Projekgruppe:

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

21. September 2022

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

SS 2022 - WS 2022/2023 · TU Dortmund

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung

12 Teilnehmer...

1.2 Zielsetzung und Vorgehensweise

Für die Umsetzung des prozedural generierten 3D Rollenspieles wurde motiviert die verschiedenen Bereiche des Rollenspiels aufzuteilen, sodass in Kleingruppen parallel an diesen gearbeitet werden konnte. Diese Bereiche sollen folgende Themengebiete umfassen: 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 Vorgaben bzw. 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

Ein besonderer Schwerpunkt soll dabei auf die Gruppen 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 Design 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 in zwei Untergruppen aufzuteilen: die **Creature-Generator** und die **Creature-Animator**.

Hier weiter bei Seminar- und Praktikumsphase bzw. mit der Einteilung der Gruppen.... (oder direkt in Übersicht überleitend?)

1.3 Übersicht

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

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

Wie sind wir vorgegangen... auf alles bezogen. Wer hat an welchen Kapiteln mitgearbeitet.

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

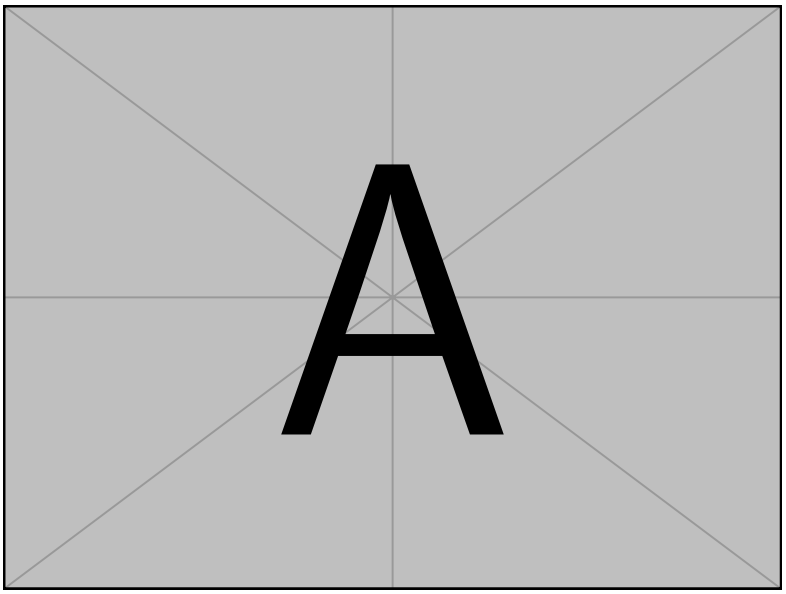

Abbildung 5.1: Konfigurationsmöglichkeiten des `DynamicEnviormentGenerator` im Unity-Editor.

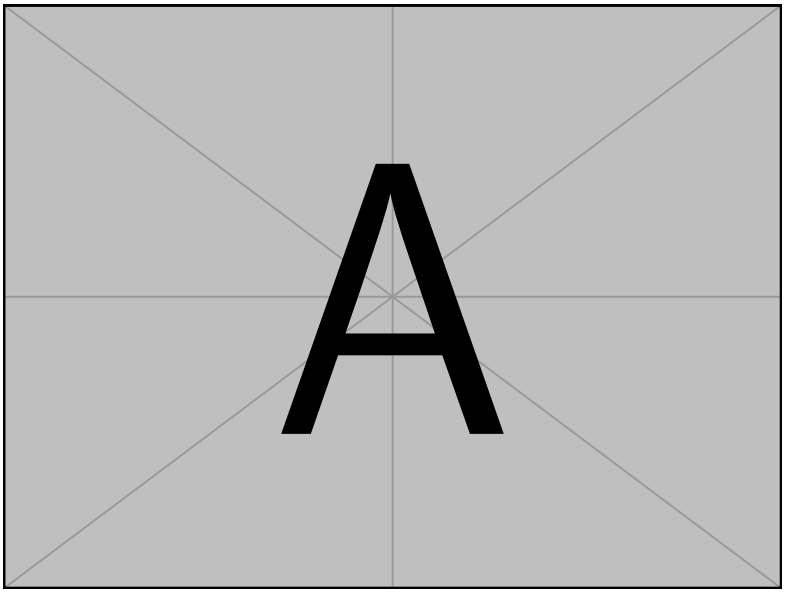

Abbildung 5.2: Ein Beispiel der generierten Trainingsumgebung mit mehreren Arenen.

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