Umetna inteligenca 2017-2018

Seminarska naloga 2

Naloga se izvaja **v parih**. Zagovori bodo potekali v terminu vaj v tednu **15. 1. – 19. 1. 2018.**

Tema druge seminarske naloge je spodbujevano učenje. Želimo naučiti agenta (plen) preživeti čim dlje v simuliranem okolju. Plen mora piti vodo, jesti travo in bežati pred plenici. Agent ima na voljo 5 akcij:

Akcija	**Opis**
1	premik proti severu
2	premik proti jugu
3	premik proti vzhodu
4	premik proti zahodu
5	pij/jej (če je v vodi oziroma na travi)

Agentov cilj je preživeti čim daljši čas. Simulacija se zaključi, ko se agenta dotakne plenilec oziroma umre zaradi žeje ali lakote.

Na učilnici so objavljene datoteke "**simulation.R**", "**RL.R**", "**utils.R**" in "**main.R**". Prva datoteka vsebuje funkcije za učenje agenta ter premikanje in izris simuliranega okolja. Te funkcije skrbijo za delovanje celotnega sistema in jih ne spreminjajte (lahko jih popolnoma ignorirate).

V drugi datoteki sta funkciji **getStateDesc** in **getReward**, ki predstavljata vsebinski del seminarske naloge. Vaša naloga je dopolniti implementacijo teh funkcij tako, da bo učenje agenta čim bolj učinkovito.

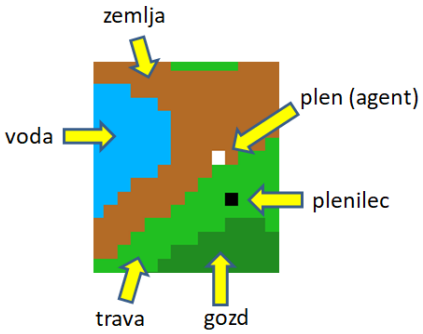
Tretja datoteka vsebuje nekaj primerov pomožnih funkcij, ki jih lahko uporabite pri reševanju naloge. Po potrebi lahko dodate svoje oziroma predelate obstoječe funkcije.

Četrta datoteka predstavlja glavni program, ki izvede učenje ter sproži simulacijo z naučeno strategijo.

Simulirano okolje

Simulirano okolje predstavlja teren v obliki dvodimenzionalne tabele. Vsak element tabele označuje tip terena, ki je lahko voda, zemlja, trava ali gozd. Teren je poseljen s poljubnim številom plenilcev (privzeta nastavitev je 1) in agentov, ki predstavljajo plen (privzeta nastavitev je 1). Na sliki 1 je prikazan primer simuliranega okolja.

Obe entiteti (plenilec in plen) se najhitreje premikata po zemlji, nekoliko počasneje po travi, še počasneje po gozdu in najpočasneje skozi vodo. Plen je na vseh terenih nekoliko hitrejši od plenilca. Če se neka entiteta nahaja v gozdu, je nevidna za entitete zunaj gozda in obratno. Gozd predstavlja tudi neprosojno oviro v vidnem polju vseh entitet (glej sliko 2). Plenilec lovi plen, ki mu je najbližje, ampak samo, če ga vidi. Plenilec lahko tudi zavoha plen v svoji neposredni bližini, tudi če je le-ta skrit. Plenilec in plen morata občasno piti vodo in jesti - plen se prehranjuje s travo, plenilec pa z ulovljenim plenom. Ko plen poje travo, se teren na tistem mestu spremeni v zemljo.



Slika 1: Primer simuliranega okolja

Slika 2: Plen in plenilec se zaradi gozda ne vidita

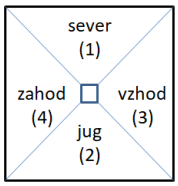
Funkcija getStateDesc

Funkcija **getStateDesc** se avtomatično kliče ob vsaki spremembi simuliranega okolja. Funkcija prejme popoln opis simuliranega okolja (argument ***simData***) in oznako agenta (argument preyId), ki ga trenutno obravnavamo. Rezultat funkcije je **opis diskretnega stanja**, ki predstavlja strnjen opis simuliranega okolja.

```
getStateDesc <- function(simData, preyId) {
    …

    state
}
```

Namen funkcije ***getStateDesc*** je povzeti vso relevantno informacijo o trenutni situaciji na terenu, ki bo pomagala agentu (plenu) daljše preživetje v simuliranem okolju. Preprosta začetna ideja je, da stanje predstavimo z vektorjem elementov, ki predstavljajo razdaljo in smer do najbližjega plenilca, vode, trave in gozda. Če prostor okoli agenta razdelimo na štiri cone (glej sliko 3), lahko stanje na sliki 1 predstavimo z vektorjem c(4, 2, 4, 4, 1, 2, 6, 2). Plenilec je na razdalji 4 (Manhattanska razdalja) in se nahaja južno od agenta. Najbližja voda je na razdalji 4 in se nahaja zahodno od agenta. Najbližja trava je na razdalji 1 in se nahaja južno od agenta. Najbližji gozd je oddaljen 6 korakov in je prav tako južno od agenta. Ker želimo imeti čim manj različnih stanj se lahko odločimo, da vse razdalje navzgor omejimo z neko vrednostjo, na primer 30.



Slika 3: Okolica agenta razdeljena na štiri cone

Vaša naloga je določiti način kodiranja stanj in ga implementirati v funkciji getStateDesc. Množica stanj mora biti končna in diskretna, zato morate upoštevati naslednje omejitve:

- **vsa stanja morajo biti opisana z vektorjem enake dolžine**
- **vsak element vektorja opisa mora biti pozitivno celo število**

Zaradi hitrosti in zanesljivosti učenja je zaželeno, da je različnih stanj čim manj!

Funkcija getReward

Funkcija ***getReward*** se avtomatično kliče med učenjem agenta. Funkcija prejme opis prejšnjega (argument ***oldstate***) in naslednjega stanja (argument ***newstate***), ter nazadnje izvedeno akcijo (argument ***action***). Rezultat funkcije je nagrada (ali kazen), ki jo agent sprejme v opisani situaciji.

```
getReward <- function(oldstate, action, newstate) {
    …

    reward
}
```

Vaša naloga je implementirati funkcijo getReward za smiselno nagrajevanje agenta v fazi učenja. Nagrada mora spodbujati agenta, da izvaja koristne akcije (na primer pitje in hranjenje) oziroma ga odvračati od negativnih manevrov (na primer približevanje plenilcem).

Q-učenje

Ko pripravite implementacije funkcij ***getStateDesc*** in ***getReward***, lahko začnete z učenjem agenta s pomočjo priložene funkcije ***qlearning***, ki se nahaja v datoteki "**simulation.R**". Funkcijo kličemo z naslednjimi argumenti:

```
qlearning(dimStateSpace, gamma = 0.99, maxtrials = 1000, maxsteps = 1000)
```

Argument ***dimStateSpace*** je vektor iste dolžine kot opisi stanj, ki jih vrača funkcija ***getStateDesc***. Vsak element vektorja ***dimStateSpace*** določa največjo vrednost, ki jo lahko zavzame istoležni element v opisu stanja.

Ostali argumenti so opcijski in določajo faktor zmanjševanja (argument ***gamma***), maksimalno število poskusov učenja (argument ***maxtrials***) ter najdaljšo sekvenco premikov v enem poskusu (argument ***maxsteps***).

Rezultat klica funkcije ***qlearning*** je matrika, ki jo potrebujemo za izvajanje simulacij.

Glavni program

Glavni program se nahaja v datoteki "**main.R**". Vsebuje klic funkcije za učenje agenta, njen rezultat pa se potem uporabi kot vhodni argument funkcije za simulacijo okolja.

Na primer, za zgoraj opisani način kodiranja stanj:

```
qmat <- qlearning(dimStateSpace = c(30, 4, 30, 4, 30, 4, 30, 4))
simulation(qmat)
```

Cilj naloge

V okvirju seminarske naloge je potrebno:

- implementirati funkciji ***getStateDesc*** in ***getReward***,
- naučiti agenta čim dlje preživeti v simuliranjem okolju,
- analizirati dosežen uspeh (povprečno število premikov na epizodo) glede na izbrane parametre (način kodiranja stanja, način podajanja nagrade, število plenilcev, število poskusov učenja, faktor zmanjševanja…),
- predstaviti ugotovitve v poročilu (dokument v pdf ali doc formatu),
- uspešno zagovoriti nalogo na laboratorijskih vajah.

Na končno oceno seminarske naloge vplivajo inovativnost in elegantnost rešitve, ambicioznost pri raziskovanju problema, argumentacija izbranih postopkov, analiza in predstavitev dobljenih rezultatov.