Inovativni pristup žanru 2D igara gađanja kroz integraciju umjetne inteligencije

Bečeić, Felix

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 440

INOVATIVNI PRISTUP ŽANRU 2D IGARA GAĐANJA KROZ INTEGRACIJU UMJETNE INTELIGENCIJE

Felix Bečeić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 440

INOVATIVNI PRISTUP ŽANRU 2D IGARA GAĐANJA KROZ INTEGRACIJU UMJETNE INTELIGENCIJE

Felix Bečeić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 440

Pristupnik:	**Felix Bečeić (0036526835)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Mirko Sužnjević

Zadatak: **Inovativni pristup žanru 2D igara gađanja kroz integraciju umjetne inteligencije**

Opis zadatka:

Umjetna inteligencija (UI) postala je neizostavan element u videoigrama. Nedavni značajni napretci u području UI-a omogućuju ne samo standardno upravljanje računalno kontroliranim likovima, već i napredne mehanike i dinamično prilagođavanje protivnika, pružajući igračima izazovnije i personalizirano iskustvo. Kroz algoritamsko učenje, UI omogućuje adaptivno prilagođavanje igre prema stilu igrača, poboljšavajući korisničko iskustvo. Algoritmi UI-a za generiranje medijskih sadržaja otvaraju potencijal direktnog modificiranja grafičkog izgleda igara. Vaš zadatak je razviti modularnu 2D igru gađanja kombinirajući klasičnu mehaniku igre "Space Invaders" s algoritmima umjetne inteligencije za generiranje slike. Koncept se temelji na dinamičkom generiranju sučelja i tekstura igre prema korisničkim upitima definiranim pri pokretanju igre. Kroz slobodan tekst, korisnik može zadati izgled određenih elemenata igre, poput sučelja, izgleda igrača i neprijatelja. Pomoću AI usluga, igra bi trebala generirati grafičke elemente i integrirati ih u prikaz. Ovaj pristup omogućuje visoku razinu personalizacije i imerzivnosti za svakog igrača, istovremeno demonstrirajući potencijal UI-a u transformaciji interaktivnog iskustva videoigara.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

Uvod

Krajem 2022. godine se pojavom naprednije i komercijalne umjetne inteligencije (engl. *Artificial Intelligence – AI*) znatno proširio vidokrug mogućnosti koje se mogu postići integracijom iste u različite aplikacije i programe, ali i u videoigre. Specifično, integracijom umjetne inteligencije u videoigre moguće je implementirati pametnije i kontekstualno svjesnije računalno kontrolirane likove u svijetu (engl. *Non-Playable Characters – NPCs*) s kojima će dijalog uvijek biti drugačiji, ali i obogatiti videoigru dinamičnijim elementima poput proizvoljnih tekstura, pružajući unikatno i posebno iskustvo prilikom svakog igranja. Cilj ovog rada je istražiti inovativne pristupe u razvoju 2D igara gađanja kroz primjenu algoritama umjetne inteligencije. Inspiriran klasičnom igrom Space Invaders, videoigra Prompt Invaders razvijena u sklopu ovog rada kombinira tradicionalne mehanike igranja s naprednim mogućnostima AI-ja za generiranje i prilagođavanje vizualnih elemenata igre u stvarnom vremenu. Videoigra namjerava korisničkim unosom teksta oblikovati izgled raznih elemenata igre, poput neprijatelja, metaka, pozadine i izgleda glavnog izbornika, dok rad namjerava razmotriti pretvorbu teksta u sliku te raspon mogućnosti umjetne inteligencije za unaprjeđivanje ne samo tehničkog aspekta razvoja igara, već i ukupnog iskustva igrača – pružajući novi smjer u dizajnu i razvoju videoigara.

1. Umjetna inteligencija

Umjetna inteligencija je grana računalne znanosti koja se bavi razvojem sustava i algoritama sposobnih za obavljanje zadataka koje inače zahtijevaju ljudsku inteligenciju. To uključuje sposobnosti kao što su učenje, zaključivanje, percepcija, razumijevanje prirodnog jezika, prepoznavanje uzoraka i donošenje odluka [1].

Umjetna inteligencija koristi različite tehnike, poput:

- strojnog učenja (engl. *machine learning - ML*) – za učenje iz podataka i unaprjeđivanje performansi bez eksplicitnog programiranja,
- dubokog učenja (engl. *deep learning – DL*) – podskup strojnog učenja za analizu složenih podataka i prepoznavanje uzoraka višeslojnim neuronskim mrežama,
- prirodnog jezika (engl. *natural language processing – NLP*) – tehnike koje omogućuju računalima interpretaciju i generiranje ljudskog jezika i
- računalnog vida – metode koje omogućuju računalima da analiziraju i interpretiraju vizualne podatke iz okoline.

Umjetna inteligencija u današnjem svijetu obuhvaća velik opseg pristupa i metoda, ali kao najbitniji pristup moglo bi se izdvojiti strojno učenje. Koncept strojnog učenja, kao dijela umjetne inteligencije, predstavlja način upravljanja radom računala takav da ne iziskuje ikakvu potrebu za programiranjem [6].

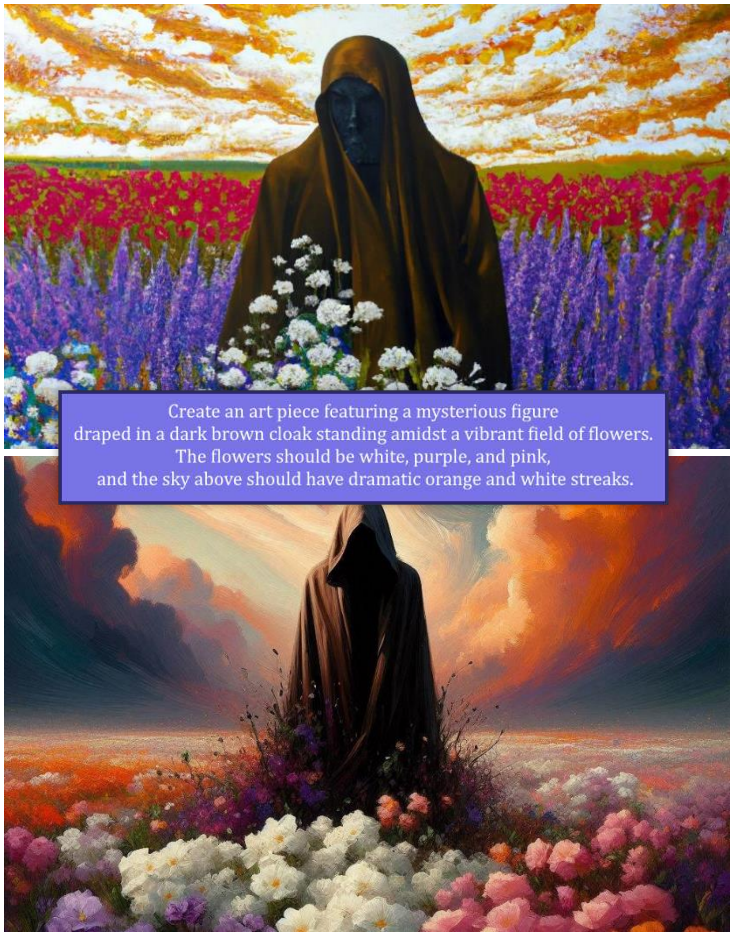
Pristupe strojnom učenju možemo razdijeliti prema tipu podataka i radom s istima na ulazu i izlazu, a tri pristupa koja danas dominiraju su [5]:

- nadzirano učenje (engl. *supervised learning*),
- nenadzirano učenje (engl. *unsupervised learning*) i
- podržano učenje (engl. *reinforcement learning*).

1.1. Evolucija modela umjetne inteligencije

Umjetna inteligencija se u svojim različitim oblicima kroz vrijeme integrirala u puno zanimanja – poput medicine, automobilske industrije, marketinga, financija,s IT industrije, itd. U razdoblju od 2022. – 2024. godine skoro pa eksponencijalno rastu joj mogućnosti, a time i popularnost pojavom ChatGPT-ja [19]. ChatGPT (od kompanije OpenAI) predstavlja

chatbota koji koristi veliku količinu podataka s interneta te ih različitim algoritmima analizira i logički evaluira kako bi dao inteligentan i čovjekolik odgovor na navedeni upit. ChatGPT primarno prima tekst kao ulaz (engl. *input*), a kao izlaz (engl. *output*) vraća tekst koji predstavlja odgovor na navedeni upit. Isti također ima mogućnost pretvorbe iz teksta u sliku (engl. *text-to-image conversion*), za koju se koristi *text-to-image* model DALL-E (isto od OpenAI-a). DALL-E je pušten u javnost početkom 2021. godine, dok je aktualni model DALL-E 3 (sa znatno većim mogućnostima i boljim rezultatima) pušten u javnost već u listopadu 2023. godine. DALL-E je svojim razvitkom evoluirao *text-to-image* scenu rezultatima koji su mogli izvrsno ispasti, pogotovo ako je zadani upit (engl. *prompt*) detaljan i jasno objašnjen (Sl. 1.1).



Slika 1.1: Primjer generirane slike s DALL-E 2 modelom (gore) i DALL-E 3 modelom (dolje)

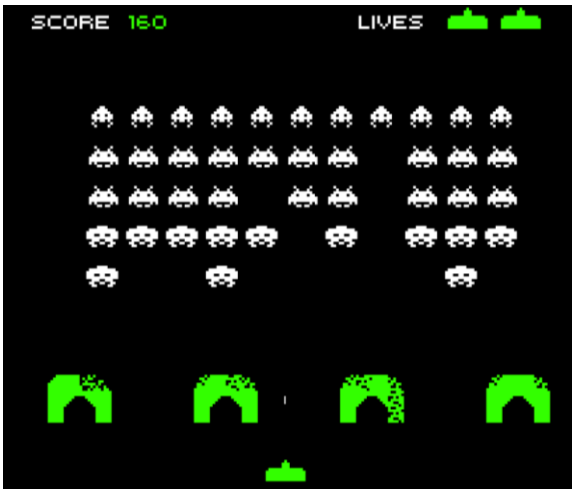
Pojavom *chatbota* Microsoft Copilota DALL-E je integriran u njihov ekosustav kao model koji izvršava *text-to-image* generaciju. Osim DALL-E-ja, MidJourney se ističe kao jedan od najboljih, ako ne i najbolji *text-to-image* model, međutim nije besplatan.

Model koji se može u potpunosti preuzeti na računalo i pruža *text-to-image* generaciju zove se Stable Diffusion, a integriran je u alatima poput Stable Diffusion WebUI i Stable Diffusion WebUI Forge. WebUI Forge zahtijeva visoke računalne performanse, a nudi i razne dodatke (engl. *plugins*) kako bi se precizirala i podesila (engl. *fine*-tune) generacija slike.

1.2. Primjene umjetne inteligencije u videoigrama

Utjecaj umjetne inteligencije na videoigre postaje sve značajniji posljednjih godina. Neki izdavači videoigri – poput *Rockstar Gamesa* – najavili su da bi u njihovoj nadolazećoj videoigri *Grand Theft Auto VI* umjetna inteligencija mogla imati značajnu ulogu, pružajući inteligentnije i interaktivnije NPC-jeve koji će neskriptirano reagirati na određene interakcije [2].

Međutim, umjetna se inteligencija (u svojoj primitivnijoj definiciji nego danas) koristi u videoigrama već desetljećima. Neki klasici i franšize u svijetu videoigara – kao što su Pac-Man, Half-Life, Halo, Metal Gear Solid i Grand Theft Auto – integriraju umjetnu inteligenciju u različitim aspektima igranja. Najčešće se tu ističu ponašanja neprijatelja koja se određuju na temelju različitih ulaznih parametara koji proizlaze iz kontekstualne svjesnosti samog neprijatelja ili NPC-ja. Naravno, tu se ističe i vjerojatno najutjecajnija videoigra ikad – Space Invaders – u koju je integriran AI smatran relativno jednostavnim u usporedbi s današnjim standardima, ali je i dalje bio odskočna daska (engl. *stepping stone*) za sve što slijedi (Sl. 1.2).



Slika 1.2: Space Invaders videoigra (iz https://tropedia.fandom.com/wiki/Space_Invaders)

U ovom radu cilj je uzeti ovaj *stepping stone* i unaprijediti ga davanjem videoigri točno onakvo ruho kakvo igrač zahtijeva, stvarajući moderniji pristup videoigri inovativnom integracijom umjetne inteligencije.

2. Pregled literature

Iza svake nove *text-to-image* generacije stoji mnoštvo algoritama koji nastoje razriješiti određene probleme i izazove prilikom takve pretvorbe. Ti algoritmi moraju biti „inteligentniji“ od standardnih, odnosno moraju imati sposobnosti prepoznavanja napisanih slova, povezivanja tih slova u riječi koje nose značenje i težinu, razumijevanja grupi riječi i njihovog konteksta, itd. Svi ti algoritmi pod okriljem su strojnog učenja, a većina njih igra ulogu u pretvorbi teksta u sliku barem djelomično.

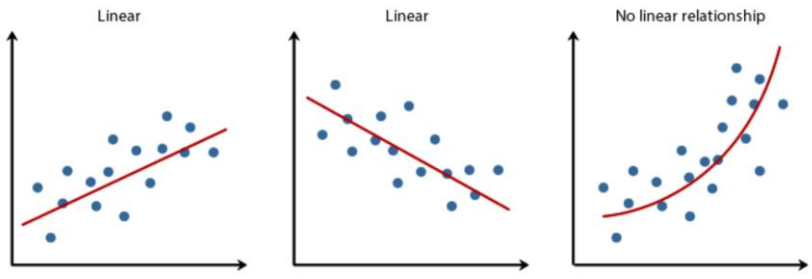
2.1. Nadzirano učenje

Nadzirano učenje (engl. *supervised learning*) je tip učenja koji podrazumijevaju poznavanje kategoriziranih izlaznih podataka pomoću kojih model pod nadzorom stvara odnose između obilježja ulaznih podataka i pripadajućim obilježenim kategorijama [5]. Podaci u ovakvom modelu dijele se na:

- skup za učenje (engl. *training set*) – koji predstavlja podatke za učenje stroja i
- skup za testiranje (engl. *testing set*) – koji predstavlja ostale drugačije podatke od skupa za učenje preko kojih se provjerava učinkovitost učenja u određenom trenutku.

Najvažnije primjene ovakvog tipa učenja vidljivi su u modelima regresije i klasifikacije.

Regresija (engl. *regression*) je nadzirani model koji se temelji na aproksimiranju funkcije koje služi kao alat za mapiranje (engl. *mapper*) ulaznog skupa na izlazni. Određivanjem ovakve funkcije postiže se mogućnost sustava da predvidi vrijednosti izlaza za nove ulazne podatke, a funkcija može biti linearna ili polinomna (Sl 2.1).



Slika 2.1: Usporedba linearne i polinomne regresije (iz [5])

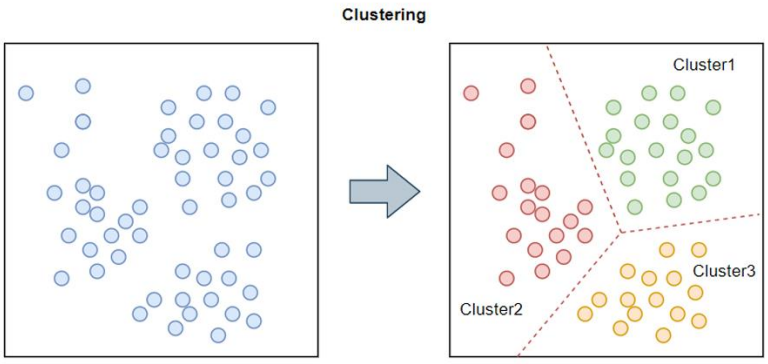
Klasifikacija (engl. *classification*) razmatra vrijednosti i kategorizira ih prema njihovoj pripadnosti, smještajući ih u pripadajuće kategorije koje predstavljaju izlaz modela, odnosno oznaku dodijeljenu podatku. Neki od modela koji koriste klasifikaciju su drva odluke (engl. *decision trees*), naivni Bayes, neuronske mreže, itd.

U *text-to-image* generaciji, modeli poput Stable Diffusiona koriste neke tehnike nadziranog učenja prilikom procesiranja slike, poput:

- pred treniranja modela – koristeći nadzirane tehnike za bolje razumijevanje obrazaca u podacima,
- poboljšanja tekstualnih opisa – modelima za obradu prirodnog jezika (engl. *NLP - Natural Language Processing*) koji poboljšavaju ulazne tekstualne opise i
- kontrole izlaznih varijabli – korištenjem tzv. klasifikatora koji bi osigurali da generirane slike zadovoljavaju određene kriterije.

2.2. Nenadzirano učenje

Nenadzirano učenje (engl. *unsupervised learning*) razlikuje se od nadziranog po tome što na temelju zapažanja počinje raspoznavati podatke i njihovu povezanost, a ne na temelju odgovora iz kojih može nešto naučiti [5]. Stroj koji se zasniva na ovakvom učenju najviše se oslanja na grupiranje (engl. *clustering*), odnosno svrstavanju podataka u određene grupe prema povezanosti i sličnosti (Sl. 2.2).



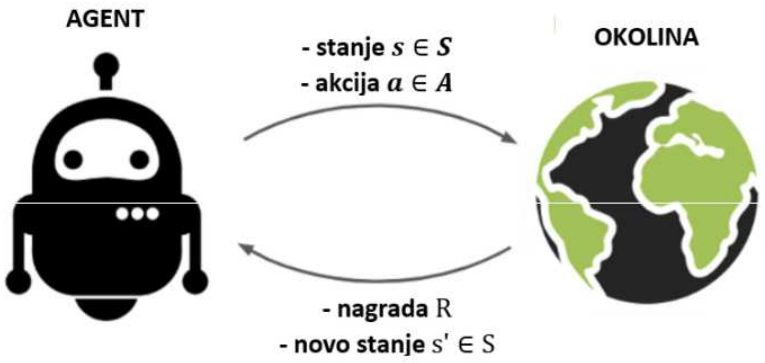
Slika 2.2: Primjer grupiranja (iz [6])

U *text-to-image* generaciji, modeli poput Stable Diffusiona koriste neke tehnike nenadziranog učenja prilikom procesiranja slike, kao što su:

- grupiranje – za poboljšanu analizu i evaluaciju kvalitete generiranih rezultata,
- smanjenje dimenzionalnosti – kao pomoć pri smanjenju kompleksnosti podataka (što može ubrzati proces treniranja i poboljšati performanse modela) i
- otkrivanje anomalija – za prepoznavanje i ispravljanje slučajeva gdje model proizvodi neobične rezultate te filtraciju generiranih slika.

2.3. Podržano učenje

Podržano učenje (engl. *reinforcement learning*) je vrsta strojnog učenja gdje agent uči kako djelovati u okolišu kako bi maksimizirao nagradu. Umjesto da bude treniran na unaprijed označenim podacima (kao u nadziranom učenju), agent otkriva optimalne akcije kroz pokušaje i pogreške, koristeći povratne informacije iz okoliša. Proces uključuje interakciju između agenta – koji provodi akcije – i okoliša – koji vraća povratne informacije u obliku nagrada ili kazni.



Slika 2.3: Ilustracija podržanog učenja (iz [7])

Za razliku od nadziranog učenja u kojem je poznat izlaz za svaki ulazni podatak, kod podržanog učenja odluke su međuovisne i donose se sekvencijalno. Za razliku od nenadziranog učenja u kojem je cilj grupirati podatke, kod podržanog učenja uspješnim i neuspješnim interakcijama agent uči kako postupiti u određenoj situaciji [7] .

Dobivanjem ili nedobivanjem nagrada za pojedinu interakciju agent razlikuje ispravne od neispravnih akcija i konstantno uči na temelju takvih povratnih informacija. Primjera radi, zamislimo robota u skladištu koji pokušava pronaći optimalan put do određenih točaka bez prethodnog znanja o rasporedu skladišta. Nadzirano učenje koristilo bi algoritme poput Naivnog Bayesa za prikupljanje podataka o putanjama u raznim okruženjima, trenirao se na

temelju povijesnih podataka, učio na označenim primjerima i primjenjivao unaprijed definirane strategije. Nenadzirano učenje bi grupiranjem elemenata u okruženju robota učilo o prostoru u kojem se nalazi, stvarajući razumijevanje prostora temeljeno na uzorcima koje prepoznaje. Podržano učenje bi – pak – definiralo neke nagrade (npr. dolazak do točke interesa, sudaranje s preprekom, itd.), istražilo prostor u kojem se nalazi i optimizirao tehnike kroz iteracije kako bi na temelju povratnih informacija naučio optimalne akcije kroz interakciju s okruženjem.

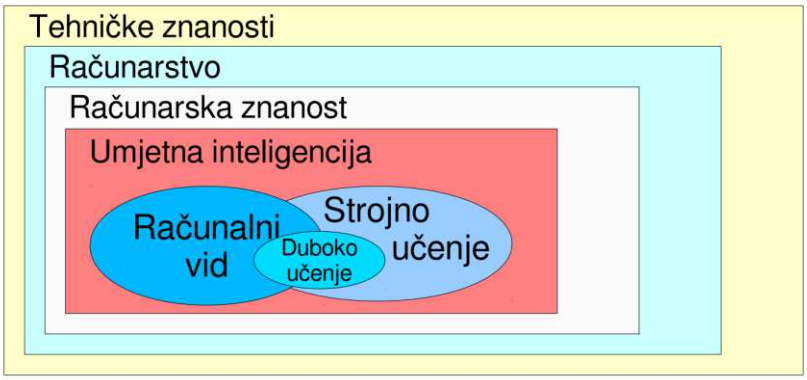
2.4. Duboko učenje

Duboko učenje (engl. *deep learning*) predstavlja podskup strojnog učenja koji se primarno oslanja na neuronske mreže – strukture „inspirirane“ ljudskim mozgom sastavljene od dubokih slojeva neurona – za analizu i učenje iz složenijih podataka poput prirodnog jezika. Što je veći broj slojeva (što je „dublja“ neuronska mreža), to su zadaci koje bi mreža mogla obavljati složeniji [5]. Duboko učenje koristi se za razne primjene, poput analize medicinskih slika, prevođenja jezika, autonomna vozila, generiranje glazbe, izradu modela za igre, itd.

Prilikom *text-to-image* generacije, neke od ključnih tehnika dubokog učenja koje igraju veliku ulogu u procesu generacije slike su:

- umjetne neuronske mreže,
- konvolucijske neuronske mreže,
- generativne kontradiktorne mreže i
- varijacijski autokoderi.

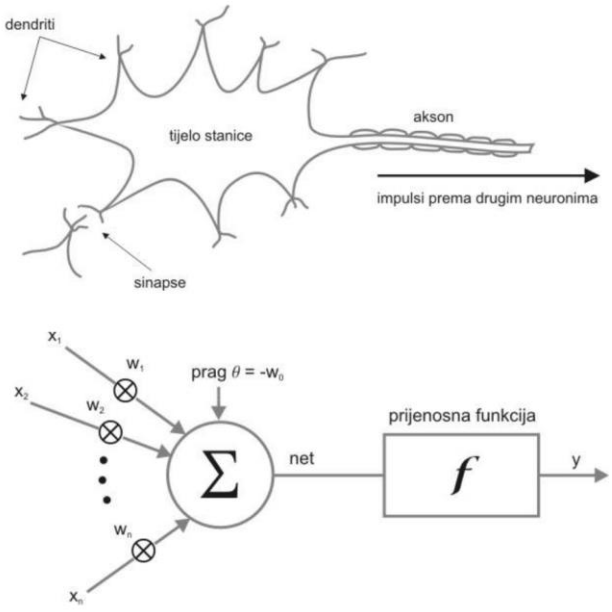
Prilikom *text-to-image* i *image-to-image* generacije, grana umjetne inteligencije koja se bavi automatskom analizom i razumijevanjem vizualnih informacija zove se računalni vid. Ova grana omogućuje sustavima da bolje razumiju i interpretiraju vizualne podatke. Računalni vid preklapa se sa strojnim učenjem i dubokim učenjem samo djelomično, dok je duboko učenje svojim metodama u potpunosti dio domene strojnog učenja (Sl. 2.4).



Slika 2.4: Dijagram pripadnosti domena tehničkih znanosti (iz [3])

2.4.1. Neuronske mreže

Neuronske mreže (engl. *neural networks*) predstavljaju umjetnu repliku ljudskog mozga koja simulira proces učenja, odnosno skup međusobno povezanih jednostavnih procesnih elemenata – neurona. Neuronske mreže su karakterizirane obradom podataka distribuiranim paralelnim radom čvorova te sposobnošću učenja iz iskustva. Prilagodbom okolini imaju sposobnost elegantno riješiti probleme klasifikacije i predviđanja [4]. Mreže se uče iterativnim postupkom predočavanja ulaznih primjera i prilagodbom težinskih faktora (Sl. 2.5).



Slika 2.5: Usporedba građe pravog neurona (gore) i strukture umjetnog (dolje) (iz [4])

Ulazni signali, njih ukupno n, označeni su sa $x_1, x_2, \dots, x_n$. Težine su označene sa $\omega_1, \omega_2, \dots, \omega_n$. Ulazni signali su općenito realni brojevi u intervalu [-1,1], [0,1] ili jednostavno elementi iz skupa $\{0,1\}$ kada se radi o Booleovim ulazima [4]. Težinska suma net dana je s:

$$net = \omega_0 x_0 + \omega_1 x_1 + \omega_2 x_2 + \ \dots \ + \omega_n x_n$$

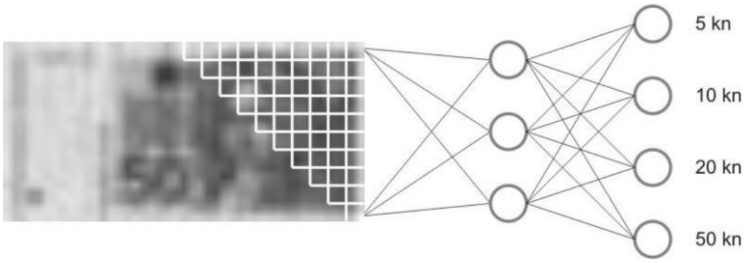
Izlaz y, odnosno rezultat prijenosne funkcije $f(net)$ računa se na sljedeći način:

$$y = f\left(\sum_{i=0}^{n} \omega_i x_i\right)$$

Osnovna arhitektura neuronskih mreža uključuje:

- ulazni sloj – prvi sloj mreže koji prima ulazne podatke,
- skrivene slojeve – jedan ili više slojeva između ulaznog i izlaznog sloja gdje se odvija većina računskih operacija i
- izlazni sloj – završni sloj koji daje konačne predikcije ili klasifikacije.

Na primjer, ako uzmemo novčanicu i pokušamo je klasificirati pomoću neuronskih mreža, na ulaznom sloju nalazit će se sken novčanice, a na izlaznom će biti predikcije o kojoj se novčanici radi (Sl. 2.6).



Slika 2.6: Primjer korištenja umjetnih neuronskih mreža za klasifikaciju novčanica (iz [4])

Neuronske mreže mogu biti plitke ili duboke, ovisno o broju slojeva. Duboke neuronske mreže, poznate kao duboko učenje, imaju veći broj skrivenih slojeva što im omogućava da uče složenije funkcije.

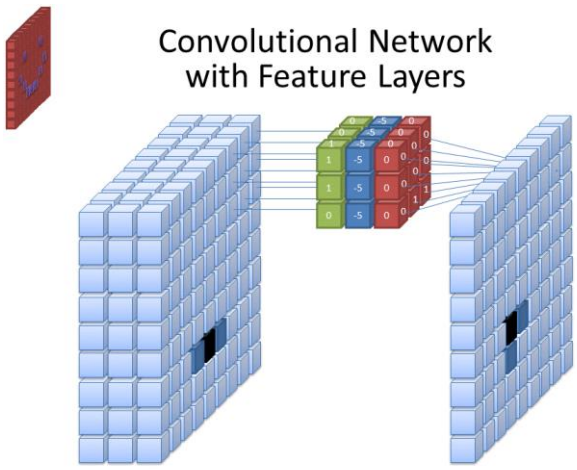
Treniranje neuronskih mreža uključuje nekoliko tehnika. Širenjem unaprijed (engl. *forward propagation*) ulazni podaci prolaze kroz mrežu, a svaka veza množi podatke svojim

težinama, dok neuroni primjenjuju aktivacijske funkcije na svoje ulaze. Pogreška između predikcije mreže i stvarne vrijednosti izračunava se korištenjem funkcije gubitka. Algoritam sa širenjem pogreške unazad (engl. *backpropagation*) propagira pogrešku unatrag kroz mrežu, a težine se prilagođavaju kako bi se smanjila pogreška. Što se tiče optimizacije, algoritmi poput gradijenta spusta koriste se za optimizaciju težina i smanjenje funkcije gubitka [5].

2.4.2. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (engl. *convolutional neural networks - CNN*) su specijalizirane vrste neuronskih mreža namijenjene pretprocesiranju nestrukturiranih podataka poput slika, teksta, zvuka i govora. Inspirirane su radom Hubela i Wiesela, koji su 1981. godine dobili Nobelovu nagradu za istraživanje vizualnog korteksa mačaka. Otkrili su da se neuroni u korteksu aktiviraju kao odgovor na određene vizualne podražaje, što je postalo temelj za razvoj CNN-ova [11].

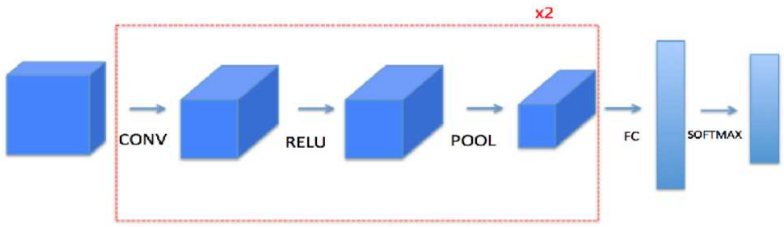
Ime "konvolucijske neuronske mreže" dolazi od matematičke operacije konvolucije – specijalnog linearnog operatora koji se primjenjuje na barem jednom sloju mreže. Konvolucija modificira ulazni signal kako bi se izvukla bitna svojstva – poput rubova na slikama – što je ključno za prepoznavanje objekata i drugih vizualnih značajki. Funkcija konvolucijskih slojeva je pretvoriti sliku u numeričke vrijednosti koje neuronska mreža može interpretirati i zatim iz njih izvući relevantne uzorke. Posao filtera u konvolucijskoj mreži je stvoriti dvodimenzionalni niz vrijednosti koje se mogu prenijeti u kasnije slojeve neuronske mreže, one koji će naučiti uzorke na slici (Sl. 2.7).



Slika 2.7: Prikaz filtra i kanala konvolucijskih neuronskih mreža (iz [10])

Osnovne komponente konvolucijske neuronske mreže su:

- ulazni sloj – ulazna slika,
- konvolucijski sloj – konvoluira ulaznu sliku s određenim brojem filtara za stvaranje mape značajki,
- aktivacijska funkcija – uglavnom se koristi ReLU funkcija koja dodaje nelinearnost i pomaže u sprječavanju zasićenja gradijenta,
- sloj sažimanja (engl. *pooling*) – smanjuje dimenzionalnost aktivacijskih mapa i
- potpuno povezani slojevi neurona.



Slika 2.8: Prikaz arhitekture konvolucijske neuronske mreže (iz [11])

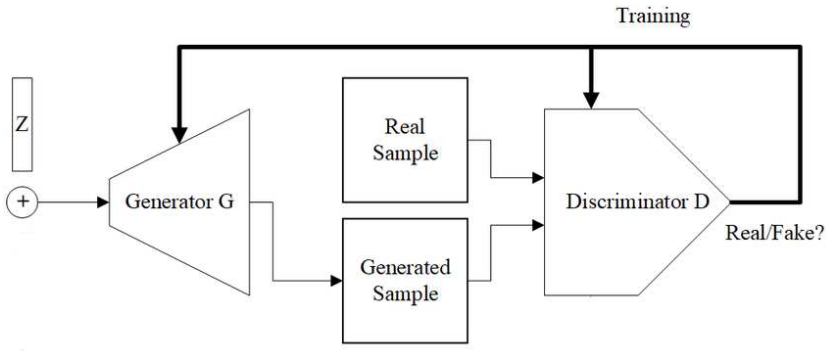
Tekstualni podaci mogu se također koristiti kao ulaz za CNN-ove, ali prvo moraju biti transformirani u odgovarajući format metodama poput tokenizacije i ugrađivanja. Tokenizacija predstavlja dijeljenje teksta na manje jedinice, poput riječi ili znakova (tokena), dok ugrađivanje (engl. *embedding*) predstavlja transformiranje svakog tokena u vektor fiksne dužine.

2.4.3. Generativne kontradiktorne mreže

Generativne kontradiktorne mreže (engl. *generative adversial networks - GAN*) vrsta su dubokih neuronskih mreža koje se koriste za generiranje novih podataka s obzirom na naučene obrasce. Generativni su modeli i većinom koriste konvolucijske neuronske mreže kao osnovne građevne blokove, što znači da uče distribuciju skupa podataka iz kojih izvlače uzorke kojih pomažu pri generiranju novih podataka [9].

Arhitektura GAN-a uključuje dva modela – generator i diskriminator. Generator nastoji stvoriti nove primjere podataka na temelju obrazaca naučenih iz podataka za obuku, dok

diskriminator analizira slike i utvrđuje jesu li slike lažne (generirane) ili originalne. Cilj generatora je stvoriti uzorke dovoljno uvjerljive kako bi zavarali diskriminator, dok diskriminator pokušava nadvladati generator i uhvatiti lažne proizvedene slike (Sl. 2.9).



Slika 2.9: Arhitektura generativnih kontradiktornih mreža (iz [12])

GAN se trenira koristeći stvarne slike kao dio skupa podataka za obuku, što omogućava modelu diskriminatora da nauči razlikovati stvarne slike od generiranih. Generator zatim uzima vektor slučajnih podataka i transformira ga u sliku. Te generirane slike, zajedno s originalnim slikama, unose se u diskriminator koji procjenjuje njihovu autentičnost, dajući ocjene između 0 i 1.

Diskriminator i generator rade u dvostrukoj povratnoj sprezi: diskriminator poboljšava svoje razlikovanje stvarnih i lažnih slika, dok generator poboljšava svoje generiranje slika kako bi zavaralo diskriminator. U igri s nultim zbrojem diskriminator ažurira svoje parametre kada pogrešno klasificira slike, a generator se kažnjava ili nagrađuje na temelju uspjeha zavaravanja diskriminatora.

Cilj u obuci je da generator poboljša svoje performanse do te mjere da diskriminator ne može razlikovati lažne slike od stvarnih, što bi rezultiralo ocjenama od 50% za obje vrste slika. Iako GAN-ovi rijetko dosegnu ovu točku savršenstva, generirane slike često su dovoljno kvalitetne za mnoge primjene.

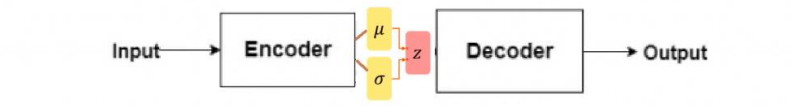
Primjena GAN-ova je najviše u generiranju slika, gdje se isti koriste u slučajevima u kojima su slikovni podaci ograničeni ili nepostojeći za generiranje podataka. Konkretno, primjene uključuju:

- *image-to-image* konverziju,
- *text-to-image* konverziju i
- poboljšavanje slika (engl. *image enhancement*).

2.4.4. Varijacijski autokoderi

Varijacijski autokoderi (engl. *variational autoencoder - VAE*) predstavljaju napredni tip generativnih modela koji kombiniraju karakteristike tradicionalnih autokodera i probabilističkih grafičkih modela. Cilj VAE-a je naučiti latentnu reprezentaciju podataka na način koji omogućava generiranje novih podataka sličnih onima iz trening skupa, a primjenjuju se u različite svrhe – primarno za generiranje slika, tekstova i zvukova [12].

VAE se sastoji od dva dijela – enkodera i dekodera. Enkoder je mreža koja uzima ulazne podatke i mapira ih na distribuciju u latentnom prostoru. Umjesto da enkodira ulaz u točku u latentnom prostoru, VAE enkodira ulaz kao distribuciju (obično gaussovsku). Dekoder je mreža koja uzima uzorak iz latentne distribucije i rekonstruira originalne podatke (Sl. 2.10).



Slika 2.10: Arhitektura varijacijskih autokodera (iz [13])

Latentni prostor predstavlja niz dimenzija koje nisu izravno promatrane ili mjerene, već su apstraktne i služe za komprimiranje i predstavljanje kompleksnih podataka u pojednostavljenom obliku. On omogućava VAE-u da generira nove podatke uzimanjem uzoraka iz naučene distribucije i prolaskom tih uzoraka kroz dekoder. Prilikom treniranja mreže kodirani podaci se analiziraju, a model prepoznavanja generira dva vektora: srednju vrijednost i standardnu devijaciju slika. Na temelju tih vrijednosti formira se distribucija. Ovo se provodi za različita latentna stanja. Dekoder zatim uzima nasumične uzorke iz odgovarajuće distribucije i koristi ih za rekonstrukciju početnih ulaza u mrežu.

Varijacijski autokoderi igraju ključnu ulogu u *text-to-image* generaciji zbog njihove sposobnosti da uče bogate i smisleno strukturirane latentne reprezentacije. VAE uči latentni prostor koji može efikasno sažeti složene informacije iz tekstualnih opisa i koristiti se za generiranje slika koje odgovaraju tim opisima. Dekoder VAE-a može koristiti uzorke iz latentnog prostora kako bi generirao slike koje odražavaju karakteristike skupa za treniranje. Također, tehnikom regularizacije VAE može generirati glatki latentni prostor koji osigurava konzistentnost i koherentnost u generiranim slikama, što je ključno za visoko kvalitetne rezultate u text-to-image generaciji [13].

3. Prompt Invaders

Pokazna videoigra Prompt Invaders naglašavat će inovativan pristup korištenju tehnologije za generiranje vizualnih elemenata na temelju tekstualnih unosa korisnika. Osnovna petlja igre – koja uključuje klasične mehanike poznate iz videoigre Space Invaders – osmišljena je tako da igrač može upravljati glavnim likom, kretati se lijevo i desno po ekranu, te ispaljivati projektile prema neprijateljima koji se nalaze iznad njega. Glavni cilj je uništiti sve neprijatelje prije nego što oni unište igrača na način da mu istroše broj života.

Jedan od najvažnijih aspekata ove videoigre jest dinamičko kreiranje izgleda igre, gdje su svi vizualni elementi prilagodljivi prema korisničkim unosima. To uključuje izgled glavnog lika, neprijatelja, projektila, te pozadinu same igre. Ova prilagodljivost postignuta je integracijom naprednih alata za generiranje slika iz tekstualnih opisa (*text-to-image* generacija), što omogućuje igraču da kroz tematski izbornik unese vlastite ideje i vizije koje će se potom automatski pretvoriti u vizualne elemente igre.

Grafičko sučelje omogućava korisnicima pregled generiranih elemenata, njihov odabir i eventualnu ponovnu generaciju ako nisu zadovoljni prvim rezultatom. Funkcionalnost ponovne generacije (engl. *reprompting*) omogućava igračima da brzo i jednostavno generiraju nove verzije elemenata dok ne postignu željeni izgled. Osim toga, igra uključuje opcije za spremanje i kasniju upotrebu već generiranih elemenata, što dodatno povećava fleksibilnost i korisničko iskustvo.

Sve ove funkcionalnosti dizajnirane su s ciljem stvaranja visoko prilagodljivog i osobnog iskustva za svakog igrača, što čini Prompt Invaders ne samo igrom već i platformom za kreativno izražavanje kroz igru. Implementacija ovih specifikacija omogućuje igračima da sudjeluju u procesu dizajniranja igre na način koji do sada nije bio moguć u tradicionalnim videoigrama.

4. Implementacija pokazne videoigre

Za implementaciju pokazne AI Space Invaders videoigre korišteni su sljedeći alati:

- *Stable Diffusion* AI model za generiranje slika temeljenih na tekstualnim opisima (što je omogućilo stvaranje vizualnih elemenata igre) i
- *Unity* za razvoj svih drugih aspekata videoigre, uključujući korisničko sučelje, mehaniku igre, integraciju s AI modelom i ostale funkcionalnosti.

4.1. Stable Diffusion

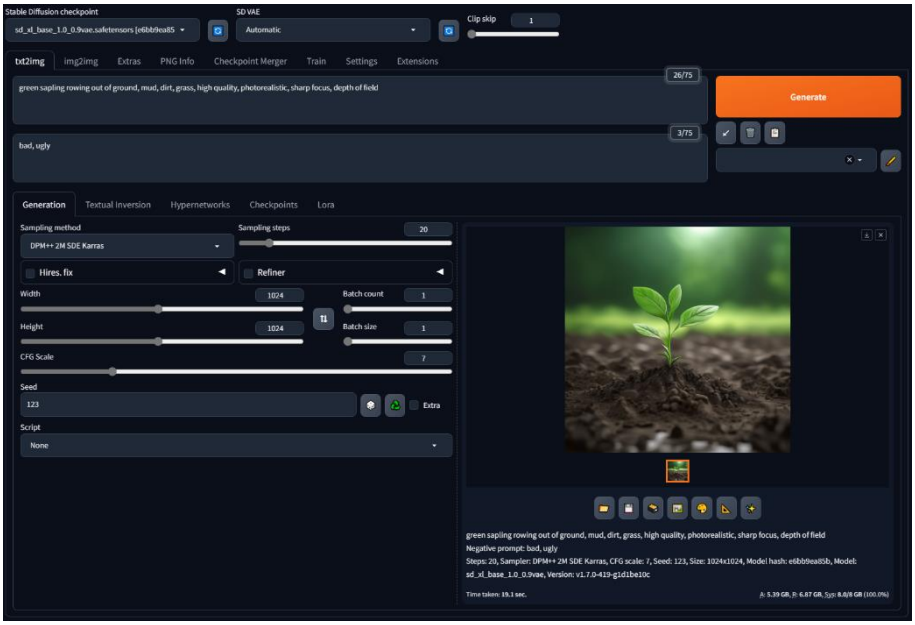
Stable Diffusion je model umjetne inteligencije za generaciju slika temeljen na tekstualnim opisima. Razvijen je kao projekt firme Stability AI s otvorenim izvornim kodom (engl. *open-source*), zbog čega je – između ostalog – i stekao na popularnosti [8].

Model koristi napredne algoritme za stvaranje slika na temelju tekstualnih opisa koje korisnici unose, a treniran je na velikim skupovima podataka koji uključuju milijune slika i pripadajućih tekstualnih opisa. Ovo omogućava modelu da razumije i interpretira različite vizualne stilove i sadržaje. Koristi i koncept latentnog prostora za generiranje slika, što znači da model ne generira slike izravno, već koristi kompleksan matematički prostor za stvaranje detaljnih i koherentnih slika koje odgovaraju tekstualnim uputama.

Stable Diffusion 1 bio je prvi model i postavio temelje za generaciju slika temeljenih na tekstualnim opisima. Stable Diffusion 2 donio je značajna poboljšanja u kvaliteti generiranih slika i preciznosti (Sl. 4.x). U ovoj verziji uvedeni su napredniji trenirani modeli koji poboljšavaju razumijevanje i interpretaciju tekstualnih opisa. Također, dodan je novi algoritam za dekodiranje latentnih vektora, što rezultira detaljnijim i realističnijim slikama. **Stable Diffusion 2** također uključuje poboljšanja u brzini obrade i optimizaciji resursa, čineći proces generacije slika učinkovitijim. **Stable Diffusion 3** predstavlja najnoviju evoluciju modela, s dodatnim unapređenjima u kvaliteti slike, brzini i mogućnostima prilagodbe. Ova verzija je još uvijek u izradi i čeka puštanje u javnost.

Stable Diffusion WebUI Forge je platforma koja omogućava korisnicima da koriste Stable Diffusion model putem web preglednika. Ova platforma pruža grafičko korisničko sučelje

koje olakšava interakciju s modelom, omogućujući korisnicima da unesu tekstualne upite, prilagode parametre generacije, koriste *pluginove* itd. (Sl. 4.1)



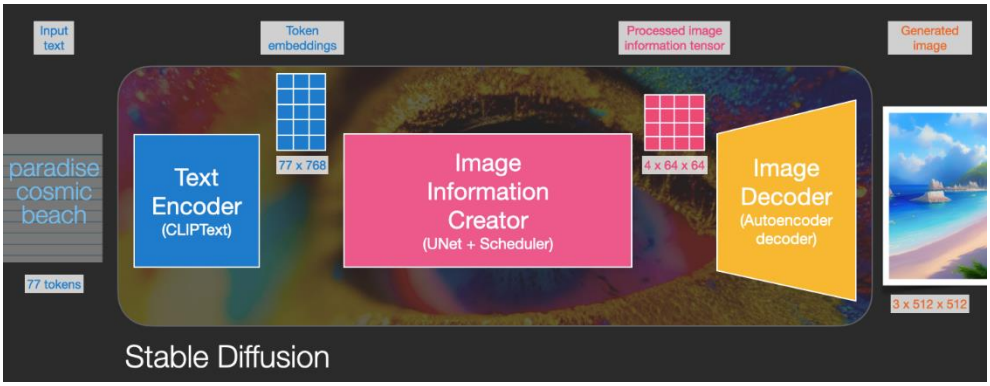
Slika 4.1: Izgled sučelja *Stable Diffusion Web UI Forge* alata

Stable Diffusion Web UI Forge alat zahtijeva vrlo moćne grafičke kartice za uspješno pokretanje, jer proces zahtijeva veliku količinu memorije grafičke kartice (engl. *Video Random Access Memory - VRAM*). Minimalna količina *VRAM-a* za funkcioniranje Stable Diffusiona 2 (koju koristi *Stable Diffusion Web UI Forge*) je 4 GB [17].

Ovaj alat koristit će se kao pristupna točka za *text-to-image* generaciju prilikom izrade Prompt Invaders videoigre.

4.1.1. Proces *text-to-image* generacije unutar modela

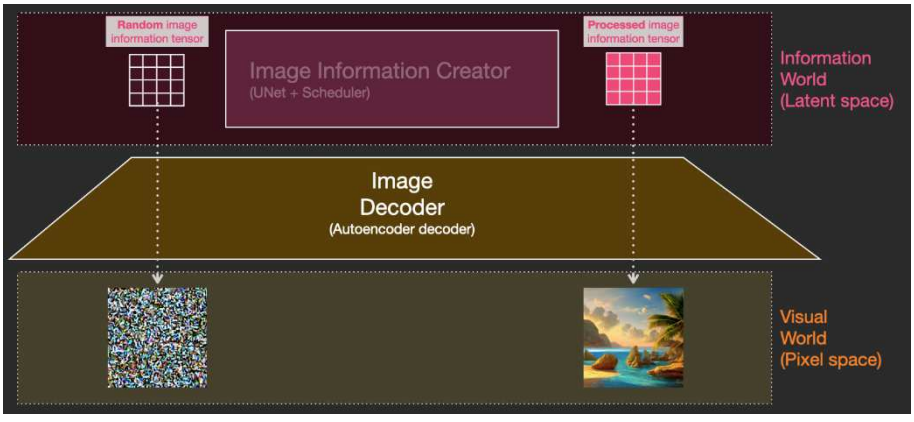
Stable Diffusion koristi kombinaciju konvolucijskih neuronskih mreža, generativnih kontradiktornih mreža i varijacijskih autokodera (između ostaloga) za stvaranje visokokvalitetnih slika iz tekstualnih opisa. Jednostavan prikaz procesa generacije prikazan je na Sl. 4.2.



Slika 4.2: Proces generacije slike iz korisničkog unosa pomoću *Stable Diffusion* modela (iz [14])

Ukratko, proces se izvršava na sljedeći način:

1. Korisnik unosi tekstualni opis slike koju želi generirati. Unosi glavni upit, odnosno *prompt*, i proizvoljno unosi i negativan upit (engl. *negative prompt*) kojim navodi koje značajke model treba izbjegavati prilikom generacije.
2. Odvija se pretvorba tekstualnog opisa u latentni prostor na način da varijacijski autokoder enkodira tekstualni opis u latentni prostor kako bi se stvorila komprimirana reprezentacija slike (Sl. 4.3).
3. Generacija slike vrši se pomoću konvolucijskih neuronskih mreža na način da isti koriste vektore za generiranje slike i obrađuju ih kako bi stvorili vizualni prikaz koji odgovara *promptu*.
4. Generativne kontradiktorne mreže *fine-tuneaju* sliku kako bi se osigurala visoka kvaliteta i realističnost.

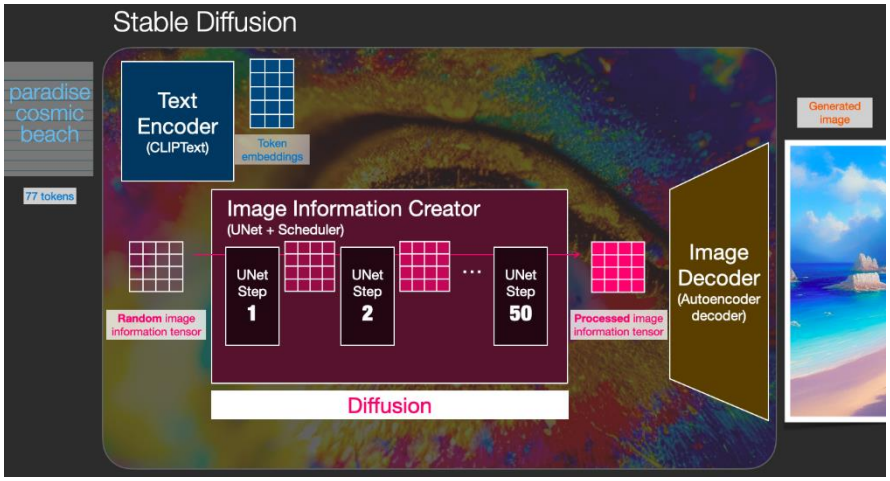


Slika 4.3: Pobliža reprezentacija latentnog prostora u kreatoru slikovnih informacija (iz [14])

Difuzija je proces koji se odvija unutar ružičaste komponente "kreatora informacija o slici". Imajući ugrađene tokene koji predstavljaju ulazni tekst i nasumični početni niz informacija

o slici (također nazvani *latenti),* proces proizvodi informacijski niz koji dekoder slike koristi za slikanje konačne slike [14].

Difuzija se događa u više koraka (Sl. 4.4). Svaki korak radi na ulaznom nizu latenta i proizvodi još jedan niz latenta koji više nalikuje ulaznom tekstu i svim vizualnim informacijama koje je model pokupio sa svih slika na kojima je model treniran.



Slika 4.4: Proces difuzije prilikom *text-to-image* generacije (iz [14])

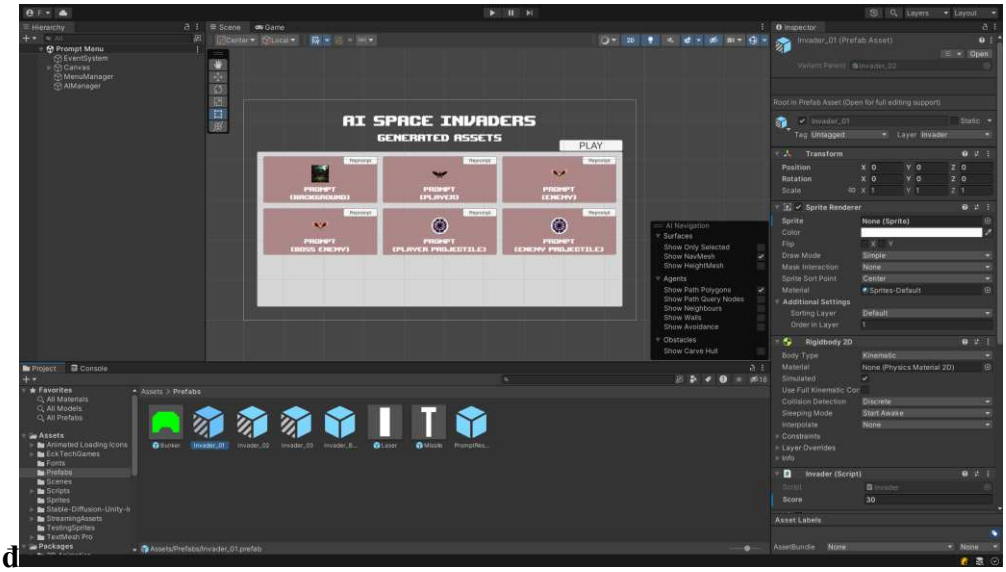
4.2. Unity

Unity je višenamjenski sustav za izradu videoigara (engl. *game engine*) koji omogućava stvaranje interaktivnih iskustava koristeći komponente, objekte i skripte. Ključne komponente u Unityju pridružuju se objektima i proširuju njihovu funkcionalnost, uključujući manipulaciju fizikom, zvukom, korisničkim unosom, materijalima, osvjetljenjem i korisničkim sučeljem.

Kada se uspoređuje popularnost različitih motora za igre, Unity se nalazi na prvom mjestu s 48% tržišnog udjela dok je na drugom mjestu Unreal Engine s 13% tržišnog udjela [18]. Ostali motori, poput CryEnginea ili RAGE-a, koriste se specifično za igre koje razvijaju njihove matične tvrtke.

Razvoj igara u Unityju odvija se preko Unity Editor aplikacije uz izbor softvera za kodiranje, najčešće Visual Studio Code ili IntelliJ Rider. Sučelje Unity Editora može se prilagoditi prema potrebama, s početnim prikazom koji uključuje hijerarhijski pregled objekata u sceni na lijevoj strani, upravitelj datotekama na dnu, i Inspector View s informacijama o odabranim objektima i komponentama na desnoj strani. Na vrhu sučelja moguće je prebaciti

se u prikaz igre pritiskom na gumb Game, a sučelje se može prilagoditi premještanjem prozora ili dodavanjem novih (Sl. 4.5).



Slika 4.5: Sučelje Unity Editora

Unity Asset Store, dostupan iz Unity Editora ili putem web preglednika, nudi sadržaje poput materijala, tekstura, modela, skripti i zvukova, koje je moguće kupiti (ili preuzeti besplatno) i integrirati u projekte.

Za pisanje skripti Unity koristi programski jezik C# i pristup objektno orijentiranom programiranju. Skripte napisane u C# mogu se dodati kao komponente objektima kako bi proširile njihovu funkcionalnost. Poboljšanje performansi obuhvaća uvođenje Unity DOTS (engl. *Data-Oriented Technology Stack*) tehnologije, koja smanjuje fragmentaciju i poboljšava učinkovitost implementacijom višedretvenog koda.

4.3. Konfiguracija i pristup Stable Diffusion sučelju iz Unityja

Stable Diffusion WebUI Forge alat instalira se na osobno računalo putem kloniranja „*stable-diffusion-webui-forge*“ GitHub repozitorija lokalno na disk. Zatim, sam alat pokrećemo putem .bat skripti u korijenskom direktoriju. Međutim, kako bi Unity mogao komunicirati sa sučeljem za programiranje aplikacije (engl. *Application Programming Inteface* – API), potrebno ga je izložiti za spajanje (engl. *expose*) prilikom paljenja. To činimo dodavanjem

zastavice (engl. *flag*) `--api` u *.bat* datoteku na mjestu `set COMMANDLINE_ARGS` (Kod 4.1).

```
@echo off
set PYTHON=
set GIT=
set VENV_DIR=
set COMMANDLINE_ARGS=--api
@REM Uncomment following code to reference an existing A1111
checkout.
@REM set A1111_HOME=Your A1111 checkout dir
@REM
@REM set VENV_DIR=%A1111_HOME%/venv
@REM set COMMANDLINE_ARGS=%COMMANDLINE_ARGS% ^
@REM  --ckpt-dir %A1111_HOME%/models/Stable-diffusion ^
@REM  --hypernetwork-dir %A1111_HOME%/models/hypernetworks ^
@REM  --embeddings-dir %A1111_HOME%/embeddings ^
@REM  --lora-dir %A1111_HOME%/models/Lora

call webui.bat
```

Kod 4.1: Izgled „*webui-user.bat*“ datoteke

Sada je moguće pristupiti krajnjoj točki alata (engl. *endpoint*) iz Unityja putem lokalne adrese i porta. Međutim, s obzirom na to da je osim pozadine potrebno generirati elemente koji mogu poslužiti kao objekti u igri, mora se alatu signalizirati da generirane slike moraju imati transparentnu pozadinu i biti siluete traženog upita (engl. *sprite*). Ovdje u pomoć dolaze *pluginovi* u WebUI Forge alatu – specifično Layer Diffusion koji služi za generiranje slika iz tekstualnih opisa s nevidljivom, odnosno transparentnom pozadinom (više kasnije).

Znači, potrebno je putem C# skripti unutar Unity Editora integrirati Stable Diffusion u Unity za uspješnu komunikaciju i slanje potrebnih parametara u alat.

4.4. Integracija Stable Diffusiona u Unity

U svrhu integracije Stable Diffusiona u Unity potrebno je implementirati pojedine C# skripte u Unityju za uspješnu integraciju i pristupanje SD WebUI Forge sučelju te SD modelu.

Prvo ćemo definirati podatke potrebne za pristupanje API-ju i lokacije spremanja, uz neke glavne parametre generacije. U tu svrhu kreiramo klasu `SDSettings` u kojoj definiramo takve atribute (Kod 4.2).

```
// Struktura podataka za specificiranje postavki za Stable
// Diffusion server API ili predodređene postavke korištene
// prilikom dodavanja novog StableDiffusionImage na Unity
// GameObject
public class SDSettings : ScriptableObject
{
    public string StableDiffusionServerURL  =
„http://127.0.0.1:7860";
    public string ModelsAPI = "/sdapi/v1/sd-models";
    public string TextToImageAPI = "/sdapi/v1/txt2img";
    public string OptionAPI = "/sdapi/v1/options";
    public string ProgressAPI = "/sdapi/v1/progress";
    public string OutputFolder = "/streamingAssets";
    public string sampler = "Euler a";
    public int width = Constants.GeneratedSpriteWidth;
    public int height = Constants.GeneratedSpriteHeight;
    public int steps = 35;
    public float cfgScale = 7;
    public long seed = -1;
    public bool useAuth = false;
}
```

Kod 4.2: Izgled `SDSettings` klase

Parametri poput *width* i *height* se koriste na puno mjesta u kodu, stoga su njihove vrijednosti izdvojene u zasebnu `Constants` klasu (Kod 4.3). U istoj se uz ove vrijednosti nalaze i drugačije definirane vrijednosti za duljinu i širinu pozadine (jer je bitno da je pozadina većih dimenzija kako ne bi došlo do degradiranje kvalitete prilikom prikaza preko cijelog ekrana).

```
public static class Constants
{
    public const int GeneratedSpriteWidth = 512;
    public const int GeneratedSpriteHeight = 512;
```

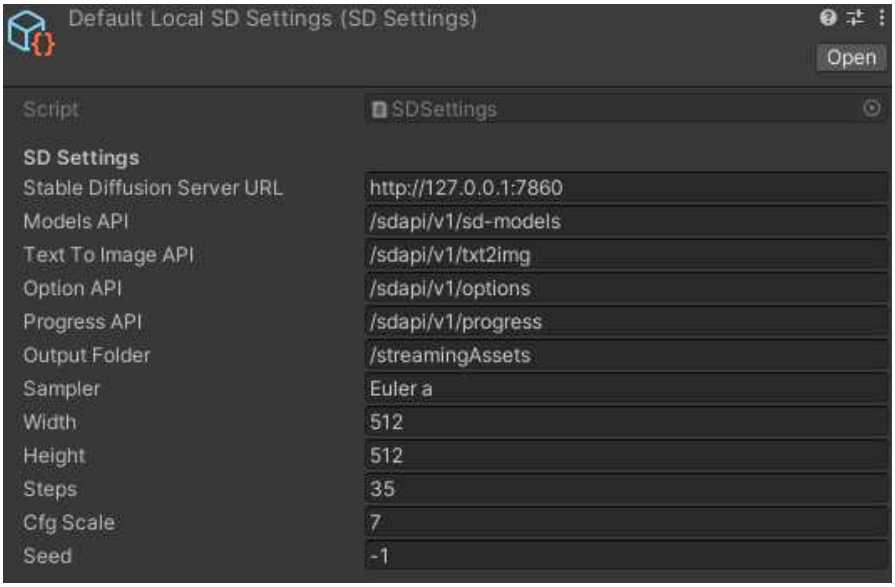
```
        public const int GeneratedBackgroundWidth = 1024;
        public const int GeneratedBackgroundHeight = 1024;
    }
```

Kod 4.3: Izgled statične `Constants` klase

Ostali definirani parametri uključuju:

- *sampler* - označava algoritam uzorkovanja koji se koristi za generaciju slika (uzorkovači su metode koje određuju kako se uzorkuju i generiraju pikseli slike tijekom procesa stvaranja),
- *steps* - broj iteracija kroz koje model prolazi kako bi poboljšao kvalitetu slike,
- *cfgScale* - kontrolira koliko će model pratiti tekstualni opis u odnosu na svoju kreativnost ili slobodu u generiranju slike (viša vrijednost znači da će model biti precizniji u ispunjavanju opisa),
- *seed* - početna vrijednost za slučajni generator koji model koristi za generiranje slika (kada je *seed* postavljen na -1 model generira slučajan *seed* pri svakom pokretanju, što rezultira različitim slikama svaki put) i
- *useAuth* – postavljen na *false* jer nećemo koristiti zaštitnu autentikaciju prilikom pristupa API-ju.

Klasu ćemo definirati kao `ScriptableObject` - poseban tip objekta koji se koristi za pohranu podataka i konfiguracija unutar Unity okruženja. Zatim ćemo je integrirati posebnom tipu objekta u Unityju za definiranje postavki kojeg ćemo dalje koristiti u skripti `StableDiffusionConfiguration` (Sl. 4.6).



Slika 4.6: Izgled objekta `DefaultLocalSDSettings`

Sada se definirane postavke mogu integrirati u glavnu konfiguracijsku skriptu `StableDiffusionConfiguration` čija je zadaća iskoristiti definirane postavke unutar `DefaultLocalSDSettings` objekta za generaciju. Pojednostavljen prikaz globale konfiguracijske skripte (bez definiranih tijela metoda) prikazan je u Kodu 4.4.

```
// Globalna Stable Diffusion konfiguracija parametara.
[ExecuteInEditMode]
public class StableDiffusionConfiguration : MonoBehaviour
{
    [SerializeField]
    public SDSettings settings;

    [SerializeField]
    public string[] samplers = new string[]{
        "Euler a", "Euler", "LMS", "Heun", "DPM2", "DPM2 a",
"DPM++ 2S a", "DPM++ 2M", "DPM++ SDE", "DPM fast", "DPM
adaptive",
        "LMS Karras", "DPM2 Karras", "DPM2 a Karras", "DPM++
2S a Karras", "DPM++ 2M Karras", "DPM++ SDE Karras", "DDIM",
"PLMS"
    };
    [SerializeField]
    public string[] modelNames;
```

```
        // Vraća listu svih dostupnih SD modela.
        IEnumerator ListModelsAsync()

        // Postavlja model koji će SD koristiti.
        public IEnumerator SetModelAsync(string modelName)
    }
```

Kod 4.4: Potpisi metoda koje koristi globalna `StableDiffusionConfiguration` klasa

U svrhu primjenjivanja ovih postavki na globalnoj razini, potrebno je kreirati objekt koji će sadržavati ovu skriptu. Bitno je da taj objekt bude kreiran u samom početku igre te da se prilikom promjena scena isti ne uništi, nego ostane aktivan. Stoga, kreiramo objekt *AIManager* na koji će se prikačiti skripta `StableDiffusionConfiguration`, ali i jednostavna `AIManager` skripta koja će definirati zadržavanje aktivnosti objekta prilikom izmjena scena (Kod 4.5). Jednostavnim kreiranjem instance i naredbom `DontDestroyOnLoad(gameObject)` govorimo skripti da sačuva objekt prilikom učitavanja drugih scena. Skripta će također sadržavati spremljene generirane rezultate (slike i pripadajuće *promptove*) za lakši pristup iz ostalih skripti.

```
public sealed class AIManager : MonoBehaviour
{
    public Dictionary<PromptTheme, Prompt> PromptResults =
new();

    public static AIManager Instance;

    private void Awake()
    {
        Instance = this;
        DontDestroyOnLoad(gameObject);
    }
}
```

Kod 4.5: `AIManager` skripta s definiranim rezultatima generacije

Završni korak integracije uključuje definiranje parametara koji će se koristiti prilikom slanja web zahtjeva SD API-ju. To se lako definira kreiranjem obične `SDParamsInTxt2Img` klase s definiranim parametrima koje API očekuje prilikom slanja zahtjeva. Parametara je

puno i svaki definira određene karakteristike generacije, ali postavljene vrijednosti su dovoljno dobre stoga se neće u detaljima opisivati i modificirati svaki.

Međutim, ako želimo generirati bilo što osim pozadine igre potrebno je ukloniti pozadinu iz generirane teksture, odnosno generirati nešto slično objektu u 2D igrama. Već spomenuti Layer Diffusion *plugin* nam ovo pospješuje, ali potrebno ga je integrirati u SD WebUI Forge i prilikom slanja zahtjeva *„reći“* API-ju da uključi to proširenje u proces generacije slike.

4.5. Layer Diffusion proširenje i integracija

Layer Diffusion predstavlja napredan alat za rad sa slojevima u Stable Diffusion modelu. Omogućava manipulaciju slikovnih slojeva i micanje pozadine iz generiranih slika metodama poput latentne transparentnosti (engl. *latent transparency*) – tehnike koja upravlja prozirnošću različitih dijelova slike tijekom procesa generacije. U kontekstu generativnih modela poput Stable Diffusiona, već spomenuti latentni prostor predstavlja komprimirani prikaz slike kojim Layer Diffusion manipulira za postizanje transparentnosti generiranih slika [16].

Instalacija *plugina* izvršava se lako putem WebUI Forge sučelja lijepljenjem GitHub linka repozitorija u sučelje instalacije *pluginova*, ali uključivanje istog prilikom pristupanja API-ju iz Unityja malo je složenije.

Potrebno je prilikom u SDParamsInTxt2Img klasu dodati parametar `alwayson_scripts` kojeg API (ako je definiran) iščitava i time uključuje definiranu skriptu s pripadajućim parametrima (Kod 4.6). Za uključivanje definiramo rječnik skripti u koji stavljamo „LayerDiffuse“ skriptu i pripadajuće argumente (`args`) u kojima definiramo da je metoda koju želimo koristiti broj 4, odnosno *(SD1.5) Only Generate Transparent Image (Attention Injection)*. Mehanizam pažnje (engl. *attention injection*) omogućava modelima da se usmjere na različite dijelove ulaznih podataka prema njihovoj važnosti.

```
[CanBeNull] public Dictionary<string, object>
alwayson_scripts = new()
{
    {
```

```
            "LayerDiffuse", new Dictionary<string, object>
            {
                {
                    "args", new object[]
                    {
                        new Dictionary<string, object>
                        {
                            { "method", 4 }
                        }
                    }
                }
            }
        }
};
```

Kod 4.6: Definicija `alwayson_scripts` objekta unutar `SDParamsInTxt2Img` klase

Međutim, ovaj *plugin* želimo koristiti za sve osim pozadine igre, stoga ga je potrebno prije slanja zahtjeva API-ju isključiti za slučaj kada se generira pozadina. Jednostavno, prije generacije ćemo definirati rječnik `alwayson_scripts` kao prazan kako bi se generacija izvršavala normalno, bez uključivanja Layer Diffusion *plugina*. Također, s obzirom na to da se generira pozadina, izmijenit ćemo i duljinu i širinu generirane slike tako da bude veća za bolji i kvalitetniji prikaz pozadinske slike (Kod 4.7).

```
if (promptTheme == PromptTheme.Background)
{
    // 'sd' predstavlja objekt tipa SDParamsInTxt2Img
    sd.alwayson_scripts = new Dictionary<string, object>();
    sd.width = Constants.GeneratedBackgroundWidth;
    sd.height = Constants.GeneratedBackgroundWidth;
}
```

Kod 4.7: Izmjena parametara prilikom generacije pozadinske slike

4.6. Proširivanje upita

Prilikom ulaska u igru korisnik popunjava praznine kako bi definirao stil igre. Međutim, slanje isključivo popunjenih praznina kao *promptove* nije dovoljno za kvalitetnu generaciju slika. Stoga, iscrpnim testiranjem SD modela odlučilo se proširiti upite dodatnim „skrivenim“ upitima koji će se nadodati na postojeće korisničke *promptove*.

U tu svrhu *promptovi* su najprije kategorizirani prema temi tipom nabrajanja (engl. *enum*) naziva `PromptTheme`, ali i prema tipu (radi li se o glavnom ili o negativnom *promptu*) *enumom* `PromptType` (Kod 4.8).

```
public enum PromptTheme
{
    Background,
    Player,
    Enemy,
    BossEnemy,
    PlayerProjectile,
    EnemyProjectile,
}
public enum PromptType
{
    Main,
    Negative
}
```

Kod 4.8: Definicija *enuma* za razdijelu *promptova* prema temi i prema tipu

Zatim je kreirana nova statična klasa `PromptExtensions` sa statičnim rječnikom `Extensions` u kojem će se prema temi i tipu *prompta* iz rječnika izvući dodatne riječi koje će se nadodati (razdvojeni zarezima) na korisničke *promptove*. Npr. ekstenzije za *promptove* tipa *Background* i *Player* bit će znatno drugačije jer su teme različite, a time i zahtjevi generacije (Kod 4.9).

```
public static readonly Dictionary<PromptTheme,
Dictionary<PromptType, List<string>>> Extensions = new()
{
    {
        PromptTheme.Background, new()
        {
            { PromptType.Main, new() { "land", "background" }
},
            { PromptType.Negative, new() { "words", "text",
"letters", "realistic", "photograph", "logo", "watermark",
"nudity" } }
        }
    },
    {
```

```
            PromptTheme.Player, new()
            {
                { PromptType.Main, new() { "spaceship",
"vertical", "top perspective", "90 degrees angle", "centered"
} },
                { PromptType.Negative, new() { "words", "text",
"letters", "realistic", "photograph", "logo", "watermark",
"nudity" } }
            }
        },
};
```

Kod 4.9: Definicije ekstenzija za *promptove* tipa *Background* i *Player*

Znači, svakim slanjem zahtjeva za *text-to-image* generaciju na SD API, ovisno o temi će se na *prompt* „zakačiti“ i dodatni glavni i negativni *promptovi* kako bi generacija bila što preciznija i zadovoljavajuća.

Kako bismo upotrijebili ove ekstenzije, kreirana je statična klasa `PromptHelper` s metodom `ExtendPrompt` čija je zadaća zaprimljeni *prompt* proširiti dodatnim glavnim i negativnim *promptovima* (Kod 4.10).

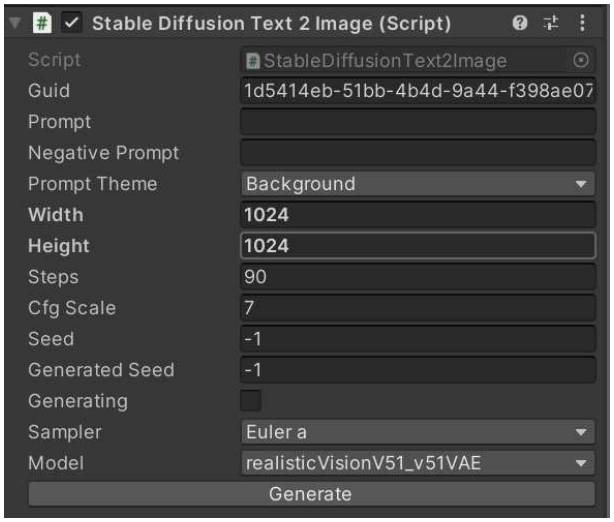
```
public static string ExtendPrompt(string prompt, PromptTheme
theme, PromptType type)
    => (type == PromptType.Main ? prompt : "") + (type ==
PromptType.Main ? "," : "") +
        string.Join(", ",
PromptExtensions.Extensions.GetValue(theme).GetValue(type));
```

Kod 4.10: `ExtendPrompt` metoda za proširivanje *promptova* pozvana prije slanja upita API-ju

4.7. Proces *text-to-image* generacije i primjenjivanja slika

Nakon podešavanja parametara i upita vrijeme je za sam proces *text-to-image* generacije slanjem upita na SD WebUI Forge API nakon što se praznine u tematskom izborniku ispune. Da bi se to pospješilo potrebno je svakoj slici rezultata u izborniku generiranih elemenata (Sl. 3.2) dodati kreiranu skriptu `StableDiffusionText2Image` – glavnu generacijsku

skriptu koja pristupa API-ju i nakon generacije slike primjenjuje generiranu sliku kao *Image* komponentu objekta (Sl. 4.7).



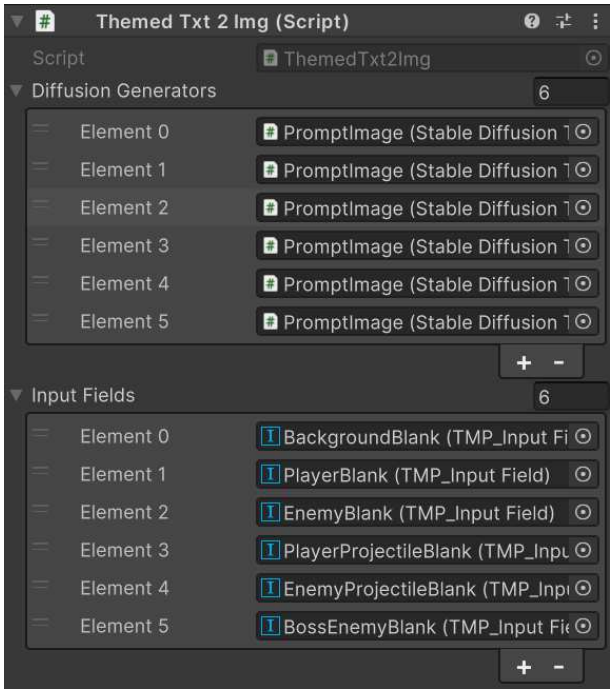
Slika 4.7: Izgled `StableDiffusionText2Image` skripte

Nadalje, gore spomenutoj skripti pristupa druga skripta `ThemedTxt2Img` – pomoćna skripta koja je dodana kao komponenta na već spomenuti `AIManager`, a ima pristup:

- svim objektima sa `StableDiffusionText2Image` skriptom (tzv. *diffusion generators*) i
- svim poljima za unos (engl. *input field*), odnosno prazninama (Sl. 4.8).

`ThemedTxt2Img` skripta ima metodu `StartTxt2ImgGeneration` koja se pokreće pritiskom na gumb za odlazak na izbornik generiranih elemenata (Sl. 3.2) te:

- sakuplja sve tekstove napisane u prazninama tematskog izbornika,
- vodi na izbornik generiranih elemenata i
- pokreće `GenerateAsync` metodu u svim `StableDiffusionText2Image` skriptama (jedan po jedan) – što označava početak samog generacijskog procesa.



Slika 4.8: `ThemedTxt2Img` skripta s definiranim generatorima i poljima za unos

Proces sakupljanja teksta u poljima za unos vrši se u lokalnoj metodi `SetInputPrompts` koja sve korisničke unose sprema u listu tipa `Prompt` – klasu koja služi za enkapsulaciju pojedinog korisničkog unosa s obzirom na temu, ali i pohranjivanje rezultata generacije (Kod 4.11).

```
public class Prompt
{
    public string Text { get; set; }
    public PromptTheme Theme { get; set; }
    [CanBeNull] public Sprite Result { get; set; }
}
```

Kod 4.11: `Prompt` klasa korištena za spremanje korisničkog unosa s obzirom na temu

Također, svaka praznina na sebi ima jednostavnu skriptu `PromptThemedInput` koja kategorizira pojedino polje za unos, što je ključno za cijeli proces (Kod 4.12).

```
public class PromptThemedInput : MonoBehaviour
{
    [SerializeField] public PromptTheme promptTheme;
}
```

Kod 4.12: `PromptThemedInput` klasa pridijeljena svakom polju za unos na tematskom izborniku

Glavna metoda `PromptHelper` klase `InvokeTxt2ImgGeneration` sada direktno komunicira sa `StableDiffusionText2Image` skriptom svake slike rezultata u izborniku generiranih elemenata, postavlja odgovarajuću temu i proširene *promptove* na generator te pokreće samu generaciju (Kod 4.13).

```
        public static IEnumerator
InvokeTxt2ImgGeneration(MonoBehaviour monoBehaviour,
            StableDiffusionText2Image diffusionGenerator,
string prompt, PromptResult promptResult)
        {
            ...
            diffusionGenerator.PromptTheme = theme;
            diffusionGenerator.Prompt = ExtendPrompt(prompt,
theme, PromptType.Main);
            diffusionGenerator.NegativePrompt =
ExtendPrompt("", theme, PromptType.Negative);

            if (!diffusionGenerator.generating)
            {
                  monoBehaviour.StartCoroutine(diffusionGener
            ator.GenerateAsync(promptResult.UpdateGenerationP
            rogress));
            }
            while (diffusionGenerator.generating)
            {
                yield return null;
            }
            ...
        }
```

Kod 4.13: `InvokeTxt2ImgGeneration` metoda pozvana prilikom svake generacije

4.7.1. Konfiguracija mapa i naziva

Kako bi se elementi u igri mogli izmijeniti u generirane slike potrebno je iste spremiti u odgovarajuću mapu iz koje će Unity tokom igranja moći čitati podatke. Da bi se putanji moglo pristupiti prilikom igranja igre spremat ćemo sve slike u Unityjevu `Application.streamingAssets` putanju. Ta putanja služi za spremanje datoteka u nekomprimiranom obliku i lak pristup istima dok je igra pokrenuta.

Prilikom text-to-image generacije svaki generirani element igre spremat će se u ovu mapu, a time i prikazati u galeriji prethodno generiranih elemenata kojoj će se moći pristupiti za zamjenu bilo kojeg elementa s nekim već prethodno generiranim. Radi deskriptivnosti struktura imena spremljenog elementa bit će u sljedećem formatu: tema_korisnickiUnos.png (Kod 4.14).

```
private void SetupFolders()
{
  ...
  var root = Application.streamingAssetsPath;
  var mat = Path.Combine(root, "SDImages");
  filename = Path.Combine(mat,
$"{promptTheme.ToString().ToLower()}_{prompt.Split(", ")[0]}.png");

  if (!Directory.Exists(root))
      Directory.CreateDirectory(root);
  if (!Directory.Exists(mat))
      Directory.CreateDirectory(mat);

  if (File.Exists(filename))
      File.Delete(filename);
  ...
}
```

Kod 4.14: `SetupFolders` metoda pozvana prije svake *text-to-image* generacije

4.7.2. Podešavanje parametara i slanje

Prije same generacije potrebno je postaviti model koji će API koristiti prilikom generacije. Predefinirani model – *realisticVisionV51_v51VAE* – koji se instalira na računalo prilikom instalacije WebUI Forge alata dovoljno je dobar za potrebe *text-to-image* generacije elemenata, a i funkcionira dobro s Layer Diffusion proširenjem. Za dohvat dostupnih modela s API-ja potrebno je pristupiti predefiniranom *endpointu "/sdapi/v1/sd-models"* kako bi se u slanjem zahtjeva za generacijom mogao predati odabran model. Skripta StableDiffusionConfiguration će pomoću metode `ListModelsAsync()` dohvatiti listu modela i spremiti ih u varijablu konfiguracije `modelNames` (Kod 4.15).

```
IEnumerator ListModelsAsync()
{
```

```
        var url = settings.StableDiffusionServerURL +
settings.ModelsAPI;

        var request = new UnityWebRequest(url, "GET");
        request.downloadHandler = (DownloadHandler)new
DownloadHandlerBuffer();
        request.SetRequestHeader("Content-Type",
"application/json");

        yield return request.SendWebRequest();

        try
        {
            var ms =
JsonConvert.DeserializeObject<Model[]>(request.downloadHandle
r.text);
            modelNames = ms.Select(m =>
m.model_name).ToArray();
        }
        catch (Exception)
        {
            ...
        }
    }
```

Kod 4.15: `ListModelsAsync` metoda za dohvat dostupnih SD modela

Metoda za postavljanje modela `SetModelAsync(string modelName)` funkcionira na sličan način, uzimajući ime modela kao parametar koji se šalje kao atribut `sd_model_checkpoint` unutar objekta kojeg API očekuje u tijelu zahtjeva (Kod 4.16).

```
            HttpWebRequest httpWebRequest =
(HttpWebRequest)WebRequest.Create(url);
            httpWebRequest.ContentType = "application/json";
            httpWebRequest.Method = "POST";
            using (var streamWriter = new
StreamWriter(httpWebRequest.GetRequestStream()))
            {
                var sd = new SDOption
                {
                    sd_model_checkpoint = modelName
```

```
            };

            var json = JsonConvert.SerializeObject(sd);
            streamWriter.Write(json);
        }
```

Kod 4.16: Dio `SetModelAsync` metode postavljanje SD modela

Sad kada smo API-ju rekli koji model da koristi prilikom generacije potrebno je pristupiti *endpointu* za generaciju *"/sdapi/v1/txt2img"* uz potrebne parametre u tijelu zahtjeva (engl. *request body*). Već spomenuta klasa SDParamsInTxt2Img poslužit će kako bismo poslali validan *request body*, a popunit ćemo je pripadajućim unesenim vrijednostima `StableDiffusionText2Image` skripte (Sl. 4.7). Kao što je spomenuto u poglavlju 4.5, za skoro sve elemente (osim elemenata sučelja i pozadine igre) koristit ćemo dodatan parametar `alwayson_scripts` kako bismo rekli SD API-ju da uključi Layer Diffusion proširenje prilikom generacije. Također, za elemente sučelja (gumb i pozadina sučelja) koristit ćemo dodatan parametar `tiling` kojim ćemo – kao što ime govori – dobiti sliku nalik popločenoj teksturi, što je idealno za elemente sučelja (Kod 4.17).

```
        if (promptTheme is PromptTheme.UIBackground or
PromptTheme.UIButton)
        {
            sd.alwayson_scripts = new();
            sd.tiling = true;
            if (promptTheme is PromptTheme.UIButton)
            {
                sd.width = Constants.GeneratedUiButtonWidth;
                sd.height =
Constants.GeneratedUiButtonHeight;
            }
        }
```

Kod 4.17: Dio `GenerateAsync` metode za uključivanje `tiling` opcije za elemente sučelja

Sad kada su svi parametri uključujući i *promptove* podešeni možemo napraviti poziv API-ju s parametrima u *request bodyju*. API sada započinje procesiranje zahtjeva i proces generacije slike, a prilikom završetka vraća tekstualnu reprezentaciju JSON objekta kojeg ćemo deserijalizirati u klasu SDResponseTxt2Img (Kod 4.18).

```
class SDResponseTxt2Img
{
```

```
    public string[] images;
    public SDParamsOutTxt2Img parameters;
    public string info;
}
```

Kod 4.18: `SDResponseTxt2Img` klasa koja predstavlja tip povratnog objekta nakon *text-to-image* generacije

Ako se pri generaciji koristio Layer Diffusion proširenje atribut `images` imat će dvije slike, od kojih je prva s uključenom pozadinom, a druga bez. Nama je naravno u tom slučaju potrebna druga, stoga ćemo nju konvertirati iz `Base64String` formata slike u listu bajtova koje ćemo zapisati na lokaciju definiranu u kodu 4.14. Cijeli proces prikazan je u kodu 4.19. Ovime je proces generiranja i spremanja slike iz proširenog korisničkog unosa završen.

```
        using var request =
            new
UnityWebRequest(sdc.settings.StableDiffusionServerURL +
sdc.settings.TextToImageAPI, "POST");
        request.uploadHandler = new
UploadHandlerRaw(bodyRaw);
        request.downloadHandler = new
DownloadHandlerBuffer();
        request.SetRequestHeader("Content-Type",
"application/json");

        request.SendWebRequest();

        while (!request.isDone || request.result ==
UnityWebRequest.Result.InProgress)
        {
            ...
        }

        string result = request.downloadHandler.text;

        SDResponseTxt2Img jsonResponse =
JsonConvert.DeserializeObject<SDResponseTxt2Img>(result);

        ...
```

```
        var shouldRetrieveTransparentImage = promptTheme is
PromptTheme.Background or PromptTheme.UIBackground or
PromptTheme.UIButton;
        byte[] imageData =
Convert.FromBase64String(shouldRetrieveTransparentImage ?
jsonResponse.images[0] : jsonResponse.images[1]);

        File.WriteAllBytes(filename, imageData);
```

Kod 4.19: Glavni dio `GenerateAsync` metode u kojoj dolazi do *text-to-image* generacije

Naravno – s obzirom na to da je trajanje zahtjeva dugo zbog procesa generacije – potrebno je da igra tada ne zastane nego da korisniku pruži vizualnu indikaciju da je generacija za specifičan element u toku, što ćemo u nastavku i implementirati.

4.7.3. Prikaz formiranja slike i postotka učitavanja

Kako bismo spriječili zamrzavanje igre tokom generacije koristit ćemo Unityjeve tzv. korutine (engl *coroutines*) - posebne vrste funkcija koje omogućavaju izvođenje operacija koje zahtijevaju vrijeme – poput čekanja, animacija, ili poziva na web servise – bez blokiranja glavne grane igre (engl. *main thread*).

Potrebno je prenamijeniti već spomenutu `InvokeTxt2ImgGeneration` metodu statične klase `PromptHelper` da regulira učitavanje tokom generacije. S obzirom na to da se radi o statičnoj klasi, a ona ne može naslijediti Unityjevu `MonoBehaviour` klasu dodat ćemo `MonoBehaviour` kao parametar `InvokeTxt2ImgGeneration` metode te će svaka `MonoBehaviour` klasa predati `this` kao parametar pri pozivu metode (Kod 4.20). Metoda će unutar sebe pokrenuti `GenerateAsync()` korutinu i svaku sekundu ažurirati učitavanje i napredak generiranja slike za pojedini element.

```
                yield return
PromptHelper.InvokeTxt2ImgGeneration(this,
diffusionGenerator, matchingPrompt.Text,
matchingPromptResult);
```

Kod 4.20: Primjer poziva `InvokeTxt2ImgGeneration` metode iz `MonoBehaviour` klase

`While` dio koda 4.19 izvršavat će se cijelo vrijeme dok generacija traje i koristit ćemo ga kao povratnu funkciju (engl. *callback function*) koja će propagirati postotak generacije nazad u `PromptResult` klasu, ali i napredak stvaranja elementa. Da bismo pristupili postotku

generacije, SD API nudi *endpoint "/sdapi/v1/progress"* koji vraća objekt s informacijama o postotku generacije (`progress`), trenutno formiranoj slici (`current_image`) i ostalim informacijama koje nećemo koristiti. Metoda `GetGenerationData()` služit će za pristup ovom *endpointu* i dohvat ovih informacija (Kod 4.21).

```
    (string imageData, float percentage) GetGenerationData()
    {
        var url = sdc.settings.StableDiffusionServerURL +
sdc.settings.ProgressAPI;

        using WebClient client = new WebClient();
        string responseBody = client.DownloadString(url);

        SDProgress sdp =
JsonConvert.DeserializeObject<SDProgress>(responseBody);

        return (sdp.current_image, sdp.progress);
    }
```

Kod 4.21: `GetGenerationData` metoda za ekstrakciju trenutne slike i postotka generacije

Dohvaćene informacije potrebno je prikazati u sučelju, stoga ćemo u `while` dijelu:

- dohvatiti trenutnu sliku i postotak generacije,
- spremiti dohvaćenu sliku na disk i zamijeniti element s istom,
- pozvati *callback* funkciju za prikaz postotka nad elementom i
- reći Unityju da stane na jednu sekundu prije ponovnog prolaska kroz petlju kako bi se spriječili prečesti prolasci koji bi rezultirali lošijim performansama (Kod 4.22).

```
        while (!request.isDone || request.result ==
UnityWebRequest.Result.InProgress)
        {
            var (imageData, percentage) =
GetGenerationData();

SaveAndLoadImage(Convert.FromBase64String(imageData ?? ""));
            loadingCallback?.Invoke((int)(percentage * 100));
            yield return 1.0f;
        }
```

Kod 4.22: `While` dio `GenerateAsync` metode koji regulira ponašanje tokom generacije

Također, svaki `PromptResult` imat će `LoadingSpinner` objekt koji će se prikazati i animirati početkom generacije, a sakriti na kraju (Sl. 4.10).

Slika 4.9: Prikaz učitavanja prilikom generacije pojedine slike

Funkcionalnost `SaveAndLoadImage` metode – ključne za spremanje i prikaz tranzicijskih slika i finalne slike – bit će opisana u idućem potpoglavlju.

4.7.4. Primjenjivanje generirane slike

Da bismo tokom i pri završetku procesa generacije prikazali generirane slike na za to predviđenom mjestu pojedinog elementa potrebno je:

- proslijediti sliku kao listu bajtova `SaveAndLoadImage` metodi,
- kreirati Texture2D klasu iz podataka slike,
- iskoristiti Texture2D teksturu za kreiranje Sprite klase i
- primijeniti kreirani *sprite* u *Image* komponentu pojedinog elementa.

Međutim, kako bismo osigurali ispravan prikaz formiranja slike tokom generacije, dodat ćemo uvjet da se spremanje ne izvršava sve dok lista bajtova nije popunjena nekim bajtovima. Ovime izbjegavamo čudne artefakte koji nastanu kada se kao slika elementa pokuša primijeniti prazna lista bajtova. Kompletna implementacija vidljiva je u kodu 4.23.

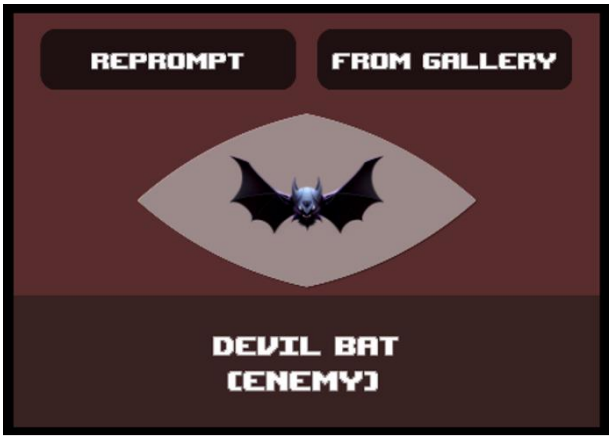
```
private void SaveAndLoadImage(byte[] imageData)
{
    if (imageData.Length == 0)
    {
        return;
    }
```

```
            try
            {
                File.WriteAllBytes(filename, imageData);

                if (!File.Exists(filename))
                {
                    return;
                }

                var texture = new Texture2D(2, 2);
                texture.LoadImage(imageData);
                texture.Apply();

                LoadIntoImage(texture);
            }
            catch (Exception e)
            {
                Debug.LogError(string.Join("\n\n", e.Message,
          e.StackTrace));
            }
        }

        private void LoadIntoImage(Texture2D texture)
        {
            try
            {
                Sprite sprite = Sprite.Create(texture, new
    Rect(0, 0, texture.width, texture.height), Vector2.zero);

                if (GetComponent<Image>() != null)
                {
                    Image image = GetComponent<Image>();

                    if (image != null)
                    {
                        image.sprite = sprite;
                    }
                }
                else
                {
```

```
                    Debug.LogWarning("Image component not found
on the GameObject.");
                }
            }
            catch (Exception e)
            {
                Debug.LogError(string.Join("\n\n", e.Message,
e.StackTrace));
            }
        }
```

Kod 4.23: Metode spremanja i učitavanja spremljene slike za pojedini element

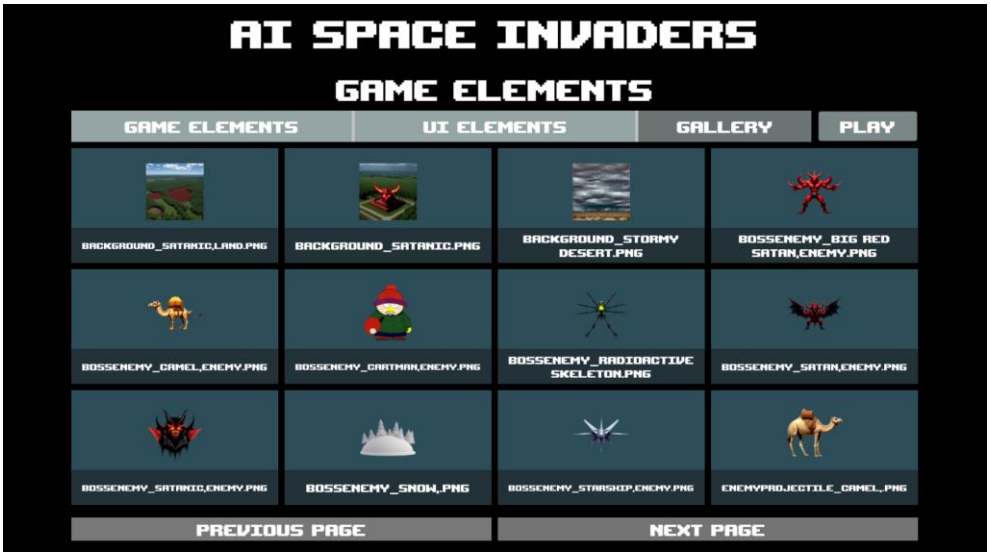
Ovime je upotpunjen cjelokupni proces konfiguracije mape i naziva, podešavanja potrebnih parametara ovisno o temi elementa, pristupa SD API-ju, prikaza učitavanja i tranzicijskih slika te primjenjivanja finalne slike. Rezultat je element igre u potpunosti dobiven korisničkim *promptom* za pripadajuću temu elementa (Sl. 4.11).



Slika 4.10: Finalna slika generirana na temelju korisničkog *prompta „devil bat“*

4.8. Galerija prethodno generiranih elemenata

Za bolje iskustvo igranja i mogućnost korištenja prethodno generiranih elemenata implementirana je i galerija kao treći izbornik. Ulaskom u izbornik korisnik vidi prikaz svih elemenata spremljenih u već spomenutu `Application.streamingAssets` putanju (Sl. 4.11).



Slika 4.11: Prikaz galerije prethodno generiranih elemenata

Svaki element kojeg SD generira moguće je zamijeniti s proizvoljnim prethodno generiranim elementom iz galerije. Ovo je jednostavno implementirano na način da kada korisnik pritisne da želi za pojedini element odabrati sliku iz galerije (gumb je vidljiv na Sl. 4.11) u `AIManager` sprema se odabrana `PromptResult` klasa u `EditingPromptResult` varijablu kako bi joj se moglo pristupiti iz druge skripte.

Za logiku odabira slike iz galerije pobrinut će se skripta `GalleryResult` koja je prikačena na svaki element u galeriji. Ako je u `AIManager` zabilježen neki `EditingPromptResult`, prilikom pritiska na neki `GalleryResult` metoda `SwitchPromptImage` pobrinut će se da se zamjena elementa izvrši i da se izađe iz načina zamjene elementa (Kod 4.24).

```
public void SwitchPromptImage()
{
    if (!AIManager.Instance.EditingPromptResult)
    {
        return;
    }

    var editingPromptResult =
AIManager.Instance.EditingPromptResult;
    var imageToChange =
editingPromptResult.imageGameObject.gameObject.GetComponent<I
mage>().sprite;
```

```
            editingPromptResult.imageGameObject.gameObject.GetComponent<I
            mage>().sprite = image.sprite;

            AIManager.Instance.PromptResults[editingPromptResult.theme] =
                        new() { Theme = editingPromptResult.theme, Text =
            editingPromptResult.text, Result = imageToChange };

                    tabGroup.ToGameElementsTab();
                    editingPromptResult.DisableEditMode();
                }
```

Kod 4.24: `SwitchPromptImage` metoda za zamjenu pojedinog elementa elementom iz galerije prethodno generiranih elemenata

4.9. Pokretanje igre i optimizacija elemenata

Nakon što je korisnik zadovoljan sa svim odabranim elementima spreman je ući u igru.

Pritiskom na *Play* gumb ulazi se u Prompt Invaders s personaliziranim elementima igre i sučelja (Sl. 4.15).

Slika 4.12: Primjer procesa igranja (engl. *gameplay*) proizvoljnog scenarija Prompt Invadersa

Pri učitavanju scene dolazi do primjene svih generiranih slika na odgovarajuće elemente. Za to je odgovorna skripta `PromptThemedObject` koja je prikačena za svaki element koji se izmjenjuje. Skripta pri pokretanju pokreće metodu `CreateSpriteFromPromptResult` koja dohvaća iz globalne `AIManager` skripte rezultat s obzirom na temu elementa. Nakon što dohvati teksturu, pomoćnom metodom

`GetSpriteSize` dohvaća veličinu 2D objekta (engl. *sprite*) kojeg će kreirati na temelju dohvaćene teksture te stvara generirani *sprite* kao *Image* komponentu elementa (Kod 4.25).

```
        private Sprite CreateSpriteFromPromptResult()
        {
            var spriteTexture =
PromptHelper.GetPromptResult(promptTheme).texture;
            var (width, height) = GetSpriteSize();
            var spriteRect = new Rect(0.0f, 0.0f, width,
height);

            return Sprite.Create(spriteTexture, spriteRect,
new Vector2 { x = 0.5f, y = 0.5f });
        }

        private (int width, int height) GetSpriteSize() =>
            promptTheme switch
            {
                PromptTheme.Background =>
(Constants.GeneratedBackgroundWidth,
Constants.GeneratedBackgroundHeight),
                PromptTheme.UIButton =>
(Constants.GeneratedUiButtonWidth,
Constants.GeneratedUiButtonHeight),
                PromptTheme.PlayerProjectile or
                    PromptTheme.Enemy or
                    PromptTheme.Player or
                    PromptTheme.BossEnemy or
                    PromptTheme.EnemyProjectile or
                    PromptTheme.UIBackground =>
(Constants.GeneratedSpriteWidth,
Constants.GeneratedSpriteHeight),
                _ => (Constants.GeneratedSpriteWidth,
Constants.GeneratedSpriteHeight)
            };
```

Kod 4.25: `CreateSpriteFromPromptResult i GetSpriteSize` metode za postavljanje *Image* komponente svakog `PromptThemedObject` tipa

Potrebno je i modificirati veličine samih objekata u igri tako da npr. *player* i *enemy* nisu iste veličine, da je *boss enemy* malo veći od ostalih *enemyja*, itd.

Također, s obzirom na to da se mijenja *Image* komponenta u novu, moramo reinicijalizirati sudarne točke objekta (engl. *collider*). Ovo ćemo napraviti `ReinitializeCollider` metodom koja će trenutni `PolygonCollider2D` uništiti te iznova kreirati kako bi se *collider* izračunao iznova i postavio na ispravne sudarne točke objekta (Kod 4.26).

```
        private void ReinitializeCollider()
        {
            var currentPolygonCollider =
GetComponent<PolygonCollider2D>();
            if (currentPolygonCollider == null)
            {
                return;
            }

            Destroy(currentPolygonCollider);

            var newPolygonCollider =
gameObject.AddComponent<PolygonCollider2D>();
            newPolygonCollider.isTrigger = true;
        }
```

Kod 4.26: `ReinitializeCollider` metoda za ponovni izračun sudarnih točki svih objekata `PromptThemedObject` tipa

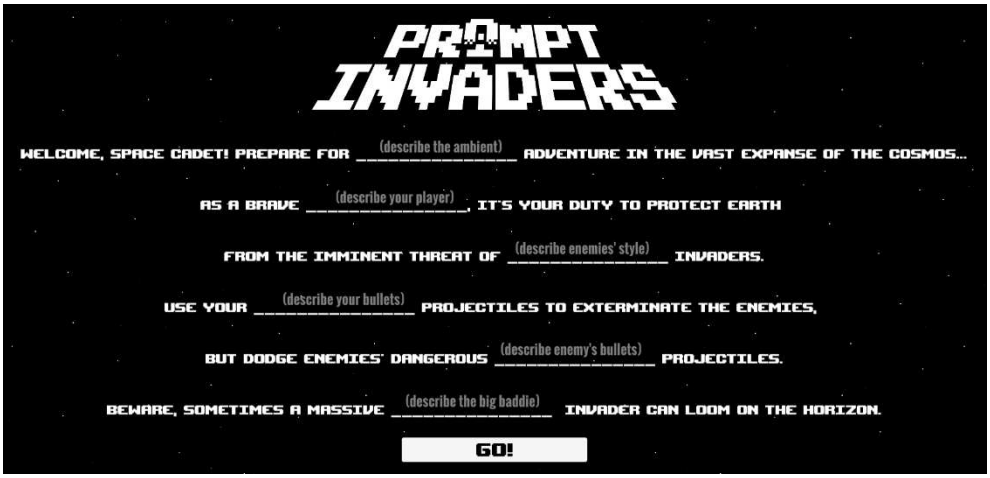
4.10. Rezultat – Prompt Invaders

Rezultat svih dosad opisanih postupaka je potpuno personalizirana videoigra Prompt Invaders inspirirana klasičnom videoigrom Space Invaders koja iskorištava umjetnu inteligenciju i njenu sposobnost *text-to-image* generacije za generiranje tekstura iz proizvoljnog unosa igrača.

Ulaskom u videoigru korisniku je predstavljen tzv. tematski izbornik s pričom koju je potrebno nadopuniti i na taj način opisati i formirati sljedeće:

- grafičke elemente sučelja (poput izgleda izbornika prilikom igranja),
- izgled glavnog igrača (engl. *player*) čija je uloga pucati u neprijatelje,
- izgled neprijateljskih igrača (engl. *enemies*) koji u grupi lete područjem,

- izgled projektila koje ispaljuje *player*,
- izgled projektila koje ispaljuju neprijatelji,
- izgled većeg i opasnijeg neprijatelja koji periodično uletava u igru i
- pozadinu videoigre generiranu pomoću opisanog ambijenta (Sl. 3.1).



Slika 4.13: Tematski izbornik prikazan prilikom pokretanja Prompt Invaders videoigre

Popunjavanjem svih praznina i klikom na gumb za nastavak korisnik definira temu videoigre, čime se pokreće proces generacije slike na temelju teksta (*text-to-image* generacija). Ovaj proces uključuje slanje svih korisničkih unosa zajedno s dodatnim ključnim riječima (koje korisnik nije izričito naveo) u obliku proširenog upita (engl. *prompt*) u alat za generiranje slika. Alat zatim vraća generirane teksture koje se koriste u videoigri. Korisnik može pratiti napredak procesa generiranja tekstura na idućem izborniku, a to je izbornik generiranih elemenata (Sl. 3.2).

Slika 4.14: Izbornik generiranih elemenata

Na ovom izborniku korisniku se prikazuju pojedini generirani elementi, inicijalni korisnikov upit (nadopunjeni dijelovi tematskog izbornika) te kategorija generiranog elementa (npr. pozadina, neprijatelj itd.). Ako generirani rezultat nije prihvatljiv, za svaki je moguće:

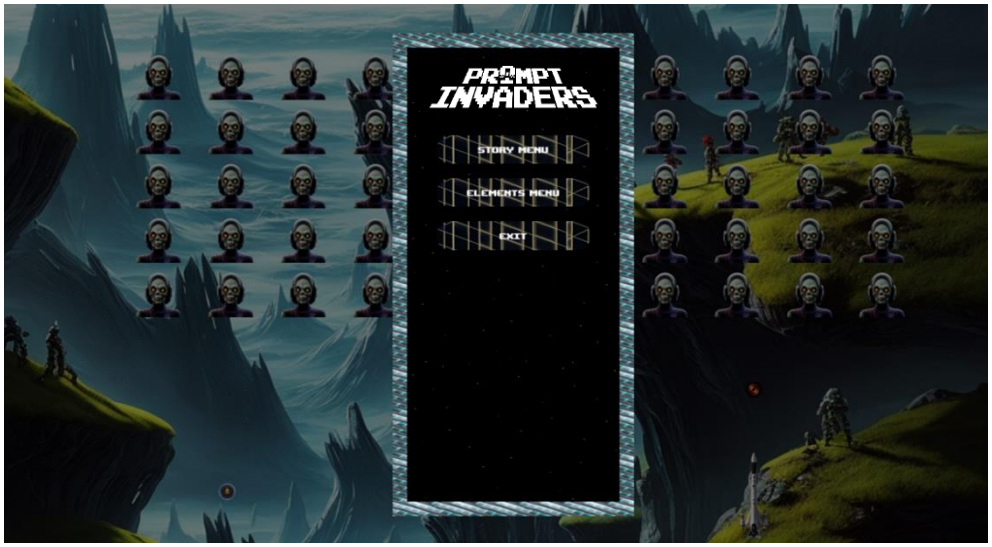
- iznova generirati rezultat (engl. *reprompt*) ili
- uzeti novi rezultat iz galerije prethodno generiranih elemenata.

Kada je korisnik zadovoljan sa svim rezultatima i spreman je zaigrati igru, klikom na gumb za pokretanje igre odmah je prebačen u Prompt Invaders igru potpuno prilagođenu korisnikovim željama (Sl. 3.3).

Slika 4.15: Prikaz primjera *gameplaya* Prompt Invadersa s generiranim personaliziranim elementima

Procesom prijelaza u sam *gameplay* igre korisnik zaigrava videoigru s poznatim mehanikama, ali vizualnim ruhom koje je on sam odredio prijašnjim upitima. Te poznate mehanike uključuju pomicanje lijevo i desno do rubova ekrana, ispaljivanje projektila, izbjegavanje neprijateljskih projektila i gađanje neprijatelja.

Korisnik u bilo kojem trenutku može zaustaviti igru nakon čega mu se prikazuje personalizirani izbornik u igri uz opcije za povratak u izbornik generiranih elemenata, povratak skroz na tematski izbornik ili potpuni izlazak iz videoigre (Sl. 3.4).



Slika 4.16: Personalizirani izbornik prikazan prilikom pauziranja igre

5. Rezultat

Svrha ovog rada bila je implementirati klasičnu videoigru Space Invaders s personaliziranim elementima koji se dinamički generiraju na temelju korisničkog unosa uz pomoć umjetne inteligencije. Specifično, za *text-to-image* generaciju koristio se Stable Diffusion i SD WebUI Forge API za generiranje vizualnih elemenata u videoigri. Jedan od ključnih izazova bio je optimizirati generaciju, što je zahtijevalo instalaciju proširenja za micanje pozadine iz generiranih slika, pristupanje različitom *endpointu* za postotak generacije, proširivanje upita, itd.

Zanimljiva je razina prilagodbe potrebna za uspješnu integraciju ovakvih tehnika u razvoj igre, posebice jer su problemi koji se moraju razriješiti relativno drugačiji od onih na koje smo naviknuli prilikom klasičnog razvoja videoigri. Ovim radom nastojalo se objasniti ključne aspekte i tehnike potrebne za uspješnu implementaciju ovakve dinamične videoigre i time pružiti smjernice za buduće projekte slične prirode.

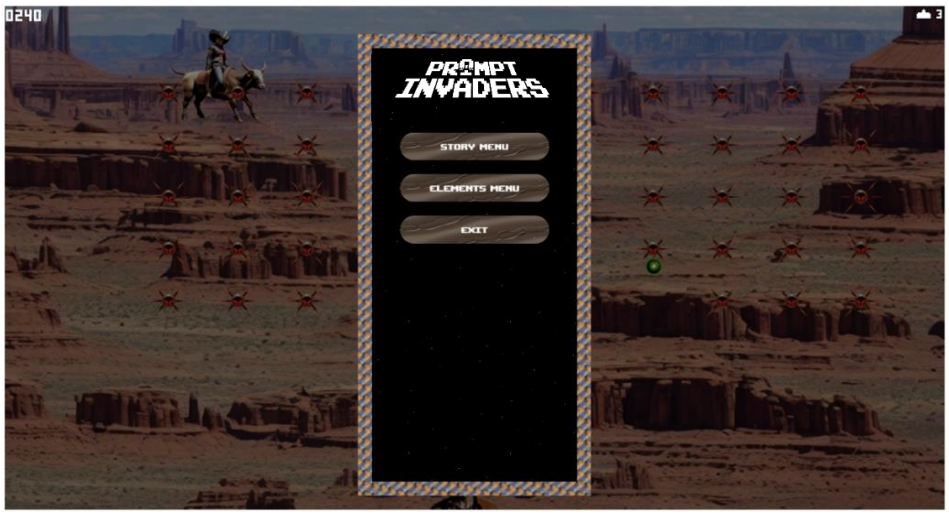
Razvoj igre trajao je nekoliko mjeseci, tijekom kojih je bilo potrebno iterativno prilagođavati i testirati različite elemente igre. Generacija vizualnih elemenata zahtijevala je poseban pristup, budući da je svaki ciklus generacije trajao između nekoliko sekundi do nekoliko minuta, ovisno o složenosti upita. Optimizacija je bila ključna za osiguranje glatkog i efikasnog procesa generacije unutar prihvatljivog vremenskog okvira, stoga se (osim za pozadinu) nije pretjerivalo s kvalitetom elemenata u igri jer se ionako radi o manjim objektima. Također, bilo je zahtjevno uključiti Layer Diffusion proširenje prilikom pristupanja API-ju jer iz specifikacije API-ja nije bilo jasno kakva se struktura očekuje za uključivanje, što je zahtijevalo detaljnije izučavanje koda API-ja i internih procedura.

Rezultat je videoigra koja koristi generativne modele za stvaranje vizualnih elemenata, čime se postiže jedinstven izgled i osjećaj igre. Implementacija galerije prethodno generiranih elemenata dodatno omogućuje igračima da rekreiraju prethodna iskustva igranja zamjenjujući proizvoljne elemente onima prethodno generiranim. Tehnike opisane u radu omogućuju repliciranje sličnih procesa u drugim projektima, s naglaskom na skalabilnost i optimizaciju performansi.

Primjer jedne personalizirane Prompt Invaders igre (uz prikaz korisničkih *promptova* u izborniku) vidljiv je na slikama 5.1, 5.2 i 5.3.

Slika 5.1: Primjer tematskog izbornika s proizvoljnim korisničkim *promptovima*

Slika 5.2: Primjer *gameplaya* igre s temom divljeg zapada



Slika 5.3: Primjer izbornika za pauzu tokom igranja igre

Zaključak

Ovim radom istraženi su koncepti umjetne inteligencije u kontekstu razvoja videoigara te je uspješno implementirana metoda generiranja vizualnih elemenata 2D videoigre korištenjem Stable Diffusion modela. Proces razvoja igre bio je usmjeren na integraciju naprednih tehnika umjetne inteligencije u razvojni okvir Unity, s naglaskom na generiranje sadržaja u stvarnom vremenu.

Za postizanje rezultata bilo je potrebno detaljno razumijevanje osnovnih pojmova strojnog učenja i dubokog učenja, s fokusom na generativne modele kao što su generativne kontradiktorne mreže (GAN) i varijacijski autokoderi (VAE).

Uz teorijsku podlogu, definirane su i praktične tehnike prilagodbe modela specifičnim potrebama igre, što uključuje optimizaciju performansi prilikom renderiranja slika i integraciju generiranog sadržaja u igru. Ovaj rad je nastojao pružiti jasno objašnjenje postupaka i tehnika, omogućujući time repliciranje procesa u sličnim projektima.

Izrada igre detaljno je dokumentirana kako bi replikacija bila što jednostavnija, a primarni cilj igre bio je da se mehanike igranja integriraju s generativnim modelima umjetne inteligencije i pruže potpuno personaliziranu videoigru različitu prilikom svakog igranja.

Literatura

[1] Stuart Russell, Peter Norvig (1995). Artificial Intelligence: A Modern Approach

[2] Nepoznato. (n.d.). Take-Two CEO Reveals How AI Can Make GTA 6 Even Better, s interneta https://www.inverse.com/gaming/gta-6-generative-ai-take-two-ceo

[3] Nepoznato. (n.d.) Računalni vid (uvodno predavanje), s interneta https://www.zemris.fer.hr/~ssegvic/vision/cv_intro.pdf

[4] B. Dalbelo Bašić, M. Čupić, J. Šnajder (May 2008). *Umjetne neuronske mreže*, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb

[5] L. Pleše (May 2019). *Duboko učenje*, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb

[6] K. Špehar (September 2022). *Umjetna inteligencija u videoigrama*, Fakultet informatike, Sveučilište Jurja Dobrile, Pula

[7] L. Blažević (July 2024). *Podržano učenje I Q-učenje*, Sveučilište Josipa Jurja Strossmayera, Odjel za matematiku, Osijek

[8] K. Rücker (September 2019). *Pretvaranje teksta u sliku u proširenoj stvarnosti*, Sveučilište Josipa Jurja Strossmayera, Fakultet elektrotehnike, računarstva i informacijskih tehnologija, Osijek

[9] Nepoznato (n.d.) Što je Generative Adversarial Network (GAN)?, s interneta https://www.unite.ai/hr/what-is-a-generative-adversarial-network-gan

[10] Nepoznato (n.d.) Što su CNN (konvolucijske neuronske mreže)? https://www.unite.ai/hr/what-are-convolutional-neural-networks

[11] K. Džomba (July 2018). *Konvolucijske neuronske mreže*, Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet, Zagreb

[12] M. Obrvan (July 2023). *Umjetna inteligencija pri stvaranju vizualne umjetnosti*, Sveučilište u Zagrebu, Filozofski fakultet, Zagreb

[13] Nepoznato (n.d.) Što je autoenkoder? https://www.unite.ai/hr/%C5%A1to-je-autoenkoder

[14] Nepoznato (n.d.) The Illustrated Stable Diffusion https://jalammar.github.io/illustrated-stable-diffusion/

[15] Nepoznato (n.d.) Stable Diffusion 1 vs 2 - What you need to know https://www.assemblyai.com/blog/stable-diffusion-1-vs-2-what-you-need-to-know/

[16] Nepoznato (n.d) GitHub - lllyasviel/sd-forge-layerdiffuse https://github.com/lllyasviel/sd-forge-layerdiffuse

[17] Nepoznato (n.d) GitHub - lllyasviel/stable-diffusion-webui-forge https://github.com/lllyasviel/stable-diffusion-webui-forge

[18] Nepoznato (n.d) Unity vs Unreal Engine: What Works For You?
https://www.incredibuild.com/blog/unity-vs-unreal-what-kind-of-game-dev-are-you

[19] Nepoznato (n.d) ChatGPT
https://chatgpt.com/

Sažetak

Rad istražuje inovativne pristupe integracije umjetne inteligencije u razvoju 2D igara gađanja, s posebnim fokusom na konverziju teksta u sliku za izmjenu izgleda klasične videoigre "Space Invaders". Analizom nekoliko ključnih algoritama umjetne inteligencije ideja je da se stekne razumijevanje za procese koji se u pozadini prilikom generiranja vizualnih elemenata videoigre prema korisničkim zahtjevima. U pokaznoj videoigri korisnicima se omogućuje prilagodba grafičkog sučelja putem tekstualnih unosa čime se znatno povećava dinamičnost, a time i cjelokupno iskustvo igranja.

Ključne riječi: umjetna inteligencija, strojno učenje, duboko učenje, videoigre, unity, stable diffusion, space invaders

Summary

This paper explores innovative approaches to the integration of artificial intelligence in the development of 2D games, with a special focus on text-to-image conversion for modifying the look of the classic svideo game "Space Invaders". By analyzing several key artificial intelligence algorithms, the idea is to gain an understanding of the processes behind the generation of visual elements in a video game according to user input. In the sample video game, users are allowed to customize the graphic interface through text entries, which significantly increases immersiveness, and thus the overall gaming experience.

Keywords: artificial intelligence, ai, machine learning, deep learning, videogames, unity, stable diffusion, space invaders