

# Projektdefinition

“Mensch ärgere dich nicht”



Ein Projekt im Rahmen der Veranstaltung  
“Webbasierte Anwendungen 2: verteilte Systeme”

## Inhaltsverzeichnis

<b>1. PROJEKTDURCHFÜHRUNG</b>	<b>4</b>
1.1. WORKSHOP PROJEKTDEFINITION	4
1.2. SPEZIFIKATION DES REST WEBSERVICES	4
1.3. ERSTELLUNG VON USE-CASES	5
1.4. IMPLEMENTIERUNG DES DIENSTNUTZERS	6
1.5. REALISIERUNG DES PROJEKTES	7
1.5.1. WEITERE MODULE EINBEZIEHEN	7
1.5.2. WÜRFELLOGIK	8
1.5.3. SPIELLOGIK	8
<b>2. SPIELLOGIK DIENSTNUTZERSICHT</b>	<b>12</b>
<b>3. START DES PROJEKTES</b>	<b>13</b>

## “Mensch-ärgere-dich-nicht” Projektdefinition

In diesem Projekt zum Modul “Webbasierte Anwendungen 2: Verteilte Systeme” geht es um die Umsetzung eines webbasierten Systems, welches aus einem Dienstgeber, einem Dienstinutzer und einem Client (Browser) besteht und nach den REST Prinzipien gestaltet werden soll.

Unsere Gruppe besteht aus den Teammitgliedern Daniel Buchholz, Nico Ferdinand und Daniel Gofman. Wir haben uns für die Umsetzung des Brettspielklassikers “Mensch ärgere dich nicht” entschieden.

Ziel dieses Projektes ist es, ein webbasiertes System nach den REST Spezifikationen zu entwickeln, wobei die verwendeten Komponenten Dienstgeber, Dienstinutzer und Client mit verschiedenen node.js Modulen kommunizieren. Zur Zieleingrenzung sollte erwähnt werden, dass es keine Parallelspiele geben wird (Lobby-Prinzip). Es kann immer nur ein Spiel gleichzeitig gespielt werden und es können immer nur genau vier Spieler ein Spiel beginnen. Ein Spiel mit drei oder weniger Spielern wird nicht möglich sein. Über die Realisierung der Spielfindung und der wait-times bis das Spiel startet muss noch diskutiert werden. Es muss ein Modul gefunden werden, welches eine Kommunikation in beide Richtungen zulässt.

Im Verlauf dieses Projektes, werden einige Realisierungsschritte durchlaufen. Den Verlauf und die Entwicklung des Projektes kann im Github-Repository der Gruppe eingesehen werden: <http://bit.ly/28OjKQ3>. Dort werden alle Entwicklungsschritte und Quellcode, sowie Regeln des Spiels und Arbeitsmatrix festgehalten. Außerdem sind dort die Ergebnisse der einzelnen Workshops im Rahmen der WBA 2 Veranstaltung und die Use-Case Diagramme zu finden.

Das gesamte Projekt wird über das oben genannte Github-Repository und diesem Dokument in Google Docs organisiert. Dieses Dokument wird aktuell gehalten und beinhaltet den Projektfortschritt und Meilensteine. Außerdem beinhaltet es Probleme auf die wir während der Umsetzung des Projektes gestoßen sind, Lösungsvorschläge und die entsprechenden Entscheidungen mit Begründungen.

# 1. Projektdurchführung

In diesem Abschnitt der Projektdefinition werden alle Schritte festgehalten, die zur Erarbeitung der Workshop Termine erforderlich sind und Probleme und deren Lösungen die während der Erarbeitung aufgetreten sind.

## 1.1. Workshop Projektdefinition

Zum ersten Workshop sollte eine Definition des Projektes erfolgen, inkl. der zugehörigen Funktionalitäten. Wir entschieden uns nach kurzer Diskussion für das Projekt "Mensch-ärgere-dich-nicht", da wir etwas Graphisches realisieren wollen und nicht nur mit Formularen und ähnlichem arbeiten möchten. Das Ziel ist es hier, unser Projekt zu definieren, zu erklären und in diesem Falle die Regeln des Spiels aufzustellen. Probleme ergaben sich in diesem Teil des Projektes keine, da jeder mit der Idee des Spiels einverstanden waren und die Regeln bereits existieren, sodass es keinerlei Konflikte gab. Die Regeln wurden zunächst in einer PDF Datei festgehalten, später in eine JSON und eine HTML Datei geschrieben um sie auf dem Client anzeigen zu können.

## 1.2. Spezifikation des REST Webservices

Im nächsten Schritt, sollten für die Vorbereitung des REST Services, die verschiedenen benötigten Spezifikationen definiert werden. Dazu mussten Überlegungen zu den benötigten Ressourcen und deren Implementierung gemacht werden. Zunächst definierten wir die unterschiedlichen Ressourcen die vom Dienstgeber zum Dienstanwender übergeben werden müssen. Dazu gehörten unter anderem das Spielfeld an sich, die Figuren in vier unterschiedlichen Farben, ein Würfel und die Spielregeln. Die jeweiligen Spezifikationen können wie alle anderen erarbeiteten Ergebnisse hier im Github eingesehen werden: <http://bit.ly/293K9fc>. In diesem Dokument sind die verschiedenen Ressourcen und deren Implementierung hinterlegt, inkl. Semantik und content-types der Ressourcen. Zusätzlich sind dort auch die Methoden angegeben, wie diese Ressourcen gehandelt werden. Handelt es sich beispielsweise um den Würfelvorgang, so wird dies mit einem POST realisiert. Der Dienstanwender schickt eine Anfrage an den Dienstgeber, der eine Zahl für den jeweiligen Spieler ermittelt. Das Würfeln wird aus Sicherheits- und Fairnessgründen auf der Seite des Dienstgebers durchgeführt, da auf Seite des Dienstanwenders Manipulationen am Würfelvorgang vorgenommen werden könnten. Andere Ressourcen, wie z.B. der Spielzug werden überwiegend vom Dienstanwender angefordert, da die gesamte Spiellogik auf dem Dienstgeber realisiert wird. Der Dienstanwender dient als Kommunikationsbrücke zwischen Dienstgeber und dem Client

(Browser). Der Dienstnutzer rendert die vom Dienstgeber erhaltenen Ressourcen in eine für den Client lesbare HTML Datei und gibt diese entsprechend weiter.

In dieser Phase des Projektes kamen erste Spezifikationsprobleme auf. Teilweise war es schwierig eine geeignete Methode zur Behandlung der Ressource zu finden.

Außerdem stellten wir fest, dass einige Ressourcen doppelt erfasst wurden oder sich widersprachen. Des Weiteren stellten wir erste Probleme mit der Projektidee fest, da sich ein Spiel nur schwer über POST und GET Methoden realisieren lässt, da es z.B. auch eine Game-Loop geben muss, damit das Spiel nach jeder Runde fortgesetzt werden kann, oder, dass die Kommunikation zwischen Client und Dienstgeber nicht via POST und GET Methoden umgesetzt werden kann, da z.B. für den Spielstart eine Möglichkeit benötigt wird, dass der Dienstnutzer, ohne eine vorhergehende Anfrage des Clients, das Signal für den Spielstart schicken kann. Dies ist über die bis dato genutzten Module nicht möglich.

### 1.3. Erstellung von Use-Cases

Die nächste Aufgabe im Ablauf des Projektes bestand darin, Use-Cases für das Projekt zu erstellen, damit Abläufe im System deutlich werden, eventuelle Schwachstellen aufgedeckt werden und Zusammenhänge der einzelnen Funktionalitäten deutlich gemacht werden.

Zunächst wurde der Use-Case "Spielzug durchführen" erstellt, in dem es um die Realisierung eines einzelnen Zuges innerhalb des Spiels geht. Behandelt wird der reine Spielzug, ohne Abfragen nach Würfelhäufigkeit, das heißt, in diesem Diagramm fehlt die Auswertung ob ein Spieler ein Mal würfeln darf, was den Normalfall darstellt, oder ob ein Spieler bis zu drei Mal würfeln darf, im Fall, dass der Spieler keine Figur hat, die einen Spielzug durchführen kann und die anderen Figuren noch in der Basis stehen.

Im nächsten Use-Case ist die Anforderung des Regelwerks dargestellt. Der Fall tritt ein, wenn ein Benutzer bei Spielbeginn die Regeln nicht kennt oder wenn während des Spiels Unklarheiten bzgl. der Regeln auftreten. Hierbei handelt sich lediglich um die Anzeige der Regeln in HTML.

Der letzte Use-Case der von uns erstellt worden ist, ist der eigentliche Aufbau des Spiels, also das Spielfeld, die Figuren usw. Die Use-Cases können hier eingesehen werden: <http://bit.ly/28TvsJA>.

Auch in diesem Teil des Projektes zeige sich wieder, dass es Probleme gibt, da es nicht viele verschiedene Use-Cases gibt, da das Spiel nach dem Aufbau lediglich auf Spielzügen basiert und nahezu das gesamte Spiel auf dem Dienstgeber abläuft und während des Spielverlaufs lediglich die Positionen der Spieler aktualisiert werden und weitere Würfelwürfe ausgeführt werden.

Für den nächsten Workshop Termin soll mit der Implementierung des Dienstnutzers begonnen werden.

## 1.4. Implementierung des Dienstnutzers

Im nun folgenden Teil soll damit begonnen werden den Dienstnutzer zu implementieren. Lt. Wiki-Seite bzw. der Agenda des nächsten Workshop Termins, soll ein implementierter Use-Case vorgeführt werden. Wir haben uns für den Use-Case "Spielfeld aufbauen" entschieden, da der Use-Case zum anzeigen der Regeln zu simpel wäre und der Use-Case zum Spielzug eine Implementation des Spielfelds voraussetzt. Die einzelnen Stationen des Use-Cases haben wir in unterschiedliche Scribbles gefasst, um die Vorgänge auf Dienstgeber und Dienstnutzer planen zu können. Die Scribbles können hier eingesehen werden: <http://bit.ly/28TDHft>. Zunächst wird auf dem Dienstgeber auf neue Dienstnutzer gewartet. Sobald sich ein Dienstnutzer via GET verbindet bzw. einen Spieler auf dem Dienstgeber anlegt, wird auf dem Dienstgeber id++ ausgeführt und dem Dienstnutzer die Spieler-ID mitgeteilt. Außerdem wird jeweils den verbundenen Dienstnutzern die passenden Spielfiguren zugewiesen und diese inkl. des Spielfelds per Response zugeschickt. Der Dienstnutzer wird nun in einen Idle Status versetzt, aus dem heraus er alle 10 Sekunden ein GET an den Dienstgeber schickt um zu überprüfen, ob das Spiel gestartet werden kann. Der Dienstgeber antwortet entsprechend mit dem Spielstart oder nicht. Auf dem Dienstgeber läuft derweil die Abfrage ob bereits vier Spieler verbunden sind. Wenn ja wird an alle verbundenen Dienstnutzer das gesamte Spielfeld inkl. aller Figuren in deren Basis geschickt und das Spiel wird gestartet. Damit ist dieser Use-Case abgeschlossen und das Spiel kann beginnen.

Problematisch war an der Planung zunächst nur zu ermitteln, wie die Kommunikation zwischen Dienstnutzer und Dienstgeber stattfinden kann. Da der Dienstnutzer auf andere Spieler warten muss, der Dienstgeber aber nicht von sich aus mit dem Dienstnutzer kommunizieren kann, muss der Dienstnutzer selbst anfragen, ob das Spiel gestartet werden kann. Wir haben uns auf eine Zeit von 10 Sekunden geeinigt, da der Dienstgeber nicht dauerhaft mit dem Abarbeiten von inhaltslosen GET Methoden beschäftigt sein soll, es sollte aber auch nicht zu lange dauern, damit das Spiel zeitig beginnen kann und keine große Latenz zwischen den Startzeiten der Spieler entsteht. Wenn Spieler 1 z.B. eine Sekunde vor dem Verbinden von Spieler 4 anfragt ob das Spiel gestartet werden kann, muss dieser wieder 10 Sekunden warten, bis das Spiel bei ihm startet. Bei Spieler 4 hingegen startet das Spiel direkt, da der vom Dienstgeber direkt beim Anlegen seiner Spielfiguren das Spielfeld inkl. der positionierten Figuren zurück schickt. Diese Implementierung ist sehr rudimentär und basiert auf einer statischen HTML Seite, die Änderungen nur bei einem Refresh des Browsers anzeigt. Die Implementierung ist also vorhanden, ist jedoch so nicht umsetzbar, da jeder Spieler die Seite alle paar Sekunden neu laden müsste, damit Änderungen auf dem Spielfeld sichtbar werden.

## 1.5. Realisierung des Projektes

### 1.5.1. Weitere Module einbeziehen

An diesem Punkt des Projektes konnten wir mit den bisherigen Lösungsansätzen nicht weiterkommen. Eine reine Kommunikation via express.js kann nicht zwischen allen Komponenten entstehen, da eine beidseitige Kommunikation zwingend erforderlich ist und dem Client nach jeder Aktion eine HTML Datei geschickt werden muss, um die Änderungen sichtbar zu machen.

Um dieses Problem zu lösen, haben wir uns ejs, socket.io und faye angeschaut.

Nach Abwägung der Möglichkeiten haben wir uns für eine Umsetzung mit ejs und socket.io entschieden. Ejs wird an dieser Stelle im Dienstanbieter verwendet, um ein HTML-Template zu erstellen und im Anschluss zu rendern, welches vom Dienstgeber mit Daten gefüllt wird und an die eigentlichen Clients weitergegeben wird, damit am Ende im Browser keine statische HTML Seite erscheint, die nur bei einem refresh aktualisiert wird.

Für die Kommunikation zwischen Client und Dienstgeber greifen wir auf socket.io zurück und haben uns gegen faye entschieden, da die Kompatibilität von socket.io größer ist. Mobilgeräte, bzw. Mobilbrowser unterstützen Faye nicht komplett, socket.io hingegen funktioniert reibungslos auf allen Plattformen. Ein weiterer Vorteil von socket.io ist, dass es Abwärtskompatibel ist, das heißt, dass wenn bei älteren Browsern (IE 6/7 usw.) socket.io nicht unterstützt wird, es ältere Modelle gibt, die auf der gleichen Basis wie socket.io agieren (Websockets, Flashsockets, XHR Longpolling). Die älteren Technologien werden automatisch nach dem Fallback-Prinzip verwendet, wenn socket.io nicht unterstützt wird.

Nun kann damit begonnen werden, die verschiedenen Teile des Spiels umzusetzen. Zum einen die Anzeige und Positionierung der Spielfiguren inkl. der Würfel- und Spielzugesfrage, als auch die Spiellogik, bestehend aus Würfellogik (Anzahl der Würfelwürfe) und Validierung der vom Benutzer angeforderten Spielzüge.

### 1.5.2. Würfellogik

Im Folgenden gehen wir zunächst auf die Würfellogik ein, da es hier einige Sonderfälle zu beachten gibt. Zunächst sollte erwähnt werden, dass es eine unterschiedliche Anzahl an Würfelwürfen geben kann. Laut Regelwerk sind hier zunächst ein oder drei Würfe vorgesehen, je nach Zustand der Spielfiguren. Kann ein Spieler vor dem Wurf keinen Zug ausführen, das heißt es stehen alle Figuren in der Basis (im folgenden Home genannt) oder es stehen Figuren in Home und die restlichen sind bereits in ihrer finalen Position im Häuschen (im folgenden goal genannt) ist kein Spielzug möglich, unabhängig vom Ergebnis des Wurfes. Im goal muss zudem beachtet werden, dass alle sich im goal befindlichen Figuren die obersten Positionen belegen müssen, es reicht also nicht aus, wenn eine Figur auf der zweiten Position im goal steht, sie muss sich an oberster Stelle befinden, damit drei Würfe gewährt werden. Würfelt der Spieler eine 6 im ersten Versuch, darf er nur einen weiteren Wurf tätigen, außer es handelt sich bei seinem nächsten Wurf wieder um eine 6. In diesem Fall würfelt der Spieler so lange weiter, bis er keine 6 mehr würfelt. In allen anderen Fällen ist nach dem dritten Wurf, die Würfelrunde vorbei.

### 1.5.3. Spiellogik

Nun folgt die Implementation der Spiellogik, also die Validierung des vom Spieler ausgewählten Spielzugs.

Zuerst sollte erwähnt werden wie eine Runde innerhalb des gestarteten Spiels funktioniert. Der erste Spieler der sich verbindet, beginnt automatisch mit dem ersten Spielzug, die anderen Spieler kommen in der Reihenfolge an die Reihe, in der sie sich verbunden haben. Ein Spielzug wird pro Person immer dann beendet, wenn die Person gewürfelt und eine Figur auswählt die bewegt wird. Zusätzlich wird der Zug erst dann beendet, wenn der Zug gültig ist und ausgeführt werden kann. Darf ein Spieler mehrere Züge tätigen (wenn er eine 6 würfelt), wird die Runde erst nach dem nächsten Zug beendet, außer er würfelt wieder eine 6 usw. Hat ein Spieler einen Zug beendet und der Zug ist validiert und umgesetzt worden, wird ein Token an den nächsten Spieler weitergegeben. Dieser gibt den Token dann an den dritten, der an den vierten und der vierte schließlich wieder an den ersten Spieler. Hat der letzte Spieler das Token wieder an den ersten Spieler gegeben, beginnt eine neue Runde. Wie eine Spielrunde, bzw. der in der Runde auszuführende Zug validiert und umgesetzt wird, wird in den folgenden Unterkapiteln erklärt.



#### 1.5.3.1. Spiellogik "home"

Die Spiellogik von home wird für den Fall benötigt, in dem der Spieler eine 6 würfelt und eine Figur aus home auf das Spielfeld setzen möchte. Dieser Vorgang ist nicht in jedem Fall möglich. Befindet sich bereits eine Figur auf dem Startfeld vor home, darf der Spieler dort keine weitere Figur platzieren. Ist das Startfeld leer, kann der Spieler dort eine Figur platzieren.

#### 1.5.3.2. Spiellogik "Spielfeld"

Anschließend an den Fall aus Spiellogik "home" muss bei jedem bewegen der Figuren überprüft werden, ob das Zielfeld frei oder belegt ist und wenn es belegt ist, muss überprüft werden welcher Spieler dort mit welcher Figur steht, damit diese im Falle einer Ausführung des Zugs zurück in das home des Spielers gesetzt werden kann. Diese Validierung wurde über ein eindimensionales Array gelöst, welches unterschiedliche Zustände haben kann. Das Array besteht aus 40 Indizes, der Anzahl der Felder auf dem Spiel, ohne die vier goal Felder. Diese wurden absichtlich ausgelassen, da in diesem Array nun das Spielfeld mit allen für jeden Spieler zugänglichen Feldern abgebildet ist und nicht validiert werden muss ob ein Spieler, das angepeilte Feld betreten darf. Das Array an Stelle 0 steht für den Startpunkt vor home, an Stelle 39 steht man direkt vor dem Eingang zum goal. Für goal gibt es ein weiteres Array, welches je nach Spieler an das Spielfeld-Array angehängen wird, im Fall, dass ein Zug über den Index des Spielfeld-Array hinausgeht.

Die einzelnen Indizes des Spielfeld-Arrays werden zunächst alle mit dem Wert 0 gefüllt, das bedeutet, dass das keine Figur auf dieser Position steht. Wird nun eine Figur auf eines der Felder bewegt, ändert sich der Wert des jeweiligen Indizes. Zunächst war die Überlegung, die Werte 0-4 zu benutzen, 0 = frei, 1= Spieler 1, 2 = Spieler 2 usw. Doch im Falle einer Überlagerung, also wenn eine Figur an die Position einer gegnerischen bewegt werden soll, muss die ID der Spielfigur bekannt sein, damit diese zurück ins home des Gegners bewegt werden kann.

Wir entschieden uns also dafür die Zustände 0-16 zu definieren. 0 steht für ein freies Feld und 1-16 sind die jeweiligen Figuren (Figur 1-4 für Spieler 1, Figur 5-8 für Spieler 2, usw.).

Dazu nun ein kleiner Einschub. Es gibt für den home Bereich des Spielfeldes ein weiteres Array, in dem gespeichert ist, ob ein Feld in home frei oder belegt ist. Dies dient zur Ermittlung ob ein Spieler ein oder drei Mal würfeln darf. Anstatt für jeden Spieler ein eigenes home-Array zu erstellen, erstellten wir nur ein home-Array für alle. Der Vorteil ist, dass es eindimensional bleiben kann, da neben dem Zustand des Feldes die Spieler-ID nicht pro Feld mitgespeichert werden muss. Wir haben also ein Array mit 16 Einträgen. Die home-Felder werden jedem Spieler zugeordnet. Spieler 0 hat die Indizes 1-4, Spieler 1 die von 5-8, Spieler 2 die von 9-12 und Spieler 3 die von

13-16. Die passenden Felder können automatisch mit einer Funktion ermittelt werden. Der Startindex jedes Spielers liegt bei "SpielerID\*4+1" und der Letzte Index liegt bei "SpielerID\*4+4". Hat ein Spieler die ID "2" ergibt sich also ein Startindex von  $2*4+1 = 9$  und ein Endindex von  $2*4+4 = 12$ . So kann jedem Spieler automatisch sein home zugeordnet werden.

Zurück zum Spielfeld-Array. Wie gerade erklärt, bekommt jede Figur bereits eine ID zugesprochen. Diese IDs werden einfach als Wert in den Index des jeweiligen Spielfeld-Feldes geschrieben. Steht z.B. von Spieler 3 die erste Figur auf Position 14 steht im Spielfeld-Array im Index 13 die Zahl 13. Das Array kann also Werte von 0 für nicht besetzt, bis 16, vierte Figur von Spieler 3 annehmen. Dadurch kann auch direkt ermittelt werden, ob es sich um eine eigene Spielfigur oder um die eines Gegners handelt. Die Platzierung der Figuren wird später behandelt.

#### 1.5.3.3. Spiellogik "Spielzug"

Ein Spielzug kann nur dann ausgeführt werden, wenn die Position der ausgewählten Figur ermittelt wird. Nachdem diese ermittelt wurde wird überprüft ob sich kein Spieler auf der aktuellen Position + Würfelwurf befindet, ein gegnerischer Spieler befindet oder eine eigene Figur. Tritt einer der ersten beiden Fälle ein, wird die neue Position auf aktuelle Position + Würfelwurf gesetzt und die alte Position wird auf 0 (=frei) gesetzt. Tritt der letzte Fall ein, darf die Figur nicht bewegt werden.

#### 1.5.3.4. Spiellogik „rausschmeißen“

Um gegnerische Spieler „rauszuschmeißen“, also zurück in ihr Home zurückzuschicken erfordert die Implementierung einiger Sonderfälle und mehrerer Abfragen.

Als erstes muss zum rausschmeißen einer Figur auf dem normalen Spielfeld die Zielposition der ausgewählten Figur ermittelt werden, um zu überprüfen ob das Zielfeld frei, von eigener oder fremder Figur besetzt ist. Dazu wird die aktuelle Position der Figur ausgelesen und der gewürfelte Wert addiert und das Spielfeldarray an der Stelle ausgelesen. Danach wird zwischen den drei genannten Fällen differenziert. Steht dort nun eine fremde Figur wird die Position mit der eigenen ID überschrieben und der Wert der zuvor dort stand zurück in das passende Home geschrieben.

Ein Sonderfall ist es, wenn eine Figur beim verlassen des Homes rausgeschmissen wird, denn dort gibt es nicht die Möglichkeit über die aktuelle Position plus Würfelwurf das Zielfeld zu bestimmen, also muss die Herkunft der Figur zuvor ermittelt werden. Dazu gibt es die Möglichkeit eine Figur zu „suchen“. Dabei wird

vom Dienstinutzer eine Anfrage mit einer FigurID gesendet. Der Dienstgeber sucht diese Figur nun zunächst auf dem Spielfeld (da dieser Fall am häufigsten Eintritt). Wird die Figur gefunden, wird die Position zurückgegeben (was auch zur visuellen Positionierung der Figuren benötigt wird). Wird die Figur nicht gefunden, wird eine 40 zurückgegeben (außerhalb des Spielfeldes). Danach wird die Figur in Home und Goal gesucht. Dadurch wird ermittelt, dass die angeklickte Figur im Home steht und mit  $\text{playerID} * 10$  wird das passende Startfeld im Spielfeldarray ermittelt (für Spieler 2 z.B.  $2 * 10 = 20$ ).

Ein weiterer Sonderfall ist es, wenn eine Figur am Übergang des Spielfeldarrays rausgeschmissen wird. Steht die Zielfigur auf Feld 2 und die ausführende Figur auf Feld 38 wird die Figur einen Übergang des Arrayendes durchlaufen und muss komplett neu positioniert werden. Das wird über eine Hilfsvariable gelöst. Die Felder die man noch in der aktuellen Runde gehen kann werden vom Würfelwurf abgezogen und die restlichen Züge in der neuen Runde gelaufen.

#### 1.5.3.5. Spiellogik „Goal“

Das Goal ist ein sehr komplexer Teil der Logik, da es auch hier einige Sonderfälle zu beachten gibt. Zunächst zum Eintritt in das eigene Goal. Um in das eigene Goal einzutreten wird die aktuelle und die Zielposition der ausgewählten Figur ermittelt. Wird bei diesem Zug das  $\text{playerID} * 10 - 1$  Feld überschritten (also das Feld vor dem Startfeld) wird vom Würfelwurf die Züge die noch im normalen Spielfeldarray zu gehen sind abgezogen, der Rest wird zwischengespeichert. Nun wird die Position in ein neues Array, das Goalarray, platziert. Dazu wird der eigene Bereich im gemeinsamen Goalarray mit  $\text{playerID} * 4$  ermittelt. Der Bereich des eigenen Goals liegt bei  $\text{playerID} * 4$  bis  $\text{playerID} * 4 + 3$  (für Spieler 0  $0 * 4 = 0$  bis  $0 * 4 + 3 = 3$ ). Die Figur wird nun wie gewohnt platziert.

Da man im Goal keine anderen Figuren überspringen kann erfolgt eine weitere Abfrage für den Weg der Figur. Jedes Feld welches die Figur zur Zielposition im Goal überschreitet muss in der Abfrage 0, also leer sein, damit die Figur bewegt wird. Außerdem kann das Goal nicht betreten werden wenn die restlichen Züge  $> 4$  sind, da man in dem Falle die Grenzen des eigenen Goals überschreiten würde.

Ein kleiner Exkurs zur Würfellogik. Auch dazu gibt es hier mehrere Sonderfälle, da man drei Mal würfeln darf wenn:

- Das letzte Feld im Goal besetzt und alle anderen Figuren im Home sind,
- Die letzten beiden Felder im Goal besetzt und die anderen Figuren im Home sind oder
- Die letzten drei Felder im Goal besetzt sind und die letzte Figur im Home ist

Gibt es eine Lücke zwischen den Figuren im Goal, darf nur einmal gewürfelt werden.

Für alle Züge die im Goal oder beim Eintritt in das Goal betätigt werden, muss die Logik getrennt implementiert werden, da die Berechnung Grundlegend anders ist. Zu erwähnen ist hier auch noch die Gewinnerfunktion. Sind alle eigenen Spieler im Goal platziert, hat der aktuelle Spieler gewonnen und das Spiel ist vorbei.

## 2. Spiellogik Dienstnutzersicht

Der Dienstnutzer dient in dieser Architektur zum Verwalten von den Spielern. Während der Dienstgeber an sich die Spiellogik verwaltet und das Würfeln absichert, nimmt der Dienstnutzer neue Verbindungen von Clients (Webbrowser) entgegen und teilt diesen eine eindeutige und bis zum Spielende oder –Abbruch andauernde ID zu. Diese SpielerID wird bei Spielzügen (die vom Client abgeschickt werden) vom Dienstnutzer an den Dienstgeber weitergeleitet damit dieser damit weiterarbeiten und korrekte Daten zurückliefern kann. Der Dienstnutzer verlässt sich somit auf die Rückgabewerte des Dienstgebers und ist eine Art Mittelsmann zwischen Client und Dienstgeber. Intern basiert das Rundensystem auf einem Token der nach jeder Runde an den nächsten Client weitergereicht wird. Ist die Runde vom letzten Spieler beendet, wird der Token wieder an den ersten Spieler weitergegeben.

Bei den Runden ist der Dienstnutzer ebenfalls dafür verantwortlich diese zu beenden bzw. die nächste Runde zu starten. Die Informationen dafür errechnet sich der Dienstnutzer durch die Daten die durch den Dienstgeber bereitgestellt wurden. Beispielsweise schickt der Dienstnutzer eine Anfrage an den Dienstgeber ob ein Spielzug möglich ist und wenn diese Möglichkeit besteht, dann führt der Dienstnutzer auf dem Dienstgeber über einen HTTP Request diesen Spielzug durch und schickt ein Signal zum Client dass sich die angeklickte Figur bewegen kann. Nun muss der Dienstgeber aber noch prüfen ob die Runde zu Ende ist oder ob zuletzt eine sechs gewürfelt wurde. Abhängig davon wird entweder die Runde beendet oder nicht.

Um einen Spielzug auf dem Client durchzuführen muss vorher einmal gewürfelt werden. Nach dem Würfeln werden erst die eigenen Figuren freigeschaltet damit man diese anklicken kann. Wenn nun eine eigene Spielfigur angeklickt wurde wird vom Client ein sogenannter „movewish“ über socket.io an den Dienstnutzer geschickt. Dieser leitet die Anfrage weiter an den Dienstgeber, der dann den Spielzug prüft. Wenn der Spielzug möglich ist, gibt der Dienstgeber je nach Situation einen anderen Rückgabewert. Beispielsweise kann ein Feld entweder frei sein oder auch mit einer gegnerischen Figur besetzt sein, in beiden Fällen wäre ein Spielzug möglich. Wenn nun der jeweilige Rückgabewert zurückkommt wird der Dienstnutzer den „movewish“ stattdessen und somit ein Signal an den Client zurückschicken und dazu auffordern die angeklickte Figur zu bewegen. Der Client rendert am Ende dann nur noch die Spielfigur an der neuen Position.

Nach jeder Runde fragt der Dienstnutzer beim Dienstgeber ab, ob alle eigenen Figuren schon im Ziel sind. Sind alle Figuren eines Spielers am Ende einer Runde im Ziel, wird vom Dienstnutzer ein „gamewon“ Signal an den Client geschickt, der dann eine Meldung anzeigt, welcher Spieler gewonnen hat.

### 3. Start des Projektes

1. Das Projekt wird mit verschiedenen Schritten gestartet:
2. Navigieren Sie zu dem Ordner „Abgabe\_06\_09\_2016“ unseres Github-Repository (<http://bit.ly/29mkLla>)
3. Den Server starten (node app.js)
4. Den Dienstnutzer starten (node client/dienstnutzer.js)
5. Im Browser (nach Wahl) localhost:3001 eingeben und bestätigen (für mehrere Spieler weitere Tabs öffnen oder statt localhost:3001 die IP des Hosts eingeben (Ports 3000 und 3001 müssen freigegeben werden))
6. Spiel über den Button „Spiel starten“ starten

### 4. Fazit

Ein Großteil der von uns genannten Kriterien und Funktionalitäten wurden von uns umgesetzt. Es bestehen weiterhin kleinere Fehlfunktionen in bestimmten Situationen, die von uns nicht fristgerecht gefunden und behoben werden konnten. Dazu gehört die nicht immer korrekte Anzahl an Würfeln und dass der Gewinner erst eine Runde später feststeht. Weiterhin fehlt der letzte Feinschliff der Anwendung. Es gibt einige Codedopplungen und nicht konsequent eingehaltene Namenskonventionen innerhalb des Quellcodes. Ebenso gibt es einen Punkt in der Anwenderfreundlichkeit bei dem es einer Optimierung bedarf. Eine Runde muss bei einem nicht durchführbaren Zug durch den Anwender selbst beendet werden, da für eine automatische Beendigung der Runde, der Dienstgeber immer auswerten muss, ob eine Zug möglich ist oder nicht. Diese Implementation ist uns auf Grund von Zeitmangel nicht gelungen und wurde von uns ausgelassen.

Die Ermittlung der Würfelanzahl beinhaltet sehr viele Sonderfälle die in den verschiedensten Spiellagen zusammengefasst und ausgewertet werden müssen.

Der von uns als Optionales Feature geplante Chat innerhalb des Spiels konnte ebenfalls auf Grund von zu wenig Zeit nicht umgesetzt werden.

Zusammenfassend lässt sich sagen, dass ein spielbares Ergebnis vorliegt, welches alle Grundlegenden Elemente und Logiken funktionstüchtig beinhaltet. Einige kleinere Anforderungen wurden nicht erfüllt, das Ergebnis ist trotzdem, mit kleineren, sich nicht auf das Spiel auswirkenden, Fehlern behaftet. Einzig die Dokumentation ist zu kurz geraten, da eine chronologische Protokollierung unserer Arbeit fehlt und die Ergebnisse aus dem Gedächtnisprotokoll geschrieben worden sind.

Eine genauere Absprache für Namenskonventionen wie z.B. die Verwendung von englischen oder deutschen Variablennamen, würde den Quellcode professioneller wirken lassen. Auch ein organisierterer Umgang mit Git, hätte weniger Aufwand mit sich gebracht.