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

Inhaltsverzeichnis

Inhaltsverzeichnis I

Vorwort III

Erklärung V

1 Einleitung 1
- 1.1 Hinführung . . . 1
- 1.2 Aktueller Forschungsstand . . . 3
- 1.3 Motivation / Problemstellung . . . 4
- 1.4 Zentrale Begriffe . . . 4
  - 1.4.1 Was versteht man unter Webtechnologien? . . . 4
  - 1.4.2 Was versteht man unter 3D-Technologien? . . . 5
- 1.5 Ziel dieser Arbeit . . . 5
- 1.6 Aufbau . . . 6

2 Hauptteil 7
- 2.1 Einleitung . . . 7
- 2.2 Analyse . . . 7
  - 2.2.1 Analyse des klassischen Spiels . . . 7
  - 2.2.2 Anforderungsanalyse an neue Realisierung . . . 12
- 2.3 Implementierung . . . 14
  - 2.3.1 benötigte Technologien und Frameworks . . . 14
  - 2.3.2 Hilfsmittel und Vereinfachungen . . . 15
  - 2.3.3 Architektur . . . 16
  - 2.3.4 Schritte der Implementierung . . . 23
- 2.4 Resultate . . . 24
  - 2.4.1 fertige Architektur . . . 24
  - 2.4.2 Erfüllung der Anforderungen . . . 24
  - 2.4.3 Überprüfung der Software mit Unit-Tests . . . 24
- 2.5 Fazit . . . 24

3 Fazit und Ausblick 25
- 3.1 Fazit . . . 25
  - 3.1.1 erreichte Ziele . . . 25
- 3.2 Ausblick . . . 25

Verzeichnisse **VII**

Literaturverzeichnis VII
Abbildungsverzeichnis IX
Tabellenverzeichnis XI
Quellcodeverzeichnis XIII
Abkürzungsverzeichnis XV
Stichwortverzeichnis XIX

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 17. 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 große Vielfalt an unterschiedlichsten Applikationen, die nahezu überall auf der Welt benutzbar sind, seien es Business- oder Unterhaltungs-Applikationen, Informationsdienste oder *Spiele*.
Auch Spiele auf mobilen Plattformen, wie Smartphones, besitzen immer mehr Grafikdetails und Leistungsanforderungen. Dem gegenüber steht ein immer noch wachsender Markt für Spiele-Klassiker unterschiedlichster Spielekonsolen, die dem Drang nach mehr Grafikdetails und Leistungshunger aus dem Weg gehen und durch Charme mit Pixeln und Nostalgie punkten. Spiele-Klassiker wie PacMan, Tetris, Super Mario und uvm. sind auch heute noch auf vielen, teils portablen, Konsolen zu finden. Es gibt weiterhin Absatz auch für alte Plattformen wie die Atari, oder den Commodore 64. Es werden sogar Neuauflagen herausgebracht von Spielen, sowie Konsolen.[1] Außerdem finden sich, durch den Fortschritt bei der Virtualisierung von Computersystemen bedingt, immer mehr Emulatoren für alte Spielekonsolen inklusive vieler Spiele im Internet. Spiele wurden für die Pioniere des Konsolen- und Heimcomputermarktes Commodore 64 und Atari zuhauf entwickelt.[2] Die beliebtesten Spiele waren dabei unter anderem die Folgenden:[3].
Mit dabei ist stets auch das Spiel, um das sich die Arbeit dreht: *Archon*.

Spielen im Allgemeinen sind kaum noch Grenzen gesetzt. Die Spieleindustrie hat 2018 das erste Mal den Umsatz der Film- und Unterhaltungsindustrie überstiegen, sodass auch für jegliche Ideen genug Absatz und Kapital gefunden wird.
Plattformen für Spiele gibt es ebenfalls genügend für jeden Geschmack, ob Konsole, Computer und heutzutage auch Smartphone, Browser und sogar der Fernseher. Die klaren Vorteile des Web als Plattform, auch für Spiele, sind dabei die Offenheit, der einfache Zugang und die immer weiter steigende Unterstützung neuer Technologien.
So sind heutzutage hardwarebeschleunigte 3D-Visualisierung und Echtzeitkommunikation im Webbrowser Stand der Technik. In dieser Arbeit soll die Verschmelzung von State of the Art Webtechnologien mit einem Spieleklassiker erfolgen.
"Archon" genoss für Atari und C64 großen Erfolg, ähnlich vieler anderer Spiele für die Konsolen der 80er. Das Spiel dient daher als sehr gutes Exempel für die Möglichkeiten des Webs als

Anmerkun
Recherche
Erfolg von
chon beleg

[1] Beleg fehlt noch für Artikel zur Neuauflage!
[2] Es wurden etwa 17.000 kommerzielle Titel entwickelt.[17]
[3] Rankings nach Seiten X und Y
[4] Quelle: C64-Wiki, https://www.c64-wiki.com/wiki/Archon

Abb. 1.1: Archon - Cover[4]

Spieleplattform, aber auch der Möglichkeiten und Freiheiten des Webs für jede andere Art von Applikation.

- Verkaufte Konsolen
- Produzierte Spiele und Verkäufe
- Neuauflagen von Spielen und Konsolen
- Neuauflagen von Archon
- Virtualisierung und Emulatoren um alte Spiele zum Anfassen zu bekommen
- Aber, nur auf richtigen PCs zu finden und meist nicht wirklich legal
- Web als freie, riesige Plattform mit großem Zugriffsbereich
- Web APIs, die in den letzten Jahren erschienen sind (kurze Recherche zu Arbeiten des W3C!)
- Schiere Vielfalt an JS-Frameworks zu Web APIs und Komfort bei der Entwicklung (npm packages um Zahlen zu bekommen)
- Beispiele zu 3D-Möglichkeiten im Web, Stichwort: THREE.JS-Demos

1.2 Aktueller Forschungsstand

Hier sollen dann Extrema zum einen in Richtung 3D-Entwicklung, in Richtung Web-Entwicklung und aber auch bei der Spiele-Entwicklung kurz herausgearbeitet werden, um dem Leser einen kurzen Einblick in aktuelle Möglichkeiten zu geben, und zu zeigen, dass die Arbeit "State of the Art" ist.

- Das Web

  Viele APIs als Standard freigegeben

  Verschlüsselung immer wichtigeres Thema

  Mobile & Offline First

- Grafik Rendern im Browser

  Flash

  SVG, pures HTML + CSS

  Canvas

  WebGL für aufwändige 3D-Grafik (noch kein vollständiger Standard!)

- Games

  komplexeste Spiele sind MMO*

  Serverlose Strukturen/Nutzung von Cloud-Services

  Skalierung

  Budget

Dieses Kapitel behandelt den Einstieg in aktuelle Themen der Spieleentwicklung und des Internet. Zunächst wird dabei auf aktuell stark behandelte Themen des 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 acht Kernthemen im Rahmen Web Design und Applikationen. 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 die Audio & Video API, sowie Graphics.

Sogenannte Spiele-Engines, also Bibliotheken die sich um die meisten, generellen Problemstellungen eines Spiels und seiner Entwicklung kümmern, gibt es daher auch für browserbasierte Spiele allerlei.

1.3 Motivation / Problemstellung

Diese Sektion soll vom vorherigen Stand der Forschung und Möglichkeiten auf Anwendbarkeit überleiten und zeigen, dass die evtl. surreal erscheinende Vermischung von Genres, Plattformen und Technologien anwendbar ist, und das sogar mit beschränktem Aufwand. Vereinbarkeit von aktuellen Technologien mit Spiele-Klassikern und Spiele-Klassiker als Anwendungsbeispiele dafür.
Bisher sind oftmals nur Demonstrationen, kurze Starter-Kits bzw. Einstiegspunkte einzelner Web-Technologien und Frameworks im Web zu finden.
Als Entwickler bietet es sich jedoch an auf gleiche, oder ähnliche gelöste Problemstellungen zurückgreifen zu können. Anhand der Webapplikation sollen die Möglichkeiten des Webs dargestellt werden und 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 ausführbar. Das Web leistet an dieser Stelle die perfekte Abhilfe: Es gibt unzählige Geräte, die heutzutage einen Webbrowser installiert haben, somit hat jeder Zugriff auf diesen Ableger, der ein webfähiges Gerät besitzt.

1.4 Zentrale Begriffe

1.4.1 Was versteht man unter Webtechnologien?

Webtechnologien betiteln meist die Sammlung aller Aspekte zum Erstellen, Warten und Entwickeln einer Webanwendung.
Webanwendungen selbst bestehen aus einem Client-Server-Modell. Typische Bestandteile des Clients sind:

- HTML zur Beschreibung des Inhalts

- CSS zur Beschreibung des Aussehens
- JavaScript zur Dynamisierung des Clients

Ein Webserver gibt auf HTTP-Anfragen die entsprechenden Inhalte und Medien an den Client heraus, welcher durch einen Webbrowser angezeigt wird. Außerdem können hier Daten des Clients verarbeitet, gespeichert und verteilt werden werden.

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.
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, 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 hier 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:

- Es 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, sodass eine Liste an Funktionalitäten entsteht, die von den 3D- und Webtechnologien erfüllt werden muss. Anschließend wird der Aufbau der Entwicklungsumgebung, sowie benutzte Hilftmittel und getroffene Vereinfachungen beschrieben. Alle Anforderungen werden dann in eine Beschreibung zur Umsetzung und in eine entsprechende Architektur ausgearbeitet. Die Implementierung wird in Ausschnitten gezeigt, die die einzelnen Schritte und die Vorgehensweise bei der Umsetzung von Architektur und Programmierung aufzeigen sollen. Den Abschluss bildet eine Überprüfung auf die Erfüllung der Anforderungen mit kurzem Einblick auf den abschließenden Stand und eine Reflexion des Ergebnisses mit Ausblick auf Folgearbeiten an der Webapplikation.

2 Hauptteil

2.1 Einleitung

2.2 Analyse

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]

In diesem Abschnitt soll die Vorgehensweise bei der Analyse kurz aufgezeigt werden und der Inhalt der einzelnen Teile klar gemacht werden. Also, zum einen, dass hier eben eine Anforderungsanalyse stattfindet, und zum Anderen, dass diese Analyse zweigeteilt ist.

2.2.1 Analyse des klassischen Spiels

Hier werden alle Aspekte des klassischen Spiels herausgearbeitet, gegliedert nach Teilen: Generelle Regeln, das Board, die Figuren, Abläufe und Wirkungen, mögliche Einstellungen
Stichwort: Komponentendiagramm!
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

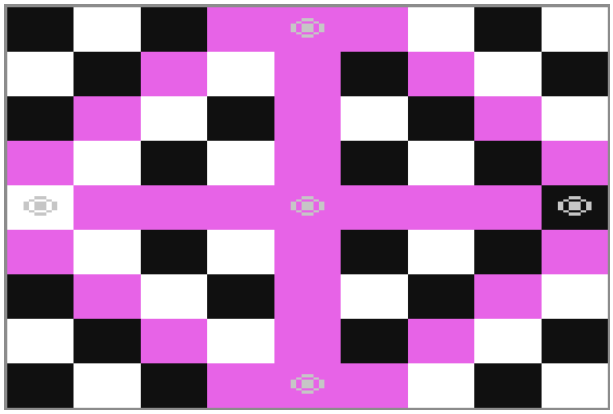

Abb. 2.1: Das Spielbrett [5]

Das Spielbrett besteht aus 9x9 Feldern von drei Typen:

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

- 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. [7] Weiterhin gibt es die

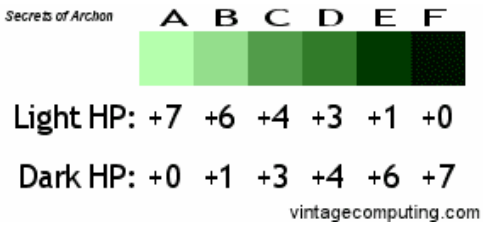

Abb. 2.2: Lebenspunkte-Bonus anhand der Feldfarben[6]

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

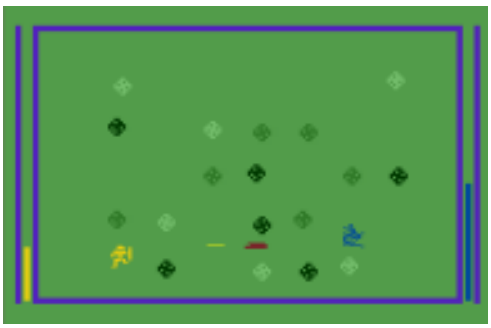

Abb. 2.3: Das Kampfareal

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

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 Eigenschaften bereits in einer anderen Zusammenfassung zu finden sind und an diesem Punkt für einen generellen Einblick in das Spiel nicht relevant sind.[8]

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

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

werden können. Die vier Elementare sind nach den vier Elementen Feuer, Wasser, Erde und Luft eingeteilt und besitzen, ähnlich zu normalen Figuren, unterschiedliche Sprites[9] 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.

- **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.

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

Zusammenfassung für die Realisierung

ew fällig

Alle vorher genannten Aspekte des Spiels bilden den Kern, der das Spiel ausmacht: Ein Strategiespiel auf einem 9x9 Schachbrett mit Action-Elementen in einem Kampfareal und gut gewichteten Spielfiguren.
Um nun also 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 machen. 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 sollten zumindest einige Typen 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 einen eigenen Charme 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 Archon und jede einzelne Partie besonders. Der Stil der Zauber ist auch hier nicht wichtig, sondern ausschließlich ihre Funktion.

2.2.2 Anforderungsanalyse an neue Realisierung

Hier wird analysiert, welche Aspekte das Spiel von der Seite der technologisch gestellten Anforderungen erfüllen muss. Also: Es muss eine 3D- Engine geben, die irgendwie im Browser renderbar ist.
Es muss möglich sein, alle Spiel-Mechaniken mittels Server-Client-Kommunikation umzusetzen, etc.
Stichwort: Komponentendiagramm!
Der Titel, sowie die Einleitung haben schon erläutert, dass bei dieser Realisierung zwei essentielle Technologien zum Einsatz kommen sollen:

ist noch
t nötig!

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 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[10] 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[11]-Format übertragen werden. Durch die weite Verbreitung des JSON-Formats ist die weitere Verarbeitung der Objekte stark vereinfacht und standardisiert. Da Websockets, inklusive möglicher Frameworks[12], in JavaScript implementiert werden ist das Dreieck der Webentwicklung die Wahl für Stil, Inhalt und Dynamisierung des Spiels.

Das gefällt noch gar n

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[13] 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 ausgeführt werden. Das erlaubt jede Kommunikation – Server-Client und Spieler-Client über eine Schnittstelle abzufertigen. Auf dem Client können Inhalte durch HTML[14]-Elemente repräsentiert und der Stil mittels CSS[15]-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.

[10] Overhead – In der IT-Umgebung gängiger Begriff für Meta-Daten oder Programmcode der zusätzlich zur eigentlich Funktion ausgeführt, oder versendet werden muss
[11] JSON – JavaScript Object Notation
[12] Framework – softwaretechnischer Rahmen, der Funktionen und Programmstukturen bereitstellt
[13] Peer-to-Peer – Kommunikation zwischen gleichen End-Teilnehmern ohne einen Server als Medium
[14] HTML – Hypertext Markup Language ist eine Sprache um Dokumente im Webbrowser in ihrem Inhalt zu strukturieren
[15] CSS – Cascading Style Sheets bilden die Regeln zum Aussehen von Webdokumenten

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 müssen mittels JavaScript programmiert werden. 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[16] 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 Showstopper, oder Einschränkungen für eine Realisierung von Archon mittels der oben genannten Technologien gibt folgt nun die Implementierung.

2.3 Implementierung

Hier soll dann aufgezeigt werden, welche Anforderungen vom Spiel durch welche Technologie umgesetzt werden und nötige Bedingungen/grundsätzliche Ausschlüsse aufgezeigt werden.
Hier, oder im nächsten Abschnitt muss klar gestellt werden, dass ein KI-Modus nicht implementiert wird!
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. 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

Hier wird dann von den benötigten Technologien die Festlegung auf eine bestimmte Implementierung getroffen, also Frameworks, Programmiersprachen, Toolsets etc. festgelegt.
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.

[16] Hardwarebeschleunigt – Rechenintensive Aufgaben werden vom Hauptprozessor eines Computers an dafür dedizierte Hardware delegiert, heute vermehrt Grafikkarten und grafikintensive Aufgaben

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.

Hier fehlt noch was

Die node.js-Umgebung erlaubt es 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 daher mit Dev-Dependencies und anderen Hilftmitteln und Vereinfachungen für die Implementierung.

2.3.2 Hilfsmittel und Vereinfachungen

Hier würden Dinge zu lesen sein, wie die Benutzung von TypeScript, oder Browser-Beschränkungen etc, da diese nicht relevant für den eigentlichen Entwicklungsvorgang sind, aber dennoch nützlich sind und eben Vereinfachungen darstellen.

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[17] bzw. einem Code-Editor mit Auto-Vervollständigung genutzt werden können.

2.3.3 Architektur

Dieser Abschnitt enthält eine klassische objektorientiere Analyse des Spiels, sowie der Anforderungen aus der technologischen Sichtweise. 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 grob die Beziehungen einzelner Elemente aufzeigt, ohne sich direkt auf Code-Ebene zu bewegen. Auch diese Analyse soll dem Muster folgen. Die Abbildung 2.4 stellt lediglich die im Kapitel 2.2.1 dargestellten

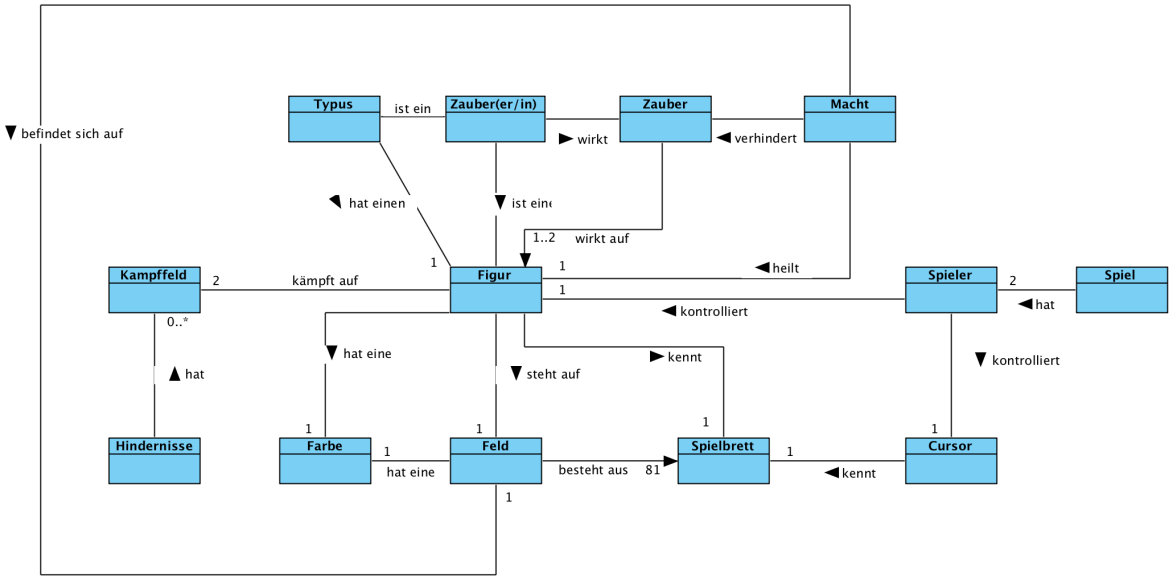

Abb. 2.4: Begriffsmodell

Inhalte grafisch dar, was aber 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.

[17] Integrated Development Environment

Am Ende dieses Kapitels werden die technologischen Abhängigkeiten aufgezeigt und mit in die Architektur integriert.

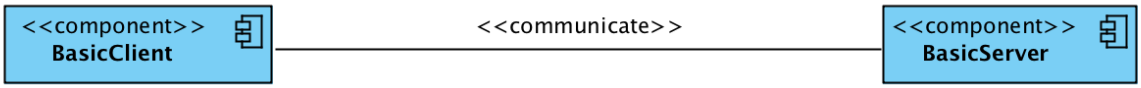

Abb. 2.5: Client-Server-Architektur

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. Aus den Kommunikationsanforderungen und der Aufteilung auf Client und Server ist dann ein Ansatz mit dem Model-View-Controller-Modell entstanden, um Abhängigkeiten zu reduzieren und Funktionalitäten zu kapseln. Das Modell 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

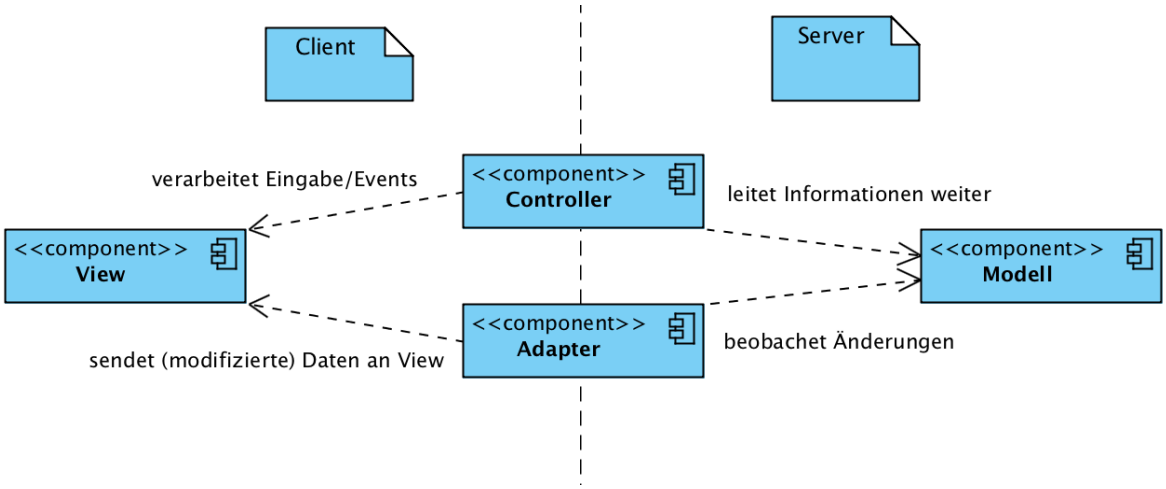

Abb. 2.6: Darstellung der MVC-Architektur mit Aufgaben

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. 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.

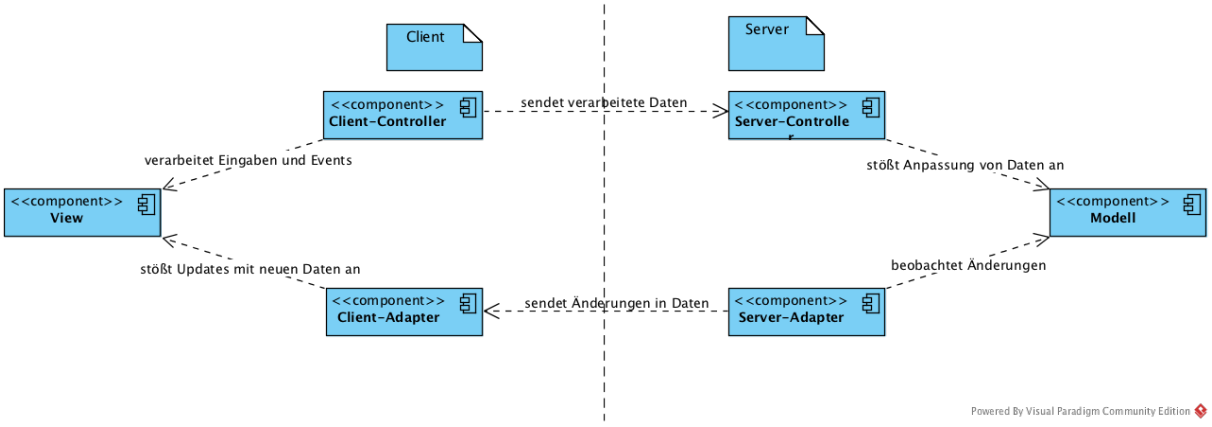

Abb. 2.7: Aufgeteilte Kommunikationsklassen

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.

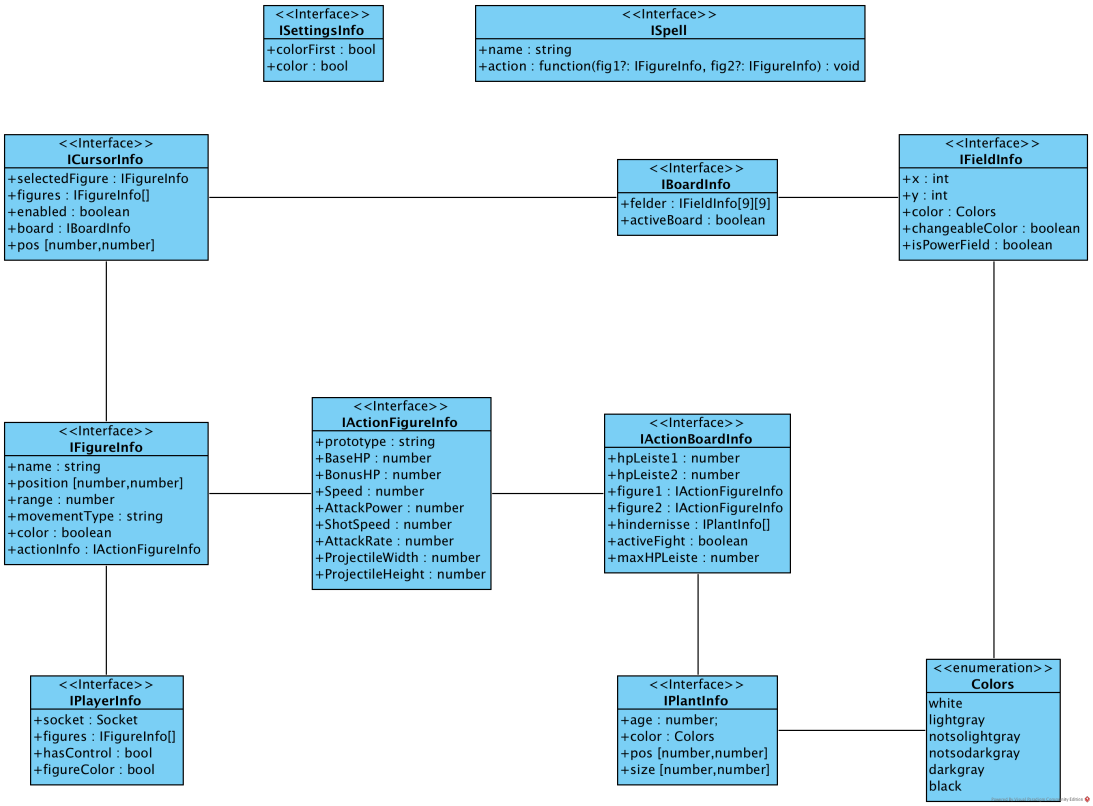

Abb. 2.8: Daten-Schnittstellen für Datenmodell

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 fehlen jetzt noch die passenden Klassen für Anzeige und Logik 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

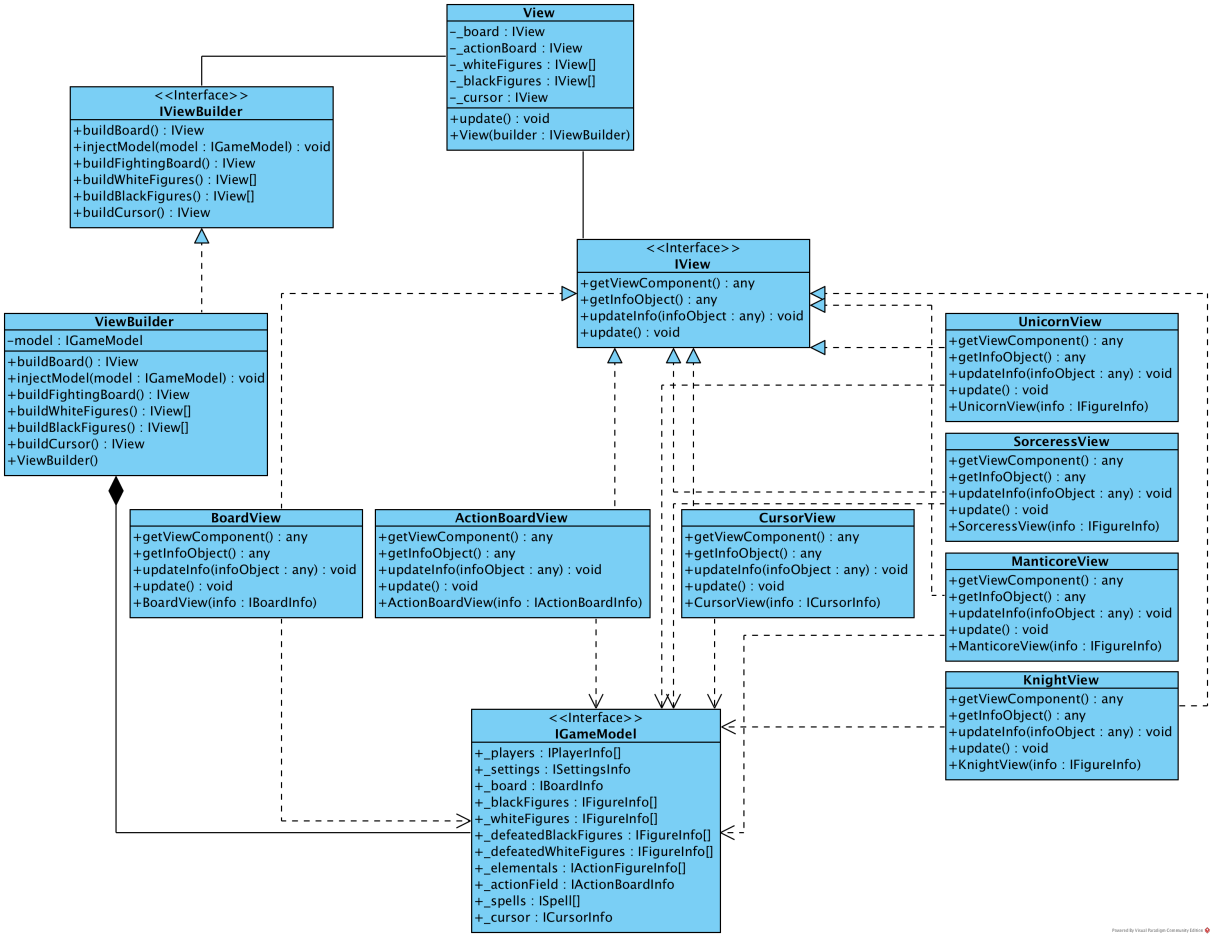

Abb. 2.10: leicht vereinfachtes Klassendiagramm der View

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

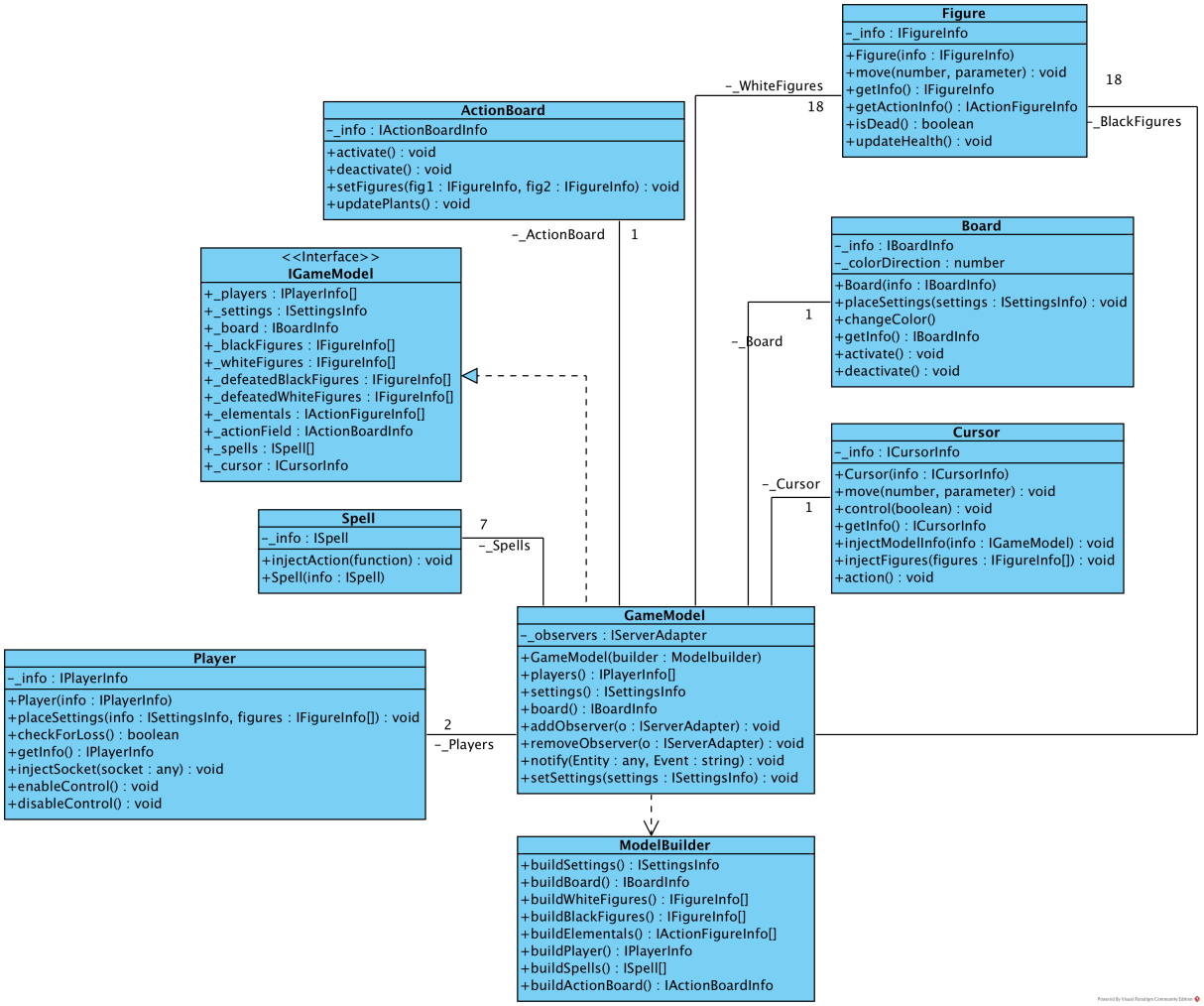

Abb. 2.11: Klassendiagramm des Modells

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 Implementierung 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 beschrieben wurden, bleibt nun lediglich die Ansicht der technologischen Abhängigkeiten zu zeigen.

Abhängigkeiten

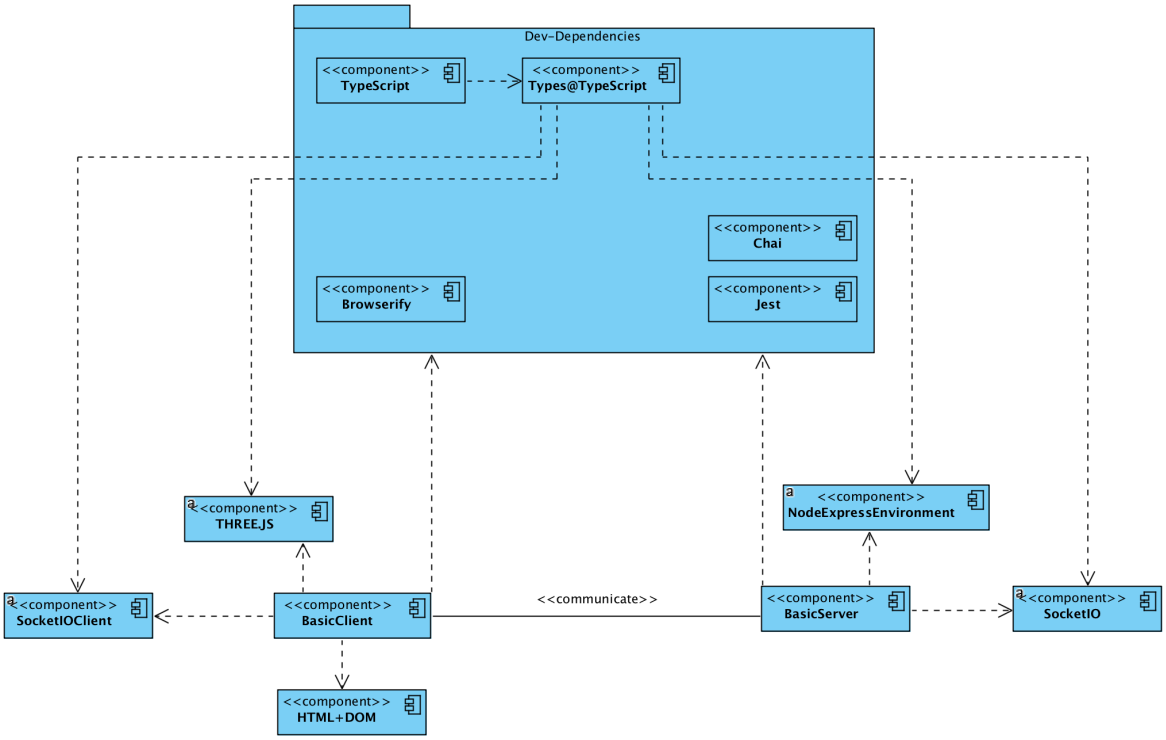

Abb. 2.12: Komponentendiagramm der Abhängigkeiten

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.[18] Anschließend wurde die

[18] So zum Beispiel für Socket.IO: https://socket.io/get-started/chat/

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. Generell ist ein solches 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 etc. ausgeführt werden sollen. Auch sind in dieser Datei die Abhängigkeiten und 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.

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 fertige Architektur

Die fertige Architektur kann und soll durchaus von der geplanten Abweichen und das schlussendliche Ergebnisse wird hier in Form von Diagrammen gezeigt, die dann einzeln erklärt werden.

2.4.2 Erfüllung der Anforderungen

Hier werden die Anforderungen aus dem Analyse-Teil aufgegriffen und mit der fertigen Anwendung und ihrer Architektur abgeglichen, also so was wie "die einzelnen Figuren und ihre Unterschiede, sind hier und hier da und da durch umgesetzt worden."

2.4.3 Überprüfung der Software mit Unit-Tests

Hier wird dann die breite der Unit-Tests gezeigt, deren Anzahl und die Beschränkungen, also Code-Abdeckung. Anschließend ein Ergebnis-Log von einem Lauf auf dem finalen Stand.

2.5 Fazit

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

Hier kann auch noch in der bisher fehlende KI-Modus in Aussicht gestellt werden.

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

1.1 Archon - Cover 2

2.1 Das Spielbrett 8
2.2 Lebenspunkte-Bonus 9
2.3 Kampfareal 9
2.4 Begriffsmodell 16
2.5 Client-Server-Architektur 17
2.6 MVC 17
2.7 ArchitekturSplit 18
2.8 InfoModel 19
2.9 IGameModel 20
2.10 View 21
2.11 Modell 22
2.12 Abhängigkeiten 23

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

Abb.	Abbildung
AJAX	ermöglicht asynchronen Datenaustausch mit z. B. Webservern (engl. für Asynchronous JavaScript and XML)
API	Programmschnittstelle nach aussen (engl. für Application Programming Interface)
ASCII	7-Bit Zeichencodierung (engl. für American Standard Code for Information Interchange)
AWL	Anweisungsliste, Assembler ähnlich
Code-Folding	logisch zusammengehörende Quelltextabschnitte werden in Abschnitte gruppiert, um diese einfach ein- bzw. auszublenden, erhöht die Lesbarkeit und Übersichtlichkeit
CPU	Elektronischer Rechner (engl. für Central Processing Unit)
CR	Wagenrücklauf (engl. Carriage Return)
CRC	zyklische Redundanzprüfung (engl. Cyclic Redundancy Check)
CSS	gestufte Gestaltungsbögen, legt die Darstellung des HTML Quellcodes im Browser fest (engl. für Cascading Style Sheets)
DB	Datenbaustein, Baustein zur Datenhaltung
DOM	Dokumentstruktur der Webseite (engl. für Document Object Model)
FB	Funktionsbaustein, wie FC nur mit Gedächtnis (Speicher in Form eines Datenbaustein
FC	Funktion
Field PG	Spezieller Laptop (Computer) für industrielle Umgebungen zum Programmieren einer Steuerung
FUP	Funktionsplan, Digitalen Logik Gattern ähnlich
GIF	Grafikaustausch Format, Animationsfähig (engl. für Graphics Interchange Format)
GUI	Grafisches Benutzer Interface (engl. für Graphical User Interface)
HAP	HTML Agility Pack, `C#`HTML Bibliothek zum verarbeiten von Webdokumenten
HMI	Mensch-Maschine Interface (engl. für Human Machine 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
IPC	Industrie PC - Computer für Industriellen Einsatz
Java-Applet	Hilfsprogramm oder Tool, was in eine Webseite integriert wird
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)
KOP	Kontaktplan, Schaltplan ähnlich
LF	Zeilenvorschub (engl. Line Feed)
LRC	Längsparitätsprüfung (engl. Longitudinal Redundancy Check)
Mockup	Attrappe oder auch rudimentärer Prototyp (auch Maquette)
MSDN	Das Microsoft Entwickler Netzwerk (engl. für MicroSoft Developer Network)
MWSL	Mini Web Server Language, Serverbasierende Scriptsprache
o.V.	ohne Verfasser (bei Literaturverweisen)
OB	Organisationsbaustein
OS	das Betriebssystem (engl. für Operating System)
Parser	engl. to parse - „analysieren“ bzw. lateinisch pars - „Teil“ im Deutschen gelegentlich auch Zerteiler, Analysiert die Semantik des Scripts um daraufhin Aktionen durchzuführen
PC	Elektronischer Rechner (engl. für Personal Computer)
PG	Programmier Gerät, meist ein PC
PLC	s. a. SPS (engl. für Programmable logic controller)
PLCVarTab	Variablentabelle (Symboltabelle)
PN	s. a. Profinet
PNG	Grafikaustausch Format (engl. für Portable Network Graphics)

PROFINET	Bezeichnung für industriellen Netzwerkstandard (engl. für Process Field Network)
SCL	Strukturierter Text (engl. für Structured Control Language)
SDK	s. a. IDE (engl für Software Development Kit)
SFB	System Funktionsbaustein
SFC	System Funktion
SOP	Same Origin Policy
SPS	Speicher Programmierbare Steuerung
ST	Strukturierter Text, s. a. SCL
SVG	skalierbare Vektorgrafik (engl. für Scalable Vector Graphics)
TCP	Transmission Control Protocol
TIA	Totally Integrated Automation
TN	Teilnehmer
UDT	Benutzerdefinierter Datentyp in Form einer Struktur (engl. für User Defined Typ)
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
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. In einer CPU mit PROFINET-Schnittstelle ist ein Webserver integriert, der mit anwenderdefinierten Webseiten erweiterbar ist
WPO	Webdaten Optimierung (engl. für Web Performance Optimization)
WWW	Internet (engl. für World Wide Web)
WYSIWYG	Man sieht im Editor sofort was man bekommt, sowohl textuell als auch grafisch dargestellt (engl. für what you see is what you get)

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.

Analyse, 7
Analyse neue Realisierung, 12
Architektur, 16
Archon
    Cover, 1
Aufbau, 6
Ausblick, 25

Erfüllung, 24

Fazit, 24, 25
Forschungsstand, 3

Hilfsmittel, 15
Hinführung, 1

Implementierung, 14, 23

Motivation, 4

Resultate, 24

Spiel Analyse, 7

Technologien, 14

Unittests, 24

Zentrale Begriffe, 4
Ziel, 5