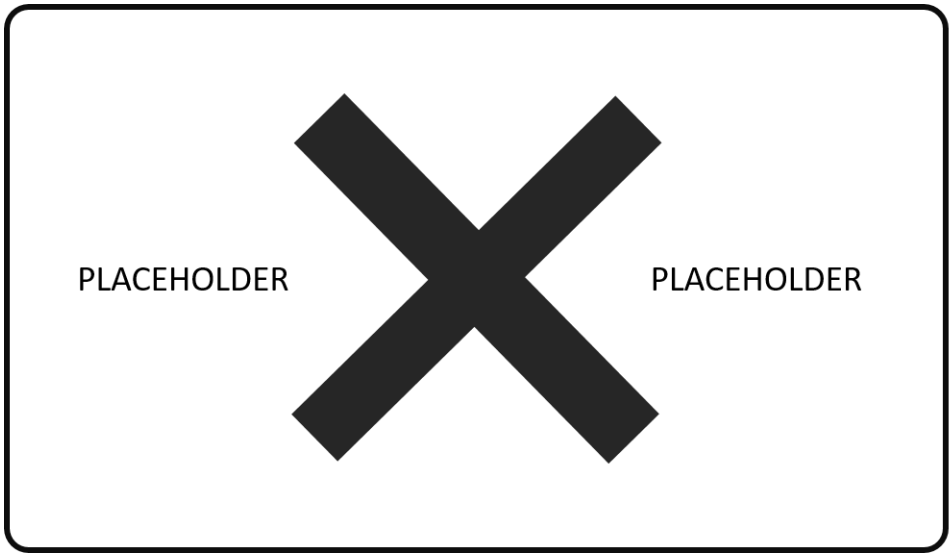

Heilbronn University

Master Project

Thesis Title

Denise Baumann, Martin Haag

supervised by

Prof. Dr. -Ing. Nicolaj Stache
&
Pascal Graph

April 26, 2022

Master Thesis

Thesis Title

at

Author:	Denise Baumann, Martin Haag
Matriculation Number:	123456, 194980
Course of Studies:	ASE, MMR
Supervisor:	Prof. Dr. -Ing. Nicolaj Stache & Pascal Graph
Time Frame:	Summer Semester / Winter Semester

Contents

1 | Einleitung

In Einleitungskapitel werden die Rahmenbedingungen des Projekts erläutert. In diesem Zuge sollen die Ausgangssituation und das Ziel genauer behandelt werden.

1.1 Ausgangssituation

Airhockey ist ein hochdynamisches Geschicklichkeitsspiel, bei dem zwei Personen gegen einander spielen. Hierfür sind die Spieler mit einem Pusher ausgestattet. Ziel des Spiels ist es, den auf einem Luftfilm gleitenden Puck im gegnerischen Tor zu versenken. Durch die hohen Geschwindigkeiten, welche sowohl den Puck, als auch die Pusher erreichen, besteht ein immenser Anspruch an die Mechanik, sowie an die informationsverarbeitenden Systeme des gesamten Roboters. Am Zentrum für maschinelles Lernen (ZML) wurde bereits in einer vorangegangenen Arbeit an einem selbst spielenden Airhockeytisch gearbeitet. Dazu wurde bereits ein Airhockeytisch mit der benötigten Hardware ausgestattet. Die Kinematik basiert dabei auf der des Roboters von jjRobots [**?**] Das Zentrum für maschinelles Lernen (ZML) möchte einen selbstlernenden und spielenden AirHockeyRoboter entwickeln. Dafür statteten Maschinenbaustudenten einen Airhockeytisch mit einem Kamerahalter und einem Roboter aus, welcher auf dem Konzept der Firma jjRobots basiert.

2 | Simulationsumgebung

Ohne eine angemessene Simulationsumgebung ist das ganze Projekt undenkbar. Nicht nur, dass das Training am realen Demonstrator schon wegen des Zeitaufwandes praktisch nicht möglich ist, auch die Konsistenz der Umgebung wage ich in Frage zu stellen. Die Integration der vielen Rewards sind mit der Bildverarbeitung auch komplizierter und in einer Simulation zusätzlich präziser. Da wegen vielen Gründen eine Simulation nötig ist und diese auch einen großen Teil der Arbeit ausgemacht hat, wird im folgenden Kapitel das Programm Unity vorgestellt. Außerdem wird unser Unity Projekt hinsichtlich der Implementierung und der Nutzung vorgestellt.

2.1 Auswahl der Umgebung

Da bereits aus einer vorhergegangenen Arbeit und ein Projekt in Unity vorhanden war ist die Entscheidung hier sehr schnell gefallen. Selbst ohne diesen Aspekt ist Unity aber eine gute Wahl. Neben einer Pythonschnittstellte, die für Reinforcemet Learning immer nützlich werden kann kann Unity auch noch mit einer großen Community und sehr guter Dokmentation punkten. Dadurch ist die Einarbeitungsphase relativ kurz und angenehm. Hinzu kommt noch die Toolbox ML-Agents. Diese stellt bereits Agenten zur verfügung, bei denen nur noch Hyperparameter gewählt werden müssen. Eine schnelle, unkomplizierte Möglichkeit mit dem Training zu beginnen.

2.2 Unity

Unity ist eine von Unity Technologies entwickelte multiplattform Entwicklungsumgebung zum Erstellen von Videospielen. Unsere Anwendungist zwar kein Videospiel im herkömmlichne Sinn, aber von der Physikengine können wir trotzdem gebrauch machen. Neben dreidimensionalen Umgebungen bietet Unity auch einen 2D-Modus an, der für unsere Anwendung ausreichend ist. Die Programmiersprache unserer Wahl ist c#, jedoch ist es auch möglich in UnityScript und Boo benutzerdefiniertes Verhalten zu programmieren.
Wir haben die Unity Version 2020.1.6f1 genutzt.
Die Version des ML-Agents Toolkits, die zum Einsatz kam, lautet 2.1.0-exp.1

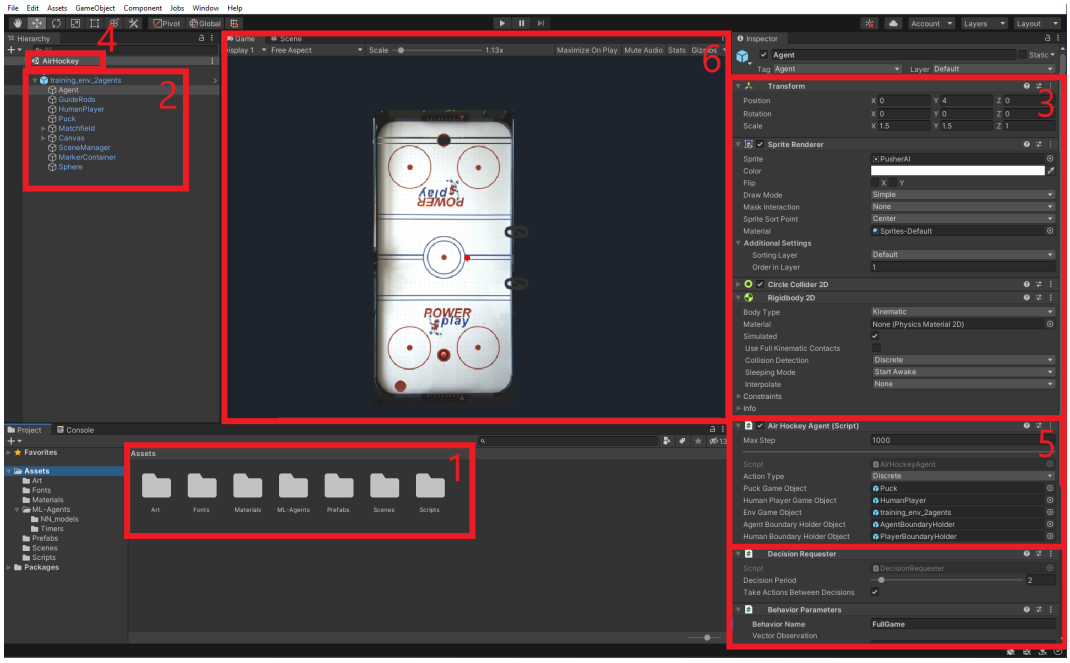

Figure 2.1: Benutzeroberfläche von Unity mit Markierungen

2.2.1 Kurze Einführung

Die kurze Einführung in Unity soll anhand des User Interfaces geschehen. Diese ist keineswegs vollständig und soll auch nur ein grobes Verständnis für Fachfremde ermöglichen. Die Markierungen im Bild **??** werden im Folgenden erklärt.

- Assets (1)
  Assets beinhaltet diverses. Neben Grafiken, vorgefertigten Materialien und Szenen sind hier auch die Skripte zu finden

- GameObjects (2)
  GameObjects sind das zenrale Konzept von Unity. Jedes Objekt, jede Kamera und jede Grafik ist durch ein Gameobject definiert. Die Funktionalitäten eines GameObjects werden durch die Components hinzugefügt. Die GameObjects sind in einem Hierariebaum in der Scene eingeordnet.

- Components (3)
  Components geben den GameObjects ihre Funktionalität. Ein GameObject kann mehrere Components haben. Beispiele für Components sind Transform (zuständig für die Position), Collider(zuständig für die Interaktion in der Physiksimulation) und auch Skript.

- Scene (4)

Eine Scene (Szene) besteht aus GameObjects. Es ist im Prinzip die Wurzel des Hierarachiebaums. Zur Spieleentwicklung könnten hier unterschiedliche Level als underschiedliche Scenes interpretiert werden. In unserem Fall sind nur zwei Scenen vorhanden: eine mit einem Spielfeld zum selbst spielen und testen und eine mit acht Feldern zum beschleunigten Training.

- Script (5) :
  Scripts sind auch Components. Sie sind die Möglichkeit selbst Verhalten zu definieren. Es kann sich in Scripts auf GameObjects bezogen werden. Mit ihrer Hilfe können Parameter wie Material oder Position geändert werden. Objekte können auch entfernt oder eingefügt werden.

- Visualisation (6) :
  In diesem Bereich kann sowohl das Bild einer Kamera, und damit die Ansicht im Spiel, beobachtet werden als auch eine Darstellung aller GameObjects in der Scene. Objekte können hier auch verschoben oder gedreht werden.

2.2.2 Airhockey Projekt

In diesem Unterkapitel werden die einzelnen GameObjects und die wichtigstens Components im Projekt vorgestellt und erklärt. Ziel ist es dabei nicht auf jedes Detail einzugehen, sondern die Arbeit für Folgeprojekte zu erleichtern. Rein optische Aspekte, wie zum Beispiel die Anzeige des Spielstandes, werden nicht betrachtet. Außerdem wird nur auf die Scene mit einem Feld eingegangen, dann die Anpassungen, die zum Kopieren gemacht werden müssen sind nicht maßgeblich. Die folgenden GameObjects sind auch in der Abbildung 2.2.1 zu finden.

Agent :

Das Agent ist eines der zentralen GameObjects. Es hat eine Sprite Renderer Component. Dadurch kann es mit einer .png Datei Visuallisiert werden. Der Agent ist also ein Objekt, dass im Spiel zu sehen ist. Auf der rechten Seite ist zu sehen, wie es im Projekt zu sehen ist.

Figure 2.2: Ansicht des Agenten in Unity

Das Agent GameObject enthält auch eine Circle Colider Component. Mit der Rigidbody Component zusammen wird die Phsyik(Reibung, Bewegung,

Kollision) simuliert.
Ein weiterer Component des Agent Objekts ist das Script Airhockey Agent. Die Funktion von diesem Script ist die Interaktion mit der ML Agents Toolbox. Es werden sowohl in die anfalleden Rewards entsprechend des Spielverlaufes an das Netzwerk zurück gegeben, als auch die Actions entgegen genommen und damit die Umgebung (Bewegung des Agenten) beeinflusst. Zu Episodenbeginn werden in diesem Script auch die Positionen von Agent und Spieler (Pusher) zurück gesetzt.
Des weiteren beinhaltet das Agent GameObject auch noch eine Decission Requester Componente. Mit ihrer Hilfe wird regelmäßig, entsprechend des Parameters Decision Periode, eine Action vom Netzwerk angefordert. Die Behavior Parameter Componente ist, neben Decison Requester, zur Parametrisierung der Nutzung des ML Agents Toolkits nötig. Hier kann die Dimension sowohl des Actionspaces als auch die der Obseration festgelegt werden. Auch Angaben zur Interferenz können hier gemacht werden.

HumanPlayer :

HumanPlayer ist das GameObject des zweiten Spielers. Es hat auch eine Sprite Renderer Component. Die Visualisierung ist rechts zu sehen. Da diese Objekt auch am Spiel teilnimmt hat es auch die Components Circle Collider und Rigidbody.

Figure 2.3: Ansicht des Pushers in Unity

HumanPlayer hat auch die Components Decission Requester und Behavior Parameters. Damit wird das Selfplay ermöglicht. In unserer Implemetierung wird die Verion des Agenten, die den HumanPlayer bewegt nicht trainiert. Das ML Agents Toolkit wird hier nur zur Interferenz genutzt.
Das Objekt enthält auch ein Script. Im Gegensatz zum Airhockey Agents Script werden hier aber eine Rewards zurück gegeben sondern nur die Observations und die Actions behandelt. Das reicht auch aus, denn dieser Agent soll ja garnicht ständig mittrainieren, sondern nur das Selfplay ermöglichen. Wenn er zu schwach wird, wird einfach die stärkere Version vom Agent übertragen. Im Script ist auch die Option den HumanPlayer selbst zu steuern implementiert. Entweder mit der Tastatur oder mit einem Controller kann so der Agent herrausgefordert werden.

Puck :

Abhänig vom Spielszenario wird in diesem Script die Startposition und die Startgeschwindigkeit des Pucks festgelgt. Auch der Spielstand wird hier

Der Puck ist auch ein Objekt, dass wie der HumanPlayer und der Agent am Spiel teilnehmen. Deshalb hat es auch die Components Circle Collide und Rigidbody. Die Components, die für das ML Agents Toolkit nötig sind, sind hier aber nicht nötig. Trotzdem gibt es auch hier ein c# Script, dass das Verhalten des Pucks bestimmt.

Figure 2.4: Ansicht des Pucks in Unity

mitgezählt. Die Circle Collider Componente des Pucks erlaubt es ein Event bei Kollision mit anderen Objekten auszulösen. Wenn der Spielfeldrand am Tor des Agenten getroffe wird, kann das mit Hilfe eines GameObjects ohne Renderer erkannt werden. Es kann mit einer Box Collider Componente ein Event ausgelöst werden, das den Spielstand anpasst. Das gleiche System wird auch auf der anderen Seite angewendet.

Matchfield :

Das GameObject Matchfield hat selbst nur eine Sprite Renderer Component, die ein Bild des Airhockeytisches zeigt. Jedoch sind diesem Objekt hierarchisch weitere untergeordnet. Diese untergeordneten Objekte sind aber alle ohne Renderer Component und deshalb in der Spielansicht unsichtbar. Sie sind nur wegen ihrer Position interessant. Sie dienen um das Spielfeld in Zonen einzuteilen. Die zulässigen Positionen von Agent, HumanPlayer und Puck werden damit auf das Spielfeld oder die jeweilige Hälfte limitiert. Die GameObjekts, die zum Tore zählen genutzt werden sind auch Childobjekte (hierarchisch untergeordnete Objekte) des Matchfieldes.

Figure 2.5: Spielfeldbegrenzungen in Unity

Das Matchfield hat kein Script, es wird sich nur von anderen Scripts auf die GameObjects von Matchfield bezogen. Die Transform Component, die die Position eines Objects angibt, wird dabei ausgelesen. Mit vier Objekten lässt sich damit ein rechteckiger Bereich festlegen.

training_env_2agents :
Diesem Objekt sind, abgesehen von der Kamera, alle anderen GameObjects untergeordnet. Es dient als Container für das ganze Spiel. Das Objekt dient hauptsächlich zwei Zecken:

- Das ganze Spielfeld kann einfacher kopiert und verschoben werden. Die Positionen der Childobjekte bleiben beim Verschieben des übergeordneten Objekts relativ zu diesem unverändert. Felder könne dadurch schnell über und nebeneinander kopiert werden.
- Das envScript ist in diesem GameObject ein Component. In diesem Script werden alle Rewards festgelegt. Auch der Spielmodus wird hier ausgewählt. Näheres zu den Rewards und den Spielmodi wir im Kapitel 2.2.2 erläutert.

3 | Rewards und Training

In diesem Kapitel wird auf das Training eingegangen. Hierbei werden die unterschiedlichen Rewards erklärt, die genutzten Netzwerke, die Hyperparameter und die einzelnen Schritte im Training. Dazu wird auch das envScript aus Unterkapitel 2.2.2 nochmal genauer beleuchtet.

3.1 Rewards und Umgebungsparameter

Hier sollen als erstes alle implementierten Rewards vorgestellt werden, damit in den folgenden Teilen der Beschreibung des Trianings klar ist, welche Effekte sie bedingen.

- taskType
  Legt fest, welcher Spielmodus gewählt wird. Siehe dazu Abschnitt 3.2.
- V_max_puck
  Legt die Maximalgeschwindigkeit des Pucks fest. Einheit ist dabei nicht m/s, der Tisch ist in der Realität kleiner als in Unity. Die Ermittlung der Maximalgeschwindigkeit kann im Abschnitt 4.1 nachvollzogen werden.
- V_max_robo
  Legt die Maximalgeschwindigkeit des Roboters fest. Messungen erfolgten analog zu denen der Maximalgeschwindigkeit des Pucks.
- V_max_human
  Legt die Maximalgeschwindigkeit des HumanPlayer fest. Messungen erfolgten analog zu denen der Maximalgeschwindigkeit des Pucks.
- neghumanGoalReward
  Dieser Reward wird gegeben, wenn der Puck in das Tor des Agenten trifft. Da es ein Gegentor ist sollte der Reward negativ gewählt werden.
- agentGoalReward
  Dieser Reward wird gegeben, wenn der Puck in das Tor des HumanPlayer trifft.

- avoidBoundaries
  Dieser Reward wird gegeben, wenn der Roboter die Bande berührt. Er sollte negativ sein.

- avoidDirectonChanges
  Dieser Reward wird gegeben, wenn der Roboter die Bewegungsrichtung ändert. Er sollte negativ sein. Der Betrag wird mit dem Betrag der Bewegungsänderung im letzten Zeitschritt multpliziert. Da dieser Reward in jedem Zeitschritt gegeben werden kann sollte dieser Reward betragsmäßig sehr klein gewählt werden.

- stayCenteredReward
  Dieser Reward belohnt es weder links noch rechts am Spielfeldrand zu sein. Damit kann das Verteidigungsverhalten verbessert werden. Der Reward wird zu jedem Zeitschritt gegeben und sollte deshalb sehr niedig gewählt werden. Der Reward sinkt proportional mit der Distanz vom Rand.

- negoffCentereReward
  Dieser Reward ist vergleichbar mit dem stayCenteredReward. Jedoch ist dieser negativ zu wählen, denn er fällt betragsmäßig am hochsten aus, wenn der Agent an der Bande steht(links oder rechts).

- encouragePuckMovement
  Dieser Reward belohnt Puckbewegungsgeschwindigkeit. Er wird zu jedem Zeitschritt gegeben und sollte deshalb klein sein.

- encouragePuckContact
  Dieser Reward belohnt Kontakte mit dem Puck.

- contacthalf
  Dieser Parameter ist kein Reward sondern eine Booleanvariable. Wird sie zu True gesetzt hat das zur Folge, das der Reward encouragePuckContact jedes mal wenn er gegeben wird halbiert wird. Damit wird verhindert, dass der Roboter alleine spielt oder den Puck einklemmt um den encouragePuckContact Reward auszunutzen.

- playForwardReward
  Dieser Reward wird gegeben, wenn der Puck von hinten getroffen wird, also in die richtige Richtung geschossen wird.

- negplaybackReward
  Dieser Reward wird gegeben, wenn der Puck von vorne getroffen wird, also in die falsche Richtung geschossen wird. Er sollte negativ gewählt werden.

- negStepReward
  Dieser Reward wird in jedem Fall zu jedem Zeitschritt gegeben. Er soll langweiliges Zeitspiel verhindern. Dazu muss er negativ sein.

- negMaxStepReward
  Dieser Reward wird gegeben, wenn die Maximallänge einer Episode erreicht wird und das Spiel deshalb abgebrochen wird(wird nur im Training abgebrochen, nicht im Spiel). Er sollte negativ sein.

- behindPuckReward
  Dieser Reward wird zu jedem Zeitschritt, zu dem sich der Agent näher am eigenen Tor befindet als der Puck gegeben. Er sollte niedrig gewählt werden, da er in jedem Zeitschritt geholt werden kann.

- behindPuckReward
  Dieser Reward ist exklusiv für das Training des Spielmoduses Defending. Er wird jedes mal gegeben, wenn ein Schuss erfolgreich verteidigt wurde.

3.2 Spielmodi

In diesem Abschnitt werden die implementierten Spielmodi beschrieben. Mit ihrer Hilfe kann die Eignung eines Netzwerk oft schneller festgestellt werden, als mit einem ganzen Spiel, da die anderen Spielsituationen nicht so komplex sind. Ein weiterer Vorteil von einfacheren Spielszenarien ist, dass damit der Agent Stufe für Stufe traininiert werden kann.

- Reaching
  In diesem Spielmodus ist es das Ziel den Puck zu erreichen. Dazu wird der Puck zu Beginn der Episode an einem zufälligen Ort auf der Spielfeldseite des Agenten platziert. Da HumanPlayer bei diesem Szenario keine Rolle spielt wird seine Bewegung gestopt. Die Episode endet mit der Kollision von Agent und Puck oder beim Erreichen der maximalen Simulationsschritten. Damit ist dieser Modus von sehr niedriger Komplexität und auch für einen untrainierten Agenten schnell erlernbar.

- Scoring
  Im Scoring Modus geht es nur ums Zielen und Tor treffen. Hier wird die Episode nicht beendet wenn der Puck getroffen wurde, sondern wenn der Puck entweder ein Tor erreicht oder die Bande trifft. Wie beim Reaching steht der HumanPlayer unbeweglich neben seinem Tor. Da mit einem diskreten Actionspace gearbeitet wird ist dieser Modus nicht viel verwendet worden. Wegen des diskreten Actionspace kann der Agent sich nicht präzise genug platzieren um zu zielen. Sollte in einem Folgeprojekt jedoch mit einem kontinuierlichen Actionspace gearbeitet werden kann dieser Modus wiede interessant werden.

- Defending
  Hier hat der Agent das Ziel sein Tor zu verteidigen. Auch in diesem Modus spielt der HumanPlayer keine Rolle. Er steht dabei unbeweglich neben seinem Tor. Der Agent wird zu Beginn der Episode an eine

zufällige Position auf seiner Spielfeldseite platziert. Defending ist der einzige Modus, bei dem der Puck nicht ohne Geschwindgkeit auf das Feld gesetzt wird. Der Puck wird hier von der Seite des HumanPlayer aus mit einem zufälligen Bewegungswinkel zwischen 70 und -70 auf die Seite des Agenten geschossen. Auch die Startposition wird variiert. Beendet wird die Episode wenn entweder ein Tor kassiert wurde oder der Ball abgewehrt wurde. Das ist der Fall wenn die X-Komponente der Geschwindigkeit des Pucks negativ wird, sich der Puck also auf die Hälfte des HumanPlayer zurück bewegt. In diesem Modus ist ein schneller Lernfortschritt nur mit dem defenceReward zu erwarten (bei fast allen Netzwerken).

- FullGame
  FullGame ist selbsterklärend. Der Puck wird an einer zufälligen Position auf dem ganzen Spielfeld ins Spiel gebracht. Die Kontrahenten Agent und HumanPlayer werden auf ihre jeweilige Seite gesetzt und beiden ist die Bewegung im Rahmen der Parameter in envScript erlaubt. Beendet wird die Episode nur wenn ein Tor erzielt wird oder die maximale Anzahl an Simulationsschritten erreicht wird. Diesen Modus zu meistern ist das Ziel der Trainings und eine Teilaufgabe unseres Masterprojekts. Er ist zu komplex um einen untrainierten Agenten ohne das Stufenweise ändern von Rewards zu trainieren.

4 | Realer Demonstrator

alles real

4.1 Geschwindigkeitsmessungen

vmessung

Bibliography