1 Einleitung

1.1 Motivation und Problemstellung

1.2 Verwandte Arbeiten

Falls zu groSS/zu viel, auslagern in ein eigenes Kapitel nach Grundlagen. Beschreiben warum andere Quellen nicht ausreichend waren und weshalb der eigene Ansatz jene fehlende Themen ergänzt.

1.3 Zielsetzung und Vorgehensweise

1.4 Übersicht

2 Grundlagen

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

3 Fachliches Vorgehen

Keine technischen Details (wie z.B. Implementierung)

3.1 Projektorganisation

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

3.1.1 Creature Animator

- Zwei Gruppen Animator/Nero RL
  - Zuerst nach Skills
  - Einzelarbeit an verschiedenen Projekten
    - Jan Training
    - Carsten Parameter finden, NavMeshes,
    - Arena Generierung, Config Zeugs
- On Demand treffen
- Protokolle, nur wenn wichtig, sonst git

3.2 Creature Generation

3.3 Creature Animation

3.3.1 Trainingsumgebung

ML-Agent Walker -> Eigene Umgebung, gleicher Walker ->

- zu Statisch Umgebung -> Dynamischer gestalten/ Mehr feature
- Kreatur austauschen -> L-System ging nicht -> Jona/Markus Methode nutzen

LiDo

Konfiguration

Da die Trainingungebung auf den ML-Agent-Walker basiert, waren viele Konfiguration fest-codiert. Zuerst wurden diese über den Unity-Inspektor änderbar gemacht. Als mit dem aktiven Training auf LIDO begonnen wurde, stellte sich diese Methode als nicht flexible genug heraus. Die Trainingsumgebung musste für jede Änderung neu gebaut werden. Deshalb wurde ein neues System geschrieben, welches über Dateien die Konfiguration dynamisch lädt.

3.3.2 Training

Generalisierung

- PPO
- ML-Agents
- Nero?

3.4 Terraingeneration

4 Technische Umsetzung

Eingehen auf Status Quo.

4.1 Creature Animation

4.1.1 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:

$$\begin{pmatrix} \lceil \frac{\text{Anzahl der Arenen}}{\sqrt{\text{Anzahl der Arenen}}} \rceil \\ 0 \\ \text{Anzahl der Arenen} \mod \sqrt{\text{Anzahl der Arenen}} \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4.1}$$

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` Klasse eingeführt, welche die im Editor eingestellten Optionen für die

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

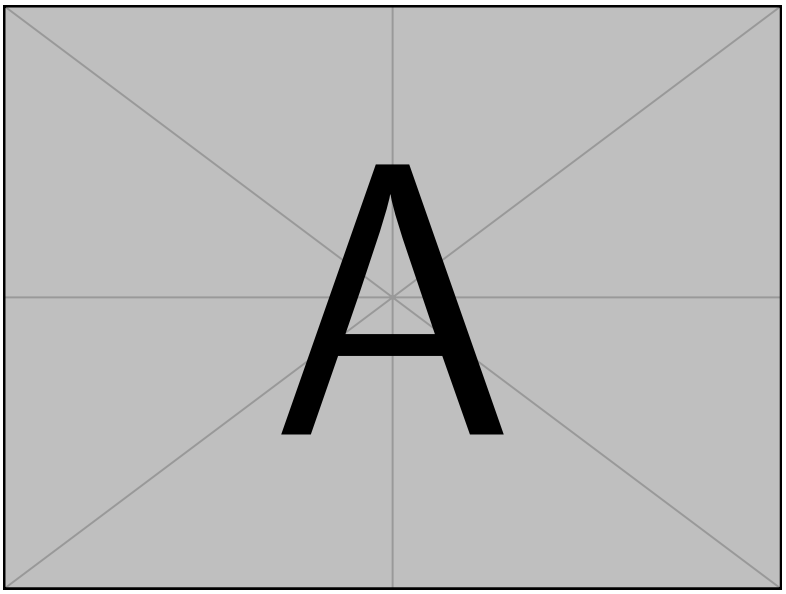

Figure 4.1: Konfigurationsmöglichkeiten des `DynamicEnviormentGenerator` im Unity-Editor.

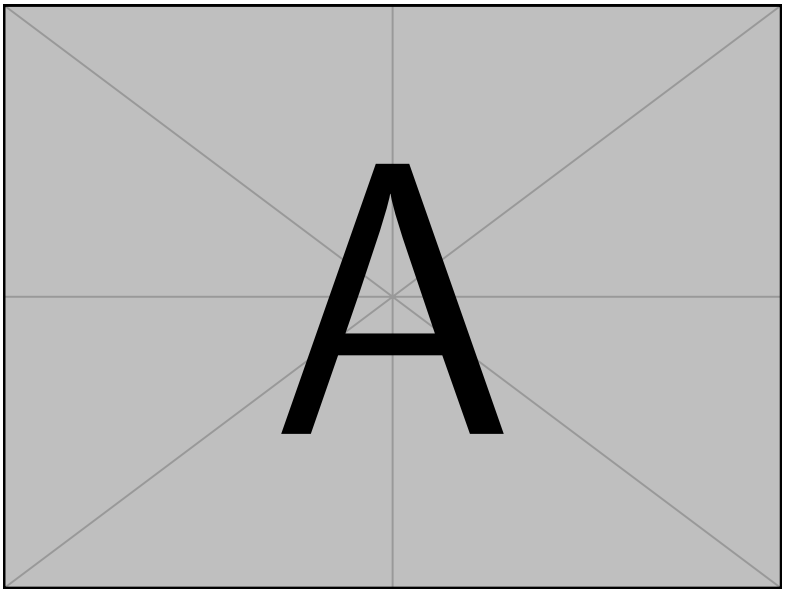

Figure 4.2: Ein Beispiel der generierten Trainingsumgebung mit mehreren Arenen.

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.

4.1.2 WalkerAgent/blabla

Jan mach mal! Irgendwo auch Generalisierung

- UML von den Klassen
- WalkerAgent/JointDriveController/TargetController
- Reward Funktion

4.1.3 LiDo Training

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

5 Vorläufige Ergebnisse

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

5.1 Diskussion

Diskussion der Ergebnisse in Bezug auf die initiale Zielsetzung.

6 Ausblick