

# Python simuliacija

## Gynimo ataskaita

Milita Songailaite

2020 m. gegužės 28 d.

## 1 Ką pavyko įgyvendinti

Šio projekto tikslas buvo sukurti simuliaciją, kurioje agentai, mokinami neuroninio tinklo, išmokyti susirasti maisto ir išgyventi tam tikrą laikotarpį. Projektą sudaro šios dalys:

- Sukurta grafinė sąsaja su simuliacijos aplinka ir agentais;
- Simuliacijoje agentai gali vaikščioti po visą žemėlapią bei susidūrę su maistu, jį suvalgyti;
- Agentų smegenys, t.y. mąstymo mechanizmas yra neuroninis tinklas, kuris veikia pagal Advantage Actor Critic (A2C) algoritmą.

### Genetiniai algoritmai

Pradžioje semestro buvo planuojama programoje panaudoti ir genetinius algoritmus. Jie turėjo būtų naudojami agentų organizmams besidauginant - genetiniais algoritmais būtų nusprendžiami palikuonių turimi požymiai bei turimos žinios aplinkoje. Ši dalis dar nebuvo pridėta, nes tam reikėjo skirtingų smegenų kiekvienam agentui.

Taip pat, buvo mąstyta genetinius algoritmus naudoti greitesniam tinkamų tinklo svorių suradimui, bet ši dalis taip pat dėl laiko stokos buvo palikta ateičiai.

## 2 Simuliacijos aplinka

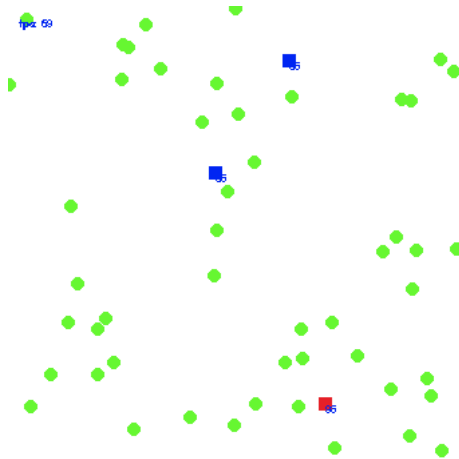
Simuliacijos aplinką sudaro tam tikras kiekis mėlyna spalva pažymėtų agentų (po pirmo susidūrimo su maistu, agentas pakeičia savo spalvą į raudoną). Kiekvieną kartą pradedant naują epochą, agentas atsiranda atsitiktinėje žemėlapių vietoje. Agentai gali judėti visame žemėlapyje, susidūrę su maistu (žemėlapyje pažymėtas žalia spalva), jie jį suvalgo. Po susidūrimo su agentu, maistas iš savo vietos yra pašalinamas ir iškart atsitiktinėje žemėlapių vietoje pridedamas naujas maistas.

Kol programoje dar nebuvo neuroninio tinklo, nusakančio į kurią pusę eiti, agentas paprasčiausiai eidavo iki artimiausio jam esančio maisto. Atstumams paskaičiuoti buvo naudojamas Euklido atstumas:

$$d(x, y) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

### Simuliacijos stadijos aprašymas

Vėliau, pridėjus neuroninį tinklą, visas aplinkos žemėlapis buvo paverstas į 10 kartų mažesnę tinklę, t.y. jei žemėlapių išmatavimai yra  $980 \times 740$ , tai jis bus verčiamas į  $98 \times 74$  langelių tinklę. Taip buvo padaryta dėl to, kad būtent tas tinklas bus perduodamas neuroniniam tinklui, kaip esama stadija. Jei tinklui duotume kiekvieną žemėlapių pikselį, jo skaičiavimai užtruktų per ilgai ir simuliacija negalėtų reikiamai veikti.



1 pav.: Simuliacijos aplinka

### Agentų duomenys

Simuliacijoje visi agentai juda vienodu greičiu, bei turi vienodą alkio rodiklį - 100. Kuo agento alkio rodiklis didesnis, tuo jis yra geresnėje situacijoje. Agentų alkio rodiklis kiekvieną milisekundę krenta po 0.1, o suvalgius maistą, jis padidėja 10. Kai agento alkio rodiklis pasiekia 0, agentas miršta. Bent vienam agentui mirus, simuliacijos epocha yra nutraukiama ir iškart pradedama nauja epocha. Taip pat, jei visi agentai sugeba išgyventi, epocha yra baigiama praėjus tam tikram nustatytam laikui.

## 3 Algoritmai

Kuriant simuliaciją, buvo išbandyti keli algoritmai, tačiau finale nuspręsta pasilikti prie A2C algoritmo. Žemiau nurodytos priežastys, kodėl išbandyti algoritmai simuliacijai tiko, ir kodėl netiko.

### 3.1 Q-learning

Vienas pirmųjų pasirinkimų buvo Q-learning algoritmai. Jų veikimas yra paremtas funkcija, kuri sako agentui, kaip reikėtų elgtis tam tikrose situacijose, taigi, tai buvo paprasčiausias pasirinkimas, nuo ko pradėti. Pradžioje algoritmas buvo išbandytas ant nedidelio žemėlapio (1x5), kuriame pirmame ir paskutiniame langelyje buvo maistas - pirmame langelyje vertas 2 tašku, paskutiniame - 10 taškų.

$$| 2 | \_ | * | \_ | 10 |$$

2 pav.: Q-learning algoritmo aplinka, kur agentas pažymėtas “\*”

Kadangi pradinis projekto tikslas buvo naudojantis neuroniniu tinklu išmokinti agentus surasti maistą, toliau buvo nuspręsta pereiti prie deep Q-learning algoritmo. Tačiau šis algoritmas nepilnai tiko pasirinktam uždaviniui, nes jis apmokina agentą atlikti diskrečius veiksmus. Kadangi simuliacijoje veiksmas turėtų būti tolydūs (t.y. vektoriai vedantys agentą link artimiausio maisto), šis algoritmas nebuvo pasirinktas.

### 3.2 Neuroninis tinklas

Kitas bandymų etapas buvo pasirašyti neuroninį tinklą, kuris išmokytų rasti geriausias koordinatas (x ir y reikšmes), į kurias agentui reikėtų eiti. Tam, kad būtų galima paprasčiau keisti visus parametrus, tinklas

buvo rašytas nenaudojant papildomų neuroninių tinklų bibliotekų. Naudojant paprastą neuroninį tinklą buvo gana sunku nustatyti, ar jo parinktos koordinatės yra geresnės, ar blogesnės, todėl buvo pereita prie paskutinio algoritmo - A2C.

### 3.3 A2C algoritmas

Šis algoritmas veikia panašiai, kaip ir deep Q-learning algoritmai. Kiekvieną iteraciją, jis vertina dabartinę situaciją ir būsimą situaciją. Palygindamas abejas situacijas, jis gali įvertinti dabartinio veiksmo gerumą. Dar vienas pranašumas yra tas, kad algoritmas gali dirbti su tolydžiais inputais ir outputais - tai buvo vienas svarbiausių veiksnių, dėl ko simuliacijai buvo išrinktas šis algoritmas.

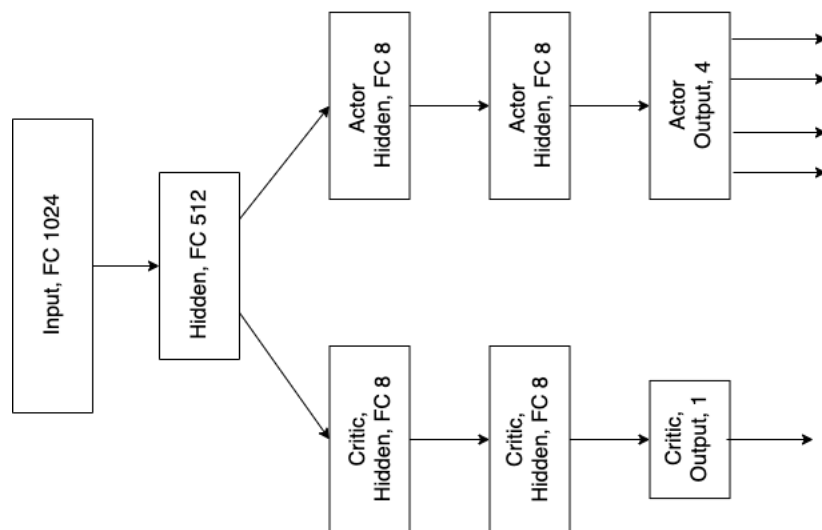
## 4 Pasirinkto algoritmo ypatumai

### 4.1 Tinklo inputai/outputai

A2C algoritmas kaip tinklo inputus pasiima esamas agentų būsenas žemėlapyje. Kadangi perduodant kiekvieną langelį net ir iš sumažinto žemėlapių, tinklas skaičiuotų svorius pakankamai ilgai, tai buvo nuspręsta sumažinti agentų matomumą. Tai yra, dabar kiekvienas agentas mato ne visą žemėlapi, o tik jo dalį - 32x32 langelių. Taigi, vieno agento inputai į tinklą bus 32x32 dydžio matrica, kurioje 1 yra pažymetas maistas, o 0 - tušti langeliai.

Apsimokęs tinklas grąžina 4 reikšmes - jos parodo keturias kryptis (Š, P, V, R), į kurias agentas turėtų judėti, norint pasiekti artimiausią maistą. Pagal šias kryptis yra sudaromi judėjimo vektoriai ir perduodami agentui, kad jis galėtų judėti reikiama kryptimi.

### 4.2 Tinklo struktūra



3 pav.: Neuroninio tinklo struktūra