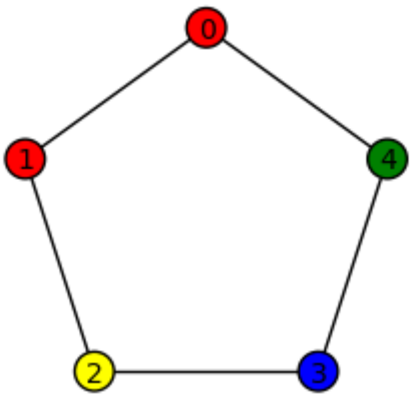
# examples

```
attach "/Users/araichev/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.
```

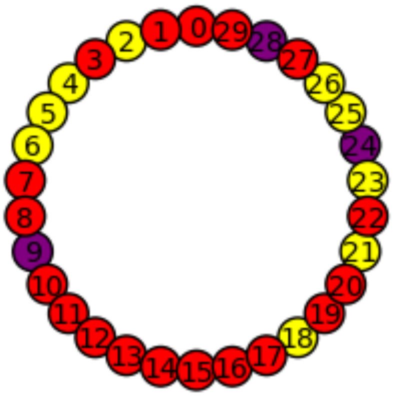
```
# Example: Use color()

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'}
```

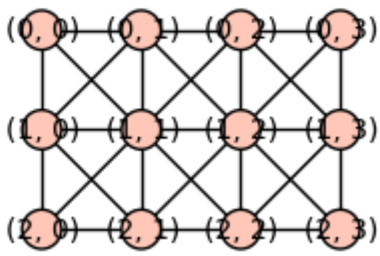



```
# Example: Use color_randomly() and color_count()

G = graphs.CycleGraph(30)
coloring = color_randomly(G, {'red': 0.6, 'yellow': 0.3,
'purple': 1/10})
print(coloring)
print('Color count = {!s}'.format(color_count(coloring)))
G.show(vertex_colors=invert_dict(coloring), figsize=3)
```

```
{0: 'red', 1: 'red', 2: 'yellow', 3: 'red', 4: 'yellow', 5:
'yellow', 6: 'yellow', 7: 'red', 8: 'red', 9: 'purple', 10: '
11: 'red', 12: 'red', 13: 'red', 14: 'red', 15: 'red', 16: 'r
17: 'red', 18: 'yellow', 19: 'red', 20: 'red', 21: 'yellow',
'red', 23: 'yellow', 24: 'purple', 25: 'yellow', 26: 'yellow'
'red', 28: 'purple', 29: 'red'}
Color count = Counter({'red': 18, 'yellow': 9, 'purple': 3})
```

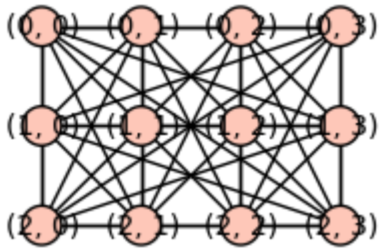


```
# Test the Moore lattice generator

G = moore_lattice(3, 4)
G.show(figsize=2)
print('num edges = {!s}'.format(G.num_edges()))
print('degrees = %s' % G.degree())
print(G.edges())


G = moore_lattice(3, 4, toroidal=True)
G.show(figsize=2)
print('num edges = %s' % G.num_edges())
print('degrees = %s' % G.degree())
print(G.edges())

G = moore_lattice(2, 4)
coloring = color(G, ['red', 'red', 'yellow', 'blue', 'green',
'green', 'blue', 'blue'])
G.show(vertex_colors=invert_dict(coloring), figsize=3)
print(coloring)
```

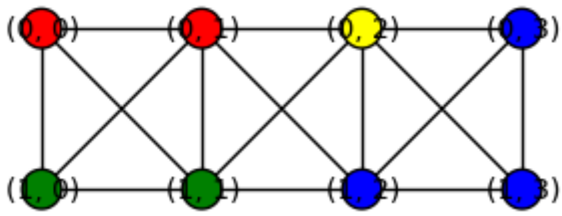


```
num edges = 29
degrees = [5, 8, 3, 5, 8, 3, 5, 3, 5, 5, 3, 5]
[((0, 0), (0, 1), None), ((0, 0), (1, 0), None), ((0, 0), (1,
None), ((0, 1), (0, 2), None), ((0, 1), (1, 0), None), ((0, 1
1), None), ((0, 1), (1, 2), None), ((0, 2), (0, 3), None), ((
(1, 1), None), ((0, 2), (1, 2), None), ((0, 2), (1, 3), None)
3), (1, 2), None), ((0, 3), (1, 3), None), ((1, 0), (1, 1), N
((1, 0), (2, 0), None), ((1, 0), (2, 1), None), ((1, 1), (1,
None), ((1, 1), (2, 0), None), ((1, 1), (2, 1), None), ((1, 1
2), None), ((1, 2), (1, 3), None), ((1, 2), (2, 1), None), ((
```

```
(2, 2), None), ((1, 2), (2, 3), None), ((1, 3), (2, 2), None)
3), (2, 3), None), ((2, 0), (2, 1), None), ((2, 1), (2, 2), N
((2, 2), (2, 3), None)]
```



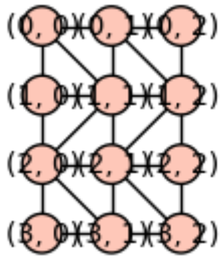
```
num edges = 48
degrees = [8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]
[((0, 0), (0, 1), None), ((0, 0), (0, 3), None), ((0, 0), (1,
None), ((0, 0), (1, 1), None), ((0, 0), (1, 3), None), ((0, 0
0), None), ((0, 0), (2, 1), None), ((0, 0), (2, 3), None), ((
(0, 2), None), ((0, 1), (1, 0), None), ((0, 1), (1, 1), None)
1), (1, 2), None), ((0, 1), (2, 0), None), ((0, 1), (2, 1), N
((0, 1), (2, 2), None), ((0, 2), (0, 3), None), ((0, 2), (1,
None), ((0, 2), (1, 2), None), ((0, 2), (1, 3), None), ((0, 2
1), None), ((0, 2), (2, 2), None), ((0, 2), (2, 3), None), ((
(1, 0), None), ((0, 3), (1, 2), None), ((0, 3), (1, 3), None)
3), (2, 0), None), ((0, 3), (2, 2), None), ((0, 3), (2, 3), N
((1, 0), (1, 1), None), ((1, 0), (1, 3), None), ((1, 0), (2,
None), ((1, 0), (2, 1), None), ((1, 0), (2, 3), None), ((1, 1
2), None), ((1, 1), (2, 0), None), ((1, 1), (2, 1), None), ((
(2, 2), None), ((1, 2), (1, 3), None), ((1, 2), (2, 1), None)
2), (2, 2), None), ((1, 2), (2, 3), None), ((1, 3), (2, 0), N
((1, 3), (2, 2), None), ((1, 3), (2, 3), None), ((2, 0), (2,
None), ((2, 0), (2, 3), None), ((2, 1), (2, 2), None), ((2, 2
3), None)]
```



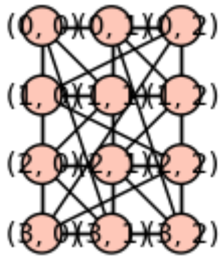
```
{(0, 1): 'red', (1, 2): 'blue', (0, 0): 'red', (0, 2): 'yello
3): 'blue', (1, 0): 'green', (0, 3): 'blue', (1, 1): 'green'}
```

```
# Test the triangular lattice generator

G = triangular_lattice(4, 3)
G.show(figsize=2)
print('num edges = {!s}'.format(G.num_edges()))
print('degrees = %s' % G.degree())
print(G.edges())


G = triangular_lattice(4, 3, toroidal=True)
G.show(figsize=2)
print('num edges = %s' % G.num_edges())
```

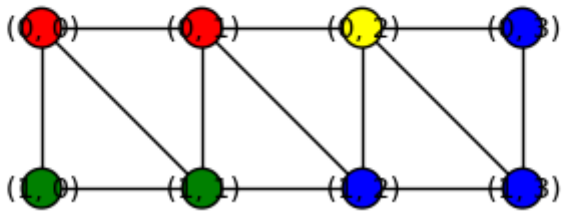
```
print('degrees = %s' % G.degree())
print(G.edges())

G = triangular_lattice(2, 4)
coloring = color(G, ['red', 'red', 'yellow', 'blue', 'green',
'green', 'blue', 'blue'])
G.show(vertex_colors=invert_dict(coloring), figsize=3)
print(coloring)
```



```
num edges = 23
degrees = [4, 5, 3, 3, 2, 4, 6, 6, 5, 3, 3, 2]
[((0, 0), (0, 1), None), ((0, 0), (1, 0), None), ((0, 0), (1,
None), ((0, 1), (0, 2), None), ((0, 1), (1, 1), None), ((0, 1
2), None), ((0, 2), (1, 2), None), ((1, 0), (1, 1), None), ((
(2, 0), None), ((1, 1), (1, 2), None), ((1, 1), (2, 0), None)
1), (2, 1), None), ((1, 2), (2, 1), None), ((1, 2), (2, 2), N
((2, 0), (2, 1), None), ((2, 0), (3, 0), None), ((2, 0), (3,
None), ((2, 1), (2, 2), None), ((2, 1), (3, 1), None), ((2, 1
2), None), ((2, 2), (3, 2), None), ((3, 0), (3, 1), None), ((
(3, 2), None)]
```

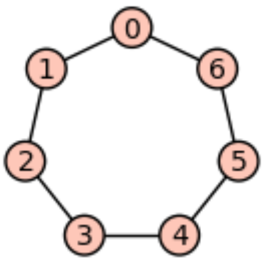


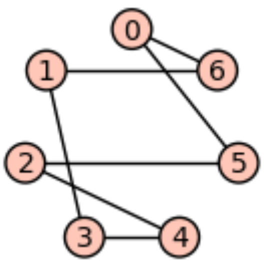
```
num edges = 36
degrees = [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]
[((0, 0), (0, 1), None), ((0, 0), (0, 2), None), ((0, 0), (1,
None), ((0, 0), (1, 1), None), ((0, 0), (3, 0), None), ((0, 0
1), None), ((0, 1), (0, 2), None), ((0, 1), (1, 1), None), ((
(1, 2), None), ((0, 1), (3, 1), None), ((0, 1), (3, 2), None)
2), (1, 0), None), ((0, 2), (1, 2), None), ((0, 2), (3, 0), N
((0, 2), (3, 2), None), ((1, 0), (1, 1), None), ((1, 0), (1,
None), ((1, 0), (2, 0), None), ((1, 0), (2, 2), None), ((1, 1
2), None), ((1, 1), (2, 0), None), ((1, 1), (2, 1), None), ((
(2, 1), None), ((1, 2), (2, 2), None), ((2, 0), (2, 1), None)
0), (2, 2), None), ((2, 0), (3, 0), None), ((2, 0), (3, 1), N
((2, 1), (2, 2), None), ((2, 1), (3, 1), None), ((2, 1), (3,
None), ((2, 2), (3, 0), None), ((2, 2), (3, 2), None), ((3, 0
1), None), ((3, 0), (3, 2), None), ((3, 1), (3, 2), None)]
```

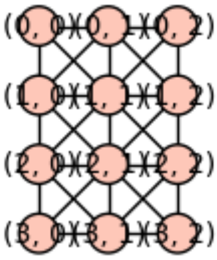


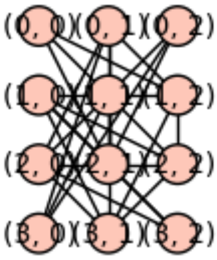

```
{(0, 1): 'red', (1, 2): 'blue', (0, 0): 'red', (0, 2): 'yello
3): 'blue', (1, 0): 'green', (0, 3): 'blue', (1, 1): 'green'}
```

```
# Test the Maslov-Sneppen rewiring method

G = graphs.CycleGraph(7)
G.show(figsize=2)
H = maslov_sneppen(G)
H.show(figsize=2)

G = moore_lattice(4, 3)
G.show(figsize=2)
H = maslov_sneppen(G)
H.show(figsize=2)
print(G.num_verts() == H.num_verts())
print(G.num_edges() == H.num_edges())
print(G.degree() == H.degree())

G = graphs.RandomBarabasiAlbert(50, 5)
H = maslov_sneppen(G)
print(G.num_verts() == H.num_verts())
print(G.num_edges() == H.num_edges())
print(G.degree() == H.degree())
```





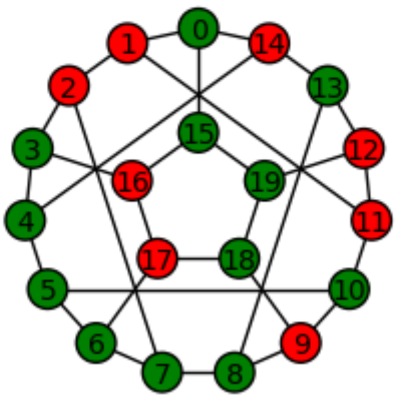




```
True
True
True
True
True
True
```

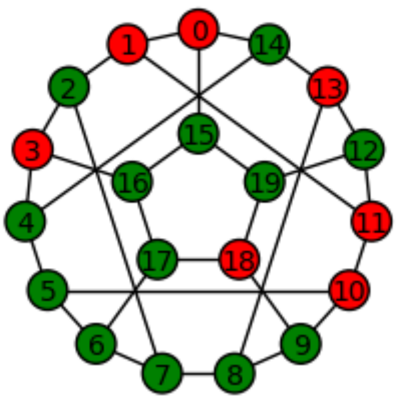
```
# Example: Run the majority rule

G = graphs.FlowerSnark()
color_bias = {'green': 0.7, 'red': 0.3}
ur = majority_rule
ur_kwargs = {}
initial_coloring = color_randomly(G, color_bias)
s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
print(s)
show_colorings(G, s, vertex_labels=True)
```

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

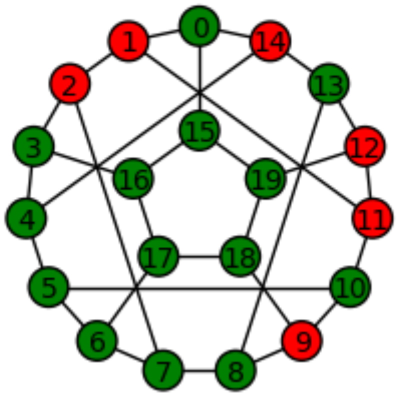
```
8: 'green', 9: 'green', 10: 'green', 11: 'red', 12: 'red', 13
'green', 14: 'red', 15: 'green', 16: 'green', 17: 'green', 18
'green', 19: 'green'}, {0: 'red', 1: 'green', 2: 'green', 3:
'green', 4: 'green', 5: 'green', 6: 'green', 7: 'green', 8: '
9: 'green', 10: 'green', 11: 'red', 12: 'green', 13: 'red', 1
'green', 15: 'green', 16: 'green', 17: 'green', 18: 'green',
'green'}, {0: 'green', 1: 'red', 2: 'green', 3: 'green', 4: '
5: 'green', 6: 'green', 7: 'green', 8: 'green', 9: 'green', 1
'green', 11: 'green', 12: 'red', 13: 'green', 14: 'red', 15:
'green', 16: 'green', 17: 'green', 18: 'green', 19: 'green'},
'red', 1: 'green', 2: 'green', 3: 'green', 4: 'green', 5: 'gr
6: 'green', 7: 'green', 8: 'green', 9: 'green', 10: 'green',
'red', 12: 'green', 13: 'red', 14: 'green', 15: 'green', 16:
'green', 17: 'green', 18: 'green', 19: 'green'}, {0: 'green',
'red', 2: 'green', 3: 'green', 4: 'green', 5: 'green', 6: 'gr
7: 'green', 8: 'green', 9: 'green', 10: 'green', 11: 'green',
'red', 13: 'green', 14: 'red', 15: 'green', 16: 'green', 17:
'green', 18: 'green', 19: 'green'}, {0: 'red', 1: 'green', 2:
'green', 3: 'green', 4: 'green', 5: 'green', 6: 'green', 7: '
8: 'green', 9: 'green', 10: 'green', 11: 'red', 12: 'green',
'red', 14: 'green', 15: 'green', 16: 'green', 17: 'green', 18
'green', 19: 'green'}, {0: 'green', 1: 'red', 2: 'green', 3:
'green', 4: 'green', 5: 'green', 6: 'green', 7: 'green', 8: '
9: 'green', 10: 'green', 11: 'green', 12: 'red', 13: 'green',
'red', 15: 'green', 16: 'green', 17: 'green', 18: 'green', 19
'green'}]
Step 0
```



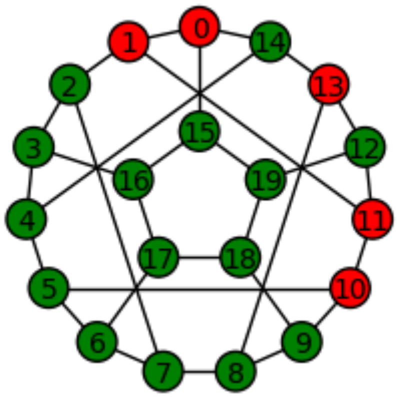
```
Step 1
```



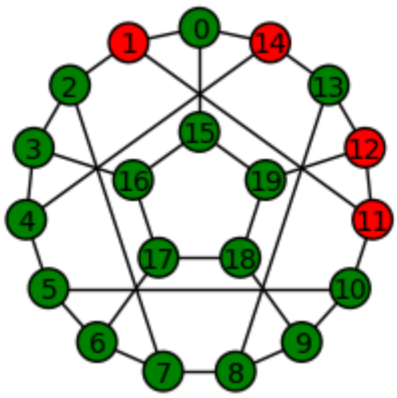
```
Step 2
```



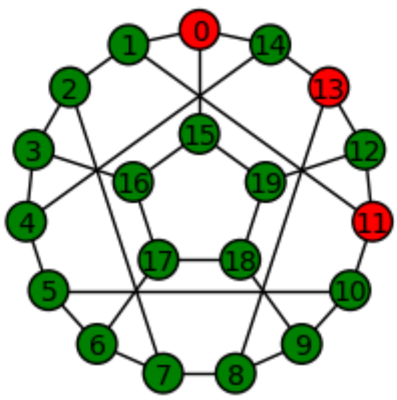

Step 3



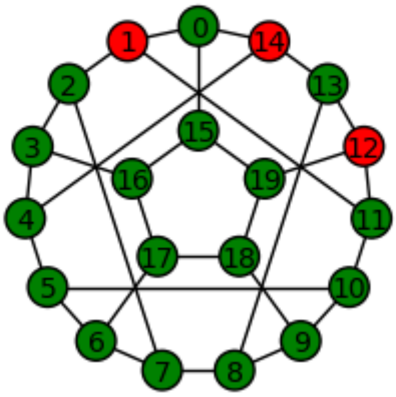

Step 4



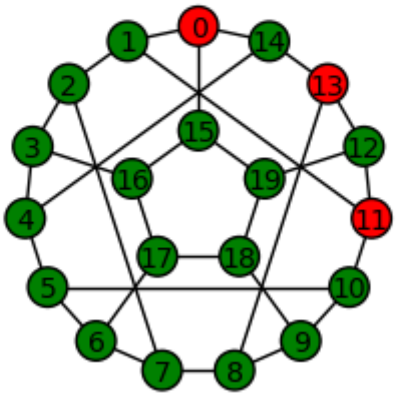

Step 5



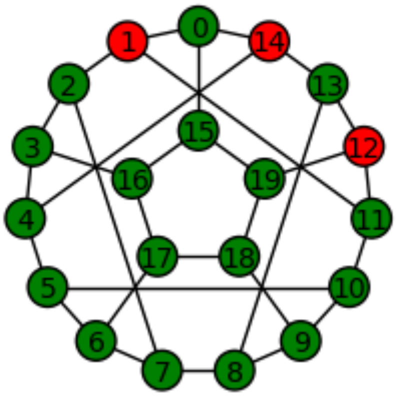

Step 6



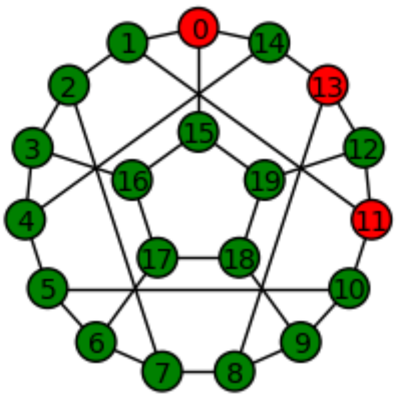

Step 7



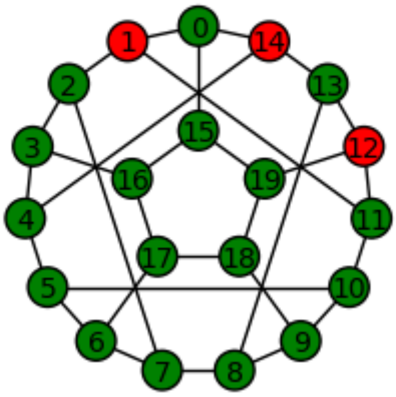

Step 8




Step 9

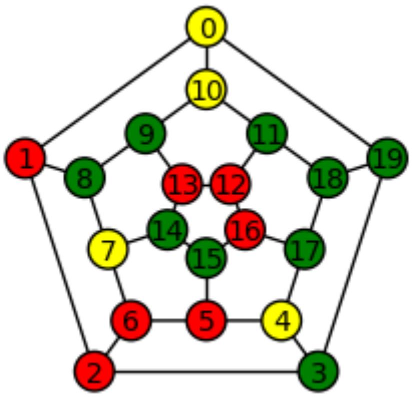



Step 10

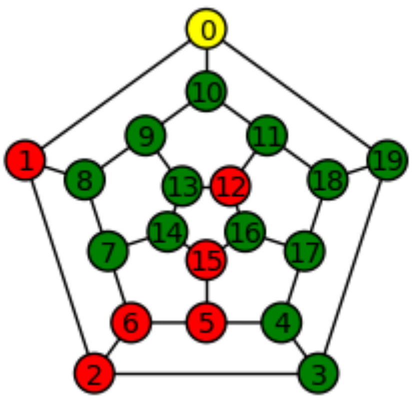



```
# Example: Run the plurality rule.

G = graphs.DodecahedralGraph()
ur = plurality_rule
ur_kwargs = {}
color_bias = {'green': 0.6, 'red': 0.3, 'yellow': 0.1}
initial_coloring = color_randomly(G, color_bias)

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

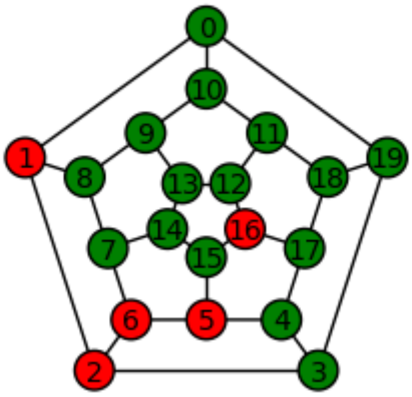
```
Stabilized?
    True
Step 0
```



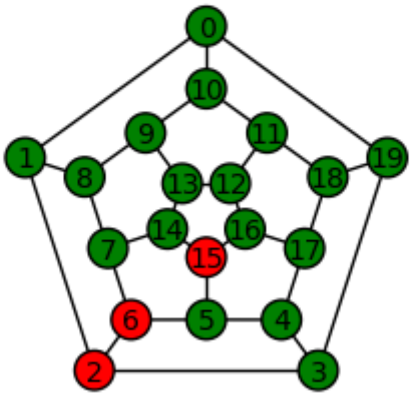

```
Step 1
```



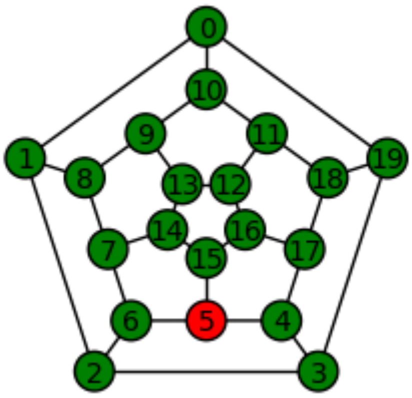
Step 2



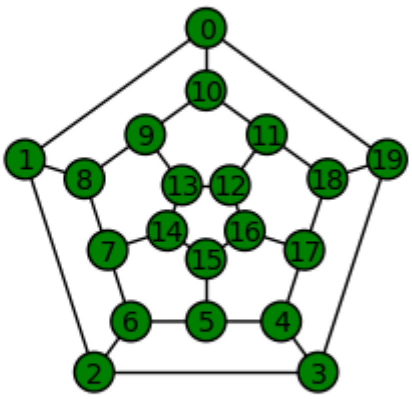

Step 3




Step 4

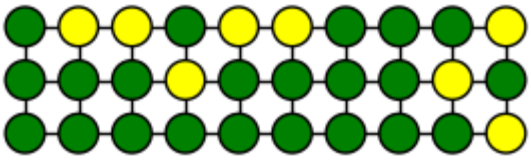



Step 5

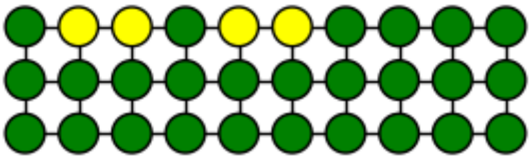


```
# Example: Run the GSL2 rule.

G = graphs.Grid2dGraph(3, 10)
color_bias = {'green': 0.6, 'yellow': 0.4}
ur = gsl2_rule
ur_kwargs = {'palette': color_bias.keys(), 'T': 0.7}
initial_coloring = color_randomly(G, color_bias)

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

```
Stabilized?
    True
Step 0
```

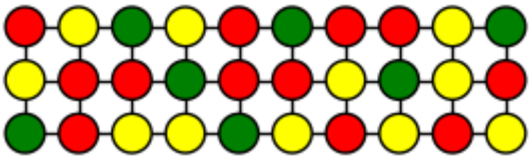


```
Step 1
```

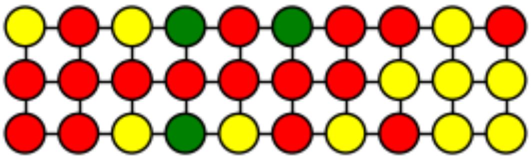


```
# Example: Run the GSL3 rule.

G = graphs.Grid2dGraph(3, 10)
color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
ur_kwargs = {'palette': color_bias.keys(), 'T': 0.6}
initial_coloring = color_randomly(G, color_bias)

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

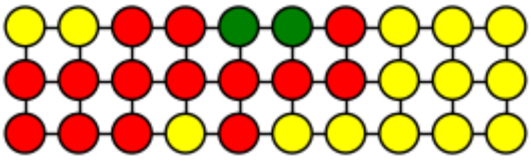
```
Stabilized?
    True
Step 0
```



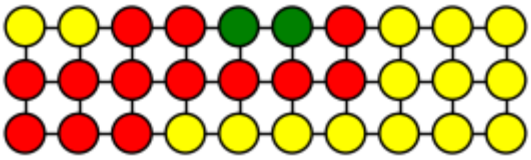
```
Step 1
```



Step 2

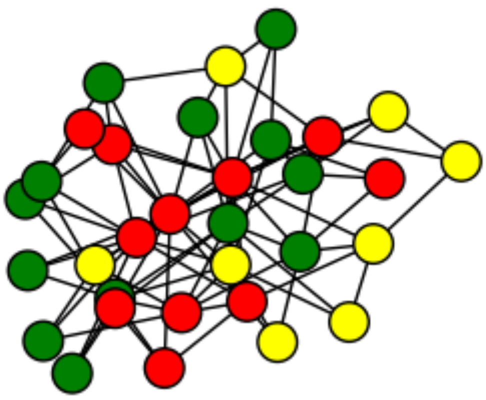


Step 3

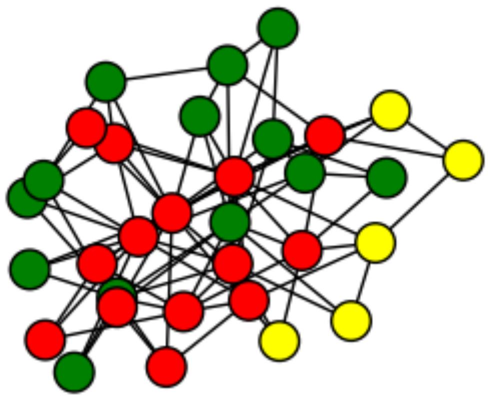


```
# Example: Run the GSL3 rule on a random graph

gg = graphs.RandomBarabasiAlbert
print(gg)
G = gg(32, 3)
color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
ur_kwargs = {'palette': color_bias.keys(), 'T': 0.6}
initial_coloring = color_randomly(G, color_bias)

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

```
<function RandomBarabasiAlbert at 0x119a0baa0>
Stabilized?
    True
Step 0
```



Step 1



```
# Example: Use run_rule_many_times() on one graph

color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
urk = {'palette': color_bias.keys()}
gg = moore_lattice
ggk = {'r': 8, 'c': 16, 'toroidal': True}
cf = color_randomly
cfk = {'bias': color_bias}

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

```
<function gsl3_rule at 0x11ae788c0>
<function moore_lattice at 0x11ae78938>
<function color_randomly at 0x11ae786e0>
--------------------
Number of runs: 1000
Number of runs that stabilized: 760
Mean number of steps required to stabilize: 6.38
Mean initial color counts:
    green: 42.3
    red: 43.4
    yellow: 42.3
Mean finial color counts:
    green: 43.8
    red: 40.5
    yellow: 43.7
```

```
# Example: Use run_rule_many_times() on many instances of a
random graph

color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
urk = {'palette': color_bias.keys(), 'T': 0.5, 't': 0.25, 's':
0.25}
gg = maslov_sneppen
ml = moore_lattice(r=8, c=16, toroidal=True)
```

```
ggk = {'graph': ml}  # Warning: Maslov-Sneppon does 4096 (=
4*num_edges) steps on this graph. Slow!
cf = color_randomly
cfk = {'bias': color_bias}

num_runs = 10
num_stabilized, mean_steps, mean_initial, mean_final =
run_rule_many_times(ur, urk, gg, ggk, cf, cfk,
num_runs=num_runs)
```

```
<function gsl3_rule at 0x11b0026e0>
<function maslov_sneppen at 0x11b002848>
<function color_randomly at 0x11b002500>
--------------------
Number of runs: 10
Number of runs that stabilized: 10
Mean number of steps required to stabilize: 5.5
Mean initial color counts:
    green: 43.9
    red: 42.2
    yellow: 41.9
Mean finial color counts:
    green: 50
    red: 39.7
    yellow: 38.3
```

```
# Exploring the random Barabasi Albert graph

G = graphs.RandomBarabasiAlbert(128, 4)
n = G.num_verts()
degrees = G.degree()
ave_degree = sum(degrees)/n

print('nvertices = {!s}'.format(n))
print('degrees = {!s}'.format(degrees))
print('ave degree = {:.3f}'.format(ave_degree))
```

```
nvertices = 128
degrees = [7, 7, 18, 36, 34, 34, 37, 25, 19, 18, 9, 12, 20, 2
15, 6, 12, 7, 9, 16, 12, 8, 6, 5, 6, 9, 10, 8, 10, 6, 9, 12,
7, 4, 5, 8, 6, 9, 8, 7, 5, 5, 5, 12, 7, 6, 7, 6, 11, 4, 7, 6,
7, 6, 6, 9, 4, 8, 5, 4, 6, 4, 5, 6, 4, 6, 5, 6, 8, 7, 7, 8, 5
6, 4, 4, 6, 6, 4, 4, 6, 4, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6
4, 4, 5, 4, 5, 4, 4, 4, 4, 4, 4, 4, 5, 4, 4, 4, 5, 4, 4, 4, 4
4, 4]
ave degree = 7.000
```

```
# Example: Use run_rule_many_times() on many instances of a
random graph

color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
urk = {'palette': color_bias.keys(), 'T': 0.5, 't': 0.25, 's':
0.25}
```

```
gg = graphs.RandomBarabasiAlbert
ggk = {'n': 128, 'm': 4}
cf = color_randomly
cfk = {'bias': color_bias}

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

```
<function gsl3_rule at 0x119955500>
<function RandomBarabasiAlbert at 0x119a0baa0>
<function color_randomly at 0x119955320>
--------------------
Number of runs: 1000
Number of runs that stabilized: 907
Mean number of steps required to stabilize: 4.97
Mean initial color counts:
    green: 42.6
    red: 42.6
    yellow: 42.7
Mean finial color counts:
    green: 44.9
    red: 38.7
    yellow: 44.4
```