TECHNISCHE UNIVERSITÄT
CHEMNITZ

Aufgabenstellung

-

Erzeugung interaktiver Umgebungen für verkörperte digitale Technologien

---

Fakultät
Informatik

Professur
Graphische Datenverarbeitung und Visualisierung

Aufgabentyp
Teamorientiertes Praktikum

Ausgabe der Aufgabenstellung
15. Mai 2023

Studenten

Leon Rollenhagen
Angelique Gräfe

Lisa Neuhaus
Carlo Kretzschmann

Sophie Neuhaus
Linus Thriemer

Prüfer und Betreuer

Prof. Dr. Guido Brunnett
Technische Universität Chemnitz

Tom Uhlmann M.Sc.
Technische Universität Chemnitz

Titel: *Erzeugung interaktiver Umgebungen für verkörperte digitale Technologien*

Vorbetrachtungen und Zielstellungen:
Im Sonderforschungsbereich Hybrid Societies wird die Interaktion von Menschen und verkörperten digitalen Technologien (EDTs) untersucht. EDTs können autonome Fahrzeuge oder Roboter, Service- oder Info-Roboter, virtuelle Charaktere oder Menschen sein, die mit technischen Erweiterungen wie intelligenten Brillen, Prothesen oder Exoskeletten ausgestattet sind. Obwohl uns im täglichen Leben noch wenige dieser Dinge begegnen, ergeben sich viele technische, logistische und soziale Implikationen, welche sich mit dem Einzug solcher Technologien in unseren Alltag ergeben und bereits jetzt untersucht werden sollten. Diese Implikationen betreffen unter anderem die Wahrnehmung, Kommunikation, Interaktion, Umsetzung und den räumlichen Bedarf der jeweiligen Akteure. Die Erforschung dieser Aspekte ist schwierig, da die entsprechenden Geräte und Roboter entweder noch nicht existieren oder sehr teuer sind. Daher ist es praktisch, eine virtuelle Umgebung zu haben, in der diese Aspekte erforscht werden können.

In diesem teamorientierten Praktikum werden Möglichkeiten erforscht und umgesetzt, wie eine lebendige, interaktive Welt erschaffen werden kann, in der ein Nutzer mit EDTs interagieren kann oder die Interaktion von Agenten untereinander und in der Welt beobachten kann. Um dies zu erreichen, müssen die entsprechenden Objekte wie die Welt, die Agenten und die Charaktere erschaffen werden. Zusätzlich müssen die Objekte interagierbar gemacht werden, und die Agenten müssen sich selbständig in der Welt bewegen und mit ihr interagieren können. Der Fokus sollte dabei auf Glaubwürdigkeit und Plausibilität liegen. Das bedeutet, dass Ansätze aus der wissenschaftlichen Literatur und bereits existierende EDTs verwendet werden sollten. Die virtuelle Welt sollte so angelegt sein, dass neue Agenten in die Welt eingefügt oder das Verhalten bestehender Agenten geändert werden kann, um die Auswirkungen zu untersuchen.

Inhaltsverzeichnis

1 Motivation **1**

1.1 Projektziel Innenstadt 1

1.2 Minimum Viable Product 2

2 Konzept **3**

2.1 Vorbedingungen 3

2.2 Architektur und benötigte Systeme 3

2.3 Entity Component System 4

2.4 Partikelsystem 4

2.5 Multiagentensystem 4

2.5.1 Sensoren 4

2.5.2 Entscheidungsfindung 4

2.5.3 Entscheidungsausführung 5

2.5.4 Bewegungssteuerung 5

2.6 Navigationssystem 6

2.7 Physiksystem 6

2.8 Szenen Editor 7

2.8.1 GLTF als Szenenbeschreibung 7

2.8.2 Eigenes Format zur Szenenbeschreibung 8

2.9 Dialogsystem 8

3 Implementierung **9**

3.1 Entity Component System 9

3.2 Multiagenten System 9

3.2.1 Sate Machines 9
3.2.2 Behaviour Tree 9
3.2.3 Steering Behaviour 9
3.3 Partikelsystem 9
3.4 Navigationssystem 9
3.5 Physiksystem 9
3.6 Szenen Editor 10
3.7 Dialogsystem 10
3.8 Management 10

4 Ergebnisse **11**

5 Zusammenfassung & Ausblick **13**

A Nachsätze **15**
A.1 Arbeitsaufteilung 15

Kapitel 1

Motivation

1.1 Projektziel Innenstadt

Gemeinsam als Gruppe haben wir uns für das Szenario Innenstadt entschieden. Inspiriert von Solarpunk ist es das Ziel ein modernes, menschenorientiertes aber trotzdem platzeffizientes Gelände zu erschaffen. Der Baustil soll modern und luxuriös aber trotzdem mit der Natur verbunden sein. Deswegen möchten wir die Umgebung mit geschwungenen organischen Formen gestalten und verschiedene moderne und klassische Materialien wie Granit, Sandstein, Holz und Glas mit einander verbinden. Die Gebäude sollen über Brücken miteinander verbunden werden, sodass man nicht ins Erdgeschoss zurück gehen muss, um von einem Haus zum nächsten zu gelangen. An allen Fassaden und in den öffentlichen Aufenhaltsbereichen wachsen Planzen, um für ein angenehmes Klima zu sorgen.

In dieser Stadt leben Menschen und werden von verschiedenen Robotern im Alltag unterstützt. Die Roboter übernehmen dabei die Arbeit, die nicht erfüllend, stupide oder gefährlich ist. Putzroboter halten Fußböden sauber, entleeren Mülleimer und heben Müll auf. Sie können auch komplexere Oberflächen wie Toiletten, Waschbecken oder Türgriffe reinigen. Essensverkäufer-Roboter arbeiten in Geschäften wie McDonalds, Nordsee oder Subway und nehmen Bestellungen entgegen und bereiten diese zu. Gepäckroboter helfen den Menschen ihre Einkäufe nach Hause zu tragen. Gärtnerroboter gießen Pflanzen, sägen Äste ab und sperren Bereiche ab. Sie arbeiten mit Menschen zusammen, welche Anweisungen geben, welche Äste abgeschnitten werden sollen.

Die Menschen sollen sich möglichst realistisch verhalten und haben ein eigenes Leben mit eigenen Zielen. Die Tagesabläufe von Kinder, Erwachsenen und Rentnern sollen sich je nach den individuellen Bedürfnissen unterscheiden. Diese Menschen interagieren mit den Robotern und arbeiten eng mit ihnen zusammen. Einige Kombinationen könnten zum Beispiel: Koch und Kellnerroboter, Gärtner und Gärtnerroboter oder Händler und Lagerroboter sein. Die Menschen interagieren aber

nicht nur mit Robotern, sondern auch miteinander. Sie gehen gemeinsam Essen, verreisen miteinander und haben sinnvolle, natürliche Unterhaltungen.

Die Menschen haben auch tragbare Computergestützte Systeme, die sie im Alltag begleiten. So kann zum Beispiel eine AR-Brille die Navigation übernehmen und über ein HUD die Wegpunkte direkt in der Welt anzeigen. Zusätzlich können noch weitere Informationen wie Ankünfte, Wartezeiten oder die nächsten Termine dargestellt werden.

1.2 Minimum Viable Product

Das Projektziel ist sehr umfangreich, weswegen wir ein zuerst ein Minimum Viable Product (MVP) erstellen möchten, das inhaltlich stark vereinfacht ist, aber alle Grundideen umsetzt. Der Plan ist zuerst das MVP zu entwickeln und dann aus dem MVP das Projektziel zu verwirklichen.

Die Szene soll aus zwei Platformen bestehen, die über eine Rampe miteinander verbunden sind. Auf jeder Platform befinden sich Pflanzen, die von fahrenden Robotern gegossen werden, wenn sie Wasser benötigen. Als Spieler kann man die Roboter fragen, um welche Pflanze es sich handelt.

Obwohl man das Minimum Viable Product in wenigen Sätzen beschreiben kann, sind trotzdem alle wichtigen Aspekte des Projektziels einbegriffen: die Szene ist nicht eben, sondern erstreckt sich über mehrere Plattformen. Als unabhängige Agenten hat man die Roboter, die selbständig Handlungen planen und ausführen. Man kann mit den Robotern Dialoge führen, die sich je nach Zustand der Welt anpassen.

Kapitel 2

Konzept

2.1 Vorbedingungen

Gemeinsam mit unserem Betreuer haben wir uns darauf geeinigt CrossForge zu verwenden. Die Engine unterstützt Windows, Linux und WebAssembly und benutzt nur OpenGL als Grafik API. Sie unterstützt physikalisch basiertes Rendering und Materialien, Deferred und Forward Rendering Pipelines und ein Shadersystem. Aber besonders wichtig für uns sind der Szenengraph, die Skelettanimationen und Text Rendering.

2.2 Architektur und benötigte Systeme

Aus dem MVP-Szenario und der Engine Wahl ergeben sich mehrere Anforderungen, die über verschiedene Systeme gelöst werden können. Die Roboter sind selbständige Agenten, die Entscheidungen treffen sollen und diese auch ausführen sollen. Dieses Problem löst das Multiagentensystem. Sie müssen ihren Weg zur nächsten durstigen Pflanze finden, dafür ist das Navigationssystem zuständig. Die Roboter müssen die Pflanze gießen, was durch ein Partikelsystem visualisiert werden soll. Der Spieler und die Roboter sollen sich auf verschieden Plattformen bewegen und von einer zur nächsten laufen können. Zusätzlich sollen die Pflanzen verschiebbar sein. Um diese beiden Sachen zu realisieren, wird ein Physiksystem benötigt. Der Spieler soll sich mit den Robotern unterhalten können, was durch das Dialogsystem abgedeckt wird. Da CrossForge keinen eigenen Szenen Editor hat und die MVP Szene schon zu komplex ist, um diese mit Programmcode zu beschreiben, ist ein Szenen Editor nötig.

Um alle Systeme möglichst flexibel und unabhängig voneinander zu gestalten, haben wir uns für das Entity Component Pattern entschieden.

2.3 Entity Component System

2.4 Partikelsystem

2.5 Multiagentensystem

Das Multiagentensystem ist das Herz unseres Projektes da das Verhalten von allen belebten und unbelebten Agenten von diesem System gesteuert werden sollen. Man unterscheidet zwischen Zentraler KI und Agentenbasierter KI. Bei Zentraler KI werden die Agenten von einem Agentexternen, globalen, allwissenden System gesteuert und haben deswegen keine Kontrolle über ihre eigenen Handlungen. Zentrale KI wird sehr oft eingesetzt, weil sich damit Gruppendynamiken und taktische Vorgehensweisen einfacher Implementieren lassen. Bei Agentenbasierter KI treffen die Agenten unabhängige und individuelle Entscheidungen basierend auf den Informationen, die dem Agent bereit stehen. Es gibt zwar trotzdem globale Informationen, die aber nicht dafür misbraucht werden dürfen die Handlungen eines Agenten zu diktieren.

Wie in verschiedenen GDC Talks empfohlen, soll unser Agentensystem aus mehreren Schichten bestehen:

- Sensoren
- Entscheidungsfindung
- Entscheidungsausführung
- Bewegungssteuerung

2.5.1 Sensoren

Sensoren sind ein Querschnittskonzept und tauchen deshalb in allen Ebenen auf. Sie filtern Informationen aus der Umgebung und stellen diese der Schicht bereit, in der sie eingesetzt werden. Zusätzlich können Informationen über Blackboards mit anderen Agenten geteilt werden.

2.5.2 Entscheidungsfindung

Für die Entscheidungsfindung habe ich mir Beliefs, Desires, Intentions (BDI), Goal Oriented Action Planning (GOAP) und Finite State Machines (FSM) näher angeschaut. Am Ende haben wir uns für Finite State Machines entschieden, weil das das einfachste Verfahren war und vorerst für die Roboter ausreicht.

Eine State Machine ist ein gerichteter Graph mit limitierter Anzahl an Stati und Aktionen. Der Agent wechselt von einem Status in den nächsten, falls eine Bedingung

erfüllt ist. In unserer Simulation sollen FSMs für die Roboter eingesetzt werden, um die Stati *Pflanzen gießen*, *Dialog mit Spieler*, *Spieler folgen*, etc abdecken zu können.

2.5.3 Entscheidungsausführung

Wenn der Agent eine Entscheidung getroffen hat, dann lässt sich diese Entscheidung meistens in weitere Teilprozesse zerlegen. Wenn ein Roboter zum Beispiel entschieden hat, dass er jetzt Pflanzen gießt, dann muss er:

- eine durstige Pflanze finden
- zur Pflanze hin fahren
- die Gießkanne zur Pflanze ausrichten
- und schließlich die Pflanze gießen

Diese Sequenz von Handlungen lässt sich nur sehr schlecht mit FSMs darstellen, weswegen dieser Nachteil durch Behaviour Trees ausgeglichen werden soll. Behaviour Trees sind sehr gut darin solche Sequenzen darzustellen oder sogar nebenläufige Handlungen zu beschreiben. Ihr Nachteil ist jedoch, dass sie nur sehr schwierig auf Übergänge reagieren können. Ein Beispiel hierfür ist, dass ein anderer Roboter die angefahrene Pflanze schon gießt. Um das jetzt in einem Behaviour Tree festzustellen, müssen überall Monitore eingebaut werden, welche den Baum unübersichtlich und nicht wartbar machen. Behaviour Trees und State Machines ergänzen sich also sehr gut.

In Behaviour Trees werden Handlungen durch Knoten beschrieben. Die Knoten können die Stati: *Laufend*, *Fehler* oder *Abgeschlossen* haben. Knoten können dabei über die Bearbeitung ihrer Kindknoten entscheiden. Bei einem Sequenzknoten werden die Kinder von links nach rechts abgearbeitet. Nur wenn der Knoten den Status *Abgeschlossen* hat, wird der rechte Geschwisterknoten aufgerufen. Wenn der Status *Laufend* ist, dann wird der Knoten solange bearbeitet, bis der Status *Abgeschlossen* oder *Fehler* erreicht ist. Bei dem Status *Fehler* wird die Abarbeitung abgebrochen und dieser nach oben propagiert. Der Elternknoten kann dann entscheiden, wie dieser Fehlerstatus behandelt wird. Der Fallbackknoten behandelt den Fehler zum Beispiel, indem er den ersten Kindknoten findet, der nicht fehlschlägt und somit erfolgreich ausführt. Nur wenn alle Kindknoten fehlschlagen, ist der Status des Fallbackknotens *Fehler*.

2.5.4 Bewegungssteuerung

Die Ebene der Bewegungssteuerung ist dafür verantwortlich die Entscheidungen in Beschleunigung, Geschwindigkeit und Rotationen umzuwandeln. Dafür haben wir uns Steering Behaviour näher angeschaut. Es gibt verschiedene Bewegungsmuster:

- Seek
- Wander
- Collision Avoidance
- Queue
- ...

Für uns ist vor allem Collision Avoidance in Verbindung mit Seek interessant.

2.6 Navigationssystem

In der Mitte der MVP-Szene befindet sich die Wendeltreppe, weswegen die Roboter nicht immer den direkten Weg zur Pflanze fahren können, weil sie dieses Hindernis beachten müssen. Das Navigationssystem übernimmt die Aufgabe einen Pfad von einem Startpunkt zum Zielpunkt zu finden. Um Zeit zu sparen und uns auf die Hauptaufgabe konzentrieren zu können haben wir uns entschieden die Bibliothek Recast und Detour einzusetzen. Recast berechnet das Navigationsmesh aus der statischen Geometrie und Detour findet zur Laufzeit einen Pfad auf diesem Navigationsmesh. Da das Navigationsmesh nicht dynamisch angepasst werden muss, reicht es aus, wenn man es einmal vor der Kompilierung berechnet, speichert und dann in CrossForge lädt. Die Bibliothek stellt hierfür ein Beispielprogramm bereit, was wir nutzen.

Um mit Detour einen Pfad mit Start und Ende zu finden, muss man zuerst die Start- und Endpolygone finden. Danach kann man mit Detour die Polygone finden, die das Start- und Endpolygon verbinden. Zum Schluss kann man mit einem weiteren Bibliotheksaufruf die Liste an Polygonen in eine Liste von Wegpunkten umwandeln und hat somit das Ergebnis.

Um einen Pfad für ein Entity zu finden, muss man dem Entity eine PathRequest-Component mit Start und Ziel hinzufügen. Das Navigationssystem ließt die Daten aus dem Request aus, entfernt diesen und fügt stattdessen eine PathComponent mit dem gefundenen Pfad hinzu.

2.7 Physiksystem

Die Entities und der Spieler sollen mit der statischen Szenengeometrie und sich selber kollidieren. Um diese Kollisionen zu erkennen und aufzulösen, ist das Physiksystem nötig. Da Linus schon einmal den Separating Axis Theorem Algorihmus in 2D implementiert hat und das sich als sehr Zeitaufwändig und Fehleranfällig herausgestellt hat, haben wir uns dazu entschieden die Bullet Bibliothek zu benutzen.

Bekommt ein Entity eine Physikkomponente zugewiesen oder wird das Entity entfernt, dann stellt das Physiksystem dies mit sogenannten Observern fest und fügt den Rigidbody der Physikwelt hinzu oder entfernt ihn.

Um jetzt die Entities kollidieren zu lassen benötigt man drei Schritte. Zuerst werden alle Entities mit Physikkomponente und Transformationskomponente gesucht und die Geschwindigkeit und Position der Transformationskomponente auf den Rigidbody der Physikkomponente übertragen. Dann wird die Physikwelt mit dem jetzigen Zeitschritt aktualisiert. Zum Schluss wird die Geschwindigkeit und Position des Rigidbodies wieder in die Transformationskomponente geschrieben.

2.8 Szenen Editor

Wir benötigen einen Szenen Editor, weil schon bei einer geringen Anzahl an platzierten Entitäten der Code unübersichtlich und schwer zu warten ist. Als alternative könnten wir die Agenten und Pflanzen prozedural platzieren, was aber ein zu komplexes Gebiet wäre und damit den Rahmen sprengen würde. Selber einen Szenen Editor von Grund auf neu zu schreiben ist keine Option, weil auch das eine riesige Aufgabe ist. Deswegen haben wir uns dazu entschieden die Open Source Software Blender zu verwenden. Blender ist ein mächtiger 3D-Editor den man über Addons erweitern kann. Somit haben wir zwei Möglichkeiten, unsere Szenen zu erstellen und in CrossForge zu laden. Wir können die Szene als GLTF-Datei exportieren und dann die vorhandenen Funktionen in CrossForge erweitern um aus der GLTF-Datei die einzelnen Transformationen für die Entitäten zu extrahieren. Oder wir schreiben ein Addon, welches die Szene nach unseren Anforderungen exportiert.

2.8.1 GLTF als Szenenbeschreibung

Khronos Group veröffentlichte am 19.10.2015 das offene GLTF Format, um dreidimensionale Szenen und Modelle darzustellen. In dem Format können Szenen mit ihren Knoten, Kamerainformationen, Animationen, Texturen, Materialien und natürlich auch Modellinformationen gespeichert werden. CrossForge unterstützt über die Assimp Bibliothek verschiedene Dateiformate, unter anderem auch GLTF. Assimp hat einen eigenen Szenengraph, den man traversieren kann und wenn man ein einm Knoten mit bestimmer Namenskonvention angelangt ist, weiß man, dass es sich um ein Entity handelt und kann alle Kindknoten zu einem Modell zusammen fassen und das Entity der Welt hinzufügen.

Dieser Ansatz hat den Vorteil, dass er Editoragnostisch ist und wir können vorhandenes Wissen über Assimp nutzen. Leider hat diese Variante aber auch große Nachteile: man muss die Änderungen ziemlich tief in CrossForge vornehmen, was sehr viele Code Änderungen nach sich zieht. Der Szenenersteller muzss die Knoten im Editor korrekt benennen, weil man sie in CrossForge sonst nicht mehr erkennen

kann.

2.8.2 Eigenes Format zur Szenenbeschreibung

Wir können ein Blender Plugin schreiben, was die Transformationen der einzelnen Entities in eine JSON-Datei exportiert. Das Problem ist, dass Blender erkennen muss, was statische Geometrie ist und was Entities sind. Es gibt aber eine Funktion um externe .blend Dateien in die aktuelle zu linken. Jetzt kann man die Regel aufstellen, dass alles, was gelinkt ist ein eigenes Entity ist. Die Vorteile sind, dass keine Eingriffe ins Engine-Innere nötig sind und da man das Dateiformat selber bestimmen kann, ist die Implementierung auf CrossForge Seite auch sehr simpel. Der Nachteil ist, dass noch niemand von uns ein Blender Addon geschrieben hat und wir deswegen noch keine Erfahrung in diesem Bereich haben.

2.9 Dialogsystem

Kapitel 3

Implementierung

3.1 Entity Component System

3.2 Multiagenten System

3.2.1 Sate Machines

nicht mehr dazu gekommen, weil Zeit vorbei war und Aufgaben von anderen Projektteilnehmern übernommen werden musste aber auch nicht nötig, weil Dialogsystem noch nicht fertig war

3.2.2 Behaviour Tree

3.2.3 Steering Behaviour

3.3 Partikelsystem

3.4 Navigationssystem

Keine Besonderheiten. Ist halt so wie im Konzept

3.5 Physiksystem

Da das Physiksystem mit Observern feststellt, ob eine Physikkomponente hinzugefügt wurde, muss die Komponente schon initialisiert sein und man kann die lazy Initialisierung nicht benutzen. Deswegen mit emplace statt add.

Die Platform und Wendeltreppe ist statische Geometrie und zusätzlich noch Konkav. Deswegen haben wir diese mit einem Dreieckskollider ausgestattet. Dabei wird einfach das 3D-Modell als Kollider benutzt.

Alle Entities haben Kapsel Collider, da die Entities sonst an den Dreieckskanten der statischen Geometrie hängen geblieben sind.

3.6 Szenen Editor

Die Assimp Variante hat nicht funktioniert, weil die eingriffe zu tief in der Engine vorgenommen werden mussten. Außerdem war das Abstraktionslevel zu gering: wenn man nur Knoten hat ist es schwer Entities zu erkennen.

Das Blender Plugin wurde in Python programmiert und man hat zugriff auf alle Optionen, auf die man auch im Editor Zugriff hat. Um jetzt die Entities zu exportieren, wurde über alle Objekte der Szene iteriert und überprüft, ob diese gelinkt sind. Wenn das der Fall ist, dann wurde die Transformation, der Name und der Name der Modelldatei in eine JSON-Datei geschrieben. Um jetzt auch noch die statische Geometrie zu exportieren, wurden alle Entities unsichtbar gemacht und nur die sichtbaren Modelle wurden exportiert.

In CrossForge wurde dann die JSON-Datei geladen und für jeden Eintrag wurde ein entsprechendes Entity zu Welt hinzugefügt.

3.7 Dialogsystem

3.8 Management

- Zusammensetzung .. generell nach versucht aufgaben nach stärken der Personen zu verteilen .. Angelique bricht Informatik Studim ab -> bekommt kreative Modellierungsaufgaben .. Lisa und Sophie haben bis jetzt weniger Erfahrung und bekommen leichtere Aufgaben .. Leon hat gewisse leidensfähigkeit und bekommt mittlere schwere Aufgaben .. Carlo und Linus haben meisten Erfahrungen, arbeiten neben dem Studium und bekommen schwersten Aufgaben

- Arbeisweise: .. wir haben uns auf 6h/Woche geeinigt, da Arbeit/Studium war nicht mehr drin .. Jira am Anfang angelegt, nicht wieder benutzt .. gegen Scrum entschieden, weil Retros/Plannings/Sprintwechsel zu viel overhead sind und wir leichtgewichtigen prozess brauchen .. Linus hat mit Lisa, Sophie, Leon eine Stunde in der Woche Pair Codings gemacht, um bei den Aufgaben zu helfen .. halbzeit haben wir eine retro gemacht .. retro hat festgestellt, dass wir verschiedene Arbeitsweisen mögen: Lisa und Sophie bringen Pair Codings weniger, Leon schätzt sie sehr und würde gerne fortfahren

- Angelique .. eigenständige einarbeitung in Blender .. völlig autark und angenehmes management: ich hab aufgaben rüber geworfen und die wurden erledigt .. hat mir Zeit erspart

- Dialogsystem wurde nicht fertig

- Carlo hat das Praktikum abgebrochen .. meiste CPP Erfahrung und arbeitet 18h pro Woche -> meiste Erfahrung und somit Aufgabe mit größten unsicherheiten bekommen .. Carlo hatte Aufgabe Detour in CrossForge zu integrieren .. Carlo hat aufgrund von Arbeit, Uni, Umzug, Problemen mit seiner Entwicklungsumgebung nur langsamen Fortschritt mit seiner Aufgabe was verständlicherweise zu frustration führt .. bei Retro hat er Angebot bekommen eine andere Aufgabe zu übernehmen -> hat er nach mehreren Wochen ohne nennenswerte Fortschritte angenommen .. nächste Aufgabe war Partikelsystem, Linus übernimmt Detour integration -> wieder IDE Probleme und kein Fortschritt .. Lust am programmieren verloren -> nächste Aufgabe war mit Angelique zusammen Modelle (Aufgabe vor Weihnachten erteilt), Leon übernimmt Partikelsystem .. über Jahreswechsel hat niemand von uns am Projekt gearbeitet und da Leon am Anfang Februar nach Polen fährt, muss das Projekt bis dahin abgeschlossen sein und die volle Konzentration sollte jetzt auf Bericht und Präsentation liegen -> keine neuen Modelle/Szenen .. somit hat Carlo nur extrem wenig beigetragen und somit entschieden, dass er das TP abbricht

.. mehr unterstützung und klein geschnittene Aufgaben wären hilfreich gewesen, konnte ich aber nicht machen, weil ich mich dafür so sehr in die Materie einlesen müsste, dass ich das Problem auch selber lösen kann. Ich hatte dafür auch keine Zeit, weil ich mit den restlichen Aufgaben und Teilnehmern überfordert war.

Kapitel 4

Ergebnisse

Jede Arbeit hat normalerweise Ergebnisse. Dies können Messreihen, Beweise und vieles mehr sein. In diesem Kapitel werden die Ergebnisse präsentiert und diskutiert. Meist ist die Implementation nicht vollkommen und zeigt in Randbereichen Schwächen. Hier ist der Platz dies aufzuzeigen.

Kapitel 5

Zusammenfassung & Ausblick

Die Zusammenfassung ist häufig das erste, was nach dem Titel einer Arbeit gelesen wird. Es sollte ein kurzer (circa eine Seite) Überblick über das Erreichte gegeben werden ohne sich in Details zu verlieren.

Im Ergebniskapitel wurden vielleicht Schwächen der Implementierung oder auch des Konzeptes aufgezeigt. Im Ausblick können hier nun Lösungsmöglichkeiten aufgezeigt werden, die sich im Verlauf der Bearbeitung nicht umsetzen ließen. Es sollten fundierte Lösungskonzepte erarbeitet werden. Weiterhin können anschließende Arbeiten in angrenzenden Gebieten vorgeschlagen werden.

Anhang A

Nachsätze

Im Anhang kommen die Sachen unter, die in der Arbeit keinen Platz haben. Hier finden sich eventuell ausführliche Algorithmenbeschreibungen. Aber auch Ergebnisse, die in der Arbeit sonst überflüssig wären, weil sie beispielsweise das gleiche zeigen wie die restlichen, können hier untergebracht werden.

A.1 Arbeitsaufteilung

- Angelique: Modelle
- Lisa und Sophie: Dialogsystem
- Leon:
  - Anbindung von Flecs
  - Implementierung von Multiagentensystem
  - Konzept und Implementierung Partikelsystem
- Linus:
  - Projekt Management
  - Architektur
  - Konzept Multiagentensystem
  - Konzept und Implementierung von Navigationssystem, Physiksystem, Szenen Editor