Využití generativní umělé inteligence při vývoji her

Bc. Lukáš Gazdík

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Lukáš Gazdík**
Osobní číslo:	**A22713**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Využití generativní umělé inteligence při vývoji her**
Téma práce anglicky:	**Application of Generative AI in Game Development**

Zásady pro vypracování

1. Analyzujte využití klasických modelů umělé inteligence využívaných ve hrách.
2. Vyberte a popište funkčnosti vhodných nástrojů AI pro projekt.
3. Navrhněte vhodný formát pro demonstraci vybraných prvků vývoje pomocí AI.
4. Implementujte algoritmy/generativní nástroje AI.
5. Vytvořte prototyp hry ve vybraném engine.
6. Porovnejte tento přístupu s klasickým vývojem (plusy/minusy určení potenciálu v budoucnosti).

Forma zpracování diplomové práce: **tištěná/elektronická**
Jazyk zpracování: **Slovenština**

Seznam doporučené literatury:

1. BOURG, David a SEEMANN, Glenn. *AI for Game Developers: Creating Intelligent Behavior in Games*. O'Reilly Media, 2004. ISBN 978-1491900109.
2. BUONTEMPO, Frances. *Genetic algorithms and machine learning for programmers: Create AI Models and Evolve Solutions*. Pragmatic Bookshelf, 2019. ISBN 978-1680506204.
3. FOSTER, David. *Generative Deep learning: Teaching Machines to Paint, Write, Compose, and Play*. O'Reilly Media, 2019. ISBN 978-1492041894.
4. MILLINGTON, Ian. *AI for Games, Third Edition*. 3rd ed. CRC Press, 2019. ISBN 978-1351053297.
5. NYSTROM, Robert. *Game Programming Patterns*. 2014. ISBN 978-0990582908.
6. SCHELL, Jesse. *The Art of Game Design: A Book of Lenses, Third Edition*. 3rd ed. A K Peters/CRC Press, 2014. ISBN 978--1138632059.
7. TUNSTALL, Lewis; WERRA, Leandro von a WOLF, Thomas. *Natural Language Processing with Transformers, Revised Edition*. O'Reilly Media, 2022. ISBN 978-1098136796.

Vedoucí diplomové práce: **Ing. Bc. Pavel Vařacha, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne .....

podpis studenta

ABSTRAKT

Cieľom tejto práce je otestovať integráciu, použiteľnosť a efektivitu generatívnej umelej inteligencie v oblasti vývoja hier. Tento typ AI je všeobecne známy svojou kreativitou a využiteľnosťou v najrôznejších oblastiach. Projekt predstavujúci prototyp hry, implementuje lokálne spustiteľné AI modely vo forme veľkých jazykových modelov na generovanie textu a text to image generátor na tvorbu obrázkov. V závislosti od potrieb hry je možné napríklad dynamicky vytvárať nové textúry pre herné postavy alebo zadávať im príkazy generované chatbotom. Takýto prístup ku vývoju však nie je priamočiary a integrácia takýchto programov má rôzne limitácie. Práca preto tiež rozoberá rôzne výzvy, ktoré z použitia AI plynú, ako napríklad vysoké nároky na výkon, nepredvídateľný povaha generovaného obsahu, tvorba v rámci kapacít daného AI modelu, alebo komunikácia medzi viacerými AI modelmi.

Klíčová slova: Generatívna AI, Vývoj hier, Herná AI, Text generation, Text-to-image generation

ABSTRACT

The aim of this thesis is to test the integration, usability and effectiveness of generative AI in the field of game development. This type of AI is widely known for its creativity and applicability in a wide variety of fields. The project, representing a game prototype, implements locally runnable AI models in the form of large language models for text generation and a text to image generator for image creation. Depending on the needs of the game, it is possible, for example, to dynamically create new textures for the game characters or give them chatbot-generated commands. However, such an approach to development is not straightforward and the integration of such programs has various limitations. Therefore, the thesis also discusses various challenges that arise from the use of AI, such as the high performance requirements, the unpredictable nature of the generated content, creation within the capabilities of a given AI model, or communication between multiple AI models.

Keywords: Generative AI, Game development, Game AI, Text generation, Text-to-image generation

Chcem sa na tomto mieste poďakovať vedúcemu práce pánovi Ing. Pavlovi Vařachovi, PhD. za pomoc s návrhom, ochotu, inšpiráciu a aj za spätnú väzbu pri vypracovaní tejto práce, a taktiež rodine a kolegom za podporu.

OBSAH

ÚVOD

Tvorba obsahu v hrách ako písanie dialógov, prispôsobovanie parametrov hry, dizajn a kreslenie textúr, modelovanie 3D objektov, alebo dizajn herného prostredia je prevažne manuálny a pomerne zdĺhavý proces. Na zjednodušenie tohto procesu sa už bežne využíva napríklad procedurálne generovanie alebo strojové učenie, ktoré prináša veľmi zaujímavé výsledky aj výšenie efektivity vývoja. Avšak ďalším krokom pri vývoj a tvorbe herného obsahu v podobe textov, textúr, hudby, alebo dokonca 3D modelov môže byť použitie generatívnej AI. Táto práca má za úlohu preskúmať možnosti takéhoto prístupu ku vývoju hier.

Teoretická časť práce sa zameriava nie len na analýzu a využitie klasických modelov a AI algoritmov používaných v hrách, ale aj na popis modernej generatívnej umelej inteligencie. Na základe tohto prieskumu sú vybrané také modely a programy, ktoré sú najviac flexibilné a prispôsobiteľné konkrétnym požiadavkám hry. Súčasťou práce je tiež popis ich funkcionalít a použitia, spolu s pomocnými programami, ktoré umožňujú spúšťanie a ladenie týchto modelov. Predovšetkým sa jedná o lokálne jazykové modely a obrázkové generátory, ako je Mistral alebo Stable Diffusion, ktoré môže užívateľ alebo vývojár používať na svojom vlastnom počítači.

Praktickú časť sa venuje návrhu, tvorbe a testovaniu prototypu hry, ktorý používa vybrané generatívne modely na generovanie rôzneho herného obsahu v reálnom čase. Hra bude vytváraná v hernom engine Unity pomocu programovacieho jazyku C#. Súčasťou implementácie hry je návrh vhodného herného dizajnu tak, aby využitie týchto modelov malo v hre zmysel, popis hernej logiky a skriptov, ktoré spracovávajú komunikáciu s AI modelmi, či príprava a tvorba herných assetov pre tento projekt. Práca pojednáva aj o testovaní rôznych herných nastavení a hľadaniu vhodných parametrov pre prispôsobenie AI modelov.

Záverečná časť práce sa zameriava na analýzu rôznych výhod a nevýhod vývoja spojeného s použitím generatívnej umelej inteligencie. Jedná sa prevažne o posúdenie výstupov a poznatkov z testovania a hrania vyvíjanej hry. Zahrnutý je aj popis možných optimalizácií a vylepšení pre použitie tejto technológie v budúcne.

I. TEORETICKÁ ČÁST

1 TRADIČNÁ AI

Umelá inteligencia v hrách sa tradične využíva na ovládanie nehráčskych postáv (NPC) spoluhráčov alebo protivníkov, ktorí imitujú inteligenciu tým, že sú naprogramovaní ako konečný automat s konečným počtom stavov. Tým dosahujú zdanlivo autonómne správanie a možnosť rozhodovať na základe snímania svojho okolia v rámci vymedzenej oblasti danej hry. V hrách sa neustále rozvíjajú aj generatívne technológie a prostredníctvom procedurálneho generovania je v hrách možné automaticky generovať rôzny obsah, ako sú textúry, animácie, grafické prvky, herné scenáre, hádanky alebo celé environmentálne systémov, ako sú mestá, ostrovy, mapy alebo dokonca planéty. Pri vstupe do éry Deep learningu sa objavujú rôzne pokusy o začlenenie umelej inteligencie do takmer každého aspektu hier. Príkladom je AI AlphaGo, ktorý dokáže poraziť najlepších hráčov v hre GO, interakcie medzi NPC a hráčmi posilnené umelou inteligenciou a tiež aplikácia navrhovaní úrovní a tvorba príbehových nápadov. [1]

1.1 Typy hernej AI

Pojem umelá inteligencia v hrach ako ju poznáme, je mierne zavádzajúci. Je totiž rozdiel medzi umelou inteligenciou a umelým správaním, ktoré napodobňuje inteligenciu. V hrách nie je úplne cieľ aby agenti, nepriatelia, alebo iné NPC prekabátili hráčov, tým že budú inteligentnejší ako on. Cieľom je aby boli títo agenti tak inteligentní, ako je potrebné, aby hra bola zábavná a poskytovala dostatočnú angažovanosť. V iných odvetviach je snahou posúvať limity a efektívnosť AI čo najďalej, zatiaľ čo v hrách by tento prístup nespĺňal účel. Počítačom riadený súper musí naopak byť do istej miery nedokonalý a napodobňovať ľudské správanie. [2]

1.2 Deterministická vs nedeterministická AI

Z tohto pohľadu sa AI v hrách dá rozdeliť na 2 kategórie. Deterministické správanie je špecifické v tom, že je predvídateľné. Herný agent alebo mechanika implementovaná takýmto spôsobom neobsahuje žiadny prvok neistoty. Medzi tieto typy AI patria základné napríklad algoritmy na vyhľadávanie cesty alebo prenasledovanie. Takéto algoritmy sú jednoduchšie na implementáciu a sú postačujúce vo väčšine prípadov kedy nie je potrebné realistické správanie herných postáv alebo mechaník. Takýto prístup však bude produkovať väčšinou rovnaký výstup, čo niekedy nemusí byť postačujúce pre danú hru. Naopak správanie ktoré je nedeterministické implementuje nejaký mechanizmus, ktorý do hry vnáša neistotu alebo náhodnosť, a je tým pádom viac nepredvídateľná. Miera neistoty potom závisí od metódy od konkrétnej AI metódy aj jej implementácie. Takéto správanie sa najčastejšie dá dosiahnuť nejakou formou učenia, kedy herný objekt / agent dokáže prispôsobovať svoje rozhodovanie na základe ostatných objektov, stavu

hry, alebo rozhodovania hráča. Taktiež je možné implementovať učenie, ktoré dovoľuje objektom zapamätať si svoje predchádzajúce akcie a rozhodovať sa na základe toho aký spravia ďalší ťah. Takéto učenie môže byť realizované napríklad pomocou neurónových sietí, evolučných, genetických, rojových algoritmov alebo fuzzy logiky. [2] [3] [4]

2 NÁVRHOVÉ VZORY V HRÁCH

2.1 Game loop

Herná slučka je jedným z najzákladnejších návrhových vzorov v hernom programovaní. Tento mechanizmus sa používa, aby sa hra mohla v pravidelných intervaloch opakovať a priebeh hry sa tak mohol vyvíjať smerom vpred. Každá iterácia tejto slučky posúva stav hry o určitú hodnotu dopredu. Rýchlosť akou sa tieto iterácie menia sa udáva najčastejšie v snímkach za sekundu (FPS). Čím rýchlejšie hra jednu snímku spracuje tým vyšší je počet FPS a tým je hra plynulejšia. Počas každej iterácie tejto slučky musí herný engine spracovať všetku nadefinovanú hernú logiku. To zahŕňa spracovanie herného kódu a skriptov, spracovanie hernej fyziky, vykreslenie všetkých potrebných objektov a časticových efektov a ostatných výpočtov potrebných pre beh hry. To akou rýchlosť budú tieto výpočty spracované závisí od hardvéru konkrétneho zariadenia a hernej optimalizácie.

Kombinácia výkonných grafických kariet, procesorov a dedikovaného hardvéru všeobecne najviac ovplyvňujú rýchlosť spracovania. Pokiaľ má počítač zastaralý hardvér a hra implementuje príliš komplexné grafické spracovanie, spracovanie jednej snímky môže trvať značný čas. Taktiež zle optimalizovaný herný kód bude nevyhnutne zaťažovať procesor na dlhšiu dobu, čím zase klesá efektivita. Spracovanie kódu paralelne na viacerých vláknach môže v tomto prípade pomôcť spracovať väčšie množstvo kódu za rovnaký čas. Takéto asynchronné spracovanie je dôležité pri spracovaní sieťových udalostí, keď napríklad hra čaká na odpoveď od serveru alebo pri spracovaní zložitých výpočtov, ktorých spočítanie nie je možné počas jednej snímky. Existuje aj možnosť použiť takzvané korutiny, ktoré umožnia určitú časť kódu rozdelená na viacero snímok a spracovávať ho rychlosťou nezávislou od iterácií hlavnej slučky. Obe tieto techniky je nutné použiť pre zachytenie a spracovanie signálov od hráča ako je stláčanie kláves, čím sa spracovanie presmeruje na vedľajšiu koľaj a tým sa zaistí že hlavná slučka nie je blokovaná. [5] [6]

2.2 Observer

Observer pattern je návrhový vzor, ktorý definuje spôsob nepriamej komunikácie medzi dvomi alebo viacerými objektami, široko rozšírený v rôznych softvérových oblastiach. Ide o väzbu jeden objekt k viacerým, kde sa pracuje s dvomi typmi objektov, subjektom a pozorovateľmi. Subjekt si drží zoznam svojich pozorovateľov a nový pozorovatelia môžu byť pridávaní alebo odoberaní z tohto zoznamu bez nutnosti subjekt priamo editovať. Keď sa stav subjektu zmení podľa nejakého preddefinovaného správania, oznámi túto udalosť všetkým svojim pozorovateľom. Úlohou pozorovateľov je tieto udalosti zachytiť a na základe nich reagovať podľa definovaného správania.

Týmto spôsobom môžu herné objekty rôznych typov sledovať a reagovať na správanie iného objektu bez toho, aby museli byť tieto objekty byť na sebe závislé. Implementácia je tým pádom jednoduchšia pretože objekty nepotrebujú mať na seba referencie, a tak je implementácia vytvárania a mazania objektov jednoduchšia. Vo vývoji hier môže byť tento mechanizmus byť použitý napríklad pri reagovanie NPC postáv na stav hráča, zmenu správania AI, spracovanie postupu cez herné úrovne, kolízie alebo plnenie herných úloh a výziev, kde určitá akcia hráča môže jednoducho zmeniť stav aktívnej úlohy.

V prípade synchrónneho spracovania pri vytvorení eventu subjektom je potrebné aby všetci pozorovatelia reagujúci na túto udalosť plne vykonali svoju funkcie a až potom môže subjekt pokračovať v spracovávaní svojho kódu. Toto je možné obísť spracovávaním udalostí asynchrónne, kedy udalosť je vytvorená v jednom vlákne, a spracovaná inom. [5] [6] [7]

2.3 Object pool

V hrách je bežné, že mnohé objekty počas hrania vznikajú a zanikajú, či už ide tvorbu objektov v rámci prechodu úrovňami, vymazanie zničených objektov zo scény, alebo generovanie častíc v časticových systémoch. Vytváranie a mazanie nových objektov za behu hry je jednoduché na implementáciu, avšak v závislosti na type objektu a výpočetnej náročnosti jeho inicializácie, môže nárazová tvorba veľkého množstva nových objektov mať negatívny dopad na výpočetný výkon hry.

Object pool rieši tento problém spôsobom, že určitý počet objektov rovnakého typu je naalokovaný na začiatku hry a vložený do dátového kontajneru, v ktorom ostávajú po dobu celej hry. Ak je potrebné nejaký objekt nainicializovať, hra jednoducho vyberie voľný objekt z tohto poolu, prepne ho do aktívneho stavu, a keď objekt nebude ďalej potrebný, prepne jeho stav na neaktívny. Nevýhoda však je, že objekty musia byť držané v pamäti aplikácie aj ak sa práve nepoužívajú, preto je treba rozlišovať, pri ktorých objektoch sa tento mechanizmus oplatí využiť. [5] [6]

2.4 Singleton

Singleton predstavuje skôr návrhový anti-vzor, ktorý zabezpečuje že v programe môže existovať iba jedna inštancia danej triedy. Trieda môže byť opakovane vytvorená a zničená, ak však trieda už existuje, musí však byť zaistené aby sa program nedokázal vytvoriť ďalšiu takúto triedu, aby ostatné objekty mali statickú referenciu na tento objekt. Tým sa zaistí globálny prístup ku logike, ktorú Singleton trieda obsluhuje.

Najčastejšie je takýto mechanizmus potrebný pri manažérskych objektoch, ktoré napríklad pracujú so súborovým systémom, obsluhujú API sieťovej komunikácie, alebo

riadia správu herných zdrojov. Výhoda je v tom, že sa pomocou takého objektu dajú inicializovať a spravovať globálne premenné, ku ktorým môžu mať prístup všetky ostatné objekty v hre. Pri single-player sa týmto spôsobom dá napríklad implementovať hráčov profil alebo objekt, ktorý rozosiela príkazy AI agentom. [5] [7]

2.5 Prototype

Herný návrhový vzor Prototype je vzor, ktorý umožňuje efektívne vytvárať komplexné objekty bez nutnosti ich priamej inštancie. Namiesto toho sa klonuje existujúci prototyp, čím sa ušetrí čas a zdroje. Prototyp je väčšinou abstraktná trieda, ktorá definuje rozhranie pre klonovanie a ďalšie vlastnosti daného objektu. Klonovaním sa zaistí, že komplexné objekty stačí definovať iba raz, tvorba nových objektov je rýchlejšia, kód je rovnaký u všetkých klonov, pričom sa ale ich parametre môžu meniť dynamicky počas hry. Herné enginy ako Unity už majú funkcionalitu prototypov väčšinou implementovanú. [5] [6]

2.6 State pattern

Tento jednoduchý, ale mocný návrhový vzor je implementáciu konečného/stavového automatu. Je to model umožňujúci objektu meniť jeho správanie v závislosti od jeho vnútorného stavu. Základom tohoto modelu sú tri časti - fixná množina stavov, aktuálny stav a prechodné podmienky. Objekt, ktorý implementuje stavový vzor, môže mať napríklad definované stavy ako *obrana*, *útok*, **prenasledovanie** a *útek*. Objekt vykonáva činnosť (volá funkcie) podľa toho, v ktorom stave sa práve nachádza. Prechodovými podmienkami môže byť napríklad stav života entity, počet jej aktívnych spojencov, jej aktuálne skóre, alebo v prípade hráča aktuálne stlačené klávesy. Ak je nejaká z týchto podmienok splnená, stav sa zmení, a to sa odrazí na ďalšom správaní hernej entity.

Vo vývoji hier je tento model možné použiť na modelovanie rôznych komplexných operácií, ako je zmena správania herných entít v závislosti od ich aktuálneho stavu, controllery AI agentov a NPC postáv alebo riadenie animácií. Každý objekt môže mať svoj vlastný vnútorný stavový automat, definujúci jeho jedinečné správanie. Prípadne môže byť tento model súčasťou Prototypu, a jednotlivé klony si ho môžu dodatočne upravovať. Takto sa dá rozdeliť komplexná logika hernej entity do menších a prehľadnejších častí. Tento model je väčšinou viac robustný, pretože je možné rozširovať logiku pridávaním a odoberaním stavov, bez toho, aby sme tím rozbili ostatné časti hry. Problém môže nastať v prípade, keby chceme do možných stavov zaznamenať všetky kombinácie akcií, ktoré môže postava vykonávať. Tým sa môže ohromne zvýšiť počet potrebných stavov a kód bude znovu jeden veľký chaos. Pomerne jednoduchým rie-

šením tohoto problému môže byť pridanie ďalších automatov pre daný objekt, ktoré môžu spracovávať vlastné stavy. [5] [7]

2.7 Component

V hrách budú vždy existovať entity, ktoré na svoj beh potrebujú používať rôzne mechanizmy herného enginu. Ide väčšinou o spracovanie vykresľovania, hernej fyziky, kolízií, zvukov, AI alebo navigácie. Každý typ objektu v hre potrebuje s týmito doménami pracovať podľa svojich parametrov, ak majú však všetky tieto objekty jednotlivé domény implementovať ako súčasť svojho kódu, skončilo by to hromadou neprehľadného redundantného kódu a zložitými závislosťami medzi jednotlivými modulmi. Preto sa tento problém rieši delením zdielanej logiky na komponenty.

Každý kľúčový mechanizmus hry, ako napríklad logika spracúvajúca pozíciu a rotáciu daného objektu, je izolovaný a implementovaný ako generický komponent, v tomto prípade komponent *Transform*. To z komponentov robí znovu použiteľné balíky kódu, ktoré môžu všetky herné objekty potom vlastniť ako svoje inštancie. Tak zaistíme že komponenty v danom objekte nebudú na sebe závislé, ale budú komunikovať len s nevyhnutnou časťou kódu. Do komponentu je môžeme zabaliť v podstate hoci akú časť kódu, ktorá je príliš rozsiahla na implementáciu, alebo ktorú chceme mať spoločnú v rôznych typoch objektov. [5] [7]

3 VYHĽADÁVANIE CESTY A NAVIGÁCIA

Existuje množstvo spôsobov akým sa postavy v hernom svete pohybujú. V niektorých hrách úplne stačí, keď sa postava pohybuje sprava doľava, alebo po priamke medzi dvomi bodmi. Takýto pohybom môže postava napríklad strážiť alebo prehľadávať nejakú oblasť, alebo len náhodne blúdiť. Tieto pevné pravidlá sú primitívne na implementáciu, ale nie sú skoro vôbec škálovateľné, pretože aj malé prekážky na ceste môžu túto AI zmiasť tak, že postava nebude vedieť čo robiť.

Čím viac je potrebné, aby hra bola realistickejšia, tým robustnejší algoritmus na určenie cesty bude potrebný. Postavy väčšinou nevedia dopredu, ktorou cestou sa budú musieť vydať, alebo či dokonca existuje cesta k ich ďalšiemu cieľu. Cieľ cesty sa navyše v priebehu hry môže dynamicky meniť, napríklad v závislosti od akcií hráča. Pre postavu je preto potrebné, aby jej umelá inteligencia dokázala vypočítavať po akej trase sa má pohybovať cez hernú úroveň aby sa od jej aktuálnej pozície dostala k cieľu.

Tento problém, ktorý sa nachádza na rozhraní rozhodovania a plánovania trasy, rieši rieši skupina algoritmov zvaných Path Finding. Väčšinou je používaný samostatne, ako modul na výpočet trasy ku cieľu, ktorý určí iná časť umelej inteligencie a volá sa iba vtedy, keď je potrebné plánovať novú trasu. Tieto algoritmy však nebudú môcť fungovať pri použití v surovom hernom leveli ani na obyčajnej hernej mape. Ako každý iný model umelej inteligencie, potrebujú zjednodušenú číselnú reprezentáciu herného priestoru, v ktorom môžu efektívne počítať. V tejto časti sa pozrieme, na niektoré z implementácií Path Finding algoritmov. [3] [8]

3.1 Dijkstrov algoritmus

Dijkstrov algoritmus je jeden zo základných algoritmov používaných na vyhľadávanie cesty. Pôvodne však bol navrhnutý ako optimalizačný algoritmus pre hľadanie najkratšej cesty v grafe. Jeho fungovanie sa dá zhrnúť, ako prehľadávanie všetkých uzlov grafu, pričom sa neustále rozširuje preskúmavaná oblasť a aktualizujú sa hodnoty najkratších ciest k jednotlivým uzlom. V prípade hry môžu uzly grafu predstavovať polohy orientačných bodov v hernom svete, a hrany môžu reprezentovať možné pohyby medzi týmito polohami.

Algoritmus iteratíve prehľadáva uzly v grafe smerom od počiatočného uzlu. Pri tom si v pamäti drží zoznam všetkých uzlov, ktoré už spracoval a zoznam uzlov, ktoré bude spracovávať v aktuálnej iterácií. Zároveň aktualizuje ceny jednotlivých ciest (lacnejšia cesta je kratšia). Vďaka spôsobu, akým je proces šírenia implementovaný, sa zaručuje, že prvá nájdená platná cesta do cieľa, bude zároveň najlacnejšia (podľa váh hrán). Keďže algoritmus prehľadáva každú možnú cestu, nie je zrovna najrýchlejší. Preto

existujú rôzne jeho rozšírenia. [8]

3.2 A* algoritmus

A* je jedným z najpoužívanejších nástrojov pre hľadanie ciest v hernom vývoji. Je populárny hlavne kvôli jeho dobrej škálovateľnosti a vyššej rýchlosti oproti klasickým algoritmom. A* je navrhnutý na hľadanie optimálnej cesty z jedného bodu do druhého, a to aj v komplexnejších grafoch. Je veľmi podobný Dijkstrovému algoritmu, ale s rozdielom, že implementuje metódu Greedy Best-First Search s cieľom minimalizovať celkovú hodnotu ohodnotenia nájdenej cesty. Taktiež musí pracovať so zjednodušenou reprezentáciu prehľadávaného priestoru, v ktorom ohodnocuje cestu na základe dĺžky cesty medzi jednotlivými hranami v grafe a heuristickej hodnoty.

Heuristikou znamená hľadanie nie úplne presného, ale zároveň dostatočne dobrého výsledku v obmedzenom čase. Integrácia heuristiky v tomto prípade znamená, že A* predpovedá, alebo odhaduje, vzdialenosť od aktuálneho uzla po najkratšiu cestu k cieľu, použitím rôznych heuristických funkcií. Táto informácia umožňuje algoritmu efektívne vyhodnocovať uzly a smerovať k najoptimálnejšej ceste. Taktiež sa zabraňuje opakovanému prehodnocovaniu už spracovaných uzlov, čím sa zvyšuje jeho efektivita. V hernom prostredí sa heuristika môže prispôsobiť špecifickým vlastnostiam prostredia. To umožňuje A* efektívne reagovať na rôzne herné situácie a dynamicky upravovať cestu v závislosti od stavu herného prostredia. Dobre navrhnutá heuristika môže výrazne urýchliť proces hľadania ciest a prispieť k realistickému a inteligentnému správaniu postáv v hre. [3] [8]

3.3 Waypoints

Hľadanie cesty komplexným prehľadávaním grafu môže byť časovo a výpočetne náročné. Jedným zo spôsobov, ako obísť tento problém je prepočítať si cestu dopredu. To sa dá napríklad použitím navigačných bodov, ktoré sú umiestnené do herného prostredia a slúžia ako uzly pre jednoduché výpočty ciest medzi týmito uzlom.

Cieľom je, aby každý navigačný bod na mape bol v zornom poli aspoň jedného ďalšieho bodu. Tým sa zaistí, že postava pri prechode medzi bodmi je ušetrená pred potrebou riešiť ako obchádzať (väčšinu) prekážok na ceste. V takejto sieti pevne definovaných bodov bude herná AI vždy schopná postavu dostať sa na akúkoľvek pozíciu na mape. Stačí jej len vedieť, ako sú uzly navzájom pospájané. [3]

4 PROCEDURÁLNE GENEROVANIE

Definíciou procedurálneho generovania obsahu je algoritmická tvorba herného obsahu s obmedzeným alebo nepriamym vstupom používateľa a je väčšinou založená na metódach umelej inteligencie. PCG sa vzťahuje na počítačový softvér, postupy alebo algoritmy ktoré dokáže vytvárať herný obsah sám alebo spolu s ľudského dizajnéra / vývojára. Obsah ktorý takáto algoritmus generuje je väčšina toho, čo hra obsahuje: úrovne, mapy, pravidlá hry, textúry, príbehy, predmety, úlohy, hudba, zbrane, vozidlá, postavy atď. Sú to teda spravidla fyzické objekty v hre. Naopak pod túto oblasť nespadá správanie neherných postáv alebo umelá inteligencia NPC. Táto metóda môže byť aplikovaná na videohry, počítačové hry, stolové hry, kartové hry, alebo hlavolamy. Je dôležité, aby systém generovania obsahu zohľadňoval dizajn, možnosti a obmedzenia hry, pre ktorú sa generuje. Kľúčovou požiadavkou na takto generovaný obsah je najmä to, aby bol hrateľný. Procedurálne vygenerovanú úroveň by malo byť možné dokončiť, nemal by byť problém pohybovať sa po generovanom teréne, s vygenerovanými objektami by sa malo dať manipulovať a celkovo musí byť možné takúto hru dohrať. Tým sa PCG odlišuje od takých činností, ako je generatívne umenie a mnohé druhy počítačovej grafiky, ktoré nezohľadňujú osobitné obmedzenia a možnosti herného dizajnu.

Najzrejmejším dôvodom na generovanie obsahu procedurálne je, že odpadá potreba, aby obsah vytváral človek, vývojár alebo dizajnér manuálne. Od začiatku počítačových hier, úsilie vynaložené na vývoj úspešnej komerčnej hry, sa viac-menej neustále zvyšuje. V súčasnosti je bežné, že hru vyvíjajú stovky ľudí v priebehu roka alebo dlhšie. To vedie k tomu, že menej hier je ziskových a menej vývojárov si môže dovoliť vyvinúť hru, čo následne vedie k menšiemu riskovaniu a menšej rozmanitosti na trhu s hrami. Spoločnosť zaoberajúca sa vývojom hier, ktorá by mohla nahradiť časť umelcov a dizajnérov algoritmami, by mala konkurenčnú výhodu, pretože hry by mohla vyrábať rýchlejšie a lacnejšie pri zachovaní kvality. Ďalším podstatným dôvodom je aj to, že procedurálne generovanie obsahu, najmä v nástrojoch inteligentného dizajnu, môže výrazne zvýšiť kreativitu jednotlivých ľudských tvorcov. To by mohlo umožniť malým tímom, ktoré ešte nemajú zdroje ako veľké spoločnosti, a dokonca aj amatérskym tvorcom, vytvárať obsahovo bohaté hry tým, že ich oslobodí od starostí o detaily a ťažkú prácu, pričom si zachovajú celkovú réžiu hier. Ďalším dôvodom na používanie PCG je aj to, že môže vývojárom pomôcť byť kreatívnejší. Algoritmické prístupy by môžu prísť s radikálne odlišným obsahom, než by vytvoril človek, a to tým, že ponúknu nečakané, ale správne riešenie daného problému. [9] [10]

Medzi známe hry, ktoré sú známe využívaním procedurálneho generovania, patria napríklad No Man's Sky, Minecraft, Rain World, séria Borderlands, séria Diablo, Elite: Dangerous a mnohé iné. Ide väčšinou o hry, ktoré obsahujú veľké herné mapy a svety

alebo značne rozmanitý počet a diverzitu herných objektov. Tento herný rozsah je možné niekdy dosiahnuť iba nejakou formou AI generovania, nakoľko tvoriť všetky objekty ručne by vývojárom zabralo večnosť. [11]

4.1 Parametre PCG

Požadované vlastnosti konkrétneho procedurálneho generovania sú pre každú aplikáciu odlišné. Bežne ale PCG potrebuje spĺňať:

- **Rýchlosť** - Požiadavky na rýchlosť sa líšia, od maximálneho času generovania milisekúnd až po mesiace. V závislosti od toho, či sa obsah generuje počas hrania, kedy musí algoritmus byť rýchly aby bol obsah generovateľný v reálnom čase, alebo počas vývoja hry.
- **Spoľahlivosť** - Niektoré generátory majú voľnú ruku v tom čo môžu generovať, pri iných je nutné zaručiť, že obsah, ktorý vytvoria bude spĺňa určité kritéria. Napríklad vygenerovaná budova bez vchodu by znamenala problém, zatiaľ čo náhodné rozloženie rastlín alebo stromov v ekosystéme mapy môže zvýšiť prirodzenosť.
- **Diverzita** - Často je potrebné vytvárať rôznorodý obsah, aby sa nestávalo, že vygenerované objekty budú príliš rovnaké. To napríklad znamená aby pri opakovanom hraní hry nevyzerali vygenerované úrovne, alebo NPC postavy vždy rovnako. Naopak vygenerovať úplne náhodnú a unikátnu mapu, ktorou ale nebude možné prejsť, tiež nie je žiadúce.
- **Efektivita** - Procedurálne generovanie obsahu byť všeobecne malo byť ekonomické z hľadiska výpočtovej náročnosti. To je dôležité najmä pre hry, ktoré sa hrajú na mobilných zariadeniach alebo iných systémoch s obmedzenými zdrojmi.
- **Prispôsobivosť** - Často je potrebné, aby procedurálny generátor bol v určitom zmysle nastavovateľný, dizajnérom alebo iným algoritmom, ktorý bližšie špecifikuje, čo sa má generovať. [9] [12]

4.2 Algoritmy PGO

4.2.1 Celulárne automaty

Ide o sú diskrétne výpočtové modely, široko využívané v informatike, fyzike alebo biológií. Celulárne automaty sú pomerne jednoduché na implementáciu a dajú sa použiť na simuláciu rôznych systémov, ako sú napríklad rastliny, živočíchy, doprava, ekonomika alebo dokonca celé herné svety. Celulárny automat pozostáva z n-rozmernej mriežky,

množiny stavov a množiny pravidiel pre prechody. Má mať tvar najčastejšie jednorozmerný - vektor, alebo dvojrozmerný - matica. Každá bunka v mriežke môže byť v jednom z niekoľkých stavov, v najjednoduchšom stave môžu byť bunky zapnuté alebo vypnuté. V priebehu času sa automat vyvíja v diskrétnych krokoch na základe predom daných pravidiel. V každej iterácií, každá bunka rozhodne o svojom novom stave na základe stavu samej seba a všetkých buniek v jej okolí. [9] [13]

4.2.2 L-Systemy

Taktiež známy ako Lindenmayerov systém, je matematický model, pôvodne vytvorený na popis rastu rastlín. V oblasti počítačovej grafiky a generovania obsahu v hrách sa L-systémy využívajú na procedurálne generovanie komplexných a prírodne vyzerajúcich vzorov, najčastejšie rastlín. L-systém definuje vývoj istého systému pomocou pravidiel, ktoré opisujú, ako sa tento systém bude vyvíjať v čase. Jeho komponentam je axiom, čo je počiatočný reťazec znakov a pravidlá, ktoré sú definované pre každý symbol abecedy, ktorú L-systém používa. V každom kroku je počiatočný reťazec pretransformovaný podľa pravidiel, čo vytvára nový reťazec, ktorý môže byť ďalej transformovaný alebo interpretovaný ako výsledný obraz, vzor alebo model. [9] [14]

4.2.3 Evolučné algoritmy

Evolučné algoritmy sú typom umelej inteligencie, ktorá napodobňuje prírodný proces evolúcie. Evolučné algoritmy pracujú s populáciou jedincov, ktorí reprezentujú možné riešenia úlohy, sadu parametrov alebo pravidlá definujúce herné prvky. Jedinci postupne prechádzajú procesom ohodnotenia ich kvality, selekcie, kríženia a mutácie, ktoré menia ich vlastnosti a smerujú populáciu smerom k optimálnemu riešeniu. Spôsoby reprezentácie, hodnotenia a modifikácie jedincov závisia od obsahu, ktorý chceme generovať. Algoritmus môže napríklad hodnotiť kritéria, ako estetika, zaujímavosť, hrateľnosť, náročnosť, vyváženosť, rozmanitosť herných objektov, a na základe toho môžu modifikovať parametre ako rotácia, translácia, škálovanie, deformovanie, pridávanie alebo odstraňovanie prvkov a podobne. [9]

4.2.4 Gradientový Šum

Gradientový šum je šumová funkcia, ktorá využíva pseudo-náhodné gradienty, teda vektory smeru a veľkosti, priradené k bodom na mriežke. Jeho princíp spočíva následnej interpolácií gradientov medzi susednými bodmi v mriežke, čím vytvára plynulé zmeny v hodnotách šumu. Používa sa na to, aby náhodne vygenerované hodnoty nevyzerali príliš chaoticky, ale aby boli spojité a prirodzené. Gradientový šum sa používa na vytváranie prírodných a realistických textúr alebo terénov, pretože dokáže napodobniť

rôzne javy, ako sú mraky, oheň, voda, reliéfy, kopce alebo údolia. [9]

4.3 Aplikácia PCG v hrách

4.3.1 Animácie

Štandardný prístup ku tvorbe pohybov pre animované postavy alebo pohyblivé objekty vo filmoch a hrách je animovanie pomocou key frames alebo pomocou technológie motion capture. V filmoch to znamená naanimovť každú snímku v scéne samostatne ručne. V hre je tento proces o to zložitejší, pretože cieľom je animovať každý pohyb postavy tak, aby boli pohyby plynulé, bez toho, aby si hráč počas hrania nevšimol prechod z jednej animácie na druhú. Tento proces síce ponúka plnú kontrolu nad riadením a kvalitou animácií, na druhú stranu je to zdĺhavý, pracný a často nákladný proces, ktorý pri vývoji zaberá veľa času.

Jedným z úspešných riešení tohto problému sú procedurálne animácie. Ide v skratke o spôsob dynamického generovania animácií pomocou matematických algoritmov a funkcií, namiesto fixne preddefinovaného správania. Tento spôsob umožňuje animovaným objektom do istej miery interagovať a prispôsobovať svoje pohyby na základe svojho okolia, či fyzikálnych podmienok v reálnom čase počas hrania hry. Táto metóda tiež prináša možnosť flexibilnejšie prenášať animácie medzi rôznymi postavami, čím odpadá potreba animovať neustále všetko odznova. Nevýhodou tu ale môže byť fakt, že takto generované animácie sú viac nepredvídateľné a môžu vizuálne pôsobiť neprirodzene, keďže sú vytvárané dynamicky. Jedným riešením môže byť použitie neurónových sietí alebo genetických algoritmov, ktoré dokážu animáciu dodatočne upraviť na základe učenia sa z reálnych alebo ručne vytvorených ukážkových animácií. [15] [16]

4.3.2 Mapy

Herná mapa, hlavne v hrách s otvorenými svetmi, je väčšinou najrozsiahlejší 3D model v hre. V hrách čoraz viac rastú nároky, aj snaha tieto herné svety modelovať väčšie, komplexnejšie a realistickejšie. To znamená, že proces modelovania týchto svetov je viac pracný a vyžadujú si špecializované zručnosti v oblasti 3D modelovania. Preto je aktívne rozvíjanou témou generovať terény, mapy, herné oblasti procedurálnou cestou (PTG). Procedurálne generovať herný terén môže byť ale náročné kvôli nezvyčajným tvarom a vzorom, ktoré sa bežne v prírode vyskytujú. Postupom času však vznikali rôzne metódy ktoré tieto problémy riešia. Moderné PTG môže zahŕňať používanie softvérových agentov, ktorí napodobňujú geologické procesy, používanie techník umelej inteligencie na rozpoznávanie a napodobňovanie vzorov ktoré sa v krajine vyskytujú alebo umožňuje dizajnérom so systémom priamo interagovať a kresliť niektoré prvky ručne. [17] [18] [19]

4.4 Textúry

Procedurálne textúry, sú také, ktoré sú čiastočne alebo úplne generované počítačom pomocou pevne stanovených parametrov. Výhodou je, že týmto spôsobom je možné generovať veľa rôznych variácií šablónovej textúry, a to v rôznych rozlíšeniach aj veľkostiach. Keďže proces tvorby textúry má pod kontrolou počítač, bývajú takto generované textúry omnoho kvalitnejšie, ako keby sa použije ručne nakreslená, alebo oskenovaná textúra.

Nemusí sa jednať len o tvorbu difúznych textúr, ktoré nesú informácie iba o farbe, ale textúry môžu byť rôznych typov. Napríklad pomocou gradientového šumu spomenutého vyššie je možné generovať výškové, šumové alebo normálové mapy. Nimi sa potom dá meniť obrazivosť textúry a tým simulovať drsný povrch, rôzne nedokonalosti, škrabance, poškodenie alebo špecifický povrch rôznych materiálov. Dnes sa už nástroje na procedurálne generovanie takýchto textúr štandardne vyskytujú v špecializovaných 2D aj 3D programoch, ako je napríklad Blender, Substance Painter, alebo Substance Designer. [9] [20]

4.5 3D Modely

Keď máme jasne definované procedurálne pravidlá, a výkonný program, ktorý ich dokáže implementovať, môžeme algoritmickým spôsobom generovať aj 3D objekty. To, že nie je potrebné vytvárať nové modely vždy od začiatku, je veľký prínos, hlavne keď sa jedná modelovane rozmanitých objektov ako je vegetácia, postavy, zbrane, budovy, jaskyne alebo vozidlá. Implementácia môže byť v tomto prípade rôzna, od jemných úprav na už existujúcich modeloch, cez kombinovanie menších objektov do väčších a komplexnejších celkov, alebo vytvorenie celého 3D modelu úplne od nuly.

Pri generatívnom modelovaní 3D objektov sa môžu používať fraktály aj L-systémy. Zo všetkého generovaného obsahu je však pri 3D modely väčšinou najjednoduchšie spozorovať nejakú nedokonalosť, preto sa takéto generovanie môže použiť na prototypovanie a výsledný model sa doladí ručne. Veľkou výhodou tohto prístupu je, že modely je možné upravovať nedeštruktívnym spôsobom. To znamená, že úprava topológie objektov prebieha na rozdielnych vrstvách. Jednotlivé vrstvy sa dajú vzájomne kombinovať, a zmeny aj úpravy sa dajú jednoducho vrátiť do podvodného stavu. [21]

5 UMELÁ INTELIGENCIA A STROJOVÉ UČENIE

Umelá inteligencia je počítačová veda, ktorá sa zaoberá vývojom strojov a algoritmov, ktoré dokážu *myslieť* a rozhodovať sa bez ľudského zásahu. To týmto algoritmom umožňuje vykonávať rôzne komplexné úlohy, ktoré by zvládol inak len človek. Tieto algoritmy sú väčšinou tvorené komplexným súborom matematických pravidiel, ktoré definujú to, akým spôsobom sa tieto modely učia a pracujú. Podmnožinou umelej inteligencie je aj Strojové učenie (ML), ktorého účelom je umožniť počítačom, aby sa mohli učiť z údajov a rozhodovať sa, alebo predpovedať určité javy bez explicitného naprogramovania. Toto odvetvie zahŕňa vývoj rôznych algoritmov a štatistických modelov, na základe ktorých počítač vykonáva špecifické úlohy a v priebehu času sa v nich zlepšuje vďaka učeniu sa z veľkého množstva dát.

Toto učenie prebieha tak, že AI program buď obdrží, alebo sám získa množinu trénovacích dát, na ktorých sa učí vykonávať nejakú činnosť. Takto naučený model môže ďalej vykonávať tieto činnosti na reálnych dátach v praxi. Spôsob, akým sa tieto trénovacie údaje získavajú a označujú, väčšinou udáva, o aký typ algoritmu sa jedná, a akú formu učenia využíva. [22] [23]

Základné formy učenia využívané v strojovom učení sú:

5.1 Učenie s učiteľom

Tento typ učenia je implementovaný spôsobom, že algoritmus ako trénovacie dáta prijíma jasne označené a roztriedené údaje. Takto označené údaje môže ďalej používať na predpovedanie výsledkov iných údajov. Toto učenie vychádza z konceptu, že algoritmus sa učí pod niečim dohľadom. Táto forma učenia zahŕňa vyhodnocovanie a prieskum výsledkov, ktoré algoritmy generujú, testovanie vytvorených modelov, alebo rôzne porovnávania s pôvodnými údajmi. Medzi známe algoritmy založené na tejto metóde učenia sú Rozhodovacie stromy, Support vector machines, Naivné bayesovské klasifikátory, Lineárna regresia alebo Logická regresia.

5.2 Učenie bez učiteľa

V tomto prípade ide o učenie opačným spôsobom, kde algoritmy dostávajú trénovacie údaje, ktoré nie sú označené. Algoritmy tak musia viac menej samy prísť na to, akým spôsobom majú s údajmi pracovať. To robia spôsobom, že počítajú a vyhodnocujú rôzne vzťahy medzi danými údajov, čím o nich získavajú lepší prehľad. Na základe toho sa napríklad algoritmus dokáže bez vonkajšej pomoci naučiť triediť a klasifikovať rôzne dáta na základe ich spoločných vlastností. Medzi algoritmy tohto typu patrí K-means clustering, Gaussian mixture models, Autoenkodéry, Generatívne adverzné siete

alebo Samoorganizačné mapy.

5.3 Učenie posilňovaním

Pri tomto type učenia sa algoritmy učia , na základe dostávania spätnej väzby z výsledkov úloh ktoré plnia. Táto spätná väzbe je väčšinou vo forme pozitívnej odmeny, prípadne penalty. V tomto prípade algoritmus pracuje v iteráciách, v ktorých sekvenčne vykonáva rôzne činnosti a dostáva na ne spätnú väzbu, podľa preddefinovaných pravidiel. Algoritmus tak môže vyhodnotiť svoje predošlé akcie a na základe nich sa upravovať svoje rozhodovanie a ďalší postup tak, aby celkovú odmenu maximalizoval. Spätná väzbe môže byť v závislosti od implementácie pozitívna aj negatívna. [22] [23]

5.4 Fitness funkcia

Algoritmy implementujúce nejakú formu učenia potrebujú spôsob, ktorým budú overovať vhodnosť aktuálnych riešení. Na to presne slúži účelová, alebo fitness funkcia. Funguje tak, že konkrétnemu riešeniu priraďuje skóre, väčšinou je to jednorozmerná číselná hodnota, podľa toho ako dobre toto riešenie rieši daný problém. Existuje veľké množstvo spôsobov, ako účelovú funkciu definovať, v závislosti na tom čo chceme vo vyhľadávacom priestore nájsť. To ako bude definovaná zase výrazne ovplyvní rozhodovanie algoritmov, a tým aj rýchlosť nájdenia vhodného riešenia. [24]

6 UMELÉ NEURÓNOVÉ SIETE

Táto špecifická rodina AI algoritmov, je založená napodobňovaní neurónov v ľudského mozgu a proces učenia. Tieto algoritmy sú zvyčajne značne komplexnejšie oproti ostatným AI algoritmom, avšak ich výkon môže byť rádovo vyšší. Základná stavba neurónových sieti pozostáva z uzlov (neurónov), ktoré sú uložené vo vrstvách, a komunikujú s inými uzlami prostredníctvom spojení. Každá neurónová sieť obsahuje vstupnú a výstupnú vrstvu, a premenlivý počet vnútorných vrstiev. Počet neurónov v jednotlivých vrstvách sa líši v závislosti od konkrétneho typu neurónovej siete. Vstupná vrstva pôsobí ako vstupný bod pre dáta s ktorými sieť pracuje, a výstupná vrstva predstavuje finálnu odpoveď neurónovej.

Dáta sa v sieti spracovávajú vo vnútorných vrstvách, tak, že jednotlivé neuróny tieto dáta preberajú od neurónov pred nimi, a vyhodnotia ich pomocou aktivačnej funkcie a pošlú dáta ďalším neurónom. Aktivačné funkcie sú lineárne alebo nelineárne matematické funkcie, ktoré udávajú, či daný neurón aktivuje alebo nie (pošle/nepošle dáta do ďalšieho neurónu). To akú aktivačnú funkciu použijeme významne ovplyvňuje ako rýchlo a efektívne sa dokáže sieť učiť. Medzi najpoužívanejšie aktivačné funkcie patria ReLU, Sigmoid, TanH alebo Softmax. Jednotlivé spojenia medzi neurónmi sú navyše zaťažené váhami, mierne upravujú správanie neurónov. Na základe porovnania aktuálneho výstupu a požadovaného výstupu, neurónová sieť upravuje tieto váhy tak, tak aby minimalizovala rozdiel medzi týmito výstupmi. Tento iteratívny proces upravovania váh predstavuje učenie neurónovej siete.

Takto natrénovanú sieti je možné použiť na klasifikáciu a rozpoznávanie vzorov, rozhodovanie, alebo riešenie rôznych optimalizačných problémov, a to aj z dáta, ktoré predtým nikdy nevidela. Svoje predpovede robí práve na základe toho čo sa naučila počas trénovacieho procesu. Rôzne rozloženie neurónov, spojení a tok dát v sieti definuje to, o aký typ siete ide. Medzi jednoduché siete používané na klasifikáciu dát patrí napríklad Perceptron alebo Feedforward Neural Networks. Na zložitejšie úlohy ako spracovanie obrazu alebo počítačové videnie sa používajú Convolutional Neural Networks, ktorý stavba je značne komplexnejšia. Siete ako Recurrent Neural Networks alebo Long Short-Term Memory Networks sú zase vhodné na spracovanie prirodzeného jazyka vo forme reči alebo textu. [22] [25] [26] [27]

7 EVOLUČNÉ A GENETICKÉ ALGORITMY

Evolučné algoritmy sú triedou optimalizačných algoritmov, ktoré sú založené na princípe evolúcie a prirodzeného výberu. Na základe simulovania týchto mechaník sa chytrým spôsobom pokúšajú nájsť najlepšie riešenie na nejaký kombinatorický problém, bez toho, aby museli overiť všetky možné výstupy v rámci vyhľadávacieho priestoru. Tieto algoritmy nepredpovedajú výstup na základe vstupných alebo trénovacích dát, ale namiesto toho sa snažia optimalizovať / minimalizovať daný problém. Ich výsledkom môže byť jedna hodnota, ale aj celé pole hodnôt, ktoré budú predstavovať najlepšie riešenie.

Evolučné a genetické algoritmy sú silné v tom, že môžu rýchlo získať čiastočne správne riešenie z masívneho, občas až enormne veľkého vyhľadávacieho priestoru. Práve svojou schopnosťou generovať rôzne variácie rovnako dobrého výsledku alebo riešenia, môžu v kombinácií s procedurálnym generovaním zabezpečiť aby bol vytváraný obsah zaujímavý aj pri opakovaných hraniach rovnakej hry alebo úrovne. Tieto algoritmy väčšinou disponujú širokou škálou nastaviteľných parametrov, čím sa dá výstup prispôsobiť širokej škále problémov. Ich výstupy sú však viac menej ne-deterministické, čo môže viesť k výskytu neočakávaných výsledkov. Preto je veľmi dôležitou súčasťou implementácie týchto programov starostlivo navrhnutá fitness funkcia, a parametre prispôsobené danej aplikácií. [24] [28]

7.1 Genetický algoritmus

Jeden z najznámejších evolučných výpočetných techník je zrejme Genetický algoritmus. Je založení primárne na princípe mutácií a prirodzenom výbere najsilnejšieho jedinca, a tak ako ostatné podobné algoritmy, pracuje s populáciou, ktorá predstavuje možné riešenia. Príkladom použitia tohto prístupu v hrách, môže byť snaha nastaviť parametre pre ne-hráčsku postavu tak, aby predstavovala pre hráča v boji dostatočnú výzvu, ale aby bola zároveň dobre balancovaná. Dá sa tiež použiť pre naučenie hernej AI vykonávať nejakú konkrétnu logiku, ako kráčanie po vyznačenej trase.

Prvý krok pri inicializácií algoritmu je vytvorenie počiatočnej populácie, ktorá predstavuje náhodné prvky z priestoru možných riešení. Jednotliví jedinci zároveň predstavujú možné riešenia daného problému. Táto populácia sa za behu algoritmu postupne v priebehu času obmieňa a zdokonaľuje, čím sa zlepšuje aj celkové finálne riešenie. Noví jedinci populácie sú tvorení krížením aktuálnych jedincov, ktorí sú vyberaní podľa stanovených pravidiel. Najčastejšie chceme aby boli krížení najlepší jedinci, čím sa zvyšuje šanca, že novo vzniknuté riešenie bude ešte lepšie. To, akým spôsobom GA vie či, je nejaké riešenie lepšie alebo horšie oproti ostatným, zistí na základe otestovania tohto riešenia fitness funkciou. Lepší jedinci budú mať jednoducho lepšie skóre.

Ak však len kombinujeme jednotlivé prvky jedincov v populácií, boli by sme veľmi limitovaní možnými riešeniami. Aby sa zvýšila variabilita možných riešení, algoritmus navyše napodobňuje genetickú mutáciu tým, že z času na čas robí malé náhodné zmeny v parametroch jednotlivých jedincov, čím jednoducho rozširuje vyhľadávací priestor. Ideálne parametre ovplyvňujúce mutáciu budú zase odlišné pre každý riešený problém. [24] [28]

8 GENERATÍVNA UMELÁ INTELIGENCIA

Generatívna umelá inteligencia zaznamenala v poslednom období pozoruhodný rast a neustále napreduje závratnou rýchlosťou, čo vedie k vývoju širokej škály aplikácií v rôznych oblastiach. Tento koncept sa vzťahuje na umelú inteligenciu, ktorá dokáže generovať nový obsah, namiesto toho, aby jednoducho analyzovala alebo konala na základe existujúcich údajov ako expertné systémy. Proces komunikácie s generatívnou AI sa často začína tzv. *promptom*, ktorí umožňuje používateľovi alebo zdroju údajov poskytnúť počiatočný dotaz/príkaz spolu s údajmi na usmernenie generovania obsahu. Tradičné AI algoritmy naopak často nasledujú preddefinovaný súbor pravidiel na spracovanie údajov a tvorbu výsledku.

Najmä najmodernejšie jazykové a obrazové generatívne modely, využívajúce zdatnosť hlbokého učenia a transformačných architektúr, umožnili generovanie širokého spektra obsahu. Modely generatívnej AI, s komplexnými implementáciami, trénované na rozsiahlych datasetoch údajov, majú mimoriadnu schopnosť vytvárať nový a rôznorodý obsah. Ich sila je v tom, že môžu spracovávať a učiť sa z informácií získaných z množstva zdrojov, naprieč internetom a využívaním znalostí z týchto údajov, dokážu tieto modely vytvárať širokú škálu multimediálnych formátov, vrátane grafiky, videa, zvuku alebo textu.

V posledných rokoch sa vďaka neustálemu rastu výpočtového výkonu využívajú hlboké neurónové siete (DNN), transformátory (GPT) a ďalšie inovatívne modely, ako sú generatívne adverzné siete (GAN) a variačné autoenkodéry. Všetky tieto modely dokážu efektívne zachytiť komplexnosť údajov, vďaka čomu sú schopné modelovať vysokorozmerné pravdepodobnostné rozdelenia jazyka alebo obrazov z rôznych oblastí. Táto všestrannosť umožňuje bezproblémovú konverziu medzi multimediálnymi formátmi, vďaka čomu sú generatívne modely značne nápomocné v mnohých aplikáciách. Špeciálne v kreatívnych oblastiach a pri riešení nových problémov, kvôli schopnosti autonómne generovať rôzne typy a variácie nových výstupov. Jedným z najvýznamnejších aspektov generatívnej umelej inteligencie je jej potenciál pre nekonečné množstvo aplikácií.

Aktuálne generatívnu AI možno použiť na vytváranie realistických obrázkov z textových opisov, vytvárať video zo zvuku, alebo dokonca generovať hudobné skladby na základe špecifických štýlov alebo emócií. Okrem toho má generatívna AI potenciál spustiť revolúciu v odvetviach, ako je reklama, zábava a vzdelávanie, ale aj vedecká sféra, a to automatizáciou tvorby obsahu a poskytovaním personalizovaných odpovedí. So schopnosťou generovať širokú škálu výstupov môžu tieto modely pomôcť podnikom aj jednotlivcom ušetriť čas a zdroje a zároveň využívať nové kreatívne možnosti. Svojou schopnosťou učiť sa z rôznych zdrojov, možnosťou generovať rôzne výstupné formáty a kombinovať tieto výstupy medzi sebou, so seba robia fenomenálny nástroj v dnešnej

svete riadenom technológiami. [29] [30] [32]

8.1 Použitie

Táto technológia sa stáva stále dostupnejšou pre používateľov všetkých druhov vďaka možnosti jednoducho ju integrovať a vyladiť pre rôzne druhy aplikácií. Populárne prípady použitia generatívnej AI zahŕňajú:

- Integráciu chatbotov do prehliadačov, zákazníckeho servisu alebo technickej podpory,
- Zlepšenie dabingu filmov a vzdelávacieho obsahu v rôznych jazykoch,
- Písanie e-mailov, zoznamovacích profilov, životopisov, dokumentácií...,
- Vytváranie fotorealistického či abstraktného umenia v konkrétnom štýle,
- Sumarizácia obsahu a zjednodušenie komplexných informácií,
- Automatizácia tvorby obsahu,
- Tvorba hudby v špecifickom štýle alebo tóne,
- Navrhovanie nových zlúčenín a liekov,
- Optimalizácia nových návrhov čipov,
- Reklama produktov,

8.2 Obmedzenia a nevýhody

Oproti ostatným typom umelej vyvíjanej v minulosti, tento typ umelej inteligencie nepracuje deterministicky a preto jednoduché odvodiť, akým spôsobom prichádza ku výstupom, ktoré generuje. To môže sťažiť ladenie chýb alebo zabezpečenie súladu s požadovanými výsledkami.

8.2.1 Nároky na výpočetný výkon

Kľúčovou súčasťou trénovanie a aj následnej prevádzky generatívnych modelov sú grafické karty GPU, kvôli ich schopnosti spracovať veľké množstvo dát paralelne na veľkom množstve procesorov. Beh týchto grafických kariet si počas procesu trénovania vyžaduje značné výpočtové zdroje a čas. Dôsledný tréning modelu na môže bežne trvať v rádoch stoviek hodín, čo ho robí nákladným a energeticky náročným. To v konečnom dôsledku zvyšuje náklady na nasadenie aj následnú réžiu modelu, čo môže obmedziť jeho široké prijatie.

8.2.2 Obmedzená kreativita

Aj keď generatívna umelá inteligencia dokáže vytvárať nový obsah a vyniká v napodobňovaní vzorov a štruktúr na základe existujúcich údajov, z hľadiska kreativity a originality je obmedzená práve dátami, ktoré boli použité pri jej učení. To znamená, že generovať obsah môže iba na základe toho, čo sa naučila z trénovacej množiny dát. S tým je spojená aj skutočnosť, že AI chýba schopnosť skutočne pochopiť význam a kontext údajov, ktoré vytvorí. To môže viesť k výstupom, ktoré sú síce presvedčivé, ale môže sa jednať o úplný nezmysel.

8.2.3 Predpojatosť

Obsah generovaný Generatívnou AI môže byť skreslený aj vtedy, ak sú skreslené údaje, na ktorých bola AI trénovaná. Ak sú napríklad údaje použité na trénovanie generatívneho modelu zaujaté voči určitému riešeniu problému alebo skupine ľudí, vygenerované údaje môžu túto zaujatosť odrážať, a tým generovať diskriminačný alebo škodlivý obsah. V tomto bode je dôležitý starostlivý výber údajov a stratégie vyčistenia trénovacieho datasetu.

8.2.4 Obmedzené zdroje informácií

Generatívna umelá inteligencia je najvhodnejšia pre aplikácie, kde existuje veľké množstvo existujúcich údajov na ktorých sa dá AI natrénovať. Z toho vyplýva, že v prípadoch, keď sú vstupné údaje obmedzené alebo sú veľmi zložité, generatívna AI stráca účinnosť aj kreativitu.

8.2.5 Generovanie dezinformácií

Generatívna AI môže byť veľmi jednoducho vedome použitá na škodlivé účely, ako je generovanie falošných správ, deepfakes alebo iných typov falošných dezinformácií. S extrémne rýchlo pribúdajúcim technológiami v tejto oblasti už nemusí ísť len o obsah v podobe textu, ale aj v podobe zvuku, obrázkov alebo videa. To vyvoláva etické obavy z potenciálneho zneužitia technológie pre súkromie jednotlivcov ale aj národnú bezpečnosť. [30] [33]

8.3 Architektúry generatívnej AI

Jednotlivé typy generatívnej AI závisia od stavby ich podkladovej neurónovej siete. Medzi tieto siete patria:

8.3.1 Autoencoder

Autoenkóder (AE) je špeciálny typ architektúry neurónovej siete, ktorá využíva metódu učenia bez učiteľa. Táto architektúra sa skladá z troch častí ktorými sú Enkodér, Bottleneck a Dekodér. Je navrhnutá tak, aby dokázala efektívne skomprimovať (zakódovať) vstupné dáta vrátane ich základných vlastností a potom ich spätne rekonštruovať (dekódovať) z tejto komprimovanej podoby. Základným cieľom tejto siete je zistenie minimálneho počtu dôležitých údajov potrebných na efektívnu rekonštrukciu vstupných dát.

Enkodér má za úlohu prostredníctvom redukcie dimenzionality postupne komprimovať vstupné dáta. Ako dáta prechádzajú stále menšími a menšími vrstvami neurónovej siete, sú stláčané do menšieho počtu rozmerov. Toto enkodér núti, aby sa naučil extrahovať len tie informácie, ktoré sú najvhodnejšie na presnú rekonštrukciu pôvodného vstupu. Bottleneck potom obsahuje túto najviac komprimovanú reprezentáciu vstupných dát. Slúži ako výstup enkodéru a zároveň je vstupom pre dekodér. Dekodér naopak obsahuje vrstvy s narastajúcim počtom uzlov, ktoré dekomprimujú zakódované údaje. Tým sa snaží efektívne rekonštruujú údaje späť do ich pôvodnej formy. Účinnosť tejto siete sa spotom meria porovnaním pôvodého vstupu a jeho rekonštrukcie.

Väčšina typov autoenkdérových sieti má uplatnenie v úlohách súvisiacich s počítačovým videním ako je extrakcia vzorov/kľúčových bodov, kompresia dát, odšumovanie/rekonštrukcia obrazu, detekcia anomálií alebo rozpoznávanie tváre. Niektoré varianty ako variačné autokódery (VAE) alebo adversariálne autokódery (AAE), sú viac prispôsobené na generatívne úlohy, ako je generovanie obrázkov alebo generovanie údajov časových radov. Variačné autoenkodéry sa navyše nesnažia len presne zrekonštruovať dáta do pôvodnej podoby, ale učí sa zároveň ako z týchto dát vytvárať nové dáta, ktoré predtým nikdy nevidel. To sa špeciálne dá uplatniť pri generovaní grafiky, kde je potrebná vysoká rozmanitosť. [34] [35] [36]

8.3.2 Transformer

Transformer je typ architektúry neurónových sietí, ktorý je adaptovaný na spracovanie sekvenčných údajov, ako je text v prirodzenom jazyku, zvukové signály alebo údaje časových radov. Najčastejšie sa Transformerové neuronové siete využívajú na spracovanie prirodzeného jazyka (NPL), ale dajú sa použiť aj na iné operácie zahŕňajúce text, ako strojový preklad, vyhľadávanie informácií, klasifikácia textu, sumarizácia dokumentov alebo popisovanie obrázkov.

Tento model je tiež zložený z enkodéru a dekodéru. Enkodér je zodpovedný za premapovanie vstupného textu na číselné vektory, ktorým dokáže program porozumieť a Dekodér z týchto vektorov dokáže spätne poskladať text vo forme ľudského jazyka.

Ide v podstate o náhradu architektúry Rekurentnej neurónovej siete (RNN) a siete Long Short Term Memory (LSTM) oproti ktorým sa dokáže rýchlejšie učiť a spracovávať dlhšie reťazce vstupného textu.

Transformery môže efektívne pochopiť kontext a vzťahy medzi slovami v texte vďaka mechanizmu zvanému *attention*. Táto vlastnosť umožňuje modelu sledovať spojenia medzi slovami nielen v rámci jednotlivých viet, ale aj v celých odsekoch, stranách alebo kapitolách kníh. To výrazne zlepšuje schopnosť modelu zachytávať širší kontext informácií. Nové modely preto môžu byť trénované na miliardách strán textu, čoho výsledkom sú hlbšie a komplexnejšie odpovede. Transformeri majú výhodu v tom, že nemusia byť predtrénované na označených údajoch, čo umožňuje výskumníkom trénovať stále väčšie modely bez výraznejšieho manuálneho označovania dát.

Jedným z hlavných rozdielov tiež je, že vstupnú sekvencia dát vie Transformer spracovávať paralelne pomocou GPU, čo dodatočne zvyšuje rýchlosť jeho tréningu. Medzi najvýkonnejšie modely tohto typu patria napríklad modely BERT a séria GPT-3, GPT-3.5 či GPT-4. [30] [31] [37] [38] [39]

8.3.3 Generative Adversarial Network

GAN patria medzi najúspešnejšie generatívne modely učenia bez učiteľa pre generovanie grafického obsahu. Zatiaľ čo väčšina ostatných AI generatívnych prístupov je založená na optimalizácii, GAN sú založené na teórií hier, medzi dvoma modelmi strojového učenia, typicky implementované pomocou neurónových sietí. Tento typ neurónovej siete je zložený z dvoch hlavných komponentov: generátoru a diskriminátoru.

Generátor má za úlohu vytvoriť obraz alebo vzor, tak aby bol čo najmenej rozoznateľný od skutočných dát. Trik je v tom že generátor ku reálnym dátam nemá priamy prístup a je obmedzený iba na komunikáciu s Diskriminátorom. Diskriminátor naopak vidí na skutočné dáta, a aj na tie ktoré vytvorí generátor. Jeho úlohou je potom posúdiť, či dané dáta sú skutočné alebo sú vygenerované generátorom. Jeho cieľom je byť čo najefektívnejší pri odhaľovaní rozdielov medzi reálnymi a generovanými dátami, do doby dokiaľ bude mať problém dané dáta rozlíšiť. Učenie takejto siete potom prebieha spôsobom, že Generátor produkuje čím ďalej tým lepšie výstupy, zatiaľ čo Diskriminátor sa snaží čím úspešnejšie odhaliť, či sú falošné. Tento prístup umožňuje široké využitie v oblasti generovania grafiky ako sú obrázky alebo videá, transformácia jedného obrázku alebo štýlu na druhý, zväčšenie rozlíšenia obrázkov, alebo ich klasifikáciu. Ich hlavná výhoda spočíva v tom, že umožňujú vytváranie nových dát bez potreby presného programovania pravidiel. [29] [40] [41]

9 GENERATÍVNE MODELY

Generatívne modely sú triedou modelov strojového učenia umelej inteligencie, ktoré dokážu generovať nové údaje na základe tréningových údajov (vybrané údaje z reálnych dát použitých na naučenie modelu robiť určitú vec). Dva popredné typy modelov sú:

9.1 Veľké jazykové modely

Veľký jazykový model (LLM) je typ neurónovej siete založenej na architektúre Tranformer, ktorá využíva metódú *učenia bez učiteľa* alebo *s polo učiteľom*. Neurónová sieť je teda trénovaná prostredníctvom dátovej sady, ako je text, ktorý je prevažne neoznačený alebo nekategorizovaný. To umožňuje použiť pri trénovaní obrovské množstvo údajov, ktoré sa model môže sám učiť spracovávať. Vstupom môžu byť v podstate akékoľvek textové údaje dokonca v rôznych jazykoch, ako napríklad literatúra, odborné články, repozitáre kódu alebo akékoľvek iné dáta načítané z internetu.

Pri spracovávaní vstupných dát a hľadaní vzorov sa pri tomto modeli uplatňuje už vyššie spomenutý *attention* mechanizmus. Keď sa model pozerá na všetky slová v texte naraz, postupne začne chápať, ktoré slová sa najčastejšie vyskytujú spolu, a ktoré slová sú pre význam vety najdôležitejšie. Na základe týchto naučených vzorov sa jazykový model následne snaží predpovedať ďalšie slovo, ktoré najlepšie pasuje do generovaného textu alebo odpovede. Chyby, ktoré model počas generovania urobí, môže zohľadniť pri úprave váh jednotlivých neurónových spojení, ktoré priraďuje rôznym slovám, a na základe toho sa rozhodovať pri ďalšom generovaní. Pri generovaní dlhých blokov textu môže dokonca predpovedať ďalšie slovo v kontexte všetkých slov, ktoré doposiaľ vygeneroval. Veľmi zjednodušene, veľký jazykový model na základe štatistky jednoducho háda, aké slovo bude v texte nasledovať, na základe toho čo doposiaľ zanalyzoval. Týmto spôsobom však vždy existuje šanca, že model môže vygenerovať text ktorý znie vierohodne, ale z logického hľadiska nedáva zmysel.

Hlavnou vlastnosťou Veľkých Jazykových Modelov, ktorá vo všeobecnosti udáva ich výkon a rozmer, sú ich parametre. Ide o súčasť modelu, ktorá riadi transformáciu vstupných dát na výstup. Tieto parametre, určujú vplyv konkrétnych vstupných vlastností na výsledný výstup. Čím viac parametrov daný model má, tým je komplexnejší a zároveň schopnejší rozpoznať zložitejšie vzorce z dát. Na druhej strane komplexita modelu zvyšuje výpočetný výkon aj pamäťové nároky. V závislosti od komplexity a účelu modelu sa počet parametrov môže hýbať od 3 miliárd (3B) pri jednoduchších open source modeloch až po desiatky miliárd pri top modeloch ako je ChatGPT (175B). [30] [31] [42] [43] [44]

9.2 Difúzne modely

Difúzne modely sú podskupina generatívnych modelov, ktoré sú trénované na obrázkových dátach. Fungujú tak, že namiesto operovania vo vysoko rozmernom priestore obrazu, najprv komprimujú obraz do latentného priestoru. Model potom postupne ničí obraz pridávaním šumu. Cieľom trénovacieho procesu je následne opačný proces ku zašumovaniu, čiže snaha o regeneráciu dát do pôvodného stavu. Natrénovaný difúzny model je tak možné použiť na generovanie nových dát, napríklad generovanie vysoko kvalitných obrázkov.

Medzi takéto modely patrí napríklad Stable Diffusion, model hlbokého učenia, ktorý dokáže konverziou textu na obrázky vytvárať obrázky v fotorealistickom, aj animovanom štýle. Tento model od svojej prvej verzie 1.5 prešiel rozsiahlym vývojom a vylepšeniami vo forme rôznych nových modelov a funkcií. Najnovšia verzia tejto rodiny modelov je Stable Diffusion XL. Oproti ostatným Text to Image generátorom, SD ponúka rozsiahle možnosti kontroly nad výstupom, medzi ktoré patrí napríklad možnosť voľby rozlíšenia, štýlov alebo počtu obrázkov. Okrem vytvárania čisto nových obrázkov môže SD pridávať alebo nahrádzať časti obrázkov, zväčšovaniu veľkosti a rozlíšenie obrázku, alebo prevádzať jeden obrázok na druhý, prípadne kopírovať štýly medzi obrázkami. [45]

Rôzne ďalšie modely môžu byť trénované napríklad na zvukových dátach, alebo 3D modeloch. Podľa toho sa jednotlivé modely dajú označovať ako modely Text-to-text, Text-to-image, Text-to-video, Text-to-3D, Text-to-code, Text-to-speech. To značí, že generujú obsah na základe textového vstupu. Tento proces je však možné implementovať aj opačným smerom. Príklaom môžu byť modely Image-to-Text, Image-to-image, Image-to-3D alebo image-to video. To znamená, že z obrázku je AI schopná popísať vizuálny obsah obrázku slovami, vytvoriť nový obrázok alebo 3D model z 2D predlohy, alebo dokonca obrázok animovať. Doplnenie tohto setu o modely Video-to-seech môžu navyše takto tvorené videa ozvučiť, alebo doplniť o hudbu. [29] [46]

II. PRAKTICKÁ ČASŤ

10 VÝVOJOVÉ NÁSTROJE

10.1 Unity Engine

Unity, jeden z popredných herných enginov, a populárna voľba pre vývoj Indie hier. Tento engine umožňuje vyvíjať hry pre širokú škálu platforiem vrátane PC, Mac, mobilných zariadení aj konzol. Ako platforma na tvorbu hier je veľmi flexibilná, dobre zdokumentovaná a veľmi rozšíriteľná na tvorbu takmer všetkých žánrov 2D aj 3D hier. Má tiež plnú podporu pre vývoj vo virtuálnej a rozšírenej realite, a preto môže byť použitá ako nástrojom na skúmanie architektúry, automatizáciu alebo simulácie.

Hlavným vývojovým prostredím je Unity Editor, primárne prostredie pre vývoj hry s 3D vizuálnym rozhraním. Tu je možné konfigurovať jednotlivé nastavenia hry, modelovať scény, upravovať skripty, testovať vývojové verzie hry a mnohé ďalšie. O build hry, vykresľovanie grafiky, fyzické simulácie a spracovávanie zvuku sa následne stará samotný Unity Engine. Samotné prostredie implementuje tiež integrovaný verzovací systém, profiler pre analýzu výkonu hry v reálnom čase, rôzne animačné nástroje, rendering a funkcie Drag-and-Drop vo všetkých častiach editora.

Medzi zaujímavé doplnky patrí Asset Store, rozsiahla databázapred pripravených assetov, modelov, textúr, animácií, alebo rôznych skriptov. Tu sa dá nájsť obrovské množstvo oficiálnych aj komunitných zdrojov pre rýchlejší vývoj. [47] [48]

10.2 Jazyk C#

Primárny programovací jazyk prostredia Unity je C#. Výkonný a všestranný, interpretovaný programovací jazyk vyvíjaný spoločnosťou Microsoft. C# je pomerne jednoduchý na naučenie a prechod z jazykov ako C++, Java nie je vôbec problém. Tým, že je objektovo orientovaný a staticky typovaný, je možné v ňom písať čistý a modulárny kód. Je postavený na bezplatnej, open-source vývojárskej platforme .NET Framework. Vďaka tomu je muliplatformný a tak použiteľný na vytváranie rôznych typov aplikácií. Najčastejšie sa používa na vývoj server-side webových aplikácií pomocou platformy .NET, tvorbu desktopových aplikácií pre Windows, vývoj natívnych mobilných aplikácií pre iOS a Android, implementáciu cloudových služieb, alebo na vývoj hier.

Okrem Unity je C# možné použiť aj iných herných enginoch, ako napríklad Godot, alebo Unreal Engine. V kombinácií s herným enginom Unity je efektívny hlavne pre jeho bezproblémovú integráci so skriptovacím rozhraním API prostredia Unity. C# natívne podporuje rôzne funkcie, vďaka ktorým sa v kontexte vývoja hier dá dobre implementovať herná logika, spúšťanie a riadenie udalosti, sieťovú komunikáciu, alebo reagovanie na vstupy od používateľa. Práve vďaka jeho optimalizácií v oblasti práce so sieťovou komunikáciu je C# o to viac vhodný pre tento projekt, keďže časť hernej

logiky bude implementovať komunikáciu so serverom pomocou REST API. [49]

11 POUŽITÉ AI TECHNOLÓGIE

Generatívnu umelú inteligenciu je aktuálne možné používať v dvoch variantoch. Buď ako cloudovú službu, kedy užívateľ komunikuje s AI modelom na vzdialenom serveri, alebo lokálne spustením AI modelu priamo na svojom počítači. Pre tento projekt som sa zameral práve na lokálne technológie z viacerých dôvodov.

Vyššia dostupnosť: Lokálne modely fungujú v offline režime, nepotrebujú teda žiadne spojene so servermi alebo platformami tretích strán. Takýto model je preto možné prevádzkovať, v akejkoľvek situácií, bez nutnosti riešiť podmienky používania, poplatky, čakanie vo fronte, či limitácie vo forme obmedzeného počtu generovaní konkrétnej platformy. Taktiež nie všetky online platformy poskytujú bezplatný prístup ku API jednotlivých modelov, čo znemožňuje použitie a integráciu modelu do osobných aplikácií. Lokálne modely je však pomocou dedikovaných aplikácií práve možné spúšťať ako lokálny serveru. To zároveň znižuje latenciu na minimum a tým aj rýchlosť komunikácie medzi aplikáciou a modelom.

Prispôsobivosť: Lokálnych modelov je na výber veľké množstvo, a ide väčšinou o voľne šíriteľné modely, často aktívne spravované komunitou. Preto je možné vybrať, si model podľa zamerania, pamäťovej, alebo výkonnostnej náročnosti. Rozsiahle dodatočné úpravy vo forme doplnkov a knižníc však poskytuje aj softvér, nad ktorým sa tieto modely spúšťajú. V konečnom dôsledku je tieto modely možné voľne rozširovať a trénovať použitím vlastných datasetov.

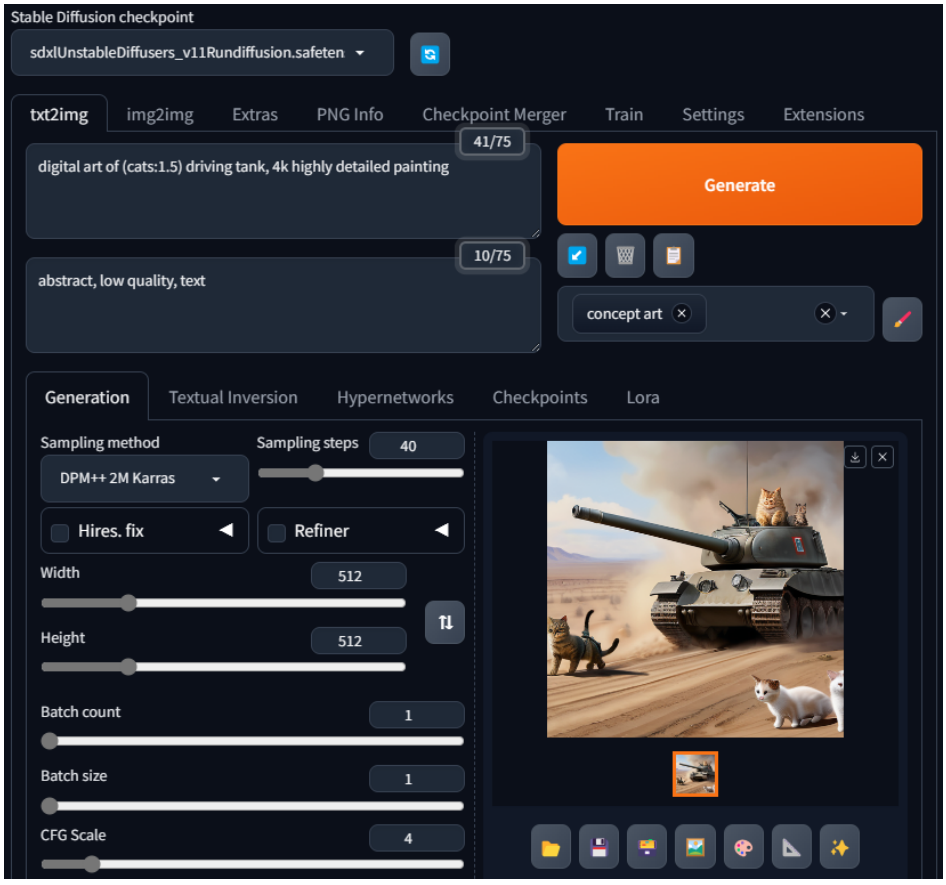
Plná kontrola: Je logické, že sa spoločnosti, ktoré poskytujú prístup ku cloudovým generatívnym modelom, snažia regulovať a cenzurovať výstupy, ktoré tieto AI produkuje. Z bezpečnostných aj etických dôvodov dáva zmysel, aby komerčne dostupné nástroje boli pre užívateľov bezpečné a negenerovali explicitný a nebezpečný obsah, ako napríklad návod na výrobu dynamitu. Použitím lokálnych projektov je, však tieto limitácie možné do určitej miery obísť, a tak v určitých prípadoch spraviť komunikáciu s modelom menej obmedzujúcu.

2 hlavné generatívne AI technológie implementované v hre teda predstavujú Text to Image generátor - Stable Diffusion, a generátor textu v podobe vybraného chatbota.

12 STABLE DIFFUSION WEBUI

Na to aby bolo vo vývoji a aj počas hrania hry použiť AI model, ktorý generuje obrázky je najprv potrebné mať program, ktorý s týmto modelom dokáže komunikovať. Na tento účel už vzniklo množstvo komunitných projektov, každý so svojím jedinečným zameraním.

Projekty ako **InvokeAI** alebo **ComfyUI** ponúkajú výkonné webové rozhranie s pozoruhodnými funkciami outpainting, inpainting, unified canvas či možnosťou priraďovať rôzne prompty a váhy konkrétnym oblastiam obrázka. Taktiež vo webovom rozhraní umožňujú použiť node based systém, v ktorom jednotlivé parametre generovania predstavujú uzly, ktoré je možné medzi sebou spájať a kombinovať. Projekty **Stable Diffusion WebUI** alebo **Stable Diffusion WebUI Forge** sú zase komplexné platformy s robustným webovým rozhraním. Medzi najzaujímavejšie vlastnosti patrí možnosť výberu rôznych vzorkovacích metód, mód img2img, inpainting, upscaling, custom skripty alebo veľkú databázu doplnkov. Podstatné je, že tieto projekty ponúkajú možnosť hosťovania lokálneho serveru. To je kľúčový prvok, ktorý umožní pomocou API komunikácie tento program prepojiť s hrou.



Obrázek 1. Domovská stránka SD Webui [50]

Pre tento projekt som sa rozhodol použiť Stable Diffusion WebUI od autora AUTOMATIC1111. Ide o webové rozhranie pre lokálne hosťovanie Stable Diffusion modelov, implementované pomocou knižnice Gradio v jazyku Python. Tento projekt je jeden z najpopulárnejších v tejto oblasti a obrovské množstvo nastaví, ktoré ponúka je ideálne na experimentovanie. Konkrétne generatívne modely založené na rôznych verziách StableDiffusion je možné sťahovať, importovať z databáz ako Civitai, HuggingFace, Stability-AI alebo priamo z GitHubu jednotlivých vývojárov. [46] [50] [51] [52]

12.1 Stable Diffusion WebUI API

Aby sa API Stable Diffusion dalo používať, je najprv potrebné nastaviť správne príkazy pre spustenie v príkazovom riadku pri spúšťaní programu. Pre aktiváciu API je potrebné nastaviť paramter *–**api***. Tak sa sprístupní komunikácia s FastAPI serverom aj mimo hlavnú webovú stránku. Samotné API obsahuje veľké množstov endpointov ako napríklad:

```
POST /sdapi/v1/text2img
POST /sdapi/v1/img2img
POST /sdapi/v1/png-info
POST /sdapi/v1/options
GET  /config/
GET  /info/
```

Hlavné endpointy pre generovanie sú však /**text2img** a /**img2img**. Endpoint text2img slúži na tvorbu úplne nových obrázkov z textového promptu, img2img zase na generovanie obrázkov na základe preddefinovaného obrázku. Oba z týchto endpointov potrebujú správu od užívateľa vo forme JSONu, kde sú špecifikované údaje, o tom čo chceme generovať. Ďalej sa projekt bude zameriavať výhradne na používanie **text2img**. Väčšinu parametrov, ktoré je možné nastavovať pre generovaní obrázkov na webe je možné použiť aj pri komunikácií s API. Najpoužívanejšie parametre request JSONu bývajú:

```
prompt
negative_prompt
styles
seed
batch_size
steps
cfg_scale
width
height
tiling
enable_hr
hr_upscale
```

Týmito prametrami sa dá špecifikovať popis obrázku aký chceme vygenerovať, popis toho čo obrázok nemá obsahovať, hodnotu ako prísne má generátor nasledovať opis

obrázku, počet generovaných obrázkov, ich rozlíšenie, možnosti upscalingu alebo počet iterácií v ktorých sa obrázky majú formovať. Parametrov môžeme v jednom requeste volať koľko potrebujeme, alebo naopak môžeme špecifikovať iba samotný prompt. Nešpecifikované parametre ostanú nastavené na prednastavené hodnoty. Čím viac informácií ku generovaním však poskytneme, tým lepšie vie model, aký výstup má vytvoriť. [50]

Príklad základného promptu na adresu **http://localhost:7860/sdapi/v1/text2img** môže vyzerať takto:

```
{
  "prompt": "A band of astronauts playing the biggest (rock
   ↪ concert:1.5) on the Moon, crazy vibrant colorful space
   ↪ opera, high resolution, correct lightning, proper shadows,
   ↪ 8k",
  "negative_prompt": "wrong lightning, low resolution, noise,
   ↪ low detail, blur, out of frame, boring, lame",
  "batch_size": 4,
  "steps": 40,
  "cfg_scale": 7,
  "width": 512,
  "height": 512
}
```

Tento request pri generovaní použije aktívne zvolený model na hlavnej stránke webového rozhrania Stable Diffusion. Odpoveďou na request je JSON objekt, ktorý obsahuje pole vygenerovaných obrázkov v textovom formáte base64. Z tejto správy je jednoducho možné vo väčšine programovacích jazykov obrázky konvertovať do binárneho formátu jpg alebo png.

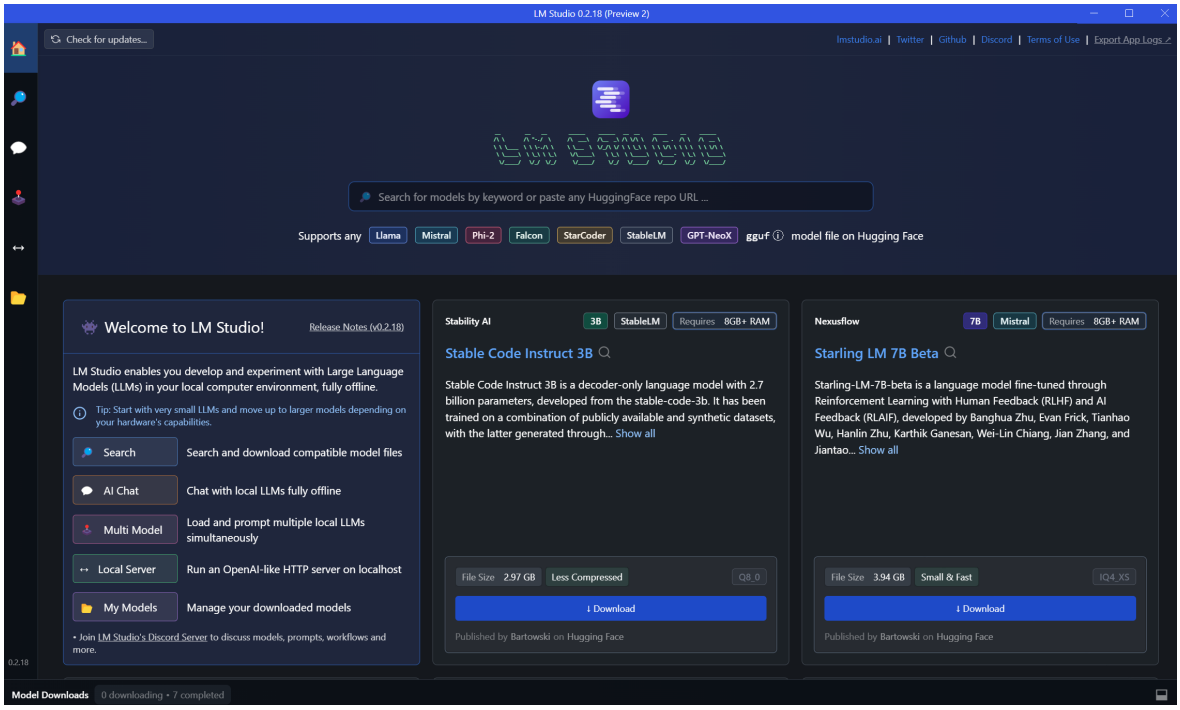
Obrázek 2. Príklad výstupu zo Stable Diffusion [53]

12.2 Stable Diffusion-1.5

Model použitý v tomto projekte je Stable Diffusion verzie 1.5. Tento model je možné používať ako vo verzií rozhrania online API, kedy je generovanie spracovávané na strane serveru tretej strany, tak aj na lokálne, na užívateľskom počítači za použitia hardvéru daného stroja. Tým sa celý image generátor stáva plne nastaviteľným a spravovateľným konkrétnym užívateľom. Tento prístup dovoľuje v podstate neobmedzené generovanie obsahu, bez potreby zriaďovania účtu na webe, čakania vo fronte pri preťažení serveru, alebo používania kreditov či predplatného, ktoré určuje napríklad limit generovaných obrázkov na daný mesiac. Taktiež odpadá akékoľvek obmedzenie cenzúrou, alebo kontrola a zber informácií treťou stranou. Tento prístup tiež umožňuje použite použitie rozsiahleho množstva open source nástrojov a modelov, ktoré sú vyvíjané komunitou.

Dôvodom pre tento model je aj to, že ostatné testované modely neboli schopné vytvárať použiteľné herné textúry. Primárne voľba **tiling** niektoré modely zmetie natoľko, že výstupný obrázok je iba neforemná zmes farieb a abstraktných tvarov. Taktiež oproti modelom Stable Diffusion XL je generovanie približne 2x rýchlejšie pri rovnako nastavených parametroch v prostredí mimo hry. Pri testoch generovania obrázkov modelom Stable Diffusion XL trvalo generovnaie v rádoch niekoľkých minút, čo je pre projet nerpijateľne dlhý čas. [46] [53]

13 LM STUDIO



Obrázek 3. LM Studio Home screen [55]

LM Studio je jedna z mnohých platforiem, ktorá umožňuje spúšťať Veľké jazykové modely offline na lokálnom počítači. Táto aplikácia poskytuje užívateľsky prívetivé rozhranie na sťahovanie, konfiguráciu a rôzne formy používania modelov. Modely je možné jednoducho manažovať a sťahovať z online zdrojov, ako je Hugging Face. S týmito modelmi sa dá následne komunikovať pomocou klasického chatovacieho rozhrania alebo prostredníctvom lokálneho servera je tieto modely možné hosťovať a komunikovať s nimi pomocou API.

Výhoda je aj prispôsobivosť konkrétnych modelov. Online modely ako ChatGPT, Google Gemini alebo Bing Copilot sú často všestranne zamerané aby vyhovovali či najväčšiemu množstvu ľudí a neposkytujú tak pokročilé možnosti prispôsobenia. Mnohé lokálne jazykové modely sú práve naopak prispôsobené na konkrétne použitie. Tak isto je konkrétny model možné komplexne upravovať, a prispôsobovať tak mieru náhodnosti textových výstupov, dĺžku správ, penalizáciu z opakovanie rovnakých výstupov alebo rozloženie výkonu medzi CPU a GPU. V neposlednom rade je možné vytvárať rôzne užívateľské šablóny a formáty, v akým má chatbot generovať správy.

Výhodou LM Studia je aj súkromie. Často informácie, ktoré sú s cloudovími chatbotmi zdieľané, môžu byť citlivé, a nemusí byť isté, akým spôsobom sa s týmito informáciami ďalej pracuje. To sa môže týkať osobných, pracovných alebo korporátnych údajov, alebo aj programátorského kódu. Tieto dáta môžu byť použité dokonca použí-

vané na trénovanie ďalších verzií jazykových modelom. Lokálnym spustením jazykového modelu však tieto problémy odpadajú. [46] [54]

14 LLM STUDIO SERVER

LM Studio má viaceré rozhranie pre rôzne potreby práce s jazykovými modelmi. Priamo v aplikácií je možné prezerať a sťahovať jednotlivé modely z online zdrojov, v sekcií AI Chat sa s týmito modelmi dá štandardne rozprávať formou chatovacieho okna, alebo v sekcií multimodálneho playgroundu je možné načítať viacero modelov naraz a komunikovať so všetkými zároveň.

Podstatné je však rozhranie lokálneho serveru. Ide o jednoduché spustenie tejto aplikácie v móde API serveru na localhost adrese. Komunikácia s týmto API je navrhnutá podľa rozhrania OpenAI API. [55] [56]

14.1 Komunikácia s API

Dostupné sú teda endpointy:

```
GET /v1/models
POST /v1/chat/completions
POST /v1/embeddings
POST /v1/completions
```

Na komunikáciu s jazykovým modelom z prostredia hry, však postačí endpoint `v1/chat/completions`. Pre zahájenie komunikácie s modelom touto cestou, stačí odteraz posielať POST requesty na túto adresu.
Celá adresa requestu: **http://localhost:7861/v1/chat/completions**

Telo tohoto requestu musí obsahovať užívateľský prompt, ktorý predstavuje správu od užívateľa ku modelu, tak ako vo webovom rozhraní bežných chatbotov. Telo POST requestu má formu JSON objektu, kde je možné špecifikovať rôzne parametre OpenAI API rozhrania. Príklad základnej správy:

```
{
  "messages": [
    {
      "role": "system",
      "content": "You're a helpful coding assistant."
    },
    {
      "role": "user",
      "content": "How to get out of git merge conflict hell?"
    }
  ],
  "temperature": 0.7,
  "max_tokens": -1,
  "stream": true
}
```

Použitím `"role": "system"` sa dá bližšie špecifikovať rola, akú má chatbot plniť. `"role": "user"` predstavuje konečnú správu užívateľa.

Celkové parametre tohoto JSONu sú:

```
model
top_p
top_k
messages
temperature
max_tokens
stream
stop
presence_penalty
frequency_penalty
logit_bias
repeat_penalty
seed
```

Takýmto spôsobom sa dá do veľkej miery kontrolovať a korigovať aktívna komunikácia. Jednotlivé parametre môžu ovplyvňovať požadovanú dĺžku chatbotových odpovedí, diverzitu alebo náhodnosť týchto odpovedí, ukladať odpovede do listu pre uchovanie celkovej komunikácie, alebo udeľovať penalizáciu za časté opakovanie sa, či používanie rovnakých fráz dokola. Odpoveďou jazykového modelu na tento request je ďalší JSON, z ktorého je možné vyextrahovať finálnu chatbotovu odpoveď. Reťazením užívateľských správ a chatbotových odpovedí do nových requestov je možné počas rozprávania sa s charbotom udržovať celý kontext od začiatku komunikácie.

14.2 Jazykové modely

Počas vývoja hry boli testované viaceré modely, z dôvodu, že žiadny nebol univerzálne použiteľný, alebo funkčný na prvý pokus. Prevažne je však bola hra testovaná s týmito modelmi:

14.2.1 Llama

Large Language Model Meta AI (LLAMA) je séria open-source jazykových modelov vydaných spoločnosťou Meta. Tieto modely sú v rozsahu 7B, 13B až 70B parametrov a rôznych prispôsobených verziách. Llama-2-Chat je LLM model vyladený pre dialógovú komuniáciu. Code Llama je model špeciálne prispôsobený na písanie programátorského kódu, buď z prirodzeného jazyka, alebo z už existujúceho kódu. [57]

14.2.2 Mistral-7B

Mistral je to všestranný model navrhnutý tímom Mistralai, na generovanie textu pre aplikácie v reálnom svete. Verzia 7B v0.1, so 7 miliardami parametrov je navrhnutá pre vysoký výkon a efektivitu. Z tohto základného modelu vznikli viaceré varianty ako inštrukčne trénovaný Mistral-7B-Instruct-v0.1, alebo vysoko účinná verzia Mixtral-8x7B-v0.1, ktorá vďaka svojej optimalizácií, patrí medzi popredné open-source modely. [58]

14.2.3 Zephyr-7B

Zephyr je séria jazykových modelov, ktoré sú trénované na to, aby fungovali ako pomocní asistenti. Zephyr-7B-a a Zephyr-7B-b sú prvé modely v sérii a ide o jemne ladenú verziu modelu Mistral-7B-v0.1. Tento model bol dolaďovaný na variante datasetu, ktorý obsahuje veľké množstvo syntetických dialógov generovaných pomocou ChatGPT a hodnotených modelom GPT-4. [59]

14.2.4 Gemma

Gemma je skupinka ľahkých a bezpečných modelov vydaných spoločnosťou Google. Tieto modely sú založené na rovnakom výskume aj technológiách použitých pri vytváraní modelov Gemini. Tento predtrénovaný model je dostupný ako Gemma 2B a Gemma 7B. Vďaka svojej veľkosti sú tieto modely jednoducho použiteľné aj na slabších zariadeniach ako sú notebooky alebo bežné počítače. [60]

14.2.5 ChatGPT

ChatGPT je jeden z prvých komerčne dostupných veľkých jazykových modelov, vydaný spoločnosťou OpenAI, ktorý v roku 2022, od kedy bol sprístupnený širokej verejnosti, odštartoval AI závody naprieč spoločnosťami a IT sektorom po celom svete. Jeho najpoužívanejší model GPT-3.5, ktorý má približne 175B parametrov, bol trénovaný na 570 GB textových údajov z celého internetu, ako knihy, články, webové stránky alebo sociálne médiá. ChatGPT prešiel od svojho vzniku mnohými vylepšeniami, pričom najnovší model GPT-4 dokáže pracovať v multimodálnom režime. To znamená že je schopný generovať a zároveň rozumieť textovému, obrázkovému, aj zvukovému vstupu. Aktuálne je model GPT-3.5 dostupný ako cloudová služba, preto je v projekte použitý iba pri testovaní mimo hry. [44] [56]

15 DESIGN HRY

Hlavným cieľom tohto projektu je otestovať a zistiť, aké miesto môže mať v hrách generatívna umelá inteligencia. Vďaka svojej univerzálnosti sú rôzne úlohy, ktoré dokáže plniť. Najprv je potrebné špecifikovať čo všetko v kontexte hier dokážeme pomocou nej tvoriť. Akákoľvek hra je spracovaná graficky, prvá voľba je teda použitie generovania grafiky. To je možné aplikovať ako v 2D tak v 3D hrách. Máloktorá hra sa zaobíde bez textu, vo forme dialógov, užívateľského rozhrania, alebo naratívnu. Tu je priestor pre skúmanie možností chatbotov. V prípade 3D hier je možnosť použitia aj pokročilejších modelov na generovanie 3D objektov, dostupnosť týchto modelov v čase písania tejto práce je veľmi obmedzený. Táto technológia je zatiaľ vo vývojovej fáze, taktiež kvalita výstupov zatiaľ nie je pre väčšinu aplikácií ako napríklad hry úplne postačujúca.

V rámci tohto projektu som sa preto rozhodol otestovať možnosti ako jazykových modelov, tak obrázkových modelov. Ku týmto typom AI existuje najviac voľne dostupných zdrojov, čo tento projekt robí pomerne flexibilným. Prvý krok je vytvoriť nápad na hru.

15.1 Námet

Ako teda takúto hru správne nadizajnovať? Generatívna AI je dobrá vo vytváraní rozmanitého obsahu pomerne veľkou rýchlosťou. Hra by preto mala byť modelovaná tak, aby bolo túto vlastnosť bolo možné jednoducho aplikovať a aktívne ju využívať. Ponúka sa teda mechaniky hry postaviť na štýle Rouglike alebo Endless Survival. Štýl kde hráč preskúmava, bojuje, alebo sa snaží prežiť v hernom svete, ktorý sa naprieč hrou neustále vyvíja, alebo mení pri každom hraní hry. Nepriatelia, spojenecké postavy, objekty v hre alebo herná mapa sa môže naprieč hrami meniť tak isto. Tento žáner je síce menej zameraný na striktne daný príbeh a naratív, a viac na akciu a neočakávané situácie. Toto sa javí ako jeden z hlavných námetov, kde by sa dá generatívna AI aktívne používať počas hry. Avšak pri hoci akom inom type hry by si svoje využitie našla tiež.

15.2 Overview

Príbeh a zasadenie hry je teda následovné. Hráčova postava v hre je Mechabot. To je jednoduchý štvornohý robot vybavený dvomi zbraňami. Na výber má zo siedmich zbraní, ktoré je možné vyberať pri postupe úrovňami. Jeho úlohou je jednoducho prežiť vlny útočiacich nepriateľov. Hra sa odohráva na mape v podobe štvorcovej arény. V každej novej vlne je rozloženie mapy náhodne pozmenené, a objaví sa na nej určitý počet nepriateľských postáv modelovaných rovnakým spôsobom ako hráč. Títo Mechaboti sa rovnako ako hráč skladajú z nôh, trupu a zbraní. Každý z týchto komponentov

má viacero variánt, s rozdielnymi parametrami ako rýchlosť, výdrž alebo rozsah poškodenia. Tak sa dajú ovplyvňovať parametre a správanie jednotlivých postáv. Hráč môže postúpiť do ďalšieho levelu buď zničením všetkých nepriateľov, alebo obsadením strategických bodov na mape. Prehrá naopak ak je v boji zničený, alebo ak sa nepriateľom podarí obsadiť všetky body skôr.

Obrázek 4. Player Character

15.3 Herné mechaniky

Do tohto scenáru vstupuje AI spôsobom, že v každej novej vlne sú textúry jednotlivých Mechabotov vygenerované pomocou Stable Diffusion. Postavy teda môžu mať akúkoľvek textúru od púštnej digitálne kamufláže, až po cukrovú polevu. V tomto smere je kreativita v podstate neobmedzená. Rovnaké generovanie textúr je možné použiť na jednotlivé elementy mapy, alebo dokonca na samotný Skybox (textúra pozadia, alebo 3D prostredie, ktoré obklopuje celú hernú scénu). Nechať však AI úplnú voľnosť v tomto smere môže viesť ku generovaniu vizuálne nezmyselnej grafiky. Rôzne skripty a funkcie preto riešia ako tomu zabrániť.

Mechaboti, ktorý sa na hráča vrhnú na začiatku každej novej vlny sú z jednotlivých komponentov vždy poskladaný nanovo. To z akých dielov sa budú skladať je kombinatorický problém, ktorý je možné riešiť rôznymi rozhodovacími algoritmami, avšak v tomto prípade sú dané blueprinty (zoznam komponentov pre konkrétneho Mechabota) tvorené pomocou jazykového modelu. Je teda na chatbotovi, akých nepriateľov proti hráčovi pošle. Keď sa však nepriatelia objavia na mape, majú rôzne možnosti, ako sa ďalej zachovať. O tom či zaútočia na hráča a akým spôsobom, alebo či sa vrhnú na obsadzovanie bodov je tiež možné serializovať ako zoznamu úloh, ktoré môže generovať

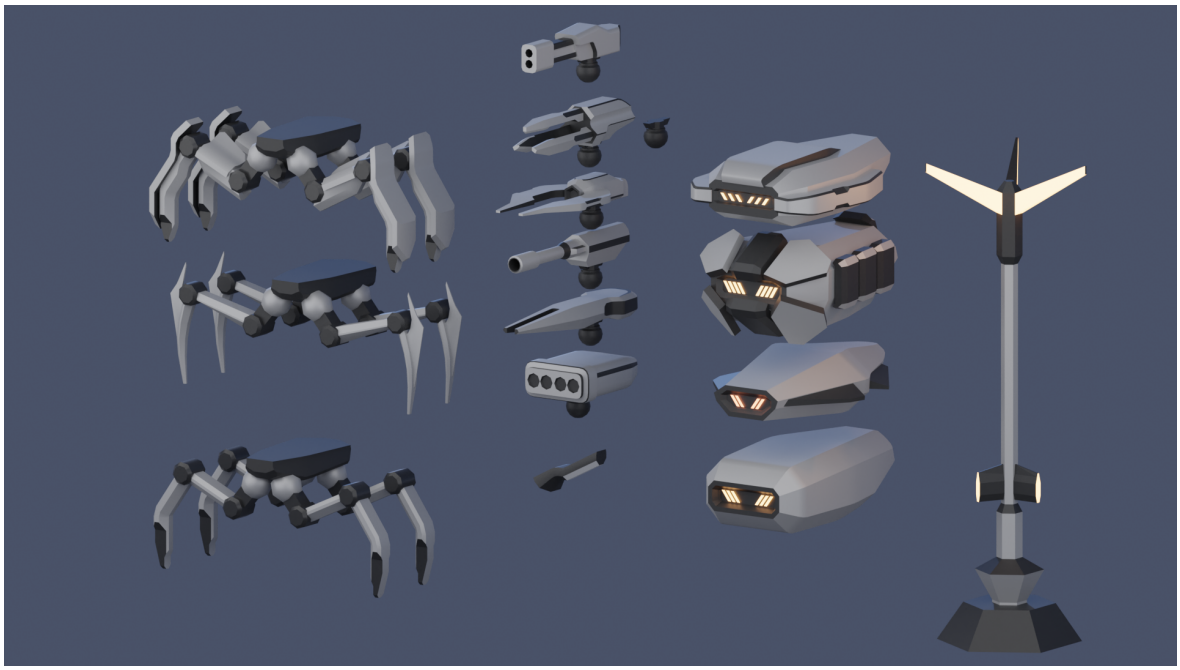
chatbot. Z väčšej časti je teda výzor aj správanie nepriateľov riadené generatívnou AI. V konečnom dôsledku, hráč vlastne nepriamo bojuje proti chatbotovi, ktorý s sa mu snaží pokaziť deň.

Možným vylepšením toho konceptu je aj prepojenie chatbota so Stable Diffusion, kedy jednotlivé nápady na textúry nepriateľov alebo vzhľad mapy môžu byť založené na výstupoch z jazykového modelu. Tento prístup predstavuje jednu z mnohých implementačných výziev, ktoré v tomto projekte vznikajú. Implementácie jednotlivých algoritmov a integrácia AI je podrobnejšie rozpísaná v ďalších kapitolách. [61]

16 VIZUÁLNY ŠTÝL

16.1 Modely

3D modely pre postavu hráča, Mechabotov, to je nohy, trupy, zbrane, projektily, ale ostatné objekty na mape, sú takmer všetky modelované ručne, pomocou programu Blender. Modely potrebujú mať jednak konzistentný vizuálny štýl, tak aj správne rozmery a tvary. Mechaboti v hre sú dynamicky vyskladaný z jednotlivých komponentov, preto musia tieto diely pasovať jeden na druhý. To znamená, že na hoci ktorý trup, musí byť možné pripojiť hocijakú zbraň, alebo nohy, tak aby v hre tento model vyzeral prirodzene. V prostredí Unity je z týchto modelov možné vytvoriť Prefaby (znovupoužteľné herné objekty v prostredí Unity), a dodatočne upravovať ich parametre v závislosti od hry.



Obrázek 5. Herné modely

Čo sa týka modelov tvoriacich herné prostredie, je modelovanie a príprava značne jednoduchšia. Mapu tvorí veľmi jednoduchá štvorcová aréna. Táto voľba plynie z komplikovaného ovládania Mechabotov. Tých ovláda navigačný systém Unity z balíku AI Navigation. Na mape je komponent NavMeshSurface a Mechaboti, teda agenti, majú komponent NavMeshAgent. Táto navigácia funguje spôsobom, že každý agent má cieľ kam sa chce dostať. Na pozadí NavMeshSystem prepočítava pomocou pathfindingu cesty, po ktorých sa agenti ku týmto cieľom môžu dostať. To je trochu limitujúci prístup, pretože agenti nemôžu skákať, alebo robiť príliš akčné manévry, bez implementácie komplexnej logiky ovládania. Na demonštračné účely však takáto navigácia stačí.

16.2 Materiály

Príprava materiálov je takisto podstatná časť. Keď z generátoru obrázkov dostaneme nové textúry, táto akcia sa deje počas hrania hry, kedy sú minimálne možnosti ako textúry upravovať, či ručne mapovať. Modely preto potrebujú mať správne vygenerované UV mapy, aby textúry 'nalepené' na ne vizuálne pasovali. Inak sa môže stať, že budú textúry neprirodzene skreslené. Dôležité je aj poradie materiálov na objekte.

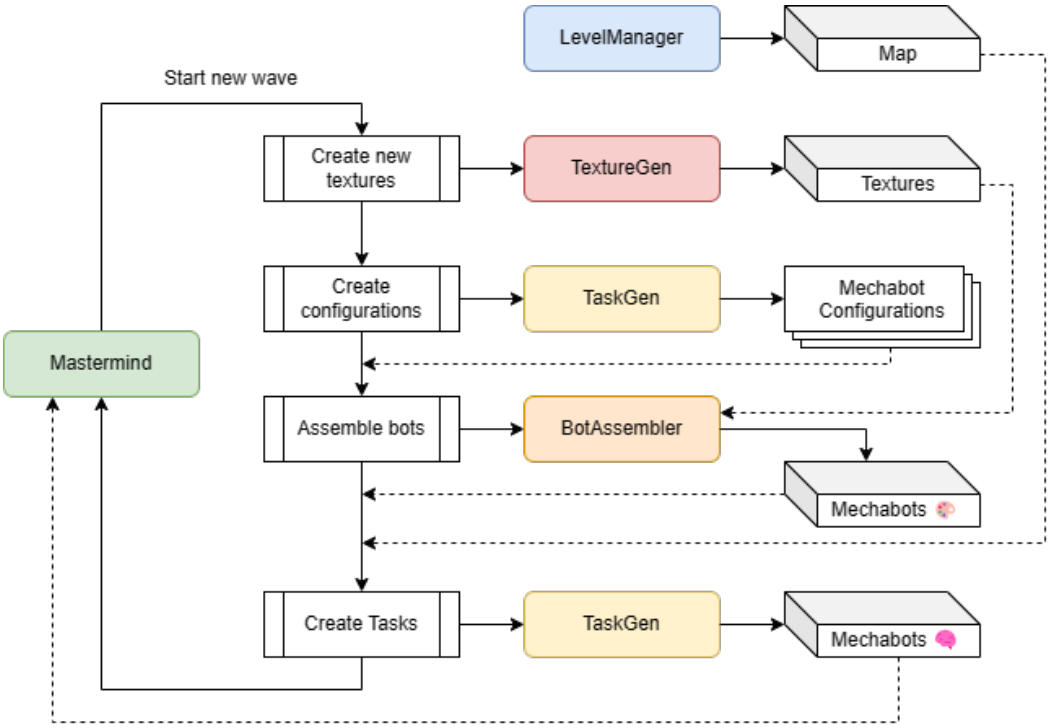
Každý objekt má minimálne 2 materiály, jeden hlavný, ktorý pokrýva väčšinu plochy modelu a druhý podkladový tmavo šedý. Hlavný materiál je počas hry prekresľovaný textúrami vygenerovanými AI modelom, jeho farba v editore teda nie je podstatná. Sekundárny tmavý je použitý ako podklad a na zvýraznenie detailov. Niektoré modely majú tretí materiál, ktorý je čisto emisný, použitý ako signalizačný bod na obsadzovacích bodoch alebo ako oči Mechabotov.

Obrázek 6. Materiály

V tomto prípade stačí nastaviť **Material 1** ako prvý v zozname materiálov daného objektu, a herné skripty podľa tohoto indexu budú presne vedieť, ktorý materiál majú meniť. Táto mechanika je implementovaná v kapitole 22 BotAssembler.

17 INTEGRÁCIA GENERATÍVNEJ AI

Táto kapitola sa zameriava primárne na popis a fungovanie skriptov, ktoré riešia AI logiku hry. Hra komunikuje s dvomi rozdielnymi AI modelmi zároveň, veľmi podobným spôsobom. Každý model má v hre priradený vlastný skript, z dôvodu vyššej flexibility a rozšíriteľnosti. Tieto skripty však sami o sebe iba komunikujú s nejakým API a vracajú textovú odpoveď. Do hry ich zapája až ďalší skript, ktorý riadi ich činnosť a zodpovedá za preklad ich odpovedí do použiteľnej podoby. Následne tieto dáta použije iný skript pri tvorbe nových nepriateľov v každej novej úrovni hry. Zjednodušená komunikácia medzi AI zameranými tými skriptami, je zobrazená na obrázku 7..



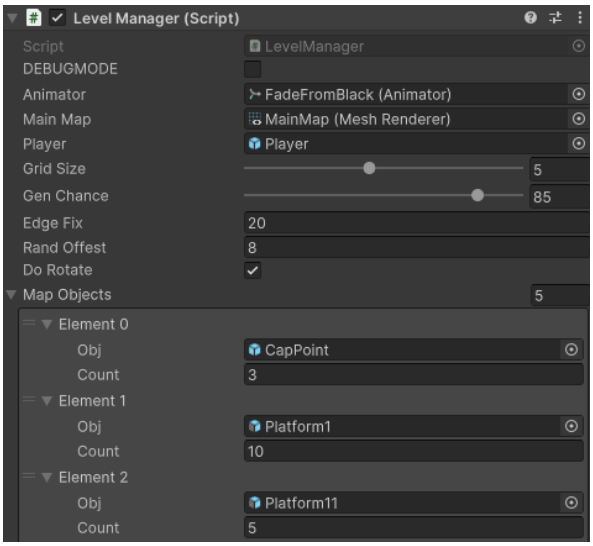
Obrázek 7. Tok operácií

Celý tento proces je v hre reprezentovaný, ako vytvorenie novej vlny nepriateľov, ktorí útočia na hráča. Keďže komunikácia medzi týmito programovými triedami je pomerne zložitá, sú tieto triedy vysvetlené samostatne, spolu s možnosťami ich optimalizácie aj možnými problémami.

18 LEVELMANAGER

Trieda LevelManager je zodpovedná za riadenie generovania a umiestňovania objektov v rámci hernej úrovne. Sama o sebe neimplementuje generatívnu AI, avšak jej prvky sú použité pri komunikácií s AI modelmi. Hlavná funkcionalita triedy zahŕňa generovanie a umiestňovanie objektov na mapu, spracovávanie udalostí smrti hráča a správnu **NavMeshSurface** objektu, zodpovedného za riadenie AI navigácie v hre.

Hlavná metóda **GenerateMapItems** generuje a umiestňuje na mapu objekty definované v zozname objektov **MapObjectEntry**. Tento zoznam obsahuje modely jednotlivých objektov a ich maximálny počet udávajúci, koľko objektov daného typu sa môže na mape objaviť. Objekty sa vytvárajú na náhodných pozíciách v rámci hraníc mapy s voliteľnou rotáciou, pravdepodobnosťou objavenia, alebo náhodnosti usporiadania. Toto generovanie je prispôsobiteľné na základe súboru pravidiel a parametrov upravovateľných v Unity Editore. Jednotlivé parametre je možné vidieť na Obrázku 8.



Obrázek 8. Perametre LevelManageru

Medzi možné objekty v zozname patria aj strategické body. Tie slúžia ako objekty, ktoré môže hráč alebo Mechaboti obsadzovať. Strategickým bodom, ktoré sa na mape objavia sa automatický priradí značka (A, B, C, atď.), ktorá jednoznačne identifikuje daný bod. Údaje o týchto bodoch sa následne dajú použiť pri generovaní dát jazykovým modelom.

19 TASKGEN

Toto je skript, ktorý na pozadí hry komunikuje pomocou REST API s aplikáciu, nad ktorou beží nejaký veľký jazykový model, v tomto prípade LM Studiom. Výstupy tohto modelu sú použité pre tvorbu konfigurácií pre skladanie Mechabotov aj pre tvorbu ich úloh.

19.1 Vstupné parametre

Trieda pomocou C# objektu HttpClient posiela requesty na API LM Studio serveru. Hlavnými vstupnými parametrami pre túto triedu sú:

- adresa nášho lokálneho serveru: `http://127.0.0.1:7861`
- komunikačný endpoint: `/v1/chat/completions`
- telo requestu (rola AI modelu, užívateľský prompt, ...)

19.2 Telo requestu

Podľa komunikačnej schémy LM Studio serveru je telo requestu definované ako:

```
1  private struct Payload
2  {
3      public Messages[] messages;
4      public float temperature;
5      public int max_tokens;
6      public bool stream;
7  }
8
9  private struct Messages
10 {
11     public string role;
12     public string content;
13 }
```

Pri volaní akéhokoľvek requestu stačí v podstate meniť iba hodnotu **role** a **content**. Tie udávajú, ako sa má model chovať, a čo chceme aby generoval. Takto by mal jazykový model pochopiť, ako ho chceme využívať, a taktiež formulovať svoje odpovede v špecifickom formáte. Bez tohto bodu model takmer vôbec nevygeneruje odpoveď, ktorú bude možné v hre užitočne spracovať. Konkrétna rola závisí od toho čo chceme v hre generovať, preto môže v iných hrách vyzerať úplne inak.

Pred vyvolaním requestu je potrebné dátovú **Payload** štruktúru serializovať do JSONu aby ju mohla trieda **HttpClient** použiť ako telo requestu. Na to je použitá trieda **Newtonsoft.Json**. Keď je všetko pripravené stačí request vyslať v asynchrónnej metóde a počkať na odpoveď serveru. Návratový objekt bude znovu JSON, s rôznymi objektami. Odpoveď chatbota na užívateľský prompt je v objekte **choices** > **0** > **messages** > **content**.

```
string result = await response.Content.ReadAsStringAsync();
var jObject = Newtonsoft.Json.Linq.JObject.Parse(result);
string output = jObject["choices"][0]["message"]["content"].ToString
    ();
```

Takto sa dá extrahovať chatbotová odpoveď do premennej **string output**. Odteraz ju môžu ďalšie skripty používať a pracovať s ňou podľa potreby.

19.3 Použitie

Jednotlivé prompty, smerované na AI model sa môžu mierne líšiť v závislosti od toho s akým modelom komunikujeme. Jednoduchý model (do veľkosti 7B) má pochopiteľne väčšie ťažkosti porozumieť príkazu, ako komplexný model typu ChatGPT (100B a viac parametrov). Pre jednoduchosť implementácie by však prompt mal byť čo najviac univerzálny, teda napísaný tak aby sa dal použiť s akýmkoľvek AI modelom.

V našom prípade vyzerá **role** parameter pre AI model približne takto: ***You are game AI assistant, you generate text only in specified format by user, nothing more*** Toto nastavenie by malo zaistiť že model nebude vytvárať zbytočný text navyše, čo by predlžovalo čas generovania aj zložitosť prekladu tejto odpovede. Toto nastavenie môžeme použiť vo všetkých ďalších požiadavkách, ktoré budeme na jazykový model smerovať.

19.3.1 Generovanie konfigurácií

Čo sa týka tvorby nepriateľov v hre, je viac možností ako vytvoriť prompt pre tvorbu Mechabotov AI modelom. Tak ako máme definované jednotlivé modely v BotAssembler skripte (základné komponenty: nohy, trup, zbrane), musíme ich definovať aj v prompte, aby AI model vedel s čím má pracovať. Modelu môžeme povedať, koľko nepriateľov chceme, z akých komponentov sa majú skladať a hlavne formát v akom má odpoveď vytvoriť. Po mnohonásobnom upravovaní a testovaní sa konfiguračný prompt dostal do takejto podoby:

```
You are a game AI responsible for configuring characters designed to
attack players.
Each character consists of three components: legs, weapons, and hulls,
each with multiple models identified by numbers.
The ranges for each component are as follows:

Legs: IDs range from 0 to 3 (higher number means faster movement)
Weapons: IDs range from 0 to 5 (higher number means bigger fire power)
Hulls: IDs range from 0 to 4 (higher number means more health)

Now generate 10 configurations using the following format: 'B# : L# W# H
#,' (letters must be there) where:

B# represents the ID of the character (ranging from 0 to 9)
```

```
L indicates the leg model, W indicates the weapon model, and H indicates
the hull model, each with an ID within its respective range mentioned
above.
Ensure that configurations stay within the valid model ID ranges and
always follow the specified format, include a comma after each
configuration.
Do not generate other text than the specified format.
```

Táto finálna podoba bola vytvorená s miernou pomocou ChatGPT 3.5, ktorý bol tiež použitý ako jeden z testovacích modelov. Čo znamenajú jednotlivé sekcie promptu ?

- Úvodná sekcia má za úlohu letmo oboznámiť jazykový model s pozadím hry a jeho úlohou. Celý prompt by mal byť dostatočne informatívny, aby model vedel čo má generovať, zároveň však tak stručný, aby sa v ňom model nestratil. To môže byť problém hlavne pri menších modeloch.
- Sekcia **Legs, Weapons, Hulls** modelu dáva na výber komponenty, ktoré môže použiť, označené indexom 0 až posledný model (N-1).
- Ďalší odstavec udáva formát v akom má prísť daná odpoveď. V tomto prípade by mala jedna konfigurácia mať tvar **B# : L# W# H#,**
- Posledný odsek je objasnenie jednotlivých častí konfigurácií, spolu s pripomenutím nutnosti dodržať špecifikovaný formát.

Výstupom takéhoto promptu by v ideálnom prípade mal byť reťazec:

B0 : L2 W5 H4, B1 : L0 W3 H2, B2 : L1 W2 H3, B3 : L1 W4 H0, B4 : L2 W2 H3, B5 : L2 W4 H1, B6 : L0 W5 H2, B7 : L2 W3 H1, B8 : L1 W1 H2, B9 : L2 W2 H0, B10 : L0 W4 H3,

Jednotlivé čísla v prompte udávajúce maximálny index daného komponentu, sa v hre nastavuje dynamicky, podľa počtu objektov nastavených v BotAssembler skripte. Aj keby v časti **...Now generate 10 configurations...** nastavíme počet konfigurácií na 100, chatboti nemajú väčšinou problém všetky tieto konfigurácie vygenerovať. S ďalšími časťami promptu je tak isto možné voľne experimentovať. Takýto reťazec je už možné jedonoducho serializovať programovacím jazykom do dátovej štruktúry ako je **List** alebo **Dictionary**. O túto časť sa stará skript **Mastermind**, ktorý je rozoberaný neskôr.

19.3.2 Generovanie príkazov

Keď vieme, akých Mechabotov v danom leveli budeme mať a koľko ich bude, môžeme sa na jazykový model obrátiť znovu. Tento krát chceme aby chatbot rozhodol, aké úlohy majú Mechaboti v hre plniť. Tento proces je mierne zložitejší, keďže chatbotovú

odpoveď budeme musieť pretransformovať do akcií postáv v hre. Jedna z možností je najprv definovať logiku, ktorú majú postavy plniť a následne v hre pre každú akciu vytvoriť obsluhujúcu metódu. Takže každá úloha, ktorú chatbot v zozname vytvorí sa namapuje na nejakú hernú funkciu riadiacu pohyb, streľbu, alebo navigáciu. Prompt pre chatbota v tejto úlohe vyzerá takto:

```
You are the game mastermind AI tasked with assigning tasks to characters
attacking a player. The characters can perform tasks from the following
list:"

Capture point A = CP–A
Capture point B = CP–B
Capture point C = CP–C
Attack player – short range = AP–S
Attack player – min range = AP–M
Attack player – long range = AP–L
Stay idle = SI

Please generate one task for each character in the following format: 'B–#
 : XX–X,'
where # represents the ID of the character ranging from 0 to 9 + @",
and XX–X represents one of the possible tasks from the list above (it's
short version).
Ensure to include a comma after each line.
```

V skratke prompt znovu obsahuje:

- Uvedenie chatbota do deja a príkaz vytvoriť zoznam úloh pre herné postavy útočiace na hráča.
- Zoznam možných úloh, ktoré môžu postavy plniť
- Špecifikácia formátu v akom má prísť odpoveď a vysvetlenie jednotlivých častí formátu

Ideálna odpoveď chatbota na tento prompt je:

B-0: CP-A, B-1: CP-B, B-2: CP-C, B-3: AP-S, B-4: AP-M, B-5: AP-L, B-6: SI, B-7: SI, B-8: SI, B-9: SI,

V závislosti od definovaných funkcií, môžeme chatbotovi dať akýkoľvek zoznam úloh o ktorom môže rozhodovať, pričom početnejší zoznam možných úloh prináša zaujímavejšie možnosti akými sa môže hra vyvinúť. V tomto prompte sú pre jednoduchosť zahrnuté iba úlohy rozdielnych typov útoku na hráča a obsadzovanie konkrétnych strategických bodov. Tieto body sa do promptu načítajú dynamicky, v závislosti od vytvorenej mapy.

Takýmto spôsobom je možné implementovať ľubovolnú komunikáciu s jazykovým modelom. V príkaze na generovanie konfigurácií môžeme navyše chatbotovi na konci príkazu dať návrh aby konfigurácie zameriaval na rýchlosť postáv, alebo naopak na maximalizáciu obranyschopnosti. Rovnako tak je možné nabádať generovanie úloh napríklad aby Mechaboti boli agresívnejší a útočili na hráča z blízka, alebo sa naopak sústreďovali na defenzívne obsadzovanie bodov.

20 TEXTUREGEN

Táto trieda modelovaná veľmi podobne ako TaskGen, implementuje komunikáciu s AI modelom Stable Diffusion. Vstupné parametre pre triedu sú už komplexnejšie, pretože máme viac možností, ktoré môžeme upravovať. Kompletný zoznam parametrov posielaných modelu v hre aktuálne vyzerá takto:

```
1 private struct Payload
2 {
3     public string prompt;
4     public string negative_prompt;
5     public int steps;
6     public int batch_size;
7     public int cfg_scale;
8     public int resolution;
9     public bool tiling;
10 }
```

prompt a **negative_prompt** znamenajú čo má a čo nemá generovaný obrázok obsahovať, s tým, že tieto polia môžu obsahovať jedno slovo, až celé vety. Komplexnejší popis v tomto prípade znamená presnejší výstup. **cfg_scale** navyše nastavuje mieru, ako veľmi sa má generátor držať náško príkazu, menšia hodnota znamená viac náhodné výstupy. Parametrom **steps** a **resolution** sa dá nastaviť kvalita a rozlíšenie obrázkov, a nastavením **batch_size** definujeme, koľko obrázkov chceme generovať. V prípade hier bude pravdepodobne veľmi často dôležitý aj parameter **tiling**, ktorý zaisťuje, že generované obrazký budú tvorené ako dlaždice. To znamená, že keď takýto uložíme do nekonečnej mriežky, bude sa jeho vzor do nekonečná opakovať, bez viditeľných hraníc.

Tieto parametere sú nastaviteľné v editore, prípadne by ich bolo možné nastavovať aj počas hrania napríklad v menu. Aj v tomto prípade potrebujeme modelu čo najpresnejšie povedať čo má generovať. Niečo podobné, ako parameter **role** pri komunikácií s jazykovým modelom, však v tomto prípade nie je dostupné. Namiesto toho sa dá jednoducho do užívateľského promptu pridať fixný reťazec údajov, ktoré by mali byť vždy modelom použité.

To znamená že užívateľ môže zadať svoju predstavu textúry: ***"blue and gray naval marine camouflage, multicolore, striking geometric shapes"***. Toto dá modelu jasnú predstavu čo má generovať, avšak je to veľmi stručný opis, z ktorého môžu vznikať značne nepresný výstupy. Preto tento skript definuje dodatočný popisok, ktorý sa pribalí ku základnému návrhy, čím ho spraví na pozadí viac robustným. Doplnkový popisok použitý v hre je : ***", 2D texture, diffuse light, high resolution, (high contrast), game texture, colorfull, big geometric shapes"***.

Rovnakým spôsobom je dôležité nastaviť aj negatívny príkaz, aby sa obmedzila tvorba obrázkov, ktoré obsahujú nechcený obsah. Tento popisok je už rozsiahlejší, keďže rôzne modely často pridávajú do obrázkov rôzny text, čísla, objekty, alebo farby, ktoré s hlavným príkazom nesúvisia. Príklad záporného doplnkového popisku pri tvorbe

herných textúr: ***"carpet, clothing, cartoon, drawing, painting anime, low resolution, bad quality, distortion, blur, objects, out of boundaries, small rsolution, watermark, text, numbers"***.

20.1 Použitie

Finálny JSON, ktorý hra posiela na API generatívneho modelu môže vyzerať takto:

```
{
    "prompt":"blue and gray naval marine camouflage,
   ↪ multicolore, striking geometric shapes, 2D texture,
   ↪ diffuse light, high resolution, (high contrast), game
   ↪ texture, colorfull, big geometric shapes",
    "negative_prompt":"yellow, red, orange, vibrant
   ↪ colorscarpet, clothing, cartoon, drawing, painting, anime,
   ↪ low resolution, bad quality, distortion, blur, objects,
   ↪ out of boundaries, small rsolution",
    "steps":20,
    "batch_size":4,
    "cfg_scale":4,
    "resolution":256,
    "tiling":true
}
```

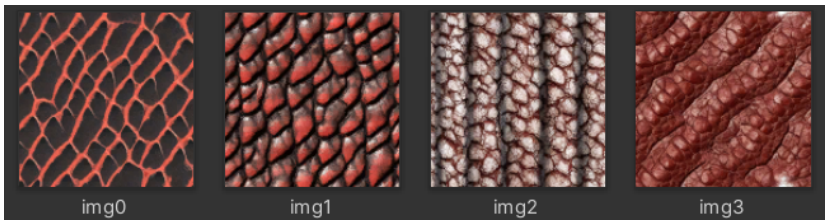
Tento príkaz v skratke definuje, že chceme 4 rôzne textúry s motívom námornej kamufláže s rozlíšením 256*256 pixelov.

20.2 Ukladanie

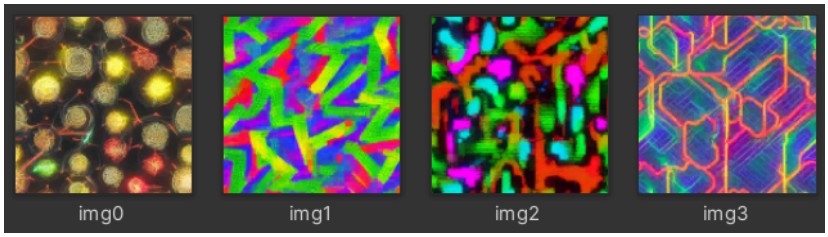
Pri Stable Diffusion je odpoveď modelu JSON objekt obsahujúci obrázky zakódované do formátu **base64**, spolu s rôznymi ďalšími metadátami. Aby boli vygenerované obrázky prístupné prostrediu hry, je ich potrebné konvertovať napríklad do formátu png a uložiť do adresára hry.

```
1  string jsonResponse = await response.Content.ReadAsStringAsync();
2  string[] result = JsonUtility.FromJson<string[]>(jsonResponse);
3  for (int i = 0; i < result.Length; i++)
4  {
5      string base64Image = result[i];
6      byte[] imgBytes = Convert.FromBase64String(base64Image);
7      using (var imageFile = new FileStream(GameData.Instance.
       TexturesPath + "/img" + i + ".png", FileMode.Create))
8      {
9          imageFile.Write(imgBytes, 0, imgBytes.Length);
10         imageFile.Flush();
11     }
12 }
```

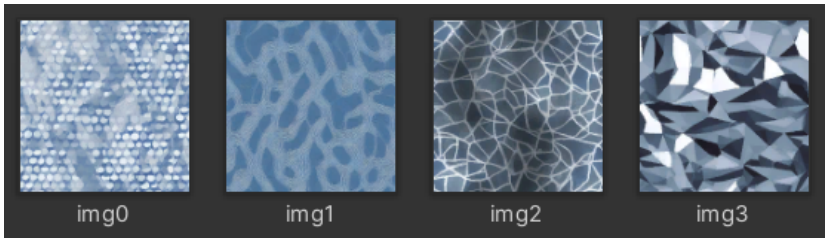
Po tomto bode sú vygenerované obrázky uložené do herného adresára, **TexturesPath** ako **imgX.png**, kde X je index obrázku. Názvy je možné upravovať ľubovoľne, avšak ostatné skripty musia túto pomenovaciu schému dodržiavať. V našom prípade do tejto zložky potrebuje prístup 22 BotAssembler.



Obrázek 9. Príklad vytvorených textúr 1



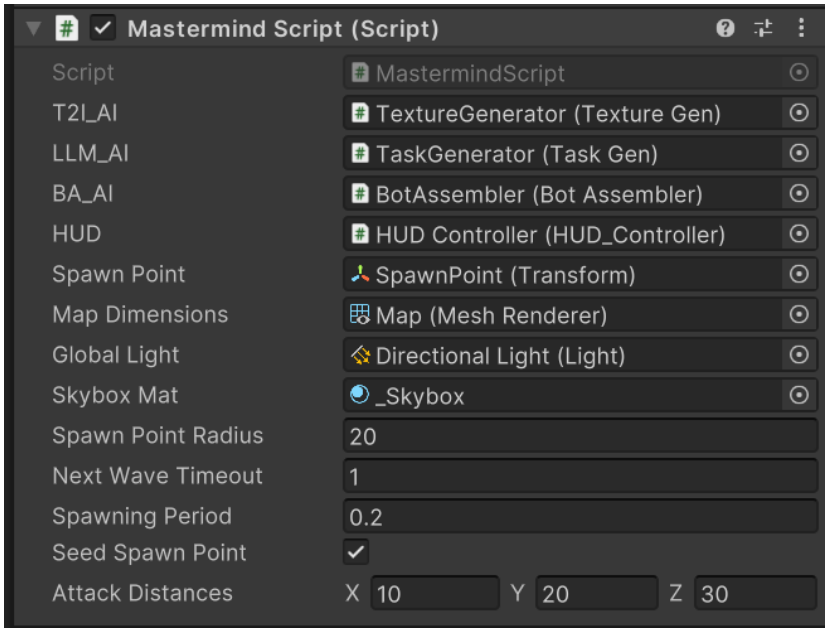
Obrázek 10. Príklad vytvorených textúr 2



Obrázek 11. Príklad vytvorených textúr 3

21 MASTERMIND

Tento skript je mozog celej AI operácie, ktorý riadi beh ostatných AI skriptov. Jednotlivé skripty pritom drží ako svoje pod objekty čo mu zaisťuje priamy prístup ku ich metódam a funkcionalite. Nastaviteľné parametre skriptu sú zobrazenéné Obrázku 12.



Obrázek 12. Mastermind skript

Tento skript má v skratke za úlohu:

- Spravovať proces vĺn útočiacich nepriateľov.
- Prekladať výstupy z AI modelov do dátových štruktúr, z ktorých môže hra tvoriť obsah.
- Pomocou **TaskGen** tvoriť konfigurácie pre konštruovanie Mechabotov a priraďovanie úloh.
- Pomocou **Texturegen** a **BotAssembler** skriptov týchto Mechabotov konštruovať.
- Opravovať výstupy z AI modelov, ktoré sú nepoužiteľné.
- Spravovať aktívnych mechabotov.

Tento priebeh je zhodný s grafom 7.. Nová vlna začína požiadavkou na jazykový model aby vytvoril konfigurácie pre Mechabotov aktuálnej vlny. Keď tieto konfigurácie obdrží, preloží a skontroluje, že sú použiteľné, posiela ich do BotAssembler, ktorý z konfigurácií nepriateľov vyrobí a vloží do hry. Počas toho posiela ďalšiu požiadavku

na jazykový chatbotovi, ktorý z nej pripraví úlohy pre týchto novo vzniknuté postavy. Znovu prebehne kontrola odpovede, a v prípade správnych dat úlohy vyšle jednotlivým Mechabotom, ktorý na základe nich začnú konať. Ak počas prekladu a overovania správnosti odpovede dôjde ku chybe, alebo skript zistí, že dáta sú nepoužiteľné, posiela sa požiadavka na generovanie znovu. Po tom, ako sú Mechaboti nasadení do akcie, skript ešte zašle request generátoru obrázkov na vygenerovanie textúr pre ďalšiu vlnu. Z dôvodu, že generovanie obrázkov zaberá značne dlhší čas ako generovanie textu, bolo by neefektívne predlžovať príchod novej vlnu čakaním na odpoveď tohto modelu. Preto je tento model volaný až na konci vlny a nové textúry sa tak načítajú až v ďalšom kole.

22 BOTASSEMBLER

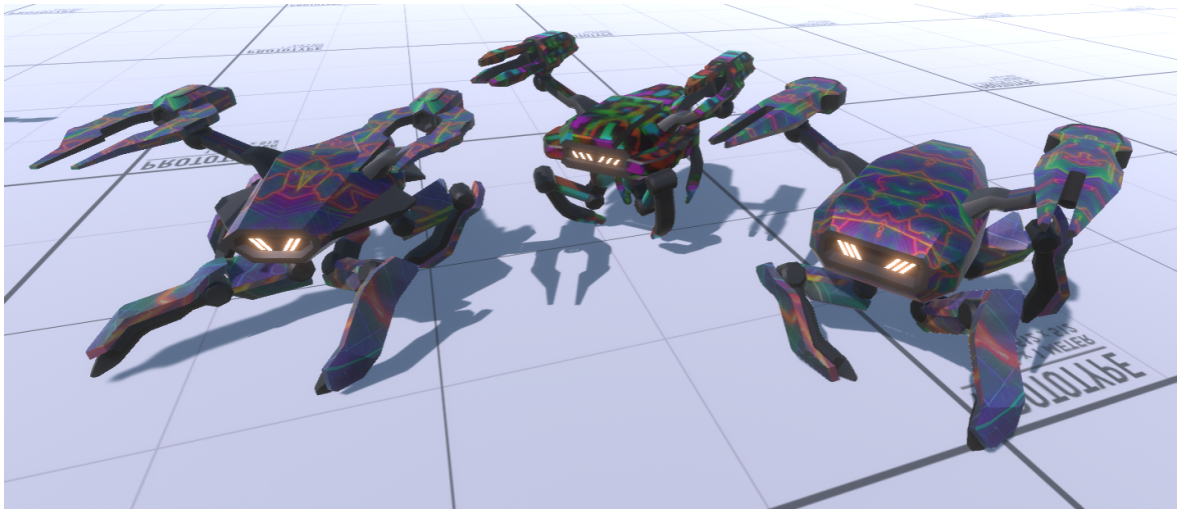
Po tom ako **Mastermind** skript úspešne preloží odpoveď od jazykového modelu a serializuje ju do **Dictionary** objektu, posielajú sa tieto dáta do skriptu **BotAssembler**. Tento skript sa stará o konštrukciu Mechabotov z jednotlivých dielov, nastavuje ich parametre a upravuje textúry. Skript má rôzne verejne prístupné premenné vo forme listov, do ktorých je možné v Unity Editore priradiť prefaby jednotlivých dielov. To zaznamená, že pri importovaní nových dielov do prostredia hry je možné ich do týchto zoznamov dynamicky pridávať alebo odoberať.

Konkrétna stavba prebieha v metóde **AssembleBot**, kde sa sekvenčne jednotlivé komponenty tvoria metódou **Instantiate**. V priebehu metódy sa jednotlivé komponenty vyberajú na základe ID priradených jazykovým modelom. Aj pri dôkladnom usmernení jazykového modelu sa môže stať, že AI chatbot vytvorí v konfigurácií odkazy na modely, ktoré v skutočnosti tento skript v svojich zoznamoch nemá. Jedným z riešení by mohlo byť požiadať jazykový model o pre generovanie konfigurácií, ktoré sú chybné, prípadne chybné konfigurácie nahradiť náhodnými. Z dôvodu vyššej rýchlosti a jednoduchosti sú však v tomto projekte pred tvorbou modelu neplatné ID jednoducho nahrané nulou:

```
1 int hullIndex = Mathf.Clamp(blueprint['H'], 0, _hulls.Count-1);
2 int weapIndex = Mathf.Clamp(blueprint['W'], 0, _weapons.Count-1);
3 int legIndex  = Mathf.Clamp(blueprint['L'], 0, _legMounts.Count-1);
```

Objektom sa pri vytváraní tiež nastavuje rodičovský objekt, aby Mechabot držal pohromade, priradzuje sa **Rig** komponent, ktorý ovláda animácie modelu a na konci sa Mechabotovi priradia udalosti vo forme C# eventov, na ktoré môže hra reagovať.

Keď je Mechabot poskladaný, z jednotlivých modelov skript extrahuje materiály, ktoré bude prefarbovať. Ak je model vytvorený správne podľa kapitoly 16.2 Materiály, stačí sa odkazovať na materiál jeho indexom. Metóda **PaintBot** následne každému extrahovanému materiálu zmení textúru podľa dostupných textúr vytváraných AI modelom. Výber konkrétnych textúr je zase možné implementovať ľubovoľne, pre jednoduchosť sú však v ukážkach textúry zo zoznamu vyberané náhodne. V cykle sa táto textúra nastaví na všetky vybrané materiály Mechabota. Počas toho ako tento skript tvorí mechabotov, predáva referencie na nich späť **Mastermind** skriptu, ktorý nad nimi ďalej v hre preberá kontrolu. Obrázok 13. zobrazuje Mechabotov v hre.



Obrázek 13. Poskladaní a nafarbení Mechaboti

23 KRITICKÉ STAVY

Použitie generatívnej AI spôsobom, že od nej požadujeme tvorbu veľmi špecifického herného obsahu, predstavuje určité výzvy, ktorým treba čeliť.

23.1 Presnosť odpovedí

Často sa v práci spomína, že by model **"mal"** vygenerovať výstup aký chceme. Problém je skutočnosť, že aj keď modelu povieme presne akým spôsobom má odpoveď vytvoriť, to čo vygeneruje nevieme priamo ovplyvniť a sme odkázaní viac menej na jeho schopnosti.

Testovanie vyššie rozoberaných promptov 19.3.1 na veľkých modeloch ako ChatGPT 3.5 a ChatGPT 4 má za výsledok odpoveď, ktorá je v 99% prípadoch správna čo sa týka formátu aj kontextu. Toto sú však najpokročilejší chatboti, ktorí sú aktuálne dostupní. Nanešťastie je možné ich použiť iba pri testovaní mimo hrania hry, keďže nie je možné voľne používať ich komunikačné API rozhranie. Z toho dôvodu hra na všetky textové generovania používa lokálne jazykové modely (14.2), prevažne model Zephyr Beta alebo Mistral 7B. Tieto menšie modely sú zvolené jednak preto, že sú relatívne rýchle a pamäťovo nenáročné. Hra s týmito modelmi pracovala od začiatku, preto sú aj jednotlivé komunikačné skripty tomu prispôsobené.

Avšak aj pri veľmi starostlivo tvorenom prompte sa môže stávať, že tieto modely zlyhajú pri dodržaní požadovaného formátu, alebo odpovedajú úplne nesprávnym spôsobom. Tieto prípady veľmi závisia od konkrétneho modelu, a aj malá úprava promptu môže mať značný dopad na kvalitu odpovede.

Príkladom nesprávných odpovedí pri tvorbe úloh formát (B-# : XX-X,):

- 0 : CP-A, 1 : AP-M, 2 : AP-L, 3 : SI, 4 : CP-B, (chýba znak **B**)
- 0-3 : CP-B, 1-0 : SI, 2-2 : AP-L, 2-3 : SI, 3-0 : AP-S, 4-3 : SI, (nesprávne označenie príkazu)
- B-0 : AP-S,CP-C, B-1 : SI,AP-M, B-2 : CP-A,AP-L, B-3 : CP-B,SI, B-4 : SI,AP-S, (viacero príkazov pre jedno **ID**)
- 87 : L1, W3. Stay idle., // Focused on speed and staying away in an open space fight scenario... (nekompletný zoznam)
- I am unable to generate tasks for each character in the format you requested, as I am unable to access or reference character attributes or abilities within the context of this task-generating process. (model generuje úplne mimo požadovaný formát)

Toto sú ukážky len niektorých nevydarených odpovedí od rôznych chatbotov. Všetci však reagovali na rovnaký prompt pre tvorbu úloh uvedený vyššie. Aj pri správnej formulácií promptu sa približne v 1/4 prípadov stáva že je odpoveď nesprávna. Nie však všetky odpovede, ktoré nedodržia formát sú nepoužiteľné. Ak chatbot iba zamení text **B-0** za **0-B** alebo **Command 0 B-0**, je stále možné programovo z tejto odpovede dostať informáciu, ktorú potrebujeme. Tieto chyby v hre ošetruje **Mastermind** skript pomocou rôznych Regex funkcií. Stále však nie je žiadúce aby také odpovede boli frekventované, nakoľko každá nesprávna odpoveď predlžuje celkový proces generovania obsahu.

23.2 Dostatočný kontext

Cieľom komunikácie s AI modelom je aby bola čo najefektívnejšia ale zároveň dostatočne presná. Prompt je preto nutné vybalansovať tak, bol pre model dostatočne informatívny ale zároveň aby nebol príliš dlhý. Veľké jazykové modely dokážu porozumieť komplexným príkazom, ktoré často definujú rôzne úlohy naraz. Avšak menšie modely sa v obsiahlom prompte môžu ľahko stratiť, a kvalita odpovede tým pádom klesá. Na druhú stranu, použitie všestranného komerčného jazykového modelu znamená, že je veľmi málo spôsobov, akými sa model môže dozvedieť o pozadí hry a jeho úlohe v nej.

Najjednoduchší prístup je všetok nevyhnutý kontext objasniť priamo v užívateľskej správe, prípadne v roli daného modelu. Univerzálne funkčný príkaz by preto mal objasňovať kontext iba do takej miery aby model dokázal vygenerovať relevantné dáta pre hru. Toto nastavenie je predmetom testovania a experimentovania s konkrétnym modelom.

23.3 Rýchlosť

Po úspešnom spojazdnení komunikácie s AI modelmi, je najkritickejší bod hry načasovanie tejto komunikácie. Generatívny model oproti ostatným aplikáciám spotrebováva značne vysoký výkon. To znamená, že jedna generatívna požiadavka na AI model si vyžiada značné množstvo času. Časový priebeh kompletného cyklu operácií znázornených na grafe 7. vyzerá približne takto:

LM model: **Zephyr-7B**
SD model: **StableDiffusion-1.5**
Generovanie 10 Mechabotov

- 8-11s - generovanie konfigurácií pre Mechabotov
- Instancovanie mechabotov je hotové takmer okamžite

- 4-7s - generovanie úloh pre Mechabotov
- 20-25s - generovanie textúr (4 textúry, veľkosť 256*256px, samples: 20)

Z toho vyplýva že kompletné spustenie novej vlny nepriateľov zaberie priemerne 36 sekúnd. Jednotlivé rýchlosti sú samozrejme priamo úmerné veľkosti odpovede. Pri generovaní 30 Mechabotov už konfigurácie zaberajú 25-30s a úlohy 9-10s. Rovnako textúry veľkosti 256*256 pixelov sú dostupné skôr, ako textúry o veľkosti 1028*1028 pixelov. Jednotlivé rýchlosti taktiež závisia od použitého modelu, väčšie modely generujú kvalitnejšie výstupy cenou za dlhší čas. Toto je však ideálny stav v základnom nastavení parametrov.

Ak by sme obe AI, veľký jazykový model aj generátor obrázkov spustili naraz, priemerný čas generovania textu 10 sekúnd sa môže predĺžiť na 2x dlhšiu dobu. Rovno to platí aj na strane obrázkového modelu, kde je tento stav ešte problematickejší, keďže základná doba generovania je 20 a viac sekúnd. Hra preto potrebuje byť implementovaná spôsobom, že AI modely musia generovať obsah sekvenčne.

V každom prípade však proces generovania obsahu v hre predstavuje značne dlhý čas oproti ostatným operáciám. Z toho univerzálne vyplýva, že generovanie by malo ideálne prebiehať mimo dobou, kedy by hráč musel nutne čakať na výsledok odpovede, aby sa hra mohla hýbať ďalej. V tomto projekte prebieha generovanie úloh v dobe keď má hráč čas sa pripraviť na ďalšiu vlnu, kde je časový odstup od akcie prirodzený. Generovanie textúr ale napríklad prebieha už počas aktívnej hry, keď sú Mechaboti vyslaní do boja, čo znamená, že keď hráč vyhrá/prehrá dané kolo, textúry sú už pripravené na použitie. Vo všeobecnosti je však na generovanie priestor napríklad počas herných ukážok, načítavania ďalšieho levelu, alebo počas doby, kedy je hráč mimo akcie, napríklad v menu.

Ak máme správne nastavené všetky parametre spomenuté vyššie, treba myslieť na to, že generovanie bude vždy závisieť v provom rade od výkonu samotného počítača a použitej grafickej karty. Čím väčšia je na grafickú kartu záťaž počas generovania, tým dlhšie bude pochopiteľne generovanie trvať. Tento projekt bol testovaný počítači si grafickou kartou NVIDIA GeForce RTX 4060 s VRAM 8GB. Všetky rýchlosti generovaní uvedené v projekte sú na základe testov na týchto komponentoch. Táto skutočnosť však znamená, že použitie lokálne generatívne AI na nehernom počítači je aktuálne takmer nemožné. Taktiež pokus o generovanie na hernom notebooku bez pripojeného zdroja napájania je extrémne nepraktické. Jednotlivé AI technológie a programy väčšinou ani neumožnia nejaký model spustiť pokiaľ detegujú že hardvérové komponenty počítača sú nepostačujúce. To aspoň v čase písania tejto práce neumožňuje integrovať generatívnu AI lokálne v mobilných hrách alebo štandardných počítačoch. Zakiaľ sa táto technológia bude vylepšovať a adaptovať aj na použitie v menej výkonných stro-

joch, je možné naďalej jej maximálny potenciál využívať jej použitím ako serverovej technológie.

24 POROVNANIE S KLASICKÝM PRÍSTUPOM

Tak ako bolo ukázané v projekte, generatívna AI dokáže plniť úlohy ako manuálnej tvorby assetov, tak aj napodobovať procedurálne generovanie. Je však potrebné rozlišovať kedy sa ju oplatí aplikovať a kedy je jednoduchšie zvoliť štandardný prístup. Manuálna tvorba obsahu, prípadne pomocné procedurálne generátory poskytujú najväčšiu kontrolu nad tvorbou, avšak diverzita tvorby je viac obmedzená. Taktiež je tento prístup viac časovo náročný. Generatívna AI dokáže jednak udržať diverzitu, aj zvýšiť prirodzenosť.

25 VÝHODY POUŽITIA GENAI

Teoretický je možné implementovať jazykové modely alebo generátory 2D/3D grafiky do každej aplikácie aj hry. Stačí definovať a implementovať spôsob, ako bude hra s generatívnym modelom komunikovať a akým spôsobom bude generovaný obsah prezentovať. Generatívna AI nie je dokonalá, ale dokáže vytvárať obsah, ktorý je značne rôznorodý. To ponúka rozsiahle možnosti pri modelovaní dizajnu úrovní, generovaní postáv a textúr, vetvení príbehu alebo tvorbe herného naratívu. Tak môže hratelnosť byť viac nepredvídateľná, svieža a pútavá, čo môže udržať opakovateľnosť hrania. Pri správnej implementácií môže aj jednoduchá hra pri každom hraní vyzerať a pôsobiť trochu inak ako naposledy.

25.1 Preprodukčný nástroj

Veľkým benefitom tejto technológie je, že je schopná generovať veľké množstvo dát a výstupov, v relatívne krátkom čase. Pri použití štandardných postupov môže práca na hre ako vymýšľanie príbehu, prototypovanie, dizajnovanie, kreslenie a celková tvorba herného obsahu v závislosti od veľkosti hry, trvať v rádoch niekoľkých mesiacov až rokov. Generatívna AI však väčšinu týchto vecí dokáže vytvoriť na počkanie, v podstate z jednoduchého textového dotazu. To neznamená, že skupina AI modelov by teraz dokázala nahradiť väčšinu vývojárskeho týmu, ale že môže fungovať ako veľmi mocný nástroj na rýchlu tvorbu konceptov, dizajnu alebo náhodných nápadov, ktoré pomôžu vykresať finálnu podobu hry. Toto platí pri použití AI pred, a počas vývoja hry, avšak rovnako môže mať svoje využitie aj za behu hry, kde môže jej implementácia a využitie byť ešte zaujímavejšie.

25.2 Prispôsobivosť a všestrannosť

Dôvod prečo je Generatívna AI tak populárna, je aj pre to, aký má veľký rozsah možného pôsobenia. Priemerný AI chatbot sa dokáže s užívateľom rozprávať o hoci čom, od receptu na palacinky, cez kvantovú fyziku, až po programovanie komplexnej a špecifickej logiky, a to všetko v rôznych jazykoch. Taktiež stačí pomyslieť na nejakú scénu alebo obraz, popísať ho dostatočne dobre slovami, a iný typ chatbota tento text do chvíľky premení na obrázok. Tieto príklady sú síce značne zjednodušené, avšak poukazujú aké široké spektrum aplikácií dokáže túto technológiu využiť. Tak isto je túto AI možné použiť v hre pre tvorbu zábavného obsahu.

V závislosti od hry je možne implementovať rôzne typy Genratívnej AI naraz a dokonca tieto typy medzi sebou kombinovať. Vygenerovaný text môže byť použitý buď ako herný dialóg, popis nejakého objektu v hernom svete, alebo dokonca ako vstup do iného modelu. Tak môže napríklad jazykový model implementovaný ako textový gene-

rátor, priamo dávať inštrukcie modelu na generovanie obrázkov. Toto správanie však už predstavuje dosť pokročilú formu komunikácie, má však najväčší potenciál vytvárať nečakaný a prekvapivý obsah.

25.3 Rôznorodý obsah

Generatívne modely, hlavne také ktoré sú trénované na ohromnom množstve dát, väčšinou vykazujú vlastnosť, že na jednu vec dokážu vytvoriť veľké množstvo odpovedí. Občas síce tieto odpovede nemusia byť správne, vo väčšine prípadov však rôzni chatboti dokážu vyprodukovať rôzne správne riešenia na rovnaký problém. Tak isto, keď generátor obrázkov dostane príkaz, aby vytvoril nejaký obrázok viac krát, vo väčšine prípadov dokáže vytvoriť obrovské množstvo obrázkov, ktoré pasujú na tento príkaz. Špeciálne táto vlastnosť sa v hrách veľmi hodí, keďže rovnaký herný scenár, môže byť vďaka generovanému obsahu vždy iný, ale zároveň konzistentný s priebehom hry.

Napríklad dialógy postáv, ktoré sú bežne generické a repetitívne, môžu byť v hre nahradené chatbotom, ktorý dokáže tieto dialógy konštruovať dynamicky v závislosti od situácie, alebo hráčových akcií. To sa dá naopak využiť aj pri tvorbe jedinečných postáv, ktoré musia mať vlastnú a unikátnu osobnosť. Pri tvorbe textúr sa dá napríklad vygenerovať veľké množstvo asstetov jedného typu, ako napríklad kamufláže, rôzne vzory, alebo farebné kombinácie. Cieľom je zaistiť, aby pri rôznych opakovaných hraniach hry bol obsah stále dostatočne rôznorodý. Ak AI povieme aby vytvorila určitú zostavu postáv a tiež upravila ich parametre a správanie, skoro vôbec sa nebude stávať, že by tieto zostavy boli rovnaké viac krát po sebe. Vygenerované mapy môžu mať vždy iné prekážky, skratky, alebo nepriateľov. To všetko je možné do určitej miery naviazať na akcie hráča v danej situácií, kedy aktuálny stav hry predstavuje súčasť parametrov pre ďalšie generovanie.

25.4 Znovupoužiteľnosť

Keď sa nám už raz podarí vytvoriť stabilné a funkčné rozhranie, ktoré spojí pozadie hry a konkrétny AI model, nie je zložité túto zostavu prenášať medzi projektami. Samozrejme záleží od žánru hry, aký obsah je potrebné generovať, avšak dostatočne schopný model môže byť použitý vo veľa scenároch bez toho aby ho bolo nutné špeciálne upravovať a prispôsobovať. V prípade integrovania textových generátorov je len potrebné zadefinovať, alebo upraviť funkcie, ktoré budú tento text prekladať tak, aby sa na základe nich hra vedela rozhodnúť čo spraví ďalej. Pri generovaní obrázkov je zase potrebné dôkladne zadať aký grafický výstup požadujeme. Jeden model môže generovať realistické textúry v jednej hre, a rovnako dobre tvoriť pixel art v ďalšej hre. Potom stačí aby hra len na pozadí komunikovala s modelom, kontrolovala stabilitu

spojenia, a štandardným spôsobom ošetrovala možné chyby a nedostatky generovania.

26 NEVÝHODY A NEDOSTATKY

Použitie skutočnej nedeterministickej umelej inteligencie má však aj svoje negatíva a veľmi špecifické limitácie. Napríklad technologická komplexita, vysoké výpočetné nároky, pomerne obmedzená kontrola nad generovaným obsahom spojená s nepredvídateľnosťou, túto technológiu nerobí vhodnou pre využitie vo všetkých žánroch alebo typoch hier. To platí aj pre zariadenia, pre ktoré sú hry určené. Herný stolový počítač bude mať vždy vyššie možnosti využitia tejto technológie ako napríklad smartfón.

V tomto projekte sa mnoho z týchto nedostatkov často prejavovalo, čo znamenalo nutnosť implementovať rôzne optimalizácie alebo malé úpravy hernej logiky, ktoré by v štandardnej hre nebolo pravdepodobne potrebné riešiť. Herný kód preto musel byť robustnejší a rafinovanejší, aby bolo možné tieto problémy za behu hry čo najviac minimalizovať. Nižšie rozoberaný zoznam nedostatkov sa týka hlavne AI použitej v tomto projekte, čiže generovanie textu a obrázkov, avšak tieto limity sú aplikovateľné na tuto technológiu všeobecne.

26.1 Vysoké nároky na výkon

Generatívna umelá inteligencia je už vo všeobecnosti známa svojou extrémnou nenásytnosťou po zdrojoch počítača. Veľké množstvo energie a výpočetného výkonu je potrebné na natrénovanie modelu, zber dostatočného množstva trénovacích a validných dát, ale hlavne na prevádzku takéhoto modelu, teda samotné generovanie obsahu. Koncový užívateľ, ktorý by napríklad komunikoval s chatbotom a generoval obrázky pomocou najnovšieho modelu ChatGPT, žiadnu záťaž na svojom počítači nepocíti, keďže samotný model generuje obsah na vzdialenom serveri. Avšak prevádzka takejto zostavy na domácom počítači už pre počítač predstavuje poriadny záťažový test. Postupným vývojom tejto technológie a aj vývojom špecializovaného hardvéru, sa tieto lokálne modely stávajú stále efektívnejšími a tým aj dostupnejšími na väčšom množstve zariadení, avšak stále znamenajú pre väčšinu bežných počítačov zaujímavú výzvu.

Keďže ja som sa v práci zameral práve na použitie AI modelov spôsobom, ktorý umožňuje ich spúšťať na lokálnom zariadení, znamená to, že generovanie obsahu prebieha zároveň so spustenou hru, ktorú hráč hrá.

Počítač je teda zaťažený spracovávaním hernej logiky, vykresľovaním grafiky, a zároveň aj generovaním textu alebo obrázkov z konkrétneho AI modelu. Z toho automaticky vyplýva, že takýto počítač potrebuje značne výkonné komponenty aby celú túto zostavu udržal v behu. Samotný model je najprv potrebné nahrať do operačnej pamäte a až potom je možné posielať požiadavky na generovanie obsahu. To už prebieha zvyčajne výpočtami na grafickej karte. Z toho jasne vyplýva že veľkosť modelu, ktorý môžeme

v hre použiť je ovplyvnený hlavne veľkosťou RAM a VRAM grafickej karty, kde minimálne požiadavky sa pohybujú 16GB RAM a 6GB VRAM. Dôležité je poznamenať, že zdroje v rovnakom čase stále spotrebováva aj samotná hra, prípadne operačný systém. V prípade vrcholových stolných počítačov takýto viac programový beh, nemusí predstavovať až takú limitáciu, avšak v prípade bežných PC alebo notebookov už ide znateľný problém. S tým je spojená nasledovná limitácia.

26.2 Rýchlosť tvorenia obsahu

Rýchlosť akou dokáže generatívny model vytvárať obsah závislý od výkonu grafickej karty a procesoru, ako aj ich aktuálneho vyťaženia bežiacimi procesmi hry a OS, ale aj komplexity samotného modelu. V prípade veľkých jazykových modelov ide o počet ich parametrov, ktoré sa môžu hýbať v rádoch jednotiek až stoviek GB. Od toho je odvodená aj ich veľkosť, ktorá je približné úmerná počtu parametrov. Jazykový model o veľkosti 7B (7 miliárd parametrov) na disku zaberá približne 7GB. Čím väčší model chceme používať tým viac priestoru v operačnej a grafickej pamäti zaberie. Taktiež väčšiemu modelu trvá vygenerovať odpoveď dlhší čas.

Problém teda nastáva v momente, keď chceme použiť schopný generatívny model (13B parametrov a viac), ale zároveň je na počítači už spustená hra, ktorá tiež spotrebováva značné zdroje. Generovanie textovej odpovede na nejaký dotaz, alebo tvorba obrázku z textu, ktoré by za štandardných okolností trvali pár sekúnd, sa môže generovanie natiahnuť na doslova niekoľko minút. Táto skutočnosť výrazne zmenšuje zoznam použiteľných modelov pre tento scenár. V praxi to znamená, že použitý jazykový model nebude ani zďaleka môcť byť, taký všestranný, ako je napríklad model GPT-4. To isté platí pre generovanie grafiky, kde väčšie modely ako je SDXL dokážu generovať kvalitnú a realistickú grafiku, avšak potrvá to omnoho dlhšie ako menším modelom. V tomto prípade však ide rýchlosť jednoduchšie ovplyvniť znížením požiadavkov na kvalitu generovaných obrázkov.

V kontexte hry, problematika rýchlosti môže znamenať veľmi nepraktické zdržanie v momentoch, kedy by hra musela čakať na výstup z modelu, aby napríklad aktualizovala nové texúry alebo text. O to horšie ak by jeden AI model musel čakať na druhý, aby generoval obsah na základe výstupu z prvého modelu. Preto je nutné pri vývoji takejto hry dôkladne plánovať, kedy má model obsah generovať. Generovanie môže prebiehať napríklad ak je hráč práve v menu, alebo čaká na načítanie nového levelu. Vygenerovaný obsah nemusí použiť hneď, ale uloží sa a použije sa až pri načítaní ďalšej úrovne.

26.3 Závislosť na kapacitách modelu

To že výpočetný výkon počítača limituje vo variabilite použiteľných generatívnych modelov sa odráža hlavne na tom, kde všade môžeme generovanie aplikovať. Vo všeobecnosti čím je model väčší, tým viac všestrannejšie úlohy vie plniť, dokáže si zapamätať a udržať dlhší kontext, alebo môže generovať odpovede v konzistentnom formáte. Keďže jediným vstupom a výstupom z textového modelu je čistý text, je jeho formát veľmi dôležitý. Herné skripty totiž musia byť schopné tento text preložiť a vyvodiť akú logiku majú na základe neho aktivovať.

Ak je jazykový model príliš jednoduchý, nemusí vôbec rozumieť požiadavkom, ktoré mu hrá dáva. Pri použití chatbota na generovanie dialógov postáv to nemusí byť problém, pretože tu je určitá miera voľnosti vítaná. Omnoho komplikovanejšie ale je prinútiť model, aby vytvoril nastavenie ideálnych parametrov pre herné postavy a objekty, alebo zoznam úloh ktoré sa budú postavy plniť. Ak model nerozumie kontextu, alebo robí základné chyby v jednoduchej matematike, tak vytvorí zoznam nastavení, ktoré vôbec nedávajú zmysel. Prípadne to môže pre hráča vyzerať, že sú herné nastavenia generované náhodne. V tomto prípade stojí za zváženie či tento model nie je lepšie nahradiť v hre procedurálnym generovaní, alebo inými jednoduchšími algoritmami, ktoré sú presnejšie a menej výpočetne náročné.

26.4 Malá kontrola nad generovaným obsahom

Ak by sme Generatívnu AI použili pred vydaním hry na tvorbu herných dialógov, nastavení, tvorbu textúr alebo hudby, mali by sme takmer plnú kontrolu nad vytvoreným obsahom vzhľadom ku kontrole vytvorených dát vývojármi. Výstupy je možné jednoducho vygenerovať nanovo, dokiaľ nebudú spĺňať požiadavky. Avšak, ak chceme aby AI dynamicky reagovala na aktuálny stav hry, bude generovanie musieť prebiehať za behu hry, čím odpadá možnosť pohodlného kontrolovania či sú výstupy validné. Tu už nemôže programátor vstúpiť do hry a ovplyvniť akým smerom sa hra bude vyvíjať.

Keď na chatbotovi necháme rozhodnutie navrhnúť hernú mapu, môže sa stať, že navrhne level, ktorý nebude možné prejsť. Pre generovanie obrázkov zase platí, že vygenerovaná grafika môže byť veľmi kvalitná, ale nemusí vôbec pasovať do aktuálneho grafického štýlu. Do určitej miery môže tento stav ovplyvňovať hráč, napríklad tak, že dostane možnosť vygenerovaný obsah prijať a nechať hru sa vyvíjať aktuálnym smerom, alebo požiada AI aby skúsila generovanie znovu. Avšak tento problém by mala byť schopná hra vyriešiť sama, nakoľko získavať od hráča neustále spätnú väzbu či je všetko v poriadku, by mohlo pokaziť herný zážitok.

Tento problém by bolo možné elegantne vyriešiť vytvorením špeciálneho jazykového alebo grafického modelu, ktorý by bol trénovaný primárne na dátach spojených s kon-

krétnou cieľovou hrou. Trénovacími údajmi by mohol byť scenár, námet hry, concept art alebo dokumentácia danej hry, odborná literatúra a historické dáta v prípade realistických hier alebo rôzne databázy. Takýto AI model by jednoduchšie rozumel pozadiu a podstate hry, výtvarnému štýlu alebo dynamickým požiadavkám hráča počas hry a správal by sa tak samo korekčne. Čím užšie by bol model zameraný na danú hru, tým menej by bol náchylný na tvorbu obsahu, ktorý s hrou nesúvisí. Takýto prístup je však predmetom extenzívneho testovania a vylaďovania, ktoré by si pravdepodobne vyžadovalo samostatné odvetvie a špecializovaný tým vývojárov. Preto je dôležité vedieť sa rozhodnúť, čo je vhodné generovať, a čo je lepšie nechať vytvoriť skúseným grafikom alebo vývojárom.

27 POTENCIÁL V BUDÚCNOSTI

Už v čase písania tejto práce je využitie Generatívnej technológie v hre veľmi zábavná skúsenosť. Nemusí ísť len o generovanie herného obsahu, ale aj o pomoc programovať komplexný herný kód, tvoriť nápady a scenáre. Táto práca implementuje prevažne generovanie textu a grafiky, pretože sú tieto aplikácie aktuálne najviac komerčne dostupné a funkčné. Ako sa však AI technológie vyvíjajú, pribúdajú aj možnosti, toho čo je možné generovať. Sem patrí napríklad generovanie hudby, zvukov, videa, animácií alebo dokonca 3D modelov. Z ďaleka však ešte táto technológia nie je na takej úrovni, aby bolo z textu možné v priebehu okamihu vytvoriť funkčnú, alebo unikátnu hernú postavu.

Táto kapitola popisuje možno viac negatív ako pozitív, vyplýva to však z veľmi špecifickej implementácie hry, ktorú aktuálne dostupné generatívne technológie umožňujú. Aj napriek určitým prekážkam, ktoré sú zatiaľ neoddeliteľnou súčasťou práce s touto technológiu, je možné všeobecne vytvoriť funkčnú a hrateľnou hru, ktorá túto technológiu kreatívnym spôsobom implementuje. Väčšina súčasných limitácií generatívnej AI vyplýva z toho, že tu táto technológia existuje ešte pomerne krátko. Najvýkonnejšie modely prevádzkované veľkými spoločnosťami na trhu sú väčšinou dostupné len v obmedzenej forme a ich pokročilé funkcie sú väčšinou dostupné len ako súčasť plateného balíka. Výkonné lokálne modely je naopak extrémne zložité, až nemožné prevádzkovať na bežnom stolnom počítači bez vysokovýkonných grafických kariet, a vysokého výkonu počítača. Taktiež nie je úplne praktické, aby si hráč musel inštalovať hru o veľkosti niekoľko gigabajtov, a ku tomu rovnako veľký jazykový a grafický generatívny model.

Preto zatiaľ najoptimálnejším riešením týchto limitácií sa javí používať generatívnu technológiu na strane herného serveru. Tým by sa čiastočne mohol vyriešiť problém nedostatočného výkonu a nízkej rýchlosti generovania na hosťovskom počítači, keďže AI model by na generovanie mohol používať výhradne výpočetný výkon serveru. V závislosti od výkonu serveru by bolo možné používať lepšie a kvalitnejšie AI modely, prispôsobené konkrétnej hre. V prípade Multiplayer hier je táto možnosť viac efektívna, keďže generovaný obsah by bol spoločný pre všetkých hráčov na danom serveri. Trochu nevýhodné by to bolo v prípade Singleplayer hier, nakoľko by bolo neustále nutné byť pripojený ku internetu, aby AI mohla dynamicky generovať obsah do hry.

Postupné adoptovanie tejto technológie v konečnom dôsledku môže zmeniť to akým spôsobom sa ku vývoju hier pristupuje. Zdĺhavé a časovo náročne modelovanie a animovanie postáv bude možno časom možné tvoriť pomocou textovej špecifikácie a iba detaily budú upravované ručne. Taktiež tvorba koncept-art dizajnu sa môže byť výrazne rýchlejšia a kreatívnejšia. Stále však obsah vytváraný AI generatívnymi modelmi

bude len niečo, čo už vytvoril niekto pred nimi. Preto zatiaľ bude potrebné aby mal kontrolnú ruku nad vývojom človek, a AI bude slúžiť primárne ako pomocný nástroj, ktorý bude tento vývoj pomáhať urýchľovať a zjednodušovať.

ZÁVER

Generatívna umelá inteligencia už dnes dosahuje impresívne výsledky v oblasti napodobovania funkcionalít bežných algoritmov pre tvorbu obsahu. Teoretická časť práce sa zameriavala na analýzu bežne používaných AI algoritmov, ako je procedurálne generovanie, vyhľadávanie cesty alebo použitie strojového učenia v hrách. Rozoberaný bol však aj princíp fungovania generatívnych AI technológií a rôzne oblasti ich využitia. Spomenuté boli aj niektoré z dôležitých návrhových vzorov, bez ktorých by bolo problematické hru implementovať.

V praktickej časti práci boli popísané možnosti spustenia, konfigurovania aj obsluhy generatívnych AI technológií, konkrétne jazykových modelov a obrázkových generátorov. Súčasťou praktickej časti práce bola tvorba a návrh prototypu hry v hernom engine Unity. Hra bola dizajnovaná tak, aby dokázala s AI modelmi komunikovať a generovať pomocou nich herný obsah, napríklad na základe podnetov hráča alebo v závislosti od vytovreného herného sveta. Týmto spôsobom je možné rýchlo a efektívne tvoriť herné textúry, testovať personalizovaných herných agentov alebo implementovať nepredvídateľné chovanie herných postáv. Jednotlivé herné skripty riešia komunikáciu s modelmi, tvorbu generatívnych príkazov, preklad odpovedí z modelov, potrebnú serializáciu dát, alebo šetrenia chybových stavov. Týmto sa podarila splniť úloha demonštrácie a implementácie generatívnych modelov počas vývoja aj hrania hry. Ako prostredník pre túto komunikáciu medzi modelmi a hrou slúžili programy Stable Diffusion Webui a LM Studio. Vyberané boli také nástroje AI, ktoré z hľadiska dostupnosti a výkonu bolo možné používať ako počas vývoja a testovania hry, tak aj počas samotného hrania. Preto sa projekt zameriava hlavne na generovanie textu a obrázkov, avšak generovanie zvuku alebo 3d modelov je oblasť, ktorá bude v budúcne rovnako použiteľná.

V závere práce sú popísané rôzne výhody, vďaka ktorým tento typ AI dokáže konkurovať bežným technikám používaným na tvorbu obsahu, alebo ich v určitých oblastiach efektívne prekonávať. Avšak jej veľmi špecifické limitácie jej aspoň zatiaľ zabraňujú, aby bola použiteľná univerzálne naprieč celým vývojom. Tieto limitácie vyplávajú ako z obmedzených zdrojov vo forme nedostatočného výpočetného výkonu, tak nízkej spoľahlivosti tejto technológie. Zatiaľ sa ukazuje, že jej ideálne použitie je v kombinácií s ostatnými algoritmami, ktoré sú otestované a optimalizované na konkrétny typ úloh. Túto technológiu čaká ešte mnoho vylepšení, či už z pohľadu výkonu, efektivity, alebo možností samotného tvoreného obsahu. Tak ako každá nová technologická vychytáva, aj táto technológia sa časom môže stať neoddeliteľnou súčasťou vývoja hier, či už ako pomocný nástroj pre vývojárov, alebo ako plne autonómna vývojová jednotka.

SEZNAM POUŽITÉ LITERATURY

[1] JAEJUN, Lee a JUNHEE, Lee. *GENERATIVE AI FOR THE SUPER GAME DESIGNER.* 2023. ISBN 978-989-8704-49-8.

[2] COMI, Mauro. *How to teach AI to play Games: Deep Reinforcement Learning.* Towards Data Science. 2018. Dostupné z: https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a. [cit. 2024-04-23].

[3] M. BOURG, David a SEEMANN, Glenn. *AI for Game Developers.* O'Reilly Media, 2004. ISBN 978-1-491-90010-9.

[4] JUSTESEN, Niels; BONTRAGER, Philip; TOGELIUS, Julian a RISI, Sebastian. Deep Learning for Video Game Playing. Online. *IEEE Transactions on Games.* 2019, č. 12, ISSN 2475-1510. Dostupné z: https://ieeexplore.ieee.org/document/8632747. [cit. 2024-04-23].

[5] NYSTROM, Robert. *Game Programming Patterns.* Robert Nystrom, 2014. ISBN 0990582906.

[6] KROGH-JACOBSEN, Thomas. *Level up your code with game programming patterns.* Online. Unity Blog. 2022. Dostupné z: https://blog.unity.com/games/level-up-your-code-with-game-programming-patterns. [cit. 2024-04-24].

[7] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph a VLISSIDES, John. *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education, 1994. ISBN 978-0-321-70069-8.

[8] MILLINGTON, Ian. *AI for Games, Third Edition.* 3rd ed. CRC Press, 2019. ISBN 978-1-351-05330-3.

[9] SHAKER, Noor; TOGELIUS, Julian a J. NELSON, Mark. *Procedural Content Generation in Games.* Springer International Publishing, 2016. ISBN 978-3-319-42716-4.

[10] INVORLD TEAM. *AI and the future of gaming for game devs.* Online. Inworld AI. 2023. Dostupné z: https://inworld.ai/blog/ai-in-gaming-future-of-gaming-for-game-developer. [cit. 2024-04-24].

[11] IKEADA, Aoi. *UNRAVELING THE MYSTERIES OF PROCEDURAL GENERATION IN GAMING.* Online. Token Gamer. 2023. Dostupné z: https://tokengamer.io/unraveling-the-mysteries-of-procedural-generation-in-gaming/. [cit. 2024-04-23].

[12] BOMING, Xia; XIAOZHEN, Ye a ADNAN O.M, Abuassba. *Recent Research on AI in Games.* Limassol: IEEE, 2020. ISBN 978-1-7281-3129-0.

[13] F. CODD, Edgar a L. ASHENHURST, Robert. *Cellular Automata.* Elsevier Science, 2014. ISBN 978-1-4832-2517-3.

[14] A. MARTIN, Glenn; E. HUGHES, Charles; SCHATZ, Sae a NICHOLSON, Denise. *The use of functional L-systems for scenario generation in serious games.* 6th ed. Association for Computing Machinery, 2010. ISBN 978-1-450-30023-0.

[15] ANDERSSON, Johan; BERGMAN, Lage; HILDINGE, Erik; IVERSEN, Johan a JOHANSSON, Daniel. *A Video Game Using Procedural Animation.* Gothenburg: Chalmers University of Technology, 2017.

[16] CHRISTO, Alex. *PROCEDURAL CREATURE GENERATION AND ANIMATION FOR GAMES.* Bournemouth: Bournemouth University, 2022.

[17] M. SMELIK, Ruben; KLASS, Jan; DE KRAKER, Jan; A. GROENWEGEN, Saskia; TUTENEL, Tim et al. *A Survey of Procedural Methods for Terrain Modelling.* Amsterdam: Delft University of Technology, 2009.

[18] J. ROSE, Thomas a G. BAKAOUKAS, Anastasios. *Algorithms and Approaches for Procedural Terrain Generation.* Northampton: University of Northampton, 2016.

[19] ROSADO, Valencia; OSWALDO, Luis a STAROSTENKO, Oleg. *Methods for Procedural Terrain Generation: A Review.* Cham: Springer International Publishing, 2019. ISBN 978-3-030-21077-9.

[20] DENHAM, Thomas. *What are Procedural Textures (Procedural Maps)?* Online. Concept Art Empire. 2023. Dostupné z: https://conceptartempire.com/procedural-textures-maps/. [cit. 2024-04-24].

[21] THEPRO3DSTUDIO. *Everything You Wanted to Know About Procedural Modeling.* Online. ThePro3DStudio. 2023. Dostupné z: https://professional3dservices.com/blog/procedural-modeling.html. [cit. 2024-04-24].

[22] CHOI, Rene Y.; COYNER, Aaron S.; KALPATHY-CRAMER, Jayashree; CAMPBELL, J. Peter a CHIANG, Michael F. Introduction to Machine Learning, Neural Networks, and Deep Learning. Online. *Translational Vision Science & Technology.* 2020, roč. 9, č. 2. Dostupné z: https://tvst.arvojournals.org/article.aspx?articleid=2762344. [cit. 2024-04-23].

[23] Artificial intelligence (AI) algorithms: a complete overview. Online. *Tableau*, 2024. Dostupné z: https://www.tableau.com/data-insights/ai/algorithms. [cit. 2024-04-23].

[24] BUONTEMPO, Frances. *Genetic Algorithms and Machine Learning for Programmers.* Pragmatic Bookshelf, 2019. ISBN 978-1-680-50620-4.

[25] GREAT LEARNING TEAM. *Types of Neural Networks and Definition of Neural Network.* Online. In: Great Learning. 2023. Dostupné z: https://www.mygreatlearning.com/blog/types-of-neural-networks/. [cit. 2024-04-23].

[26] GRIEVE, Patrick. *Deep learning vs. machine learning.* Online. In: Zendesk. 2023. Dostupné z: https://www.zendesk.com/blog/machine-learning-and-deep-learning/. [cit. 2024-04-23].

[27] SHARMA, Sarag. *Activation Functions in Neural Networks.* Online. Towards Data Science. 2017. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [cit. 2024-04-23].

[28] KELLY, Patrick. *Genetic Algorithms in Games (Part 1).* Online. Game Developer. 2018. Dostupné z: https://www.gamedeveloper.com/design/genetic-algorithms-in-games-part-1-. [cit. 2024-04-24].

[29] GOZALO-BRIZUELA, Roberto a C. GARRIDO-MERCH´AN, Eduardo. *A survey of Generative AI Applications.* Universidad Pontificia Comillas, 2023.

[30] LAWTON, George. *What is generative AI? Everything you need to know.* Online. Tech Target. 2024. Dostupné z: https://www.techtarget.com/searchenterpriseai/definition/generative-AI. [cit. 2024-04-24].

[31] TUNSTALL, Lewis; WOLF, Thomas a WERRA, Leandro von. *Natural Language Processing with Transformers, Revised Edition.* O'Reilly Media, 2022. ISBN 978-1-098-13679-6.

[32] FOSTER, David. *Generative Deep Learning.* O'Reilly Media, 2019. ISBN 978-1-492-04194-8.

[33] FP TEAM. *Generative AI: Advantages, Disadvantages, Limitations, and Challenges.* Online. Fact Protocol. 2023. Dostupné z: https://fact.technology/learn/generative-ai-advantages-limitations-and-challenges/. [cit. 2024-04-24].

[34] STRYKER, Cole; BERGMANN, Dave. *What is an autoencoder?* Online. IBM. 2023. Dostupné z: https://www.ibm.com/topics/autoencoder. [cit. 2024-04-24].

[35] CHAUDHARY, Shreya. *A High-Level Guide to Autoencoders.* Online. Medium. 2019. Dostupné z: https://towardsdatascience.com/a-high-level-guide-to-autoencoders-b103ccd45924. [cit. 2024-04-24].

[36] MICHELUCCI, Umberto. An Introduction to Autoencoders. Online. *ArXiv.* 2022. Dostupné z: https://arxiv.org/abs/2201.03898. [cit. 2024-04-24].

[37] ARVINDPDMN. *Transformer Neural Network Architecture.* Online. DEVOPEDIA. 2019. Dostupné z: https://devopedia.org/transformer-neural-network-architecture. [cit. 2024-04-24].

[38] UTKARSH, Ankit. *Transformer Neural Networks: A Step-by-Step Breakdown.* Online. BuiltIn. 2022. Dostupné z: https://builtin.com/artificial-intelligence/transformer-neural-network. [cit. 2024-04-24].

[39] WOOD, Thomas. *What is a Transformer Neural Network?* Online. DeepAI. 2023. Dostupné z: https://deepai.org/machine-learning-glossary-and-terms/transformer-neural-network. [cit. 2024-04-24].

[40] GOODFELLOW, Ian; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing a WARDE-FARLEY, David. Generative adversarial networks. Online. *Communications of the ACM.* 2020, roč. 63, č. 11. Dostupné z: https://dl.acm.org/doi/10.1145/3422622. [cit. 2024-04-24].

[41] CRESWELL, Antonia; WHITE, Tom; DUMOULIN, Vincent a ARULKUMARAN, Kai. *Generative Adversarial Networks: An Overview.* Online. IEEE. Roč. 35, č. 1. ISSN 1053-5888. [cit. 2024-04-24].

[42] MEARIAN, Lucas. *What are LLMs, and how are they used in generative AI?* Online. Computerworld. 2024. Dostupné z: https://www.computerworld.com/article/3697649/what-are-large-language-models-and-how-are-they-used-in-generative-ai.html. [cit. 2024-04-24].

[43] BROADHEAD, Greg. *A Brief Guide To LLM Numbers: Parameter Count vs. Training Size.* Online. Medium. 2023. Dostupné z: https://medium.com/@greg.broadhead/a-brief-guide-to-llm-numbers-parameter-count-vs-training-size-894a81c9258. [cit. 2024-04-24].

[44] WILSON, Mark. *ChatGPT explained – everything you need to know about the AI chatbot.* Online. TechRadar. 2024. Dostupné z: https://www.techradar.com/news/chatgpt-explained. [cit. 2024-04-24].

[45] O'CONNOR, Ryan. *Introduction to Diffusion Models for Machine Learning.* Online. AssemblyAI. 2022. Dostupné z: https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/. [cit. 2024-04-23].

[46] HUGGING FACE. *Hugging Face.* Online. 2023. Dostupné z: https://huggingface.co/. [cit. 2024-04-23].

[47] UNITY TECHNOLOGIES. *Unity.* Online. 2024. Dostupné z: https://unity.com/. [cit. 2024-04-23].

[48] DOLAN, Hugo. *The Ultimate Beginners Guide To Game Development In Unity.* Online. In: FreeCodeCamp. 2019. Dostupné z: https://www.freecodecamp.org/news/the-ultimate-beginners-guide-to-game-development-in-unity-f9bfe972c2b5/. [cit. 2024-04-23].

[49] ZENVA. *C# Game Development – The Complete Beginner's Guide.* Online. In: Zenva. 2023. Dostupné z: https://gamedevacademy.org/csharp-definitive-guide/. [cit. 2024-04-23].

[50] AUTOMATIC1111. *Stable-diffusion-webui.* Online. GitHub. Dostupné z: https://github.com/AUTOMATIC1111/stable-diffusion-webui. [cit. 2024-04-23].

[51] CIVITAI. *Civitai.* Online. 2024. Dostupné z: https://civitai.com/. [cit. 2024-04-23].

[52] STABILITY AI LTD. *Stability AI.* Online. 2024. Dostupné z: https://stability.ai/. [cit. 2024-04-23].

[53] STABLE DIFFUSION. *Stable Diffusion Online.* Online. Stable Diffusion. 2023. Dostupné z: https://stablediffusionweb.com/. [cit. 2024-04-24].

[54] TOOLIFY. *How to Install LM Studio Locally and Run Any Model on Windows or Mac.* Online. In: Toolify. 2024. Dostupné z: https://www.toolify.ai/ai-news/how-to-install-lm-studio-locally-and-run-any-model-on-windows-or-mac-590611. [cit. 2024-04-23].

[55] LM Studio Server. Online. *LM Studio.* 2024. Dostupné z: https://lmstudio.ai/docs/local-server. [cit. 2024-04-24].

[56] OPENAI. *Chat.* Online. 2024. Dostupné z: https://platform.openai.com/docs/api-reference/introduction. [cit. 2024-04-24].

[57] META. *Llama 2: open source, free for research and commercial use.* Online. 2024. Dostupné z: https://llama.meta.com/llama2/. [cit. 2024-04-24].

[58] MISTRAL AI TEAM. *Mistral 7B in short.* Online. Mistral AI. 2023. Dostupné z: https://mistral.ai/news/announcing-mistral-7b/. [cit. 2024-04-24].

[59] HUGGINGFACEH4. *Model Card for Zephyr 7B Alpha.* Online. In: Hugging Face. 2023. Dostupné z: https://huggingface.co/HuggingFaceH4/zephyr-7b-alpha. [cit. 2024-04-24].

[60] GOOGLE. *Gemma Open Models.* Online. Google AI for Developers. 2024. Dostupné z: https://ai.google.dev/gemma. [cit. 2024-04-24].

[61] SCHELL, Jesse. *The Art of Game Design: A Book of Lenses.* CRC Press, 2008. ISBN 978-0-123-69496-6.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AE	Autoencoder
VAE	Variational Autoencoder
AAE	Adversarial Autoencoders
AI	Artificial Intelligence
ML	Machine Learning
PCG	Procedural Content Generation
GAN	Generative Adversarial Network
LSTM	Long Short Term Memory
RNN	Recurent Neural Network
NPL	Natural Language Processing
GPT	Generative Pre-trained Transformer
NPC	Non Player Character
SD	Stable Diffusion
LLM	Large Language Model
FPS	Frames Per Second
API	Application Programming Interface
JSON	JavaScript Object Notation
CPU	Central Processing Unit
GPU	Graphics Processing Unit
LLAMA	Large Language Model Meta AI
RAM	Random-Access Memory
VRAM	Video Random-Access Memory

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I. CD

- DP_Gazdik_Lukas.pdf
- Scripts.zip

PŘÍLOHA P II. ZDROJOVÉ KÓDY PRE OBSLUHU AI MODELOV

Trieda TaskGen

```
using System;
using System.Collections;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;

public class TaskGen : MonoBehaviour, IGenerator
{
    [TextArea(2, 5)][SerializeField] private string _customSpecs;

    private const string _kURL = "http://127.0.0.1:7861";
    private const string _kEndpoint = "/v1/chat/completions";

    private IGenerator.AIState _LMState;
    public IGenerator.AIState LMState => _LMState;

    public enum PromptType { ASSEMBLE_PROMPT, TASK_PROMPT, TEXTURE_PROMPT, STATUAS_PROMPT }


    private struct Payload
    {
        public Messages[] messages;
        public float temperature;
        public int max_tokens;
        public bool stream;
    }
    private struct Messages
    {
        public string role;
        public string content;
    }

    private Messages kDefSystem = new()
    {
        role = "system",
        content = "You are game AI assistant, you generate text only in specified format by user, nothing more"
    };
    private Messages kDefUser = new()
    {
        role = "user",
        content = ""
    };


    /* This is system pre-prompt that works good with Zepthyr 3B
     You are game AI assistant, you generate text only in specified format by user, nothing more
     */

    private string GetNewAssemblePrompt()
    {
        return @"
You are a game AI responsible for configuring characters designed to attack players.
Each character consists of three components: legs, weapons, and hulls, each with multiple models identified by numbers.
The ranges for each component are as follows:

Legs: IDs range from 0 to " + (BotAssembler.Instance.LegCount() - 1) + @" (higher number means faster movement)
Weapons: IDs range from 0 to " + (BotAssembler.Instance.WeapCount() - 1) + @" (higher number means bigger fire power)
Hulls: IDs range from 0 to " + (BotAssembler.Instance.HullCount() - 1) + @" (higher number means more health)

Now generate " + GameData.Instance.NumberOfBots + @" configurations using the following format: 'B# : L# W# H#,' (letters must be there) where:

B# represents the ID of the character (ranging from 0 to " + (GameData.Instance.NumberOfBots - 1) + @")
L indicates the leg model, W indicates the weapon model, and H indicates the hull model, each with an ID within its respective range mentioned above.
Ensure that configurations stay within the valid model ID ranges and always follow the specified format, include a comma after each configuration.
Do not generate other text than specified the specified format.
";
    }

    private string GetNewCommandsPrompt()
```

```
    {
        //Capture point C = CP-C
        static string fillCP()
        {
            string cpFull = "";
            foreach (var cp in MastermindScript.Instance.CapturePoints)
            {
                if (!cp.IsCaptured && cp.Marking != 0)
                {
                    cpFull += "\nCapture point " + cp.Marking + " = CP - " + cp.Marking;
                }
            }
            return cpFull;
        }

        return @"
You are the game mastermind AI tasked with assigning tasks to characters attacking a player. The characters can perform tasks from the following list:"
+ fillCP() + @"
Attack player - short range = AP-S
Attack player - min range = AP-M
Attack player - long range = AP-L
Stay idle = SI

Please generate one task for each character in the following format: 'B-# : XX-X,'
where # represents the ID of the character ranging from 0 to " + (GameData.Instance.NumberOfBots - 1) + @",
and XX-X represents one of the possible tasks from the list above (it's short version).
Ensure to include a comma after each line.
";
    }

    //------------------------------------------------------------------
    private void Start()
    {
        _LMState = IGenerator.AIState.UNKNOWN;
        StartCoroutine(PingAI());
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.H))
        {
            CallGenerator(PromptType.TASK_PROMPT);
        }
    }

    //------------------------------------------------------------------
    public static event Action<string> NotifyGenerationFinished;

    public async void CallGenerator(PromptType pt)
    {
        if (_LMState == IGenerator.AIState.GENERATING)
        {
            Debug.Log("Generating already in process, waiting for finish");
            return;
        }

        if (_LMState == IGenerator.AIState.UNREACHABLE || _LMState == IGenerator.AIState.UNKNOWN)
        {
            Debug.Log("LLM is unreachable");
            return;
        }


        switch (pt)
        {
            case PromptType.TASK_PROMPT:
                kDefUser.content = GetNewCommandsPrompt();
                break;
            case PromptType.ASSEMBLE_PROMPT:
                kDefUser.content = GetNewAssemblePrompt() + _customSpecs;
                break;
            case PromptType.TEXTURE_PROMPT:
                break;
            case PromptType.STATUAS_PROMPT:
                break;
        }
```

```
150
151
152             _LMState = IGenerator.AIState.GENERATING;
153             float elapseS = Time.realtimeSinceStartup;
154             string returned = await GenerateBatch();
155             NotifyGenerationFinished?.Invoke(returned);
156             float elapseE = elapseS - Time.realtimeSinceStartup;
157
158             Debug.Log($"<color=lightblue>{elapseE}</color>");
159             System.IO.File.AppendAllText(GameData.Instance.DebugFilePath
     + ".txt", "LM "+pt.ToString()+" time: " + elapseE.ToString() +
    Environment.NewLine);
160
161             _LMState = IGenerator.AIState.READY;
162         }
163
164         private async Task<string> GenerateBatch()
165         {
166             Payload payload = new()
167             {
168                 messages = new Messages[] { kDefSystem, kDefUser },
169                 temperature = 0.7f,
170                 max_tokens = -1,
171                 stream = false
172             };
173             Debug.Log("Generating... " + kDefUser.content);
174
175             string output = string.Empty;
176             using (HttpClient httpClient = new())
177             {
178                 var jsonRequest = Newtonsoft.Json.JsonConvert.
    SerializeObject(payload);
179                 var content = new StringContent(jsonRequest, Encoding.
    UTF8, "application/json");
180                 HttpResponseMessage response = await httpClient.
    PostAsync(_kURL + _kEndpoint, content);
181                 response.EnsureSuccessStatusCode();
182
183                 string result = await response.Content.ReadAsStringAsync
    ();
184                 var jObject = Newtonsoft.Json.Linq.JObject.Parse(result)
    ;
185                 output = jObject["choices"][0]["message"]["content"].
    ToString();
186                 Debug.Log(output);
187             }
188             return output;
189         }
190
191         public IEnumerator PingAI()
192         {
193             for (; ; )
194             {
195                 yield return Task.Run(async () =>
196                 {
197                     try
198                     {
199                         using HttpClient client = new();
200                         client.Timeout = TimeSpan.FromSeconds(3);
201                         HttpResponseMessage response = await client.
    GetAsync(_kURL);
202
203                         if (response.IsSuccessStatusCode)
204                         {
205                             if (_LMState != IGenerator.AIState.
    GENERATING)
206                                 _LMState = IGenerator.AIState.READY;
207                         }
208                     }
209                     catch
210                     {
211                         _LMState = IGenerator.AIState.UNREACHABLE;
212                     }
213                 });
214                 yield return new WaitForSeconds(3f);
215             }
216         }
217 }
```

Trieda TextureGen

```
using System;
using System.Collections;
using System.IO;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;

public class TextureGen : MonoBehaviour, IGenerator
{
    [TextArea(2, 5)][SerializeField] private string _prompt;
    [TextArea(2, 5)][SerializeField] private string _negaPrompt;
    [SerializeField] private int _bathSize;
    [SerializeField] private int _resolution;
    //[SerializeField] string _outputDir; //  "./Assets/Generations"

    private const string kDefPromp = "gray metal scratched worn texture";
    private const string kDefUpgrade = ", 2D texture, diffuse light, high resolution, (high contrast), game texture, colorfull, big geometric shapes";
    private const string kDefNegaPromp = "carpet, clothing, cartoon, drawing, painting anime, low resolution, bad quality, distortion, blur, objects, out of boundaries, small rsolution, watermark, text, numbers";

    private const string kURL = "http://127.0.0.1:7860";
    private const string kEndpoint = "/sdapi/v1/txt2img";

    private IGenerator.AIState _SDState;
    public IGenerator.AIState SDState => _SDState;

    private struct Payload
    {
        public string prompt;
        public string negative_prompt;
        public int steps;
        public int batch_size;
        public int cfg_scale;
        public int resolution;
        public bool tiling;
    }
    //-------------------------------------------------------------------------

    private void Start()
    {
        _SDState = IGenerator.AIState.UNKNOWN;
        StartCoroutine(PingAI());
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.G))
        {
            CallGenerator();
        }
    }
    //-------------------------------------------------------------------------

    public static event Action NotifyGenerationFinished;

    public async void CallGenerator()
    {
        if (_SDState == IGenerator.AIState.GENERATING)
        {
            Debug.Log("Generating already in process, waiting for finish");
            return;
        }
        if (_SDState == IGenerator.AIState.UNREACHABLE || _SDState == IGenerator.AIState.UNKNOWN)
        {
            Debug.Log("SD is unreachable");
            return;
        }
        _SDState = IGenerator.AIState.GENERATING;
        float elapseS = Time.time;

        bool returned = await GenerateBatch();
        if (returned) NotifyGenerationFinished?.Invoke();

        float elapseE = elapseS - Time.time;
        Debug.Log($"<color=lightblue>{elapseE}</color>");
```

```
        File.AppendAllText(GameData.Instance.DebugFilePath + ".txt", "SD time: " + elapseE.ToString() + Environment.NewLine);
        _SDState = IGenerator.AIState.READY;
    }

    private struct SDImagesRes
    {
        public string[] images;
    }

    private async Task<bool> GenerateBatch()
    {
        Payload payload = new()
        {
            prompt = _prompt != null || _prompt != "" ? _prompt + kDefUpgrade : kDefPromp + kDefUpgrade,
            negative_prompt = _negaPrompt != null || _negaPrompt != "" ? _negaPrompt + kDefNegaPromp : kDefNegaPromp,
            steps = 20,
            batch_size = _bathSize != 0 ? _bathSize : 4,
            cfg_scale = 4,
            resolution = _resolution != 0 ? _resolution : 512,
            tiling = true
        };

        using HttpClient httpClient = new();
        var content = new StringContent(JsonUtility.ToJson(payload), Encoding.UTF8, "application/json");

        Debug.Log("Generating textures... \n" + JsonUtility.ToJson(payload));

        HttpResponseMessage response = await httpClient.PostAsync(kURL + kEndpoint, content);

        response.EnsureSuccessStatusCode();

        string jsonResponse = await response.Content.ReadAsStringAsync();
        SDImagesRes result = JsonUtility.FromJson<SDImagesRes>(jsonResponse);

        if (result.images == null) return false;

        ClearDirFirst(GameData.Instance.TexturesPath);

        for (int i = 0; i < result.images.Length; i++)
        {
            string base64Image = result.images[i];
            byte[] imgBytes = Convert.FromBase64String(base64Image);
            using (var imageFile = new FileStream(GameData.Instance.TexturesPath + "/img" + i + ".png", FileMode.Create))
            {
                imageFile.Write(imgBytes, 0, imgBytes.Length);
                imageFile.Flush();
            }
            //Debug.Log(base64Image);
        }
        Debug.Log("SD generation finished x" + result.images.Length);

        return true;
    }

    private void ClearDirFirst(string path)
    {
        string[] files = Directory.GetFiles(path);
        foreach (var f in files)
        {
            if (f.StartsWith("img") && f.EndsWith(".png"))
            {
                File.Delete(f);
            }
        }
    }

    /// <summary>
    /// Pings SD server periodically so Game knows that it can generate textures
    /// </summary>
    /// <returns>updates _serverRunning field</returns>
    public IEnumerator PingAI()
    {
        for (; ; )
        {
            yield return Task.Run(async () =>
            {
```

```
157                 try
158                 {
159                     using HttpClient client = new();
160                     client.Timeout = TimeSpan.FromSeconds(3);
161                     HttpResponseMessage response = await client.
    GetAsync(kURL);
162 
163                     if (response.IsSuccessStatusCode)
164                     {
165                         if (_SDState != IGenerator.AIState.
    GENERATING)
166                             _SDState = IGenerator.AIState.READY;
167                     }
168                 }
169                 catch
170                 {
171                     _SDState = IGenerator.AIState.UNREACHABLE;
172                 }
173             });
174             yield return new WaitForSeconds(3f);
175         }
176     }
177 }
```