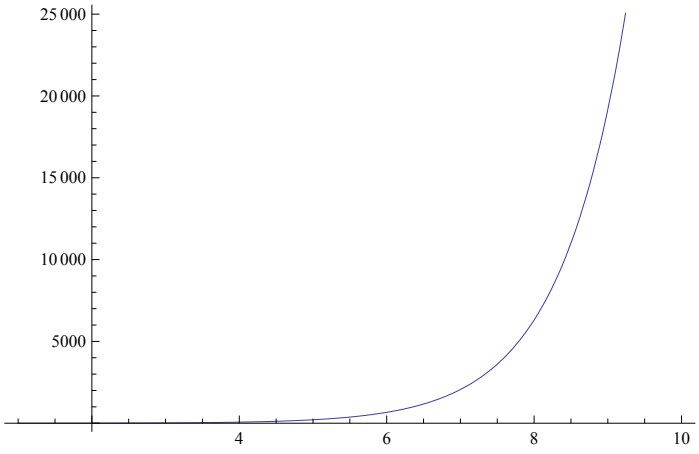
```
(* Problem 1 *)
Clear["`*"];
(*This command on line 1 is very
 useful. Clears all symbols from previous runs.*)
iv1 = a[1] == 1;
(* iv1 is the initial value. Note == 2 equality signs!*)
(* rr is the recurrence relation (RR). Note == 2 equality signs! *)
rr = a[n + 1] == 3 a[n] + (2^n);
(* In this example the RR is first order and non-homogeneous*)
sol = RSolve[{rr, iv1}, a[n], n] // Simplify
(* RSolve solves the RR.
   Do you wonder what // and Simplify means? Select them
  with mouse and click on "Find Selected Function" in Help *)
a[n_] = a[n] /. sol[[1]];
(* The last line looks strange. /.
 means Replace All. The function a[n] is defined from the solution sol in
  this way. sol is a list and [[1]] picks out the first
  element in this list (there is only one element here).*)
(* If you want to run the program then press Shift +
 Enter at the end of the last line *)
Print[a[2], "   ", a[3], "   ", a[4], "   ", a[5]]
Plot[a[n], {n, 1, 10}]
```

Out[4]= $\{\{a[n] \to -2^n + 3^n\}\}$

```
5   19   65   211
```

Out[7]=



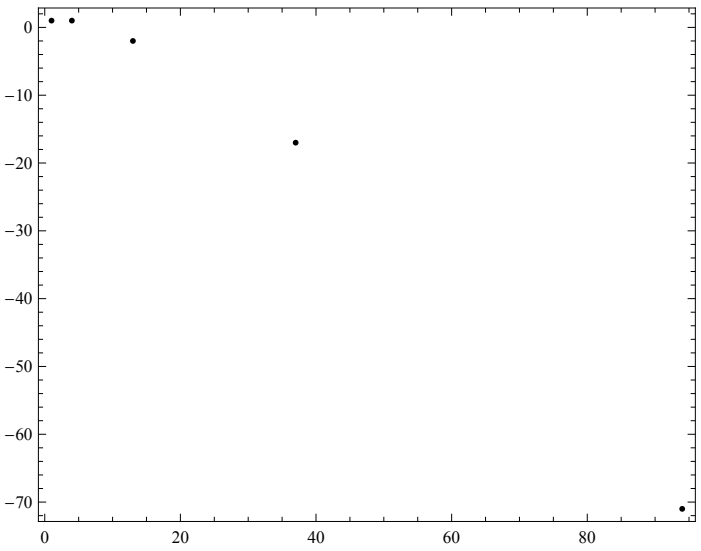
```
(* Problem 1 continued *)
Clear["`*"];
(*This command on line 1 is very
 useful. Clears all symbols from previous runs.*)
iva = a[0] == 1;
ivb = b[0] == 1;
(* iva and ivb are the initial values for a and b. Note ==. 2
  equality signs and then press space and they go together. *)
(* rr1 and rr2 is the system of recurrence relations (RR). Note ==  *)
rr1 = a[n + 1] == 3 a[n] + b[n];
rr2 = b[n + 1] == -a[n] + 2 b[n];
(* This example is a system of 2 first order RR*)
sol = RSolve[{rr1, rr2, iva, ivb}, {a[n], b[n]}, n] // Simplify
(* RSolve solves the RR.
   Do you wonder what // and Simplify means? Select them
  with mouse and click on "Find Selected Function" in Help *)
(* To solve this system Mathematica (and we humans) has to use
 complex numbers. The answer must of course be real at the end.
  N[Re[...]] are used below to get the numerical value of the real part
 of a[n] and b[n]. Numerically the imaginary parts are incredible small*)
a[n_] = a[n] /. sol[[1]];
b[n_] = b[n] /. sol[[1]];
(* The last line looks strange. /. means Replace All. The
  functions a[n] and b[n] are defined from the solution sol in
  this way. sol is a list and [[1]] picks out
  the first element in this list .*)
(* If you want to run the program then press Shift +
 Enter at the end of the last line *)
Print[N[Re[a[1]]], "   ", N[Re[a[2]]], "   ", N[Re[a[3]]], "   ", N[Re[a[4]]]]
Print[N[Re[b[1]]], "   ", N[Re[b[2]]], "   ", N[Re[b[3]]], "   ", N[Re[b[4]]]]
(* In this way we can plot the points jumping in the (a,b)-plane *)
data = Table[{a[n], b[n]}, {n, 0, 4}];
plot1 = Graphics[Point[data], Frame → True]
```

Out[37]=
$$\left\{\left\{a[n] \to \frac{2^{-1-n}\left(\left(5+i\sqrt{3}\right)^{n}\left(-3\,i+\sqrt{3}\right)+\left(5-i\sqrt{3}\right)^{n}\left(3\,i+\sqrt{3}\right)\right)}{\sqrt{3}},\right.\right.$$
$$\left.\left. b[n] \to \frac{2^{-1-n}\left(\left(5-i\sqrt{3}\right)^{n}\left(-3\,i+\sqrt{3}\right)+\left(5+i\sqrt{3}\right)^{n}\left(3\,i+\sqrt{3}\right)\right)}{\sqrt{3}}\right\}\right\}$$

```
4.   13.   37.   94.

1.   -2.   -17.   -71.
```

Out[43]=



In[61]:=

```
(* Problem 2 *)
Clear["`*"]
(* This program solves the logistic map. It is a non-linear problem,
note -x^2 on next line, and RSolve can not be used.
  Instead Nest and NestList are used. You don't have to go into detalis
 of the commands, just change the a-values and other input values.
   Some will test other functions and then you have to change g[x] *)
a =.; (* another way to clear parameter a. 0<a<
 4. The really interesting region is between 3 and 4*)
g[x_] := a x (1 - x);
(* x_(n+1)=g(x_(n)), parameter a is in example below 2.5. x_0=
 0.05=start and we do 30 iterations in this example*)
solg1 = Solve[x == g[x], x] // Simplify
solg2 = Solve[x == g[g[x]], x] // Simplify
(* but first we find fix points of g with solg1. Especially the
  second element in the list is interesting. It is the only fix point
  which is different from zero. It is created at a=1 when x=
  0 fix point becomes unstable. solg2 gives the period 2 solutions. That is
     the fix points of g(g(x)). Here we are interested of the genuine period
     2 solution which are the third and fourth elements in the list
     At http://en.wikipedia.org/wiki/Logistic_map you can read about
     this map. There you see that the period 2 solution was created at a=3
    when the fix point to the right became unstable.*)
(* Here you give values for the iteration*) a = 2.5; start = 0.05; iter = 30;
xfp = x /. solg1[[2]]
x2c1 = x /. solg2[[3]]
x2c2 = x /. solg2[[4]]
(* Last lines give numerical values for fix point and period 2 solution *)
(* The following lines you don't have
 think about. It produces an illuminating plot
 of the iterations *)
points = NestList[g, start, iter - 1]
lines[x_] := Line[{{x, x}, {x, g[x]}, {g[x], g[x]}}]
listlines = lines /@ points;
Plot[ g[x], {x, 0, 1},
 AspectRatio → Automatic,
```

```
 PlotLabel -> "a=2.5",
 PlotRange → {{0, 1}, {0, 1}},
 Epilog → {Line[{{0, 0}, {1, 1}}], listlines}]
(* Below follows commands that generate the
  bifurcation diagram in the interesting region 3.5<a<3.6.  *)
nstart = 500; number = 128;
(* It works like this I
  think: after 500 iterations the program takes a look at the following 128
    iterations. Union takes away doublets etc. Left
    is the stable periodic orbit. In the region from
    a = 3.5 to 3.6 period doublings take place. For  higher a-
  values there is a lot of chaos,
that is irregular orbits. No stable periodic orbit at all. But in some places
 there are "windows". A sign of a short stable periodic orbit! *)
afixpoint :=
  Union[
   Map[
    {a, #} &,
    NestList[g, Nest[g, start, nstart], number - 1]
   ]
  ];
amin = 3.5; amax = 3.6; step = 0.0001;
fixpoints = Flatten[Table[afixpoint, {a, amin, amax, step}], 1];
ListPlot[ fixpoints,
 PlotLabel -> "fix points and cycles",
 PlotRange → {{3.5, 3.6}, {0, 1}},
 PlotStyle → PointSize[0.001]]
```

Out[63]= $\left\{\{x \to 0\}, \left\{x \to \frac{-1+a}{a}\right\}\right\}$

Out[64]= $\left\{\{x \to 0\}, \left\{x \to \frac{-1+a}{a}\right\}, \left\{x \to \frac{1+a-\sqrt{-3-2a+a^2}}{2a}\right\}, \left\{x \to \frac{1+a+\sqrt{-3-2a+a^2}}{2a}\right\}\right\}$

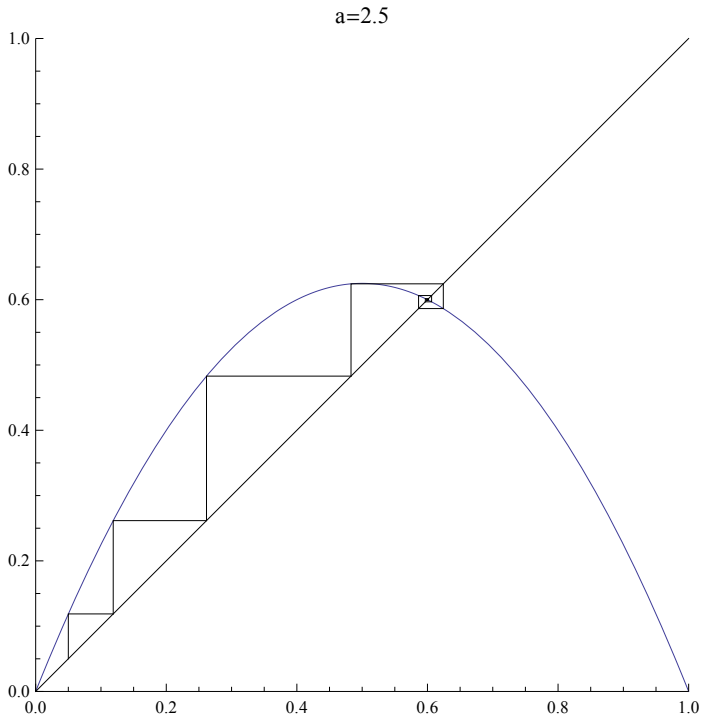
Out[66]= 0.6

Out[67]= 0.7 - 0.264575 i

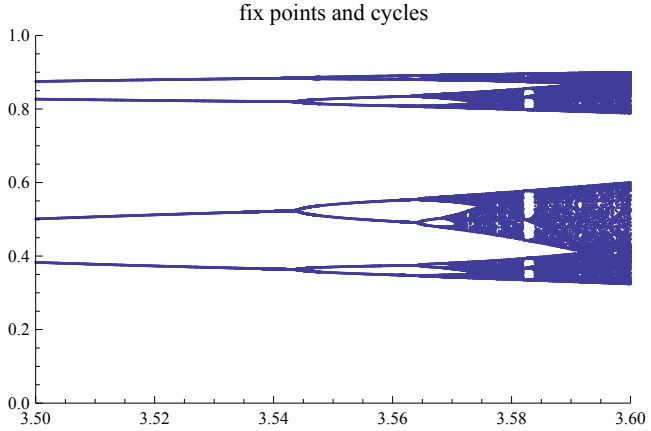
Out[68]= 0.7 + 0.264575 i

Out[69]= {0.05, 0.11875, 0.261621, 0.482939, 0.624272, 0.586391,
 0.606341, 0.596729, 0.601609, 0.599189, 0.600404, 0.599798,
 0.600101, 0.599949, 0.600025, 0.599987, 0.600006, 0.599997,
 0.600002, 0.599999, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6}

Out[72]=

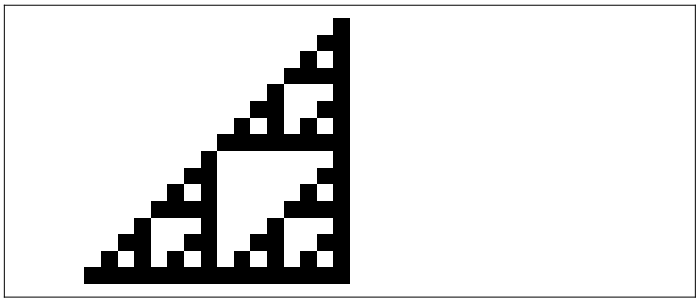

a=2.5
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Out[77]=


fix points and cycles
1.0
0.8
0.6
0.4
0.2
0.0
3.50
3.52
3.54
3.56
3.58
3.60

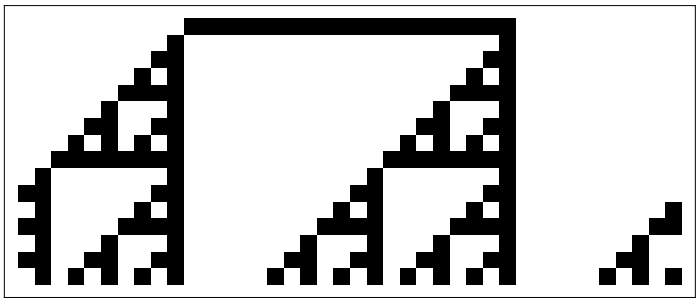
```
(* Problem 3. This example is for rule 102*)
Clear["`*"]
(* This is one of the 256 cellular automata which is one dimensional,
has two states (black or white) and next state
   in a cell depends only on previous state of itself and
   the 2 nearlying neighbours. Think of a long chain of houses.
    Each person in there can be happy or sad. The state next
   day depends on your mood the day before and on the mood
   of the people next-doors. There are 2^8=256 different rules for
   what can happen. Most of them are boring but especially rule 30
   and 110 are very interesting. See http://
  en.wikipedia.org/wiki/Cellular_automaton *)
dim = 40;
dim2 = dim / 2;
seed1 = Table[0, {dim}];
seed1[[dim2]] = 1;
(* seed1 is the start state. We have 40 cells in a row
 and all are white (0) execpt for the one in the middle (1) .
  The array plot below shows the evolution for
 15 generations. Time is downwards.*)
ArrayPlot[CellularAutomaton[102, seed1, 15]]
(* CellularAutomaton is thus a command in Mathematica *)
(* If you want to see Rule 102 -
 The Movie you use instead: Animate[ArrayPlot[CellularAutomaton[30,seed1,n]],
      {n,0,100,1},AnimationRunning→False].
     One black cell to start with.  100 iterations with step 1.*)
(* The new seed *)
seed2 = seed1;
seed2[[11 ;; 30]] = 1
ArrayPlot[CellularAutomaton[102, seed2, 15]]
Animate[ArrayPlot[CellularAutomaton[102, seed1, n]],
 {n, 0, 100, 1}, AnimationRunning → False]
```

Out[119]=



Out[121]= 1

Out[122]=



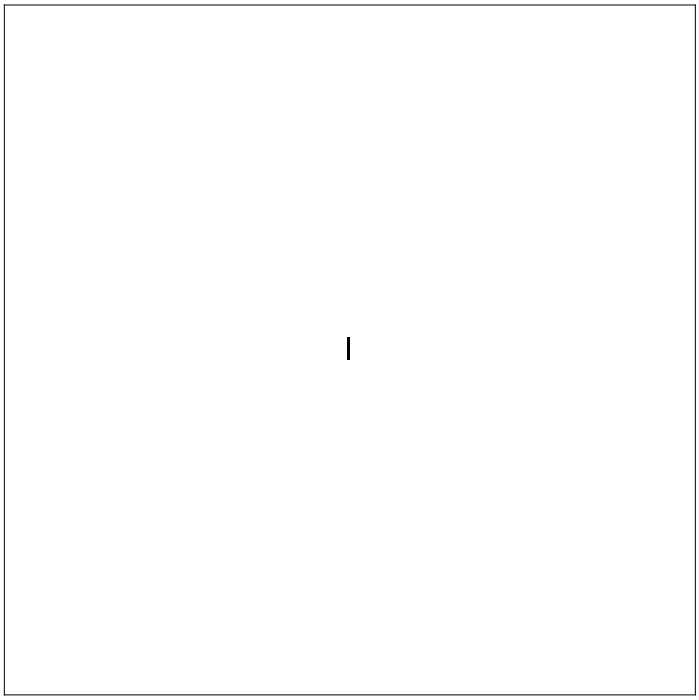
Out[123]=

n

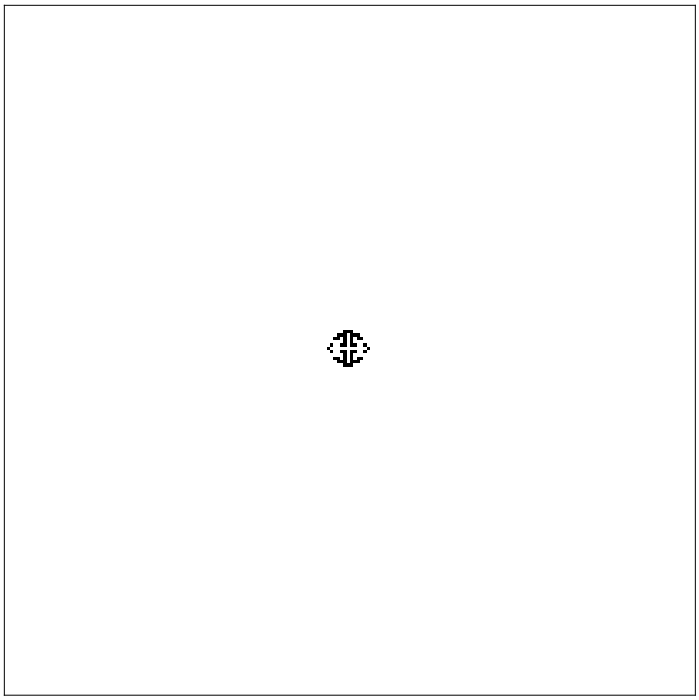
```
ArrayPlot[CellularAutomaton[102, seed1, 41]]
```

In[162]:=
```
(*Problem 4*)
Clear["`*"]
(* After 1D CA we turn to two dimensional
  CA. Still each cell can have two states. BUT
  there are now 8 neighbours. 3 above and below and one
  to the right and one to the left. So including the
  middle cell there are 2^9=512 states for these
  cells. Can you tell how many CA rules we then have?
   The most famous of these is called Game of Life,
http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
   and http://mathworld.wolfram.com/GameofLife.html *)
(* This example is for rule 746. I call it circular growth. The
  birth rule (white->black) is exactly 3 living neighbours.
   Survival (black→black) if no more than 4 neighbours are
  alive. Death ( black→white) if there are 5-8 living neighbours
  in the previous generation. Game of Life is rule 224. Same birth
  condition as for Circular Growth but death happends due to lonliness
  (0 or 1 living cell next to the cell) or over population
  (4-8 living cells around the middle cell). As in our world!
  If you write 746 and 224 in base 2 I hope you can
  figure out how this rule numbers are obtained.*)
CircularGrowth = {746, {2, {{2, 2, 2}, {2, 1, 2}, {2, 2, 2}}}, {1, 1}};
(* This list above specifies the 2D CA*)
seed = Table[0, {200}, {200}];
seed[[97, 100]] = 1;
seed[[98, 100]] = 1;
seed[[99, 100]] = 1;
seed[[100, 100]] = 1;
seed[[101, 100]] = 1;
seed[[102, 100]] = 1;
seed[[103, 100]] = 1;
(* seed is the input matrix. Here we start with a "worm",
or better seed, with 7 cells in a row. We use a 200 times 200 grid.
   200 iterations. In fact something shocking happends after 3000 iterations
  for a similar seed BBBWB. To see it you have to have a big matrix and
  it takens around 25 minutes one my machine. For five healthy
  cells in a row BBBBB this effect has not been seen. It seems like
  radius of ball is =C * (number of iterations). Can you estimate C?*)
ArrayPlot[CellularAutomaton[CircularGrowth, seed, {{{0}}}]]
ArrayPlot[CellularAutomaton[CircularGrowth, seed, {{{10}}}]]
ArrayPlot[CellularAutomaton[CircularGrowth, seed, {{{20}}}]]
ArrayPlot[CellularAutomaton[CircularGrowth, seed, {{{30}}}]]
ArrayPlot[CellularAutomaton[CircularGrowth, seed, {{{40}}}]]
ArrayPlot[CellularAutomaton[CircularGrowth, seed, {{{50}}}]]
ArrayPlot[CellularAutomaton[CircularGrowth, seed, {{{60}}}]]
ArrayPlot[CellularAutomaton[CircularGrowth, seed, {{{200}}}]]
(* If you want to see Circular Growth - The Movie you use instead:*)
Animate[ArrayPlot[CellularAutomaton[CircularGrowth, seed, {{{n}}}],
  Mesh → False], {n, 0, 200, 1}, AnimationRate → 5, AnimationRunning → False]
```

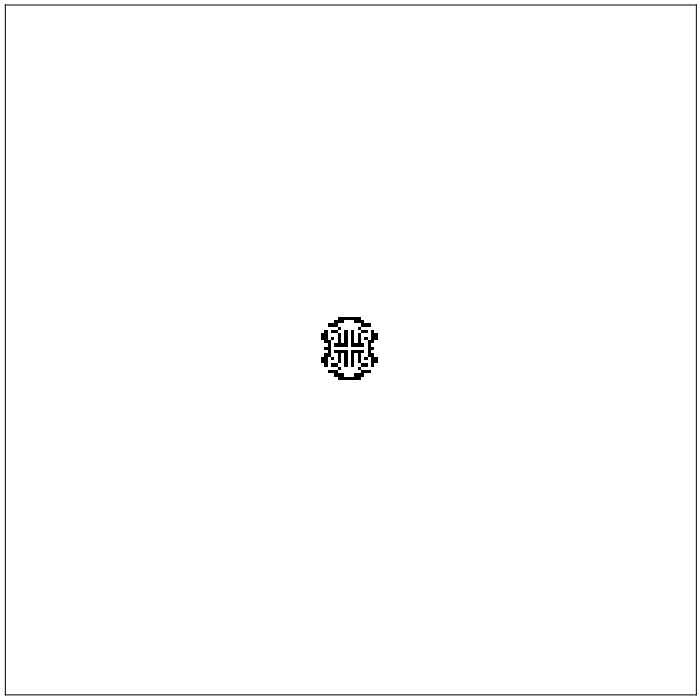
Out[172]=



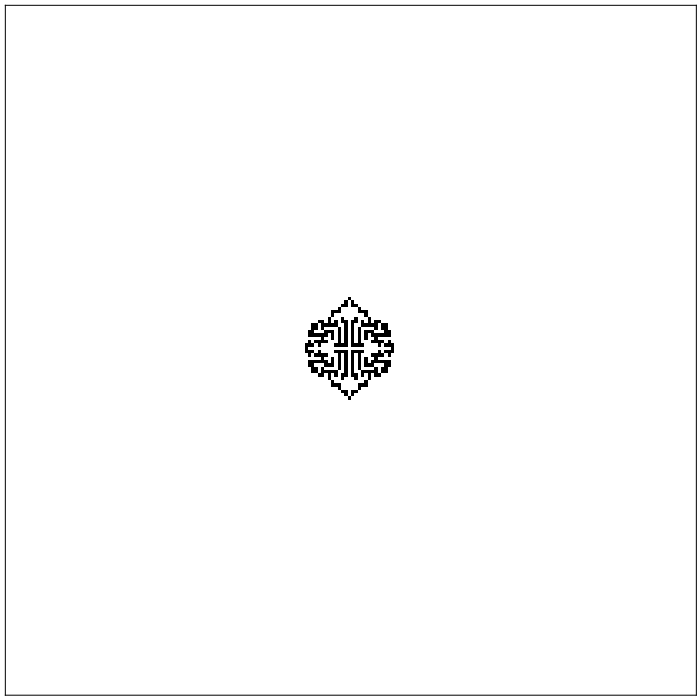
Out[173]=



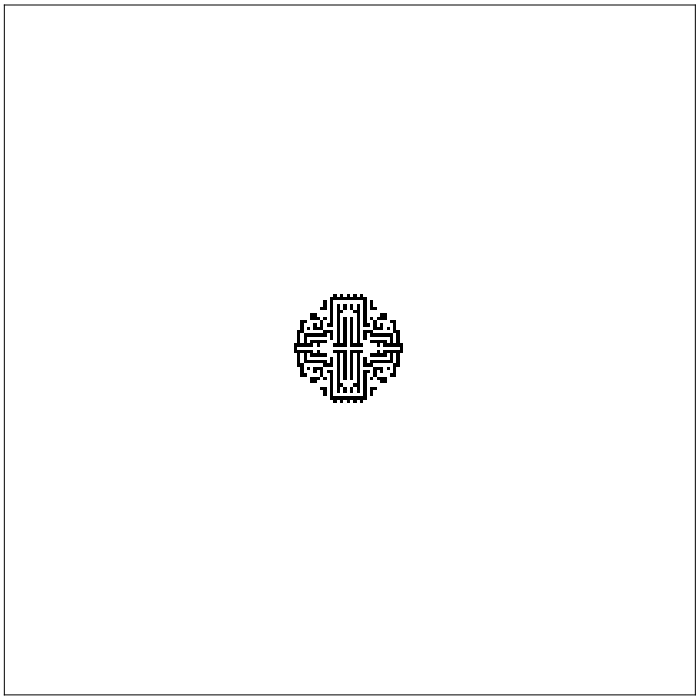
Out[174]=



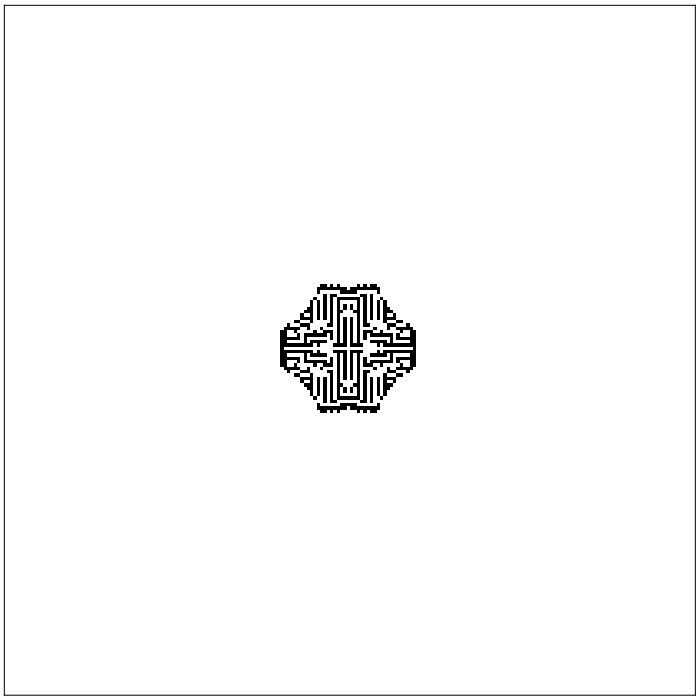
Out[175]=



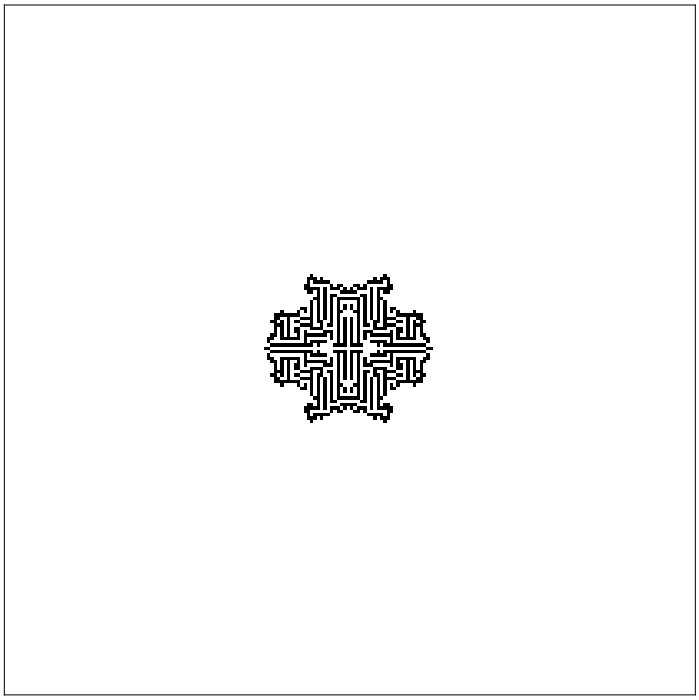
Out[176]=



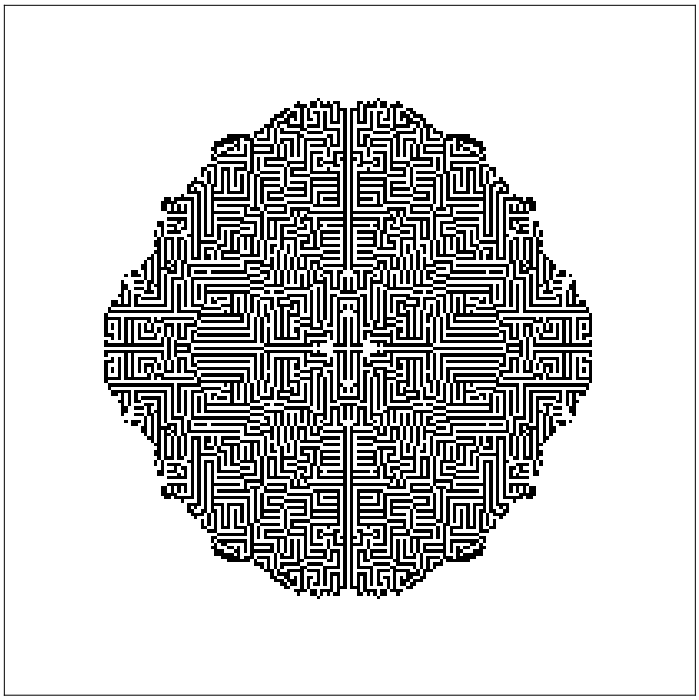
Out[177]=



Out[178]=



Out[179]=



n

Out[180]=