

Primjena umjetne inteligencije u dinamičkom modeliranju ponašanja neigrivih likova

Omazić, Antonio

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:200:810494>

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

Download date / Datum preuzimanja: **2024-12-26**

Repository / Repozitorij:

[Faculty of Electrical Engineering, Computer Science and Information Technology Osijek](#)



**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I
INFORMACIJSKIH TEHNOLOGIJA**

Sveučilišni diplomski studij Računarstvo

**PRIMJENA UMJETNE INTELIGENCIJE U
DINAMIČKOM MODELIRANJU PONAŠANJA
NEIGRIVIH LIKOVA**

Diplomski rad

Antonio Omazić

Osijek, 2024

**FERIT**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK**Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju****Ocjena diplomskog rada na sveučilišnom diplomskom studiju**

Ime i prezime pristupnika:	Antonio Omazić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1312R, 07.10.2022.
JMBAG:	0165083336
Mentor:	izv. prof. dr. sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Alfonzo Baumgartner
Član Povjerenstva 1:	izv. prof. dr. sc. Časlav Livada
Član Povjerenstva 2:	prof. dr. sc. Krešimir Nenadić
Naslov diplomskog rada:	Primjena umjetne inteligencije u dinamičkom modeliranju ponašanja neigrivih likova
Znanstvena grana diplomskog rada:	Umjetna inteligencija (zn. polje računarstvo)
Zadatak diplomskog rada:	Zadatak rada je primijeniti umjetnu inteligenciju u dinamičkom modeliranju ponašanja neigrivih likova u Unityju. Diplomski rad bi trebao uključivati razvoj AI modela koji može simulirati ponašanje likova koji se ne mogu igrati u okruženju igre. Model treba ocijeniti na temelju njegove sposobnosti stvaranja privlačnih i impresivnih iskustava za korisnika. Rezervirano za: Antonio Omazić
Datum ocjene pismenog dijela diplomskog rada od strane mentora:	04.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	03.10.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	26.09.2024.



FERIT

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 26.09.2024.

Ime i prezime Pristupnika:	Antonio Omazić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1312R, 07.10.2022.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom: **Primjena umjetne inteligencije u dinamičkom modeliranju ponašanja neigrivih likova**

izrađen pod vodstvom mentora izv. prof. dr. sc. Časlav Livada

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.

Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD	1
1.1 Zadatak diplomskog rada	2
2. PREGLED PODRUČJA RADA	3
2.1 Tradicionalni NPC modeli	3
2.2 NPC modeli temeljeni na velikim jezičnim modelima	4
3. KORIŠTENE TEHNOLOGIJE I ALATI	6
3.1 Unity	6
3.2 Rider	7
3.1.1 C# u Unityu	8
3.3 LLM for Unity dodatak	8
3.4 Strojno učenje	9
3.4.1 Nadzirano učenje	10
3.4.2 Nenadzirano učenje	10
3.4.3 Podržano učenje	11
3.5 Neuronske mreže	11
3.6 Veliki jezični modeli	14
3.6.1 Transformator model	14
3.6.2 Transformator model u velikim jezičnim modelima	16
3.6.3 Phi-3	19
3.6.4 PEFT	20
3.6.5 LoRA	20
3.7 Google colab	21
3.7.1 Jupyter	22
3.7.2 Hugging Face Hub	22
3.7.3 Pytorch	23
4. IMPLEMENTACIJA RJEŠENJA	24
4.1 Koncept rješenja	24
4.2 Podešavanje velikog jezičnog modela	25
4.2.1 Priprema skupa podataka	25
4.2.2 Primjena LoRa tehnike	27
4.2.3 Treniranje modela	29
4.2.4 Završni koraci podešavanja	30

4.3 Implementacija osnovnih akcija neigrivih likova	32
4.3.1 Kreiranje okruženja	32
4.3.2 Komponente objekta neigrivog lika	34
4.3.3 Implementacija mehanike spremnika	36
4.3.4 Implementacija mehanike kretanja.....	37
4.3.4 Implementacija statistika neigrivih likova.....	38
4.3.5 Implementacija vida neigrivog lika	40
4.3 Implementacija baze podataka u Unityu.....	42
4.4 Integracija velikog jezičnog modela u Unity.....	44
4.5 Vrednovanje simulacije	50
5. POTENCIJALNA POBOLJŠANJA.....	57
6. ZAKLJUČAK	59
LITERATURA.....	61
SAŽETAK	66
ABSTRACT.....	67
PRILOZI.....	68
P.5.1 Kompletan programski kôd projekta.....	68

1. UVOD

Neigrivi likovi (engl. *Non playable character - NPC*) su jedan od kamena temeljaca dizajna videoigara od samog početka industrije. Ovi likovi, koje igrač ne kontrolira izravno, imaju različite uloge u igrama, od pružanja narativne dubine i informacija o misijama (engl. *quest*) do njihovog djelovanja kao protivnici i saveznici. Prema [1] neigrivi likovi u videoigrama su značajno evoluirali od kada su se pojavili, krećući se od jednostavnih, predvidljivih entiteta do složenih likova koji pokazuju sofisticirana ponašanja i interakcije.

Rani neigrivi likovi su često bili ograničeni tehnološkim ograničenjima, što je rezultiralo ponavljajućim i predvidljivim ponašanjima [1]. Međutim, napredak u umjetnoj inteligenciji (engl. *artificial intelligence - AI*) revolucionirao je potencijal neigrivih likova, omogućujući dinamičnije i realističnije interakcije. Dobro dizajnirani neigrivi likovi mogu uvelike poboljšati iskustvo igranja videoigara čineći virtualne svjetove življima i interaktivnijima [2].

Potencijalni značajan napredak u ovom području je primjena strojnog učenja (engl. *machine learning - ML*) i tehnika dubokog učenja na modeliranje ponašanja neigrivih likova. Ovi pristupi umjetne inteligencije omogućuju neigrivim likovima da uče iz svog okruženja i prilagode svoje akcije na temelju akcija igrača i promjena u svijetu igre. Na primjer, neigrivi likovi u borbenim scenarijima sada mogu koristiti složene strategije i reagirati na igračeve taktike u stvarnom vremenu, stvarajući izazovnije i zanimljivije iskustvo, ili korištenje velikih jezičnih modela (engl. *Large Language Model - LLM*) za simuliranje realističnog razgovora.

Unatoč ovom napretku, još uvijek postoje značajni izazovi koje treba prevladati u području dizajna neigrivih likova. Osiguravanje da neigrivi pokazuju uvjerljive emocije, održavaju dosljedne osobnosti i sudjeluju u značajnim društvenim interakcijama s igračima su stalna područja istraživanja. Pronalaženje ravnoteže između računalnih performansi i kompleksnosti ponašanja neigrivih likova predstavlja ključni faktor dizajna videoigara.

U ovom diplomskom radu opisan je razvoj sustava koji može simulirati složeno i prilagodljivo ponašanje neigrivih likova te njegova implementacija u Unity višeplatformskom (engl. *cross-platform*) alatu za razvoj videoigara.

U nastavku rada slijedi pregled područja rada gdje će biti navedena slična rješenja. Zatim slijedi poglavlje u kojemu će biti opisane tehnologije i alati koje će se koristiti u svrhu

zadovoljavanja zahtjeva izrade rada. Nakon toga opisan je koncept sustava koji će zadovoljiti zadane uvjete, zatim implementacija koncepta, te potencijalna poboljšanja nakon implementacije.

1.1 Zadatak diplomskog rada

Potrebno je primijeniti umjetnu inteligenciju u dinamičkom modeliranju ponašanja neigrivih likova u Unityu što uključuje razvoj AI modela koji može simulirati ponašanje likova koji se ne mogu igrati u okruženju igre.

2. PREGLED PODRUČJA RADA

Razvoj i implementacija neigrivih likova u videoigrama značajno je napredovala tijekom godina. Dok strojno učenje i veliki jezični modeli nude obećavajuće nove smjerove, te su tehnike još uvijek u eksperimentalnoj fazi. Posljedica toga je da tradicionalni modeli (NPC modeli koji ne koriste modele strojnog učenja) ostaju znatno popularniji u industriji igara zbog svoje dokazane pouzdanosti, predvidljivosti i nižih zahtjeva za resursima.

2.1 Tradicionalni NPC modeli

Tradicionalne metode bile su okosnica razvoja neigrivih likova, pružajući predvidljiva i kontrolirana ponašanja. Najveće ograničenje ovog pristupa je to što razvojni programeri moraju smisliti svaku rečenicu dijaloga za neigrive likove i napraviti sustav koji će određivati koju od postojećih rečenica dijaloga neigrivi lik treba reći u kojem trenutku. Unatoč tome velikom ograničenju postoje vrlo impresivna rješenja poput video igre Red Dead Redemption 2 (skraćeno RDR2) gdje neigrivi likovi imaju dinamične dnevne rutine, dinamički reagiraju na radnje igrača, pamte prethodne interakcija igrača s njima, ali i interakcije igrača s okolinom. Što dodaje dubinu i realizam u svijet igre [3]. Ovaj pristup naglašava učinkovitost kombiniranja detaljnog skriptiranja sa stabilima ponašanja za stvaranje zanimljivih neigrivih likova [4]. Neigrivi likovi opisani u ovom odlomku su vidljivi na slici 2.1.



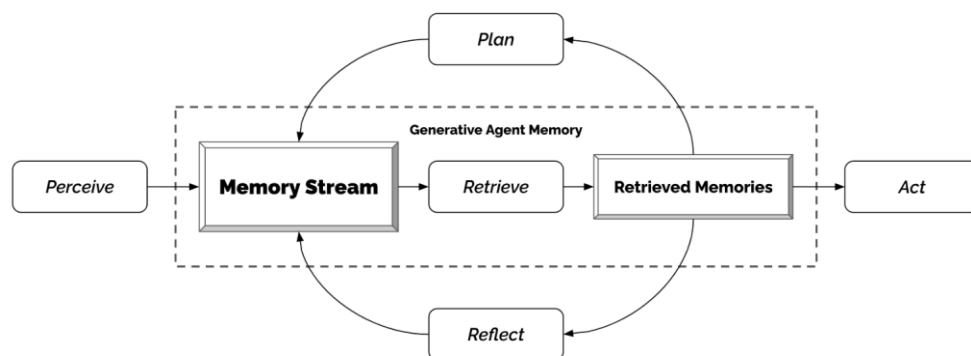
Slika 2.1 Scena iz videoigre RDR2 na kojoj su vidljivi neigrivi likovi [5].

Videoigre poput Red Dead Redemption 2 dokazuju da tradicionalna rješenja mogu proizvesti vrlo učinkovite neigrive likove korištenjem strukturiranih tehnika izrade umjetne inteligencije kao što su konačni automati, stabla ponašanja i detaljno skriptiranje. Ove metode nude visok stupanj kontrole i predvidljivosti, koji su ključni za održavanje dosljednog iskustva igre.

2.2 NPC modeli temeljeni na velikim jezičnim modelima

Kao što je navedeno, kreiranje modela neigrivih likova korištenjem velikih jezičnih modela je još u eksperimentalnoj fazi što pokazuje činjenica da još ne postoji visoko profilna videoigra koja koristi ovaj pristup [6]. Međutim postoje više istraživačkih radova koja pokazuju potencijal korištenja velikih jezičnih modela u videoigramama.

Cilj jednog takvog istraživanja je simulirati ljudsko ponašanje u neigrivim likovima. Istraživanje naglašava kako se ti modeli mogu koristiti za stvaranje neigrivih likova koji će osim dinamičkog reagiranja na igračeve akcije, reagirati i na akcije drugih neigrivih likova [7]. Također istraživanje pokazuje kako ovaj pristup kreiranja neigrivih likova omogućuje tim likovima razvijanje neovisno o igračevim akcijama, na primjer neigrivi likovi će stupiti u veze s drugim neigrivim likovima ili mogu poboljšati znanje u nekom području čitajući knjige. Takvo ponašanje je omogućeno korištenjem kompleksnog sustava odabira akcije vidljivog na slici 2.2, gdje se koriste modeli strojnog učenja za procjenu važnosti i relevantnosti pamćenja neigrivih likova.



Slika 2.2 prikaz sustava odabira akcija rada [7].

Drugi primjer istraživanja pokazuje potpuno drugačiji način implementiranja velikih jezičnih modela u videoigre [8]. Tim iz Microsofta je kreirao dinamičke neigrive likove na način

da veliki jezični model generira kôd koji će omogućiti i izvršiti akciju za koju se taj isti model odlučio. Ovakav pristup pruža gotove beskrajne mogućnosti neigrivim likovima uz pretpostavku da je model sposoban napisati valjani kôd koji će omogućiti točnu akciju koju neigrivi lik želi napraviti.

Mnogi vide ovakav pristup modeliranja neigrivih likova kao budućnost, te je logično pretpostaviti da će postojati platforme koji će nuditi implementaciju takvih rješenja prilikom kreiranja, ali i u postojeće videoigre, uz naplatu. Jedan primjer takvog alata je Convai [9]. Convai omogućuje razvojnim programerima kreiranje, opisivanje i definiranje skupa mogućih akcija neigrivog lika. Convai će onda koristiti veliki jezični model za generiranje odgovarajuće akcije s obzirom na upit.

3. KORIŠTENE TEHNOLOGIJE I ALATI

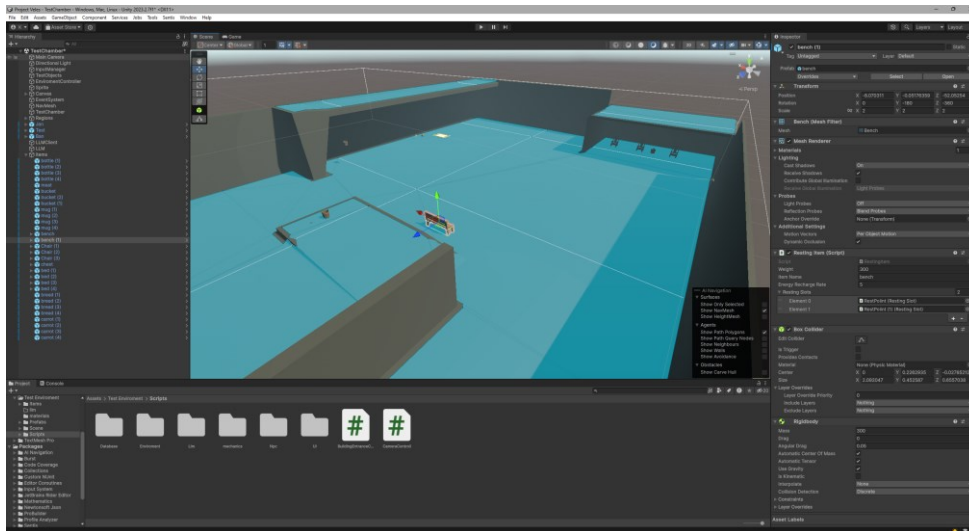
Ovo poglavlje pruža pregled tehnologija i alata korištenih u razvoju modela ponašanja neigrivih likova. Glavni alat u projektu je Unity, platforma u kojoj će se integrirati ostale tehnologije. Te tehnologije uključuju LLM for Unity dodatak, Rider, strojno učenje, neuronske mreže, velike jezične modele, Phi 3 mini model, Google Colab, Hugging Face Hub, te LoRA i PEFT dodatke.

3.1 Unity

Unity je softver koji podržava stvaranje višeplatformskih (engl. *cross-platform*) 2D i 3D igara. Poznat je po svom lako za koristiti korisničkom sučelju, opsežnoj dokumentaciji i velikoj zajednici [10], što sve zajedno olakšava razvoj igara novim programerima [11], dok u isto vrijeme pruža napredne mogućnosti iskusnim profesionalcima.

Jedna od istaknutih sposobnosti Unitya je sposobnost pokretanja igara na više platformi, uključujući računala, konzole, mobilne uređaje i web preglednike. Ova podrška za više platformi osigurava da razvojni programeri mogu doprijeti do široke publike bez potrebe za ponovnim kreiranjem svoje videoigre za različite platforme [11].

Ostale bitne značajke Unitya su napredna simulacija fizike, renderiranje u stvarnom vremenu, razne animacijske i audio mogućnosti. Ovi alati omogućuju razvojnim programerima stvaranje detaljnih i realističnih okruženja videoigara. Unity razvojno podržava učinkovito upravljanje resursima (engl. *asset*) i optimizaciju tijekom rada (slika 3.1) [11]. U slučaju nedostatka mogućnost u samom Unityu, postoji mogućnost da je razvojni programer ne mora sam razvojni programer implementirati već se može pronaći u Unity Asset storeu. Unity Asset Store je online trgovina koja sadrži veliki broj raznih sredstava, skripti i dodataka koji mogu znatno ubrzati proces razvoja videoigara.



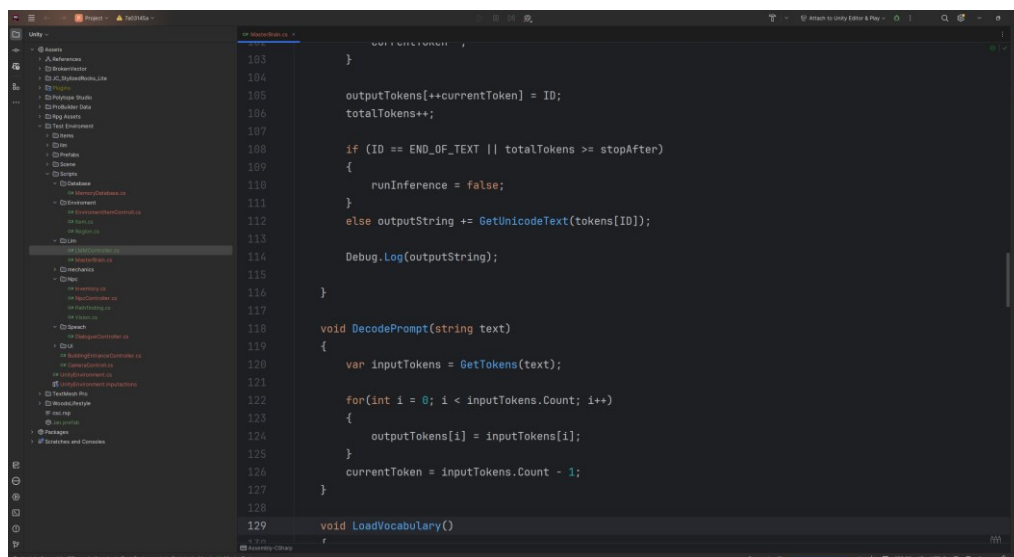
Slika 3.1 Unity razvojno okruženje.

Iako je Unity prije svega poznat kao softver za razvoj videoigara, Unity se sve više koristi u područjima kao što su vlada, zrakoplovstvo, automobilska industrija, arhitektura i obrazovanje. Razlog toga su Unityeve 3D mogućnosti u stvarnom vremenu, što ga čine prikladnim za stvaranje simulacija, programa obuke i interaktivnih vizualizacija [12].

Navedeni široki opseg mogućnosti i alata čini Unity idealnim odabirom za kreiranje naprednih neigrivih Likova.

3.2 Rider

Rider je integrirano razvojno okruženje (engl. integrated development environment - IDE) koje je razvila firma JetBrains u svrhe razvoja C++ i C# projekata. Unity prvenstveno koristi C# programski jezik za kreiranje skripti, što čini Rider osnovnim alatom za Unity programere. Postoje više razvojnih okruženja koje se mogu koristiti u Unityu, ali Rider je izabran zato što nudi naprednu pomoć programiranja, otkrivanje pogrešaka i alate za profiliranje, koji pojednostavljuju razvojni proces i povećavaju produktivnost. Rider pruža podršku za različite operativne sustave kao što su Windows, macOS i Linux, čineći ga pristupačnim alatima za širu zajednicu programera. Osim toga, Rider ima integrirane alate za refaktoriranje kôda, navigaciju kroz kôd, i automatizirano testiranje, što dodatno olakšava održavanje i razvoj kompleksnih projekata. Sve ostale mogućnosti i upute za korištenje mogu se pronaći na službenim stranicama alata [13].



Slika 3.2 Rider sučelje

3.1.1 C# u Unityu

C# je višestruko-paradigmatski programski jezik razvijen od strane Microsofta kao dio .NET inicijative. Dizajniran je za jednostavnost, robusnost i sigurnost, a koristi se za širok spektar aplikacija, uključujući razvoj igara [14]. U kontekstu Unitya, C# je glavni programski jezik korišten za pisanje skripti koje definiraju ponašanje objekata unutar igre [10].

Sintaksa C# je intuitivna i slična drugim C-stil jezicima, što olakšava njegovo učenje i upotrebu. Osim toga, C# skripte se jednostavno integriraju s Unity okruženjem, omogućavajući direktno vezivanje kôda uz objekte i scene unutar igre. C# također dolazi s bogatom standardnom bibliotekom koja nudi mnoštvo funkcionalnosti, od manipulacije podacima do naprednih mrežnih operacija.

3.3 LLM for Unity dodatak

Dodatak (engl. *plugin*) LLM za Unity pojednostavljuje integraciju velikih jezičnih modela s Unityem, čineći ga tako jednim od osnovnih dodataka za kreiranje dinamičkih neigrivih likova. Dodatak je izgrađen na vrhu llama.cpp i llamafile biblioteka koje mu daju mogućnosti pokretanja modela i slanja upita modelu [15]. Dodatak zapravo služi kao most biblioteke llamafile, koja će

pokrenuti model na lokalnom računalu koristeći llama.cpp, i Unitya koji će slati upite na model i primati odgovore modela. Optimizacija pokretanja modela se postiže upotrebom grafičkih kartica ako su dostupne, osiguravajući glatko igranje i učinkovitu potrošnju resursa [16]. Dodatak Podržava većinu popularnih modela i besplatan je za osobnu i komercijalnu upotrebu. Za korištenje dodataka potrebno ga je prvo instalirati putem Unity Asset Storea ili GitHuba. Nakon instalacije i dodavanja dodatka u projekt pojaviti će se skup gotovih skripti i biblioteka koje se koriste za rad s modelima. Fleksibilne postavke skripti dodatka omogućuju razvojnim programerima višu kontrolu nad modelima.

3.4 Strojno učenje

Umjesto da budu programirani s fiksnim skupom pravila, modeli strojnog učenja uče uzorke iz podataka. Pojam strojno učenje je definirao Arthur Samuel 1953 kao vrstu algoritama i modela koji izvršavaju određene zadatke bez eksplicitnih uputa poboljšavaju rezultate izvođenja tijekom vremena [17]. Ovaj pristup koji se temelji na podacima omogućuje modelima strojnog učenja da se prilagode i daju zadovoljavajući rezultat na nove dosada neviđene ulazne podatke, što ih čini posebno korisnim za složene zadatke gdje bi tradicionalno programiranje bilo nepraktično ili nemoguće. Svestranost strojnog učenja dokazana je primjenom u raznim područjima poput poljoprivrede [18], zdravstva, financija, marketinga i autonomnih sustave, gdje pokreće tehnologije kao što su sustavi preporuka, otkrivanje prijevara i obrada prirodnog jezika [19].

Osnova kreiranja modela strojnog učenja uključuje obuku modela na skupu podataka, gdje model uči donositi predviđanja ili odluke na temelju tih podataka [20]. Ovaj proces obično uključuje više koraka od kojih prvi korak je prikupljanje i pred obrada podataka na kojima će se model učiti. Podaci se moraju očistiti, kako bi se uklonio bilo kakav šum ili nevažne informacije, i transformirati u prikladan format za obuku modela. Ovaj korak također uključuje tehnike poput normalizacije i standardizacije.

Nakon što su podaci prikupljeni i obrađeni potrebno je odabrati prikladni algoritam ili arhitekturu koja najbolje odgovara problemu koji je u pitanju. Odabir ovisi o čimbenicima kao što su priroda podataka, vrsta zadatka (klasifikacija, regresija, grupiranje itd.) i željena preciznost modela. Algoritmi učenja iterativno prilagođavaju svoje parametre kako bi se smanjila pogreška

između predviđanja i stvarnih ishoda. Proces prilagođavanja se obično radi korištenjem tehnika optimizacije poput algoritma gradijentnog spusta. Obuka je uobičajeno podijeljena u više epoha što znači da se procesu obuke prolazi kroz sve podatke više puta.

Nakon obuke, model se procjenjuje korištenjem različitih metrika kako bi se osiguralo da se dobro generalizira na nove, dosad nepoznate podatke. Uobičajene metrike procjene uključuju točnost, preciznost, F1 rezultat i druge specifične za domenu problema. Za dobivanje robusne procjene modela [20] spominje korištenje tehnika unakrsne validacije, kao što su k-fold i stratificirana unakrsna validacija.

Metode strojnog učenja obično se dijele na nadzirano učenje (engl. *Supervised Learning*), nenadzirano učenje (engl. *Unsupervised Learning*) i podržano učenje (engl. *Reinforcement Learning*).

3.4.1 Nadzirano učenje

U nadziranom učenju, model se trenira na označenom skupu podataka, što znači da za svaki ulazni podatak postoji neki izlazni podatak. Cilj je da model nauči preslikavanje vrijednosti postojećih ulaza na postojeće izlaze što se onda može koristiti za predviđanje izlaza novih nevidenih ulaznih podataka. Uobičajene primjene uključuju klasifikaciju (npr. prepoznavanje neželjene e-pošte) i regresiju (npr. predviđanje cijena kuća). Nadzirano učenje uključuje algoritme kao što su linearna regresija, metoda potpornih vektora (engl. *Support vector machine*) i neuronske mreže [20].

3.4.2 Nenadzirano učenje

Za razliku od nadziranog učenja, nenadzirano radi samo s ulaznim podacima bez odgovarajuće izlazne veličine. Model pokušava izvesti zaključak o prirodnoj strukturi prisutnoj unutar skupa podatkovnih točaka. Uobičajeni zadaci nenadziranog učenja su: grupiranje (npr. grupiranje kupaca na temelju ponašanja pri kupnji) i smanjenje dimenzionalnosti (npr. smanjenje broja značajki u skupu podataka). U nenadzirano učenje spadaju algoritmi k-srednjih vrijednosti, hijerarhijskog grupiranja, analize glavnih komponenti (PCA). Opisi navedenih i drugih algoritama nenadziranog učenja se nalaze u [21].

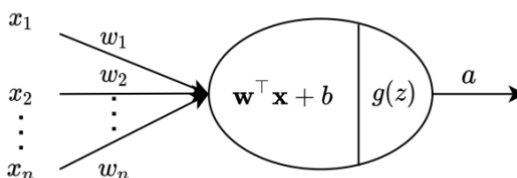
3.4.3 Podržano učenje

Podržano učenje (engl. reinforcement learning) radi na principu da agenti učenja teže maksimiziranju kumulativne nagrade [20]. Ove metode učenja mapiraju stanja svijeta na radnje koje agent treba poduzeti. Agent uči kroz interakciju s okolinom, primajući nagrade ili kazne na temelju svojih postupaka. Ova vrsta strojnog učenja je posebno korisna u scenarijima gdje je potrebno donošenje odluka tijekom vremena, kao što su robotika, igranje igrica (npr. AlphaGo [23]) i autonomna vožnja. Ključni algoritmi u podržanom učenju uključuju Q-učenje, duboke Q-mreže (engl. deep Q-networks) i metode gradijenta politike (engl. policy gradient methods).

3.5 Neuronske mreže

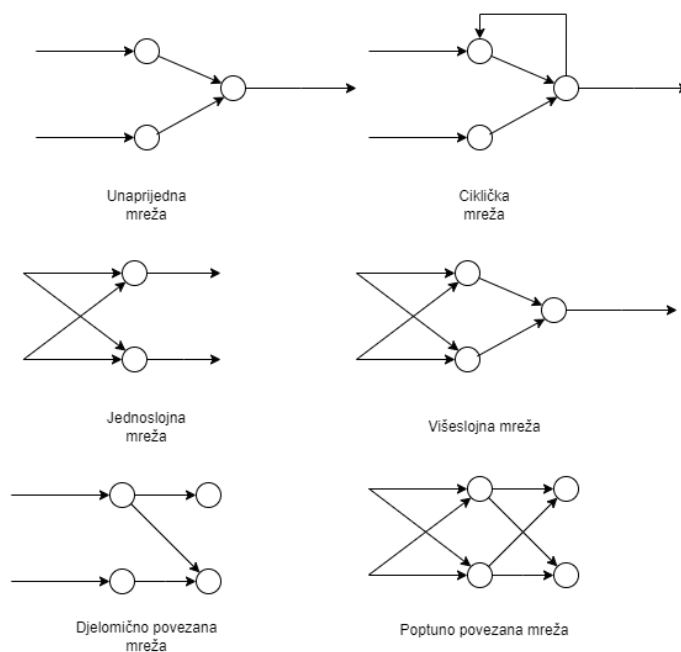
Neuronske mreže predstavljaju ključnu tehnologiju unutar strojnog učenja, koje je samo po sebi podskup šireg područja umjetne inteligencije. Inspirirane strukturom i funkcijom bioloških neuronskih mreža [24], te se mreže sastoje od slojeva međusobno povezanih čvorova ili neurona koji obrađuju ulazne podatke, i uče izvršavati složene zadatke prilagođavanjem veza između tih čvorova [25].

Neuroni su osnovne jedinice neuronske mreže jer obrađuju i prenose informacije. snove rada neurona što uključuje primanje ulaznih podatka, primjenjivanje linearne transformacije koristeći težine i bias, te primjenjivanje aktivacijske funkcije za stvaranje izlaza [25]. Vizualna reprezentacija neurona je vidljiva na slici 3.3.



Slika 3.3 Vizualna reprezentacija neurona [26]

Prema načinu spajanja neuronske mreže se dijele na unaprijedne (engl. *Feedforward neural network*) ili cikličke, jednoslojne ili višeslojne i djelomično ili potpuno povezane [26]. Razlika unaprijednih i cikličkih je prisutnost povratne veze, gdje cikličke mreže sadrže povratne veze dok unaprijedne ne. Navedena podjela je vidljiva na slici 3.4.



Slika 3.4 Podjela neuronskih mreža prema načinu spajanja [26]

Kako povećanje broja slojeva omogućuje mreži da izvršava kompleksnije zadatke, većina mreža u stvarnoj primjeni su višeslojnog tipa. Osnovna višeslojna neuronska mreža sastoji se od ulaznog sloja, jednog ili više skrivenih slojeva i izlaznog sloja.

Ulazni sloj je prvi sloj neuronske mreže i služi kao ulazna točka za ulazne podatke. Svaki neuron u ulaznom sloju odgovara jednoj značajki ulaznih podataka. Na primjer, u obradi slike, svaki neuron može predstavljati vrijednost piksela slike. Ulazni sloj jednostavno prosljeđuje ulazne vrijednosti sljedećem sloju bez ikakve transformacije.

Skriveni slojevi su među slojevi koji se nalaze između ulaznog i izlaznog sloja [27]. Neuronska mreža može imati više skrivenih slojeva, a dubina mreže određena je brojem tih slojeva. Svaki neuron u skrivenom sloju prima ulaz od neurona u prethodnom sloju, primjenjuje

linearnu transformaciju koristeći težine i bias, a zatim prosljeđuje rezultat linearne operacije kroz aktivacijsku funkciju.

Aktivacijske funkcije služe za određivanje izlaza neurona. Mogu biti linearne i nelinearne s time da su nelinearne popularniji izbor zato što uvode nelinearnost u mrežu što povećava sposobnost prilagođavanja mreže. Jedna od uobičajenih primjera aktivacijskih funkcija je sigmoidna funkcija (3-1) koja daje vrijednost između 0 i 1, što je čini prikladnom za probleme binarne klasifikacije [28].

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (3-1)$$

Zadnji sloj neuronske mreže je izlazni sloj čija je uloga davanje predviđanja mreže. Broj neurona u izlaznom sloju obično odgovara broju klasa u problemu klasifikacije ili broju izlaznih vrijednosti u problemu regresije. Izlaz neurona u ovom sloju često se transformira korištenjem aktivacijske funkcije prikladne za zadatak. Primjer takve funkcije je softmax aktivacijska funkcija (3-2) koja se koristi u problemima klasifikacije zato što pretvara neobrađene izlazne rezultate u vjerojatnosti čiji je zbroj jednak jedan [29].

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (3-2)$$

Neuronske mreže revolucionirale su područje strojnog učenja, omogućivši značajan napredak u raznim domenama kao što su računalni vid, obrada prirodnog jezika i prepoznavanje govora. Njihova sposobnost učenja složenih obrazaca i pravljenja točnih predviđanja iz golemih količina podataka dovela je do razvoja brojnih aplikacija, od autonomnih vozila i medicinske dijagnostike do personaliziranih preporuka i financijskih predviđanja. Među značajnijim postignućima su veliki jezični modeli kao što su OpenAI Gpt modeli [30], koji koriste duboke neuronske mreže za razumijevanje i generiranje ljudskog teksta, mijenjajući način na koji komuniciramo s tehnologijom i pristupamo informacijama.

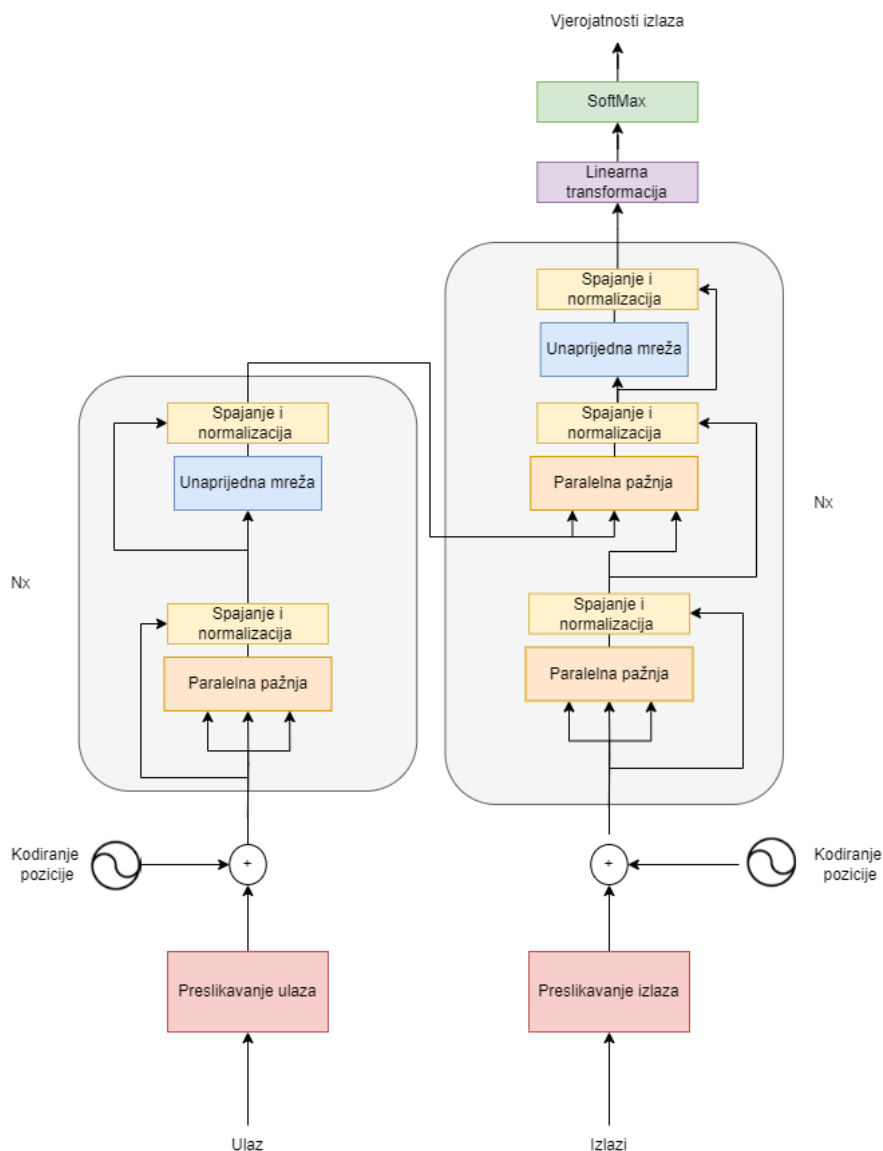
3.6 Veliki jezični modeli

Veliki jezični modeli predstavljaju značajan napredak u području obrade prirodnog jezika (engl. *Natural language processing* - *NLP*) i umjetne inteligencije. To su modeli koji imaju sposobnost razumijevanja, generiranja i interakcije s ljudskim jezikom na izuzetno sofisticiran način [31]. Arhitektura velikih jezičnih modela bazirana je na modelu transformatora, koji je postao standard zbog svoje učinkovitosti i djelotvornosti u rukovanju sekvencijalnim podacima.

3.6.1 Transformator model

Predstavili Vaswani i ostali u 2017 [32]. Transformator je arhitektura dubokog učenja koja koristi mehanizam samopažnje (engl. *self-attention*) za paralelnu obradu podataka, rješavajući ograničenje pronađeno u prethodnim arhitekturama neuronskih mreža koje su koristile računalno intenzivne ciklične neuronske mreže.

Model Transformatora izgrađen je na strukturi koder-dekoder (slika 3.5). Mnogi model koriste varijacije ovog okvira, fokusirajući se ili na koder (npr. BERT [33]) ili dekoder (npr. GPT).



Slika 3.5 Shematski prikaz tipičnog transformator modela [32]

Koder se sastoji od niza slojeva za kodiranje koji sekvencijalno obrađuju ulazne tokene, dok dekođer sadrži slojeve za dekodiranje koji iterativno obrađuju izlaz kodera zajedno s dosadašnjim tokenima generiranim tijekom dekodiranja [33].

Svaki sloj kodera generira kontekstualizirane reprezentacije tokena, pri čemu svaka reprezentacija „miješa“ informacije iz drugih ulaznih tokena putem mehanizma samopažnje. Svaki sloj dekodera sadrži dva podsloja pažnje: križnu samopažnju za inkorporiranje izlaza kodera (kontekstualizirane reprezentacije ulaznih tokena) i samopažnju za „miješanje“ informacija među ulaznim tokenima dekodera (tj. Tokenima generiranim do tog trenutka tijekom interakcije s modelom).

Slojevi kodera i dekodera također uključuju unaprjeđenu neuronsku mrežu za dodatnu obradu izlaza te sadrže rezidualne veze i korake normalizacije slojeva.

3.6.2 Transformator model u velikim jezičnim modelima

Transformator arhitektura je originalno bila osmišljena za prevođenje jezika, ali ispostavila se vrlo fleksibilna za druge svrhe, što je dovelo do revolucije u modernom strojnom učenju. Pokazatelj toga su modeli poput DALL·E 2 [34] koji kreira slike iz opisnog teksta i Whisper [35] koji pretvara zvuk u tekst.

Veliki jezični modeli koriste modifikacije transformator modela koje imaju mogućnosti generiranja novog teksta ovisno o ulaznom na način da konstanto predviđaju iduću riječ u tekstu kao što je prikazano na slici 3.6.



Slika 3.6 Koncept transformatora za generiranje teksta [36]

Svaki veliki jezični model koristi dodatnu varijantu te varijante, ali osnovna arhitektura je dijeljena među modelima.

Prvi korak u obradi teksta s transformatorskim modelom je tokenizacija, koja uključuje rastavljanje ulaznog teksta na manje jedinice koje se nazivaju tokeni [32]. Tokeni mogu biti riječi, dijelovi riječi ili čak znakovi, ovisno o korištenoj strategiji tokenizacije. Na primjer, rečenica

„Transformatori su super!“ može se tokenizirati u ["Transformatori ", "su", " super", "!"]. Tokenizacija omogućuje modelu rukovanje sekvencama ulaznog teksta promjenjive duljine i pretvaranje teksta u numerički format koji se može obraditi neuronskim mrežama. Svakom tokenu dodijeljen je jedinstveni vektor brojeva, pretvarajući ga u format koji model može razumjeti.

Nakon tokenizacije, svaki se token preslikava (engl. *embed*) na višedimenzionalni prostor dodavajući semantičke informacije o tokenima, omogućujući modelu da prepozna i obradi slične tokene na kontekstualno relevantan način. Svaki model ima matricu, tj. vokabular riječi koja sadrži preslikane koordinate tih riječi u višedimenzionalnom prostoru. U početku procesa treniranja te koordinate su nasumično popunjene i tijekom treniranja vrijednosti se mijenjaju ovisno o podacima. U slučaju nepostojanja tokena u vokabularu, postoji niz različitih rješenja. Jedan uobičajeni pristup je uporaba posebnog OOV tokena (npr. <NEPZ> za "nepoznato") kako bi se predstavljao bilo koji token koji nije u vokabularu. Na ovaj način se svi nepoznati tokeni preslikavaju u isti vektor. Drugi pristup je tokenizacija dijelova riječi gdje tehnike poput kodiranja parova bajtova (engl. *Byte-Pair Encoding*) ili WordPiece rastavljaju nepoznate tokene na manje dijelove ili znakove koji su dio vokabulara. Na primjer, ako riječ "nesreća" nije u vokabularu, ali „ne“ i "sreća" jesu, riječ se može tokenizirati u te dijelove, a njihovi vektori se mogu kombinirati [37].

Kako bi transformatori mogli razumjeti redosljed tokena u ulaznim podacima, vektoru preslikavanja se dodaje vektor kodiranja pozicije (engl. *position encoding*). Za svrhe kodiranja pozicije koriste se funkcije sinusa (3-3) i kosinusa (3-4) koje će generirati traženi kodirani vektor. U navedenim formulama *pos* predstavlja poziciju tokena u ulaznim podacima, *d* predstavlja dimenziju izlaznog vektora preslikavanja i varijabla *i* se koristi se za mapiranje u indekse stupaca.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \quad (3-3)$$

$$PE_{(pos,2i)} = \cos\left(\frac{pos}{10000^{2i/d}}\right) \quad (3-4)$$

Mehanizam samopažnje ključna je komponenta velikih jezičnih modela, omogućujući im da procijene važnost i relevantnost različitih tokena unutar niza. U samopažnji, svaki token u ulaznoj sekvenci transformira se u tri vektora: vektor upita Q, vektor ključa K i vektor vrijednosti V. Ovi se vektori izračunavaju pomoću matrica težine koje preslikavaju ulazne embeded vektore

u navedena tri prostora. Matrice težine predstavljaju jedan od parametara koji mijenja vrijednosti tijekom procesa učenja. Matematički operacija preslikavanja izgleda kao (3-5) gdje W^q , W^k i W^v predstavljaju matrice težine, a X predstavlja ulazni vektor.

$$Q = XW^q, \quad K = XW^k, \quad V = XW^v \quad (3-5)$$

Mehanizam samopažnje uspoređuje svaki vektor upita sa svim ključnim vektorima kako bi se izračunali rezultati pažnje koji ukazuju na relevantnost svakog tokena za trenutni token koji se obrađuje. Operacija je iskazana formulom (3-6) gdje $\sqrt{d_k}$ predstavlja faktor skaliranja koji pomaže stabilizirati gradijente sprječavanjem pretjerano velikih vrijednosti u funkciji softmax.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3-6)$$

Ovaj mehanizam omogućuje svakom tokenu da prati svaki drugi token u nizu, učinkovito integrirajući informacije s različitih pozicija na temelju njihove relevantnosti. Ova sposobnost modeliranja dugotrajnih ovisnosti i kontekstualnih odnosa ono je što samopažnju čini tako snažnom u rješavanju složenih jezičnih zadataka. Rezultat je bogat, kontekstno svjestan prikaz svakog tokena koji bilježi njegove vlastite značajke i odnose s drugim tokenima u nizu. Operacije samopažnje se odvijaju paralelno (engl. *MultiHead*) a izlazi tih operacija se spajaju i linearno transformiraju.

Nakon svakog bloka operacija samopažnje slijedi unaprijeđena neuronska mreža kroz koju prolaze podaci iz blokova samopažnje pružajući dodatnu nelinearnost i složenost mogućnostima obrade modela.

Da bi izlaz modela bio čitljiv ljudima potrebno ga je dekodirati u riječi, tj. transformirati izlaz modela u distribuciju vjerojatnosti preko vokabulara. Tipično postoji sloj projekcije koji će preslikati višedimenzionalne prikaze tokena u tokene iz vokabulara. Može se odabrati token s najvećom vjerojatnosti ali i nasumično uz faktor slučajnosti koji može ublažiti nasumičnost. Većina poznatih modela koristi nasumični pristup kako bi spriječili davanje uvijek istog izlaza za isti ulaz [38].

3.6.3 Phi-3

Phi-3 je inovativan veliki jezični model koji se ističe svojom sposobnošću da uravnoteži učinkovitost i performanse [39]. Zasnovan na transformatorskoj dekođer arhitekturi s fiksnom duljinom konteksta od 4K tokena, Phi-3 nudi brzu i pouzdanu obradu podataka uz očuvanje bogatog kontekstualnog razumijevanja. Za zadatke koji zahtijevaju šire kontekstualno razumijevanje, Phi-3 nudi i verziju s produženom duljinom konteksta Phi-3-mini-128K. Ova verzija modela ima duljinu konteksta od 128K tokena, što ga čini idealnim za rukovanje složenijim jezičnim zadacima. Uz 32 glave i 32 sloja, Phi-3 postiže optimalnu ravnotežu između složenosti modela i računalne učinkovitosti. To ga čini savršenim izborom za implementaciju na mobilnim uređajima, gdje su resursi ograničeni, te u kontekstu videoigara, gdje je brzina i pouzdanost obrade jezika ključna za fluidno korisničko iskustvo. Učinkovitost modela se može vidjeti u slici 3.7 koja prikazuje metrike phi – 3 modela zajedno s metrikama drugih popularnih modela.

	Phi-3-mini 3.8b	Phi-3-small 7b	Phi-3-medium 14b	Phi-2 2.7b	Mistral 7b	Gemma 7b	Llama-3-In 8b	Mixtral 8x7b	GPT-3.5 version 1106
MMLU (5-Shot) [HBK*21]	68.8	75.7	78.0	56.3	61.7	63.6	66.5	70.5	71.4
HellaSwag (5-Shot) [ZHB*19]	76.7	77.0	82.4	53.6	58.5	49.8	71.1	70.4	78.8
ANLI (7-Shot) [NWD*20]	52.8	58.1	55.8	42.5	47.1	48.7	57.3	55.2	58.1
GSM-8K (8-Shot; CoT) [CKB*21]	82.5	89.6	91.0	61.1	46.4	59.8	77.4	64.7	78.1
MedQA (2-Shot) [JFO*20]	53.8	65.4	69.9	40.9	50.0	49.6	60.5	62.2	63.4
AGIEval (0-Shot) [ZCG*20]	37.5	45.1	50.2	29.8	35.1	42.1	42.0	45.2	48.4
TriviaQA (5-Shot) [JCWZ17]	64.0	58.1	73.9	45.2	75.2	72.3	67.7	82.2	85.8
Arc-C (10-Shot) [CCE*18]	84.9	90.7	91.6	75.9	78.6	78.3	82.8	87.3	87.4
Arc-E (10-Shot) [CCE*18]	94.6	97.0	97.7	88.5	90.6	91.4	93.4	95.6	96.3
PIQA (5-Shot) [BZGC19]	84.2	86.9	87.9	60.2	77.7	78.1	75.7	86.0	86.6
SociQA (5-Shot) [BZGC19]	76.6	79.2	80.2	68.3	74.6	65.5	73.9	75.9	68.3
BigBench-Hard (3-Shot; CoT) [SRR*22, SSS*22]	71.7	79.1	81.4	59.4	57.3	59.6	51.5	69.7	68.32
WinoGrande (5-Shot) [SLBBC19]	70.8	81.5	81.5	54.7	54.2	55.6	65.0	62.0	68.8
OpenBookQA (10-Shot) [MCKS18]	83.2	88.0	87.4	73.6	79.8	78.6	82.6	85.8	86.0
BoolQ (2-Shot) [CLC*19]	77.2	84.8	86.5	–	72.2	66.0	80.9	77.6	79.1
CommonSenseQA (10-Shot) [THLB19]	80.2	80.0	82.8	69.3	72.6	76.2	79.0	78.1	79.6
TruthfulQA (10-Shot; MC2) [LHE22]	65.0	70.2	75.1	–	53.0	52.1	63.2	60.1	85.8
HumanEval (0-Shot) [CTJ*21]	58.5	61.0	62.2	59.0	28.0	34.1	60.4	37.8	62.2
MBPP (3-Shot) [AON*21]	70.0	71.7	75.2	60.6	50.8	51.5	67.7	60.2	77.8
Average	71.2	75.7	78.5	–	61.2	61.7	69.4	69.8	74.3
GPQA (2-Shot; CoT) [RHS*20]	32.8	34.3	–	–	–	–	–	–	29.0
MT Bench (2 round ave.) [ZCS*20]	8.38	8.70	8.91	–	–	–	–	–	8.35

Slika 3.7 Usporedba Phi – 3 modela s ostalim popularnim [39]

Phi-3 se trenira na strogo odabranim podacima koji uključuju filtrirane podatke s interneta i sintetički generirani sadržaj. Skupovi podataka pružaju raznolik i informativnu osnovu za razumijevanje jezika i sposobnost razmišljanja modela.

Razvoj Phi-3 modela ne završava početnim treniranjem. Model se kontinuirano poboljšava kroz niz post-trening tehnika, uključujući nadzirano podešavanje (engl. *fine tuning*) i direktnu optimizaciju preferencija (engl. *direct preference optimization*). Posebno važnu ulogu u ovom procesu ima učenje iz ljudskih povratnih informacija (engl. *Reinforcement Learning from Human Feedback - RLHF*). Zahvaljujući RLHF-u, Phi-3 može učiti iz stvarnih interakcija s korisnicima. Model analizira povratne informacije, bilo eksplicitne ili implicitne, kako bi prilagodio svoje odgovore i poboljšao kontekstualnu točnost i relevantnost tijekom vremena.

3.6.4 PEFT

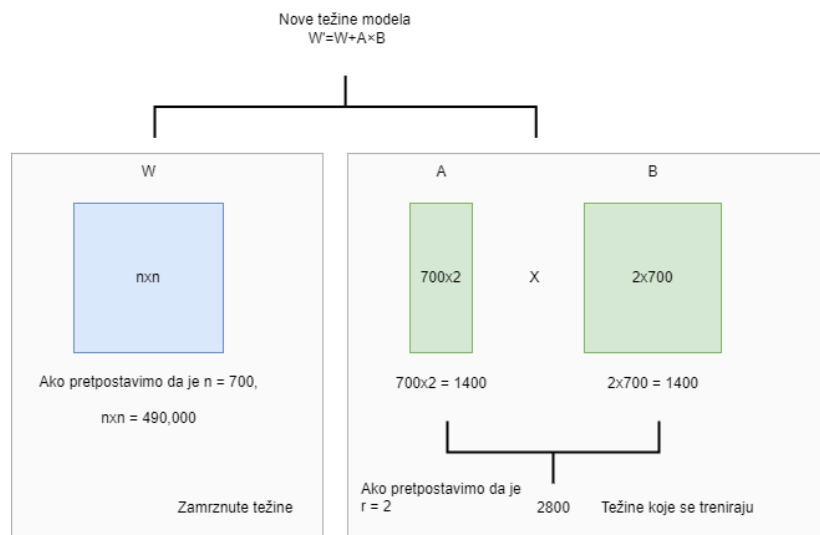
Brzi napredak LLM-a doveo je do modela s milijardama parametara, kao što su GPT-4 i BERT. Istraživanje je pokazalo da zbog velike količine parametara podešavanje svih parametara modela za specifične upotrebe zahtijeva puno računalnih resursa [40]. Zbog toga metoda podešavanja modela koje zahtijevaju ažuriranje svih parametara modela su nepraktične i računalno skupe.

Učinkovito podešavanje parametara (engl. *Parameter Efficient Fine Tuning - PEFT*) rješava te izazove omogućavajući učinkovite prilagodbe specifičnim zadacima. Temeljna ideja PEFT-a je ažurirati parametre specifične za zadatak, a pritom zadržati većinu parametara modela netaknutim. Ovaj pristup značajno smanjuje računalne zahtjeve čineći taj pristup idealnim za okruženja s ograničenim računalnim resursima [41].

3.6.5 LoRA

Jedna od tehnika korištenih u prilagodbi velikih jezičnih modela je prilagodba niskog ranga (engl. *Low-Rank Adaptation - LoRA*). Izvorni rad opisuje da umjesto ažuriranja parametara cijelog modela, mogu se uvesti matrice niskog ranga kako bi se modificirale izvorne matrice težine [42]. To se postiže dekompozicijom težinskih matrica na umnožak dviju manjih matrica (slika 3.8), omogućujući da se tijekom podešavanja modela ažuriraju samo te komponente niskog ranga.

Ovaj pristup značajno smanjuje broj parametara koje je potrebno ažurirati, čime se smanjuju računalni resursi potrebni za podešavanje. Korištenjem ažuriranja niskog ranga, održava se prethodno naučeno znanje modela dok se podešava za specifične upotrebe, pružajući snažno i resursno učinkovito rješenje za implementaciju velikih jezičnih modela u različitim praktičnim scenarijima.



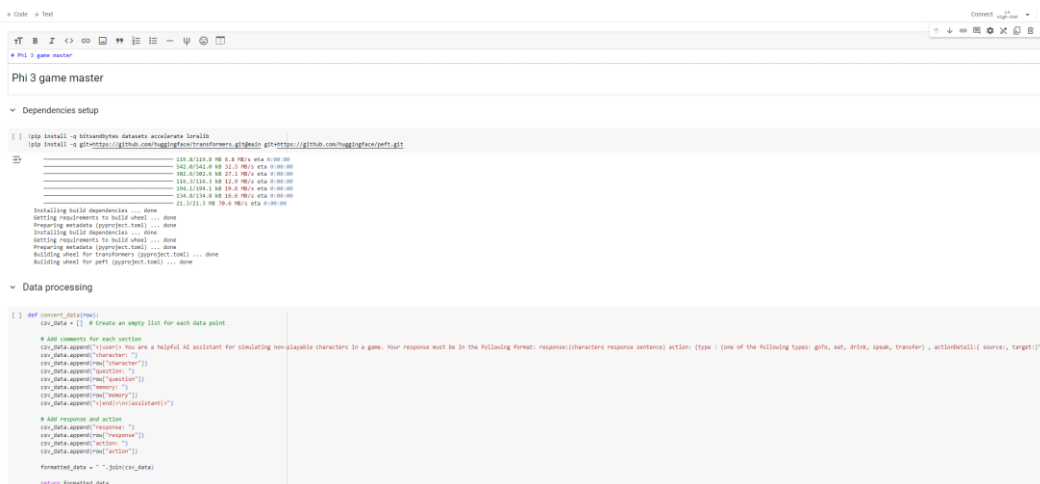
Slika 3.8 *Primjer dekompozicije težinske matrice*

3.7 Google colab

Google Colab ili "Collaboratory" [43] je platforma na oblaku koja pruža Jupyter okruženje za pisanje i izvršavanje Python kôda. Colab podržava cijeli Python [44] ekosustav, omogućujući korisnicima uvoz biblioteka i okvira kao što su TensorFlow, PyTorch i scikit-learn. Platforma nudi sposobnost suradnje u stvarnom vremenu, omogućujući većem broju korisnika istovremeno uređivanje i pokretanje koda. Dodatno, Google Colab pruža pohranu podataka putem integracije s Google diskom olakšavajući korisnicima spremanje modela. Usluga na oblaku eliminira potrebu za lokalnim postavljanjem i održavanjem računalnih resursa, što čini Colab učinkovitim i svestranim alatom za obrazovnu i profesionalnu upotrebu. Razlog upotrebe je pristup jačim računalnim resursima koji su potrebni za pripremanje modela.

3.7.1 Jupyter

Jupyter [45] je projekt otvorenog koda koji pruža svestrano sučelje temeljeno na webu za interaktivno računalstvo. Glavna komponenta Jupytera su bilježnice (Slika 3.8), koje korisnicima omogućuju stvaranje i dijeljenje dokumenata koji sadrže kod, jednadžbe, vizualizacije i tekst. Ovo interaktivno okruženje podržava više od 40 programskih jezika, uključujući Python, R i Julia, što ga čini vrijednim alatom za analizu podataka, znanstveno istraživanje i obrazovanje. Jupyterova sposobnost da integrira izvršavanje koda s dokumentacijom i vizualizacijom olakšava proces razvoja i kolaborativni rad.



```
Code | Text | Comment

Phi 3 game master

Dependencies setup

! pip install -q bitsandbytes datasets accelerate loralib
! pip install -q git+https://github.com/huggingface/transformers.git@2e21-gf+https://github.com/huggingface/peft.git

-----
110.0/111.0 MB  0.0 MB/s  eta 0:00:00
242.0/242.0 MB 23.5 MB/s  eta 0:00:00
582.0/582.0 MB 27.2 MB/s  eta 0:00:00
134.0/141.0 MB 12.0 MB/s  eta 0:00:00
186.0/194.0 MB 18.0 MB/s  eta 0:00:00
134.0/134.0 MB 16.0 MB/s  eta 0:00:00
21.0/21.0 MB 76.4 MB/s  eta 0:00:00

Installing build dependencies ... done
Detecting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Installing build dependencies ... done
Installing build wheel ... done
Setting up dependencies ... done
Preparing metadata (pyproject.toml) ... done
Building wheel for transformers (pyproject.toml) ... done
Building wheel for peft (pyproject.toml) ... done

Data processing

! def convert_data(row):
    csv_data = [] # Create an empty list for each data point

    # Add comments for each section
    csv_data.append({"user": "You are a helpful AI assistant for simulating non-playable characters in a game. Your response must be in the following format: response:(characters response sentence) action: (type | one of the following types: gifts, eat, drink, speak, transfer) , actionDetail:[ source, target]?"])

    csv_data.append("character: ")
    csv_data.append("transfer: ")
    csv_data.append("position: ")
    csv_data.append("role: transaction")
    csv_data.append("memory: ")
    csv_data.append("memory: ")
    csv_data.append("memory: ")
    csv_data.append("inventory: ")
    csv_data.append("inventory: ")

    # Add response and action
    csv_data.append("response: ")
    csv_data.append("response: ")
    csv_data.append("action: ")
    csv_data.append("action: ")
    csv_data.append("action: ")

    formatted_data = " ".join(csv_data)

    return formatted_data
```

Slika 3.8 Primjer Jupyter bilježnice

3.7.2 Hugging Face Hub

Hugging Face Hub [46] je platforma za dijeljenje, otkrivanje i korištenje unaprijed treniranih modela strojnog učenja. Sadrži veliko spremište modela, skupova podataka i drugih resursa koje doprinosi globalna zajednica, potrebnih za zadatke strojnog učenja. Integracija Hugging Facea s bibliotekama kao što su Transformers i Datasets pojednostavljuje implementaciju i podešavanje modela. Kolaborativna svojstva platforme omogućuju korisnicima da učitaju modele, prate eksperimente i spremne verzije modela na platformu. Navedene sposobnosti platforme omogućavaju korisnicima da znatno ubrzaju projekte vezane uz strojno učenje.

3.7.3 Pytorch

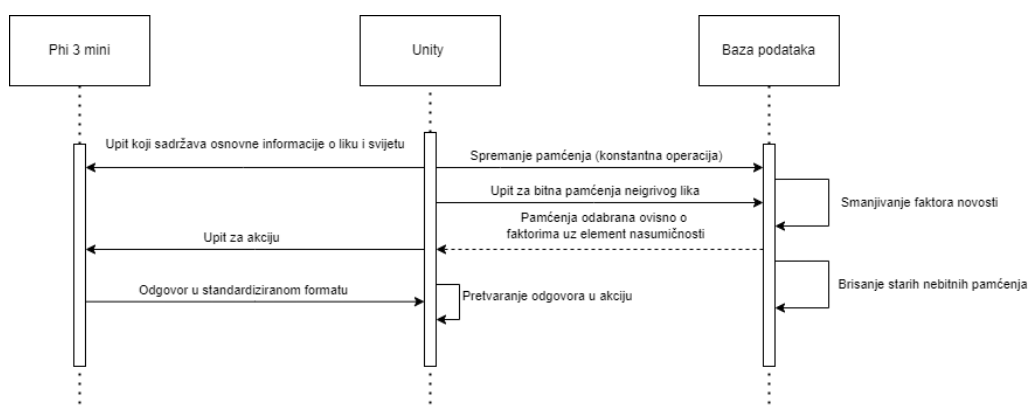
PyTorch [47] je okvir za duboko učenje otvorenog koda temeljen na biblioteci torch. Pruža fleksibilnu i učinkovitu platformu za izgradnju i treniranje neuronskih mreža. Podržava GPU ubrzanje i ima bogat ekosustav biblioteka koje dodavaju dodatne mogućnosti za specifične potrebe. Navedene značajke čine PyTorch jednu ključnih alata za rad s modelima strojnog učenja.

4. IMPLEMENTACIJA RJEŠENJA

Ovo poglavlje će opisati koncept rješenja te korake implementacije predloženog rješenja.

4.1 Koncept rješenja

Rješenje se temelji na modificiranju velikog jezičnog modela podešavanjem ili inženjerstvom ulaza [48] na način da Unity skripte mogu izvući informaciju iz izlaza modela te pretvoriti ih u akcije. Veliki jezični model će simulirati mozak neigrivog lika generirajući dinamične odgovore i akcije lika koje su u skladu s mišljenjima i osobinama lika. Koncept rješenja prikazan je sekvencijalnim dijagramom koji se nalazi na slici 4.1.



Slika 4.1 Sekvencijalni dijagram koncepta rješenja

Unity će koristiti bazu podataka u koju će se konstantno spremati informacije o neigrivim likovima, kao što su razina žeđi, promatranja lika, koje je doba dana, i druge relevantne podatke koji utječu na ponašanje lika. Svaki neigrivi lik će biti povezan s instancom modela koja će služiti kao njegov mozak. Na početku, prvi ulaz u model koji će svaki neigrivi lik pozvati bit će opsežan skup podataka o svijetu i liku, kako bi model imao temeljan kontekst za donošenje odluka.

Osim ulaza za podatke o svijetu i liku, Unity će također češće periodično slati upite modelu koji se sastoji od pitanja "What to do now" zajedno s nekoliko zapisa iz memorije lika, koji uključuju važne informacije potrebne za donošenje odluka. Model će na temelju tih ulaznih

podataka generirati odgovor u standardiziranom formatu, specifičan za trenutnu situaciju lika, s akcijom koju lik treba poduzeti. Nakon što je model dao odgovor Unity će ga pročitati i pretvoriti u konkretne akcije koje lik treba izvršiti. Ako je lik u interakciji s drugim likom, ulaz modela će uključivati i izlaz modela drugog lika, omogućujući koordiniranu interakciju između njih.

Zapisi iz memorije će se dohvaćati iz Sqlite [49] baze podataka zbog efikasnijeg spremanja podataka. Čitanje i pisanje iz datoteka su resursno intenzivne operacije, za potrebe realistične memorije takvih operacija će biti mnogo. Iz tog razloga umjesto korištenja Unity rješenja za spremanje podataka koristiti će se Sqlite zbog lagane integracije s Unityom i zbog efikasnosti baza podataka.

Ovaj pristup omogućit će svakom neigrivom liku da reagira na dinamičke promjene u svom okruženju na inteligentan i kontekstualno svjestan način, koristeći moć naprednih jezičnih modela za donošenje odluka.

4.2 Podešavanje velikog jezičnog modela

Prema konceptu rješenja, prvi korak pri implementaciji je priprema velikog jezičnog modela. Ovaj korak uključuje nekoliko ključnih aktivnosti koje osiguravaju da je model spreman za integraciju i upotrebu unutar Unity okruženja. Proces započinje izborom odgovarajućeg modela, kao što je phi 3 mini 4k instruct, koji će se koristiti kao osnova za daljnje prilagodbe.

4.2.1 Priprema skupa podataka

Nakon što je model odabran idući korak je pripremiti podatke koji će se koristiti za prilagođavanje modela. Slika 4.2 prikazuje dio skupa podataka za podešavanje. Podaci su kreirani u JSON [50] obliku radi lakšega procesuiranja skupa u idućem koraku.

```

▼ array [36]
  ▼ 0 [5]
    ▼ character {3}
      name : Fiona
      age : 29
      description : Fiona is a skilled farmer who tends to her crops with care and expertise.
      question : what to do now
    ▼ memory [5]
      ► 0 [4]
      ► 1 [4]
      ► 2 [4]
      ► 3 [5]
      ► 4 [5]
      response : Today calls for a day of labor on the farm. Whether it's planting seeds, tending to crops, or preparing meals from fresh produce, there's much to be done.
    ▼ action (2)
      type : goTo
      target : Fionas House
  ▼ 1 [5]
    ▼ character {3}
      name : Eldric
      age : 34
      description : Eldric, the village blacksmith, forges weapons and armor with unmatched skill.
      question : what to do now
    ▼ memory [3]
      ► 0 [4]
      ► 1 [4]
      ► 2 [4]
      response : Today, Eldric plans to work on a special project: forging a sword said to be as light as a feather but as strong as a dragon's scale.
    ► action (2)
  ▼ 2 [5]
    ▼ character {3}
      name : Mira
      age : 26
      description : Mira is a healer known for her knowledge of herbs and potions.
      question : what to do now

```

Slika 4.2 *Primjer zapisa skupa podataka*

Kreirani podaci se zatim učitavaju iz JSON datoteke u Pandas DataFrame [51], što omogućava lakšu manipulaciju podacima. Dataframe objekt omogućava automatsku konverziju objekta u csv fromat (slika 4.3) koji je potreban kako bi se olakšalo učitavanje podataka kao dataset objekt koristeći biblioteku Hugging Face Datasets. Podaci moraju biti u dataset tipu objekta zato što taj tip objekta predstavlja jedan od argumenata funkcije za podešavanje modela.

character	question	memory	response	action
{'name': 'Fiona', 'age': 29, 'description': 'Fiona is a skilled farmer who tends to her crops with care and expertise.'}	what to do now	[[{'date_created': '1100-02-02 08:00:00', 'reccency': 0.8, 'importance': 0.7, 'activity': 'Harvesting a bountiful crop of ripe tomatoes from her garden'}, {'date_created': '1100-02-01 10:00:00', 'reccency': 0.7, 'importance': 0.6, 'activity': 'Discovering a new hybrid plant species with unique medicinal properties'}, {'date_created': '1100-01-31 12:00:00', 'reccency': 0.6, 'importance': 0.5, 'activity': 'Preparing a hearty stew using fresh ingredients from her farm'}, {'date_created': '1100-01-30 14:00:00', 'reccency': 0.5, 'importance': 0.4, 'relevance': 0.6, 'activity': 'Constructing a new greenhouse to expand her farming operations'}, {'date_created': '1100-01-29 16:00:00', 'reccency': 0.4, 'importance': 0.3, 'relevance': 0.5, 'activity': 'Teaching local children about the importance of sustainable agriculture'}]]	Today calls for a day of labor on the farm. Whether it's planting seeds, tending to crops, or preparing meals from fresh produce, there's much to be done.	{'type': 'goTo', 'target': 'Fionas House'}
{'name': 'Eldric', 'age': 34, 'description': 'Eldric, the village blacksmith, forges weapons and armor with unmatched skill.'}	what to do now	[[{'date_created': '1100-02-03 09:00:00', 'reccency': 0.9, 'importance': 0.8, 'activity': 'Crafting a durable set of armor for the town guard'}, {'date_created': '1100-02-02 11:00:00', 'reccency': 0.8, 'importance': 0.7, 'activity': 'Repairing a plow for Fiona's farm'}, {'date_created': '1100-02-01 13:00:00', 'reccency': 0.7, 'importance': 0.6, 'activity': 'Teaching an apprentice the art of sword making'}]]	Today, Eldric plans to work on a special project: forging a sword said to be as light as a feather but as strong as a dragon's scale.	{'type': 'EquipItem', 'target': 'Leather Apron'}
{'name': 'Mira', 'age': 26, 'description': 'Mira is a healer known for her knowledge of herbs and potions.'}	what to do now	[[{'date_created': '1100-02-04 10:00:00', 'reccency': 1.0, 'importance': 0.9, 'activity': 'Gathering rare herbs in the deep forest'}, {'date_created': '1100-02-03 12:00:00', 'reccency': 0.9, 'importance': 0.8, 'activity': 'Brewing a potion to cure a mysterious illness afflicting the village'}, {'date_created': '1100-02-02 14:00:00', 'reccency': 0.8, 'importance': 0.7, 'activity': 'Consulting ancient texts to discover new healing techniques'}]]	Mira decides to visit the village square to offer her healing services and share her knowledge of preventative care.	{'type': 'talkTo', 'target': 'Villagers'}
{'name': 'Garrick', 'age': 31, 'description': 'Garrick is a seasoned adventurer, always in search of treasure and glory.'}	what to do now	[[{'date_created': '1100-02-05 08:00:00', 'reccency': 1.0, 'importance': 0.9, 'activity': 'Mapping an unexplored cave rumored to hold ancient artifacts'}, {'date_created': '1100-02-04 10:00:00', 'reccency': 0.9, 'importance': 0.8, 'activity': 'Negotiating with a dragon to secure passage through its territory'}, {'date_created': '1100-02-03 12:00:00', 'reccency': 0.8, 'importance': 0.7, 'activity': 'Training with the village guard to hone his combat skills'}]]	Garrick plans to gather a party for an expedition into the Dark Woods, rumored to be the site of a fallen star.	{'type': 'pickUpItem', 'target': 'Compass'}
{'name': 'Sylvia', 'age': 28, 'description': 'Sylvia is a skilled archer who roams the forest in search of adventure.'}	what to do now	[[{'date_created': '1100-02-06 09:00:00', 'reccency': 1.0, 'importance': 0.9, 'activity': 'Tracking a band of goblins that have been raiding nearby villages'}, {'date_created': '1100-02-05 11:00:00', 'reccency': 0.9, 'importance': 0.8, 'activity': 'Competing in an archery contest to win a prize'}, {'date_created': '1100-02-04 13:00:00', 'reccency': 0.8, 'importance': 0.7, 'activity': 'Exploring a hidden cave filled with ancient ruins'}]]	Sylvia decides to venture into the forest to practice her archery skills and test her accuracy against moving targets.	{'type': 'EquipItem', 'target': 'Bow and Arrows'}
{'name': 'Roland', 'age': 32, 'description': 'Roland is a knight known for his valor and strength, serving the kingdom with loyalty.'}	what to do now	[[{'date_created': '1100-02-07 08:00:00', 'reccency': 1.0, 'importance': 0.9, 'activity': 'Spotted eastern watchtower at sunrise'}, {'date_created': '1100-02-06 12:00:00', 'reccency': 0.9, 'importance': 0.8, 'activity': 'Hungry level at 40% around noon, shared a hearty meal with fellow knights'}, {'date_created': '1100-02-05 15:00:00', 'reccency': 0.8, 'importance': 0.7, 'activity': 'Sir Eldric said, 'The forge's fire burns as fiercely as our spirits in battle.'"}]]	Roland decides to patrol the kingdom's outskirts on horseback, ensuring the safety of the realm.	{'type': 'EquipItem', 'target': 'Sword and Shield'}
{'name': 'Lilith', 'age': 24, 'description': 'Lilith is a cunning sorceress, weaving spells with ease and grace, feared and respected by many.'}	what to do now	[[{'date_created': '1100-02-07 10:00:00', 'reccency': 1.0, 'importance': 0.9, 'activity': 'Spotted at the ancient library, delving into forbidden tomes'}, {'date_created': '1100-02-06 14:00:00', 'reccency': 0.9, 'importance': 0.8, 'activity': 'Thirst level at 30% in the afternoon, concocted a potion to quench thirst and enhance focus'}, {'date_created': '1100-02-05 17:00:00', 'reccency': 0.8, 'importance': 0.7, 'activity': 'The librarian said, 'Beware the knowledge that comes without a price.'"}]]	Lilith decides to experiment with a new spell that could control the elements, seeking solitude in her tower.	{'type': 'useItem', 'target': 'Spellbook'}
{'name': 'Thorne', 'age': 40, 'description': 'Thorne, a grizzled veteran of many battles, now serves as a trainer for the kingdom's new recruits.'}	How can I improve my swordsmanship?	[[{'date_created': '1204-03-14 12:00:00', 'reccency': 1.0, 'importance': 0.9, 'activity': 'Spotted training recruits in the castle courtyard at dawn'}, {'date_created': '1204-03-14 12:00:00', 'reccency': 0.9, 'importance': 0.8, 'activity': 'Fatigue level at 50% after a rigorous training session, took a moment to rest under the old oak'}, {'date_created': '1204-03-13 16:00:00', 'reccency': 0.8, 'importance': 0.7, 'activity': 'A recruit said, 'Thorne's guidance is harsh but fair, leading us towards becoming true warriors.'"}]]	Start by focusing on defense and footwork, then practice with a partner to improve your reflexes and timing.	{'type': 'talkTo', 'target': 'Recruits'}

Slika 4.3 *Dio skupa podataka u CSV formatu.*

Kako jezični modeli primaju jedan blok teksta kao ulaz potrebno je dodatno procesirati podatke na način da se cijeli jedan zapis pretvori u jedan string objekt. U te svrhe kreirana je funkcija *convert_data* (slika 4.4).

```
[ ] def convert_data(row):
    csv_data = [] # Create an empty list for each data point

    # Add comments for each section
    csv_data.append("<|system|>\n character:")
    csv_data.append(row["character"])
    csv_data.append("memory: ")
    csv_data.append(row["memory"])
    csv_data.append("<|end|>\n<|user|>\n")
    csv_data.append("question: ")
    csv_data.append(row["question"])
    csv_data.append("<|end|>\n<|assistant|>\n")

    # Add response and action
    csv_data.append("response: ")
    csv_data.append(row["response"])
    csv_data.append("action: ")
    csv_data.append(row["action"])

    formatted_data = " ".join(csv_data)

    return formatted_data
```

Slika 4.4 Funkcija za spajanje zapisa u jedan string

Osim spajanja stupaca u jedan string funkcija također dodaje oznake specifične za model koji se trenira, u slučaju phi 3 instruct modela to su oznake: <|system|>, <|end|>, <|assistant|>, <|user|>.

4.2.2 Primjena LoRa tehnike

Nakon što su podaci pripremljeni i pretvoreni u odgovarajući format, sljedeći korak je učitavanje prethodno obučenog jezičnog modela phi 3 Phi-3-mini-128k-instruct. To uključuje korištenje biblioteke Hugging Face Transformers za učitavanje modela i njegovog tokenizatora .

Za učinkovito treniranje modela s ograničenim računalnim resursima koristeći LoRa tehniku, bitno je zamrznuti izvorne težine prethodno obučenog modela. To znači da će se moći trenirati samo novo dodani parametri, dok ostatak modela ostaje nepromijenjen. Funkcionalnost je ostvarena programskim kôdom na slici 4.5. Osim zamrzavanja kôd na slici 4.5 pretvara određene

parametre u 32-bitni format s pomičnim zarezom, omogućuje kontrolne točke gradijenta, te osigurava da se izlaz modela pretvara u float32 za učinkovitije treniranje.

```
[ ] for param in model.parameters():
    param.requires_grad = False
    if param.ndim == 1:
        param.data = param.data.to(torch.float32)

model.gradient_checkpointing_enable()
model.enable_input_require_grads()

class CastOutputToFloat(nn.Sequential):
    def forward(self, x): return super().forward(x).to(torch.float32)
model.lm_head = CastOutputToFloat(model.lm_head)
```

Slika 4.5 Programski kôd za zamrzavanje težina modela

Za kreiranje LoRa adaptera koristi se metoda `get_peft_model` iz Hugging Face Peft biblioteke. Metoda za argumente prima objekt modela te objekt konfiguracije kojim se mogu kontrolirati parametri LoRa adaptera (slika 4.6). Na slici 4.6 je također vidljiv broj parametara koja će se podešavati prilikom procesa treniranja te ukupan broj parametara modela, što je dokaz resursne efikasnosti korištenja PEFT tehnike.

```
def print_trainable_parameters(model):
    """
    Prints the number of trainable parameters in the model.
    """
    trainable_params = 0
    all_param = 0
    for _, param in model.named_parameters():
        all_param += param.numel()
        if param.requires_grad:
            trainable_params += param.numel()
    print(
        f"trainable params: {trainable_params} || all params: {all_param} || trainable%: {100 * trainable_params / all_param}"
    )

[ ] from peft import LoraConfig, get_peft_model
target_modules= ['k_proj', 'q_proj', 'v_proj', 'o_proj', "gate_proj", "down_proj", "up_proj"]
config = LoraConfig(
    r=16,
    lora_alpha=32,
    target_modules=target_modules,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, config)
print_trainable_parameters(model)

trainable params: 8912896 || all params: 3829992448 || trainable%: 0.23271314815918928
```

Slika 4.6 Programski kôd za postavljanje LoRa adaptera

4.2.3 Treniranje modela

Uz pripremljene podatke i dodanim LoRa adapterima, može se pokrenuti proces treniranja modela. Proces uključuje podešavanje parametara treniranja te pozivanje metode za treniranje modela. Slika 4.7 prikazuje programski kôd koji se koristi za realizaciju procesa treniranja modela. Kôd uključuje funkcionalnosti za treniranje modela, kao i ispis trenutačnog stanja tijekom treniranja. Ovaj ispis prikazuje trenutačni korak u fazi treniranja, zajedno s informacijama o gubitku u tom koraku, što omogućuje praćenje napretka i performansi modela tijekom cijelog procesa.

```
import transformers
from datasets import load_dataset

instruction_template = "<|system|>\n"
response_template = "<|assistant|>\n"
collator = DataCollatorForCompletionOnlyLM(instruction_template=instruction_template, response_template=response_template, tokenizer=tokenizer, mlm=False)

training_arguments = TrainingArguments(
    output_dir="./results",
    num_train_epochs=1,
    per_device_train_batch_size=4,
    gradient_accumulation_steps=1,
    save_steps=5,
    logging_steps=5,
    learning_rate=2e-4,
    weight_decay=0.001,
    fp16=True,
    bf16=False,
    max_grad_norm=0.3,
    max_steps=50,
    warmup_ratio= 0.03,
    group_by_length=True,
    lr_scheduler_type="constant",
    report_to="tensorboard"
)

trainer = SFTTrainer(
    model=model,
    train_dataset=data,
    peft_config=config,
    dataset_text_field="prediction",
    max_seq_length=None,
    tokenizer=tokenizer,
    args=training_arguments,
    packing=False,
    data_collator=collator
)
trainer.train()
```

```
/usr/local/lib/python3.10/dist-packages/trl/trainer/utils.py:162: UserWarning: The pad_token_id and eos_token_id values of this tokenizer are identical. If you are using a custom tokenizer, this is likely not intended. Please use a different eos_token_id.
warnings.warn(
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_deprecation.py:100: FutureWarning: Deprecated argument(s) used in '__init__': dataset_text_field
Deprecated positional argument(s) used in SFTTrainer, please use the SFTConfig to set these arguments instead.
warnings.warn(message, FutureWarning)
/usr/local/lib/python3.10/dist-packages/trl/trainer/sft_trainer.py:289: UserWarning: You didn't pass a `max_seq_length` argument to the SFTTrainer, this will default to the tokenizer's max_seq_length.
warnings.warn(
/usr/local/lib/python3.10/dist-packages/trl/trainer/sft_trainer.py:318: UserWarning: You passed a `dataset_text_field` argument to the SFTTrainer, the value of which is None. This argument is deprecated and will be removed in a future version.
warnings.warn(
Map: 100% ██████████ 36/36 [00:00<00:00, 462.69 examples/s]
/usr/local/lib/python3.10/dist-packages/trl/trainer/sft_trainer.py:408: UserWarning: You passed a tokenizer with `padding_side` not equal to `right` to the SFTTrainer. This might lead to unexpected behavior.
warnings.warn(
max_steps is given, it will override any value given in num_train_epochs
[50/50 02:02, Epoch 5/6]
```

Slika 4.7 Programski kôd za treniranje modela

Zadnji korak pripreme modela je spajanje LoRa adaptera u originalni model te spremanje tog modela lokalno ili na platformi Hugging face Hub. Hugging Face nudi metode za korištenje modela bez spajanja ali za svrhe korištenja u Unityu, potreban je model s svim parametrima. Spajanje se obavlja gotovom metodom *merge_and_unload*, a spremanje na platformu s metodom *push_to_hub*.

4.2.4 Završni koraci podešavanja

Nakon što je model spremljen lokalno ili eksterno spreman je za pretvorbu u format koji će se moći koristiti u Unityu. Ali prije toga model testira kako bi se provjerila uspješnost prilagođavanja modela. Način na koji će se prilagođeni model testirati je slanje upita modelu te evaluacija odgovora na način jeli odgovor u traženom formatu i ima li smisla. Kôd za testiranje je vidljivi na slici 4.8. Za korištenje modela u Unityu model se treba pretvoriti u prikladni format što se ostvaruje pokretanjem skripte llama.cpp/convert.py (ime modela).

```

batch = tokenizer("""
<system>
You are a helpful AI assistant for simulating non-playable characters in a fantasy RPG. Simulate response for following character: {'name': 'Eldrin', 'age': 356, 'description': 'A wise and powerful wizard, known for his vast knowledge of arcane }
</system>
<user>
Greetings, Master Eldrin. I seek your guidance on a matter of great importance. A dark shadow looms over the kingdom...
</assistant>
""", return_tensors='pt')
with torch.cuda.amp.autocast():
    output_tokens = model.generate(**batch, max_new_tokens=400)
print('\n\n', tokenizer.decode(output_tokens[0], skip_special_tokens=True))

```

```

You are a helpful AI assistant for simulating non-playable characters in a fantasy RPG. Simulate response for following character: {'name': 'Eldrin', 'age': 356, 'description': 'A wise and powerful wizard, known for his vast knowledge of arcane }
Greetings, Master Eldrin. I seek your guidance on a matter of great importance. A dark shadow looms over the kingdom...
Ah, a quest of great significance indeed. I shall prepare a potion of focus and clarity to aid in our discussions. response: Eldrin decides to prepare a potion of focus and clarity to aid in discussions. action: {'type': 'drink', 'target': 'po

```

Slika 4.8 Programski kôd za testiranje modela, te rezultati podešavanja phi 3 modela

Ulaz	Izlaz - podešeni model	Izlaz- originalni model (ulaz sadrži više opisa)
<system> character: {'name': 'Eldrin', 'age': 356, 'description': 'A wise and powerful wizard, known for his vast knowledge	Ah, a quest of great significance indeed. I shall prepare a potion of focus and clarity to aid in our discussions. response: Eldrin decides to prepare a potion of focus and clarity to aid in	response: "Ah, the shadows of darkness always seem to find their way into places most unexpected." action: {'type': talkTo, target: Greeter}

<p>of arcane lore and his dry sense of humor.'} memory...</p> <p>< user ></p> <p>Greetings, Master Eldrin. I seek your guidance on a matter of great importance. A dark shadow looms over the kingdom...</p> <p>< assistant ></p>	<p>discussions. action: {'type': 'drink', 'target': 'potion of focus and clarity'}</p>	
<p>< system ></p> <p>character: {'name': 'Aella', 'age': 25, 'description': 'A skilled ranger with a sharp eye and a quick wit.'} memory:...</p> <p>< user ></p> <p>Hail, ranger. I require your assistance in tracking down a dangerous beast that has been terrorizing the nearby village...</p> <p>< assistant ></p>	<p>Greetings, traveler. I am Aella, a skilled ranger. I will aid you in this quest. action: {'type': 'goTo', 'target': 'Dangerous Beast Territory'}</p>	<p>response: "Tracking beasts is my specialty, though I must admit, this one sounds particularly formidable."</p> <p>action: {type: talkTo, target: Villager}</p>
<p>< system ></p> <p>character: {'name': 'Baruk', 'age': 45, 'description': 'A grizzled warrior with many battle scars.'} memory: ...</p> <p>< user ></p> <p>Warrior, the village is under attack again! We need your help!</p> <p>< assistant ></p>	<p>I will lead the defense. {'type': 'EquipItem', 'target': 'Sword'}</p>	<p>response: "Village under siege, you say? Let's show these goblins the might of Baruk!"</p> <p>action: {type: talkTo, target: Villager}</p>
<p>< system ></p> <p>character: {'name': 'Lyra', 'age': 22, 'description': 'A cunning rogue with a talent for stealth.'} memory: ...</p> <p>< user ></p> <p>We've been caught! What do we do now?</p> <p>< assistant ></p>	<p>I'll use my skills to escape and find a way out of this. action: {'type': 'useItem', 'target': 'Daggers'}</p>	<p>response: "Time to disappear before they notice."</p> <p>action: {type: goTo, target: Hidden Spot}</p>

Tablica 4.1 *Usporedba izlaza podešenog i originalnog modela*

Podešavanje modela omogućava prilagodbu modela tako da odgovara specifičnim zahtjevima narativa, svijeta igre i karakterizacije likova. Model daje izlaze koji su duboko integrirani u svijet igre i konzistentni s karakterom. Iz tablice 4.1 je također vidljivo da je moguće

podesiti model da daje odgovor u željenom formatu bez dodatnih uputa modelu što će značajno smanjiti veličinu svakog ulaza.

S druge strane, korištenje originalnog modela s detaljnim instrukcijama u svakom ulazu je korisna metoda kada se model koristi u širem spektru konteksta ili kada je potrebno brzo testiranje i iteracija odgovora. Međutim, bez podešavanja, odgovori mogu biti općenitiji i manje specifični, kao što je vidljivo u primjeru s originalnim modelom gdje Eldrin daje generički odgovor o "sjenama tame". Ovakav odgovor može manje angažirati igrača jer ne nudi specifične smjernice ili radnje, što je često važno u igrama gdje su odlučni odgovori i smjernice ključni za napredak priče i misija.

Stoga, u razvoju igara s dobro definiranom pričom i svijetom, podešavanje modela predstavlja prikladniji pristup, pogotovo u slučajevima gdje precizno kontrolirani odgovori neigrivih likova su ključni igraču za napredovanje i razumijevanje svijeta igre. No zbog povećane fleksibilnosti korištenje inženjerstva ulaza je prikladnije u početku razvoja video igre kada nisu točno definirane moguće akcije i svijet u kojemu je smještena igra. Zbog tog razloga u radu za testiranje će se koristiti pristup inženjerstva ulaza.

4.3 Implementacija osnovnih akcija neigrivih likova

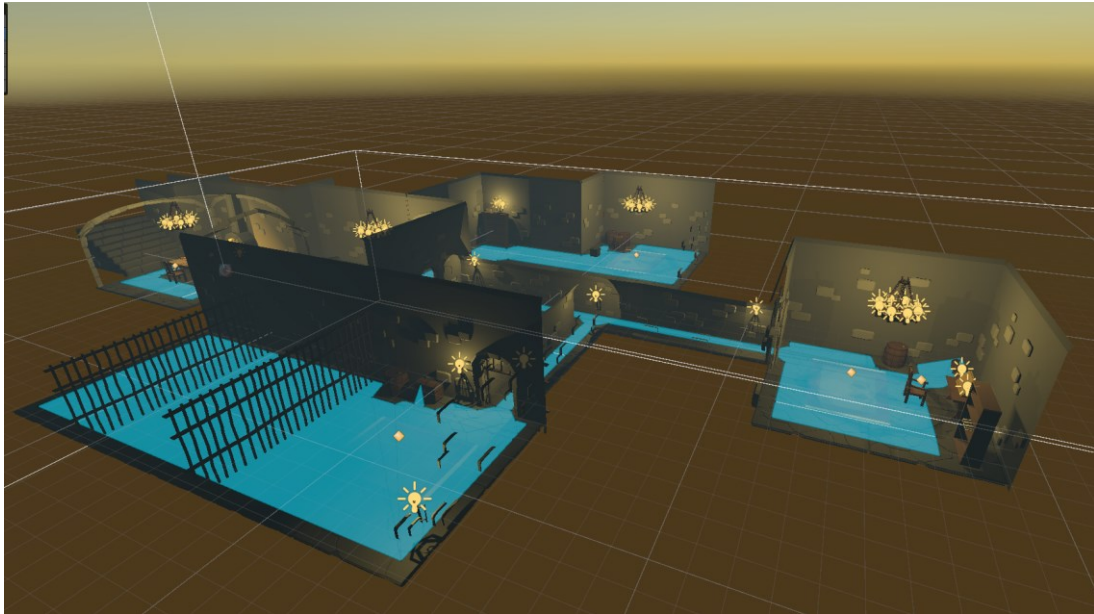
Prema konceptu rješenja veliki jezični model će generirati odgovor i akcije koje neigrivi lik treba poduzeti. Kako bi se akcije koje je model odlučio napraviti mogle izvesti potrebno je napraviti Unity skripte koje će omogućavati te akcije te okruženje u kojemu će se izvoditi akcije.

4.3.1 Kreiranje okruženja

Okolina, vidljiva na slici 4.9, kreirana je koristeći elemente paketa [52]. Okolina se sastoji od nekoliko ključnih lokacija, svaka sa svojom jedinstvenom funkcijom i estetikom. Središnja lokacija je Main Hall, koja služi kao srce ovog virtualnog prostora. Dvorana predstavlja mjesto okupljanja ili početnu točku za istraživanje drugih dijelova tamnice.

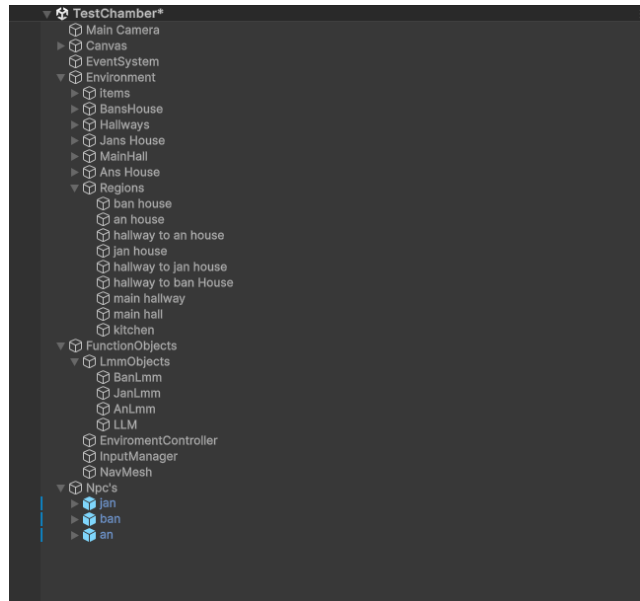
Prostorija direktno povezana s dvoranom je Kitchen. Ova prostorija je zamišljena kao mjesto za druženje i konzumaciju hrane i pića, te se sastoji od stola, hrane i ostalih elemenata

neophodnih za uživanje u obroku. Hodnici (Hallways) služe kao veze između raznih prostorija u svijetu. Svaki hodnik vodi do kuća pojedinih neigrivih likova. Kuće tih likova sadrže osnovne elemente sobe poput kreveta, stolice i spremišta.



Slika 4.9 *Prikaz izgleda okoline*

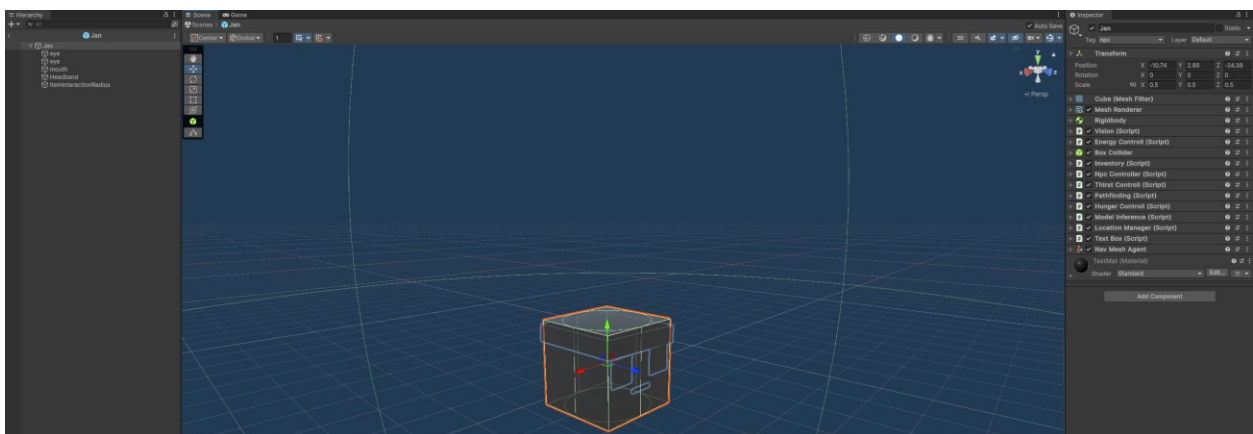
Objekti su hijerarhijski podijeljeni na funkcionalne skupine (slika 4.10). Skupina Enviroment sadrži objekte okruženja s kojima neigrivi likovi mogu imati interakcije. Canvas sadrži objekte korisničkog sučelja, a FunctionObjects sadrži nevidljive objekte koji imaju komponente s raznim funkcionalnostima.



Slika 4.10 Hijerarhija svih elemenata okruženja

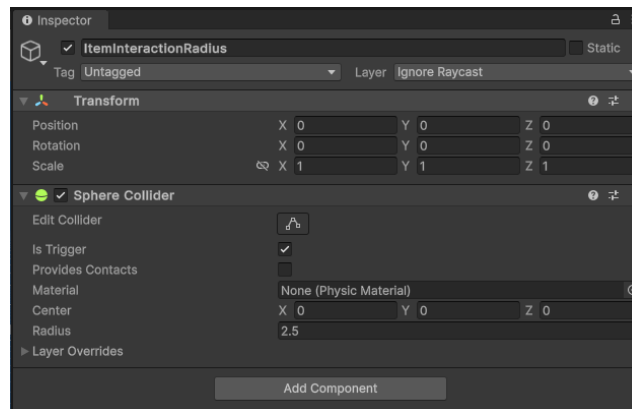
4.3.2 Komponente objekta neigrivog lika

Zadnja skupna NPC's sadržava objekte koji predstavljaju neigrive likove. Ti objekti su temeljeni na prefabu vidljivog na slici 4.11. Prefab se sastoji od više pod objekata koji predstavljaju dijelove tijela neigrivog lika i nemaju nikakvu funkcionalnost osim vizualne.



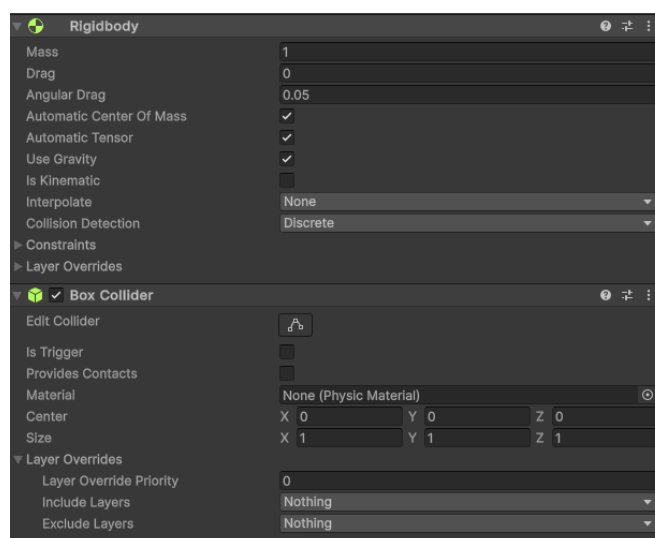
Slika 4.11 Prefab neigrivog lika

Jedini pod objekt koji odstupa od toga pravila je `ItemInteractionRadius` čija je svrha definirati domet interakcije neigrivog lika. Domet je defiliran promjerom (engl. *radius*) Sphere Collider komponente objekta (4.12) koja se koristi za detektiranje sudara s drugim objektima koji imaju bilo koji tip Collider komponente.



Slika 4.12 *ItemInteractionRadius* komponenta

Sve druge funkcionalnosti neigrivog lika nalaze se na korijenskom objektu NPC. Neke od osnovnih komponenata su `Rigidbody` koja omogućuje simulaciju fizike za objekt neigrivog lika i `BoxCollider` komponenta koja omogućuje sudaranje lika s ostalim objektima što sprječava lika da prolazi kroz ostale objekte (Slika 4.13).



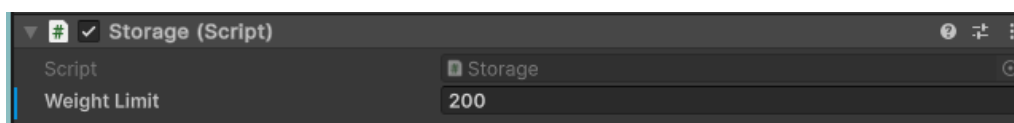
Slika 4.13 *Box Collider i Rigidbody* komponente neigrivog lika

Ostale komponente NPC objekta vidljive na slici 4.11 su vezane za mehanike koje će biti objašnjenje kasnije.

4.3.3 Implementacija mehanike spremnika

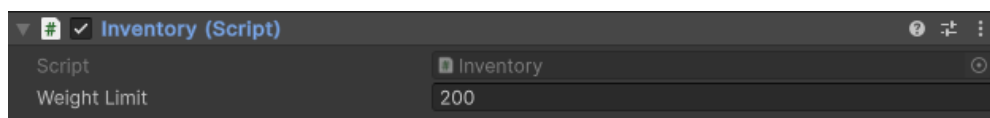
Mehanika spremnika treba omogućiti neigrivim likovima spremanje objekata različitih tipova u osobni spremnik ali i u razne objekte koji mogu predstaviti spremnike poput škrinje.

Spremnici su implementirani skriptom Storage koja upravlja listom objekata tako da sadrži metode za dodavanje objekata u listu, uklanjanje objekata iz liste i dohvaćanje svih objekata liste. Skripta također sadrži logiku za upravljanje kapacitetom koji se računa sumom težina objekata tipa Item u spremniku. Dodavanjem skripte kao komponente na bilo koji objekt pretvara taj objekt u spremnik s podesivim maksimalnim kapacitetom spremnika (slika 4.14).



Slika 4.14 Komponenta spremnika

Za efektivni rad neigrivih s raznim objektima kreirana je skripta Inventory i dodana kao komponenta na svaki objekt neigrivog lika (Slika 4.15). Inventory upravlja objektima klase Item koja je dodana kao komponenta svim objektima s kojima će likovi moći imati interakcije.



Slika 4.15 Komponenta spremnika

Klasu Item (slika 4.16) nasljeđuju sve skripte koje daju funkcionalnosti objektima omogućavajući tako dijeljenje logike među tim objektima, što će u ovom slučaju biti logika spremanja. Osim standardnih metoda za bacanje, dodavanje, opremanje objekata, Inventory komponenta također sadrži specijalizirane metode za interakciju s tipovima objekata. To uključuje metode za konzumiranje hrane, pića, punjenje spremnika tekućina i sve ostale metode vezane za posebne tipove objekata.

```

public class Item : MonoBehaviour
{
    [SerializeField]
    protected float weight;  # Changed in 0+ assets

    # 5 usages  # aomazic
    public float Weight
    {
        # Frequently called
        get => weight;
        set => weight = value;
    }

    # 3 usages  # aomazic
    public Item(float weight)
    {
        this.weight = weight;
    }
}

```

Slika 4.16 *Klasa Item*

4.3.4 Implementacija mehanike kretanja

Sustav kretanja neigrivih likova ostvaren je korištenjem Unity NavMesh paketa [53] koji nudi jednostavnu implementaciju računanja najbližih puteva i kretanja. Skripta PathFinding je kreirana i dodana kao komponenta na objekt neigrivog lika. Bitna metoda u skripti je *GoTo* koja nalazi objekt s jednakim imenom kao što je argument metode i koristeći NavMesh paket postaviti destinaciju lika na točku NavMesh površine koja je blizu lokacije destinacijskog objekta. Osim komponente PathFinding za uspješni rad sustava kretanja potrebno je dodati komponentu Nav Mesh Agent na neigrivog lika te komponentu NavMeshSurface na bilo koji objekt. Navedene metode su vidljive na slici 4.17.

```

public class Pathfinding : MonoBehaviour
{
    private NavMeshAgent navMeshAgent;

    Event function 1 usage aomazic
    private void Start()
    {
        navMeshAgent = GetComponent<NavMeshAgent>();
    }

    Frequently called 1 usage aomazic
    public void GoTo(string destinationName)
    {
        GameObject destination = GameObject.Find(destinationName);
        if (destination)
        {
            Vector3 offset = transform.forward * -2f;
            Vector3 destinationPosition = destination.transform.position + offset;

            // Try to find a point on the NavMesh within a certain radius
            Vector3 randomPoint = destinationPosition + UnityEngine.Random.insideUnitSphere * 10f;
            NavMeshHit hit;
            if (NavMesh.SamplePosition(randomPoint, out hit, maxDistance: 10f, areaMask: NavMesh.ALLAreas))
            {
                // If a point is found, move to that point
                Move(hit.position);
            }
        }
    }

    Frequently called 1 usage aomazic
    private void Move(Vector3 destination)
    {
        navMeshAgent.SetDestination(destination);
        transform.LookAt(destination);
    }

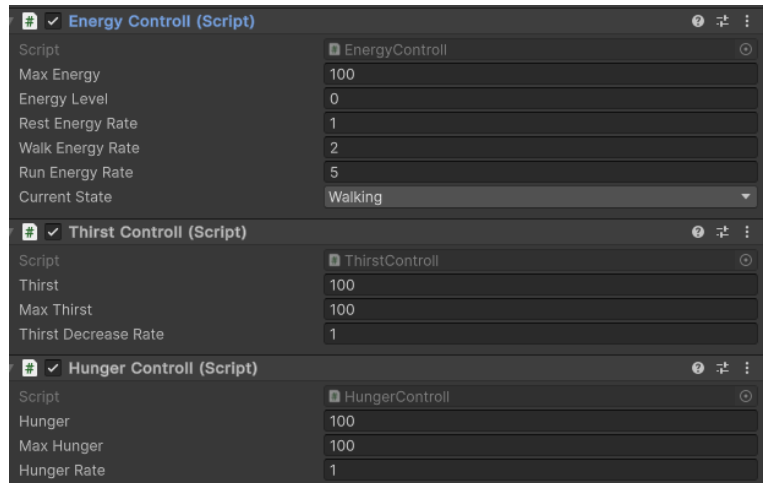
    aomazic
    public void Stop()
    {
        navMeshAgent.isStopped = true;
    }
}

```

Slika 4.17 Metoda GoTo i Move skripte PathFinding

4.3.4 Implementacija statistika neigrivih likova

Za dodatno povećanje kvalitete simulacije kreirani su sustavi upravljanja statistika neigrivih likova. Upravljanje statistikama uključuje kontroliranje umora, žeđi i gladi. Funkcionalnost je omogućena dodavanjem kreiranih komponenata, vidljivih na slici 4.18, objektu neigrivog lika.



Slika 4.18 Komponente za kontrolu statistika neigrivog lika

Dio logike koji se dijeli među komponentama je kontinuirani rast broja koji predstavlja te statistike simulirajući povećanje žeđi, gladi i umora kroz vrijeme. Rast toga broja se obavlja metodom vidljivoj na slici 4.19 gdje `thirstIncreaseRate` predstavlja varijablu koja se zadaje kroz inspektor prozor na komponenti i koja služi za kontroliranje razine rasta te statistike. Razlog množenja s vremenom je da se ova kalkulacija vrši konstantnim periodom neovisno o frekvenciji stvaranja slike videoigre.

```

Frequently called 1 usage aomazic * More...
private void IncreaseThirst()
{
    thirst += thirstIncreaseRate * Time.deltaTime;
    if (thirst > maxThirst)
    {
        thirst = maxThirst;
    }
}

```

Slika 4.19 Metoda za povećanje vrijednosti statistike

Neigrivi likovi mogu smanjiti glad konzumiranjem objekta tipa hrana a žeđ konzumiranjem tekućine, dok se energija obnavlja odmaranjem. Odmaranje se može ostvariti pozivanjem metode `EnterRest` (slika 4.20) koja radi na način da traži objekt prikladnog tipa u blizini i ako postoji takav objekt provjerava ima li slobodnih mjesta na tom objektu. U slučaju da nema slobodnih mjesta ili ako objekt nije u blizini kreira se prikladni zapis u bazi podataka.

```

public void EnterRest(string itemName)
{
    var item = npcController.itemEnvironmentControll.GetItemByNameInRange(itemName, npcController.itemInteractCollider) as RestingItem;
    if (!item)
    {
        npcController.memoryDb.genericObsevation(transform.name, observationText: itemName + " not found", importance: 2f);
        return;
    }

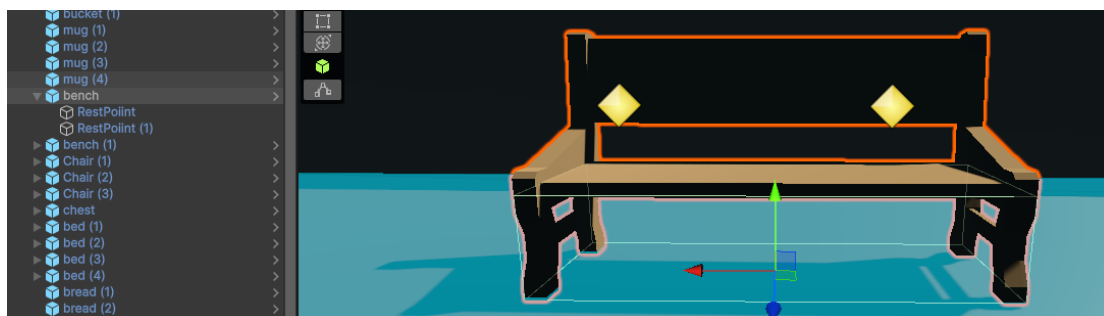
    var closestSlot = item.restingSlots // List<RestingSlot>
        .Where(slot => !slot.isOccupied) // IEnumerable<RestingSlot>
        .OrderBy(slot => Vector3.Distance(a: transform.position, b: slot.position.position)) // IObservable<RestingSlot>
        .FirstOrDefault();

    if (closestSlot)
    {
        closestSlot.isOccupied = true;
        restPoint = closestSlot.position;
        transform.rotation = closestSlot.position.rotation;
        ChangeState(State.Resting);
        npcController.memoryDb.genericObsevation(transform.name, observationText: "Entered rest at " + Time.time + " on " + restPoint.name, importance: 2f);
    }
    else
    {
        npcController.memoryDb.genericObsevation(transform.name, observationText: itemName + " is full", importance: 2f);
    }
}

```

Slika 4.20 Metoda EnterRest

Primjer objekta za odmaranje je vidljiv je na slici 4.21. Bitni pod objekti objekta za odmaranje su RestPoint objekti koji definiraju lokaciju gdje se lik može odmarati. RestPoint objekti također imaju komponentu koja sadrži informaciju o okupiranosti tog RestPointa.



Slika 4.21 Primjer objekta za odmaranje

4.3.5 Implementacija vida neigrivog lika

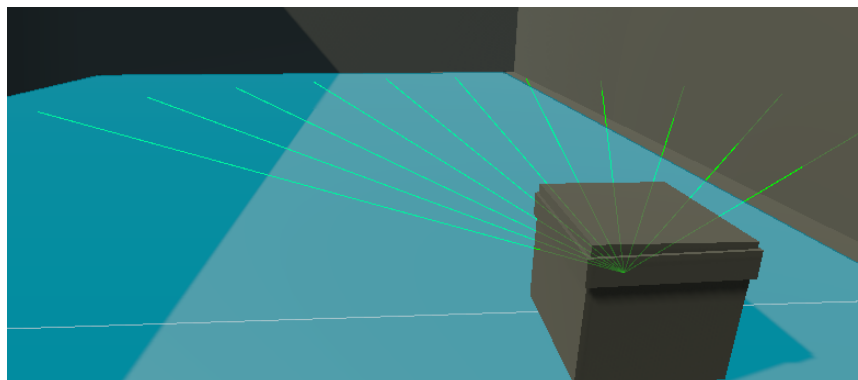
Bitan element simulacije ponašanja likova je njihova mogućnost da promatraju okolinu vidom. Simulacija vida likova ostvarena je kreiranjem skripte Vision (Slika 4.22) koja je dodana

kao komponenta na objekt neigrivog lika. Skripta radi na način da konstanto generira određen broj zraka tipa RayCast koje vraćaju informaciju o objektu koji sudare. Zrake su pravilno poredane u oblik stošca simulirajući tako vidno polje lika. Komponenta omogućava podešavanje duljine zraka, broj zraka, kut stošca i frekvenciju spremanja podataka vida u bazu.

```
Event function aomazic
void Update()
{
    var stepAngleSize:float = fieldOfViewAngle / numberOfRays;
    for (int i = 0; i <= numberOfRays; i++)
    {
        var angle:float = transform.eulerAngles.y - fieldOfViewAngle / 2 + stepAngleSize * i;
        var direction = new Vector3(Mathf.Sin(angle * Mathf.Deg2Rad), 0, Mathf.Cos(angle * Mathf.Deg2Rad));
        direction = transform.TransformDirection(direction) * visionDistance;
        if (Physics.Raycast(origin:transform.position, direction, out RaycastHit hit, visionDistance))
        {
            var objectName:string = hit.transform.name;
            if (!lastSeenTimes.ContainsKey(objectName) || Time.time - lastSeenTimes[objectName] >= saveCooldown)
            {
                Region region = null;
                if (hit.transform.parent)
                {
                    region = hit.transform.parent.GetComponent<Region>();
                }
                npcController.memoryDb.SaveVisionObservation(NPCName:transform.name, objectName, region);
                lastSeenTimes[objectName] = Time.time;
            }
        }
        Debug.DrawRay(start:transform.position, dir:direction, Color.green);
    }
}
```

Slika 4.22 Datoteke potrebne za rad s SQLite bazom podataka

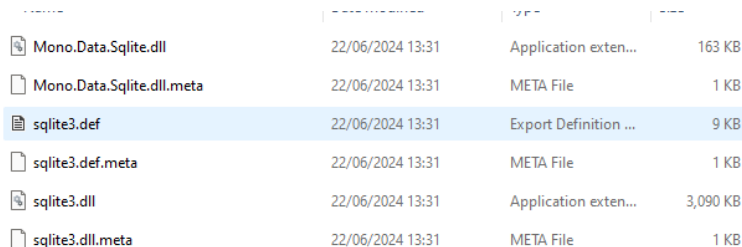
Pozivanjem funkcije *DrawRay* te zrake se prikazu u scenskom pogledu u Unityu (Slika 4.23), što olakšava testiranje i podešavanje parametara komponente.



Slika 4.23 Vizualizacija vida neigrivog lika

4.3 Implementacija baze podataka u Unity

Unity ne podržava nativno Sqlite, ali zbog toga što se Unity pokreće na programskom jeziku *c#* koji podržava tu bazu, implementacija je jednostavna. Za omogućavanje rada s bazom podataka potrebno je dodati datoteke prikazane na slici 4.24 u Plugins direktoriji projekta. Datoteke se mogu preuzeti [49] ili kopirati iz posebnog direktorija koji se nalazi u direktoriju gdje je instaliran Unity program.



Ime datoteke	Datum i vrijeme	Tip datoteke	Veličina
Mono.Data.Sqlite.dll	22/06/2024 13:31	Application exten...	163 KB
Mono.Data.Sqlite.dll.meta	22/06/2024 13:31	META File	1 KB
sqlite3.def	22/06/2024 13:31	Export Definition ...	9 KB
sqlite3.def.meta	22/06/2024 13:31	META File	1 KB
sqlite3.dll	22/06/2024 13:31	Application exten...	3,090 KB
sqlite3.dll.meta	22/06/2024 13:31	META File	1 KB

Slika 4.24 Datoteke potrebne za rad sa Sqlite bazom podataka

Rad s bazom se obavlja metodama paketa *Mono.Data.Sqlite* koji sadrži sve metode potrebne za rad s bazom što uključuje metode za kreiranje baze i za slanje SQL [54] upita. Za upravljanje bazom kreirana je skripta *MemoryDatabase* koja je dodana kao komponenta na objekt *Environment Controll*. Prilikom pokretanja skripta prvo kreira bazu, ili je samo otvori ukoliko postoji (slika 4.25). Nakon inicijalizacije baze skripta kreira tablicu *Observations* ako ne postoji u toj bazi, u koju se spremaju sva opažanja neigrivih likova.

```
private IDbConnection CreateAndOpenDatabase()
{
    var dbUri = "uri=file:" + Application.persistentDataPath + "/Memory.db";
    dbConnection = new SqliteConnection(dbUri);
    dbConnection.Open();

    IDbCommand dbCommandCreateTable = dbConnection.CreateCommand();
    dbCommandCreateTable.CommandText =
        "CREATE TABLE IF NOT EXISTS Observations (ObservationID INTEGER PRIMARY KEY AUTOINCREMENT, NPCName TEXT, Observation TEXT, RecencyScore DECIMAL(10,2), ImportanceScore DECIMAL(10,2), Created TEXT)";
    dbCommandCreateTable.ExecuteReader();

    return dbConnection;
}
```

Slika 4.25 Metoda za kreiranje baze podataka ukoliko ne postoji

Skripta također sadrži sve metode koje pozivaju skripte mehanika u svrhu spremanja promatranja koja dolaze od tih mehanika u bazu. Primjer jednog takvog zapisa je prikazan na slici 4.26 na kojoj se također vidi struktura tablice, i na kojoj su opisani stupci.

WHERE	ORDER BY
ObservationID	NPCName
1	Jan
TestChamber spotted at 01/01/1000 00:00 at location Somewhere	
10.00	
4.00	
01/01/1000 00:00	

Slika 4.26 *Primjer zapisa promatranja neigrivog lika u bazi podataka*

Spremanje u bazu je integrirano s svim prijašnje objašnjenjem mehanikama na način da te skripte pozivaju odgovarajuće metode MemoryDatabase komponente prilikom bitnog događaja. Primjer jedne metode koju poziva skripta za simulaciju vida je prikazana na slici 4.27. Metode za spremanje u bazu su unikatne zbog načina računanja faktora važnosti, teksta promatranja i ostalih posebnosti svake mehanike.

```

public void SaveVisionObservation(String NPCName, String objectName, Region region)
{
    IDbCommand dbCommand = dbConnection.CreateCommand();
    string currentTime = timeManager.GetGameDateTime();
    string regionName = "region ? "Somewhere" : region.name;
    string observationText = objectName + " spotted at " + currentTime + " at location " + regionName;

    dbCommand.CommandText =
        "INSERT INTO Observations (NPCName, Observation, RecencyScore, Created) VALUES (@NPCName, @Observation, @RecencyScore, @Created)";

    IDbDataParameter paramNPCName = dbCommand.CreateParameter();
    paramNPCName.ParameterName = "@NPCName";
    paramNPCName.Value = NPCName;
    dbCommand.Parameters.Add(paramNPCName);

    IDbDataParameter paramObservation = dbCommand.CreateParameter();
    paramObservation.ParameterName = "@Observation";
    paramObservation.Value = observationText;
    dbCommand.Parameters.Add(paramObservation);

    IDbDataParameter paramRecencyScore = dbCommand.CreateParameter();
    paramRecencyScore.ParameterName = "@RecencyScore";
    paramRecencyScore.Value = 10f;
    dbCommand.Parameters.Add(paramRecencyScore);

    IDbDataParameter paramCreated = dbCommand.CreateParameter();
    paramCreated.ParameterName = "@Created";
    paramCreated.Value = currentTime;
    dbCommand.Parameters.Add(paramCreated);

    dbCommand.ExecuteReader();
}

```

Slika 4.27 *Metoda za spremanje opservacija viđenja*

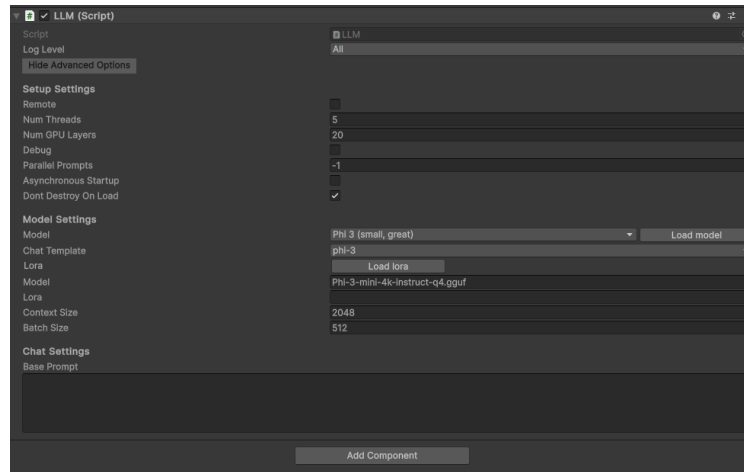
U bazu će se spremati velika količina podataka što može uzrokovati resursne probleme. Slika 4.28 prikazuje način na koji je taj problem riješen. Metoda *UpdateDatabase* periodično briše sve zapise s niskim faktorom novosti i važnosti. Osim brisanja metoda također smanjuje faktor novosti svim zapisima prolaskom vremena.

```
StartCoroutine(routine: UpdateDatabase());  
  
6 Frequently called 1 usage 2 somazic  
private IEnumerator UpdateDatabase()  
{  
    while (true)  
    {  
        yield return new WaitForSeconds(updateInterval);  
  
        IDbCommand dbCommandUpdateRecency = dbConnection.CreateCommand();  
        dbCommandUpdateRecency.CommandText =  
            "UPDATE Observations SET RecencyScore = RecencyScore - (@recencyDecayFactor / (ImportanceScore + 1))";  
        IDbDataParameter param = dbCommandUpdateRecency.CreateParameter();  
        param.ParameterName = "recencyDecayFactor";  
        param.Value = recencyDecayFactor;  
        dbCommandUpdateRecency.Parameters.Add(param);  
        dbCommandUpdateRecency.ExecuteReader();  
  
        IDbCommand dbCommandDeleteOld = dbConnection.CreateCommand();  
        dbCommandDeleteOld.CommandText =  
            "DELETE FROM Observations WHERE RecencyScore <= 0";  
        dbCommandDeleteOld.ExecuteReader();  
    }  
}
```

Slika 4.28 Metoda *updateDatabase*

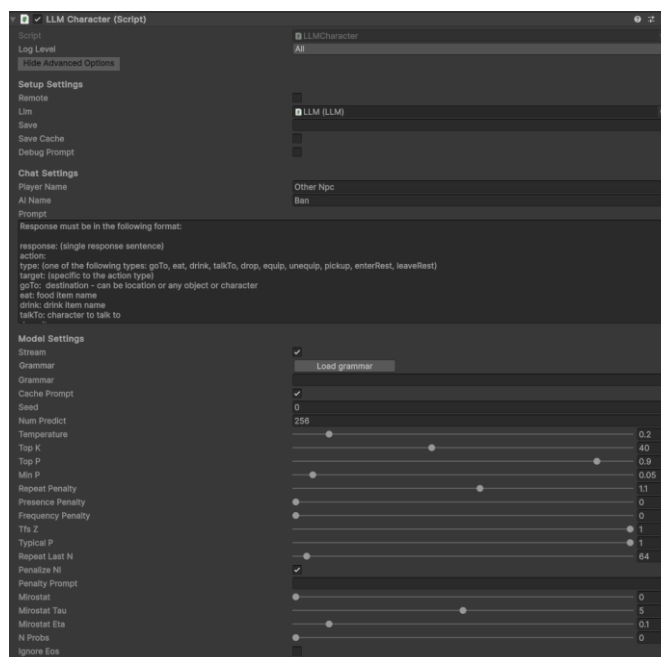
4.4 Integracija velikog jezičnog modela u Unity

Prvi korak integracije Velikog jezičnog modela je omogućiti njegovo pokretanje u Unity okruženju. Taj korak uključuje uvoz paketa u projekt kroz prozor package manager te dodavanje komponente LLM na bilo koji objekt. Komponenta LLM je dodana na funkcijski objekt LMM (Slika 4.29). Inspektor prozor komponente sadrži razne parametre s kojima se može optimizirati rad modela i promijeniti način rada modela. Osim toga biranje modela koji će se pokretati se također obavlja kroz inspektor prozor. Za svrhe testiranja odabran je phi 3 model.



Slika 4.29 Komponenta LLM

Za istovremenu komunikaciju s modelom s različitim instancama potrebno je dodati komponentu LLMCharacter (slika 4.30) na svaki objekt koji će imati svoju instancu [13]. Ova komponenta je dodana svakom neigrivom liku koji će se simulirati, s različitim parametrom Ai name. Bitni odjeljak za kvalitetu simulacije je odjeljak Prompt u kojemu se postavljaju kontekstualni podaci za model. To uključuje instrukcije za generiranje teksta te kontekstualne podatke svijeta.



Slika 4.30 Komponenta LLM Character

Instrukcije za model sadrže opis svih mogućih akcija koje neigrivi lik može poduzeti i lokacije u svijetu. Trenutačne instrukcije:

Response must be in the following format:

response: (single response sentence)

action:

type: (one of the following types: goTo, eat, drink, talkTo, drop, equip, unequip, pickup, enterRest, leaveRest)

target: (specific to the action type)

goTo: destination - can be location or any object or character

eat: food item name

drink: drink item name

talkTo: character to talk to

drop: item name

equip: item name

unequip: no target needed

pickup: item name

enterRest: rest item (example: bed, chair)

leaveRest: no target needed

Example Format:

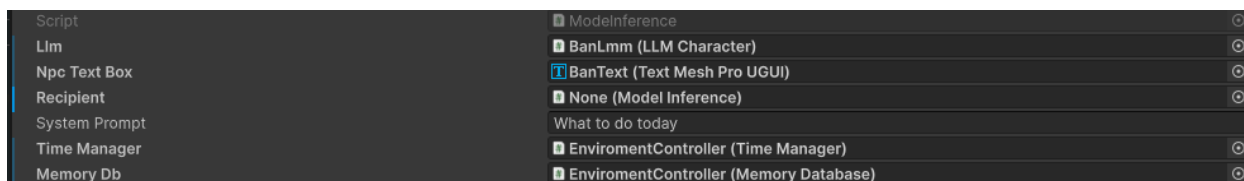
response: "I am heading to the kitchen."

action: {type: goTo, target: Kitchen}

Game World Locations: kitchen, main hall, main hallway, hallway to ban house, hallway to jan House, jan house, ban house, an house, hallway to an house

Write the response as if you are the character. Ensure the response is consistent with the character's personality and current situation.

Sljedeći korak je razvoj komponenti koje će upravljati interakcije s modelom za svakog neigrivog lika. Kreirana je skripta ModelInference koja se dodaje na svakog lika kao komponenta (slika 4.31) i služi za kontroliranje interakcije s modelom i klasa ResponseParser koja sadrži model klase i metode za oblikovanje ulaza i čitanje izlaza.



Slika 4.31 *Komponenta ModelInference*

ModelInference sadrži nekoliko serijaliziranih polja, kao što je npcTextBox za prikaz teksta, llm za objekt klijenta jezičnog modela te razne kontrolere koji upravljaju ponašanjem neigrivog lika i varijablu delay koja kontrolira vrijeme između slanja ulaza u model (Slika 4.32).

```
[SerializeField] private LLMCharacter llm; // Serializable
[SerializeField] TextMeshProUGUI npcTextBox; // Serializable
private string prompt = "";
public ModelInference recipient; // Serializable
private string reply;
private NPCController npcController;
private Inventory inventory;
private EnergyControll energyControll;
private Pathfinding pathfinding;
private LocationManager locationManager;
[SerializeField] private string systemPrompt = "Hmmm what to do next?"; // Serializable
[SerializeField] private TimeManager timeManager; // Serializable
[SerializeField] MemoryDatabase memoryDb; // Serializable
private float delay = 5f;
List<string> importantRecords = new List<string>();
```

Slika 4.32 *Sva polja i varijable ModelInference skripte*

Nakon inicijalizacije, skripta postavlja okruženje dodjeljivanjem referenci potrebnim komponentama kao što su NPCController, Inventory, EnergyControll, Pathfinding i LocationManager. Ove komponente daju neigrivom liku mogućnost izvođenja niza radnji, od kretanja po svijetu igre do interakcije s objektima i drugim likovima. Metoda Start poziva metodu Inference da započne proces donošenja odluka. Postavljanjem variable systemPrompt u Start metodi se također može sastaviti ulaz koji sadrži informacije svijeta s ciljem model dobije kontekst u kojemu se nalazi lik kojeg simulira. Kreirani svijet je jednostavan pa ovaj korak nije potreban.

Metoda *Inference* (Slika 4.33) ključna je funkcija koja konstruira upit za jezični model koristeći trenutno stanje neigrivog lika i relevantne memorijske zapise dohvaćene iz baze podataka. Memorijski zapisi se dohvaćaju ovisno o kalkulaciji faktora novosti i važnosti. Nakon

konstruiranja taj se upit pošalje modelu i zadaje se metoda *HandleReply* kao metoda za obrađivanje izlaza i metoda *ReplayComplete* kao metoda koja se izvrši nakon kraja odgovora modela. *HandleReply* obrađuje odgovor jezičnog modela, ažurira tekst sučelja i pohranjuje odgovor za daljnje radnje.

```
private void Inference()
{
    importantRecords = memoryDb.FetchImportantRecords(recordNum:2, npcController.name);
    prompt = ResponseParser.ConstructPrompt(new Prompt(
        locationManager.currentRegion, timeManager.GetGameDateTime(),
        npcController.description, systemPrompt, memory:importantRecords[0],
        memory2:importantRecords[1]));
    npcTextBox.text = "";
    _ = llm.Chat(prompt, HandleReply, ReplyCompleted);
}
```

Slika 4.33 *Metoda Inference*

ReplayComplete poziva metodu *PerformAction* (Slika 4.34) koja pretvara akciju iz odgovora u konkretnu akciju lika pozivajući odgovarajuće funkcije iz prikladnog kontrolera. Tip akcija se dobije obrađivanjem izlaza korištenjem *ResponseParser* funkcija.

```

10 private void PerformAction(ResponseAction action)
11 {
12     string target = action.Target.ToLower();
13     switch (action.Type)
14     {
15         case "equip":
16             inventory.EquipItem(target.ToLower());
17             Debug.Log(message:"Equip");
18             break;
19         case "unequip":
20             inventory.UnequipItem();
21             Debug.Log(message:"unequip");
22             break;
23         case "drink":
24             Debug.Log(message:"drink");
25             npcController.DrinkItem(target.ToLower());
26             break;
27         case "drop":
28             Debug.Log(message:"drop");
29             inventory.DropItem(target.ToLower());
30             break;
31         case "pickup":
32             Debug.Log(message:"pickup");
33             inventory.PickupItem(target.ToLower());
34             break;
35         case "eat":
36             Debug.Log(message:"eat");
37             inventory.EatFood(itemName:target.ToLower());
38             break;
39         case "enterrest":
40             Debug.Log(message:"rest");
41             energyControll.EnterRest(itemName:target.ToLower());
42             break;
43         case "leaveRest":
44             Debug.Log(message:"leaveRest");
45             energyControll.LeaveRest();
46             break;
47         case "talkto":
48             Debug.Log(message:"talkto");
49             SetRecipient(target.ToLower());
50             break;
51         case "goto":
52             Debug.Log(message:"goto");
53             pathfinding.GoTo(destinationName:target.ToLower());
54             break;
55         default:
56             recipient = null;
57             Debug.LogError(message:"Unkown action type: " + action.Type);
58             break;
59     }
60 }

```

Slika 4.34 *Metoda PerformAction*

Posebni tip akcije je TalkTo zato što tip ne poziva neku metodu već postavlja vrijednost Recipient objekta ukoliko se objekt nalazi u blizini objekta koji poziva. Ako vrijednost recipient varijable nije prazna onda će poruka izlaza služiti kao ulaz u model recipient objekta. Ako objekt ne postoji ili je predaleko dodaje se prikladni zapis u memoriju.

```

private void SetRecipient(string recipientName)
{
    var recipientObject = GameObject.Find(recipientName).GetComponent<ModelInference>();
    if (!recipientObject)
    {
        memoryDb.genericObsevation(npcController.name, observationText:$"Tried to talk to {recipientName} but they do not exist", importance:5f);
        return;
    }

    var recipientCollider = recipientObject.GetComponent<NpcController>().itemInteractCollider;
    if (!npcController.itemInteractCollider.bounds.Intersects(recipientCollider.bounds))
    {
        memoryDb.genericObsevation(npcController.name, observationText:$"Tried to talk to {recipientName} but they are too far away", importance:5f);
        return;
    }
}

```

Slika 4.35 *Metoda SetRecipient*

Slika 4.36 prikazuje pomoćne funkcije klase `ResponseParser` koje se koriste u cijelom procesu interakcije s modelom. `ParseResponse` tekst izlaza pretvara u objekt `Response` koji ima više objekta modela kao svojstva. Dok funkcija `ConstructPrompt` kreira tekst ulaza iz predanih podataka koji su u obliku objekta `Prompt`.

```
40 public static class ResponseParser
41 {
42     ▲ Frequently called (01 usage) @ somazc:~
43     public static Response ParseResponse(string response)
44     {
45         var parts = response.Split(separator: new[] { "action: " }, StringSplitOptions.None);
46         var responsePart = parts[0].Replace(oldValue: "response: ", newValue: "").Trim().Replace(oldValue: "\n", newValue: "");
47         var actionPart = parts.Length > 1 ? parts[1] : "";
48
49         var actionTypeMatch = Regex.Match(input: actionPart, pattern: @"type: (.+)", "");
50         var targetMatch = Regex.Match(input: actionPart, pattern: @"target: (.+)", "");
51
52         var cleanedActionType = actionTypeMatch.Groups[1].Value.Trim();
53         var cleanedTarget = targetMatch.Groups[1].Value.Trim();
54
55         return new Response
56         {
57             ResponseText = responsePart.ToLower(),
58             ResponseAction = new ResponseAction
59             {
60                 Type = cleanedActionType.ToLower(),
61                 Target = cleanedTarget.ToLower()
62             }
63         };
64     }
65     ▲ Frequently called (01 usage) @ somazc:~
66     public static string ConstructPrompt(Prompt prompt)
67     {
68         return "Location : " + prompt.CurrentLocation + ", dateAndTime : " + prompt.DateAndTime +
69             "\n Who am I : " + prompt.NpcDescription + "\n" +
70             prompt.SystemPrompt + "\n memory1 : " + prompt.Memory1 +
71             "\n memory2 : " + prompt.Memory2 + "\n memory3 : " + prompt.Memory3;
72     }
73 }
```

Slika 4.36 Klasa `ResponseParser`

4.5 Vrednovanje simulacije

U simulaciji su prisutna tri lika s specifičnim osobinama. Jan je ljubazan i uživa u razgovoru s An i Ban. Ban je glasan i dosadan, a posebno voli razgovarati s Janom, što ponekad može biti naporno za druge. An preferira spavanje, ali se raduje razgovoru s Janom kad je budna. Pokretanjem igre se pokreće i simulacija. Radi lakšeg vrednovanja izlaza iznad svakog lika se nalazi tekst koji predstavlja cijeli izlaz modela.

Na slici 4.37 su vidljivi prvi izlazi koje model generira nakon pokretanja igre. Jan se odlučio na razgovor s An koja se isto odlučila za razgovor s Janom postavljanjem pitanjem koji je u skladu s njezinim opisom međutim kako je instanca modela Jana brže generirala odgovor Anino pitanje se ignorira te ona će odgovoriti na pitanje Jana u idućem izlazu. Ban se odlučio na razgovor

s Janom referencirajući objektom koje je učio vidom. Kako Jan nije u blizini Jana u Banovo pamćenje će se spremiti zapis koji opisuje da Jan nije u blizini.



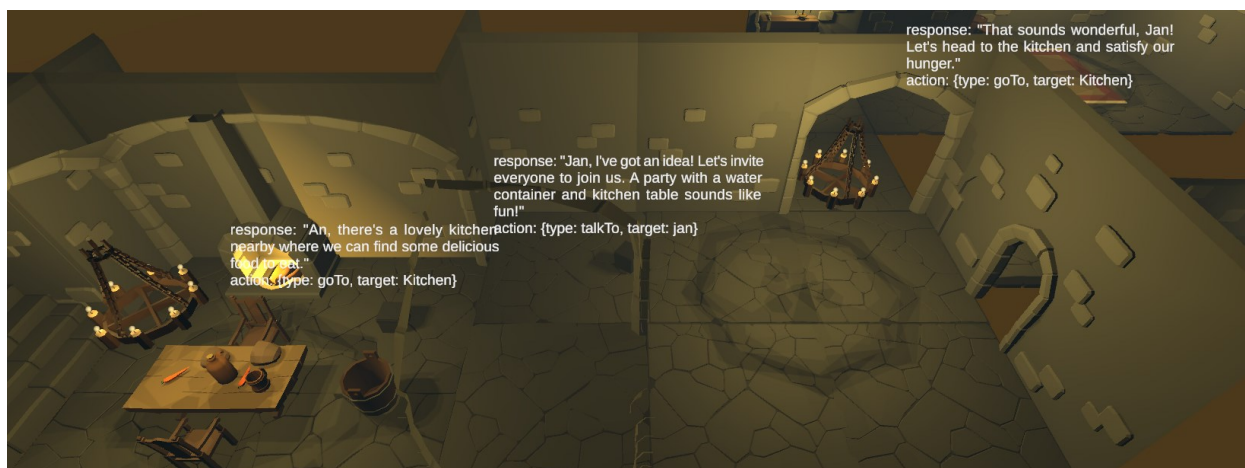
Slika 4.37 Prvi izlazi modela nakon pokretanja simulacije

Drugi izlaz (Slika 4.38) sadrži Anin odgovor Janu i Janov odgovor na Anino pitanje. Ana je rekla da je gladna pa se Jan odlučio na akciju kretanja do kuhinje. Ban i dalje pokušava razgovarati s Janom iako nije u blizini, razlog toga je što se zapis u memoriji o tome da Jan nije u blizini nije dohvatio u kreiranju upita. Potencijalna rješenja problema su opisana u idućem poglavlju.



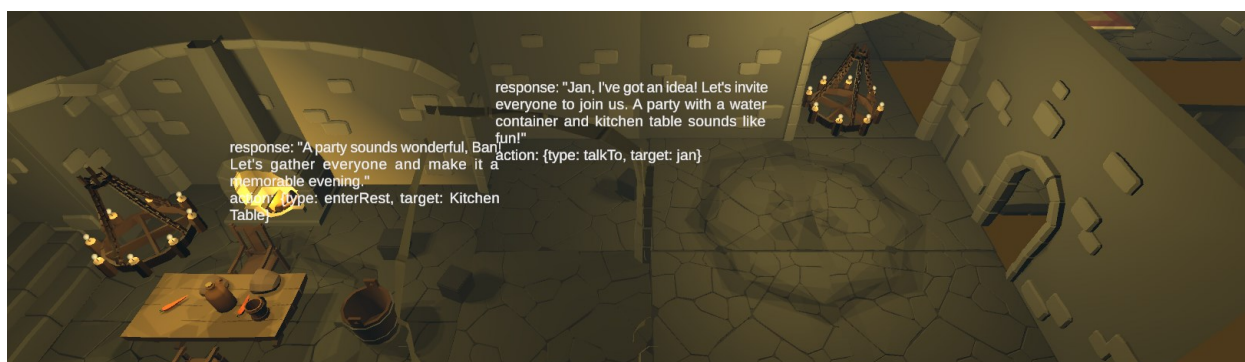
Slika 4.38 Drugi izlazi modela nakon pokretanja simulacije

Na slici 4.39 je vidljiv nastavak simulacije. Jan je otišao u kuhinju dok se Ana također odlučila za posjet kuhinje zbog prijedloga Jana. Ban dalje pokušava razgovarati s Janom, ali sada kako je Jan u blizini on će uspješno odgovoriti na pitanje (slika 4.40).



Slika 4.39 Treći izlazi modela nakon pokretanja simulacije

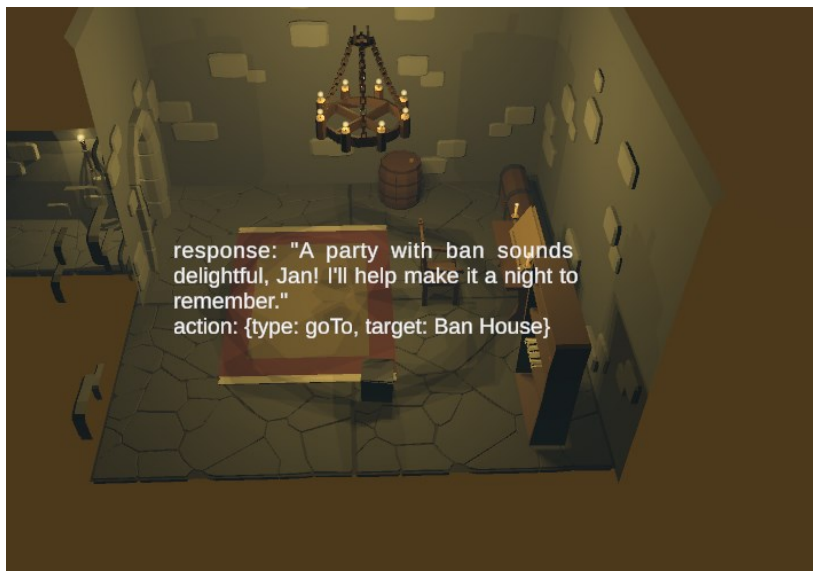
Janu se svidjela ideja pa je odlučio nastaviti s realizacijom ideje Bana odlukom da će pozvati sve na događaj.



Slika 4.40 Odgovor Jana na pitanje Bana vezano za organiziranje događaja

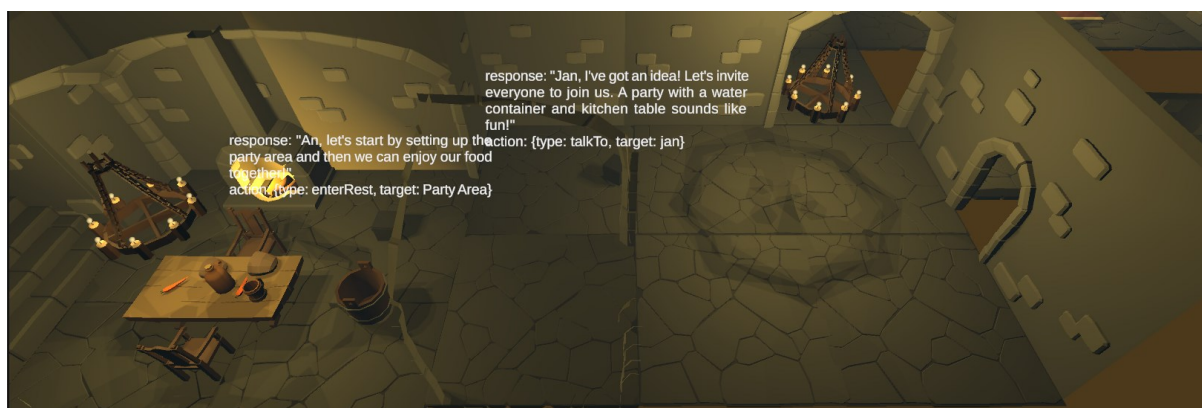
Ban se odlučio vratiti kući (slika 4.41) s mogućim ciljem da pokupi dekoracije ili je model greškom pomislio da je ban neki drugi lik a ne sami model, što se može vidjeti u odgovoru i tako pomislio

da se događaj odvija u banovoj kući. U nastavku će lik Bana pokušati pokupiti razne objekte vezan uz događaj koji ne postoje i zapeti u krugu.



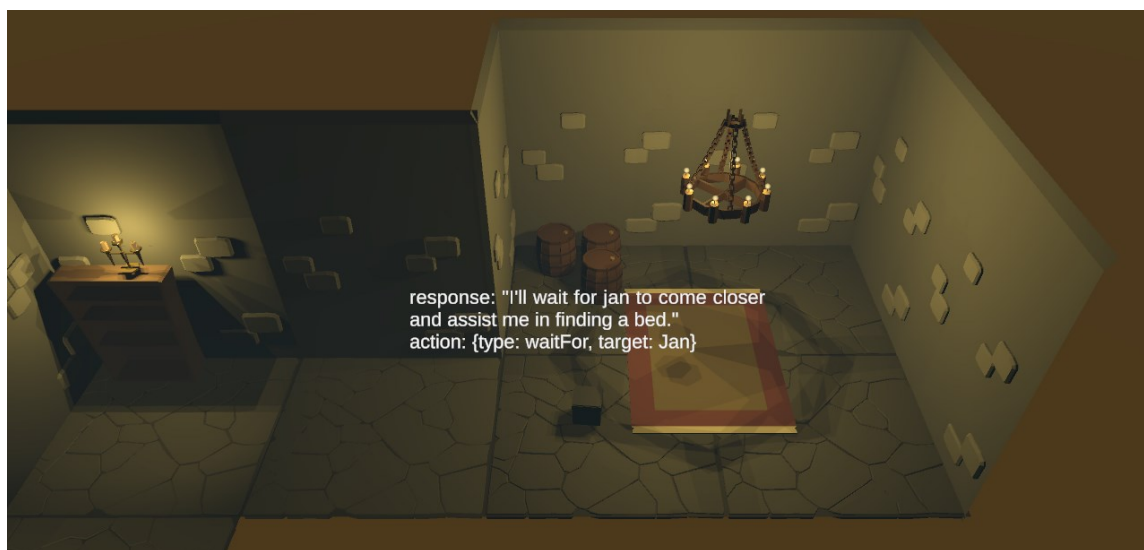
Slika 4.41 *Odgovor Jana na pitanje Bana*

Slika 4.41 prikazuje kako je Jan uspješno pozvao An na događaj ali nakon toga pokušava otići na lokaciju koja ne postoji.



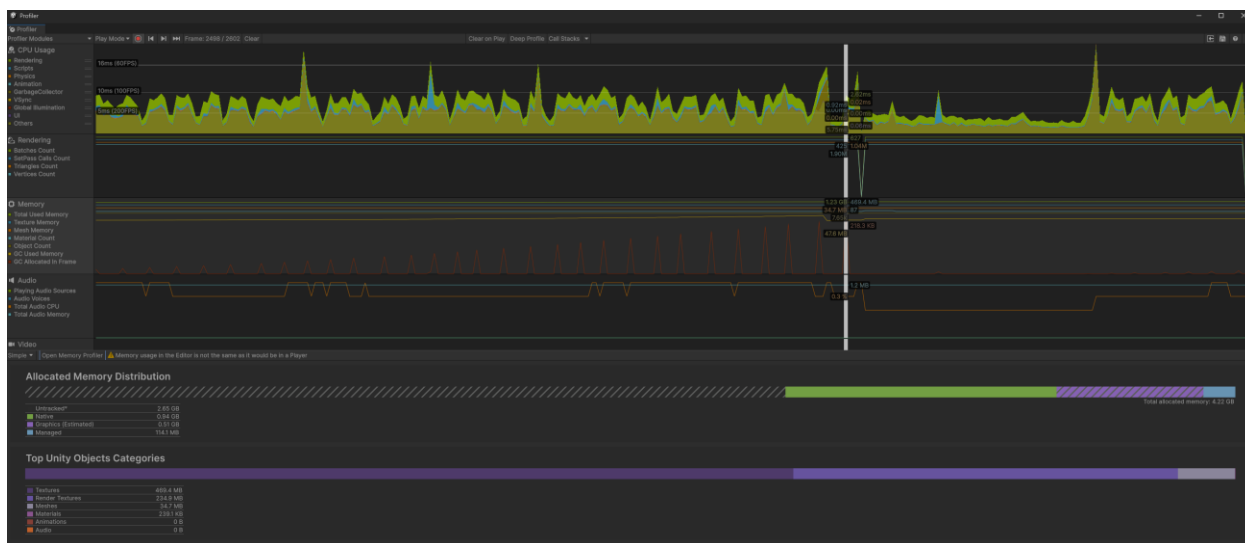
Slika 4.41 *Razgovor Jana i Ane vezan uz događaj*

Pokretanjem simulacije više puta dobiju se slični rezultati. Likovi pokazuju sposobnost za organiziranje događaja, pričanje i stvaranje međusobnih veza, ali uz često generiranje akcija koje je nemoguće obaviti i ulaženje u krug konstantnog pozivanja akcije koju taj lik ne može obaviti. Također uočeno je da nakon određenog pozivanja iste akcije koja vraća grešku model počinje pozivati akcije koje nisu definirane u instrukcijama. Slika 4.42 prikazuje takav primjer gdje lik nije mogao naći jedan objekt pa je tražio pomoć od lika koji nije u blizini što je rezultiralo pozivanjem akcija koje ne postoje.



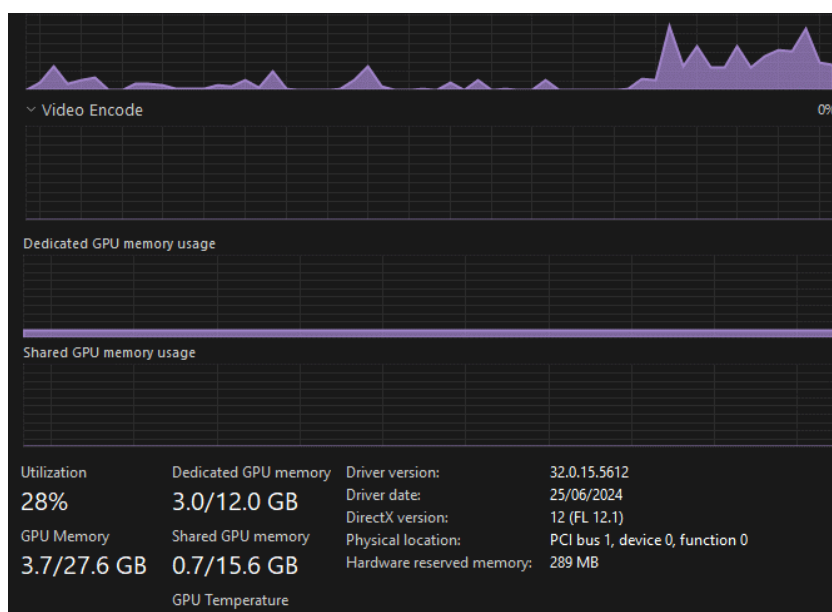
Slika 4.42 Model poziva akciju koja ne postoji

Upotrebom Unity profiler alat prikazano je korištenje računalnih resursa tokom rada simulacije. Analiza na slici 4.43 prikazuje efikasnost lama.cpp biblioteke i phi 3 modela. Vidljivo je da prilikom generiranja odgovora povećanje korištenja memorije je zanemarivo.



Slika 4.43 Analiza korištenja računalnih resursa

Osim korištenja memorije računala llama.cpp koristi resurse grafičkih kartica ukoliko su prisutne na računalu. Skok korištenja memorije grafičke kartice vidljiv je na slici 4.44. Vidi se ograničenje pokretanja modela lokalno na računalu, a to je visoka potrošnja memorije potrebna za pokretanje modela. Ovakva potrošnja ograničava videoigru na računala s modernim grafičkim karticama.



Slika 4.44 Memorija grafičke kartice nakon pokretanja simulacije

Zaključno, model se pokazao posebno dobrim za generiranje dijaloga. Likovi mogu sudjelovati u smislenim razgovorima i međusobno komunicirati na način koji djeluje prirodno i uvjerljivo. Međutim, kontrola njihovih akcija zahtijeva više optimizacije. Trenutni model nije sposoban adekvatno upravljati složenijim zadacima i neočekivanim situacijama, što dovodi do

problema poput pozivanja nepostojećih akcija i zapinjanja u petljama grešaka. Također potvrdila se briga o računalnim resursima korištenja modela. Iako se koristi lagani model potrošnja resursa je dalje visoka s obzirom na grafičku kompleksnost simulacije.

5. POTENCIJALNA POBOLJŠANJA

Ovo poglavlje navodi poboljšanja sustava kako bi bio efikasniji i korisniji u različitim scenarijima.

Za veću razinu realizma simulacije mogu se implementirati više mehanika i tako proširiti fond sposobnosti neigrivih likova. Primjer je uvođenje sluha likova ili sustava razmjene objekata među likova. Osim uvođenjem novih poštujuće mehanike se mogu poboljšati na razne načine. Mehanika vida se može proširiti na način da se zrake šalju pod više vertikalnih kutova omogućavajući likovima da uoče objekte koji su nisko na podu ili visoko na zidu.

Jedan od ključnih izazova trenutnog sustava je potreba da model daje izlaze u unaprijed definiranom formatu kako bi se ti izlazi mogli interpretirati i pretvoriti u akcije. Testiranje je pokazalo da model može dati izlaze u krivom formatu i pozivati akcije koje ne postoje unatoč tome što su akcije i format strogo definirani u instrukcijama. Ovo ograničenje može se riješiti uvođenjem modela sličnosti rečenica [55].

Modeli sličnosti rečenica koriste različite tehnike, uključujući vektorske reprezentacije i duboko učenje, za mjerenje sličnosti između dviju rečenica. U kontekstu trenutnog rješenja, takvi modeli mogu omogućiti interpretaciju izlaza modela bez potrebe za strogim formatiranjem. Umjesto toga, sustav može upotrijebiti model kako bi usporedio izlaz modela s unaprijed definiranom bazom mogućih akcija i odabrao onu koja je najbližnja.

Drugi izazov je određivanje važnosti memorijskih zapisa koristeći unaprijed definirane vrijednosti. Trenutno sustav koristi fiksne vrijednosti za važnost memorijskih zapisa, što može biti neefikasno jer ne uzima u obzir kontekstualne promjene i dinamičku prirodu interakcija. Problem se može riješiti uvođenjem modela za procjenu važnosti memorijskih zapisa. Ovaj model će koristiti metode strojnog učenja kako bi analizirao kontekstualne informacije i odredio važnost memorijskih zapisa u realnom vremenu.

Također realizam modela poboljšala bi direktna povratna veza s mehanikama u slučaju neočekivanog rezultata. Umjesto da se samo spremaju zapisi u bazu u takvom slučaju mehanike mogu vratiti novi posebni ulaz u model koji opisuje pogrešku. Ovaj pristup bi smanjio vjerojatnost lika da zapne u konstantom generiranju sličnih nemogućih akcija.

Dodavanje Unity ML-Agents sustava može dodatno povećati dinamičnost i realizam simulacije. ML-Agents [56] omogućuju integraciju agenata s umjetnom inteligencijom koji uče iz interakcija s okolinom koristeći metodu podržanog učenja. Ovi agenti mogu poboljšati ponašanje neigrivih likova čineći obavljanje akcije na dinamičnije i nepredvidljive načine. Primjer je korištenje agenta za kretanje u prostoru umjesto trenutnog rigidnog rješenja.

Još jedno značajno poboljšanje može biti omogućavanje modelu da generira koliko akcija treba djelovati umjesto unaprijed definiranih intervala između ponovnog generiranja akcija, s mogućnošću da se prekid prekine u nekim slučajevima, poput prekida odmora zbog zvuka ili drugih vanjskih podražaja.

6. ZAKLJUČAK

U ovom diplomskom radu su istražene i primijenjene tehnike i alati iz područja umjetne inteligencije za dinamičko modeliranje ponašanja neigrivih likova videoigrama. Prvo je kreiran skup podataka potreban za podešavanje modela. Skup sadrži zapise u JSON formatu koji predstavljaju željene izlaze na kreirane ulaze modela. Korištenjem Google Colaba skup podataka je pretvoren iz JSON formata u obični tekstualni format s posebnim dodanim oznakama specifične za model. Procesuirani skup podataka se uspješno koristio za podešavanje modela korištenjem LORA tehnike. Model je testiran i uspoređen s originalima modelom i donesena je odluka o korištenju originalnog modela za daljnji razvoj. Potencijalno poboljšanje ovog koraka može biti kreiranje više raznovrsnijih podataka u skupu koji su više usklađeni s tematikom igre.

U Unityu kreirana je testna scena koja sadrži objekte različitih tipova koji omogućiti interakciju s neigrivim likovima. Okruženje je zatim pažljivo hijerarhijski organizirano, s objektima kategoriziranim na temelju njihove funkcije. Takva organizacija olakšala je daljnji razvoj i pravljenje potencijalnih izmjena. Zatim je uslijedila izgradnja prefaba neigrivih likova. Kreirana je temeljna struktura neigrivog lika s pod objektima koje služe vizualne ili funkcionalne uloge. Na sve objekte su dodane osnovne funkcionalnosti poput fizičke simulacije i otkrivanja sudara pomoću komponenti kao što su RigidBody i BoxCollider. Te su komponente osiguravale realistično kretanje i sprječavale prolazak objekta kroz zidove.

Idući korak je bio implementiranje posebnih funkcionalnosti potrebne za ostvarivanje osnovne razine simulacije. Skripte Storage i Inventory razvijene su za upravljanje predmetima koje likovi mogu nositi i s kojima mogu imati interakcije. Ove skripte, osim da su pratile stanje objekata, su i također definirale i omogućile interakcije poput opremanja ili konzumiranja predmeta. Kretanje likova okolinom omogućeno je skriptom PathFinding koja poziva potrebne funkcije NavMesh paketa omogućujući pronalaženje i kretanje do destinacije. Kako bi se dodatno podigao realizam likova, implementiran je sustav koji upravlja različitim statistikama poput gladi, žeđi i umora. Ove mehanike utječu na ponašanje likova na način da ih potiče da potraže hranu ili odmor kada su njihove statistike dosegle kritične razine. Pomoću RayCasta implementiran je sustav vida za neigrive likove poboljšavajući njihove kontekstualne sposobnosti.

U svrhe poboljšavanja problema pamćenja konteksta modela uvedena je baza podataka u projekt koja je povezana s ostalim mehanikama na način da izvršenjem tih mehanika se sprema zapis u bazu o tom izvršenju.

Nakon baze u projekt je dodan paket LLM for Unity i pripadajuće komponente iz paketa koje omogućuju pokretanje modela u videoigri. Nakon podešavanja parametara i učitavanja modela kreirana i dodana je komponenta ModelInference na svakog neigrivog lika. Komponentom je definirana logika komunikacije svakog neigrivog lika s velikim jezičnim modelom. Logika uključuje učitavanje kontekstualnih podataka o liku i svijetu u model prilikom inicijalizacije, periodično slanje upita na model i obrada odgovora modela.

Cijeli sustav je zatim testiran pokretanjem simulacije i praćenjem ponašanja likova. Uočene su prednosti i mane trenutačnog pristupa. Također je analizirano korištenje računalnih resursa tijekom rada simulacije.

LITERATURA

- [1] Jose J. Martinez, AI in Video Games: A Historical Evolution, from Search Trees to LLMs, Medium, 2024, dostupno na: <https://medium.com/@jjmcarrascosa/ai-in-video-games-a-historical-evolution-from-search-trees-to-llms-chapter-1-1949-1980-f3b04d6e9dc8> [pristupljeno 24. svibnja 2024.]
- [2] M. S. Cornet, Implementing Procedural Narrative and Characters to Explore Player -NPC Relationships, WORCESTER POLYTECHNIC INSTITUTE, 2022, dostupno na: <https://digital.wpi.edu/concern/etds/sf268830g?locale=en> [pristupljeno 25. svibnja 2024.]
- [3] Red Dead Redemption 2 Review, IGN, dostupno na: <https://www.ign.com/articles/2018/10/25/red-dead-redemption-2-review> [pristupljeno 30. kolovoza 2024]
- [4] Matt Gibs, Nonplayable Characters (NPCs), rdr2.org, 2023, dostupno na: <https://www.rdr2.org/wiki/nonplayable-characters-npcs/> [pristupljeno 5. lipnja 2024.]
- [5] Slika je preuzeta s <https://screenrant.com/rdr2-towns-ranked-valentine-saint-denis-blackwater/> [pristupljeno 25. svibnja 2024.]
- [6] Niall Firth, How generative AI could reinvent what it means to play, MIT Technology Review, 2024, dostupno na: <https://www.technologyreview.com/2024/06/20/1093428/generative-ai-reinventing-video-games-immersive-npcs/> [pristupljeno 8. kolovoza 2024.]
- [7] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, Michael S. Bernstein, Generative Agents: Interactive Simulacra of Human Behavior, Association for Computing Machinery, 2023, dostupno na: <https://arxiv.org/abs/2304.03442> [pristupljeno 5. lipnja 2024.]
- [8] Ryan Volum, Sudha Rao, Michael Xu ,Gabriel DesGarennes, Chris Brockett, Benjamin Van Durme, Olivia Deng, Akanksha Malhotra, Bill Dolan, „Grounded Conversational Characters“, Microsoft Corporation, 2022, dostupno na: <https://www.microsoft.com/en-us/research/project/grounded-conversational-characters/> [pristupljeno 6. lipnja 2024.]

- [9] Convai tim, Integrating Dynamic NPC Actions for Game Development with Convai, Convai, 2024, dostupno na: <https://convai.com/blog/integrating-dynamic-npc-actions-for-game-development-with-convai> [pristupljeno 6. lipnja 2024.]
- [10] Unity (game engine), dostupno na: [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)) [pristupljeno 7. lipnja 2024.]
- [11] Dealessandri Marie, What is the best game engine: is Unity right for you?, GamesIndustry.biz, 2020, dostupno na: <https://www.gamesindustry.biz/what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you> [pristupljeno 7. lipnja 2024.]
- [12] Government & Aerospace, dostupno na: <https://unity.com/solutions/government-aerospace> [pristupljeno 8. lipnja 2024.]
- [13] JetBrains Rider, dostupno na: <https://www.jetbrains.com/rider/> [pristupljeno 10. lipnja 2024.]
- [14] C#, dostupno na: [https://en.wikipedia.org/wiki/C_Sharp_\(programming_language\)](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) [pristupljeno 10. lipnja 2024.]
- [15] LLM for Unity dokumentacija, dostupno na: <https://undream.ai/LLMUnity/> [pristupljeno 14. lipnja 2024.]
- [16] llama.cpp, dostupno na: <https://github.com/ggerganov/llama.cpp> [pristupljeno 4. lipnja 2024.]
- [17] What is Machine Learning?, IBM, 2023, dostupno na: <https://www.ibm.com/topics/machine-learning> [pristupljeno 15. lipnja 2024.]
- [18] Yoosefzadeh-Najafabadi, Mohsen; Hugh, Earl; Tulpan, Dan; Sulik, John; Eskandari, Milad (2021). „Application of Machine Learning Algorithms in Plant Breeding: Predicting Yield From Hyperspectral Reflectance in Soybean?“, Front. Plant Sci., 2021, dostupno na: <https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2020.624273/full> [pristupljeno 16. lipnja 2024.]
- [19] Applications of Machine Learning, geeksforgeeks, 2023, dostupno na: <https://www.geeksforgeeks.org/machine-learning-introduction/> [pristupljeno 16. lipnja 2024.]
- [20] Mohri, Mehryar; Rostamizadeh, Afshin; Talwalkar, Ameet, Foundations of Machine Learning., The MIT Press [pristupljeno 16. lipnja 2024.]

- [21] M. Emre Celebi, Kemal Aydin, Unsupervised Learning Algorithms, Springer, 2016 [pristupljeno 18. lipnja 2024.]
- [23] AlphaGo, dostupno na: <https://deepmind.google/technologies/alphago/> [pristupljeno 18. lipnja 2024.]
- [24] Hardesty L, Explained: Neural networks, MIT News Office, 2017, dostupno na: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414> [pristupljeno 19 lipnja 2024.]
- [25] James A. Anderson, An Introduction to Neural Networks, The MIT Press, 1995, dostupno na: https://books.google.hr/books?hl=hr&lr=&id=_ib4vPdB76gC&oi=fnd&pg=PP11&dq=neural+networks&ots=WdarTctMZ9&sig=-NfJx35h53xeEzTeriD0x3ezolI&redir_esc=y#v=onepage&q=neural%20networks&f=false [pristupljeno 20. lipnja 2024.]
- [26] Ratko Grbić, Raspoznavanje uzoraka i strojno učenje Neuronske mreže. Potpuno povezane mreže, FERIT, 2022 [pristupljeno 20. lipnja 2024.]
- [27] Antoniadis, Panagiotis, Hidden Layers in a Neural Network | Baeldung on Computer Science, Baeldung, 2024, dostupno na: <https://www.baeldung.com/cs/hidden-layers-neural-network> [pristupljeno 20. lipnja 2024.]
- [28] SAGAR SHARMA, Activation Functions in Neural Networks, Towards Data Science, 2017, dostupno na: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6> [pristupljeno 20. lipnja 2024.]
- [29] Multi-Class Neural Networks: Softmax, dostupno na: <https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax> [pristupljeno 20. lipnja 2024.]
- [30] GPT-4, dostupno na: <https://openai.com/index/gpt-4-research/> [pristupljeno 20. lipnja 2024.]
- [31] What are large language models (LLMs)?, IBM, dostupno na: <https://www.ibm.com/topics/large-language-models> [pristupljeno 21. lipnja 2024.]

- [32] Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N; Kaiser, Łukasz; Polosukhin, Illia, Attention is All you Need. Advances in Neural Information Processing Systems. 30. Curran Associates, Inc., 2017, dostupno na: <https://arxiv.org/abs/1706.03762> [pristupljeno 21. lipnja 2024.]
- [33] Rani Horev, BERT Explained: State of the art language model for NLP, Towards Data Science, 2018, <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270> [pristupljeno 22. lipnja 2024.]
- [34] DALL·E 2, dostupno na: <https://openai.com/index/dall-e-2/> [pristupljeno 22. lipnja 2024.]
- [35] Introducing Whisper, dostupno na: <https://openai.com/index/whisper/>, 2022 [pristupljeno 22. lipnja 2024.]
- [36] Large Language Model, dostupno na: https://en.wikipedia.org/wiki/Large_language_model [pristupljeno 22. lipnja 2024.]
- [37] Rico Sennrich, Barry Haddow, Alexandra Birch, Neural Machine Translation of Rare Words with Subword Units, School of Informatics, University of Edinburgh, 2016, dostupno na: <https://arxiv.org/abs/1498.07909> [pristupljeno 23. lipnja 2024.]
- [38] Marie-Alice Blete, LLMs: Determinism & Randomness, 2023, Medium, dostupno na: <https://medium.com/@mariealice.blete/llms-determinism-randomness-36d3f3f1f793> [pristupljeno 23. lipnja 2024.]
- [39] Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone, dostupno na: <https://arxiv.org/abs/2404.14219> [pristupljeno 23. lipnja 2024.]
- [40] Edouard d'Archimbaud, The Challenges, Costs, and Considerations of Building or Fine-Tuning an LLM, 2023, dostupno na: <https://hackernoon.com/the-challenges-costs-and-considerations-of-building-or-fine-tuning-an-llm> [pristupljeno 25. lipnja 2024.]
- [41] Parameter-Efficient Fine-Tuning (PEFT), dostupno na: https://docs.nvidia.com/nemo-framework/user-guide/latest/nemotoolkit/nlp/nemo_megatron/peft/landing_page.html [pristupljeno 25. lipnja 2024.]
- [42] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, LoRA: Low-Rank Adaptation of Large Language Models, 2021,

Microsoft Corporation, dostupno na: <https://arxiv.org/abs/2106.09685> [pristupljeno 23. lipnja 2024.]

[43] Welcome to Colab!, dostupno na: <https://colab.research.google.com/> [pristupljeno 25. lipnja 2024.]

[44] Python, <https://www.python.org/> [pristupljeno 8. srpnja 2024.]

[45] Jupyter, dostupno na: <https://jupyter.org/> [pristupljeno 25. lipnja 2024.]

[46] Hugging Face Hub documentation, dostupno na: <https://huggingface.co/docs/hub/index> [pristupljeno 25. lipnja 2024.]

[47] Pytorch, dostupno na: <https://pytorch.org/> [pristupljeno 25. lipnja 2024.]

[48] Prompt engineering, dostupno na: <https://platform.openai.com/docs/guides/prompt-engineering/strategy-write-clear-instructions> [pristupljeno 26. lipnja 2024.]

[49] Sqlite, dostupno na: <https://www.sqlite.org/> [pristupljeno 27. lipnja 2024.]

[50] JSON, dostupno na: <https://www.json.org/json-en.html> [pristupljeno 27. lipnja 2024.]

[51] Pandas Dataframe, dostupno na: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html> [pristupljeno 27. lipnja 2024.]

[52] Low Poly Dungeons Lite, dostupno na: <https://assetstore.unity.com/packages/3d/environments/dungeons/low-poly-dungeons-lite-177937> [pristupljeno 27. lipnja 2024.]

[53] NavMesh, dostupno na: <https://docs.unity3d.com/ScriptReference/AI.NavMesh.html> [pristupljeno 27. lipnja 2024.]

[54] SQL, dostupno na: [What is SQL \(w3schools.com\)](https://www.w3schools.com/sql/) [pristupljeno 28. lipnja 2024.]

[55] Sencince similarity, dostupno na: <https://huggingface.co/tasks/sentence-similarity> [pristupljeno 29. lipnja 2024.]

[56] ML-Agents, github projekta, dostupno na: <https://github.com/Unity-Technologies/ml-agents> [pristupljeno 1. kolovoza 2024.]

SAŽETAK

U ovom diplomskom radu su istražene i primijenjene tehnike i alati iz područja umjetne inteligencije za dinamičko modeliranje ponašanja neigrivih likova u videoigrama. Kroz korištenje velikih jezičnih modela, razvijen je sustav koji omogućuje realističnu i složenu interakciju likova unutar virtualnih okruženja. Implementacija je obuhvatila kreiranje skupa podataka, podešavanje velikog jezičnog modela i integraciju modela u Unity okruženje, čime je postignuta veća razina autonomije i prilagodljivosti neigrivih likova. Ovaj rad doprinosi razumijevanju i unapređenju tehnika za kreiranje inteligentnih i dinamičnih neigrivih likova, što može značajno poboljšati iskustvo igrača.

Ključne riječi: neigrivi likovi, umjetna inteligencija, Unity, veliki jezični modeli, videoigre

ABSTRACT

Application of artificial intelligence in dynamic modeling of the behavior of non-playable characters

This thesis explores and applies techniques and tools from the field of artificial intelligence for dynamic modeling of non-playable character (NPC) behavior in video games. By utilizing large language models, a system has been developed that enables realistic and complex interactions of characters within virtual environments. The implementation included creating a dataset, fine-tuning a large language model, and integrating the model into the Unity environment, resulting in a higher level of autonomy and adaptability for NPCs. This work contributes to the understanding and advancement of techniques for creating intelligent and dynamic NPCs, which can significantly enhance the player's experience.

Keywords: Artificial intelligence, large language models, non-playable characters, Unity, video games

PRILOZI

P.5.1 Kompletan programski kôd projekta

Dostupan na: <https://github.com/aomazic/Project-E>