RacingLines - Technical Report

Martin Zizler, Christoffer Mayer, Samuel Kern, Alina Gast, Anika Fabig, Ute Koller, Esra Kellecioglu

***Abstract*— Dieser Bericht zeigt alle relevanten technischen Informationen des Projektes "RacingLines". Damit soll sich der Leser einen schnellen Überblick über unsere Qualitätsziele und ihre Lösungsansätze verschaffen können. Letztlich wird der Aufwand des Projekts im Absatz Lessons Learned reflektiert.**

I. Mission Statement

RacingLines ist ein Online Browser Game, welches den Spielern erlaubt rundenbasiert gegeneinander anzutreten. Dabei steuert jeder Spieler einen Kreis, der eine immer länger werdende Linie hinter sich herzieht, und muss versuchen gegen keine Linie und auch gegen keine Wand zu fahren. Der am längsten überlebende Spieler bekommt jeweils die meisten Punkte. Das Spiel soll dabei leicht zu lernen und schwer zu meistern sein und damit eine große Zielgruppe ansprechen.

II. Rahmenbedingungen

A. Technische Vorgaben

1) Programmiersprache
   - Javascript
2) Frameworks & wichtige Libraries:
   - Express, React, React Router, MaterialUI, Socket.io
3) Zielumgebung (Plattform, Betriebssystem(e))
   - V-Server, Debian 11, PM2 (Node process manager)

B. Organisatorische Vorgaben

1) Zeitrahmen
   - 16.06.2022 bis 04.07.2022
2) Vorgehen und Organisation des Teams
   - Organisation über Discord Server und Miro Board. Feste wöchentliche Meetings über momentanen Stand montags um 9:45 Uhr

C. Konventionen

1) Source Code
   - Quelltextverwaltung bei GitLab, https://git.oth-aw.de/waecyan/racinglines
2) Defect Tracking
   - Kanban-Board in Miro
3) Community
   - Interaktion mit Nutzer über BigBlueButton

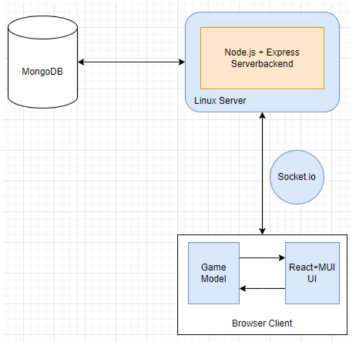


Fig. 1. Tech-Stack-Grafik

III. Qualitätsziele

TABLE I
Qualitätsziele Beschreibungen

Qualitätsziel	Unterziel	Beschreibung
Betreibbarkeit	Gute Benutzbarkeit	System kann verstanden, erlernt und verwendet werden und ist attraktiv für Benutzende.
	Gute Benutzbarkeit	Gelegenheitsnutzer können, die ihnen bekannten Shortcuts zum Kopieren des Links benutzen.
	Gute Benutzbarkeit	Die Steuerung der Linie ist durch die Pfeiltasten und die Tasten "A" und "D" realisiert. Die Richtung der Linie ist somit selbsterklärend.
	Schnelle Reaktion auf Eingaben	Die Inputs des Spielers sollen während dem Spiel direkt verarbeitet werden, sodass der Spieler nach der Eingabe keine Verzögerung hat.
Interoperabilität	Bestehende Frontends nutzen	Racing Lines lässt sich mit angemessenem Aufwand in bestehende grafische Game-Frontends einbinden.
Funktionale Eignung	Akzeptable Spielstärke	Neue Spieler werden mit angemessener Liniengeschwindigkeit und einer großen Spielfläche nicht überfordert. Bei kleineren Spielfeldern und höheren Liniengeschwindigkeiten können sich erfahrenere Spieler fördern lassen.
	Erlernbarkeit	Gelegenheitsnutzer sollten in weniger als 3 Sekunden in der Lage sein, das Spiel zu steuern.

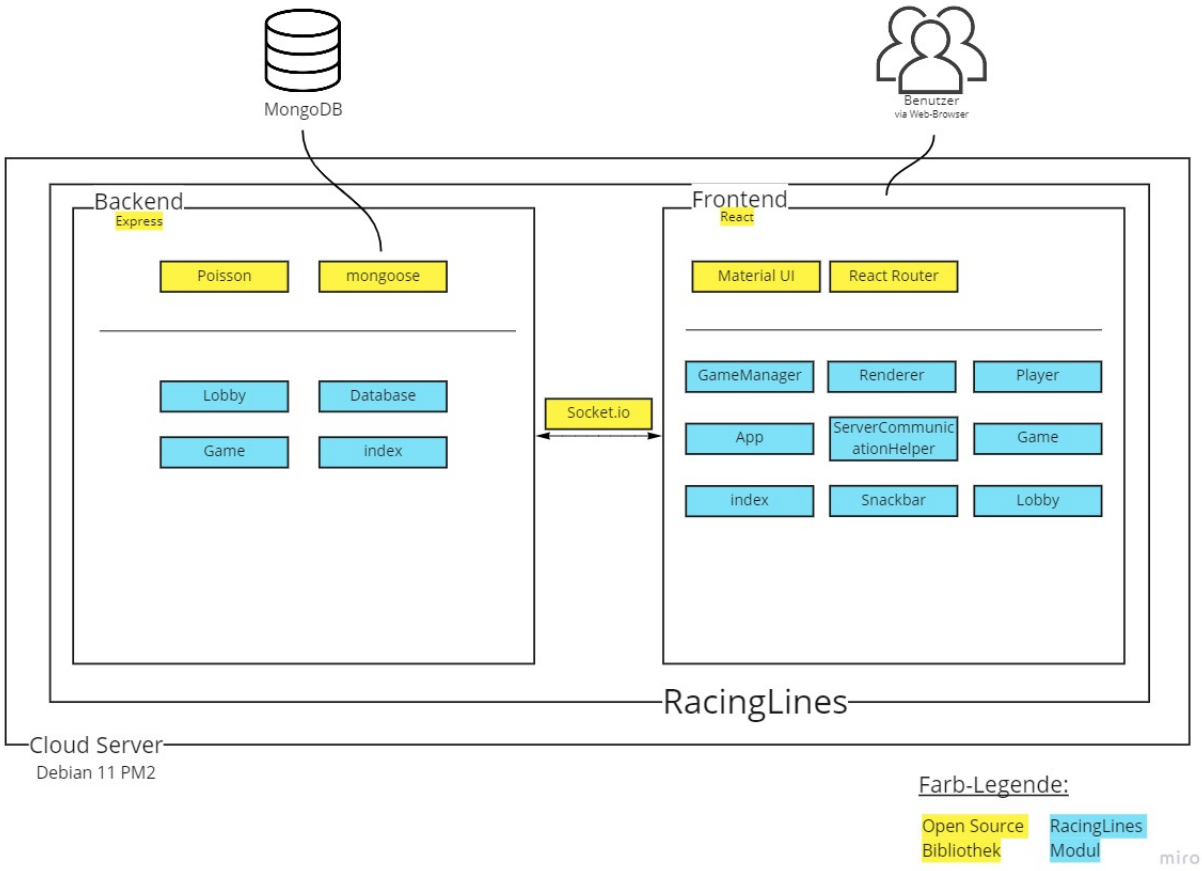
Qualitätsziel	Unterziel	Beschreibung
Effizienz	Gute Stabilität	Das Spiel soll mit einer konsistenten Framerate laufen, damit das Spiel möglichst flüssig erscheint.
	Browser-übergreifende Kompatibilität	Die Anwendung soll auf möglichst vielen Browsern lauffähig sein.
Zuverlässigkeit	Synchronisierter Spielstatus	Alle Spieler sollen immer einen möglichst synchronen Spielzustand haben.
Wartbarkeit	Erweiterbare Architektur	Hinzufügen neuer Spielmechaniken (z.B. Power-Ups) wird durch eine erweiterbare Software-Architekur ermöglicht.
	Testbarkeit	Durch Unterteilung in einzelne React-Komponenten, lässt sich die Anwendung leichter testen.
	Testbarkeit	Fehler werden durch einzeln zu testende Komponenten einfacher gefunden.
	Nutzung von Open-Source-Lösungen	Durch das Nutzen gängiger Open-Source-Lösungen lassen Probleme schneller lösen, da es viel Dokumentation dazu im Netz gibt.

IV. Lösungsstrategie

TABLE II
LÖSUNGSSTRATEGIEN

Qualitätsziel	Lösungsansätze in RacingLines
Leicht zu betreiben	• Node als Basis für Frontend (React) und Backend (Express) • Normaler Web-Browser reicht als Client • Deployment pipeline erlaubt schnell testbare Veränderungen • Durch Umgebungsvariablen kann das System sowohl lokal, als auch auf dem Server ausgeführt werden, ohne etwas ändern zu müssen
Gute Benutzbarkeit	• Verständlichkeit: Am Anfang des Spiels wird ein Snackbar-Element zur Erklärung der Steuerung angezeigt. • Verständlichkeit: Während der Countdown für das Spiel am Anfang der Runde herunterzählt, bekommt der Nutzer angezeigt, in welche Richtung er sich bewegt, nachdem das Spiel beginnt. • Verständlichkeit: Die minimalistische Gestaltung des Spiels lenkt den Spieler nicht vom Spielgeschehen ab. • Erlernbarkeit: Durch das simple Spielprinzip kann das Spiel sehr schnell erlent werden. • Bedienbarkeit: Durch häufiges updaten des Game-States, fühlen sich die Eingaben responsive an.
Hohe Zuverlässigkeit	• Wichtige Statistiken (wie Scores) werden mit Datenbank gefestigt (MongoDB auf AWS) • Deployment auf Linux Server mit PM2 sorgt für stabile uptime von backend und frontend und erlaubt monitoring • Alle Unittests werden vor jedem Deployment durchgeführt
Gute Stabilität	• Canvas-Layering: Die Spieler und die Linien werden auf verscheidenen Canvas Elementen gezeichnet. Der Linien-Canvas wird nicht geleert, sondern es wird nur dazu gezeichnet. Somit muss man die Positionen der Linien nicht speichern.
Synchronisierter Spielstatus	• Mit der "socket.io"-Library wird eine bidirektionale Echtzeitkommunikation implementiert. • Nach dem Rundenstart generiert der Server die Startpositionen für die Spieler, die sich in der Lobby befinden und sendet diese an jeden Client mit den ausgewählten Farben und Namen der Spieler. • Nach jedem Update des Spielstatus wird der Spielerstatus des Clients an den Server übertragen. Der Spielerstatus beinhaltet die Position des Clients und ob er gerade zeichnet. Der Sever leitet den Status an die anderen Clients, in der Lobby weiter, die diesen Spielerstatus in ihrem Spielstatus verarbeitet, sodass das Spiel auf allen Clients möglichst synchron ist. • Im Client werden die empfangenen Spielerdaten in einem Buffer gespeichert und der Reihenfolge nach abgearbeitet. • Um die Änderungen am Score von dem Player-Objekt kontinuierlich an die Scoreboard-Komponente weiterzugeben, haben wir uns für das Observer-Pattern entschieden.
Synchronisierte Lobby	• Mit der "socket.io"-Library wird eine bidirektionale Echtzeitkommunikation implementiert. • Nachdem ein neuer Nutzer der Lobby beitritt, wird er dem jeweiligen Socket.io Raum hinzugefügt und es wird eine neue Spielerkomponente angelegt, deren Namen und Farbe verändert werden kann. Dieser Name und die Farbe werden bei einer Änderung auch bei den anderen Spielern aktualisiert.
Gute Wartbarkeit	• Einsatz von einzelnen React Komponenten führt zu einer Vereinfachung der Anwendungtests • Es besteht die Möglichkeit für das Hinzufügen neuer Spielmechaniken durch eine erweiterbare Software-Architektur • Einsatz von gängigen Open-Source-Lösungen vereinfacht das Debugging mit Hilfe von vielen Hilfestellungen und Dokumentationen • Einsatz von der Code-Erweiterung Prettier für eine einheitliche Formatierung des Codes

V. Überblicksbild



VI. Ausblick

Racing Lines ist bereits auf Webbrowsern als Multiplayergame verfügbar, doch es gibt noch einige Vorschläge zur Weiterentwicklung, die dieses Spiel aufwerten würden:

- Benutzbarkeit: Racing Lines auch mit Touch auf mobilen Geräten spielen können.
- Networking: Implementierung des Spiels mit einem Server-Authoritative Networking Ansatz, um Cheats zu verhindern und einer Client-Side Prediction des GameStates um dem Nutzer das Gefühl zu geben, dass das Spiel direkt auf seine Eingaben reagiert.
- Accounts: Benutzer können sich registrieren und Statistiken sammeln.
- Ranking-System: Nutzer sammeln Ranglistenpunkte bzw. verlieren diese nach einem Spiel.
- Öffentliche Lobbysuche: Nutzer können nach offenen Lobbys suchen und diesen beitreten.
- Power-ups: Spieler können während dem Spiel Power-ups "überfahren" und somit, ihre Geschwindigkeit/Größe verringern oder erhöhen.
- Chatfunktion: Mit einer Chatfunktion können können sich Spieler nach einer Runde austauschen.

VII. Lessons Learned

Beim Umsetzen des Spiels ist schnell aufgefallen, dass es mehr Arbeit als erwartet ist, einen synchronisierten Spielstatus auf allen Clients herzustellen, während auch die Inputs responsive bleiben. Hier müssen sehr viele Daten zu den richtigen Zeiten zwischen Server und Clients über socket.io verschickt werden, dabei ist es sehr wichtig, dass die Daten mit minimaler Verzögerung und in richtiger Reihenfolge ankommen und verarbeitet werden. Um dies im angedachten zeitlichen Rahmen schaffen zu können, wäre mindestens Vorerfahrung im Bereich von Echtzeit Online Multiplayer Spielen erforderlich. Außerdem wäre es besser gewesen, wenn wir vorher eine ausführliche Architekturskizze, rein für die socket.io Verbindungen erarbeitet hätten, was aber auch schwierig war, da wir viel erst über trial and error herausfinden konnten.
Bei Übergabe von Daten zwischen Lobby und Game gab es auch noch weitere Probleme, da wir sicherstellen mussten, dass der Client auch identifizierbar bleibt, wenn die Seite gewechselt wird.
Ein Spiel, welches nicht auf durchgängige Echtzeitdatenübertragung setzt, wie Schach oder Tic-Tac-Toe wäre wohl deutlich einfacher bei der Umsetzung

und man könnte sich auch mehr auf kreative Use-Cases und Features konzentrieren.
Außerdem hatte im Team niemand Vorerfahrung mit socket.io oder React, wodurch sich sehr schlecht einschätzen lies, wie lange wir für die Einarbeitung und die Fertigstellung von Aufgaben brauchen. Durch das Fehlen von Erfahrung mit React-Components, war es nicht möglich, einfach React-Hooks zu verwenden, und deshalb mussten wir manchmal auf klassische Observer-Pattern zurückgreifen, um States zu aktualisieren. Vielleicht wäre hier Vue.js eine leichtere Alternative zu React gewesen.
Lange waren wir uns auch unsicher, wie wir das eigentliche Kernspiel realisieren können und ob dafür eine game engine, wie PixiJS nötig ist. Um die Performance zu optimieren haben wir auch erst versucht Pixi.js/ReactPixi zu verwenden, aber wir konnten keine Verbesserung der Performance feststellen, weswegen wir die Idee dann auch verworfen haben. Letztendlich haben wir uns dafür entschieden, dass wir unser Spiel mit dem normalen HTML5 Canvas umsetzen.

Insgesamt konnten wir dennoch extrem viele wertvolle Erfahrungen machen, die erst durch unseren "Sprung ins kalte Wasser" möglich geworden sind.

VIII. Glossar

- Backend: Teil des Systems welcher die Spieler verwaltet
- Frontend: Teil des Systems, welches der Benutzer sieht
- Node.js: Open-Source Javascript backend Umgebung
- Express: Backend Framework für Node.js, um Webanwendungen und APIs zu erstellen
- React: Frontend Javascript Bibliothek, um Benutzeroberflächen zu erstellen
- Socket.io: Javascript Bibliothek für echtzeit Webanwendungen, für Backend und Server
- Big Blue Button (BBB): Open-Source Web-Konferenzsystem
- Discord: Online Plattform für Anrufe und Textnachrichten
- GitLab: Open-Source DevOps Webanwendung
- Miro: Online Kollaborationsplattform mit Tafel
- MaterialUI: UI-Bibliothek für Frontend
- Poisson: Poisson disk sampling als Bibliothek für Spielerverteilung
- MongoDB: Open-Source Datenbank Programm
- Prettier: Einheitliche Formatierung von Code
- React Router: Bibliothek, mit der das Routing implementiert wurde.