How to make an app using JavaScript on GeoGebra

藤本 光史（福岡教育大学)*

概要

GeoGebra で少し凝ったアプリを作成しようとすると、JavaScript の使用は避けられない。本稿では GeoGebra 初心者がアプリを作成する際の手助けになるように、一筆書きアプリの作成方法を通して JavaScript の利用方法の解説を試みる。

1 はじめに

2017 年の本研究会において、筆者は GeoGebra 上での 15 パズルアプリの作成方法を解説した [1]。このアプリは GeoGebra スクリプトのチュートリアルとして作成したものであり、GeoGebra スクリプトのみを用いている。そして、その作成過程で GeoGebra スクリプトが持つ欠点に気づくことができた。次はそのうちの主なものである。

- 文の途中で改行できない。
- If 文は多分岐に対応しているが複数コマンドは実行できない。
- ループ文がないため、Sequence と Execute を用いて代用しなくてはならない。
- クリックしたオブジェクトの名前を取得する関数がない。
- 存在する点や線分などの全オブジェクト情報を取得する関数がない。

これらのことから GeoGebra で少し複雑な教材を作るには、GeoGebra スクリプトは不向きであると言わざるを得ない。GeoGebra は GeoGebra スクリプト以外に JavaScript をスクリプト言語として利用可能である。しかし、GeoGebra で JavaScript を利用するための文書は、筆者の知る限り [2] と [3] しかない。そこで、本稿では JavaScript を利用した GeoGebra 教材作成のチュートリアルとして、一筆書きアプリを題材に点や線分などの全オブジェクトの把握が必要なアプリが JavaScript を使用することで作成できることを解説する。

注意事項：本稿の内容は GeoGebra 5 Classic（バージョン 5.0.564.0）で動作確認している。原稿執筆時点での GeoGebra 6 は JavaScript の挙動が GeoGebra 5 と異なっており、本稿で作成したアプリの動作は保証できない。

* fujimoto@fukuoka-edu.ac.jp

2　一筆書きアプリ完成までのステップ

一筆書きアプリを作成するまでのステップは以下の 7 ステップである。

1. 全オブジェクト情報の取得
2. 点に接続する線分の把握
3. 奇点と偶点の個数
4. 一筆書き可能かどうかの判定
5. 一筆書き順路の生成
6. 解答アニメーションの作成
7. リセットボタンの作成

2.1　【ステップ 1】全オブジェクト情報の取得

描いた図形が一筆書き可能かどうかの判定には点とそこに繋がる線分の情報が必要である。ここでは、描画エリア（グラフィックスビュー）にある全ての点と線分が格納された配列を作る。この実現には GeoGebra Apps API として提供されている次のイベントリスナーが有用である。これらは点や線分などのオブジェクトが追加・削除・名前変更された時に指定された関数を呼び出す。

- `void registerAddListener(String JSFuncName)`
- `void registerRemoveListener(String JSFuncName)`
- `void registerRenameListener(String JSFuncName)`

GeoGebra で JavaScript のコードを入力するには何らかのオブジェクトが必要である。そこで、ツールバーの右から 2 つ目のツールボタンの▽から「ボタン」を選択後、描画エリア（グラフィックスビュー）をクリックして判定ボタンを作成する。

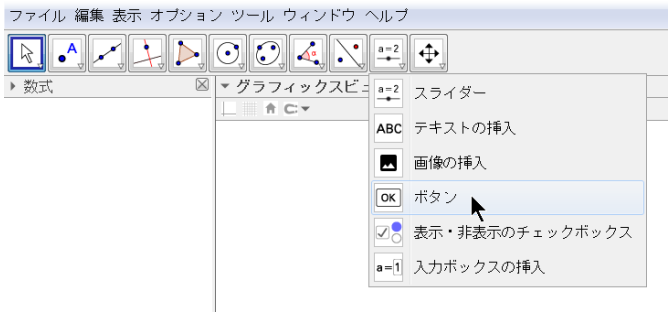

図 1　判定ボタンの作成

そして、判定ボタンを右クリック→「オブジェクトのプロパティ」→「スクリプト記述」→「グローバル Java スクリプト」欄に以下のコードを入力する。枠外の数字は行番号であり、入力して

はいけない。

```
function ggbOnInit() {
  ggbApplet.registerAddListener("ListenerA");
  ggbApplet.registerRemoveListener("ListenerRM");
  ggbApplet.registerRenameListener("ListenerRN");
  objectList();
}
function ListenerA(P1) {
  objectList();
}
function ListenerRM(P1) {
  objectList();
}
function ListenerRN(P1,P2) {
  objectList();
}
```

このように GeoGebra Apps API は `ggbApplet.` を付けて使用する。さらに、いずれのリスナーからも呼ばれる `objectList()` 関数を追加する。先頭に行番号がない行は、直前の行から続いていることを表している。

```
function objectList() {
  ggbApplet.unregisterAddListener("ListenerA");
  ggbApplet.unregisterRemoveListener("ListenerRM");
  Pname = [];
  Sname = [];
  for (i = 0; i < ggbApplet.getObjectNumber(); i++) {
    obj = ggbApplet.getObjectName(i);
    if (ggbApplet.getObjectType(obj) == "point") Pname.push
      (obj);
    else if (ggbApplet.getObjectType(obj) == "segment")
      Sname.push(obj);
  }
  ggbApplet.registerAddListener("ListenerA");
  ggbApplet.registerRemoveListener("ListenerRM");
}
```

この objectList() 関数は、存在する全てのオブジェクトを調べ、そのオブジェクトが点ならば配列 Pname に、線分ならば配列 Sname にそのオブジェクト名を格納していく。「グローバル Java スクリプト」欄で定義された配列 Pname と Sname はグローバル変数として利用できる。

ここまで作成したものが問題なく動作しているか（つまり、全ての点と線分のオブジェクト名が取得できているか）チェックするために、次のコードを上の objectList() 関数の**下から 3 行目と 4 行目の間に挿入**する。

```
1  if (!ggbApplet.exists('Points')) ggbApplet.evalCommand('
     Points={}');
2  ggbApplet.setVisible('Points', false)
3  if (!ggbApplet.exists('Segments')) ggbApplet.evalCommand
     ('Segments={}');
4  ggbApplet.setVisible('Segments', false)
5  PnameS = "" + Pname.sort();
6  SnameS = "" + Sname.sort();
7  PnameStr = '{"' + PnameS.replace(/,/g, '\",\"') + '"}';
8  SnameStr = '{"' + SnameS.replace(/,/g, '\",\"') + '"}';
9  ggbApplet.evalCommand('SetValue[Points,' + PnameStr +
     ']');
10 ggbApplet.evalCommand('SetValue[Segments,' + SnameStr +
     ']');
```

注意事項：GeoGebra スクリプトの使用時と異なり、「グローバル Java スクリプト」欄を修正した場合、ggb ファイルを保存し、再読み込みしないと**修正は反映されない**。

上記のコードを入力した ggb ファイルを再読み込みしてから、点や線分を入力してみよう。左の「数式」ペインの Points と Segments リストに点や線分のオブジェクト名が格納されていく様子がわかるはずである。

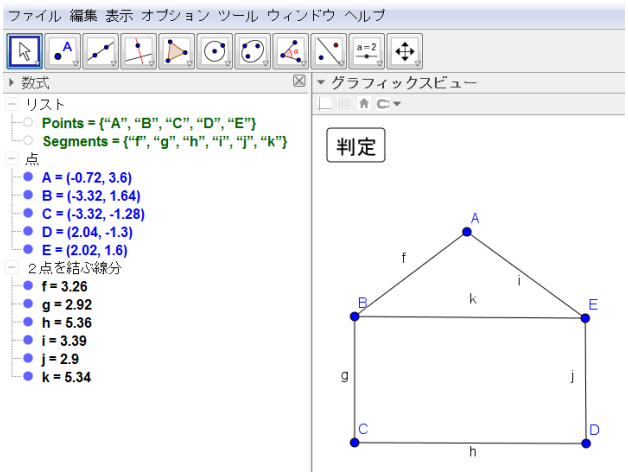

図 2　点と線分の取得

2.2 【ステップ 2】点に接続する線分の把握

点 P に接続する線分の数を P の次数という。P の次数が奇数（偶数）のとき、P は奇点（偶点）という。図形が一筆書き可能かどうかは奇点の個数で決まる。そこで、各点の偶奇性を調べるために点に接続する線分を把握したい。GeoGebra の線分オブジェクトは 2 点で定義されており、端点情報を有している[*1]。よって、ある点を共有している線分を見つけるのは困難ではない。しかし、ここでは点 A に対して、中心 A、半径 0.5 の小さな円 $tmpC$ を描き、$tmpC$ と線分が交点を持つかどうかで A と接続する線分を見つける[*2]。この手法を採用したのは後の演習課題で図形の線を「線分」だけでなく「ペンストローク」にも対応できるようにするためである。

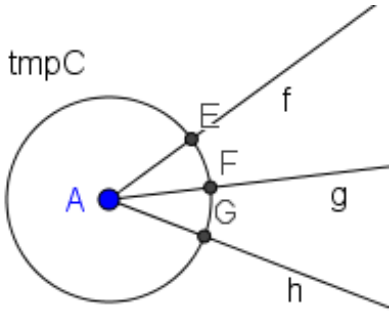

図 3　点と接続する線分の把握

そのために、判定ボタンを右クリック→「オブジェクトのプロパティ」→「スクリプト記述」→「クリックして」欄に以下のコードを入力する。

```
connect = [];
for (i = 0; i < Pname.length; i++) {
  ggbApplet.evalCommand('tmpC=Circle(' + Pname[i] +
    ',0.5)');
  cnt = 0;
  for (j = 0; j < Sname.length; j++) {
    ggbApplet.evalCommand('tmpP=Intersect(tmpC, ' + Sname[j
      ] + ')');
    if (ggbApplet.isDefined('tmpP')) cnt++;
    ggbApplet.deleteObject('tmpP');
  }
  connect.push(cnt);
  ggbApplet.deleteObject('tmpC');
```

[*1] 例えば、`f=Segment(A,B)` で定義された線分 `f` の始点は `Point(f,0)` で、終点は `Point(f,1)` で取り出せる。
[*2] この手法は [4] で使用されたものである。

```
12 }
```

ここでは、tmpC と線分 f に対して、tmpP = Intersect(tmpC, f) を evalCommand で実行することで交点 tmpP を求めている。isDefined('tmpP') が true の時、交点が存在する、つまり、線分 f が点に接続していることになり、カウンター cnt が増加される。各点の次数は配列 connect に保存される。

注意事項：「グローバル Java スクリプト」欄と異なり、「クリックして」欄では、下部のドロップダウンリストで入力するコードを「GeoGebra Script」から「JavaScript」に変更する必要がある。

図 4　入力コードを JavaScript に指定

2.3　【ステップ 3】奇点と偶点の個数

ステップ 2 により、各点の次数が配列 connect に保存される。この情報を元に奇点と偶点の個数をカウントし、奇点の色を青から赤に、サイズをデフォルトの 5 から 7 に変更（少し大きく）する。また、次数を各点のキャプションに設定し、それを点のオブジェクト名の代わりに表示する。

そのために、判定ボタンを右クリック→「オブジェクトのプロパティ」→「スクリプト記述」→「クリックして」欄に以下のコードを**追加**する。

```
1 for (even = 0, odd = 0, i = 0; i < Pname.length; i++) {
2   ggbApplet.setCaption(Pname[i], connect[i]);
3   ggbApplet.setLabelStyle(Pname[i], 3);
4   if (connect[i] % 2 != 0) {
5     ggbApplet.setColor(Pname[i], 255, 0, 0);
6     ggbApplet.setPointSize(Pname[i], 7);
7     odd++;
```

```
  } else {
    ggbApplet.setColor(Pname[i], 0, 0, 255);
    ggbApplet.setPointSize(Pname[i], 5);
    even++;
  }
}
```

2.4 【ステップ 4】一筆書き可能かどうかの判定

一筆書き可能かどうかの判定方法は、Euler と Hierholzer による次の定理で与えられる。

定理（Euler − Hierholzer） 連結な図形が一筆書き可能であるための必要十分条件は、奇点の個数が 0 個、または 2 個であることである。

ステップ 3 において、奇点の個数が変数 odd に、偶点の個数が変数 even に保存される。ここでは、それらの値を表示し、さらに odd の値が 0 または 2 であるならば「一筆書き可能」、そうでなければ「一筆書き不可能」と表示するようにする。

そのために、判定ボタンを右クリック→「オブジェクトのプロパティ」→「スクリプト記述」→「クリックして」欄に以下のコードを**追加**する。

```
ggbApplet.evalCommand('odd=FormulaText(\"\\Large奇点： ' +
  odd + '\")');
ggbApplet.setCoords('odd', 3, 5);
ggbApplet.evalCommand('even=FormulaText(\"\\Large 偶点： ' +
   even + '\")');
ggbApplet.setCoords('even', 6.5, 5);
if (odd == 0 || odd == 2) {
  ggbApplet.evalCommand('result=FormulaText(\"\\Large \\
    Rightarrow 一筆書き可能\")');
  ggbApplet.setCoords('result', 3, 4);
} else {
  ggbApplet.evalCommand('result=FormulaText(\"\\Large \\
    Rightarrow 一筆書き不可能\")');
  ggbApplet.setCoords('result', 3, 4);
}
```

次の図は 判定 ボタンの実行例である。

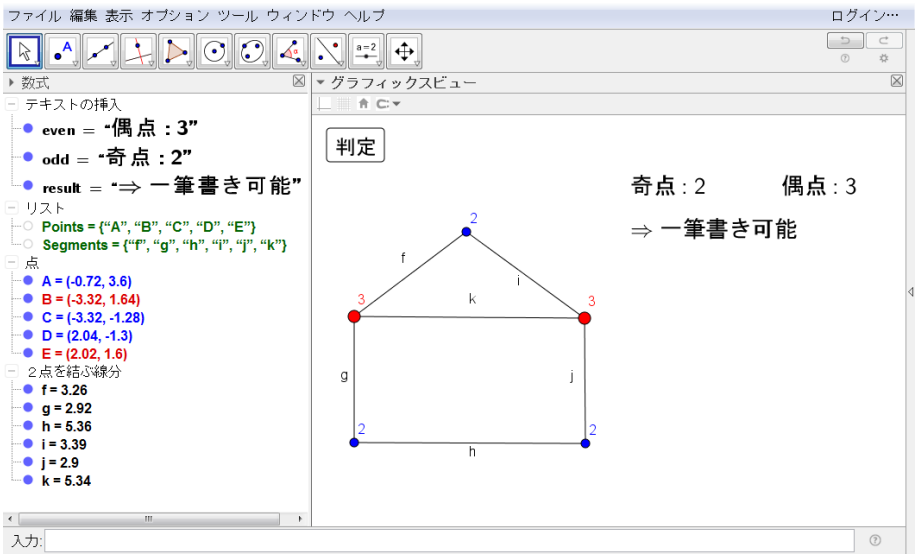

図 5　一筆書き判定

2.5　【ステップ 5】一筆書き順路の生成

奇点が 0 個の場合は一筆書き順路の始点としてどの点を選んでも良い。そして、その点が終点になる。この順路を求めるために次の Hierholzer のアルゴリズム（例えば [5] の Algorithm 5.1.2）を用いる。

Algorithm 1 一筆書き順路の生成（奇点が 0 個の場合）

Input: 奇点が 0 個の連結グラフ G

Output: 始点 v と順路 C

1: 始点 v を G に含まれる点の中からランダムに選ぶ。
2: v を出発して未通過の辺を経由して v に戻る順路 C を見つける。
3: **while** $C \neq G$ **do**
4: 　C に含まれ、未通過の辺に接続する点を v' を見つける。
5: 　v' を出発して未通過の辺を経由して v' に戻る順路 C' を見つける。
6: 　(v を始点として C を v' に到着するまで順に辿る) → (C' を順に辿る) → (C の残りを v に到着するまで順に辿る) という順路を改めて C とおく。
7: **end while**
8: **return** v, C

奇点が 2 個の場合は一方が一筆書き順路の始点となり、もう一方が終点になる。この順路を求めるアルゴリズムは Algorithm 1 を少し修正すればよい。

Algorithm 2 一筆書き順路の生成（奇点が 2 個の場合）

Input: 奇点が 2 個の連結グラフ G

Output: 始点 v と順路 C

1: G に含まれる 2 個の奇点を v, w とおく。
2: v を出発して未通過の辺を経由して w に至る順路 C を見つける。
3: **while** $C \neq G$ **do**
4: C に含まれ、未通過の辺に接続する点を v' を見つける。
5: v' を出発して未通過の辺を経由して v' に戻る順路 C' を見つける。
6: (v を始点として C を v' に到着するまで順に辿る) → (C' を順に辿る) → (C の残りを w に到着するまで順に辿る) という順路を改めて C とおく。
7: **end while**
8: **return** v, C

これらのアルゴリズムを JavaScript で実装しよう。まず、ツールバーの右から 2 つ目のツールボタンの▽から「ボタン」を選択後、描画エリア（グラフィックスビュー）をクリックして順路ボタンを作成する。そして、順路ボタンを右クリック→「オブジェクトのプロパティ」→「スクリプト記述」→「クリックして」欄を開き、下部のドロップダウンリストで入力するコードを「GeoGebra Script」から「JavaScript」に変更してから、以下のコードを入力する。

```
connect = [];
segment = [];
path = [];
for (j = 0; j < Sname.length; j++) {
  segment.push([]);
  path.push(false);
}
for (i = 0; i < Pname.length; i++) {
  ggbApplet.evalCommand('tmpC=Circle(' + Pname[i] +
    ',0.5)');
  seg = [];
  for (j = 0; j < Sname.length; j++) {
    ggbApplet.evalCommand('tmpP=Intersect(tmpC, ' + Sname[j
      ] + ')');
    if (ggbApplet.isDefined('tmpP')) {
      seg.push(j);
      segment[j].push(i);
    }
```

```
    ggbApplet.deleteObject('tmpP');
  }
  connect.push(seg);
  ggbApplet.deleteObject('tmpC');
}
odd = [];
for (i = 0; i < Pname.length; i++) {
  if (connect[i].length % 2 != 0) odd.push(i);
}
```

ここでは、ステップ 2 で作成したコードをベースに順路生成アルゴリズムで用いる次の 4 つの配列を準備している。

- connect: 長さが Pname に等しく、i 番目の要素は点 Pname[i] に接続する線分 (Sname のインデックスによる) の配列。接続線分情報として利用。
- segment: 長さが Sname に等しく、i 番目の要素は線分 Sname[i] の端点 (Pname のインデックスによる) の配列。線分端点情報として利用。
- path: 長さが Sname に等しく、i 番目の要素は線分 Sname[i] が通過済みならば true、未通過ならば false の配列。初期値は全て false。通過線分情報として利用。
- odd: 奇点と判定された点 (Pname[i] のインデックスによる) の配列。奇点情報として利用。

ただし、Pname は点のオブジェクト名を格納した配列、Sname は線分のオブジェクト名を格納した配列である。

以上の配列を利用して次の 5 つの関数を定義する。

- get_path(start,end): start を始点、end を終点とする順路（全線分を通る必要はない）を求める。
- nextv(c): 順路 c 上にあり、未通過の線分に接続されている点を求める。
- complete(): 全線分を通過したか判定する。
- euler_path(start,end): start を始点、end を終点とする（全線分を通る）順路を求める。
- euler(): 順路生成のメイン関数

これらのコードは全部で 100 行近くあるため、本稿に載せることができない。以下の URL からコピーして、順路ボタンを右クリック→「オブジェクトのプロパティ」→「スクリプト記述」→「クリックして」欄に**追加**する。

URL https://ww1.fukuoka-edu.ac.jp/~fujimoto/geogebra/euler.html

2.6 【ステップ 6】解答アニメーションの作成

ここでは、ステップ 5 で求めた一筆書き順路をアニメーションで表示する。GeoGebra には Point(線分リスト, スライダー変数) というコマンドがあり、スライダー変数の値を 0 ～ 1 に変化させるだけで、線分リストで与えられた線分上を点が順に移動する。このコマンドと残像表示（トレース）機能を用いれば、一筆書きの解答アニメーションが簡単に作成できる。

まず、スライダー変数 t を用意する。ツールバーの右から 2 つ目のツールボタンの▽から「スライダー」を選択後、描画エリア（グラフィックスビュー）をクリックする。現れたダイアログで、名前を t、区間を「最小 : 0，最大 : 1，増分 : 0.001」と設定し、さらに [アニメーション] タブの「反復」を「⇔ 振動」から「⇒ 増加（1 回)」に変更する。

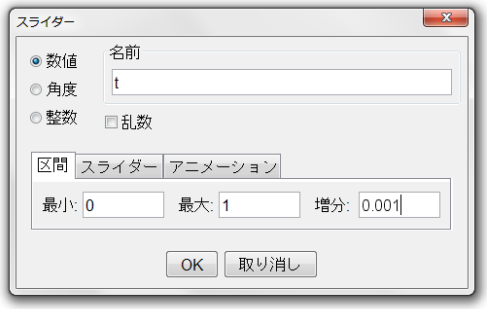

図 6　スライダーの区間設定

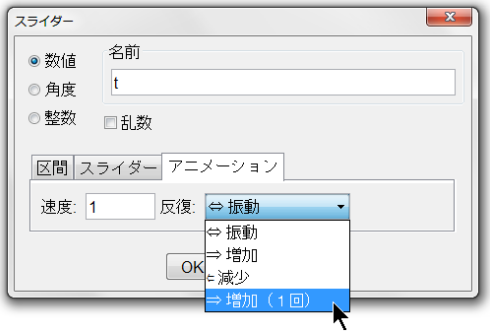

図 7　アニメーション設定

そして、順路ボタンを右クリック→「オブジェクトのプロパティ」→「スクリプト記述」→「クリックして」欄に以下のコードを**追加**する。

```
ans = euler();
if (ans.length != 0) {
  ans_path = [];
  for (i = 0; i < ans.length; i++) {
    ans_path.push(Sname[ans[i]]);
  }
  alert("一筆書き順路 : " + ans_path);
  ggbApplet.evalCommand('TraceP=Point({' + ans_path + '},t
    )');
  ggbApplet.setTrace('TraceP', true);
  ggbApplet.setLabelVisible('TraceP', false);
  ggbApplet.evalCommand('SetValue(t,0)');
  ggbApplet.setAnimating('t', true);
  ggbApplet.startAnimation();
}
```

描画エリア（グラフィックスビュー）に一筆書き可能な図を描き、順路ボタンをクリックしてみよう。まず、一筆書きの順路が次のように「線分オブジェクトの列」で表示される。

図 8　順路の表示

続いて、OKボタンをクリックすると、順路を示すアニメーションが表示される。

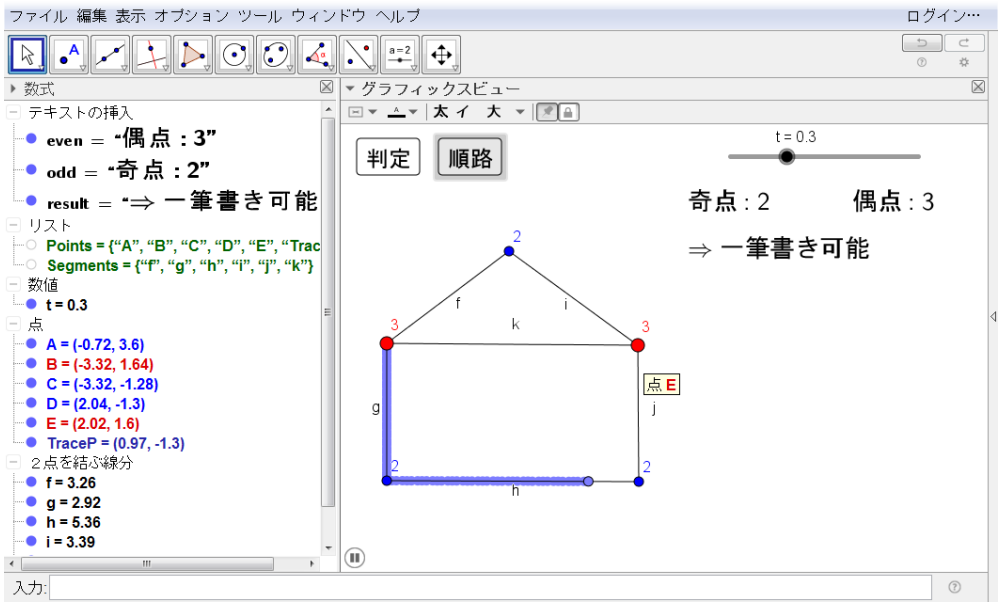

図 9　順路のアニメーション

2.7　【ステップ 7】リセットボタンの作成

最後にリセットボタンを作成する。このボタンをクリックすることで、判定ボタンの実行結果や一筆書きアニメーションの残像を消去する。

ツールバーの右から 2 つ目のツールボタンの▽から「ボタン」を選択後、描画エリア（グラフィックスビュー）をクリックしてリセットボタンを作成する。そして、リセットボタンを右クリック→「オブジェクトのプロパティ」→「スクリプト記述」→「クリックして」欄を開き、下部のドロップダウンリストで入力するコードを「GeoGebra Script」から「JavaScript」に変更してから、以下のコードを入力する。

```
1 ggbApplet.refreshViews();
2 ggbApplet.deleteObject('even');
```

```
ggbApplet.deleteObject('odd');
ggbApplet.deleteObject('result');
ggbApplet.deleteObject('TraceP');
ggbApplet.evalCommand('SetValue(t,0)');
```

以上で一筆書きアプリの完成である。

2.8 演習課題

ここまでの全 7 ステップを実際に実行すれば、JavaScript を用いた GeoGebra アプリ作成のコツを掴むことができると思われる。その後、次のような課題に取り組むことで更に理解が深まるはずである。

課題 1 順路生成のアルゴリズムの過程が見えるように修正せよ。

課題 2 描いた図が連結かどうか判定するボタンを作成せよ。

課題 3 一筆書きの順路を求める Fleury のアルゴリズムを実装し、Hierholzer のアルゴリズムと比較せよ。

課題 4 線分だけでなく円弧やペンストロークを用いた図でも動作するように修正せよ。

課題 5 ペンストロークで自由に描いた図に対して、自動的に交点を求め、一筆書きの可否判定や順路生成ができるようにせよ。

3 おわりに

本稿では、JavaScript を用いることで、GeoGebra 上に入力した点や線分などをオブジェクト別に (JavaScript 側の) 配列で管理し、さらにそれらを利用することによって、一筆書きの可否判定や順路生成のようなアルゴリズムを実装できることを示した。オブジェクト別の情報を JavaScript 側でなく、「GeoGebra オブジェクトのリスト」として管理することも可能であるが、JavaScript の方が実装は簡単だと思われる。

本稿で作成した GeoGebra ファイル（.ggb ファイル）は [6] から入手可能である。プログラミングの学習は実際に動かしてみるのが一番である。このファイルをベースに 2.8 で挙げたような機能を追加しながら学習していただきたい。自分が作成した JavaScript のコードが思い通りに動かない時は、GeoGebra 公式サイトの Help Center での議論 (例えば [7, 8] など) やインターネット上の（GeoGebra とは無関係の）JavaScript に関する掲示板及びサンプルコードが参考になるはずである。また、デバッグ作業で変数の値を確認したい時は JavaScript の alert コマンドを用いて表示すると良いだろう。今後も多くの楽しい GeoGebra アプリが生み出されることを期待する。

参考文献

[1] 藤本光史, How to create sliding puzzles on GeoGebra – A tutorial of GeoGebra Script –, 統計数理研究所共同研究リポート 396, 動的幾何学ソフトウェア GeoGebra の整備と普及 (3), (2018) 33–42.

[2] GeoGebra 日本, D JavaScript, https://sites.google.com/site/geogebrajp/script/c-javascriptno-gai-yao

[3] The GeoGebra Team, Reference: GeoGebra Apps API, https://wiki.geogebra.org/en/Reference:GeoGebra_Apps_API

[4] 百武卓人, GeoGebra のスクリプトによる拡張, 福岡教育大学卒業論文, 2018 年 1 月.

[5] Ronald Gould, *Graph thery*, Dover Books on Mathematics, Dover Publications, 2012.

[6] 藤本光史, 一筆書きアプリ（GeoGebra）, https://ww1.fukuoka-edu.ac.jp/~fujimoto/geogebra/eulerian.ggb

[7] GeoGebra Topic: Getting all the points, https://help.geogebra.org/topic/getting-all-the-points

[8] GeoGebra Topic: How to set the font size of a text object by script?, https://help.geogebra.org/topic/how-to-set-the-font-size-of-a-text-object-by-script