Playing Flappy Bird with AI

Domen Grzin
dg6781@student.uni-lj.si
Faculty of Computer and
Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Urban Cör
uc09767@student.uni-lj.si
Faculty of Computer and
Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

, ,

POVZETEK

V tem poročilu predstaviva implementacijo in proces učenja agenta za igranje igre *Flappy Bird*. Z uporabo *Q-learning-a* sva dosegla, da ptič "nikoli" ne umre. Pokaževa tudi kako sva določila parametre, ki so agentu omogočile učenje. Ko se je agent naučil igranja igre sva tudi spremenila okolje in opazovala, če se je hitreje naučil igranja v novem okolju.

KLJUČNE BESEDE

Flappy bird, Reinforcement learning, Q-Learning, Q-table

1 UVOD

Flappy bird je arkadna mobilna igra, ki je izšla 24. maja leta 2013 za platformi iOS in Android. Igra se je hitro razširila in pridobila na popularnosti. Cilj igre je, da ptiček z imenom Faby pride skozi čim več cevi, ki se pojavljajo na ekranu.

Čez nekaj let pa so na youtubu postali popularni videi z naslovi "AI plays flappy bird". Takrat sva ravno začenjala s študijem in se nama je to zdel tak projekt, ki bi ga nekoč sama želela realizirati. Takrat sva imela še premalo znanja za kaj takega, pri tem predmetu pa se je zdela smiselna izbira.

2 RAZVOJ IGRE IN AGENTA

2.1 Razvoj igre

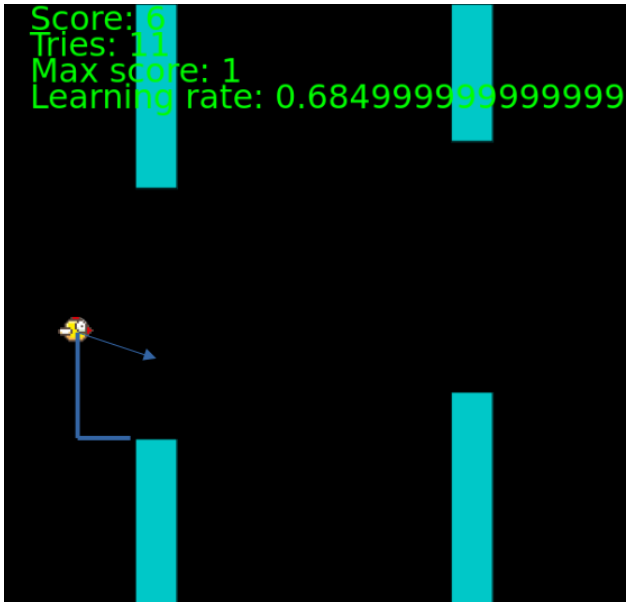
Igro sva razvila s pomočjo *javascript* ogrodja **P5.js**. Tu bi omenila, da ker sva najprej razvila igro direktno v grafičnem vmesniku je to precej podaljšalo, čas testiranja, kot če bi razvila samo "backend" igre in na to navezala še grafični vmesnik. Pri grafičnem vmesniku sva bila odvisna od "frame rate"-a monitorja na katerem sva učila najingega agenta.

Klasično igro sva malo modificirala, in sicer da bi pohitrila učenje agenta, Faby umre, če se dotakne tudi stropa (in ne samo v primeru, ko pade na tla ali se zaleti v cev). Povečala sva tudi odprtino med cevmi in sicer je bila odprtina na začetku velika 240 pikslov oz. 40% v y smeri.

2.2 Razvoj agenta

Za učenje agenta sva uporabila *Q-learning*. Za predstavitev stanja sva izbrala 3 parametre:

- razdalja v x smeri do prve cevi,
- razdalja v y smeri do prve spodnje cevi,
- hitrost Faby-a.



Slika 1: Vizualizacija stanj, ki jih vidi agent

Ker je stanj veliko, je potrebna diskretizacija prostora. Sprva sva razdelila prostor na 173 * 300 * 20 = 1038000 stanj. To se je izkazalo za veliko preveč stanj, saj se Faby ni naučil prav veliko tudi po 30000+ poskusih. Edino kar se je uspel naučiti je bilo to, da je obstal nad tlemi in se je nato zaletel v prvo cev.

Pri razdelitvi prostora na 10 košev za x smer, 20 za y smer in 5 za hitrost, se agent nauči leteti v neskončnost po, v povprečju, 16-ih poiskusih.

2.3 Nagrajevanje in kaznovanje

Med tekom učenja si za agenta zapomniva njegova stanja (x razdalja, y razdalja in hitrost), ki jih je prečkal ter akcijo, ki jo je izbral, ali naključno ali po greedy metodi. Zgodovino stanj in akcij shranjujeva v seznam *history*. Ko se agent zaleti v tla, strop ali cev ga kaznjujeva s kaznijo -1000. Če agent uspešno pride skozi odprtino v ceveh ga nagradiva z nagrado v velikosti števila prečkanih odprtin v trenutnem teku. To se pravi 1 ko prečka prvo odprtino, 2 ko prečka drugo in tako dalje.

Nagrada, oziroma kazen, se nato prenaša nazaj po stanjih iz seznama *history*, kjer v vsakem naslednjem stanju dušiva vrednost

z $\gamma = 0.9$. Najina implementacija tako sledi enačbi:

$$Q_{new}(s_t, a_t) = (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a))$$

kjer je s_t stanje v trenutnem časovnem intervalu in a_t akcija. Nagrada r_t je neničelna samo v zadnjem časovnem intervalu.

Agent mora med trenutno odprtino v cevi in naslednjo narediti 200 odločitev ali se dvigne ali ne. V primeru, ko je prostor razdeljen na 10 košev za x smer, 20 za y smer in 5 za hitrost, to pomeni, da je verjetnost, da bo agent večkrat v enakem stanju velika. Pri vzratnem prenašanju nagrade sva zato pazila, da take primere iz *history* preskočiva, da ne pride do posodabljava istega stanja večkrat.

Najmanjše število poskusov, ki jih je agent potreboval, da je prišel mimo 100 cevi je bilo 11. Zakaj meja 100 cevi? V večini primerov je potem prišel skozi več 1000 cevi oz. ni umrl dokler nisva prekinila izvajanja. V redkih primerih se je zgodilo, da je prišel do še neznanega stanja in umrl. Parameter α je bil na začetku enak 0.9 in se je vsakič, ko je dosegel nov najboljši dosežek zmanjšal za 0.025. δ je bil ves čas enak 0.9.

2.4 Sprememba okolja

Ko sva agenta uspela naučiti neskončnega letenja na osnovnem okolju, sva mu poiskusila otežiti nalogo. Za spremembo okolja so na voljo trije parametri:

- velikost odprtine, kjer lahko agent varno prečka cevi ne da bi se zaletel v njih. Manjša kot je odprtina težje jo najde, ne da bi se zaletel v cevi.
- Razmak cevi, bolj kot so cevi skupaj manj časa ima, da prilagodi višino na naslednjo odprtino.
- Velikost skoka, za koliko se dvigne Faby, ko se agent odloči za skok. Večji kot je skok več in hitreje bo skočil, se pravi bo težje obdržal natančno višino. Prav tako lahko prveč skoči in prav tako zgreši odprtino.

Agentu sva poskusila otežiti okolje glede na vse tri kriterije. Najlažje se uči, če mu zmanjšamo odprtino. Dokler ima dovolj košev v y smeri bo lahko našel odprtino in varno prečkal cevi.

Skok je malo težji, saj če mu damo prevelik skok se s podanim diskretiziranim prostorom ne more dovolj natančno naučiti kdaj skočiti in tako preskoči odprtino in se zaleti v zgornjo cev. Če mu damo več stanj, kar bi omogočilo bolj natančne akcije, pa učenje traja zelo dolgo.

Razmak cevi je tudi težaven, saj manjši kot je hitreje po prehodu odprtine mora agent začeti izbirati prave akcije in naključne mu bolj pokvarijo verjetnost uspeha. Zaradi načina posodabljanja Q-tabele se nagrade počasi vzvratno prenesejo do začetka. Tako učenje traja dolgo preden agent uspešno začne izbirati pravilne akcije takoj po prehodu odprtine.

2.5 Doučenje

Poskusila sva doučiti obstoječi delujoči model v zahtevnejšem okolju. V primeru doučevanja sva prilagodila le velikost odpritne in razmak cevi, saj velikost skoka spremeni akcijo in ne okolja. Primerjala sva, ali to, da agent začne z že obstoječim delujočim modelom pohitri učenje v primerjavi z prazno Q-tabelo.

Model z začetnim številom stanj se je pri doučenju slabo odzval in potreboval dalj kot če se je agent učil iz nič. Zdi se nama, da se model z manj stanji preveč prilagodi izvornemu problemu in zato rabi dalj, da se douči novega problema.

Iz tega razloga sva povečala število stanj. Tak agent se je hitreje doučil kot prejšnji, vendar v povprečju primerljivo z agentom, ki se je učil iz prazne Q-tabele. Verjetno bi nam večja kompleksnost modela prinesla boljše rezultate, kar se tiče doučenja, vendar postane čas učenja osnovnega modela predolg, da bi lahko to preiskusila.

3 ZAKLJUČEK

Z spodbujevalnim učenjem sva uspešno ustvarila model, ki se uspe ne zaleteti v igri Flappy Bird. Rešitev sva uspela implementirati v programskem jeziku Javascript, brez uporabe zunanjih knjižnic za spodbujevalno učenje.

Najpomembnejša stvar, ki sva jo odkrila s tem projektom je, da ni vedno več stanj bolje. Od začetka sva dolgo poiskušala z veliko stanj, saj sva tako pričakovala boljšo natančnost agenta. Izkazalo se je, da nikoli nisva prišla do delujočega modela, polnjenje tako velike tabele traja predolgo.

Preiskušala sva tudi kako malo stanj lahko uporabiva, da še vedno dobiva delujoči model. Najmanjši ni imel košev za x smer, 20 za y in 2 za hitrost.