

Programmierung von astronomischen Apps

Besondere Lernleistung im Abitur 2024

vorgelegt von

Dominik Fuchs

vom

Carl-Duisberg-Gymnasium Wuppertal

geprüft am

Carl-Fuhlrott-Gymnasium Wuppertal

angefertigt bis

12.04.2024

Dominik Fuchs

Unterschrift des Schülers

Inhaltsverzeichnis

Einleitung.....	1
Kontakt und Zugang.....	1
Hauptteil	3
Auswahl der Technologien	3
Plattform.....	3
Webtechnologien	4
Weitere Technologien.....	7
Sternenkarte.....	11
Zielsetzung.....	11
Funktionen.....	11
Ansatz	12
Entwicklung und Probleme.....	12
Einbindung der Karten.....	12
Rotation der Karten	13
Interaktivität durch Ziehen	14
Einfügen eines zusätzlichen Ringes	16
Platzierung weiterer Elemente	17
Platzierung von Planeten.....	20
Dynamische Skalierung.....	21
Simulation des Zeitlaufes (abgebrochen)	25
Sonnenkuppel.....	26
Zielsetzung.....	27
Ansatz	27
Entwicklung und Probleme.....	27
Grundeinstellungen und Erstellung der Basis.....	28
Kamerakontroller.....	29
Berechnung der Sonnenposition	30
Erstellung des Obeliskens.....	31
User Interface	32
Darstellung des Polarkoordinatensystems	33
Weitere Kleinigkeiten	35
Zusätze.....	35
CI/CD.....	35
Bereitstellung (Electron und Docker)	38
Fazit	41

Anhang	43
Danksagung	43
Anmerkungen	43
Links	43
Abbildungsverzeichnis	45
Abbildungsquellen	46
Quellen	46
Anlage: „Erklärung“:	47

Einleitung

Diese Besondere Lernleistung ist im Anschluss an den Astronomie Projektkurs der Q1 vom Schülerlabor Astronomie entstanden. Die dazugehörige Projektarbeit kann zur Vollständigkeit und für weitere Informationen über das Projekt hier abgerufen werden: <https://bit.ly/3SqWUGO>. Es ist wichtig zu beachten, dass diese Besondere Lernleistung **keine** Fortführung der Projektarbeit ist, sondern eine neue separate Arbeit. Diese Besondere Lernleistung wird in derselben Codebase¹ wie die Projektarbeit entwickelt, wodurch viele Konfigurationen und Boilerplate-Code² eingespart werden. Das ermöglicht das schnelle Warten und eine vereinfachte Verbreitung der Software, da so nur ein Projekt gehostet³ werden muss. Der gesamte Quellcode und die verfassten Arbeiten werden unter folgendem Link der Öffentlichkeit als **Open-Source** Projekt unter der MIT Lizenz zur Verfügung gestellt (<https://github.com/domx4q/astro-Project>). Unter diesem GitHub Link kann der Versionsverlauf eingesehen werden (mögliche Bewertungsgrundlage). Dadurch wird ersichtlich, welche Änderungen nach Abschluss des Astronomie Projektkurses eingegangen sind. Dabei ist zu beachten, dass die Sternenkarte zwar im Zeitraum des Projektkurses teils angefertigt wurde, jedoch nicht zu diesem gehört hat und somit auch nicht in der Projektarbeit erwähnt wurde. Weitere Hinweise und eine kurze Anleitung zur Nachverfolgung der Änderungen sind ebenfalls in der README.md⁴ Datei im Startverzeichnis des Projektes auf GitHub zu finden. Die Veröffentlichung unter Open-Source Richtlinien ermöglicht es anderen Entwicklern oder Astronomie Enthusiasten Einblicke in die Funktionsweise zu bekommen und den Code für deren eigenes Projekt zu nutzen oder individuelle Anpassungen vorzunehmen. Außerdem kann man so sichergehen, dass die App sicher ist und die Privatsphäre berücksichtigt wird (viele Onlinedienste verkaufen die Nutzerdaten, um Profit zu generieren). Zusätzlich ermöglicht es anderen Entwicklern, Änderungen und Optimierungen vorzuschlagen, welche ich nach sorgfältigem Prüfen akzeptieren oder ablehnen kann, wodurch die Anwendung deutlich schneller wächst. Bei der Bewertung ist zu beachten, dass der Praxisteil dieser Besonderen Lernleistung deutlich mehr Zeit als diese schriftliche Ausarbeitung in Anspruch genommen hat.

Vor der weiteren Lektüre empfehle ich Ihnen den Abschnitt *Anmerkungen* zu lesen.

Kontakt und Zugang

Bei Fragen, Anmerkungen oder weiteren Anliegen können Sie mich unter (ev3dom@gmx.de) erreichen.

Die Anwendung wird aufgrund von Redundanz, damit diese immer abrufbar ist, bei mehreren Providern gehostet. Sie können demnach (im Optimalfall) über jeden der folgenden Links die App verwenden. Falls die Seite dennoch nicht erreichbar ist, können Sie die anderen Links probieren.

Basis-Linkadresse	Beschreibung
https://cloudster.online/astro/	Mein eigener V-Server (evtl. fällt dieser in der Zukunft aufgrund von Kosten weg.)
https://astro-project-pi.vercel.app/	Vercel als kostenloser Provider
https://domx4q.github.io/astroProject/	GitHub Pages als kostenloser Provider

¹ Die Codebase bezeichnet in der Softwareentwicklung die gesamte Sammlung an Quellcode Dateien, Bibliotheken und Ressourcen, welche zu einem Projekt gehören.

² Code ‚Schnipsel‘ welche häufig verwendet werden und dabei kaum oder gar nicht abgeändert werden, bezeichnet man als Boilerplate-Code.

³ Unter Hosting versteht man die Veröffentlichung eines Dienstes wie einer App, API oder Webseite mithilfe eines Servers.

⁴ GitHub und manch andere Onlinedienste verwenden eine Datei namens „README.md“. Diese ist im Markdown Format und wird auf der Startseite eines Projektes direkt formatiert angezeigt.

Mit diesen Links können Sie während des Lesens genannte Funktionen ausprobieren und so ein tieferes Verständnis für diese entwickeln.

Mit den Linkadressen kommen Sie jedoch nur auf die Anwendung, welche ich im Rahmen des Astronomie Projektkurses entwickelt habe. Um auf die neuen/aktuellen Teile des Projektes zugreifen zu können, müssen Sie die zuvor aufgelisteten Links noch verändern. Dazu hilft Ihnen folgende Tabelle.

Subpath⁵	Beschreibung
/	Hauptseite; Planetenapp (Projektkurs)
/stars	Sternenkarte (BesLL)
/sun	Sonnenkuppel (BesLL)
/download	Weiterleitung zum neusten Build ⁶ der App

Es gibt noch weitere Seiten (meist kleinere Spielereien/= „Easter Eggs“), da während der in Summe zweijährigen Entwicklung, dieses Projekt meinen (einzigen) V-Server in Anspruch genommen hat, wodurch kleinere Tests oder Spielereien in diesem Projekt verwirklicht wurden. Um alle Pfade und Alias⁷ zu verstehen, können Sie sich diese Datei anschauen (<https://bit.ly/48GEw2n>).

Um nun diese Links zusammenzufügen, müssen Sie den Subpath an den Basislink anhängen. Im Falle der Sternenkarte, kombiniert mit meinem V-Server, sähe das dann wie folgt aus: <https://cloudster.online/astro/stars>

Die Anwendung steht nicht nur in Form einer Webseite zur Verfügung, sie kann auch offline entweder über die integrierte PWA („Progressive Web App“) oder als Desktop App für Linux oder Windows heruntergeladen werden. Um die PWA zu nutzen, müssen Sie nur die Webseite im Google Chrome öffnen. Daraufhin werden Sie gefragt, ob Sie die App installieren möchten. Diese PWA funktioniert nur Semi-Offline. Das heißt, die meisten Teile der App funktionieren offline, jedoch brauchen Sie hin und wieder eine aktive Internetverbindung, da sonst die Cache⁸ dafür sorgt, dass manche Bilder oder Ressourcen nicht mehr offline zur Verfügung stehen. Die Desktop App ist daher empfehlenswerter, jedoch müssen Sie diese, wenn ein Update zur Verfügung steht, manuell aktualisieren. (Das heißt im Detail: Sie werden von der App aufgefordert, eine neue Version zu installieren. Die App liefert direkt den passenden Download Link für die neue Version. Die heruntergeladene Datei muss dann nur noch ausgeführt werden.) Über diesen Link können Sie sich die Desktop App herunterladen: <https://github.com/domx4q/astro-Project/releases/latest>

⁵ Ein Subpath (auf Deutsch „Unterpfad“) beschreibt zunächst, einen Unterpfad eines Verzeichnisses. Da Webseiten ursprünglich nur aus Quelldateien mit einer Indexdatei bestanden, welche auf andere Dateien verwies, hat sich die Webentwicklung dieses Prinzip der Dateiverwaltung zunutze gemacht und übernommen. Demnach bedeutet ein Subpath einfach nur eine Unterseite, also eine andere Seite als die Hauptseite.

⁶ Ein Build bezeichnet in der Softwareentwicklung die Kompilierung des Quellcodes zu einer (für den Verbraucher) ausführbaren Datei. Beim Kompilieren wird der für den Programmierer verständliche Code zu einem für den Computer verständlichen Code umgewandelt.

⁷ In diesem Kontext bedeutet ein Alias eine alternative Bezeichnung oder in diesem Fall Route, so führen beispielsweise / oder /home zur gleichen Seite oder /extra/stars zum gleichen Ziel wie /stars. Diese alternativen Möglichkeiten erleichtern es dem Nutzer, falls dieser nicht mehr die genau URL kennt, die richtige Seite aufzurufen.

⁸ Der Cache ist der Zwischenspeicher vom Browser (oder anderen Programmen). Manche Regeln sorgen dafür, dass Inhalte nicht unbegrenzt lang verwendet werden dürfen. Nach einer festgelegten Zeit wird der lokale Inhalt gelöscht und neuer Inhalt angefordert. Dabei ist eine Internetverbindung notwendig.

Hauptteil

Auswahl der Technologien

Zu Beginn der Projektarbeit musste ich mich entscheiden, wie ich die App aufbaue, welche Technologien ich dafür am besten verwende und für welche Plattformen ich die App bereitstelle. Diesen Entscheidungsprozess werde ich im folgenden Kapitel erläutern.

Plattform

Als Erstes ist es wichtig in der Anwendungsentwicklung festzulegen, welche Ziele die Anwendung hat und über welche Plattformen der Nutzer diese nutzen kann. Häufig muss man sich für eine einzelne Plattform entscheiden, wie Desktop oder Mobile. Dann muss man bei der Nativen-Entwicklung⁹ das Betriebssystem auswählen, d.h. soll die App auf Android, Linux, iOS, macOS oder auf Windows ausgeführt werden. Native-Softwareentwicklung bietet viele Vorteile im Vergleich zur „**Cross-Platform**“¹⁰ Entwicklung, da so die individuellen Funktionen des jeweiligen Gerätes direkt angesprochen und in die App integriert werden können. Dadurch wird die Benutzung für den Anwender einfacher und die Anwendung wirkt „polierter“¹¹. Andererseits kann die Anwendung dann nur auf diesem System genutzt werden. Wenn der Entwickler nun die Anwendung auch noch für andere Systeme verfügbar machen möchte, bedeutet das deutlich mehr Arbeit, da meist die gesamte App noch einmal neu implementiert¹² werden muss. Deshalb setzt nahezu jede moderne Anwendung auf das Cross-Platform Modell. Dieses ermöglicht die einmalige Implementation der App, sodass diese auf vielen oder sogar allen (aktuellen) Systemen läuft. Dieses Modell ermöglicht meist keinen Zugriff auf systemabhängige Funktionen. Deshalb muss man zuvor abwägen, ob das eigene Projekt mit diesem Modell erfolgreich umgesetzt werden kann.

Damit die Einsetzbarkeit der Anwendung so groß wie möglich ist, entschied ich mich für die Entwicklung im Cross-Platform Modell.

Da dieser Entschluss nun gesetzt war, musste ich mir als nächstes überlegen, auf welcher Basis ich die App aufbauen werde. Infrage kam die richtige Anwendungsentwicklung mit Qt, Flutter oder anderen Alternativen oder die Webentwicklung. An dieser Stelle ist mir ein Kurzvortrag einer meiner Kollegen eingefallen, indem er mir erläutert hat, dass die Webentwicklung mittlerweile in Hinblick auf die Anwendungsentwicklung fast immer eine bessere Wahl ist als die richtige Anwendungsentwicklung. Im Anschluss hat er mir seinen eigenen kleinen PWA-Test vorgeführt. Das Fazit seines Vortrags war, dass solange man nicht auf die Gerätefunktionen, den Gerätespeicher oder weiteren tieferen Zugang zum Gerät benötigt (welcher durch eine installierte App gegeben wäre), eignet sich eine Webapp mit Service-Worker¹³ besser.

Die Ursprungszielsetzung der App war das Visualisieren von Planeten mithilfe von 3D Modellen. Das Rendern¹⁴ von 3D Modellen wäre als lokale Anwendung leichter gewesen, da ich in dem Falle direkten Zugriff auf die Grafikkarte hätte, doch durch Technologien wie WebGL und Three.js¹⁵ war mir bewusst, dass das auch im Browser möglich ist. Speicher oder Gerätezugriff waren nicht notwendig und das

⁹ Bei der Nativen-Entwicklung entscheidet man sich bewusst nur für ein Betriebssystem, auf welches die Anwendung dann spezialisiert/zugeschnitten wird. Diese Anwendung funktioniert dann auch nur auf diesem System und auf keinen anderen Systemen.

¹⁰ Plattformübergreifend (kann auf verschiedenen Systemen genutzt werden)

¹¹ In diesem Kontext: aufgeräumter, besser, moderner

¹² Die Realisierung/Umsetzung eines Programmes. (Das eigentliche programmieren)

¹³ Der Service-Worker ist ein fundamentaler Teil der PWA. Dieser ermöglicht die Offline-Ausführung der App.

¹⁴ Berechnen und Darstellen von Bildern oder 3D Inhalten.

¹⁵ Three.js ist eine JavaScript Bibliothek, welche das Darstellen von 3D Inhalten im Browser ermöglicht.

Updaten der App ist im Browser auch deutlich einfacher, da ich einfach nur die Inhalte auf meinem Server austauschen muss. Aus diesen Gründen habe ich mich im Rahmen dieses Projekt für die Webentwicklung entschieden.

Webtechnologien

Wie Ihnen mittlerweile vielleicht schon aufgefallen ist, versucht man in der Informatik, Probleme in immer kleinere Probleme zu unterteilen und diese nacheinander zu lösen. Der Aufbau eines Projektes wird genauso gehandhabt. Wenn man den Prozess visualisieren würde, würde man eine ziemlich umfangreiche Baumstruktur erhalten (hiermit ist die Datenstruktur Baum, kein natürlicher Baum gemeint). Diese Herangehensweise ermöglicht es, größere Probleme einfacher und schneller zu beheben. In diesem Sinne folgt der nächste Unterpunkt (Knoten).

Nun musste ich entscheiden, auf welcher Technologie die Webseite basieren soll. Ich hätte es mit reinem HTML5, JavaScript und CSS versuchen können, doch da ich mich für diesen Weg bereits einmal entschieden hatte (bei meinem eigenen einfachem Cloud Provider), weiß ich wie aufwendig es ist, die Webseite nur mit JavaScript ausreichend interaktiv zu machen. Außerdem wusste ich bereits, dass ich eine Bibliothek für das 3D rendern benötigen würde, somit bräuchte ich einen Package-Manager¹⁶ wie **npm** oder **bun**. Da diese beide auf **Node.js**¹⁷ basieren, hätte ich auch Node direkt in mein Projekt integriert, wodurch es keinen Sinn mehr macht auf eine Pure-JS (reines JavaScript, kein Framework) Methode zu setzen. Ein JavaScript-Framework hat viele Vorteile. Zum einen muss ich deutlich weniger Code schreiben, wodurch mir das Leben deutlich erleichtert wird. Dieser Code wird kompiliert, gebündelt, minimiert und auf grobe Fehler geprüft. Nach diesen Schritten kann ich diesen Code direkt über ein Node Modul namens „**serve**“ als Server hosten, also fällt der Schritt des Einrichtens eines statischen HTTP-Servers ebenfalls weg (ein Beispiel wäre „Apache“). Durch solch ein Framework läuft der Code und dadurch auch die App meist deutlich schneller, als wenn man die App vollständig manuell schreiben würde. Die Auswahl des Frameworks fiel mir nicht schwer. Da ich schon bei der Arbeit und auch privat mit „**Vue.js**“ gearbeitet hatte, habe ich dieses erneut ausgewählt. Vue ist zwar etwas langsamer als andere Frameworks wie „React“ oder „express“, doch da Vue eine deutlich bessere Entwicklererfahrung bietet also eine einfachere und angenehmere Entwicklung der Seite, bevorzuge ich Vue. Ein großer Vorteil von Vue ist die „Reaktivität“ von Variablen¹⁸. Das bedeutet, wenn ich eine Variable wie einen Text (String) oder eine Zahl (in JS Number) dem Nutzer anzeige, dann wird die Anzeige aktualisiert und mit dem neuen Wert ersetzt, sobald ich an irgendeiner Stelle in meinem Code diese Variable ändere. Ein anderes Beispiel sind sogenannte „**computed properties**“. Diese stellen ebenfalls eine Variable dar, doch wenn eine Abhängigkeit von dieser geändert wird, werden diese automatisch neu berechnet. Ein Beispiel dafür wäre: Der Nutzer kann eine Temperatur in Grad Celsius eingeben. Diese speichere ich mir automatisch durch ein „v-model“ in einer Variable. Sobald eine Änderung (also in diesem Fall eine Eingabe des Nutzers) eingeht, möchte ich die daraus berechneten Werte geändert anzeigen. Dann ‚erwähne‘ ich die Variablen mit den berechneten Werten (diese können auch als computed property verwendet werden) einfach im HTML-Template¹⁹ Code, um diese anzuzeigen. Doch jetzt habe ich ein Problem. Alle meine Formeln benötigen die Temperatur in Grad Fahrenheit und nicht in Grad Celsius. Ich könnte nun alle Formeln anpassen, um die Temperatur umzurechnen, doch es ist viel einfacher und sinnvoller, dafür eine computed property zu verwenden, auf welche ich dann in meinen Formeln zugreife. Diese würde dann „tempInFahrenheit“ heißen. Da dieser Wert abhängig von der

¹⁶ Ein Package-Manager ist ein Programm, welches die Nutzung und Verwaltung von Bibliotheken erlaubt.

¹⁷ Node.js erlaubt das Ausführen von JavaScript außerhalb von Webseiten (beispielsweise als Server).

¹⁸ In der Informatik wird ein Datenspeicher als Variable bezeichnet. Diese kann (solange nicht typisiert) jeden Datentyp annehmen.

¹⁹ Das HTML-Template gibt den Aufbau der Webseite an, welcher durch logische Verzweigungen und Schleifen verändert werden kann.

Variable „temp“ ist, wird dieser umgehend geändert, wodurch die davon abhängigen Variablen auch umgehend geändert werden.

Vue umfasst noch zwei weitere (theoretisch optionale) Bibliotheken. Diese wären „**vuex**“ und „**vue-router**“. Mit theoretisch optional meine ich, dass Vue auch ohne diese verwendet werden könnte, doch in nahezu jedem Projekt werden diese ebenfalls benutzt; „vue-router“ noch häufiger als „vuex“, da es für „vuex“ Alternativen wie „pinia“ gibt. „Vuex“ ist ein Session-Storage Management System. Dieses ermöglicht es, Informationen in der momentanen Sitzung zu speichern. Doch durch Schließen der Seite wird auch die Sitzung geschlossen, wodurch normalerweise diese Daten verloren gehen. Um das zu verhindern, habe ich mir einen kurzen Code geschrieben, welcher, sobald das Event²⁰ zum Schließen der Seite kommt, diese Daten unmittelbar in die Local-Storage²¹ speichert, sodass diese beim erneuten Betreten der Seite wieder geladen werden können. Ich habe hierfür die Local-Storage gewählt, da diese neben Cookies am längsten hält, jedoch deutlich größere Datenmengen als Cookies speichern kann. Diese Persistenz ist wichtig für die Speicherung des ausgewählten Themes (hell oder dunkel oder automatisch), damit der Nutzer dieses nicht jedes Mal neu einstellen muss. Zusätzlich speichere ich noch einige Client-Informationen mit der Bibliothek „**clientjs**“. Diese werden nur lokal auf dem Gerät erhoben und nirgendwo hin übermittelt (Thema: Datenschutz) und dienen nur für eine bessere Nutzererfahrung. Durch diese Informationen erfahre ich, ob der Nutzer am Handy oder Desktop ist und wie groß sein Bildschirm ist. Diese Informationen sind wichtig, um die Seite für die verschiedenen Geräte richtig darzustellen. Um alle erhobenen Informationen einzusehen, können Sie hier im Quellcode nachlesen (<https://bit.ly/3UcVVLG>).

„Vue-Router“ ist für dieses Projekt sehr wichtig, da diese Bibliothek die verschiedenen Seiten und Apps miteinander verbindet. Wenn man Vue in vollen Zügen anwenden möchte, dann nutzt man Vue als eine Single-Page App (SPA)²², d.h. für den Browser ist die Seite, egal welche Unterseite man besucht, immer die gleiche und zählt nur als eine Seite. Die Seite muss nie neu laden, da jegliche Logik, auch das Laden von anderen Inhalten/Seiten, von Vue erledigt wird. Dadurch muss die Seite keine neuen Inhalte laden und bleibt somit sehr schnell und flüssig. Das Laden/Rendern verschiedener Seiten wird von dem „vue-router“ ermöglicht. Dieser rendert den Inhalt der jeweiligen Seiten, doch jedes Element außerhalb des Router-Elements bleibt unverändert. Das kann man sich wie einen Fernseher in einem Raum vorstellen. Die Webseite ist der Raum und anstatt in einen neuen Raum zu gehen, um eine andere Seite zu öffnen, wechselt man einfach den Sender im Fernseher. Somit bleibt der Raum gleich, doch man sieht trotzdem neue Inhalte. Der Fernseher entspricht in diesem Vergleich dem Router-Element. In dieser App ist momentan (*Stand: Ende Januar*) noch ein neu laden der Seite durch die Änderung der URL nötig, da keine Router-Links vorhanden sind, welche diese Aufgabe übernehmen würden. Das ist beabsichtigt, da während des Astronomie Projektkurses die weiteren Seiten nicht zum Projekt gehörten und somit nur die eigentliche Anwendung direkt sichtbar war. Wenn ich mit allem fertig bin und dann noch Zeit verfügbar ist, werde ich noch eine Übersichtsseite erstellen, von der man jede integrierte Seite aufrufen kann. Die fehlenden Links stellen kein Problem dar, da der Router zusätzlich die Möglichkeit aufweist, dass man verschiedenen Subpaths (siehe *Kontakt und Zugang*) verschiedene Seiten zuordnet.

Um die App auch offline nutzbar zu machen, habe ich mich für zwei weitere Abhängigkeiten entschieden (wie bereits unter *Kontakt und Zugang* erwähnt). Dabei handelt es sich um „**register-service-**

²⁰ In der Webentwicklung und auch manchmal in der Anwendungsentwicklung, läuft alles über Events. Diese übermitteln Informationen, so wie Signale vom Nervensystem im menschlichen Körper.

²¹ Der Browser ermöglicht Webseiten das Speichern von Daten durch Session-, Cache-, Indexed-, Local-Storage oder Cookies. Die Dauer der Persistenz dieser Methoden ist hier aufsteigend sortiert, d.h. die Session hält am kürzesten und die Cookies am längsten.

²² Quelle: (01) <https://vuejs.org/guide/extras/ways-of-using-vue>

worker“ und „electron“. Das Modul „register-service-worker“ ist von Google und erlaubt die Nutzung der App als PWA. Genauere Informationen zu der Funktionsweise eines Service-Workers finden Sie in meiner Astronomie Projektarbeit im Abschnitt Grundlagen zur App.

Electron ermöglicht die Plattformübergreifende (nur für Desktop Systeme) Anwendungsentwicklung in Form der Webentwicklung. Das ermöglicht die einfache Übernahme von Webseiten in die Form der nativen Anwendung. Da die App durch Electron auf der eigenen Maschine läuft, startet diese deutlich schneller und benötigt keine aktive Internetverbindung. Diese verbreiteten Anwendungen basieren auch alle auf Electron (Abb.1). Der einzige Nachteil an Electron ist die Speicherauslastung, da jede Electron Instanz²³ intern einen Chrome-Driver ausführt und dieser ziemlich viel Arbeitsspeicher beansprucht. *Abbildung 1 - Electron basierende Anwendungen*



Bei der normalen Nutzung sorgt diese Auslastung nicht für Probleme, doch wenn man viele verschiedene Electron basierende Programme ausführt und jedes dieser Programme eine eigene Chrome Instanz realisiert, dann kann der Arbeitsspeicher (RAM) schnell stark oder sogar voll ausgelastet werden.

Die letzte wichtige Bibliothek ist „unity-webgl“. Der Name WebGL ist eine Kombination aus Website und OpenGL. OpenGL ist eine programmiersprachenunabhängige Schnittstelle zur Darstellung von Grafiken. Der Fokus von OpenGL liegt dabei auf Geschwindigkeit, wodurch es sich gut für das Rendern von 3D-Objekten und Grafiken eignet. Für die 2D Umgebung gibt es viele andere schnelle Alternativen, weswegen sich OpenGL auf die Darstellung in 3D fokussiert. Unity ist eine Spiel-Engine, welche das Erstellen und Ausführen von Spielen deutlich verbessert hat. Neben Unity gibt es noch andere populäre Spiel-Engines wie Godot, PyGame oder Unreal Engine. Ich habe mich im Rahmen dieser Besonderen Lernleistung für Unity entschieden, da es für die Größe und Komplexität des Projektes ideal geeignet ist. PyGame kann nur lokal mittels Python oder per Build ausgeführt werden. Da ich den Rest der App bereits als Webseite entwickelt hatte, wollte ich das hier fortführen, wodurch PyGame ausschied. Godot ist eine vielseitige Spiele-Engine, jedoch eher für abstraktere und einfachere Spiele gedacht. Da ich eine akkurate Schattensimulation für die Sonnenkuppel brauche, ist Godot auch keine Option. Mit Unreal Engine kann man sehr realistische Spiele erstellen, in denen man kaum noch den Unterschied zwischen Realität und Videospiel erkennt, jedoch benötigt die Ausführung solch eines Spieles sehr viel Rechenleistung, welche mir im Browser nicht zur Verfügung steht. Da Unity schnell läuft, die Entwicklung stark vereinfacht, realistische Schatten berechnen kann und eine Build Option für den Browser über WebGL bietet, ist es die perfekte Wahl. Die Bibliothek „unity-webgl“ ermöglicht nun die Einbettung der Unity Build-Artefakte²⁴/des Spieles im Browser.

Die zuvor genannten Technologien, sind die wichtigsten Webtechnologien. Für eine vollständige Liste können Sie sich diese Datei anschauen: <https://bit.ly/3vW1e8j>. Die ausgelassenen Bibliotheken sind in diesem Zusammenhang zu vernachlässigen und verändern nur geringe Aspekte der Anwendung. Um diesen Abschnitt übersichtlicher zu machen, sind alle in diesem Projekt verwendete Abhängigkeiten in diesem Abschnitt fett hervorgehoben.

²³ Eine Instanz ist die Realisierung eines Objektes, in diesem Fall einfach ein weiteres Programm.

²⁴ Das Resultat eines Build Prozesses, also die generierten Dateien, bezeichnet man als Build-Artefakt.

Weitere Technologien

Die zuvor genannten Technologien beschäftigen sich alle mit den Inhalten der App und gehören somit alle zum Front-End²⁵. Das Besondere an dieser App ist, dass es eigentlich eine reine Front-End Anwendung ist. Das Einzige, was der Server/Back-End erledigen muss, ist es, die Webseite und benötigte Ressourcen wie Bilder an den Client²⁶ zu senden. In den meisten Anwendungen gibt es zusätzlich ein Back-End, welches Prozesse wie Authentifizierung, Übermitteln von Ergebnissen aus einer Datenbank oder das Anlegen von Daten übernimmt. All diese Aktionen werden hier nicht benötigt, wodurch die App sehr sicher ist.

Der Nutzer/Angreifer kann zwar die App bei sich selbst lokal im Front-End ändern oder manipulieren, jedoch betreffen diese Änderung dann nur ihn selbst und keine anderen. Da diese App keine Möglichkeit hat, Zugang zum Server zu bekommen, wie sonst beispielsweise durch eine API oder andere Methoden, werden keine (neuen) Schwachstellen durch diese App geöffnet. Deswegen müssen sich Betreiber oder Privatanwender keine Sorgen machen, wenn sie diese App hosten,²⁷ da keine Zugriffsmöglichkeit zum Server besteht, wodurch die Sicherheit gewährleistet ist.

Im Laufe der Zeit hat sich herausgestellt, dass das einfache Ausführen des „serve“ Befehls auf meinem Server keine gute Idee war, da es häufig zu Problemen und Abstürzen geführt hat. Im Nachhinein weiß ich nun, dass diese Probleme auf das veraltete Betriebssystem des V-Servers zurückzuführen waren, jedoch ließ sich dieses nur durch eine vollständige Neuinstallation des Servers aktualisieren. Aufgrund dieser Probleme habe ich mich für eine robustere Lösung entschieden und bin dabei auf **Docker** gestoßen.

Docker ist eine Virtualisierungsplattform, welche dem Betriebssystem das unabhängige Ausführen von Diensten erlaubt. Bei der normalen Ausführung von Programmen und Diensten wird auf die betriebssystemabhängigen Ressourcen wie DLLs²⁸ oder andere Tools wie Bibliotheken gesetzt. Diese werden von dem entsprechenden Programm direkt angesteuert, um Berechnungen oder andere Operationen auszuführen. Diese Vorgehensweise kann jedoch zu Problemen führen. Denn so können manche Programme oder Dienste nur unter bestimmten Plattformen ausgeführt werden. Im besten Falle sorgt man dafür, dass dieser Dienst plattformunabhängig läuft. Dabei hilft Docker, indem es die benötigten Ressourcen direkt in Kombination mit dem Dienst bereitstellt. Im sogenannten Container-Modell werden die verschiedenen Dienste voneinander separiert. Dabei bekommt jeder Container die für die Ausführung benötigten Ressourcen mitgeliefert, wodurch dieser unabhängig vom restlichen System läuft. Das ermöglicht eine zusätzlich erhöhte Sicherheit. Falls nun ein Dienst eine Sicherheitslücke aufweist, wodurch Angreifern der Zugriff auf diesen gelingt, können sie nur den entsprechenden Dienst manipulieren, da die anderen Dienste und das Host-System isoliert von diesem ausgeführt werden. Vollständige Isolation mag in manchen Fällen erwünscht sein, doch dann kann dieser Dienst auch nur wenig bewirken oder produzieren. Deswegen erlaubt Docker die Freigabe ausgewählter Ressourcen.

²⁵ Das gesamte Front-End ist dem Nutzer zugänglich. Es stellt die Präsentation/alles Visuelle des Dienstes dar. Das Gegenteil ist das Back-End, dort findet die eigentliche Logik und weitere Sicherheitsrelevante Operationen statt. Dieses ist nur für die Entwickler zugänglich.

²⁶ Im Server-Client Modell repräsentiert der Client den Empfänger. Dieser nimmt die Dienste eines Servers in Anspruch. In dem Fall der Webseite ist der Client der Browser des Nutzers.

²⁷ Das Betreiben/Veröffentlichen eines Dienstes in diesem Fall einer Webapp.

²⁸ DLL steht für Dynamic Link Library. Diese Dateien beinhalten Programmbibliotheken, welche von mehreren Prozessen simultan verwendet werden können. Die DLLs sind typisch für Windows.

Die typischen Freigaben sind Port²⁹-Freigaben und Dateisystemfreigaben. Die Port-Freigabe benötige ich zwingend in diesem Projekt, da ich die App über einen Server als Web-App publiziere. Docker erlaubt auch noch das „mappen“ von Freigaben. Dadurch können beispielsweise viele Dienste auf einmal den gleichen Port z.B. 443 freigeben, welcher dann von Docker auf verschiedene Host-Ports übertragen wird. Das verhindert die in der Fußnote angesprochenen Kollisionen. Das gleiche System gilt auch für die Dateisystemfreigabe. Dabei wird ein „mount“ Verzeichnis festgelegt. Der Container darf dann nur auf dieses und keine höher liegenden Verzeichnisse zugreifen. Zusätzlich können hier auch noch die Berechtigungen festgelegt werden, ob der Container nur den Inhalt lesen oder auch schreiben darf.

Um einen Docker Container zu erstellen, muss zunächst ein Docker Image erstellt werden. Dieses legt die Abhängigkeiten fest. Es wird in eine [Dockerfile](#) geschrieben, wodurch nach Fertigstellung mittels des „docker build“ Befehls das Image erstellt wird. Zuerst wird das Basis Image festgelegt (in meinem Fall **Ubuntu** Version 23.10). Von dem Punkt an existiert eine funktionierende Linux Maschine, auf der die bekannte Distribution Ubuntu ausgeführt wird. Danach werden Konfigurationen wie Umgebungsvariablen und Docker spezifische Konfigurationen wie das Arbeitsverzeichnis und der Einstiegspunkt festgelegt. Danach werden alle benötigten Bibliotheken und Abhängigkeiten über verschiedene Befehle installiert und zum Schluss wird die App/der Dienst in dieses Image kopiert (in diesem Fall über git). Wenn die Erstellung des Images abgeschlossen ist, kann dieses gleich verwendet werden. Die schnelle und einfachere Möglichkeit wäre in dem Fall der „docker run“ Befehl, dabei werden die benötigten Parameter für die Konfiguration und der Name des Images übergeben. Die andere Möglichkeit wäre die Docker-Compose Variante. Diese Variante habe ich für meine Besondere Lernleistung gewählt, da es so einfacher ist, Änderungen vorzunehmen und die App leichter zu verbreiten (falls jemand diese selbst hosten möchte). Im Folgenden werde ich den Inhalt meiner [docker-compose.yml](#) Datei auflisten und erklären, um die oben genannten Prinzipien nochmal besser erläutern zu können.

```
1  version: "3.3"
2  services:
3    astro:
4      container_name: astro_vue
5      ports:
6        - "3511:3000"
7      environment:
8        - TZ=Europe/Berlin
9        - customPrefix=/astro/
10     volumes:
11       - "/home/cloudster/certificates/cert.crt:/opt/certs/cert.crt:ro"
12       - "/home/cloudster/certificates/cloudster.online.key:/opt/certs/cert.key:ro"
13     image: domx4q/astro:autoGit
14     tty: true # -t flag
15     stdin_open: true # -i flag
16     restart: always
17
18     ulimits:
19       nproc: 65535
```

Abbildung 2 - Inhalt: docker-compose.yml

²⁹ Ein Computer/Server kann mehrere Dienste gleichzeitig bereitstellen. Damit es keine Kollisionen im Netzwerkverkehr gibt, stehen dem Gerät Ports bis 65535 zur Verfügung. Manche dieser Ports sind für bestimmte Zwecke reserviert und können nicht frei genutzt werden, doch die meisten stehen zur freien Auswahl zur Verfügung. Ein Port (pro Gerät) kann immer nur von einem Prozess gleichzeitig beansprucht werden.

In Zeile 1 gebe ich die Version der Docker-Compose Datei an. Diese verwendet Docker, um festzustellen, welchen Syntax ich benutze, da sich dieser im Laufe der Zeit verändert hat. Generell wird empfohlen mindestens die Version 2.0 oder höher zu verwenden, da die Version 1.x nicht mehr aktualisiert wird.³⁰ Zeile 2 ist für den korrekten Syntax der Datei erforderlich. In den folgenden (eingerückten) Zeilen, werden die Dienste, welche zu diesem Stack³¹ gehören, festgelegt. In Zeile 3 erstelle ich einen Container namens „astro“. Bei der Betrachtung einer YAML-Datei ist zu beachten, dass zugehörige ‚Kinder‘ durch die Einrückung zum ‚Elternteil‘ festgelegt werden. Ein Dienst („service“) ermöglicht die Verwaltung geteilter Ressourcen. Für jeden Stack wird ein eigenes internes Netzwerk erstellt, wodurch die Container dieses Stacks sicher und isoliert miteinander kommunizieren können. Es gibt außerdem die Möglichkeit, geteilte namentliche Volumes zu erstellen. Diese ermöglichen den Dateiaustausch, sind jedoch nicht über einen Mount-Point beim Host eingebunden. In meinem Fall genügt ein einzelner Container, da die App nicht groß genug ist, damit eine Normalisierung durch Aufteilung nötig oder sinnvoll wäre. Es werden häufig mehrere Container pro Stack/Dienst verwendet, wenn eine zusätzliche Datenbank im Spiel ist. Diese wird dann durch einen separaten Container realisiert. Zeile 4 legt den Container Namen fest. Diese Zeile ist optional, jedoch wird sonst eine Kombination aus Stack Namen und Service Namen und einem zufälligen Suffix³² gewählt. Deswegen sorgt die manuelle Definition des Namens für eine bessere Übersicht. In den Zeilen 5 und 6 lege ich die Port-Freigaben fest. Dabei gebe ich den Container-Port 3000, auf dem der Webserver läuft, an den Host-Port 3511 weiter. Von Zeile 7 bis 9 definiere ich die Umgebungsvariablen. Dabei muss die Zeitzone für Ubuntu festgelegt werden und ich lege zusätzlich einen customPrefix für meine App fest. Normalerweise kann der customPrefix weggelassen werden, doch da ich die App hinter einem Reverse-Proxy namens **Nginx** laufen lasse, wobei ich die App über den Port 443 also den normalen https Port und den Subpath /astro/ veröffentliche, muss ich diese Option angeben, da sonst der Vue-Router durchgehend Fehler wirft, weil die URL /astro/* nicht definiert ist. In den Zeilen 10 bis 12 lege ich die Dateisystemfreigaben fest. Diese Zeilen können ebenfalls weggelassen werden. Dann generiert die App ein neues SSL-Zertifikat, doch in dem Falle warnt der Browser vor dem selbstsignierten Zertifikat. Da ich jedoch ein SSL-Zertifikat für die Domain „cloudster.online“ besitze, kann ich dieses direkt übergeben, um die App und den Nutzer zu schützen. In den meisten Fällen ‚verlinkt‘ man nur ein Host und ein Container Verzeichnis und keine einzelnen Dateien, jedoch ist der Datei-Ansatz hier besser geeignet, da ich die App universell gehalten habe, wodurch diese nach den Dateien „cert.crt“ und „cert.key“ in dem entsprechenden Verzeichnis sucht. Jedoch habe ich die Dateien auf meinem Host-System in anderen Anwendungen bereits unter anderem Namen eingebunden und kann diese deswegen nicht umbenennen. Durch die Verlinkung der Dateien kann ich dieses Problem umgehen. Wichtig an den Zeilen 11 und 12 ist das „:ro“ am Ende. Dieses steht für „read only“, wodurch der Container die Dateien nur einsehen aber nicht verändern kann. Das schützt meine Zertifikate vor einer eventuellen Fehlfunktion, welche diese überschreiben könnte. Die Zeile 13 ist sicherlich die wichtigste Zeile, da diese das zu verwendende Docker Image angibt. Das geschieht in diesem Format: <Name des Erstellers>/<Name des Images>:<Version>. Die Version „autoGit“ steht bei mir für ein automatisch aktualisiertes Image (durch GitHub Actions), wenn sich Docker betreffende Inhalte ändern. Das Image muss lediglich bei großen Änderungen aktualisiert werden, da der Inhalt der App durch ein internes Skript alle 5 Minuten, sofern Änderungen vorhanden sind, aktualisiert wird. Die Zeilen 14 und 15 sind für eine interaktive Umgebung zuständig. In den meisten Docker Containern braucht man diese nicht, doch da ich mehrere Prozesse für das Aktualisieren mithilfe von „screen“ laufen lasse, brauche ich einen interaktiven Container. Zeile 16 sorgt dafür, dass die App immer läuft. Wenn diese

³⁰ Quelle: (02) <https://docs.docker.com/compose/compose-file/>

³¹ Die Docker-Compose Datei erlaubt die Erstellung von Stacks. Dadurch können ganze Systeme auf einmal eingerichtet werden, welche aus dutzenden von Containern bestehen können.

³² Eine Endung, welche den Basis String erweitert.

abstürzt, Docker neu startet oder andere Ereignisse auftreten, in jedem Fall startet die App wieder. Die letzten beiden Zeilen sind eher optional, doch da ich Probleme mit der Prozessanzahl auf meinem Server hatte, wodurch dieser die ausgeführten Prozesse pro Container limitiert, umgehe ich diese Limitierung mit den Zeilen.

Durch diese Umsetzung mittels Docker, konnte ich mich im weiteren Verlauf der BesLL/des Projektes rein auf die eigentliche Entwicklung konzentrieren.

Als letzte weitere Technologie möchte ich noch Nginx aufführen. Nginx ist ein Reverse-Proxy. Das bedeutet, die Clients steuern nicht direkt den (eigentlichen) Server an, sondern Nginx, welcher dann entscheidet, an welchen Server diese Anfrage weitergeleitet wird und ob diese Anfrage weitergeleitet werden soll

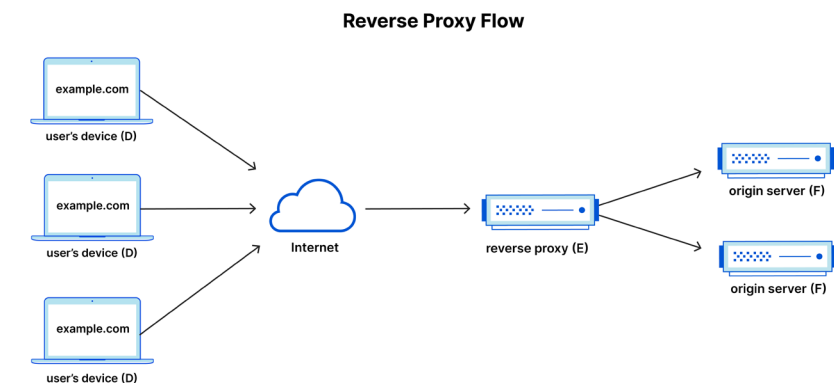


Abbildung 3 - Reverse-Proxy Modell

(siehe Abb.3). Nginx wird bei 44% der 10.000 größten Webseiten verwendet. Der größte Vorteil im Vergleich zu anderen Konkurrenten ist die Geschwindigkeit. Die Verarbeitung der Anfragen läuft innerhalb weniger Millisekunden ab, wodurch Nginx nicht nur für kleine, sondern auch für größere Projekte geeignet ist. Häufig wird Nginx als Load-Balancer verwendet (wie in Abb.3), wobei Nginx anhand von unterschiedlichen (konfigurierbaren) Kriterien den am besten/schnellsten/leersten Server auswählt. Da ich in meinem Fall nur einen Server nutze, hat das Load-Balancer Feature für mich keinen Vorteil. Ich verwende Nginx für die bessere Bereitstellung verschiedener Dienste. Bei meiner Domain „cloudster.online“ bekomme ich ein SSL-Zertifikat direkt mitgeliefert, jedoch ist dieses kein Wildcard-Zertifikat, welches für jede Subdomain also „xxx.cloudster.online“ (xxx durch beliebige Werte ersetzbar) gilt. Es erlaubt mir nur die Nutzung für „www.cloudster.online“ und „cloudster.online“. Da man normalerweise Subdomains für die Differenzierung verschiedener Dienste nutzt (beispielsweise: „astro.cloudster.online“), ich diese aber nicht über ein SSL-Zertifikat verschlüsseln darf, fällt diese Methode weg. Deswegen nutze ich Nginx, um für die verschiedenen Dienste verschiedene Subpaths zu verwenden, wie im Abschnitt *Kontakt und Zugang* genanntes Beispiel: <https://cloudster.online/astro/>. Dadurch muss ich zum einen in meiner Firewall³³ keine weiteren Ports freigeben (für jeden neuen Dienst) und ich kann auch reine HTTP-Anwendungen, also Anwendungen, welche keine TLS-Verschlüsselung³⁴ anbieten, über mein SSL-Zertifikat verschlüsseln. Zusätzlich nutze ich Nginx noch für eine weitere Sicherheitsebene, zum Beispiel im Download Bereich. Sie können beispielsweise Inhalte über diesen Link (<https://cloudster.online/download/astro/>) herunterladen, doch für diesen Link (<https://cloudster.online/download/secure/>) brauchen Sie die passenden Zugangsdaten. Die entsprechende Nginx Konfigurationsdatei werde ich im Rahmen dieser Besonderen Lernleistung aus Sicherheitsgründen nicht veröffentlichen.

³³ Eine Firewall kontrolliert, von wo und wohin Netzwerkanfragen ausgeführt werden dürfen.

³⁴ TLS ist der Nachfolger von SSL, auch wenn es immer noch SSL-Zertifikat heißt, obwohl dieses mittlerweile auf TLS basiert.

Sternenkarte

Die Idee der digitalen Sternenkarte ist während des Astronomie Projektkurses am CFG entstanden. Da wir im Unterricht solch eine Sternenkarte auf Papier selbst gebastelt und dann auch verwendet haben. Dachte ich mir, dass diese als App sicherlich einfacher zu benutzen wäre und auch mobiler wäre. Daraufhin habe ich diese Idee meinem Astronomielehrer Bernd Koch vorgestellt. Diese hat ihm sehr gefallen, woraufhin er mich bei der Erstellung der Sternenkarte unterstützt hat. Die ursprüngliche Version auf Papier wurde von den Schülerinnen Pauline Post und Pauline Plaster entworfen. Deren Arbeit kann über diesen Link (<https://bit.ly/3HMYPQ0>) eingesehen werden.

Sobald man die [Sternenkarte](#) öffnet, erscheint diese Ansicht:

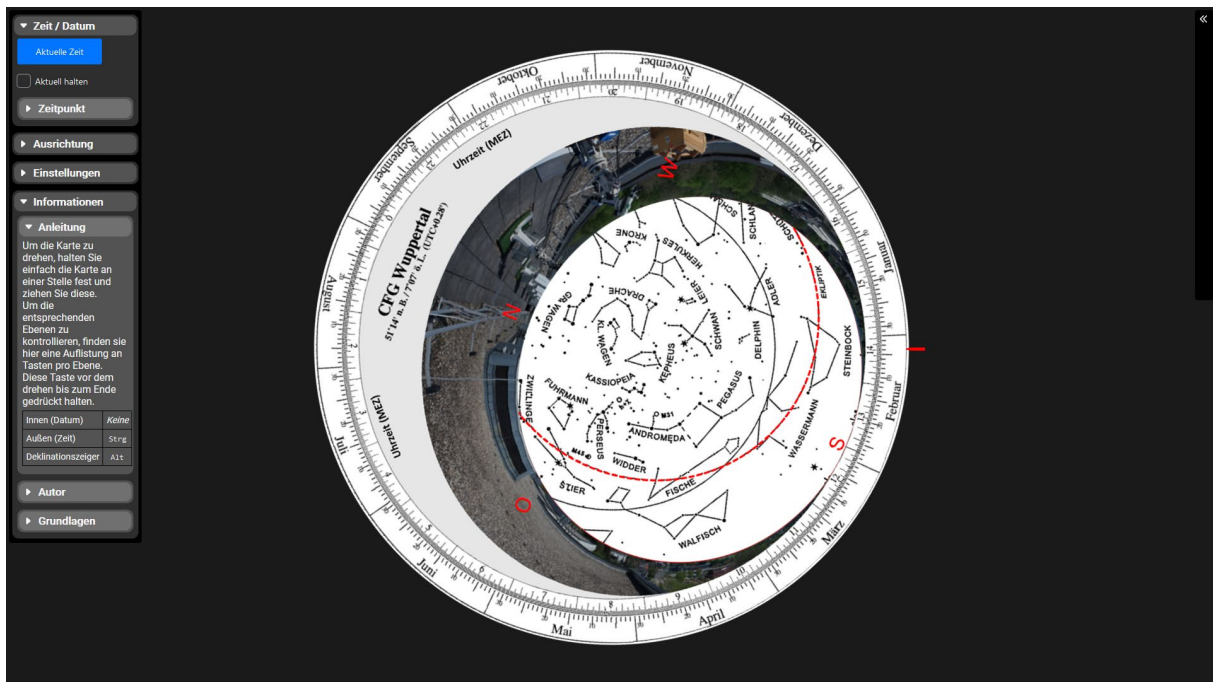


Abbildung 4 - Sternenkarte (Standardansicht)

Zielsetzung

Das Ziel der digitalen Sternenkarte war zunächst nur die Realisierung einer interaktiven Sternenkarte für den Computer. Der ursprüngliche Sinn war das einfache Demonstrieren der Benutzung und Funktionen. Es sollte möglich sein, den Schülern, über den im Astronomie-Raum vorhandenen Beamer, die Sternenkarte vorzuführen. Frühere Versuche, eine Dokumentenkamera zu nutzen, waren nicht optimal. Mit der Zeit kamen jedoch immer mehr Ideen und Funktionen hinzu, welche die Einsetzbarkeit dieser App deutlich erweitern.

Funktionen

- Darstellung der beiden (innere und äußere) Scheiben
- Drehen der beiden Scheiben durch das Ziehen per Maus
- Die Karte auf eine Uhrzeit und ein Datum via Eingabefeld setzen
- (optional) das Animieren der Karte, sodass die Bewegung zwischen Zeitpunkten besser nachvollzogen werden kann
- Die Karte auf den aktuellen Zeitpunkt setzen
- Die Karte aktuell halten
- Die MOZ (Mittlere Ortszeit) relativ zum ausgewählten Zeitpunkt angeben
- Die Karte auf eine Ausrichtung sperren (Blickrichtung) [Norden, Osten, Süden, Westen]

- Rektaszensionsskala (extra Ring) ein- und ausblenden können
- Deklinationszeiger dazuschalten können
- Zirkumpolar Kreis dazuschalten können
- Eigene Sternenkarten (innen und außen) hochladen und anzeigen lassen
- (optional) Design wechseln können (global) [Dunkel, Hell]
- Planeten auf die Karte (Ekliptik) legen und fixieren
- (optional) Zeitlauf simulieren (*abgebrochen*)

All diese Funktionen habe ich im Laufe dieser Besonderen Lernleistung umgesetzt, sodass diese App in vielerlei Hinsichten im Unterricht und von Hobbyastronomen genutzt werden kann. Bei der privaten Nutzung sollte beachtet werden, dass diese Sternenkarte auf den Standort des Carl-Fuhlrott-Gymnasiums eingerichtet ist, sodass diese bei zu starken geografischen Abweichungen nicht mehr akkurat genutzt werden kann.

Ansatz

Zuerst habe ich mir das bereits existierende (physische) Modell der Sternenkarte angeguckt. Das Prinzip war einfach, es wurden zwei sich frei drehende Scheiben am Mittel-/ Rotationspunkt mit einer Bastelklammer befestigt. Die äußere Scheibe hatte einen Ausschnitt, um ein wertabhängiges Blickfenster auf die innere Scheibe zu kreieren.

Da ein Ausschnitt nötig ist, musste das Bild Transparenz unterstützen. Transparenz ist nur im PNG und GIF-Format verfügbar. Das GIF-Bildformat ist zwar komprimiert, wodurch die jeweiligen Scheiben nur 200KB statt ca. 600KB groß wären, da jedoch der Unterschied so gering ist und der Browser Support beim PNG-Format deutlich höher ist, habe ich die Scheiben als PNG in Photoshop erstellt. Mein Ansatz zur Umsetzung lag darin, beide Scheiben als Bild im Browser einzubinden und mittig zu zentrieren. Die Rotation der einzelnen Scheiben konnte ich durch CSS-Transformationen erledigen.

Diese Idee ist eigentlich sehr simpel, da ich so einfach die Funktionsweise der physischen Karte auf die digitale übertragen kann. Umsetzung, Probleme und Methoden werden im folgenden Abschnitt erläutert.

Entwicklung und Probleme

Einbindung der Karten

In der Webentwicklung ist jedes Element im Aspekt des Layouts, also der Anordnung, eine Box oder um es mathematisch auszudrücken, ein Rechteck. Dieses Rechteck hat seinen Ursprung in der linken oberen Ecke (siehe Bild). Um dieses Rechteck zu zentrieren, bewegen wir es um 50% nach rechts und um 50% nach unten. Die prozentualen Werte werden vom Browser automatisch zum nächsthöheren Element (mit dem Attribut „display: relative“) in dem Fall der Viewport, also die gesamte Seite berechnet. Nun haben wir das Element verschoben, jedoch ist es noch nicht zentriert, da der Ursprung in der linken oberen Ecke und nicht in der Mitte liegt. Das ändern wir durch die „transform: translate“ CSS-Eigenschaft. Damit verschieben wir den Inhalt, aber nicht den Ursprung. Wir versetzen den Inhalt um -50% nach rechts und -50% nach unten. Da dieses Mal das eigene Element von den Prozenten betroffen ist, haben wir nun unser Rechteck perfekt zentriert. In der Abbildung wird die Position des Elementes durch die farbigen Kreise und der Viewport durch das gestrichelte Rechteck repräsentiert.

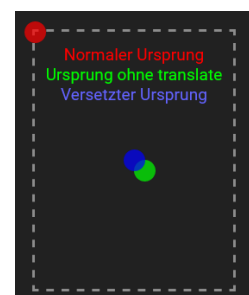


Abbildung 5 -
HTML-Box-Modell

Diese Transformation habe ich auf beide Karten, welche ich in Form eines Bildes eingebunden hatte, angewandt, wodurch die Basis der App stand. Als nächstes habe ich mich mit der Rotation der Karten beschäftigt.

Rotation der Karten

Zuerst habe ich zum Testen zwei Slider Eingabefelder von 0 bis 360 erstellt. Die Werte dieser Felder habe ich mithilfe eines v-models in den CSS-Parameter (Zeile 701; Datei StarsView.vue)

```
transform: `rotate(${this.finalRotation.inner}deg)`
```

auf die Scheiben angewandt. Da dieser Test funktioniert hatte und ich nun die Scheiben dynamisch je nach Nutzer-Eingabe verändern konnte, ließ sich die App weiter ausbauen. Zuerst hatte ich keine zuverlässige Methode gefunden, wie ich die Scheiben realistisch durch ‚Ziehen‘ bewegen konnte. Deswegen habe ich zuerst mit der Datums- und Zeiteingabe weitergemacht. Dadurch kann der Nutzer einen Zeitpunkt eingeben und die Karte springt anschließend zu diesem (mit einer Animation).

Der folgende Code sorgt für die Umwandlung des Datums und der Zeit in Grad, damit ich diese Werte im CSS-Code verwenden kann.

```
733   dateRotation() {
734       const rotationoffset = 171;
735       const dayNumber = Math.floor(
736           x: (new Date(this.convertedDate) -
737               new Date(this.convertedDate.getFullYear(), monthIndex: 0, date: 0)) /
738               (1000 * 60 * 60 * 24),
739       );
740       return -((dayNumber / 365) * 360 - rotationoffset);
741   },
742   timeRotation() {
743       const rotationoffset = 82.5;
744       let hourOffset = 0;
745       if (this.timezone === "MESZ") {
746           hourOffset = -1;
747       }
748       let hour = this.convertedTime.hours + hourOffset;
749       if (hour < 0) {
750           hour = 24 + hour;
751       }
752       const timeNumber = hour * 60 + this.convertedTime.minutes;
753       return +((timeNumber / 1440) * 360 + rotationoffset);
754   },
```

Abbildung 6 - Zeit- und Datumskonvertierung Code

Zuerst werde ich die erste Methode „dateRotation“ erklären. Beide Methoden legen zuerst einen Offset fest. Diesen habe ich durch näherungsweise Ausprobieren festgelegt. Der Offset-Wert gibt an, wie weit das Bild der Karte von dem idealen Wert Null entfernt ist. Dieser Wert ist notwendig, da ich die originale Orientierung der Ressourcen, also der Karten des PDF-Dokumentes beim Einbetten erhalte. Dadurch ist das Hinzufügen weiterer Karten deutlich einfacher, da die Rotation beibehalten werden kann. Deswegen werde ich die Zeilen 734 und 743 bei der weiteren Besprechung ignorieren. Die Hauptlogik findet in den Zeilen 735 bis 739 statt. Eigentlich ist es eine einzelne Zeile, doch zur Lesbarkeit habe ich diese auf mehrere aufgeteilt. In Zeile 737 erstelle ich ein neues Datumsobjekt, welches das Datum 31.12.JAHR-1 trägt (da „date“ auf 0 gesetzt ist). Wäre es auf 1 gesetzt, würde das Datum 01.01.JAHR lauten). Das Jahr ist dasselbe, wie das Jahr des eingegebenen Wertes. Dieses Datumsobjekt ziehen wir in Zeile 736 von der Eingabe ab. Durch diese Operation erhalten wir die Tage der Differenz in Millisekunden. Um aus den Millisekunden die Anzahl der Tage zu bekommen, nutzen wir Zeile 738. In dieser

erstelle ich den Dividenden für die Tage durch die Multiplikation der: Millisekunden zu Sekunden (1000) * Sekunden zu Minuten (60) * Minuten zu Stunden (60) * Stunden in Tage (24). In Zeile 737 findet die Division statt. Jedoch ist das Ergebnis nun eine Kommazahl (Beispiel: 38.71963780092592). Deswegen nutzen wir die Methode Math.floor in Zeile 735, um die Zahl abzurunden. Das Ergebnis für das Beispiel wäre somit 38 Tage seit Jahresbeginn. Diesen Tag des Jahres dividiere ich in Zeile 740 mit dem Wert 365 und multipliziere ihn mit 360, um die für den Tag passende Grad-Zahl zu erhalten. Zusätzlich subtrahiere ich noch den Offset, damit die Rechnung auch beim ersten Tag beginnt und ich gebe den Wert negiert zurück, da die Karte sich mit dem Fortschritt des Datums gegen den Uhrzeigersinn dreht.

Nun erkläre ich die zweite Methode „timeRotation“, welche die Zeilen 742 bis 754 umfasst. Diese Methode ist genauso wie die erste Methode, eine der simpelsten des Projektes, jedoch ist diese etwas komplexer als die vorherige, da hier auf weitere Faktoren Rücksicht genommen werden muss. Die Zeilen 744 bis 747 beschäftigen sich mit der Sommer- und Winterzeit. In einer weiteren Methode ermittle ich die aktuelle Zeitzone. Diese kann entweder MESZ oder MET sein. Da die Karte auf die Uhrzeit in MEZ gepolt ist, muss eine Stunde abgezogen werden, falls der Nutzer sich momentan in der Sommerzeit befindet. Die Zeilen 748 bis 751 wenden diese Zeittransformation an. Normalerweise kann die Stunde nicht unter 0 fallen, doch durch die Zeitzone und die dadurch folgende Subtraktion kann dies der Fall sein. Deswegen prüfe ich diesen Fall und normalisiere die Stunde falls nötig. In Zeile 752 wandle ich die Stunden in Minuten um und addiere diese mit den verbleibenden Minuten des Eingabefeldes. In dieser Methode findet die meiste Logik in Zeile 753 statt. Ich dividiere die Minuten durch 1440. Diese Zahl ergibt sich aus $60 \cdot 24$. Dadurch erhalten wir den prozentualen Fortschritt relativ zum Tag gesehen, abhängig von der eingegebenen Zeit zurück. Diesen multiplizieren wir wieder mit 360, um die Rotation zu erlangen. In diesem Fall addieren wir den Offset und geben das Ergebnis als positiven Wert zurück, da die äußere (Zeit) Scheibe sich bei fortlaufender Zeit mit dem Uhrzeigersinn dreht.

Der letzte Schritt für die Zeit-/ Datumseingabe war die Animation. Die App würde auch ohne diese funktionieren, jedoch wäre die Änderung nur sehr schwer nachzuvollziehen, da die Karte sonst einfach auf den neuen Wert springen würde. Ich gebe die Rotation über CSS-Eigenschaften an und konnte dadurch eine einfache CSS-Animation erstellen. Dafür musste ich nur folgende Zeile einfügen:

```
transition: transform v-bind(transitionSpeed + "s");
```

Diese habe ich auf alle Ebenen der Karte angewandt. Die Variable „transitionSpeed“ kann vom Nutzer festgelegt werden und gibt an, wie schnell sich die Karte drehen soll. Zusätzlich musste ich noch die Logik hinzufügen, dass die im folgenden Abschnitt beschriebene Interaktion durch Ziehen funktioniert. Die Animation hatte mit dem manuellen Bewegen interferiert, da der Browser versucht hat die Bewegung noch während des Ziehens zu animieren, wodurch fehlerhafte Sprünge entstanden. Dazu habe ich die Logik eingebaut, dass so lange gezogen wird, also die Maus gedrückt wird, diese transition Eigenschaft deaktiviert und nach Beendigung der Bewegung wieder reaktiviert wird.

Interaktivität durch Ziehen

Die Hauptfunktionalität dieser App hat mir eine lange Zeit Schwierigkeiten bereitet. Ich wusste nicht, wie ich es schaffe die Mauskoordinaten in eine Gradzahl umzuwandeln. Glücklicherweise habe ich auf meiner Suche nach einer Möglichkeit die Methode Math.atan2() entdeckt. Die Mozilla Referenz kann hier (<https://bit.ly/3UAoeni>) gefunden werden. Diese Methode bekommt als Eingabe einen x- und einen y-Wert und gibt den Winkel in Radiant Relativ zur x-Achse des Strahles vom Nullpunkt bis zu den Koordinaten zurück (siehe Abb.7).

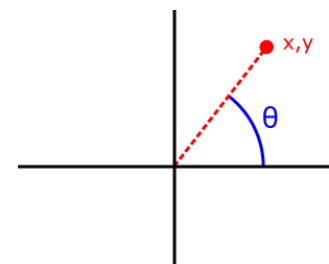


Abbildung 7 -
Atan2 Funktionsweise

Um diese Funktion nun verwenden zu können, musste ich natürlich noch den Nullpunkt festlegen, da der Koordinatenursprung im Browser genauso wie der Ursprung der Elemente in der linken oberen

Ecke liegt. Hätte ich diese Funktion nun einfach mit den Mauskoordinaten verwendet, würde die Rotation nicht um den Mittelpunkt der Karte, sondern von diesem Koordinatenursprung berechnet werden und würde somit zu falschen Ergebnissen führen. Um das Problem zu lösen, habe ich die Koordinaten des Mittelpunktes des Viewports von den eigentlichen Koordinaten abgezogen. Die x-Werte stimmten somit, doch die y-Werte waren nun negiert. Mir ist das Problem mit den negierten y-Werten jedoch bei der Entwicklung zunächst nicht aufgefallen, da ich zum korrekten Anwenden der Rotation noch eine weitere Berechnung durchgeführt habe, welche das Problem überflüssig macht. Um die neue Rotation nun anzuwenden, musste ich drei Schritte durchführen. Zuerst habe ich mir die Rotation jeder Ebene beim (ersten) Klick gespeichert, das heißt, wenn die Bewegung beginnt, jedoch noch vor einer Veränderung. Im gleichen Zuge habe ich mir auch die Koordinaten des Klickstarts gespeichert. Zuletzt berechne ich mir bei jeder Bewegung (währenddessen die Maus noch gehalten wird), also während des Ziehens, den Winkel der aktuellen Mausposition mit der `Atan2` Methode und den Winkel des Klickstarts. Darauf subtrahiere ich den Startwinkel von dem neuen Winkel, wodurch ich die Differenz erhalte. Anschließend addiere ich den Winkel vor der Bewegung, um den neuen finalen Winkel zu erhalten. Eventuell denken Sie sich jetzt, die Reihenfolge der Subtraktion ist falsch, da man eigentlich den neuen Winkel von dem Startwinkel abziehen würde. Dieser ‚Fehler‘ ist jedoch gewollt. Wenn Sie sich nochmal Abb.7 anschauen, werden Sie merken, dass der Winkel mit der `Atan2` Methode gegen den Uhrzeigersinn wächst. Jedoch bedeutet ein wachsender Wert des Winkels bei den CSS-Attributen eine Rotation mit dem Uhrzeigersinn. Dadurch wird dieser Unterschied in Kombination mit dem negierten y-Wert ausgeglichen.

Sobald die Maus wieder losgelassen wird, aktiviere ich wieder die Transition/Animation und setze die zuvor gespeicherten Werte wieder zurück, damit es keine Probleme bei einer nachfolgenden Bewegung gibt. Diese Methoden („`handleMouseDown`“, „`handleMouseMove`“ und „`handleMouseUp`“) ermöglichen somit eine realistische/intuitive Interaktion mit der Sternenkarte. Der Quellcode der zuvor erklärten Funktionen kann hier eingesehen werden (<https://bit.ly/3HWHwMq>).

Jetzt gab es noch ein Problem mit der interaktiven Rotation. Die obengenannten Funktionen/Methoden³⁵ habe ich über einen Event-Listener eingebunden. Dieser (wie der Name schon sagt) hört auf Events. Weitere Informationen zu Event-Listnern können hier (<https://bit.ly/49ISMgW>) eingesehen werden. Ich muss einen Event-Listener auf ein HTML-Element anwenden. Ich habe mich zu Beginn für die individuellen Layer entschieden. Dabei ist mir aufgefallen, dass immer nur der höchste Layer ein Event gesendet hat. Das lag daran, dass dieser die anderen Layer für solche Events verdeckt und diese abgefangen hat. Deswegen habe ich die unteren Listener entfernt und nur die entsprechenden Listener für das oberste Element behalten. Doch dann ist mir noch ein Problem aufgefallen. Damit die Karte über die Bilder richtig eingebunden werden konnte, musste ich die Höhe und Breite dieser festlegen. Beides lag bei 1000px. Das hatte jedoch zur Folge, dass die Events nicht mehr gesendet wurden, wenn der Nutzer außerhalb dieses Bereiches mit der Maus war. Wenn man die Karte nun so verwenden würde, als wäre diese real, würde es kein Problem darstellen. Wenn man jedoch nur leicht abrutscht und diesen Bereich verlassen hatte, hat der Browser das so interpretiert, als würde die Aktion vom Nutzer abgebrochen worden sein, woraufhin dieser neu beginnen musste. Deswegen habe ich mich dafür entschieden, um die Benutzung dem Nutzer zu erleichtern, dass ich um jedes Kartenelement noch einen sogenannten Wrapper hinzufüge, welcher die volle Viewport Größe einnimmt. Zusätzlich habe ich noch einen Wrapper um alle kleineren Wrapper hinzugefügt, welcher über allen

³⁵ Eine Funktion wird Funktion genannt, wenn diese außerhalb einer Klasse (beim objektorientierten Programmieren) steht. Diese wird Methode genannt, wenn diese Teil einer Klasse ist. Da JavaScript keine objektorientierte Programmiersprache ist und es somit keine Klassen gibt, es jedoch trotzdem ein geteiltes Scope mittels „`this`“ gibt, kann bei JavaScript der Name Methode oder Funktion als Synonym verwendet werden.

Kartenelementen lag, damit ich alle Event-Listener auf diesen anwenden konnte und mir keine Sorgen mehr um den z-index³⁶ der drunter liegenden Elemente zu machen.

Einen separaten Abschnitt für das Ausrichten der Karte, habe ich mir hier gespart, da das Thema sehr schnell behandelt ist. Die Manipulation der Ausrichtung der Karte verwendet man, um die Blickrichtung zu simulieren. Wenn ich beispielsweise nach Norden gucke, drehe ich die Karte so, dass N (Norden) nach unten zeigt. Dadurch werden die Sterne richtig angezeigt. Um diese Funktion nun in meine App zu übernehmen, habe ich es mir einfach gemacht, indem ich noch ein Wrapper Element (unter dem gesamten Wrapper, jedoch um die beiden Karten herum) hinzugefügt habe. Dieses Element hat seine eigene Rotation, welche durch das Auswählen einer Himmelsrichtung in 90 Grad Schritten gedreht wird. Da die Ausrichtung von der äußeren Scheibe abhängt, wird diese verändert, sobald der Nutzer die äußere Scheibe durch eine Änderung der Zeit dreht. Deswegen habe ich noch eine Checkbox hinzugefügt mit dem Titel „Rotation sperren“. Wenn diese ausgewählt ist, dann wird bei der Veränderung der äußeren Scheibe die gesamte Rotation des Wrappers neu berechnet in Abhängigkeit von der äußeren Rotation. Da sich dann der Wrapper entgegengesetzt zu der äußeren Scheibe dreht, bleibt diese gleich/stabil.

Einfügen eines zusätzlichen Ringes

Da die App nun verwendet werden konnte und keine Grundfunktionen mehr fehlten, konnte ich mich nun an die komplexeren/erweiterten Funktionen begeben. Dazu gehörte die Ergänzung eines weiteren Ringes. Dieser Ring zeigt die Rektaszension an und kann in Kombination mit der Deklination genutzt werden, um die Position eines Himmelskörpers festzulegen. Dieser Ring ist statisch an die untere, also die Datumsebene/scheibe gebunden. Somit wäre mein erster Ansatz gewesen zwei Versionen der unteren Ebene zu erstellen, eine mit und eine ohne Rektaszensionsskala. Jedoch war meine Aufgabe, dass man diese optional während der Verwendung der App einblenden können soll. Die Version mit den zwei verschiedenen Bildern hätte funktionieren können, jedoch hätte dieser Ansatz zu hoher Wahrscheinlichkeit viele Probleme verursacht. Dazu gehört die Skalierung (diese muss, auch wenn sich das Bild verändert, für den Inhalt des Bildes gleichbleiben) und das Neuladen eines Elementes. Da im Hintergrund viele Methoden ablaufen, welche zum Teil von „refs“³⁷ abhängig sind und auf diese zugreifen, entsteht das Problem, dass auf ein leeres/nicht existierendes/null Element nicht zugegriffen werden kann. Deswegen war es klüger, die Ebene separat in die App einzubinden und über Logik (das Anwenden der gleichen Rotation wie für die untere Scheibe) die Ebenen synchron zu halten. Dieser Ansatz ermöglicht außerdem das Austauschen der Karten (welches noch angesprochen wird) ohne Probleme.

Da ich keinen Zugang zu den Quelldateien der angepassten Sternenkarte von P. Post und P. Plaster habe, sondern nur das PDF eines Scans von dieser, sind die Karten durch die perspektivische Verzerrung leicht verformt. Bei der normalen Verwendung würde das nicht auffallen, doch diese Unterschiede (z.B. 4px höher als breiter), sorgen für eine leichte Überlappung des (perfekten) Ringes und der Karte. Ich habe es so gut es geht versucht, diese leichte Verformung in Photoshop zu korrigieren, doch ein bisschen Asymmetrie kann durch den drüber liegenden Ring immer noch wahrgenommen werden.

³⁶ Der z-index legt die Höhe eines Elementes in der Webentwicklung fest. Dadurch können auch Kinder eines Elternteils höher in der Hierarchie liegen als das Elternteil selbst.

³⁷ „refs“ sind in Vue.js eine Möglichkeit für das einfache Zugreifen auf verschiedenste reaktive und somit dynamische Elemente.

Platzierung weiterer Elemente

Damit man die Rektaszensionsskala nun auch sinnvoll verwenden konnte, fehlte nun noch die Deklination. Diese wird im Gegensatz zu den anderen Instrumenten nicht als Ring oder Scheibe, sondern als Zeiger dargestellt. Der Deklinationszeiger kann in Abb.8 betrachtet werden.

Die Erstellung von diesem war ziemlich aufwendig (auch wenn es erstmal nicht danach aussieht). Zur Erstellung hatte ich mir eine physische Sternenkarte mit Rektaszension und Deklination gescannt, um die Abstände und Skalierung korrekt hinzubekommen. Nach der Erstellung der Rektaszensionsskala, habe ich mit dem Deklinationszeiger in Photoshop begonnen. Damit ich diesen besser vom Hintergrund (die Karte konnte man nicht demontieren) unterscheiden konnte, hatte ich ein weißes Blatt Papier zwischen Karte und Zeiger gelegt. Als ich dann in Photoshop den Zeiger fast fertiggestellt hatte (ich habe die Transformationen immer wiederholt, um das Ergebnis konsistent zu halten), musste ich diesen noch so skalieren, dass dieser auf die Sternenkarte des CFGs passte und mit den entsprechenden Kontrollpunkten übereinstimmte. Als ich das versucht habe, hat es zwar grundlegend funktioniert, doch man konnte die Zahlen nicht mehr lesen und die Striche waren manchmal fast transparent. Das lag daran, dass Photoshop eine Software zur Bearbeitung von Rastergrafiken war. Jedes typische Bild, welches Sie kennen, angucken oder aufnehmen, ist in der Regel eine Rastergrafik. Das bedeutet, das Bild hat eine festgelegte Anzahl an Pixeln, wobei jeder einen Farbwert annimmt, wodurch das Bild entsteht. Das Problem entstand nun dadurch, dass ich diese Pixel skaliert habe. Da die Skalierung nicht durch eine natürliche Zahl, sondern durch eine Gleitkommazahl erfolgt ist, hat Photoshop versucht diese Transformation so gut es geht anzuwenden. Da es keine halben Pixel, sondern nur 1 oder 0 gibt, musste Photoshop nun abwägen, ob der Inhalt noch auf den benachbarten Pixel kopiert wird oder nicht. Dadurch gingen manche Pixel verloren und manche neuen Pixel wurden ‚erfunden‘. Das hatte zur Folge, dass die Skala, welche klein und präzise war, nicht mehr zu lesen war. Zusätzlich hat noch das Anti-Aliasing das ganze verschlimmert. Diese Technik verwendet der Computer bei jeder Pixelverbindung, welche nicht perfekt vertikal oder horizontal ist. Wenn man zum Beispiel eine um 45° gedrehte Linie darstellen möchte, dann scheitert der Computer an dieser Aufgabe, da es keine kleinere Einheit als einen Pixel für den Computer bei der Darstellung gibt. Deswegen füllt dieser jeden Pixel, welcher von der Linie geschnitten wird. Doch so entsteht eine Treppe, keine Linie. Deswegen fügt der Computer noch abfallend transparente Pixel um diese hinzu, damit es besser und glatter aussieht (Siehe Abb.9).

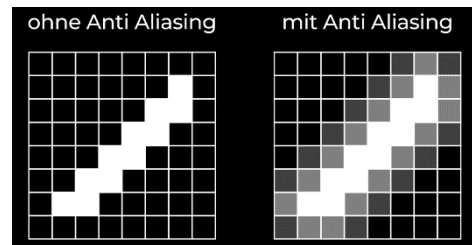
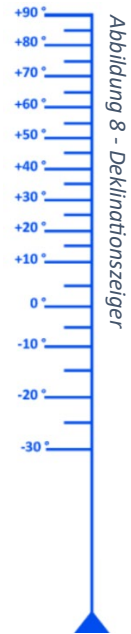


Abbildung 9 - Beispiel: Anti-Aliasing

Um das Problem nun zu lösen, habe ich die gesamte Skala erneut in dem Programm Adobe Illustrator kreiert. Da dieses Programm SVGs erstellt (Scalable Vector Graphics), welche nicht aus Pixeln, sondern aus mathematischen Formeln und Befehlen bestehen, konnte ich diese nach der Erstellung richtig skalieren und erst im Anschluss rastern³⁸.

Als ich dann dachte ich wäre fertig, habe ich das Ergebnis meinem Lehrer (Herrn Koch) gesendet, welcher mir mitteilte, dass der Zeiger doch nicht linear skaliert ist, sondern gestreckt und gestaucht werden muss. Daraufhin hat er mir eine überarbeitete Version zugesendet. Diese war allerdings wieder fehlerhaft gerastert, weswegen ich diese zwar als Vorlage nehmen konnte, doch trotzdem den Zeiger in Illustrator nochmal komplett neu erstellen musste.

³⁸ Umwandeln eines SVGs in eine Rastergrafik.

Bei der Einbettung des Zeigers konnte ich dieselben Prinzipien wie bei den vorherigen Einbettungen verwenden, weswegen dieser Teil nun sehr einfach war. Ich konnte alles kopieren und musste nur Anpassungen an den Variablennamen und am zugehörigen Hotkey durchführen. Zusätzlich habe ich den Deklinationszeiger sowie die Rektaszensionsskala über eine eigene Checkbox optional gemacht, sodass der Nutzer diesen bei Bedarf einblenden kann.

Zusätzlich zu der Rektaszension und der Deklination, habe ich noch den Zirkumpolar Kreis eingebunden. Der Zirkumpolar Kreis markiert die Sterne, welche niemals unter den Horizont fallen und somit das ganze Jahr über sichtbar sind. Man sollte beachten, dass die gesamte Karte standortabhängig ist, wodurch die angegebenen relativen Sternpositionen nur für diesen Standort gelten. Da sich der Nachthimmel abhängig vom Standort verändert, befinden sich natürlich andere Sterne im Zirkumpolar Kreis (relativ zum Beobachter). Die Implementation von diesem war sehr einfach, da ich bereits die Logik zur Positionierung und Zentrierung von diesem von den anderen Objekten übernehmen konnte. Um die Rotation musste ich mir beim Kreis selbstverständlich keine Gedanken machen, da dieser den Ursprung der Karten teilt und kein Ellipsoid ist, weswegen die Rotation irrelevant ist.

Bekannte Sternbilder (an unserer Position), welche im Zirkumpolar Kreis liegen, sind der Große Wagen, Kassiopeia und der Kleine Wagen. Im folgenden Bild sind alle zuvor angesprochenen Informationen aktiviert und aufgeklappt. Sie können die Unterschiede mit dem Bild auf Seite 11 vergleichen.

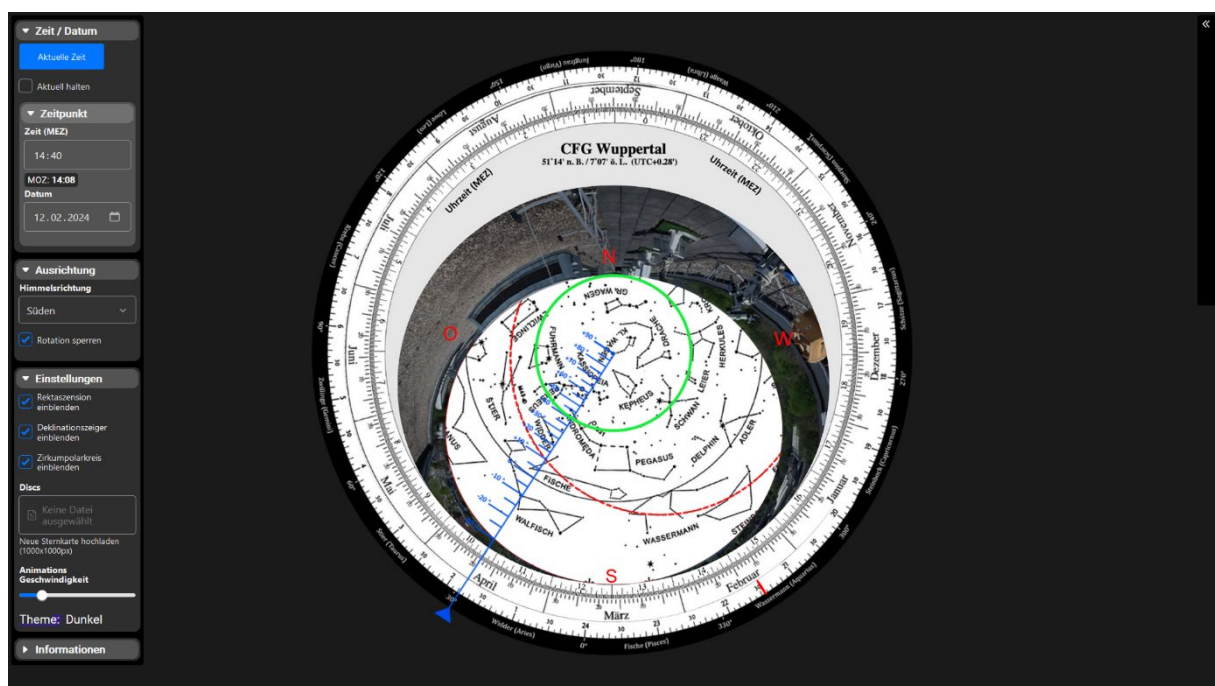


Abbildung 10 - Sternkarte + alle Extras

In der App, also auch auf dem Bild wird die Ekliptik (statisch)³⁹ rot gekennzeichnet, der Deklinationszeiger blau und der Zirkumpolar Kreis grün. Die Rektaszensionsskala befindet sich als Ring um die Datums-skala. Der Marker für den aktuellen Zeitpunkt befindet sich rot in dem rechten unteren Teil der Karte. Normalerweise würde sich dieser (wie in *Abbildung 4* zu sehen) außerhalb der Karte befinden, da sich jedoch nun die Kartengröße aufgrund des zusätzlichen Ringes verändert, behalte ich den Marker an derselben Position, damit die Gleichung zur Positionierung noch funktioniert.

³⁹ In diesem Fall bedeutet statisch, dass die Ekliptik fest in die Karte ‚gebrannt‘ ist. Da ich keine Version der Karte ohne diese habe, kann ich diese nicht optional einblendbar machen, wodurch diese dauerhaft sichtbar ist.

Die Erstellung des Markers war eine meiner ersten Aktionen nach der Fertigstellung der *Interaktivität durch Ziehen*. Jedoch erwähne ich diesen erst jetzt, da nun die benötigten Vorkenntnisse bereits durch die vorherigen Abschnitte geschaffen sind.

Der Marker ist eigentlich eine optionale Funktion, da die Sternekarte so funktioniert, dass man diese an jeder Stelle ablesen kann. Um die Karte auf einen Zeitpunkt einzustellen, dreht man diese so, dass die gewünschte Uhrzeit über dem gewünschten Datum liegt. Da ich diese Herangehensweise immer als sehr unübersichtlich empfunden habe und man auch nur schwer auf die Schnelle einsehen konnte, ob der eingestellte Zeitstempel übernommen wurde, habe ich mich dafür entschieden, einen Marker für die leichtere Benutzung hinzuzufügen.

Für den Marker habe ich ein einfaches HTML-Element (ein div) genutzt. Zuerst erstellte ich eine rote Box. Für diese habe ich die Höhe auf 1/6 der Breite eingestellt. Als letztes habe ich beide linken (oben und unten) Ecken zu 100% abgerundet, wodurch diese Form entstanden ist:



Die Erschaffung der Form war der einfache Teil. Der schwere Teil war die Positionierung des Markers. Um die Position nun zu berechnen, nutze ich sowohl relative als auch absolute Werte und die CSS „calc“ Funktion. Da die Karte rund ist, konnte ich mit dem Radius arbeiten. Zuerst habe ich das Objekt erstellt, formatiert und mittig zentriert (durch einen Abstand von 50% zum oberen Ende des Viewports). Dann habe ich das Element um 50% nach rechts versetzt, so dass sich dieses nun in der Mitte befindet. Nun verwende ich eine aufwendigere Berechnung (Details können in Abb.12 eingesehen werden), um den Radius der Karte zu berechnen (abzüglich der Breite des Markers) und um die finale Position zu erhalten. Die Berechnung ist so aufwendig, da die Karte nicht immer gleich groß dargestellt wird, sondern von einem Durchmesser von maximal 1000px (die Grafik ist nicht größer) bis auf die minimale zur Verfügung stehenden Bildschirmgröße runterskaliert wird. Weitere Informationen dazu erwähne ich in dem Abschnitt *Dynamische Skalierung*.

Abbildung 11 -
Sternekarte: Marker

```
911 #marker {
912   --adaptedSize: v-bind(adaptedSizeStyle);
913   --adaptedSizeRaw: v-bind(adaptedSize);
914   /*creates a marker on the edge of the circle*/
915   position: absolute;
916   top: 50%;
917   left: calc(
918     50% +
919     (
920       (var(--adaptedSize) - (60px * (var(--adaptedSizeRaw) / (1000 - 90)))) /
921       2
922     )
923   );
924   width: calc(30px * (var(--adaptedSizeRaw) / 1000));
925   height: calc(5px * (var(--adaptedSizeRaw) / 1000));
926   background: #ff0000;
927   z-index: 7;
928
929   border-top-left-radius: 100%;
930   border-bottom-left-radius: 100%;
931 }
```

Abbildung 12 - Marker CSS Code

Platzierung von Planeten

Eine weitere Idee für die Einsetzbarkeit der App im Astronomie Unterricht ist das Platzieren von Planeten auf der Karte. Da man nun die Rektaszension und Deklination für einen Punkt auf der Karte ablesen kann, kann man auch Planeten oder andere bewegliche Himmelskörper auf der Karte platzieren. Die Planeten können sich nur auf der Ekliptik (rot-gestrichelte Linie) befinden. Ich kann die Planeten nicht automatisch ohne weitere Informationen und Berechnungen einblenden, da sich diese im Gegensatz zu den Sternen relativ zu uns bewegen. Jedoch kann man mit der Rektaszension (RA) und der Deklination (Dec) die Position jedes Himmelskörpers für einen bestimmten Zeitpunkt errechnen. Die Aufgabe für die Schüler wäre nun beispielsweise: „Berechnet die Positionen für die Planeten: Jupiter, Saturn und Mars für den Zeitpunkt: 14.07.2024 um 3:20 Uhr. Präsentiert eure Ergebnisse mit der Sternenkarte“.

Das Implementieren dieser Funktion war mit Abstand die schwerste Aufgabe in der Sternenkarten App, da ich mit neuen Technologien wie der HTML Drag and Drop API arbeiten musste und ich noch nie Koordinatentransformationen durchgeführt hatte. Wenn Sie diese Funktion selbst testen möchten, empfehle ich die Verwendung des Firefox Browsers. Ich konnte manche Standardverhalten im Chrome nicht überschreiben, wodurch ein leichter Versatz und eine verschobene Darstellung während des Ziehens entsteht. Zum Safari Browser habe ich keinen Zugang, da ich kein Apple-Endgerät benutze. Das Verhalten dieser Funktion variiert so stark von Browser zu Browser, da diese Technologie noch relativ neu ist und jeder Browser keine standardisierte Implementation, sondern deren eigene nutzt.

Zunächst habe ich einen separaten Vue-Component erstellt, da jeder Planet die gleiche Logik benötigt, jedoch wird duplizierter Code (unter Programmierern) nicht gerne gesehen. So kann ich den Code an einer Stelle anpassen und die Änderungen werden auf alle Planeten übernommen. Um jeden Planeten eindeutig erkennbar zu machen, hatte ich zuerst die Idee, eine stilisierte Version jedes Planeten zu erstellen, dabei hätte z.B. Saturn einen Ring. Jedoch sind mir keine weiteren eindeutigen Merkmale für die anderen Planeten eingefallen, weshalb ich mich dafür entschieden habe, ein Bild von jedem Planeten zu nehmen, wie die Planeten von der Erde aus zu sehen/fotografieren sind. (Für die Venus habe ich ein Bild ohne die dicke Atmosphäre gewählt, damit diese auffälliger ist). Die verfügbaren Planeten (die Erde ist natürlich nicht dabei, da wir von der Erde aus dem Himmel beobachten) sehen dann wie folgt aus (Abb.13).



Abbildung 13 -
platzierbare Planeten

Nach der Erstellung der Planeten und der Darstellung dieser in dem Flexbox⁴⁰ Element (siehe Abb.13), musste ich eine „dropzone“ erstellen. Durch das Festlegen, dass dieses Element eine Dropzone ist, werden bestimmte Verhalten im Browser geändert. Dadurch sieht der Browser dieses Element nun als „drop-target“ an. Wenn nun ein bewegliches („draggable“) Element auf dieses Element gezogen wird, dann bricht der Browser das Drag-Event nicht so wie bei der gleichen Interaktion mit anderen Elementen ab, sondern er sendet von dem Zielelement/Dropzone ein Event „drop“ aus. Dieses Event liefert zusätzliche Informationen, welche ich ohne die Drag and Drop API nicht erhalten würde. Ich bekomme die Information, welches Element auf das Ziel gezogen wurde und wohin. Darüber hinaus erhalte ich noch deutlich mehr Informationen, doch diese benötige ich nicht. Dem zu verschiebenden Element,

⁴⁰ Ein Flexbox Element ermöglicht die automatische Anordnung der enthaltenen Inhalte in Bezug auf deren Größe und festgelegten Regeln. In dem obigen Beispiel habe ich einen Abstand zwischen den Elementen von 10px und ein automatisches Wrapping eingestellt. Dadurch füllen die Planeten automatisch und fortlaufend den Container.

also dem Planeten, kann ich am Anfang noch zusätzliche Informationen mitgeben. Diese Gelegenheit nutze ich natürlich. Ich übergebe die ID (welchen Planet ich verschoben habe) und die Position, an welcher Stelle ich diesen gegriffen habe. Diese Informationen werden später noch hilfreich. Damit sich die Planeten hinterher mit der Karte drehen, muss ich diese als Kind der Karte realisieren, damit die Transformation (Rotation) der Karte auch auf die Planeten angewendet wird. Doch nun habe ich ein Problem: Die Zielposition wird relativ zum Zielelement angegeben. In den meisten Anwendungsfällen ist das hilfreich, jedoch habe ich ein Zielelement, welches ich, sobald das Ziehen startet, über alle anderen Elemente teleportiere. Wenn ich nun die erhaltenen Koordinaten für den Planeten anwende, stimmt die Position nicht, da ich die Rotation der Karte mit beachten muss. Um dieses Problem zu lösen, habe ich lange nach Formeln gesucht, doch leider nichts Passendes gefunden. Das Stichwort hier lautet Koordinatentransformation. Da dieses Thema nicht mehr in der Schule gelehrt wird, wusste ich nicht, wie ich dieses umsetzen sollte. Doch dann hatte ich eine Idee. Ich kann einfach die Dropzone auf dieselbe Art und Weise drehen, wie ich die Karte gedreht habe. Dadurch stimmen die Koordinaten wieder überein und ich kann diese verwenden.

Doch nun entstand das nächste Problem. Aus irgendeinem Grund waren die Planeten teilweise versetzt (in verschiedenste Richtungen) und teilweise nicht. Ich habe mich eine ganze Zeit lang gewundert, warum dieses Problem auftrat. Dann ist es mir aufgefallen: je nachdem, wo ich einen Planeten greife, ist dieser versetzt oder nicht, also wurde diese Position vom Browser auch berücksichtigt und so interpretiert, als würde ich einen unsichtbaren Ursprung an die Greif Position setzen. Da jedoch mein Ursprung in der linken oberen Ecke und in der Mitte liegt (ich habe beide Varianten ausprobiert), entsteht so ein Versatz. Deswegen habe ich zu Beginn die Klickposition beim Starten mit übergeben, sodass ich diese bei Beendigung der Verschiebung von der Verschiebung subtrahieren kann, um ein genaues Ergebnis zu erhalten.

Während der Entwicklung dieser Funktion ergaben sich noch viele weitere Schwierigkeiten, doch diese sind zu umfassend, um diese hier alle aufzulisten und nach einem Drittel Jahr erinnere ich mich auch an vieles nicht mehr. Für weitere Informationen können Sie sich in meiner GitHub Repository die Pull-Requests [437, 444, 453, 459] anschauen (Link: <https://github.com/domx4q/astroProject/pulls>).

Dynamische Skalierung

Einer der schwersten Aufgaben in der Webentwicklung ist das Erstellen eines „responsive designs“. Das bedeutet, dass die Webseite je nach Bildschirmgröße anders dargestellt ist. Dabei wird die Seite nicht nur hoch- oder runterskaliert, sondern die gesamte Webseite wird je nach Aufbau komplett umstrukturiert. Eine bewerte Methode dabei ist die Verwendung von Elementen, welche sich einklappen können und so Platz sparen, wenn diese nicht verwendet werden. Häufig werden dafür sogenannte Dropdowns verwendet. Diese zeigen nur die Überschrift, bis diese aufgeklappt werden. Dann wird der Inhalt angezeigt. Ein anderer Ansatz ist das Einklappen von Seitenleisten. Manchmal wird ein anderes Element für die gleiche Funktion gewählt, je nachdem, ob der Nutzer am Desktop oder am Handy ist. In diesem Projekt habe ich mir jede dieser Techniken zunutze gemacht, damit die App unabhängig vom Endgerät ausgeführt werden kann.

Ich hatte die Entwicklung der dynamischen Skalierung zuerst nach hinten geschoben, da diese wie zuvor schon erwähnt sehr aufwendig ist und ich mich zunächst auf die großen Funktionen konzentrieren wollte. Deswegen funktionierte die App mit der Bildschirmauflösung (1920px * 1080px) sehr gut, da ich die App in dieser Auflösung entwickelt und getestet habe, jedoch mussten Nutzer kleinerer Geräte zunächst die Browser Funktionen zum Rauszoomen nutzen, damit alles angezeigt wurde. Das war für mich noch nicht Grund genug, um mit der aktiven Entwicklung dieser Funktion zu beginnen, da man die App immerhin verwenden konnte. Allerdings hatte mich mein Astronomielehrer Bernd Koch einige Zeit später darauf angesprochen, dass er die Sternenkarte gerne auf seiner Webseite

(<https://www.schuelerlabor-astronomie.de>) einbinden würde. Dabei habe ich ihm geholfen, jedoch standen der App auf seiner Seite aufgrund des iFrames⁴¹ nur noch (716px * 900px) zur Verfügung. Die Browserzoom-Funktionen konnte man nun auch nicht mehr nutzen, da diese nur die Skalierung der Host-Seite, also „schuelerlabor-astronomie“ verändern, jedoch nicht das iFrame. Deswegen habe ich mich nun mit der Entwicklung der dynamischen Skalierung beschäftigt.

Die meisten Seiten verwenden für diesen Zweck CSS @media tags. Diese erlauben das konditionale Anwenden des CSS-Codes je nach Gerätegröße oder anderen Parametern. Für CSS ist diese Funktion besonders, da es in anderen Fällen nicht die Ausführung von Logik erlaubt, um den CSS-Code zu ändern. In meinem Fall jedoch habe ich für das meiste JavaScript verwendet, da ich sowieso durch Vue.js bereits genug Kontrolle übers DOM⁴² habe, weswegen ich dann auch die Vorteile von JS nutzen kann.

Das Wichtigste ist, dass die Karte unabhängig von der Gerätegröße funktioniert. Dafür habe ich mir diese kleine Methode als computed-property geschrieben:

```
644  adaptedSize() {
645      const DEFAULT_SIZE = 1000;
646      const widthOffset = 60;
647      const heightOffset = -30; // make the region a bit larger to prevent clipping
648      const screenSize = this.screenSize;
649      const smallerSide = Math.min(
650          screenSize.width - widthOffset,
651          screenSize.height - heightOffset,
652      );
653      const result = Math.min(DEFAULT_SIZE, smallerSide);
654      if (this.showRektaszension) {
655          return result - 90; // the rektaszension disc is 90px wider than the inner disc
656      }
657      return result;
658  },
659  showFullSidePanel() {
660      return this.screenSize.width - this.adaptedSize > 480;
661  },
```

Abbildung 14 - adaptedSize Code

Vorab: Der grüne Strich an der linken unteren Seite bedeutet nur, dass ich Code hinzugefügt habe. Ich habe diese Methode nach oben geschoben, damit diese mit auf den Screenshot passt. Die Methode `adaptedSize` (von Zeile 644 bis 658) berechnet den Durchmesser/ die Größe für die Ebenen der Karte (Datum und Zeit). In den ersten drei Zeilen des Methoden-Inhaltes lege ich benötigte Werte für die Berechnung fest. Die `DEFAULT_SIZE` liegt bei 1000, da die Bilder der Karten (1000px * 1000px) groß sind. Die beiden Offset-Werte geben an, wie viele Pixel transparent auf dem Bild am Rand sind, also nach wie vielen Pixeln die Karte anfängt. Wenn ich diese mitberücksichtige, kann ich die Karte noch etwas größer machen, ohne diese abzuschneiden. Der Offset für die Höhe ist negativ, da ich oben weniger Abstand benötige. Somit kann ich sogar etwas von der Höhe abziehen, um die Karte noch größer zu machen. Der Wert -30px hat sich beim Testen als optimal erwiesen. Die Zeile 648 mag redundant erscheinen, jedoch gewährleiste ich so die automatische Neuberechnung der Größe, falls die Bildschirmgröße verändert wird. Das liegt an einem momentan vorhandenen Fehler in Vue, da vom Watcher teilweise nicht erkannt wird, wenn ein reaktiver Wert in einer computed property nur in einer externen Methode verwendet wird. Durch die Zwischenspeicherung dieses Wertes kann ich

⁴¹ Ein iFrame ist ein HTML-Element, welches das Einbetten anderer Seiten auf der eigenen Seite erlaubt.

⁴² Das DOM (Document Object Model) verwaltet die HTML-Struktur und erlaubt die Manipulation dieser.

gewährleisten, dass der Watcher die Verknüpfung dieser Variable erkennt und die computed property aktualisiert. Die Zeilen 649 bis 652 gehören zusammen und sind eigentlich eine Zeile, doch prettier⁴³ hat in Kombination mit eslint⁴⁴ die Zeile so formatiert, dass diese besser zu lesen ist. Diese Zeilen berechnen in Rücksichtnahme der beiden Offsets die kleinere/schmalere Seite. In Zeile 653 wird das Ergebnis mit der Math.min Funktion berechnet. Ich gebe nicht direkt die kleinere Seite als Größe zurück, sondern nutze diese Methode, damit die Karte die eigentliche/normale Größe nicht überschreiten kann. Das hat wieder mit der Rasterung von Bildern zu tun. Da die Karte nur eine bestimmte Auflösung hat, wird so gewährleistet, dass diese nicht überschritten wird und das Bild scharf bleibt. Somit begrenzt die Zeile das Ergebnis auf maximal 1000px. Die folgenden Zeilen bis 658 sorgen nur dafür, dass das Ergebnis zurückgegeben wird und falls die Rektaszension eingeblendet wird, die Karte kleiner skaliert wird, damit die Rektaszension um diese herum passt.

Die Positionierung des Markers (siehe *Abbildung 12 - Marker CSS Code*) basiert ebenfalls auf dieser Berechnung, wie in den Zeilen 912 und 913 zu sehen ist. Wäre dieser nicht an die dynamische Skalierung der Karte gebunden, würde dieser irgendwo, aber nicht mehr am Rand der Karte angezeigt werden.

Wie schon zuvor angekündigt, habe ich auch sich aus- und einklappbare Elemente verwendet. Der erste Schritt lag für mich in der Umsetzung dieser Funktion für das linke Seitenmenu, da dieses im Schülerlabor Embed über die Karte ragte. Dafür habe ich den Vue Component „collapsableContainer“ erstellt (Quellcode: <https://bit.ly/3T1Ucro>). Dieser konnte über ein Slot-Element den Inhalt ganz normal darstellen, jedoch konnte man diesen auch einklappen. Die Zeilen 659 bis 661 der Abbildung 14 erstellen den Boolean, welcher angibt, ob dieser Container als normaler Container angezeigt werden soll, oder ob er zugeklappt sein soll und durch den Nutzer zu öffnen ist. Das Schwerste an der Erstellung dieser Komponente war die Animation zwischen offen und geschlossen. Wenn ich keine Höhe festgelegt habe, wurde der Inhalt ohne Überschneidungen dargestellt. Wenn ich diesen jedoch schließe, dann verschwindet der Inhalt, wodurch die Höhe auf 0 + padding⁴⁵ sinkt. Ich wollte erreichen, dass diese Container nur zur Seite wegklappen, doch die Höhe bestehen bleibt. Mit JS habe ich keine Möglichkeit, die Höhe von Elementen, welche relativ angeordnet sind, zu bestimmen. Das hat mir lange Probleme bereitet. Letztendlich bin ich auf eine ungefähre Berechnungsmöglichkeit gestoßen, welche ich zusammen mit einem von mir getesteten Referenzwert angebe, um das nächstmögliche Ergebnis zu erhalten. Diesen Komponenten habe ich ebenfalls für das Planetenmenü genutzt, jedoch ist dieses immer standardmäßig zugeklappt. Durch diese Überprüfung schließt sich der Container (links) automatisch, sobald dieser aufgrund der Bildschirmgröße mit der Sternenkarte kollidieren würde.

Eine weitere Herausforderung lag in der Änderung der Bildschirmgröße. Wenn sich die Bildschirmgröße oder die Skalierung geändert hat, haben sich die auf der Karte (fest) platzierten Planeten verschoben, da diese über absolute Koordinaten platziert wurden. Um dieses Problem zu lösen, habe ich folgenden Code geschrieben. Da ich schon häufig Code erklärt habe, werde ich dieses Mal diesen nicht erklären. Dieser ist nur zur Vollständigkeit aufgelistet, da dieser sonst nur schwer zu finden ist. Die grundlegende Funktion ist die Koordinatentransformation durch Skalieren, wenn eine Bildschirmgrößenänderung stattfindet.

⁴³ Prettier ist ein Programm, welches Code so formatiert, dass bestimmte Richtlinien oder Vorschläge eingehalten werden.

⁴⁴ Eslint ist ein Werkzeug, welches problematische Stellen und Muster identifiziert und kennzeichnet. In Kombination mit anderen Programmen wie Prettier, kann dieses Tool auch automatisch Änderungen vornehmen, um diese Stellen zu beheben. Dabei finden keine Logik-, sondern nur Formatierungsänderungen statt.

⁴⁵ Padding ist eine CSS-Eigenschaft, welche den Abstand zwischen Kindselementen angibt.

```

504     handleResize() {
505         // skipcq: JS-0049
506         if (this.checkEmpty(this.$refs["entireDisc"])) return;
507         this.center = {
508             x: this.$refs["entireDisc"].offsetWidth / 2,
509             y: this.$refs["entireDisc"].offsetHeight / 2,
510         };
511         this.reevaluatePositions();
512     },
513     reevaluatePositions() {
514         this.planets.forEach((p) => {
515             if (!this.checkEmpty(p.adaptedSize)) {
516                 p.position = {
517                     x: p.position.x * (this.adaptedSize / p.adaptedSize),
518                     y: p.position.y * (this.adaptedSize / p.adaptedSize),
519                 };
520                 p.adaptedSize = Math.max(this.adaptedSize, 0);
521             }
522         });
523     },

```

Abbildung 15 - Planeten Skalierungscode

Als letztes werde ich in diesem Abschnitt noch zeigen/erklären, wie ich feststelle, wann sich die Bildschirmgröße geändert hat.

Zum einen erstelle ich einen Event Listener für das Event „resize“, wenn ich die App mounte⁴⁶. Dieser ruft meine Methode „handleResize“ auf (siehe Abbildung 15). Diese führt die benötigten Aktionen aus, sodass die App weiterverwendet werden kann. Den gleichen Schritt mache ich in einem mixin namens „default.“ Ein mixin ist eine Möglichkeit eine andere Datei so einzubinden, als wäre es dieselbe Datei. Jedoch kann diese an vielen Stellen verwendet werden, wodurch sich diese gut für Tools eignet. Sobald nun ein „resize“ Event eintritt, ändere ich meine „__resizeUUID“. Diese kann ich an anderen Stellen nutzen, um festzustellen, ob ein Event aufgetreten ist. Wenn sich diese UUID⁴⁷ ändert, führe ich eine Methode aus, welche die Bildschirmgröße neu berechnet. Der Code zu dieser mixin Datei kann hier (<https://bit.ly/3T0JP7q>) gefunden werden.

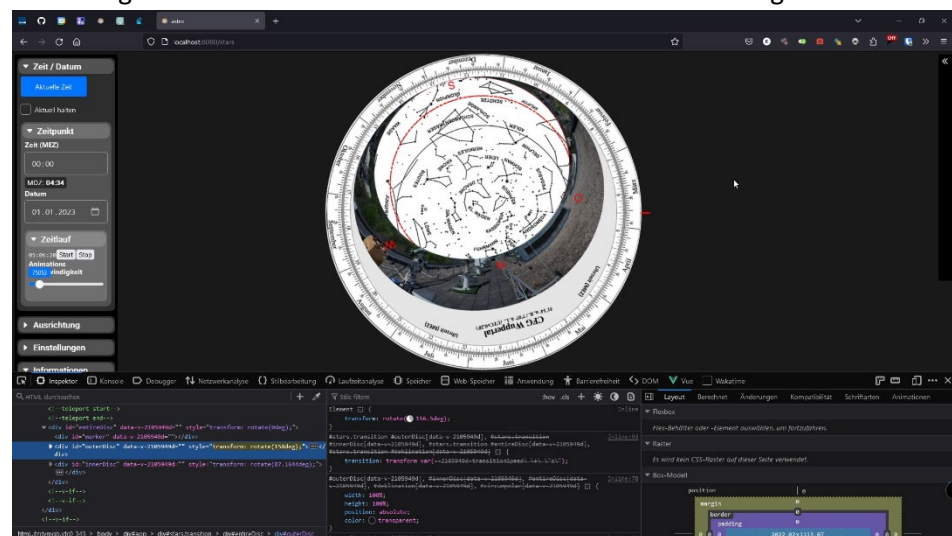
Um die Einleitung dieses Abschnitts zu bewahrheiten, muss ich noch meine Verwendung von Dropdowns/Details erzählen. Nicht nur aus Platzgründen, sondern auch zur übersichtlichen Strukturierung, nutze ich meinen eigenen Details-Komponenten (<https://bit.ly/3OK8YAE>). Die Verwendung kann in *Abbildung 10 - Sternenkarte + alle Extras* im linken Menü eingesehen werden.

⁴⁶ Das Mounten der App ist das Einbinden der App. Erst nachdem diese gemountet ist, können Aktionen ausgeführt werden.

⁴⁷ Eine UUID ist eine Nummer zur Identifikation von Daten. Diese kann universell eingesetzt werden. Ein Beispiel für eine UUID wäre: „e3507863-83a6-4b1f-ad9f-47041d716ef9“

Simulation des Zeitlaufes (abgebrochen)

Die Simulation des Zeitlaufs wäre eine schöne optionale Ergänzung zur Sternkarte gewesen, da man sich so die Veränderung des Nachthimmels besser vorstellen und anschauen könnte. Leider wurde aus dieser Idee nichts. Diese Funktion oder Eigenschaft, ist bis jetzt (Stand: Ende Februar) das einzige Problem, welches ich nicht lösen konnte. Natürlich gab es zwischendurch mal kleinere Probleme (eigentlich immer, da dies ein Bestandteil des Programmierens ist (Problemlösung)). Jedoch ist mir in dieser gesamten Arbeit bis zu diesem Punkt es nicht einmal untergekommen, dass ich ein Problem nicht lösen konnte. In diesem Fall ist es kein fehlendes Wissen oder Können, sondern die Begrenzung der mir zur Verfügung stehenden Technologien. Wie Sie aus den vorherigen Abschnitten und Kapiteln wissen, habe ich die App so entwickelt, dass diese im Browser und somit mit minimalen Ressourcen und auf fast allen Geräten (selbst auf einem Fernseher) ausgeführt werden kann. Meine vorherigen Probleme und Funktionen, konnte ich alle mit den mir durch den Browser zur Verfügung stehenden Ressourcen bewältigen und umsetzen. Bei der Simulation des Zeitlaufes bin ich zunächst davon ausgegangen, dass diese Ressourcen ausreichen würden, doch da habe ich mich getäuscht. Eigentlich ist es keine schwere und auch keine leistungsaufwendige Aufgabe. Die Umsetzung bestand in der je nach Skalierung angegebenen Addition auf einen Zeitstempel, welcher an zuvor definierte Funktionen für die Formatierung und die Anwendung auf die Karte übergeben wurde. Bei der Synchronisierung mit der echten Zeit mache ich genau dasselbe, bis auf den Unterschied, dass ich dort den Zeitstempel nicht manuell erhöhe. Mein Code hat auch einwandfrei funktioniert, jedoch hat der Browser die Änderungen nicht richtig dargestellt. Ich gehe davon aus, dass die Änderungen zu schnell für den Browser erfolgten. Jedoch war eine weitere Reduzierung der Aktualisierungsgeschwindigkeit nicht möglich, da die Kreisbewegung ansonsten weniger einem Kreis und mehr einem Sechseck entsprochen hätte. Ich hätte noch die Möglichkeit gehabt, ab einer bestimmten (getesteten) Geschwindigkeit, die äußere Karte (Zeit) zu fixieren und nur noch die Datumskarte weiterzudrehen. Allerdings wäre diese Toleranz so niedrig gewesen, dass sich dieses Problem auf die Datumskarte übertragen hätte. Wenn sich eine Karte lediglich linear mit einer festgelegten Geschwindigkeit dreht, dann kann man diese Bewegung genauso gut mit der Hand (per Ziehen) erstellen. Da der Browser nicht die benötigten Kapazitäten zur Umsetzung hatte, muss ich leider sagen, dass ich diese Funktion nicht in der App (als einzige) umsetzen konnte. Der Quellcode zu dieser Funktion, welcher nicht gemerged⁴⁸ wurde, kann natürlich trotzdem unter diesem Link (<https://bit.ly/3IhImEx>) eingesehen werden. Zur besseren Veranschaulichung des Problems habe ich dieses Video (<https://youtu.be/eohcLZYyud0>) aufgenommen und auf YouTube aufgrund des zu hohen Speicherplatzverbrauches hochgeladen. Hier ist das Video noch ins PDF-Dokument eingebettet:



⁴⁸ Mergen ist ein Prozess in der Softwareentwicklung, welcher mit dem VCS (Version Control System) git zusammenhängt. Dabei werden Änderungen mit dem Hauptcode oder mit anderen Änderungen zusammengeführt.

Sonnenkuppel

Diese Besondere Lernleistung besteht aus 2 großen Teilen. Der erste Teil ist die zuvor beschriebene Sternenkarte, welche ca. $\frac{2}{3}$ des praktischen Teils ausmacht. Der zweite Teil ist die digitale/virtuelle Sonnenkuppel, welche die restlichen $\frac{1}{3}$ ausmacht. Die digitale Sternenkarte wurde überwiegend von Herrn Koch gewünscht und gefördert. Bei der Sonnenkuppel hat diese Rolle der Physiklehrer Herr Winkhaus übernommen. Er hat das Projekt vorgeschlagen, mir nötige Ressourcen beschafft und die relevanten Formeln bereitgestellt. Da der Quellcode für die digitale Sonnenkuppel nicht auf GitHub zu finden ist, kann dieser über folgenden Link (<https://bit.ly/3J5I2JD>) heruntergeladen werden.

Das physische Modell der Sonnenkuppel wurde von der Studentin Gianna-Maria Praum angefertigt. Dieses Modell wird am CFG dazu genutzt, um den Schattenwurf der Sonne je nach Jahres- und Tageszeit zu simulieren (siehe Abbildung 17). Dazu wird eine Taschenlampe an zuvor errechnete Punkte auf einer Plexiglas-Kuppel gehalten. Daher stammt auch der Name „Sonnenkuppel“. Wie auf den Bildern zu sehen ist, befindet sich auf dem Boden des Modells ein Polarkoordinatensystem, welches das Ablesen der Werte erlaubt. In der Mitte des Modelles befindet sich ein Holzstäbchen (Hindernis), welches den Schattenwurf provoziert.



Abbildung 16 - Schattenwerfer (Sonnenkuppel)

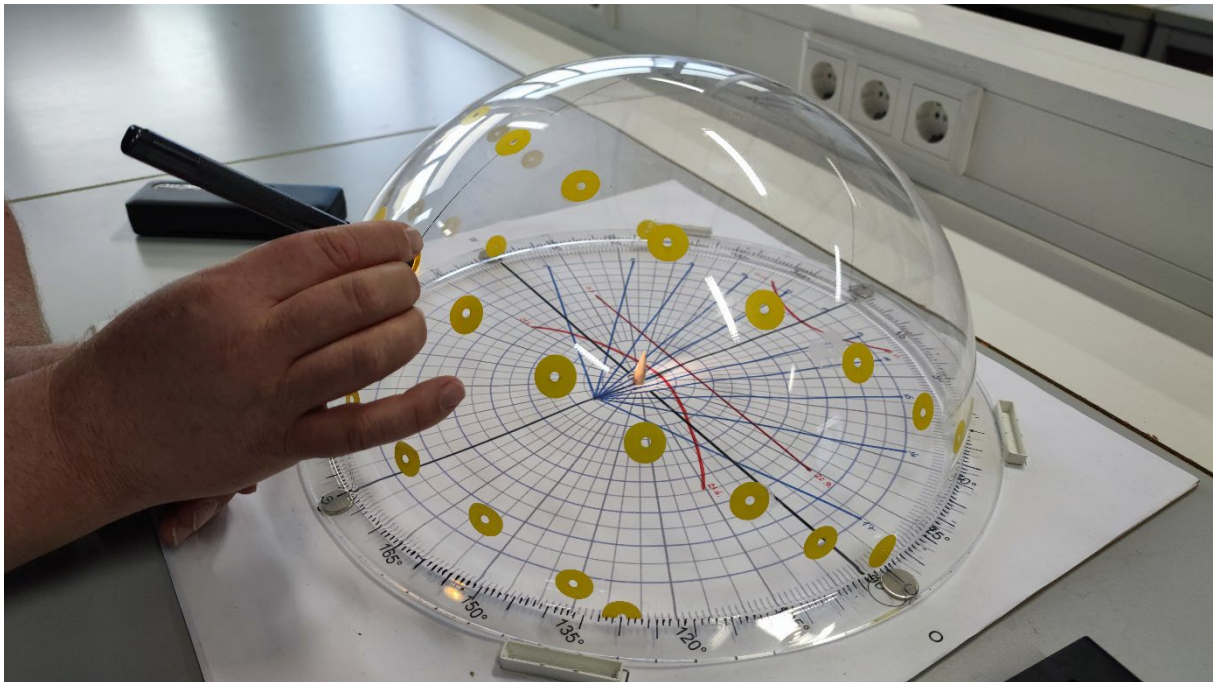


Abbildung 17 - physisches Modell der Sonnenkuppel

Der Grund für die Digitalisierung dieses Modelles war, dass mit dem Modell nur eine kleine Gruppe an Schülern dieses nutzen und damit experimentieren konnte. Durch die Konvertierung können nun alle Schüler gleichzeitig individuell experimentieren und Aufgaben erledigen. Außerdem hat die digitale Version nicht nur eine festgelegte Anzahl von möglichen Positionen für die Lichtquelle, sondern eine freie Auswahl an Zeitpunkten über eine numerische Texteingabe oder einen Schieberegler. Zusätzlich wird nicht nur der Schatten simuliert, sondern auch die atmosphärischen Auswirkungen des Sonnenstandes, wie die Abenddämmerung oder Nacht. Ergänzend visualisiert diese Anwendung das Phänomen, dass der Schatten ab einem gewissen Punkt sich endlos schnell bewegt und endlos lang wird. In der Realität nehmen wir dieses zwar nicht wahr, da die Sonne in diesen Fällen so tief steht, dass wir nur die Abwesenheit des Lichtes feststellen, dennoch ist es (meiner Meinung nach) faszinierend.

Zielsetzung

Die Zielsetzung der digitalen Sonnenkuppel ist die Dezentralisierung und Optimierung des physischen Modells durch die Replizierung in einer digitalen Welt. Der Nutzer soll in der Lage sein, den Zeitpunkt, den Ort (geographische Breite) und die Höhe des Schattenwerfers (in der App Obelisk genannt, da dieser durch einen sphärischen Obelisk dargestellt wird) einzustellen. Die Anwendung soll basierend auf diesen Daten die Position der Sonne berechnen, den Schatten visualisieren und (optional) den Verlauf des Schattens anzeigen. Außerdem soll der Nutzer in der Lage sein, die Kamera frei zu bewegen, um die Ergebnisse besser zu verstehen.

Dadurch kann der Sonnenverlauf im Jahreslauf nachvollzogen werden. Für Interessierte: Die Ergebnisse können zum Beispiel durch die App Google Wetter oder andere Anbieter überprüft werden (siehe Abbildung 18). Um das zu tun, kann man sich den Verlauf der Sonne für den aktuellen Tag in der digitalen Sonnenkuppel anzeigen lassen. Dort achtet man auf folgende Werte; zum einen auf den Sonnenaufgang, also sobald der Schatten sichtbar ist/ der Verlauf nicht mehr ins Unendliche geht. Das gleiche gilt für den Sonnenuntergang und für den Sonnenhöchststand, also wenn der Schatten am kürzesten ist/ Richtung Norden zeigt. Wenn man diese Werte mit den Werten einer Wetterapp vergleicht, wird man feststellen, dass diese (bis auf eine kleine Abweichung, da die Geographische Breite in der App nur auf 2 Nachkommastellen angegeben wird) übereinstimmen.

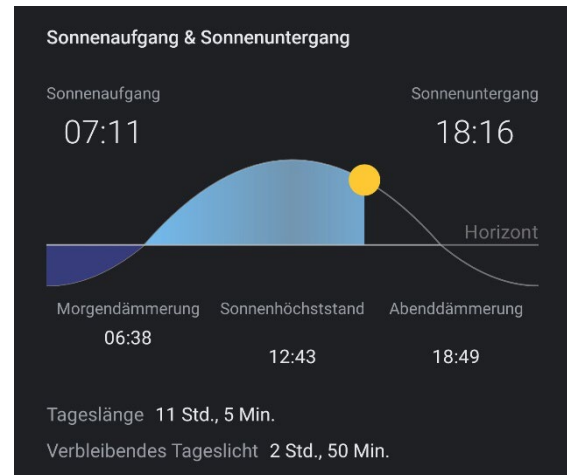


Abbildung 18 - Beispiel Google Sonnenverlauf

Ansatz

Um diese Idee zu realisieren, musste ich mir zunächst ein passendes Framework aussuchen, da diese Ansprüche zu komplex und außerhalb des machbaren Bereiches für Vue.js liegen. Diesen Schritt habe ich bereits im Abschnitt *Webtechnologien* auf Seite 6 ab der Hälfte der Seite erwähnt. Da ich sowas wie die Simulation und Animation der Sonne in Unity noch nie gemacht hatte, musste ich dort erstmal etwas rumprobieren, um zu entscheiden, wie ich mit der Entwicklung fortfahre. Dabei stellte sich heraus, dass die Sonne in Unity unendlich weit weg ist, weshalb die Positionsänderung dieser keinen Unterschied macht. Die einzige Möglichkeit die Sonne zu verändern ist die Anpassung der eulerschen Winkel der Sonne, weshalb die Transformation auf Azimut und Höhe zurückläuft. Also brauchte ich eine Funktion oder Gleichung, welche basierend auf dem Eingabedatum mir die Höhe und den Azimut errechnet. Um den Schatten zu werfen, nutzte ich ein einfaches 3D-Modell und habe den Rest Unitys Raytracing⁴⁹-System überlassen. Für die Kamera habe ich zuerst ein öffentliches „Free-Flight-Cam“ Skript verwendet. Jedoch habe ich dieses im Laufe der Entwicklung so weit angepasst, dass es nun mein eigenes ist. Über die Darstellung des Verlaufes habe ich mir zu Beginn keine Gedanken gemacht, da ich zuerst die Basis umsetzen wollte. Um die Karte auf den Boden zu projizieren, hatte ich mir vorgenommen, diese als SVG nachzuzeichnen, damit ich diese beliebig groß skalieren kann. Weitere Informationen folgen im nächsten Abschnitt.

Entwicklung und Probleme

Vorab: Da ich mit der Entwicklung von Unity Spielen deutlich weniger Erfahrung als mit der Entwicklung von Webanwendungen mit Vue.js habe, werden in diesem Kapitel, deutlich mehr Überraschungen und

⁴⁹ Raytracing ist ein Verfahren, um den Verlauf eines Strahles zu simulieren und mögliche Kollisionen herauszufinden. Wenn man dieses dann noch etwas erweitert, kann man realistisch Licht simulieren.

temporärer Rückschläge als in der Entwicklung der Sternenkarte vorkommen. Jedoch gehören Rückschläge zu der Softwareentwicklung dazu und ohne Fehler lernt man nichts. Außerdem werde ich nun hier schon einmal (global) ein Bild der fertigen App (Standardansicht) anhängen, damit ich auf dieses im Folgenden verweisen kann, ohne dieses (aufgrund von Platzgründen) erneut zu zeigen.

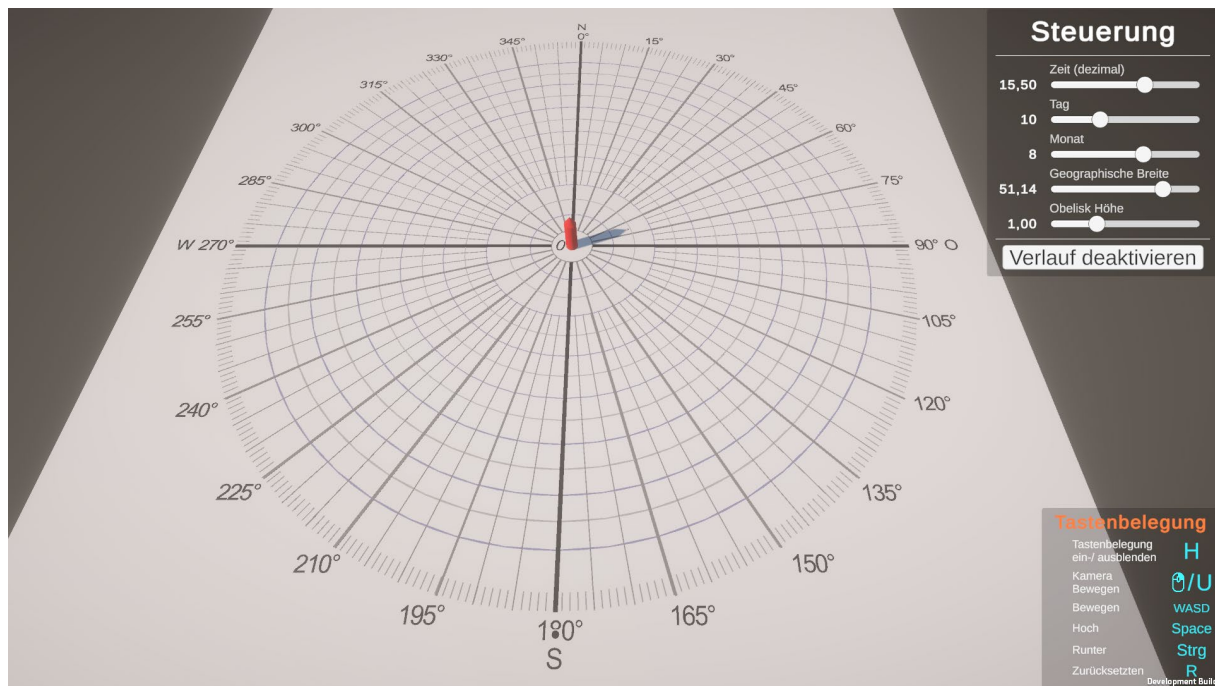


Abbildung 19 - Sonnenkuppel (Standardansicht)

Grundeinstellungen und Erstellung der Basis

Da ich mich nun auf Unity festgelegt hatte, musste ich mich bei der Projekterstellung dafür entscheiden, welche Render-Engine ich verwende. Die Standardoption wäre die „standard render pipeline“. Diese ist für einfache Projekte nützlich, welche beinahe keine Ressourcen benötigen. Dann gibt es noch URP („universal render pipeline“). Diese ermöglicht eine höhere Qualität und Realismus, jedoch ermöglicht diese kein Pathtracing⁵⁰. Die letzte Option ist HDRP („high definition render pipeline“), welche sehr realistische und gute Ergebnisse liefert, jedoch auf keinem Fall im Web, sondern nur auf leistungsstarken PCs ausgeführt werden kann. Ich habe mich für URP entschieden, da ich realistische Schatten, aber auch eine hohe Kompatibilität brauche.

Als nächstes habe ich eine einfache Szene zum Testen der Einstellungen erstellt. Dafür hatte ich meine Grundfläche, einen Kegel als Schattenwerfer, die Kamera und die Sonne. Da die Schatten noch nicht scharf waren, habe ich viel recherchiert und mit den Lichteinstellungen experimentiert, bis ich ein gutaussehendes Ergebnis erzielt hatte. Dabei war der Schlüssel Soft Shadows. Auch wenn man es nicht erwartet, sind diese aufgrund des Anti-Aliasing schärfer als Hard Shadows. Danach habe ich die URP-Einstellungen angepasst. Normalerweise gibt es verschiedene Detailstufen, welche je nach Geräteleistung ausgewählt werden und angeben, wie qualitativ hochwertig das Spiel⁵¹ dargestellt werden soll. Da jedoch jede Einstellung unter Maximum die Schatten verfälscht, habe ich jede Qualitätsstufe bis auf

⁵⁰ Pathtracing ist eine nochmals erweiterte Form des Raytracings. Diese ermöglicht so realistische Ergebnisse, dass diese häufig nicht mehr von der realen Welt zu unterscheiden sind, es benötigt aber auch sehr viel Leistung und teure Hardware.

⁵¹ Auch wenn ich hier eine Anwendung/Simulation erstelle, ist Unity eine Game-Engine, entwickelt, um Spiele zu erstellen.

die höchste gelöscht, sodass immer diese gewählt wird. Mit diesem Aufbau war die Basis gegeben, wodurch ich mit der weiteren Entwicklung fortfahren konnte.

Kamerakontroller

Wie bereits im Abschnitt *Ansatz* erwähnt, habe ich für den Beginn des Projektes ein frei verfügbares Kameraskript verwendet. Der Grund dafür war, dass vor allem Player-Controller, wozu die Kamera auch gehört, in der Regel alle gleich sind (sofern die Art der Bewegung (laufen, fliegen, rollen, etc.) gleich ist). Da ich weniger Erfahrung in der Spieleentwicklung, als in der Webentwicklung besitze, habe ich mir diesen Schritt gespart, da es keine der Hauptfunktionen/Ziele der App ist und man muss das Rad ja auch nicht neu erfinden. Das Prinzip zur Bewegung ist einfach. Wenn die richtigen Tasten (WASD+Space+Strg) gedrückt werden, wird auf die Position der Kamera der entsprechende Vektor mit der Länge abhängig von der Geschwindigkeit addiert. Da diese Überprüfung (auf Tastendruck) bei jedem Update-Tick, also jedes Mal, wenn das Bild neu gezeichnet wird, neu überprüft wird, entsteht eine flüssige Bewegung. Die Rotation ist etwas aufwendiger, da die Transformation dieser in zwei Teile geteilt werden muss. Andernfalls könnte man sich während einer Bewegung nur horizontal oder vertikal bewegen, da es sonst einen Fehler in der Vektorrechnung gäbe. Da wir Menschen aber solche linearen Bewegungen mit der Maus nicht wirklich umsetzen können und die Steuerung ja interaktiv sein soll, werden beide Methoden separiert. Weil der Code zur Sonnenkuppel aufgrund der Größe des Unity-Projektes nicht auf GitHub publiziert und so nur schwer einzusehen ist, habe ich in den folgenden zwei Bildern beide Teile des Codes (Transformation und Rotation) abgebildet. Erklären werde ich diese hier nicht, da es nur einen Überblick geben soll und die Rotation sehr viel Text aufgrund der Komplexität in Anspruch nehmen würde.

```
204 // Movement
205 if (enableMovement)
206 {
207     Vector3 deltaPosition = Vector3.zero;
208     float currentSpeed = movementSpeed;
209
210     if (Input.GetKey(boostSpeed))
211         currentSpeed = boostedSpeed;
212
213     if (Input.GetKey(KeyCode.W))
214         deltaPosition += transform.forward;
215
216     if (Input.GetKey(KeyCode.S))
217         deltaPosition -= transform.forward;
218
219     if (Input.GetKey(KeyCode.A))
220         deltaPosition -= transform.right;
221
222     if (Input.GetKey(KeyCode.D))
223         deltaPosition += transform.right;
224
225     if (Input.GetKey(moveUp))
226         deltaPosition += transform.up;
227
228     if (Input.GetKey(moveDown))
229         deltaPosition -= transform.up;
230
231     // Calc acceleration
232     CalculateCurrentIncrease(moving.deltaPosition != Vector3.zero);
233
234     transform.position += deltaPosition * currentSpeed * _currentIncrease;
235 }

237 // Rotation
238 if (enableRotation)
239 {
240     // Pitch
241     transform.rotation *= Quaternion.AngleAxis(
242         angle - Input.GetAxis("Mouse Y") * mouseSense,
243         Vector3.right
244     );
245
246     // Paw
247     transform.rotation = Quaternion.Euler(
248         transform.eulerAngles.x,
249         transform.eulerAngles.y + Input.GetAxis("Mouse X") * mouseSense,
250         transform.eulerAngles.z
251     );
252 }
```

Abbildung 20 - Kamerarotation

Eine der großen Anpassungen, welche ich am Kameraskript vorgenommen habe, ist das Fokusverhalten. In der ursprünglichen Version wurde die Maus sofort gefangen und man musste eine Taste (U) drücken, um diese wieder freizuschalten. In vielen Spielen ist das sicherlich hilfreich, da jedoch ein wichtiger Aspekt in dieser Anwendung die Interaktion mit dem User Interface ist, erleichtert meine Veränderung die Nutzung sehr. Die Veränderung liegt in der Umstellung des Fokussierungsverhaltens. Vorher wurde ein einfacher boolean⁵² genutzt, welcher bei Tastendruck (U) auf false gesetzt wurde. Jetzt nutze ich immer noch diesen boolean, jedoch setzte ich diesen nur auf true, sofern der Nutzer die rechte Maustaste gedrückt hält. Dadurch besteht der Fokus nun in der Veränderung der Werte und nicht auf der Verschiebung der Kamera, da diese

Abbildung 21 - Kameratransformation

Die Veränderung liegt in der Umstellung des Fokussierungsverhaltens. Vorher wurde ein einfacher boolean⁵² genutzt, welcher bei Tastendruck (U) auf false gesetzt wurde. Jetzt nutze ich immer noch diesen boolean, jedoch setzte ich diesen nur auf true, sofern der Nutzer die rechte Maustaste gedrückt hält. Dadurch besteht der Fokus nun in der Veränderung der Werte und nicht auf der Verschiebung der Kamera, da diese

⁵² Ein boolean ist ein primitiver Datentyp, welcher nur zwei Zustände besitzt: true und false == wahr oder falsch. Dieser wird für konditionale Verzweigungen und andere logische Vergleiche genutzt.

Funktion nur Nebensache ist. Während der Entwicklung ist mir noch ein mögliches Problem eingefallen. Ich nutze an meinem PC zwar eine Maus, jedoch werden die Astronomie-Laptops vorwiegend über das Touchpad benutzt. Daraufhin habe ich die Kamerabewegung auf meinem Laptop getestet, wobei mir aufgefallen ist, dass die Navigation zwar möglich ist, aber es nicht gerade angenehm ist, die ganze Zeit auf dem Touchpad zu verkrampfen, um die Kamera zu bewegen. Deswegen habe ich noch einen alternativen Modus eingebaut. Dieser wird mit der Taste „U“ aktiviert und deaktiviert. Dadurch kann die App jetzt auch auf Laptops nutzerfreundlich verwendet werden.

Die Positionierung der Kamera möchte ich auch noch in diesem Abschnitt ansprechen, da es auch zum Arbeitsprozess gehört und diese Besondere Lernleistung vom Leser als Anleitung für die erweiterte Benutzung verwendet werden mag. Am Anfang habe ich die Kamera so positioniert, dass der Schatten gut zu erkennen war und man eine dynamische Sicht auf die Szene hatte. Jedoch hatte man keine gute Übersicht und man musste die Kameraposition zwangsweise verändern, um die Werte für andere Ergebnisse zu sehen. Aus diesem Grund habe ich, nachdem die Skala am Boden fertig war, die Kamera (mittels des eigenen Kamerakontrollers) so positioniert, dass man direkt beim Laden der App eine gute Übersicht hat. Um nach der Veränderung der Kamera zu dieser Ansicht zurückzukehren, kann man einfach die Taste „R“ drücken.

Berechnung der Sonnenposition

Die Berechnung der Sonnenposition war für mich einfach, da ich die benötigten Formeln von Michael Winkhaus nach einem Gespräch zu den Rahmenbedingungen der App und der benötigten Werte von ihm ausgerechnet und zur Verfügung gestellt bekommen habe. Nach der Implementation der Funktionen basierend auf seinen Formeln, sind mir jedoch einige Fehler aufgefallen. Nach einer Weile des Suchens, wurde mir das Problem klar. Jede der mathematischen Funktionen wie Sinus oder Cosinus etc. erwarten die übergebenden Werte in Radianen, nicht in Grad. Da sich die trigonometrischen Funktionen in den Formeln miteinander verketteten, stimmen die Ergebnisse vorne und hinten nicht. Deswegen habe ich jede benötigte trigonometrische Funktion selbst definiert, damit in dieser zusätzlich an der richtigen Stelle die Umrechnung in Grad stattfindet. Nachdem ich diese Änderungen angewendet habe, funktionierten die Funktionen und die Berechnung war erledigt. Beim Experimentieren mit der App ist mir jedoch ein Fehler aufgefallen. Wenn die Zeit genau auf 12 Uhr steht, dann invertierte sich die Richtung des Schattens. Nach genauerer Betrachtung der Formeln ist mir dann die Problemstelle aufgefallen. Das hier ist die Formel für den Südazimut, wobei ϕ für die geographische Breite, δ für die Sonnendeklination und z für die Zeit steht.

```
return 2 * Atan((Cos(Height()) + Cos(phi) * Sin(delta) - Sin(phi) *  
Cos(delta) * Cos((z - 12) * 15)) / (Cos(delta) * Sin((z - 12) * 15)));
```

Da in dieser Formel an dieser Stelle: \uparrow durch 0 geteilt wird, da $\uparrow 12 - 12 \times 15 = 0$ gilt und $\sin(0) = 0$ ist, tritt hier ein Problem auf. Deswegen habe ich über dieser Zeile die Bedingung eingefügt, dass wenn $z == 12$ ist, dass z dann auf 12.0001 erhöht wird. Diese minimale Abweichung ist in der App nicht bemerkbar, korrigiert aber die Formel.

Übrigens: Diese Information könnte zwar auch zum Abschnitt *User Interface* passen, jedoch geht es dabei auch um Daten. Wenn Sie die App nutzen, können Sie die Taste „i“ drücken um die „Development Console“ zu öffnen. Diese habe ich so angepasst, dass hilfreiche Werte bei Veränderung in dieser angezeigt werden. Diese Funktion kann zur Verifizierung der App oder zur automatischen Berechnung der Werte für einen festgelegten Zeitpunkt genutzt werden. Diese Tastenkombination ist nicht in dem Menu Tastenbelegung zu finden, da eine mögliche Aufgabe für die Schüler sein könnte, die Werte anhand des Polarkoordinatensystems abzulesen.

Erstellung des Obelisken

Wie bereits im Abschnitt *Grundeinstellungen und Erstellung der Basis* erwähnt, habe ich in den Anfangszeiten der digitalen Sonnenkuppel einen Kegel verwendet, welcher als Schattenwerfer fungierte. Nach einer der ersten Revisionen hat jedoch Herr Winkhaus angemerkt, dass es besser wäre, einen Obelisken wie bei realen Modellen (siehe Abbildung 22) zu nutzen, da der Schatten dann klarer zu erkennen ist. Dabei war mir freigestellt, ob ich einen Obelisken mit quadratischer Grundfläche oder einen mit einer kreisförmigen Grundfläche erstelle. Da Unity im URP-Framework mit der Einstellung „Soft-Shadows“, unschärfere Ergebnisse, in Bezug auf den Schatten, bei einer quadratischen Grundfläche liefert, habe ich den Obelisken mit einer kreisförmigen Grundfläche konstruiert. Für die Erstellung habe ich das kostenfreie Open-Source 3D-Programm „Blender“ verwendet. Eine erweiterte Beschreibung des Funktionsumfangs von Blender kann in meiner vorherigen Arbeit (der Projektarbeit im Fach Astronomie aus dem Jahr 2022/2023) eingesehen werden. Es gibt in diesem Programm sogenannte „Mesh-Primitives“. Dazu gehören Würfel, Kugel, Zylinder, Kegel und Torus. Aus diesen einfachen Objekten kann durch das Zusammenfügen oder durch die Veränderung der Topologie ein komplexeres 3D-Modell erstellt werden. Für die Erstellung des kreisförmigen Obelisken, habe ich das Mesh-Primitive eines Zylinders gewählt und die obere Fläche parallel zur Grundfläche auf der Z-Achse nach oben extrudiert. Die neu entstandene Fläche habe ich mit dem Faktor 0 multipliziert/skaliert. Dadurch wurde die neue Mantelfläche am obersten Punkt in der Mitte zusammengefügt. Anschließend habe ich die Blender-Funktion „Merge by Distance“ verwendet, um die Topologie des Objektes so aufzuräumen, dass die sich am obersten Punkt überlappenden Punkte zu einem Punkt zusammengefügt werden. Der Schritt mag unwichtig erscheinen, da keine sichtbare Änderung geschieht, jedoch ist dieser Schritt für die Optimierung, gerade bei größeren 3D-Modellen, sehr wichtig. Als letztes habe ich die mittlere Fläche (ursprünglich die Oberseite des Zylinders) so weit nach oben geschoben, bis ich mit dem Winkel und der Größe der angewinkelten Mantelfläche zufrieden war.



Abbildung 22 - Obelisk (Halde Hohward)

Als letztes musste ich das 3D-Modell nur noch exportieren (als FBX⁵³), um es in Unity verwenden zu können. Dieses habe ich eingebettet und getestet, es hat funktioniert. Dann habe ich mich noch mit dem Ziel der Höhenveränderung des Obelisken beschäftigt. Es gibt zwar Möglichkeiten, mittels Unity die Geometrie eines Objektes zu verändern, jedoch wäre das sehr aufwendig zu programmieren und zusätzlich der Leistung der App nicht gerade dienlich. Deswegen habe ich das ganze viel einfacher gemacht. Ich habe das 3D-Modell noch einmal bearbeitet. Dabei habe ich die unterste Fläche (Grundseite), deutlich nach unten verschoben, wodurch der Obelisk höher wurde. Anschließend habe ich die Höhe des Objektes in Unity so angepasst, dass es bei der normalen Höheneinstellung (1m) so aussah wie vorher und tatsächlich 1m hoch war. Daraufhin habe ich eine Funktion geschrieben, welche basierend auf der Höheneinstellung den Obelisken nach oben verschiebt. Da noch viele Meter Geometrie unter dem Polarkoordinatensystem versteckt sind, wirkt es so, als würde sich der Obelisk verändern, auch wenn dieser nur nach oben verschoben wird.

Damit war der Obelisk erfolgreich dynamisch in Unity eingebunden. Fun Fact: Sie können mit der Kamera unter das Polarkoordinatensystem fliegen, um das zuvor beschriebene Verhalten selbst zu erkunden.

⁵³ Das Dateiformat FBX ermöglicht den nahezu verlustfreien Austausch von Informationen zwischen 3D-Programmen. Meiner Erfahrung nach ist es die beste Möglichkeit für den Export von 3D-Modellen von Blender für Unity oder Unreal-Engine.

User Interface

Auch wenn man es nicht erwartet, war das User Interface zusammen mit dem Polarkoordinatensystem der schwerste Teil dieser Anwendung. Für die Slider wollte ich Unitys eigenen Slider Komponenten verwenden, jedoch bietet dieser keine Möglichkeit für die automatische Anzeige des Wertes in Form von Text. Außerdem wollte ich das Eingeben von konkreten Werten über eine Texteingabe ermöglichen, das funktioniert mit diesem ebenfalls nicht. Deswegen habe ich mir mein eigenes Slider Prefab für die Reproduktion erstellt. Ein Prefab ist in Unity ein Objekt, welches ein voreingestelltes Objekt oder eine Sammlung von Objekten angibt. Diese können als Vorlage eingesetzt werden, um komplexere Systeme einfacher zu erstellen. Außerdem erkennt ein Unity Prefab automatisch Änderungen an dessen Instanzen. Diese können nach Genehmigung des Entwicklers gespeichert und automatisch auf andere Instanzen dieses Prefabs übernommen werden.

Für meinen eigenen Slider habe ich einen normalen Unity Slider als Basis genommen. Zusätzlich habe ich ein „Slider Controller“ Skript geschrieben, welches an diesen Slider angehängt ist. Dieses bekommt über einen Unity Parameter den dazugehörigen Text (ein TMPro⁵⁴ Objekt) übergeben, welches sich ebenfalls in dem Slider Prefab befindet und links neben dem Slider angeordnet ist. Das Skript übergibt dem Unity Editor einen Parameter für eine Funktion, welche mit dem neuen Wert bei einer Werteänderung von dem Slider aufgerufen werden soll. Dadurch kann dieses Prefab nach der Instanziierung für verschiedenste Methoden und Werte genutzt werden, da immer eine andere Funktion übergeben werden kann. Das zuvor erwähnte TMPro Objekt befindet sich in einem „Text Area“ Objekt, wodurch dieses vom Nutzer für die genauere Eingabe von Werten genutzt werden kann. Die Funktionsweise des Slider Controller Skriptes ist ziemlich simpel. Es verbindet das Textfeld mit dem Slider. Zu Beginn setzt es den Inhalt des Textfeldes auf eine formatierte Version des Wertes des Sliders. Im weiteren Verlauf existieren drei Funktionen: („InvokeOnChange“, „SliderChanged“, „ValueChanged“). Die Funktion InvokeOnChange ruft die individuell übergebende Funktion zur Übermittlung der Werte auf. Die Funktion SliderChanged wird von dem Unity Slider automatisch bei einer Änderung aufgerufen. Diese verändert den Inhalt des Textfeldes, um die Werte dem Nutzer anzuzeigen. Je nachdem, ob der Slider als Ganzzahl oder Gleitkommazahl gewertet wird, wird der Text in dieser Funktion anders formatiert. Die Funktion ValueChanged interpretiert den Wert aus dem Textfeld und aktualisiert den Slider, damit die Werte übereinstimmen. Zusätzlich ruft diese Funktion die Funktion InvokeOnChange auf, damit die geänderten Werte an den Manager gesendet werden. Die Methode InvokeOnChange wird nur hier aufgerufen, da auch wenn der Slider verändert wird, die Methode ValueChanged aufgerufen wird, da der Text von der Methode SliderChanged geändert wird. So wird ein doppelter Aufruf und somit Leistung eingespart.

Es befindet sich noch ein weiteres Objekt in dem Slider Prefab. Es ist ein Textobjekt, welches als Label fungiert. Dieses wird benötigt, da der Nutzer andernfalls nicht wüsste, welchen Wert er verändert. Der Textinhalt dieses Labels kann auch über das Prefab individuell angepasst werden.

Dieses Slider Prefab habe ich mehrfach für die verschiedenen Eingabemöglichkeiten instanziiert und angepasst. Die restlichen UI-Elemente sind einfache Text-, Sprite⁵⁵- und Buttonobjekte. Für das Einklappen des Tastenbelegungsmenüs habe ich einen Animation Controller in Kombination mit zwei Animation Clips genutzt. Diese verschieben das Menü nach unten (aus dem Viewport hinaus) oder nach oben

⁵⁴ TMPro (Text Mesh Pro) ist die moderne Methode, wie Unity Text darstellt. Es ermöglicht deutlich mehr Kontrolle, wie z.B. die Schriftart und viele andere Optionen gegenüber Unitys altem Text System.

⁵⁵ Man nennt ein Bild ein Sprite (in der Spielebranche), wenn es als Objekt, beispielsweise im User Interface, verwendet wird.

(wieder hinein). Die flüssige Animation wird automatisch durch die Animation Clips gewährleistet, da diese automatisch zwischen zwei Werten „easen“.

Darstellung des Polarkoordinatensystems

Wie Sie sicherlich festgestellt haben (siehe *Abbildung 19 - Sonnenkuppel (Standardansicht)*), spielt die Darstellung des Polarkoordinatensystems eine wichtige Rolle, da andernfalls die Schattensimulation ohne eine nutzbare Skala sinnlos wäre. Während der anfänglichen Planung, Gewichtung und Priorisierung der verschiedenen Aspekte der App habe ich diese als leicht, aber aufwendig eingestuft. Aus diesem Grund habe ich mich in der Entwicklungsphase zuerst mit den anderen Funktionen beschäftigt, da diese schwerer und entscheidend über das mögliche Scheitern dieses Projektes waren. Die Reihenfolge der Funktionen der Sonnenkuppel in diesem Dokument stimmt mit der Reihenfolge der Entwicklung der Funktionen überein.

Ich hatte die Karte als Raster Grafik (PNG) gegeben. Mir war bewusst, dass ich dieses als SVG neu zeichnen muss, da ich diese Grafik auf einen Durchmesser von 20m strecken muss. Mit der gegebenen Auflösung von ca. 1000px * 1000px wäre das Polarkoordinatensystem nicht mehr zu erkennen gewesen. Als ich dann mit der Entwicklung begonnen hatte, nutzte ich Adobe Illustrator, um jeden Strich, Kreis, Zahl Stück für Stück als Vektorgrafik nachzuzeichnen. Als ich zu 80% fertig war, bemerkte ich, dass Adobe Illustrator ein Polarkoordinatenwerkzeug beinhaltet, welches mir den Großteil meiner bereits erledigten Arbeit abgenommen hätte. Ich blieb dann jedoch beim manuellen Verfahren, da ich so mehr Kontrolle über die Grafik hatte. Damit das Polarkoordinatensystem gleichmäßig blieb, habe ich sehr häufig die Option „Transformation wiederholen“ genutzt. So musste ich zum Beispiel die äußeren Linien nur einmal um den Mittelpunkt rotieren und in der gleichen Aktion kopieren. Diese Transformation konnte ich dann so häufig wiederholen, bis das Polarkoordinatensystem vollständig war. Dadurch wurde mir viel Zeit erspart, jedoch habe ich für die (erste) Erstellung dennoch ca. 2 Stunden benötigt, da ich alles so akkurat und perfekt wie möglich machen wollte. Nach der Fertigstellung des SVGs wollte ich dieses kopieren und in Unity einbinden. Doch das funktionierte nicht. Das hat mich sehr gewundert, da SVGs mittlerweile von jedem modernen Programm, vor allem in der 3D-Branche, unterstützt werden. Daraufhin habe ich lange recherchiert, warum das nicht funktioniert hat. Unity unterstützt keine SVGs, da es die Inhalte direkt an die Grafikkarte sendet und die Grafikkarte SVGs nicht unterstützt, da diese auf zu aufwendigen Berechnungen basieren. Diese Hürde hat mich ordentlich zurückgeworfen, da das Polarkoordinatensystem notwendig ist, ich jedoch zu diesem Zeitpunkt keine funktionierende Möglichkeit sah, dieses einzublenden. Ich hatte die Idee, das SVG als ein so enorm hochauflösendes Bild zu exportieren, dass man dieses verwenden konnte. Jedoch erlaubt Adobe Illustrator keine größeren Exporte als 24000 * 24000px. Diese Option habe ich versucht, jedoch wäre ein Bild mit mindestens 4-facher Auflösung erforderlich gewesen. Deswegen habe ich andere Programme getestet, ob diese ein höher auflösendes Bild erlauben. Ich hatte eines gefunden, jedoch war das Resultat beschädigt und konnte nicht verwendet werden, außerdem war dieses Bild mittlerweile 1.2GB groß. Auch wenn dieses genügt hätte, konnte ich diese Option nicht verwenden, da ich diese Anwendung im Browser zur Verfügung stelle und nicht erwarten kann, dass sich jeder Nutzer (jedes Mal) 1.2GB runterladen muss. Somit musste ich nach weiteren Alternativen suchen. Nach langem Suchen habe ich ein Unity Plugin gefunden, welches das Darstellen von SVGs ermöglichen soll. Das hat auch funktioniert, jedoch nutzt dieses einen Unlit⁵⁶-Shader, wodurch es für meinen Verwendungszweck nicht zu gebrauchen war, da der Schatten das wichtigste Element ist. Ich habe versucht, den Shader so anzupassen, dass dieser Lit ist, also auf Licht reagiert, jedoch waren diese Änderungen nicht mit dem SVG-Plugin kompatibel.

⁵⁶ Unlit bedeutet, dass das mit diesem Shader ausgestattete Objekt nicht auf Licht reagiert. Es erscheint auch wenn kein Licht vorhanden ist, als vollständig ausgeleuchtet, zeigt keine Schatten an und reagiert nicht auf die Umgebung.

Deswegen habe ich versucht, dass SVG ohne Hintergrund zu exportieren, damit ich eine normale Plane⁵⁷ unter das SVG legen kann, sodass der Schatten auf dieser Fläche angezeigt wird. Das hat leider auch nicht funktioniert, da der Shader automatisch einen Hintergrund hinzufügt. Deswegen hatte ich probiert, die Plane genau überlappend mit dem SVG zu legen, in der Hoffnung, dass die Schatten von der Plane dargestellt und die Linie vom SVG übernommen werden. Das scheiterte ebenfalls, es entstanden nur irritierende Interferenzen. Ich hatte noch über den Ansatz des hochauflösenden Bildes, welches ich in viele kleine Teile zerlege, nachgedacht, jedoch hätte die Gesamtgröße wieder bei 1.2GB gelegen. Dann kam mir die Idee: Ich erstelle das Polarkoordinatensystem als 3D-Modell, da ich weiß, dass Unity diese darstellen kann und diese auch sehr speichereffizient sind.

Es gibt die Möglichkeit, in Blender aus einem SVG ein 3D-Mesh zu erstellen. Da dieser Prozess jedoch nicht so gut mit so komplexen SVGs funktioniert, habe ich in Adobe Illustrator sechs verschiedene Ebenen erstellt, welche die verschiedenen Sichtbarkeits-/Prioritätsebenen darstellen. Dann habe ich das SVG gruppiert und die verschiedenen Elemente den verschiedenen Ebenen zugeordnet. Diese Ebenen habe ich jeweils separat als SVG exportiert. Diese habe ich anschließend nacheinander in Blender importiert, umgewandelt korrigiert⁵⁸ und wieder zusammengefügt. In jeder Iteration habe ich die Ebenen basierend auf deren Hierarchie unterschiedlich hoch extrudiert. Dadurch stehen höhere Ebenen im Vordergrund, auch wenn diese mit unteren überlappen.

Nach dem Export aus Blender und dem Import in Unity, hat das Polarkoordinatensystem endlich funktioniert. Das Objekt musste ich dann in Unity nur noch so skalieren, dass es gut in die Szene passt. Wenn man mit der Kamera dicht an den Boden fliegt, kann man die Höhenunterschiede sogar noch nachvollziehen (siehe Abbildung 23). Diese Höhendifferenz ist jedoch so gering, dass diese keine Auswirkungen auf den Schatten hat.

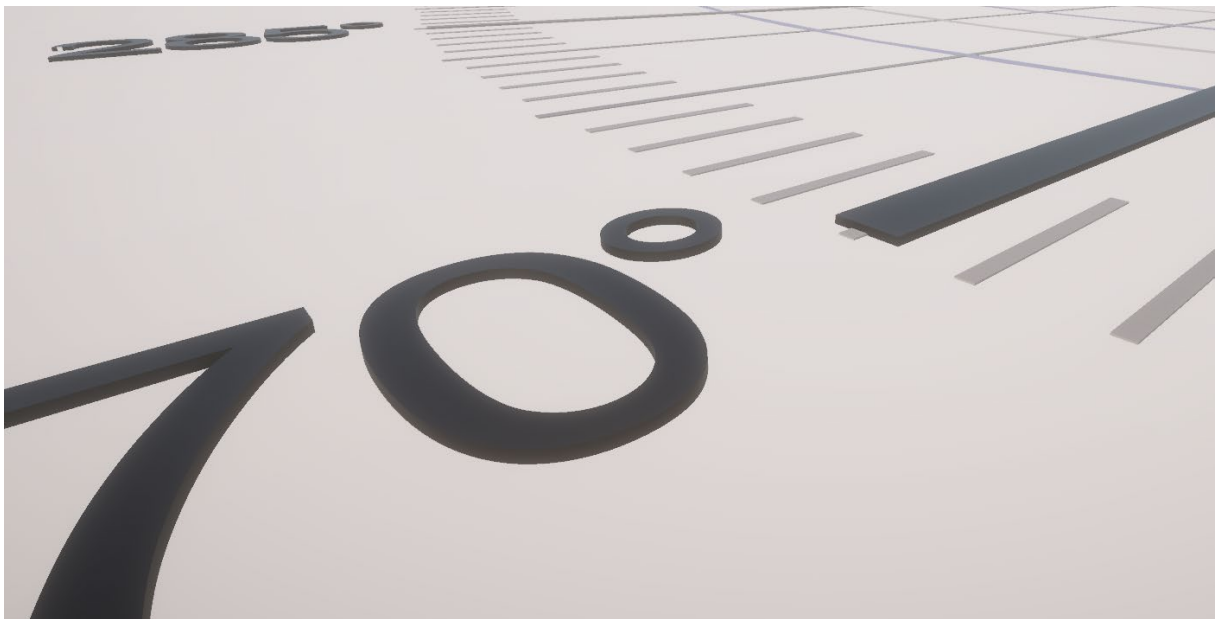


Abbildung 23 - Höhendifferenz in der Sonnenkuppel

⁵⁷ Eine Plane ist ein 3D-Objekt, welches eine einfache Ebene ohne Tiefe darstellt. Also nimmt die Tiefe auf einer Achse immer 0 an.

⁵⁸ Das Korrigieren ist hier wichtig, da das Umwandeln eines SVGs in ein 3D-Mesh kein fehlerfreier Prozess ist und so häufig fehlerhafte Geometrien entstehen, welche dann von Hand korrigiert werden müssen.

Weitere Kleinigkeiten

Mit den oben genannten abgeschlossenen Funktionen konnte die App nun verwendet werden und war sozusagen fertig. Allerdings gibt es immer Verbesserungspotential. Eine Idee war die zusätzliche Darstellung des Schattenverlaufes. Diese habe ich umgehend umgesetzt. Dafür habe ich ein bisschen getrickst, da die Konstruktion einer solchen Verlaufslinie ziemlich schwierig ist. Ich habe den Unity Komponenten „Trail Renderer“ verwendet. Dieser kann an ein Objekt angehängt werden, woraufhin dieser bei einer Bewegung des Objektes eine Spur im dreidimensionalen Raum einzeichnet, im Verlauf, wie sich dieses Objekt bewegt hat. Nun habe ich nur noch ein sich bewegendes Objekt benötigt, welches der Position des Schattens entspricht. Zum Glück hatte mir Herr Winkhaus eine zusätzliche Formel für die Berechnung der x- und z-Koordinate bereitgestellt. Für die y-Koordinate konnte ich einfach 0 einsetzen, da der Boden eben ist. In meinem Manager-Skript aktualisiere ich dann die Position einer kleinen Kugel, welche mit dem Trail-Renderer verbunden ist, auf diese Schattenposition. Dadurch kann man sich den Sonnenverlauf anzeigen lassen. Die Spur des Trail-Renderers verschwindet wieder nach 5 Sekunden, da sonst die Szene ziemlich voll und unübersichtlich wird. Mit der Taste „p“ kann man das Verschwinden der Spur pausieren. Dadurch kann man sich die zuvor berechnete Spur besser anschauen und analysieren.

Die Einbettung des Unity Spiels in meiner Vue.js Webanwendung werde ich hier nun kurz erläutern. Wie bereits im Abschnitt *Webtechnologien* erwähnt, nutze ich WebGL, um Unity Spiele im Browser anzuzeigen. Deswegen muss ich Unity auch so einstellen, dass es das Spiel im WebGL Format exportiert. Dabei ist die Option „Development Build“ wichtig. Ein Dev-Build ist zwar etwas größer, langsamer und eigentlich nicht für die Produktion empfohlen, jedoch muss diese Option aktiviert sein, da Unity andernfalls den Build verschlüsselt und ich diesen dann nicht mit „unity-webgl“ einbetten kann. Zusätzlich ist beim „builden“ wichtig, dass die Komprimierung deaktiviert ist, da sonst die App am Ende die Schatten schlechter darstellt. Als nächstes musste die Einbettungsmöglichkeit in der Webanwendung geschaffen werden. Dafür habe ich mir drei Dateien erstellt. Als erstes die Datei „SonnenkuppelView.vue“ (<https://bit.ly/3uWxqZ4>), welche vom Vue Router aufgerufen wird und als einzigen Inhalt die Datei „FullscreenUnityRenderer.vue“ (<https://bit.ly/3uW5vbj>) mit dem Ordernamen „Sonnenkuppel“ aufruft, welche die Datei „UnityRenderer.vue“ (<https://bit.ly/4cacRZD>) aufruft und diese als Vollbild anzeigt. Die Datei UnityRenderer.vue beinhaltet die gesamte Logik, um einen Unity Build darzustellen.

Diese Einbettung habe ich so modularisiert, da nicht nur das Sonnenkuppel Unity-Projekt, sondern auch das Pacman Unity-Projekt auf die gleiche Weise eingebettet wird. So brauche ich für die verschiedenen Projekte jeweils nur eine Zeile, in der ich einen String anpasse, um anzugeben, welcher Ordner genutzt werden soll.

Zusätze

CI/CD

CI/CD steht für „Continuous Integration mit Continuous Deployment“. Es beschreibt die automatisierte Bereitstellung und ermöglicht eine zuverlässigere und flüssigere Entwicklung durch die automatische Durchführung von Tests. Es gibt verschiedene Systeme, welche die Umsetzung eines solchen Aktionsnetzwerkes erlauben. Ich habe mich in diesem Projekt für GitHub-Actions entschieden, da ich den Code ohnehin durch das Versionskontrollsystem git auf GitHub verwalte und die Nutzung einfach und kostenlos ist. Außerdem habe ich mit diesem System bereits durch kleinere Projekte Erfahrung gesammelt. GitHub speichert die entsprechenden Programme in sogenannten „workflows“. In diesem Projekt nutze ich die in Abbildung 24 aufgelisteten workflows. Ich werde die Funktionen dieser im Folgenden erklären.

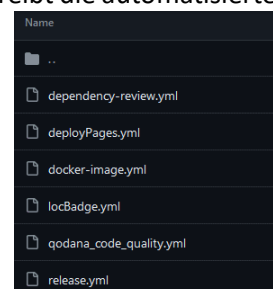
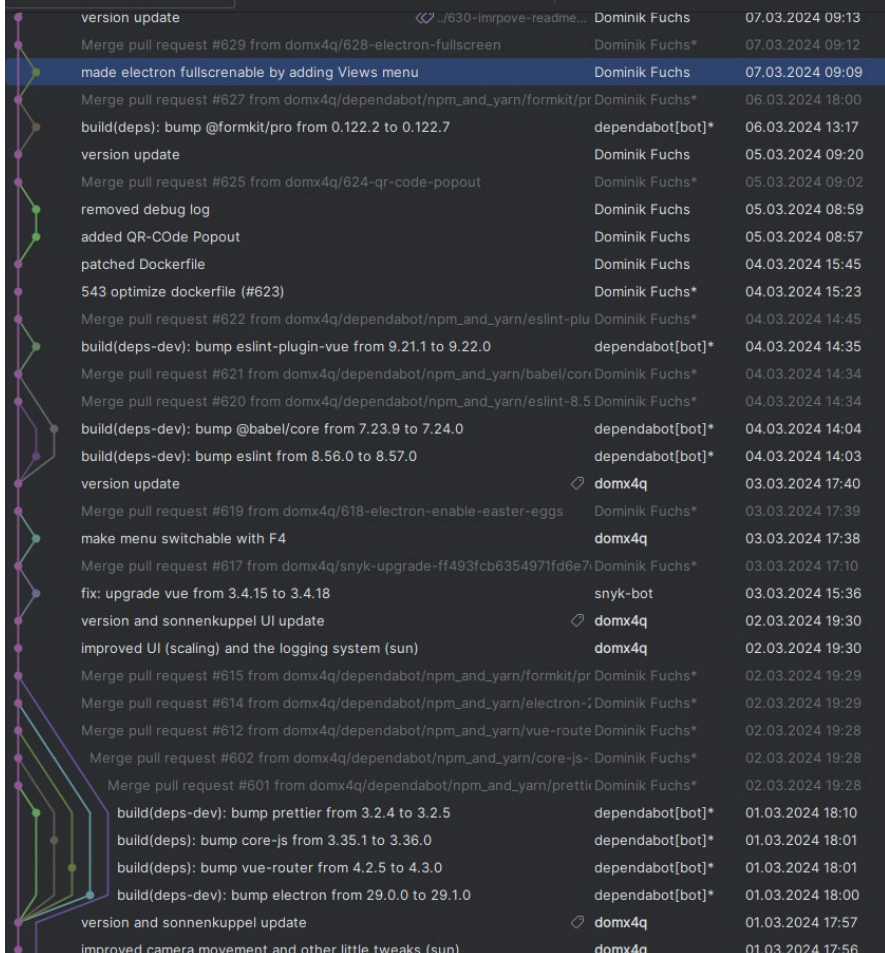


Abbildung 24 -
GitHub workflows

Die individuellen Konfigurationsinhalte können über diesen Link (<https://bit.ly/4c8MsLY>) eingesehen werden. Die oberste Datei habe ich, sowie die zweitletzte Datei, nicht selbst verfasst. Die Datei „dependency-review.yml“ stammt von GitHub selbst und wird für jedes Projekt empfohlen. Diese wird bei einem neuen Pull-Request ausgeführt und überprüft, ob die eingeführten Änderungen, Abhängigkeiten einführen, welche zu bekannte Sicherheitsschwachstellen führen können. Da dieser Workflow als erforderlich in meinen Konfigurationen eingestellt ist, können keine neuen Änderungen übernommen werden, welche das Projekt gefährden würden. An dieser Stelle habe ich als Repository-Eigentümer das Veto-Recht, welches mir im Notfall trotzdem das Übernehmen von Änderungen erlaubt. Ein Pull-Request ist Teil des git VCS⁵⁹. Dieser gibt an, dass ein Autor seine angefertigten Änderungen in einen anderen Branch, meist in den master/main/production Branch, mit dem merge-Befehl zusammenfügen möchte. Der merge-Befehl vergleicht zwei Versionen des Codes und versucht beide Seiten ohne Konflikte zusammenzufügen. Es kann dabei zu sogenannten Kollisionen kommen, wenn beide Version relativ zum Ursprung an der gleichen Stelle eine Änderung enthalten. In dem Fall muss dann ein Entwickler manuell entscheiden, welche Seite übernommen wird oder wie er beide Änderungen ohne Fehler kombinieren kann. Alle gängigen VCS nutzen das Branching-Modell. Dabei erstellt man für die Entwicklung neuer Funktionen oder die Lösung eines Problems immer wieder Abzweigungen von der nächsthöheren Quelle (in meinem Fall, da ich allein arbeite und keine anderen Deployment-Branche außer dem „master“ Branch habe, immer vom „master“ Branch aus). In der folgenden Abbildung (25) sieht man einen solchen git-Verlauf der letzten Tage aus dieser Repository.

Zu beachten ist, dass die Namen „Dominik Fuchs“ und „domx4q“ beide meinen Account bezeichnen, da mein GitHub Nutzernamen domx4q lautet und mein echter Name natürlich Dominik Fuchs ist. Je nach dem, auf welchem Gerät (Laptop, Web oder PC) ich an der App programmiere, wird entweder mein richtiger Name, oder mein Nutzername verwendet. Der Nutzer dependabot ist ein automatisierter Nutzer von GitHub, welcher mir automatisch Pull-Requests erstellt, wenn eine neue Version einer verwendeten Abhängigkeit verfügbar ist. „snky-bot“ ist ein weiteres automatisches System. Dieses erledigt die gleichen



Commit Message	Author	Timestamp
version update	Dominik Fuchs	07.03.2024 09:13
Merge pull request #629 from domx4q/628-electron-fullscreen	Dominik Fuchs*	07.03.2024 09:12
made electron fullscreenable by adding Views menu	Dominik Fuchs	07.03.2024 09:09
Merge pull request #627 from domx4q/dependabot/npm_and_yarn/formkit/pr	Dominik Fuchs*	06.03.2024 18:00
build(deps): bump @formkit/pro from 0.122.2 to 0.122.7	dependabot[bot]*	06.03.2024 13:17
version update	Dominik Fuchs	05.03.2024 09:20
Merge pull request #625 from domx4q/624-gr-code-popup	Dominik Fuchs*	05.03.2024 09:02
removed debug log	Dominik Fuchs	05.03.2024 08:59
added QR-CODE Popup	Dominik Fuchs	05.03.2024 08:57
patched Dockerfile	Dominik Fuchs	04.03.2024 15:45
543 optimize dockerfile (#623)	Dominik Fuchs*	04.03.2024 15:23
Merge pull request #622 from domx4q/dependabot/npm_and_yarn/eslint-plu	Dominik Fuchs*	04.03.2024 14:45
build(deps-dev): bump eslint-plugin-vue from 9.21.1 to 9.22.0	dependabot[bot]*	04.03.2024 14:35
Merge pull request #621 from domx4q/dependabot/npm_and_yarn/babel/core	Dominik Fuchs*	04.03.2024 14:34
Merge pull request #620 from domx4q/dependabot/npm_and_yarn/eslint-8.5	Dominik Fuchs*	04.03.2024 14:34
build(deps-dev): bump @babel/core from 7.23.9 to 7.24.0	dependabot[bot]*	04.03.2024 14:04
build(deps-dev): bump eslint from 8.56.0 to 8.57.0	dependabot[bot]*	04.03.2024 14:03
version update	domx4q	03.03.2024 17:40
Merge pull request #619 from domx4q/618-electron-enable-easter-eggs	Dominik Fuchs*	03.03.2024 17:39
make menu switchable with F4	domx4q	03.03.2024 17:38
Merge pull request #617 from domx4q/snyk-upgrade-ff493fcb6354971fd6e7	Dominik Fuchs*	03.03.2024 17:10
fix: upgrade vue from 3.4.15 to 3.4.18	snyk-bot	03.03.2024 15:36
version and sonnenkuppel UI update	domx4q	02.03.2024 19:30
improved UI (scaling) and the logging system (sun)	domx4q	02.03.2024 19:30
Merge pull request #615 from domx4q/dependabot/npm_and_yarn/formkit/pr	Dominik Fuchs*	02.03.2024 19:29
Merge pull request #614 from domx4q/dependabot/npm_and_yarn/electron-;	Dominik Fuchs*	02.03.2024 19:29
Merge pull request #612 from domx4q/dependabot/npm_and_yarn/vue-route	Dominik Fuchs*	02.03.2024 19:28
Merge pull request #602 from domx4q/dependabot/npm_and_yarn/core-js-	Dominik Fuchs*	02.03.2024 19:28
Merge pull request #601 from domx4q/dependabot/npm_and_yarn/pretti	Dominik Fuchs*	02.03.2024 19:28
build(deps-dev): bump prettier from 3.2.4 to 3.2.5	dependabot[bot]*	01.03.2024 18:10
build(deps): bump core-js from 3.35.1 to 3.36.0	dependabot[bot]*	01.03.2024 18:01
build(deps): bump vue-router from 4.2.5 to 4.3.0	dependabot[bot]*	01.03.2024 18:01
build(deps-dev): bump electron from 29.0.0 to 29.1.0	dependabot[bot]*	01.03.2024 18:00
version and sonnenkuppel update	domx4q	01.03.2024 17:57
improved camera movement and other little tweaks (sun)	domx4q	01.03.2024 17:56

Abbildung 25 - git-log (01.03 - 07.03)

⁵⁹ Version Control System (auf Deutsch: Versionskontrollsystem). Dieses erlaubt die Überwachung und Verwaltung von Änderungen in einem Projekt.

Aufgaben wie dependabot, jedoch erstellt dieser nur einen Pull-Request, wenn zu einer meiner Abhängigkeiten eine Sicherheitslücke bekannt wird. Dieser aktualisiert diese Abhängigkeit dann automatisch auf die nächstsichere Version und stellt mir Informationen über die Sicherheitslücke bereit.

An diesem Beispiel kann man sehr gut das Branching-System nachvollziehen. Mein „master“-Branch wird von dem violetten Strang repräsentiert. Alle Änderungen gehen von diesem aus und werden auch wieder nach einem genehmigten Pull-Request in diesen zurückführen. Im unteren Viertel des Bildes sieht man eine Schachtelung. Diese hat nichts Besonderes zu bedeuten, sondern gibt nur an, dass mehrere Branches mit Änderungen nacheinander (unterster zuerst) erstellt und dann in der gleichen Reihenfolge gemerged wurden. Man sollte optimaler Weise, keine Änderungen direkt in den „master“-Branch pushen, jedoch lässt es sich in diesem Projekt aufgrund der Build-Artefakte von Unity und der Aktualisierung der Version nicht vermeiden. Da ich allerdings allein arbeite und Funktionen nacheinander implementiere und merge, sorgt es hier, solange ich nur solche Dateien wie package.json o.ä. ändere, für keine Probleme. Die verschiedenen ‚Ausartungen‘, sind die einzelnen Branches und deren Änderungen. Diese müssen nicht immer wieder zurück zur Quelle laufen. Da ich jedoch jedes meiner hier sichtbaren Features bereits gemerged habe, ist es hier nicht anders zu sehen.

Kommen wir wieder zurück zu den GitHub-Action Workflows. Der Workflow „deployPages.yml“ ist sehr wichtig. Dieser publiziert die App bei einer in den „master“-Branch übernommenen Änderung. Da ich nach der Fertigstellung dieser Besonderen Lernleistung aufgrund von laufenden Kosten, voraussichtlich meinen V-Server, erreichbar unter der Domain „cloudster.online“, kündigen werde, ist der Fortbestand dieser App relevant, weswegen ich GitHub Pages nutze, um diesen zu gewährleisten. Für die Publizierung der App legt der Workflow zunächst die Umgebungseigenschaften fest. Dabei gebe ich an, dass dieser unter der neusten Ubuntu Version mit der Node Version 16.14.0 laufen soll. Als nächstes nutze ich die „checkout“ Aktion, um der Laufzeitumgebung Zugang zu der neusten Version meines Codes zu geben. Anschließend bereite ich mich auf den Build-Prozess durch die Installation der „node-modules“ vor. Daraufhin führe ich den Build-Prozess mit den Umgebungsvariablen „NODE_ENV: production“ und „iAmOnGithubPages: true“ durch. Dadurch wird ein Produktionsreifer Build mit dem modifizierten Router für GitHub erstellt. Abschließend lade ich den Build auf GitHub Pages hoch und weise GitHub an, dass sie mein Pages-Deployment aktualisieren sollen. Dadurch befindet sich immer eine aktuelle und zuverlässige Version meiner App im Internet (ohne zusätzliche Serverkosten).

Der Workflow der Datei „docker-image.yml“ wird nur dann ausgeführt, wenn eine Docker-konfigurationsrelevante Datei geändert wird oder dieser manuell ausgeführt wird. Wenn dies geschieht, dann erstellt der Workflow eine aktuelle Version des Docker-Images, welches unter dem Versionstag „autoGit“ mithilfe meine verschlüsselt gespeicherten Zugangsdaten zum Dockerhub hochgeladen wird, so dass das Docker-Image aktualisiert wird. Dadurch wird automatisch ein immer aktuelles Docker-Image bereitgestellt, weshalb die Server-Installation auf weiteren Geräten ohne Probleme funktioniert.

Der Workflow „locBadge.yml“ berechnet die Zeilen an Code, welche dieses Projekt beinhaltet. Auch wenn in vielen Programmiersprachen die Anzahl der Codezeilen (mittlerweile) keine Rolle mehr spielt, werden diese Zahlen dennoch angegeben, um die Größe und den Aufwand eines Projektes einzuschätzen. Um die Anzahl an Codezeilen darzustellen, erstellt dieser Workflow eine SVG-Datei, welche anschließend auf GitHub Pages hochgeladen wird, um diese dann in der README.md Datei als Bild einzubetten. Dabei werden durch die folgende Zeile bestimmte Dateien und Verzeichnisse ignoriert, da diese generierten Code und nicht selbstgeschriebenen Code enthalten.

```
ignore: "node_modules|dist|archive|unity|package*.json"
```

Die daraus entstehende SVG-Datei sieht dann beispielsweise so aus:

Lines of Code **9,432** *Abbildung 26 - LoC Badge Beispiel*

Der LoC (Lines of Code) Workflow wird bei jeder übernommenen Änderung in den „master“-Branch ausgeführt. Zusätzlich kann der Workflow durch eine manuelle Nutzerinteraktion gestartet werden.

Wie zuvor bereits angesprochen, stammt der „qodana“ Workflow nicht von mir. Dieser wird benötigt, um eine Integration von JetBrains anzusteuern, welche den Code auf Schwachstellen überprüft. Deswegen werde ich die weitere Erklärung unterlassen und mit dem selbstgeschriebenen Workflow „release.yml“ fortfahren. Dieser Workflow wird ausgeführt, sobald ein neuer „tag“ erkannt wird. Ein tag wird im Versionskontrollsystem git dafür genutzt, um Markierungen für neue Versionen festzulegen. Wenn nun ein neuer tag erkannt wird, beginnt die Ausführung dieses Workflows in einer sogenannten Matrix. Diese sorgt für die wiederholte Ausführung der Aktionen, jedoch unter verschiedenen Bedingungen. In meinem Fall wechsele ich das Betriebssystem für die zweite Ausführung auf Windows, sodass die zu erstellende Electron Desktopanwendung sowohl unter Linux als auch unter Windows verwendet werden kann. Nach der Fertigstellung des Builds werden die Build-Artefakte⁶⁰ als neuer Draft⁶¹-Release hochgeladen. Diesen muss ich dann nur noch genehmigen und schon ist automatisch eine neue und aktuelle Version der App erstellt.

Die oben aufgelisteten Systeme nehmen natürlich Zeit und Aufwand in der Entwicklung dieser in Anspruch, jedoch erleichtern diese den weiteren Entwicklungsverlauf des Projektes ungemein. Zusätzlich gewährleisten diese einen sicheren und sauberen Code. Ein weiterer Vorteil von CI/CD ist, dass diese Systeme nicht jedes Mal wieder neu entwickelt, sondern häufig projektübergreifend übernommen werden können. Dadurch wird der Zeitaufwand noch weiter reduziert.

Bereitstellung (Electron und Docker)

Wie bereits im Abschnitt *Auswahl der Technologien* erwähnt, habe ich mich für das Cross-Platform Modell entschieden, damit so viele Nutzer wie möglich die App nutzen können. Da ich die App als Webanwendung programmiert habe, können bereits sehr viele Nutzer diese ohne Probleme nutzen. Jedoch muss nun eine aktive Internetverbindung gegeben sein, um die App aufrufen zu können und es muss ein Server, diesen Inhalt bereitstellen. Damit die Bereitstellung der App einfacher und zuverlässiger ist, habe ich ein Docker-Image erstellt, welches mit einem einzelnen Befehl installiert und gestartet werden kann. Dadurch können Institutionen wie Schulen eine lokale und zuverlässige Version der App hosten, um den internen Zugang zu gewährleisten⁶². Der aus dem Docker-Image entstehende Docker-Container, läuft autark und aktualisiert sich selbst, wodurch dieser nach der initialen Einrichtung nicht mehr angepasst werden muss.

Das Docker Setup besteht aus drei relevanten Dateien (für die Produktionsumgebung). Die erste Datei ist die sogenannte „Dockerfile“ (<https://bit.ly/3TzNaKO>). Diese definiert den Aufbau des neuen „Betriebssystems“⁶³. In der Datei führe ich Schritte aus, welche nicht dynamisch jedoch erforderlich sind. Da diese Datei nur beim Erstellen des Images ausgeführt wird, kann man viel Zeit einsparen, wenn man hier Abhängigkeiten o.ä. installiert, da diese dann nur einmal anstatt bei jedem Start installiert werden. Es muss das Basis Image festgelegt werden (in meinem Fall node:16), welches als Grundlage für das Docker-Image fungiert. Zusätzlich lege ich noch relevante Umgebungsvariablen fest und installiere Werkzeuge (git, screen, openssl). Als letztes lade ich mir das Projekt von GitHub herunter und führe die Datei „dockerEntrypoint.sh“ (<https://bit.ly/48RgWil>) aus.

⁶⁰ Die Resultate von Github-Actions werden Artefakte genannt.

⁶¹ Ein vorzeitiger Release (noch nicht für die Öffentlichkeit einsehbar).

⁶² Am CFG läuft bereits seit ca. einem Jahr ein solcher Docker Server auf einem Astronomie Laptop.

⁶³ Es wird kein richtiges Betriebssystem erstellt, sondern ein Docker-Image, jedoch simuliert dieses das Verhalten verschiedener Betriebssysteme, weshalb diese Analogie passend und für das leichtere Verständnis hilfreich ist.

Diese Datei definiert das Skript, welches bei jedem Containerstart ausgeführt wird und legt somit auch die Funktionen des Containers fest. Im ersten Teil dieser Datei prüfe ich, ob dem Container Zertifikate (Inhalt und Schlüssel) übergeben wurden. Wenn dem nicht so ist, generiere ich ein Zertifikat mit OpenSSL für die Nutzung von TLS für das https Protokoll. Falls dem Container ein hostname übergeben wurde, erstelle ich ein Zertifikat für diesen hostname, andernfalls für den FQDN⁶⁴ „localhost“. Im zweiten Teil der Datei installiere ich das node-module „serve“ und „@vue/cli“. Da ich beide Module für die Ausführung des Buildprozesses benötige, diese jedoch auch aktuell gehalten werden müssen, kann ich diese nur an dieser Stelle installieren. Im dritten und letzten Teil der Datei, findet die eigentliche Logik statt. Dort erstelle ich meine benötigten „screens“, welche ich für die simultane Ausführung mehrerer Prozesse benötige. Das Programm „screen“ ermöglicht die parallele Ausführung von mehreren Konsolen in der reinen Serverumgebung. Es simuliert sozusagen mehrere Bildschirme, auf denen verschiedene Inhalte geöffnet sein können. In meinen screen namens „astro“ sende ich den Befehl für die Ausführung der Datei „updateDaemon.sh“ (<https://bit.ly/3v3dfc1>). Anschließend warte ich auf die Erstellung des screens „AUTO-astro“ von dem zuvor genannten Skript, an den ich mich dann anähne, um die Ausgabe von diesem in meinem Docker-log zu sehen. Das Skript „updateDaemon.sh“ erstellt den screen „AUTO-astro“ und sendet den Befehl zur Ausführung des Skripts „updateServer.sh“ (<https://bit.ly/43o61Mb>) an diesen. Anschließend wird geprüft, ob die lokale Version der Repository auf dem gleichen Stand wie die neuste Version auf GitHub ist. Wenn das der Fall ist, wird nur 5 Minuten gewartet, woraufhin die nächste Überprüfung der gleichen Bedingung stattfindet. Wenn es jedoch neue Änderungen gibt, dann wird die Ausführung des „updateServer.sh“ Skriptes in dem Container „AUTO-astro“ abgebrochen und die erneute Ausführung wird verursacht. Das Skript „updateServer.sh“ aktualisiert durch eine Reihe an Befehlen die lokale Version auf den neusten Stand. Anschließend werden (zur Sicherheit) alle Abhängigkeiten neu installiert und es wird ein neuer Build erstellt. Daraufhin wird die Datei „startServer.sh“ (<https://bit.ly/3lEmvpZ>) ausgeführt. Diese Datei führt dann den eigentlichen Server aus und stellt damit die App der Öffentlichkeit bereit.

Diese Verkettung erscheint zwar auf den ersten Blick sehr unübersichtlich, jedoch ermöglicht diese eine Modulare und zuverlässige Ausführung der App im Docker-Container. Die letzte (dritte) Docker relevante Datei ist die Datei „docker-compose.yml“ (<https://bit.ly/3x5rdKL>). Der Docker-Container kann sowohl über den folgenden (unübersichtlichen) Befehl oder mithilfe dieser Datei erstellt werden.

```
docker run -p 443:3000 -d -it -v
/path/to/certificates/cert.crt:/opt/certs/cert.crt:ro -v
/path/to/certificates/privatekey.key:/opt/certs/cert.key:ro --restart
always domx4q/astro:autoGit
```

Die Docker-Compose Datei beinhaltet die gleichen Informationen, jedoch sind diese hier strukturiert und geordnet (siehe Abbildung 27). Das Compose-Modell bietet noch weitere Vorteile. Das Modell wurde für die einfache Ausführung von größeren Diensten entwickelt, welche nicht nur einen Container, sondern eine Reihe von ineinandergreifenden Containern beinhaltet. Da man nur den Befehl `docker compose up` ausführen muss, ist die Einrichtung solcher Systeme nun deutlich einfacher. Allerdings ist der größte Vorteil meiner Meinung nach (diese App ist auch davon betroffen), das einfache Aktualisieren. In den meisten Fällen wird ein Update nicht intern im Container, sondern durch das Docker-Image übermittelt. Wenn man nun versucht, seinen Container zu aktualisieren, ist das nur sehr schwer möglich, da man seine ursprüngliche Konfiguration verliert, den alten Container löschen muss, einen neuen wieder aufsetzt und den neuen auch Container auch wieder einrichten muss. All diese Arbeit erledigt docker compose selbstständig für einen. Man muss lediglich das neue Image pullen

⁶⁴ Der „Fully Qualified Domain Name“ spezifiziert den vollen Domainnamen, also im Beispiel <https://maps.google.com/index> wäre der FQDN nicht nur google.com, sondern „maps.google.com“.

durch `docker compose pull` und dann den Stack⁶⁵ neu starten durch ein erneutes Ausführen von `docker compose up`. (Die Links hinter den Dateinamen führen zum Quellcode).

```
1  version: "3.3"
2  services:
3    astro:
4      container_name: astro_vue
5      ports:
6        - "3511:3000"
7      environment:
8        - TZ=Europe/Berlin
9      volumes:
10       - "/home/cloudster/certificates/cert.crt:/opt/certs/cert.crt:ro"
11       - "/home/cloudster/certificates/cloudster.online.key:/opt/certs/cert.key:ro"
12     # build:
13     #   context: .
14     #   dockerfile: Dockerfile
15     image: domx4q/astro:autoGit
16     tty: true # -t flag
17     stdin_open: true # -i flag
18     restart: always
19
20     ulimits:
21       nproc: 65535
```

Abbildung 27 - `docker-compose.yml`

Mit diesem Aufbau kann der Server zuverlässig und einfach aufgesetzt werden. Damit der Nutzer/Verbraucher die App noch zugänglicher nutzen kann, habe ich noch ergänzend eine Implementation dieser in dem JavaScript Framework Electron angefertigt. Dieses Framework ermöglicht die Erstellung von Desktopapps aus Webapps. Dadurch muss der Nutzer sich nur einmal die Electron Desktopapp über den folgenden Link (<https://github.com/domx4q/astroProject/releases/latest>) herunterladen und kann diese dann im Anschluss dauerhaft ohne eine vorhandene Internetverbindung nutzen. Da ich in der Desktopapp noch zusätzliche Berechtigungen habe, konnte ich nun zusätzlich einfache Tastenkombinationen implementieren. So kommt man mit Strg + 1, Strg + 2 und Strg + 3 zu den jeweiligen Apps (1=Planeten, 2=Sternenkarte, 3=Sonnenkuppel) gelangen. Diese Optionen sind jedoch auch über das Menü in der linken oberen Ecke unter „Anwendungen“ aufgelistet. Da die Desktopapp ohnehin jede Seite meiner Webseite enthält, habe ich es als Verschwendung angesehen, wenn nur ein Teil der Seiten erreichbar ist. Deswegen habe ich über die (versteckte) Taste F4 die Option geschaffen, in dem Menü auch noch die anderen Apps einzublenden. Da diese jedoch keinen wissenschaftlichen Nutzen haben und meist nur kleine Spielerein sind, habe ich diese standartmäßig versteckt. Wenn man auf Leistungsprobleme im Browser stößt, kann ich diese Desktopapp nur empfehlen. Da diese lokal auf dem Gerät läuft und somit auch mehr Rechenleistung bereitgestellt bekommt, läuft diese deutlich schneller als die Webapp. Da die Electron App nur die Darstellung der eigentlichen Webapp übernimmt, ist mein Quellcode (Datei: `background.js` (<https://bit.ly/3PmDbpM>)) recht kurz und umfasst nur ca. 240 Zeilen. Eine detaillierte Erklärung dessen, werde ich mir hier sparen, da es nicht mehr viel mit dem eigentlichen Ziel dieser Besonderen Lernleistung zu tun hat. Auch wenn der Code kurz ist, ist dieser an manchen Stellen sehr komplex.

⁶⁵ Den Inhalt und das Resultat der `docker-compose.yml` Datei nennt man Stack.

Fazit

Da ich in dieser Besonderen Lernleistung keine wissenschaftlichen Ergebnisse geschaffen habe, sondern Anwendungen für das Verständnis solcher Ergebnisse entworfen habe, werde ich das Fazit für meine persönliche Bewertung dieses Projektes in Bezug auf den Aufwand, Nutzen und die resultierenden Erfahrungen und Triumphe nutzen.

Um ein Projekt dieser Größe erfolgreich umzusetzen, ist eine umfangreiche Planung erforderlich. Dafür habe ich mir zu Beginn der Besonderen Lernleistung eine Grobgliederung für die Organisation der jeweiligen benötigten Komponenten zur Realisierung dieses Projektes erstellt. Dabei habe ich die benötigte Zeit für jedes Feature abgeschätzt und eine Priorisierung geschaffen, um das Projekt erfolgreich zu entwickeln. Ich hatte ca. 1 bis 2 Monate Puffer für mögliche Schwierigkeiten oder Probleme, welche ich aufgrund der Schwierigkeiten mit dem Polarkoordinatensystem und der nutzerfreundlichen Platzierung der Planeten auf der Sternenkarte auch benötigt habe, eingeplant. Zusätzlich habe ich KI-gesteuerte Programme (wie [Reclaim.ai](#)) für die automatische Planung der jeweiligen Termine verwendet, in denen ich an der Besonderen Lernleistung gearbeitet habe. Bei der anfänglichen Planung habe ich einen Fehler begangen. Ich habe die Zeit für das Schreiben dieser schriftlichen Arbeit vollkommen vergessen. Das liegt vermutlich daran, dass ich schon viele Projekte entwickelt habe und mein Fokus auf der Programmierung liegt, weshalb ich diesen Aspekt nicht beachtet habe. Dadurch wurde es nun in den letzten Wochen der Erarbeitung deutlich stressiger, da ich den benötigten Zeitaufwand für den schriftlichen Teil auf keinen Fall so hoch eingeschätzt hätte. Dennoch konnte ich all meine Ziele für diese Besondere Lernleistung umsetzen und bin mit den Resultaten sehr zufrieden.

Der Zeitaufwand war natürlich ziemlich hoch (im Durchschnitt 6 Stunden und 31 Minuten pro Woche (ohne Ferien, Krankheitstage oder die Stufenfahrt auszuschließen)), dennoch hat mir die Besondere Lernleistung viel Spaß gemacht. Ich habe einige neue Techniken erlernt und ich hoffe natürlich auf eine positive Auswirkung auf meine Abiturnote.

Ich würde den Nutzen dieses Projektes als ziemlich hoch beurteilen, da ich die Verwendung meiner Apps bereits am CFG miterlebt habe und nur positives Feedback von den nutzenden Lehrern erfahre. Im Bildungswesen besteht eine stetige Nachfrage nach interaktiven Anwendungen, welche die Phänomene der Natur den Schülern und Schülerinnen näherbringen. Es wird zwar versucht, die Digitalisierung auszubauen, jedoch existiert in diesem Bereich ein großer Nachholbedarf. Große Firmen konzentrieren sich überwiegend auf die Entwicklung von Anwendungen, von denen eine Vielzahl an Verbrauchern profitiert, um deren Profit zu steigern. Da steht das Bildungswesen leider nicht im Vordergrund. Deswegen schätze ich dieses Projekt als positiven Impuls zur Realisierung dieses Zieles ein. Die Anwendungen dieser Besonderen Lernleistung wurden durch den Unterricht am CFG inspiriert und teilweise sogar auf Wunsch vom CFG angefertigt (digitale Sonnenkuppel). Ich plane nach Beendigung dieser Besonderen Lernleistung, das Projekt **nicht** zu archivieren, sondern bei Bedarf oder Vorschlägen vom CFG fortzuführen und durch weitere hilfreiche und lehrreiche Anwendungen zu erweitern. Damit meine bisherige Leistung unverfälscht und abgetrennt von nachträglichen Veränderungen bewertet werden kann, werde ich einen separaten git-tag erstellen, welcher eine Momentaufnahme des Projektes repräsentiert.

Durch diese Besondere Lernleistung habe ich viel gelernt und zum Teil neue Fähigkeiten erlangt. Darunter fällt unter anderem die Projektplanung und die Verwaltung und Organisation über das Versionskontrollsystem git. In vorherigen Projekten habe ich breites git genutzt, da dieses Projekt jedoch so umfangreich ist, habe ich viele neue Funktionen entdeckt und verwendet. Eine weitere neue Technologie ist das sogenannte Kanban-Board. Dieses hilft bei der Organisation von anstehenden Änderungen. Meine Kenntnisse in dem Webframework Vue.js habe ich ebenfalls erweitert und gefestigt, außerdem habe ich viel im Bereich der Spiele Entwicklung mithilfe von Unity erlernt. Das Wissen, wie man eine

solche schriftliche Arbeit anfertigt und vorallem die formalen Aspekte werden mir für zukünftige Arbeiten (Bachelor etc.) sehr von Nutzen sein. Jede dieser neu erworbenen Fähigkeiten wird sich in meinem Berufsleben voraussichtlich als wertvoll erweisen.

Da mir die statistische Analyse von Daten Freude bereitet und ich durch die Protokollierung meiner Termine (die Zeiten, an denen ich an der Besonderen Lernleistung gearbeitet habe) einen umfangreichen Datensatz vorliegen habe, habe ich diese kleine Visualisierung für den Verlauf der Arbeitszeiten in Relation zu den Wochentagen geschaffen (siehe Abbildung 28). Um diesen Datensatz zu erstellen habe ich mir ein Python⁶⁶-Programm geschrieben, welches automatisch auf meinen Google Kalender zugreift (in dem die Ereignisse auf das nächstliegende 5min Intervall gerundet eingetragen sind), die Ereignisse filtert, diese Daten auswertet und anschließend in diesem Diagramm visualisiert. Zuerst habe ich nur das hellblaue Balkendiagramm erzeugt, um zu vergleichen, an welchen Tagen ich wie häufig an der Besonderen Lernleistung gearbeitet habe (mehrere Ereignisse pro Tag werden als ein einzelnes Ereignis gewertet (im Fall von Pausen)). Dann stellte ich jedoch fest, dass diese Informationen noch nicht viel aussagen, da die Dauer vollkommen ignoriert wird. Deswegen habe ich den roten Graphen hinzugefügt. Jedoch bestanden nun beide Graphen nur aus Durchschnittswerten, weshalb ich für eine bessere Aussage noch die gesamte Zeit pro Tag mit dem grünen Graphen angegeben habe. Damit man jeden Graphen gut erkennen kann, nutze ich insgesamt drei verschiedene y-Achsen.

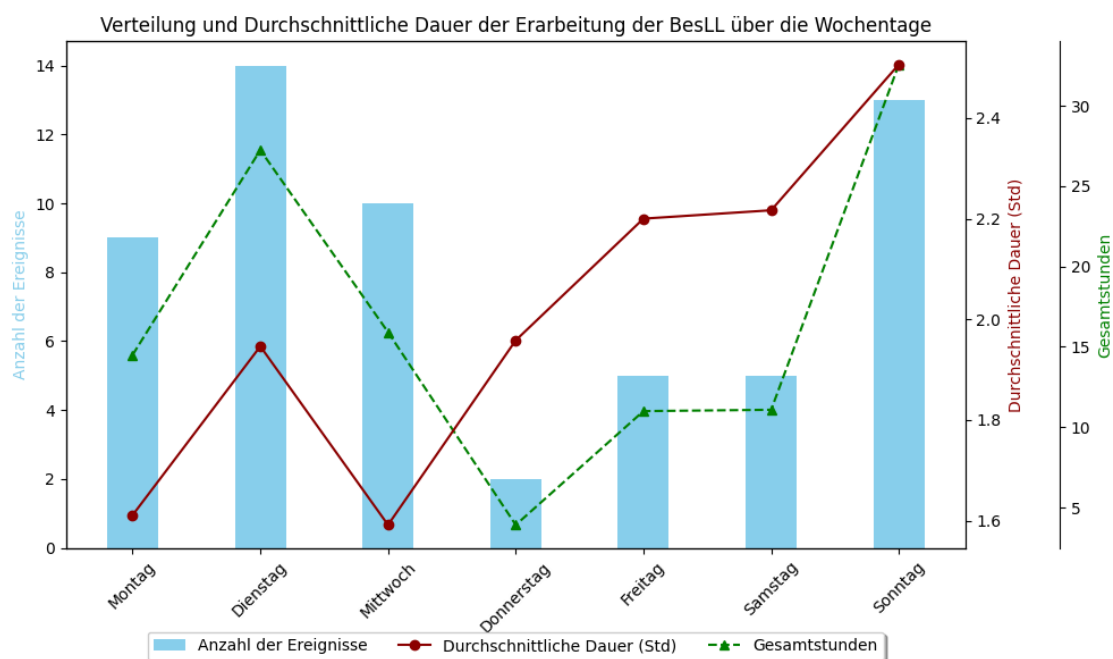


Abbildung 28 - Analyse der Ereignisse pro Wochentag (bis zum 15. März)

Ich finde es faszinierend, wie man anhand dieses Diagramms meinen Stundenplan nachvollziehen kann. Da ich am Donnerstag immer bis 18 Uhr ohne eine Freistunde durchgehend Schule habe, habe ich an diesem Tag kaum an der Besonderen Lernleistung gearbeitet. Am Dienstag habe ich zwar auch jeden Tag 10 Stunden, jedoch habe ich immer eine lange Freistunde in der Mitte des Tages, weshalb ich an diesem Tag auch gut arbeiten kann. Die restlichen Daten lassen sich ebenfalls auf ähnliche Gründe zurückführen, jedoch lasse ich diese aufgrund der Monotonie hier aus. Das Diagramm hat zwar nichts mit der eigentlichen Arbeit zu tun, jedoch wollte ich dieses hier trotzdem erwähnen.

⁶⁶ Python ist eine Programmiersprache. Diese wird gerne im Bereich Data-Science genutzt.

Anhang

Danksagung

Hiermit möchte ich mich sowohl beim CFG im Allgemeinen als auch bei ausgewählten Lehrern für die Unterstützung mit und die Ermöglichung der Besonderen Lernleistung bedanken.

Ich bin der Kooperation meiner Schule (CDG) mit dem CFG (Carl-Fuhlrott-Gymnasium) sehr dankbar, da diese Besondere Lernleistung andernfalls nicht stattgefunden hätte.

Ich bedanke mich bei meinem Betreuer und Astronomielehrer Bernd Koch. Dieser motivierte, unterstützte und half mir bei anstehenden Fragen. Der Schwerpunkt seinerseits bestand in der Weiterentwicklung der digitalen Sternenkarte. In Ergänzung möchte ich noch dem Physiklehrer Michael Winkhaus für die Bereitstellung der benötigten Formeln für die Entwicklung der digitalen Sonnenkuppel danken. Dieser hat mich ebenfalls in der Entwicklung der Besonderen Lernleistung unterstützt. Sein Fokus lag in der Entwicklung der Sonnenkuppel.

Außerdem bedanke ich mich bei GitHub für die Bereitstellung des „[GitHub Student Developer Pack](#)“, da ich dieses Projekt ohne die kostenlos enthaltenden Programme und Dienste nicht hätte umsetzen können.

Anmerkungen

1. Wenn in dieser Arbeit ein Text Kursiv gedruckt ist, dann handelt es sich immer um einen Querverweis. Wenn Sie die PDF-Version verwenden, dann können Sie auf diesen Querverweis drücken, um zum Ziel zu springen.
2. In manchen Abschnitten verwende ich häufig das Wort ‚wir‘. Damit keine Irritation entsteht, wer mit diesem ‚wir‘ gemeint ist, erkläre ich mich kurz. Da diese Abschnitte in der Form einer Anleitung geschrieben sind und ich dem Leser meine Schritte so erkläre, dass dieser diese nachvollziehen und übernehmen kann, verwende ich das Wort ‚wir‘ (um den Leser mit einzubeziehen).

Links

Ich habe aufgrund von Lesbarkeit, Nutzerfreundlichkeit und Platz bei längeren Links den Linkverkürzungsdienst „Bitly“ verwendet. Zur Sicherheit werde ich dennoch hier im Anhang die Zuordnung der verkürzten Links auflösen.

Bitly Link	Titel	Aufgelöster Link (Original)
https://bit.ly/3SqWUGO	Astronomie Projektarbeit.pdf	https://github.com/domx4q/astroProject/blob/master/Astronomie%20Projektarbeit.pdf
https://bit.ly/48GEw2n	router/index.js	https://github.com/domx4q/astroProject/blob/master/src/router/index.js
https://bit.ly/3UcVVLG	src/App.vue	https://github.com/domx4q/astroProject/blob/master/src/App.vue
https://bit.ly/3vW1e8j	package.json	https://github.com/domx4q/astroProject/blob/master/package.json
https://bit.ly/3HMYPQO	Post-Plaster-Drehbare-Sternkarte-fuers-CFG.pdf	https://www.schuelerlabor-astromie.de/wp-content/uploads/2022/06/Post-Plaster-Drehbare-Sternkarte-fuers-CFG.pdf

https://bit.ly/3UAoeni	Math.atan2() - JavaScript MDN	https://developer.mozilla.org/en-US/docs/web/javascript/reference/global_objects/math/atan2
https://bit.ly/3HWHwMq	src/views/extra/StarsView.vue Selected Lines 524 to 584	https://github.com/domx4q/astroProject/blob/master/src/views/extra/StarsView.vue#L524-L584
https://bit.ly/49ISMgW	EventTarget: addEventListener() method - Web APIs MDN	https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener
https://bit.ly/3T1Ucro	src/components/collapsibleContainer.vue	https://github.com/domx4q/astroProject/blob/master/src/components/collapsibleContainer.vue
https://bit.ly/3T0JP7q	src/mixins/defaults.js	https://github.com/domx4q/astroProject/blob/master/src/mixins/defaults.js
https://bit.ly/3OK8YAE	src/components/details.vue	https://github.com/domx4q/astroProject/blob/master/src/components/details.vue
https://bit.ly/3IhlmEx	Difference master to 385-simulate-from-timestamp	https://github.com/domx4q/astroProject/compare/master...385-simulate-from-timestamp
https://bit.ly/3uWxqZ4	src/views/extra/SonnenkuppelView.vue	https://github.com/domx4q/astroProject/blob/master/src/views/extra/SonnenkuppelView.vue
https://bit.ly/3uW5vbJ	src/FullscreenUnityRenderer.vue	https://github.com/domx4q/astroProject/blob/master/src/fullscreenUnityRenderer.vue
https://bit.ly/4cacRZD	src/components/UnityRenderer.vue	https://github.com/domx4q/astroProject/blob/master/src/components/unityRenderer.vue
https://bit.ly/4c8MsLY	GitHub Workflows	https://github.com/domx4q/astroProject/tree/master/.github/workflows
https://bit.ly/3TzNaKO	Dockerfile	https://github.com/domx4q/astroProject/blob/master/Dockerfile
https://bit.ly/48RgWil	dockerEntrypoint.sh	https://github.com/domx4q/astroProject/blob/master/dockerEntrypoint.sh
https://bit.ly/3v3dfc1	updateDaemon.sh	https://github.com/domx4q/astroProject/blob/master/updateDaemon.sh
https://bit.ly/43o61Mb	updateServer.sh	https://github.com/domx4q/astroProject/blob/master/updateServer.sh

https://bit.ly/3IEmpvZ	startServer.sh	https://github.com/domx4q/astroProject/blob/master/startServer.sh
https://bit.ly/3x5rdKL	docker-compose.yml	https://github.com/domx4q/astroProject/blob/master/docker-compose.yml
https://bit.ly/3PmDbpM	src/background.js	https://github.com/domx4q/astroProject/blob/master/src/background.js
https://bit.ly/3J5I2JD	Download - Sonnenkuppel.zip	https://cloudster.online/download/astro/Sonnenkuppel.zip

Abbildungsverzeichnis

Abbildung 1 - Electron basierende Anwendungen	6
Abbildung 2 - Inhalt: docker-compose.yml	8
Abbildung 3 - Reverse-Proxy Modell	10
Abbildung 4 - HTML-Box-Modell	10
Abbildung 4 - Sternenkarte (Standardansicht)	11
Abbildung 5 - HTML-Box-Modell	12
Abbildung 6 - Zeit- und Datumskonvertierung Code	13
Abbildung 7 - Atan2 Funktionsweise	14
Abbildung 8 - Deklinationszeiger	17
Abbildung 9 - Beispiel: Anti-Aliasing	17
Abbildung 10 - Sternenkarte + alle Extras	18
Abbildung 11 - Sternenkarte: Marker	19
Abbildung 12 - Marker CSS Code	19
Abbildung 13 - platzierbare Planeten	20
Abbildung 14 - adaptedSize Code	22
Abbildung 15 - Planeten Skalierungscode	24
Abbildung 16 - Schattenwerfer (Sonnenkuppel)	26
Abbildung 17 - physisches Modell der Sonnenkuppel	26
Abbildung 18 - Beispiel Google Sonnenverlauf	27
Abbildung 19 - Sonnenkuppel (Standardansicht)	28
Abbildung 20 - Kamerarotation	29
Abbildung 21 - Kameratransformation	29
Abbildung 22 - Obelisk (Halde Hohward)	31
Abbildung 23 - Höhendifferenz in der Sonnenkuppel	34
Abbildung 24 - GitHub workflows	35
Abbildung 25 - git-log (01.03 - 07.03)	36
Abbildung 26 - LoC Badge Beispiel	37
Abbildung 27 - docker-compose.yml	40
Abbildung 28 - Analyse der Ereignisse pro Wochentag (bis zum 15. März)	42

Abbildungsquellen

1. <https://www.electronjs.org/de/>
2. *Selbst erstellt* (Screenshot von eigenem Code (Syntax-Highlighting: portainer.io))
3. <https://www.cloudflare.com/de-de/learning/cdn/glossary/reverse-proxy/>
4. *Selbst erstellt* (Screenshot der App)
5. *Selbst erstellt* (Screenshot eines Beispiels)
6. *Selbst erstellt* (Screenshot eines Ausschnittes des Quellcodes)
7. https://developer.mozilla.org/en-US/docs/web/javascript/reference/global_objects/math/atan2#description
8. *Selbst erstellt* (Screenshot in Photoshop)
9. <https://paulblunt.de/blog/Anti-Aliasing>
10. *Selbst erstellt* (Screenshot der App)
11. *Selbst erstellt* (Screenshot eines Beispiels)
12. *Selbst erstellt* (Screenshot eines Ausschnittes des Quellcodes)
13. *Selbst erstellt* (Screenshot der App)
14. *Selbst erstellt* (Screenshot eines Ausschnittes des Quellcodes)
15. *Selbst erstellt* (Screenshot eines Ausschnittes des Quellcodes)
16. *Selbst aufgenommen* (Foto des Holzstückes der Sonnenkuppel)
17. *Selbst aufgenommen* (Foto der Sonnenkuppel (mit aktiver Beleuchtung))
18. *Selbst erstellt* (Screenshot der Google Wetter App)
19. *Selbst erstellt* (Screenshot der App)
20. *Selbst erstellt* (Screenshot eines Ausschnittes des Quellcodes)
21. *Selbst erstellt* (Screenshot eines Ausschnittes des Quellcodes)
22. <https://www.flickr.com/photos/jorbasa/26370215665>
(Lizenz: „CC BY-ND 2.0 Deed“ (Änderungen: Zugeschnitten))
23. *Selbst erstellt* (Screenshot der App)
24. *Selbst erstellt* (Screenshot aus Github)
25. *Selbst erstellt* (Screenshot des git-logs)
26. *Selbst erstellt* (Screenshot aus Github)
27. *Selbst erstellt* (Screenshot eines Ausschnittes des Quellcodes)
28. *Selbst erstellt* (Diagramm mit Matplotlib (Python))

Quellen

1. <https://vuejs.org/guide/extras/ways-of-using-vue>
2. <https://docs.docker.com/compose/compose-file/>

Anlage: „Erklärung“:

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Außerdem erkläre ich mich einverstanden, dass diese Besondere Lernleistung sowohl digital als auch analog publiziert wird.

Wuppertal, 09.04.2024

Ort, Datum

Dominik Fuchs

Unterschrift