Erstellung von KI-Trainingsumgebungen auf Grundlage von OpenAI Procgen

Praktikumsbericht von

Tobias Telge

Institut für Programmstrukturen und Datenorganisation (IPD)
Betreuender Mitarbeiter: Daniel Zimmermann

Das Ziel des Praktikums ist die Implementierung der von OpenAI entwickelten Procgen-Umgebung BigFish in der Programmiersprache Julia. Damit kann das ursprünglich in C++ geschriebene Spiel mit Julia auf der CPU gespielt werden. In diesem Bericht werden zunächst die Hintergründe des Praktikums erläutert. Dann folgt eine Beschreibung des gewählten Vorgehens. Zudem wird beschrieben, wie das fertig implementierte Spiel gestartet werden kann und welche Spieloptionen es gibt. Abschließend wird ein Ausblick über das weitere Vorgehen gegeben.

1. Hintergründe

Die 2019 veröffentlichten 16 Procgen-Umgebungen sind von OpenAI entwickelte Videospiele die im Kontext des Trainings von Künstlicher Intelligenz genutzt werden können [1]. Mit den prozedural generierten Umgebungen ist es möglich, die Qualität von Methoden des bestärkenden Lernens zu messen. Dabei wird überprüft, wie schnell ein Agent generalisierbare Fähigkeiten lernt.

Beim Procgen-Spiel BigFish spielt der Spieler einen Fisch der kleinere Fische fressen soll und dabei Fischen, die größer als er selbst sind, ausweicht. Durch das Fressen von Fischen erhält der Spieler eine Belohnung und wird größer. Wird er jedoch durch Kontakt mit einem größeren Fisch selbst gefressen, endet die Episode. Sobald der Spieler größer wird als alle anderen Fische erhält er eine große Belohnung und die Episode ist vorbei.

Das Praktikum findet im Kontext der Entwicklung sogenannter NaturalNets statt. Mit NaturalNet werden neuartige neuronale Netze bezeichnet, die ein vereinfachtes Modell biologischer Neuronen verwenden und biologisch plausibler sind als etablierte neuronale Netze [2].

Das Ziel des Praktikums ist es, das Procgen-Spiel BigFish in Julia zu implementieren, so dass es auf der CPU gespielt werden kann. Die ursprüngliche Programmiersprache des Spiels ist C++. In an das Praktikum anschließenden Projekten soll BigFish dann in Julia auf der GPU implementiert werden. Der Code dieses Praktikums soll dabei verwendet werden, um die GPU-Variante gegen die CPU-Variante zu testen. Durch die GPU-Variante soll das Training von NaturalNets beschleunigt werden, indem es auf der GPU durchgeführt wird. Julia bietet sich als Programmiersprache an, da für die Implementierung auf der GPU das Julia-Cuda Framework Cuda.jl genutzt werden kann.

2. Vorgehen

Zuerst werden in Unterabschnitt 2.1 der Aufbau des C++- und des Julia-Codes erläutert. Anschließend wird in Unterabschnitt 2.2 näher darauf eingegangen, wie Klassen und Vererbungen aus dem C++-Code in den Julia-Code überführt werden.

2.1. Aufbau

Im Folgenden wird zuerst der grundlegende Aufbau des C++-Codes skizziert.

Die konkreten Procgen-Spiele erben alle von der Klasse BasicAbstractGame. Die Klasse BasicAbstractGame wiederum erbt von der Klasse Game. Die beschriebenen Klassen-Beziehungen werden in Abbildung 1 dargestellt.

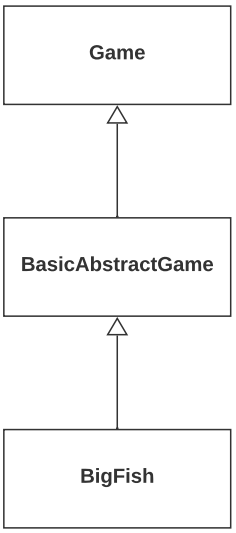

Abbildung 1: Die Klassen-Beziehungen im C++-Code

Die drei zentralen Methoden sind die step-, die reset- und die init-Methoden, die sich respektive um einen einzelnen Spielschritt, das Zurücksetzen des Spiels und die Initialisierung des Spiels kümmern. Große Teile dieser Methoden sind in den Oberklassen der

konkreten Procgen-Spiele implementiert. Das Zeichnen des Bildes findet in der Klasse BasicAbstractGame statt.

Zudem gibt es noch die Klasse Entity, die die Entitäten repräsentiert. Im Beispiel BigFish sind das die einzelnen Fische. Die Fische werden dabei im Code als Rechtecke behandelt.

Außerdem gibt es noch einige Hilfsklassen. Ein Beispiel ist die Klasse RandGen, welche sich um die Erzeugung von Zufallszahlen kümmert. In resources werden die in den Spielen verwendeten Bilder verwaltet.

Der Vorteil dieses Aufbaus ist, dass sehr viel Logik aus den konkreten Spielen ausgelagert wurde. Somit können einzelne konkrete Spiele durch Vererbung und Nutzung der Hilsklassen ohne viel zusätzlichen Aufwand implementiert werden.

Der Aufbau des Julia-Codes orientiert sich eng am Aufbau des C++-Codes. Eine genauere Beschreibung, wie die einzelnen Konzepte in Julia überführt werden, findet sich in Unterabschnitt 2.2.

Zur Visualisierung wird die Julia-Bibliothek Luxor verwendet. In der image.jl-Datei wird die Logik für ein einzelnes Bild implementiert. Dabei ist insbesondere die show-Methode von Bedeutung, die ein Bild in einem übergebenen Rechteck zeichnet. Der C++-Code verwendet stattdessen das Qt-Framework. Der Code für die Anzeige des Fensters sowie für die Verwaltung von Tastatureingaben wird unter Nutzung der Julia-Bibliothek MiniFB im Ordner tools implementiert. Mithilfe dieses Codes wird die Logik, um das Spiel zu starten und auszuführen, in der Datei visualization.jl implementiert. Der C++-Code verwendet dafür das OpenAI Gym-Framework.

2.2. Klassen und Vererbung

Da es in Julia keine Klassen gibt, werden im Julia-Code für jede C++-Klasse ein Struct als Datenstruktur verwendet. Diese Structs enthalten dabei die Attribute der jeweiligen C++-Klasse. Für die Methoden der C++-Klassen wird jeweils eine Methode implementiert, die beim Aufruf zusätzlich zu den Parametern im C++-Code den der Klasse entsprenden Struct als Eingabe erhält. Dieser Struct wird dann in der Julia-Methode entsprechend den Anweisungen in der C++-Methode manipuliert.

Auch Vererbungen wie in C++ sind in Julia nicht enthalten. Stattdessen wird im Julia-Code eine hat-Beziehung verwendet. So hat zum Beispiel der Struct BigFish ein Struct BasicAbstractGame als Attribut.

Um die Logik des Überschreibens von Methoden, also der Neudefinition einer vererbten Methode in einer Kindklasse, in Julia darstellen zu können, werden drei verschiedene sich ergänzende Konzepte verwendet.

Das erste ist Julias Typ-System. Dabei muss beachtet werden, dass in Julia konkrete Typen final sind und nicht Subtypen von konkreten Typen sein können, sondern nur von abstrakten Typen. Abstrakte Typen können im Gegensatz zu konkreten Typen wie Structs nicht instantiiert werden. Sie dienen ausschließlich zum Formen einer Typ-Hierarchie und beschreiben, wie konkrete Typen miteinander in Verbindung stehen. Um die Vererbung im C++-Code von Game nach BasicAbstractGame und von BasicAbstractGame nach BigFish in Julia darzustellen, wird daher zuerst eine Struktur abstrakter Typen aufgebaut. Somit ist der abstrakte Typ AbstractBigFish ein Subtyp des abstrakten Typs AbstractBasicAbstractGame. Dieser Typ ist wiederum Subtyp des abstrakten Typs AbstractGame. Der Struct BigFish ist als konkreter Typ ein Subtyp von AbstractBigFish. Analog ist der Struct BasicAbstractGame ein Subtyp von AbstractBasicGame und der Struct Game ein Subtyp von AbstractGame. Die beschriebene Typ-Hierarchie wird in Abbildung 2 dargestellt.

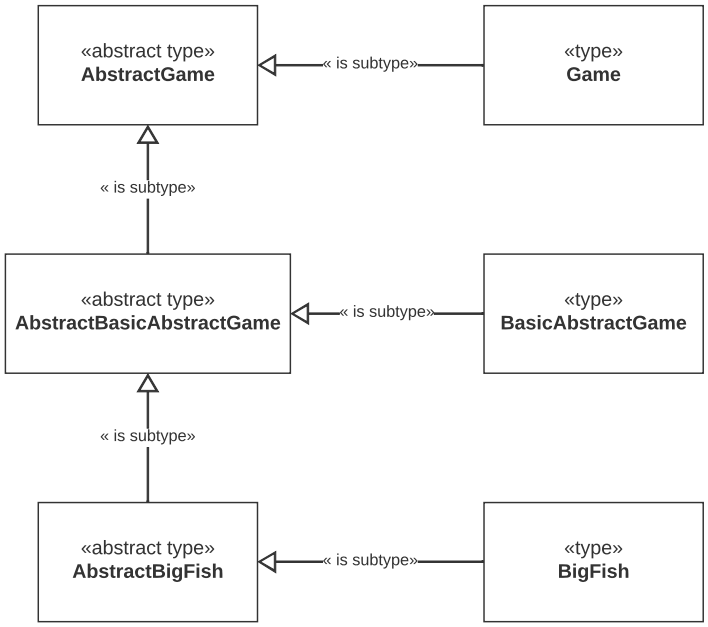

Abbildung 2: Die Typ-Hierarchie in Julia

Das zweite verwendete Konzept sind Julias Funktionen und Methoden. In Julia definiert eine Methode eine mögliche Verhaltensweise einer Funktion. Wenn beim Aufruf einer Funktion mehrere mögliche Methoden mit passenden Parametertypen existieren, so wird die spezifischste Methode ausgeführt. Damit und mit dem oben eingeführten Typ-System ist es nun in Julia möglich, dass dynamisch abhängig vom Typ die korrekte Methode ausgeführt wird. Im C++-Code ist zum Beispiel die Methode game_step sowohl in BigFish als auch in BasicAbstractGame und in Game enthalten. Im Julia-Code gibt es für die Funktion game_step dementsprechend auch drei Methoden. Diese Methoden haben entweder den abstrakten Typ AbstractBigFish, AbstractBasicAbstractGame oder AbstractGame als Parametertyp. Da in Julia immer die spezifischste existierende Methode ausgeführt wird,

wird bei Aufruf der Funktion game_step mit einem BigFish-Struct als Eingabe also auch die richtige Methode, und zwar diejenige mit AbstractBigFish als Parametertyp, ausgeführt.

Das dritte verwendete Konzept sind get-Funktionen. Um in einer Methode, die einen abstrakten Typen als Parametertyp hat, auf das konkrete Struct zugreifen zu können, werden entsprechende get-Funktionen verwendet. Die get_basic_game-Funktion kann zum Beispiel mit jedem Subtypen von AbstractBasicAbstractGame aufgerufen werden, und gibt dann das konkrete BasicGame-Struct zurück. Dafür werden wieder die oben beschriebenen ersten beiden Konzept verwendet.

3. Spielstart und Optionen

Das Spiel kann mithilfe folgender Anweisungen zum ersten Mal gestartet werden.

- Instantiiere die notwendigen Pakete wie in der Project.toml definiert durch Befolgen folgender Schritte:
  - Navigiere in einem Terminal zu dem Ordner, der die Project.toml enthält.
  - Führe `julia -project=.` aus, um ein julia REPL zu starten.
  - Gehe dann in den Paket-Modus, indem du] eingibst.
  - Führe das Kommando `instantiate` aus.
  - Nachdem alle Pakete instantiiert wurden, verlasse den Paket-Modus, indem du die Backspace-Taste oder STRG-C drückst.
  - Verlasse das julia REPL mit STRG-D.
- Führe `julia -project=. procgen/games/visualization.jl` aus, um das Spiel zu starten.

Die Spielweise entspricht dabei genau dem Originalspiel.

Zudem stehen verschiedene Optionen zur Verfügung, die in der visualization.jl-Datei durch Aufheben der entsprechenden Auskommentierung aktiviert werden können.

- `restrict_themes = true` - Alle Fische außer dem Spieler sehen gleich aus.
- `use_monochrome_assets = true` - Das Spiel verwendet einfarbige Rechtecke anstatt Bildern.
- `use_backgrounds = false` - Das Spiel verwendet einen schwarzen Hintergrund.
- `use_sequential_levels = true` - Wird das Ende eines Levels erreicht, endet die Episode nicht, sondern ein neues Level wird gestartet.
- `distribution_mode = EasyMode` - Der Schwierigkeitsgrad des Spiels wird gesenkt.

4. Zusammenfassung und Ausblick

In diesem Praktikumsbericht wurde erläutert, wie das Procgen-Spiel BigFish in Julia implementiert wurde, so dass es auf der CPU gespielt werden kann. Durch den Code-Aufbau ist es zudem möglich, weitere Procgen-Spiele ohne viel Aufwand zu implementieren.

In zukünftigen Projekten sollen BigFish und weitere Procgen-Spiele in Julia auf der GPU implementiert werden, um das Training von Künstlicher Intelligenz wie zum Beispiel NaturalNets zu beschleunigen. Dabei kann der in diesem Praktikum entstandene Code helfen. So kann dann die GPU-Variante gegen die CPU-Variante getestet werden.

Literatur

[1] Karl Cobbe u. a. „Leveraging Procedural Generation to Benchmark Reinforcement Learning“. en. In: *Proceedings of the 37th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, Nov. 2020, S. 2048–2056.

[2] Daniel Zimmermann u. a. „NaturalNets: Simplified Biological Neural Networks for Learning Complex Tasks“. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Sep. 2021, S. 4896–4902. DOI: 10.1109/IROS51168.2021.9636471.

A. Anhang

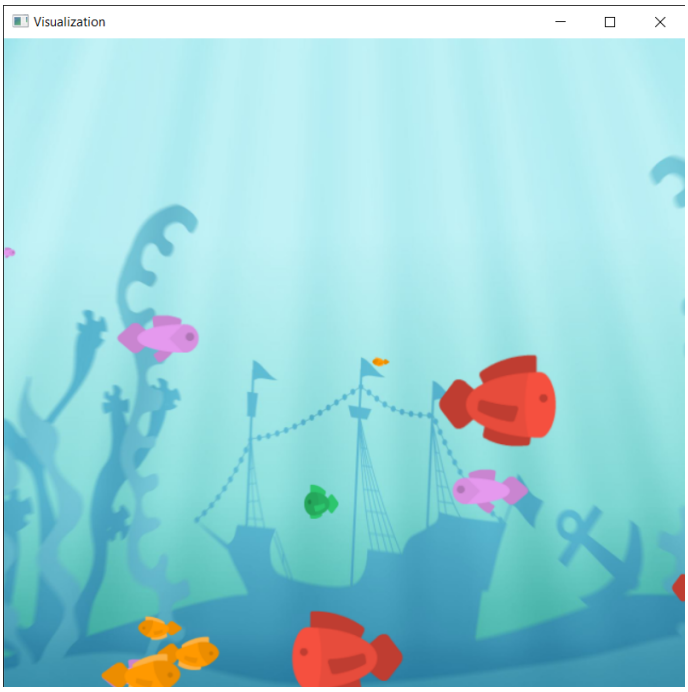

Abbildung 3: Ein Screenshot vom in Julia implementierten Spiel BigFish

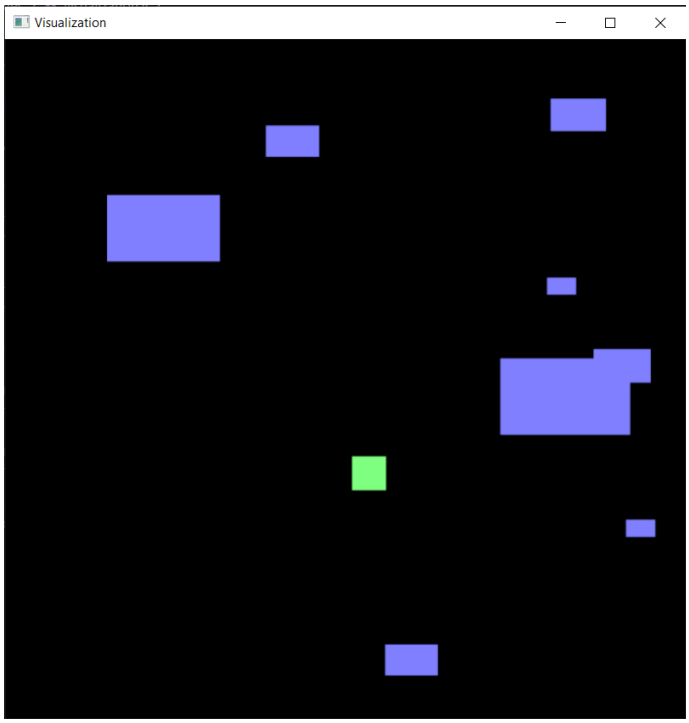

Abbildung 4: Ein Screenshot vom in Julia implementierten Spiel BigFish mit aktivierten Optionen `restrict_themes = true`, `use_monochrome_assets = true` und `use_backgrounds = false`.