Nautical Nonsense: Because Sometimes You Just Need to Sink Something

Jakob Götz
j.goetz@oth-aw.de

Uwe Kölbel
u.koelbel@oth-aw.de

Maximilian Schlosser
m.schlosser@oth-aw.de

Oliver Schmidts
o.schmidts@oth-aw.de

Jan Schuster
j.schuster@oth-aw.de

Philipp Seufert
p.seufert@oth-aw.de

Fabian Wagner
f.wagner@oth-aw.de

***Zusammenfassung*—Wir entwickeln ein Cloud Native Browser-Game nach dem Vorbild von „Schiffe versenken“. Dazu stellen wir die Webanwendung *Nautical Nonsense* vor.**

I. Einleitung

Spiele, die im Browser ausführbar sind, erfreuten sich während der Corona-Pandemie größter Beliebtheit. Die Menschen durften sich aufgrund der hohen Ansteckungsrate des Virus nicht in der Realität treffen.

Dies war der Grund, weshalb diese regelmäßigen Treffen häufig in die digitale Form wechselten. Es wurden Spielabende mit beispielsweise Skribbl.io **skribblio** veranstaltet, da für die Teilnahme keine leistungsfähigen Rechner, sondern meist lediglich ein Smartphone oder Ähnliches benötigt wurde.

Das Brettspiel „Schiffe versenken“ ist ein sehr beliebtes und intuitives Spiel, das seit Jahrzehnten Generationen von Spielern begeistert. Dieses Spiel bietet eine perfekte Mischung aus Strategie, Glück und Unterhaltung, die es zu einem zeitlosen Klassiker gemacht hat.

Aus diesem Grund wollen wir in dieser Modularbeit das Spiel im Browser implementieren, damit unabhängig von äußeren Faktoren wie beispielsweise der Corona-Pandemie, das Gesellschaftsspiel Battleship gespielt werden kann.

II. Verwandte Arbeiten

Im Jahr 2015 veröffentlichte Matheus Valadares ein simples Browserspiel namens Agar.io **agario**. Das Spielprinzip ist einfach: Man steuert eine Zelle in einer Petrischale und versucht größer zu werden, indem man Agar-Pellets und kleinere Zellen verschluckt. Das Spiel erfreute sich großer Beliebtheit und die Zahl der monatlichen Spieler ist schnell in die Millionen gestiegen **takahashi2017**. Um einen solch rapiden Anstieg an Spielern ohne Ausfälle oder Überlastungen bewältigen zu können, ist eine Cloud-native Architektur ideal. Dazu wird typischerweise die Anzahl der aktiven Server an die Anzahl der aktiven Spieler angepasst. Ein solcher Skalierungsmechanismus ist auch für unser Spiel vorgesehen.

Agar.io bietet mittlerweile eine Vielzahl von Features, die für Online-Spiele typisch sind. Dazu gehören verschiedene Spielmodi, eine Login-Funktion, Mikrotransaktionen und Skins. Es ist denkbar, jedes dieser Features auch in unser Spiel zu integrieren.

III. Anforderungen

In der Anforderungsanalyse wurden drei primäre Komponenten identifiziert, die es umzusetzen gilt: (1) das Spiel (2) die Cloud-Fähigkeit (3) das Verarbeiten der Daten von den Nutzern. Als Stakeholder wurden Spieler und Entwickler identifiziert. Die Anforderungen werden im Folgenden als User Stories beschrieben.

A. Menüführung

Als Spieler möchte ich ein intuitives Menü haben, in dem ich folgende Funktionen auswählen kann:

1) Meinen Spielernamen eingeben
2) Ein Spiel starten
3) Eine Anleitung aufrufen
4) Eine Bestenliste einsehen
5) Ein Button zum Ein- und Ausschalten der Musik

B. Spielmodi

Als Spieler möchte ich zwischen verschiedenen Spielmodi wählen können:

1) Gegen Computer
2) Gegen zufälligen Gegner
3) Gegen Freund

C. Spielbeginn

Als Spieler möchte ich das Spiel nach folgenden Regeln vorbereiten:

1) Jeder Spieler eine feste Anzahl an spezifischen Schiffen, welche auf dem Spielfeld platziert werden
2) Schiffe dürfen sich bei der Platzierung nicht überschneiden
3) Sind alle Schiffe platziert, kann der Spieler seine Spielbereitschaft über einen Button signalisieren

D. Spielbrett

Als Spieler möchte aktuelle Informationen über den Spielverlauf auf dem Spielbrett angezeigt bekommen:

1) Die eigenen Schiffe werden aufgedeckt angezeigt, die gegnerischen Schiffe sind verdeckt
2) Treffer und verfehlte Schüsse sind auf dem Spielbrett markiert
3) (optional) Ein Chat mit dem Gegner angezeigt

E. Spielablauf

Als Spieler möchte ich in der aktiven Phase des Spiels:

1) Ein Feld auf dem Spielbrett markieren können, welches beschossen werden soll
2) Ein Button haben, um den Zug zu beenden
3) Der Gegenspieler kann nicht agieren, bis der Zug vom aktiven Spieler beendet ist

F. Erreichbarkeit

Als Spieler möchte ich, dass die Anwendung immer erreichbar ist:

1) Unabhängig von der aktuellen Spielerzahl
2) Service Level Requirement von 99.9
3) (optional) Unabhängig von dem Endgerät

G. (optional) Verzögerung

Als Spieler möchte ich minimale Verzögerung, damit das Spielerlebnis nicht darunter leidet:

1) Die Verzögerung zwischen Spielbeginn und Spielablauf darf nicht mehr als 2 Sekunden betragen
2) Die Verzögerung zwischen beenden des Zugs und beginn des gegnerischen Zugs darf nicht mehr als 2 Sekunden dauern
3) Die Verzögerung bei der Auswahl der Menü-Punkte darf nicht mehr als 0.5 Sekunden betragen

H. Cloud Native

Als Entwickler möchte die Anwendung als Cloud Native bereitstellen, damit wir skalierbar nach der Nutzeranzahl operieren können:

1) Es wird eine Container-Technologie eingesetzt
2) Die Container müssen unabhängig voneinander sein
3) Es kommt ein Orchestrierungs-Tool zum Einsatz

I. (optional) Dashboard

Als Entwickler möchte ich ein Dashboard, welches mir Informationen zur Infrastruktur und der Nutzung anzeigt:

1) Wahl eines geeigneten Tools zur Darstellung der Informationen
2) Die Informationen im Dashboard müssen aktuell sein

J. (Optional) Bereitstellung in der Cloud

Als Entwickler möchte die Anwendung über einen Cloud-Anbieter bereitstellen:

1) Die Kosten hierfür müssen im Maße bleiben
2) Die Anwendung ist über das Internet zu erreichen
3) Automatisiertes Deployment über eine CI/CD Pipeline

IV. Methoden

Das Projekt setzt sich aus drei Komponenten zusammen: Frontend, Backend und Datenbank. Die einzelnen Komponenten werden jeweils in einem Docker-Container **docker** ausgeliefert. Sie werden mit einem in der Cloud gehosteten Kubernetes-Cluster **kubernetes** orchestriert und erreichbar gemacht.

Für die Visualisierung des Spielgeschehens im Frontend wird das Game Framework Phaser **phaser** mit JavaScript verwendet. Ein Webserver stellt das Frontend für den Nutzer zur Verfügung. Das Backend wird in Python implementiert und stellt eine RESTful-API zur Verfügung. Zusätzlich wird für die technische Kommunikation zwischen zwei Clients eine geeignete Technologie eingesetzt. Dies wird mithilfe des Frameworks FastAPI **fastapi** realisiert und dient zur Kommunikation mit dem Frontend. Als Datenbank wird MongoDB **mongodb** verwendet. Die Spielverläufe werden als JSON-Objekte serialisiert und für spätere Analysen gespeichert.