Applying NKS

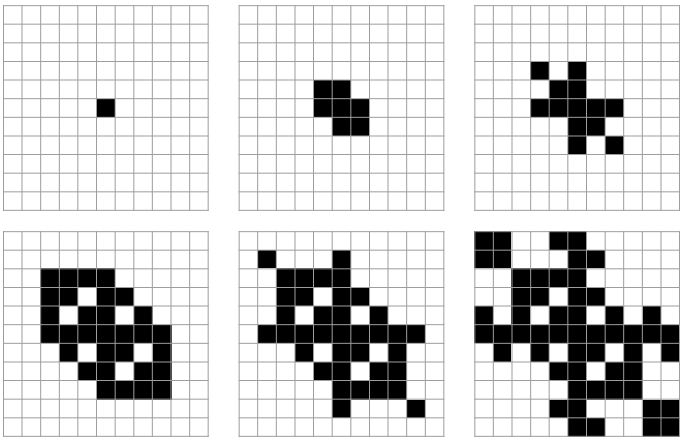
Simple rules will do something useful or capture an important property, often in surprising and unexpected ways.

It is unlikely that a simple rule will do everything you expect or hope for.

Don't forget the simplest rules. Persevere.

Make sure you have the proper visualization tools before you begin.

Snowflakes [p.369]

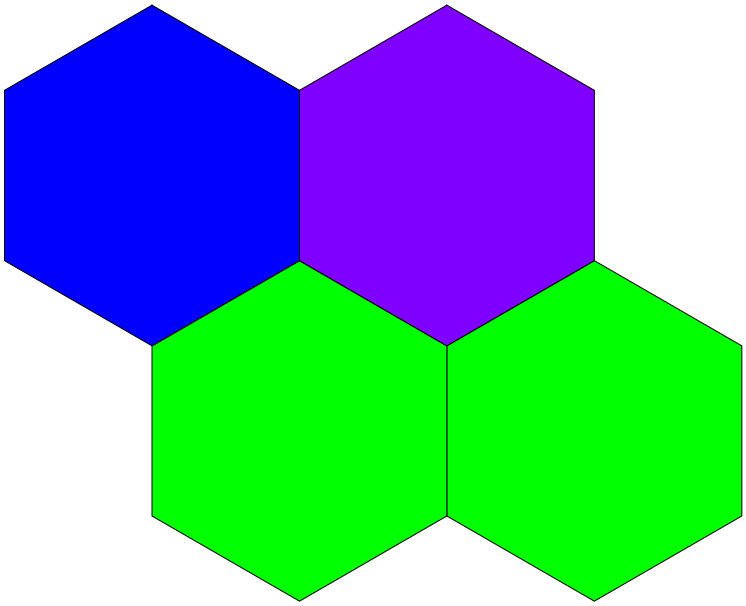


Code for better diagrams of hexagonal CA's

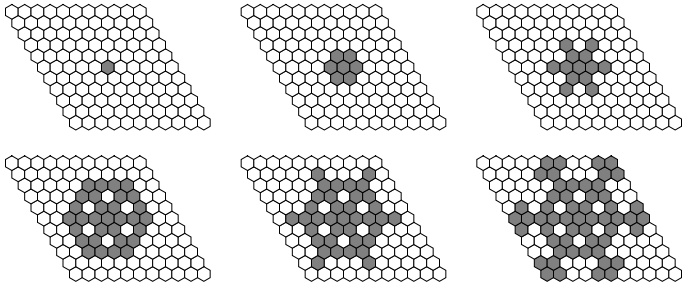
```
Directions6 = Table[{Sin[2 Pi i / 6], Cos[2 Pi i / 6]}, {i, 0, 6}] / Sqrt[3];

hex0[{i_, j_}, col_] := With[{pt = {i - j / 2, Sqrt[3] j / 2}},
  {col, Polygon[#], GrayLevel[0], Line[Append[#, First[#]]]} &[
   Map[# + pt &, Directions6]]]
```

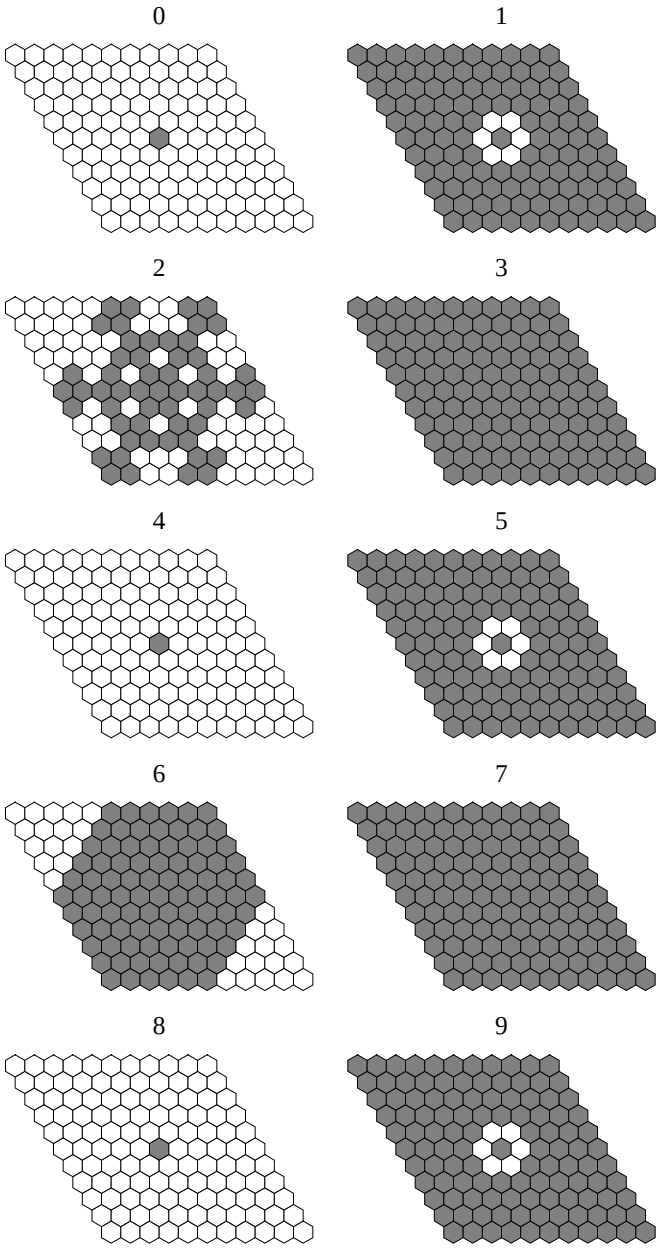
Sanity-check the code!



```
hexPlot[arr_, k_, opts___] :=
 Graphics[MapIndexed[hex0[#2, GrayLevel[1 - # / k]] &, arr, {2}], opts]
```

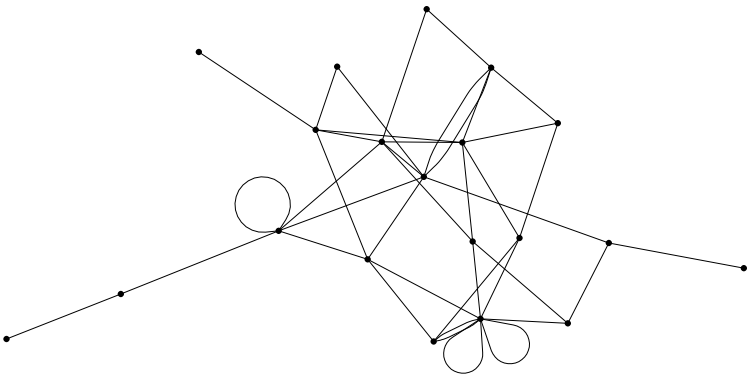


```
AggregateHexRuleNumber[rn_] := rn + 2^7 (2^7 - 1)
```



Coloring graphs

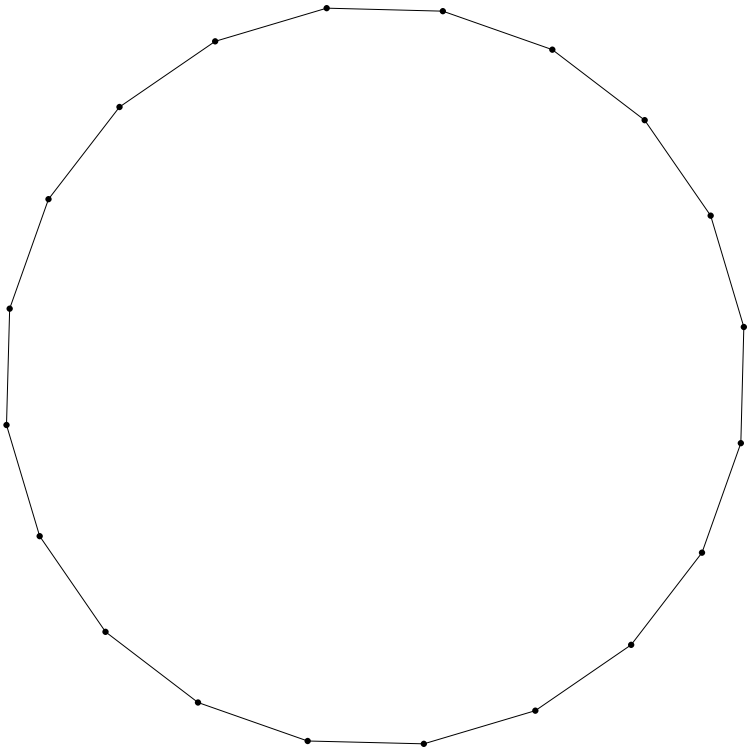
Consider a graph, as a collection of nodes and edges.



We have 3 colors, and we want to to color each node so that every node is a different color than its neighbors. In general we might have k colors.

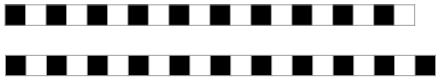
Continuing hypothetical problems, suppose we cannot color the graph so each node is a different color than its neighbors. We might want the number of such nodes to be as small as possible.

Let's go down to two colors on a cyclic network.



Cyclic Network

How can we color it using two colors so that the number of neighbors having the same colors is minimized?



2−colorings of cyclic networks of even and odd length

Now what if we don't have a way of specifying a single node, that we must use a local rule?

Assumption 1. Consider only so-called greedy algorithms. Restrict the CA space to those which do not change a node which is correct.

```
filtered1 =
 Select[Range[0, 255], Function[rn, CellularAutomaton[rn, {0, 1, 0}, 1][[2, 2]] == 1]]
```

```
{4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31, 36, 37, 38, 39, 44, 45, 46,
 47, 52, 53, 54, 55, 60, 61, 62, 63, 68, 69, 70, 71, 76, 77, 78, 79, 84, 85, 86, 87,
 92, 93, 94, 95, 100, 101, 102, 103, 108, 109, 110, 111, 116, 117, 118, 119, 124,
 125, 126, 127, 132, 133, 134, 135, 140, 141, 142, 143, 148, 149, 150, 151, 156, 157,
 158, 159, 164, 165, 166, 167, 172, 173, 174, 175, 180, 181, 182, 183, 188, 189, 190,
 191, 196, 197, 198, 199, 204, 205, 206, 207, 212, 213, 214, 215, 220, 221, 222, 223,
 228, 229, 230, 231, 236, 237, 238, 239, 244, 245, 246, 247, 252, 253, 254, 255}
```

```
filtered2 =
 Select[filtered1, Function[rn, CellularAutomaton[rn, {1, 0, 1}, 1][[2, 2]] == 0]]
```

```
{4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31, 68, 69, 70,
 71, 76, 77, 78, 79, 84, 85, 86, 87, 92, 93, 94, 95, 132, 133, 134, 135,
 140, 141, 142, 143, 148, 149, 150, 151, 156, 157, 158, 159, 196, 197,
 198, 199, 204, 205, 206, 207, 212, 213, 214, 215, 220, 221, 222, 223}
```

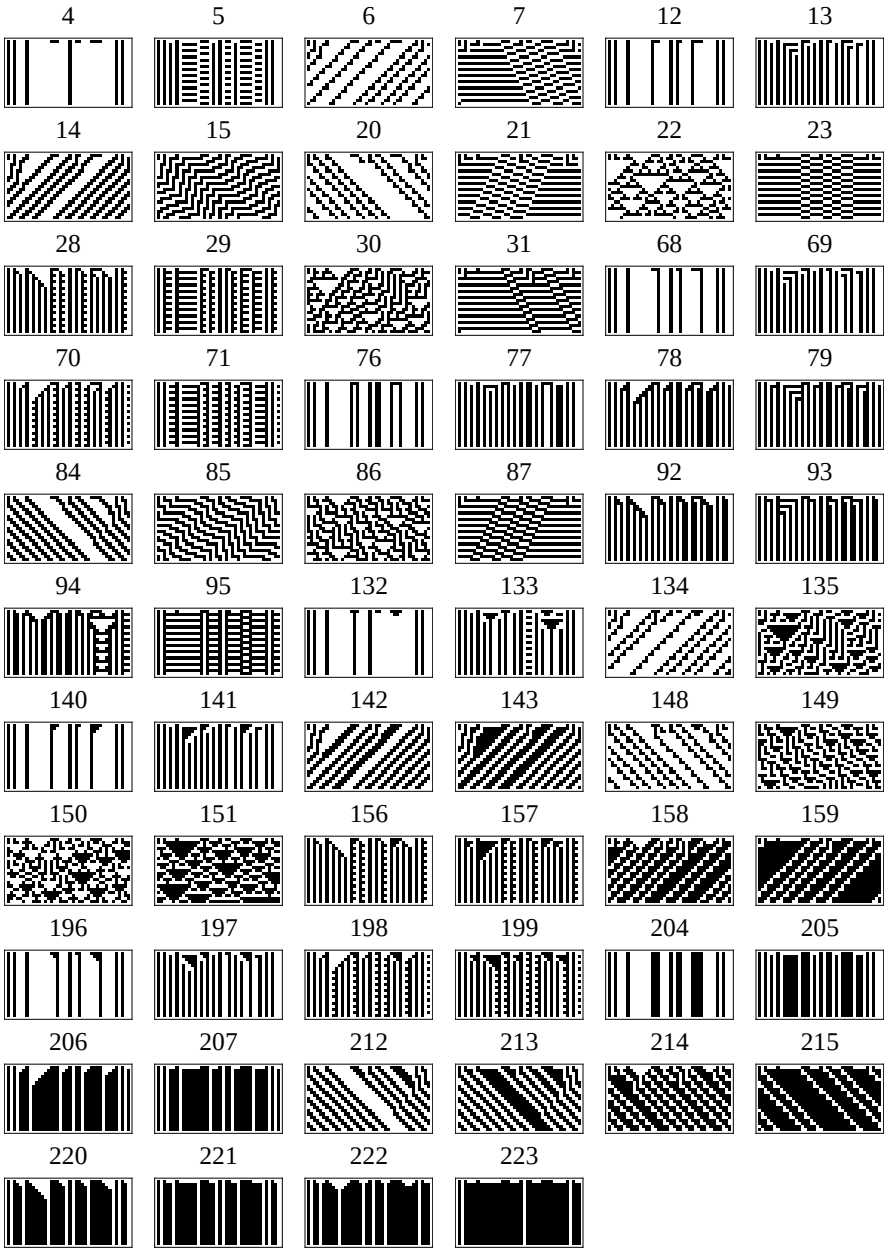
```
Length /@ {filtered1, filtered2}
```

```
{128, 64}
```

```
SeedRandom[30]
```

```
init = Table[Random[Integer], {40}];
```

```
Grid[Partition[Map[Function[rn, ArrayPlot[CellularAutomaton[rn, init, 20],
     PlotLabel → rn, ImageSize → 60]], filtered2], 6, 6, 1, ""]]
```



Those that stabilize do so within the twenty steps shown.

We can write a function to count the number of neighbors which are the same.

```
SameNeighborCount[ls_] := Count[SameQ @@@ Partition[ls, 2, 1, 1], True]
```

```
SameNeighborCount[{1, 2, 3, 1}]
```

```
1
```

```
Partition[Range[10], 2, 1, 1]
```

```
Select[filtered2,
 Function[rn, SameNeighborCount[Last[CellularAutomaton[rn, init, 20]]] == 4]]
```

```
{13, 28, 69, 77, 78, 79, 92, 93, 141, 156, 157, 197}
```

```
{#[[1, 1]], Last /@ #} &[First[Split[Sort[Map[
     Function[rn, {SameNeighborCount[Last[CellularAutomaton[rn, init, 20]]], rn}],
     filtered2]], #[[1]] == #2[[1]] &]]]
```

```
{4, {13, 28, 69, 77, 78, 79, 92, 93, 141, 156, 157, 197}}
```

```
init2 = Table[Random[Integer], {40}];
```

```
{#[[1, 1]], Last /@ #} &[First[Split[Sort[Map[
     Function[rn, {SameNeighborCount[Last[CellularAutomaton[rn, init2, 20]]], rn}],
     {13, 28, 69, 77, 78, 79, 92, 93, 141, 156, 157, 197}]], #[[1]] == #2[[1]] &]]]
```

```
{4, {13, 69, 78, 79, 92, 93, 141, 197}}
```

Step 2. Go back and try them all.

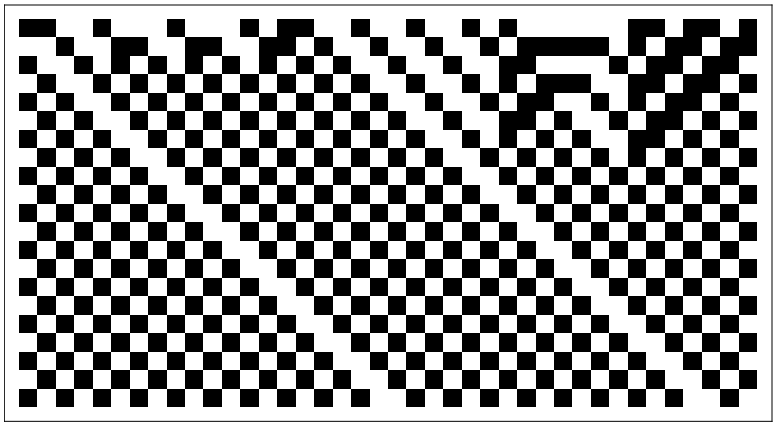
```
{#[[1, 1]], Last /@ #} &[First[Split[Sort[Map[
     Function[rn, {SameNeighborCount[Last[CellularAutomaton[rn, init2, 20]]], rn}],
     Range[0, 255]]], #[[1]] == #2[[1]] &]]]
```

```
{0, {99}}
```

```
{#[[1, 1]], Last /@ #} &[First[Split[Sort[Map[
     Function[rn, {SameNeighborCount[Last[CellularAutomaton[rn, init, 20]]], rn}],
     Range[0, 255]]], #[[1]] == #2[[1]] &]]]
```

```
{0, {57}}
```

```
ArrayPlot[CellularAutomaton[57, {1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,
   0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1}, 20]]
```



```
ArrayPlot[CellularAutomaton[99, {1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,
   0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1}, 40]]
```