SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6660

SIMULACIJA EVAKUACIJE AGENATA

Tin Blažević

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6660

SIMULACIJA EVAKUACIJE AGENATA

Tin Blažević

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 6660

Pristupnik:	**Tin Blažević (0036507355)**
Studij:	Računarstvo
Modul:	Računarska znanost
Mentor:	prof. dr. sc. Željka Mihajlović

Zadatak: **Simulacija evakuacije agenata**

Opis zadatka:

Proučiti modele više-agentskih sustava te algoritme strojnog učenja i mogućnosti njihove primjene na više-agentskom modelu. Razraditi problem evakuacije agenata s lokalnom percepcijom okoliša u interijerima. Razraditi simulaciju kretanja agenata te ostvariti prikaz različitih scenarija. Diskutirati utjecaj različitih parametara. Načiniti ocjenu rezultata i implementiranih algoritama. Izraditi odgovarajući programski proizvod. Koristiti grafički programski pogon Unity te dodatak Unity ML-Agents Toolkit. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 12. lipnja 2020.

SADRŽAJ

1. Uvod

U današnje vrijeme značajna sredstva ulažu se u razvoj različitih područja umjetne inteligencije. Aktivno područje istraživanja je savladavanje trodimenzionalnog prostora i fizičkih prepreka. Neke nesreće, zbog velikih rizika po zdravlje pružatelja pomoći, ispostavljaju se kao velik izazov za brzu i efektivnu ljudsku reakciju. Jedan od ciljeva je razvijanje agenata koji su sposobni izvršavati složene akcije u opasnim okolišima, pritom koristeći dostupne alate, od ručnih alata do vozila [1]. Velike pomake donosi tvrtka Boston Dynamics sa svojim humanoidnim i četveronožnim robotima, od kojih neke financira DARPA [2]. Razvoj takvih sustava efikasniji je uz prethodno provedene simulacije u virtualnom okolišu.

U središtu ovog rada je proučavanje metoda umjetne inteligencije pogodnih za jednostavan zadatak savladavanja različitih prepreka u virtualnoj sceni na putu do cilja. Za uspješno savladavanje scena sa složenim prostornim rasporedom dodatno je potrebna određena razina suradnje. Implementirana je interaktivna simulacija kroz platformu Unity koristeći dva različita pristupa. Jedan pristup je baziran na neuronskim mrežama i dubokom potpornom učenju, a drugi je baziran na simboličkom pristupu i automatskom planiranju.

2. Korištene tehnologije i alati

2.1. Unity

Unity je platforma za razvoj 3D aplikacija koje rade u realnom vremenu. Radi se o alatu koji ima mnoštvo ugrađenih sustava i popratnih paketa koji olakšavaju i ubrzavaju razvoj. Kao jezik za pisanje skripti koristi se jezik C#. Baziran je na intuitivnoj Entitet-komponenta-sustav razvojnoj arhitekturi. Zbog toga, Unity se nameće kao idealan izbor za implementaciju sustava umjetne inteligencije obrađenih u ovom radu.

2.2. ML-Agents

Unity ML-Agents projekt je otvorenog koda koji omogućava da igre i simulacije služe kao okoliš za treniranje inteligentnih agenata [3]. Potpuni izvorni kod projekta dostupan je na githubu [9]. Agente je moguće trenirati potpornim učenjem, imitacijskim učenjem i neuroevolucijom kroz jednostavno programsko sučelje. Dane su implementacije aktualnih algoritama u području, bazirane na *TensorFlowu*. Trenirani agenti mogu se koristiti kao računalom upravljani likovi u igrama, za automatsko testiranje igara, kao i za evaluaciju različitih mogućnosti u dizajnu igre.

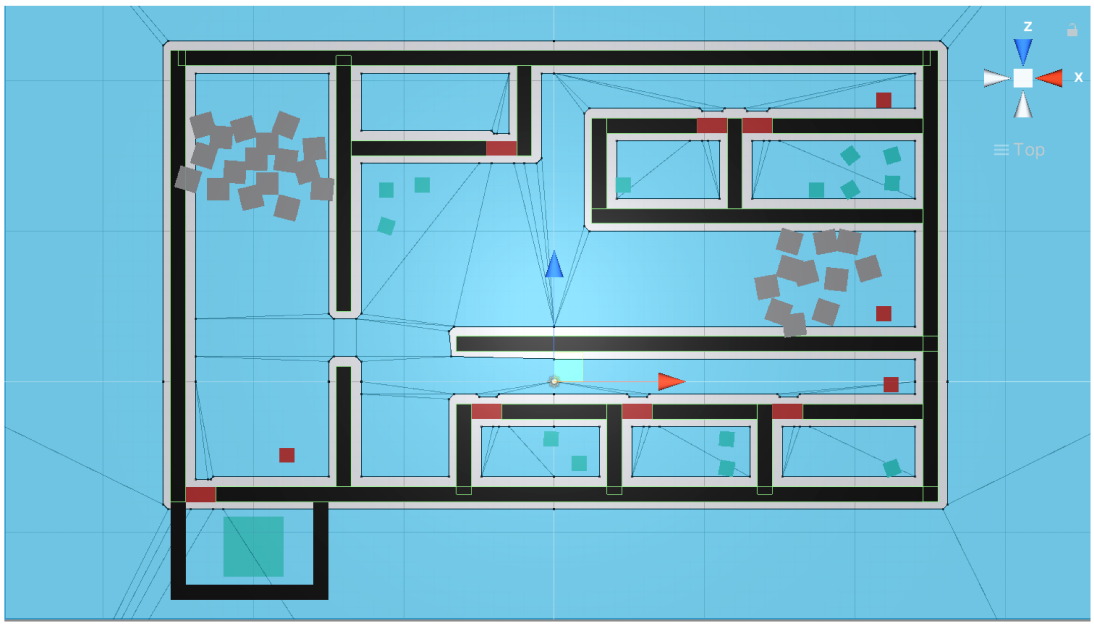
2.3. NavMesh komponente visoke razine

Unity ima ugrađen NavMesh sustav koji iz geometrije u sceni generira navigacijsku mrežu po kojoj se virtualni likovi mogu kretati. Dodatne komponente dostupne su na githubu [10].

Navigacijske mreže su efikasnije od klasične cjelobrojne rešetke. Prilikom generiranja obrađuju se elementi volumena u kojemu se nalazi označena geometrija. Mreža se sastoji od 3D poligona i bridova koji su generirani iz geometrije označenih objekata. Zbog ovih svojstava navigacijske mreže omogućavaju istinsku reprezentaciju 3D prostora. Za pretraživanje prostora stanja prilikom traženja puta najčešće se koristi nekakva varijanta A* algoritma.

Na slici 2.1 primjer je generirane navigacijske mreže, označen plavom bojom. Moguće je jasno vidjeti bridove i poligone navigacijske mreže. Prilikom pokretanja scene, poligoni mijenjaju svjetlinu boje ovisno o cijeni puta kada bi agent njima kročio. Neki objekti su dio prohodne geometrije, a neki nisu. Pojedini objekti koji imaju ulogu prepreke imaju komponentu *NavMeshObstacle* koja unutar navigacijske mreže dinamički stvara šupljine primitivnih oblika kao što su kutija i kapsula.

Za kretanje po navigacijskoj mreži služi komponenta *NavMeshAgent*. Komponenta traži put po navigacijskoj mreži i sprema ga u obliku niza 3D kontrolnih točaka koje služe kao vodilja. U računanje puta ugrađeno je izbjegavanje drugih agenata čime se postiže efikasniji protok. Za slučaj da se putovi više agenata sijeku, gleda se pojedinačni prioritet pri izbjegavanju - agenti s većim prioritetom imaju prednost, a oni s manjim usporavaju i propuštaju prioritetnije. U slučaju da svi agenti imaju jednak prioritet, moguć je zastoj. Problem se rješava tako da se na početku svakom agentu dodijeli nasumični prioritet iz intervala. Prevelik interval rezultira prevelikim razlikama u prioritetu, čime neki manje prioritetni agenti previše usporavaju i popuštaju zbog čega se može dogoditi da ga ostali odguraju s puta.



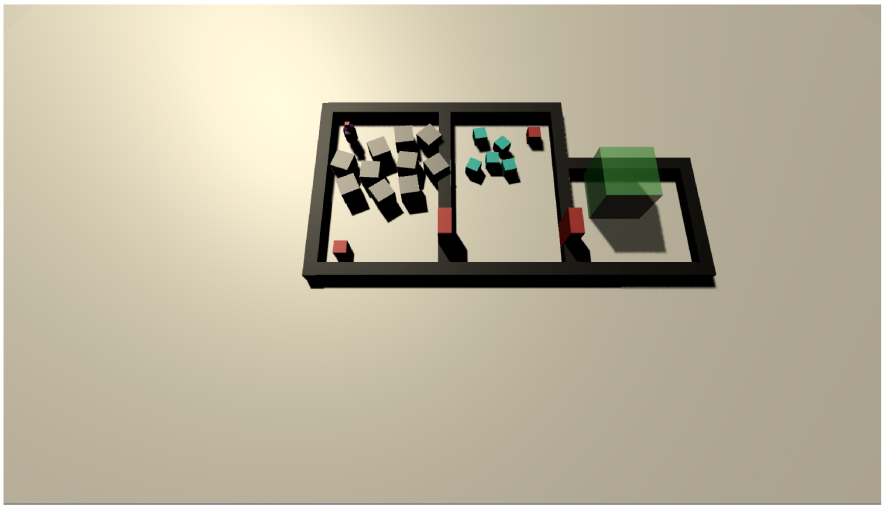
Slika 2.1: Navigacijska mreža (plavo)

3. Virtualni okoliš

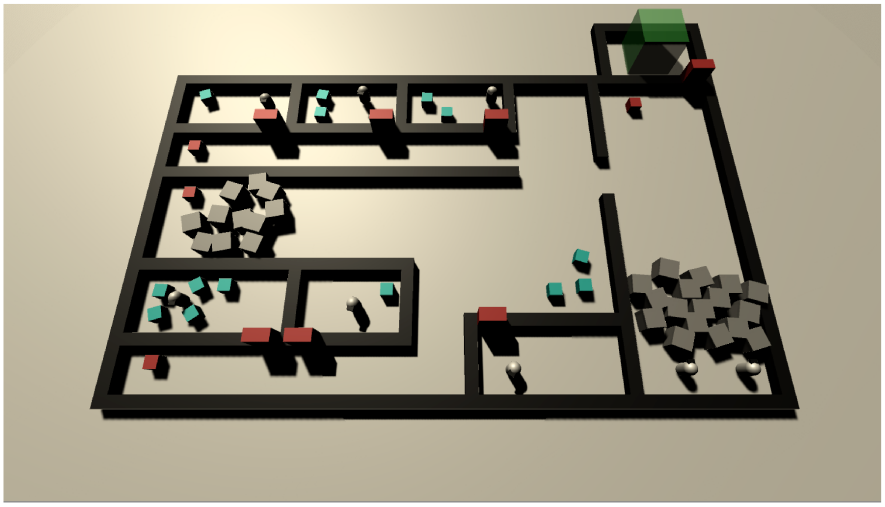
Okoliš u kojem se agenti nalaze sastoji se od skaliranih 3D geometrijskih primitiva ugrađenih u Unity koje imaju različite uloge. Zidovi su kutije crne boje i služe kao granice prostorija. Zidovi ulaze u geometriju iz koje se generira navigacijska mreža, osim onih oko cilja, koji služe kako se ML agenti ne bi previše udaljili. Osim zidova, u navigacijsku mrežu ulazi i ravnina koja predstavlja tlo. Prepreke su kutije sive boje i služe kao smetnja prilikom kretanja. Vrata su visoke kutije crvene boje koje odvajaju pojedine prostorije i kroz njih je moguće proći samo ako su otključana, a za to je potreban ključ - manja kutija crvene boje. Vrata imaju komponentu *NavMeshObstacle*. Poželjno je da agenti skupljaju bitne predmete - kutije svijetlo plave boje. Cilj je označen velikom zelenom poluprozirnom kutijom. Svi objekti imaju sudarače (eng. *Collider*), osim cilja, koji ima okidač (eng. *Trigger*).

Na slici 3.1a primjer je jednostavne scene koja služi za evaluaciju jednostavnih akcija i planova. Na slici 3.1b primjer je složene scene u kojoj se nalazi više agenata i potrebna je suradnja te napredno snalaženje u prostoru za uspješno savladavanje.

Agenti se sastoje od kapsule kao tijela i kocke kao očiju, kako bi bio vidljiv smjer u kojemu su okrenuti. Kada agent pokupi ključ, iznad kapsule aktivira se jednostavni indikator u obliku crvene kocke koji označava da agent sada ima ključ, kako bi se vizualno razlikovao od onoga bez ključa.



(a) Jednostavna scena



(b) Složena scena

Slika 3.1

4. Strojno učenje i ML-Agents

4.1. Implementacija ML-Agents sustava

Na github stranicama projekta dostupna je opsežna dokumentacija, uz popratne jednostavne primjere [9]. Implementacija se svodi na stvaranje agenta, područja u kojem se kreće, stvaranje konfiguracijske datoteke i treninga.

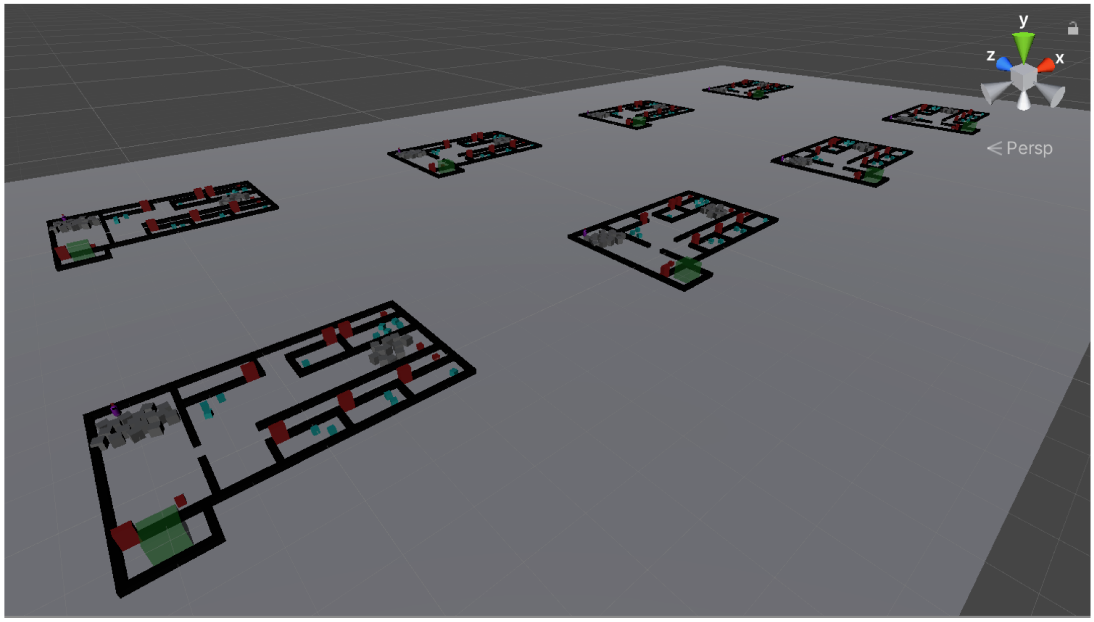
4.1.1. Područje

Područje sadrži popise objekata koji su sadržani unutar njega, kako bi im moglo mijenjati stanje prilikom početka nove epizode. Osim toga, područje odgovara na upite agenata o tome koji su mu objekti najbliži i u kojem smjeru je cilj. Na ovaj način, gdje područje automatski skuplja reference samo na objekte koji su unutar njega, a agenti ne mogu izaći izvan granica, moguće je raditi različite varijacije rasporeda objekata unutar područja, a u samoj Unity sceni moguće je imati više instanci različitih područja u kojima agenti mogu paralelno trenirati, kao što je prikazano na slici 4.1, što bitno ubrzava trening.

4.1.2. Agent

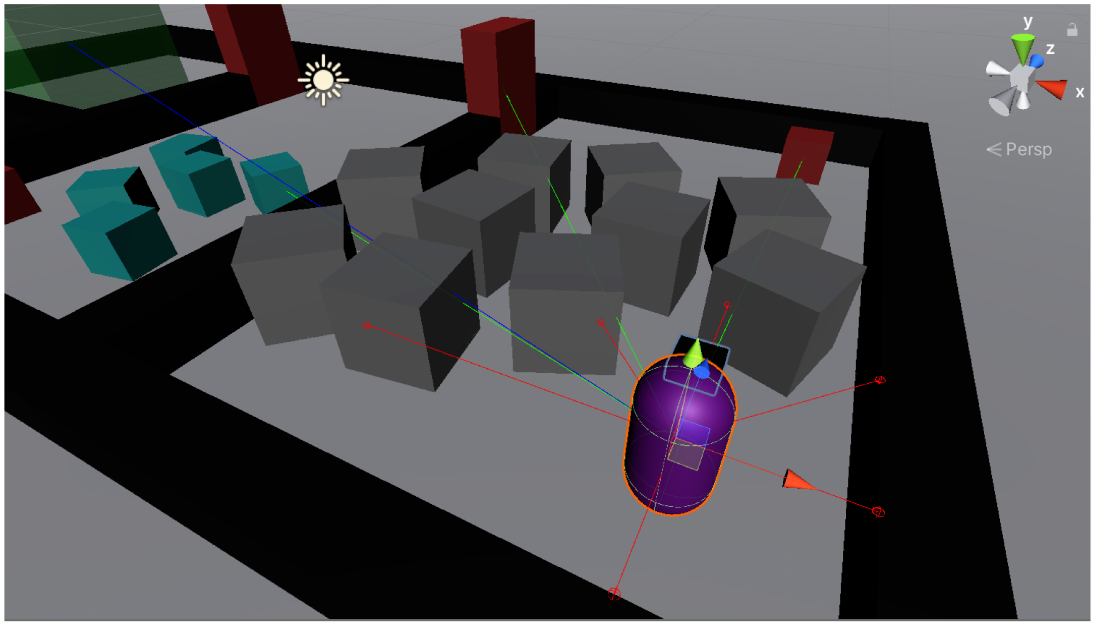
Razred konkretnog agenta nasljeđuje razred apstraktnog agenta danog u paketu. Konkretni agent nadjačava dostupne metode kojima se implementiraju njegove moguće akcije, nagrada, skupljanje opažanja sa senzora, završetak epizode i slično. Bitne metode su:

- *void Initialize()* - zamjena za često korištenu Unity *Start()* metodu u kojoj se postavljaju reference na komponente, dodatno se postavljaju reference na senzorske komponente
- *void OnEpisodeBegin()* - resetira stanje agenta na početku svake epizode



Slika 4.1: Područja za istovremeno treniranje više agenata

- *void OnActionReceived(float[] vectorAction)* - definira akcije koje agent izvršava na temelju izlaza modela ili heuristike koji je spremljen u vektoru akcija. Ovdje konkretno agent koristi sile za kretanje u prostoru, a osim toga može ukloniti prepreku, pokupiti predmet, pokupiti ključ ali samo jednom i otključavati vrata ali samo ako ima ključ. Za interakciju s okolinom agent dobiva određenu nagradu. Vrijednosti nagrade koje agent dobiva trebale bi biti po apsolutnoj vrijednosti manje ili jednake 1 zbog stabilnosti treninga
- *void CollectObservations(VectorSensor sensor)* - popunjava vektor opažanja proizvoljnim informacijama. Ovdje konkretno sprema trenutno stanje agenta, smjer, normaliziranu udaljenost i kosinus kuta između agenta i promatranog objekta, a to su cilj, najbliži ključ, najbliža vrata i najbliži predmet. Jednom kada agent pokupi ključ, više ne prima informacije o najbližem ključu nego se taj dio vektora popunjava nulama. Općenito, vrijednosti opažanja bi trebale biti u rasponu od -1 do 1 (normalizirane) jer algoritmi tako brže uče i trening je stabilniji
- *void Heuristic(float[] actionsOut)* - definira način na koji se popunjava vektor akcija, najčešće se mapira ulaz s miša i tipkovnice, za potrebe testiranja ili skupljanja demonstracija od ljudskog igrača u slučaju imitacijskog učenja



Slika 4.2: Senzori agenta

Agent periodički traži od područja u kojem obitava informacije o najbližim vratima, ključu i bitnom predmetu. U slučaju da agent traži te informacije svaki put kada prikuplja opažanja, može doći do pada performansi ako u području postoji mnogo agenata ili mnogo objekata jer se pri svakom pozivu popisi sortiraju po udaljenosti od agenta. Osim tih opažanja, agent ima dvije komponente *RayPerceptionSensor3D* koje služe za promatranje okolnog prostora. Prednji senzor ima 5 zraka, a stražnji jednu. Prevelik broj zraka može prouzročiti pad performansi i može usporiti trening. Oba senzora moraju imati popis oznaka (eng. *tags*) koje mogu opaziti za razlikovanje tipova objekata koje zrake pogode. Opažanja koja ti senzori generiraju su u obliku normalizirane udaljenosti od pogođenog objekta i tipa objekta u zapisu vrućom jedinicom. Na slici 4.2 prikazana su opažanja senzora (crveno) i vektori prema cilju (plavo), najbližem ključu, vratima i bitnom predmetu (zeleno).

Na kraju se na objekt koji na sebi ima komponentu s implementacijom konkretnog agenta mora dodati komponenta *DecisionRequester* bez koje agent neće funkcionirati. Kroz tu komponentu definira se broj koraka nakon kojeg se traži novi vektor opažanja.

4.2. Treniranje i iteriranje

Konfiguracijska *.yaml* datoteka definira hiperparametre algoritma koji se koristi za trening i parametre generiranog modela. Dostupna su dva algoritma - PPO (eng. *Proximal Policy Optimization*) i SAC (eng. *Soft-Actor Critic*). Usporedbe algoritama dane su u literaturi [3]. SAC algoritam ima bolje performanse od algoritma PPO samo u specifičnim slučajevima, pa je ovdje korišten algoritam PPO. Neki generirani modeli su obične neuronske mreže, a neke su neuronske mreže s povratnom vezom, kako bi model imao pamćenje.

Za pokretanje treninga potrebno je preuzeti *python* paket *mlagents* koristeći *pip*. Trening se pokreće iz naredbenog retka naredbom *mlagents-learn*. Jednom pokrenuti trening moguće je prekinuti. Stanje prekinutog treninga serijalizira se i sprema, kako bi se kasnije moglo nastaviti. Time je moguće usred treninga napraviti promjene u okolišu, u funkciji nagrade ili pak u nekom dijelu programskog koda.

Alat dolazi s podrškom za *TensorBoard* pomoću kojeg je moguće tijekom treninga pratiti performanse modela. Tijekom treninga periodički se generiraju kontrolne točke i sažetci iz kojih se generiraju grafovi. Kumulativna nagrada pokazuje dobrotu naučene strategije. Moguće je prije treninga na temelju funkcije nagrade i rasporeda objekata u sceni pretpostaviti koliko bi kumulativna nagrada trebala iznositi za dobru strategiju. Trajanje epizode je broj koraka koliko je agentu bilo potrebno da završi epizodu. Ako je on manji od maksimalnog broja koraka, agent je uspio doći do cilja.

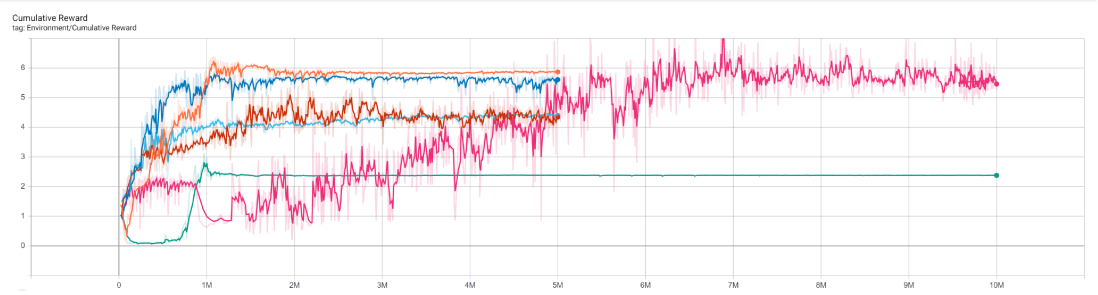
Na slici 4.3 nalaze se grafovi kumulativne nagrade i trajanja epizode svih modela. Takav združeni graf nije reprezentativan u odnosu na pravo stanje jer su pojedini modeli imali nešto drugačije funkcije nagrade i nisu imali isti raspored područja za treniranje. Zato grafove treba razmatrati odvojeno.

Generirano je ukupno 6 modela:

1. model obične neuronske mreže s 2 skrivena sloja po 256 neurona, treniran na jednostavnom području u 5 milijuna epizoda (na slici svijetlo plava)

2. model obične neuronske mreže s 2 skrivena sloja po 512 neurona, treniran na jednostavnom području u 5 milijuna epizoda (narančasta)

3. model obične neuronske mreže s 2 skrivena sloja po 512 neurona, treniran na kombinaciji jednostavnih i složenih područja u 5 milijuna epizoda (tamno plava)



(a) Graf kumulativne nagrade

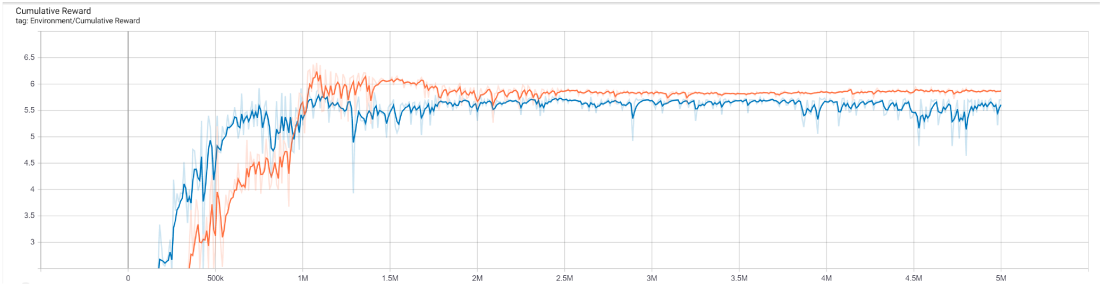
(b) Graf trajanja epizode

Slika 4.3: Grafovi svih modela

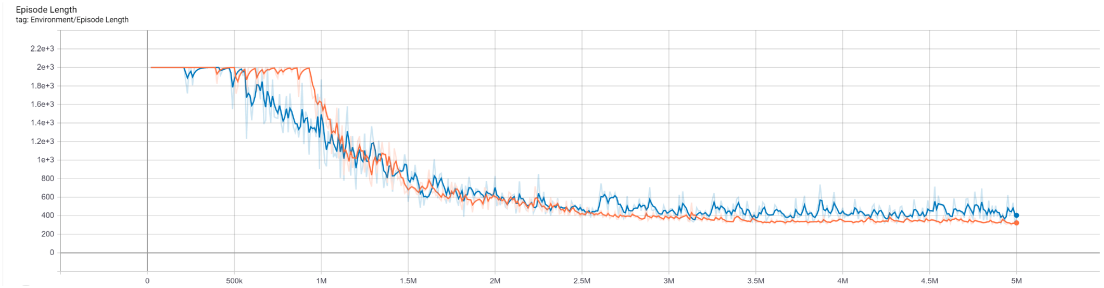
4. model neuronske mreže s povratnom vezom, 2 skrivena sloja po 512 neurona, treniran na kombinaciji jednostavnih i složenih područja u 5 milijuna epizoda (smeđa)

5. model neuronske mreže s povratnom vezom, 2 skrivena sloja po 512 neurona, treniran na složenom području u 10 milijuna epizoda, s povećanim maksimalnim brojem koraka (zelena)

6. model neuronske mreže s povratnom vezom, 2 skrivena sloja po 512 neurona, treniran na kombinaciji jednostavnih i složenih područja u 10 milijuna epizoda, s povećanim maksimalnim brojem koraka, scena je bila modificirana za vrijeme treninga (ružičasta)



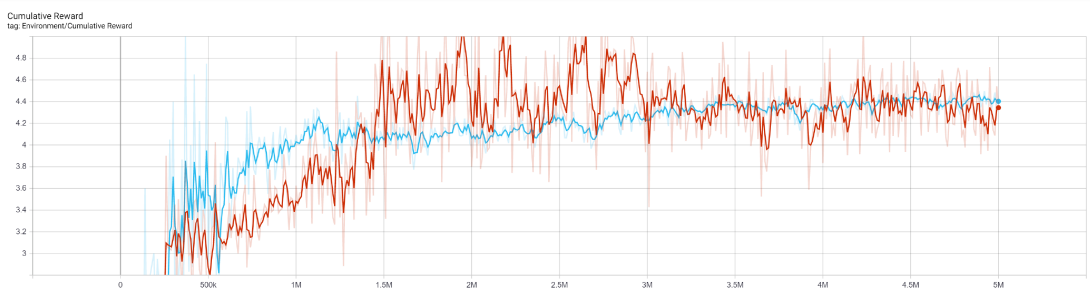
(a) Graf kumulativne nagrade



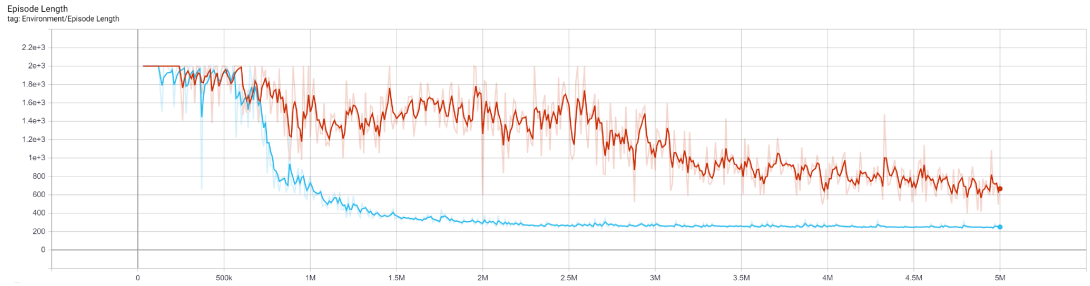
(b) Graf trajanja epizode

Slika 4.4: Grafovi modela 1 (plavo) i 2 (narančasto)

Usporedba prva dva modela, treniranih na jednostavnoj sceni, na slici 4.4 prikazuje učinak broja neurona u skrivenim slojevima. Neuronska mreža s manje neurona brže je naučila strategiju, ali konačna strategija ima manju nagradu i veći broj koraka od neuronske mreže s više neurona. Uvidom u dodatne grafove moguće je zaključiti i da neuronska mreža s više neurona ima stabilniji trening. Ovo međutim ne znači da treba pretjerivati s brojem neurona.



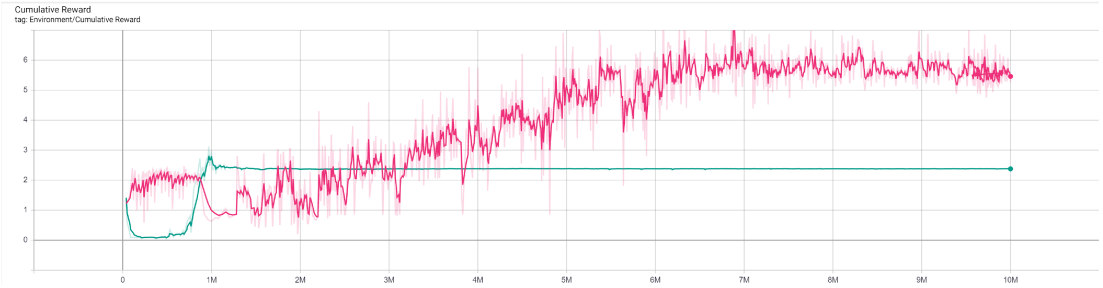
(a) Graf kumulativne nagrade



(b) Graf trajanja epizode

Slika 4.5: Grafovi modela 3 (plavo) i 4 (smeđe)

Nakon testa na jednostavnoj sceni, napravljen je trening na složenoj sceni. Korištena je kombinacija područja s jednostavnom i složenom scenom kao pokušaj generalizacije na obje scene. Kako agenti paralelno treniraju, ovo je rezultiralo time da je veći broj epizoda završen na jednostavnoj sceni, jer je potreban manji broj koraka za postizanje cilja. Zbog toga je agent prikupljao više iskustva na jednostavnoj sceni nego na složenoj što se odrazilo na njegovo ponašanje. Četvrti model i oni nakon njega imaju neuronsku mrežu s povratnom vezom, kako bi agent imao mogućnost pamćenja prostornog rasporeda. Oba modela prikazana na slici 4.5 naučila su sličnu strategiju, a to je zgrabiti ključ najbliže cilju, otvoriti vrata najbliže cilju i pobjeći, ne uzimajući u obzir preostala vrata i predmete kojima je moguće ostvariti dodatnu nagradu.



(a) Graf kumulativne nagrade

(b) Graf trajanja epizode

Slika 4.6: Grafovi modela 5 (zeleno) i 6 (ružičasto)

Činilo se da je za složenu scenu nedovoljan maksimalni broj koraka pa je povećan. Nakon 5 milijuna epizoda, neuronska mreža s povratnom vezom nije imala stabilnu strategiju pa je maksimalni broj epizoda povećan na 10 milijuna. Možda nagrada nije bila dovoljno primamljiva pa je funkcija nagrade izmijenjena tako da je nagrada od dostizanja cilja bitno manja u odnosu na otključavanje vrata i skupljanje predmeta. Rezultat je bio isti kao i bez promjena. Zadnji model je treniran na kombinaciji jednostavnih i složenih područja kako bi ga usmjerilo iskustvo skupljeno na jednostavnim područjima. Također, nakon milijun epizoda, trening je prekinut kako bi bili deaktivirani ključ najbliže cilju i jednostavna područja, da bi na kraju ostala samo složena područja. U nastavku treninga, model je pokazivao da očekuje ključ na starom mjestu, ali je postepeno počeo ići prema ostalima. Nakon nekog vremena počeo je kupiti jedan od ostalih ključeva i vraćati se prema cilju, pri tome otključavajući samo vrata i skupljajući samo predmete koji su bili usput. Rezultati su prikazani na slici 4.6.

4.3. Rezultati i moguća poboljšanja

Svi modeli su patili od problema priljubljivanja uza zid i nemogućnosti odvajanja. Trebalo bi implementirati negativnu nagradu ako je agent preblizu zida. Problem s negativnom nagradom je što agent može prestati izvršavati određene akcije i strategije koje su bile korisne.

Modeli s pamćenjem su razvili strategiju konstantnog rotiranja i bili su sposobni vratiti se u početnu prostoriju i dostići cilj većinu vremena dok je trajao trening. Međutim, agenti se čine zbunjeni kada ih u području ima više prilikom realne situacije jer je za vrijeme treninga vrijedilo: jedno područje - jedan agent. Potrebno je implementirati način kojim bi više agenata moglo istovremeno trenirati na jednom području.

Čini se da postepeno otežavanje prostornog rasporeda područja ima pozitivan učinak na naučenu strategiju. Trebalo bi napraviti mnoštvo područja različitih težina i zatim ih tijekom treninga postepeno aktivirati kako agent napreduje.

Možda je način na koji se agenti gibaju, korištenjem sile, previše zahtjevan za naučiti. Trebalo bi isprobati nekakav manje fizikalno točan, ali i čovjeku intuitivniji način gibanja kakav se inače koristi u igrama.

5. Automatsko planiranje i GOAP

5.1. STRIPS i GOAP

STRIPS je sustav za automatsko planiranje i raspoređivanje razvijen na sveučilištu Stanford 1970-ih, skraćeno od STanford Research Institute Problem Solver. STRIPS je postao naziv za općeniti sustav za automatsko planiranje koji se sastoji od skupa simbola, početnog stanja, konačnih stanja (ciljeva) i akcija. Ciljevi su skup stanja svijeta koje treba postići iz početnog korištenjem akcija. Akcije su definirane kao par - skup preduvjeta i skup posljedica. Akciju je moguće izvršiti samo ako su u trenutnom stanju svijeta zadovoljeni svi njeni preduvjeti, a izvršavanje akcije u stanje svijeta ugrađuje posljedice.

GOAP (eng. *Goal Oriented Action Planning*) varijanta je STRIPS sustava prilagođen za primjenu u realnom vremenu. Razlika je prvenstveno u uvođenju cijene akcije, a popis svih razlika nalazi se na [7]. Time je moguće primijeniti A* algoritam na pretraživanje prostora stanja, gdje je heuristička funkcija stanja jednaka zbroju cijena nezadovoljenih ciljeva. Rezultat je optimalan slijed akcija za ostvarivanje cilja. Arhitekturu je ranih 2000-ih razvio Jeff Orkin za potrebe igre F.E.A.R [8] [4] [5] [6]. Od tada je arhitektura postala opće prihvaćena i primijenjena je u igrama različitih žanrova, od brzih akcijskih igara u prvom licu, do strategija i raznih simulacija. U odnosu na klasične konačne automate, prijelazi se ne programiraju eksplicitno, nego postoje implicitno kroz korištenje preduvjeta i posljedica. Jedan mogući način gledanja na GOAP sustav je kao generator jednostavnih konačnih automata za određeni cilj. Druga velika prednost je visoka ponovna upotrebljivost koda. Za primjenu na drugom projektu, planer nije potrebno mijenjati, već je samo potrebno prilagoditi skup akcija.

5.2. Implementacija GOAP sustava

GOAP agent ima skup ciljeva, plan kojeg prati, trenutnu akciju koju izvršava i inventar. Konkretni agent može imati različite ciljeve i dostupne akcije. Znanje o svijetu je zapisano u poseban razred koji osim simbola ima i globalni popis resursa. Simboli su parovi ključ-vrijednost. Resursi su reference na predmete u sceni koji mogu biti meta akcije. Ciljevi su zbog jednostavnosti konjunkcija simbola, imaju definiran prioritet i mogu biti ponavljajući ili jednokratni. Akcije imaju konjunkciju preduvjeta i konjunkciju posljedica. Svaka akcija ima 3 faze: kretanje prema meti akcije, dolazak u domet za izvršavanje akcije i izvršavanje akcije. Prije prve faze, akcija traži najbliži resurs do kojeg postoji put i postavlja ga kao metu. U svakoj fazi potrebno je provjeravati valjanost akcije. Ako akcija prestane biti valjana, agent poništava plan i traži novi. Za valjanost akcije provjerava se brojčana vrijednost simbola iz stanja svijeta ili lokalnog stanja agenta (uvjerenja), kao i postojanje odgovarajućeg resursa u globalnom ili lokalnom inventaru. Za kretanje prema meti akcije koristi se Unity NavMesh sustav opisan u poglavlju 2.3. Konkretna akcija definira promjene u stanju svijeta i promjene lokalnog stanja agenta. Te promjene uključuju mijenjanje brojčane vrijednosti simbola i prebacivanje resursa iz globalnog u lokalni inventar ili obrnuto. U slučaju da je akcija prekinuta, sve promjene koje je napravila treba poništiti kako resursi ne bi ostali zaključani i nedostupni drugim agentima. Agenti traže planove - niz akcija koji ih dovodi do cilja, od planera. Planovi se grade iterativno, za najprioritetniji cilj koji je moguće ostvariti. Planer je jednostavan objekt koji prima skup dostupnih akcija agenta, popis ciljeva i uniju stanja svijeta i lokalnih stanja agenta i na temelju tih informacija vraća plan pretraživanjem prostora stanja. Za potrebe ovog rada dovoljno je bilo iscrpno pretraživanje kao algoritam pretraživanja prostora stanja. Generiraju se svi mogući planovi i odabire se onaj s najnižom cijenom.

5.2.1. Konkretni ciljevi i akcije

Konkretni agent ima 5 ciljeva:

- *pobjegao* - jednokratni cilj najnižeg prioriteta, zadovoljen kada agent stigne do cilja
- *prepriječenPut* - ponavljajući cilj najvećeg prioriteta. Poseban senzor mijenja vrijednost simbola zapisanih u lokalno znanje agenta i prekida trenutnu akciju
- *pokupitiPredmet* - ponavljajući cilj prioriteta nižeg od prioriteta cilja *otključatiVrata*
- *imaKljuč* - jednokratni cilj prioriteta većeg od prioriteta cilja *otključatiVrata*
- *otključatiVrata* - ponavljajući cilj

Konkretne akcije koje su dostupne agentu su:

- *PokupiPredmet* - akcija s preduvjetom da postoji predmet i posljedicom da postoji jedan predmet manje
- *PokupiKljuč* - akcija s preduvjetima da postoji ključ, postoje vrata koja treba otključati i agent nema ključ. Posljedice: agent ima ključ i postoji jedan ključ manje
- *OtključajVrata* - akcija s preduvjetima da postoje vrata koja treba otključati i agent ima ključ. Posljedice: otključana vrata i postoje jedna vrata manje
- *UkloniPrepreku* - akcija s preduvjetom da je agentu prepriječen put i posljedicom da je prepreka uklonjena
- *IdiPremaCilju* - akcija bez preduvjeta koja zadovoljava cilj *pobjegao*

Uz ovakav raspored ciljeva i akcija, agenti mogu riješiti bilo koju scenu. Agenti će prije svega uklanjati prepreke s puta. Osigurano je da agenti prije svega ostaloga pokupe ključ i otključavaju vrata, čime mogu pomoći agentima koji su zarobljeni u prostoriji bez ključa. Nakon što u sceni nema vrata, agenti tek onda prikupljaju bitne predmete. Na samom kraju, agenti se upućuju prema cilju.

5.3. Rezultati i moguća poboljšanja

GOAP agenti sposobni su riješiti bilo kakvu scenu koja je rješiva, ali postoji mnoštvo prostora za poboljšanja i proširenje.

Moguće je primijetiti da neki agenti idu prema objektima koji im nisu nužno najbliži. To se događa jer su u međuvremenu drugi agenti zaključali taj resurs, koji je njima bio najbliži, iako je ukupni prijeđeni put u tom slučaju veći.

Agenti koji su zarobljeni u sobama bez ključa, konstantno zahtijevaju plan koji će ih dovesti do ključa ako on postoji, a taj plan propada jer ne postoji put do ključa. Ovakvi slučajevi mogu biti zahtjevni za procesor, a problem je moguće riješiti ograničavanjem broja upita agenta prema planeru na bilo kakav način. Također ti isti agenti bi mogli pokupiti bitne predmete umjesto čekanja da ponestane ključeva. Komercijalno korišteni GOAP sustavi koriste logiku prvog reda, za razliku od propozicijske logike korištene u implementaciji ovoga rada i imaju posebnu akciju za navigaciju koja traži put do bilo kojeg objekta i takvi planovi bi bili odbačeni od strane planera, a ne za vrijeme izvršavanja akcije, što u ovom slučaju dovodi do čekanja da se ostvari uvjet postojanja puta. Ako agent nema uvjete za baš nikakav cilj, moguće je implementirati akciju *NeRadiNišta*, bez preduvjeta, koja ostvaruje nekakav cilj najnižeg prioriteta i služi za ograničavanje broja upita prema planeru.

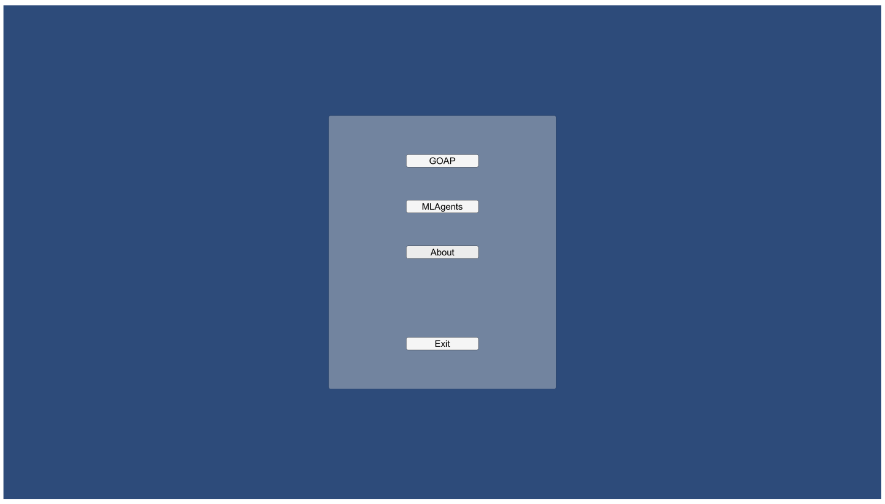
Iako nije implementirana eksplicitna suradnja, agenti pokazuju složena grupna ponašanja ostvarena jednostavnim akcijama. To je rezultat korištenja centraliziranog izvora znanja o svijetu i zaključavanja resursa iz jednog izvora. Veći stupanj suradnje u komercijalno korištenim GOAP sustavima postiže se uvođenjem metaagenta ili koordinatora. Metaagent ne postoji fizički unutar scene, nego služi za generiranje posebnih planova u kojima su upareni fizički agenti i akcije. Tako generiran plan je opet niz akcija i predstavlja konačni automat, ali moguće je paralelizirati njegovo izvršavanje. U tom slučaju moguće je ugraditi, ali nije nužno, određenu otpornost na propadanje dijelova plana metaagenta.

6. Korisničko sučelje

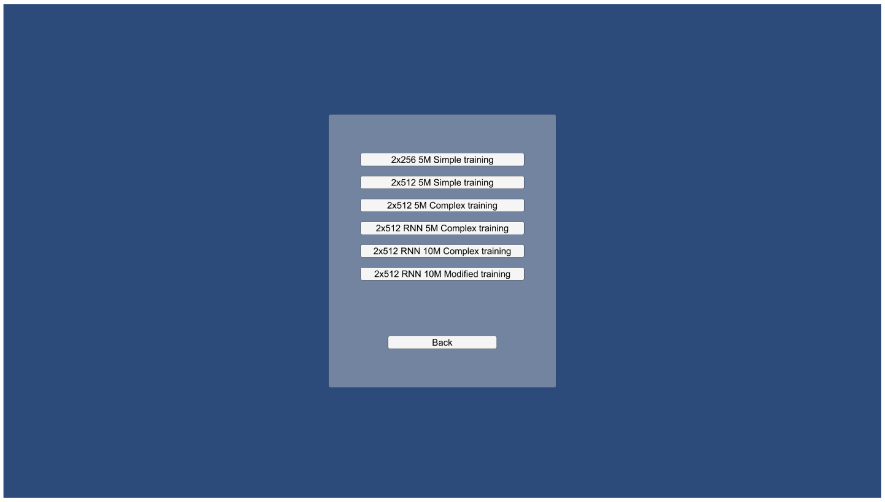
Prilikom otvaranja izvršne verzije, pred korisnikom je glavni izbornik, slika 6.1a. Moguće je odabrati GOAP ili ML agenta, slika 6.1b, a nakon toga scenu, slika 6.1c. Uključene su varijante jednostavne scene koje su služile za iterativno testiranje akcija.

Unutar simulacije na raspolaganju je jednostavno grafičko sučelje za interakciju sa scenom, slika 6.2. Korisnik može pritiskom na gumb u donjoj traci promijeniti aktivni objekt, a pritiskom na tlo može postaviti instancu aktivnog objekta. Moguće je pauzirati simulaciju i vratiti se u glavni izbornik. Ako je aktivan GOAP agent, u gornjem lijevom kutu ekrana prikazuju se simboli koji su dio stanja svijeta i njihove vrijednosti.

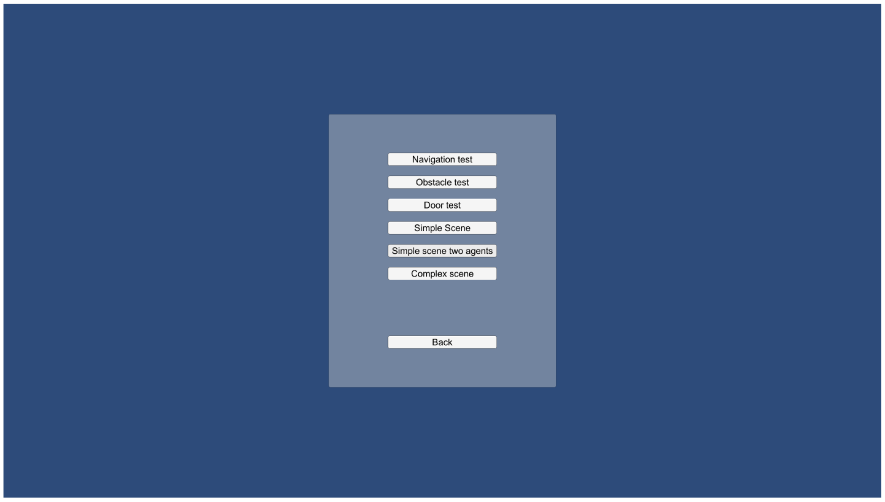
Prilikom postavljanja vrata u scenu koristeći korisničko sučelje, GOAP agenti koji imaju ključeve zaglave. Iz nekog razloga GOAP agenti vrata percipiraju kao nedostupna jer su stvorena dinamički i imaju *NavMeshObstacle* komponentu. Problem je inicijalno riješen tako da agenti provjeravaju je li dostupna barem jedna od 4 lokacije odaljena 1 metar od vrata. Problem se ne javlja ako ih korisnik barem na trenutak pomiče držanjem miša. U slučaju da korisnik ispusti vrata čim su stvorena, problem ostaje. Ručno pomicanje objekta problematičnih vrata u Scenskom pogledu unutar Unity Uređivača rješava problem i GOAP agenti vrata percipiraju kao dostupna. Ovo je rezultat nedostataka u trenutnom Unity NavMesh sustavu.



(a) Početni ekran

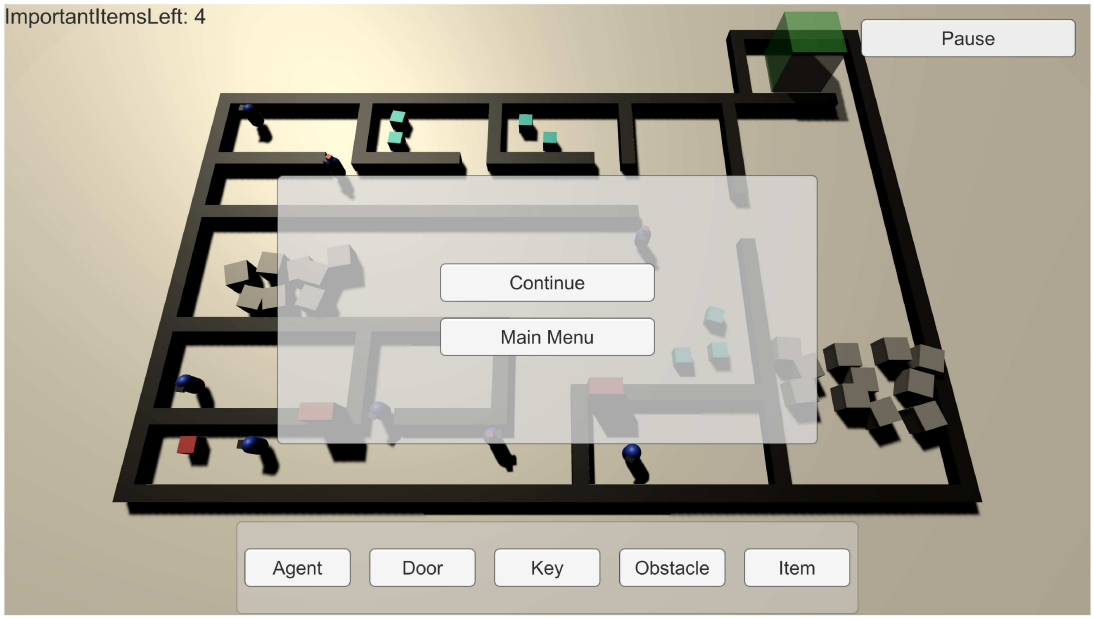


(b) Odabir modela



(c) Odabir scene

Slika 6.1: Glavni izbornik



Slika 6.2: Sučelje unutar simulacije

7. Zaključak

Paket ML-Agents nudi način za jednostavnu i brzu implementaciju potpornog učenja u virtualnim okolišima. Fokus nije na implementaciji algoritama nego na zadatku kojeg agent treba obaviti u određenom okolišu. Za upotrebljive rezultate potrebno je mnogo eksperimentiranja, vremena i resursa. Za dizajniranje okoliša i agenata ipak je potrebno određeno iskustvo i znanje iz domene dubokog potpornog učenja. Nadam se da ću više o tome naučiti na diplomskom studiju.

Automatsko planiranje zanimljivo je područje umjetne inteligencije koje nudi način za automatizaciju zadataka koje je moguće precizno opisati. GOAP arhitektura predstavlja moćan alat za automatsko planiranje u realnom vremenu, najčešće korišten u raznim računalnim igrama. Problemi koje treba prevladati za šire primjene su precizno opisivanje problema i dizajniranje akcija koje rješavaju taj problem. U slučaju velikog broja akcija i ciljeva, problem može biti veličina prostora stanja ako je potrebno planiranje u realnom vremenu.

Dobrota rješenja koje je generirano kroz sustav za automatsko planiranje ovisi o osobi koja ga je dizajnirala. Za kvalitetno rješenje određenog problema potrebno je specifično domensko znanje i razumijevanje. Kombinacija ova dva područja umjetne inteligencije u vidu agenta koji je sposoban sam naučiti akcije i ciljeve, odnosno reprezentaciju svijeta oko sebe, omogućila bi šire i složenije primjene.

LITERATURA

[1] Defense Advanced Research Projects Agency. Darpa robotics challenge. https://www.darpa.mil/program/darpa-robotics-challenge, . Pristupano: 2020-06-10.

[2] Defense Advanced Research Projects Agency. Debut of atlas robot. https://www.darpa.mil/about-us/timeline/debut-atlas-robot, . Pristupano: 2020-06-10.

[3] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, i Danny Lange. Unity: A general platform for intelligent agents. *CoRR*, abs/1809.02627, 2018. URL http://arxiv.org/abs/1809.02627.

[4] Jeff Orkin. Applying goal-oriented planning for games. http://alumni.media.mit.edu/~jorkin/GOAP_draft_AIWisdom2_2003.pdf, . Pristupano: 2020-04-22.

[5] Jeff Orkin. Symbolic representation of game world state: Toward real-time planning in games. http://alumni.media.mit.edu/~jorkin/WS404OrkinJ.pdf, . Pristupano: 2020-04-22.

[6] Jeff Orkin. Agent architecture considerations for real-time planning in games. http://alumni.media.mit.edu/~jorkin/aiide05OrkinJ.pdf, . Pristupano: 2020-04-22.

[7] Jeff Orkin. 3 states and a plan: The ai of f.e.a.r. http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf, . Pristupano: 2020-04-22.

[8] Jeff Orkin. Goal-oriented action planning (goap). http://alumni.media.mit.edu/~jorkin/goap.html, . Pristupano: 2020-04-22.

[9] Unity-Technologies. ml-agents. https://github.com/Unity-Technologies/ml-agents, . Pristupano: 2020-05-30.

[10] Unity-Technologies. Navmeshcomponents. https://github.com/Unity-Technologies/NavMeshComponents, . Pristupano: 2020-05-30.

Simulacija evakuacije agenata

Sažetak

U ovom radu opisane su i implementirane dvije vrste umjetne inteligencije na jednostavnom zadatku snalaženja u virtualnom okolišu, u obliku evakuacije u *escape room* stilu. Agenti moraju surađivati i skupljati te koristiti predmete kako bi uklonili prepreke sa svog puta prema cilju. Prvi pristup baziran je na strojnom učenju i koristi Unity ML-Agents paket s algoritmima za potporno učenje. Drugi pristup baziran je na automatskom planiranju i implementiran je jednostavan GOAP sustav za planiranje uz korištenje Unity navigacijske mreže za kretanje okolišem. Rezultat je interaktivna simulacija u kojoj korisnik može birati scenu i vrstu agenta te dodavati objekte kroz jednostavno grafičko sučelje.

Ključne riječi: umjetna inteligencija, Unity, strojno učenje, potporno učenje, ML-Agents, automatsko planiranje, GOAP, interaktivna simulacija

Agent evacuation simulation

Abstract

In this thesis two types of artificial intelligence are described and implemented to solve a task of overcoming a virtual environment in the form of an escape room. The agents have to cooperate and reach a goal while collecting items and using them to remove obstacles from their path. The first approach is machine learning based and uses the Unity ML-Agents package and its reinforcement learning algorithms. The second approach is based on automated planning and uses a simple implementation of the GOAP planning system along with the Unity NavMesh system for navigating the environment. The result is an interactive simulation in which the user can choose the scene and agent type as well as add objects through a simple graphical interface.

Keywords: artificial intelligence, Unity, machine learning, reinforcement learning, ML-Agents, automated planning, GOAP, interactive simulation