ELEKTRO IN RAČUNALNIŠKA ŠOLA VELENJE
TRG MLADOSTI 3, 3320 VELENJE
MLADI RAZISKOVALCI ZA RAZVOJ SAŠA REGIJE

RAZISKOVALNA NALOGA

SAMOSTOJNO UČENJE UMETNE INTELIGENCE PRI RAČUNALNIŠKI IGRI

Tematsko področje: RAČUNALNIŠTVO

AVTORJA:
Tim Rednjak, 3.TRA
Andraž Dimec, 3.TRA

MENTORJA:
Uroš Remenih inž., inf.
Samo Železnik inž., inf.

VELENJE, 2025

Raziskovalna naloga je bila opravljena na Elektro in računalniški šoli Velenje

Mentorja: Uroš Remenih, inž., inf., Samo Železnik, inž., inf.

Datum predstavitve: marec 2025

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

ŠD Elektro in računalniška šola Velenje

KG programiranje / računalništvo / umetna inteligenca

AV Dimec Andraž / Rednjak Tim

SA Remenih Uroš / Železnik Samo

KZ 3320 Velenje, SLO, Trg mladosti 3

ZA Elektro in računalniška šola Velenje

LI 2025

IN **SAMOSTOJNO UČENJE UMETNE INTELIGENCE PRI RAČUNALNIŠKI IGRI**

TD Raziskovalna naloga

OP VI, 25 str., 1 pregl., 0graf., 15 sl., 3 pril., 10 vir.

IJ SL

JI sl / en

AI Raziskovalna naloga Samostojno učenje umetne inteligence temelji na uporabi programa z uporabo programskega jezika Python. Program ustvari umetno inteligenco, ki se uči na svojih napakah, za katere je kaznovana.Za vse dobre odločitve, ki jih je naredila pa je nagrajena. Ta program sva vstavila tudi v različne računalniške igre kot je Trackmania. Z minimalnimi informacijami o tej igri je umetna inteligenca ugotovila kaj more delati in na kakšen način je to najlažje narejeno.

KEY WORDS DOCUMENTATION

ND SCV, Electro and Computer School Velenje

CX Programming / Computer Science / Artificial Intelligence

AU Dimec Andraž / Rednjak Tim

AA Remenih Uroš / Železnik Samo

PP 3320 Velenje, SLO, Trg mladosti 3

PB Elektro and Computer Science School Velenje

PY 2025

TI **INDEPENDENT LEARNING OF ARTIFICIAL INTELLIGENCE IN A COMPUTER GAME**

DT RESEARCH WORK

NO VI, 25 p., 1 tab., 0 graf., 15 fig., 3 ann., 10 ref.

LA SL

AL sl / en

AB The research paper on independent learning of artificial intelligence is based on the use of a program written in the Python programming language. The program creates an artificial intelligence that learns from its mistakes, for which it is penalized, and from good decisions, for which it is rewarded. We also integrated this program into various computer games, such as Trackmania. With minimal information about the game, the artificial intelligence figured out what it needed to do and how to do it most efficiently.

KAZALO VSEBINE

KAZALO SLIK

KAZALO TABEL

SLOVAR BESED IN KRATIC

UI - umetna inteligenca

Bot - robot, ki igra igro

NPC - Ne igralni karakter

API - aplikacijski programski vmesnik

FPS - prvoosebna strelska igra

1 UVOD

Najina raziskovalna naloga se osredotoča na samostojno učenje umetne inteligence pri različnih računalniških igrah, kot so Trackmania in Rocket League, v posebej ustvarjenih 2D in 3D okoljih. Umetna inteligenca je imela podatke o tem, kako igrati igro, ni pa imela informacij o cilju igre ali načinu zmage. Te podatke je morala pridobiti sama skozi proces igranja in prilagajanja.

Pri raziskavi sva obravnavala več izzivov, ki se pogosto pojavljajo pri razvoju umetne inteligence v videoigrah. Eden glavnih problemov je bil sam postopek učenja umetne inteligence na podlagi podatkov, ki jih prejema med igranjem. Drugi izziv je predstavljala kompleksnost različnih iger – umetna inteligenca ni bila zasnovana za eno specifično igro, temveč se je morala prilagajati različnim igralnim okoljem. Največja težava pa je bil čas, potreben za učenje, saj je ta proces lahko zelo dolgotrajen. Cilj je bil skrajšati čas učenja, kolikor je to mogoče, ne da bi pri tem zmanjšali učinkovitost algoritmov.

Raziskave sva se lotila, ker v številnih računalniških igrah umetna inteligenca pogosto ne dosega pričakovane ravni – bodisi je prešibka in ne predstavlja izziva, bodisi je nerealno močna in ne omogoča uravnotežene igralne izkušnje. Z najino raziskovalno nalogo sva želela izboljšati delovanje tovrstnih sistemov ter optimizirati proces učenja umetne inteligence, da bi bil hitrejši in učinkovitejši.

1.1 Hipoteze

Pred začetkom raziskav in kreiranja umetne inteligence sva si zadala naslednje hipoteze:

1. hipoteza: Umetna inteligenca bo zmožna igrati igro na stopnji povprečnega igralca.
2. hipoteza: Umetna inteligenca je zmožna prepoznati svoje napake.
3. hipoteza: Umetna inteligenca lahko interacira z igro na več načinov.
4. hipoteza: Uporaba umetne inteligence na e-športnih tekmovanjih je prepovedana.

2 PREGLED STANJA TEHNIKE

Midva seveda nisva prva, ki sva se odločila raziskovati področje umetne inteligence na področju računalniških iger. Med raziskovanjem po spletu sva naletela na različne objavljene raziskave, ki so se tega problema že lotile. Opazila sva, da ni nobena naloga enaka drugi zaradi številnih različnih načinov učenja in prikazovanja napredka umetne inteligence. Nekaj jih bova predstavila v nadaljevanju.

2.1 ResearchGate

Njihova raziskava je temeljila na igri, ki jo lahko igra en igralec proti računalniško vodenim nasprotnikom (NPC). Cilj raziskovalne naloge je bil nadgraditi NPC-je ter jim omogočiti logiko, podobno človeški. Uporabili so naslednje metode:[1]

- Algoritem za iskanje premikajočih se tarč, ki izboljša natančnost NPC-jev.
- Odločanje po metodi Lyapunova, ki optimizira poti NPC-jev.
- Estetika prikritih poti, kjer so analizirali, kako lahko NPC-ji izberejo poti, ki so manj vidne igralcem.

Raziskovalna ekipa je uspešno dosegla zastavljene cilje in izboljšala delovanje NPC-jev v igrah.[1]

2.2 Yosh

Yosh je spletna osebnost, ki je izvedla raziskavo o umetni inteligenci in jo predstavila na platformi YouTube. Osredotočila se je na učenje umetne inteligence z minimalnimi podatki in ciljem ustvariti enega najboljših igralcev na svetu. Po treh letih učenja je njegova umetna inteligenca postala izjemno natančna in zanesljiva ter je danes znana kot najboljši nečloveški igralec igre Trackmania.[2]

2.3 Razpoložljivi API-ji in orodja

Za razvoj umetne inteligence v računalniških igrah obstaja več API-jev in orodij, ki omogočajo hitro implementacijo in testiranje:

- OpenAI Gym – okolje za testiranje in treniranje AI modelov z uporabo okrepitvenega učenja.[3]
- Unity ML-Agents – platforma, ki omogoča integracijo umetne inteligence v Unity igre.
- TensorFlow & PyTorch – odprtokodna ogrodja za razvoj in izvajanje nevronskih mrež.
- NEAT (NeuroEvolution of Augmenting Topologies) – metoda, ki uporablja evolucijske algoritme za optimizacijo nevronskih mrež.[4]

[1] Raziskave na platformi ResearchGate: https://www.researchgate.net/, 12.10.2024

[2] Yosh na YouTube: https://www.youtube.com/@yosh-rl, 12.10.2024

[3] Raziskave na področju umetne inteligence v igrah: https://openai.com/research, 12.10.2024

[4] Yannakakis, G. N., & Togelius, J. (2018). *Artificial Intelligence and Games*. Springer: https://link.springer.com/book/10.1007/978-3-319-63519-4, 12.10.2024

2.4 Uporabne skripte in odprtokodne rešitve

Na področju umetne inteligence v igrah je na voljo več odprtokodnih skript in projektov:

- Roboti za prvoosebne strelske igre (FPS) – AI, ki uporablja nevronske mreže za analizo okolja in premikanje.
- Dirkalni AI agenti – uporaba okrepitvenega učenja za izboljšanje vožnje v dirkalnih igrah.
- AI za strategije v realnem času (RTS) – agenti, ki se učijo igranja strategij z metodo okrepitvenega učenja.

2.5 Umetna inteligenca v e-športnem okolju

V hitro razvijajočem svetu e-športa je tudi umetna inteligenca ključ do zmage, saj pomaga ekipam pri analiziranju nasprotnih ekip in prav tako tudi pri pripravljanju igre proti nasprotnim ekipam. Lahko si pomagajo na več načinov:

- Simulacija igre proti nasprotnikom – UI se prilagodi igri nasprotnika in jo simulira. Tako se lahko ekipa pripravi na tehnike igranja nasprotne analize.
- Analiza nasprotnikove igre – UI naredi analizo nasprotnikove igre, kar pomeni, da analizira, kje je najpogosteje, da bo nasprotna ekipa napadla ali pa kje so najpogosteje postavljeni pri igranju.
- Analiziranje preteklih iger – UI analizira pretekle odigrane igre in jih opozori na napake. S tem lahko ekipa popravi svoje napake in se izboljša. [5]

2.6 Zaključek

Obstoječe raziskave kažejo, da umetna inteligenca v računalniških igrah ponuja številne različne pristope in tehnike, ki omogočajo izboljšanje igranja NPC-jev in razvijanje samostojnih AI agentov. Ključnega pomena je izbira prave metode učenja glede na vrsto igre in cilje, ki jih želimo doseči.

[5] **Umetna inteligenca pri e-sportnih tekmovanjih:** https://digitaldefynd.com/IQ/ai-in-esports-case-studies/, https://recsports.uga.edu/wp-content/uploads/sites/17/2020/11/Esports_Rules_-_Updated_Fall_2020.pdf, 14.10.2024

3 MATERIAL

Za izvedbo raziskovalne naloge sva uporabila veliko različnih materialov in metod dela. Pri izvedbi sva uporabila veliko različnih aplikacij in programskih jezikov. Raziskovalna naloga je temeljila na izdelavi umetne inteligence, zato nisva uporabila veliko strojne opreme, ampak večinoma samo programsko.

3.1 Programska oprema

Programska oprema so orodja ali aplikacije, ki jih uporabljamo na pametnih telefonih, računalnikih, tablicah itd. Pomemba je za pravilno delovanje naprav ali aplikacij. Pri uporabi lahko izberemo že narejene in preverjene aplikacije/programske opreme ali pa se potrudimo in naredimo svojo.

3.1.1 Open planet

Slika 1: Logotip Openplanet

Aplikacija "Openplanet" je napredna platforma za razširitve, namenjena igri Trackmania. Igralcem omogoča dostop do številnih novih funkcij, vključno s prilagojenimi dodatki, orodji in izboljšavami igre. Z namestitvijo Openplaneta lahko igralci uporabljajo vgrajen upravitelj dodatkov za enostavno namestitev in posodabljanje dodatkov, raziskujejo notranjost igre prek Openplanet-ovega vmesnika ter izkoriščajo napredne skriptne zmogljivosti z uporabo jezika AngelScript za ustvarjanje lastnih vtičnikov. Poleg tega Openplanet omogoča popolno integracijo z Discord Rich Presence, povečuje velikost zemljevidov in omogoča mešanje okolij v igri Trackmania.[6]

[6] Openplanet avtomatiziranje: https://github.com/Openplanet-nl/Openplanet, 14.10.2024

Open planet nama je pomagal pri povezovanju najine programske kode z igro trackmania. Glavna prednost te aplikacije je, da je lahko najina koda čim hitreje pošiljala informacije igri in s tem je bilo igranje umetne inteligence bolj učinkovito in natančno.

3.1.2 Angel script

Slika 2: Angel script

3.1.2.1 Opis angel script

AngelScript je vgrajen skriptni jezik, ki je podoben jeziku C++, vendar je lažji za uporabo in prilagojen za vgradnjo v aplikacije in igre. Zasnovan je za integracijo v različne projekte, kjer omogoča razširljivost in prilagajanje programske logike brez spreminjanja izvorne kode glavnega programa.

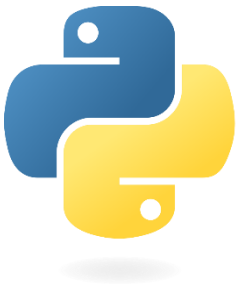
3.1.2.2 Ključne značilnosti angel script-a

- Podobnost s C++ – Sintaksa je zelo podobna jeziku C++, kar omogoča lažje učenje za programerje, ki poznajo C-jevske jezike.
- Varna integracija – Omogoča varno izvajanje skriptov v gostiteljskem programu, saj ima nadzor nad dostopom do sistemskih virov.
- Orientiran na igre – Pogosto se uporablja v igrah in aplikacijah za dodajanje skriptnih funkcij (npr. Openplanet za Trackmania).
- Podpora za objektno usmerjeno programiranje (OOP) – Omogoča ustvarjanje razredov, dedovanje, virtualne funkcije itd.
- Dinamična in statična tipizacija – Podpira tako dinamične kot statične tipe za bolj prilagodljivo programiranje.

3.1.2.3 Uporaba angel script-a

- AngelScript se uporablja v različnih igrah in pogonih, kot so:
- Trackmania (prek Openplanet)
- Amnesia: The Dark Descent
- S.T.A.L.K.E.R. Anomaly modifikacije
- 3D Game Studio

3.1.3 Python



Slika 3: Logotip programskega jezika Pyhton

Python je visokonivojski, interpretirani programski jezik, ki je znan po svoji enostavnosti, berljivosti in vsestranskosti. Uporablja se za razvoj spletnih aplikacij, podatkovno analitiko, umetno inteligenco, avtomatizacijo, znanstveno računalništvo in še mnogo več.

3.1.3.1 Ključne lastnosti Phyton-a

- Enostavna in berljiva sintaksa – Prijazen začetnikom, saj uporablja preprosto strukturo (brez oklepajev {} kot v C ali Java, temveč zamike).
- Večnamenskost – Uporablja se v spletnih aplikacijah, strojni inteligenci, podatkovni analitiki, avtomatizaciji in še več.
- Velika zbirka knjižnic – Ima ogromno število knjižnic za skoraj vsako nalogo, npr. NumPy, Pandas, TensorFlow, Flask, Django itd.
- Interpreted Language – Koda se izvaja vrstico po vrstico, kar olajša odpravljanje napak.
- Podpora za več programskih paradigem – Podpira objektno usmerjeno programiranje (OOP), funkcijsko programiranje in proceduralno programiranje.
- Odprtokoden in brezplačen – Python je odprtokoden, kar pomeni, da ga lahko uporablja in prispeva vanj vsakdo.

3.1.4 Visual studio code

Slika 4: Logotip programske opreme Visual studio code

Visual Studio Code (VS Code) je brezplačen in lahko razumljiv urejevalnik kode, ki ga je razvilo podjetje Microsoft. Je eden najbolj priljubljenih urejevalnikov zaradi svoje hitrosti,

prilagodljivosti in veliko možnosti razširitve. Pri najini nalogi sva uporabila to aplikacijo, saj je najbolj uporabniku prijazna aplikacija

3.1.4.1 Ključne lastnosti VS-ja

- Podpora za več programskih jezikov – Podpira Python, JavaScript, C++, Java, HTML/CSS, Go, Rust in še mnogo več.
- IntelliSense – Pametno dopolnjevanje kode z namigi in predlogi.
- Vgrajena podpora za Git – Omogoča enostavno uporabo Git repozitorijev neposredno v urejevalniku.
- Razširitve (Extensions) – Lahko dodaš tisoče razširitev za nove funkcije (npr. podpora za Jupyter Notebook, AI asistente, debuggerje itd.).
- Debugger – Vgrajena podpora za odpravljanje napak v različnih jezikih.
- Integrirani terminal – Omogoča izvajanje ukazov brez zapustitve urejevalnika.
- Lahek in hiter – Hitrejši od klasičnega Visual Studio IDE-ja.

3.1.5 Trackmania

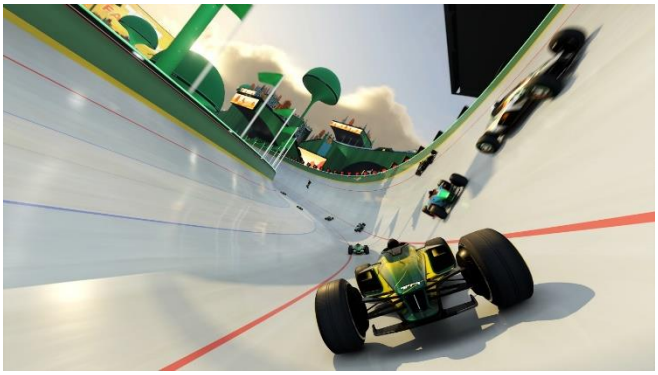
Slika 5: Trackmania

Trackmania je serija dirkaških računalniških iger, ki jo je razvilo francosko podjetje Nadeo. Gre za eno najbolj edinstvenih dirkaških iger, saj se ne osredotoča na klasične dirke proti drugim igralcem, temveč na tekmovanje s časom. Cilj igre je, da igralci odpeljejo stezo v najkrajšem možnem času, pri čemer šteje vsaka stotinka sekunde. Igra je znana po arkadnem slogu vožnje, visokih hitrostih, ostrih zavojih in spektakularnih skokih, zaradi česar je igranje izjemno dinamično in napeto.

Ena od največjih posebnosti Trackmanie je močno orodje za ustvarjanje lastnih prog. Igralci lahko sestavijo svoje dirkališče z uporabo različnih elementov, kot so ovinki, skakalnice, zanke in celo posebni moduli, ki spreminjajo fiziko vožnje. Skupnost ima dostop do tisočih prog, ki jih delijo drugi igralci, kar pomeni, da je igranje vedno sveže in zanimivo. Poleg tega je igra močno povezana z e-športom, saj vsebuje lestvice najboljših časov na posameznih

progah, uradne turnirje ter izzive, v katerih se lahko igralci pomerijo proti duhovom najboljših voznikov na svetu.

Serija Trackmania se je začela leta 2003 z izvirno igro Trackmania, ki je hitro pridobila zveste igralce. Leta 2006 je izšla brezplačna različica Trackmania Nations, ki je bila posebej zasnovana za e-šport in predstavila priljubljeno okolje Stadium. Kasneje je prišla nadgrajena različica Trackmania United, ki je združila vsa prejšnja okolja v eno igro. Med letoma 2011 in 2017 so izšle igre v seriji Trackmania 2, ki so ponudile nova okolja, kot so Canyon, Valley in Lagoon, vsako s svojo unikatno fiziko vožnje. Leta 2020 je Ubisoft izdal moderno različico preprosto imenovano Trackmania, ki uvaja naročniški model in redno dodaja nove uradne steze ter tekmovanja.



Slika 6: Trackmania gameplay

Eden od razlogov, zakaj je Trackmania tako priljubljena, je njena dostopnost in preprosta, a zahtevna igralna mehanika. Igro lahko igraš s tipkovnico ali igralnim ploščkom, brez potrebe po zapletenih nastavitvah. Fizika vožnje je zasnovana tako, da je enostavna za učenje, a težka za obvladovanje, saj zahteva natančne gibe, hitro reakcijo in popolno poznavanje prog. Igra je tudi zelo priljubljena v skupnosti modderjev, ki ustvarjajo dodatne vsebine in orodja, kot je Openplanet, ki omogoča razširitev funkcionalnosti igre z dodatnimi skriptami in vtičniki.

Trackmania ostaja edinstvena in izstopa med drugimi dirkaškimi igrami, saj ne temelji na klasičnih avtomobilskih simulacijah, ampak na preprostosti, hitrosti in neomejeni ustvarjalnosti. Čeprav je na prvi pogled videti preprosta, ponuja izjemno globino in skoraj neskončno vsebino, saj igralci nenehno ustvarjajo nove steze in tekmujejo za boljše čase.

3.1.6 Rocket league

Slika 7: Rocket league

Rocket League je priljubljena športno-dirkaška igra, ki združuje nogomet in avtomobile ter ponuja hitro in dinamično igranje. Razvilo jo je ameriško podjetje Psyonix, prvič pa je izšla

leta 2015. Igra je edinstvena, saj namesto klasičnih nogometašev igralci nadzorujejo avtomobile z raketnim pogonom, s katerimi skušajo zadeti velikansko žogo v nasprotnikova vrata. Zaradi svoje preprostosti, a hkrati globoke mehanike igranja, je postala izjemno priljubljena tako med običajnimi igralci kot tudi v svetu e-športa.

Igranje temelji na osnovnem konceptu nogometa, vendar s številnimi zanimivimi preobrati. Igralci se pomerijo v ekipah, najpogosteje v formatih ena na ena, dva na dva ali tri na tri, pri čemer uporabljajo avtomobile za vodenje in udarjanje žoge. Avtomobili lahko skačejo, izvajajo akrobatske trike, pospešujejo z uporabo raketnega pogona ter vozijo po stenah in celo po stropu, kar omogoča izjemno dinamično igranje. Fizični model igre zahteva natančno upravljanje avtomobila in dobro koordinacijo, saj se mora igralec naučiti, kako pravilno zadeti žogo, kako nadzorovati svoj zračni gib in kako se pravilno pozicionirati na igrišču.

Poleg klasičnih nogometnih tekem igra ponuja različne druge načine igranja. Na voljo so načini, kot je Hoops, ki je podobna košarki, Rumble, ki dodaja različne posebne sposobnosti, Dropshot, kjer igralci razbijajo tla pod nasprotnikovo stranjo igrišča, in Snow Day, ki je hokejska različica igre. Igralci lahko sodelujejo tudi v tekmovalnem načinu, kjer se uvrščajo v različne rangirane stopnje, od Bronze do Grand Champion in celo Supersonic Legend, kar je najvišja možna stopnja.

Rocket League je zelo priljubljena v e-športni sceni. Ima uradno profesionalno ligo, imenovano Rocket League Championship Series (RLCS), kjer se najboljši igralci in ekipe z vsega sveta pomerijo za visoke denarne nagrade. Poleg tega je igra močno vpeta v pretočne platforme, kot je Twitch, kjer jo redno spremljajo milijoni gledalcev.

Ena od največjih prednosti Rocket League je, da je dostopna brezplačno na več platformah, vključno s PC-jem, PlayStationom, Xboxom in Nintendo Switchom. Poleg tega podpira cross-play, kar pomeni, da lahko igralci različnih platform igrajo skupaj. Od leta 2020 je igra prešla pod lastništvo Epic Games, kar je prineslo še več vsebin, sezonske dogodke in kozmetične dodatke, s katerimi lahko igralci prilagodijo svoj avtomobil.

Slika 8: Rocket league gameplay

Rocket League je zaradi svoje enostavne zasnove, a hkrati visoke stopnje spretnosti, ki je potrebna za obvladovanje igre, ena najbolj priljubljenih iger na svetu. Je igra, ki jo lahko kdorkoli hitro vzame v roke in začne igrati, a hkrati ponuja neomejen prostor za izboljševanje in tekmovalnost, zaradi česar ostaja zanimiva tudi po letih igranja.

3.1.7 Open CV

Slika 9: logotip OpevCV

OpenCV (Open Source Computer Vision Library) je odprtokodna knjižnica za računalniški vid in obdelavo slik, ki omogoča hitro in učinkovito analizo vizualnih podatkov. Razvil jo je Intel leta 2000, danes pa je ena najpogosteje uporabljenih knjižnic za računalniški vid na svetu. OpenCV je priljubljen med raziskovalci, razvijalci umetne inteligence, strokovnjaki za strojno učenje in inženirji, saj ponuja širok nabor funkcij za analizo slik, prepoznavanje objektov, sledenje gibanja, obdelavo videa in še mnogo več.[7]

Knjižnica je zasnovana tako, da je optimizirana za hitrost in lahko deluje na različnih platformah, vključno z Windows, macOS, Linux, Android in iOS. Podpira več programskih jezikov, kot so Python, C++, Java in MATLAB, kar omogoča prilagodljivost pri razvoju aplikacij. Poleg tega je združljiva s strojno pospešenimi tehnologijami, kot so CUDA, OpenCL in Intel TBB, kar omogoča učinkovito izvajanje zahtevnih algoritmov tudi na grafičnih procesorjih (GPU).

Ena od ključnih lastnosti OpenCV-ja je njegova bogata zbirka vgrajenih algoritmov in funkcij, ki pokrivajo široko področje obdelave slik. Knjižnica omogoča branje in pisanje slik ter videoposnetkov, barvno pretvorbo, filtriranje, detekcijo robov, segmentacijo objektov in popravljanje popačenja leč. Prav tako vključuje napredne funkcije, kot so prepoznavanje obrazov, sledenje objektov, optično prepoznavanje znakov (OCR), 3D rekonstrukcijo in stereoskopski vid.

Zelo pomemben vidik OpenCV-ja je njegova uporaba v aplikacijah umetne inteligence in strojnega učenja. Knjižnica vsebuje module, kot je cv::dnn, ki omogoča izvajanje globokih nevronskih mrež (DNN) za prepoznavanje objektov in razvrščanje slik. Integracija z ogrodji, kot so TensorFlow, PyTorch in Caffe, omogoča enostavno uporabo predhodno naučenih modelov za naloge, kot so detekcija vozil, klasifikacija predmetov, zaznavanje gibanja in analiza obraznih izrazov.

OpenCV se uporablja v številnih industrijskih in raziskovalnih področjih, vključno z avtonomnimi vozili, medicinskim slikanjem, nadzornimi sistemi, roboti, AR/VR aplikacijami in športno analitiko. V avtomobilskem sektorju ga uporabljajo za sisteme zaznavanja objektov

[7] Uporaba OpenCV za računalniški vid v AI projektih: https://opencv.org/, 3.11.2024

in samodejno vožnjo, v medicini za analizo rentgenskih slik, v nadzornih sistemih pa za prepoznavanje obrazov in zaznavanje gibanja.

Ker je OpenCV odprtokoden, ima zelo aktivno skupnost razvijalcev, ki neprestano izboljšujejo knjižnico in dodajajo nove funkcionalnosti. Poleg tega obstaja veliko spletnih virov, kot so učbeniki, vodiči in forumi, ki pomagajo razvijalcem pri učenju in uporabi knjižnice v svojih projektih. OpenCV ostaja ena najpomembnejših in najzmogljivejših knjižnic za računalniški vid ter je ključno orodje pri razvoju sodobnih vizualnih aplikacij in inteligentnih sistemov.

3.2 Strojna oprema

Strojna oprema so pripomočki, ki jih uporabljamo fizično to so: računalnik, tipkovnica, miška, ... Računalnik je bil pomemben del najine raziskovalne naloge, saj se je vse dogajalo na njemu.

3.2.1 Računalnik

Slika 10: Računalnik

Računalnik je elektronska naprava, ki obdeluje podatke in izvaja različne naloge s pomočjo programov in algoritmov. Uporablja se za izračune, shranjevanje in obdelavo informacij, avtomatizacijo procesov, komunikacijo in številne druge namene. Sestavljen je iz strojne in programske opreme, ki skupaj omogočata njegovo delovanje.

Osnovne komponente računalnika vključujejo procesor (CPU), pomnilnik (RAM), trdi disk ali SSD, grafično kartico (GPU), matično ploščo in napajalnik. Procesor je osrednja enota, ki izvaja ukaze, RAM omogoča hitri dostop do podatkov, medtem ko trdi disk ali SSD služi za dolgoročno shranjevanje informacij. Grafična kartica je ključna pri obdelavi vizualnih podatkov, še posebej pri igranju iger, obdelavi videa in uporabi umetne inteligence.

Programska oprema računalnika vključuje operacijski sistem (kot so Windows, macOS ali Linux), ki omogoča uporabniku upravljanje naprave in poganjanje različnih programov. Poleg operacijskega sistema računalnik uporablja različne aplikacije, kot so spletni brskalniki, pisarniški programi, igre, programska orodja za razvijalce in številni drugi programi, ki omogočajo izvajanje različnih nalog.

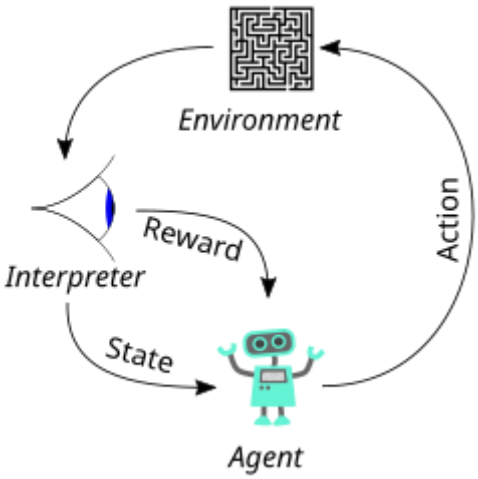
Računalniki obstajajo v različnih oblikah in velikostih, od osebnih računalnikov (PC-jev) in prenosnikov do superračunalnikov in vgrajenih sistemov. Osebni računalniki se uporabljajo za vsakodnevna opravila, kot so delo, zabava in izobraževanje, medtem ko so superračunalniki namenjeni izjemno zahtevnim izračunom, kot so vremenske napovedi, simulacije v znanosti in raziskave umetne inteligence. Manjši računalniki, kot so pametni telefoni, tablice in vgrajeni sistemi, se uporabljajo v avtomobilih, gospodinjskih aparatih in industrijskih napravah.

Z razvojem tehnologije postajajo računalniki vse zmogljivejši in učinkovitejši. Sodobni računalniki omogočajo povezovanje z internetom, izvajanje kompleksnih nalog z umetno inteligenco, igranje iger s foto realistično grafiko in avtomatizacijo številnih procesov v industriji in vsakdanjem življenju. Zaradi svoje vsestranskosti so postali nepogrešljiv del sodobne družbe in so ključni pri razvoju številnih področij, kot so znanost, medicina, inženirstvo in komunikacija.

4 METODE DELA

Z raziskovalno nalogo sva začela pri osnovah umetne inteligence, najprej sva se morala naučiti kako narediti program, ki se bo učil na svojih napakah. Raziskala sva po spletu in ugotovila, da lahko uporabiva metodo “reinforcement learning”, poleg te sva uporabila tudi druge metode. Kot drugo metodo sva uporabila metodo, kjer program shrani podatke, ki so pomembni, to so tipke,katere je pritisnil ob določenem času in kakšno nagrado je dobil za določeno odločitev.

4.1 Reinforcement learning



Slika 11: Reinforcement learning

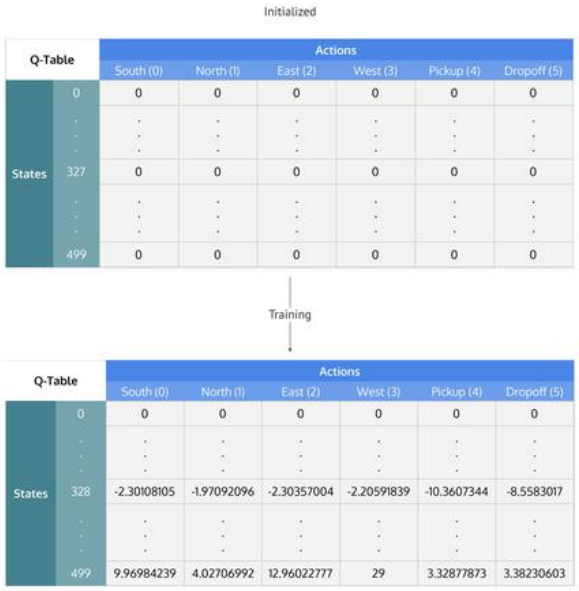
Reinforcement learning je metoda učenja umetne inteligence. Uporablja vizualni podatek (v najinem primeru sliko, ki jo dobiva s pomočjo openCV knjižnice) in prebere sliko v RGB in jo za lažje branje spremeni v črno-belo sliko oziroma v “grayscale”. Sliko dobi v hitrem času, saj dobi 20 slik na sekundo, zato da je položaj avta vedno znan umetni inteligenci in posledično je tudi lažje in bolj učinkovito odločanje. Po-tem ko dobi program vse potrebne informacije, se odloči za akcijo glede na podane informacije s slike. Za določeno akcijo se spremeni tudi spremenljivka z imenom nagrada.[8]

Spremenljivka nagrada je zelo pomemben del programa, saj iz te spremenljivke program izve ali je naredil dobro potezo ali slabo. Nagrada mu še ni znana po končanem programu, ampak izve število nagrad tudi med določenim tekom. Spremenljivka se začne na 0, po prvi potezi se lahko spremenljivka spremeni v pozitivno ali v negativno. Če je nagrada pozitivna, to pomeni dobro potezo umetne inteligence, če pa je negativna, je še prostor za izboljšavo.[8]

Tak način učenja porabi manj časa kot ostali, poleg tega se napredek shrani tudi, če se program zaradi napake zaustavi.

[8] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction (2nd Edition)*. MIT Press: https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf, 15.10.2024

4.2 Q-table learning



Slika 12: Primer Q-table learning

Q-learning je metoda strojnega učenja, kjer se računalnik (agent) uči sprejemati najboljše odločitve na podlagi preizkušanja različnih dejanj in analiziranja njihovih rezultatov. To pomeni, da agent postopoma odkriva, katera dejanja ga pripeljejo do želenega cilja, pri tem pa ne potrebuje vnaprej določenih pravil ali podatkov o okolju.[9]

Ko agent deluje v določenem okolju, se vedno nahaja v nekem stanju in lahko izvede različna dejanja. Na primer, če bi bil agent robot, ki se premika po labirintu, bi njegovo trenutno stanje predstavljalo položaj v labirintu, dejanja pa bi bila premiki v različne smeri. Po vsakem dejanju prejme povratno informacijo v obliki nagrade ali kazni – če se premakne proti izhodu, dobi pozitivno nagrado, če pa zadene v zid, dobi kazen.

Za shranjevanje informacij o vrednosti posameznih dejanj v določenih stanjih uporablja agent Q-tabelo. Ta tabela vsebuje ocene, kako dobra so posamezna dejanja, in se sproti posodablja glede na prejete nagrade. Na začetku so vse vrednosti naključne, vendar se s časom izboljšujejo, saj agent vedno bolj natančno razume, katera dejanja vodijo do boljših rezultatov.[9]

Osnovna formula za posodabljanje teh vrednosti upošteva prejšnje izkušnje in novo pridobljene informacije. Če agent ugotovi, da je neko dejanje prineslo dobro nagrado, bo v prihodnosti to dejanje pogosteje izbral. Če pa dobi kazen, bo poskusil drugačno strategijo. S ponavljanjem in izboljševanjem svojih odločitev postopoma najde optimalen način delovanja.

Pomemben del učenja je ravnotežje med raziskovanjem in izkoriščanjem. Na začetku agent raziskuje – preizkuša različne možnosti, tudi če še ne ve, katera je najboljša. Ko pa se dovolj

[9] Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3-4), 279-292: https://link.springer.com/article/10.1007/BF00992698, 15.10.2024

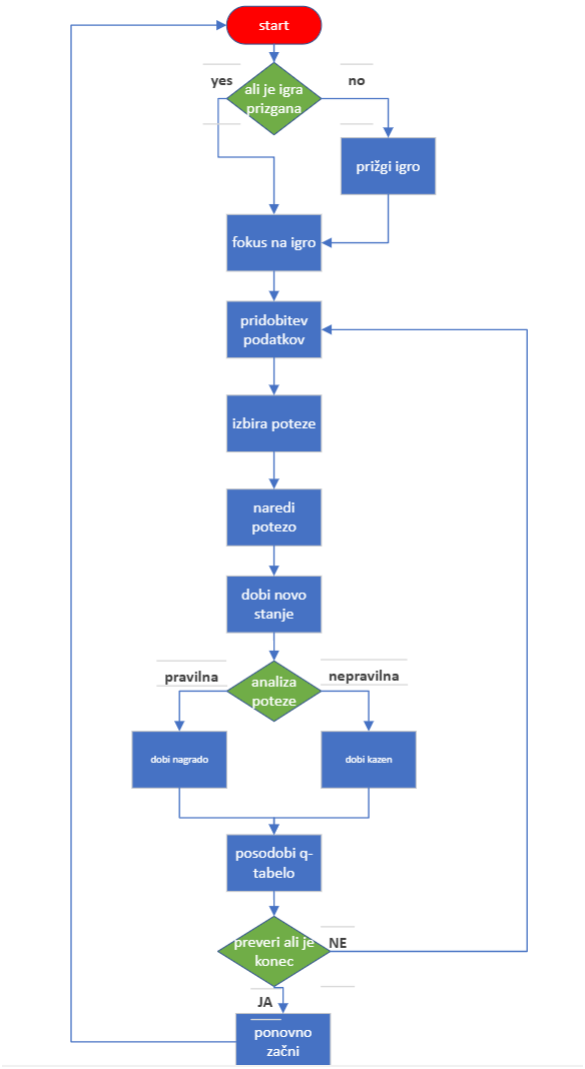
nauči, začne izkoriščati pridobljeno znanje in večinoma izbira tista dejanja, ki so mu v preteklosti prinesla največ koristi. Na ta način se Q-learning sčasoma izboljšuje in agent postane vedno bolj učinkovit pri reševanju nalog.

5 POTEK DELA

Delo sva začela z raziskavo po spletu, da sva dobila informacije, kakšen je pravilen začetek za ustvarjanje umetne inteligence. Opazila sva, da je pomembno že na začetku se odločiti, s katero metodo bova učila umetno inteligenco. Za začetek sva uporabila reinforcement learning in preizkusila samo s to metodo, ampak ni šlo vse po načrtu, zato sva spremenila program na 1-learning. Začela sva na posebej narejenemu 2D okolju.

5.1 Diagram poteka programske kode umetne inteligence za igro Trackmania

Program najprej preveri ali je igra že prižgana in jo fokusira, če še ni prižgana, jo prižge. Potem pridobi podatke o hitrosti in o poziciji. Glede na vse podane podatke se agent odloči za potezo in nato naredi potezo, za katero dobi pozitivno ali negativno nagrado. Po vsakem koraku pregleda ali je že agent prišel do konca in resetira dirko ali pa nadaljuje trenutno.

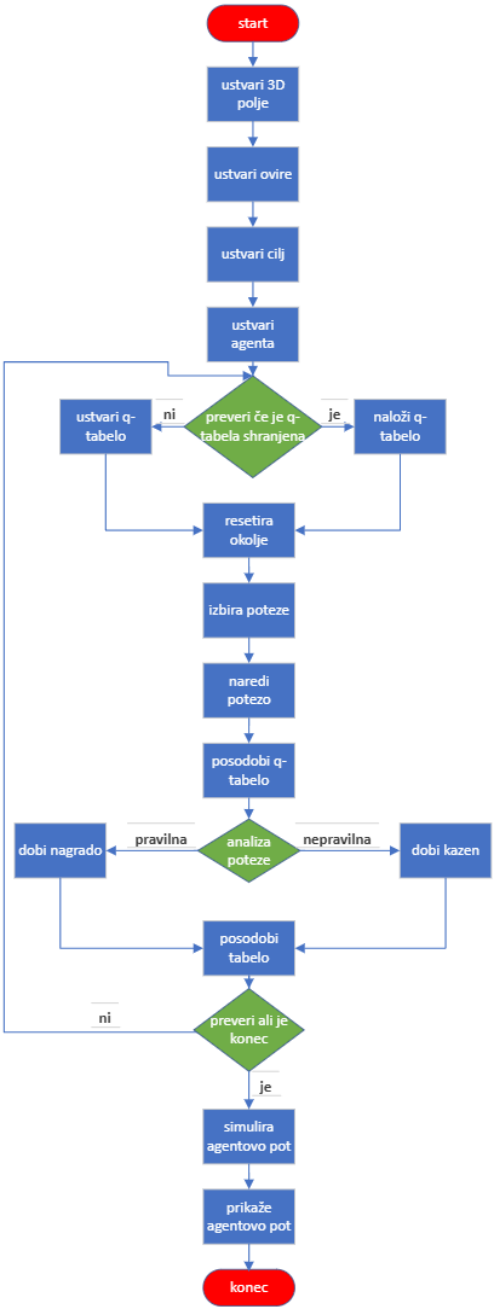


Slika 13: Diagram poteka Trackmania

5.2 Diagram poteka programske kode za reševanje 3D labirinta

Program za umetno inteligenco je prikazan spodaj za lažje razumevanje programa. Program najprej ustvari 3D okolje (kocko). Po-tem program izbere naključne lokacije in na njih postavi

ovire. Po-tem program naključno izbere, kje je cilj. Ko je okolje pripravljeno, se ustvari agent oz. igralec nato preveri ali je q-tabela že narejena, če je, jo naloži, če pa ni, ustvari novo. Nato se igra začne in agent dobiva nagrado pozitivno ali negativno glede na premik, ki ga naredi. Po vsakem koraku napredek napiše v q-tabelo in preveri ali je agent prišel do cilja. Če je uspešno prišel do cilja, se program zaključi, če pa ni prišel do cilja, se koraki ponovijo.



Slika 14: Diagram poteka 3D okolje

6 REZULTATI

V sklopu raziskave smo razvili in preizkusili umetno inteligenco (UI) z uporabo metod globokega ojačevalnega učenja (Deep Reinforcement Learning) v različnih računalniških igrah. Naš cilj je bil analizirati sposobnost UI pri učenju optimalnih poti in strategij v 2D in 3D okoljih ter preveriti njeno učinkovitost pri kompleksnejših igrah, kot sta Rocket League in Trackmania.

Doseženi rezultati

Razvita UI se je v 2D in 3D okoljih izkazala iza zjemno uspešno. Pri simulacijah je bila sposobna:

- Učenja popolnoma optimalnih linij gibanja ter iskanja najboljših poti skozi zahtevne ovire.
- Optimizacije strategij gibanja, kar je omogočalo hitrejši in bolj učinkovit prehod skozi nivoje.
- Sprotnega prilagajanja odločitev glede na spremembe v okolju, kar kaže na visoko stopnjo prilagodljivosti.

V preprostih 2D igrah je UI hitro dosegla optimalno pot in se naučila igrati na ravni popolnosti. Tudi v bolj zapletenih 3D okoljih je uspešno našla idealne linije premikanja ter učinkovito navigirala skozi prostor.

Omejitve in neuspehi

Kljub uspehu v 2D in 3D simulacijah pa je bila UI manj učinkovita pri kompleksnih igrah, kot sta Rocket League in Trackmania.

- Rocket League: Zaradi visoke dinamike igre, ki vključuje nelinearno fiziko, hitro spreminjajoča se okolja in zahtevo po izjemno preciznem nadzoru, se UI ni bila zmožna naučiti popolnih linij gibanja v realnem času. Proces učenja bi zahteval enormno računsko moč in dolge čase treniranja, kar je preseglo omejitve raziskave.[10]
- Trackmania: Čeprav je igra deterministična, je optimizacija vožnje na najvišji ravni izredno zahtevna. UI je uspela odkriti dobre, a ne popolne linije, saj je iskanje absolutno najhitrejše poti po stezi zahtevalo izjemno natančno prilagajanje, ki bi zahtevalo mesece ali leta učenja s trenutno razpoložljivimi sredstvi.

[10] Razvoj umetne inteligence v Rocket League: https://deepmind.com/research/highlighted-research/rocket-league, 8.11.2024

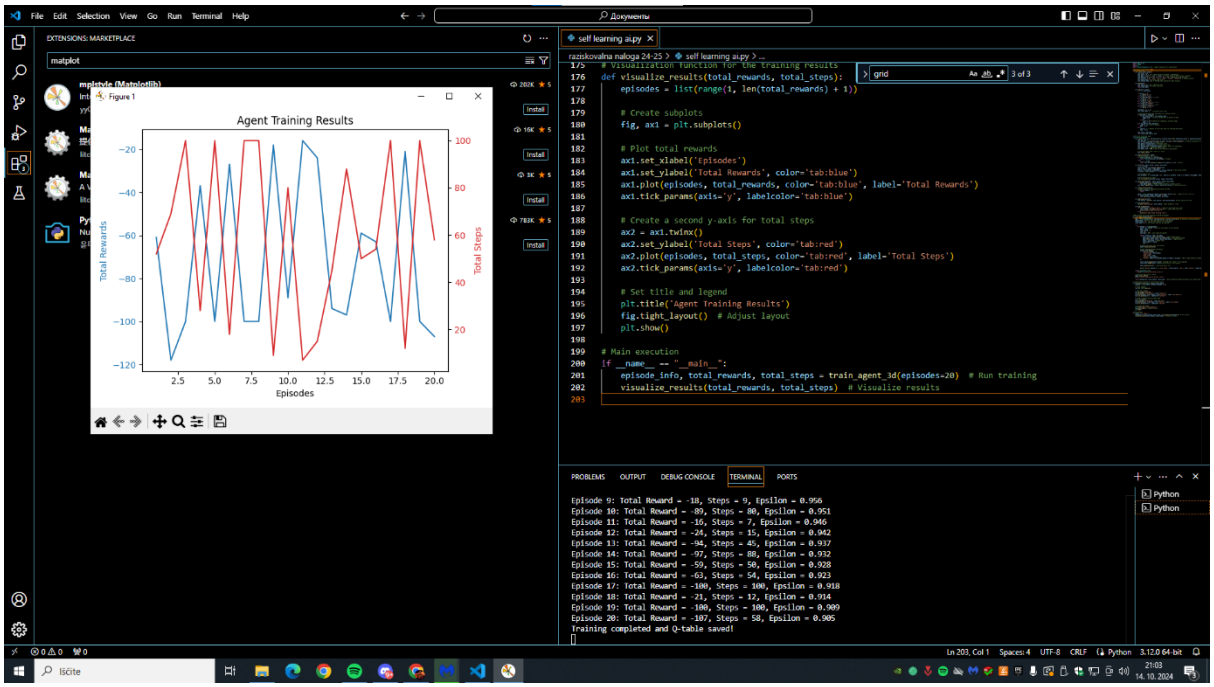
Med učenjem umetne inteligence sva vse podatke zapisala v najino tabelo. Zapisala sva najslabši in najboljši čas iz vsakega treninga.

	najslabši	najboljši	povprečje
trening 1	/	115,055	115,06
trening 2	120,507	105,605	113,06
trening 3	117,476	100,204	108,84
trening 4	102,221	99,244	100,73
trening 5	100,553	92,971	96,76

Tabela 1: Časi treninga UI Trackmania

Kot je iz podatkov razvidno so se časi izboljšali iz treninga v trening. V prvem treningu se je zgodilo, da agent, ki je vozil avto, ni bil uspešen, zato najslabšega časa ni. Izračunala sva tudi povprečen čas, ki ga je agent porabil, da je odpeljal od začetka do cilja.

Med posameznimi treningi UI v 3D in 2D labirintu sva podatke lahko izbrala iz grafa. Graf nama je prikazal, kakšno nagrado je dobil agent za vsak korak in tudi koliko korakov je potreboval, da je prišel do konca. Primer:



Slika 15: Prikaz grafa agenta med treningom

7 RAZPRAVA

7.1 Hipoteza 1

1. hipoteza: Umetna inteligenca bo zmožna igrati igro na stopnji povprečnega igralca.

Hipoteza je potrjena. UI je zmožna napredovati v igri vsaj do stopnje povprečnega igralca. Stopnja igranja umetne inteligence je močno odvisna od časa, ki ga porabimo za njeno učenje, ter od metode, ki jo uporabljamo za treniranje.

Umetna inteligenca lahko doseže raven povprečnega igralca z različnimi tehnikami, kot so okrepitevno učenje (Reinforcement Learning), nevronske mreže in evolucijski algoritmi. S temi metodami se UI postopoma uči optimalnih strategij igranja, prepoznava vzorce in izboljšuje svoje odločitve na podlagi preteklih izkušenj. Daljše in boljše treniranje omogoča UI, da se prilagaja različnim situacijam v igri in zmanjšuje napake, podobno kot bi to storil človek.

V številnih primerih je UI že presegla povprečnega igralca in se povzpela na raven najboljših profesionalcev. Projekti, kot sta AlphaStar (StarCraft II) in OpenAI Five (Dota 2), so dokazali, da lahko umetna inteligenca s primernim učenjem doseže ali celo preseže svetovno elito v določenih igrah.

Zaradi svoje sposobnosti hitrega prilagajanja, natančnosti in neomejene vzdržljivosti ima UI potencial, da ne samo igra na ravni povprečnega igralca, temveč postane eden najboljših igralcev v skoraj vsaki igri, če ji omogočimo dovolj časa za učenje.

7.2 Hipoteza 2

2. hipoteza: Umetna inteligenca je zmožna prepoznati svoje napake.

Hipoteza je potrjena. Med testiranjem umetne inteligence za igro Trackmania sva ugotovila, da je bila UI zmožna prepoznati svoje napake in jih popraviti.

Umetna inteligenca prepoznava napake z različnimi pristopi, odvisno od načina učenja in zasnove algoritma. Najpogosteje uporablja okrepitevno učenje (Reinforcement Learning), kjer prejema povratne informacije o svojih dejanjih in se na podlagi tega prilagaja. Če UI opazi, da določen pristop vodi v slabši rezultat (npr. trčenje v oviro ali počasnejši čas kroga), bo v naslednjih poskusih poskušala najti boljšo rešitev.

Poleg tega lahko UI uporablja analizo podatkov – primerja prejšnje vožnje, prepozna vzorce in ugotovi, kje je naredila napako. Če so na voljo predhodno shranjeni podatki (ghost posnetki boljših voženj ali simulacije optimalnih poti), jih lahko UI analizira in izboljša svojo strategijo.

V nekaterih primerih lahko umetna inteligenca uporablja prenos znanja (Transfer Learning), kjer se iz napak uči na podlagi predhodno pridobljenih izkušenj ali izkušenj drugih modelov. To ji omogoča, da hitreje izboljšuje svoje zmogljivosti brez ponavljanja istih napak.

Zaradi teh sposobnosti UI ne le prepozna svoje napake, ampak jih lahko tudi sistematično odpravlja, kar ji omogoča stalno napredovanje in boljše rezultate v igri.

7.3 Hipoteza 3

3. Hipoteza: Umetna inteligenca lahko intericira z igro na več načinov.

Tudi to hipotezo lahko potrdiva. Med testiranjem sva ugotovila, da lahko UI dostopa do igre preko Open planeta in preko simulacije pritiskanja gumbov na tipkovnici. Do igre lahko dostopa tudi na druge načine, ki jih midva nisva uporabila. To so:

- **Simulacija gibanja miške in klikov** – Program premika miško in klikne na določene dele zaslona, kot bi to storil igralec.
- **Uporaba makrov in skript** (npr. AutoHotkey, PyAutoGUI) – Program predvaja vnaprej določene sekvence akcij, ki se ponavljajo.
- **Modifikacija vhodnih naprav** (npr. virtualni kontrolerji) – Program emulira igralni plošček ali tipkovnico in pošilja ukaze igri.
- **Programska manipulacija pomnilnika igre** – Program lahko poseže v pomnilnik računalnika in na ta način dobi informacije o igri. Ta način ni etičen zaradi tega, ker se to uporablja za goljufije pri igrah.

7.4 Hipoteza 4

4. hipoteza: Uporaba umetne inteligence na e-športnih tekmovanjih je prepovedana.

Hipoteza je potrjena. Uporaba umetne inteligence v e-športu je dovoljena za treniranje ekip, vendar ostaja prepovedana na tekmovanjih zaradi ohranjanja poštenosti med igralci.

E-športna tekmovanja temeljijo na veščinah, refleksih, strategiji in sposobnosti igralcev, da se hitro prilagajajo različnim situacijam v igri. Medtem ko umetna inteligenca lahko pomaga ekipam pri analizi iger, optimizaciji strategij in izboljšanju reakcij igralcev med treningi, njena uporaba v tekmovalnem okolju ni dovoljena, saj bi igralcem omogočila nepošteno prednost z avtomatizacijo nalog, izboljšanjem natančnosti in hitrejšim sprejemanjem odločitev.

Večina organizatorjev e-športnih turnirjev, kot so ESL, Riot Games, Valve in Blizzard, ima stroga pravila glede uporabe zunanjih programov ali pripomočkov, ki bi lahko vplivali na igro. Vsaka oblika avtomatizacije, ki presega človeške sposobnosti med tekmovanji, se šteje za goljufijo in je kaznovana z diskvalifikacijo ali celo doživljenjsko prepovedjo sodelovanja.

Kljub temu pa umetna inteligenca igra pomembno vlogo pri pripravah na tekmovanja. Z njeno pomočjo ekipe analizirajo nasprotnike, preučujejo vzorce igranja in izvajajo simulacije različnih scenarijev. Napredni AI-sistemi, kot sta AlphaStar za StarCraft II in OpenAI Five za Doto 2, so pokazali izjemne sposobnosti pri igranju iger, zato jih ekipe uporabljajo za izboljšanje svojih strategij in usposabljanje igralcev.

Čeprav UI pomembno prispeva k razvoju e-športa, njena uporaba v tekmovalnem okolju ostaja prepovedana, da se ohrani poštenost in enakopravnost med igralci.

8 ZAKLJUČEK

Raziskava je pokazala, da lahko umetna inteligenca (UI) zelo učinkovito osvoji optimalne linije gibanja v klasičnih 2D in 3D okoljih. V teh simulacijah je bila sposobna najti najkrajše in najučinkovitejše poti, premagovati ovire ter se sproti prilagajati spremembam v okolju. Njena sposobnost učenja in prilagajanja je omogočila optimizacijo strategij gibanja, kar je pripomoglo k hitrejšemu in bolj natančnemu doseganju ciljev.

Vendar pa so se pri kompleksnejših igrah, kot sta Rocket League in Trackmania, pojavile pomembne omejitve. Ti naslovi zahtevajo izjemno natančno gibanje, hiter odziv na nepredvidljive situacije in sposobnost prilagajanja fizikalnim zakonitostim igre. Zaradi teh dejavnikov bi bilo učenje umetne inteligence bistveno zahtevnejše in dolgotrajnejše, saj bi zahtevalo izredno veliko računsko moč in dolge čase treniranja, ki presegajo zmogljivosti naše raziskave.

8.1 Možnosti za nadaljnje raziskave

Da bi izboljšali delovanje UI v kompleksnih okoljih in premagali obstoječe omejitve, bi bilo smiselno:

- Uporabiti naprednejše metode učenja, kot je učenje na podlagi modelov (model-based reinforcement learning), ki omogoča hitrejše in učinkovitejše načrtovanje gibanja.
- Preučiti možnost kombinacije nevronskih mrež s tradicionalnimi algoritmi iskanja poti, kar bi lahko pripomoglo k večji natančnosti odločanja.
- Izkoristiti zmogljivejšo strojno opremo, ki bi omogočila hitrejše procesiranje podatkov ter krajši čas učenja umetne inteligence.

Z nadgradnjo metod in tehnologije bi bila UI morda sposobna osvojiti tudi igre, ki zahtevajo izjemno natančnost gibanja in kompleksno razumevanje fizike, ter tako doseči še višjo stopnjo učinkovitosti in prilagodljivosti.

9 POVZETEK

V raziskovalni nalogi sva preučevala samostojno učenje umetne inteligence (UI) v različnih računalniških igrah, vključno s Trackmanio, Rocket League ter posebej zasnovanima 2D in 3D simulacijama. Cilj raziskave je bil ugotoviti, kako učinkovito se lahko UI prilagodi različnim igram, najde optimalne poti in izboljša svoje strategije igranja s pomočjo metod ojačevalnega učenja (Reinforcement Learning).

9.1 Glavni rezultati

V 2D in 3D okoljih se je UI izkazala izjemno uspešno, saj je bila sposobna:

- Naučiti se popolnih linij gibanja in optimalnih poti skozi zahtevne ovire.
- Optimizirati strategije gibanja, kar je omogočalo hitrejši in učinkovitejši napredek skozi nivoje.
- Sproti prilagajati svoje odločitve glede na spremembe v okolju, kar kaže na visoko stopnjo prilagodljivosti.

V kompleksnejših igrah (Trackmania, Rocket League) je UI dosegla omejene rezultate:

- V Trackmanii je bila zmožna prepoznati dobre linije vožnje, vendar ne popolnih, saj je iskanje optimalne poti zahtevalo preveč časa za učenje.
- V Rocket League je zaradi dinamične fizike in hitrih sprememb v igrišču proces učenja zahteval preveč računske moči in je bil zato neučinkovit.

9.2 Zaključek in nadaljnje smeri raziskav

Rezultati so pokazali, da je UI sposobna samostojnega učenja in prilagajanja v enostavnejših okoljih, vendar pa so za uspešno igranje zahtevnejših iger potrebne dodatne optimizacije in večja računska moč.

Za izboljšanje zmogljivosti UI v prihodnosti priporočava:

- Uporabo naprednejših metod učenja, kot je model-based reinforcement learning za hitrejše prilagajanje strategij.
- Kombinacijo nevronskih mrež in tradicionalnih algoritmov iskanja poti, kar bi izboljšalo natančnost gibanja.
- Izrabo močnejše strojne opreme, ki bi omogočila krajši čas učenja in učinkovitejšo optimizacijo gibanja.

Z nadaljnjim razvojem bi lahko UI dosegla boljše rezultate tudi v igrah s kompleksno fiziko, kot sta Rocket League in Trackmania, ter tako postala še bolj učinkovita pri samostojnem učenju in prilagajanju igranju različnih računalniških iger.

11 ZAHVALA

Najprej bi se iskreno zahvalila najinima mentorjema Urošu Remenihu in Samu Železniku, ki sta nama s svojimi nasveti, znanjem in usmeritvami pomagala pri oblikovanju in izvedbi te raziskovalne naloge. Njuna podpora in strokovno vodstvo sta bila ključna pri premagovanju izzivov ter pri izboljšanju najinih raziskovalnih in programerskih veščin.

Posebna zahvala gre tudi najinima družinama, ki so nama skozi celoten proces raziskovanja nudile neprecenljivo podporo, potrpežljivost in motivacijo. Njihova vzpodbuda in razumevanje sta nama omogočila, da sva lahko raziskavi posvetila svoj čas in energijo.

Prav tako se zahvaljujeva najinim prijateljem, ki so naju spodbujali, verjeli v najin projekt ter nama s svojimi idejami in mnenji pomagali pri izboljšanju najine naloge. Njihova podpora nama je pomenila veliko in pripomogla k temu, da sva raziskavo uspešno zaključila.

12 PRILOGE

Kot prilogo prilagava:

- Programsko kodo 2D umetne inteligence,

```
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Ustvarjanje okolja
class Environment2D:
    def __init__(self, size):
        self.size = size
        self.grid = np.zeros((size, size))
        self.goal = (size - 1, size - 1)
        self.grid[self.goal] = 2  # Ciljna točka
        self.obstacles = self._add_obstacles()
        self.reset()

    def _add_obstacles(self):
        obstacles = []
        for _ in range(int(self.size * self.size * 0.2)):  # 20% mreže so
ovire
            x, y = random.randint(0, self.size - 1), random.randint(0,
self.size - 1)
            if (x, y) not in [self.goal, (0, 0)]:  # Izogibamo se začetne in
ciljne točke
                self.grid[x, y] = -1
                obstacles.append((x, y))
        return obstacles

    def reset(self):
        self.agent_pos = (0, 0)
        self.grid[self.agent_pos] = 1  # Začetna točka
        return self.agent_pos

    def step(self, action):
        # Akcije: 0 = gor, 1 = dol, 2 = levo, 3 = desno
        x, y = self.agent_pos
        if action == 0 and x > 0:  # gor
            x -= 1
        elif action == 1 and x < self.size - 1:  # dol
            x += 1
        elif action == 2 and y > 0:  # levo
            y -= 1
        elif action == 3 and y < self.size - 1:  # desno
```

```
            y += 1

        # Preverimo, če smo zadeli oviro
        if (x, y) in self.obstacles:
            return self.agent_pos, -1, False  # Ni spremembe položaja, kazen

        self.agent_pos = (x, y)
        reward = 1 if self.agent_pos == self.goal else -0.1
        done = self.agent_pos == self.goal
        return self.agent_pos, reward, done

# Q-Learning
class QLearningAgent:
    def __init__(self, size, actions, alpha=0.1, gamma=0.9, epsilon=0.1):
        self.q_table = np.zeros((size, size, actions))
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.actions = actions

    def choose_action(self, state):
        if random.uniform(0, 1) < self.epsilon:
            return random.randint(0, self.actions - 1)  # Naključna akcija
        x, y = state
        return np.argmax(self.q_table[x, y])  # Najboljša akcija

    def update_q(self, state, action, reward, next_state):
        x, y = state
        nx, ny = next_state
        best_next_action = np.argmax(self.q_table[nx, ny])
        td_target = reward + self.gamma * self.q_table[nx, ny,
best_next_action]
        td_error = td_target - self.q_table[x, y, action]
        self.q_table[x, y, action] += self.alpha * td_error

# Glavna zanka učenja
env = Environment2D(10)
agent = QLearningAgent(size=10, actions=4)

episodes = 5000
for episode in range(episodes):
    state = env.reset()
    total_reward = 0
    for _ in range(100):  # Omejeno število korakov
        action = agent.choose_action(state)
        next_state, reward, done = env.step(action)
        agent.update_q(state, action, reward, next_state)
```

```
        state = next_state
        total_reward += reward
        if done:
            break
    if episode % 100 == 0:
        print(f"Epizoda: {episode}, Skupna nagrada: {total_reward}")

# Priprava za simulacijo
state = env.reset()
path = [state]
done = False

while not done:
    action = agent.choose_action(state)
    state, _, done = env.step(action)
    path.append(state)

# Simulacija
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(env.grid, cmap='gray_r', origin='upper')

# Prikaz ovir (modre)
for obs in env.obstacles:
    ax.add_patch(plt.Rectangle((obs[1] - 0.5, obs[0] - 0.5), 1, 1,
color='blue'))

# Prikaz cilja (rdeče)
ax.add_patch(plt.Rectangle((env.goal[1] - 0.5, env.goal[0] - 0.5), 1, 1,
color='red'))

agent_marker, = ax.plot([], [], 'o', color='green', label="Agent")

def update(frame):
    x, y = path[frame]
    agent_marker.set_data([y], [x])  # Popravljen argument za set_data
    return agent_marker,

ani = FuncAnimation(fig, update, frames=len(path), interval=500, blit=True)

plt.title("Simulacija gibanja agenta")
plt.legend()
plt.grid(True)
plt.show()
```

- Programsko kodo 3D umetne inteligence,

```
import numpy as np
import random
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation

# 3D Environment
class Environment3D:
    def __init__(self, size):
        self.size = size
        self.grid = np.zeros((size, size, size))
        self.goal = (size - 1, size - 1, size - 1)
        self.grid[self.goal] = 2  # Goal point
        self.obstacles = self._add_obstacles()
        self.reset()

    def _add_obstacles(self):
        obstacles = []
        for _ in range(int(self.size ** 3 * 0.2)):  # 20% of the grid are
obstacles
            x, y, z = (
                random.randint(0, self.size - 1),
                random.randint(0, self.size - 1),
                random.randint(0, self.size - 1),
            )
            if (x, y, z) not in [self.goal, (0, 0, 0)]:  # Avoid start and
goal points
                self.grid[x, y, z] = -1
                obstacles.append((x, y, z))
        return obstacles

    def reset(self):
        self.agent_pos = (0, 0, 0)
        self.grid[self.agent_pos] = 1  # Starting point
        return self.agent_pos

    def step(self, action):
        # Actions: 0 = up, 1 = down, 2 = left, 3 = right, 4 = forward, 5 =
backward
        x, y, z = self.agent_pos
        if action == 0 and x > 0:  # up
            x -= 1
        elif action == 1 and x < self.size - 1:  # down
            x += 1
        elif action == 2 and y > 0:  # left
            y -= 1
        elif action == 3 and y < self.size - 1:  # right
            y += 1
        elif action == 4 and z > 0:  # forward
```

```
            z -= 1
        elif action == 5 and z < self.size - 1:  # backward
            z += 1

        # Check if we hit an obstacle
        if (x, y, z) in self.obstacles:
            return self.agent_pos, -1, False  # No position change, penalty

        self.agent_pos = (x, y, z)
        reward = 1 if self.agent_pos == self.goal else -0.1
        done = self.agent_pos == self.goal
        return self.agent_pos, reward, done

# Q-Learning
class QLearningAgent:
    def __init__(self, size, actions, alpha=0.1, gamma=0.9, epsilon=0.1):
        self.q_table = np.zeros((size, size, size, actions))
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.actions = actions

    def choose_action(self, state):
        if random.uniform(0, 1) < self.epsilon:
            return random.randint(0, self.actions - 1)  # Random action
        x, y, z = state
        return np.argmax(self.q_table[x, y, z])  # Best action

    def update_q(self, state, action, reward, next_state):
        x, y, z = state
        nx, ny, nz = next_state
        best_next_action = np.argmax(self.q_table[nx, ny, nz])
        td_target = reward + self.gamma * self.q_table[nx, ny, nz,
best_next_action]
        td_error = td_target - self.q_table[x, y, z, action]
        self.q_table[x, y, z, action] += self.alpha * td_error

# Training Loop
env = Environment3D(10)
agent = QLearningAgent(size=10, actions=6)

episodes = 500000
for episode in range(episodes):
    state = env.reset()
    total_reward = 0
    for _ in range(200):  # Limited number of steps
        action = agent.choose_action(state)
```

```
        next_state, reward, done = env.step(action)
        agent.update_q(state, action, reward, next_state)
        state = next_state
        total_reward += reward
        if done:
            break
    if episode % 100 == 0:
        print(f"Episode: {episode}, Total Reward: {total_reward}")

# Prepare for simulation
state = env.reset()
path = [state]
done = False

while not done:
    action = agent.choose_action(state)
    state, _, done = env.step(action)
    path.append(state)

# Visualization
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')
ax.set_box_aspect([1, 1, 1])  # Equal aspect ratio

# Draw obstacles (blue cubes)
for obs in env.obstacles:
    ax.bar3d(obs[1] - 0.5, obs[0] - 0.5, obs[2] - 0.5, 1, 1, 1, color="blue",
alpha=0.7)

# Draw goal (red cube)
goal = env.goal
ax.bar3d(goal[1] - 0.5, goal[0] - 0.5, goal[2] - 0.5, 1, 1, 1, color="red",
alpha=1)

agent_marker, = ax.plot([], [], [], "o", color="green", label="Agent")

def update(frame):
    x, y, z = path[frame]
    agent_marker.set_data([y], [x])
    agent_marker.set_3d_properties([z])
    return agent_marker,

ani = FuncAnimation(fig, update, frames=len(path), interval=500, blit=True)

plt.title("3D Simulation of Agent Movement")
plt.legend()
plt.show()
```

➢ Programsko kodo umetne inteligence za igro Trackmania.

```
import psutil
import time
import subprocess
import numpy as np
import random
import cv2
import pyautogui
import pydirectinput
import json
import os
import pygetwindow as gw  # For window management
import signal  # For handling program termination

# Constants
GAME_PROCESS_NAME = "TrackMania"  # Update with actual game name
GAME_PATH = r"C:\Program Files
(x86)\Steam\steamapps\common\Trackmania\Trackmania.exe"  # Update with actual
path
EPISODES = 500000  # Number of training episodes
STUCK_THRESHOLD = 6  # Seconds for speed 0 condition
MAX_TIME_LIMIT = 300  # Maximum track completion time in seconds
Q_TABLE_FILE = "best_q_table.json"  # File to store the best Q-table

# Q-Learning Parameters
LEARNING_RATE = 0.1
DISCOUNT_FACTOR = 0.95
EPSILON = 1.0
EPSILON_DECAY = 0.99
EPSILON_MIN = 0.1
ACTIONS = ['up', 'left', 'right']  # Actions corresponding to the state-action
space
q_table = np.zeros((10, 10, len(ACTIONS)))  # Simplified state-action table

best_reward = -float('inf')  # Initialize best reward as negative infinity


def is_game_running():
    """Check if the game is running."""
    for proc in psutil.process_iter(['pid', 'name']):
        if GAME_PROCESS_NAME.lower() in proc.info['name'].lower():
            return True
```

```
    return False


def launch_game():
    """Launch the game."""
    subprocess.Popen(GAME_PATH)
    time.sleep(5)  # Adjust based on loading time


def focus_game_window():
    """Bring the game window to the foreground."""
    windows = gw.getWindowsWithTitle(GAME_PROCESS_NAME)
    if windows:
        try:
            windows[0].activate()
        except Exception as e:
            print(f"Error activating window: {e}")
    else:
        print(f"No window found with title: {GAME_PROCESS_NAME}.")


def perform_action(action, currently_held_keys):
    """Perform a game action by holding down a key."""
    actions = {'up': 'w', 'left': 'a', 'right': 'd'}
    key = actions.get(ACTIONS[action])

    # Release any previously held keys if different from the current action
    for held_key in currently_held_keys:
        if held_key != key:
            pydirectinput.keyUp(held_key)

    # Hold the new key down
    pydirectinput.keyDown(key)
    return [key]  # Return the currently held key list


def reset_environment(currently_held_keys):
    """Reset the game environment by pressing 'R' and releasing any held
keys."""
    for held_key in currently_held_keys:
        pydirectinput.keyUp(held_key)  # Ensure all keys are released
    pydirectinput.press('r')
    time.sleep(3)  # Allow the game to reset
    return []  # Return an empty list of held keys


def capture_game_state():
```

```
    """Capture and process the game screen for state estimation."""
    screenshot = pyautogui.screenshot()
    frame = np.array(screenshot)
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    position = random.randint(0, 9)  # Replace with actual detection logic
    velocity = random.randint(0, 9)  # Replace with actual detection logic
    wall_hit = detect_wall_collision(frame)
    off_track = detect_off_track(frame)
    finish_line = detect_finish(frame)

    return position, velocity, wall_hit, off_track, finish_line


def detect_wall_collision(frame):
    """Detect wall collisions."""
    wall_detected = False  # Implement logic based on game visuals
    return wall_detected


def detect_off_track(frame):
    """Detect if the car goes off track."""
    off_track_detected = False  # Implement logic based on game visuals
    return off_track_detected


def detect_finish(frame):
    """Detect if the car has reached the finish line."""
    finish_detected = False  # Implement logic based on game visuals
    return finish_detected


def get_reward(wall_hit, off_track, velocity, new_velocity):
    """Calculate a reward based on game state."""
    reward = 0
    if wall_hit:
        reward -= 50
    if off_track:
        reward -= 100
    if new_velocity > velocity:
        reward += 10
    if velocity > 0 and not wall_hit and not off_track:
        reward += 1
    return reward


def save_q_table(reward):
```

```
    """Save the Q-table to a file."""
    global best_reward
    if reward > best_reward:
        best_reward = reward
        with open(Q_TABLE_FILE, 'w') as file:
            json.dump(q_table.tolist(), file)
        print(f"Q-table saved with reward {best_reward}.")


def load_q_table():
    """Load the Q-table from a file."""
    global q_table
    if os.path.exists(Q_TABLE_FILE):
        with open(Q_TABLE_FILE, 'r') as file:
            q_table = np.array(json.load(file))
        print("Q-table loaded.")
    else:
        print("No previous Q-table found, starting fresh.")


def handle_exit_signal(signum, frame):
    """Handle program termination signals to save the Q-table."""
    global best_reward
    save_q_table(best_reward)
    print(f"Program terminated (signal {signum}). Q-table saved.")
    exit(0)


def train_agent(episodes=500000):
    """Training loop for the AI agent."""
    global EPSILON
    for episode in range(episodes):
        position, velocity, wall_hit, off_track, finish_line =
capture_game_state()
        done = False
        start_time = time.time()
        zero_speed_start_time = None  # Track when the car first stops moving
        currently_held_keys = []  # Keep track of keys being held down
        reward = 0  # Initialize reward for each episode

        while not done:
            if random.uniform(0, 1) < EPSILON:
                action = random.randint(0, len(ACTIONS) - 1)  # Random action
            else:
                action = np.argmax(q_table[position, velocity])  # Exploit Q-
table
```

```
            currently_held_keys = perform_action(action, currently_held_keys)
            new_position, new_velocity, wall_hit, off_track, finish_line =
capture_game_state()

            if finish_line:
                print("Finish line reached!")
                time.sleep(2)
                currently_held_keys = reset_environment(currently_held_keys)
                done = True
                continue

            if new_velocity == 0:
                if zero_speed_start_time is None:
                    zero_speed_start_time = time.time()
                elif time.time() - zero_speed_start_time > STUCK_THRESHOLD:
                    print("Car stuck. Resetting environment.")
                    currently_held_keys =
reset_environment(currently_held_keys)
                    done = True
                    zero_speed_start_time = None
                else:
                    zero_speed_start_time = None

            reward = get_reward(wall_hit, off_track, velocity, new_velocity)
            old_value = q_table[position, velocity, action]
            next_max = np.max(q_table[new_position, new_velocity])
            q_table[position, velocity, action] = old_value + LEARNING_RATE *
(reward + DISCOUNT_FACTOR * next_max - old_value)

            position, velocity = new_position, new_velocity

            if wall_hit or off_track or time.time() - start_time >
MAX_TIME_LIMIT:
                print("Episode reset due to wall hit, off-track, or time
limit.")
                currently_held_keys = reset_environment(currently_held_keys)
                done = True

        EPSILON = max(EPSILON_MIN, EPSILON * EPSILON_DECAY)
        save_q_table(reward)


# Signal handling for graceful exit
signal.signal(signal.SIGINT, handle_exit_signal)

# Start the game and training
if not is_game_running():
```

```
    launch_game()
focus_game_window()
load_q_table()
train_agent(EPISODES)
```