

deck.glを利用した、マップ上に移動体の情報を表示するライブラリの開発について

目次

1. 開発中のライブラリ (Harmoware-VIS) の紹介
2. React, Reduxの説明
3. deck.glの紹介
4. Harmoware-VISを使ってみる

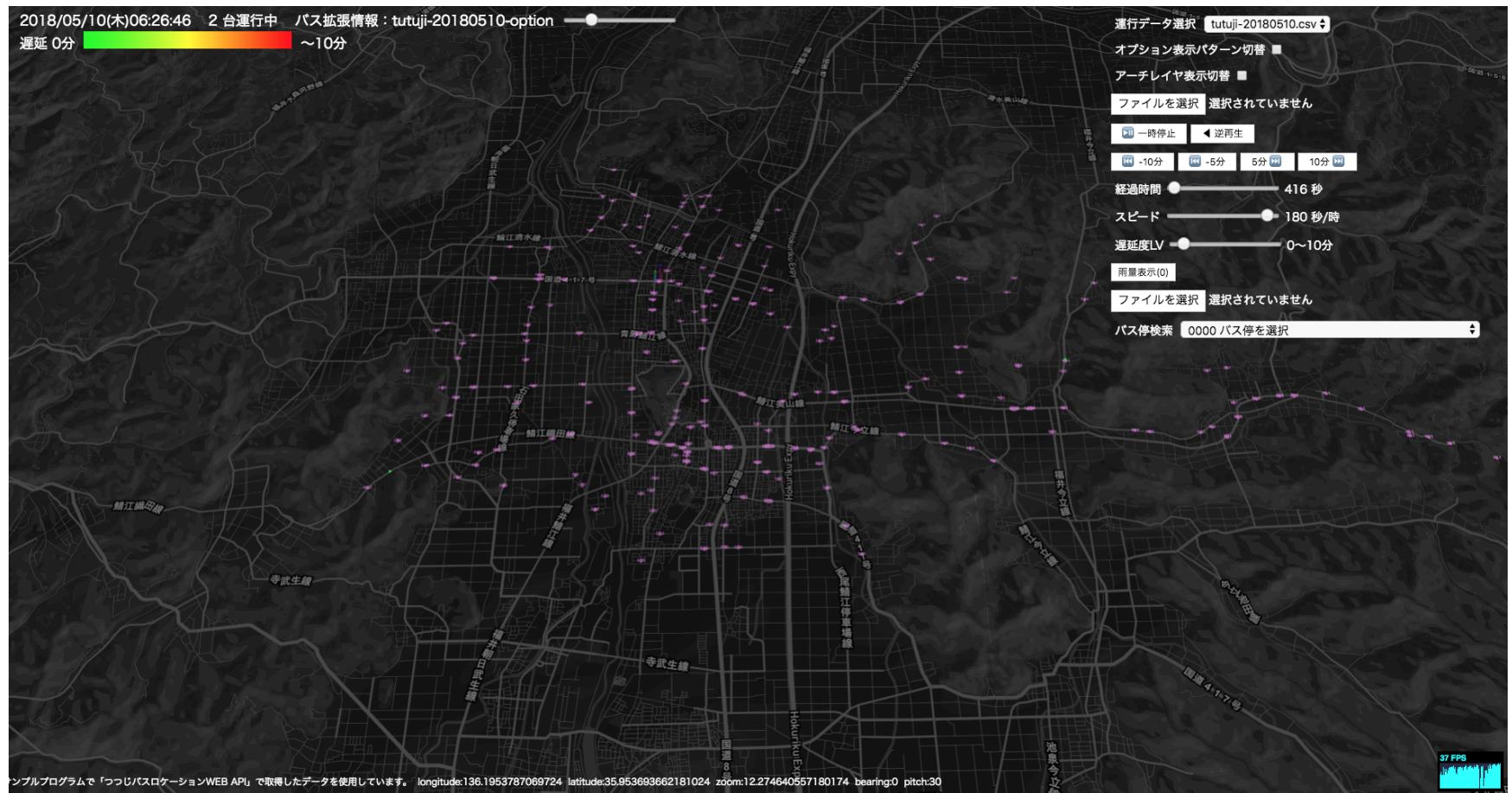
1. 開発中のライブラリ (Harmoware-VIS) の紹介

JavaScriptライブラリである、React Redux deck.glを使って、WebGL上で移動体の情報可視化や管理などを行うためのライブラリーで、主に名古屋大学河口研究室が中心になって開発しています。
先ほど紹介した以下のライブラリに依存しています。

- react-map-gl
- [luma.gl](#)
- [deck.gl](#)

1. 開発中のライブラリ (Harmoware-VIS) の紹介

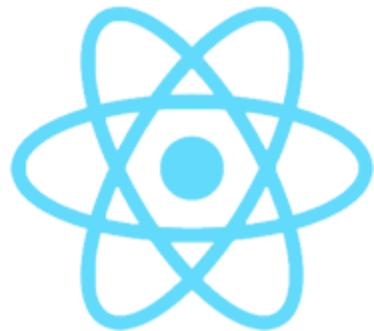
Harmoware-VIS の実行画面



2. React, Reduxの説明

Reactとは

- 平たく言えば、テンプレートエンジン
- 仮装DOMを利用したHTMLの差分更新が特徴



React

Reactとは

他の似たようなUI系ライブラリとして以下があります。

- Vue.js (<https://vuejs.org/>)
- Riot.js (<https://riot.js.org/>)
- Angular.js (<https://angularjs.org/>)
- Polymer (<https://www.polymer-project.org/>)



ではなぜReactなのか？

- あくまでViewの差分更新のみの機能に特化している
- ES2015以降の知識とJSXの知識さえあれば実装できる。

Reactを実行してみよう

```
git clone https://github.com/steelydylan/react-sample.git .
npm install
npm run start
```

Reactを実行してみよう

出力結果

増加ボタンを押すことで、数字が1増加し、減少ボタンを押すことで1減少します。

カウント数: 0

増加 減少

Reactを実行してみよう

カウント数を0にするボタンをSampleコンポーネントに追加してみましょう。

1. renderメソッド内にボタンを追加

```
<button onClick={this.reset.bind(this)}>リセット</button>
```

Reactを実行してみよう

2. resetメソッドを定義

```
...
reset() {
  this.setState({
    count: 0
  });
}
...
```

Reactを利用する際に便利なJSX記法

JSXはReactを完結に記述するためにFacebook社によって開発された記法です。

HTMLに似た書き方ですが、`class` を `className` と書いたり、若干、HTMLとは記述がことなります。

Reactを利用する際に便利なJSX記法

JSXの記法例

<https://github.com/steelydylan/react-sample/blob/master/src/index.js>

Reactを利用する際に便利なJSX記法

属性に代入する値は文字列の場合はそのまま " " で囲うことができるが、数字や変数などは {} で囲う必要があります。 style はオブジェクトで記述する点にご注意ください。

```
const Modal = <div className="modal" tabIndex={-1} style={{display:'none'}}>
  <div className="modal-inner">
    <div className="modal-header">
    </div>
    <div className="modal-body">
    </div>
  </div>
</div>
```

Reactのライフサイクル

また、Reactにはライフサイクルというものがあり、ライフサイクルを覚えておくと、コンポーネントの生成時や削除時に処理を挟むことができます。

Reactのライフサイクル

componentWillMount()

コンポーネントがhtml上に出力される寸前の処理

Reactのライフサイクル

componentDidMount()

コンポーネントがhtml上に出力された直後の処理

Reactのライフサイクル

componentWillReceiveProps(props)

コンポーネントが親コンポーネントから値を受け取る直前の処理。引数には受け取る予定の `props` の値が入っている。

Reactのライフサイクル

componentWillUnmount()

コンポーネントがhtml上に出力されなくなった時の処理

Reactの導入

パッケージをインストール

```
npm install react react-dom --save
```

Reactの導入

JSXをサポートするための文法プリセットをインストール

```
npm i babel-preset-react --save-dev
```

.babelrcの設定

```
{  
  "presets": ["env", "react"]  
}
```

create-react-appを利用

create-react-appとは内部にwebpackを内包した、react環境を素早く構築するためのコマンドラインツール。

npmでインストール可能。

webpack初心者でも導入が簡単

create-react-appの実行

```
npm install create-react-app -g  
create-react-app パッケージ名  
yarn start
```

付隨するReact周辺ツールの知識

フロントエンド開発者はReact以外にも様々なツールを知っておくことで開発の幅が広がります。

<https://github.com/adam-golab/react-developer-roadmap>

- Redux（アプリケーションの状態を管理するためのツール）
- jest（Reactに特化したJavaScriptのユニットテストツール）
- GraphQL（REST APIに代わる新たなAPI サーバーへの問い合わせのためのデータクエリ言語）

などなど

Reduxとは

アプリケーションの状態（State）を環境に左右されず一貫したルールのもと管理するための仕組みです。



Reduxとは

Reduxを理解するのに一番重要なのが `Store` という概念です。

`Store` はアプリケーションの状態である `State` を保持し、そのストートは `Action` というチケットを介して、`Reducer` という関数が実行され、その結果が新しい `State` として再び `Store` にセットされます。



Redux導入方法

```
npm install redux --save
```

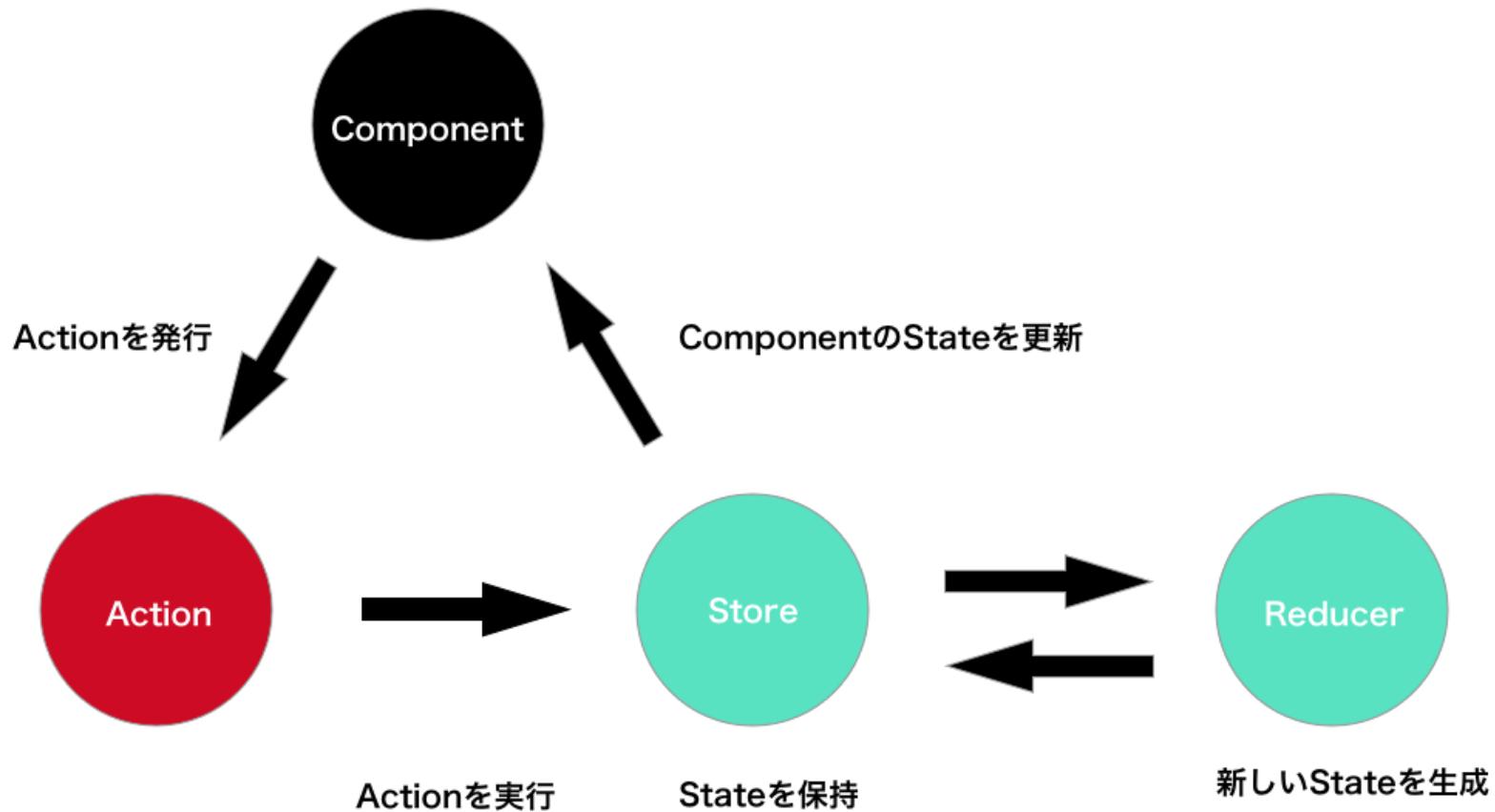
Redux導入方法

```
npm install redux --save
```

ReactとReduxを連携する

ReduxのState管理の仕組みとReactのコンポーネントを紐づけるために `react-redux` というツールを使います。 `react-redux` を使うことによって、 `Store` の状態を `Component` に紐づけたり、 `Component` から `Action` を実行することができます。

ReactとReduxを連携する



ReactとReduxを連携する

インストール方法

```
npm install react-redux --save
```

ReactとReduxを連携する

下記のコードでは `mapStateToProps` `mapDispatchToProps` を `connect` することで、`Reducer` の `state` や `action` をAppコンポーネントのpropsで参照可能にしています。

<https://gist.github.com/steelydylan/6654d72c6c95f5b9a1fbeaa209e6d280>

ReactとReduxを連携する

連携後は、`connect`されたコンポーネントから、
`this.props.state`、`this.props.actions.action()`のような形で
呼び出すことができます。

React Redux 連携サンプル

<https://github.com/steelydylan/react-redux-sample>

```
git clone https://github.com/steelydylan/react-redux-sample.git .
npm install
npm run start
```

React Redux 連携サンプル

実行画面

カウント数: 0

増加 減少

React Redux 連携サンプル

現在、増加ボタンと減少ボタンがあり、ボタンをクリックすると、値が ± 1 増減します。さらに、RESETというアクションを追加して、値を0に戻すボタンを追加してみましょう。

React Redux 連携サンプル

1. constants/ActionTypes.js で RESET を定数として追加

```
export const RESET = 'RESET';
```

2. actions/index.js で reset をアクションとして追加

```
export const reset = () => ({ type: types.RESET });
```

3. reducers/index.js で RESET に対する処理を記述

```
...
case types.RESET:
  return Object.assign({}, state, { count: 0 });
...
```

4. components/sample.js でアクションを実行するためのボタンを追加

```
<button onClick={() => { this.props.reset(); }}>  
  リセット  
</button>
```

4. deck.glの紹介

deck.glの紹介

WebGLベースの地理情報視覚化用のコンポーネント集
 Reactとの連携が可能。



deck.gl 導入方法

以下の3つのライブラリーが必要

- react-map-gl
- luma.gl
- deck.gl

```
npm install deck.gl luma.gl react-map-gl --save
```

react-map-gl

OpenStreetMapのライブラリであるmap-glをReactベースで使えるようにしたライブラリ。

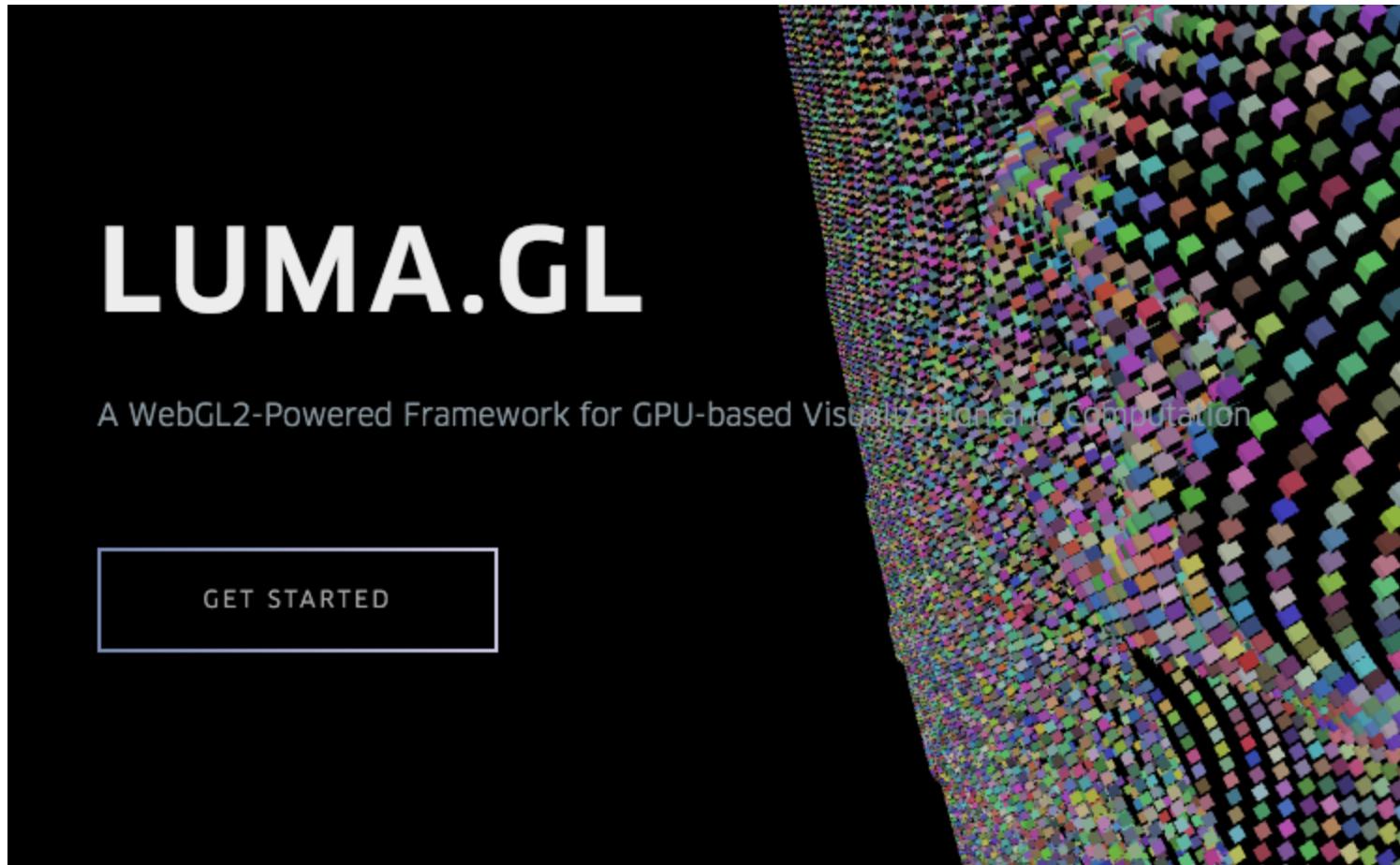
Uber社が開発



luma.gl

Uber社が開発した、 WebGL用データビジュアライゼーションライブラリー。

これを利用して、 deck.glが動作します。



記述例

```
import DeckGL, {ArcLayer} from 'deck.gl';

const flights = new ArcLayer({
  data: [] // Some flight points,
  ...
});

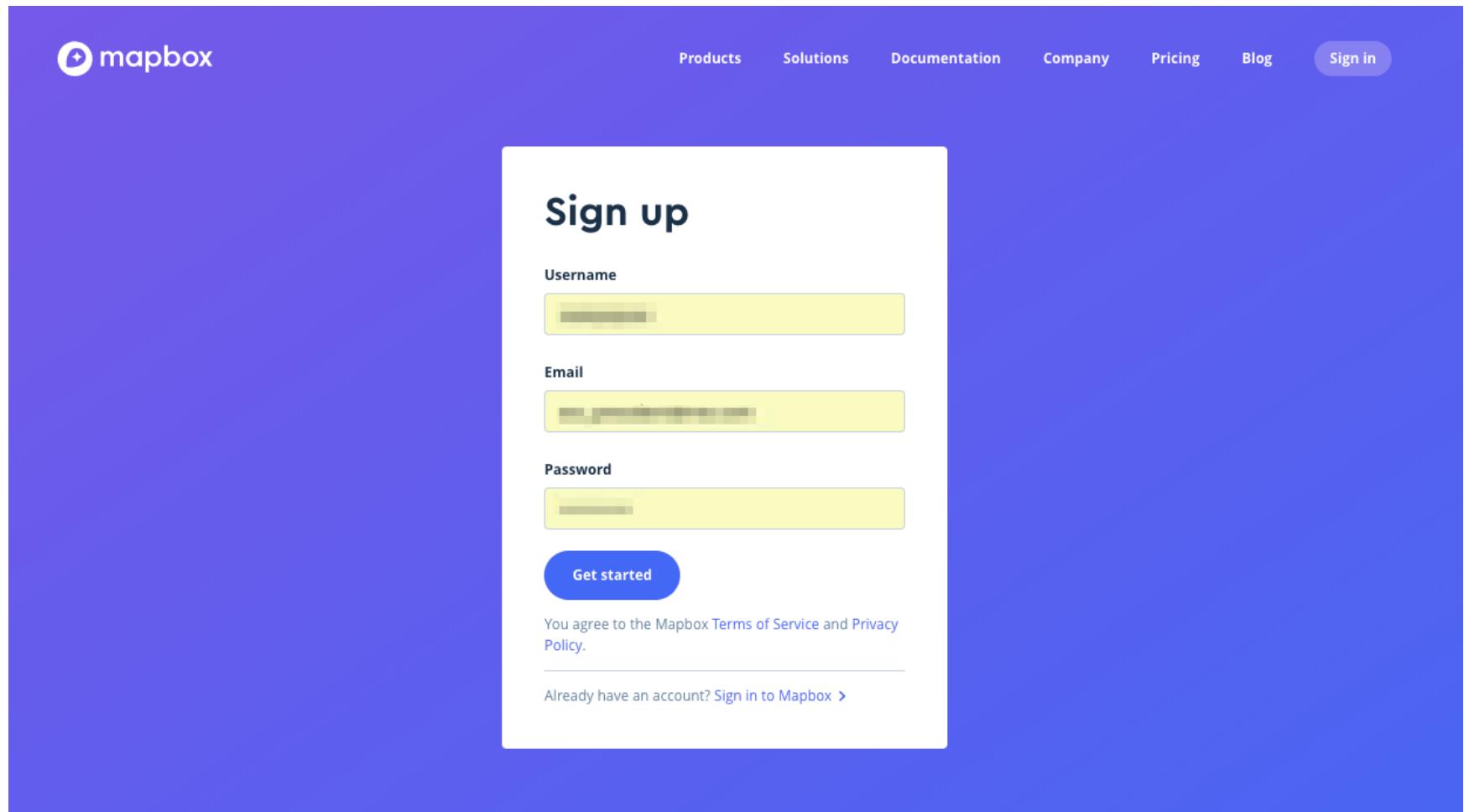
<MapGL
  {...viewport} mapStyle={mapStyle} perspectiveEnabled
  onChangeViewport={onChangeViewport}
  mapboxApiAccessToken={mapboxApiAccessToken}
>
  <DeckGL width={1920} height={1080} layers={[flights]} />
</MapGL>
```

5. Harmoware-VISを使ってみる

Harmoware-VISを使ってみる

まずは、mapboxに登録してアクセストークンを取得する必要があります。

<https://www.mapbox.com/signup/?plan=paygo-1>



Harmoware-VISを使ってみる

登録後は下記のURLより アクセストークン を取得できます。

<https://www.mapbox.com/install/>

Add Mapbox to your app or website

Install the Maps SDK

Install beautiful interactive maps in your app or website.



iOS



Android

JS

Web



Unity

Get your access token

Just looking for your [access token](#)?

Here it is!



Design a map style

Design a [map style](#) with your own data and visual appearance.

[Design a map style →](#)

Harmoware-VISを使ってみる

取得したアクセストークンは環境変数として登録しておきます。

```
set MAPBOX_ACCESS_TOKEN=XXXXXXXXXX
```

Harmoware-VISを使ってみる

まずは `harmoware-demo` というディレクトリーを作成して、そこに `package.json` を作成します。

```
mkdir harmoware-demo  
cd harmoware-demo  
npm init
```

package.jsonの編集

以下のようなpackage.jsonを作成しましょう。

<https://github.com/steelydylan/harmoware-demo/blob/master/package.json>

Harmoware-VISはまだ npm に公開されていませんので、packageの場所が github となることに注意してください。

webpack.config.jsを作成

<https://github.com/steelydylan/harmoware-demo/blob/master/webpack.config.js>

依存関係を解決するために `mapbox-gl` のエイリアスだけ作っておくことに注意してください。

.babelrcの作成

<https://github.com/steelydylan/harmoware-demo/blob/master/.babelrc>

babel の設定は react と env があれば充分です。さらに、
transform-object-rest-spread があればオブジェクトの受け渡しの
際に便利かもしれません。

src/index.jsの作成

<https://github.com/steelydylan/harmoware-demo/blob/master/src/index.js>

Harmoware-VISでは react , redux を使用して、 action および state を管理しています。

src/containers/app.jsの作成

<https://github.com/steelydylan/harmoware-demo/blob/master/src/containers/app.js>

connectToHarmowareVis 関数を使ってコンポーネントにHarmoware-VISの state および action を container コンポーネントに紐付けます。

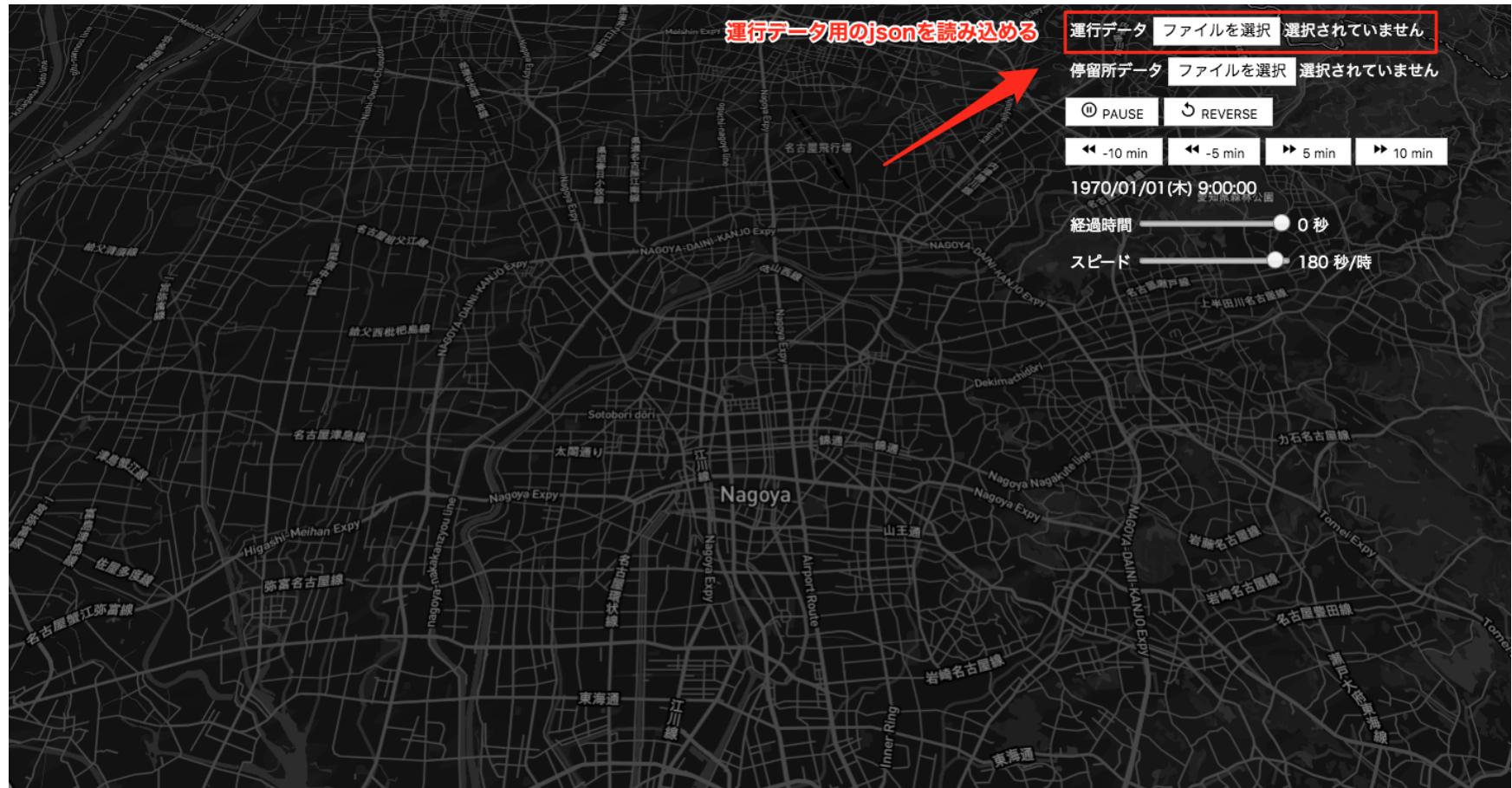
Harmoware-VISの実行

```
npm install  
npm run start
```

Harmoware-VISの実行



Harmoware-VISの実行

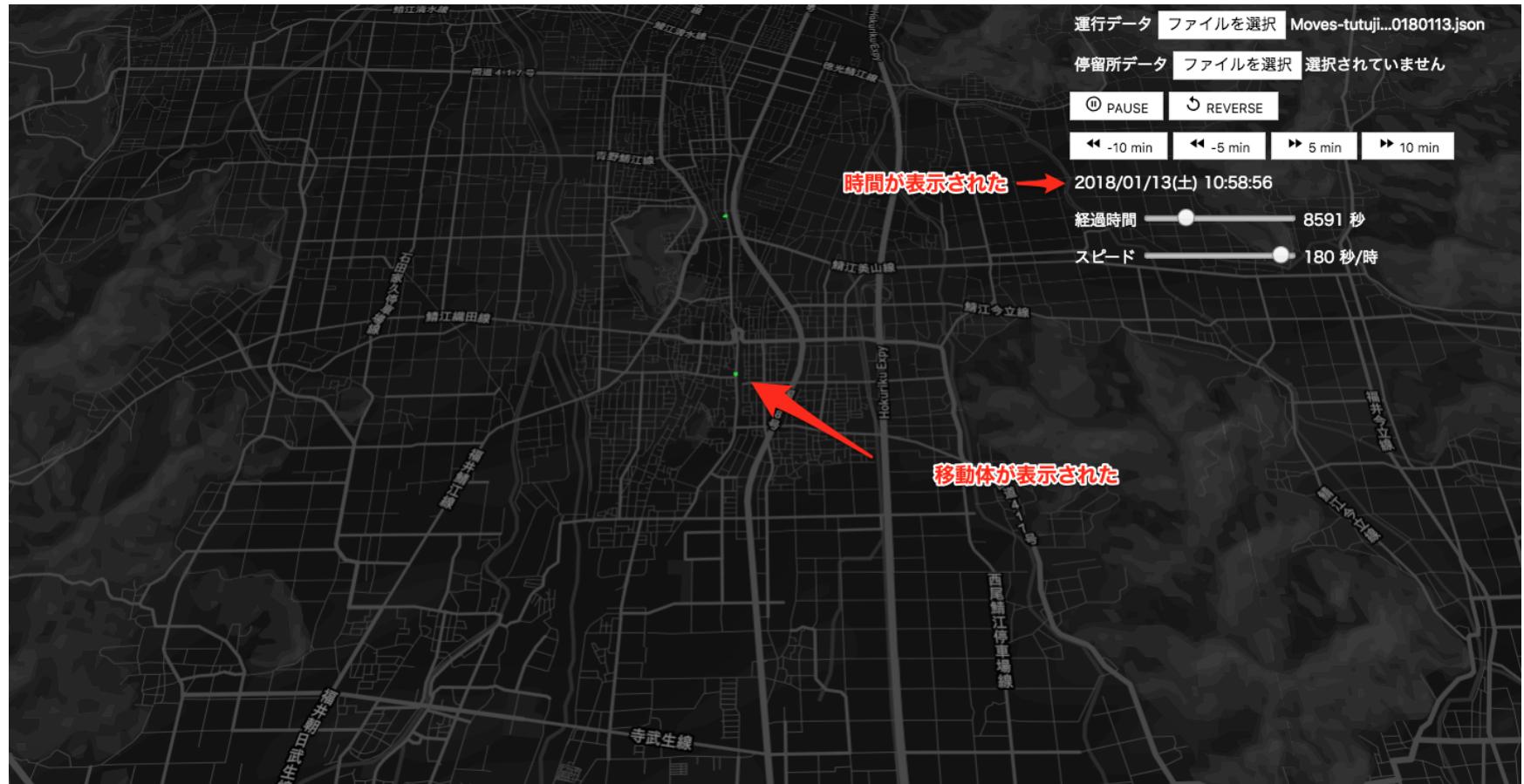


運行データのjsonを読み込むことができます。

以下のjsonファイルを読み込んでみましょう。

<https://raw.githubusercontent.com/steelydylan/harmoware-demo/master/json/Moves-tutuji-20180113.json>

Harmoware-VISの実行



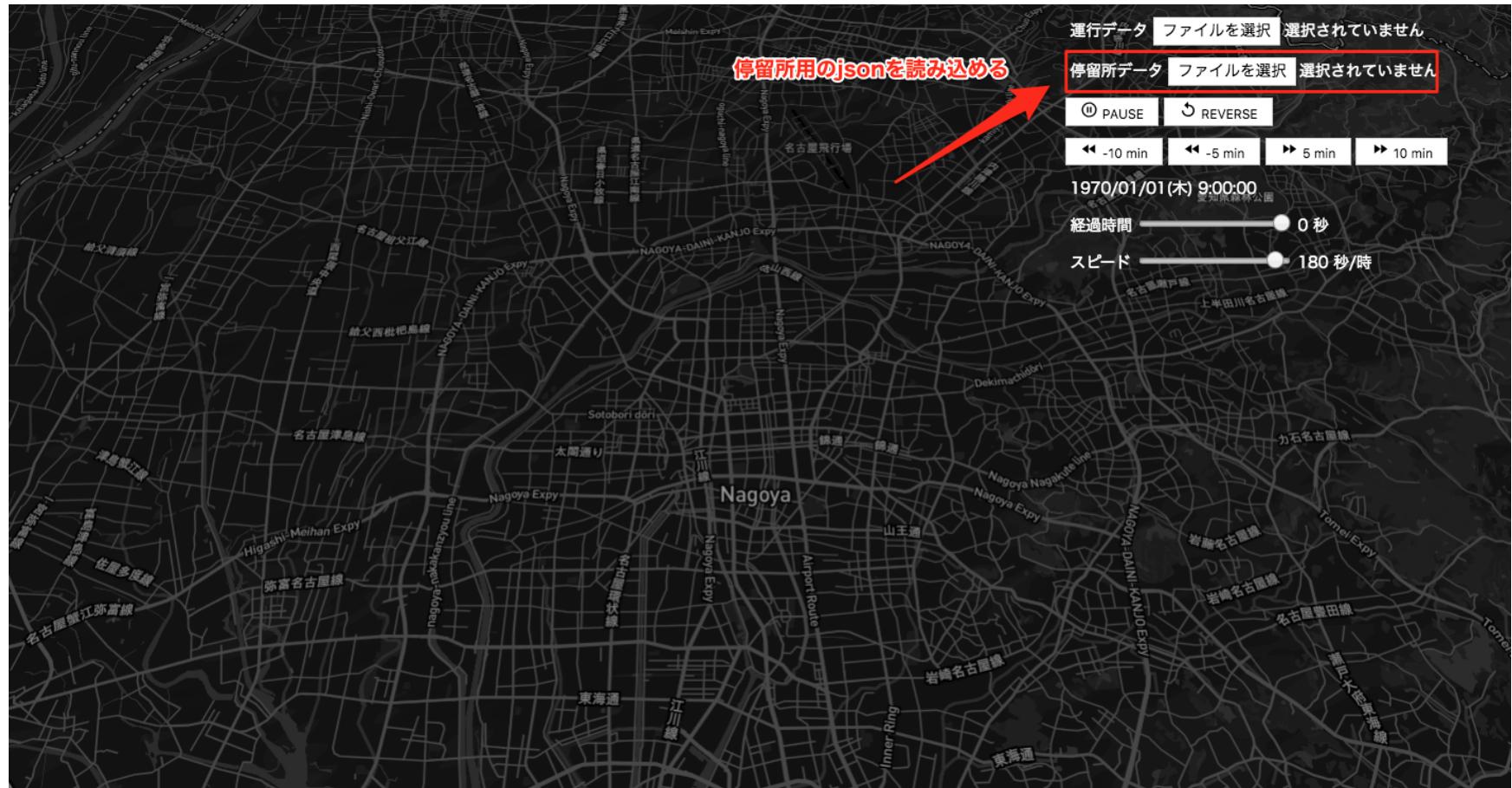
マップ状に移動体が表示され、どの時間にどの位置に移動体がいるのか
が可視化されました。

Harmoware-VISの実行

運行データのJSON形式

<https://github.com/Harmoware/Harmoware-VIS/#運行シミュレーションデータファイルのjsonフォーマット>

Harmoware-VISの実行

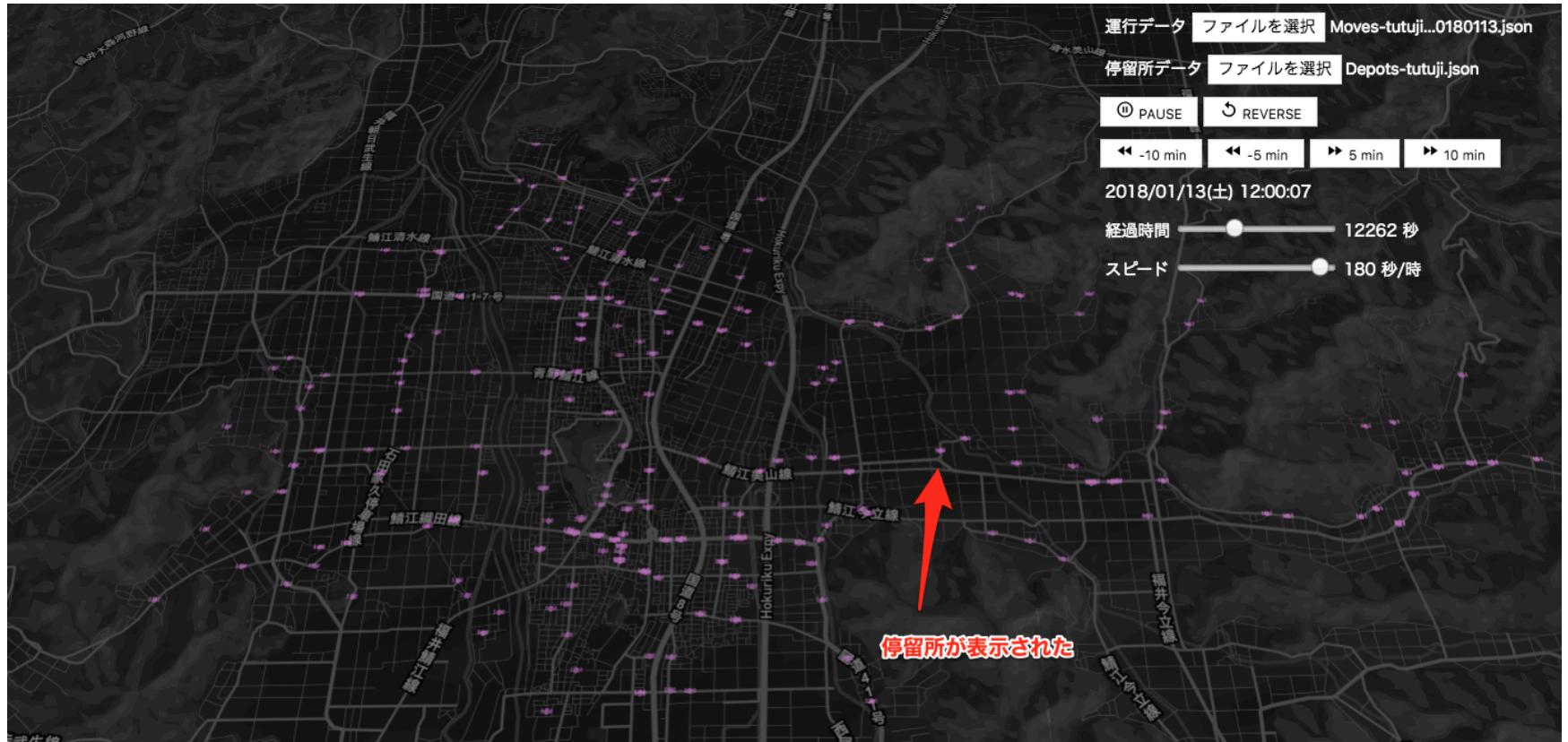


停留所データのjsonを読み込むことができます。

以下のjsonファイルを読み込んでみましょう。

<https://raw.githubusercontent.com/steelydylan/harmoware-demo/master/json/Depots-tutuji.json>

Harmoware-VISの実行



移動体の他に、その移動体が停止する停留所（バス停など）が表示されました。

Harmoware-VISの実行

停留所データのJSON形式

[https://github.com/Harmoware/Harmoware-VIS/#停留所情報データの
jsonフォーマット](https://github.com/Harmoware/Harmoware-VIS/#停留所情報データのjsonフォーマット)

Harmoware-VISの実行

mouseover 時に移動体の情報を表示する

また、MovesLayer, DepotsLayerは onHover の callback より
mouseover 時にhoverされたオブジェクトの情報を取得することができます。

```
new MovesLayer({ routePaths, movesbase,  
  movedData, clickedObject, actions,  
  onHover: (el) => {console.log(el)}  
});
```

Harmoware-VIS レイヤー一覧

また、Harmoware-VISではいくつかのLayerをデフォルトで用意しています。

<https://github.com/Harmoware/Harmoware-VIS#harmoware-vis-layers>

各レイヤーの import

```
import {
  MovesLayer,
  MovesNonmapLayer,
  DepotsLayer,
  FixedPointLayer,
  LineMapLayer
} from 'Harmoware-VIS';
```

MovesLayer

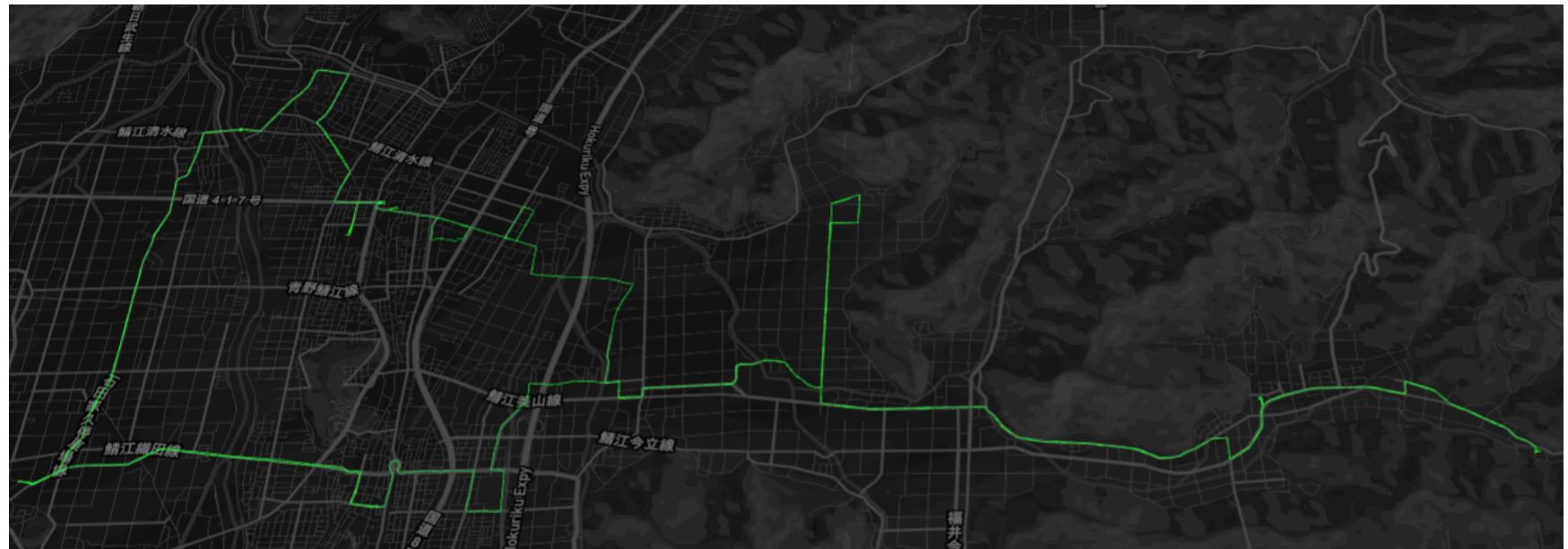
車両などの移動体をmapboxより取得したマップ上に表示します。下の図の緑の円が MovesLayer で表示されているオブジェクトです。

```
import { HarmovisLayers, MovesLayer } from 'harmoware-vis';
...
<HarmovisLayers layers={[new MovesLayer({
  routePath,
  movesbase,
  movedData,
  clickedObject,
  actions
})]}/>
```



MovesLayer

クリックすることで、その移動体の辿る経路が緑のライン上で表示されます。

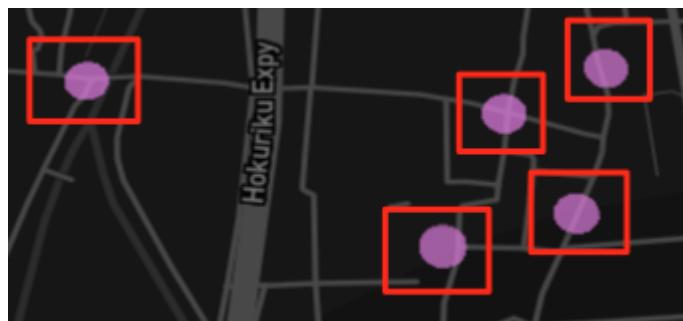


DepotsLayer

停留所や駅などをmapboxより取得したマップ上に表示します。下の図のピンクの円が DepotsLayer で表示されているオブジェクトです。

```
const {depotsData} = this.props;

<HarmoVisLayers
  layers={[
    new DepotsLayer( { depotsData } )
  ]}
/>
```



Harmoware-VIS コントロール用のコンポーネント一覧

Harmoware-VISでは自身のState更新用のコンポーネントをいくつか提供しています。

<https://github.com/Harmoware/Harmoware-VIS#harmoware-vis-control-component>

各コンポーネントの `import`

```
import { MovesInput, DepotsInput,
  LinemapInput, AddMinutesButton,
  PlayButton, PauseButton, ForwardButton,
  ReverseButton, ElapsedTimeRange, SpeedRange,
  SimulationDateTime
} from 'Harmoware-VIS';
```

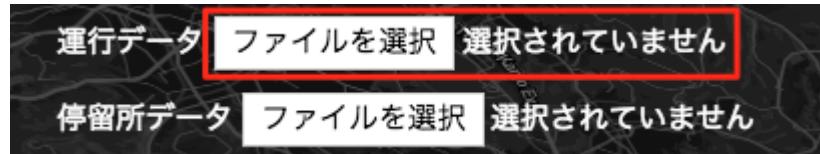
MovesInput

移動体データを地図上に可視化するためのファイルを読み込むコンポーネント。

「運行シミュレーションデータ」を設定したファイルを選択するダイアログを表示し、読み込んだデータより Harmoware-VIS の `bounds`、
`timeBegin`、`timeLength`、`movesbase` を設定します。

ここからjsonを読み込むことにより、`MovesLayer` が表示されます。

```
<MovesInput actions={actions} />
```



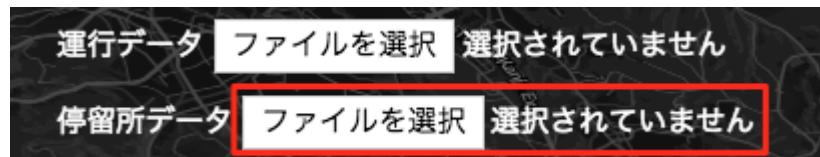
DepotsInput

停留所データを地図上に可視化するためのファイルを読み込むコンポーネント

「停留所情報データ」を設定したファイルを選択するダイアログを表示し、読み込んだデータより Harmoware-VIS の depotsBase に設定します。

```
<DepotsInput actions={actions} />
```

ここからjsonを読み込むことにより、 DepotsLayer が表示されます。



Harmoware-VISの State

以下が、Harmoware-VISが持っている state です。

action を介して、state を更新することができます。

<https://github.com/Harmoware/Harmoware-VIS#harmoware-vis-reducer>

Harmoware-VISのaction

Harmoware-VISでは以下のactionをデフォルトで用意しています。

<https://github.com/Harmoware/Harmoware-VIS#harmoware-vis-actions>

containerからstateの取得

例) Harmoware-VISの現在の時間を取得する

```
const { settime } = this.props;
```

Harmoware-VISのプロジェクト構成

基本的に、Harmoware-VISのプロジェクト構成は型定義のための `types` ディレクトリーや、レイヤーのための `layer` ディレクトリー、ユーティリティのための `library` ディレクトリーが存在する点を除いて、`redux` のスタンダードな構成に従っています。

The screenshot shows the GitHub repository page for 'Harmoware / Harmoware-VIS'. The repository has 3 issues, 0 pull requests, 0 projects, and a wiki. It has 9 stars and 4 forks. The commit history for the 'master' branch is displayed, showing the following commits:

| Commit | Message | Date |
|--------|---|-----------------------------------|
| | refactor getcontainerprop method | Latest commit 007bc93 4 hours ago |
| .. | | |
| | webpackを最新に更新、babelの設定などの見直し | 11 days ago |
| | srcコンポーネントのstyleをclass指定可能に変更 | 6 days ago |
| | XbandmeshLayer関連コードをharmoware-VISからbus3dサンプルに移動 | 20 days ago |
| | webpackを最新に更新、babelの設定などの見直し | 11 days ago |
| | nonmapmoveで描画したrouteが消えない件の対策 | 7 days ago |
| | refactor getcontainerprop method | 4 hours ago |
| | XbandmeshLayer関連コードをharmoware-VISからbus3dサンプルに移動 | 20 days ago |
| | eslintエラー対応 | 7 days ago |
| | export default reducer | 10 hours ago |

Harmoware-VISのプログラム構成

Harmoware-VIS 本体は拡張されて使用されることが前提にあるので、前述の actions や reducers 、 components 、 constants などの設定は全て Harmoware-VIS のエントリーポイントで export しています。

<https://github.com/Harmoware/Harmoware-VIS/blob/master/src/index.js>

従って、Harmoware-VISをライブラリとして利用する際は、以下のように actions や reducers に関わらず、すべからく同一の方法でimport 可能です。

```
import {Actions, MovesInput, DepotsInput,  
LinemapInput, PlayButton, settings, Container, MovesLayer,  
DepotsLayer, connectToHarmowareVis, getCombinedReducer, reducer}  
from 'harmoware-vis';
```

Harmoware-VIS の拡張

Harmoware-VISでは `redux` を使用しており、移動体や停車場の情報管理のための `store` をデフォルトで用意しています。Harmoware-VISでは、この `store` を拡張するための機能が用意されています。`store` を拡張することで、移動体や停留所以外の情報も管理できるようになります。今回は下のデモのようにバンクーバーの地域ごとの時価をグラフ表示できるようにしてみましょう。

<http://deck.gl/showcases/gallery/geojson-layer>

以下のjsonデータを読み込み、表示できるようにします。

<https://raw.githubusercontent.com/uber-common/deck.gl-data/master/examples/geojson/vancouver-blocks.json>

Harmoware-VIS の拡張

1. src/reducer/geo.jsの作成
2. action及びconstantsの作成
3. baseのreducerと結合
4. 結合したreducerの使用
5. geo情報を読み込むためのコンポーネントを設置
6. containerとstoreの接続

1. src/reducer/geo.jsの作成

まずは、jsonデータをstateとして保持するためのreducerを作成します。

今回は geo.js としています。

<https://github.com/steelydylan/harmoware-demo2/blob/master/src/reducers/geo.js>

2. action及びconstantsの作成

先ほど作成した `geo.js` を活用するためにそれ用の `action` と `constants` を定義します。

`src/constants/index.js`

```
export const SETGEO = 'SETGEO';
```

`src/actions/index.js`

```
import * as types from '../constants';

export const setGeo = (geo) => ( {
  type: types.SETGEO, geo
});
```

3. baseのreducerと結合

`getCombinedReducer` という `harmoware-vis` で用意されたAPIを使用して自作reducerとHarmoware-VISのreducerを結合できます。

```
import geo from './geo';
import { getCombinedReducer } from 'harmoware-vis';

export default getCombinedReducer({ geo });
```

4. 結合したreducerの使用

先ほど合体したreducerをimportして store を作成します。
react-reduxモジュールから import した Provider コンポーネントで
container (App) を囲います。

```
import App from './containers';
import { Provider } from 'react-redux';
import reducer from './reducers';
const store = createStore(reducer);

render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('app')
);
```

5. geo情報を読み込むためのコンポーネントを設置

次に実際にファイルを読み込み `action` を実行するためのコンポーネントを作成します。

この時に、`actions.setGeo()` だけではなく、読み込まれた位置に `viewport` を移動するために、`actions.setViewport()` も実行すると親切です。

<https://github.com/steelydylan/harmoware-demo2/blob/master/src/components/geo-layer-input.js>

6. containerとstoreの接続

先ほどの `geo-layer-input.js` を含んだ `container` コンポーネントを作成し、`connectToHarmowareVis` メソッドで、`container` と Harmoware-VIS を結びつけます。

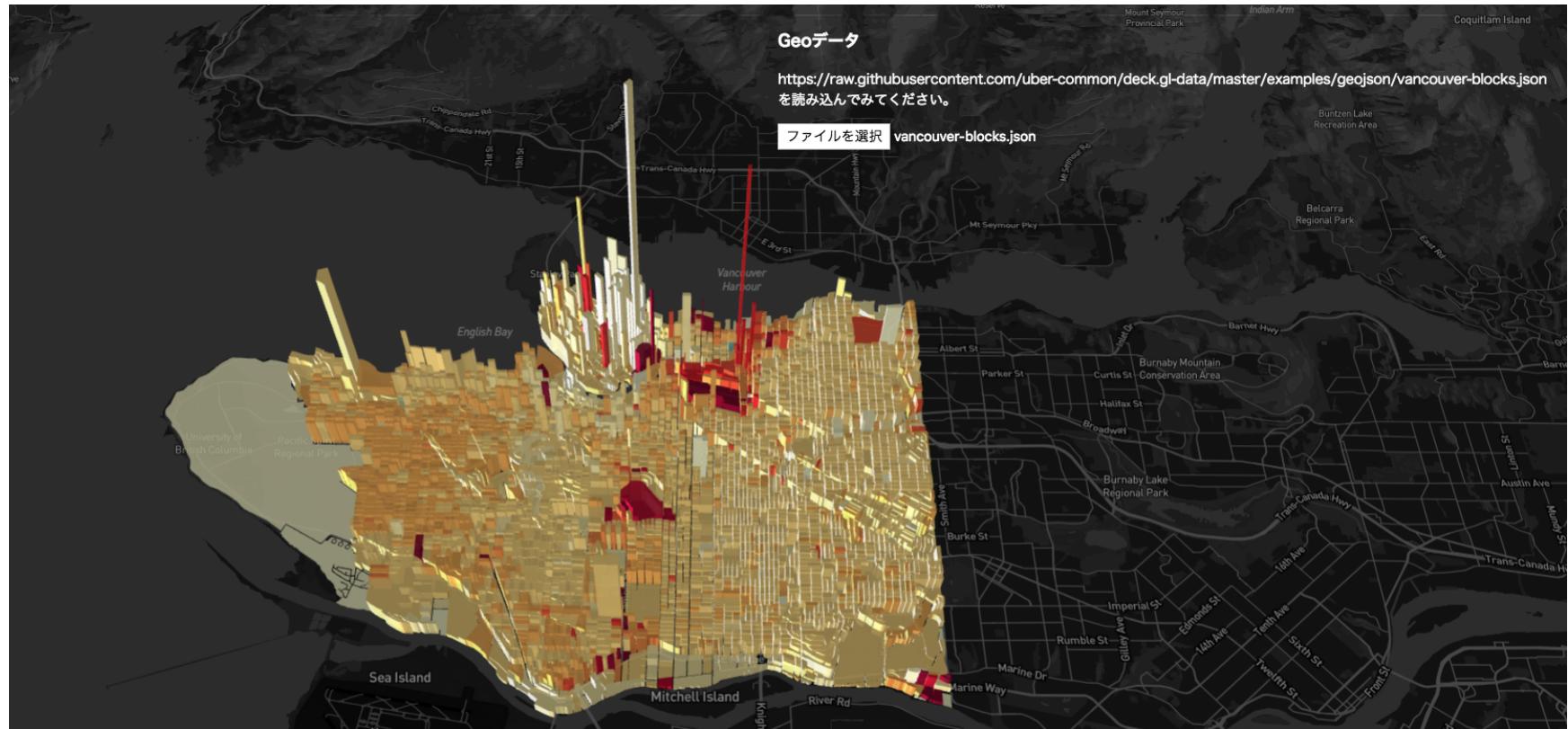
<https://github.com/steelydylan/harmoware-demo2/blob/master/src/containers/app.js>

以下のように json を読み込むとバンクーバーの時価情報が表示されれば成功です。

このように、Harmoware-VISを拡張するには redux の知識が必要になってきます。

実際のソースコードはこちらにおいてあります。

<https://github.com/steelydylan/harmoware-demo2/>



以上で、JavaScriptの入門から Harmoware-VIS の使い方、カスタマイズ方法までのチュートリアルは終了です。Harmoware-VIS のさらに詳しいカスタマイズ方法は以下の3つのサンプルを見ていただくと参考になるかと思います。

- <https://github.com/Harmoware/Harmoware-VIS/tree/master/examples/bus3d>
- <https://github.com/Harmoware/Harmoware-VIS/tree/master/examples/visualize-sample-nonmap>
- <https://github.com/Harmoware/Harmoware-VIS/tree/master/examples/visualize-sample>

今後の課題

- npmへの公開
- i18n対応
- かっこいいスタイルをつける！

などなど！