Universität Ulm

Entwickler-Handbuch Editor

Battle of the Centerländ

- ein Code sie alle zu knächten

Version: 0.21 - TEAM 11
Softwaregrundprojekt WiSe 2022/23 und SoSe 2023

Katharina Böcker, Bill Akhter, David Hamberger, Dennis Authaler,
Nick Bethke, Tom Haßler

22. Mai 2023

Inhaltsverzeichnis

1 Allgemein **3**

2 Frameworks **4**

2.1 JavaScript/TypeScript 4

2.2 Node.js 5

2.3 Electron 5

2.4 React 6

2.5 Tailwind CSS 6

3 Voraussetzungen **7**

3.1 Node.js 7

3.2 NPM / Yarn 7

3.3 Vorbereitung 7

4 Projektstruktur **9**

4.1 Haupt-/Main-Prozess 9

4.1.1 IPC 9

4.2 Rendererprozess 10

4.2.1 React 10

4.2.2 Translation-Helper 10

4.2.3 Components 11

Inhaltsverzeichnis

1 Allgemein

Der **Editor** für das Spiel **Battle of the Centerländ - ein Code sie alle zu knächten** dient der Bearbeitung und Erstellung von Spielbrettern und Spielkonfigurationen. Er ist in der Programmiersprache **JavaScript**/**TypeScript** und den **Node.js**-Frameworks **Electron** und **React** geschrieben.

Der Editor ist in der Lage, die Spielbretter und Spielkonfigurationen zu generieren, zu bearbeiten und zu speichern. Die Spielbretter und Spielkonfigurationen werden in **JSON**-Dateien gespeichert und können von dem Editor geöffnet werden.

Der Editor ist in der Lage, die Spielbretter und Spielkonfigurationen zu validieren. Die Validierung wird durchgeführt, wenn der Benutzer das Spielbrett oder die Spielkonfiguration bearbeitet. Die **JSON**-Dateien werden nach den **JSON Schemas** validiert, welche im Standard-Komitee entwickelt wurden, und auf unsinnige Werte und Konfigurationen geprüft. Desweiteren Prüft der Editor, ob die Spielbretter und Spielkonfigurationen die Anforderungen des Spiels erfüllen oder ob sie unvollständig sind.

Durch die Implementierung des Editors in **JavaScript**/**TypeScript** und der Verwendung des Framework **Electron**, ist der Editor auf allen gängigen Betriebssystemen lauffähig, da **Electron** auf **Chromium** basiert, welches auf allen gängigen Betriebssystemen verfügbar ist. Der Editor wurde ausführlich auf **Windows**, **Linux** und **macOS** getestet.

In diesem Entwickler-Handbuch wird erkärt, wie der Editor funktioniert sowie weiterentwickelt, aktualisiert und installiert werden kann.

2 Frameworks

Der Editor ist in der Programmiersprache **JavaScript/TypeScript** und den **Node.js**-Frameworks **Electron** und **React** geschrieben.
Das Styling des Editors wird mit **Tailwind CSS** realisiert.

In der `package.json`-Datei sind alle Abhängigkeiten des Editors aufgelistet unter **dependencies** und **devDependencies**.

2.1 JavaScript/TypeScript

JavaScript[1] ist eine weit verbreitete Programmiersprache, die hauptsächlich für die Entwicklung von interaktiven Webseiten und Webanwendungen verwendet wird. Sie ermöglicht es Entwicklern, Funktionen, Abläufe und Interaktionen in den Browsern der Benutzer einzubetten.
JavaScript ist eine interpretierte Sprache, was bedeutet, dass der Code zur Laufzeit ausgeführt wird. **JavaScript** ist eine objektorientierte Sprache, was bedeutet, dass **JavaScript** Objekte verwendet, um Daten zu speichern und zu verarbeiten.
JavaScript wird seit 1995 von **Netscape** entwickelt und ist eine Open-Source-Programmiersprache.

TypeScript[2] ist eine Programmiersprache, die auf **JavaScript** basiert. **TypeScript** ist eine objektorientierte Sprache, was bedeutet, dass **TypeScript** Objekte verwendet, um Daten zu speichern und zu verarbeiten. **TypeScript** ist eine kompilierte Sprache, was bedeutet, dass der Code vor der Ausführung in **JavaScript** kompiliert wird. **TypeScript** ist eine typisierte Sprache, was bedeutet, dass **TypeScript** Variablen und Funktionen einen Typ haben. TypeScript wurde 2012 von **Microsoft** entwickelt und ist eine Open-Source-Programmiersprache.

[1] https://www.w3schools.com/js/DEFAULT.asp
[2] https://www.typescriptlang.org/

2.2 Node.js

Node.js[3] ist eine Open-Source-Plattform, die es Entwicklern ermöglicht, serverseitige Anwendungen mit **JavaScript** zu erstellen. **Node.js** basiert auf der **V8 JavaScript Engine**[4] von **Google**. **Node.js** ist eine kompilierte Plattform, was bedeutet, dass der Code vor der Ausführung in **JavaScript** kompiliert wird. Es wurde 2009 von **Ryan Dahl** entwickelt und ist eine Open-Source-Plattform.

2.3 Electron

Electron[5] ist ein Open-Source-Framework, das es Entwicklern ermöglicht, Desktop-Anwendungen mit **JavaScript**, **HTML** und **CSS** zu erstellen. **Electron** basiert auf **Node.js** und **Chromium**[6]. **Electron** ist eine kompilierte Plattform, was bedeutet, dass der Code vor der Ausführung in **JavaScript** kompiliert wird. Es wurde 2013 von **GitHub** entwickelt und ist ein Open-Source-Framework.

[3] https://nodejs.org/en/
[4] https://v8.dev/
[5] https://electronjs.org/
[6] https://www.chromium.org/

2.4 React

React[7] ist ein Open-Source-Framework, das es Entwicklern ermöglicht, Benutzeroberflächen mit **JavaScript** und **HTML** zu erstellen. Es vereinfacht die Entwicklung von Benutzeroberflächen, indem es Komponenten verwendet, um die Benutzeroberfläche in unabhängige, wiederverwendbare Teile zu unterteilen. Es wird seit 2013 von **Facebook** entwickelt und ist ein Open-Source-Framework.

2.5 Tailwind CSS

Tailwind CSS[8] ist ein Open-Source-Framework, das es Entwicklern ermöglicht, Benutzeroberflächen mit **CSS** zu erstellen. Es vereinfacht die Entwicklung von Benutzeroberflächen, indem es Klassen verwendet, um die Benutzeroberfläche in unabhängige, wiederverwendbare Teile zu unterteilen. Es wurde 2017 von **Adam Wathan** entwickelt und ist ein Open-Source-Framework.

[7] https://reactjs.org/
[8] https://tailwindcss.com/

3 Voraussetzungen

3.1 Node.js

Um den Editor zu installieren, muss **Node.js** auf dem System installiert sein. Für die Installation von **Node.js** gibt es unterschiedliche Installationsanweisungen, abhängig von dem Betriebssystem, auf dem **Node.js** installiert werden soll. Zu finden sind diese unter https://nodejs.org/en/download/.

3.2 NPM / Yarn

Um den Editor zu installieren, muss **NPM** oder **Yarn** installiert sein. **NPM** wird mit **Node.js** installiert.//

Yarn kann unter https://yarnpkg.com/en/docs/install installiert werden, oder mit **NPM** installiert werden, indem der Befehl `npm install -g yarn` ausgeführt wird.

NPM und Yarn werden für die Installation der Abhängigkeiten des Editors benötigt. Sie sind beide Paketmanager für **Node.js**.

3.3 Vorbereitung

1. Lade den Quellcode des Editors herunter.
2. Entpacke den Quellcode des Editors.
3. Öffne ein Terminal.
4. Navigiere in das Verzeichnis, in dem der Quellcode des Editors entpackt wurde.
5. Führe den Befehl `npm install` oder `yarn install` aus.
6. Warte, bis die Abhängigkeiten installiert wurden. Das kann einige Minuten dauern. Es kann sein, dass bei der Installation der Abhängigkeiten Warnungen oder Fehler auftreten. Diese können ignoriert werden.

7. Führe den Befehl `npm i -g ts ts-node` oder `yarn global add ts ts-node` aus. Dieser Befehl installiert die **TypeScript**-Compiler, welche für die Entwicklung des Editors benötigt werden.
8. Führe den Befehl `npm run start` oder `yarn start` aus.
9. Nun, sollte der, Editor gestartet werden und als ein Fenster angezeigt werden.

Dies ist auch alles auch noch einmal in der **README**-Datei des Editors beschrieben.

4 Projektstruktur

Ein Electron-Projekt besteht aus zwei Teilen: dem Haupt-/Main-Prozess und dem Rendererprozess. Der Haupt-/Main-Prozess ist der Prozess, der die **Node.js**-Instanz startet und die **BrowserWindow**-Instanz erstellt, welche den Rendererprozess startet. Der Rendererprozess ist der Prozess, der die Benutzeroberfläche rendert. Im Falle des Editors wird die Benutzeroberfläche mit **React** gerendert.

4.1 Haupt-/Main-Prozess

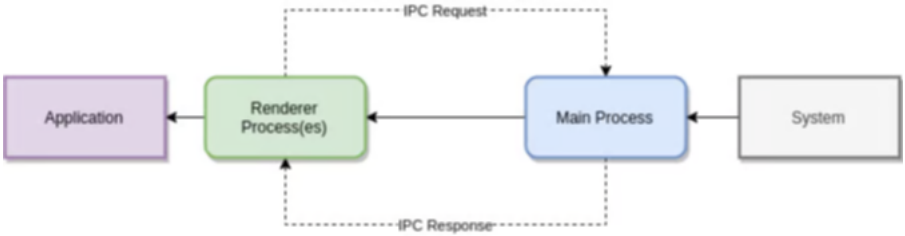
`src/main/main.ts` ist die Hauptdatei des Haupt-/Main-Prozesses.
In der Datei `src/main/main.ts` wird die **BrowserWindow**-Instanz erstellt und die **React**-Instanz gestartet. Des Weiteren werden in der Datei `src/main/main.ts` die **IPC**-Nachrichten behandelt, welche vom Rendererprozess gesendet werden.

4.1.1 IPC

IPC steht für **Inter-Process Communication**. Im Falle von Electron ist **IPC** ein Mechanismus, der es dem Hauptprozess und dem Rendererprozess ermöglicht, miteinander zu kommunizieren.

Der Hauptprozess kann Nachrichten an den Rendererprozess senden, indem die **send**-Methode der **BrowserWindow**-Instanz aufgerufen wird. Der Rendererprozess kann Nachrichten an den Hauptprozess senden, indem die **send**-Methode der **ipcRenderer**-Instanz aufgerufen wird.

Die **IPC**-Nachrichten werden in der Datei `src/main/main.ts` behandelt. Die Handler werden durch die Methode **registerHandlers** alle gemeinsam registriert.



Hier kann man gut erkennen, wie die **IPC**-Nachrichten zwischen dem Hauptprozess und dem Rendererprozess ausgetauscht werden.

Für weitere Informationen zu **IPC** siehe https://electronjs.org/docs/api/ipc-main.

4.2 Rendererprozess

`src/renderer/index.tsx` ist die Hauptdatei des Rendererprozesses.
Der Rendererprozess rendert die Benutzeroberfläche des Editors und verwendet dabei eine **React**-Instanz eines **BrowserWindow**-Objekts.

4.2.1 React

Mithilfe von **React** wird die Benutzeroberfläche des Editors gerendert. Initial wird die **React**-Instanz in der Datei `src/renderer/index.tsx` gestartet mit der Methode **run**. Die **React**-Instanz rendert die Komponente **App** in das **DOM** des **BrowserWindow**-Objekts. Jedoch, bevor die **React**-Instanz gestartet wird, wird an den Main-Prozess eine **IPC**-Nachricht gesendet (prefetch), um wichtige Daten zu erhalten, wie z.B. die Sprache des Editors, und die Einstellungen des Editors. Danach wird der Translation-Helper initialisiert und die **React**-Instanz gestartet.

4.2.2 Translation-Helper

Der Translation-Helper ist eine Klasse, die die Übersetzungen des Editors verwaltet. Er wird in der Datei `src/renderer/helper/TranslationHelper.ts` implementiert und bildet die Grundlage für die Übersetzungen des Editors. Er liefert folgende Funktionalitäten:

- **translate**: Liefert die Übersetzung für einen bestimmten String in der aktuellen Sprache.
- **translateVars**: Liefert die Übersetzung für einen bestimmten String in der aktuellen Sprache und ersetzt dabei Variablen in dem String.
- **switchLanguage**: Ändert die aktuelle Sprache des Editors.
- **getAvailableLanguages**: Liefert die verfügbaren Sprachen des Editors als ein Array von Enum-Werten.

Der Translation-Helper wird in der Datei `src/renderer/index.tsx` initialisiert und ist dann global unter **window.t** verfügbar.

4.2.3 Components

Der Editor besteht aus 5 Komponenten: **Home**, **BoardConfiguratorV2**, **RiverPresetEditor**, **GameConfigurator** und **Validator** Diese Komponenten werden in der Datei `src/renderer/App.tsx` gerendert.

4.2.3.1 Home

`src/renderer/screens/Home.tsx` ist die Datei, in der die **Home**-Komponente implementiert ist.

Die **Home**-Komponente rendert den Startbildschirm des Editors. Der Startbildschirm besteht aus 5 Buttons: **Board-Konfigurator**, **Game-Konfigurator**, **Validierer**, **Einstellungen** und **Beenden**.

4.2.3.2 BoardConfiguratorV2

`src/renderer/screens/BoardConfiguratorV2.tsx` ist die Datei, in der die **BoardConfiguratorV2**-Komponente implementiert ist.

Die **BoardConfiguratorV2**-Komponente rendert den **Board-Konfigurator** des Editors.

4.2.3.3 RiverPresetEditor

`src/renderer/screens/RiverPresetEditor.tsx` ist die Datei, in der die **RiverPresetEditor**-Komponente implementiert ist.

Die **RiverPresetEditor**-Komponente rendert den **Fluss-Vorlagen Editor** des Editors.

4.2.3.4 GameConfigurator

`src/renderer/screens/GameConfigurator.tsx` ist die Datei, in der die **GameConfigurator**-Komponente implementiert ist.

Die **GameConfigurator**-Komponente rendert den **Game-Konfigurator** des Editors.

4.2.3.5 Validator

`src/renderer/screens/Validator.tsx` ist die Datei, in der die **Validator**-Komponente implementiert ist.

Die **Validator**-Komponente rendert den **Validierer** des Editors.