GenIATraP: Splošna platforma za učenje inteligentnih agentov

Marko Šmid, Matej Črepinšek, Miha Ravber

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko
Koroška cesta 46, 2000 Maribor, Slovenija.
E-pošta: {marko.smid2, matej.crepinsek, miha.ravber}@um.si

GenIATraP: General Intelligent Agent Training Platform

This paper introduces GenIATraP, a novel, comprehensive training platform designed for the development and training of intelligent agents within both single and multi-agent systems. Following a review of existing platforms and frameworks, the article delineates the architecture of GenIATraP, emphasizing its modular design and scalability. The platform's applicability across a broad spectrum of problem domains is outlined, underscoring its versatility and adaptability. A test case demonstrates the platform's efficacy in addressing the collector problem domain. The paper concludes with an overview of anticipated developments for GenIATraP, highlighting ongoing efforts to enhance its capabilities and broaden its scope. This contribution aims to advance artificial intelligence and machine learning by offering a robust, user-friendly platform for training intelligent agents across diverse applications.

1 Uvod

Optimizacija agentnih sistemov v fizičnem svetu zahteva veliko virov in časa. Poleg tega predstavlja veliko nevarnost nesreč, ki se lahko zgodijo kot posledica neustreznega obnašanja avtonomnih agentov. Zaradi tega raziskovalci uporabljajo simulacijska okolja, ki posnemajo fizični svet in tako omogočajo cenejšo, hitrejšo in varnejšo optimizacijo takšnih sistemov.

Simulacijska okolja je mogoče pripraviti na več različnih načinov, eden izmed njih je tudi z uporabo igralnih pogonov [1]. Igralni pogoni predstavljajo odlično izhodišče za pripravo takšnih simulacij, saj imajo številne funkcionalnosti, kot sta na primer simulacija fizike in vizualizacija, že vključene, hkrati pa omogočajo hitro prototipiranje rešitev.

V obstoječi literaturi obstaja kopica ogrodij za optimizacijo agentnih sistemov. V tem članku se bomo osredotočili samo na ogrodja, ki temeljijo na igralnih pogonih in vključujejo tehnike strojnega učenja. Med njimi sta najpogosteje uporabljena igralna pogona Unity in Unreal Engine. Za igralni pogon Unity je najbolj razširjena zbirka orodij ML-Agents (Machine Learning Agents) [2], ki z uporabo iger in simulacij znotraj igralnega pogona omogoča razvoj inteligentnih agentov. Pri tem ponuja različne tehnike umetne inteligence, kot so okrepitveno učenje (angl. Reinforcement Learning), učenje s posnemanjem (angl. Imitation Learning) in nevroevolucijo. Ponuja tudi vnaprej pripravljene testne primere, ki zajemajo eno- in večagentne sisteme. Song in drugi [3] predstavijo zbirko orodij za optimizacijo več-agentnih sistemov, ki temelji na zbirki orodij ML-Agents. Platforma vključuje 35 iger, ki predstavljajo več-agentna okolja ter nabor algoritmov za optimizacijo več-agentnih sistemov. Johansen in drugi [4] predstavijo ogrodje UnityVGDL (Unity Video Game Description Language), ki temelji na ogrodju GVGAI (General Video Game AI). Ogrodje omogoča definiranje iger z uporabo opisnega jezika VGDL (Video Game Description Language). V ogrodje je integrirana predhodno omenjena zbirka orodij ML-Agents, s čimer ogrodje omogoča razvoj inteligentnih agentov. Ciprian Paduraru in Mirna Paduraru [5] predstavita ogrodje, ki omogoča razvoj vedenjskih dreves (angl. Behavior Trees) s pomočjo genetskega algoritma, za različne težavnosti. Vedenjska drevesa so hierarhične strukture, ki se uporabljajo za modeliranje procesov odločanja, pri čemer vsako vozlišče predstavlja določeno akcijo, pogoj ali kontrolni tok. S kombiniranjem teh vozlišč omogočajo kompleksna obnašanja, visoko stopnjo modularnosti in razširljivosti. [6].

Podobno kot igralni pogon Unity, tudi Unreal Engine ponuja vtičnik za razvoj inteligentnih agentov, imenovan Learning Agents [7]. Vtičnik omogoča uporabo okrepitvenega učenja in učenja s posnemanjem za razvoj inteligentnih agentov. Partlan in drugi [8] predstavijo ogrodje EvolvoBehavior, ki je implementirano v igralnem pogonu Unreal Engine in omogoča razvoj vedenjskih dreves s tehniko genetskega programiranja. Osnovna ideja ogrodja se osredotoča na soustvarjalni proces, v katerem človeški dejavnik sodeluje pri iskanju rešitev. Seznam ostalih obstoječih platform je mogoče najti v članku avtorja Wrona in drugih [9].

Slabost obstoječih ogrodij je slaba razširljivost, saj nudijo podporo samo specifičnim tehnikam, kar otežuje uporabo drugih tehnik. Obstoječa ogrodja prav tako ponujajo omejeno paralelizacijo. V ta namen v prispevku predstavimo platformo za optimizacijo agentnih sistemov, ki je zasnovana modularno in omogoča visoko stopnjo paralelizacije. Platforma trenutno podpira igralni pogon Unity in ogrodje za strojno učenje EARS (angl. Evolutionary Algorithm Rating System) [10]. Obnašanje agentov

znotraj platforme je definirano z uporabo vedenjskih dreves.

Prispevek sestavlja pet poglavij. V naslednjem poglavju predstavimo arhitekturo predlagane platforme. V tretjem poglavju predstavimo obstoječa problemska področja. V četrtem poglavju sledi predstavitev eksperimenta. V zadnjem poglavju so podani zaključki.

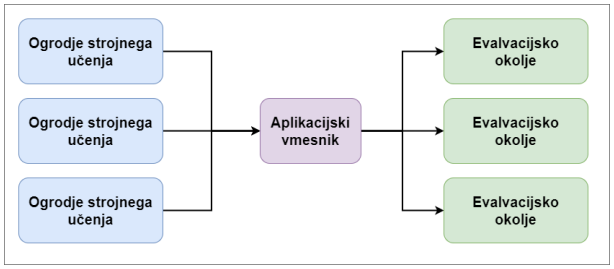
2 Arhitektura platforme GenIATraP

V poglavju najprej podamo opis konceptualne zasnove platforme, nato nadaljujemo s predstavitvijo njene trenutne arhitekture in optimizacijskega procesa.

2.1 Konceptualna zasnova

Konceptualno zasnovo platforme sestavljajo tri osnovne komponente (prikazane na sliki 1): ogrodje za strojno učenje (angl. Machine Learning Framework), evalvacijsko okolje (angl. Evaluation Environment) in spletni aplikacijski vmesnik (angl. Web API). Ogrodje za strojno učenje vsebuje nabor algoritmov, prilagojenih za optimizacijo eno- in večagentnih sistemov. Evalvacijsko okolje vsebuje nabor problemskih področij, ki predstavljajo kopije problemov iz resničnega življenja. Spletni aplikacijski vmesnik je namenjen povezovanju in komunikaciji predhodno omenjenih komponent. Njegova ciljna naloga je pretvoriti posamezne rešitve iz ogrodja za strojno učenje v obliko, ki jo lahko interpretira in evalvira evalvacijsko okolje.

Neodvisnost posameznih komponent zagotavlja, da lahko preprosto povežemo več različnih komponent, ne da bi pri tem vplivali na delovanje drugih. Ta modularni pristop pri dodajanju ali zamenjavi komponente zahteva samo prilagoditev spletnega aplikacijskega vmesnika, da bo ustrezal standardom nove komponente, če so ti različni od obstoječih.



Slika 1: Konceptualna zasnova platforme.

2.2 Arhitektura in delovanje

Kot že omenjeno, smo kot ogrodje za strojno učenje izbrali EARS. EARS je odprtokodno ogrodje, implementirano v Javi, in je namenjeno rangiranju, razvoju in eksperimentiranju z eno- ali večkriterijskimi evolucijskimi algoritmi. Ogrodje ponuja širok nabor že implementiranih evolucijskih algoritmov in problemov, nad katerimi jih je mogoče med seboj primerjati. Za izbiro tega ogrodja smo se odločili na podlagi predhodnih izkušenj z njim ter zmožnosti enostavne prilagoditve našim potrebam.

Za evalvacijsko okolje smo uporabili igralni pogon Unity, ki velja za eno izmed standardnih orodij za razvoj računalniških iger. Za izbiro tega igralnega pogona smo se odločili zato, ker omogoča preprosto implementacijo problemov v obliki iger ter vključuje širok spekter že pripravljenih skript in modelov, ki poenostavijo razvojni proces. Poleg tega vsebuje fizikalni pogon, ki omogoča realistično simuliranje gibanja in trkov, grafični urejevalnik, ki olajša prototipiranje, ter nudi podporo za vizualizacijo v 2D in 3D.

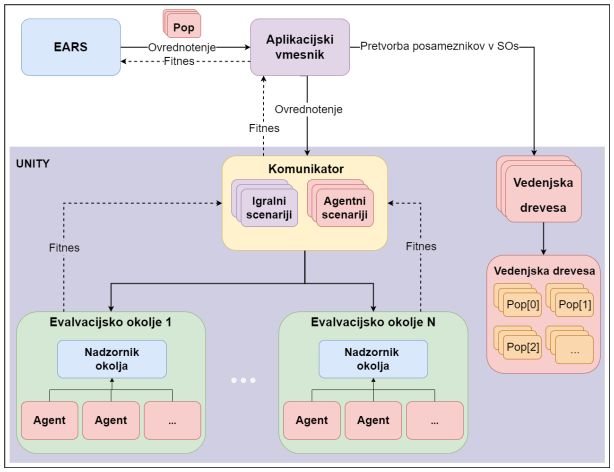
Za tehniko strojnega učenja smo izbrali GP (genetsko programiranje) [11], ki je vrsta evolucijskih algoritmov, kjer so posamezniki predstavljeni kot programi, ki so nato razviti s pomočjo GA (genetskega algoritma).

Kot mehanizem za nadzorovanje agentov smo izbrali vedenjska drevesa, ki v primerjavi z njihovim predhodnikom, končnim avtomatom stanj (angl. Finite State Machine), nudijo številne prednosti [12] ter jih je mogoče preprosto prikazati kot posameznike v genetskem programiranju.

Arhitektura in proces optimizacije z uporabo naše platforme sta prikazana na sliki 2. Optimizacijski proces se začne v EARS-u, kjer se ustvari začetna populacija in začne evolucijski proces. V fazi evalvacije se naredi zahteva POST spletnemu aplikacijskemu vmesniku, ki se mu posreduje trenutna populacija za evalvacijo v formatu JSON (angl. JavaScript Object Notation). Spletni aplikacijski vmesnik posameznike iz populacije pretvori v skriptne objekte (angl. Scriptable Objects) in jih shrani igralnemu pogonu Unity na dostopno mesto. Skriptni objekti so datoteke, ki vsebujejo podatke o enem posamezniku, ki se skozi evalvacijo ne spreminjajo. Po uspešnem shranjevanju se pošlje zahteva GET, komunikatorju, ki predstavlja HTTP (angl. Hypertext Transfer Protocol) strežnik, ki se izvaja znotraj igralnega pogona. Komunikator prebere skriptne datoteke in začne proces evalvacije posameznikov. Odvisno od nastavitev, komunikator za vsakega posameznika oziroma skupino posameznikov naloži ustrezne scene, kjer so posamezniki evalvirani. Vsaka scena je sestavljena iz dveh podscen: scena, ki vsebuje okolje, v katero so objekti postavljeni, in scene, ki vsebuje krmilnik, čigar namen je nadzorovanje in spremljanje trenutne evalvacije. Ko je pri evalvaciji dosežen zaustavitveni pogoj (število korakov, čas izvajanja ali drug zaustavitveni pogoj), krmilnik vrne komunikatorju končno stanje. Vse scene, povezane s trenutno evalvacijo, se uničijo. S tem se sprostijo viri in začne se evalvacija novega posameznika. Po zaključeni evalvaciji vseh posameznikov komunikator posreduje končne rezultate nazaj spletnemu aplikacijskemu vmesniku, ki jih nato posreduje EARS-u, da lahko nadaljuje evolucijski proces. Celoten postopek se ponavlja, dokler v EARS-u niso doseženi zaustavitveni pogoji (število evalvacij, število generacij, čas izvajanja).

V opisanem procesu je evalvacija najzahtevnejša operacija, zato ogrodje zagotavlja paralelizacijo na več ravneh. Na sliki 2 je predstavljena arhitektura, ki vsebuje le eno instanco igralnega pogona Unity, vendar lahko uvedemo tolikšno število instanc, kolikor jih zmore podpirati fizični sistem, na katerem se izvaja platforma. V tem primeru spletni aplikacijski vmesnik pred pošiljanjem zahteve za evalvacijo preveri število instanc okolja Unity, ki

se trenutno izvajajo, in vsaki instanci pošlje le del populacije za evalvacijo (npr. če imamo populacijo velikosti 100 in 5 instanc Unity okolja, se bo vsaki instanci poslalo samo 20 posameznikov). Druga stopnja paralelizacije omogoča, da se znotraj igralnega pogona Unity, vzporedno izvaja več evalvacijskih okolij. Agenti so v simulacijskem svetu na enakem položaju, vendar med seboj nimajo interakcij in so neodvisni. Zaradi fizikalnih omejitev igralnega pogona je največje število dovoljenih vzporednih evalvacijskih okolij 20. Privzeto igralni pogon Unity omogoča tudi 100-kratno pospešitev simulacije z zmanjšanjem časovnega koraka med dvema stanjema, kar pa drastično poveča porabo fizičnih virov računalnika.



Slika 2: Arhitektura platforme z eno instanco okolja Unity.

3 Problemska področja

V platformo smo dodali problemska področja, katerih namen je pokazati, da je platformo mogoče uporabiti na raznolikih problemih in doseči kompleksna obnašanja. Vključili smo 4 igre, in sicer: Bomberman, Robostrike, Soccer in Collector. V podpoglavjih sledijo kratki opisi problemov, slike posameznih evalvacijskih okolij pa so prikazane na sliki 3.

3.1 Problemsko področje Bomberman

Bomberman predstavlja kopijo videoigre, ki temelji na večagentnem sistemu, v katerem lahko agenti med seboj sodelujejo ali tekmujejo. Igranje vključuje strateško postavljanje in izmikanje bombam, ki služijo za uničevanje ovir ter napadanje nasprotnikov. Agenti lahko pridobijo različne dodatke, ki izboljšajo njihove sposobnosti. Cilj igre je čim hitreje premagati nasprotnike in preživeti čim dlje.

3.2 Problemsko področje Robostrike

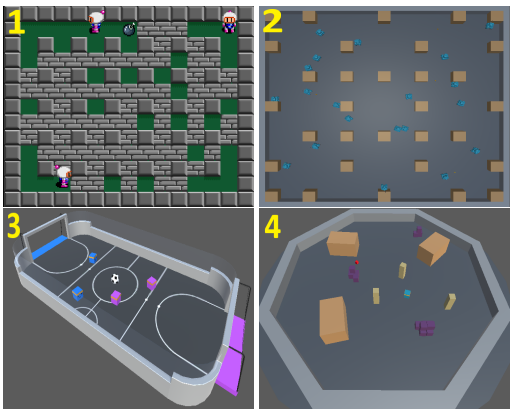
Robostrike je igra, v kateri več agentov sodeluje ali tekmuje, kar jo uvršča med več-agentne sisteme. Vsak agent je opredeljen kot tank, ki ima zmožnost premikanja po igralnem polju ter izstreljevanja raket. Cilj igre je preživeti čim dlje, hkrati pa z izstreljevanjem raket zadeti čim več nasprotnikov.

3.3 Problemsko področje Soccer

Igra Soccer je kopija sistema, predstavljenega v [2]. Pri tej igri se dve skupini (dvojici) agentov pomerita med seboj, kar uvršča igro med več-agentne sisteme. V igri se agenti premikajo po igralnem polju, njihova hitrost ob trku z žogo odraža silo, s katero jo bo agent zadel. Tekmovanje se zaključi po določenem časovnem obdobju, zmaga pa tista skupina, ki je uspela zadeti čim več golov nasprotnika, in jih hkrati prejela čim manj.

3.4 Problemsko področje Collector

Collector je igra, pri kateri se agent premika po površini in pobira tarče. Ko je tarča pobrana, se ta prestavi na drugo naključno lokacijo, nato pa se postopek ponovi. Postopek se ponavlja, dokler agentu ne zmanjka časa. Cilj igre je pobrati čim več tarč in pri tem porabiti čim manj energije. Če agent v določenem času ne najde tarče, ga kaznujemo, tarčo pa prestavimo na drugo mesto.



Slika 3: Problemska področja: (1) Bomberman, (2) Robostrike, (3) Soccer in (4) Collector.

Tabela 1: Parametri algoritma GP.

Parameter algoritma	Vrednost
pos_size	800
crossover_probability	0.7
mutation_probability	0.04
fitness_evaluations	100800
selection	turnirska (velikost 4)
crossover	eno-točkovno križanje
mutation	mutacija poddrevesa
ind_gen_method	naključno
min_tree_depth	5
max_tree_depth	12
pruning_operator	na podlagi globine
elitism_rate	0.05
functions	Sekvencer, Izbiralnik, Negator
terminals	Premik, Rotacija, Lidar Žarek

4 Eksperiment

Zaradi prostorskih omejitev smo učinkovitost platforme testirali le na problemskem področju Collector, opisanem v 3.4. Uporabili smo vrsto GP algoritma, ki vključuje elitizem. Nabor funkcij in terminalov ter uporabljeni parametri za GP algoritem so prikazani v tabeli 1. Uporabljeni terminali so nizkonivojski, kar pomeni, da ne vključujejo kompleksnega obnašanja, ampak le osnovne akcije, kot so premik naprej in nazaj, rotacija levo in desno ter zaznavanje objektov s pomočjo Lidar senzorja.

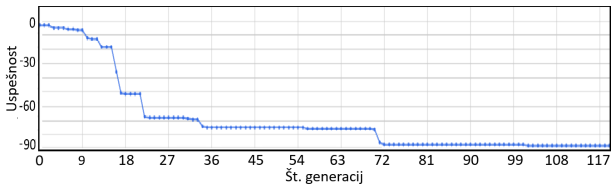
Za uspešno delovanje GP algoritma smo za problemsko področje Collector, definirali ocenitveno funkcijo, katere vrednosti so prikazane v tabeli 2. Problem smo zasnovali kot minimizacijski, kar pomeni, da dobre akcije agentu doprinesejo negativne vrednosti k njegovi uspešnosti, slabe pa pozitivne vrednosti.

Tabela 2: Vrednosti ocenitvene funkcije za področje Collector: negativna vrednost pomeni nagrado, pozitivna kazen.

Spremenljivka	Vrednost
AgentPickedTarget	-5
AgentExploredSector	-0.03
AgentReExploredSector	0.002
AgentSpottedTarget	-0.02
AgentsBtContainsMainObject	-0.1
AgentTouchedStaticObject	0.02
AgentBTNodePenalty	0.01

Trajanje ene scene smo omejili na 70 sekund. Da bi dobili agente s splošnim obnašanjem, smo vsakega agenta v eni evalvaciji testirali 10-krat na isti sceni, pri čemer so bile začetne pozicije agenta in tarče različne.

Slika 4 prikazuje uspešnost najboljšega posameznika skozi proces optimizacije. Na začetku so vidni veliki preskoki v uspešnosti, kar nakazuje na visoko stopnjo preiskovanja [8], skozi generacije pa ta postopoma upada, saj rešitve počasi konvergirajo.



Slika 4: Konvergenčni graf zagona optimizacijskega procesa za problemsko področje Collector.

Po optimizacijskem procesu, smo za končno evalvacijo izbrali najboljšega posameznika, kateremu smo izračunali uspešnost iskanja tarče z izvebo 100-ih zagonov. Pri vsakem zagonu sta agent in tarča postavljena na naključno mesto. Najboljša rešitev, pridobljena z GP algoritmom, je bila testirana pod temi pogoji in dosegla 99 % natančnost, kar pomeni, da agentu samo v enem primeru ni uspelo doseči tarče.

5 Zaključek

V prispevku smo predstavili GenIATraP, novo in celovito platformo za učenje inteligentnih agentov v eno in več-agentnih sistemih. Platforma vključuje vnaprej definirana problemska področja, ki omogočajo hitro prototipiranje novih problemov. S pomočjo platforme smo izvedli eksperiment za problemsko področje Collector in z uporabo genetskega programiranja pridobili rešitev, ki je dosegla 99 % natančnost. V prihodnje nameravamo obstoječo platformo nadgraditi z mehanizmi, ki omočajo učinkovitejše reševanje več-agentnih problemov ter vključiti nova ogrodja za strojno učenje in evalvacijska okolja.

Zahvala

Razvoj platforme GenIATraP je finančno podprla Javna agencija za znanstvenoraziskovalno in inovacijsko dejavnost Republike Slovenije (ARIS) v okviru programov P2-0041 in P2-0114.

Literatura

[1] Wydmuch, M., Kempka, M., & Jaśkowski, W. (2018). Vizdoom competitions: Playing doom from pixels. IEEE Transactions on Games, 11(3), 248-259.

[2] A. Juliani et al., ‘Unity: A General Platform for Intelligent Agents’. arXiv, May 06, 2020. doi: 10.48550/arXiv.1809.02627.

[3] Y. Song et al., ‘Arena: A General Evaluation Platform and Building Toolkit for Multi-Agent Intelligence’, Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 05, Art. no. 05, Apr. 2020, doi: 10.1609/aaai.v34i05.6216. https://dev.epicgames.com/community/learning/tutorials/8OWY/unreal-engine-learning-agents-introduction (Pridobljeno 16.6.2024).

[4] M. Johansen, M. Pichlmair, and S. Risi, ‘Video Game Description Language Environment for Unity Machine Learning Agents’, in 2019 IEEE Conference on Games (CoG), Aug. 2019, pp. 1–8. doi: 10.1109/CIG.2019.8848072.

[5] C. Paduraru and M. Paduraru, ‘Automatic difficulty management and testing in games using a framework based on behavior trees and genetic algorithms’, presented at the Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS, 2019, pp. 170–179. doi: 10.1109/ICECCS.2019.00026.

[6] M. Mateas and A. Stern, ”A behavior language for story-based believable agents,”in IEEE Intelligent Systems, vol. 17, no. 4, pp. 39-47, July-Aug. 2002, doi: 10.1109/MIS.2002.1024751.

[7] Learning Agents, https://dev.epicgames.com/community/learning/tutorials/8OWY/unreal-engine-learning-agents-introduction

[8] N. Partlan, L. Soto, J. Howe, S. Shrivastava, M. S. El-Nasr, and S. Marsella, ‘EvolvingBehavior: Towards Co-Creative Evolution of Behavior Trees for Game NPCs’. arXiv, Sep. 01, 2022. doi: 10.48550/arXiv.2209.01020.

[9] Z. Wrona et al., ‘Overview of Software Agent Platforms Available in 2023’, Information, vol. 14, no. 6, p. 348, Jun. 2023, doi: 10.3390/info14060348.

[10] N. Veček, M. Mernik, and M. Črepinšek, ‘A chess rating system for evolutionary algorithms: A new method for the comparison and ranking of evolutionary algorithms’, Information Sciences, vol. 277, pp. 656–679, Sep. 2014, doi: 10.1016/j.ins.2014.02.154.

[11] J. R. Koza, ‘Genetic programming as a means for programming computers by natural selection’, Stat Comput, vol. 4, no. 2, pp. 87–112, Jun. 1994, doi: 10.1007/BF00175355.

[12] M. Iovino, J. Förster, P. Falco, J. J. Chung, R. Siegwart and C. Smith, ”On the programming effort required to generate Behavior Trees and Finite State Machines for robotic applications,”2023 IEEE International Conference on Robotics and Automation (ICRA), London, United Kingdom, 2023, pp. 5807-5813, doi: 10.1109/ICRA48891.2023.10160972.