Realisierung des Spieleklassikers "Archon"

mit 3D- und Webtechnologien

Bachelorarbeit

im Fachgebiet Software-Engineering

zur Erlangung des akademischen Grades

Bachelor in Engineering

Vorgelegt von:	Kevin Dyes
Matrikelnummer:	2694420
Hochschule:	Technische Hochschule Georg-Simon-Ohm
Studienbereich:	Elektro- und Informationstechnik
Erstgutachter:	Prof. Dr. Röttger
Zweitgutachter:	Prof. Dr. Hopf

Kurzfassung

Die folgende Arbeit beschäftigt sich mit aktuellen Technologien in den Bereichen Web-, 3D- und Spieleentwicklung. Als Anwendung dieser Themenbereiche wurde ein klassisches Spiel aus den 1980er Jahren analysiert um die Umsetzbarkeit im Umfeld des Web zu prüfen. Mit den Ergebnissen wurde anschließend eine Realisierung des Spiels durchgeführt.

Inhaltsverzeichnis

Vorwort

Blabla

Am Ende führen

Erklärung

Ich, Kevin Dyes / Matrikel Nummer 2694420, versichere hiermit, dass ich diese Bachelorarbeit mit dem Thema

Realisierung des Spieleklassikers "Archon"
mit 3D- und Webtechnologien

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Nürnberg, den 31. Juli 2018

Kevin Dyes

1. Einleitung

1.1. Hinführung

Im Jahr 2017 alleine wurden mehr als 1,8 Milliarden browser-fähige Endgeräte weltweit verkauft.[6] Der technologische Fortschritt in Hardware und Software geben dieser großen Masse an Zugängen zum Internet eine riesige Vielfalt an unterschiedlichsten Applikationen, die nahezu überall auf der Welt benutzbar sind. Einen großen Anteil machen dabei Spiele aus. Dabei haben auch Spiele auf mobilen Plattformen, wie Smartphones, immer mehr Grafikdetails und Leistungsanforderungen. Aber auch der Trend der Spieleklassiker ist immer noch belebt und beliebt: Klassische Spieleplattformen, wie der Commodore 64 und die Atari-Spielekonsole bekommen Neuauflagen[1] und auch klassische Spiele werden immer wieder für diverse Plattformen neu aufgelegt. Sie gehen dem Drang nach mehr Grafikdetails und Leistungshunger aus dem Weg gehen und sollen durch Charme mit Pixeln und Nostalgie punkten.
Spiele wurden für die Vorreiter des Konsolen- und Heimcomputermarktes, den Commodore 64 und die Atari, in Massen entwickelt.[2] Die beliebtesten Spiele waren dabei unter anderem die Folgenden:[3]

- Pirates
- Maniac Mansion
- Turrican II
- Archon
- The Great Giana Sisters

Mit dabei ist auch das Spiel, um das sich diese Arbeit dreht: *Archon*.

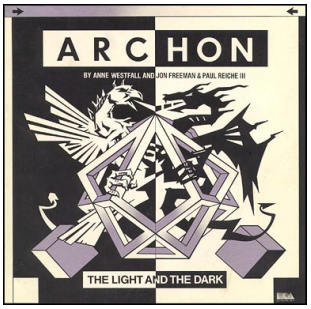

Abb. 1.1.: Archon-Cover[4]

[1] Auflistung von Neuauflagen bekannter Konsolen: http://www.computerbild.de/fotos/cbs-News-PC-Bilder-Die-coolsten-Kultkonsolen-Remakes-18443703.html
[2] Es wurden etwa 17.000 kommerzielle Titel entwickelt.[17]
[3] Auszug aus Liste nach https://www.c64-wiki.de/wiki/C64-Wiki:TOP-Liste_des_C64-Wiki
[4] Quelle: C64-Wiki, https://www.c64-wiki.com/wiki/Archon

Abb. 1.2.: Archon – Das Spielfeld[5]

Archon ist ein Spiel, dass den ewigen Kampf von Licht und Schatten austrägt. Verschiedene Fabelwesen stehen sich dabei auf einem Schachbrett gegenüber. Abbildung 1.2 zeigt die Aufstellung am Anfang des Spiels. Auch was den Spielverlauf angeht kann Archon sehr gut mit Schach verglichen werden: Es werden abwechselnd Figuren gezogen mit dem Ziel den Gegner zu besiegen. Anders als beim Schach werden die Fabelwesen jedoch nicht durch ein Aufeinandertreffen direkt geschlagen, sondern tragen ihre Kämpfe auf dem Kampffeld aus, das in Abbildung 1.3 zu sehen ist. Der Sieger des Kampfes bleibt im Spiel, der Verlierer wird aus dem Spiel entfernt.

Abb. 1.3.: Archon – Das Kampffeld[6]

Archon genoss für Atari und Commodore 64 großen Erfolg, ähnlich vieler anderer Spiele für die Konsolen der 80er. Auch hat Archon schon einige Neuauflagen und Nachfolger bekommen.[7] Die meisten davon wurden für den Computer entwickelt. Der Computer ist jedoch nicht die einzige Plattform für Spiele. Plattformen für Spiele gibt es genügend für jeden Geschmack: Ob Konsole – die ggf. tragbar ist –, Computer, Smartphone oder Webbrowser. Die klaren Vorteile des Web als Plattform, auch für Spiele, sind dabei die Offenheit, der einfache Zugang und die immer weiter steigende Unterstützung neuer, standardisierter Technologien.

[5] Quelle: C64-Wiki, https://www.c64-wiki.com/wiki/Archon
[6] Quelle: C64-Wiki, https://www.c64-wiki.com/wiki/Archon
[7] Eine Neuauflage ist Archon Classic: https://store.steampowered.com/app/65400/Archon_Classic/

So sind heutzutage hardwarebeschleunigte[8] 3D-Visualisierung und Echtzeitkommunikation im Webbrowser Stand der Technik. In dieser Arbeit soll daher die Verschmelzung von State of the Art Webtechnologien mit einem Spieleklassiker erfolgen.

1.2. Aktueller Forschungsstand

Dieses Kapitel behandelt den Einstieg in aktuelle Themen der Spieleentwicklung und des Internets. Zunächst wird dabei auf aktuell stark behandelte Themen des World Wide Web Consortium (W3C) – dem Konsortium zur Standardisierung des Web–, eingegangen. Dabei wird das Thema Grafik im Web noch einmal gesondert eingeführt. Abschließend wird auf aktuelle Probleme, Herausforderungen und Eigenschaften der heutigen Spieleindustrie eingegangen.

aktuelle Themen des W3C

Das W3C hat im Moment neun Kernthemen im Rahmen "Web Design and Applications".
Mit MathML soll ein Standard für die Anzeige von mathematischen Ausdrücken geschaffen werden. Weiterhin wird in den Themen "Privacy", "Accessibility", "Internationalization" und "Mobile Web" stark geforscht und an Standards gearbeitet um möglichst jedem Menschen der Welt, mit jedem internetfähigen Gerät, den vollen und sicheren Zugang zum Internet zu ermöglichen. Die in dieser Auflistung noch fehlenden Themen sind "Audio & Video", "JavaScript Web APIs"[9] sowie "Graphics".[10] Das Thema "Audio & Video" behandelt vor allem die Standardisierung von Formaten von Audio- & Videoinhalten. "JavaScript Web APIs" dienen der Entwicklung von umfangreicheren clientseitigen Web-Anwendungen.

Das Thema "Graphics" behandelt nicht nur Formate zu Darstellung, sondern auch Schnittstellen zur dynamischen Erzeugung und Veränderung, von Grafiken. Die Schnittstellen "Scalable Vector Graphics" (SVG) und die "Canvas API" sind in diesem Zusammenhand zu nennen. Sie werden beide inzwischen von den meisten, modernen Browsern unterstützt. Beide Schnittstellen erlauben die dynamische Erzeugung und Manipulation von zweidimensionalen (2D) Grafiken. Für dreidimensionale (3D) Grafiken ist allerdings noch kein Standard des W3C ausgearbeitet worden. Es gibt allerdings den "WebGL" Standard, welcher von der "Khronos Gruppe" ausgearbeitet wurde.[11] Der Standard erlaubt es der "Canvas API" einen neuen Kontext zu geben und damit 3D-Grafiken zu erzeugen. Die meisten, modernen Browserhersteller sind Teil der Arbeitsgruppe zur Erstellung des Standards.

[8] Hardwarebeschleunigung entlastet den Prozessor durch Auslagerung an spezialisierte Hardware für bestimmte Aufgaben.
[9] Application Programming Interface (API) – Eine Schnittstelle zum programmatischen Zugriff auf eine Anwendung
[10] Die Themen sind dem W3C direkt entnommen: https://www.w3.org/standards/webdesign/
[11] Für weitere Informationen zur Khronos Gruppe: https://www.khronos.org/webgl/

aktuelle Themen der Spieleindustrie

Spielen sind heutzutage kaum noch Grenzen gesetzt. Mit "Virtual Reality" (VR), teilweise Millionen von Spielern – auf der ganzen Welt verteilt–, oder auch höchsten Anforderungen an Rechen- und Kommunikationsleistung bringen Entwickler auch aktuelle Hardware teils an ihre Grenzen. Dabei entstehen, dank der teils riesigen Budgets[12], unglaubliche, digitale Welten.
Weiterhin wird immer größerer Fortschritt im Bereich der künstlichen Intelligenz gemacht, die auch für die Spieleindustrie eine große Rolle spielt.[13]

1.3. Motivation / Problemstellung

Bisher sind oftmals nur Demonstrationen, kurze Starter-Kits bzw. Einstiegspunkte einzelner Web-Technologien und Frameworks im Web zu finden. Anhand dieser Arbeit sollen die Möglichkeiten des Webs in ihrer Gesamtheit dargestellt werden und die Realisierbarkeit von 3D-Spielen für Webbrowser aufgezeigt werden. Weiterhin kann das Spiel als Beispiel für typische Problemstellungen eines Entwicklers für Browserspiele dienen.
Versionen, Ableger und Kopien des Spieleklassikers Archon gibt es zahlreich. Meist sind diese jedoch auf ein System beschränkt, oder tatsächlich nur mit einem Emulator[14] ausführbar. Das Web leistet an dieser Stelle die perfekte Abhilfe: Es gibt unzählige Geräte, die heutzutage einen Webbrowser installiert haben, somit kann jeder Zugriff auf diesen Ableger haben, der ein webfähiges Gerät besitzt.

[12] Siehe https://en.wikipedia.org/wiki/List_of_most_expensive_video_games_to_develop
[13] Siehe dazu: https://openai.com/
[14] Emulator – Ein Emulator simuliert ein anderes System (z. B. Spielekonsolen werden auf einem Computer emuliert)

1.4. Zentrale Begriffe

1.4.1. Was versteht man unter Webtechnologien?

Webtechnologien betiteln meist die Sammlung aller technischen Aspekte zum Erstellen, Warten und Entwickeln einer Webanwendung.
Webanwendungen selbst bestehen immer aus einem Client-Server-Modell. Dabei stellt, auf Anfrage eines Clients, der Server passende Inhalte für den Client bereit. Der Client wird dabei meist von einem Webbrowser repräsentiert, der die Anfragen an Webserver sendet und Antworten empfängt. Der Webbrowser ist dabei für die anschließende korrekte Interpretierung der Antworten zuständig. Typische Bestandteile des Clients und ihre Bedeutungen sind:

- Hypertext Markup Language (HTML) zur Beschreibung von Inhalt und seiner Struktur
- Cascading Style Sheets (CSS) zur Beschreibung des Aussehens des Inhalts
- JavaScript zur Dynamisierung des Clients und der gesendeten Inhalte

Ein Webserver ist eine Anwendung, die auf Anfragen von Webprotokollen, wie dem Hypertext Transer Protocol (HTTP), mit Inhalten und Daten antworten kann. Ein Webserver kann dabei noch weitere Eigenschaften haben, wie Fehlerverarbeitung, Verschlüsselung der Kommunikation oder Verwaltung einer Datenbank für Anwendungsdaten.

1.4.2. Was versteht man unter 3D-Technologien?

Da ein Bildschirm, beispielweise eines Computers, nur zweidimensional ist, muss durch andere Methodiken der Effekt einer dritten Dimension geschaffen werden. Wie dieser Prozess im Allgemein funktioniert wird folgend erläutert.
Objekte im 3D-Raum werden über ihre Eckpunkte aufgespannt und im Code somit in einem 3D-Raum mit normalem 3-Achsen-Koordinatensystem dargestellt. Anschließend folgt der Prozess des Renderns, bei dem zunächst aus den einzelnen Punkten der Objekte die Formen errechnet werden, indem die Eckpunkte zu Flächen verbunden werden. Anschließend erfolgt eine Ausrichtung aller Flächenpunkte am Pixel-Raster des Bildschirms, sodass eine 3D-Projektion der 2D Pixel entsteht. Der nächste Schritt, Fragment-Bearbeitung, behandelt die Einfärbung der, im vorherigen Schritt gebildeten, "Fragmente" anhand von Licht und verwendeten Texturen. Der finale Schritt des Renderns wandelt die 3D-Projektion in ein 2D-Pixel-Bild, dass dann auf dem Bildschirm angezeigt wird. Außerdem werden im letzten Schritt Prüfungen für die Sichtbarkeit von Objekten unternommen, sodass nicht sichtbare Objekte, oder Teile von ihnen, auch nicht weiter verarbeitet werden.

1.5. Ziel dieser Arbeit

Der Schwerpunkt dieser Arbeit befasst sich mit der Umsetzung des bekannten Spieleklassikers Archon mit Web- und 3D-Technologien. Ziel dieser Arbeit ist es eine Webapplikation zu erstellen, die den Spieleklassiker Archon in einem 3D-Stil widerspiegelt. Die Webapplikation soll dabei folgende generelle Anforderungen erfüllen, die im Laufe dieser Arbeit näher spezifiziert werden:

- Die Applikation muss mittels Webtechnologien implementiert werden
- Das Spiel muss in einem 3D-Stil angezeigt werden
- Das Spiel muss als solches wiedererkennbar sein

1.6. Aufbau

Im ersten Teil dieser Arbeit sollen die Anforderungen an eine Neuauflage von "Archon" erfasst werden und eine Beschreibung der Features und Spielmechaniken zur Umsetzung erstellt werden. Darauffolgend werden die Anforderungen der technologischen Seite herausgearbeitet. Dabei soll untersucht werden, ob die technologischen Anforderungen Einschränkungen für eine Realisierbarkeit bilden.
Nachdem ggf. vorhandene Einschränkungen und die Realisierbarkeit der Spielmechaniken herausgearbeitet wurden, wird die Implementierung beschrieben. Hier werden zunächst zu benutzende Software-Bibliotheken für die jeweiligen Technologien und deren Nutzen und Vorteile beschrieben. Anschließend werden der Aufbau der Entwicklungsumgebung, sowie benutzte Hilfsmittel und getroffene Vereinfachungen beschrieben. Nachdem alle Rahmenbedingungen für eine Realisierung und Realisierbarkeit geklärt wurden, wird eine Architektur für das Spiel entworfen. Sie dient als Beschreibung zum Zusammenhang der einzelnen Komponenten und als Anleitung zur Umsetzung. Die Umsetzung der Architektur, die Implementierung, wird dann in Ausschnitten gezeigt. Diese Ausschnitte sollen die einzelnen Schritte und die Vorgehensweise bei der Umsetzung von Architektur und Programmierung aufzeigen.
Ein kurzer Einblick auf den abschließenden Stand und eine Reflexion des Ergebnisses, mit anschließendem Ausblick auf Folgearbeiten an der Webapplikation, bildet den Abschluss dieser Arbeit.

2. Hauptteil

2.1. Einleitung

Die Anforderungen an eine Neuauflage von "Archon" werden durch eine vorhergehende Analyse des Spiels und seiner einzelnen Komponenten und Mechaniken herausgearbeitet.
Anschließend werden die Möglichkeiten und Optionen der zu benutzenden Technologien analysiert und eine Verbindung zu den Spielmechaniken hergestellt.

2.2. Analyse

2.2.1. Analyse des klassischen Spiels

In diesem Abschnitt werden die Anforderungen und Regeln aus Archon herausgearbeitet. Die Anforderungen und Regeln werden, der Übersicht halber, in folgende Kategorien unterteilt:

- Generelle Regeln und Ziele des Spiels
- Das Spielbrett
- Das Kampfareal
- Die Figuren
- Die Zauber

Die folgenden Abschnitte enthalten eine kurze Zusammenfassung des Spiels und seinen Inhalten und Regeln anhand der obigen Aufteilung. Anschließend wird herausgearbeitet, welche Eigenschaften eine Realisierung des Spiels inne haben muss, um das Spiel ausreichend widerzuspiegeln.

Generelle Regeln und Ziele des Spiels

Archon ist ein Spiel, dass auf einem 9 x 9 Schachbrett stattfindet. Ähnlich wie beim Schach gibt es zwei Parteien, Licht und Dunkelheit, die sich im Wettstreit gegenüberstehen. Die Spieler ziehen dabei immer abwechselnd ihre Figuren auf dem Spielbrett, wobei kein Zug gepasst werden kann. Jeder Spieler beginnt mit 18 Figuren, die in acht verschiedene Typen gegliedert sind. Die Figuren von Licht und Dunkelheit sind vollständig unterschiedlich. Treffen zwei Figuren auf dem Spielbrett zusammen, stehen sie sich in einem Kampfareal gegenüber. Jede Figur hat dabei ihre eigenen Lebenspunkte und ihre eigene Angriffsstärke. Der Sieger dieses Kampfes bleibt auf dem Spielbrett, während der Verlierer aus dem Spiel genommen wird. Sollte der Kampf in einem Unentschieden ausgehen, werden beide Figuren aus dem Spiel genommen.
Das generelle Ziel des Spiels ist es alle 5 speziellen Machtfelder gleichzeitig mit eigenen Figuren

zu besetzen. Desweiteren kann das Spiel gewonnen werden, indem alle gegenerischen Figuren besiegt werden oder die letzte Figur mit einem Gefängnis-Zauber belegt wird. Ein Unentschieden tritt auf wenn das Spiel zu lange passiv ist, also wenn für mindestens zwölf Züge kein Kampf stattfindet und kein Zauber gewirkt wird.
Das Spiel kann in folgenden Modi gestartet werden:

- Spieler gegen Spieler
- Spieler gegen Computer
- Computer gegen Computer (Demo-Modus)

Wobei der erste Spieler immer die Wahl der startenden Farbe, sowie der eigenen Farbe hat. Weitere Einstellmöglichkeiten gibt es nicht.

Das Spielbrett

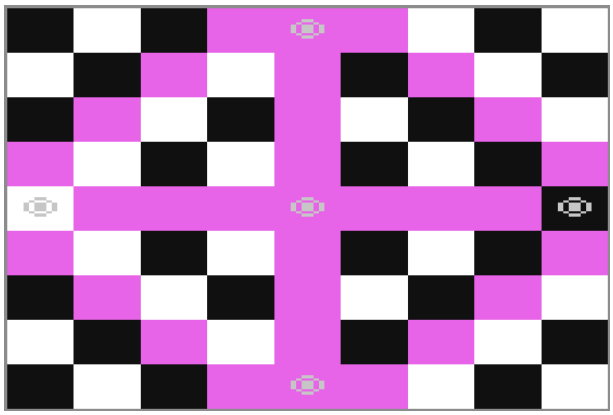

Abb. 2.1.: Das Spielbrett [15]

Das Spielbrett besteht aus 9x9 Feldern von drei Typen:

- Permanent weiße Felder
- Permanent schwarze Felder
- Felder, die zwischen Schwarz, vier anderen Farben und Weiß wechseln

Alle farbwechselnden Felder ändern ihre Farbe nach dem Zug des zweiten Spielers, sodass jeder Spieler einen Zug der gleichen Farbfelder hat. Ein Farbzyklus dauert damit zwölf Züge pro Seite: Sechs von Weiß zu Schwarz und Sechs zurück. Da sich die mittleren vier Farben je nach Ableger unterscheiden können, werden sie fortan mit Zahlen von 1 bis 6 durchnummeriert, wobei 1 Weiß darstellt und 6 Schwarz. Wenn die Licht-Seite das Spiel beginnt, startet der Farbzyklus bei Farbe 4 und wird dunkler. Beginnt die Dunkelheit-Seite, so startet der Zyklus bei Farbe 3 und wird heller. Die Farbe der Felder gibt den darauf stehenden Figuren einen Lebenspunkte-Bonus, der größer ausfällt, je näher die Feld-Farbe an der Team-Farbe der Figur ist. [17] Weiterhin gibt es die

[15] Quelle: C64-Wiki, https://www.c64-wiki.com/wiki/File:ArchonStrategiieBildschirm.gif

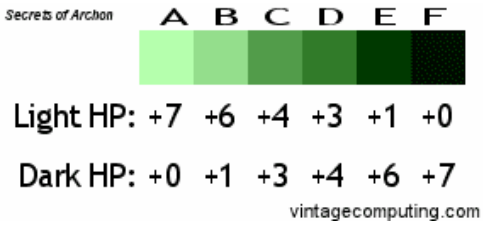

Abb. 2.2.: Lebenspunkte-Bonus anhand der Feldfarben[16]

fünf Machtfelder, deren Einnahme die Siegbedingung darstellt. Diese Felder haben außerdem zwei spezielle Effekte auf Figuren, die darauf stehen. Zum einen können darauf stehende Figuren – und auch die Felder an sich, nicht Ziel eines Zaubers werden. Zum Anderen werden die Figuren nach jedem eigenen Zug um einen gewissen Betrag geheilt, sollten sie im Kampf verletzt worden sein. Diese fünf Machtfelder verteilen sich so, dass jeweils eines auf einem permanent schwarzen bzw. weißen Feld ist - dort wo der Zauberer bzw. die Zauberin ihre Ausgangsposition haben. Die anderen drei Felder sind auf farbwechselnden Feldern in der Mitte des Spielbretts verteilt.

Das Kampfareal

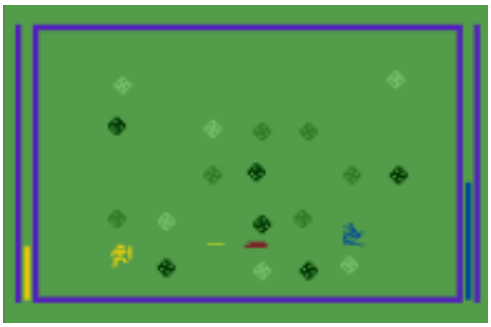

Abb. 2.3.: Das Kampfareal

Das Kampfareal erscheint wenn zwei Figuren auf dem Spielbrett aufeinander treffen. Es stellt eine große Fläche dar, bei der von Zeit zu Zeit pflanzenähnliche Hindernisse erscheinen und wieder verschwinden. An den Seiten des Kampfareals werden die Lebenspunkte der kämpfenden Figuren in Form von Balken dargestellt.
Die Figuren können sich hier frei bewegen und haben, je nach Typ, unterschiedliche Eigenschaften, Attacken und Geschwindigkeiten.

Die Figuren

Das Spielbrett ist zu Spielbeginn mit 18 Spielfiguren pro Seite belegt. Es gibt 16 verschiedene Typen von Figuren, wobei die Typen von entsprechenden Figuren von Licht und Dunkelheit unterschiedlich sind. Die Beschreibung der Typen wird hier oberflächlich in Form einer Auflistung gestaltet, da die exakten Zahlenwerte der Lebenspunkte, Geschwindigkeiten und andere

[17] Quelle: vintagecomputing, http://www.vintagecomputing.com/wp-content/images/archon/archon_colorchart.gif

Eigenschaften bereits in einer anderen Zusammenfassung zu finden sind und an diesem Punkt für einen generellen Einblick in das Spiel nicht relevant sind.[18]

Die Figuren des Lichts sind:

- Ritter
- Bogenschütze
- Walküre
- Golem
- Einhorn
- Phönix
- Dschinn
- Zauberer

Die Figuren der Dunkelheit sind:

- Goblin
- Mantikor
- Banshee
- Troll
- Basilisk
- Gestaltwandler
- Drache
- Zauberin

Sich gegenüber stehende Figuren-Klassen in dieser Auflistung entsprechen sich in etwa in ihrer Stärke. Die Figuren haben dabei auch auf dem Spielbrett unterschiedliche Eigenschaften, so können Boden-Figuren, wie der Troll oder der Ritter, nicht über andere Figuren hinweg gesetzt werden und nicht diagonal bewegt werden. Je nach Typ ist die Bewegungsreichweite unterschiedlich: Drache und Dschinn können beispielsweise bis zu 4 Felder bewegt werden, während die Walküre und die Banshee nur 3 Felder bewegt werden können.
Außerdem kann mit Hilfe eines Zaubers einer von vier Elementaren einmal pro Seite und Spiel beschworen werden. Sollte ein Elementar beschworen werden wird er aus der Liste der möglichen Elementare entfernt, sodass niemals in einem Spiel zwei gleiche Elementare beschworen werden können. Die vier Elementare sind nach den vier Elementen Feuer, Wasser, Erde und Luft eingeteilt und besitzen, ähnlich zu normalen Figuren, unterschiedliche Sprites[19] und Werte. Welcher Elementar beschworen wird ist zufällig. Weiterhin bekommen Elementare keinen Lebenspunkte-Bonus von Feldfarben, sie haben also immer die gleichen Lebenspunkte.

Die Zauber

Der Zauberer und die Zauberin können, jeweils, einmal pro Spiel die folgenden sieben Zauber wirken:

- **Teleportieren**
  Teleportiert eine eigene Figur auf ein Feld der eigenen Wahl. Sollte das Feld belegt sein, startet direkt ein Kampf.

[18] Für mehr Details siehe: The secrets of Archon http://www.vintagecomputing.com/index.php/archives/44

[19] Sprite - graphisches Objekt, dass vor dem Hintergrund dargestellt wird, oft auch als Shape bezeichnet

- **Heilen**
  Heilt eine eigene, verwundete Figur vollständig.

- **Wiederbeleben**
  Belebt eine eigene, tote Figur wieder. Die Figur muss auf eines der umliegenden Felder des Zauberers bzw. der Zauberin platziert werden, egal wo sich die Figur befindet.

- **Tauschen**
  Tauscht die Position zweier Figuren – unabhängig ihrer Farbe.

- **Zeitumkehr**
  Dreht den Farbzyklus um. Sollten die Felder jedoch Schwarz bzw. Weiß sein wenn der Zauber gewirkt wird, ändern sie ihre Farbe zur jeweils gegenteiligen Farbe beim nächsten Farbwechsel.

- **Bewegungsunfähigkeit**
  Eine Einheit wird unfähig gemacht Bewegungen auszuführen und Zauber zu wirken. Die Einheit bleibt bewegungsunfähig bis die Farb-Felder die eigene Team-Farbe erreicht haben. Die Einheit kann sich aber weiterhin normal im Kampfareal bewegen und angreifen.

- **Elementar beschwören**
  Beschwört einen Elementar auf ein auf ein besetztes Feld, sodass direkt ein Kampf stattfindet.

Das wirken eines Zaubers ist nicht auf Machtfelder möglich. Außerdem schwächt das Wirken jedes Zaubers den Lebenspunkte-Bonus auf dem Heimat-Feld des Zauberers bzw. der Zauberin um einen Punkt, sodass nach dem Wirken aller Zauber in einem Spiel der Bonus auf null Punkte geht.

Zusammenfassung für die Realisierung

Alle vorher genannten Aspekte des Spiels lassen sich in den Kern, der das Spiel ausmacht, zusammenfassen: Ein Strategiespiel auf einem 9x9 Schachbrett mit Action-Elementen in einem Kampfareal und verschieden starken Spielfiguren. Im folgenden werden die Aspekte herausgearbeitet, die eine große Priorität für eine Neuauflage inne haben. Dabei wird, ähnlich der Analyse, Komponentenweise vorgegangen.
Um eine ausreichende Schnittmenge mit dem Original zu haben und die neue Realisierung als "Ableger von Archon" bezeichnen zu können, sollten alle Aspekte unter "generelle Regeln und Ziele des Spiels" auf jeden Fall implementiert werden. Eine Ausnahme bildet hier die Auswahl der Spielmodi: Das Spiel an sich bietet zwar drei Modi an, jedoch ist es für eine erste Realisierung nur wichtig einen der Modi zu implementieren.
Das Spielbrett muss in seiner kompletten Funktionalität implementiert werden, da ganze Mechaniken sonst wegfallen und andere Mechaniken nutzlos werden. So zum Beispiel der Lebenspunkte-Bonus durch den Farbwechsel oder die Machtfelder als Einnahmepunkte mit speziellen Funktionen.

Das Kampfareal kann in einer ersten Version einfach implementiert sein. Wichtig ist hier nur das Vorhandensein an sich, sowie eine entsprechende Gewichtung der einzelnen Figuren nach ihrer Kampfkraft im Original.
Bei den Figuren sollte zumindest ein Typus implementiert sein. Es müssen nicht alle Figuren originalgetreu nachgebildet werden, sondern es sollte eine gewichtete Auswahl von Figuren geben, sodass ein Spiel zustandekommen kann. Das Aussehen kann dabei nach belieben angepasst werden um so der Realisierung ein eigenes Design zu geben. Wichtig ist hier, wieder einmal, nur die richtige Gewichtung von Kräften. Die Zauber machen beim Original einen großen Teil der strategischen Mechanik aus, sodass sie vollständig zu implementieren sind. Die Zauber an kritischen Zeitpunkten im Spiel zu wirken macht jede einzelne Partie Archon besonders. Der Stil der Zauber ist auch hier nicht wichtig, sondern ausschließlich ihre Funktion.

2.2.2. Anforderungsanalyse an neue Realisierung

Der Titel, sowie die Einleitung haben schon erläutert, dass bei dieser Realisierung zwei essentielle Technologien zum Einsatz kommen sollen:

1. 3D-Technologien
2. Web-Technologien

Dieser Abschnitt soll nun darstellen welche Aspekte der einzelnen Spiel-Funktionalitäten durch welche Technologie umgesetzt werden kann, oder ob es evtl. Hürden auf technologischer Ebene gibt, die durch Kompromisse in der Spiel-Mechanik gelöst werden müssen.

Web-Technologien

Da das Spiel mittels Web-Technologien realisiert werden soll, muss jegliche Kommunikation, unabhängig vom umgesetzten Modus, mittels Server-Client-Mechanismen gelöst werden können.

Die größten Anforderungen an Kommunikation stellt dabei der Action-Aspekt im Kampfareal dar. Hier wird Echtzeit-Kommunikation auf einem ziemlich hohen Niveau benötigt um möglichst kleine Latenzen zwischen Updates der Spielelemente zu vermeiden und so das Spiel überhaupt spielbar zu machen. Als Lösung für dieses Problem bietet sich der Websocket-Standard an, der von vielen Browsern unterstützt wird. Dieser Standard ermöglicht bidirektionale, asynchrone Kommunikation, ohne dass erneute TCP-Verbindungen aufgebaut werden müssen und damit kein großer Overhead[20] entsteht. Der Client muss sich dazu einmalig mit einem Websocket unterstützenden Server verbinden. Über Websockets versendete Nachrichten können auch mit Objekten als Daten gefüllt werden, welche dann meist im JSON[21]-Format übertragen werden. Durch die weite Verbreitung des JSON-Formats ist die weitere Verarbeitung der Objekte stark

[20] Overhead – In der IT-Umgebung gängiger Begriff für Meta-Daten oder Programmcode der zusätzlich zur eigentlich Funktion ausgeführt oder versendet werden muss
[21] JSON – JavaScript Object Notation

vereinfacht und standardisiert.
Alle anderen Aspekte des Spiels sind sehr gut vereinbar mit einem webbasierten Server-Client-Modell:
Daten können vom Server vorgehalten und, falls nötig auch gespeichert werden. Dadurch wird keine Peer-to-Peer-Kommunikation[22] nötig und alle Synchronisation von Daten geschieht über den Webserver. Die gemeinsame Datenhaltung in einer Komponente hat weiterhin den Vorteil, dass viele Überprüfungen auf Konsistenz der Daten und Regeln des Spiels ebenfalls nur an einer einzigen Stelle implementiert werden müssen. Die weitere Dynamisierung, also auch die Reaktion auf Eingaben eines Nutzers kann durch JavaScript und die Standard-DOM-API ausgeführt werden. Das erlaubt jede Kommunikation – Server-Client und Spieler-Client über eine Schnittstelle abzufertigen. Auf dem Client können Inhalte durch HTML[23]-Elemente repräsentiert und der Stil mittels CSS[24]-Regeln angepasst werden.
Da aber auch 3D-Technologien in diese Realisierung einfließen sollen, bedarf es mehr als nur HTML und CSS zur Darstellung der Spielinhalte. Allein mit diesen Mitteln ist es sehr schwierig bis unmöglich teils komplexe Spielinhalte darzustellen.

3D-Technologien

Die Darstellung der Spielinhalte soll mittels 3D-Technologien stattfinden. Die Lösung für dieses Problem bieten "Canvas" und WebGL. Das HTML-Element "Canvas" ermöglicht es Inhalte im Webbrowser sehr frei zu gestalten. Das HTML-Element bietet dabei nur eine Leinwand (engl. Canvas) und die Inhalte werden mittels JavaScript programmiert. Gepaart mit dem WebGL-Standard ergibt sich die Möglichkeit vielfältige, aufwändige 3D-Darstellungen im Webbrowser zu synthetisieren. Denn: Der WebGl-Standard erlaubt es ohne Erweiterungen hardwarebeschleunigte[25] 3D-Grafiken im Browser darzustellen. WebGL-Elemente können dabei beliebig mit herkömmlichen Webseiten-Elementen, also HTML und CSS, kombiniert werden.

Das "Canvas"-Element bildet also die Grundlage für eine mittels WebGL programmierte Szenerie des Spiels Archon, die mit Daten aus der Echtzeitkommunikation über den WebSocket-Standard gespeist wird. Da es, oberflächlich betrachtet, keinerlei Einschränkungen für eine Realisierung von Archon mittels der oben genannten Technologien gibt folgt nun die Implementierung des Spiels.

[22] Peer-to-Peer – Kommunikation zwischen gleichen End-Teilnehmern ohne einen Server als Medium
[23] HTML – Hypertext Markup Language ist eine Sprache um Dokumente im Webbrowser in ihrem Inhalt zu strukturieren
[24] CSS – Cascading Style Sheets bilden die Regeln zum Aussehen von Webdokumenten
[25] Hardwarebeschleunigt – Rechenintensive Aufgaben werden vom Hauptprozessor eines Computers an dafür dedizierte Hardware delegiert, heute vermehrt Grafikkarten und grafikintensive Aufgaben

2.3. Implementierung

Die Implementierung von Archon wird im Folgenden gezeigt. Dabei wird zunächst betrachtet, welche Frameworks, Software-Bibliotheken, Werkzeuge und Hilfsmittel benutzt werden und welche Vereinfachungen bezüglich des Spiels getroffen werden um den Entwicklungsaufwand und die Komplexität möglichst weit zu reduzieren. Anschließend wird aus den Anforderungen von Technologien und Spiel eine Grobarchitektur erstellt, die mit jedem Entwicklungsschritt verfeinert und ggf. angepasst wird.

2.3.1. benötigte Technologien und Frameworks

Durch die bisherigen Ausarbeitungen steht fest, dass HTML, CSS und JavaScript zur Pflicht für die Implementierung des Clients werden um alle technologischen Anforderungen einhalten zu können. Desweiteren gibt es aber noch Bibliotheken, die die Entwicklung des Clients beschleunigen und vereinfachen, sowie den Umgang mit Websockets und WebGL erleichtern. Außerdem muss noch die Festlegung auf einen Webserver stattfinden, der bisher lediglich der Einschränkung bedarf, dass Websockets unterstützt werden müssen.

Die Liste an Webservern ist ähnlich groß, wie die von Programmiersprachen. Viele grundlegende Sprachen, wie die .NET-Umgebung, C++, PHP u. a. unterstützen dabei über Bibliotheken oder Module auch den Websocket-Standard. Um die Entwicklung möglichst aufwandsarm zu gestalten und schnelle Fortschritte zu ermöglichen wurde sich in diesem Fall auf die node.js-Umgebung festgelegt, da hier der Server mittels JavaScript programmiert wird, was das zusätzliche Erlernen einer weiteren Programmiersprache und damit verbundene Kontext-Wechsel und Einarbeitungen in Eigenheiten, Bibliotheken und Programmiertechniken erspart.
Um die Entwicklung weiterhin zu beschleunigen wurde das Framework "Express" verwendet, welches es erlaubt auf einfachstem Weg über Routing-Mechanismen Webseiten-Anfragen eines Webservers zu beantworten.
Als Hilfsmittel für den Websocket-Standard wurde sich für Socket.io entschieden. Socket.io ist eine Library, welche Websockets wrapped und den Umgang stark vereinfacht. Socket.io erlaubt dabei eine stark Event-basierte Kommunikation, was dem Event-Emitter-Prinzip der node.js-Umgebung stark ähnelt und einen gleich Programmierstil ermöglicht. Die Library bietet ebenfalls einen Clientseitigen Teil an, welcher sich mit entsprechenden Optionen automatisch mit der Basis-Route des Webserver verbindet.
Für die Darstellung, also die Implementierung des WebGL-Standard wird three.js genutzt. three.js ist eine stark ausgebaute und weit verbreitete Library für Hochleistungs-3D-Entwicklung im Webbrowser. three.js besitzt weiterhin eine umfangreiche Dokumentation und eine sehr große Auswahl an Beispiel-Applikationen, welche den Einstieg und auch den weiteren Entwicklungsprozess beschleunigen. three.js arbeitet dabei mit einer Szene für die Zusammenfassung aller graphischen Objekte. Ein Kamera-Objekt legt den zu sehenden Ausschnitt der Szene fest und

ein Renderer-Objekt zeichnet die Szene dann auf ein HTML-Canvas-Element. Dabei wird der bereits erwähnte WebGL-Standard als Kontext des Canvas benutzt. Um graphische Objekte zu erzeugen können dabei entweder Modelle mit populären Werkzeugen wie "Blender" oder "Maya" erzeugt und anschließend importiert werden, oder auch aus vorhandenen, primitiven Formen gebildet werden. Graphische Objekte bestehen dabei immer aus zwei Komponenten: Der Form (Geometry) und einem Material. Die Form gibt dabei die Eigenschaften wie Position, Skalierung und eben die einzelnen Punkte im 3D-Raum vor. Das Material entscheidet über die Interaktion mit Lichtquellen, sowie Farbe und Reflexion. Ein Objekt muss dabei nicht aus einem einzelnen Material bestehen. Außerdem können auch Bilder auf die Oberfläche von Formen projiziert werden. Da viele Projekte, die mit three.js implementiert wurden, frei verfügbar sind, findet man außerdem viele Beispielimplementierung, die auch produktiv im Einsatz sind.

Die node.js-Umgebung erlaubt es außerdem sogenannte Dev-Dependencies einzufügen, also Abhängigkeiten oder Bibliotheken, welche nur im Entwicklungsprozess benutzt werden und nicht in der Produktionsversion zu finden sind. Das folgende Kapitel befasst sich mit diesen Dev-Dependencies und anderen Hilftmitteln und Vereinfachungen für die Implementierung.

2.3.2. Hilfsmittel und Vereinfachungen

Dev-Dependencies

Als größte Dev-Dependency sei an dieser Stelle TypeScript zu nennen. TypeScript ist ein sogenanntes Superset zu JavaScript: Jedes valide JavaScript-Programm ist also auch ein valides TypeScript-Programm. TypeScript wird direkt zu JavaScript mittels des Compilers tsc kompiliert. TypeScript fügt zu normalem JavaScript-Code statische Typen hinzu, sodass eine statische Analyse des Codes zur Kompilierzeit auf Fehler erfolgen kann, was die Sicherheit von Schnittstellen und Funktionsaufrufen erhöht. Zu TypeScript gehören als weitere Dev-Dependency die zugehörigen Typen-Deklarationen für bereits existierende JavaScript-Bibliotheken, sodass diese auch im TypeScript korrekt erkannt und von einer IDE[26] bzw. einem Code-Editor mit Auto-Vervollständigung genutzt werden können.

Werkzeuge

Zur Entwicklung von Applikationen sind nicht nur Bibliotheken zur Unterstützung nötig, sondern auch Werkzeuge. So z. B. der erwähnte Editor und in diesem Fall auch eine Testumgebung in Form eines Webbrowsers. Dieser Abschnitt enthält alle benutzten Werkzeuge zur Implementierung der Applikation.

[26] Integrated Development Environment – Ein Entwicklungswerkzeug, dass die Aufgaben der Softwareentwicklung ohne Kontextwechsel erlaubt.

Als Editor wurde Visual Studio Code (VS Code)[27] benutzt. Es handelt sich dabei um einen universellen Editor. Die Erweiterungen, welche auch von vielen Nutzern des Editors geschrieben werden, spezialisieren diesen Editor und geben ihm weitere Funktionalitäten für diverse Programmiersprachen, Werkzeuge und Umgebungen. Die Liste an installierten Erweiterungen für diese Entwicklung ist an diese Arbeit angehängt[28].

Zum Test der Applikation wurden die Browser Mozilla Firefox[29] in der Version 61.0.1 und Google Chrome[30] in der Version 68 benutzt. Dabei handelt es sich um zwei Browser, die sehr viele Internet-Standards unterstützen und gute Werkzeuge für Entwickler bereitstellen. Außerdem sind die zwei Browser sehr weit verbreitet und habe viele Nutzer[31].

Desweiteren wurden die Terminalwerkzeuge der benutzten Node.js-Umgebung genutzt und der Paketmanager npm von Node.js[32].

Für experimentelle Komponenten und kleine Tests wurden die Plattformen JSFiddle[33] und der Playground von Typescript[34] benutzt.

Für für Klassen- und Komponentendiagramme u. a. Visualisierungen wurde das UML-Werkzeug[35] Visual Paradigm[36] benutzt.

Vereinfachungen

Wie im Abschnitt Werkzeuge bereits erwähnt wurden zum Test die beiden Browser Mozilla Firefox und Google Chrome herangezogen. Weiterhin wurde die Vereinfachung getroffen, dass die Applikation nur auf diesen Browsern funktionfähig sein muss. Auf die Kompatibilität zu anderen und älteren Browsern wurde kein Fokus gelegt.
Weiterhin wird nur der Spieler-gegen-Spieler-Modus implementiert. Die Nachbildung des virtuellen Spielers würde nichttrivialen Aufwand nach sich ziehen, da die originale Implementierung unklar und nicht ersichtlich ist.
Außerdem wird die Audio-API vernachlässigt und auch Touch-Events werden außer Acht gelassen. Auch wird die Parallelisierung von Spielen nicht Fokus dieser Implementierung sein. Das Ziel eine möglichst einfache Erstimplementierung und eine Parallelisierung ist bei geeigneter Architektur kaum Mehrarbeit im Anschluss dieser Arbeit.

[27] Visual Studio Code – Ein freier Editor von Microsoft: https://code.visualstudio.com/
[28] Siehe Anhang: A.1
[29] Mehr zu Mozilla Firefox unter https://www.mozilla.org/de/firefox/
[30] Mehr zu Google Chrome unter https://www.google.de/chrome/
[31] Quelle: https://www.w3counter.com/globalstats.php
[32] Mehr zu Node.js und npm unter https://nodejs.org/en/ und https://www.npmjs.com/
[33] JSFiddle ist eine Online-Plattform, die Editoren zur Webentwicklung enhält und eine Live-Vorschau: https://jsfiddle.net/
[34] Ein Online-Editor für TypeScript-Code, der eine Live-Ausführung ermöglicht: https://www.typescriptlang.org/play/index.html
[35] Die Unified Modeling Language (UML) ist eine standardisierte, graphische Sprache zur Visualisierung von Software und anderen Systemen.
[36] Mehr zu Visual Paradigm unter https://www.visual-paradigm.com/

2.3.3. Architektur

The goal of every video game is to present the user(s) with a situation, accept their input, interpret those signals into actions, and calculate a new situation resulting from those acts. Games are constantly looping through these stages, over and over, until some end condition occurs (such as winning, losing, or exiting to go to bed). Not surprisingly, this pattern corresponds to how a game engine is programmed. The specifics depend on the game. [13]

Der obige Paragraph sagt sehr genau aus, wie Spiele im allgemeinen aufgebaut sein sollten. Auch die Architektur dieser Realisierung orientiert sich daher an dem EVA[37]-Ansatz. Die Architektur folgt außerdem den Prinzipien einer objektorientieren Analyse. Aus dieser Analyse soll dann eine Software-Architektur entstehen, die alle Anforderungen vereint und einen möglichst gut wartbaren und erweiterbaren Rahmen darstellt.
Am Anfang einer objektorientieren Analyse steht meist ein Begriffsmodell, dass die Beziehungen einzelner Elemente aufzeigt, ohne sich direkt auf Code-Ebene zu bewegen. Die Abbildung 2.4

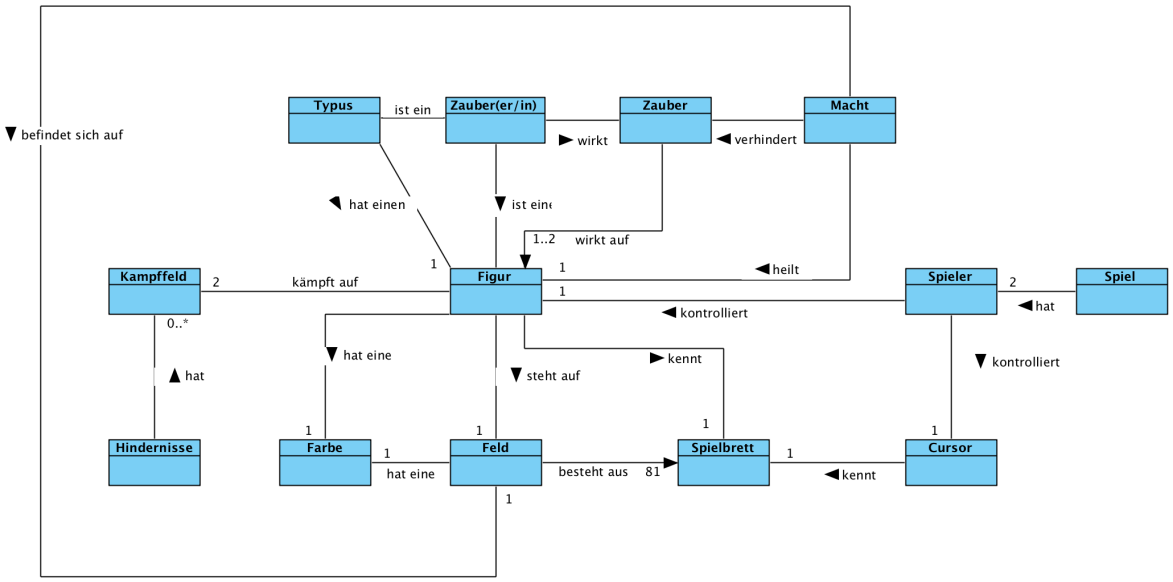

Abb. 2.4.: Begriffsmodell

stellt also die im Kapitel 2.2.1 dargestellten Inhalte grafisch dar, was den Entwurf und die Zusammenhänge von Klassen und Objekten im Folgenden stark erleichtert. Das Begriffsmodel ist nur für die reine Spielelogik gültig und beinhaltet noch keinerlei Ansätze für den technologischen Aspekt der Architektur! Die folgenden Abschnitte sollen daher den Entwurf der generellen Systemarchitektur näher erläutern. Dabei wird, soweit möglich, noch nicht auf externe Abhängigkeiten eingegangen und die Architektur auch möglichst frei von konkreten Festlegungen auf Technologien und Frameworks gehalten. Am Ende dieses Kapitels werden die technologischen Abhängigkeiten

[37] EVA – Eingang-Verarbeitung-Ausgang

aufgezeigt und mit in die Architektur integriert. Dieses Vorgehen erhöht die Übersicht stark und vereinfach die einzelnen Teilmodelle stark.

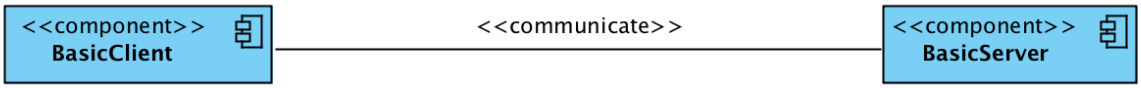

Abb. 2.5.: Client-Server-Architektur

Die Umsetzung mittels Webtechnologien verlangt es zwei generelle Komponenten zu entwickeln: Einen Webserver, sowie Clients welche sich mit dem Server verbinden. Der Webserver und die Clients müssen dabei in beide Richtungen kommunizieren, was schematisch in Abbildung 2.5 aufgezeigt ist.

Da die Darstellung des Spiels nur in den Clients erfolgt und im Server sich eine Datenhaltung mit entsprechenden Operationen zur Manipulation befindet, bietet sich ein Ansatz nach dem Model-View-Controller-Muster (MVC) an. Bei MVC handelt es sich um ein klassisches Entwurfsmuster der Softwareentwicklung, dass Abhängigkeiten reduzieren und Funktionalitäten kapseln soll. Das Entwurfsmuster sieht dabei vor die wesentlichen Aufgaben von Applikationen so aufzutrennen, dass die Ansicht, in den meisten Fällen eine graphische Benutzeroberfläche, möglichst unabhängig von den zugrunde liegenden Daten, also dem (Daten-)Modell ist. Dazu wird eine Klasse zur Kommunikation, der Controller, von Daten und Ansicht erstellt. Der Controller sorgt dabei dafür, dass Kommandos vom Benutzer, also in der Ansicht, an die Daten und dazugehörige Operationen weitergeleitet werden. Hier wurde eine klassische Interpretierung des MVC-Musters

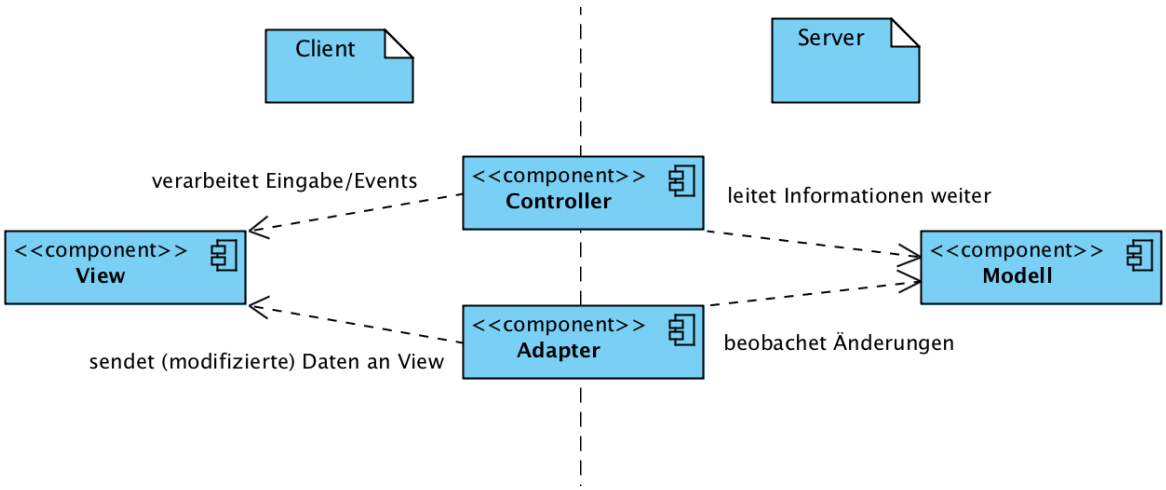

Abb. 2.6.: Darstellung der MVC-Architektur mit Aufgaben

gewählt, bei der der Controller lediglich Informationen aus Events der View verarbeitet und an

das Modell weiterleitet. Für den zweiten Kommunikationsweg, vom Modell zur View, wurde eine separat zu implementierende Komponente gewählt: Der Adapter. Die Abbildung 2.6 zeigt die einzelnen Komponenten und die Aufgaben der Kommunikationskomponenten auf. In der Mitte der Abbildung 2.6 ist eine Trennlinie zu erkennen. Sie soll die physische, sowie logische Grenze zwischen der Server- und der Client-Komponente repräsentieren. Auch zu erkennen ist, dass Controller und Adapter über diese Grenzen hinweg operieren müssen. Die View befindet sich auf dem Client und bekommt Benutzereingaben und soll möglichst aktuelle Daten aus dem Modell zur Anzeige bringen. Währenddessen befindet sich das Datenmodell des Spiels auf dem Server und implementiert die Regeln den Spiels, die für eine Integrität der Daten sorgen. Nun gibt es die Möglichkeit, dass der Controller komplett auf dem Client implementiert wird. Dies bringt den Vorteil, dass eine einzige Klasse an einer Position im Code hat, die für einen Kommunikationsweg zuständig ist. Der große Nachteil kommt jedoch auf der Server-Seite zum tragen: Hier muss das Modell stark an die Nachrichten des Controllers binden und Kommunikation im Datenmodell durchführen. Wenn dagegen der geteilte Ansatz eines Controller-Senders und -Empfängers gewählt wird, bleiben Nachrichten, also Kommunikation, auch in Klassen, die für Kommunikation zuständig sind und müssen nicht in anderen Klasse verarbeitet werden. Als Nachteil dieses Ansatzes ergibt sich der erhöhte Aufwand bei der Erstellung einer zusätzlichen Klasse. Die Vor- und Nachteile der Ansätze für den Entwurf des Controllers sind vollständig übertragbar auf den Adapter.
Es wurde sich im Folgenden für den Ansatz der geteilten Controller und Adapter entschieden, sodass sich 4 Klassen zur Kommunikation zwischen Ansicht und Datenmodell ergeben: Jeweils ein Controller und ein Adapter auf Server- bzw. Client-Seite. Die Aufgaben der Klassen sind bereits in Abbildung 2.6 an den Pfeilen abzulesen.
Aus den Anpassungen für die Kommunikation ergibt sich damit der Aufbau aus Abbildung 2.7.

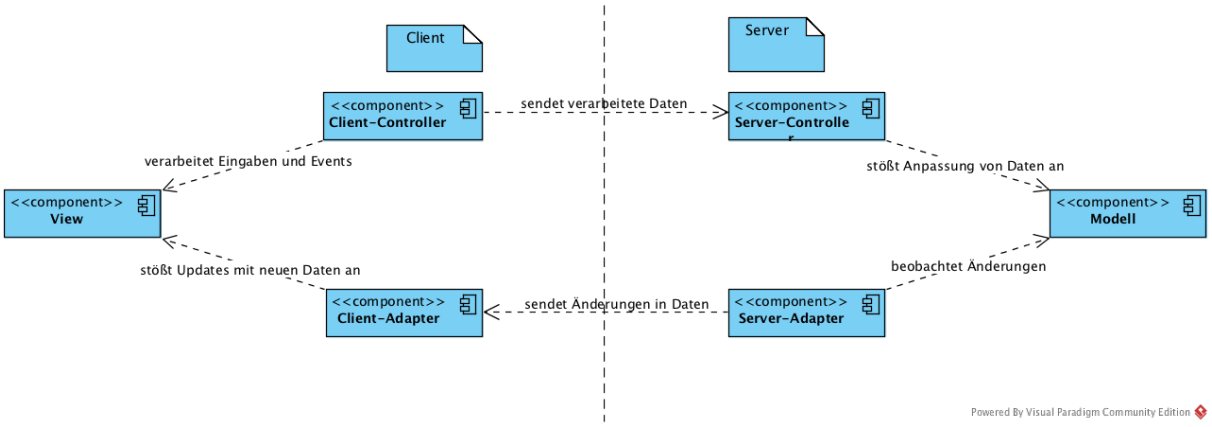

Abb. 2.7.: Aufgeteilte Kommunikationsklassen

Da zur Kommunikation von Daten zwangläufig auch zu übertragende Daten gehören muss das Datenmodell so gestaltet werden, dass immer auf die aktuellen Daten zugegriffen werden kann. Außerdem müssen die Daten in einem Format vorliegen, dass eine Übertragung erlaubt. Da

gleichzeitig lesender Zugriff im Client und schreibender Zugriff im Server auf die komplette Datenhaltung möglich sein muss, bietet es sich an ein zentrales Modell aufzustellen. Das zentrale Modell wird dann in einem Initialzustand vom Server an den Client gesendet. Im Spielbetrieb müssen dann nur noch Aktualisierungen übertragen werden, was die Last der Kommunikationswege deutlich reduziert. Da das Datenmodell stark an das, eingangs dieses Kapitels, aufgestellte Begriffsmodell angelehnt sein muss, ist schnell ersichtlich welche Menge an Daten und auch welche Daten vorgehalten werden müssen. Die Abbildung 2.8 zeigt die gewählten Daten-Schnittstellen für das Datenmodell.

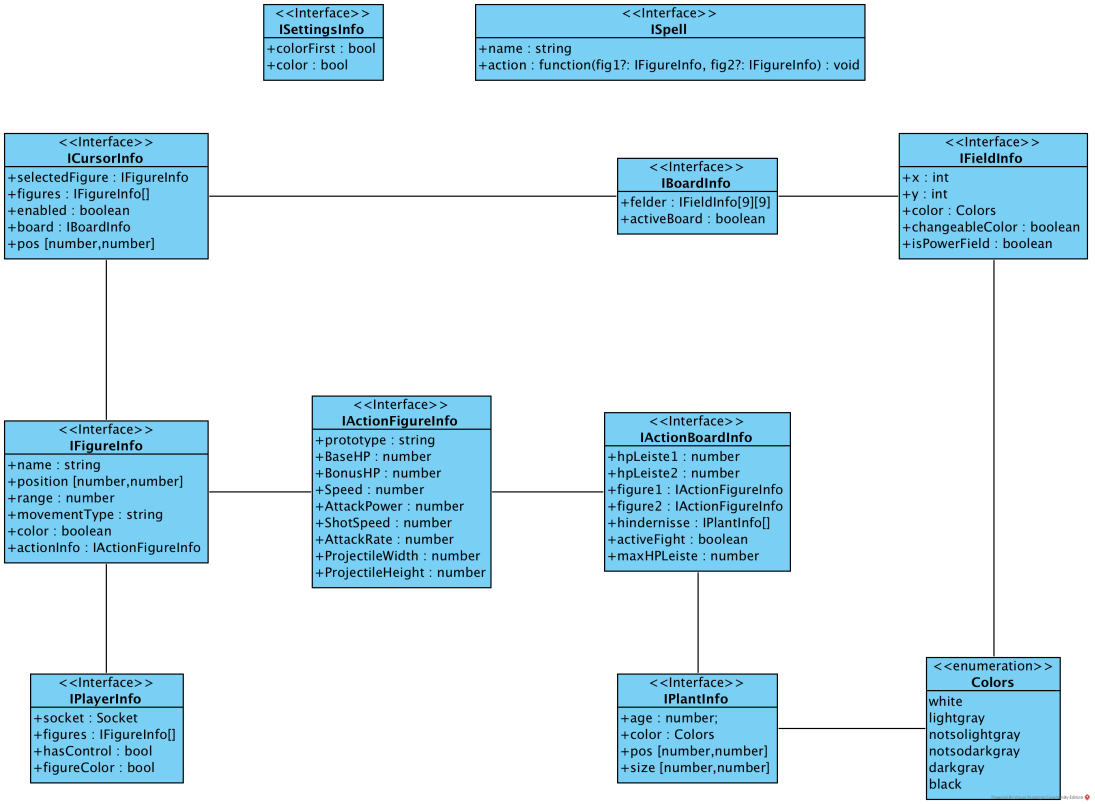

Abb. 2.8.: Daten-Schnittstellen für Datenmodell

Die Daten-Schnittstellen enthalten keine Methoden. Sie sind nur Container bzw. Verträge für Daten. Die Schnittstellen für die Einstellungen und die Zauber besitzen keine direkten Beziehungen zu den anderen Schnittstellen, sondern stellen ihre Beziehungen erst durch die Regeln des Spiels her. Die Ausarbeitung der Daten für die einzelnen Schnittstellen ergibt sich aus dem Begriffsmodell und der Beschreibung des Spiels. Die meisten Schnittstellen werden zur gebündelten Übertragung in der Schnittstelle IGameModel vereint:

```
export interface IGameModel {
    _players: IPlayerInfo[];
    _settings: ISettingsInfo;
    _observers: IServerAdapter[];
    _board: IBoardInfo;
    _blackFigures: IFigureInfo[];
    _whiteFigures: IFigureInfo[];
    _defeatedFiguresWhite: IFigureInfo[];
    _defeatedFiguresBlack: IFigureInfo[];
    _elementals: IActionFigureInfo[];
    _actionField: IActionBoardInfo;
    _spells: ISpell[];
}
```

Abb. 2.9.: Daten-Schnittstelle IGameModel

Zu jeder Daten-Schnittstelle müssen jetzt die passenden Klassen für Anzeige und Logik entworfen werden um die wesentlichen Teile der Grobarchitektur fertigzustellen. Zu jeder Info-Schnittstelle gibt es also zwei weitere Klassen z. B. zur IBoardInfo-Schnittstelle die Klassen Board für Spiellogik im Server und BoardView zur Anzeige im Client. Die Komponenten View und Modell werden im Folgenden nacheinander vorgestellt.

View

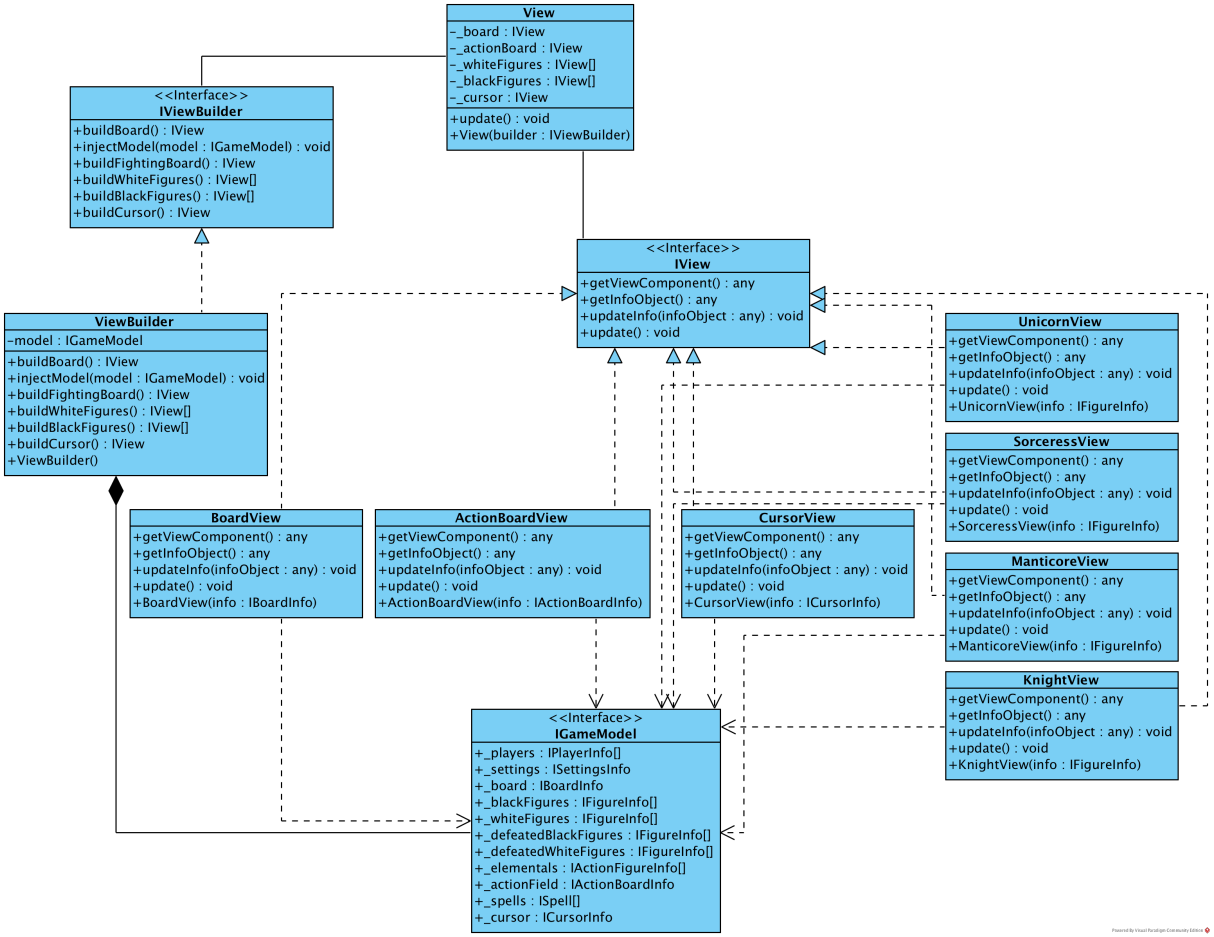

Abb. 2.10.: leicht vereinfachtes Klassendiagramm der View

Die Abbildung 2.10 zeigt eine leicht vereinfachte Darstellung der View-Komponente. Die Vereinfachungen bestehen in der reduzierten Darstellung der Anzeige-Klassen für die einzelnen Figurentypen. Exemplarisch sind hier nur die Typen "Knight", "Manticore", "Sorceress" und "Unicorn" dargestellt, da sich die Klassen nur in der Implementierung ihres Konstruktors unterscheiden, der das eigentliche 3D-Objekt anhand eines Informationsobjektes aus dem Datenmodell erzeugt.

Für den "ViewBuilder" wurde der Ansatz einer Injektionsmethode zur Anbindung an das Datenmodell gewählt. Damit kann die Objekt-Erzeugung vom Zeitpunkt der Injektion der Abhängigkeit entkoppelt werden. Das ist hier sinnvoll, da das Datenmodell zu einem Zeitpunkt zum Client gesendet wird, den dieser nicht beeinflussen kann. Der "ViewBuilder" hat die Aufgabe die einzelnen Erzeugungen von Anzeigeobjekten von der View zu entkoppeln, was den einfacheren Tausch eines Frameworks für die 3D-Objekte erlaubt und die bessere Bündelung von Abhängigkeiten möglich macht. Außerdem entkoppelt er die View von der Kenntnis über ein Datenmodell, sodass diese lediglich Anzeigeobjekte besitzt und auch nur für deren Aktualisierungen und tatsächliche Anzeige zuständig ist.

Desweiteren besitzen alle Anzeigeklassen eine Referenz auf ihr Informationsobjekt aus dem

Datenmodell, sodass der Adapter Änderungen nur in das Modell auf dem Client laden muss und die View alle ihre Anzeigeobjekte in einer Schleife regelmäßig zu Aktualisierungen auffordert. Die einzelnen Anzeigeklassen interpretieren dabei die Informationen aus ihren Datenobjekten und wandeln diese in eine passende Darstellung basierend auf dem gewählten 3D-Framework.

Modell

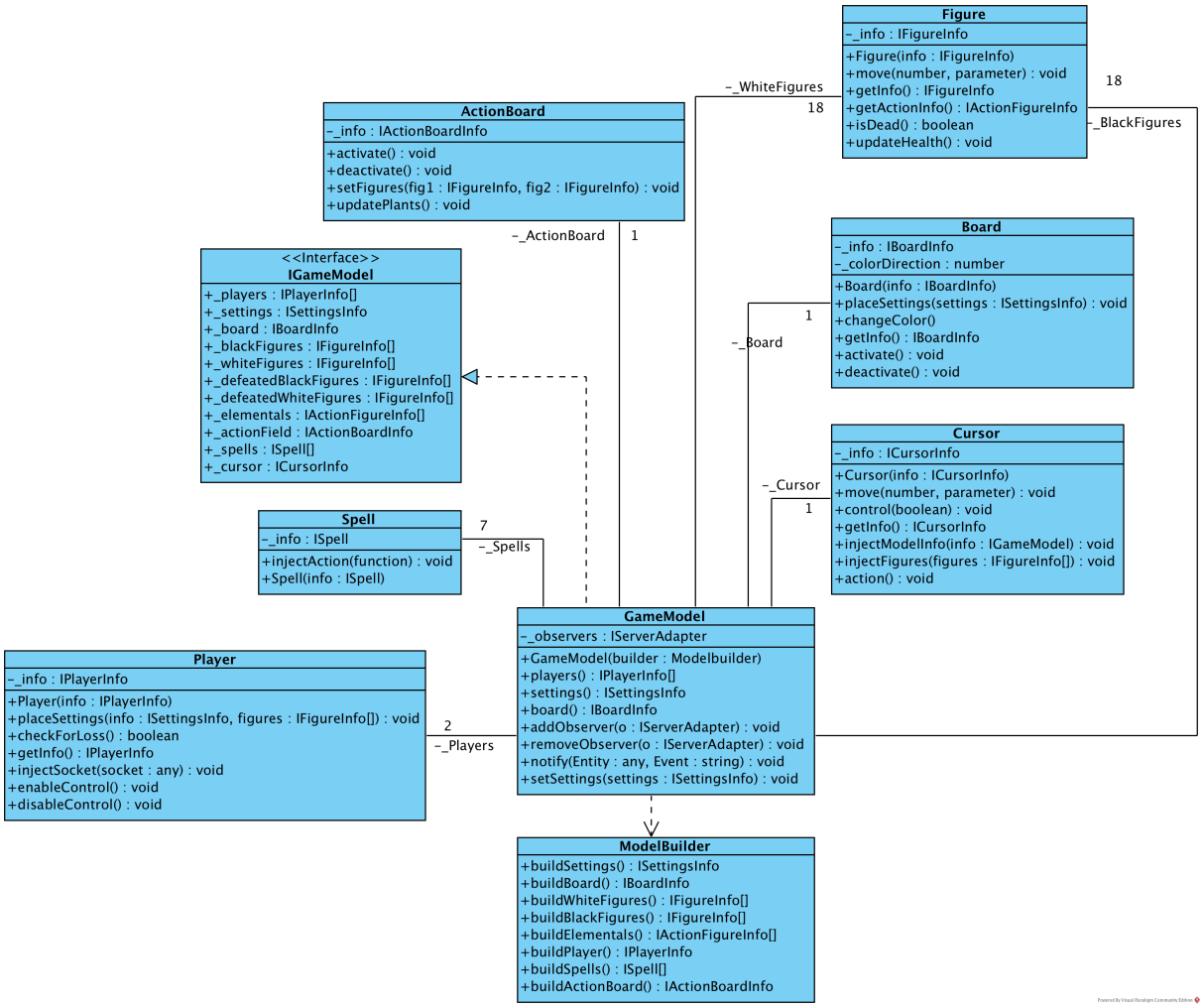

Abb. 2.11.: Klassendiagramm des Modells

Das Modell wurde sehr ähnlich zur View strukturiert. Die Klasse "ModelBuilder" ist für die korrekte Erzeugung eines Initialzustandes des Datenmodells zuständig. Während die Klasse "GameModel" die Erzeugung der Logik-Objekte selbst übernimmt, da diese keine aufwändigen Initialisierungsroutinen besitzen. Jedes Logik-Objekt kann auf sein Informationsobjekt zugreifen und besitzt Methoden zum Schreiben der Daten. Die Methoden sind dabei zuständig für das Überprüfen von Spielregeln, wie die Reichweite von Figuren oder das Ändern der Farbe der Felder auf dem Spielbrett.

An dieser Stelle sei noch die Benutzung des Beobachter-Musters zu erwähnen. Die Klasse "GameModel" muss den Adapter nach der Änderung von Daten informieren können, sodass der Adapter die geänderten Daten an den Client senden kann. Dafür trägt sich der Adapter in die Liste der Beobachter im "GameModel" ein und wird anschließend mittels der "notify"-Methode vom "GameModel" darüber benachrichtigt welche Daten sich geändert haben und kann die Änderungen einfach weiterleiten.

Da nun alle wesentlichen Komponenten des System und deren Aufbau beschrieben wurden, bleibt nun die Ansicht der technologischen Abhängigkeiten zu zeigen.
Zu sehen ist, dass die erwähnten Dev-Dependencies der Node.js-Umgebung auch im Client benutzt werden. Dies wird durch die Benutzung eines Paketes zur Bündelung und Minimierung der JavaScript-Dateien im Client ermöglicht: Browserify. Browserify wird dabei mit dem Paket tsify kombiniert um den Typescript-Code vorher bündeln und in JavaScript zu konvertieren.

Außerdem sind die Abhängigkeiten zu Jest und Chai in Abbildung 2.12 aufgezeigt. Jest ist eine Bibliothek, die eine Umgebung für den Test von JavaScript-Code bereitstellt, es werden dabei auch Berichte über den Erfolg von Tests und die Abdeckungsrate der Tests erstellt. Chai ist eine Bibliothek, die weitere Methoden für das Abfragen in Tests bereitstellt und damit das Test von JavaScript-Code weiter beschleunigt und vereinfacht.

Abhängigkeiten

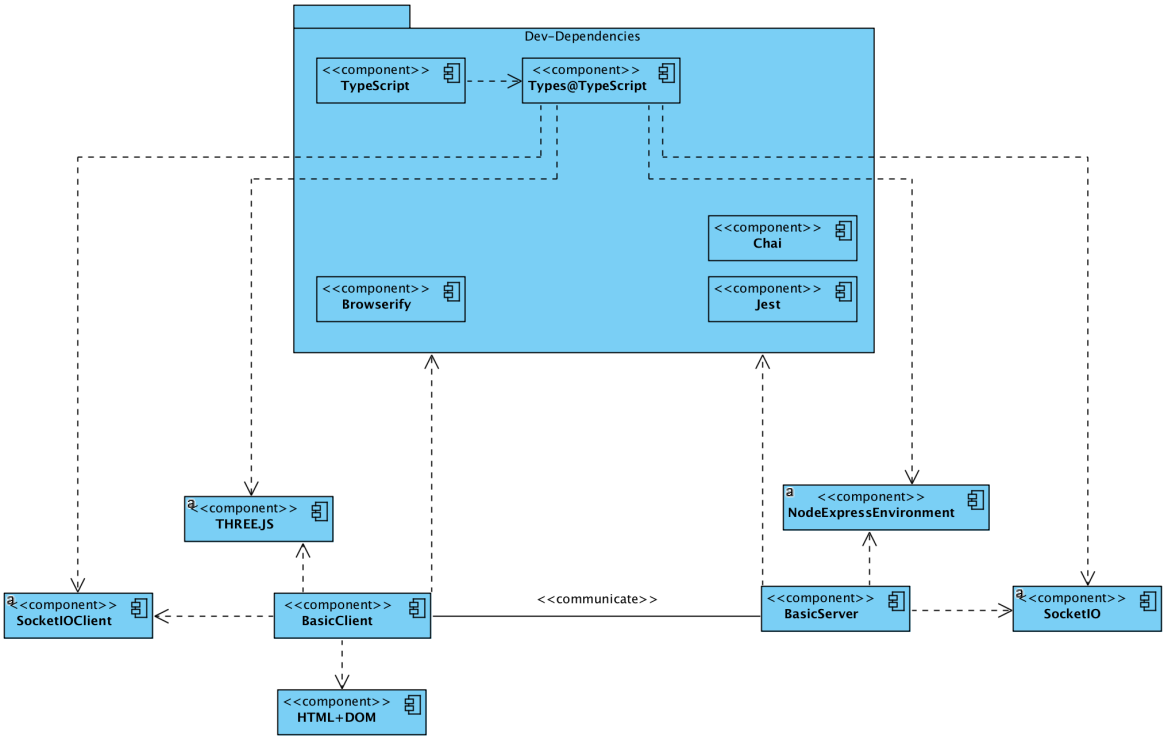

Abb. 2.12.: Komponentendiagramm der Abhängigkeiten

Die Grafik 2.12 zeigt eine Übersicht über die Abhängigkeiten des Servers und des Clients zu den gewählten Frameworks und Software-Bibliotheken. Da deren Verwendung schon in Kapitel 2.3.1 bzw. 2.3.2 erläutert wurde dient diese Übersicht nur noch einmal zur Visualisierung und als Einstieg für die Implementierung der Architektur im folgenden Abschnitt.

2.3.4. Schritte der Implementierung

Bei der Implementierung wurde sich zunächst mit den einzelnen Technologien aus der Grafik 2.12 vertraut gemacht. Dafür wurden verschiedene Beispiele herangezogen und die Dokumentation, sowie ggf. vorhandene Getting-Started-Guides durchgearbeitet.[38] Anschließend wurde die Entwicklungsumgebung aufgesetzt auf Basis eines Starter-Projektes für Node.js-Projekte mit Typescript und Express.[1] Dabei wurde das Starter-Projekt so angepasst, dass die Applikation auf das Bereitstellen von statischen Webseiten spezialisiert ist. Einige Skripte zum Erstellen, Debuggen, Bündeln und Minimieren der Applikation wurden angepasst bzw. hinzugefügt. Der Aufbau des Projektes wird im Folgenden näher erläutert.

Projektaufbau

Abb. 2.13.: Das Projektverzeichnis

Die Abbildung 2.13 zeigt das Projektverzeichnis. Ordner sind dabei blau dargestellt und Dateien gelb.
Generell ist ein Projekt mit Node.js immer auf einer Datei mit dem Namen "package.json" aufgebaut. Diese Datei enthält neben wichtigen Meta-Informationen, wie dem Namen des Projektes und seiner Version, auch Informationen, wie bestimme Befehle wie ein Start, das Debuggen[39] etc. ausgeführt werden sollen. Auch sind in dieser Datei die (Entwicklungs-)Abhängigkeiten enthalten. Dabei wird ein Paket immer mit seiner benutzen Version angegeben und möglichst große Portabilität zu erreichen und einen Projekt-Stand exakt reproduzieren zu können. Die Abbildung 2.14 zeigt einen Ausschnitt der Datei. Dabei sind aus Übersichtsgründen die Abhängigkeiten und Skripte ausgeblendet.

[38] So zum Beispiel für Socket.IO: https://socket.io/get-started/chat/
[39] Debugging beschreibt die Untersuchung einer laufenden Applikationen auf Fehler und deren Behebung

Abb. 2.14.: Ein Ausschnitt der Datei package.json

Für die Verwaltung der Pakete und Abhängigkeiten wird dabei der Paketmanager npm[40] benutzt, der auch die Ausführung der in package.json eingetragenen Befehle und Skripte übernimmt. Die Befehle und Skripte dieses Projektes werden in Tabelle 2.1 aufgelistet und beschrieben.
Alle weiteren Dateien mit der Endung *.json* aus Abbildung 2.13 stellen Konfigurationsdateien dar. Das Verzeichnis *dist* enthält die gebaute Version der Applikation – aus diesem Verzeichnis wird die Anwendung gestartet und es enthält alle fertigen Applikationsdateien. Das Verzeichnis *test* enthält alle Dateien, die Tests definieren. Das Verzeichnis *src* enthält alle Quelldateien. Die unterliegende Struktur des Verzeichnisses *src* wird beim Bau der Applikation immer auf das Verzeichnis *dist* übertragen.

[40] npm – Node.js package manager, verwaltet das Laden, Aktualisieren und Einbinden von externen Paketen in ein Projekt

Befehl	Beschreibung
build	Baut die gesamte Applikation und führt bundle, sowie tslint aus, kopiert auch die statischen Dateien
build-ts	konvertiert nur TypeScript in JavaScript
bundle	Minimiert, konvertiert und bündelt die Client-Komponente zu einer Datei bundle.js
copy-static-assets	Kopiert statische Dateien wie Bibliotheken und Bilder mit Hilfe des Skriptes aus copyStaticAssets.ts in das Ausgabeverzeichnis
createDiagrams	Erzeugt mit Hilfe des Pakets tsviz[41] und Graphviz[42] Klassendiagramme von diversen Unterverzeichnissen
debug	Baut die Applikation und startet sie dann im Debug-Modus
serve	Startet die Anwendung mit server.js als Einstiegspunkt
serve-debug	Startet nodemon[43], welcher die Datei server.js nach Änderungen überwacht und automatisch die Anwendung neustartet
start	Ein Alias für serve
test	Startet jest und führt alle Testfälle aus und nimmt die Abdeckung auf
tslint	Startet tslint und überprüft anhand von tsconfig.json den Code nach Lesbarkeit unf funktionalen Fehlern
watch	Startet nodemon und überwacht Node.js und Typescript mittels Dateien, die eine Zuordnung von TypeScript zu JavaScript enthalten
watch-*	überwacht jeweilige Dateien auf Änderungen, während die Applikation läuft. Baut und Startet die Applikation ggf. neu

Tabelle 2.1.: Skripte aus package.json

Anfänge der Entwicklung

Zunächst ist auf Basis des Projektes eine HTML-Seite mit einem Canvas-Element angelegt worden, sowie ein Aufruft der Socket.IO-Bibliothek auf der Seite. Der Aufruf verbindet sich mit dem Webserver.
Passend dazu wurde eine einfache Klasse "Server" erstellt, die einen http-Server, sowie einen Websocket-Server bereitstellt und initialisiert. Die Abbildungen 2.1 und 2.15 zeigen diese beiden Entwürfe respektive.

[41] tsviz – Ein npm-Paket, dass aus Graphviz aufbaut und aus TypeScript-Code Klassendiagramme erzeugt
[42] Graphviz ist eine freie Visualisierungssoftware für Graphen und Diagramme jeglicher Art
[43] nodemon ist eine Applikation, die Dateiänderungen im Projektverzeichnis überwacht und daraufhin die Node.js-Umgebung neustartet

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset=utf-8>
5          <title>My first three.js app</title>
6          <style>
7              body { margin: 0; }
8              canvas { width: 100%; height: 100% }
9          </style>
10     </head>
11     <body>
12         <script src="js/three.js"></script>
13         <script src="/socket.io/socket.io.js"></script>
14         <script>
15             var socket = io();
16         </script>
17     </body>
18 </html>
```

Quellcode 2.1: HTML - Erster Prototyp der Einstiegseite

```
13  // Express configuration
14  app.set("port", process.env.PORT || 3000);
15  app.use(compression());
16  app.use(logger("dev"));
17  app.use(errorhandler());
18  app.use(express.static(path.join(__dirname, "public"), { maxAge: 31557600000 }));
19
20  // basic route
21  app.get("/", (req, res) => {
22    res.sendFile(path.join(__dirname + "/index.html"));
23  });
24
25
26  // start server
27  export const server = app.listen(app.get("port"), () => {
28    console.log("  App is running at http://localhost:%d in %s mode"), app.get("port"), app.get("env"));
29    console.log("  Press CTRL-C to stop\n");
30  });
31
32  // init socket io server
33  export const io = SocketIO(server);
34  io.serveClient(true);
35
36  io.on("connection", (socket) => {
37    console.log("Client connected on port " + app.get("port"));
38    socket.on("disconnect", () => {
39      console.log("Client disconnected.");
40    });
41  });
```

Abb. 2.15.: Der anfängliche Webserver

Die Abbildung 2.1 zeigt in in Zeile 15 dabei, wie der Aufruf der Socket.IO-Bibliothek erfolgt. Der Aufruf erzeugt automatisch eine Verbindung zum Webserver und stellt einen Socket bereit. In Zeile 13 und 14 werden die externen Bibliotheken inkludiert.

Der Server aus Abbildung 2.15 erzeugt zunächst eine Konfiguration des Express-Frameworks. Anschließend wird ein Http-Server erzeugt. Der Http-Server wird dann konfiguriert um auf

Anfragen von Clients auf den Wurzelpfad der Webadresse mit der Seite "Index.html" zu antworten. Anschließend wird der Socket-Server initialisiert auf Basis des Http-Servers. Die Ereignisse zur Verbindung und dem Beenden von Clients werden vorab nur mit Log-Einträgen gefüllt.

Entwurf eines Layouts für den Client

Als nächstes wurde das Layout und Design für den Client fertiggestellt. Dabei wurde sich an Vorlagen mit dem Framework Bootstrap orientiert. Es bietet den Vorteil, dass das Design direkt "responsive" ist, sich also an die Größe des Bildschirms (Display) und seiner Auflösung anpasst. Abbildung 2.16 zeigt die generelle Struktur der Webseite.
Die seitlichen Spalten, links und rechts, sind dabei für Nachrichten zu dem bzw. über den jeweiligen Spieler da. Die "Navbar" wurde zu einer Titelzeile, da es bei diesem Spiel nur eine statische Webseite gibt und daher keine Navigation benötigt wird. Der Footer wurde ebenfalls zu Gunsten von mehr Spielfläche im Zentrum entfernt, kann aber für Hinweise und ein Impressum wieder eingefügt werden. Wie bereits erwähnt stellt der mittlere Teil die Spielfläche dar. Die Spielfläche besteht dabei lediglich aus dem Canvas-Element.
Das Framework benutzt dabei ein Design, dass auf einem Raster aufbaut. Die Angaben in den Spalten stellen daher die Raster-Einheiten dar. Das Raster wird je nach Display-Größe dann so umsortiert, dass eine möglichst gute Erfahrung für den Nutzer entsteht. Auf kleinen Bildschirmen werden die seitlichen Spalten über bzw. unter der Spielfläche angeordnet, sodass die Spielfläche die gesamte Breite des Bildschirms einnimmt.
Es wurden außerdem noch folgende weitere Elemente aus dem Bootstrap-Framework direkt eingesetzt:

- Modale Dialoge, um Einstellung machen zu lassen bzw. über Sieg und Niederlage zu informieren
- Panels und Listen für die Nachrichten an Spieler
- Buttons für den direkten Test von Funktionalitäten (diese wurden im Anschluss natürlich entfernt)
- Ein "Jumbotron" – also eine große Box, mit farblicher Abhebung

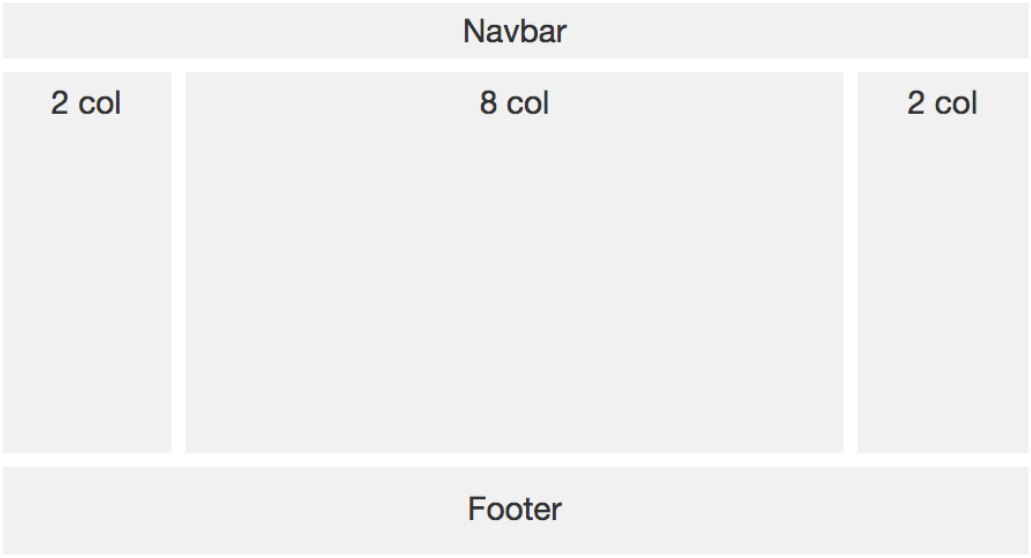

Abb. 2.16.: Die Layout-Vorlage mit Bootstrap[44]

Eine Parallelisierung von Spielen, wie in Abschnitt 2.3.2 bereits erklärt, ist nicht vorgesehen. Das erfordert allerdings einen Umgang mit Nutzern, die außer den beiden Spielenden die Webseite aufrufen. Auch hier kommen das Layout und Design der Webseite zum Einsatz.
Clients, die sich nicht erfolgreich in ein Spiel verbinden können, weil dieses belegt ist, werden auf eine Webseite weitergeleitet, die dem Nutzer diese Information mitteilt. Die Abbildung 2.17 zeigt die fertige Webseite für dieses Vorgehen.
Es wird außerdem eine neue Anforderung der Webseite gestartet, sobald ein aktiver Spieler die Verbindung schließt. Die Mitteilung über einen freien Platz, also die Aufforderung zum erneuten Anfordern der Webseite erteilt dabei der Webserver dem Client über ein Event der Socket-Kommunikation. Das weitere Vorgehen zum Verbindungsmanagement wird an dieser Stelle noch nicht betrachtet.

[44] Quelle: https://www.w3schools.com/bootstrap/bootstrap_templates.asp

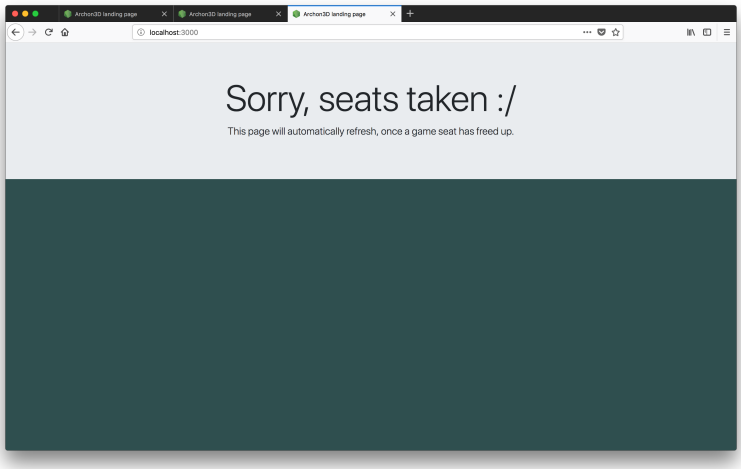

Abb. 2.17.: Die Webseite für abgewiesene Nutzer

Umsetzung des Layouts

Nun wird die Vorlage für das Layout der Webseite mittels Bootstrap soweit angepasst und mit Inhalten, wie dem Canvas gefüllt, dass ein erster Entwurf der fertigen Webseite entsteht. Die Abbildung 2.18 zeigt die Umsetzung, in die bereits einige Elemente des Canvas und three.js integriert wurden.

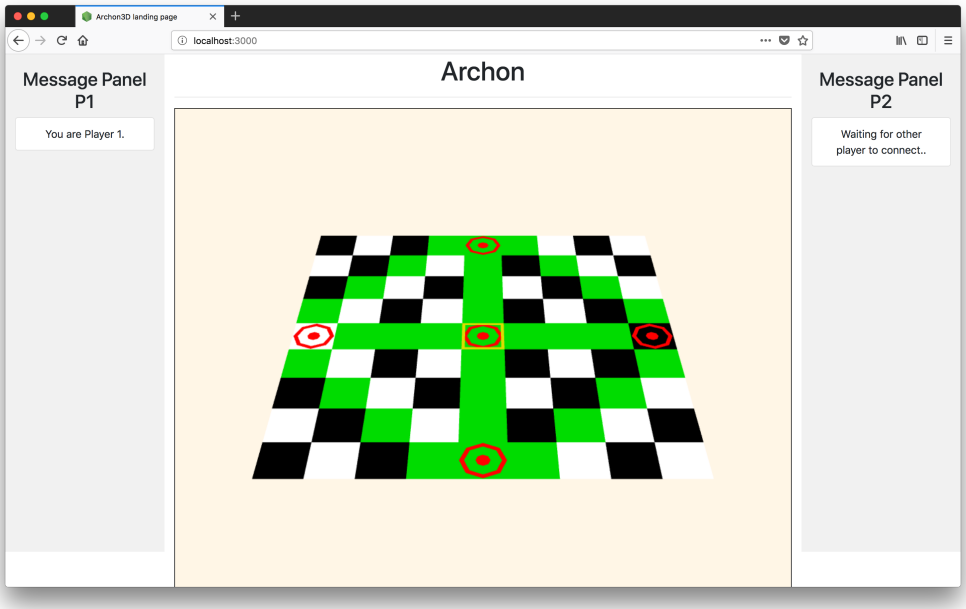

Abb. 2.18.: Die Webseite für aktive Spieler

Klar zu sehen sind in Abbildung 2.18 die einzelnen Layout-Bereiche der Vorlage aus Abbildung 2.16. Die Abbildung zeigt auch einen Entwurf des Spielbretts, der für frühe, sichtbare Fortschritte und einen sicheren Umgang mit der Grafik-Bibliothek three.js implementiert wurde.

Entwicklung des Servers

Da nun Client-Webseiten existieren, sind die entsprechenden Mechanismen für Kommunikation, Events, Verbindungsmanagement und das Datenmodell im Server zu implementieren. Der folgenden Abschnitt zeigt diese Aspekte in Ablaufdiagrammen, die den Code widerspiegeln.

Die Applikation startet damit ein Server-Objekt zu erzeugen. Eine erste Implementierung dessen ist bereits in Abbildung 2.15 zu sehen. Dort fehlen bisher die Erzeugung des Datenmodells, sowie ein erweitertes Verbindungsmanagement.

2.4. Resultate

Hier soll dann ein Screenshot des Ergebnisses rein und erläutert werden, dass als nächste der Endstand mit seiner Architektur gezeigt wird und anschließend die Erfüllung aller Anforderungen sichergestellt wird. Als letztes (falls genug Zeit!) werden die (hoffentlich) programmierten Unit-Tests erwähnt, und deren Ergebnisse dargestellt.

2.4.1. Überprüfung der Software mit Unit-Tests

Bei der Erstellung von Unittests[45] wurde sich auf Server-Komponenten beschränkt. Es wurden Tests für die korrekte Erzeugung und das Hochlaufen der Server-Komponente geschrieben. Auch wurden Tests für die Initialisierung der Socket-Kommunikation geschrieben. Die Abbildung 2.19 zeigt einen entsprechenden Testfall.

```
test("functional Test socket-server", () => {
    server.ioServer.on("connection", (socket) => {
        expect(socket).toBeDefined();
    });
    const httpServerAddr = server.httpServer.address();
    const socket = io(`http://[${httpServerAddr.address}]:${httpServerAddr.port}`, {
        reconnectionDelay : 0,
        forceNew: true,
        transports: ["websocket"]
    });
});
```

Abb. 2.19.: Ein Testfall zur Kommunikation

[45] Unittests – deutsch: Modultests werden benutzt um einzelne Komponenten auf Fehlerfreiheit zu überprüfen

Desweiteren wurde durch Unittests sichergestellt, dass der Initialzustand der Datenmodells richtig erzeugt wird. Dadurch wurden wesentliche Fehler im Grundbetrieb ausgeschlossen und eine möglichst fehlerfreie Server-Komponente erzeugt. Durch die erfolgreichen Unittests des Datenmodells konnten Initialisierungsfehler ausgeschlossen werden. Dies ist besonders wichtig, da auf Basis des initialen Datenmodells auch der Client aufgebaut wird.

Die Benutzung von Unittests hat stark zur Erfüllung einiger Anforderungen beigetragen, indem die Fehlerfreiheit essentieller Komponenten festgestellt wurde und nötige Änderungen frühzeitig erkannt wurden.

2.4.2. Erfüllung der Anforderungen

2.5. Zusammenfassung

3. Fazit und Ausblick

Die Ergebnisse wurden bisher nur dargestellt und erklärt und nicht bewertet und analysiert, dass soll hier geschehen unter "erreichte Ziele". Das Fazit soll Punkte zu Anwendbarkeit der Technologien, Entwicklung/Nachbau eines Spieleklassikers und Problempunkte, aber auch positives zu Support und Inbetriebnahme enthalten. Der Ausblick soll kommende Technologien und weitere Entwicklungspunkte für das Spiel beleuchten.

3.1. Fazit

3.1.1. erreichte Ziele

3.2. Ausblick

Die Entwicklung dieses Ablegers von Archon ist mit dieser Arbeit keinesfalls abgeschlossen. Es gibt viele offene Features und Themen für weitere Abschlussarbeiten, die im Folgenden kurz angerissen werden.

Generelle Weiterentwicklung

Das Spiel an sich ist noch nicht komplett reproduziert worden. Es können mehr Sprites implementiert und eine bessere visuelle Erfahrung entwickelt werden.
Weiterhin kann die HTML-Audio-API benutzt werden um Soundeffekte einzufügen.
Die Kampf-Komponente ist noch nicht wirklich fertiggestellt und auch die Zauber sind noch nicht fertiggestellt. Daher ist hier noch Bedarf für weitere Entwicklungsarbeiten.

KI-Modus / Demo-Modus

Diese Realisierung hat bisher nur menschliche Spieler realisiert, jedoch erlaubt die Architektur genauso den Ersatz der View-Komponente durch einen Computer-Spieler. Eine spannende Arbeit kann daher die Entwicklung einer künstlichen Intelligenz (KI) für den Demo-Modus bzw. den Modus Spieler-vs-Computer sein.

Score-Boards

Es können eine Datenbank und ein Authentifizierungsservice angebunden werden und damit Spieler registriert werden. Das würde die Implementierung einer Rangliste (Score-Board) oder eines Systems von Spielerfolgen (Achievements) ermöglichen.

Verschiedenes

Das Spiel ist gemacht für die Benutzung mit einem GamePad[46]. Die HTML-GamePad-API ist allerdings noch in einem sehr frühen Stadium der Standardisierung, sodass zunächst eine Evaluierung stattfinden muss, ob sie ausreichend benutzbar ist, oder ob andere Bibliotheken diese Funktionalität hergeben.
Außerdem kann der Inputcontroller um die Touch-Events für die Benutzung auf mobilen Endgeräten erweitert werden.

Desweiteren ist bisher keine Verschlüsselung des Datenverkehrs und auch kein Zugriff auf die Webseite via HTTPS umgesetzt.

Als Abschluss ist eine Parallelisierung der Architektur ein offener Punkt. Dies erlaubt es mehrere Spiele gleichzeitig auf einem Server spielen zu lassen, dafür müssten auch Leistungs- und Lasttests durchgeführt werden, um leistungsgemäß viele Spiele laufen zu lassen.

[46] Ein GamePad ist ein Spielecontroller mit Joystick und diverses generellen Tasten.

A. Anhang - Informationen zu Tools

A.1. Erweiterungen von Visual Studio Code

Die installierten Erweiterungen des Editors Visual Studio Code werden folgend alphabetisch aufgelistet:

- alefragnani.project-manager
- austin.code-gnu-global
- christian-kohler.path-intellisense
- dbaeumer.vscode-eslint
- donjayamanne.githistory
- eamodio.gitlens
- eg2.tslint
- formulahendry.code-runner
- James-Yu.latex-workshop
- mitaki28.vscode-clang
- ms-vscode.azure-account
- ms-vscode.cpptools
- ms-vscode.csharp
- msjsdiag.debugger-for-chrome
- redhat.java
- robertohuertasm.vscode-icons
- rokoroku.vscode-theme-darcula
- streetsidesoftware.code-spell-checker
- streetsidesoftware.code-spell-checker-german
- WallabyJs.wallaby-vscode
- zhuangtongfa.Material-theme

Literaturverzeichnis

[1] Bowden Kelly: *TypeScript Node Starter*. https://github.com/Microsoft/TypeScript-Node-Starter. Version: 0.1.0, Abruf: 31.03.2018

[2] Contributors, Mozilla: *MDN-Web-Dokumentation*. https://developer.mozilla.org/de/, Abruf: 31.03.2018

[3] expressjs.com contributors: *express.js*. https://expressjs.com/. Version: 4.16.2, Abruf: 31.03.2018

[4] https://github.com/mrdoob: *three*. https://threejs.org/. Version: 0.91.0, Abruf: 31.03.2018

[5] https://github.com/socketio/socket.io/graphs/contributors: *Socket.IO*. https://socket.io/. Version: 2.1.0, Abruf: 31.03.2018

[6] IDC: *Absatz von Tablets, PCs und Smartphones weltweit von 2010 bis 2017 und Prognose für 2022 (in Millionen Stück)*. https://de.statista.com/statistik/daten/studie/256337/umfrage/prognose-zum-weltweiten-absatz-von-tablets-pcs-und-smartphones/, Abruf: 31.03.2018

[7] Medarch: *The Secrets Of Archon*. http://www.vintagecomputing.com/index.php/archives/44. Version: 1.0, Abruf: 31.03.2018

[8] Microsoft: *TypeScript - JavaScript that scales*. https://www.typescriptlang.org/index.html. Version: 2.8, Abruf: 31.03.2018

[9] Microsoft: *Visual Studio Code*. https://code.visualstudio.com/, Abruf: 31.03.2018

[10] Node.js Foundation: *node.js*. https://nodejs.org/en/. Version: 9.4.0, Abruf: 31.03.2018

[11] Nystrom, Bob: *Game Programming Patterns*. 1. Auflage 2014. Genever Benning, 2014 http://gameprogrammingpatterns.com/. – ISBN 0990582906

[12] The jQuery Foundation: *jQuery - Webportal*. http://jquery.com/, Abruf: 31.03.2018

[13] https://developer.mozilla.org/en-US/docs/Games/Anatomy$history: *Anatomy of a video game*. https://developer.mozilla.org/en-US/docs/Games/Anatomy. – is licensed under https://creativecommons.org/licenses/by-sa/2.5/

[14] https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/Basic_theory$history: *Explaining basic 3D theory*. https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/Basic_theory. – is licensed under https://creativecommons.org/licenses/by-sa/2.5/

[15] W3Schools, Refsnes Data: *W3Schools Online Web Tutorials*. http://www.w3schools.com/. Version: 2015, Abruf: 11.02.2015

[16] Wikipedia: *Archon (Computerspiel)*. https://de.wikipedia.org/wiki/Archon_(Computerspiel). Version: Dezember 2014, Abruf: 31.03.2018

[17] Wikipedia: *Commodore 64*. https://de.wikipedia.org/wiki/Commodore_64. Version: Juli 2018, Abruf: 09.07.2018

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

Abb.	Abbildung
API	Programmschnittstelle nach aussen (engl. für Application Programming Interface)
CSS	gestufte Gestaltungsbögen, legt die Darstellung des HTML Quellcodes im Browser fest (engl. für Cascading Style Sheets)
DB	Datenbank
DOM	Dokumentstruktur der Webseite (engl. für Document Object Model)
GIF	Grafikaustausch Format, Animationsfähig (engl. für Graphics Interchange Format)
GUI	Grafisches Benutzer Interface (engl. für Graphical User Interface)
HTML	Hypertext Auszeichnungssprache, HTML-Dateien sind die Grundlage des World Wide Web und werden von einem Webbrowser dargestellt (engl. für Hypertext Markup Language)
HTTP	Hypertext-Übertragungsprotokoll, Protokoll zur Übertragung von Daten über ein Netzwerk (engl. für Hypertext Transfer Protocol)
HTTPS	sicheres Hypertext-Übertragungsprotokoll s. a. HTTP (engl. für Hypertext Transfer Protocol Secure)
IDE	integrierte Entwicklungsumgebung (engl. für Integrated Development Environment)
Interface	Schnittstelle
IP	Internet Protocol
JavaScript	Skriptsprache, ursprünglich für dynamisches HTML in Webbrowsern entwickelt
JPEG	komprimierte Grafikdatei, auch JPG (engl. für Joint Photographic Expert Group)
JSON	kompaktes Datenformat zum Datenaustausch mit z. B. Webservern (engl. für JavaScript Object Notation)
Mockup	Attrappe oder auch rudimentärer Prototyp (auch Maquette)
MSDN	Das Microsoft Entwickler Netzwerk (engl. für MicroSoft Developer Network)
o.V.	ohne Verfasser (bei Literaturverweisen)
Parser	engl. to parse - „analysieren“ bzw. lateinisch pars - „Teil“ im Deutschen gelegentlich auch Zerteiler, Analysiert die Semantik des Scripts um daraufhin Aktionen durchzuführen

PNG	Grafikaustausch Format (engl. für Portable Network Graphics)
SCL	Strukturierter Text (engl. für Structured Control Language)
SDK	s. a. IDE (engl für Software Development Kit)
SVG	skalierbare Vektorgrafik (engl. für Scalable Vector Graphics)
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
W3C	Organisation zur Standardisierung von Webtechnologien (engl. für World Wide Web Consortium)
Webbrowser	auch kurz Browser (engl. to browse) steht für durchstöbern, abgrassen, durchsuchen - Software zum Darstellen von Daten, hauptsächlich Webseiten und deren Inhalt, können zu diesem Zweck mit Webservern kommunizieren
Webseite	s. a. HTML-Datei
Webserver	Ein Webserver speichert Webseiten und stellt diese zur Verfügung. Der Webserver ist eine Software, die Dokumente mit Hilfe standardisierter Übertragungsprotokolle (HTTP, HTTPS) an einen Webbrowser überträgt.
WWW	Internet (engl. für World Wide Web)

Stichwortverzeichnis

Das Verzeichnis ist in Haupt- und Unterbegriffe gegliedert. Ist ein Stichwort nicht unter den Hauptbegriffen gelistet, so ist es womöglich als Untereintrag zu finden.