Corner The Queen ゲームと Grundy 数

Shigeo Hayashi, Jan. 17, 2024; Aug. 30, 2023

「高校生にMathematica!?」という問題提起から始まった話である。

1　Corner the Queen ゲーム

Corner the Queen という名前のゲームがある。訳せば「早く女王様を隅にお連れ申せ」となろうか。チェス盤を挟んで二人が競うゲームである。チェス盤の一隅がゴールであり、縦か横か斜めの3方向でしかゴールに近づけない女王様を二人が交互に移動させる。ゴールにお連れした側が勝ちである。以後、なじみの深い将棋盤を想定しながら話を進める。まず女王様は駒である (フィギュアのほうが現代的という意見もあろう)。女王様がいるのはマス目の内側である (碁盤のように線が交わるところにしても構わないのであるが)。そしてマス目には縦横に $0..(N-1)$ と番号を割り振る。つまり縁のマス目はN個ある。

実はこのゲームは決定論的といって、スタート場所が自分に有利であったば場合、セオリーに従って指せば必ず勝てる。自分に不利な場所の場合は相手のミスを待つ。そのセオリーで重要な役割を果たすのがマス目に付随する Grundy 数である。

ここでセオリーの一端を紹介しよう。図1の $(2,1)$ にあなたが駒を置いたとする。ゴールに近づくような手は4通りあるが(図の中段)、どれをとっても相手は上がれない。あなたは悠々とゴールすることができる。$(2,1)$ のような位置を今後最強位置と呼ぶ[1]。

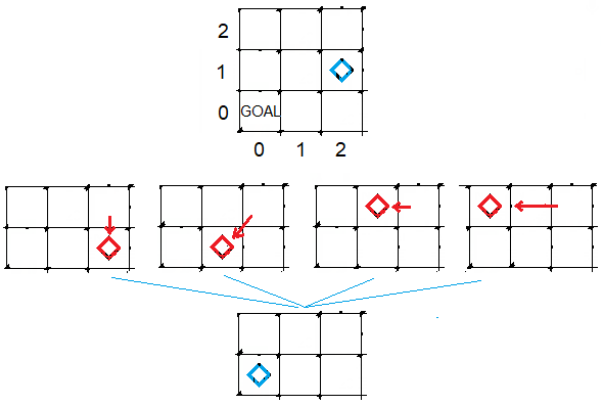

Figure 1: $(2,1)$ は最強位置

逆に置いてはだめという最悪位置がないか調べてみよう。相手が一手でゴールできる位置であるから、自分が置いた位置は、ゴールから縦・横・斜めに延びるマス目のどこかのはずである。そのような場所は図2 の$\times$ である。ではこの図の空白ならばどこに置いても勝てるのかという疑問が起こる。

[1] §5.1 のゲームソフトでは winning position と呼ばれている。

確かにゴールに最も近い$(2,1)$は図1で見たとおり最強位置であり、対角線について対称な$(1,2)$も同様に最強である。では$(3,1)$の空白に置くのはどうであろうか。この場合、相手が1マス左に動かせばそこは図1の最強位置である。図2 の空白は、最強になりうるが、そうでないものも混ざっている。

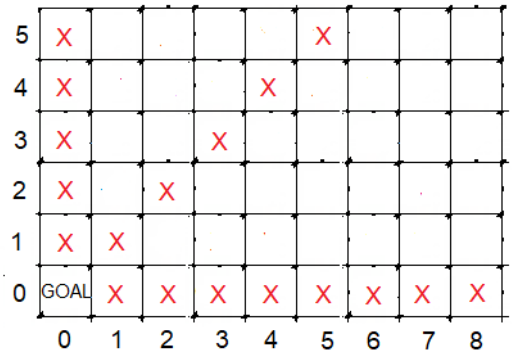

Figure 2: $\times$ に置くのは最悪

「混ざり具合」をきれいにするために図2 の最強位置から縦・横・斜めに$\times$を書き入れていこう。その結果が図3である。予想通りゴールに最も近い$(5,3)$と$(3,5)$は最強である。

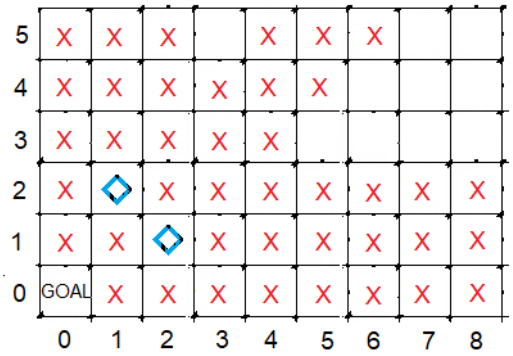

Figure 3: 空白の中に次の最強位置がある

2 Grundy数

最強位置をもっと高い視点から取りあげよう。

Queen の場所を、横座標 $i = 0, 1, 2, \cdots, N-1$ と縦座標 $j = 0, 1, 2, \cdots, N-1$ で表すことにする。 Grundy数 $G_{i,j}$ jはすべてのマス(i,j)に付随する整数であり、最強位置でゼロをとる。他のマスでは正である。ここでは天下り式に定義式を提示しよう。まず

$$G_{i,j} = \mathcal{G}(\mathbf{r}_{i,j}), \quad \mathbf{r}_{i,j} = (i,j) \tag{1}$$

によって$\mathcal{G}$を定義する。

$$\mathcal{G}(\mathbf{p}) = \mathrm{mex}\,\{\mathcal{G}(\mathbf{q}) : \mathbf{q} \in \mathrm{move}\ \mathbf{p}\} \tag{2}$$

ここで move($\mathbf{p}$) は $\mathbf{p}$ から縦・横・斜めに一手で行けるマス目の集合である。ただし、$\mathbf{p}$自体はその集合に含まれない。 mex(S) は，非負整数からなる集合 $\Gamma = \{0, 1, 2, \cdots\}$ から S を除外したあと、残りの要素の中で最も小さい数を返す関数である。言い換えればSの補集合における最小値を返す関数である。実際の計算においては

$$\Gamma = \{0, 1, 2, \cdots, \gamma\} \tag{3}$$

と有限集合にするほうが処理しやすいかもしれない。$\gamma = S$の要素の数 $+ 1$は一つの選択肢である。

例を挙げると、 Sがゼロを含まなければ $\mathrm{mex}(S) = 0$である。 0, 1, 2, 4, 5 $\cdots$のように途中で a_k が抜けていれば $\mathrm{mex}(S) = a_k$ である。 0, 1, 2, 3, 4 のように途中で抜けがなければ $\mathrm{mex}(S) = 5$である。

$\mathbf{q}$ が原点にある場合、縦・横・斜めには行けないので改めて定義する必要がある。そこで

$$G_{0,0} = \mathcal{G}(\mathbf{0}) = 0 \tag{4}$$

とする。

式(2) は不思議な式であるが、順を追って解いてみると意外と簡単である。まずゴールに隣接した3つのマス目について

$$\begin{aligned} G_{0,1} &= \mathrm{mex}\{G_{0,0}\} = 1 \\ G_{1,0} &= \mathrm{mex}\{G_{0,0}\} = 1 \\ G_{1,1} &= \mathrm{mex}\{G_{0,1}, G_{0,0}, G_{1,0}\} = \mathrm{mex}\{1, 0, 1\} = 2 \end{aligned}$$

が得られる。

それを囲む5個のマス目について計算すると次が得られる。

$$\begin{aligned} G_{0,2} &= \mathrm{mex}\{G_{0,1}, G_{0,0}\} = \mathrm{mex}\{1, 0\} = 2 \\ G_{2,0} &= \mathrm{mex}\{G_{1,0}, G_{0,0}\} = \mathrm{mex}\{1, 0\} = 2 \\ G_{1,2} &= \mathrm{mex}\{G_{0,2}, G_{0,1}, G_{1,1}, G_{1,0}\} = \mathrm{mex}\{2, 1, 2, 1\} = 0 \\ G_{2,1} &= \mathrm{mex}\{G_{2,0}, G_{1,0}, G_{1,1}, G_{0,1}\} = \mathrm{mex}\{2, 1, 2, 1\} = 0 \\ G_{2,2} &= \mathrm{mex}\{G_{0,2}, G_{12}, G_{1,1}, G_{0,0}, G_{2,1}, G_{2,0}\} = \ \mathrm{mex}\{2, 0, 2, 0, 0, 2\} = 1 \end{aligned}$$

$G_{i.0}, G_{0,j}$の値は上の方式を参考にして一般化できる。$G_{i.0}$についていえば move は横軸沿いのみであるから mex 関数の入力パラメータは

$$G_{0,0}, G_{1,0}, \cdots, G_{i-1,0}$$

である。よって

$$G_{i,0} = \mathrm{mex}\{0, G_{1,0}, \cdots, G_{i-1,0}\}$$

であり、

$$G_{i,0} = j \tag{5}$$

は解である。同様にして$G_{0,j} = j$である。

以上の手順を整理すると図4のようになる。nが小さいものから、そして同じnなら両サイドから対角線位置に進む。

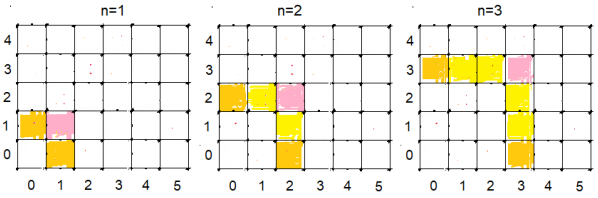

Figure 4: $G_{i,j}$を求める順序

3　コンピュータ·プログラムによる Grundy数の計算

3.1　Mathematica

宮寺と福井[1]は、Mathematicaを用いて簡潔なプログラムを作成している。筆者は Mathematica のライセンスを取得してそのプログラムを実際に走らせることができた。ここでは筆者の発想でそのプログラムを解説しよう。まず delayed assignment 記号 := を用いて以下の関数を宣言する。

```
moves[{x_,y_}]:= Union[Table[{u,y},{u,0,x-1}],Table[{x,v},{v,0,y-1}],
                       Table[{x-t,y-t},{t,1,Min[x,y]}]]]      (a)
mex[S_]:= Min[Complement[Range[0,Length[S]],S]]               (b)
gr[pos_]:= gr[pos]= mex[gr/@ moves[pos]]                      (c)
```

ここで式(a)の `moves[x,y]` は駒を置きうる範囲である (元のプログラムでは `move[z]`)。三つの `Table` が指定する場所は順に、(x,y)について横・縦の位置、そして(x,y)から斜めの位置である。

式(b) は $\gamma =$ `mex[S]` $+1$ である。いくつかの適用例を通して正しさを納得することにしよう。

```
mex[{1,1,2,2}]=0,  mex[{0,0,0,2,2,2}]=1
```

である。

式(c) では immediate assign 記号の = に従って代入し、結果をGrundy数 `gr[pos]`とする。前節では $G_{0,0}=0$ をあらかじめ指定する必要があると述べたが、ここではその必要がない。なぜなら `pos={0,0}` において `moves[pos]={}` であり、= の右辺が 0 となるからである。もちろん式(5) も必要ない。

ここで式(c) について補足しておこう。式(2)の = は等号 (equality) である。両辺を入れ替えても意味は変わらない。然るに

(i) 式(c)の = は assignment であるから両辺を入れ替えるとエラーが発生する。

(ii) 式(c)の = を equality test == に変えるとエラーが発生する (test 結果は Boolean)。

Grundy数の具体的な数値は、サイズ `len` を指定したのち (e)式を評価する際に求める。

```
len = 20                                                      (d)
Grid[Table[gr[{len-x,y}], {x,0,len}, {y,0,len}]]              (e)
```

Out[9]=

10	11	9	8	13	12	0	15	16	17	14
9	10	11	12	8	7	13	14	15	16	17
8	6	7	10	1	2	5	3	4	15	16
7	8	6	9	0	1	4	5	3	14	15
6	7	8	1	9	10	3	4	5	13	0
5	3	4	0	6	8	10	1	2	7	12
4	5	3	2	7	6	9	0	1	8	13
3	4	5	6	2	0	1	9	10	12	8
2	0	1	5	3	4	8	6	7	11	9
1	2	0	4	5	3	7	8	6	10	11
0	1	2	3	4	5	6	7	8	9	10

Figure 5: gr の値

その結果はFig.5の通りである。

さて宮寺と福井[1]はFig.5の基本構造をさらに広げて

```
Grid[Table[f[{n,m}], {n,-1,len}, {m,-1,len}], Option]        (f)
```

としている。`f[-1,m]=0,1,2,...,len` および `f[n,-1]=0,1,2,...,len` は座標軸の値である。そのような `f` を定義するコードは

```
f[x_,y_]:= Which[{x,y}=={-1,-1},Null, y==-1,x, x==-1,y,
          True, gr[{x,y}]]                                   (g)
```

であり3つの判断を含む (3つめは else に相当)。`Option` として基本的なものは枠線を描く `Frame` → `All` であり、式(f) を実行して Fig.6 が得られる。

Out[14]=

	0	1	2	3	4	5	6	7	8	9	10
0	0	1	2	3	4	5	6	7	8	9	10
1	1	2	0	4	5	3	7	8	6	10	11
2	2	0	1	5	3	4	8	6	7	11	9
3	3	4	5	6	2	0	1	9	10	12	8
4	4	5	3	2	7	6	9	0	1	8	13
5	5	3	4	0	6	8	10	1	2	7	12
6	6	7	8	1	9	10	3	4	5	13	0
7	7	8	6	9	0	1	4	5	3	14	15
8	8	6	7	10	1	2	5	3	4	15	16
9	9	10	11	12	8	7	13	14	15	16	17
10	10	11	9	8	13	12	0	15	16	17	14

Figure 6: Frame→All を付加した Grid

もう一つの `Option` は背景色の設定であり、`Background`を用いる。ここで留意すべきは、 Fig.6 の範囲が縦横 1 ～ `len+2` であることである。ここでは分かりやすい次式を用いよう。

```
Background -> {1->Yellow, 1->Cyan,
               Table[zpos[[s]]-> Orange, {s,Length[zpos]}]}        (h)
```

`zpos` について補足すると、宮寺と福井[1]は、すべての点の座標 {x,y} のリスト `allcases` を作り、gr[{x,y}]==0 を満足するものを`Select` で選んでいるが、筆者は別の方式を採用した。つまり

```
zposition = Flatten[Table[If[gr[{m,n}]==0, {m,n}, Nothing],
                          {m,0,len}, {n,0,len}]]                    (i)
```

でゼロ点を取り出し、

```
zpos = {#[[1]]+2,#[[2]]+2}&/@ zposition                             (j)
```

で原点を移動させた。この式(j) は他の流儀でも書けて、例えば

```
zpos = {Part[#,1]+2,Part[#,2]+2}&/@ zposition                       (j1)
zpos = (#+2)&/@ zposition                                           (j2)
zpos = Map[{#[[1]]+2,#[[2]]+2}&, zposition]                         (j3)
```

が挙げられる。式(j3)が原著者の方法である。どの方法でも zpos={{2,2},{3,4},...} である。さらに原点に$x\backslash y$ を入れるために式(f)を修正して

```
Grid[Table[If[m==0 && n==0,x\y,f[{n,m}]], {n,-1,len}, {m,-1,len}],
                                          Option]                   (f1)
```

としよう。\ は \[Backslash] と入力する。こうして Fig.7 が得られた(宮寺と福井[1]の図8の $x\backslash y$ はもっと見栄えが良い。グラフィックス機能を用いたとのことである)。

Out[92]=

x\y	0	1	2	3	4	5	6	7	8	9	10
0	0	1	2	3	4	5	6	7	8	9	10
1	1	2	0	4	5	3	7	8	6	10	11
2	2	0	1	5	3	4	8	6	7	11	9
3	3	4	5	6	2	0	1	9	10	12	8
4	4	5	3	2	7	6	9	0	1	8	13
5	5	3	4	0	6	8	10	1	2	7	12
6	6	7	8	1	9	10	3	4	5	13	0
7	7	8	6	9	0	1	4	5	3	14	15
8	8	6	7	10	1	2	5	3	4	15	16
9	9	10	11	12	8	7	13	14	15	16	17
10	10	11	9	8	13	12	0	15	16	17	14

Figure 7: Background を指定した Table

3.2 Modula-2

筆者は XDS Modula-2 を用いてCornerTheQueen.MOD を作った。メインのプロシージャーCornerTheQueen.MODの仕事はテキストウインドウ上でメニューを作ることである。実質的なコードは SubQueenGameというモジュールに記述してある。以下で要点を記す。集合 $G_{i,j}$ は SET で表現し、SubQueenGame.DEF で型の宣言をした。

```
TYPE
  grundyTp = ARRAY[0..boardSize-1] OF ARRAY[0..boardSize-1] OF INTEGER;
  iRange   = [0..maxMember];
  gSetTp   = SET OF iRange;
```

```
VAR
  Gr  :  grundyTp;
```

二次元配列grundyTp型の変数Grは$G_{i,j}$を表す。 maxMember は盤サイズの2倍とした。実際、$G_{i,j}$ の最大値はたかだか盤サイズの 1.8倍である。gSetTp は mex 関数の入力 Set を規定する型である。 Set には $G_{i,j}$ のダブリはない。

mex関数は、有限だが十分大きい[2]集合 Γ = wholeSetとSetの差集合、つまり補集合 complementary set の最小要素を返す。

```
PROCEDURE moveH(x,y: INTEGER): gSetTp;
  VAR
    xx: INTEGER;
    mySet : gSetTp;
  BEGIN
    mySet := gSetTp{};
    FOR xx:=0 TO x-1 DO
      INCL(mySet, Gr[xx,y]);
    END;
    RETURN mySet;
  END moveH;

PROCEDURE minComplementSet(Set: gSetTp;
VAR j: INTEGER);
  VAR
    gSet: gSetTp;
    i: iRange;
  BEGIN
    gSet := wholeSet - Set;
    i:= 0;
    LOOP
      IF i IN gSet THEN EXIT END;
      INC(i);
    END;
    j:= i;
  END minComplementSet;

PROCEDURE mex(Set: gSetTp): INTEGER;
  VAR
    i: INTEGER;
  BEGIN
    prepareWholeSet();
    minComplementSet(Set,i);
```

[2]盤の大きさをnとすると$G_{i,j}$の上限はおおむね $1.5n$である。

```
    RETURN i;
  END mex;

PROCEDURE calcGrundy(VAR Gr: grundyTp);
  VAR
    x,y,z,n,xx,yy,N: INTEGER;
    mySet : gSetTp;
  BEGIN
    N := boardSize;
    Gr[0,0]:= 0;
    FOR n:=1 TO N-1 DO
      FOR x:=0 TO n-1 DO
        mySet := moveH(x,n) + moveV(x,n) + moveT(x,n);
        Gr[x,n]:= mex(mySet);
      END;
      FOR y:=0 TO n-1 DO
        mySet := moveH(n,y) + moveV(n,y) + moveT(n,y);
        Gr[n,y]:= mex(mySet);
      END;
      mySet := moveH(n,n) + moveV(n,n) + moveT(n,n);
      Gr[n,n]:=mex(mySet);
    END;
  END calcGrundy;
```

`moveH(a,b)` は `(a,b)`から横に動いてGrundy数を読み込んでいく関数である。同様に `moveV(a,b)` と `moveT(a,b)` は `(a,b)`からそれぞれ縦と斜めに動いて読み込む関数である。読み込む順は図4の通りである。 `calcGrundy` で Grundy数を計算し、`Gr` という配列を出力する。`outGrundy` で `Gr` の出力、`listOfZeros` で値がゼロのマス位置の出力をそれぞれ行う。

Mathematica プログラムに比べて随分長いが、アルゴリズムそのものは分かりやすいのではないか。

boardSize=31 として$G_{i,j}$を計算してから部分的領域について表示をした。図8は宮寺と福井[1]の図8と内容は同じである。

3.3 Python

Pythonプログラムはメインとモジュール`CornerTheQueen`から成る。コンソール画面上に$G_{i,j}$を出力する。

Python の List は Modula-2 の SET と似ているが、要素がダブってもListに取り込められる。モジュール`CornerTheQueen` では `calcGrundy` というクラスを定義している。メソッド `outGr()` はGrundy値を表示し、`listOfZeros` はGrundy値をゼロとする座標を表示する。

```
class calcGrundy:
    def __init__(self, N):
```

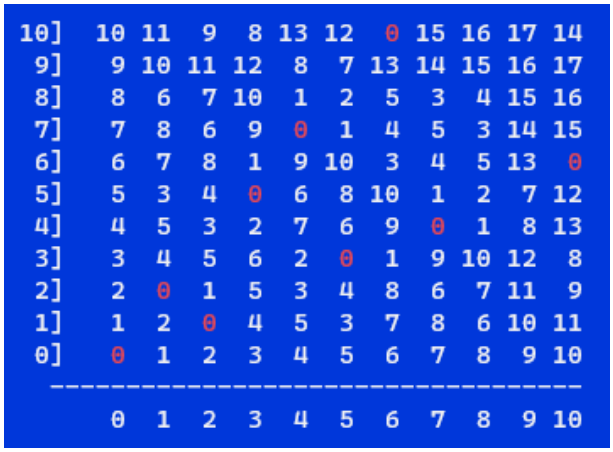

Figure 8: サイズ11のGrundy数

```
self.Gr = np.zeros((boardSize,boardSize),dtype=int)
self.N = N
def moveH(x,y):
    mySet = []
    for xx in range(x):
        mySet.append(self.Gr[xx,y])
    return  mySet
def moveV(x,y):
    mySet = []
    for yy in range(y):
        mySet.append(self.Gr[x,yy])
    return mySet
def moveT(x,y):
    mySet = []
    tn = min(x,y)
    for tt in range(tn):
        mySet.append(self.Gr[x-tn+tt,y-tn+tt])
    return mySet

for z in range(N):
    self.Gr[z,0] = z
    self.Gr[0,z] = z
for n in range(N-1):
    nn = n + 1      # nn=1..N-1
    for x in range(nn):
        xSet = sumLists(moveH(x,nn), moveV(x,nn), moveT(x,nn))
        self.Gr[x,nn] = mex(xSet)
    for y in range(nn):
        ySet = sumLists(moveH(nn,y), moveV(nn,y), moveT(nn,y))
        self.Gr[nn,y] = mex(ySet)
```

```
            tSet = sumLists(moveH(nn,nn), moveV(nn,nn), moveT(nn,nn))
            self.Gr[nn,nn] = mex(tSet)

###     outGr(self, isDn):
```

ここで mex 関数は次の通りである。なお $\gamma =$ maxMember である。

```
def mex(gList):
    wList = []
    for i in range(maxMember):
        wList.append(i)
    for p in gList:
        if wList.count(p)>0:
            wList.remove(p)
    k = 0
    while(True):
        if wList.count(k)>0:
            break
        k += 1
        if k>maxMember:
            k=-1
            break
    return k
```

次にmoveH 関数は次のように書ける。

```
        def moveH(x,y):
            mySet = []
            for xx in range(x):
                mySet.append(self.Gr[xx,y])
            return  mySet
```

図9はサイズ9の表である (VScode を利用。サイズ11ではコンソール画面表示が途中でスキップされた)。

4 Grundy数がゼロの位置

Corner The Queen ゲームに勝つためには$G_{i,j} = 0$の場所がどこかを知ってさえいればよいと述べた。そこでそのような場所、つまりゼロ点(i, j)をすみやかに得る方法を論じよう。

実はゼロ点の位置を求める方法は既に上で述べた。整理すると、 Mathematica については、§3.1 のの式(i) の zposition がゼロ点のリストであると述べた。

```
Grundy values for board size =  9
---------------------------------------
 8 ]   8  6  7 10  1  2  5  3  4
 7 ]   7  8  6  9  0  1  4  5  3
 6 ]   6  7  8  1  9 10  3  4  5
 5 ]   5  3  4  0  6  8 10  1  2
 4 ]   4  5  3  2  7  6  9  0  1
 3 ]   3  4  5  6  2  0  1  9 10
 2 ]   2  0  1  5  3  4  8  6  7
 1 ]   1  2  0  4  5  3  7  8  6
 0 ]   0  1  2  3  4  5  6  7  8
---------------------------------------
       0  1  2  3  4  5  6  7  8
```

Figure 9: サイズ9のGrundy数

Modula-2 については、ソースコードを省略したがモジュール `SubQueenGame` の中の `calcGrundy` の中でゼロ点を拾い出し、同じく`listOfzeros` で `zero` を定義している。 Python については、クラス`calcGrundy` の `outGr` でGrundy値を表示し、同じく`listOfZeros` でゼロ点の場所を出力する。このようにゼロ点の位置は解決済みのテーマであるといえるが、ここではパソコンゲームへの展開を視野に入れて簡便にデータを得ることを考える。

まずサイズが31の盤についてゼロ位置を図示したものが図10である。破線は直線 $y = ax$ であり、傾きは

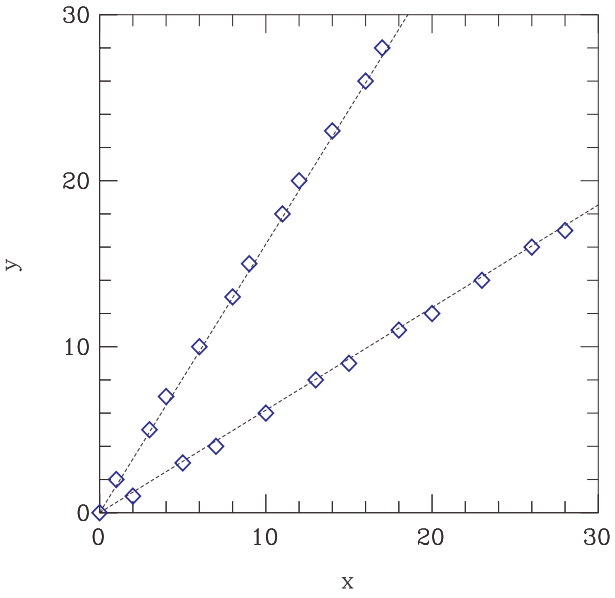

Figure 10: サイズ31のGrundy数ゼロの位置

$$a = \frac{\sqrt{5} \pm 1}{2}$$

である。この値はUCDavisのTAが作成した授業資料[3]に出ていた。ただし、この直線はいわゆる回帰直線ではない[3]。この a を用いるとGrundy数ゼロの場所が小さい順に計算できる[3] (§5.1 の関数(7))。Pythonでは次の関数で実現できる。なお `mirrorPos(i,j)` は i, j の入れ替えをする関数である。

```
def calcWinningPos(index):
     rindex = index // 2
     winj = int(a*float(rindex))
     wini = winj + rindex
     if (index % 2 == 0):
          wini,winj = mirrorPos(wini, winj)
     return wini,winj
```

5 パソコンゲーム

ゲームの対戦者がパソコンに変えれば碁盤ではなくコンピュータ画面を前にして戦う。

5.1 C言語

文献[3]に`CornerTheQueen.c` というプログラムが出ている (Oliver Kreylos, 1999)。このソースコードは MSYS2[2] の gcc でコンパイルできた。コンソール画面上で走らせたところ、確かにゲームとして成立していた。しかし、駒の位置を数値データ $(i,\ j)$ で入出力するのでヒューマンインターファースに問題があった。

1) $(i,\ j)$ を自分で把握しなければならない。例えばマス目を自分で用意しておくとか。

2) $(i,\ j)$ の入力ルールが厳しい。例えばコンマを入れると暴走する。

これらは今後の改良点として記憶に留めておくとして、ソースコードを分析しながめていこう。 `main()` の他に 11 の関数が使われている。コメントと共に関数の一覧を次に示す。

```
/******************************************************
(1) Display the main menu and ask for user's menu choice:
******************************************************/\\
int queryMainMenu(int boardSize)

/*******************************
(2) Ask user for the new board size:
*******************************/\\
int changeBoardSize(int boardSize)

/*********************************************************
(3) Let the user position the queen on a board of given size:
*********************************************************/\\
Position getInitialPosition(int boardSize)

/*******************************************************
(4) Check if the queen is located in the lower left corner:
*******************************************************/\\
int isLowerLeftCorner(Position p)

/************************
(5) Check if a move is legal:
************************/\\
int isMoveLegal(Position old,Position new)
```

[3]◇の真ん中あたりを貫く直線ではない。盤サイズが変われば傾きがわずかに変わるはずである。

```
/***********************************************
(6) Mirror a position about the board's main diagonal:
***********************************************/\\
Position mirrorPosition(Position current)

/***********************************************
(7) Calculate the winning position with a given index:
***********************************************/\\
Position calculateWinningPosition(int index)

/*******************************
(8) Test if two positions are equal:
*******************************/\\
int arePositionsEqual(Position pos1,Position pos2)

/*********************************
(9) Calculate a very intelligent move:
*********************************/\\
Position calculateComputerMove(Position current)

/********************
(10) Read the user's move:
********************/\\
Position getUserMove(Position current)

/*****************************************
(11) Play the game on a board of the current size:
*****************************************/\\
void playGame(int boardSize)

/*****************************************
(12) Main function
*****************************************/\\
int main(void)
```

この中で一番肝になるのが (9) の“Calculate a very intelligent move”である。この関数のPython流コードを次に示す。相手は (i, j) に置いた。もしそれが最強位置(winning position)であれば、(a)自分は最強位置に置けないのでひとつ移動するだけにとどめる。逆に (i, j) が最強位置でなければ、(b)今のところから行ける最強位置をみつける。

```
def calculateComputerMove(i, j):
     isFound = False
     index = 0
     while (not isFound):
          wi,wj = calcWinningPos(index)
          if (i, j)==(wi, wj):  # (a)
               if newi>=newj:
                      wi -= 1
               else:
                    wj -= 1
               isFound = True
          else:  #(b)
               if isMoveLegal(i,j,wi,wj):
                    isFound = True
          index += 1
     return wi,wj
```

5.2 Modula-2

C と Modula-2 は同じ imperative language のカテゴリーに属する言語であるから、Cで書かれたプログラムをModula-2に書き換えるのは容易である。そこで前項 (1)∼(11)の関数を名前と受け渡し方法を同じにして`main()` 以外をすべて`SubQueenGameK`というモジュールに収めた。こうして得られたプログラムはコンソ

ール画面を介して入出力を行う。よって C プログラムに伴う不便さはそのまま引き継がれている。

さて、筆者が愛用してきた XDS Modula-2/TopSpeed extension では、テキストウィンドウを介して入出力を行うことで先の欠点が改善されている (グラフィックスウィンドウも利用可能だが解像度は低い)。そこでチェス盤のインデックスを1ではなく 0 から始まるように書き換えて`SubQueenGameK0` とし、また上で述べた Grundy 数の計算モジュールを`SubQueenGame` とし、さらに両者に共通するタイプと変数を`QueenGameGlobal` で定義した。こうして`QueenTheCorner` の拡張版 Ver. 1.3 を作った。プログラムの構成を図式化したものが図11である。 MainMenu で Game of Corner The Queen ('G') を選択してゲームが始まる。`(x,y)` はセルを指定する座標であり、範囲は `0,..,N` である。`Display(x,y)` は画面をクリアしたあと$(N+1)\times(N+1)$ のマス目に文字を書き入れる。`Idle` によって外面表示を1秒程度維持する。実際の画面のようすは図12から把握できるであろう。

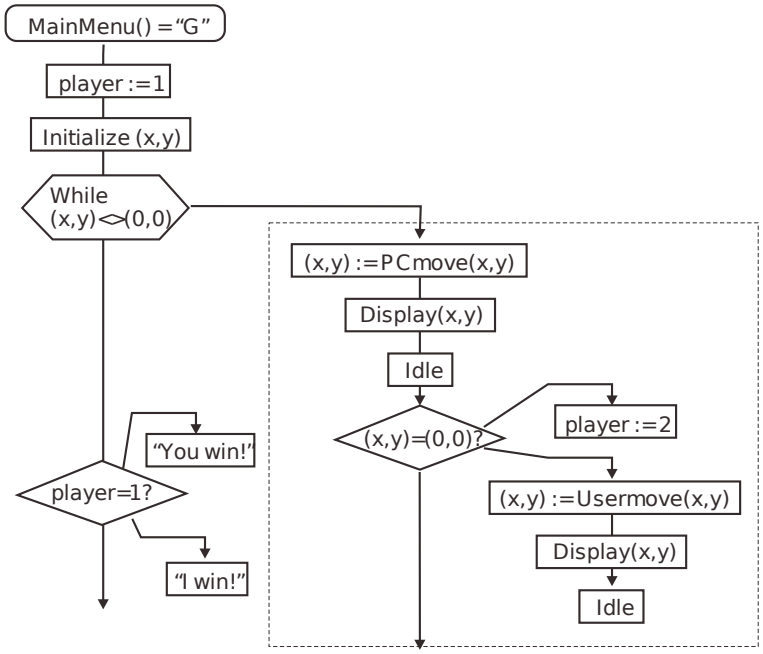

Figure 11: XDS Modula-2/TopSpeed extension でテキストウィンドウが利用できる

このプログラムには次のような長所がある。

1) マス目とQueen位置を "o" という文字で表現した (図12)。必ずしも正方形ではないが実用上は問題なかろう。盤サイズは11であるが20くらいまでなら対応可能である。それ以上になるとグラフィックス画面を使う必要がある。

2) 位置の整数入力 (図12の一番下) を、文字列入力・整数出力の関数 `InputIntVal` で行った。入力文字列が残っていればリターンキーを押だけで整数値が返される。プログラムが暴走することはない。

5.3 Python

最も単純なPython プログラムは、Cプログラム (§5.1) の場合と同様にコンソール画面で位置データのやりとりをする方式のものであろう。Modula-2プログラ

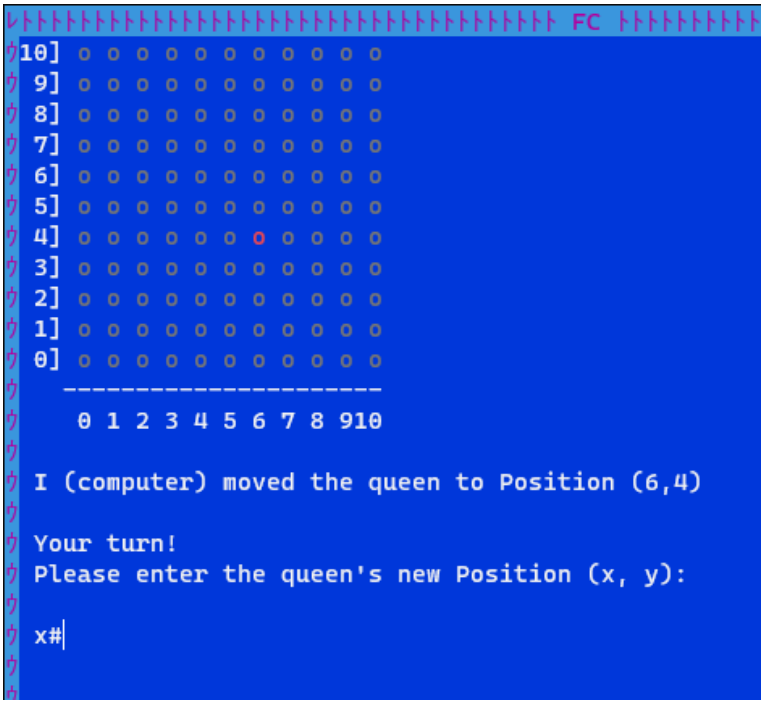

Figure 12: Kreylos のゲームアプリの XDS Modula-2版

ム (§5.2) の場合と同様に位置データを図式的に表示できれば状況の把握が楽になる。しかし Python ではさらに上を行くことができて、Queenの座標を上記のプログラムのように数値として入力するのではなく、マウスをクリックして場所を指示することができる。今回筆者は `pygame` と `pygwidgets` [4] を引用して`CornerTheQueenMain.py` と `CornerTheQueenMove.py` を書き上げた。前者の骨子はWindows画面上での無限ループ (図13) である。無限ループではまずマウスクリックを検出する。終了スイッチのクリック、再スタートスイッチのクリック、そして Queen の移動場所のクリックの三つがある。もしQueen を移動させた位置がルール通りであれば、次の場合分けを考える。

1) 移動させた位置が原点 (0, 0) であれば、音を鳴らせてユーザーの勝ちと表示し、Queen をそこに留めておく。

2) 原点以外に移動させた場合、パソコンが次の移動場所を計算する。

逆に移動させた位置がルール違反であれば、音を鳴らして元の場所に戻す。

図の二極スイッチの片方は、0.5秒のアイドリングである。これかないとユーザーが置いた場所から直ちにパソコンが別の位置に打ってしまうので、ユーザーは多分一人できょとんとした表情で打つことになるであろう。

図の二極スイッチのもう片方では、パソコンが次の移動場所を計算する。もし場所が原点 (0, 0) であれば音を鳴らして PC の勝ちと表示し、Queen をそこに留めておく。

もう一つのモジュール `CornerTheQueenMove.py` には、C関数のNo.4~No.9 に相当する関数が入っていて、パソコンが打つ手を計算する。

図14左はアプリを起動した直後の画面である。左下の隅 (0,0) がゴール位置、右上が待機位置である。ユーザーが格子上のどこかをクリックしてゲームが始ま

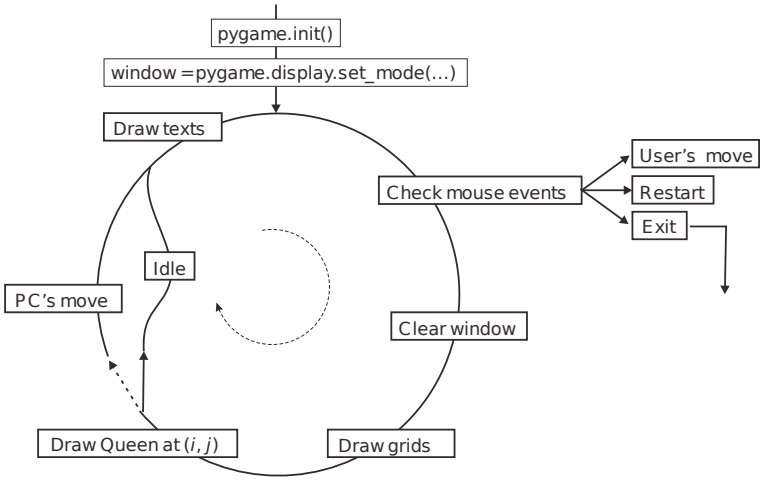

Figure 13: プログラムの無限ループ

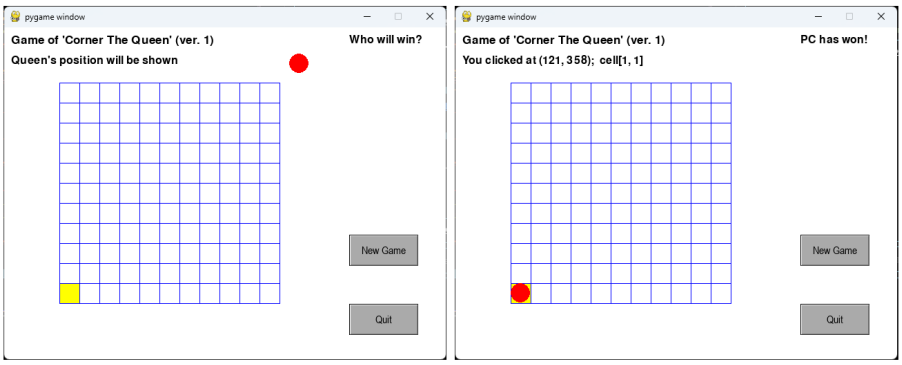

Figure 14: Kreylos のゲームアプリの Python/Windows版

る。ユーザーがクリックしたセルに●が移動したあと0.5秒待ってパソコンが計算した次の位置に●が移動する。もしルールに違反した場所をクリックすれば元の位置に戻される。図14右はパソコン側がゴールした時の画面である。New Game のテキストボックスをクリックすれば図14左の状態に戻り、Quit のテキストボックスをクリックすればプログラムが終了する。

6　おわりに

主題があちこちぶれていて、いうなれば随想的な感じのする話になったが、実はそれなりの理由がある。というのは、旧知のO氏から「高校生にMathematicaの話をしてはどうですか」と提案されたのがそもそものきかけである。大学の共通教育でMapleを扱っていたことがあるのでまんざら知らないわけでもないが、高校生となると果たして意味があるのか疑問がわく。

そこでネットで情報収集をしたところ宮寺と福井の論文に遭遇した[1]。それによると平成 30 年に公示された高等学校新学習指導要領において“理数探究”、“理数探究基礎” が設置されているとのことであった。この科目では生徒が自身で探究的に学んでいくことが求められており、発見学習と捉えることができる。ここにMathematica の入り込む余地がある。実際、この論文では“Corner The Queen”という数理的ゲームを題材としてMathematicaに触れさせている。そして Grundy数を求めるだけでなくグラフィックス表示も経験する (§3.1)。高校のカリキュラムの中でじっくり時間をかけていく内容である。

しかしO 氏のねらいは、このようなじっくり系の取り組みを推進することではなく、短期セミナーで多くの高校にMathematicaのすごさを実感してもらうことである。そのためには高校生がMathematicaや数式処理ということばを受け入れてくれるか、少なくとも違和感を感じないかを把握しておく必要がある。

そこで参考になるのが、教科書や指導要領の中にことばが使われているかということである。さすがにMathematica という登録商標は出てきないが、「数式処理ソフトウェア」ということばは高等学校情報の学習指導要領（平成30年告示の解説 p.52）に一箇所だけ登場するが、情報の教科書にも数学の教科書にも出てこない。これらのことから、一般の高校生にとってはMathematicaや数式処理ということばに馴染みがないと判断できる。彼らに「Mathematicaという素晴らしいソフトがあるから使ってみよう!」と話をもちかけて果たして手応えがあるだろうか?

そこで視点を変えて Corner The Queen ゲームのほうに力点を置いてみた。高校でも大学でもまず耳にしないGrundy数や mex関数の話はそれなりに興味を引くであろう。一見簡単であるが中身の濃い方程式(2)で解が得られることに私は感銘を受けた。さらに Mathematica であれば (Wolfram言語によれば) 式(2) をそのまま焼き直したような式(c)で解が得られることに驚いた。AI を先取りしたソフトの感がある。

しかし、答えを出すだけなら汎用言語でもできる。式(2)をパラフレーズすれば、つまり式に潜むインデックス間の関係性を考慮して解く順序を工夫すれば閉じた式で表される。そうなれば汎用計算機言語で処理できる。高校生を呼び込むのは楽になるであろうが O 氏の描いたビジョンからは逸脱する。思案のしどころである。

References

[1] 宮寺良平、福井昌則, “高校生が数式処理システムを用いて数学研究を行うための方法の提案”, 数式処理 **26**, No. 2, 3 – 23 (2020).

[2] https://www.msys2.org/

[3] https://web.cs.ucdavis.edu/ okreylos/TAship/Fall1999/CornerTheQueen.html

[4] I. Kalb, *Object-Oriented Python* (No Starch Press, 2022).