Implementacija inteligentnog agenta koji igra MOBA igru

Skeledžija, Boris

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:976816

Rights / Prava: Attribution-NoDerivs 3.0 Unported/Imenovanje-Bez prerada 3.0

Download date / Datum preuzimanja: **2024-10-13**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Boris Skeledžija

IMPLEMENTACIJA INTELIGENTNOG AGENTA KOJI IGRA MOBA IGRU

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Boris Skeledžija

Matični broj: 0016140129

Studij: Informacijski sustavi

IMPLEMENTACIJA INTELIGENTNOG AGENTA KOJI IGRA MOBA IGRU

ZAVRŠNI RAD

Mentor:

izv. prof. dr. sc. Markus Schatten

Varaždin, srpanj 2022.

Boris Skeledžija

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovim radom će čitatelj dobiti uvid kako funkcionira umjetna inteligencija, gdje je prikazana pomoću inteligentnog agenta koji igra računalnu igru. Na samom početku rada su opisane metode i tehnike rada te alati koji su korišteni u izradi ovog rada, s tim da su Python i biblioteka namijenjena za računalni vid detaljno opisani. Nakon toga će se objasniti umjetna inteligencija te strojno učenje kao dijelom istoga te gdje su sve primjene umjetne inteligencije. Najvažniji dio rada, praktični dio, gdje će se razraditi proces izrade inteligentnog agenta koji igra računalnu igru. Tu će biti predstavljene ideje i načini na koji je ovaj agent implementiran i sama suština toga.

Ključne riječi: Python, agent, računalni vid, umjetna inteligencija

SADRŽAJ

1. Uvod

U ovome radu objasnit će se izrada inteligentnog agenta koji igra MOBA [16] (engl. Multiplayer Online Battle Arena) igru, koristeći Python te njegove razne biblioteke koje čija je svrha olakšati rad s umjetnom inteligencijom. Prvo će biti opisane tehnike i metode rada te koje su sve biblioteke korištene u ovom radu, uključujući biblioteku OpenCV [14] (engl. Open-source Computer Vision), namijenjenu za ovakve teme. Potom će se vidjeti što je to uopće umjetna inteligencija te ćemo vidjeti kako se ona zapravo razvijala kroz povijest s jednim od važnijih događaja kada je IBM-ov Deep Blue [15] pobijedio tadašnjeg svjetskog prvaka u šahu pod nazivom Garry Kasparov. U tom poglavlju isto će se vidjeti primjena i bitnost umjetne inteligencije u današnjem svijetu. Naposljetku, objasnit će se kako je strojno učenje podskup umjetne inteligencije i vezu između ta dva pojma. Na kraju se dolazi do praktičnog dijela, biti ovog rada, gdje će se implementirati inteligentni agent i vidjeti rad umjetne inteligencije za izvršenje zadatka koji je u ovom slučaju igranje računalne igre League of Legends (u nastavku LoL).

Kroz ovaj rad će se vidjeti na koji način se može napraviti inteligenti agent sposoban da samostalno igra računalnu igru. Razlog tomu je kako dolaze novi igrači koji žele naučiti igrati ovu računalnu igru, ovi agenti im mogu biti lakši protivnici koje mogu savladati te tako poboljšati svoje sposobnosti kako bi kasnije mogli igrati u tzv. PvP (engl. Player versus Player) okruženju.

2. Metode i tehnike rada

2.1 Uvod u korištene tehnike i alate

Za razvoj inteligentnog agenta koji igra MOBA igru League of Legends korišten je programski jezik Python te njegove razne biblioteke. Uz to, korišten je i alat za razvojno okruženje Visual Studio Code koji je optimiziran za stvaranje modernih aplikacija. U ovom poglavlju opisat će se korištene tehnologije za izradu ovog rada.

2.2 Programski jezik Python

Prva službena inačica programskog jezika Python je izašla 1991. godine, a izumitelj Pythona je Guido Van Rossum. Idemo opisati bitne značajke Pythona:

- **Objektno orijentiran** – sve u Pythonu je objekt, a objekt je način reprezentiranja podataka i veza među njima.
- **Interpreterski jezik** – Python čita kod i evaluira liniju po liniju i pretvara kod u strojni jezik tijekom izvođenja.
- **Dinamički tipiran** – ovo znači da su objekti u memoriji tijekom izvođenja koda te nije potrebno deklarirati koji tip podataka koristimo prije izvođenja samog programa, za razliku od statički tipiranih jezika poput C++ i Java.

Pošto je jako popularan jezik, Python ima razne biblioteke koje pomažu u ostvarenju različitih ciljeva. Python se koristi se za automatizaciju zadataka, programiranja za Web, analizu podataka te umjetnu inteligenciju i prema tome se pokazuje kao dobar programski jezik za ostvarenje cilja ovog rada.

2.3 OpenCV biblioteka

OpenCV biblioteka po samom imenu nam kaže, napravljena je da rješava probleme računalnog vida. Primarno se fokusira na procesiranje slika, analiziranje videa i detektiranja lica ili različitih oblika. Računalni vid se može definirati kao disciplina koja opisuje kako rekonstruirati i razumjeti 3D scenu iz njenih 2D slika pomoću svojstava strukture koja je prisutna u sceni. [14] Ukratko, bavi se modeliranjem i repliciranjem ljudskog vida koristeći računalni hardware i software. Ovo sve omogućava postizanje cilja izrade ovog rada, igrajući ulogu u detektiranju različitih oblika na ekranu, uspoređivanje tih oblika s uzorcima te postoji granica koja se može prikazati kao postotak koji označava koliko se neki oblik podudara s

uzorcima te u rezultate ulaze oblici čiji je postotak veći od granice i potom poduzeti određene radnje.

Za primjer ove biblioteke može se uzeti tzv. „Podudaranje predloška“ (engl. Template Matching) koje spada u kategoriju procesiranja slike (engl. Image Processing). Primjer je uzet sa [21]. Podudaranje predloška se može objasniti kao traženje neke slike (predloška) u nekoj većoj slici tj. traženje lokacije tog predloška. Za to se koristi punkcija matchTemplate() koja za potrebne parametre prima veću sliku u kojoj se objekt/predložak traži, sama slika predloška te metoda kojom će se gledati podudarnost.

Za primjer predloška je slika ispod:

Slika 1: Lice nogometaša: Lionel Messi (Izvor: [21])

Postoje različite metode traženja podudarnosti gdje svaka ima različitu matematičku formulu kojom se računa, ali na ovom primjeru pogledat ćemo metodu TM_CCOEFF.
Korištenje funkcija matchTemplate (cv označava biblioteku OpenCV):

```
rezultat = cv.matchTemplate(slika,predlozak,metoda)
```

Predložak „prođe“ preko slike te gleda podudarnost grupe piksela slike koja je „ispod“ predloška. Rezultat je slika u sivim tonovima gdje svaki piksel označava koliko se podudara piksel blizak njemu sa predloškom. Kako bi našli gdje su maksimalne i minimalne vrijednosti tih podudarnosti, koristi se funkcija minMaxLoc().

```
min_vrijednost,maks_vrijednost,min_lokacija,maks_lokacija=cv.minMaxLoc(rezu
ltat)
```

Pošto koristimo TM_CCOEFF metodu, gornja lijeva točka je maks_lokacija (to je tuple koji sadrži koordinate piksela (x, y)) koja je „početak“ pravokutnika koji je tzv. regija našeg predloška pronađenog u slici, a donja desna točka se dobije zbrajajući širinu predloška sa x koordinatom dobivenom iz gornje lijeve točke te zbrajanjem y koordinate sa visinom slike predloška.

```
      gornja_lijeva_tocka = maks_lokacija
      donja_desna_tocka=(donja_lijeva_tocka[0]+širina,donja_lijeva_tocka[1]
+ visina)
```

Onda se taj pravokutnik može nacrtati preko funkcije rectangle() koja uz sliku i točke za parametre još i prima boju pravokutnika te debljinu linije:

```
cv.rectangle(slika,gornja_lijeva_tocka, donja_desna_tocka, 255, 2)
```

Rezultati se vide na slici ispod:

Slika 2: Rezultati pronalaska predloška u slici (Izvor: [21])

2.4 Ostale korištene biblioteke

U ovom poglavlju će se opisati ukratko ostale korištene biblioteke i alati koji su pridonijeli izvršavanju cilja ovog rada:

- **Pywin32** – omogućava nam interakciju s COM (engl. Component Object Model) objektima i automatizaciju Windows aplikacija s Pythonom. Bit ovoga je da možemo uraditi s Pythonom sve što Microsoft aplikacija može. Referenca na što sve ova biblioteka može je na [23]. Za primjer se može uzeti slučaj korištenja kontrola miša:

```
pozicija = (500, 500)
win32api.SetCursorPos(pozicija)
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN,
pozicija[0], pozicija[1], 0, 0)
win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,
pozicija[0], pozicija[1], 0, 0)
```

- **Pydirectinput** – ova biblioteka je recimo preslika PyAutoGUI biblioteke, ali ona koristi DirectInput sken kodove i modernije SendInput() win32 funkcije. Lakše rečeno, njene funkcije rade bolje od PyAutoGUI specifično u video igrama koje ovise o DirectX te služi za kontroliranje miša i tipkovnice. [24]
  Može se vidjeti u nekoliko primjera kako radi ova biblioteka:

```
    pozicija_x, pozicija_y = 200, 300
    pydirectinput.moveTo(pozicija_x, pozicija_y) # pomjeriti miš na koordinate x 200 i y 300
    pydirectinput.click() # potom kliknuti na trenutnu poziciju miša
    pydirectinput.click(pozicija_x, pozicija_y) # objedinjenje prethodne 2 funkcije u jedno
    pydirectinput.move(20, 10) # relativno pomjeranje miša udesno za 20 piksela te dolje za 10
    pydirectinput.press('b') # pritisak tipke b
```

- **PyAutoGUI** – pored kontroliranja miša i tipkovnice, ova biblioteka pruža još dodatne mogućnosti koje su isto bitan faktor ovog rada: slikanje ekrana i lociranje slike na ekranu te isto tako može nam dati poziciju miša i piksele koji se nalaze na toj lokaciji. Još neke značajke ove biblioteke: slanje pritiska tipke aplikaciji, lociranje prozora aplikacije te micati ga, maksimizirati/minimizirati ili zatvoriti, pokazivanje okvira upozorenja (engl. alert box). [25]
  Može se vidjeti rad ove biblioteke kroz par primjera:

```
sirina_ekrana, visina_ekrana = pyautogui.size() # dohvati dimenzije monitora
x_pozicija_misa, y_pozicija_misa = pyautogui.position() # dohvati x,y coordinate trenutne pozicije miša
pyautogui.click() # klik miša
pyautogui.alert('Python je super!') # prozor s porukom
```

- **NumPy** – jedna od najkorištenijih Python biblioteka, nudi matematičke funkcije, generiranje nasumičnih brojeva, ali ono što je bitno spomenuti je njena vektorizacija i indeksiranje, koje je dosta brže od ugrađenih Python konstrukata koji se bave time. Korištena je u gotovo svim poljima znanosti i inžinjeringa. Ova biblioteka se isto koristi u kombinaciji s ostalim Python modulima poput: Pandas, Matplotlib, SciPy. Jedna od bitnih struktura podataka su njeni nizovi (engl. arrays) tzv. ndarrays koji su multidimenzionalni, homogeni nizovi na kojima se mogu upotrijebiti razne metode. [26]

Neki primjeri ove biblioteke mogu se vidjeti ispod:

```
niz = np.arange(10) # kreiranje multidimenzionalnog niza (ndarray)
[0, …, 9]
niz2 = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]) #
kreiranje n-dimenzionalnog niza
np.ones(5) # kreiranje niza koji sadrži samo jedinice
niz_x = np.array([[4, 5], [6, 7])
niz_y = np.array([[8, 9]])
ulancani_niz = np.concatenate((niz_x, niz_y), axis=0) # ulančavanje
2 niza po osi y
```

- **Pytesseract** – biblioteka za detektiranje i čitanje teksta iz slike. Bolje rečeno, optičko prepoznavanje znakova u slikama. Bitan faktor u čitanju određenih brojeva s ekrana računalne igre. Može čitati sve tipove slika koji su podržani od strane biblioteka Pillow i Leptonica, uključujući: png, jpeg, gif, bmp, tiff i druge. [27]
  Rad biblioteke se može vidjeti ispod:

  ```
  tekst_na_slici=pytesseract.image_to_string(Image.open('slika.png')
  ) # pretvaranje teskta iz slike u strukturu string s pomoću otvaranje
  slike putem klase Image iz biblioteke Pillow
  ```

- **Visual Studio Code** – razvojno okruženje (engl. Integrated Development Environment) s mnogo značajki koje olakšavaju rad te je pogodno za Python. Sadrži tzv. IntelliSense koji omogućava pametno dovršavanje koda (baziran na tipu varijable ili definiciji funkcije) te je nasljednik podcrtavanja sintakse i običnog samodovršavanja koda. Pored toga, otklanjanje pogreški ima svoj vlastiti prozor te tijekom samog procesa možemo vidjeti informacije o lokalnim i globalnim varijablama, o rezultatima funkcije, stanjima petlje i sl. Sadrži svoj vlastiti terminal što znači da ne moramo otvarati zasebnu komdandnu ploču (engl. Command Prompt) već možemo koristiti značajku integriranog terminala (engl. Integrated Terminal) u Visual Studio Code-u.

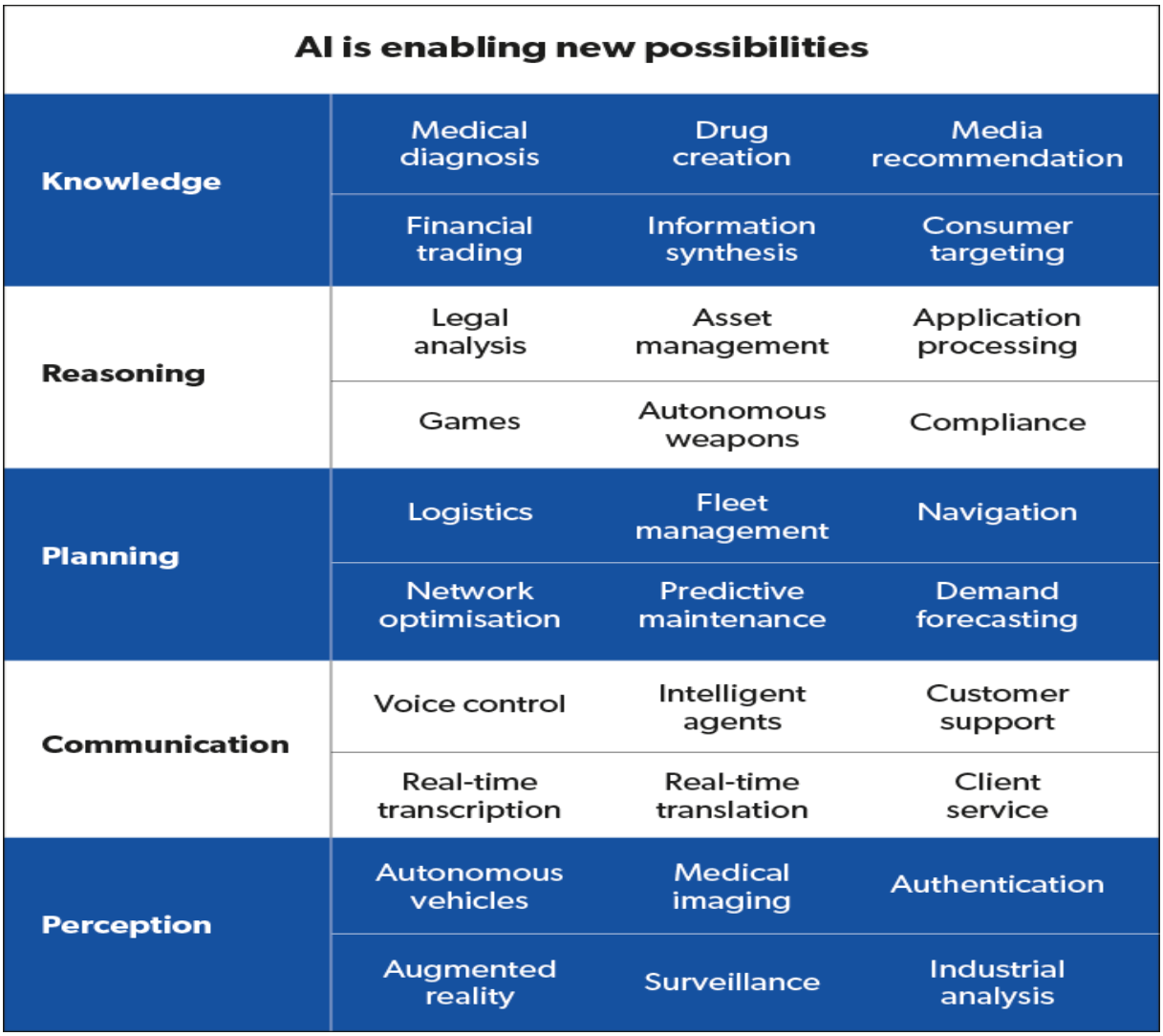
3. Umjetna inteligencija

Umjetna inteligencija se u povijesti pokušala interpretirati na različite načine. Dok su neki preferirali verziju koja govori o umjetnoj inteligenciji u smislu točnosti ljudskog ponašanja, neki pak govore kako formalniju, apstraktnu definiciju gdje se umjetna inteligencija odnosi na racionalnost. S jedne strane istražitelji se fokusiraju na svojstva umjetne inteligencije poput načina razmišljanja i razumijevanja kao internu karakterizaciju, a s druge na inteligentno ponašanje kao eksternu. [1]

Postoje razni primjeri UI u našim životima. Neki od primjera su: autonomno vozilo proizvedeno od tvrtke Tesla, proizvedena od Apple-a, preporuke muzike i videa na Youtube-u, navigacijski servisi od strane Google-a i Apple-a koji koriste UI koji npr. daju informaciju o trenutnom prometu, pametni asistenti poput Siri i Alexe, superračunalo Deep Blue itd.[22]

Značajnost UI je velika, jer ljudske sposobnosti poput razumijevanja, rasuđivanja, planiranja, komunikacije, percepcije, postaju posjedovane od softwarea efektivno i efikasno.[2] Analitički zadaci kao što su pronalaženje uzorka u podacima obavljaju se još bolje koristeći UI.

Ispod na slici možemo vidjeti kako sve ove perspektive umjetna inteligencija koristi.



Slika 3: Mogućnosti umjetne inteligencije (Izvor: MMC Ventures)

3.1 Povijest umjetne inteligencije

Kroz povijest 20. stoljeća, pojam umjetne inteligencije se počeo pojavljivati. Način na koji je bio interpretiran se može vidjeti iz primjera poput "bezsrčanog" čovjeka Tin iz Čarobnjaka iz Oz-a te čovjekolikog robota koji je oponašao Mariju u filmu Metropolis. [3]

Ideja inteligentnog stroja do 1950ih je bila samo ideja te pored računala kojeg se vidjeli kao pomagalo koje može pretvoriti tu idjeu u materiju, dogodilo se još nekoliko događaja te znatan prvi događaj na kojem se pojam umjetne inteligencije službeno upotrijebio od strane znanstvenika John McCarthyja. Na ljeto godine 1956. na univerzitetu Dartmouth u mjestu Hanover, New Hampshire, došlo je do istraživanja umjetne inteligencije čija je baza bila kako

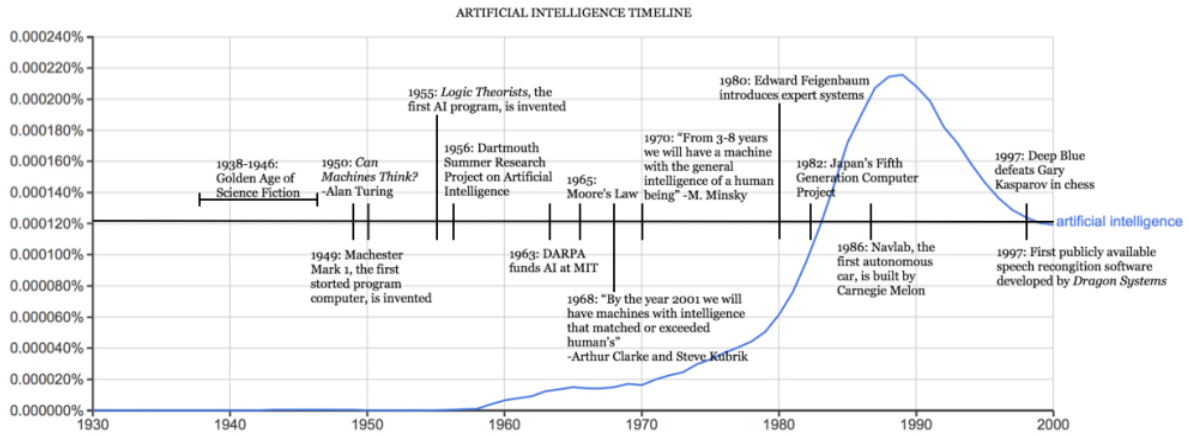
bilo koji aspekt učenja ili značajka inteligencije se može opisati kao stroj napravljen da simulira te stvari. [28]

U 1958. godini John McCarthy je doprinio umjetnoj inteligenciji tako što je definirao programski jezik LISP (dolazi od "LISt Processor") koji je ubrzo postao dominantan programski jezik sljedećih 30 godina. Pošto je za programiranje u LISP-u potrebno korištenje mnogo zagrada postoji jedna izreka "Mislio sam da LISP znači izgubljen u glupim zagradama (engl. Lost In Stupid Parentheses). [1]

U to vrijeme računala su bila skupa te nisu mogli spremati komande već ih samo izvršavati. Kroz par sljedećih desetljeća, kako su računala postajala jeftinija i brža te su mogla spremati informacije, umjetna inteligencija je cvjetala. Neke od vladinih agencija poput DARPA (engl. Defense Advanced Research Projects Agency) su bile uvjerene u rast umjetne inteligencije te su pružale sredstva za istraživanje u nekoliko institucija. U 1980im, Edward Feigenbaum je predstavio ekspertne sustave, koji su mogli oponašali proces donošenja odluka ljudskih eksperata te su bili znatno korišteni u industriji. [3]

11. svibnja 1997. godine dolazi do prekreta između ljudi i strojeva. Superračunalo DeepBlue, razvijen od strane IBM-a je porazilo tadašnjeg svjetskog prvaka u šahu, Garry Kasparov-a. IBM istraživači su započeli ovu studiju u 1985. godini, tako da je DeepBlue u mogućnosti evaluirati više od 200 milijuna mogućih pozicija u sekundi zahvaljujući bazi podataka u kojoj su bile sve šahovske igre 20og stoljeća. Umjetna inteligencija je poslije toga vidjela dosta napretka u polju igara uključujući AlphaGo, AlphaZero i AlphaStar. [29]

Ispod vidimo sliku koja dobro opisuje vremensku liniju umjetne inteligencije.Ispod vidimo sliku koja dobro opisuje vremensku liniju umjetne inteligencije.



Slika 4: Vremenska linija umjetne inteligencije (Izvor: SITN Harvard University, The History of AI)

3.2 Primjena u industriji

Može se reći kako će umjetna inteligencija revolucionizirati svijet i promijeniti budućnost s rješenjima koji se već primjenjuju u gotovo svakoj industriji s dobrim rezultatima. Nabrojit ćemo neke od grana gdje se umjetna inteligencija pokazala izuzetno efektivnom:

- **Zdravstvo** – kada se govori o doprinosu u zdravtstvenom sektoru, moraju se spomenuti velike tvrtke poput Microsofta, Google-a i Apple-a, uključujući rudarenje podataka za prepoznavanje uzoraka te samim time dolazak do točnijih dijagnoza i njege zdravstvenih stanja te otkrivanje lijekova i robotska kirurgija. Jedan primjer toga je IBM Watson koji može izvući značenje i kontekst danog skupa strukturiranih i nestrukturiranih podataka koji mogu biti kritični za odabir određenog plana liječenja te analiziranje pacijentovih zdravstvenih nalaza kako bi identificirali potencijalne planove liječenja. Može se reći da IBM Watson funkcionira poput ljudskog doktora.[5]

  Primjena UI u zdravstvu se može još vidjeti u podsjetnicima za pacijente kako trebaju uzeti svoje lijekove, ili preporuke određenih vježbi koje bi trebali vršiti. [4]

  Umjetna inteligencija se isto može iskoristiti kako bi se razvili algoritmi koji predviđaju zdravstvene rizike na nivou pojedinca, ali i populacije i na taj način poboljšati ishode zdravlja pacijenata. [30]

- **Maloprodaja i e-trgovina** – ovo je vjerojatno jedino područje gdje primjera UI je najviše primijećena od strane krajnjih korisnika. Pošto je ovo dosta natjecateljsko područje, maloprodajne organizacije uvijek traže tehnike kako bi otkrili načine ponašanja kod kupaca i tako uskladiti svoju strategiju kako bi nadmudrili svoju kompeticiju. [5]

  Za primjer možemo uzeti predlaganje proizvoda na Amazonu, gdje UI aplikacija u stvarnom vremenu putem kompleksnih UI algoritama određuje koje proizvode bi kupac želio kupiti. Pored toga, UI aplikacije se isto koriste u svrhu pojačanja korisničkog iskustva kao što su chatbots na stranicama e-trgovina te imaju širok raspon odgovara na pitanja kupaca. [4]

  Postoje razna polja u trgovini i maloprodaji koja mogu iskoristiti umjetnu Inteligenciju: raspodjela inventara (rudarenje podataka i predviđanje), prilagođena web stranica (prepoznavanje kupca i prilagođavanje iskustva temeljenog na prethodnim kuupovinama kupca i njegovim ponašanjima), vizualno selektiranje (pružanje kupcu da otkrije nove ili slične proizvode preko traženja temeljenog na slici tj. prilagođavanje preporuka temeljenih na estetici i sličnosti.) [31]

- **Bankarstvo i financijske usluge** – velika je transformacija u ovom području zbog umjetne inteligencije. U mnogo scenarija, ljudi su zamijenjeni s inteligentnim softverskim robotom za procesiranje zahtjeva za kredit u djeliću sekunde. Pored toga, UI bazirani chatbotovi su raspoređeni u sektoru osiguranja kako bi poboljšali korisničko iskustvo i kreirali osigurani plan i proizvod baziran na podacima kupca. Imamo još i UI za detekciju prijevara. Npr. Mastercard koristi UI baziranu tehnologiju donošenja odluka kako bi otkrila prevarantske transakcije analizirajući razne točke u podacima.[4]

  Automatiziranje zadataka koji zahtijevaju ručan unos i rad s podacima je bitan utjecaj UI u bankarstvu. Isto tako se može iskoristiti protiv kriminala tako što može provjeriti sumnjive aktivnosti u financijskim transakcijama. [5]

  Mnoštvo financijskih servisa uvodi UI za prevenciju rizika, banke koriste razne algoritme koji služe za bolju validaciju i autorizaciju korisnika. Isto tako se žele bolje nositi sa pranjem novca (engl. Anti Money Laundering) te provoditi tzv. provjere „upoznaj svog kupca“ (engl. Know Your Customer). [32]
- **Zabava i računalne igre** – UI pomaže proizvođačima programa identificirati koje emisije/filmove ili programe bi trebali predložiti određenim korisnicima s obizorm na njihovu aktivnost. Ovo znatno koristi Netflix i Amazon kako bi kupci imali svojstven odabir. [4] Algoritmi strojnog učenja se široko koriste kako bi proučavali ponašanja korisnika i ti algoritmi sve više „postaju“ inteligentniji s vremenom do te točke gdje mogu odrediti je li korisnik želi kupiti proizvod za sebe ili možda ga pak pokloniti. Intuitivno, industrija računalnih igara je bila jedna od ranijih koja je prihvatila UI i njen dubok utjecaj na korisničko iskustvo. UI može kontrolirati akcije NPCa [17] (engl. Non-Player Character) koji imaju ulogu u napretku priče same računalne igre u određenom smjeru. [33]

3.3 Strojno učenje kao dio umjetne inteligencije

Iako su pojmovi umjetna inteligencija i strojno učenje blisko povezani, oni nisu isti. Strojno učenje je ustvari dio umjetne inteligencije. Kao što smo prethodno ustanovili, UI oponaša ljudsku inteligenciju te je to polje računalne znanosti. Strojno učenje je primjena UI, to je proces korištenja matematičkih modela podataka kako bi pomogli računalu „naučiti“ bez

direktnih instrukcija. Postoje algoritmi poput podupiranog učenja, duboke neuronske mreže.[6] Strojno učenje omogućava računalu predvidjeti ili napraviti neke odluke bazirane na podacima bez da je eksplicitno isprogramirano što znači da programer ne mora izričito napisati programski kod koji govori računalu što raditi već računalo samo može zaključiti nakon procesa učenja. [7]

Prema Russellu i Norvigu može se reći da agent uči ako mu se poboljšava točnost prepoznavanja stvari u svijetu nakon određenog opažanja. Agent opaža neku određenu količinu podataka te na osnovu toga pravi model i koristi taj model kao hipotezu o svijetu, ali i softver koji može rješavati probleme. Postoje 3 glavna tipa učenja:

- Pod nadzorom – ovdje agent promatra parove ulaza i izlaza i uči funkciju koja mapira ulaze do izlaza. Za primjer se mogu uzeti slike pasa, ili mačaka, ali koje u izlazu isto imaju i labelu koja govori „pas“, ili „mačka“. Agent proučava funkciju pomoću koje će moći predvidjeti labelu, s obzirom koja slika mu je predočena.
- Bez nadzora – agent uči uzorke iz ulaza bez da mu je dana labela u izlazu. Ovdje je poznat zadatak grupiranja gdje se agentu mogu prikazati milioni slika, a računalni vid će identificirati velike grupe sličnih slika.
- Podupirano učenje – ovdje agent uči iz niza potkrepljivanja tj. nagrada i kazni. Ako uzmemo za primjer igru League of Legends, agent će na kraju dobiti povratnu informaciju je li pobijedio (nagrada) ili izgubio (kazna). Onda agent treba odlučiti koje akcije su dovele do tog rezultata i koje bi trebao izmijeniti kako bi bio nagrađivan u budućnosti.

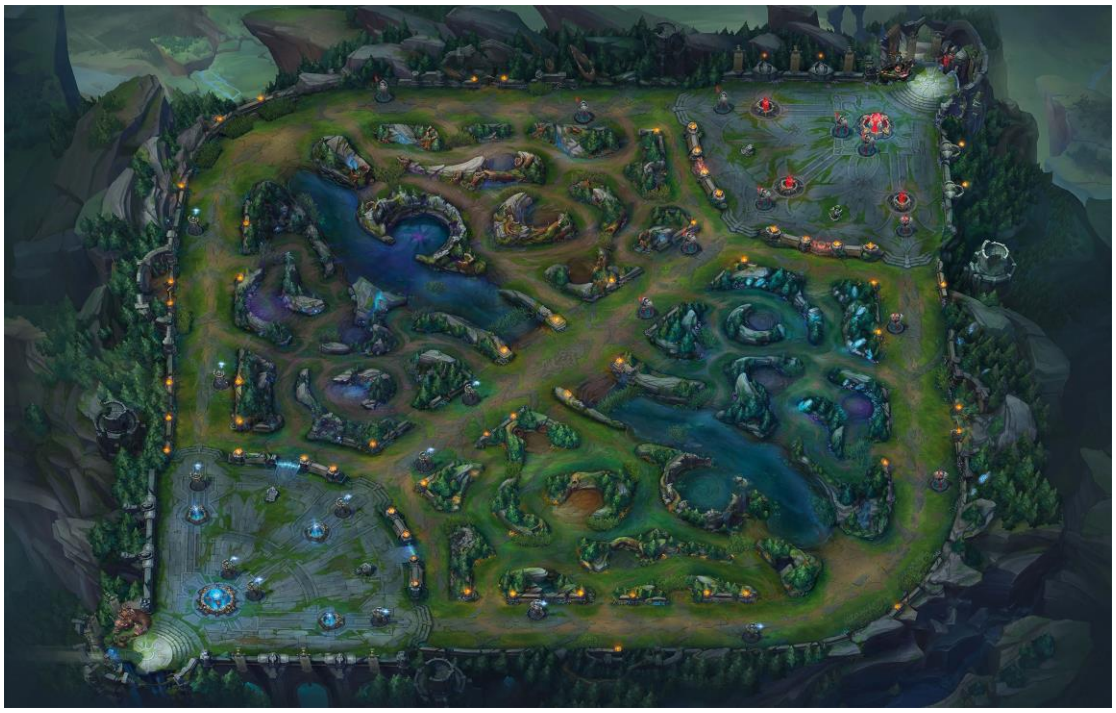
Možemo zaključiti da je strojno učenje podskup od UI te da je to način za računala da razviju svoju vlastitu inteligenciju. Cilj strojnog učenja je da računala uče iz podataka koji su im dani kako bi mogli dati ispravne izlazne podatke. Možemo reći da su računala trenirana da izvršavaju određene zadatke.[7]

4. Uvod u računalnu igru – League of Legends

League of Legends je računalna igra koja spada u žanr zvan MOBA (Multiplayer Online Battle Arena što znači da LoL spada u kategoriju online strateških igara u stvarnom vremenu gdje se dva time od nekoliko heroja/likova bore u cilju zaštite svoje baze te uništavanje protivničkih struktura kako bi uništili njihovu bazu. [16]

Izdana je od strane tvrke Riot Games koja je osnovana 2006. godine od strane Brandon Beck-a i Marc Merrill-a. Stvaranje LoL-a je započelo negdje u 2008. godini te prva verzija je izdana 2009. godine. U to vrijeme nije postojalo mnogo MOBA igara, osim jedne igre koja je bila uspjeh na tržištu pod nazivom DOTA (engl. Defense of the Ancients). Trenutno je LoL jedna od najpoznatijih računalnih igara u svijetu. [19]

LoL je igra koja zahtijeva dosta uloženog vremena jer postoji mnogo stvari za naučiti kako bi se uopće mogla igrati na početničkom nivou. Postoje 2 velika aspekta: sama mehanika koju igrač može imati i razne strategije tj. donošenje odluka tijekom igre. Za oba aspekta potrebno je imati veliko znanje o igri dok se mehanika još može poboljšati repetitivnim putem na način da se ponavljaju određene radnje kako bi one postale druga priroda igraču. Igra se može igrati na različitim mapama te kroz zadnjih 10 godina neke mape su otišle u zaborav te više nisu dostupne, ali ovdje će se fokusirati na određenu mapu koja se najviše igra/koristi „Summoner's rift“ te ovo ime označava magično borbeno polje na kojem se 2 tima od pet igrača bori za pobjedu. Ispod na slici je prikaz ove mape.



Slika 5: Mapa League of Legends (Izvor: League of Legends Wiki)

Srce svake baze je tzv. Nexus, struktura koja je locirana u donjem lijevom kutu i gornjem desnom i cilj je uništiti tu strukturu. Do toga se dolazi putem uništavanja tzv. turret-a, kule koja brani svoje područje napadajući protivnike i ubijajući minione, stvorenja koja svaki tim ima, a možemo ih definirati kao podanike koji se bore za svoj tim. Pored turreta postoji i tzv. inhibitor, koji se može definirati kao vrata do jačih svojstvenih miniona te pojačavanja linija napada. Na dijagonali sjeverozapad-jugoistok postoje gornja i donja rijeka, u kojima se nalaze neutralna čudovišta koje oba tima želi ubiti kako bi dobili razna pojačanja i novac. Heroji koje igrači kontroliraju idu na trake (gornja, srednja i donja) kroz koje minioni prolaze i na kojima se turreti nalaze. Pored toga postoji i džungla koja se nalazi na na svim stranama, ali je smještena između traka te većinom 1 igrač ide u džunglu kako bi ubio neutralna čudovišta koje džungla sadrži. Sva stvorenja u igri (minioni, neutralna čudovišta, heroji koje igači kontroliraju) te strukture (turret, inhibitor i nexus) daju novac i iskustvo kad su ubijeni/uništeni. Dobijeno iskustvo podiže level (nivo) samog heroja koje mu daje pojačane atributre poput životnih bodova, otpornosti, napada i sl. Tim dobijenim novcem igrači kupuju iteme tj. stvari koje se mogu kupiti u trgovini koja se nalazi u bazi. Te stvari daju heroju razna pojačanja, ovisno o tome koje igrač odluči da treba.

Prema [20] postoji 6 kategorija u koje heroji mogu spadati:

- Ubojice – brzo se kreću i prave mnogo štete u kratkom vremenskom roku. Cilj im je ulijetanje i brzo iskakanje na sigurno područje.
- Borci – heroji ove kategorije mogu činiti puno štete, ali i podnijeti Odlični su za borbe koje duže traju.
- Čarobnjaci – najbolje su korišteni na velikoj distanci. Mogu napraviti mnogo štete protivnicima, ali su lako ranjivi te moraju biti zaštićeni od strane suigrača.
- Strijelci – poput čarobnjaka su, ali razlika je u tome što čarobnjaci koriste čarolije kako bi pravili štetu, a strijelci svoje osobne napade koje svaki heroj ima, ali u slučaju ove klase strijelci su odlični u tome.
- Podrška – uloga ove kategorije je pojačavanje mogućnosti svojih suigrača te zaštita i sigurnost istih.
- Tenk – slični su borcima, ali prave manje štete i mogu izdržati više napada.

Pored ove podjele postoji način na koji bi svaki tim trebao rasporediti svoje heroje na mapi. Jedan heroj ide na top traku (engl. Toplane), jedan u džunglu (engl. jungle), jedan u srednju traku (engl. Midlane) te dva u donju traku (botlane), s tim da neki heroji su specifično dobri za džunglu, dok na donjoj traci inače idu strijelac i support. Ispod su prikazane slike i opisi za različita čudovišta, minione te strukture.

Slika 6: minioni (Izvor: https://notagamer.net/league-of-legends-riot-games-should-nerf-minions-immediately-after-seeing-this-it)

Na slici iznad su minioni, tj. podanici koje svaki tim ima. Postoje 3 vrste miniona: casteri (minioni koji napadaju sa distance, tj. imaju dug domet napada), melees (minioni sa sjekirama, bliska borba) te cannon (najjači minion, vidimo ga u sredini s dugim topom).

Slika 7: džungla i čudovišta u njoj (Izvor: https://www.pinterest.com/pin/443745369504598028/)

Na ovoj slici se vidi džungla te razna čudovišta koja u njoj mogu biti. Različita čudovišta daju različita poboljšanja (brže regeneriranje životnih bodova ili bodova koji su potrebni za korištenje čarolija, jači napadi i sl.). Sva čudovišta daju iskustvo i novac koji su potrebni za podizanje heroja na viši nivo koji mu omogućuje poboljšanja u napadu, obrani i čarolija.

Slika 8: Baron (Izvor: https://esportsone.com/complete-guide-to-league-of-legends/)

Ovo je jedan od najsnažnijih neutralnih čudovišta u igri. Timovi se natječu ubiti Barona jer time oni dobijaju novac, iskustvo, razna pojačanja svojih heroja te osnažuje njihove vlastite minione. Ti efekti traju 3 minute.

Slika 9: Turret (Izvor: https://esportsone.com/complete-guide-to-league-of-legends/)

Na ovoj slici se vidi turret (hrv. toranj) koji je obrambena struktura. Svaka traka ima 3 turreta, uključujući i 2 turreta ispred Nexusa (u samoj bazi oba tima). Uništenjem turreta protivnički tim dobija novac te može nastaviti napadati dublje u protivnički teritorij.

Slika 10: Inhibitor (Izvor: https://esportsone.com/complete-guide-to-league-of-legends/)

Slika iznad prikazuje inhibitor, struktura koja nema napad, ali uništenjem ove strukture, minioni protivničkog tima postanu snažniji te ih je teže ubiti i stvaraju veći pritisak protvnicima.

Slika 11: Nexus (Izvor: https://esportsone.com/complete-guide-to-league-of-legends/)

Srce svakog tima je Nexus, struktura čijim uništenjem protivnici završavaju igru pobjedom. Nema osobni napad te iako uništenjem Nexusa igra odmah završava, heroj koji uništi Nexus dobije 50 gold-a (novac u zlatu). Cilj igre je doći do Nexusa i uništiti ga.

Ovim se završava uvod u igru League of Legends te u sljedećem poglavlju obradit će se implementacija inteligentnog agenta koji igra ovu igru.

5. Izrada agenta koji igra MOBA igru (S.I.A. – Seraphine Intelligent Agent)

LoL se čini kao jako kompleksna igra te uvod u samu igru je zahtjevan proces, stoga dolazi do potrebe inteligentnog agenta (Seraphine Intelligent Agent) koji može igrati računalnu igru i olakšati proces uvođenja novih igrača, a način rada samog agenta je korištenje automatizacije i raznih procesa odlučivanja. Računalna igra se osnovno sastoji od heroja koje igrači odaberu za igrati, od stvari koje mogu kupiti u trgovini računalne igre te četverokutna mapa na kojoj se igra ima 3 trake: gornja, srednja i donja. Pored toga na mapi ima i džungla koja sadrži razna čudovišta, a pored njih postoje i tzv. „minioni“ (u prijevodu „podanici“), slabiji likovi koje je potrebno ubijati kako bi dobili novac kako bi igrači mogli kupiti stvari u trgovini. Cilj igre je uništiti protivnički „Nexus“ koji je građevina u samoj bazi svakog tima.

Dok će se kasnije u radu vidjeti kako točno SIA radi, prvo će biti objašnjen heroj koji je izabran za ulogu inteligentnog agenta. Seraphine je mage midlaner (čarobnjak namijenjen za srednju traku), što znači da ide u srednju traku mape te su njene glavne sposobnosti korištenje čarolija i pritom radeći dosta štete protivničkom timu te ona je pop pjevačica u samoj povijesti računalne igre. „Jednog dana, muzika će stati pa bolje pleši dok možeš!“ Ispod je slika kako Seraphine izgleda.

Slika 12: Seraphine (Izvor: https://www.leagueoflegends.com)

Igrači mogu koristiti čarolije putem miša ili tipkovnice. Puno je lakše koristiti tipkovnicu jer time se ne troši vrijeme koje je potrebno za pozicioniranje na sliku čarolije putem miša. LoL ima postavke koje se odnose na korištenje tipkovnice. Te postavke se zovu prečaci (engl. hotkeys) te pomoću njih se odaberu tipke koje će služiti za korištenje čarolija. Uobičajenost je da igrači izaberu 4 prečaca: Q, W, E i R za osobne/obične čarolije heroja te D i F za posebne čarolije koje ima svaki heroj. Čarolija Q će označavati prvu osobnu čaroliju heroja te W, E i R će označavati drugu, treću i ultimativnu. Bitno je napomenuti kako svaka njena čarolija je AoE (engl. Area of Effect) tipa što znači da može kliknuti na bilo koje područje te će šteta biti raspršena na tom području. U nastavku su opisane čarolije od Seraphine.

Njezina pasivna moć (čarolija) se zove „Prisutnost na bini“ što znači da će za svaku 3. običnu čaroliju ona pozvati silu, koja će automatski baciti čaroliju još jednom. Pored toga, kad god poziva bilo koju čaroliju blizu svojeg prijateljskog heroja, kreirat će notu. Svaka nota daje njenim običnim napadima (kojeg svi heroji imaju-klikanje na neprijatelja) više dometa te pravi dodatnu magičnu štetu, koja „potroši“ notu.

Seraphinina prva obična čarolija (ili korištenje prečaca Q) se zove „Visoka nota“ te pomoću nje ona projektira čistu notu, koja pravi magičnu štetu na terenu na kojem je projektirana te što više HP-a (engl. Health Points, u prijevodu „životni bodovi“) fali protivniku, toliko će biti povećana nanesena šteta. Ova čarolija se često koristi jer je vrijeme potrebno da bude opet slobodna nisko, ali i bitna je za samu borbu s minionima i neprijateljskim herojima.

Druga obična čarolija (ili korištenje prečaca W) se zove „Okružujući zvuk“ te pomoću toga ona okružuje prijateljske heroje koji su blizu s pjesmom, koja daje njoj i njenim prijateljskim herojima brzinu kretanja te štit. Ako je Seraphine već pod štitom, ona može izliječiti prijateljske heroje koji su blizu te im povratiti HP bazirano ovisno o tome koliko prijateljskih heroja je blizu nje. Ova čarolija omogućuje Seraphine i njenim prijateljskim herojima da ostanu što duže u borbi te životni bodovi koje ova čarolija pruža mogu promijeniti ishod borbe.

Treća obična čarolija (ili korištenje prečaca E) se zove „Puštanje ritma“ pomoću kojeg ona ispušta jak zvučni val, koji pravi magičnu štetu neprijateljima koji se nađu u liniji te ih usporava. Neprijatelji koji su već usporeni su stopirani u mjestu, te oni koji su već stopiranu u mjestu, su šokirani (ne mogu ništa raditi ni kretati se). Ova čarolija spada u tip čarolije zvan CC (engl. Crowd Control) što označava da ona moža kontrolirati kretnje neprijatelja, u ovom slučaju ne dopušta im kretanje uopće, ili ga usporava.

Njena ultimativna (najjača) čarolija se zove „Ponavljanje pjesme“, te njome Seraphine „zauzima binu“ te projektira zapanjujuću silu pomoću koje zavede neprijatelje (krenu ići prema njoj) te im nanosi magičnu štetu. Bilo koji neprijatelj koji je pogođen s tim (uključujući i prijatelje) postanu dio njene performanse, koji produžuju domet čarolije te daje prijateljima maksimalne note. Ova čarolija isto spada u CC tip te se inače koristi na početku borbe kako bi se dobila

znatna prednost ako Seraphine uspije njom pogoditi neprijatelje jer domet koji ova čarolija potencijalno ima je velik. Ispod vidimo sliku ovih čarolija (u poretku: pasiv, Q, W, E, R).

Slika 13: Čarolije (Izvor: League of Legends Wiki)

Seraphine se može vidjeti kao heroj koji je jednostavan. Iako je midlane heroj može biti i support (podrška) na donjoj traci mape uparena s drugim prijateljskim herojem. Neke njene čarolije poput W su jako dobar razlog za to jer može pomoći prijateljima davajući/regenerirajući im HP, štit i brzinu. Pored toga, njezina čarolija Q je jako dobra za čišćenje (ubijanje) miniona te guranja svojih miniona naprijed. Čarolija E je jako korisna kada treba pripremiti neku kombinaciju čarolija sa svojim prijateljskim herojima jer ih ova čarolija zadržava u mjestu ili usporava, ovisno o situaciji. I na kraju, njezina ultimativna čarolija koja je najjača može promijeniti ishod borbe u sekundama te omogućiti pobjedu svom timu zbog velikog dometa čarolija. Seraphine uvijek želi ostati na distanci na kojoj je neprijateljski heroj ne može lako napasti jer je dosta podložna povredama ako je neprijatelj blizu jer ona nije namijenjena za blisku borbu.

Razlog zbog kojeg je izabrana Seraphine jesu jer je jednostavna, kombinacije koje njene čarolije mogu učiniti te davanje štita koji omogućuje duže preživljavanje u borbi te nastavak same borbe ili pobjeda borbe te isto tako je ona ima velik domet bazičnih napada (auto attacks) koji se odrađuju pomoću klikanja na protivnike (bez korištenja čarolija). Njezine čarolije djeluju na određenu arenu (nisu orijentirane na neprijatelje već na područje) te je to glavni razlog zbog kojeg sam ju izabrao. Imaju dobar domet koji je čini sigurnom te sam štit i HP koji dobiva od čarolije W te samim tim može duže ostati u traci i boriti se prije povratka u bazu. Ovi razlozi je čine dobrim izborom da bude League UI (umjetna inteligencija).

Tu postoje isto i tzv. Rune (pojačanja) koje su pored samih čarolija heroja te stvari koje se mogu kupiti u trgovini, dodatno pojačanje raznih aspekata heroja. To može biti defenzivan ili ofenzivan odabir, ovisi na koji način i stil igrač želi igrati. Na slici ispod su rune koje su odabrane za Seraphine, većinom su napadačkog tipa. Ispod možemo vidjeti sliku ovoga.

Slika 14: Rune/Pojačanja (Izvor: Vlastita izrada)

Naposljetku, tu su i stvari (engl. items) koje su u modernom LoL-u prerasli u jednu od najvažnijih vještina u računalnoj igri. Slika ispod prikazuje stvari koje SIA kupuje te oni se dijele na komponente, kompleksnije komponente te kompletne stvari. Svaka kompletna stvar je sačinjena od komponenata koje imaju znatno manju cijenu od same stvari. Stvari koje sam odabrao za Seraphine su orijentirani na stvaranje štete protivniku te brzinu kretanja. Bot (inteligentni agent) će uvijek prvo kupiti komponente jer nema razloga čekati na novac za kompletne stvari. Ispod vidimo sliku.

Slika 15: Itemi (Izvor: Vlastita izrada)

Ideja na kojoj je napravljen S.I.A. je sljedeća: detekcija objekata u stvarnom vremenu te automatizacija akcija/stvari koje bi čovjek radio ako bi on igrao računalnu igru. Prije izrade samog agenta, potrebno je nabrojati listu stvari koje bi se trebale automatizirati:

- Na početku igre kupiti stvari te otići u neku traku (u ovom slučaju srednju)
- Čekati prijateljske minione te onda stupiti u borbu s protivničkim minionima te neprijateljskim herojima
- Napasti protivnički toranj kada je to sigurno
- Sama borba s neprijateljskim herojima u raznim slučajevima
- Kupovina stvari ako postoji dovoljno novca za to
- Samo kretanje po traci
- Korištenje čarolija/posebnih čarolija i bazičnih/osobnih napada
- Povući se u bazu ako ima malo HP-a
- Otvaranje trgovine te pronalazak stvari za kupiti
- Interakcija toranj-neprijateljski heroj-neprijateljski minioni-prijateljski minioni
- Napraviti sigurno vraćanje u bazu
- Provjeriti koliko novca ima
- Postaviti čarolije na početnu/višu razinu

- Provjera dometa neprijateljskih heroja/tornja
- Usporedba piksela

Za detekciju objekata prvo je bilo potrebno naći način na koji bi usporedba uzorka i slike u kojoj se traži kako bi podudarnost mogla biti dovoljno visoka kako ne bi došlo do lažnih pozitiva. Tako da, za objekte poput tornjeva, heroja, miniona, potrebno je naći za svaku skupinu nešto zajedničko preko čega se mogu identificirati i usporediti. U tom slučaju, zajedničko što svi oni imaju, ali razlika u izgledu, je njihova HP linija, iznad samog lika. Ona nam omogućava da istovremeno smanjimo broj slučajeva, ali isto tako da može dobiti dovoljno velik stupanj pouzdanosti tijekom identificiranja. Nakon uspješne detekcije, radimo razne radnje s mišom/tipkovnicom bazirane na raznim slučajevima. U par navrata, potrebno je i „izvaditi“ tekst iz slike, u ovom slučaju brojeve, koji nam mogu dati informaciju koliko novca trenutno imamo. U nastavku ćemo pobliže vidjeti kod te ga opisati.

```
1. object_level_up = ObjectFinder('images/levelUP.png')
2. object_enemy_turret = ObjectFinder('images/enemyTurretHP.png')
3. object_enemy_minion = ObjectFinder('images/enemyMinionHP.png')
4. object_enemy_champion = ObjectFinder('images/enemyHP.png')
5. object_enemy_turret_plate =
   ObjectFinder('images/enemyTurretHPplate.png')
6. object_ally_minion = ObjectFinder('images/alliedMinionHP.png')
7. object_ally_champion = ObjectFinder('images/alliedChampion.png')
```

Isječak programskog koda 1. Objekti detekcije

Ovdje se može vidjeti da koristimo klasu ObjectFinder koju ćemo uskoro vidjeti što ona radi, instanciramo objekte preko kojih ćemo moći raditi detekciju te u rezultatima će biti koordinate na kojim se nalaze ako su pronađeni. Slike su izuzetno male, ali kao što je prije navedeno, to su HP linije iznad samog lika, tj. Mali početak te linije, s obzirom ako bi uzeli više od toga ponekad kad bi objektov HP se spusti ispod toga, ne bi došlo do velike podudarnosti.

Idemo vidjeti što točno ObjectFinder klasa radi.

```
1. class ObjectFinder:
2.     sample = None
3.     sample_width = 0
4.     sample_height = 0
5.     method = None
6.     def __init__(self, sample_path, method=cv.TM_CCOEFF_NORMED):
7.         self.sample = cv.imread(sample_path, cv.IMREAD_UNCHANGED)
8.         self.sample_width = self.sample.shape[1]
9.         self.sample_height = self.sample.shape[0]
10.          self.method = method
11.
12.      def find(self, haystack_img, threshold=0.5):
13.          result = cv.matchTemplate(haystack_img, self.sample,
    self.method)
14.          locations = np.where(result >= threshold)
15.          locations = list(zip(*locations[::-1]))
16.          rectangles = []
17.          for loc in locations:
18.              rect = [int(loc[0]), int(loc[1]), self.sample_width,
    self.sample_height]
19.              rectangles.append(rect)
20.              rectangles.append(rect)
21.          rectangles, weights = cv.groupRectangles(
22.              rectangles, groupThreshold=1, eps=0.5)
23.          points = []
24.          if len(rectangles):
25.              for (x, y, w, h) in rectangles:
26.                  center_x = x + int(w/2)
27.                  center_y = y + int(h/2)
28.                  points.append((center_x, center_y))
29.
30.          return points
```

Isječak programskog koda 2. Klasa ObjectFinder

Ovdje se može vidjeti na samoj inicijalizaciji objekta da koristimo metodu iz OpenCV biblioteke, imread(), te ona prima 2 parametra: put to slike koja je uzorak te zastavicu kojom govorimo u kojem modu želimo učitati sliku, bilo to nepromijenjeno, u sivoj nijansi te isto možemo upravljati dimenzijama slike. Isto tako postoji i metoda find() kojom tražimo dani uzorak u „plastu sijena“ (haystack_img), koristeći matchTemplate funkciju koja prima parametre za sliku u kojoj se obavlja detekcija, sami uzorak, te metoda kojom će se izvršiti proces. Svaka metoda je matematički opisana kako ona radi i u to nećemo ulaziti, ali za reći može se da jednostavno trebamo probati različite metode i vidjeti koja za danu upotrebu daje najbolje rezultate. Poslije toga od linije 14 do 30, koristimo NumPy biblioteku preko čijih funkcija ćemo usporediti rezultate od matchTemplate s granicom (threshold) kojom postavi, po defaultu je 0.5. Na liniji 15 imamo zanimljiv kod koji jednostavno dobije listu lokacija gdje se rezultat podudara, te otpakiramo listu pomoću * kako bi mogli zipati x,y koordinate. Ostali dio koda su pravokutnici koji su napravljeni od x,y koordinata koje su gornja lijeva pozicija te širina i visina uzorka te ih dodajemo u listu 2 puta jer grupiramo pravokutnike (može se pojaviti više njih na 1 mjestu) te grupiranje se vrši ako ih ima više od 1 inače ih eliminira. Na kraju, ako se pronađe

objekt, pravokutnici će biti tu te ćemo napraviti točke koje ćemo vratiti našim objektima kreiranim u slici prije.

```
1. class WindowCapture:
2.
3.     w = 0
4.     h = 0
5.     hwnd = None
6.     def __init__(self, window_name=None):
7.         if window_name is None:
8.             self.hwnd = win32gui.GetDesktopWindow()
9.         else:
10.             self.hwnd = win32gui.FindWindow(
11.                 None, window_name)
12.             if not self.hwnd:
13.                 raise Exception('Window not found:
   {}'.format(window_name))
14.         self.w = 1920
15.         self.h = 1080
16.
17.     def get_screenshot(self):
18.         wDC = win32gui.GetWindowDC(self.hwnd)
19.         dcObj = win32ui.CreateDCFromHandle(wDC)
20.         cDC = dcObj.CreateCompatibleDC()
21.         dataBitMap = win32ui.CreateBitmap()
22.         dataBitMap.CreateCompatibleBitmap(dcObj, self.w, self.h)
23.         cDC.SelectObject(dataBitMap)
24.         cDC.BitBlt((0, 0), (self.w, self.h), dcObj,
25.                    (self.cropped_x, self.cropped_y),
   win32con.SRCCOPY)
26.         signedIntsArray = dataBitMap.GetBitmapBits(True)
27.         img = np.fromstring(signedIntsArray, dtype='uint8')
28.         img.shape = (self.h, self.w, 4)
29.         dcObj.DeleteDC()
30.         cDC.DeleteDC()
31.         win32gui.ReleaseDC(self.hwnd, wDC)
32.         win32gui.DeleteObject(dataBitMap.GetHandle())
33.         img = img[..., :3]
34.         img = np.ascontiguousarray(img)
35.         return img
36.
37.     @staticmethod
38.     def list_window_names():
39.         def winEnumHandler(hwnd, ctx):
40.             if win32gui.IsWindowVisible(hwnd):
41.                 print(hex(hwnd), win32gui.GetWindowText(hwnd))
42.         win32gui.EnumWindows(winEnumHandler, None)
```

Isječak programskog koda 3. Klasa WindowCapture

Ovo je klasa koja služi za dohvaćanje aktivnog prozora kojeg ćemo konstantno skenirati. Postoji statička metoda preko koje samo izlistavamo prozore koji su otvoreni,

get_screenshot() metoda koja je istražena da je najbolji/najbrži način da slikanje ekrana, tj. Daje najbolji rezultat glede slika u sekundi. Programski kod ove klase je preuzet sa [18].

```
1. pydirectinput.press('p')
2. right_click(341, 329)
3. pydirectinput.press('p')
4. bought_items.append('Amplifying tome')
5. last_bought_timer = time.time()
6.
7.
8. pydirectinput.press('y')
9. right_click(1755, 948)
10.
11.  loop_time = time.time()
```

Isječak programskog koda 4. Početak igre

Ovaj kod se izvršava samo jednom, na početku same igre, gdje otvara trgovinu (linija 1) te kupuje prvu komponentu i dodaje je u listu kupljenih stvari koja će kasnije biti bitna za sam proces kupnje te isto tako postavljamo timer kad smo zadnji put kupili i s mišem onda usmjeravamo Seraphine do najdaljeg prijateljskog tornja na srednjoj traci. Linija 8 zaključava kameru na našeg lika, tj. Konstantno ga prati.

```
1. while(True):
2.     dead_pixels = pyautogui.screenshot().getpixel((691, 1002))
3.     if dead_pixels == (184, 184, 184):
4.         while dead_pixels == (184, 184, 184):
5.             dead_pixels = pyautogui.screenshot().getpixel((691,
   1002))
6.
7.         else:
8.             is_worth_recalling, money = worth_recalling()
9.
10.              if is_worth_recalling:
11.                 pydirectinput.press('p')
12.
13.                 for item in list_of_items:
14.                     if not item[0] in bought_items and item[1] <
   money:
15.                         x, y = item[2]
16.                         win32api.SetCursorPos((x, y))
17.
   win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTDOWN, 0, 0)
18.                         time.sleep(0.1)
19.
   win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTUP, 0, 0)
20.                         bought_items.append(item[0])
21.                         money -= item[1]
22.                         time.sleep(0.1)
23.                 pydirectinput.press('p')
24.                 last_bought_timer = time.time()
25.                 time.sleep(6)
26.             right_click(1869, 832)
27.             time.sleep(0.01)
```

```
28.             pydirectinput.press('f')
29.             time.sleep(0.01)
30.             continue
```

Isječak programskog koda 5. Početak glavne petlje

I ovdje započinje petlju koja će se odvijati tijekom cijele igre. Ovdje je korištena biblioteka PyAutoGUI (nova, bolja verzija od AutoPY) kako bi dohvatili piksele na određenoj lokaciji i tako ustanovili je li S.I.A živ ili ne. Dok god nije, iznova dohvaćamo piksel. Imamo zanimljivu strukturu while-else, ne tako česta, ali dobra u ovom slučaju, nakon što S.I.A. ponovno oživi, imamo funkciju worth_recalling() koja služi da pregleda je li potrebno vratiti se i kupiti stvari (nju ćemo opisati u sljedećoj slici). Ako jeste, onda listamo kroz listu stvari koje su grubo kodirane gdje je svaka stvar podatkovne strukture tuple (ime, lokacija, cijena) te ako nije u listi kupljenih stvari, dodamo ga, zatvaramo trgovinu (linija 23) i postavljamo našeg lika na daljnju lokaciju na mapi koristeći ubrzanje (linija 28). Continue u ovom i u svakom slučaju što prije završava petlju kako bi se moglo ponovno skenirati što prije.

```
1. list_of_items = [('Amplifying tome', 435, (341, 329)), ('Boots', 300,
   (395, 326)),  ('Sapphire Crystal', 350, (451, 329)),
2.  ('Ruby Crystal', 400, (504, 325)), ('Blasting Wand', 850, (561,
   325)), ('Rod', 1250, (618, 328)), ('Sorcs', 1100, (345, 441)),
3.   ('Lost Chapter', 1300, (394, 448)),('Codex', 900, (453, 441)),
   ('Aether Wisp', 850, (507, 438)), ('Jewel', 1250, (558, 441)),
4.    ('Giants belt', 1000, (619, 442)), ('Liandrys', 3200, (342, 557)),
   ('Cosmic Drive', 3000, (397, 557)),
5.    ('Void Staff', 2800, (447, 562)), ('Rylais', 2600, (507, 556)),
   ('Rabscap', 3600, (561, 558))]
6.
7. def worth_recalling():
8.     pydirectinput.press("p")
9.     coins_picture = pyautogui.locateOnScreen('images/coins.png',
   confidence=0.8)
10.      print(coins_picture)
11.      x, y = coins_picture[0], coins_picture[1]
12.      win32api.SetCursorPos((x, y))
13.      pic = pyscreenshot.grab(
14.          bbox=(x+coins_picture[2], y, x+coins_picture[2]+70,
   y+coins_picture[3]))
15.      pydirectinput.press("p")
16.
17.      pic.save("moneyCurrent.png")
18.      worth_recalling = False
19.
20.      money = pytesseract.image_to_string('moneyCurrent.png')
21.
22.      if not money:
23.          return False, 0
24.      money = int(money)
25.      if any(item[1] <= int(money) for item in list_of_items):
26.          worth_recalling = True
27.
```

```
28.      return worth_recalling, money
```

Isječak programskog koda 6. Lista stvari te funkcija worth_recalling()

Prethodno spomenutu funkciju worth_recalling() je ključna za kupovanje tvari. Otvara trgovinu, koristi funkciju locateOnScreen() kojoj šaljemo uzorak kojeg želimo pronaći s određenom pouzdanošću te dobijemo njene koordinate kako bi mogli vidjeti koje je stanje novca trenutno. Spremamo tu sliku te vadimo tekst iz nje (pytesseract biblioteka), tj. broj novaca te vraćamo rezultat koji kaže je li potrebno kupovati i koliko novca imamo.

```
1.      ultimate_pixels = pyautogui.screenshot().getpixel((955, 1001))
2.      low_hp_pixels = pyautogui.screenshot().getpixel((840, 1045))
3.      if low_hp_pixels == (1, 13, 7):
4.          time.sleep(0.05)
5.          pydirectinput.press('w')
6.          time.sleep(0.05)
7.          pydirectinput.press('d')
8.          time.sleep(0.05)
9.          low_hp_pixels = pyautogui.screenshot().getpixel((840, 1045))
10.           if low_hp_pixels == (1, 13, 7):
11.               print('low')
12.               go_back('recalling')
13.
14.       screenshot = window_capture.get_screenshot()
15.       level_up = object_level_up.find(screenshot, 0.9)
16.       if len(level_up) > 0:
17.           if len(level_up) == 1:
18.               left_click(level_up[0][0], level_up[0][1])
19.           if len(level_up) == 2:
20.               left_click(level_up[random.randint(0, 1)][0],
21.                          level_up[random.randint(0, 1)][1])
22.           if len(level_up) == 3:
23.               left_click(level_up[random.randint(0, 2)][0],
24.                          level_up[random.randint(0, 2)][1])
25.           if len(level_up) == 4:
26.               left_click(level_up[random.randint(0, 3)][0],
27.                          level_up[random.randint(0, 3)][1])
28.           win32api.SetCursorPos((sia_x, sia_y))
```

Isječak programskog koda 7. Nizak HP/Level up

U ovom dijelu se dohvaćaju piksele od naše najjače čarolije, kao i što provjeravamo je li imam nizak HP te ako jeste, koristimo čarolije koje će ga povisiti te ako je još uvijek nizak, jednostavno idemo nazad u bazu pomoću funkcije go_back() koju ćemo uskoro opisati. Poslije toga koristimo klasu WindowCapture preko koje dohvaćamo sliku ekrana te provjeravamo ako možemo dobiti neku čaroliju i onda nasumično povisimo neku razinu čarolije.

```
    ally_minion = object_ally_minion.find(
         screenshot, 0.93)
     enemy_minion = object_enemy_minion.find(
         screenshot, 0.93)
     enemy_turret = object_enemy_turret.find(
         screenshot, 0.93)
     enemy_turret_plate = object_enemy_turret_plate.find(
         screenshot, 0.93)
     enemy_champion = object_enemy_champion.find(
           screenshot, 0.95)
       go_buy_timer = time.time()
```

Isječak programskog koda 8. Traženje objekta u slici ekrana

Ovdje samo koristimo prethodno objašnjenu metodu find() te rezultate spremamo za određene stvari poput prijateljskih miniona, neprijateljskih, tornjeva, heroja te naposljetku imamo timer koji sprema trenutno vrijeme koje se kasnije uspoređuje s timerom kada smo zadnji put kupovali.

```
     if go_buy_timer - last_bought_timer > 270:
         print(f"{go_buy_timer-last_bought_timer}")
         go_back()
         time.sleep(2)
         is_worth_recalling, money = worth_recalling()
         if is_worth_recalling:
             right_click(887-500, 496+450)
             time.sleep(3)
             right_click(887-115, 496+300)
               time.sleep(3)
               pydirectinput.press('b')
               time.sleep(9)
               pydirectinput.press('s')
               pydirectinput.press('p')
               for item in list_of_items:
                   if not item[0] in bought_items and item[1] < money:
                       x, y = item[2]
                       win32api.SetCursorPos((x, y))
                       win32api.mouse_event(
                           win32con.MOUSEEVENTF_RIGHTDOWN, 0, 0)
                       time.sleep(0.1)
                       win32api.mouse_event(
                           win32con.MOUSEEVENTF_RIGHTUP, 0, 0)
                       bought_items.append(item[0])
                       money -= item[1]
                       time.sleep(0.1)
               pydirectinput.press('p')
               last_bought_timer = time.time()
               time.sleep(6)
               right_click(1869, 832)
               print('bought normally after backing')
               continue
           else:
               right_click(sia_x_move_forward, sia_y_move_forward)
```

```
35.                 right_click(sia_x_move_forward, sia_y_move_forward)
```

Isječak programskog koda 9. Provjera kupovine

I sada uspoređujemo 2 prethodno spomenuta timera te ponovno koristimo funkciju worth_recalling() i koristimo njene rezultate kao prije navedeno. U nastavku, pokazat ću ideju preko koje S.I.A. donosi odluke. Način na koji se sve to odvija se svodi na to je li postoji prijateljskih miniona ili ne. Ako su pronađeni, gledamo je li postoji tornjeva. Dodatna pitanja poslije toga su razne kombinacije neprijateljskih miniona i heroja te se sve svodi na to. Kod je jako dug te ću pokazati par bitnih stvari.

```
1.          if len(enemy_turret) or len(enemy_turret_plate):
2.              screenshot = window_capture.get_screenshot()
3.              enemy_minion = object_enemy_minion.find(
4.                  screenshot, 0.95)
5.              enemy_champion = object_enemy_champion.find(
6.                  screenshot, 0.95)
7.              # if no en. min and no en. champ.
8.              if not enemy_minion and not enemy_champion:
9.                  screenshot = window_capture.get_screenshot()
10.                   enemy_turret = object_enemy_turret.find(
11.                       screenshot, 0.95)
12.                   enemy_turret_plate =
    object_enemy_turret_plate.find(
13.                       screenshot, 0.95)
14.                   if len(enemy_turret):
15.                       right_click(enemy_turret[len(
16.                           enemy_turret)-1][0]+70,
    enemy_turret[len(enemy_turret)-1][1]+140)
17.
18.                   if len(enemy_turret_plate):
19.                       right_click(enemy_turret_plate[len(
20.                         enemy_turret_plate)-1][0]+70,
    enemy_turret_plate[len(enemy_turret_plate)-1][1]+140)
21.               continue
```

Isječak programskog koda 10. Slučaj toranj bez miniona i heroja

Ovo je jedan slučaj gdje ako postoje prijateljski minioni i neprijateljski toranj, skenirat ćemo opet za neprijateljske minione i heroje te u slučaju da ih nema napast ćemo toranj. Funkcija right_click() sama po sebi intuitivna, ali kasnije ćemo i nju objasniti. Način na koji dolazimo do svakog objekta kao ovdje do tornja je da mi točke koje dobijemo iz metode find(), to je lista te tako možemo indeksirati da jednostavno uzmemo dužinu te liste te zadnji element iz nje i pošto je HP linija malo iznad samog objekta, prilagodimo lokaciju ručno kako bi kliknuli izravno na sam objekt. I naravno continue, kako petlja ne bi išla dalje već što prije opet iznova detektirala.

```
1.              if len(enemy_champion) and not enemy_minion:
2.
3.                  distance = calculateDistance(sia_x, sia_y,
   enemy_champion[len(
4.                      enemy_champion)-1][0]+35,
   enemy_champion[len(enemy_champion)-1][1]+120)
5.                  if distance < 600:
6.                      win32api.SetCursorPos((enemy_champion[len(
7.                      enemy_champion)-1][0]+35,
   enemy_champion[len(enemy_champion)-1][1]+120))
8.                      pydirectinput.press('q')
9.                      time.sleep(0.1)
10.                       pydirectinput.press('e')
11.                 time.sleep(0.05)
12.                 indicated_auto_attack(enemy_champion[len(
13.                     enemy_champion)-1][0]+45,
   enemy_champion[len(enemy_champion)-1][1]+75)
14.                 # pydirectinput.press('s')
15.                 time.sleep(0.01)
16.                 low_hp_pixels =
   pyautogui.screenshot().getpixel((840, 1045))
17.                 if low_hp_pixels == (1, 13, 7):
18.                     time.sleep(0.05)
19.                     pydirectinput.press('w')
20.                     time.sleep(0.05)
21.                 screenshot = window_capture.get_screenshot()
22.                 enemy_champion =
   object_enemy_champion.find(screenshot, 0.92)
23.                 ultimate_pixels =
   pyautogui.screenshot().getpixel((955, 1001))
24.
25.                 if len(enemy_champion):
26.
27.                     if ultimate_pixels == (194, 69, 153):
28.                         win32api.SetCursorPos((enemy_champion[len(
29.                             enemy_champion)-1][0]+40,
   enemy_champion[len(enemy_champion)-1][1]+100))
30.                         pydirectinput.press('r')
31.                         time.sleep(0.1)
32.                         indicated_auto_attack(enemy_champion[len(
33.                             enemy_champion)-1][0]+45,
   enemy_champion[len(enemy_champion)-1][1]+75)
34.                 continue
```

Isječak programskog koda 11. Interakcija s herojem

Ovdje možemo vidjeti način na koji napadamo neprijateljskog heroja. Koristimo funkciju calculateDistance() koja nam kaže je li distanca dovoljno blizu da je vrijedno koristiti čarolije na neprijatelja. Ako jeste koristimo funkcije od biblioteke Pydirectinput kako bi koristili tipkovnicu i čarolije, funkciju indicated_auto_attack(), koja radi isto što i right_click(), ali malo je sigurnija na način postavljanja miša na lokaciju. Opet provjeravamo ako imamo mali HP, u

tom slučaju koristimo čaroliju W za povećanje HP-a te opet skeniramo za neprijateljske heroje kako bi dobili još svježiju sliku te ako imamo ultimativnu čaroliju dostupnu, iskoristit ćemo je.

```
1.              if not enemy_champion and len(enemy_minion):
2.                  right_click(ally_minion[len(ally_minion)-1]
3.                              [0]+30, ally_minion[len(ally_minion)-
   1][1]+60)
4.                  screenshot = window_capture.get_screenshot()
5.                  enemy_minion = object_enemy_minion.find(screenshot,
   0.92)
6.                  if len(enemy_minion):
7.                      right_click(enemy_minion[len(
8.                          enemy_minion)-1][0]+27,
   enemy_minion[len(enemy_minion)-1][1]+55)
9.                  continue
```

Isječak programskog koda 12. Interakcija s minionima

U interakciji s minionima (bez neprijateljskih heroja), isto kao i kod ostalih interakcija, u ovom slučaju malo ćemo bolje pozicionirati agenta iza prijateljskih miniona te dohvatiti svježu sliku ekrana i vidjeti gdje su neprijateljski minioni te onda ih napasti s right_click() i naposljetku continue.

```
1.              if len(enemy_minion) and len(enemy_champion):
2.                  if len(ally_minion) < len(enemy_minion)+1:
3.                          …
4.                  elif (len(enemy_minion) > len(ally_minion)+3):
5.                      print(
6.                          f"Len of enemy minions: {len(enemy_minion)},
   Len of ally minion: {len(ally_minion)}")
7.                      print('retreating!')
8.                      go_back()
9.                      continue
10.                 else: …
```

Isječak programskog koda 13. Interakcija s minionima i herojima

Ovdje su zatvorena 2 bloka jer je kod jako dug, u suštini ovdje imamo slučaj ako su tu neprijateljski i minioni i heroji. Tu ćemo jednostavno usporediti dužinu 2 liste: prijateljski i neprijateljski minioni. U slučaju da je broj približan (linija 2), napast ćemo heroja (ista interakcija kao i gore opisana), ako ima više neprijateljskih miniona (linija 4) onda ćemo se povući s funkcijom go_back(), a inače opet vršimo interakciju s neprijateljskim herojem.

```
    cv.imshow('League Bot', screenshot)

    # print('FPS {}'.format(1 / (time() - loop_time)))
    loop_time = time.time()

    if cv.waitKey(1) == 27:  # escape
        cv.destroyAllWindows()
        break

cv.destroyAllWindows()
```

Isječak programskog koda 14. Izlazak iz petlje

Poslije svega toga, imamo OpenCV funkciju imshow() koja prikazuje prozor koji se skenira i na njemu se crtaju pravokutnici, loop_time nam jednostavno služi kao pomagalo za FPS (engl. frames per second – broj slika u sekundi) te waitKey() funkcija kojom možemo završiti skeniranje te izaći iz petlje i zatvoriti otvorene prozore.

```
def left_click(x, y):
    win32api.SetCursorPos((x,y))
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN,0,0)
    time.sleep(0.01)
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,0,0)
    time.sleep(0.01)

def indicated_auto_attack(x, y):
    win32api.SetCursorPos((x,y))
      pydirectinput.press('a')
      win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN,0,0)
      time.sleep(0.01)
      win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,0,0)
      time.sleep(0.01)
      step_back()

 def right_click(x, y):
      win32api.SetCursorPos((x,y))
      win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTDOWN,0,0)
      time.sleep(0.05)
      win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTUP,0,0)
      time.sleep(0.05)
```

Isječak programskog koda 15. Pomoćne funkcije 1. dio

I ovdje napokon vidimo funkcije koje se koriste dosta puta u cijelom kodu. Intuitivne su same po sebi, ali tehnički koristimo win32api biblioteku preko kojeg postavljamo poziciju miša na dane koordinate, te onda ovisno o lijevom ili desnom kliku koristimo odgovarajuće događaje za miš. Funkcija sleep() je tu kako računalna igra ne bi otkrila čudna ponašanja te je isto da

događaji bolje rade. Funkcija indicated_auto_attack() je sigurnija na način napadanja protivnika jer iako koordinate nisu najtočnije ona će koristiti indikator s tipkom „a“ u računalnoj igri te opet vršiti događaj s mišem.

```
1. def calculateDistance(x1,y1,x2,y2):
2.     dist = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
3.     return dist
4.
5. def go_back(action=None):
6.     right_click(887-500, 496+400)
7.     time.sleep(2.2)
8.
9.     if action is not None:
10.          right_click(887-500, 496+400)
11.
12.          time.sleep(2.2)
13.          right_click(887-500, 496+400)
14.          time.sleep(2.2)
15.          pydirectinput.press('b')
16.          time.sleep(12)
17.          right_click(1869, 832)
18.
19.  def step_back():
20.      right_click(887-130, 496+200)
21.      time.sleep(0.5)
```

Isječak programskog koda 16. Pomoćne funkcije 2. dio

Još par pomoćnih funkcija gdje calculateDistance() kao prije navedeno, pomaže nam odrediti distancu između položaja od S.I.A. te neprijateljskog heroja. Go_back() koristi klikove miša kako bi se povukli iz borbe ili ako želimo ići nazad u bazu da budemo na sigurnijem mjesto prije no što pozovemo liniju 15. koja zahtjeva da naš lik stoji u mjestu neometan od strane neprijatelja. Funkcija step_back() je mali pomak unazad za bolje, rafinirano pozicioniranje. Ove kretnje su relativne od pozicije S.I.A.

I tako smo opisali cijeli kod te ustanovili što on točno radi. U nastavku je link gdje možemo vidjeti kako S.I.A. igra, iako rezultati nisu najbolji, vidimo da je automatizirano dosta radnji te je detekcija poprilično uspješna. Brzina detekcije isto ovisi i o brzini računala te je to isto bio ovdje faktor.

Link: https://youtu.be/PpsOTzRM0ek

6. Zaključak

Inteligentni agenti koji igraju računalne igre postaju sve više i više popularniji i samim tim što postoji mnoštvo koje igra računalne igre te u početku samog igranja su im potrebne igre protiv botova (inteligentnih agenata) koji će im olakšati upoznavanje s igrom te nekad žele da trivijalne levele/akcije prelaze botovi te im tako uštede vremena u slučaju da su već iskusniji igrači.

Tema ovog rada je bila pokazati na koji način oni rade te samu težinu izrade jednog. Python je svemoćan programski jezik, iako možda nije najbrži u usporedbi s kompajlerima poput C/C++, veoma je dobar i koristan kada dođe do pitanja automatizacije zadataka i umjetne inteligencije. Sam proces je zahtijevao mnogo vremena, ali dosta sam i naučio kroz to vrijeme. Vjerujem da je sama detekcija te algoritam na koji agent radi dobro napravljen, a isto tako svezi protivničkih čarolija, moglo bi se unaprijediti glede izmicanja iz linije napada, koje bi zahtijevalo dosta jako računalo i mnogo slika u sekundi. Glede toga, mislim da bi konkurencija, tj. Dretve i multiprocesi mogli biti bolji u odnosu na sekvencijalan rad. Korisnost inteligentnih agenata se povećava svakim danom i ne samo u računalnim igrama.

6. Popis literature

[1] Artificial Intelligence: A Modern Approach (4th Edition), S. Russell, P. Norvig (2020.)

[2] „Zašto je UI bitna?“ [Na internetu]

Dostupno: https://www.stateofai2019.com/chapter-2-why-is-ai-important/

[Pristupano 29.6.2022.]

[3] „Povijest UI“ [Na internetu]

Dostupno: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence

[Pristupano 30.6.2022.]

[4] „Industrija i UI“ [Na internetu]

Dostupno: https://csuglobal.edu/blog/why-ai-important

[Pristupano 30.6.2022.]

[5] „Značajnost UI u industriji“ [Na internetu]

Dostupno: https://www.leewayhertz.com/ai-applications-across-major-industries/

[Pristupano 1.7.2022.]

[6] „Strojno učenje i UI“ [Na internetu]

Dostupno: https://www.analyticsinsight.net/the-difference-between-artificial-intelligence-and-machine-learning

[Pristupano 1.7.2022.]

[7] „Razlika UI i strojno učenje“ [Na internetu]

Dostupno: https://azure.microsoft.com/en-us/overview/artificial-intelligence-ai-vs-machine-learning/#introduction

[Pristupano 1.7.2022.]

[8] „Gary Bradski, Adrian Kaehler - Learning OpenCV - O'Reilly“ (2008.)

[9] „Intelligent Systems for Engineers and Scientists - A Practical Guide to Artificial Intelligence - CRC Press“ (2021.)

[10] „Paul Roberts - Artificial Intelligence in Games-CRC Pr I Llc“ (2022.)

[11] „Playing Smart On Games, Intelligence, and Artificial Intelligence -The MIT Press“ (2018.)

[12] „Pradeepta Mishra - Practical Explainable AI Using Python_ Artificial Intelligence Model Explanations Using Python-based Libraries, Extensions, and Frameworks-Apress“ (2022.)

[13] „Learning OpenCV 4 Computer Vision with Python 3 Get to grips with tools, techniques, and algorithms for computer vision (Joseph Howse, Joe Minichino) „ (2020.)

[14] OpenCV biblioteka [Na internetu]

Dostupno: https://opencv.org/about

[Pristupano 22.7.2022.]

[15] IBM Deep Blue [Na internetu]

Dostupno: https://www.techopedia.com/definition/31625/deep-blue

[Pristupano 22.7.2022.]

[16] MOBA pojam [Na internetu]

Dostupno: https://www.dictionary.com/browse/moba

[Pristupano 22.7.2022.]

[17] NPC pojam [Na internetu]

Dostupno: https://www.techopedia.com/definition/1920/non-player-character-npc

[Pristupano 22.7.2022.]

[18] Programski kod klase WindowCapture [Na internetu]

Dostupno: https://www.coursehero.com/file/77088162/moebot2py

[Pristupano 22.7.2022.]

[19] Riot Games te nastanak League of Legends [Na internetu]

Dostupno: https://www.unrankedsmurfs.com/blog/what-is-league-of-legends

[Pristupano 25.7.2022.]

[20] Kategorije heroja LoL-a [Na internetu]

Dostupno: https://www.hotspawn.com/league-of-legends/guides/what-is-league-of-legends

[Pristupano 25.7.2022.]

[21] OpenCV primjer [Na internetu]

Dostupno: https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

[Pristupano 05.08.2022.]

[22] Primjeri UI u svakodnevnom životu [Na internetu]

Dostupno: https://geekflare.com/daily-life-ai-example/

[Pristupano [10.8.2022.]

[23] Referenca WIN32API [Na internetu]

Dostupno: https://docs.microsoft.com/en-us/windows/win32/api/

[Pristupano 10.8.2022.]

[24] Referenca PyDirectInput [Na internetu]

Dostupno: https://pypi.org/project/PyDirectInput/

[Pristupano 10.8.2022.]

[25] Referenca PyAutoGUI [Na internetu]

Dostupno: https://pyautogui.readthedocs.io/en/latest/index.html

[Pristupano 10.8.2022.]

[26] Referenca NumPy [Na internetu]

Dostupno: https://numpy.org/doc/stable/user/absolute_beginners.html

[Pristupano 10.8.2022.]

[27] Referenca pytesseract [Na internetu]

Dostupno: https://pypi.org/project/pytesseract/

[Pristupano 10.8.2022.]

[28] Rana povijest UI [Na internetu]

Dostupno: https://www.ijcai.org/Proceedings/77-2/Papers/083.pdf

[Pristupano 11.8.2022.]

[29] Povijest UI – DeepBlue [Na internetu]

Dostupno: https://www.greentropism.com/general-news/deepblue-a-turnpoint-in-the-history-of-artificial-intelligence/

[Pristupano 11.8.2022.]

[30] UI u zdravstvu [Na internetu]

Dostupno: https://www.usa.edu/blog/how-ai-is-revolutionizing-healthcare/

[Pristupano 11.8.2022.]

[31] UI u maloprodaji i trgovini [Na internetu]

Dostupno: https://global.hitachi-solutions.com/blog/ai-in-retail/

[Pristupano 11.8.2022.]

[32] UI u bankarstvu [Na internetu]

Dostupno: https://www.businessinsider.com/ai-in-banking-report

[Pristupano 11.8.2022.]

[33] UI u zabavi i igrama [Na internetu]

Dostupno: https://www.arm.com/glossary/ai-in-gaming

[Pristupano 11.8.2022.]

Popis slika

Prilozi (main.py, captureWindow.py, objectFinder.py)

1. Izvorni kod main.py

```
import cv2 as cv

import time
from captureWindow import WindowCapture
import pydirectinput
from objectFinder import ObjectFinder
from utils import *
import random
import pytesseract
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'

bought_items = []

time.sleep(15)

window_capture = WindowCapture()
window_capture.list_window_names()

sia_x, sia_y = 887, 496
sia_x_move_forward, sia_y_move_forward = sia_x+170, sia_y-120

object_level_up = ObjectFinder('images/levelUP.png')
object_enemy_turret = ObjectFinder('images/enemyTurretHP.png')
object_enemy_minion = ObjectFinder('images/enemyMinionHP.png')
object_enemy_champion = ObjectFinder('images/enemyHP.png')
object_enemy_turret_plate = ObjectFinder('images/enemyTurretHPplate.png')
object_ally_minion = ObjectFinder('images/alliedMinionHP.png')
object_ally_champion = ObjectFinder('images/alliedChampion.png')

pydirectinput.press('p')
right_click(341, 329)
pydirectinput.press('p')
bought_items.append('Amplifying tome')
last_bought_timer = time.time()


pydirectinput.press('y')
right_click(1755, 948)

loop_time = time.time()
```

```
while(True):
    dead_pixels = pyautogui.screenshot().getpixel((691, 1002))
    if dead_pixels == (184, 184, 184):
        while dead_pixels == (184, 184, 184):
            dead_pixels = pyautogui.screenshot().getpixel((691, 1002))

        else:
            is_worth_recalling, money = worth_recalling()

            if is_worth_recalling:
                pydirectinput.press('p')

                for item in list_of_items:
                    if not item[0] in bought_items and item[1] < money:
                        x, y = item[2]
                        win32api.SetCursorPos((x, y))
                        win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTDOWN
, 0, 0)
                        time.sleep(0.1)
                        win32api.mouse_event(win32con.MOUSEEVENTF_RIGHTUP,
0, 0)
                        bought_items.append(item[0])
                        money -= item[1]
                        time.sleep(0.1)
                pydirectinput.press('p')
                last_bought_timer = time.time()
                time.sleep(6)
            right_click(1869, 832)
            time.sleep(0.01)
            pydirectinput.press('f')
            time.sleep(0.01)
            continue


    ultimate_pixels = pyautogui.screenshot().getpixel((955, 1001))
    low_hp_pixels = pyautogui.screenshot().getpixel((840, 1045))
    if low_hp_pixels == (1, 13, 7):
        time.sleep(0.05)
        pydirectinput.press('w')
        time.sleep(0.05)
        pydirectinput.press('d')
        time.sleep(0.05)
        low_hp_pixels = pyautogui.screenshot().getpixel((840, 1045))
        if low_hp_pixels == (1, 13, 7):
            print('low')
            go_back('recalling')

    screenshot = window_capture.get_screenshot()
    level_up = object_level_up.find(screenshot, 0.9)
```

```
if len(level_up) > 0:
    if len(level_up) == 1:
        left_click(level_up[0][0], level_up[0][1])
    if len(level_up) == 2:
        left_click(level_up[random.randint(0, 1)][0],
                   level_up[random.randint(0, 1)][1])
    if len(level_up) == 3:
        left_click(level_up[random.randint(0, 2)][0],
                   level_up[random.randint(0, 2)][1])
    if len(level_up) == 4:
        left_click(level_up[random.randint(0, 3)][0],
                   level_up[random.randint(0, 3)][1])
    win32api.SetCursorPos((sia_x, sia_y))

ally_minion = object_ally_minion.find(
    screenshot, 0.93)
enemy_minion = object_enemy_minion.find(
    screenshot, 0.93)
enemy_turret = object_enemy_turret.find(
    screenshot, 0.93)
enemy_turret_plate = object_enemy_turret_plate.find(
    screenshot, 0.93)
enemy_champion = object_enemy_champion.find(
    screenshot, 0.95)
go_buy_timer = time.time()

if go_buy_timer - last_bought_timer > 270:
    print(f"{go_buy_timer-last_bought_timer}")
    go_back()
    time.sleep(2)
    is_worth_recalling, money = worth_recalling()
    if is_worth_recalling:
        right_click(887-500, 496+450)
        time.sleep(3)
        right_click(887-115, 496+300)
        time.sleep(3)
        pydirectinput.press('b')
        time.sleep(9)
        pydirectinput.press('s')
        pydirectinput.press('p')
        for item in list_of_items:
            if not item[0] in bought_items and item[1] < money:
                x, y = item[2]
                win32api.SetCursorPos((x, y))
                win32api.mouse_event(
                    win32con.MOUSEEVENTF_RIGHTDOWN, 0, 0)
                time.sleep(0.1)
                win32api.mouse_event(
                    win32con.MOUSEEVENTF_RIGHTUP, 0, 0)
```

```
                    bought_items.append(item[0])
                    money -= item[1]
                    time.sleep(0.1)
            pydirectinput.press('p')
            last_bought_timer = time.time()
            time.sleep(6)
            right_click(1869, 832)
            print('bought normally after backing')
            continue
        else:
            right_click(sia_x_move_forward, sia_y_move_forward)
            right_click(sia_x_move_forward, sia_y_move_forward)



    if len(ally_minion):
        screenshot = window_capture.get_screenshot()
        enemy_turret = object_enemy_turret.find(
            screenshot, 0.95)
        enemy_turret_plate = object_enemy_turret_plate.find(
            screenshot, 0.95)
        # if see turret
        if len(enemy_turret) or len(enemy_turret_plate):
            screenshot = window_capture.get_screenshot()
            enemy_minion = object_enemy_minion.find(
                screenshot, 0.95)
            enemy_champion = object_enemy_champion.find(
                screenshot, 0.95)
            # if no en. min and no en. champ.
            if not enemy_minion and not enemy_champion:
                screenshot = window_capture.get_screenshot()
                enemy_turret = object_enemy_turret.find(
                    screenshot, 0.95)
                enemy_turret_plate = object_enemy_turret_plate.find(
                    screenshot, 0.95)
                if len(enemy_turret):
                    right_click(enemy_turret[len(
                        enemy_turret)-1][0]+70,
enemy_turret[len(enemy_turret)-1][1]+140)

                if len(enemy_turret_plate):
                    right_click(enemy_turret_plate[len(
                        enemy_turret_plate)-1][0]+70,
enemy_turret_plate[len(enemy_turret_plate)-1][1]+140)
                continue
            # if en. min. and no en. champ.
            if len(enemy_minion) > 2 and not enemy_champion:
                step_back()
                step_back()
```

```
                screenshot = window_capture.get_screenshot()
                enemy_minion = object_enemy_minion.find(screenshot, 0.92)
                if len(enemy_minion):
                    right_click(enemy_minion[len(
                        enemy_minion)-1][0]+27,
enemy_minion[len(enemy_minion)-1][1]+55)
                continue
            if len(enemy_minion) and len(enemy_champion):
                go_back()
                continue
        else:
            # if en. champ. and no en. min.
            if len(enemy_champion) and not enemy_minion:

                distance = calculateDistance(sia_x, sia_y,
enemy_champion[len(
                    enemy_champion)-1][0]+35,
enemy_champion[len(enemy_champion)-1][1]+120)
                if distance < 600:
                    win32api.SetCursorPos((enemy_champion[len(
                    enemy_champion)-1][0]+35,
enemy_champion[len(enemy_champion)-1][1]+120))
                    pydirectinput.press('q')
                    time.sleep(0.1)
                    pydirectinput.press('e')
                time.sleep(0.05)
                indicated_auto_attack(enemy_champion[len(
                    enemy_champion)-1][0]+45,
enemy_champion[len(enemy_champion)-1][1]+75)
                # pydirectinput.press('s')
                time.sleep(0.01)
                low_hp_pixels = pyautogui.screenshot().getpixel((840,
1045))
                if low_hp_pixels == (1, 13, 7):
                    time.sleep(0.05)
                    pydirectinput.press('w')
                    time.sleep(0.05)
                screenshot = window_capture.get_screenshot()
                enemy_champion = object_enemy_champion.find(screenshot,
0.92)
                ultimate_pixels = pyautogui.screenshot().getpixel((955,
1001))

                if len(enemy_champion):

                    if ultimate_pixels == (194, 69, 153):
                        win32api.SetCursorPos((enemy_champion[len(
                            enemy_champion)-1][0]+40,
enemy_champion[len(enemy_champion)-1][1]+100))
```

```
                        pydirectinput.press('r')
                        time.sleep(0.1)
                        indicated_auto_attack(enemy_champion[len(
                            enemy_champion)-1][0]+45,
enemy_champion[len(enemy_champion)-1][1]+75)
                continue
            # if no en. champ. and en. min.
            if not enemy_champion and len(enemy_minion):
                right_click(ally_minion[len(ally_minion)-1]
                            [0]+30, ally_minion[len(ally_minion)-1][1]+60)
                screenshot = window_capture.get_screenshot()
                enemy_minion = object_enemy_minion.find(screenshot, 0.92)
                if len(enemy_minion):
                    right_click(enemy_minion[len(
                        enemy_minion)-1][0]+27,
enemy_minion[len(enemy_minion)-1][1]+55)
                continue
            # if en. min and en. champ.
            if len(enemy_minion) and len(enemy_champion):
                if len(ally_minion) < len(enemy_minion)+1:
                    right_click(
                        ally_minion[len(ally_minion)-1][0]+30,
ally_minion[len(ally_minion)-1][1]+60)
                    screenshot = window_capture.get_screenshot()
                    enemy_champion = object_enemy_champion.find(
                        screenshot, 0.95)
                    if len(enemy_champion):

                        distance = calculateDistance(sia_x, sia_y,
enemy_champion[len(
                            enemy_champion)-1][0]+35,
enemy_champion[len(enemy_champion)-1][1]+100)
                        if abs(distance) < 600:
                            win32api.SetCursorPos((enemy_champion[len(
                            enemy_champion)-1][0]+35,
enemy_champion[len(enemy_champion)-1][1]+100))
                            pydirectinput.press('q')
                            time.sleep(0.1)
                            pydirectinput.press('e')

                        time.sleep(0.05)
                        step_back()
                    low_hp_pixels = pyautogui.screenshot().getpixel((840,
1045))
                    if low_hp_pixels == (1, 13, 7):
                        time.sleep(0.05)
                        pydirectinput.press('w')
                        time.sleep(0.05)
                    time.sleep(0.01)
```

```
                    screenshot = window_capture.get_screenshot()
                    enemy_champion = object_enemy_champion.find(
                        screenshot, 0.95)
                    ultimate_pixels = pyautogui.screenshot().getpixel((955,
1001))
                    if len(enemy_champion):

                        if ultimate_pixels == (194, 69, 153):
                            win32api.SetCursorPos((enemy_champion[len(
                                enemy_champion)-1][0]+40,
enemy_champion[len(enemy_champion)-1][1]+100))
                            pydirectinput.press('r')
                            time.sleep(0.1)
                            indicated_auto_attack(enemy_champion[len(
                                enemy_champion)-1][0]+45,
enemy_champion[len(enemy_champion)-1][1]+75)
                        continue
                elif (len(enemy_minion) > len(ally_minion)+3):
                    print(
                        f"Len of enemy minions: {len(enemy_minion)}, Len of
ally minion: {len(ally_minion)}")
                    print('retreating!')
                    go_back()
                    continue
                else:
                    right_click(ally_minion[0][0]+30, ally_minion[0][1]+60)
                    time.sleep(0.1)
                    screenshot = window_capture.get_screenshot()
                    enemy_champion = object_enemy_champion.find(
                        screenshot, 0.95)
                    if len(enemy_champion):
                        win32api.SetCursorPos((enemy_champion[len(
                            enemy_champion)-1][0]+40,
enemy_champion[len(enemy_champion)-1][1]+100))
                        pydirectinput.press('q')
                        time.sleep(0.01)
                        pydirectinput.press('e')

                        indicated_auto_attack(enemy_champion[len(
                            enemy_champion)-1][0]+40,
enemy_champion[len(enemy_champion)-1][1]+100)
                        time.sleep(0.05)

                        print('s pressed in else')
                        screenshot = window_capture.get_screenshot()
                        enemy_champion = object_enemy_champion.find(
                            screenshot, 0.95)
                        ultimate_pixels =
pyautogui.screenshot().getpixel((955, 1001))
```

```
                        if len(enemy_champion):

                            if ultimate_pixels == (194, 69, 153):
                                win32api.SetCursorPos((enemy_champion[len(
                                    enemy_champion)-1][0]+40,
enemy_champion[len(enemy_champion)-1][1]+100))
                                pydirectinput.press('r')
                                time.sleep(0.05)
                                indicated_auto_attack(enemy_champion[len(
                                    enemy_champion)-1][0]+45,
enemy_champion[len(enemy_champion)-1][1]+100)
                    continue

            if not enemy_champion and not enemy_minion:
                screenshot = window_capture.get_screenshot()
                enemy_minion = object_enemy_minion.find(screenshot, 0.92)
                enemy_champion = object_enemy_champion.find(screenshot,
0.92)
                counter = 0
                while len(enemy_minion) < 1 or len(enemy_champion):
                    counter += 1
                    if counter == 20:
                        right_click(1869, 832)
                        print('breaking out of loop!')
                        break

                    right_click(sia_x_move_forward, sia_y_move_forward)
                    screenshot = window_capture.get_screenshot()
                    enemy_minion = object_enemy_minion.find(screenshot,
0.92)
                    enemy_champion = object_enemy_champion.find(
                        screenshot, 0.92)

                time.sleep(0.01)

                continue
    else:
        screenshot = window_capture.get_screenshot()
        enemy_turret = object_enemy_turret.find(
            screenshot, 0.92)
        enemy_turret_plate = object_enemy_turret_plate.find(
            screenshot, 0.92)
        # if see turret
        if len(enemy_turret) or len(enemy_turret_plate):
            go_back()
            continue
        else:
            screenshot = window_capture.get_screenshot()
            enemy_champion = object_enemy_champion.find(screenshot, 0.92)
```

```
            enemy_minion = object_enemy_minion.find(screenshot, 0.92)
            # if en. min. and en. champ.
            if len(enemy_minion) and len(enemy_champion):
                go_back()
                continue
            # if no en. min. and en. champ.
            if len(enemy_champion) and not enemy_minion:
                screenshot = window_capture.get_screenshot()
                enemy_champion = object_enemy_champion.find(screenshot,
0.95)
                # step_back()
                if len(enemy_champion):
                    win32api.SetCursorPos((enemy_champion[len(
                        enemy_champion)-1][0]+40,
enemy_champion[len(enemy_champion)-1][1]+120))
                    pydirectinput.press('q')
                    time.sleep(0.1)
                    pydirectinput.press('e')
                    time.sleep(0.05)

                # pydirectinput.press('s')
                time.sleep(0.01)
                screenshot = window_capture.get_screenshot()
                enemy_champion = object_enemy_champion.find(
                    screenshot, 0.92)
                ultimate_pixels = pyautogui.screenshot().getpixel((955,
1001))
                if len(enemy_champion):
                    if ultimate_pixels == (194, 69, 153):
                        win32api.SetCursorPos((enemy_champion[len(
                            enemy_champion)-1][0]+40,
enemy_champion[len(enemy_champion)-1][1]+100))
                        pydirectinput.press('r')
                        time.sleep(0.1)
                        indicated_auto_attack(enemy_champion[len(
                            enemy_champion)-1][0]+45,
enemy_champion[len(enemy_champion)-1][1]+75)
                continue
            # if en min. and no. en. champ.
            if len(enemy_minion):
                go_back()
                continue

    cv.imshow('League Bot', screenshot)

    # print('FPS {}'.format(1 / (time() - loop_time)))
    loop_time = time.time()

    if cv.waitKey(1) == 27:  # escape
```

```
            cv.destroyAllWindows()
            break

    cv.destroyAllWindows()
    2.
```

2. Izvorni kod objectFinder.py

```
import cv2 as cv
import numpy as np


class ObjectFinder:
    sample = None
    sample_width = 0
    sample_height = 0
    method = None
    def __init__(self, sample_path, method=cv.TM_CCOEFF_NORMED):
        self.sample = cv.imread(sample_path, cv.IMREAD_UNCHANGED)
        self.sample_width = self.sample.shape[1]
        self.sample_height = self.sample.shape[0]
        self.method = method

    def find(self, haystack_img, threshold=0.5):
        result = cv.matchTemplate(haystack_img, self.sample, self.method)
        locations = np.where(result >= threshold)
        locations = list(zip(*locations[::-1]))
        rectangles = []
        for loc in locations:
            rect = [int(loc[0]), int(loc[1]), self.sample_width,
self.sample_height]
            rectangles.append(rect)
            rectangles.append(rect)
        rectangles, weights = cv.groupRectangles(
            rectangles, groupThreshold=1, eps=0.5)
        points = []
        if len(rectangles):
            for (x, y, w, h) in rectangles:
                center_x = x + int(w/2)
                center_y = y + int(h/2)
                points.append((center_x, center_y))

        return points
```

3. Izvorni kod captureWindow.py

```
import numpy as np
import win32gui
import win32ui
import win32con

class WindowCapture:

    # properties
    w = 0
    h = 0
    hwnd = None
    cropped_x = 0
    cropped_y = 0
    def __init__(self, window_name=None):
        if window_name is None:
            self.hwnd = win32gui.GetDesktopWindow()
        else:
            self.hwnd = win32gui.FindWindow(
                None, window_name)
            if not self.hwnd:
                raise Exception('Window not found: {}'.format(window_name))
        self.w = 1920
        self.h = 1080

    def get_screenshot(self):
        wDC = win32gui.GetWindowDC(self.hwnd)
        dcObj = win32ui.CreateDCFromHandle(wDC)
        cDC = dcObj.CreateCompatibleDC()
        dataBitMap = win32ui.CreateBitmap()
        dataBitMap.CreateCompatibleBitmap(dcObj, self.w, self.h)
        cDC.SelectObject(dataBitMap)
        cDC.BitBlt((0, 0), (self.w, self.h), dcObj,
                   (self.cropped_x, self.cropped_y), win32con.SRCCOPY)
        signedIntsArray = dataBitMap.GetBitmapBits(True)
        img = np.fromstring(signedIntsArray, dtype='uint8')
        img.shape = (self.h, self.w, 4)
        dcObj.DeleteDC()
        cDC.DeleteDC()
        win32gui.ReleaseDC(self.hwnd, wDC)
        win32gui.DeleteObject(dataBitMap.GetHandle())
        img = img[..., :3]
        img = np.ascontiguousarray(img)
        return img

    @staticmethod
```

```
def list_window_names():
    def winEnumHandler(hwnd, ctx):
        if win32gui.IsWindowVisible(hwnd):
            print(hex(hwnd), win32gui.GetWindowText(hwnd))
    win32gui.EnumWindows(winEnumHandler, None)
```