Pronalaženje rješenja platformske igre primjenom strojnog učenje u okruženju Unreal Engine

Stipančić, Damir

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni preddiplomski studij računarstva

PRONALAŽENJE RJEŠENJA PLATFORMSKE IGRE PRIMJENOM STROJNOG UČENJA U OKRUŽENJU UNREAL ENGINE

Završni rad

Damir Stipančić

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 11.09.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Damir Stipančić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3828, 24.09.2019.
OIB studenta:	89806528032
Mentor:	Izv. prof. dr. sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Pronalaženje rješenja platformske igre primjenom strojnog učenje u okruženju Unreal Engine
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	11.09.2020.
Datum potvrde ocjene Odbora:	23.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2020.

Ime i prezime studenta:	Damir Stipančić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3828, 24.09.2019.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Pronalaženje rješenja platformske igre primjenom strojnog učenje u okruženju Unreal Engine**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Časlav Livada

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Tema završnog rada je „Pronalaženje rješenja platformske igre primjenom strojnog učenja u okruženju Unreal Engine“. Završni rad se sastoji od teorijskog i praktičnog dijela. Teorijski dio rada se sastoji od opisa radnog okruženja Unreal Engine sa svrhom upoznavanja sa samim okruženjem. To okruženje je iskorišteno za samu izradu igre te definiranje logike strojnog učenja i svih potrebnih parametara. Također je pokrivena i teorijska osnova samog strojnog učenja te različitih pristupa rješenju problema. Dodatno su još opisani i specifični alati poput programskog jezika C++ te Blueprint alata unutar samog Unreal Engine-a. U praktičnom dijelu rada izrađena je jednostavna platformska igra pomoću koje se vrše simulacije strojnog učenja u svrhu uspješnog rješavanja same igre. Pri izradi je korišteno osobno računalo sa Windows 10 OS te Unreal Engine 4.25.1+.

1.1 Zadatak završnog rada

Zadatak ovog završnog rada jeste istražiti samo strojno učenje te primjenu istog u okruženju Unreal Engine. Glavni fokus je na samoj implementaciji te upotrebi strojnog učenja u Unreal Engine okruženju te prikazu te implementacije na praktičnom primjeru, u ovom slučaju u obliku platformske igre.

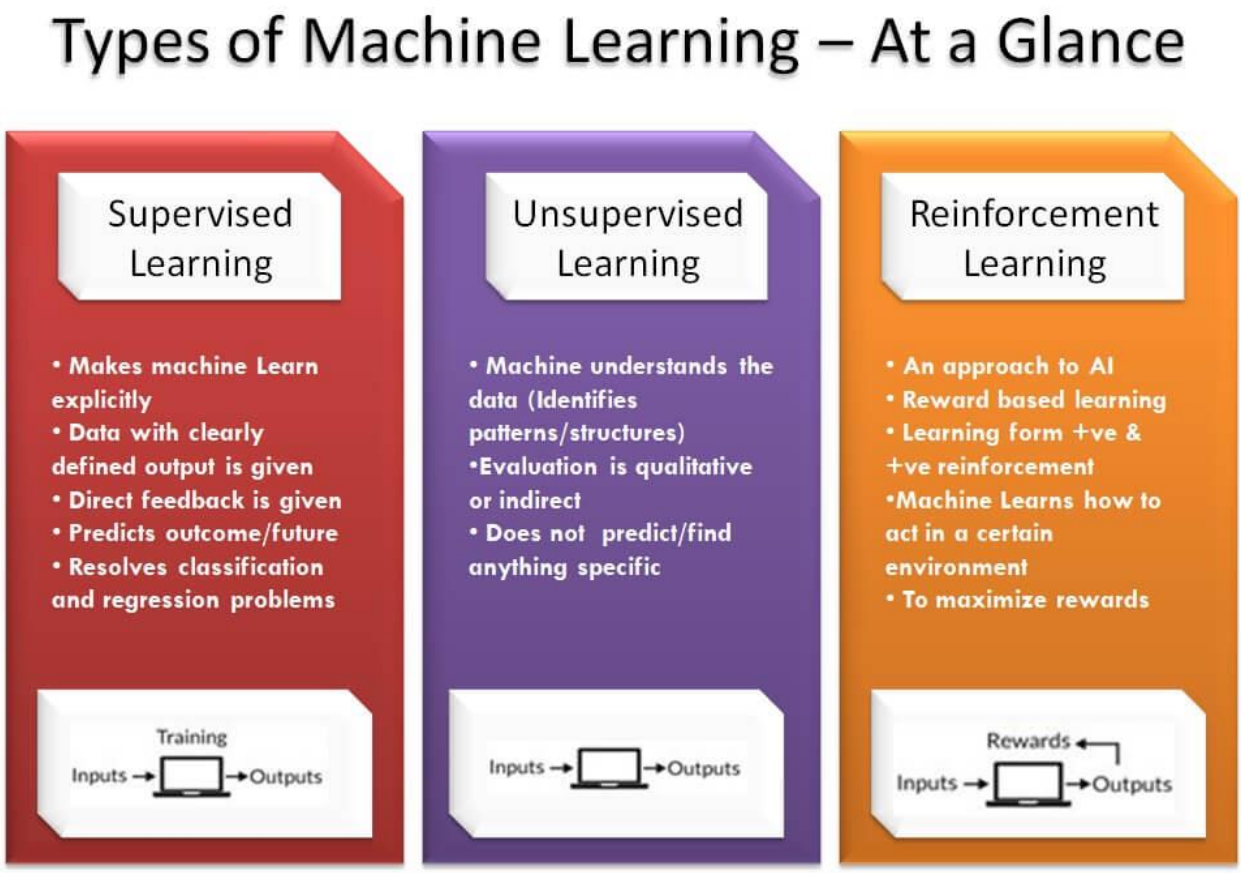
2. STROJNO UČENJE

Strojno učenje se može opisati kao koncept umjetne inteligencije (engl. Artificial Intelligence) čija je svrha optimizirati ponašanje određenog sustava kroz kontinuiranu promjenu same strukture i podataka korištenih za izvođenje određenog zadatka. Glavna ideja strojnog učenja je pružiti potrebne alate i resurse koje sustav može onda iskoristiti da automatski ili nauči nešto novo ili unaprijedi već postojeće ponašanje umjesto da se određeno ponašanje direktno programira. To se postiže na razne načine ali generalna ideja je da sam sustav prima određene unose (engl. input) u vidu podataka kao npr. primjeri očekivanog ponašanja, određena instrukcija i sl. Koristeći te unose sustav pokušava pronaći određene uzorke ili potvrde pravilnog ponašanja te izvodi akciju za koju misli da je pravilna. Ovaj proces se provodi kroz mnogo iteracija te znanja stečena prilikom svake iteracije se prenose u sljedeću. Tako se postiže kontinuirano poboljšanje samog ponašanja sustava.[1]

Glavna primjena strojnog učenja je za izvođenje vrlo kompleksnih zadataka koji se nužno ne mogu direktno programirati. Također se koristi ako postoji potreba za jako fleksibilnim ili prilagodljivim ponašanjem.

Postoje različite vrste strojnog učenja, sa drugačijim pristupima i ciljevima. Najpopularnije su:

- Nadzirano učenje (engl. Supervised learning)
- Nenadzirano učenje (engl. Unsupervised learning)
- Učenje s potporom (engl. Reinforcement learning)[1]



Sl. 2.1 Podjela strojnog učenja (https://www.newtechdojo.com/list-machine-learning-algorithms/)

Slika 2.1 prikazuje jednostavnu podjelu strojnog učenja te neke glavne odlike svake vrste sa jako pojednostavljenim grafičkim prikazom.

2.1 Nadzirano učenje

Nadzirano učenje se koristi u većini praktičnih primjena strojnog učenja. Ova vrsta strojnog učenja pokušava odrediti funkciju algoritma s obzirom na međusobno povezane ulazne i izlazne varijable. Zove se nadzorno jer postoji tzv. posrednik odnosno učitelj koji konstantno ispravlja sam proces i polako ga navodi na pravo rješenje. Postoje dvije različite kategorije problema:

- Klasifikacijski

- Regresijski

Klasifikacijski problemi za izlaz imaju specifičnu diskretnu vrijednost dok u regresijskim kao izlaz se promatra kontinuirana vrijednost ili skupina.[2]

2.2 Nenadzirano učenje

Nenadzirano učenje zapravo ne pokušava pronaći jedno točno rješenje. Primarna svrha mu je proučiti ulazne podatke te doći do vlastitih zaključaka o tim podacima. Nema učitelja koji ga navodi na točan odgovor jer takav odgovor ne postoji. Ovdje također postoje 2 grupe problema:

- Grupiranje (engl. Clustering)
- Asocijacija (engl. Association)

U grupiranju svrha je dakako grupirati ulazne podatke na osnovu nekih zajedničkih svojstava dok u asocijaciji se primarno otkrivaju određena pravila grupiranja i uzorci koji povezuju određene podatke.[2]

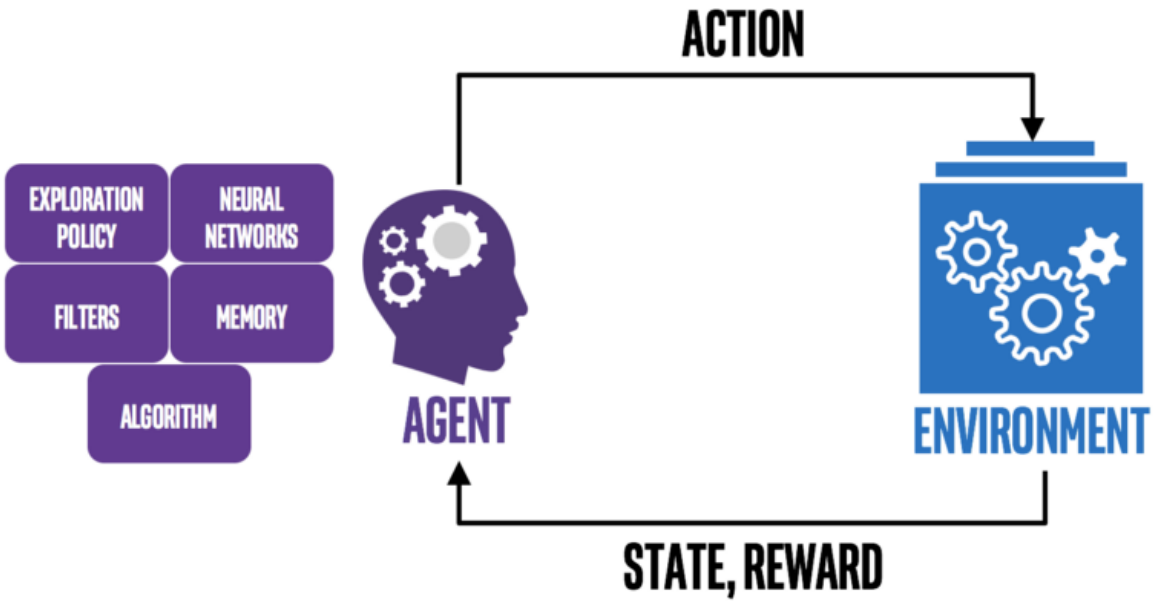
2.3 Učenje s potporom

Učenje s potporom se koristi kada nam je potrebno određeno kontinuirano ponašanje od strane agenta kojeg pokušavamo naučiti. Takvo ponašanje je predstavljeno sekvencama različitih odluka i akcija. Glavna ideja je pustiti agenta da sam pokuša shvatiti što se očekuje od njega putem tzv. „trial and error“ metode. Agent promatra svoju blisku okolicu, prati što se događa te poduzima određene akcije koje imaju svoje specifične nagrade ili kazne. Agent eventualno može raspoznati koje akcije u kojim situacijama će rezultirati nagradom te tako poboljšati svoje ponašanje. Jednostavno rečeno, kreator definira pravila igre ali nikakve upute za rješavanje iste. Na agentu je da sam shvati što treba činiti kako bi bio nagrađen i eventualno uspio riješiti igru.[3]

Strojno učenje sadrži 4 glavna elementa:

- Agent – program na kojem se vrši treniranje
- Okruženje – Područje u kojem agent obavlja akcije

- Akcija – Specifično djelovanje agenta koje uzrokuje promjene u okruženju i učenju
- Nagrada ili kazna – procjena određene akcije i odgovarajući poticaj[3]



Slika 2.2 Shema učenja s potporom (https://medium.com/ai%C2%B3-theory-practice-business/reinforcement-learning-part-1-a-brief-introduction-a53a849771cf)

Slika 2.2 prikazuje jednostavnu shemu učenja s potporom gdje agent ima određena stanja te na osnovu njih vrši određene akcije i prima nagrade ili kazne zauzvrat.

2.4 Strojno učenje u video igrama

Može se reći da je umjetna inteligencija jedan od glavnih aspekata video igara. U video igrama, umjetna inteligencija se koristi od samog početka. Od šaha ili sličnih igara sve do danas kada su u pitanju moderne igre kao npr. Dota 2, Starcraft, League of Legends, Counter Strike: Global Offensive i sl. Sve te igre imaju neku vrstu umjetne inteligencije pomoću koje simuliraju jednostavne verzije protivnika koje igrači obično koriste za vježbanje u ranim fazama upoznavanja sa samom igrom. To je ujedno i glavna ideja primjene strojnog učenja u video igrama. Razviti

jedinstvene i izazovne AI protivnike koji će pružiti popriličan izazov za igrače i omogućiti razne opcije težine i kompliciranosti samih igara.[4]

Danas, strojno učenje se koristi sve više i u industriji video igara. Najčešće su to samo eksperimentalni poduhvati u svrhu kreiranja vrlo sposobne AI logike koja može parirati i najiskusnijim igračima.

Jedan primjer bi bio OpenAI Five[5] iskorišten za igru DOTA 2. Taj AI je pokazao da se strojno učenje može uspješno primijeniti i u samim video igrama. Naime, taj AI je bez problema uspješno nadmudrio neke od najbolji profesionalnih igrača. Eventualno je taj AI proširen te je uspješno pobijedio čitav tim profesionalnih igrača. To je ukazalo na samu moć strojnog učenja kao cjeline te također potvrdilo njegovu široku primjenu u svim sferama umjetne inteligencije.

Još jedan primjer je Alphastar[6]. To je AI za popularnu igru Starcraft 2. Isprva je treniran pomoću nadziranog učenja no eventualno je prešao na učenje s potporom i od tada se poprilično poboljšao te čak uspio i pobijediti profesionalne igrače.

3. UNREAL ENGINE

Okruženje u kojemu je odrađen praktični dio ovog završnog rada je Unreal Engine. Unreal Engine je jedan od raznih game engine-a odnosno to je softverski alat pomoću kojega se izrađuju video igre.

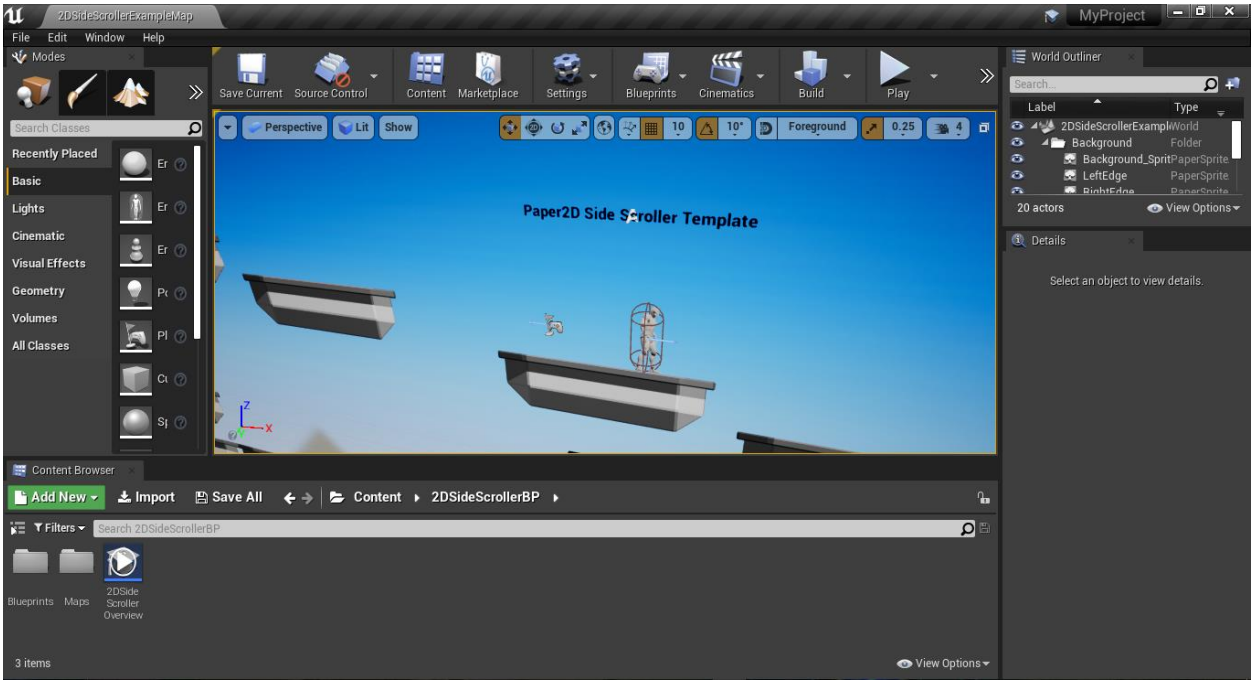
Glavna svrha ovih alata jeste pojednostavniti sam proces kreiranja video igre, od svijeta i interakcija pa sve do AI logike i grafike. Oni, grubo rečeno, sakrivaju tehničke aspekte svih mogućih dijelova dizajna video igara. Na developerima je samo da iskoriste to što imaju na najbolji mogući način.[7]

Kao takvi, game engine alati su poprilično kompleksni softverski paketi i velik broj developera se oslanja na već dobro ustanovljene ponude poput naravno Unreal Engine-a ili pak Unity-a. Oni su potpuno besplatni te je zbog toga relativno jednostavno početi raditi u jednom od njih. Zahtijevaju minimalnu pripremu i vrlo brzo se može početi sa kreiranje konkretnih dijelova igre.[7]

Neki developeri, poput CD Projekt Red, se odluče da kreiraju vlastiti engine umjesto da se oslone na neke od prije spomenuti opcija. Razlog tomu je obično to što oni žele potpunu kontrolu nad svim mogućnostima samog alata te također žele imati fleksibilno i dugotrajno rješenje koje će moći iskoristiti za razne potrebe. Naziv tog alata je REDengine i prošao jer kroz nekoliko velikih iteracija tokom godina. Njihova trenutno najpopularnija igra pod nazivom Witcher 3 je napravljena uz pomoć REDengine 3 dok su prijašnje igre u serijalu napravljene sa REDengine 2 i REDengine 1. Trenutno, CD Projekt Red radi na novoj igri, Cyberpunk 2077, i za to koriste najnoviju iteraciju, REDengine 4. Witcher 3 i Cyberpunk nemaju puno toga zajedničkog, štoviše, to su dvije potpuno različite igre, ali koriste praktički isti game engine. To ukazuje na samu fleksibilnost ovih alata i njihovu široku paletu primjene.[8]

Dizajn ovakvog alata je, kao što je ranije spomenuto, iznimno kompleksan poduhvat i zahtjeva veliko znanje raznih grana matematike, fizike, generalnog programiranja i primjene raznih drugih, manjih, softverskih paketa koji se uključuju u kompletnu strukturu alata. To naravno iziskuje velike financijske i vremenske troškove ali, ako pravilno odrađeno, se poprilično isplati u dugoročnom pogledu.[7]

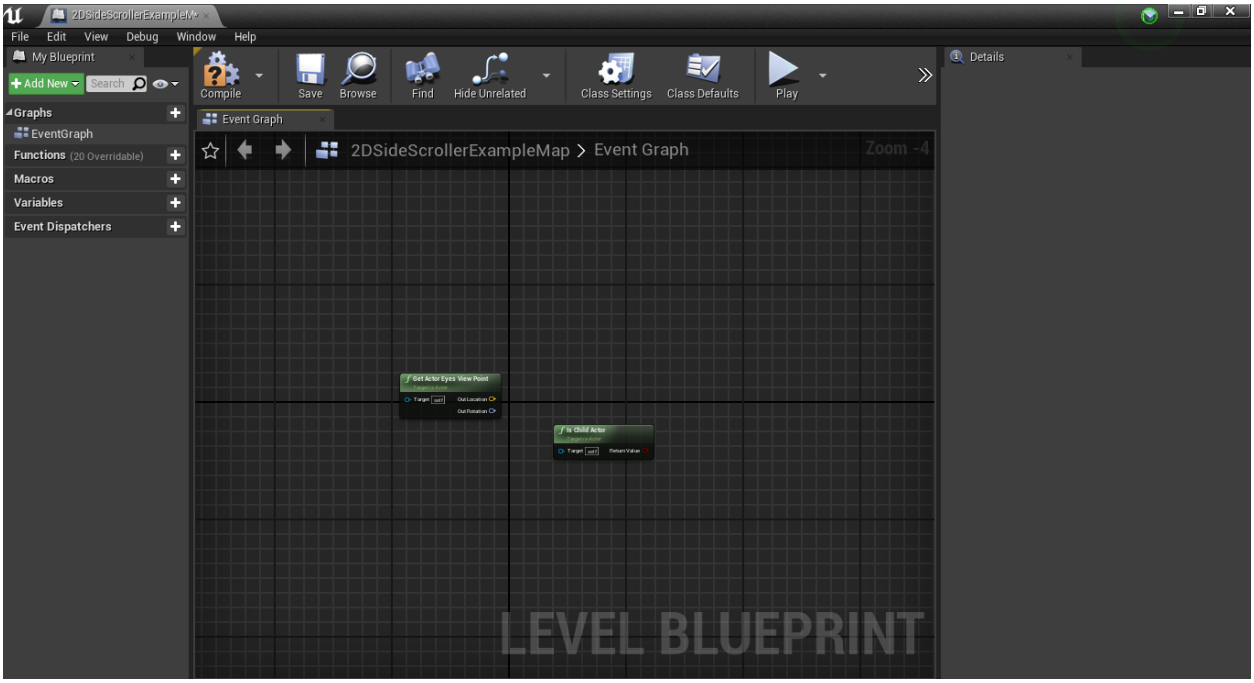
Unreal Engine, za razliku od Unity-a, nema neku posebnu podršku za strojno učenje. To je upravo i bio glavni izazov ovog završnog rada. Čitava implementacija je odrađena samostalno pomoću Blueprint alata. S obzirom na to, za strojno učenje je iskorišten Q-learning algoritam te je njegova implementacija također objašnjena u sklopu praktičnog dijela.



Slika 3.1 Izgled Unreal Engine Editor sučelja

3.1 Blueprint sistem

Blueprint sistem unutar Unreal Engine okruženja je zapravo jedna vrsta vizualnog kodiranja. Korisnik definira klase i objekte putem čvorova te spajanjem tih čvorova definira logiku i razne elemente igre. Sistem je sam po sebi vrlo fleksibilan te ne zahtjeva posebno predznanje i može ga koristiti bilo tko bez obzira da li je ta osoba programer, dizajner ili nešto posve drugo. Blueprint sistem je blisko povezan sa C++ jezikom te se može jednostavno preobraziti u C++ kod ili nadograditi već postojeći C++ kod sa Blueprint modelom. Blueprint sistem se može iskoristiti za kreiranje velikog broja različitih dijelova igre kao npr. kontrolirati određenu scenu, kreirati interaktivne elemente unutar igre, kreirati lika unutar igre, kreirati HUD elemente i dr.[9]



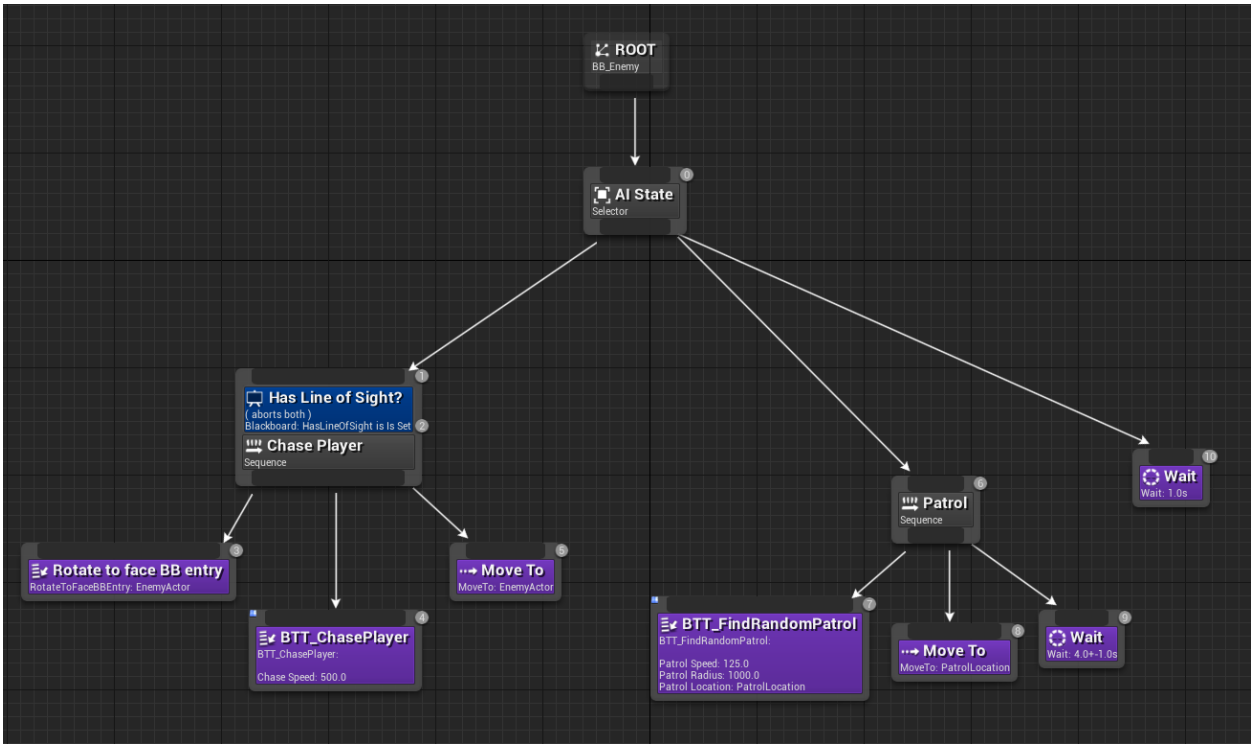
Slika 3.2 Izgled Blueprint sučelja unutar Unreal Engine Editor-a

Postoje razni Blueprinti u jednom projektu kao npr. Level blueprint u kojemu se kontrolira sam level, objekti i akteri u njemu. Postoji i Character blueprint pomoću kojega se kontroliraju razne akcije samog lika unutar igre te je također jednostavno kreirati vlastite Blueprint-e za razne druge stvari.

3.2 Behaviour tree i Blackboard sistem

Behaviour tree je još jedna iznimno važna komponenta Unreal Engine-a. Pomoću njega moguće je kreirati AI za likove koji nisu kontrolirani od strane igrača. Behaviour tree koristi takozvane grane, što se može zaključiti iz samog naziva, koje sadrže razne akcije te na osnovu određenih uvjeta korisnik bira koju granu aktivirati u određenim situacijama. Da bi to mogao izvesti, Behaviour tree radi u skladu sa još jednom komponentom, koja je direktno povezana sa samim Behaviour tree-om, a to je Blackboard.[10]

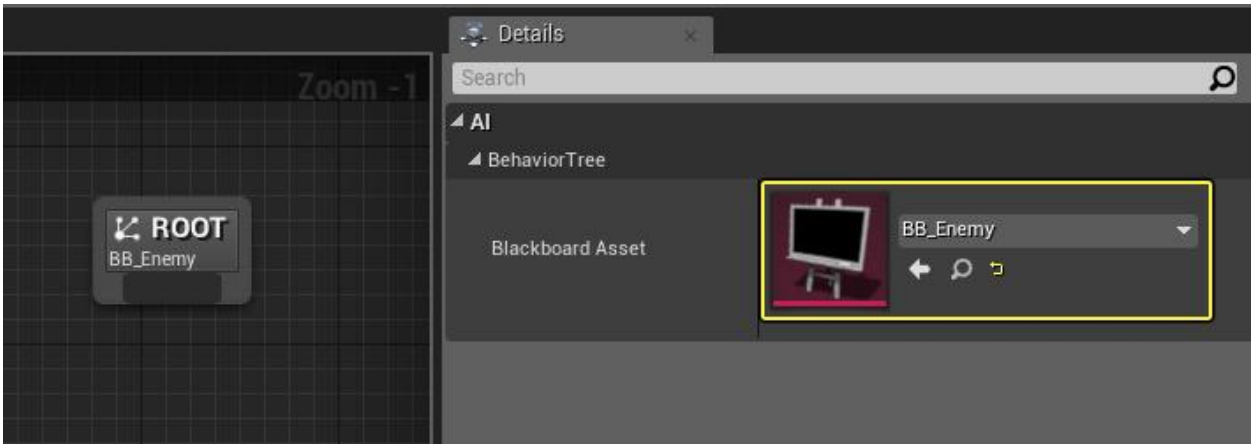
Blackboard je tzv. „mozak“ čitavog sustava. On sadržava određene pojmove, tzv. „Ključevi“ (engl. Keys) koji sadrže razne informacije u obliku klasičnih varijabli, npr. boolean, integer i sl. Njih naravno određuje sam developer te pomoću njih regulira tokove Behaviour tree-a.[10]



Slika. 3.3 Izgled Behaviour tree sučelja (https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreesOverview/index.html)

Slika 3.3 prikazuje jednostavan primjer umjetne inteligencije gdje AI protivnik ima dva različita stanja. U jednom on jednostavno vrši određenu patrolu područja dok u drugom pokušava uloviti samog igrača nakon što ga ugleda.

Behaviour tree je vizualni alat vrlo sličan Blueprint alatu. Funkcioniraju na vrlo sličan način gdje korisnik postavlja određene čvorove i povezuje ih kako bi postigao određeno ponašanje od strane AI lika. Kao što je ranije spomenuto, Behaviour tree blisko surađuje sa Blackboard sistemom. Nakon što korisnik definira razne pojmove unutar Blackboard-a potrebno je taj Blackboard povezati sa samim stablom. To se postiže tako što se pridoda „Blackboard asset“ samom Behaviour tree-u.[11]



Slika 3.4 Povezivanje Behaviour tree i Blackboard alata (https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreesOverview/index.html)

Behaviour tree izvršava zadane naredbe određenim redoslijedom, odnosno, s lijeva na desno te odozgo prema dolje. Redoslijed izvršavanja pojedinog čvora je vidljiv u gornjem desnom kutu svakog čvora. Predstavljen je rednim brojem izvršavanja.[11]

4. Q LEARNING

Glavni fokus čitavog rada je naravno sama implementacija strojnog učenja. Za potrebe ovog rada dovoljna će nam biti implementacija Q learning algoritma i sustava potrebnih za njegovo izvršavanje.

Q learning predstavlja dobar izbor u ovom slučaju jer je relativno jednostavan za razumjeti i implementirati te je također odličan za rješavanje samog problema ovog rada, što je naravno uspješno shvaćanje i rješavanje određenog problema koji je predstavljen u igri.

Q learning spada u već prethodno spomenutu vrstu strojnog učenja a to je učenje sa potporom. Učenje sa potporom je popularan izbor kada je u pitanju strojno učenje unutar video igara. Već prethodno spomenuti primjeri poput Dote 2 koriste ovaj način strojnog učenja da razviju svoje napredne AI protivnike.

Sam koncept Q learning algoritma je na površini poprilično jednostavan za razumjeti no implementacija istoga se može pokazati zahtjevnom. Naime, Q learning se može podijeliti u dvije glavne faze.

Prva faza je **trening faza** (engl. training phase). U ovoj fazi agent provodi vrijeme upoznavajući se sa svojim okruženjem i svime što ste događa u njemu. On svoja „zapažanja“ istovremeno sprema u Q matricu. Ta zapažanja su najčešće predstavljena određenim brojčanim vrijednostima. To se još naziva i „nagrada“. Agent će onda tu tablicu iskoristiti u narednoj fazi kako bi uspješno odlučio koje su akcije najoptimalnije i koje će mu dati najveće nagrade.[12]

Ta druga faza je **faza eksploatacije** (engl. exploitation phase). U ovoj fazi, agent provjerava popunjenu tablicu nagrada te na osnovu nje poduzima određene akcije. Ako je trening faza uspješno odrađena, agent bi odmah na početku druge faze trebao pokazivati specifična ponašanja koja vode ka uspješnom rješenju problema.[12]

Kako bi agent uopće znao koje su mu sve nagrade dostupne, njih treba prethodno definirati. Te nagrade se spremaju u matricu nagrada (R matrica) s kojom se agent konstantno „konzultira“ te izvlači zaključke. Ta tablica naravno sadrži sve moguće kombinacije određenih stanja i akcija koje agent može poduzeti u tom stanju te su one predstavljene nagradama koje agent može primiti.

Razlikuje se od Q tablice po tome što Q tablica sadrži samo one nagrade koje su otkrivene od strane agenta i koja se koristi u finalnoj fazi.[12]

Sam Q learning algoritam je zapravo funkcija koja prima određene parametre te na osnovu njih daje rezultat. Taj rezultat je naravno sljedeća akcija koju agent treba poduzeti.

$$\mathbf{Q(stanje, akcija) = R(stanje, akcija) + \gamma * MAX[Q(sljedeće\ stanje, sve\ akcije)]}$$

Parovi „stanje, akcija“ se odnose naravno na koordinate unutar Q ili R tablice. Znak gamma predstavlja broj između 0 i 1 te nam govori koliko zapravo agent preferira trenutne nagrade u odnosu na buduće. MAX funkcija određuje maksimalnu buduću nagradu koju agent može ostvariti.[12]

5. IZRADA IGRE

5.1 Level dizajn

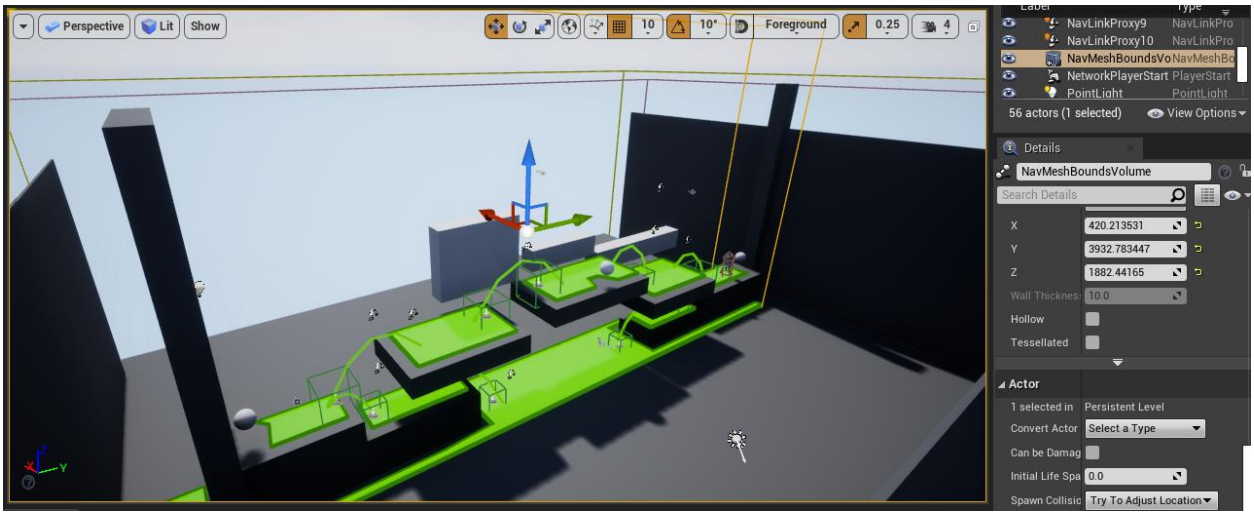
Za izradu igre iskorišten je Side Scroller template ponuđen od strane Unreal Engine-a te su načinjene male promjene. Sama video igra nije ništa posebno jer sam cilj ovoga rada nije nužno kreirati potpunu i vizualno primamljivu platformsku igru, nego prikazati mogućnost primjene strojnog učenje za rješavanje problema i kreiranje AI protivnika u takvom stilu igre. Zbog toga, igra se sastoji od samo jedno vrlog jednostavnog level-a unutar kojega će se agent kretati i pokušati pronaći rješenje problema.

Načinjene su promjene na samom Side Scroller template level-u. S obzirom da igra ne sadrži neki oblik samostalnog pronalaska puta do određenih objekata u level-u (engl. pathfinding) level je poprilično jednostavan te se sastoji od 3 uzvišene platforme i dvije platforme na istoj razini. Također postavljene su 3 sfere unutar level-a, jedna na središnjoj platformi i po jedna na krajnjim. Ovo će pojednostavniti samu implementaciju kretanja agenta, što je za potrebe ovog rada sporedna stvar.

Slika 5.1 Level dizajn igre

Agent počinje na jednoj strani level-a te se kreće po platformama, lijevo i desno. Iako je ovo u jednu ruku 3D Side Scroller template, logika je ista kao i u 2D Side Scroller-u no ovdje postoji više slobode u vidu same implementacije svih potrebnih dijelova.

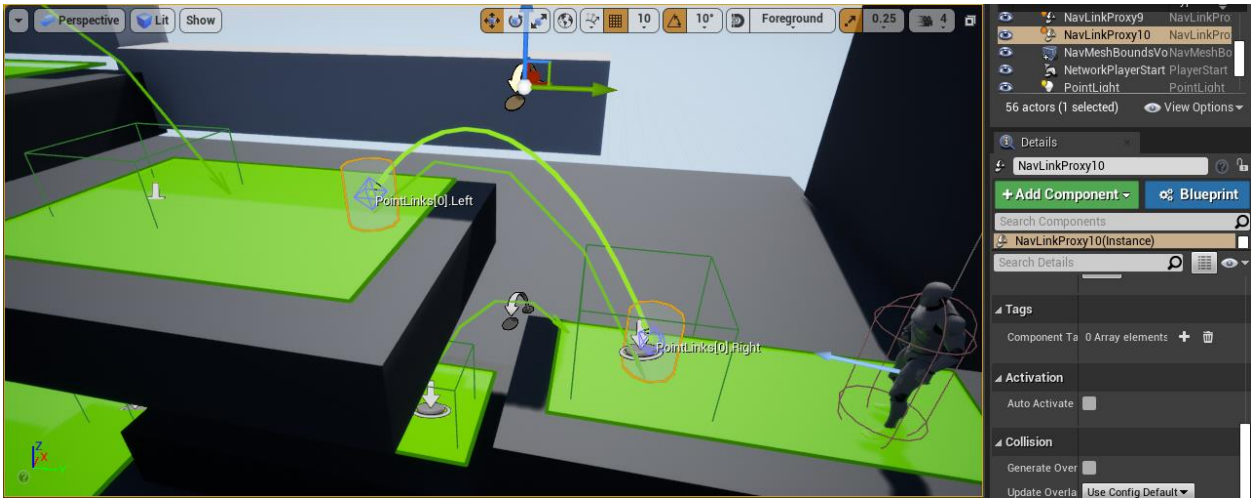
Sljedeći korak je zapravo omogućiti agentu da se kreće. To se postiže pomoću NavMesh Bounds Volume komponente. Pomoću nje, definira se područje kretanja agenta, odnosno sve one dijelove level-a unutar kojih će agentu biti omogućeno kretanje. Bez ovoga, agent bi samo stajao u mjestu.



Slika 5.2 NavMesh Bounds Volume komponenta

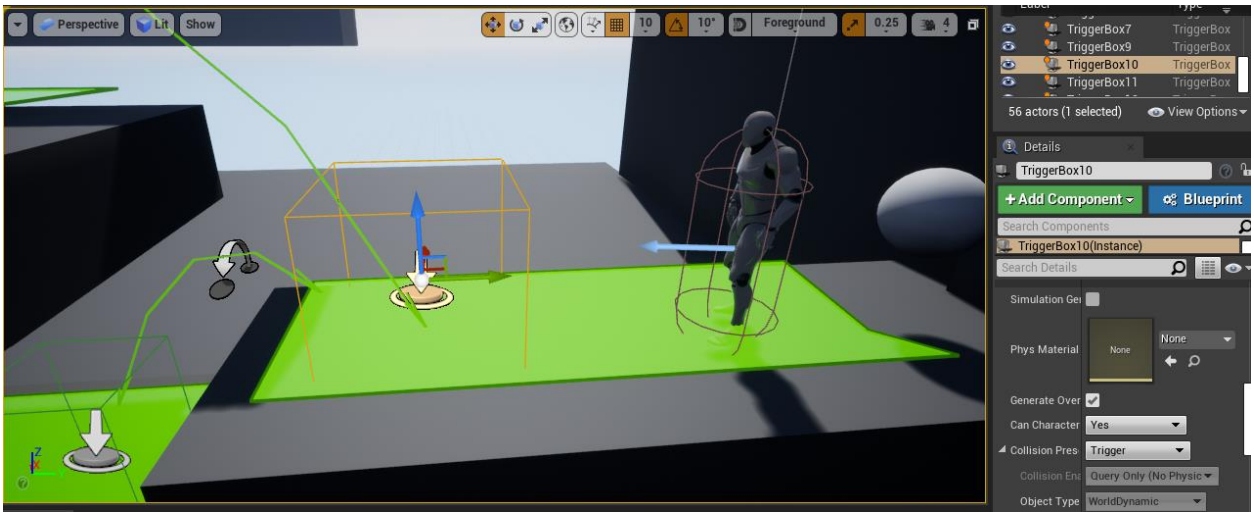
Slika 5.2 predstavlja dakle područje na kojem je definiran sam NavMesh Bounds Volume. Predstavljen je zelenom površinom.

No, u ovom primjeru Side Scroller-a, nije dovoljno samo definirati ovo područje. Potrebno je također spojiti razne platforme kako bi agent prepoznao da se NavMesh Bounds tu ne prekida u potpunosti. To se postiže implementacijom NavLinkProxy elemenata. Između svake susjedne platforme definiran je po jedan NavLinkProxy pomoću kojeg su one povezane. Te veze su predstavljene zelenim strelicama između platformi.



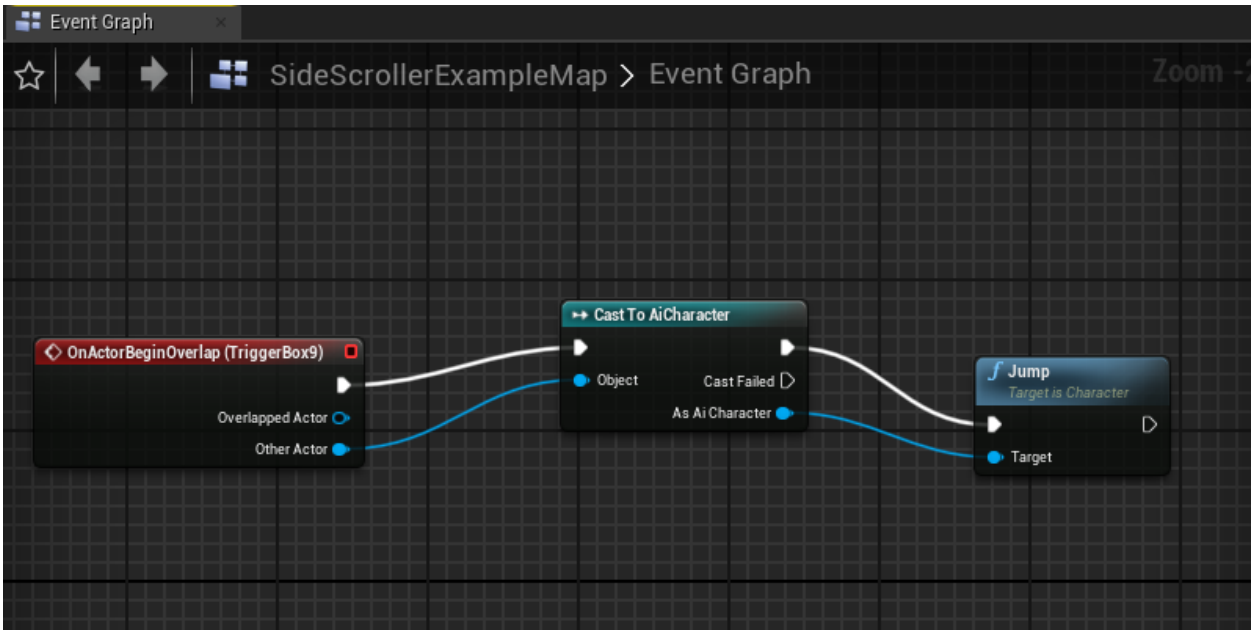
Slika 5.3 Primjer jednog NavLinkProxy elementa

Sljedeći korak je implementacija okidača koji će omogućiti agentu da skoči na samu platformu. Ovo se postiže pomoću TriggerBox elementa. Na svakoj platformi se nalazi barem jedan takav element. Kada se agent „sudari“ odnosno preklopi fizički sa ovim elementom, on će automatski skočiti te tako uspješno stići na sljedeću platformu.



Slika 5.4 Prikaz jednog TriggerBox elementa

Nakon toga implementirana je sama logiku koja će to sve realizirati. To je definirano unutar level Blueprint-a. Odabran je jedan TriggerBox element unutar level editora te tada unutar Blueprint-a je dodan „OnActorBeginOverlap“ node. Pošto je taj node instanciran za odabrani TriggerBox (i svaki budući kopirani TriggerBox) samo je potrebno definirati drugi element na osnovu kojega će se taj Trigger aktivirati. Dovoljno je samo iskoristiti „Cast To“ node te nakon njega izvršiti „Jump“ node.

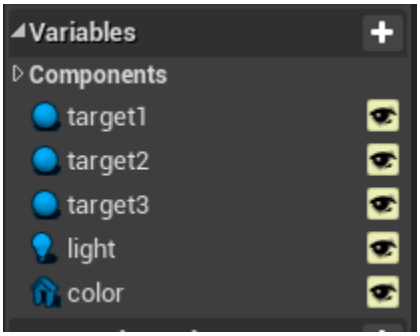


Slika 5.5 Implementacija skoka na sudar sa TriggerBox elementom

Sada se agent može uspješno kretati čitavom površinom dizajnirane razine.

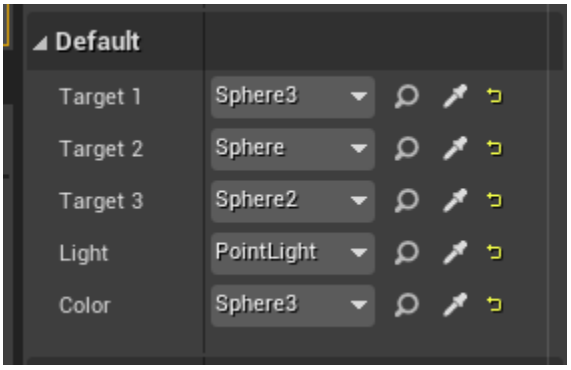
Nakon toga, potrebno je bilo kreirati samog lika koji će predstavljati agenta te definirati pojedine parametre. Kreiran je novi Blueprint Class tipa Character. Njemu je tada pridodan klasični izgled lika. Osim toga, potrebno je definirati određene varijable unutar same Blueprint klase lika. To će onda omogućiti povezivanje lika sa određenim objektima unutar samog level-a što će nam biti potrebno kasnije. Definirane su 3 varijable tipa „Actor“. Nazvane su target (1-3). One su onda iskorištene za povezivanje samih sfera između kojih se agent treba kretati i samog agenta. Također definirana je jedna varijabla tipa „Point Light“ pomoću koje je naravno povezan Point Light

element unutar level-a. Zadnja varijabla će biti tipa „Static Mesh Actor“ i pomoću nje je povezana jedna sfera i agent te je iskorištena za promjenu boje tj. same teksture sfere.



Slika 5.6 Varijable sadržane unutar Blueprint klase agenta

Povratkom u level editor, potrebno je ubaciti agenta u sam level te ga odabrati. Nakon toga, potrebno je povezati te varijable sa objektima unutar level-a.



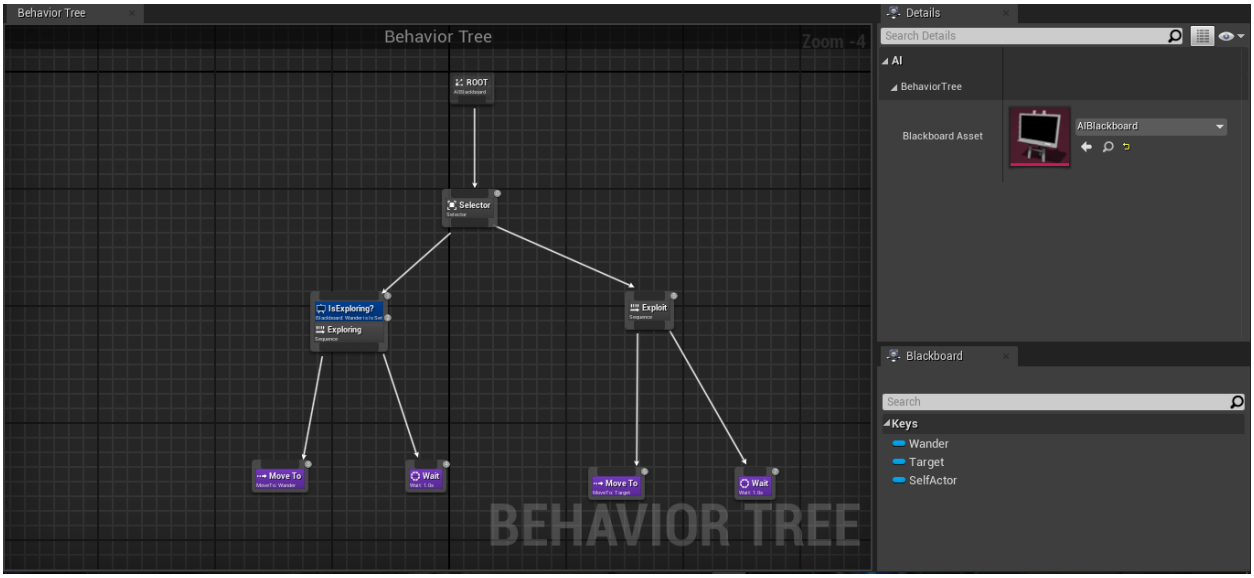
Slika 5.7 Povezivanje varijabli s objektima unutar level-a

Ovo će omogućiti da se kasnije mogu pozivati ove varijable unutar Blueprint dizajna samog Q learning algoritma.

5.2 Dizajn Behaviour Tree i Blackboard komponenti

Za kontrolu kretanja između točnih lokacija iskorišten je jedan Behaviour Tree u kombinaciji sa Blackboard komponentom. Dodani su u projekt te unutar Blackboard-a su dodane 2 nove varijable odnosno ključa tipa „Object“. Nazvani su „Wander“ i „Target“. Pomoću njih će biti moguće razlikovati između kretanja unutar trening faze i faze eksploatacije.

Behaviour Tree je zapravo vrlo jednostavno dizajniran, sadrži samo 2 Sequence elementa za 2 faze. Dokle god je „Wander“ ključ aktivan, agent će biti u trening fazi i kretati se između svih sfera unutar level-a. To se postiže jednostavnom „Move To“ naredbom. Kada to nije slučaj agent će promijeniti stil kretanja te će se kretati sa nekakvom svrhom.

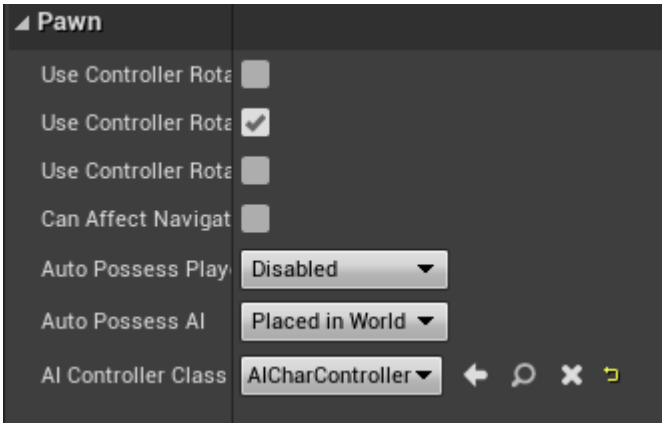


Slika 5.8 Izgled Behaviour Tree komponente

5.3 Implementacija Controller Blueprint klase i Q learning algoritma

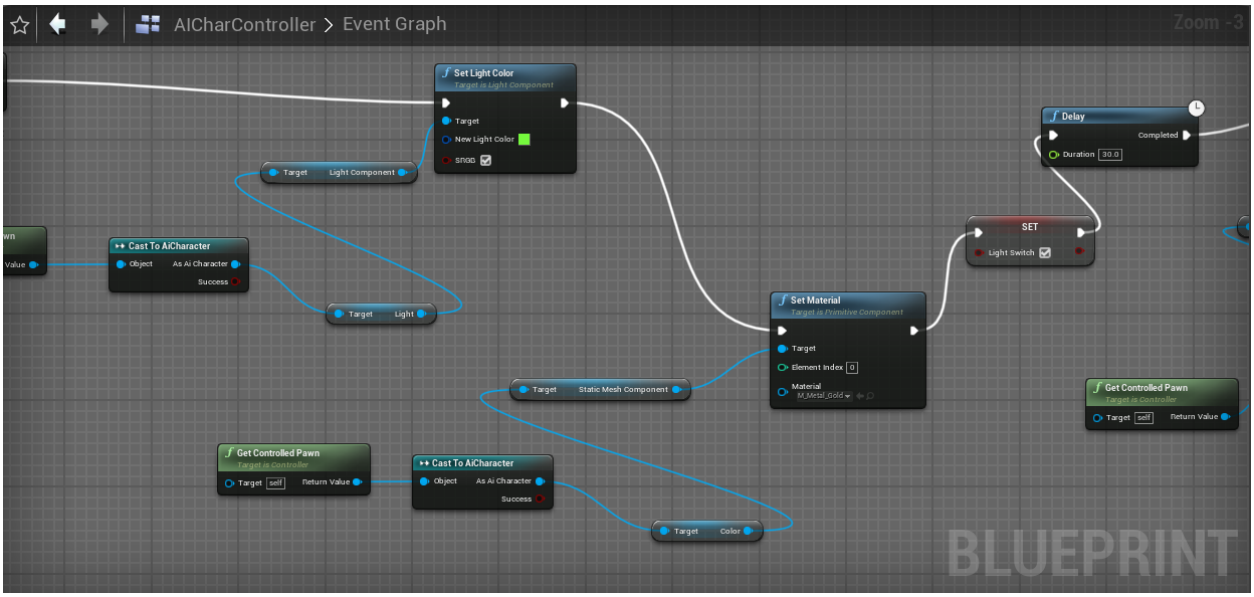
Najbitniji dio rada će biti Controller Blueprint klasa u kojoj se nalazi kontrola svjetla i teksture jedne sfere te implementacija Q learning algoritma i svih potrebnih dijelova. Prvi korak je naravno kreirati novu Blueprint klasu koja je nazvana „AICharController“. Sada je potrebno povezati taj

Blueprint sa agentom tako da ga se može kontrolirati pomoću naredbi unutar tog Blueprinta. Unutar Blueprint klase agenta pod „Pawn“ sekcijom odabrana je „AI Controller Class“ opcija te iz izbornika je odabran prethodno kreiran Controller Blueprint.



Slika 5.9 Povezivanje Controller Blueprint-a sa agentom

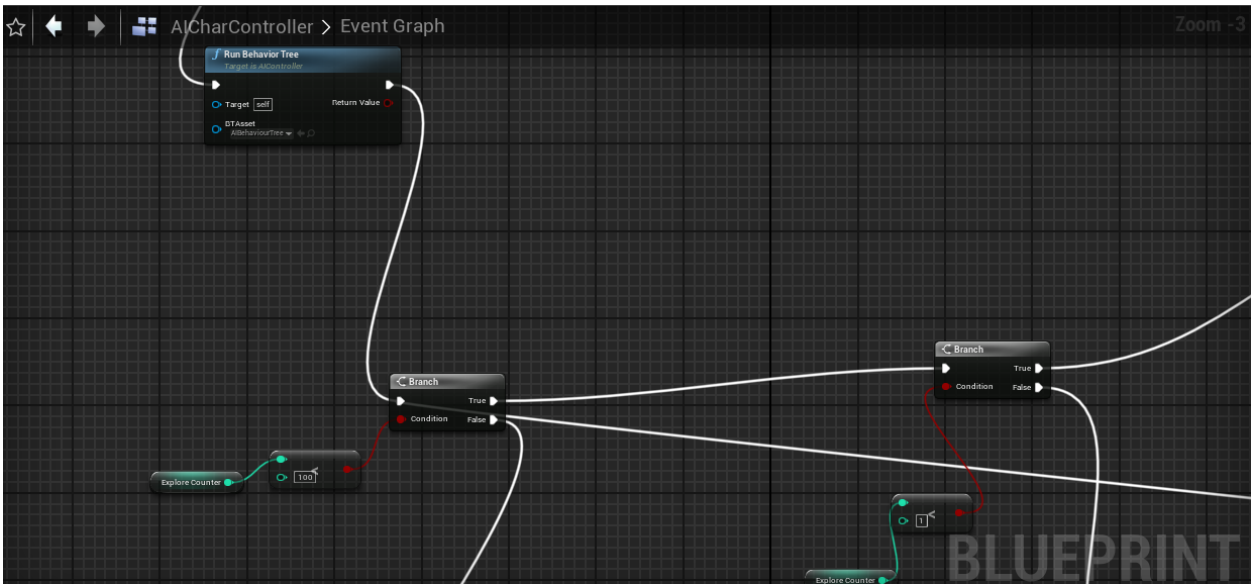
Nakon toga, implementirana je logika kojom se kontrolira stanje svjetla i tekstura glavne sfere unutar level-a. Oni će slučajnim odabirom početi u jednom od dva stanja. Prvo stanje je obična bijela boja svjetla i obična siva tekstura dok drugo stanje će biti predstavljeno zelenom bojom svjetla i zlatnom teksturom sfere. Nakon toga, svakih 30 sekundi će se promijeniti u drugo neaktivno stanje. Controller Blueprint je poprilično velik te će slike sadržavati manje komade radi preglednosti.



Slika 5.10 Promjena boje svjetla i teksture sfere

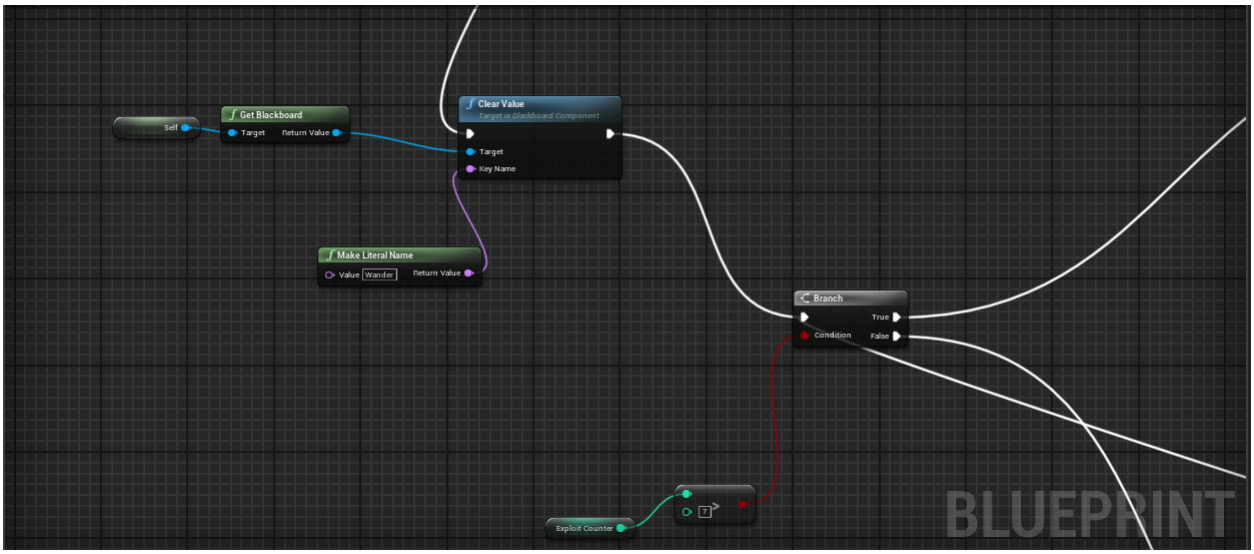
Prethodno spomenute varijable koje su definirane unutar Blueprint klase agenta su ovdje iskorištene. Prvo treba odraditi Cast na agenta te onda preko njega vršiti potrebne izmjene. Ovo je prvi dio „Sequence“ čvora te će se konstantno izvršavati u pozadini dok agent bude učio i primjenjivao naučeno. Ovo svjetlo i sfera igraju ključnu ulogu u rješavanju igre. Uvjet za uspješno izvršavanje igre je direktno povezan sa stanjima svjetla i sfere. Naime, kada je svjetlo zeleno i kada je sfera zlatna, to će predstavljati uvjete pod kojima agent mora izvršiti određenu akciju. Agent će morati prvo otputovati do sfere na suprotnom kraju level-a te se onda zaputiti do zlatne sfere. Upravo ta specifična akcija će rezultirati uspješnim rješenjem problema. Ako zlatna sfera u međuvremenu promjeni teksturu i svjetlo se „ugasi“ odnosno promjeni u bijelu boju, tada problem neće biti uspješno riješen.

Drugi dio Sequence čvora će biti podijeljen u 2 dijela. To su naravno dvije faze u kojima se agent može nalaziti. Ovdje je određeno koliko iteracija će agent izvršiti unutar svake faze. Za trening fazu broj iteracija je 100 dok za fazu eksploatacije je 8. Ovo se može mijenjati po volji.



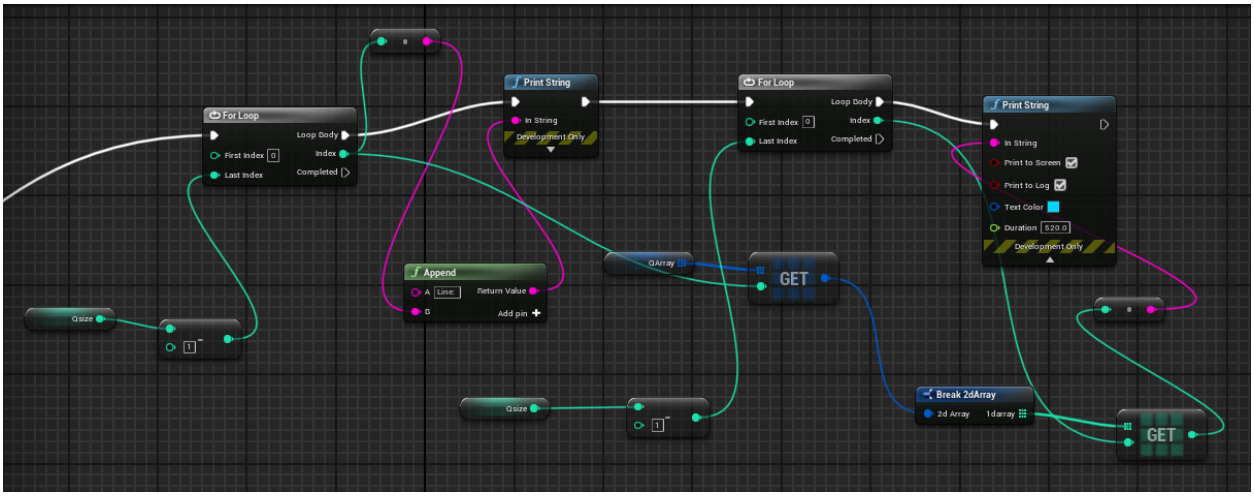
Slika 5.11 Provjera broja iteracija trening faze

Broj iteracija se provjerava jednostavnim brojačem te se izvršavaju akcije specifične za tu fazu dokle god brojač nije veći od 100. Nakon što on postane veći od 100 prelazi se na fazu eksploatacije. Tu se također provjera brojač te nakon što on postane veći od 8 vrši se ispis finalne matrice svih naučenih nagrada.



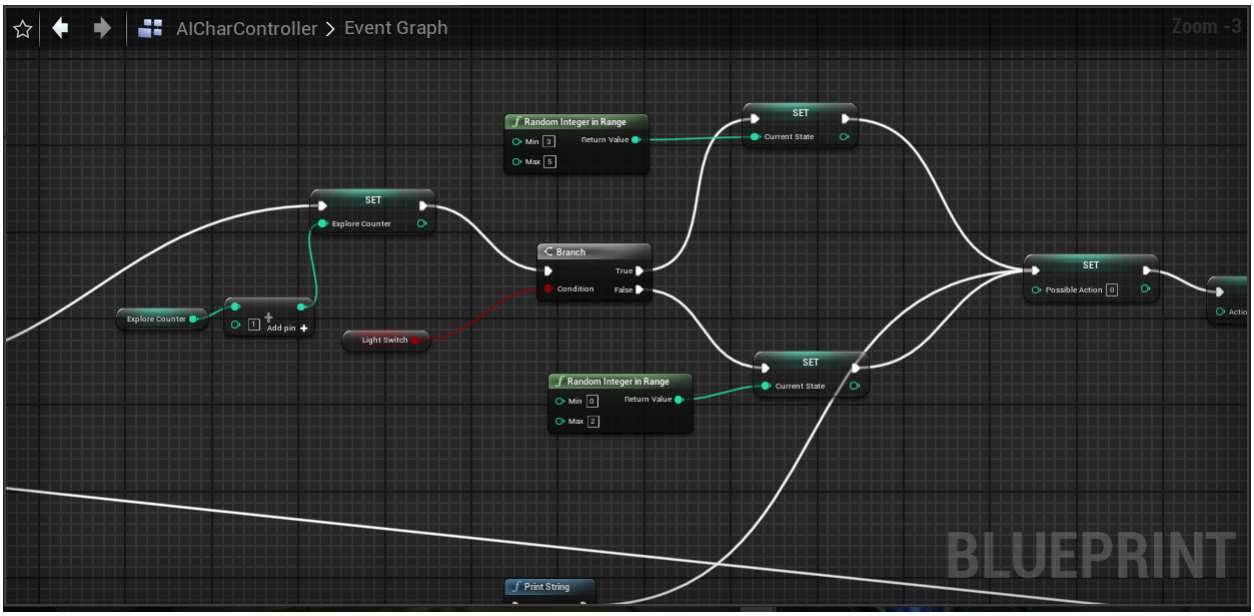
Slika 5.12 Provjera brojača faze eksploatacije i čišćenje Blackboard ključa

Također je bitno naglasiti potrebno čišćenje Wander ključa unutar Blackboard komponente kako bi se agent uspješno prebacio na Target ključ i kretnje specifične za tu fazu.



Slika 5.13 Ispis finalne matrice svih nagrada naučenih od strane agenta

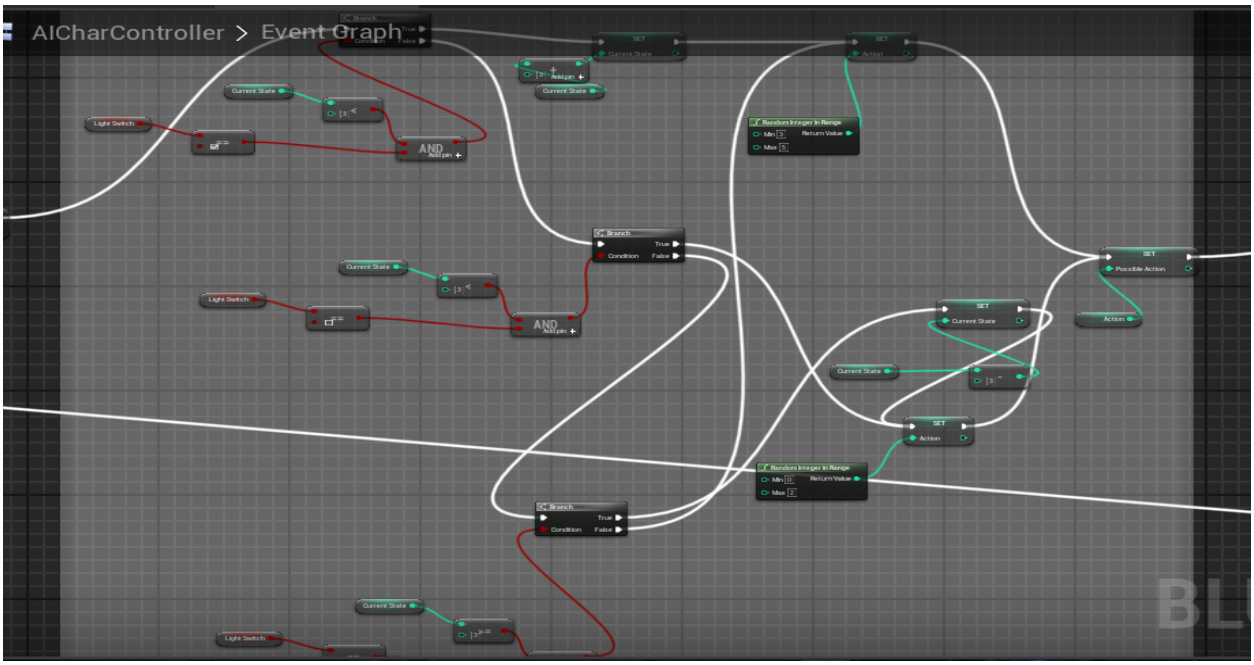
Trening faza počinje dodjeljivanjem jednog od 6 mogućih stanja, s obzirom na stanje svjetla. Pošto postoje 3 sfere i svaka od njih može postojati u 2 stanja, kada je svjetlo bijelo ili kada je zeleno, postoji 6 mogućih stanja. Stanja 0 do 2 predstavljaju 3 sfere kada je svjetlo bijelo a 3-5 predstavljaju 3 sfere kada je svjetlo zeleno. Provjerom stanja svjetla (koje, kao što je prethodno spomenuto, može početi ili kao bijelo ili kao zeleno) se dodjeljuje slučajnim odabirom jedan broj unutar specifičnog opsega.



Slika 5.14 Dodjela početnog stanja agenta

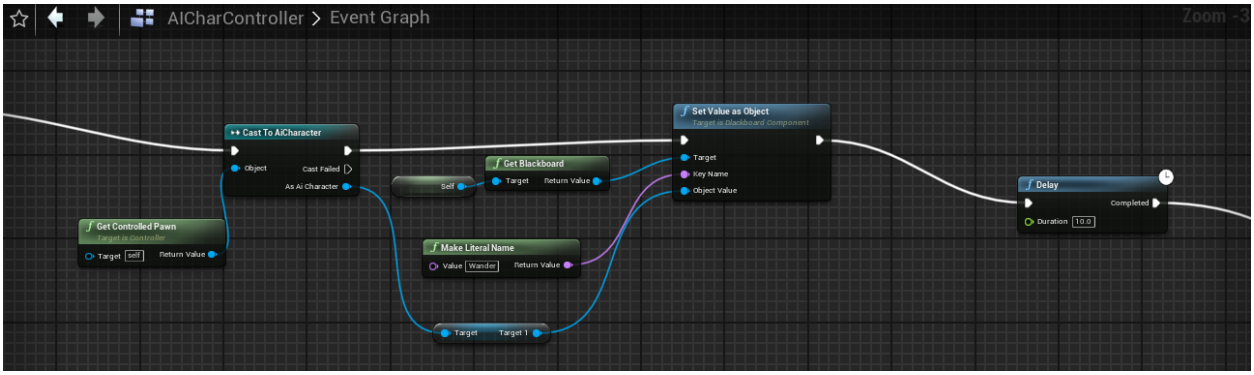
Ovaj dio se izvršava samo ako je u pitanju prva iteracija trening faze, u suprotnom samo se povećava brojač.

Nakon toga, nasumično se bira nova lokacija (jedna od 3 moguće sfere) do koje agent treba otputovati te se također dodaje provjera za stanje svjetla i teksture sfere u slučaju da se ona promijeni dok agent putuje na novu lokaciju.



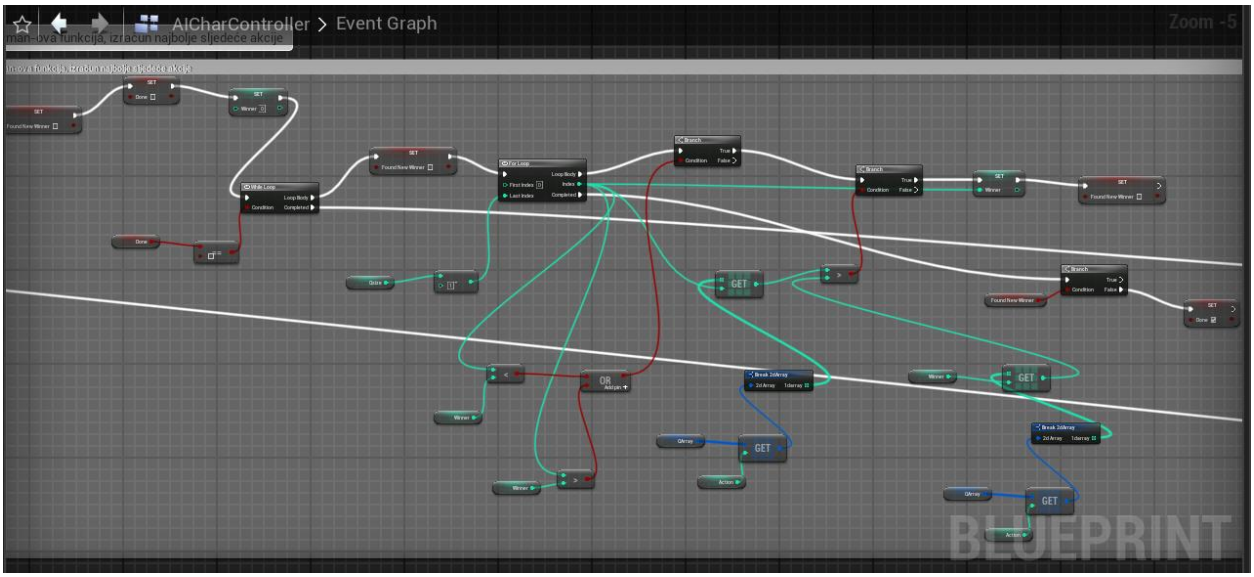
Slika 5.15 Nasumična dodjela sljedeće lokacije

Nakon toga, obavljena je vizualizacija samog kretanja agenta. Ovdje sada igra ulogu prethodno spomenuti Behaviour Tree i Blackboard komponenta. Pomoću „Set Value as Object“ čvora, govori se Blackboard komponenti što da referencira kao „Wander“ ključ te ovisno o prethodno dodijeljenom stanju, bira jednu od sfera. Također ubačen je i „Delay“ node. Agent se naravno kreće određenom brzinom te zbog toga je ubačena mala odgoda sljedećeg koraka jer agentu treba otprilike 7-8 sekundi da stigne do najudaljenije sfere, u slučaju da je ta odabrana kao sljedeća destinacija.



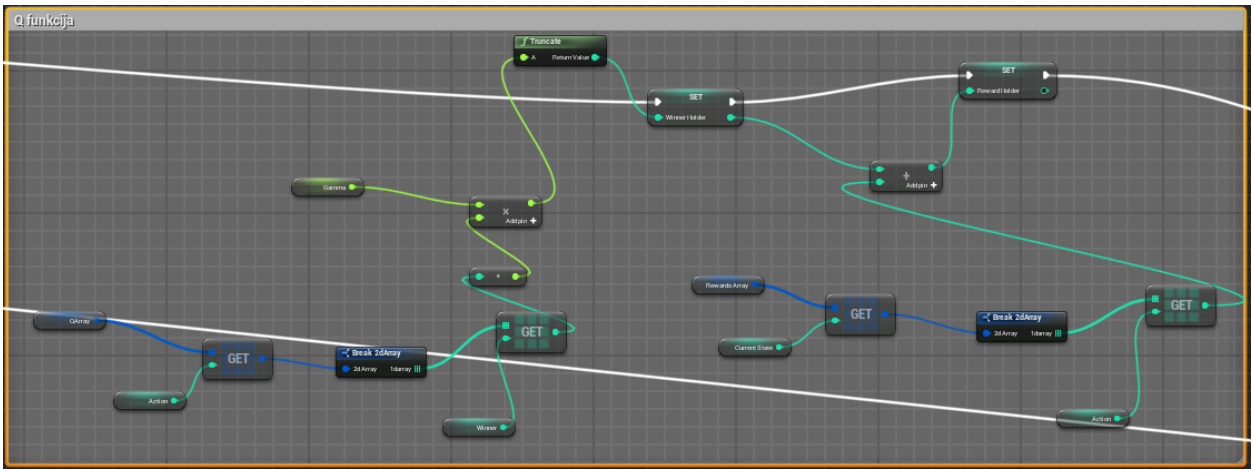
Slika 5.16 Vizualizacija kretanja agenta

U sljedećem koraku, izvršeno je spremanje naučenih nagrada i ishoda u privremenu matricu te će se naknadno ponovno sve objediniti u jednu veliku matricu. Nakon odabira sljedeće destinacije, vrši se jedan iznimno bitan dio Q learning algoritma. U pitanju je Bellamova funkcija. Pomoću nje, agent pokušava predvidjeti najbolju sljedeću akciju s obzirom na trenutno stanje u kojemu se nalazi. To se naziva princip indukcije unatrag (engl. backward induction) gdje agent praktički unaprijed rješava određeni problem i novostečena saznanja vraća natrag u trenutno stanje i na osnovu toga poduzima sljedeći korak. Nakon toga izvršen je ostatak Q learning algoritma, množenje sa gammom te dodavanje nagrade dobivene trenutno izvršenom radnjom.

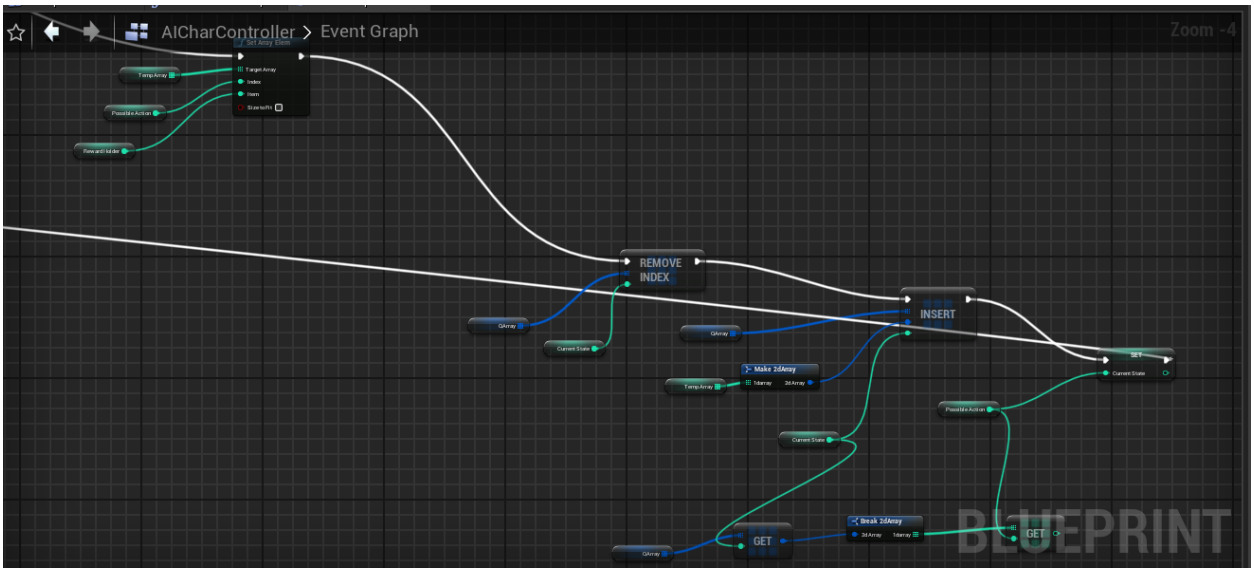


Slika 5.17 Bellmanova funkcija i ostatak Q learning algoritma

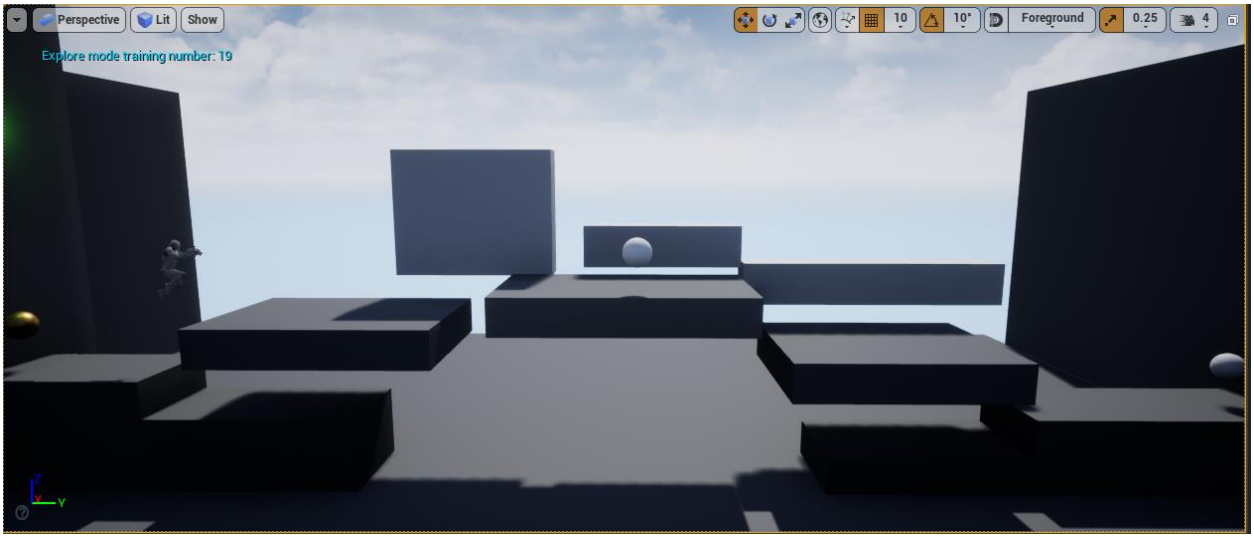
Kada su sve vrijednosti Q learning algoritma poznate, ubacuju se u samu jednadžbu te nakon izračuna vrši se nadogradnja Q matrice te se također postavlja trenutno stanje kao prethodno stanje tako da agent može nastaviti sa novom iteracijom. Ovo se ponavlja naravno 100 puta i upravo to je ključ uspjeha agenta. Svaku iteraciju Q matrica se nadograđuje sa novim vrijednostima koje su točnije od prijašnjih i eventualno se dobiva Q matrica sa nagradama koje će dovesti agenta do uspješnog rješenja problema.



Slika 5.18 Q funkcija



Slika 5.19 Nadogradnja Q matrice pomoću privremene

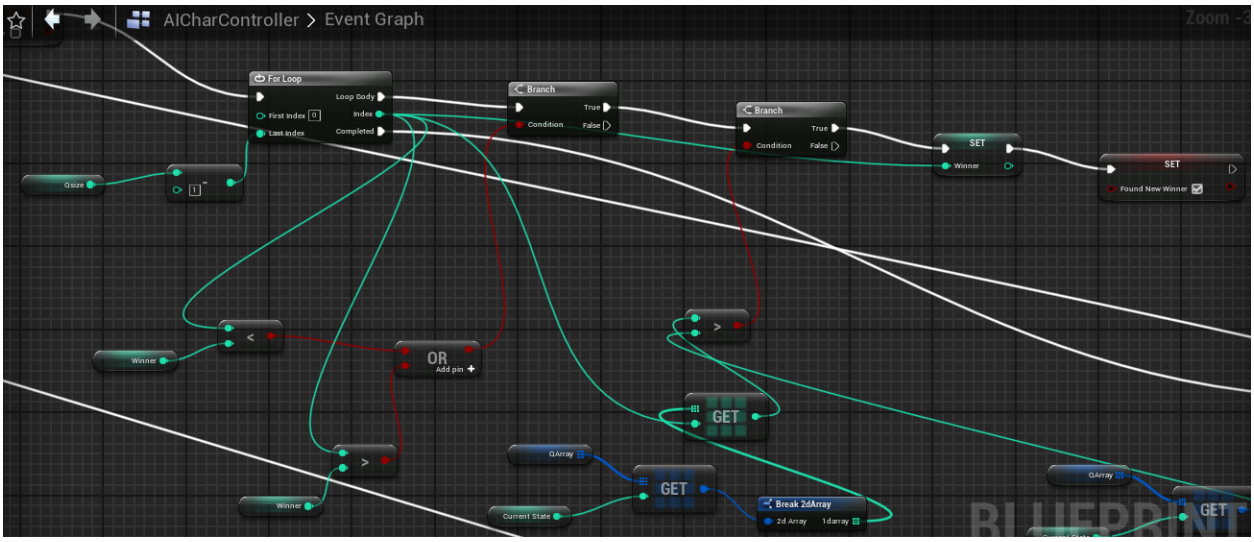


Slika 5.20 Agent se kreće po razini u trening fazi

Ostatak Controller blueprint-a je posvećen zadnjoj fazi a to je faza eksploatacije, u kojoj agent primjenjuje ono što je naučio u prethodnoj fazi. Ova faza je nešto jednostavnija jer sada agent samo mora iskoristiti Q matricu koju je popunio u prošloj fazi da odluči koje akcije će mu omogućiti da uspješno riješi problem ove razine. Dakle, agent će uspješno riješiti problem ako otputuje do sfere na kraju suprotnom od onoga na kojemu se nalazi svjetlo te tek nakon što svjetlo postane zeleno te se tekstura sfere ispod njega promijeni on se zaputi do te zlatne sfere.

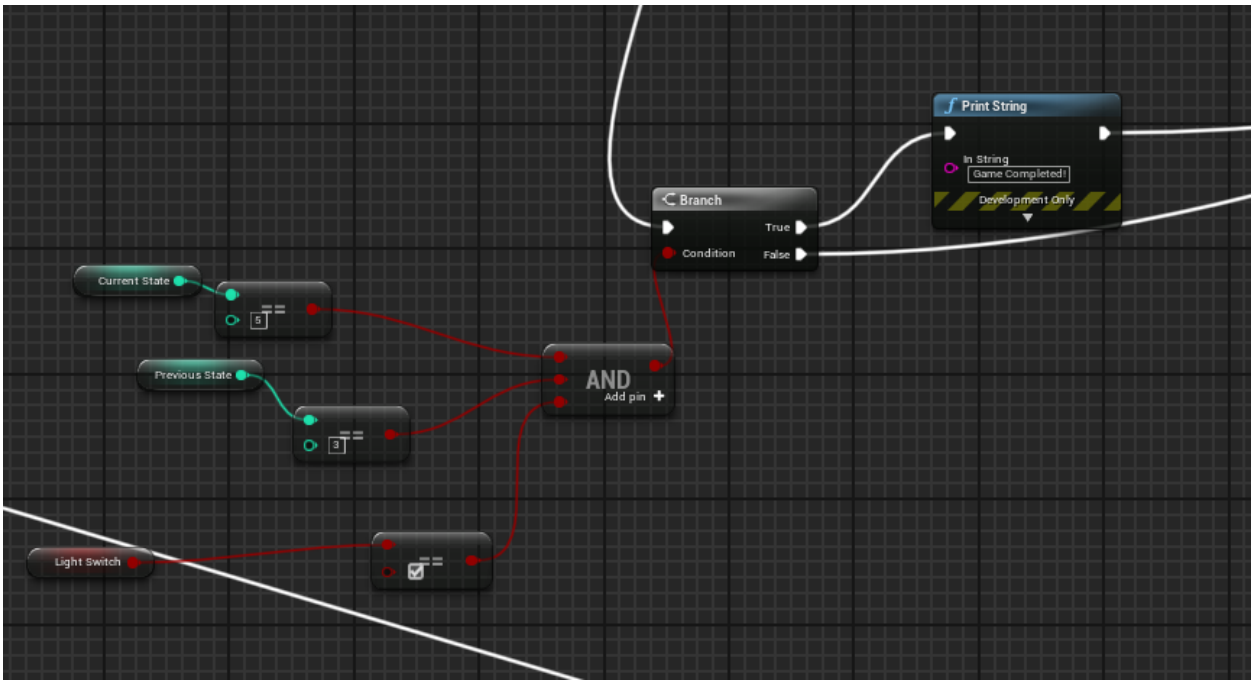
Faza eksploatacije naravno započinje na isti način kao i prethodna fazu, povećava se brojač za trenutnu fazu te se vrše provjere i potrebne izmjene ako se stanje svjetla promijenilo tokom prijelaza iz prošle faze.

Nakon toga, prolazi se kroz Q matricu naučenih nagrada te se utvrđuje najpovoljnija sljedeća destinacija agenta. To će naravno biti destinacija sa najvećom nagradom.



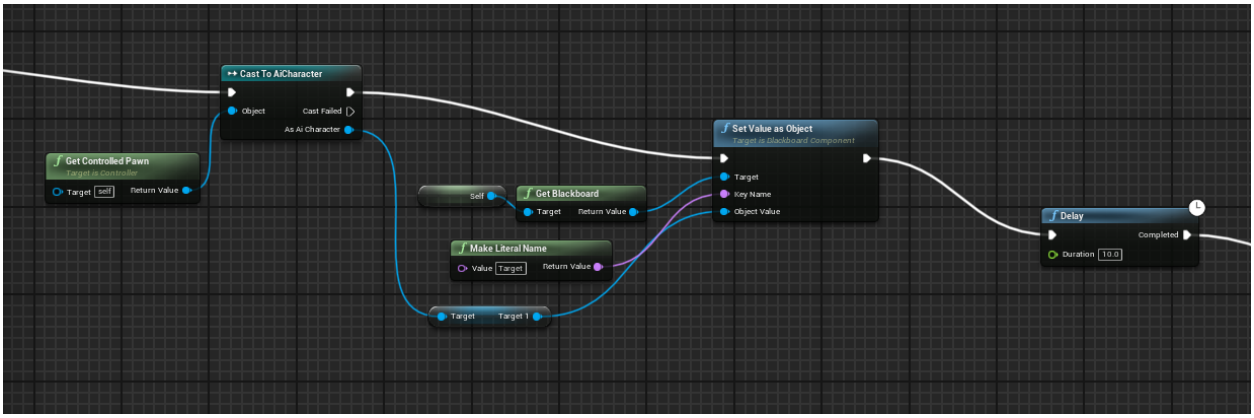
Slika 5.21 Pronalaženje sljedeće najbolje akcije

Također, agent provjerava da li je dobio određenu nagradu za svoju akciju te se na ekran ispisuje tekst „Game Completed“ u slučaju da agent dobije očekivanu nagradu. Ova faza se ponavlja 8 puta te je moguće da dobro naučeni agent više puta pokaže očekivano ponašanje. Na kraju svega ispisuje se Q matrica poznatih nagrada.

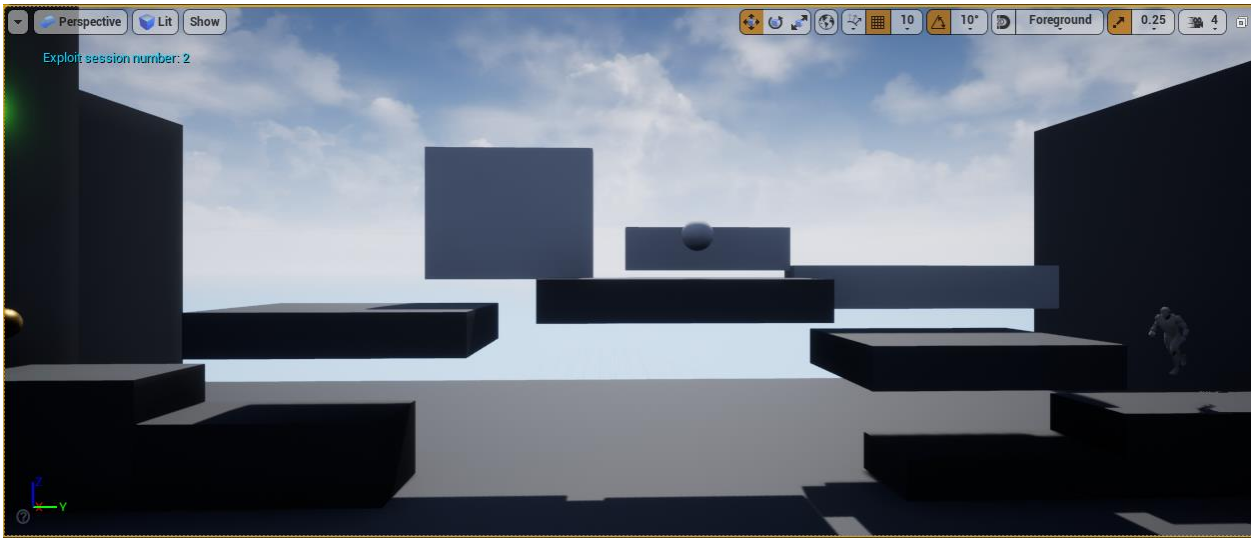


Slika 5.22 Ispis u slučaju uspješnog rješenja

U ovoj fazi naravno također je potrebno kao i u prošloj. Izvršiti samu vizualizaciju kretanja agenta te to činimo na posve identičan način.



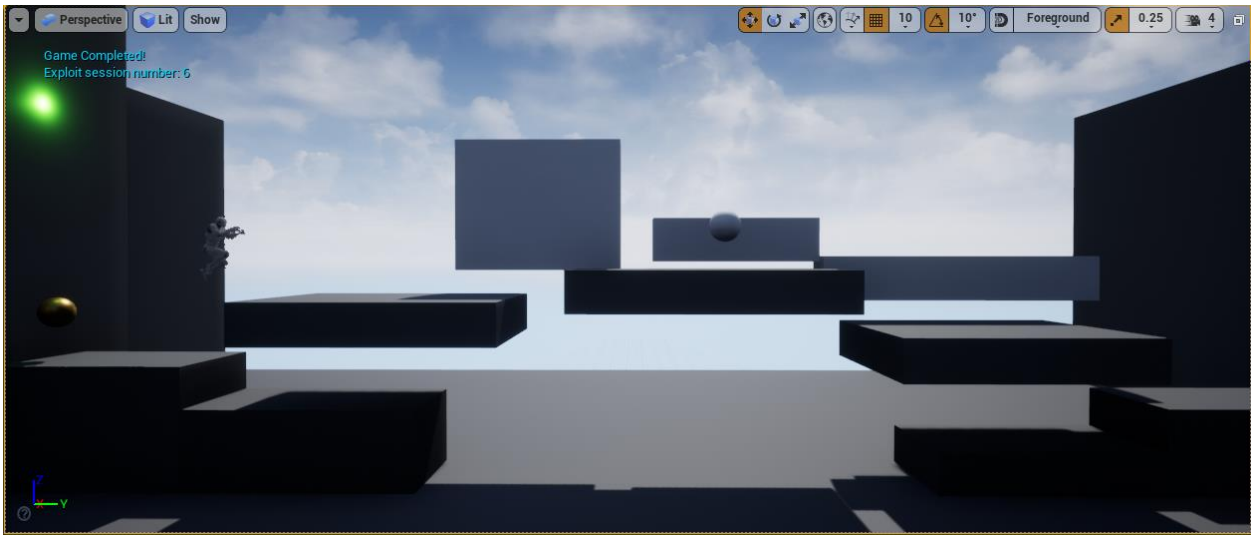
Slika 5.23 Vizualizacija kretanja agenta



Slika 5.24 Kretanje agenta u fazi eksploatacije

6. REZULTAT

Nakon 20 minuta treninga agent u potpunosti shvaća što mu je činiti da bi ostvario najveću nagradu i pri tome naravno uspješno riješio zadani problem. U fazi eksploatacije agent vrlo jasno ponavlja akcije koje ga dovode do rješenja, dakle sve dok je svjetlo zeleno agent će konstanto prvo otputovati do sfere nasuprot zlatnoj te se onda tek zaputiti ka zlatnoj sferi. Ako svjetlo nije zeleno, agent će mirno čekati da ono postane zeleno te onda učiniti to isto.



Slika 6.1 Agent uspješno rješava igru

Na kraju čitavog procesa ispisujemo Q matricu sa nagradama koje je agent sam otkrio u trening fazi i koje je naravno iskoristio u fazi eksploatacije da bi došao do uspješnog rješenja.

Slika 6.2 Ispis Q matrice

Iz log dokumenta možemo vidjeti čitavu Q matricu koja izgleda ovako:

	Akcija					
Stanje	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	0	0	0
	0	0	0	461	327	577
	0	0	0	409	327	327
	0	0	0	409	327	327

Dio samog procesa je snimljen te se može pogledati na sljedećem link-u: https://youtu.be/uuuAZpkVvIg

Video prikazuje završni dio trening faze te prijelaz u fazu eksploatacije i sam završetak simulacije gdje se može vidjeti ispis Q matrice. U ovom slučaju vidi se da je agent uspješno riješio problem već u prvoj fazi eksploatacije. Razlog tomu je to što je agent već bio kod zlatne kugle u tom trenutku a u zadnjoj fazi trening faze je bio kod kugle na suprotnoj strani razine. Nakon toga može se primijetiti promjena boje svjetla. Tada agent mirno čeka kod tog svjetla dok ono ne promijeni boju i kugla ne postane zlatna. Nakon toga vidi se ponovno specifično ponašanje agenta gdje on odlazi do kugle na suprotnoj strani i onda se vraća do zlatne kugle što naravno aktivira uspješno

rješenja problema. Konačno vidi se ispis Q matrice, ne kompletan doduše jer jednostavno ne stane na ekran.

7. ZAKLJUČAK

Ideja i glavni zadatak rada su uspješno ostvareni. Iako Unreal Engine i nema baš najbolju podršku za samo strojno učenje, barem ne u usporedbi sa Unity alatom, ipak je moguće samostalno napraviti barem jednostavnije oblike strojnog učenja koji se mogu primijeniti na razne načine i u većim okruženjima. Velik izbor raznih alata unutar Unreal Engine-a daju dosta fleksibilnosti korisnicima i uz malo domišljatosti moguće je ostvariti razne principe strojnog učenja. U ovom slučaju, predstavljen je jedan od relativno jednostavnijih oblika pomoću Q learning algoritma. Za kompleksnije oblike potrebe u obliku vremena i memorije bi vjerojatno bile puno veće ali za ovaj slučaj, nakon samo 20 minuta i nekoliko matrica popunjenih sa vrlo jednostavnim podacima, uspješno se dolazi do rješenja problema. Naravno, ova implementacija se može proširit i pokušati sa njom riješiti neki malo kompleksniji problem ali za potrebe ovoga rada dovoljno je dokazati da implementacija Q learning algoritma pravilno radi te je moguće riješiti jednostavan problem u relativno brzom periodu.

LITERATURA

[1] Machine Learning: What is it and why it matters, SAS, dostupno na: https://www.sas.com/en_us/insights/analytics/machine-learning.html 5.7.2020

[2] J. Bronwlee, Supervised and Unsupervised Machine Learning Algorithms, 2016, Machine Learning Mastery, dostupno na: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ 5.7.2020

[3] D. Lee, Reinforcement Learning, Part 1: A Brief Introduction, 2019, dostupno na: https://medium.com/ai%C2%B3-theory-practice-business/reinforcement-learning-part-1-a-brief-introduction-a53a849771cf 6.7.2020

[4] M. Tim Jones, Machine learning and gaming, 2019, dostupno na: https://developer.ibm.com/articles/machine-learning-and-gaming/?cm_mmc=OSocial_Facebook-_-Developer_IBM+Developer-_-WW_WW-_-ibmdev-&cm_mmca1=000037FD&cm_mmca2=10010797&2384887362&linkId=68609448#IBMDeveloper 7.7.2020

[5] OpenAI, 2016, dostupno na: https://openai.com/projects/five/ 8.7.2020

[6] AlphaStar team, AlphaStar: Mastering the Real-Time Strategy Game Starcraft 2, 2019, dostupno na: https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii 8.7.2020

[7] Unity, Game engines-how do they work?, dostupno na: https://unity3d.com/what-is-a-game-engine 8.7.2020

[8] The Witcher 3 Interview: A Deeper Look into REDengine 3, 2015, dostupno na: https://80.lv/articles/the-witcher-3-interview-a-deeper-look-into-redengine-3/ 9.7.2020

[9] Introduction to Blueprints, dostupno na: https://docs.unrealengine.com/en-US/Engine/Blueprints/GettingStarted/index.html#:~:text=The%20Blueprints%20Visual%20Scripting%20system,or%20objects%20in%20the%20engine. 20.7.2020

[10] Behaviour Trees, dostupno na: https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/index.html 20.7.2020

[11] Behaviour Tree Overview, dostupno na: https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreesOverview/index.html 20.7.2020

[12] Sacredgames, Machine Learning in Unreal Engine: Uncommon Adventures in Artificial Intelligence, 2017, dostupno na: https://unrealai.wordpress.com/2017/12/19/q-learning/ 20.7.2020

[13] J. Halpern "Developing 2D Games with Unity", (Apress Publishing, 2018)

SAŽETAK

Ovaj rad pokriva temu strojnog učenja u sklopu Unreal Engine okruženja te su pokrivene teorijske i praktične osnove potrebne za izradu istog. Strojno učenje je područje istraživanja koje omogućava implementaciju raznih algoritama pomoću kojih se trenira umjetna inteligencija bez potrebe da se vrši direktno programiranje određenih ponašanja. Unreal Engine je skup programskih alata koji omogućavaju razvoj video igara, umjetne inteligencije, simulacija i sl. Korištenjem alata poput Blueprint-a, Blackboard-a, Behaviour Tree-a te samog sučelja za dizajn grafičkih komponenti kreirana je jednostavna platformska igra koja se riješila upravo pomoću strojnog učenja. Zadatak agenta je bio da otputuje na 2 različite lokacije u točno određenim trenutcima. Za razliku od Unity-a, Unreal Engine nema posebnu podršku za strojno učenje te je zbog toga za izradu čitavog rada bilo potrebno znanje o raznim alatima unutar Unreal Engine-a, strojnom učenju te posebno je trebalo proučiti Q learning i njegove razne implementacije.

Ključne riječi: agent, platformer, strojno učenje, učenje s potporom, umjetna inteligencija, Unreal Engine

ABSTRACT

This thesis covers the topic of machine learning in Unreal Engine and it covers the theoretical and practical basics needed to successfully complete it. Machine learning i san area of research that enables the implementation of various algorithms which are used to train artificial intelligence without the need to explicitly program specific behaviours. Unreal Engine is a collection of tools which enable the creation of video games, artificial intelligence, simulations and others. Using tools like Blueprints, Blackboard, Behaviour Tree and the graphical design editor a simple platformer game was created and then successfully completed using machine learning. The main goal for the agent was to move between 2 specific locations at specific times. Unlike Unity, Unreal Engine lacks a dedicated support for machine learning and because of that, the completion of this thesis required knowledge about the various tools available inside Unreal Engine and machine learning, especially the theory of Q learning and its various implementations.

Keywords: agent, artificial intelligence, machine learning, platformer, reinforcement learning, Unreal Engine