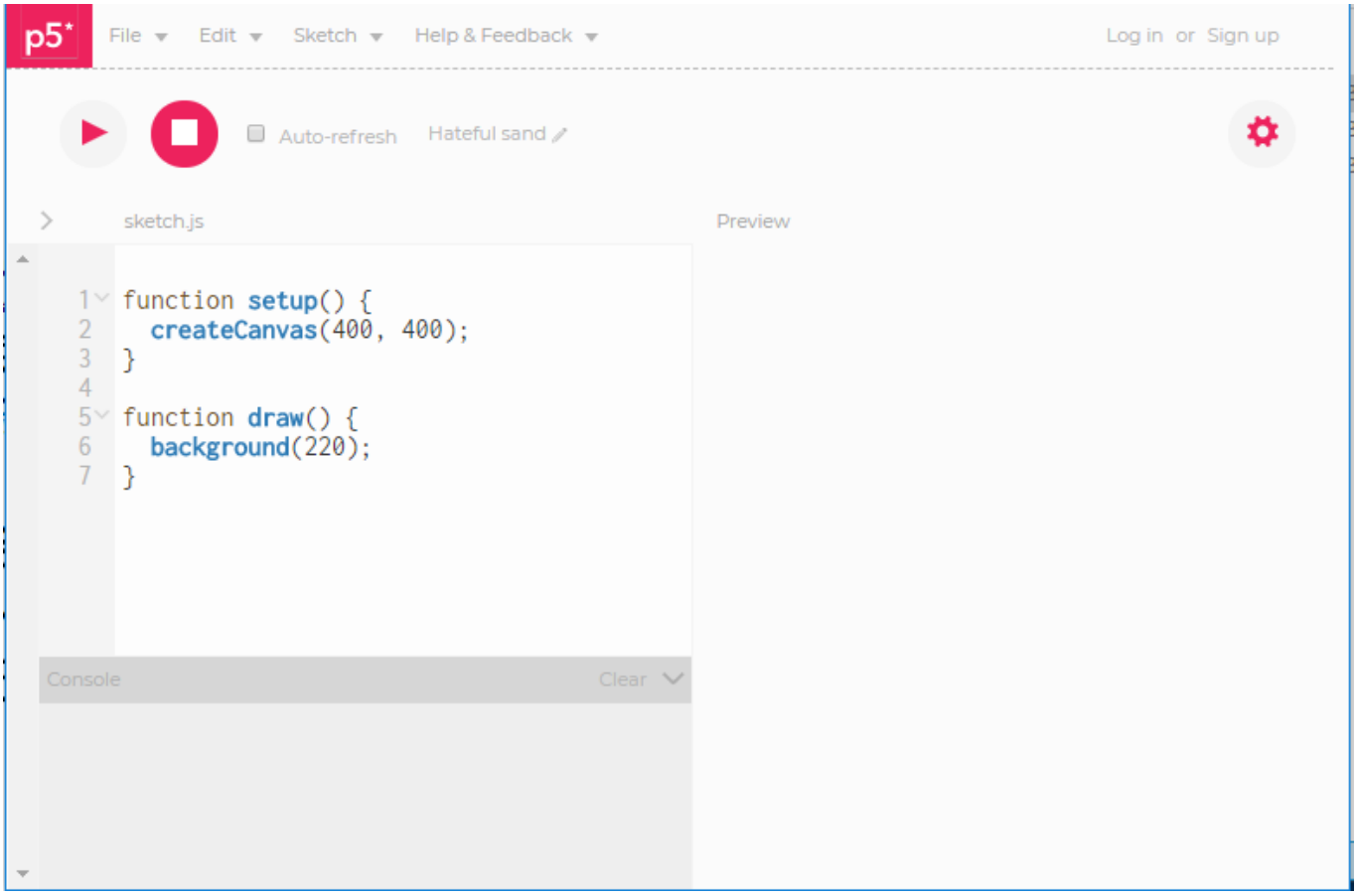
プログラミングで遊ぶ：セルオートマトン編

セルオートマトンは格子状のセルと単純な規則による計算モデルです。Gollyという大規模シミュレータもありますが、今回は、p5.js という描画用言語でのプログラミングを通じて、セルオートマトンの世界を探検します。



１． p5.js の利用

p5.js のウェブエディタ（https://alpha.editor.p5js.org/）にアクセスして利用します。（右図）

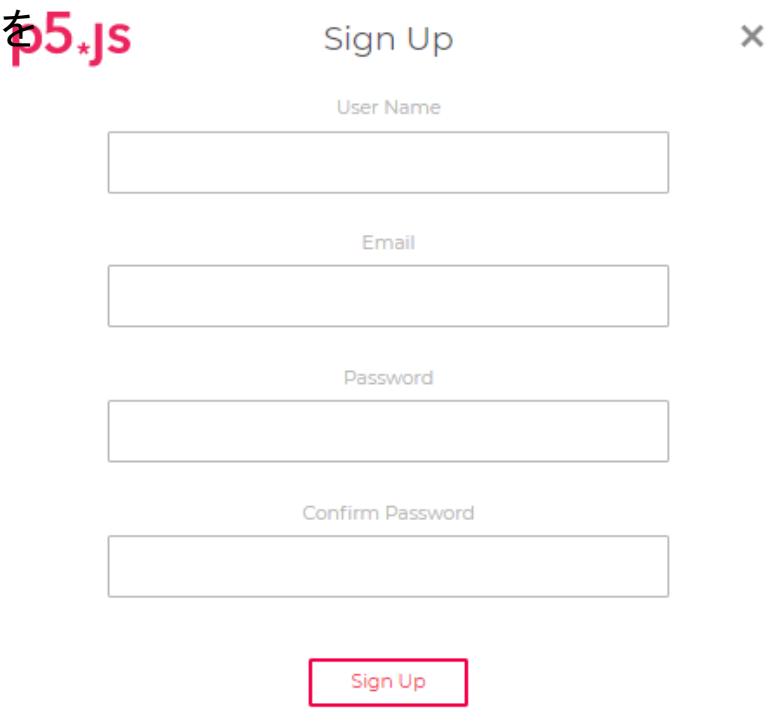
1. 図左のエディタ欄にプログラムを入力し(▶)で実行すると、結果が右の Preview 欄に表示されます
2. draw()関数はデフォルトで毎秒60回実行され、(■)で停止します
3. プロジェクト名は、最初自動的につきますがクリックして変更できます
4. ユーザ登録しなくても使えますが、プログラムの保存(読込)ができないので、以下の手順で登録を行います

利用登録【Sign Up】

1. 利用に際し、次の情報をあらかじめ決めておきます
   User Name ニックネームでかまいません
   Email Emailアドレス

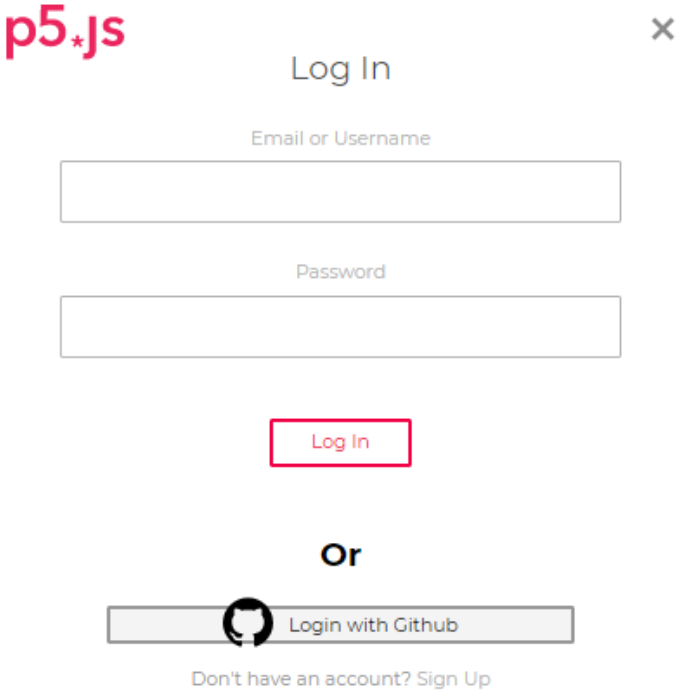
Password　忘れないものを

2. ウェブエディタ画面右上の Sign up をクリックすると、右の画面になります
3. １．の情報を入力（Confirm Password欄には確認のためパスワードを再入力）し Sign Up をクリックします
4. 現れた画面の右上に Hello ユーザ名！ と表示されれば O.K. です



再利用【Log In】

1. ウェブエディタ画面右上の Log In をクリックすると、右の画面になります
2. 利用登録時に入力した情報、「Emailアドレス又は User Name」と「パスワード」を入力し Log In をクリックします
3. 現れた画面の右上に Hello ユーザ名！ と表示されればO.K. です

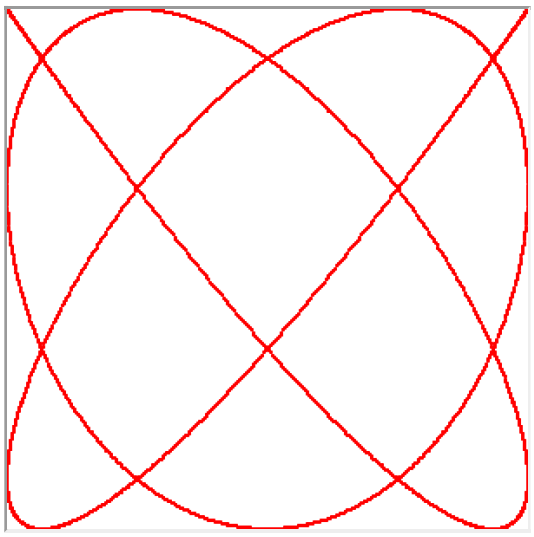


２．最初の p5.js プログラム

リサジュー図形はセルオートマトンではありませんが、p5.js の簡単な入門を兼ねて、作成します。

このプログラム は $(\cos(nt), \sin(mt+d))$（$0 \leqq t \leqq 360°$）で媒介変数表示されるリサジュー図形を描きます。プログラムを自分のp5.jsウェブエディタにコピーし、**パラメータ** n, m, d の値を変えながら実行させてみましょう。

アニメーション版 はdraw()関数を利用して、描画過程をアニメーション表示します。

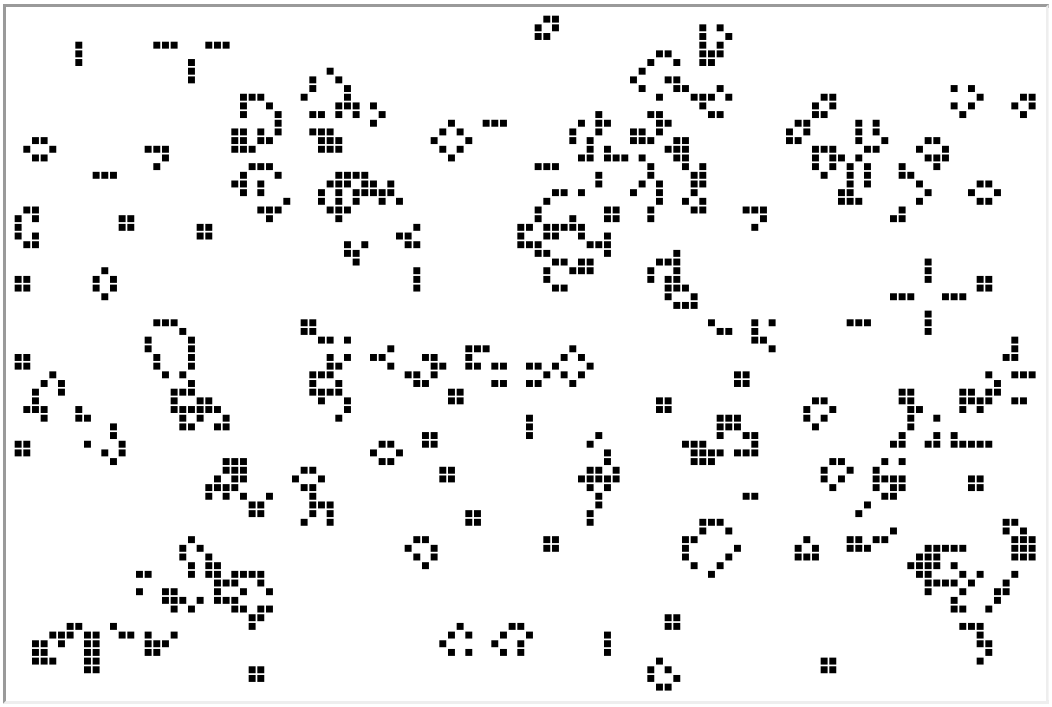


【補足】自分でプログラミングしてみようという方へ（p5.jsレファレンス）

- createCanvas$(width, height)$ は、左上が $(0,0)$ で右下が $(width-1, height-1)$ の描画面を生成します
- プログラムは、左上が $(-1,1)$ で右下が $(1,-1)$ の領域内の点(x,y)を描画面の点 $(width/2*(1+x), height/2*(1-y))$ にマッピングしています
- point(x,y) は、位置 (x,y) に指定された色と太さの点を表示します
- 三角関数 sin(),cos() に与える値はラジアン(弧度法)なので、radians() で変換しています
- for (var t=0; t<360; t+=360/maxt) ・・・;
  t の値を 0° から360° まで、360/maxt° ずつ増やしながら、・・・を繰り返します
- 描画に不具合が生じたら、歯車アイコンから Acessibiliy を開き、Accessible text-based canvas のチェックをすべて外してください

このプログラムは、子供のころよく遊んだスピログラフ®を描きます。パラメータを色々変えて楽しんでください。

３．ライフゲーム



イギリスの数学者コンウェイが定義しました。２次元格子の各セルには、周囲８個の隣接セル(近傍)があり、次の規則で生き死にを繰返します。

1. 生き(黒)セルは、近傍に２または３個の生きセルがいるときのみ生き続ける
2. 死に(白)セルは、近傍に３個の生きセルがいるときのみ生き返る

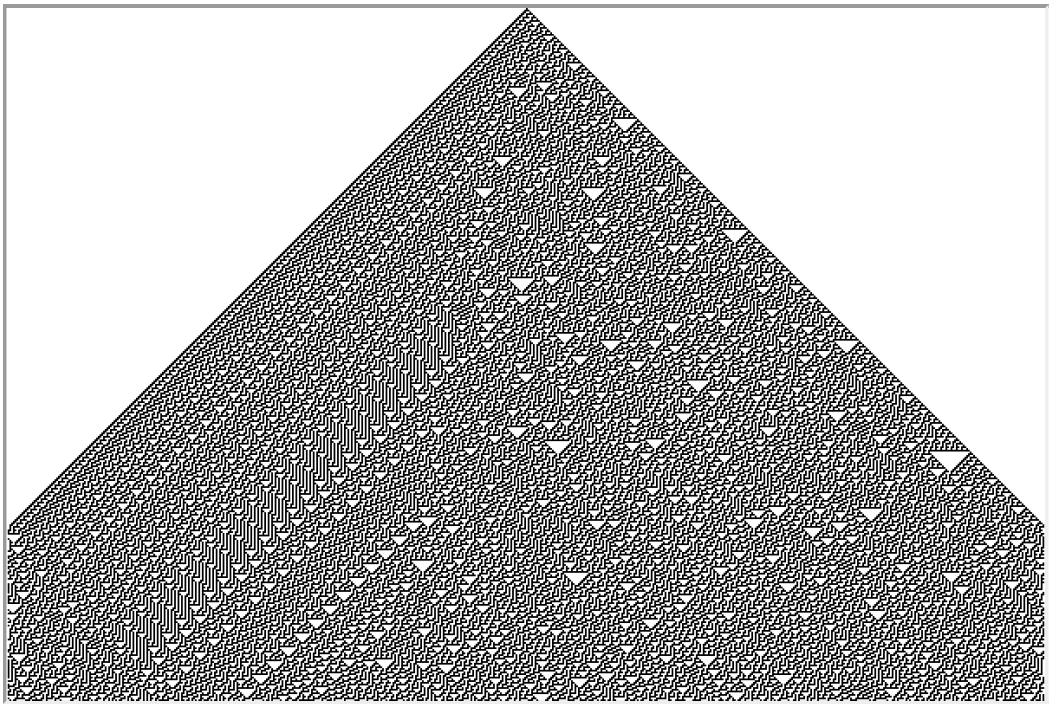
これだけの規則で、セル空間は複雑な挙動を示し、コンピュータと同等の複雑さ(計算能力を持つ)ことが知られています。右のデモは無秩序な配置から開始しています。画面を(左・右)クリックすると手動設定モードに移り、左クリックで生きセルを、D,R,Kキーでパターンを配置でき、[Space]キーで再開します。

プログラムはこちらです。様々な初期配置を楽しんでください。 参考

書：ライフゲイムの宇宙

４．一次元セルオートマトン

30番	7	6	5	4	3	2	1	0
現状態	111	110	101	100	011	010	001	000
次状態	-0-	-0-	-0-	-1-	-1-	-1-	-1-	-0-



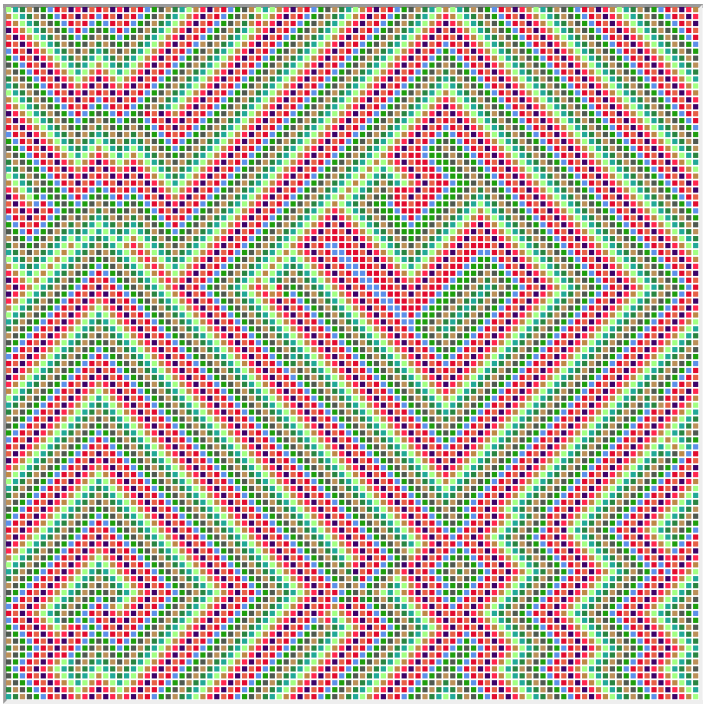
一列に並んだセルの場合、左右の２セル近傍の２状態セルでは、近傍も含めた状態は右の7～0の8通りになり、次の状態を定める規則は８桁の２進数、つまり0～255の10進数で番号づけられます。デモは 00011110＝30番の規則で、中央のセルだけが 1 の状態から成長していく様子を表しています。こちらのプログラムで様々な規則番号での成長の様子を楽しんで下さい。規則90番(01011010)ではフラクタル図形であるシェルピンスキーのガスケットが現れます。規則184番(10111000)は右方向へ進む車列を模しています。出現確率を変えて、渋滞(黒い線)が現れ逆方向に進む様子を観察してみましょう。島根大学岩本先生の解説も参考になります。

５．二次元セルオートマトン

こちらのプログラムをもとに二次元セルオートマトンの世界を探検してみましょう。2次元セル平面は本来無限格子上に広がっていますが、コンピュータでは有限の領域しか扱えません。そこで周囲に境（状態0のセル）が有るとする場合と、左と右、上と下がつながり（トーラスといいます）境が無いとする場合があります。プログラムでは変数 境 の値でその有り(1)無し(0)を区別しています。

セル近傍（隣接セル）には、上下左右の4セル近傍とそれに斜め4方向を加えた周囲8セル近傍とがあります。プログラムでは // で一方をコメントアウトすることで選べるようにしています。

プログラムの例は、境の無い平面でセルは4近傍、状態数は12（0～11）です。次状態は基本今の状態を維持しますが、近傍に自分の状態番号



+1（ただし状態数に達すると 0 に戻る）があればその状態という規則になっています。セル平面は最初混沌としていますが、やがて水滴のように同色の領域が現れては消えし、最後はいくつかのらせんで埋め尽くされるようになります。状態数や境、近傍を変えて楽しんでみてください。

こちらのプログラムは２大政党制の下での支持者分布の変化を模倣します。２状態で周囲８近傍、境界無し、遷移規則は「自分を含めた9人の中で多数派の政党を支持するが、多数派が5人の場合は少数派の政党を支持する」というものです。条件を変えて試してみてください。こちらのプログラムではごった煮の鍋の様子（？）が再現されます。また柴田克己氏のサイトも参考になります。

```
var 状態数 = 12, //状態は 0～状態数-1
    近傍 = [[1,-1,0,0],[0,0,1,-1]], //上下左右4近傍
           //[[1,0,-1,1,-1,1,0,-1],[1,1,1,0,0,-1,-1,-1]], //周囲8近傍
    境 = 0, //0:境界無＝トーラス。1:状態0の境界有
    幅 = 4, //セルサイズ
    描画速度 = 60;
var 色=[], 行数, 列数, 盤=[], 次盤=[], 行, 列;

function setup() { createCanvas(400,400); frameRate(描画速度); noStroke();
  for(k=0; k<状態数; k++) //状態に付ける色を設定
    色[k] = color(random(256),random(256),random(256));
  列数 = floor(width/幅); 行数 = floor(height/幅);
  for(列=0;列<列数;列++){ // 盤の生成と初期設定
    盤[列] = new Array(行数).fill(0); 次盤[列] = new Array(行数).fill(0);
    for(行=境;行<行数-境;行++) 盤[列][行] = floor(random(状態数)); //ランダム割当
}}
function draw() {//盤を表示後、次の盤を計算
  for(列=境;列<列数-境;列++)for(行=境;行<行数-境;行++){
    fill(色[盤[列][行]]); rect(列*幅,行*幅,幅-1,幅-1); //セルの描画
    次盤[列][行] = 盤[列][行]; //基本は状態維持
    for(k=0; k<近傍[0].length; k++){//盤[c][r]は近傍セル
      c = 境==1?列+近傍[0][k]:(列+近傍[0][k]+列数)%列数;
      r = 境==1?行+近傍[1][k]:(行+近傍[1][k]+行数)%行数;
      //規則：近傍に1上の状態があれば、その状態に
      if (盤[c][r]==(盤[列][行]+1)%状態数) 次盤[列][行] = 盤[c][r];
    }
  }
  退避 = 盤; 盤 = 次盤; 次盤 = 退避; //次盤⇔盤
}
```

```
s = 描画速度;//画面クリックで停止／再開
function mouseClicked(){ frameRate(s =描画速度-s); }
```
