

HEILBRONN UNIVERSITY

MASTER PROJECT

Thesis Title

Denise Baumann, Martin Haag

supervised by

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

April 22, 2022

MASTER THESIS

Thesis Title

at



HEILBRONN UNIVERSITY
OF APPLIED SCIENCES

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	2
1.1	Ausgangssituation	2
2	Simulationsumgebung	3
2.1	Auswahl der Umgebung	3
2.2	Unity	3
2.2.1	Kurze Einführung	4
2.2.2	Airhockey Projekt	5

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 Pythonschnittstelle, die für Reinforcement Learning immer nützlich werden kann kann Unity auch noch mit einer großen Community und sehr guter Dokumentation 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 Anwendung ist zwar kein Videospiel im herkömmlichen 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

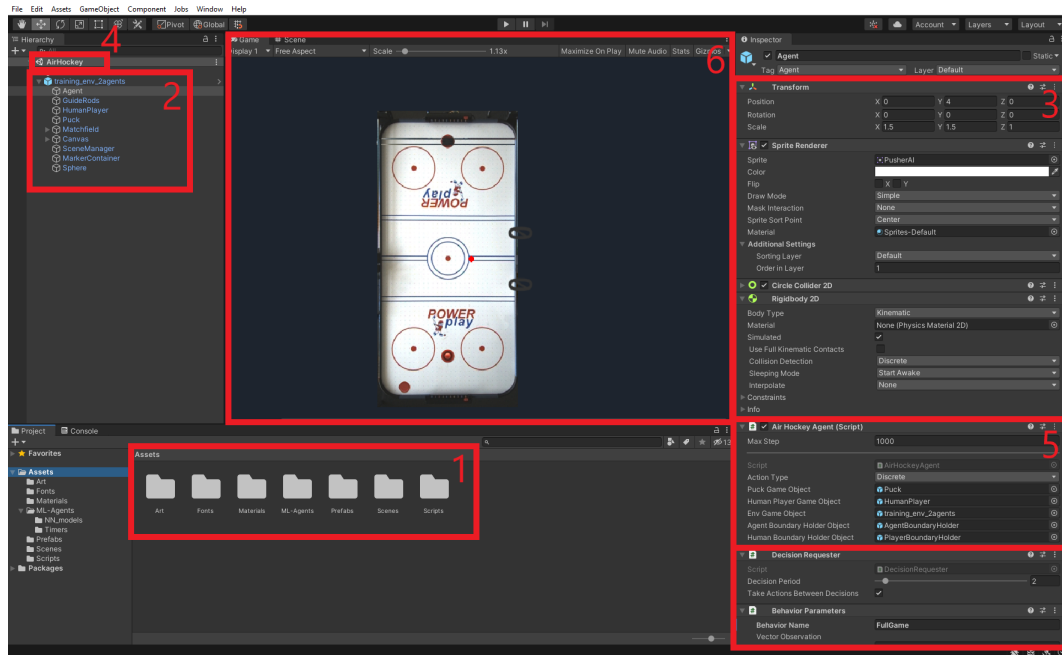


Figure 2.1: Benutzeroberfläche von Unity mit Markierungen

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 zentrale 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 Hierarchiebaum 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 Hierarchiebaums. Zur Spieleentwicklung könnten hier unterschiedliche Level als unterschiedliche Scenes interpretiert werden. In unserem Fall sind nur zwei Scenes 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 wichtigsten 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 Visualisiert 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 Physik(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 anfallenden 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 Decision Requester Componente. Mit ihrer Hilfe wird regelmäßig, entsprechend des Parameters Decision Periode, eine Action vom Netzwerk angefordert. Die Behavior Parameter Componente ist, neben Decision Requester, zur Parametrisierung der Nutzung des ML Agents Toolkits nötig. Hier kann die Dimension sowohl des Actionspace als auch die der Observation 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 Decision Requester und Behavior Parameters. Damit wird das Selfplay ermöglicht. In unserer Implementierung wird die Version 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 keine Rewards zurück gegeben sondern nur die Observations und die Actions behandelt. Das reicht auch aus, denn dieser Agent soll ja gar nicht 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 herausgefordert werden.

Puck :

Abhängig vom Spielszenario wird in diesem Script die Startposition und die Startgeschwindigkeit des Pucks festgelegt. 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 Komponente des Pucks erlaubt es ein Event bei Kollision mit anderen Objekten auszulösen. Wenn der Spielfeldrand am Tor des Agenten getroffen wird, kann das mit Hilfe eines GameObjects ohne Renderer erkannt werden. Es kann mit einer Box Collider Komponente 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 GameObjects, 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 Zwecken:

- 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.

Bibliography