

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6848

Podržana evolucija agenta za igru Dota 2

Jakov Ivković

Zagreb, lipanj 2020.

Umjesto ove stranice umetnite izvornik Vašeg rada.

Da bi ste uklonili ovu stranicu obrišite naredbu \izvornik.

Zahvaljujem se mentoru prof. dr. sc. Domagoju Jakoboviću na vrlo zanimljivoj temi te podršci tijekom izrade rada.

Zahvaljujem se kolegici Lauri Torić na dragocjenim savjetima te na međusobnoj razmjeni ideja.

SADRŽAJ

1. Uvod	1
2. Opis problema	2
2.1. DotA	2
2.1.1. Pravila	2
2.2. 1v1 Shadow Fiend	4
2.2.1. Shadowraze	5
2.2.2. Necromastery	6
2.3. Breezy	6
3. Programska implementacija	9
3.1. Algoritam	9
3.1.1. Evolucijska strategija	12
3.1.2. Genetski algoritam	12
3.2. Jedinka	13
3.2.1. Kartezijsko genetsko programiranje	14
3.2.2. Neuronska mreža	15
3.3. Selekcija	16
3.4. Križanje	16
3.5. Mutacija	17
3.5.1. CGP mutacije	17
3.6. Reporter	18
3.7. Evaluator	18
3.7.1. Evaluator klasifikacije iris cvijeta	18
3.7.2. Evaluator simboličke regresije	19
3.7.3. Breezy evaluator	19
4. Rezultati	22

5. Zaključak 25

Literatura 26

1. Uvod

Rad se bavi problemom s GECCO 2020¹ natjecanja naziva *Dota 2 1-on-1 Shadow Fiend Laning Competition*. Cilj natjecanja je izraditi što boljeg agenta za računalnu MOBA² igru Dota 2 u 1v1 formatu.

Problemu će se pristupiti koristeći evolucijske algoritme; evolucijska strategija i genetski algoritam te dva različita modela jedinke; neuronska mreža i CGP³. Jedinke ćemo evaluirati s obzirom na odluke koje donose tijekom igre.

U poglavlju 2 nalazi se kratki pregled pravila igre te opis problema. Nakon toga, u poglavlju 3 nalazi se opis svih komponenti programskog rješenja te osvrt na njihovu implementaciju. Na kraju, u poglavlju 4 je opis postignutih rezultata te, nakon toga, u poglavlju 5 zaključak temeljen na tim rezultatima.

¹The Genetic and Evolutionary Computation Conference

²Multiplayer Online Battle Arena

³Kartezijsko genetsko programiranje (engl. *Cartesian Genetic Programming*)

2. Opis problema

2.1. DotA

DotA¹ je MOBA igra koja je nastala 2003. godine kao mod za stratešku igru Warcraft III. 10 godina kasnije, 2013. godine izlazi Dota 2. Zapravo potpuno ista igra, samo u standalone verziji s boljom grafikom i kontrolama.

Izlaskom Dote 2 popularnost igre raste te do danas Dota ostaje jedna od najpopularnijih² igri u MOBA žanru. Također, zbog svoje kompleksnosti Dota ubrzo postaje najkompetitivnija igra u esports svijetu s dotad neviđenim nagradnim fondovima na turnirima. Samo prošlo svjetsko prvenstvo (pod imenom *The International 9*) imalo je ukupan nagradni fond od nešto više od 34 milijuna američkih dolara.[2]

Izrazita kompleksnost igre zainteresirala je i računalne znanstvenike te su, 2017. godine, znanstvenici iz neprofitne organizacije OpenAI izradili agenta za 1v1 inačicu Dote sa *Shadow Fiend* herojem³, što je upravo problem kojim se bavi i ovaj rad. Agent je vrlo brzo savladao najbolje svjetske igrače[5], međutim, 1v1 je egzibicijski mod i kompleksnošću nije ni blizu standardnom 5v5 Dota formatu. Stoga, već godinu dana kasnije, 2018. godine OpenAI je izradio *OpenAI Five* — 5 nezavisnih agenata koji zajedno igraju u standardnom⁴ 5v5 formatu. Ubrzo je OpenAI Five počeo pobjeđivati naprednije igrače i amaterske timove, a već godinu dana kasnije pobijedio je i svjetske prvake.[6]

2.1.1. Pravila

Dota se sastoji od mnogo mehanika koje ju čine kompleksnom. U ovom poglavlju ćemo se dotaknuti samo najbitnijih mehanika za ovaj rad.

Dota se igra s 10 igrača koji su suprotstavljeni u dva tima po 5 igrača. Svaki igrač

¹Defense of the Ancients

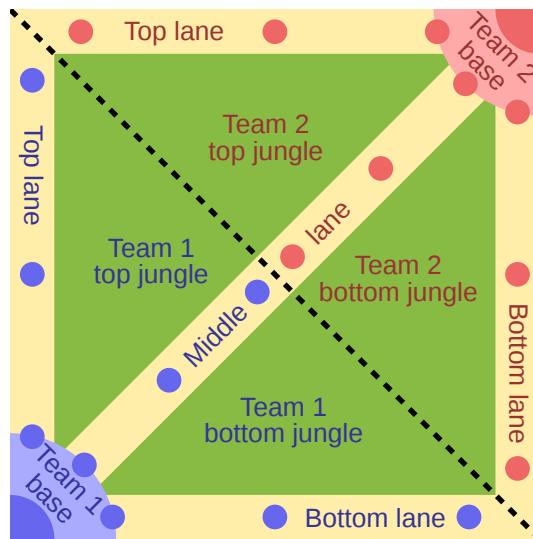
²Preko 8 milijuna MAU (engl. *Monthly Active Users*)

³Heroji — Likovi koje igrači kontroliraju (engl. *Hero*).

⁴uz određena ograničenja

kontrolira svog heroja. Ukupan broj heroja je 119 te svaki igrač, na početku meča, bira kojeg će igrati. Heroji se međusobno razlikuju po moćima koje posjeduju te zbog toga zahtijevaju različite taktike kod igranja. Svaki tim pokušava nadvladati protivnički tim kako bi mu uništio bazu čime meč završava. Trajanje jednog Dota meča je uglavnom između 20 minuta i sat vremena s prosječnim trajanjem od pola sata.

Baze dvaju timova povezane su preko 3 *lanea*⁵, kao što je to vidljivo na slici 2.1. Na svakom laneu postoje po 3 obrambena tornja (plavi i crveni kružići na slici) koje je potrebno uništiti kako bi se došlo do protivničke baze. Svakih 30 sekundi se u bazama stvaraju po 3 vala *creepova*⁶ koji putuju po laneovima prema protivničkoj bazi.



Slika 2.1: MOBA map

CC BY-SA 3.0 — Original by Raizin, SVG rework by Sameboat



Slika 2.2: Toranj i val creepova

⁵3 trake, puteljka (engl. *lane*)

⁶Slabi, nekontrolabilni likovi



Slika 2.3: Borba creepova

Dva vrlo bitna faktora na koja igrači moraju paziti, o kojima ovisi jačina njihovih heroja, su zlato (engl. *gold*) i iskustvo (engl. *experience*). Svi heroji na početku meča prilično su slabi te, pošto nisu sposobni napasti protivničku bazu, vrlo velik naglasak u meču je upravo u borbi za ta 2 resursa. U velikoj većini mečeva onaj tim koji ima značajnije veći posjed zlata i iskustva na kraju i pobjedi.

Iskustvo određuje herojev *level*. Heroji počinju s levelom 1 te su sa svakim levelom jači (maksimalni level je 30). Heroji zarađuju iskustvo kada u njihovoј blizini⁷ umre neprijateljski lik⁸.

Drugi bitan resurs je zlato. Glavni izvor zlata je *last hitanje* creepova. *Last hit*⁹ je zadavanje zadnjeg udarca neprijateljskom creepu. Ako igrač uspije zadati zadnji udarac creepu, dobit će određenu količinu zlata kao nagradu. Igrač zlato koristi kako bi kupovao *iteme*, predmete koji dodatno ojačavaju heroja.

Vrlo bitna mehanika u doti, kojom se pokušava usporiti napredovanje protivnika, je *denjanje* (engl. *deny*). Denjanje je last hitanje vlastitih creepova. Denjanjem igrač ništa ne dobiva za sebe, međutim, time onemogućuje neprijateljskom heroju da dobije last hit nad tim creepom. Također, ako je creep denjan, on neprijateljskim herojima daje upola manje iskustva.

2.2. 1v1 Shadow Fiend

1v1 format mnogo je jednostavniji od standardnog 5v5 formata. Za razliku od standardnog formata gdje se pobjeđuje uništavanjem protivničke baze, u 1v1 formatu pobjeđuje prvi igrač koji ubije protivničkog igrača dva puta ili prvi koji uništi protivnički toranj na

⁷U točno određenom radiusu prikazanom zelenom kružnicom na slici 2.4

⁸Bilo neprijateljski creep ili neprijateljski heroj

⁹zadnji udarac (engl. *last hit*)



Slika 2.4: Radius za iskustvo



Slika 2.5: Igrač (dolje desno) pokušava last hitati protivničkog creepa napadajući ga dok mu je health nizak.

srednjem (engl. *middle*) laneu (pogledati sliku 2.1). Također, ostali laneovi su “isključeni” tako da se na njima ne stvaraju creepovi te se tornjevi na njima ne mogu uništiti. Dodatno, oba igrača igraju istog heroja, Shadow Fienda.

Shadow Fiend ima četiri različite moći od kojih ćemo proučiti dvije najvažnije za ovaj format.

2.2.1. Shadowraze

Shadowraze je moć koja se sastoji od tri *podmoći* koje se mogu nezavisno koristiti. Shadowraze čini veliku štetu neprijateljima u malom radiusu ispred Shadow Fienda. Tri podmoći su identične s jedinom razlikom udaljenosti tog radiusa od Shadow Fienda. Ova moć ima veliku korist kod uspješnog last hitanja crepova te kod napadanja protivnika.

Vrlo bitna stvar na koju igrač mora paziti kod korištenja ove moći je orijentacija heroja. Nažalost, u Breezy serveru je postojala greška zbog koje je agent dobivao pogrešne informacije o svojoj orijentaciji te orijentaciji protivnika tako da agenti koji su trenirani u sklopu rada nisu mogli uspješno naučiti koristiti, a ni izbjegavati protivnički, Shadowraze.

2.2.2. Necromastery

Necromastery je pasivna moć koja povećava štetu udarca Shadow Fienda za svaki last hit ili deny koji napravi. Ova moć, posebno u 1v1 formatu, stvara učinak snježne grude (engl. *snowball effect*). Što heroj više last hita i denyja to će stvarati veću štetu kod udaraca. Što heroj stvara veću štetu to mu je lakše last hitati i denyjati. Zbog ove moći je izrazito veliki naglasak na što efikasnijem last hitanju te denyjanju. Onaj igrač koji bolje last hita i denyja ima puno bolje izglede za pobijediti.



Slika 2.6: Sve tri Shadowraze podmoći

Nubtrain — Nevermore — The Soultrain Stealer

2.3. Breezy

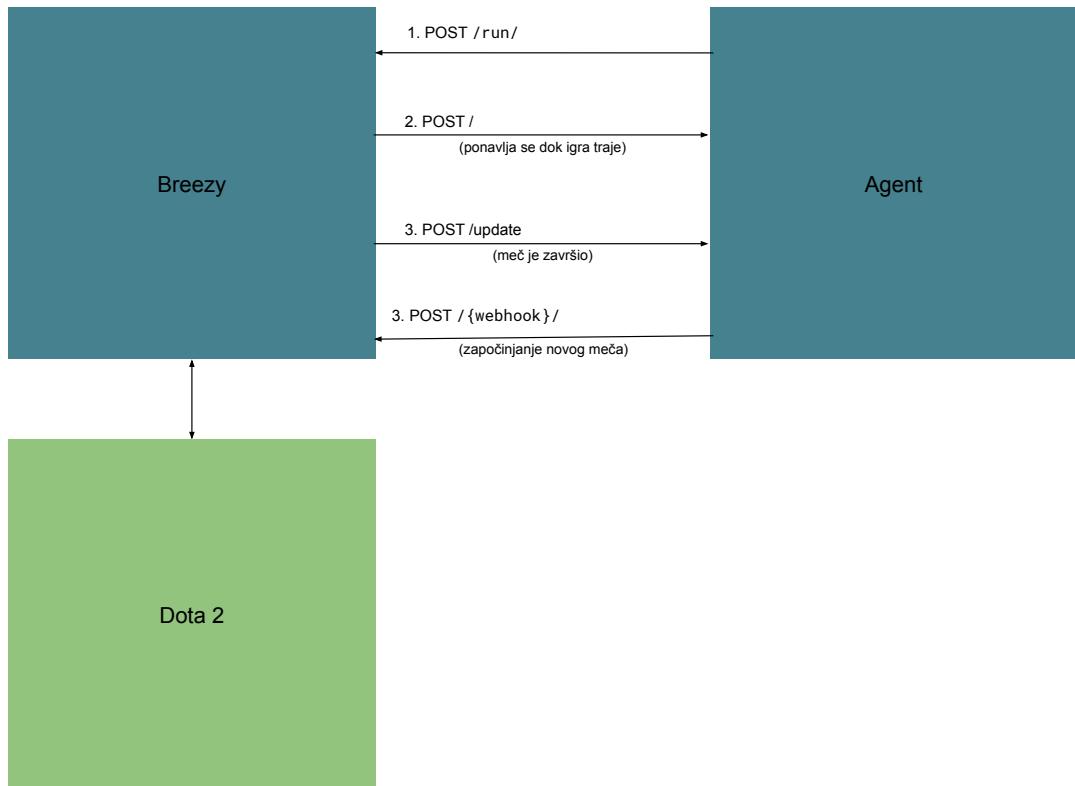
Breezy je HTTP¹⁰ server i klijent koji agentu služi kao sučelje prema igri. Upute za postavljanje servera mogu se naći na stranici natjecanja.[8]

Kako bi započeli igru, trebamo poslati POST zahtjev Breezy serveru na rutu /run/. U tijelu zahtjeva, u JSON¹¹ formatu, šljemo broj mečeva koliko ih želimo pokrenuti. Nakon toga, Breezy server pokreće igru te periodički šalje značajke igre agentu. Breezy server šalje 310 značajki putem POST zahtjeva na predefiniranu rutu na agentu. Agent

¹⁰Hyper Text Transfer Protocol

¹¹JavaScript Object Notation

zatim odgovara na te zahtjeve jednom od 30 dostupnih akcija.¹² Agent igra protiv hardkodiranog *bota* koji je dostupan u igri te koji je težine *hard*.¹³ Kada dođe do kraja igre, Breezy server šalje informacije o kraju igre putem POST zahtjeva na predefiniranu rutu na agentu. Ako je ostalo još mečeva za odigrati¹⁴, u tijelu tog zahtjeva nalazit će se webhook ruta na Breezy serveru na koju treba poslati GET zahtjev kako bi započeo novi meč.



Slika 2.7: Interakcija Breezy servera i agenta

Tri vrlo otežavajuća faktora agentima su nepoznavanje orijentacije svog ni protivničkog heroja, nemogućnost denjanja te veliki vremenski razmak između dvije akcije.

- Kao što je već spomenuto u poglavlju 2.2.1, postojala je greška u Breezy serveru zbog koje značajke za orijentacije heroja nisu radile te, radi toga, agenti nisu

¹²Popis značajki može se pogledati na dokumentaciji Breezy servera, a dostupne akcije na stranici natjecanja.

¹³Botovi dostupni u igri imaju više dostupnih težina: pasivan (engl. *passive*), lagan (engl. *easy*), srednje težak (engl. *medium*), težak (engl. *hard*) i nepošten (engl. *unfair*)

¹⁴Ako smo kod prvotnog zahtjeva prema Breezy serveru označili da želimo pokrenuti više od jednog meča

mogli naučiti koristiti ni izbjegavati Shadowraze moć.¹⁵

- Druga greška u Breezy serveru je nemogućnost denyjanja. Naime akcije za denyjanje creepova (njih 5), tijekom izrade rada, nisu radile.¹⁶
- Treći otežavajući faktor je brzina kojom Breezy server šalje značajke. Latencija od Breezy servera do igre je ~110ms. Sama igra je ubrzana deset puta, odnosno jedna sekunda stvarnog vremena je 10 sekundi u igri. Zbog velike latencije između Breezy servera i igre, Breezy server šalje, otprilike, 9 značajki po sekundi. S obzirom na to da je igra ubrzana deset puta, to znači da agent prima značajke i donosi akcije u razmacima od, otprilike, 0.1 sekunde stvarnog, odnosno 1 sekunde vremena u igri.

Postoje dvije mehanike igre na koje agent ne mora paziti; “levelanje” moći te kupovanje itema.[8] Kod svakog novog levela kojeg heroj dosegne, igrač mora odabratи moć heroja koju želi unaprijediti. Agent o tome ne mora brinuti jer breezy server ima točno određeni redoslijed unaprjeđivanja moći te automatski, kod svakog novog levela heroja, bira moć za unaprjeđivanje. Druga mehanika je kupovanje itema. Itemi su predmeti koje igrač može kupiti sa zarađenim zlatom te imaju veliki utjecaj na jačinu heroja. Kupovanje itema je također automatizirano tako da će se itemi, u točno određenom redoslijedu, kupiti čim igrač skupi dovoljno zlata.

¹⁵Značajke za orientacije su ispravljene sljedećim git commitom:
<https://gitlab.com/LivinBreezy/project-breezy/-/commit/e86d0a8a3a892182817892867a525941a7bb47f4>

¹⁶Akcije su ispravljene sljedećim git commitom: <https://gitlab.com/LivinBreezy/project-breezy/-/commit/078583ce5b19bc152995c1561eb173d4091fb560>

3. Programska implementacija

Rad je implementiran u jeziku Python¹ koristeći vanjske biblioteke *numpy* i *requests*. Numpy biblioteka koristi se u implementaciji neuronske mreže, a biblioteka requests koristi se za slanje HTTP zahtjeva Breezy serveru.

Kod se sastoji od 7 različitih komponenata koje su apstrahirane te se mogu nezavisno kombinirati. Svaka komponenta nalazi se u svom paketu unutar vršnog direktorija repozitorija te sadrži svoj apstraktni razred. Za svaku komponentu se može na lagan način, nasljeđivanjem definiranog apstraktnog razreda, dodati nova implementacija komponente. Dijagram razreda može se vidjeti na slici 3.1. Komponente su sljedeće:

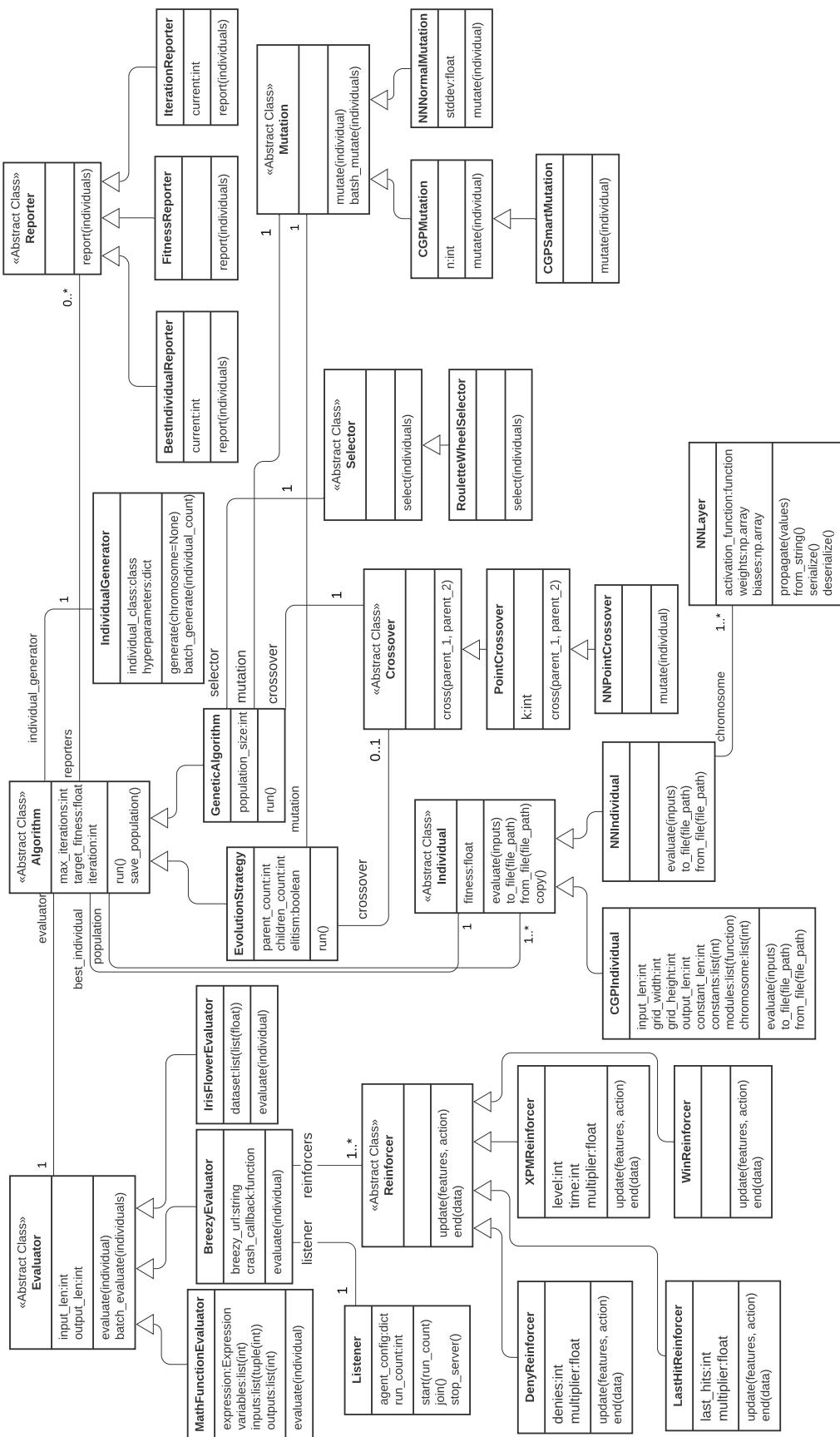
- algoritam (engl. *algorithm*)
- jedinka (engl. *individual*)
- selekcija
- križanje (engl. *crossover*)
- mutacija (engl. *mutation*)
- evaluator
- reporter

3.1. Algoritam

Apstraktni razred `Algorithm` predstavlja evolucijski algoritam. Razred se nalazi u modulu `algorithm` u paketu `algorithms`. U konstruktoru razreda `Algorithm` postavljaju se sljedeći parametri:

- `max_iterations` — Maksimalni broj iteracija algoritma.
- `target_fitness` — Ciljana dobrota (engl. *fitness*) jedinke. Nakon što dosegne ciljanu vrijednost dobrote, algoritam bi se trebao zaustaviti te spremiti najbolju jedinku.

¹Python verzija 3.8.3



Slika 3.1: UML dijagram razreda

- `individual_generator` — Tvornica jedinki. Objekt razreda `IndividualGenerator`
- `evaluator` — Evaluator, odnosno implementacija razreda `Evaluator`. Evaluator se koristi pri evaluaciji jedinki.
- `reporters` — Reporteri, odnosno lista objekata `Reporter`. Algoritam bi kod svake iteracije algoritma trebao zvati sve reportere putem njihove metode `report`.²

Razred sadrži članske varijable `best_individual`, `population` te `iteration` koje se mogu koristiti u implementaciji podrazreda. Razred sadrži apstraktну metodu `run` koju podrazredi moraju implementirati. Razred `Algorithm` sadrži i pomoćne metode koje se mogu koristiti u implementaciji podrazreda:

- `_save_best_individual` — Metoda sprema jedinku koja se nalazi u članskoj varijabli `best_individual` u tekstualnu datoteku u direktorij `best_individuals` u radnom direktoriju. Ime datoteke u koju se spremi jedinka je u sljedećem formatu `{datum i vrijeme}3-{ime algoritma}-{tip jedinke}4.txt`. Ako direktorij `best_individual` ne postoji u radnom direktoriju, metoda će ga stvoriti.
- `_report` — Metoda obavještava sve reportere o trenutnoj populaciji.
- `_stop_condition` — Metoda provjerava je li postignut zadani `target_fitness`. Vraća `True` ili `False`.
- `_check_iteration` — Inkrementira brojač iteracija (`iteration`) te provjerava je li postignut zadani `target_fitness` koristeći metodu `_stop_condition`.
- `save_population` — Metoda sprema trenutnu populaciju (članska varijabla `population`) u direktorij formata `{datum i vrijeme}5-{ime algoritma}-{tip jedinke}6`. Direktorij se stvara u direktoriju `saved_populations` u radnom direktoriju. Ako direktorij `saved_populations` ne postoji u radnom direktoriju, metoda će ga stvoriti.

Kod primitka signala SIGINT, algoritam se prekida te se poziva metoda `save_population`.

²Oblikovni obrazac promatrač

³Datum i vrijeme su u formatu ISO8601.

⁴Kao ime algoritma te tip jedinice koriste se pripadna imena razreda koji se koriste.

⁵Pogledati fusnotu 3

⁶Pogledati fusnotu 4

3.1.1. Evolucijska strategija

U sklopu rada implementiran je algoritam evolucijske strategije. Algoritam se nalazi u paketu `algorithms`, modulu `evolution_strategy`, pod imenom `EvolutionStrategy`. Parametri algoritma, koji se predaju konstruktoru, (osim onih opisanih u poglavlju 3.1) su:

- `parent_count` — Broj jedinki u roditeljskoj populaciji. (μ)
- `children_count` — Broj jedinki u populaciji djece. (λ)
- `mutation` — Mutacija (implementacija apstraktnog razreda `Mutation`) koja će se koristiti.
- `elitism` — Boolean. Ako je `True`, za stvaranje nove populacije roditelja koriste se najbolje jedinke iz obje populacije (i djece i roditelja) — $(\mu + \lambda)$ -ES. Ako je `False`, za stvaranje nove populacije roditelja koriste se najbolje jedinke isključivo iz populacije djece — (μ, λ) -ES.
- `crossover` — Križanje (implementacija apstraktnog razreda `Crossover`) koja će se koristiti. Neobavezan parametar. Ako se koristi križanje, jedinke u novoj populaciji djece stvaraju se križanjem dvaju nasumično odabralih jedinki iz populacije roditelja (te naknadnom mutacijom nastale jedinke). Ako se križanje ne koristi, jedinke u novoj populaciji djece stvaraju se mutacijom nasumično odabralih jedinki iz populacije roditelja.

3.1.2. Genetski algoritam

U sklopu rada implementiran je i genetski algoritam. Algoritam se nalazi u paketu `algorithms`, modulu `genetic_algorithm`, pod imenom `GeneticAlgorithm`. Parametri algoritma, koji se predaju konstruktoru, (osim onih opisanih u poglavlju 3.1) su:

- `selector` — Selekcija koja će se koristiti. Implementacija apstraktnog razreda `Selector`.
- `crossover` — Križanje koje će se koristiti. Implementacija apstraktnog razreda `Crossover`.
- `mutation` — Mutacija koja će se koristiti. Implementacija apstraktnog razreda `Mutation`.
- `population_size` — Veličina populacije algoritma.

3.2. Jedinka

Jedinka je definirana apstraktnim razredom `Individual` iz modula `individual` koji se nalazi u paketu `genotypes`. `Individual` sadrži člansku varijablu `fitness`, koja označava dobrotu jedinke, te 3 apstraktne metode:

- `evaluate` — Metoda vraća izlaze jedinke za ulaze⁷ koji su predani metodi kao argument.
- `to_file` — Metoda prima put do datoteke u koju je potrebno spremiti jedinku.
- `from_file` — Statička metoda koja vraća jedinku koja je spremljena u datoteci čiji put je predan kao argument.

Generator jedinki

Algoritam koristi generator jedinki⁸ kako bi stvorio nove jedinke bez da je vezan za samu implementaciju jedinke. Generator je predstavljen razredom `IndividualGenerator` u modulu `individual` paketa `genotypes`. Konstruktor generatora prima 3 parametra:

- `individual_class` — Razred implementacije jedinke koja će se koristiti.
- `hyperparameters` — Hiperparametri jedinki sadržani u obliku rječnika⁹. Hiperparametri ovise o implementaciji jedinke koja se koristi.
- `population_path` — Opcionalni parametar. Put do direktorija u kojem se nalaze jedinke koje želimo koristiti kako početnu točku algoritma (umjesto nasumičnih jedinki).

Generator sadrži dvije metode koje algoritam može zvati kako bi generirao nasumične jedinke. Ako se pri inicijalizaciji generatora predao `population_path` parametar, metode će vraćati učitane jedinke, a nakon toga, kada sve učitane jedinke budu “iskorištene”, vraćat će nasumične jedinke.

- `generate` — Metoda vraća nasumično generiranu ili učitanu jedinku.
- `batch_generate` — Metoda vraća listu nasumično generiranih ili učitanih jedinki veličine specificirane parametrom `individual_count`.

⁷listu cijelih ili realnih brojeva

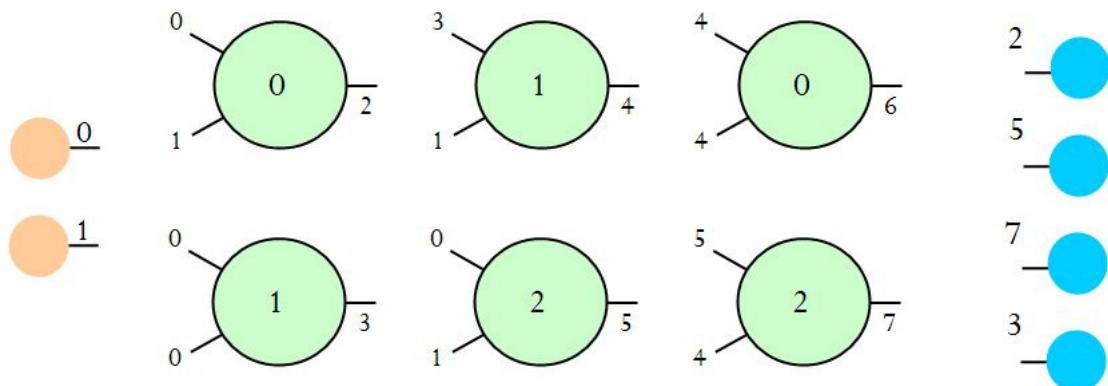
⁸Oblikovni obrazac tvornica

⁹Rječnik (engl. *dictionary*) je kolekcija uređenih parova (ključ, vrijednost) u Pythonu.

3.2.1. Kartezijsko genetsko programiranje

Kartezijsko genetsko programiranje (engl. *Cartesian genetic programming*) oblik je genetskog programiranja u kojem umjesto stablaste strukture, jedinku predstavlja fiksna dvodimenzionalna koordinatna mreža čvorova. Svaki čvor predstavlja funkciju s dva ulaza i jednim izlazom, na primjer: zbroj, razlika, umnožak. Čvor može biti i funkcija jednog ulaza tako da jedan od ulaza koristimo, a drugi ignoriramo. Na primjer, neke od funkcija s jednim ulazom bile bi trigonometrijske funkcije sinusa i kosinusa. Jednu CGP jedinku određuje razmještaj funkcija po čvorovima te međusobna povezanost čvorova. Na ulaze svakog čvora dovodimo ulaze u jedinku, izlaze iz drugih čvorova ili predefinirane konstante. Na čvor smijemo povezati izlaze samo onih čvorova koji su u prijašnjim slojevima¹⁰.

Primjer CGP jedinke veličine 3×2 s 2 ulaza, 4 izlaza te bez konstanti može se vidjeti na slici 3.2.



Slika 3.2: Primjer CGP-a

Miller, PPSN 2014 Tutorial: Cartesian Genetic Programming

CGP jedinka implementirana je razredom `CGPIndividual` u modulu `cgp` paketa `genotypes.cgp`. Parametri, koji se predaju konstruktoru su sljedeći:

- `input_len` — Broj ulaza u jedinku.
- `grid_size` — Dimenzije jedinke. Na primjer, `grid_size` jedinke s 30 stupaca i 20 redaka je $(30, 20)$.
- `output_len` — Broj izlaza iz jedinke.
- `constants_len` — Broj konstanti u jedinki.

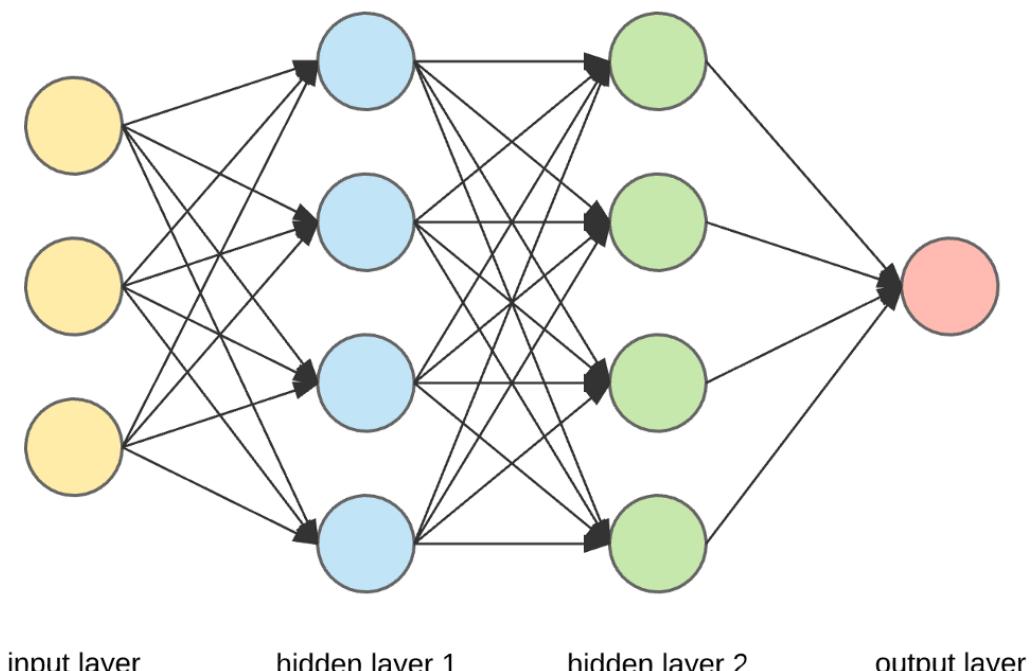
¹⁰Prijašnji slojevi — bliži ulazima u jedinku

- constants — Lista konstanti koje se mogu pojaviti u jedinku.
- modules — Lista funkcija za čvorove. Funkcije moraju primati dva argumenta te vratiti cijeli ili realni broj. Primjer funkcija može se naći u modulu `modules` paketa `genotypes.cgp`.

3.2.2. Neuronska mreža

Model neuronske mreže implementiran je razredom `NNIndividual` u modulu `nn` paketa `genotypes.nn`. Parametri, koji se predaju konstruktoru, su sljedeći:

- layers — Lista cijelih brojeva koja predstavlja broj ulaza, broj i veličinu slojeva te broj izlaza. Primjerice, za neuronsku mrežu sa slike 3.3 parametar `layers` bio bi sljedeći $(3, 4, 4, 1)$.
- activation_functions — Lista aktivacijskih funkcija. Svaka funkcija u listi predstavlja aktivacijsku funkciju za jedan sloj neuronske mreže. Duljina liste aktivacijskih funkcija uvijek je za jedan manja od duljine parametra `layers`. Primjerice, ako 2 skrivena sloja mreže sa slike 3.3 koriste sigmoid funkciju, a izlazni sloj relu funkciju, parametar `activation_functions` je $(\text{sigmoid}, \text{sigmoid}, \text{relu})$.



Slika 3.3: Primjer neuronske mreže

Yang, Build up a Neural Network with Python

3.3. Selekcija

Selekcija (engl. *selection*) predstavljena je apstraktnim razredom Selector modula selector paketa selections. Uloga selekcije je odabrat jednu jedinku iz populacije evaluiranih jedniki¹¹. Selekcije to čine implementirajući apstraktну metodu select koja prima listu jedinki te vraća izabranoj jedinku. Metoda također prima i optionalni parametar not_same_as putem kojeg se može predati jedinku koju ne želimo da se izabere. Primjerice, ako trebamo izabrati 2 različita roditelja u svrhu križanja, nakon što smo izabrali prvog roditelja, prilikom pozivanja metode select za izbor drugog roditelja, možemo se pobrinuti da ne dobijemo istu jedinku predavanjem prvog roditelja parametrom not_same_as.¹²

U sklopu rada implementirana je proporcionalna selekcija (engl. *Roulette-wheel selection*) razredom RouletteWheelSelector modula roulette_wheel_selector paketa selections.

3.4. Križanje

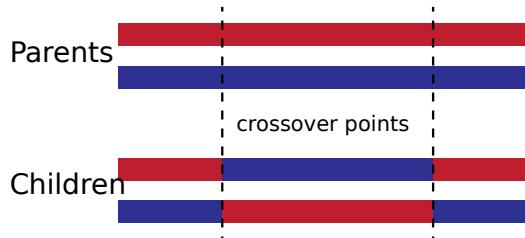
Križanje (engl. *crossover*) predstavlja način na koji kombiniramo dvije jedinke roditelja kako bi dobili djecu. Križanje je definirano apstraktnim razredom Crossover modula crossover paketa crossovers. Razred ima jednu apstraktну metodu, cross. Metoda prima dvije jedinke roditelja te vraća listu djece.

Napisane su dvije implementacije križanja s n-točaka prekida (engl. *n-point crossover*) u paketu crossovers. Razred PointCrossover modula point_crossover predstavlja generičnu implementaciju križanja s n-točaka prekida koja radi na bilo kojoj implementaciji jedinke kojoj je kromosom predstavljen listom u članskoj varijabli chromosome.¹³ Implementacija neuronske mreže nema tako predstavljen kromosom, za nju je implementiran podrazred NNPointCrossover u modulu nn_point_crossover. Metoda cross obje implementacije vraća listu s dvije jedinke djeteta.

¹¹Jedinke imaju postavljenu člansku varijablu dobrote — fitness

¹²Selekcija implementirana u sklopu rada ne uspoređuje jednakost jedinki već samo osigurava da se ne odabere isti objekt jedinke koji je predan parametrom not_same_as. Dakle, ako postoje dvije efektivno iste jedinke u populaciji, predavanjem jedne od njih kroz parametar not_same_as neće osigurati da se druga jedinka neće izabrati.

¹³S obzirom na to da mutacije i križanja nisu u potpunosti apstrahirani od modela jedinki, očekuje se od korisnika da zna koja križanja i mutacije može kombinirati s modelom jedinke koji koristi.



Slika 3.4: Križanje s dvije točke prekida

CC BY-SA 3.0 — Rooland

3.5. Mutacija

Mutacija (engl. *mutation*) je način uvođenja nasumičnih promjena u jedinku. Svrha mutacije je istraživanje prostora stanja te izbjegavanje lokalnih optimuma.[1] Mutacija je definirana apstraktnim razredom `Mutation` modula `mutation` u paketu `mutations`. Razred `Mutation` sadrži apstraktну metodu `mutate` koju podrazredi trebaju implementirati. Metoda `mutate`, kao argument, prima jedinku koju je potrebno mutirati. Razred `Mutation` sadrži i metodu `batch_mutate` kojoj se predaje lista jedinki koje treba mutirati.

3.5.1. CGP mutacije

U paketu `mutations.cgp` implementirane su dvije mutacije za CGP jedinke; `CGPMutation` te `CGPSmartMutation`. `CGPMutation` iz modula `simple_mutation` implementacija je mutacije u n-točaka (engl. *n-point mutation*). Konstruktor prima parametar `n` — broj mutacija. Pozivanjem metode `mutate`, jedinka će mutirati na `n` različitim mjestima u kromosomu.

Bitna značajka CGP modela jedinke jest neaktivni dio genotipa — čvorovi koji se ne koriste, odnosno čiji izlaz ne koristi niti jedan drugi čvor. Kod jedinke veličine 1500 čvorova¹⁴ udio aktivnog genotipa je u prosjeku ~7.5%.[4] Sukladno tome, možemo zaključiti da veći broj nasumičnih mutacija će pogoditi upravo neaktivni dio genotipa te neće uzrokovati vidljive promjene na jedinku. Mutiranje neaktivnog dijela genotipa naziva se neutralni drift (engl. *neutral drift*) te predstavlja način pretraživanja prostora stanja.[3] Dodatno, pokazalo se da CGP postiže bolje rezultate kod mnogo veće stope mutacije neaktivnog dijela genotipa.[7] Stoga je razredom `CGPSmartMutation` u modulu `smart_mutation` implementiran podrazred mutacije `CGPMutation` koja razlikuje aktivni od neaktivnog dijela genotipa. Pozivom metode `mutate` razreda

¹⁴1500 čvorova je veličina jedinki koja se kasnije koristila na treniranju agenta za igru.

CGPSmartMutation, dogodit će se n mutacija isključivo na aktivnom dijelu genotipa dok će neaktivni dio genotipa potpuno mutirati.

3.6. Reporter

Apstraktni razred Reporter predstavlja elegantan način praćenja tijeka algoritma. Algoritam se inicijalizira s listom predanih Reporter-a te kod svake iteracije algoritam zove metodu report svakog od njih¹⁵. Razred Reporter nalazi se u modulu reporter paketa reporters te sadrži apstraktну metodu report koja, kao argument, prima listu jedinki.

Implementirana su tri različita razreda reporter-a, u paketu reporters, koji ispisuju informacije na standardni izlaz:

- BestIndividualReporter iz modula best_individual_reporter ispisuje pojavu nove najbolje jedinke te njenu dobrotu.
- FitnessReporter iz modula fitness_reporter ispisuje prosječnu dobrotu populacije svake iteracije.
- IterationReporter iz modula iteration_reporter jednostavan je reporter koji ispisuje trenutnu iteraciju algoritma.

3.7. Evaluator

Evaluator je razred koji je zadužen za evaluaciju jedinki te koji definira problem koji se rješava. Predstavljen je apstraktnim razredom Evaluator modula evaluator u paketu evaluators. Razred sadrži apstraktну metodu evaluate koja prima jedinku koju treba ocijeniti te joj dobrotu zabilježiti u njenoj članskoj varijabli fitness. Razred Evaluator sadrži i metodu batch_evaluate koja prima listu jedinki koje treba ocijeniti.

3.7.1. Evaluator klasifikacije iris cvijeta

U svrhu testiranja genetskog algoritma te implementacije neuronske mreže, implementiran je razred IrisFlowerEvaluator koji evaluira jedinke nad poznatim skupom podataka o cvijeću iris (engl. *Iris flower data set*). Razred se nalazi u modulu

¹⁵Obliskovni obrazac promatrač.

`iris_flower` paketa `evaluators.iris`. Ukupan broj podatkovnih točaka (engl. *datapoints*) je 150. Dobrota se računa kao broj ispravno pogodjenih točaka. U modulu `flower_classification` u vršnom direktoriju repozitorija nalazi se primjer pokretanja genetskog algoritma s neuronskom mrežom te `IrisFlowerEvaluator`om.

3.7.2. Evaluator simboličke regresije

Evaluator `MathFunctionEvaluator` iz modula `math_function_evaluator` paketa `evaluators` evaluira dobrotu jedinki kod simuliranja zadane matematičke funkcije. Razred je implementiran u svrhu testiranja algoritma evolucijske strategije i CGP jedinke.

Kod inicijalizacije evaluatora, predaje se matematički izraz poput ' $x - 8*y - z$ ' te kombinacije vrijednosti parametara¹⁶ s kojima će se evaluirati jedinke. Primjer korištenja `MathFunctionEvaluator`a, u kombinaciji s evolucijskom strategijom te CGP implementacijom jedinke, može se vidjeti u modulu `symbolic_regression` u vršnom direktoriju repozitorija.

3.7.3. Breezy evaluator

Razred `BreezyEvaluator` modula `breezy` u paketu `evaluators.breezy` implementacija je evaluatora za igru. Evaluator prilikom inicijalizacije prima 4 parametra:

- `reinforcers` — Lista objekata koji implementiraju apstraktni razred `Reinforcer`, odnosno, koji su zaduženi za računanje dobrote jedinki.
- `listener_address` — IP¹⁷ adresa i port na lokalnom računalu na kojem će `BreezyEvaluator` slušati HTTP zahtjeve od `Breezy` servera. Primjer: ('127.0.0.1', 8086).
- `breezy_url` — URL¹⁸ od `Breezy` servera.
- `crash_callback` — Opcionalni parametar. Može se predati funkcija koja će se pozvati u slučaju da u igri i/ili `Breezy` serveru dođe do greške te se algoritam mora prekinuti. Preporučljivo je postaviti metodu `save_population` algoritma, kako se spremila trenutna populacija prije nego što se algoritam prekine.

`crash_callback` metoda može se postaviti i nakon inicijalizacije evaluatora pozivom metode `register_crash_callback`.

¹⁶U obliku dvodimenzionalne liste

¹⁷Internet Protocol

¹⁸Uniform Resource Locator

Evaluator implementira metode `evaluate` i `batch_evaluate`, međutim, kod evaluacije više jedinki, preporučljivo je koristiti metodu `batch_evaluate` zbog manjeg *overheada* prilikom pokretanja igre.

Breezy evaluator tijekom igre, putem HTTP POST zahtjeva, prima značajke od Breezy servera, djelomično ih, na jednostavan način, obrađuje te ih šalje na ulaz jedinke pozivom metode `evaluate`. Kao akcija koju jedinka odigra uzima se indeks najvećeg izlaza jedinke.¹⁹ Značajke i odigrana akcija šalju se svim reinforcerima putem njihove metode `update`²⁰, a nakon toga se Breezy serveru šalje akcija kao odgovor na HTTP POST zahtjev sa značajkama. Kada dođe do kraja igre, svim reinforcerima, putem metode `end`, predaju se informacije o završetku igre. Reinforceri pozivom metode `end` vraćaju iznos dobrote koji je potrebno pridodati ukupnoj dobroti jedinke. Dakle, ukupna dobrota jedinke računa se kao zbroj dobrota koje je dobila od svih reinforcera.

Breezy evaluator prima od Breezy servera 310 različitih značajki. Neke od značajki su nepotrebne te se zanemaruju, neke se bez izmjena propuštaju prema jedinki, a neke se na jednostavan način obrađuju. Primjer značajke koja se zanemaruje je značajka `team` koja označava pripadnost timu.²¹ Značajka je beskorisna jer se jedinke nalaze uvejk na istom timu. Primjer obrađene značajke su značajke `health` i `max health` koje označavaju trenutni te maksimalni `health` našeg heroja. Te dvije značajke Breezy evaluator kombinira u jednu značajku — normalizirani `health` heroja.

Ukupan broj značajki koje jedinke primaju je 111. Za detaljniji pregled značajki koje jedinke primaju, može se pogledati metoda `_process` razreda `BreezyEvaluator`.

Akcije koje jedinka može donijeti su sve akcije koje podržava Breezy server osim 4 akcije za skupljanje runa (engl. *rune*).²² Akcije za skupljanje runa su isključene zbog pogrešnih značajki koje Breezy server šalje o njihovim lokacijama zbog kojih agenti ne bi mogli naučiti efikasno korisiti te 4 akcije.

Reinforcer

Ideja iza reinforcera je razbijanje puta do pobjede na više manjih ciljeva koji osiguravaju dobre izglede za pobjedu. Reinforceri na temelju akcija koje jedinke donose u određenim situacijama pozitivno ili negativno ocjenjuju njihovo ponašanje.

Implementacija apstraktnog razreda `Reinforcer` nalazi se u modulu `reinforcer`

¹⁹Broj izlaza jedinke jednak je broju mogućih akcija

²⁰Obliskovni obrazac promatrač

²¹Dolje lijevo ili gore desno na slici 2.1.

²²U sklopu rada nije objašnjena mehanika runa upravo iz razloga što su akcije za njihovo skupljanje isključene.

paketa `evaluators.breezy.reinforcers`. Razred sadrži dvije apstraktne metode:

- `update` — Metodi se predaje lista značajki te akcija koju je jedinka odigrala. Metoda se zove kod svake akcije jedinke.
- `end` — Metodi se predaju informacije o završetku igre²³. Kao povratna vrijednost vraća se ukupan fitness koji je jedinka zaradila kod tog reinforcera.

U paketu `evaluators.breezy.reinforcers` implementirani su sljedeći reinforceri:

- `DenyReinforcer` — Dodaje vrijednost $multiplier * broj_denyjeva$ dobroti jedinke. `multiplier` se postavlja prilikom inicijalizacije reinforcera.²⁴
- `LastHitReinforcer` — Dodaje vrijednost $multiplier * broj_last_hitova$ dobroti jedinke. `multiplier` se postavlja prilikom inicijalizacije reinforcera.
- `XPMReinforcer` — Dodaje vrijednost $multiplier * XPM$ ²⁵ dobroti jedinke. `multiplier` se postavlja prilikom inicijalizacije reinforcera.
- `WinReinforcer` — Dodaje 1000000 bodova jedinki, ako pobjedi meč.

²³Podaci koji se predaju na završetku igre definirani su u dokumentaciji Breezy servera: https://app.swaggerhub.com/apis-docs/aianta/breezy-agent/0.0.1#/default/post_run_update_route

²⁴Tek nakon izrade rada je otkriveno da, zbog greške u Breezy serveru, akcije za denjanje creepova nisu radile. Stoga ovaj reinforcer nije imao nikakvog utjecaja u evaluaciji jedinki.

²⁵XPM (engl. *Experience per minute*) — broj iskustvenih bodova po minutu

4. Rezultati

Jedinke su trenirane iterativnim zaustavljanjem algoritma, namještanjem parametara reinforcera te ponovnim nastavljanjem rada algoritma. Svrha mijenjanja parametara reinforcera, odnosno funkcije cilja je usmjeravanje ponašanje agenata u željenom smjeru. Samim time, vrijednost dobrote jedinki u različitim etapama algoritma ne mogu se uspoređivati te je stoga u ovom poglavlju dan osvrt na rezultate u obliku opisa ponašanja jedinki tijekom različitih faza naučenosti.

Funkciju cilja čini suma funkcija cilja svih reinforcera koji se koriste. Za jedinke trenirane u sklopu rada funkcija cilja je sljedeća:

$$F = k_{\text{deny}} * N_{\text{deny}} + k_{\text{last_hit}} * N_{\text{last_hit}} + k_{\text{XPM}} * XP/min + k_{\text{win}} * win$$

Oznakom k označeni su koeficijenti koji se predaju prilikom inicijalizacije pripadnim reinforcerima. Svi koeficijenti su mijenjani tijekom različitih faza naučenosti agenta, osim vrijednosti koeficijenta WinReinforcera (k_{win}), koji je uvijek bio iznosa 1000000. win je vrijednosti 0 ili 1, ovisno o tome je li agent izgubio ili pobjedio.

U sklopu rada koristio se algoritam evolucijske strategije u kombinaciji s CGP jedinkama¹ te genetski algoritam u kombinaciji s neuronskim mrežama². Obje varijante trajale su 500 iteracija algoritma.³ Kao što je navedeno u prošlom odlomku, algoritmi nisu trajali 500 iteracija u jednom pokretanju već su više puta prekidani⁴ kako bi se namjestili koeficijenti reinforcera i utjecaj mutacije te time usmjerilo ponašanje jedinki u željenom smjeru.

U varijanti CGP-ES koristio se CGP sa sljedećim hiperparametrima:

- `grid_size = (50, 30)`
- `constant_len = 4`

¹Varijanta CGP-ES

²Varijanta NN-GA

³S obzirom na parametre koji su se koristili kod algoritama, a koji su navedeni niže u poglavlju, obje varijante koristile su, približno, 5000 evaluacija.

⁴koristeći mogućnost spremanja populacije te ponovnog korištenja spremljene populacije

- constants = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, π)
- modules = (zbroj, razlika, produkt, kvocijent, sinus, kosinus, negativ, min, max)

Koristila se mutacija CGPSmartMutation s n=3 u prvih 250 iteracija, a s n=6 u posljednjih 250 iteracija. Evolucijska strategija koristila je populaciju roditelja veličine 2, populaciju djece veličine 10, uz elitizam — (2 + 10)-ES

Varijanta NN-GA koristila je neuronske mreže s 30 skrivenih slojeva od 200 neurona. Aktivacijska funkcija sigmoid koristila se u svim slojevima. Genetski algoritam imao je populaciju veličine 10, proporcionalnu selekciju, križanje u 5 točaka prekida te Gaussovnu mutaciju⁵ standardne devijacije 0.3 u prvih 250 iteracija algoritma te 0.5 u posljednjih 250 iteracija.

Obe varijante pokazivale su slične promjene ponašanja s razlikom da je CGP-ES varijanta, na kraju, pokazivala bolje rezultate. Opis ponašanja jedinki tijekom trajanja algoritama može se, ugrubo, podijeliti na sljedeće stadije:

1. Nasumične jedinke — jedinke rade nasumične akcije ili stoje na mjestu dok protivnik ne pobjedi uništavanjem tornja.
2. Jedinke su naučile doći na mid lane, međutim s malim zakašnjenjem zbog kojeg bi propustile prva 2 vala creepova. Jedinke uspiju postići 1–3 last hita, međutim vrlo lagano umru od strane protivnika.
3. Jedinke su naučile da je protivnik opasan te, radi toga, stoje u sigurnosti svog tornja. Tek kad se protivnički creepovi približe tornju jedinke ih napadaju.
4. Jedinke su “shvatile” radijus unutar kojega dobivaju iskustvo od protivničkih creepova koji umiru. Jedinke drže sigurnu udaljenost od protivničkog heroja, u isto vrijeme, osiguravajući da su creepovi unutar radijusa za dobivanje iskustva.
5. Jedinke su naučile odmah, na početku meča, doći na mid lane te više ne propuštaju prvi 2 vala creepova.
6. Jedinke su naučile da protivnik zna povremeno otići s lanea te koriste takve situacije tako da izađu iz sigurnosti svog tornja i krenu napadati creepove. Kad se protivnik vratи na lane, jedinke se vraćaju pod sigurnost svog tornja. Također, kada napadaju creepove, jedinke su naučile fokusirati ranged creepa⁶. Razlog je

⁵NNNormalMutation

⁶Creep koji napada iz daljine. Po jedan je u svakom valu. Pogledati slike 2.2–2.5

što ranged creep ima manji health te ga je stoga lakše ubiti, a daje više zlata i iskustva od ostalih creepova.

NN-GA varijanta uspjela je doći samo do stadija 4 dok je CGP-ES uspjela doći do stadija 6. Dva vrlo bitna aspekta koje jedinke nisu ni u jednom trenutku naučile su denyjanje te korištenje Shadowraze moći⁷.

⁷Jedinke nisu mogle naučiti koristiti Shadowraze moć ni denyjati zbog greški u Breezy serveru.

5. Zaključak

Dani zadatak nezahvalan je iz više razloga:

- Jedinke nisu imale točnu informaciju o orijentaciji svog ni protivničkog heroja. Rezultat toga je da nisu mogle naučiti koristiti svoju niti izbjegavati protivničku Shadowraze moć.
- Jedinke nisu mogle denyjati pošto akcije za denyjanje nisu funkcionalne.
- Jedinke primaju značajke te donose akcije svakih ~1 sekundu što iznimno otežava last hitanje te efikasno donošenje akcija.
- Evaluacija jedne jedinke traje prilično dugo — 1–2 minute stvarnog vremena. Posljedica toga je da se algoritmi moraju izuzetno dugo vrtjeti te je, efektivno, onemogućeno promatranje utjecaja različitih hiperparametara algoritama i jedinki.

Usprkos svemu tome te usprkos tome što su jedinke dobivene ovim radom vrlo daleko od pobjede, prilično sam ugodno iznenađen stvarima koje su naučile.

Iznimno me impresionirala činjenica da su jedinke, u obje varijante, naučile radijus unutar kojeg dobivaju iskustvo s obzirom na to da taj radijus nije ni na koji način hardkodiran u jedinke. Jedinke su isključivo uz pomoć XPMReinforcera uspjele shvatiti mehanizam dobivanja iskustva.

Druga stvar koja me impresionirala je što su jedinke naučile obrasce kretanja protivnika. Naime, bot protiv kojeg jedinke igraju povremeno tijekom meča odlazi s lana te ga nema na nekoliko desetaka sekundi. Jedinke su shvatile da je to vrijeme kada se ne moraju skrivati pod sigurnosti svog tornja te se mogu približiti valu creepova i tako postići veći broj last hitova.

Bitno je napomenuti da, za vrijeme pisanja ovog rada, Breezy server nije ponudio mogućnost mijenjanja težine protivnika te su se sve evaluacije odigrale protiv teške (engl. *hard*) varijante defaultnog bota. Prvotno je bilo planirano jedinke trenirati protiv slabije verzije protivnika te, postepeno, povećavati težinu.

LITERATURA

- [1] Domović D. Vizualizacija rada algoritama zasnovanih na evolucijskom računanju. *Fakultet elektrotehnike i računarstva*, 2012. Diplomski rad.
- [2] Liquipedia. The international 9. URL https://liquipedia.net/dota2/The_International/2019. Datum pristupa: 2020-06-01.
- [3] Julian F Miller. Cartesian genetic programming, cartesian genetic programming (julian f. miller, ed.). *Natural Computing Series, Springer*, stranice 17–34, 2011.
- [4] Julian F Miller i Stephen L Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.
- [5] OpenAI. More on dota 2, . URL <https://openai.com/blog/more-on-dota-2/>. Datum pristupa: 2020-06-01.
- [6] OpenAI. Openai five, . URL <https://openai.com/projects/five/>. Datum pristupa: 2020-06-01.
- [7] Eduardo Pedroni. An explicitly neutral mutation operator in cartesian genetic programming.
- [8] Robert Smith i Malcolm Heywood. Project breezy. URL <https://web.cs.dal.ca/~dota2/>. Datum pristupa: 2020-06-01.

Podržana evolucija agenta za igru Dota 2

Sažetak

Podržano učenje agenta za igru Dota 2 koristeći evolucijske algoritme evolucijske strategije i genetskog algoritma te modele jedinke kartezijskog genetskog programiranja i neuronske mreže.

Ključne riječi: CGP,GA,ES,NN,Dota

Reinforced evolution of Dota 2 agent

Abstract

Reinforced evolution of Dota 2 agent using evolutionary algorithms evolution strategy and genetic algorithm as well as models of cartesian genetic programming and neural network.

Keywords: CGP,GA,ES,NN,Dota