Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Künstliche Intelligenz

Studienarbeit AR-VR

von

Stephan **Prettner**

4-Detect

Eine auf KI basierte Spielunterstützung mit AR

Bearbeitungszeitraum: von 24. November 2021
bis 2. Februar 2022

1. Prüfer: Prof. Dr.-Ing. Gerald Pirkl

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Projektidee

Es soll ein System entwickelt werden mit dem mittels Kamera und Bilderkennung der Status eines 4-Gewinnt Spiels ermittelt werden kann. Mithilfe einer künstlichen Intelligenz soll anhand des aktuellen Spielstatus ein Zugvorschlag ermittelt werden, welche dem Nutzer helfen ein Spiel mit höherer Wahrscheinlichkeit zu gewinnen. Der Grundgedanke ist es, Spielern mit weniger Erfahrung so weit zu unterstützen, dass sie mit besseren Spielern mithalten können. Ein Anwendungsfall wäre es, wenn jemand mit jüngeren Geschwistern spielen möchte. So könnte der jüngere Spieler mit dem System unterstützt und seine Gewinnchance gesteigert werden. Der erfahrene Spieler hätte somit einen stärkeren Gegner und der unerfahrene Spieler würde nicht nur verlieren, wodurch der Spielspaß beider Spieler gesteigert wird.

1.2 Persönliche Motivation

Aus der Aufgabenstellung der Studienarbeit ging für mich hervor, dass die Umsetzung relativ frei gestaltet ist. Das hat mich persönlich dazu motiviert, einige alternative Ansätze, die ich sonst eventuell nicht getestet hätte, zu implementieren. Ein wichtiger Punkt war mir hier zum Beispiel die Umsetzung der einzelnen Anforderungen mithilfe verschiedener KI-Ansätze.

Beispielsweise ist die Spielzugbewertung des Spiels 4-Gewinnt ein gelöstes Problem. Dennoch wollte ich hier ein eigenes Modell trainieren, das dem Benutzer einen Zug vorschlägt. Weiters sind alle Entscheidungen während der Umsetzung dieses Projekts mit dem Hintergrundgedanken getroffen worden, das System auch in einem Browser mit WebGL verwenden zu können. Dadurch hat sich das in Unity verfügbare Framework AR Foundation disqualifiziert, da es zwar viele Plattformen, aber nicht WebGL unterstützt.

Kapitel 2

Spielzugbewertung

2.1 Unity ML-Agents Toolkit

2.1.1 Allgemein

Dieses Framework dient zur Anwendung von Machine Learning Ansätzen innerhalb von Unity. Das Ziel ist es, komplexere Simulationen, die in Unity umgesetzt werden, direkt zu verwenden. Beispielsweise können so Systeme auf existierenden Programmen implementiert werden, die auf Gravitation, Kollisionen, Lichtquellen oder weitere physische Effekte zurückgreifen.

Innerhalb dieses Projektes habe ich mich dazu entschlossen, mithilfe von ML-Agents Agenten zu implementieren, die den nach ihrem Modell besten Spielzug prognostizieren. Dazu muss dieses Modell zuerst erstellt werden, wozu ich Reinforcement Learning anwende. Bei dieser Trainingsart spielen zwei Agenten mit einem jeweils an den Gewichten leicht modifizierten Modell gegeneinander. Das Modell, das gewinnt, erreicht damit eine höhere Bewertung. Dieser Vorgang wird immer wieder wiederholt, bis die Vorhersage eines Spielzuges ein zufriedenstellendes Ergebnis erreicht.

2.1.2 Vorraussetzungen

Das Toolkit benötigt zur Verwendung verschiedene Teilkomponenten, die erst eingerichtet werden müssen, bevor es eingesetzt werden kann. Wichtig ist dabei, dass bei der Installation von Python maximal die Version 3.9.* verwendet werden darf, da ML-Agents zu diesem Zeitpunkt noch keine höhere Version unterstützt.

- Python (Version <= 3.9.*)
- PyTorch (Optional: CUDA)
- PIP Pakete: torch, mlagents
- Unity Paketmanager: ML-Agents Version 2.0

2.2 Aufbau in Unity

Um ein Training des Modells durchführen zu können, musste zuerst überhaupt erst das Spielmodell selbst hinzugefügt werden. Die Chips und das Spielfeld, die mithilfe von Blender erstellt wurden, sind in Unity Szenen hinzugefügt worden. Die Chips werden dabei von der Gravitation beeinflusst und können miteinander kollidieren. Diese Spielobjekte wurden anschließend in einer Prefab zusammengefasst, um für die Trainingsszene mehre Objekte davon erstellen zu können. Durch die Verwendung mehrerer Spielfelder und Agenten kann eine größere Variation der Modelle während einer Trainingsphase gewährleistet werden sowie das Training selbst schneller durchgeführt werden, da ML-Agents das gleichzeitige Training mehrerer Agenten unterstützt.

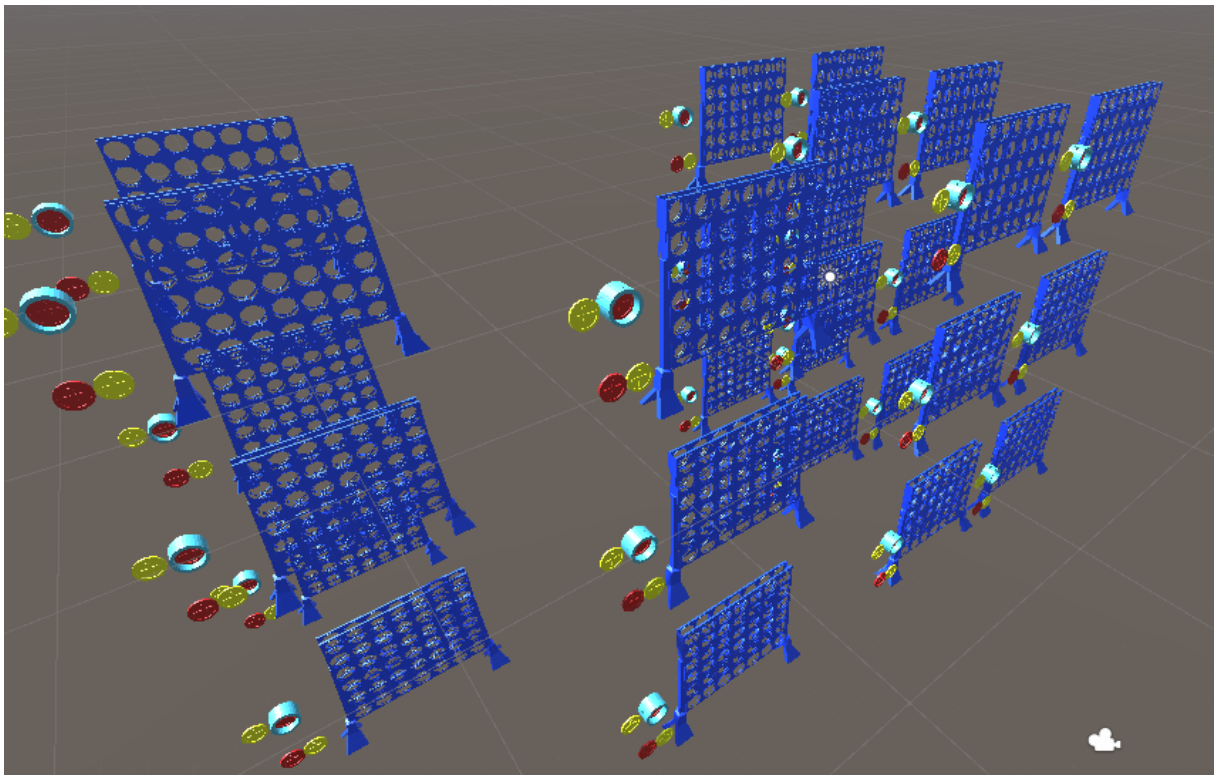

Abbildung 2.1: Trainingsszene mit mehreren Spielfeldern und Agenten

2.3 Evaluation des Spielzuges

Der wichtigste Teil des Trainings mithilfe von Reinforcement Learning ist die Belohnung des Agenten nach einem Spiel. Hierbei hatte ich anfangs versucht, die Agenten nur anhand des Endzustandes zu belohnen. Das bedeutet, dass es für ein abgeschlossenes Spiel Punkte für das Gewinnen, Verlieren oder Unentschieden in Abhängigkeit des Startspielers gibt (Der Startspieler hat bei 4-Gewinnt einen Vorteil). Das Training brachte so aber kaum Fortschritte bei der Vorhersage. Deshalb habe ich die Bewertung des Fortschritts erweitert, indem ich, ähnlich wie Baby und Goswami (2019), jeden einzelnen Spielzug, also das Einwerfen jeden Chips, bewerte. Es gibt im aktuellen Modell Belohnungen bzw. Bestrafungen für das Gewinnen, Verlieren, Unentschieden,

Verhindern von 2 bzw. 3 Chips in der Reihe sowie das Erstellen von 2 bzw. 3 Chips in einer Reihe. Meine ersten Versuche mit diesem neuen System führten zwar zu korrekten, aber nicht erwarteten Ergebnissen (Siehe: Abbildung 2.2 und 2.3). Hier wurde das Sammeln von Punkten priorisiert, das Verhindern eines Sieges war also nicht hoch genug belohnt.

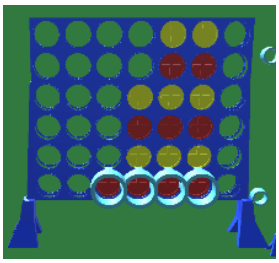

Abbildung 2.2: Das Sammeln von Punkten ist zu wichtig.

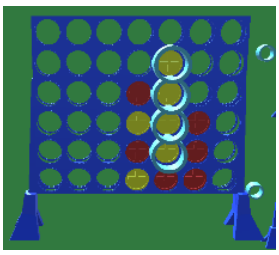

Abbildung 2.3: Das Verhindern eines Sieges ist nicht wichtig genug.

Nach den Anpassungen der Punkte für die einzelnen Spielzüge und einigen Evaluationen dieser hat sich das sehr verbessert und führt zu sehr brauchbaren Spielzügen mit dem Modell, das anschließend für das Projekt verwendet wurde. Es schlägt zwar nicht die perfekten Spielzüge vor, das war aber nie mein Ziel des Trainings.

Kapitel 3

Spielfelderkennung

Ziel dieses Abschnitts ist es, für die spätere Erkennung der Chips eine Groberkennung des Spielfeldes innerhalb des von der Kamera erfassten Bildes durchzuführen. Dadurch erwarte ich mir eine höhere Performance, da die rechenintensiveren OpenCV-Operationen zur exakten Erfassung des aktuellen Spielfeldes nur auf einem kleineren Teilbereich durchgeführt werden müssen. Dazu habe ich mir mehrere Möglichkeiten angesehen, wobei ich auf das Feature Matching sowie Haar Cascades genauer eingehen möchte.

3.1 Feature Matching

Die Idee war es, mithilfe eines Bildes des Spielbretts den Bereich des Kamerabildes einzugrenzen. Mithilfe von Deskriptoren konnten damit auch ziemlich gute Ergebnisse erzielt werden, vor allem, ob sich das Spielbrett überhaupt auf dem Bild befindet. Diese Ergebnisse hatten mich anfangs sehr positiv gestimmt, bis ich dann aber feststellen musste, dass es deutliche Probleme bei der Erkennung der Bounding Box bei leicht ändernden Hintergrundumgebungen, Lichtverhältnissen oder sogar beim Einwerfen einzelnen Chips gibt. Zwar ist die Erkennung sehr schnell, jedoch habe ich mich schlussendlich dazu entschlossen, alternative Ansätze, wie beispielsweise die Haar Cascades, auszuprobieren.

3.2 Haar Cascades

3.2.1 Allgemein

Haar Cascades verwenden Haar Features zur Erkennung von markanten Stellen innerhalb eines Bildes. Dabei werden kleinere Regionen nach Kanten, Linien etc. überprüft und bewertet. Bei den Beispielimplementationen werden dabei häufig die Gesichtserkennung genannt, wozu es schon viele vortrainierte Cascades gibt. Dabei gibt es beispielsweise existierende Haar Features zur Erkennung der Kanten eines Gesichts, Augen, Mund usw. Möchte man weitere Objekte erkennen, muss, wie von

Viola und Jones (2001) beschrieben, einer Haar Cascade erst beigebracht werden, auf welche Features sie dabei achten soll.

Haar-Cascades arbeiten damit ähnlich wie CNNs (Convolutional Neural Networks), die ebenso mithilfe von Kernels versuchen, markante Features zu finden.

3.2.2 Training

Für das Training von Haar Cascades müssen Trainingsdaten existieren, in denen die erwünschten Ergebnisse selektiert werden müssen. Dazu habe ich 300 Bilder mit einer Kamera aufgenommen. Auf 150 davon existiert kein Spielbrett. Auf den anderen 150 Bildern habe ich das Spielbrett in unterschiedlichen Variationen platziert. Dabei habe ich auf verschiedene Variationen geachtet:

- Beleuchtung
- Rotation
- Umgebung
- Ohne Chips
- Mit Chips

Nachdem der Datensatz erstellt wurde, habe ich sie in einer Ordnerstruktur abgelegt und Bild für Bild definiert, ob sich ein Spielbrett in diesem Bild befindet bzw. an welcher Position. Dafür habe ich von OpenCV (Version 3.4.0) das Tool opencv_annotation.exe verwendet (Siehe Abbildung 3.1 und 3.2).

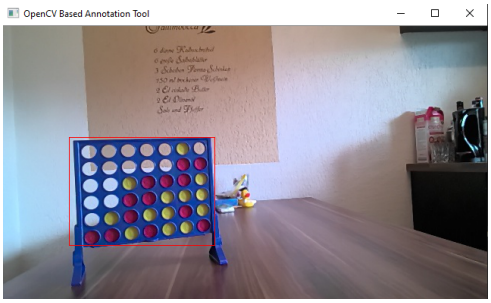

Abbildung 3.1: Befülltes Spielbrett bei heller Umgebung

Abbildung 3.2: Leeres Spielbrett bei dunkler Umgebung

Mit diesem Datensatz kann man anschließend mithilfe des Tools opencv_traincascade.exe von OpenCV (Version 3.4.0) das Training starten. Die Trainingsdauer hat bei meinen konfigurierten Parametern dabei ca. 50 Minuten gedauert. Hier ist, ebenso wie beim Training eines CNNs, auf ein Overfitting zu achten und dieses zu vermeiden. Als Ausgabe bekommt man schlussendlich eine XML-Datei, die die einzelnen Gewichtungen der Haar-Features beinhaltet.

3.2.3 Verwendung in Unity

Mithilfe von OpenCV kann in Unity ein CascadeClassifier erzeugt werden. Diesem kann man die im vorherigen Schritt erzeugte XML-Datei übergeben. Bei der Überprüfung eines Bildes auf den Haar Classifier zur Detektion einer Bounding Box wird dabei das Bild schrittweise durchgegangen, um so das passendste Match zu finden, falls vorhanden.

Das Ergebnis dieses Schritts wird dann für die weitere Erkennung der einzelnen Spielchips verwendet. So kann ein großer Teilbereich für die intensivere Verarbeitung ausgespart werden (Siehe Abbildung 3.3).

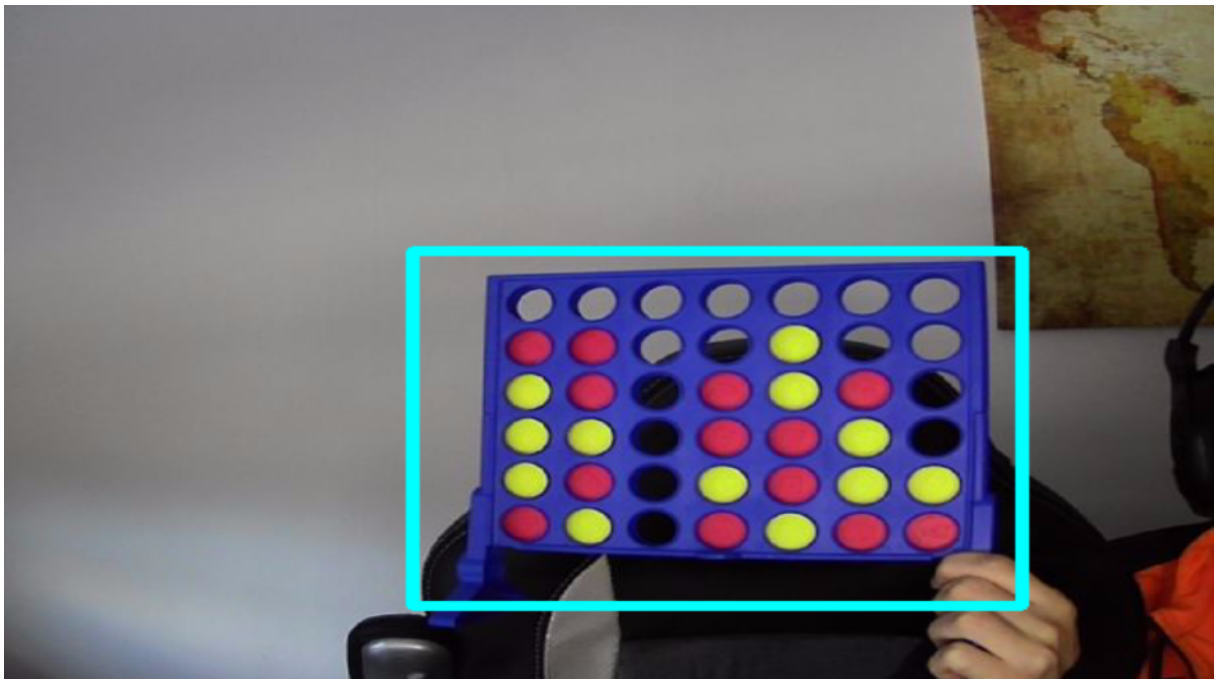

Abbildung 3.3: Erkennung der Region des Spielbretts mit Haar Cascades.

3.2.4 Vor- und Nachteile

Im Vergleich zum Feature Matching kann man es als Nachteil ansehen, einen Datensatz erstellen zu müssen. Ist dieser aber erstmal vorhanden, verfügen Haar Cascades aber eine deutlich bessere Unabhängigkeit gegenüber Störquellen wie Beleuchtung oder Rotation. Nachdem die Haar Cascades auf primitive Features zurückgreifen und das Training bereits vor der eigentlichen Ausführung der Detektion ausgeführt wird, kann ebenso eine hohe Geschwindigkeit gewährleistet werden. Im Vergleich zu CNNs, die eine deutlich höhere Parameteranzahl innerhalb eines Modells besitzen können, schneiden diese aber hier schlechter ab. Außerdem können mit meinen Trainingsdaten keine teilverdeckte Spielbretter erkannt werden. Das ist in unserem Projekt aber kein relevanter Nachteil, da alle Chips erkennbar sein müssen, um eine Prognose für den nächsten Spielzug erstellen zu können.

Kapitel 4

Performance

Die ersten Prototypen unseres Projektes liefen in Unity mit unter 10 FPS. Zwar funktionierten die einzelnen Funktionalitäten der Bilderkennung, jedoch würde das auf anderen Plattformen sehr wahrscheinlich zu Performanceproblemen führen. Während der Umsetzung sind wir dabei auf einige Optimierungen gestoßen, die bei einer Umsetzung eines Projektes mit Unity unbedingt miteinbezogen werden sollten.

4.1 Memory Leaks

Wir konnten relativ schnell feststellen, dass nach kurzer Zeit Objekte aus OpenCV nicht direkt automatisch wieder freigegeben werden. Das führt zu immer mehr belegtem Arbeitsspeicher, was schlussendlich zu einem Absturz der Anwendung führt. Anfangs hatten wir zur Behebung dieses Fehlers die Unity-Funktion Resources.UnloadUnusedAssets() verwendet, mit dem das Problem des Memory Leaks grundsätzlich behoben war.

Bei einer späteren Optimierung, bei dem die Erhöhung der FPS im Fokus war, konnte ich feststellen, dass diese Funktion sehr rechenintensiv ist. Ab diesem Zeitpunkt haben wir Objekte, die von OpenCV verwendet werden, mit der Funktion <Object>.Dispose() manuell wieder freigegeben, nachdem sie nicht mehr verwendet werden. Das verhindert die Erzeugung von Memory Leaks und ist deutlich weniger rechenintensiv.

4.2 Multi-Threading

Um den niedrigen FPS entgegenzuwirken, habe ich für die Berechnungen der Bilderkennung und Spielzugprognose einen eigenen Thread hinzugefügt. Damit werden diese nicht mehr im gleichen Thread wie der Unity-Engine ausgeführt. Das hat zu einer deutlichen Performanceverbesserung geführt. Ab diesem Zeitpunkt konnten wir bei 200-300 FPS erreichen, da damit nicht nur ein Kern ausgelastet wird.

Kapitel 5

Plattformen

5.1 PC

Die Entwicklung der Anwendung wurde von Anfang an zuerst mit Unity für den PC optimiert. Die Anbindung der Kamera funktionierte dabei problemlos, Performanceprobleme konnten wie im vorherigen Kapitel beschrieben gelöst werden. Der Zugriff auf die Haar Cascade XML-Datei stellt keine Probleme dar.

5.2 WebGL

Webanwendungen werden heutzutage nach Rourke (2018) immer noch nachgesagt, im Vergleich zu nativen Anwendungen performant schlechter abzuschneiden. Deshalb haben wir, wo möglich, schon im vorhinein versucht eine gute Performance zu erreichen. Mithilfe der Web Graphics Library (WebGL) konvertiert Unity das Programm in eine browserfähige Anwendung. Durch die Transpilierung des Quellcodes und Konvertierung der Modelle benötigt der WebGL-Export jedoch einige Minuten Zeit, was das Debugging erschwert.

Im Gegenzug zur Ausführung auf dem PC ist der direkte Aufruf einer XML-Datei, die als Resource von Unity mit ausgeliefert werden kann, aufgrund von Sicherheitsbeschränkungen nicht möglich. Mit der Auslieferung als StreamingAsset ist dies aber doch möglich.

Ein größeres Problem ist aber die Bibliothek OpenCVSharp, das vom Unity Asset „OpenCV plus Unity“ installiert worden ist. Der Export als WebAssembly für die Webanwendung scheint, im Gegensatz zur kostenpflichtigen Version „OpenCV for Unity“, nicht möglich sein. Ein ebenso ähnlich heißendes Projekt „OpenCvSharp“ unterstützt zwar den Export als WebAssembly, jedoch Unity nicht. Die etwas wirre Namensgebung führt schlussendlich zu der Erkenntnis, dass die Umsetzung unseres Projektes mit diesem Asset grundsätzlich zu keinem validen Export als WebAnwendung führen

kann.

5.3 Android

Die XML-Datei der Haar-Cascades wird bei Android automatisch in das Apk-Archiv gepackt. Um diese verwenden zu können, muss sie zuerst in eine eigene Datei auf dem Android-Gerät exportiert werden. Anschließend kann sie, wie am PC, verwendet werden.

Leider gibt es, wie bei den Webanwendungen, Probleme bei der Verwendung der Bibliothek OpenCVSharp. Hier tritt der Fehler „DllNotFoundException: Unable to load DLL 'OpenCvSharpExtern': The specified module could not be found“ auf. Der Fehler konnte mithilfe des adb-Tools ausgelesen werden. Meine Testgeräte, die ich zur Verfügung hatte, bauen alle auf ARMv8 auf. Anscheinend unterstützt „OpenCV plus Unity“ aber nur ARMv7. Das konnte ich aber nicht nachprüfen, da ich kein Testgerät zur Verfügung hatte. Nach den Dateien, die in das Unity Projekt eingebunden wurden (Siehe Abbildung 5.1), scheint das aber eine valide Vermutung zu sein.

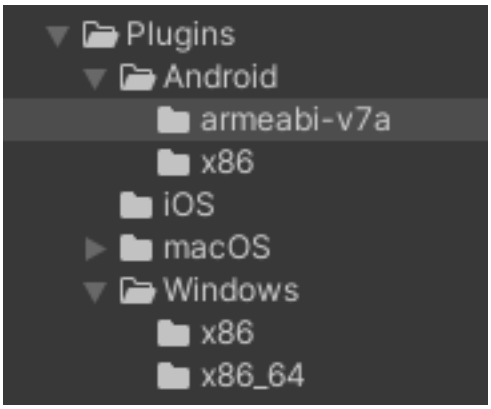

Abbildung 5.1: Zur Verfügung gestellte Bibliotheken für Android.

Kapitel 6

Zusammenfassung und Ausblick

Die Umsetzung des Projekts sowie die Erfahrungen, die gesammelt werden konnten, waren sehr aufschlussreich. Das Training der Agenten gestaltete sich als sehr spannend, da Probleme aufgetreten sind, die ursprünglich nicht vorhersehbar waren. Beim Training sollte frühzeitig auf eine gute Belohnung der Agenten geachtet werden. Bei einem Brettspiel, das spielzugbasierend ist, sollte nicht nur das Spielergebnis belohnt werden, sondern auch die individuellen Spielzüge, wenn die insgesamt mögliche Spielzuganzahl hoch ist.

Bei der Erkennung des Spielbretts konnte ich feststellen, dass unterschiedliche Beleuchtungen durchaus noch Probleme darstellen. Um dies zu verbessern, wäre zu prüfen, ob ein größerer Datensatz für das Training der Haar Cascades dem entgegenwirken. Dabei sollte im Datensatz auf genügend unterschiedliche Lichtverhältnisse geachtet werden. Eventuell könnte man hier mithilfe von Data Augmentation die existierenden Datensätze wiederverwerten und somit die Datenmenge, auf die trainiert wird, ohne großem Aufwand erhöhen.

Der Export als Webanwendung mit dem Unity Asset „OpenCV plus Unity“ ist nicht möglich. Hier muss die kostenpflichtige Version „OpenCV for Unity“ verwendet werden, was ebenso wie der manuelle Export von OpenCV, wie von Zhuravleva (2020) beschrieben, zu prüfen wäre. Außerdem wäre zu prüfen, ob eine der anderen Bibliotheken ARMv8 unterstützt, damit die Anwendung auch auf Android ausgeführt werden kann.

Literaturverzeichnis

Baby, N. & Goswami, B. (2019). Implementing Artificial Intelligence Agent Within Connect 4 Using Unity3d and Machine Learning Concepts. *International Journal of Recent Technology and Engineering (IJRTE)* (7), 193–200.

Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001. (2001). IEEE Comput. Soc.

Rourke, M. (2018). *Learn WebAssembly*. Birmingham: Packt Publishing.

Viola, P. & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* (S. I-511-I-518). IEEE Comput. Soc. doi: 10.1109/CVPR.2001.990517

Zhuravleva, A. (2020). *Progressive Web Camera Application using OpenCV WebAssembly module* (Master's thesis). Aalto University, Espoo.