Govorno sučelje za interaktivnu fikciju

Ribarić, Marijan

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetni diplomski studij informatike

Informacijski i komunikacijski sustavi

Marijan Ribarić

Govorno sučelje za interaktivnu fikciju

Diplomski rad

Mentor: doc. dr. sc. Miran Pobar

Rijeka, prosinac 2020.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 01.06.2020.

Zadatak za diplomski rad

Pristupnik: Marijan Ribarić

Naziv diplomskog rada: Govorno sučelje za interaktivnu fikciju

Naziv diplomskog rada na eng. jeziku: Speech interface for interactive fiction

Sadržaj zadatka:

Opisati pojam i ukratko opisati povijesni razvoj interaktivne fikcije. Opisati princip rada i osnovne dijelove sustava za interaktivnu fikciju. Izraditi sustav za interaktivnu fikciju koji se sastoji od komponente za autore fikcije i komponente za korisnike (igrače). Komponenta za korisnike treba omogućiti interakciju sa sustavom putem govornog sučelja.

Mentor: doc. dr. sc. Miran Pobar

Voditeljica za diplomske radove: izv. prof. dr. sc. Ana Meštrović

Komentor:

Zadatak preuzet: 01.06.2020.

(potpis pristupnika)

Sažetak

U osamdesetim godinama prošlog stoljeća interaktivna fikcija bila je na vrhuncu popularnosti. Međutim, s razvojem novih tehnologija popularnost je pala. Jedan od mogućih načina za vraćanje zainteresiranosti za interaktivnu fikciju je implementiranje mogućnosti korištenja govornih sučelja. Kroz ovaj rad opisan je pojam interaktivne fikcije i način na koji takvi sustavi rade. Također su opisani govorni sustavi i njihove komponente. Cilj diplomskog rada je kreirati sustav za razvoj interaktivne fikcije te omogućiti komunikaciju s sustavom putem govornog sučelja. Sustav je kreiran koristeći programske okvire *Asp.Net Core* i *React*. Govorno sučelje je realizirano kao mobilna Android aplikacija koja koristi Googleov *SpeechRecognizer* i *TextToSpeech* API.

Ključne riječi: interaktivna fikcija, tekstualne avanture, govorno sučelje, Asp.Net Core, React

Sadržaj

1. Uvod

Razvojem računala počeli su se razvijati i načini kako iskoristiti računala za zabavu. Jedan od prvih oblika zabavnog softvera bila je interaktivna fikcija, također poznata i pod nazivom tekstualna avantura. Interaktivna fikcija je vrsta igre i/ili elektroničke literature (Montfort 2005) koja je koristila komandno-linijsko sučelje te tekst kao ulaz i izlaz iz programa. Na taj način program je igraču pričao priču na koju je on svojim akcijama mogao utjecati. Tijekom godina je popularnost interaktivne fikcije padala pa je danas igra mali broj ljudi.

Jedna od mogućih ideja za povećanje popularnosti interaktivne fikcije je govorno sučelje koje bi omogućavalo igraču da igra djela interaktivne fikcije bez tipkanja teksta, koristeći glas kao metodu unosa i sintezu govora kao izlaz.

Cilj ovog rada je opisati što je to interaktivna fikcija, kako je nastala i kako radi te izraditi sustav za razvoj interaktivne fikcije u kojem korisnik može izrađivati svoje igre. Ovaj sustav je realiziran kao web aplikacija. Osim toga, kreirati će se i govorno sučelje koje će omogućiti korisniku govornu interakciju s tim sustavom. Govorno sučelje biti će izvedeno kao mobilna aplikacija.

U nastavku ovog rada biti će opisan i definiran pojam interaktivne fikcije. Ukratko će biti objašnjen način na koji se igraju igre interaktivne fikcije i način na koji se interaktivna fikcija razvijala kroz povijest. Zatim će biti opisane glavne komponente sustava interaktivne fikcije i način na koji su realizirane. Potom će biti opisani postojeći sustavi za razvoj interaktivne fikcije. Također, biti će opisani govorni sustavi i ukratko objašnjene komponente od kojih se sastoji. Nakon toga će biti opisan projektni zadatak (sustav za razvoj interaktivne fikcije i govorno sučelje za njega) koji je kreiran kao dio ovog rada. Biti će opisana arhitektura sustava, funkcionalnosti sustava te način na koji je sustav realiziran. U zaključku će biti dan osvrt na rad te moguća poboljšanja za sustav.

2. Interaktivna fikcija

Postoji više mogućih definicija za pojam interaktivne fikcije. Šira definicija interaktivne fikcije podrazumijeva sve oblike fikcije u kojoj korisnik svojim akcijama utječe na razvoj priče. Ovakva definicija podrazumijeva različitu vrstu dijela poput knjiga u kojima korisnik vlastitim odabirom određuje kojom će granom priča nastaviti (Granade, Brass Lantern 2010). U ovu definiciju bi također pripadale video igre koje omogućuju igraču da svojim akcijama utječe na priču igre. Video igre su dizajnirane na način da igraču omoguće interaktivnost s različitim dijelovima igre, pa zato ne postoje tehničke prepreke za implementaciju na razini priče za razliku od knjiga i filmova kod kojih je to teško realizirati (Lebowitz and Klug 2011).

Međutim, pojam interaktivne fikcije se u većini slučajeva koristi za specifični oblik fikcije s kojom korisnik vrši interakciju putem tekstualnog sučelja. Jedna od mogućih definicija koju predlaže autor Montfort je: „Interaktivna fikcija je računalni program koji prikazuje tekst, prihvaća tekstualni unos te zatim prikazuje dodatni tekst kao odgovor na tekst koji je korisnik upisao.“. Pri tome računalni program mora barem donekle razumjeti unos koji prihvaća te s obzirom na taj unos smisleno reagirati na način da obavi promjene u svijetu koji simulira (Montfort 2005).

Termin tekstualna avantura (engl. *text adventure*) često se koristi zajedno s pojmom interaktivne fikcije. U većem broju literature (pogotovo na internetu) ti pojmovi se koriste kao sinonimi. Međutim, neki autori smatraju da su tekstualne avanture samo jedna podvrsta interaktivne fikcije te postoje i druge vrste interaktivne fikcije koje se ne mogu smatrati ni avanturama ni igrama (Montfort 2005). Također, postoji i razmišljanje da je interaktivna fikcija vrsta medija koja je nastala iz tekstualnih avantura budući da moderna interaktivna fikcija može uključivati grafičke i zvučne elemente, iako je i dalje uglavnom fokusirana na tekst (Roberts n.d.).

2.1. Igranje interaktivne fikcije

Na početku interakcije s programom interaktivne fikcije korisniku se prikaže poruka koja ukratko objašnjava situaciju i okoliš u kojemu se lik s kojim korisnik upravlja trenutno nalazi. Potom program od korisnika traži unos teksta u obliku akcije koju korisnik želi napraviti. Nakon što korisnik unese tekst, taj tekst se procesira te program pokuša pretvoriti unos u neku od mogućih akcija u simuliranom svijetu. Ukoliko je prikladna akcija pronađena korisniku se ispiše novonastalo stanje koje je posljedica rezultata akcije. U suprotnom,

korisniku se ispiše poruka koja ga obavještava da akcija nije pronađena. Ovaj postupak se ponavlja sve dok korisnik ne završi igru (pobjedom ili porazom) ili dok ne prekine interakciju iz nekog drugog razloga (Maher 2006). Ovaj postupak će se u nastavku rada nazivati igranjem interaktivne fikcije.

Novim igračima može biti teže za razumjeti koje akcije su mu dostupne u kojem trenutku. Djela interaktivne fikcije dolaze u mnogo različitih žanrova poput misterija, trilera i djela znanstvene fantastike pa samim time se razlikuju i akcije koje igrač može obavljati (Granade and Short, A Beginner's Guide to Interactive Fiction 2010). Međutim, većina modernijih djela prihvaća naredbe koje spadaju u slijedeće kategorije (Granade, Brass Lantern 2010):

- Akcija koji se sastoji od jedne riječi – Naredbe poput *look, jump, hide* za koje nije potreban objekt
- Akcija i izravni objekt – Naredbe koje se odvijaju nad određenim objektom. Npr. *take the axe*
- Akcija, izravni objekt i neizravni objekt – Naredbe koje se odvijaju nad izravnim objektom uz korištenje neizravnog objekta. Npr. *cut tree with axe*
- Naredbe drugim likovima – slično prije navedenim naredbama, ali se naredbe vrše nad likovima umjesto objektima. Npr. *give me sword*
- Razgovor s drugim likovima – specifične vrste akcija u kojima se vodi razgovor s drugim likovima. Npr. *ask about location*
- Posebne naredbe – naredbe poput *save* za spremanje i *restart* za ponovno pokretanje

2.2. Povijest interaktivne fikcije

2.2.1. Nastanak interaktivne fikcije

Prvi računalni program koji je imao elemente koji podsjećaju na djela interaktivne fikcije bio je program pod nazivom *Eliza* (Maher 2006). Program je mogao simulirati razgovor o različitim temama ovisno o skripti koju je koristio. Najpoznatija skripta pod nazivom *Doctor* simulirala je razgovor između psihoterapeuta i pacijenta – korisnika koji je koristio sustav (Montfort 2005). *Elizu* je razvio Joseph Weizenbaum na sveučilištu MIT 1966. godine. Program je analizirao rečenicu koju je korisnik unio te je u njoj pretraživao ključne riječi (engl. *keywords*). Kada se takva riječ pronašla program bi stvorio izlazni tekst ovisno o pravilu povezanim s ključnom riječi i kontekstom iz prijašnjim korisnikovih unosa (ako je pravilo

povezano s ključnom riječi ovisilo o kontekstu) (Weizenbaum 1966). Iako je način komunikacije koji je koristila *Eliza* sličan komunikaciji koja se pojavljuje kod sustava interaktivne fikcije, *Eliza* ne provodi simulaciju nekog svijeta pa se zato ne smatra interaktivnom fikcijom. Bez obzira na to *Eliza* je bila bitna za razvoj interaktivne fikcije zato što je bila prvi program koji je koristio model interakcije između korisnika i igre koji se koristi kod interaktivne fikcije (Maher 2006).

Prvom tekstualnom avanturom, kao i prvim djelom interaktivne fikcije smatra se *Adventure* koju je napisao Will Crowther 1975. godine (Montfort 2005). Crowther je napisao igru te ju je učinio dostupnom na ARPANetu kako bi je njegove kćeri mogle igrati. Tematika igre bilo je putovanje špiljom.

Budući da je igra bila dostupna na ARPANetu do nje su često dolazili hakeri te bi nakon što su je odigrali neki od njih pokušali unaprijediti. Na taj način je igra došla i do Don Woodsa koji je, uz dopuštenje Will Crowther-a, igru unaprijedio tako što je popravio greške, proširio igru (između ostalog dodavanjem stvorenja poput trolova i vilenjaka) i pretvorio je u kompletnu igru (Maher 2006).

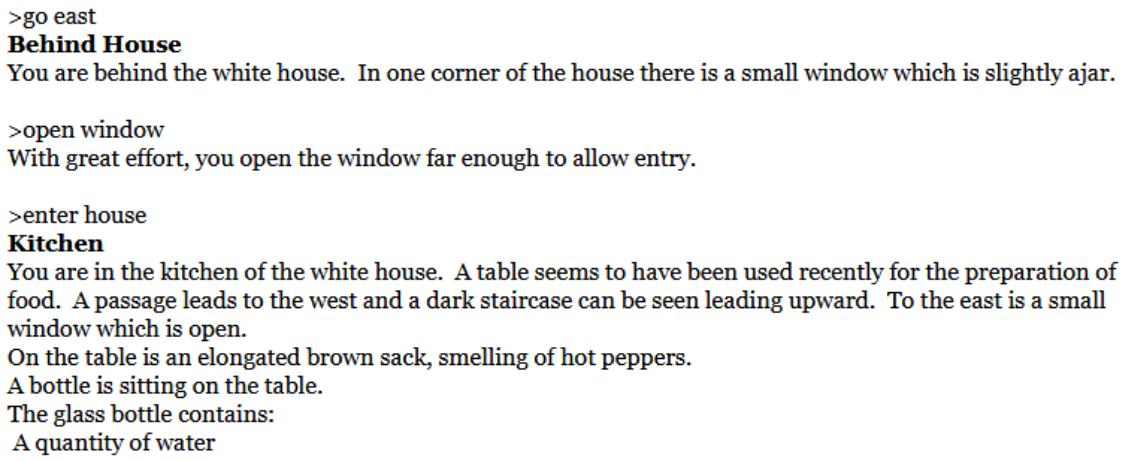
2.2.2. Komercijalni uspjeh

Problem s prije navedenim igrama je to što su se pokretale isključivo na središnjim računalima (engl. *mainframe*). Naime, osobna računala imala su malu procesorsku snagu i veoma ograničenu memoriju pa izvođenje programa interaktivne fikcije nije bilo moguće. Taj problem je riješio Scott Adams koji je napisao igru *Adventureland* (donekle baziranu na igri *Adventure)*. Prema autoru Granade, Adams je za svoju igru napisao interpreter s čime je povećao ponovnu iskoristivost dijelova koda i olakšao portabilnost igara na različite platforme. To je također omogućilo da jedan interpreter pokreće više igara koristeći samo podatke o igri čime se dodatno smanjuje potreba za memorijskim prostorom (Granade, Brass Lantern 2010). Iz istog razloga Adams je smanjio dužinu opisa koji se prikazuju korisniku s nekoliko rečenica na nekoliko riječi i pojednostavio parser (Maher 2006).

Igra *Adventureland* dobro se prodavala te je Adams, zajedno sa svojom suprugom, osnovao tvrtku *Adventure International* koja je tijekom slijedećih nekoliko godina proizvela još dvadesetak igara (Maher 2006).

Vjerojatno najpoznatije djelo interaktivne fikcije je *Zork*. *Zork* su napisali Tim Anderson, Marc Blanc, Bruce Daniels i Dave Lebling 1977. godine na sveučilištu MIT.

Napisan je u MDL programskom jeziku te je imao sofisticiraniji parser koji je prihvaćao kompleksnije naredbe te je program ispisivao mnogo detaljnije opise soba i objekta u usporedbi s igrom *Adventure* (Briceno, et al. 2000). Isječak s početka igre *Zork* koja se izvodi na stranici textadventures.co.uk korištenjem programa *Inform* može se vidjeti na slici 1.


>go east
Behind House
You are behind the white house. In one corner of the house there is a small window which is slightly ajar.

>open window
With great effort, you open the window far enough to allow entry.

>enter house
Kitchen
You are in the kitchen of the white house. A table seems to have been used recently for the preparation of food. A passage leads to the west and a dark staircase can be seen leading upward. To the east is a small window which is open.
On the table is an elongated brown sack, smelling of hot peppers.
A bottle is sitting on the table.
The glass bottle contains:
A quantity of water

Slika 1 – Isječak iz igre Zork, prevedena u jezik Inform, dostupna na http://textadventures.co.uk

Autori igre *Zork* osnovali su 1979. godine tvrtku *Infocom*. Autori Briceno i ostali navode kako cilj tvrtke nije bio izrada interaktivne fikcije već „ozbiljnog“ softvera. Međutim, budući da nisu imali proizvod koji su mogli prodavati odlučili su napraviti verziju igre *Zork* koja se može pokretati na osobnim računalima te nju prodavati. Kako su osobna računala bila mnogo slabija od središnjih računala na kojima se originalni *Zork* pokretao razvili su programski jezik *ZIL* (engl. *Zork Implementation Language*¸ Zork implementacijski jezik) i virtualni stroj *Z-Machine* (Briceno, et al. 2000). *ZIL* je bio specijaliziran za pisanje interaktivne fikcije te je apstrahirao implementacijske detalje niže razine te se s time olakšavalo i ubrzavalo kreiranje novih igara, dok je *Z-Machine* omogućavao pokretanje na različitim konfiguracijama osobnih računala (Maher 2006).

Prema autoru Maher *Infocom* je imao revolucionarni utjecaj na budućnost interaktivne fikcije te je postavio standarde koji se i danas koriste. Prakse s posebnim programskim jezicima za pisanje interaktivne fikcije te interpretiranje datoteka igre umjesto kreiranja izvršnim datoteka, kao što su to radili ZIL i *Z-Machine* zadržale su se i danas u alatima poput *Informa* (Maher 2006).

Osim *Infocoma*, u istom razdoblju djela interaktivne fikcije počele su, između ostalih, objavljivati i slijedeće tvrtke:

- Topologika
- Level 9 Computing
- Melbourne House
- Telarium
- Synapse's Electronic Novels

Popis nekih od djela iz ovog razdoblja te informacije o njihovim autorima, tvrtkama koje su djelo objavile i godine objavljivanja mogu se vidjeti u tablici 1.

Tablica 1 - Značajnija komercijalna djela interaktivne fikcije

Naziv djela	Autor	Tvrtka	Godina
Zork Trilogija	David Lebling, Marc Blank	Infocom	1980, 1981, 1982
Enchanter	David Lebling, Marc Blank	Infocom	1983
Sorcerer	Steve Meretzky	Infocom	1984
Spellbreaker	David Lebling	Infocom	1985
The Hitchhiker's Guide to the Galaxy	Douglas Adams, Steve Meretzky	Infocom	1984
Avon	Jonathan Partington	Topologika	1983
Knight Orc	Pete Austin	Level 9 Computing	1987
Mindwheel	Robert Pinsky	Synapse	1984
Essex	Bill Darrah	Synapse	1985
The Hobbit	Philip Mitchell Veronika Megler	Melbourne House	1982
Amnesia	Thomas M. Disch	Electronic Arts	1987

2.2.3. Pad komercijalne popularnosti

Postoji više razloga za pad popularnosti interaktivne fikcije, no najveći razlog je sigurno rapidni razvoj računalne grafike. Računala su postala sve snažnija, omogućavala su prikaz bolje grafike i bila su jeftinija i dostupna široj javnosti (Briceno, et al. 2000). Međutim, padu popularnosti su pridonosili i ostali faktori poput starenja publike koji su imali sve manje vremena za interaktivnu fikciju te prezasićenost tržišta nekvalitetnim djelima koja su se prodavala po visokoj cijeni (Maher 2006).

2.2.3.1. Kraj Infocoma

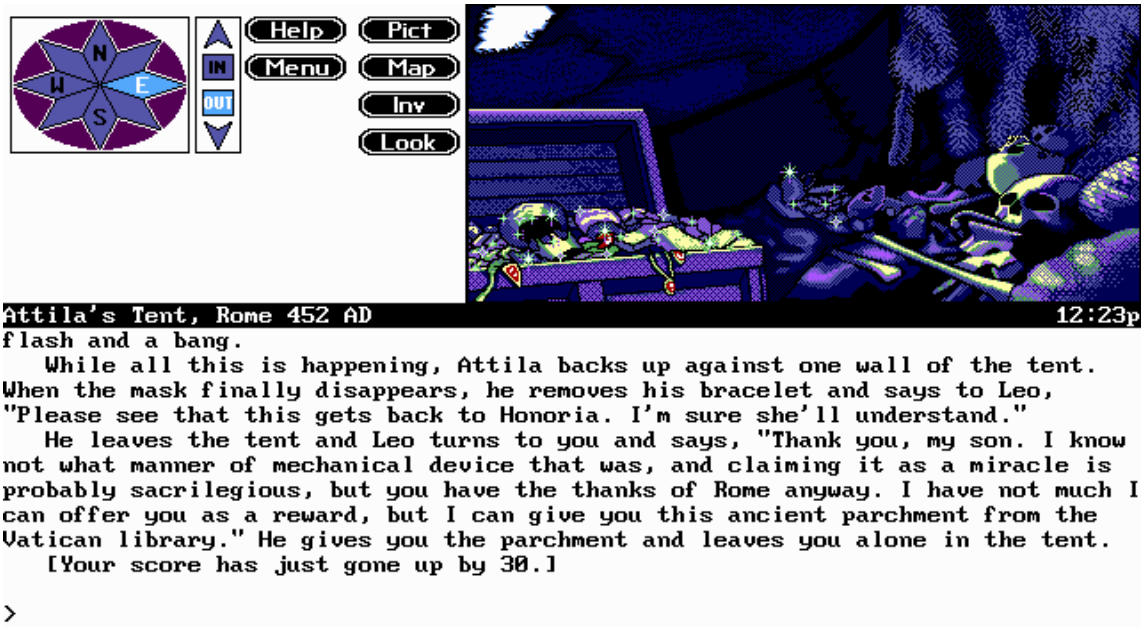
Infocomov pad započeo je kreiranjem softvera za relacijske baze podataka *Cornerstone* 1985. godine. Autori Briceno i ostali navode kako je većina resursa tvrtke bilo uloženo u *Cornerstone*. Reakcije na program su bile podijeljene. Dok su nekim korisnicima svidjela jednostavnost sučelja i jednostavnost izvođenja operacijama nad bazom podataka, drugim korisnicima se nije svidjelo nemogućnost programiranja novih funkcija. Također performanse programa bile su mnogo slabije od sličnih programa od konkurentnih tvrtki (Briceno, et al. 2000).

Zbog duga nastalog neuspjehom *Cornerstonea Infocom* je bio primoran prodati se vanjskom kupcu što je i učinio 1986. godine kada ga je kupio *Activision*. *Infocom* je neko vrijeme nastavio izdavati tekstualne avanture, međutim zbog manjka interesa pokušali su napraviti grafičke igre koje također nisu bile uspješne (Briceno, et al. 2000).

Infocom je nastavio poslovati s gubitkom od 1987. do 1989. godine kada ju je *Activision* zatvorio te prebacio dio zaposlenika u glavnu tvrtku (Briceno, et al. 2000).

2.2.3.2. Ostale tvrtke

Jedna od tvrtki koja je nastavila s djelima interaktivne fikcije bila je *Legend Entertaiment*. Autor Maher tvrdi kako je osnivač tvrtke, Bob Bates, vjerovao da interaktivna fikcija i dalje može biti komercijalni uspjeh ukoliko je popraćena ostalim multimedijskim sadržajima. Njihova igra *Timequest* kombinirala je tekstualni unos s unosom putem miša (Maher 2006). Isječak iz igre *Timequest* koji prikazuje tekstualno sučelje koje je igra koristila, zajedno s gumbima koji su se koristili pritiskom miša, može se vidjeti na slici 2.



Slika 2 - Isječak iz igre Timequest. Izvor: https://korseby.net/spiele/legend/

Autor Maher opisuje kako je sa svakom slijedećom igrom *Legend Entertainment* koristio sve više grafičkih elemenata i miš kao glavnu metodu unosa. Taj trend se nastavio sve dok parser i tekstualni unos nisu bili u potpunosti izbačeni. Ostale tvrtke su također izbacivale tekstualni unos te je to označilo kraj komercijalne interaktivne fikcije (Maher 2006).

2.2.4. Interaktivna fikcija nakon pada popularnosti

Nakon kraja komercijalne popularnosti, interaktivna fikcija ostala je popularna kao hobi. Autori su djela objavljivali besplatno kako bi ih ostatak zajednice mogao igrati. Razvoj igara omogućili su besplatni sustavi za razvoj interaktivne fikcije poput sustava *Inform* i *TADS.*

Ovakvi sustavi, zajedno s slobodom eksperimentiranja zbog nedostatka pritiska za komercijalnim uspjehom, omogućili su kreaciju kompleksnih djela koja su se nastavila stvarati do današnjeg dana (Maher 2006).

3. Komponente sustava za interaktivnu fikciju

Programi interaktivne fikcije sastoje se od dvije ključne komponente: parsera i modela svijeta (Montfort 2005).

3.1. Parser

Parser je dio programa interaktivne fikcije koji analizira korisnikov tekstualni unos napisan na prirodnom jeziku (Montfort 2005). Cilj parsera je pretvoriti korisnikov unos u akciju koju model svijeta može obraditi. Dok su parseri u prvim igrama interaktivne fikcije bili veoma jednostavni i mogli su prihvaćati samo unaprijed definirani skup riječi, većinom u obliku *glagol objekt*, tijekom godina su se unaprjeđivali i prihvaćali više riječi i kompleksnije naredbe. Zbog manjka tehnoloških limita i razvoja tehnika obrade prirodnog jezika (engl. *natural langauge processing, NLP)* današnji parseri mogu razumjeti akcije za koje nisu eksplicitno programirani.

3.1.1. Ispravljanje grešaka u pisanju

Za ispravljanje grešaka u pisanju (tzv. *typo*) koriste se algoritmi za približno podudaranje niza znakova (engl. *approximate string matching*). Mjera koja označava koliko su dva niza znakova različiti zove se udaljenost uređivanja (engl. *edit distance*). Udaljenost se obično mjeri prema broju operacija koje je potrebno napraviti na prvom nizu znakova da se dobije drugi niz znakova. U praksi se najčešće koriste slijedeće operacije (Navarro 2001):

- Umetanje (engl. *insertion*) – umetanje nekog znaka
- Brisanje (engl. *deletion*) – brisanje nekog znaka
- Zamjena (engl. *replacement*) – zamjena dva različita znaka
- Transpozicija (engl. *transposition*) – zamjena dva različita susjedna znaka

Jedna od takvih mjera je Levenshteinova udaljenost. Levenshteinova udaljenost definira se kao minimalni broj umetanja, brisanja ili zamjene znakova kako bi se iz jednog niza znakova dobio drugi (Navarro 2001).

Neke druge mjere udaljenosti uređivanja su (Navarro 2001):

- Hammingova udaljenost (engl. *Hamming distance*)
- Epizodna udaljenost (engl. *Episode distance*)
- Najduža zajednička podsekvenca (engl. *Longest Common Subsequence, LCS*)

Kod interaktivne fikcije ova tehnika se može koristiti kada se uspoređuje očekivani unos (sve dostupne akcije u tom trenutku) s unosom koji je korisnik unio. To smanjuje

frustraciju kod korisnika koja bi se dogodila kad parser ne bi prihvatio unos zbog male greške kod tipkanja. Također, zato što postoji samo ograničeni broj potencijalnih naredbi koje parser prihvaća dovoljno je korisnikov unos uspoređivati samo s tim akcijama, umjesto s cijelim rječnikom. To ubrzava postupak ispravljanja što može biti značajno kod uređaja s slabim performansama.

3.1.2. Zaustavne riječi

Korisnikov unos može često sadržavati više riječi nego što je potrebno parseru da prepozna akciju. To može predstavljati problem jer u tom slučaju moramo definirati što će parser napraviti s dodatnim riječima. Obično se to rješava na način da parser ne prihvati unos.

Zaustavne riječi (engl. *stopwords*) su riječi koje se često pojavljuju u jeziku te nude malu ili nikakvu pomoć pri razumijevanju rečenice (Manning, Schütze i Raghavan 2008). Micanjem tih riječi iz rečenice možemo dobiti smanjeni broj riječi koje trebamo procesirati.

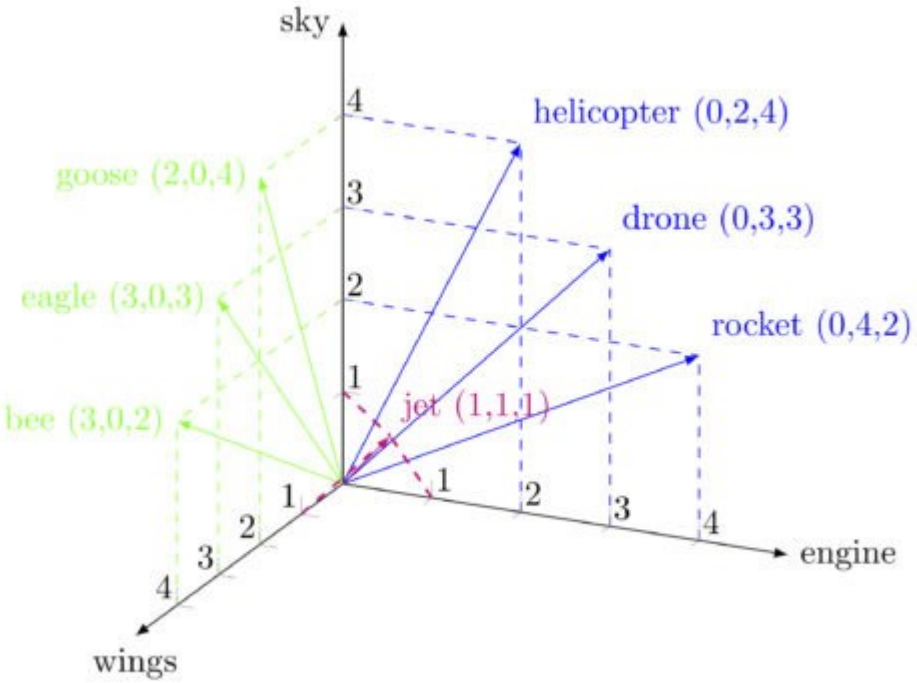
Ovisno o području na kojem se ova tehnika primjenjuje potrebno je koristiti različite liste zaustavnih riječi. Kod interaktivne fikcije standardnom setu zaustavnih riječi korisno je dodati riječi *want*, *using, with*. Na primjer, korisnik može unijeti rečenicu „*I want to cut the tree with axe"* (Želim posjeći drvo s sjekirom.). Nakon micanja zaustavnih riječi dobivamo unos „*cut tree axe"* što se direktno može mapirati na akciju *cut* na izravnom objektu *tree* koristeći neizravni objekt *axe*. Time se omogućuje puno šira mogućnost izražavanja kod korisnika bez dodatnog programiranja za svaku akciju. Neke od čestih zaustavnih riječi u engleskom jeziku mogu se vidjeti na slici 3.

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

Slika 3 - Primjeri zaustavnih riječi u engleskom jeziku. Izvor: https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html

3.1.3. Ugrađivanje riječi

Jedan od načina za reprezentaciju riječi kako bi nad njima mogle izvoditi operacije je reprezentacija u multidimenzijalnom vektorskom prostoru. Ovaj način zapisa kreće od pretpostavke da riječi koje se pojavljuju u sličnom kontekstu imaju slično značenje (Pilehvar and Camacho-Collados 2020). Opisana tehnika naziva se ugrađivanje riječi (engl. *word embeddings*).



Slika 4 - Primjer prikaza riječi u vektorskom prostoru. Izvor: https://corpling.hypotheses.org/495

Na slici 4 može se vidjeti kako su riječi sa sličnim značenjem blizu u vektorskom prostoru.

Kod interaktivne fikcije ova tehnika se može primijeniti za generiranje sinonima kako bi se mogle definirati dodatne riječi koje izazivaju određenu akciju. To se radi tako da se uzme određeni broj riječi koje su najbliže određenoj riječi u vektorskom prostoru. Međutim, ovaj pristup zahtjeva da se određena riječ usporedi sa svim ostalim riječima u vokabularu što zahtjeva vremena. Druga moguća korist od ove tehnike je provjera sličnosti korisnikovog unosa u usporedbi sa svim mogućim unosima koristeći mjeru kosinusove sličnosti (engl. *cosine similarity*). Kosinusova sličnost daje nam kosinus kuta između vektora A i B u vrijednosti između -1 i 1, gdje 1 označava potpuno slične vektore, a -1 potpuno različite vektore (neo4j n.d.). Formula (3.1) po kojoj se računa kosinusova sličnost prikazana je ispod. Simboli A_i i B_i predstavljaju pojedinu komponentu vektora A i B, a simbol *n* predstavlja broj komponenti tih vektora.

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \quad (3.1)$$

Dobivenu vrijednost možemo interpretirati kao sličnost između riječi. Ukoliko je sličnost veća od odabrane granice možemo prihvatiti riječ koju je korisnik upisao te je smatrati sinonimom za riječ koju parser prihvaća.

U ovom diplomskom radu, kao dio projektnog zadatka, koristi se predtrenirani word2vec (Mikolov, et al. 2013) model koji je treniran na dijelu Google News skupa podataka. Model se sastoji od 300-dimenzijalnih vektora za 3 milijuna riječi i fraza. Model je dostupan na slijedećoj poveznici: https://code.google.com/archive/p/word2vec/.

3.2. Model svijeta

Autor Montfort opisuje model svijeta kao dio programa interaktivne fikcije koji „predstavlja fizičko okruženje interaktivne fikcije, kao i stvari u okruženju, uključujući fizičke objekte s kojima se može manipulirati ili dodatno istražiti na bilo koji način te lika kojim upravlja igrač“ (Montfort 2005).

Model svijeta ima nekoliko zadaća:

- Praćenje stanja igre
- Praćenje stanja svakog objekta
- Generiranje akcija koje su korisniku dostupne
- Procesiranje akcija koje korisnik želi izvesti
- Vođenje razgovora s likovima u igri

Glavni koncept koji koristi većina sustava za razvoj interaktivne fikcije su sobe. Sobe predstavljaju prostor u kojemu se mogu nalaziti igrač i ostali rekviziti (poput objekata). Igrač može interaktirati samo s objektima koji se nalaze u istoj sobi u kojoj se nalazi i igrač (Reed 2010). Korisnik se može kretati između soba koristeći strane svijeta. Na primjer, ukoliko igrač želi ići sjeverno može reći *go north* ukoliko je ta navigacija implementirana i dozvoljena. Iako je ovaj način kretanja samo jedan od mogućih, kroz godine je postao standardni način za kretanje među sobama i koristi se u većini igara (Reed 2010).

Unutar sobe nalaze se objekti s kojima igrač može interaktirati. Nad objektima su definirane akcije koje se mogu obavljati s njime (npr. pogledaj, uzmi, uništi), te svojstva koja određuju kako se objekt ponaša u sobi (npr. vidljiv/nevidljiv, otvoren/zatvoren).

Likovi, često zvani NPC (engl. *non-player character*, likovi s kojima ne upravlja igrač) su posebna vrsta objekata s kojima je moguće voditi razgovor. Razgovor s NPC-jevima jedan

je od najkompleksnijih problema u interaktivnoj fikciji. Dok je simuliranje predvidljivih akcija relativno jednostavno, kod simuliranja razgovora treba dobro odabrati koliku slobodu će se dati igraču kako bi iskustvo razgovora bilo što manje frustrirajuće (Maher 2006).

3.2.1. Vođenje razgovora s likovima

U svom članku „*NPC Conversation Systems*“ autorica Short opisuje različite načine komunikacije s likovima (Short 2011). U nastavku su navedeni i objašnjeni neki od popularnijih načina.

3.2.1.1. Komunikacija bez riječi

Kod komunikacije bez riječi likovi u sceni reagiraju na akcije koje igrač vrši nad drugim objektima. Prednosti ovog načina su to što likove nije potrebno posebno programirati već su komentari likova na akcije dio samih akcija nad objektima. Također, igrač se ne mora prilagođavati na nove akcije već koristi akcije s objektima na koje je već naviknut. S druge strane, mana je to što je jedini način komunikacije preko akcija drugih objekata (Short 2011).

3.2.1.2. Pričaj sa

U ovom načinu komunikacije jedina akcija koju korisnik poduzima je početak razgovora s likom, dok je tijek razgovora predefiniran. Drugim riječima, autor određuje sve što će reći korisnik i lik. Na ovaj način ne moramo procesirati korisnički unos, ali samim time korisnik nema mogućnosti da interaktira na način koji on to želi (Short 2011).

3.2.1.3. Izbornik razgovora

Korisniku se prezentira izbornik iz kojega može odabrati što želi reći. Prednost je što autor ima potpunu kontrolu nad time što korisnik može reći, dok je mana to što je ovaj pristup i dalje restriktivan te korisnik možda ne želi odabrati ni jednu od ponuđenih opcija (Short 2011).

3.2.1.4. Pitaj/kaži i teme razgovora

Jedan od često korištenih načina razgovora je pitaj/kaži (engl. *ask/tell*). Ovaj način dopušta korisniku razgovor korištenjem glagola pitaj i kaži te ključne riječi koja označava temu razgovora. Ograničenje ovog sustava je to što igrač može dobiti samo odgovor na svoj upit te nije moguće voditi razgovor kao kod izbornika. Prednost je to što korisnik ima slobodu da pita o bilo čemu (iako neke od tema neće biti isprogramirane) (Short 2011).

Pojednostavljenje ovog pristupa su teme razgovora, gdje korisnik ne treba izreći glagole pitaj/kaži već je dovoljno da parser prepozna neku od ključnih riječi da bi dobio odgovor (Short 2011).

3.2.1.5. Unaprjeđenja sustava za razgovor

Ovakvi sustavi mogu se dodatno proširiti te se mogu kombinirati s ostalima kako bi se dobio napredniji sustav, ukoliko je to potrebno za specifičnu igru i ako sustav u kojem se igra kreira to dozvoljava. Neki od tih unaprjeđenja su (Short 2011):

- Hibrid između izbornika i tema razgovora
- Dodavanje glagola koji označavaju ton razgovora
- Unos poput onog u programima koji automatiziraju razgovor (engl. *chatbot*)

4. Sustavi za razvoj interaktivne fikcije

Sustavi za razvoj interaktivne fikcije su sustavi koji omogućavaju autorima da kreiraju djela interaktivne fikcije bez da trebaju programirati cjelokupni program od početka. Takvi programi nude opcije poput kreiranja soba, objekata i definiranja pravila po kojima će se objekti ponašati.

Ovakvi sustavi počeli su se razvijati zajedno s razvojem interaktivne fikcije. Sustavom za razvoj interaktivne fikcije može se smatrati i ZIL koji je bio korišten za izradu igra *Zork* (Briceno, et al. 2000). Danas se većina igara interaktivne fikcije piše u nekim od sustava za razvoj interaktivne fikcije (Maher 2006).

U ovom poglavlju dati će se kratki pregled slijedećih sustava:

- Inform 7
- TADS 3
- Quest 5

Kako bi se ukratko demonstrirao način na koji rade ovi sustavi, u slijedećim potpoglavljima biti će demonstrirano dodavanje nove sobe i implementiranje navigacije u svakom od alata.

4.1. Inform 7

Inform 7 je alat i programski jezik koji se koristi za kreiranje interaktivne fikcije. Jedna od glavnih značajki programskog jezika *Inform* 7 je to što koristi sintaksu prirodnog jezika (Inform n.d.). Drugim riječima, korisnik upisuje naredbe koje sustav prepoznaje na engleskom jeziku te na taj način kreira igru.

Na primjer, za kreiranje sobe *Forest* korisnik upisuje rečenicu *„Forest is a room.“*. Prilikom prevođenja programskog koda, prevoditelj razumije riječ *is* kao naredbu za kreiranje elemenata u igri. Ovakve rečenice u *Informu* nazivaju se tvrdnje (engl. *assertions*) te se pomoću njih definiraju svi elementi i sva pravila u *Informu* (Reed 2010). Ukoliko želimo dodati navigaciju sjeverno iz sobe *Forest* u sobu *Coutyard* možemo koristiti tvrdnju „*Courtyard is north of Forest“*.

Da bi *Inform* razumio rečenicu mora biti napisana u skladu s pravilima sintakse koja se mogu promaći u dokumentaciji. Riječima autora Reed: „Iako vaša priča izgleda kao prirodni

jezik, Inform ne razumije sav prirodni jezik“ (Reed 2010). Iz ovog razloga *Inform* može biti težak za korištenje novim korisnicima dok se ne naviknu na sintaksu koju *Inform* prihvaća.

4.2. TADS 3

TADS 3 je sustav za kreiranje interaktivne fikcije i programski jezik koji pristupa kreiranju interaktivne fikcije s perspektive programera. Jezik ima sličnu sintaksu kao C++ ili Javascript te daje programerima razinu kontrole koju očekuju od programskog jezika (Roberts n.d.).

Na slici 5 može se vidjeti način na koji se definira soba u sustavu TADS. Riječ *hallway* (plava boja) označava interno ime za sobu koje programski jezik koristi. *Room* predstavlja ključnu riječ pomoću koje se definira soba. Riječ *'Hallway'* (crvena boja) označava ime sobe koje se ispisuje korisniku, dok je rečenica u crvenoj boji ispod opis sobe koji se također ispisuje korisniku. Navigacija između soba se označava korištenjem ključnih riječi za strane svijeta. Na slici su definirana navigacija južno koja vodi u sobu *entryway* i navigacija sjeverno koja vodi u sobu *kitchen*.

```
hallway: Room 'Hallway'
    "This broad, dimly-lit corridor runs north and south. "

    south = entryway
    north = kitchen
;
```

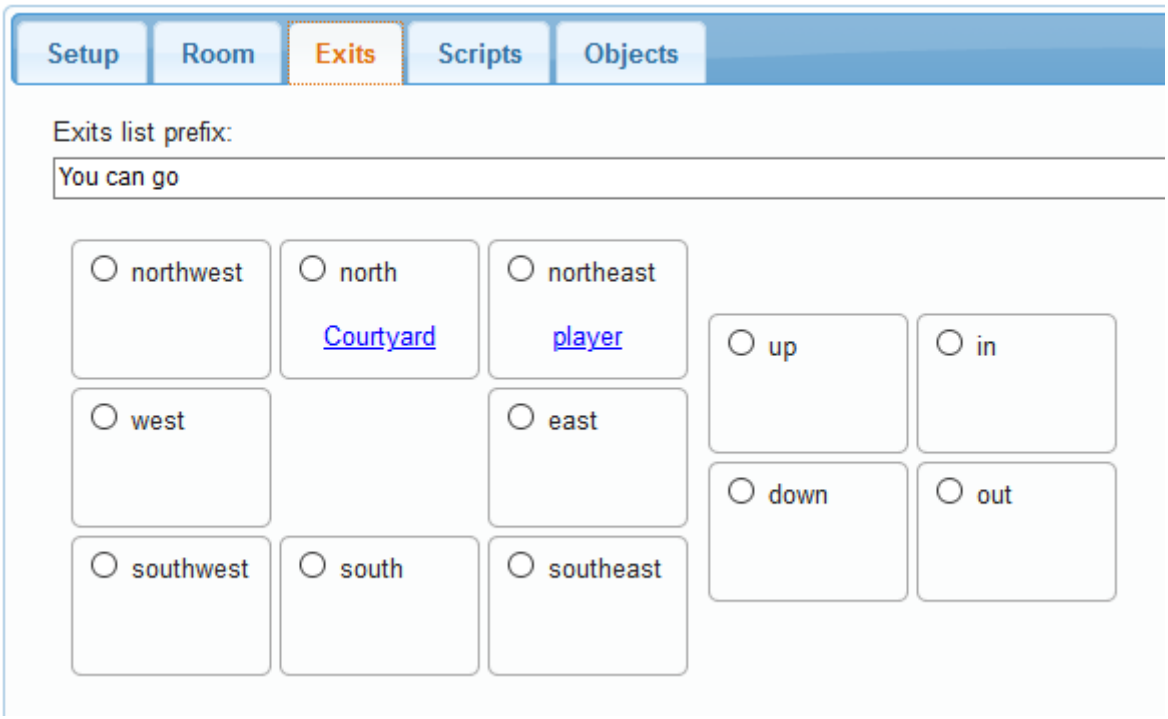
Slika 5 - Isječak programskog koda u jeziku TADS 3

4.3. Quest 5

Quest 5 je besplatni alat otvorenog koda za kreiranje tekstualnih avantura. Jednostavan je za korištenje te je veliki broj funkcionalnosti moguće implementirati bez ikakvog programiranja budući da se oslanja na izbornike. Za stvari koje nije moguće implementirati samo putem izbornika dostupan je jezik za pisanje skripti *ASL* koji je inspiriran jezicima *C++, Javascript i Visual BASIC* (Quest n.d.).

Jedna od prednosti *Questa* je to što se može koristiti u web preglednicima te nije potreban dodatni program (iako postoji verzija i za desktop). *Quest* je jedan od dva sustava koji se koriste na stranici https://textadventures.co.uk.

U sustavu *Quest* soba se kreira pritiskom na gumb. Za dodavanje navigacije prema nekoj drugoj sobi koristi se kartica *Exits* te se u njoj odabere koja će strana svijeta voditi gdje. Izgled ove postavke na web verziji sustava može se vidjeti na slici 6.



Slika 6 - Postavke navigacije u sustavu Quest Izvor: https://textadventures.co.uk

5. Govorna sučelja

Govorno korisničko sučelje (engl. *voice user interface,* VUI) je sučelje koje omogućuje korisniku upravljanje sustavima korištenjem glasa (Rivera 2020). Govorna sučelja sve se češće koriste. Jedan od primjera korištenja govornih sustava su osobni asistenti poput Alexe (Amazon) i Siri (Apple).

Neke od prednosti govornih sučelja su (Pearl 2016):

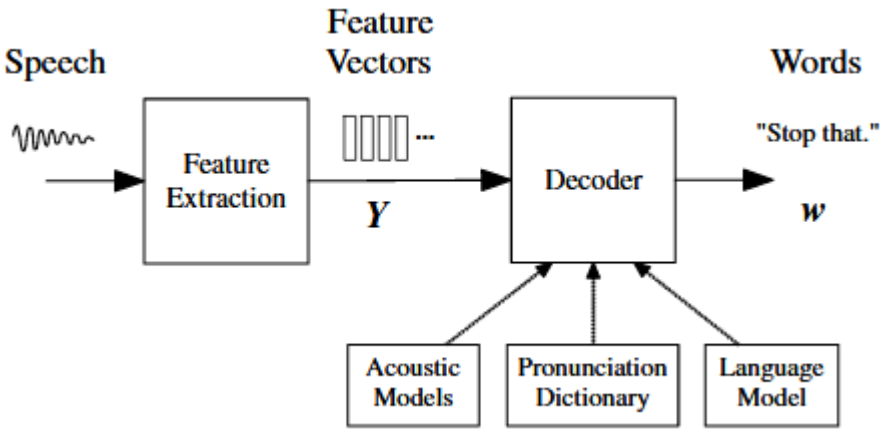
- Brzina – brzina govora često je brža od brzine tipkanja
- Slobodne ruke – korištenjem govornog sučelja aplikacija se može koristiti kad se klasičnim korištenjem ne bi mogla, npr. kod vožnje automobila
- Intuitivnost – kod korištenja novog sustava govorno sučelje može biti lakše za korištenje

Kako bi se omogućila komunikacija s korisnikom, govorna sučelja se sastoje od dvije komponente: automatsko raspoznavanje govora i sinteza govora.

5.1. Automatsko raspoznavanje govora

Kompleksnost raspoznavanja govora ovisi o slijedećim parametrima (Jurafsky and Martin 2008):

- Broj riječi koje sustav treba moći raspoznati – raspoznavanje manjeg broja riječi (npr. da/ne) lakši je zadatak nego raspoznavanje cijelog vokabulara nekog jezika
- Koliko je govor prirodan i razgovoran – izolirane riječi je lakše raspoznati nego kontinuirani govor. Također, razgovor čovjeka s strojem je lakše raspoznati nego razgovor čovjeka s drugim čovjekom
- Kvaliteta opreme s kojom se govor snima i pozadinska buka – što je oprema kojom se govor snima kvalitetnija i pozadinska buka manje to će raspoznavanje govora biti bolje
- Varijacije u naglasku i ostalim karakteristikama govora – govor je lakše raspoznati ukoliko govornik priča standardnim naglaskom na kojemu je sustav za raspoznavanje treniran



Slika 7 - Arhitektura sustava za raspoznavanje govora (Gales and Young 2008)

Na slici 7 prikazana je arhitektura sustava za raspoznavanje govora prema autorima Gales i Young. Ulaz u sustav je snimljen zvuk govora. Kako bi se zvuk mogao dalje procesirati potrebno ga je pretvoriti u vektor značajki. Za to se obično koriste MFCC (engl. *mel-frequency cepstral coefficients*). Nakon što dobijemo vektor značajki za svaki prozor, cilj dekodera (engl. decoder) je pronaći niz riječi za koje je najveća vjerojatnost da odgovaraju sekvenci vektora značajki. Ovo je prikazano na formuli (5.1) gdje je Y sekvenca vektora značajki, a w sekvenca riječi. Budući da je ovo kompleksan problem za modelirati primjenjuje se Bayesov teorem te se dobiva formula (5.2)[1]. Vjerojatnost P(Y|w) određuje se pomoću akustičnog modela (engl. *acoustic model*, dok se P(w) određuje pomoću modela jezika (engl. *language model*) (Gales and Young 2008).

$$\omega^* = \underset{\omega}{\operatorname{argmax}}\{P(\omega|Y)\} \quad (5.1)$$

$$\omega^* = \underset{\omega}{\operatorname{argmax}}\{P(Y|\omega)P(\omega)\} \quad (5.2)$$

Akustični model određuje koji je fonem izgovoren na temelju vektora značajki. Akustični modeli su često realizirani koristeći skrivene Markovljeve modele (engl. *hidden Markov model*). Rječnik izgovora (engl. *pronounciation dictionary*) sadrži riječi zapisane u obliku fonema od kojih se riječ sastoji. Uz pomoć rječnika izgovora od fonema se dolazi do riječi. Model jezika se obično realizira koristeći n-gram koji određuje vjerojatnost riječi u sekvenci prema n-1 prethodniku. Po završetku govora dekoder određuje najvjerojatniju sekvencu riječi koju je korisnik izgovorio (Gales and Young 2008).

[1] Nazivnik P(Y) koji se dobije koristeći Bayesov teorem je konstantan te se može zanemariti budući da se traži maksimalna vrijednost izraza

Prije navedeni sustav sastoji se od više modela koje je potrebno posebno trenirati. Moderna istraživanja na ovom području okreću se prema sustavima s jednim modelom (tzv. *end-to-end)*. Jedan primjer ovog je rješenje prezentirano od strane autora Graves i Jaitly gdje se za raspoznavanje govora koristi samo jedna rekurentna neuronska mreža (engl. *recurrent neural network*, kraće RNN) (Graves and Jaitly 2014).

5.2. Sinteza govora

Sinteza govora (engl. *speech synthesis*) ili pretvaranje teksta u govor (engl. *text-to-speech*, TTS) je proces u kojemu računalo generira govor, odnosno proces u kojemu se tekst pretvara u govor (Taylor 2009).

Neke od metoda za sintezu govora su (Sciforce 2020):

- Sinteza spajanjem (engl. *concatenative* synthesis)
- Sinteza formanata (engl. *formant synthesis*)
- Parametarska sinteza (engl. *parametric synthesis*)
- Sinteza korištenjem dubokih neuronskih mreža

5.3. Govorna sučelja i interaktivna fikcija

Budući da korisnik s djelima interaktivne fikcije komunicira isključivo putem teksta, moguće je na jednostavan način implementirati govorno sučelje. Na ovaj način korisnici bi mogli igrati igre interaktivne fikcije dok obavljaju neke druge poslove. Međutim, sve igre nisu prilagođene za igranje preko govornog sučelja. Na primjer, igre poput *Zorka* koje su bazirane na zagonetkama bilo bi jako teško igrati s govornim sučeljem. Zato je bitno odabrati igre koje su manje bazirane na zagonetkama, a više na priči.

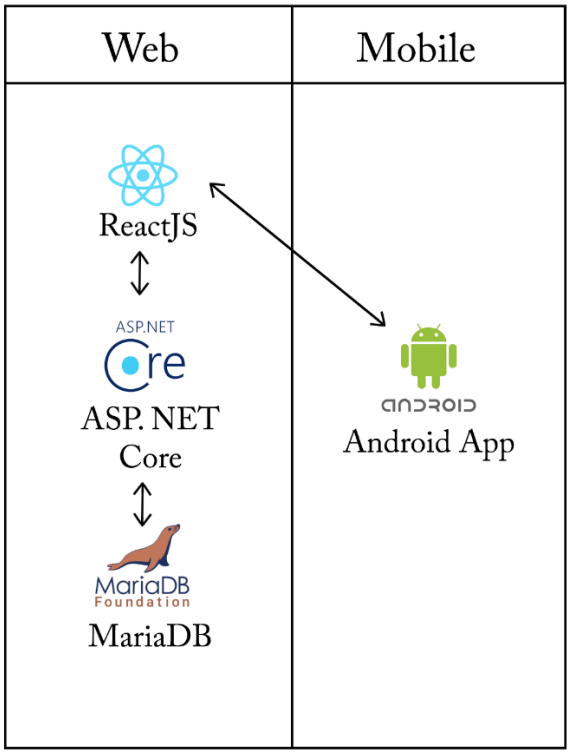
6. Projektni zadatak

6.1. Opis zadatka

Kao dio ovog diplomskog rada osmišljen je i izrađen sustav za razvoj interaktivne fikcije koji korisniku omogućuje da izrađuje i igra jednostavnije igre interaktivne fikcije. Osim samog sustava za razvoj interaktivne fikcije, izrađena je i mobilna Android aplikacija koja korisniku pruža mogućnost da igra igre pomoću govornog sučelja.

6.2. Arhitektura sustava

Sustav se sastoji od web aplikacije i Android aplikacije. Klijentska strana web aplikacije napisana je koristeći programski okvir (engl. *framework*) *ReactJS*. Većina programske logike odvija se na klijentskoj strani. Klijentska strana web aplikacije šalje zahtjeve na serversku stranu. Serverska strana je realizirana kao web aplikacijsko programsko sučelje (engl. *application programming interface,* kraće API) napisano s programskim okvirom *ASP.NET Core* koristeći programski jezik C#. Većina zadataka serverske strane sastoje se od komunikacije s bazom i autentifikacije. Za pohranjivanje podataka koristi se baza podataka *MariaDB*. Android aplikacija koristi modul *WebView* koji omogućuje otvaranje web stranica unutar same aplikacije. Svaka od ovih aplikacija biti će dodatno opisana u nastavku. Na slici 8 može se vidjeti vizualni prikaz arhitekture sustava.



Slika 8 - Arhitektura sustava

6.3. Funkcionalnosti sustava

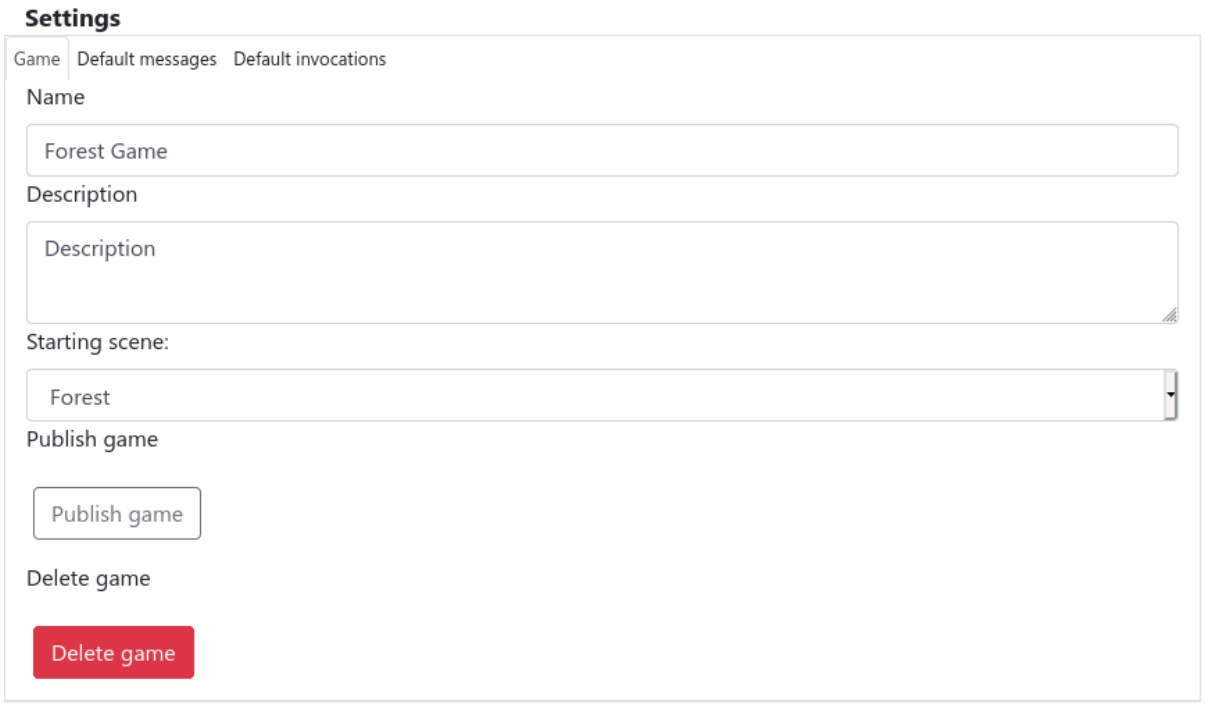
6.3.1. Web aplikacija

U web aplikaciji korisnik može kreirati i igrati igre interaktivne fikcije. Ukoliko korisnik želi kreirati igre mora se prijaviti u sustav s postojećim korisničkim računom ili napraviti novi račun. Za igranje igara nije potreban račun.

Ukoliko se korisnik odluči za igranje igara klikom na *Play* otvara mu se popis igara koje su do sada objavili ostali korisnici. Nakon što odabere igru otvora mu se stranica koja sadrži okvir u kojem mu se pokazuje inicijalna poruka igre koju je otvorio. Korištenjem okvira za tekstualni unos koji se nalazi ispod okvira s porukama korisnik može upisivati komande sustavu. Nakon što korisnik unese unos i pritisne tipku za novi red poruka se obrađuje te se rezultat nadodaje u okvir s porukama. Na ovaj način korisnik može igrati igru. Taj postupak može se vidjeti na slici 9.

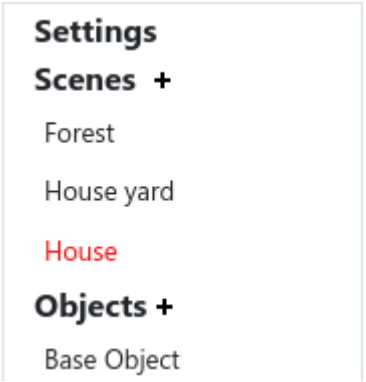
Slika 9 - Izgled stranice za igranje igara

Za kreiranje igara korisnik treba kliknuti na opciju *Create*. Tada mu se otvara popis igara koje je prije kreirao te ih može nastaviti uređivati. Također, korisnik može kreirati novu igru. U tom slučaju korisnik unosi ime igre te se potom otvara stranica za kreiranje igara. Na početku se korisniku prikazuje okvir u kojemu može mijenjati osnovne postavke igre: Ime igre, opis igre i početnu scenu. Izgled osnovnih postavki igre može se vidjeti na slici 10.



Slika 10 - Osnovne postavke igre

Putem navigacije koja se nalazi lijevo od okvira za uređivanje korisnik može uređivati scene i objekte koje igra sadrži. Koncept scene u ovome sustavu ekvivalentan je konceptu soba u sustavima koji su prethodno opisani. Klikom na znak plus koji se nalazi do riječi *Scenes* i *Objects* korisnik može kreirati nove scene i nove objekte. Izgled navigacije može se vidjeti na slici 11.

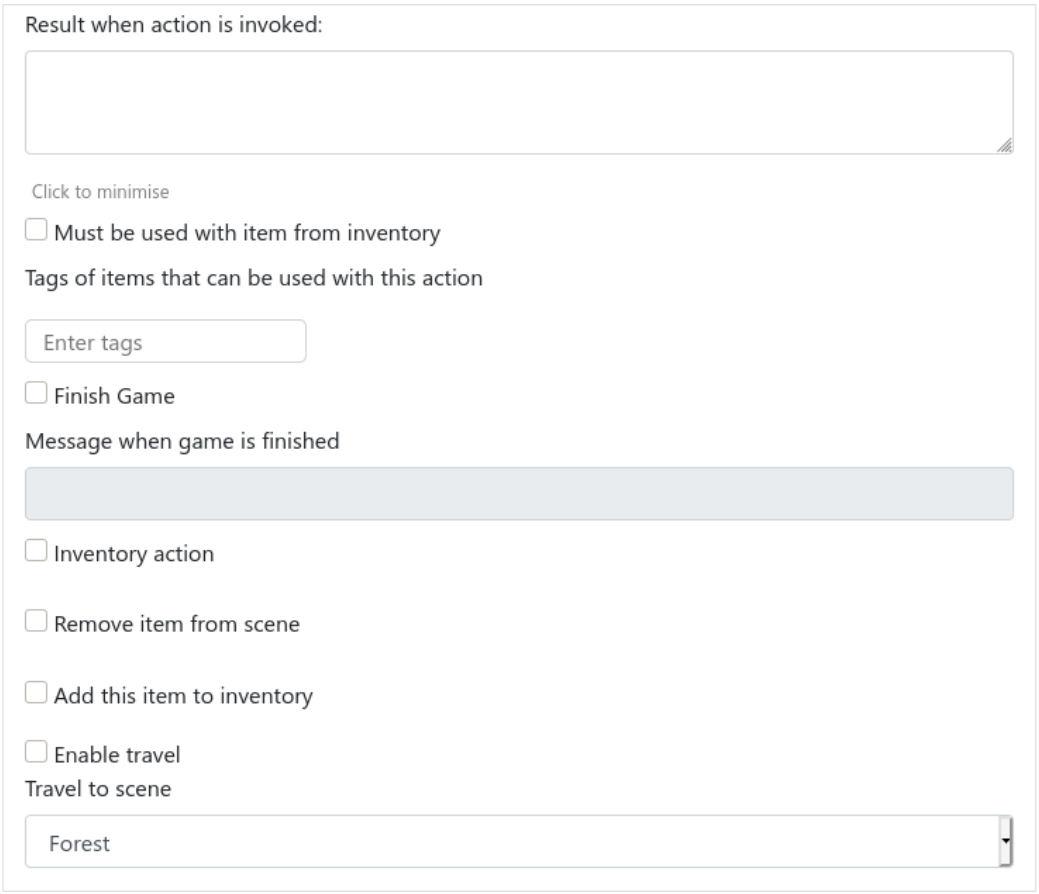


Slika 11 - Izgled navigacije kod kreiranja

Osnovni način za interakciju s igrom je putem akcija. Akcije se događaju kada parser prepozna unos korisnika. Akcije mogu biti vezane uz scenu, objekte ili likove. Najjednostavniji rezultat akcije je ispis teksta bez da se mijenja interno stanje igre. Osim samog ispisa teksta akciju se može prilagoditi sa dodatnim postavkama. Akciju se može ograničiti tako da ju se može koristiti samo korištenjem objekta koji je korisnik prethodno uzeo. Za to se definiraju oznake objekata koje se moraju podudarati s oznakama koje akcija dozvoljava. Rezultat akcije također može biti i kraj igre. U tom slučaju može se definirati poruka koja se korisniku prikazuje kako bi mu objasnila zašto je došlo do kraja igre. Također, kao rezultat akcije može se definirati putovanje na neku drugu scenu. Ukoliko akcija pripada objektu ona može imati slijedeća svojstva:

- Akcija se može ograničiti tako da se koristi isključivo kad je objekt kojemu akcija pripada uzet od strane igrača.
- Kao rezultat akcije objekt se briše iz scene
- Kao rezultat akcije igrač uzima objekt iz scene

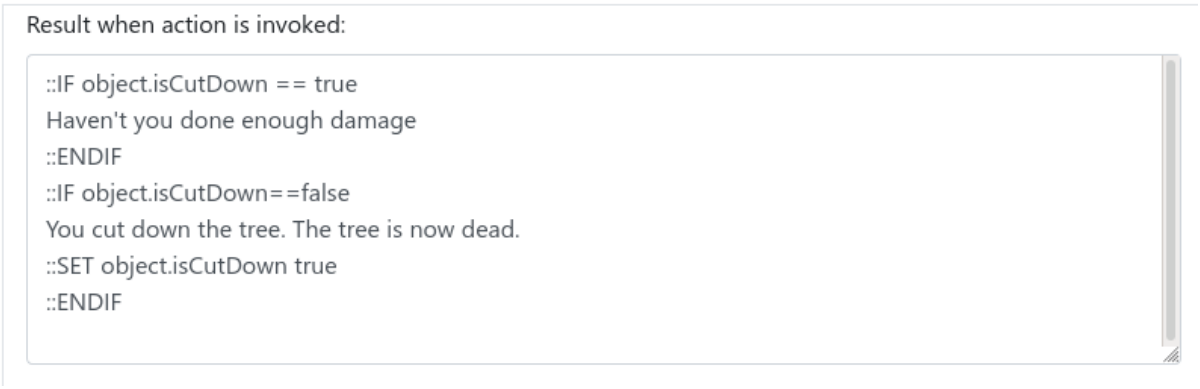
Postavke kojima korisnik implementira akciju prikazane su na slici 12.



Slika 12 - Sve mogućnosti za definiranje rezultata akcije

Za svaku akciju dodaju se nazivi koji tu akciju prizivaju. Svaka akcija može imati više naziva. Ti nazivi moraju biti jedinstveni unutar jednog objekta, scene ili lika. U suprotnom se korisniku prikazuje greška te nije moguće igrati ili objaviti igru dok se greška ne popravi. Na svakom mjestu gdje korisnik mora upisati nazive koji prizivaju akciju postoji mogućnost za generiranje sinonima već upisanih akcija kako bi se igrač mogao izraziti na što više načina.

Kako bi se omogućilo grananje kod izvođenja akcija, svakom objektu, sceni i liku moguće je definirati varijable. Varijable mogu biti vrijednosti *true/false*, brojčane vrijednosti ili niz znakova. Inicijalne vrijednosti varijabli se postavljaju kod kreiranja igre. Unutar prostora za upis teksta kod akcija mogu se koristiti naredbe za grananje (::IF i ::ENDIF) i naredba za postavljanje varijable (::SET). Naredba *::IF* provjerava uvjet u nastavku naredbe te izvodi dio unosa koji se nalazi između te naredbe i oznake za kraj grananja *::ENDIF*. Sustav dopušta ugniježđena grananja. Naredba *::SET* postavlja vrijednost varijable na neku od dopuštenih. Ova aplikacije je ograničena na ove naredbe te je to jedno od područja na kojima bi se ova aplikacija mogla unaprijediti u budućnosti. Prikaz korištenja naredbi može se vidjeti na slici 13.



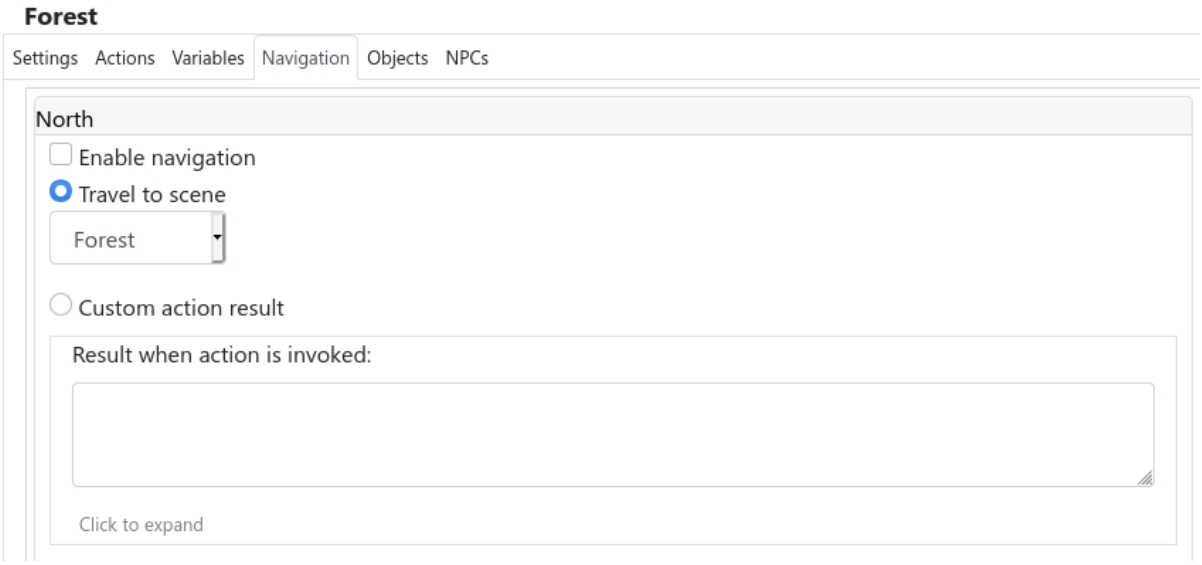
Slika 13 - Primjer korištenja naredbi

Scene definiraju tri događaja kojima se mogu pridružiti akcije. Ti događaji se pozivaju u slijedećim situacijama:

- Kad igrač prvi put uđe u scenu
- Svaki put kad igrač uđe u scenu
- Svaki put kad igrač izađe iz scene

Navigacija između scena se definira u kartici *navigation* gdje se određuje što će se dogoditi kad igrač upiše naredbu za kretanje. Navigacija se može uključiti ili isključiti. Ukoliko je navigacija uključena korisnik može odabrati da se prilikom poziva određene navigacije

promjeni scena na neku drugu ili može definirati posebnu akciju. Primjer definiranja navigacije može se vidjeti na slici 14.



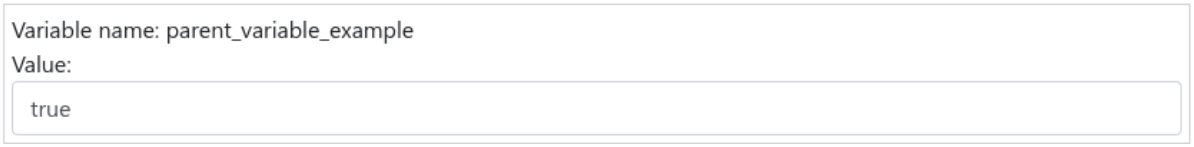
Slika 14 - Primjer navigacije

Putem kartica *Objects* i *NPCs* u scenu se mogu dodavati objekti i likovi. Objektima se definiraju nazivi koji ih pozivaju te oni moraju biti jedinstveni unutar iste scene. Za objekte se definiraju akcije koje se mogu izvoditi nad njima.

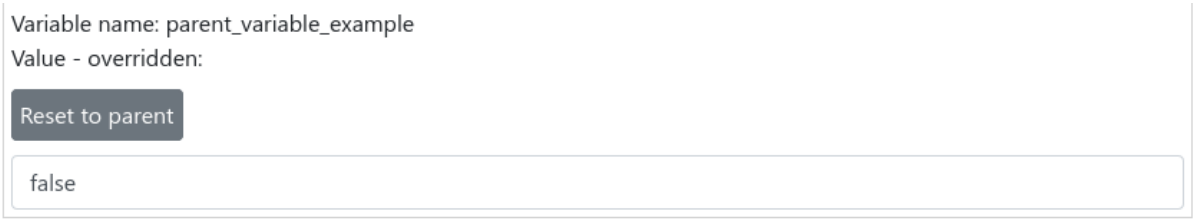
Aplikacija implementira koncept nasljeđivanja među objektima. Korisnik može kreirati predloške objekata kojima se mogu dodavati akcije i varijable kao i običnom objektu. U igri uvijek postoji jedan predložak koji se zove *Base object* (engl. temeljni objekt). Svaki objekt ili predložak objekta koji korisnik kreira može kao roditelja imati ili temeljni objekt ili neki drugi predložak objekta. Time se stvara hijerarhijska struktura objekata. Kad se kreira novi objekt za svaku akciju može se ili ostaviti akcija od roditeljskog objekta ili se akciju može nadjačati. Kad se akcija nadjača može se odabrati opcija da se roditeljska akcija izvede prije akcije s kojom je nadjačana. U tom slučaju izvodi se samo akcija u okviru za tekstualni opis roditeljskog objekta, ali ne i dodatne opcije. Izgled postavki s kojima se nadjačavaju akcije može se vidjeti na slici 15.

Slika 15 - Primjer nadjačane akcije

Na sličan način se nadjačavaju i varijable gdje se u novom objektu može promijeniti inicijalna vrijednost varijabli koje su definirane u roditeljskom objektu. Ovime se smanjuje količina akcija koju korisnik mora definirati ukoliko se akcije ponavljaju za više objekata. Izgled varijabli koja nije nadjačana može se vidjeti na slici 16, dok se nadjačana varijabla može vidjeti na slici 17.

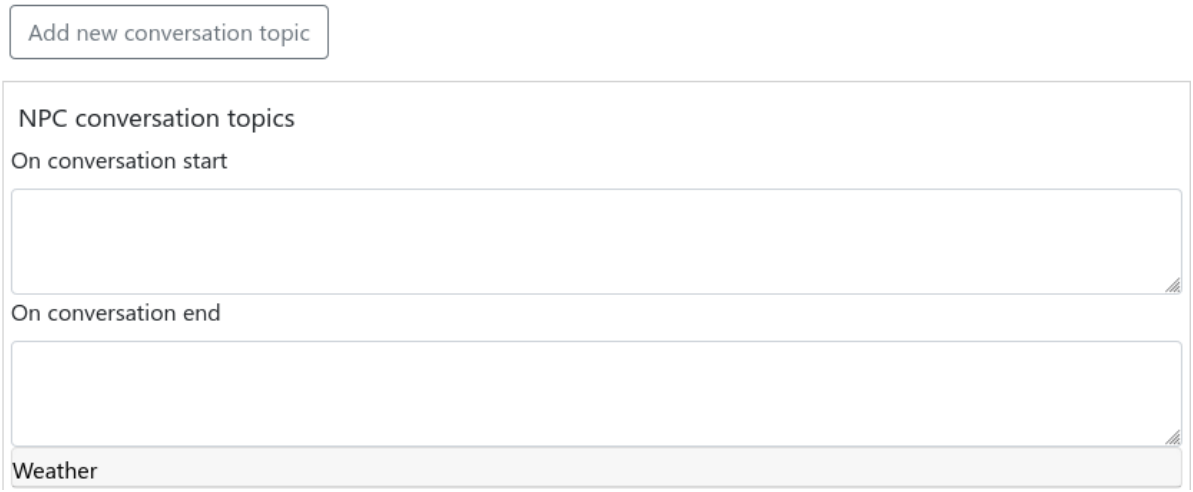


Slika 16 - Primjer varijable koja nije nadjačana



Slika 17 - Primjer varijable koja je nadjačana

Likovi (*NPCovi*) su posebna vrsta objekata koji ne podržavaju nasljeđivanje, ali imaju dodatnu mogućnost dodavanja razgovora. Razgovor s likom se odvija prema modelu „*tema razgovora*“ koji je prethodno opisan. Korisnik može dodati tekst koji se igraču ispisuje prilikom početka i kraja razgovora te definirati teme razgovora. Svaka tema razgovora se realizira kao akcija. Izgled kartica za razgovore može se vidjeti na slici 18.



Slika 18 - Kartica teme razgovora

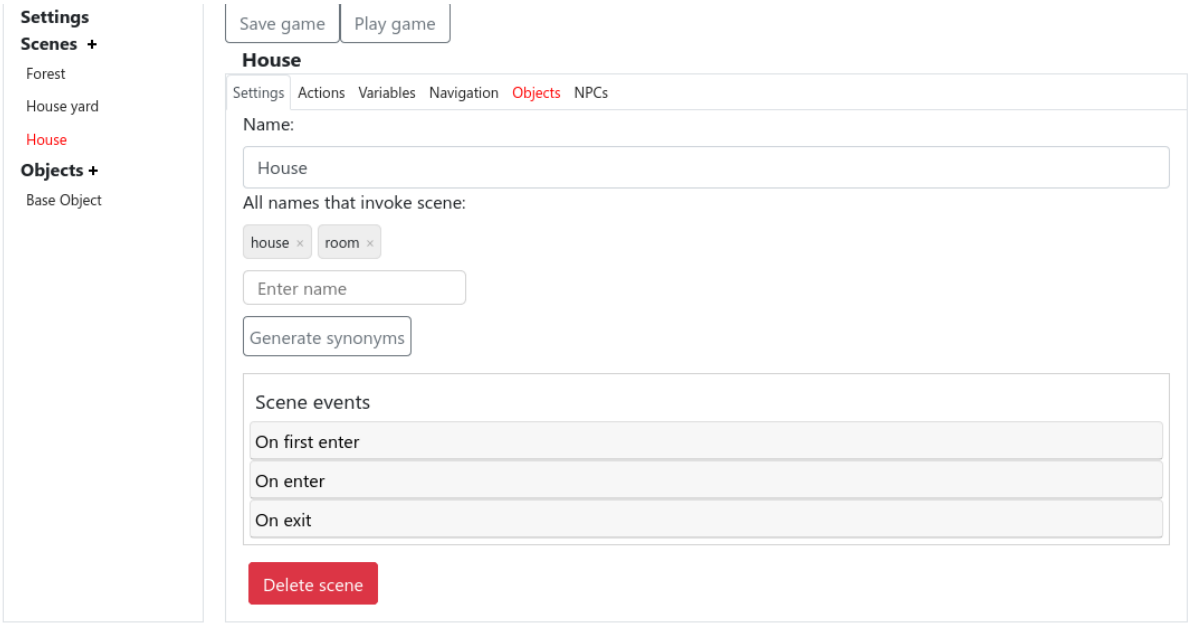
U uređivaču *settings* (postavke) korisnik može definirati poruke koje su univerzalne za cijelu igru te se korisniku ispisuje u slijedećim situacijama:

- Kada korisnik zatraži navigaciju koja nije dostupna
- Kada korisnik pokuša koristiti akciju koja zahtjeva neizravni objekt za korištenje
- Kada parser ne može pronaći odgovarajuću akciju za unos korisnika
- Kada je akcija uspješno obavljena, ali nije definirana poruka za tu akciju

Osim toga korisnik može dodati imena koja prizivaju često korištene akcije:

- Korištenje navigacije
- Pričanje s likovima
- Završetak pričanja s likovima
- Ispuštanje objekta koji je prethodno uzet

Pri svakom unosu sustav provjerava postoje li neke od grešaka. Ukoliko postoji greška, nad elementom gdje se nalazi greška prikazati će se poruka o grešci. Također, slova u navigaciji koja vodi do tog elementa, kao i kartica gdje se greška nalazi, biti će označeni crvenim slovima. Moguće greške su: nevaljano ime varijable, ime varijable koje nije jedinstveno u objektu, nevaljana vrijednost varijable, nazivi koji prizivaju akciju nisu jedinstveni, nazivi koji prizivaju objekte u istoj sceni nisu jedinstveni, nazivi koji prizivaju likove u istoj sceni nisu jedinstveni. Način označavanja grešaka može se vidjeti na slici 19.



Slika 19 - Prikaz uređivača kad postoji greška

6.3.2. Mobilna aplikacija

Mobilna aplikacija se sastoji od komponente koja otvara web aplikaciju i gumba koji započinje govornu interakciju. Korisniku se otvara naslovna stranica web aplikacije. Korisnik zatim može koristiti aplikaciju na isti način koji bi koristio i web aplikaciju. Kad korisnik dođe do stranice na kojoj se igraju igre (bilo preko *play* izbornika, ili kod kreacije igre) može pritisnuti na gumb. Pritiskom se čita posljednja poruka koja je ispisana u okviru sa porukama. Po završetku čitanja poruke čuje se zvučni signal koji označava da program sluša glasovni unos od korisnika. Nakon što korisnik završi govoriti, poruku koju je izgovorio se pretvara u tekst te se prosljeđuje web aplikaciji. Kad web aplikacija generira odgovor on se čita korisniku. Ovaj postupak se odvija sve dok korisnik ne odluči prestati igrati. Ukoliko govor nije razumljiv ili korisnik nije govorio sustav za raspoznavanje govora će se ponovno pokrenuti nekoliko puta. Ako i nakon toga nema raspoznatljivog govora govorno sučelje se gasi te treba biti ponovno ručno pokrenuto pritiskom na gumb. Izgled mobilne aplikacije, zajedno s porukom koja se pojavi ako korisnik nije na stranici gdje se igraju igre kad pokuša pokrenuti govornu interakciju, može se vidjeti na slici 20.

Slika 20 - Izgled mobilne aplikacije

6.4. Realizacija projekta

6.4.1. Serverska aplikacija

Serverski dio web aplikacije napisan je koristeći programski okvir *Asp.Net Core*. Prethodnik programskog okvira *Asp.Net Core* bi je *Asp.Net* čija je najveća mana bila to što se mogao izvoditi isključivo na *Windows* platformi. Cilj *Asp.Net Corea* bio je razvoj web platforme koja je imala slijedeća svojstva (Lock 2018):

- Mogućnost razvoja i pokretanja na različitim platformama
- Modularna arhitektura za lakše održavanje
- Razvijena u potpunosti kao softver otvorenog koda
- Mogućnost korištenja s trenutnim trendovi u svijetu web razvoja

Pomoću *Asp.Net Corea* mogu se kreirati tradicionalne web aplikacije čiji se sadržaj generira na serverskoj strani (engl. *server side rendering*), kao i aplikacije koje generiraju sadržaj na klijentskoj strani te im je potreban REST[2] servis za komunikaciju (Lock 2018). Za

[2] REST (engl. *Representational State Transfer,* reprezentacijski prijenos stanja) je arhitektura koja se temelji na „HTTP pozivima bez stanja radi čitanja, kreiranja, ažuriranja i brisanja podataka“ (Lock 2018)

ovaj projekt serverska strana realizirana je kao web API s kojim klijentska strana komunicira kako bi dobila podatke koje su joj potrebne za rad.

Asp.Net Core nudi niz predložaka koji omogućuju brži početak razvoja. Za ovaj projekt korištena je naredba:

```
dotnet new react -o <ime mape> -au Individual
```

Prema službenoj dokumentaciji, ova naredba kreira novu *Asp.Net Core* aplikaciju kao API *backend* i *React* klijentsku aplikaciju za korisničko sučelje. Također, u aplikaciju se dodaje *Identity* sustav koji služi za autorizaciju i autentifikaciju korisnika. Sustav *Identity* generira potrebne tablice u bazi podataka za pohranu podataka o korisnicima, korisničko sučelje za prijavu i registraciju korisnika te korisničko sučelje u kojem korisnik mijenja postavke svog računa . Kada se kreira *React* aplikacija sustav također kreira i slijedeće *React* komponente:

- *Login.js* – upravlja tokom prijave korisnika
- *Logout*.js – upravlja tokom odjave korisnika
- *LoginMenu.js* – prikazuje profil korisnika i poveznice za odjavu kad je korisnik prijavljen te poveznice za prijavu i registraciju kad korisnik nije prijavljen
- *AuthorizeRoute.js* – komponenta koja zahtjeva od korisnika da bude prijavljen da bi se stranica prikazala

Osim toga se generira i instanca *authService* koja upravlja s detaljima procesa autentifikacije i daje ostatku aplikacije podatke o prijavljenom korisniku (Microsoft 2020). Na slici 21 može se vidjeti korištenje komponente *AuthorizeRoute* u klijentskoj aplikaciji kako bi se ograničilo prikazivanje komponente *Create* (dio aplikacije gdje se kreira igra) samo korisnicima koji su prijavljeni.

```
return (
  <Layout>
    <Route exact path="/" component={Home} />
    <AuthorizeRoute path="/create" component={Create} />
    <Route path="/play" component={Play} />
    <Route
      path={ApplicationPaths.ApiAuthorizationPrefix}
      component={ApiAuthorizationRoutes}
    />
  </Layout>
);
```

Slika 21 - Korištenje komponente AuthorizeRoute

Web aplikacije u *Asp.Net Core* koriste MVC (*engl. Model-View-Controller,* model-pogled-kontroler) uzorak dizajna (engl. *design pattern*). Ovaj uzorak se sastoji od tri komponente (Lock 2018):

- Model koji označava podatke koji se trebaju prikazati
- Pogled koji predstavlja predložak koji prikazuje podatke sadržane u modelu
- Kontroler koji ažurira model i odabire odgovarajući pogled.

Opisani uzorak dizajna koristi se i kod web API aplikacija, ali umjesto odgovora koji je namijenjen korisniku vraća se odgovor koji je namijenjen nekom drugom programu (Lock 2018).

Kontroler prihvaća HTTP zahtjeve na putanji koja mu je specificirana te prosljeđuju zahtjev jednoj od akcija (funkcija) koja je definirana u njemu (Lock 2018). Kontroleri nasljeđuju klasu *Controller*. Kod definicije klase mogu se specificirati atributi koji upravljaju načinom ponašanja kontrolera. Na slici 22 vidi se definicija jednog kontrolera za koji su definirani atributi *Authorize, ApiController i Route*. Atribut *ApiController* omogućuje dodatne opcije za API poput automatskog odbijanja zahtjeva nepravilnog formata i omogućavanje korištenja atributa *Route*. *Route* omogućuje specificiranje putanje (URL-a) do određenog kontrolera i njegove akcije (Microsoft 2020). Atribut *Authorize* dio je *Identity* sustava te omogućuje izvođenje akcije samo ako je korisnik autoriziran (Microsoft 2020). Primjer jednog kontrolera može se vidjeti na slici 22.

```
[Authorize]
[ApiController]
[Route("[controller]/[action]")]
public class GameEditorController : Controller
```

Slika 22 - Primjer kontrolera GameEditorController

U aplikaciji su definirana tri kontrolera:

- *GameEditorController* – nudi pristup tablici *GameEditors* u bazi koja sadrži podatke o uređivačima za igre
- *GamePublishController* – nudi pristup tablici *GameEditors* u bazi koja sadrži podatke o objavljenim igrama
- *Word2VecController* – nudi mogućnost pronalaženja sličnosti među riječima

Asp.Net Core nudi mogućnost automatskog pretvaranja (engl. *binding*) podataka koji se šalju putem HTTP zahtjeva u C# objekte. Za to je dovoljno definirati objekt kao parametar metode koja se poziva. Na slici 23 može se vidjeti primjer akcije *CreateGame* u kontroleru *GameEditorController*. Atribut *HttpPut* označava da se akcija izvodi kod *HTTP PUT* zahtjeva. Budući da se u PUT zahtjevu podaci šalju u tijelu zahtjeva potrebno je ispred parametra u koji se spremaju podaci staviti atribut *FromBody* kako bi došlo do automatskog pretvaranja.

```
[HttpPut]
public IActionResult CreateGame([FromBody] GameEditor gameEditor)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    gameEditor.UserId = userId;
    _db.Add(gameEditor);
    _db.SaveChanges();
    return StatusCode(StatusCodes.Status200OK);
}
```

Slika 23 - Primjer akcije u kontroleru

5.4.1.1. Baza podataka

Kao baza podataka korištena je *MariaDB*. *MariaDB* je relacijska baza podataka otvorenog koda koja je nastala kao *fork MySql-a* (MariaDB n.d.). Kako bi se izbjeglo direktno upravljanje s bazom podataka i pisanje *SQL* upita u ovom projektu se koristi *Entity Framework Core* (kraće EF). EF omogućava objektno-relacijsko mapiranje (engl. *object-relational mapping*, ORM), odnosno omoguće rad s bazom podataka korištenjem objekata. Također, EF smanjuje potrebu za pisanjem koda za pristup podacima (Microsoft 2020). Ukoliko želimo koristiti EF s bazom podataka *MariaDB* potrebno je instalirati paket Pomelo.EntityFrameworkCore.MySql.

Pristup podacima realizira se pomoću modela koji se sastoji od klasa entiteta i konteksta koji predstavlja vezu s bazom podataka (Microsoft 2020). Postoje tri pristupa za kreiranje modela (Microsoft 2020):

- Generiranje modela iz postojeće baze podataka
- Ručno kodiranje modela da bi odgovarao bazi podataka
- Kreiranje modela te generiranje baze iz njega pomoću migracija (tzv. *code-first* pristup)

U ovom projektu korišten je treći pristup. Prvo su napisane klase entiteta koje predstavljaju podatke koje se spremaju u bazu. Primjer ovakve klase može se vidjeti na slici 24. Također, mogu se vidjeti atributi[3] koji se mogu dodavati članovima klase, tj. svojstvima (engl. *property*). Atribut *Key* označava primarni ključ tablice. Atribut *DatabaseGenerated* označava automatsko generiranje atributa od strane baze podataka. Atribut *ForeignKey* označava vanjski ključ, kao što je u ovom slučaju atribut *Id* iz tablice *ApplicationUser*. Virtualno svojstvo tipa *ApplicationUser* označava tablicu iz koje je vanjski ključ, dok kolekcija tipa *PublishedGame* označava listu objekata tog tipa zato što se u tablici *PublishedGames* nalazi vanjski ključ koji pokazuje na tablicu *GameEditors*.

```
public class GameEditor
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public string GameEditorId { get; set; }
    [BindNever]
    [ForeignKey("Id")]
    public string UserId { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public string GameData { get; set; }

    public virtual ApplicationUser User { get; set; }
    public virtual ICollection<PublishedGame> PublishedGames { get; set; }
}
```

Slika 24 - Primjer klase entiteta

Drugi dio modela je kontekst. U njemu se definiraju svi entiteti kojima EF ima pristup. Entiteti se definiraju kao virtualna svojstva tipa *DbSet*. Na slici 25 može se vidjeti kako se definiraju svojstva za svaku od tablica: *GameEditor* i *PublishedGame*. Također, u metodi *OnModelCreating* definirana su pravila brisanja kod veza jedan-prema-više (engl. *one-to-many*). Ukoliko se *GameEditor* (na kojemu *PublishedGame* baziran) izbriše, vanjski ključ u tablici *PublishedGames* se postavlja na *null*. S druge strane, ukoliko se izbriše korisnik kaskadno se brišu i svi elementi *GameEditor* tablice koje je on kreirao.

[3] Atribut *BindNever* nije vezan uz bazu podataka, već se upotrebljava kako bi se sustavu signaliziralo da kod procesa automatskog pretvaranja ignorira ovo svojstvo. Time se sprječavaju *over-posting* napadi (Lock 2018).

```
public class ApplicationDbContext : ApiAuthorizationDbContext<ApplicationUser>
{
    public ApplicationDbContext(
        DbContextOptions options,
        IOptions<OperationalStoreOptions> operationalStoreOptions) : base(options, operationalStoreOptions)
    {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<PublishedGame>()
            .HasOne(e => e.Editor)
            .WithMany(e => e.PublishedGames)
            .OnDelete(DeleteBehavior.SetNull);
        modelBuilder.Entity<GameEditor>()
            .HasOne(e => e.User)
            .WithMany(e => e.GameEditors)
            .OnDelete(DeleteBehavior.Cascade);
        base.OnModelCreating(modelBuilder);
    }

    public virtual DbSet<GameEditor> GameEditors { get; set; }
    public virtual DbSet<PublishedGame> PublishedGames { get; set; }
}
```

Slika 25 - Primjer konteksta baze podataka

Nakon što se obave promjene nad modelom potrebno je dodati migraciju te ažurirati bazu podataka. To se radi sa slijedećim naredbama (Microsoft 2020):

```
dotnet ef migrations add [ime_migracije]

dotnet ef database update
```

6.4.2. Klijentska aplikacija

Klijentski dio web aplikacije napisan je koristeći programski okvir *ReactJs*. *ReactJS* je *Javascript* biblioteka koja služi za kreiranje korisničkih sučelja. Bazira se na komponentama koje upravljaju vlastitim stanjem te se mogu kombinirati kako bi se stvorilo kompleksno korisničko sučelje (ReactJS n.d.).

Dvije osnovne vrste komponenti su funkcionalne komponente (engl. *functional components*) i klasne komponente (engl. *class componenents*). Funkcionalne komponente nemaju vlastito stanje nego obično prikazuju podatke koje su im proslijeđene iz roditeljske komponente. Klasne komponente imaju vlastito stanje te se obično koriste za stvaranje kompleksnijih korisničkih sučelja tako da kombiniraju funkcijske komponente (Bhandari 2020).

```
import React from 'react';
import {Row, InputGroup, Input, Label} from "reactstrap";

export default function Name(props) {
    return (
        <InputGroup>
            <Row>
            <Label>
                {props.type} Name:
            </Label>
            </Row>
            <Row>
                <Input type="text" value={props.currentObject[props.property]}
                 onChange={(e) => props.onChange(props.property, e.target.value)} />
            </Row>
        </InputGroup>
    )
}
```

Slika 26 - Primjer funkcijske komponente

Na slici 26 prikazan je primjer funkcijske komponente. Sadržaj *return* tvrdnje koji liči na *XML* naziva se *JSX* te služi kao pojednostavljena sintaksa za kreiranje elemenata u *Reactu*. Na ovaj način mogu se stvarati *HTML* oznake, ali i *React* komponente (bilo funkcijske ili klasne). *JSX* elementi se u procesu prevođenja koda pretvaraju u poziv funkcije *React.CreateElement()* (ReactJS n.d.). Atributi koji se nalaze uz oznaku (na primjer, *type* atribut s vrijednošću *text* kod oznake *Input*) prosljeđuju se komponenti kao *props*. *Props* se funkcijskoj komponenti prosljeđuju kao parametar funkcije, dok se klasnoj komponenti prosljeđuje kao parametar konstruktora. Na ovaj način se prenose podaci o stanju iz roditeljske komponente na djecu. Za slanje podataka u suprotnom smjeru koriste se funkcije povratnog poziva (engl. *callback functions*). To su funkcije koje su definirane u roditelju te se prosljeđuju djetetu. Kad se element koji mijenja stanje roditeljske klase promijeni poziva se funkcije koja mu je proslijeđena. Elementi koji se nalaze unutar neke druge oznake prosljeđuju se kao *props.children*. Time se omogućuje hijerarhijsko kreiranje komponenti te se povećava ponovna iskoristivost koda. Kod funkcijske komponente na ekranu se pokazuju elementi koji se nalazi u *return* tvrdnji, dok se kod klasne komponente prikazuju elementi u *return* tvrdnji funkcije *render()*.

U ovom projektu korišten je veliki broj komponenti kako bi se kreiralo korisničko sučelje. Većina komponenata su funkcijske komponente, dok su klasne komponente samo komponente koje predstavljaju cijele stranice (*Home.js, Create.js* i *Play.js*).

6.4.3. Algoritmi za model svijeta i parser

Parser i model svijeta za igru interaktivne fikcije implementirani su na klijentskoj strani aplikacije. Prilikom pokretanja igre prvi korak je pretvorba strukture pomoću koje je igra zapisana u dvije različite strukture. Jedna struktura sadrži podatke o scenama, objektima i likovima koji se ne mijenjaju. Druga struktura sadrži podatke o trenutnom stanju igre: trenutnu scenu, je li igrač u razgovoru ili u sceni, podaci o varijablama, podaci o dostupnosti navigacije. Igra se može nalaziti u dva glavna stanja: stanje kada je igrač u sceni i stanje kada je igrač u razgovoru. Ukoliko se igrač nalazi u sceni igra se provodi na slijedeći način:

1. Iz korisničkog sučelja pozove se funkcija za procesiranje korisničkog unosa
2. Model svijeta generira sve akcije koje su moguće u tom trenutku ovisno o stanju igre
3. Parser s obzirom na korisnikov unos određuje koja će se akcija izvesti (ukoliko odgovarajuća akcija postoji)
4. Ukoliko akcija nije valjana vraća se zadana poruka. U suprotnom se akcija procesira na slijedeći način:
    1. Procesira se tekst i komande iz tekstualnog okvira akcije
    2. Procesira se kraj igre ukoliko je to rezultat akcije
    3. Procesira se uzimanje objekta ukoliko je to rezultat akcije
    4. Procesira se brisanje objekta iz scene ukoliko je to rezultat akcije
    5. Procesira se promjena scene ukoliko je to rezultat akcije
5. U korisničko sučelje se vraća tekstualni izlaz, kao i novo stanje igre

Ovaj postupak se ponavlja sve dok se igra ne završi. Ukoliko je igrač u razgovoru izvodi se prijašnji proces, ali se umjesto akcija generiraju teme razgovora te se određuje koja će se tema izvesti. Budući da su teme realizirane kao posebna vrsta akcija ostatak algoritma je isti.

Parser radi na način da uneseni tekst uspoređuje s tekstom koji je generiran za svaku akciju. Ovisno o vrsti akcije generira se različiti tekst. Kod akcija koje pripadaju sceni tekst koji priziva akciju se sastoji od svih imena koji prizivaju akciju ili kartezijevog produkta imena koji prizivaju akciju i imena koji prizivaju scenu. Kod akcija koje pripadaju objektu tekst koji priziva akciju se generira kao kartezijev produkt imena koji prizivaju akciju i imena koji prizivaju objekt. Ukoliko se akcija mora koristiti s neizravnim objektom kartezijev produkt se proširuje s imenima koja prizivaju neizravni objekt. Ekvivalentno se generiraju imena i za akcije nad likovima, samo što se umjesto imena objekata koriste imena likova.

Parser ima dva načina rada: način kad pronalazi akciju i način kad pronalazi temu razgovora. U oba načina rada prvi korak parsera je da izbaci zaustavne riječi. Kod načina rada gdje se pronalazi akcija parser prvo uspoređuje očišćeni unos sa svim generiranim izrazima. Ukoliko ne pronađe identične unose, izračunava se Levenshteinova udaljenost između svih mogućih akcija. Kao rezultat se odabire akcija koja ima najmanju udaljenost, pod uvjetom da je udaljenost manja ili jednaka 2. Ako to nije slučaj šalje se zahtjev na server gdje se korisnički unos uspoređuje s generiranim tekstom akcijama koristeći word2vec. Naredbe se uspoređuju tako da se za svaki niz riječi koji čine naredbu generiraju sve permutacije, odnosno svi mogući redoslijedi riječi. Zatim se svaka permutacija uspoređuje s naredbom koju je korisnik unio riječ po riječ (prva riječ permutacije s prvom riječi korisnikove naredbe, itd.). Nakon toga se računa prosječna sličnost za tu permutaciju. Na taj način se pronalazi naredba koja ima najveću sličnost s naredbom koju je korisnik unio. Naredba s najvećom sličnosti, kao i vrijednost koja označava sličnost, vraća se u klijentsku aplikaciju. Ukoliko je sličnost veća od definiranog praga naredba se prihvaća, u suprotnom se korisniku vraća poruka o krivoj naredbi. Tijekom izrade aplikacije eksperimentiralo se više vrijednosti za prag te je odlučeno koristiti prag od 0.65.

Za način rada kod pronalaženja teme za razgovor postupak je nešto jednostavniji. U unesenom korisničkom tekstu potrebno je pronaći ključne riječi koje su definirane za određenu temu razgovora. Prvo se iz teksta izbace zaustavne riječi te se pokuša pronaći riječ koja u potpunosti odgovara temi razgovora. Ako je to neuspješno za svaku riječ iz korisnikovog unosa se računa Levenshteinova udaljenost do svake od mogućih tema razgovora. Kao temu razgovora odabire se tema čija je udaljenost najmanja, pod uvjetom da je manja od 2. Slično kao i kod prvog načina rada, ukoliko prijašnji postupci nisu uspješno odabrali temu razgovora, korisnikov unos i moguće teme šalju se na server. Kod ovog načina rada uspoređuje se riječ po riječ koristeći word2vec. Tema razgovora koja je najsličnija korisnikovom unosu i vrijednost njezine sličnosti vraćaju se na klijentsku aplikaciju gdje se tema prihvaća ukoliko je sličnost veća od praga. Prednost ovog pristupa je to što se korisnik može izraziti na više načina. Mana ovog pristupa je to što se teme razgovora moraju sastojati od samo jedne riječi te postoji veća vjerojatnost da korisnik odabere temu koju nije htio.

Pronalaženje sličnosti među riječima koristeći word2vec mora se odvijati na serverskom dijelu aplikacije zato što su modeli koji sadrže vektorske vrijednosti riječi zauzimaju mnogo prostora na disku i u radnoj memoriji. Googleov predtrenirani model koji se koristi za ovaj projekt zauzima 3.1GB na disku te oko 8GB kada je učitan u radnu memoriju.

Ovakva veličina datoteke predstavlja problem i na serveru zato što je za učitavanje datoteke potrebno nekoliko sekundi što je neprihvatljivo kod programa interaktivne fikcije. Iz tog razloga potrebno je objekt spremiti u predmemoriju (engl. *cache*) kako bi se objekt učitavao samo jednom. Na slici 26 prikazan je način spremanja objekta u predmemoriju koristeći klasu *MemoryCache* koju pruža *Asp.Net Core*.

```
Vocabulary vocabulary;
if (!_cache.TryGetValue("Vocabulary", out vocabulary))
{
    vocabulary = new Word2VecBinaryReader().Read("GoogleNews-vectors-negative300.bin");

    var cacheEntryOptions = new MemoryCacheEntryOptions()
        .SetSlidingExpiration(TimeSpan.FromHours(1));

    _cache.Set("Vocabulary", vocabulary, cacheEntryOptions);
}
```

Slika 27 - Način za spremanje objekta u predmemoriju

6.4.4. Mobilna aplikacija

Mobilna aplikacija realizirana je kao Android aplikacija napisana u programskom jeziku *Java*. Aplikacija se sastoji samo od jedne aktivnosti (engl. *activity*) na kojoj se nalaze dva elementa: gumb koji pokreće govorno sučelje i komponente *WebView* koja prikazuje prije opisanu web aplikaciju. Očita prednost korištenja komponente *WebView* je brži razvoj budući da nije potrebno isti kod pisati za više različitih platformi. Također, moguće je pozivati funkcije iz Android aplikacije direktno iz web aplikacija koristeći *Javascript*. Neke od mana korištenja su lošije performanse od potpuno izvorne (engl. *native*) aplikacije i manje dostupnih mogućnosti u usporedbi s internet preglednikom. Osim toga, budući da web aplikacije koje se pokreću u *WebViewu* mogu pozivati Android funkcije bitno je da se u njemu ne izvodi kod kojeg nismo sami napisali zato što to predstavlja sigurnosni rizik (Google 2020).

Da bi se *React* aplikacija mogla koristiti u *WebViewu* potrebno je postavke *JavascriptEnabled* i *DomStorageEnabled* postaviti na *true* budući da se *React* oslanja na te dvije tehnologije. Kako bi se omogućilo pozivanje funkcija iz web aplikacije koristi se *JavascriptInterface* (Google 2020). Na slici 28 prikazana je klasa koja definira jedno *Javascript* sučelje koji pri promjeni podataka u web aplikaciji pokreće sintezu teksta. Ovo sučelje se u *WebView* dodaje koristeći slijedeću metodu:

```
webView.addJavascriptInterface(new WebAppInterface(this), "Android");
```

```
public class WebAppInterface {
    Context mContext;
    WebAppInterface(Context c) {
        mContext = c;
    }

    @JavascriptInterface
    public void dataChanged(String data) {
        tts.speak(data, 0, null, "ttsId");
    }
}
```

Slika 28 - Primjer klase u kojoj je definiran JavascriptInterface

Kako bi se govorno sučelje radilo potrebno je da se kod svake promjene na web aplikaciji pročita novi izlaz korisniku. To je realizirano koristeći sučelje *MutationObserver* koje prati promjene nad specificiranim elementom. To se izvodi koristeći *Javascript* kod koji je prikazan na slici 29. Na slici se može vidjeti kako se prilikom promjene nad elementom *playerDiv* (koji predstavlja prostor gdje se prikazuju poruke korisniku kada igra igru) poziva funkcija *dataChanged* koja je definirana na sučelju *Android* (koje smo specificirali koristeći metodu *addJavascriptInterface*).

```
private String javascript = "const targetNode = document.getElementById('playerDiv');\n" +
        "const config = { attributes: false, childList: true, subtree: false };\n" +
        "const callback = function(mutationsList, observer) {\n" +
        "el = document.getElementById(\"playerInput\");\n" +
        "Android.dataChanged(document.getElementById('textToSpeech').innerHTML)\n" +
        "};\n" +
        "const observer = new MutationObserver(callback);\n" +
        "observer.observe(targetNode, config);";
```

Slika 29 - Javascript kod koji se umeće u web aplikaciju

Govorno sučelje implementirano je koristeći klase *SpeechRecognizer* i *TextToSpeech*. Ove klase koriste ugrađene Android aplikacije za raspoznavanje i sintezu govora. Servis *SpeechRecognizer* kombinira mrežno i izvanmrežno raspoznavanje govora (Google 2020). Prednost korištenja ovih servisa je u tome što nisu ograničeni API limitima kao što je često slučaj, dok je mana to što se oslanjaju na Googleove servise u Androidu koji ne smiju biti izbrisani ili onemogućeni da bi aplikacija radila. Proces se odvija na slijedeći način:

1. Web aplikacija na prije opisani način poziva Android metodu koja pretvara najnoviji tekst iz web aplikacije u govor
2. Pri završetku reproduciranja sintetiziranog govora pokreće se raspoznavanje govora koristeći metodu *startListening*

3. Nakon što *SpeechRecognizer* uspješno raspozna govor korisnika poziva se povratna funkcija *onResults* u kojoj se korisnikov govor ubacuje u web aplikaciju. Ukoliko raspoznavanje govora nije uspješno poziva se povratna funkcija *onError* te se još nekoliko puta pokušava raspoznati govor. Ako i nakon tih pokušaja raspoznavanje nije uspješno govorno sučelje se prekida
4. Nakon što web stranica primi unos, ona će generirati odgovor koji će ponovno pokrenuti sintezu govora

Ovaj postupak se ponavlja dok korisnik ne prestane igrati igru.

6.4.5. Korištene biblioteke

U tablici ispod prikazane su dodatne biblioteke koje su korištene u ovom projektu. Osim navedenih biblioteka instalirane su i biblioteke o kojima te biblioteke ovise.

Tablica 2 - Korištene biblioteke u projektu

Ime paketa	Verzija	Način korištenja u projektu	Repozitorij
Word2Vec.Tools	2.0.1	Korišteno za učitavanje word2vec modela i računanje udaljenosti riječi	NuGet
Pomelo.EntityFrameworkCore.MySql	3.1.2	Pruža mogućnost korištenja *MariaDB* s EF *Core*	NuGet
Combinatorics	1.1.0.19	Izračunavanje permutacija kod uspoređivanja riječi	NuGet
bootstrap	4.5.1	Omogućava izradu responzivnih web sučelja	npm
classnames	2.2.6	Pojednostavljuje korištenja CSS klasa u *React* komponentama	npm
immutability-helper	3.1.1	Omogućava mijenjanje kopije objekta bez da se mijenja originalni objekt	npm
js-levenshtein	1.1.6	Izračunavanje Levenshteinove udaljenosti	npm
logical-expression-parser	1.0.0	Parsiranje logičkih izraza kod pisanja skripti kod akcija	npm
react-tag-input	6.4.3	Nudi mogućnost unosa tagova koja se koristi kod imena koji prizivaju objekte/akcije	npm
reactstrap	8.5.1	Omogućava korištenje bootstrap klasa kao *React* komponente	npm

stopword	1.0.3	Omogućava micanje zaustavnih riječi što se koristi u parseru	npm
synonyms	1.0.1	Nudi mogućnost generiranja sinonima što se koristi kod imena akcija i objekata	npm
uuid	8.3.0	Generira UUID-ove što se koristi za generiranje identifikacijski oznaka za objekte, scene, likove i akcije	npm

7. Zaključak

U ovome radu opisan je pojam interaktivne fikcije, njezin razvoj kroz povijest te način na koji funkcionira. Također su opisana i govorna sučelja, njihove prednosti i komponente od kojih se sastoje. Osim toga, opisan je postupak izrade sustava za razvoj interaktivne fikcije koristeći programske okvire *Asp.Net Core* i *React*. Uz to je objašnjen način na koji se može u mobilnoj Android aplikaciji implementirati govorno sučelje za takav sustav koristeći *SpeechRecogniser* i *TextToSpeech* API.

Sustav kreiran kao dio ovog rada omogućuje korisnicima kreiranje jednostavnijih djela interaktivne fikcije. Jedno od mogućih unaprjeđenja bilo bi implementiranje složenijeg sustava za pisanje skripti u sustav. To bi korisniku omogućilo dodatnu kontrolu nad igrom. U ovakvu vrstu sustava bi idealno bilo dodati mogućnost vizualnog programiranja što bi korisnicima koji nemaju prethodnog iskustva s programiranjem olakšalo pisanje skripti. Osim sustava za pisanje skripti bilo bi korisno implementirati neke od opcija koje imaju slični sustavi. Primjer toga bila bi mogućnost dodavanja akcija koje se događaju nakon određenog broja poteza ili ulaska u sobe.

Što se tiče govornog sustava, u budućnosti bi se mogao koristiti *Web Speech API* kako bi se izbacila potreba za korištenjem posebne mobilne aplikacije. Međutim, u trenutku pisanja ovog rada, ta se opcija smatra eksperimentalnom te nije podržana u svim web preglednicima.

Popis literature

Bhandari, Bikash. »React Functional Components VS Class Components.« *Medium.* 12. veljača 2020. https://medium.com/wesionary-team/react-functional-components-vs-class-components-86a2d2821a22 (pokušaj pristupa 3. prosinac 2020).

Briceno, Hector, Wesley Chao, Andrew Glenn, Stanley Hu, Ashwin Krishnamurthy, i Bruce Tsuchida. *Down From the Top of Its Game: The Story of Infocom, Inc.* 15. prosinac 2000. http://web.mit.edu/6.933/www/Fall2000/infocom/infocom-paper.pdf (pokušaj pristupa 28. studeni 2020).

Gales, Mark, i Steve Young. »The application of hidden Markov models in speech recognition.« *Foundations and Trends® in Signal Processing*, 21. veljača 2008: 195-304.

Google. »Building web apps in WebView.« *Android Documentation.* 22. siječanj 2020. https://developer.android.com/guide/webapps/webview (pokušaj pristupa 5. prosinac 2020).

—. »RecognizerIntent.« *Android Documentation.* 30. rujan 2020. https://developer.android.com/reference/android/speech/RecognizerIntent (pokušaj pristupa 5. prosinac 2020).

Granade, Stephen. *Brass Lantern.* 2010. http://brasslantern.org/beginners/introif.html (pokušaj pristupa 26. studeni 2020).

Granade, Stephen, i Emily Short. »A Beginner's Guide to Interactive Fiction.« *Brass Lantern.* 2010. http://brasslantern.org/beginners/beginnersguide.html (pokušaj pristupa 27. studeni 2020).

Graves, Alex, i Navdeep Jaitly. »Towards end-to-end speech recognition with recurrent neural networks.« *International conference on machine learning*, 2014: 1764-1772.

Inform. »About Inform.« *Inform.* n.d. http://inform7.com/about/ (pokušaj pristupa 1. prosinac 2020).

Jurafsky, Dan, i James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition.* Prentice Hall, 2008.

Lebowitz, Josiah, i Chris Klug. *Interactive Storytelling for Video Games.* Amsterdam: Elsevier Inc., 2011.

Lock, Andrew. *ASP. NET Core in Action.* Manning Publications, 2018.

Maher, Jimmy. *Let's Tell a Story Together (A History of Interactive Fiction).* 2006. http://maher.filfre.net/if-book/index.html (pokušaj pristupa 27. studeni 2020).

Manning, Christopher D., Hinrich Schütze, i Prabhakar Raghavan. *Introduction to information retrieval.* Cambridge university press, 2008.

MariaDB. *Github.* n.d. https://github.com/MariaDB/server (pokušaj pristupa 4. prosinac 2020).

Microsoft. »Authentication and authorization for SPAs.« *Microsoft Docs.* 27. listopad 2020. https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity-api-authorization?view=aspnetcore-3.1 (pokušaj pristupa 4. prosinac 2020).

—. »Create web APIs with ASP.NET Core.« *Microsoft Docs.* 20. srpanj 2020. https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-3.1 (pokušaj pristupa 4. prosinac 2020).

—. »Entity Framework Core.« *Microsoft Docs.* 20. rujan 2020. https://docs.microsoft.com/en-us/ef/core/ (pokušaj pristupa 12. prosinac 2020).

Mikolov, Tomas, Kai Chen, Greg Corrado, i Jeffrey Dean. »Efficient estimation of word representations in vector space.« *arXiv preprint*, 2013.

Montfort, Nick. *Twisty Little Passages - An Approach to Interactive Fiction.* London: MIT Press, 2005.

Navarro, Gonzalo. »A guided tour to approximate string matching.« *ACM computing surveys (CSUR)*, 2001: 6-8.

neo4j. *Cosine Similarity.* n.d. https://neo4j.com/docs/graph-data-science/current/alpha-algorithms/cosine/ (pokušaj pristupa 30. studeni 2020).

Pearl, Cathy. *Designing voice user interfaces: principles of conversational experiences.* O'Reilly Media, Inc., 2016.

Pilehvar, Mohammad Teher, i Jose Camacho-Collados. »Embeddings in Natural Language Processing: Theory and Advances in Vector Representations of Meaning.« *Synthesis Lectures on Human Language Technologies* , 2020: 27-28.

Quest. *Quest Documentation.* n.d. https://docs.textadventures.co.uk/quest/ (pokušaj pristupa 1. prosinac 2020).

ReactJS. *React.* n.d. https://reactjs.org/ (pokušaj pristupa 3. prosinac 2020).

Reed, Aaron. *Creating interactive fiction with Inform 7.* Nelson Education, 2010.

Rivera, Alejandra. »What is the Voice User Interface (VUI)?« *UX Collective.* 13. siječanj 2020. https://uxdesign.cc/what-is-the-voice-user-interface-vui-786765463b28 (pokušaj pristupa 1. prosinac 2020).

Roberts, Michael J. *TADS - the Text Adventure development system, an Interactive Fiction authoring tool.* n.d. http://tads.org/ (pokušaj pristupa 27. studeni 2020).

Sciforce. »Text-to-Speech Synthesis: an Overview.« *Medium.* 13. veljača 2020. https://medium.com/sciforce/text-to-speech-synthesis-an-overview-641c18fcd35f (pokušaj pristupa 2. prosinac 2020).

Short, Emily. »NPC Conversation System.« U *IF Theory Reader*, autor Kevin Jackson-Mead i J. Robinson Wheeler, 331-358. Boston: Transcript On Press, 2011.

Taylor, Paul. *Text-to-speech synthesis.* Cambridge university press, 2009.

Weizenbaum , Joseph. »ELIZA - A Computer Program For the Study of Natural Language Communication Between Man and Machine.« *Communications of the ACM*, siječanj 1966: 36-45.

Popis slika

Popis tablica

Popis priloga

Uz ovaj rad priložena je datoteka programski_kod.zip koja sadrži cjelokupni programski kod za projekt koji je izrađen kao dio ovog rada.