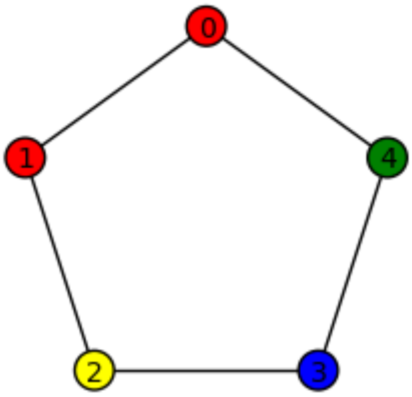
Graph dynamics examples

```
attach "/Users/raichev/graph_dynamics/graph_dynamics.py"
# attach "/Users/arai021/graph_dynamics/graph_dynamics.py"

# For a list of Sage's graph generators, see
http://wiki.sagemath.org/graph_generators.
```

```
# Example: Using color()

G = graphs.CycleGraph(5)
coloring = color(G, ['red', 'red', 'yellow', 'blue', 'green'])
print(coloring)
G.show(vertex_colors=invert_dict(coloring), figsize=3)
```

```
{0: 'red', 1: 'red', 2: 'yellow', 3: 'blue', 4: 'green'}
```

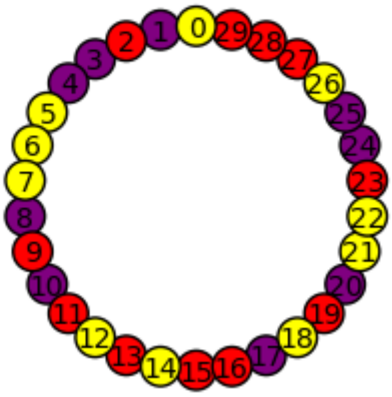



```
# Example: Using color_randomly()

G = graphs.CycleGraph(30)
coloring = color_randomly(G, ['red', 'yellow', 'purple'])
print(coloring)
G.show(vertex_colors=invert_dict(coloring), figsize=3)
```

```
{0: 'purple', 1: 'yellow', 2: 'yellow', 3: 'yellow', 4: 'red'
'yellow', 6: 'yellow', 7: 'red', 8: 'purple', 9: 'red', 10: '
11: 'purple', 12: 'yellow', 13: 'yellow', 14: 'red', 15: 'rec
'yellow', 17: 'red', 18: 'purple', 19: 'red', 20: 'red', 21:
'purple', 22: 'red', 23: 'yellow', 24: 'red', 25: 'purple', 2
'red', 27: 'purple', 28: 'purple', 29: 'purple'}
```


```
# Example: Using color_count()

G = graphs.CycleGraph(30)
coloring = color_randomly(G, ['red', 'yellow', 'purple'])
print(coloring)
G.show(vertex_colors=invert_dict(coloring), figsize=3)
print(color_count(coloring))
```

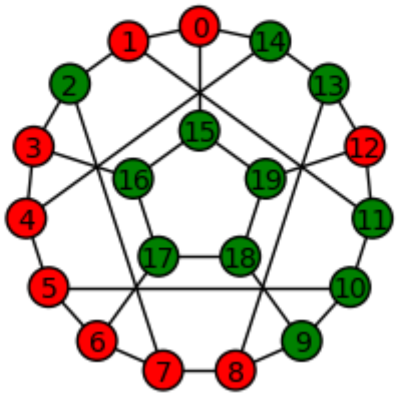
```
{0: 'yellow', 1: 'purple', 2: 'red', 3: 'purple', 4: 'purple'
'yellow', 6: 'yellow', 7: 'yellow', 8: 'purple', 9: 'red', 1
'purple', 11: 'red', 12: 'yellow', 13: 'red', 14: 'yellow', 1
'red', 16: 'red', 17: 'purple', 18: 'yellow', 19: 'red', 20:
'purple', 21: 'yellow', 22: 'yellow', 23: 'red', 24: 'purple'
'purple', 26: 'yellow', 27: 'red', 28: 'red', 29: 'red'}
```



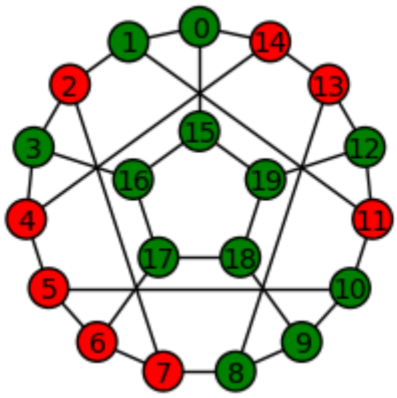

```
Counter({'red': 11, 'yellow': 10, 'purple': 9})
```

```
# Example: Iterating the majority rule

G = graphs.FlowerSnark()
color_palette = ['green', 'red']
ur = majority_rule
ur_kwargs = {}
initial_coloring = color_randomly(G, color_palette)
s, stabilized = iterate(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
print(s)
show_colorings(G, s, vertex_labels=True)
```

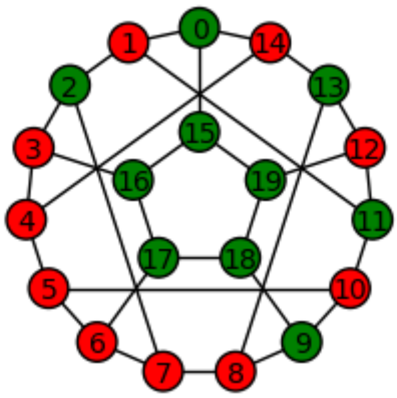
```
Stabilized?
    False
[{0: 'red', 1: 'red', 2: 'green', 3: 'red', 4: 'red', 5: 'rec
'red', 7: 'red', 8: 'red', 9: 'green', 10: 'green', 11: 'gree
'red', 13: 'green', 14: 'green', 15: 'green', 16: 'green', 17
'green', 18: 'green', 19: 'green'}, {0: 'green', 1: 'green',
'red', 3: 'green', 4: 'red', 5: 'red', 6: 'red', 7: 'red', 8:
'green', 9: 'green', 10: 'green', 11: 'red', 12: 'green', 13:
14: 'red', 15: 'green', 16: 'green', 17: 'green', 18: 'green'
'green'}, {0: 'green', 1: 'red', 2: 'green', 3: 'red', 4: 're
'red', 6: 'red', 7: 'red', 8: 'red', 9: 'green', 10: 'red', 1
'green', 12: 'red', 13: 'green', 14: 'red', 15: 'green', 16:
'green', 17: 'green', 18: 'green', 19: 'green'}, {0: 'red', 1
'green', 2: 'red', 3: 'green', 4: 'red', 5: 'red', 6: 'red',
'red', 8: 'green', 9: 'red', 10: 'green', 11: 'red', 12: 'gre
13: 'red', 14: 'green', 15: 'green', 16: 'green', 17: 'green'
'green', 19: 'green'}, {0: 'green', 1: 'red', 2: 'green', 3:
4: 'green', 5: 'red', 6: 'red', 7: 'red', 8: 'red', 9: 'greer
'red', 11: 'green', 12: 'red', 13: 'green', 14: 'red', 15: 'g
16: 'green', 17: 'green', 18: 'green', 19: 'green'}, {0: 'rec
'green', 2: 'red', 3: 'green', 4: 'red', 5: 'red', 6: 'red',
'red', 8: 'green', 9: 'red', 10: 'green', 11: 'red', 12: 'gre
13: 'red', 14: 'green', 15: 'green', 16: 'green', 17: 'green'
'green', 19: 'green'}, {0: 'green', 1: 'red', 2: 'green', 3:
4: 'green', 5: 'red', 6: 'red', 7: 'red', 8: 'red', 9: 'greer
'red', 11: 'green', 12: 'red', 13: 'green', 14: 'red', 15: 'g
16: 'green', 17: 'green', 18: 'green', 19: 'green'}, {0: 'rec
'green', 2: 'red', 3: 'green', 4: 'red', 5: 'red', 6: 'red',
'red', 8: 'green', 9: 'red', 10: 'green', 11: 'red', 12: 'gre
13: 'red', 14: 'green', 15: 'green', 16: 'green', 17: 'green'
'green', 19: 'green'}, {0: 'green', 1: 'red', 2: 'green', 3:
4: 'green', 5: 'red', 6: 'red', 7: 'red', 8: 'red', 9: 'greer
'red', 11: 'green', 12: 'red', 13: 'green', 14: 'red', 15: 'g
16: 'green', 17: 'green', 18: 'green', 19: 'green'}, {0: 'rec
'green', 2: 'red', 3: 'green', 4: 'red', 5: 'red', 6: 'red',
'red', 8: 'green', 9: 'red', 10: 'green', 11: 'red', 12: 'gre
13: 'red', 14: 'green', 15: 'green', 16: 'green', 17: 'green'
'green', 19: 'green'}, {0: 'green', 1: 'red', 2: 'green', 3:
4: 'green', 5: 'red', 6: 'red', 7: 'red', 8: 'red', 9: 'greer
'red', 11: 'green', 12: 'red', 13: 'green', 14: 'red', 15: 'g
16: 'green', 17: 'green', 18: 'green', 19: 'green'}]
Step 0
```



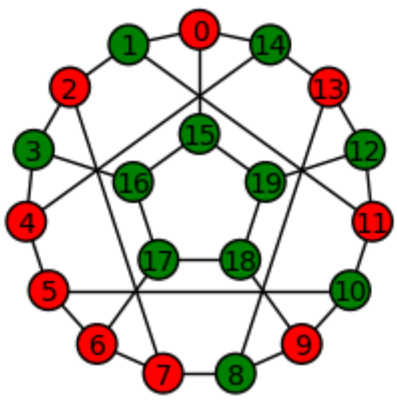
Step 1



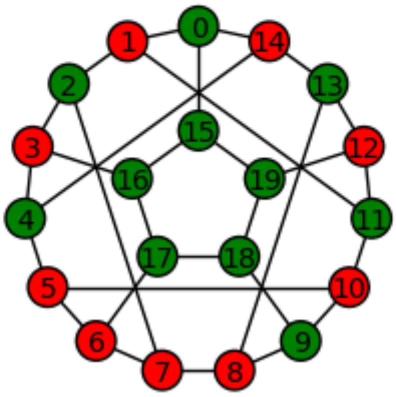
Step 2



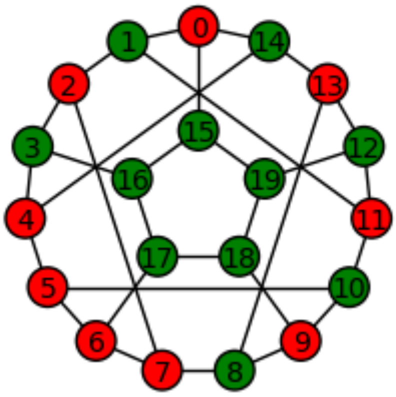
Step 3



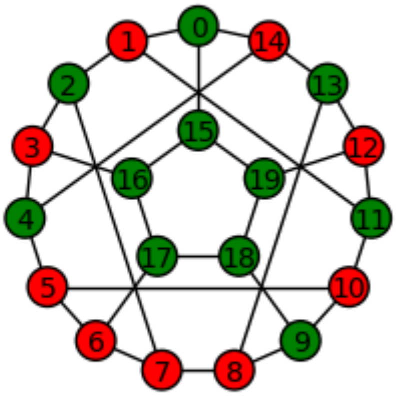
Step 4



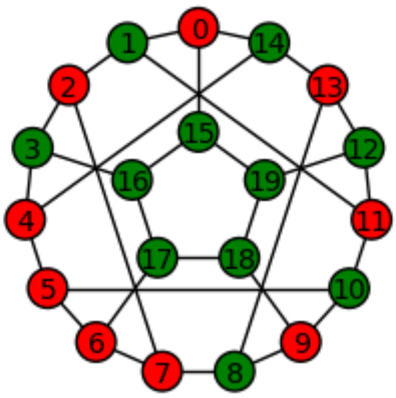
Step 5



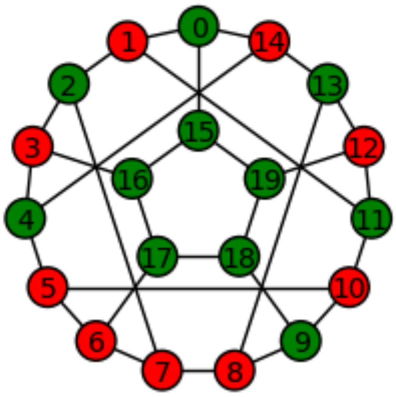
Step 6



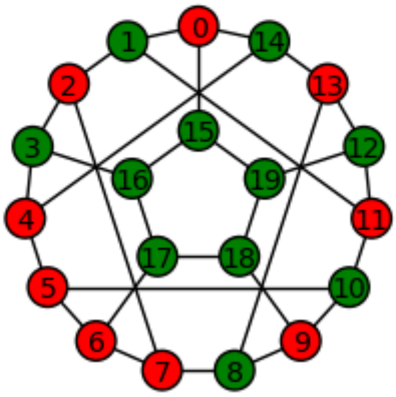
Step 7



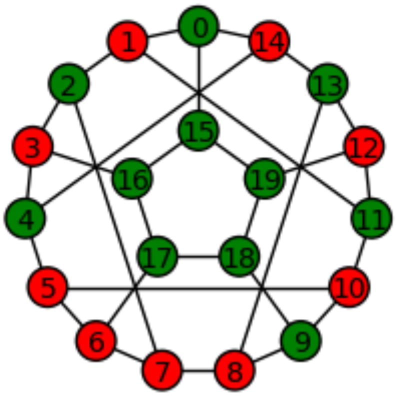
Step 8



Step 9



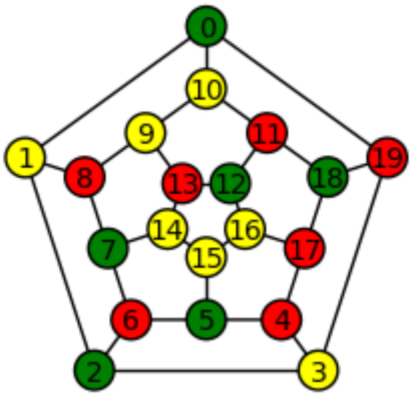
Step 10



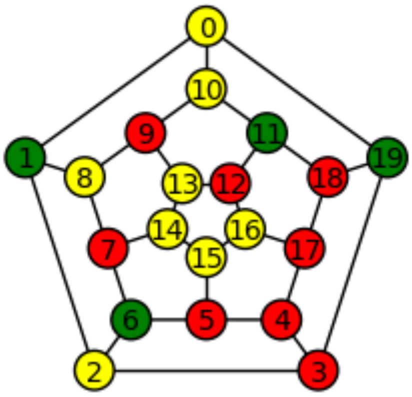

```
# Example: Iterating the plurality rule.

G = graphs.DodecahedralGraph()
ur = plurality_rule
ur_kwargs = {}
color_palette = ['green', 'red', 'yellow']
initial_coloring = color_randomly(G, color_palette)

s, stabilized = iterate(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s, vertex_labels=True)
```

```
Stabilized?
    False
```

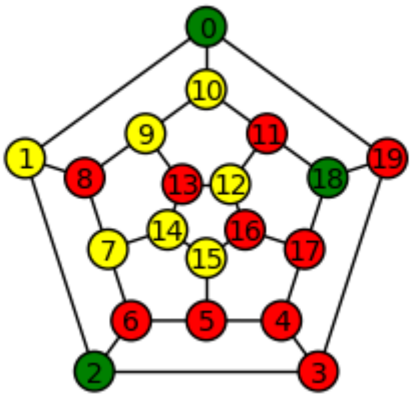
Step 0



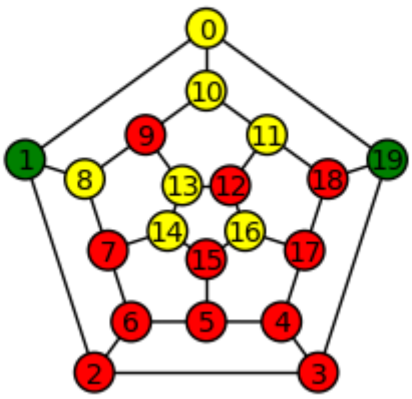
Step 1



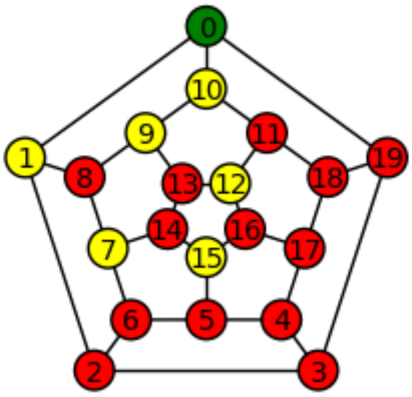
Step 2



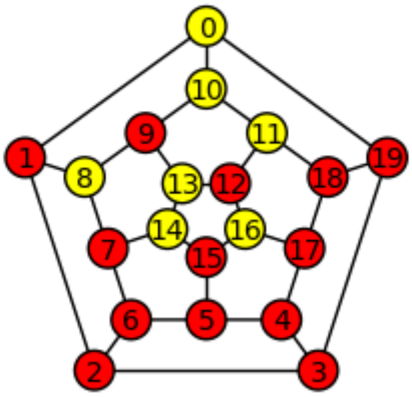
Step 3



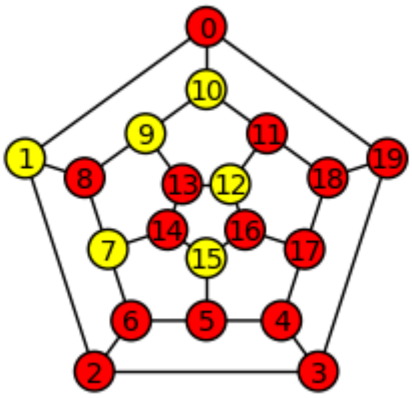
Step 4



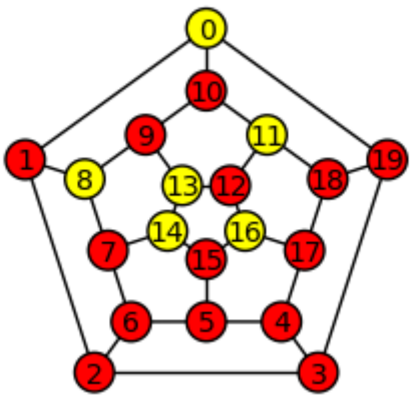
Step 5



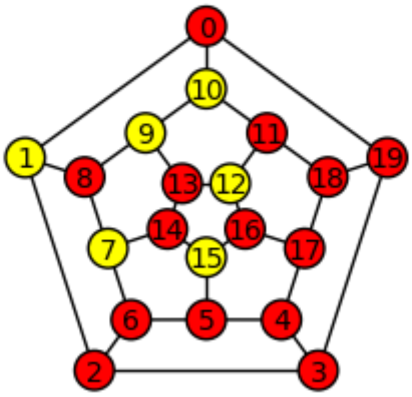
Step 6



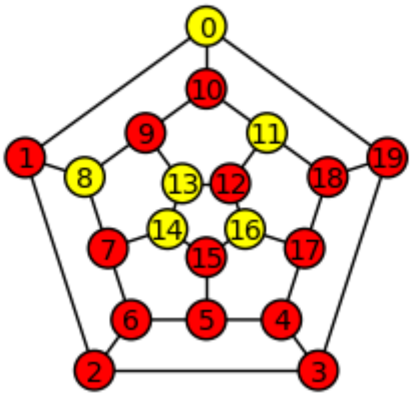
Step 7



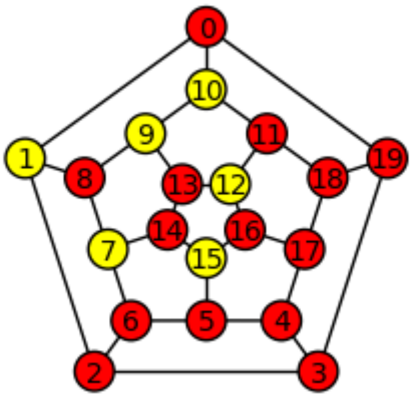
Step 8



Step 9



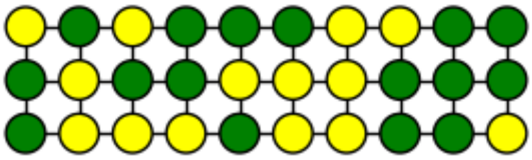
Step 10



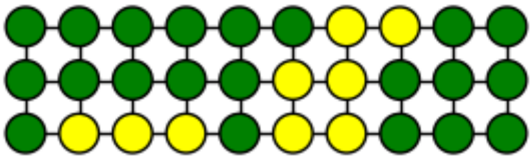

```
# Example: Iterating the GSL2 rule.

G = graphs.Grid2dGraph(3, 10)
color_palette = ['green', 'yellow']
ur = gsl2_rule
ur_kwargs = {'color_palette': color_palette, 'T': 0.7}
initial_coloring = color_randomly(G, color_palette)

s, stabilized = iterate(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s)
```

Stabilized?

```
    True
Step 0
```

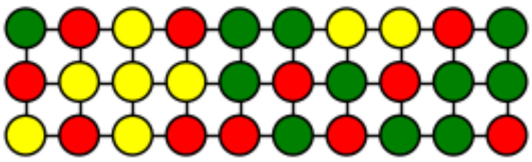


```
Step 1
```

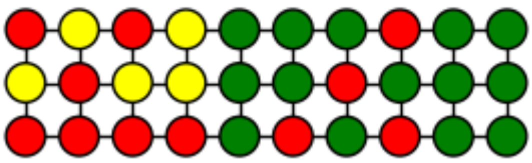


```
# Example: Iterating the GSL3 rule.

G = graphs.Grid2dGraph(3, 10)
color_palette = ['green', 'red', 'yellow']
ur = gsl3_rule
ur_kwargs = {'color_palette': color_palette, 'T': 0.6}
initial_coloring = color_randomly(G, color_palette)

s, stabilized = iterate(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s)
```

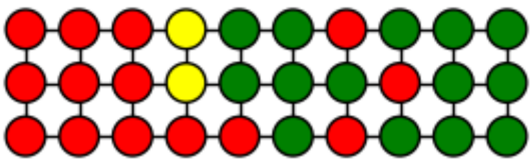
```
Stabilized?
    False
Step 0
```



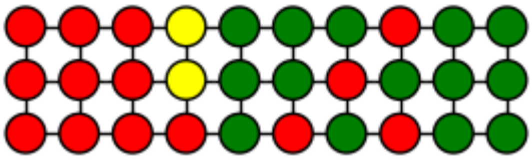
```
Step 1
```



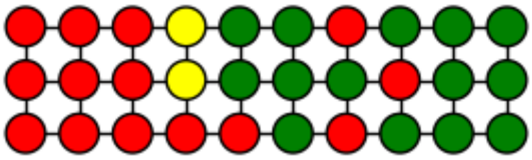
```
Step 2
```



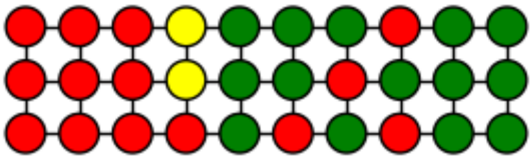
```
Step 3
```



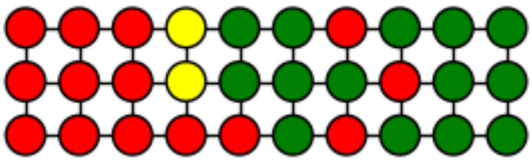
Step 4



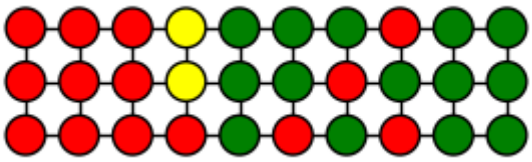
Step 5



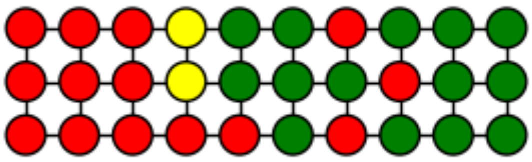
Step 6



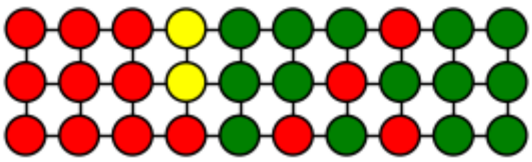
Step 7



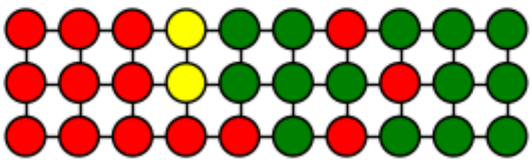
Step 8



Step 9



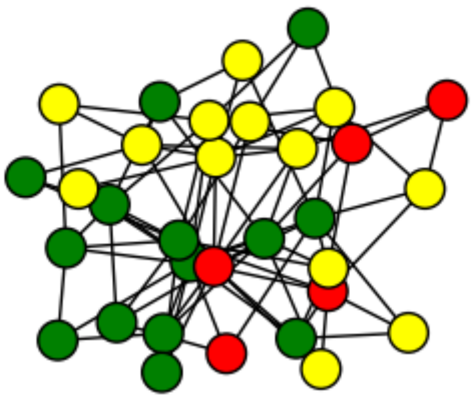
Step 10



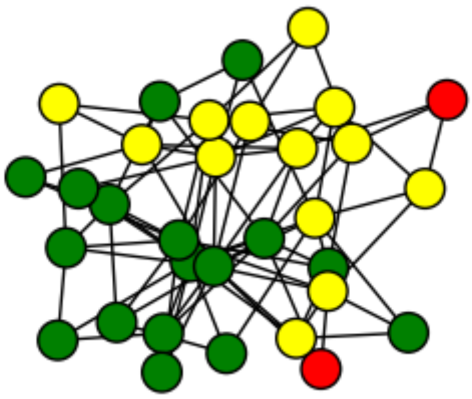
```
# Example: Iterating the GSL3 rule on a random graph
```

```
gg = graphs.RandomBarabasiAlbert
print(gg)
G = gg(32, 3)
color_palette = ['green', 'red', 'yellow']
ur = gsl3_rule
ur_kwargs = {'color_palette': color_palette, 'T': 0.6}
initial_coloring = color_randomly(G, color_palette)

s, stabilized = iterate(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s)
```

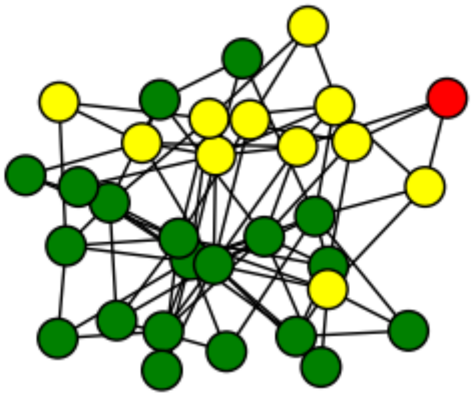
```
<function RandomBarabasiAlbert at 0x114c9f9b0>
Stabilized?
    True
Step 0
```



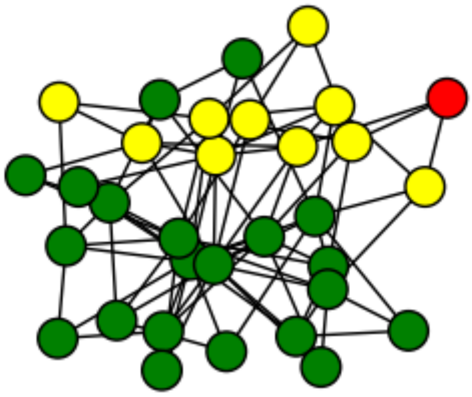
```
Step 1
```



```
Step 2
```



Step 3



```
graphs.Grid2dGraph??
```

File: /Applications/sage/local/lib/python2.7/site-packages/sage/graphs/generators/basic.py

Source Code (starting at line 719):

```
def Grid2dGraph(n1, n2):
    r"""
    Returns a `2`-dimensional grid graph with `n_1n_2` nodes (`n_1` ro
    `n_2` columns).

    A 2d grid graph resembles a `2` dimensional grid. All inner nodes
    connected to their `4` neighbors. Outer (non-corner) nodes are
    connected to their `3` neighbors. Corner nodes are connected to th
    2 neighbors.

    This constructor depends on NetworkX numeric labels.

    PLOTTING: Upon construction, the position dictionary is filled to
    override the spring-layout algorithm. By convention, nodes are
    labelled in (row, column) pairs with `(0, 0)` in the top left corn
    Edges will always be horizontal and vertical - another advantage o
    filling the position dictionary.

    EXAMPLES: Construct and show a grid 2d graph Rows = `5`, Columns =

    ::

        sage: g = graphs.Grid2dGraph(5,7)
        sage: g.show() # long time
```

```
    TESTS:

    Senseless input::

        sage: graphs.Grid2dGraph(5,0)
        Traceback (most recent call last):
        ...
        ValueError: Parameters n1 and n2 must be positive integers !
        sage: graphs.Grid2dGraph(-1,0)
        Traceback (most recent call last):
        ...
        ValueError: Parameters n1 and n2 must be positive integers !
    """

    if n1 <= 0 or n2 <= 0:
        raise ValueError("Parameters n1 and n2 must be positive intege

    pos_dict = {}
    for i in range(n1):
        y = -i
        for j in range(n2):
            x = j
            pos_dict[i,j] = (x,y)
    import networkx
    G = networkx.grid_2d_graph(n1,n2)
    return graph.Graph(G, pos=pos_dict, name="2D Grid Graph")
```

```
# Example: Using get_stats() on a fixed graph

color_palette = ['green', 'red', 'yellow']
ur = gsl3_rule
urk = {'color_palette': color_palette}
gg = graphs.Grid2dGraph
ggk = {'n1': 3, 'n2': 10}
cf = color_randomly
cfk = {'color_palette': color_palette}

num_runs = 1000
num_stabilized, mean_steps, mean_initial, mean_final =
get_stats(ur, urk, gg, ggk, cf, cfk, num_runs=num_runs)
```

```
<function gsl3_rule at 0x114c2b500>
<function Grid2dGraph at 0x114c42c80>
<function color_randomly at 0x1163c4c80>
--------------------
Number of runs: 1000
Number of runs that stabilized: 802
Mean number of steps required to stabilize: 4.5
Mean initial color counts:
    green: 9.86
    red: 9.75
    yellow: 10.4
Mean finial color counts:
    green: 11.7
    red: 11.4
    yellow: 6.9
```

```
graphs.RandomBarabasiAlbert??
```

File: /Applications/sage/local/lib/python2.7/site-packages/sage/graphs/generators/random.py

Source Code (starting at line 127):

```
def RandomBarabasiAlbert(n, m, seed=None):
    u"""
    Return a random graph created using the Barabasi-Albert preferenti
    attachment model.

    A graph with m vertices and no edges is initialized, and a graph o
    vertices is grown by attaching new vertices each with m edges that
    attached to existing vertices, preferentially with high degree.

    INPUT:

    - ``n`` - number of vertices in the graph

    - ``m`` - number of edges to attach from each new node

    - ``seed`` - for random number generator

    EXAMPLES:

    We show the edge list of a random graph on 6 nodes with m = 2.

    ::

        sage: graphs.RandomBarabasiAlbert(6,2).edges(labels=False)
        [(0, 2), (0, 3), (0, 4), (1, 2), (2, 3), (2, 4), (2, 5), (3, 5

    We plot a random graph on 12 nodes with m = 3.

    ::

        sage: ba = graphs.RandomBarabasiAlbert(12,3)
        sage: ba.show()  # long time

    We view many random graphs using a graphics array::

        sage: g = []
        sage: j = []
        sage: for i in range(1,10):
        ....:     k = graphs.RandomBarabasiAlbert(i+3, 3)
        ....:     g.append(k)
        sage: for i in range(3):
        ....:     n = []
        ....:     for m in range(3):
        ....:         n.append(g[3*i + m].plot(vertex_size=50, vertex_
        ....:     j.append(n)
        sage: G = sage.plot.graphics.GraphicsArray(j)
        sage: G.show()  # long time

    """
    if seed is None:
        seed = current_randstate().long_seed()
    import networkx
    return Graph(networkx.barabasi_albert_graph(n,m,seed=seed))
```

```
# Example: Using get_stats() on a random graph

color_palette = ['green', 'red', 'yellow']
ur = gsl3_rule
urk = {'color_palette': color_palette}
gg = graphs.RandomBarabasiAlbert
ggk = {'n': 52, 'm': 3}
cf = color_randomly
cfk = {'color_palette': color_palette}

num_runs = 100
num_stabilized, mean_steps, mean_initial, mean_final =
get_stats(ur, urk, gg, ggk, cf, cfk, num_runs=num_runs)
```

```
<function gsl3_rule at 0x114c2b500>
<function RandomBarabasiAlbert at 0x114c9f9b0>
<function color_randomly at 0x1163c4c80>
--------------------
Number of runs: 100
Number of runs that stabilized: 98
Mean number of steps required to stabilize: 5.13
Mean initial color counts:
    green: 17.1
    red: 17
    yellow: 18
Mean finial color counts:
    green: 22.4
    red: 17.1
    yellow: 12.5
```