Självlärd artificiell intelligens i Unity

Anton Sällström

Examensarbete

Informationsteknik

2022

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identification number:	
Författare:	Anton Sällström
Arbetets namn:	Självlärd artificiell intelligens i Unity
Handledare (Arcada):	Jonny Karlsson
Uppdragsgivare:	Jonny Karlsson

Sammandrag:
Detta examensarbete beskriver tillämpning av maskininlärning i spelmotorn Unity. Unity används för att utveckla diverse 2D- och 3D-spel. Unity Technologies utvecklar verktyg som stöder maskininlärning i Unity:s spelmotor. Unity Technologies har egna olika utvecklade spel baserade på maskininlärning vilka hittas på deras egna Github. Examensarbetet förklarar teoretiskt olika maskininlärningsmetoder samt går igenom hur man bygger upp ett fungerande spel i en 3D-miljö för maskininlärning. Spelets 3D-miljö har en simpel struktur varav en maskininlärningsagent ska hitta spelets mål. Dessa simulationer utförs i spelets 3D-miljö och bygger på förstärkningsinlärning, imitationsinlärning och en blandning av båda två.
Efter simulationerna visualiseras data med hjälp av Tensorflow. Tensorflow är ett program som illustrerar dataflödesdiagram. Resultaten analyseras och diskuteras i slutet av arbetet.

Nyckelord:	Unity, AI, Agent, Förstärkningsinlärning, Imitationsinlärning
Sidantal:	42
Språk:	Svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Informtation Technology
Identification number:	
Author:	Anton Sällström
Title:	Självlärd artificiell intelligens i Unity
Supervisor (Arcada):	Jonny Karlsson
Commissioned by:	Jonny Karlsson

Abstract:
This thesis describes the application of machine learning in the game engine Unity. Unity is used to develop various 2D and 3D games. Unity Technologies develops tools that support machine learning in the Unity game engine. Unity Technologies has its own various developed games based on machine learning which can be found on their own Github. The thesis theoretically explains different machine learning methods and goes through how to build a working game in a 3D environment for machine learning. The game's 3D environment has a simple structure from which a machine learning agent must find the game's goal. These simulations are performed in the game's 3D environment and are based on reinforcement learning, imitation learning and a mixture of both. After the simulations, the data is visualized using Tensorflow. Tensor-flow is a program that illustrates data flow diagrams. The results are analyzed and discussed at the end of the work.

Keywords:	Unity, AI, Agent, Förstärkningsinlärning, Imitationsinlärning
Number of pages:	42
Language:	Swedish
Date of acceptance:	

Innehåll

Förord

Mitt intresse för det här ämnet började när vi hade våra första lektioner i Unity game Engine när man fick för första gången programmera ett mindre spel. Efter det har jag funnit alla kurser med spelprogrammering intressanta, och jag valde även som breddstudier medias ”Game Theory” för att lära mig mer om speldesign. Efteråt började jag på min egen fria tid programmera mindre projekt för att lära mig tekniker samt deltagit på gamejams för att få testa mina kunskaper och lära mig känna nya människor som har gemensamma intressen. Nu på senare dagar har jag velat lära mig programmera en AI bot som skulle adaptera sig basis på hur spelaren själv beter sig och tvinga spelaren att adaptera som i sin tur gör att AI adapterar sig med.

Annars vill jag tacka Jonny Karlsson för att ha väckt detta intresse åt mig samt Mirko Ahonen för att ha gett mig ett bredare kunnande inom gamedesign. Till sista vill jag tacka mina vänner som hjälpt mig på vägen igenom Arcada,

Helsingfors oktober 2022

Anton Sällström

Figurer

Begrepp

AI – Artificiell intelligens

ML – Maskinlärning

API – applikationsprogrammeringsgränssnitt (application programming interface)

SDK – mjukvaruutvecklings kit (Software development kit)

1 Inledning

1.1 Bakgrund

Artificiell intelligens i Unity är aktuellt för spelutveckling och därför intressant att fördjupa sig i. Artificiell intelligens har använts i spel redan på 1950-talet när Alan Turing publicerade ett program som kunde spela schack (Martin Stezano. 2017). Artificiell intelligens började bli en del av speldesign senare under 1970-talet. På 1980-talet publicerades Pacman-spelet där artificiell intelligens används för att navigera igenom en labyrint för att fånga spelaren (Tommy Thompson. 2014). Med tiden har artificiell intelligens blivit bättre och bättre på att kunna utföra diverse uppgifter. En av de mer populära användningarna av avancerad AI är i spelet Alien: Isolation, där AI består av två hjärnor som kommunicerar med varandra för att göra beslut (Laura E. Shummon Maass & Andy Luc. 2019). Med tiden kommer AI troligen vara på en sådan nivå att den kan spela som en människa. Det finns redan exempel på var AI har börjat vinna över människor i allt mer komplicerade spel. Som exempel Go, där Google byggde en AI som vann över en världsmästare (BBC. 2019). I dagens värld finns det olika inlärningsmetoder som kan användas för att lära in AI i Unity.

1.2 Syfte & mål

Syftet med examensarbetet är att utforska förstärkningsinlärning och imitationsinlärning i Unity samt att vägleda nya spelutvecklare att få den information de behöver för att kunna besluta vilken AI-inlärningsmetod passar deras behov. Syftet är även att jämföra förstärkningsinlärning och imitationsinlärning i praktiken i en testmiljö samt att utvärdera dess för och nackdelar.

Målsättning med arbetet är att man skall förstå vad dessa inlärningsmetoder går ut på och veta hurdana möjligheter det finns för att kunna träna in AI i Unity. Det är meningen att efter genomläsning av arbetet skall man kunna förstå vilka inlärningsmetoder som är lämpligast för ens uppgift. Dessutom ska läsaren kunna fatta ett beslut om vilken inlärningsmetod är lämpligt för ens ändamål.

1.3 Metoder

Teoretiska delen av arbetet baserar sig på systematisk litteraturöversikt på maskininlärningsmetoderna förstärkningsinlärning och imitationsinlärning.
Praktiska delen av arbetet har utförts med hjälp av Unity:s spelmotor i en 3D-testmiljö. I 3D-testmiljön har det utförts tester på maskininlärningsmetoderna förstärkningsinlärning, imitationsinlärning och en blandning av båda två.
Resultaten och jämförelserna är baserade och gjorda på AI inlärningsresultaten med hjälp av Tensorflow samt exempel på tidigare forskning inom maskininlärning. Tensorflow användes att visualisera data i dataflödesdiagram

1.4 Struktur

Resten av examensarbetet är strukturerat enligt följande; Kapitel 2 beskriver AI inlärningsmetoderna och vad de går ut på. Kapitel 3 beskriver maskininlärningsbaserade verktyg. Kapitel 4 berättar om hur man installerar ML-agents i Unity. Kapitel 5 berättar om hur man bygger upp en agent i Unity. Kapitel 6 görs jämförelse mellan förstärkningsinlärning och imitationsinlärning. Kapitel 7 innehåller slutsatsen av arbetet.

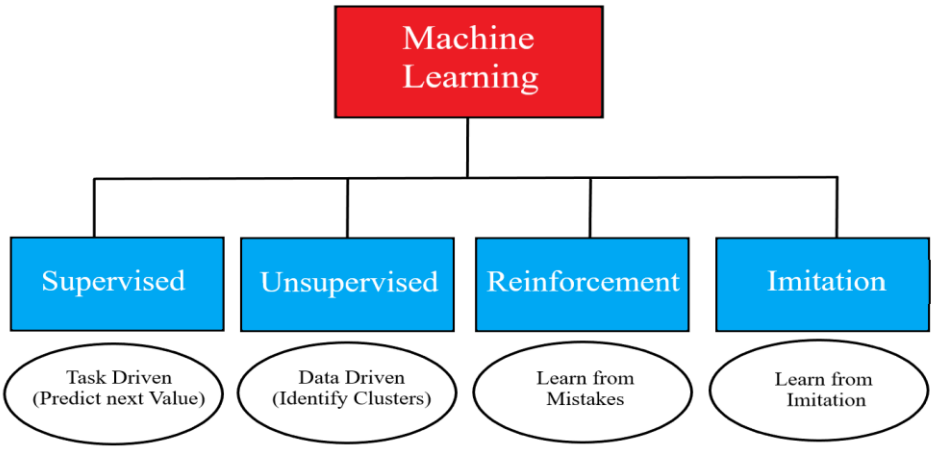
2 MASKININLÄRNINGSMETODER

2.1 Introduktion

Maskininlärning är en teknologi, som ger datorer förmågan att lära sig utan behov av hårdkodade kommandon. Huvudsaken med det är att hitta mönster i stora mängder med data, varefter maskinen skall kunna utföra prognoser och förutsägelser baserat på informationen. Maskinen kan sedan utföra både enkla och mer avancerade uppgifter.

Arthur Lee Samuel var en amerikansk pionjär inom AI och datorspel. Arthur Samuel är personen som uppfann termen ”machine learning” 1959 och gjorde konceptet mer populärt. Dessutom bör man inte glömma bort att Alan Turing hade redan tidigare programmerat en schackalgoritm ”Turochamp” 1948 som klarade av att spela på en låg nivå mot mänskliga spelare. Algoritmen klarade av att kalkylera potentiella speldrag samt potentiella spelares drag i respons. Arthur Samuel lyckades skapa självlärande program som klarade av att spela dam. Maskinens huvudsakliga drivkraft var ett sökträd över kontrollpositioner som var nåbara från det nuvarande tillståndet. Bägge två är pionjärer inom maskininlärning. (Wiederhold, McCarthy & Feigenbaum)

Det finns många olika typer av maskininlärningsmetoder av vilka fyra kommer att gås igenom. Av dessa fyra är tre huvudtyper av maskininlärning. Förstärkningsinlärning är en av de tre huvudtyperna av maskininlärning där en agent blir belönad genom att avklara sin givna uppgift. Övervakat lärande är den andra huvudmetoden och baserar sig på uppgiftsdriven markerad data. Oövervakat lärande är den tredje huvudmetoden och baser sig på omarkerade datamängder som klustras tillsammans genom att använda identifikationsalgoritmer. Imitationsinlärning baserar sig på data som utförs av en person. Detta illustreras i Figur 1.(Dwivedi. 2018) Originalfiguren har modifierats för detta arbete.



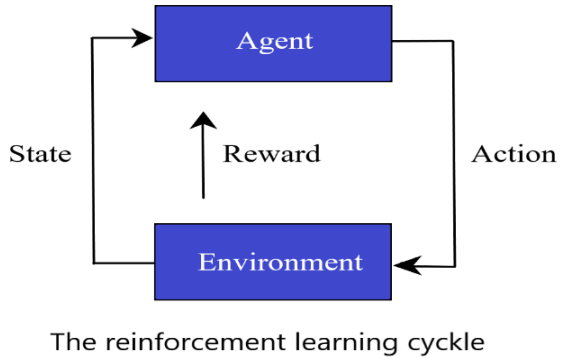
Figur 1. Typer av maskininlärningsmetoder

2.2 Förstärkningsinlärning

Förstärkningsinlärning är metoden att använda en agent som utför en sekvens av beslut för att uppnå ett givet mål i en miljö. Agenten genomgår en process med försök och misstag för att komma på bästa möjliga lösning på ett problem. När agenten utför rätta åtgärder skall den bli belönad medan om agenten gör fel åtgärder blir den bestraffad med negativa belöningar. På det här viset kommer agenten göra det som programmeraren vill. Förstärkningsinlärning är en kraftfull inlärningsmetod som kan träna en agent flera års erfarenhet inom en kort tids intervall. Förstärkningsinlärning låter agenten, på ett kreativt sätt, ta kontroll över de beslut den kommer att att utföra

Det självständiga forskningsinstitutet för AI OpenAI utvecklade 2018 AI-drivna bottar som spelade Dota 2 och lyckades vinna mot ett semi-professionellt lag. Dota 2 är ett spel där två lag på fem spelar mot varandra. Laget som OpenAI spelade mot högre globalt sett till absoluta toppen (enbart 1% av existerande lag höll samma nivå). AI-bottarna var tränade med hjälp av förstärkningsinlärning där AI:n spelade mot sig själv miljontals gånger. Varje dag som AI:n spelade, fick den ungefär 180 år av erfarenhet i människoår. Målet med AI:n var inte att bevisa att den kan vinna mänskliga spelare för det har redan bevisats tidigare. Målet var att utveckla AI för att se hur den klarar av att jobba tillsammans som ett team. Dota 2 användes som ett experiment för att tillämpa maskininlärning samt att fånga den verkliga världens oförutsägbarhet. (Simonite. 2018)

Förstärkningsinlärning består av tre huvudkomponenter. Handlingar ”*Action*” som representerar de interaktioner och allting AI agenten kan göra. Belöningar ”*Reward*” som ger feedback till agenten och ”*State*” observationer som agenten utför i miljön. Förstärkningsinlärning har två komponenter till som behandlar huvudkomponenter. Agenten, vars uppgift är att fatta beslut i inlärningsmodellen. Miljön ”*Environment*” som agenten inverkar på. Detta demonstreras i Figur 2. (Github. 2021, Background-Machine-Learning)



Figur 2. Förstärkningsinlärning

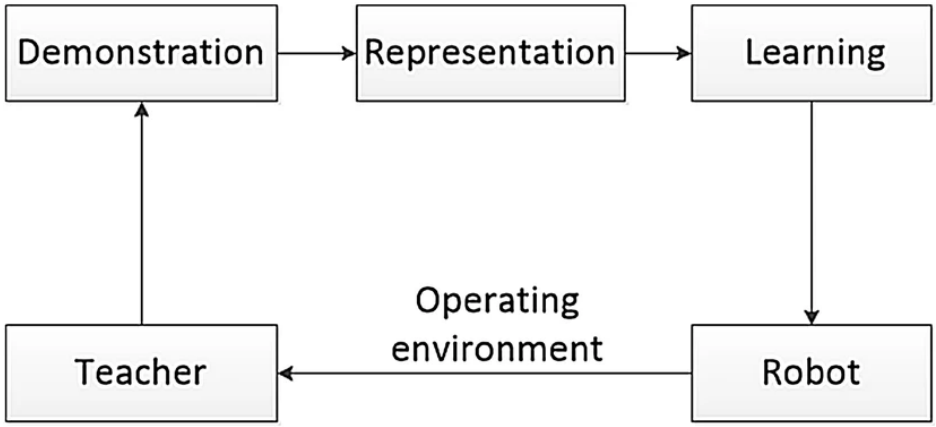
2.3 Imitationsinlärning

Imitationsinlärning är då en agent blir matad direkt information om miljön den skall jobba i och försöka imitera det. Istället för att försök lära sig av belöningar eller från markerad data, ger en expert (vanligtvis en människa) demonstrationer på vad som skall göras. Agenten försöker sedan lära sig det som har demonstrerats genom att imitera expertens utförande.

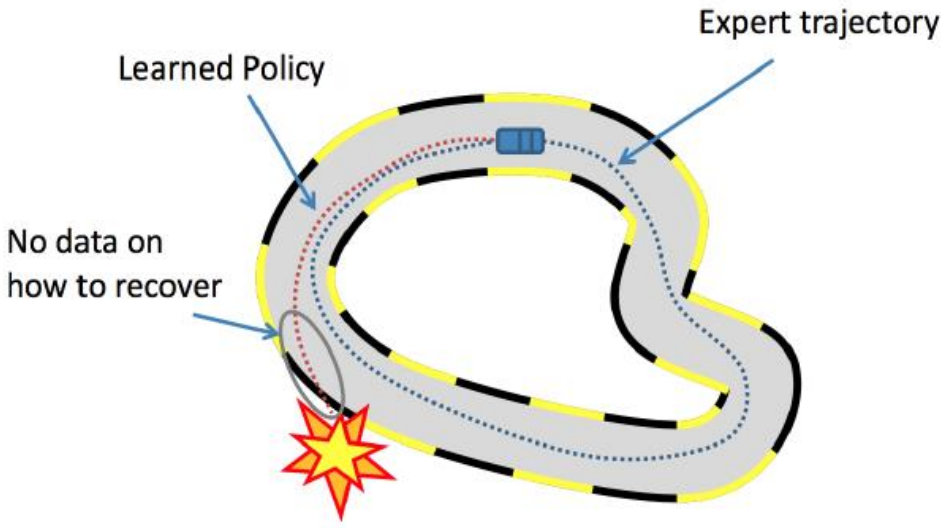
Som exempel på imitationsinlärning vore en testmiljö där agenten kommer att få en uppgift som den skall utföra. En expert (Människa) kör demonstrationer på uppgiften som skall utföras, till exempel lyfta en låda och bära den till ett mål. Dessa demonstrationer sparas i en datamängd som sedan används för att träna agenten. Detta demonstreras i figur 3.(SpringerLink. 2019).

Ett speciellt undantag som kommer till imitationsinlärning är om agenten måste göra något som den inte blivit inlärd i. Det brukar sluta med att agenten inte vet vad den skall göra samt blir uppgiften inte slutförd. Lösning till problemet är att någon expert

(människa) måste utföra fler demonstrationer vad agenten skall göra i en sådan situation var den inte vet vad den skall göra. Dessa scenarion händer i undantagsfall om miljön har ändrats sig. Detta demonstreras i figur 4. (SmartLabai. 2019)



Figur 3. Imitationsinlärning



Figur 4. Ett fall där imitationsinlärning misslyckas.

3 MASKININLÄRNINGSBASERADE AI-LÖSNINGAR FÖR UNITY

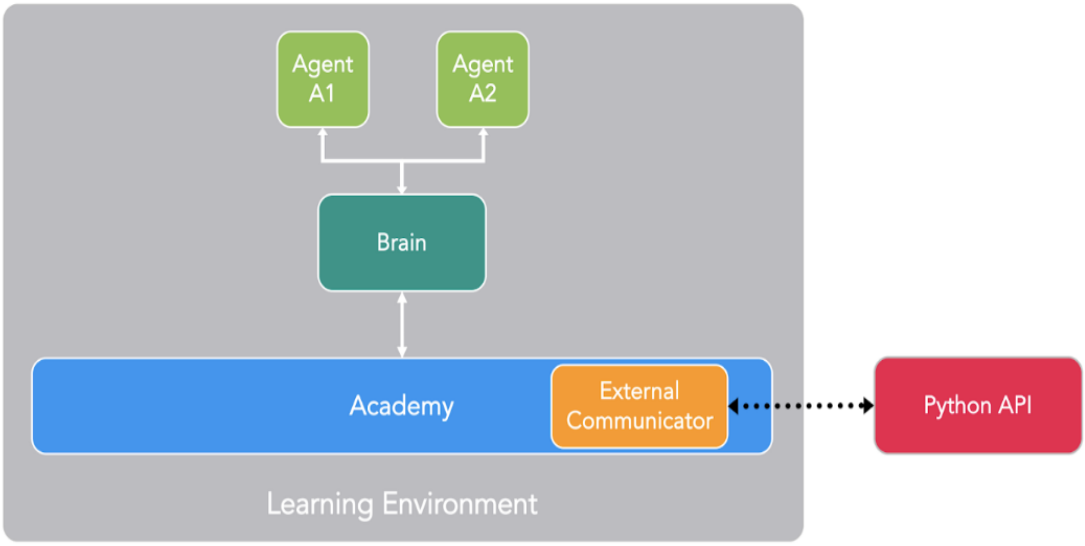
3.1 Unity ML-Agents Toolkit

Unity ML-Agents Toolkit är en instickningsmodul med öppen källkod för Unity som möjliggör implementering av maskininlärningsbaserad AI i en 3D-miljö i Unity. Användare kan lätt implementera Python API för att träna ML-agenter i 3D-miljön. Dessa ML-agenter kan tränas med hjälp av förstärkningsinlärning, imitationsinlärning eller någon annan inlärningsmetod för maskinlärning. Unity ML-Agents Toolkit erbjuder en uppsättning av funktioner som möjliggör att definiera en miljö med hjälp av Unity Editor och C# skript som sedan kan exponera miljön för interaktion med Python API. (Simonini 2020)

Unity ML-Agents Toolkit är baserad på TensorFlow som är ett ramverk för maskininlärning gjort av Google. TensorFlow underlättar processen att hämta in data, träningsmodeller, prognoser samt man behöver inte oroa sig för komplexa algoritmer. (Yegulalp, 2022).

3.2 ML-Agent SDK

ML-Agent SDK består av tre enheter (Agent, Hjärna och Akademi). Genast som man laddat upp SDK filerna till ett Unity-projekt kan användaren ändra på scenerna i en inlärningsmiljö för ML-agenten. Figur 5 visar hur denna miljö ser ut. (Simonini 2020)



Figur 5. Inlärningsmiljö som innehåller agenter, hjärna och akademin.

Första enheten i en ML-Agent SDK är en agentkomponent. Agenten själv är ett spelobjekt i en spelscen eller miljö. Spelobjektet visar var ens agent är i miljön, annars skulle man inte kunna se den. Agent-spelobjektet används för att utföra handlingar inom inlärningsmiljön samt nödvändiga observationer för att fatta beslut. Agenten är den som vi kommer lära in när det kommer till maskininlärning.

Den andra komponenten är hjärnan. Hjärnkomponenten används för att fatta alla beslut för agenten. Utan hjärnkomponenten är agenten hjärndöd och skulle inte kunna fatta beslut. Varje agentkomponent är kopplad till en enda hjärna som hanterar dess beslut. Det är möjligt att flera agentkomponenter kan använda samma hjärna, men en agent kan inte ha flera hjärnor. Själva Hjärnbeteendet bestäms av hjärntypen som ML-agenten kan definiera som fyra hjärntyper. Dessa hjärntyper kallas för Spelare, Heuristisk, Internt och Extern och beskriver hur man lär in hjärnan. Med Spelare så lär en person in hjärnan, med Heuristisk så har man ett definierat skript som berättar vad som skall göras, Internt är ett inbäddat neuralt nätverk som det agenten lär sig ifrån och Extern interaktion via Python API.

Den tredje komponenten är akademin som används inom en scen för att följa med stegen agenten gör. Akademin ger basfunktioner som ger användaren förmågan att kunna starta

om miljön, modifiera miljön för det syfte man vill, samt kunna försnabba inlärningsprocessen genom att duplicera den eller höja på simulationens hastighet.

3.3 TensorFlow

Tensorflow är ett öppet källbibliotek som används för att utföra beräkningar med dataflödesdiagram. Tensorflow tillåter användare att fokusera på logiken i sina applikationer och de behöver inte fundera på komplexa algoritmer. Tensorflow kommer med ett verktyg som kallas för Tensorboard. Detta verktyg sparar data och statistik från varje träningssession som sedan kan kollas på med Tensorboard. Med Tensorboard kan man få statistik om agenten hade problem med att lära sig under träningssessionen, hur länge det tog för agenten att lära sig, hur agenten blev belönad och mer info om vad som händer under träningsprocessen. Detta illustras i Figur 6. (Github. 2021, Using TensorBoard to Observe Training).

- Miljö/kumulativ belöning – Den genomsnittliga kumulativa episodbelöningen för alla agenter. Bör öka under ett lyckat träningspass.
- Miljö/avsnittslängd – Medellängden av varje avsnitt i miljön för alla agenter.
- Miljö/Lektion – Ritar framsteg från lektion till lektion.
- Förluster/Policyförlust – Hur mycket processen för att besluta om åtgärder förändras. Minskar under ett lyckat träningspass.
- Förluster/Värdeförluster – Medelförlusten av värdefunktionens uppdatering. Korrelerar med hur väl modellen kan förutsäga värdet av varje tillstånd. Detta bör öka medan agenten lär sig, och sedan minska när belöningen stabiliseras
- Policy/Entropi – Hur slumpmässiga besluten i modellen är. Bör sakta minska under en framgångsrik träningsprocess. Om den minskar för snabbt bör man justera hyperparametrarna ökas.
- Policy/Inlärningshastighet – Hur stort steg träningsalgoritmen tar när den söker efter den optimala policyn. Bör minska med tiden.
- Policy/Värdeuppskattning – Uppskattning av medelvärde för alla tillstånd som besökts av agenten. Bör öka under ett lyckat träningspass.

Figur 6. Tensorboard datagrafer.

4 INSTALLATION AV ML-AGENT VERKTYGET

För att kunna använd ML-Agents Toolkit behöver man installera dessa program och komponenter.

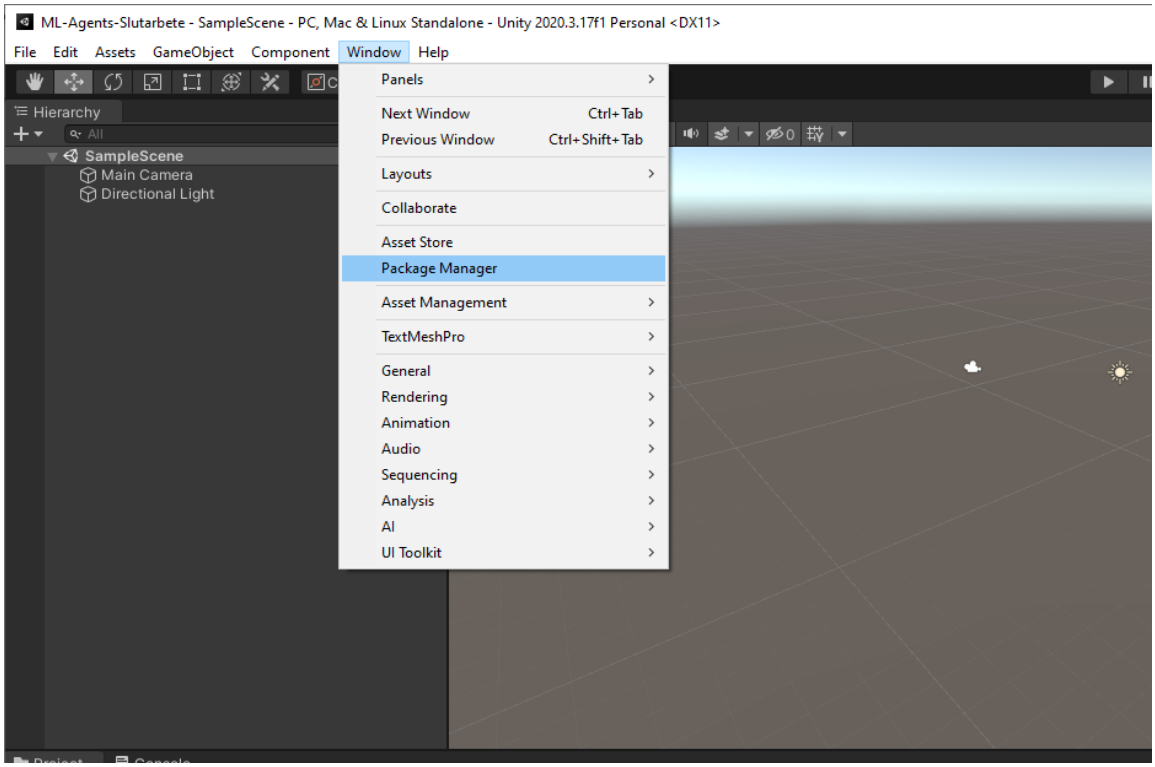
- Unity (2020.3 eller nyare version)
- Python (3.6.1 eller nyare version)
- Klona Unity-Technologies egna förvar (Frivilligt)
  - Obs: Om du inte klonar Unitys filer så kommer man inte åt exempelmiljöerna och träningskonfigurationerna och Unitys guides baserar sig på att man laddar ned deras filer.
- Unity paket (com.unity.ml-agents).
- Tre stycken Python paket (mlagents, mlagents_envs, gym_unity).
- Unity nyaste paket (com.unity-agents.extensions).

Unity Hub är en applikation som används för att hitta, ladda ner, spara, hantera och installera Unity projekt. Unity Hub används med för att dela projekt och underlätta arbetsflödet mellan arbetskollegor. Som en extra fördel kan man även installera olika versioner av Unity i Unity Hub. (Unity Technologies. 2022, Every creative journey starts with Unity Hub).

När man installerar Unity så rekommenderas det att göra det via Unity Hub eftersom man då kan använda olika versioner av Unity. Att installera själva ML-Agents Toolkit är enkelt. Unity har skapat en förvaringsplats på Github som innehåller projekt, ML-agentverktygen samt dokumentation om hur man installerar de obligatoriska paketen som behövs. För att komma igång laddar vi ned ML-agentfilerna från Unity´s Github. Skriv följande text inuti kommandtolken ” *git clone --branch release_19 https://github.com/Unity-Technologies/ml-agents.git*”.

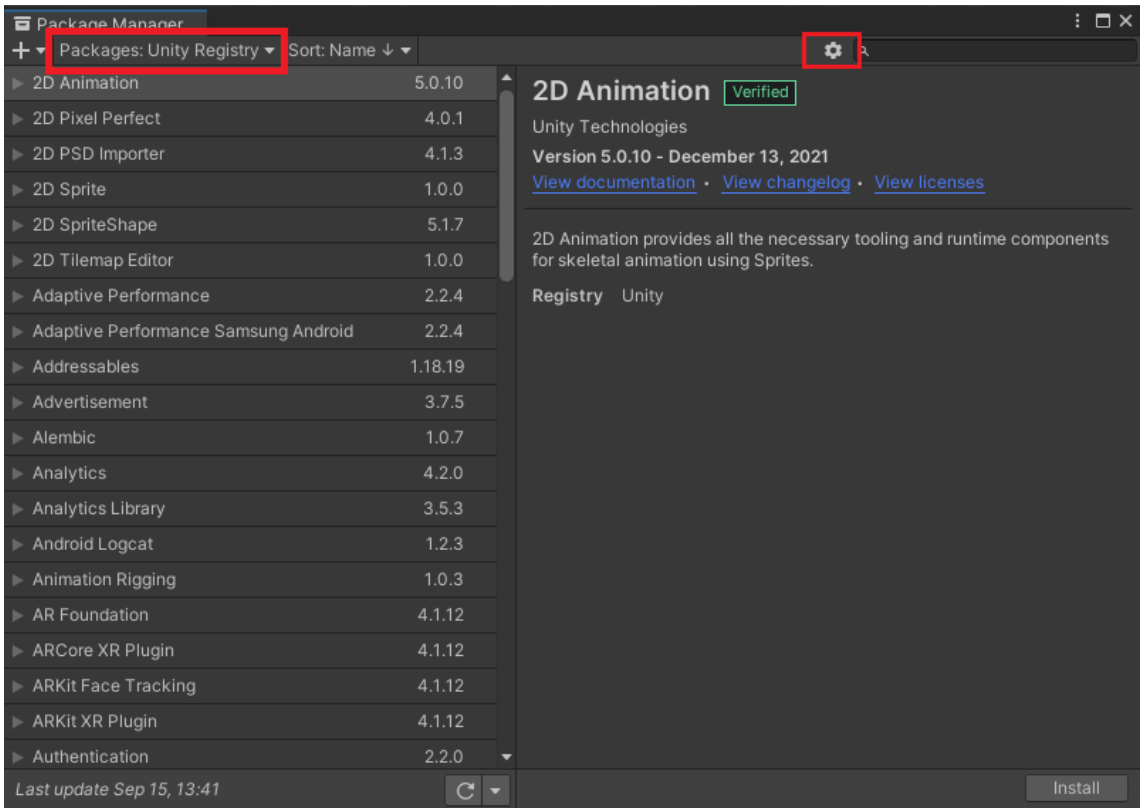
För att kunna ladda ner filerna via Git-terminalen krävs att själva Git-verktyget först är installerat på datorn. Om man inte vill använda Git kan man också ladda ned filerna direkt från Unitys Github som en Zip-fil. Kom ihåg vart dessa filer har sparats för man kommer att söka en fil från dem senare. När man har filerna på datorn öppna Unity Hub och skapa ett nytt 3D projekt. När 3D projektet är skapat vill man öppna det och navigera till

pakethanterare (Package manager). Man hittar det under Window uppe i vänstra hörnet. Detta illustreras i Figur 7.

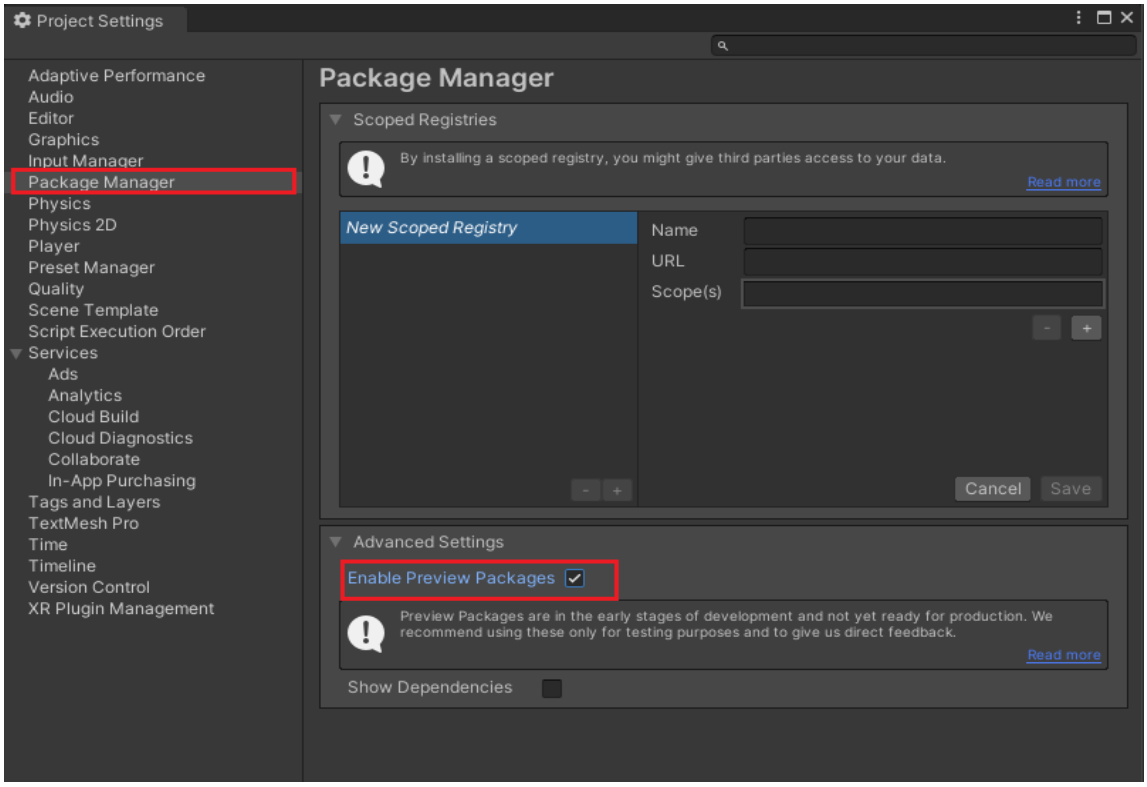


Figur 7. Unity pakethanterare

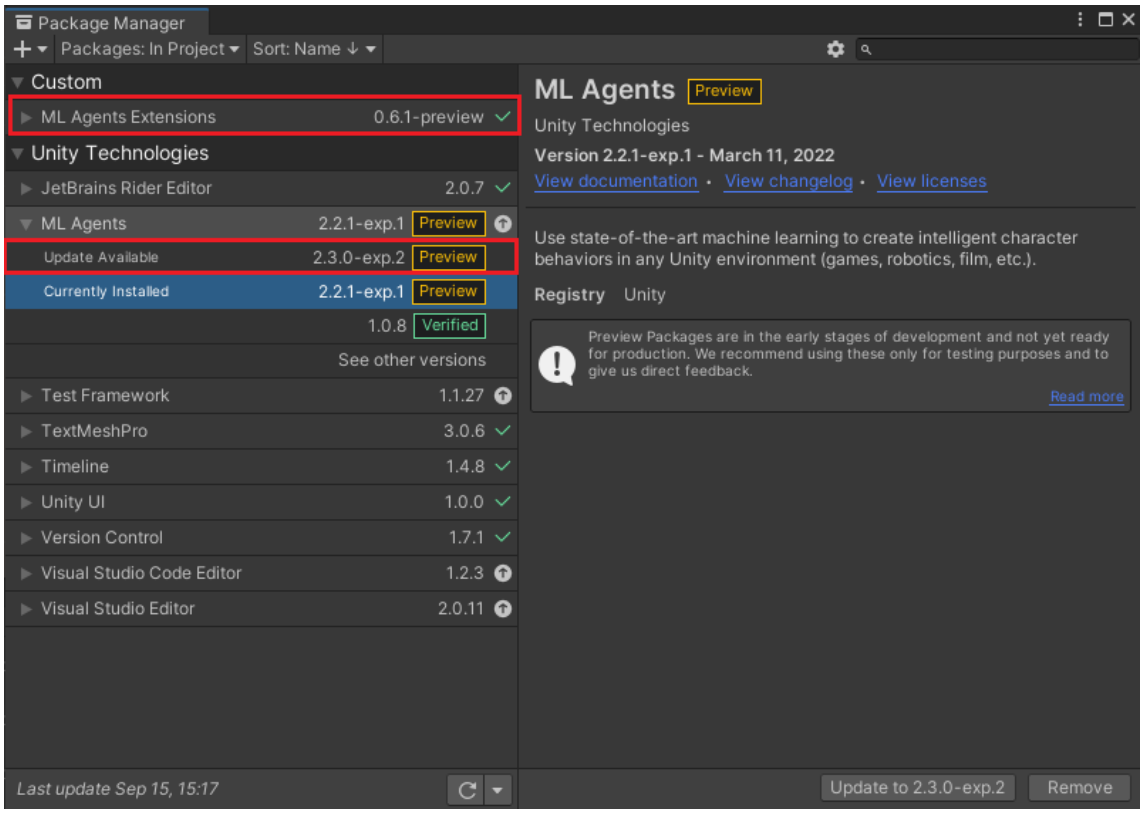
Efter att man öppnat ”Package manager” trycker man på ”Packages” knappen uppe i vänstra hörnet och ändra på innehållet till Unity Register (Unity Register). Paketet som skall installeras finns i förhandsvisning så vi måste aktivera förhandsgranskningspaketen. För att komma åt dessa inställningar vill vi trycka på kugghjulet (symbolen för inställningar) och aktivera förhandsgranskningspaketen. Kugghjulsikonen hittas uppe i högra hörnet. Efter att vi har gjort dessa ändringar vill vi söka igenom paketen och nyaste versionen av ML-Agents. Till följande installera Unity agenttillägg (Unity agents Extensions) var som först vill ni ändra på paket register tillbaka till i projekt (In Project). Tryck på pluset uppe i högra hörnet och välj lägg till paket från disk (add packages from disk) där vi söker efter JSON-FILER från ML-agent filer. Filens namn är ” com.unity.ml-agents.extensions”. Efter detta kan det löna sig att starta om Unity så att alla installationer får en omstart. Detta illustras i Figur 8, figur 9 och figur 10.



Figur 8. Unity Register



Figur 9. Aktivering av förhandsgranskningspaketen.



Figur 10. Pakethanteraren efter installation.

Till följande söker man efter exempelmiljöerna som finns inuti ML-agent mappen som vi klonade. Dessa filer finns sparade på denna stig på din dator *”<din lokala väg>\ml-agents\Project\Assets*”. När ni har hittat denna mapp på datorn så kopiera ML-agent mappen in till Unity 3D projekt. Efter att ni har kopierat filerna in till 3D projektet kommer det troligen ge ett felmeddelande som måste fixas eftersom det annars kommer att leda till problem. Problemet beror på en av testmiljöerna som laddades ner, och den lättaste lösningen är att radera bort miljön. Testmiljön som vi raderar hittas enligt denna dator väg i ditt Unity 3D projekt ”*Assets/Ml-agents/Examples*” och vi raderar ” *PushBlockwithInput*”. Nu borde felmeddelandet vara borta och vi kan testa en testmiljö för att verifiera att det fungerar genom att öppna en scen och trycka på spela-knappen.

4.1 Installation av Anaconda Python

För att starta processen krävs det en Python miljö på datorn och en möjlighet är att använda Anaconda, som är en öppen källkoddistribution av Python som kan användas för vetenskapliga beräkningsmetoder inklusive maskininlärning. När Anaconda har installerats är det en bra idé att kontrollera om installationen har lagt till de obligatoriska systemmiljövariablerna, annars kan det leda till problem. Ett bra exempel som man kan testa är att skriva ”Conda” i Anaconda terminalen och om det kommer ett felmeddelande så lönar det sig att fixa genom att skriva i sökfältet redigera systemmiljövariablerna (Environment Variables) och dubbelklicka väg (Path) och klicka på ny och lägga till dessa nya vägar. Detta illustras i Figur 11.

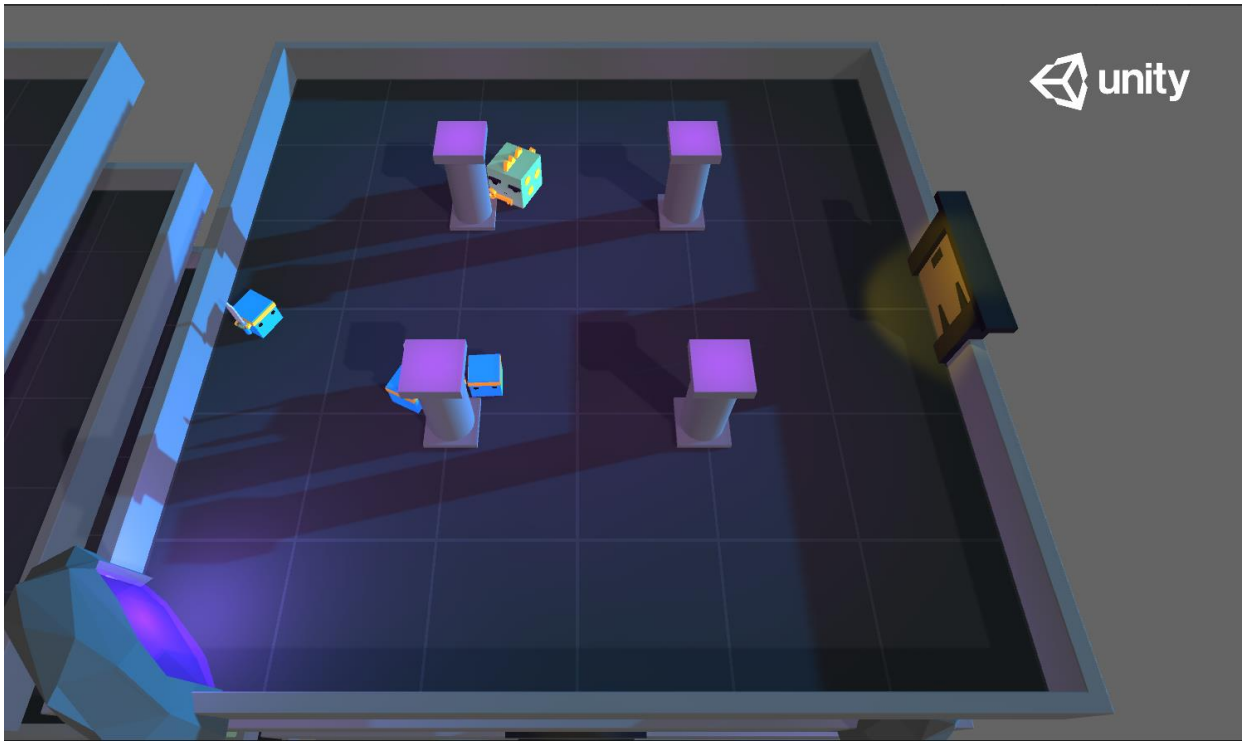
```
%UserProfile%\Anaconda3\Scripts
%UserProfile%\Anaconda3\Scripts\conda.exe
%UserProfile%\Anaconda3
%UserProfile%\Anaconda3\python.exe
```

Figur 11. Anaconda miljövariabler

Följande steg är att skapa en ny Anaconda miljö. För att skapa en ny miljö öppna Anaconda terminalen och skriva ” *conda create -n ml-agents python=3.6*” när terminalen frågar om man vill installera dessa paket vill man välja ja. För att kunna använda denna miljö kommer terminalen att ge två kommandon för att kunna aktivera respektive avaktivera miljön. Dessa två kommandon är ”*conda activate ml-agents*” och ”*conda deactivate*”.

Till sist skall man installera några Python paket genom att skriva dessa kommandon inne i Anaconda terminalen ”*pip install mlagents*” och ” *pip install tensorflow*”. Dessa kommandon installerar ML-agent paketen som behövs för att kunna kommunicera mellan Python och Unity samt Tensorflow för att skapa modeller och grafer. Om man skriver ”*conda activate ml-agents*” och ”*mlagents-learn*” så startar vi inlärningsprocessens varefter terminalen kommer att säga att gå till Unity och tryck spela knapen för att börja träna. Man kan välja någon testmiljö och tryck på spela-knappen för att börja inlärningen. Dessa Unity miljöer kan användas för att utföra tester. Detta illustras i figur 12 och figur 13.

Figur 12.ML-agent börjar lära sig.



Figur 13.Scen från Unity Dungeon Escape.

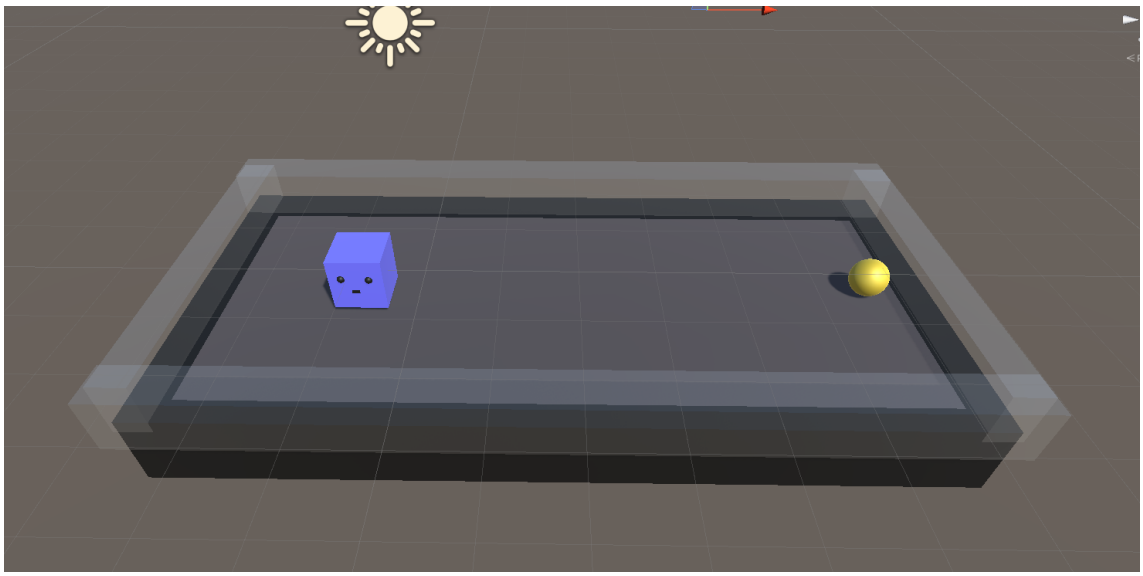
5 INLÄRNING AV AGENTER

Vid den här fasen valde jag att skapa ett nytt 3D-projekt. Orsaken var att jag inte ville använda Unitys färdigt tränade modeller utan istället själv träna en egen ML-agenthjärna. Detta innebär att man måste köra installationsfasen på nytt i det nya 3D projektet. Jag ville testa använda Python utan Anaconda, eftersom Unity föreslår att använda traditionell Python. Av detta skäl förklaras det hur man får det och fungera med Python. I det här fallet använde jag en äldre version av Python (3.7.9) eftersom den nyaste Python inte ännu har fullt stöd till ML-agentmodulerna.

5.1 Skapa en ny ML-agentmiljö

Det första vi gör är att skapa ett nytt 3D-projekt i Unity. Efter detta ska vi öppna kommandotolken och gå in till det nya Unity-projektet. I kommandotolken skriver vi kommandot ”*python -m venv venv*” för att skapa en virtuell miljö. Därefter skriver vi ”*venv\Scripts\activate*” för att aktivera miljön. Sedan skriver vi ”*python -m pip install –upgrade pip*”. Efter att vi har installerat nyaste versionen av pip skriver vi ” *pip3 install torch~=1.7.1 -f https://download.pytorch.org/whl/torch_stable.html*”. Pytorch är ett öppet källkodsbibliotek för att utföra beräkningar med hjälp av datagrafer, likt Tensorflow men mer objektorienterat. Efter att installationen är klar behöver vi ännu installera ML-agents genom att skriva ”*pip install mlagents*”. Nu borde allt vara färdigt installerat och vi kan lämna terminalen öppen. Nästa steg är att gå till vårt Unity 3D projekt och öppna scenen och bygga upp en testmiljö som vår agent skall spela i. Jag har använt Unitys basprojekt som botten men gjorde modifieringar till det för mitt syfte. Man bör inte glömma bort att installera ML-agent modulen i Unity, så vi behöver navigera upp till vänstra hörnet och trycka på ”*Window>Package Manager*” som kommer att öppna ett nytt fönster där vi navigerar upp till vänstra hörnet och väljer ”*Package >Unity Registry”* för att hitta ML agent paketen. Unity kommer att rekommendera version 1.0.6 men man bör inte använda den eftersom det kommer att leda till problem, och därför navigerar vi upp till högra hörnet och trycker på kugghjulet där vi väljer ”Advanced Project Settings” och väljer ”*Enable Preview Packages*”. Nu om man går tillbaka och kollar på ML agentmodulerna så kan vi se fler alternativ där vi behöver installera nyaste versionen. För att verifiera att installationen fungerar kan man skapa ett nytt objekt i

testmiljön och lägga till en komponent och söka efter ML-agent där Unity kommer att föreslå olika alternativ, vilket innebär att installationen fungerade. Själva testmiljön består av några objekt. Det första är den blåa kuben som är vår agent, den gula sfären är målet, plattformen som sfären och agenten står på och till sist har vi genomskinliga väggar för att hålla agenten innanför testmiljön. Agentens uppgift blir hitta gula sfären för att få en belöning och om agenten träffar en vägg börjar spelet från början. Detta illustreras i figur 14 samt kolla Figurerna 8–10



Figur 14. AI testmiljö.

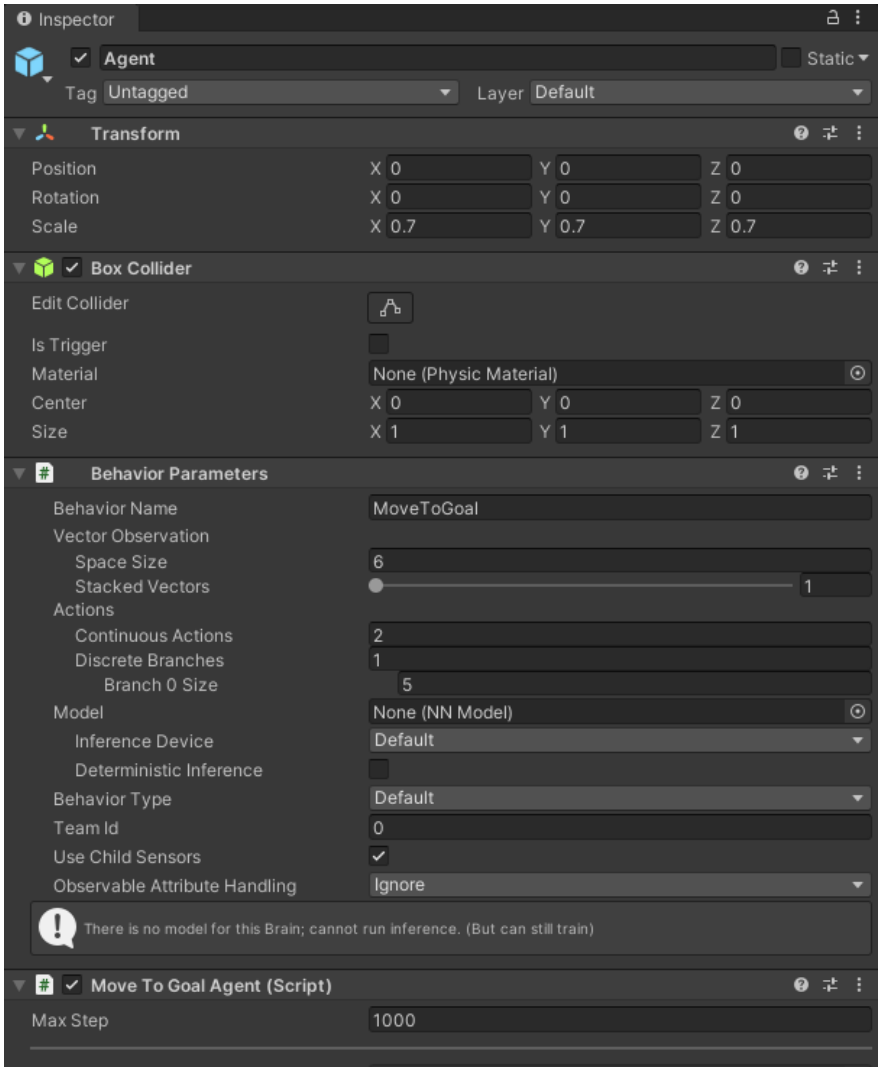
5.2 Förstärkningsinlärning i Unity

Vid denna fas har vi en testmiljö och nästa steg blir att skapa logiken för agenten. Vi ska som följande steg skapa ett nytt C# skript i vårt 3D projekt. I C# skall man ta bort bassstrukturen som ges åt oss och istället skriva det som illustreras i figur 15.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using Unity.MLAgents;
5   using Unity.MLAgents.Actuators;
6
7
8   public class MoveToGoalAgent : Agent{
9
10      public override void OnActionReceived(ActionBuffers actions) {
11          Debug.Log(actions.DiscreteActions[0]);
12      }
13
14  }
```

Figur 15. Början på ett nytt script.

Kommandona ”*using.Unity.MLAgents & using Unity.MLAgents.Actuators*” behöver vi för att kunna använda ML-agents i skriptet. Samt vill vi ärva från våran agent istället för ” *MonoBehaviour*”. Som följande steg behövs att agenten skall utföra förstärkningsinlärning som innebär att agenten skall kunna observera, göra beslut, utföra handlingar och få belöning. Vi börjar med ”*Actions*” och använder oss av ”*ActionBuffers*” som kommer att innehålla agentens beslut. Det kan löna sig att tänka på hur maskininlärning fungerar bara med siffror som betyder att maskinen inte har fullständig förståelse för spelobjekt eller vad det betyder att röra på sig i testmiljön. Därför har vi ”*Debug.Log*” för att printa ut dessa siffor. Till nästa går vi tillbaka till testmiljön och lägger till vårt script till agenten. Detta illustreras i figur 16.



Figur 16. Inspektör layout.

Det som vi vill se i Unity *”inspector”* är vårt skript. Dessutom kommer Unity att lägga till ”*Behavior Parameters*” skript som innehåller olika parametrar som vår AI använder. Följande steg så byter vi namnet på ”Behavior Name” till ”*MoveToGoal*”. Före vi börjar skriva mera kod till skriptet är det bra om vi testar att allting fungerar så borde man ändra på dessa parametrar ”*Space Size: 1, Continuous Actions: 1, Discrete Branches: 1, Branch 0 Size: 5* ”.

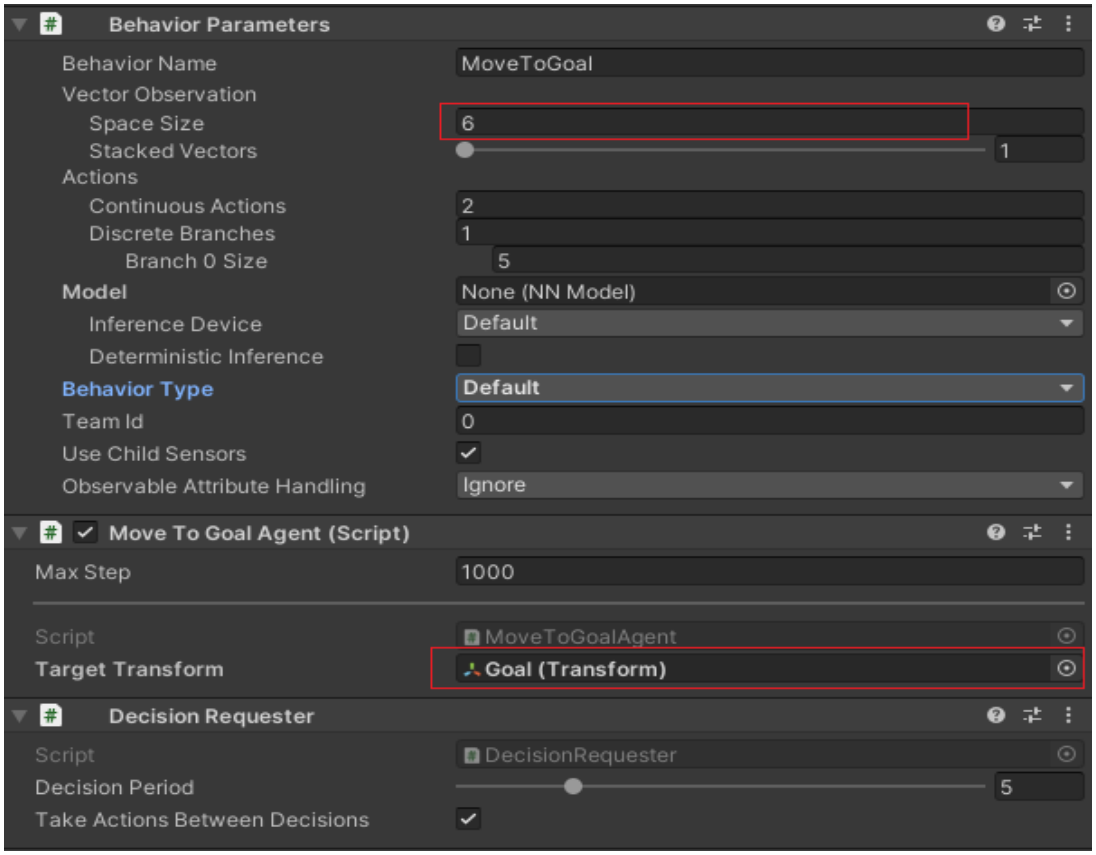
- Diskret består av heltal ”*int*”.
- Kontinuerlig består av decimaltal ”*floats*”.
- Space storlek handlar om x, z, y vektorvärden.
- Varje ” *branch*” kan ha sin egen storlek beroende på uppgiften som exempel gå framåt, bakåt eller stå stilla, eller gå snabbare, långsammare.

Om vi vill testa att miljön fungerar, så kan vi printa ut ”*DiscreteActions[0]”* eftersom vi bara har en ”*branch*”. Före vi kan testa att miljön fungera behöver vi ännu lägga til”*Decision Requester*” detta skript kommer begära ett beslut efter att en given tid har gått och sedan göra en åtgärd. Lägg till skriptet genom att klicka på ”*Add Component*” och sök under ML-agent ”*Decision Requester”.* Efter detta behöver man gå tillbaka till kommandotolken och skriva ”*mlagents-learn*”. Om allt har gått rätt, så kommer Unity-logon att visas därefter kommer kommandotolken att begära om att trycka på play-knappen för att börja träna. Kolla illustration från figur 12.

Till nästa går vi tillbaka till skriptet och lägger till att vi kommer att använda ML-agent-sensor paketet. Sedan skapar vi en ny metod som kommer att observera och ge data till agenten om var den befinner sig och om var målet finns. Därefter skapas en ”*targetTransform*” som hittas i Unity ” *Inspector*” var vi vill lägga målobjektet, och ändra på ”*Space Size*” till 6. Detta beror på att agenten och målet ger oss x, y, z vektor som observeras, och vi behöver ha alla 3 koordinator från båda två. Detta illustreras i figur 17 och figur 18

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;


public class MoveToGoalAgent : Agent{

    [SerializeField] private Transform targetTransform;

    public override void CollectObservations(VectorSensor sensor) {
        sensor.AddObservation(transform.localPosition);
        sensor.AddObservation(targetTransform.localPosition);
    }

    public override void OnActionReceived(ActionBuffers actions) {
        Debug.Log(actions.DiscreteActions[0]);

    }


}
```

Figur 17. Observations kod.



Figur 18. Inspektör parametrar.

Nästa steg är att göra så att agenten kan röra sig så vi går tillbaka till skriptet och vi kan ta bort ”*Debug.Log*” och skapar x rörelse samt z rörelse. Följande steg blir att använda x- och z-variablerna i ”*transform.localposition*”. Vi vill inte glömma bort att ge agenten hastighet så vi skapar med en hastighetsvariabel. Med x , z och hastighets variablerna kan man i ”*transform.localposition*” skapa rörelse till agenten. Detta illustreras i figur 19.

```
19      public override void OnActionReceived(ActionBuffers actions) {
20          Debug.Log(actions.DiscreteActions[0]);
21          float moveX = actions.ContinuousActions[0];
22          float moveZ = actions.ContinuousActions[1];
23
24          float moveSpeed = 3f;
25          transform.localPosition += new Vector3(moveX, 0, moveZ) * Time.deltaTime * moveSpeed;
26      }
27
```

Figur 19. Rörelse kod.

Det nästa som behövs är ett belöningssystem för vår agent för att den skall kunna lära sig när den gör fel eller rätt. Vi kommer sätta att agenten blir belönad när den kolliderar med målet och blir bestraffad när den kolliderar med väggar. Man behöver komma ihåg att aktivera ”*Is Trigger*” på vår vägg samt målobjekt. Detta görs via Unity ” inspector” under ”*Collider*” och om ni inte har en ”*Collider*” på ert spelobjekt borde ni tillägga ett. Som följande skapar vi en ny metod i vårt skript som kommer att behandla varje gång agenten kolliderar med väggar eller målet. Koden belönar agenten och startar om spelet efter att agenten kolliderar med målet. På ett liknande sätt, om agenten kolliderar med väggen så blir den bestraffad och startar om spelet. För att starta om spelet behövs en till metod som kommer att flytta agenten och målet inom vissa koordinater för att ge mångsidighet för agenten för att den ska lära sig olika alternativ istället för att lära sig en och samma sätt. Detta illustreras i figur 20 och figur 21.

```
private void OnTriggerEnter(Collider other) {
    if (other.TryGetComponent<Goal>(out Goal goal)) {
        SetReward(+1f);
        EndEpisode();
    }
    if (other.TryGetComponent<Wall>(out Wall wall)){
        SetReward(-1f);
        EndEpisode();
    }
}
```

Figur 20. Belöningskod.

```
public override void OnEpisodeBegin() {
    transform.localPosition = new Vector3(Random.Range(-1f, +3f), 0 , Random.Range(-1f, +1f));
    targetTransform.localPosition = new Vector3(Random.Range(4f, 5.88f), 0 , Random.Range(-1.43f, +1.43f));
}
```

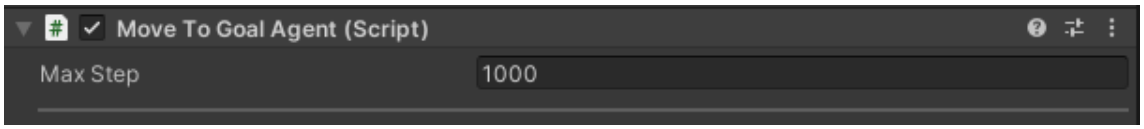
Figur 21. Omstarts kod.

För att kunna testa allting själv lades det en metod till in i koden som tillåter en person att kunna styra agenten själv via tangentbordets pilar. I metoden kommer vi att använda heuristisk när en person spelar och kontinuerlig när agenten spelar för att undvika att blanda mellan de två fallena. Detta illustreras i figur 22.

```
public override void Heuristic(in ActionBuffers actionsOut) {
    ActionSegment<float> continuousActions = actionsOut.ContinuousActions;
    continuousActions[0] = Input.GetAxisRaw("Horizontal");
    continuousActions[1] = Input.GetAxisRaw("Vertical");

}
```

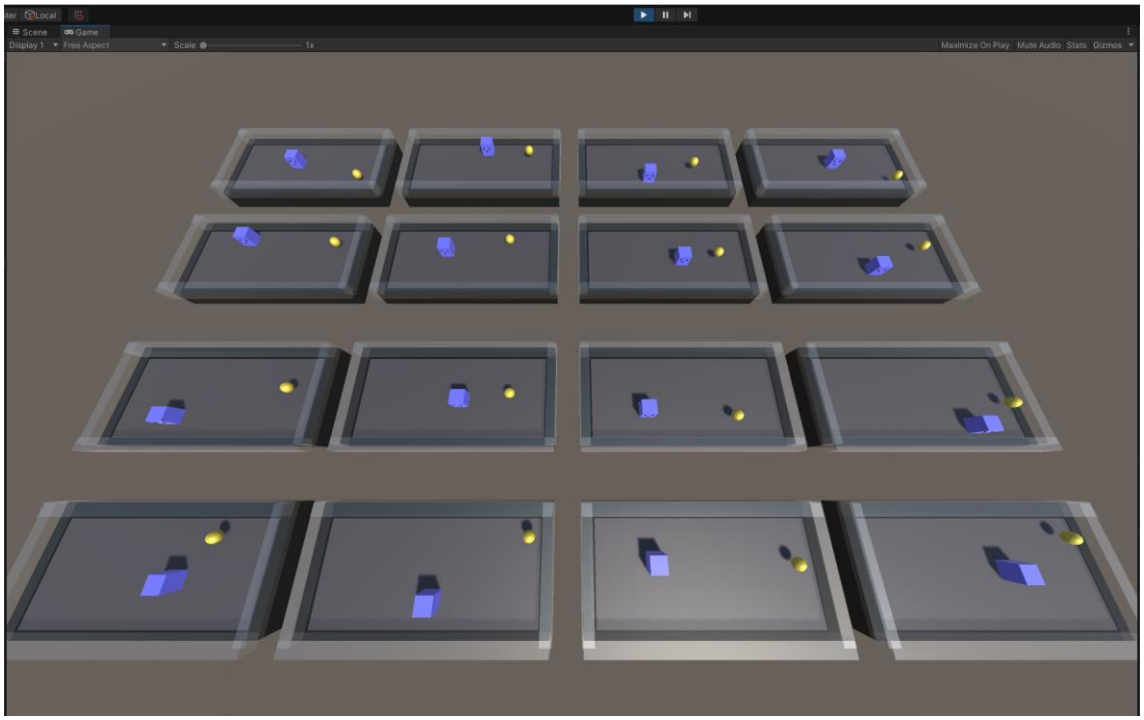
Figur 22. Koden för spelarkontrollerad styring

För att testa att en person kan spela och att miljön fungerar skall man återvända till testmiljön och välja vår agent. I agenten behöver man navigera till Unity *”inspector”* och klicka på ”*Behavior Parameter*> *Behavior type*” och välja heuristisk för att kunna spela själv. Klicka på spela-knappen för att börja spela och testa om man kan röra på sig och att spelet börjar från början när man kolliderar med väggar eller målet. Efter detta är vi redo för att aktivera ML-agenten och börja träna. Ändra ” *Behavior type*” tillbaka till ”*Default*” och gå till kommandotolken och skriv in följande kommando ”*mlagents-learn --run-id-TestLearn*”. Id blir namnet på inlärningsfilen. Om allting fungerar kommer kommandotolken att begär om en att klicka Unity spela-knappen. Nu borde vi ha en ML-agent som lär sig att spela och i början kommer agenten att försöka hitta något som belönar den. Däremot finns det en risk att agenten inte hittar någon positiv belöning utan endast negativa. Det kan sluta med att agenten börjar stå stilla för att undvika väggarna. Detta kan åtgärdas genom att öppna *”inspector”* och navigera till ”*Max Step*” där vi kan bestämma hur länge det får ta för agenten att utföra uppgiften, där vi använde 1000. Efter detta kan en episod inte vara för evigt. Detta illustreras i figur 23.



Figur 23. Max Step.

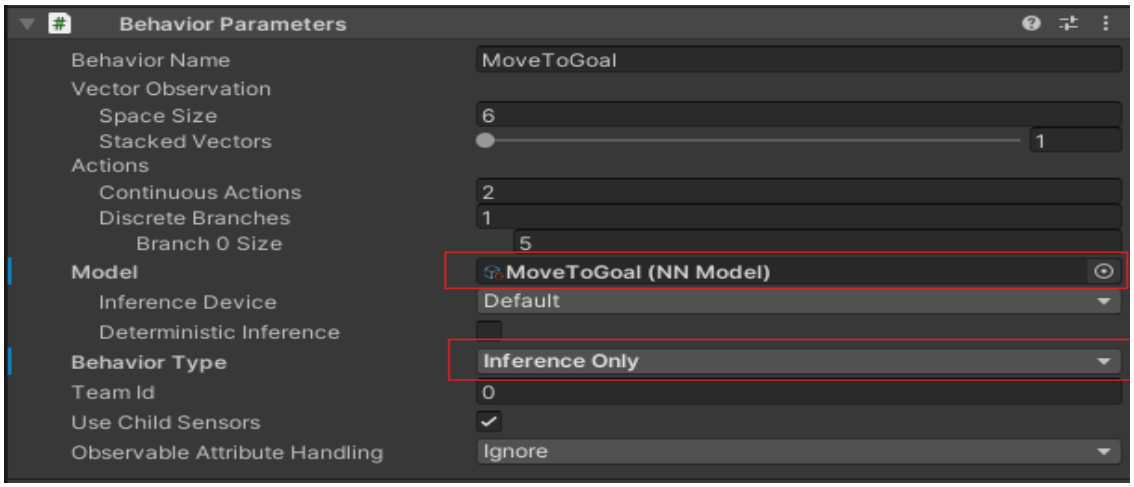
Om man vill försnabba inlärningsprocessen kan man skapa flera testmiljöer som utför samma uppgift. För att göra det här på det lättaste sättet bör man skapa ett nytt ”*Empty object*” och flytta alla spelobjekt in i den. Efteråt bör vi spara objektet som en ”prefab” i ”Assets”. Nu kan vi duplicera miljön till så många kopior som man vill använda. Dessutom användes ” *localPosition*” i koden för att undvika att man blir tvungen att skriva in deras startkoordinater manuell eller använda ”*transform.position*” eftersom alla agenter då hade börjat på samma position. Nu kan vi börja träna agent snabbare så vi skriver i kommandotolken ” *mlagents-learn --run-id-MoveToGoal*” och trycker på spela knappen i Unity. Efter detta kan vi börja träna agenten och spela tills den lärt sig att utföra uppgiften. Detta illustreras i figur 24.



Figur 24. Duplicerad testmiljö.

5.3 Använda ML-agenthjärnan

För att använda agenthjärnan kommer kommandotolken ge information om var filen är sparad. I det här fallet var den sparad i själva Unity projektet under resultat och namnet som vi gav filen i kommandotolken ”*MoveToGoal*”. Filen kommer att sparas som en ”*onnx*” fil och vi kan kopiera den till ”*Assets*” för att hitta den lättare. Nu kan man fästa filen på agenten genom att flytta filen in till inspektör under ”*Behavior Parameters*” och lägga den på ”*Model*”. Sedan vill man ändra på ”*Behavior Type*” till ” *inference*” för det betyder att agenten kommer använda inlärningsmodellen istället och inte börja träna. Om man nu trycker på spela knappen får man se resultat från sin tränade modell. Detta illustreras i figur 25.

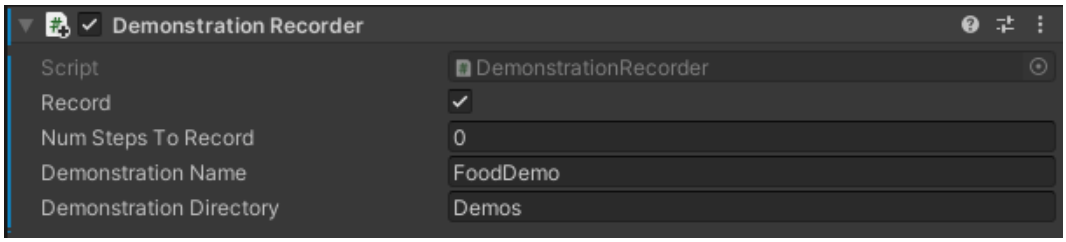


Figur 25. Fästa tränade hjärnan på agenten.

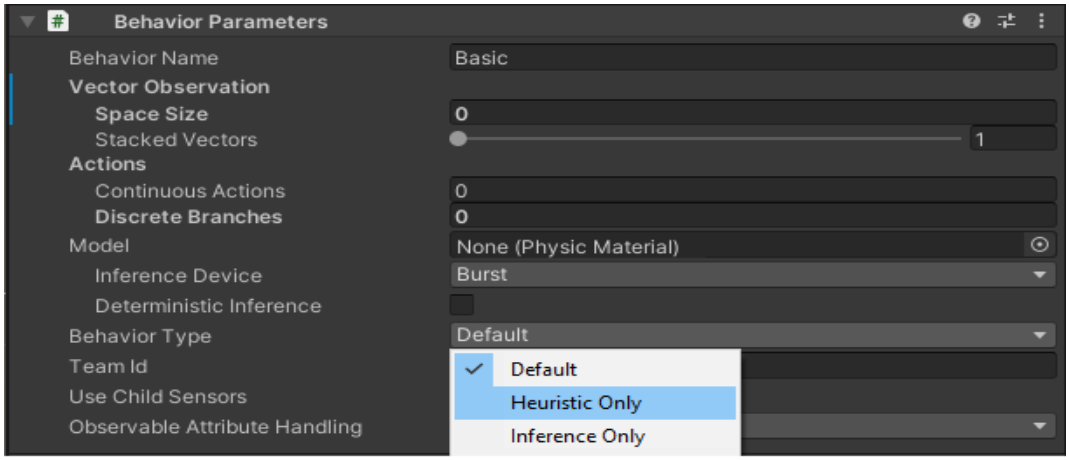
5.4 Imitationsinlärning i Unity

För att använda imitationsinlärning i Unity kan man använda samma testmiljö som byggts till förstärkningsinlärning. I testmiljön behöver man lägga till en komponent i ”Inspector” som kallas för ”Demonstration Recorder”. I komponenten finns det fyra parametrar som kan ändras på. ”*Record*” fungerar som en av/på knapp för att börja spela in demonstrationer. ”*Num Steps To Record*” begränsar hur många steg man vill spela in, men om man lämnar den på 0 spelar den in tills man trycker på stop. *”Name”* och *”Directory”* blir inlärnings filens namn och där filen kommer sparas. Detta illustreras i figur 26. Till följande ändrar man på ”*Behavior Type*” till ” *Heuristic*” för att ta kontroll över agenten och

klickar på spela-knappen för att börja spela in demonstrationer. Detta illustreras i figur 27.



Figur 26. Demonstrations komponent.



Figur 27. Heuristisk beteende.

Följande steg skapar man en ny mapp i vårt projekt och kallar den för *”config”*. Inuti mappen ska vi skapa ett nytt textdokument som vi kallar för ”*MoveToGoal.yaml*”. I text-dokumentet ska vi sätta dessa parametrar som illustreras i figur 28. Efter detta kan vi skriva i kommandotolken *”mlagents-learn config/MoveToGoal.yaml –run-id=Immitat-ion”* och klicka på spela-knappen för att börja träna agenten. Alternativt, om man vill förbättra på en tidigare modell kan man skriva istället ” *mlagents-learn cofig/Move-ToGoal.yaml –initialize-from=MoveToGoal --run-id=MovetogoalVer3*”. En förklaring på dessa parametrar kan läsas nedan.

```
behaviors:
  MoveToGoal:
    trainer_type: ppo
    hyperparameters:
      batch_size: 10
      buffer_size: 100
      learning_rate: 3.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
      gail:
        strength: 0.5
        demo_path: Demos/MoveToGoalPlayer.demo
    behavioral_cloning:
        strength: 0.5
        demo_path: Demos/MoveToGoalPlayer.demo
    max_steps: 500000
    time_horizon: 64
    summary_freq: 10000
```

Figur 28. Konfigurationsparametrar.

- *”Trainer_type: ppo”* En inlärningsalgoritm för förstärkningsinlärning.
    - *”Batch_size”* Antalet handlingar i varje repetition.
    - *”Buffer_size”* Antalet handlingar som samlas in före modellen uppdateras.
    - *”learning_rate”* Bestämmer hur mycket algoritmen ändras när den söker efter den optimala policyn.
    - *Beta”* ser till att agenten utforskar handlingsutrymmet ordentligt under träningen. Ökar man på detta värde får man mer slumpmässiga beslut.
    - *”Epsilon”* Påverkar hur snabbt modellen kan utvecklas under utbildning.
    - *”lambd”* Regulariseringsparameter som används vid beräkning av *” Generalized Advantage Estimate”*
    - *”num_epoch”* Antalet genomgångar för att komma igenom *” experience buffer”*. Minskar man på denna parameter får man stabilare uppdateringar, men långsammare träning.
    - *”learning_rate_schedule”* Bestämmer hur inlärningshastigheten förändras över tiden.
    - *”beta_schedule”* Bestämmer hur *”beta”* ändras med tiden.
- *”Network_settings”* Antalet lager i det neurala nätverket.
- ”*Reward signals*” Bestämmer hur man belönar agenten.
    - *”Extrinsic” B*elöningen som ges av miljön.
    - *”Gail”* belönar agenten för att bete sig liknade enligt demonstrationerna.
    - *”Behavioral cloning”* Tränar agenten att imitera exakta demostations handlingar.
    - *”gamma”* och *”strength”* påverkar agentens beteende.
- *” keep_checkpoints”* Hur många modeller som sparas efter en viss mängd steg har utförts.
- *”Max_steps”* Totala mängden steg som fås ta i miljön.
- *”Time_horizon”* Hur många steg erfarenhet som skall samlas in per agent.
- *”Summary_freq”* Mängden data som behövs innan statistik kan genereras och visas.

6 ANALYS

6.1 Jämförelse

Jämförelse mellan förstärkningsinlärning och imitationsinlärning i Unity var inte svårt eftersom Unity testerna gav tillräckligt med data. Förstärkningsinlärning klarar sig utan stora mängder data för att börja träna agenten medan imitationsinlärning var beroende av data för att kunna tränas. Imitationsinlärning var snabb i början på att träna in sig, men fanns det brister i datamängderna så ledde det till att agenten gjorde fel. Träningen med förstärkningsinlärningen var betydligt långsammare för att agenten utforskar omgivningen tills den hittar positiva belöningar. När agenten har hittat positiva belöningar börjar den anpassa sig samt förbättra på träningsmodellen. Det som imitationsinlärning inte klarar av att utföra kan förstärkningsinlärning lätt förbättra på sin modell. En begränsning för förstärkningsinlärning blir belöningssystemet. Om belöningssystemet ger svaga belöningar eller inga belöningar alls blir det svårt att träna agenten. För att få en effektiv förstärkningsinlärning modell måste man ge agenten regler som agenten lär sig ifrån.

6.2 Resultat

Tensorboard har olika diagram som kan användas för att förstå hur träningsfasen har gått. Statistiken som kommer användas representerar informationen för att få förståelse på en grundnivå. I graferna kommer färgen blå representera förstärkningsinlärning, grå imitationsinlärning och lila kombination av båda två.

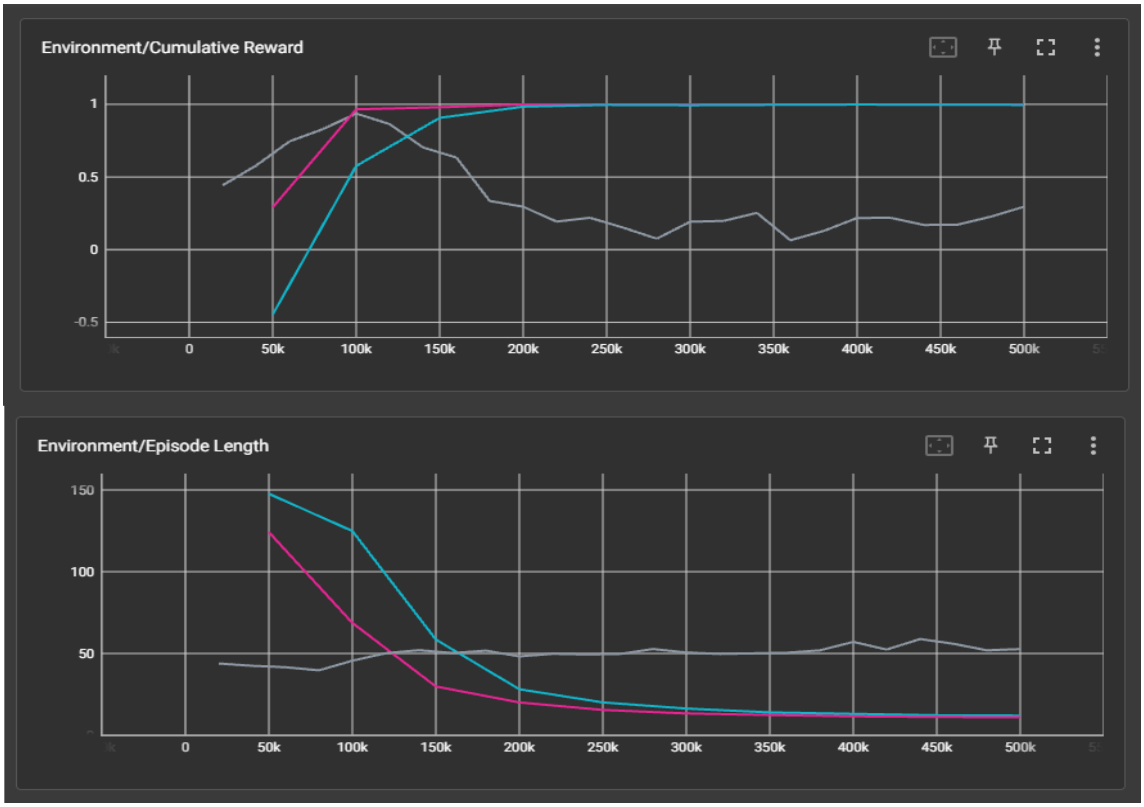
- *” Environment/Cumulative Reward”* - Den genomsnittliga kumulativa episodbelöningen för alla agenter.
- *” Environment/Episode Length”*- längden av varje avsnitt i miljön för alla agenter
- *” Losses/GAIL Loss”* - Hur bra modellen imiterar demonstration data
- *” Losses/Pretraining Loss”* - Hur bra modellen imiterar demonstration data

Tensorboard statistiken för förstärkningsinlärning kan ses i figur 29, och man kan se att kumulativa belöningarna ökar medan avsnitts längden blir kortare. Början av träningen tog tidsmässigt längre för agenten, men vid 50k börjar agenten hitta positiva belöningar och förbättra på inlärningsmodellen. Ungefär vid 200k klarar agenten av att hitta

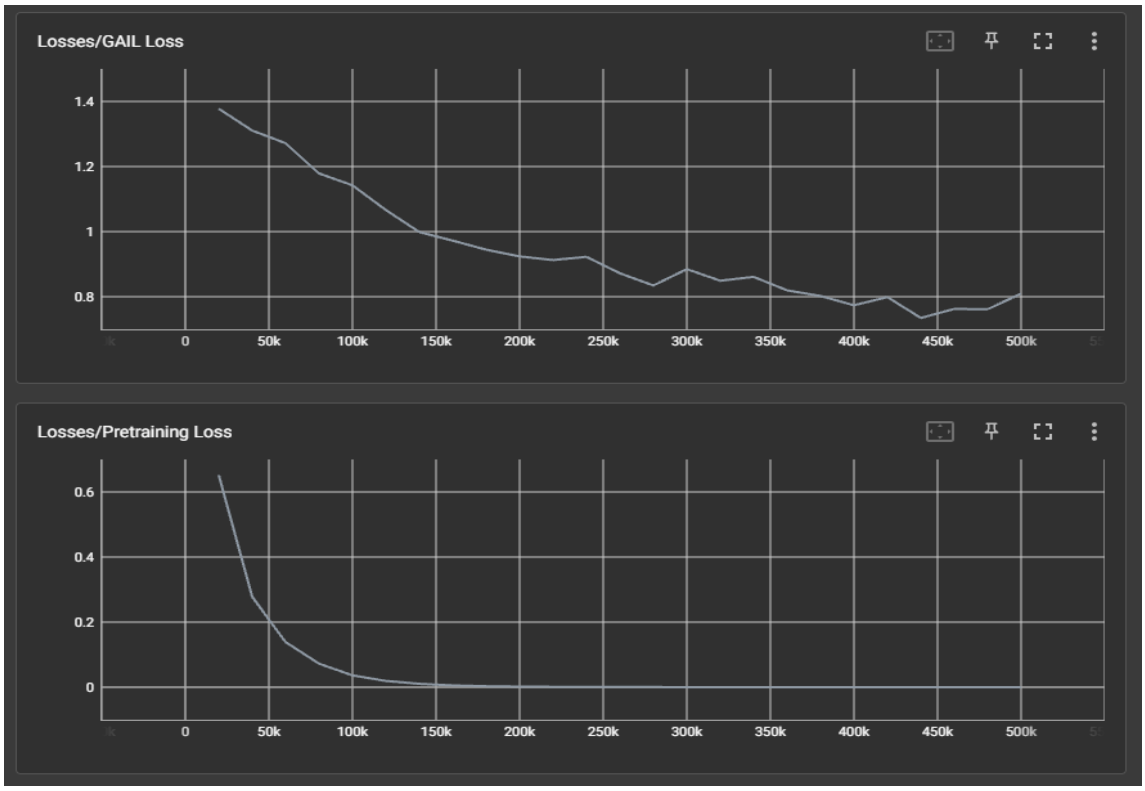
belöningarna utan problem medan aavsnittslängden blir kortare. Detta betyder att träningen varit lyckad för agenten.

Ur statistiken för imitationsinlärning ser man att det finns brister i demonstrationsdatan. Vid 100k börjar agenten missa belöningar för det inte finns tillräckligt med data för att hitta alla belöningar medan tidsmässigt har den hållit samma avsnittslängd. Detta betyder inte att träningen varit dålig eftersom om man tittar på figur 30, kan man se att agenten har utfört imitation av demonstrationer rätt. I figur 30 ser man när agenten börjar imitera demonstrations datan och följer demonstrationer utan större problem.

Statistiken för en blandning mellan förstärkningsinlärning och imitationsinlärning ser man att agenten lär sig snabbare att hitta belöningar. Agenten använde tid i början att utforska okända testmiljön, men lyckades utnyttja imitationsinlärning datan för att träna sig snabbare än förstärkningsinlärning.



Figur 29. Tensorboard statistik (blå-förstärkningsinlärning, grå-imitationsinlärning och lila kombination av båda två)



Figur 30. Tensorboard imitations statistik.

7 SLUTSATS

I detta examenarbete har förstärkningsinlärning och imitationsinlärning förklarats teoretiskt samt implementerats i praktiken i en testmiljö. Själva teoretiska delen går systematisk igenom båda metoderna samt förklarar maskininlärningsbaserade verktyg som har använts i detta arbete. Praktiska delen går först igenom installations fasen av arbetet var det förekom små installations problem som fixades genom att börja från början. 3D-testmiljön fungerade utmärkt för testerna var små justeringar på 3D-testmiljön måste göras i början av testfasen. Resultaten för förstärkningsinlärning var inte svåra att få medan imitationsinlärning var lite mer komplicerat. Imitationsinlärning visade sig hålla problem så som för lite demonstationsdata och felaktiga parametrar. Som ett sista test utfördes det en blandning av förstärkningsinlärning och imitationsinlärning. Själva testerna var delvis problematiska att hitta rätta parametrar för att agenten skall börja med imitationsinlärning och inte förstärkningsinlärning. Själva resultaten är jag nöjd med, men man kunde säker optimera parametarana och få ett bättre resultat annars har helheten av praktiska delen har varit lyckad. Jag tycker att examenarbetet har lyckats gå igenom teoretiskt vad

inlärningsmetoder går ut på och ger en bild av helheten samt introducera olika alternativ av AI träning såsom förstärkningsinlärning, imitationsinlärning och en blandning av bägge två. Dock tycker jag att examenarbetet misslyckades att förklara till läsaren vilken inlärningsmetod som lämpar dem bäst för att det behövs ytterliga testmiljöer för att få mera data.

Eftersom arbete endast går igenom en testmiljö finns det möjlighet för vidareutveckling. Man kunde utveckla olika varianter av uppgifter för agenten som till exempel en körbana eller en 3D-plattform och träna agenten med mera komplicerade uppgifter för att få mera data på agentens prestanda. Ett annat exempel man kunde göra är att testa förstärknings-inlärningsmetoden kombinerat med ”*behaviour trees*”, detta skulle dock kräva vidare forskning och utveckling.

8 KÄLLOR

BBC. 2019, Go master quits because AI 'cannot be defeated.
Tillgänglig: https://www.bbc.com/news/technology-50573071
Hämtad 26.9.2022

Bharat K. 2021, AI in chess: The Evolution of Artificial Intelligence In Chess Engines.
Tillgänglig: https://towardsdatascience.com/ai-in-chess-the-evolution-of-artificial-intelligence-in-chess-engines-a3a9e230ed50
Hämtad 28.3.2022

Divyansh Dwivedi. 2018, Machine Learning For Beginners
Tillgänglig: https://towardsdatascience.com/machine-learning-for-beginners-d247a9420dab
Hämtad 9.6.2022

Jacky Feng, 2020, Machine Learning: Introduction
Tillgänglig: https://medium.com/analytics-vidhya/machine-learning-introduction-50f55ef693e8
Hämtad 9.6.2022

Gio Wiederhold, John McCarthy, Ed Feigenbaum. u.å, Professort Arthur Samuel.
Tillgänglig: https://cs.stanford.edu/memoriam/professor-arthur-samuel
Hämtad 17.6.2022

Github. 2018, Unity-Technologies ml-agents.
Tillgänglig: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/images/rl_cycle.png
Hämtad 22.6.2022

Github. 2019, Installing ML-Agents Toolkit for Windows.
Tillgänglig: https://github.com/Unity-Technologies/ml-agents/blob/0.10.1/docs/Installation-Windows.md
Hämtad 20.9.2022

Github. 2021, Background-Machine-Learning.
Finns på: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Background-Machine-Learning.md
Hämtad 22.6.2022

Github. 2021, Getting Started Guide.
Tillgänglig: https://github.com/Unity-Technologies/ml-agents/blob/release_19_docs/docs/Getting-Started.md
Hämtad 23.9.2022

Github. 2021, Making a New Learning Environment.
Tillgänglig: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Learning-Environment-Create-New.md
Hämtad 23.9.2022

Github. 2021, Training Configuration File.
Tillgänglig: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md#sac-specific-configurations
Hämtad 6.10.2022

Github. 2021, Using TensorBoard to Observe Training.
Tillgänglig: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Using-Tensorboard.md
Hämtad 23.9.2022

Github. 2022, Installing ML-Agents Toolkit for Windows (Deprecated).
Tillgänglig: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Installation-Anaconda-Windows.md
Hämtad 23.9.2022

Github. 2022, Unity-Technologies / ml-agents / Installation.
Tillgänglig: https://github.com/Unity-Technologies/ml-agents/blob/release_19_docs/docs/Installation.md
Hämtad 14.9.2022

Laura E. Shummon Maass & Andy Luc. 2019, Artificial Intelligence in Video Games.
Tillgänglig: https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22
Hämtad 28.3.2022

Martin Stezano. 2017, In 1950, Alan Turing Created a Chess Computer Program That Prefigured A.I.
Tillgänglig: https://www.history.com/news/in-1950-alan-turing-created-a-chess-computer-program-that-prefigured-a-i
Hämtad 27.10.2022

Serdar Yegulalp. 2022, What is TensorFlow? The machine learning library explained.
Finns på: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html
Hämtad 13.7.2022

Shailja Gupta. 2019, The Inception of Machine learning.
Tillgänglig: https://towardsdatascience.com/the-inception-of-machine-learning-90b9fc3737ff
Hämtad 17.6.2022

SmartLabai. 2019, A brief overview of Imitation Learning.
Tillgänglig: https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c
Hämtad 23.6.2022

SpringerLink. 2019, Survey of imitation learning for robotic manipulation.
Tillgänglig: https://link.springer.com/article/10.1007/s41315-019-00103-5
Hämtad 29.9.2022

Thomas Simonini. 2020, An Introduction to Unity ML-Agents.
Tillgänglig: https://towardsdatascience.com/an-introduction-to-unity-ml-agents-6238452fcf4c
Hämtad 13.7.2022

Tommy Thompson. 2014, Why AI Researchers Love Playing Pac-Man
Tillgänglig: https://medium.com/@t2thompson/ailovespacman-9ffdd21b01ff
Hämtad 31.10.2022

Tom Simonite. 2018, Can Bots Outwit Humans in One of the Biggest Esports Games?
Tillgänglig: https://www.wired.com/story/can-bots-outwit-humans-in-one-of-the-biggest-esports-games/
Hämtad 22.6.2022

Unity Technologies. 2022, Every creative journey starts with Unity Hub
Tillgänglig: https://unity.com/unity-hub
Hämtad 28.10.2022

Unity Technologies, 2022, Unity Machine Learning Agents.
Tillgänglig: https://unity.com/products/machine-learning-agents
Hämtad 28.3.2022

Viraj Vaitha. 2022, Unity ML-Agents 2022: Installation.
Tillgänglig: https://virajvaitha.medium.com/unity-ml-agents-2022-installation-e8af1eab2dd
Hämtad 20.9.2022