

Disentis

Dieses Spiel wurde in Disentis (Schweiz) erfunden, um eine lange Wartezeit zu verkürzen. Zerschneide dazu eine Postkarte in, sagen wir 5x5 Felder, und lege die einzelnen Teile verdeckt auf den Tisch. Das Ziel ist es, alle Teile herumzudrehen, um das ursprüngliche Bild wiederzusehen. Ein Zug besteht darin, ein noch verdecktes Teil auszuwählen und alle Teile darum herum, also bis zu acht, zu wenden. Bereit offenliegende Teile können dabei also wieder verdeckt werden! Das letzte verbleibende verdeckte Teil darf einzeln gewendet werden.

Trisentis

Nacheinander wenig Überlegung stellt man fest, dass die Bedingung, man müsse immer ein verdecktes Teil anwählen, im Wesentlichen die Reihenfolge der Züge (und vielleicht einigespäterrückgängigzumachende Züge) bestimmt. Lassen wir dies weg! Außerdem ist es nicht mehr notwendig, ein letztes Teil einzeln wenden zu können, und der Einfachheit halber verzichten wir auch auf diese Spezialregel. So erhalten wir das strukturell viele einfachere Spiel Trisentis. Erstes grundlegende Feststellung zu dieser Spielvariante ist folgende: **Die Reihenfolge der Spielzüge ist gleichgültig.**

Darstellung

Gegeben ist ein Spielfeld, das in kleinere Teile aufgeteilt ist. Dies können wir leicht als Matrix darstellen.

Zwei unterschiedliche Dinge sind von Interesse:

- Der Zustand des Spielfeldes. Stellen wir mit 0 (blau) ein offenes und mit 1 (gelb) ein verdecktes Teil dar, so können wir jeden Zustand notieren. Wir brauchen also hier Matrizen mit Einträgen in $\{0,1\}$.
- Die verwendeten Züge. Ein Abfolge von Zügen können wir auch in einer Matrix notieren, da die Reihenfolge der Züge irrelevant ist. Ferner hebt sich ein doppelt gespieltes Zug auf! Wenn wir eine 0 (grün) notieren für ein nicht gespieltes Feld und eine 1 (rot) für ein gespieltes Feld, so müssen wir $1+1=0$ rechnen, damit die Addition die Hintereinanderausführung von Zügen wiedergibt.

Wir werden also beide Dinge durch Matrizen mit Einträgen im Körper $GF(2)$ darstellen. Zur Ausgabe verwenden wir kleine bunte Quadrate für jeden Eintrag.

Über die Spielregeln hinaus werden wir uns die Freiheit lassen, mit beliebigen Anfangszuständen und Zielzuständen zu arbeiten.

- ```
reset():
GF2 := Dom::IntegerMod(2):
MatGF2 := Dom::SparseMatrix(GF2):
GF2::print := GF2::expr:
```
- ```
plotmatrix:= proc(M, c0, c1, s)
```

```

    local i, j, c, L, dim;
begin
    if args(0)<2 then c0:=RGB::Blue; c1:=RGB::Yellow; end_if;
    if args(0)<4 then s:=[0,0]; end_if;

    dim:= linalg::matdim(M);
    L:= table();

    for i from 1 to dim[1] do
        for j from 1 to dim[2] do
            if iszero(M[i,j]) then
                c:= c0
            else
                c:= c1
            end_if;
            L[i,j]:= plot::Rectangle2d([ j-1/2+s[1], dim[1]-i+1/2+s[2]],
                1, 1,
                Color = c, Filled = TRUE);

            if x<=6 and y<=6 then
                L[i,j]:=L[i,j],plot::Rectangle2d([ j-1/2+s[1], dim[1]-
                i+1/2+s[2]], 1, 1, Color = RGB::Black);
            end_if;
        end_for;
    end_for;

    plot::Scene(L[i,j] $ i = 1..dim[1] $ j = 1..dim[2], Axes = None,
        Scaling = Constrained);

```

```
end_proc:
```

- `plotsolution:=proc(L)`

```

begin
    P:=plotmatrix( L[1], RGB::Green, RGB::Red);
    for i from 1 to nops(L[2]) do
        P:=P,plotmatrix( L[2][i], RGB::Green, RGB::Orange, [i*(y+1),0]
        );
    end_for;
    plot::Scene(map(P,op),map(P,x->op(x::options)));
end_proc:

```
- `zugdelta:= proc(a,b)`

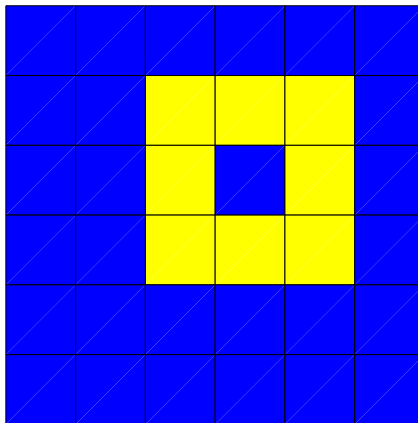
```

    local i,j,M;
begin
    M:= MatGF2(x,y,0);
    for i from max(a-1,1) to min(a+1,x) do
        for j from max(b-1,1) to min(b+1,y) do
            if (i<>a or j<>b)
            then
                M[i,j]:= 1;
            end_if;
        end_for;
    end_for;
    return(M);
end_proc:

```

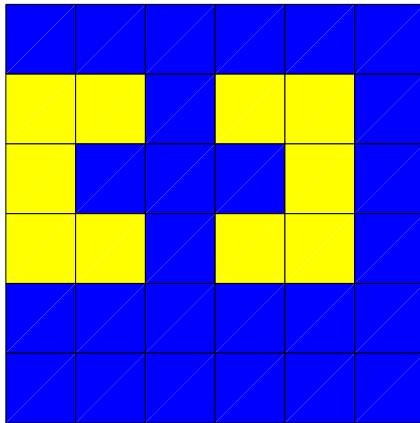
- `x:=6: y:=6:`
`zugdelta(3,4);`
`plot(plotmatrix(%));`

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



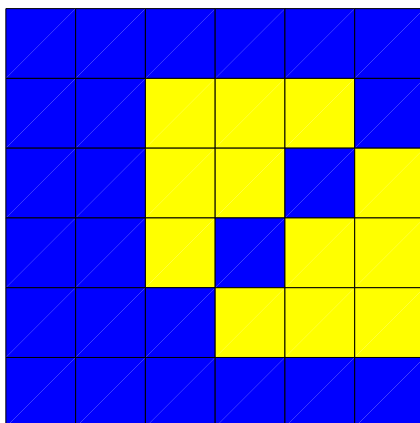
- `zugdelta(3,4) + zugdelta(3,2);`
`plot(plotmatrix(%));`

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



- `zugdelta(3,4) + zugdelta(4,5);`
`plot(plotmatrix(%));`

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



- ```

zug:=proc(Z)
 local i,j,S;
begin
 S:=MatGF2(x,y);
 for i from 1 to x do
 for j from 1 to y do
 if not iszero(Z[i,j]) then
 S:=S+zugdelta(i,j);
 end_if;
 end_for;
 end_for;
 S;
end_proc;

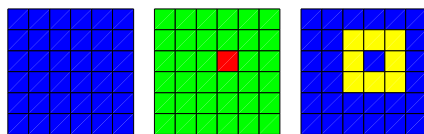
zeigezug:=proc(Z)
 local S,P;
begin
 S:=MatGF2(x,y);
 P:=plotmatrix(S),plotmatrix(Z,RGB::Green,RGB::Red,[y+1,0]);
 S:=S+zug(Z);
 P:=P,plotmatrix(S,RGB::Blue,RGB::Yellow,[2*(y+1),0]);
 plot(plot::Scene(map(P,op),map(P,x->op(x::options))));
end_proc;

```
- ```

Z:=MatGF2( x,y, (i,j)->if i=3 and j=4 then 1 else 0 end_if );
zeigezug( Z );

```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$




```

A:= linalg::concatMatrix(Matrix2Vector(zugdelta(i,j)) $ i = 1..x $
j = 1..y):
b:= Dom::SparseMatrix(Dom::IntegerMod(2))([1 $ x*y]):

L:= linalg::matlinsolve(A, b):
if domtype(L) <> DOM_LIST then L:=[L,[]]; end_if:
L[1]:= Vector2Matrix(L[1]):
L[2]:= map(L[2], Vector2Matrix):

print(Unquoted,"#Lösungen = 2^".nops(L[2])." = ".(2^nops(L[2])) );
plot( plotsolution( L ) );
L;
end_proc:

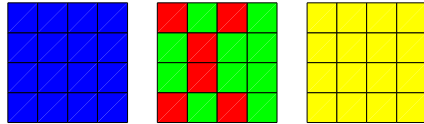
```

- L:=bruteforce(4,4):

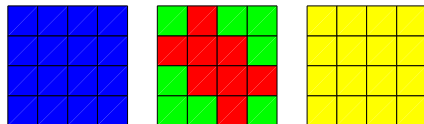
#Lösungen = $2^4 = 16$



- zeigezug(L[1]);



- `zeigezug(L[1]+L[2][3]);`



BeschleunigeLösungsmethode

- `loesefast:=proc(S)`
`local i, j, Zi;`
`begin`

```

Z:=MatGF2(op(linalg::matdim(S)));
for j from 1 to y-1 do
  for i from 1 to x-1 do
    if not iszero(S[i,j]) then
      S:=S+zugdelta(i+1,j+1);
      Z[i+1,j+1]:=Z[i+1,j+1]+1;
    end_if;
  end_for;
  if not iszero(S[x,j]) then
    S:=S+zugdelta(1,j+1);
    Z[1,j+1]:=Z[1,j+1]+1;
  end_if;
  for i from 1 to x-1 do
    if not iszero(S[i,j]) then
      S:=S+zugdelta(i+1,j+1);
      Z[i+1,j+1]:=Z[i+1,j+1]+1;
    end_if;
  end_for;
  if not iszero(S[x,j]) then
    S:=plotmatrix(S);
    plot( op(S), op(plotmatrix(Z,RGB::Green,RGB::Red,[y+1,0])),
          op(S::options) );
    error( "... ".j.". Spalte ist nicht lösbar." );
  end_if;
end_for;
return([S,Z]);
end_proc:

```

```

letzte:=proc(v)
  local S, Z, i, j;
begin
  if linalg::matdim(v)<>[x,1] then
    error("Falsche Dimension");
  end_if;
  S:=MatGF2(x,y);
  for i from 1 to x do
    if not iszero(v[i,1]) then
      S:=S+zugdelta(i,1);
    end_if;
  end_for;
  S:=loesefast(S);
  linalg::submatrix(S[1], 1..x, y..y ), S[2];
end_proc:

```

```

alleletzten:=proc()
  local A,L,k,a;
begin
  L:=[];
  A:=MatGF2(x,x);
  for k from 1 to x do
    a:=letzte( MatGF2([(0$(k-1)),1,(0$(x-k))])));
    A:=linalg::substitute(A,a[1],1,k);
    L:=[op(L),a[2]];
  end_for;

```

```

[A,L];
end_proc:

loesel:=proc( S )
  local A,Z,L,z,Z1,i;
begin
  S:=loesefast(S);
  Z:=S[2];
  S:=S[1];
  if not iszero(S) then
    A:=plotmatrix(S);
    plot( op(A),
          op(plotmatrix(Z,RGB::Green,RGB::Red, [y+1,0] )),
          op(A::options) );
    print(Unquoted,"loesefast genügt nicht! Brauchen
  alleletzten...");
  end_if;
  A:=alleletzten();
// plot(plotmatrix(A[1],RGB::White,RGB::Black));
  print(Unquoted,"Defekt alleletzten=".(linalg::matdim(A[1])[1]-
  linalg::rank(A[1])));
  L:=linalg::matlinsolve( A[1], linalg::submatrix( S, 1..x, y..y )
  );
  if L=[] then
    error( "... so nicht lösbar." );
  end_if;
  if domtype(L)<>DOM_LIST then L:=[L,[]]; end_if;

  z:=L[1];
  Z1:=Z;
  for i from 1 to x do
    if not iszero(z[i,1]) then
      Z1[i,1]:=Z1[i,1]+1;
      Z1:=Z1+A[2][i];
    end_if;
  end_for;
  L[1]:=Z1;
  S:=op([]);
  for z in L[2] do
    //z:=L[2][i];
    Z1:=Z;
    for i from 1 to x do
      if not iszero(z[i,1]) then
        Z1[i,1]:=Z1[i,1]+1;
        Z1:=Z1+A[2][i];
      end_if;
    end_for;
    S:=S,Z1;
  end_for;
  L:=[L[1],[S]];
end_proc:

loeselxy:=proc( a, b )
  local L;

```

```

begin
  if x mod 3=2 then
    print("Warnung: x mod 3=2, manche Spalten nicht lösbar!");
  end_if;
  // if x
  x:=a;
  y:=b;
  L:=loesel( MatGF2(x,y,1) );
  print(Unquoted,"#Lösungen = 2^".nops(L[2])." = ".(2^nops(L[2])) );
  plot(plotsolution( L ) );
  L;
end_proc:

```

Schnelle, verbesserte Lösungsmethode

- ```

loeseteilweise:=proc(S)
 local i, j, Z;
begin
 Z:=MatGF2(op(linalg::matdim(S)),0);
 for s from 2 to x+y-2 do
 for i from max(1,s-y+1) to min(x-1,s-1) do
 j:=s-i; // 1<= j <=y-1 erzwingt s-y+1<= i=s-j <=s-1.
 if not iszero(S[i,j]) then
 S:=S+zugdelta(i+1,j+1);
 Z[i+1,j+1]:=Z[i+1,j+1]+1;
 end_if;
 end_for;
 end_for;
 return([S,Z]);
end_proc:

startfeld:=proc(k)
 local i,j;
begin
 i:=1; j:=y+1-k;
 if j<1 then i:=2-j; j:=1; end_if;
 [i,j];
end_proc:

zielfeld:=proc(k)
 local i,j;
begin
 i:=k; j:=y;
 if i>x then i:=x; j:=x+y-k; end_if;
 [i,j];
end_proc:

randzurandltest:=proc(k)
 local S,Z;
begin
 S:=zugdelta(op(startfeld(k)));

```

```

P:=plotmatrix(S);
Z:=loeseteilweise(S);
S:=Z[1]; Z:=Z[2];
P:=P,plotmatrix(Z,RGB::Green,RGB::Red,[y+1,0]),
 plotmatrix(S,RGB::Blue,RGB::Yellow,[2*y+2,0]);
plot(plot::Scene(map(P,op),map(P,x->op(x::options))));
return([MatGF2([S[op(zielfeld(k))] $ k=1..x+y-1]),Z]);
end_proc:

randzurand1:=proc(k)
 local S,Z;
begin
 S:=zugdelta(op(startfeld(k)));
 Z:=loeseteilweise(S);
 S:=Z[1]; Z:=Z[2];
 return([MatGF2([S[op(zielfeld(k))] $ k=1..x+y-1]), Z]);
end_proc:

randzurand:=proc()
 local Z,S;
begin
 Z:=[randzurand1(k) $ k=1..x+y-1];
 [
 linalg::concatMatrix(op(map(Z, x->op(x,1)))),
 map(Z, op, 2)
];
end_proc:

loese2:=proc(S)
 local A,Z,L,z,Z1,i;
begin
 S:=loeseteilweise(S);
 Z:=S[2];
 S:=S[1];
 A:=randzurand();
 // plot(plotmatrix(A[1],RGB::White,RGB::Black));
 print(Unquoted,"Kerndimension(randzurand)=".(linalg::matdim(A[1])
 [1]-linalg::rank(A[1])));

 L:=linalg::matlinsolve(A[1], MatGF2([S[op(zielfeld(k))] $
 k=1..x+y-1]));
 if L=[] then
 error("... so nicht lösbar.");
 end_if;
 if domtype(L)<>DOM_LIST then L:=[L,[]]; end_if;

 z:=L[1];
 Z1:=Z;
 for i from 1 to x+y-1 do
 if not iszero(z[i,1]) then
 startfeld(i);
 Z1[op(%)]:=Z1[op(%)]+1;
 Z1:=Z1+A[2][i];
 end_if;
 end_for;
end_proc:

```

```

end_for;
L[1]:=Z1;
S:=op([]);
for z in L[2] do
 Z1:=Z;
 for i from 1 to x do
 if not iszero(z[i,1]) then
 startfeld(i);
 Z1[op(%)]:=Z1[op(%)]+1;
 Z1:=Z1+A[2][i];
 end_if;
 end_for;
 S:=S,Z1;
end_for;
L:=[L[1],[S]];
end_proc:

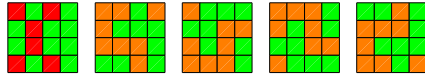
loese2xy:=proc(a, b, S)
 local L;
begin
 x:=a;
 y:=b;
 if args(0)>2 then
 L:=loese2(MatGF2(S));
 else
 L:=loese2(MatGF2(x,y,1));
 end_if;
 print(Unquoted,"#Lösungen = 2^".nops(L[2])." = ".(2^nops(L[2])));
 plot(plotsolution(L));
 L;
end_proc:

```

### Beispiele

- `st:=time();`  
`L4_b:=bruteforce( 4,4 );`  
`L4_btime:=(time()-st)*.001*sec;`

#Lösungen =  $2^4 = 16$

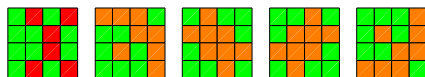


0.701 · sec

- `st:=time():`  
`L4_1:=loeselxy( x,y ):`  
`L4_1time:=(time()-st)*.001*sec;`

Defekt alleletzten=4

#Lösungen =  $2^4 = 16$

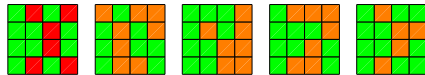


0.541 · sec

- `st:=time():`  
`L4_2:=loese2xy( x,y ):`  
`L4_2time:=(time()-st)*.001*sec;`

Kerndimension(randzurand)=4

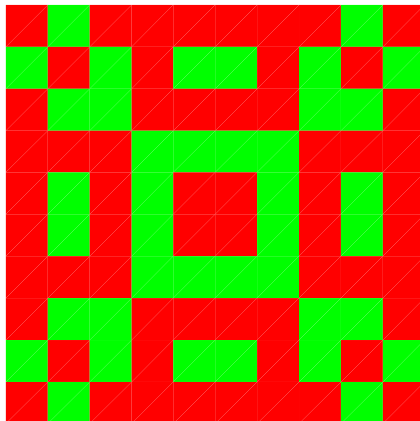
#Lösungen =  $2^4 = 16$



0.541 · sec

- `st:=time():`  
`L10_b:=bruteforce( 10,10 ):`  
`L10_btime:=(time()-st)*.001*sec;`

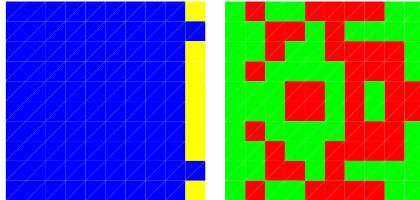
#Lösungen =  $2^0 = 1$



5.838 · sec

- `st:=time():`

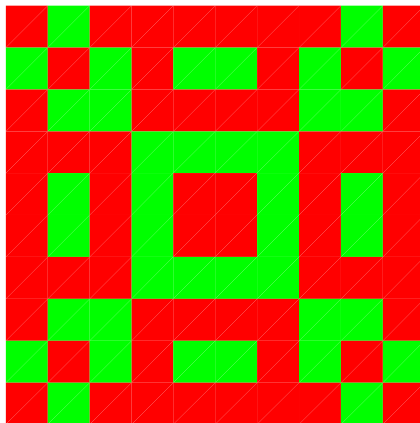
```
L10_1:=loeselxy(x,y):
L10_1time:=(time()-st)*.001*sec;
```



alleletzten... loesefast genügt nicht! Brauchen

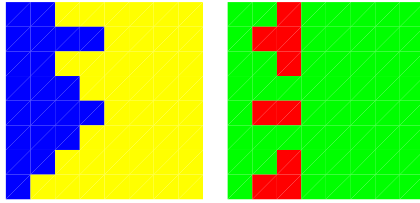
Defekt alleletzten=0

#Lösungen =  $2^0 = 1$



4.537 · sec

- ```
st:=time():
L8_1:=loeselxy( 8,8 ):
L8_1time:=(time()-st)*.001*sec;
```

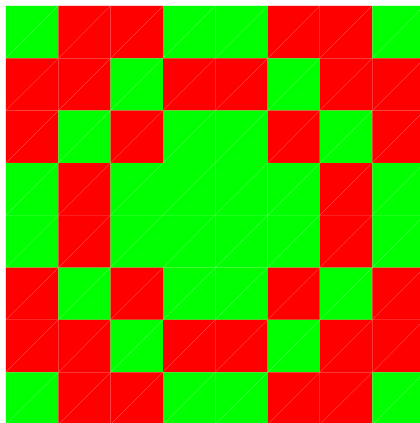


Error: ... 2. Spalte ist nicht lösbar. [loesefast]

- `st:=time():`
`L8_2:=loese2xy(x,y):`
`L8_2time:=(time()-st)*.001*sec;`

`Kerndimension(randzurand)=0`

`#Lösungen = $2^0 = 1$`

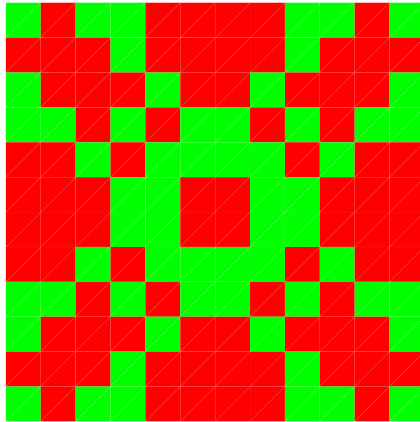


1.452 · sec

- `st:=time():`
`L12_2:=loese2xy(12,12):`
`L12_2time:=(time()-st)*.001*sec;`

`Kerndimension(randzurand)=0`

#Lösungen = $2^0 = 1$

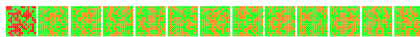


5.798 · sec

- `st:=time():`
`L14_2:=loese2xy(14,14):`
`L14_2time:=(time()-st)*.001*sec;`

Kerndimension(randzurand)=12

#Lösungen = $2^{12} = 4096$



20.249 · sec

- `st:=time():`

```
L16_2:=loese2xy( 16,16 ):
L16_2time:=(time()-st)*.001*sec;
```

Kerndimension(randzurand)=16

#Lösungen = 2^{16} = 65536



38.966 · sec

NocheineLösungfür6x6-Disentis(ohneReihenfolge!)

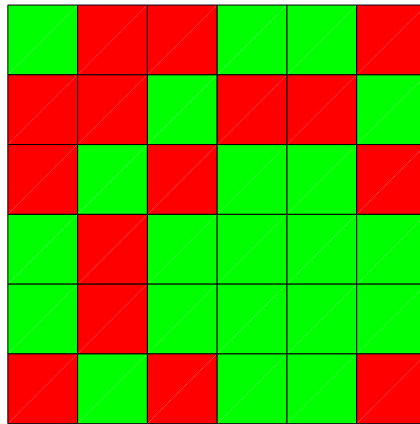
- ```
bruteforceb:=proc(B)
 local A,b,L;
 begin
 x:=linalg::matdim(B)[1];
 y:=linalg::matdim(B)[2];

 A:= linalg::concatMatrix(Matrix2Vector(zugdelta(i,j)) $ i = 1..x $
 j = 1..y):
 b:= Matrix2Vector(B):

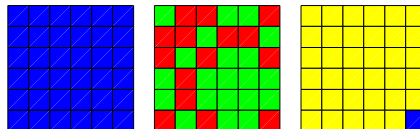
 L:= linalg::matlinsolve(A, b):
 if domtype(L) <> DOM_LIST then L:=[L,[]]; end_if:
 L[1]:= Vector2Matrix(L[1]):
 L[2]:= map(L[2], Vector2Matrix):

 print(Unquoted,"#Lösungen = 2^".nops(L[2])." = ".(2^nops(L[2])));
 plot(plotsolution(L));
 L;
 end_proc:
bruteforceb(MatGF2(6,6,(i,j)->if i=6 and j=6 then 0 else 1 end_if)
);
zeigezug(%[1]);
```

#Lösungen =  $2^0 = 1$



$$\left[ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \square \right]$$



•