Project Report: BamBirds 2019

Creating an Intelligent Game Playing Agent for Angry Birds

Samet Akcabay

Patrick Haller

28. September 2019

Inhaltsverzeichnis

1 Aufgabenstellung des Smart Environment Projekts

AngryBirds, ein in 2009 von Rovio Entertainment entwickeltes Casual-Puzzle-Videospiel, ist ideal als Maßstab für künstliche Intelligenz geeignet. Der Weg zum Erfolg setzt voraus, dass ein Verständnis der physikalischen Zusammenhänge auf das Spiel übertragen werden können und strategische Entscheidungen getroffen werden müssen. Ziel des Projekts war es, die Schwächen des BamBirds-Agenten zu verbessern und um Funktionen zu erweitern, um mit diesem an der Angry Birds AI Competition teilnehmen zu können und diesen zu gewinnen.
Parallel zu diesem Wettkampf, lief die sogenannte AIBirds COG 2019 Level Generation Competition, in welchem das Ziel daraus bestand, mit einem selbsterstellten Programm der Wahl, automatisiert Level zu generieren, welche viel Spaß und Herausforderung anbieten sollen. Dieser Aufgabe haben wir uns gewidmet.

2 Machine Learning als möglicher Levelgenerator

2.1 Hintergrund [wie ist Patrick drauf gekommen?]

Im Gegensatz zu dem gegebenen Levelgenerator, wollten wir einen Ansatz wählen, der nicht auf prozedualer Generieriung beruht, sondern mit Machine Learning Techniken arbeitet.

Der Generator soll anhand von bestehenden Leveln lernen und daraus neue generieren.

2.2 Generative Adversial Networks (GAN)

Generative Adverserial Networks, kurz GAN (zu deutsch ërzeugende generische Netzwerke"), stellen in der Informatik eine Gruppe von Algorithmen zu unüberwachtem Lernen dar. Sie bestehen aus zwei Neuronalen Netzwerken, eines erstellt Kandidaten (**Generator**), das zweite bewertet diese (**Diskriminator**).
Der Generator lernt, Ergebnisse nach einer bestimmten Verteilung zu erzeugen. Der Diskriminator hingegen lernt, die Ergebnisse des Generators gegen die echte, vorgegeben Verteilung zu evaluieren (hier: konkret spielbare Level). Findet der Diskriminator keine Unterschiede mehr im direkten Vergleich der vorgegebenen Verteilung, so wird das Ziel erreicht.
Neuronale Netzwerke kommen häufig zur Visualisierung verschiedener Gegenstände, zur Erstellung von 2D- bzw. 3D-Modellen oder zur Bildbearbeitung (astronomischer Bilder) zum Einsatz.
Aus diesem Grund entschieden wir uns für GAN als geeigneten Kandidaten eines Level-Generators.

2.3 ...

3 Allgemeine Vorgehensweise

Im Folgenden wird die allgemeine Vorgehensweise beschrieben. Zur besseren Strukturierung, Dokumentierung und klaren Aufgabenverteilung, haben wir uns ein eigenes GitHub-Repository [1] aufgesetzt, in welchem ebenfalls die Arbeitszeiten an den bestimmten Issues festgehalten

[1] https://github.com/HallerPatrick/AI-Birds-LeveGANerator

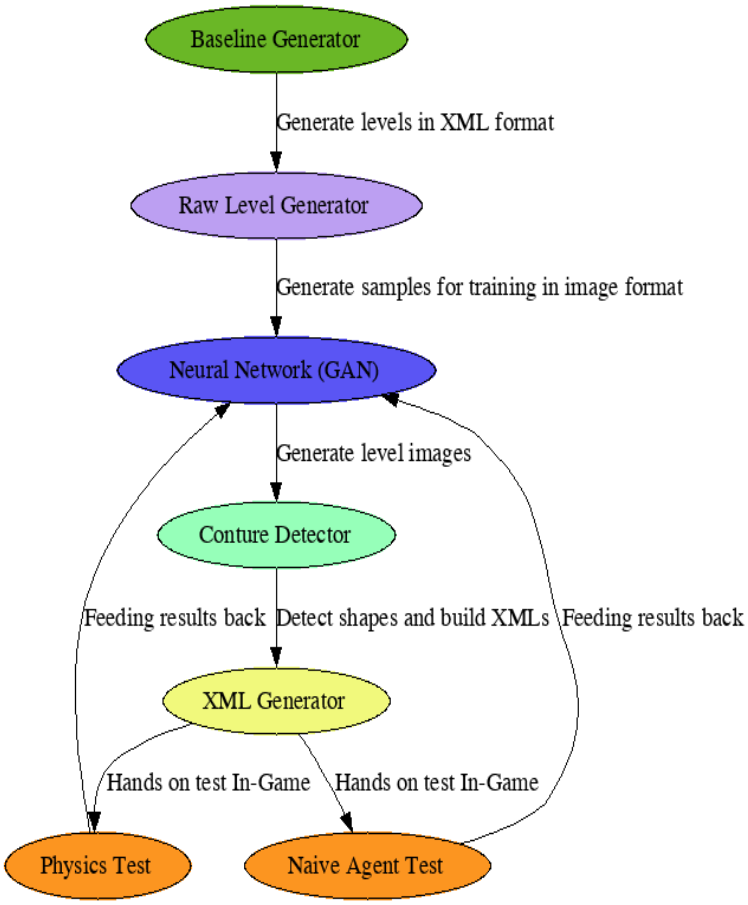

Abbildung 1: Projektstruktur dem Sourcecode entsprechend

wurden. Die wichtigsten Aufgaben werden in den nachfolgenden Unterkapiteln näher erläutert. [Graphen, allgemeine Struktur]

3.1 Projektstruktur

3.1.1 Baseline Generator

Der Baseline Generator beschreibt, den XML Levelgenerator der vom Wettbewerb zur Verfügung gestellt. Dabei wurden minimale Änderungen vorgenommen um ihn automatisiert zu verwenden.

3.1.2 Raw Level Generator

Der Raw Level Generator generiert anhand von XML Dateien, die vom Baseline Generator kommen, Bilder die einzelnen Spieleobjekte darstellen. Für jede Art von Spielobjekt, heißt *Schweine*, *Platformen*, *TNT* und *Blöcke*

Der Grund ist, dass die Darstellung von mehreren Spieleobjekten in einem Bild "überfüllen"würde. Der Grund wird in der Sektion 3.1.3 weiter ausgeführt.

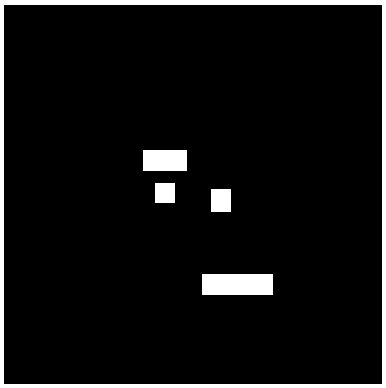

Abbildung 2: Schweine Spieleobjekte als Schemadarstellung

3.1.3 Neural Network (GAN)

Dem GAN Model, mit der *Keras* [2] Library gebaut, nimmt die "Raw Level", wie in *Abbildung 2*, und wird anhand dieser trainiert. Dabei werden wird mit dem gleichen Model 4 verschieden Generatoren trainiert, für jedes Spielobjekt einer. Das ist notwending, da wie schon in 3.1.2 erwähnt, das Bild sonst zu überfüllt wäre. Das würde zu Ergebnisses führen, wie in *Abbildung 3*.

3.1.4 Conture Detector

Der *Conture Detector* liest aus den generierten Bilder der trainierten GAN Modelle die einzelnen Konturen aus. Die genaue Beschreibung der Ermittlung der Koordinaten wird in Sektion 3.2 beschrieben. Von Diesen werden dann die Koordinaten ermittelt und an den XML Generator übergeben werden. (siehe 3.1.5)

3.1.5 XML Generator

Der XML Generator konstruiert aus den gelabelten Koordinaten valide Level XML Dateien, die von der Unity Version von Angry Birds eingelesen werden kann.

3.1.6 Physics and Naive Agent Test

Das Testsystem, wird im Genauren in Sektion 3.5 beschriebene. Es dient hauptsächlich um SSanity Checksän den generieten Level durch zuführen. Um so gefiltert, "gute"Level erneut in das GAN Model zu füttern.

3.2 Erkennen der Konturen von Zentroiden

Um generell mit den generierten Bildern arbeiten zu können war es wichtig, diese erst durch die Erzeugung und Verstärkung der Konturen besser auf den Konturenerkenner abzustimmen. Nachfolgend möchten wir die Klasse *conture_ detector.py* näher erläutern.

[2] https://keras.io/

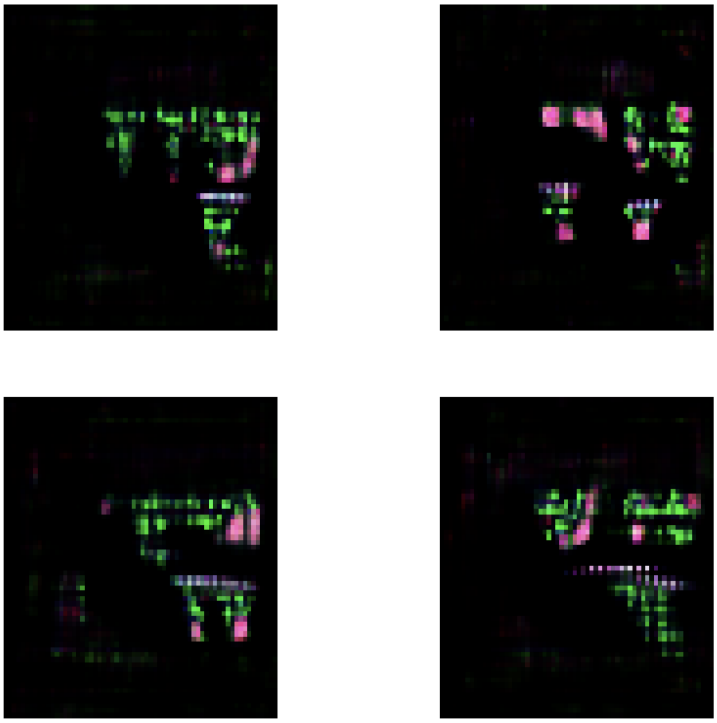

Abbildung 3: Verschwommene Konturen führen zu Problemen bei der Erkennung

Zur allgemeinen Verarbeitung der Bilder bedarf es eines Imports der cv2-Klasse, welches zuvor installiert werden muss (unter OS X und Python3: *"pip3 install opencv-python"*). In der Klasse wird nun das Bild ausgelesen und alle x- sowie y-Koordinaten in eine Liste **points* geschrieben. Das eingelesene Bild im HSV-Farbraum ("**h**ue": Farbwert, "**s**aturation": Farbsättigung, "**v**alue": Hell-/Dunkelwert)[3] wird in Graustufen umgewandelt, um Konturen innerhalb des Bildes zu ermöglichen (dieser Schritt kann allerdings auch übersprungen werden, da die Objekte verschiedene Farben aufweisen).
Zur Ermittlung der Zentroiden wird nun die Formel

$$centrX = \sum XCoord/Length$$

wobei folgende Zuweisungen gelten:

- $centrX$ entspricht den zu ermittelnden Zentroiden der X-Koordinaten (gilt analog für Y-Koordinaten mit $centrY$)
- $\sum XCoord$ entspricht der Summe aller X-Koordinaten (gilt analog für Y-Koordinaten mit $\sum YCoord$)
- $Length$ entspricht der Länge aller Punkte

[3] https://de.wikipedia.org/wiki/HSV-Farbraum

3.3 Automatisierung der Abläufe mithilfe von Powershell unter Windows

Um zu vermeiden, dass alle Komponenten einzeln und umständlich gestartet werden müssen, haben wir es uns außerdem zur Aufgabe gemacht, ein Automatisierungsskript aufzusetzen, welcher diesen Schritt für uns übernimmt. Als Skriptsprache erschien uns PowerShell am sinnvollsten, da dieses ein fester Bestandteil von Windows 10 (dem gängigsten Betriebssystem) ist und aufgrund dessen keine extra Tools installiert und erläutert werden müssen.
Das Skript startet zunächst ScienceBirds und skaliert diesen mithilfe von Window-Resizer (Tool zur nutzerbasierten Steuerung der Standardgröße eines Fensters, hier: ScienceBirds) auf einen bestimmten Wert, da der Agent sonst mit der Größe des ScienceBirds-Fensters nicht einverstanden ist. Im Anschluss wird der Agent und der Server automatisch in der Eingabeaufforderung gestartet. Wir konnten beobachten, dass durch den Startklick des Skripts alle benötigten Fenster ordnungsgemäß gestartet wurden und der Agent das Spiel wie erwartet, selbstständig gespielt hat.
In den folgenden Stichpunkten werden die einzelnen Fenster der Automatisierung näher erläutert (vgl. Abb.1).

1) Shell des Automators
   Aus diesem Fenster wird der Automator gestartet. Dazu wird zunächst überprüft, ob alle Umgebungsvariablen vorhanden und passend eingestellt sind, um volle Funktionalität gewährleisten zu können. Findet der Automator alle benötigten Tools, so wird der Nutzer gefragt, ob er diesen starten, den Vorgang abbrechen oder Umgebungsvariablen einstellen möchte.

2) ScienceBirds-Fenster
   In diesem Fenster läuft das eigentliche Spiel. Da die Größe dieses Fensters für unseren Agenten nicht gepasst hat, musste ein Window-Resizer eingeschaltet werden, um ScienceBirds auf die benötigte Fenstergröße zu skalieren.

3) AutoSizer-Fenster
   Hier befindet sich unser Window-Resizer. Dabei haben wir uns für den AutoSizer als Tool entschieden, da dieser wenig Speicherplatz einnimmt, kostenlos und ohne Bloatware zu installieren ist. Damit die Größe eines spezifischen Fensters auf einen immer gleichbleibenden Wert angepasst werden muss, haben wir im AutoSizer selbst einen Hotkey festgelegt, welcher beim Betätigen einer bestimmten Tastenkombination (hier: "%9") die Größe von einem Fenster mit dem Namen SScienceBirdsäuf die benötigte Auflösung ändert. Diese Methode ist stark auf unser persönliches System abgestimmt, kann aber durch wenige Änderungen im Skriptcode auf die eigenen Parameter angepasst werden.

4) Agent-Fenster
   Hieraus wird der Agent gestartet. Dazu wird zuerst der Agent gestartet und im Anschluss darauf der Server. Hier lässt sich beobachten, dass sich der Status von "Waitingäuf "Pending"ändert. Startet man nun den Agenten mit dem Klick auf SStart", so wechselt der Agent in den Status "Running".

5) Server-Fenster Der Server muss gestartet werden um gewährleisten zu können, dass der Agent ordnungsgemäß läuft. Hier kann man die verschiedenen Aktionen beobachten, welche gerade stattgefunden haben (hier: "Go to Level Selection Page").

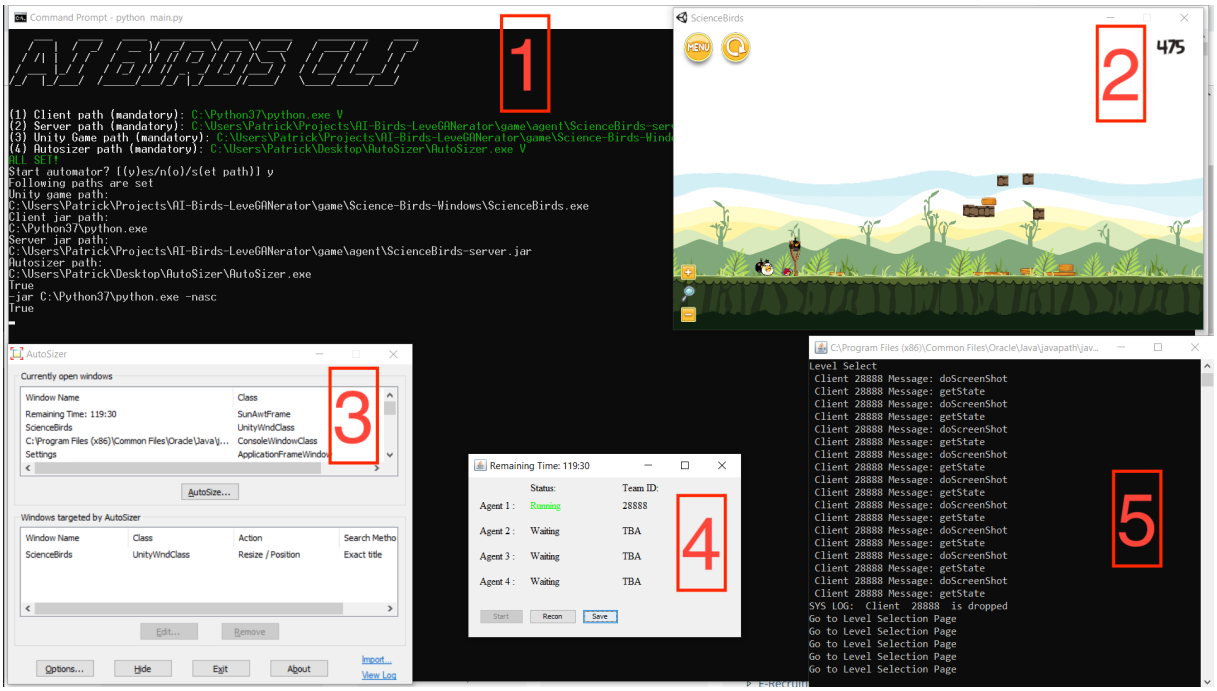

Abbildung 4: Vorläufige GUI der Automatisierung

3.4 Verdeutlichen der Konturen der erzeugten RAW-Bilder zur besseren Erkennung vor Training

Der Konturen-Erkenner hatte in unserem Durchlauf Probleme, die Umrisse der erzeugten Zentroide zu erkennen. Aufgrund dessen mussten wir die Konturen, welche erzeugt werden, vor dem Beginn des Trainings verstärken, damit diese eindeutig von unserem Erkenner gefasst werden können.

3.5 Aufsetzen eines Re-Evaluierungssystems

Das Re-Evaluierungssystems dient hauptsächlich als manuel implementiertes rückgekoppeltes Neural Network System. Die neuronale Netzwerk geneiert neue Bilder anhand bereits bekannten Bildern auf die das Model trainiert wurde, als auch auf einem zufälligen initialisiertem Vektor, der für die Varianz in dem Bildern sorgt. Dies führt dazu das auch Level geniert werden die nicht sinnvoll physikalisch sind. Das Ziel ist es dem Model rückgekoppelt Bilder einzuspeißen, dass sich die Varianz in sofern verändert, dass die Level grundlegenden Test, wie physikalischen Tests standhält.
Dafür werden neu generierte Level in die Unity-Version des Spiel reingeladen und von dem BamBird Naive Agent gespielt. Fällt nun das Level direkt nach dem Start in sich zusammen, kann der Agent seinen ersten/nächsten Zug nicht evaluieren und versuch neue Screenshots des Levels zu machen um eine erneute Evaluation durchzuführen. Nach einer gewissen Anzahl an Iteration schreibt der modifizierte Agent die ID des Levels in eine externe Datei, die nun ausgelesen werden kann. Die Selektion der Level die ßpielbar"waren, werden nun gesammelt und wiederverwertet.

Ein weiterführender Test wäre der Naive Agent Test, der das Level auf seine Spielbarkeit prüft. Dies konnte jedoch nicht mehr implementiert werden.

4 Ergebnisse

Wir haben unsere Generative Adversial Networks nun zur Generierung von Plattformen, Schweinen, TNTs und Blöcke eingesetzt. Dabei stellen wir für die verschiedenen Gegenstände fest, dass unterschiedliche Ergebnisse erzielt wurden. Im ersten Fall widmen wir uns den Plattformen. Bei diesen fällt auf, dass aus dem Ursprungsbild Konturen herausstechen und eine Form annehmen. In unserem Beispiel haben wir knapp sechs Tausend, sich immer weiter verbessernde Bilder generieren lassen. Auffällig dabei ist jedoch, dass die Konturen und Formen, die Plattformen darstellen sollen, immer flächenartiger werden und somit eine geeignete Grundlage darstellen sollten, diese allerdings in ihrer Struktur sinnfrei und zum Teil identisch mit einem vollständig flachen Level ist. Es sind weder Plattformen in der Höhe bzw. außergewöhnliche Formen aufzufinden, welche den Spielspaß verbessern könnten.
Anders sieht es hier bei den punktförmigen Gegenständen, wie den Schweinen und TNTs, aus. Diese können mithilfe des GAN bisher gut innerhalb des Levels verstreut werden. Es kommt weiterhin vor, dass Schweine sehr nah beieinander stehen und deshalb in den GAN als eine Linie generiert werden (si. Verdeutlichen der Konturen der erzeugten RAW-Bilder zur besseren Erkennung von Training).
Leider besteht auch hier das Problem, dass Faktoren wie Spaß und Herausforderung (bisher) nicht eingebracht werden konnten, weswegen die Verteilung der einzelnen Objekte derzeit willkürlich ausfällt. Dieses Problem zieht sich auch durch die anderen Gegenstände.

Ein weiteres Problem stellt die Verteilung der punktförmigen Gegenstände dar. Am Beispiel der Schweine fällt diese auf den Blick zwar sinnvoll aus, da diese passend über das Level verstreut oder nachvollziehbar angeordnet sind. Nur häufig finden sich keine Blöcke unter den Gegenständen, sodass diese beim Start des Levels durch das Fallen entweder zwingend ihren Standort verändern oder sogar ohne Einwirkung des Spielers sofort sterben. Dies ist jedoch fatal für das Spielprinzip, da die Schweine nur durch die Aktionen des Spielers zu eliminieren sein sollten (vgl. Abbildung 3).

5 Fazit

Aus unseren Ergebnissen wird deutlich, dass GAN weniger zum Generieren von Level geeignet ist, da die erzeugten Level nicht sinnvoll sind. Das heißt, dass häufig obsolete Plattformen vorzufinden sind, welche in ihrem Aufbau und ihrer Struktur kein spielbares Level darstellen. Zudem erfolgt die Platzierung der Schweine und Hindernisse ohne jeglichen Wert auf Faktoren wie Spaß und Herausforderung zu legen, weswegen die generierten Level wenig für den Spieler bieten zu haben. Aufgrund dessen finden GANs ihren Einsatz oft nur bei der Erstellung photorealistischen Bildern zur Visualisierung verschiedener Gegenstände, bei der Modellierung von Bewegungsmustern in Videos, bei der Erstellung von 3D-Modellen von Objekten aus 2D-Bildern und bei der Bildbearbeitung von astronomischen Bildern. Da unsere Methode der Neuronalen Netze keine brauchbaren Ergebnisse geliefert hat, wurde dieser beim Levelgenerator-Wettkampf nicht eingereicht.

Abbildung 5: Ergebnis eines durch Neuronale Netze generierten Levels

6 Ausblick

Für zukünftige Teams, die sich mit dem Projekt ÄI Birds"des Lehrstuhls für Smart Environments befassen wollen, möchten wir als Abschluss Verbesserungsvorschläge des aktuellen Stands erläutern. Für die genannten Punkte bietet es sich an, den bereits vorhandenen Fortschritt zu verwenden und weiter auszubauen. Natürlich ist es aber auch möglich, komplett eigene Ansätze anzuwenden bzw. den vorhandenen Stand neu zu entwerfen.
Die Automatisierung der Abläufe funktioniert bereits einwandfrei und gemäß der Erwartungen. Trotzdem wäre es sinnvoll. eine GUI (z.B. mit JavaFX) zu entwerfen, um diesen Prozess überhaupt optisch darzustellen und dem Nutzer vor allem Übersicht über das Geschehen zu gewähren. Visuelles Feedback, was genau gerade passiert, wie weit der Automator vorangeschritten ist oder ob Fehler aufgetreten sind (mit genauem Fehlercode) sind sinnvolle Ergänzungen.
Eine weitere sinnvolle Ergänzung wäre die Umstellung auf Bordmittel, sodass so wenig wie möglich zusätzlich vom Nutzer installiert und angepasst werden muss. Der aktuelle Stand des Automators ist, dass der Code auf unser System zugeschnitten und zusätzlich Python, sowie der AutoSizer benötigt werden. Dies ist allerdings keine effiziente Lösung, weswegen eine Anpassung dem Nutzer ungemein helfen könnte. Dabei können externe Tools natürlich weiterhin installiert werden, die Installation davon sollte nur automatisch erfolgen. [Verbesserungsvorschläge][Wie kann man GAN passend für Level-Generierung gestalten?] [RNN als mögliche Alternative?]

6.1 GAN im Vergleich zu Alternativen

Im direkten Vergleich von GAN mit den Alternativen empfiehlt es sich, die Meinungen und Erfahrungen von Experten heranzuziehen, die diese auch in der Praxis umgesetzt und angewendet haben, um so am effektivsten darüber urteilen zu können, wie GAN in Abhängigkeit seiner Alternativen abschneidet. [...]

6.2 Weitere Tools als allgemeine Hilfestellung für das Projekt

Im Laufe des Projekts war es gelegentlich der Fall, dass von uns entwickelte Tools für das Projekt obsolet wurden und wir diese letztlich nicht effektiv in unsere Aufgabenstellung miteinbringen konnten. Da diese aber trotzdem aufgebrachte Zeit für das Projekt in Anspruch genommen haben und sicherlich für zukünftige Teams von Vorteil sein könnten, möchten wir diese ebenfalls in diesem Abschnitt kurz erwähnen und in unsere Abgabe mit einfließen lassen. Einige dieser Tools sind bereits "ready-to-use", andere müssen jedoch ergänzt und weiter verfeinert werden.

6.2.1 Doctor

Dieses Tool wurde von Patrick Haller entwickelt und überprüft im eigenen System, ob die Umgebungsvariablen, welche für die Ausführung des Agenten und des Servers benötigt werden, existieren. Um den Doctor auszuführen steuert man in seinem Terminal das Verzeichnis other/doctor an und führt mit Python die *doctor.py*-Datei aus (unter OS X aus dem Root-Verzeichnis: "*cd ./other/doctor*und führt den Befehl *"python3 doctor.py"* aus). Das Tool überprüft nun, ob SWI-Prolog und die Nebenpakete von Python installiert sind. Die Version

des Python-Interpreters wird ebenfalls überprüft, und falls diese nicht dem aktuellen Stand entspricht (Stand 24.09.2019: Python 3.7), erfolgt eine Aktualisierung. Zudem muss der Java-Pfad ebenfalls ordnungsgemäß konfiguriert sein, weswegen der Doctor diesen in seine Checks miteinbezieht. Auf OS X kommt Homebrew bei der Installation zum Einsatz, unter Linux verwenden wir den Standard-Paketmanager und unter Windows haben wir uns für Chocolatey als Paketmanager entschieden. Auf Linux läuft dieser leider noch nicht einwandfrei, da manche Pakete nicht auf Linux zum Laufen gebracht werden konnten.

6.2.2 JSON-XML-Parser

Zur Generierung der Level war es notwendig, vorhandene Level, welche im JSON-Format codiert wurden, in äquivalente XML-Formate umzuwandeln. Dazu entwickelten wir einen XmlWriter sowie JsonToXmlParser in Python.
Der XmlWriter schreibt zunächst den standardisierten Kopf der XML-Dateien, welche die Codierung, die Breite des Levels (hier: 2) sowie die Minimum-/Maximum-Weite der Kamera beinhaltet. Der Parser liest nun die JSON-Datei und schreibt für jeden Block mithilfe der äddMethoden des XmlWriters die XML-Datei. Jeder eingelesene Vogel, Block, Schwein etc. wird dabei in seine eigene Liste eingeschrieben (inklusive der Attribute, z.B. ïd", "rotation"...). Nach dem Erreichen des Endes der JSON-Datei erfolgt letztendlich der Abschluss des XML-Dokuments, bei welchem das Tag der GameObjects und vom Level geschlossen werden.

6.2.3 Re-Evaluierungssystems

Das Re-Evaluierungssystems kann modularisiert werden und eigenen sich auch um automatisiert weitere Levelgeneratoren oder auch Agenten zu testen. Hierfür müsste die Schnittstelle angepasst werden, wie die Informationen, die während des Spiels gesammelt werden, aufgezeichnet werden. Desweiteren wurde das Automatisierungskript nur Windows entwickelt und sollte mehr "deployable"refaktorisiert werden.