Bomb.io
Studie

Auftraggeber	Thomas Fehr
Projektleiter	Mario Aeberhard
Autor	Nicolas Ammeter, Mario Aeberhard, Loic Tobler
Klassifizierung	Nicht klassifiziert~~, Intern, Vertraulich, GEHEIM~~
Status	In Arbeit~~, Genehmigt, zur Prüfung~~

Änderungsverzeichnis

Datum	Version	Änderung	Autor
11.09.2025	V1.0	Initiale Erstellung	Nicolas Ammeter, Mario Aeberhard, Loic Tobler
14.09.2025	V2.0	Finale Änderung	Nicolas Ammeter, Mario Aeberhard, Loic Tobler

Inhaltsverzeichnis

1 Situationsanalyse

1.1 Ausgangslage

Heute gibt es sehr viele Multiplayer-Spiele, aber die meisten sind komplex, teuer oder benötigen leistungsstarke Computer. Bekannte Spiele wie Fortnite, Valorant oder Call of Duty erfordern häufig viel Speicherplatz, minutenlange Start und ladezeiten, mindestens 8 GB Ram, leistungsstarke Prozessoren oder auch einen eigenen Launcher.

Gerade in der Schule oder auf älteren Geräten ist das ein Problem. Viele Schüler möchten spontan ein Spiel starten, ohne vorher große Dateien herunterzuladen oder komplizierte Setups durchzuführen. Bomb.io macht das möglich. Mit dem eingeben der URL im Browser ist man direkt im Spiel, ohen noch etwas herunterladen zu müssen.

1.2 Stärken der IST-Situation

Es gibt bereits eine große Auswahl an Spielen und viele unterschiedliche Genres, aus denen man sich Inspiration holen kann. Außerdem sind moderne Spiele technisch sehr weit entwickelt und bieten hohe Grafikqualität sowie flüssige Animationen, was uns zeigt, welchen Standard Spieler heute erwarten.

Ein weiterer Vorteil ist, dass Multiplayer-Spiele allgemein sehr bekannt und beliebt sind. Viele Spielerinnen und Spieler kennen die grundlegenden Prinzipien bereits, was bedeutet, dass wir weniger erklären müssen. Zusätzlich hat fast jedes Gerät heutzutage einen aktuellen Webbrowser, wodurch die Browsertechnologie weit verbreitet ist. Das erleichtert uns die Entwicklung, da wir mit Bomb.io eine Plattform nutzen können, die auf nahezu allen Geräten verfügbar ist.

1.3 Schwächen der IST-Situation

Nr.	Schwachpunkt	Beschreibung	Verbesserung durch Bomb.io
S1	Hohe Hardware-Anforderungen	Viele aktuelle Spiele benötigen leistungsstarke Computer oder Konsolen.	Läuft auch auf älteren Geräten, da es direkt im Browser läuft.
S2	Lange Downloads & Installationen	Spiele sind oft mehrere GB groß und brauchen lange Installationen.	Keine Installation, kein Download – einfach URL öffnen und spielen.
S3	Komplizierte Anmeldung & Launcher	Häufig ist ein Account oder ein zusätzlicher Launcher nötig.	Kein Account, keine Registrierung, kein Launcher notwendig.
S4	Wenige moderne Multiplayer-Browsergames	Viele Browsergames sind veraltet, grafisch schwach oder ohne Echtzeit-Multiplayer.	Bomb.io bietet modernes Design und Echtzeit-Multiplayer im Browser.
S5	Ungeeignet für kurze Spielrunden	Bekannte Spiele sind oft zeitintensiv und kompliziert.	Kurze, schnelle Runden, perfekt für Pausen oder spontane Spiele.

2 Ziele

2.1 Systemziele

Nr.	Kategorie	Beschreibung (Ziel)	Messgröße	Priorität	Bezug zu Schwachpunkt
1	Benutzerfreundlichkeit	Bomb.io muss ohne Installation direkt im Browser spielbar sein.	Zugriff über URL ohne Download oder Setup	M	S2, S3
2	Kompatibilität	Das Spiel muss auf aktuellen Browsern und schwächeren Geräten laufen.	Läuft fehlerfrei auf min. 90 % getesteten Geräten	M	S1
3	Multiplayer-Echtzeit	Bomb.io bietet Multiplayer für mindestens 4 Spieler pro Runde.	Erfolgreicher Test mit 4–6 verbundenen Clients	M	S4
4	Kurze Spielrunden	Eine Runde darf maximal 3 Minuten dauern.	Rundenzeit < 3 Min. in Testläufen	1	S5
5	Design & Spielspaß	Ein modernes, übersichtliches Interface mit einfacher Steuerung.	Spielerfeedback ≥ 80 % positiv (Befragung)	2	S4, S5
6	Zugriffsgeschwindigkeit	Der Start des Spiels muss in unter 10 Sekunden möglich sein.	Ladezeit ≤ 1 Sek. auf Schul-PCs und Laptops	1	S2

Legende: Priorität: M=Muss /1=hoch, 2=mittel, 3=tief

2.2 Vorgehensziele

Nr.	Kategorie	Beschreibung	Messgröße	Priorität
1	Projektorganisation	Jeder im Team hat klare Rollen und Aufgaben.	Aufgabenübersicht erstellt, Zuständigkeiten geklärt	M
2	Qualität der Umsetzung	Eine fehlerfreie Testinstallation wird durchgeführt.	Erfolgreicher Testlauf ohne Abstürze	1
3	Kommunikation	Regelmäßige Team-Updates und Absprachen.	Wöchentliche Protokolle der Meetings	2
4	Dokumentation	Projektdokumentation wird vollständig erstellt.	Abgabe der Dokumentation bis 09.01.2026	M

Rahmenbedingungen

- Projektmethode: HERMES-gibb
- Zeitrahmen: 22.08.2025 – 22.01.2026
- Team: Loic Tobler, Mario Aeberhard, Nicolas Ammeter
- Ressourcen: VS Code, GitHub
- Technologie: Node.js, Express.js, Socket.IO, Three.js, HTML5
- Kosten: Keine zusätzlichen Ausgaben, Nutzung der vorhandenen Infrastruktur
- Plattform: Browserbasiert, optimiert für Desktop-PCs
- Kommunikation: Discord / Microsoft Teams

Abgrenzung

- Keine Mobile-App
- Kein Punktesystem
- Keine Erstellung von Konten
- Keine komplexen 3D Modelle

3 Nutzen

Das Projekt Bomb.io bietet uns als Entwicklerteam und auch den späteren Spielern einen klaren Nutzen. Da es sich um ein Schulprojekt handelt, steht für uns in erster Linie der Lernerfolg im Vordergrund. Wir sammeln wertvolle praktische Erfahrungen in der Programmierung und lernen dabei den Umgang mit modernen Technologien wie Node.js, Express.js, Socket.IO und Three.js für die 3D-Entwicklung. Außerdem üben wir, wie man im Team arbeitet, Aufgaben sinnvoll verteilt und gemeinsam eine funktionierende Lösung erarbeitet. Ein weiterer wichtiger Lerneffekt besteht darin, zu verstehen, wie Echtzeitkommunikation über WebSockets funktioniert und wie man ein synchrones Multiplayer-Spiel entwickelt.

4 Liste der Stakeholder

Stakeholder	Rolle / Funktion	Interesse am Projekt	Einfluss auf Projekt
Thomas Fehr	Lehrperson, Auftraggeber	Begleitet und bewertet das Projekt, erwartet vollständige Abgabe	Hoch
Mario Aeberhard	Projektleiter, Entwickler	Erfolgreiche Leitung und Umsetzung des Spiels	Hoch
Loic Tobler	Entwickler, Tester	Mitgestaltung und Umsetzung der Software	Hoch
Nicolas Ammeter	Entwickler, Dokumentation	Beteiligung an Entwicklung und Erstellung der Unterlagen	hoch
Mitschüler/innen	Testspieler, Nutzer	Wollen ein funktionierendes, spassiges Spiel ausprobieren	Niedrig

5 Anforderungen

Nr.	Anforderung	Beschreibung	Deckt Ziel(e)
A1	Browserzugriff	Spiel läuft direkt im Browser, keine Installation nötig.	Z1, Z6
A2	Kompatibilität	Funktioniert auf aktuellen Browsern und Schul-PCs.	Z2
A3	Multiplayer	Mindestens 4 Spieler können gleichzeitig spielen.	Z3
A4	Kurze Runden	Eine Runde dauert maximal 3 Minuten.	Z4
A5	Echtzeit-Synchronisation	Bewegungen und Bombenübergaben laufen ohne spürbare Verzögerung.	Z3
A6	Benutzerfreundlichkeit	Einfache Steuerung und übersichtliches HUD.	Z5
A7	3D-Grafik	Verwendung von Three.js für einfache, performante 3D-Modelle.	Z2, Z5

6 Lösungsvarianten

6.1 Variantenübersicht

Variante	Beschreibung	Vorteile	Nachteile
V1	Node.js + Express.js + Socket.IO + Three.js – Alles in JavaScript/TypeScript, Server und Client in einer Sprache, 3D mit Three.js	Läuft direkt im Browser, keine Installation nötig- Nur eine Sprache nötig- Große Community, viele Tutorials- Gute Echtzeit-Performance	Erfordert gute JavaScript-Kenntnisse- Mehr Lernaufwand zu Beginn
V2	Unity mit WebGL-Export – Spiel in Unity entwickelt und als WebGL-Version in den Browser gebracht	Sehr gute 3D-Grafikqualität- Viele Assets verfügbar- Einfache Entwicklung mit visuellen Tools	Lange Ladezeiten im Browser- Hoher Ressourcenverbrauch- Komplexere Build-Prozesse
V3	Python-Server (Flask/FastAPI) + JavaScript-Client (Three.js) – Server in Python, Client in JS	Serverlogik in Python einfach umsetzbar- Gut, falls Team bereits Python-Erfahrung hat- Flexibel erweiterbar	Zwei Sprachen notwendig- Echtzeit-Multiplayer komplizierter- Weniger performant als Node.js

6.2 Beschreibung der Varianten

6.2.1 Lösungsvariante Node.js / Three.js

Struktur (grobe Architektur)

Das Spiel wird mit Node.js und Express.js als Server umgesetzt. Die Kommunikation zwischen den Spielern läuft über Socket.IO, die 3D-Grafik wird mit Three.js direkt im Browser dargestellt.

Schnittstellen

Es sind keine Schnittstellen zu anderen Systemen notwendig.

Abdeckung der Anforderungen

Alle Anforderungen können abgedeckt werden. Es gibt keine gravierenden Lücken, lediglich der Lernaufwand ist etwas höher.

Realisierbarkeitsbetrachtung

Sehr gut realisierbar, alle eingesetzten Tools sind frei verfügbar.

6.2.2 Lösungsvariante Unity / WebGL

Struktur (grobe Architektur)

Das Spiel wird mit der Unity Engine entwickelt und als WebGL-Version exportiert. Dadurch kann es im Browser gespielt werden.

Schnittstellen

Es sind keine Schnittstellen zu anderen Systemen notwendig.

Abdeckung der Anforderungen

Nicht vollständig abgedeckt sind die Anforderungen A2 (Ladezeit) und A3 (Kompatibilität). Unity-WebGL-Spiele brauchen oft lange zum Laden und laufen auf schwächeren PCs nicht zuverlässig.

Realisierbarkeitsbetrachtung

Technisch machbar, aber die Performance auf Schul-PCs könnte problematisch sein.

6.2.3 Lösungsvariante Python / JavaScript

Struktur (grobe Architektur)

Der Server wird in Python (z. B. Flask oder FastAPI) entwickelt, während die 3D-Darstellung im Browser mit JavaScript und Three.js erfolgt.

Schnittstellen

Es sind keine Schnittstellen zu anderen Systemen notwendig.

Abdeckung der Anforderungen

Problematisch ist vor allem die Anforderung A3/A5 (Echtzeit-Multiplayer), da Python für schnelle Echtzeitkommunikation weniger geeignet ist. Außerdem steigt die Komplexität, da zwei Programmiersprachen kombiniert werden müssen.

Realisierbarkeitsbetrachtung

Grundsätzlich möglich, aber für ein kleines Team aufwendiger und fehleranfälliger.

6.3 Schutzbedarfsanalyse (ISDS)

6.3.1 Informationssicherheit

Das Spiel muss jederzeit stabil laufen und für alle Spieler erreichbar sein. Bei der Node.js-Variante ist das am einfachsten, da WebSockets sehr zuverlässig sind. Unity-WebGL kann hier Probleme machen, wenn Ladezeiten zu lang sind oder das Spiel auf schwächeren PCs abstürzt. Bei der Python-Variante besteht ein höheres Risiko, dass Verzögerungen auftreten, was die Verfügbarkeit beeinflusst.

6.3.2 Authentizität

Wichtig ist, dass die Spielregeln korrekt eingehalten werden, damit niemand schummeln kann. In allen Varianten wird dies über einen zentralen Server gelöst, der prüft, wer die Bombe hat. Node.js eignet sich hier am besten, da es sehr gut für Echtzeitspiele optimiert ist.

6.3.3 Datenschutz

Es werden keine sensiblen Daten gespeichert. Spieler brauchen nur einen Nicknamen, es gibt keine Passwörter oder persönlichen Angaben. Damit entstehen kaum Datenschutzrisiken.

6.3.4 Verfügbarkeit

Das Spiel sollte jederzeit im Browser spielbar sein. Node.js bietet hier eine stabile Grundlage. Unity-WebGL kann durch lange Ladezeiten die Verfügbarkeit einschränken. Die Python-Lösung ist zwar nutzbar, aber aufgrund der geringeren Echtzeitfähigkeit weniger zuverlässig.

6.4 Mittelbedarf

6.4.1 Kosten (CHF)

Phase	Geplant
Initialisierung*	0
Konzept	0
Realisierung	0
Einführung	0
Total	0

*Vorleistung (IST)

6.4.2 Personalaufwand

Es ist vorgesehen, das Projekt gänzlich innerhalb des Moduls 306 abzuwickeln. Bis zur Präsentation stehen noch 15x4 Lektionen zur Verfügung oder 45 Personenstunden. Davon sind rund 8h für Theorie vorgesehen. Somit stehen für die Konzeption, Realisierung und Einführung rund 37 Personenstunden oder 4,5 Personentage zur Verfügung. Die Projektziele sind auf diese Ressourcenleistung abgestimmt.

Phase	Geplant (PT)
Initialisierung*	1
Konzept	1
Realisierung	2.5
Einführung	1
Total	5.5

*Vorleistung (IST)

6.4.3 Sachmittel

- Nutzung von vorhandenen Schul-PCs mit Internetzugang
- Entwicklungsumgebungen: Visual Studio Code, Node.js, GitHub
- Für die 3D-Grafik: Three.js (Open Source, kostenlos)
- Kommunikations- und Organisationstools: Microsoft Teams / Discord
- Alle eingesetzten Programme sind entweder Open Source oder bereits an der Schule vorhanden.
- Es werden keine zusätzlichen Geräte oder Software-Lizenzen benötigt.

6.4.4 Dienstleistungen

Es ist nicht vorgesehen, externe Dienstleistungen in Anspruch zu nehmen. Die gesamte Entwicklung, Tests und Präsentationen erfolgen im Team.

6.5 Wirtschaftlichkeit

Die externen Kosten für das Projekt Bomb.io betragen null Franken, da nur vorhandene Computer und kostenlose Software verwendet werden. Der gesamte Zeitaufwand kann im Rahmen des Moduls 306 geleistet werden.

Der Nutzen ist auf den ersten Blick vor allem privat: Wir lernen neue Technologien kennen, üben Teamarbeit und setzen ein komplettes Projekt um. Für Spieler ist der Nutzen, dass man ein kleines, leicht zugängliches Multiplayer-Spiel direkt im Browser spielen kann. Sollte die Umsetzung wie geplant gelingen, steigt der Nutzen deutlich, weil wir ein funktionierendes und unterhaltsames Produkt vorzeigen können.

Damit ist das Verhältnis von Kosten und Nutzen klar positiv: keine Ausgaben, dafür ein sichtbarer Lernerfolg und ein spielbares Ergebnis.

7 Bewertung der Varianten (Tabelle)

Kriterium	**Variante 1 Node.js / Three.js**		**Variante 2 Unity / WebGL**		**Variante 3 Python / JavaScript**	
Abdeckung der Anforderungen	++		+		o	
Realisierbarkeit, Risiken	++		-	Lange Ladezeiten	-	Echtzeit schwierig
Betrieb	++		o	Braucht einen starken Laptop	o	Höhere Komplexität
Wirtschaftlichkeit	++		o		o	Wissen muss grösstenteils erarbeitet werden
Gesamterscheinung	++		+	gute Grafik, aber Ladezeiten zu lang	-	gemischte Umsetzung mit 2 Sprachen
Gesamtbeurteilung	**++**		**o**		**-**	

Bewertungen:
++ erfüllt Kriterium optimal
+ erfüllt Kriterium gut
o Vor- und Nachteile halten sich die Waage
- erfüllt Kriterium schlecht
-- erfüllt Kriterium gar nicht

Aufgrund der in der Tabelle aufgeführten Stärken und Schwächen fällt die Wahl auf Variante 1 (Node.js / Three.js). Sie ist browserfreundlich, benötigt keine Installation, erfüllt alle wichtigen Anforderungen und ist im Zeitrahmen des Moduls 306 realisierbar.
Unity wäre zwar bei der Grafik stark, scheitert aber an den Ladezeiten und der Performance

auf Schul-PCs. Die Python-Lösung ist technisch möglich, aber für Echtzeit-Multiplayer weniger geeignet und durch die Kombination zweier Sprachen unnötig kompliziert.

8 Lösungsbeschreibung und Empfehlung

8.1 Abhängigkeiten

Es gibt keine weiteren Projekte oder Arbeiten, die direkt von Bomb.io abhängig sind. Das Spiel steht für sich allein und ist ein eigenständiges Projekt im Modul 306.

8.2 Auswirkungen

Wenn Bomb.io erfolgreich umgesetzt wird, entsteht ein modernes, benutzerfreundliches Multiplayer-Spiel, das direkt im Browser gespielt werden kann. Für uns Entwickler bedeutet es einen großen Lernerfolg im Umgang mit Web-Technologien.

8.3 Konsequenzen bei Nichtrealisierung

Sollte das Projekt nicht fertiggestellt werden, passiert im Alltag zunächst nichts, da es sich um ein Schulprojekt handelt. Allerdings würde die Chance verloren gehen, praktische Erfahrungen mit modernen Webtechnologien zu sammeln.

8.4 Risiko bei Nichtrealisierung

Das Risiko besteht vor allem darin, dass wir die Lernziele nicht erreichen. Dadurch fehlen uns wichtige Grundlagen für spätere Projekte oder Ausbildungen.

8.5 Konsequenzen bei verspäteter Realisierung

Wenn sich die Fertigstellung verzögert, kann das Spiel später präsentiert oder in kleinerem Umfang gezeigt werden.

8.6 Risiko bei verspäteter Realisierung

Das Risiko ist gering. Im schlimmsten Fall kann das Projekt in abgespeckter Form vorgeführt werden.

8.7 Ausweichmöglichkeiten

Als Ausweichmöglichkeit könnten wir die Umsetzung vereinfachen, zum Beispiel nur eine 2D-Version entwickeln oder bestimmte Features (z. B. zusätzliche Bomben) weglassen. So bleibt trotzdem ein funktionierendes Ergebnis.

9 Konsequenzen

9.1 Bei Projektfreigabe

Wenn das Projekt Bomb.io durchgeführt wird, können wir versprechen, dass am Ende ein funktionierendes Multiplayer-Spiel entsteht, das direkt im Browser läuft und ohne Installation spielbar ist. Für uns bedeutet es außerdem einen großen Lernerfolg im Bereich Webtechnologien, Teamarbeit und Projektorganisation.

9.2 Wenn das Projekt nicht oder zu einem späteren Zeitpunkt freigegeben wird

Falls das Projekt gar nicht gestartet oder stark verzögert wird, verpassen wir die Chance, praktische Erfahrung mit modernen Tools wie Node.js oder Three.js zu sammeln. Auch der Spaßfaktor, ein eigenes Spiel im Browser zu entwickeln, ginge verloren. Bei einer sehr späten Umsetzung besteht zudem das Risiko, dass im Modul zu wenig Zeit bleibt, um das Projekt erfolgreich abzuschließen.

10 Planung

Meilensteine und Termine

Meilenstein	Geplant
Projektfreigabe	22.08.2025
Freigabe Konzept	25.10.2025
Freigabe Realisierung	29.11.2025
Freigabe Einführung	13.12.2025
Abschluss	22.01.2026

Details siehe Dokument Projektplan.

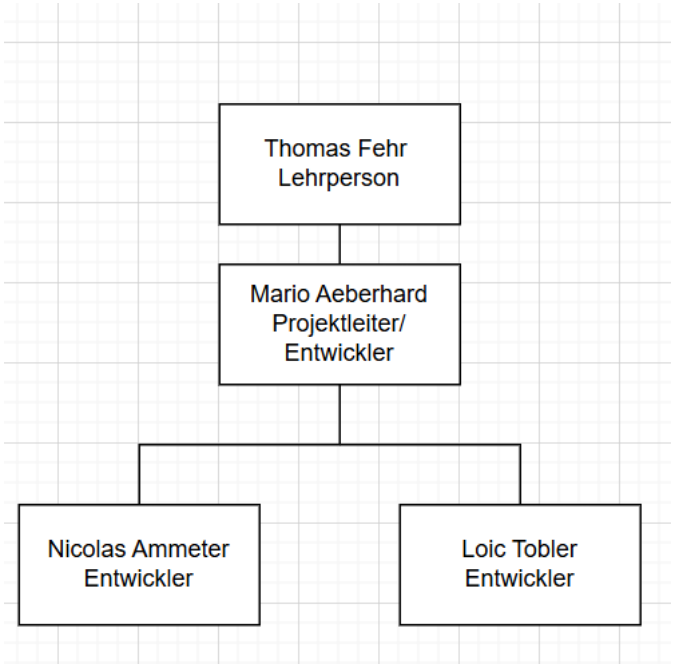
11 Projektführung

11.1 Zusammenfassung

Die Projektführung wird während des Projekts jeweils im **Projektplan** aktualisiert. Da es sich um ein kleines Schulprojekt handelt, übernehmen die drei Teammitglieder mehrere Rollen. Die wichtigsten Punkte sind:

- **Projektmanagement**: einfache Planung mit Zeitplan und Aufgabenlisten.
- **Risikomanagement**: Hauptrisiko ist Zeitknappheit oder technische Probleme bei 3D.
- **Qualitätssicherung**: regelmässige Tests und Review der Dokumente.
- **Konfigurationsmanagement**: Quellcode liegt auf GitHub, Dokumente in Teams/OneDrive.
- **Kommunikation**: Absprachen in den Lektionen, online über Discord/Teams.

11.2 Projektorganisation



Als Tabellesieht das so aus:

Rolle	**Name**	**Aufgabe / Verantwortung**
Auftraggeber / Betreuung	Thomas Fehr	Lehrperson, Abnahme, Feedback
Projektleiter / Entwickler	Mario Aeberhard	Planung, Organisation, Mitarbeit Entwicklung
Entwickler	Nicolas Ammeter	Entwicklung, Dokumentation
Entwickler	Loic Tobler	Entwicklung, Dokumentation

11.3 Projektberichte

Es werden wöchentliche kurze Statusmeldungen im Team erstellt. Die Lehrperson wird monatlich mündlich über den Fortschritt informiert. Alle wichtigen Zwischenergebnisse werden zusätzlich im Projektplan dokumentiert.

11.4 Kommunikation / Projektmarketing

Die Stakeholder (siehe Kapitel 4 „Liste der Stakeholder“) werden regelmässig über den Projektstand informiert. Dies geschieht hauptsächlich mündlich im Unterricht und zusätzlich durch den aktualisierten Projektplan. Sie haben dabei jederzeit die Möglichkeit, Rückmeldungen oder Verbesserungsvorschläge einzubringen. Falls nötig, können auch kurze Besprechungen einberufen werden, um offene Punkte zu klären.

Am Ende des Projekts erfolgt eine Präsentation mit Live-Demo von Bomb.io, die als Projektmarketing dient und das Ergebnis sichtbar macht.

11.5 Qualitätssicherung

11.5.1 Vorgehen zur Qualitätssicherung

Prüfung der Ergebnisse:
Alle Dokumente (z. B. Studie, Konzept, Projektplan) werden jeweils vor dem Abgabetermin fertiggestellt und nochmals durchgelesen.
Testen der Software:
Während der Entwicklung führen die Entwickler Unit-Tests durch (z. B. Timer, Bomben-Übergabe). Beim Systemtest wird das gesamte Spiel getestet (z. B. mehrere Spieler gleichzeitig, Ablauf mehrerer Runden). Dieser Test gilt auch als Abnahmetest.

11.5.2 Qualitätsziele

Nr.	Qualitäts-merkmal	Messbares Qualitätsziel	Besondere QS-Massnahmen
AUS BE-NUTZER-SICHT			
B1	Funktionserfüllung	100% der geplanten Spielfunktionen sind bis zur Präsentation lauffähig	2 zusätzliche Testläufe mit mehreren Spielern
B2	Effizienz	Spielstart in max. 5 Klicks ab Aufruf der URL	Review der Benutzeroberfläche
B3	Zuverlässigkeit	Spiel stürzt höchstens 1x pro 50 Spielrunden ab	Belastungstest mit mehreren Sessions
B4	Benutzbarkeit	Reaktionszeit < 100 ms pro Eingabe	Performance-Test während der Realisierungsphase
B5	Sicherheit	Kein unbefugter Zugriff auf die Spielsession möglich	Test der Lobby-Zugänge / Passwörter
AUS ENT-WICKLER-SICHT			
E1	Erweiterbarkeit	Neue Maps können innerhalb von 2 Stunden eingebaut werden	Strukturierte Code-Reviews
E2	Wartbarkeit	Jede Funktion ist mit Kommentar versehen	Peer-Review im Entwicklerteam
E3	Übertragbarkeit	Spiel läuft in Chrome, Firefox und Edge stabil	Cross-Browser-Tests
E4	Wiederverwendbarkeit	Mindestens 2 Module (z. B. Timer, Spieler-Management) sind in anderen Projekten nutzbar	Saubere Modultrennung
PROJEKT-FÜHRUNG			
P1	Kommunikation	Wöchentliches Update im Unterricht	Mündlicher Statusbericht
P2	Abstimmung mit Parallelprojekten	Nicht relevant (keine Parallelprojekte)	–
P3	Termineinhaltung	Keine Verspätung erlaubt	Wöchentliche Zeitplan-Kontrolle
P4	Projektdokumentation	Dokumente spätestens 1 Tag vor Abgabe fertiggestellt	Review durch Teammitglied

11.5.3 Initialer Prüfplan

Übersicht über die durchzuführenden QS-Massnahmen und deren Organisation (phasenweise):

Prüfobjekt	Termin	Prüfer	Prüfmethode	Prüfkriterien / Testfälle
Projektinitialisierungsauftrag	22.08.2025	Lehrperson	Review	Vollständigkeit
Studie	13.09.2025	Lehrperson	Review	Nachvollziehbarkeit
Projektplan	13.09.2025	Lehrperson	Review	Abgleich mit Terminen
Konzeptbericht	25.10.2025	Lehrperson	Review	Anforderungen abgedeckt
Spielprototyp	29.11.2025	Entwickler	White-Box-Test	Kernfunktionen: Timer, Bombe
Gesamttest Bomb.io	13.12.2025	Team + Lehrperson	Black-Box-Test	Mehrspieler-Funktion
Realisierungsbericht	29.11.2025	Lehrperson	Review	Dokumentation korrekt
Einführungsbericht	13.12.2025	Lehrperson	Review	Testprotokolle vollständig
Schlussbericht	09.01.2026	Lehrperson	Review	Vollständigkeit
Präsentationen & Demo	15./22.01.2026	Lehrperson + Klasse	Abnahmetest	Spiel lauffähig, Spassfaktor

Dieser Prüfplan wird im Dokument "Projektplan" jeweils pro Phase aktualisiert.

11.5.4 Prüfmethoden

Für das Projekt Bomb.io werden folgende Methoden der Qualitätsprüfung eingesetzt:

- **Review**
  Alle Dokumente (Studie, Projektplan, Konzept, Berichte) werden vor Abgabe von der Lehrperson und im Team durchgesehen. Dabei wird geprüft, ob Inhalte vollständig, korrekt und nachvollziehbar sind.
- **Black-Box-Test**
  Das Spiel wird von Spielern getestet, ohne dass sie den Programmcode kennen. Dabei wird kontrolliert, ob die sichtbaren Funktionen wie vorgesehen arbeiten (z.B. Timer läuft korrekt, Bombe explodiert, Spieler scheidet aus).
- **White-Box-Test**
  Entwickler testen den Code direkt, indem sie Testdaten einsetzen und mit Debugging-Tools prüfen, ob der Ablauf im Hintergrund korrekt funktioniert. So lassen sich logische Fehler frühzeitig entdecken.

11.5.5 Prüfspezifikationen

Die Anforderungen aus Kapitel 5 werden mit Testfällen überprüft. Dabei werden **funktionale Anforderungen** hauptsächlich mit Black-Box-Tests geprüft, während **nicht-funktionale Anforderungen** (wie Performance oder Stabilität) auch White-Box-Tests und Performancetests benötigen. Dokumente werden durch Reviews kontrolliert.

Funktionale Anforderungen

ID	Anforderung (Kap. 5)	Prüfmethode	Erwartetes Ergebnis	Erfüllt (Ja/Nein)	Bemerkung

F-01	Bombe weitergeben	Black-Box-Test	Bombe wechselt zum angeklickten Spieler		
F-02	Bombe explodiert nach Timer	Black-Box-Test	Spieler mit Bombe scheidet aus		
F-03	Timer läuft korrekt	White-Box-Test	Timer zählt von Startwert bis 0 ohne Fehler		
F-04	Rundenende	Black-Box-Test	Runde endet, wenn nur noch ein Spieler übrig ist		

Nicht Funktionale Anforderungen

ID	Anforderung (Kap. 5)	Prüfmethode	Erwartetes Ergebnis	Erfüllt (Ja/Nein)	Bemerkung
N-01	Performance	Performance-Test	Spiel reagiert in < 100 ms auf Klicks		
N-02	Stabilität	Dauertest / White-Box	Keine Abstürze bei 50 Runden hintereinander		
N-03	Benutzerfreundlichkeit	Review + Black-Box	Spielsteuerung verständlich, max. 3 Klicks für Aktion		

11.5.6 Checklisten für die Prüfung der Projektdokumente

Studie

Nr.	Prüfkriterium
1	Ist die Ausgangslage klar und nachvollziehbar beschrieben?
2	Sind Stärken und Schwächen der IST-Situation erkennbar?
3	Leiten sich die Projektziele logisch aus der Analyse ab?
4	Wurde der Nutzen des Projekts plausibel aufgezeigt?
5	Sind die Stakeholder sinnvoll benannt?
6	Sind Lösungsvarianten ausführlich beschrieben und ist die Auswahl begründet?
7	Wurden Konsequenzen bei Durchführung/Nichtdurchführung aufgeführt?
8	Sind Sprache und Darstellung korrekt?

Projektplan

Nr.	Prüfkriterium
1	Sind Termine, Risiken und Projektstatus vollständig dokumentiert?
2	Ist die Ressourcenplanung realistisch?
3	Sind Aktivitäten klar strukturiert und zugeordnet?
4	Ist die Konfigurationsidentifikation vorhanden und verständlich?
5	Ist die Darstellung übersichtlich und nachvollziehbar?

Konzept

Nr.	Prüfkriterium
1	Sind funktionale und nicht-funktionale Anforderungen vollständig formuliert?
2	Wurde die Benutzerschnittstelle erklärt und ggf. visualisiert?
3	Ist die Systemarchitektur verständlich dargestellt (inkl. Grafik)?
4	Sind Module und Schnittstellen klar beschrieben?
5	Wurde eine erste Fassung der Testfalltabelle erstellt?
6	Ist Sprache und Darstellung korrekt?

Realisierungsbericht

Nr.	Prüfkriterium
1	Ist die Umsetzung der Architektur bis auf Elementebene dokumentiert?
2	Sind Code-/Systembestandteile ausreichend beschrieben?
3	Sind Schnittstellen intern und extern erklärt?
4	Liegt eine Benutzer- und Supportdokumentation vor?
5	Deckt das Testkonzept alle Anforderungen ab?
6	Sind Testfälle mit Ergebnissen dokumentiert?
7	Wurden Planung und Ressourcen aktualisiert?

Einführungsbericht

Nr.	Prüfkriterium
1	Ist das Vorgehen bei der Einführung strukturiert und realistisch?
2	Ist eine Migration (falls nötig) sauber beschrieben?
3	Gibt es einen Ausbildungs-/Einführungsplan für Anwender?
4	Ist der Abnahmetest dokumentiert und protokolliert?
5	Wurden Risiken bei der Einführung beachtet?

Schlussbericht

Nr.	Prüfkriterium
1	Enthält der Bericht eine prägnante Kurzfassung?
2	Wurde das Arbeitsjournal oder eine Reflexion dokumentiert?
3	Sind Erkenntnisse und Reflexionsfähigkeit sichtbar?
4	Ist die Gliederung klar und logisch?
5	Sind Sprache, Darstellung und Rechtschreibung korrekt?
6	Wurden Testverfahren und deren Resultate dokumentiert?
7	Sind Grafiken und Darstellungen vorhanden und sinnvoll?

11.5.7 Testfalltabellen

In der Konzeptphase wird damit begonnen, Testfälle zu sammeln, soweit sie bei der Erarbeitung der Systemanforderungen und der Systemarchitektur bereits erkannt werden können.

In der Realisierungsphase werden die Testfalltabellen in den Realisierungsbericht übernommen und dort weiter detailliert.

Die Testfalltabellen sollen mindestens folgende Information enthalten:

- Nummer
- Kurzbezeichnung
- Allenfalls abgedeckten Anwendungsfall
- Erwartetes Ergebnis
- Bemerkungen

11.6 Konfigurationsmanagement

11.6.1 Konfigurationsidentifikation

Übersicht über die im Projekt entstehenden Ergebnisse:

Ergebnis	Bemerkungen
Studie (1_1_Studie_Bombio)	Enthält Situationsanalyse, Ziele, Anforderungen, Varianten
Projektplan (1_2_Projektplan_Bombio)	Enthält Termine, Ressourcen, Risiken, Projektstatus
Konzeptbericht (2_1_Konzept_Bombio)	Enthält Architektur, UI, detaillierte Anforderungen, erste Testfälle
Realisierungsbericht (3_1_Realisierung_Bombio)	Enthält Implementierungsdetails, Tests, Code-Dokumentation
Einführungsbericht (4_1_Einführung_Bombio)	Enthält Einführungsschritte, Abnahmeprotokoll
Schlussbericht (4_2_Schlussbericht_Bombio)	Enthält Reflexion, Testresultate, Bewertung
Software: Bomb.io-Code (Frontend/Backend)	Quellcode der Spielumgebung (Express, JavaScript, Three.js)
Assets: 3D-Modelle, Texturen, Sounds	Werden für die Spielgrafik und Effekte benötigt

11.6.2 Ablagestruktur

Die Ergebnisse des Projekts Bomb.io werden auf dem Github gespeichert, und zwar unter: https://github.com/DerHagelmacher/bombio
Dort erfolgt die Ablage in Unterordnern:

- **Dokumente** → enthält alle Projektdokumente (Studie, Projektplan, Konzept, Berichte)
- **Software** → enthält den Quellcode des Spiels
  - **src** (Client, Server, Shared Code)
  - **assets** (3D-Modelle, Texturen, Sounds)
  - **tests** (Unit- und Integrationstests)

Damit ist sichergestellt, dass Projektdokumente und Software klar getrennt abgelegt sind.

11.6.3 Namenskonventionen

Die Projektergebnisse werden nach einem einheitlichen Schema benannt. Grundlage sind die Hermes-Phasen, kombiniert mit einer laufenden Nummer und dem Projektnamen.

- **Beispiel Dokumente:**
  - Studie in Phase Initialisierung: 1_1_Studie_Bombio.docx
  - Projektplan in Phase Initialisierung: 1_2_Projektplan_Bombio.docx
- **Beispiel Software:**
  - Builds werden mit Datum versehen: Bombio_Build_2025-09-14.exe
  - Quellcode-Dateien folgen dem Muster <Modul>_<Version>.js, z. B.: playerControl_v1.2.js
  - Ressourcen werden beschreibend benannt, z. B.: texture_wall_v1.png oder sound_explosion_v2.wav

So ist jederzeit klar ersichtlich, ob es sich um Dokumente, Code, Ressourcen oder fertige Spielversionen handelt.

12 Projektantrag

Hiermit beantragen wir, dass das Projekt Bomb.io mit der Lösungsvariante Node.js/Three.js für die Phase Konzept freigegeben wird.

Bern, 22.08.2025

Mario Aeberhard, Loic Tobler, Nicolas Ammeter