Projektgruppe:

Entwicklung eines 3D RPG Videospiels mittels prozeduraler Inhaltsgenerierung und Deep Reinforcement Learning

Jan Beier Nils Dunker Leonard Fricke Niklas Haldorn
Kay Heider Jona Lukas Heinrichs Mathieu Herkersdorf
Carsten Kellner Markus Mügge Thomas Rysch
Jannik Stadtler Tom Voellmer

26. September 2022

„Projektgruppe: Entwicklung eines 3D RPG Videospiels mittels prozeduraler Inhaltsgenerieung und Deep Reinforcement Learning“

SS 2022 - WS 2022/2023 · TU Dortmund

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung

Die prozedurale Generierung von NPCs sowie das Erlernen von Bewegungen und das Animieren dieser stellt heutzutage in der Videospielindustrie eine zentrale Herausforderung dar. Nicht nur müssen Kreaturen effizient und somit oftmals zur Laufzeit generiert werden, sondern es müssen auch die Animationen welche durch maschinelles Lernen erlernt werden, auf ein möglichst breites Spektrum an Kreaturen anwendbar sein. Dabei ist es also essentiell, dass das Training der Kreaturen möglichst generalisiert stattfindet, sodass bei der Kreatur-Generierung eine große, sich bei den Körpermerkmalen variierende Menge der Kreaturen erzeugt werden kann, wodurch weniger Einschränkungen für die Körpereigenschaften der NPCs aufkommen. Somit sollten also die Animationen auf einen möglichst breiten Pool von Kreaturen anwendbar sein, ohne für zufällig erzeugte, potentiell neue, Körpermerkmale neu trainieren zu müssen.

Weiterhin muss bei der Generierung von Spielfiguren die relevante Herausforderung des automatischen Aufspannens eines Meshes über den bis zu diesem Zeitpunkt untexturierten Körper einer Kreatur gelöst werden; das sogenannte *Automatic Rigging*, welches ebenfalls zu dem prozeduralen Erzeugen von Kreaturen dazugehört. Es muss also hier das relevante Problem betrachtet werden, Meshes welche ggf. ebenfalls prozedural erzeugt werden, auf das breite Spektrum von verschiedenen Körperausprägungen des Kreaturen-Pools anwenden zu können, ohne dass es zu sehr von den Körperformen der Kreaturen abweicht text, da sonst die Dreiecksnetze degenerieren und nachfolgende numerische Operationen fehlschlagen würden.? Oder bereits zu technisch?

Ferner könnte die prozedurale Generierung von Inhalten in einem Videospiel nicht nur zum Erzeugen von Kreaturen zum Einsatz kommen, sondern auch für die Spielwelt selbst. Damit könnte ... Notiz für Thomas: Sobald Inhalte von World-Generation stehen, hier ein wenig ausführen.

... mesh muss zu creature passen... lernen der bewegung anpassbar bezüglich creature-pool

1.2 Zielsetzung und Vorgehensweise

Im Rahmen der Projektgruppe 649 nehmen insgesamt 12 Teilnehmer an der Planung und Entwicklung des prozedural generierten 3D Rollenspiels teil.

Für die Umsetzung des Projektes ist es vorgesehen die verschiedenen Bereiche des Rollenspiels aufzuteilen, sodass potentiell in Kleingruppen parallel an diesen gearbeitet werden kann. Diese Bereiche umfassen folgende Themengebiete: die Generierung von

Spielleveln, die Evaluation der generierten Level, Generierung der Monster, Fortbewegung der Monster und zuletzt die Strategie bzw. Verhaltensweise der Monster. Innerhalb jeder dieser Bereiche sollen bestimmte Anforderungen, welche an die Gruppe vorgegeben sind, erfüllt sein:

Die Generierung von Spielleveln

- Die Generierung der Spiellevel sollte prozedural durchgeführt werden: z.B. KD-Trees, L-Systeme, Space-Partitioning
- Die Spiellevel enthalten Böden, Wände und auch Spawn-Points für Objekte und NPCs
- Gleichzeitig sollen die Komplexität, Größe und der Schwierigkeitsgrad der Level parametrisierbar sein

Evaluation der Generierten Level

- Die Level/Spielwelten sollen anhand vorbestimmter Metriken evaluiert werden, wie z.B.: Lösbarkeit der Level, Schwierigkeitsgrad
- Folgende Ansätze sind dafür vorgeschlagen: Imitation Learning, Deep Reinforcement Learning

Generierung der Monster

- In diesem Bereich sind viele Freiheiten gelassen worden, da die Generierung der Monster auf viele unterschiedliche Weisen durchgeführt werden kann
- Relevant dabei ist nur, dass die Erstellung von NPCs algorithmus-basiert ist und dass innerhalb von Unity die von Unity bereitgestellten Joints verwendet werden sollten
- Eine Orientierungshilfe dabei kann der Unity-MLAgents-Walker sein

Fortbewegung der Monster

- Animationen sollen nicht händisch erstellt werden
- Mit Hilfe von Deep Reinforcement Learning können die Monster lernen sich zu bewegen
- Der Agent wählt die Kräfte aus, welche auf seine Joints ausgeübt werden
- Je nach lösen einer spezifischen Aufgabe wird der Agent dann belohnt
- Dabei könnten Inverse Kinematiken nützlich sein

Strategie bzw. Verhaltensweise der Monster

- Klassische Ansätze (high level) wären hier die Behavior Trees oder auch State machines

- Währenddessen lernende Ansätze (low level) wären Imitation Learning und Deep Reinforcement learning

Um ein grundlegendes Verständnis aller Teilnehmenden für alle Bereiche zu schaffen, werden im ersten Monat in einer Seminarphase alle Bereiche auf Kleingruppen, bestehend aus 3 Teilnehmern, aufgeteilt und potentielle Ansätze für die Umsetzung der Gebiete für ein Videospiel erforscht. Während dieser Seminarphase werden dann nacheinander die Themen der Gruppen den restlichen Teilnehmenden vorgestellt. Somit ist eine Verständnisgrundlage für alle Beteiligten geschaffen, sodass basierend auf dem allgemeinen Kenntnisstand mit der eigentlichen Entwicklung des Spieles angefangen werden kann.

Ein besonderer Schwerpunkt soll dabei auf die Themen der **Generierung der Monster** und der **Fortbewegung der Monster** gelegt werden, da diese die Basis für die restlichen Aufgaben bilden sollten. Es wurde sich absichtlich dafür entschieden von Anfang an kein klares Design der späteren Spielwelt, Monster und des Spielercharakters zu definieren, sodass sich daraus keine Einschränkungen für die initiale Implementierungsphase ergeben sollten. Vielmehr sollte aus den Ergebnissen der initialen Phase das spätere Spieldesign abgeleitet werden. Damit also dieser Grundstein gelegt werden konnte, wurde sich dafür entschieden die **Generierung der Monster** und die **Fortbewegung der Monster** der Spielwelt als erstes zu priorisieren und somit die 12 Teilnehmer der Projektgruppe für die Anfangsphase in zwei Untergruppen aufzuteilen: die **Creature-Generator**, bestehend aus 7 Mitgliedern, und die **Creature-Animator**, bestehend aus 5 Mitgliedern. Es wird antizipiert, dass sich mit fortschreitender Zeit die Gruppen, sobald die Basis für das Spiel besteht, in weitere (kleinere) Gruppen aufteilen werden. Das Ziel ist somit, parallel an mehreren Themen gleichzeitig arbeiten zu können und somit effizient Fortschritt zu machen. Währenddessen muss eine deutliche Kommunikation zwischen den (Unter-)Gruppen bestehen um die Themen miteinander abstimmen zu können, vor allem für die Anfangsphase zwischen den **Creature-Generator** und den **Creature-Animator**.

Für diese Anfangsphase wurde evaluiert, die Kreatur-Generierung so einfach wie möglich zu halten um das Beibringen von Bewegungen durch die **Creature-Animator** Gruppe so einfach wie möglich zu gestalten und die Freiheitsgrade auf einem Minimum zu halten. Dafür ist das Vorhaben zweibeinige, humanoide Kreaturen und auch vierbeinige Kreaturen erzeugen zu können. Zusätzliche Körperteile wie beispielsweise Flügel, mehrere Arme oder Beine und Ähnliche Extras werden weggelassen, damit also keine zusätzlichen Freiheitsgrade hinzukommen und das Lernen der Bewegung der Kreatur gegebenenfalls erschweren würden. Trotzdem soll später evaluiert werden, ob solche ergänzenden Körperteile hinzugefügt werden könnten, sobald die Basisstruktur, also das stabile Erzeugen und Lernen der Bewegungen der Kreaturen, besteht. Außerdem wird durch diesen Ansatz der Spielentwurf so schlicht wie möglich gehalten, sodass keine potentiellen Einschränkungen bezüglich des Spieldesigns existieren und damit die spätere Gestaltung des Spiels basierend auf den dann bestehenden Funktionalitäten entschieden werden kann.

1.3 Übersicht

Zunächst werden in dem zweiten Kapitel (2) alle Grundlagen beschrieben in Bezug auf die angeführten Themenbereiche der Projektgruppe welche in der Seminarphase gegenseitig vorgestellt wurden (1.2). Basierend auf diesen Themengebieten werden dann in Kapitel 3 verwandte Arbeiten und Literatur vorgestellt, welche sowohl während der Evaluation in der Seminarphase erforscht wurden, als auch in der späteren Erarbeitung weiterer Themen und Algorithmen (Kapitel 4 und 5) ausgemacht wurden. Sobald die Grundlagen über die Themen der späteren fachlichen und technischen Vorgehensweise geklärt sind, wird in Kapitel 4 das fachliche Vorgehen beschrieben, welches sich während der Entwicklungsphase der verschiedenen Bereiche (s. Kapitel 1.2) erschlossen hat. Nachdem die fachbezogene Vorgehensweise bekannt ist, wird im Anschluss die Umsetzung in Kapitel 5 evaluiert, wie die in Kapitel 4 beschriebenen Verfahren und Algorithmen technisch umgesetzt sind. Anschließend werden in Kapitel 6 die (Zwischen-)Ergebnisse präsentiert und diskutiert (6.1), hier später für den Endbericht ausführen wie genau und wodrauf bezogen die Ergebnisse diskutiert werden. Schließlich wird in Kapitel 7 zusammengefasst, auf welche (Forschungs-)Bereiche die Erkenntnisse dieser Studie ausgeweitet werden können. Weiter ausführen?

2 Grundlagen

(Die benutzten) Vortragsthemen vom Anfang hier als eigene Unterkapitel beschreiben.

3 Verwandte Arbeiten

Beschreiben warum andere Quellen nicht ausreichend waren und weshalb der eigene Ansatz jene fehlende Themen ergänzt.

4 Fachliches Vorgehen

Keine technischen Details (wie z.B. Implementierung)

4.1 Projektorganisation

? Während der Projektgruppe haben sich ... Untergruppen/Teams für die jeweiligen Bereiche ... herausgestellt.

Für die Projektorganisation haben sich ein Projektleiter und jeweils 2 Projektmanager aus den Gruppen der **Creature-Generator** und **Creature-Animator** einberufen. Wöchentlich findet ein Jour-Fixe statt um gemeinsam über die aktuellen Entwicklungen zu sprechen und Herausforderungen gemeinsam anzugehen und zu lösen. Der Projektleiter sollte darin eine Moderator- und Organisationsrolle einnehmen, die Gruppen und Themen zusammen- und im Überblick behalten und somit potentielle Konflikte frühzeitig erkennen und auflösen. Die Projektmanager übernehmen eine analoge Aufgabe im Rahmen ihrer jeweiligen Gruppen und bilden somit gemeinsam mit dem Projektleiter eine Struktur in der die Übersicht über aktuelle Themen einfach beibehalten werden kann. Im Anschluss an das Jour-Fixe treffen sich die vier Projektmanager und der Projektleiter, bereiten für den nächsten Termin des Jour-Fixe die zu besprechenden Themen vor und klären gegebenenfalls organisatorische Einzelheiten um somit die restliche Gruppe der Teilnehmer zu entlasten. Während der Jour-Fixe wird von jedem der Teilnehmer abwechselnd eine Dokumentation des Treffens manifestiert um somit Ergebnisse aber auch ToDo's festhalten und einsehen zu können. Für das wöchentliche Treffen wurde ein Tagesprotokoll (Fig.) genutzt, welches von dem Projektleiter durch Bildschirmübertragung an die Teilnehmer präsentiert und durchgesprochen wurde. Das Tagesprotokoll konnte jederzeit durch jeden Teilnehmer modifiziert und ergänzt werden, falls zu besprechende Themen durch den jeweiligen Teilnehmer aufgekommen sind.

Für die weitere Organisation (der Entwicklung und Implementierung) der Projektgruppe wurden folgende Tools und Hilfsmittel eingesetzt.

Discord wurde als Basis-Kommunikationsmittel genutzt. Hier wurden für die entsprechenden Phasen und Themenbereiche, sowie Gruppen eigene Text- und Sprach-Kanäle erstellt (Fig. (?)). Hier wurden auch aktuelle (organisatorische-) Themen aufgegriffen und besprochen. Der wöchentliche Jour-Fixe Termin wurde ebenfalls in Discord abgehalten. Weiterhin wurde GitHub als Entwicklungs-, Repository- und Speicherplattform genutzt. Sowohl in Discord als auch im GitHub-Wiki wurde die wöchentliche Dokumentation des Jour-Fixe abgelegt um somit zwei Alternativen zu haben auf jene Dokumentation zugreifen zu können. Somit wurde implizit auch durch die Verfügbarkeit auf zwei Plattformen ein Backup der Dokumentationen realisiert, falls im worst-case eine der beiden Plattformen ausfallen sollte. Weiterhin wurden in GitHub pro Gruppe bzw. Themenbereich ein

Repository erstellt mit der späteren Idee, sobald die einzelnen Gebiete fertiggestellt sind, mit Hilfe von exportierten Paketen in einem Main-Repository die verschiedenen Teile des Spiels zusammenzubringen → in anderes Kapitel auslagern? Um insbesondere in der vorlesungsfreien Zeit eine reibungslose Organisation und Ressourcenallozierung gewährleisten zu können, wurde Google-Kalender genutzt, indem sich alle Teilnehmenden in einem Kalender gesammelt haben und ihre jeweiligen Universitäts- und Urlaubsbezogenen Termine eingetragen haben. Während der ersten Monate hat sich darüber hinaus ergeben, dass für eine wirksamere Übersicht eine Darstellung erstellt werden sollte, welche ein Status-Quo abbilden sollte; also bereits fertiggestellte, momentan laufende und potentiell in Zukunft zu behandelnde Projekte und Themengebiete umfassen sollte. Dafür wurde mit Hilfe des Tools Graphviz ein simpler Graphen erzeugt der zudem am Anfang jedes Monats durch die Management-Runde aktuell gehalten wurde (Fig.). Als letztes wurde eine Projekt-Timeline evaluiert, welche für das jeweilige Semester die kurz- und langfristigen Aufgaben der entsprechenden Gruppen in tabellarischer Form festhalten sollte. Da dieses Hilfsmittel jedoch in dem ersten Semester keine Verwendung fand, wurde es zu Anfang der zweiten Hälfte der Projektgruppe als obsolet eingestuft und verworfen.

4.1.1 Creature Generator

In der ersten Phase hat sich die Creature-Generation Gruppe mit der Evaluation und Findung eines geeigneten Generator-Algorithmus beschäftigt. Zunächst wurden dafür in zwei temporären Untergruppen zwei verschiedene Ansätze erforscht:

Skelett → Skin Bei dem ersten Ansatz wurde mit einem L-System Parser experimentiert, zuerst Koordinaten zu erzeugen und das Skelett der Kreatur dann dort reinzulegen (s. Kapitel). Der Skin der Kreatur sollte dabei über Metaballs (Kapitel) und Marching Cubes (Kapitel) zusammengestellt werden.

Skin → Skelett Bei dieser Alternative sollte mit Hilfe von Metaballs und Marching Cubes zuerst ein Skin erzeugt werden, in welches dann ein Skelett durch Automatic Rigging reingelegt werden und durch Dual Quaternion Skinning animierbar gemacht werden sollte.

Während der Ausarbeitung hat sich die Creature-Generator Gruppe wöchentlich am Mittwoch getroffen und hat analog zum wöchentlichen Jour-Fixe aller Teilnehmer einen Regeltermin zum Besprechen von abgeschlossenen aber auch ausstehenden Aufgaben abgehalten. Eine Dokumentation davon wurde ebenfalls analog zum Jour-Fixe sowohl auf Discord als auch im GitHub-Wiki des Creature-Generation Repositories (Link) abgelegt.

Am Schluss der Evaluation beider Ansätze wurde sich für die erste Alternative entschieden: es sollte das L-System zum Erzeugen der initialen Koordinaten genutzt werden, um daraus dann das Skelett zu erstellen und anschließend mit Hilfe der Metaballs den Skin über das Skelett zu legen. Der Dual Quaternion Ansatz (Kapitel) wurde fortwirkend von Leonard Fricke weiterverfolgt, hatte jedoch zu diesem Zeitpunkt noch keine große

Priorität, da das erste Ziel eine funktionierende Skelett Generierung zu erhalten war, um sich danach dem Skin-Mesh widmen zu können.

Während der weiteren Ausarbeitung hat sich jedoch ein Alternativansatz zum L-System von Jona Heinrichs gezeigt (Kapitel), um effizientere Skelette erzeugen zu können. Damit wurde die Entwicklung des L-Systems an dieser Stelle eingestellt. Somit wurde evaluiert, dass beide Autoren des L-Systems, Tom Voellmer und Kay Heider, von der Creature-Generation Gruppe in eine weitere Gruppe der Terrain-Generator abzuzweigen, da sich beide Teilnehmer während der Seminarphase mit der Terrain-Generation beschäftigt haben und somit das nächste Thema angegangen werden konnte.

Inzwischen wurden an die Creature-Animator Gruppe bereits erste Skelette als Blueprints bereitgestellt, sodass diese bereits ihr Training auf den bis zu diesem Zeitpunkt erzeugten Kreaturen prüfen konnten. Dabei hat sich Markus Mügge als Vermittler zwischen den Creature-Generatorn und Creature-Animatorn bereiterklärt und war somit für die Kommunikation und den Wissensaustausch beider Teams außerhalb der Jour-Fixe zuständig. Bei dieser Kommunikation beider Teams haben sich etliche Verbesserungen welche von den Creature-Animatorn angeführt wurden von den Creature-Generatorn umgesetzt.

Dabei hat Markus Mügge die finale und relevante Innovation eines Interface den Creature-Animatorn zur Verfügung gestellt, sodass Letztere nicht mehr von einzelnen Blueprints bzw. Paketen mit Kreaturen abhängig waren, welche von den Creature-Generatorn übermittelt werden mussten, sondern nun eigene Kreaturen on-demand erzeugen und ihr Training der Bewegung der Kreaturen untersuchen konnten.

4.2 Creature Generation

4.3 Creature Animation

4.3.1 Trainingsarena

ML-Agent Walker -> zu Statisch -> Dynamischer gestalten

Konfiguration

Statisch Problematisch

4.3.2 Training

- PPO
- ML-Agents
- Nero?

4.4 Terraingeneration

5 Technische Umsetzung

Bei der Erläuterung der Wahl der Hierarchie für Knochen nur deskriptiv darauf eingehen; keine Details oder Begründung erforderlich. **Dies übernimmt die Creature-Generator Gruppe.** Eingehen auf Status Quo.

5.1 Creature Animation

5.1.1 Trainingsumgebung

Im Folgenden soll der Aufbau der Trainingsumgebung beschrieben werden, welche es erlaubt verschiedenste Kreaturen ohne große Anpassungen zu trainieren. Die Umgebung ist dabei aus den folgenden Klassen aufgebaut:

- `DynamicEnviormentGenerator`
  - `TerrainGenerator`
  - Verschiedenen Konfigurationsdateien
  - `DebugScript`
- allen anderen modifizierten ML-Agents Skripten

In diesen Abschnitt wird nur auf den Aufbaue des `DynamicEnviormentGenerator` sowie dessen Hilfsklassen und nicht auf die ML-Agent-Skripte eingegangen. Die Hilfsklassen sind der `TerrainGenerator`, `GenericConfig` und dessen Implementierungen sowie das `DebugScript`. Erstere ist verantwortlich für die Generierung des Terrains, die Config-Dateien laden dynamisch die Einstellungen aus einer Datei und das letzte Skript beinhaltet hilfreiche Debug-Einstellungen. Die grundsätzliche Idee der Trainingsumgebung stammt von dem ML-Agents-Walker. Da an diesem keine Versuche mit Unterschiedlichen Umgebungen und Kreaturen durchgeführt wurden, ist der Aufbau des Projekts nicht dynamisch genug.

Dynamic Enviorment Generator

Zur dynamischen Umsetzung der Trainingsarena werden alle Objekte zur Laufzeit erstellt. Die Generierung der Arena läuft dann wie folgt ab:

1. Erstellen von n Arenen, wobei n eine zu setzende Variable ist.
2. Füge ein Ziel für die Kreatur in die Arena ein

3. Generiere die Kreatur

Die einzelnen (Teil)-Arenen bestehen aus einem Container-Objekt unter dem ein Terrain und vier Wall-Prefabs angeordnet sind. Diese Prefabs und weitere Elemente wie Texturen werden dynamisch aus einem Ressourcen-Ordner geladen, damit möglichst wenige zusätzliche Konfigurationen den Editor verkomplizieren. Das Terrain wird mit leeren Terraindaten vorinitialisiert und später befüllt. Hierbei kann die Position des Container-Objects in der Szenen wie folgt berechnet werden:

$$\begin{pmatrix} \lceil \frac{\text{Anzahl der Arenen}}{\sqrt{\text{Anzahl der Arenen}}} \rceil \\ 0 \\ \text{Anzahl der Arenen} \mod \sqrt{\text{Anzahl der Arenen}} \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{5.1}$$

Alle anderen Objektpositionen müssen danach neu im lokalen Koordinatensystem gesetzt werden. Da die Unity-Standard-Texturen sehr hell sind sind, werden die Texturen bei der Initialisierung mit ML-Agents-Texturen, welche dunkler sind, getauscht. An das Terrain werden zuletzt Collider und ein `TerrainGenerator`-Skript angefügt.

In Schritt 2. der Arenagenerierung muss beachtet werden, dass nach dem Erstellen des Zielobjekts das `WalkTargetScript` hinzugefügt wird. Am Ende des Erstellungsprozesses wird der Walker erstellt. Hierzu wird ein von den Creature-Generator-Team bereitgestelltes Paket[1] benutzt. Das Paket stellt ein Klasse bereit, welche mit zwei Skript-Objekte konfiguriert wird. Zusätzlich wird ein seed übergeben, welcher reproduzierbare Kreaturen erlaubt. Die erstelle Kreatur muss danach mit den entsprechenden ML-Agent-Skripten versehen werden. Hierzu wird ein `WalkerAgent` Objekt als String übergeben. Dies ermöglicht es, mehrere unterschiedliche Agent-Skripte durch eine Änderung im Editor zu setzen. Somit können Reward-Funktion und Observation für zwei unterschiedliche Trainingsversuche getrennt, in eigenen Dateien, entwickelt werden.

TerrainGenerator

Da ein typisches Spieleterrain im Gegensatz zum ML-Agents-Walker-Terrain nicht flach ist, wurde ein neues Objekt erstellt, welches sowohl die Generierung von Hindernissen, als auch eines unebenen Bodens erlaubt. Um ein möglichst natürlich erscheinendes Terrain zu erzeugen wird ein Perlin-Noise verwendet. Dieses spiegelt jeweils die Höhe des Terrains an einen spezifischen Punkt wider. Im späteren Projektverlauf wurde dieses Skript durch den Terraingenerator des dazugehörigen Teams ersetzt.

Konfigurationsobjekte

Da sich die statische Konfiguration des ML-Agents-Walker als problematisch erwies, wurde die Konfiguration über die Laufzeit des Projekts dynamischer gestaltet. Zuerst wurden alle Konfigurationen im `DynamicEnviormentGenerator` gespeichert. Was unübersichtlich war und zu ständigen neubauen des Projektes führte. Deshalb wurde eine `GenericConfig`

[1] https://github.com/PG649-3D-RPG/Creature-Generation

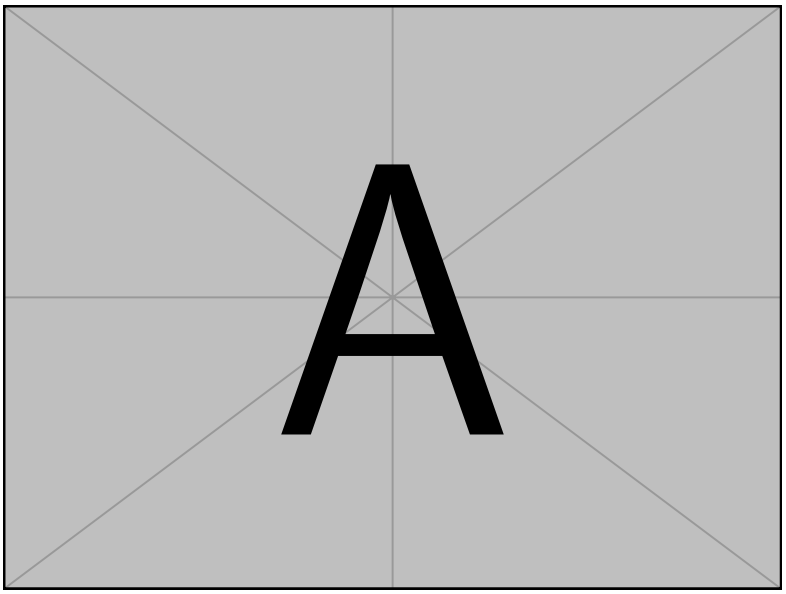

Abbildung 5.1: Konfigurationsmöglichkeiten des `DynamicEnviormentGenerator` im Unity-Editor.

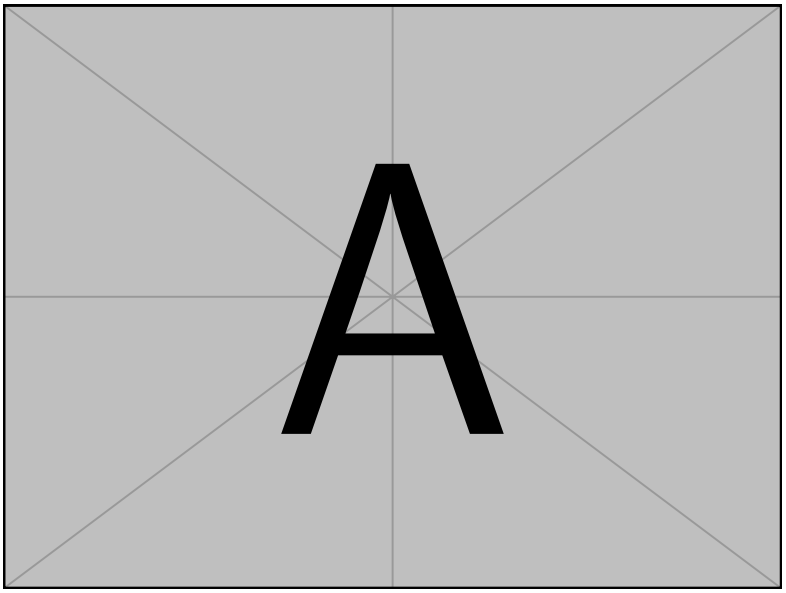

Abbildung 5.2: Ein Beispiel der generierten Trainingsumgebung mit mehreren Arenen.

Klasse eingeführt, welche die im Editor eingestellten Optionen für die einzelnen Teilbereiche Terrain, Arena und ML-Agent in Json-Format in den Streaming-Asset-Ordner speichert. Da dieser Ordner beim bauen des Projekts in das fertige Spiel übertragen wird, sind diese Konfigurationen automatisiert dort vorhanden.

Im Fall, dass das Spiel ohne Editor gestartet wird, was meist beim Training der Fall ist, lädt das generische Objekt aus den Json-Dateien die Einstellungen und ersetzt die Editorkonfiguration damit. Hierdurch ist ein ändern der Konfiguration des Spiels ohne neu-erstellen der Binärdateien ermöglicht. Diese Konfigurationsart fügt Abhängigkeiten zu dem Unity eigenen JsonUtility[2] hinzu.

[2]https://docs.unity3d.com/ScriptReference/JsonUtility.html

6 Vorläufige Ergebnisse

Unterkapitel nach Erkenntnissen. Metrik nach der bewertet wird erörtern. Objektiv ohne Wertung der Ergebnisse.

6.1 Diskussion

Diskussion der Ergebnisse in Bezug auf die initiale Zielsetzung.

7 Ausblick

Themenallokation

1. Einleitung: Thomas
2. Grundlagen
   - 2.1. Reinforcement Learning: Jannik
   - 2.2. Cellular Automata: Markus
   - 2.3. L-Systeme: Kay
3. Verwandte Arbeiten
   - 3.1. RL Frameworks
     - 3.1.1. ML-Agents: Niklas
     - 3.1.2. NeroRL: Niklas
4. Fachliches Vorgehen:
   - 4.1. Projektorganisation: Carsten, Thomas, Jannik, Leonard
   - 4.2. Creature Generation
     - 4.2.1. Metaballs: Jona
     - 4.2.2. Jonas Creature Generation Method: Jona, Markus
     - 4.2.3. L-System Creature Generation: Tom, Kay
   - 4.3. Creature Animation
     - 4.3.1. Trainingsumgebung: Carsten
     - 4.3.2. Training: Als Gruppe überlegen
   - 4.4. Terraingeneration
     - 4.4.1. Dungeon Generierung: Tom
     - 4.4.2. Space Partitioning: Tom
     - 4.4.3. Perlin Noise (Terrain-Flächen): Kay
     - 4.4.4. Vegetation via L-System: Kay
5. Technische Umsetzung
   - 5.1. Creature Animation: Nils, Carsten, Jan
     - 5.1.1. Trainingsumgebung: Nils
     - 5.1.2. Erweiterung der Agent Klasse

Abbildungsverzeichnis

Tabellenverzeichnis