Dokumentation über Monopoly

1. Technische Umsetzung

Im Folgenden werden die wichtigsten technischen Merkmale von Monopoly zusammengefasst. Dies umfasst verwendete Programmiersprachen sowie verwendete Bibliotheken von Dritten.

1.1. TypeScript

Monopoly ist ein Spiel, das für Browser konzipiert wurde. Zwangsläufig muss daher das Projekt in HTML/CSS und JavaScript umgesetzt werden. Alternativen wie Flash oder Java-Applets kommen aufgrund von mangelnder Kompatibilität und Sicherheitsrisiken nicht in Frage.
Für Monopoly haben wir jedoch trotzdem beschlossen, das Spiel nicht in JavaScript, sondern in einer davon abgeleiteten Sprache, TypeScript, zu entwickeln. TypeScript wird nicht direkt im Browser ausgeführt, sondern während der Entwicklungsphase in JavaScript umgewandelt. Daraus ergeben sich verschiedene Vorteile:

- Es wird für größere Kompatibilität gesorgt, da der Compiler den Quellcode in JavaScript einer älteren Version (z.B. ES5) kompilieren kann, welcher von Browsern ausgeführt werden kann, die nicht besonders aktuell sind. Dieser Vorteil ist jedoch in unserem Fall nicht entscheidend, da Monopoly auf Technologie wie WebGL basiert, welche noch recht neu sind. Die Features von neuen JavaScript-Versionen wie ES6 (z.B. Klassen, Arrow functions), welche auch in TypeScript vorhanden sind, sind dann meist auch verfügbar.
- Ein weiterer Vorteil ergibt sich aus der möglichen strikten Typisierung von TypeScript. In normalem JavaScript werden Variablen ohne Angabe von Typen deklariert, genau wie Parameter und Rückgabewerte von Funktionen. Eine Überprüfung kann es daher nicht geben, ob eine Zuweisung korrekt ist. Dadurch können unerwartete Fehler entstehen, die erst aufwendig gesucht werden müssen. Der TypeScript-Compiler dagegen findet solche Fehler. Langfristig lässt sich dadurch der Programmieraufwand verringern.
- Generell gilt, dass JavaScript-Code vom TypeScript-Compiler kompiliert werden kann. Bereits vorhandener Code kann so weiterverwendet werden.

Der TypeScript-Quellcode befindet sich in *js/* und zusätzliche Typdefinitionen in *typings/index.d.ts.* Weitere Dokumentation kann man unter https://monopoly.ctdev.de/doc finden.

1.2. three.js

Three.js ist ein in JavaScript geschriebenes Framework, mit dem 3D-Szenen leicht im Browser dargestellt werden können. Wir benutzen es, um den visuellen Teil des Spiels bereitzustellen. Three.js selbst basiert auf WebGL, einem plattformübergreifendem Standard zur hardwarebeschleunigten Darstellung von 3D-Grafiken. Da WebGL noch ein eher neuer Standard ist und Browserhersteller

Features teilweise sehr verzögert implementieren, sind sehr moderne Versionen von Browsern erforderlich um Monopoly zu spielen. Erfolgreich getestet haben wir Monopoly mit Chrome 54, Firefox 50.1 und Chromium 55, wobei wir eine aktuelle Version von Chrome empfehlen. Ältere Versionen der Browser sowie andere Browser könnten funktionieren, jedoch können wir Darstellungs- und sonstige Fehler nicht ausschließen.
Zusätzlich nutzen wir in Monopoly noch einige Erweiterungen für three.js, um beispielsweise OBJ-Modelle, die wir u.a. für die Spielfigur und die Häuser nutzen, zu laden.

1.3. AngularJS

AngularJS ist ein weit verbreitetes Framework zur Darstellung von Websites. Damit wird es möglich gemacht, HTML und Programmcode miteinander zu kombinieren. Es ist möglich in HTML-Tags Ausdrücke wie diesen einzubauen:

```
<p> {{ text }} </p>
```

text ist hier ein JavaScript-Ausdruck und damit eine Variable, die einfach durch das Programm geändert werden kann. Sollte sie geändert werden, wird der neue Wert automatisch auf der Website sichtbar, was eine gewaltige Menge an Arbeit spart, da man sonst im JavaScript manuell das Element, in dem man etwas ändern möchte, suchen und dann bearbeiten muss. Analog dazu gibt es neue HTML-Attribute, die es beispielsweise ermöglichen per Variable ein Objekt anzuzeigen und auszublenden. In diesem konkreten Fall wäre das die Direktive *ng-show*. Die im HTML genutzten Variablen befinden sich alle in einem Objekt, das *scope* genannt wird.
In unserem Fall fiel die Entscheidung, AngluarJS zu verwenden, da abzusehen war, dass der Programmcode viel mit der Benutzeroberfläche interagieren muss.

1.4. Git / GitLab

Mit der eigentlichen Struktur von Monopoly hat Git nichts zu tun, es ist jedoch wichtiger Teil des Entwicklungsprozesses. Git ist ein sehr weit verbreitetes Versionsverwaltungssystem, dessen Ursprung im Linux-Kernel liegt. Heute wird es jedoch auch in unzähligen anderen Open-Source-Projekten und kommerziellen Produkten verwendet. Aufgrund der in früheren Projekten gesammelten Erfahrung mit Git haben wir uns dazu entschieden es auch in unserem Projekt zu verwenden. Git ermöglicht es, seine Arbeit in kleine Teile zu unterteilen, die *Commits.* Dadurch wird es möglich zu jedem vorherigen Stand des Projektes zurückzukehren und parallel mit mehreren Personen am Projekt zu arbeiten, da parallel erzeugte Commits zusammengeführt werden können. Diesen Vorgang nennt man *mergen.* Daraus hat sich natürlich ein großer Vorteil für das Projekt ergeben, da wir mehrere Personen sind.
GitLab ist eine Plattform, die Projekte, die mit Git verwaltet werden, online verwaltbar macht. Commits können vom lokalen Rechner dort hin hochgeladen werden und die Commits der anderen Entwickler können heruntergeladen werden. Zusätzlich kann man über diese Plattform kommunizieren, was sich in anderen

Projekten als sehr nützlich erwiesen hat. Eine weitere Funktion ist automatisches Deployment. Dadurch wird jedes Mal, wenn jemand Commits hoch lädt, ein Skript ausgeführt *(siehe .gitlab-ci.yml),* das das Programm auf einen Webserver hoch lädt, der das Spiel unter https://monopoly.ctdev.de/ verfügbar macht. Ohne zusätzlichen Arbeitsaufwand befindet sich also immer eine aktuelle Version auf der Website.

2. Erstellen einer nutzbaren Monopoly-Instanz

Im Folgenden wird beschrieben, wie der Quellcode von Monopoly aus dem Git-Repository kompiliert wird und von einem Webserver verfügbar gemacht wird. Getestet wurde dieses Vorgehen nur auf Linux-Systemen, aber es sollte problemlos auf anderen System wie Windows funktionieren.

2.1. Voraussetzungen

Voraussetzungen zum Kompilieren sind:

- Node.js 7.0
- npm

Node.js ist hierbei eine JavaScript-Plattform, die es ermöglicht ohne Browser JavaScript-Programme auszuführen, was zum kompilieren benötigt wird *(siehe build2.js)*.
npm ist ein Paketmanager für Node.js-Module; dazu zählen auch die genannten Module TypeScript, Angular.JS und three.js. npm lädt diese Module automatisch anhand einer Liste *(siehe package.json)* herunter.

2.2. Kompilieren des Quellcodes

Folgende Befehle installieren die Abhängigkeiten des Projektes und kompilieren es:

```
$ npm install
$ node build2
```

Danach sollte sich in *public/bundle.js* das erzeugte JavaScript befinden.

2.3. Bereitstellung durch einen Webserver

An den Webserver werden keinerlei spezielle Anforderungen gestellt. Er muss nur in der Lage sein, statische Dateien bereitzustellen, welche sich in *public/* befinden. Dort befindet sich auch eine index.html.