po kojem se igrač može kretati. S druge strane, varijabilna rupa, iako slična višestrukoj rupi, može imati varijabilan niz blokova i platformi, odnosno polja blokova po kojima se igrač može kretati. Rupa sa neprijateljima može se jasno izdvojiti prema tome što jedino ovaj uzorak ima letećeg neprijatelja iznad blokova koji su prazni. Zadnji uzorak ove skupine jest rupa sa stupom koju karakteriziraju cijevi ili vertikalno postavljeni blokovi na platformama koje se nalaze između rupa.



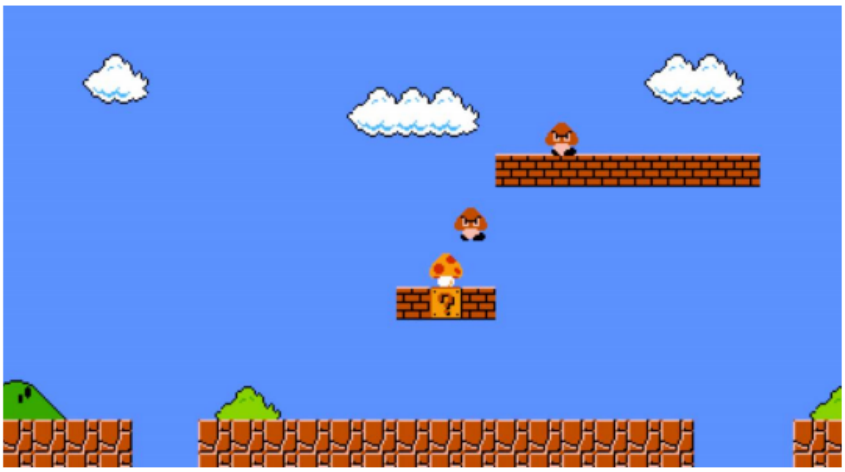
Slika 3: Primjer kombiniranja uzorka *3-horde* i uzorka "Rupa sa stupom", preuzeto iz [8].

Kategorija dolina je prazna platforma čiji su početak i kraj obilježeni preprekom, a može ih se podijeliti na 5 različitih uzoraka [8]: dolina, dolina sa cijevima, prazna dolina, dolina s neprijateljima, dolina s krovom. Klasična dolina karakterizirana je praznim prostorom omeđenim dvaju cjevima koje u sebi ne sadrže neprijatelje (najčešći slučaj u Super Mario razinama je takozvana *Piranha plant* biljka koja izlazi iz cijevi), dok dolina sa cijevima izgleda identično kao i obična dolina, ali sa bitnom razlikom jer sadrži neprijatelje u cijevima. Prazna dolina je identična kao i obična dolina, ali bitno je napomenuti kako prazna dolina ne mora nužno biti omeđena cijevima već i vertikalnim blokovima, dok dolina sa neprijateljima sadrži i dodatne prepreke u obliku neprijatelja. Dolina sa krovom karakterizirana je horizontalnom linijom blokova iznad same doline.

Stepenice su definirane postavljanjem blokova tako da simboliziraju uspon ili spust avatara. Stepenice se mogu podijeliti na pet različitih uzoraka: stepenice prema gore, stepenice prema dolje, prazna dolina sa stepenicama, dolina sa stepenicama i neprijateljima te dolina sa stepenicama i rupom. Kada su doline omeđene stepenicama tada se govori o praznoj dolini sa

stepenicama, a ako se u njoj nalazi neprijatelj tada se govori o dolini sa stepenicama i neprijateljima. Dolinu sa stepenicama i rupom karakterizira nedostatak platforme po kojoj se igrač može kretati u dolini između stepenica.

Super Mario video-igra ne bi bila tako zanimljiva bez uzoraka koji se kategoriziraju kao višestruki putevi. Postoje tri različita uzorka višestrukih putova. Dvostruki put koji u sebi sadrži platformu u zraku koja dijeli dvije sekcije. Prilikom susreta sa ovakvim uzorkom igrač može odabrati kojim će putem prijeći prepreku. Trostruki put sličan je dvostrukom putu, no dodaje još jedan put kojim igrač može krenuti. Najkompleksniji uzorak jest uzorak rizika i nagrade. Takav uzorak sastoji se od višestrukog puta u kojem jedan ili više puteva vodi prema nagradi, ali je opasnost veća, bila opasnost od neprijatelja ili od praznine, a nagrada se može predstaviti u obliku novčića ili poboljšanja za avatara igrača. Ostali putevi su lakši, ali se na njima ne nalazi nikakva ili vrlo mala nagrada u obliku novčića.



Slika 4: Primjer kombiniranja uzorka "Rupa" i uzorka "Rizik i nagrada", preuzeto sa [8]

5.2. Slaganje uzoraka

Kako bi se uzorci složili smislenim redom, razinu se mora sagledati ne iz perspektive igrača, već iz perspektive avatara, odnosno Super Maria. Iz perspektive igrača avatar se kreće lijevo i desno, dok se iz perspektive Super Maria on kreće uvijek naprijed i skače na objekte određene visine. Iz ovakve perspektive, sadržaj originalne videoigre Super Mario se može sagledati kao niz vertikalnih presjeka širine jednog bloka i visine 14 blokova što je standardna visina. Kombinirajući ovakve vertikalne presjeke dobivaju se različiti uzorci koji su opisani u protekloj sekciji. Genotip, odnosno genska struktura razine predstavljena je kao znakovni niz duljine 200 znakova koji koristi abecedu od 24 slova. Ovih 24 presjeka označeni slovima abecede su reprezentativni uzorci uzoraka koji su prethodno opisani te su uzeti iz originalne videoigre Super Mario. Bitno je napomenuti da je većina ovih uzoraka uzeta iz razine 1.1 i 1.2. [9]

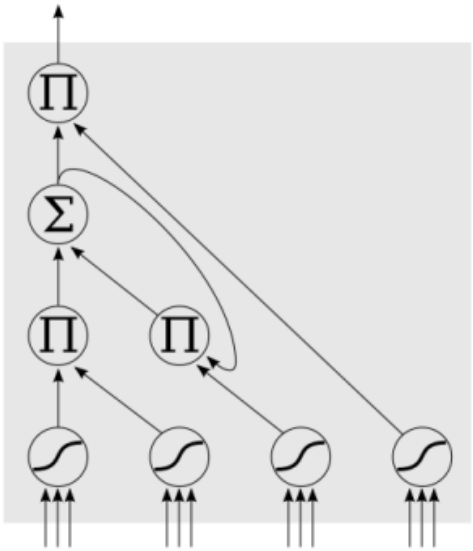
6. Metoda generiranja iz memorije

Kako bi se opisala ova metoda, prvo se ukratko mora opisati što je to Markovljev lanac i Monte Carlo metoda pretraživanja. Markovljev lanac [10] je stohastički model koji opisuje sekvencu mogućih događaja kod kojih se mogućnost pojavljivanja definira isključivo prethodnim događajima. Bitno je napomenuti kako Markovljev lanac uzima događaje koji se odvijaju u realnom vremenu te prema njima predviđa budućnost. To svojstvo uzimanja malog dijela prethodnih događaja kako bi se predvidjeli budući događaji naziva se Markovljevo svojstvo poznatije pod nazivom nepamtljivost (engl. *Memorylessness*) koje, kako i sam engleski naziv govori, ne treba cijelu povijest događanja kako bi predvidio buduće događaje. [10]

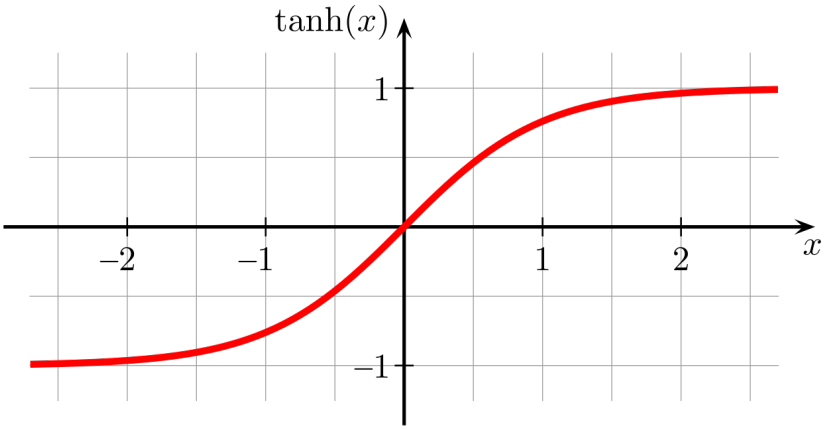
Monte Carlo metoda stabla pretraživanja [11] je algoritam koji pronalazi sve moguće ishode nakon trenutnog događaja. Dobivene ishode mapira u obliku stabla koje se neprestano ažurira novim podatcima, odnosno ishodima ili u našem slučaju dijelovima razine. Jednostavno rečeno, to je metoda kojom se traži svaki mogući ishod igre te potom odabire najbolji, odnosno finalni ishod. Sama metoda vizualizira se pomoću stabla pretraživanja.

Kako bi se kreirala razina pomoću ove metode, mora se koristiti duga kratkoročna memorija (engl. *Long Short-Term Memory*) (u daljnjem tekstu LSTM). Umjetne LSTM neuronske mreže [12] sadrže blokove koji daju kontekst programu o tome kako primati ulazne vrijednosti te kako kreirati izlazne vrijednosti. Sam naziv duga kratkoročna memorija dolazi od svojstva da se iz programa koji koristi strukturu kratkoročne memorije kreira dugoročnija memorija. Jedan blok LSTM-a [13] sadrži kompleksne komponente poput težinskih ulaznih vrijednosti, funkcija aktivacije, ulaznih vrijednosti prethodnih LSTM blokova i eventualnih izlaznih vrijednosti. Preteča LSTM umjetnih neuronskih mreža bile su rekurentne neuronske mreže (engl. *Reccurent Neural Networks*). Kod rekurentnih neuronskih mreža [13] rubni vektori nisu bili povezani od ulaza preko skrivenih slojeva do izlaza kao kod standardnih neuronskih mreža, već su bili povratno prošireni, odnosno povezani i sa samim sobom kroz vrijeme što je značilo da se povratno širenje (engl. *Back-propagation*) nije događalo samo između različitih čvorova već i u vremenu u kojem se proces odvijao. Ozbiljan problem kod ovakvih neuronskih mreža poznat je pod nazivom "*the vanishing gradient problem*", odnosno problem nestajućeg gradijenta u kojem se gradijent grešaka vrlo brzo rasipao tijekom vremena izvođenja procesa. Kod LSTM neuronskih mreža taj problem riješili su Hochreiter i Schmidhuber [14] kada su uveli dodatne čvorove u neuronsku mrežu koji su služili kao mehanizam memorije koji je mreži govorio kada da nešto zapamti, a kada da to ne zapamti.

LSTM neuronska mreža [13] funkcionira kao standardna neuronska mreža. Na donjim čvorovima nalaze se ulazni podatci koji prolaze kroz neke nelinearne funkcije. Najčešće se koriste sigmoidne funkcije koje se karakteriziraju krivuljom poput slova S. Jedna od sigmoidnih funkcija koja se može uzeti kao primjer je hiperbolička tangencijalna funkcija koja se može vidjeti na slici broj 5. Zatim dolaze čvorovi koji samo preuzimaju njihove izlaze (Π) te ih predaju sljedećem tipu čvorova (Σ) u kojem se obrađeni podatci zbrajaju. S takvim pogledom na LSTM blok može se reći da je skroz lijevi čvor na slici 6 glavni ulaz podataka, a čvor desno od njega služi kao ulazna vrata s obzirom na to da se množi s podatcima unutar prvog lijevog čvora.



Slika 5: Grafički prikaz LSTM bloka, preuzeto iz [13]



Slika 6: Grafički prikaz funkcije hiperboličke tangente u koordinatnom sustavu

Kombiniranjem tih dvaju ulaza dobiva se mehanizam koji propušta ulazni podatak kada je umnožak blizu brojke 1, a ne propušta podatak kada je umnožak blizu 0. Na sličan način sa desne strane, najdesniji čvor služi kao izlazna vrata koja odlučuju hoće li se sadržaj ostalih čvorova prve razine proslijediti čvorovima veće razine. Čvor označen simbolom Σ ima ulogu memorije. U tom čvoru string se pridodaje znakovnom nizu koji karakterizira razinu i sam po sebi ne gubi vrijednosti kroz vrijeme nego se ažurira sa podatcima iz najdesnijeg čvora LSTM bloka koji služi kao vrata zaboravljanja, odnosno koji dio podataka ne treba pamtiti. Sam znakovni niz na kraju mora biti formata 10x28x14 jer postoji 10 različitih tipova blokova o kojima će se više reći kasnije, 28 blokova širine jednog zaslona kojeg igrač vidi u jednoj instanci vremena i 14 blokova visine zaslona. [13]

Ovakvi LSTM blokovi mogu se formirati u više slojeva sa svojstvom da u svakom sloju može postojati više LSTM blokova. Ulazni dio mreže sastoji se od *One-Hot* enkodiranja u kojem svaki različiti element ima svoju binarnu zastavicu postavljenu na 1 ako je element odabran, dok su sve ostale binarne zastavice postavljene na 0. Zadnji LSTM sloj u strukturi povezuje se

sa *SoftMax* slojem koji služi kao podjela vjerojatnosti po kategorijama prema *One-Hot* enkodiranju. LSTM blokovi su najčešće korišteni kako bi predvidjeli sljedeći element u nizu. Kako bi ovaj način radio, podatci moraju biti jedan niz znakova. Ako bi se koristio tip podataka kao što je navedeno u prethodnom odlomku, odnosno tip znakovnog niza i kada bi se podatci smatrali kao vertikalni presjeci razine koji bi se slagali progresivno s lijeve na desnu stranu tada bi se uvidjela 2 problema:

- Graditelj razine mogao bi samo reproducirati vertikalne presjeke koji se nalaze u setu podataka s kojima se trenirao model, što bi značilo da ne može koristiti prije neviđeni vertikalni presjek
- Ovaj pristup uvelike bi povećao količinu ulaznih podataka.

Kako bi se adresirali ovi problemi, razinu Super Maria ne smije se gledati kao vertikalne presjeke već se mora sagledati svaki različiti element kao jedan dio niza odnosno znak u znakovnom nizu. Prema [13] postoje sljedeći tipovi elemenata:

- **Čvrsti element** - bilo koji element koji je čvrst i nema nikakve interakcije s igračem ili ostalim pokretnim elementima. Ne razlikuje se tip čvrstog elementa
- **Neprijatelj** - Običan neprijatelj, bez razlikovanja neprijatelja
- **Element koji se može uništiti** - bilo koji blok koji se može uništiti djelovanjem pojačanja igrača ili pomoću neprijatelja
- **?-blok sa novčićem** - objekt u koji avatar igrača može udariti te na sebi ima naslikan znak "?" koji sadrži jedan ili više novčića
- **?-blok sa pojačanjem** - objekt u koji avatar igrača može udariti te na sebi ima naslikan znak "?" koji sadrži pojačanje
- **Novčić**
- **Vrh topa** - Vrh topa koji ispucava neprijatelja po imenu *Bullet Bill*
- **Trup topa** - trup topa koji ispucava neprijatelja po imenu *Bullet Bill*
- **Lijeva strana trupa cijevi**
- **Desna strana trupa cijevi**
- **Lijevi vrh cijevi**
- **Desni vrh cijevi**
- **Prazan element** - Nedostatak svih ostalih elemenata, odnosno prazan prostor

Iz nabrojanih stavki može se zaključiti kako su cijevi i topovi u razinama Super Mario video-igre raščlanjeni na više dijelova iz razloga što je svaki taj dio drugačiji te se drugačije enkodira. Razlog tome jest varijabilna duljina cijevi i topova te se mora precizno enkodirati na

kojoj poziciji se nalazi vrh cijevi. Svaki od navedenih tipova elemenata može se sagledati kao jedan znak u znakovnom nizu, ali *platformer* tip videoigara ima dvije dimenzije, a ne jednu kao znakovni niz. S obzirom da klasična razina videoigre Super Mario ima između 200 i 500 blokova širine i 14 blokova visine, LSTM blokovi trebaju pamtiti znakovni niz barem te duljine. Pamćenjem tako dugog znakovnog niza otežava predviđanje budućeg elementa putem LSTM blokova. Kako bi se našao najefektivniji način slaganja različitih blokova u znakovni niz mora se postaviti 3 pitanja [13]:

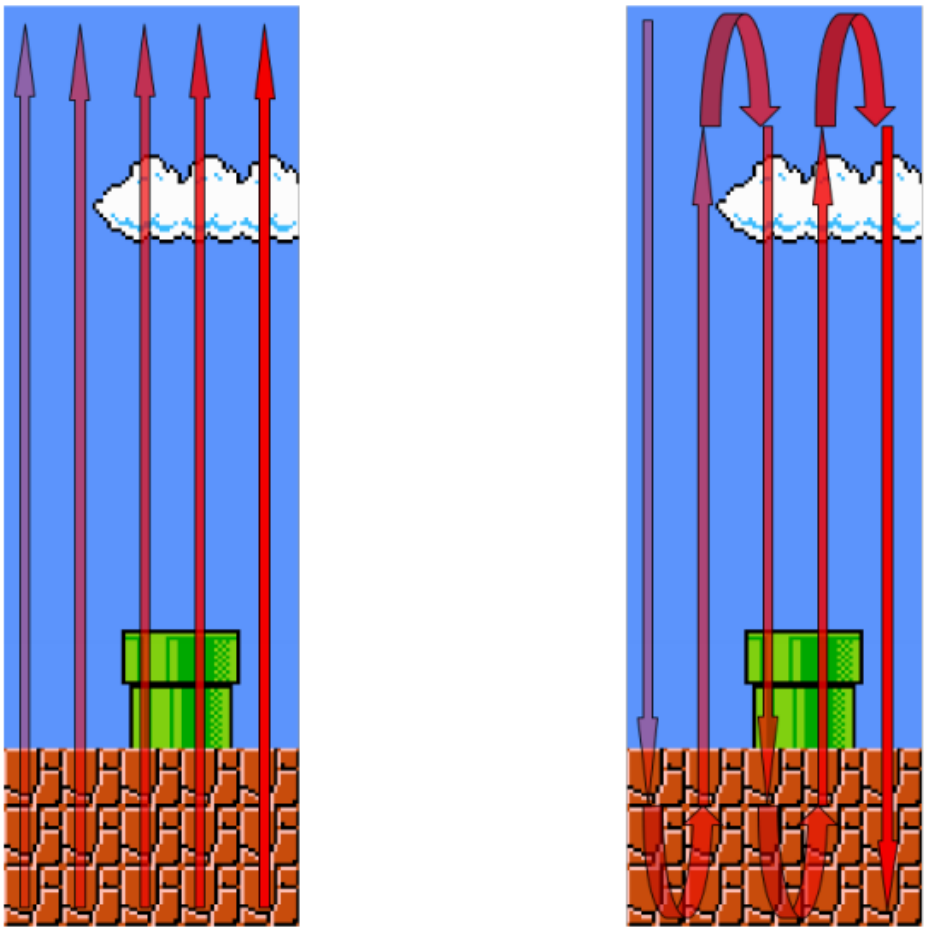
- Metoda odozdo prema gore ili metoda *Snaking*-a?
- Sadrži li blok informacije o putanji avatara igrača?
- Sadrži li blok informaciju o dubini stupca?

6.1. Odozdo prema gore ili *Snaking*

Logika odabira ove dvije metode je jednostavna. Uzimajući u obzir da se avatar igrača kroz razinu kreće horizontalno i da se novi dijelovi razine učitavaju sa desne strane od avatara igrača ne mogu se implementirati nikakve druge metode učitavanja znakovnog niza. Ukoliko bi se uzela bilo koja druga metoda tada bi učitavanje same razine bilo dugačko jer bi se morala učitati cijela razina odjednom za razliku od vertikalnog učitavanja razine stupac po stupac kako se avatar igrača kreće kroz nju.

Kako se promatraju razine, prvo se gledaju vertikalni presjeci, a zatim horizontalni poredak. Postavlja se prvo pitanje: Trebaju li se gledati blokovi odozdo prema gore ili odozgo prema dolje? Potencijalno je bolje zapisivati znakove u niz odozdo prema gore jer je intenzitet čvrstih blokova puno veći u donjem dijelu razine kako se tamo nalaze sve platforme po kojima se igrač može kretati. Nadalje, kako bi se naglasio prijelaz iz jednog stupca u drugi postavlja se poseban znak na kraj niza jednog stupca.

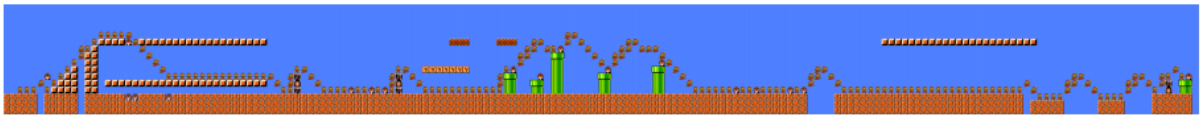
Postoji i manje intuitivna, ali logičnija metoda postavljanja blokova u niz, a naziva se *snaking* metoda. Kod ove metode nakon svakog stupca mijenja se metoda pisanja blokova u niz, odnosno između svakog stupca se alternira između metode odozdo prema gore i metode odozgo prema dolje. Ovakav pristup je logičniji jer se pokazalo kako se elementi koji se sastoje od više blokova, odnosno nisu jedan blok, mogu bolje prikazati znakovno odnosno mogu se bolje iščitati kao cjelina unutar znakovnog niza. Metoda putanje *Snaking*-a može se primijetiti na slici broj 7.



Slika 7: Razlika pisanja znakova u niz metodom odozdo prema gore (lijevo) i metodom *Snaking*-a (desno), preuzeto iz [13]

6.2. Informacije o putanji

Najveći problem u autonomnoj izradi razina predstavlja se u obliku igrivosti izrađene razine, drugim riječima može li igrač ikako doći do kraja razine odnosno krajnje prepreke koja donosi pozitivno terminirajuće stanje. Snodgrass i Ontanon [13], [15] pokazuju kako je prije korištenja informacija o putanji, kojom je avatar igrača prolazio, samo pola razina bilo igrivo, odnosno moglo ih se prijeći. Uzme li se to u obzir i u treniranje algoritma se ubaci i informacije o putanji avatara igrača kroz razinu, dobiva se generator razina koji je puno precizniji u izradi razina koje igrač može proći jer tada generator više ne kreira samo geometriju oko igrača već i put kojim igrač mora proći kako bi završio razinu. Igračeva putanja dobiva se iz prethodno testiranih razina, odnosno gotovih razina koje su bilježile putanju igračevog avatara i zatim te podatke dodale u znakovni niz. To se može postići umetanjem specijalnog znaka na prazna mjesta u znakovnom nizu kojeg generira LSTM mreža na mjestima gdje je igrač prošao, odnosno bilježi se igračeva putanja unutar znakovnog niza. [13]



Slika 8: Primjer putanje avatara kroz razinu, preuzeto sa [13]

6.3. Informacija o dubini stupca

Problem koji se pokušava riješiti enkodiranjem informacija o dubini stupca jest taj da LSTM mreža može pamtiti samo 12 unazad kreiranih stupaca, odnosno vertikalnih presjeka. Kako u Super Mario razinama postoje i strukture koje su povezane i veće od 12 stupaca, mora se razviti mehanizam koji će LSTM mrežama dopuštati brojanje stupaca, odnosno dubinu razine. Ovaj problem su Adam Summervile i Michael Mateas [13] riješili umetanjem jednog specijalnog znaka u znakovni niz razine, povećavajući broj tih specijalnih znakova svakih pet stupaca. Drugim riječima, prvih pet stupaca nije imalo nikakav znak, drugih 5 su imali po jedan specijalni znak na kraju stupca, trećih pet su imali po dva specijalna znaka na kraju svakog stupca i tome slično. Na takav način omogućeno je brojanje stupaca i kreiranje većeg intenziteta barijera kako se bližio kraj razine, što je igraču davalo dojam da se razina ponaša kao i normalna razina sa povećanjem težine prema kraju razine.

6.4. Optimalno rješenje metode

Korištenjem tri prethodno navedene stavke dobivaju se najmanja odstupanja od željenog ishoda te se može reći da su uvođenje igračeve putanje unutar znakovnog niza, zapisivanje broja stupaca u znakovnom nizu i metoda *Snaking*-a uvelike pridonijeli preciznijem generiranju Super Mario razina pomoću LSTM neuronskih mreža. Isto tako, korištenjem sva tri navedena aspekta dobiva se ne samo razina koja je igriva, već razina koja igraču osigurava iskustvo realne razine kreirane od strane čovjeka.

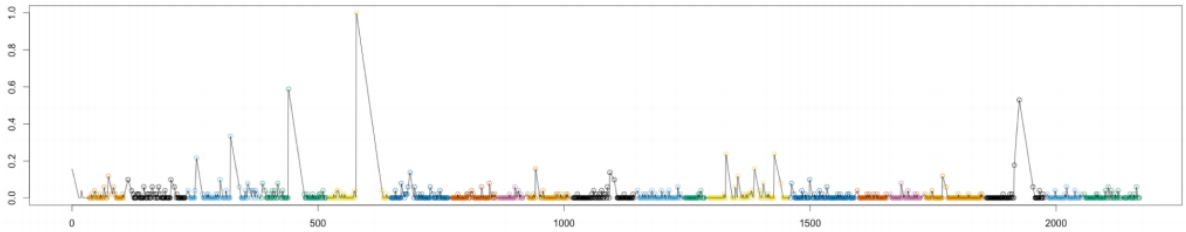
7. Metoda generiranja sadržaja iz videa

Metodom generiranja sadržaja iz videozapisa dobiva se dodatna dimenzija učenja. Uz proučavanje kako je razina sagrađena iz analize svih sličica sadržanih u sekundi videa (engl. *frames-per-second*) još se dobivaju i podatci o reakciji igrača na zadane izazove postavljene od strane razine. Ova metoda može se primjeniti ne samo za franšizu igara Super Mario Brothers, već i za ostale videoigre koje spadaju u kategoriju *platformera*. Dva bitna fokusa za ovu metodu su [16]:

1. Odlučivanje o tome što algoritam mora naučiti o samom izgledu razine
2. Reprezentativan uzorak podjele elemenata u ponovno upotrebljivoj formi koja čini jednu generaciju.

7.1. Kategorizacija sekcija razine

Kako bi se došlo do zanimljivog sadržaja s kojim bi se moglo učiti algoritam koji kreira razine, mora se odrediti koji dijelovi razine su zanimljivi i potencijalni kandidati za skup podataka za treniranje algoritma kreacije razine. Ovim procesom određuju se određeni dijelovi razine koji imaju povećanu vrijednost interakcije [16] (engl. *interaction value*), odnosno dijelovi razine u kojima igrač provodi najviše vremena u pokretu. Ovaj podatak koristi se kao metrika zanimljivosti dijela razine te njegove kompleksnosti. Strojno prepoznavanje slika je uznapredovalo do te razine da alati poput OpenCV [17] daju izvrsne rezultate za jednostavne vizualne zadatke. Klasične igre tipa platformer lako su čitljive strojnom vidu (engl. *Machine vision*) zbog regularnosti i konzistentnosti pojave objekata na ekranu. Ova dva svojstva mogu se karakterizirati kao popločavanje pozadine objektima (engl. *tiling*), dok se individualni objekt naziva spritom ili spritetom (engl. *sprite*). Sprite je dvodimenzionalna grafika koja se koristi za oslikavanje pojedinog objekta unutar razine. Prednost ovakve kategorizacije jest to što se može definirati svaki određeni oblik pomoću usporedbe sa bazom od 102 različita sprita koji se nalaze u Super Mario igrama. Iako je ovaj proces prepoznavanja objekata iz videozapisa zahtjevan i dugotrajan, on se samo treba dogoditi jednom kako bi alat OpenCV automatski dalje prepoznavao objekte istog tipa za beskonačno dugačku snimku videozapisa. Pomoću ovog alata može se definirati koji objekt se nalazi na kojim koodrinatama u razini te se prema tome može uzimati razne uzorke razina i od njih kombinirati nove. Nedostatak kod ove metode je taj što se za prepoznavanje u ovom alatu moraju koristiti isključivo snimke na kojima igrač ne dolazi u negativno terminirajuće stanje već prolazi razinu bez greške, odnosno dolazi do cilja iz prvog pokušaja.



Slika 9: Na x-osi nalazi se broj pločica fiskne dužine, na y-osi nalazi se intenzitet vrijednosti, a boja označava različite sekcije razine, preuzeto iz [16]

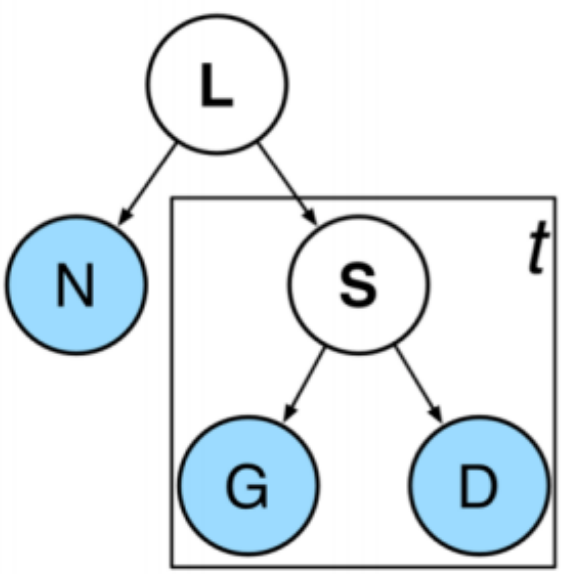
Većina igara ima varijaciju u svojim razinama, a mjerenjem upravo te varijacije možemo pratiti vrijeme koje je potrebno da se sekcija razine prođe. Sekcijsko vrijeme mjeri se u broju sličica (engl. *frames*) koje igrač provodi na određenoj sekciji nivoa. Na grafu na slici 9 može se uočiti kako su bojom zabilježene različite sekcije razine dok je cijeli graf preslika jednog videozapisa koji je proveden kroz alat OpenCV. Nadalje, sekcije razine kategoriziraju se po njihovoj interaktivnoj vrijednosti, a identificiraju se samo one sekcije koje imaju iznadprosječan faktor interaktivne vrijednosti iz razloga što su te sekcije najzanimljivije igraču ili kompleksnije od ostalih sekcija s manjim faktorom interaktivne vrijednosti te stoga služe kao fokus sekcije za treniranje algoritma kreiranja.

U svojem radu Guzdial i Riedl [16] iskazuju kako su pronašli 21 grupu sličica (engl. *frames*). Svaka grupa varira od 2 do 250 sličica sa medijanom veličine 35 sličica. 18 od tih grupa ima jasno definirane teme koje su česte u Super Mario razinama poput podvodne teme, podzemne teme, šumske teme i slično. Samo 3 grupe nemaju jasno definirane teme te od njih treba napraviti manje grupe pomoću vjerojatnosnog modeliranja (engl. *Probabilistic modeling*).

7.2. Konstrukcija vjerojatnosnog modela

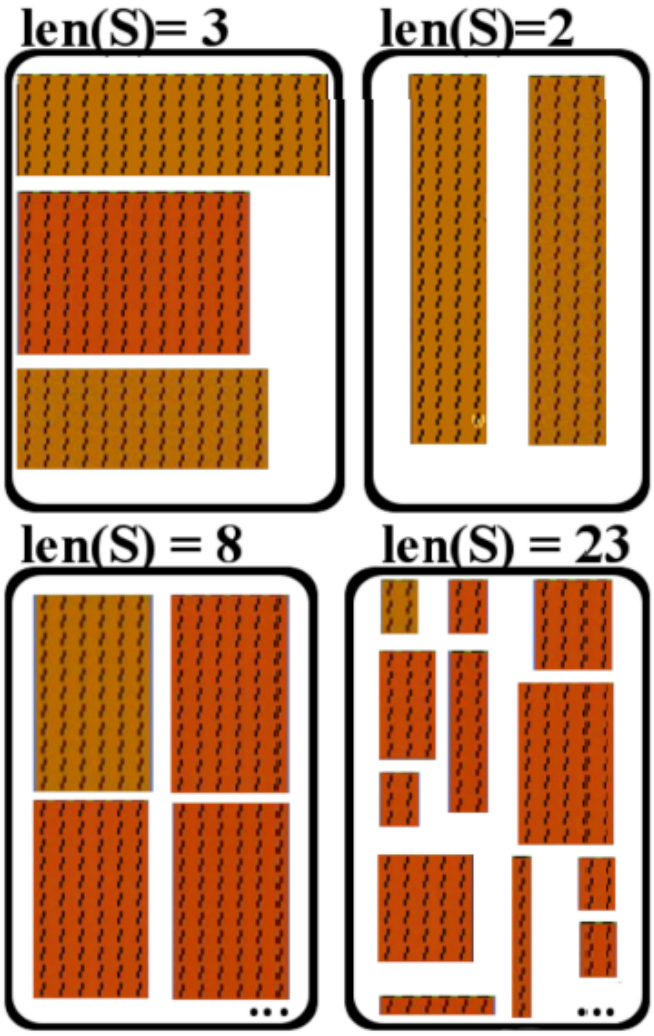
Znanje naučeno iz grupa sličica spomenutih u prethodnom odlomku može se prikazati grafički vjerojatnosni model koji povezuje određene skupine objekata sa sekcijama visoke interaktivne vrijednosti. Ovaj model može se koristiti za konstrukciju novih sekcija razine jednako interaktivnih i zanimljivih kao i one s kojima je model treniran. Guzdialov i Riedlov model [16] rađen je po uzoru na vjerojatnosni model Kalogerakisa [18] koji određuje skup pravila odnosno vrijednosti skrivenih ili latentnih varijabli koje predstavljaju pravila. Guzdial i Riedl koriste iste pretpostavke, međutim njihov model se uvelike razlikuje od Kalogerakisovog modela. Njihov model je rađen na način da što manje koristi informacije kreirane od strane ljudi, a ključna razlika jest to što je njihov model dvodimeznionalan za razliku od Kalogerakisovog trodimenzionalnog modela te mu je i struktura uvelike drugačija.

Čvor L, vidljiv na slici 10, označava stil dizajniranja razine, dok čvor S označava stil oblikovanja pojedinog sprita *t*. U čvoru N nalazi se informacija o broju različitih objekata koji se nalaze u početnoj sekciji razine. Informacije o koordinatama svih dijelova jednog objekta nalaze se u čvoru G, a sve informacije koje upućuju na relacije između objekta *t* i ostalih objekata nalaze se u čvoru D. Model prema slici koristi čvorove G, D i N kako bi sakupio jedinstvene informacije iz kojih se zatim dobivaju latentne, odnosno skrivene informacije u čvorovima L i S.



Slika 10: Vjerojatnosni model Guzdial i Riedl, preuzeto iz [16]

U čvoru G mogu se nalaziti i duplikati koji se u kasnijim koracima diseminiraju na specifične elemente, ali bitno je napomenuti da objekte može činiti više spritova, odnosno više uzoraka oblika, dok se u korespondentnom čvoru D nalaze sve informacije u obliku vektora (x, y) od lijevog gornjeg kraja objekta koji se nalazi u G čvoru pa sve do gornjeg lijevog kuta svih ostalih objekata koji se nalaze na trenutnom ekranu igrača. Zadnji čvor koji se može dobiti analizom podataka iz videozapisa jest čvor N. U njemu se nalazi broj svih spritova koji se nalaze na trenutnom ekranu igrača, odnosno broj svih uzoraka koji se nalaze na ekranu u danom vremenu. Ova informacija služi kao nit vodilja kod generiranja sadržaja visokog faktora interaktivne vrijednosti.

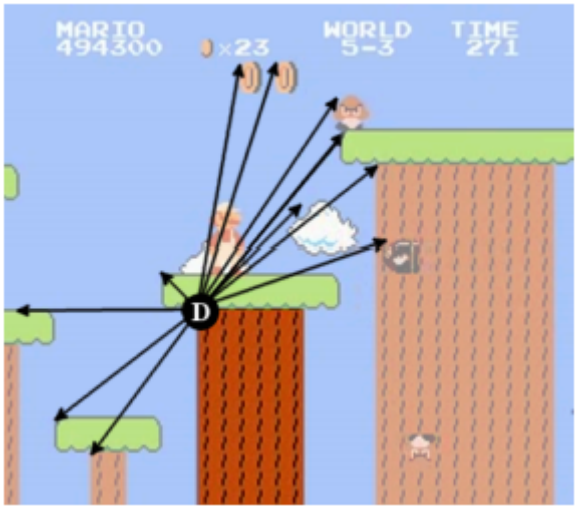


Slika 11: Reprezentativni uzorak različita čvora tipa S s obzirom na informacije o oblicima, preuzeto iz [16].

Latentne varijable [16] sadrže pravila koja se koriste kod konstrukcije razine. Čvor S sadrži kategoriju oblika i njihovo relativno postavljanje unutar razine, a te informacije dobivaju se derivacijom informacija iz para čvorova (G, D) za određeni objekt *t*. S čvor dobiva se na sljedeći način. Sustav grupira nakupine (G, D) parova koristeći *K-means++* algoritam za klasteriranje (engl. *clustering*) [19] gdje je K procijenjen kroz omjer izobličenja (engl. *distortion ratio*). Udaljenosti između dva G čvora dolaze od uređene udaljenosti između dva različita oblika istog tipa *t*, a udaljenost između dva čvora D dobivaju se zbrojem svih iteracija svih sortiranih parova koordinata (x, y) vrijednosti i razliku između relacijskih vektora. Ovim postupkom osigurava se da dva čvora G sa istim oblicima ne završe u istoj kategoriji jer se konzistentno pojavljuju u različitim relativnim pozicijama.

7.3. Generiranje sekcija razine

Generiranje dijela razine [16] je proces zadovoljavanja postavljenih pravila slaganjem parova G i D čvorova koji se odabiru na temelju njihove kompatibilnosti i potrebnih zahtjeva sekcije. Sam algoritam izrade sekcije počinje sa generiranjem prazne sekcije razine te zatim rekurzivnim dodavanjem uređenih parova (G, D) čvorova sve dok se kriterij zaustavljanja ne zadovolji. Algoritam nakon kreiranja prazne sekcije uzima inicijalni par (G, D) čvorova kao ulazni podatak te zatim probava sve moguće kombinacije sa ostalim (G, D) parovima čvorova i odabire koji čvor najbolje zadovoljava postavljene kriterije.



Slika 12: Označeni trup "stabla" kao objekt čvora G te svi vektori relacija sadržani u čvoru D, preuzeto iz [16].

Kada je par čvorova (G, D) dodan u sekciju, algoritmom pohlepe [20] (engl. *greed algorithm*) traži se drugi sljedeći čvor koji bi se najbolje slagao sa ostalim čvorovima. Algoritam [16] uvijek odabire dva moguća kandidata. Prvi kandidat je onaj objekt koji u sebi sadrži najpoželjniji oblik u sekvenci odnosno najbliži zapisani par u stablu odlučivanja, a drugi objekt koji je kandidat za ubacivanje u sekciju je onaj koji ima najveću vjerojatnost pojavljivanja unutar sekcije. Jednom kada je tip lika koji odgovara kriterijima za kandidata odabran, algoritam prolazi kroz sve S čvorove koji u sebi sadrže taj oblik objekta te se bira onaj par čvorova (G, D) koji najviše odgovara zahtjevima. Algoritam prestaje s dodavanjem čvorova kada se broj objekata približi ili bude jednak broju objekata zapisanom u inicijalnom objektu *t* u njegovom najbližem čvoru N [16]. Na kraju se dobiva izlaz u obliku jednog seta uređenih parova (G, D) čvorova koji se zatim prevodi u listu spritova i njihovih dvodimenzionalnih koordinata na način da se preslikava tip sprita po uzorku zapisanom u odgovarajućem G čvoru. Zatim se provjerava uzorak po kojem je rađena sekcija sa novonastalom sekcijom kako bi se osiguralo da ne postoji duplikat sekcije.

7.4. Procjena sekcije

Kako bi ocjenili kreaciju njihovog algoritma, Guzdial i Rield su uveli dvije varijable p_E i p_C [16]. p_C simbolizira igrivost određene sekcije, odnosno da li se sekcija može prijeći, dok varijabla p_E mjeri stil sekcije razine, odnosno kompoziciju komponenti sekcije. Prema Guzdialu i Riedlu, sekcija je igriva ako su zadovoljena tri kriterija [16]:

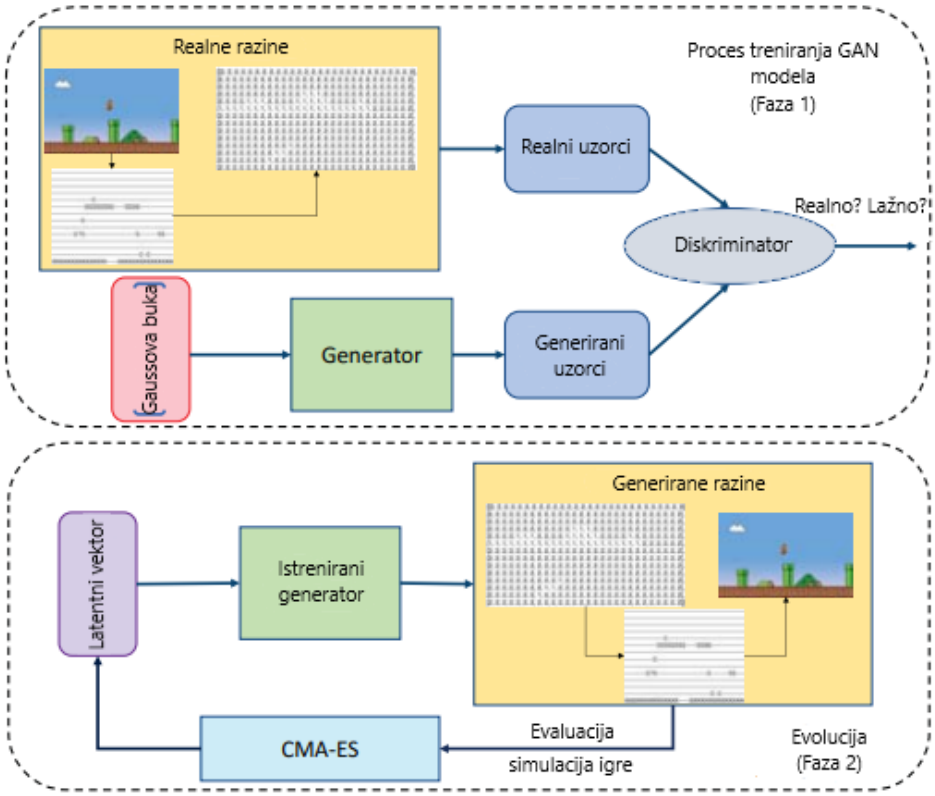
1. Postoji objekt lijevo od pozicije igrača na kojeg avatar igrača može skočiti, a nalazi se u prethodnoj sekciji
2. Postoji objekt desno od pozicije igrača na kojeg avatar igrača može skočiti
3. Postoji mogući put između ta dva objekta

Bitno je napomenuti kako nije potrebno da objekt sa lijeve i desne strane postoji već da je u mogućnosti postojati. Upravo ovo pravilo omogućava integraciju budućih generiranih sekcija kako bi se stvorile još kompleksnije i zanimljivije sekcije razine.

Varijablu stila (p_E) nije tako lako izmjeriti. Ovaj kriterij mjeri se odstupanjem pozicija postavljenih objekata u generiranoj sekciji od originalne sekcije iz koje je uzorak uzet te se uzima prosjek odstupanja za svaki pojedini tip objekta kao referenca za postavljanje ove varijable. Uzima se medijan udaljenosti svakog tipa objekta te se jasno može vidjeti kako između medijana svakog posebnog tipa objekta i varijable p_E postoji jaka korelacija.

8. Metoda generiranja korištenjem Generativnih suparničkih neuronskih mreža

Generativna suparnička neuronska mreža (engl. *Generative Adversarial Network - GAN*)(u daljnjem tekstu GAN) [21] je duboka neuronska mreža trenirana u nenadgledanom okruženju koja pokazuje velik uspjeh u reproduciranju aspekata slika iz seta za treniranje. Nadalje, kako bi se moglo objasniti generiranje razina pomoću GAN-a mora se dotaknuti i evolucijska strategija prilagodbe matrice kovarijanci[22] (engl. *Covariance Matrix Adaptation Evolutionary Strategy* (CMA-ES))(u daljnjem tekstu CMA-ES).



Slika 13: Dvije faze procesa treniranja GAN-a. Prva faza sastoji se od nenadgledanog učenja dok druga faza nadograđuje prvu traženjem latentnih vektora pomoću CMA-ES, preuzeto iz [21]

Prvo pojavljivanje GAN-a bilo je u 2014. godini kada Goodfellow *et al.*[23] opisuju proces treniranja ovakvih mreža kao igru između dva igrača koji igraju jedan protiv drugog. Ta dva igrača su zapravo generator *G* i diskriminator *D*. Generator pokušava prevariti diskrimnator na način da iz vektora nasumičnog šuma dekodira uzorke sekcije razine te ih pokušava složiti na način da izgledaju što realnije,a "posao" diskriminatora jest da prepoznaje koja sekcija je generirana, a koja je realna. Ako je generator uspio zavarati diskriminatora, tada je razina spremna za prijelaz u drugu fazu razvoja. U drugoj fazi razvoja [21] razine odbacuje se diskriminator te se u trenirani generator ubacuje vektor fiksne duljine iz latentnog prostora. Latentni vektor najčešće se uzima iz blok-uniformne ili izotropne Gaussove distribucije. Ubacivanjem latentnih

nasumičnih vektora u GAN dobivaju se primjeri razina koji će biti pogrešno klasificirani kao razine sa istim karakteristikama originalnog uzorka razine, ali čak i tada postoji dosta varijacija koje se pružaju ljudskom dizajneru razine na odabir kako bi zadovoljili određene kriterije.

8.1. Reprezentacija razine

Postoje varijacije u reprezentacijama Super mario razina između reprezentacije razine po korpusu za razine videoigara [24] (engl. *Video Game Level Corpus* - VGLC) i reprezentacije razine Mario okvira za umjetnu inteligenciju (engl. *Mario Artificial Intelligence Framework*). Obje reprezentacije su bazirane na blokovima koji "popločavaju" prazno polje razine, ali bitno je napomenuti kako se VGLC pristup više bazira na funkcionalnosti samih blokova nego na njihovoj estetici jer koriste samo jedan simbol za prikazivanje jedne funkcionalne skupine elemenata. Primjerice, neprijatelj je pokazan kao samo jedan simbol za neprijatelje te se tretira kao najčešći neprijatelj u razinama Super Maria, a to je *Goomba* neprijatelj iako taj neprijatelj ne mora nužno biti *Goomba* nego bilo koji drugi tip neprijatelja. Kako bi se razina enkodirala u skup podataka kojim možemo trenirati diskriminator mora se napraviti vektor u kojem se nalaze sve funckionalne skupine reprezentirane odgovarajućim podatkom tipa *integer*. Razine enkodirane u ovom formatu se zatim prosljeđuju Mario AI okviru kako bi dobile svoj grafički prikaz. Mario AI okvir ima mapirano puno više blokova nego što se može enkodirati VGLC metodom enkodiranja, ali koristi se samo mali dio tih preddefiniranih blokova iz razloga što je VGLC enkodiranje vrlo jednostavno i brzo, ali ne podržava više od 10 vrsta blokova.

Vrsta objekta	Simbol	Identitet	Vizualizacija
Čvrst/Tlo	X	0	
Lomljiv	S	1	
Prazan(prolazan)	-	2	
Popunjen upitni blok	?	3	
Prazan upitni blok	Q	4	
Neprijatelj	E	5	
Lijevi vrh cijevi	<	6	
Desni vrh cijevi	>	7	
Lijevi trup cijevi	[	8	
Desni trup cijevi	]	9	

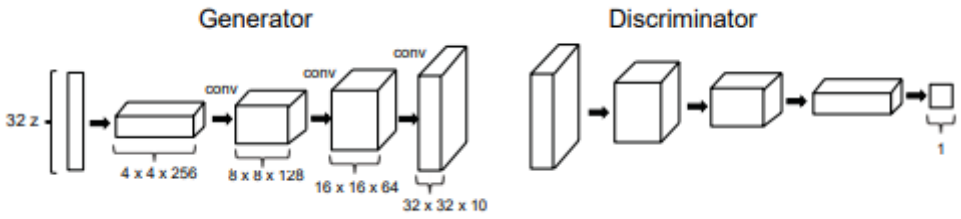
Slika 14: Simboli u VGLC formatu korišteni za odgovarajuće tipove blokova dok je identitet numerička vrijednost za MarioAI okvir kako bi se mogle predstaviti vizualizacije pokazane u desnom stupcu, preuzeto iz [21].

U datoteci, koja sadrži podatke izvedene kroz GAN, svaki redak odgovara jednom redu blokova unutar razine, a svi redovi su jednake duljine. Istom analogijom, postoji maksimalno 14 redova jer maksimalna visina razine iznosi 14 blokova. GAN je napravljen na način da uvijek vidi pravokutnu sliku (kao i igrač) veličine 14 blokova visine i 28 blokova širine te se

generiranje vizualizacije odvija pomicanjem slike u desno blok po blok kako se novi blokovi čitaju iz generirane datoteke. Za treniranje njihove verzije GAN-a [21] Volz, Schrum, Liu i dr., koristili su podatke samo iz prve razine prvog svijeta originalne Super Mario igre te su iz samo jedne razine dobili 173 različite slike veličine 14x28 blokova.

8.2. Treniranje GAN-a

Dubinski konvolucijski GAN (engl. *Deep Convolutional Generative Adversarial Network* - DCGAN) kojeg [21]Volz, Schrum, Liu i dr. koriste u svom primjeru je varijacija modela kreiranog od strane [25]Arjovsky, Chintala i Bottou i treniranog WGAN algoritmom (engl. *Wasserstein Generative Adversarial Network Algorithm*). [25]



Slika 15: Formati vektora koje koriste generator (lijevo) i diskriminator (desno) koi ujedno čine i standardnu strukrutu DCGAN-a, preuzeto iz [21]

Ako se pogleda arhitektura generatora i diskriminatora može se uočiti kako je to standardna DCGAN arhitektura koja u generatoru koristi frakcijski postepenu konvoluciju (engl. *Fractional-strided convolutions*), dok se u diskriminatoru za prevođenje vektorskih podataka u istinitosni tip podatka (istina-laž) koristi postepena konvolucija. Kod treniranja GAN-a svaki broj u datoteci iz koje se iščitavaju podatci pretvara se u *One-Hot* enkodirani vektor jer je za diskriminator potreban ulaz formata 32x32x10 kanala zbog 10 tipova mogućih oblika u VGLC enkodiranju. Na kraju kada je GAN istreniran podatcima, iz generatora se mora maknuti višak informacija kako bi izlaz generatora bio formata 10x28x14 iz razloga što su te veličine razine Super Mario igara te na tom formatu radi i MarioAI okvir za dekodiranje i vizualizaciju razine.

9. Praktični dio: implementacija algoritma za kreiranje razina pomoću GAN metode unutar PyTorch okvira

Svaki dio koda preuzet je iz [26]. Kod je otvorenog tipa te ga je, po autorovim naputcima, potrebno samo referencirati u radu.

9.1. Glavna datoteka

Isječak kôda 1: Varijabilni parametri prilikom pokretanja glavnog programa preuzeti iz [26]

```
1  parser = argparse.ArgumentParser()
2  parser.add_argument('--nz', type=int, default=32, help='veličina latentnog vektora Z
     ')
3  parser.add_argument('--ngf', type=int, default=64)
4  parser.add_argument('--ndf', type=int, default=64)
5  parser.add_argument('--batchSize', type=int, default=32, help='veličina ulazne
     serije')
6  parser.add_argument('--niter', type=int, default=5000, help='broj epoha koliko se
     puta trenira')
7  parser.add_argument('--lrD', type=float, default=0.00005, help='stopa učenja
     diskriminatora')
8  parser.add_argument('--lrG', type=float, default=0.00005, help='stopa učenja
     generatora')
9  parser.add_argument('--ngpu', type=int, default=1, help='broj jezgri grafičkog
     procesora koje se koriste')
10 parser.add_argument('--netG', default='', help="putanja do netG datoteke (za
     nastavak treniranja generatora")
11 parser.add_argument('--netD', default='', help="putanja do netD datoteke (za
     nastavak treniranja diskriminatora")
12 parser.add_argument('--Diters', type=int, default=5, help='broj ponavljanja
     diskriminatora za jedno ponavljanje generatora')
13 parser.add_argument('--n_extra_layers', type=int, default=0, help='broj dodatnih
     slojeva na diskriminatoru i generatoru')
14 parser.add_argument('--experiment', default=None, help='putanja mape u koju se
     spremaju modeli i primjeri')
15 parser.add_argument('--problem', type=int, default=0, help='Primjeri razine')
16 opt = parser.parse_args()
```

U isječku koda broj 1 nalaze se parametri prema kojima se generator i diskrimantor treniraju u glavnom programu. Parametar *--nz* sadrži informacije o veličini ulaznog vektora koji ulazi u generator kako bi se iz njega kao izlazni podatak kreirala matrica 32x32x10 znakova što je ujedno i ulazni parametar za diskriminator koji tada od takvog formata ulaza kreira izlaz od jednog bita koji znači istina ili laž, odnosno da li je razina dovoljno blizu realnoj ili se smatra lažnom, odnosno kreiranom od strane generatora. Parametar *--niter* označava broj iteracija kojima se generator i diskriminator treniraju, a parametri *--lrD* i *--lrG* označavaju stopu učenja diskriminatora i generatora. Sa parametrom *--ngpu* određuje se koliko jezgri grafičkog procesora će se

koristiti iz razloga što kompletan kod podržava višedretvenost (engl. *Multithreading*). Parametrom *--Diters* određuje se broj iteracija kojima se trenira diskriminator za svaku iteraciju treninga generatora, a sa parametrom *--nextralayers* određuje se koliko dodatnih slojeva će se koristiti u generatoru i diskriminatoru, odnosno koliko generiranih slika se može predstaviti kao jedna slika kod završne obrade generiranih slika. Svi navedeni parametri spremaju se u varijablu *opt*.

Isječak kôda 2: Priprema potrebnih varijabli za treniranje generatora i diskriminatora preuzeta iz [26]

```
map_size = 32
if opt.problem == 0:
    examplesJson = "example.json"
else:
    examplesJson = "sepEx/examplemario{}.json".format(opt.problem)
X = np.array(json.load(open(examplesJson)))
z_dims = 10
num_batches = X.shape[0] / opt.batchSize
X_onehot = np.eye(z_dims, dtype='uint8')[X]
X_onehot = np.rollaxis(X_onehot, 3, 1)
X_train = np.zeros((X.shape[0], z_dims, map_size, map_size))*2
X_train[:, 2, :, :] = 1.0
X_train[:X.shape[0], :, :X.shape[1], :X.shape[2]] = X_onehot
ngpu = int(opt.ngpu)
nz = int(opt.nz)
ngf = int(opt.ngf)
ndf = int(opt.ndf)
n_extra_layers = int(opt.n_extra_layers)
input = torch.FloatTensor(opt.batchSize, z_dims, map_size, map_size)
noise = torch.FloatTensor(opt.batchSize, nz, 1, 1)
fixed_noise = torch.FloatTensor(opt.batchSize, nz, 1, 1).normal_(0, 1)
one = torch.FloatTensor([1])
mone = one * -1
```

U isječku koda 2 nalazi se priprema dodatnih varijabli koje su potrebne za treniranje generatora i diskriminatora. Varijabla *map_size* definira veličinu slike koja će se generirati što je u ovom slučaju 32x32 bloka, odnosno znakova. Zatim u varijablu *examplesJson* učitavamo datoteku sa ekstenzijom .json u kojoj se nalaze dvodimenzionalna polja veličine 14 x 28 brojeva od 0 do 9, odnosno polja enkodirana *One_Hot* tipom enkodiranja gdje jedan broj predstavlja jedan element. Parametar *z_dims* sadrži informaciju o broju različitih blokova koji se mogu naći u kreiranju razine. Taj broj je u ovom slučaju 10 iz razloga što korisitimo *One_Hot* enkodiranje koje podržava samo 10 različitih tipova blokova jer toliko znamenaka imamo u decimalnom sustavu. U varijablu *X* učitavamo sve znakovne nizove iz varijable *examplesjson*. U varijabli *X_onehot* nalazi se matrica veličine *z_dims* x *z_dims* koja po dijagonali ima zapisane jedinice, a na svim ostalim mjestima nule. Varijabla *X_train* se popunjava nulama te se zatim koristi *slicing*, tehnika rezanja polja u programskom jeziku Python, kako bi se dvodimenzionalno polje *X_train* oblikovalo kao kvadrat. Parametri *ngpu*, *nz*, *ngf*, *ndf* i *n_extra_layers* prosljeđuju se u istoimene varijable, ali se tip podatka parsira u brojevni tip (engl. *integer*). Varijable *input*, *noise* i *fixed-noise* su dvodimenzionalna polja popunjena podatcima iz varijabli koje su prethodno opisane.

Isječak kôda 3: Funckija za dodjeljivanje tezinskih vrijednosti preuzeta iz [26]

```
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        m.weight.data.normal_(0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)

netG = dcgan.DCGAN_G(map_size, nz, z_dims, ngf, ngpu, n_extra_layers)
netG.apply(weights_init)

netD = dcgan.DCGAN_D(map_size, nz, z_dims, ndf, ngpu, n_extra_layers)
netD.apply(weights_init)
```

Isječak koda broj 3 prikazuje funkciju koja određuje težinsku vrijednost određenog podatka koji se unosi u generator i diskriminator. Ta funkcija se zatim primjenjuje na varijable *netG* i *netD* koje su objekti klase DCGAN_G i DCGAN_D koje su objašnjene u isječcima kodova 6 i 7.

Isječak kôda 4: Funkcija za spajanje generiranih slika preuzeta iz [26]

```
def combine_images(generated_images):
    num = generated_images.shape[0]
    width = int(math.sqrt(num))
    height = int(math.ceil(float(num)/width))
    shape = generated_images.shape[1:]
    image = np.zeros((height*shape[0], width*shape[1], shape[2]), dtype=
     generated_images.dtype)
    for index, img in enumerate(generated_images):
        i = int(index/width)
        j = index % width
        image[i*shape[0]:(i+1)*shape[0], j*shape[1]:(j+1)*shape[1]] = img
    return image
```

Funkcija za kombiniranje generiranih slika, opisana u isječku koda broj 4, kao ulaz uzima polje decimalnih brojeva koji variraju između -1 i 1. Funkcija se koristi kod ažuriranja generatorove mreže te se kombinirana slika ponovno koristi kao uzorak u sljedećoj iteraciji treninga.

Isječak kôda 5: Petlja u kojoj se provodi treniranje generatora i diskriminatora preuzeta iz [26]

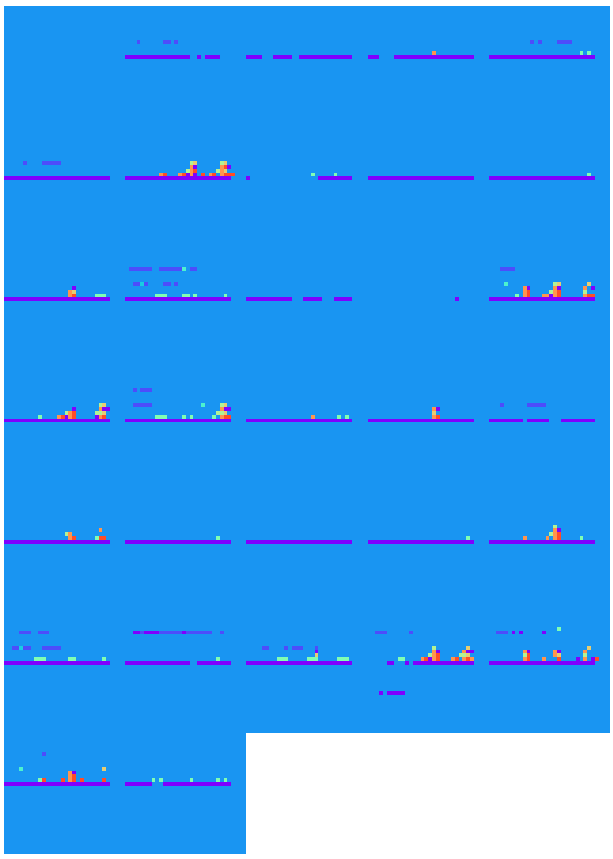
```
gen_iterations = 0
for epoch in range(opt.niter):
    X_train = X_train[torch.randperm(len(X_train))]
    i = 0
    while i < num_batches:
        for p in netD.parameters():
            p.requires_grad = True
        if gen_iterations < 25 or gen_iterations % 500 == 0:
            Diters = 100
        else:
            Diters = opt.Diters
        j = 0
        while j < Diters and i < num_batches:
            j += 1
            for p in netD.parameters():
                p.data.clamp_(opt.clamp_lower, opt.clamp_upper)
            data = X_train[i*opt.batchSize:(i+1)*opt.batchSize]
            i += 1
            real_cpu = torch.FloatTensor(data)
            netD.zero_grad()
            if opt.cuda:
                real_cpu = real_cpu.cuda()
            input.resize_as_(real_cpu).copy_(real_cpu)
            inputv = Variable(input)
            errD_real = netD(inputv)
            errD_real.backward(one)

            noise.resize_(opt.batchSize, nz, 1, 1).normal_(0, 1)
            noisev = Variable(noise, volatile=True)  # totally freeze netG
            fake = Variable(netG(noisev).data)
            inputv = fake
            errD_fake = netD(inputv)
            errD_fake.backward(mone)
            errD = errD_real - errD_fake
            optimizerD.step()


         for p in netD.parameters():
            p.requires_grad = False
        netG.zero_grad()
        noise.resize_(opt.batchSize, nz, 1, 1).normal_(0, 1)
        noisev = Variable(noise)
        fake = netG(noisev)
        errG = netD(fake)
        errG.backward(one)
        optimizerG.step()
        gen_iterations += 1
        print('[%d/%d][%d/%d][%d] Loss_D: %f Loss_G: %f Loss_D_real: %f Loss_D_fake
         %f' % (epoch, opt.niter, i, num_batches, gen_iterations, errD.data[0], errG
         .data[0], errD_real.data[0], errD_fake.data[0]))
        if gen_iterations % 50 == 0:   # was 500
            fake = netG(Variable(fixed_noise, volatile=True))
```

```
51              im = fake.data.cpu().numpy()
52              im = combine_images(tiles2image(np.argmax(im, axis=1)))
53              plt.imsave('{0}/mario_fake_samples_{1}.png'.format(opt.experiment,
                 gen_iterations), im)
54              torch.save(netG.state_dict(), '{0}/netG_epoch_{1}_{2}_{3}.pth'
55                          .format(opt.experiment, gen_iterations, opt.problem, opt.nz))
```

U isječku koda broj 5 za svaku epohu treniranja generatora i diskriminatora koja je definirana prethodno opisanim parametrom *--niter* u u varijablu *X_train* postavljaju se nasumični brojevi od 0 do broja podataka koji se nalaze u *X_train* varijabli. Zatim se svojstvo promjenjivosti omogućuje kod svih parametara diskriminatora pomoću petlje *while* te se broj iteracija kojima se diskriminator trenira određuje ovisno o varijabli *gen_iterations* definiranoj u prvom retku isječka koda broj 5. U varijablu *data* spremaju se realni pripremljeni podatkci iz varijable *X_train* te se zatim sa tim podatcima trenira diskriminator. Nakon toga se kreiraju lažni podatci u varijabli *inputv* sa kojima se zatim ponovno trenira diskriminator. Kod prelaska na treniranje generatora mora se onesposobiti svojstvo promjenjivosti kod svih parametara diskriminatora. Sljedeći korak je kreiranje nasumičnih, odnosno lažnih vrijednosti za generator te se svakih 50 iteracija jedna generirana slika razine spaja u prethodno spojene generirane slike razine pomoću funkcije *combine_images* koja je detaljno opisana u isječku koda broj 4.Nakon odrađenog cijelog postupka dobivamo sliku čiji se primjer može vidjeti na slici broj 16.



Slika 16: Primjer spojenih lažnih generiranih slika nakon 1250 iteracija programa

9.2. Duboka konvolucijska generativna suparnička mreža

9.2.1. Diskriminator

Isječak kôda 6: Klasa diskriminatora preuzeta iz [26]

```
class DCGAN_D(nn.Module):
    def __init__(self, isize, nz, nc, ndf, ngpu, n_extra_layers=0):
        super(DCGAN_D, self).__init__()
        self.ngpu = ngpu
        assert isize % 16 == 0, "isize has to be a multiple of 16"
        main = nn.Sequential()
        main.add_module('initial.conv.{0}-{1}'.format(nc, ndf),
                        nn.Conv2d(nc, ndf, 4, 2, 1, bias=False))
        main.add_module('initial.relu.{0}'.format(ndf),
                        nn.LeakyReLU(0.2, inplace=True))
        csize, cndf = isize / 2, ndf
        for t in range(n_extra_layers):
            main.add_module('extra-layers-{0}.{1}.conv'.format(t, cndf),
                            nn.Conv2d(cndf, cndf, 3, 1, 1, bias=False))
            main.add_module('extra-layers-{0}.{1}.batchnorm'.format(t, cndf),
                            nn.BatchNorm2d(cndf))
            main.add_module('extra-layers-{0}.{1}.relu'.format(t, cndf),
                            nn.LeakyReLU(0.2, inplace=True))
        while csize > 4:
            in_feat = cndf
            out_feat = cndf * 2
            main.add_module('pyramid.{0}-{1}.conv'.format(in_feat, out_feat),
                            nn.Conv2d(in_feat, out_feat, 4, 2, 1, bias=False))
            main.add_module('pyramid.{0}.batchnorm'.format(out_feat),
                            nn.BatchNorm2d(out_feat))
            main.add_module('pyramid.{0}.relu'.format(out_feat),
                            nn.LeakyReLU(0.2, inplace=True))
            cndf = cndf * 2
            csize = csize / 2
        main.add_module('final.{0}-{1}.conv'.format(cndf, 1),
                        nn.Conv2d(cndf, 1, 4, 1, 0, bias=False))
        self.main = main

    def forward(self, input):
        if isinstance(input.data, torch.cuda.FloatTensor) and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)
        output = output.mean(0)
        return output.view(1)
```

u isječku koda broj 6 nalazi se potpuna klasa sa konstruktorom i funkcijom *forward*. Funkcija *forward* prosljeđuje rad na više jezgri grafičkog procesora ukoliko je parametar *ngpu* veći od 1, odnosno raspoređuje treniranje diskriminatora na broj jezgri koji je odabran parametrom *ngpu*. Pilikom kreiranja objekta tipa DCGAN_D dodaju se moduli *Conv2d* i *LeakyReLU* u

program u kojem je objekt pozvan, a u slučaju da postoje dodatni slojevi tada se dodaje i modul *BatchNorm2d*.

Isječak kôda 7: Klasa generatora preuzeta iz [26]

```
class DCGAN_G(nn.Module):
    def __init__(self, isize, nz, nc, ngf, ngpu, n_extra_layers=0):
        super(DCGAN_G, self).__init__()
        self.ngpu = ngpu
        assert isize % 16 == 0, "isize has to be a multiple of 16"
        cngf, tisize = ngf//2, 4
        while tisize != isize:
            cngf = cngf * 2
            tisize = tisize * 2
        main = nn.Sequential()
        main.add_module('initial.{0}-{1}.convt'.format(nz, cngf),
                        nn.ConvTranspose2d(nz, cngf, 4, 1, 0, bias=False))
        main.add_module('initial.{0}.batchnorm'.format(cngf),
                        nn.BatchNorm2d(cngf))
        main.add_module('initial.{0}.relu'.format(cngf),
                        nn.ReLU(True))
        csize, cndf = 4, cngf
        while csize < isize//2:
            main.add_module('pyramid.{0}-{1}.convt'.format(cngf, cngf//2),
                            nn.ConvTranspose2d(cngf, cngf//2, 4, 2, 1, bias=False))
            main.add_module('pyramid.{0}.batchnorm'.format(cngf//2),
                            nn.BatchNorm2d(cngf//2))
            main.add_module('pyramid.{0}.relu'.format(cngf//2),
                            nn.ReLU(True))
            cngf = cngf // 2
            csize = csize * 2
        for t in range(n_extra_layers):
            main.add_module('extra-layers-{0}.{1}.conv'.format(t, cngf),
                            nn.Conv2d(cngf, cngf, 3, 1, 1, bias=False))
            main.add_module('extra-layers-{0}.{1}.batchnorm'.format(t, cngf),
                            nn.BatchNorm2d(cngf))
            main.add_module('extra-layers-{0}.{1}.relu'.format(t, cngf),
                            nn.ReLU(True))
        main.add_module('final.{0}-{1}.convt'.format(cngf, nc),
                        nn.ConvTranspose2d(cngf, nc, 4, 2, 1, bias=False))
        main.add_module('final.{0}.tanh'.format(nc),
                        nn.ReLU())
        self.main = main

    def forward(self, input):
        if isinstance(input.data, torch.cuda.FloatTensor) and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)
        return output
```

Isječak koda broj 7 vrlo je sličan isječku koda broj 6. Bitna razlika je da se u isječku 7 umjesto konvolucije *Conv2d* koristi funkcija *ConvTranspose2d* te se umjesto funkcije *LeakyReLU* koristi

funkcija *ReLU*. Funkcija *forward* raspodjeljuje rad treniranja generatora na više procesorskih jezgri kako bi cijeli proces treniranja bio brži zbog velikog broja iteracija. Prilikom kreiranja objekta tipa DCGAN_G u glavni program dodaju se moduli *Conv2d*, *ReLU* i *BatchNorm2d*. Ovisno o parametru dodatnih slojeva *n_extra_layers* ti moduli se pozivaju višestrui broj puta.

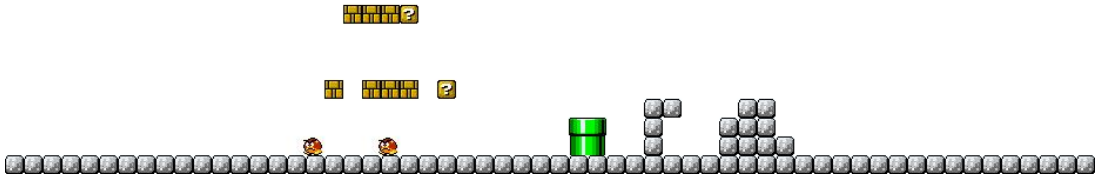
9.3. Generator razine

Isječak kôda 8: Kod za generiranje razine iz polja tipa .json

```
import torch
import torchvision.utils as vutils
from torch.autograd import Variable
import sys
import json
import numpy
import models.dcgan as dcgan
import matplotlib.pyplot as plt
import math
import random

if __name__ == '__main__':
 if len(sys.argv) ==1:
        modelToLoad = "netG_epoch_5000.pth"
 else:
        modelToLoad = sys.argv[1]
 if len(sys.argv) >=3:
        nz = int(sys.argv[2])
 else:
        nz = 32
 batchSize = 1
 imageSize = 32
 ngf = 64
 ngpu = 1
 n_extra_layers = 0
 z_dims = 10
 generator = dcgan.DCGAN_G(imageSize, nz, z_dims, ngf, ngpu, n_extra_layers)
 generator.load_state_dict(torch.load(modelToLoad, map_location=lambda storage, loc:
   storage))
  while 1:
  line = sys.stdin.readline()
  if len(line)==2 and int(line)==0:
    break
  lv = numpy.array(json.loads(line))
  latent_vector = torch.FloatTensor( lv ).view(batchSize, nz, 1, 1)
  levels = generator(Variable(latent_vector, volatile=True))
  level = levels.data.cpu().numpy()
  level = level[:,:,:14,:28]
  level = numpy.argmax( level, axis = 1)
  print(json.dumps(level[0].tolist()))
  sys.stdout.flush()
```

U isječku koda broj 8 nalazi se implementacija programskog koda za prethodno trenirani generator slika razina, odnosno generira se polje znakovnog niza tipa .json. Prvotno se u varijablu *model_to_load* učitava trenirani generator, u ovom slučaju, naziva "netG_epoch_5000.pth", a zatim se parametriziraju sve ostale varijable navedene u isječku koda broj 8. Nakon toga se kreira objekt klase DCGAN_G pod nazivom *generator* sa navedenim parametrima. U varijablu *lv* učitava se datoteka tipa .json u kojoj se nalazi polje znamenaka od 0 do 9 iz razloga što *One_Hot* tip enkodiranja podržava samo 10 različitih tipova objekata. U varijabli *level* dobivamo dvodimenzionalno polje formata 32 x 32 popunjeno znamenkama od 0 do 9 koje predstavljaju razne objekte u razini. Zatim se dio tog dvodimenzionalnog polja odbacuje kako bi se dobio format tipa 14 x 28 blokova jer je to standardni format Super Mario razine. Na slici 17 može se vidjeti gotova kreirana razina formata 14 x 28 blokova nakon zamjene znakovnog niza blokovima koji su ranije opisani na slici 14. Svaki od blokova ima svoj identitet enkodiran *One-Hot* metodom.



Slika 17: Primjer gotove kreirane Super Mario razine.

10. Zaključak

Iz priloženih informacija može se zaključiti kako postoje mnoge metode, koje se i dalje mogu razvijati i usavršavati, za izradu modela razine, ne samo za video-igru Super Mario već i za kreiranje razina ostalih video-igara. Kako komputacijska moć računala raste rapidnom brzinom tako se otvara sve više prostora za nove metode koje zahtjevaju bržu obradu podataka te obradu veće količine podataka u kratkom vremenu. To se jasno može vidjeti po evoluciji metoda opisanih u radu. Metoda generiranja iz uzoraka dizajna koristi se spajanjem uzoraka kao slagalice kako bi kreirala potpuno novu, dosad neviđenu razinu. Zatim metoda generiranja iz memorije koristi jednostavne LSTM blokove koji odlučuju da li je ulazni parametar uklapa sa ostalim elementima ili ne. Metoda kreiranja sadržaja iz videozapisa uzima sliku po sliku iz videozapisa te prepoznaje objekte i njihove relativne pozicije naspram drugih objekata postavljenih u istoj slici, a metoda generativnih suparničkih mreža koristi kompleksne metode odlučivanja koji element staviti gdje kako bi razina izgledala što sličnije originalnoj razini Super Maria. Svaka od ovih metoda odnosi razvitak algoritama za kreiranje virtualnih razina, bez ljudske interakcije, jedan korak dalje. Stoga se jasno može zaključiti kako će se u budućnosti pronaći novi način i novi algoritam da ovaj proces kreiranja razina učini još realnijim i bržim. Samo je pitanje vremena kada će tehnologija dovoljno uznapredovati da to omogući.

Popis literature

[1] N. Esposito, „A short and simple definition of what a videogame is," *Digital Games Research Conference 2005*, 2005., str. 3.

[2] T. Thompson, „"What is a Super Mario level anyway?"An Argument For Non-Formalist LevelGeneration in Super Mario Bros," *Foundation of Digital Games Conference 2016*, 2016., str. 5.

[3] R. Hunicke, M. LeBlanc i R. Zubek, „MDA: A formal approach to game design and game research," *Proceedings of the AAAI Workshop on Challenges in Game AI*, San Jose, CA, sv. 4, 2004.

[4] M. Mohri, A. Rostamizadeh i A. Talwalkar, *Foundations of machine learning*. MIT press, 2018., str. 1.

[5] A. Chakraborty i S. Sharma, „Machine Learning in Artificial Intelligence," *International Journal of Advanced Research in Engineering and Technology*, sv. 11, br. 6, 2020.

[6] M. Vuković, *Složenost algoritama, nastavni materijali*, Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu, str. 19–53, 89–98.

[7] (5. 2019.). „Game styles - super mario maker 2 wiki guide." preuzeto 20.8.2021., adresa: https://www.ign.com/wikis/super-mario-maker-2/Game_Styles.

[8] S. Dahlskog i J. Togelius, „Patterns and Procedural Content Generation: Revisiting Mario in World 1 Level 1," 5. 2012. DOI: 10.1145/2427116.2427117.

[9] S. Dahlskog i J. Togelius, „Patterns as Objectives for Level Generation," 5. 2013., str. 1–7.

[10] D. Soni. (3. 2018.). „Introduction to Markov Chains." preuzeto 22.8.2021., adresa: https://towardsdatascience.com/introduction-to-markov-chains-50da3645a50d.

[11] S. Sharma. (8. 2018.). „Monte Carlo Tree Search." preuzeto 20.8.2021., adresa: https://towardsdatascience.com/monte-carlo-tree-search-158a917a8baa.

[12] Techopedia. (8. 2018.). „What is Long short-term MEMORY (LSTM)? - definition from Techopedia." preuzeto 17.08.2021., adresa: https://www.techopedia.com/definition/33215/long-short-term-memory-lstm.

[13] A. Summerville i M. Mateas, *Super Mario as a String: Platformer Level Generation Via LSTMs*, 2016. arXiv: 1603.00930 [cs.NE].

[14] S. Hochreiter i J. Schmidhuber, „Long Short-term Memory," *Neural computation*, sv. 9, str. 1735–80, 12. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[15] S. Snodgrass i S. Ontanon, „Generating maps using Markov chains," str. 25–28, 1. 2013.

[16] M. Guzdial i M. Riedl, *Toward Game Level Generation from Gameplay Videos*, 2016. arXiv: 1602.07721 [cs.AI].

[17] D. T. Pham, S. S. Dimov i C. D. Nguyen, „Selection of K in K-means clustering," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, sv. 219, br. 1, str. 103–119, 2005. DOI: 10.1243/095440605X8298. eprint: https://doi.org/10.1243/095440605X8298.

[18] E. Kalogerakis, S. Chaudhuri, D. Koller i V. Koltun, „A Probabilistic Model for Component-Based Shape Synthesis," *ACM Trans. Graph.*, sv. 31, br. 4, str. 2–4, 7. 2012., ISSN: 0730-0301. DOI: 10.1145/2185520.2185551.

[19] I. Mirošević, „Algoritam k-sredina," *KoG : znanstveno-stručni časopis Hrvatskog društva za konstruktivnu geometriju i kompjutorsku grafiku*, sv. 20, br. 20, str. 91–98, 2017., ISSN: 1331-1611.

[20] Y. Wang, H. Wu i Z. Sheng, „A prioritized test generation method for pair-wise testing," *vol*, sv. 11, str. 136–143, 2013.

[21] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith i S. Risi, *Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network*, 2018. arXiv: 1805.00728 [cs.AI].

[22] N. Hansen, S. Müller i P. Koumoutsakos, „Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)," *Evolutionary computation*, sv. 11, str. 1–18, 2. 2003. DOI: 10.1162/106365603321828970.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville i Y. Bengio, „Generative Adversarial Nets," *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence i K. Q. Weinberger, ur., sv. 27, Curran Associates, Inc., 2014.

[24] A. J. Summerville, S. Snodgrass, M. Mateas i S. Ontañón, *The VGLC: The Video Game Level Corpus*, 2016. arXiv: 1606.07487 [cs.HC].

[25] M. Arjovsky, S. Chintala i L. Bottou, „Wasserstein Generative Adversarial Networks," *Proceedings of the 34th International Conference on Machine Learning*, D. Precup i Y. W. Teh, ur., serija Proceedings of Machine Learning Research, sv. 70, PMLR, 8. 2017., str. 214–223.

[26] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. M. Smith i S. Risi, „Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2018)*, Kyoto, Japan: ACM, 7. 2018. DOI: 10.1145/3205455.3205517.

Popis slika

Popis isječaka koda