

Fakultät   
Ingenieurwissenschaften und Informatik

Advanced Rich Media Applications   
Sommersemester 2018

Projektbericht

über das Thema

Erstellung eines Multiplayer Shooter Spiels unter Verwendung von NodeJS, ThreeJS, CannonJS, MySQL und Socket.IO

|  |  |
| --- | --- |
| Autoren: | Dennis Lammers (675452)  Henri Hesse (671798)  Alexander Beiserz (123456) |
| Prüfer: | Dipl.-Inf. (FH) Björn Plutka |
| Abgabedatum: | 14.09.2018 |

Inhaltsverzeichnis

|  |  |
| --- | --- |
| Einleitung | 1 |
| Architektur | 2 |
| Ablauf | 3 |
|  |  |
|  |  |

// Note: Tabelle vor Abgabe unsichtbar machen, wie auf dem Deckblatt ganz unten!

Einleitung

Die erstellte Anwendung ist ein webbasiertes Multiplayer-Shooter-Spiel im Science Fiction Genre. Es gibt zwei Teams, die gegeneinander im Team-Deathmatch-Szenario antreten und versuchen, möglichst viele Punkte für ihr Team zu gewinnen.



Für die Visualisierung wird die browserbasierte 3D-Grafikbibliothek ThreeJS genutzt. Durch die Verwendung von WebGL profitieren Benutzer von GPU-beschleunigten Berechnungen und der daraus resultierenden, hohen Performance. Außerdem wird dieses Framework stetig weiterentwickelt und ermöglicht eine einfache Einbindung von eigenen 3D-Modellen und Animationen. Desweiteren hat ThreeJS eine integrierte Audiobibliothek, die für Soundeffekte im Spiel genutzt wird.

Das Verhalten von Objekten im dreidimensionalen Raum, sowie die Erkennung von Kollisionen wird durch die Physics-Engine – CannonJS – berechnet.

Das Multiplayer-Feauture, der Chat, sowie die Kommunikation von Server und Clients wird durch Sockets mit Verwendung von Socket.IO ermöglicht.

Durch Nutzung von MySQL können Daten serverseitig gespeichert und bei Bedarf abgefragt werden.

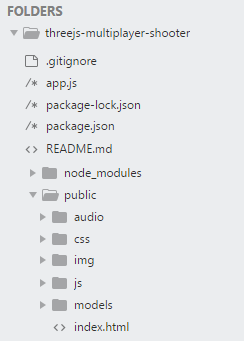
Die Login- und Registrierungsseite, sowie Textelemente im Spiel werden durch Nutzung von CSS-Flex und Bootstrap positioniert und benutzerfreundlich gestaltet. Desweiteren hilft jQuery bei der Editierung von HTML-Elementen.

Für die Versionsverwaltung kommt git mit Hilfe von Sourcetree zum Einsatz.

Architektur

Die Webanwendung läuft auf einem NodeJS-Express-Server, der über die Datei *app.js* im Root-Verzeichnis des Projektes gestartet wird. Der Server übernimmt das Routing, sowie den Request-Response-Mechanismus, um dem Client bei Anfrage die Datei *index.html* aus dem *public* Ordner zukommen zu lassen. Dieser Ordner beinhaltet alle für den Client einsehbaren Dateien, das Root-Verzeichnis bleibt unsichtbar. Die *index.html* ist in diesem Projekt die einzige HTML-Datei. Dort sind alle für die Anwendung benötigten Ressourcen eingebunden, sodass das Spiel bei dem Client direkt gestartet werden kann.

Folgende Übersicht veranschaulicht die Struktur des Projektes:



Im Root-Verzeichnis des Projektes befindet sich zudem die Datei *package.json*, die alle für die Webanwendung benötigten Module als Abhängigkeiten deklariert. Mit dem Paketverwaltungssystem npm können diese Module einfach und schnell eingebunden werden.

Eine serverseitige MySQL-Datenbank speichert nach erfolgreicher Registrierung eines Benutzers dessen Daten ab. Die ThreeJS- und CannonJS-Instanzen laufen lokal bei dem Client. Während des Spiels kommunizieren die Clients und der Server durch Sockets, um so den Stand des Spiels bei allen Clients zu synchronisieren.

Ablauf

Im Folgenden soll die Funktionsweise der Anwendung durch einen beispielhaften Ablauf dargestellt werden:

Ein Benutzer öffnet die Website des Spiels. Automatisch werden alle für das Spiel benötigten Ressourcen geladen und der Benutzer als Beobachter in das Spiel integriert. Bevor er sich registriert, kann er also schon anderen Spielern zuschauen. Bei erfolgreicher Registrierung wird eine E-Mail an den Benutzer geschickt und die Daten in der Datenbank gespeichert. Meldet sich der Benutzer nun an, so werden die eingegebenen Daten mit denen aus der Datenbank verglichen. Passen die Datensätze zusammen, so wird der Benutzer als Spieler dem Team, das weniger Spieler besitzt, hinzugefügt.

Da diese Zustandsänderung alle Spieler betrifft, muss der neue Stand des Spiels mit allen Spielern synchronisiert werden. Dies geschieht durch Socket-Events. Der neue Spieler bekommt die Daten aller aktiven Spieler und integriert diese in seine Spieldateien, wobei alle anderen Spieler lediglich den neuen Spieler integrieren müssen. Jetzt ist der Stand des Spiels bei allen Spielern wieder gleich. Um dies auch fortlaufend gewährleisten zu können, müssen jetzt alle Zustandsänderungen des neuen Spielers, wie z.B. Änderungen der Position und Rotation der Spielfigur über die Socket Schnittstelle an alle anderen Spieler geschickt werden. Andersherum müssen auch alle Änderungen der anderen Spieler auf die lokale Spielinstanz übertragen werden. Das Aktualisieren dieser Daten geschieht pro Frame genau einmal und synchronisiert die Spieler dadurch miteinander.

Beim Schießen von Kugeln wird ähnlich vorgegangen. Der Benutzer drückt die linke Maustaste und es wird der Richtungsvektor vom Kamera- in den Weltraum berechnet. Dieser wird zusammen mit der Spieler-Id an alle weiteren Spieler übertragen und löst dort das Schuss-Event aus. Wie bei dem Spieler, der an einer bestimmten Position im Weltraum in eine bestimmte Richtung geschossen hat, so wird lokal bei den anderen Spielern auch eine Kugel mit genau diesen Werten erzeugt. Durch die Physics Engine CannonJS verhält sich die Kugel bei jedem Spieler gleich. Wird ein Spieler von einem Gegenspieler des anderen Teams getroffen, so verliert dieser Lebensenergie, oder stirbt. Bei der Kollision zwischen Kugel und Spieler kann durch die Spieler-Id zwischen Freund oder Feind differenziert werden und durch Übergabe an den Server, bei einem Tod entschieden werden, wer den „Kill“ gutgeschrieben bekommt.

Stirbt ein Spieler, so erscheint er mit voller Lebensenergie in seiner Basis wieder und das gegenerische Team bekommt einen Punkt gutgeschrieben. Bei Verlassen eines Spielers wird ein Socket-Event bei allen anderen Spielern ausgelöst, das anhand der richtigen Id alle Ressourcen dieses Spielers löscht.

Nachfolgend soll eine Liste mit verwendeten Socket-Events und vereinfachtem Code die Multiplayer-Funktionalitäten verdeutlichen:

io.on('connection', function(socket)

{

socket.on('new user', function(username)

{

socket.emit('allplayers', getAllPlayers() );

socket.broadcast.emit('new user', socket.player);

socket.broadcast.emit('chat message', message );

});

socket.on('move', function(moveData)

{

socket.broadcast.emit('move', moveData);

});

socket.on('movingstate', function(state)

{

socket.broadcast.emit('movingstate', data);

});

socket.on('shoot', function(shootData)

{

socket.broadcast.emit('shoot', shootData);

});

socket.on('score', function(score)

{

io.emit('score', score );

});

socket.on('chat message', function(msg)

{

io.emit('chat message', msg);

});

socket.on('disconnect',function()

{

socket.broadcast.emit('remove', player.id );

socket.broadcast.emit('chat message', player.message );

});

});

Euer nächstes Thema