10.3 なんちゃってライフゲーム

それでは “なんちゃってライフゲーム” のコードを披露しよう。悪い見本かもしれないけど。

programming list [Lifegame.cpp]

```
#include <iostream>

class Lifegame {
    int p[6][6];
    int t[6][6];
    std::string d[6][6];

public:
    Lifegame(int s[][6], int rows) {
        for(int x = 0; x < rows; x++) {
            for(int y = 0; y < 6; y++) {
                p[x][y] = s[x][y];
                if(s[x][y] == 1) {
                    d[x][y] = "＊";
                } else {
                    d[x][y] = "・";
                }
            }
        }
    }

    void chkaround() {
        for(int x = 1; x < 5; x++) {
            for(int y = 1; y < 5; y++) {
                t[x][y] = p[x-1][y-1] + p[x-1][y  ] + p[x-1][y+1]
                        + p[x  ][y-1]               + p[x  ][y+1]
                        + p[x+1][y-1] + p[x+1][y  ] + p[x+1][y+1];
            }
        }
    }

    void replace() {
        for(int x = 1; x < 5; x++) {
            for(int y = 1; y < 5; y++) {
                if(p[x][y] == 1) {
                    switch (t[x][y]) {
                        case 2:  p[x][y] = 1; d[x][y] = "＊"; break;
                        case 3:  p[x][y] = 1; d[x][y] = "＊"; break;
                        default: p[x][y] = 0; d[x][y] = "・"; break;
                    }
                } else {
                    switch (t[x][y]) {
                        case 3:  p[x][y] = 1; d[x][y] = "＊"; break;
                        default: p[x][y] = 0; d[x][y] = "・"; break;
```

```
43:                     }
44:                 }
45:             }
46:         }
47:     }
48:
49:     void disp() {
50:         for(int x = 1; x < 5; x++) {
51:             for(int y = 1; y < 5; y++) {
52:                 std::cout << d[x][y];
53:             }
54:             std::cout << std::endl;
55:         }
56:         std::cout << std::endl;
57:     }
58: };
59:
60: int main() {
61:     int s[][6] = {{0, 0, 0, 0, 0, 0},
62:                   {0, 0, 1, 0, 0, 0},
63:                   {0, 0, 0, 1, 0, 0},
64:                   {0, 1, 1, 1, 0, 0},
65:                   {0, 0, 0, 0, 0, 0},
66:                   {0, 0, 0, 0, 0, 0}};
67:     Lifegame cells = Lifegame(s, 6);
68:
69:     int gen = 4;
70:     cells.disp();
71:     while(gen--) {
72:         cells.chkaround();
73:         cells.replace();
74:         cells.disp();
75:     }
76:     return 0;
77: }
```

あとは前節のテスト用コードに、マスの周りの状況が格納された `t[][]` を見ながら、`t[][]` の値にしたがってマスの 活き/排除/誕生 を区別すればよいだけだ。それが 30:行目からの `replace()` 関数である。`switch` 構文はこういうときに有効である。

さて、4 世代分を表示するように 69:行目で `gen = 4` としてみた。直後に `cells.disp();` を実行しているのは、まず初期配置を表示したかったからである。いわば 0 世代を表示することになる。さて、プログラムを実行すると、グライダーが 4 世代で右下へ 1 マス移動するのが分かるだろう。この様子を Terminal の画面で見るのもオツなものではないかな。

動画を見たければよい方法がある。それは、Terminal やコマンドプロンプトの画面の縦サイズを縮めることだ（おそらく 7 行だけ表示されるようにすればよいと思う）。そして、プログラムの

`while` 構文の中、たとえば `cells.disp();` の一行上に

```
for(int i = 0; i < 500000000; i++) {}
```

の一文を挿入しよう。数値は君たちのコンピュータの性能によって適切に変えてくれたまえ。すると、**C++**は画面に 7 行分の表示をして、挿入した `for` のためにコンマ何秒か無駄にし、次の表示をする。どうだ、動画っぽくなっただろう。

TRY! `s[][]` の初期値をいろいろ変えて試してみよ。以下に、ライフゲームで代表的な初期配置を示した。それぞれに、それっぽい名前がついているのだが、納得できる名前だろうか。周期は、再び同じ配置になるまでの世代を意味する。したがって、周期 1 は変化しない形である。

点滅灯（周期 2）

-	-	-	-	-	-
-					-
-	●	●	●		-
-					-
-					-
-	-	-	-	-	-

ビーコン（周期 2）

-	-	-	-	-	-
-	●	●			-
-	●	●			-
-			●	●	-
-			●	●	-
-	-	-	-	-	-

ヒキガエル（周期 2）

-	-	-	-	-	-
-					-
-	●	●	●		-
-		●	●	●	-
-					-
-	-	-	-	-	-

ボート（周期 1）

-	-	-	-	-	-
-		●	●		-
-	●			●	-
-		●	●		-
-					-
-	-	-	-	-	-

TRY! 盤面を大きくしてシミュレーションできるように変更してみよ。次の “八角形” は周期 5 で、ちょっと面白い。

-	-	-	-	-	-	-	-	-	-
-				●	●				-
-			●			●			-
-		●					●		-
-	●							●	-
-	●							●	-
-		●					●		-
-			●			●			-
-				●	●				-
-	-	-	-	-	-	-	-	-	-

TRY! このクラスでは汎用的なクラスとはいえない。任意のサイズ、たとえば横 x マス、縦 y マスの盤面が設定できたり、任意の初期配置が入力できたりするようなクラスに書き換えてみよ。

めくじらを立てることもないが、今回の景勝地はクラスのことよりアルゴリズムの方が改善の必要がある。いまは常にすべてのマスの周りを走査しているが、それでは盤面を大きくすると計算が追いつかない。実際はほとんどのマスは周囲の計算は不要なのだ。でも、それは君たちへの課題だ。十人十色、やり方はいろいろある。工夫を楽しもうじゃないか。