Razvoj razgovornog agenta za dinamičan dijalog u virtualnom okruženju temeljenog na primjeni umjetne inteligencije

Čičak, Luka

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:062743

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-22**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 441

RAZVOJ RAZGOVORNOG AGENTA ZA DINAMIČAN DIJALOG U VIRTUALNOM OKRUŽENJU TEMELJENOG NA PRIMJENI UMJETNE INTELIGENCIJE

Luka Čičak

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 441

RAZVOJ RAZGOVORNOG AGENTA ZA DINAMIČAN DIJALOG U VIRTUALNOM OKRUŽENJU TEMELJENOG NA PRIMJENI UMJETNE INTELIGENCIJE

Luka Čičak

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 441

Pristupnik:	**Luka Čičak (0036528205)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Mirko Sužnjević

Zadatak: **Razvoj razgovornog agenta za dinamičan dijalog u virtualnom okruženju temeljenog na primjeni umjetne inteligencije**

Opis zadatka:

Primjena razgovornih agenata (engl. chatbotova) temeljenih na umjetnoj inteligenciji (UI) u igrama postaje sve značajnija, pružajući igračima interaktivno iskustvo putem prirodnog jezika. Razgovorni agenti se često koriste za poboljšanje korisničke podrške, pružajući igračima brze odgovore na pitanja ili rješavanje problema unutar igre. Dodatno, integracija razgovornih agenata u igre omogućuje dinamičan dijalog između likova, unapređujući narativni aspekt igre i dodajući dubinu interakciji između igrača i virtualnog svijeta. Vaš zadatak je izraditi igru u kojoj će se omogućiti razgovor s računalno kontroliranim likom kroz razgovornog agenta. Potrebno je spojiti virtualno okruženje igre s vanjskim velikim jezičnim modelom (engl. Large Language Model, skr. LLM). Nužno je omogućiti konfiguraciju LLM-a u smislu postavljanja karakteristika računalno kontroliranog lika (primjerice, veseo, razgovorljiv i slično), ali i u smislu tehničkih parametara kao što su pauza prije početka odgovora te brzina generiranja riječi. Razvijena aplikacija koristit će se u svrhu znanstvenih testiranja te u nastavne svrhe, te je potrebno napisati i detaljne upute za korištenje i konfiguraciju, kao i moguće zadatke kroz koje bi studenti učili o primjeni razgovornih agenata temeljenih na LLM-ovima u virtualnim okruženjima.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

Uvod 4
1 Pregled literature 4
1.1 Non-player characters 4
1.2 Veliki jezični modeli 5
1.3 Razgovorni agenti u virtualnim okruženjima 6
1.3.1 NEO NPC 6
1.3.2 Convai 6
1.4 Unity 7
2 Opis formalnog modela i metodologije 8
2.1.1 Funkcionalnosti modela 8
2.1.2 Glavni izbornik 8
2.1.3 Izbornik pauze 9
2.1.4 Tradicionalni NPC 9
2.1.5 NPC s razgovornim agentom 9
2.1.6 Upravljanje predmetima 10
2.1.7 Resetiranje igre 10
3 Implementacija formalnoga modela 10
3.1 Specifikacije sustava 10
3.2 Specifikacije razgovornog agenta 11
3.3 Programski alati 11
3.3.1 Unity 11
3.3.2 Rider 11
3.4 Glavni izbornik 12
3.4.1 Korisničko sučelje 12
3.4.2 Klasa MainMenu 14
3.4.3 Klasa OptionsMenu 16
3.5 Izbornik pauze 18
3.5.1 Korisničko sučelje 18
3.5.2 Klasa PauseMenu 18
3.6 Tradicionalni NPC 19
3.6.1 Korisničko sučelje 19
3.6.2 Klasa Npc 20
3.6.3 Klasa StandardNpc 22
3.6.4 Klasa Dialogue 25
3.6.5 Klasa DialogueNode 26
3.6.6 Klasa SimpleDialogueNode 26
3.6.7 Klasa ChoiceDialogueNode 27
3.6.8 Klasa DialogueController 27
3.7 NPC s razgovornim agentom 30
3.7.1 Korisničko sučelje 30
3.7.2 Klasa ChatbotNpc 30
3.7.3 Klasa LlamaCppCom 34
3.7.4 Klasa Payload 36

3.7.5 Klasa Response 36
3.7.6 Klasa RewardLine 36
3.7.7 Klasa SetChatbotNpc 37
3.8 Upravljanje predmetima 38
3.8.1 Korisničko sučelje 38
3.8.2 Klasa PlayerInventory 38
3.8.3 Klasa PickUpItem 39
3.8.4 Klasa NotificationManager 40
3.9 Resetiranje igre 42
3.9.1 Korisničko sučelje 42
3.9.2 Klasa EndGame 43
4 Rezultati i diskusija 45
Zaključak 52
Literatura 53
Sažetak 54
Summary 55

Uvod

Računalne igre su oduvijek težile što većem realizmu, s ciljem stvaranja upečatljivih i što više imerzivnih iskustava za igrače kako bi se oni što više uživili u svijet igre. Tijekom godina, napredak u grafičkim procesorskim jedinicama (GPU) i središnjim procesorskim jedinicama (CPU) značajno je poboljšao vizualni realizam u igrama, brzinu njihovih izvođenja te sam obujam igara. Napredovali su i specijalni efekti, zvukovi koji se koriste za videoigre te ispravnost simulacija fizike. Međutim, jedna stvar koja još uvijek zaostaje po pitanju realizma i sprječava potpunu imerziju su likovi u videoigrama koji nisu igrač (*engl. Non-player characters*) ili skraćeno NPC-evi. Oni igraju ključnu ulogu u naseljavanju svjetova igara i povećavanju dubine iskustva. Problem je kod njih je što se oslanjaju se na skriptirane dijaloge i ponašanja koja dovode do ponavljajućih i predvidljivih interakcija te iskustava. Jedno moguće rješenje za taj problem na nudi tehnologija koja se velikom brzinom pojavila i razvila u proteklih nekoliko godina, umjetna inteligencija. Umjetna inteligencija ima velik potencijal za primjenu u videoigrama i nudi mogućnost generiranja kontekstualno relevantnih razgovora, čineći interakcije s NPC-evima dinamičnijima i personaliziranijima. Ovaj rad se bavi upravo razvojem aplikacije u Unity pogonskome sustavu koja integrira razgovornog agenta (*engl. Chatbot*) tj. pokazne videoigre koja primjenjuje tehnologiju umjetne inteligencije tako što omogućuje integraciju istreniranog velikog jezičnog modela (*engl. Large Language Models*) u igru umjesto jednog od dostupnih NPC-eva. Aplikacija je namijenjena za pomoć testiranju moguće razine istreniranosti modela.

1 Pregled literature

1.1 Non-player characters

Likovi koji nisu igrač (*engl. Non-player characters skr. NPC*) u videoigrama služe kao ključni elementi u pripovijedanju, interakciji i napredovanju igranja. Bez njih, okruženje bi se činilo beživotnim i potpuno nerealističnim te bi se izgubio osjećaj imerzije. Napretku igrača nedostajali bi smjer i svrha. Neki poznatiji NPC-evi koji su jako utjecali na doživljaj i naraciju igre su Cortana iz igre Halo, GLaDOS iz igre Portal i The Merchant iz igre Resident Evil. Nije pretjerano reći da su NPC-evi jedna od najvažnijih komponenti videoigara. Unatoč tome, ne možemo reci da su NPC-evi u danome trenutku onoliko napredni i inteligentni koliko bismo htjeli da jesu. Može se i reći da su jedan od glavnih faktora, ako ne i glavni, koji smanjuju realističnost i imerziju videoigara. Problem je u tome što je stvaranje realističnih NPC-eva i dalje je izazov unatoč napretku tehnologije. Stoga se počinje eksperimentirati s primjenom velikih jezičnih modela koji bi zamijenili tradicionalne NPC-eve čineći interakcije dinamičnijima i personaliziranijima te igre realističnijima [5]. Kako bismo bolje razumjeli NPC-eve i njihovu ulogu, pogledajmo ukratko njihovu povijest. Smatra se da je izraz je NPC porijeklom iz stolnih RPG-eva iz 1970-ih kao što je, daleko najpoznatiji, Dungeons & Dragons. Osmišljen je kako bi se razlikovali likovi kojima upravlja voditelj igre u odnosu na igrače. NPC-evi su u početku pružali informacije, misije i druženje, no pojavom računalnih igara, NPC-evi su postajali složeniji, služeći ulogama od davatelja zadataka do ispunjavanja atmosfere u igrama otvorenog svijeta. Današnje mogućnosti NPC-eva izgrađene su na generacijama inkrementalnih poboljšanja. Krenuvši od osamdesetih i ranijih devedesetih

godina prošloga stoljeća javljaju se prvi NPC-evi. Njihova ponašanja bila su potpuno kodirana *(engl. Hard-coded)* uz unaprijed određene staze, dopuštajući minimalnu interaktivnost. U kasnijim devedesetima javljaju se sustavi rasporeda *(engl. Scheduling systems)*. NPC-evi su bili dinamičniji tako što su imali dnevni/noćni raspored za osnovne dnevne rutine. Ranije dvije tisućite su donijele napredak koji je omogućio grananje stabala dijaloga i kontekstualne odluke o ponašanju koristeći stabla odlučivanja tj. stablastu strukturu podataka. U dvije tisuće desetima neke su igre eksperimentirale sa strojnim učenjem, poput treniranja animacija NPC-eva na podacima o ljudskom kretanju. Danas se eksperimentira s dubokim neuronskim mrežama i obradom prirodnog jezika koje imaju potencijal jednog dana omogućiti gotovo ljudske interakcije. Moderne igre kao što su Witcher 3, Grand Theft Auto V ili Red Dead Redemption 2, pokazuju dokle su interakcije dogurale s proceduralnim dijalogom, dinamičnim odnosima i kontekstualnim interakcijama, no još uvijek nismo blizu istinski inteligentnim NPC-evima. [6]

1.2 Veliki jezični modeli

Veliki jezični modeli (LLM) odnose se na vrstu modela dubokog učenja koji se treniraju na golemim količinama tekstualnih podataka i koji koriste mogućnosti obrade prirodnog jezika za učinkovito razumijevanje, generiranje i manipuliranje ljudskim jezikom [7]. Kako se treniraju na golemim količinama podataka, to im omogućuje da budu izvrsni u različitim jezičnim zadacima kao što je razumijevanje prirodnog jezika svjesno konteksta (*engl. Natural language processing*), generiranje teksta, prevođenje i sažimanje. U trenutku pisanja ovoga rada dostupna je nekolicina velikih jezičnih modela, ali ih se sve više i više razvija kako raste potražnja za njihovom primjenom u raznim industrijama. Neki od popularnih modela su ChatGPT kojeg razvija OpenAI, Gemini kojeg razvija Google te Llama kojeg razvija Meta. Ovakvi modeli što danas podrazumijevamo pod pojmom veliki jezični model postoje tek od 2017. godine, ali nekoliko jezičnih modela koji su bili veliki u usporedbi s tada dostupnim kapacitetima postojali su i dosta prije. U devedesetim godinama prošloga stoljeća IBM-ovi modeli usklađivanja bili su pioniri u modeliranju statističkog jezika. U dvije tisućitima, s eksponencijalno rastućom popularnosti interneta, neki su istraživači konstruirali jezične skupove prikupljene s interneta, na temelju kojih su uvježbavali statističke jezične modele. U 2009. godini, u većini zadataka obrade jezika, statistički jezični modeli dominirali su nad simboličkim jezičnim modelima, jer su bili u mogućnosti korisno unositi velike skupove podataka. Uvođenje neuronskih mreža oko 2012. godine označilo je ključni pomak. Rekurentne neuronske mreže (*engl. Recurrent neural network, skr. RNN*) i mreže dugog kratkoročnog pamćenja (*engl. Long short-term memory, skr. LSTM*) bile su među prvima koje su pokazale značajna obećanja u rješavanju jezičnih zadataka. Te su mreže bile sposobne uhvatiti ovisnosti u tekstu, što je bio ključni korak prema sofisticiranijim modelima. No najveći napredak dogodio se 2017. s pojavom Transformer arhitekture koju je Google predstavio na NeurIPS konferenciji. Model Transformer koristi mehanizme samoopažanja za obradu teksta, značajno poboljšavajući rukovanje dugotrajnim ovisnostima i omogućavajući paralelizaciju tijekom treniranja. Ova je arhitektura od tada postala temelj za većinu modernih LLM-ova. Serija modela Generative Pre-trained Transformer (*skr. GPT*) od tvrtke OpenAI, bazirana na Transformer modelu, bila je posebno utjecajna. Prva iteracija pojavila se 2018. godine i svaka njegova iteracija od tada vidjela je eksponencijalno povećanje veličine modela i izvedbe. Verzija GPT-4 hvaljena je zbog svoje povećane točnosti i zbog svojih multimodalnih

mogućnosti. Od 2022. pojavljuje se i velik broj konkurentskih jezičnih modela koji su pokušavali izjednačiti seriju GPT, barem u smislu broja parametara. Neki od poznatijih su Llama, BLOOM i MistralAI. Ovi modeli prikazali su potencijal LLM-ova u različitim primjenama, uključujući prevođenje, sažimanje, odgovaranje na pitanja i kreativno pisanje [8].

1.3 Razgovorni agenti u virtualnim okruženjima

Veliki jezični modeli sve više nalaze svoj put u virtualna okruženja, posebno u industriji igara, poboljšavajući interaktivno i impresivno iskustvo za igrače. Imaju potencijal poboljšati iskustvo igrača s bogatim i dinamičnim sadržajem, pojednostaviti proces razvoja igre kroz automatizaciju i omogućiti bolju personalizaciju i prilagodbu preferencijama igrača. Veliki jezični modeli pojavili su se krajem 2010-ih, ali počeli su viđati širu upotrebu u igrama tek početkom 2020-ih pojavom značajno moćnijih modela kao što je npr. GPT serija koju je stvorio OpenAI. No toliko moćniji modeli donose i svoje probleme. Potrebne su vrlo velike količine memorije i računanja za rad takvih modela i zato ih nije moguće koristiti lokalno kod klijenta. Jedina opcija trenutno je da se razgovorni agenti onda nalaze na nekome sučelju (*engl. Application programming interface, skr. API*) poslužitelju i da klijent mora slati zahtjeve. Zbog samog noviteta tehnologije i navedenih tehnoloških i financijskih prepreka, igre koje koriste razgovorne agente za generiranje sadržaja još uvijek su u dosta malome broju i ne vidimo ih još uvijek od većih kompanija na tržištu. Neki primjeri igara su AI Dungeon 2, AI Rougelite, Gandalf, Minecraft Negotiation, Nepogotchi, Under The Influencer i još nekolicina drugih. Ova grupa videoigara tijekom igranja generiraju nov sadržaj u hodu, obično kao odgovor na radnje igrača. To može uključivati: generiranje novog teksta u igri, generiranje zadataka i scenarija, generiranje statistike i više. Što se tiče korištenja razgovornih agenata kao zamjene za standardne NPC-eve, oni još nisu integrirani u niti jednu komercijalnu igru, ali su Ubisoft i Nvidia predstavili su nekoliko obećavajućih rješenja u obliku NEO NPC i Convai projekata koje bismo mogli ubrzo vidjeti u primjeni. Oba okvira koriste razgovorne agente koji se nalaze na poslužiteljima, ali sa sklopljenim partnerstvom s Nvidijom to bi se vrlo brzo moglo promijeniti.

1.3.1 NEO NPC

Ubisoftov NEO NPC projekt, razvijen s Nvidijom i Inworldom, koristi generativnu umjetnu inteligenciju za stvaranje NPC-eva sposobnih za dinamične i personalizirane razgovore s korisnicima. NPC-eve najprije ručno stvaraju pisci, a zatim ih podešavaju iterativnim procesima kako bi osigurali da ostanu vjerni osobnostima za koje su dizajnirani. Razgovorni agenti vođeni su narativnim strukturama i filtrima kako bi se održala cjelovitost priče i spriječila toksičnost. Projekt je predstavljen na konferenciji Game Developers Conference 2024 i ima za cilj integrirati ljudsku kreativnost s razgovornim agentima kako bi se poboljšala iskustva igranja u raznim Ubisoftovim projektima [9].

1.3.2 Convai

Convai je napravio napredne verzije NPC-eva koji koriste tehnologiju umjetne inteligencije i dizajnirani su za poboljšanje interaktivnosti, realizma te imerzije u virtualnim svjetovima. Opremljeni su značajkama kao što su glasovne interakcije u stvarnom vremenu, odgovori

svjesni konteksta i mogućnost izvođenja složenih radnji [10]. Ovi NPC-evi mogu razumjeti svoje okruženje i komunicirati s njim, izvršavati zadatke u više koraka i sjećati se prošlih interakcija s igračima, pružajući impresivnije i dinamičnije iskustvo igranja [10]. Platforma također podržava stvaranje NPC-eva sa specijaliziranim bazama znanja kako bi se osigurali točni i relevantni odgovori, minimizirajući probleme poput „haluciniranih“ informacija uobičajenih za velike jezične modele. Ovu tehnologiju moguće je koristiti u popularnim pogonskim sustavima kao što su Unity i Unreal Engine, čineći ga programerima dostupnim za ugradnju inteligentnih NPC-eva u svoje igre [11]. Ovaj je okvir (*engl. Framework*) uspješno testiran u popularnim igrama kao što su Hogwarts Legacy i Cyberpunk 2077, što već pokazuje veliki potencijal značajnog obogaćivanja iskustva u videoigrama.

1.4 Unity

Unity Game Engine je višeplatformski softver za razvoj igara koji razvija kompanija Unity Technologies. Uz Unreal Engine i Cry Engine smatra se najboljim za izradu digitalnih igara i za njima jedino zaostaje u pogledu realističnih grafika. Iako mu grafike nisu jača strana, Unity se ističe po tome što je jedini od prethodno nabrojanih razvojnih softvera koji izvorno podržava izradu 2D digitalnih igara s posebnim editorom i posebnim klasama za 2D igre. Također se može pohvaliti s puno većom zajednicom korisnika od ostatka, što dosta olakšava početnicima da dobiju odgovore na svoje probleme, detaljnom dokumentacijom i vrlo bogatim Asset Store-om. Logiku igre moguće ga je definirati na dva načina, programskim jezikom C# ili pomoću ugrađenog alata za vizualno skriptiranje Bolt. Čak i za osobe koje nisu nikada programirale preporučeno je koristiti C# zbog toga što Bolt još uvijek nije dovoljno zreo da bi se mogao koristiti za kompleksnije aplikacije. Unity podržava izradu igra za značajno veću količinu platformi u odnosu na konkurenciju, a one su sljedeće: iOS, Android, Tizen, Windows, Universal Windows Platform, macOS, Linux, WebGL, PlayStation 4 i 5, PlayStation Vita, Xbox One, , Xbox Series X i S, Wii U, 3DS, Oculus Rift, Google Cardboard, SteamVR, PlayStation VR, Gear VR, Windows Mixed Reality, Daydream, Android TV, Samsung Smart TV, tvOS, Nintendo Switch, Fire OS, Facebook Gameroom, Apple ARKit, Google ARCore i Vuforia. Unity je prvi puta izdan 2005. godine s ciljem da omogući razvoj videoigara što većem broju ljudi. Platforma na kojoj je Unity krenuo jest Appleov Mac OS X, a verzija za Windows operacijske sustave dodana je tek kasnije [3]. Tijekom godina pogonski sustav se nastavio razvijati velikom brzinom i stalno je dodavao nove značajke te usvajao nove tehnologije, kako bih uvijek ostao moderan [3]. 2007. godine izdan je Unity 2.0 koji je uključivao optimizirani terenski motor za detaljna 3D okruženja, dinamičke sjene u stvarnom vremenu, usmjerena svjetla i reflektore, reprodukciju videa, podršku razvoja za iOS i druge značajke [3]. Izdanje je također prvoga puta omogućilo razvoj igara za više igrača, nudeći prijevod mrežnih adresa, sinkronizaciju stanja i pozive udaljenih procedura. Unity 3.0 bio je izdan 2010. godine sa značajkama koje proširuju njegove grafičke sposobnosti za videoigre na računalima i konzolama, uvedena je podrška za Android, odgođeno renderiranje, ugrađeni uređivač stabla, izvorno renderiranje fontova, automatsko UV mapiranje i audio filteri [3]. Verzija 4.0 izdana je u studenom 2012. godine. Verzija je dodala podršku za DirectX 11 i Adobe Flash, nove alate za animaciju Mecanim i podršku za sustave Linux [3]. Unity 5.0 nudio je globalno osvjetljenje u stvarnom vremenu, preglede mapiranja svjetla, Unity Cloud, novi audio sustav, Nvidia PhysX 3.3, efekte kinematografske slike kako bi Unity igre izgledale manje generičke [3]. Također je programerima omogućeno putem WebGL-a, dodati svoje igre

u kompatibilne web preglednike. Od prosinca 2016. promijenjen je sustav numeriranja verzija za Unity iz sekvencijskog verzioniranja u verzioniranje predstavljeno godinom izdanja [4]. Moderni Unity je poboljšanje u svim značajkama naprema starijim izdanjima i nastavlja korisnicima što više olakšati proces razvijanja igara. Također u ovome razdoblju Unity Technologies je otkupio mnogo korisnih alata koje je ugradio u svoj pogonski sustav kao što su Bolt, Ziva Dynamics, Weta Digital…

2 Opis formalnog modela i metodologije

U ovome poglavlju bit će objašnjen formalni model koji opisuje kako traženo programsko rješenje treba funkcionirati tj. opis njegovih funkcionalnosti, bez ulaženja u implementacijske detalje tako da ga je moguće primijeniti u proizvoljnome programskome jeziku i programskome alatu.

2.1.1 Funkcionalnosti modela

Prije opisivanja pojedinačnih dijelova programa, potrebno je definirati sve glavne funkcionalnosti koje je potrebno implementirati. Konkretno, u slučaju razgovornog agenta potrebne su sljedeće osnovne funkcionalnosti: glavni izbornik, izbornik za pauziranje, implementirati kontrole igrača (ovo podrazumijeva kretanje igrača i njegovu interakciju s okolinom), vođenje razgovora s tradicionalnim NPC-evima, dobivanje i davanje predmeta tradicionalnim NPC-evima, vođenje razgovora s NPC-em povezanim s vanjskim velikim jezičnim modelom, dobivanje i davanje predmeta NPC-u povezanim s vanjskim velikim jezičnim modelom, resetiranje igre, postavljanje parametra za povezivanje i za konfiguraciju NPC-a povezanim s vanjskim velikim jezičnim modelom, sakupljanje predmeta, inventar u kojem će se čuvati sakupljeni i dobiveni predmeti, napredak u igri te kraj igre.

2.1.2 Glavni izbornik

Glavni izbornik treba nuditi nekoliko funkcionalnosti: mogućnost pokretanja igre, mogućnost odabira web adrese gdje se nalazi vanjski veliki jezični model, mogućnost unosa konfiguracije vanjskog velikog jezičnog modela, mogućnost odabira kojeg NPC-a želimo zamijeniti s vanjskim velikim jezičnim modelom te mogućnost napuštanja igre. Zamišljeno je da se te funkcionalnosti izbornika raspodjele na dvije sekcije. Prva sekcija se sastoji od triju gumbi i to su gumb PLAY koji pokreće igru, gumb OPTIONS koji otvara drugu sekciju te gumb EXIT koji napušta igru. Druga sekcija je zaslužna za konfiguraciju igre i sastoji se od standardnog polja za unos teksta (*engl. Input field*) u kojeg se unosi web adresa vanjskog velikog jezičnog modela, višelinijskog polja za unos teksta (*engl. Text area*) u kojeg se unosi konfiguracija vanjskog velikog jezičnog modela, padajućeg izbornika (*engl. Dropdown menu*) kojim se odabire koji će se NPC zamijeniti vanjskim velikim jezičnim modelom te gumba BACK koji služi za vraćanje na prvu sekciju izbornika.

2.1.3 Izbornik pauze

Izbornik za pauziranje igre mora igraču omogućiti nastavi trenutnu igru te da se vrati na glavni izbornik, a pritom korisniku mora biti onemogućena interakcija s igrom. Izbornik se treba pojaviti i sakriti kad god igrač pritisne tipku Escape na tipkovnici. Moguće je odabrati neku proizvoljnu tipku, no preporučuje se tipka Escape zato što će korisnicima biti intuitivno da je ona zaslužna za pauziranje. Ovo je najjednostavnije postići tako da koristimo jednu varijablu tipa boolean koja nam označava stanje izbornika, a na pritisak tipke ćemo mijenjati njeno stanje te ovisno je li varijabla istinita ili neistinita, prikazati ili sakriti izbornik. Na pritisak gumba RESUME sakrit će se izbornik i korisniku će se ponovo omogućiti interakcija s igrom. Pritiskom gumba EXIT TO MENU ćemo jednostavno vratiti korisnika na glavni izbornik.

2.1.4 Tradicionalni NPC

Tradicionalnog NPC-a možemo napraviti tako da unaprijed odredimo razgovore koje je s njima moguće voditi nakon što s njima stupimo u interakciju. Razgovori mogu biti slijedni, ali i mogu imati puno grananja, pa da bismo to simulirali potrebna nam je stablasta struktura podataka u koju ih možemo spremiti. Svaki razgovor će imati početni čvor, više internih čvorova i jedan ili više završnih čvorova. Što se tiče vrsta čvorova po broju potomaka, imati ćemo jednostavne čvorove koji imaju nijednog ili jednog potomka te čvorove grananja koji će imati između jednog ili triju odabira. Broj odabira je ograničen na tri radi jednostavnosti. NPC će onda imati listu razgovora u obliku stablastih struktura koje će obilaziti kada je s igračem u nekoj interakciji. Igrač može stupiti u interakciju s NPC-em ako mu je manje ili jednako udaljen od NPC-a nego li neka fiksna udaljenost koju smo zadali te ako je u NPC u igračevom vidokrugu. Ako su uvjeti zadovoljeni pojaviti će se poruka da je interakcija moguća i ako igrač pritisne tipku E (ili neku proizvoljnu tipku) na tipkovnici, pokrenuti će se prvi razgovor u listi. Istom tipkom možemo napredovati jednostavnim čvorovima razgovora, dok ćemo za čvorove s odabirima igraču prezentirati odabire pomoću gumbi koje je moguće pritisnuti lijevom tipkom miša. Kada smo gotovi s jednim razgovorom on se briše iz liste kako bi osigurali napredovanje. Tekstove razgovora ćemo prikazati pomoću spremnika teksta koji iza sebe ima pozadinu.

2.1.5 NPC s razgovornim agentom

NPC s razgovornim agentom je nešto lakši za implementirati jer nam nisu potrebna spremišta podataka niti njihov obilazak kao kod tradicionalnog NPC-a. Umjesto toga sagraditi ćemo korisničko sučelje sa višelinijskim poljem za unos teksta preko kojeg ćemo slati poruke razgovornom agentu. Odgovore ćemo kao i kod tradicionalnog NPC-a prikazati pomoću spremnika teksta koji iza sebe ima pozadinu. Dok čekamo odgovor od razgovornog agenta, u polje za odgovore ćemo dodati animaciju koja prikazuje jednu, dvije, pa tri točke u beskonačnost. Ovako igraču signaliziramo da se nešto događa dok čeka odgovor. U interakciju stupamo na identičan način kao i s tradicionalnim NPC-em gdje igrač mora biti na dovoljnoj udaljenosti i NPC mora biti u igračevom vidokrugu. Kako imamo zajedničko ponašanje, dobra praksa je izdvojiti funkcionalnost u zajedničku apstraktniju klasu koju će oboje nasljeđivati Interakciju završavamo klikom na gumb END CONVERSATION koji dodajemo u sučelje.

2.1.6 Upravljanje predmetima

U igri je za napredak nekih od zadataka potreban nekakav predmet, a moguće je i dobiti predmet kao nagradu za izvršavanje nekog zadatka. Stoga igraču mora biti omogućeno sakupljanje predmeta iz okoline, čuvanje predmeta u inventaru te davanje i primanje predmeta od NPC-eva. Inventar predmeta biti će jednostavna klasa koja sadrži listu predmeta i definira metode za dodavanje u listu te provjeru nalazi li se predmet u listi. Dodatno, kako bismo imali dobar dizajn igre, kada dodamo neki predmet u inventar o tome ćemo obavijestiti igrača tako da mu pokažemo notifikaciju o tome. Također, kako želimo koristiti inventar u više drugih komponenti i pritom izbjeći gomilu prosljeđivanja, potrebno je koristiti Singleton oblikovni obrazac. Sakupljanje će funkcionirati slično kao i interakcija s NPC-evima, gdje ako se dovoljno približimo predmetu pojaviti će se poruka da ga je moguće sakupiti i dodati u inventar tako da pritisnemo tipku E na tipkovnici. Sva logika će se nalaziti na samome predmetu, a ne igraču. Što se tiče davanja i primanja predmeta od NPC-eva, logika za to će biti smještena na samim NPC-evima i to na obje vrste posebno. Kod tradicionalnih NPC-eva proširiti ćemo stabla dijaloga tako da mogu imati definiran potreban predmet za napredak i blokirati njihov prolazak dok uvjet nije zadovoljen te da mogu imati predmet koji nude kao nagradu koju onda dodajemo u inventar. Za NPC-eve s razgovornim agentom ćemo svu logiku staviti na njih zato što oni ne koriste stabla. Definirati ćemo na njima predmete potrebne za napredak te nagrade koje daju zauzvrat. Davanje predmeta ćemo implementirati tako da dodamo gumb GIVE ITEM u sučelje kada smo u interakciji s NPC-em, koji će se pojaviti samo ako NPC ima definiran potreban predmet i ako se on nalazi u inventaru. Gumb će jednostavno pozvati metodu koja šalje poruku razgovornome agentu da smo mu dali predmet. Igrača ne možemo spriječiti da sam pošalje tu poruku, ali možemo spriječiti da dobije nagradu provjerom posjeduje li predmet za napredak. Dobivanje predmeta je nešto kompleksnije jer bismo trebali analizirati svaki tekst koji dođe od NPC-a i vidjeti da li nam daje predmet. Da bismo si olakšali život i unaprijedili performanse, definirati ćemo u konfiguraciji razgovornog agenta da kada igraču daje predmet da kaže egzaktno neku frazu iz skupa predefiniranih fraza.

2.1.7 Resetiranje igre

Resetiranje igre će biti omogućeno tek kada je korisnik uspješno obavio zadatak potreban za završiti igru. Korisniku će se prikazati poruka da je uspješno završio igru te će se ispod poruke nalaziti gumb RESET, koji će kako i samo ime kaže, pokrenuti igru ispočetka tako da će ponovo učitati scenu i time vratiti sve podatke na njihova početna stanja. Dok je prikazan ovaj izbornik, korisniku je onemogućena interakcija s igrom.

3 Implementacija formalnoga modela

3.1 Specifikacije sustava

Rješenje je razvijeno i isprobano na osobnom računalu s operacijskim sustavima Windows 11. Stolno računalo ima procesor AMD Ryzen 5 2600X i grafičku karticu AMD Radeon RX580. Samo programsko rješenje ne ovisi o operacijskome sustavu i može se pokrenuti u Unity-u

(naravno s odgovarajućom verzijom) s bilo kojeg sustava, ali build verzija igre jest i za napraviti ju potrebno je imati ispravan operacijski sustav.

3.2 Specifikacije razgovornog agenta

U ovome radu korišten je MistralAI i to točnije model mistral-7b-instruct-v0.1.Q2 koji možete preuzeti na poveznici https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.1-GGUF. Ovo je najmanji model, koji ima značajan gubitak kvalitete i ne preporučuje se za većinu namjena, ali je dovoljan za svrhu ovoga rada. Osim toga korišten je llama.cpp okvir za hostanje, konfiguraciju i komunikaciju s modelom. Glavni cilj llama.cpp je omogućiti korištenje istreniranih LLM-ova uz minimalno postavljanje i izvedbu na velikom broju hardvera. Moguće ga je preuzeti na poveznici https://github.com/ggerganov/llama.cpp/releases.

3.3 Programski alati

3.3.1 Unity

Verzija Unity-a korištena u ovome radu je 2021.3.0f1. Upute za instalaciju su sljedeće: otvorite stranicu https://unity3d.com/get-unity/download i pritisnite gumb na kojem piše „Download Unity Hub“. Nakon što je instalacija gotova, otvorite datoteku koju ste skinuli i prođite kroz proces instalacije. Sada možete pokrenuti Unity Hub i instalirati potrebnu verziju editora. Pri prvom pokretanju će vas tražiti da instalirate neku verziju editora. Potrebno je ugasiti taj prozor i otići na poveznicu https://unity3d.com/get-unity/download/archive te pronaći verziju 2022.3.20f1. Pritisnite gumb koji glasi „Unity Hub“ pored točne verzije i ubrzo će se u otvorenom prozoru Unity Huba pokrenuti instalacija editora koja će najprije tražiti od vas dopuštenje. Kada je instalacija gotova možete napraviti novi projekt klikom na gumb „New Project“ ili s diska otvoriti postojeći projekt pomoću gumba „Open“ te navigirati do mape u kojoj se on nalazi.

3.3.2 Rider

Za pisanje i uređivanje koda je korišten Jetbrains Rider, i to verzija 2023.3.3, ali je moguće pisati kod i u drugim programima kao što su npr. Visual Studio Code i Visual Studio. Jetbrains Rider se može instalirati preko poveznice https://www.jetbrains.com/rider/download/ i dostupan je za Windows i Mac OS i Linux sustave. Za mrežnu komponentu igre, spajanje s razgovornim agentom, potreban je .NET framework, a ovaj projekt rađen je specifično s verzijom .NET SDK 8.0.206. Moguće je da su Visual Studio ili Unity već instalirali dobru verziju .NET-a, pa je najprije potrebno provjeriti pokretanjem naredbe „dotnet --version“ u Command Promptu ili Powershellu. Verzija smije biti navedena ili veća. U slučaju da računalo ne prepoznaje naredbu, potrebno je otići na poveznicu https://dotnet.microsoft.com/en-us/download/dotnet/8.0 i odabrati verziju koju želite te pratiti proces instalacije.

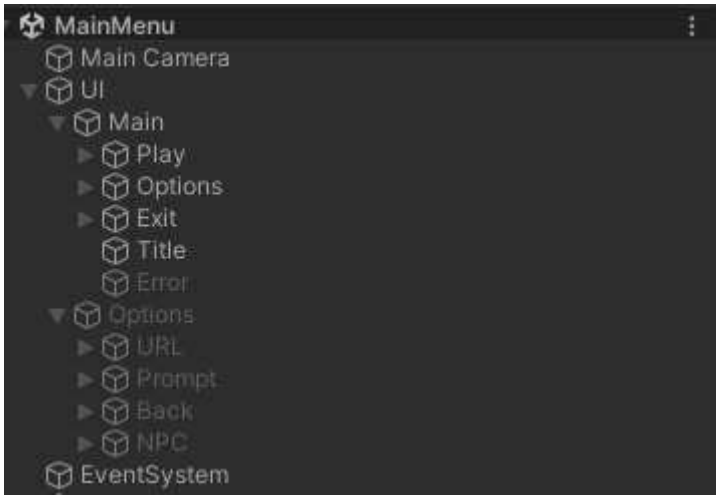
3.4 Glavni izbornik

3.4.1 Korisničko sučelje

Prije pisanja samih funkcionalnosti glavnoga izbornika najprije je potrebno izgraditi korisničko sučelje. Kako je glavni izbornik odvojena komponenta od same igre potrebno ga je staviti u zasebnu scenu. Nova scena se kreira kombinacijom tipki Control i N ili desnim klikom na gumb „+“ u Project prozoru. Ako Project prozor nije prikazan, možete ga otvoriti klikom na opciju koja se nalazi u Window -> General -> Project. Sada kada je nova scena kreirana, možemo krenuti graditi korisničko sučelje. Kao što je već rečeno u opisu modela, koristit ćemo dva različita sučelja, jedno za pokretanje i napuštanje igre te jedno za konfiguraciju razgovornog agenta. Prvo sučelje se sastoji od triju gumba za funkcionalnost izbornika: Play, Options i Exit (Slika X). Gumbe možemo stvoriti tako napravimo desni klik na hijerarhiju scene i odaberemo opciju UI -> Button - TextMeshPro. Ovo će automatski stvoriti objekt tipa Canvas koji funkcionira kao roditelj svim UI elementima. Sljedeće je potrebno ispravno postaviti opcije objekta Canvas. Klikom na objekt prikazuju se njegove značajke u prozoru Inspector. Tamo je potrebno promijeniti značajku Render Mode na Screen Space - Camera, za Render Camera odabrati trenutnu kameru u scenu (Main Camera) te za značajku UI Scale Mode odabrati Scale With Screen Size. Nakon toga slijedi uređivanje gumba kojeg smo instancirali. U hijerarhiji treba proširiti element gumba i kliknuti na element Text (TMP) kojem mijenjamo značajku text u ispravan tekst za svaki gumb. Osim toga imamo naslov, koji će biti objekt tipa Text - TextMeshPro i promijenit ćemo mu značajku text u naslov koji želimo, te imamo poruku greške koja je istog tipa. Također, kako želimo da su elementi sučelja uvijek u sredini ekrana morat ćemo im promijeniti rub Canvasa za koji je pričvršćen i to radimo mijenjajući Anchor značajku u Inspectoru. Potrebno je kliknuti ikonu za Anchor i pojavit će se izbornik za biranje ruba. Držanjem tipke Alt i lijevim klikom miša odabiremo željeni rub. Jedina iznimka je poruka greške koju pričvršćujemo za donji desni kut. Drugo sučelje će se sastojati od tekstualnog polja za unos, višelinijskog tekstualnog polja za unos, padajućeg izbornika te jednog gumba. Za drugo sučelja radimo novi Canvas objekt i tamo smještamo elemente. Gumb radimo na identičan način kao i prije. Tekstualno polje za unos možemo stvoriti ako odaberemo UI -> Input – TextMeshPro. Za opcije ContentType i LineType odabiremo postavke Standard te Single Line da bismo napravili jednostavno polje za unos. Višelinijsko polje stvaramo na identičan način, ali za opciju Line Type odabiremo Multi Line Submit. Ova opcija dopušta pisanje više linija i na pritisak tipke Enter, umjesto prelaska u novi red, poziva se OnSubmit funkcija. Padajući izbornik možemo stvoriti ako odaberemo UI -> Dropdown – TextMeshPro. U njemu samo definiramo listu opcija koje želimo kroz parametar Options. Elementi se dodaju u listu klikom na ikonu „+“ ispod liste. Kako i ovdje želimo da svi elementi sučelja uvijek budu u sredini ekrana mijenjamo im rub za koji su pričvršćeni kao i kod prvog sučelja. Za kraj u hijerarhiji odabiremo Canvas objekt koji sadrži drugo sučelje i onemogućujemo ga micanjem kvačice iz inspektora u gornjem lijevom kutu, jer ne želimo da se na početku vidi drugo sučelje.

Slika 1 Prvo sučelje glavnog izbornika

Slika 2 Prvo sučelje s vidljivom porukom greške



Slika 3 Hijerarhija glavnog izbornika

Slika 4 Drugo sučelje glavnog izbornika

3.4.2 Klasa MainMenu

Varijable

Klasa `MainMenu` mora sadržavati referencu na prvo i drugo sučelje glavnog izbornika `mainMenu` i `optionsMenu`, poruku greške `errorMessage`. Sve varijable su tipa `Gameobject`. Funkcije povezujemo s gumbima tako da u Unity Editoru postavku gumba On Click() postavimo na traženu funkciju instancirane klase `MainMenu`.

Funkcija StartGame

Funkcija `StartGame` je zaslužna za pokretanje igre i to jednostavno radi naredbom `SceneManager.LoadScene` kojoj predajemo redni broj scene kojeg smo postavili u

postavkama za build. No prije nego što to može izvršiti provjeravamo je li korisnik definirao konfiguraciju razgovornog agenta. Provjeru obavljamo pomoću `PlayerPrefs.GetString` metode koja dobavlja pohranjene postavke. Ako konfiguracija nije definirana, onda to dojavljujemo korisniku putem funkcije `ShowError`.

```
public void StartGame()
{
    var prompt = PlayerPrefs.GetString("initialPrompt", null);
    if (prompt is null or "")
    {
        StartCoroutine(ShowError());
        return;
    }

    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
```

Funkcija QuitGame

Funkcija `QuitGame` napušta aplikaciju i to pomoću metode `Application.Quit`.

```
public void QuitGame()
{
    Application.Quit();
}
```

Funkcija ShowOptions

Funkcija `ShowOptions` skriva prvo sučelje izbornika `mainMenu` i pokazuje drugo sučelje izbornika `optionsMenu`. To radi tako da nad `mainMenu` poziva funkciju `SetActive` s vrijednosti `false`, a nad `optionsMenu` s vrijednosti `true`.

```
public void ShowOptions()
{
    mainMenu.SetActive(false);
    optionsMenu.SetActive(true);
}
```

Funkcija HideOptions

Funkcija `HideOptions` jednostavno radi obrnutu radnju od `ShowOptions` tj. pokazuje prvo sučelje izbornika `mainMenu` i skriva drugo sučelje izbornika `optionsMenu`.

```
public void HideOptions()
{
    optionsMenu.SetActive(false);
    mainMenu.SetActive(true);
}
```

Funkcija ShowError

Funkcija `ShowError` je korutina (asinkrona funkcija) koja prikazuje poruku greške `errorMessage`, čeka pet sekundi i zatim skriva poruku `errorMessage`.

```
private IEnumerator ShowError()
{
    errorMessage.SetActive(true);
    yield return new WaitForSeconds(5f);
    errorMessage.SetActive(false);
}
```

3.4.3 Klasa OptionsMenu

Varijable

Klasa `OptionsMenu` mora sadržavati referencu na dva polja za unos teksta `urlInput` i `promptInput` tipa `TMP_InputField` i padajući izbornik `dropdownMenu` tipa `TMP_Dropdown`.

Funkcija Start

Funkcija `Start` je funkcija koja se poziva na početku životnog ciklusa svih klasa koje nasljeduju `MonoBehaviour` klasu. U slučaju ove klase na početku želimo inicijalizirati vrijednosti svih naših polja na one vrijednosti koje su pohranjene u postavkama aplikacije. Kao i kod klase `MainMenu` ovo obavljamo pomoću funkcija `PlayerPrefs.GetString` za tekstualna polja za unos i `PlayerPrefs.GetInt` za padajući izbornik. Ako vrijednosti ne postoje postavljamo ih na prazan tekst u slučaju polja za unos teksta i na broj nula u slučaju odabrane stavke padajućeg izbornika. Također, za postavljanje funkcionalnosti promjena vrijednosti nad našim poljima pozivamo funkciju `InitFields`.

```
private void Start()
{
    InitFields();

    var savedUrl = PlayerPrefs.GetString("chatbotUrl", null);
    var savedPrompt = PlayerPrefs.GetString("initialPrompt", null);
    var savedNpc = PlayerPrefs.GetInt("selectedNpc", 0);

    if (savedUrl != null)
        urlInput.text = savedUrl;
    if (savedPrompt != null)
        promptInput.text = savedPrompt;

    npcDropdown.value = savedNpc;
}
```

Funkcija InitFields

Funkcija `InitFields` je zadužena za postavljanje događaja kada se promijeni vrijednost polja za unos i padajućeg izbornika. Ovo obavljamo tako da na `onSubmit` i `onValueChanged` događaje naših polja i padajućeg izbornika spremamo vrijednosti u postavke aplikacije pomoću klase `PlayerPrefs`, ali ovoga puta `SetString` i `SetInt` metodama. Također, kod polja za unos konfiguracije dodajemo validator teksta da zaustavimo unos nedozvoljenih vrijednosti.

```
private void InitFields()
{
    urlInput.onSubmit.AddListener((value) =>
    {
        var valueToSave = value.Trim().NullIfEmpty();

        if (valueToSave is null)
        {
            urlInput.text = "";
            PlayerPrefs.DeleteKey("chatbotUrl");
        }
        else
        {
            PlayerPrefs.SetString("chatbotUrl", valueToSave);
        }

        PlayerPrefs.Save();
    });

    promptInput.onValidateInput = ValidateInput;
    promptInput.onSubmit.AddListener((value) =>
    {
        var valueToSave = value.Trim().NullIfEmpty();

        if (valueToSave is null)
        {
            promptInput.text = "";
            PlayerPrefs.DeleteKey("initialPrompt");
        }
        else
        {
            PlayerPrefs.SetString("initialPrompt", valueToSave);
        }

        PlayerPrefs.Save();
    });

    npcDropdown.onValueChanged.AddListener((value) =>
    {
        PlayerPrefs.SetInt("selectedNpc", value);
        PlayerPrefs.Save();
    });
}
```

Funkcija ValidateInput

Funkcija `ValidateInput` dodana je da validira polje za unos konfiguracije tako da ako je unesen znak za novi red on se ignorira i ne unosi u polje. Ovo smo dodali zbog greške koja se javlja kod polja za unos u Unity pogonskome sustavu. Naime, kada postavimo parametar Line Type na Multi Line Submit, tipka Enter bi se trebala služiti kao okidač za događaj promjene polja i vrijednost se treba ignorirati. Iz nekog razloga to nije slučaj pa je dodana validacija.

```
private char ValidateInput(string text, int charIndex, char addedChar)
{
    return addedChar == '\n' ? '\0' : addedChar;
}
```

3.5 Izbornik pauze

3.5.1 Korisničko sučelje

Korisničko sučelje izbornika pauze je vrlo jednostavno i sastoji se samo od dvaju gumba Resume i Exit. Gumb Resume je zaslužan za nastavljanje igre, a gumb Exit vraća korisnika na glavni izbornik. Izgled sučelja vidljiv je na slici ispod.

Slika 5 Izbornika za pauziranje

3.5.2 Klasa PauseMenu

Varijable

Klasa `PauseMenu` mora sadržavati referencu na sučelje izbornika `pauseMenu`, referencu na igrača `player` i `boolean` varijablu `_showUI` koja označava je treba li ili ne prikazati izbornik.

Funkcija Start

U funkciji `Start` postavljamo varijablu `_showUI` na `false` i zatim skrivamo `pauseMenu` sučelje za svaki slučaj ako je iz nekog razloga prikazano, putem metode `SetActive`.

```
void Start()
{
    _showUI = false;
    pauseMenu.SetActive(_showUI);
}
```

Funkcija Update

Funkcija `Update` je još jedna funkcija koju mogu implementirati nasljednici klase `MonoBehavior` i poziva se za svaki izračunati okvir (*engl. frame*). U njemu provjeravamo da li

je korisnik pritisnuo tipku Escape i ako je onda mijenjamo varijablu `_showUI` te skrivamo ili prikazujemo sučelje `pauseMenu` ovisno o njenoj vrijednosti. Ako smo pauzirali igru, onda i onemogućavamo skriptu koja pokreće našeg igrača te postavljamo kursor miša vidljivim tako da korisnik može imati interakciju samo s izbornikom kada je igra pauzirana. U suprotnom radimo obratne radnje od navedenih.

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape)) {
        _showUI = !_showUI;
        pauseMenu.SetActive(_showUI);
        player.GetComponent<Player>().enabled = !_showUI;

        if (_showUI) {
            Cursor.lockState = CursorLockMode.None;
            Cursor.visible = true;
        }
        else {
            Cursor.lockState = CursorLockMode.Locked;
            Cursor.visible = false;
        }
    }
}
```

Funkcija ResumeGame

Funkcija `ResumeGame` prekida pauzu i omogućava skriptu koja pokreće našeg igrača te postavlja kursor miša da je nevidljiv.

```
public void ResumeGame() {
    player.GetComponent<Player>().enabled = true;
    pauseMenu.SetActive(false);
    _showUI = false;
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
}
```

Funkcija QuitGame

Funkcija `QuitGame` je zaslužna za vraćanje korisnika na glavni izbornik i to radi naredbom `SceneManager.LoadScene` kojoj predajemo redni broj scene glavnoga izbornika kojeg smo postavili u postavkama za build.

```
public void QuitGame() {
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
}
```

3.6 Tradicionalni NPC

3.6.1 Korisničko sučelje

Za interakciju imamo jednostavno korisničko sučelje u kojem prikazujemo tekst interakcije i ikonu tipke na čiji pritisak pokrećemo interakciju. Ikonu definiramo Image komponentom u

Unity Editoru. Osim toga, dodajemo crnu poluprozirnu pozadinu za bolji kontrast. Izgled ovog sučelja vidljiv je na slici Slika 6. Imamo i sučelje pokrenute interakcije koje se sastoji od crne pozadine, Text komponente s imenom NPC-a, Text komponente sa sadržajem kojeg NPC govori i triju gumba koji služe za odabir opcija kod grananja razgovora. Komponenta sa sadržajem je obuhvaćena Scroll View komponentom kojoj su maknute trake za pomicanje, što osigurava da tekst ne izlazi iz granica. Izgled sučelja vidljiv je na slici Slika 7.

Slika 6 Sučelje za pokretanje interakcije

Slika 7 Sučelje pokrenute interakcije

3.6.2 Klasa Npc

Sučelja i apstraktne funkcije

Klasa `Npc` je apstraktna klasa koja definira zajednička ponašanja za tradicionalne NPC-eve i NPC-eve s razgovornim agentom. Kao što je rečeno u opisu modela, želimo da se interakcija

sa svim vrstama NPC-eva pokreće na ujednačen način, a same detalje interakcije ostavljamo klasama koje nasljeduju ovu klasu. Kako bismo natjerali da svi koji nasljeđuju `Npc` implementiraju funkcionalnost interakcije i prekidanja interakcije, te funkcionalnosti ćemo izdvojiti u sučelje (*engl. Interface*) `IInteractable` koje deklarira funkcije `Interact` i `EndInteraction`. Funkcije `Interact` i `EndInteraction` moraju implementirati sve klase koje implementiraju sučelje. Stavljamo da klasa `Npc` implementira sučelje, ali kako ne želimo da ona implementira te metode već njeni nasljednici samo ih deklariramo i stavljamo da su apstraktne. Na ovakav način možemo lako dodavati dodatne funkcionalnosti NPC-evima uz minimalno refaktoriranje. Kod tradicionalnih NPC-eva i NPC-eva s razgovornim agentom ćemo na isti način i iz istih razloga dodati da se pri interakciji pokreće razgovor s njima. To radimo tako da svaki od njih nasljeđuje sučelje `ITalkable` koje deklarira metodu `Talk`. Metoda `Talk` će se zvati u metodi `Interact`. U klasama ćemo implementirati metodu Talk i definirati kako razgovaramo s pojedinom vrstom NPC-a. Ovo nismo dodali u baznu klasu `Npc` jer možda nećemo htjeti da svaki NPC može pričati s nama. Npr. ako imamo NPC-a koji samo prodaje stvari.

Varijable

Klasa `Npc` mora sadržavati referencu na sučelje koje prikazuje mogućnost interakcije `_interactInterface`, konstantu `INTERACT_DISTANCE` koja označava maksimalnu udaljenost od NPC-a gdje korisnik može s njime stupiti u interakciju, `Transform` komponentu igrača `_playerTransform`, `SkinnedMeshRenderer` komponentu NPC-a `_meshRenderer` koja je odgovorna za prikazivanje 3D modela te `boolean` varijablu `IsInteracting` koja označava da li su korisnik i NPC trenutno u interakciji ili ne.

Funkcija Start

U funkciji `Start` pronalazimo igrača u sceni pomoću funkcije `FindGameObjectWithTag` i inicijaliziramo varijablu `_playerTransform` na igračev `transform` te funkcijom `GetComponentInChildren` dobavljamo `SkinnedMeshRenderer` komponentu objekta na kojem se skripta nalazi i varijablu `_meshRenderer` inicijaliziramo na tu vrijednost.

```
private void Start()
{
    _playerTransform = GameObject.FindGameObjectWithTag("Player").transform;
    _meshRenderer = GetComponentInChildren<SkinnedMeshRenderer>();
}
```

Funkcija Update

U funkciji `Update` radimo provjeru da li se igrač nalazi na dovoljno blizu NPC-u i ako se NPC nalazi u igračevom vidokrugu. Provjeru radimo pomoću metode `IsPlayerWithinDistance`. Nakon te provjere radimo i provjeru da li je korisnik već u interakciji provjerom varijable `IsInteracting`. Ako je već u interakciji onda sakrivamo sučelje za pokretanje interakcije. Dodatno, ako su uvjeti zadovoljeni i igrač pritisne tipku E na tipkovnici, pokrećemo interakciju igrača i NPC-a.

```
private void Update()
{
    if (Input.GetKeyDown(KeyCode.E) && IsPlayerWithinDistance())
    {
        Interact();
    }

    if (_interactInterface.gameObject.activeSelf &&
!IsPlayerWithinDistance())
        _interactInterface.gameObject.SetActive(false);
    else if (!_interactInterface.gameObject.activeSelf &&
IsPlayerWithinDistance())
        _interactInterface.gameObject.SetActive(true);

    if (IsInteracting)
        _interactInterface.gameObject.SetActive(false);
}
```

Funkcija IsPlayerWithinDistance

Funkcija `IsPlayerWithinDistance` provjerava da li se igrač nalazi na udaljenosti od NPC-a manjoj od konstante `INTERACT_DISTANCE` i ako je NPC trenutno vidljiv na kameri. Udaljenost dobivamo usporedbom Transform komponenti igrača i NPC-a. Vidljivost na kameri provjeravamo metodom `isVisible` od `_meshRenderer` parametra.

```
private bool IsPlayerWithinDistance()
{
    return Vector3.Distance(transform.position, _playerTransform.position) <
INTERACT_DISTANCE
           && _meshRenderer.isVisible;
}
```

3.6.3 Klasa StandardNpc

Varijable

Klasa `StandardNpc` mora sadržavati referencu na sučelje koje prikazuje razgovor `dialogueInterface`, listu razgovora `npcDialogue`, kopiju liste razgovora `_npcDialogue` koju postavljamo u `Awake` ili `Start` funkciji, komponentu sučelja tipa `TextMeshProUGUI` koje koja sadrži ime NPC-a `npcName`, kontrolera razgovora `dialogueController` tipa `DialogueController` koji je odgovoran za progresiju razgovora, varijablu `_currentNode` tipa `DialogueNode` koja čuva trenutan čvor dijaloga, `boolean` varijablu `_endInteraction` koja označava da li želimo završiti interakciju te referencu na igrača `player`.

Funkcija Talk

Funkcija `Talk` se poziva u funkciji `Interact` tj. svakoga puta kada želimo imati neku interakciju s NPC-em i zaslužna je za pokretanje razgovora, napredak razgovora i završetak razgovora. Najprije provjeravamo da li je signal za kraj razgovora `_endInteraction` postavljen na istinitu vrijednost i ako je onda ga postavljamo na neistinitu vrijednost te pozivamo funkciju `EndInteraction` koja će završiti razgovor. Sljedeća provjera koju radimo je da li smo obavili sve moguće razgovore s NPC-em provjerom duljine liste razgovora `_npcDialogue`. Ako je lista prazna onda zabranjujemo pokušaj interakcije. Tek nakon ovih

provjera slijedi pokretanje ili nastavljanje razgovora. Ako je varijabla trenutnog čvora `_currentNode` jednak `null` onda znači da tek pokrećemo razgovor i onda pozivamo funkciju `StartConversation`. U suprotnom, već razgovaramo s NPC-em i želimo napredovati do sljedećeg dijela razgovora. To radimo pozivanje funkcije `Visit` našeg `dialogueControllera` koja dohvaća sljedeći čvor razgovora.

```
public void Talk()
{
    if (_endInteraction)
    {
        _endInteraction = false;
        EndInteraction();
        return;
    }

    if (_npcDialogue.Count == 0) return;

    if (_currentNode is null && _npcDialogue.Count > 0)
    {
        StartConversation();
    }
    else
    {
        if (_currentNode is SimpleDialogueNode)
            dialogueController.Visit(_currentNode as SimpleDialogueNode);
        else
            dialogueController.Visit(_currentNode as ChoiceDialogueNode);
    }
}
```

Funkcija StartConversation

`StartConversation` je funkcija zaslužna za pokretanje razgovora s NPC-em. Prva stvar koju radi je provjera da li možda sljedeći razgovor u listi razgovora ima neki uvjet koji je sprječavao napredak do sljedećeg razgovora i jesmo li ispunili taj uvjet. Ako smo ispunili taj uvjet, onda se taj razgovor briše iz liste i odmah poslužuje sljedeći. To radi pozivanjem funkcije `CheckDialogueRequirements`. Nakon ove provjere pokrećemo razgovor koji je sljedeći na redu. Varijablu `IsInteracting` postavljamo na istinitu vrijednost kako bismo označili da razgovaramo s NPC-em i varijablu `_currentNode` postavljamo na prvi čvor razgovora kojeg pokrećemo. Zatim komponentu `npcName` postavljamo na ime zapisano u trenutnom dijalogu i onda prikazujemo sučelje razgovora `dialogueInterface`. Standardno onesposobljujemo kontrole igraču i prikazujemo kursor miša. Kontroleru dijaloga `dialogueController` postavljamo funkciju koja će se pozvati nakon svakog posjećivanja nekog čvora na funkciju `GoToNextNode` i zatim posjećujemo prvi čvor njegovom metodom `VisitNode` predajući mu kontroler kao argument.

```
private void StartConversation()
{
    CheckDialogueRequirements();

    IsInteracting = true;
    player.GetComponent<Player>().enabled = false;
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;

    _currentNode = _npcDialogue[0].firstDialogueLine;
    dialogueInterface.SetActive(true);
    npcName.text = _npcDialogue[0].speakerName;

    dialogueController.SetNextFunction(GoToNextNode);
    _currentNode.VisitNode(dialogueController);
}
```

Funkcija EndInteraction

Funkcija `EndInteraction` je zaslužna za završavanje razgovora. Najprije poziva funkciju `CheckDialogueReward` koja provjerava da li ovaj razgovor nagrađuje igrača nekim predmetom i ako daje onda ga dodaje u inventar. Nakon toga radimo provjeru da li za napredak do sljedećeg razgovora trebamo NPC-u dati neki predmet. Ako ne trebamo onda brišemo trenutni razgovor iz liste. U suprotnom provjeravamo da li imamo taj predmet pomoću funkcije `CheckDialogueRequirements`. Varijablu `_currentNode` postavljamo na `null` i varijablu `IsInteracting` na `false` da signaliziramo da vise ne razgovaramo s NPC-em. Igraču zatim vraćamo kontrole i sakrivamo kursor miša. Osim toga sakrivamo i sučelje za pokrenuti razgovor te `npcName` postavljamo na `null`.

```
public override void EndInteraction()
{
    CheckDialogueReward();

    if (_npcDialogue[0].itemToProgress.NullIfEmpty() is null)
        _npcDialogue.RemoveAt(0);
    else
        CheckDialogueRequirements();


    player.GetComponent<Player>().enabled = true;
    IsInteracting = false;
    _currentNode = null;
    dialogueInterface.SetActive(false);
    npcName.text = null;

    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
}
```

Funkcija GoToNextNode

Funkciju `GoToNextNode` je funkcija koja će se pozvati nakon svakog posjećivanja nekog čvora. Kao argumente dobiva sljedeći čvor i `boolean` varijablu koja označuje treba li se automatski preći na sljedeći čvor. Ova varijabla nam treba za čvorove s izborima da se nakon odabira ne

mora pritisnuti tipka za dalje. Ako sljedeći čvor postoji napredujemo razgovor, a inače signaliziramo kraj interakcije.

```
private void GoToNextNode([CanBeNull] DialogueNode nextNode, bool pressNext =
false)
{
    if (nextNode is not null)
    {
        _currentNode = nextNode;
        if (pressNext) Talk();
    }
    else
        _endInteraction = true;
}
```

Funkcija CheckDialogueRequirements

`CheckDialogueRequirements` funkcija provjerava da li sljedeći razgovor u listi ima neki uvjet koji sprječava napredak do sljedećeg razgovora i jesmo li ispunili taj uvjet. Ako smo ispunili taj uvjet tj. ako imamo taj predmet u inventaru, onda se taj razgovor briše iz liste i poslužuje se sljedeći.

```
private void CheckDialogueRequirements()
{
    if (_npcDialogue[0].itemToProgress.NullIfEmpty() is not null
        &&
PlayerInventory.Instance.ItemInInventory(_npcDialogue[0].itemToProgress)
       )
    {
        _npcDialogue.RemoveAt(0);
    }
}
```

Funkcija CheckDialogueReward

`CheckDialogueReward` je funkcija koja provjerava da li trenutni razgovor nagrađuje igrača nekim predmetom tj. da li ima definiran atribut `reward`. Ako je on definiran onda ga dodaje u inventar funkcijom `AddToInventory` instance klase `PlayerInventory`.

```
private void CheckDialogueReward()
{
    if (_npcDialogue[0].reward.NullIfEmpty() is not null)
    {
        PlayerInventory.Instance.AddToInventory(_npcDialogue[0].reward);
    }
}
```

3.6.4 Klasa Dialogue

Klasa `Dialogue` predstavlja instancu jednog razgovora i sastoji se od imena NPC-a, reference na prvi čvor razgovora te predmeta potrebnog za napredak i nagradnog predmeta koji oboje mogu biti nedefinirani. Neobaveznost označujemo dekoratorom `CanBeNull`. Klasa nasljeđuje `ScriptableObject` kako bismo ju mogli mijenjati unutar Unity Editora umjesto kroz kod. Da bismo mogli stvoriti `Dialogue` objekt, na klasu dodajemo dekorator `CreateAssetMenu` kako

bismo dodali `Dialogue` u izbornik za stvaranje novih objekata koji otvaramo na desni klik u Unity Editoru.

```
[CreateAssetMenu(menuName = "Dialogue/New Dialogue Container")]
public class Dialogue: ScriptableObject
{
    public string speakerName;
    public DialogueNode firstDialogueLine;
    [CanBeNull] public string itemToProgress;
    [CanBeNull] public string reward;
}
```

3.6.5 Klasa DialogueNode

Klasa `DialogueNode` predstavlja općeniti čvor dijaloga. Sadrži varijablu `dialogueLine` koja čuva rečenicu koju čvor predstavlja i njenu `get` funkciju. Osim toga deklarira i funkciju `VisitNode` za posjećivanje čvora koja primjenjuje oblikovni obrazac posjetitelja (*engl. Visitor*). Oblikovni obrazac posjetitelja je obrazac dizajna softvera koji odvaja algoritam od strukture objekta. Zbog ovog odvajanja, nove operacije mogu se dodati postojećim objektnim strukturama bez mijenjanja struktura tj. omogućuje dodavanje novih virtualnih funkcija obitelji klasa, bez mijenjanja klasa. U ovoj klasi ćemo implementaciju funkcije ostaviti klasama koje ju nasljeđuju jer je ovo apstraktna klasa koja služi kao temelj svim vrstama čvorova. Kao i za `Dialogue`, dodajemo da nasljeđuje `ScriptableObject` da bismo mogli klasom manipulirati kroz Unity Editor, ali kako je apstraktna klasa i ne smije se instancirati, već samo njeni nasljednici smiju, ne dodajemo dekorator `CreateAssetMenu`.

```
public abstract class DialogueNode : ScriptableObject
{
    [SerializeField]
    [TextArea(5, 10)]
    private string dialogueLine;
    public string DialogueLine => dialogueLine;

    public abstract void VisitNode(IDialogueNodeVisitor visitor);
}
```

3.6.6 Klasa SimpleDialogueNode

Klasa `SimpleDialogueNode` predstavlja jednostavan čvor koji nema nikakvih grananja. Ne razlikuje se puno od `DialogueNode`, osim po tome što ima referencu na sljedeći čvor `nextNode` i njenu `get` funkciju te implementira funkciju `VisitNode` pozivom funkcije `Visit` od posjetitelja nad sobom.

```
[CreateAssetMenu(menuName = "Dialogue/New SimpleDialogueNode")]
public class SimpleDialogueNode : DialogueNode
{
    [SerializeField]
    private DialogueNode nextNode;
    public DialogueNode NextNode => nextNode;

    public override void VisitNode(IDialogueNodeVisitor visitor)
    {
        visitor.Visit(this);
    }
}
```

3.6.7 Klasa ChoiceDialogueNode

Klasa `ChoiceDialogueNode` predstavlja čvor koji sadrži grananja. Sastoji se od liste mogućih odabira `choices`. Svaki odabir u listi je instanca klase `DialogueChoice` koja u sebi sadrži sadržaj čvora i referencu na sljedeći čvor. Implementira funkciju `VisitNode` na identičan način kao i `SimpleDialogueNode`.

```
[Serializable]
public class DialogueChoice
{
    [SerializeField]
    private string choiceText;
    [SerializeField]
    private DialogueNode choiceNode;

    public string ChoiceText => choiceText;
    public DialogueNode ChoiceNode => choiceNode;
}

[CreateAssetMenu(menuName = "Dialogue/New ChoiceDialogueNode")]
public class ChoiceDialogueNode : DialogueNode
{
    [SerializeField]
    private DialogueChoice[] choices;
    public DialogueChoice[] Choices => choices;

    public override void VisitNode(IDialogueNodeVisitor visitor)
    {
        visitor.Visit(this);
    }
}
```

3.6.8 Klasa DialogueController

Varijable

Klasa `DialogueController` mora sadržavati referencu na dio sučelja pokrenutog razgovora koje se odnosi na sadržaj razgovora `textUI`, listu gumba za koji predstavljaju odabire kod grananja `choiceButtons`, `boolean` varijablu `_isTyping` koja sprječava prelazak na sljedeću rečenicu ako trenutna još nije ispisana do kraja te konstante `TypeSpeed` i `MaxTypeTime` koje upravljaju brzinom prikaza teksta.

Funkcija Start

U `Start` funkciji inicijaliziramo varijablu `_isTyping` na `false` i svakom gumbu postavljamo `onClick` događaj na funkciju `SelectChoice` za odabir opcije pri grananju.

```
private void Start()
{
    _isTyping = false;
    for (var index = 0; index < choiceButtons.Length; index++)
    {
        var index1 = index;
        choiceButtons[index].onClick.AddListener(delegate {
SelectChoice(index1); });
    }
}
```

Funkcija Visit

Klasa `DialogueController` će biti naš posjetitelj čvorova i stoga implementira sučelje `IDialogueNodeVisitor` koje deklarira funkciju `Visit`. Funkcija je overloadana (postoji ista funkcija s različitim argumentima) pa stoga imamo jednu za `SimpleDialogueNode` i za `ChoiceDialogueNode`. Implementacije u kontroleru su za obje funkcije iste, ali ne moraju nužno biti, i provjeravaju ako trenutno obrađujemo neki čvor i ako obrađujemo onda ne radimo ništa tj. čekamo da prijašnji poziv završi. U suprotnom počinjemo s obradom čvora i prikazujemo tekst pomoću korutine `DisplayDialogueText`.

```
public void Visit(SimpleDialogueNode node)
{
    if (_isTyping) return;
    StartCoroutine(DisplayDialogueText(node));
}

public void Visit(ChoiceDialogueNode node)
{
    if (_isTyping) return;
    StartCoroutine(DisplayDialogueText(node));
}
```

Funkcija InitializeChoices

Funkcija `InitializeChoices` je vrlo jednostavna i služi za postavljanje tekstova gumba na ponuđene opcije od `ChoiceDialogueNode` čvora kada ga posjećujemo.

```
private void InitializeChoices(ChoiceDialogueNode node)
{
    _choices = node.Choices;

    for (int index = 0; index < node.Choices.Length; index++)
    {
        choiceButtons[index].gameObject.SetActive(true);
        TMP_Text buttonText =
choiceButtons[index].gameObject.GetComponentInChildren<TMP_Text>();
        buttonText.text = node.Choices[index].ChoiceText;
    }
}
```

Funkcija SelectChoice

Funkcija `SelectChoice` se poziva na odabir neke opcije kod čvora s grananjem. Skriva sve opcije kako ih korisnik više ne bi mogao odabrati i pokreće funkciju `_nextFunction` koja mu je proslijeđena od `StandardNpc` klase. `_nextFunction` je okidač za funkciju `GoToNextNode`.

```
private void SelectChoice(int index)
{
    EventSystem.current.SetSelectedGameObject(null);
    foreach (var choiceButton in choiceButtons)
    {
        choiceButton.gameObject.SetActive(false);
    }

    _nextFunction.Invoke(_choices[index].ChoiceNode, true);
}
```

Funkcija DisplayDialogueText

Korutina `DisplayDialogueText` je zadužena za prikaz teksta nekog čvora na sučelju za pokrenuti razgovor. Varijablu `_isTyping` stavlja na `true` kako bi se onemogućio prekid prikazivanja teksta. Nakon toga tekst sučelja postavljamo na tekst novog čvora, no kako smo u asinkronoj funkciji i Unity ne dozvoljava promjene sučelja s dretvi koje nisu glavna dretva, promjena se neće dogoditi. Rješenje ovoga problema nam se nudi u parametru `maxVisibleCharacters` od sučelja `textUI`. Kroz petlju ćemo povećavati broj vidljivih slova i za svako slovo čekati `MaxTypeTime` / `TypeSpeed` sekundi da pojava slova bude fluidna. Nakon što smo ispisali sav tekst radimo promjenu trenutnog čvora, ali bez prelaska na njega.

```
private IEnumerator DisplayDialogueText(DialogueNode node)
{
    _isTyping = true;
    var maxVisibleChars = 0;
    textUI.text = node.DialogueLine;
    textUI.maxVisibleCharacters = maxVisibleChars;

    foreach (char unused in node.DialogueLine)
    {
        maxVisibleChars++;
        textUI.maxVisibleCharacters = maxVisibleChars;
        yield return new WaitForSeconds(MaxTypeTime / TypeSpeed);
    }

    if (node is SimpleDialogueNode)
        _nextFunction.Invoke((node as SimpleDialogueNode).NextNode);
    else if (node is ChoiceDialogueNode)
    {
        InitializeChoices(node as ChoiceDialogueNode);
    }

    _isTyping = false;
}
```

3.7 NPC s razgovornim agentom

3.7.1 Korisničko sučelje

Za korisničko sučelje NPC-a s razgovornim agentom iskoristiti ćemo veći dio sučelja od tradicionalnog NPC-a i uzeti komponentu s imenom NPC-a te sadržajem razgovora. Umjesto opcija u donjem desnome kutu stavljamo višelinijsko polje za unos teksta te gumbe END CONVERSATION za prekinuti interakciju i GIVE ITEM za dati predmet NPC-u. Izgled sučelja vidljiv je na slici ispod.

Slika 8 Korisničko sučelje pokrenute interakcije s razgovornim agentom

3.7.2 Klasa ChatbotNpc

Varijable

Klasa `ChatbotNpc` mora sadržavati referencu na dio sučelja pokrenutog razgovora koje se odnosi na sadržaj razgovora `textUI`, referencu na cjelokupno sučelje `dialogueInterface`, referencu na dio sučelja s imenom NPC-a `npcNameLabel`, ime NPC-a `npcName`, referencu na polje za unos teksta `inputField`, konfiguraciju razgovornog agenta `_initialPrompt`, varijablu `_response` u koju ćemo spremati odgovor, listu `_interactions` u koju ćemo spremati sve interakcije, instancu klase za komunikaciju s agentom `_engine`, `boolean` varijablu `_waitingForResponse` da označimo kada je interakcija u tijeku, referencu na igrača `player`, reference na gumbe za kraj interakcije i davanje predmeta `endConversationButton` i `giveItemButton`, predmet potreban za napredak `itemToProgress`, nagradu `reward` koju NPC daje te konstante `TypeSpeed` i `MaxTypeTime` koje upravljaju brzinom prikaza teksta.

Funkcija Awake

U `Awake` funkciji inicijaliziramo varijablu `_engine` na novu instancu klase `LlamaCppCom` te konfiguraciju agenta `_initialPrompt` na onu pohranjenu u postavkama, `onClick` događaje gumbi `endConversationButton` i `giveItemButton` podešavamo na funkcije `EndInteraction` i `GiveItem` te polju za unos teksta podešavamo `onSubmit` događaj na funkciju `SubmitMessage` koja šalje poruku razgovornome agentu.

```
private void Awake()
{
    _initialPrompt = PlayerPrefs.GetString("initialPrompt", null).Trim();
    _engine = new LlamaCppCom();

    endConversationButton.onClick.AddListener(EndInteraction);
    giveItemButton.onClick.AddListener(GiveItem);
    inputField.onSubmit.AddListener(SubmitMessage);
    inputField.onValidateInput = ValidateInput;
}
```

Funkcija Interact

Kod NPC-eva s razgovornim agentom mi ne upravljamo komunikacijom kao kod standardnog NPC-a, pa stoga nećemo implementirati sučelje `ITalkable`. Posljedično, naša `Interact` funkcija će izgledati nešto drugačije. U njoj ćemo uvijek zvati samo funkciju `StartInteraction` za pokretanje interakcije jer ostalim stvarima kao što je rečeno, ne upravljamo. Dodatno, pokretanje interakcije dopuštamo samo ako igrač nije obavio sve zadatke za NPC-em.

```
public override void Interact()
{
    if (!IsInteracting
        && !PlayerInventory.Instance.ItemInInventory(reward))
        StartInteraction();
}
```

Funkcija SubmitMessage

`SubmitMessage` je funkcija koja se događa kada korisnik unese poruku u polje za unos teksta. Funkcija briše tekst iz polja za unos te sadržaj prijašnjeg odgovora, postavlja varijablu `_waitigForResponse` na `true` da bismo zaustavili daljnje slanje poruka, započinje korutinu `WaitForChatbotResponse` koja animirano prikazuje učitavanje odgovora, dodaje poruku u listu interakcija te šalje poruku metodom `SendMessage`.

```
private void SubmitMessage(string message)
{
    inputField.text = "";
    textUI.text = "";
    _waitingForResponse = true;
    StartCoroutine(WaitForChatbotResponse());
    AddInteraction("Traveler", message);
    SendMessage();
}
```

Funkcija SendMessage

U `SendMessage` funkciji šaljemo poruku razgovornome agentu. Najprije postavljamo funkciju `OnResponseFinished`, koja se događa kada smo dobili odgovor od agenta, da spremi dobiveni odgovor te prekine čekanje i time pokrene ispis odgovora. Nakon toga kroz `for` petlju sve dosadašnje interakcije i novu interakciju spajamo u jedan tekst odvojen proizvoljnim separatorom i zatim šaljemo poruku funkcijom `Communicate` od varijable `_engine`.

```
private void SendMessage()
{
    string interaction = _initialPrompt;

    _engine.OnResponseFinished = (response) => {
        _response = response.Trim();;
        _waitingForResponse = false;
        AddInteraction(npcName, response);
    };

    foreach (var intc in _interactions)
    {
        interaction += "\n\n::::" + intc.Item1 + ":\n" + intc.Item2.Trim();
    }

    interaction += "\n\n::::" + npcName + ":\n";

    Task.Run(() => _engine.Communicate(
        interaction,
        2048,
        new List<string> { "\n::::" }
    ));
}
```

Funkcija EndInteraction

Funkcija `EndInteraction` je slična kao i kod standardnog NPC-a, gdje sakriva sučelje pokrenute interakcije i vraća kontrole igraču.

```
public override void EndInteraction()
{
    IsInteracting = false;
    npcNameLabel.text = null;
    inputField.gameObject.SetActive(false);
    dialogueInterface.SetActive(false);
    endConversationButton.gameObject.SetActive(false);

    player.GetComponent<Player>().enabled = true;
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
}
```

Funkcija StartInteraction

`StartInteraction` funkcija jednostavno pokreće interakciju tako da sakrije sučelje za pokrenuti interakciju, onemogući kontrole igrača te prikaže dijelove sučelja koji su potrebni za interakciju kao što su polje za unos i gumb za završiti interakciju. Dodatno, ako korisnik u

inventaru ima predmet potreban za napredak zadatka, prikazati će se i gumb za davanje predmeta.

```
private void StartInteraction()
{
    IsInteracting = true;
    textUI.text = null;

    inputField.gameObject.SetActive(true);
    dialogueInterface.SetActive(true);
    endConversationButton.gameObject.SetActive(true);
    npcNameLabel.text = npcName;

    if (itemToProgress is not null
        && PlayerInventory.Instance.ItemInInventory(itemToProgress))
    {
        giveItemButton.gameObject.SetActive(true);
    }
    else
    {
        giveItemButton.gameObject.SetActive(false);
    }

    player.GetComponent<Player>().enabled = false;
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
}
```

Funkcija WaitForChatbotResponse

Korutina `WaitForChatbotResponse` je zaslužna za animiranje stanja čekanja odgovora. Animacija funkcionira tako da sadržaj odgovora svakih pola sekunde kružno alternira između jedne, dvije i tri točke. Ova animacija traje sve dok se ne dobije odgovor od razgovornog agenta i signal za čekanje `_waitigForResponse` postavi na `false`. Nakon toga pokrećemo novu korutinu `DisplayDialogueText` koja prikazuje tekst odgovora i funkcionira identični kao i kod standardnog NPC-a.

```
private IEnumerator WaitForChatbotResponse()
{
    inputField.enabled = false;
    textUI.maxVisibleCharacters = 3;
    textUI.text = "...";
    int visibleCharacters = 0;

    while (_waitingForResponse)
    {
        if (visibleCharacters == 3)
            visibleCharacters = 0;
        else
            visibleCharacters++;

        textUI.maxVisibleCharacters = visibleCharacters;
        yield return new WaitForSeconds(0.5f);
    }

    StartCoroutine(DisplayDialogueText());
}
```

Funkcija GiveItem

Funkcija `GiveItem` poziva se na klik gumba `giveItemButton` i zadužena je za davanje predmeta potrebnog za napredak NPC-u. Najprije radimo provjeru da li je uopće potreban predmet za napredak i ako nije prekidamo funkciju. Nakon toga provjeravamo da li se predmet nalazi u inventaru i ako se nalazi sakrivamo gumb i šaljemo poruku razgovornome agentu da mu dajemo predmet metodom `SendMessage`. Tek poslije slanja poruke stavljamo da nam više nije potreban predmet za napredak resetiranjem varijable `itemToProgress`. Ova varijabla nam je potrebna zato što nam omogućava da spriječimo korisnika od upisivanja poruke da NPC-u daje predmet i dobije nagradu.

```
public void GiveItem()
{
    if (itemToProgress is null) return;

    if (PlayerInventory.Instance.ItemInInventory(itemToProgress))
    {
        giveItemButton.gameObject.SetActive(false);
        _waitingForResponse = true;
        StartCoroutine(WaitForChatbotResponse());
        AddInteraction("Traveler", "The player gave you the item you
needed.");
        SendMessage();
        itemToProgress = null;
    }
}
```

Funkcija CheckIfReceivedReward

U `CheckIfReceivedReward` funkciji analizom odgovora razgovornog agenta provjeravamo da li nam je dao nagradu koju trebamo. Provjeru radimo pomoću metode `checkForReward` klase `RewardLine`.

```
private void CheckIfReceivedReward(String message)
{
    if (RewardLine.checkForReward(message) && itemToProgress is null)
    {
        PlayerInventory.Instance.AddToInventory(reward);
    }
}
```

3.7.3 Klasa LlamaCppCom

Varijable

Klasa `LlamaCppCom` je naše sučelje za komunikaciju s razgovornim agentom koji se nalazi na nekome poslužitelju. Klasa mora sadržavati web adresu na kojem je poslužen razgovorni agent `_endpoint`, instancu klase `HttpClient` koja sluzi za mrežnu komunikaciju u .NET okruženju te funkciju `OnResponseFinished` koja će se izvesti kada smo uspješno dobili odgovor od razgovornog agenta.

Funkcija Communicate

Funkcija `Communicate` je jedina funkcija klase `LlamaCppCom` i sluzi za slanje i primanje poruke od razgovornog agenta. Od ulaznih argumenata radi instancu `Payload` klase koja definira strukturu poruke prema razgovornome agentu i zatim tu instancu serijalizira i od toga gradi novi HTTP zahtjev. Nakon toga šalje zahtjev asinkronom metodom `Http.SendAsync` i čeka odgovor. Ako komunikacija nije bila uspješna bacamo iznimku, a u suprotnom slučaju otvaramo novi `stream` i iz njega čitamo dijelove odgovora dio po dio. Odgovori će biti tipa Response kojeg smo mi definirali. Nakon sto smo skupili sve dijelove odgovora prekidamo komunikaciju i pozivamo funkciju `OnResponseFinished` koja će predati klasi `ChatbotNpc` odgovor da se on može prikazati.

```
public async void Communicate(string prompt, int nPredict, List<string>
stopSeqs)
{
    var payload = new Payload
    {
        prompt = prompt,
        n_predict = nPredict,
        stop = stopSeqs,
        stream = true
    };

    var request = new HttpRequestMessage(HttpMethod.Post, _endpoint)
    {
        Content = new StringContent(
            JsonUtility.ToJson(payload),
            Encoding.UTF8,
            "application/json"
        )
    };

    using var response = await Http.SendAsync(request,
HttpCompletionOption.ResponseHeadersRead);

    if (!response.IsSuccessStatusCode)
    {
        throw new Exception("Chat-bot did not respond: response.StatusCode: "
+ response.StatusCode);
    }

    await using var stream = await response.Content.ReadAsStreamAsync();
    using var reader = new StreamReader(stream);
    var content = "";

    while (!reader.EndOfStream)
    {
        var line = await reader.ReadLineAsync();

        if (string.IsNullOrWhiteSpace(line) || !line.StartsWith("data: "))
continue;

        var data = JsonUtility.FromJson<Response>(line[6..]);

        if (data is not null && data.stop is false)
        {
            content += data.content;
        }
    }

    OnResponseFinished?.Invoke(content);
}
```

3.7.4 Klasa Payload

Klasa `Payload` predstavlja strukturu podatka koje šaljemo razgovornome agentu. Sastoji se od sadržaja poruke `prompt`, maksimalne duljine jedne poruke koju agent šalje nazad `n_predict`. liste zaustavnih znakova `stop` te `boolean` parametra `stream` koji omogućuje ili onemogućuje agentu generiranje i slanje izlaza u manjim, inkrementalnim dijelovima. Također, kako ove podatke šaljemo putem mreže, moramo označiti klasu da je serijalizabilna i koristiti samo serijalizabilne tipove podataka.

```
[Serializable]
public class Payload
{
    public string prompt;
    public int n_predict;
    public List<string> stop;
    public bool stream;
}
```

3.7.5 Klasa Response

Klasa `Response` predstavlja strukturu podatka koje razgovorni agent šalje aplikaciji. Sastoji se od sadržaja poruke `content`, parametra `slot_id` koji govori o kojem dijelu sadržaja se radi ako smo dopustili inkrementalno slanje sadržaja, `boolean` parametra `multimodal` koji označava da li sadržaj može biti nekog drugog tipa osim teksta i oznake da li je ovo zadnji inkrement poruke `stop`.

```
[Serializable]
public class Response
{
    public string content;
    public bool multimodal;
    public int slot_id;
    public bool stop;
}
```

3.7.6 Klasa RewardLine

Klasa `RewardLine` služi kao spremište dopuštenih rečenica koje će igraču dati nagradu u igri. Sadrži statičnu listu dopuštenih rečenica `rewardLines` te funkciju `checkForReward` koja prima odgovor od razgovornog agenta i provjerava da li se jedna od rečenica iz liste nalazi unutar odgovora. Ako se nalazi onda igraču dajemo nagradu, a u suprotnom naravno ne radimo ništa.

```
public class RewardLine
{
    private static String[] rewardLines = {…};

    public static bool checkForReward(String line)
    {
        foreach (var rewardLine in rewardLines)
        {
            if (line.Contains(rewardLine)) return true;
        }
        return false;
    }
}
```

3.7.7 Klasa SetChatbotNpc

Klasa `SetChatbotNpc` je zadužena za postavljanje koji će NPC koristiti razgovornog agenta na temelju odabrane opcije u glavnome izborniku. Sadrži samo jednu varijablu i to je lista svih NPC-eva u sceni. U funkciji Start dobavlja indeks odabranog NPC-a pomoću metode `PlayerPrefs.GetInt` i zatim kroz `for` petlju pronalazi tog NPC-a i aktivira mu skriptu `ChatbotNpc`. Ostalim NPC-evima kompletno brišemo tu komponentu, jer iako je deaktiviramo, referenca postoji i pokušati će se izvršiti.

```
public class SetChatbotNpc : MonoBehaviour
{
    [SerializeField] private List<GameObject> npcList;
    void Start()
    {
        var selectedNpc = PlayerPrefs.GetInt("selectedNpc", 0);

        for (int index = 0; index < npcList.Count; index++)
        {
            if (index == selectedNpc)
            {
                npcList[selectedNpc].GetComponent<ChatbotNpc>().enabled =
true;
                npcList[selectedNpc].GetComponent<StandardNpc>().enabled =
false;
            }
            else
            {
                Destroy(npcList[index].GetComponent<ChatbotNpc>());
            }
        }
    }
}
```

3.8 Upravljanje predmetima

3.8.1 Korisničko sučelje

Igraču želimo dati informaciju da je neki predmet dodan u inventar pa stoga radimo korisničko sučelje koje prikazuje notifikaciju. Sučelje se sastoji od Text komponente za sadržaj notifikacije i Image komponente za pozadinu. Sve zajedno pričvršćujemo za gornji lijevi kut i translatiramo po horizontalnoj osi u lijevo dok više nije vidljivo. Natrag ćemo sve translatirati kada budemo trebali prikazat notifikaciju. Izgled sučelja vidljiv je na slici ispod.

Slika 9 Korisničko sučelje notifikacije

3.8.2 Klasa PlayerInventory

Klasa `PlayerInventory` služi kao inventar za predmete. U sebi sadrži listu predmeta koji su predstavljeni `stringom`. Također, kako koristimo Singleton obrazac, klasa u sebi sadrži instancu same sebe `Instance` koju u funkciji `Awake` postavlja na sebe. Klasa definira dvije funkcije `AddToInventory` za dodavanje predmeta u listu i `ItemInInventory` koja provjerava da li se predmet nalazi u listi. Kada dodajemo predmet pokazujemo to korisniku notifikacijom pozivom funkcije `DisplayNotification` klase `NotificationManager`. Osim toga radimo i provjeru jesmo li dodali predmet koji nam treba da završimo igru i ako je on u inventaru pozivamo funkciju `ShowEndGameUI` klase `EndGame`.

```
public class PlayerInventory: MonoBehaviour
{
    private readonly List<string> _inventory = new();
    public static PlayerInventory Instance;

    private void Awake()
    {
        Instance = this;
    }

    public void AddToInventory(string item)
    {
        _inventory.Add(item);

        if (item == "Sword")
        {
            EndGame.Instance.ShowEndGameUI();
        }
        else
        {
            NotificationManager.Instance.DisplayNotification("Received " +
item);
        }

        if (ItemInInventory("Pastries")
            && ItemInInventory("Meat")
            && ItemInInventory("Ale")
            && !ItemInInventory("Party supplies"))
        {
            _inventory.Add("Party supplies");
        }

    }

    public bool ItemInInventory(string item)
    {
        return _inventory.Contains(item);
    }
}
```

3.8.3 Klasa PickUpItem

Varijable

Klasa `PickUpItem` mora sadržavati referencu na sučelje koje prikazuje mogućnost sakupljana predmeta `interactInterface`, konstantu `INTERACT_DISTANCE` za minimalnu udaljenost gdje je moguće sakupljanje, ime predmeta `itemName` te ako je za predmet potrebno šuljanje (varijabla `requiresStealth`) definiramo i sučelje `stealthInterface`. Osim toga imamo i referencu na igračevu `Transform` komponentu te predmetov `MeshRenderer` kao i za NPC-eve.

Funkcija Start

U funkciji `Start` pronalazimo igrača u sceni i postavljamo `_playerTransform` na njegovu `Transform` komponentu. Također, inicijaliziramo `_meshRenderer` pomoću metode `GetComponentInChildren`.

```
private void Start()
{
    _playerTransform = GameObject.FindGameObjectWithTag("Player").transform;
    _meshRenderer = GetComponentInChildren<MeshRenderer>();
}
```

Funkcija Update

Funkcija `Update` slična je kao i kod NPC-eva. Sakrivamo ili prikazujemo sučelje za interakciju ovisno je li igrač dovoljno blizu predmetu i da li se predmet nalazi u igračevom vidokrugu. Ako se zahtjeva šuljanje, sučelje za interakciju uvijek skrivamo sve dok nismo naučili vještinu šuljanja i prikazujemo samo sučelje `_stealthInterface`. Osim toga, ako su svi uvjeti ispunjeni i igrač pritisne tipku E na tipkovnici, predmet dodajemo u inventar i brišemo fizički predmet iz scene.

```
private void Update()
{
    if (Input.GetKeyDown(KeyCode.E)
        && IsPlayerWithinDistance()
        && (!requiresStealth ||
PlayerInventory.Instance.ItemInInventory("Crab stealth")))
    {
        PlayerInventory.Instance.AddToInventory(itemName);
        Destroy(gameObject);
    }

    if (requiresStealth
        && stealthInterface is not null
        && !PlayerInventory.Instance.ItemInInventory("Crab stealth"))
    {
        if (stealthInterface.gameObject.activeSelf &&
!IsPlayerWithinDistance())
            stealthInterface.gameObject.SetActive(false);
        else if (!stealthInterface.gameObject.activeSelf &&
IsPlayerWithinDistance())
            stealthInterface.gameObject.SetActive(true);
    }
    else
    {
        if (interactInterface.gameObject.activeSelf &&
!IsPlayerWithinDistance())
            interactInterface.gameObject.SetActive(false);
        else if (!interactInterface.gameObject.activeSelf &&
IsPlayerWithinDistance())
            interactInterface.gameObject.SetActive(true);
    }
}
```

3.8.4 Klasa NotificationManager

Varijable

Klasa `NotificationManager` mora sadržavati referencu na `RectTransform` dio Canvas komponente sučelja `notificationRect`, referencu na Text komponentu sučelja notifikacije `notificationText` te instancu na samu sebe jer ponovo koristimo Singleton obrazac. Osim

toga potrebne su nam i varijable s duljinom animacije `duration` te početne i krajnje pozicije notifikacije `startPosition` i `endPosition` tipa `Vector2`.

Funkcija DisplayNotification

Funkcija `DisplayNotification` je javna funkcija koja se poziva iz drugih klasa kada želimo prikazati notifikaciju i ona postavlja tekst notifikacije na onaj zadan argumentom te poziva funkciju `DisplayNotificationAnimation` koja će animirano prikazati notifikaciju.

```
public void DisplayNotification(String text)
{
    notificationText.text = text;
    StartCoroutine(DisplayNotificationAnimation());
}
```

Funkcija DisplayNotificationAnimation

Korutina `DisplayNotificationAnimation` u zadanom vremenu jednakom varijabli `duration` prikazuje notifikaciju. To postiže pomicanjem cijele Canvas komponente putem parametra `anchoredPosition` komponente `notificationRect` kroz `while` petlju. Da bismo osigurali fluidnu animaciju, u petlji računamo novu poziciju pomoću linearne interpolacije. Nakon što je animacija gotova, poziciju notifikacije fiksiramo na `endPosition` da bismo osigurali da je notifikaciju na krajnjoj poziciji. Na kraju čekamo pet sekundi da korisnik može vidjeti notifikaciju i onda pozivamo korutinu `HideNotificationAnimation` koja sakriva notifikaciju.

```
private IEnumerator DisplayNotificationAnimation()
{
    notificationRect.anchoredPosition = startPosition;

    float elapsedTime = 0f;

    while (elapsedTime < duration)
    {
        Vector2 currentPosition = Vector2.Lerp(startPosition, endPosition,
elapsedTime / duration);
        notificationRect.anchoredPosition = currentPosition;
        elapsedTime += Time.deltaTime;
        yield return null;
    }

    notificationRect.anchoredPosition = endPosition;
    yield return new WaitForSeconds(5f);
    StartCoroutine(HideNotificationAnimation());
}
```

Funkcija HideNotificationAnimation

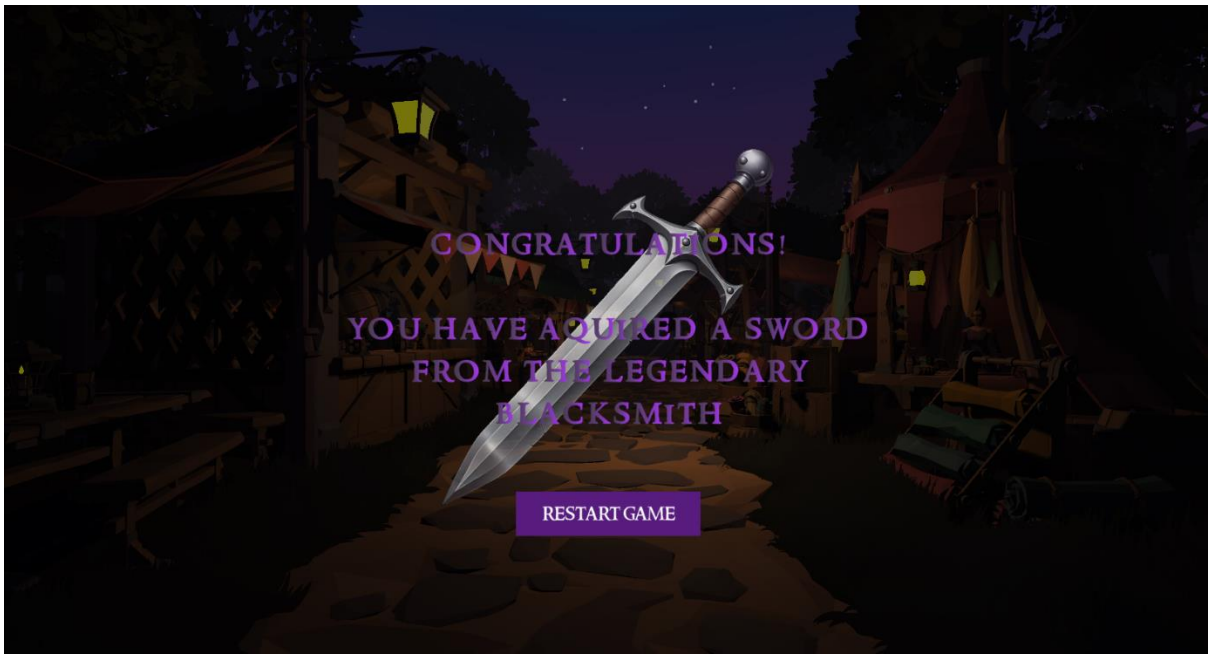
Korutina `HideNotificationAnimation` sakriva notifikaciju i funkcionira na identičan način kao i funkcija `DisplayNotificationAnimation` samo u obrnutome smjeru tj. animira poziciju s krajnje na početnu.

```
private IEnumerator HideNotificationAnimation()
{
    float elapsedTime = 0f;

    while (elapsedTime < duration)
    {
        Vector2 currentPosition = Vector2.Lerp(endPosition, startPosition,
elapsedTime / duration);
        notificationRect.anchoredPosition = currentPosition;
        elapsedTime += Time.deltaTime;
        yield return null;
    }

    notificationRect.anchoredPosition = startPosition;
}
```

3.9 Resetiranje igre

3.9.1 Korisničko sučelje

Kada igrači završi zadnji zadatak prikazati će mu se sučelje za kraj igre. Ovo sučelje se sastoji od pozadinske slike tipa Image, komponente Text s tekstom koji govori da je igra završila te gumba za resetiranje igre. Izgled sučelja vidljiv je na slici ispod. Sučelje ćemo skalirati na nulu da nije vidljivo pomoću scaleFactor atributa komponente CanvasScaler kako bismo mogli animirati njegovo pojavljivanje.



Slika 10 Korisničko sučelje kraja igre

3.9.2 Klasa EndGame

Varijable

Klasa `EndGame` mora sadržavati referencu na `CanvasScaler` komponentu `canvasScaler`, na crnu, poluprozirnu pozadinu `background`, na gumb za resetiranje igre `restartGameButton`, na igrača `player` te referencu na roditelja svih ostalih korisničkih sučelja `userInterface`. Ova klasa također prati Singleton obrazac pa sadrži instancu sebe.

Funkcija Awake

U funkciji `Awake` inicijaliziramo instancu, postavljamo faktor skaliranja na nulu da sakrijemo i sučelje te deaktiviramo pozadinu. Gumbu `restartGameButton` postavljamo `onClick` događaj na funkciju `RestartGame`.

```
private void Awake()
{
    Instance = this;
    canvasScaler.scaleFactor = 0f;
    background.SetActive(false);
    restartGameButton.onClick.AddListener(RestartGame);
}
```

Funkcija ShowEndGameUI

Funkcija `ShowEndGameUI` je javna funkcija koja se poziva iz drugih klasa kada želimo završiti igru i prikazati sučelje kraja igre. Sve što radi je poziva funkciju `WaitThenEndGame` koja će animirano prikazati sučelje.

```
public void ShowEndGameUI()
{
    StartCoroutine(WaitThenEndGame());
}
```

Funkcija WaitThenEndGame

Korutina `WaitThenEndGame` čeka jednu sekundi prije nego što deaktivira sva ostala sučelja u sceni, deaktivira kontrole igrača i poziva korutinu `ScaleGUI` koja će animirano skalirati sučelje kraja igre.

```
private IEnumerator WaitThenEndGame()
{
    yield return new WaitForSeconds(1f);

    userInterface.SetActive(false);
    player.GetComponent<Player>().enabled = false;
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
    StartCoroutine(ScaleGUI());
    endGameAudio.Play();
}
```

Funkcija ScaleGUI

Korutina `ScaleGUI` je zaslužena za skaliranje sučelja. Na početku prikazuje crnu, poluprozirnu pozadinu preko cijeloga ekrana i zatim kreće s animiranjem faktora skaliranja pomoću petlje.

```
private IEnumerator ScaleGUI()
{
    background.SetActive(true);
    while (canvasScaler.scaleFactor < 1)
    {
        canvasScaler.scaleFactor += 0.1f;
        yield return new WaitForSeconds(0.05f);
    }
}
```

Funkcija RestartGame

Funkcija `RestartGame` poziva se na klik `restartGameButton` gumba i zadužena je za resetiranje igre. Pomoću funkcije `GetActiveScene` klase `SceneManager` dobivamo referencu na trenutnu scenu i zatim funkcijom `LoadScene` ponovno učitavamo scenu.

```
private void RestartGame()
{
    Scene currentScene = SceneManager.GetActiveScene();
    SceneManager.LoadScene(currentScene.name);
}
```

4 Rezultati i diskusija

Implementirano rješenje ispunjava sve funkcionalne zahtjeve koji su bili zadani u formalnome modelu. Korisnik u glavnome izborniku može odabrati hoće li pokrenuti igru, ugasiti igru ili konfigurirati razgovornog agenta. U samoj igri implementirane su interakcije s tradicionalnim NPC-evima, što smo postigli koristeći stablaste strukture podataka koje smo implementirali pomoću Unity Scriptable Object komponenti, te interakcije s NPC-evima koji koriste razgovorne agente u pozadini, tako da smo dodali polje za unos poruke koja se šalje servisu na kojem se nalazi razgovorni agent. Kod obje vrste NPC-a smo imali isti način pokretanja interakcije pa smo ga izdvojili u jednu zajedničku nadklasu NPC koju su onda naslijedili. Omogućili smo i davanje te primanje predmeta od obje vrste NPC-eva. Kod tradicionalnih smo proširili stabla da sadrže predmete potrebne za nastavak prolaska po stablu te nagradne predmete na kraju stabla. Kod NPC-eva koji koriste razgovorne agente smo davanje predmeta omogućili pritiskom na gumb GIVE ITEM koji poziva funkciju koja šalje poruku razgovornom agentu da smo mu dali predmet. Dobivanje nagradnih predmeta smo napravili kroz analizu odgovora koje nam šalje razgovorni agent. Za sve te predmete nam je trebao i inventar pa smo ga dodali u obliku Singleton klase. Na dodavanje predmeta smo implementirali i pokazivanje obavijesti kroz klasu NotificationManager koja je također bila Singleton klasa. Osim toga prošli smo kako za sve te interakcije možemo napraviti korisnička sučelja i spojiti ih s traženim funkcionalnostima. Tijekom igre smo dodali mogućnost pauziranja igre, a na završetku igre, kada je igrač obavio sve zadatke, smo dodali i mogućnost resetiranja igre. Kompletno rješenje nalazi se na sljedećoj poveznici: https://gitlab.com/luka-cicak/diplomski-rad-luka-cicak.

U nastavku slijedi primjer korištenja aplikacije:

1. Korisnik pokreće program i dolazi na glavni izbornik. Pritišće gumb EXIT i izlazi iz aplikacije.

2. Korisnik pokreće program i dolazi na glavni izbornik. Pritišće gumb PLAY i dobiva poruku da ne može pokrenuti igru bez da upiše konfiguraciju.

3. Korisnik pritišće gumb OPTIONS gdje definira adresu na kojoj se nalazi razgovorni agent, konfiguraciju razgovornog agenta i odabire NPC-a kojeg želi zamijeniti razgovornim agentom. Pritiskom na gumb BACK vraća se na početni ekran i pritišće PLAY kako bi pokrenuo igru.

4. Korisnik je ušao u igru i dobiva obavijest kako mu je zadatak dobiti mač od NPC-a koji igra ulogu kovača.

5. Korisnik prilazi kovaču i priča s njim. Kovač mu govori kako mu korisnik treba donijeti materijale za mač i da se treba raspitati za njih. Korisnik je slobodan razgovarati s bilo kojim NPC-em.

6. Korisnik rješava zadatka od drugih NPC-eva i od njih dobiva nagrade potrebne za rješavanje drugih zadataka.

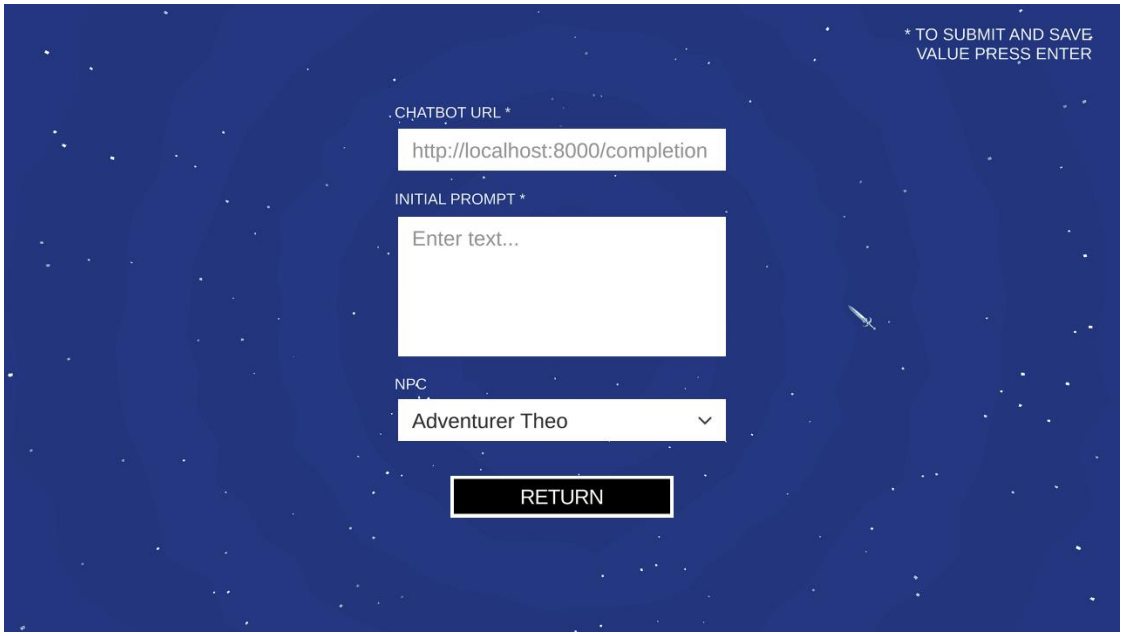
7. Korisnik pritišće tipku Escape i pauzira igru. U izborniku pauze mijenja brzinu prikaza teksta i pritiskom gumba RESUME se vraća u igru.

8. Korisnik nailazi na NPC-a koji je zamijenjen razgovornim agentom i s njime komunicira ne bi li riješio njegov zadatak.

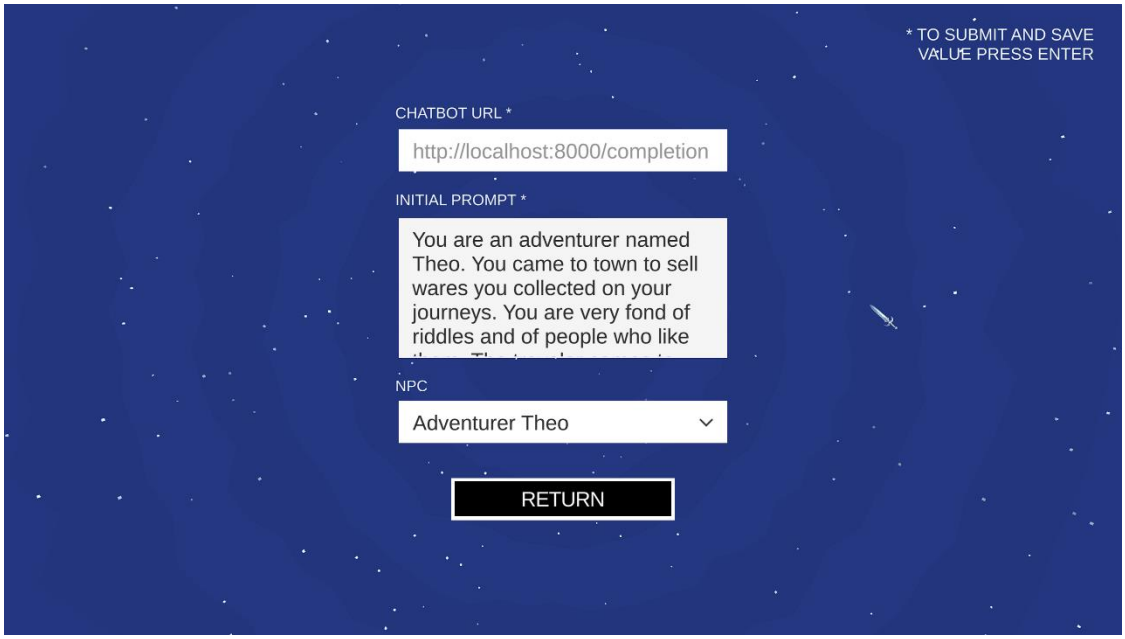
9. Korisnik rješava sve zadatke i vraća se kovaču s materijalima. Kovač korisniku daje mač i time igra dolazi kraju i prikazuje se prikladno sučelje i opcija za resetirati igru.

10. Korisnik resetira igru ili se vraća na glavni izbornik i izlazi iz aplikacije.

Slika 11 Pokretanje aplikacije



Slika 12 Odlazak na izbornik opcija



Slika 13 Popunjavanje opcija

Slika 14 Pokretanje igre i početna obavijest

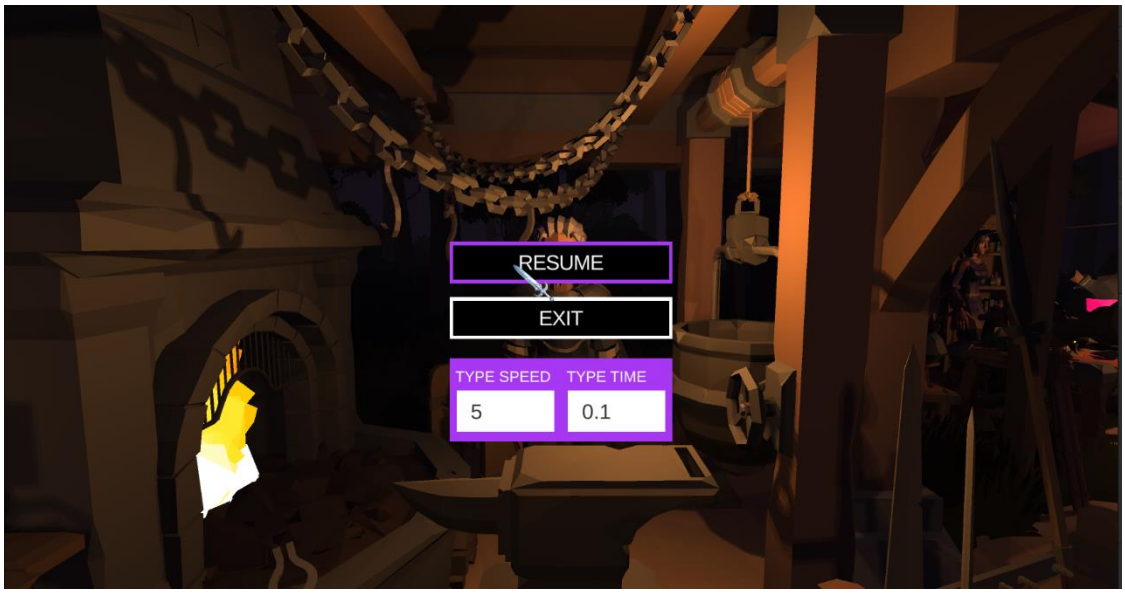
Slika 15 Razgovor s NPC-em

Slika 16 Dobivanje nagrade od NPC-a

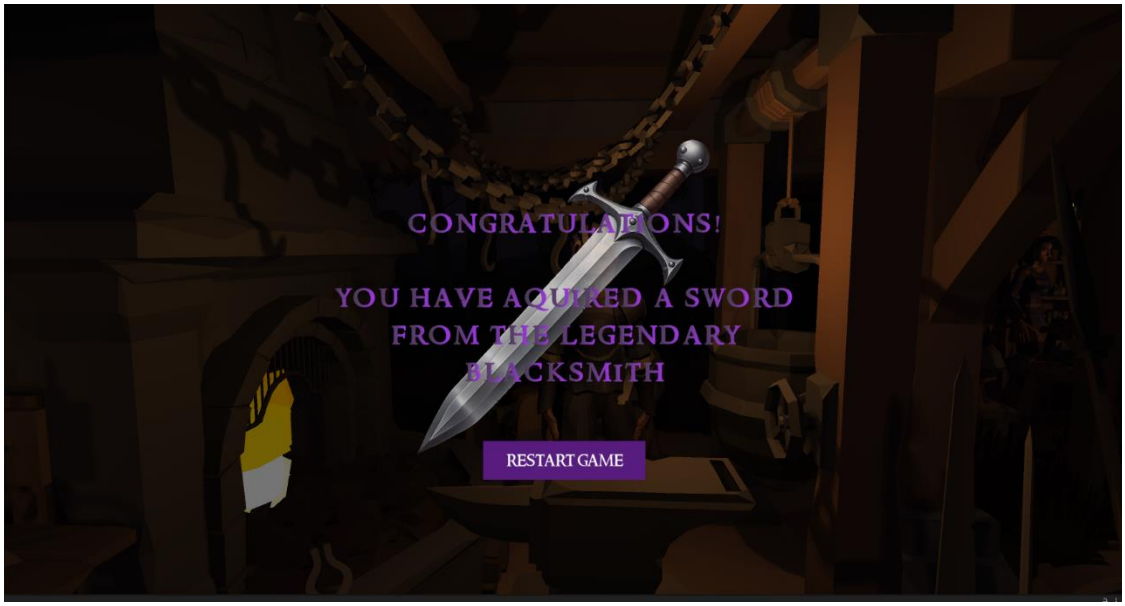
Slika 17 Slanje poruke NPC-u s razgovornim agentom

Slika 18 Čekanje odgovora od NPC-a s razgovornim agentom

Slika 19 Dobivanje odgovora od NPC-a s razgovornim agentom



Slika 20 Pauziranje igre



Slika 21 Kraj igre

Zaključak

Integracija razgovornih agenata u virtualna okruženja, predstavlja značajan napredak u stvaranju upečatljivih i što više imerzivnih iskustava za igrače. Iskorištavanjem umjetne inteligencije i obrade prirodnog jezika, razgovorni agenti mogu pružiti dinamičnije i responzivnije interakcije koje doprinose impresivnijem i personaliziranijem iskustvu igranja. Međutim, integracija razgovornih agenata također predstavlja svojevrsne izazove, uključujući potrebu za visokokvalitetnim razumijevanjem prirodnog jezika, povećanje računalnog opterećenja i nužnost održavanja zanimljivih i kontekstualno prikladnih razgovora. Rješavanje navedenih izazova zahtijeva daljnje istraživanje i razvoj kako bi ova tehnologija bilo primjenjiva i održiva u stvarnom korištenju. Ovaj rad istražio je metodologije i tehnologije uključene u razvoj i implementaciju razgovornog agenta unutar Unity pogonskog sustava, ističući potencijalne prednosti i izazove povezane s ovim pristupom. U radu je prikazano kako napraviti tradicionalnog NPC-a pomoću stablastih struktura podataka, NPC-a koji u pozadini koristi razgovornog agenta te kako izgraditi sučelje igre i povezati ga s funkcionalnostima. Zaključno, korištenje razgovornih agenata u virtualnim okruženjima otvara nove mogućnosti za obogaćivanje iskustava igrača i stvaranje dinamičnijih i interaktivnijih svjetova. Kako se tehnologija nastavlja razvijati, uloga razgovornih agenata u virtualnim okruženjima će se vjerojatno proširiti, nudeći uzbudljive mogućnosti za programere i igrače.

Literatura

[1] Unity Technologies, *Unity Real-Time Development Platform*, Unity. Poveznica: https://unity.com/; pristupljeno 10. lipnja 2024.

[2] Unity Technologies, *Unity download archive*, Unity. Poveznica: https://unity3d.com/get-unity/download/archive; pristupljeno 10. lipnja 2024.

[3] Brodkin J., *How Unity3D Became a Game-Development Beast*, Dice (2013, lipanj). Poveznica: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/; pristupljeno 10. lipnja 2024.

[4] Batchelor J., *Unity dropping major updates in favour of date-based model*, gamesindustry.biz (2016, prosinac). Poveznica: https://www.gamesindustry.biz/articles/2016-12-14-unity-dropping-major-updates-in-favour-of-date-based-model; pristupljeno 10. lipnja 2024.

[5] Nelson Ayers, *Demystifying NPCs: The Complete Guide to Non-Player Characters in Gaming*, 33rdsquare (2023, listopad). Poveznica: https://www.33rdsquare.com/what-is-a-npc-in-gaming/; pristupljeno 12. lipnja 2024.

[6] Jon-Paul Dyson, *The History of NPCs*, Museum Of Play (2024, siječanj). Poveznica: https://www.museumofplay.org/blog/the-history-of-npcs/; pristupljeno 12. lipnja 2024.

[7] OpenAI, *Better language models and their implications*, OpenAI (2019, veljača). Poveznica: https://web.archive.org/web/20201219132206/https://openai.com/blog/better-language-models/; pristupljeno 18. lipnja 2024.

[8] Samuel R. Bowman, *Eight Things to Know about Large Language Models*, Cornel University (2023, travanj). Poveznica: https://arxiv.org/abs/2304.00612; pristupljeno 18. lipnja 2024.

[9] Lucy O'Brien, *How Ubisoft's New Generative AI Prototype Changes the Narrative for NPCs*, Ubisoft (2024, ožujak). Poveznica: https://news.ubisoft.com/en-us/article/5qXdxhshJBXoanFZApdG3L/how-ubisofts-new-generative-ai-prototype-changes-the-narrative-for-npcs; pristupljeno 18. lipnja 2024.

[10] Yasmina Benkhoui, *Spotlight: Convai Reinvents Non-Playable Character Interactions*, Nvidia Developer (2024, siječanj). Poveznica: https://developer.nvidia.com/blog/spotlight-convai-reinvents-non-playable-character-interactions/; pristupljeno 18. lipnja 2024.

[11] Convai, *Embodied AI Characters For Virtual Worlds*, Convai (2024, lipanj). Poveznica: https://convai.com/; pristupljeno 18. lipnja 2024.

Sažetak

Računalne igre su oduvijek težile što većem realizmu, s ciljem stvaranja upečatljivih i što više imerzivnih iskustava za igrače kako bi se oni što više uživili u svijet igre. Tijekom godina poboljšao se vizualni realizam u igrama, brzina njihovih izvođenja te sam obujam igara. Jedna od komponenata igara koja nije napredovala su likovi u videoigrama koji nisu igrač ili skraćeno NPC-evi. No, to se može promijeniti tehnologijom razgovornih agenata koji imaju velik potencijal učiniti interakcije s NPC-evima dinamičnijima i personaliziranijima. Cilj ovoga rada je napraviti pokazni edukativni primjer programa koji integrira razgovorne agente i njima zamjenjuje tradicionalne NPC-eve. Programsko rješenje rađeno je u Unity pogonskome sustavu koristeći C# programski jezik, ali je u radu opisan i formalni model rješenja neovisan o bilo kakvim alatima.

Summary

Computer games have always strived for as much realism as possible, with the aim of creating striking and immersive experiences for players so that they can immerse themselves in the game world as much as possible. Over the years, the visual realism in the games, their performance and the scope of the game itself have reached new heights. One of the components that has not progressed are the non-player characters in videogames, or NPCs for short. But that can all change with the technology of AI chatbots, which has great potential to make interactions with NPCs far more dynamic and personalized. The goal of this work is to create a demonstrative educational example which integrates chatbots and replaces traditional NPCs with them. The software solution was created in the Unity engine using the C# programming language, but the paper also describes a formal solution model independent of any tools.