Technology
Arts Sciences
TH Köln

Murmelspiel

CGA Lernanwendung

Informatikprojekt

ausgearbeitet von

Andreas Pahlen und Stefan Förster

vorgelegt an der

Technischen Hochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang

Informatik (Bachelor)

Prüfer: Prof. Dr.-Ing. Martin Eisemann
Technische Hochschule Köln

Gummersbach, im Januar 2017

Kurzfassung

Das Murmelspiel ist eine webbasierte CGA Lernanwendung zur Orientierung im dreidimensionalem Raum. Das Ziel des Spiels ist, die Murmel in einen markierten Zielbereich zu bringen. Um dies zu erreichen, kann der Anwender mittels einfacher Programmierung (Syntax angelehnt an JavaScript) sogenannte Boxen erstellen, welche sich mittels Translation und Rotation platzieren lassen.

Um das Prinzip der sog. Modelmatrix zu verdeutlichen, lassen sich die Boxen selbst nicht direkt verschieben. Vielmehr muss der Anwender eine Matrix anlegen, welche sich dann translatieren und rotieren lässt. Den Boxen selbst kann man diese Matrizen als Modelmatrix zuweisen, sodass sich die Position der Box verändert.

Die Idee zu diesem Spiel entstand in der Projektfindungsphase. Wir berieten uns mit Herrn Prof. Dr. Eisemann über mögliche Themen für das anstehende Informatikprojekt, woraufhin wir uns vorerst grundsätzlich mit CGA und Three.js vertraut machen sollten. Während des Herumprobierens mit dem Framework haben wir festgestellt, dass uns persönlich die Orientierung im dreidimensionalen Raum schwerfällt.

Eine trockene Lernanwendung zu diesem Thema wäre allerdings nicht besonders interessant, weshalb wir uns für die sogenannte „Gamification“ (dt. „Spielifizierung“) entschieden, woraus die spezielle Idee des Murmelspiels dann entstand.

Inhaltsverzeichnis

1 Über das Murmelspiel

1.1 Graphische Oberfläche

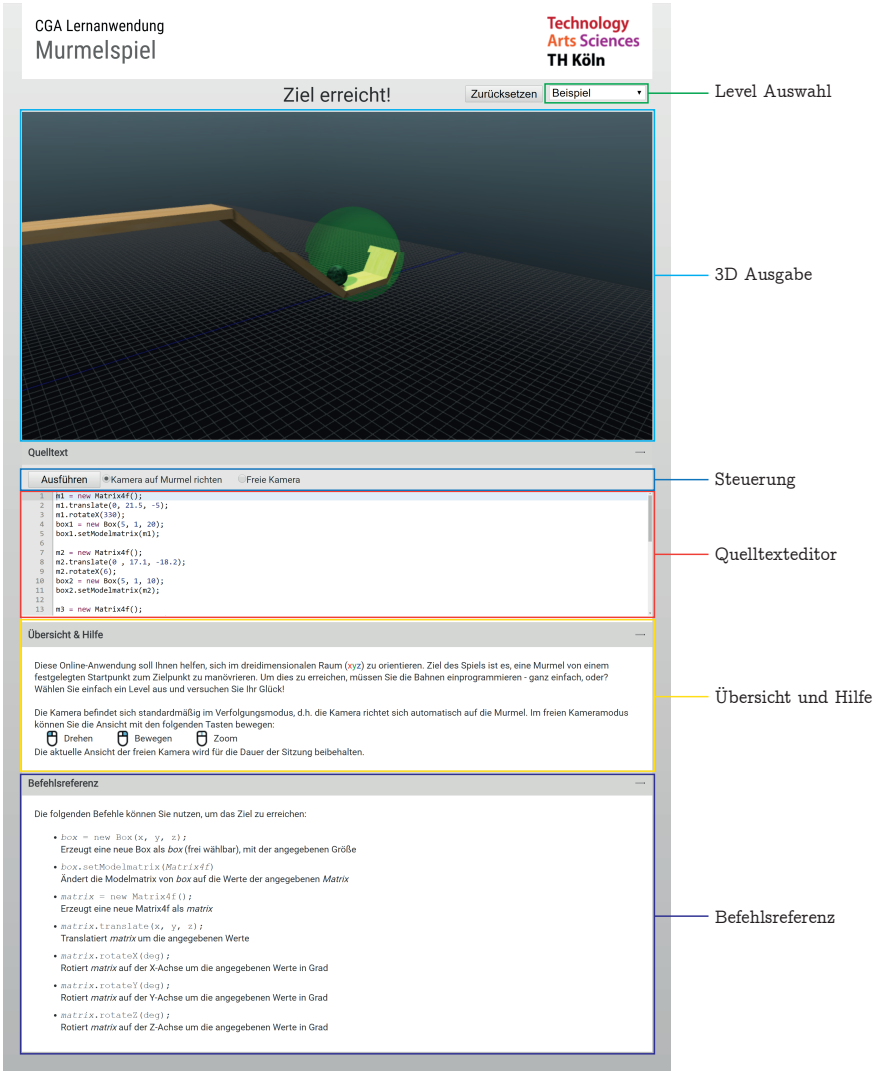

Abbildung 1.1: Spieloberfläche

Level Auswahl Die Level Auswahl dient zur Bestimmung des Spielmodi. Der Beispielmodus liefert einen vordefinierten Quelltext und dient zur Veranschaulichung des Spiels. Die eigentlichen Level geben einen festen Start- und Endpunkt der Murmel vor. Zur Lösung der Aufgabe, muss die Murmelbahn selbst programmiert werden. Mit jedem Level erhöht sich der Schwierigkeitsgrad, indem beispielsweise Wände den direkten Weg zum Ziel versperren.

3D Ausgabe In der 3D Ausgabe wird der Quelltext aus dem Editor mit Hilfe von Three.js unter den Bedingungen von Physijs gerendert und dargestellt.

Quelltexteditor An dieser Stelle wird der Quellcode der Murmelbahn geschrieben. Der Quelltexteditor selbst verfügt über eine einfache Echtzeit-Syntaxfehlererkennung und stellt erkannte Fehler mit Hilfe von Symbolen links der fehlerhaften Zeile dar. Treten syntaktische Fehler nach dem Ausführen auf, so werden die Fehlermeldungen rechts des Editors dargestellt sowie die fehlerhaften Zeilen rot eingefärbt.

Steuerung Der Steuerungsbereich dient zum Ausführen des Quellcodes und Einstellen des Kameramodus.

Übersicht und Hilfe Dieser Bereich hilft dem Nutzer, das Konzept des Murmelspiels in Kürze zu Verstehen. Zudem wird an dieser Stelle die Tastenbelegung für die Kamerasteuerung mit Hilfe der Maus beschrieben.

Befehlsreferenz Die Befehlsreferenz listet alle nutzbaren Befehle für den Quellcode auf und dient somit als Nachschlagewerk.

1.2 Befehle

Es stehen dem Nutzer die folgenden Befehle zur Verfügung:

box = new Box(x, y, z); Erzeugt eine neue Box als box (frei wählbar), mit der angegebenen Größe.

box.setModelmatrix(Matrix4f); Ändert die Modelmatrix von box auf die Werte der angegebenen Matrix.

matrix = new Matrix4f(); Erzeugt eine neue Matrix4f als matrix.

matrix.translate(x, y, z); Translatiert matrix um die angegebenen Werte.

matrix.rotateX(deg); Rotiert matrix auf der X-Achse um die angegebenen Werte in Grad.

matrix.rotateY(deg); Rotiert matrix auf der Y-Achse um die angegebenen Werte in Grad.

matrix.rotateZ(deg); Rotiert matrix auf der Z-Achse um die angegebenen Werte in Grad.

1.2.1 Beispiel

Beispiel Quelltext:

```
1 m1 = new Matrix4f();
2 m1.rotateX(330);
3 m1.translate(0, 21.5, -5);
4 box1 = new Box(5, 1, 20);
5 box1.setModelmatrix(m1);
```

In der ersten Zeile wird eine neue Matrix erstellt. In den Zeilen 2 bis 3 wird diese zuerst rotiert, und schließlich translatiert, um die geeignete Position zu erreichen. In Zeile 4 wird nun eine Box erstellt und darauf folgend die zuvor erstelle Matrix zugewiesen.

1.3 Lerneffekt

Mit dieser Lernanwendung sollen Studenten ihre Orientierung im dreidimensionalem Raum verbessern und zu gleich die Translation und Rotation von Objekten mit Hilfe von Matrizen erlernen. Die Lernanwendung stellt ein komplettes Framework mit allen benötigen Komponenten zu Verfügung, so dass ein schneller Lernerfolg und keine Frustration durch etwaiges vorbereiten einer eigenen Applikation auftritt.

Durch Vorgabe des Ziels und dem Start der Murmel, ohne dessen Koordinaten, muss sich der Anwender erstmals am Koordinatensystem orientieren, um diese Koordinaten ausfindig zu machen. Im Anschluss wird die Translation und Rotation anhand der Erstellung der Murmelbahn trainiert. Die Murmelbahn wird aus einem einfachen 3D Objekt erstellt und dann unter den Bedingungen der Matrizen angepasst.

Der Lerneffekt erhöht sich spielerisch, dank der Berücksichtigung von physikalischen Eigenschaften der Kugel und Bahn, denn somit müssen Erfahrungen aus der realen Welt mit eingebracht werden.

2 Programmierung

Das Murmelspiel wurde mit Hilfe der Programmiersprachen HTML, JavaScript und PHP entwickelt. Im Folgenden wird erklärt, wie die wichtigsten Komponenten des Spiels funktionieren.

2.1 Der Parser

Wichtigster Bestandteil des Murmelspiels ist der sogenannte Parser. Dieser übersetzt die Quellcode-Eingaben des Benutzers in ausführbaren JavaScript-Code. Diese Umwandlung ist nötig, um dem Benutzer eine möglichst problemlose Nutzung zu ermöglichen, um das Hauptaugenmerk auf das Spiel, weniger auf die Programmierung zu legen.

2.1.1 Übersetzung von Codezeilen

Der Parser wurde durch eine Verkettung von Regular Expressions realisiert. Abbildung 2.1 zeigt einen Überblick über die Funktionsweise des Parsers.

Diese Form der Übersetzung ist sehr schnell, relativ einfach zu erweitern und mit korrekten Regluar Expressions auch sehr zuverlässig. Wichtig bei der Entwicklung der Regular Expressions ist, alle möglichen und gültigen Benutzereingaben einzuplanen. Dazu zählt beispielsweise das Einfügen von beliebig vielen Leerzeichen, was in den meisten Programmiersprachen gültig ist.

Beispiel einer Regular Expression anhand von "rotateX":

```
1 /^\s*(.+\S).rotateX\s*\(\s*(-?\d+\.?\d*)\s*\);/
```

Gültige Eingaben für "rotateX":

```
1 matrix.rotateX(45);
2 matrix.rotateX   ( 45 );
3    matrix.rotateX(45);
```

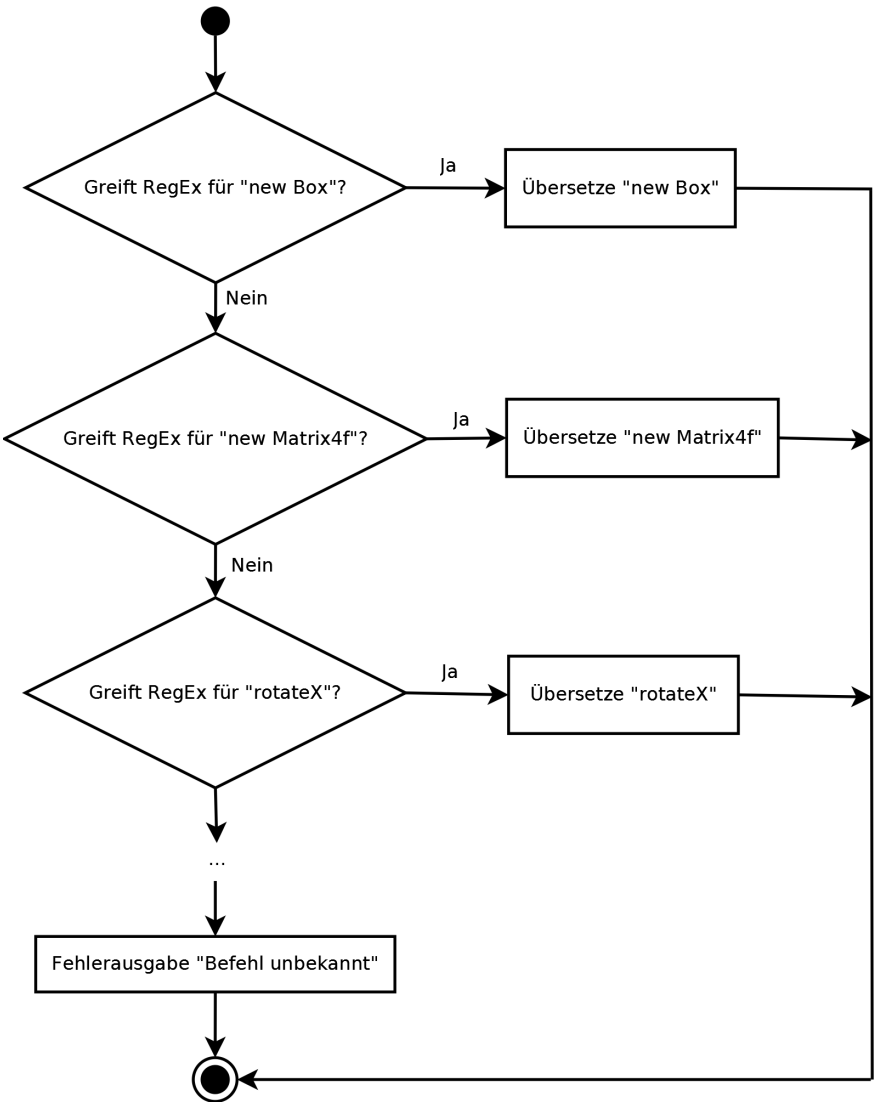

Abbildung 2.1: Funktionsweise des Parsers

`Ungültige Eingaben für "rotateX":`

```
1 matrix. rotateX(45);
2 matrix.rotateX(45) ;
```

Um eine weitere Fehlersicherheit zu gewährleisten, wird vor der Übersetzung der Code-Zeile überprüft, ob das angegebene Objekt existiert. Wird in der internen Matrizen-Liste keine Matrix mit dem angegebenen Namen gefunden, so wird die Übersetzung abgebrochen und ein Fehler ausgegeben.

Codeauszug für den Befehl "rotateX":

```
1  // RotateX Matrix
2  $result = preg_match(
3      "/^\s*(.+\S).rotateX\s*\(\s*(-?\d+\.?\d*)\s*\);/",
4      $line, $values);
5  if($result == 1) {
6      if(is_a($matrices[$values[1]], "MatrixHelper")) {
7          $matrices[$values[1]]->rotateX($values[2]);
8          return null;
9      }
10     else {
11         $error_array[] = "{\n\"line\":
12             ".($no + 1).",\n\"error\": \"\\\"
13             ".$values[1]."\\\" unkown (
14             ".str_replace(array("\r", "\n"),
15             '', htmlentities($line)).")\"\n}";
16         $error_count++;
17         return null;
18     }
19 }
```

In Zeile 2 bis 4 wird die Regular Expression ausgeführt, dabei werden die ausgelesenen Werte (hier: Name der Matrix sowie der Wert um den rotiert werden soll) in die Variable $values gespeichert. Konnte die Regular Expression erfolgreich auf $line angewendet werden, so wird in $result der Wert "1" gespeichert.

In Zeile 5 wird nun überprüft, ob die Regular Expression erfolgreich war. Wenn ja, so wurde erfolgreich der Befehl "rotateX" erkannt.

In Zeile 6 erfolgt nun die Überprüfung, ob eine Matrix mit dem angegebenen Namen ($values[1]) in der Matrizen-Liste vorhanden ist.

In Zeile 7 wird die interne Rotation ausgeführt (s. Kapitel 2.1.2). Anschließend wird mit "return null" die Ausführung der Parser-Funktion beendet.

Sollte die angegebene Matrix nicht existieren, so wird in den Zeilen 11 bis 16 ein Fehler angelegt und anschließend ebenfalls die Ausführung der Parser-Funktion beendet.

2.1.2 Implementation der Matrizen

Während der Code-Übersetzung müssen die angelegten Matrizen intern zur Weiterverarbeitung abgespeichert werden. Es werden jedoch keine tatsächlichen Matrizen gespeichert, sondern nur die durchgeführten Translationen und Rotationen, welche später (bei der Zuweisung einer Modelmatrix) in JavaScript-Code übersetzt werden.

Zur Speicherung der Matrizen dient das Array $matrices, welches Objekte der Klasse "MatrixHelper" enthält.

Codeauszug der Klasse "MatrixHelper":

```
1  class MatrixHelper {
2      public $history = [], $size = 0;
3
4      function translate($x, $y, $z) {
5          $this->history[] = "translate,".$x.",".$y.",".$z;
6          $this->size++;
7      }
8
9      function rotateX($deg) {
10         $this->history[] = "rotateX,".$deg;
11         $this->size++;
12     }
13
14     function rotateY($deg) {
15         $this->history[] = "rotateY,".$deg;
16         $this->size++;
17     }
18
19     function rotateZ($deg) {
20         $this->history[] = "rotateZ,".$deg;
21         $this->size++;
22     }
23 }
```

Das Array $history enthält die durchgeführten Translationen und Rotationen in Textform. Beim Aufruf einer Methode wird ein neuer Eintrag angelegt, der den Befehl, sowie den oder die Werte enthält.

Beispieleinträge für $history:

```
1 rotateX,45
2 translate,10,20,30
```

Beim setzen der Modelmatrix werden diese Informationen ausgelesen und in JavaScript-Code übersetzt.

Codeauszug der Funktion "setModelmatrix" (Auszug):

```
1 $parts = explode(",", $matrices[$values[2]]->history[$i]);
2 if ($parts[0] == "rotateX") {
3     $output .= $values[2]."rotX = new THREE.Matrix4();\n";
4     $output .= $values[2]."rotX.makeRotationX(
5         ".degToRad($parts[1])."");\n";
6     $output .= $values[2].".multiplyMatrices(
7         ".$values[2]."rotX, ".$values[2]."");\n";
8 }
9 [...]
```

Dieser Auszug wird in einer Schleife ausgeführt, wodurch alle abgespeicherten Einträge in der korrekten Reihenfolge abgearbeitet werden.

In Zeile 1 wird der zuvor angelegte String, der alle Informationen enthält, wieder zerteilt. Zeile 2 überprüft nun, um welche Funktion es sich handelt und erzeugt dann die Ausgabe in JavaScript-Code. Der Bedingungsblock von Zeile 2 bis 8 taucht im Code für jede Rotation, sowie der Translation auf.

2.2 Zusammenspiel der verwendeten Programmiersprachen

Die Zusammenarbeit der verwendeten Programmiersprachen bildet die funktionale Basis des Murmelspiels. Abbildung 2.2 zeigt den sequenziellen Ablauf der wesentlichen Spielmechanik, der bei einem Abschicken des Quellcodes durch den Benutzer durchlaufen wird.

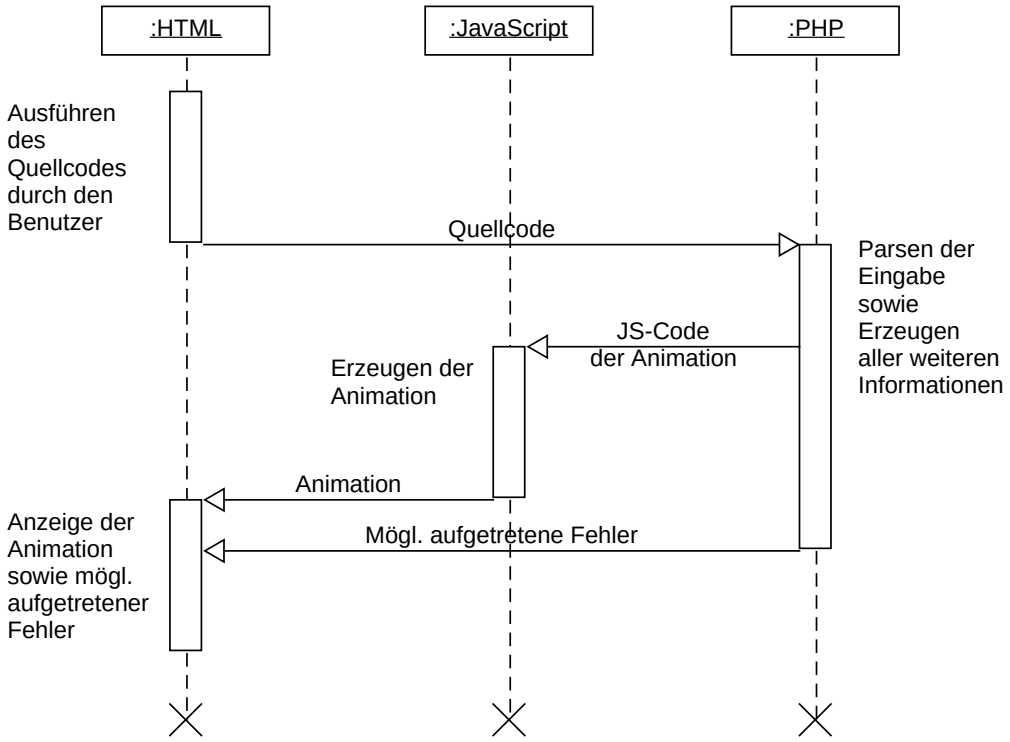

Abbildung 2.2: Zusammenspiel von HTML, JS und PHP

Der Benutzer klickt in der Webanwendung auf den Button "Ausführen". Dadurch wird der Quelltext an den Webserver übertragen. Die PHP-Komponente übersetzt nun den eingegebenen Quelltext in ausführbaren JavaScript-Code und übermittelt ebenso möglicherweise aufgetretene Fehler. Diese Informationen werden nun zurück an den Benutzer übertragen, dessen Computer nun mit Hilfe von JavaScript die Animation erstellt. Die Animation sowie alle weiteren Inhalte werden nun mit Hilfe von HTML sichtbar dargestellt.

Diese Aufgabenverteilung ermöglicht eine vergleichsweise einfache Programmierung. Grundsätzlich wäre es auch möglich, den gesamten Ablauf auch ohne PHP zu reali-

sieren, jedoch muss heutzutage immer noch damit gerechnet werden, dass verschiedene Browser die JavaScript-Programmierung unterschiedlich interpretieren. Dies hätte einen erhöhten Entwicklungsaufwand zur Folge, ohne spürbare Vorteile für den Benutzer.

2.3 Hinzufügen weiterer Level

Das Murmelspiel lässt sich einfach um weitere Level erweitern. Dabei muss zunächst im Unterordner "levels" eine Textdatei angelegt werden, welche alle nötigen Informationen enthält. In diesen Dateien lassen sich beispielsweise die Murmel, Zwischenwände und Lichter platzieren. Zudem können dort auch Eigenschaften wie z.B. die Gravitation angepasst werden. Beim Hinzufügen eines weiteren Levels sollte sich an den bereits existierenden Dateien orientiert werden.

Zudem muss in der Datei "levels.ini" im Hauptordner ein weiterer Eintrag hinzugefügt werden. Ein Beispieleintrag sieht folgendermaßen aus:

Beispieleintrag der "levels.ini"-Datei:

```
1 [level_name]
2 name = "Ausgeschriebener Level-Name"
3 filename = "dateiname.txt"
```

Der Dateiname muss dem Namen der neu angelegten Datei im Unterordner "levels" entsprechen, jedoch ohne den Ordner dabei anzugeben.

2.4 Verwendete Komponenten

Im folgenden Stellen wir die Frameworks bzw. Komponenten vor, die wir zur Umsetzung des Murmelspiels verwendet haben.

Three.js ist eine JavaScript 3D Bibliothek, die die Programmierung mit WebGL vereinfachen soll. Erstmals veröffentlicht wurde die Bibliothek im April 2010, die Ursprünge gehen jedoch bis in die frühen 2000er zurück. Die Unterstützung für WebGL wurde kurz nach der Veröffentlichung von WebGL implementiert, zuvor wurde ein Canvas-Renderer sowie ein SVGRenderer angeboten, welche allerdings schnell an ihre Leistungsgrenzen stießen.[1]

`https://www.threejs.org`

Physijs ist ein Interface zur Einbindung von physikalischen Eigenschaften in Three.js. Es verknüpft die Physik-Engine ammo.js mit Three.js und ermöglicht so eine Simulation von Gravitations- sowie Kollisionseigenschaften.

`https://chandlerprall.github.io/Physijs/`

Ace ist ein Texteditor für Internetseiten und bietet zahlreiche Funktionen zur komfortablen Programmierung, darunter beispielsweise Syntax-Highlighting und eine automatische Fehlererkennung. Ace ist ein Teilprojekt der Cloud9 IDE und wird ständig weiterentwickelt.

`https://ace.c9.io`

[1] Quelle: `https://en.wikipedia.org/wiki/Three.js`

3 Persönliche Erfahrungen

3.1 Andreas Pahlen

Meine bisherigen Erfahrungen mit CGA gingen gegen Null, weshalb ich beim ersten Termin mit Herrn Prof. Eisemann skeptisch war, ob die Durchführung eines Informatikprojektes in diesem Themenbereich die richtige Wahl ist. Bisher war ich der Meinung, dass CGA ein unglaublich schwieriges und schwer zu erlernendes Thema ist, grundsätzlich bin ich aber offen für neue Themengebiete und Herausforderungen und sah dieses Projekt als Chance, meine Kenntnisse zu erweitern.

Ich habe mich zunächst mit Three.js beschäftigt, ein wenig herum experimentiert und schnell gemerkt, dass der Umgang mit diesem Framework wesentlich einfacher als gedacht ist. Zudem ist viel darüber im Internet zu finden, was den Lernprozess stark vereinfacht, weil man auf Erfahrungen und Probleme anderer Nutzer zurückgreifen kann.

Als uns dann die Idee mit dem Murmelspiel kam, fing ich sofort an zu überlegen, wie man dieses Projekt am besten umsetzen kann. Durch meine langjährige Erfahrung im Bereich der Webentwicklung, insbesondere PHP, hatte ich schnell ein Grundgerüst im Kopf, wie man das System hinter der Benutzeroberfläche (insbesondere dem Parser) einfach und effizient entwickeln kann.

Mein Projektpartner, Stefan Förster, begann daraufhin mit der Umsetzung des Grundgerüstes und gab damit den endgültigen Anstoß für dieses Projekt. Ich entwickelte zeitgleich den Parser, was zunächst ohne größere Probleme vonstatten ging. Zu diesem Zeitpunkt verfügte das Murmelspiel jedoch nicht über eine Implementierung von Matrizen, sondern die platzierbaren Boxen waren direkt rotier- und platzierbar (so wie es Three.js anbietet).

Anschließend mussten wir stark an der Usability des Spiels arbeiten, um dem Spielerlebnis nicht mit einer schwierigen bzw. unnatürlichen Bedienung zu schaden. Dieser Teilbereich wurde während des gesamten Entwicklungsprozesses optimiert, da wir be-

reits während der Entwicklung gemerkt haben, welche Funktionen bzw. Fehlfunktionen die einfache Nutzbarkeit beeinflussen.

Beim darauf folgenden Treffen mit Herrn Prof. Eisemann einigten wir uns darauf, dass wir Modellmatrizen implementieren, um der realen CGA-Programmierung möglichst nah zu kommen. Dies stellte mich, als Entwickler des Parsers, vor Probleme, denn Three.js ist eigentlich nicht dafür ausgelegt mit Matrizen zu arbeiten, denn genau diese Schwierigkeit soll eigentlich durch das Framework abgenommen werden. Es gab viele Fehlversuche, ohne genau zu wissen, warum es nicht funktioniert.

Es folgte eine Fehlimplementierung, welche keine Rücksicht auf die Reihenfolge der Rotationen bzw. Translationen nahm. Nach einem zweiten Termin mit Herrn Prof. Eisemann kam uns dann die Idee, die Matrizen so umzusetzen, wie sie in Kapitel 2.1.2 beschrieben wird. Nun hatte ich auch die Verwendung von Matrizen in der CGA vollständig verstanden, was meine Kenntnisse erneut erweiterte.

Abschließend stelle ich fest, dass mir dieses Projekt die "Angst" vor CGA genommen hat, da es insgesamt doch einfacher ist als zunächst gedacht. Auch das hinzugekommene Thema der Modellmatrizen stellte ich mir schwerer vor, als es eigentlich ist. Ich konnte insgesamt meine Kenntnisse erweitern und bin froh, mich diesem neuen Themenbereich gestellt zu haben.

Ich bin sehr zufrieden mit dem Ergebnis des Projektes, da wir eine Lernplattform geschaffen haben, die hoffentlich vielen Studenten helfen wird, ohne dabei den Spaß am Themenbereich der CGA zu rauben. Ich bin mir relativ sicher, dass viele andere Studenten ähnliche Probleme mit der Orientierung im dreidimensionalen Raum und somit auch mit der Platzierung von Objekten haben, weshalb ich das Murmelspiel als sinnvolle Erweiterung des Repertoirs von CGA-Lernanwendungen der TH Köln einschätzen würde.

3.2 Stefan Förster

Das Thema CGA war mir bis zu diesem Projekt ziemlich unbekannt, aber dennoch sah ich es als Herausforderung es kennen zu lernen. Begriffe wie OpenGL, DirectX, CUDA waren mir bekannt, aber wie man damit programmiert und eigene Anwendungen umsetzt war für mich ein unerforschtes Gebiet. Als Herr Eisemann vorschlug eine Webanwendung mit 3D Grafik zu erstellen, war natürlich der erste Gedanke, dass dies zu komplex wird, ohne gründliches Vorwissen. Er empfahl uns aber ein Javascript Framework "Three.js", mit dessen Unterstützung es möglich ist, 3D Grafik in Webanwendungen zu implementieren. Zusätzlich sollte es eine Lernanwendung für das Fach CGA werden.

Nach einem ausführlichen Brainstorming mit meinem Projektpartner, Andreas Pahlen, kamen wir auf die Idee ein Murmelspiel zu erstellen, in dem die Studenten eine Murmelbahn programmieren müssen. Dadurch dass Andreas sehr fit in Webprogrammierung, vor allem mit PHP ist, war es meine Aufgabe das Grundgerüst der Anwendung zu bauen. Nach ausführlicher Recherche im Internet über Three.js waren auch alle Hürden aus dem Weg geräumt, da es eine große Community und viele Beispiele dazu gab. Nun fehlte nur noch eine physikalische Engine, die die Berechnung der Gravitation übernimmt. Auch hier fand sich nach kurzer Recherche ein gutes Framework, welches auf Three.js aufbaut: Physijs. Nach dem ich alle Komponenten zusammen hatte, begann ich mit dem Grundgerüst der Lernanwendung und baute eine erste Murmelbahn auf.

Ziel der Anwendung war es unter anderem ein einfaches Interface den Studenten zu bieten, um keine Einbußen an dem Lernerfolg zu erhalten. Als Vorgabe für das Design gab uns Herr Eisemann eine ältere Lernanwendung, die schon veröffentlicht wurde. Dies setzte ich auf der Seite um, währenddessen Andreas sich um einen Parser kümmerte, der zur Einfachheit der Syntax dienen soll. Natürlich darf es keinem Spiel an Leveln fehlen, so war es meine Aufgabe, mir Level von leicht bis schwer zu überlegen, die den User herausfordern, aber nicht die Freude an der Lernanwendung hindern. Unter anderem wurde ein Level mit negativer Gravitation implementiert, um auch die Orientierung der Anwender zu fördern.

Zusammengefasst, bin ich stolz auf diese Lernanwendung, da es die Orientierung im dreidimensionalen Raum und die Programmierung mit Matrizen fördert und zu dem einen gewissen Spaßfaktor bietet. Die Lernanwendung hat mich persönlich auch näher zu dem Thema CGA gebracht und die scheu vor der Programmierung mit 3D Grafik genommen.

Abbildungsverzeichnis

Glossar

HTML ..... Hypertext Markup Language. Auszeichnungssprache zur Erstellung u.a. von Internetseiten.

JavaScript ..... Programmiersprache zur Erstellung u.a. von clientbasierten, dynamischen Webanwendungen.

JS ..... Abkürzung für JavaScript.

Parser ..... Computerprogramm, das für die Zerlegung und Umwandlung einer Eingabe in ein für die Weiterverarbeitung geeigneteres Format zuständig ist. Quelle: `https://de.wikipedia.org/wiki/Parser`

PHP ..... PHP: Hypertext Preprocessor. Programmiersprache zur Erstellung u.a. von serverbasierten, dynamischen Webanwendungen.

Quellcode ..... In einer Programmiersprache geschriebener Text eines Computerprogramms. Quelle: `https://de.wikipedia.org/wiki/Quelltext`

RegEx ..... Abkürzung für Regular Expression.

Regular Expression . Eine Zeichenkette, die der Beschreibung von Mengen von Zeichenketten mit Hilfe bestimmter syntaktischer Regeln dient. Quelle: `https://de.wikipedia.org/wiki/Regulärer_Ausdruck`

Syntax ..... Regelsatz zur korrekten Schreibweise von Befehlen und Ausdrücken in Programmiersprachen.

WebGL ..... Web Graphics Library. Ein Bestandteil vieler moderner Webbrowser, mit dessen Hilfe 3D-Grafiken hardwarebeschleunigt ohne zusätzliche Erweiterungen dargestellt werden können. Quelle: `https://de.wikipedia.org/wiki/WebGL`

Anhang

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, 16. Januar 2017

Andreas Pahlen und Stefan Förster