ŠOLSKI CENTER CELJE
Srednja šola za kemijo, elektrotehniko in računalništvo

Kartanje z umetno inteligenco

raziskovalna naloga

Mentor:
mag. Boštjan Resinovič, univ. dipl. inž. rač in inf.

Avtorja:
Jan Zorko, R-4. a
Žiga Pušnik, R-4. a

Mestna občina Celje, Mladi za Celje

Laško, marec 2019

Povzetek

V tej raziskovalni nalogi je opisana umetna inteligenca in njena uporaba pri strojih, ki se učijo igrati igre. Izpostavljeno je igranje iger z umetno inteligenco na profesionalnem nivoju in poizkus ustvarjanja enostavne igre z igralnimi kartami. Stroj se uči pravila igre s pomočjo nevronske mreže in igranja proti samemu sebi.

Abstract

This research paper describes artificial intelligence (AI) and its use in machines learning to play games. The focus is on playing games against AI at professional levels and attempting to create a simplified card game for the AI to play. The AI learns by using a neural network and self-play.

Ključne besede

Umetna inteligenca, nevronska mreža, strojno učenje, Python, TensorFlow, AlphaZero, Unity, remi, šnops, igralne karte, igre.

Keywords

Artificial intelligence, neural network, machine learning, Python, TensorFlow, AlphaZero, Unity, rummy, schnapsen, playing cards, games.

Kazalo vsebine

Kazalo slik

Kazalo grafov

1 Uvod

Umetna inteligenca je v vsakdanjem življenju vse bolj pogosta. Objavlja se veliko raziskav na temo strojnega učenja programov, ki se približujejo človeškim zmožnostim in jih tudi presegajo. Za takšne programe se uporabljajo nevronske mreže, ki delujejo podobno kot človeški možgani.

1.1 Predstavitev problema

Strojno učenje se uporablja pri ustvarjanju naprednih stvari. Zdaj bi stopili korak nazaj in ustvarili igro z igralnimi kartami, pri kateri bi igralec igral proti umetni inteligenci (UI). UI bi se te igre naučila na podlagi pridobljenih podatkov in s tem poskušala premagati igralca.

1.2 Hipoteze

Postavila sva si hipoteze glede izdelave umetne inteligence in nevronske mreže. Postavljena je tudi hipoteza o odgovorih anketirancev na vprašanje v anketi.

Hipoteza 1: Za izdelavo umetne inteligence bo potrebno izvenšolsko znanje.

Hipoteza 2: Izdelava nevronske mreže bo mogoča s srednješolsko programersko in matematično izobrazbo.

Hipoteza 3: Večina anketirancev bo na vprašanje »Kaj je umetna inteligenca?« odgovorila z nejasnimi odgovori.

Hipoteza 4: Z optimizacijo konfiguracije nevronske mreže bo agent UI igral bolje.

1.3 Raziskovalne metode

Za raziskovanje sva uporabila metodo anketiranja, kjer sva anketirance vprašala o njihovem splošnem znanju o umetni inteligenci. Vprašala sva jih tudi po njihovem poznavanju nekaterih iger s kartami.

Preučila sva vse vire in primere na internetu, ki so pomagali pri razumevanju umetne inteligence in uporabi knjižnice TensorFlow.

Rezultate sva ugotavljala z eksperimentiranjem pri nastavitvah mreže.

1.4 Cilji

Cilj raziskovalne naloge je, da se ustvari aplikacija oziroma igra s kartami. V njej bi se igralec pomeril z umetno inteligenco.

Agent se, medtem ko igra sam proti sebi, nauči potek igre in potencialno doseže raven igranja, ki je vsaj primerljiva s povprečnim človekom, če ne boljša.

2 Teoretični del

2.1 Umetna inteligenca

Na področju računalništva in informatike se izraz umetna inteligenca (UI, ang. artificial intelligence) uporablja za inteligenco strojev in programske opreme v nasprotju s človeško, naravno inteligenco.

Obseg umetne inteligence je sporen. Nedavno je bila kot vrh UI videna prepoznava znakov, na primer črk. Danes pa je takšna tehnologija že tako pogosto uporabljena, da je postala rutinska in ni več videna kot UI.

Računalniška znanost opredeljuje raziskovanje UI kot študij inteligentnih agentov, to je naprav, ki zaznavajo svoje okolje in ravnajo na način, ki ima največjo verjetnost za doseganje danega cilja.

Kaplan in Haenlein UI natančneje opredeljujeta kot "zmožnost sistema, da pravilno interpretira zunanje podatke, da se iz takih podatkov uči in da ta nova znanja uporablja za fleksibilno prilagajanje in doseganje specifičnih ciljev in nalog."[1]

Pogovorno se izraz uporablja, kadar opisujemo stroj, ki posnema kognitivne funkcije, ki jih ljudje povezujemo z drugimi oblikami človeškega uma, kot sta učenje in reševanje problemov.

2.1.1 Delitev umetne inteligence

Kaplan in Haenlein sta področje UI razdelila na tri različne vrste sistemov[2]:

- Analitična UI ima samo lastnosti, ki spadajo pod kognitivno inteligenco. Ustvarja kognitivno sliko sveta in znanje iz preteklih izkušenj uporablja kot osnovo za prihodnje odločitve.
- Od človeka navdihnjena UI vsebuje elemente kognitivne in čustvene inteligence. Razume oziroma razlikuje človeška čustva in jih upošteva pri svojih odločitvah.
- Počlovečena UI kaže značilnosti vseh treh vrst inteligence: kognitivne, čustvene in socialne. Sposobna je samozavesti pri interakciji z okoljem.

[1, 2] A. Kaplan, M. Haenlein: Siri, Siri, in my hand: Who’s the fairest in the land?

2.2 Strojno učenje

Strojno učenje (ang. machine learning) predstavlja strojno pridobivanje znanja na podlagi izkušenj. Ne gre samo za preprosto pomnjenje informacij, temveč za iskanje pravil med učnimi podatki. Tako lahko dobimo odgovor tudi na vprašanje, ki ga v preteklosti še nismo razumeli. Eden izmed najbolj znanih predstavnikov strojnega učenja so nevronske mreže, ki sva jih uporabila tudi v sklopu te raziskovalne naloge.

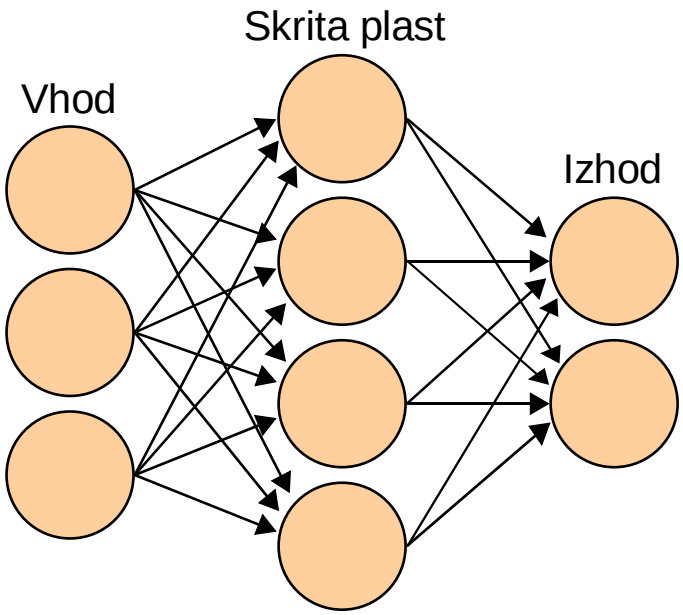
2.2.1 Nevronska mreža

Nevronska mreža je naprava oziroma program za obdelavo informacij, ki deluje po vzoru človeških možganov.

Sestavljena je iz mnogo umetnih nevronov, ki so razporejeni po plasteh. Število plasti in nevronov v vsaki plasti določi programer glede na kompleksnost naloge, vendar mora zelo paziti, da vseh nevronov ni premalo ali preveč.

Če je nevronov premalo, se nevronska mreža ne bo naučila določenih pravil, saj med različnimi vhodnimi podatki ne bo dovolj razlik, da bi lahko začela delati povezave med posameznimi podatki.

Če pa je nevronov preveč, si bo mreža lahko točno zapomnila, kateri vhodni podatki vodijo v katere želene izhodne podatke in se ne bo naučila splošnih pravil, ki bi jih potem lahko uporabila na novih vhodnih podatkih.



Slika 1: Nevronska mreža.

2.2.2 Vrste strojnega učenja

Strojno učenje lahko razdelimo na več vrst glede na nadzor:

1. nadzorovano učenje,
2. nenadzorovano učenje in
3. vzpodbujevalno učenje.

2.2.2.1 Nadzorovano strojno učenje

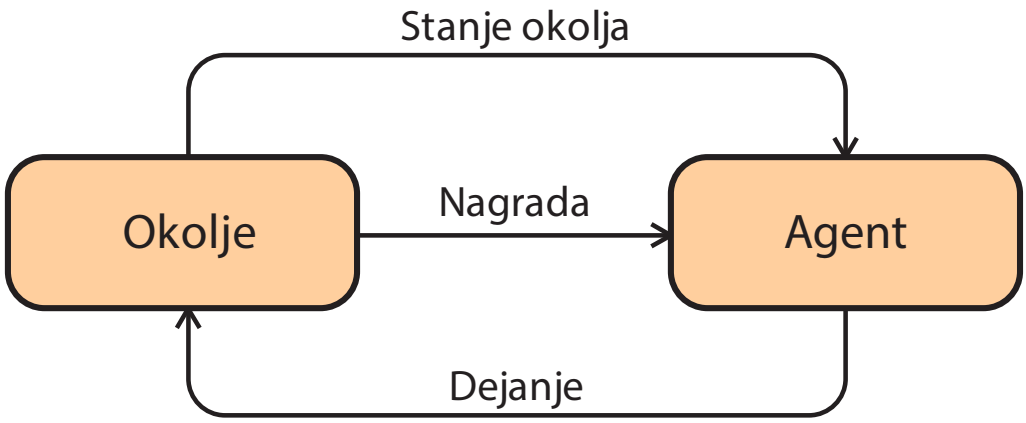
Pri nadzorovanem strojnem učenju (ang. supervised learning) algoritem uči agenta s podanimi pari vhodnih in želenih podatkov. Tukaj želene izhodne vrednosti določi učitelj – nadzornik, algoritem pa samo poskrbi, da se agent nauči, kako od vhodnih priti do želenih izhodnih podatkov.

2.2.2.2 Nenadzorovano strojno učenje

Pri nenadzorovanem učenju (ang. unsupervised learning) algoritem vhodne podatke razdeli po svojih kriterijih v več kategorij, ki imajo svoje značilnosti. To imenujemo rojenje (ang. clustering). Število kategorij in njihove značilnosti algoritem samodejno izlušči iz vhodnih podatkov brez nadzora učitelja. To naredi tako, da med vhodnimi podatki poišče strukture, ki se med podatki večkrat pojavijo, in jih nato poskuša najti tudi med novimi podatki.

2.2.2.3 Vzpodbujevalno strojno učenje

Pri vzpodbujevalnem učenju (ang. reinforcement learning) algoritem uči agenta na podlagi nagrad in kazni. Agent začne z naključnimi dejanji, ki vodijo v določena stanja, ki jim učitelj določi nagrade oziroma kazni. Nato stremi k izvajanju dejanj, ki mu na koncu prinesejo najvišjo skupno nagrado.



Slika 2: Potek vzpodbujevalnega učenja.

V primeru, ko imamo igro, pri kateri za vsako posamezno dejanje ne moremo podati nagrade oziroma kazni, saj v tistem trenutku ne vemo, če bo to dejanje vodilo k zmagi ali porazu, lahko nagrade in kazni določimo šele ob koncu igre. Ker so dejanja bliže končnemu izidu igre bolj pomembna, lahko za vsako dejanje nagrado pomnožimo s konstanto med nič in ena. Ta je pogosto kar ena polovica.

Funkcija za računanje te nagrade je splošno napisana tako:

$$l = \sum_{t} (v_\theta(s_t) - z_t)^2 - \vec{\pi} \cdot \log(\vec{p}_\theta(s_t))$$

V tej raziskovalni nalogi bo uporabljena oblika takšne vrste strojnega učenja.

2.2.3 AlphaZero

AlphaZero je računalniški program oziroma algoritem, razvit pri podjetju DeepMind, katerega lastnik je podjetje Alphabet Inc., ki si lasti tudi zelo znan Google LLC.

Razvit je bil po velikem uspehu svojega predhodnika, programa AlphaGo, ki je pri igranju igre go, ki naj bi bila številčno najkompleksnejša igra na svetu, štiri proti ena premagal večkratnega svetovnega prvaka (Lee Sedol).

AlphaGo je bil učen s tisočimi partijami človeških igralcev, tako da je vse svoje poteze zasnoval na tem, kar bi naredil človeški igralec, ampak že tukaj se je začela kazati nekakšna višja inteligenca. V drugi od petih iger proti svetovnem prvaku je AlphaGo v potezi sedemintrideset postavil svoj kamenček na pozicijo, ki je zmedla vse prisotne. Te pozicije nihče ni znal razložiti in nihče ni vedel, ali je dobra ali slaba. AlphaGo je igro zmagal in kamenček na tej poziciji mu je pri tem pomagal.

Kmalu po tekmovanju so raziskovalci pri DeepMind ustvarili novo različico algoritma: AlphaGo Zero. Ta se ni učila iz človeških iger, temveč je začela čisto na začetku. Igrala je sama proti sebi in se z vzpodbujevalnim učenjem v le dveh dneh naučila igrati tako dobro, da je AlphaGo premagala s končnim rezultatom sto proti nič.

AlphaGo Zero so raziskovalci posplošili, se pravi odstranili dele, specifične za igro go, in nastal je AlphaZero.

AlphaZero je program, ki se lahko brez človeškega znanja nauči nadčloveško dobro igrati katerokoli igro z dvema igralcema, ampak le, če je igra zasnovana tako, da imata oba igralca na voljo tako imenovano popolno informacijo. To pomeni, da oba dobita enake informacije, za razliko od na primer igre poker, ki temelji na skrivanju kart vsakega posameznega igralca.

Tu se je porodila ideja za to raziskovalno nalogo, saj če se lahko program sam nauči igrati eno najkompleksnejših iger na svetu, zakaj ne bi mogel igrati tudi kakšne preprostejše igre s kartami?

2.2.4 TensorFlow

TensorFlow je odprtokodna programska knjižnica za tako imenovano programiranje pretoka podatkov (ang. dataflow programming) in diferenčno programiranje.

Programiranje pretoka podatkov se ukvarja s podatki, ki se »pretakajo« med raznimi operacijami v grafu. Pri diferenčnem programiranju rezultati niso vedno deterministični, vendar nam program vrne razpon verjetnosti za različne vrednosti rezultata.

TensorFlow so razvili pri podjetju Google z namenom notranje uporabe, a so ga zaradi visoke učinkovitosti posplošili in odprli za javno rabo.

Najbolj se uporablja pri strojnem učenju, saj omogoča hitro in kvalitetno ustvarjanje in učenje nevronskih mrež. Danes so znanje in izkušnje s TensorFlow v industriji strojnega učenja in UI nujno potrebni, saj ga uporablja velika večina največjih podjetij na svetu.

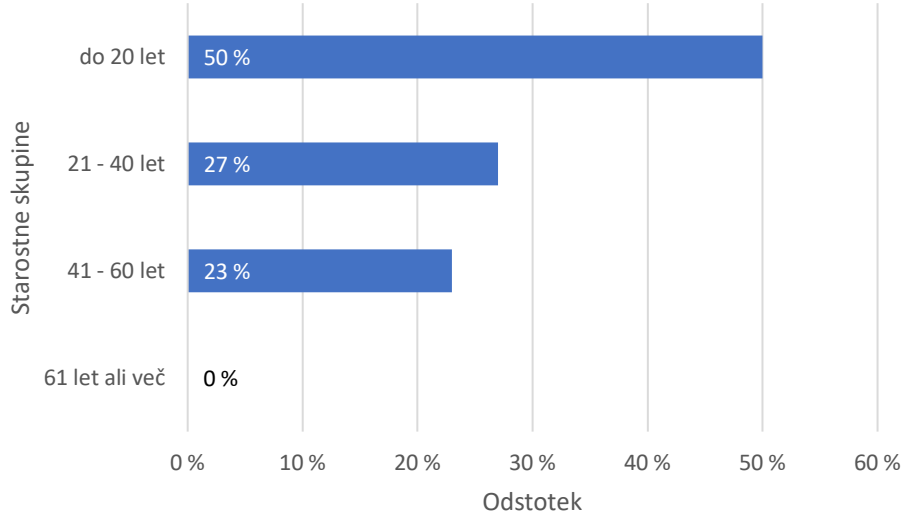
Zaradi takšne razširjenosti je TensorFlow spremenil področje strojnega učenja. Nedavno je bilo majhno in so ga zastopali samo raziskovalci, ki so mu posvetili svoje celotne kariere. Danes pa lahko teoretično vsak z osnovnim znanjem programiranja v le nekaj mesecih usvoji osnove strojnega učenja in ustvari program za svoje potrebe.

3 Praktični del

3.1 Anketiranje

Pri anketiranju je bil cilj ugotoviti poznavanje umetne inteligence, njenega razvoja in vpliva na človeštvo ter katere od navedenih iger z igralnimi kartami so najbolj znane. Anketirancev je bilo 30, iz različnih starostnih skupin, od tega polovica moških in polovica žensk. Ker je bila polovica anketirancev mlajših od 20 let, sva predvidevala, da so odgovarjali večinoma dijaki.

Graf 1: Starostne skupine anketirancev.



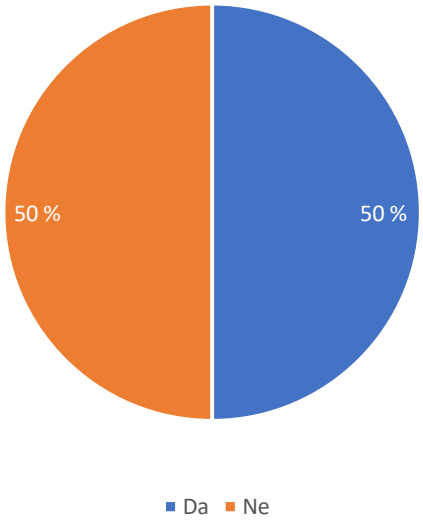
Pričakovalo se je, da bodo odgovori na vprašanja o umetni inteligenci netočni ali nejasni. Izkazalo se je, da večina anketirancev misli, da je UI nekakšen računalnik, ki prepoznava vzorce in se uči s pridobivanjem podatkov. Drugi menijo, da je UI robot ali stroj, ki posnema človeka in njegovo razmišljanje. Ostali pa menijo, da so to možgani robotov, ki so podobni človeškim.

Potem sva anketirance vprašala, kaj menijo o razvoju umetne inteligence, kako bo vplivala na življenje ljudi in ali bo prevladala nad nami ali ne. Presenetljivo jih je zadnje vprašanje 50 % odgovorilo z »da«.

Tisti, ki so odgovorili z »da«, so mnenja, da bo umetna inteligenca prepametna in je ne bo mogoče ustaviti. Poleg tega postaja vsakdanja stvar, brez katere je težko živeti.

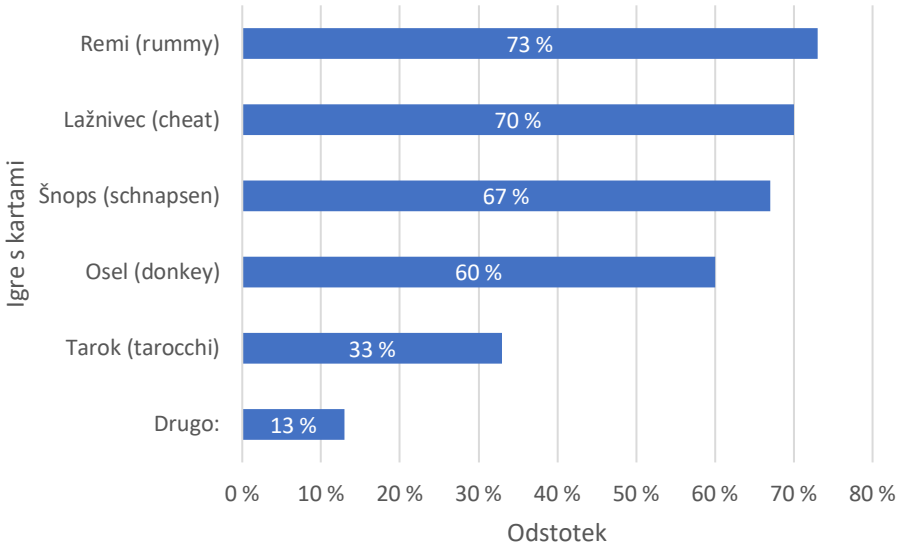
Tisti, ki so odgovorili z »ne«, pa so mnenja, da se UI ne more naučiti čustvovanja. Njen razvoj naj bi bil s tem ustavljen oziroma upočasnjen, zato ni potrebe za skrb.

Graf 2: Ali se bo UI razvila do te mere, da bo prevladala nad nami?

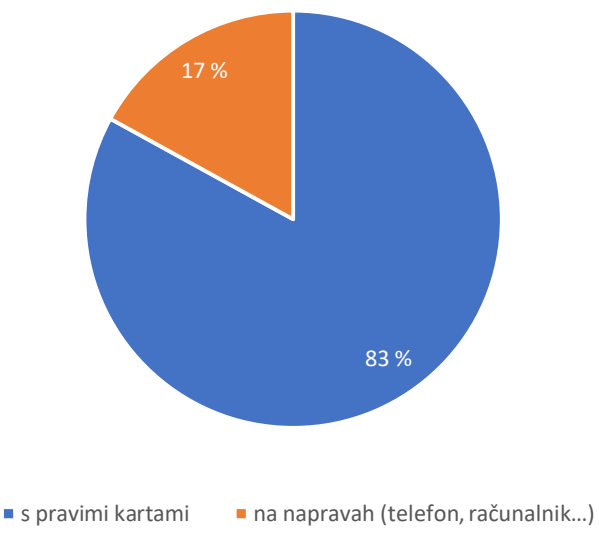


Pri anketiranju je bilo tudi vprašanje o igrah z igralnimi kartami in na kakšen način so jih igrali. Razvidno je bilo, da je najpopularnejša igra remi, tej pa sledita lažnivec in šnops ter ostale igre. Izvedelo sva tudi, da so anketiranci te igre igrali s pravimi igralnimi kartami, bilo pa je tudi nekaj takšnih, ki so te igre preigrali na svojem telefonu, računalniku ali na kakšni drugi napravi.

Graf 3: Katere igre s kartami ste že igrali?



Graf 4: Na kakšen način ste igrali igre s kartami?



Z analizo ankete je bilo ugotovljeno, da je najpopularnejša igra remi. Tako sva se odločila, da bova naredila igro remija, pri kateri bo igralec igral proti umetni inteligenci.

3.2 Težave pri implementaciji remija

Po izdelavi ogrodja igre so se kaj hitro pojavile težave. Od začetka sva bila še optimistična in sva si rekla, da jih bova že kako obšla. Izkazalo pa se je, da so bile težave veliko večje, kot so se zdele na prvi pogled.

Tako kot moramo določiti vhodne podatke, moramo določiti tudi izhodne. Izhodni podatki so običajno v obliki enodimenzionalnega vektorja, na primer 1 x 13, če bi radi, da nam nevronska mreža izbere eno vrednost kart. Tu se pojavi prva težava, in sicer to, da nevronska mreža le izbere mesto v vektorju, tja pa ne vstavi nikakršnega števila. Mreža nam sicer lahko vrne vektor vrednosti med 1 in 0, a to nam samo v odstotkih pove, kako prepričana je v izbiro.

Ker je remi bolj kompleksna igra, nam samo izbira karte ne zadošča. Najprej moramo izbrati, ali vlečemo karto iz kupčka za odlaganje ali iz glavnega kupčka, nato pa se moramo za vsako karto odločiti, ali bomo z njo igrali ali ne. Če bomo igrali, se moramo odločiti, kam jo bomo dali, saj karte tvorijo več različnih veljavnih skupin. Srčevo trojko lahko na primer igramo skupaj z dvema trojkama drugih barv ali pa v srčevi lestvici, ki se že sama lahko začne s katerokoli karto, le da na koncu vsebuje tudi trojko.

Hitro se izkaže, da je možnosti veliko preveč. Četudi bi imeli znanje in bi lahko tako kompleksno mrežo implementirali, bi imeli velik problem pri moči strojne opreme.

Že preproste mreže se morajo na osebnih računalnikih učiti več ur ali celo dni, potreben čas za učenje pa s kompleksnostjo mreže raste eksponentno. AlphaZero se je naučil igrati igro go v le treh dneh, a je to naredil s pomočjo več kot pet tisoč za strojno učenje specializiranih procesorskih jeder. Učenje lahko pospešimo tako, da ga izvedemo na grafični procesni enoti, a tudi ta ni tako močna kot specializirana jedra.

Zaradi tega sva sprejela odločitev, da bova raziskovalno nalogo osredotočila na drugo igro.

3.3 Izbira druge igre

Naslednja na listi najpopularnejših iger je bila lažnivec. Preden je bila izbrana za poustvaritev, je bilo ugotovljeno, da je prisotna uporaba čustev, ki jih kaže igralec. Zaradi tega sva lažnivca preskočila, ker je UI težko naučiti uporabljati čustva.

Tako sva za naslednjo igro izbrala šnops, kjer čustva niso neposredno prisotna. Odločila sva se, da bova poskusila narediti igro šnopsa proti UI.

3.4 Šnops

Šnops je igra s kartami, ki izvira iz nemških dežel. Ime naj bi dobil po istoimenski žgani pijači. Nastal je v 17. stoletju in se ga igra z nemškimi in francoskimi kartami. Za to igro bodo uporabljene francoske karte (srca, kara, pik, križ).

3.4.1 Pravila

Igro šnopsa lahko igrajo štirje ali trije igralci, lahko pa sta tudi samo dva. Ker je najina igra zasnovana tako, da človeški igralec igra proti UI, sva se poglobila v pravila za dva igralca. Uporabljajo se karte od desetke do kralja in asi, torej 20 kart. Pravila:

- Vsakemu igralcu se razdeli 5 kart.
- Naslednjo karto položiš z licem navzgor na mizo, njena barva je adut (glavna barva).
- Kup preostalih devetih kart postavimo na aduta tako, da se ga vidi.
- Igralec, ki ni delil, prvi izbere eno karto iz roke in jo položi na mizo.
- Tudi drugi igralec izbere eno karto in jo položi na mizo.
- Rundo zmaga igralec z močnejšo karto:
    - adut ima moč nad vsemi drugimi barvami,
    - če sta barvi enaki, se gleda vrednost kart,
    - če ni uporabljen adut in se barvi razlikujeta, zmaga tisti, ki je prvi položil karto na mizo.

- Igralec z močnejšo karto vzame obe karti in prištejejo se mu pike v vrednosti obeh kart:
  - as ima vrednost 11 pik,
  - desetka ima vrednost 10 pik,
  - kralj ima vrednost 4 pik,
  - dama ima vrednost 3 pik,
  - fant ima vrednost 2 pik.
- Zmagovalec runde prvi vleče eno karto iz kupčka, drugi igralec mu sledi:
  - igralca morata imeti v rokah pet kart (dokler se kupček ne razdeli).
- Naslednjo rundo začne zmagovalec prejšnje.
- Če ima igralec, ki prvi da karto na mizo, v roki kraljevi par, lahko razglasi poroko:
  - če ima kraljevi par, ki je adut, dobi 40 pik,
  - če ima kraljevi par drugih barv, dobi 20 pik,
  - pike od poroke se štejejo, če je igralec zmagal vsaj eno rundo.
- Ko je še kupček kart in adut pod njim, lahko igralec, ki začne rundo:
  - zamenja karto pod kupčkom, če ima v rokah fanta, ki je adut,
  - zapre kupček z adutom pod njim in se začne »dvoboj« (v kupčku morata biti več kot dve karti).
- Ko je kupček razdeljen ali ko ga igralec zapre, se začne »dvoboj«:
  - potrebno je slediti barvi,
  - če ni mogoče slediti barvi, je potrebno dati ven aduta,
  - če aduta ni, se lahko da katerakoli karta,
  - zadnji, ki zmaga rundo, dobi eno točko in s tem je konec igre.
- Če igralec zapre kupček in mu ne uspe priti do 66 pik, dobi nasprotnik dve točki
- Igro zmaga igralec, ki prvi pride do 66 pik
  - če ima poraženec 0 pik, dobi zmagovalec 3 točke,
  - če ima poraženec manj kot 33 pik, dobi zmagovalec 2 točki,
  - če ime poraženec več kot 33 pik, dobi zmagovalec 1 točko.

3.4.2 Implementacija UI

Koncept uporabe globokega vzpodbujevalnega učenja za igranje iger je precej preprost. Najprej določimo vse potrebne podatke, ki jih bomo iz stanja igre prebirali in vstavljali v nevronsko mrežo. To označimo kot vhodne podatke (ang. input). Nato določimo število skritih plasti in nevronov v vsaki plasti. Na koncu določimo še podatke, za katere želimo, da jih algoritem vrne (ang. output).

Zdaj moramo algoritmu razjasniti, katere poteze so dobre in katere slabe. V primeru šnopsa je najpreprosteje, če ob koncu igre vzamemo razliko med pridobljenimi točkami našega agenta in nasprotnega igralca. Tako če agent igro zmaga, kar pomeni, da pridobi 66 ali več točk, nasprotnik pa ostane pri nekem nižjem številu, bo preračunana nagrada imela neko pozitivno vrednost. V nasprotnem primeru, če naš agent igro izgubi, bo imela preračunana nagrada negativno vrednost.

Zdaj potrebujemo še samo podatke o igrah. Lahko bi sami odigrali veliko število iger in sproti shranjevali vse poteze in izide, a to bi nam vzelo preveč časa. V takem primeru uporabimo metodo samo-igranja.

Pri samo-igranju imamo dve možni variaciji:

V prvi variaciji naš agent igra proti nekem drugem igralcu. To je lahko tradicionalen deterministične program, ki igra po točnih navodilih. V tem primeru trenutno različico agenta ocenimo na podlagi razmerja med zmaganimi in izgubljenimi igrami proti nasprotnemu igralcu.

V drugi variaciji naš agent igra proti prejšnji različici samega sebe. Tako lahko hitro in preprosto ocenimo, katera različica je boljša, vendar je za natančen rezultat še vedno priporočeno vzeti povprečje rezultatov nekaj iger.

Ker je TensorFlow v prvi vrsti napisan za programski jezik Python, je morala biti v njem napisana tudi igra. Ker smo se v šoli vsa štiri leta učili primarno programski jezik C#, je bila to velika sprememba.

Python je namesto zavitih oklepajev, ki označujejo, kje se en odsek programske kode začne in konča, zasnovan okoli zamika vrstic. Vrstice na zunanjem robu so glavne, vse bolj zamaknjene pa spadajo k predhodni, manj zamaknjeni vrstici.

Nevronska mreža lahko kot vhodne podatke sprejema le vektorje raznih velikost in dimenzij, ki pa naj bi sami po sebi že nakazovali vrednosti. V primeru šnopsa je bil ta vektor velikosti

4 x 5, saj pri igri uporabljamo štiri barve kart, vsaka barva pa ima pet možnih vrednosti. Vsaka vrstica je prikazovala eno »barvo« kart, vsak stolpec pa eno vrednost. Tako so bile na primer vse karte v prvi vrstici srca, vse karte v dvanajstem stolpcu pa kraljice. Štetje se v programskih jezikih začne s številom nič, zato je indeks dvanajstega stolpca število enajst in ne dvanajst.

Spodaj je prikazan komentar iz kode, v katerem je označeno, kaj posamezni stolpec predstavlja.

```
#      10 J  Q  K  A
# gameState = [
#     [0, 0, 0, 0, 0],  # Srce
#     [0, 0, 0, 0, 0],  # Kara
#     [0, 0, 0, 0, 0],  # Pik
#     [0, 0, 0, 0, 0]]  # Križ
```

Vsaka vrednost v tem polju predstavlja trenutno lokacijo posamezne karte. Število nič pomeni, da je karta še v kupčku, medtem ko število štiri pomeni, da je karta med tistimi, ki jih je drugi igralec dobil.

Spodaj je prikazan komentar, v katerem so napisani pomeni posameznih števil.

```
# Value definitions:
# 1 = card in Player 1 hand
# 2 = card in Player 2 hand
# 3 = card won by Player 1
# 4 = card won by Player 2
# 5 = currently played by Player 1
# 6 = currently played by Player 2
# 7 = talon
```

Ko to polje vstavimo v nevronsko mrežo, nikjer ne označimo, kaj pomeni posamezni stolpec ali vrstica, niti število na posameznem mestu. Ti konkretni podatki za mrežo niso pomembni. Pomembno je le, da se mreža skozi igranje nauči, da so vrednosti v enem stolpcu ali eni vrstici povezane ter da čez čas poveže posamezne vrednosti z dogodki, ki so se zgodili med igro.

V kodi stvaritev nevronske mreže izgleda takole:

```
def createNewModel():
    input = keras.layers.Input(shape=(rowCount, colCount, 2),
dtype=np.float32)

    l2const = 1e-4
    layer = input
    layer = keras.layers.Flatten()(layer)
    layer = keras.layers.Dense(128,
kernel_regularizer=keras.regularizers.l2(l2const))(layer)
    layer = keras.layers.BatchNormalization()(layer)
    layer = keras.layers.Activation("relu")(layer)
    for _ in range(8 if not mtcs.isDebug else 4):
        res = layer
        layer = keras.layers.Dense(128,
kernel_regularizer=keras.regularizers.l2(l2const))(layer)
        layer = keras.layers.Activation("relu")(layer)
        layer = keras.layers.Dense(128,
kernel_regularizer=keras.regularizers.l2(l2const))(layer)
        layer = keras.layers.BatchNormalization()(layer)
        layer = keras.layers.Add()([layer, res])
        layer = keras.layers.Activation("relu")(layer)

    vhead = layer
    vhead = keras.layers.Dense(36,
kernel_regularizer=keras.regularizers.l2(l2const))(vhead)
    vhead = keras.layers.Activation("relu")(vhead)
    vhead = keras.layers.Dense(1)(vhead)
    vhead = keras.layers.Activation("tanh", name="vh")(vhead)

    phead = layer
    phead = keras.layers.Dense(64,
kernel_regularizer=keras.regularizers.l2(l2const))(phead)
    phead = keras.layers.BatchNormalization()(phead)
    phead = keras.layers.Activation("relu")(phead)
    phead = keras.layers.Dense(20)(phead)
    phead = keras.layers.Activation("softmax", name="ph")(phead)

    model = keras.models.Model(inputs=[input], outputs=[phead, vhead])
    model.compile(
        optimizer=keras.optimizers.Adadelta(),
        loss=[keras.losses.categorical_crossentropy,
keras.losses.mean_squared_error],
        loss_weights=[0.5, 0.5],
        metrics=["accuracy"])
    return model
```

Skupaj imamo v modelu štirinajst plasti nevronov, od katerih ima vsaka med 9 in 128 posameznih nevronov. Z raziskavo drugih primerov izvedb nevronskih mrež je bilo ugotovljeno, da je takšna struktura najbolj primerna. To je eden od problemov pri strojnem učenju. Prave strukture ni, obstajajo le bolj in manj primerne za našo uporabo.

3.4.3 Težave pri implementaciji šnopsa

Pri računanju izgube in optimiziranju uteži nevralne mreže hitro naletimo na pomanjkanje matematične izobrazbe.

Računanje izgube oziroma razlike od pravega rezultata opravljamo s funkcijo tako imenovane kategorične navzkrižne entropije. Ta je preveč kompleksna, da bi jo popolnoma razumeli, zato nimamo druge izbire, kot da zaupamo drugim, ki jo razumejo. Ta funkcija sprejema tudi uteži, vendar smo jih zaradi nerazumevanja primorani pustiti na privzetih vrednostih.

Optimiziranje uteži poteka z optimizatorjem. Obstaja več vrst optimizatorjev, vendar ponovno naletimo na težavo, saj njihovega točnega delovanja ne poznamo. Pri raziskavi virov je razvidno, da je v večini primerov uporabljen »Adam«, vendar je pogosto uporabljen tudi »Adadelta«. Začeli bomo z uporabo slednjega, po potrebi pa preizkusili tudi prvega.

Ob zagonu programa nam prevajalnik vrne več napak. Nekaj osnovnih, povezanih s sintakso programskega jezika, je hitro odpravljivih. Pri ostalih pa se zatakne. Ko poskušamo model nevronske mreže shraniti v datoteko, preko katere bi lahko potem program igral sam proti sebi, gre nekaj narobe. Prevajalnik nam javi napako, ki pravi, da je polje elementov datoteke prazno, čeprav so v njej podatki.

Reševanje takšne napake je v večini primerov trivialno. Tukaj imamo veliko težavo, in sicer to, da ker do potankosti ne razumemo delovanja mreže in raznih pomožnih funkcij, ne vemo, kakšni podatki naj bi bili v tej datoteki. Lahko bi preverili datoteko iz drugega primera, vendar hitro ugotovimo, da so datoteke formata .npz, ki ne vsebuje tekstovnega kodiranja. To pomeni, da datoteke ljudem niso berljive, saj v njih niso zapisane črke, tako da so nam njihove točne vsebine neznane.

3.5 Uporabniški vmesnik

Za grafični prikaz UI je bilo potrebno narediti uporabniški vmesnik. Potrebno je bilo poiskati program, s katerim bi bilo mogoče ustvariti to igro. Na internetu je veliko takšnih programov, s katerimi lahko ustvariš preprosto igro. Izbrala sva Unity, ker ima veliko dokumentacije in ga je enostavno uporabljati.

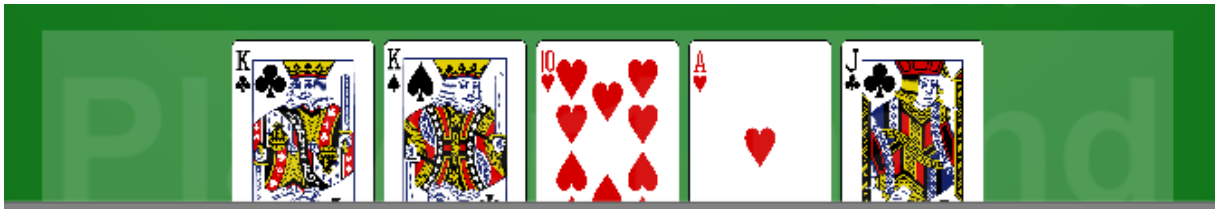
3.5.1 Unity

Program sva prenesla z njihove spletne strani, kjer je bila izbrana možnost osebne uporabe (brezplačna). Ko se je program namestil, sva ustvarila nov projekt v 2D okolju.

Znanje o uporabi programa sva pridobila z gledanjem številnih video posnetkov na YouTube. Sledenje navodilom in občasno eksperimentiranje je vodilo k cilju. Najin cilj je bil ustvariti igro šnopsa.

3.5.1.1 Grafični del

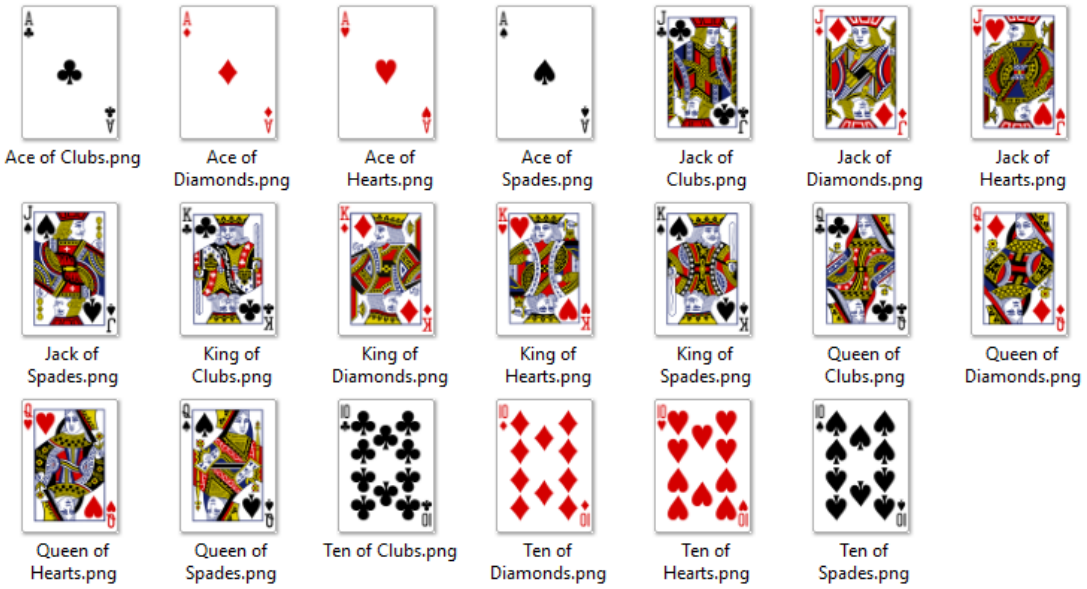
Najprej je bilo potrebno ustvariti podlago oziroma grafični del igre. Za to sva uporabila »canvas«, ki ima funkcijo platna, na katerega lahko postaviš objekte, ki se bodo videli ob zagnani igri. Kamera je nastavljena tako, da se objekti na platnu vedno vidijo. Dodala sva sliko za ozadje, ki izgleda kot igralna podlaga pri pokru. Za prikaz kart, ki so v rokah, sta bili ustvarjeni dve polji, vsako za enega igralca.



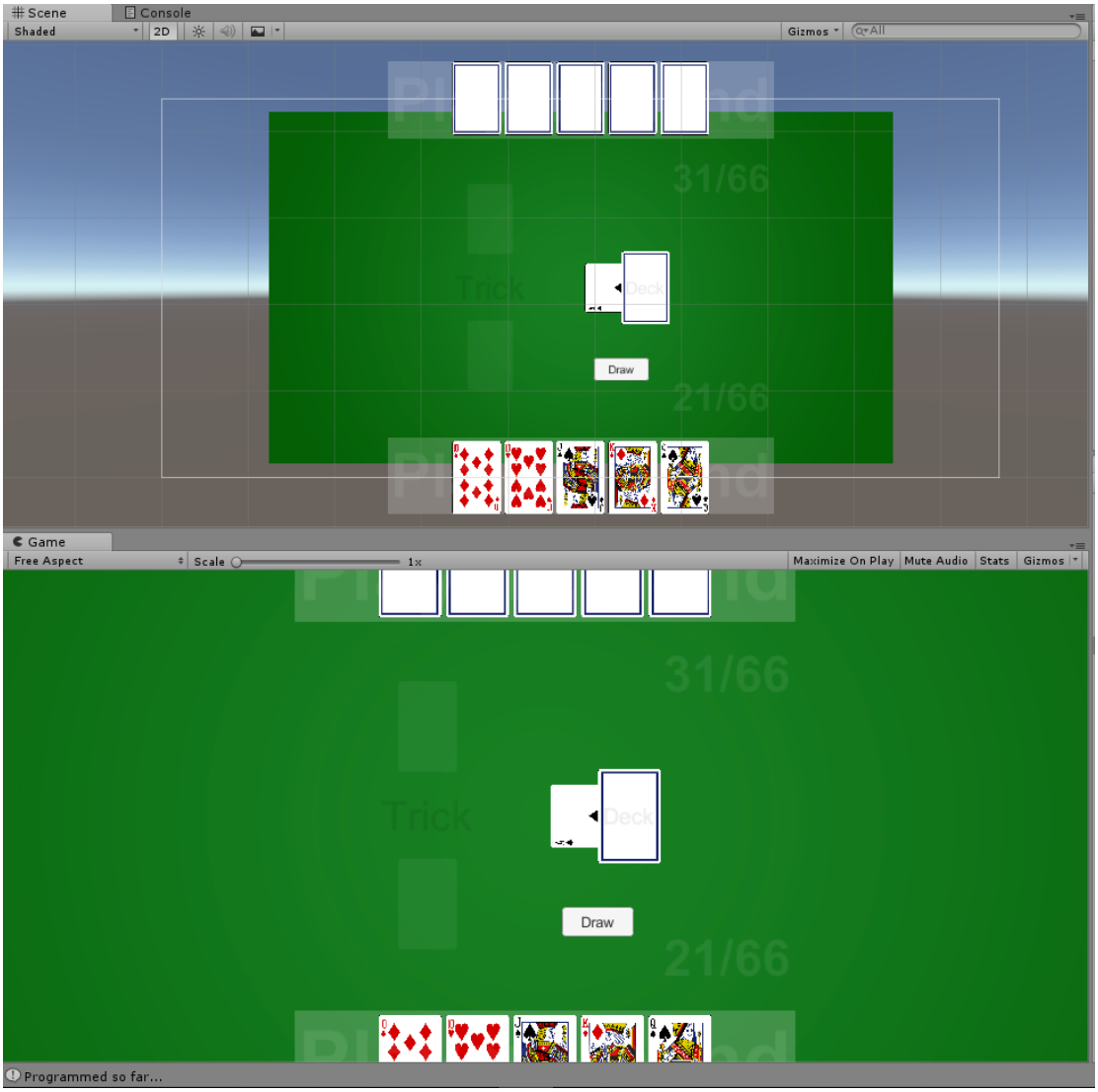
Slika 3: Karte v rokah igralca.

Da bi igralec lahko vlekel karto, je bil ustvarjen gumb »vleci« (draw). Za kupček kart in aduta je bilo potrebno narediti 2 polji. Sledili sta polji, kjer se zgodi »dvoboj kart«, ko vsak igralec odda eno karto. Igralec z močnejšo karto vzame obe karti in prišteje se mu vsota vzetih kart. Ta vsota je vidna v tekstovnem polju zraven njegovih kart.

Objektom, ki se predstavljajo kot karte, je bilo potrebno določiti sliko (ang. image). Poiskala sva slike igralnih kart in jih uvozila v projekt. S sliko se lahko vsak objekt predstavlja kot ena karta z določeno barvo in vrednostjo.



Slika 4: Slike kart, uvožene v Unity.



Slika 5: Grafični del šnopsa.

3.5.1.2 Logični del

Kodo za igro sva napisala v Visual Studiu, in sicer v programskem jeziku C#.

Za igro šnopsa sva najprej ustvarila kupček kart, roke za igralčeve karte in odlagalni kupček. To sva naredila s tehniko objektnega programiranja. Uporabila sva razrede (ang. class), konstruktorje (ang. constructor), lastnosti (ang. properties), objektne metode (ang. object methods) in še druge elemente, ki sva jih potrebovala.

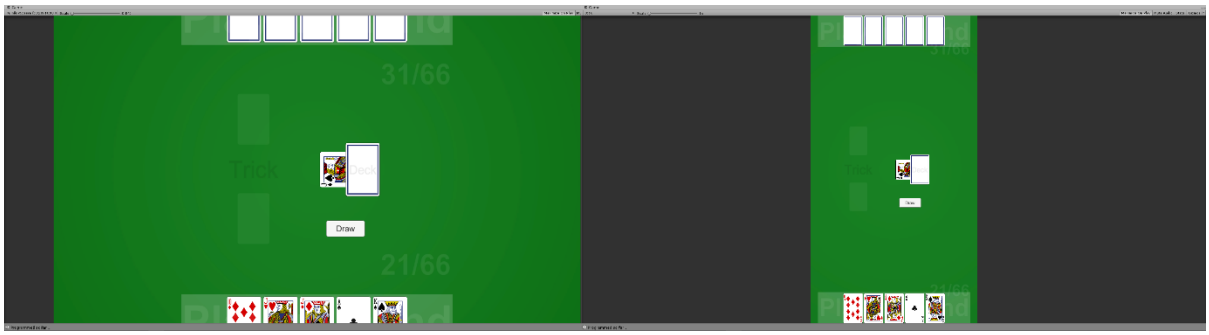
Spodaj je delček kode, uporabljen za ustvarjanje kart in njihovega kupčka:

```
public class Card   {
    private SuitEnum suit;
    private RankEnum rank;

    public SuitEnum Suit { get { return suit; } }
    public RankEnum Rank { get { return rank; } }

    public Card(SuitEnum suit, RankEnum rank)
    {
        this.suit = suit;
        this.rank = rank;
    }
}

public class Deck   {
    private List<Card> deckOfCards = new List<Card>();

    public List<Card> CardsInDeck
    {
         get { return deckOfCards; }
    }

    public Deck()
    {
         Debug.Log("Creating virtual deck...");
         for (SuitEnum suit = SuitEnum.Hearts; suit <= SuitEnum.Spades; suit++)
         {
             for (RankEnum rank = RankEnum.Ten; rank <= RankEnum.Ace; rank++)
             {
                 Card newCard = new Card(suit, rank);
                 deckOfCards.Add(newCard);
             }
         }
         Debug.Log("Virtual deck completed.");
    }
}
```

Vsaka karta potrebuje barvo in vrednost. Ko se ustvari kupček kart, dobi vsaka karta te podatke. Objekt, ki se predstavlja kot karta, mora dobiti ustrezno sliko karte iz knjižnice uvoženih slik. Za to sva ustvarila skript, ki poišče sliko karte glede na ime objekta in imena slike. Ko najde sliko, kjer se imeni ujemata, jo uporabi za objekt.

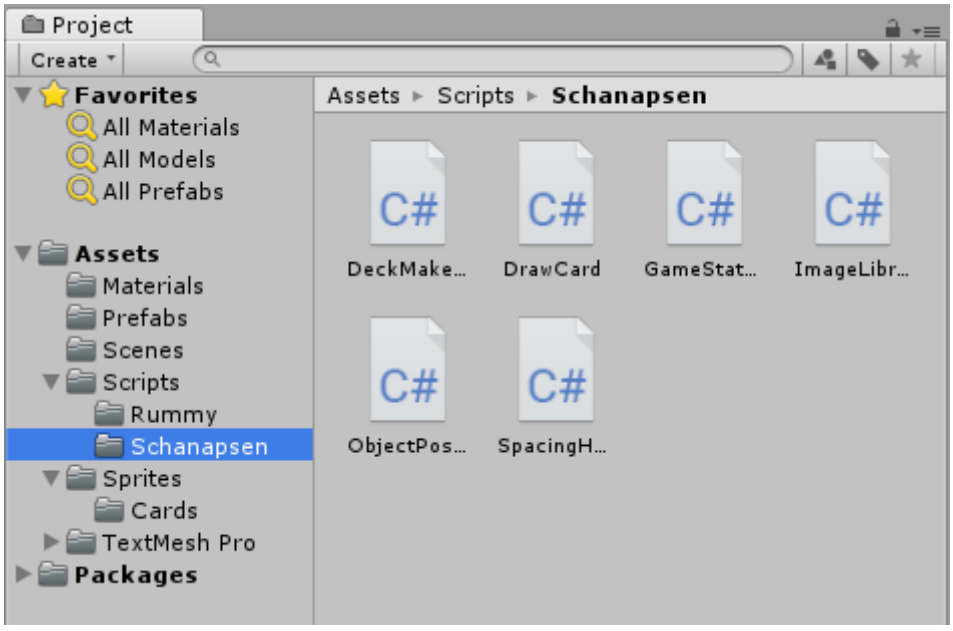
Naslednji skript je bil za funkcijo kart, da se lahko premikajo po igralni površini. Prav tako je bilo potrebno ustvariti kodo, da karta ostane tam, kjer je bila odložena, ali pa se premakne na pravilno mesto.

Da bi igra funkcionirala na več napravah, sva naredila skript, ki glede na razmerje stranic zaslona oziroma resolucijo smiselno razporedi objekte in polja. Računalniki imajo velike zaslone, zato ni potrebno, da vidimo celotno polje igralčevih kart. Na telefonu pa bi uporabnik lahko imel težave s premikanjem in izbiranjem kart, zato se vidi celotno polje.



Slika 6: Primerjava postavitve elementov na računalniku in telefonu.

Ustvarila sva še skript za potek igre. Ta določa, kateri igralec je na vrsti, kaj mora storiti in kakšne so trenutne pozicije vseh kart v igri.



Slika 7: Skripti za logični del šnopsa.

4 Razprava

Umetna inteligenca se bo še naprej razvijala, s tem pa tudi nevronske mreže, ki bodo postajale čedalje bolj kompleksne in bliže človeškim možganom.

V raziskovalni nalogi sva raziskala znanje o UI in njenem vplivu na človeštvo skozi oči anketirancev. Oni so izbrali tudi, katere igre z igralnimi kartami so najpogostejše, in sva jih potem uporabila kot ideje za izdelavo igre. Po neuspehu pri ustvarjanju remija se je najino raziskovanje tudi pri šnopsu ustavilo pri vzpostavitvi mreže in posredovanju podatkov do agenta.

Problem bova še naprej poskušala rešiti, čeprav sva ugotovila, da s trenutnim znanjem težav ni mogoče odpraviti.

4.1 Hipoteze

Hipoteze, ki sva si jih zastavila na začetku naloge, sva temeljito raziskala.

Prvo hipotezo o potrebnem izvenšolskem znanju sva potrdila. Pri tej raziskovalni nalogi je bil uporabljen programski jezik Python in knjižnica TensorFlow, ki nista v učnem načrtu srednjih šol.

Drugo hipotezo sva ovrgla. Programerski koncepti in matematika pri izdelavi kompleksnih nevronskih mrež so preveč zapleteni, da bi jih lahko obravnavali s srednješolsko matematiko. To znanje bi bilo potrebno pridobiti izven rednega šolskega programa.

Tudi tretjo hipotezo o neznanju anketirancev sva ovrgla. Z analizo ankete je bilo ugotovljeno, da večina anketirancev pozna osnove umetne inteligence in njene uporabe. Tu je potrebno poudariti, da je velik del najinih znancev tako ali drugače povezanih z računalniško stroko.

Zadnje hipoteze ne moreva ovrednotiti, saj sva ovrgla drugo hipotezo in ugotovila, da stvaritev takšne nevronske mreže s srednješolskim znanjem ni uspela.

5 Zaključek in smernice za nadaljnje delo

Na začetku se je dozdevalo, da bo izdelava UI, ki bo znala igrati remi, dokaj preprosta. Ko sva ugotovila, da je implementacija zelo zahtevna, sva spremenila igro. S šnopsom sva v primerjavi z remijem prišla nekaj korakov dlje, ampak se nama je vseeno ustavilo pri implementaciji.

Prišla sva do zaključka, da s srednješolskim znanjem matematike ni možno narediti tako kompleksne nevronske mreže, da bi delovala. Rešitev je morda v tem, da bi v učni načrt dodali več poglabljanj v umetno inteligenco.

Obstaja možnost, da se bo sčasoma našla rešitev problema in bo nevronska mreža delovala. Če pride do tega, se bo agent lahko učil igrati, tako da bo igral sam s sabo. Na koncu bi imeli izučenega agenta, ki bi proti človeku igral enakovredno, če ne bolje.

6 Bibliografija

Kaplan, A. & Haenlein, M., 2018. *Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence.* [Elektronski]
Dostopno na: https://www.sciencedirect.com/science/article/pii/S0007681318301393
[Poskus dostopa 22. februar 2019].

Nair, S., 2017. *A Simple Alpha(Go) Zero Tutorial.* [Elektronski]
Dostopno na: http://web.stanford.edu/~surag/posts/alphazero.html
[Poskus dostopa 22. februar 2019].

Unity Technologies, 2019. *Unity.* [Elektronski]
Dostopno na: https://unity3d.com/
[Poskus dostopa 22. februar 2019].

Wikipedija, 2019. *AlphaZero.* [Elektronski]
Dostopno na: https://en.m.wikipedia.org/wiki/AlphaZero
[Poskus dostopa 22. februar 2019].

Wikipedija, 2019. *Machine learning.* [Elektronski]
Dostopno na: https://en.m.wikipedia.org/wiki/Machine_learning
[Poskus dostopa 22. februar 2019].

Wikipedija, 2019. *Nevronska mreža.* [Elektronski]
Dostopno na: https://sl.m.wikipedia.org/wiki/Nevronska_mre%C5%BEa
[Poskus dostopa 22. februar 2019].

Wikipedija, 2019. *TensorFlow.* [Elektronski]
Dostopno na: https://en.m.wikipedia.org/wiki/TensorFlow
[Poskus dostopa 22. februar 2019].

Wikipedija, 2019. *Umetna inteligenca.* [Elektronski]
Dostopno na: https://sl.m.wikipedia.org/wiki/Umetna_inteligenca
[Poskus dostopa 22. februar 2019].