Fernfachhochschule Schweiz

Empire

Projektdokumentation

Autor: Remo Inverardi ‹remo.inverardi@students.ffhs.ch›

Datum: 26. November 2023

Modul: BSc Inf WebE, ZH-Sa-1, HS 23/24

Dozent: Philipp Lauwiner

Änderungshistorie

Datum	Version	Änderungen
10.09.2023		• Dokumentstruktur erstellt. • Kapitel «Einleitung» hinzugefügt.
01.10.2023	1	• Kapitel «Spielregeln und Spielziel» hinzugefügt. • Kapitel «Anforderungen und Randbedingungen» hinzugefügt. • Kapitel «Planung» hinzugefügt. • Kapitel «Protokolle und Schnittstellen» hinzugefügt. • Kapitel «Technische Umsetzung» hinzugefügt.
29.10.2023	2	• Feedback zu Version 1 eingearbeitet. • Kapitel «Anforderungen und Randbedingungen» nachgeführt sowie stellenweise präzisiert. • Kapitel «Architektur» hinzugefügt. • Kapitel «Protokolle und Schnittstellen» nachgeführt sowie stellenweise präzisiert; Wireframes vervollständigt; Nachrichtenformate für Spielzustand und Aktionen beschrieben. • Kapitel «Journal» hinzugefügt.
26.11.2023	3	• Feedback zu Version 2 eingearbeitet. • Kapitel «Anforderungen und Randbedingungen» nachgeführt sowie stellenweise präzisiert; Anforderungen von «soll» auf «kann» zurückgestuft. • Kapitel «Architektur» nachgeführt und stellenweise präzisiert; Anforderungen von «soll» auf «kann» zurückgestuft. • Kapitel «Protokolle und Schnittstellen» nachgeführt und stellenweise präzisiert; Spielzustand angepasst und mit Beispielen vervollständigt; Aktionen angepasst. • Kapitel «Journal» nachgeführt.

Inhaltsverzeichnis

Eigenständigkeitserklärung

Mit der Abgabe dieser Arbeit bestätige ich,

- dass ich die vorliegende Arbeit selbstständig verfasst habe,
- dass alle sinngemäss und wörtlich übernommenen Textstellen aus fremden Quellen kenntlich gemacht wurden,
- dass alle mit Hilfsmitteln erbrachten Teile der Arbeit präzise deklariert wurden,
- dass keine anderen als die im Hilfsmittelverzeichnis aufgeführten Hilfsmittel verwendet wurden,
- dass das Thema, die Arbeit oder Teile davon nicht bereits Gegenstand eines Leistungsnachweises eines anderen Moduls waren, sofern dies nicht ausdrücklich mit der Referentin oder dem Referenten im Voraus vereinbart wurde,
- dass ich mir bewusst bin, dass meine Arbeit elektronisch auf Plagiate und auf Drittautorenschaft menschlichen oder technischen Ursprungs überprüft werden kann und ich hiermit der Fernfachhochschule Schweiz das Nutzungsrecht so weit einräume, wie es für diese Verwaltungshandlungen notwendig ist.

Hilfsmittelverzeichnis

Hilfsmittel	**verwendete Funktionen**	**betroffene Stellen**
GNU Aspell	Rechtschreibung prüfen	ganzes Dokument

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Dieses Dokument enthält die Projektdokumentation für das Computerspiel «Empire».

1.1 Projektumfeld

Die Teilnehmer des Moduls WebE haben den Auftrag erhalten, ihr eigenes Computerspiel als semesterbegleitendes Projekt zu entwickeln. Die Spielidee und die technische Umsetzung werden von den Studierenden innerhalb der vorgegebenen Richtlinien [2] frei gewählt.

1.2 Projektziele

Die Studierenden entwickeln ihr eigenes Computerspiel als semesterbegleitendes Projekt. Während dieses Projekts lernen sie unter anderem [3]:

- Wie man aktuelle Web-Technologien charakterisiert, positioniert und einsetzt.
- Wie man Protokolle für Web-Anwendungen entwirft und implementiert.
- Wie man Daten in einer serverseitigen Persistenzschicht ablegt und abruft.
- Wie man die Performanz von Web-Anwendungen optimiert.

2 Spielregeln und Spielziel

Bei «Empire» baut ein Spieler sein Imperium auf, indem er eine Karte erkundet und darauf Ressourcen einsammelt, Ortschaften gründet und Fabriken errichtet. Gegenspieler können angegriffen und vernichtet werden.

2.1 Spielregeln

Die **Karte** besteht aus einer Menge von Kartenfeldern.

Die **Kartenfelder** sind als Hexagone angeordnet. Es gibt fünf Arten von Feldern. Je nach Art des Felds können unterschiedliche Objekte darauf platziert sein (Tabelle 1); z. B. können Spielfiguren nur Grasflächen, Hügel und Wälder betreten.

Auf der Karte sind **Ressourcen** verteilt. Die Ressourcen werden bei Spielbeginn auf zufälligen Kartenfeldern platziert. Mehrere Ressourcen auf einem Feld sind nicht möglich. Eingesammelte Ressourcen verschwinden von der Karte und wachsen nach neun Spielrunden wieder nach.

Jeder Spieler hat eine **Spielfigur**. Die Figuren werden bei Spielbeginn auf zufälligen Kartenfeldern platziert. Mehrere Figuren auf einem Feld sind nicht möglich. Eine Figur sammelt alle Ressourcen auf den angrenzenden Felder automatisch ein.

Eine Partie besteht aus einer Reihe von **Spielrunden**.

Pro Spielrunde führt jeder Spieler einen **Spielzug** durch. Es gibt sechs Arten von Zügen. Je nach Art des Zugs werden unterschiedliche Ressourcen in verschiedenen Mengen benötigt (Tabelle 2); z. B. kostet die Gründung eines Dorfs jeweils vier Einheiten Gold und Nahrung.

Eine **Fabrik** produziert pro Spielrunde eine zufällige Ressource. Fabriken können nur auf freien Grasflächen errichtet werden. Gegenspieler können fremde Fabriken betreten, aber die produzierten Ressourcen nicht stehlen.

Eine **Ortschaft** sammelt alle Ressourcen auf den angrenzenden Feldern automatisch ein. Ortschaften können als Dörfer auf freien Grasflächen und Hügeln errichtet und später zu Städten und Metropolen ausgebaut werden. Gegenspieler können fremde Ortschaften betreten, aber die angrenzenden Ressourcen nicht stehlen.

Um einen **Angriff** auf eine gegnerische Spielfigur durchzuführen, muss sich die angreifende Figur auf einem angrenzenden Kartenfeld befinden. Jeder ausgeführte Angriff kostet den angreifenden Spieler eine Einheit Waffen.

Bei Spielbeginn hat jede Spielfigur zehn **Trefferpunkte**. Mit jedem Angriff durch eine gegnerische Figur wird ein Punkt abgezogen. Verliert eine Figur alle Punkte, scheidet der angegriffene Spieler aus der Partie aus und der angreifende Spieler erhält seine Besitztümer.

Typ	**Fabriken**	**Ortschaften**	**Ressourcen**	**Spielfiguren**
Grasfläche	✓	✓	✓	✓
Hügel		✓	✓	✓
Wald			✓	✓
Gebirge				
Gewässer				

Tabelle 1: Arten von Kartenfeldern

Spielzug	**Forschung**	**Gold**	**Nahrung**	**Waffen**
Spielfigur bewegen				
Dorf gründen		4	4	
Dorf zu Stadt ausbauen	4	8	8	
Stadt zu Metropole ausbauen	8	12	12	
Fabrik errichten	12	8		
Gegenspieler angreifen				2

Tabelle 2: Arten von Spielzügen

2.2 Spielziel

Das **Spielziel** ist es, das mächtigste Imperium zu erschaffen. Es gibt zwei Möglichkeiten, um dieses Ziel zu erreichen. Die erste Möglichkeit besteht darin, eine Metropolen zu besitzen. Die Partie endet sofort, sobald ein Spieler diese Marke erreicht. Die zweite Möglichkeit besteht darin, alle Gegenspieler zu vernichten.

Für die **Highscores** relevant sind die Einheiten Gold eines Spielers bei Spielende. Wenn der Sieger ein Ergebnis erzielt, das zu den zehn besten Ergebnissen aller Zeiten gehört, dann wird das Ergebnis in den Highscores gespeichert.

3 Anforderungen und Randbedingungen

3.1 Randbedingungen

Gemäss den Informationen zur Projektarbeit [2] und den mündliche Hinweisen des Dozenten sind beim Entwurf und bei der Implementation der Spiels folgende Randbedingungen einzuhalten:

Architektur (B-1) – Das Spiel hat eine Client-Server-Architektur. Der Client ist für die Benutzerinteraktion verantwortlich und kommuniziert mit dem Server. Der Server steuert die Partien und garantiert einen regelkonformen Spielablauf.

Persistenz (B-2) – Der Server persistiert langlebige Daten in einer Datenbank.

Programmiersprache (B-3) – Sowohl der Client als auch der Server müssen in *JavaScript* programmiert sein.

Protokoll (B-4) – Das Protokoll für die Kommunikation zwischen dem Client und dem Server muss menschenlesbar und textbasiert sein.

Umfang (B-5) – Das Spiel muss mindestens drei Levels haben.

3.2 Anwendungsfälle

Die Anwendungsfälle des Spiels werden mit zwei **Anwendungsfalldiagrammen** beschrieben.

1. Das erste Diagramm zeigt diejenigen Anwendungsfälle, die **vor und nach einer Partie** möglich sind (Abbildung 1 sowie Tabellen 3 bis 9).

   Bei diesen Anwendungsfällen gibt es den Akteur «Spieler» jeweils in zwei Ausprägungen. Der «Spielleiter» erstellt und startet eine Partie. Der «Spielteilnehmer» nimmt an einer Partie Teil, die von einem Spielleiter erstellt wurde.

2. Das zweite Diagramm zeigt diejenigen Anwendungsfälle, die **während einer Partie** möglich sind (Abbildung 2 sowie Tabellen 10 bis 20).

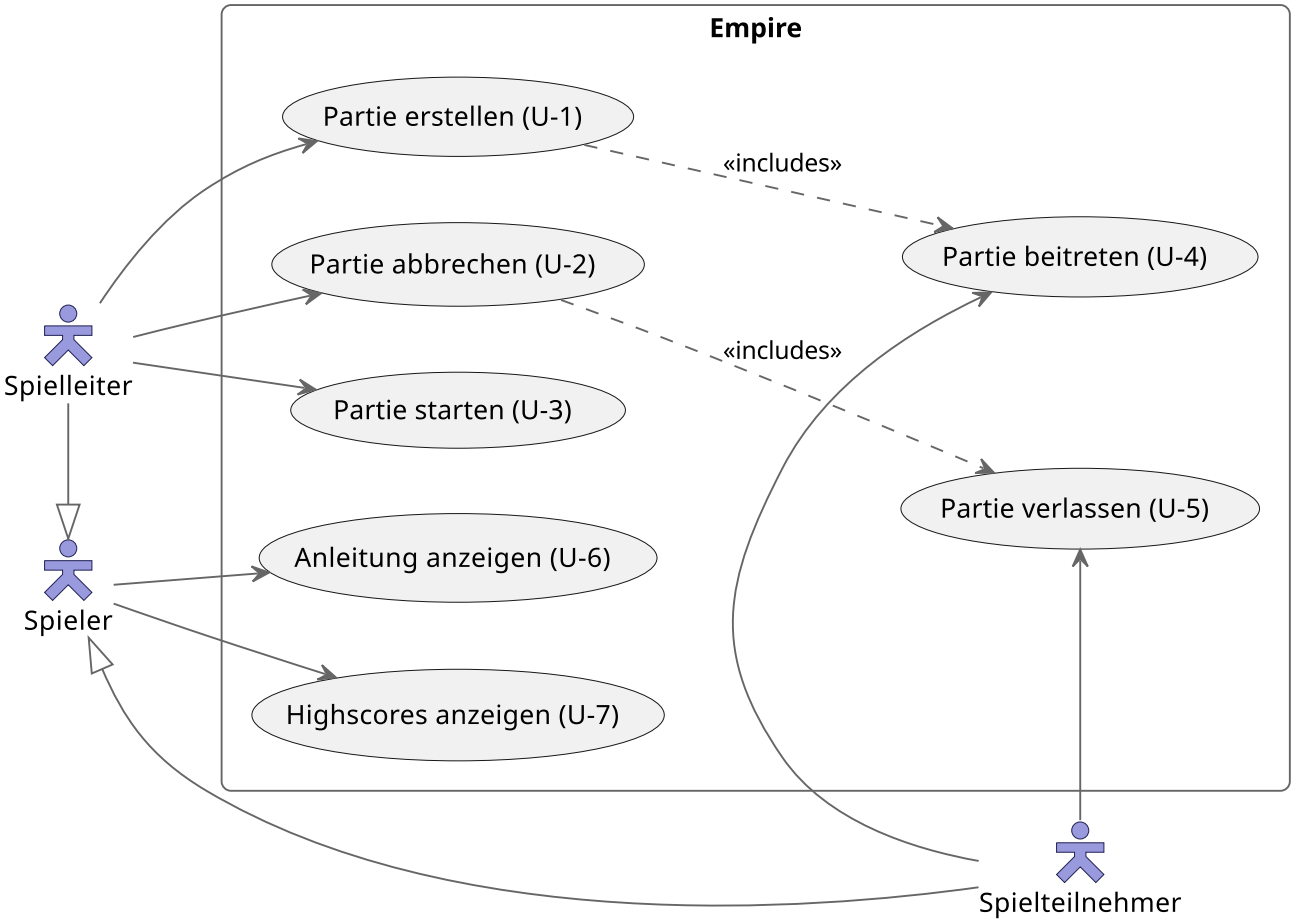

Abbildung 1: Anwendungsfälle – vor und nach einer Partie

Partie erstellen (U-1)

Ziel	Die Spielleiter kann eine neue Partie erstellen.
Akteur	Spielleiter
Auslöser	Die Spielleiter klickt auf «Partie erstellen».
Vorbedingung	Der Spielleiter nimmt an keiner Partie teil.
Ablauf im Normalfall	1. Der Spielleiter klickt auf «Partie erstellen». 2. Der Spielleiter wählt eine Karte. 3. Der Spielleiter wählt seinen Nicknamen. 4. Der Spielleiter klickt auf «Weiter».
Nachbedingung im Normalfall	Die Partie wurde erstellt.
Fehlerfall	Die maximale Anzahl paralleler Partien ist erreicht.
Ablauf im Fehlerfall	Dem Spielleiter wird eine Fehlermeldung angezeigt.
Nachbedingung im Fehlerfall	Die Partie wurde nicht erstellt.
Aufwandschätzung	mittel

Tabelle 3: Anwendungsfall – Partie erstellen (U-1)

Partie abbrechen (U-2)

Ziel	Der Spielleiter kann eine Partie abbrechen.
Akteur	Spielleiter
Auslöser	Der Spielleiter klickt auf «Partie abbrechen».
Vorbedingungen	1. Die Partie wurde erstellt. 2. Die Partie wurde weder abgebrochen noch gestartet.
Ablauf	1. Der Spielleiter klickt auf «Partie abbrechen». 2. Der Spielleiter wird auf die Startseite weitergeleitet. 3. Den Gegenspielern wird eine Fehlermeldung angezeigt.
Nachbedingung	Die Partie wurde abgebrochen.
Aufwandschätzung	mittel

Tabelle 4: Anwendungsfall – Partie abbrechen (U-2)

Partie starten (U-3)

Ziel	Der Spielleiter kann eine Partie starten.
Akteur	Spielleiter
Auslöser	Der Spielleiter klickt auf «Partie starten».
Vorbedingungen	1. Die Partie wurde erstellt. 2. Die Partie wurde weder abgebrochen noch gestartet.
Ablauf im Normalfall	1. Der Spielleiter klickt auf «Partie starten». 2. Alle Spieler werden auf die Kartenseite weitergeleitet. 3. Der Spielleiter ist an der Reihe.
Nachbedingung im Normalfall	Die Partie wurde gestartet.
Fehlerfall	Weniger als zwei Slots sind voll.
Ablauf im Fehlerfall	Dem Spielleiter wird eine Fehlermeldung angezeigt.
Nachbedingung im Fehlerfall	Die Partie wurde nicht gestartet.
Aufwandschätzung	mittel

Tabelle 5: Anwendungsfall – Partie starten (U-3)

Partie beitreten (U-4)

Ziel	Ein Spielteilnehmer kann einer Partie beitreten.
Akteur	Spielteilnehmer
Auslöser	Der Spielteilnehmer klickt auf «Partie beitreten».
Vorbedingungen	1. Die Partie wurde erstellt. 2. Die Partie wurde weder abgebrochen noch gestartet.
Ablauf im Normalfall	1. Der Spielteilnehmer klickt auf «Partie beitreten». 2. Der Spielteilnehmer wählt eine Partie. 3. Der Spielteilnehmer wählt seinen Nicknamen. 4. Der Spielteilnehmer klickt auf «Weiter». 5. Alle Spieler sehen den Spielteilnehmer.
Nachbedingung im Normalfall	Der Spielteilnehmer wurde zur Partie hinzugefügt.
Fehlerfälle	1. Alle Slots sind voll. 2. Der gewählte Nickname existiert bereits.
Ablauf im Fehlerfall	Dem Spielteilnehmer wird eine Fehlermeldung angezeigt.
Nachbedingung im Fehlerfall	Der Spielteilnehmer wurde nicht zur Partie hinzugefügt.
Aufwandschätzung	mittel

Tabelle 6: Anwendungsfall – Partie beitreten (U-4)

Partie vor Spielbeginn verlassen (U-5)

Ziel	Ein Spielteilnehmer kann eine Partie vor Spielbeginn verlassen.
Akteur	Spielteilnehmer
Auslöser	Der Spielteilnehmer klickt auf «Partie verlassen».
Vorbedingungen	1. Die Partie wurde erstellt. 2. Die Partie wurde weder abgebrochen noch gestartet. 3. Der Spielteilnehmer wurde zur Partie hinzugefügt.
Ablauf	1. Der Spielteilnehmer klickt auf «Partie verlassen». 2. Der Spielteilnehmer wird auf die Startseite weitergeleitet. 3. Die Gegenspieler sehen den Spielteilnehmer nicht mehr.
Nachbedingung	Der Spielteilnehmer wurde aus der Partie entfernt.
Aufwandschätzung	mittel

Tabelle 7: Anwendungsfall – Partie vor Spielbeginn verlassen (U-5)

Anleitung anzeigen (U-6)

Ziel	Ein Spieler kann sich die Anleitung ansehen.
Akteur	Spieler
Auslöser	Der Spieler klickt auf «Anleitung anzeigen».
Ablauf	1. Der Spieler klickt auf «Anleitung anzeigen». 2. Die Anleitung wird angezeigt.
Aufwandschätzung	tief

Tabelle 8: Anwendungsfall – Anleitung anzeigen (U-6)

Highscores anzeigen (U-7)

Ziel	Ein Spieler kann sich die Highscores ansehen.
Akteur	Spieler
Auslöser	Der Spieler klickt auf «Highscores anzeigen».
Ablauf	1. Der Spieler klickt auf «Highscores anzeigen». 2. Die Highscores werden angezeigt.
Aufwandschätzung	tief

Tabelle 9: Anwendungsfall – Highscores anzeigen (U-7)

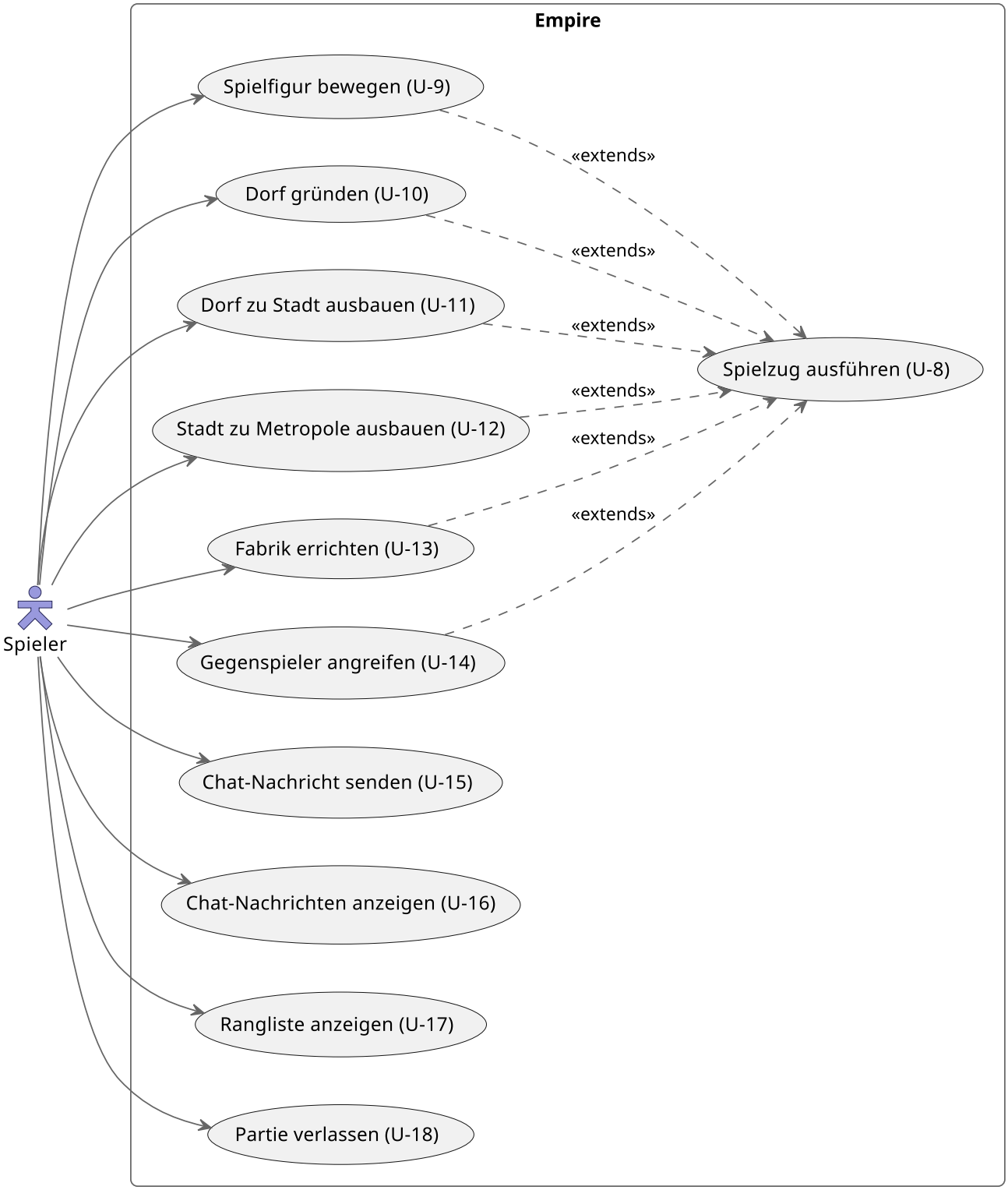

Abbildung 2: Anwendungsfälle – während einer Partie

Spielzug ausführen (U-8)

Ziel	Ein Spieler kann einen Spielzug ausführen.
Akteur	Spieler
Auslöser	Der Spieler führt einen Spielzug aus.
Vorbedingungen	1. Die Partie wurde gestartet und läuft noch. 2. Der Spieler ist an der Reihe.
Ablauf im Normalfall	1. Der Spieler führt einen Spielzug aus. 2. Alle Spieler sehen den Übergang vom bisherigen Spielzustand zum neuen Spielzustand als Animation.
Nachbedingung im Normalfall	1. Der Spielzug wurde ausgeführt. 2. Der nächste Spieler ist an der Reihe.
Fehlerfall	Der Spielzug ist ungültig.
Ablauf im Fehlerfall	Der Spielzug wird ignoriert.
Nachbedingung im Fehlerfall	1. Der Spielzug wurde nicht ausgeführt. 2. Der Spieler ist erneut an der Reihe.
Aufwandschätzung	hoch

Tabelle 10: Anwendungsfall – Spielzug ausführen (U-8)

Spielfigur bewegen (U-9)

Ziel	Ein Spieler kann seine Spielfigur bewegen.
Akteur	Spieler
Auslöser	Der Spieler bewegt seine Spielfigur auf ein Feld, auf dem kein Gegenspieler steht.
Ablauf	1. Der Spieler bewegt seine Spielfigur. 2. Der Spielzug wird ausgeführt → Spielzug ausführen (U-8).
Aufwandschätzung	mittel

Tabelle 11: Anwendungsfall – Spielfigur bewegen (U-9)

Dorf gründen (U-10)

Ziel	Ein Spieler kann ein Dorf gründen.
Akteur	Spieler
Auslöser	Der Spieler klickt auf «Dorf gründen».
Ablauf	1. Der Spieler klickt auf «Dorf gründen». 2. Der Spielzug wird ausgeführt → Spielzug ausführen (U-8).
Aufwandschätzung	mittel

Tabelle 12: Anwendungsfall – Dorf gründen (U-10)

Dorf zu Stadt ausbauen (U-11)

Ziel	Ein Spieler kann ein Dorf zu einer Stadt ausbauen.
Akteur	Spieler
Auslöser	Der Spieler klickt auf «Dorf zu Stadt ausbauen».
Ablauf	1. Der Spieler klickt auf «Dorf zu Stadt ausbauen». 2. Der Spielzug wird ausgeführt → Spielzug ausführen (U-8).
Aufwandschätzung	mittel

Tabelle 13: Anwendungsfall – Dorf zu Stadt ausbauen (U-11)

Stadt zu Metropole ausbauen (U-12)

Ziel	Ein Spieler kann eine Stadt zu einer Metropole ausbauen.
Akteur	Spieler
Auslöser	Der Spieler klickt auf «Stadt zu Metropole ausbauen».
Ablauf	1. Der Spieler klickt auf «Stadt zu Metropole ausbauen». 2. Der Spielzug wird ausgeführt → Spielzug ausführen (U-8).
Aufwandschätzung	mittel

Tabelle 14: Anwendungsfall – Stadt zu Metropole ausbauen (U-12)

Fabrik errichten (U-13)

Ziel	Ein Spieler kann eine Fabrik errichten.
Akteur	Spieler
Auslöser	Der Spieler klickt auf «Fabrik errichten».
Ablauf	1. Der Spieler klickt auf «Fabrik errichten». 2. Der Spielzug wird ausgeführt → Spielzug ausführen (U-8).
Aufwandschätzung	mittel

Tabelle 15: Anwendungsfall – Fabrik errichten (U-13)

Gegenspieler angreifen (U-14)

Ziel	Der Spieler kann einen Gegenspieler angreifen.
Akteur	Spieler
Auslöser	Der Spieler bewegt seine Spielfigur auf ein Feld, auf dem ein Gegenspieler steht.
Ablauf	1. Der Spieler klickt auf «Fabrik errichten». 2. Der Spielzug wird ausgeführt → Spielzug ausführen (U-8).
Aufwandschätzung	mittel

Tabelle 16: Anwendungsfall – Gegenspieler angreifen (U-14)

Chat-Nachricht senden (U-15)

Ziel	Der Spieler kann eine Chat-Nachricht an seine Gegenspieler senden.
Akteur	Spieler
Auslöser	Der Spieler klickt auf «Senden».
Vorbedingung	1. Die Partie wurde gestartet und läuft noch. 2. Der Spieler nimmt an der Partie teil.
Ablauf	1. Der Spieler gibt eine Chat-Nachricht ein. 2. Der Spieler klickt auf «Senden» 3. Alle Spieler sehen die Chat-Nachricht.
Aufwandschätzung	mittel

Tabelle 17: Anwendungsfall – Chat-Nachricht senden (U-15)

Chat-Nachrichten anzeigen (U-16)

Ziel	Der Spieler kann sich die Chat-Nachrichten seiner Gegenspieler ansehen.
Akteur	Spieler
Auslöser	Der Spieler klickt auf «Nachrichten».
Vorbedingung	1. Die Partie wurde gestartet und läuft noch. 2. Der Spieler nimmt an der Partie teil.
Ablauf	1. Der Spieler klickt auf «Nachrichten» 2. Der Spieler sieht die Chat-Nachrichten.
Aufwandschätzung	mittel

Tabelle 18: Anwendungsfall – Chat-Nachrichten anzeigen (U-16)

Rangliste anzeigen (U-17)

Ziel	Der Spieler kann sich die Rangliste der Partie ansehen.
Akteur	Spieler
Auslöser	Der Spieler klickt auf «Rangliste».
Vorbedingung	1. Die Partie wurde gestartet. 2. Der Spieler nimmt an der Partie teil.
Ablauf	1. Der Spieler klickt auf «Rangliste». 2. Der Spieler sieht die Rangliste.
Aufwandschätzung	mittel

Tabelle 19: Anwendungsfall – Rangliste anzeigen (U-17)

Partie nach Spielbeginn verlassen (U-18)

Ziel	Ein Spieler kann eine Partie nach Spielbeginn verlassen.
Akteur	Spieler
Auslöser	Der Spieler klickt auf «Partie verlassen».
Vorbedingung	1. Die Partie wurde gestartet. 2. Der Spieler nimmt an der Partie teil.
Ablauf	1. Der Spieler klickt auf «Partie verlassen». 2. Der Spieler wird auf die Startseite weitergeleitet. 3. Die Gegenspieler sehen den Spieler nicht mehr.
Nachbedingung	Der Spieler wurde aus der Partie entfernt.
Aufwandschätzung	mittel

Tabelle 20: Anwendungsfall – Partie nach Spielbeginn verlassen (U-18)

3.3 Funktionale Anforderungen

Die funktionalen Anforderungen legen fest, welche Funktionalität und welches Verhalten das System besitzen bzw. zeigen soll. Diese Anforderungen spezifizieren also, was das System tun soll [1]. Die Anforderungen in den folgenden Abschnitten konkretisieren und präzisieren die Spielregeln und die Anwendungsfälle.

3.3.1 Partien

Ablauf einer Partie (F-10) Muss – Eine Partie besteht aus einer Reihe von Spielrunden. Die Runden sind aufsteigend seit Spielbeginn nummeriert.

Ende einer Partie (F-11) Muss – Eine Partie endet, sobald ein Spieler eine Metropole besitzt. Der Spieler, der diese Marke zuerst erreicht, gewinnt die Partie.

Ende einer Partie (F-12) Muss – Eine Partie endet, falls der vorletzte Spieler die Partie verlässt. Der verbleibende Spieler gewinnt die Partie.

Ende einer Partie (F-13) Muss – Eine Partie endet, falls der vorletzte Spieler von seinem Gegner vernichtet wird. Der verbleibende Spieler gewinnt die Partie.

3.3.2 Karten und Kartenfelder

Anordnung der Kartenfelder (F-20) Muss – Die Kartenfelder sind als Hexagone angeordnet. Es gibt fünf Arten von Kartenfeldern (Tabelle 1).

Belegung der Kartenfelder (F-21) Muss – Ein Kartenfeld kann leer oder belegt sein. Ein belegtes Feld enthält entweder eine Ressource, ein Dorf, eine Stadt, eine Metropole oder eine Fabrik. Doppelbelegungen sind nicht erlaubt.

Sichtbarkeit der Kartenfelder (F-23) Kann – Zu Beginn einer Partie sieht ein Spieler nur das Kartenfeld, auf dem seine Spielfigur steht, sowie alle angrenzenden Felder. Die restlichen Felder sind unsichtbar.

Sichtbarkeit der Kartenfelder (F-24) Kann – Im Verlauf einer Partie sieht ein Spieler zudem diejenigen Kartenfelder, die seine Spielfigur bereits betreten hat, sowie alle angrenzenden Felder.

3.3.3 Spieler und Spielfiguren

Angreifen einer Spielfigur (F-30) Muss – Eine Spielfigur kann eine gegnerische Figur angreifen, falls sie sich auf einem angrenzenden Kartenfeld befindet. Jeder ausgeführte Angriff kostet den angreifenden Spieler eine Einheit Waffen.

Bewegen der Spielfigur (F-31) Muss – Pro Spielzug kann eine Spielfigur um höchstens ein Feld bewegt werden.

Bewegen der Spielfigur (F-32) Muss – Um eine Spielfigur auf ein Kartenfeld zu bewegen, müssen zwei Bedingungen erfüllt sein: Erstens darf sich auf dem Zielfeld noch keine Spielfigur befinden. Zweitens muss das Zielfeld für Spielfiguren geeignet sein (Tabelle 1).

Platzieren der Spielfiguren (F-33) Muss – Bei Spielbeginn werden die Spielfiguren zufällig auf der Karte platziert. Dabei werden nur Kartenfelder gewählt, die von Figuren betreten werden können (Tabelle 1). Mehrere Figuren auf einem Feld sind nicht erlaubt.

Trefferpunkte einer Spielfigur (F-34) Muss – Bei Spielbeginn hat jede Spielfigur zehn Trefferpunkte. Mit jedem Angriff durch eine gegnerische Figur wird ein Punkt abzogen. Verliert eine Figur alle Punkte, scheidet der angegriffene Spieler aus und der angreifende Spieler erhält seine Besitztümer.

3.3.4 Spielrunden und Spielzüge

Ablauf einer Spielrunde (F-40) Muss – Eine Spielrunde besteht aus einer Reihe von Spielzügen. Die Züge sind aufsteigend seit Spielbeginn nummeriert.

Ablauf einer Spielrunde (F-41) Muss – Jeder Spieler darf in jeder Spielrunde höchsten einen Zug ausführen. Die Spieler führen ihre Spielzüge nacheinander aus und müssen jeweils aufeinander warten.

Arten von Spielzügen (F-42) Muss – Es gibt sechs Arten von Spielzügen (Tabelle 2).

Spielzug nicht erlaubt (F-43) Muss – Wenn ein Spieler einen ungültigen Spielzug auszuführen versucht, dann wird dieser Zug ignoriert.

Spielzug nicht möglich (F-44) Muss – Wenn ein Spieler keine gültigen Spielzüge ausführen kann, dann wird dieser Spieler übersprungen.

3.3.5 Ressourcen und Inventar

Einsammeln von Ressourcen (F-50) Muss – Zu Beginn jeder Spielrunde sammelt jede Ortschaft alle Ressourcen auf den angrenzenden Feldern automatisch ein. Die eingesammelten Ressourcen werden dem Besitzer der jeweiligen Ortschaft gutgeschrieben.

Einsammeln von Ressourcen (F-51) Muss – Am Ende jedes Spielzugs sammelt die Spielfigur alle Ressourcen auf den angrenzenden Feldern automatisch ein.

Inventar eines Spielers (F-52) Muss – Das Inventar eines Spielers enthält die Ressourcen, die der Spieler besitzt; d. h. die Einheiten Forschung, Gold, Nahrung und Waffen, die dem Spieler für Spielzüge zur Verfügung stehen.

Lösen von Konflikten (F-53) Muss – Wenn mehrere Ortschaften an eine Ressource angrenzen, dann wird die Ressource von der ältesten Ortschaft eingesammelt.

Nachwachsen der Ressourcen (F-54) Muss – Eingesammelte Ressourcen verschwinden von der Karte und wachsen auf demselben Kartenfeld nach fünf Spielrunden wieder nach.

Platzieren der Ressourcen (F-55) Muss – Bei Spielbeginn werden die Ressourcen zufällig auf der Karte platziert. Dabei werden nur Kartenfelder gewählt, die von Spielfiguren betreten werden können (Tabelle 1). Mehrere Ressourcen auf einem Feld sind nicht erlaubt.

Produzieren von Ressourcen (F-56) Muss – Zu Beginn jeder Spielrunde produziert jede Fabrik eine zufällige Ressource. Die produzierten Ressourcen werden dem Besitzer der jeweiligen Fabrik gutgeschrieben.

3.3.6 Ortschaften und Fabriken

Betreten einer Ortschaft (F-60) Muss – Eine Spielfigur kann sowohl eigene als auch fremde Ortschaften betreten.

Betreten einer Fabrik (F-61) Muss – Eine Spielfigur kann sowohl eigene als auch fremde Fabriken betreten.

Erstellen eines Dorfs (F-62) Muss – Um ein Dorf zu gründen, müssen drei Bedingungen erfüllt sein: Erstens darf das Feld, in dem die Spielfigur steht, noch nicht belegt sein. Zweitens muss die Art des Felds für Dörfer geeignet sein (Tabelle 1). Drittens muss der Spieler über die nötigen Ressourcen verfügen (Tabelle 2). Diese Ressourcen werden dem Spieler sofort belastet.

Erstellen einer Stadt (F-63) Muss – Um ein Dort zu einer Stadt auszubauen, müssen zwei Bedingungen erfüllt sein: Erstens muss die Spielfigur im jeweiligen Dorf stehen. Zweitens muss der Spieler über die nötigen Ressourcen verfügen (Tabelle 2). Diese Ressourcen werden dem Spieler sofort belastet.

Erstellen einer Metropole (F-64) Muss – Um eine Stadt zu einer Metropole umzubauen, müssen zwei Bedingungen erfüllt sein: Erstens muss die Spielfigur in der jeweiligen Stadt stehen. Zweitens muss der Spieler über die nötigen Ressourcen verfügen (Tabelle 2). Diese Ressourcen werden dem Spieler sofort belastet.

Errichten einer Fabrik (F-65) Muss – Um eine Fabrik errichten zu können, müssen drei Bedingungen erfüllt sein: Erstens darf das Feld, auf dem die Spielfigur steht, noch nicht belegt sein. Zweitens muss die Art des Felds für Fabriken geeignet sein (Tabelle 1). Drittens muss der Spieler über die nötigen Ressourcen verfügen (Tabelle 2). Diese Ressourcen werden dem Spieler sofort belastet.

3.3.7 Ranglisten und Highscores

Anzeigen der Rangliste (F-70) Muss – Während einer Partie kann sich ein Spieler die Rangliste ansehen. Die Ränge der Spieler ergeben sich aufgrund der Einheiten Gold in absteigender Ordnung. Mehrere Spieler können sich einen Rang teilen.

Anzeigen der Highscores (F-71) Muss – Vor und nach einer Partie kann sich ein Spieler die Highscores mit den zehn besten Ergebnissen aller Zeiten ansehen. Die Ränge der Spieler ergeben sich aufgrund der Einheiten Gold in absteigender Ordnung. Mehrere Spieler können sich einen Rang teilen.

3.4 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen legen die gewünschten Qualitäten des Systems fest und beeinflussen dadurch die Systemarchitektur [4]. Diese Anforderungen spezifizieren also, wie das System arbeiten soll [1].

3.4.1 Benutzbarkeit

Persistenter Spielkontext (N-10) Muss – Der Spielkontext geht bei einem Reload der Web-Seite nicht verloren.

Unterstützung von Chrome (N-11) Muss – Das Spiel läuft in einem aktuellen Chrome-Browser (ab Version 117) fehlerfrei.

Unterstützung von Edge (N-12) Soll – Das Spiel läuft in einem aktuellen Edge-Browser (ab Version 117) fehlerfrei.

Unterstützung von Firefox (N-13) Muss – Das Spiel läuft in einem aktuellen Firefox-Browser (ab Version 118) fehlerfrei.

Unterstützung von Safari (N-14) Kann – Das Spiel läuft in einem aktuellen Safari-Browser (ab Version 17) fehlerfrei.

3.4.2 Leistungsfähigkeit

Die Referenzplattform für die Anforderungen in diesem Abschnitt ist ein Raspberry Pi 400 mit 4 GB Arbeitsspeicher.

Anzahl paralleler Partien (N-20) Muss – Bei 10 parallelen Partien werden die anderen Anforderungen, insbesondere die maximale Antwortzeit, nach wie vor erfüllt.

Anzahl paralleler Spieler (N-21) Muss – Bei 40 parallelen Spielern werden die anderen Anforderungen, insbesondere die maximale Antwortzeit, nach wie vor erfüllt.

maximale Antwortzeit (N-22) Muss – Die Antwortzeit bei Benutzerinteraktionen beträgt in 95% der Fälle weniger als 200 Millisekunden und in 99% der Fälle weniger als 1 Sekunde.

3.4.3 Responsiveness

Unterstützung verschiedener Bildschirmformate (N-30) Muss – Verschiedene Bildschirmformate werden sinnvoll genutzt; z. B. mobile Geräte im Landscape-Modus, mobile Geräte im Portrait-Modus. Die grafische Benutzeroberfläche nutzt den verfügbaren Raum, es werden keine Inhalte abgeschnitten und unnötiges Scrollen wird verhindert.

Unterstützung verschiedener Bildschirmgrössen (N-31) Muss – Verschiedene Bildschirmgrössen werden sinnvoll genutzt; z. B. Smartphones, Tablets, Laptops und Desktops. Die grafische Benutzeroberfläche nutzt den verfügbaren Raum, es werden keine Inhalte abgeschnitten und unnötiges Scrollen wird verhindert.

Unterstützung verschiedener Eingabemöglichkeiten (N-32) Muss – Verschiedene Eingabemöglichkeiten werden unterstützt. Dazu gehören insbesondere die Bedienung per Maus und die Bedienung per Touchscreen.

3.4.4 Zugänglichkeit

Bedienbarkeit bei viel Umgebungslicht (N-40) Muss – Auch bei viel Umgebungslicht ist ein angenehmes Spielen möglich. Diese Anforderung ist objektiv kaum messbar. Eine Verbesserung kann erreicht werden, indem die grafische Oberfläche einen hohen Kontrast aufweist.

Bedienbarkeit bei wenig Umgebungslicht (N-41) Soll – Auch bei wenig Umgebungslicht ist ein angenehmes Spielen möglich. Diese Anforderung ist objektiv kaum messbar. Eine Verbesserung kann erreicht werden. indem die grafische Oberfläche zwischen einem dunklen Modus und einem hellen Modus umschaltbar ist.

Bedienbarkeit durch Farbenblinde (N-42) Kann – Verschiedene Elemente des Spiels sind nicht nur durch ihre Farben sondern auch durch zusätzliche Attribute voneinander unterscheidbar.

4 Planung

Das Modul WebE umfasst fünf Präsenzveranstaltungen. In der Nachbereitung der vierten Präsenz wird das Projekt abgeschlossen. In der fünften Präsenz wird das Projekt dem Dozenten und den Mitstudierenden präsentiert.

4.1 Meilensteinplan

Die Entwicklung des Spiels erfolgt in vier Iterationen. Das Ende jeder Iteration wird durch einen Meilenstein markiert (Tabelle 21).

Meilenstein	Kriterien	Fällig
M-1	• Dokumentation aufgesetzt • Spielregeln und Spielziel beschrieben • Anforderungen und Randbedingungen gesammelt • Protokolle und Schnittstellen gesammelt • Zeitplan erstellt	30.09.2023
M-2	• Wireframes erstellt • Protokolle und Schnittstellen beschrieben • Entwicklungsumgebung aufgesetzt • Prototyp des Clients erstellt • Prototyp des Servers erstellt • Architektur festgelegt • Dokumentation nachgeführt	28.10.2023
M-3	• Implementation des Clients begonnen • Implementation des Servers begonnen • Dokumentation nachgeführt	25.11.2023
M-4	• Implementation des Clients abgeschlossen • Implementation des Servers abgeschlossen • Produktivumgebung aufgesetzt • Tests durchgeführt • Dokumentation abgeschlossen	23.12.2023

Tabelle 21: Meilensteinplan

4.2 Zeitplan

Die zeitliche Planung der ersten Iteration (I-1) wird durch den Umstand erschwert, dass die Spielregeln, das Spielziel, die Anforderungen, die Randbedingungen, die Protokolle und die Schnittstellen zu Beginn noch nicht bekannt sind, sondern erst im Verlauf dieser Iteration erarbeitet werden (Abbildung 3).

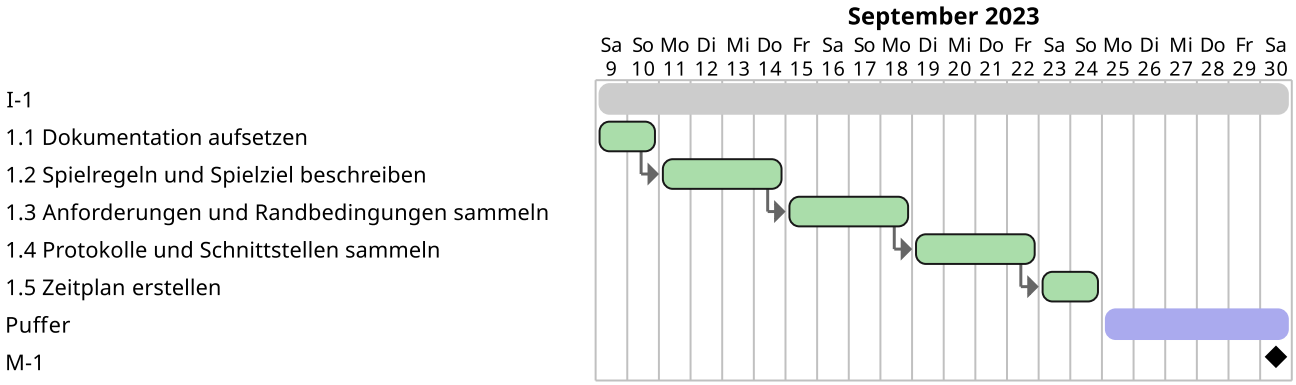

Abbildung 3: Zeitplan – erste Iteration (I-1)

Die Planung der weiteren Iterationen (I-2, I-3 und I-4) basiert auf diversen Annahmen, weil keine Erfahrungswerte aus ähnlichen Projekten vorhanden sind. Um die resultierende Unschärfe zu kompensieren, sind die Schätzungen eher grosszügig (Abbildungen 4 bis 6).

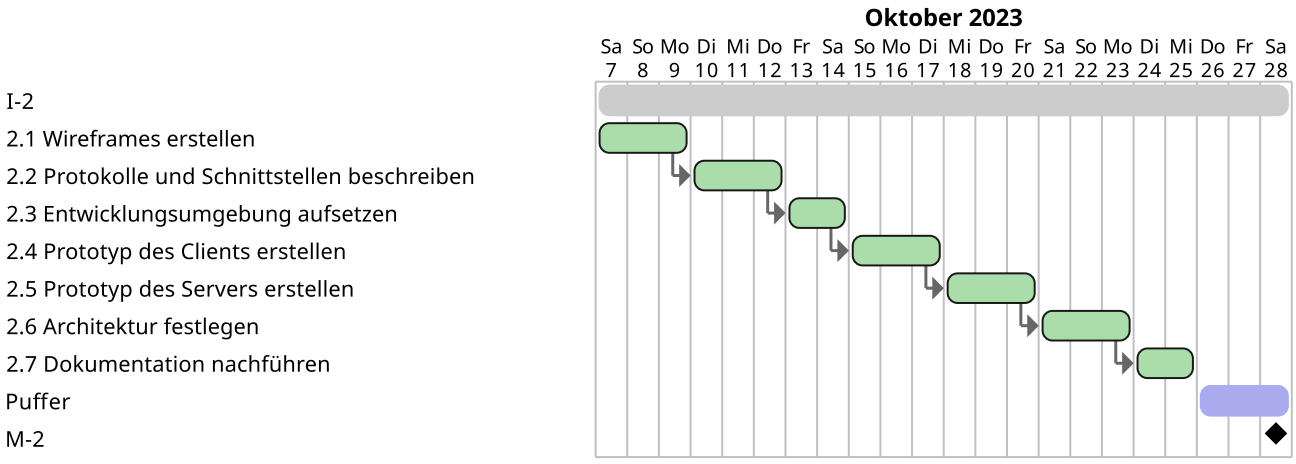

Abbildung 4: Zeitplan – zweite Iteration (I-2)

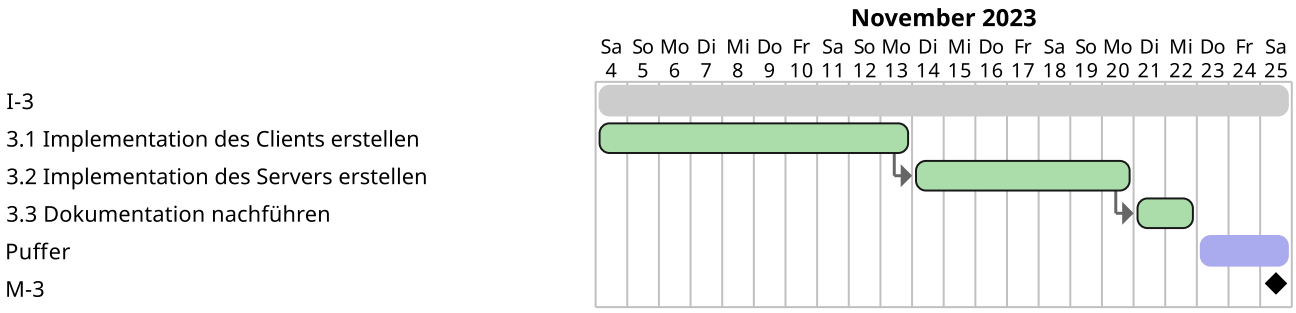

Abbildung 5: Zeitplan – dritte Iteration (I-3)

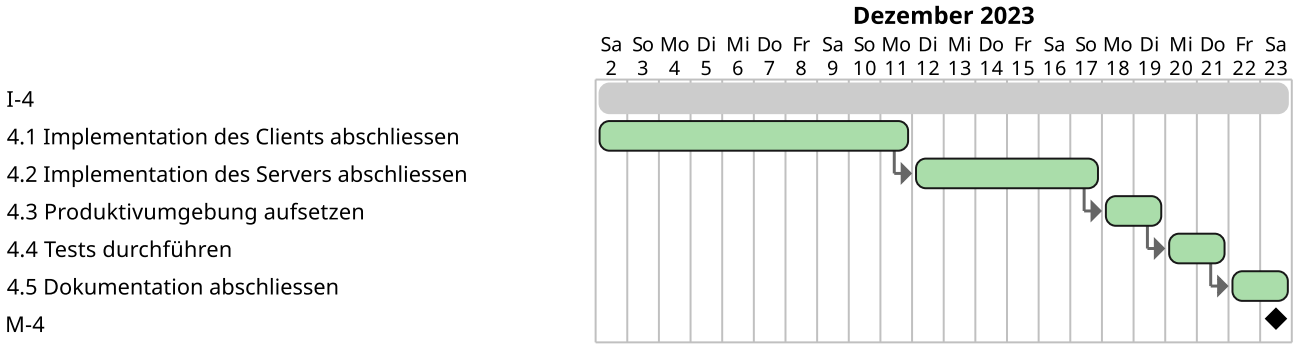

Abbildung 6: Zeitplan – vierte Iteration (I-4)

Die ersten drei Iterationen (I-1, I-2 und I-3) enden alle mit einem Puffer und einem Meilenstein. Die Meilensteine korrespondieren mit den Abgabeterminen der darauffolgenden Präsenzveranstaltungen. Die Puffer sollen sicherstellen, dass fristgerechte Abgaben möglich sind.

Die vierte Iteration (I-4) endet zwei Wochen vor dem darauffolgenden Abgabetermin und drei Wochen vor der abschliessenden Präsenzveranstaltung. Ein Puffer wird nicht benötigt, weil ausreichend Zeit für eine fristgerechte Abgabe zur Verfügung steht.

4.3 Fortschrittskontrolle

Zur Fortschrittskontrolle dient ein **Trello-Board** (Abbildung 7). Das Board ist online verfügbar:

https://www.inverardi.ch/bsc-inf-webe/board/

Die Liste «Planing» enthält die geplanten Aufgaben. Die Liste «Doing» enthält die Aufgaben, die sich gerade in Bearbeitung befinden. Die Liste «Done» enthält die abgeschlossenen Aufgaben. Alle Karten sind mit einem Fälligkeitsdatum versehen.

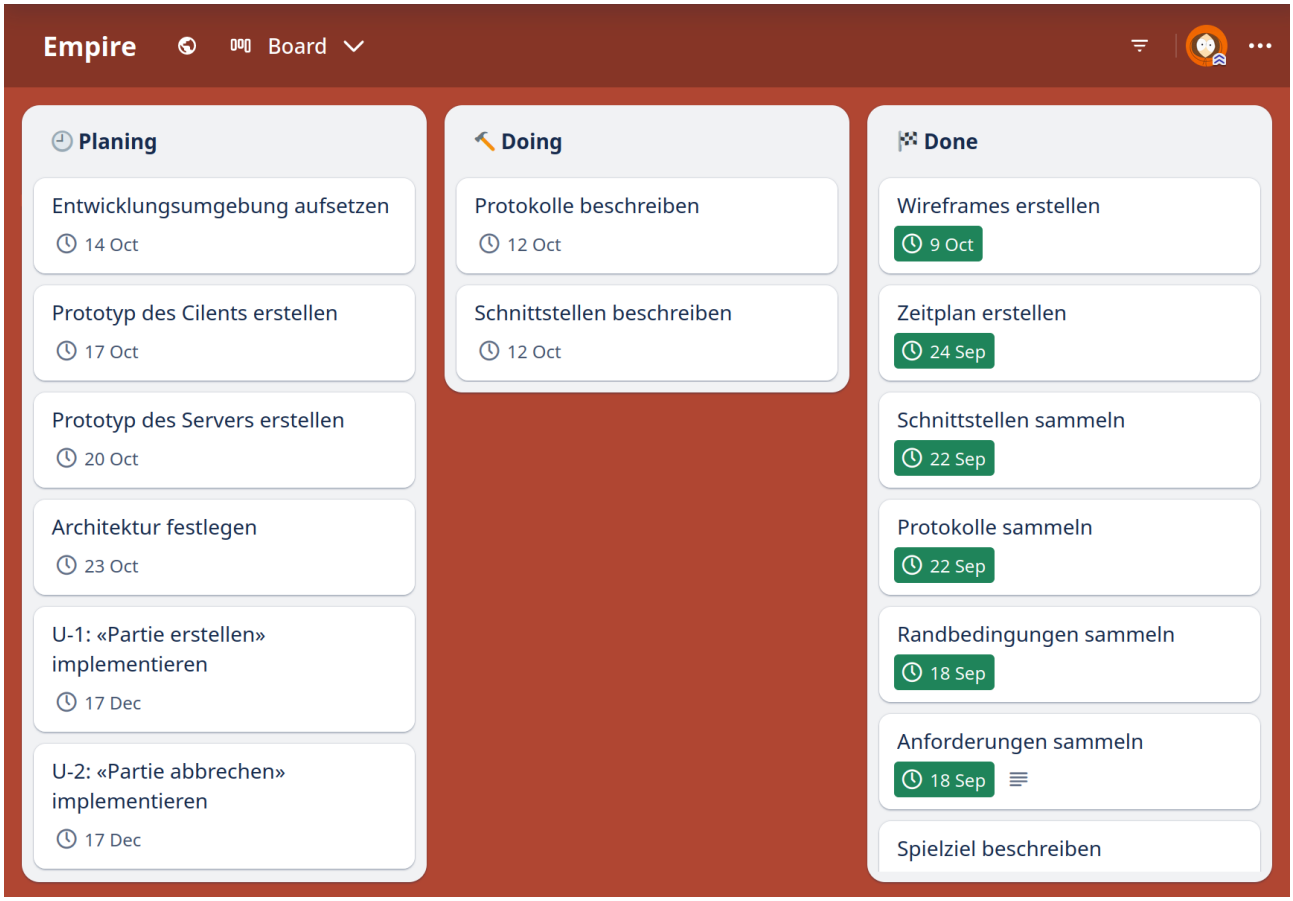

Abbildung 7: Trello-Board

5 Architektur

Die Architektur verfolgt einen pragmatischen Ansatz. Sie ist gut genug, um sowohl die funktionalen Anforderungen als auch die nichtfunktionalen Anforderungen zu erfüllen. Sie ist aber auch einfach genug, um die Projektziele innerhalb des Zeitbudgets zu erreichen.

5.1 Architekturziele

Neustart des Backends (A-1) Muss – Das Backend kann zu einem beliebigen Zeitpunkt gestoppt und erneut gestartet werden. Dabei dürfen aktive Transaktionen verloren gehen (z. B. ein während des Neustarts getätigter Spielzug). Zuvor abgeschlossene Transaktionen sind aber dauerhaft gespeichert und gehen nicht verloren (z. B. eine während des Neustarts aktive Partie).

Skalierung des Backends (A-2) Kann – Das Backend kann horizontal skaliert werden. Bei zunehmender Last können zusätzliche Instanzen gestartet werden; d. h. die Last wird auf mehrere Backend-Prozesse verteilt. Bei abnehmender Last können nicht mehr benötigte Instanzen gestoppt werden.

5.2 Architekturbeschreibung

Die Architekturübersicht (Abbildung 8) zeigt die wichtigsten Bausteine des Systems:

- Die **HTML5-Applikation** läuft im Browser des Spielers und greift über das Internet auf das Backend zu. Diese Applikation ist mit CSS, HTML und *JavaScript* implementiert.

  Über den *Apache2*-Dienst werden **statische Inhalte** geladen; z. B. Bilder, CSS-Dateien, HTML-Dateien und *JavaScript*-Dateien. Über den *Node.js*-Dienst werden **dynamische Inhalte** in beide Richtungen übertragen; z. B. Chat-Nachrichten, Highscores, Karteninhalt, Spielstände und Spielzüge.

- Der **Apache2-Dienst** ist ein regulärer Webserver, der statische Inhalte aus dem Dateisystem bezieht und über eine HTTP-Schnittstelle ausliefert. Dieser Dienst ist ein reiner Dateiserver und implementiert keine Spiellogik.

- Der ***Node.js*-Dienst** ist der eigentliche Applikationsserver, der dynamische Inhalte über eine WebSocket-Schnittstelle entgegennimmt, verarbeitet und ausliefert. Dieser Dienst implementiert somit die Spiellogik.

- Über den ***Redis*-Dienst** werden langlebige Daten persistiert; d. h. diejenigen Daten, die einen Neustart des Backend überleben müssen. Der *Node.js*-Dienst kann langlebige Daten abonnieren und wird in der Folge bei jeder Änderung benachrichtigt.

Sowohl das Frontend als auch das Backend verwalten möglichst wenig Zustand. Das Frontend kann seinen Zustand jederzeit vom Backend laden. Das Backend kann seinen Zustand wiederum aus der Persistenzschicht beziehen.

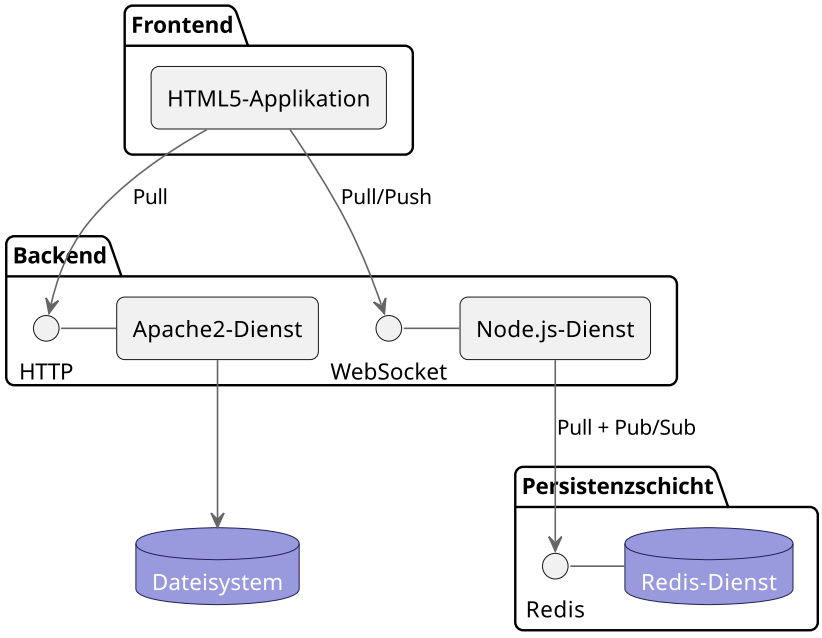

Abbildung 8: Architekturübersicht

6 Protokolle und Schnittstellen

Sowohl zwischen den Benutzern und dem System als auch zwischen den Modulen des Systems sowie zwischen den Komponenten des Systems gibt es eine Vielzahl von Schnittstellen. Die wichtigsten Schnittstellen sind:

1. **Benutzeroberfläche** – Die Schnittstelle zwischen einem Benutzer und dem System ist eine web-basierte Benutzeroberfläche. Der Zugriff erfolgt mit einem Browser über das Internet.
2. **HTTP-Schnittstelle** – Damit der Browser eines Benutzers die Benutzeroberfläche laden kann, müssen die benötigten Dateien auf einem Web-Server liegen, der über das Internet erreichbar ist.
3. **WebSocket-Schnittstelle** – Die Spiellogik wird zentral vom Backend gesteuert. Das Frontend ist lediglich für die Benutzerinteraktion verantwortlich. Die beiden Module kommunizieren mit dem Spielprotokoll über eine WebSocket-Schnittstelle.
4. ***Redis*-Schnittstelle** – Das Backend speichert die Spielzustände in einer *Redis*-Datenbank, damit sie bei einem Neustart des Backends nicht verloren gehen.

6.1 Benutzeroberfläche

Anstelle einer formalen Spezifikation wurden **interaktive Wireframes** der Benutzeroberfläche erstellt (Abbildungen 9 bis 19). Die Wireframes sind online verfügbar:

https://www.inverardi.ch/bsc-inf-webe/wireframes/

Die Wireframes sollen die wichtigsten Abläufe und Bedienelemente skizzieren. Verschiedene Bildschirmformate und Bildschirmgrössen werden bereits unterstützt; z. B. Smartphones, Tablets, Laptops und Desktops. Die abstrakte Optik ist eine bewusste Entscheidung, um das visuelle Design des Spiels nicht einzuschränken.

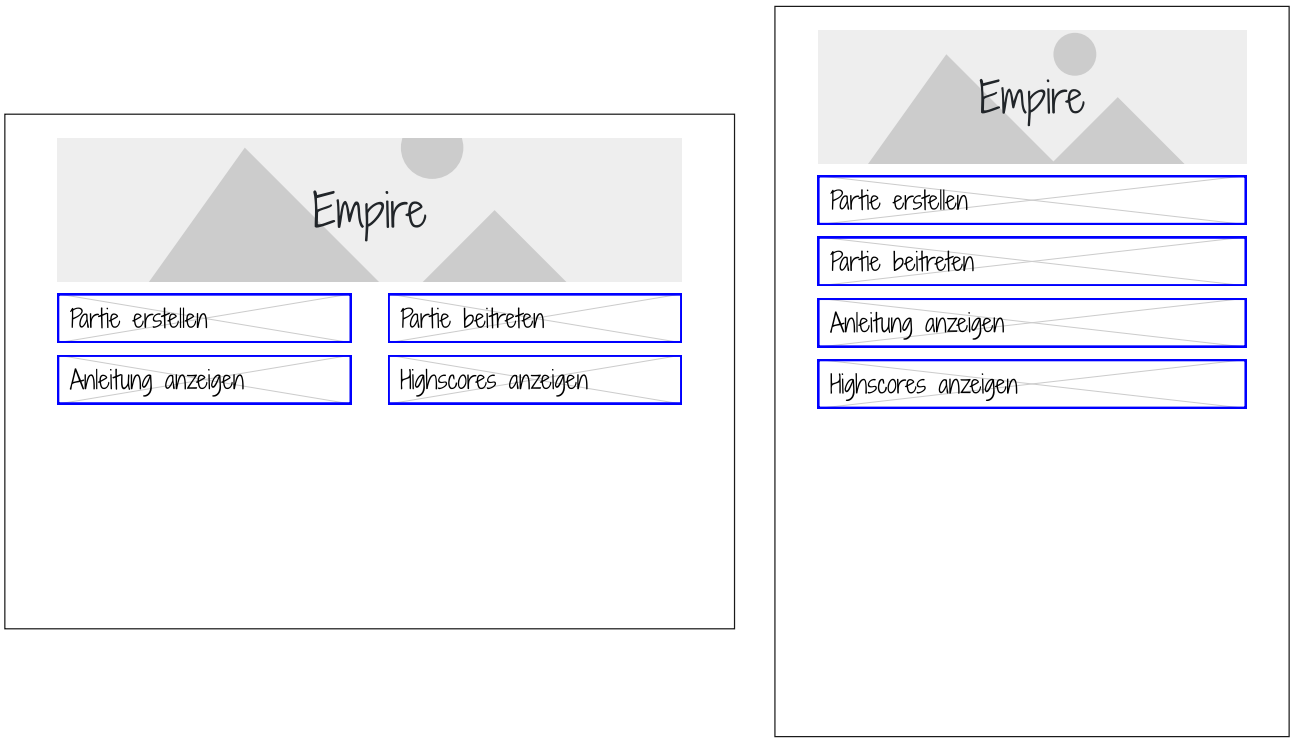

Abbildung 9: Wireframes – Startseite (W-1)

Abbildung 10: Wireframes – Partie erstellen, Schritt 1 (W-2)

Abbildung 11: Wireframes – Partie erstellen, Schritt 2 (W-3)

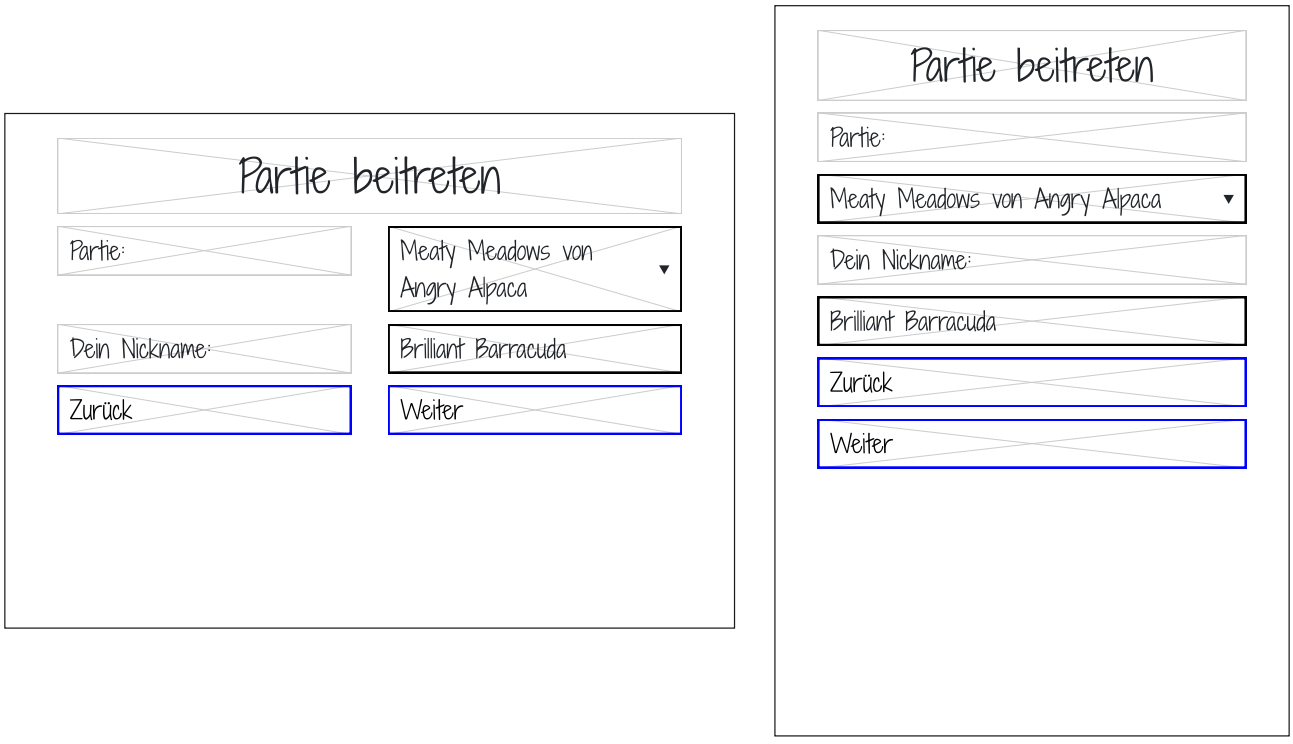

Abbildung 12: Wireframes – Partie beitreten, Schritt 1 (W-4)

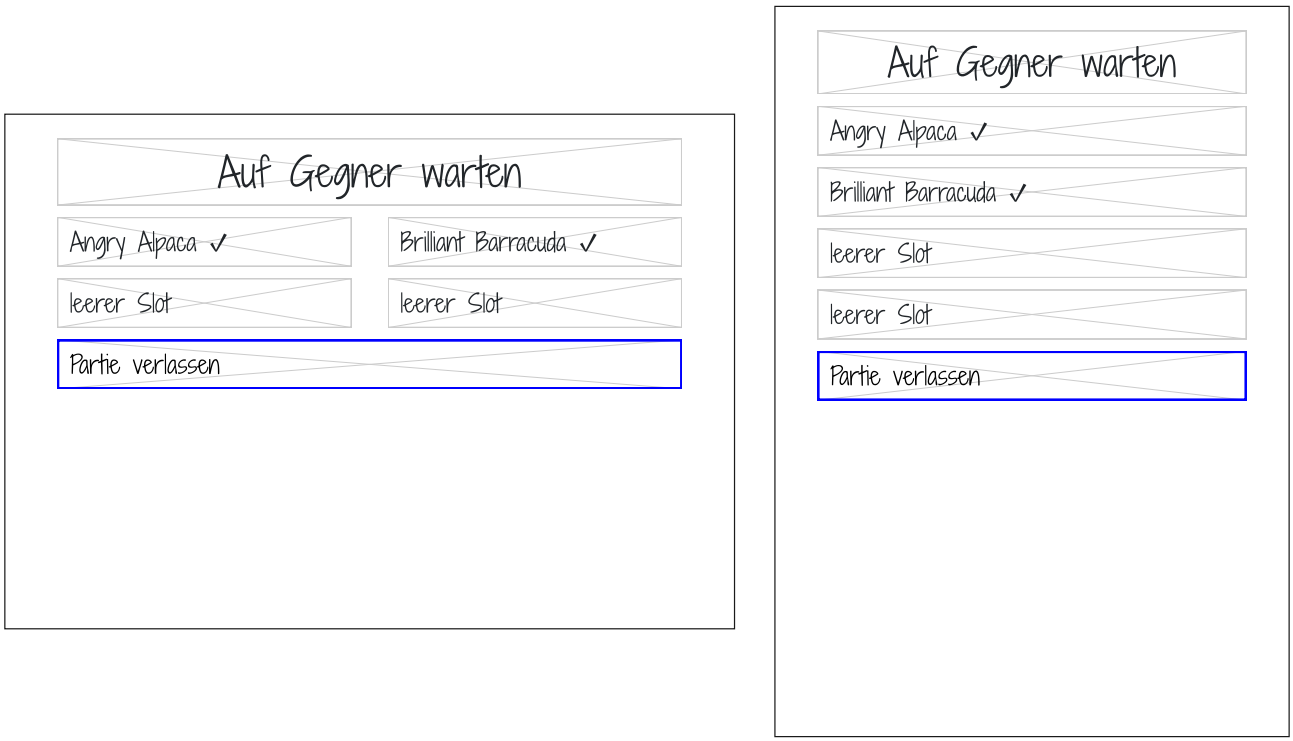

Abbildung 13: Wireframes – Partie beitreten, Schritt 2 (W-5)

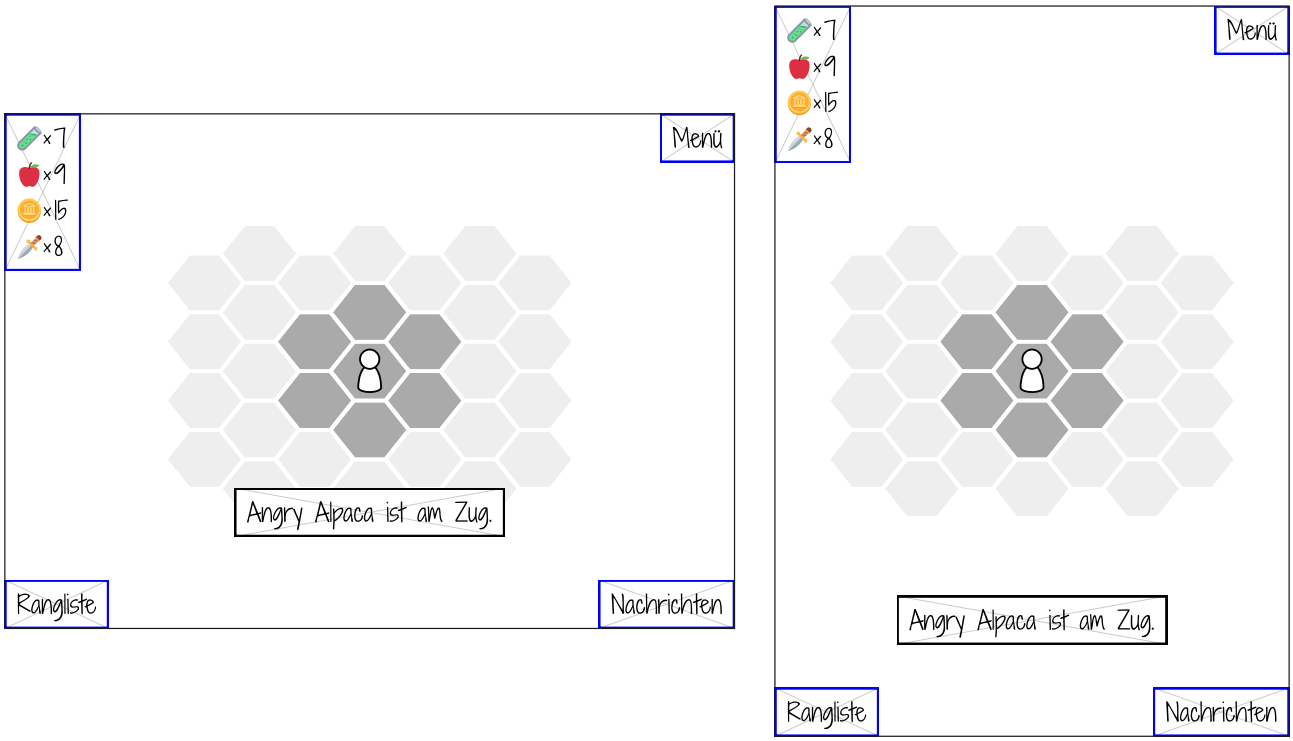

Abbildung 14: Wireframes – Karte (W-6)

Abbildung 15: Wireframes – Karte mit Aktionen (W-7)

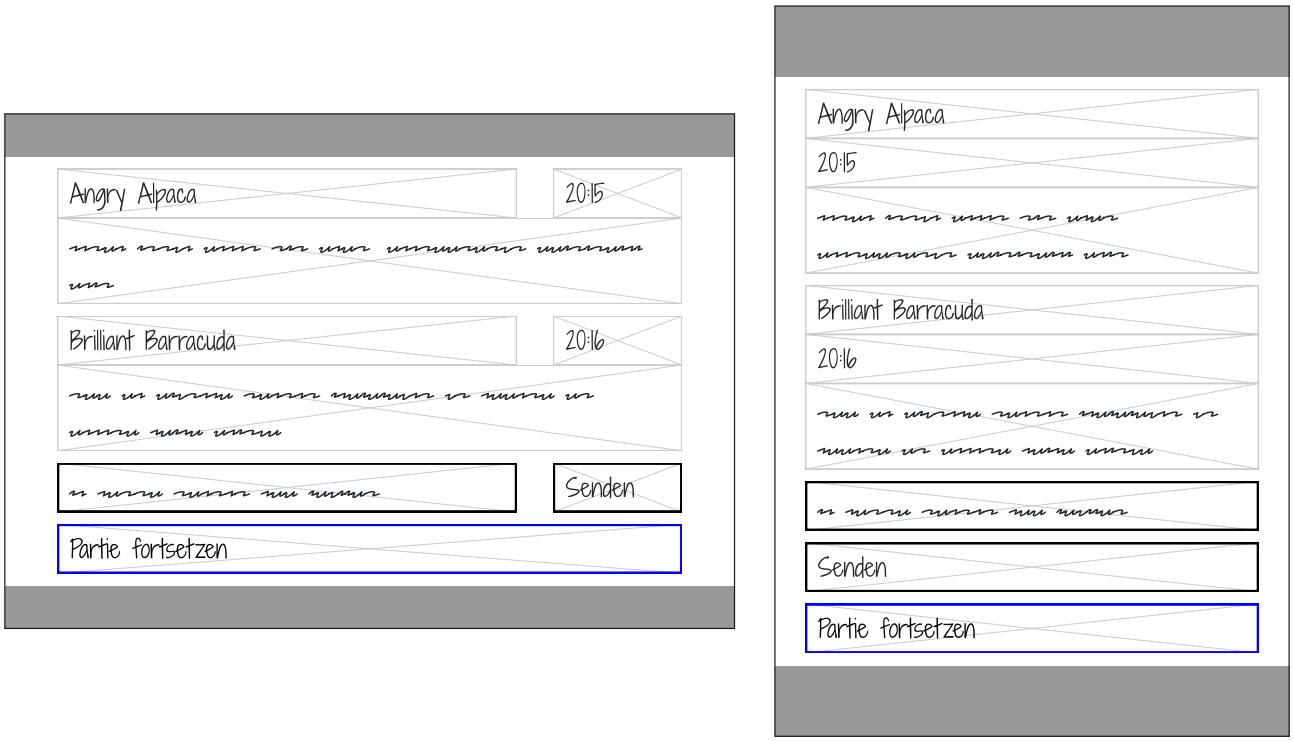

Abbildung 16: Wireframes – Karte mit Chat (W-8)

Abbildung 17: Wireframes – Karte mit Menü (W-9)

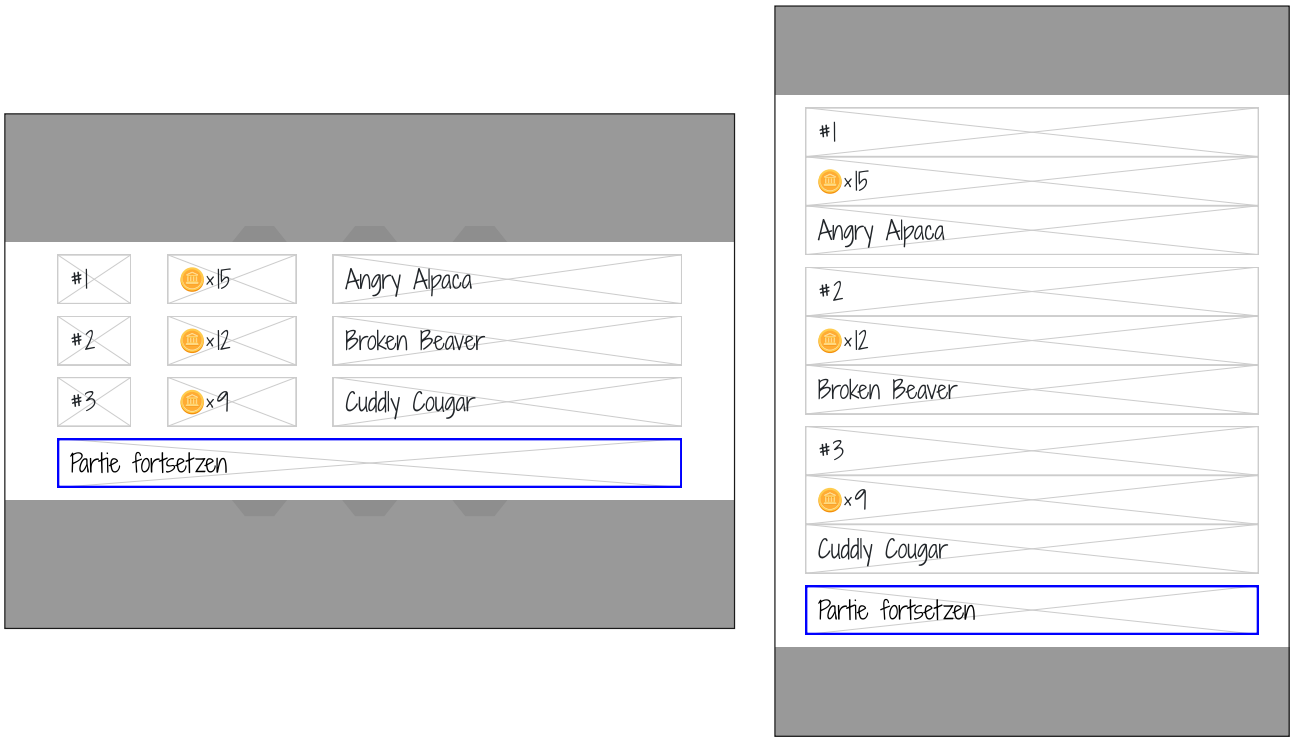

Abbildung 18: Wireframes – Karte mit Rangliste (W-10)

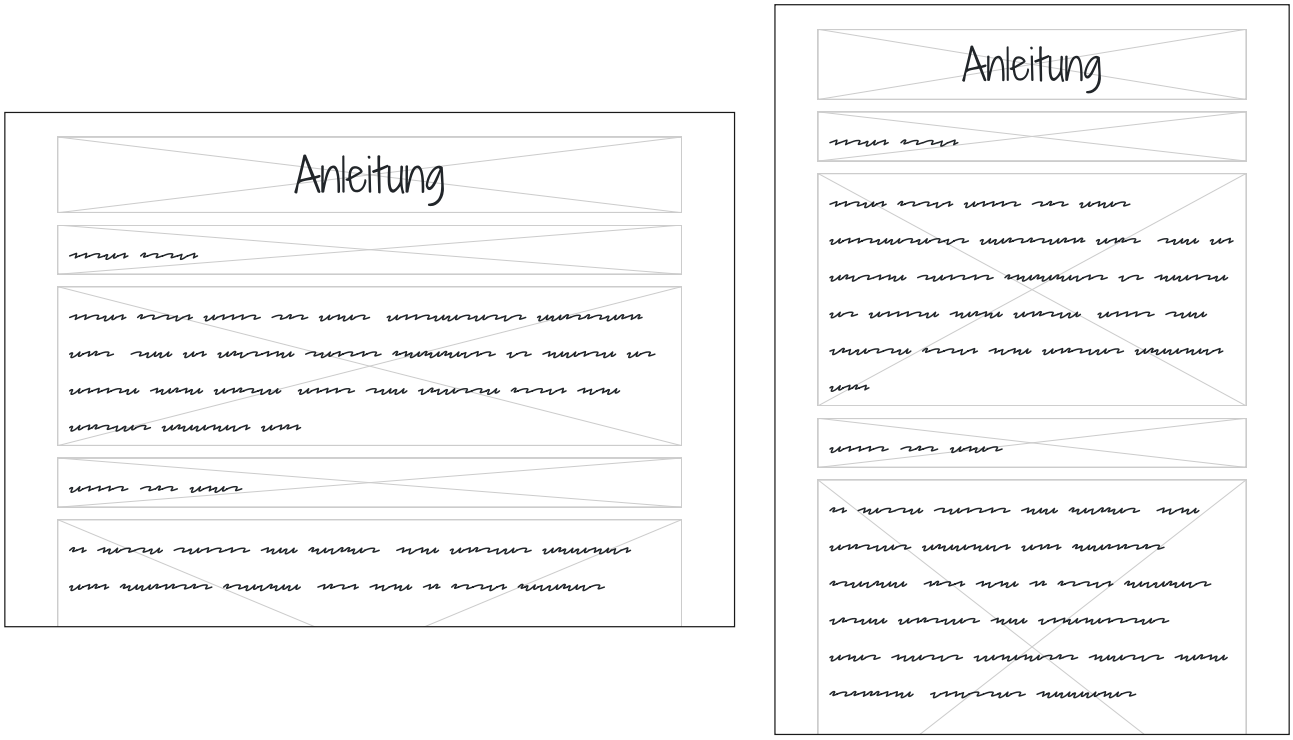

Abbildung 19: Wireframes – Anleitung (W-11)

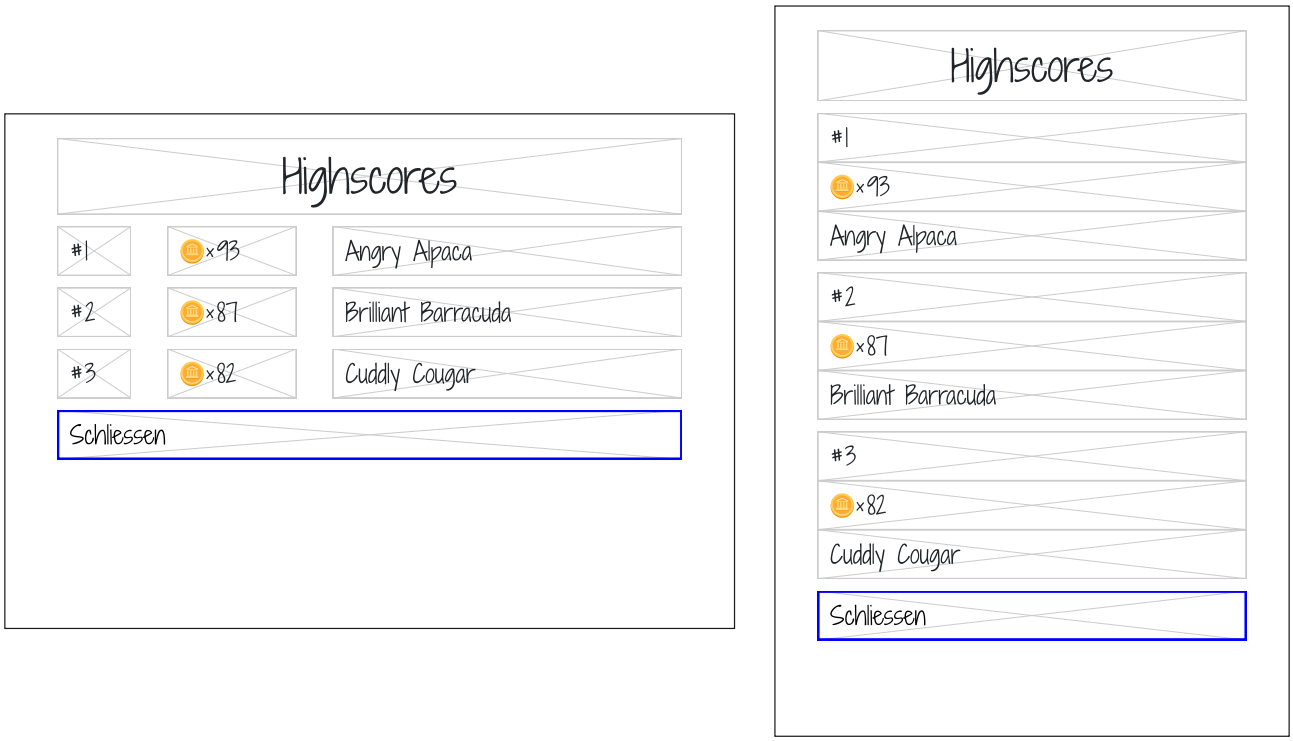

Abbildung 20: Wireframes – Highscores (W-12)

6.2 Spielprotokoll

Das Spielprotokoll umfasst Funktionen, um Partien zu verwalten, Spielzüge auszuführen, Chat-Nachrichten auszutauschen und die Highscores anzuzeigen (Tabelle 22). In den folgenden Abschnitten wird die Funktionsweise des Protokolls anhand konkreter Beispiele beschrieben.

Funktion	**Beispiel**
Partie erstellen	Abschnitt 6.2.1
Partie beitreten	Abschnitt 6.2.2
Spielzug ausführen	Abschnitt 6.2.3
Chat-Nachricht senden	Abschnitt 6.2.4
Partie verlassen	Abschnitt 6.2.5
Highscores anzeigen	Abschnitt 6.2.6

Tabelle 22: Funktionsumfang des Spielprotokolls

6.2.1 Partie erstellen

Dieser Abschnitt beschreibt, wie der Spieler «Angry Alpaca» eine neue Partie erstellt, auf einen Gegenspieler wartet und die Partie startet (Abbildung 21).

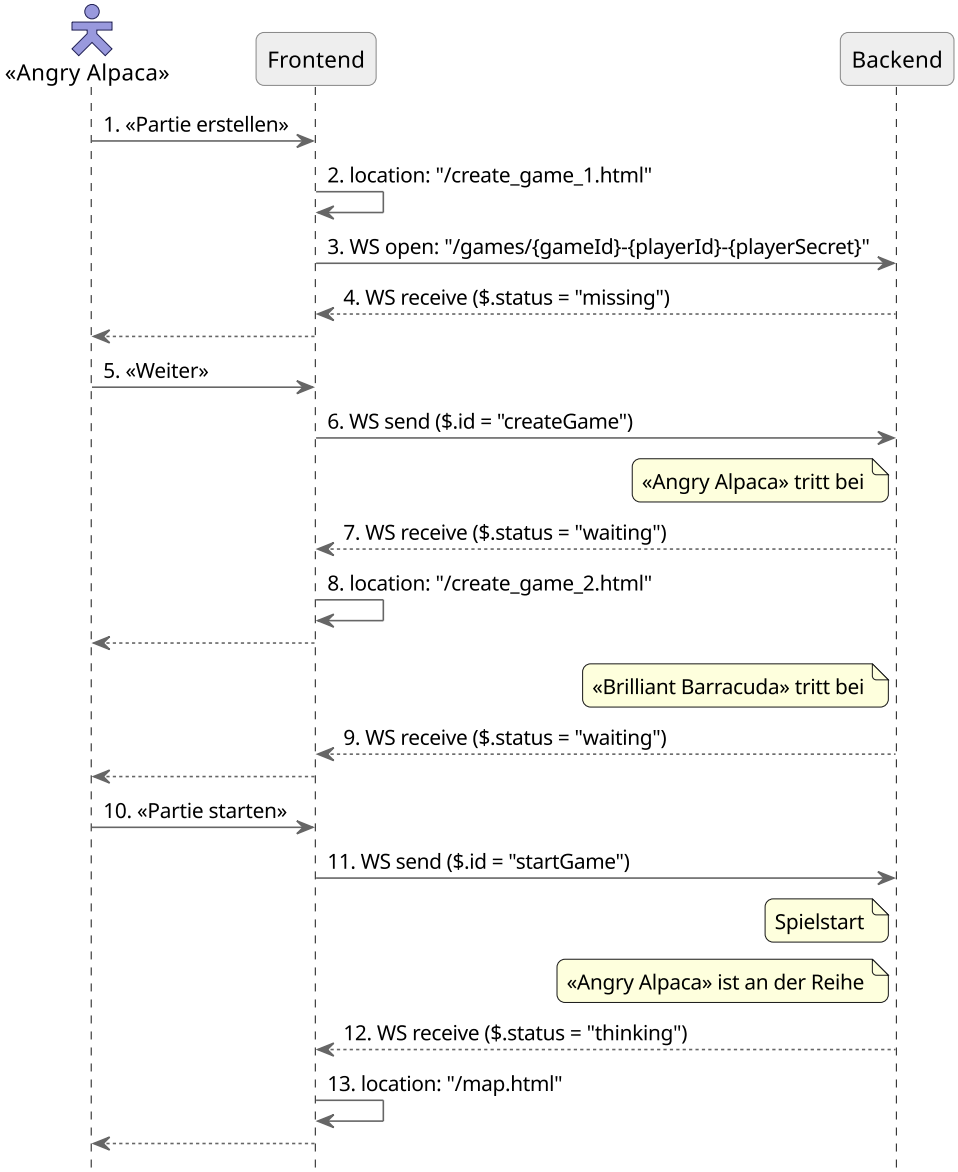

Abbildung 21: Ablauf – Partie erstellen

1. Der Spieler «Angry Alpaca» klickt auf «Partie erstellen».
2. Die Adresszeile im Browser wechselt auf:

   ```
   /create_game_1.html
   ```

3. Das Frontend öffnet eine WebSocket-Verbindung zu:

   ```
   /games/c1a39696-85a27a9f-cc699821b844a543
   ```

   Der Wert `c1a39696` ist die `gameId`, `85a27a9f` ist die `playerId` und `cc699821b844a543` ist das `playerSecret`. Die drei Werte werden zufällig gewählt.

Das Frontend speichert die Werte im Session-Storage-Bereich des Browsers, damit der Spielkontext bei einem Reload der Web-Seite nicht verloren geht. Das Backend verwendet die Werte, um das Spiel und den Spieler zu identifizieren bzw. den Spieler zu authentifizieren.

4. Das Frontend empfängt eine WebSocket-Nachricht vom Backend:

```
{
  "status": "missing"
}
```

   Der Spielstatus (`$.status = "missing"`) zeigt an, dass die Partie noch nicht existiert.

5. Der Spieler «Angry Alpaca» klickt auf «Weiter».

6. Das Frontend sendet eine WebSocket-Nachricht an das Backend:

```
{
  "id": "createGame",
  "map": { "id": 0 },
  "player": { "name": "Angry Alpaca" }
}
```

   Aufgrund der verlangten Aktion (`$.id = "createGame"`) erstellt das Backend die Partie.

7. Das Frontend empfängt eine WebSocket-Nachricht vom Backend:

```
{
  "map": { ··· },
  "players": [
    {
      "id": "85a27a9f",
      "name": "Angry Alpaca",
      "role": "master",
      "status": "alive"
    }
  ],
  "status": "waiting"
}
```

   Der neue Spielstatus (`$.status = "waiting"`) zeigt an, dass nun auf Gegenspieler gewartet wird. Das Frontend nutzt den Inhalt der WebSocket-Nachricht, um die belegten Slots und die freien Slots darzustellen bzw. die Darstellung zu aktualisieren.

8. Die Adresszeile im Browser wechselt auf:

```
/create_game_2.html
```

9. Das Frontend empfängt eine WebSocket-Nachricht vom Backend:

```
{
  "id": "c1a39696",
  "map": { ··· },
  "players": [
    {
      "id": "85a27a9f",
      "name": "Angry Alpaca",
      ...
    }, {
      "id": "2127b1ce",
      "name": "Brilliant Barracuda",
      ...
    }
  ],
  "status": "waiting"
}
```

   Der Grund für die Nachricht ist, dass der Gegenspieler «Brilliant Barracuda» der Partie beigetreten ist.

10. Der Spieler «Angry Alpaca» klickt auf «Partie starten».

11. Das Frontend sendet eine WebSocket-Nachricht an das Backend:

```
{
  "id": "startGame"
}
```

   Aufgrund der verlangten Aktion (`$.id = "startGame"`) startet das Backend die Partie.

12. Alle mit der Partie verbundenen Frontends empfangen eine WebSocket-Nachricht vom Backend:

```
{
  "id": "c1a39696",
  "map": {
    "id": 0,
    "tiles": [ ··· ]
  },
  "messages": [ ··· ],
  "players": [ ··· ],
  "resources": [ ··· ],
  "status": "thinking",
  "structures": [ ··· ],
  "turn": {
    "number": 1,
    "player": "85a27a9f"
  }
}
```

Der neue Spielstatus (`$.status = "thinking"`) zeigt an, dass die Partie nun gestartet wurde und der Spieler «Angry Alpaca» (`$.turn.player = "85a27a9f"`) an der Reihe ist. Das Frontend nutzt den Inhalt der WebSocket-Nachricht, um die Karte darzustellen bzw. die Darstellung zu aktualisieren.

13. Die Adresszeile im Browser wechselt auf:

```
/map.html
```

6.2.2 Partie beitreten

Dieser Abschnitt beschreibt, wie der Spieler «Brilliant Barracuda» der zuvor von «Angry Alpaca» erstellten Partie beitritt (Abbildung 22).

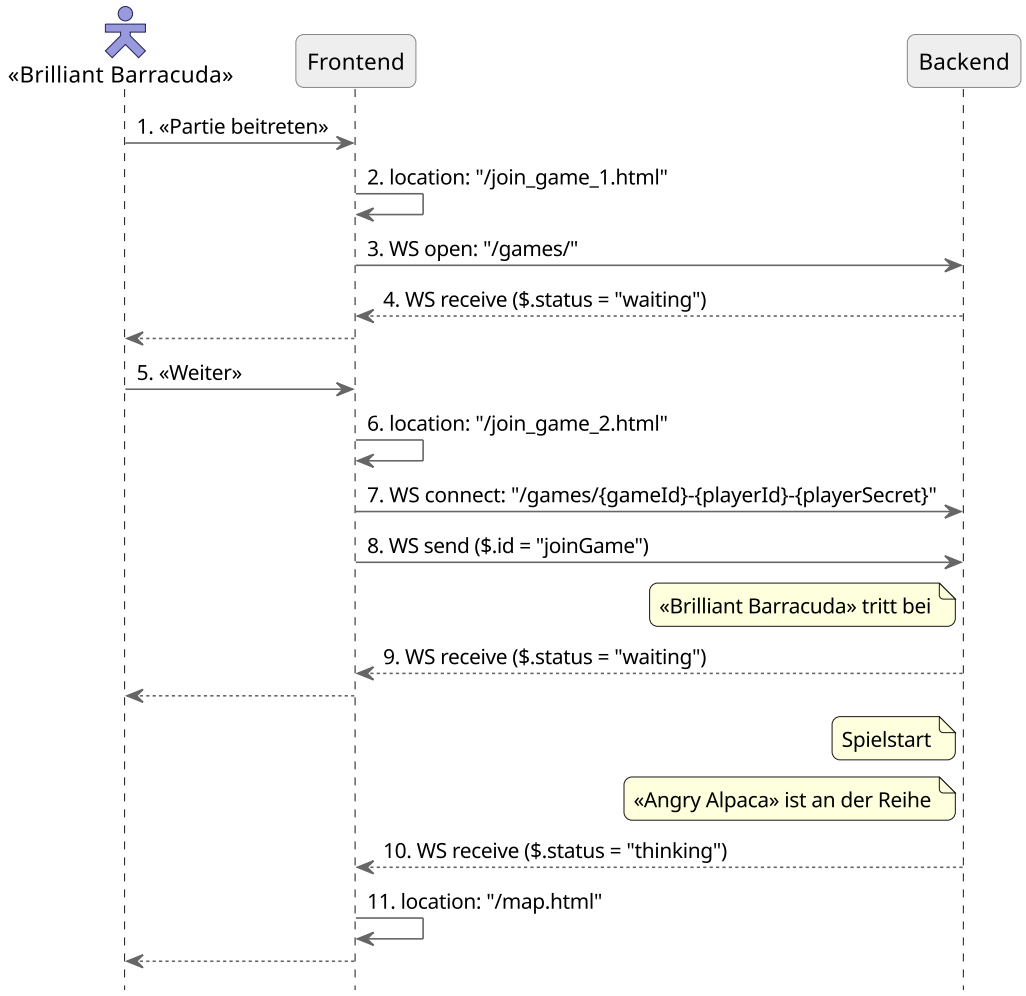

Abbildung 22: Ablauf – Partie beitreten

1. Der Spieler «Brilliant Barracuda» klickt auf «Partie beitreten».
2. Die Adresszeile im Browser wechselt auf:

   ```
   /join_game_1.html
   ```

3. Das Frontend öffnet eine WebSocket-Verbindung zu:

   ```
   /games/
   ```

4. Das Frontend empfängt eine WebSocket-Nachricht vom Backend:

```
[
  {
    "id": "c1a39696",
    "map": { "id": 0 },
    "status": "waiting"
  },
  ...
]
```

   Der Spielstatus (`$.status = "waiting"`) zeigt an, dass auf Gegenspieler gewartet wird. Das Frontend nutzt den Inhalt der WebSocket-Nachricht, um die Spielliste darzustellen bzw. die Darstellung zu aktualisieren.

5. Der Spieler «Brilliant Barracuda» klickt auf «Weiter».

6. Die Adresszeile im Browser wechselt auf:

```
/join_game_2.html
```

7. Das Frontend öffnet eine WebSocket-Verbindung zu:

```
/games/c1a39696-2127b1ce-3e26eff6ea6da984
```

   Der Wert `c1a39696` ist die `gameId`, `2127b1ce` ist die `playerId` und `3e26eff6ea6da984` ist das `playerSecret`. Die *gameId* stammt aus der letzten WebSocket-Nachricht. Die `playerId` und das `playerSecret` werden zufällig gewählt.

   Das Frontend speichert die Werte im Session-Storage-Bereich des Browsers, damit der Spielkontext bei einem Reload der Web-Seite nicht verloren geht. Das Backend verwendet die Werte, um das Spiel und den Spieler zu identifizieren bzw. den Spieler zu authentifizieren.

8. Das Frontend sendet eine WebSocket-Nachricht an das Backend:

```
{
  "id": "joinGame",
  "player": { "name": "Brilliant Barracuda" }
}
```

   Aufgrund der verlangten Aktion (`$.id = "joinGame"`) fügt das Backend den Spieler zur Partie hinzu.

9. Alle mit der Partie verbundenen Frontends empfangen eine WebSocket-Nachricht vom Backend:

```
{
  "id": "c1a39696",
  "map": { "id": 0 },
  "players": [
    {
      "id": "85a27a9f",
      "name": "Angry Alpaca",
      ...
    }, {
      "id": "2127b1ce",
      "name": "Brilliant Barracuda",
      "role": "participant",
      ...
    }
  ],
  "status": "waiting"
}
```

Das Frontend nutzt den Inhalt der WebSocket-Nachricht, um die belegten Slots und die freien Slots darzustellen bzw. die Darstellung zu aktualisieren.

10. Alle mit der Partie verbundenen Frontends empfangen eine WebSocket-Nachricht vom Backend:

```
{
  "id": "c1a39696",
  "map": {
    "id": 0,
    "tiles": [ ··· ]
  },
  "messages": [ ··· ],
  "players": [ ··· ],
  "resources": [ ··· ],
  "status": "thinking",
  "structures": [ ··· ],
  "turn": {
    "number": 1,
    "player": "85a27a9f"
  }
}
```

    Der neue Spielstatus (`$.status = "thinking"`) zeigt an, dass die Partie nun gestartet wurde und der Spieler «Angry Alpaca» (`$.turn.player = "85a27a9f"`) an der Reihe ist. Das Frontend nutzt den Inhalt der WebSocket-Nachricht, um die Karte darzustellen bzw. die Darstellung zu aktualisieren.

11. Die Adresszeile im Browser wechselt auf:

```
/map.html
```

6.2.3 Spielzug ausführen

Dieser Abschnitt beschreibt, wie der Spieler «Angry Alpaca» einen Spielzug ausführt und seine Spielfigur um ein Feld nach Nord-Osten bewegt (Abbildung 23).

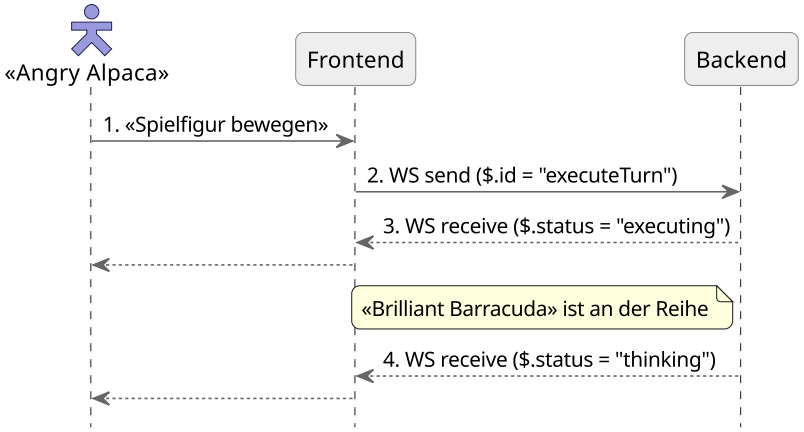

Abbildung 23: Ablauf – Spielzug ausführen

1. Der Spieler «Angry Alpaca» bewegt seine Spielfigur um ein Feld nach Nord-Osten.
2. Das Frontend sendet eine WebSocket-Nachricht an das Backend:

```
{
  "id": "executeTurn",
  "turn": {
    "direction": "northEast",
    "type": "move"
  }
}
```

   Aufgrund der verlangten Aktion (`$.id = "executeTurn"`) führt das Backend den Spielzug (`$.turn.type = "move"`) aus.

3. Alle mit der Partie verbundenen Frontends empfangen eine WebSocket-Nachricht vom Backend:


```
{
  "id": "c1a39696",
  "map": { ··· },
  "messages": [ ··· ],
  "players": [ ··· ],
  "resources": [ ··· ],
  "status": "executing",
  "structures": [ ··· ],
  "turn": {
    "direction": "northEast",
    "type": "move"
  }
}
```


   Der Spielstatus (`$.status = "executing"`) zeigt an, dass gerade ein Spielzug ausgeführt wird. Das Frontend nutzt den Inhalt der WebSocket-Nachricht, um eine Animation des Zugs anzuzeigen.

4. Alle mit der Partie verbundenen Frontends empfangen eine WebSocket-Nachricht vom Backend:


```
{
  "id": "c1a39696",
  "map": { ··· },
  "messages": [ ··· ],
  "players": [ ··· ],
  "resources": [ ··· ],
  "status": "thinking",
  "structures": [ ··· ],
  "turn": {
    "number": 2,
    "player": "2127b1ce"
  }
}
```


   Der neue Spielstatus (`$.status = "thinking"`) zeigt an, dass nun der Spieler «Brilliant Barracuda» (`$.turn.player = "2127b1ce"`) an der Reihe ist. Das Frontend nutzt den Inhalt der WebSocket-Nachricht, um die Karte darzustellen bzw. die Ansicht zu aktualisieren.

6.2.4 Chat-Nachricht senden

Dieser Abschnitt beschreibt, wie der Spieler «Angry Alpaca» eine Chat-Nachricht an seine Gegenspieler sendet (Abbildung 24).

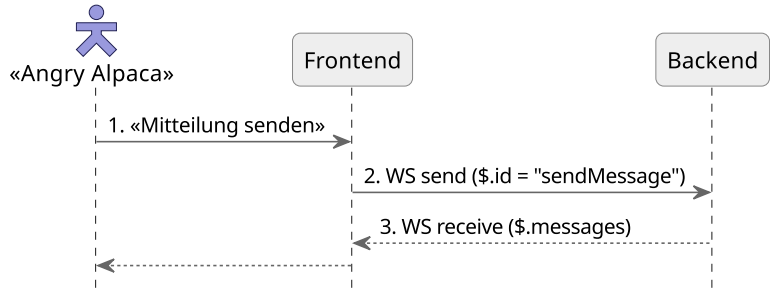

Abbildung 24: Ablauf – Chat-Nachricht senden

1. Der Spieler «Angry Alpaca» sendet eine Chat-Nachricht an seine Gegenspieler.
2. Das Frontend sendet eine WebSocket-Nachricht an das Backend:

   ```
   {
     "id": "sendMessage",
     "text": "Bow before your new master!"
   }
   ```

   Aufgrund der verlangten Aktion (`$.id = "sendMessage"`) übermittelt das Backend die Chat-Nachricht (`$.text`) an alle Spieler.

3. Alle mit der Partie verbundenen Frontends empfangen eine WebSocket-Nachricht vom Backend:

```
{
  "id": "c1a39696",
  "map": { ··· },
  "messages": [
    {
      "player": "85a27a9f",
      "text": "Bow before your new master!"
    },
    ...
  ],
  "players": [ ··· ],
  "resources": [ ··· ],
  "status": ···,
  "structures": [ ··· ],
  "turn": { ··· }
}
```

   Das Frontend nutzt den Inhalt der WebSocket-Nachricht, um die Chat-Nachrichten darzustellen bzw. die Darstellung zu aktualisieren.

6.2.5 Partie verlassen

Dieser Abschnitt beschreibt, wie der Spieler «Angry Alpaca» die Partie verlässt (Abbildung 25).

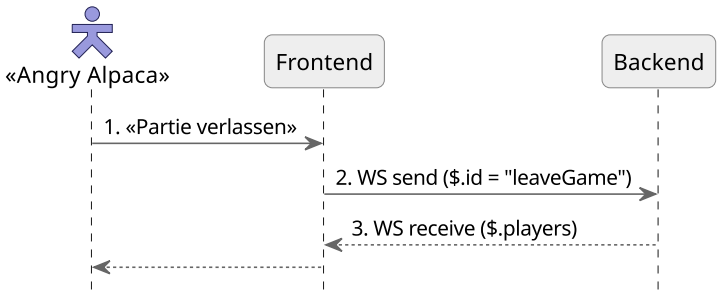

Abbildung 25: Ablauf – Partie verlassen

1. Der Spieler «Angry Alpaca» klickt auf «Partie verlassen».

2. Das Frontend sendet eine WebSocket-Nachricht an das Backend:

```
{
  "id": "leaveGame"
}
```

   Aufgrund der verlangten Aktion (`$.id = "leaveGame"`) entfernt das Backend den Spieler «Angry Alpaca» aus der Partie.

3. Alle mit der Partie verbundenen Frontends empfangen eine WebSocket-Nachricht vom Backend:

```
{
  "id": "c1a39696",
  "map": { ··· },
  "messages": [ ··· ],
  "players": [
    {
      "id": "85a27a9f",
      "name": "Angry Alpaca",
      "status": "left",
      ···
    },
    ···
  ],
  "resources": [ ··· ],
  "status": ···,
  "structures": [ ··· ],
  "turn": { ··· }
}
```

   Der neue Spielerstatus (`$.players[0].status = "left"`) zeigt an, dass der Spieler die Partie verlassen hat. Das Frontend nutzt den Inhalt der WebSocket-Nachricht, um die Darstellung zu aktualisieren.

6.2.6 Highscores anzeigen

Dieser Abschnitt beschreibt, wie sich der Spieler «Angry Alpaca» die Highscores anzeigen lässt (Abbildung 26).

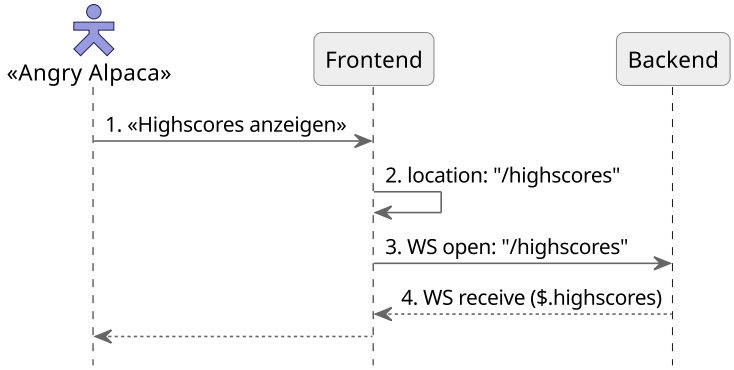

Abbildung 26: Ablauf – Highscores anzeigen

1. Der Spieler «Angry Alpaca» klickt auf «Highscores anzeigen».
2. Die Adresszeile im Browser wechselt auf:

   ```
   /highscores.html
   ```

3. Das Frontend öffnet eine WebSocket-Verbindung zu:

   ```
   /highscores/
   ```

4. Das Frontend empfängt eine WebSocket-Nachricht vom Backend:

   ```
   [
     {
       "player": "Angry Alpaca",
       "rank": 1,
       "score": 93
     }, {
       "player": "Brilliant Barracuda",
       "rank": 2,
       "score": 87
     },
     ...
   ]
   ```

   Das Frontend nutzt den Inhalt der WebSocket-Nachricht, um die Highscores darzustellen bzw. die Darstellung zu aktualisieren.

6.3 Spielzustand

Während einer Partie verwendet das Backend das Spielprotokoll, um den jeweils aktuellen Spielzustand an die Frontend-Instanzen zu übertragen. Die Spielzustände werden im JSON-Format (Listing 1) über eine WebSocket-Verbindung übertragen.

Dabei erhält jeder Spieler eine personalisierte Ansicht und sieht keine Informationen, die nicht unbedingt benötigt werden; z. B. sieht jeder Spieler nur denjenigen Bereich der Karte, den er bereits erkundet hat.

Listing 1: Spielzustand

```
{
  "id": "c1a39696",
  "map": { ··· },
  "messages": [ ··· ],
  "players": [ ··· ],
  "resources": [ ··· ],
  "status": "thinking"
  "structures": [ ··· ],
  "turn": {
    "number": 1,
    "player": "85a27a9f"
  },
  "turns": [ ··· ]
}
```

6.3.1 Ast `$.map`

Die Kartenfelder im Ast `$.map.tiles` (Listing 2) werden als Strings übertragen, damit das Format einerseits kompakt und andererseits menschenlesbar sind. Dabei wird für jedes Feld der Anfangsbuchstabe der Feldart übermittelt.

Listing 2: Spielzustand → `map`

```
...
"map": {
  "id": 0,
  "tiles": [
    " w w w w w w w",
    "w g g h w f w ",
    ...
  ]
},
...
```

6.3.2 Ast `$.messages`

Der Ast `$.map.messages` (Listing 3) enthält die Eigenschaften aller Chat-Nachrichten, die sich die Spieler während einer Partie gegenseitig senden.

Listing 3: Spielzustand → `messages`

```
...
"messages": [{
  "player": "85a27a9f",
  "text": "Bow before your new master!"
}, {
  "player": "2127b1ce",
  "text": "I don't think so."
}],
...
```

6.3.3 Ast `$.players`

Der Ast `$.map.players` (Listing 4) enthält die Eigenschaften aller Spieler, die an einer Partie teilnehmen.

Listing 4: Spielzustand → `players`

```
...
"players": [{
  "health": 10,
  "id": "85a27a9f",
  "inventory": {
    "food": 9,
    "gold": 15,
    "research": 7,
    "weaponry": 8
  },
  "name": "Angry Alpaca",
  "position": [11, 19],
  "role": "master",
  "status": "alive",
}, {
  "name": "Brilliant Barracuda",
  ...
}],
...
```

6.3.4 Ast `$.resources`

Der Ast `$.resources` (Listing 5) enthält die Eigenschaften aller Ressourcen, die bei Spielbeginn zufällig auf der Karte verteilt werden.

Listing 5: Spielzustand → `resources`

```
...
"resources": [{
  "age": 3,
  "position": [14, 8],
  "type": "food"
}, {
  "type": "gold",
  ...
}],
...
```

6.3.5 Ast `$.structures`

Der Ast `$.structures` (Listing 6) enthält die Eigenschaften aller Dörfer, Städte, Metropolen und Fabriken, die im Spielverlauf auf der Karte errichtet werden.

Listing 6: Spielzustand → `structures`

```
...
"structures": [{
  "position": [17, 11],
  "type": "city"
}, {
  "type": "factory",
  ...
}],
...
```

6.3.6 Ast `$.turns`

Der Ast `$.map.turns` (Listing 7) enthält die Eigenschaften aller Spielzüge, die der aktuelle Spieler ausführen kann.

Listing 7: Spielzustand → `turns`

```
...
"turns": [{
  "direction": "east",
  "positionFrom": [11, 19],
  "positionTo": [13, 19],
  "type": "move"
}, {
  "type": "attack"
  ...
}],
...
```

6.4 Aktionen

Während einer Partie verwendet das Frontend das Spielprotokoll, um von den Spielern ausgeführte Aktionen an das Backend zu übertragen. Die Aktionen werden im JSON-Format über eine WebSockets-Verbindung übertragen.

6.4.1 Aktion `createGame`

Die Aktion `createGame` wird dann vom Frontend an das Backend übermittelt, wenn der Spielleiter eine neue Partie erstellt (Listing 8).

Listing 8: Aktion `createGame`

```
{
    "id": "createGame",
    "map": { "id": 0 },
    "player": { "name": "Angry Alpaca" }
}
```

6.4.2 Aktion `joinGame`

Die Aktion `joinGame` wird dann vom Frontend an das Backend übermittelt, wenn ein neuer Spielteilnehmer einer Partie beitritt (Listing 9).

Listing 9: Aktion `joinGame`

```
{
  "id": "joinGame",
  "player": { "name": "Brilliant Barracuda" }
}
```

6.4.3 Aktion `leaveGame`

Die Aktion `leaveGame` wird dann vom Frontend an das Backend übermittelt, wenn ein Spielteilnehmer eine Partie verlässt (Listing 10).

Listing 10: Aktion `leaveGame`

```
{
  "id": "leaveGame"
}
```

6.4.4 Aktion `startGame`

Die Aktion `startGame` wird dann vom Frontend an das Backend übermittelt, wenn der Spielleiter eine Partie startet (Listing 11).

Listing 11: Aktion `startGame`

```
{
  "id": "startGame"
}
```

6.4.5 Aktion `executeTurn`

Die Aktion executeTurn wird dann als vom Frontend an das Backend übermittelt, wenn ein Spielteilnehmer einen Spielzug ausführt (Listing 12).

Listing 12: Aktion `executeTurn`

```
{
    "id": "executeTurn",
    "turn": {
        "direction": "northEast",
        "type": "move"
    }
}
```

6.4.6 Aktion `sendMessage`

Die Aktion sendMessage wird dann als vom Frontend an das Backend übermittelt, wenn ein Spielteilnehmer eine Chat-Nachricht ans seine Gegenspieler sendet (Listing 13).

Listing 13: Aktion `sendMessage`

```
{
    "id": "sendMessage",
    "text": "Bow before your new master!"
}
```

6.4.7 Aktion `abortGame`

Die Aktion abortGame wird dann als vom Frontend an das Backend übermittelt, wenn der Spielleiter eine Partie abbricht (Listing 14).

Listing 14: Aktion `abortGame`

```
{
  "id": "abortGame"
}
```

7 Technische Umsetzung

7.1 Programmiersprache und Laufzeitumgebung

Im Einklang mit den Randbedingungen werden sowohl das Backend als als auch das Frontend in *JavaScript* programmiert. Als Laufzeitumgebung für das Backend kommt *Node.js*[1] zum Einsatz.

7.2 Bibliotheken und Rahmenwerke

Der Einsatz von Bibliotheken und Rahmenwerken ist bei der Entwicklung von Web-Anwendungen mit zahlreichen Vorteilen verbunden. Ein kompletter Verzicht ist in den wenigsten Fällen sinnvoll, kann aber dann Sinn machen, wenn es darum geht, eine Technologie von Grund auf zu erlernen oder das vorhandene Wissen zu vertiefen.

7.2.1 Backend

Bei der Umsetzung des Backends werden keine Rahmenwerke verwendet, aber die folgenden Bibliotheken kommen zum Einsatz:

- **`redis`**[2] – *Redis* integrieren, persistente Daten laden und speichern
- **`ws`**[3] – WebSockets integrieren, Nachrichten empfangen und senden

7.2.2 Frontend

Bei der Umsetzung des Frontends werden keine Rahmenwerke verwendet, aber die folgenden Bibliotheken kommen zum Einsatz:

- **`bootstrap`**[4] – Responsiveness ermöglichen, verschiedene Bildschirmformate und Bildschirmgrössen unterstützen

[1] https://nodejs.org/ (besucht am 30.09.2023)
[2] https://www.npmjs.com/package/redis (besucht am 30.09.2023)
[3] https://www.npmjs.com/package/ws (besucht am 30.09.2023)
[4] https://getbootstrap.com/ (besucht am 30.09.2023)

8 Journal

In diesem Kapitel werden pro Iteration die erledigten Arbeiten beschrieben und die wichtigsten Ergebnisse, Problemstellungen und Problemlösungen zusammengefasst.

8.1 Erste Iteration (I-1)

- Ich habe die **Dokumentation** mit *LaTeX* aufgesetzt. Die Struktur wurde aus früheren Semesterarbeiten der Module SWEM, SWEA und PA5 übernommen und für das Modul WebE angepasst. Fiel mir das Schrieben technischer Dokumente zu Beginn meines Studiums noch schwer, bin ich mittlerweile routinierter und effizienter.
- Ich habe die **Spielregeln** und das **Spielziel** beschrieben. Sicherlich werden bei der Entwicklung der Prototypen und bei der finalen Implementation noch Fragen auftauchen. Insbesondere beim Spielziel und den Highscores erwarte ich noch Änderungen, sobald man erste Partien spielen und sich einen konkreten Eindruck verschaffen kann.
- Ich habe die **Anforderungen** und die **Randbedingungen** beschrieben. Für die wichtigsten Anforderungen habe ich Anwendungsfälle mit entsprechenden Diagrammen erstellt. Die funktionalen Anforderungen und die nichtfunktionalen Anforderungen habe ich in Textform verfasst.

  Alle Anforderungen und alle Randbedingungen haben ein eindeutiges **Kürzel**, damit ich sie später referenzieren kann; z. B. beim Erstellen von automatischen und manuellen Testfällen.
- Ich habe die **Protokolle** und die **Schnittstellen** beschrieben. Die Beschreibung des Spielprotokolls ist bereits ziemlich detailliert; d. h. ich habe schon Vorarbeit für die zweite Iteration (I-2) geleistet.
- Ich habe die **Wireframes** erstellt; d. h. ich habe wiederum Vorarbeit für die zweite Iteration (I-2) geleistet. Die Werkzeuge, die ich bisher für die Erstellung von Wireframes genutzt habe, konnten mich nicht überzeugen. Deshalb habe ich die Wireframes direkt mit HTML und CSS erstellt. Der initiale Aufwand ist zwar höher, aber ich kann die Wireframes bei der Entwicklung des Prototypen wiederverwenden.
- Ich habe den **Zeitplan** erstellt. Mein Projekt ist zwar umfangreich, sollte aber bis zur fünften Präsenzveranstaltung abgeschlossen sein. Ob ich eine fünfte Iteration (I-5) anhänge, um die Dokumentation und die Software zu polieren, weiss ich noch nicht bzw. mache ich vom Arbeitsaufwand der anderen Module in diesem Semester abhängig.
- Der aktuelle **Meilenstein** (M-1) wurde vollumfänglich erreicht.

8.2 Zweite Iteration (I-2)

- Ich habe das **Feedback** zur ersten Iteration (I-1) eingearbeitet. Die Handhabung der `gameId`, der `playerId` und des `playerSecrets` ist noch etwas wackelig. Hier werde ich mir vor der finalen Implementation noch Gedanken machen und entsprechende Anpassungen vornehmen.
- Ich habe die **Entwicklungsumgebung** unter *Linux* aufgesetzt. Statt wie bisher mit den Werkzeugen von *IntelliJ* zu arbeiten, werde ich erste Erfahrungen mit *Visual Studio Code* von *Microsoft* sammeln.

  Mit *JavaScript* im Browser habe ich in der Vergangenheit bereits gearbeitet. *JavaScript* auf dem Server bzw. *Node.js* ist hingegen Neuland für mich. Im Moment verschlingen selbst einfache Aufgaben viel Zeit.
- Ich habe **Prototypen** des Clients bzw. des Frontends und des Servers bzw. des Backends erstellt. Die Prototypen sind gerade gut genug, um den Happy Path von der Spielerstellung bis zur Kartenansicht mit mehreren Mitspielern und verschiedenen Geräten durchzuspielen. Fehlerfälle und sonstige Spezialfälle werden noch nicht behandelt.
- Die **Wireframes** aus der ersten Iteration (I-1) waren eine gute Ausgangslage, um den Prototyp des Frontends zu erstellen. Bei der Kartenansicht verwende ich vorerst «geclipte» HTML-Elemente, um die Hexagone darzustellen. Ich bin nicht sicher, ob dies der endgültige Lösungsansatz sein wird.
- Ich habe die **Architektur** beschrieben. Die grundsätzliche Idee besteht darin, möglichst wenig Zustand im Backend und im Frontend zu verwalten. Um das zu erreichen, kann das Frontend seinen Zustand jederzeit vom Backend laden. Das Backend kann seinen Zustand wiederum aus der Persistenzschicht beziehen. Mit dem gewählten Ansatz sollte es möglich sein, mehrere Instanzen des Backends laufen zu lassen und die Last mit einem Load Balancer zu verteilen.
- Ich habe die Nachrichtenformate für den **Spielzustand** und die verschiedenen **Aktionen** beschrieben. Insbesondere die Datenstruktur, die eine laufende Partie beschreibt, ist recht umfangreich. Sollten derartige Strukturen zu Performanzproblemen führen, werde ich über inkrementelle Aktualisierungen oder ähnliche Optimierungen nachdenken müssen.
- Der aktuelle **Meilenstein** (M-2) wurde vollumfänglich erreicht.

8.3 Dritte Iteration (I-3)

- Ich habe das **Feedback** zur zweiten Iteration (I-2) eingearbeitet. Die Forderung nach horizontaler Skalierbarkeit wurde von «soll» auf «kann» zurückgestuft. Die Verzeichnisstruktur des Codes wurde angepasst.

- Die **Implementation** des Frontends und des Backends ist soweit fortgeschritten, dass nun mehrere Spieler an einer Partie teilnehmen und erste Aktionen ausführen können:
  - Spielfiguren bewegen und Ressourcen sammeln (U-9)
  - Chat-Nachricht senden (U-15)
  - Chat-Nachrichten anzeigen (U-16)
  - Rangliste anzeigen (U-17)
  - Partie verlassen (U-18)
- In der nächsten Iteration (I-4) werde ich diverse **Redundanzen** im Code beseitigen. Inbesondere die doppelten Codestellen im Backend und im Frontend beeinträchtigt die Wartbarkeit und sollten beseitigt bzw. wiederverwendet werden.
- Der **Umfang** des Projekts ist hoch und aus Zeitgründen sind einige Codestellen «mit heissen Nadeln gestrickt». Ich überlege mir, eine zusätzliche Iteration (I-5) anzuhängen, um die betroffenen Codestellen aufzuräumen, die Sicherheit zu verbessern, die Stabilität zu verbessern und automatisierte Testfälle zu erstellen.
- Ich musste das **Verhalten** beim Einsammeln von Ressourcen überdenken, weil sich das bisherige Verhalten beim Spielen des Prototyps falsch anfühlte. Neu wird das Alter der Ressourcen nach jedem Spielzug statt nach jeder Spielrunde erhöht. Dafür wachsen Ressourcen erst nach 9 Zügen statt nach 5 Zügen nach.
- Ich konnte das Spiel bereits auf verschiedenen **Geräten** ausprobieren; z. B. Laptop, Mobiltelefon, Tablet. Positiv aufgefallen ist, dass die Bedienbarkeit auch auf kleinen Bildschirmen gut ist. Negativ aufgefallen ist, dass die Animationen auf älteren Geräten nicht flüssig sind.
- Ich habe eine **Kurzbeschreibung** im *Git*-Repository abgelegt. Darin sind alle Abhängigkeiten und Befehle beschrieben, die benötigt werden, um das Spiel sowohl im Entwicklungsmodus als auch im Produktivmodus zu starten.
- Der aktuelle **Meilenstein** (M-2) wurde vollumfänglich erreicht.

Literatur

[1] H. Balzert. *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. 3. Aufl. Heidelberg: Springer Spektrum, 2011.

[2] I. Fetai und P. Lauwiner. *WebE: Informationen zur Projektarbeit*. 2023. URL: https://moodle.ffhs.ch/mod/url/view.php?id=4420710 (besucht am 06. 09. 2023).

[3] I. Fetai und P. Lauwiner. *WebE: Modulplan*. 2023. URL: https://moodle.ffhs.ch/mod/url/view.php?id=4420710 (besucht am 06. 09. 2023).

[4] K. Pohl und C. Rupp. *Basiswissen Requirements Engineering*. 4. Aufl. Heidelberg: dpunkt.verlag, 2015.