How to use WebAssembly with GeoGebra

藤本 光史（福岡教育大学）*

概要

ユーザーインタフェースに GeoGebra と React を、計算エンジンに C 言語と WebAssembly を用いて Web アプリを実装する手法について解説する。React と WebAssembly を連携させるにはいくつかの配慮が必要である。それらを数学パズルのライツアウトアプリの実装を通して説明する。

1 はじめに

2021 年の本研究会において、筆者は React を用いて GeoGebra を Web アプリ化する方法について解説した [1]。これは GeoGebra の JavaScript 利用環境の改善を目的としたものであり、`react-geogebra` [2] による GeoGebra コンポーネントと HTML によるボタンを配置した非常にシンプルなユーザーインタフェース（UI）の Web アプリであった。

React では Material UI [3] や Bootstrap [4] などのリッチな UI コンポーネントが利用可能である。また、Web アプリであれば他の様々な Web 技術と連携することが可能である。ここでは、ユーザーインタフェースの作成に Material UI と GeoGebra を利用し、C 言語で作成した関数を計算エンジンとして用いるために WebAssembly を利用する手法を解説する。具体的に、数学パズルであるライツアウトの実装過程を通して本手法の詳細について述べる。

2 ライツアウトとは

ライツアウトとは、５×５のセルの一部が点灯した状態からスタートして、全てのセルを消灯状態にすることが目的の数学パズルである。セルをクリックすると、そのセルを含む上下左右のセルの点灯状態が反転する。ただし、盤面の端にある上下左右のいずれかが欠けたセルをクリックした場合は、欠けたセルは無視される。

* fujimoto@fukuoka-edu.ac.jp

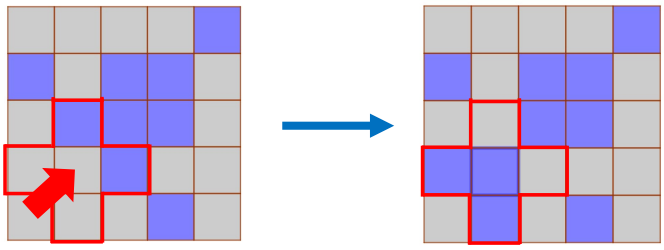

図１　ライツアウトの点灯状態の変更例

ライツアウトは５×５のタイプが最もよく知られているが、３×３、４×４、６×６など様々なバージョンがある。３×３や６×６はどんなパターンも可解でその解は１個であるが、４×４や５×５には非可解なパターンがあり、可解なパターンの解は複数個ある。最終的にセルが点灯するかどうかは、クリックで変化する部分の重ね合わせの偶奇性で定まる。このことからライツアウトには以下の性質があることがわかる。

- 各セルのクリック回数は高々１回である。
- セルをクリックする順番が異なっても結果は同じになる。

3　ライツアウトの UI

ここで作成する UI は以下である。

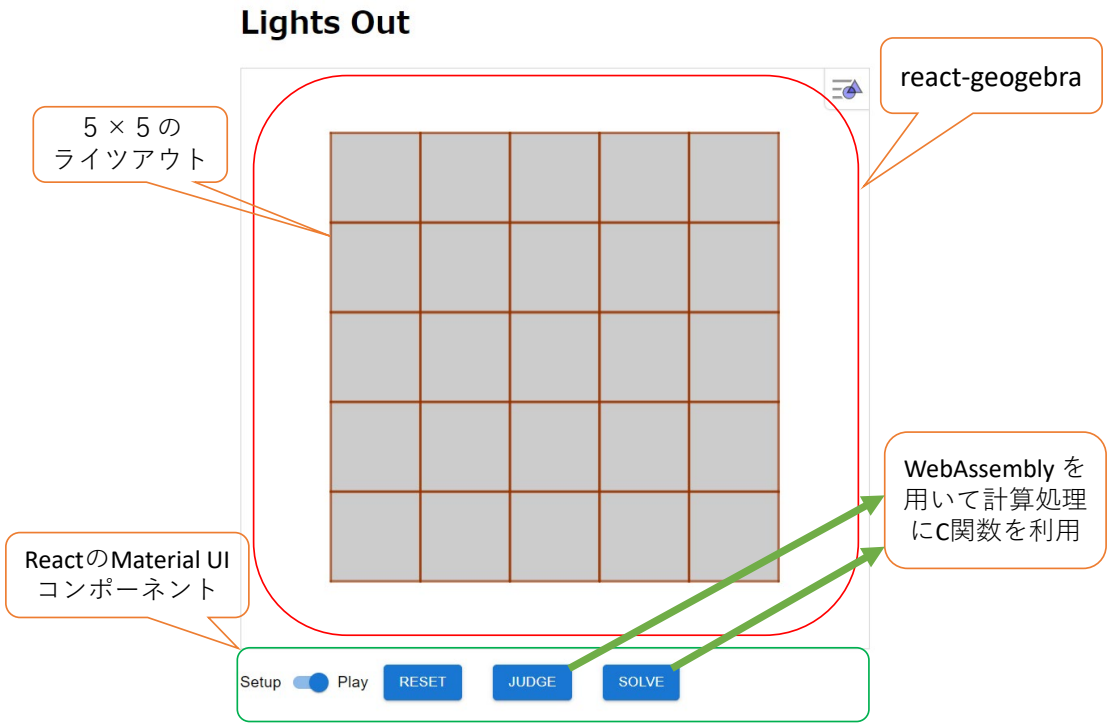

図２　ライツアウトの UI

GeoGebra コンポーネント内では、JavaScript 用 API を使用して５×５のセルを生成し、イベントリスナーを用いてクリックによりセルの点灯状態を変更できるようにする。また、Material UI コンポーネントを用いて、問題作成時の「Setup」とゲームプレイ時の「Play」のモード切り替えをトグルスイッチで実現する。この他に、初期化 (Reset) ボタン、判定 (Judge) ボタン、求解 (Solve) ボタンを配置する。

3.1 React テンプレートとコンポーネントの準備

ここで作成するアプリは React ベースにするため、プロジェクト生成ツール `create-react-app` で React テンプレートを作成する[*1]。

```
$ cd ~/react
$ npx create-react-app my-app
```

[1] を参考にして、不要なファイルを削除し、最小構成の React テンプレートを作る。さらに、GeoGebra コンポーネントと Material UI コンポーネントを利用するために以下を実行する。

```
$ cd my-app
$ npm install --save --legacy-peer-deps react-geogebra
$ npm install @mui/material
$ npm install @emotion/react
$ npm install @emotion/styled
```

3.2 React コンポーネントの設置

React の UI の要である `src/App.js` ファイルを以下のように作成する。

コード 1　`App.js`

```
1  import Geogebra from 'react-geogebra';
2  import React from 'react';
3  import { Stack, Typography, Switch, Button, Snackbar, Alert } from '@mui/material';
4
5  function App() {
6    const [mode, setMode] = React.useState(true);
7    const handleChange = (event) => {
8      setMode(event.target.checked);
9    };
10   const stateMode = React.useRef(); // ステート取得のためにオブジェクトを作成
11   stateMode.current = mode; // modeのステートをcurrentに保持
12   return (
```

[*1] ここでの作業は WSL（Windows Subsystem for Linux）を用いる。WSL の環境構築は [5] の第 4 章を参照のこと。

```
13      <div className="App">
14        <h1>Lights Out</h1>
15        <Geogebra
16          width="650" height="600" showToolBar="false" showMenuBar="false"
17          showAlgebraInput="false" allowStyleBar appletOnLoad={ggbOnInit}
18        />
19        <Stack direction="row" alignItems="center">
20          <Typography>Setup</Typography>
21          <Switch checked={mode} onChange={handleChange} />
22          <Typography>Play</Typography>
23          <Button variant="contained" onClick={init} sx={{margin:2}}>Reset</Button>
24          <Button variant="contained" onClick={judgement} sx={{margin:2}}>Judge</Button>
25          <Button variant="contained" onClick={solution} sx={{margin:2}}>Solve</Button>
26        </Stack>
27      </div>
28    );
```

1〜3 行目で必要な React コンポーネントをインポートして、15〜18 行目で GeoGebra コンポーネントを、20〜22 行目でトグルスイッチを、23〜25 行目で Reset/Judge/Solve の 3 個のボタンを設置している。

6 行目の `React.useState` 関数は React コンポーネントの状態（ステート）を管理するものであり、現在のステートとそれを更新するために関数の組を返す。`Rect.useState` 関数の引数はステートの初期値である。この 6 行目により、ステート変数 `mode` とステート更新関数 `setMode` が宣言され、`mode` に初期値 `true` が設定される。10〜11 行目は、セルのクリックイベントのリスナー内で最新のステート値を参照するために必要であり、`stateMode` の `current` プロパティにトグルスイッチのモードを保持させている。

7〜9 行目の `handleChange` 関数は、トグルスイッチ使用時に実行されるイベントハンドラで、`setMode` 関数を用いて `mode` のステート値を変更する。

3.3 ライツアウトの盤面生成

GeoGebra コンポーネントがロードされた際に実行される `ggbOnInit` 関数とライツアウトの盤面を生成する `createCell` 関数を `App.js` に追加する。

コード 2　`App.js`（追加）

```
30    // GeoGebra初期化関数
31    function ggbOnInit() {
32      const ggbApplet = window.ggbApplet;
33      ggbApplet.evalCommand('SetPerspective("2")'); // AlgebraViewを非表示("2"は
          GraphicsViewを指す)
34      ggbApplet.setAxesVisible(false,false); // 座標軸を非表示
35      ggbApplet.setGridVisible(false); // グリッドを非表示
```

```
36      ggbApplet.setCoordSystem(-2,12,-12,1.5); // 原点位置を左上に設定
37      ggbApplet.evalCommand('SetAxesRatio(1,1)'); // 座標軸の縦横比を1:1に
38      ggbApplet.enableShiftDragZoom(false); // グラフィックスビューのドラッグ&ズームを禁止
39      createCell(); // セルを表示
40      ggbApplet.registerClickListener(ListenerC); // クリックイベントのリスナーを指定
41    }
42
43    // ライツアウトの盤面を生成
44    function createCell() {
45      const ggbApplet = window.ggbApplet;
46      // 5x5格子点P_iの作成
47      ggbApplet.evalCommand('Execute(Sequence("P"+(i)+"=("+2*Mod(i,5)+","+(-2*Div(i,5))
          +")",i,0,24))');
48      // 格子点を左上に持つセルq_iをPolygonで作成
49      ggbApplet.evalCommand('Execute(Sequence("q"+(i)+"=Polygon(P"+(i)+",P"+(i)+"+(0,-2)
          ,P"+(i)+"+(2,-2),P"+(i)+"+(2,0))",i,0,24))');
50      for (var i = 0; i < ggbApplet.getObjectNumber(); i++) {
51        var obj = ggbApplet.getObjectName(i);
52        ggbApplet.setLabelVisible(obj,false); // オブジェクトのラベルは非表示
53        if (ggbApplet.getObjectType(obj) === "point") {
54          ggbApplet.setVisible(obj,false); // 点は非表示
55        } else if (ggbApplet.getObjectType(obj) !== "quadrilateral") {
56          ggbApplet.setFixed(obj,true,false); // セルq_i以外は選択できないように
57        } else {
58          ggbApplet.setFixed(obj,true,true); // ドラッグでセルが動かないように固定化
59          ggbApplet.evalCommand("SetDynamicColor("+(obj)+",0,0,0,0.2)"); // セルのデフォル
              トカラーを白に
60        }
61      }
62    }
```

ggbOnInit 関数は、GeoGebra のグラフィックスビューの座標を設定・固定し、クリックイベントのリスナーを指定している。createCell 関数は、25 個のセルを GeoGebra コマンドの Polygon で作成し、クリック時の誤動作を避けるための設定を行っている。

3.4 イベントリスナー

セルをクリックした際、そのセルを含む上下左右のセルの点灯状態を反転させるイベント処理を次の ListenerC 関数と reverseColor 関数で実現する。

コード 3　App.js（追加）

```
64    // クリックされた場所にあるセルの色を変更
65    function ListenerC(obj) { // objにはクリックされたオブジェクト名が渡される
66      const ggbApplet = window.ggbApplet;
```

```
67      if (obj.startsWith('q')) { // セルq_iがクリックされたら
68        reverseColor(obj);
69        var num = parseInt(obj.substring(1));
70        if (stateMode.current === true) { // Playモードなら十字に色反転
71          if ((num-1)%5 !== 4) reverseColor("q"+(num-1));
72          if ((num+1)%5 !== 0) reverseColor("q"+(num+1));
73          if ((num-5) >= 0) reverseColor("q"+(num-5));
74          if ((num+5) < 25) reverseColor("q"+(num+5));
75        }
76        // クリックしたセルに解答の白丸がある場合は削除
77        if (ggbApplet.isDefined("AP"+(num))) {
78          ggbApplet.deleteObject("AP"+(num));
79        }
80      }
81    }
82
83    // セルの色を反転
84    function reverseColor(obj) {
85      const ggbApplet = window.ggbApplet;
86      if (ggbApplet.getColor(obj) === "#000000") {
87        ggbApplet.evalCommand('SetDynamicColor('+(obj)+',0,0,1,0.5)');
88      } else {
89        ggbApplet.evalCommand('SetDynamicColor('+(obj)+',0,0,0,0.2)');
90      }
91    }
```

70 行目で、stateMode.current を利用して Setup モードか Play モードかの判定を行い、反転させるセルを決めている。

4 計算エンジンの作成

4.1 ライツアウトの求解と可解性判定

ここでは、点灯したセルを 1、消灯したセルを 0 で表すこととする。

1	1	0	0	0
1	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

図 3　$(1,1)$ のセルをクリックした場合

1	1	1	0	0
0	1	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

図 4　$(1,2)$ のセルをクリックした場合

全てのセルが消灯している盤面で最上段左端のセルをクリックした場合、左上から順に点灯状態

を並べると

$$(1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$

と表せ、その右隣のセルをクリックしたときの状態は、

$$(1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$

と表せる。左上から数えて i 番目のセルをクリックしたときの状態を第 i 列とする 25×25 行列を用いて、次の連立一次方程式を考える。

$$\begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1
\end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \\ x_{17} \\ x_{18} \\ x_{19} \\ x_{20} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{25} \end{pmatrix}
=
\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \\ b_{11} \\ b_{12} \\ b_{13} \\ b_{14} \\ b_{15} \\ b_{16} \\ b_{17} \\ b_{18} \\ b_{19} \\ b_{20} \\ b_{21} \\ b_{22} \\ b_{23} \\ b_{24} \\ b_{25} \end{pmatrix}$$

変数 x_i にはセルのクリック情報が入る。i 番目のセルをクリックする場合は $x_i = 1$、そうでない場合は $x_i = 0$ となる。b_i は最終的なセルの点灯状態を表す。i 番目のセルが点灯する場合は $b_i = 1$、そうでない場合は $b_i = 0$ となる。ライツアウトの解を求めるには、b_i に盤面の点灯状態を代入してこの連立一次方程式を二元体 $GF(2)$ で解けば良い。

この連立一次方程式にガウスの消去法の前進消去を行うと、次の拡大係数行列が得られる。

$$\left(\begin{array}{cccccccccccccccccccccccc|c}
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_1 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_2 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_3 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_4 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_5 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_6 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_7 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_{10} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_{11} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_{12} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & b'_{13} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & b'_{14} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & b'_{15} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & b'_{16} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & b'_{17} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & b'_{18} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & b'_{19} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & b'_{20} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & b'_{21} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & b'_{22} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & b'_{23} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_{24} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b'_{25}
\end{array}\right)$$

係数行列の第 24〜25 行がすべて 0 となることから、$b'_{25} = b'_{24} = 0$ であれば可解、そうでなければ非可解であることがわかる。可解な場合は x_{25} と x_{24} の値を定めれば、後退代入により x_{23}〜x_1 が順に求められる。(x_{25}, x_{24}) が取り得る値は $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$ であるから、可解な場合の解の個数は丁度 4 個である[*2]。

4.2 C 言語プログラム

前節で解説したアルゴリズムを C 言語で実装する。可解かどうかを判定する Judge ボタンと解を求める Solve ボタンから呼び出す関数は以下の 2 つである。

`int solvable(int *num, int n)` ライツアウトの可解性を判定する関数。点灯しているセル番号（0〜24）を int 型の配列 `num` で、点灯しているセルの個数を `n` で渡すと、可解なら 1 を、非可解なら 0 を返す。

`int *solve(int *num, int n)` ライツアウトの解を求める関数。点灯しているセル番号（0〜24）を int 型の配列 `num` で、点灯しているセルの個数を `n` で渡すと、可解であればライツアウトの解を `[0,1,0,0,0,1,1,1,0,0,...]` という長さ 25 の int 型の配列で返す。クリックするセルは `1` で、クリックしないセルは `0` で表される。この関数は 4 個の解の中からランダムに一つを出力する。また、非可解であれば `[-1,0,0,0,0,...]` を返す。

[*2] これは盤面が 5 × 5 のタイプの結果であり、タイプが異なれば結果も異なる。ちなみに、3 × 3 や 6 × 6 のタイプは係数行列の階数が下がらないため、常に可解であり、解はただ一つである。ライツアウトの解法アルゴリズムについては [6, 7] が詳しい。

これらの関数を含む計算エンジンの C ソースファイル `lightsout.c` は [8] から入手できる。この `lightsout.c` を Emscripten で WebAssembly に変換する。

4.3 WebAssembly への変換

WebAssembly（Wasm）[9] は、Web ブラウザ上で実行可能なバイナリーフォーマットの一種である。Web アプリのパフォーマンス向上を目指して 2015 年に開発が開始され、2017 年に主要 Web ブラウザが WebAssembly に対応した。2019 年 12 月には W3C が仕様を勧告したことにより、Web 技術の標準になっている。

直接 WebAssembly のコードを記述することも可能であるが、C/C++/Rust などのプログラミング言語からコンパイルしてコード生成するように設計されている。Emscripten [10] は、C/C++ のコードを Wasm バイナリに変換するコンパイラである。[5] の第 4 章に Emscripten の環境構築について詳しい解説がある。

前節で準備した C ソースファイル `lightsout.c` を WebAssembly に変換するために、以下のように `emcc` を実行する。

```
$ emcc --no-entry lightsout.c -o lightsout.mjs \
  -s ENVIRONMENT='web' \
  -s SINGLE_FILE=1 \
  -s EXPORT_NAME='createModule' \
  -s EXPORTED_FUNCTIONS='["_solvable","_solve","_free"]' \
  -s EXPORTED_RUNTIME_METHODS='["ccall"]' \
  -O3
```

各オプションの意味を説明する。

`--no-entry` `main` 関数を使用しないようにする。
`-s ENVIRONMENT='web'` Web ブラウザのみで使用する。
`-s SINGLE_FILE=1` `.mjs` ファイルと `.wasm` ファイルを一つにまとめる。
`-s EXPORT_NAME` モジュール生成関数の名前を設定する。
`-s EXPORTED_FUNCTIONS` JavaScript 側から利用する C 関数を設定する。
`-s EXPORTED_RUNTIME_METHODS` C 関数の実行に利用する関数を設定する。

以上の操作で、Wasm モジュールファイル `lightsout.mjs` が生成される。

4.4 Wasm モジュールファイルの利用

Wasm モジュールファイルを React で利用するためには準備が必要である。まず、Wasm モジュールファイル `lightsout.mjs` を `my-app/src` ディレクトリにコピーする。そして、React テンプレートのルートディレクトリ `my-app` にあるファイル `package.json` を以下のように修正する。

コード 4　package.json（修正）

```
26   "eslintConfig": {
27     "extends": [
28       "react-app",
29       "react-app/jest"
30     ],
31     "ignorePatterns": [
32       "src/lightsout.mjs"
33     ]
34   },
```

ESLint [11] は JSX のコードをトランスパイルする際に利用される構文チェッカーであるが、lightsout.mjs の Wasm コードはチェックできない。上の修正により ESLint が lightsout.mjs ファイルを無視するようになり、トランスパイルが通るようになる。

次に、createModule 関数を利用するために、App.js の頭部に import 文を追加する。

コード 5　App.js（追加）

```
1  import Geogebra from 'react-geogebra';
2  import React from 'react';
3  import { Stack, Typography, Switch, Button, Snackbar, Alert } from '@mui/material';
4  import createModule from './lightsout.mjs';
5
6  function App() {
```

4.5　Judge ボタンのイベント処理

Judge ボタンをクリックすると、C 言語の solvable 関数を用いて可解性を判定し、可解であれば “Solvable!”、非可解であれば “Unsolvable!” というメッセージを画面下部に一時的に表示[*3]するようにする。

ここではスナックバーを Material UI の Snackbar コンポーネントと Alert コンポーネントを組み合わせて実現する。そのために、App.js に次を追加する。

コード 6　App.js（追加）

```
13   const [open, setOpen] = React.useState(false);
14   const handleClose = (event, reason) => {
15     if (reason === 'clickaway') return;
16     setOpen(false);
17   };
18   const [message, setMessage] = React.useState('');
```

[*3] このような何らかのアクションに関連する簡易的なメッセージ表示を**スナックバー**と呼ぶ。

13 行目でスナックバーの表示を管理するステート変数 open とステート更新関数 setOpen を宣言し、18 行目でメッセージ内容を管理するステート変数 message とステート更新関数 setMessage を宣言している。14〜17 行目の handleClose 関数は、スナックバーを閉じるイベントハンドラで、15 行目でスナックバーの領域外のクリックによりスナックバーが閉じないようにしている。

そして、App.js に Snackbar と Alert コンポーネントを追加する。

コード 7　App.js（追加）

```
34      <Snackbar open={open} autoHideDuration={5000} onClose={handleClose}
35        anchorOrigin={{vertical: 'bottom', horizontal: 'center'}}
36      >
37        <Alert icon={false} onClose={handleClose} variant="filled" severity="success"
38          sx={{width:'200px','& .MuiAlert-message':{textAlign:'center',width:'inherit'
              }}}
39        >
40          <strong>{message}</strong>
41        </Alert>
42      </Snackbar>
```

34 行目の autoHideDuration={5000} により、5 秒後に自動的にメッセージが非表示になる。また、37 行目の onClose={handleClose} により、スナックバーの右側に×ボタンが表示され、これをクリックすることで 5 秒を待たずにスナックバーを閉じることができるようにしている。

さらに、App.js に judgement 関数を追加する。

コード 8　App.js（追加）

```
109   // 可解性の判定
110   function judgement() {
111     const ggbApplet = window.ggbApplet;
112     var numlist = []; // 点灯セルの番号を保存する配列
113     // 解答の白丸を削除
114     for (var i = 0; i < 25; i++) {
115       if (ggbApplet.isDefined("AP"+(i))) {
116         ggbApplet.deleteObject("AP"+(i));
117       }
118     }
119     for (i = 0; i < ggbApplet.getObjectNumber(); i++) {
120       var obj = ggbApplet.getObjectName(i);
121       if (ggbApplet.getObjectType(obj) === "quadrilateral" && ggbApplet.getColor(obj)
            !== "#000000") {
122         numlist.push(obj.substring(1));
123       }
124     }
```

```
125     var len = numlist.length;
126     var numarray = new Uint8Array(new Uint32Array(numlist).buffer); // C関数に渡す配列を
          準備
127     createModule().then(mod => { // solvable関数の呼び出し
128       var result = mod.ccall('solvable','number',['array','number'],[numarray,len]);
129       if (result === 1) setMessage('Solvable!');
130       else setMessage('Unsolvable!');
131       setOpen(true);
132     });
133   }
```

119〜124 行目で点灯しているセルの番号を配列 numlist に保存し、126 行目で C 関数に渡すための前処理を行っている。128 行目で Emscripten の ccall 関数を用いて、C 言語の solvable 関数を実行し、131 行目でステート変数 open の値を true に変更することで実行結果をスナックバーで表示している。

4.6 Solve ボタンのイベント処理

Solve ボタンをクリックすると、C 言語の solve 関数を用いてライツアウトの解を求め、クリックするべきセル上に白丸を付加するようにする。

そのために、App.js に solution 関数を追加する。

コード 9　App.js（追加）

```
135   // ライツアウトの求解
136   function solution() {
137     const ggbApplet = window.ggbApplet;
138     var numlist = []; // 点灯セルの番号を保存する配列
139     // 解答の白丸を削除
140     for (var i = 0; i < 25; i++) {
141       if (ggbApplet.isDefined("AP"+(i))) {
142         ggbApplet.deleteObject("AP"+(i));
143       }
144     }
145     for (i = 0; i < ggbApplet.getObjectNumber(); i++) {
146       var obj = ggbApplet.getObjectName(i);
147       if (ggbApplet.getObjectType(obj) === "quadrilateral" && ggbApplet.getColor(obj)
            !== "#000000") {
148         numlist.push(parseInt(obj.substring(1))); // q_iの添え字番号iを整数に変換して
              numlistに保存
149       }
150     }
151     var len = numlist.length;
152     var numarray = new Uint8Array(new Uint32Array(numlist).buffer); // C関数に渡す配列を
```

```
        準備
153     createModule().then(mod => { // solve関数の呼び出し
154       var ptr = mod.ccall('solve', 'number', ['array','number'], [numarray, len]);
155       var rslt = new Int32Array(mod.HEAP32.buffer, ptr, 25); // JavaScriptの配列に変換
156       if (rslt[0] === -1) { // 非可解な場合
157         setMessage('Unsolvable!');
158         setOpen(true);
159       } else { // 可解な場合
160         for (i = 0; i < 25; i++) {
161           if (rslt[i] === 1){ // 解答の白丸をセルに付加
162             var x = ggbApplet.getXcoord("P"+(i))+1;
163             var y = ggbApplet.getYcoord("P"+(i))-1;
164             ggbApplet.evalCommand("AP"+(i)+"=("+(x)+","+(y)+")");
165             ggbApplet.setLabelVisible("AP"+(i),false);
166             ggbApplet.setFixed("AP"+(i),true,false);
167             ggbApplet.evalCommand("SetDynamicColor(AP"+(i)+",1,1,1,0.2)");
168           }
169         }
170       }
171       mod._free(ptr);
172     });
173   }
```

155 行目で C 言語の `solve` 関数が返した配列（ポインタ）を JavaScript の配列に変換している。160〜169 行目でライツアウトの解となるセル上に白丸を付加している。

4.7 Reset ボタンのイベント処理

Reset ボタンをクリックすると、解答の白丸を削除し、すべてのセルを消灯状態にして、モードを Play に戻し、初期状態にする。

そのために、`App.js` に `init` 関数を追加する。

コード 10　`App.js`（追加）

```
175   // ゲームの初期化
176   function init() {
177     const ggbApplet = window.ggbApplet;
178     // 解答の白丸を削除
179     for (var i = 0; i < 25; i++) {
180       if (ggbApplet.isDefined("AP"+(i))) {
181         ggbApplet.deleteObject("AP"+(i));
182       }
183     }
184     // 色つきセルを白に
185     for (i = 0; i < ggbApplet.getObjectNumber(); i++) {
```

```
186       var obj = ggbApplet.getObjectName(i);
187       if (ggbApplet.getObjectType(obj) === "quadrilateral" && ggbApplet.getColor(obj)
             !== "#000000") {
188         ggbApplet.evalCommand("SetDynamicColor("+(obj)+",0,0,0,0.2)");
189       }
190     }
191     setMode(true);
192   }
193 }
194 export default App;
```

5　アプリのビルドと実行

React テンプレートのルートディレクトリ（my-app）で次を実行する。

```
$ npm run build
```

これにより、App.js の JSX コードがトランスパイルされ、公開用ファイルが build ディレクトリに生成される。アプリを実行するには、次のように Web サーバを起動して Web ブラウザから http://localhost:8000/build/ にアクセスすれば良い。

```
$ python3 -m http.server &
```

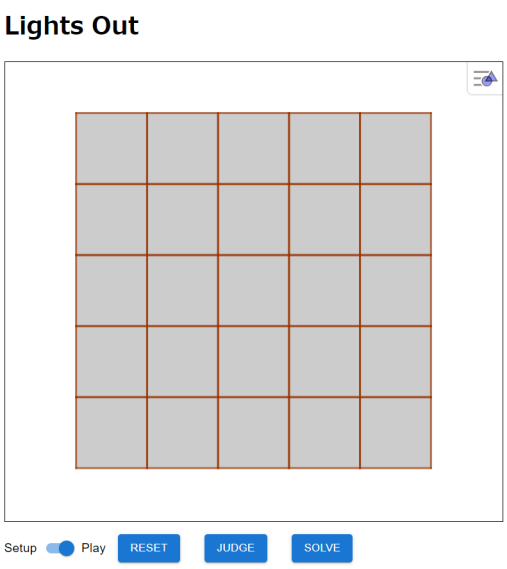

図 5　初期状態

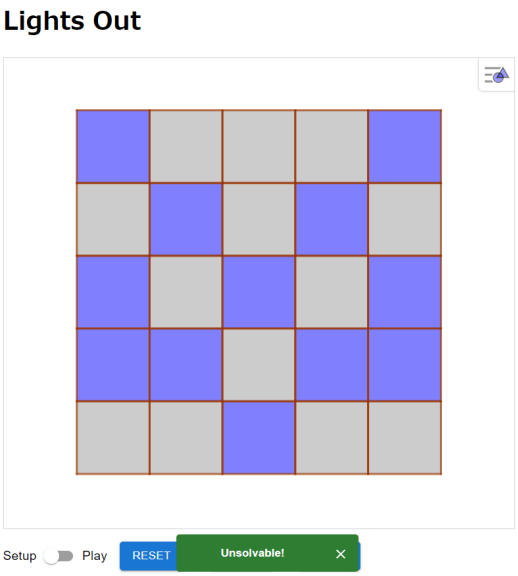

図 6　非可解なパターン

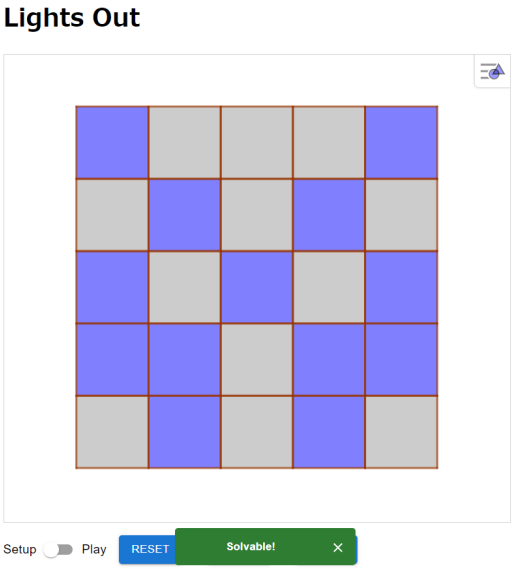

図 7　可解なパターン

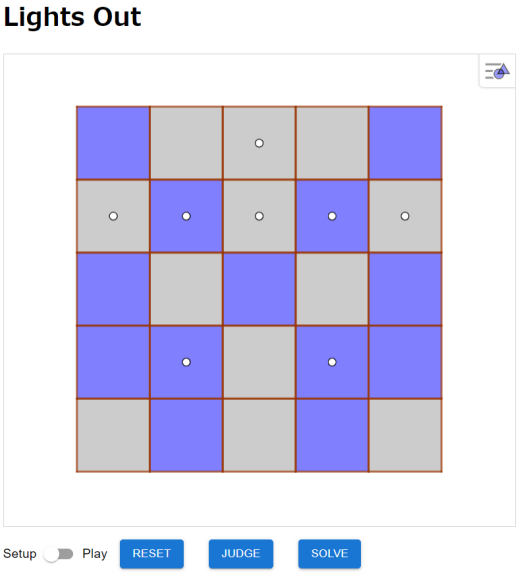

図 8　Solve ボタンで解を表示

以下に本アプリの遊び方を解説する。

1. モードを **Setup** に変更してパターンを作成する。Setup モードではクリックしたセルのみが反転する。
2. パターンが出来たら Judge ボタンをクリックする。可解であれば “Solvable!”が、そうでなければ “Unsolvable!”のスナックバーが下部中央に表示される。スナックバーは 5 秒で自動的に消えるが、×ボタンのクリックで消すことも可能である。
3. 可解なパターンが出来たらモードを **Play** に変更してパズルを解く。Play モードではクリックしたセルを含む上下左右のセルが反転する。
4. すべての点灯を消すことができれば完成。
5. 解答が知りたくなったら Solve ボタンをクリックする。白丸が表示されるので、そのセルをクリックすると全ての点灯を消すことができる。Solve ボタンを再度クリックすることにより別の解を表示できる。解は 4 種類あり、その表示順はランダムである。
6. 初期状態に戻したいときは Reset ボタンをクリックする。すべてのセルが消灯状態になり、モードも Play に戻る。

完成したアプリは以下の URL または QR コードから利用可能である。

`https://mitsushi-fujimoto.github.io/geo-lights/`

6 おわりに

本稿では、UI の作成に GeoGebra と React を、計算エンジンの作成に C 言語と WebAssembly を用いて Web アプリを実装する方法を解説した。実装例として数学パズルのライツアウトを選んだのは、研究室の卒業研究プロジェクト [12] が背景にある。そこでは GeoGebra スクリプトのみでライツアウトの作成に取り組んだが、全てのパターンが可解な 3×3 と 6×6 のタイプしか実現できなかった。この原因は GeoGebra スクリプトによる連立方程式の扱い（特に、解が複数ある場合の処理）に不慣れだったことにあった。この反省を元に、GeoGebra スクリプトでなく（数値解析の演習で慣れ親しんでいる）C 言語を用いる方が実装や処理速度において効率的でないかと考えた。

結果的に本稿で解説した方法により、非常に短いソースコードで 5×5 のライツアウトアプリを実現することができた。UI の実装に用いた `App.js` は 200 行、計算エンジンの C ソースファイル `lightsout.c` は 100 行である。本手法を用いる上で重要な箇所は次の 2 点である。

- **React から WebAssembly を利用するための準備**
  `emcc` のオプションと ESLint の設定を適切に行う[*4]。
- **JavaScript と C 関数との配列の受け渡し**
  受け渡しの前後に変換処理を適切に行う。

また、UI と計算エンジンを切り離して作成したことにより、各機能の実装・テストが容易で、問題が発生した際の原因も特定し易いという利点に気付くことができた。本手法を応用すれば、高機能な数式処理システムと GeoGebra を連携させた数学ソフトウェアの開発が可能と考える。

今後の課題はビルドシステムを変更することである。[1] 及び本稿では、React テンプレートの作成に `create-react-app` を利用した。`create-react-app` はビルドシステムに Webpack を利用したプロジェクト生成ツールであり、長らく標準ツールとして利用されてきたが、2022 年 4 月 12 日にリリースされたバージョン 5.0.1 を最後に更新が途絶えている。また、2023 年 3 月にリニューアルされた React の公式ドキュメント [14] にも紹介されなくなったことから、標準ツールの役割を終えたと考えられる。`create-react-app` の後継には、プロジェクトのディレクトリサイズが小さく、ビルドが高速な Vite [15] が挙げられる。Vite で生成した React テンプレートで Wasm モジュールファイルを利用する方法の確立、Webpack と Vite におけるビルド速度や実行速度の比較などが必要と思われる。

謝辞　本研究は JSPS 科研費 JP21K12157 の助成を受けたものである。

[*4] これらの設定については [13] を参考にした。

参考文献

[1] 藤本光史, How to embed GeoGebra into Web, 統計数理研究所共同研究リポート 452, 動的幾何学ソフトウェア GeoGebra の整備と普及 (7), (2022) 11–29.

[2] react-geogebra, https://www.npmjs.com/package/react-geogebra

[3] Material UI, https://mui.com/

[4] Bootstrap, https://getbootstrap.com/

[5] 高山信毅, 野呂正行, 小原功任, 藤本光史, 数学ソフトウェアの作り方, 共立出版, 2022.

[6] 安田健彦, ゲームで大学数学入門, 共立出版, 2018.

[7] 大槻兼資, パズルで鍛えるアルゴリズム力, ソフトウェアデザイン（2021 年 10 月号）, 技術評論社, 2021.

[8] 藤本光史, geo-lights, https://github.com/mitsushi-fujimoto/geo-lights

[9] WebAssembly, https://webassembly.org/

[10] Emscripten, https://emscripten.org/

[11] ESLint, https://eslint.org/

[12] 高時浩太, 線形代数を用いた lights out 解法アプリの開発, 福岡教育大学卒業論文, 2023.

[13] Bobbie Chen, React C/C++ WASM demo, https://github.com/bobbiec/react-wasm-demo/blob/main/README-inlined-version.md

[14] React, https://react.dev/

[15] Vite, https://ja.vitejs.dev/