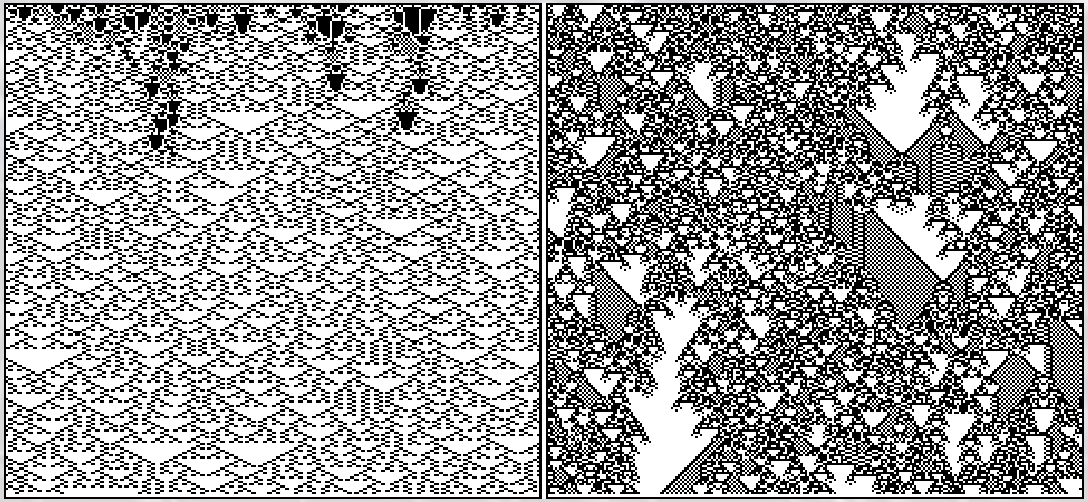
A.I.: Genetic Algorithms



Some Examples of Biologically Inspired AI

- Neural networks
- Evolutionary computation (e.g., genetic algorithms)
- Immune-system-inspired computer/network security
- Insect-colony optimization (ants, bees, etc.)
- Slime-mould path-finding
- Swarm intelligence (e.g., decentralized robots)

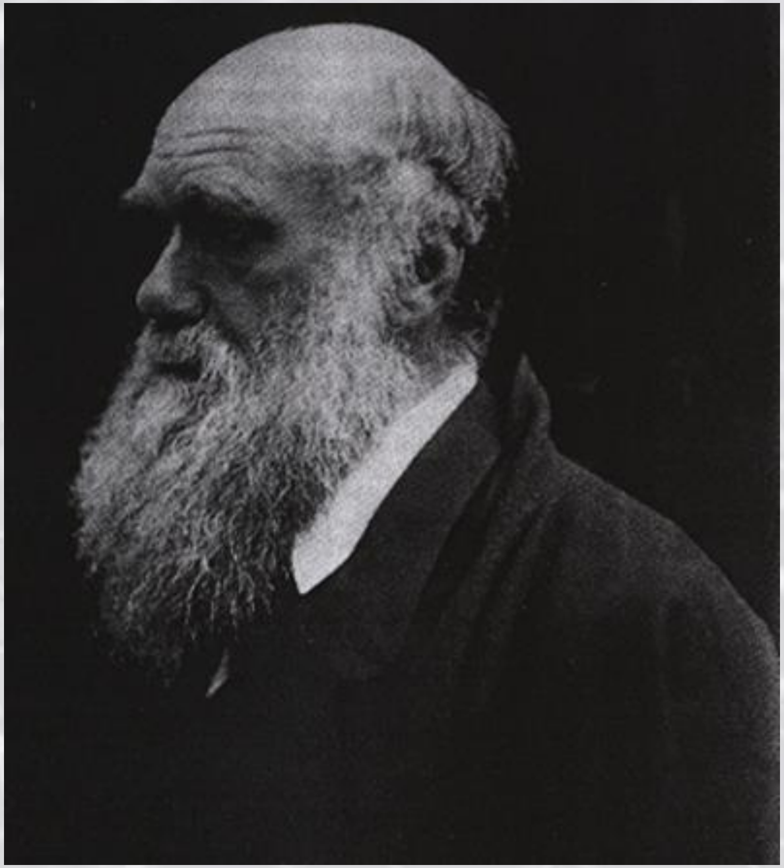
Evolutionary Computation

A collection of computational methods inspired by biological evolution:

- A population of candidate solutions evolves over time, with the fittest at each generation contributing the most offspring to the next generation

- Offspring are produced via crossover between parents, along with random mutations and other “genetic” operations.

Evolution made simple



Charles Darwin
1809–1882

Essentials of Darwinian evolution:

- Organisms reproduce in proportion to their *fitness* in the environment
- Offspring inherit traits from parents
- Traits are inherited with some variation, via mutation and sexual recombination

Evolution made simple

Essentials of evolutionary algorithms:

- Computer “organisms” (e.g., programs) reproduce in proportion to their *fitness* in the environment (e.g., how well they perform a desired task)
- Offspring inherit traits from their parents
- Traits are inherited, with some variation, via mutation and “sexual recombination”

Essentials of Darwinian evolution:

- Organisms reproduce in proportion to their *fitness* in the environment
- Offspring inherit traits from parents
- Traits are inherited with some variation, via mutation and sexual recombination

Appeal of ideas from evolution:

- Successful method of searching large spaces for good solutions (chromosomes / organisms)
- Massive parallelism
- Adaptation to environments, change
- Emergent complexity from simple rules

Genetic Algorithms

Components of a GA:

- *Population* of candidate solutions to a given problem ("chromosomes")
- *Fitness function* that assigns fitness to each chromosome in the population
- *Selection procedure* that selects individuals to reproduce
- *Genetic operators* that take existing chromosomes and produce offspring with variation (e.g., mutation, crossover)

A Simple Genetic Algorithm

1. Start out with a randomly generated population of chromosomes (candidate solutions).

2. Calculate the fitness of each chromosome in the population.

3. Select pairs of parents with probability a function of fitness rank in the population.

4. Create new population: Cross over parents, mutate offspring, place in new population.

5. Go to step 2.

Genetic operators

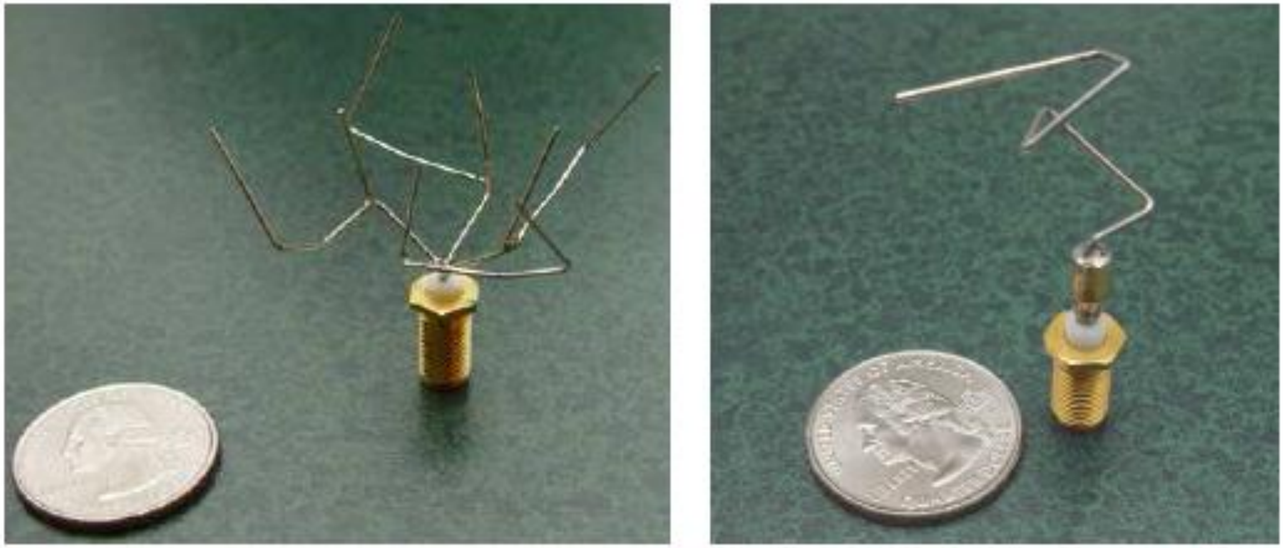
- *Crossover:* exchange subparts of two chromosomes:

0 0 0 0 0 1 0 0 0 1 → 0 0 0 1 1 1 1 1

0 0 1 1 1 1 1 1 1 0 → 0 0 1 0 0 1 0 0

- *Mutation:* randomly change some loci:

From Hornby et al., 2006

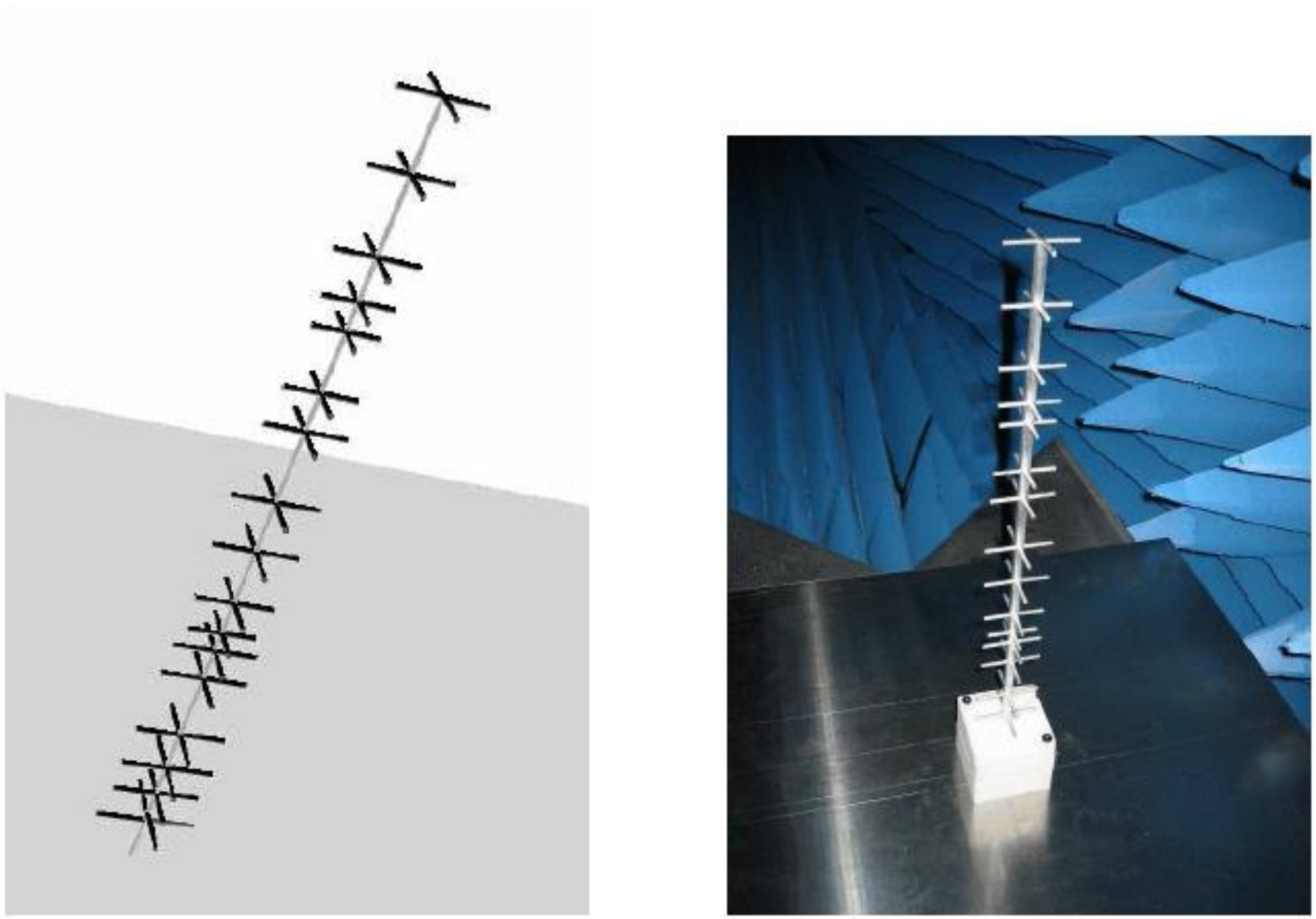


(a) (b)

Figure 2. Photographs of prototype evolved antennas: (a) the best evolved antenna for the initial gain pattern requirement, ST5-3-10; (b) the best evolved antenna for the revised specifications, ST5-33-142-7.

Evolvable hardware work at NASA Ames
(Hornby, Lohn, et al.)

From Hornby et al., 2006



(a) (b)

Figure 4. Best evolved TDRS-C antenna: (a) simulation and (b) fabricated.

Example: Evolving Strategies for Robby the Robot

Sensors:

N,S,E,W,C(urrent)

Actions:

Move N

Move S

Move E

Move W

Move random

Stay put

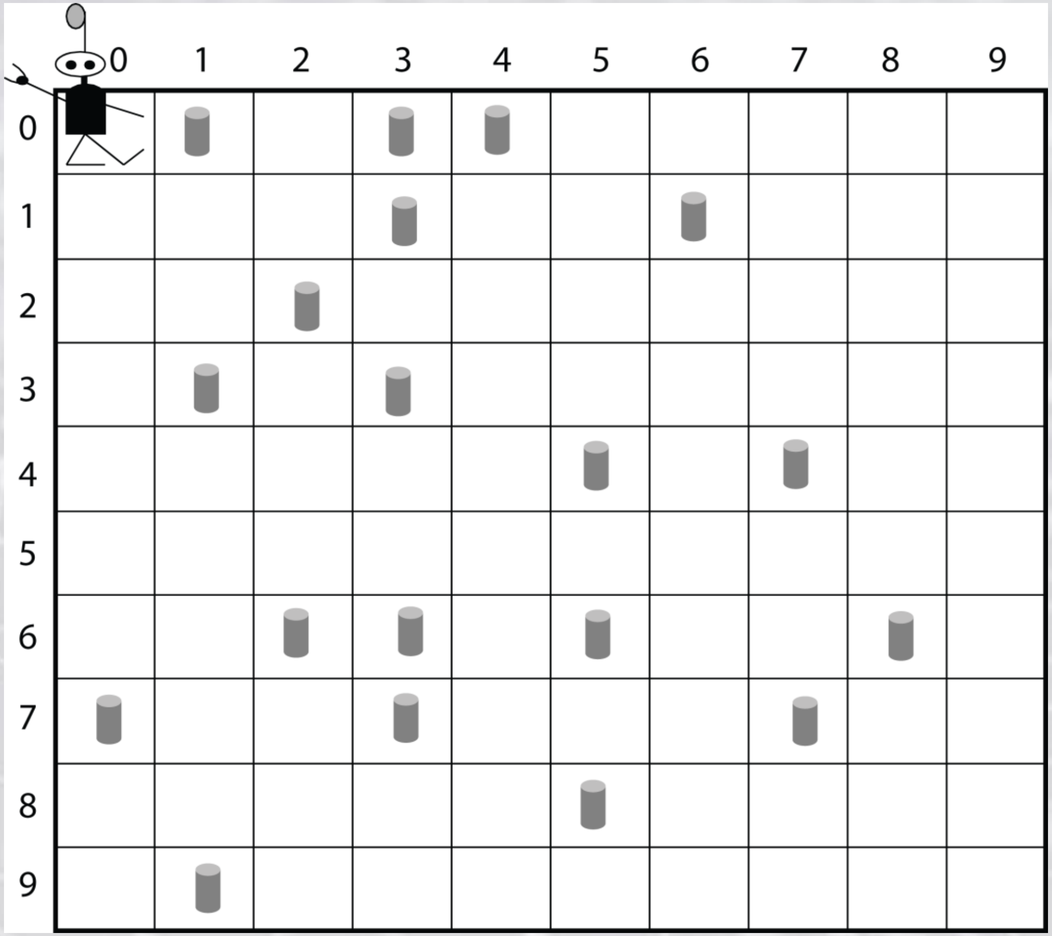
Try to pick up can

Rewards/Penalties (points):

Picks up can: 10

Tries to pick up can on empty site: -1

Crashes into wall: -5



Robby's fitness function

```
Calculate_Fitness (Robby) {
  Total_Reward = 0 ;
  Average_Reward = 0 `
  For i = 1 to NUM_ENVIRONMENTS {
     generate_random_environment( ); /* .5 probability
                                      * to place can at
                               * each site */
     For j = 1 to NUM_MOVES_PER_ENVIRONMENT {
       Total_Reward = Total_Reward + perform_action(Robby);
     }
  }

  Fitness = Total_Reward / NUM_ENVIRONMENTS;
  return(Fitness);
}
```

Genetic algorithm for evolving strategies for Robby

1. Generate 200 random strategies (i.e., programs for controlling Robby)

Random Initial Population

Individual 1:
23300323421630343530546006102562515114162260435654334066511514
15650220640642051006643216161521652022364433363346013326503000
40622050243165006111305146664232401245633345524126143441361020
150630642551654043264463156164510543665346310551646005164

Individual 2:
16411343121025360340361241431201104235462525304202044516433665
61035322153105131440622120614631432154610256523644422025340345
30502005620634026331002453416430151631210012214400664012665246
351650154123113132453304433212634555005314213064423311000

Individual 3:
20423344402411226132136452632464212206122122252660626144436125
32512664061335340153411110206164226653145522540234051155031302
22020065445125062206631426135532010000400031640130154160162006
134440626160505641421553133236021503355131253632642630551

.

.

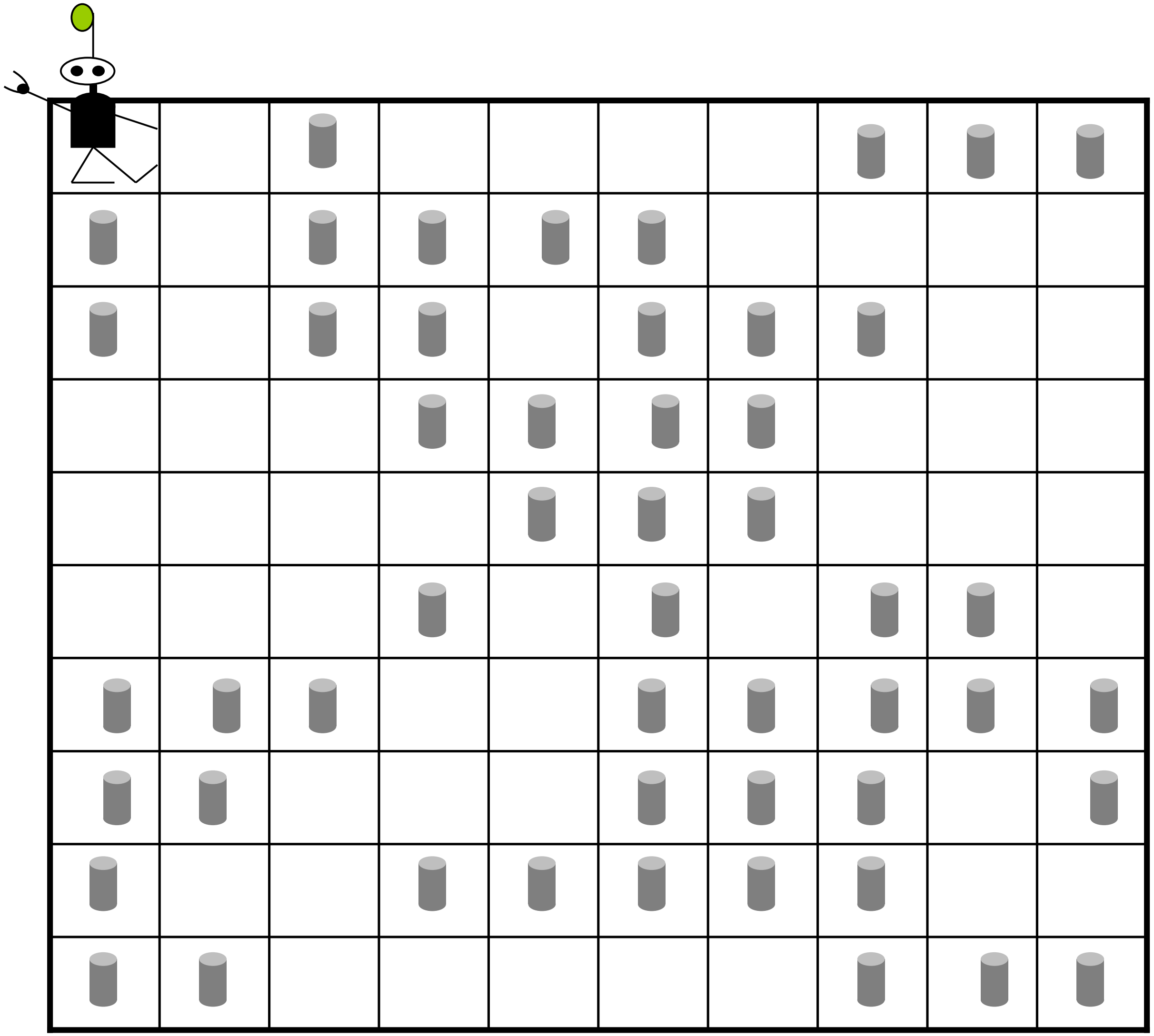
.

Individual 200:
34632525136001012225612106043301135205155320130656005322235043
32425064124255265534635345523053326612010632124554423440613654
30246240160663016464641103026540006334126150352262106063624260
550616616344255124354464110023463330440102533212142402251

Genetic algorithm for evolving strategies for Robby

1. Generate 200 random strategies (i.e., programs for controlling Robby)

2. For each strategy in the population, calculate fitness (average reward minus penalties earned on random environments)



Fitness =
Average final score from N moves on each of M random environments

Genetic algorithm for evolving strategies for Robby

1. Generate 200 random strategies (i.e., programs for controlling Robby)

2. For each strategy in the population, calculate fitness (average reward minus penalties earned on random environments)

3. Strategies are selected according to fitness to become parents. (See code for choice of selection methods.)

Genetic algorithm for evolving strategies for Robby

1. Generate 200 random strategies (i.e., programs for controlling Robby)

2. For each strategy in the population, calculate fitness (average reward minus penalties earned on random environments)

3. Strategies are selected according to fitness to become parents. (See code for choice of selection methods.)

4. The parents pair up and create offspring via crossover with random mutations.

Parent 1:

1641134312102536034036124143120110423546252530420204451643366５
6103532215310513144062212061463143215461025652364442202534034５
3050200562063402633100245341643015163121001221440066401266524６
351650154123113132453304433212634555005314213064423311000

Parent 2:

2042334440241122613213645263246421220612212225266062614443612５
3251266406133534015341111020616422665314552254023405115503130２
2202006544512506220663142613553201000040003164013015416016200６
13444062616050564142155313323602150335513125363264263055１

Parent 1:

1641134312102536034036124143120110423546252530420204451643366
5610353221531051314406221206146314321546102565236444220253403
45305020056206340263310024534164301516312100122144006640126652
46351650154123113132453304433212634555005314213064423311000

Parent 2:

2042334440241122613213645263246421220612212225266062614443612
5325126640613353401534111102061642266531455225402340511550313
022202006544512506220663142613553201000040003164013015416016200
61344406261605056414215531332360215033551312536326426305 51

Parent 1:

16411343121025360340361241431201104235462525304202044516433665
61035322153105131440622120614631432154610256523644422025340345
30502005620634026331002453416430151631210012214400664012665246
351650154123113132453304433212634555005314213064423311000

Parent 2:

20423344402411226132136452632464212206122122252660626144436125
32512664061335340153411110206164226653145522540234051155031302
22020065445125062206631426135532010000400031640130154160162006
134440626160505641421553133236021503355131253632642630551

Child:

16411343121025360340361241431201104235462525304202044516433665
61035322153105131440622120614631432154610256523644422025340345
30502005620634026331002453 6135532010000400031640130154160162006
134440626160505641421553133236021503355131253632642630551

Parent 1:

16411343121025360340361241431201104235462525304202044516433665
61035322153105131440622120614631432154610256523644422025340345
30502005620634026331002453416430151631210012214400664012665246
351650154123113132453304433212634555005314213064423311000

Parent 2:

20423344402411226132136452632464212206122122252660626144436125
32512664061335340153411110206164226653145522540234051155031302
22020065445125062206631426135532010000400031640130154160162006
134440626160505641421553133236021503355131253632642630551

Child:

Mutate to “0”

16411343121025360340361241431201104235462525304202044516433665
61035322153105131440622120614631432154610256523644422025340345
3050200562063402633100245 6135532010000400031640130154160162006
134440626160505641421553133236021503355131253632642630551

Mutate to “4”

Genetic algorithm for evolving strategies for Robby

1. Generate 200 random strategies (i.e., programs for controlling Robby)

2. For each strategy in the population, calculate fitness (average reward minus penalties earned on random environments)

3. Strategies are selected according to fitness to become parents. (See code for choice of selection methods.)

4. The parents pair up and create offspring via crossover with random mutations.

5. The offspring are placed in the new population and the old population dies.

Genetic algorithm for evolving strategies for Robby

1. Generate 200 random strategies (i.e., programs for controlling Robby)

2. For each strategy in the population, calculate fitness (average reward minus penalties earned on random environments)

3. Strategies are selected according to fitness to become parents. (See code for choice of selection methods.)

4. The parents pair up and create offspring via crossover with random mutations.

5. The offspring are placed in the new population and the old population dies.

6. Keep going back to step 2 until a good-enough strategy is

My hand-designed strategy:

“If there is a can in the current site, pick it up.”

“Otherwise, if there is a can in one of the adjacent sites, move to that site.”

“Otherwise, choose a random direction to move in.”

My hand-designed strategy:

"If there is a can in the current site, pick it up."

"Otherwise, if there is a can in one of the adjacent sites, move to that site."

"Otherwise, choose a random direction to move in."

Average fitness of this strategy: **346**
(out of max possible ≈ 500)

My hand-designed strategy:

"If there is a can in the current site, pick it up."

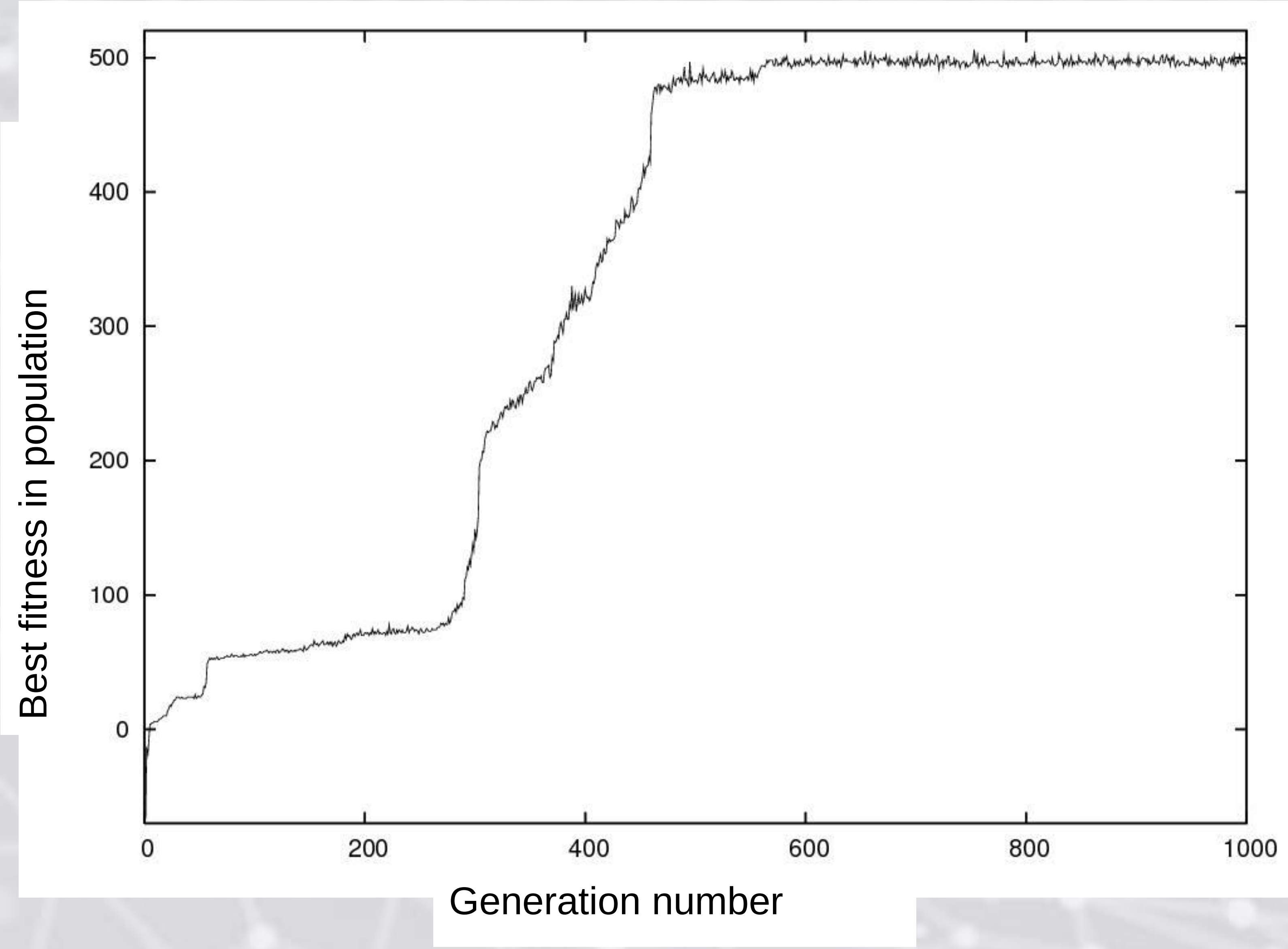
"Otherwise, if there is a can in one of the adjacent sites, move to that site."

"Otherwise, choose a random direction to move in."

Average fitness of this strategy: **346**
(out of max possible ≈ 500)

Average fitness of GA evolved strategy: **486**
(out of max possible ≈ 500)

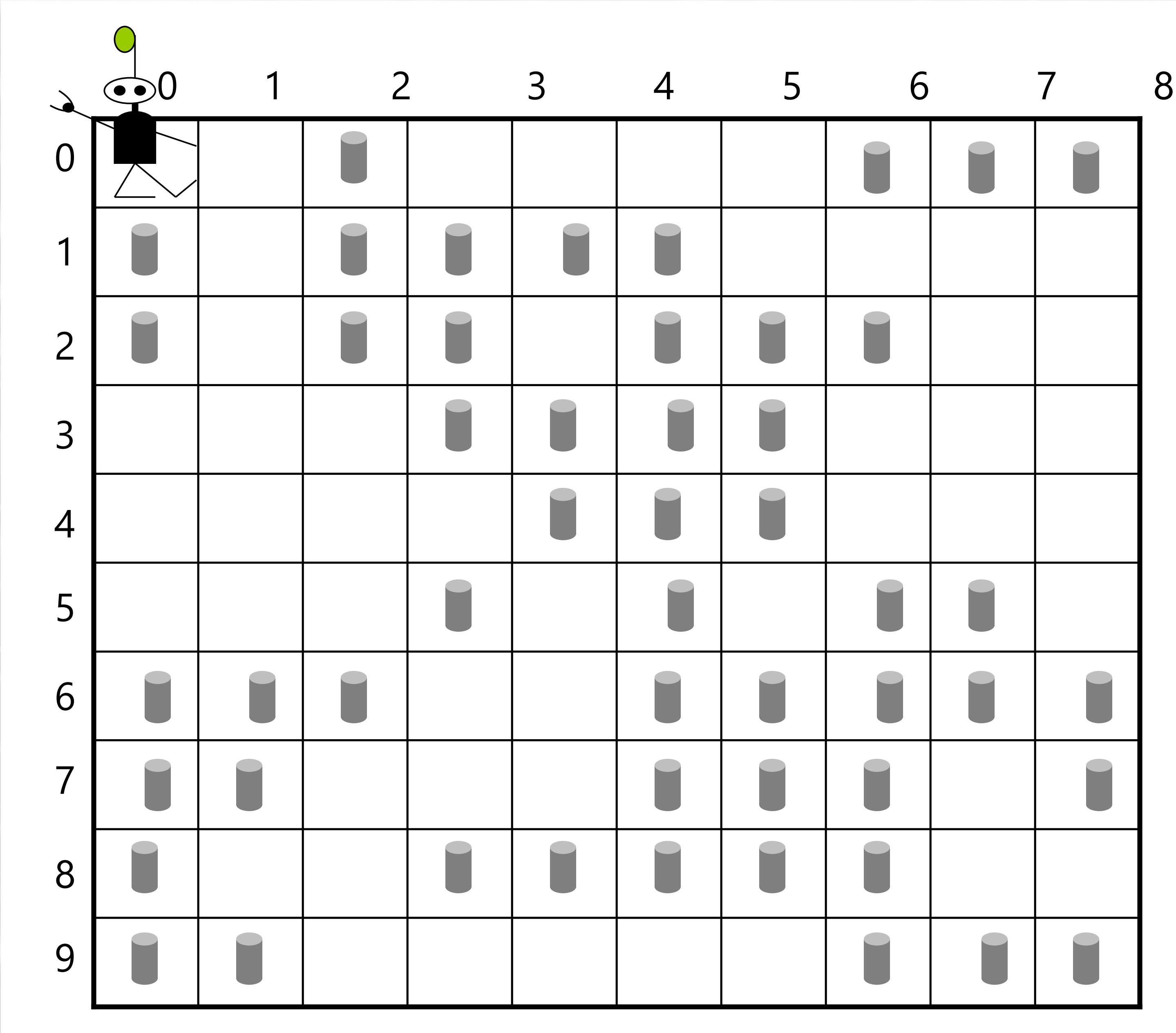
One Run of the Genetic Algorithm



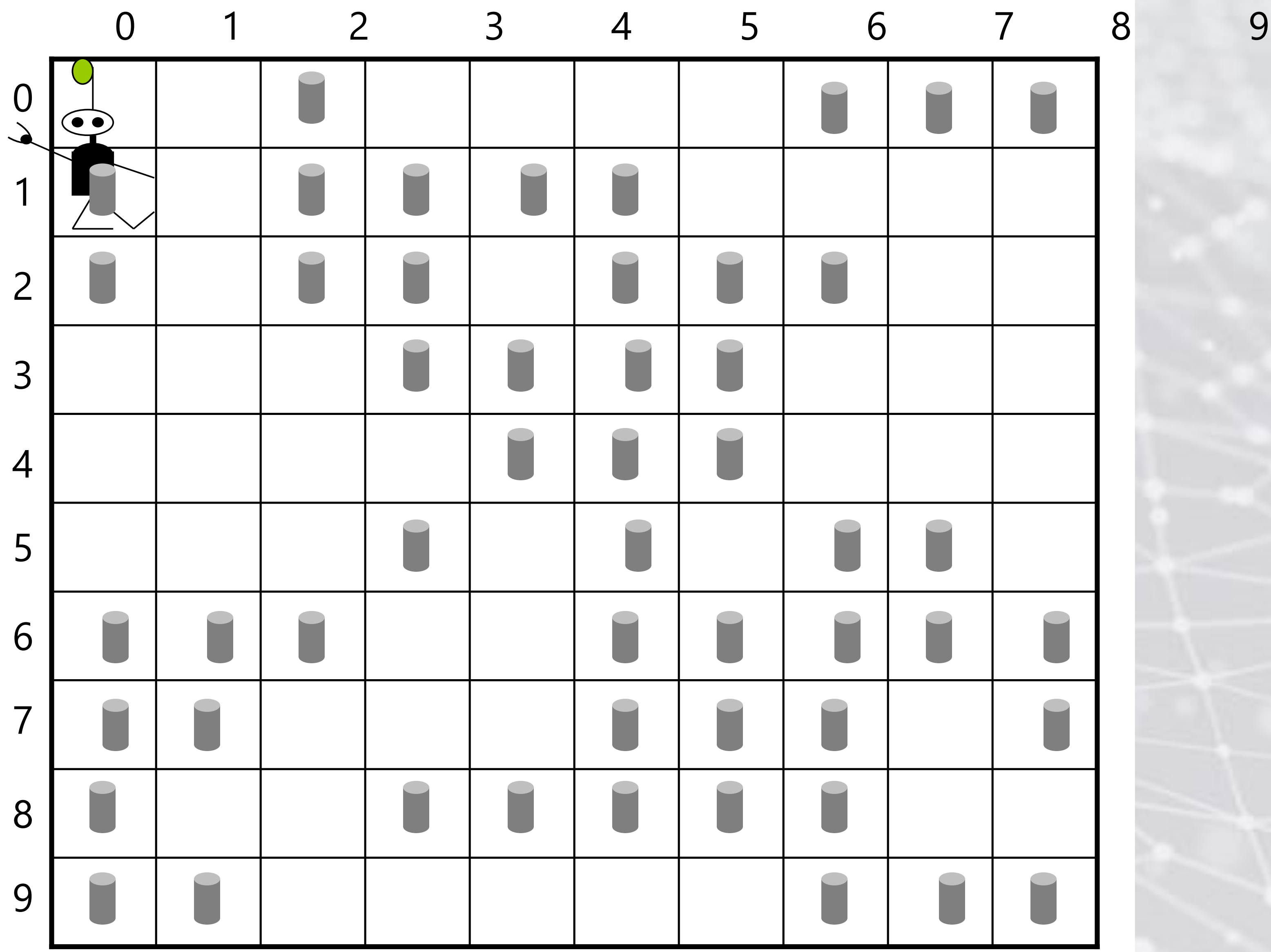
Generation 1

Best fitness = -81

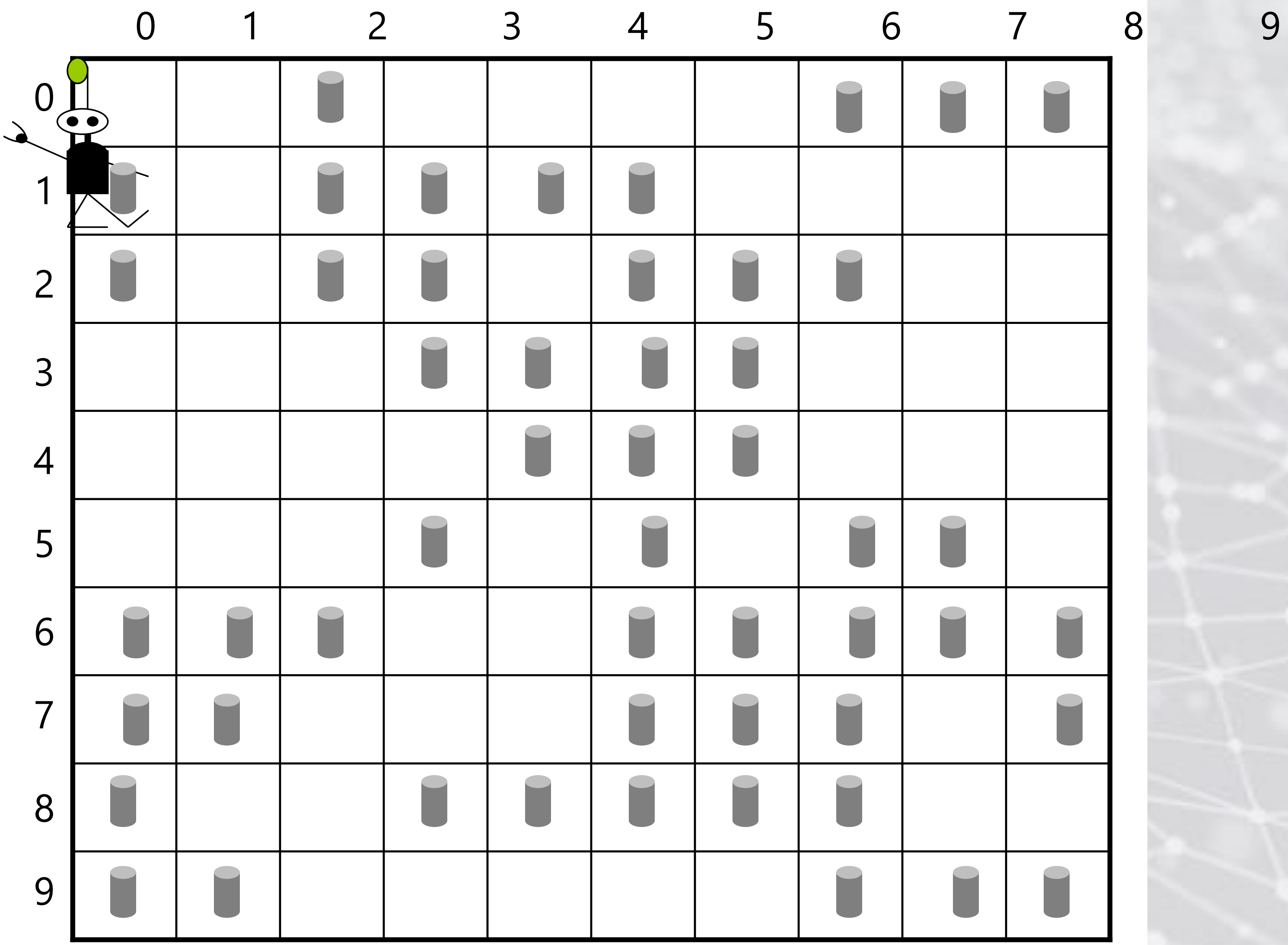
Time: 0 Score: 0



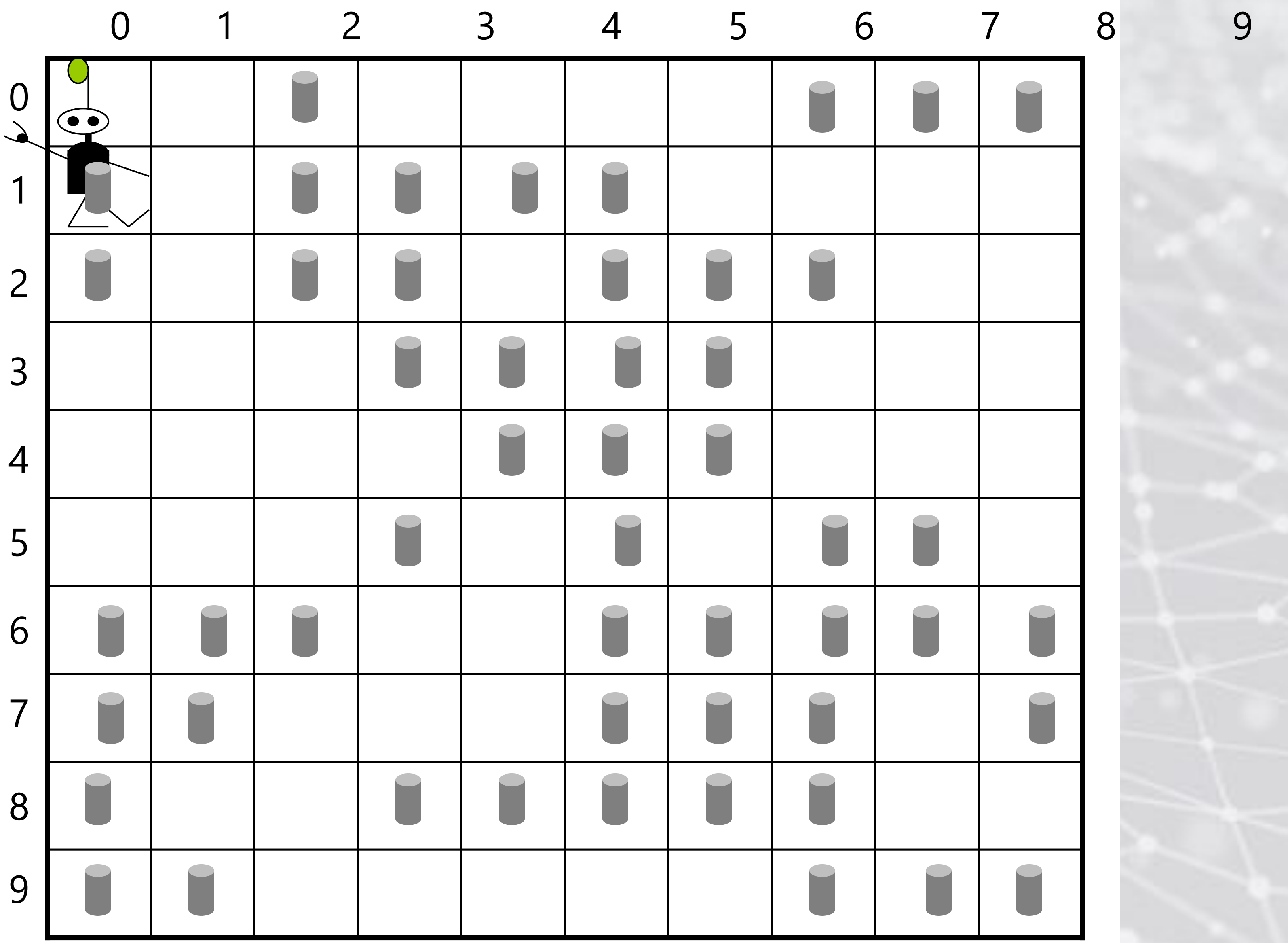
Time: 1 **Score: 0**



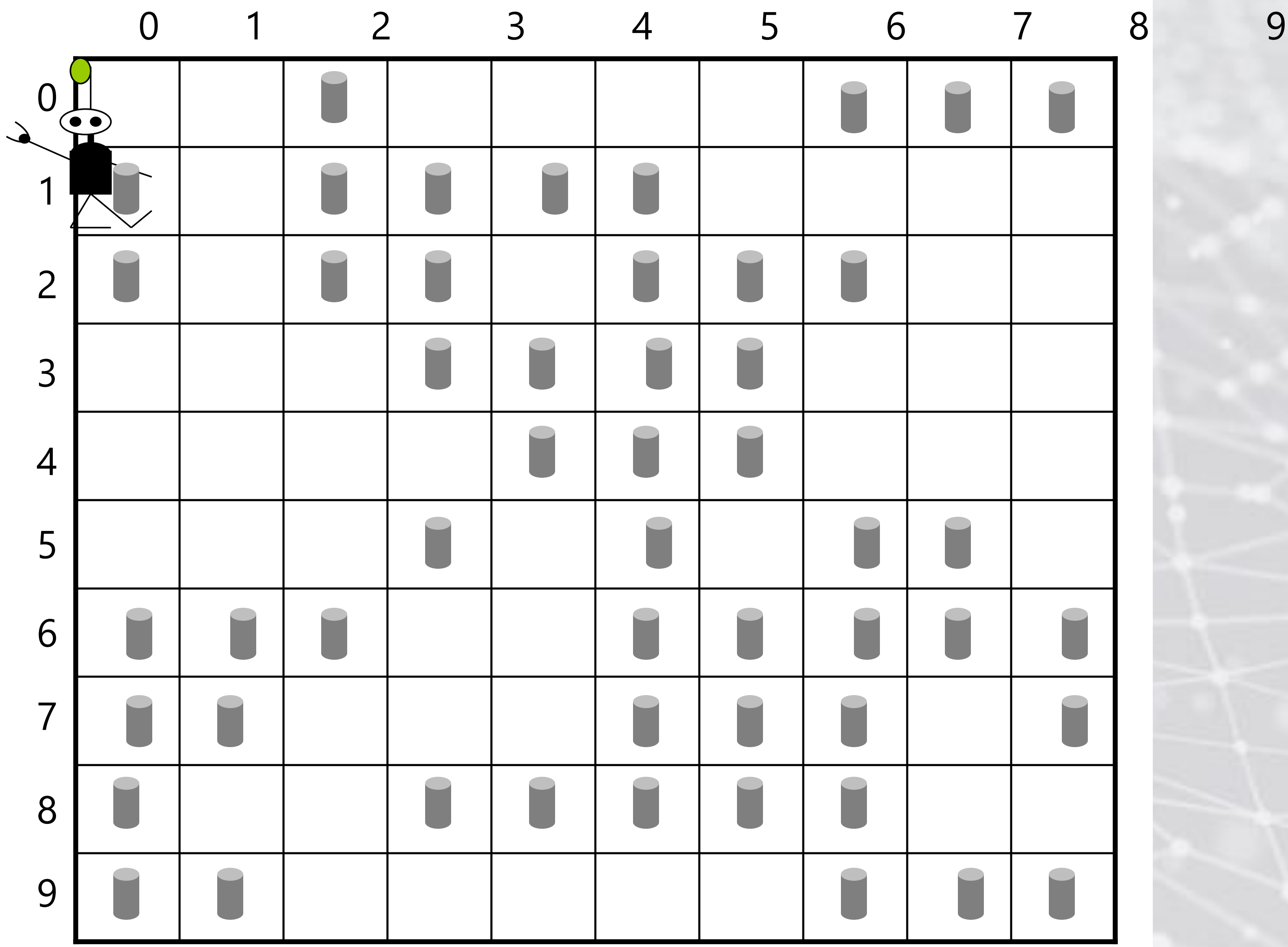
Time: 2 **Score: –5**



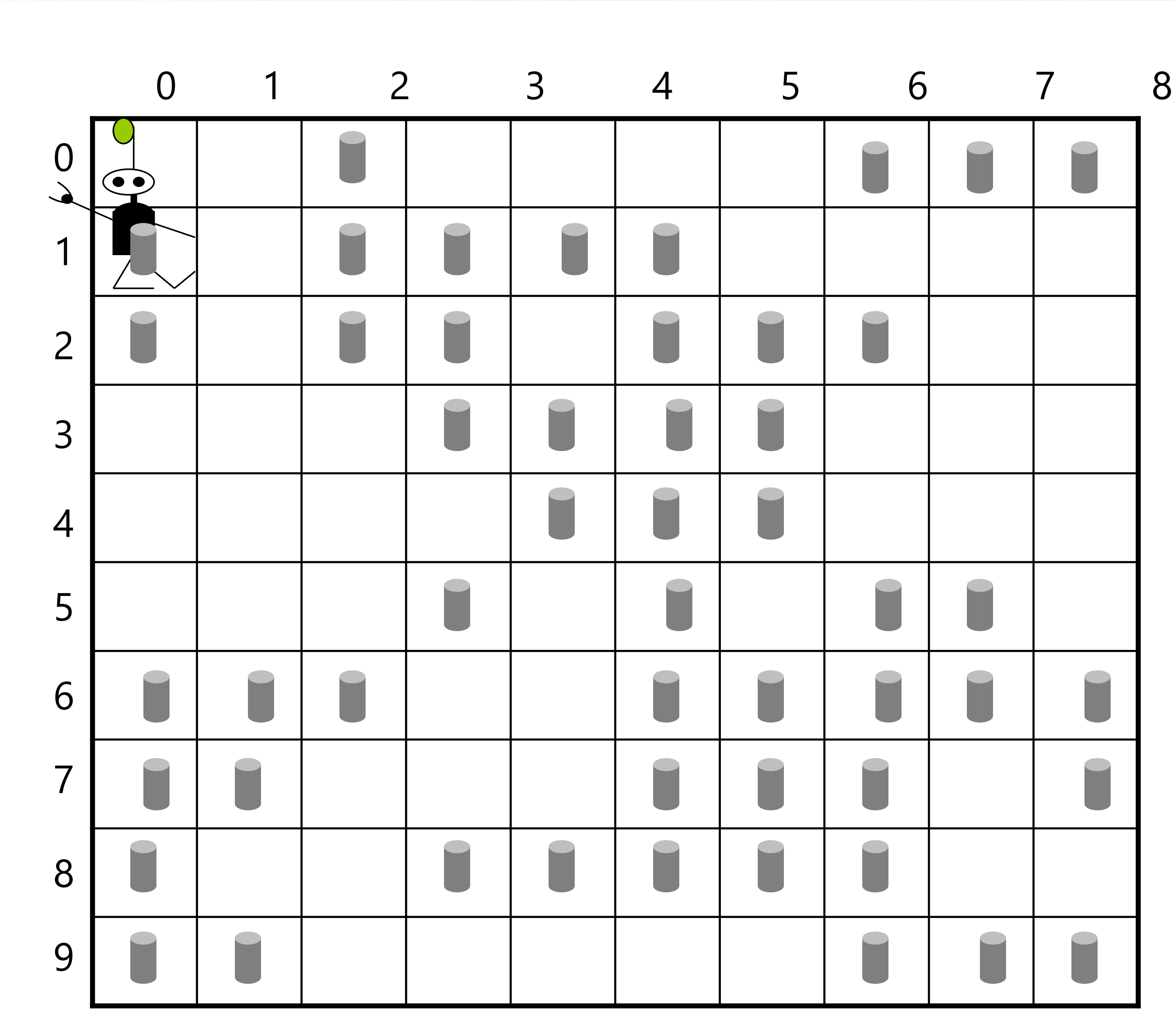
Time: 2 **Score: –5**



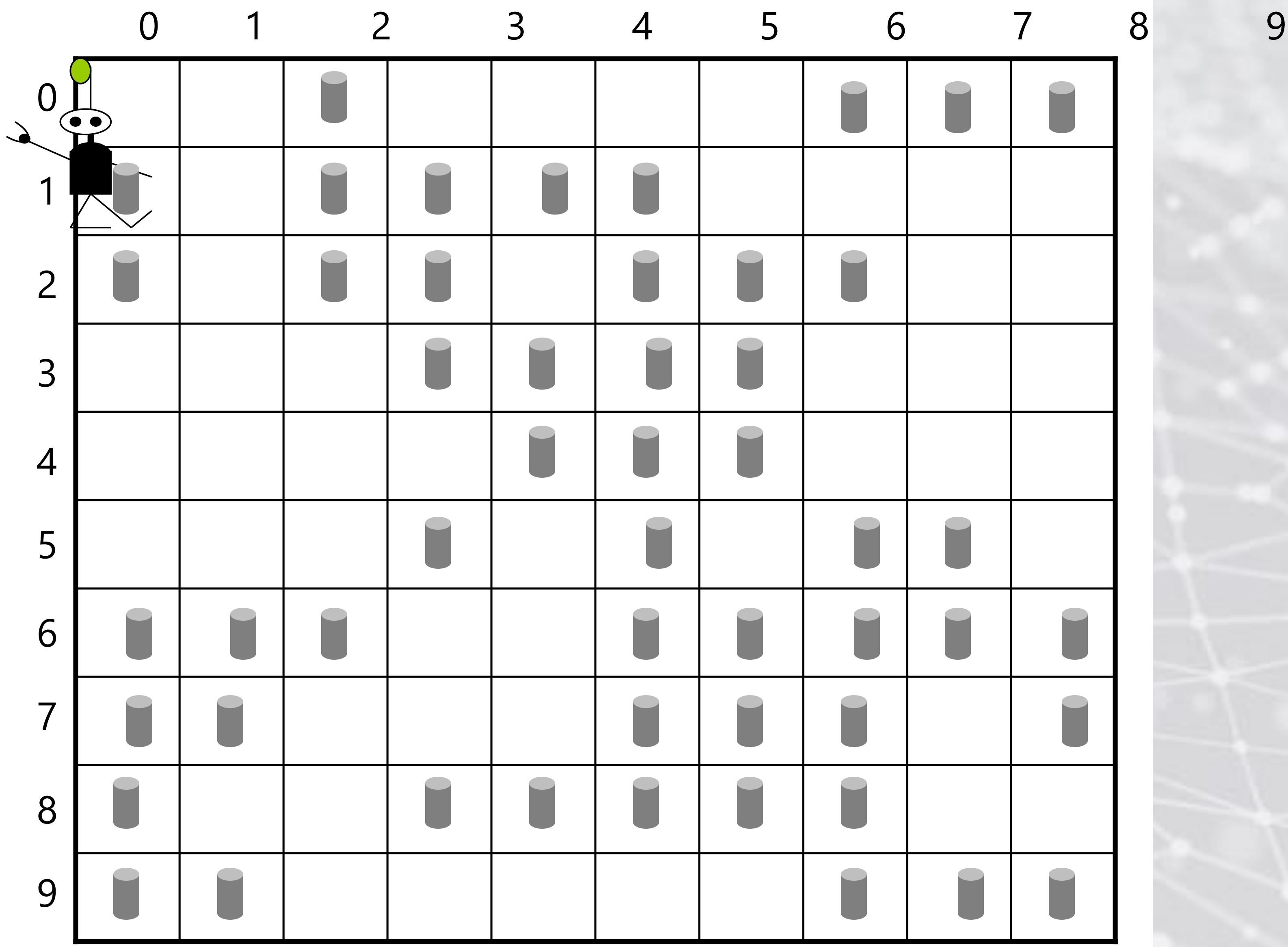
Time: 3 Score: –10



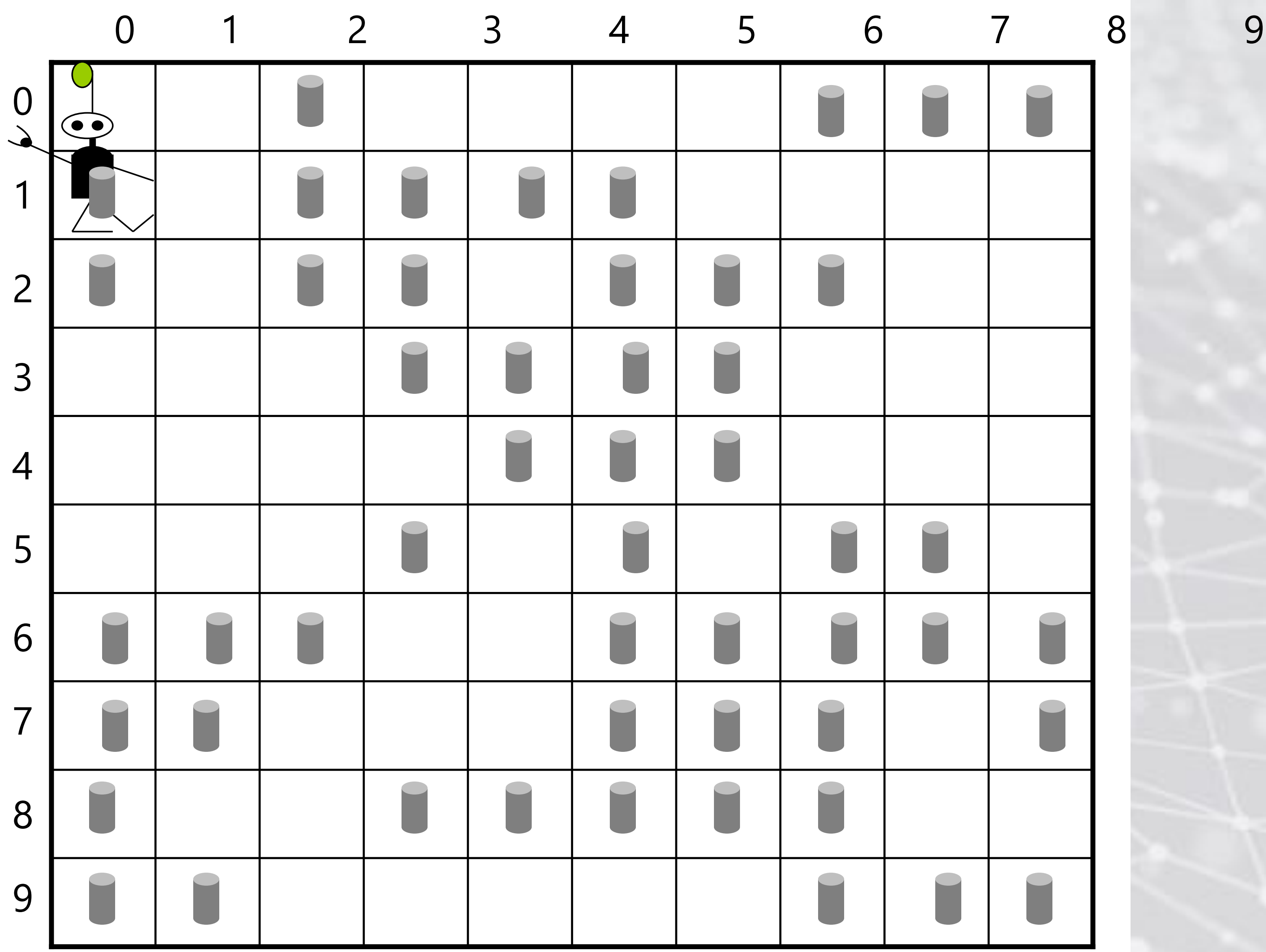
Time: 3   Score: −10



Time: 4 Score: –15



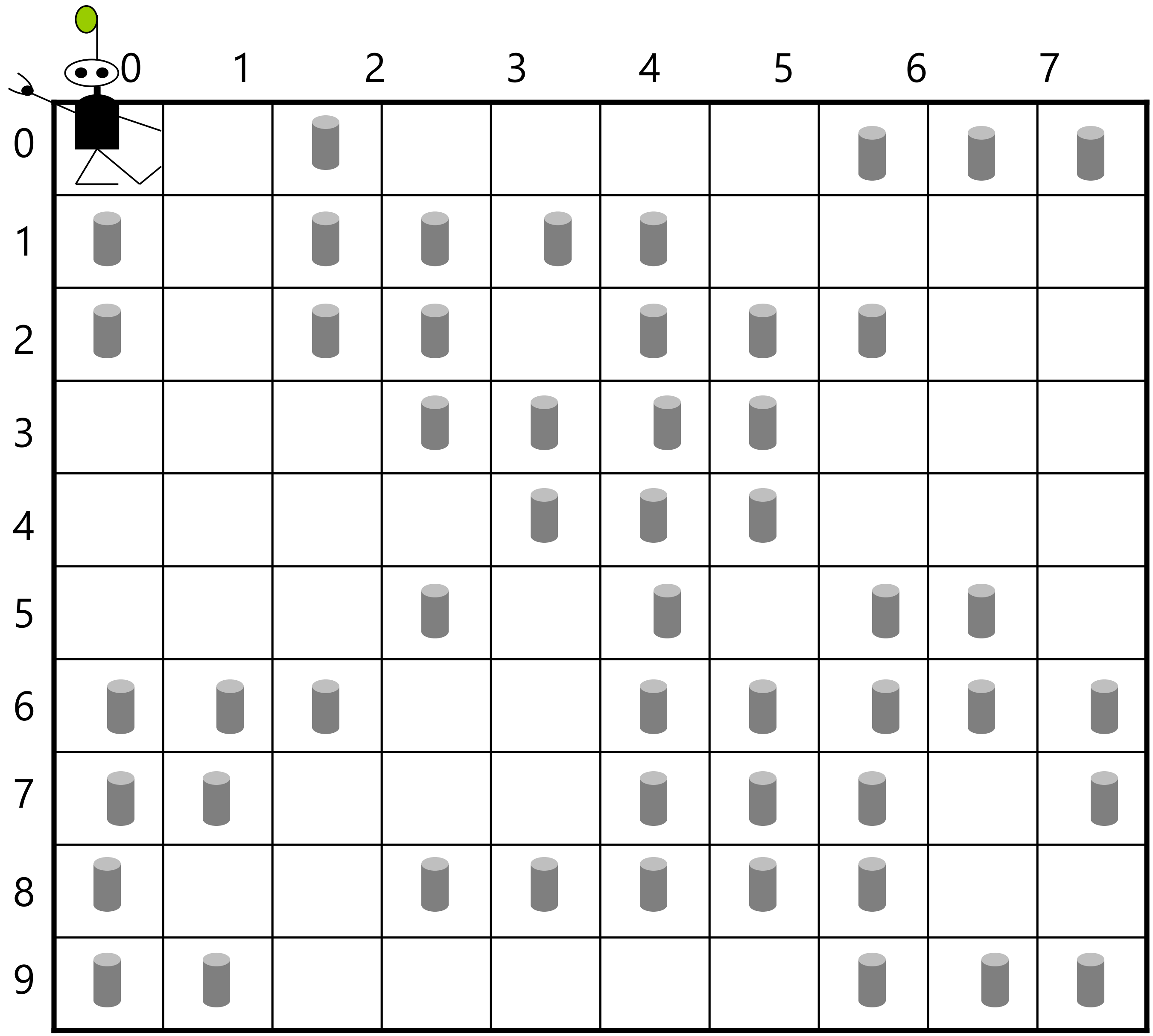
Time: 4   Score: –15



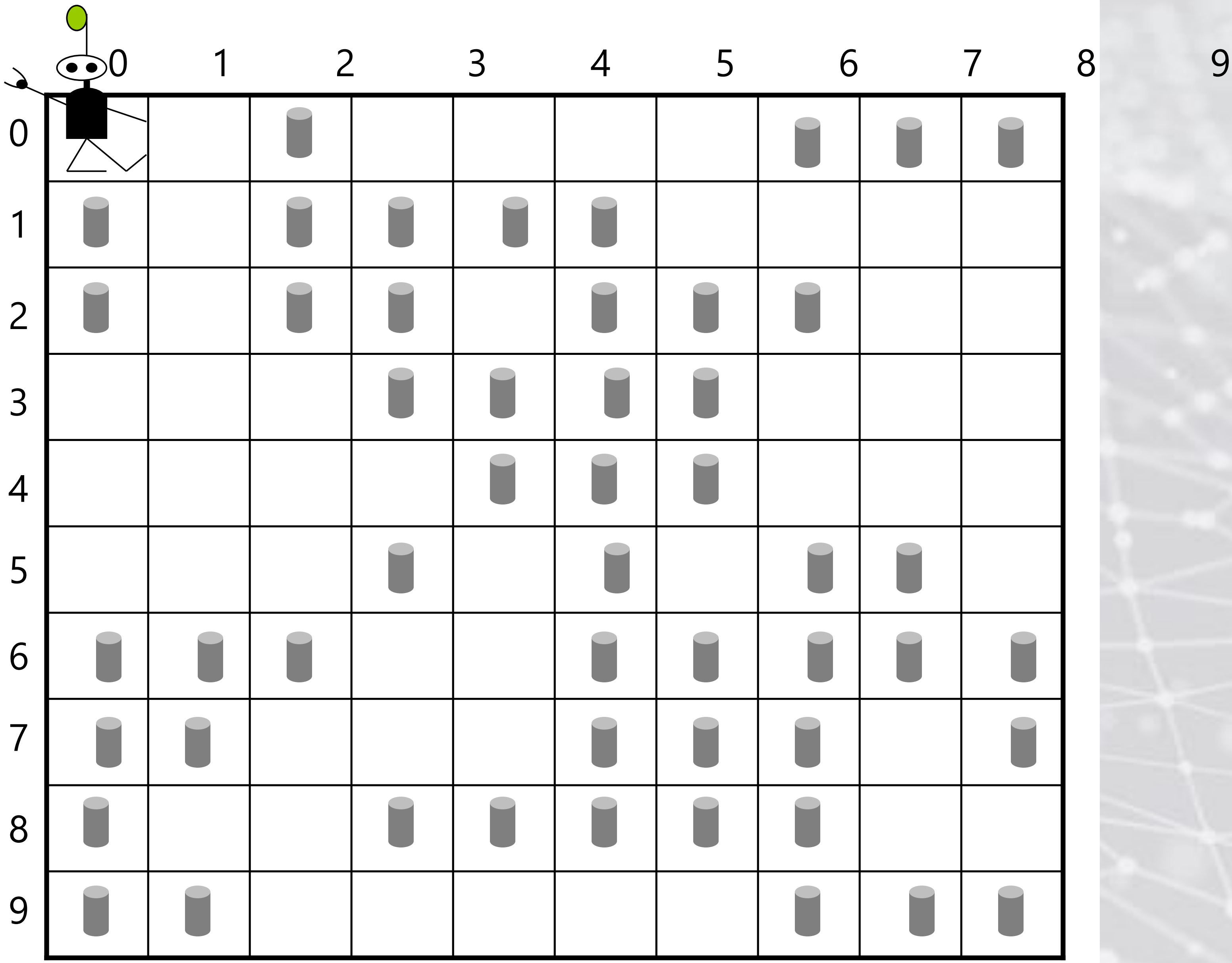
Generation 14

Best fitness = 1

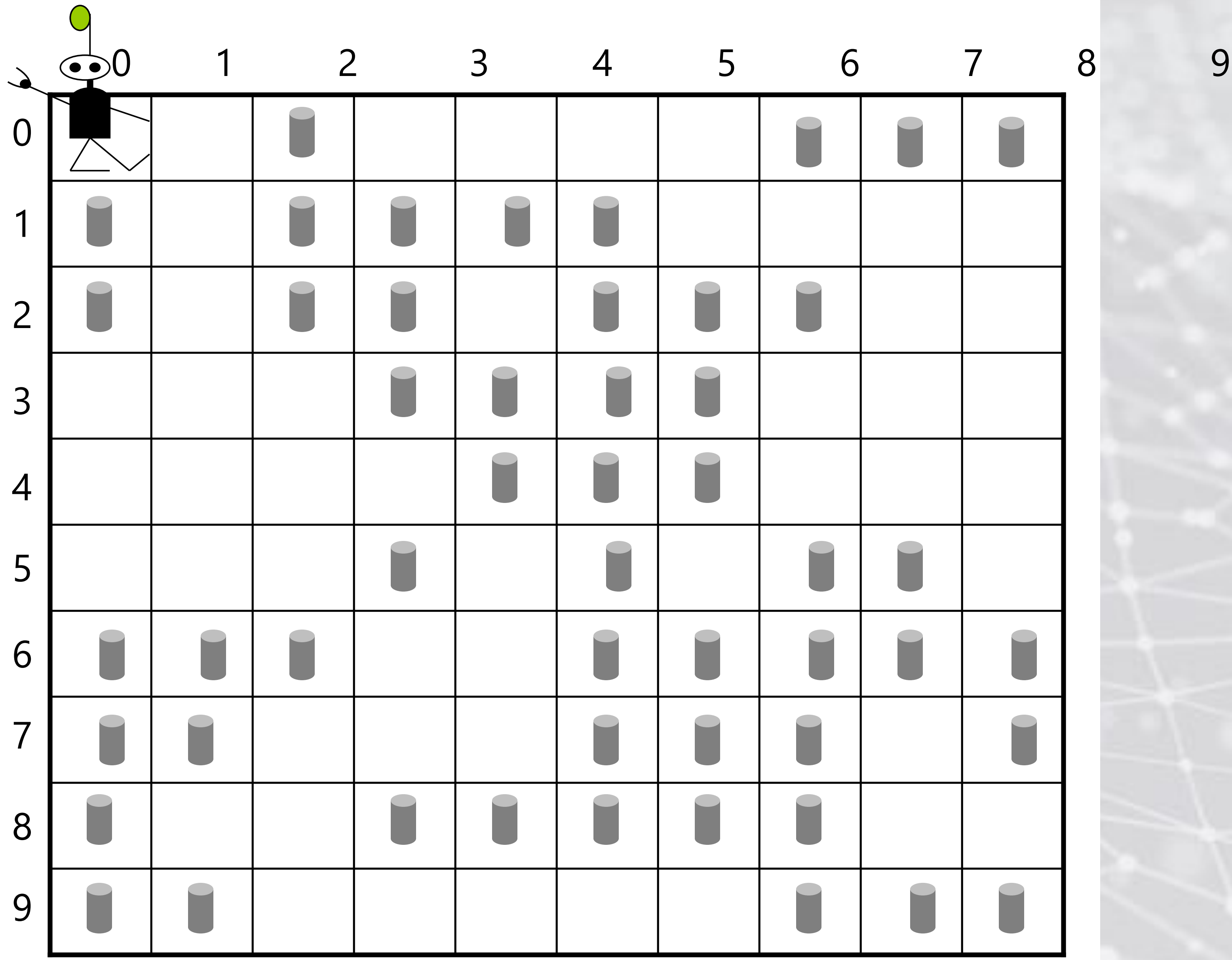
Time: 0 **Score: 0**



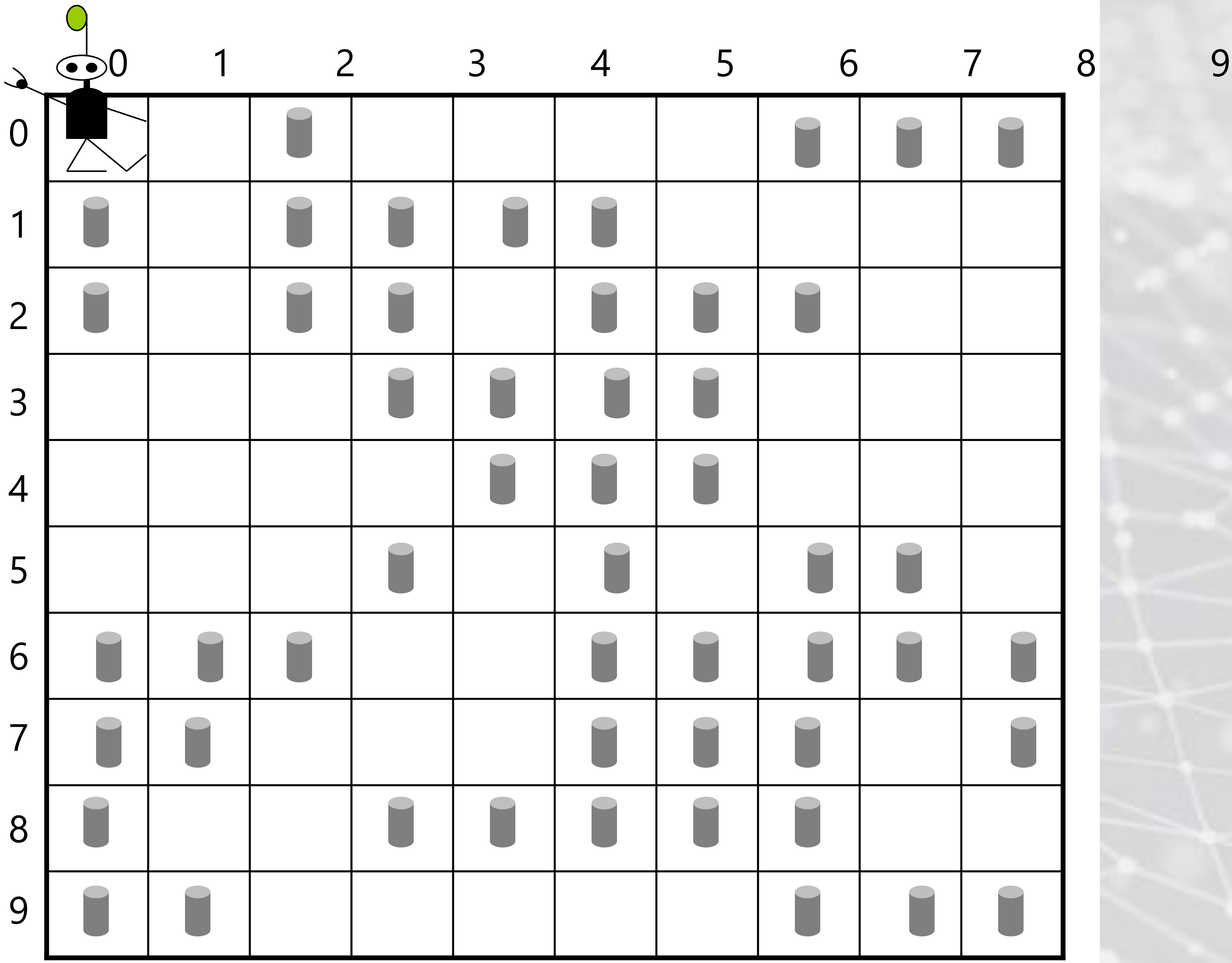
Time: 1 **Score: 0**



Time: 2 **Score: 0**



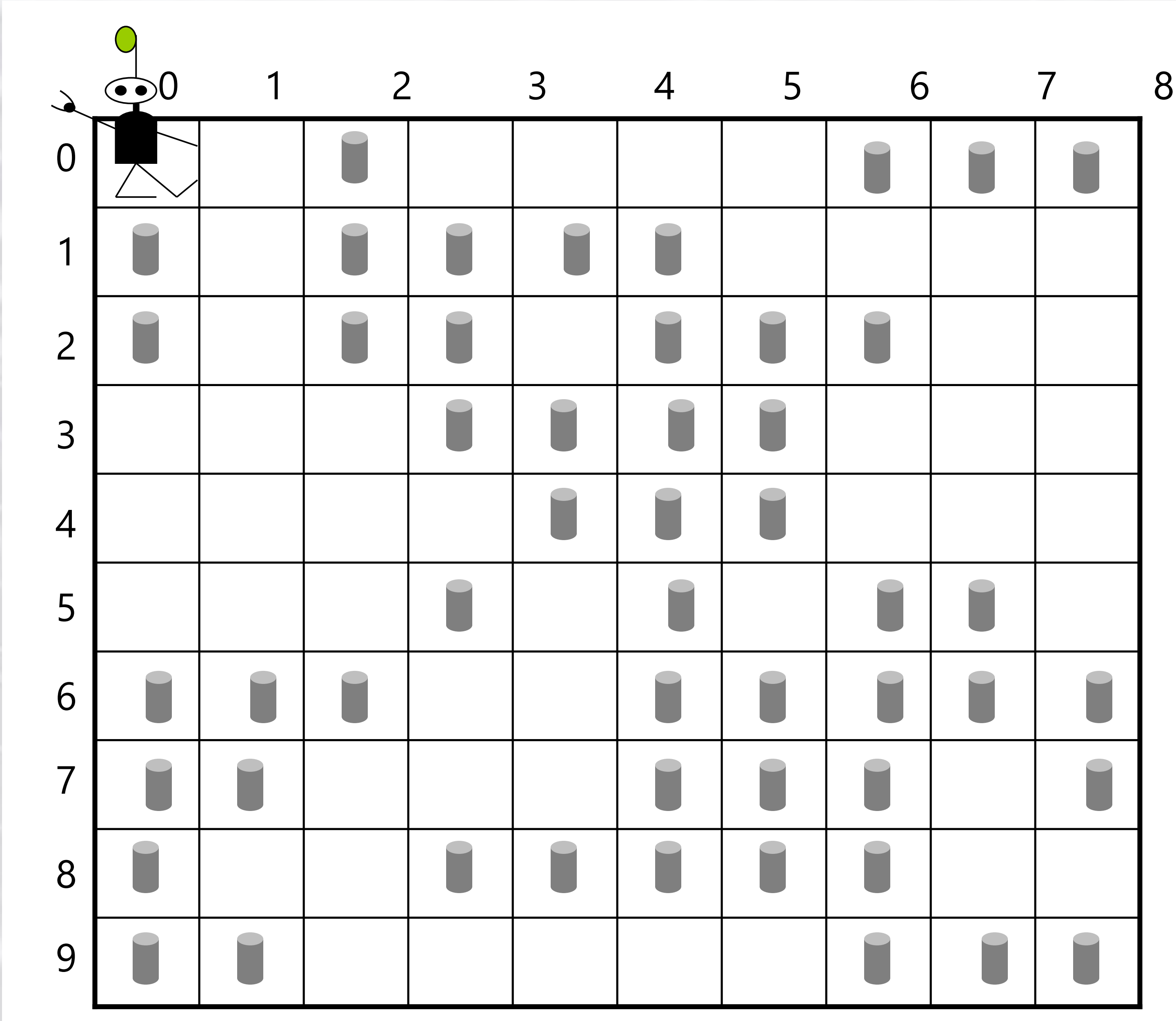
Time: 3 Score: 0



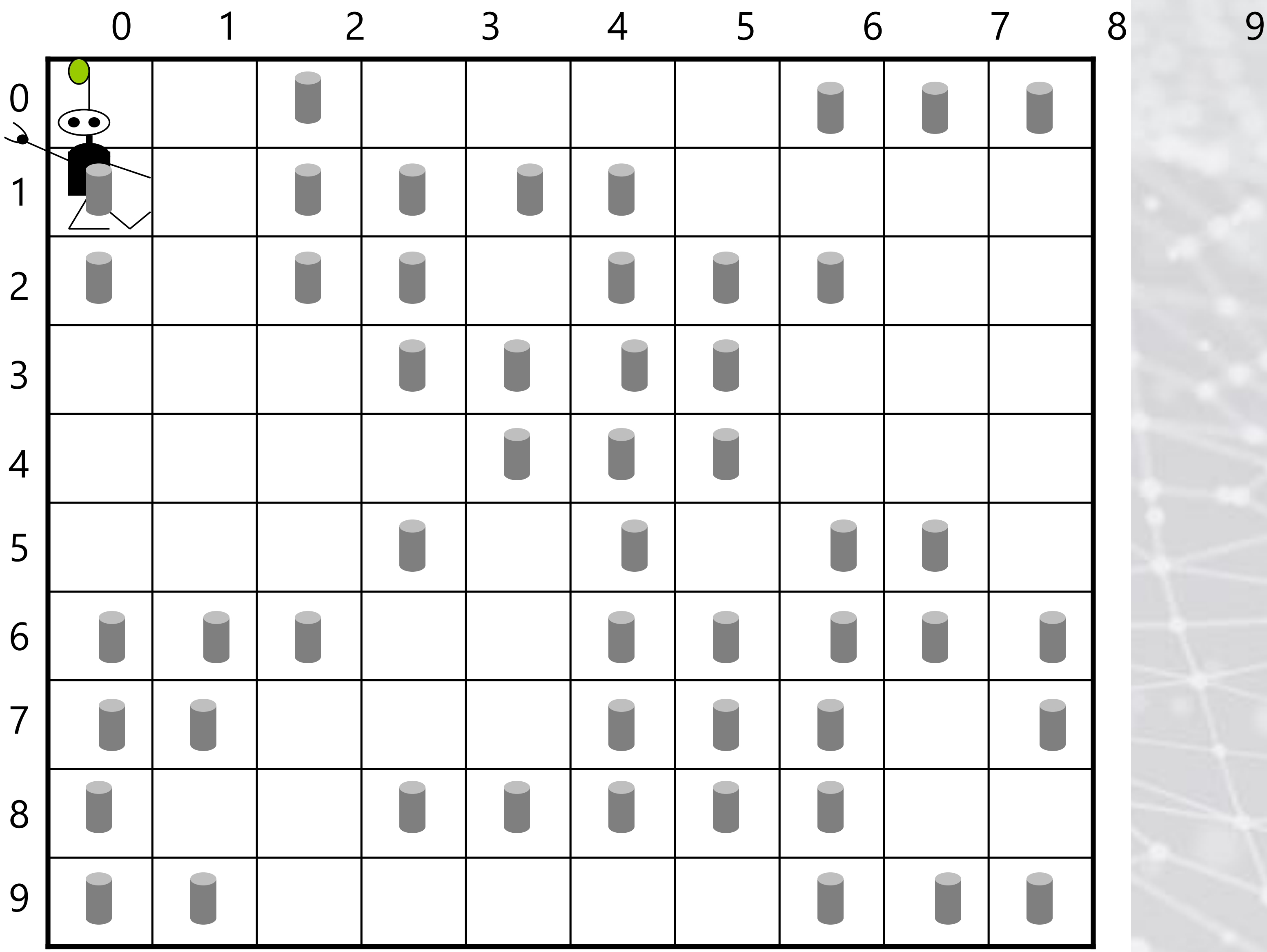
Generation 200

Fitness = 240

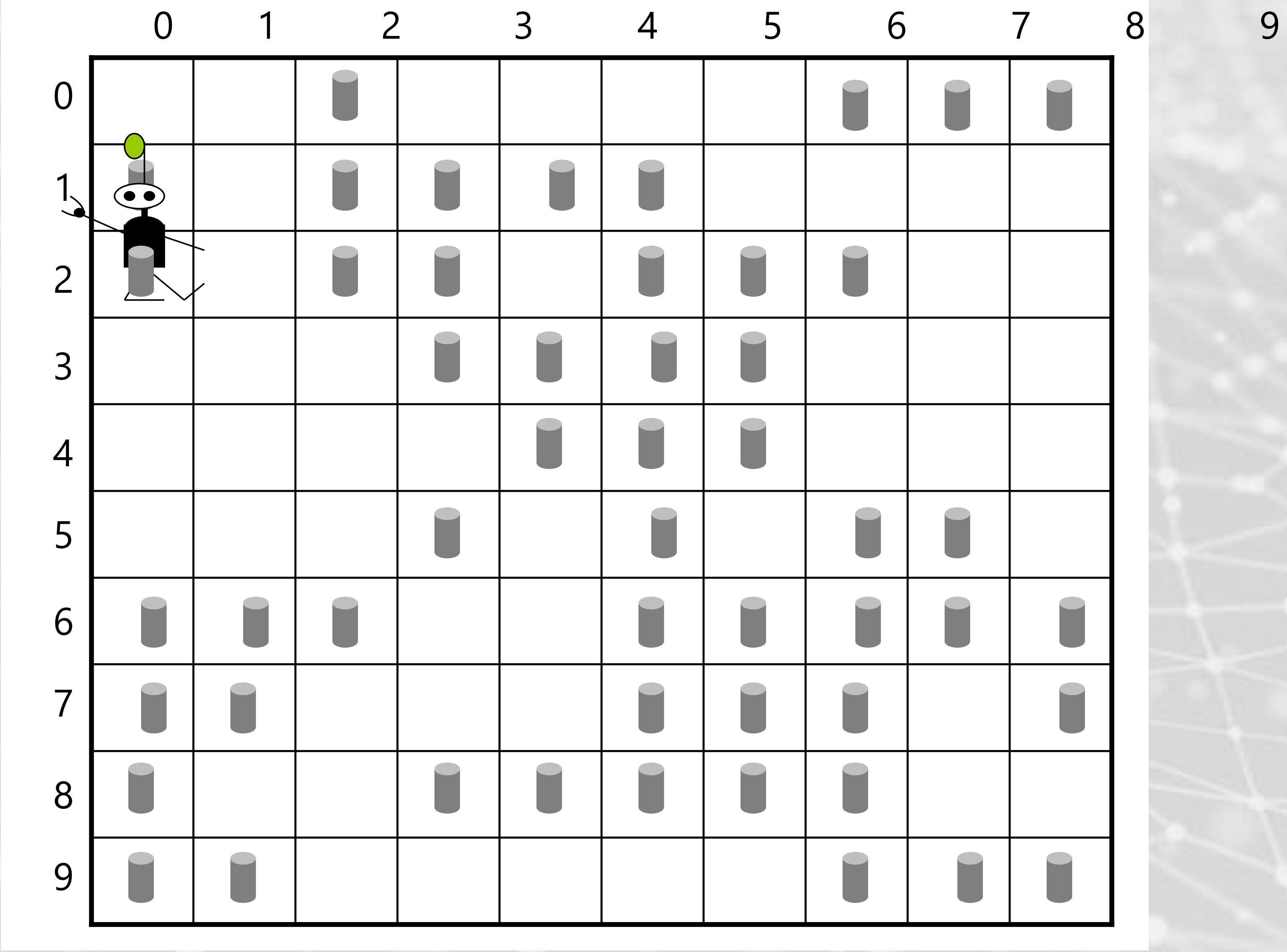
Time: 0 **Score: 0**



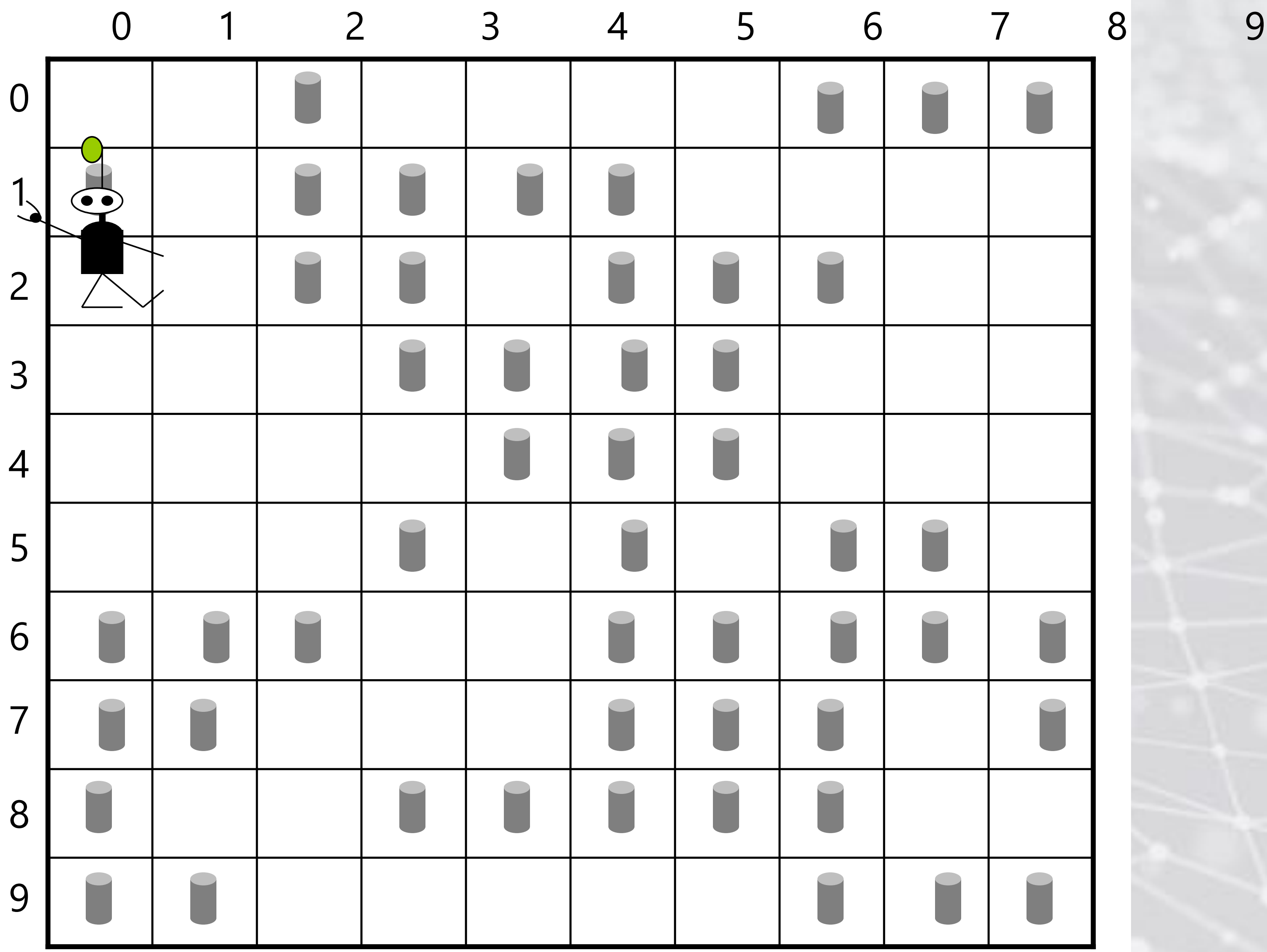
Time: 1 **Score: 0**



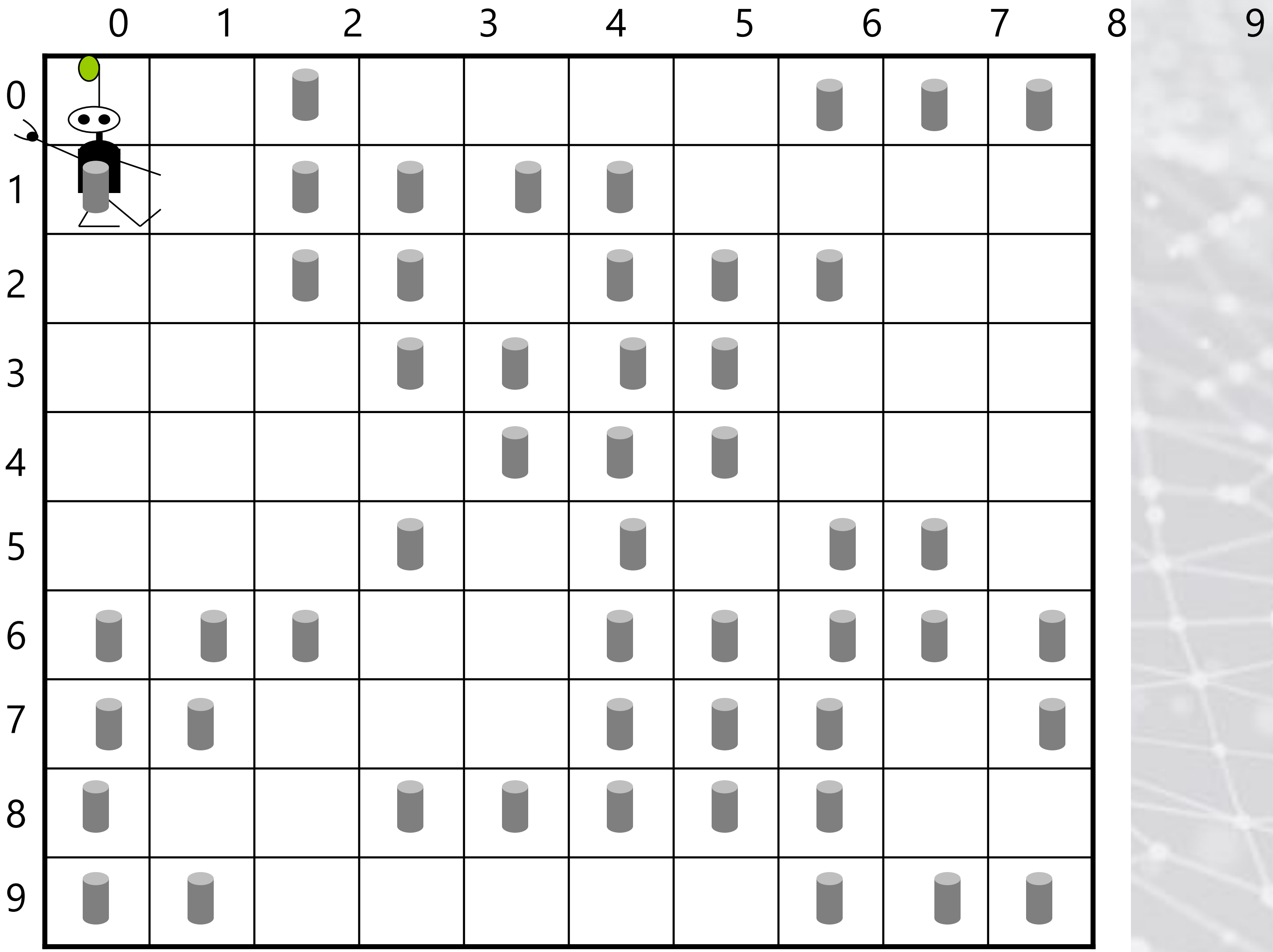
Time: 2 Score: 0



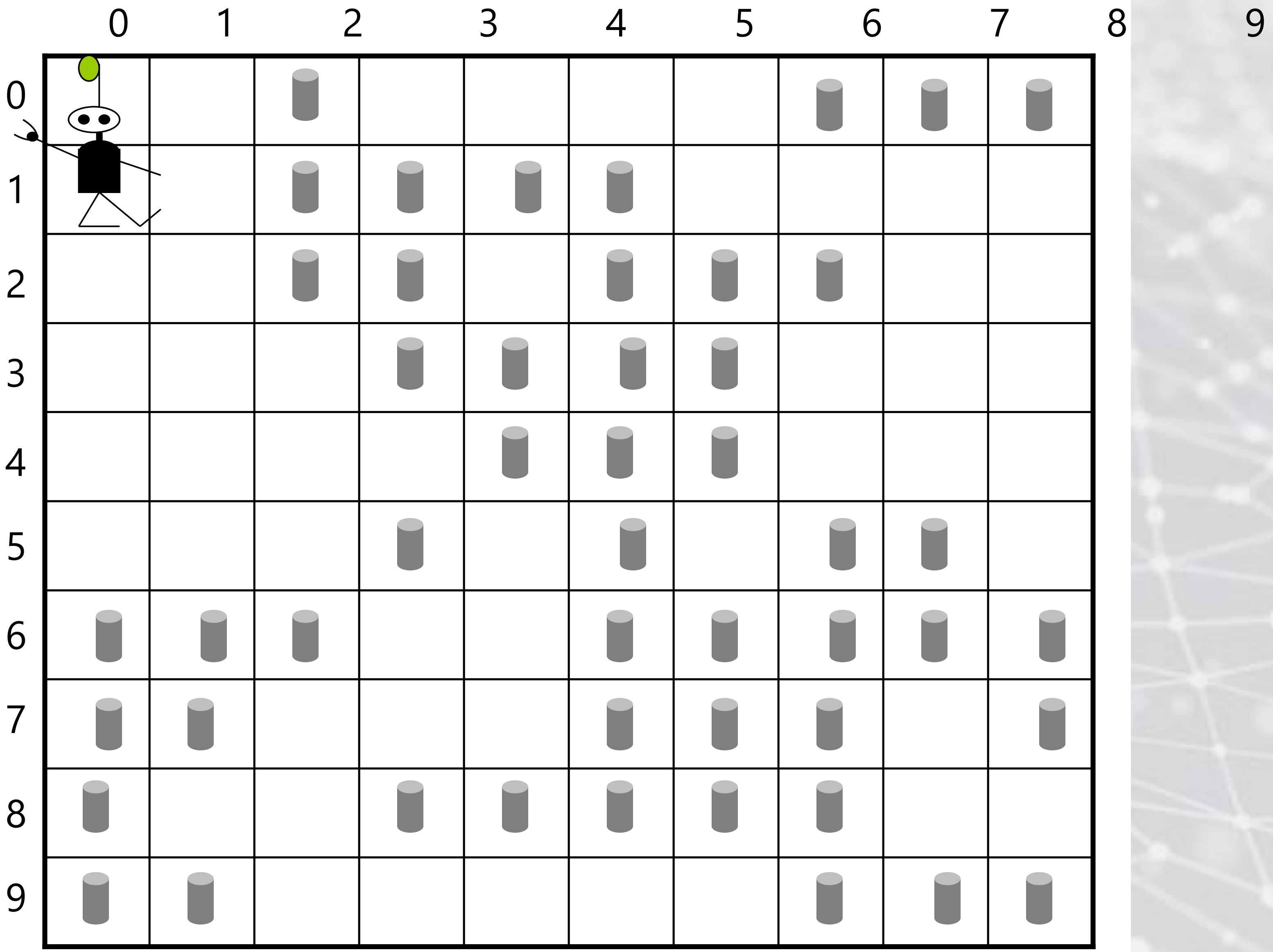
Time: 3 Score: 10



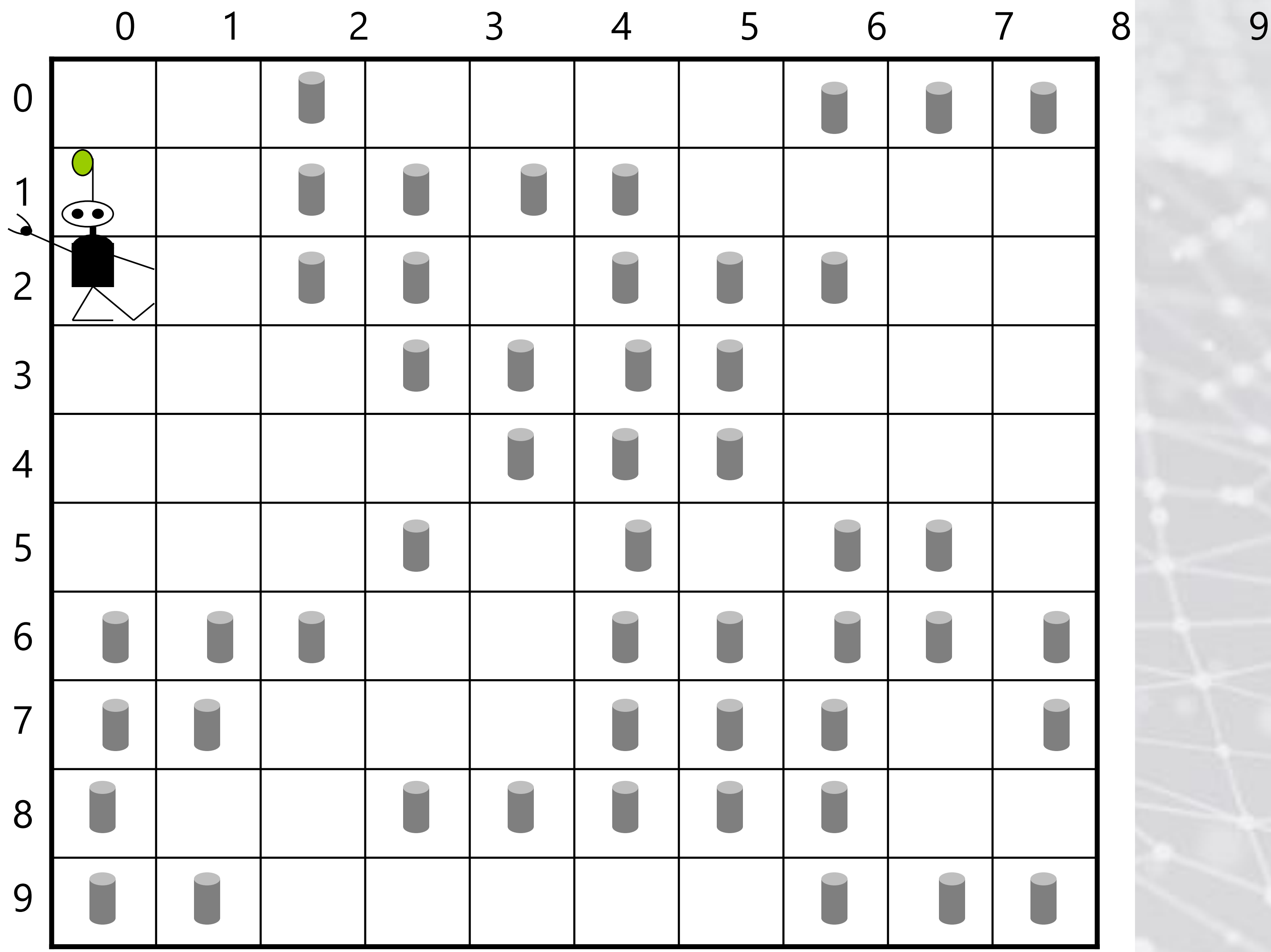
Time: 4 **Score: 10**



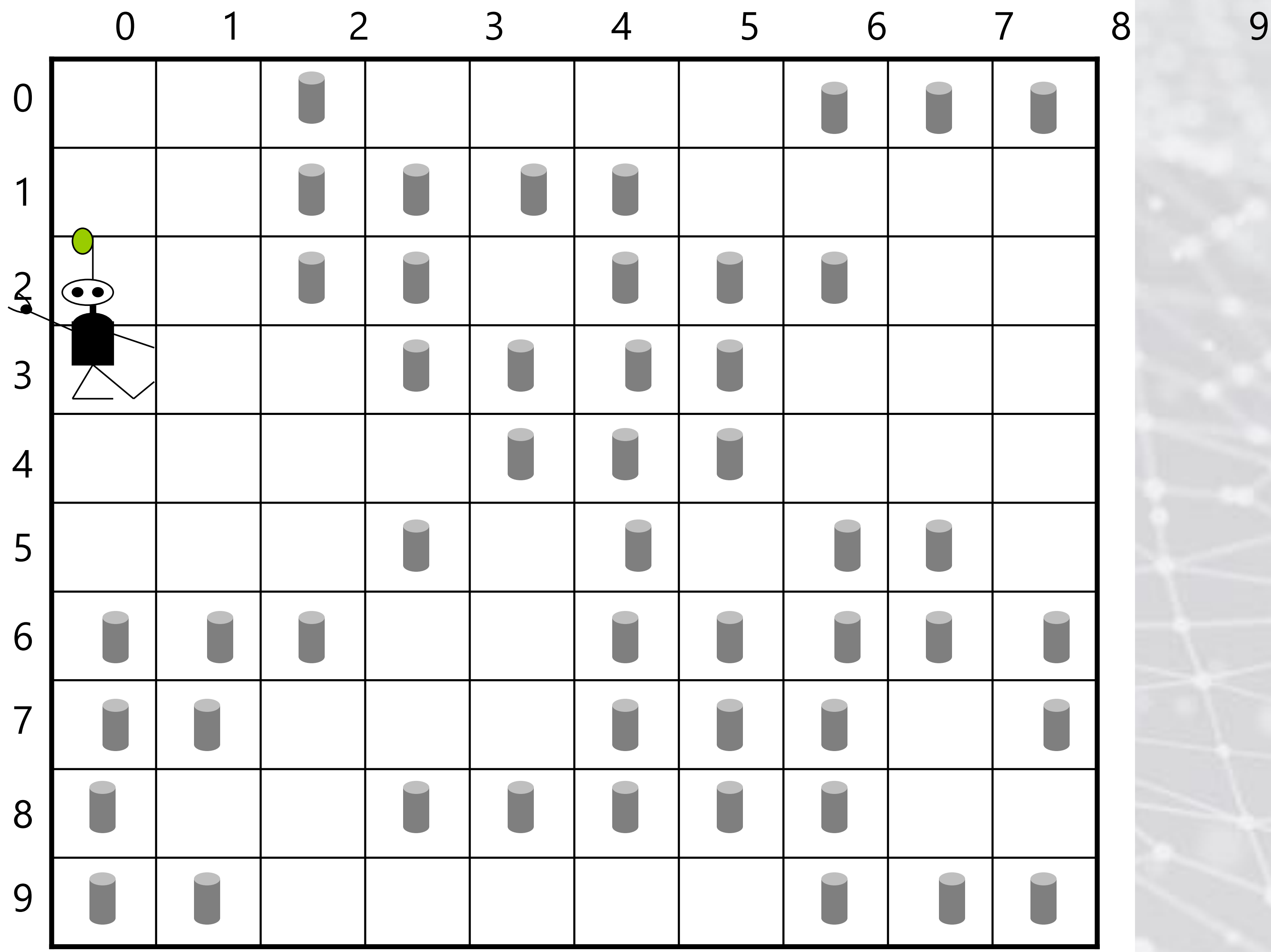
Time: 5 **Score: 20**



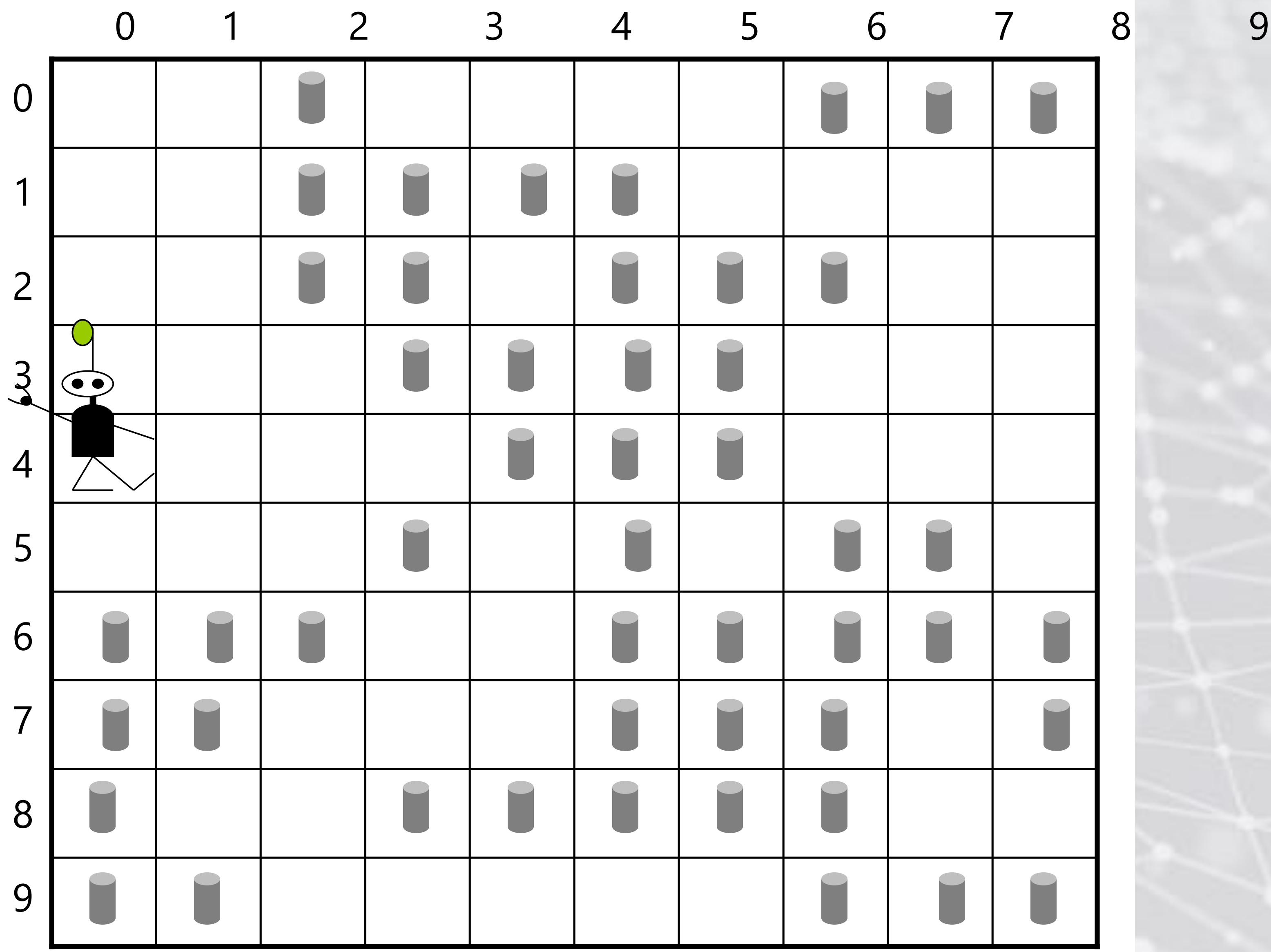
Time: 6 **Score: 20**



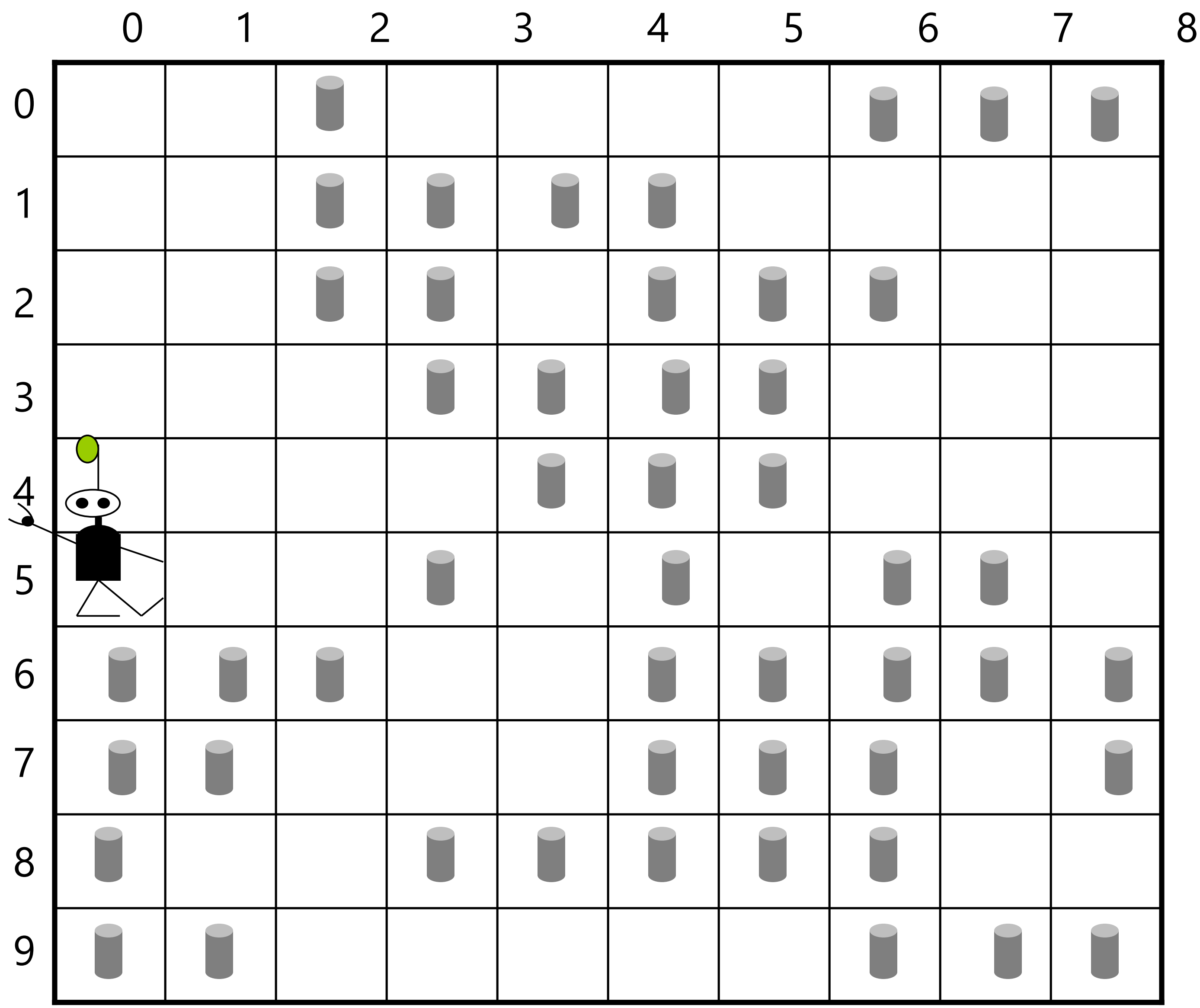
Time: 7 Score: 20



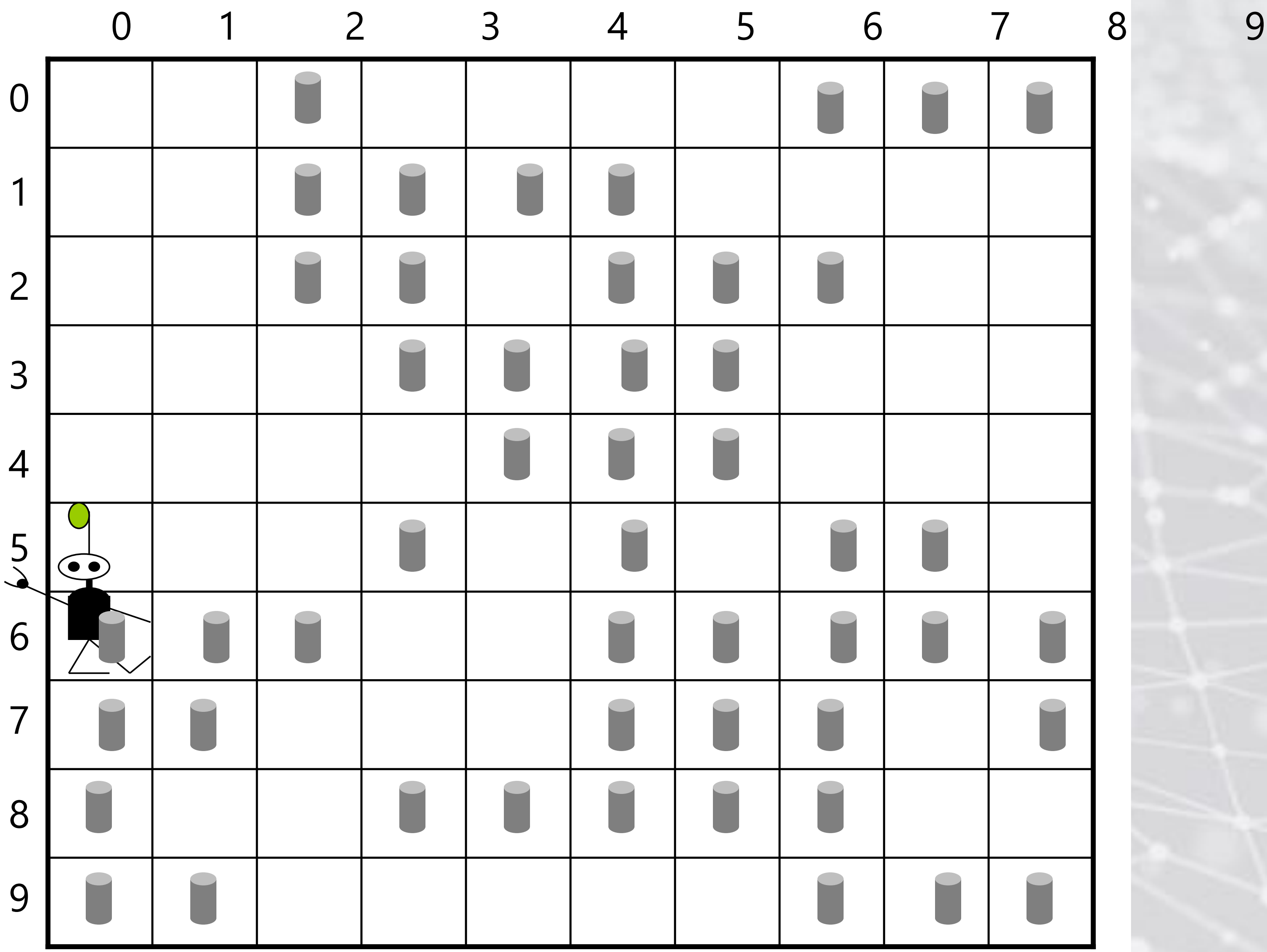
Time: 8 **Score: 20**



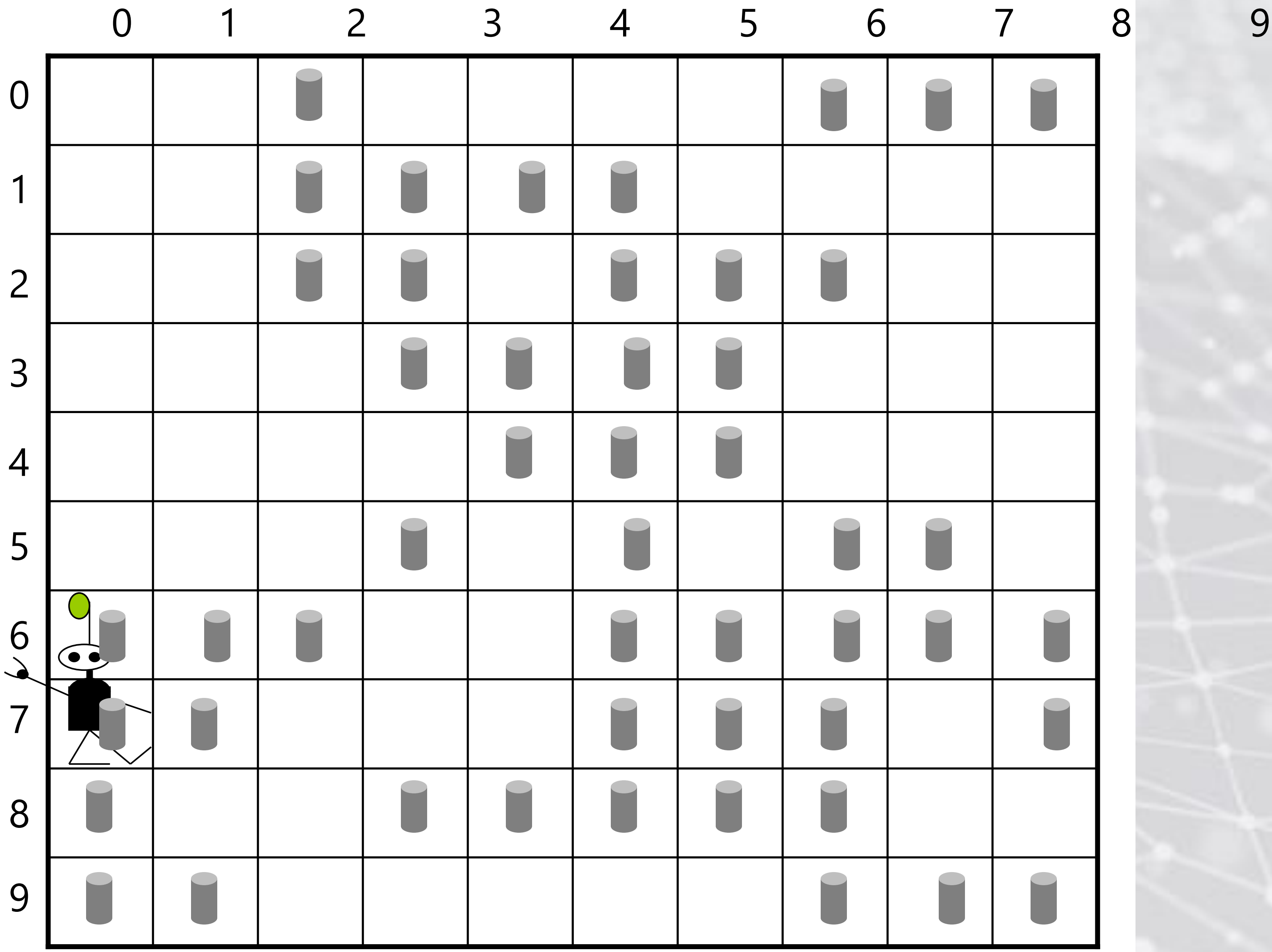
Time: 9 **Score: 20**



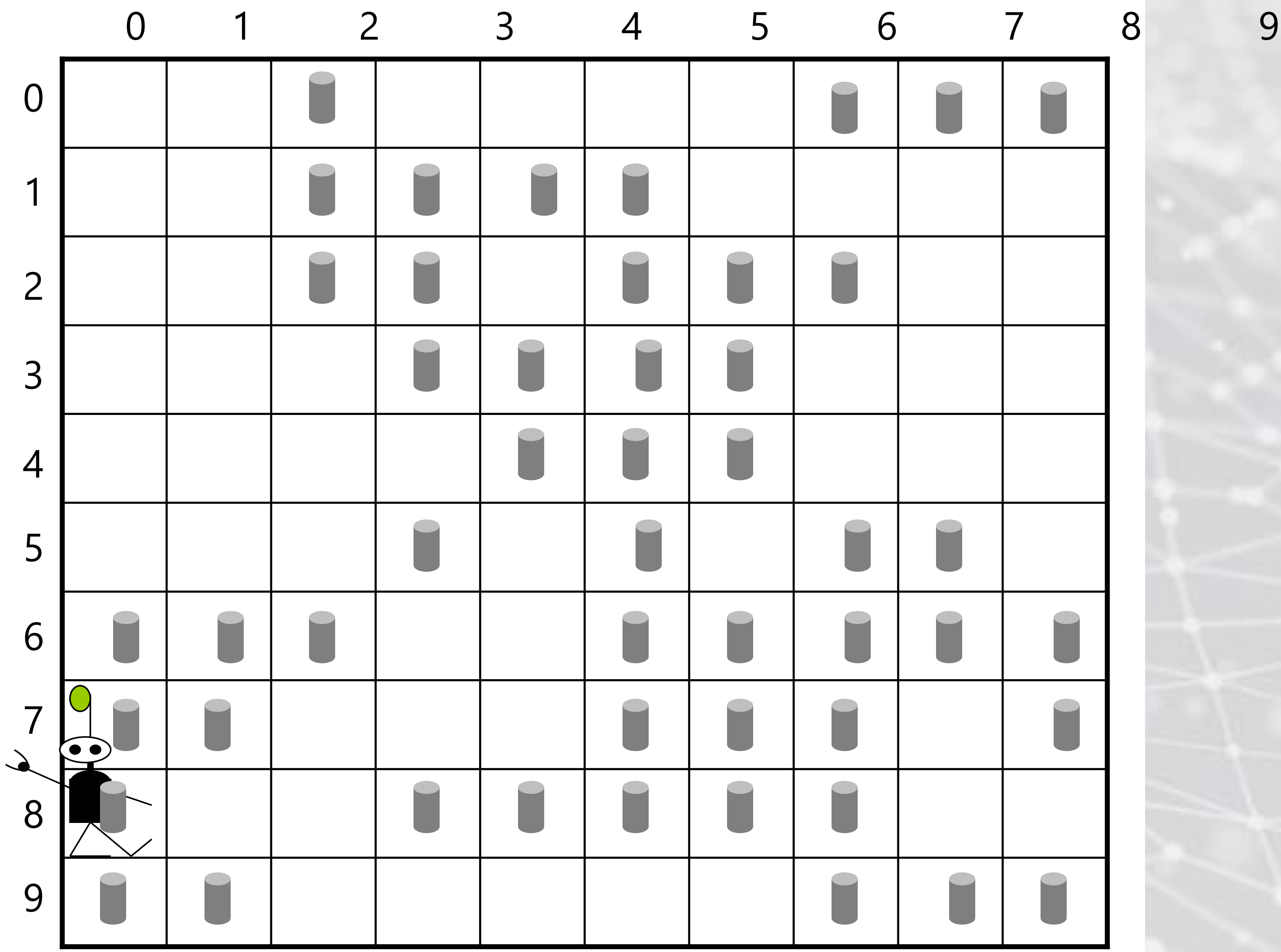
Time: 10   Score: 20



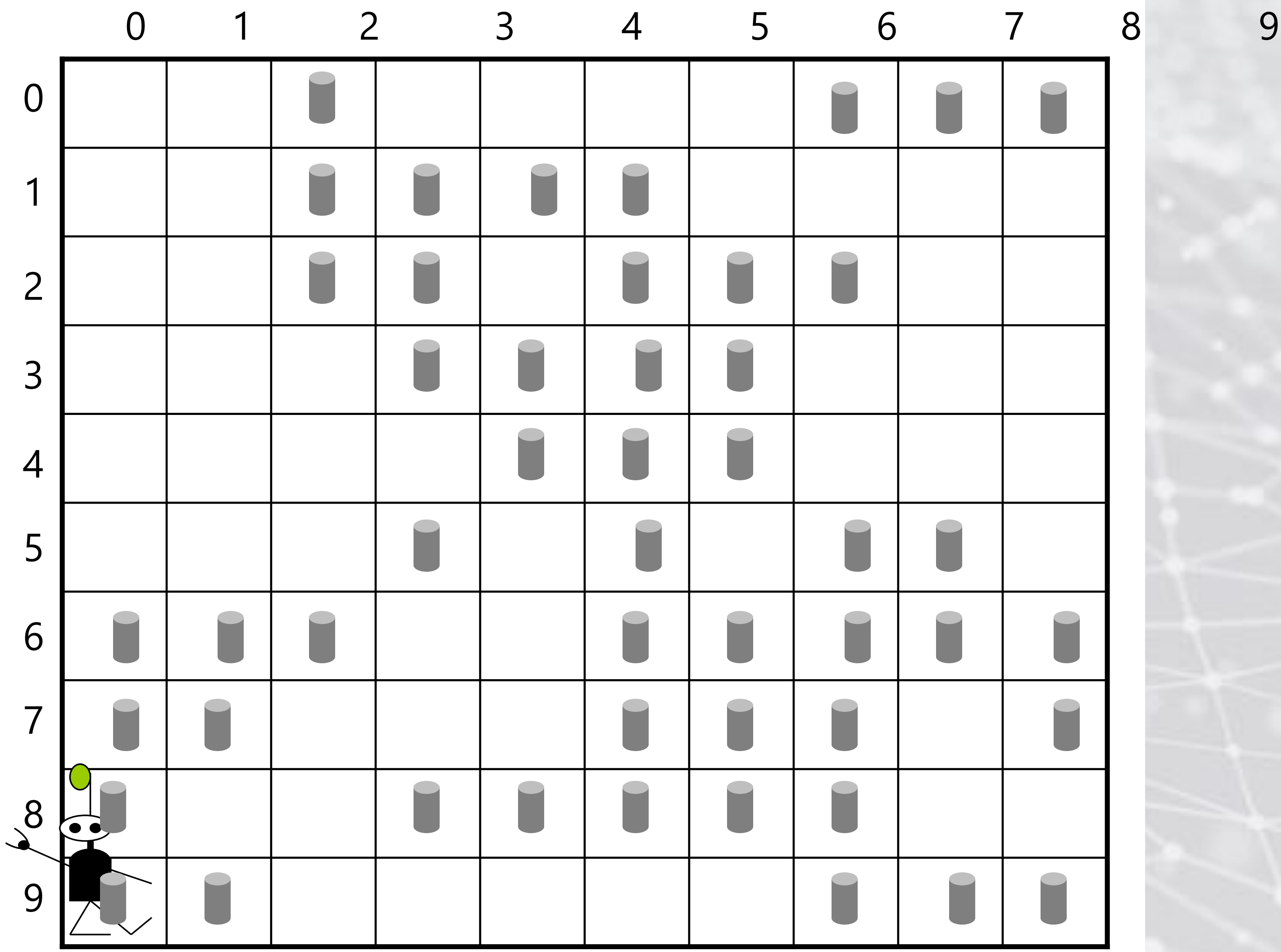
Time: 11 Score: 20



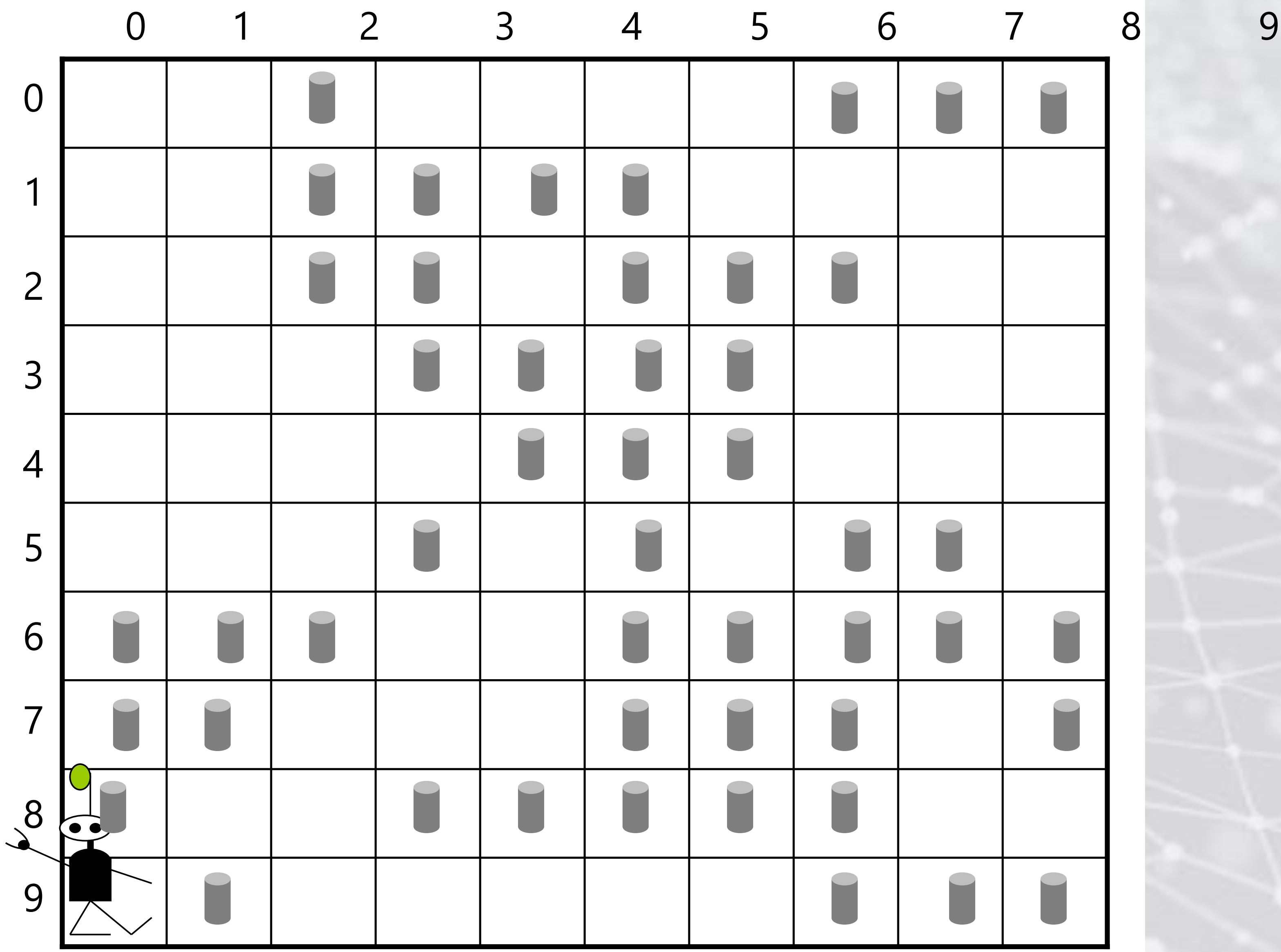
Time: 12   Score: 20



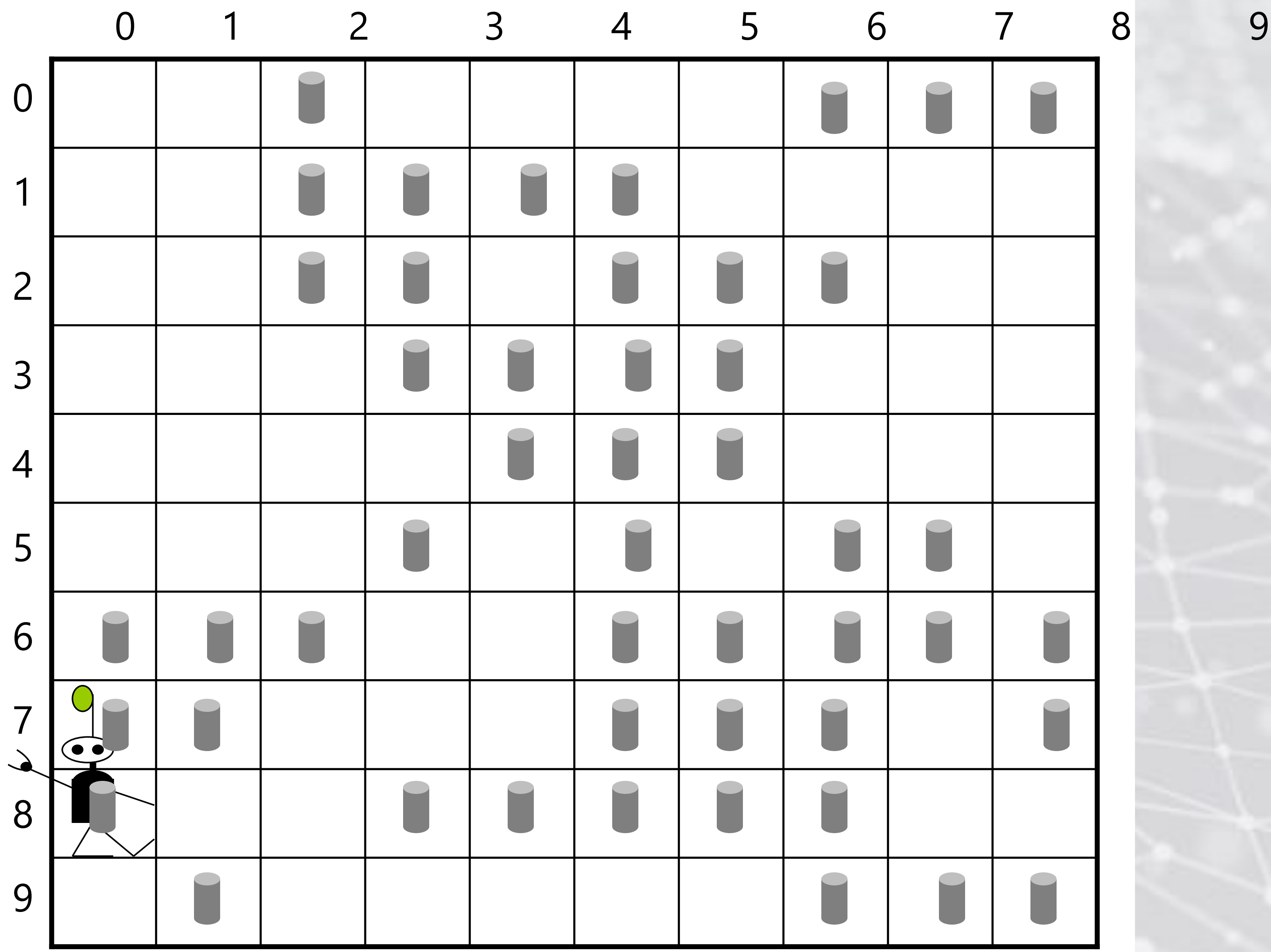
Time: 13 Score: 20



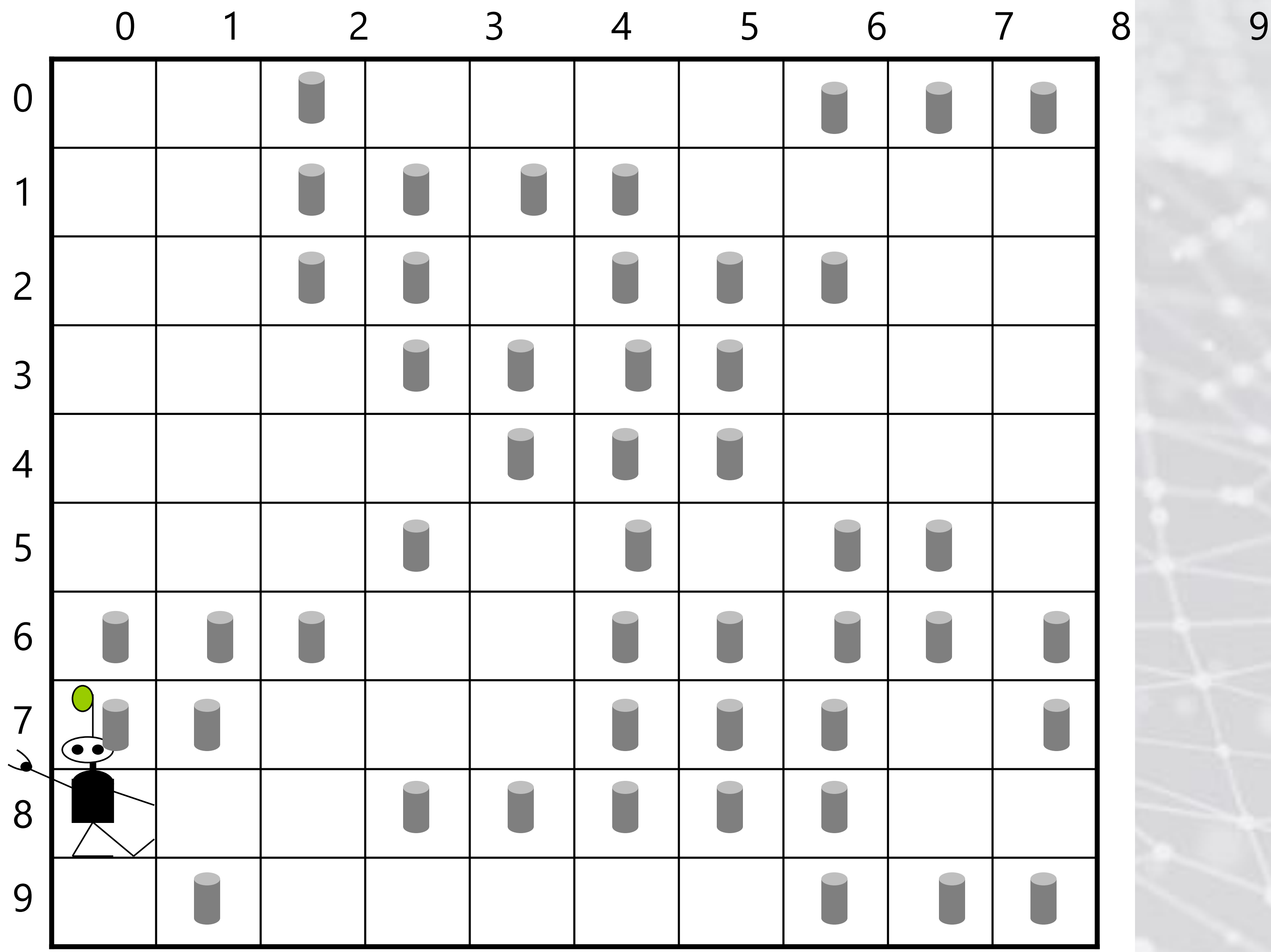
Time: 14 Score: 30



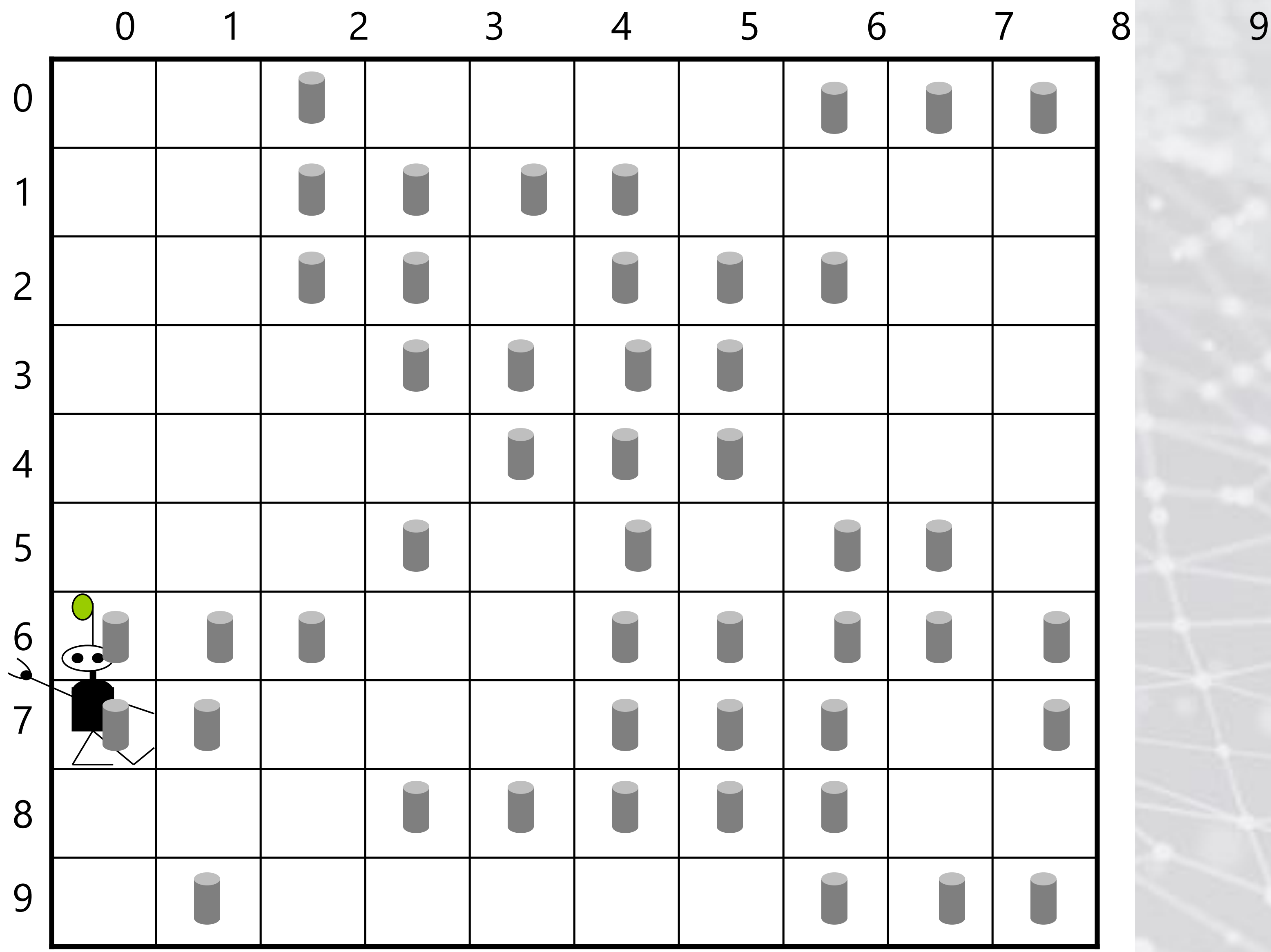
Time: 15 Score: 30



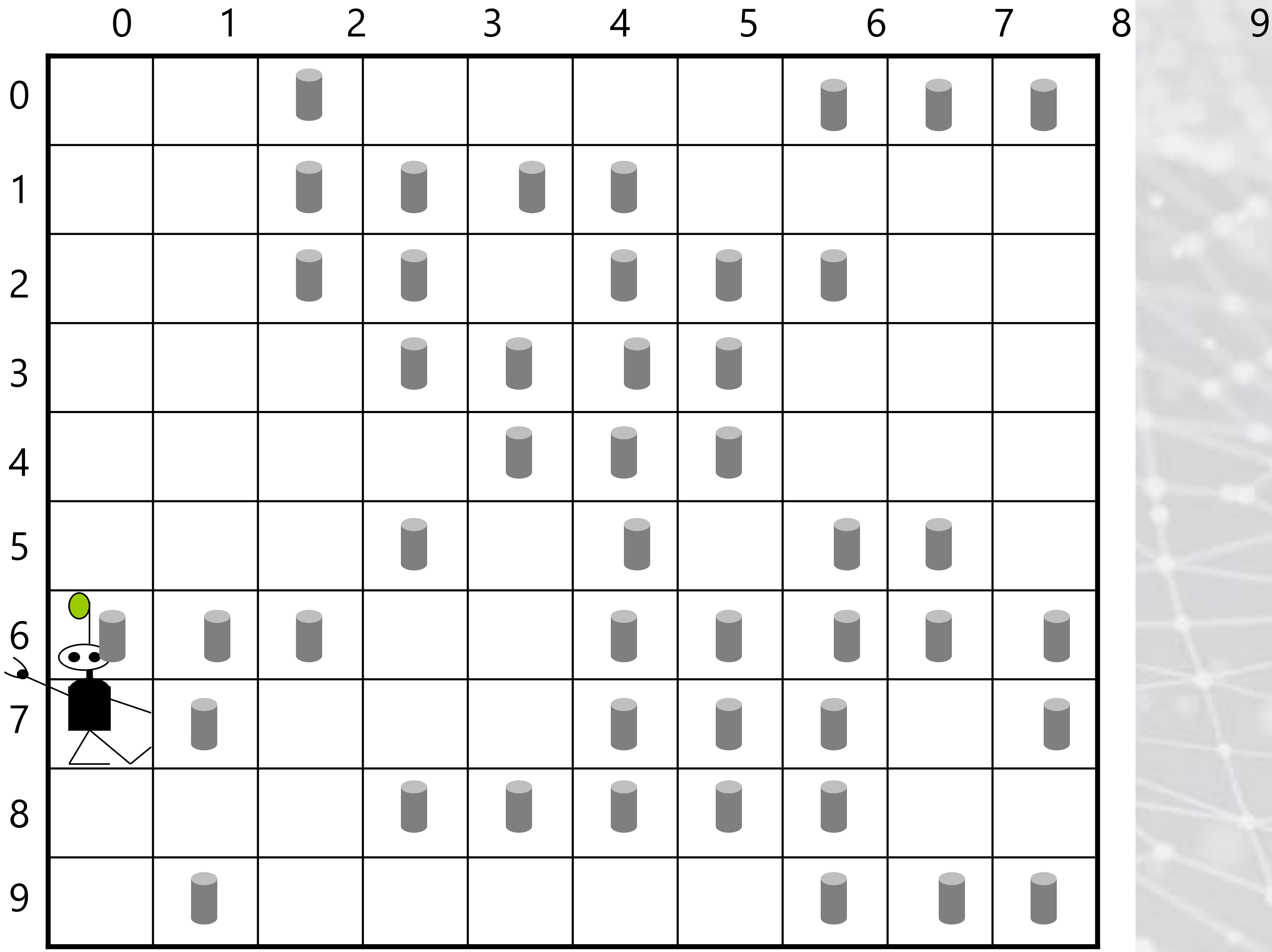
Time: 16 Score: 40



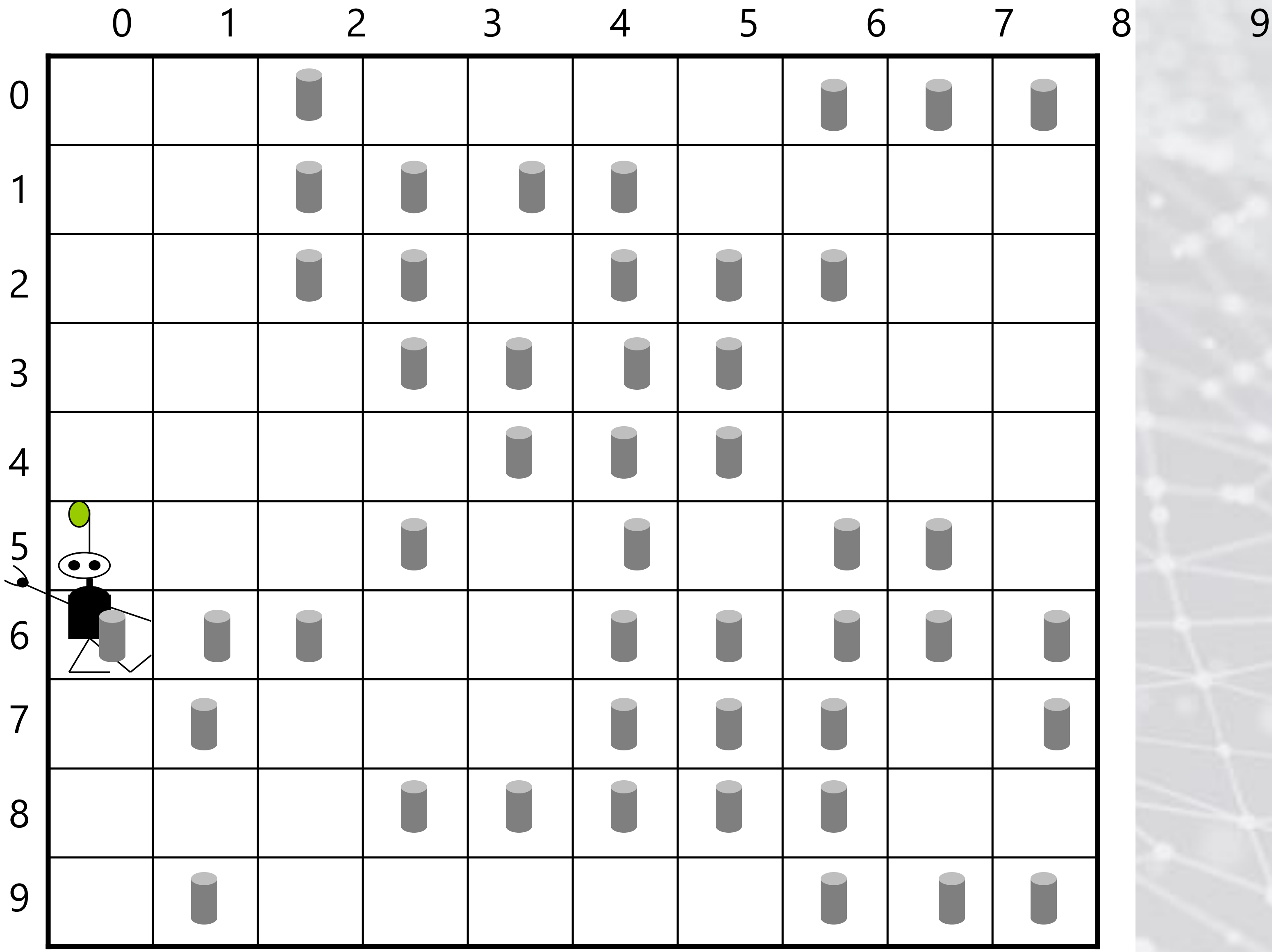
Time: 17 Score: 40



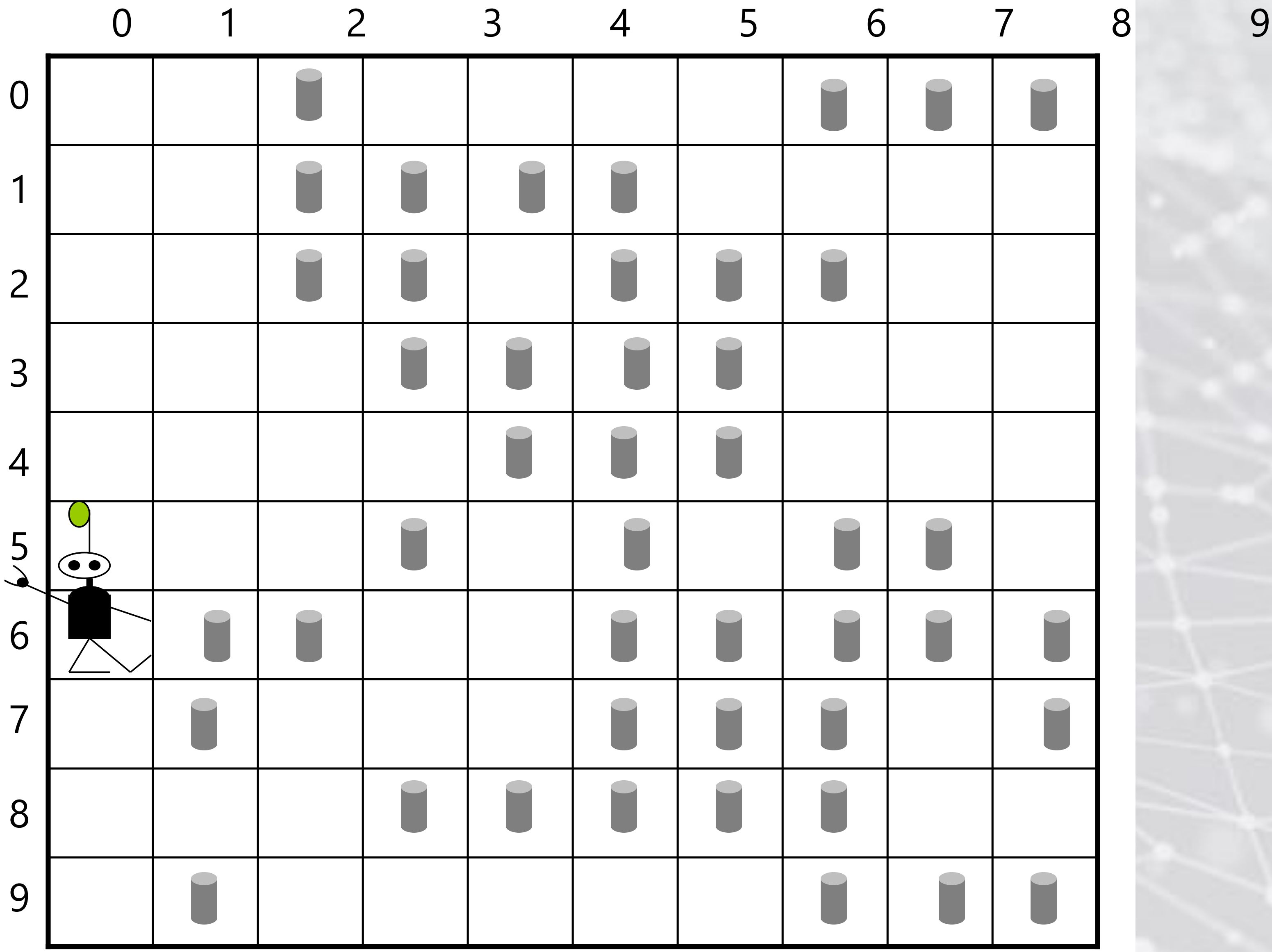
Time: 18 Score: 50



Time: 19 Score: 50



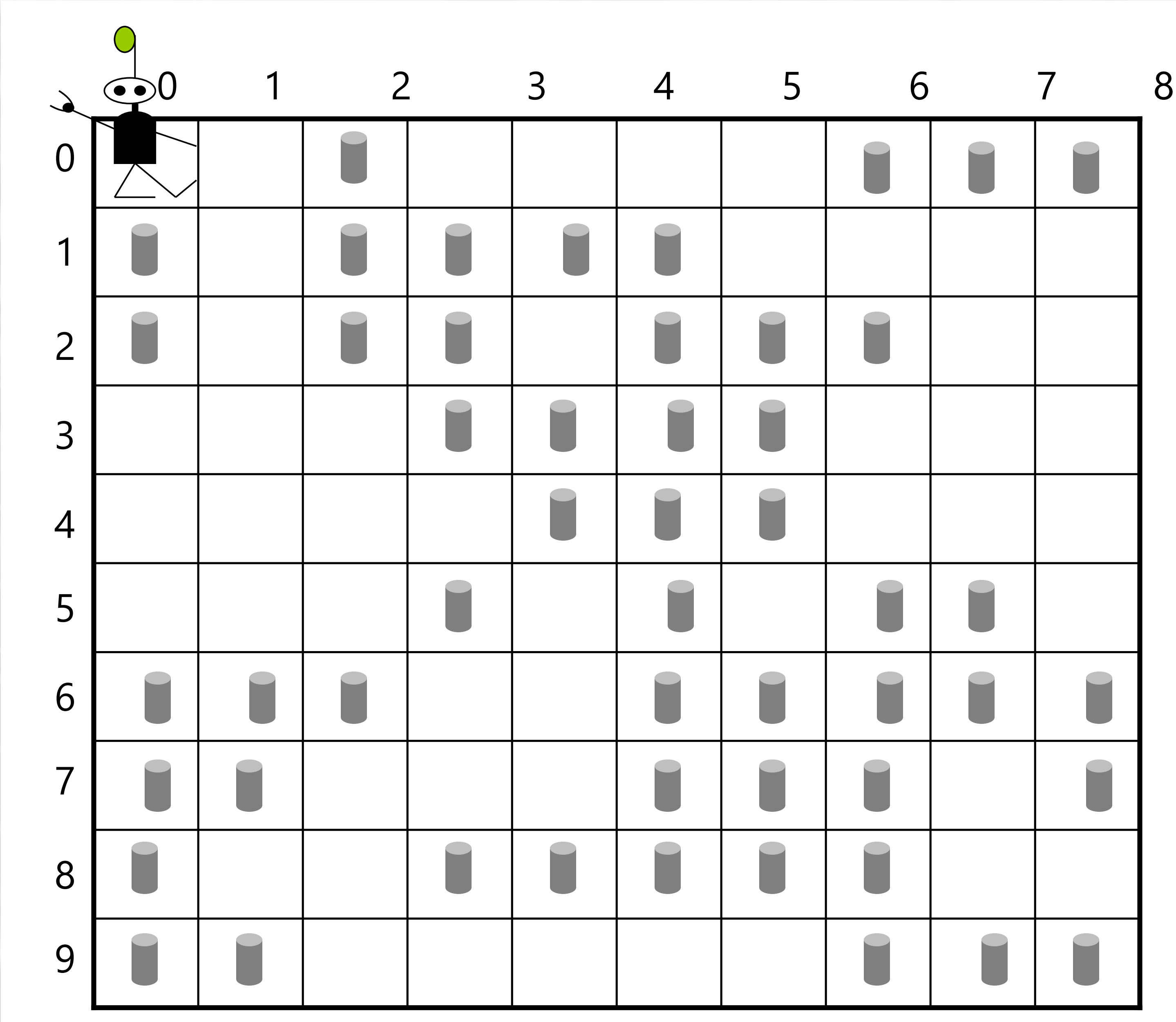
Time: 20   Score: 60



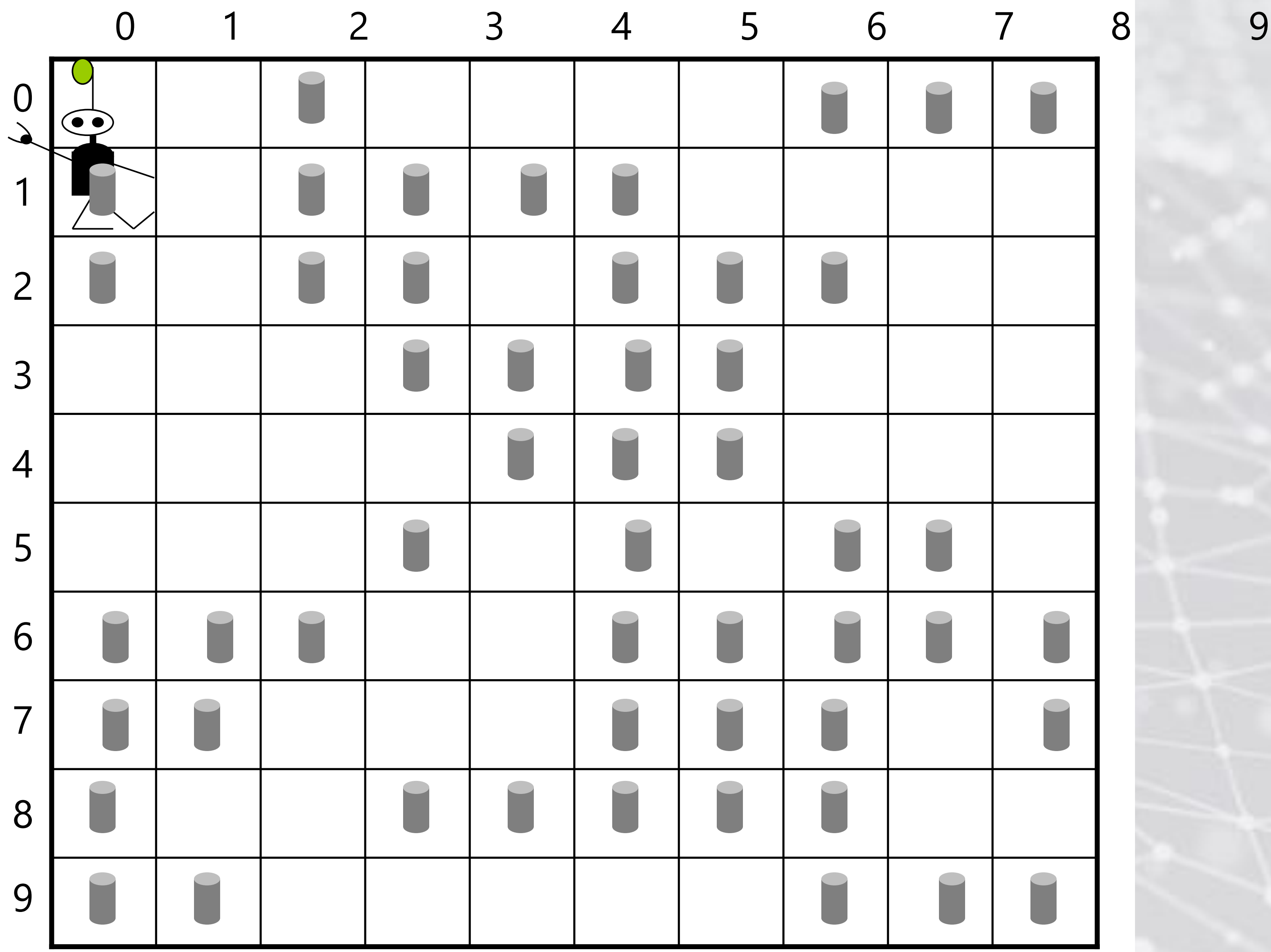
Generation 1000

Fitness = 492

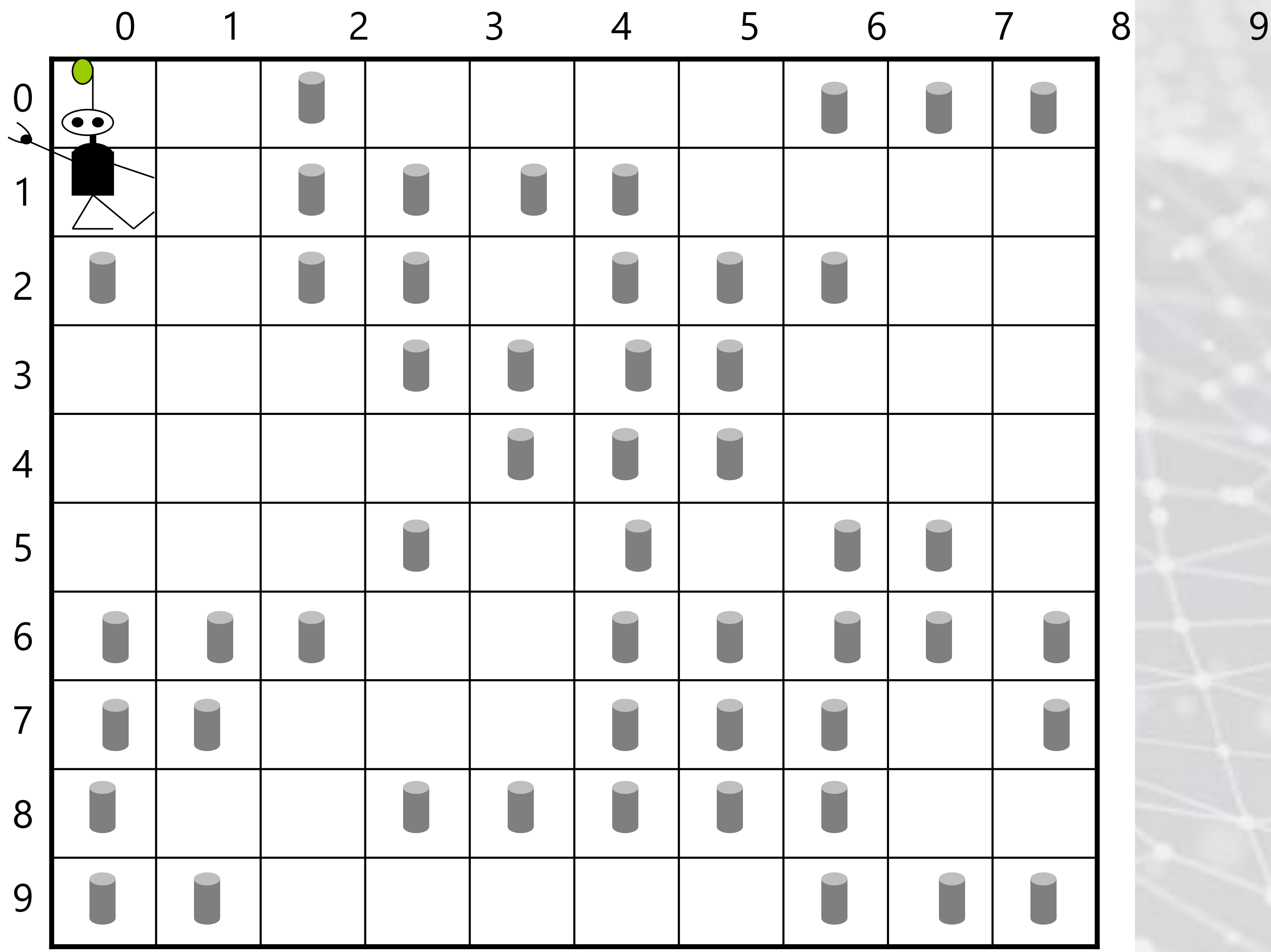
Time: 0 Score: 0



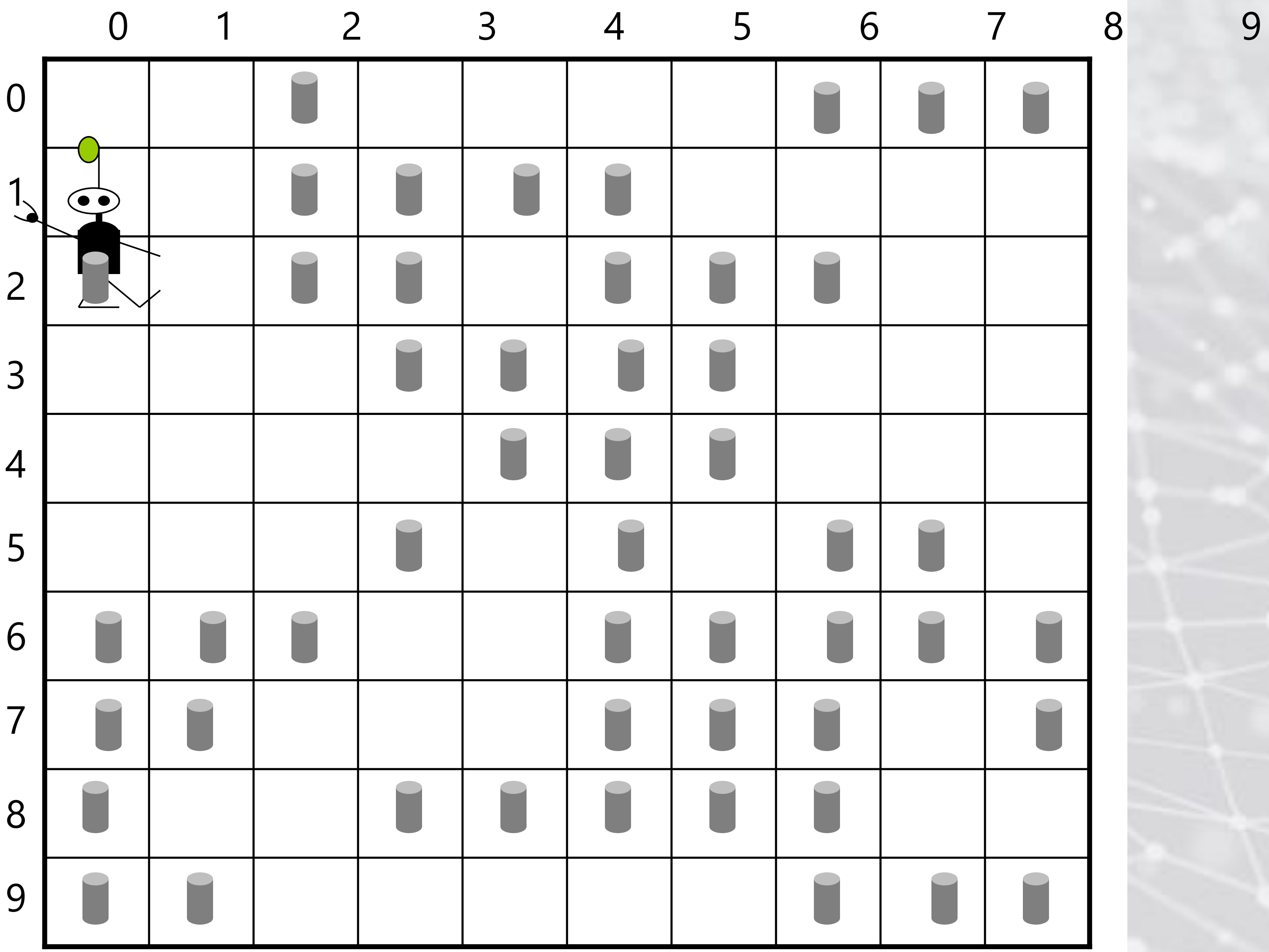
Time: 1   Score: 0



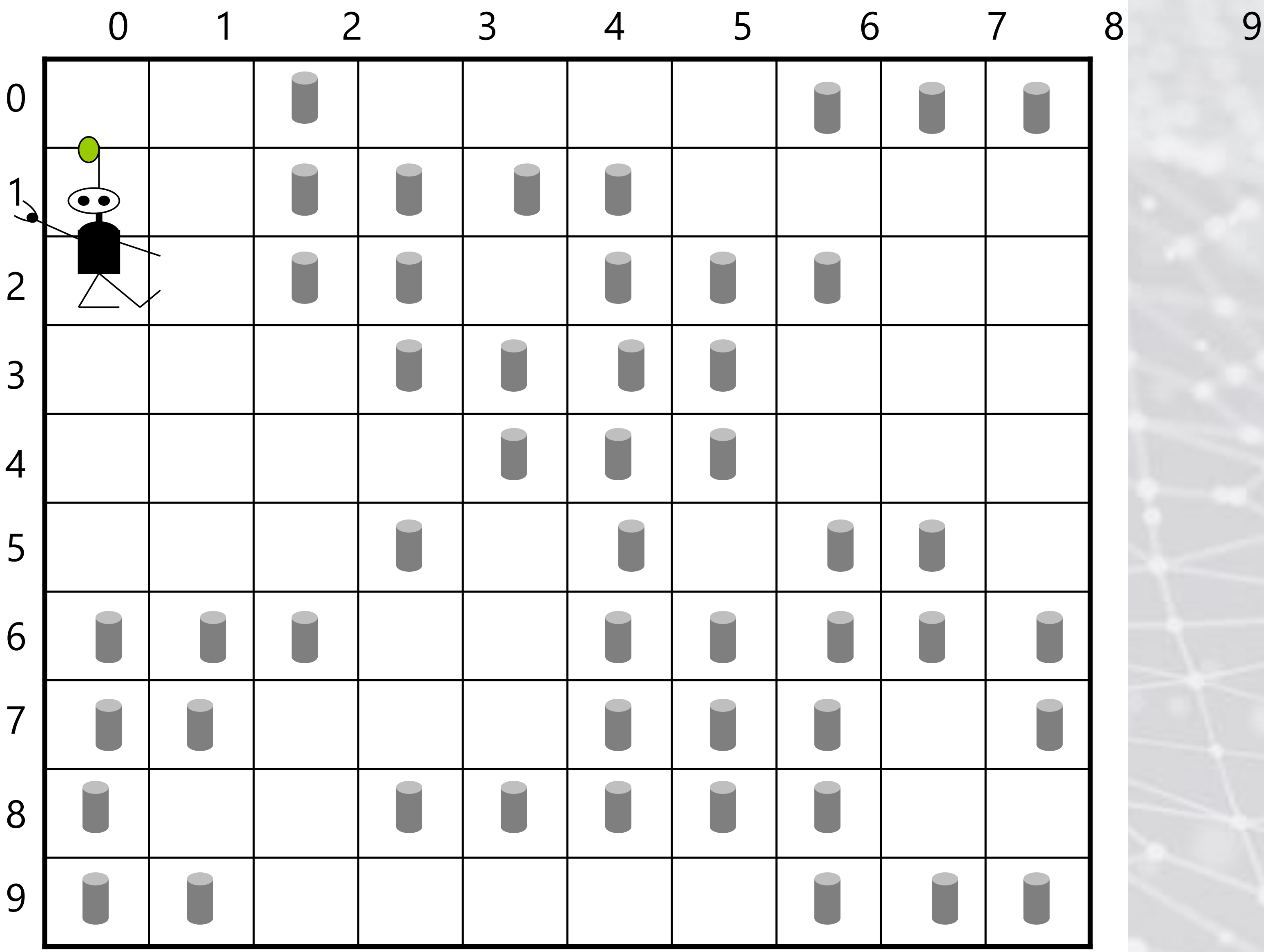
Time: 2 **Score: 10**



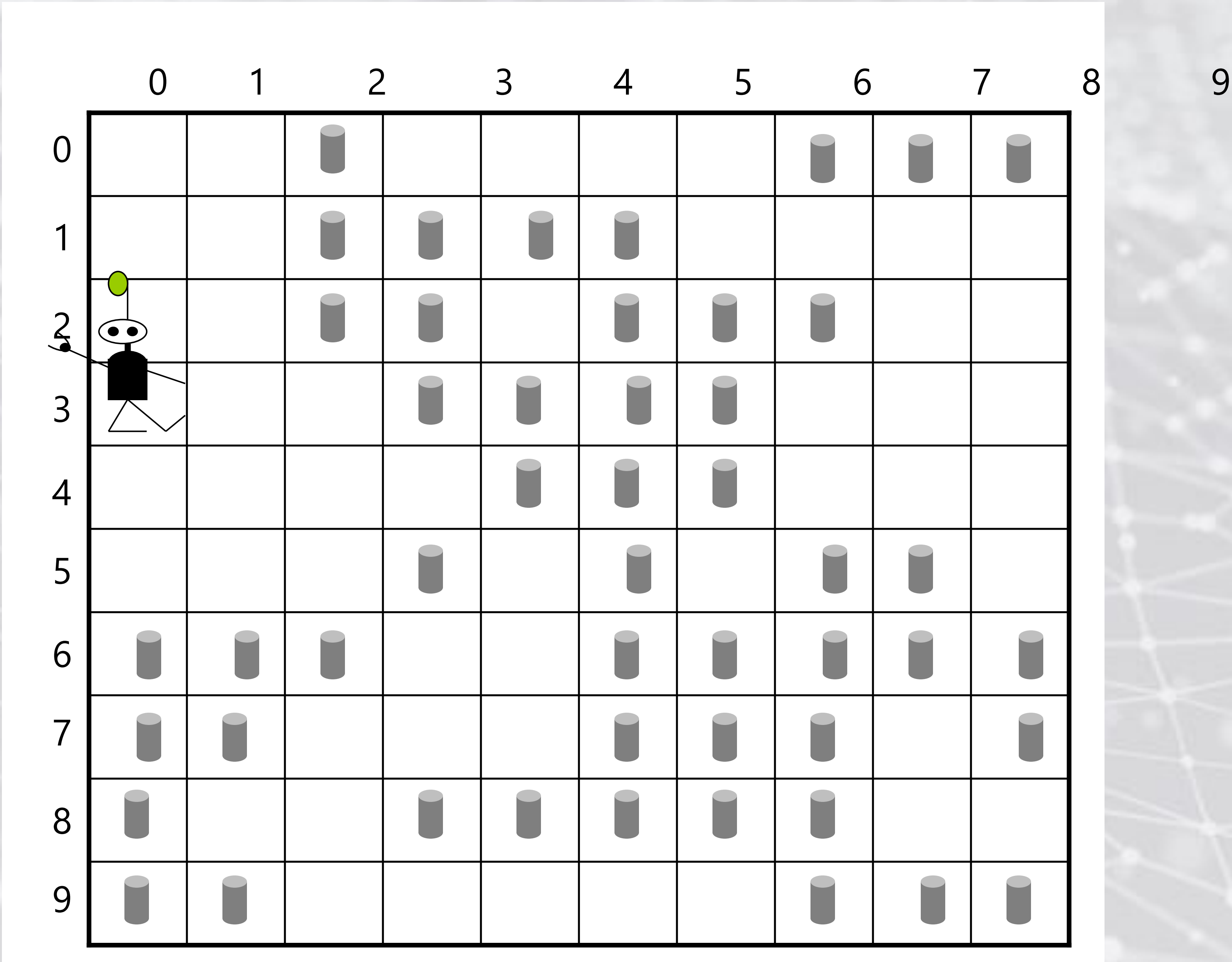
Time: 3 Score: 10



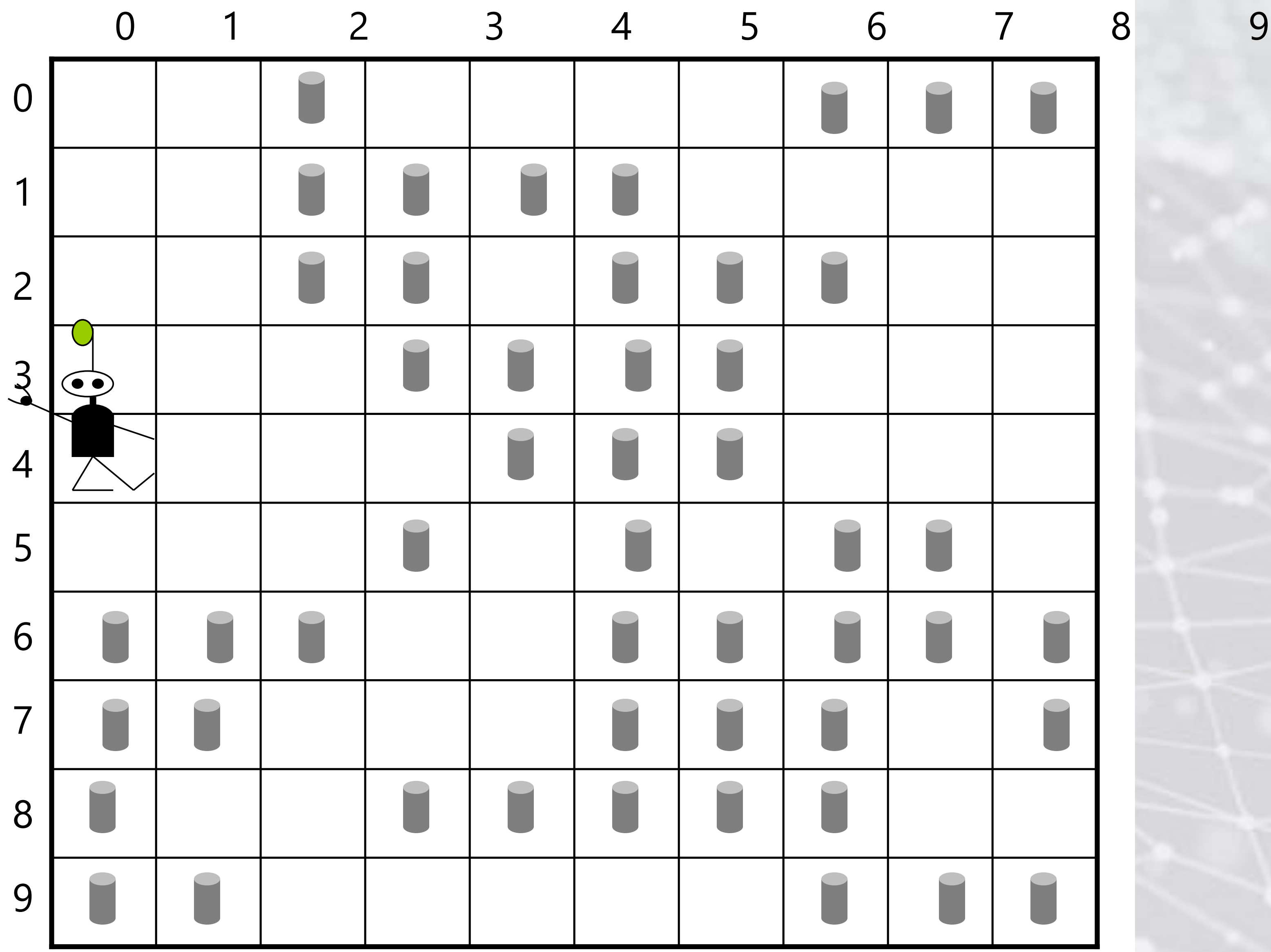
Time: 4 **Score: 20**



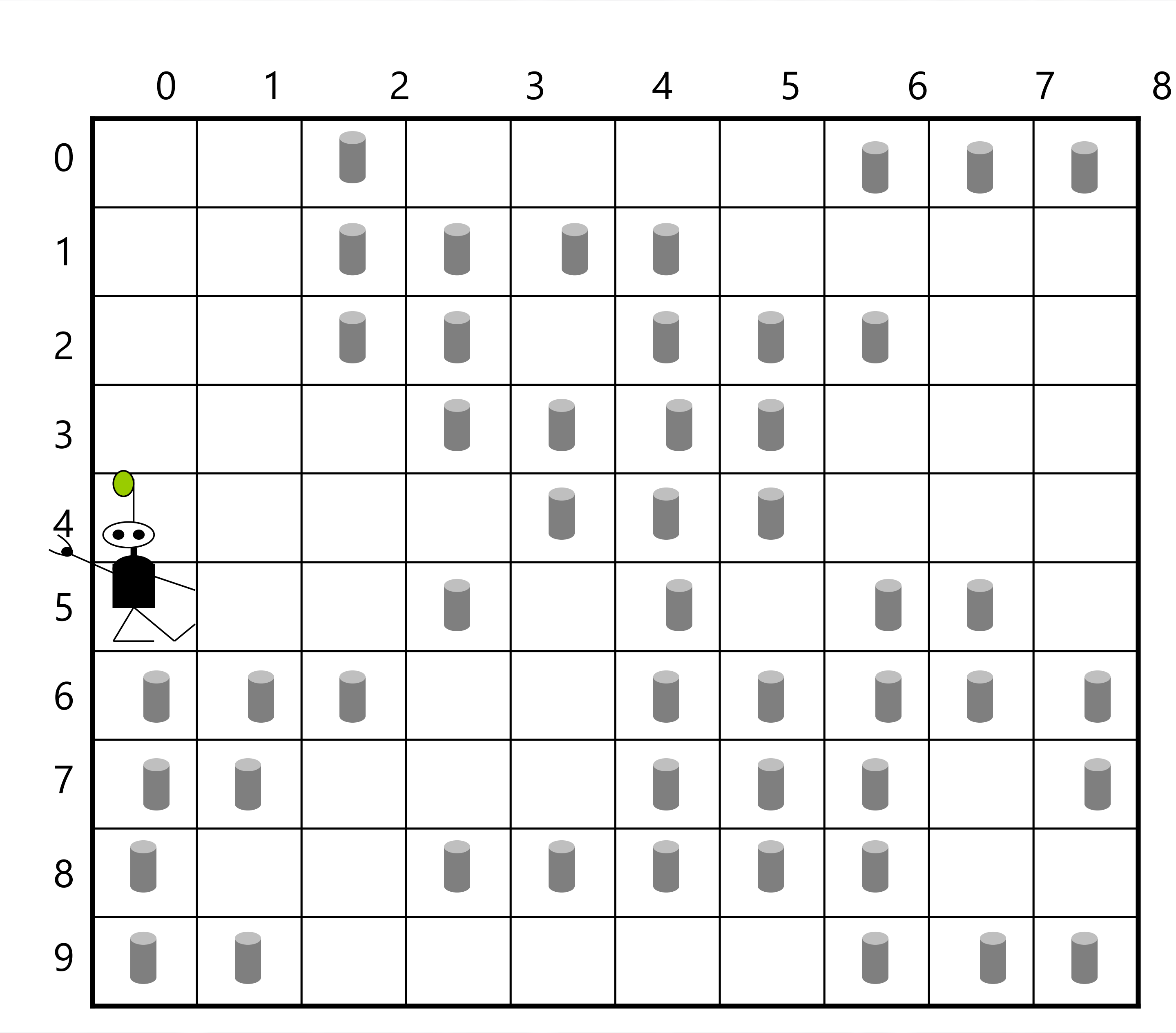
Time: 5 Score: 20



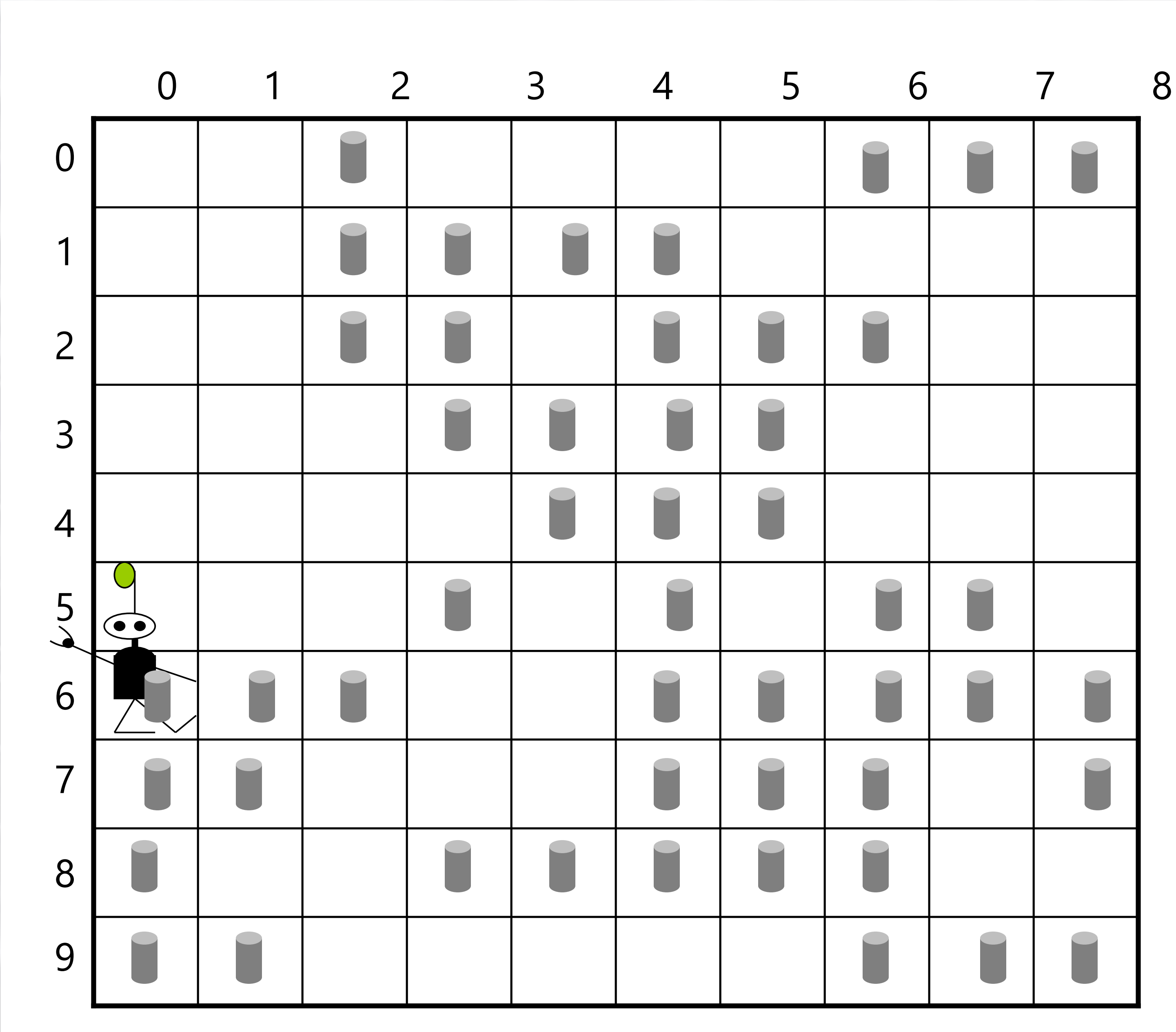
Time: 6 **Score: 20**



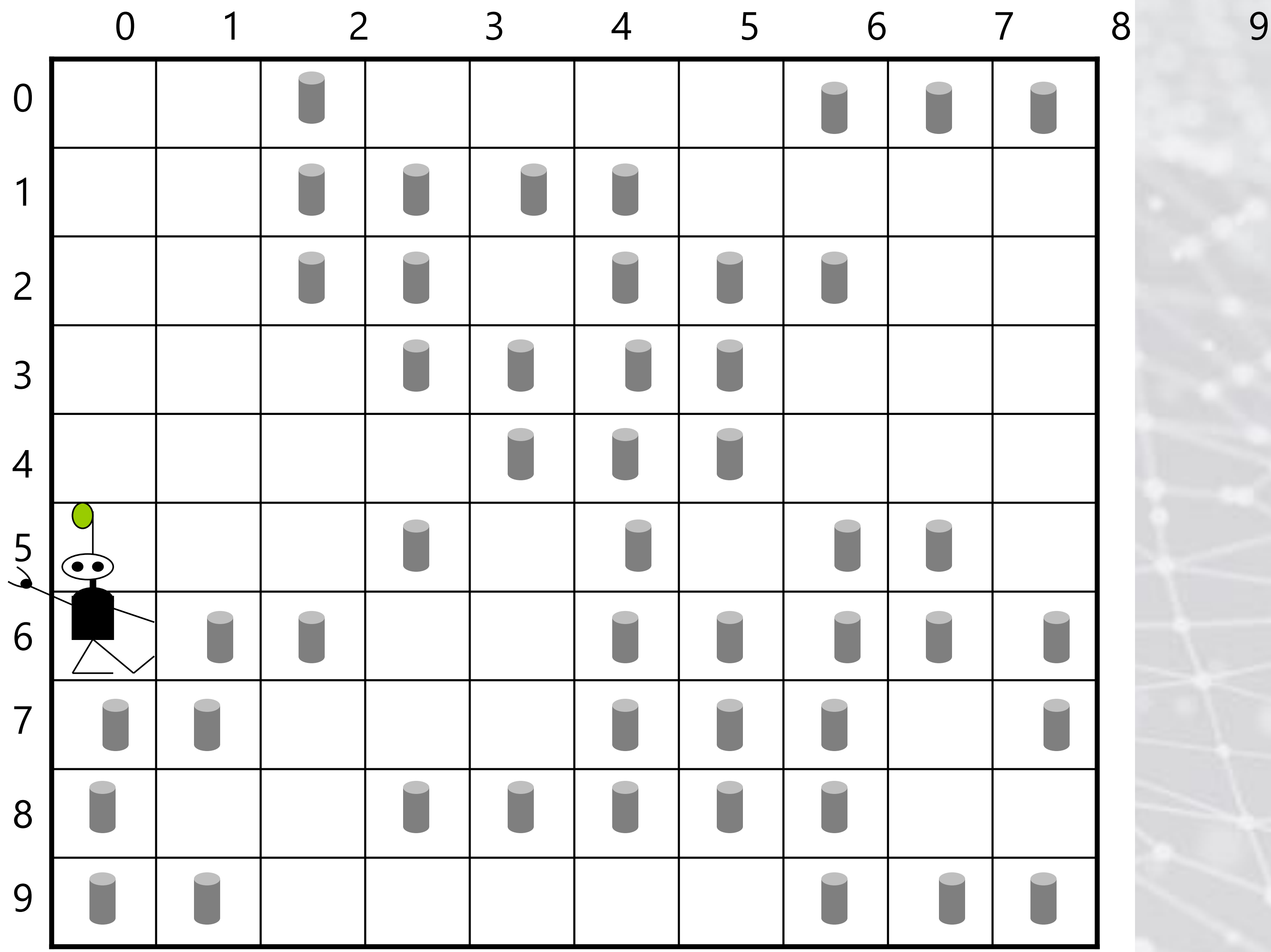
Time: 7 Score: 20



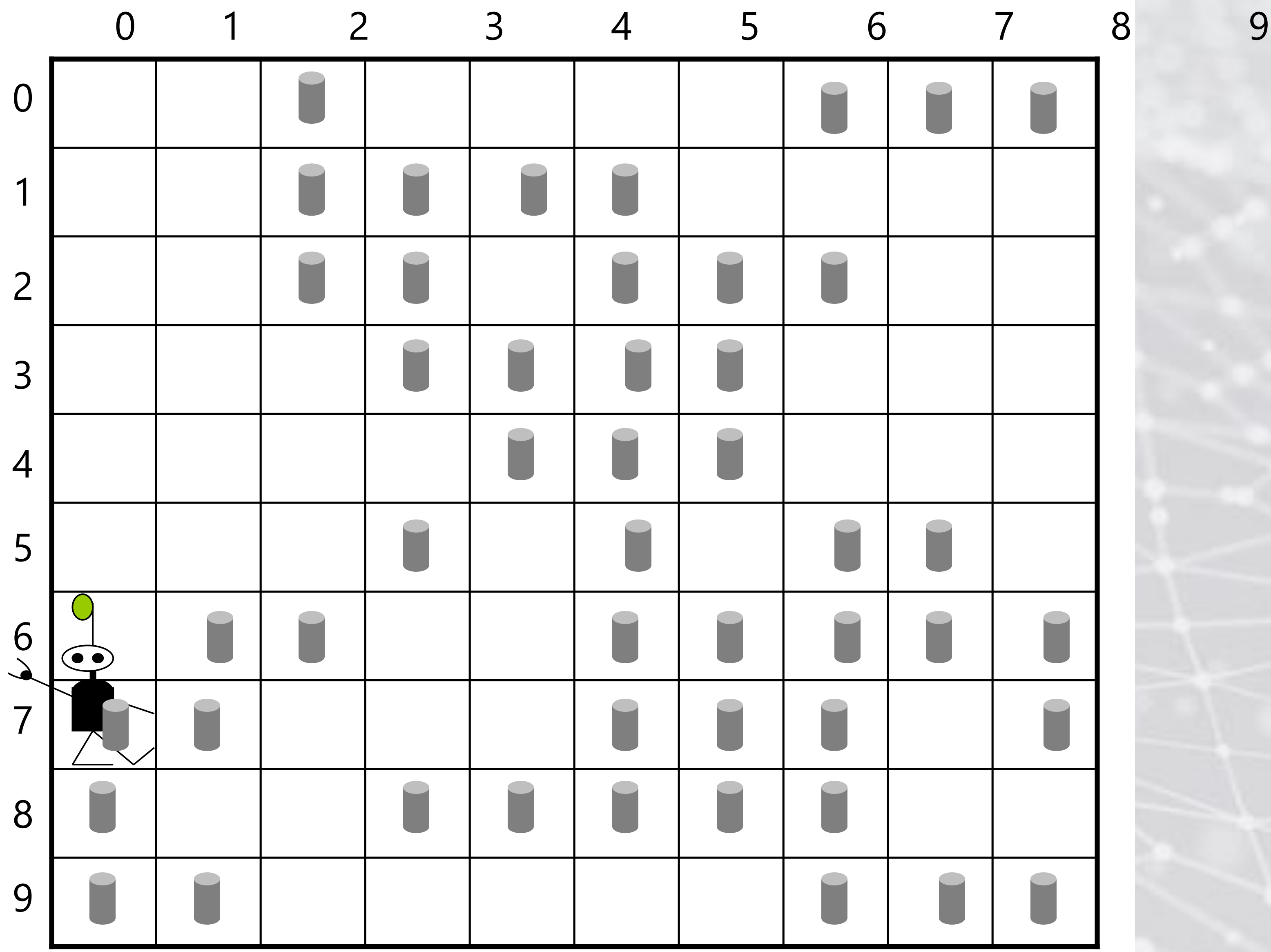
Time: 8 **Score: 20**



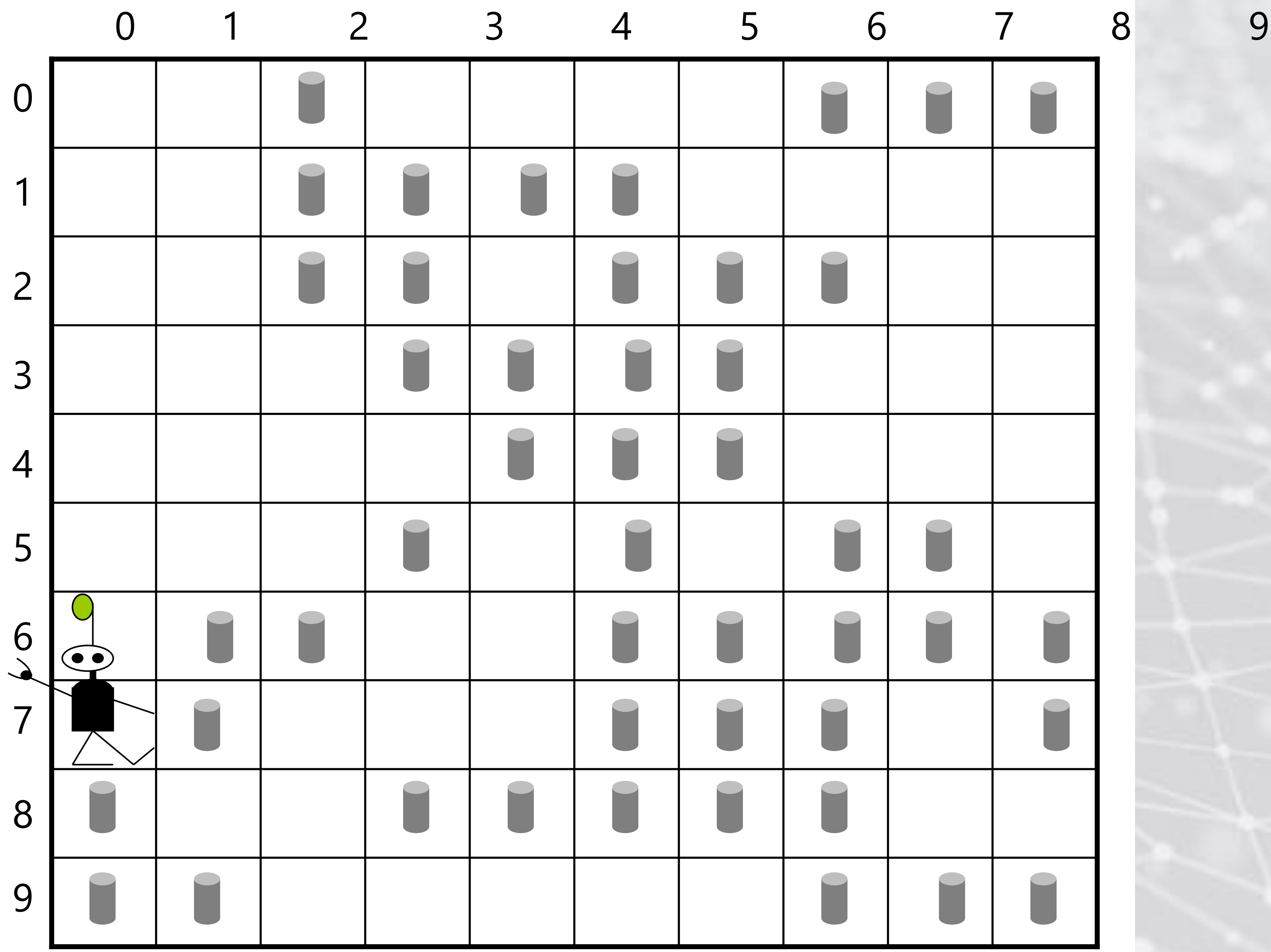
Time: 9 Score: 30



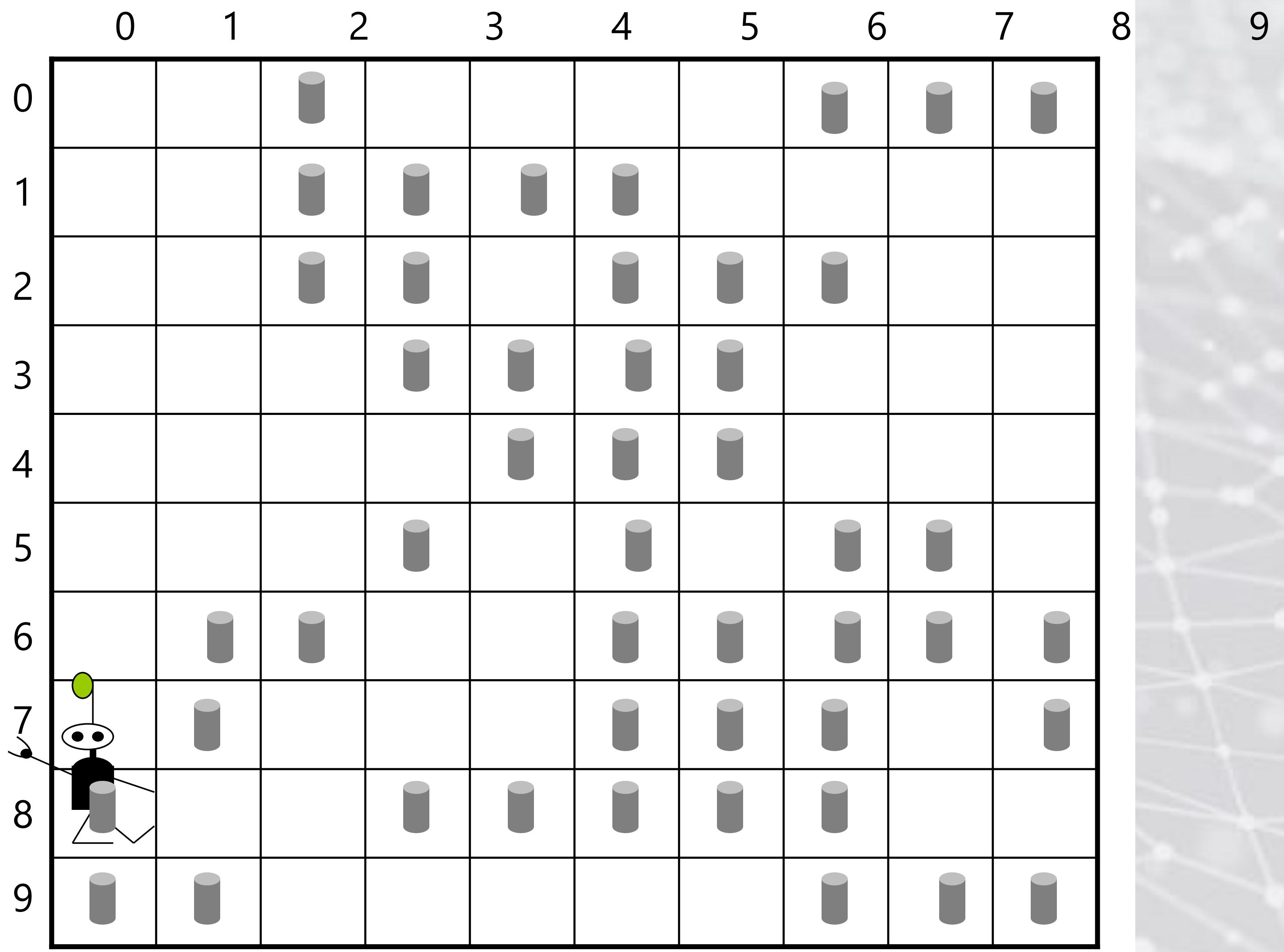
Time: 10 Score: 30



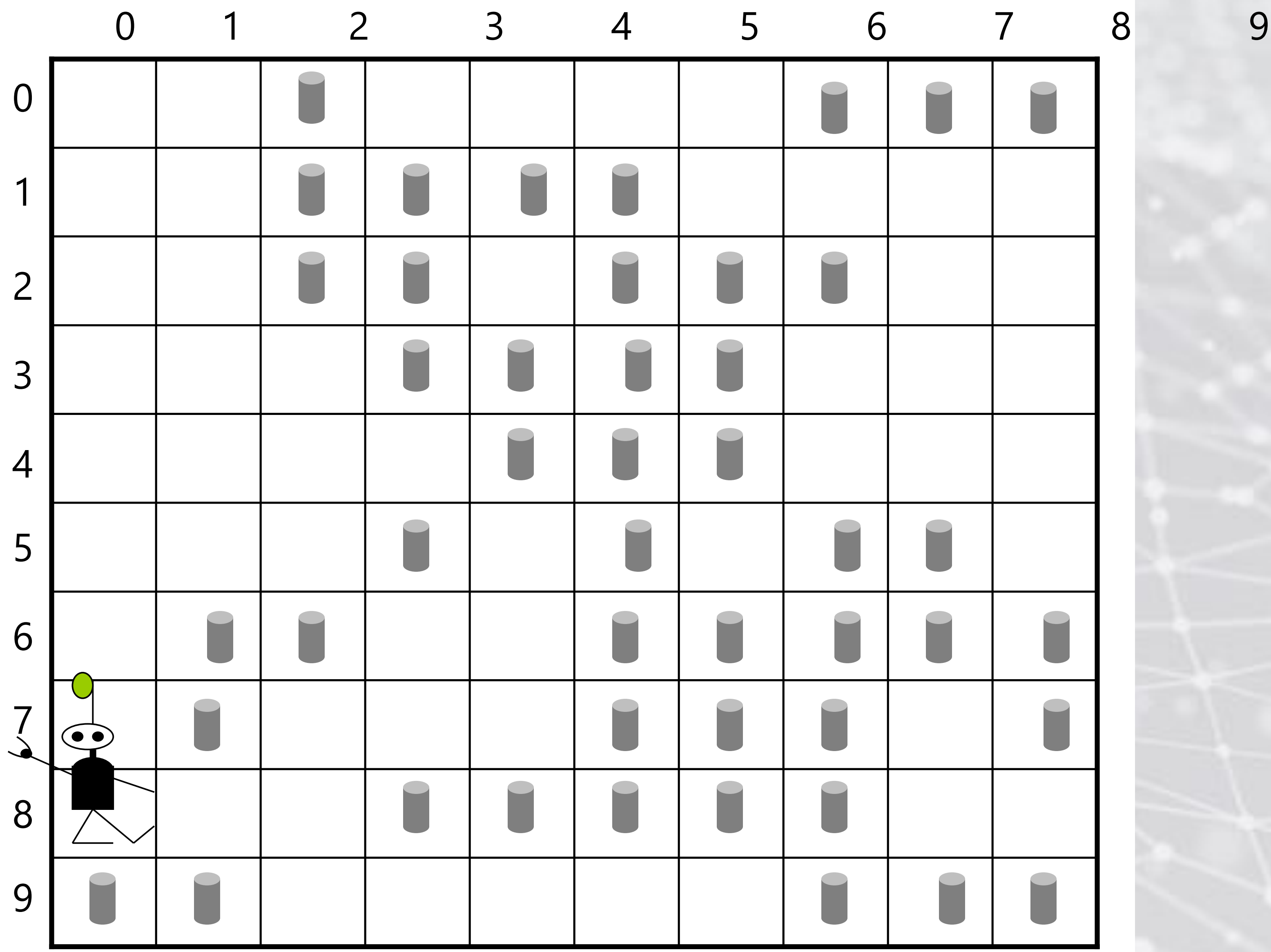
Time: 11 Score: 40



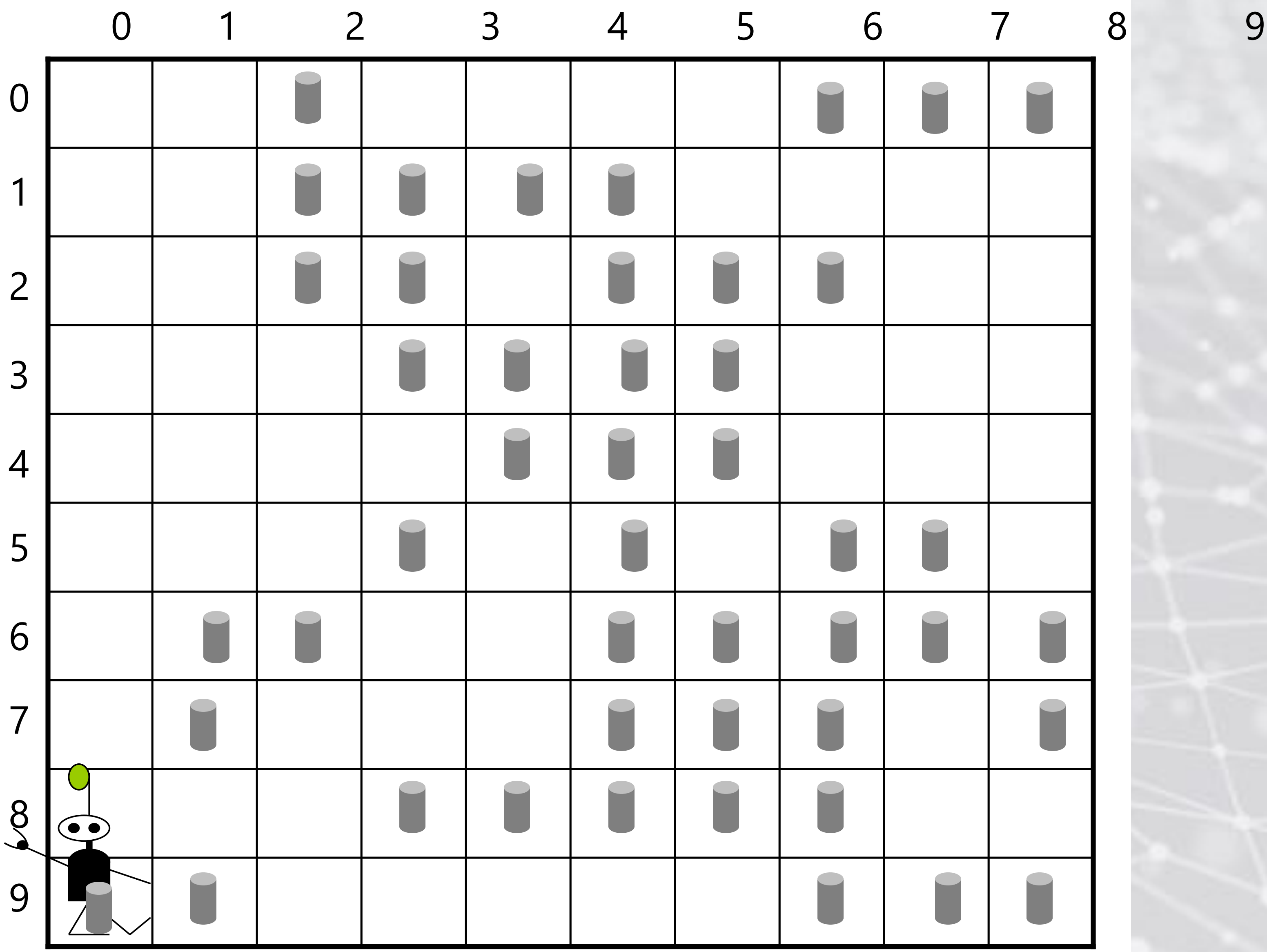
Time: 12 Score: 40



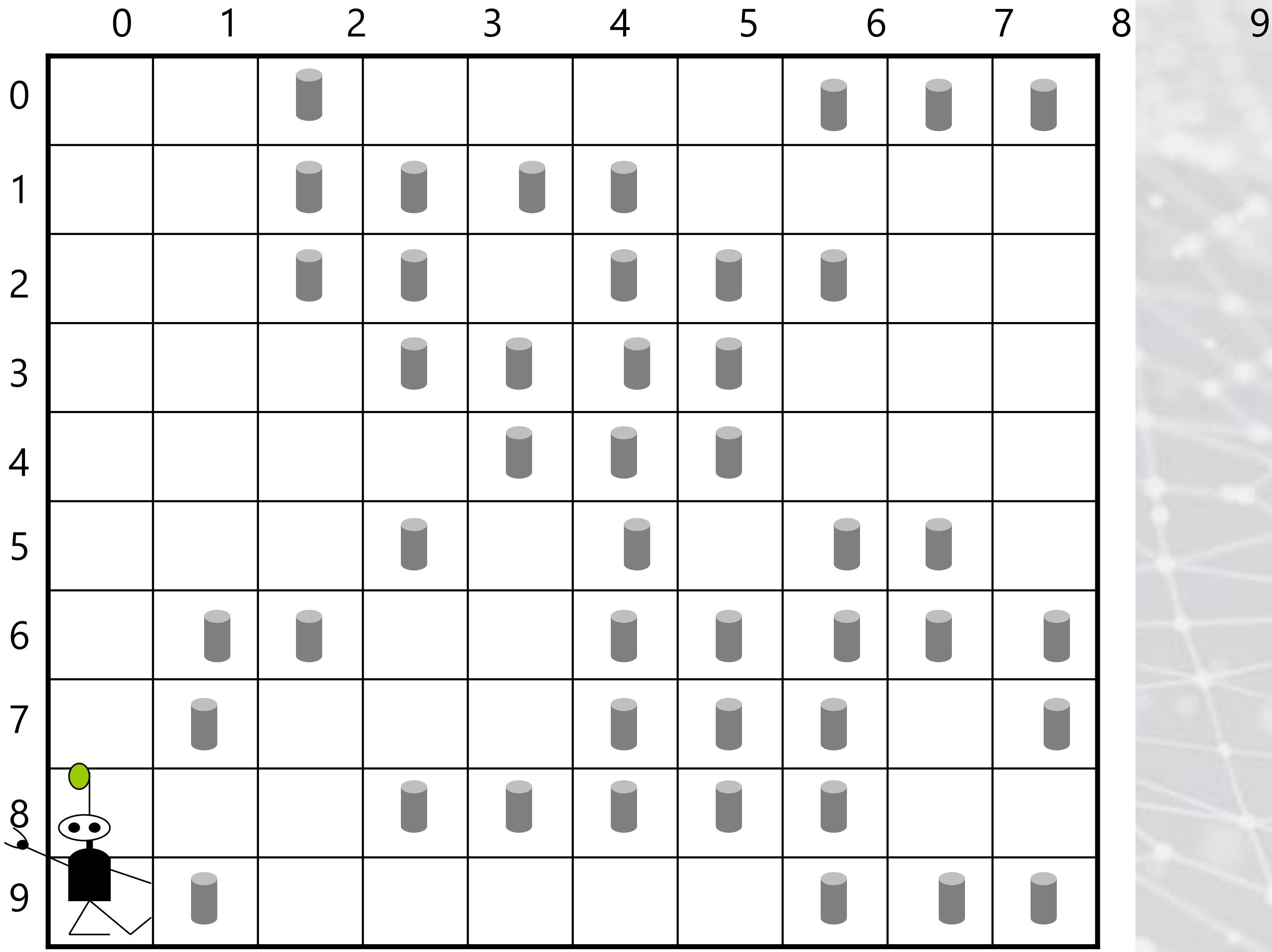
Time: 13 **Score: 50**



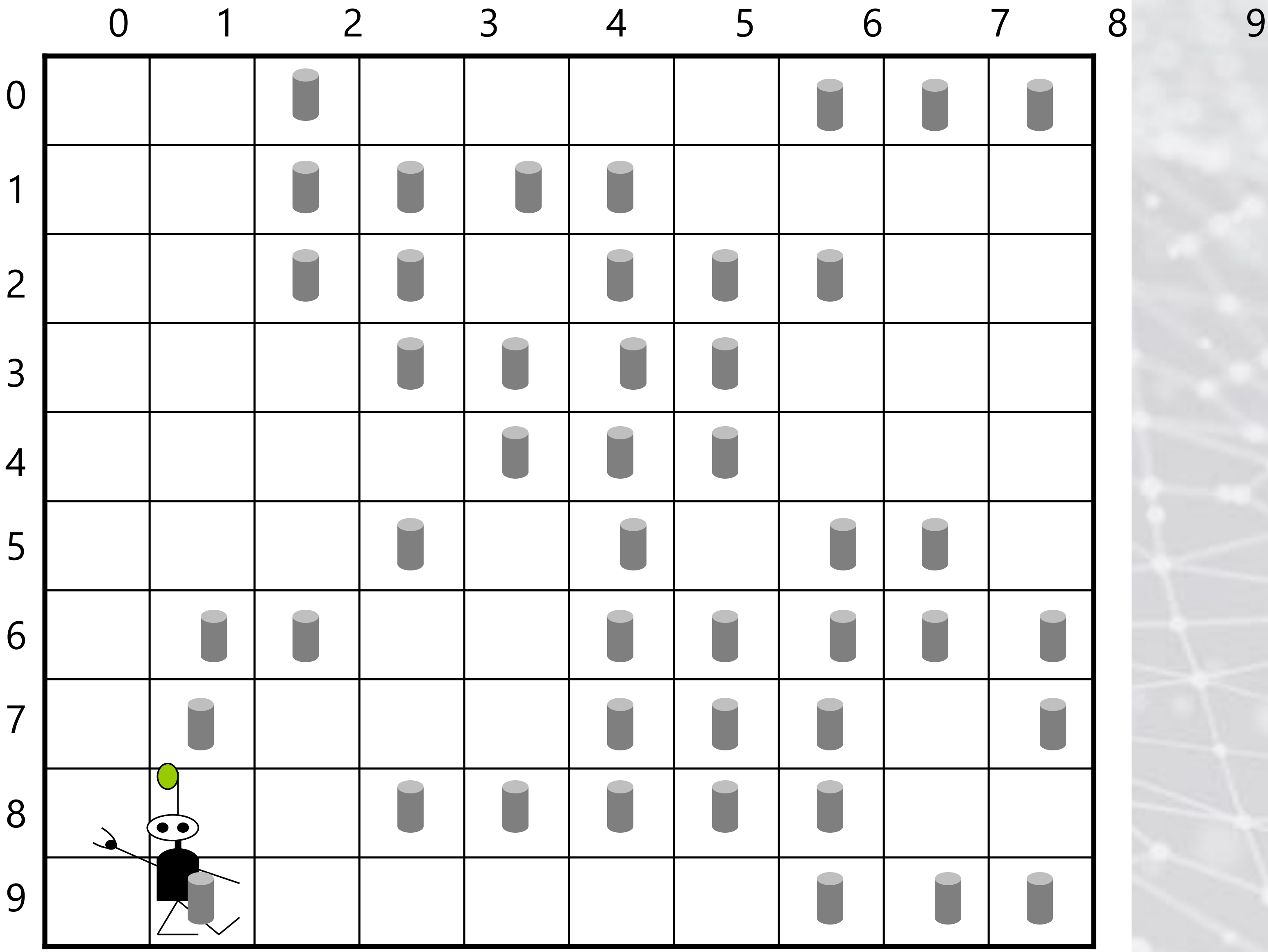
Time: 14 Score: 50



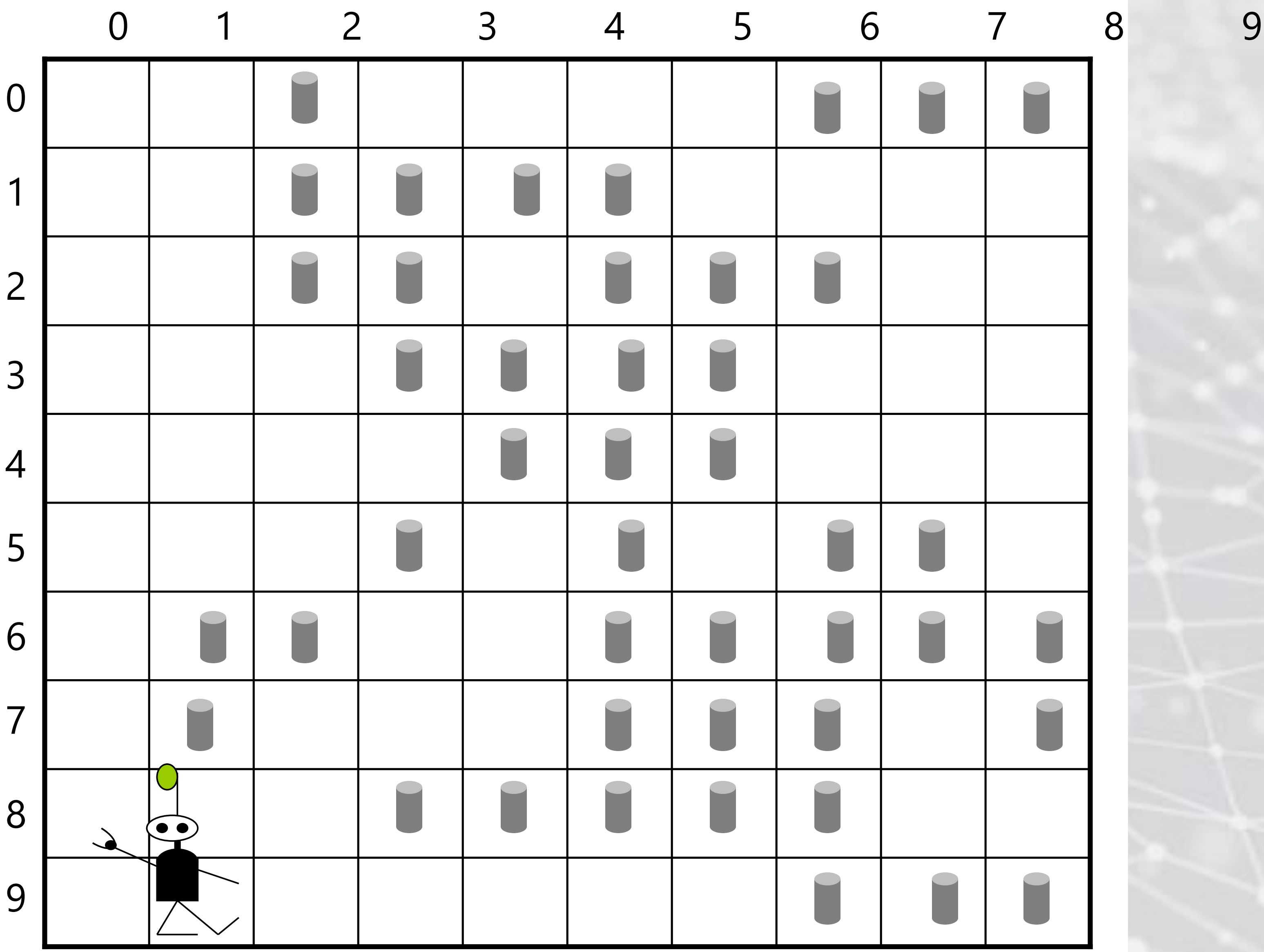
Time: 15 **Score: 60**



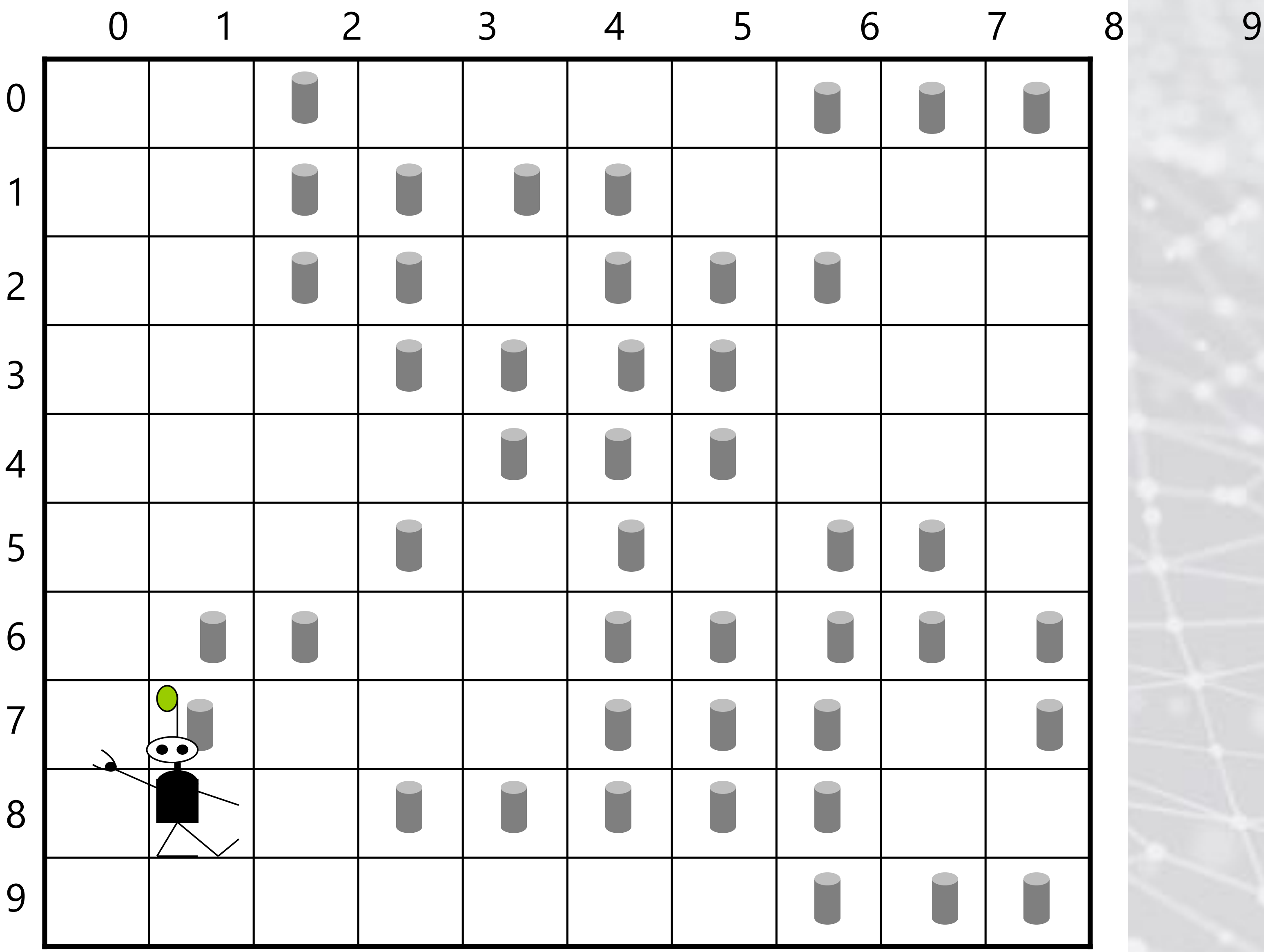
Time: 16 Score: 60



Time: 17 Score: 70

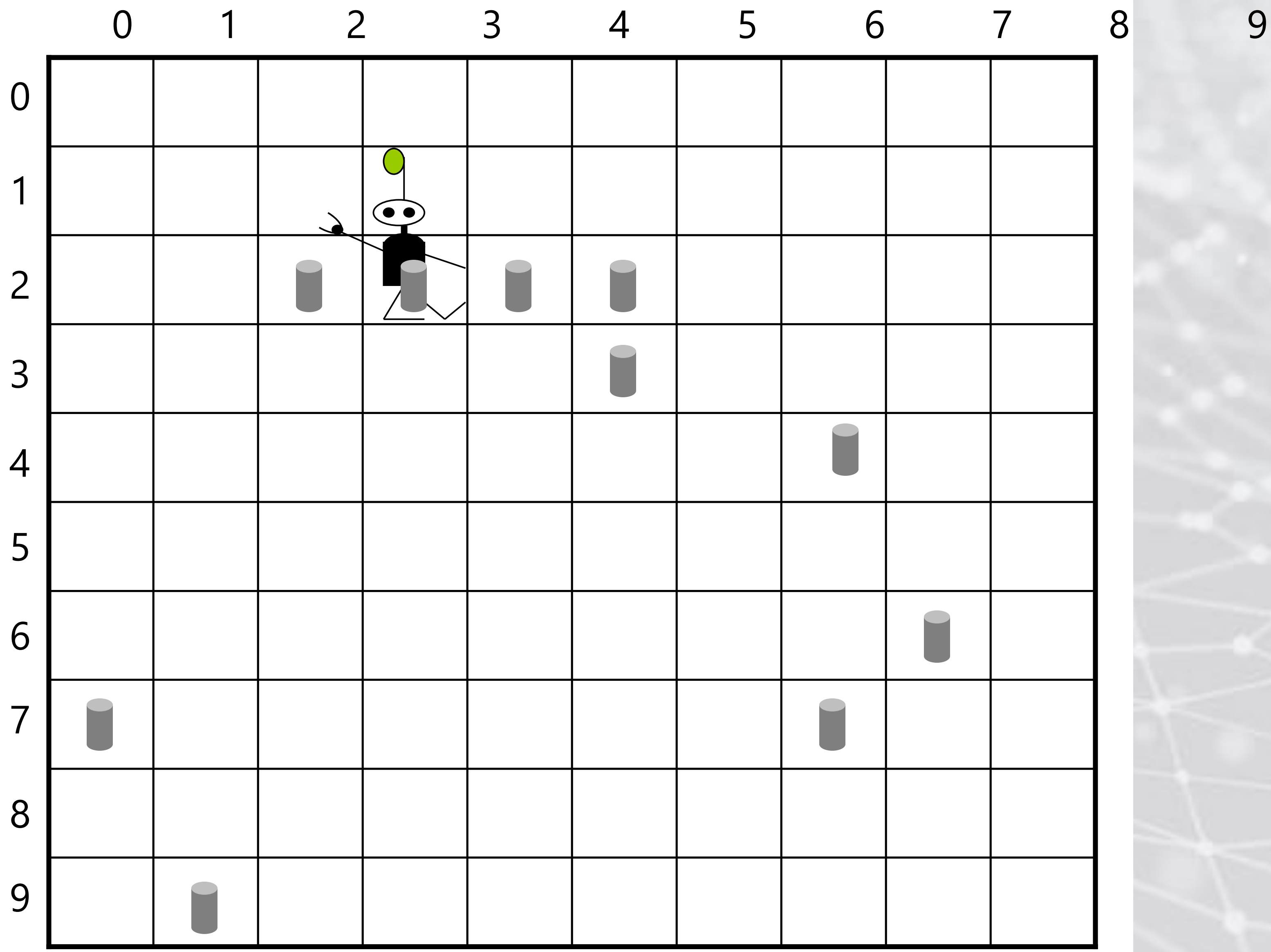


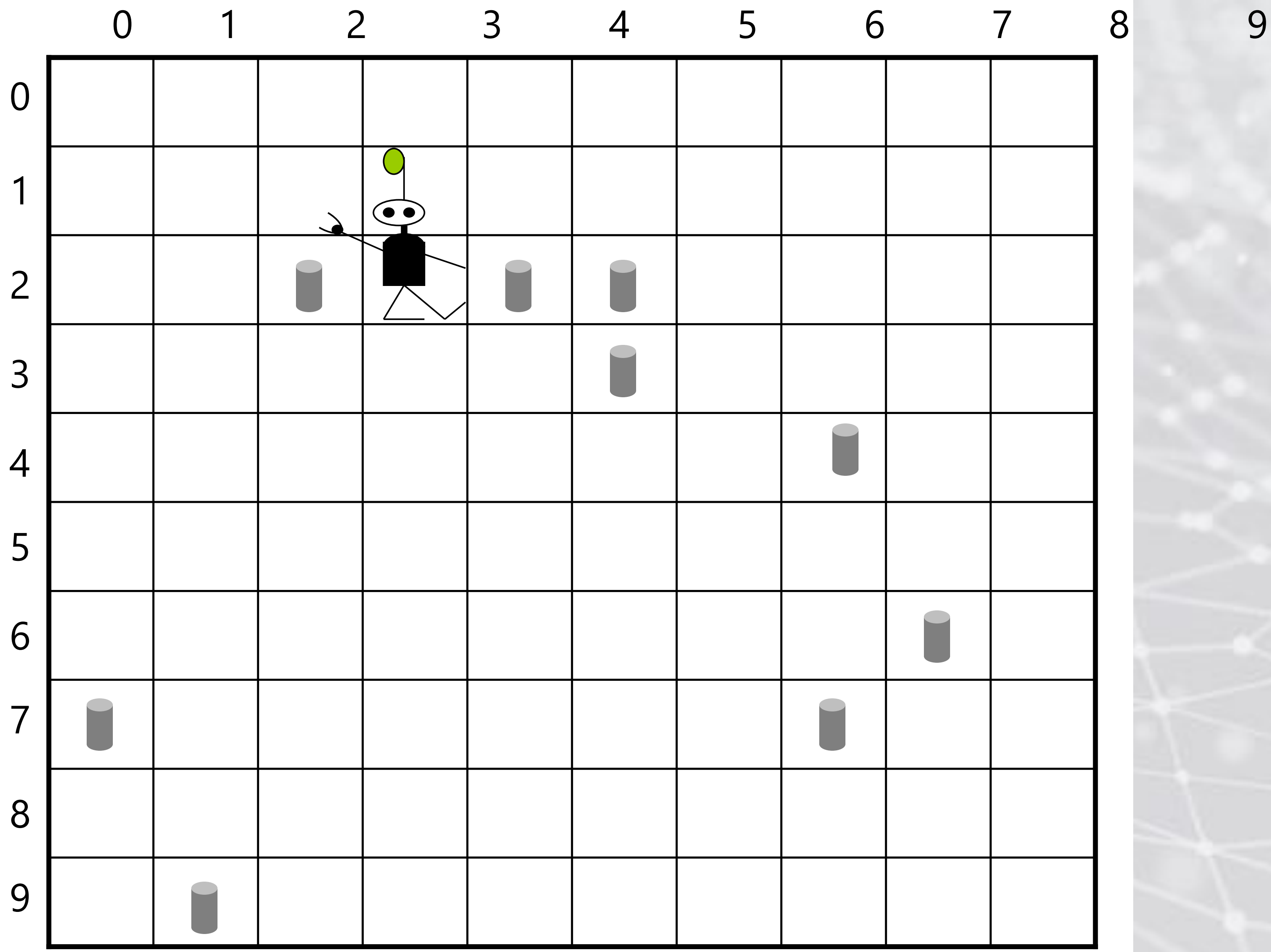
Time: 18 Score: 70

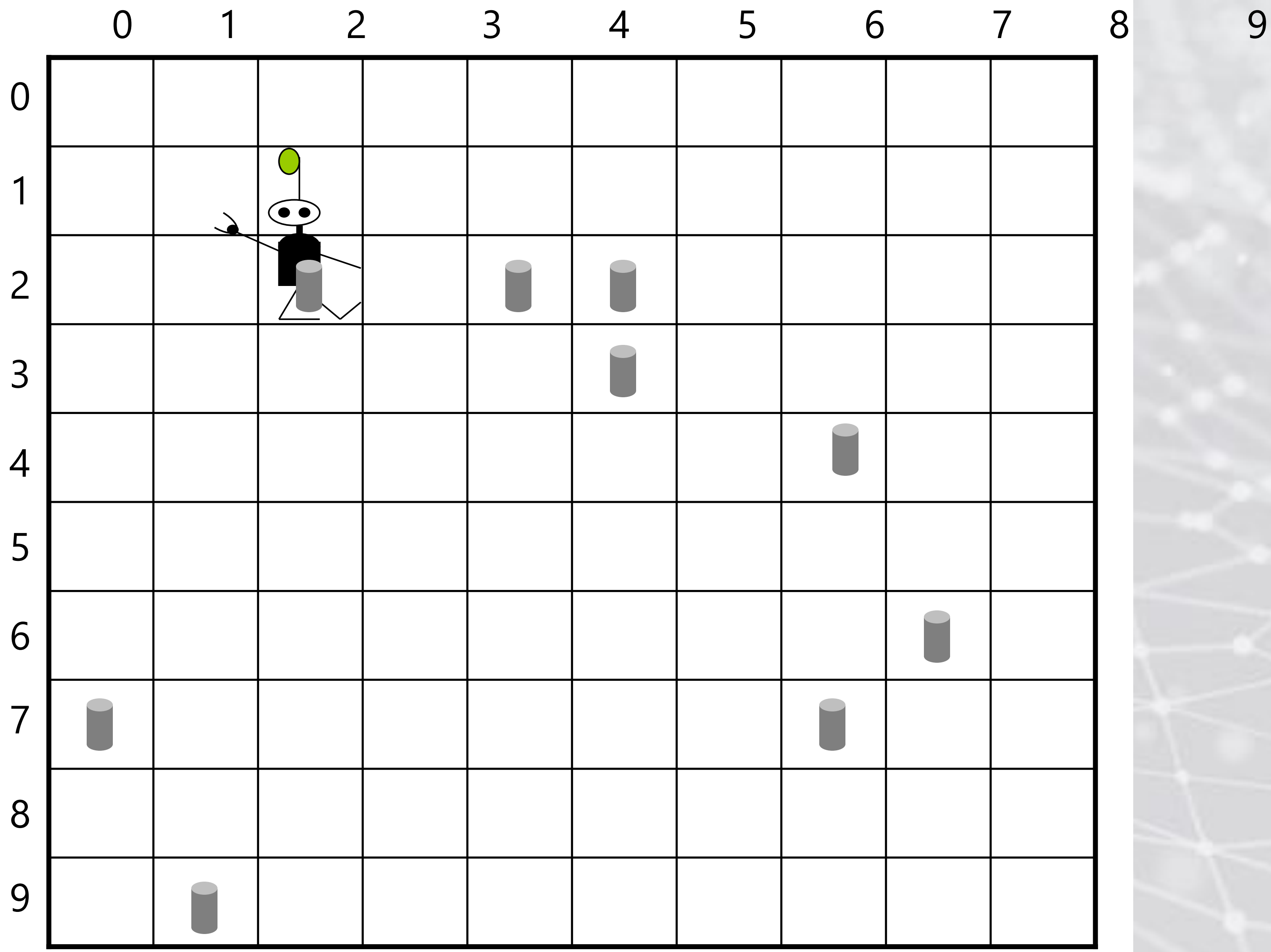


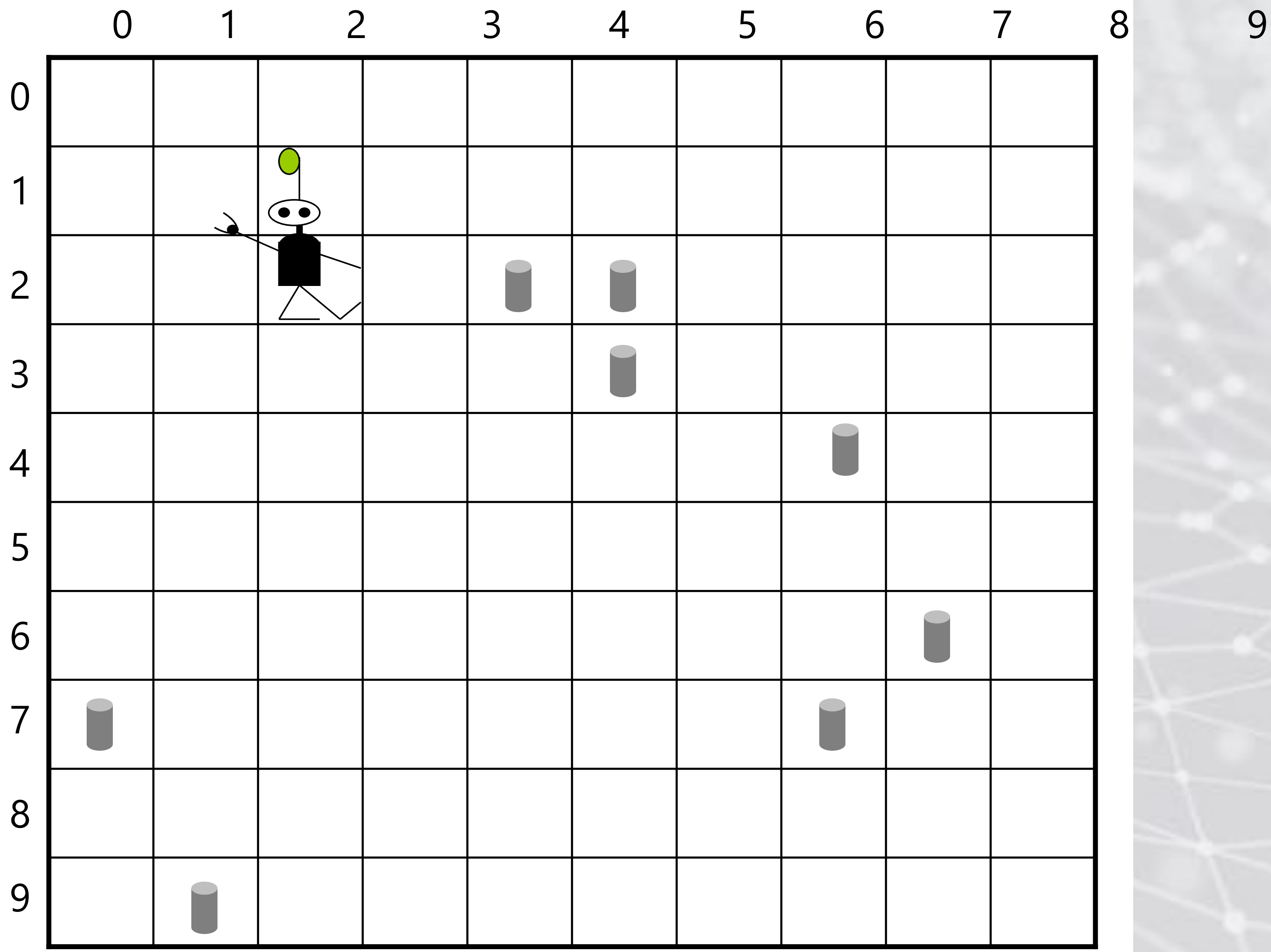
Why Did The GA's Strategy Outperform Mine?

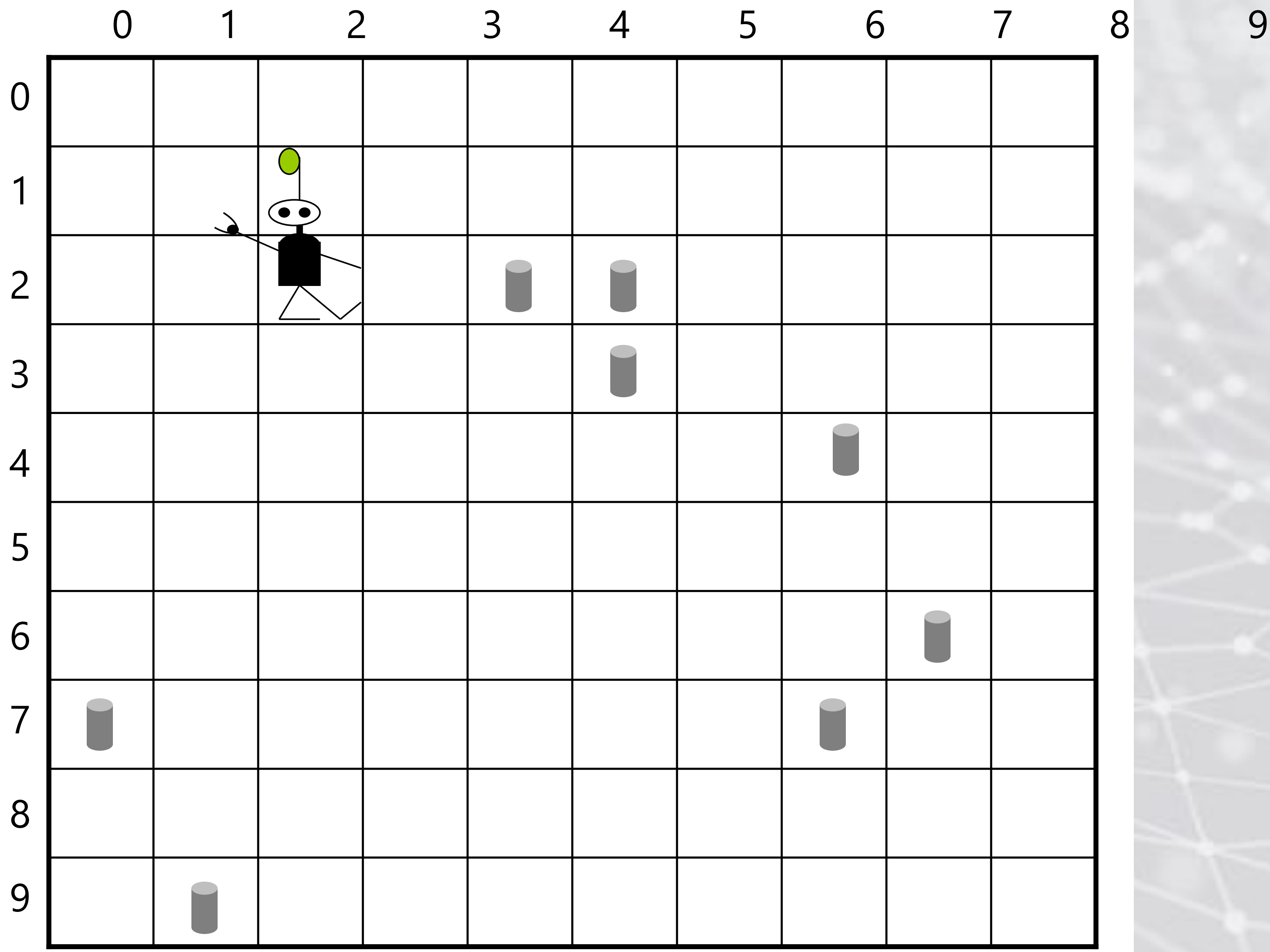
My Strategy


0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

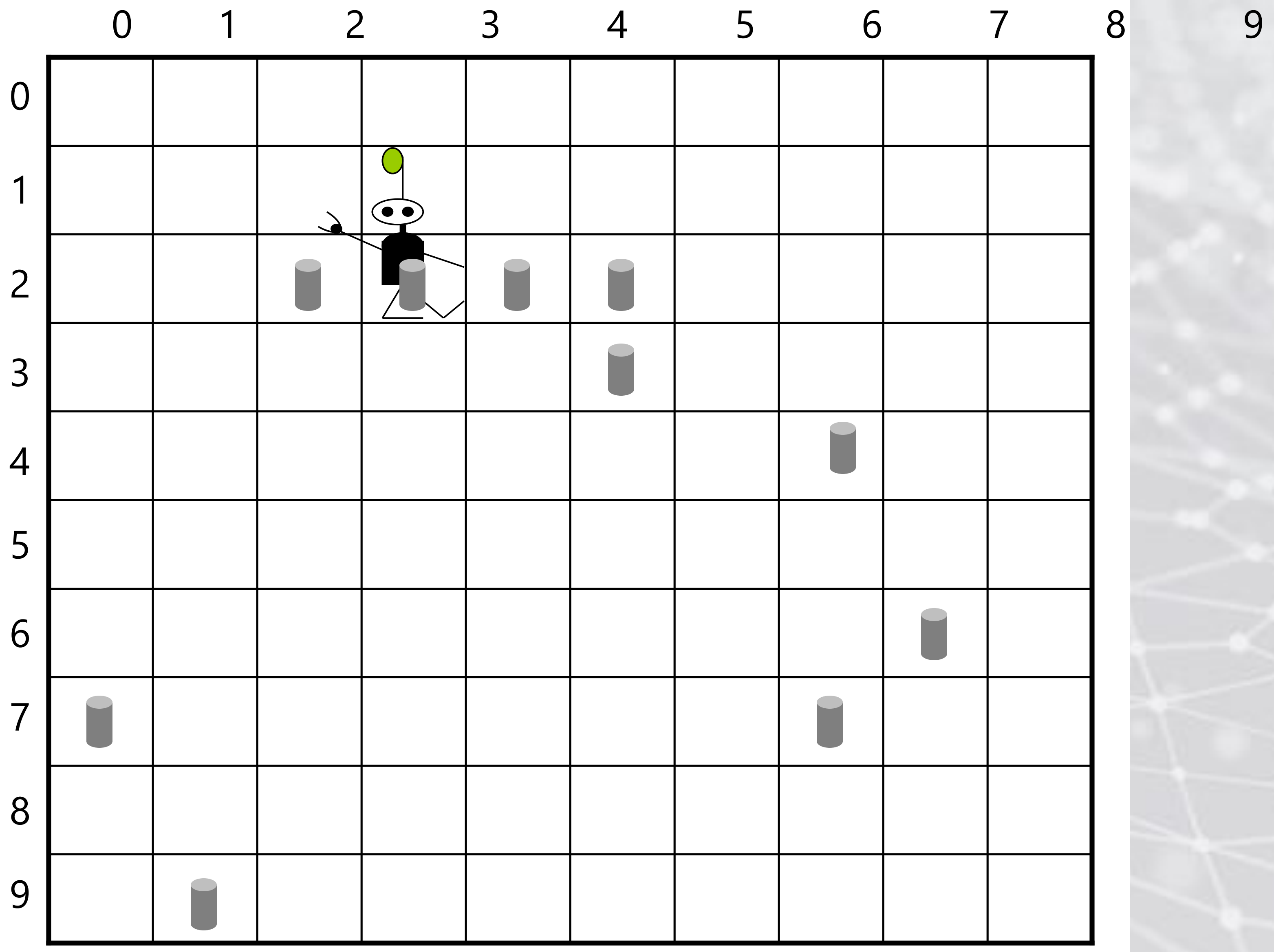

0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9

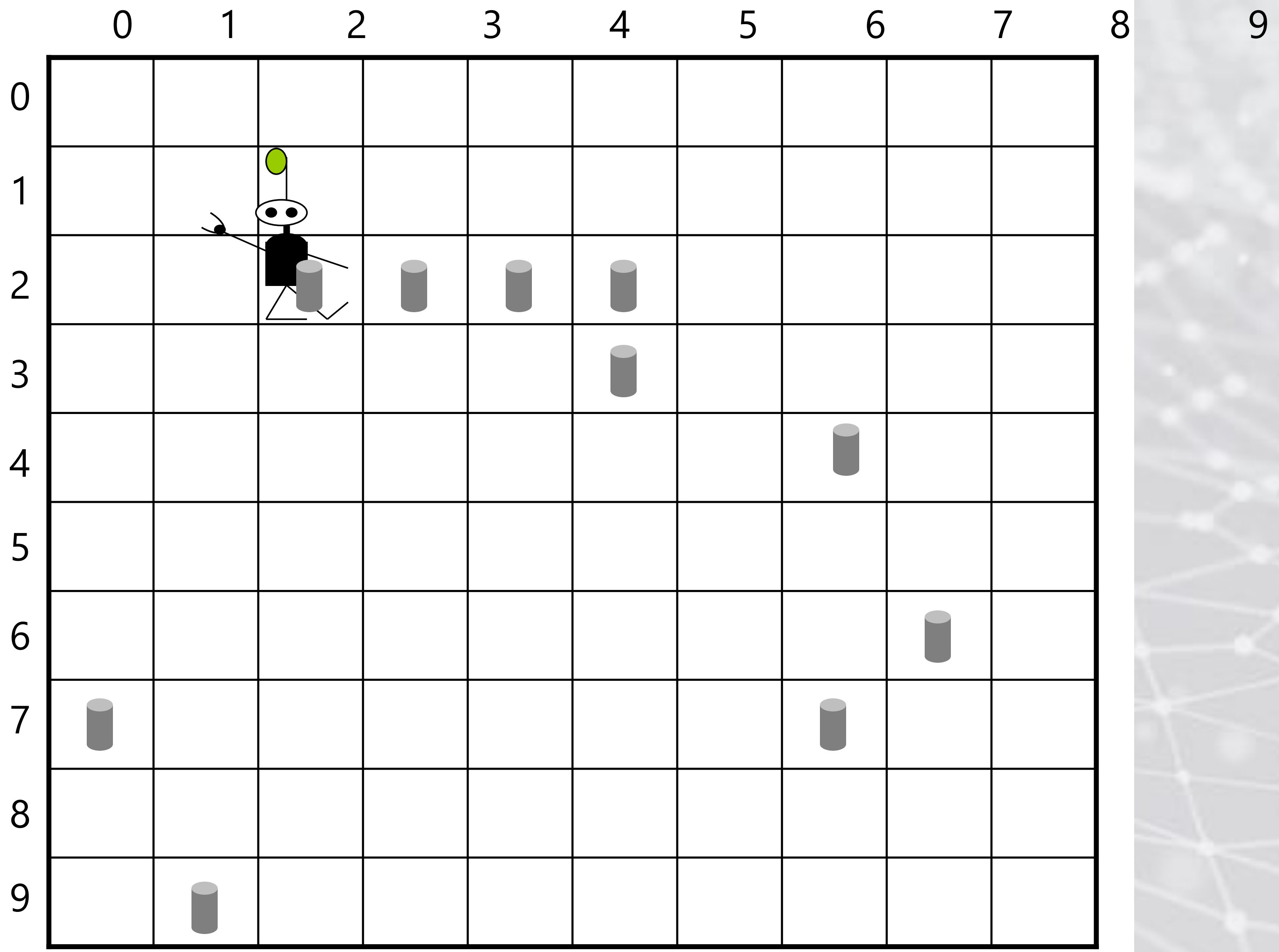

0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9


0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

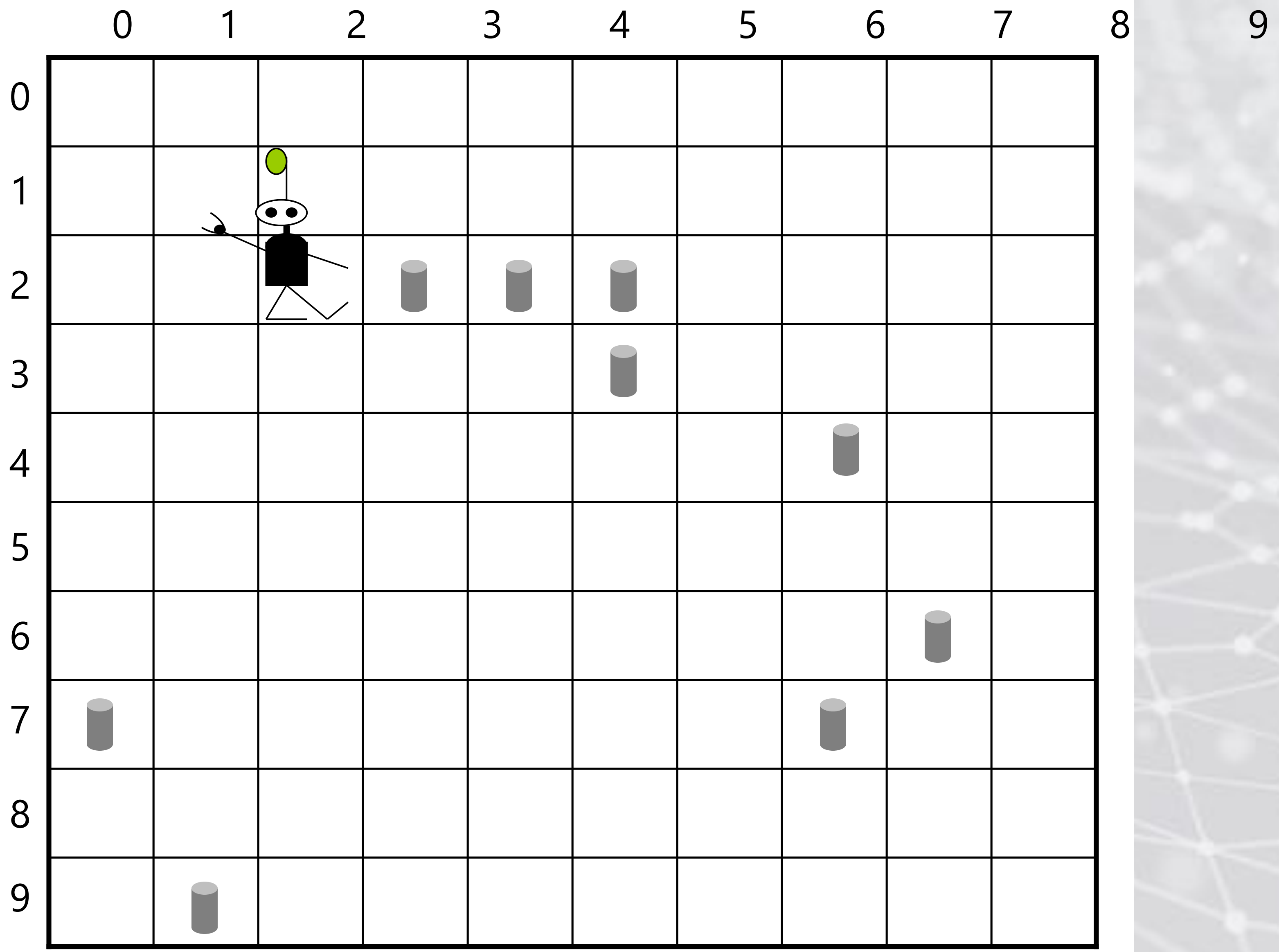

0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9

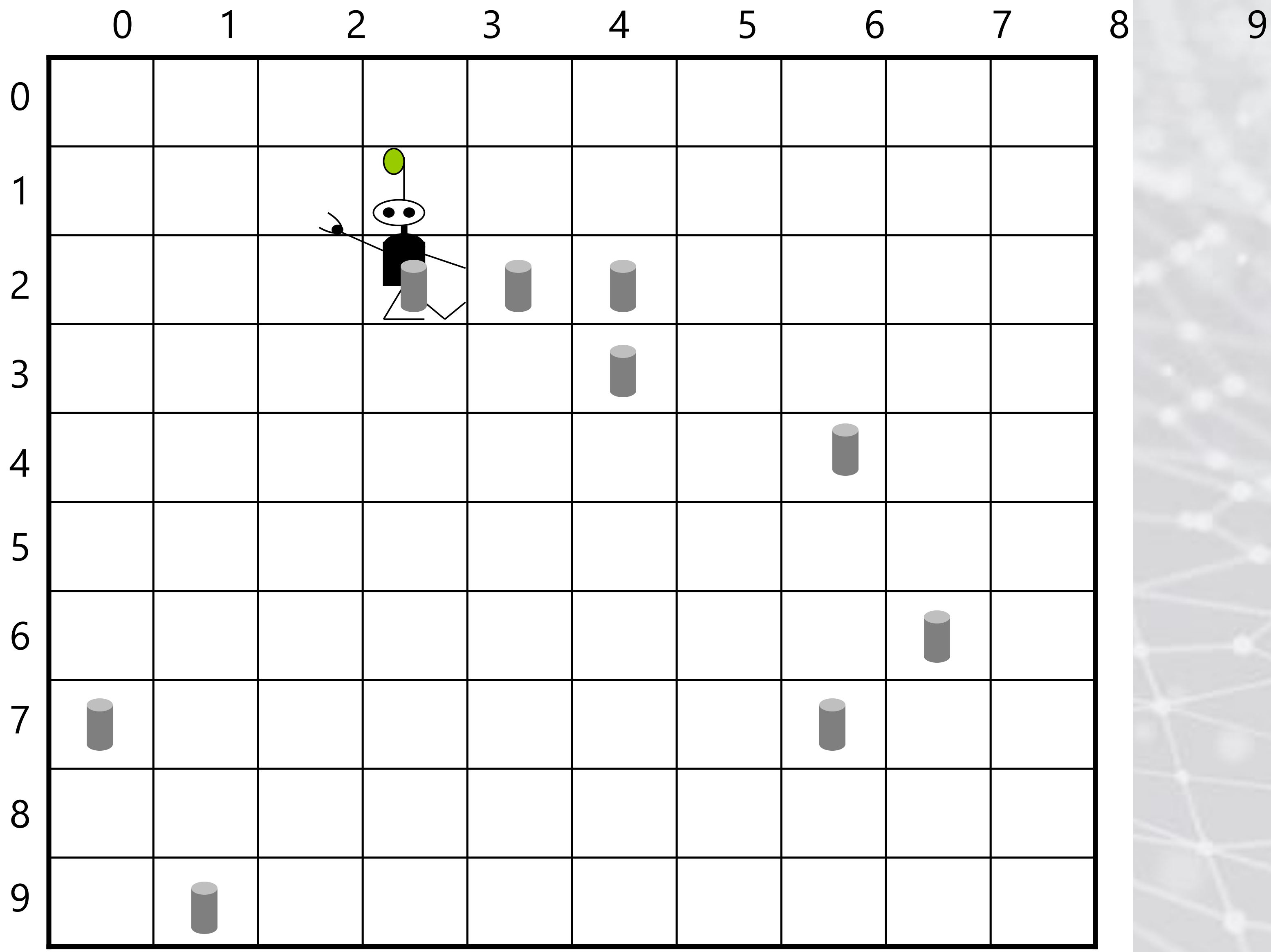
The GA's Evolved Strategy


0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9

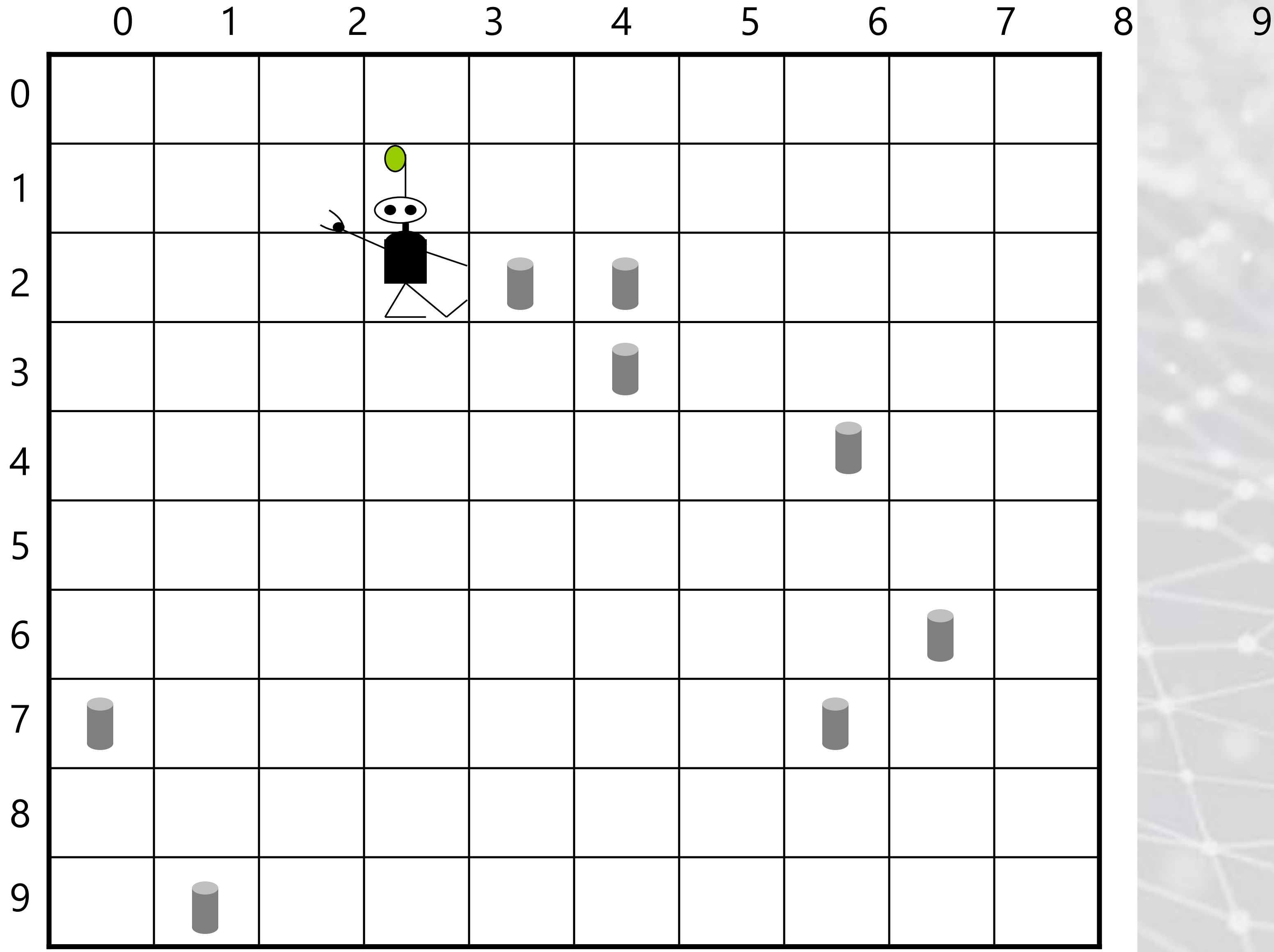


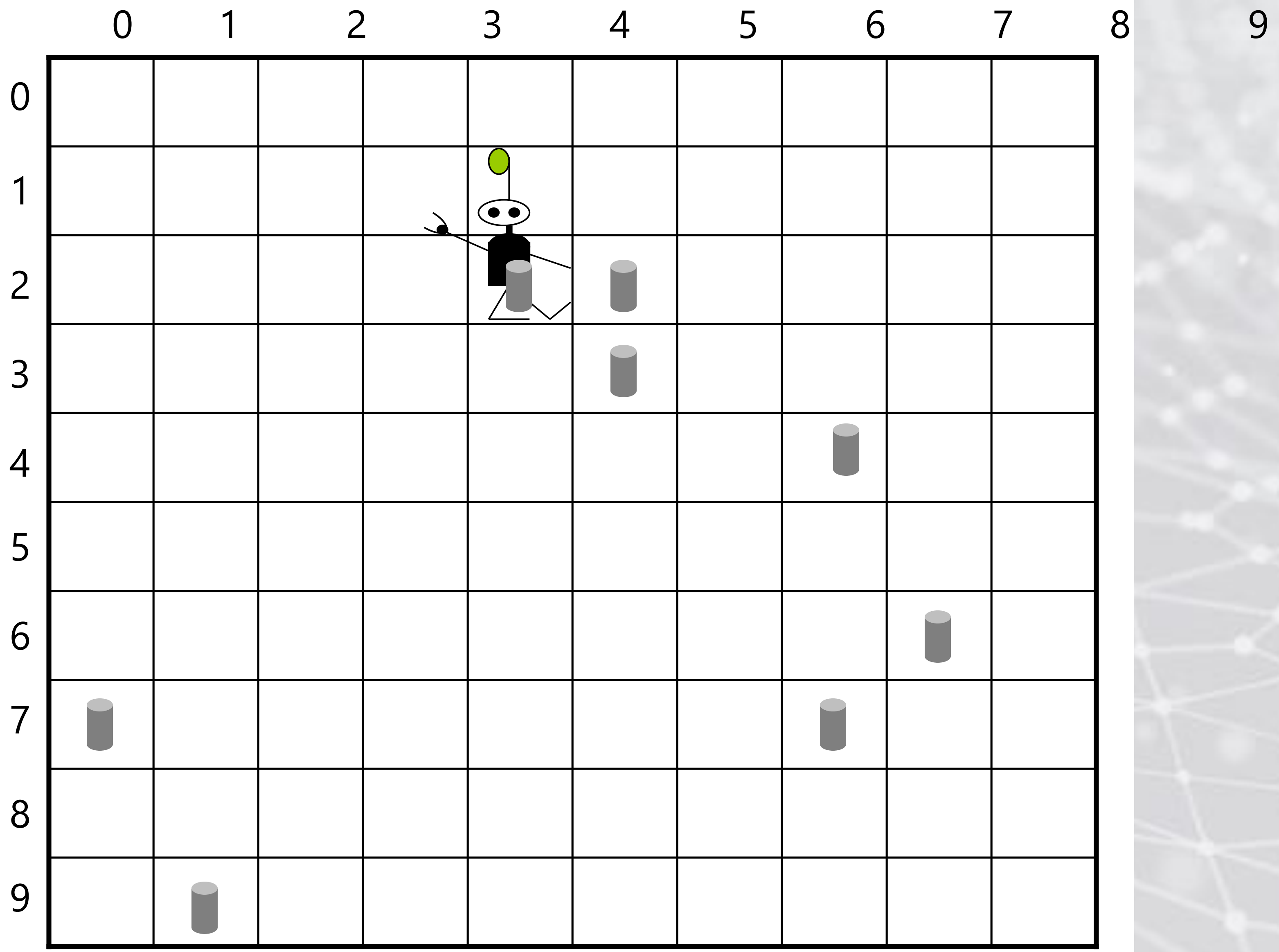
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9


0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9

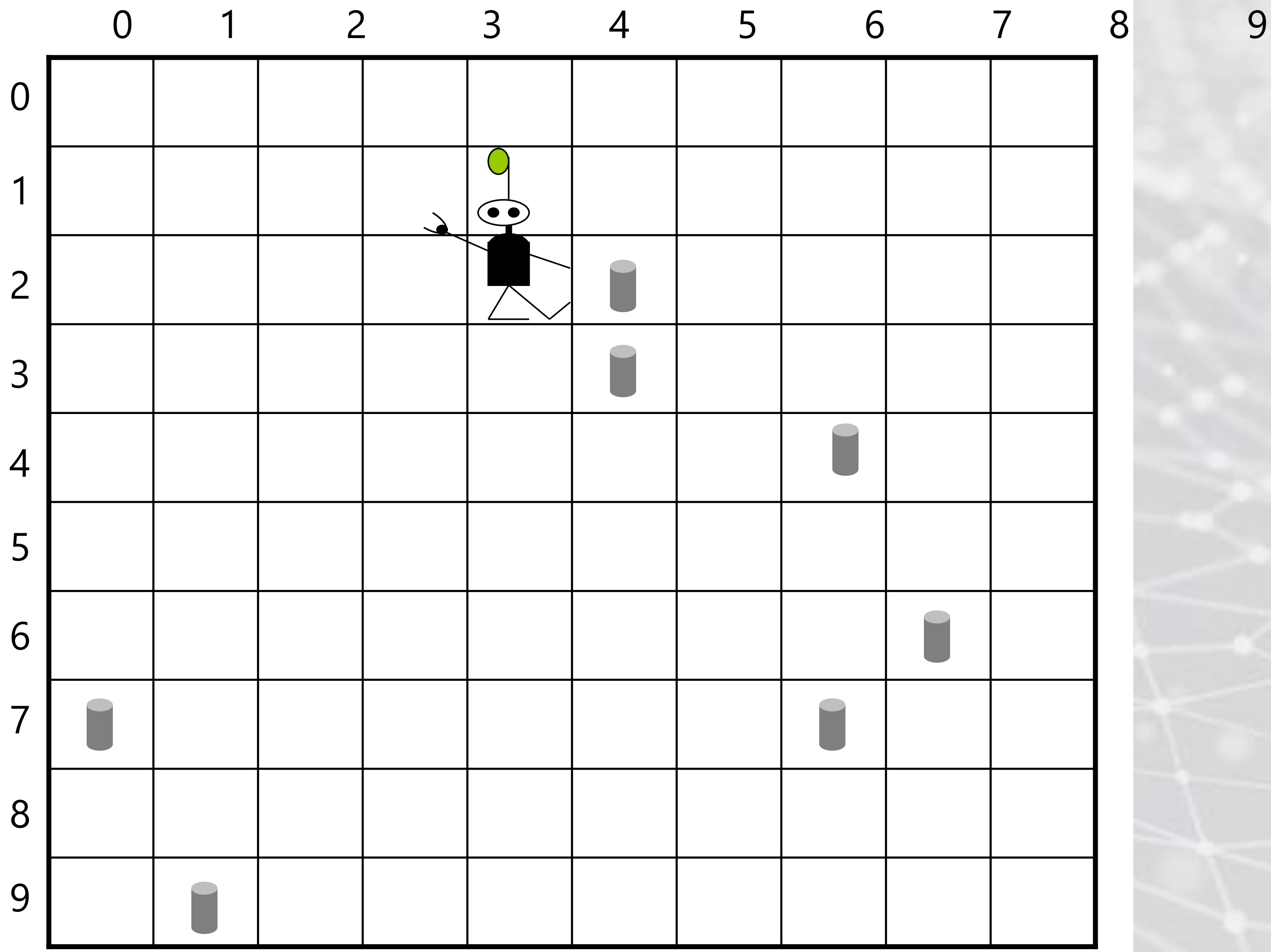


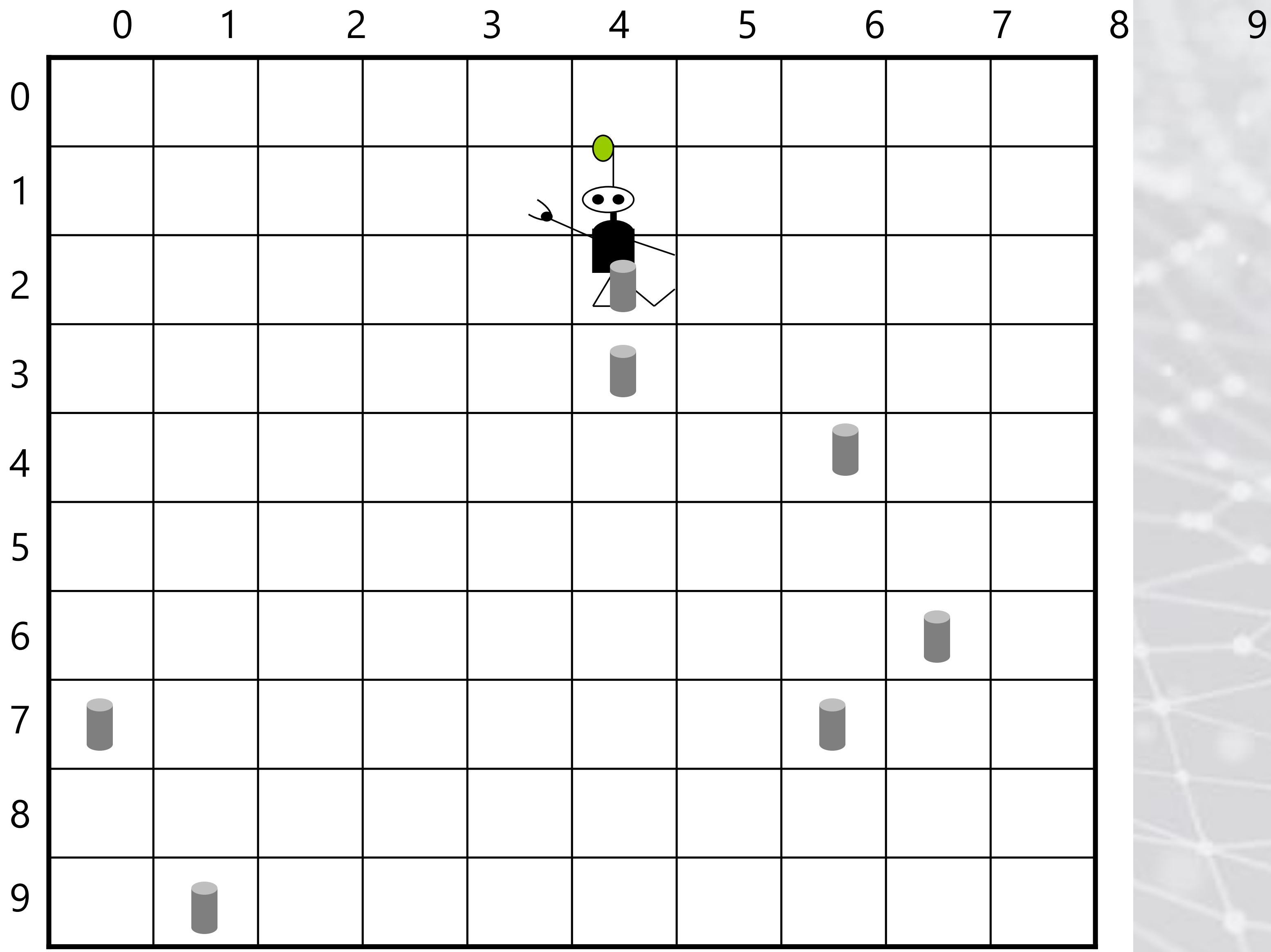
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9


0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9

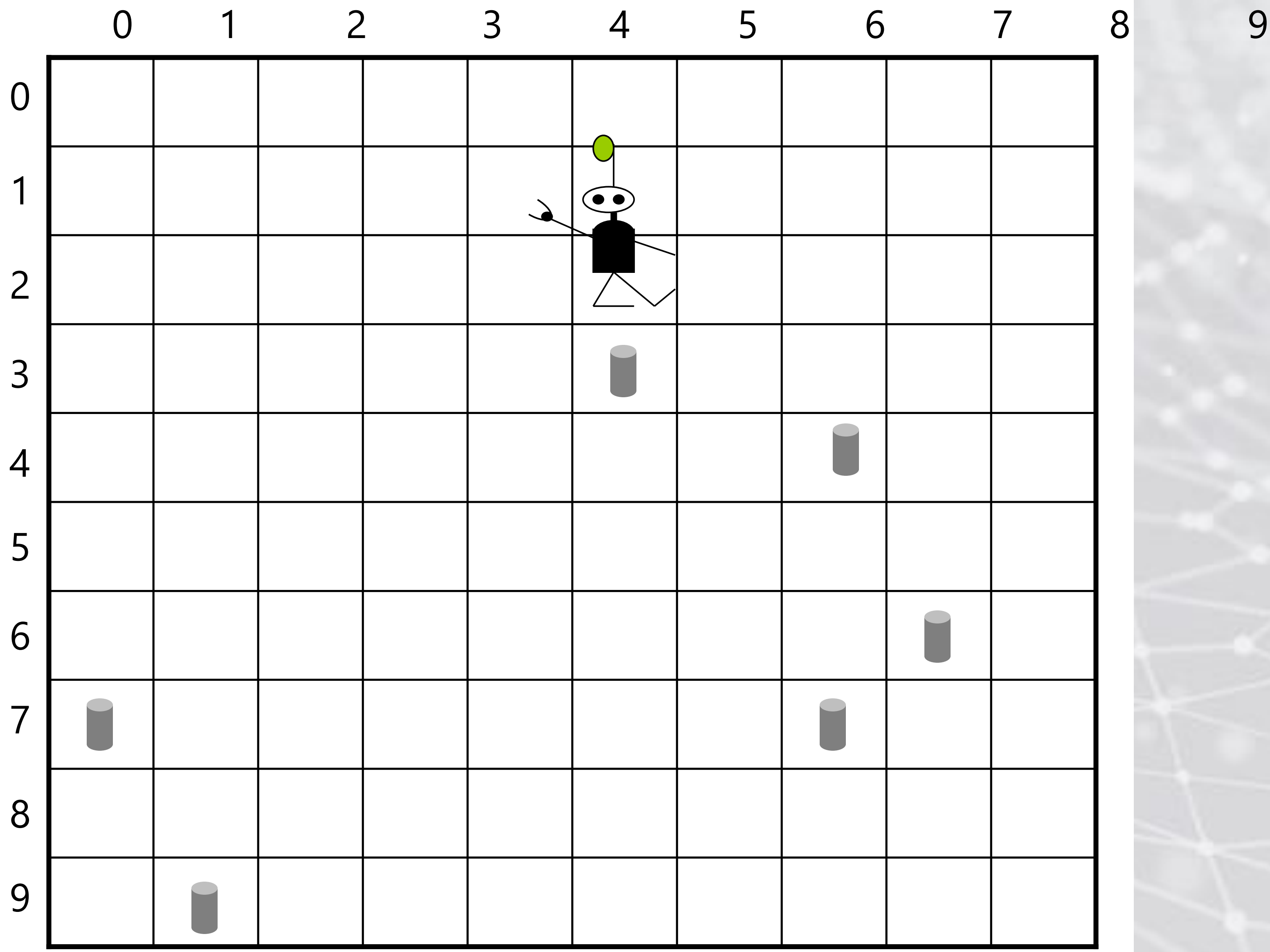


0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9

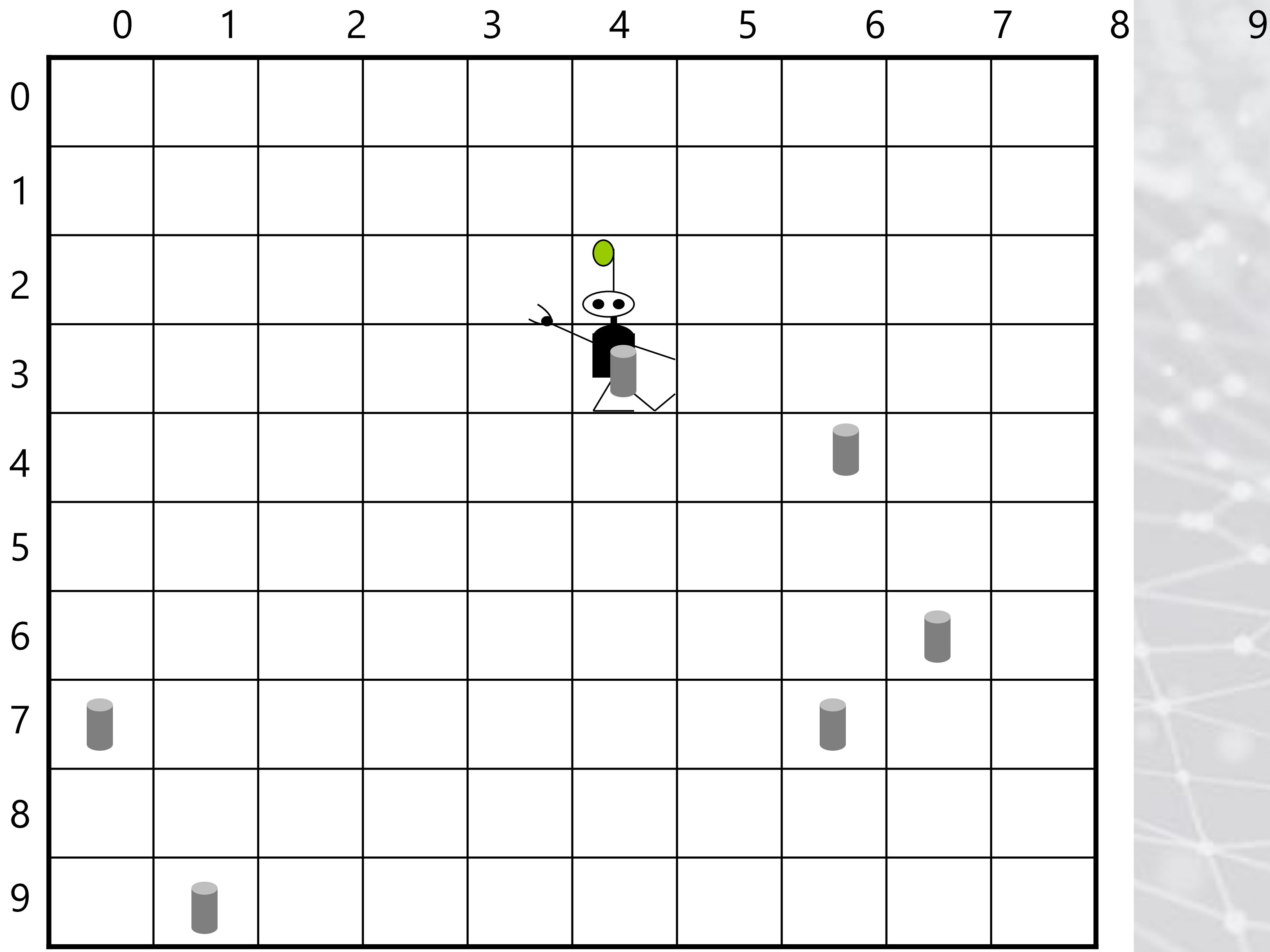

0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9

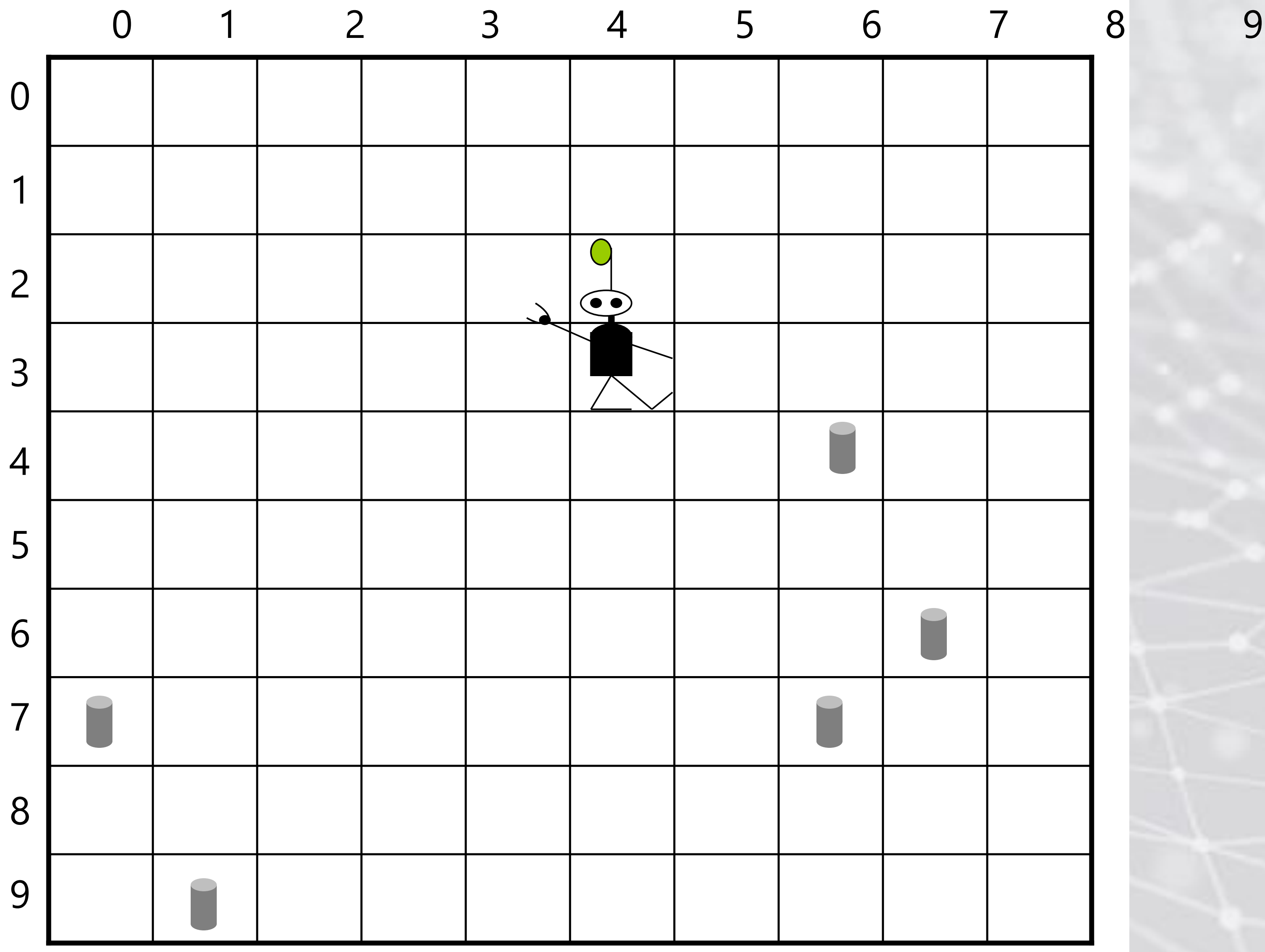


0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9



0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9


0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9


0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9

Genetic Algorithms, Part 2

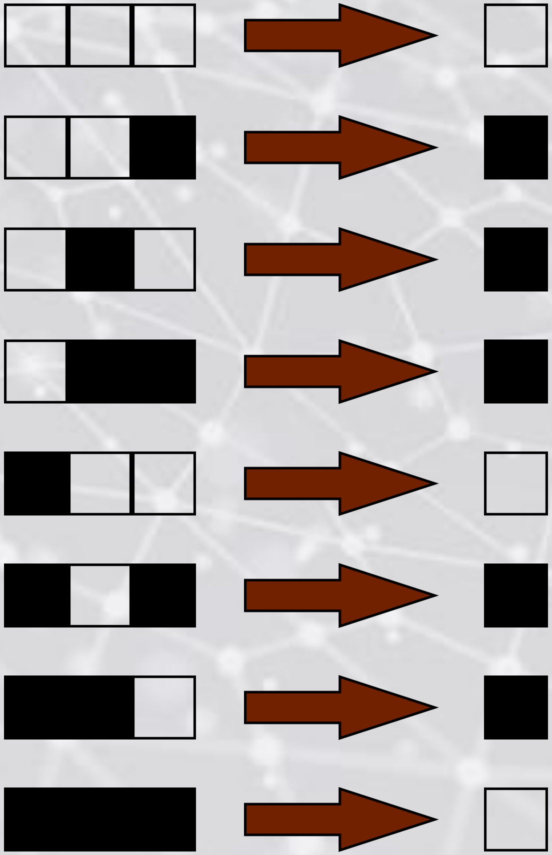
(Application to Cellular Automata
-- Bonus material)

Evolving (and co-evolving) one-dimensional cellular automata to perform a computation

One-dimensional cellular automata

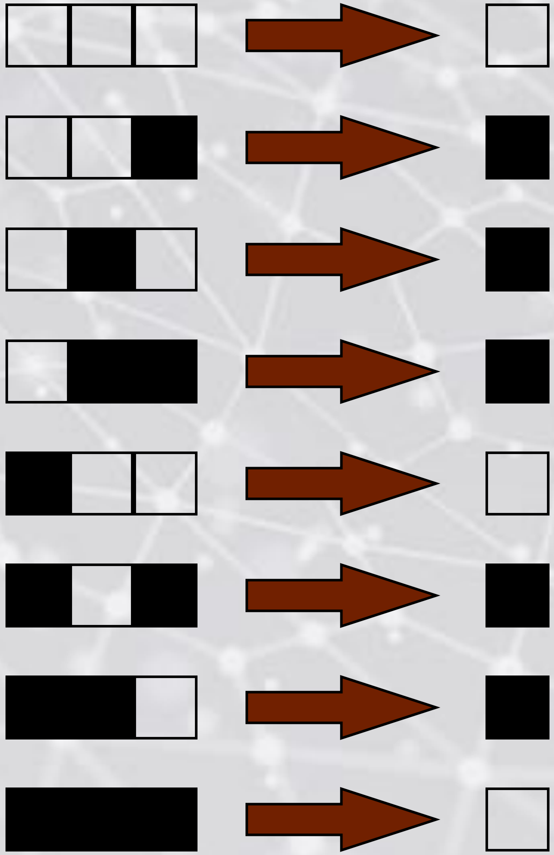
One-dimensional cellular automata

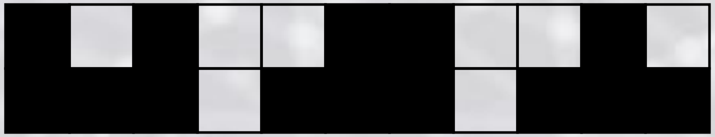
Rule:



One-dimensional cellular automata

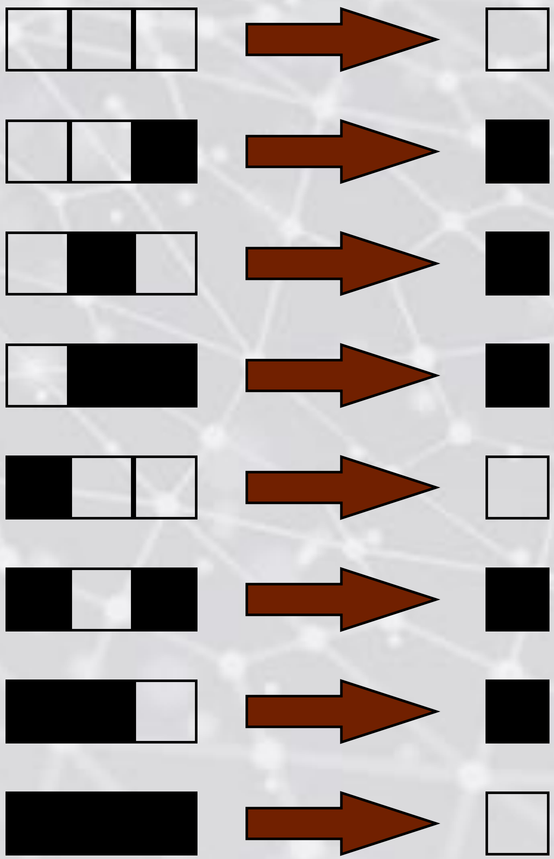
Rule:

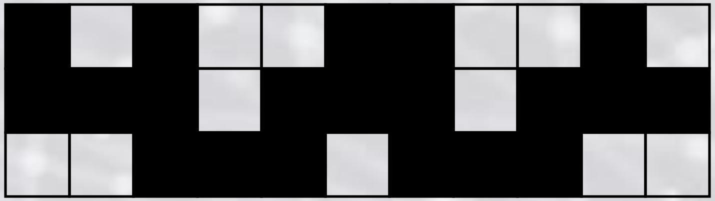




One-dimensional cellular automata

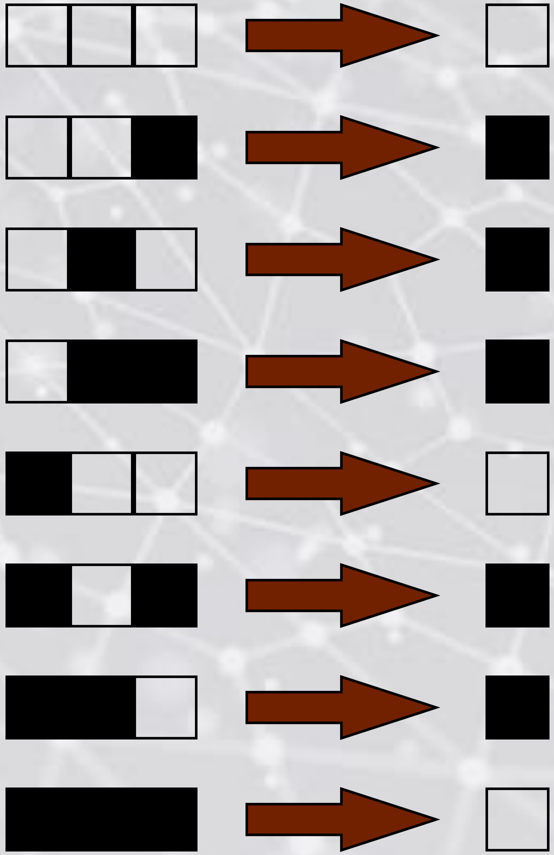
Rule:

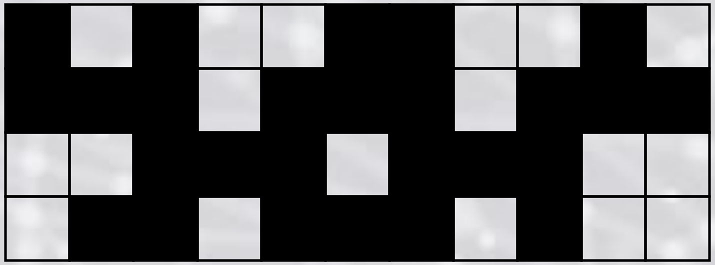




One-dimensional cellular automata

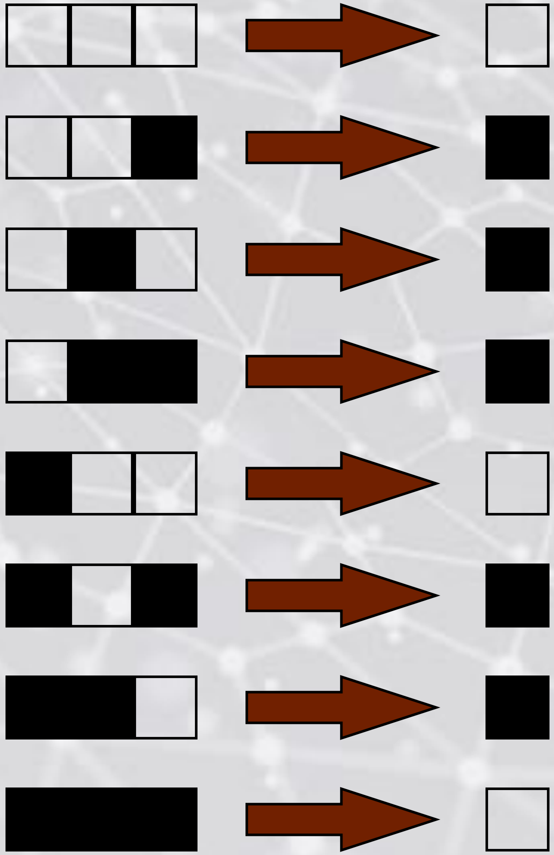
Rule:

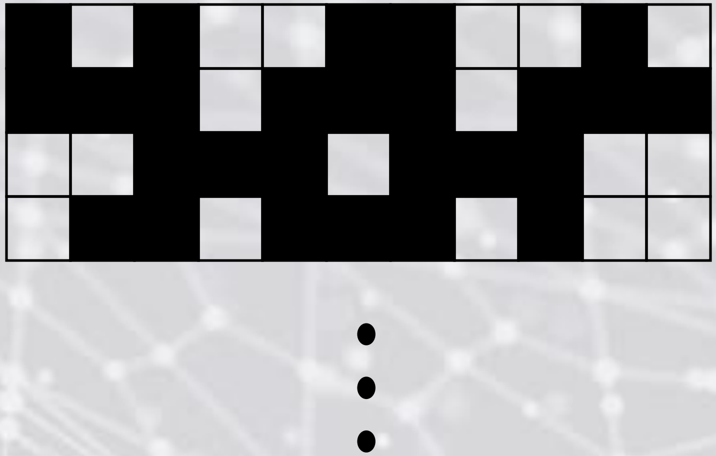


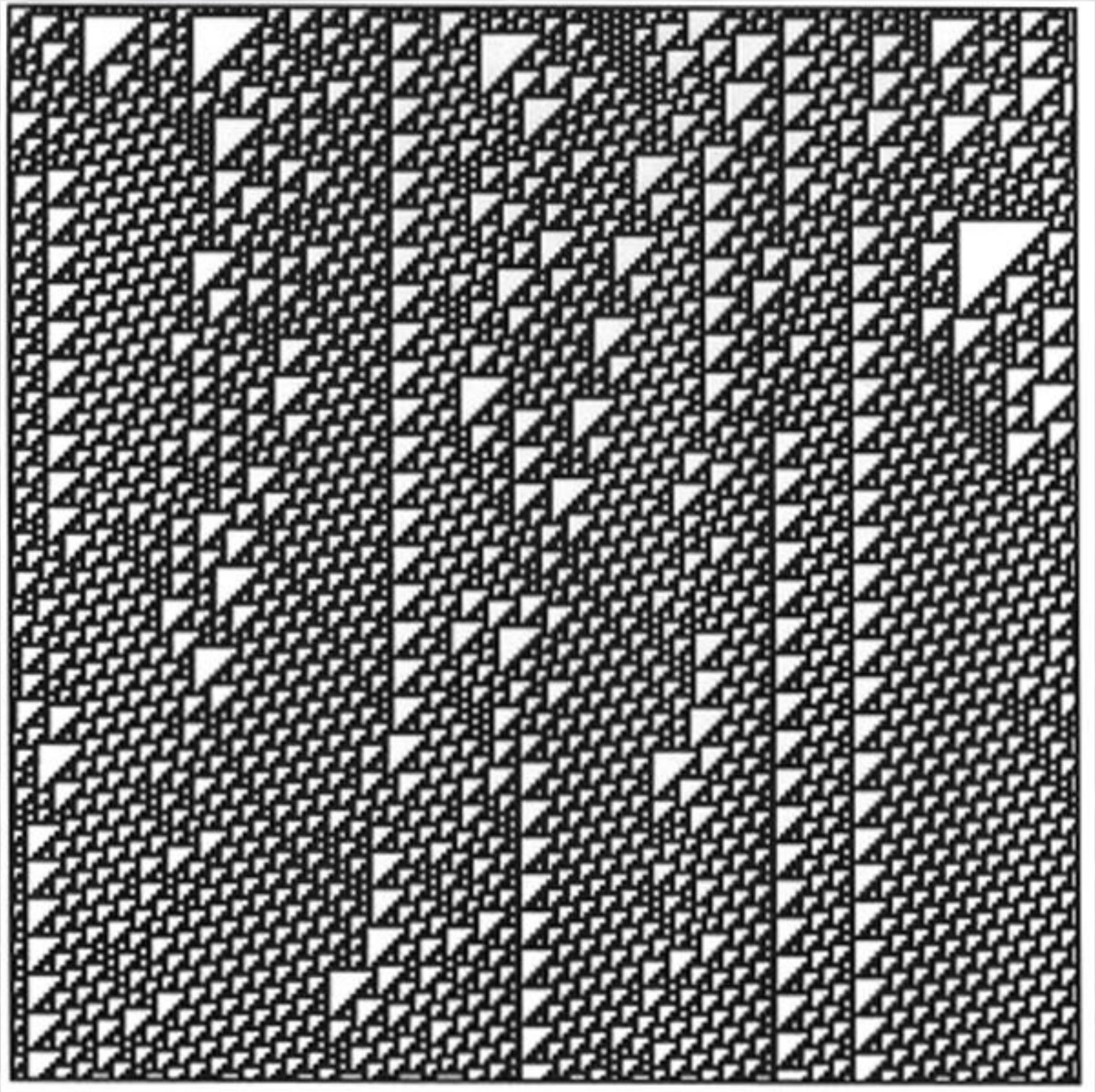


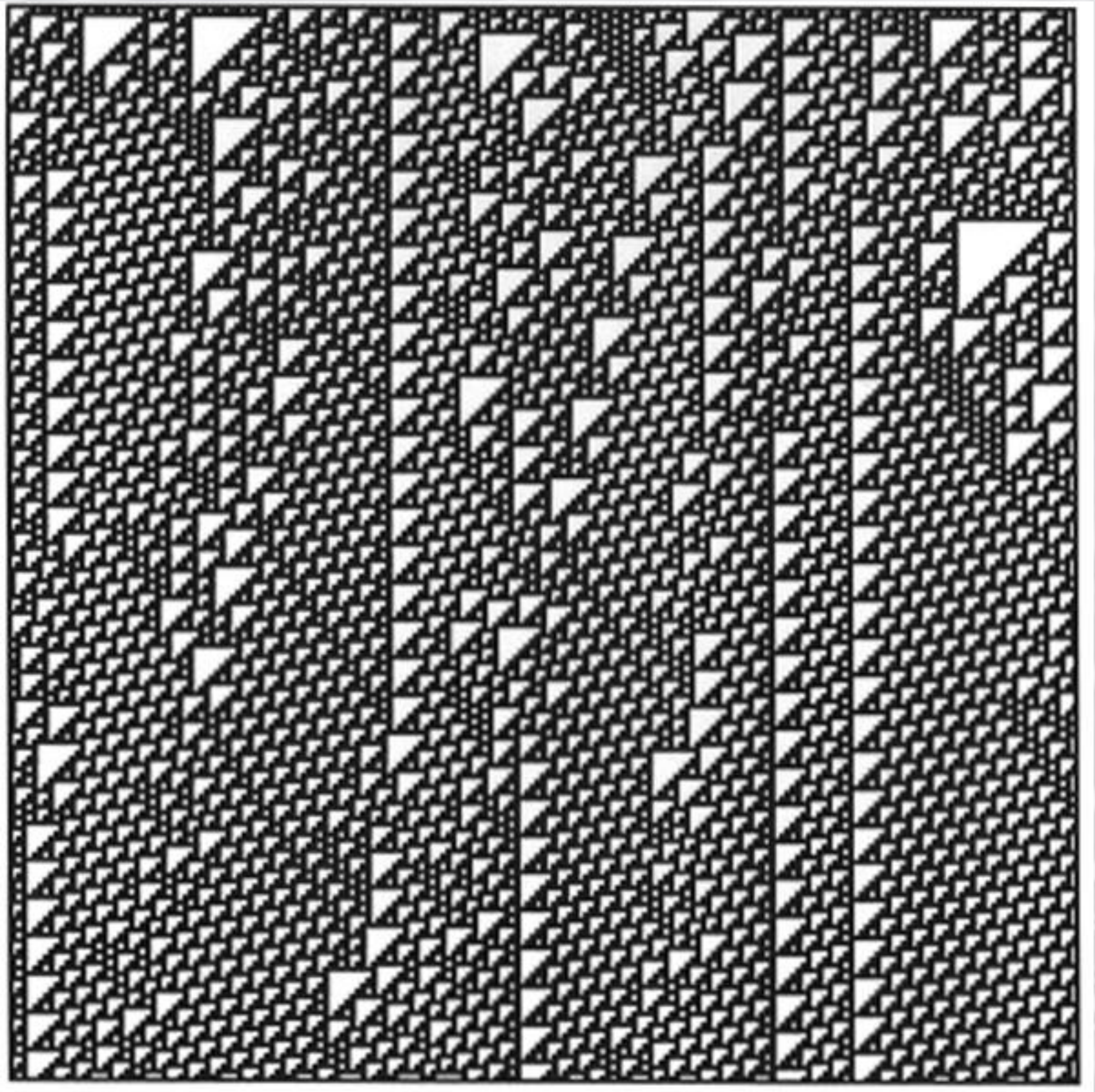
One-dimensional cellular automata

Rule:









Can the complex dynamics be harnessed by evolution
to perform collective information processing?

A task requiring collective computation in cellular automata

A task requiring collective computation in cellular automata

- Design a cellular automata to decide whether or not the initial pattern has a majority of “on”cells.

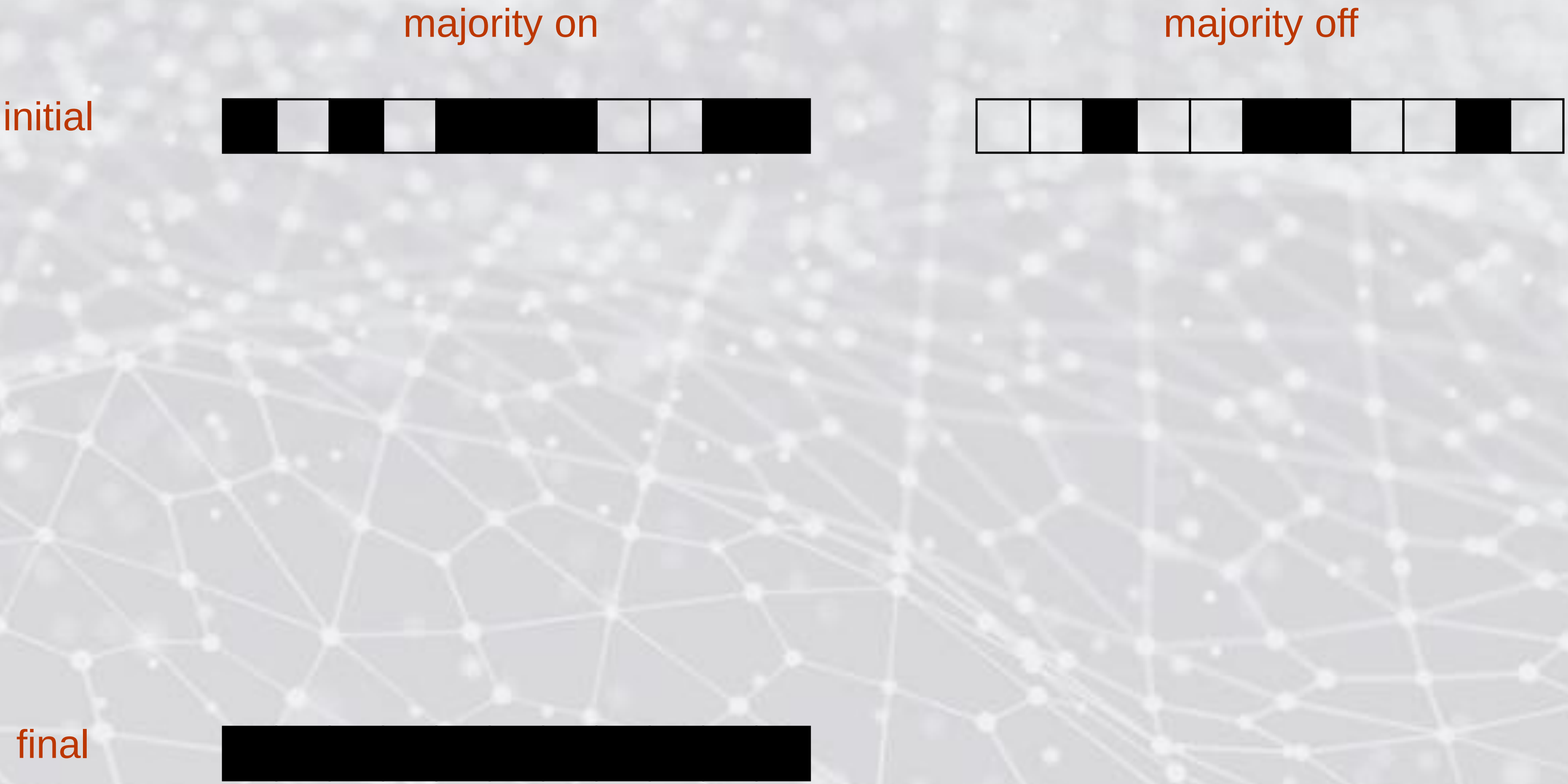
majority on

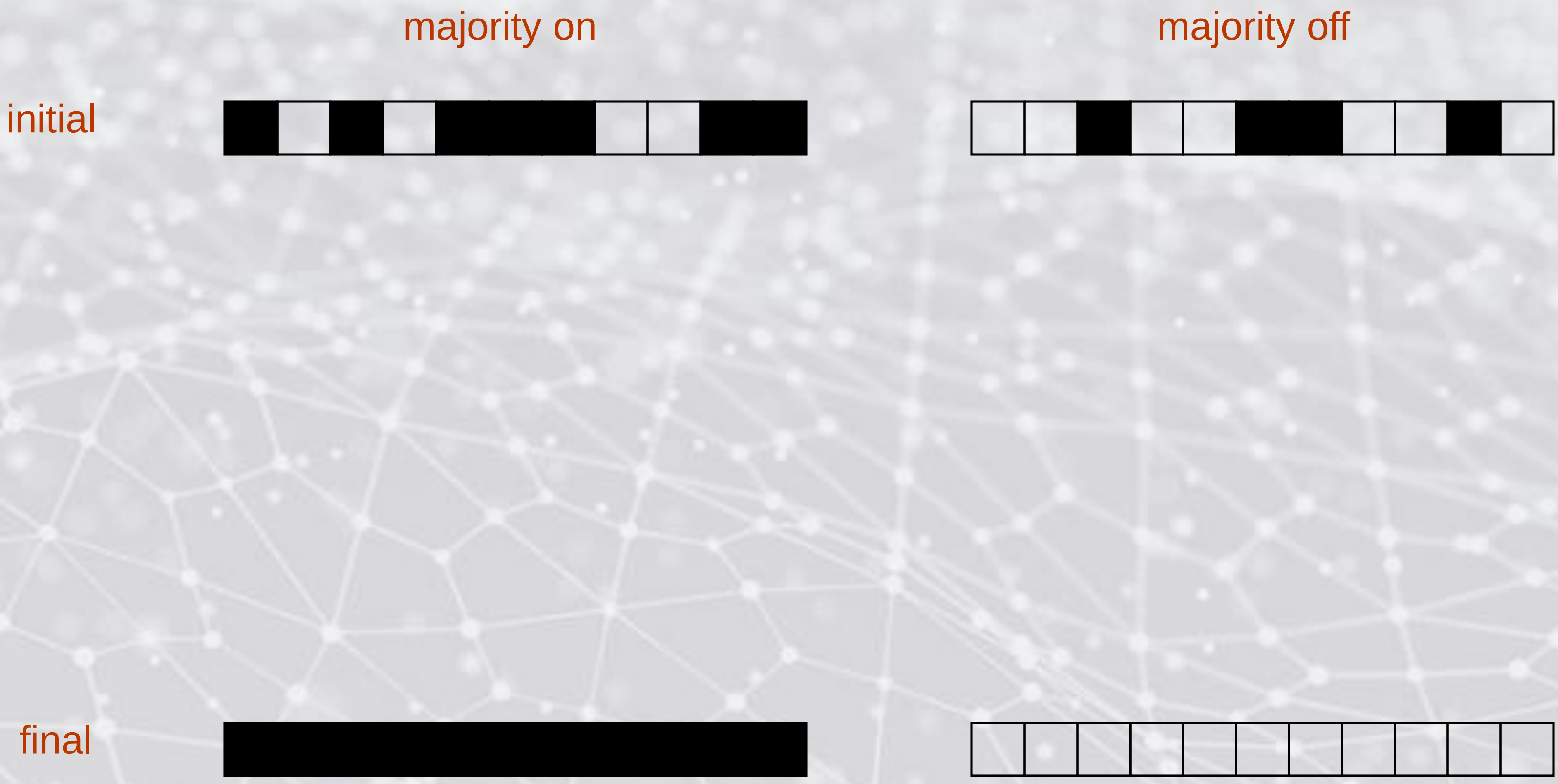
initial

majority on

initial

final


majority on
majority off
initial
final


majority on
majority off
initial
final

majority on

majority off

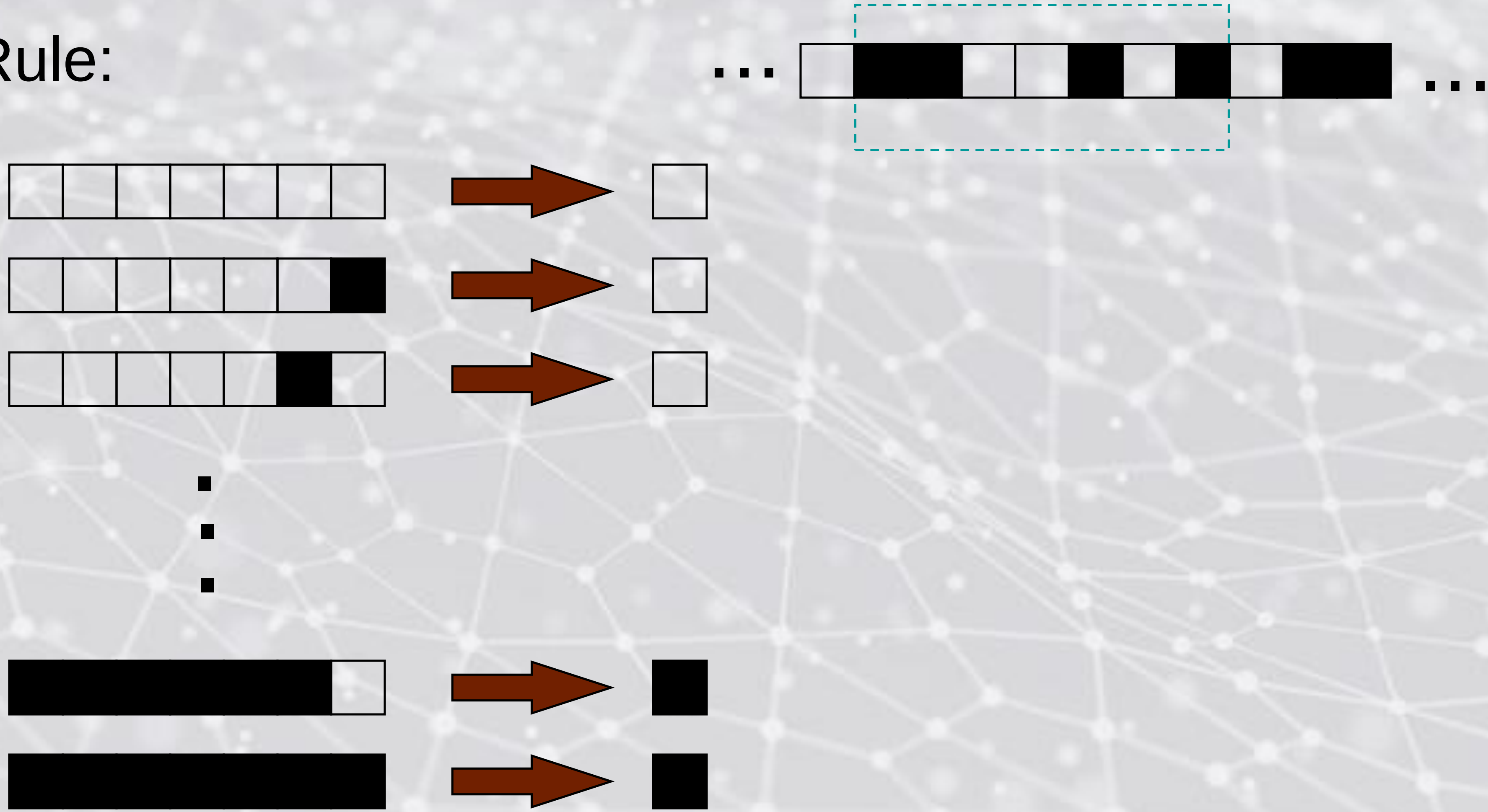
initial

How to design a cellular automaton
that will do this?

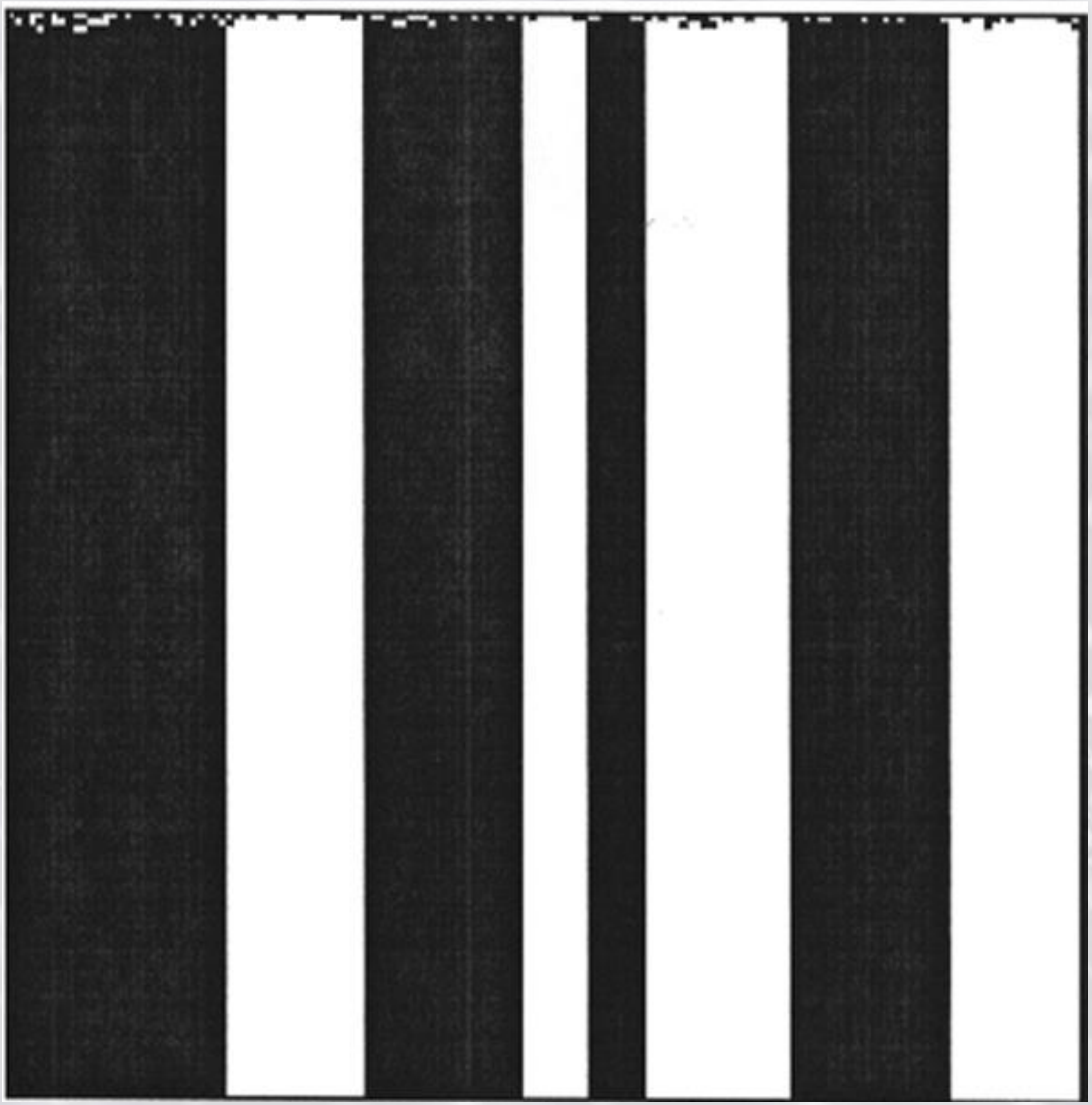
final

We used cellular automata with 6 neighbors for each cell:

Rule:



A candidate solution that does not work: Local majority voting



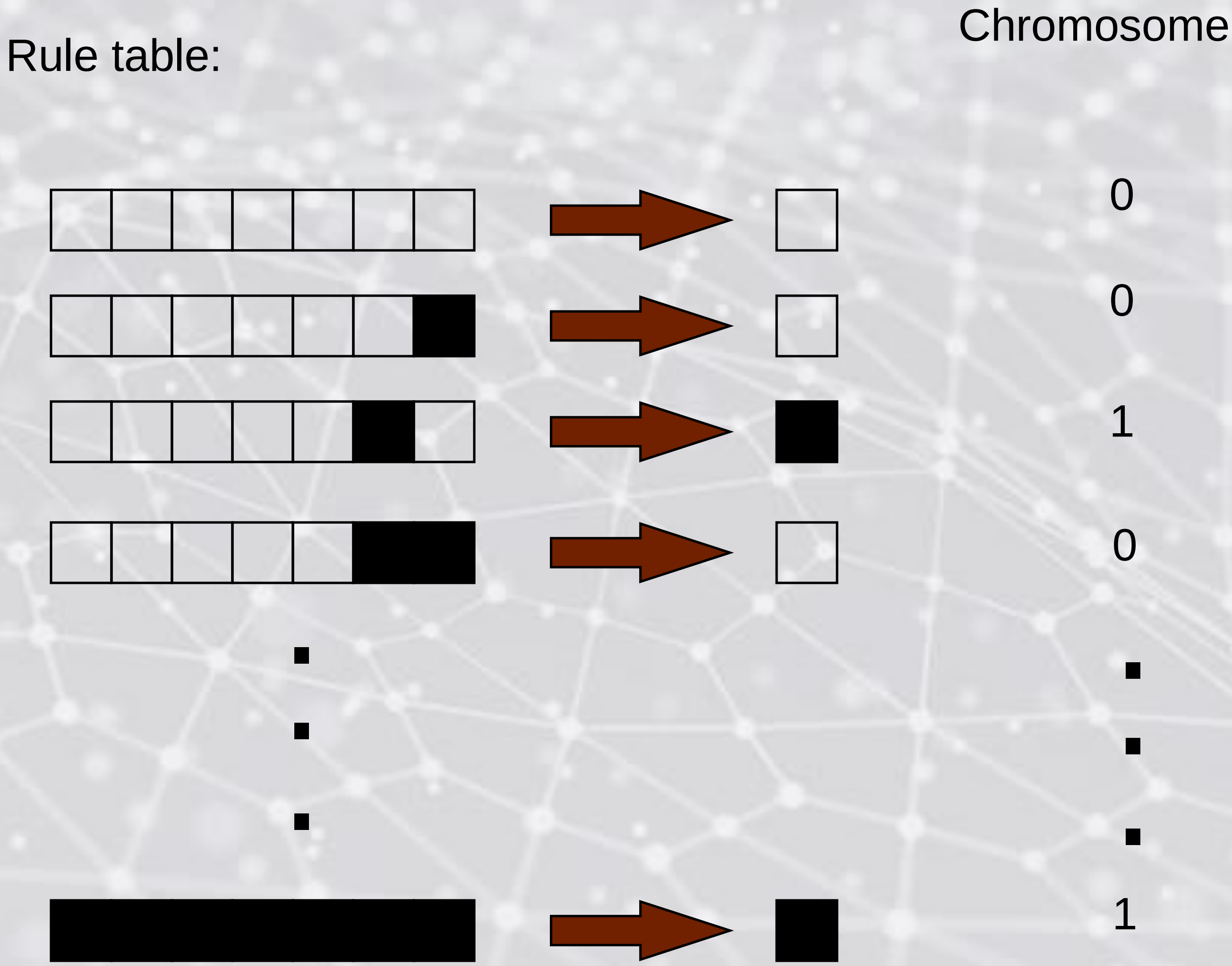
Evolving cellular automata with genetic algorithms

- Create a random population of candidate cellular automata rules

- The “fitness” of each cellular automaton is how well it performs the task. (Analogous to surviving in an environment.)

- The fittest cellular automata get to reproduce themselves, with mutations

The “chromosome” of a cellular automaton is an encoding of its rule table:



Create a random population of candidate cellular automata rules:

rule 1: 0010001100010010111100010100110111000...
rule 2: 0001100110101011111111000011101001010...
rule 3: 1111100010010101000000001110001001 0101...

.

.

.

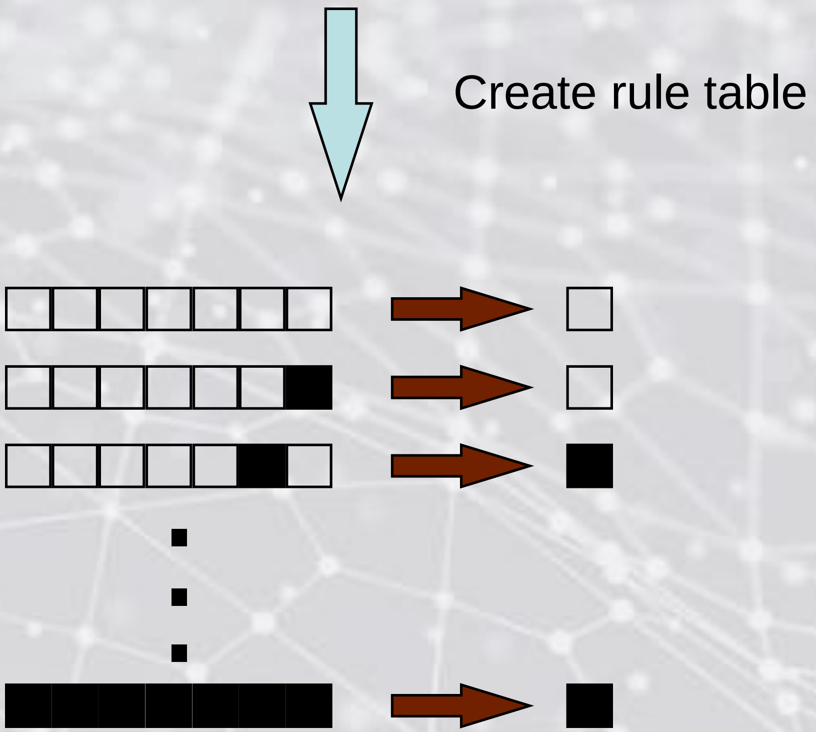
rule 100: 0010111010000000111110000001010010111111...

Calculating the Fitness of a Rule

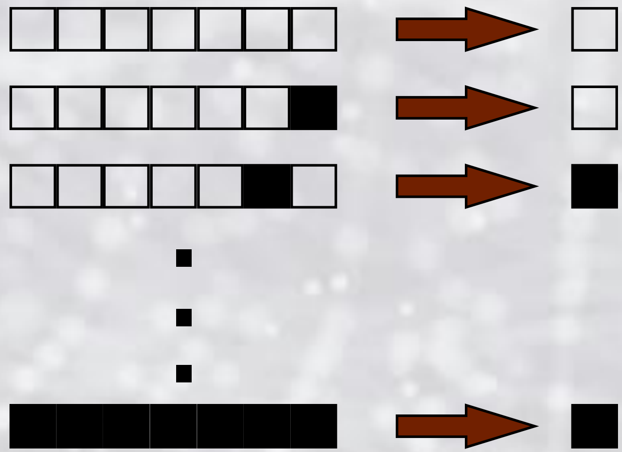
- For each rule, create the corresponding cellular automaton. Run that cellular automaton on many initial configurations.
- Fitness of rule = fraction of correct classifications

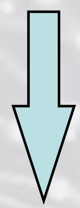
For each cellular automaton rule in the population:

rule 1: 0010001100010010111100010100110111000...1

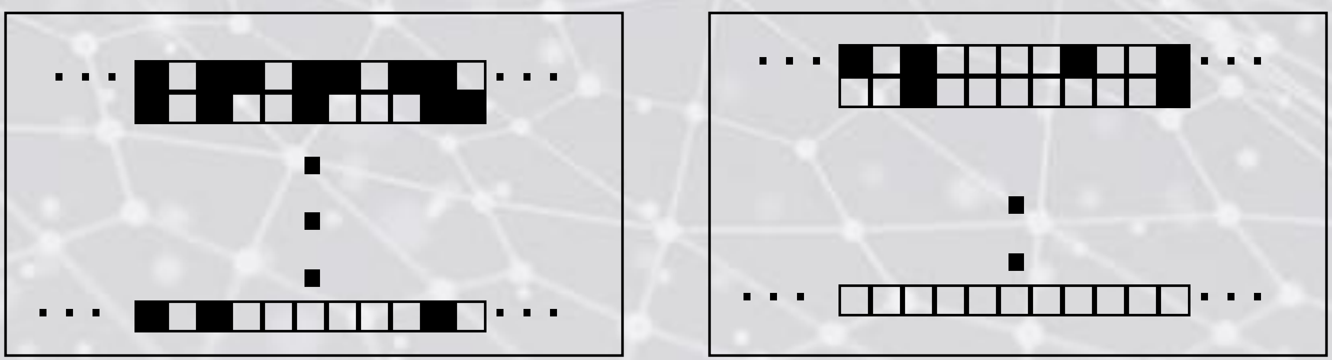


rule 1 rule table:





Run corresponding cellular automaton on many random initial lattice configurations



incorrect correct

etc.

Fitness of rule = fraction of correct classifications

GA Population:

rule 1: 0010001100010010111100010100110111000... Fitness = 0.5
rule 2: 0001100110101011111111000011101001010... Fitness = 0.2
rule 3: 1111100010010101000000011100010010101... Fitness = 0.4

.
.
.

rule 100:0010111010000001111100000101001011111... Fitness = 0.0

Select fittest rules to reproduce themselves

rule 1: 0010001100010010111100010100110111000... Fitness = 0.5
rule 3: 1111100010010101000000011100010010101... Fitness = 0.4

.
.
.

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 1000100101111000101001101110000...

rule 3: 1111100 0100101010000000111000100101101...

mutate

Children:

00100010100101010000000111000100101101...

1111100 1000100101111000101001101110000...

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 100010010111100010100110111000...

rule 3: 1111100 010010101000000011100010010101...

mutate

Children:

0010000010010101000000011100010010101...

1111100 100010010111100010100110111000...

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 100010010111100010100110111000...
rule 3: 1111100 010010101000000011100010010101...

Children:

0010000010010101000000011100010010101...
1111100 100010010111100010100110111000...

mutate

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 1000100101111000101001101111000...
rule 3: 1111100 0100101010000000111000100101 01...

Children:

0010000010010101000000011100010010101...
1111100 100010010111100010100010111000...

mutate

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 1000100101111000101001101110000...

rule 3: 1111100 0100101010000000111000100101010...

Children:

001000001001010100000001110001001010 1...

1111100 100010010111100010100010111000...

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 1000100101111000101001101110000...
rule 3: 1111100 0100101010000000111000100100101...

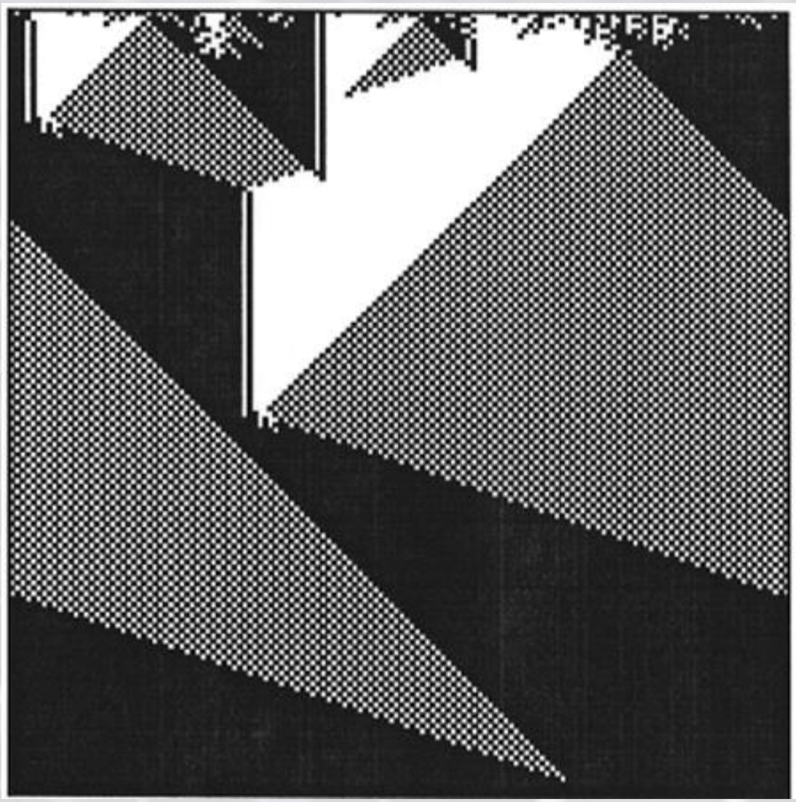
Children:

0010000 01001010100000001110001001010 1...
1111100 1000100101111000101000101110000...

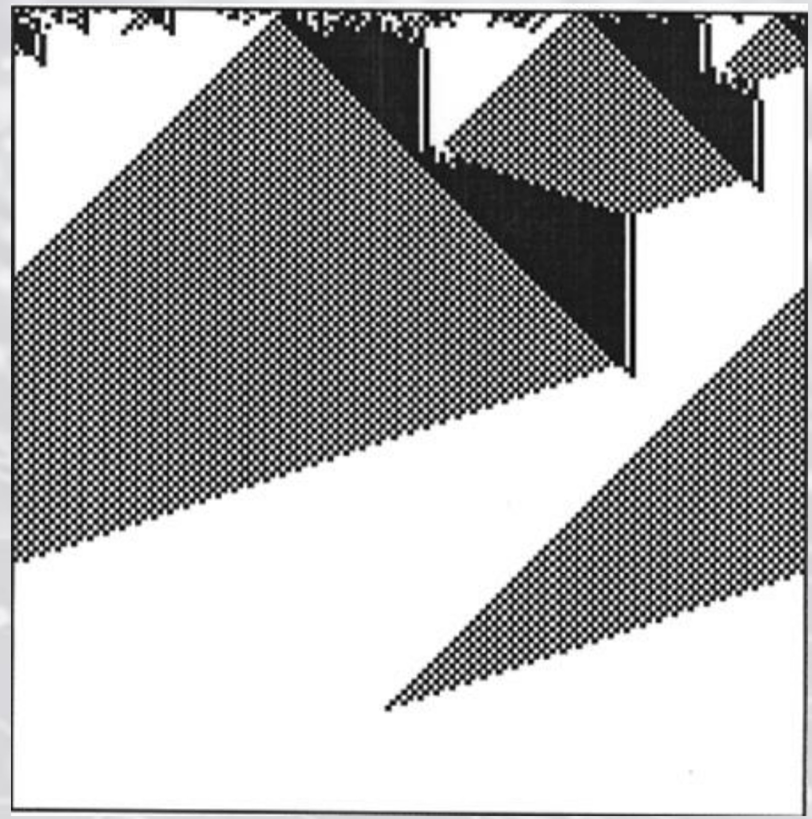
Continue this process until new generation is complete.
Then start over with the new generation.

Keep iterating for many generations.

majority on



majority off

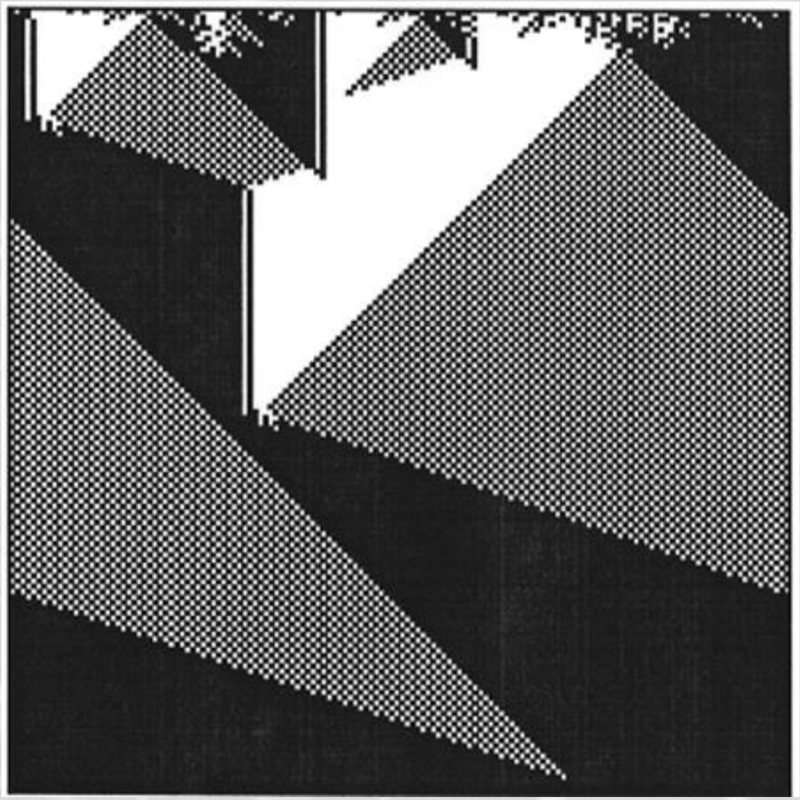


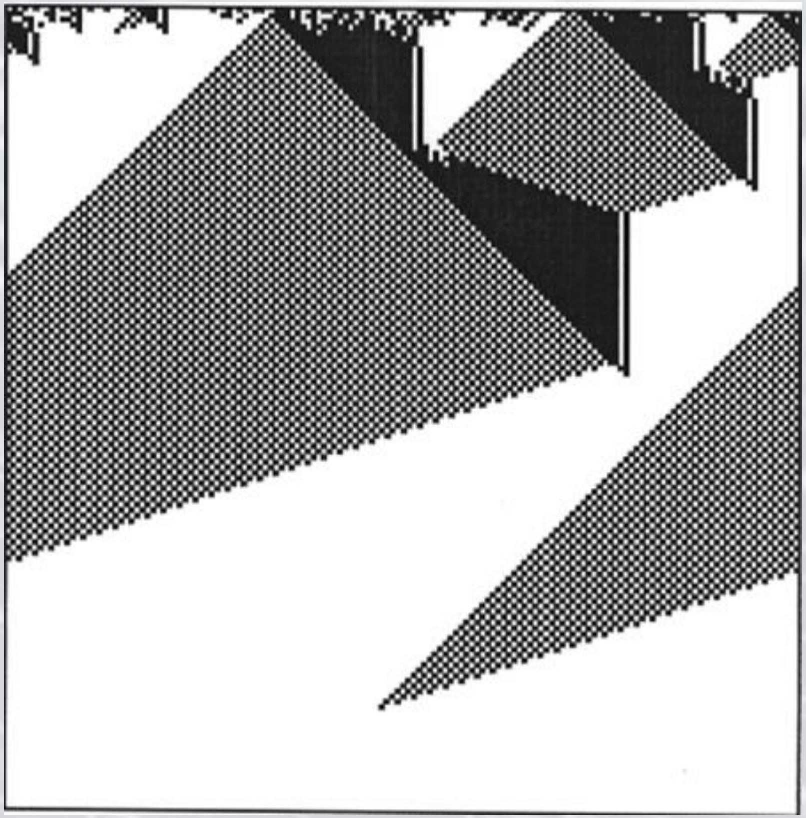
A cellular automaton evolved by the genetic algorithm

How does it perform the computation?

majority on

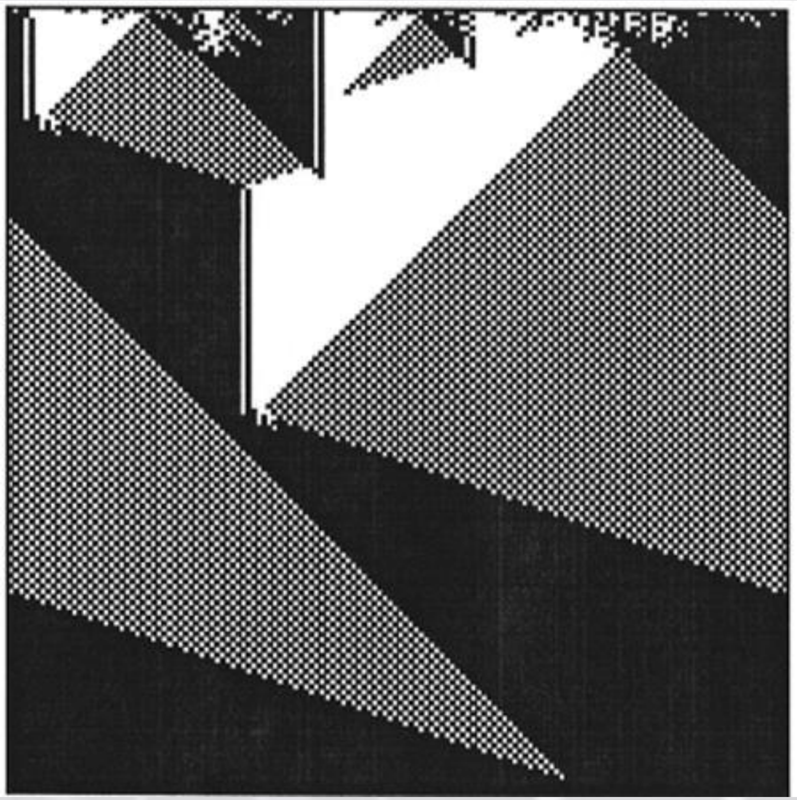
majority off



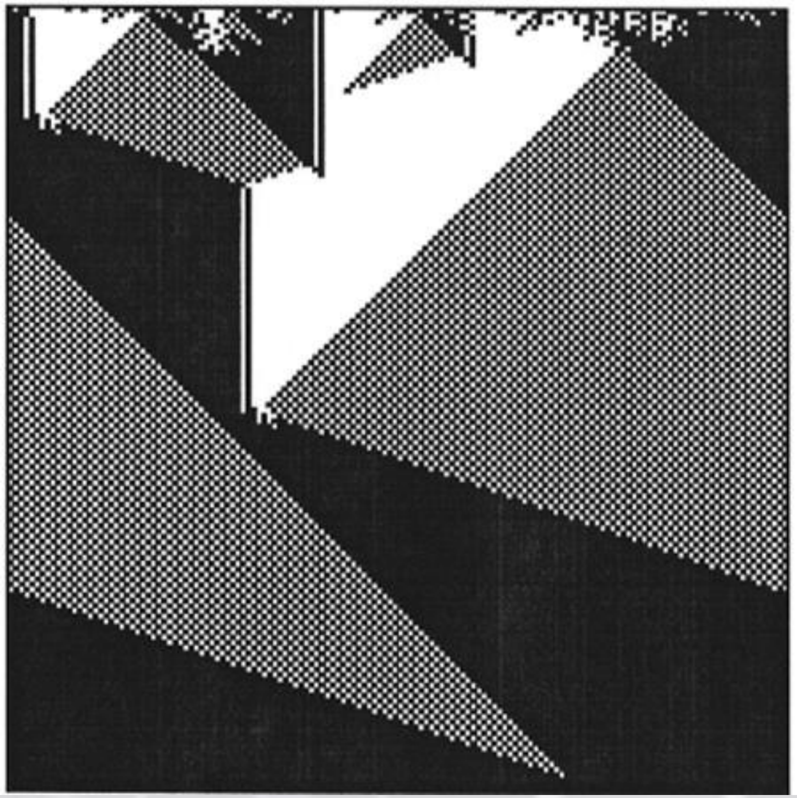


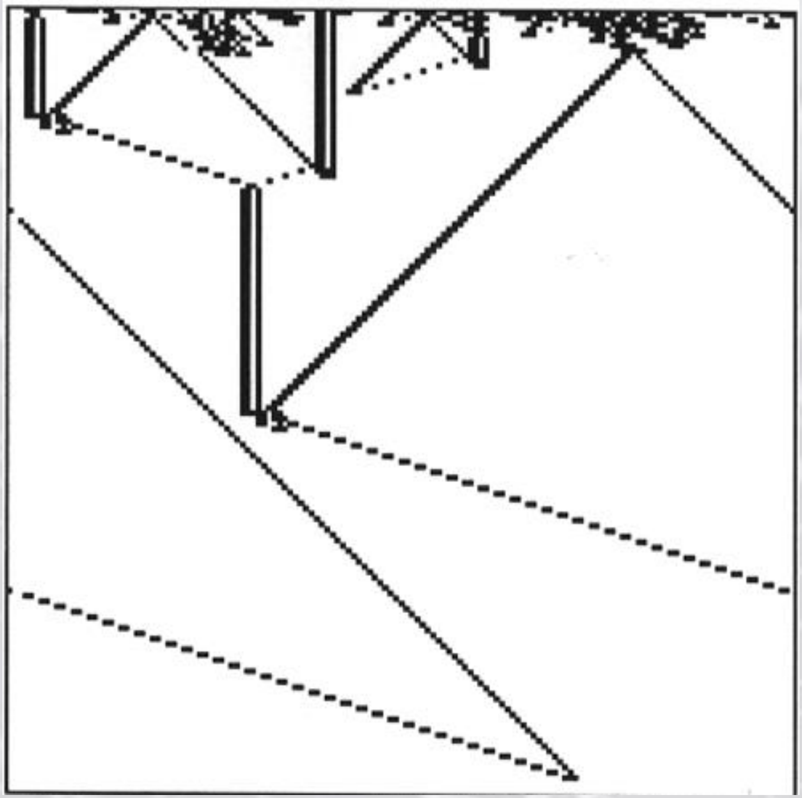
A cellular automaton evolved by the genetic algorithm

How do we describe information processing in complex systems?

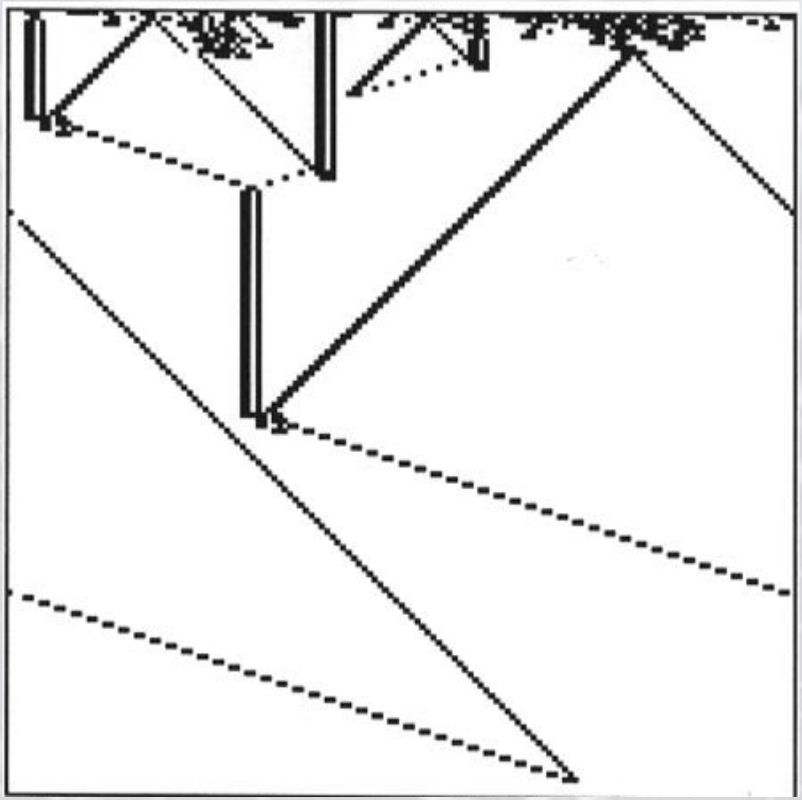


“simple patterns”:
black, white, checkerboard

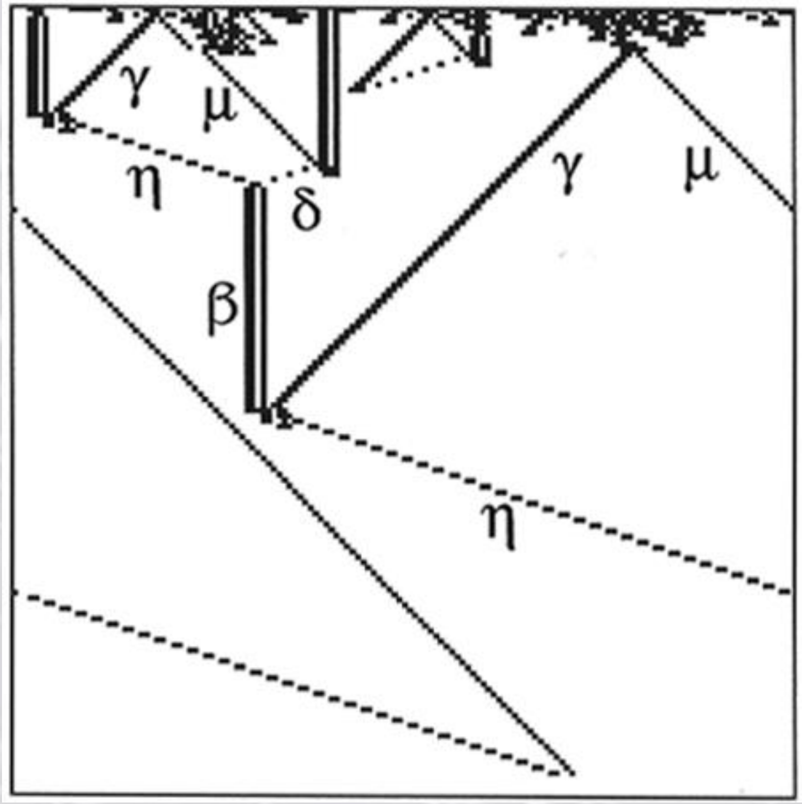




Simple patterns
filtered out



“particles”

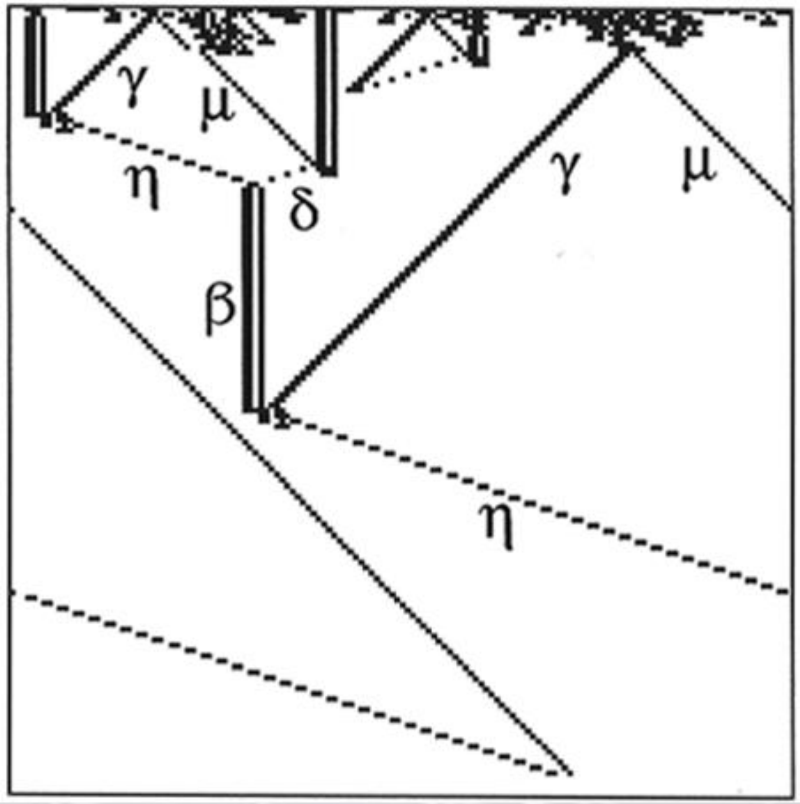


“particles”

Regular Domains		
$\Lambda^0 = 0^*$	$\Lambda^1 = 1^*$	$\Lambda^2 = (01)^*$

Particles (Velocities)	
$\alpha \sim \Lambda^0\Lambda^1$ (0)	$\beta \sim \Lambda^1 01 \Lambda^0$ (0)
$\gamma \sim \Lambda^0\Lambda^2$ (-1)	$\delta \sim \Lambda^2\Lambda^0$ (-3)
$\eta \sim \Lambda^1\Lambda^2$ (3)	$\mu \sim \Lambda^2\Lambda^1$ (1)

Interactions	
decay	$\alpha \rightarrow \gamma + \mu$
react	$\beta + \gamma \rightarrow \eta$, $\mu + \beta \rightarrow \delta$, $\eta + \delta \rightarrow \beta$
annihilate	$\eta + \mu \rightarrow \emptyset_1$, $\gamma + \delta \rightarrow \emptyset_0$



laws of
"particle physics"

"particles"

- Level of particles can explain:

- Level of particles can explain:
  - Why one CA is fitter than another

- Level of particles can explain:
  - Why one CA is fitter than another
  - What mistakes are made

- Level of particles can explain:
  - Why one CA is fitter than another
  - What mistakes are made
  - How the GA produced the observed series of innovations

- Level of particles can explain:
  - Why one CA is fitter than another
  - What mistakes are made
  - How the GA produced the observed series of innovations

- Particles give an “information processing” description of the collective behavior

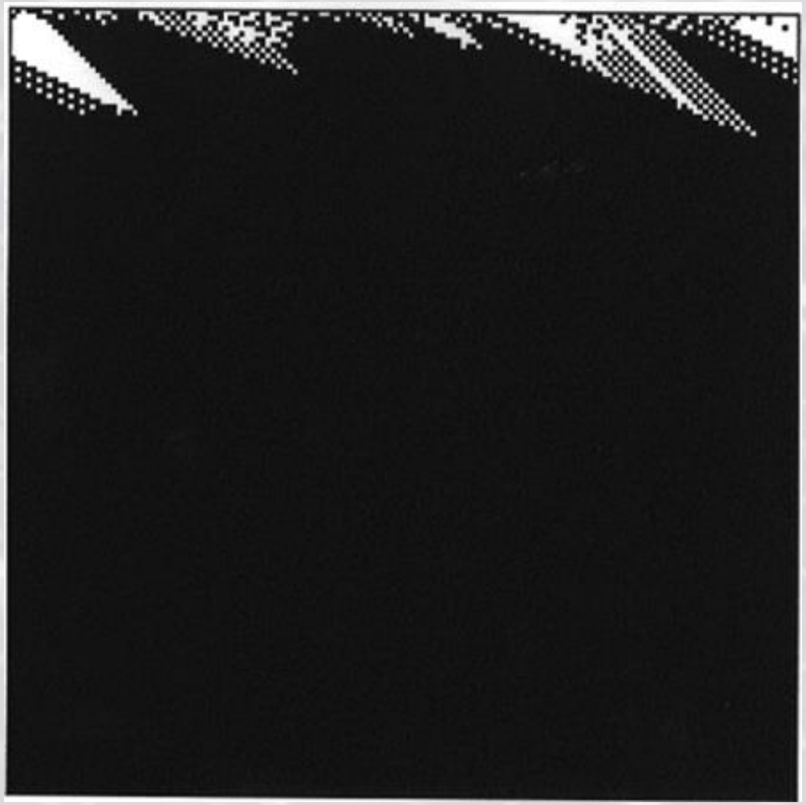
- Level of particles can explain:
  - Why one CA is fitter than another
  - What mistakes are made
  - How the GA produced the observed series of innovations

- Particles give an “information processing” description of the collective behavior

  ----> “Algorithmic” level

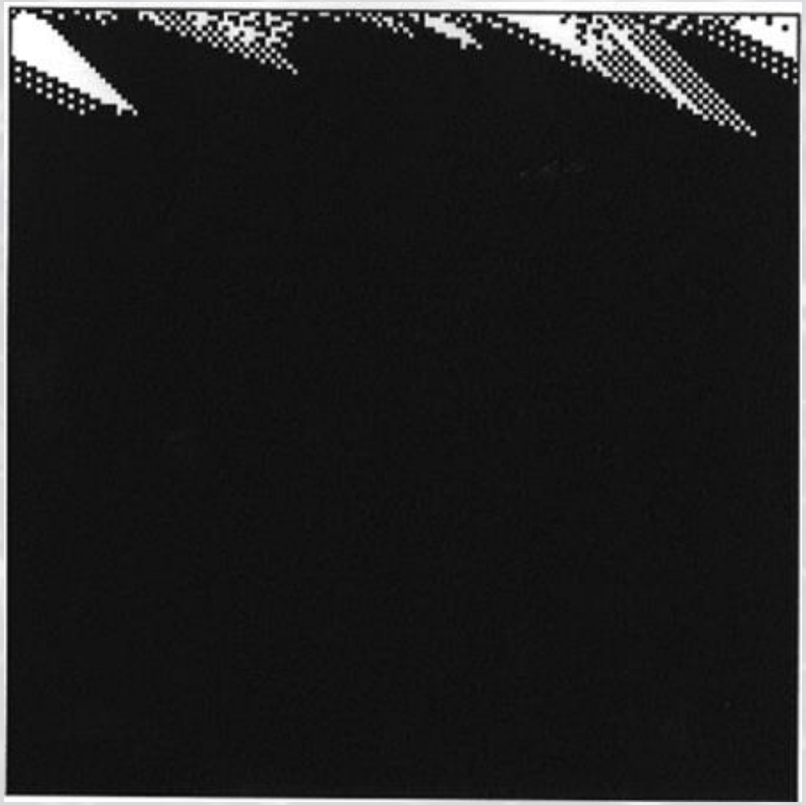
How the genetic algorithm evolved cellular automata

How the genetic algorithm evolved cellular automata



generation 8

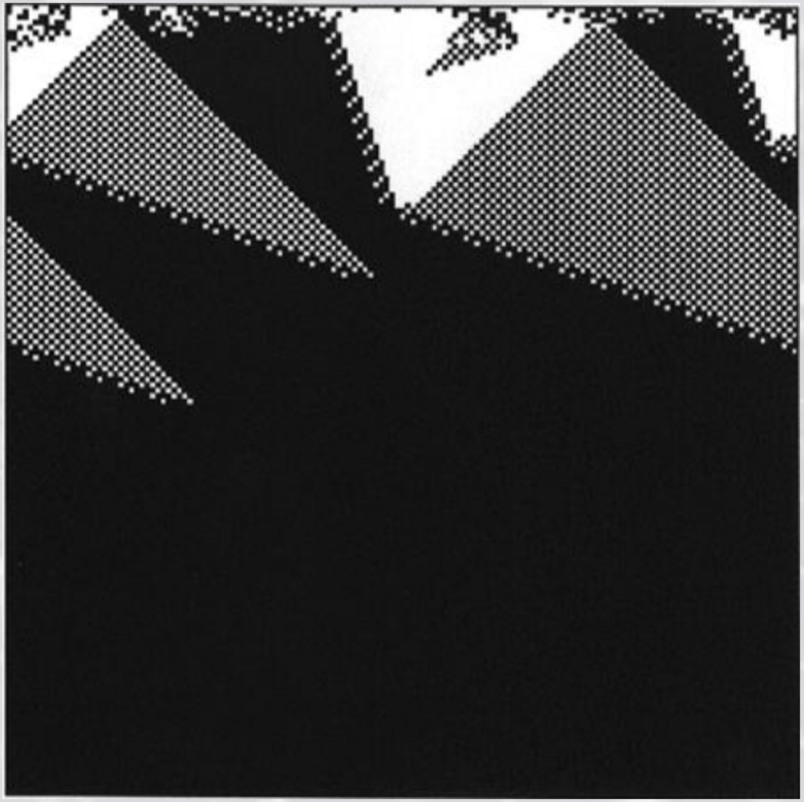
How the genetic algorithm evolved cellular automata



generation 8

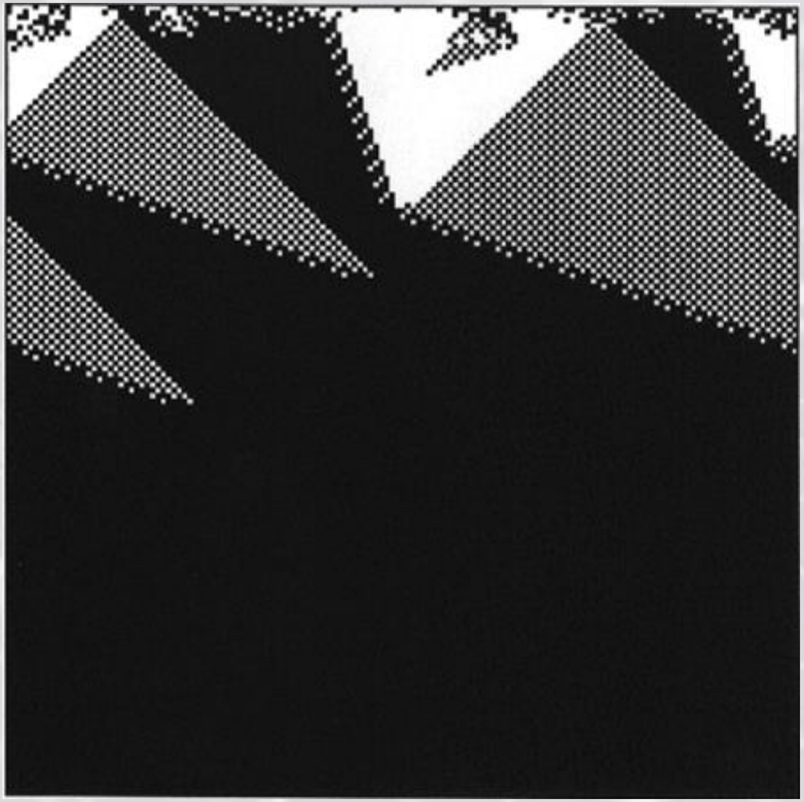
generation 13

How the genetic algorithm evolved cellular automata

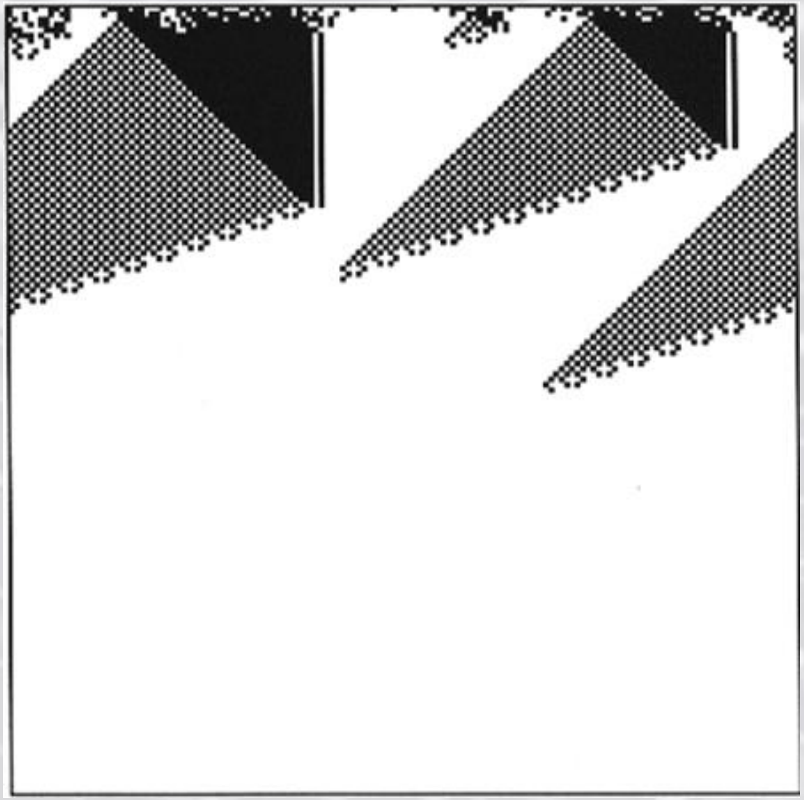


generation 17

How the genetic algorithm evolved cellular automata

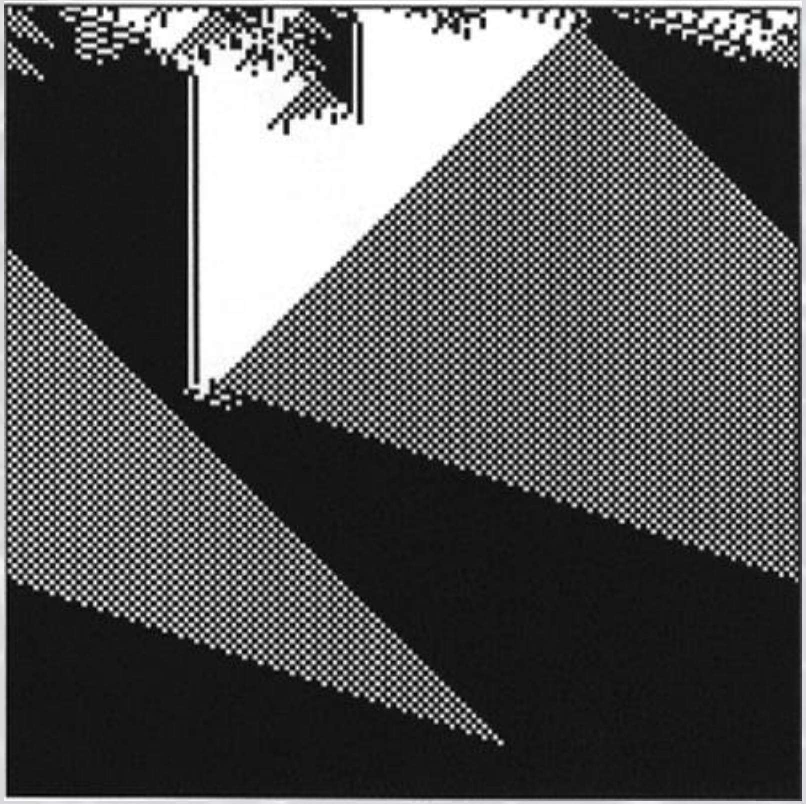


generation 17



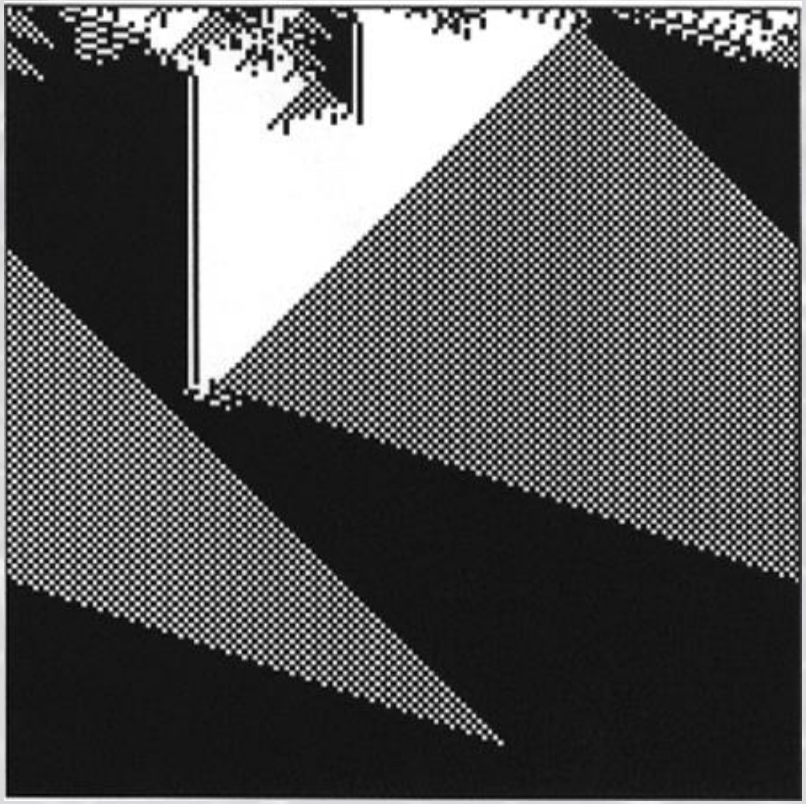
generation 18

How the genetic algorithm evolved cellular automata

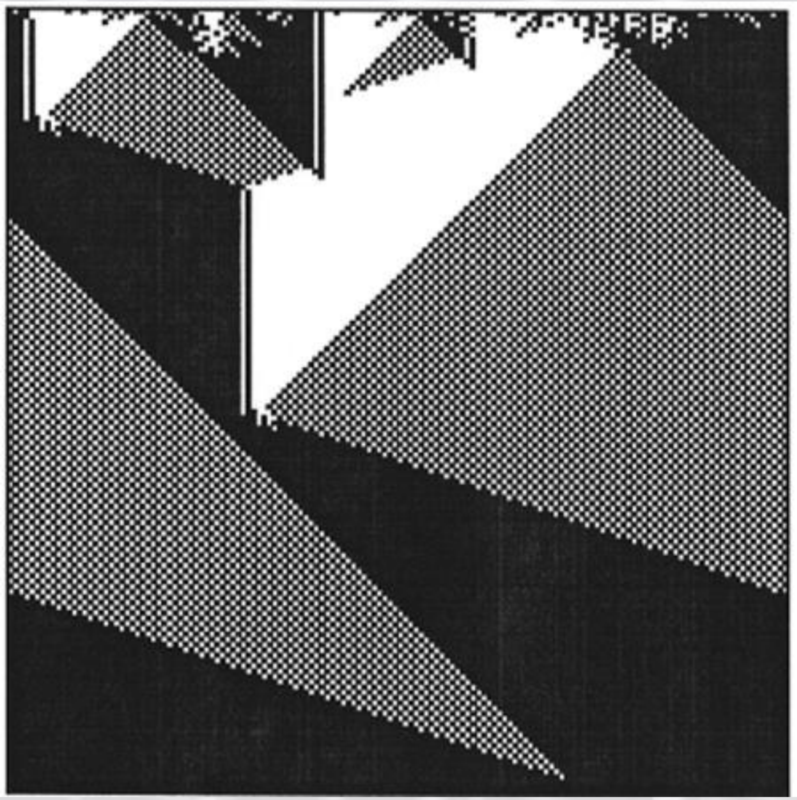


generation 33

How the genetic algorithm evolved cellular automata



generation 33



generation 64

	Cellular Automata
Traditional GA (with crossover)	20%
Traditional GA (mutation only)	0%

Percentage of successful runs

	Cellular Automata
Traditional GA (with crossover)	20%
Traditional GA (mutation only)	0%

Percentage of successful runs

Problem: GA often gets stuck in local optima, with “too easy” training examples

Problem for learning algorithms:

How to select training examples appropriate to different stages of learning?

One solution:

Co-evolve training examples, using inspiration from host-parasite coevolution in nature.

Host-parasite coevolution in nature

- Hosts evolve defenses against parasites
- Parasites find ways to overcome defenses
- Hosts evolve new defenses
- Continual “biological arms race”

Heliconius-egg mimicry in *Passiflora*

http://www.ucl.ac.uk/~ucbhdjm/courses/b242/Coevol/Coevol.html

- Darwin recognized the importance of coevolution in driving evolution

- Darwin recognized the importance of coevolution in driving evolution

- Coevolution was later hypothesized to be major factor in evolution of sexual reproduction

Coevolutionary Learning

Candidate solutions and training examples coevolve.

Coevolutionary Learning

Candidate solutions and training examples coevolve.

- **Fitness of candidate solution (host):** how well it performs on training examples.

Coevolutionary Learning

Candidate solutions and training examples coevolve.

- **Fitness of candidate solution (host):** how well it performs on training examples.

- **Fitness of training example (parasite):** how well it defeats candidate solutions.

Sample Applications of Coevolutionary Learning

- Competitive:
  - Coevolving minimal sorting networks (Hillis)
    - Hosts: Candidate sorting networks
    - Parasites: Lists of items to sort

Sample Applications of Coevolutionary Learning

– Game playing strategies (e.g., Rosin & Belew; Fogel; Juillé & Pollack)

  - Hosts: Candidate strategies for Nim, 3D Tic Tac Toe, backgammon, etc.

  - Parasites: Another population of candidate strategies

– HIV drug design (e.g., Rosin)

  - Hosts: Candidate protease inhibitors to match HIV protease enzymes

  - Parasites: Evolving protease enzymes

- Robot behavior (e.g., Sims; Nolfi & Floreano)
  - Hosts: Robot control programs
  - Parasites: Competing robot control programs

- Cooperative:
  - Cooperative coevolution of neural network weights and topologies (e.g., Potter & De Jong; Stanley, Moriarty, Miikkulainen)

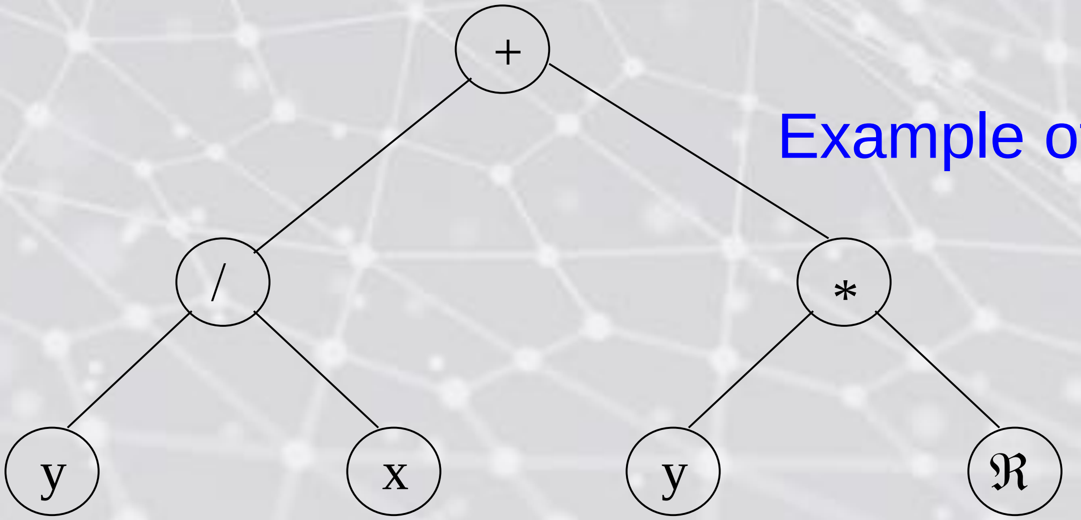
Problem domains used in experiments

1. **Function induction:** 2D function-induction task (Pagie & Hogeweg, 1997)

   – Evolve function tree to approximate

$$\mathrm{f}(x, y) = \frac{1}{1 + x^{-4}} + \frac{1}{1 + y^{-4}}$$

- Hosts are candidate trees
  - Function set: {+, -, $*$, %}
  - Terminal set: {x, y, $\Re$}



Example of candidate tree

– Parasites are *(x,y)* pairs

- Fitness (*h*) = Average inverse error on sample of *p*

- Fitness (*p*): Error of *h* on problem *p*

- **Success** = Correct host (on complete set of problems) in population for 50 generations

2. Evolving cellular automata

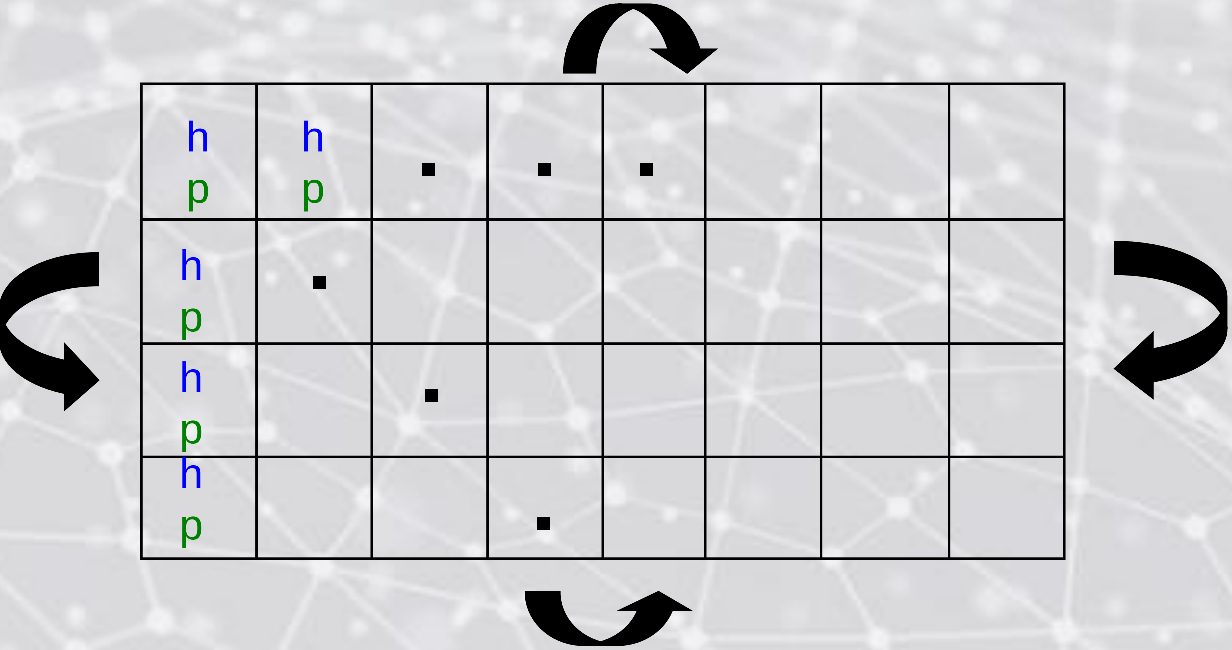
- Problem is to design 1D CA that classifies initial configurations (ICs) as “majority 1s” or “majority 0s”.

Spatial Coevolution

- 2D toroidal lattice with one host (*h*) and one parasite (*p*) per site

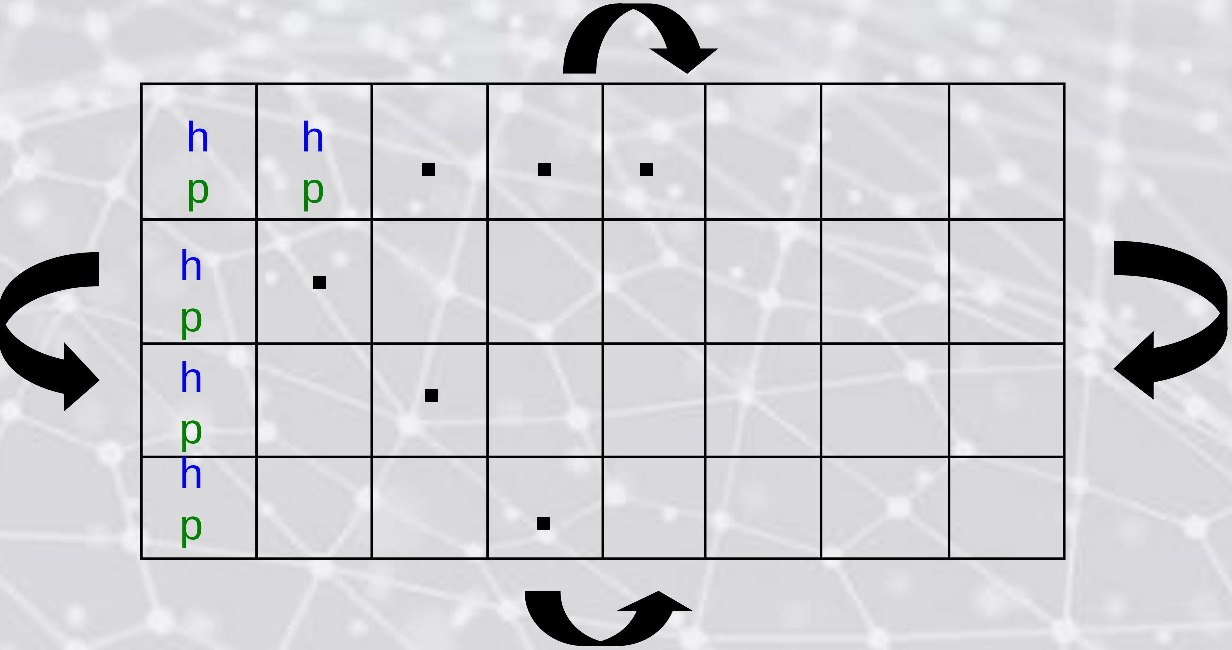
Spatial Coevolution

- 2D toroidal lattice with one host (*h*) and one parasite (*p*) per site



Spatial Coevolution

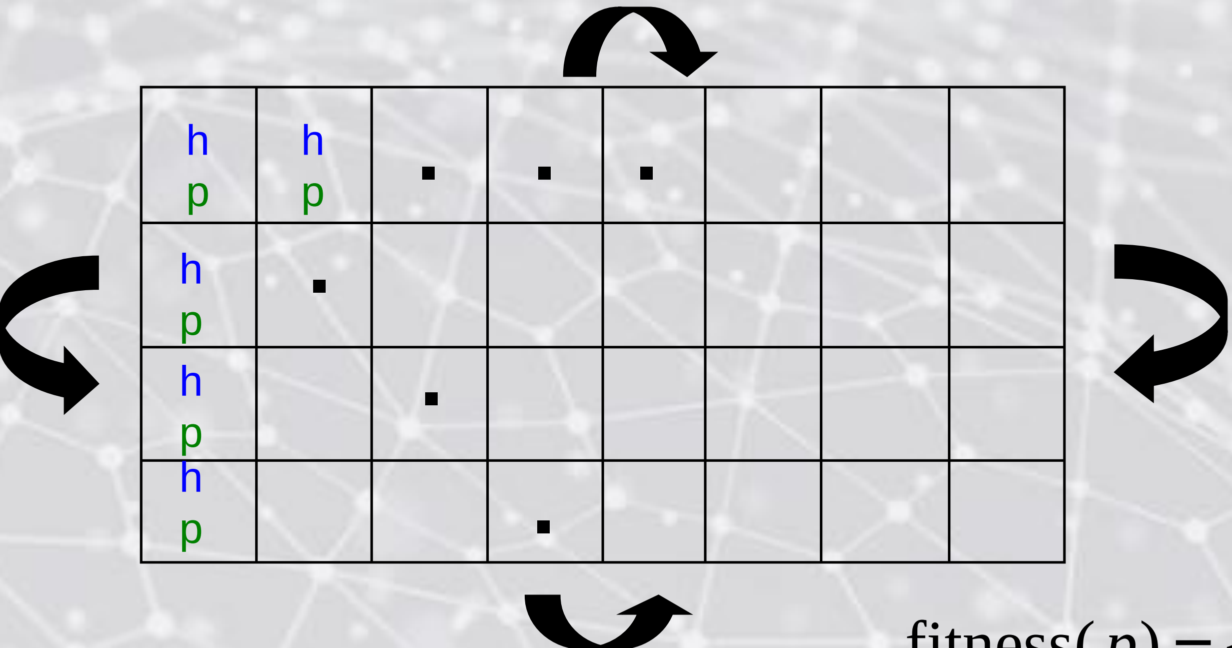
- 2D toroidal lattice with one host (*h*) and one parasite (*p*) per site



$$\text{fitness}(h) = \text{fraction of 9 neighboring } p \text{ answered correctly}$$

Spatial Coevolution

- 2D toroidal lattice with one host (*h*) and one parasite (*p*) per site

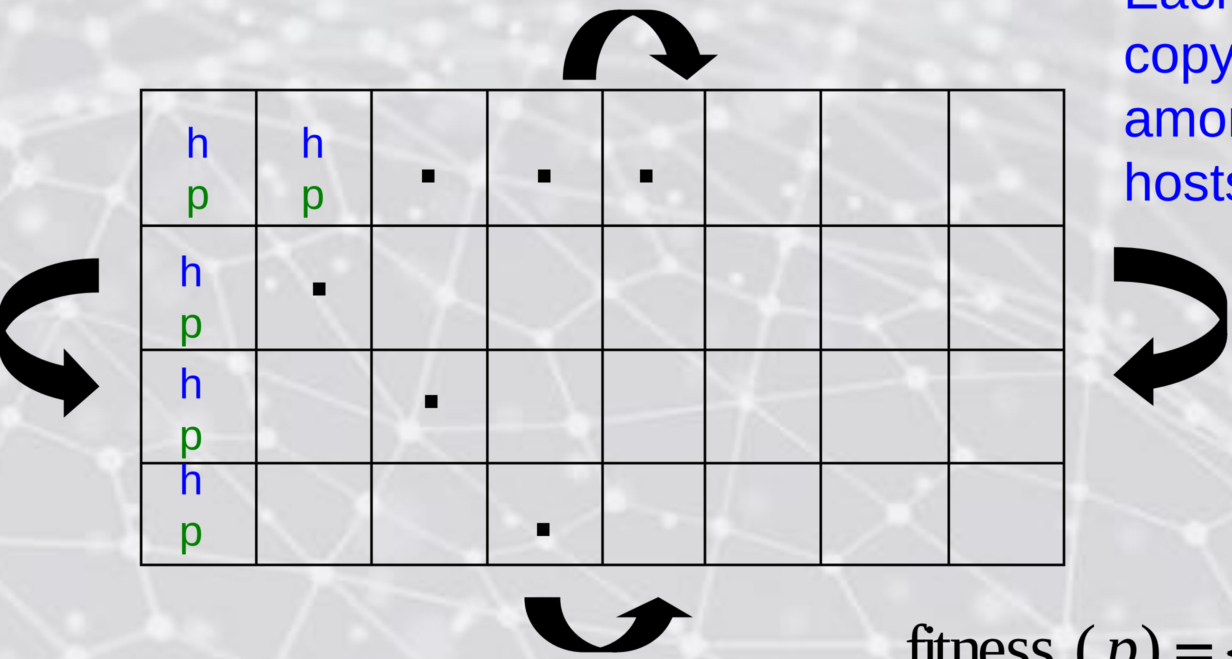


$$\text{fitness}(p) = \begin{cases} 0 & \text{if } h(\text{p}) \text{ is correct} \\ > 0 & \text{if } h(p) \text{ is not correct} \end{cases}$$

$$\text{fitness}(h) = \text{fraction of 9 neighboring } p \text{ answered correctly}$$

Spatial Coevolution

- 2D toroidal lattice with one host (*h*) and one parasite (*p*) per site



Each *h* is replaced by mutated copy of winner of tournament among itself and 8 neighboring hosts.

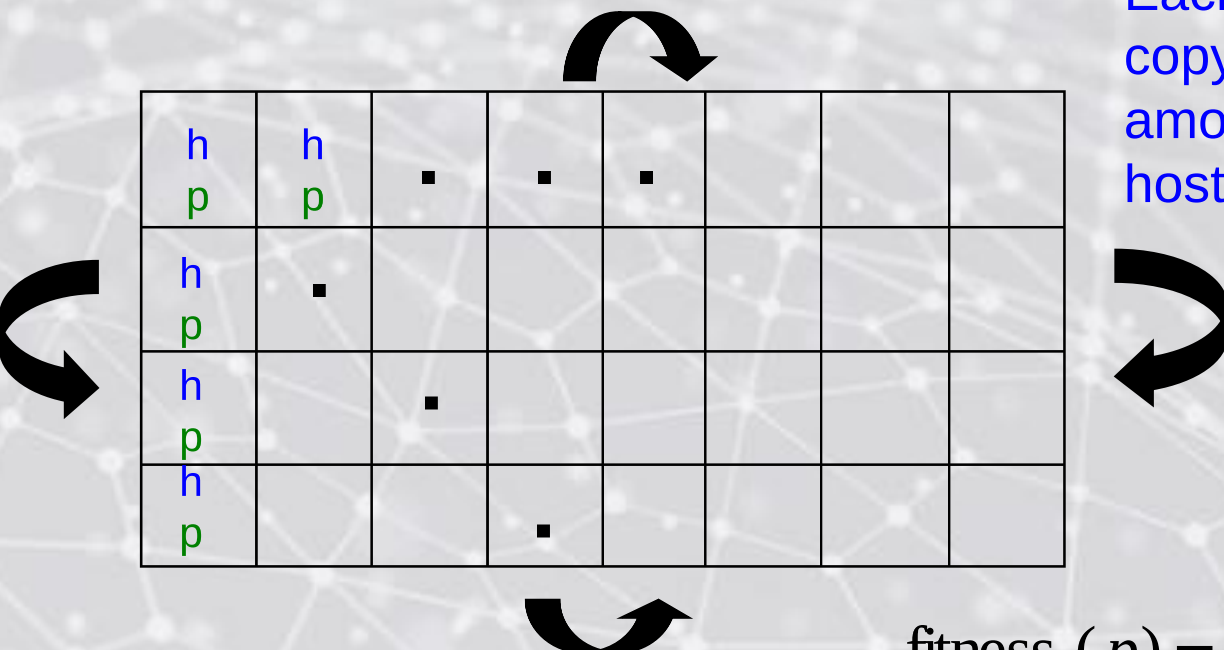
$$\text{fitness}\,(p) = \begin{cases} 0 & \text{if } \; h(\mathrm{p}) \text{ is correct} \\ > 0 & \text{if } \; h(p) \text{ is not correct} \end{cases}$$

$$\text{fitness}\,(h) = \text{ fraction of 9 neighboring } p \text{ answered correctly}$$

Spatial Coevolution

- 2D toroidal lattice with one host (*h*) and one parasite (*p*) per site

Each *h* is replaced by mutated copy of winner of tournament among itself and 8 neighboring hosts.



Each *p* is replaced by mutated copy of winner tournament among itself and 8 neighboring parasites.

$$\text{fitness }(p) = \begin{cases} 0 & \text{if } h(\text{p}) \text{ is correct} \\ > 0 & \text{if } h(p) \text{ is not correct} \end{cases}$$

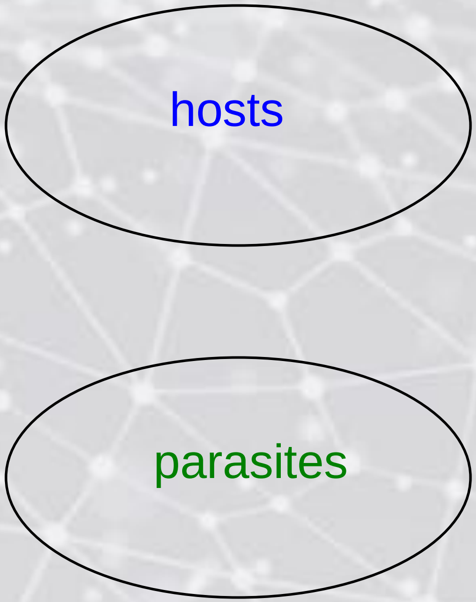
$$\text{fitness }(h) = \text{ fraction of 9 neighborin g } p \text{ answered correctly}$$

Non-Spatial Coevolution

- No spatial distribution of host and parasite populations

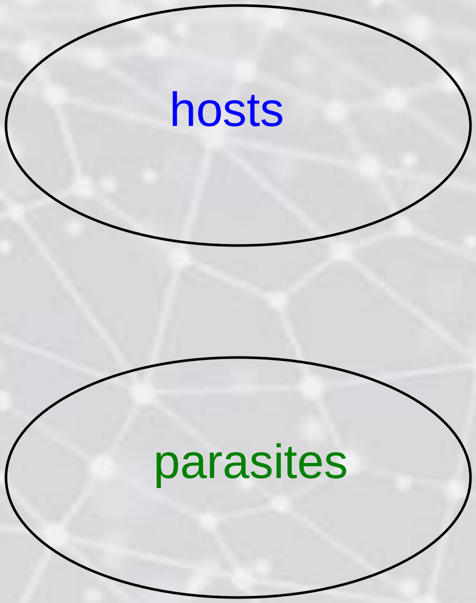
Non-Spatial Coevolution

- No spatial distribution of host and parasite populations



Non-Spatial Coevolution

- No spatial distribution of host and parasite populations



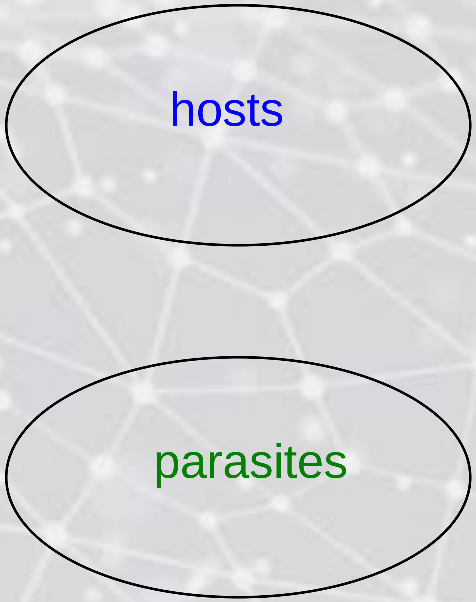
$\text{fitness}(h) = \text{fraction of } 9 \; \textit{parasites } p$

(randomly chosen from parasite population)

answered correctly

Non-Spatial Coevolution

- No spatial distribution of host and parasite populations



$\text{fitness}\,(h) =$ fraction of 9 *parasites p* (randomly chosen from parasite population) answered correctly

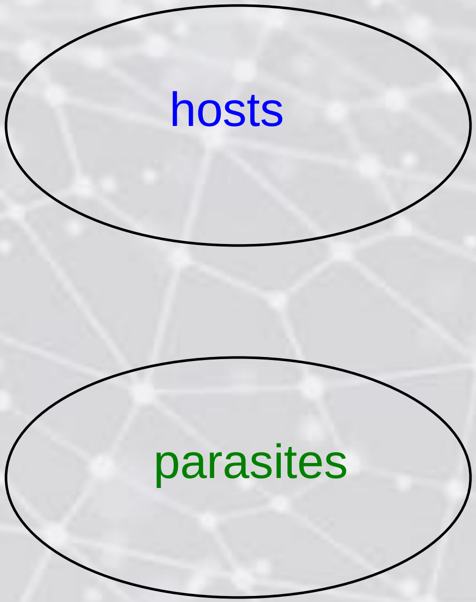
$$\text{fitness}(p) = \begin{cases} 0 & \text{if } h(\text{p}) \text{ is correct} \\ > 0 & \text{if } h(p) \text{ is not correct} \end{cases}$$

for host h randomly chosen from host population

Non-Spatial Coevolution

- No spatial distribution of host and parasite populations

Each *h* is replaced by mutated copy of winner of tournament among itself and 8 randomly chosen hosts.



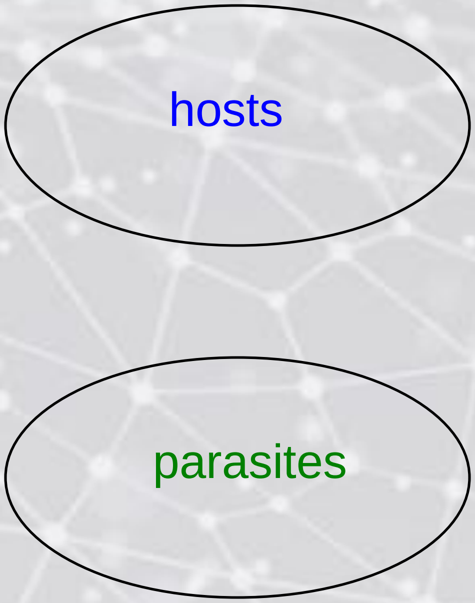
$\text{fitness}\,(h) = $ fraction of 9 *parasites p* (randomly chosen from parasite population) answered correctly

$$\text{fitness}\,(p) = \begin{cases} 0 & \text{if } h(\mathrm{p}) \text{ is correct} \\ > 0 & \text{if } h(p) \text{ is not correct} \end{cases}$$

for host h randomly chosen from host population

Non-Spatial Coevolution

- No spatial distribution of host and parasite populations



Each *h* is replaced by mutated copy of winner of tournament among itself and 8 randomly chosen hosts.

Each *p* is replaced by mutated copy of winner tournament among itself and 8 randomly chosen parasites.

$\text{fitness}\,(h) = \text{ fraction of } 9 \; \textit{parasites } p$ (randomly chosen from parasite population) answered correctly

$$\text{fitness}\,(p) = \begin{cases} 0 & \text{if } h(\mathrm{p}) \text{ is correct} \\ > 0 & \text{if } h(p) \text{ is not correct} \end{cases}$$

for host h randomly chosen from host population

- **Spatial Evolution:**
  - Same as spatial coevolution, except parasites don’t evolve.
  - A new population of random parasites is generated at each generation.

- **Non-Spatial Evolution:**

  – Same as non-spatial coevolution, except parasites don’t evolve.

  – A new sample of 100 random parasites is generated at each generation.

  – Fitness of a host is classification accuracy on these 100 randomly generated parasites

Results

	Function Induction	Cellular Automata
Spatial Coev.	**78%** (39/50)	**67%** (20/30)
Non-Spatial Coev.	**0%** (0/50)	**0%** (0/20)
Spatial Evol.	**14%** (7/50)	**0%** (0/30)
Non-Spatial Evol.	**6%** (3/50)	**0%** (0/20)

Percentage of successful runs

In short: Spatial coevolution significantly out-performs other methods on both problems

Possible applications to real-world problems

- Drug design to foil evolving viruses/bacteria
- Coevolving software/hardware with test cases
- Evolving game-playing programs
- Coevolving computer security systems with possible threats