Spieleprogrammierung
Übung 2

Tillmann Faust

06.07.2025

Inhaltsverzeichnis

1 Vorüberlegungen

Ziel dieses Projekts ist es eine Chat Modell (z.B. ChatGPT, Deepseek,...) so in ein Spiel zu integrieren, dass NPCs mindestens teilweise durch diese gesteuert werden, das NPC Verhalten also nicht vollständig geskripted ist.

1.1 Mögliche Realisierungen

Basierend auf diesen vorgaben lassen sich verschiedene Realisationspfade erkunden, welche im folgenden kurz erläutert werden sollen:

Bewegungssteuerung. Eine Möglichkeit der KI-Integration ist die Bewegungsteuerung des NPCs durch das Chat Modell. Dem Modell werden Informationen wie z.B. die Spielwelt, die eigene Position in der Welt, Spielerposition, andere NPCs, Hindernisse, etc. übermittelt und die KI kann anschließend entscheiden wohin und auf welche Art das Textmodell sich bewegen möchte.

Entscheidungsfindung. Spiele müssen nicht immer mit Bewegung einhergehen (z.B. Brettspiele). Die Entscheidungsfindung beschäftigt sich damit, das Textmodell basierend auf dem Aktuellen Zustand des Spiels und eventuell zuvor gemachten Zügen über einen nächsten Zug entscheiden zu lassen.

Kommunikation. Zuletzt kann man der KI auch die Möglichkeit geben nicht Bewegung oder Entscheidungen zu steuern, sondern dem Spieler generierte Antworten basierend auf Aussagen (sei es Text oder Speech to Text) und Handlungen zurückzugeben. Diese können anschließend in einen Chat geschrieben werden oder über Text to Speech vertont werden.

Theoretisch lassen sich all diese Ansätze auf beliebige Weise kombinieren um im extremsten Fall einen nahezu ”autonomen” NPC zu erhalten. Für die Projektabgabe wurde sich dazu entschieden den ersten Pfad, also die Bewegungssteuerung, weiter zu verfolgen.

1.2 Informationsübermittlung

Damit die KI die Möglichkeit hat die Handlungen, Entscheidungen oder Aussagen des NPCs zu steuern benötigt die sie für die Aufgabenbewältigung relevante Informationen in einem für die KI verständlichen Format, also als Text. Beim Stellen von Fragen und Aufgaben an Chat Modelle haben sich verschiedene Prompting-Frameworks wie z.B. RFT (Rola Task Format) oder RISEN (Role Instruction Steps Endgoal Narrowing) herauskristallisiert. Grundsätzlich ist ein gut strukturierter Ansatz, welcher dem Modell die gestellte Aufgabe, möglichen Kontext, Einschränkungen, sowie im besten Fall ein paar wenige Beispiele übermittelt, eine produktive Grundlage. Weiterhin eignet sich eine Übermittlung von Informationen im XML Format. Basierend darauf wird in diesem Projekt das Format

```
"""
<Context>...</Context>
<Task>...</Task>
```

```
<Constraints>
  <Constraint>...</Contraint>
  ...
</Constraints>
<Examples>
  <Example>...</Example>
  ...
</Examples>
"""
```

verwendet.

1.3 Mögliche Probleme

2 Umsetzung

2.1 Spielwelt

Da es sich bei diesem Projekt um eine Art Proof of Concept handelt wurde sich für eine simple Spielwelt entschieden. Der Aufbau richtet sich dabei nach einem NxM Raster, welches zuvor in einer Textdatei (vgl. Übung 1) definiert wurde. Das in der Abgabe verwendete Layout ist

```
1000001100
00E0100100
0100001001
0000P00001
0100101111
0000000001
```

wobei 0 ein freies Quadrat, 1 ein belegtes Quadrat, E einen NPC-Spawnpunkt und P den Spielerspawn markiert. Das unter Assets/Map/Scripts befindliche Skript "MapGenerator.cs" nimmt eine solche Text-Datei, generiert eine Plane als Boden, große Quader auf Blockierten Feldern und spawnt NPCs und den Spieler an den entsprechenden Stellen. Zusätzlich werden die Informationen WorldOrigin, WorldCenter, WorldSize und WorldScaleFactor in einem WorldInfo Objekt gespeichert um diese Werte an NPCs weiterzugeben.

2.2 Entities

Als Grundlage für die Bewegungssteuerung von Spieler und NPCs wird eine Klasse EntityStateManager verwendet, welche sich auf die Dateien EntityStateManager.cs und EntityVariables.cs im Ordner Assets/Scripts/Player verteilt. In ihr werden zur Bewegung notwendige Variablen wie verschiedene Geschwindigkeiten, Bewegungs- und Rotationsvektoren, verschiedene States (siehe nächste Untersektion), sowie der Animator definiert und in verschiedenen, für eine bewegliche Entität üblichen, Methoden verwendet.

2.3 Bewegung, States und Animation

Die Umsetzung der Bewegung und Animationen von Spieler und NPCs erfolgt über eine State Machine. Die vordefinierten States sind

- SimpleIdleState
- SimpleWalkState
- SimpleFallState
- SimpleRunIdle
- SimpleRunState
- SimpleCrouchIdleState
- SimpleCrouchWalkState

wobei die Crouch-States zwar existieren und der Spieler auch in diese übergehen kann, sie jedoch keine eigene Animation besitzen und nicht für den NPC erreichbar sind. Der Übergang zwischen den States wird durch die Variablen IsRunning, IsCrouching, die CharacterController Variable IsGrounded, so wie über die Betrag von MoveVector nach folgendem Schema gesteuert werden

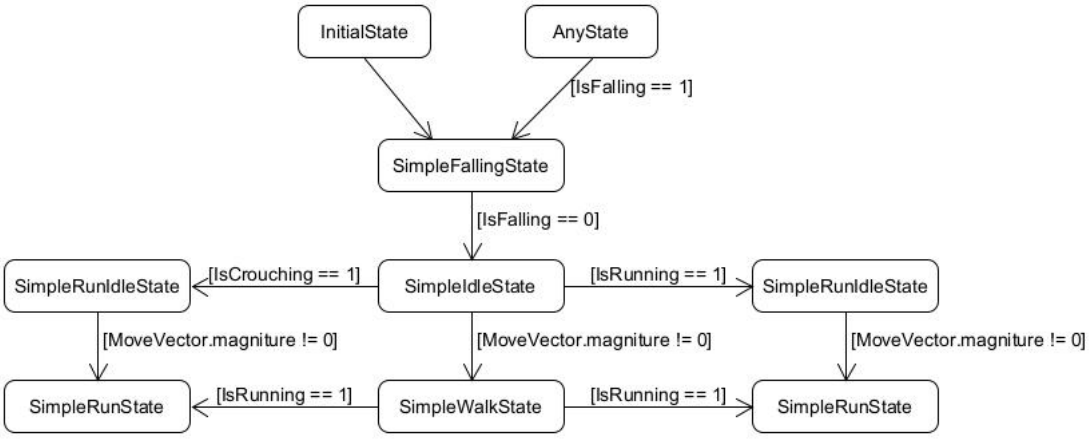

Abbildung 1: Simple State-Machine zum Steuern der Entity-Bewegung

Jeder Übergang, bis auf die von InitalState und AnyState sind symmetrisch, das heißt die Gegenbedingung führt zu einem Übergang in die Andere Richtung. Die Move()-Funktion einer Entität wurde auf die States ausgelagert und wird lediglich in SimpleRun-, SimpleWalk- und SimpleCrouchState aufgerufen. Weiterhin wird beim Eintreten in einen neuen State auch ein Animationswechsel per Trigger ausgelöst.

2.4 NPCs

Das Skript zur Steuerung von NPCs heißt NpcStateManager.cs. Seine Implementation befindet sich unter Assets/Entities/NPC/Scripts. Anders als der Spieler, welcher durch einen User-Input gesteuert wird, muss ein NPC Programma-

tisch bewegt werden. Dazu implementiert die Klasse NpcStateManager ein Interface IControllable (Assets/Entities/Npc/BehaviorController.cs), welches die Vector2 Properties SpawnXZ, CurrentXZ und TargetXZ zum Bestimmen und Setzen einer neuen Zielposition, den Transform Player zum Bestimmen der Spielerposition und die Methoden SetRunning und SetWalking verlangt.

Eine Instanz der abstrakten Klasse BaseBehaviorController (instantiiert über eines seiner Derivate) wird anschließend in einer NpcStateManager-Instanz verwendet, greift auf seine IControllable Variablen zu und setzt TargetXZ, sowie IsRunning oder IsWalking basierend auf dem definierten KI-Verhalten. Zusätzlich erhält ein behavior controller noch die WorldInfo. Im NpcStateManager wird lediglich TargetXZ verwendet und der NPC versucht sich auf direktem Weg zu dieser Zielposition zu bewegen, bis er diese erreicht. Es ist ersichtlich, dass diese Art der Bewegungssteuerung entweder sehr Primitiv ist oder sehr kleinkariert vollzogen werden muss. Entweder der NPC läuft regelmäßig gegen Wände, oder die Variable TargetXZ muss häufig geändert werden.

2.5 KI Integration

Modell

Als Modell wurde Gemma3:4b gewählt, da es unter allen betrachteten Modellen (deepseek-r1:1.5b, deepseek-r1-distilled:7b, gemma3, gemma3:4b) über die schnellste Antwortzeit kombiniert mit der höchsten "Intelligenz" verfügt und sich auf der eigenen Hardware über Ollama mit einer vergleichsweise geringen RAM-Nutzung gut self-hosten lässt.

LLMGridBehaviorController

Die KI Integration erfolgt über das Vermitteln der Spielerposition, der eigenen Position, sowie der Dimensionen des Weltenrasters, welcher anschließend mit einer neuen Koordinate im Raster antworten soll, auf welcher basierend der Wert für TargetXZ getzt wird.
Bei der Umsetzung ist zu beachten, dass die Anfragen an die KI asynchron laufen müssen, da sonst der gesamte Programmablauf blockiert wird. Verwendet werden die Variablen currentlyThinking und lastLLMResponse in einem BusyWaiting Ansatz. In einer Hauptschleife wird der Wert currentlyThinking abgefragt und die Schleife sofort unterbrochen, wenn er auf wahr gesetzt ist. Anschließend wird die neue Zielkoordinate aus lastLLMResponse mithilfe des Regex-Musters "`\((\d+)\s*(\d+)\)`" extrahiert, sofern es in der Antwort enthalten ist. Das Muster erkennt Koordinatenpaare der Form (x,y). Anschließend werden die Grid-Koordinaten zu einer Koordinate in der echten Spielwelt umgerechnet und TargetXZ entsprechend gesetzt. Zuletzt wird eine neue Anfrage per SendLLMQuery() an die Chat-KI gesendet und das Busy-Waiting geht von vorne los.

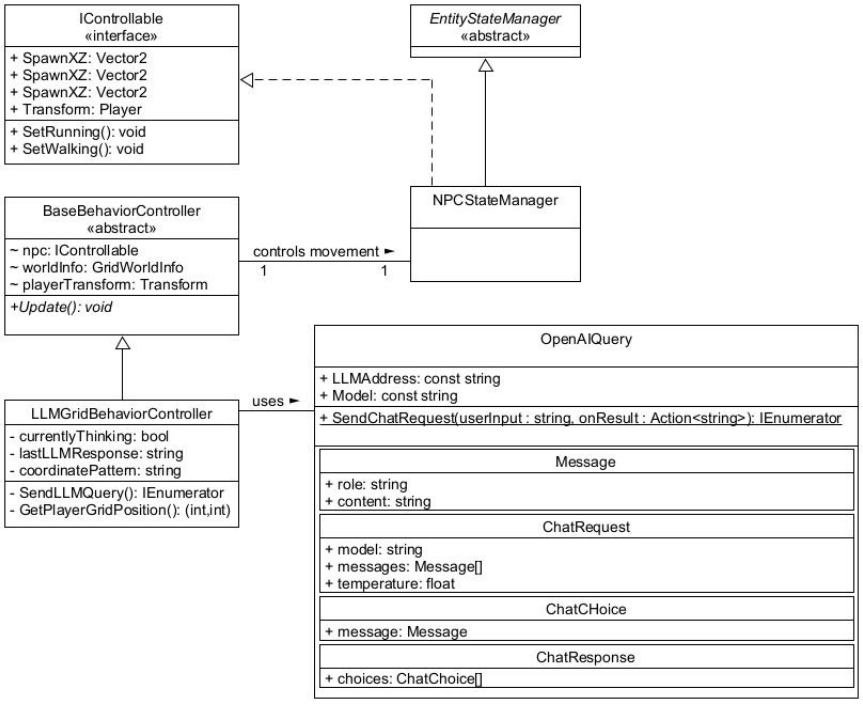

Abbildung 2: Klassendiagramm der LLM Interaktion

Der Anfrage String basiert auf dem in 1.2 präsentierten Schemas und ist:

```
"""
<Context>
  You are an NPC on a {worldSizeString} sized grid.
  Your current position is {currentGridPosString}
  and the Player is at Position {playerGridPosString}
</Context>
<Task>
  Chose a position on the grid where you want to go.
</Task>
<Constraints>
  <Constraint>
    Your answer should only consist of the x,y
   coordinate of your target point.
  </Contraint>
  <Constraint>Stay within the specified grid size</
   Contraint>
  <Constraint>(x,y) may not be larger than {
   worldSizeString}</Constraint>
</Constraints>
<Examples>
  <Example>(0,0)</Example>
  <Example>(1,2)</Example>
```

```
20    <Example>(3,3)</Example>
21  </Examples>
22  """
```

Der originale Anfrage-String enthält keine Zeilenumbrüche. Diese wurden zur besseren Visualisierung eingefügt.

3 Ergebnisse und Fazit

Zusammenfassend lässt sich sagen, dass der gewählte Ansatz eine gute Grundlage für eine Weiterentwicklung bietet, der in diesem Projekt entstehende NPC nicht als sonderlich intelligent bezeichnet werden kann. Die Antwortzeit des LLMs lag in der Regel zwischen 500ms und 2s, was für ein LLM recht schnell ist und das Modell dadurch sogar gerade so für die gewählte Umsetzung eignet. Leider ist das Modell nicht intelligent genug um Komplexere Aufgaben zu lösen, beziehungsweise ist es für diese Aufgabe in Hinsicht auf Intelligenz nicht wirklich geeignet. Die Wahl der Zielkoordinate hängt dem Anschein nach nicht wirklich vom Spieler ab. Auch die Antwortgeschwindigkeit eignet sich zwar für diese Art der Bewegung, ein wirkliches Pathfinding über die KI ist damit jedoch nicht möglich und auch Situationen, die eine bessere Reaktionszeit erfordern sind für den NPC nicht erfüllbar. Dennoch lässt der Proof of Concept darauf schließen, dass bei besseren Antwortgeschwindigkeiten und einem intelligenteren Modell eine Bewegungssteuerung des NPCs durch ein Chat Modell durchaus möglich ist und zu interessantem Verhalten führen kann.