# Implementacija tehnologija za pomoć slabovidnim osobama u programskom jeziku Python

**Barić, Fran**

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
Ak. god. 2022./2023.

Fran Barić

# Implementacija tehnologija za pomoć slabovidnim osobama u programskom jeziku Python

Završni rad

Mentor: doc. dr. sc. Ivan Dunđer

Zagreb, rujan 2023.

# Izjava o akademskoj čestitosti

Izjavljujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(potpis)

*Zahvaljujem se prijateljima, obitelji i mentoru.*

# Sadržaj

# 1. Uvod

Većina ljudi rođenih u digitalnom dobu uzima sposobnost korištenja računala zdravo za gotovo. Štoviše oni koji za to nisu sposobni često se smatraju čudnima, kao da nisu u toku s vremenom. Premda je kod nekih to uzrokovano načinom na koji su odgojeni, sredinom u kojoj su odrasli ili pak samo osobnom preferencijom, to nije slučaj kod svih. Nažalost, ljudi s određenim oblicima invaliditeta fizički su spriječeni koristiti računala na način na koji bi ih zdrava osoba koristila. Ipak, to ne znači da ih uopće ne mogu koristiti, jer su se s vremenom razvili brojni alati i pomagala za tu svrhu. Među njih se uvrštavaju i tehnologije prepoznavanja govora i njegove sinteze. One omogućavaju ljudima interakciju sa strojevima na način sličan onome između dvije osobe. Jedna od stvari koja se istražuje u ovom radu je učinak slabovidnosti i sljepoće na rad s računalima. Onima koji ne mogu vidjeti sadržaj prikazan na zaslonu niti pročitati slova na tipkovnici, prepoznavanje govora i sinteza govora predstavlja način da zaobiđu ograničenja svojih tijela i koriste se računalom isključivo uz pomoć govora i sluha. Ta zamisao primamljiva je i ljudima bez njihovih problema, ona predstavlja jedan nov, zabavan i intuitivan način za rad u ovom digitalnom dobu. Ovaj rad sastoji se od teorijskog i praktičnog dijela, u teorijskom dijelu analizira se kako se slabovidni i slijepi služe računalom te kakve probleme susreću pri njegovom korištenju. Zatim se analiziraju polja istraživanja prepoznavanja i sinteze govora, sažima se njihov povijesni razvoj, uzima se u obzir kako smo došli do njihovih suvremenih oblika i razmišlja se o njihovoj budućnosti. U praktičnom dijelu rada izrađen je program u programskom jeziku Python u kojem su se pokušale implementirati te dvije tehnologije. Riječ je o igri nalik potapanju brodova, koja bi se trebala moći igrati isključivo glasovnim naredbama. U samom radu pojasnit će se kako je ona isprogramirana, te će se na kraju sagledati koliko je bila uspješna u toj implementaciji uzimajući u obzir podatke dobivene od dobrovoljaca koji su pristali isprobati igru, ali i vlastitih dojmova.

# 2. Slabovidnost, pristupačnost i asistivne tehnologije

## 2.1. Slabovidni ljudi i računala

S ciljem razumijevanja kako problemi s vidom utječu na nečije bavljenje Pythonom, prvo se treba uzeti u obzir kako slabovidne osobe uopće koriste računala. Ono što im omogućuje njihovo korištenje jest pristupačna tehnologija, ali kako bi se razumjela korist i ponajprije važnost takve tehnologije potrebno je prvo steći shvaćanje o ljudima kojima su ti alati i namijenjeni (Grucza, 2022). Bolje razumijevanje njihovih okolnosti pomaže u stvaranju boljih i poboljšavanju postojećih rješenja (Lee, 2007). Trenutno u svijetu ima više od 2 milijarde slabovidnih ljudi, a vrijeme je kad su računala svakodnevna stvar, čak neophodna u većini područja života (WHO, 2022; Grucza, 2022). Da situacija bude gora računala su prvenstveno vizualni uređaji, ovisna o korisnikovoj mogućnosti prepoznavanja sadržaja na ekranu (Grucza, 2022). Shodno tome mislilo bi se da je gotovo nemoguće da slijepa osoba koristi računalo, ali to bi bio pogrešan zaključak. Naime, zahvaljujući tehnološkim napredcima slabovidne osobe koriste računala u sve većem broju (Grucza, 2022). Oštećenje vida može biti urođeno ili posljedica ozljede, ali gubitak vida uzrokovan starošću je nešto što nas sve čeka (Lee, 2007). Stoga pristupačna računala bit će svima od koristi jednog dana, a onima koji uopće ne mogu vidjeti ona predstavljaju priliku za svojevrsnu samostalnost omogućavajući im biti u toku s aktualnim događanjima, dobavljati informacije s interneta kao ostali, ali i jednostavno stupiti u kontakt sa suradnicima, prijateljima i obitelji (Grucza, 2022).

## 2.2. O asistivnim tehnologijama

Problemi s vidom mogu varirati od blagih poteškoća pri izvršavanju nekog zadatka do potpune nemogućnosti interakcije (Lee, 2007). Ovisno o stupnju manjka vida primjenjuju se različite metode pristupačnosti. Onima s blagim problemima s vidom pomaže se većim fontovima na ekranu, većim kontrastom slike ili pak posebnim kombinacijama boja korištenim za prikaz (Lee, 2007). Malo teži slučajevi još su donekle rješivi primjenom alata za uvećavanje teksta ili pak jednostavno zumiranjem (Lee, 2007). Složeniji su slučajevi gdje vida gotovo ili uopće nema, tu je potrebna upotreba takozvane asistivne tehnologije koja prilagođuje računalo korisnikovim sposobnostima i pomaže im u svakodnevnim radnjama (Arambašić i Dunđer, 2013; Lee, 2007). Asistivna tehnologija za slijepe je bilo koji hardver ili softver koji pomaže slijepim ili slabovidnim osobama koristiti uređaje koji konvencionalno zahtijevaju korištenje vida (Grucza, 2022). Najkorišteniji je možda čitač zaslona, riječ je o programu koji radi u pozadini i prati aktivnosti operativnog sustava, te pritom obavještava korisnika računalno sintetiziranim govorom ili fizički Brailleovim pismom, inače kolokvijalno zvanim i „brajicom” (Stack Overflow, 2009). Ono se sastoji od opipljivih točaka, te sadrži mehanizme koji omogućavaju navigaciju po sadržaju zaslona (Lee, 2007). Pomicanjem pokazivača ili prelistavanjem kroz stranicu točke se spuštaju i rastu kako bi se prikazao odgovarajući sadržaj (Grucza, 2022). Generalno prikazuje od 20 do 80 znakova odjednom (Stack Overflow, 2009). Ono je jedini alat koji je dostupan gluhoslijepim osobama (Lee, 2007). Brailleovo pismo se može koristiti zajedno s računalno sintetiziranim čitačem u situacijama kad zvučna poruka prenosi nešto teško razumljivo korisniku, poput grafikona ili tablica (Grucza, 2022). Čitači mogu pročitati cijelu web stranicu, samo dijelove nekog teksta ili čak samo zasebne riječi.

## 2.3. Pristupačnost kao značajka dobrog dizajna

Svi imaju svoje osobne preference i potrebe pri korištenju računala, ali iako suvremeni operativni sustavi i programi daju poprilično slobodu korisnicima po pitanju personalizacije i konfiguracije istih, ima još prostora za poboljšanja (Lee, 2007). Potrebno je pridati više pozornosti potrebama onih čija je svakodnevica itekako izazovnija zbog svojevrsnog oblika invaliditeta, a kojima konvencionalne opcije konfiguracije sustava i programa nisu dostatne. Prva stvar koja zanima slabovidnu osobu kad bira neki alat, ili u ovom slučaju preciznije računalo, jest je li jednostavno za uporabu, je li dostupna obuka za njegovo korištenje i je li kompatibilno s uređajima i alatima koje osoba već posjeduje (Grucza, 2022). Ova zadnja stavka je i najproblematičnija jer danas je većina korisničkih sučelja dizajnirana isključivo za tri računalne komponente: miš, tipkovnicu i zaslon (Lee, 2007). Zbog toga, programeri i dizajneri navikli su se raditi s njima, te trenutno propagiraju ovaj problem, uzimajući takav pristup zdravo za gotovo jer odgovara većini njihovih korisnika (Lee, 2007). Među njima, doduše, ima ljudi koji se ne mogu služiti tim komponentama, poput slijepih osoba. A u slučaju mobilnih uređaja neke od tih komponenti uopće ni nisu podržane (Lee, 2007). Pojavljuje se fenomen gdje su stvaratelji suvremenih sustava i aplikacija često prespori u davanju podrške za raznorazne nekonvencionalne alate i uređaje, kao što su primjerice sintetiziran govor i prikaz na „brajici” (Lee, 2007). Ali zato postoje asistivne tehnologije koje često služe kao posrednik između izvorno nepristupačne aplikacije i korisnika spomenutih nekonvencionalnih alata i uređaja (Lee, 2007). One to ostvaruju prateći i u nekim slučajevima utječući na rad spomenutih nepristupačnih softvera, a to je pak izvedivo uz pomoć takozvanih Application Programming Interface-ova (API) (Lee, 2007). API je definiran kao skup pravila koja omogućavaju različitim programima da međusobno razmjenjuju podatke i time stvaraju brojne mogućnosti za suradnju među developerima, odnosno firmama, i zajedničko pospješivanje (IBM, 2021). Razlog zašto asistivne tehnologije postižu tako dobre rezultate svodi se na jedinstven pristup u dizajnu programa odnosno arhitekture njegovih elemenata (Lee, 2007). Za asistivne tehnologije elementi korisničkog sučelja pojavljuju se u obliku stabla pristupačnih objekata koji uglavnom zrcale strukturu korisničkog sučelja (Lee, 2007). Ono je svojevrsni pseudo-API koji opisuje strukturu programa ili web stranice, nalik

je modelu objekata dokumenta(DOM) koji predstavlja interakciju objekata napisanih na označnim jezicima poput HTML-a i XML-a, ali ne mora biti podjednako informativan jer većina tih informacija služi samo za vizualnu reprezentaciju, također može imati manje čvorova (Boxhall, Kearney i Gash, 2016). Asistivne tehnologije onda navigiraju kroz to stablo, te se takoreći pretplaćuju na događaje kako bi mogle reagirati na bilo kakvu promjenu stanja aplikacije (Lee, 2007). Nusproizvod ovog pristupa arhitekturi je nastanak idealnog načina za testiranje ili automatiziranje aplikacija grafičkog korisničkog sučelja, što je inače teško izvesti (Lee, 2007).

## 2.4. Udruge za pomoć slabovidnima pri korištenju računala

Svima koji koriste računala trebala je pomoć barem par puta u procesu učenja korištenja i privikavanja na uređaj, a mnogi susreću probleme još i dugo nakon toga. To je univerzalna istina za sve koji vide, a pogotovo onda za one koji to ne mogu. Premda skoro svako računalo može biti prilagođeno za slabovidne, još uvijek moraju od nekoga naučiti kako se njime služiti, a i iako su neke asistivne tehnologije dosta povoljne ili čak besplatne, mnoge su još uvijek iznimno skupe za popriličan broj ljudi (Grucza, 2022). Zbog toga su se pojavile brojne udruge i zajednice koje pripomažu slijepima i slabovidnima od kojih ćemo nabrojati par utjecajnijih.

Prvo treba spomenuti NV Access ili NonVisual Access. To je neprofitna organizacija koja razvija i podržava razvoj besplatnih i programa otvorenog koda koji omogućavaju pristup tehnologiji slijepima i slabovidnima. Razvili su možda i najpopularniji čitač zaslona danas – NVDA (NonVisual Desktop Access), koji je prilagođen za više od 55 jezika volonterskim radom i koristi se u više od 175 država (Grucza, 2022).

Zatim postoji organizacija Computers for the Blind koja je isto neprofitna i koja si je za zadatak uzela dijeljenje povoljnih računala, asistivne tehnologije i obuke za korištenje istih (Grucza, 2022). Oni zaprimaju donirana računala koja onda čine pristupačnima i daju ih na korištenje slijepim i slabovidnim za djelić vrijednosti uređaja (Grucza, 2022).

U Hrvatskoj postoji Udruga slijepih koja brine o interesima slijepih i slabovidnih osoba u Zagrebu i okolici, te oni nude brojne tečajeve korištenja računala i asistivne tehnologije, pored brojnih drugih stavki kojima isto pomažu tim ljudima.

## 2.5. Slijepi developeri i Python

Što se tiče programiranja, što je jedna od točaka interesa u kontekstu ovog rada, postoji online zajednica PythonVis koja pomaže slijepim i slabovidnim korisnicima računala naučiti programski jezik Python bez ikakvog potrebnog predznanja (PythonVis, 2015). Zajednica je osmišljena kao jedan virtualni prostor gdje si svi mogu međusobno pomagati i zajedno učiti.

Unatoč pokušajima da se integrirano razvojno okruženje za učenje Pythona (IDLE) učini pristupačnim slijepim i slabovidnim osobama, tu nije postignut uspjeh. Naime, problem je u tome što IDLE prikazuje svoj output na način koji je teško prepoznatljiv čitačima zaslona, što onda developere prisiljava da output iz IDLE-a kopiraju u alat Notepad ili neki drugi uređivač teksta koji podržava njihov čitač zaslona (Lee, 2015; Stack Overflow, 2022). Stoga većina njih koriste druga razvojna okruženja koja nisu vezana za Python, a iako bi mogli koristiti jednostavan uređivač teksta poput Notepad-a, mnogi preferiraju moćnije primjerke kao što je EdSharp (Lee, 2015; PythonVis, 2015). Mnogi rade znatno brže koristeći čitače govora, ali se pokazalo da je u situacijama gdje je interpunkcija vrlo bitna, poput gniježđenja zagrada, efikasnije koristiti Brailleov prikaz (Stack Overflow, 2022).

## 2.6. Značajni projekti otvorenog koda u polju asistivne tehnologije

Prisutan je veliki interes za ovo polje, razvija se mnogo pomagala za slabovidne i slijepe, ali i ljude s drugim invaliditetima. Prije govora o projektu ovoga rada, vrlo je korisno sagledati što drugi ljudi rade, stoga će se nabrojati nekolicina zanimljivih projekata u polju asistivne tehnologije.

Orca je besplatan čitač zaslona pisan u Pythonu. On radi na programima koji podržavaju AT-SPI što je osnovna infrastruktura za asistivnu tehnologiju okruženja sličnim operativnom sustavu Unix, kao što su Linux i Solaris (GNOME Project, 2016). Korisnici Orce mogu koristiti aplikacije, uključujući programe s grafičkim korisničkim sučeljem pomoću kombinacije sintetiziranog govora, Brailleovog prikaza i alata za uvećavanje teksta (Lee, 2007).

Speech Dasher omogućuje unos teksta kombinacijom govora i gesta. Prednosti njegova korištenja trebale bi biti ponajprije brzina, istovremeno korištenje govora i gesta trebalo bi biti brže nego korištenje samo jednog od njih, a trebalo bi rezultirati i većom razinom točnosti (Vertanen, 2016). U suštini cijeli proces bi trebao biti dosta zabavniji od drugih dostupnih načina unosa teksta jer ne zahtijeva tipkovnicu, a može se i koristiti na mobilnim uređajima (Vertanen, 2016).

GOK je virtualna tipkovnica koja korisniku omogućuje unos teksta i kontrolu aplikacija pomoću sklopke, oblika alternativnog ulaznog uređaja u obliku gumba ili prekidača kojeg je potrebno samo lagano pritisnuti bilo gdje na površini nakon čega sklopka korisniku pruža taktilnu povratnu informaciju, izrađena je u programskom jeziku C, ali je razmatrana i migracija u Python (Lee, 2007).

FatBits je alat koji je u stanju uvećati neko područje na zaslonu i prikazati ga u novim prozoru u obliku velikih obojenih piksela, čija se boja može mijenjati, a tekst se može zagladiti (Ridges, 2016). FatBits je originalno dizajniran za umjetničke svrhe i za dizajnere korisničkih sučelja, ali se pokazao korisnim ljudima s problemima s vidom (Ridges, 2016).

CroSS je formantni sintetizator govora na hrvatskom jeziku koji tekstualni unos pretvara u govor s ciljem pomoganja osobama s jezičnim i govornim problemima, ali i onima s problemima pri čitanju (Dunđer, 2013).

# 3. Sinteza govora

## 3.1. Što je sinteza govora?

U ovom radu pridat će se malo više pozornosti konceptima sintetiziranog govora i prepoznavanju govora s obzirom na program o kojem će se govoriti nešto kasnije. Ponajprije, što li je uopće sintetiziran govor? Paul Taylor je u svojoj knjizi Text-to-Speech Synthesis za računalnu sintezu govora ili text-to speech (TTS) rekao da je polje koje se bavi trima stvarima: čitanjem, govorom i na kraju metodom kako natjerati računalo da napravi prijašnje dvije stvari (Taylor, 2009). Realizacija ovog zahtijeva iznimno složen sustav koji istovremeno mora biti sposoban za apstraktne lingvističke analize strukture diskursa i kodifikacije govora, što nam govori da je to polje od samog početka interdisciplinarno jer zahtijeva znanja iz područja inženjerstva, lingvistike, računalne znanosti, matematike, psihologije i akustike (Hirschberg et al., 1997). Uz to, za sintezu govora može se reći da je komplementarna drugim dvjema jezičnim tehnologijama, naime prepoznavanju govora, kojeg ćemo se dotaći odmah nakon sinteze govora, i strojnom prijevodu (Taylor, 2009). Razvoj TTS tehnologije popularno je područje istraživanja više od 60 godina, međutim stroj koji govori bio je jedan od čovjekovih snova stoljećima (Rabiner, 2009; Lemmetty, 1998). Krajnji cilj je napraviti sustave koji su u stanju imitirati ili potencijalno nadići ljudske mogućnosti razumijevanja i reproduciranja govora u međuljudskim i čovjek-stroj interakcijama (Rabiner, 2009). Kako bi se razumio način na koji suvremeni sustavi funkcioniraju napravit će se kratki povijesni pregled razvoja sinteze govora.

## 3.2. Razvoj sinteze govora

Prvi zabilježeni pokušaj sinteze govora dogodio se prije više od 200 godina u Sankt-Peterburgu (Lemmetty, 1998). Riječ je o stroju Kristijana Gottlieba Kratzensteina kojeg je napravio 1780. godine za natječaj sponzoriran od strane ondašnje akademije znanosti, a najvjerojatnije na prijedlog njegova poznanika Leonharda Eulera, s kojim je dijelio interes za fizikalnu prirodu čina govorenja (Story, 2019). Euler je 19 godina ranije u jednom od svojih radova napisao svojevrsni poziv na izradu stroja sposobnog artikulacije zvuka u korist ljudi čiji su glasovi pretihi ili neprijazni (Story, 2019). I to je skoro ono što je Kratzenstein napravio. Godinu prije predstavljanja stroja na akademiji objasnio je fiziološke razlike između 5 poznatih samoglasnika i onda konstruirao akustične rezonatore slične ljudskom govornom traktu koji su ih bili u stanju reproducirati (Lemmetty, 1998; Story, 2019).

Pukom slučajnošću u istom razdoblju mađarski inženjer Wolfgang von Kempelen osmislio je uređaj još složeniji od Kratzensteinovog (Story, 2019). Njegov stroj je također imitirao ljudsku anatomiju, ali je uspio reproducirati i samoglasnike i suglasnike, te njihovom kombinacijom čak riječi, ali još uvijek ne pune rečenice zbog tehničkih limitacija stroja (Story, 2019). Zanimljivo jest što je glas stroja navodno zvučao poput djeteta, što neki smatraju pokušajem izbjegavanja kritika, jer bi dječji glas bio manje kritiziran pri demonstraciji uređaja (Story, 2019).

Idući iskorak u dizajniranju stroja koji govori napravio je Robert Willis, profesor sveučilišta u Cambridgeu otkrivši poveznicu između zvuka zasebnog samoglasnika i geometrije govornog trakta 1838. (Lemmetty, 1998). On je bio zgrožen metodama prijašnjih pokušaja, stoga je pripremio posebne cijevi orgulja, te zaključio da se svi samoglasnici mogu stvoriti podešavanjem duljina tih cijevi i vibracija u zraku u njima (Story, 2019).

Willisovi eksperimenti u kojima je istraživao proces nastanka samoglasnika van konteksta ljudske anatomije, odnosno govornih organa započeli su brojna i dugotrajna istraživanja valnih oblika zvukova iz ljudskog govora (Story, 2019). Moglo bi se reći da su ta istraživanja kulminirala 1909. s uređajem Daytona Millera nazvanim *phonodeik*, pretečom osciloskopa (Story, 2019). Premda Millerov stroj nije proizvodio zvuk, on je bitan jer je za postupak kojim je radio Miller dao ime

harmonijska sinteza, što je očigledno utjecalo na budući razvoj ovog polja, jer je od te točke svaki sustav koji proizvodi umjetni govor, bez obzira na metodu kojom je govor postignut, smatran sintetizatorom govora (Story, 2019).

Sve što se spomenulo do sad dio je razdoblja mehaničkih i elektromehaničkih analognih sustava ili uređaja za sintezu govora, eksperimenti s tom tehnologijom rađeni su još do 1960-ih godina, ali bez postignuća vrijednih spomena (Lemmetty, 1998; Story, 2019). Tom dobu došao je kraj. Temeljna ideja koja je proizašla iz ovog doba jest da govor nastaje rezonancama koje pojačavaju ili ističu spektralne komponente nekog izvora zvuka, a kod ljudi one nastaju u govornom traktu (Arambašić, Dunđer i Seljan, 2013; Story, 2019). A ti djelići spektra su zahvaljujući Ludimaru Hermannu počeli biti zvani formantima, terminom koji će biti od velikog značaja u idućoj etapi razvoja računalne sinteze govora (Story, 2019).

Prvi potpuno električni sintetizator pojavio se 1922. kao rezultat rada mladog fizičara John Stewarta (Lemmetty, 1998). Riječ je o stroju koji je mogao izgovoriti samoglasnike s dva formanta, što uređaj tehnički ujedno čini prvim formantnim sintetizatorom, nije doduše mogao reproducirati konsonante (Lemmetty, 1998). Ipak, Stewart je uspio postići diftonge zahvaljujući praktičnim načinom upravljanja stroja – kotačićima i klizačima (Story, 2019). Treći formant su našli japanski istraživači Obata i Teshima 1932. što je bilo od velikog značaja jer se smatra da su prva tri formanta dovoljna za sintezu razumljiva govora (Lemmetty, 1998).

Otprilike u to vrijeme komunikacijski inženjer Homer Dudley istraživao je analognu proizvodnju zvuka i radio prijenosa (Story, 2019). Rezultat tog istraživanja je uređaj koji je bio u stanju sakupiti niskofrekvencijske zvučne signale na jednoj strani, prenijeti taj signal kroz niskopropusni kabel i reproducirati ga na drugoj lokaciji ili čak u drugom vremenu, tj. kasnije kada zatreba (Story, 2019). Uređaj je predstavljen 1936. godine kao VOCODER, skraćeno od Voice Coder, te je demonstrirao sintezu govora i pjeva, a Dudley je rekao da je i sposoban reprodukcije instrumentalne glazbe ili zvukova iz okruženja poput rada lokomotive (Story, 2019). Njegova prva namjena, doduše, nije bila zabava, već prenošenje poruka u drugom svjetskom ratu (Story, 2019). VOCODER je inspirirao razvoj VODER-a 1939., prvog pravog uređaja za sintezu govora (Lemmetty, 1998). Njegov naziv je skraćenica od Voice Operation Demonstrator i isto je djelo Homera Dudleya (Lemmetty, 1998). Voderom se upravljalo putem tipkovnice, poluga i pedala, što nije bio jednostavan posao (Story,

2019). Kako bi se njime služili ljudi su morali ići na obuku koja je trajala otprilike godinu dana (Story, 2019). Reprodukcija i samo jedne rečenice zahtijevala je dosta vještine od strane operatera i kvaliteta zvuka bila je daleko od današnje, ipak stroj je bio impresivan (Lemmetty, 1998). Njegova demonstracija zapalila je iskru znatiželje u znanstvenom svijetu (Lemmetty, 1998). U nadolazećim godinama dizajnirani su brojni sintetizatori govora od koji iskaču PAT i OVE, oba su formantni sintetizatori i po njima je polje formantne sinteze i ostalo zapamćeno (Story, 2019). Krajem 1950-ih i početkom 1960-ih pojavila se nova paradigma u sintezi govora – artikulacijski pristup sintezi (Story, 2019). Taj pristup je bio veoma obećavajući, jer se pretpostavljalo da su njegova pravila za proizvodnju zvuka bliža ljudskom govorniku, nego ona korištena za upravljanje akustičnih vrijednosti formantnih sustava (Story, 2019). Prvi pravi TTS uređaj pojavljuje se 1968. u Japanu kao rezultat rada Noriko Umede i njenih kolega (Lemmetty, 1998). On se temelji na maloprije spomenutom artikulacijskom modelu (Lemmetty, 1998). Govor tog uređaja bio je razumljiv, ali još uvijek previše monoton (Lemmetty, 1998). Paralelno s razvojem artikulacijskih sintetizatora, poboljšavali su se i formantni sustavi, primjeri te nove generacije poboljšanih formantnih sintetizatora bili bi MITalk i Klattalk TTS sustavi koji su činili temelj za kasnije sustave poput DECtalka koji se koristio još do ranih 2000-ih i popularizirali su TTS među širom publikom (Story, 2019). Jedan od ponuđenih glasova zvao se Perfect Paul i postao je vrlo poznat jer ga je koristio britanski znanstvenik Stephen Hawking (Story, 2019). Hawking je najvjerojatnije doprinio popularizaciji sintetiziranog govora više nego i jedna druga osoba (Taylor, 2009). Komercijalni sintetizatori govora i TTS uređaji počeli su se pojavljivati kasnih 70-ih i ranih 80-ih (Lemmetty, 1998). U modernija vremena mnoge vrste sinteze govora razvijene su s ciljem postizanja visokokvalitetnog automatiziranog zvučnog obavještavanja ili čak virtualnih asistenata koji bi bili u stanju komunicirati s korisnicima (Story, 2019). U zadnje vrijeme obećavajući rezultati u razvijanju sinteze govore postižu se s primjenom neuronskih mreža (Lemmetty, 1998). Primjerice pristup koji se zove direktno modeliranje valnog oblika koristi se dubokim neuronskim mrežama kako bi generirao nove zvučne signale koji se temelje na analiziranim značajkama snimljenog govora (Story, 2019). Dakle, najnoviji modeli sinteze govora u stanju su, uz pomoć algoritama dubokog učenja, analizirati intonaciju, visine tona i obrasce ljudskog izražavanja (Sabir, 2023). Time se postiže sintetizirani govor koji je puno prirodniji od onog starijih modela.

## 3.3. Primjene sintetiziranog govora

Kako bi se važnost ove tehnologije bolje shvatila, treba se postaviti pitanje koja je njena korist, ima li stvarnih uporabnih situacija za nju ili je samo za ukras. U samim začetcima, teško je za reći, oni koji su radili demonstracije prvih govorećih uređaja zasigurno su bili vođeni skoro dječjom radoznalošću i samo su htjeli vidjeti je li to uopće izvedivo, a pritom nisu nužno smatrali da rade nešto korisno. Ipak, u novije vrijeme situacija je drugačija, suvremeni TTS sustavi imaju bezbroj primjena, od kojih ćemo nabrojati nekoliko. Njihova prva, a vjerojatno i najbitnija primjena jest upravo za pomoć slijepim i slabovidnim osobama, kao što je već spomenuto ranije u radu (Arambašić i Dunđer, 2013; Lemmetty, 1998). Oni su ih objeručke prihvatili, pa čak još i u ranijim inačicama gdje je zvuk bio niže kvalitete, što u stvari i nije tako iznenađujuće jer Brailleov prikaz nije uvijek i svugdje primjenjiv i uporabljiv, a nisu mogli ni od ljudi s dobrim vidom očekivati da će im prenositi ono što vide cijelo vrijeme (Taylor, 2009). TTS čitači su im otvorili mnogo vrata, a time i dodatno osamostalili. Iako su isprva bili jako skupi te korišteni samo u knjižnicama i drugim javnim mjestima, danas su u obliku programa priuštivi skoro svima (Lemmetty, 1998). Kod takvih sustava bitno je da su razumljivi i barem donekle zvuče prirodno, doduše neki smatraju da ne bi trebali biti previše prirodni kako ne bi došlo do zabune po pitanju govori li stroj ili prava osoba (Lemmetty, 1998). Uz to, slijepe osobe ne mogu vidjeti koliko je velik tekst koji se čita, pa im TTS čitači unaprijed to izračunaju i prenesu (Lemmetty, 1998).

Osim slijepih, TTS koriste nijemi i oni koji su gluhi od rođenja kako bi komunicirali s drugima, a moguće ga je i koristiti pri telefonskim pozivima (Lemmetty, 1998).

Mora se priznati da je njegovo korištenje u široj publici dugo bilo dosta ograničeno samom kvalitetom govora, jer u pravilu zdrava osoba ne bi imala potrebu za njim (Taylor, 2009). Tek nedavno se počeo koristiti u pozivnim centrima, zdravstvu i bankama (Rabiner, 2007). Treba se istaknuti kako su brojna lingvistička istraživanja rađena uz pomoć TTS sustava, te zahvaljujući njima mnogo toga saznalo, ponajprije to da su brojne teorije bili pobijene (Taylor, 2009). Sintetizatori govora široko su korišteni i u industriji zabave od sporta, do glazbe i drugih umjetnosti (Dunđer i Seljan, 2013). Zasigurno jedna od novih primjena TTS-a je u internetskoj kulturi na

*streaming* servisima poput Twitch-a, gdje on omogućava publici da ima interakciju sa zabavljačima koje uživo prate na svojim zaslonima.

## 3.4. Sinteza govora u kontekstu rada

Pristup koji je odabran za projekt predstavljen u ovom radu je korpusni pristup sintezi govora. On se pojavio u 90-ima i donedavno je bio najpopularniji način sintetiziranja govora, jer se samo trebalo snimiti govor pravih osoba i konkatenirati individualne zvučne datoteke ovisno o potrebi (Watanabe et al., 2006). Kvaliteta zvuka je odlična, jer se radi o pravom ljudskom glasu, jedina mana pristupa jest da je dosta limitiran, te u složenijim projektima jednostavno nije isplativ s obzirom na potrebnu količinu zvučnih snimaka. Ipak, projekt predstavljen u ovom radu dovoljno je malog opsega da korištenje korpusne sinteze bude isplativo. Ipak vrijedi, istaknuti kako zbog osobnih, ali i tehničkih limitacija neće biti korišten glas prave osobe, već Google Cloud TTS koji je dovoljno kvalitetan da ispuni tu ulogu. Odabran je glas engleskog govornika iz dva razloga, ponajprije na engleskom modelu je najviše rađeno, pa i zvuči najprirodnije, a drugi razlog je taj što za sad još nema hrvatskih modela u ponudi, što je učestali problem kod manje govorenih jezika u svijetu zbog ograničenih podataka potrebnih za treniranje ovakvih modela (Dunđer i Seljan, 2014).

# 4. Prepoznavanje govora

## 4.1. Što je prepoznavanje govora?

Iduća tehnologija koja zahtjeva pozornost jest prepoznavanje govora. Ono se još naziva i automatskim prepoznavanjem govora (engl. *automatic speech recognition/*ASR) i speech-to-text-om (STT) (Karatas, 2023). Ovo polje proučavanja za cilj ima stvaranje sustava za preciznu identifikaciju i transkripciju različitih jezika, ali i njihovih naglasaka i dijalekata, uglavnom uz pomoć umjetne inteligencije i strojnog učenja (Karatas, 2023). Riječ je, u stvari o procesu suprotnom od računalne sinteze govora. Jedan od vodećih razloga za razvoj ove tehnologije je automatizacija pojedinih jednostavnih zadataka koji zahtijevaju interakciju između čovjeka i računala (Furui, 2005). Ne smiju se pomiješati prepoznavanje govora i prepoznavanje glasa, tj. radi se o različitim konceptima s različitim ciljevima (Karatas, 2023). Dok prvi proces utvrđuje sadržaj izrečene poruke, drugi utvrđuje identitet govornika (Karatas, 2023). Razvoj ove tehnologije ide unatrag nekih 70-ak godina i svjedoči mnogim inovacijama i generalno tehnološkim postignućima (Furui, 2005). Kako bi se u potpunosti razumjela tematika napravit će se opet jedan kratki povijesni pregled razvoja ASR-a.

## 4.2. Razvoj prepoznavanja govora

Početci u kreiranju sustava za prepoznavanje govora dogodili su se u 1950-ima i 1960-ima kad su se istraživači pozabavili osnovnim principima akustične fonetike (Furui, 2005). Službeno prvi uređaj koji je mogao prepoznati govor je „Audrey" koji su dizajnirali Davis, Biddulph i Balashek iz Bell Laboratorija, bio je veličine jedne cijele prostorije, a mogao je prepoznati brojeve od nula do devet (Summa Linguae, 2021). Radio je na principu prepoznavanja formantnih frekvencija izmjerenih pri izgovaranju samoglasničkih dijelova znamenaka (Juang i Rabiner, 2004). To je postizao s možda iznenađujućom razinom točnosti od 90%, ali ipak se pokazao nepraktičnim zbog svoje visoke cijene i otežane programibilnosti za više glasova (Summa Linguae, 2021). Prepoznavanje riječi je postalo moguće s IBM-ovim „Shoebox" sustavom 1962 (Summa Linguae, 2021). Do tada su svi projekti još bili na razini prepoznavanja zasebnih fonema, dobar primjer toga jest sustav koji su razvili japanski istraživači Sakai i Doshita (Furui, 2005; Juang i Rabiner, 2004; Summa Linguae, 2021). Shoebox je, s druge strane, bio u stanju razlikovati do 16 riječi što je bilo jako veliko postignuće tada, ipak njegova efikasnost je još bila dosta niska za današnje poimanje STT-a (Summa Linguae, 2021).

Početkom 1970-ih pojavio se prvi komercijalni ASR model zvan VIP-100 System, razvila ga je tvrtka Threshold Technology i dok nije bio korišten za išta vrijedno spomena, važan je jer je potaknuo američku vladinu organizaciju za napredne projekte(ARPA), a time i američko ministarstvo obrane da financira istraživanje i razvoj tehnologija za računalnu obradu jezika i prepoznavanje govora (Juang i Rabiner, 2004). Program se zvao SUR (engl. *Speech Understanding Research*) i trajao je 5 godina (Juang i Rabiner, 2004; Summa Linguae, 2021). Najznačajnije postignuće ovog programa jest sustav „Harpy" koji je bio u stanju prepoznati više od 1000 riječi s poprilično dobrom preciznošću (Summa Linguae, 2021). Harpy je to postigao novim pristupom prepoznavanja govora koji je podrazumijevao korištenje mreže sačinjene od leksičkih reprezentacija riječi sa sintaksnim pravilima produkcije i onima o granicama među riječima (Furui, 2005). Generalno rečeno, u 1970-ima sustavi sposobni prepoznavanja pojedinih riječi pokazali su se sasvim upotrebljivima, barem djelomice zahvaljujući radu japanskih i ruskih istraživača (Furui, 2005). Nimalo iznenađujuće, iduće desetljeće je kao fokus imalo razvoj sustava koji bi bio u

stanju prepoznati niz tečno izgovorenih riječi (Furui, 2005). Polako se napuštao pristup prepoznavanja govora na temelju obrazaca kao intuitivan pristup, u korist brojnih statističkih metoda (Juang i Rabiner, 2004). Prestalo se inzistirati na akustičnim principima kao temelju ASR modela, te su se stručnjaci okretali obradi prirodnog jezika i lingvistici kako bi napravili model koji ako susretne poteškoće pri prepoznavanju govora, može napraviti prosudbu uz pomoć sintaksnih, semantičkih i tonskih pravila nekog jezika (Summa Linguae, 2021). Najznačajniji model tog perioda jest „Hidden Markov model" (HMM), riječ je o dvostruko stohastičkom procesu koji uzima u obzir varijabilnost govora koji se prepoznaje i strukturu jezika kojim se govori kako bi na temelju brojnih statističkih podataka napravio prosudbu o tome što je najvjerojatnije bilo rečeno (Furui, 2005; Juang i Rabiner, 2004). Jednom kad je ova metodologija bila objavljena, počela se koristiti više manje u svim istraživačkim centrima koji su se bavili ASR-om, među njima bila je i bivša ARPA koja se preimenovala u DARPA (Juang i Rabiner, 2004). Oni su integrirali HMM-statističku metodologiju s funkcionalnošću svog Harpy modela, te je time nastao SPHYNX sustav (Furui, 2005). Još jedna tehnologija koja je pobudila interes u istraživačima 80-ih jest umjetna neuronska mreža, koja nije doduše bila sasvim nova tada, ali su bila stečena nova razumijevanja vezana uz njen odnos s intuitivnim prepoznavačkim sustavima, a iduća ideja bila je pokušati je primijeniti u kombinaciji s HMM-om (Juang i Rabiner, 2004). U 90-ima izuzev napretku pristupa prepoznavanja obrazaca, nije došlo do novih prekretnica, HMM je još uvijek bio jako zastupljen, a možda najkorišteniji ASR model tog doba bio je HTK sustav kojeg je napravio tim s Cambridge Sveučilišta, a bazirao se na HMM-u (Juang i Rabiner, 2004). Još neki sustavi vrijedni spomena su Dragon Dictate i njegov nasljednik NaturallySpeaking koji je vrlo efikasno prepoznavao prirodni govor, do tada su korisnici morali polako govoriti i raditi stanke pri govoru kako bi on bio prepoznat, ali s NaturallySpeaking sustavom za to nije bilo potrebe (Summa Linguae, 2021). Postignuća su pomalo stagnirala jer su tehnološke mogućnosti kaskale za brojnim inovacijama u polju, ASR sustavi su bili ograničeni procesorskom snagom i memorijom hardverskih komponenata (Summa Linguae, 2021).

U ranim 2000-ima DARPA je započela program EARS za razvoj sustava za poboljšano prepoznavanje prirodnog govora u radio i TV prijenosima na više stranih jezika, jer su dotadašnji modeli bili dosta uske primjene s obzirom na to da nisu bili

lokalizirani, niti globalizirani, te su ih korisnici s nesvakidašnjim naglascima jedva bili u stanju koristiti (Furui, 2005; Summa Linguae, 2021). Još jedan doprinos u poboljšavanju postojećih modela napravili su japanski istraživači s petogodišnjim projektom izrade korpusa spontanog govora koji se završetkom projekta sastojao od oko 700 sati govora, uz korpus razvili su nekolicinu novih fleksibilnih tehnika za ASR (Furui, 2005). U 2010-ima došlo je do prekretnice u razvoju pojavom glasovnog pretraživanja na Googleu na samom početku desetljeća, samo godinu nakon toga pojavio se Siri glasovni pomoćnik (engl. *voice assistant*) sa svojom sposobnošću ne samo precizno prepoznati glasovne naredbe, već i pružiti odgovor korisniku na razgovornom jeziku (Summa Linguae, 2021). Nedugo zatim počeli su se pojavljivati i drugi virtualni asistenti poput Alexe i Google asistenta, odjednom su svi počeli upotrebljavati ASR tehnologiju, a ne samo oni kojima je najpotrebnija, time je to cijelo desetljeće postalo kolokvijalno smatrano dobom virtualnih asistenata u kontekstu razvoja ASR-a.

## 4.3. Primjena prepoznavanja govora

ASR tehnologija se često koristi u obrazovanju, preciznije u računalno potpomognutom učenju (eng. CALL) i sustavima za e-učenje (Dunđer i Seljan, 2014). Danas možda najpoznatija primjena ASR-a je u virtualnim glasovnim pomoćnicima koje su već bili spomenuti, ali ono što nije bilo rečeno jest da oni ne dolaze samo u ovom općem oblika za širu publiku, već se primjenjuju u brojnim poljima kao specijalizirani pomoćnici spremni odgovoriti na određen tip pitanja ili barem korisnika preusmjeriti prema živućem agentu, tj. pravoj osobi (Karatas, 2023). Tako primjerice postoje prodajni i medicinski asistenti u obliku chatbotova s kojima se može glasovno komunicirati (Karatas, 2023). Takvi pomoćnici su korišteni kako bi automatizirali birokratske procese između pojedinaca i institucija ili korporacija, te efikasno uputili korisnike prema informacijama (Karatas, 2023). Ali, osim njih ASR koristi se i u pozivnim centrima za preusmjeravanje poziva, ali i pri prikupljanju podataka o korisnicima – često se korisnik obavijesti kako će razgovor biti sniman, te se onda snimljeni razgovor SST-om transkribira, te analizira za utvrđivanje korisnikova stava ili mišljenja prema nečemu (Karatas, 2023). Također se koristi pri glasovnoj navigaciji kod vožnje automobila, vozač može glasovno unijeti odredište i slično (Summa Linguae, 2021).

## 4.4. Prepoznavanje govora u kontekstu rada

U programu koji je predstavljen u ovom radu prepoznavanje govora izvršeno je pomoću Python biblioteka SpeechRecognition i PyAudio. SpeechRecognition biblioteka daje sve alate potrebne za prepoznavanje govora, a ponajprije funkcije i metode kojima se može odrediti izvor zvuka koji treba biti prepoznat, jezik govora koji se prepoznaje i sami model korišten za ASR. Ponuđeno je desetak modela uključujući Googleov, IBM-ov i OpenAI-ov, a za potrebe ovog projekta odabran je Googleov. Što se tiče jezika koji se prepoznaje, iako je hrvatski podržan, koristi se engleski jer je glas odabran za sintezu govora onaj engleskog govornika, pa je smislenije da se isti jezik koristi i za prepoznavanje. U vezi biblioteke PyAudio, ona je potrebna prvenstveno kako bi se mikrofon mogao koristiti kao ulazni uređaj, odnosno kao izvor zvuka u programu.

# 5. Implementacija u Python-u

## 5.1. O programu

Kao praktičan dio rada izrađena je pristupačna igra u Pythonu koja se može igrati govorom i sluhom, bez gledanja u zaslon. Radi se o igri nalik potapanju brodova, ali dosta pojednostavljenoj verziji koja koristi samo najmanja 3 broda i igra se na 8 puta 8 mreži koordinata kako igra ne bi trajala predugo. Igra se upravlja isključivo govorom, pa bi se moglo reći da se time imitira proces razgovora, ali između čovjeka i računala. Ponajprije treba opisati tok igre, s jedne strane kako bi ljudi koji nisu upoznati s potapanje brodova razumjeli što se događa, a s druge strane jer ona ipak nije izravna kopija igre potapanja brodova, a i ljudi često igraju igre prema takozvanim kućnim pravilima (engl. *home rules*). Nakon kraćeg opisa toka igre, s pitanja „što?" prelazi se na pitanje „kako?", te će se korak po korak objasniti kako je svaka mehanika igre postignuta.

## 5.2. Opis toka igre

Na samom početku TTS glas naratora poželi dobrodošlicu igraču, nakon čega slijede pravila igre koja se također izgovore. Igračevog suparnika zvat će se računalom ili protivnikom nadalje. Računalo u tajnosti bira pozicije svojih brodova, nakon čega je red na igraču. Narator obavještava igrača da izgovori prvu koordinatu svog razarača, nakon što igrač to napravi koordinata će biti ili zaprimljena ili odbijena. U oba slučaja igrač je obaviješten. Ako je koordinata odbijena znači da ili prepoznata riječ nije koordinata ili STT nije uspio prepoznati što je rečeno, u oba slučaja narator kaže igraču da pokuša ponovno. Kad se jedna koordinata uspješno zaprimi, prelazi se na iduću. Prvi brod je razarač i on je najmanji, sastoji se od dvije koordinate, zatim slijede podmornica i krstarica koje imaju po tri koordinate. Nakon što igrač uspješno odabere sve koordinate svojih brodova igra zapravo počinje. Igrač je prvi na potezu, narator ga traži za prvu metu, igrač je izgovara nakon čega saznaje ili da je pogodio ili da je promašio, u svakom slučaju nakon njega je red na računalu čiji potez narator prenosi igraču. Kada igrač nešto pogodi, ne saznaje što je pogodio dok ne potopi brod u pitanju, ali kada igrač bude pogođen, narator ga obavještava o kojem je brodu riječ kako igrač ne bi morao pamtiti koordinate i svojih brodova. U jednom momentu netko će potopiti zadnji brod i ta strana pobjeđuje.

## 5.3. Uvid u kod programa

Na samom početku, prije ikakvog drugog koda učitavaju se moduli potrebni za ispravan rad programa izjavom import (Slika 1). Moduli koji su korišteni su random, winsound i speech_recognition. Speech_recognition modul je dio SpeechRecognition biblioteke koju smo spominjali ranije, njime provodimo prepoznavanje govora u programu. Winsound je modul koji nam daje pristup funkcijama za reprodukciju zvučnih datoteka koje se nalaze u istoj mapi kao i py datoteka programa. Random modul nam daje pristup funkcijama za stvaranje pseudo-nasumičnih brojeva (engl. *pseudorandom*), te funkcije su potrebne kako bi računalo moglo igrati protiv igrača.

```
import random
import winsound
import speech_recognition
```

Slika 1. Učitani moduli

Nakon toga definirana je funkcija prepoznaj(), koja se temelji na funkciji Recognizer() iz speech_recognition modula, ona je realizirana kao while petlja u kojoj se kontinuirano prepoznaje govor dok se ne zadovolji kriterij duljine prepoznate riječi od 2 znaka. Ta petlja uz to filtrira i neuspješno prepoznate riječi, kao i prepoznate riječi krive duljine. U oba slučaja igraču se poručuje PlaySound() funkcijom, iz modula winsound, da pokuša ponovno (Slika 2).

```
def prepoznaj(recog):
    while True:
        try:
            with speech_recognition.Microphone() as mic:
                recog.adjust_for_ambient_noise(mic, duration=0.2)
                audio=recog.listen(mic)
                text=recog.recognize_google(audio, language='en-US')
                text=text.lower()
                print(f"Recognized {text}")
                if len(text)==2:
                    return text
                else:
                    print(text)
                    print('pokusaj ponovo')
                    winsound.PlaySound(".\\again.wav",winsound.SND_FILENAME)
                    continue
        except speech_recognition.UnknownValueError:
            winsound.PlaySound(".\\again.wav",winsound.SND_FILENAME)
            recog=speech_recognition.Recognizer()
            continue
```

Slika 2. Funkcija prepoznaj()

Iduća stvar koja se odvija u kodu jest dodjela koordinata brodovima računala. Svaki brod je inicijaliziran kao prazna lista, a zatim i samo polje na kojem bi se brodovi trebali nalaziti kao *field1*, ali u obliku rječnika sa 64 elemenata. Naime, svaki element predstavlja jednu koordinatu polja i to je odraženo u njihovim ključevima, dakle od a1 do h8, njihove vrijednosti za početak iznose 0 (Slika 3).

```
destroyer1=[]
submarine1=[]
cruiser1=[]
field1={'a1':0,'a2':0,'a3':0,'a4':0,'a5':0,'a6':0,'a7':0,'a8':0,
        'b1':0,'b2':0,'b3':0,'b4':0,'b5':0,'b6':0,'b7':0,'b8':0,
        'c1':0,'c2':0,'c3':0,'c4':0,'c5':0,'c6':0,'c7':0,'c8':0,
        'd1':0,'d2':0,'d3':0,'d4':0,'d5':0,'d6':0,'d7':0,'d8':0,
        'e1':0,'e2':0,'e3':0,'e4':0,'e5':0,'e6':0,'e7':0,'e8':0,
        'f1':0,'f2':0,'f3':0,'f4':0,'f5':0,'f6':0,'f7':0,'f8':0,
        'g1':0,'g2':0,'g3':0,'g4':0,'g5':0,'g6':0,'g7':0,'g8':0,
        'h1':0,'h2':0,'h3':0,'h4':0,'h5':0,'h6':0,'h7':0,'h8':0}
```

Slika 3. Igraće polje

Zatim je otvorena for petlja namještena da se iterira stotinjak puta argumentom funkcije range(), prethodno inicijaliziranom varijablom *counter1* pohranjuje se redni broj svake iteracije. Ovisno o broju iteracije aktivira se pojedina if naredba, na početku se dodjeljuje prva koordinata razaraču metodom append(). Do vrijednosti

koja se dodjeljuje dolazi se metodom choice() iz modula random koja kao argument u ovom slučaju uzima varijablu *field1* koja je pretvorena u listu zato što metoda choice() ne radi nad rječnicima. Sljedeća koordinata izvodi se iz prve u drugoj iteraciji, tako da se prije definiranom funkcijom split() prijašnja koordinata rastavi na znakove da bi se pak oni pohranili u listu. Zatim se novom ugniježđenom if naredbom provjerava broj u koordinati kako bi se ustvrdilo je li ona preblizu rubu igraćeg polja. Ako je, iduća koordinata razarača bit će u istom redu, ali jedno mjesto ulijevo, u suprotnom iduća koordinata se uvijek stavlja jedno mjesto udesno. To nije savršen pristup, jer znači da su protivnički brodovi uvijek smješteni vodoravno, ali to dosta pojednostavljuje algoritam kojim se pozicije određuju, a ne čini igru išta manje igrivom. Isti postupak se ponavlja i za druge brodove, s jednim dodatnim detaljem. S obzirom na to da su druga dva broda dulja za jedno mjesto, njihova treća koordinata se izvodi iz druge, ali u slučaju da je prva koordinata bila preblizu rubu, druga bi bila onda lijevo od nje, kako treća ne bi bila vraćena desno jer nije više preblizu rubu, koristimo prethodno inicijaliziranu varijablu *pull* koja određuje smjer izvođenja. Ako je vrijednost pulla „+” izvodi se udesno, a ako je „-” izvodi se ulijevo. Kad se za prvu koordinatu zaključi da je preblizu ruba *pull* se odmah pretvara u „-” i ostaje tako dok se ne dodjeli zadnja koordinata broda u pitanju, tada prelazi natrag u „+”. Kada se dodjele koordinate svim brodovima radi se provjera preklapa li se ijedna od njih, ako je to slučaj brišu se dane koordinate, vrijednost varijable *counter1* vraća se na nulu i cijeli proces se ponavlja (Slika 4).

```
def split(word):
    return [char for char in word]
counter1=0
cont=[]
pull="+"
for i in range(130):
    if counter1==0:
        destroyer1.append(random.choice(list(field1)))
        counter1+=1
    elif counter1==1:
        cont=split(destroyer1[0])
        if int(cont[1])<8 and pull=="+":
            cont[1]=str(int(cont[1])+1)
            destroyer1.append("".join(cont))
            counter1+=1
        else:
            cont[1]=str(int(cont[1])-1)
            destroyer1.append("".join(cont))
            counter1+=1
    elif counter1==2:
        submarine1.append(random.choice(list(field1)))
        counter1+=1
    elif counter1==3:
        cont=split(submarine1[0])
        if int(cont[1])<7 and pull=="+":
            cont[1]=str(int(cont[1])+1)
            submarine1.append("".join(cont))
            counter1+=1
        else:
            cont[1]=str(int(cont[1])-1)
            submarine1.append("".join(cont))
            counter1+=1
            pull="-"
```

Slika 4. Dodjeljivanje koordinata protivničkim brodovima

Nakon što je postavljanje protivnikovih brodova završeno, red je na igraču. Proces se vodi tako da on dobiva upute putem PlaySound() funkcije, a zatim se pojedinom igračevom brodu, koji je isto u obliku liste, pridodaje koordinata metodom append() koja za argument uzima ono što funkcija prepoznaj(), definirana na početku programa, prepozna. Poslije toga igrač se obavještava o koordinati koja je prepoznata kao svojevrsni oblik potvrde da ga je program razumio, a to se postiže konkatenacijom i indeksiranjem u argumentu funkcije PlaySound() kako bi se pročitala upravo ona koordinata koja je bila prepoznata. I tako igrač bira koordinate jednu po jednu dok ih nije sve odabrao (Slika 5).

```
winsound.PlaySound(".\\Upute.wav",winsound.SND_FILENAME)
destroyer2=[]
submarine2=[]
cruiser2=[]

print("Unesite prvu koordinatu svojeg razarača:")
winsound.PlaySound(".\\raza1.wav",winsound.SND_FILENAME)
winsound.PlaySound(".\\podsjetnik.wav",winsound.SND_FILENAME)
destroyer2.append(prepoznaj(recog))
winsound.PlaySound(".\\"+destroyer2[0]+".wav",winsound.SND_FILENAME)

print("Unesite drugu koordinatu svojeg razarača:")
winsound.PlaySound(".\\raza2.wav",winsound.SND_FILENAME)
destroyer2.append(prepoznaj(recog))
winsound.PlaySound(".\\"+destroyer2[1]+".wav",winsound.SND_FILENAME)
print(destroyer2)

print("Unesite prvu koordinatu svoje podmornice:")
winsound.PlaySound(".\\podmo1.wav",winsound.SND_FILENAME)
submarine2.append(prepoznaj(recog))
winsound.PlaySound(".\\"+submarine2[0]+".wav",winsound.SND_FILENAME)
```

Slika 5. Odabir koordinata igračevih brodova

Kada su obje strane odredile pozicije svojih brodova, igra zapravo počinje. Sav daljnji kod nalazi se u for petlji koja se iterira 130 puta, taj broj je odabran uzevši u obzir da polje jedne strane sadrži 64 koordinata, udvostručivši taj iznos i zaokruživši ga na okrugli broj dobije se 130. Tako da čak i ako se tijekom igre pogode sva polja, for petlja je dovoljno robusna za to. Broj iteracija se prati varijablom *count* koja se povećava za jedan svakom iteracijom, ako je njen iznos paran broj igračev je potez, a ako je neparan računalo je na potezu. Prvo će se objasniti blok koda koji predstavlja igračev potez. Prvo se igrač obavijesti PlaySound() funkcijom da odabere koordinatu koju želi pogoditi, zatim se ona prepoznaj() funkcijom pohranjuje u varijablu *meta* (Slika 6). Zatim se if i elif naredbama provjerava nalazi li se ta koordinata u listama koje predstavljaju brodove računala. Ako to nije slučaj, igrač bude kao i prije zvučno obaviješten o tome, te se *count* varijabla uveća za jedan i završava igračev potez, odnosno ta iteracija for petlje. Ali ako se koordinata nalazi u jednom od brodova, ona se remove() metodom miče iz liste u pitanju. Zatim se igrač obavijesti o pogotku na način kao i do sada, nadalje ako nakon micanja pogođene koordinate iz liste broda ona ostane prazna, igrač se obavijesti da je potopio protivnički brod. Na kraju iteracije provjerava se je li igrač potopio sve protivničke brodove, ako je, on bude obaviješten o tome i igra završava, a for petlja se prekida.

```
print("Igra može početi.")
count=0
hit=0
memo=""
posib=[]
for i in range(130):
    if count%2==0:
        winsound.PlaySound(".\\player.wav",winsound.SND_FILENAME)
        print("Unesite koordinatu koju želite pogoditi")
        winsound.PlaySound(".\\meta.wav",winsound.SND_FILENAME)
        meta=prepoznaj(recog)
        if meta in destroyer1:
            destroyer1.remove(meta)
            print("Pogodak")
            winsound.PlaySound(".\\pogodak.wav",winsound.SND_FILENAME)
            if destroyer1==[]:
                print("Potopili ste protivnički razarač!")
                winsound.PlaySound(".\\potopili_ste_razarac.wav",winsound.SND_FILENAME)
        elif meta in submarine1:
            submarine1.remove(meta)
            print("Pogodak")
            winsound.PlaySound(".\\pogodak.wav",winsound.SND_FILENAME)
            if submarine1==[]:
                print("Potopili ste protivničku podmornicu!")
                winsound.PlaySound(".\\potopili_ste_podmornicu.wav",winsound.SND_FILENAME)
        elif meta in cruiser1:
            cruiser1.remove(meta)
            print("Pogodak")
            winsound.PlaySound(".\\pogodak.wav",winsound.SND_FILENAME)
            if cruiser1==[]:
                print("Potopili ste protivničku krstaricu!")
                winsound.PlaySound(".\\potopili_ste_krstaricu.wav",winsound.SND_FILENAME)
        else:
            print("Promašaj")
            winsound.PlaySound(".\\promasaj.wav",winsound.SND_FILENAME)
        count+=1
        if destroyer1==[] and submarine1==[] and cruiser1==[]:
            print("Pobjedili ste")
            winsound.PlaySound(".\\pobjeda.wav",winsound.SND_FILENAME)
            break
```

Slika 6. Igračev potez

Kod bloka koda za protivnički potez priča je malo zamršenija jer se u stvari pokušava imitirati način na koji ljudi igraju potapanje brodova, dakle cilj nije da računalo isključivo nasumično bira koordinate, već mora donositi odluke uzevši u obzir svoje prijašnje poteze. Ponajprije se u varijablu *aiturn* pohranjuje nasumično odabrana koordinata metodom choice() kao i pri dodjeljivanju koordinata brodovima računala (Slika 7). Zatim se provjerava vrijednost varijable *hit* kojom se bilježi je li neki od igračevih brodova već pogođen, svakim pogotkom njena vrijednost povećava se za jedan, a u slučaju da brod bude potopljen vraća se na nulu.

```
else:
    winsound.PlaySound(".\\npc.wav",winsound.SND_FILENAME)
    aiturn=random.choice(list(field1))
    if hit>0:
```

Slika 7. Nasumičan odabir

Početkom igre njena vrijednost je nula, stoga se prelazi na ispitivanje nalazi li se nasumično odabrana koordinata u listi jednog od igračevih brodova uzastopnim elif naredbama. Ako to je slučaj, prvo se naredbom *del* ta koordinata briše iz rječnika *field1*, što miče mogućnost nasumičnog odabira iste koordinate u narednim iteracijama for petlje (Slika 8). Zatim se igrač obavijesti o odabranoj koordinati, te da mu je brod u pitanju pogođen, nakon čega se koordinata miče i iz liste tog broda metodom remove(). Dodatno, trenutna vrijednost *aiturn* varijable pohranjuje se u drugu *memo* varijablu, čija će svrha uskoro biti objašnjena. Na kraju se provjerava, je li tim pogotkom brod potopljen, tj. je li lista broda postala prazna, ako je igrača se obavještava o tome i *hit* se vraća na nulu.

```
elif aiturn in destroyer2:
    del field1[aiturn]
    print(aiturn)
    winsound.PlaySound(".\\"+aiturn+".wav",winsound.SND_FILENAME)
    destroyer2.remove(aiturn)
    print("Pogođeni ste")
    winsound.PlaySound(".\\Vas_je_razarac_pogodjen.wav",winsound.SND_FILENAME)
    hit+=1
    memo=aiturn
    if destroyer2==[]:
        print("Potopljen vam je razarač!")
        winsound.PlaySound(".\\Vas_je_razarac_potopljen.wav",winsound.SND_FILENAME)
        hit=0
```

Slika 8. Brod prvi put pogođen

S druge strane, ako nasumično odabrana koordinata nije u listama igračevih brodova, ona se samo briše iz rječnika i igrač se obavijesti da je promašen (Slika 9).

```
else:
    del field1[aiturn]
    print(aiturn)
    winsound.PlaySound(".\\"+aiturn+".wav",winsound.SND_FILENAME)
    print("Protivnik vas je promašio")
    winsound.PlaySound(".\\protivnik_promasio.wav",winsound.SND_FILENAME)
count+=1
```

Slika 9. Brod promašen

U slučaju da je vrijednost varijable *hit* veća od nule provjerava se iznosi li ona točno jedan, ako da koordinata pohranjena u varijabli *memo* rastavlja se na listu znakova pomoću split i ta lista se pohranjuje u varijablu *cont*. Zatim se radi kontrola if naredbama kako bi se uspostavilo je li koordinata jedna od rubnih koordinata, tj.

počinje li na „a” ili „h”, i završava li na „1” ili „8”. Ako počinje na „a”, drugim riječima ako je to vrijednost prvog elementa varijable *cont* provjerava se je li koordinata ispod nje već gađana i time maknuta iz rječnika *field1*. Ako nije dodaje se u varijablu *posib* koja je oblika liste i u njoj se pohranjuju koordinate oko one koja je u prijašnjoj iteraciji pogođena, dakle riječ je o potencijalnim metama, koordinatama na kojima bi bilo logično da se nalazi ostatak pronađenog broda. S druge strane, ako početna koordinata počinje na „h”, isto se radi za koordinatu iznad. Poredak tih slova utvrđuje se uz pomoć njihovih vrijednosti u ASCII tablici, ona na početku abecede niže su vrijednosti od onih na kraju, a to je relevantno jer se redovi polja na kojem se postavljaju brodovi redaju abecedno. Zatim se u oba slučaja početnih slova, provjerava je li broj u koordinati, odnosno vrijednost drugog elementa niza veći od 1 i manji od 8, ako je znači da se u listu *posib* mogu dodati i koordinate lijevo i desno od početne. Nakon toga provjerava se if naredbom iznosi li broj u početnoj koordinati „1” ili „8”. Ako iznosi „1”, provjerava se koordinata desno od početne, a ako iznosi „8” onda lijevo od početne. Ako se još nalaze u rječniku dodaju se u *posib* listu. Zatim ako u oba slučaja slova u početnoj koordinati nisu ni „a”, niti „h”, u *posib* dodaju se i koordinate iznad i ispod početne. Na kraju, ako početna koordinata ne sadrži ni jednu od tih rubnih vrijednosti u *posib* se dodaju sve četiri koordinate oko početne, ako još postoje u rječniku *field1* (Slika 10).

```
if hit>0:
    if hit==1 or posib==[]:
        cont=split(memo)
        if cont[0]=="a":
            if ''.join([chr(ord(cont[0])+1),cont[1]]) in field1:
                posib.append(''.join([chr(ord(cont[0])+1),cont[1]]))
            if int(cont[1])>1 and int(cont[1])<8:
                if ''.join([cont[0],str(int(cont[1])+1)]) in field1:
                    posib.append(''.join([cont[0],str(int(cont[1])+1)]))
                if ''.join([cont[0],str(int(cont[1])-1)]) in field1:
                    posib.append(''.join([cont[0],str(int(cont[1])-1)]))
        if cont[0]=="h":
            if ''.join([chr(ord(cont[0])-1),cont[1]]) in field1:
                posib.append(''.join([chr(ord(cont[0])-1),cont[1]]))
            if int(cont[1])>1 and int(cont[1])<8:
                if ''.join([cont[0],str(int(cont[1])+1)]) in field1:
                    posib.append(''.join([cont[0],str(int(cont[1])+1)]))
                if ''.join([cont[0],str(int(cont[1])-1)]) in field1:
                    posib.append(''.join([cont[0],str(int(cont[1])-1)]))
        if cont[1]=="1":
            if ''.join([cont[0],str(int(cont[1])+1)]) in field1:
                posib.append(''.join([cont[0],str(int(cont[1])+1)]))
            if cont[0]!="a" and cont[0]!="h":
                if ''.join([chr(ord(cont[0])+1),cont[1]]) in field1:
                    posib.append(''.join([chr(ord(cont[0])+1),cont[1]]))
                if ''.join([chr(ord(cont[0])-1),cont[1]]) in field1:
                    posib.append(''.join([chr(ord(cont[0])-1),cont[1]]))
        if cont[1]=="8":
            if ''.join([cont[0],str(int(cont[1])-1)]) in field1:
                posib.append(''.join([cont[0],str(int(cont[1])-1)]))
            if cont[0]!="a" and cont[0]!="h":
                if ''.join([chr(ord(cont[0])+1),cont[1]]) in field1:
                    posib.append(''.join([chr(ord(cont[0])+1),cont[1]]))
                if ''.join([chr(ord(cont[0])-1),cont[1]]) in field1:
                    posib.append(''.join([chr(ord(cont[0])-1),cont[1]]))
        if (cont[0] not in "ah") and (cont[1] not in "18"):
            if ''.join([chr(ord(cont[0])+1),cont[1]]) in field1:
                posib.append(''.join([chr(ord(cont[0])+1),cont[1]]))
            if ''.join([chr(ord(cont[0])-1),cont[1]]) in field1:
                posib.append(''.join([chr(ord(cont[0])-1),cont[1]]))
            if ''.join([cont[0],str(int(cont[1])+1)]) in field1:
                posib.append(''.join([cont[0],str(int(cont[1])+1)]))
            if ''.join([cont[0],str(int(cont[1])-1)]) in field1:
                posib.append(''.join([cont[0],str(int(cont[1])-1)]))
```

Slika 10. Kalkulacije iduće mete

Nakon određivanja elemenata *posib* varijable, u slučaju da je vrijednost varijable *hit* veća od nule, a varijabla *posib* ima još elemenata, meta koja će se gađati u toj iteraciji bira se nasumično, ali ne iz koordinata rječnika, već onih u varijabli posib. U pravilu, čak i ako se niz mogućih meta *posib* isprazni, a brod ne bude potopljen, postupak odabira novih izglednih meta se ponavlja ovaj puta s novom početnom koordinatom, odnosno s drugom vrijednosti varijable *memo*. Samo ako se ni tada

varijabla *posib* ne uspije napuniti novim elementima, računalo će, takoreći, odustati i gađa se nanovo nasumična koordinata od onih preostalih u rječniku *field1*, a vrijednost varijable *hit* vraća se na nulu (Slika 11).

```
if posib==[]:
    aiturn=random.choice(list(field1))
    hit=0
    print(aiturn)
    winsound.PlaySound(".\\"+aiturn+".wav",winsound.SND_FILENAME)
if posib!=[]:
    print("mogućnosti su:")
    print(posib)
    aiturn=random.choice(posib)
    print(aiturn)
    winsound.PlaySound(".\\"+aiturn+".wav",winsound.SND_FILENAME)
```

Slika 11. Niz mogućih meta

Zatim se provodi provjera je li gađana koordinata u listi jednog od igračevih brodova, kao što je i provođena dok je vrijednost varijable *hit* nula, ali s dva dodatna koraka. U slučaju pogotka, pogođena koordinata ne miče se samo iz rječnika, već i iz varijable *posib* koja se pritom pretvara u praznu listu ako je brod potopljen (Slika 12).

```
if aiturn in destroyer2:
    posib.remove(aiturn)
    del field1[aiturn]
    destroyer2.remove(aiturn)
    print("Pogođeni ste")
    winsound.PlaySound(".\\Vas_je_razarac_pogodjen.wav",winsound.SND_FILENAME)
    hit+=1
    memo=aiturn
    if destroyer2==[]:
        print("Potopljen vam je razarač!")
        winsound.PlaySound(".\\Vas_je_razarac_potopljen.wav",winsound.SND_FILENAME)
        hit=0
        posib=[]
```

Slika 12. Uzastopno gađanje već pogođenog broda

Na kraju poteza računala provjerava se jesu li svi igračevi brodovi potopljeni, ako jesu igrača se obavještava o porazu i for petlja se prekida (Slika 13).

```
if destroyer2==[] and submarine2==[] and cruiser2==[]:
    print("Game over")
    winsound.PlaySound(".\\poraz.wav",winsound.SND_FILENAME)
    break
```

Slika 13. Računalo pobjeđuje

## 5.4. Evaluacija programa

Kako bi se prikupili podatci o kvaliteti izrađene igre i implementacije STT-a i TTS-a u njoj, primljena je pomoć od dobrovoljaca s problemima s vidom. Glavne stavke koje su se testirale su pristupačnost, igrivost i užitak igranja, odnosno koliko je jednostavno igrati tu igru bez vida, koliko je općenito iskustvo igranja glatko i koliko zadovoljstva ono pruža. Što se tiče pristupačnosti ispitanici su bili u glavnom zadovoljni, igra bi ih obavijestila kad god bi nešto trebali napraviti, za naratora su rekli da je sasvim razumljiv i mogli su sve odluke u igri prenijeti glasovno. Problem leži u tome što pri uključivanju mikrofona dolazi do kašnjenja od par sekunda, tako da igrači zapravo ne znaju kada se točno mikrofon pali, te često počnu govoriti prerano što dovodi do neuspješnog prepoznavanja. Što se igrivosti tiče, zaključak je bio da je model za prepoznavanje govora problematičan jer neke koordinate otežano prepoznaje dok druge uopće ne prepoznaje, te zbog toga igra traje dosta dugo zbog potrebe neprestanog ponavljanja. Konačno, užitak i zbog prijašnjih razloga nije baš bio jako visok, ali se uz to pokazalo da je igru samu po sebi teško pratiti s isključivo zvučnim podatcima, jer u suštini, bez pristupa prikazu igraćeg polja, ona od igrača traži da pamti sve koordinate koje je već gađao što se pokazalo dosta izazovnim i napornim za neke od ispitanika. Ovaj problem je zapravo već i predviđen, te je zbog toga smanjena veličina igraćeg polja s 10x10 na 8x8 i broj brodova s pet na tri, kako bi igra kraće trajala i ne bi bilo toliko koordinata za pamtiti, ipak tehnički problemi čine igru puno duljom i napornijom nego bi trebala biti.

# 6. Zaključak

Nemaju svi ljudi na svijetu iste mogućnosti korištenja računala, neki od njih to ne mogu zbog određene vrste invaliditeta. Kod ljudi s problemima s vidom, to je posebice izraženo jer se suvremena računala većinski dizajniraju za ljude koji mogu vidjeti. Ipak, slijepi i slabovidni mogu se služiti asistivnim tehnologijama kako bi koristili računala barem donekle poput ljudi koji mogu vidjeti. Pomoću toga se upuštaju i u programiranje. Premda Pythonovo razvojno okruženje nije jako pristupačno, u njemu se stvaraju brojni alati za slabovidne i slijepe. Među asistivnom tehnologijom koja se koristi danas, učestalo je korištenje računalne sinteze govora i automatskog prepoznavanja govora, bilo da je riječ o klasičnim čitačima zaslona ili virtualnim asistentima. Jedan od načina na koji se te dvije tehnologije mogu se koristiti u Pythonu, jest uz pomoć speech_recognition i winsound modula, a to se u radu i pokazalo u izrađenom programu. Implementacija doduše nije prošla savršeno zbog problema s prepoznavanjem govora. Razlog za to potencijalno ovisi o više faktora, ponajprije treba uzeti u obzir da je korišteni model javan i besplatan, što znači da bi u trenutcima moglo doći do opterećenja zbog prekomjerne upotrebe, te shodno tome lošijih rezultata. Sljedeći faktor koji se treba uzeti u obzir jest stabilnost i kvaliteta internetske veze, naime Google-ov API ne radi bez nje, a time ni sam program. Također treba razmišljati o mikrofonu koji se koristi, govor u kvalitetan mikrofon i u kvalitetnom okruženju za snimanje govora, drugim riječima bez pozadinskih smetnji trebao bi dati i bolje rezultate pri prepoznavanju govora. Na kraju, treba uzeti i ljudski faktor u obzir, ponajprije jezik kojem se govori, ali i način na koji se govori. U ovom projektu prepoznavao se engleski jezik američkog izgovora, jer je pretpostavka da je za njega model najbolje istreniran. Ispitanici koji su testirali program nisu izvorni govornici engleskog, pa je moguće da je i tu došlo do nekog problema. Ipak, najizglednije jest da je razlog za lošu kvalitetu prepoznavanja govora kombinacija svih navedenih faktora. Unatoč osrednjim rezultatima, moglo bi se reći da je postignut sam cilj implementacije TTS-a i STT-a u Pythonu i to dokazuje da nije tako teško interaktivan program, u ovom slučaju igru, učiniti pristupačnim slabovidnim i slijepim ljudima.

# Literatura


1. Arambašić, M., Dunđer, I. (2013) *Computer-based Assistive Technologies in Education for Students with Disabilities.* The Future of Information Sciences: INFuture2013 - Information Governance. Zagreb: Sveučilište u Zagrebu, Filozofski fakultet, str. 237-247.
2. Arambašić, M., Dunđer, I., Seljan, S. (2013) *Domain-Specific Evaluation of Croatian Speech Synthesis in CALL.* 7th European Computing Conference (ECC '13) - Recent Advances in Information Science (Recent Advances in Computer Engineering Series 13) / Language and Text Processing, str. 142-147.
3. Boxhall, A., Kearney, M., Gash, D. (2016) *The Accessibility Tree*. Web.dev. Dostupno na: https://web.dev/the-accessibility-tree/ [10. lipnja 2023.]
4. Dunđer, I. (2013) *CroSS: Croatian Speech Synthesizer - design and implementation.* 16th International Multiconference INFORMATION SOCIETY - IS 2013 / Collaboration, Software and Services in Information Society (CSS'2013), str. 257-260.
5. Dunđer, I., Seljan, S. (2013) *Automatic Word-Level Evaluation and Error Analysis of Formant Speech Synthesis for Croatian.* 4th European Conference of Computer Science (ECCS '13) - Recent Advances in Information Science (Recent Advances in Computer Engineering Series 17) / Image, Speech and Signal Processing, str. 172-178.
6. Dunđer, I., Seljan, S. (2014) *Combined Automatic Speech Recognition and Machine Translation in Business Correspondence Domain for English-Croatian.* International Conference on Embedded Systems and Intelligent Technology (ICESIT 2014). International Journal of Computer, Information, Systems and Control Engineering-International Science Index (1307-6892), 8(11), str. 1069-1075.
7. Furui, S. (2005) *50 Years of Progress in Speech and Speaker Recognition Research*. ECTI Transactions on Computer and Information Technology, 1(2), 64-68.

8. GNOME Project (2016) *Orca*. Open Assistive. Dostupno na: https://openassistive.org/item/orca/ [10. lipnja 2023.]
9. Grucza, A. (2022) *What to Know About Computers for the Visually Impaired*. WebMD. Dostupno na: https://www.webmd.com/eye-health/what-to-know-computers-visually-impaired [10. lipnja 2023.]
10. Hirschberg, J., Olive, J. P., Sproat, R. W., Van Santen, J. P. H. (1997) *Preface: Progress in speech synthesis* [online]. New York: Springer. Dostupno na: https://archive.org/details/progressinspeech0000unse/page/n9/mode/2up [12. lipnja 2023.]
11. IBM: *What is an API?* (2021) Dostupno na: https://www.ibm.com/topics/api [10. lipnja 2023.]
12. Juang, B. H., Rabiner, L. R. (2004) *Automatic Speech Recognition – A Brief History of the Technology Development*. Dostupno na: https://web.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/354_LALI-ASRHistory-final-10-8.pdf [14. lipnja 2023.]
13. Karatas, G. (2023) *Speech Recognition: Everything You Need to Know in 2023*. AIMultiple. Dostupno na: https://research.aimultiple.com/speech-recognition/ [14. lipnja 2023.]
14. Lee, J. (2015) *Writing a python editor for blind developers*. CompLangPython. Dostupno na: https://comp.lang.python.narkive.com/yLosmZtw/writing-a-python-editor-for-blind-developers [10. lipnja 2023.]
15. Lee, S. (2007) *Python Powered Accessibility*. GNOME. Dostupno na: https://wiki.gnome.org/Accessibility/PythonPoweredAccessibility [10. lipnja 2023.]
16. Lemmetty, S. (1998) *Review of Speech Synthesis Technology*. Diplomski rad. Helsinki: Sveučilište Aalto, laboratorij za akustiku i obradu zvučnih signala. Dostupno na: http://research.spa.aalto.fi/publications/theses/lemmetty_mst/ [12. lipnja 2023.]
17. PythonVis: *Recommended Text Editor* (2015) Dostupno na: https://www.freelists.org/webpage/pythonvis [10. lipnja 2023.]
18. Rabiner, L. (2009) *Preface: Text-to-Speech Synthesis*. Cambridge: Cambridge University Press

19. Ridges, J. (2016) *FatBits*. Open Assistive. Dostupno na: https://openassistive.org/item/fatbits/ [10. lipnja 2023.]

20. Sabir, A. (2023) *The Future of Text-to-Speech Technology*. Fliki. Dostupno na: https://fliki.ai/blog/future-text-to-speech [12. lipnja 2023.]

21. Stack Overflow: *How can you program if you're blind?* (2009) Dostupno na: https://stackoverflow.com/questions/118984/how-can-you-program-if-youre-blind [10. lipnja 2023.]

22. Stack Overflow: *Python's IDLE support for the blind is lacking - Am I missing something?* (2022) Dostupno na: https://stackoverflow.com/questions/73887177/pythons-idle-support-for-the-blind-is-lacking-am-i-missing-something [10. lipnja 2023.]

23. Story, B. H. (2019) *History of speech synthesis*[online]. Tucson. Dostupno na: https://www.researchgate.net/publication/342693675_History_of_speech_synthesis [12. lipnja 2023.]

24. Summa Linguae: *Speech Recognition Software: Past, Present, and Future* (2021) Dostupno na: https://summalinguae.com/language-technology/speech-recognition-software-history-future/ [14. lipnja 2023.]

25. Taylor, P. (2009) *Text-to-Speech Synthesis*. Cambridge: Cambridge University Press

26. Vertanen, K. (2016) *Speech Dasher*. Open Assistive. Dostupno na: https://openassistive.org/item/speechdasher/ [10. lipnja 2023.]

27. Watanabe, S., Iwaki, T., Kaneyasu, T., Miki, K. (2006) *Corpus-Based Text-to-Speech and Its Application*. Oki Technical Review, 73(2), 62-65. Dostupno na: https://www.oki.com/en/otr/2006/n206/pdf/otr-206-R21.pdf [12. lipnja 2023.]

28. WHO (World Health Organization): *Blindness and vision impairment* (2022) Dostupno na: https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment#:~:text=Globally%2C%20at%20least%202.2%20billion,has%20yet%20to%20be%20addressed [10. lipnja 2023.]

# Popis slika

# Implementacija tehnologija za pomoć slabovidnim osobama u programskom jeziku Python

## Sažetak

U ovom radu proučeno je kako napraviti da Python program bude pristupačan osobama s oštećenjem vida kroz implementaciju tehnologija poput računalnog prepoznavanja govora i računalne sinteze govora u samom kodu programa. Ponajprije je utvrđeno koje su mogućnosti trenutno dostupne slabovidnim i slijepim osobama pri korištenju Pythona i programa izrađenih u Pythonu. Zatim se definiralo računalno prepoznavanja govora, računalna sinteza govora i druge tehnologije odabrane za istraživanje. U idućem koraku je demonstrirana implementacija odabranih tehnologija na primjeru igre u stilu potapanja brodova kroz opis postupka i priložene snimke zaslona. Nakon toga su predstavljeni i analizirani rezultati evaluacije programa od strane slabovidnih i slijepih ljudi koji su bili voljni sudjelovati u istraživanju. U zaključku je rezimirano sve što se predstavilo u ovom radu zajedno s osobnim osvrtom na odabranu temu.

**Ključne riječi:** Python, programiranje, oštećenje vida, prepoznavanje govora, sinteza govora

# Implementation of technologies to assist the visually impaired in the Python programming language

## Summary

This paper has studied how to make a Python program accessible to visually impaired people through the implementation of technologies such as computer speech recognition and computer speech synthesis in the program code itself. First of all, it was determined what possibilities are currently available to visually impaired and blind people when using Python and programs created in Python. Then computer speech recognition, computer speech synthesis and other technologies selected for research were defined. In the next step, the implementation of selected technologies was demonstrated on the example of a game in the style of sinking ships through a description of the procedure and attached screenshots. After that, the results of the evaluation of the program by visually impaired and blind people who were willing to participate in the research were presented and analyzed. In the conclusion, everything presented in this thesis is summarized together with a personal review of the selected topic.

**Key words:** Python, programming, visual impairment, speech recognition, speech synthesis