Diferencialna evolucija za učenje agenta umetne inteligence pri igranju splošnih videoiger

Aleš Zamuda, Matjaž Vöröš

Univerza v Mariboru, FERI, Koroška cesta 46, 2000 Maribor
E-pošta: ales.zamuda@um.si, matjaz.voeroes@student.um.si

Differential Evolution for Artificial Inteligence Agent Learning in General Video Game Playing

Abstract. *In recent years, a new chapter for video gaming has opened in the form of General Video Game Artificial Intelligence (GVG-AI) Competition. The Competition challenges the contestant to implement an agent that can maximize the score of played video games with usage of modern optimization algorithms. In this paper, a GVG-AI agent built upon recent optimization algorithms is hence presented. This agent uses Differential Evolution (DE), which was not yet used in a GVG-AI Competition. The performance of the agent is assessed and compared using the GVG-AI benchmark and the results demonstrate that the agent is statistically significantly better than most of the existing ones, but there is still room for improvement.*

1 Uvod

V pametni proizvodnji postajajo z razvojem virtualizacijske tehnologije in zajemanja podatkov vse bolj ključne digitalne sence (digitalni dvojčki) [1]. Tako je tudi pri proizvodnji igralcev računalniških iger, ki skozi digitalno virtualnost dobivajo obliko računalniškega agenta napram poprej le človeških igralcev teh iger. Tak agent vsebuje umetno inteligenco (UI), angl. *Artificial Intelligence* (AI), ki je računska, angl. *Computational Intelligence* (CI), in opravlja operacijo v danem okolju raziskave (angl. *Operational Research* [2]), v tem primeru, igre. V tem prispevku je tako predstavljen pristop k modeliranju in proizvodnji takega agenta UI (AI/CI), kjer proizvodnjo vodimo s pomočjo optimizacijskega algoritma. S tem je tako izhodni rezultat kot učenje proizvedeno v UI, zato tak razvoj igralnega agenta lahko umestimo tudi v področje umetnega življenja (angl. *Artificial Life,* AL) [2].

Rešitve, ki s pomočjo UI igrajo določeno igro ali pa so narejene z določenim namenom, redko služijo čemu drugemu kot lastni domeni [3]. Pravo testno okolje za realno-časovne agente sta tako ogrodji ALE (angl. *Association for the Advancement of Artificial Intelligence*) [4] kot GVG-AI (angl. *General Video Game AI Competition*) [5]. Razlike med ogrodjema ALE in GVG-AI so predvsem, da ogrodje GVG-AI agenta neposredno informira o stanju okolja s pomočjo javanskih razredov in mu omogoča, da stanje razvija ali kopira, s tem pa dobiva informacije o možnih prihodnjih stanjih, v katere lahko napreduje iz trenutnega [3].

Pod pojmom agent sicer razumemo napravo ali program, ki sprejema podatke iz okolice s pomočjo svojih senzorjev in nanjo deluje z aktuatorji [6][7]. Pri inteligentnih agentih je naloga UI, da predstavlja takega agenta, ki signale pretvori v akcije [7]. Pri GVG-AI potrebujemo agenta**,** temelječega na cilju (angl. *Goal-Based Agent*) [7]. Poznati nekaj o trenutnem stanju okolja ni vedno dovolj za določitev naslednje akcije; torej na pravilno odločitev vpliva cilj, do katerega želimo priti [7]. Pot do cilja se tako določa s pomočjo iskalnih strategij in planiranja [6][7]. Različna uporaba UI v videoigrah je tako koristna za oblikovanje boljših in zanimivejših iger. UI ima pri igranju dve vidnejši vlogi [8]: ali videoigro igra kot igralec, s tem izboljša avtomatiziranje igre in omogoča lažji pregled ter ocenitev videoigre ali pa v videoigri sodeluje kot nasprotnik, kar lahko privede do lažje dinamične nastavitve težavnosti videoigre.

Za iskanje optimalne rešitve je potrebno definirati problem in izbrati ustrezno metodo [9]. Pri optimizacijskih problemih imamo podane množice možnih rešitev, kriterijsko funkcijo za vrednotenje in določene omejitve, iščemo pa najboljšo možno rešitev problema oziroma rešitev z najboljšo vrednostjo ocenitvene funkcije [10]. Pojem optimizacije je sicer precej širok, v splošnem pa se algoritmi za optimizacijo uporabljajo tako za UI, kot pri splošnem računanju [8]. Evolucijski algoritmi (EA) so globalni optimizacijski algoritmi [2], saj za razliko od lokalnih optimizacijskih algoritmov [3] [4] točke na začetku razporedimo po večjem prostoru, z njimi pa iščemo najboljšo možno globalno rešitev v nelinearnem odločitvenem modelu, ki pogosto vsebujejo nepopolne lokalne rešitve [5]. EA uporabljamo predvsem za reševanje težkih optimizacijskih problemov, npr. na področju strojnega učenja [2] [11]. V nadaljevanju predstavljen zasnovan agent iz [12] razvija načrt v navideznem modelu (digitalni senci) nekaj milisekund, nato pa deluje v videoigri skozi izvedbo prve akcije v snovanem načrtu ter se ponovno vrne v sprotno razvijanje tega načrta za igro [11]. Agent se od predhodnih agentov razlikuje po tem, da v učenju z EA vključuje nekatere izbrane operatorje modernih optimizacijskih algoritmov [2]. Za tega agenta poročamo tako teste in rezultate, ki jih primerjamo z drugimi pristopi nad operacijo igranja videoiger.

V naslednjem poglavju so predstavljena sorodna dela, v poglavju 3 sledi predstavitev snovanja algoritma GVG-AI in implementacija, nato so v poglavju 4 podani rezultati in v poglavju 5 zaključek.

2 Sorodna dela

V tem poglavju je najprej opisana diferencialna evolucija za optimizacijo, nato sledi še opis ogrodja za tekmovanje inteligentnih agentov za igranje videoiger.

2.1 Diferencialna evolucija za optimizacijo

Osnovno predlogo diferencialne evolucije (lahko tudi diferenčne evolucije, v nadaljevanju DE) [13] lahko opišemo v korakih EA iz [8]:

a) *Inicializacija*: začetno populacijo napolnimo z rešitvami, ki jih generiramo naključno; na primer naključne točke razporejene po iskalnem prostoru. Če že poznamo dobre rešitve, jih lahko vključimo v začetno populacijo. Naj bo $\boldsymbol{I}$ poljubna množica (prostor posameznikov) rešitev $\boldsymbol{x}_{i,G}$, ki tvorijo populacijo posamezne generacije G z velikostjo populacije μ^G, kjer je i v obsegu od $1,2,\dots,\mu^G;\ \mu^G, G \in \mathbb{N}$ [14].

Začetno populacijo v času $G = 0$ zapišemo kot nam kaže enačba (1). Pri DE se morajo komponente rešitve nahajati v mejah komponent $\boldsymbol{B}_{\min} = [B_{\min_1}, B_{\min_2}, \dots, B_{\min_D}]$ in $\boldsymbol{B}_{\max} = [B_{\max_1}, B_{\max_2}, \dots, B_{\max_D}]$ [2] (D nam predstavlja dimenzijo); izračun naključne komponente posameznika v mejah $[\boldsymbol{B}_{\min}, \boldsymbol{B}_{\max}]$ vidimo v enačbi (2). Tukaj $rand(0,1)$ označuje naključno spremenljivko v prostoru $\mathbb{R}$ med 0 in 1.

$$\boldsymbol{P}_0 := \{\boldsymbol{x}_{1,0}, \dots, \boldsymbol{x}_{\mu^0,0}\} \in \boldsymbol{I}^{\mu^0} \tag{1}$$

$$\boldsymbol{x}_{i,j,G} := B_{\min_j} + \left(B_{\max_j} - B_{\max_j}\right) \times rand(0,1) \tag{2}$$

b) *Ocenitev*: za ocenitev vseh rešitev v populaciji uporabimo funkcijo uspešnosti in vsem rešitvam dodelimo oceno. Tako s preslikavo $f : \boldsymbol{x}_{i,G} \to \mathbb{R}$ ocenimo uspešnost rešitve $\boldsymbol{x}_{i,G}$, npr. imenovano *evaluateGVGAI* za primer ocenitvene funkcije GVG-AI.

$$f(\boldsymbol{x}_{i,G}) := evaluateGVGAI(\boldsymbol{x}_{i,G}) \tag{3}$$

c) *Selekcija staršev*: pri DE za selekcijo uporabimo vse posameznike v populaciji in izmed posameznikov $\boldsymbol{x}_{i,G}, \forall i \in \{1,2,\dots,NP\}$, kjer je NP velikost populacije v trenutni generaciji G [15], $NP = \mu^G$, izberemo tri naključne posameznike $\boldsymbol{x}_{r_1,G}, \boldsymbol{x}_{r_2,G}, \boldsymbol{x}_{r_3,G}$, za katere velja $r_1 != r_2 != r_3 != i$, ter po enačbi (4) izračunamo novega posameznika $\boldsymbol{v}_{i,G+1}$ [2], F predstavlja skalirni faktor [15].

$$\boldsymbol{v}_{i,G+1} := \boldsymbol{x}_{r_1,G} + F \times (\boldsymbol{x}_{r_2,G} - \boldsymbol{x}_{r_3,G}) \tag{4}$$

d) *Reprodukcija*: potomce v generaciji ustvarimo s pomočjo križanja staršev. Pri DE najprej izračunamo spremenljivko $j_{rand} = randint(1, D)$, nato v zanki od $1, 2, \dots, D$ po enačbi (2.5), kjer CR predstavlja stopnjo križanja [15], $randint(1, D)$ pa predstavlja $\mathbb{N}$ vrednost v obsegu od 1 do D, izračunamo novega posameznika $u_{i,j,G+1}$ [16]. Tako vektor $\boldsymbol{v}_{i,G}$ kot $\boldsymbol{x}_{i,G}$ sta sestavljena iz komponent $\boldsymbol{v}_{i,G} = [v_{i,1,G}, v_{i,2,G}, \dots, v_{i,D,G}]$ oziroma $\boldsymbol{x}_{i,G} = [x_{i,1,G}, x_{i,2,G}, \dots, x_{i,D,G}]$. Pri križanju lahko uporabimo dvomestno križanje, večmestno križanje ali uniformno križanje [17]; slednje uporabimo pri DE kot:

$$u_{i,j,G+1} := \begin{cases} v_{i,j,G+1} & \text{če } rand(0,1) \le CR \text{ ali } j = j_{rand} \\ x_{i,j,G+1} & \text{sicer} \end{cases} \tag{5}$$

e) *Variacija*: mutacija se izvede nad nekaterimi starši in/ali potomci.

f) *Zamenjava*: v tem koraku izberemo, kateri izmed posameznikov (izbiramo tako med starši kot med potomci) je dovolj dober, da nadaljuje svoje poslanstvo v prihodnji generaciji. Priljubljene metode zamenjave so generacijska (angl. *generational*), kjer starši umrejo in jih nasledijo potomci, stabilna (angl. *steady state*), to uporablja tudi DE, kjer potomci nadomestijo starše le, če je njihova ocena boljša kot ocena staršev, in elitizem, ki je sicer podoben generacijski zamenjavi, a najboljših *x* % staršev preživi.

g) *Prekinitev*: vprašamo se, če smo končali. Odločitev temelji na tem, koliko generacij oziroma ocenitev je preteklo, najvišji oceni pridobljeni s strani rešitev in/ali kakšnem drugem zaključnem pogoju.

h) *Če še nismo končali, pojdimo na korak (b)*.

Vsaka iteracija glavne zanke (vsakič, ko pridemo do koraka *b*) se imenuje generacija, saj je terminologija povzeta iz teorije evolucije [2]. Končno število ocenitev je tako sorazmerna velikosti populacije krat številu generacij [8].

DE je novejši evolucijski algoritem, ki se v globalni optimizaciji uporablja za realno kodiranje numeričnih funkcij [15]. DE je zaradi svoje preprostosti, zmogljivosti in stabilnosti postal zelo priljubljen [18].

DE ima majhno število parametrov, vendar zaradi svoje stabilnosti in prilagajanja problemu, daje boljše rezultate od ostalih evolucijskih algoritmov [15].

Pri DE po začetni inicializaciji populacije, ki nam jo prikazuje enačba (1) sledita mutacija in križanje po enačbah (4) in (5). Po križanju s pomočjo enačbe (6) preverimo, ali se posameznik nahaja v območju $[\boldsymbol{B}_{\min}, \boldsymbol{B}_{\max}]$.

$$\boldsymbol{u}_{i,G+1} := \begin{cases} \boldsymbol{B}_{\min} + (\boldsymbol{B}_{\min} - \boldsymbol{u}_{i,G+1}) & \text{če } \boldsymbol{u}_{i,G+1} < \boldsymbol{B}_{\min} \\ \boldsymbol{B}_{\max} - (\boldsymbol{u}_{i,G+1} - \boldsymbol{B}_{\max}) & \text{če } \boldsymbol{u}_{i,G+1} > \boldsymbol{B}_{\max} \\ \boldsymbol{u}_{i,G+1} & \text{sicer} \end{cases} \tag{6}$$

Nazadnje sledi še izbira med staršem $\boldsymbol{x}_{i,G}$ in dobljenim vektorjem $\boldsymbol{u}_{i,G+1}$. Novega posameznika (za maksimizacijski problem) $\boldsymbol{x}_{i,G+1}$ tako dobimo po enačbi (7) [16].

$$\boldsymbol{x}_{i,G+1} := \begin{cases} \boldsymbol{u}_{i,G+1} & \text{če } f(\boldsymbol{u}_{i,G+1}) > f(\boldsymbol{x}_{i,G}) \\ \boldsymbol{x}_{i,G} & \text{sicer} \end{cases} \tag{7}$$

Med optimizacijskim procesom obdrži algoritem DE točno določene krmilne parametre. Težava se pojavi,

ker najbolj učinkovitih vrednosti pri algoritmu ne poznamo in so še vedno obstoječa vprašanja mnogih raziskovalcev.

Potrjena je bila tudi potreba po spreminjanju parametrov med procesom optimiziranja [16]. Tako je pri algoritmu jDE, ki razširja algoritem DE [18], vsak i-ti posameznik razširjen s samoprilagodljivima parametroma F in CR, ki ju naključimo s stopnjo verjetnosti τ_F oziroma τ_{CR}[16]. Nova parametra $F := F_{i,G+1}$ in $CR := CR_{i,G+1}$ se izračunata po enačbah (8) in (9) [16], kjer sta F_l in F_u spodnja in zgornja meja za F.

$$F_{i,G+1} := \begin{cases} F_l + rand_1 \times F_u & \text{če } rand_2 < \tau_F \\ F_{i,G} & \text{sicer} \end{cases} \quad (8)$$

$$CR_{i,G+1} := \begin{cases} rand_3 & \text{če } rand_4 < \tau_{CR} \\ CR_{i,G} & \text{sicer} \end{cases} \quad (9)$$

Ta metoda nastavljanja parametrov s poskušanjem (*angl. Trial-and-error method*) se uporablja za prilagajanje krmilnih parametrov in te nastavlja skozi posamezni zagon optimizacije [16][18].

2.2 Ogrodje tekmovanja inteligentnih agentov za igranje videoiger

Ker večino tekmovanj izziva inteligentne agente k igranju ene same igre, so se ta tekmovanja izkazala kot prevelika omejitev na določeno področje (domeno) [19].

V ta namen, kot protiutež dosedanjim tekmovanjem, je bilo ustvarjeno *mednarodno tekmovanje inteligentnih agentov za igranje videoiger*, ki za namene tekmovanja uporablja lastno ogrodje.

Agenti so tako izpostavljeni več, do uradnega dela tekmovanja še ne videnim igram, kar pomeni, da se morajo pripraviti na najrazličnejše možne scenarije. Vse različice tekmovanja GVG-AI se držijo enakega reda iger. Videoigre so razdeljene v različne sete (10 iger v setu, vsaka igra pa ima 5 različnih stopenj). Javni nabori, ki so vključeni v ogrodje so namenjeni tekmovalcem za preizkušanje izdelanega agenta.

Ogrodje GVG-AI lahko uporabimo tudi za postopno ustvarjanje vsebine (*angl. Procedural Content Generation*) [11]; ustvarimo lahko poljubne stopnje ali pravila. V obeh primerih ogrodje ponuja vnaprejšnje modeliranje (angl. *Forward Modeling*) [11], kar omogoča agentu, da stanje igre kopira in jo s pomočjo danih potez razvije, da bi s tem po možnosti dosegel naslednje stanje igre.

Ocena uspešnosti avtomatskega igranja se v GVG-AI izračuna po desetih igrah. Dejavniki, ki vplivajo na vrednotenje igre so: število zmag, skupno število točk (rezultat) in porabljen čas. Po vsakem odigranem krogu se tekmovalci ovrednotijo po sistemu točkovanja Formule 1 [19] in sicer po vrsti od prvega do desetega mesta s točkami 25, 18, 15, 12, 10, 8, 6, 4, 2 ter 1. Ostali prejmejo 0 točk [20]. Končen rezultat oziroma ovrednotenje prinese seštevek vseh pridobljenih točk. V primeru neodločenih izidov zmaga agent, ki ima več prvih pozicij, če pa sta tudi v tem segmentu agenta izenačena se štejejo druge, tretje, četrte pozicije in tako naprej [20].

V okviru ogrodja GVG-AI je en od že objavljenih agentnih algoritmov, EA sprotnega razvijanja (angl. *Rolling Horizion EA*, v nadaljevanju RHEA) [21], ki za iskanje uporablja preprost EA. Pri tem je EA z ogrodjem GVGA-AI povezan tako, da ocenitveno funkcijo optimizacijskega algoritma kličemo s pomočjo simulatorja v GVG-AI, kjer agent operira. Ocenitvena funkcija (za klic funkcije simulatorja dodamo vektorju $\boldsymbol{x}_{i,G}$ še trenutno stanje v igri in hevristiko) v EA nam za določen vektor $\boldsymbol{x}_{i,G}$ vrne dosežen rezultat iz simulatorja, pomnožen z $0{,}9^D$ (D je število dimenzij) [22]. V primeru, da je igre konec in je igralec izgubil, ocenitvena funkcija vrne vrednost $-1000 \times 0{,}9^D$ kar pomeni, da smo od zmage zelo oddaljeni. V primeru, da je igre konec in je igralec zmagal, ocenitvena funkcija vrne $1000 \times 0{,}9^D$ kar pomeni, da smo blizu zmagi. Če nismo dosegli nobenega izmed kriterijev, funkcija vrne $trenutni\ rezultat \times 0{,}9^D$. Optimizacijski problem v tej funkciji je predstavljen kot maksimizacija ocene. Ocenitvenih funkcij je posledično toliko, kolikor je možnih scenarijev – pod scenarije štejemo igranje različnih iger, vsaka igra pa ima več stopenj. Pri vsakem scenariju se namreč razlikujejo vhodi in stanje okolja, ki ju kot vektor $\boldsymbol{x}$ vključujemo v ocenitveno funkcijo.

$$f(\boldsymbol{x}_{i,G}) := \begin{cases} 1000 \times 0{,}9^D & \text{če } gameOver \text{ in } playerWins \\ -1000 \times 0{,}9^D & \text{če } gameOver \text{ in } playerLoses \\ rezultat \times 0{,}9^D & \text{sicer} \end{cases} \quad (10)$$

3 Snovanje agenta in implementacija

Snovanje agenta smo pričeli s pregledom obstoječih algoritmov za splošno igranje videoiger. Ugotovili smo, da so EA druga največja zastopana skupina algoritmov na tekmovanju. Odločili smo se za nadomestitev in delno nadgradnjo kode RHEA, v kateri smo dele RHEA zamenjali z algoritmom DE s prilagodljivimi krmilnimi parametri v tistem delu, ki implementira operatorje EA. Agenta smo poimenovali *VorosZamuda* [12]. Osnovan je na agentu *vzorčiRHEA* (angl. *sampleRHEA*), ki kot optimizacijski algoritem za GVG-AI uporablja algoritem RHEA. Ohranili smo dve obstoječi funkciji iz izvorne kode agenta *vzorčiRHEA, evaluate* in *get_best_action*, a spremenili vhodne parametre tako, da so slednji ustrezali našemu agentu. Nato smo implementirali pet novih funkcij, *initialize*, *mutate*, *crossover*, *correction* in *selection*, kamor smo vnesli kodo za računanje z DE iz poglavja 2.1. Vse skupaj smo uporabili v javni metodi *act*, ki je potrebna za zagon agenta. Podrobnejši psevdokodo metode *act* za algoritem *VorosZamuda* je dostopen v [12] na strani 34.

4 Rezultati

Odločili smo se, da lahko nekatere spremenljive parametre, ki jih imata oba agenta – velikost populacije in velikost dimenzije, pustimo na enakih vrednostih, za lažje kasnejše primerjanje rezultatov. Dodati smo morali tudi dve nastavljivi spremenljivki, in sicer skalirni

faktor ter verjetnost križanja, potrebni za izvedbo DE algoritma in dve spremenljivki $B_{\min}$, $B_{\max}$, ki smo ju nastavili v inicializaciji. Razen omenjenih spremenljivk smo ostale spremenljivke pustili nedotaknjene, torej enake kot pri agentu *vzorčiRHEA*.

Za večjo pristnost rezultatov smo se odločili, da oba agenta, tako *VorosZamuda*, kot *vzorčiRHEA*, testiramo na uradni spletni strani GVG-AI, namesto da bi zagone delali sami s pomočjo ogrodja. Tako smo na njihovem strežniku po nekaj urah izvajanja tega bremena hkrati dobili tudi ustrezno rangiranje obeh agentov med ostale agente, ki so na obstoječi lestvici uvrščeni od vključno 1. maja 2016 [23]. Na tem testnem naboru je agent *VorosZamuda* tako dosegel drugo mesto, kar lahko vidimo v tabeli 1. Bil je najboljše uvrščeni slovenski predstavnik. Agent *VorosZamuda* se je uvrstil takoj za agentom *combination*, dosegel pa je vsega 3 točke manj od prvouvrščenega agenta (*combination* je dosegel 114, *VorosZamuda* pa 111 točk). Za agenta *VorosZamuda* se je uvrstilo kar dvajset preostalih agentov.

Tabela 1. Tabela uvrstitve agentov.

Uvrstitev	Uporabniško ime	Država	Točke	Št. zmag (diskvalif.)
1	combination	Nemčija	114	23/50 (0)
2	**VorosZamuda**	Slovenija	111	17/50 (0)
3	fanatax	Nemčija	107	16/50 (0)
4	JACAM	Nemčija	79	12/50 (0)
5	asd592	Nemčija	71	11/50 (0)
6	matze1234	Nemčija	69	11/50 (0)
7	Jaybot	Nemčija	66	16/50 (0)
8	**vzorčiRHEA**	Slovenija	59	15/50 (0)
9	crazypet	Tajska	54	9/50 (0)
10	Cyclus	Nemčija	45	11/50 (0)
11	fraBOT2	Nemčija	44	13/50 (0)
12	JamieHutchison	VB	43	4/50 (9)
13	ToVo1	Slovenija	43	8/50 (0)
14	sampleOLMCTS	VB	42	11/50 (0)
15	sampleRS	VB	34	7/50 (0)
16	YOLOBOT	Nemčija	33	8/50 (0)
17	omnipotent	Nemčija	23	6/50 (0)
18	Damorin	VB	22	6/50 (0)
19	scott877	VB	22	4/50 (5)
20	sampleRHEA	VB	16	5/50 (0)
21	random	VB	14	4/50 (0)
22	treesakul	Tajska	12	3/50 (0)

Nad rangi agentov *VorosZamuda* in *vzorčiRHEA*, iz scenarija prikazanega v tabeli 1 smo izvedli še Wilcoxon Ranked-Sum Test [24]. Agent *VorosZamuda* je pri zagonih dosegel mesta 17, 4, 4, 1, 2, 13, 13, 3, 8 in 1, agent *vzorčiRHEA* mesta 3, 2, 18, 7, 8, 20, 13, 8, 17 in 4. Po izračunu smo dobili vrednost $p = 0{,}0858$. Tako smo ugotovili, da sta agenta z 91 % verjetnostjo signifikantno različna in pokazali, da je nov agent izboljšan.

5 Zaključek

V tem prispevku smo predstavili novega agenta za GVG-AI, ki deluje s pomočjo DE. Agenta smo ocenili in primerjali z drugimi in izkazalo se je, da z verjetnostjo 91 % algoritem deluje signifikantno boljše od osnovnega algoritma *vzorčiRHEA* ter zasede drugo mesto na rangu vseh preostalih algoritmov za planiranje v GVG-AI na podlagi uradnih meritev na strežniku za primerjalne teste GVG-AI.

V nadaljevanju bi algoritem lahko dodatno uglasili in razširili z izboljšanimi mehanizmi za optimizacijo in ga razširili na še več scenarijev v GVG-AI. Ta pristop AL bi lahko uporabili sicer tudi v katerem drugem okolju ML iz realnega sveta na primerih avtomobilske industrije ali globokomorske robotike.

Zahvala

Ta prispevek je nastal s podporo raziskovalnega programa P2-0041 pri ARRS ter CA15140 pri COST.

Literatura

[1] Y. Zheng, S. Yang in H. Cheng, „An application framework of digital twin and its case study,“ *Journal of Ambient Intelligence and Humanized Computing,* izv. 10, št. 3, str. 1141-1153, 2019.

[2] Zamuda, Aleš. "Operacijske raziskave logističnih, transportnih in ekonomskih sistemov: zbrano gradivo." (2020).

[3] J. Madge in T. Oplatek, „Game AI : Agent for GVGAI,“ 2018. http://www.tomasoplatek.com/ index.php/2018/07/09/game-ai/. [Dostop: 9. maj 2019].

[4] M. G. Bellemare, Y. Naddaf, J. Veness in M. Bowling, „The arcade learning environment: An evaluation platform for general agents,“ *Journal of Artificial Intelligence Research,* izv. 47, str. 253-279, 2013.

[5] D. Perez, „The General to Video Game AI Competition,“ [Elektronski]. Available: http://www.gvgai.net/. [Dostop 10. maj 2019].

[6] N. Guid in D. Strnad, Umetna inteligenca, Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, 2007.

[7] S. J. Russell in P. Norvig, Artificial intelligence: a modern approach, Malaysia: Pearson Education Limited, 2016.

[8] G. N. Yannakakis in J. Togelius, Artificial Intelligence and Games, Springer, 2018.

[9] A. E. Eiben in J. E. Smith, Introduction to evolutionary computing, izv. 53, Berlin: Springer, 2003.

[10] B. Filipič, „Optimizacija in evolucijsko računanje,“ https://dis.ijs.si/filipic/courses/is/50_Optimizacija_in_evolucijsko_racunanje.pdf. [Dostop: 1. april 2019].

[11] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius in S. M. Lucas, „General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms,“ [Dostop: 18. marec 2019].

[12] Vörös, Matjaž. Evolucijski algoritmi za učenje agenta umetne inteligence pri igranju splošnih videoiger. Diss. Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, 2019.

[13] S. Rainer in K. Price, „Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces,“ *Journal of global optimization,* izv. 11, št. 4, str. 341-359, 1997.

[14] C. A. Coello, G. B. Lamont in D. A. Van Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition, izv. 5, New York: Springer, 2007.

[15] A. Zamuda, „Magistrsko delo: Samoprilagajanje krmilnih parametrov pri algoritmu diferencialne evolucije za večkriterijsko optimizacijo,“ Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, 2008.

[16] A. Zamuda in J. Brest, „Self-adaptive control parameters' randomization frequency and propagations in differential evolution,“ Swarm and Evolutionary Computation, izv. 25, str. 72-99, 2015.

[17] M. Mernik, M. Črepinšek in V. Žumer, Evolucijski algoritmi, Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za računalništvo, 2003.

[18] J. Brest, S. Greiner, B. Bošković, M. Mernik in V. Žumer, „Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems,“ IEEE Transactions on evolutionary computation, izv. 10, št. 6, str. 646-657, 2006.

[19] J. Liu, „GVGAI Single-Player Learning Competition at IEEE CIG17,“ 7 september 2017. [Elektronski]. https://tinyurl.com/y4hyvayh. [Dostop: 18. marec 2019].

[20] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim in T. Thompson, „The 2014 General Video Game Playing Competition,“ v *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.

[21] D. Perez, S. Samothrakis, S. Lucas in P. Rohlfshagen, „Rolling horizon evolution versus tree search for navigation in single-player real-time games,“ v *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013.

[22] D. Perez, „Sample Controllers,“ GVGAI.net, http://www.gvgai.net/sampleControllers.php. [Dostop: 9. maj 2019].

[23] D. Perez, „Training Set Games 11 (Continuous Physics),“ http://www.gvgai.net/gvg_rankings.php?rg=11. [Dostop: 27. maj 2019].

[24] H. B. Mann in D. R. Whitney, „On a test of whether one of two random variables is stochastically larger than the other,“ *The annals of mathematical statistics,* izv. 18, št. 1, str. 50-60, 1947.