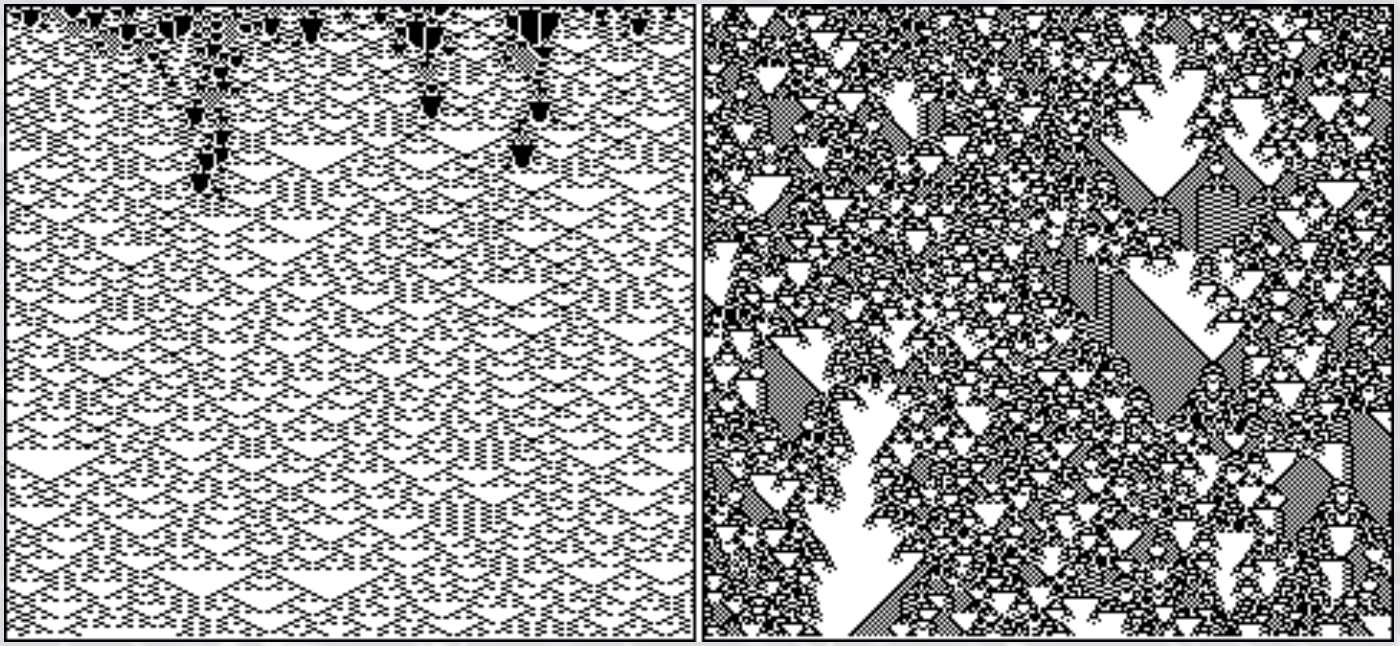
A.I.: Genetic Algorithms



Some Examples of Biologically Inspired AI

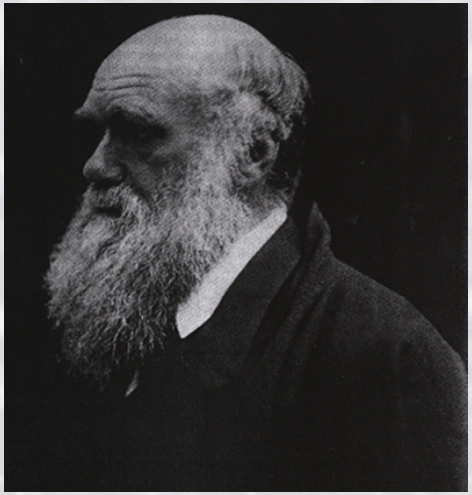
- Neural networks
- Evolutionary computation (e.g., genetic algorithms)
- Immune-system-inspired computer/network security
- Insect-colony optimization (ants, bees, etc.)
- Slime-mould path-finding
- Swarm intelligence (e.g., decentralized robots)

Evolutionary Computation

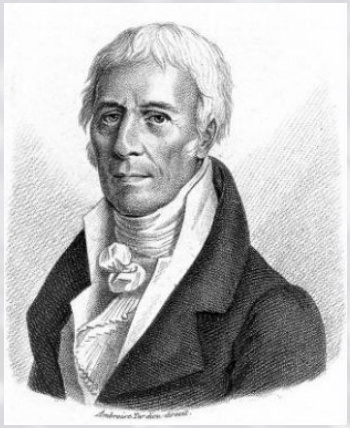
A collection of computational methods inspired by biological evolution:

- 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



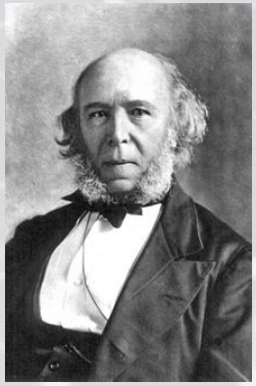
Charles Darwin
1809–1882



Lamarck

Wallace

Mendel



Spencer

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:

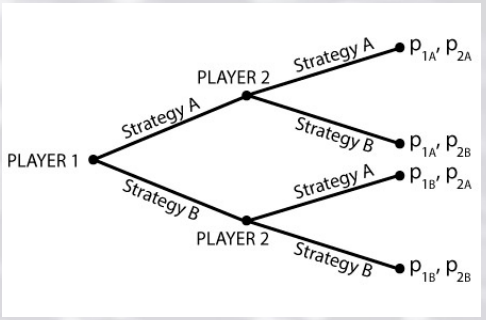
- Successful method of searching large spaces for good solutions (chromosomes / organisms)
- Massive parallelism
- Adaptation to environments, change
- Emergent complexity from simple rules

Genetic Algorithms: A very brief history

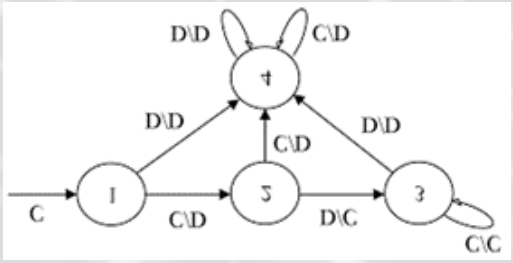
(*) Game Theory: is the study of mathematical models of strategic interaction between rational decision-makers; uses in social sciences, economics, A.I.

von Neuman

Nash



(*) In the 1960s Rechenberg introduced "evolution strategies" (1965), Walsh (1966) introduced "evolutionary programming", a technique in which candidate solutions to given tasks were represented as finite-state machines, which were evolved by randomly mutating their state-transition diagrams and selecting the fittest.



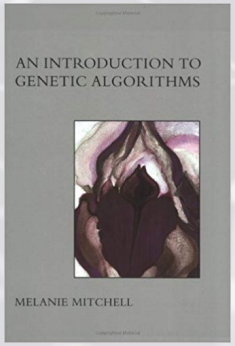
Genetic Algorithms: A very brief history

(*) Also in the 1960s, John Holland presented GAs as an abstraction of biological evolution and gave a theoretical framework for adaption under the GA.

Holland's GA is a method for moving from one population of "chromosomes" to another (e.g., strings of zeros/ones) to a new population by using a kind of "natural selection" together with the genetics-inspired operators of **crossover**, **mutation and inversion** (inversion reverses the order of a contiguous section of chromosomes).

(*) Until the 1990s, Holland's theoretical framework for GAs was the de facto approach for almost all subsequent theoretical work on genetic algorithms.

Holland

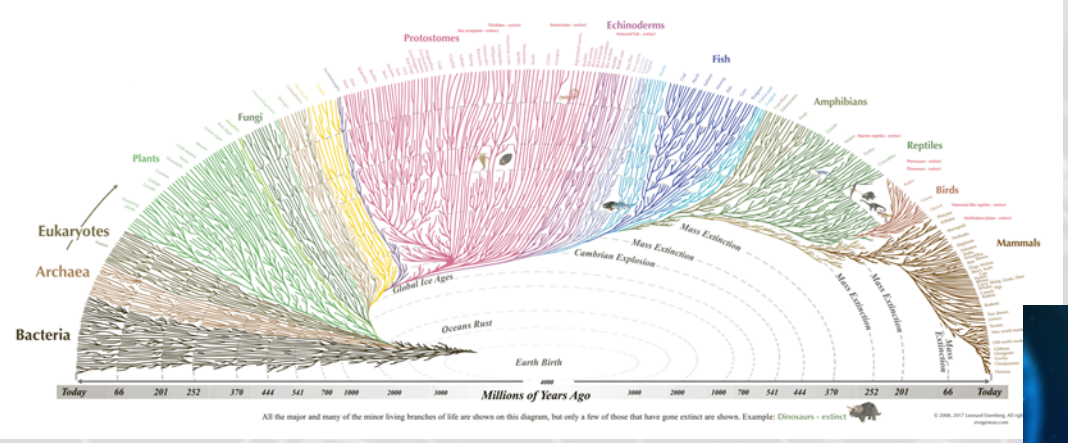


Mitchell

The Appeal of Evolution

(*) Many computational problems require searching through a huge number of possible solutions (e.g. protein folding).

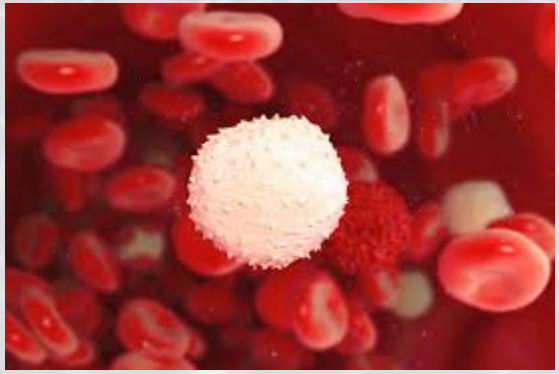
(*) In particular, for problems that benefit from an effective use of **parallelism**, GAs provide an attractive option.



The Appeal of Evolution

(*) In addition, many computational problems require a computer program to be **adaptive** – to continue to perform well in a changing environment (e.g. robot control). Furthermore, many computational problems require complex solutions that are difficult to program explicitly by hand (we often prefer an automated, bottom-up solution, using – if possible – "simple" rules).

(*) Biological evolution provides a method for searching among an enormous number of possibilities for "solutions": the enormous set of possibilities is the set of genetic sequences, and the desired solutions are highly fit organisms. In addition, evolution can be seen a method for **designing** innovative solutions to complex problems (e.g. mammalian immune system).



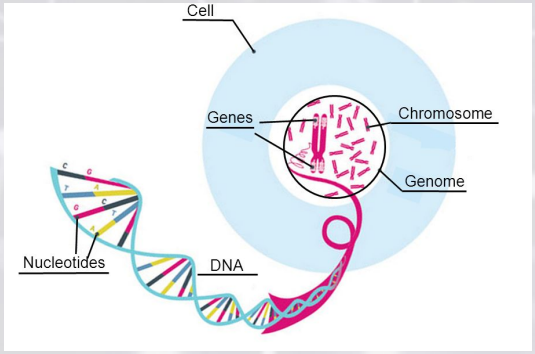
Biological Terminology

(*) All living organisms consist of cells, and each cell contains the same set of one or more **chromosomes** – strings of DNA – that serve as a "blueprint" for the organism.

(*) A chromosome can be conceptually divided into **genes** – functional blocks of DNA, each of which encodes a particular protein.

(*) Very roughly, we think of a gene as encoding a particular trait (e.g. eye color); the different possible settings for a trait (e.g. blue, green, hazel) are called **alleles** (alleles can be dominant/recessive); each gene is located at a particular **locus** (position) on the chromosome.

(*) Many organisms have multiple chromosomes in each cell. The complete collection of genetic material (all chromosomes) is called the organism's **genome**.

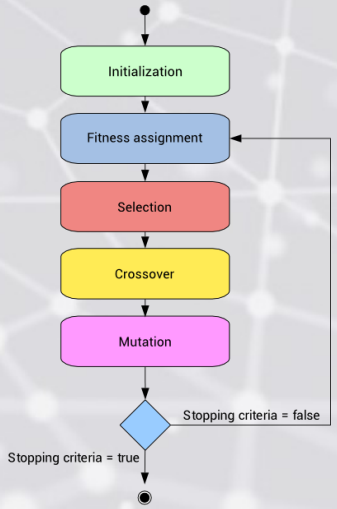


Biological Terminology

(*) **Genotype** refers to the particular set of genes contained in the genome; **phenotype** constitutes the physical and mental characteristics that manifest from the genotype.

(*) During sexual reproduction, **recombination** (or **crossover**) occurs: in each parent, genes are exchanged between each pair of chromosomes to form a *gamete* (a single chromosome), and then *gametes* from the two parents pair up to create a full set of chromosomes.

Offspring are subject to **mutation,** in which single nucleotides (elementary bits of DNA) are changed from parent to offspring; the **fitness** of an organism is typically defined as the probability the organism will live to reproduce.

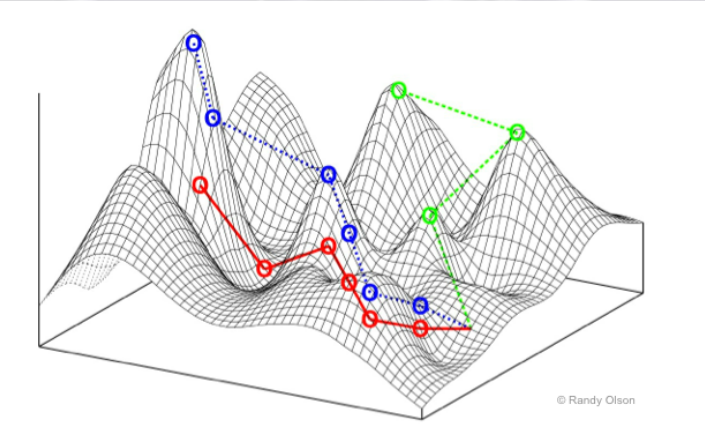


Search Spaces & Fitness Landscapes

(*) **Search space** refers to some collection of candidate solutions to a problem and some "distance" (e.g. Euclidean distance, Hamming distance) between candidate solutions.

(*) A **fitness landscape** (Wright, 1932) is a representation of the space of all possible genotypes along with their fitnesses. The fitness is typically represented by the "height" of the landscape; genotypes which are similar are said to be "close" to each other. Collectively, the set of all possible genotypes, their degree of similarity, and their related fitness values is then called the fitness landscape.

(*) Under Wright's formulation, evolution causes populations to move along landscapes in particular ways, and "adaptation" can be seen as the movement toward local peaks.



Some Applications of GAs

(*) **Optimization**: GAs have been used in a wide variety of optimization tasks, including numerical optimization and such combinatorial optimizations problems as circuit layout and job-shop scheduling.

(*) **Automatic Programming**: GAs have been used to evolve computer programs for specific tasks, and to design other computational strctures such as cellular automata.

(*) **Machine Learning**: GAs have been used for many ML applications, including classification and prediction; GAs have also been used to evolve aspects of particular ML systems, such as weights of a NN, learning a network architecture, and hyperparameter tuning (see examples below in lecture).

(*) **Economics**: GAs have been used to model processes of innovation, the development of bidding strategies, and the mergence of economic markets.

(*) **Immune system**: GAs have been used to model various aspects of natural immune systems, including somatic mutation during an individual's lifetime.

(*) **Ecology/Social Systems**: GAs have been used to model ecological phenomena such as "biological arms races", host-parasite coevolution, symbiosis and resource flow, insect colonies and cooperation-communication in multi-agent systems.

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.

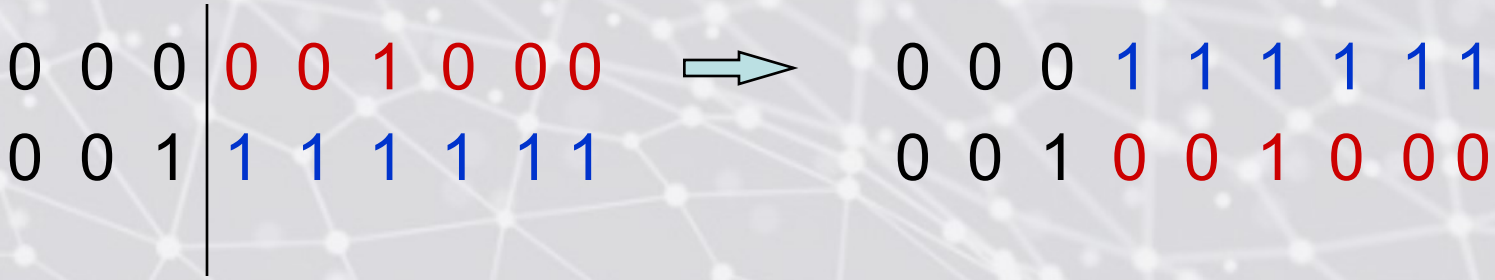
3. Select pairs of parents with probability a function of fitness rank in the population.

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

5. Go to step 2.

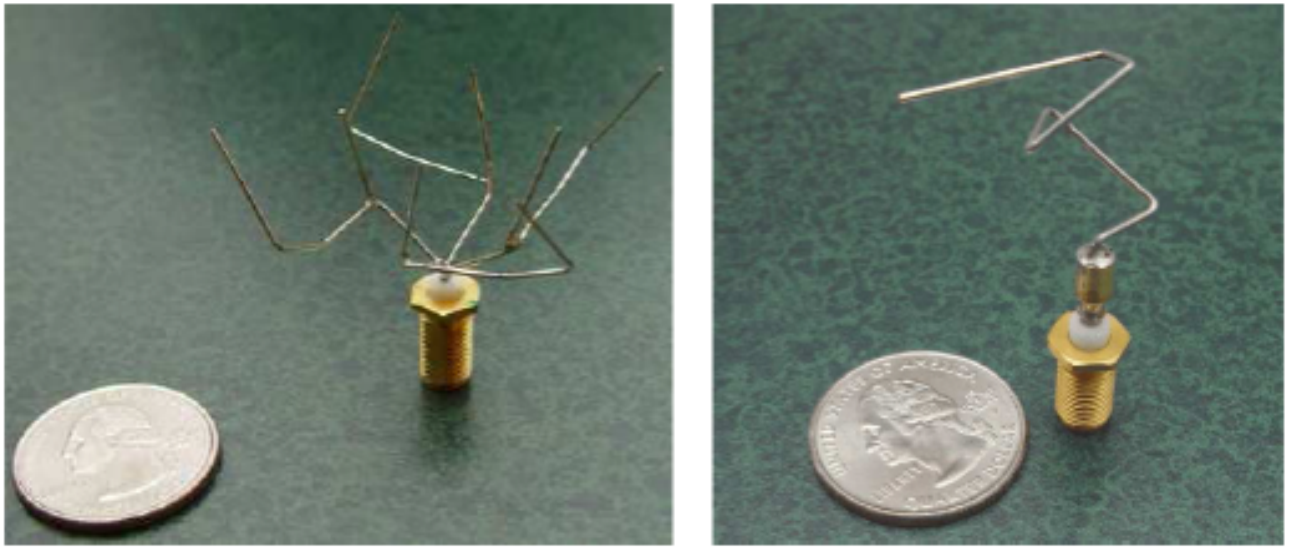
Genetic operators

- *Crossover:* exchange subparts of two chromosomes:



- *Mutation:* randomly change some loci:

From Hornby et al., 2006

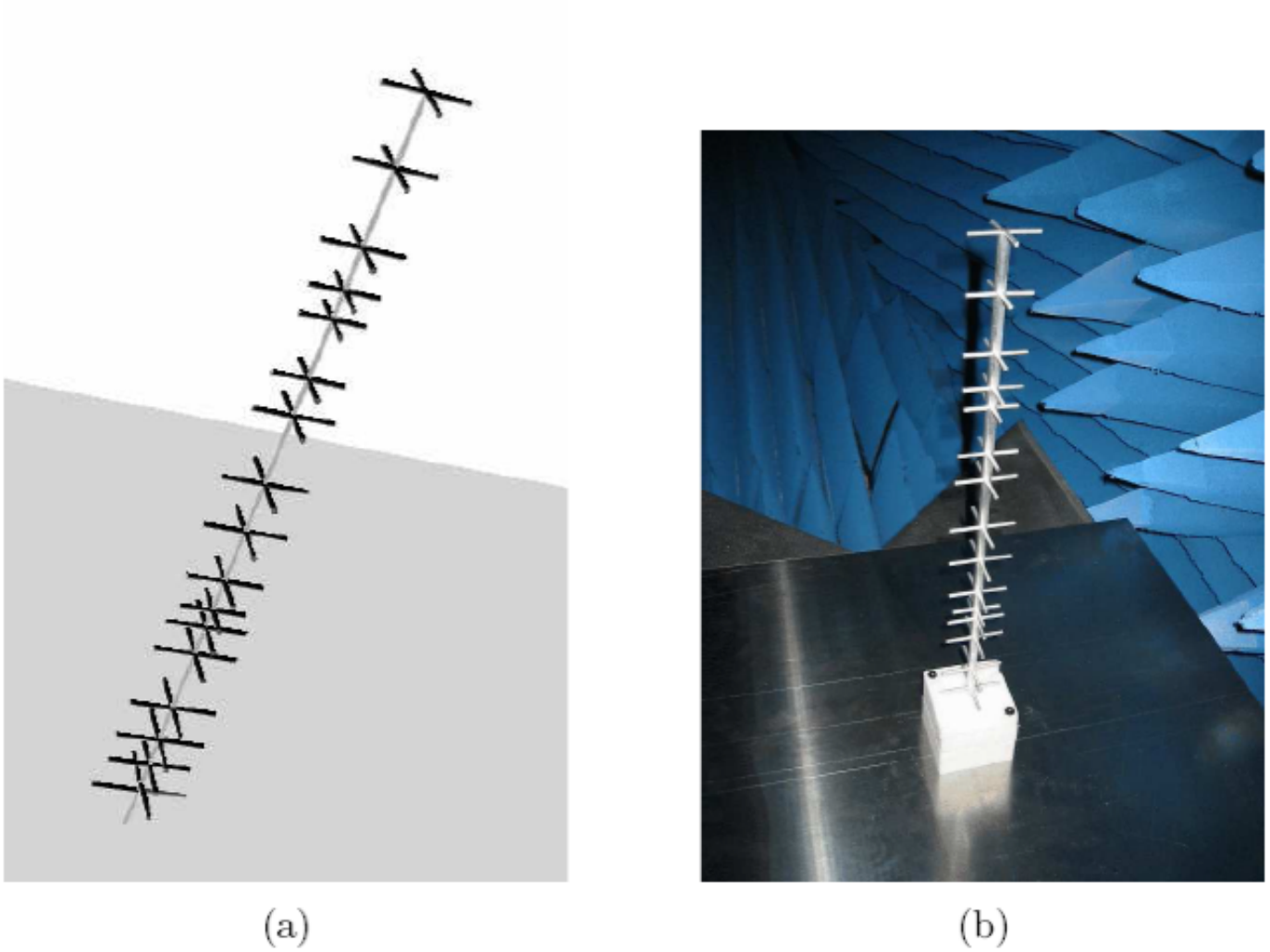


(a) (b)

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.

Using GAs to Evolve Strategies for PD

(*) **Prisoner's Dilemma** (Flood/Fresher, 1950) is a canonical example from game theory that shows why two completely rational individuals might not cooperate, even if it appears it is in their best interests to do so.

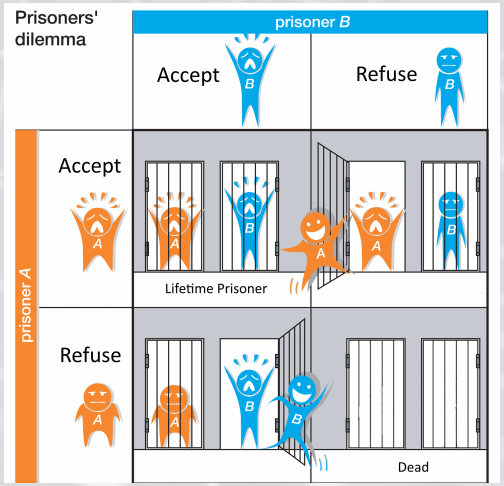
(*) PD has many variants, but in the standard form it is usually stated as follows:

Two members of a criminal ~~administration~~ gang are arrested and imprisoned. Each prisoner is in solitary confinement with no means of communicating with the other. The prosecutors lack sufficient evidence to convict the pair on the principal charge, but they have enough to convict both on a lesser charge. Simultaneously, the prosecutors offer each prisoner a bargain. Each prisoner is given the opportunity either to betray the other by testifying that the other committed the crime, or to cooperate with the other by remaining silent. The offer is:

If A and B each betray the other, each of them serves four years in prison

If A betrays B but B remains silent, A will be set free and B will serve five years in prison (and vice versa)

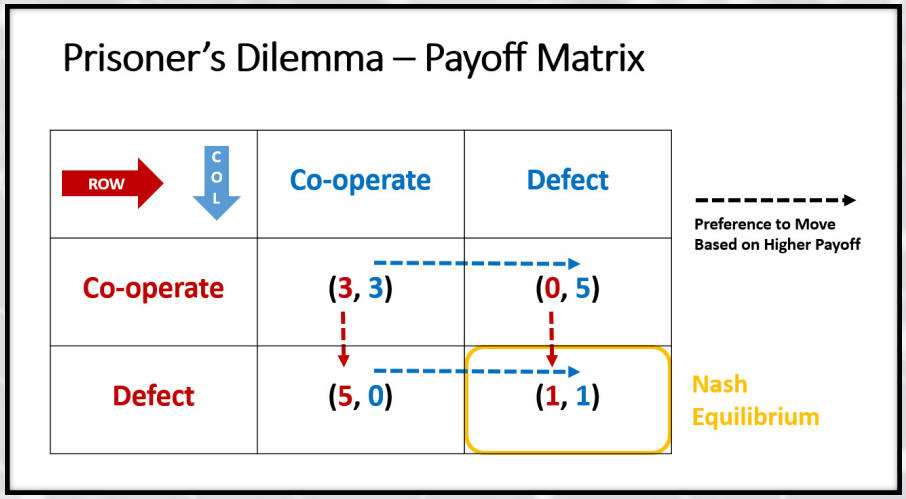
If A and B both remain silent, both of them will only serve two years in prison (on the lesser charge).



Using GAs to Evolve Strategies for PD

(*) The game can be described more abstractly. Each player independently decides which move to make – i.e., whether to cooperate or defect. A "game" consists of each player's making a decision (viz., a "move").

(*) One can use a **payoff matrix** to summarize the possible results:



(*) The payoff in each case can be interpreted as "5 minus the number of years in prison."

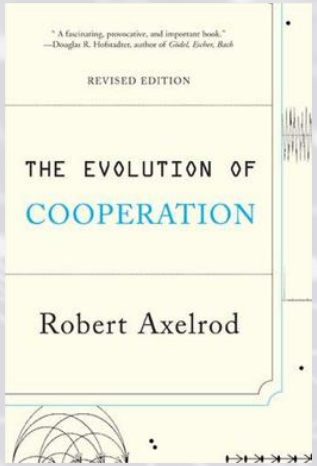
(*) Notice that no matter what the other played does, it is always better to defect. The dilemma is that if both players defect, each gets a worse score than if they cooperate.

Using GAs to Evolve Strategies for PD

(*) PD can be played iteratively (that is, if two players play several games in a row). For the iterated version, if both players' always defect, this will lead to a much lower total payoff than the players would get if they cooperated.

How can reciprocal cooperation be induced? This question takes on special significance when the notion of cooperating and defecting correspond to actions in, say, a real-world arms race.

(*) Robert Axlerod of University of Michigan decided to see if a GA could evolve strategies to play iterated PD successfully (1987).



Encoding scheme for PD:

C C (both players cooperate)

D C (first player defects, second cooperates), etc.

(*) Axlerod's tournaments involved strategies that remembered three previous games. There are 64 (2^6) possibilities for the previous games:

C C C C C C, C C C C C D, etc.

Using GAs to Evolve Strategies for PD

(*) Technically, Axlerod used 70-letter strings, where the six extra letters encoded three hypothetical previous games used by the strategy to decide how to move in the first actual game (so the search space was of size: 2^{70})

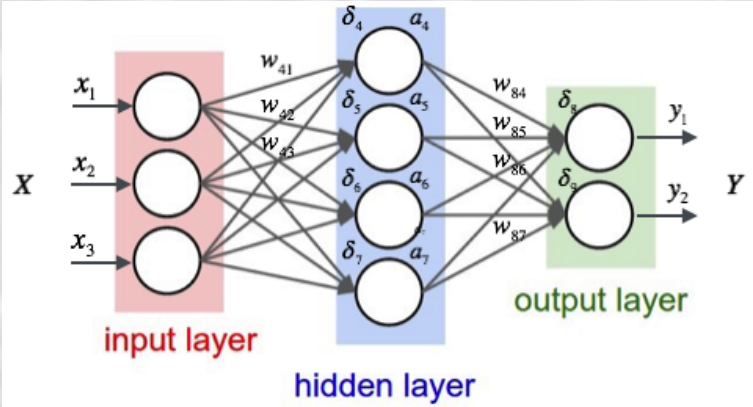
(*) As a fitness measure, he scored a given strategy against a set of 8 representative, hand-crafted strategies.

(*) Axlerod performed 40 different runs of 50 generations each, using different random-number seeds for each run. Most of the strategies that evolved were similar to the best human-crafted strategy ("TIT for TAT" – reciprocate cooperation and punish defection), several other strategies evolved by the GA scored even higher than any human-crafted strategy. (this is remarkable if we consider that only 40 x 50 = 2000 individuals out of 2^{70} were actively searched).

(*) Axlerod concluded that the GA is good at doing what evolution does: developing highly specialized adaptations to specific characteristics of the environment.

(*) Note that this initial experiment consisted of a fixed environment (i.e. the competing strategies did not change); he later applied GAs to PD with a non-static fitness landscape.

Using GAs to Evolve Weights in a NN



(*)David Montana and Lawrence Davis (1989) first attempted to evolve the weights of a (fixed size) NN using GAs. That is to say, they used a GA instead of backpropagation.

(*) They used a shallow NN with only two layers of hidden units (7-10); the networks were feedforward and fully-connected. In total there were 108 weighted connections between units; in addition there were 18 weighted connections between the non-input units and a "threshold unit", for a **total of 126 weights**.

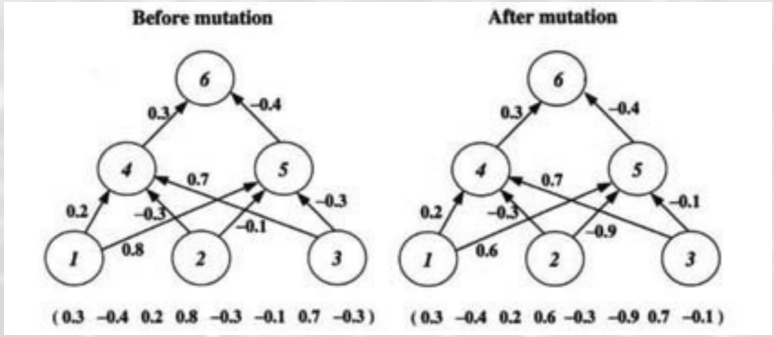
(*) Each chromosome was a list of 126 weights; to calculate the fitness of a given chromosome, the weights in the chromosome were assigned to the links in the network, the network was then run on the training data (236 images for binary classification).

(*) As usual, the "error" was the difference between the desired output activation and the actual network output; low error equates with high fitness.

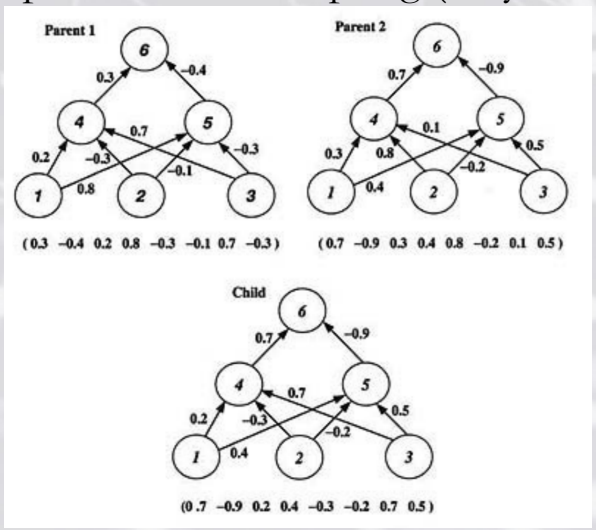
Using GAs to Evolve Weights in a NN

(*) An initial population of 50 weight vectors was chosen randomly in the range [-1,1].

(*) The **mutation operator** selects *n* non-input units and, for each incoming link to those units, adds a random value between -1 and +1 to the weight link.

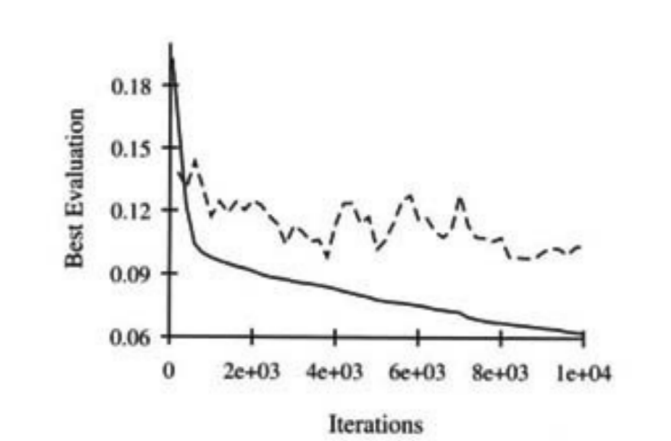


(*) The **crossover operator** takes two parent weight vectors and, for each non-input unit in the offspring vector, selects one of the parents at random and copies the weights on the incoming links from that parent to the offspring (only one offspring is created).



Using GAs to Evolve Weights in a NN

(*) The performance of a GA using these operators was compared with the performance of a back-propagation algorithm (solid line: GA; broken line: backprop; x-axis gives the number of iterations, and y-axis gives the best evaluation found by that time).



The GA had a population of 50 weight vectors and ran for 200 generations; backprop ran for 5000 iterations.

(*) The experiment showed that in some situations the GA is a better training method than simple backprop; this does not mean the GA will outperform backprop in all cases (or its variants).

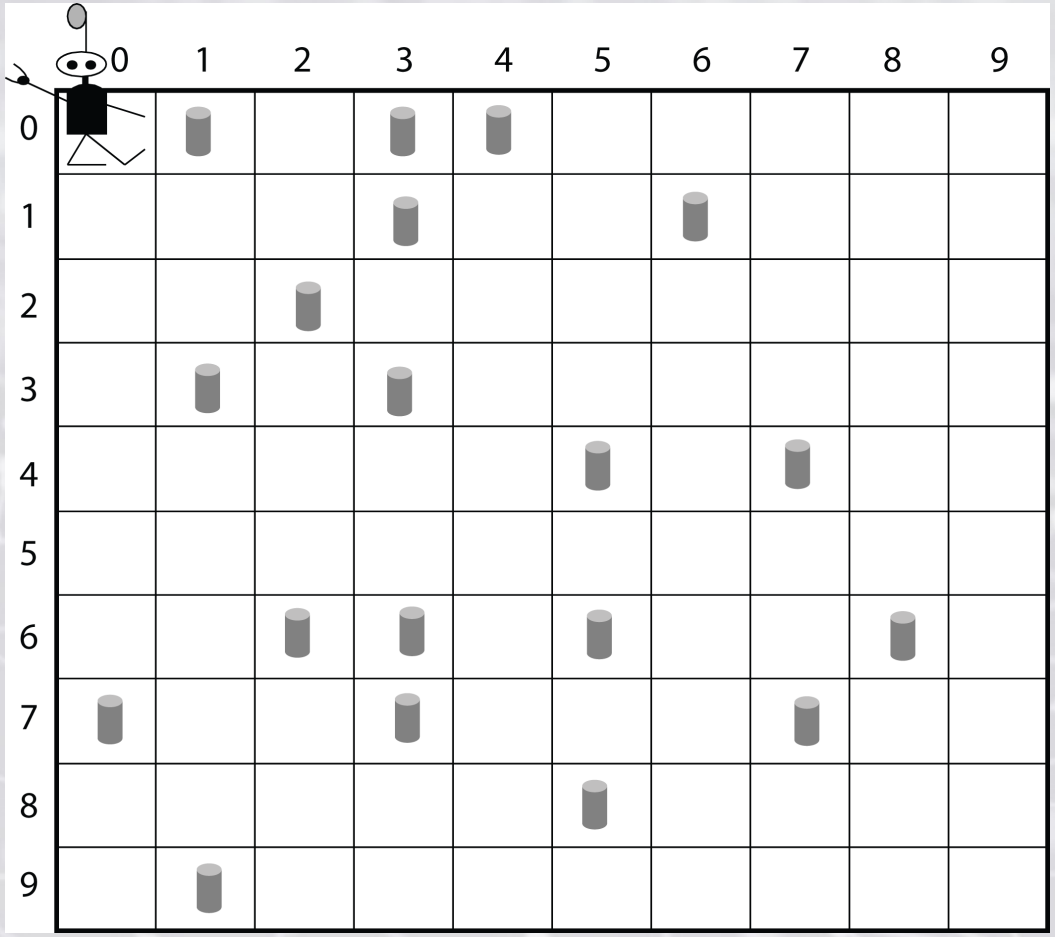
(*) Subsequent research has used GAs to also learn/evolve network architectures (using both direct and indirect encoding schemes), in addition to evolving a learning rule for weight updates.

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:
2330032342163034353054600610256251511416226043565433406651151415650220640642051006643216161521652022364433363346013326503000406220502431650061113051466642324012456333455241261434413610201506306425516540432644631561645105436653463105516460051 64

Individual 2:
1641134312102536034036124143120110423546252530420204451643366561035322153105131440622120614631432154610256523644422025340345305020056206340263310024534164301516312100122144006640126652463516501541231131324533044332126345550053142130644233110 00

Individual 3:
2042334440241122613213645263246421220612212225266062614443612532512664061335340153411110206164226653145522540234051155031302220200654451250622066314261355320100004000316401301541601620061344406261605056414215531332360215033551312536326426305 51

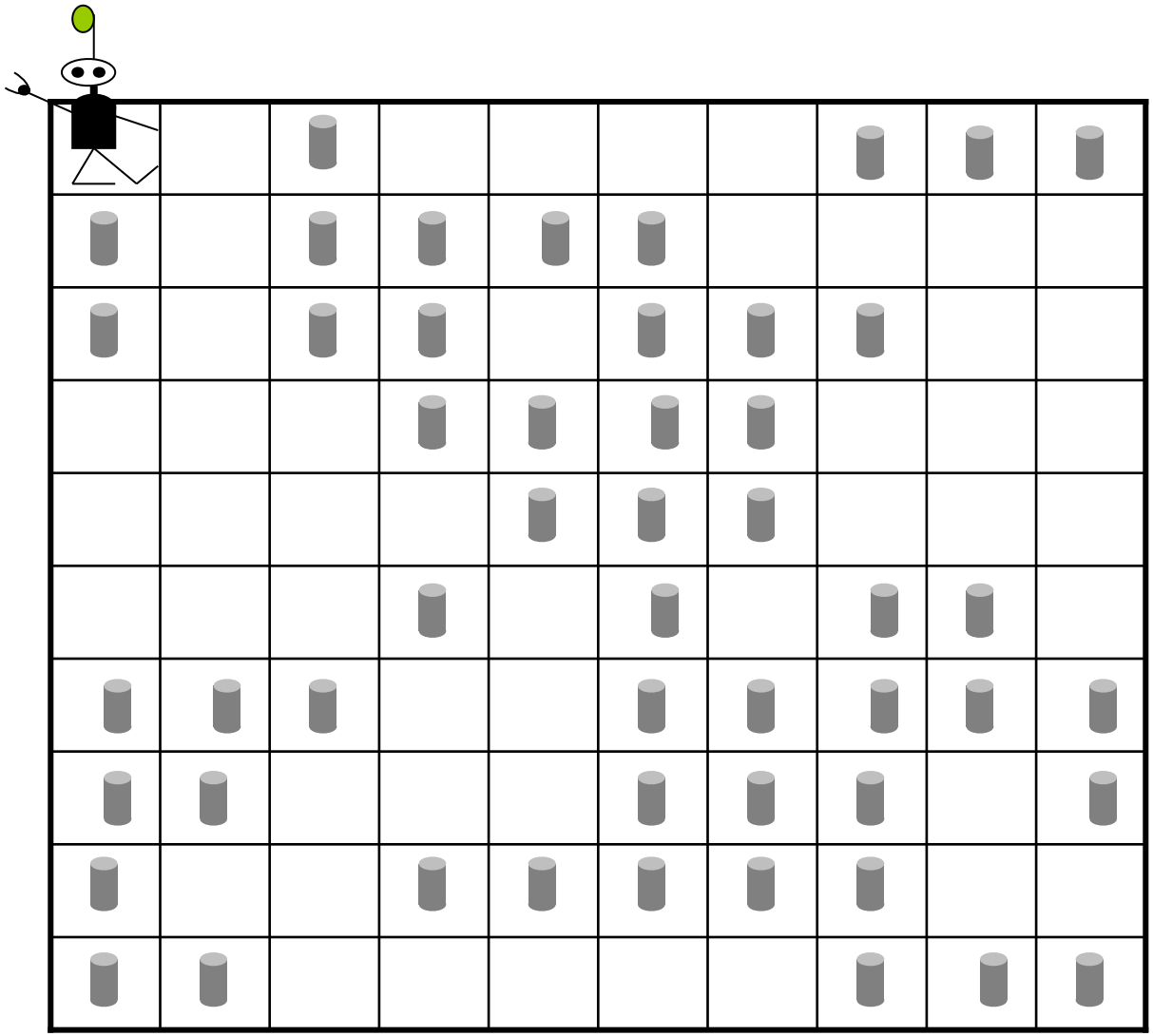
.
.
.

Individual 200:
3463252513600101222561210604330113520515532013065600532223504332425064124255265534635345523053326612010632124554423440613654302462401606630164646411030265400063341261503522621060636242605506166163442551243544641100234633304401025332121424022 51

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:

164113431210253603403612414312011042354625253042020445164336656103532215310513144062212061463143215461025652364442202534034530502005620634026331002453416430151631210012214400664012665246351650154123113132453304433212634555005314213064423311000

Parent 2:

204233444024112261321364526324642122061221222526606261444361253251266406133534015341111020616422665314552254023405115503130222020065445125062206631426135532010000400031640130154160162006134440626160505641421553133236021503355131253632642630551

Parent 1:

164113431210253603403612414312011042354625253042020445164336656103532215310513144062212061463143215461025652364442202534034530502005620634026331002453416430151631210012214400664012665246351650154123113132453304433212634555005314213064423311000

Parent 2:

204233444024112261321364526324642122061221222526606261444361253251266406133534015341111020616422665314552254023405115503130222020065445125062206631426135532010000400031640130154160162006134440626160505641421553133236021503355131253632642630551

Parent 1:

1641134312102536034036124143120110423546252530420204451643366561035322153105131440622120614631432154610256523644422025340345305020056206340263310024534164301516312100122144006640126652463516501541231131324533044332126345550053142130644233110000

Parent 2:

2042334440241122613213645263246421220612212225266062614443612532512664061335340153411110206164226653145522540234051155031302220200654451250622066314261355320100004000316401301541601620061344406261605056414215531332360215033551312536326426305510

Child:

1641134312102536034036124143120110423546252530420204451643366561035322153105131440622120614631432154610256523644422025340345305020056206340263310024561355320100004000316401301541601620061344406261605056414215531332360215033551312536326426305510

Parent 1:

1641134312102536034036124143120110423546252530420204451643366561035322153105131440622120614631432154610256523644422025340345305020056206340263310024534164301516312100122144006640126652463516501541231131324533044332126345550053142130644233110000

Parent 2:

2042334440241122613213645263246421220612212225266062614443612532512664061335340153411110206164226653145522540234051155031302220200654451250622066314261355320100004000316401301541601620061344406261605056414215531332360215033551312536326426305510

Mutate to “0”

Child:

1641134312102536034036124143120110423546252530420204451643366561035322153105131440622120614631432154610256523644422025340345305020056206340263310024561355320100004000316401301541601620061344406261605056414215531332360215033551312536326426305510

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 found!

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."

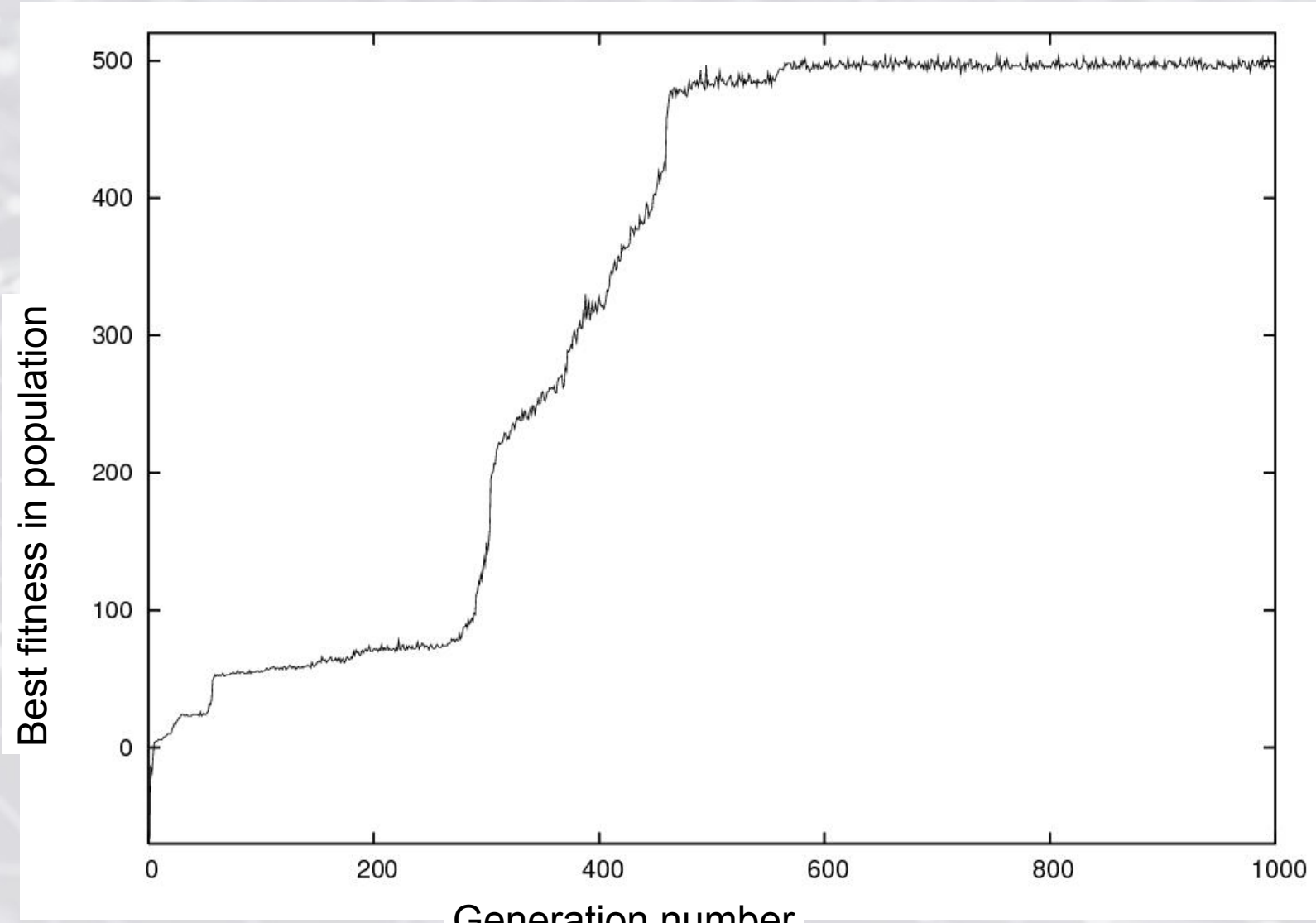
"Otherwise, if there is a can in one of the adjacent sites, move to that site."

"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)

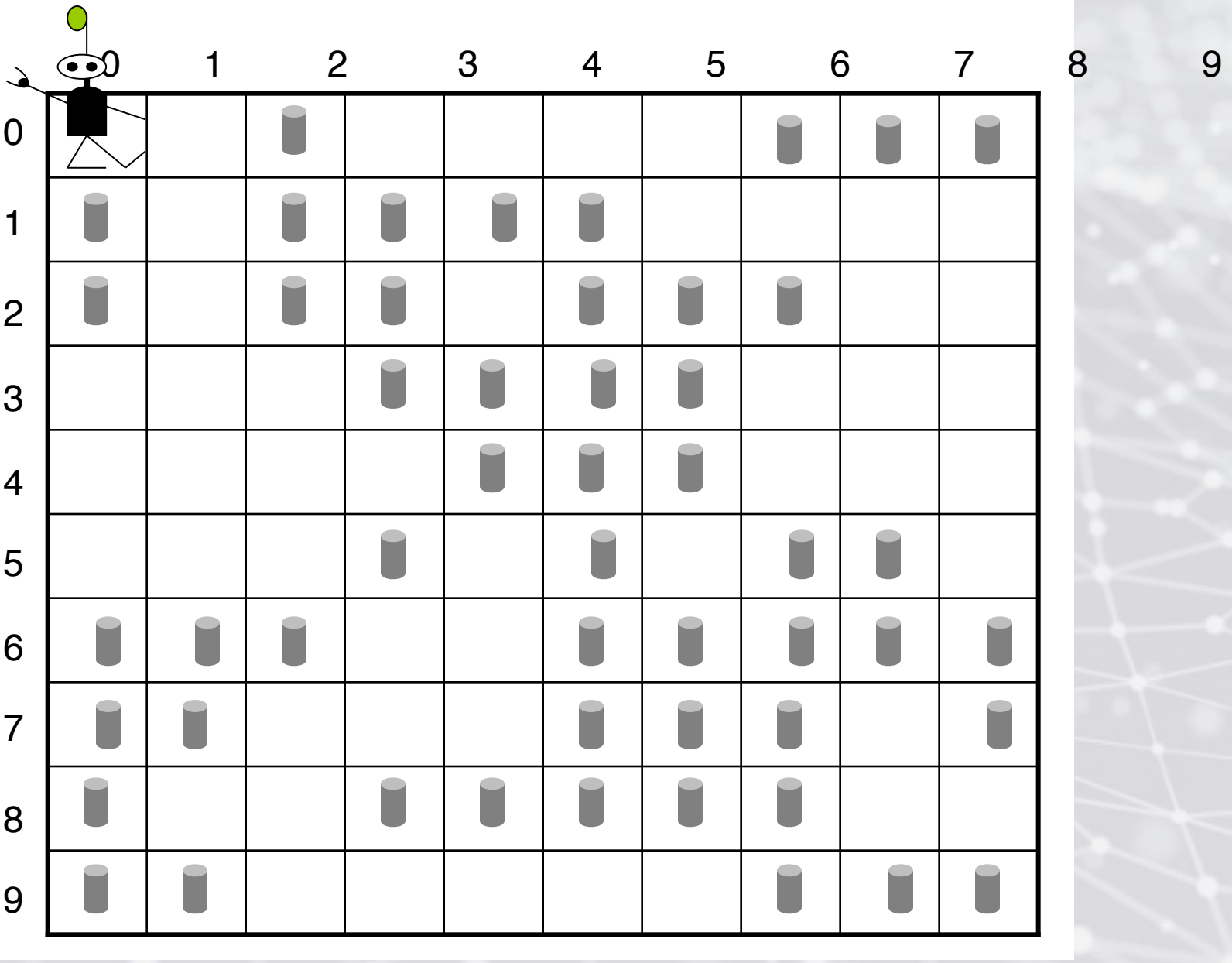
One Run of the Genetic Algorithm



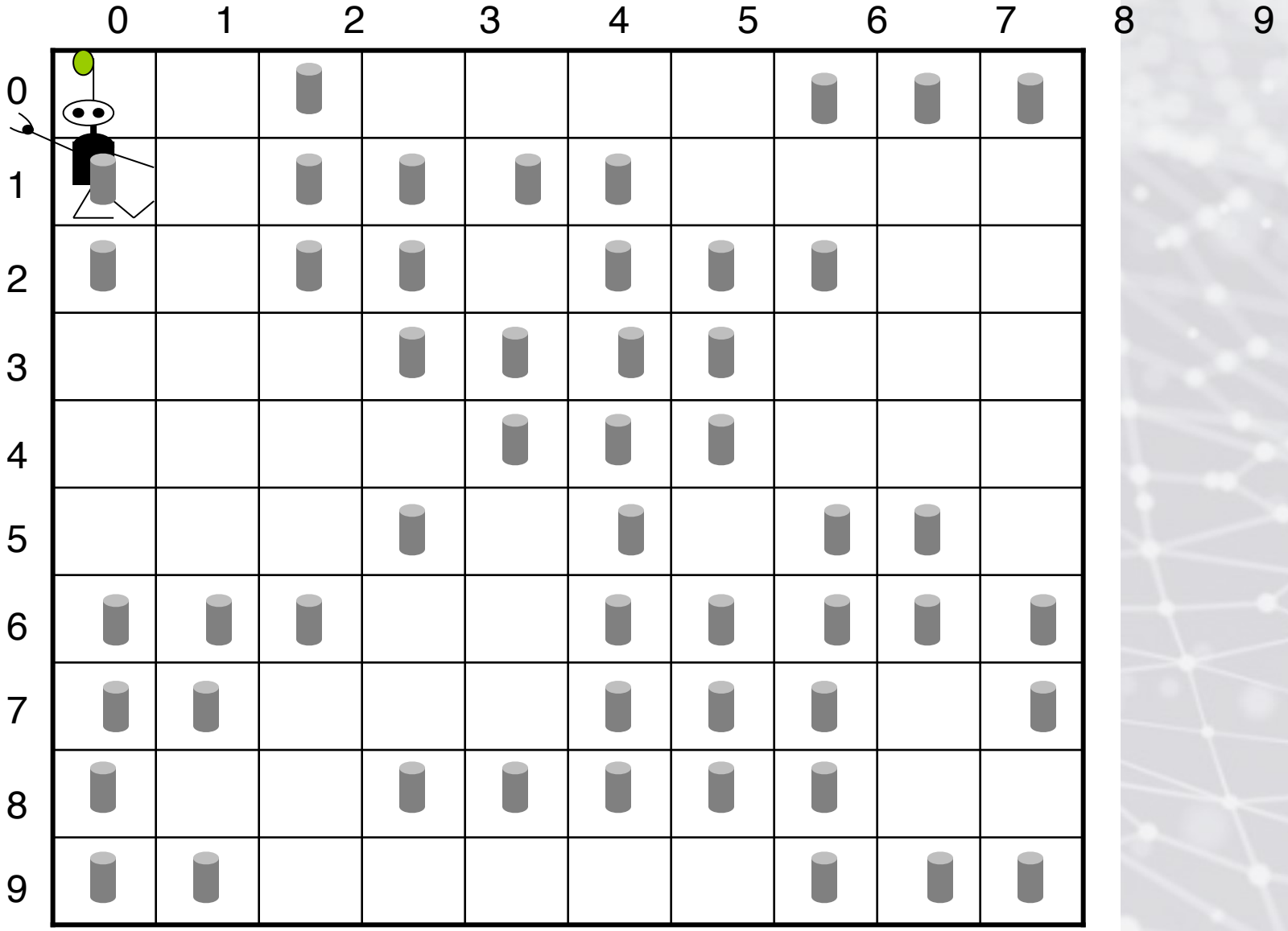
Generation 1

Best fitness = –81

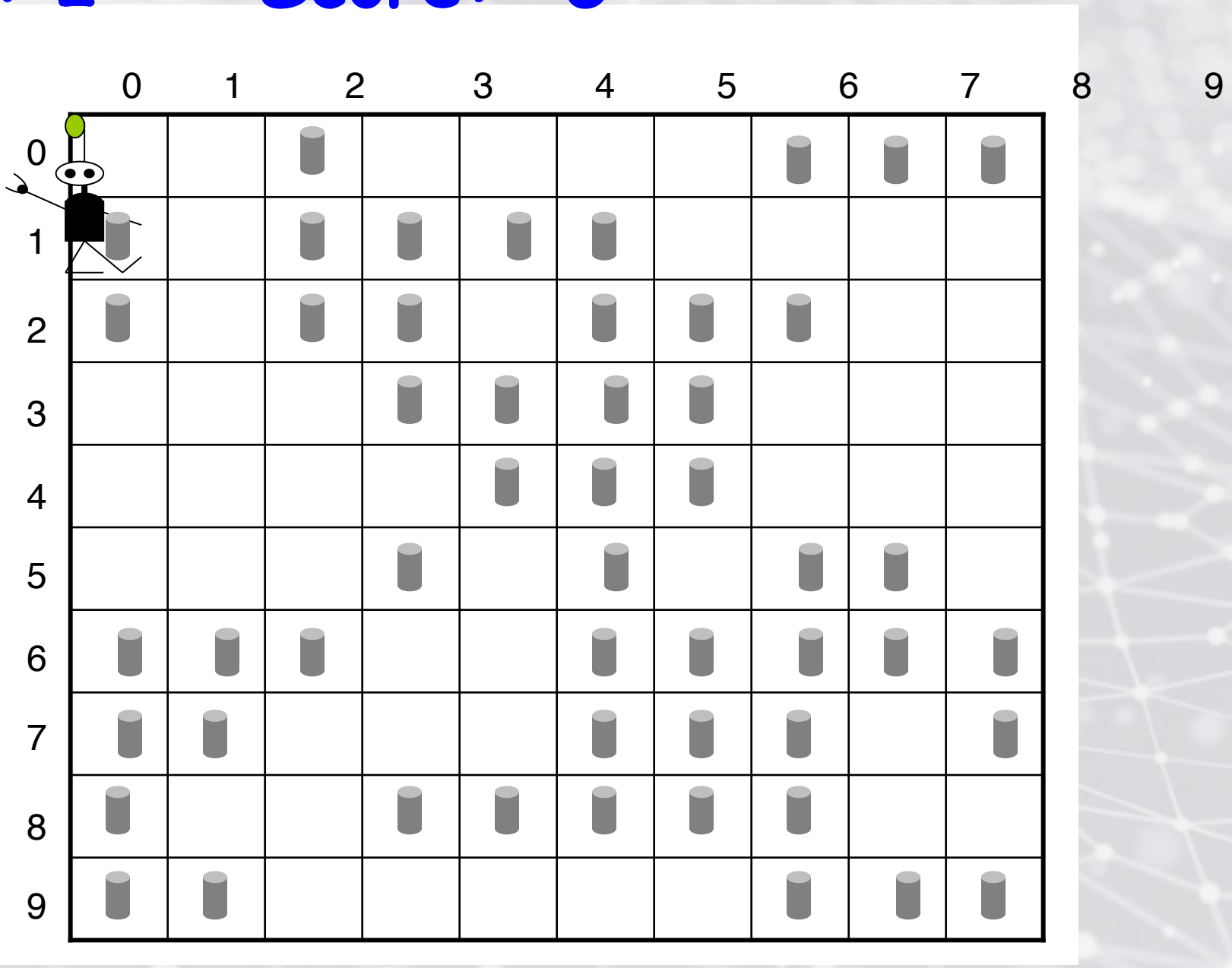
Time: 0 Score: 0



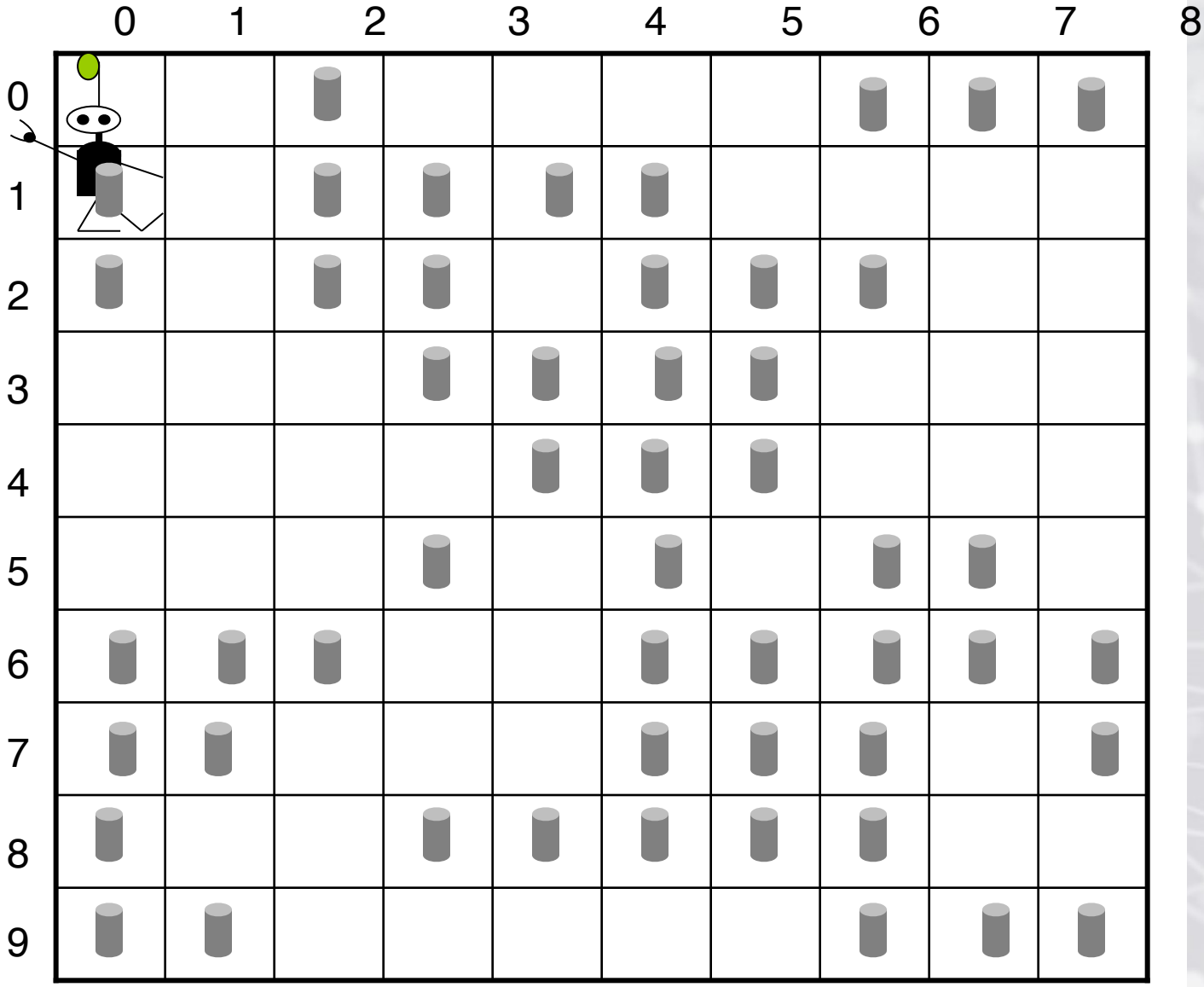
Time: 1 Score: 0



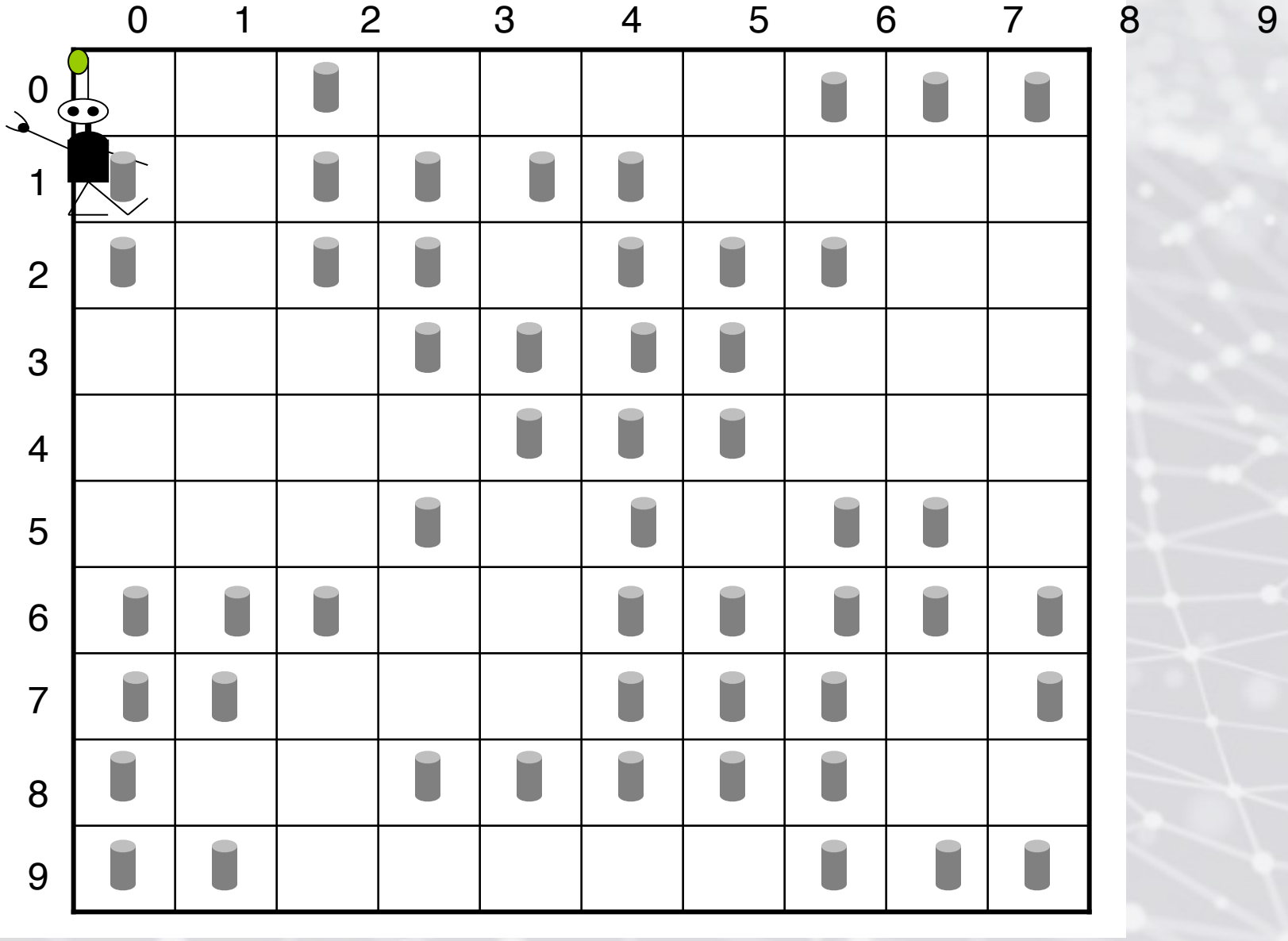
Time: 2 Score: −5



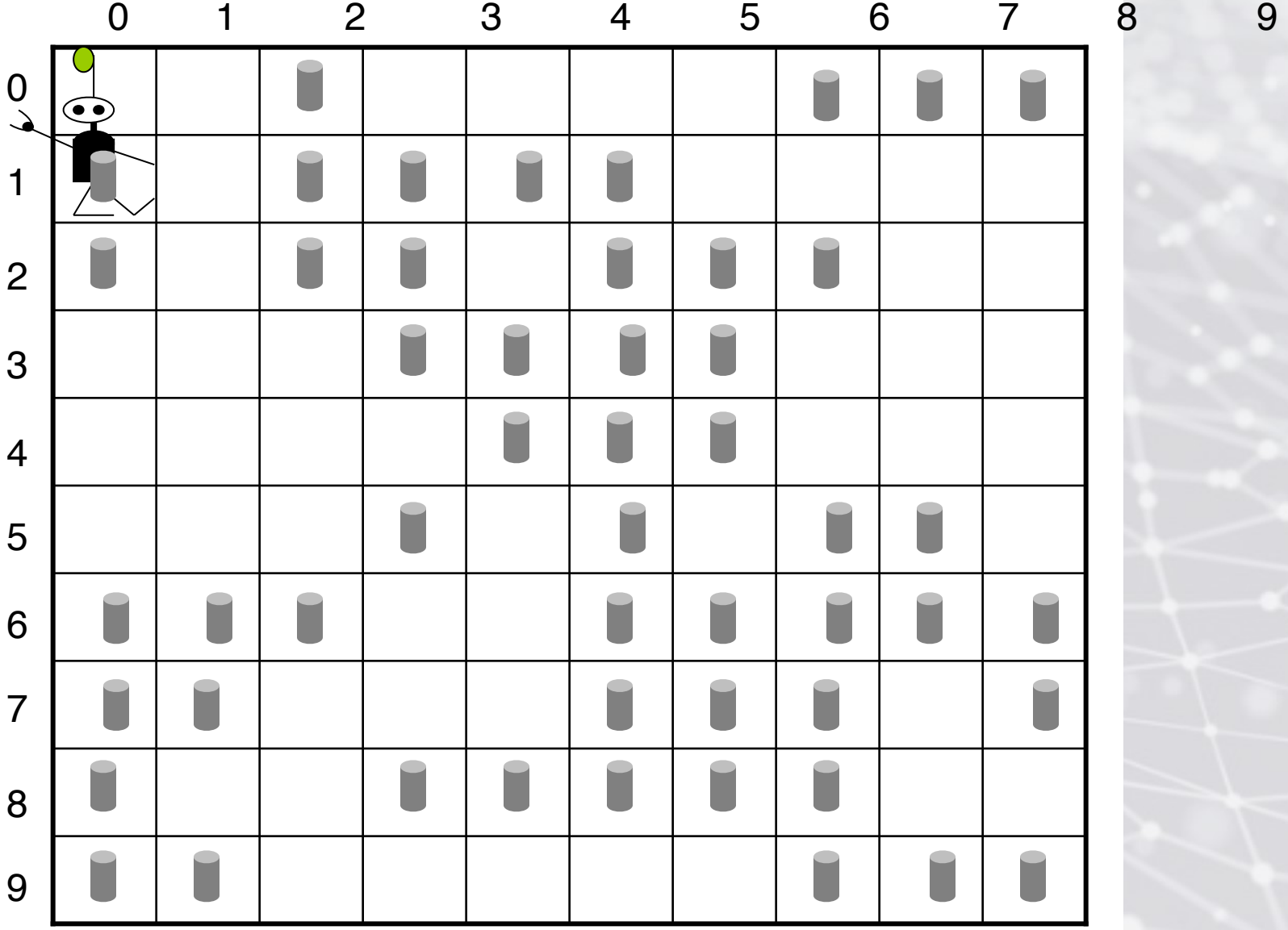
Time: 2 Score: –5



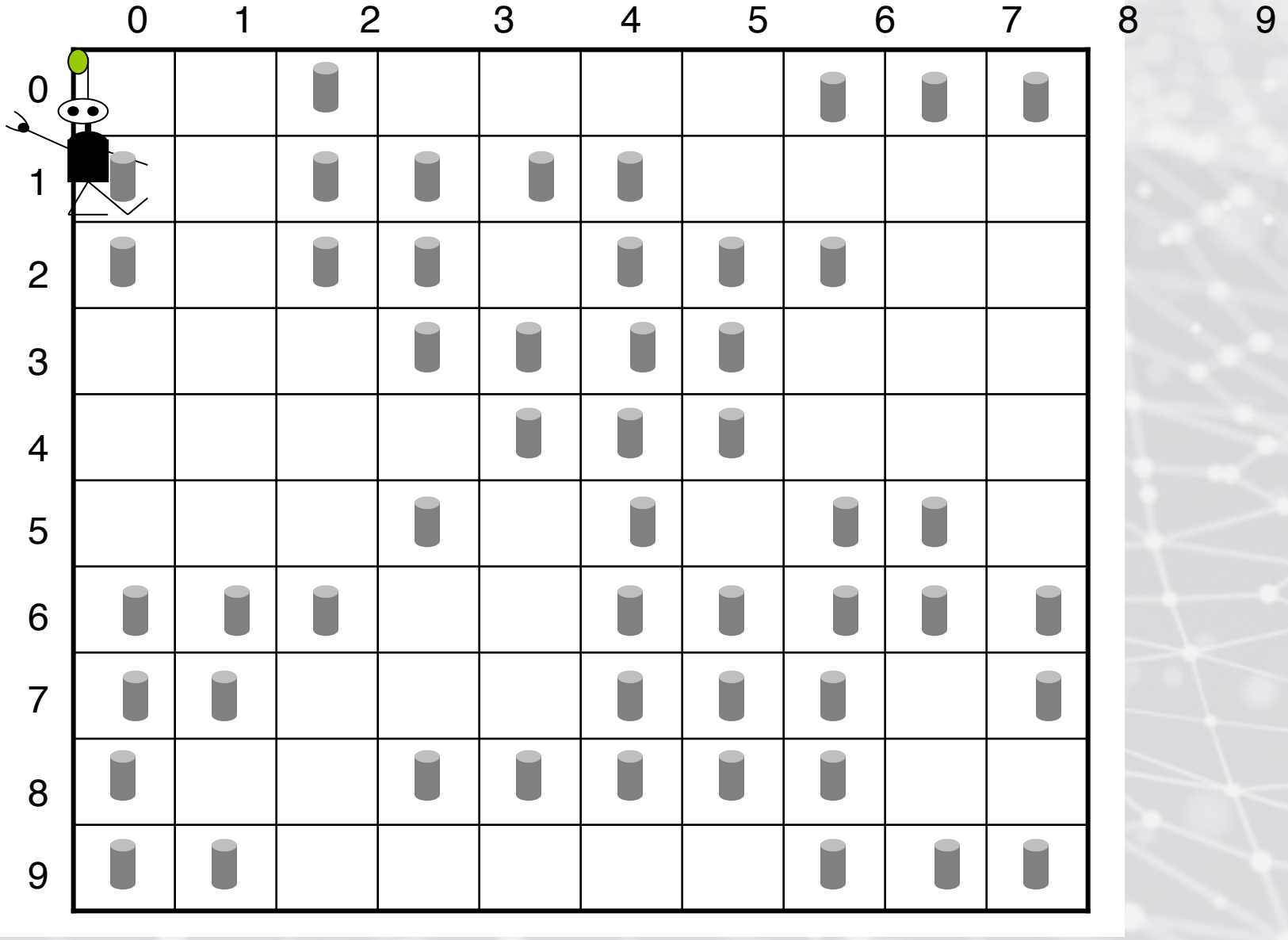
Time: 3 Score: -10



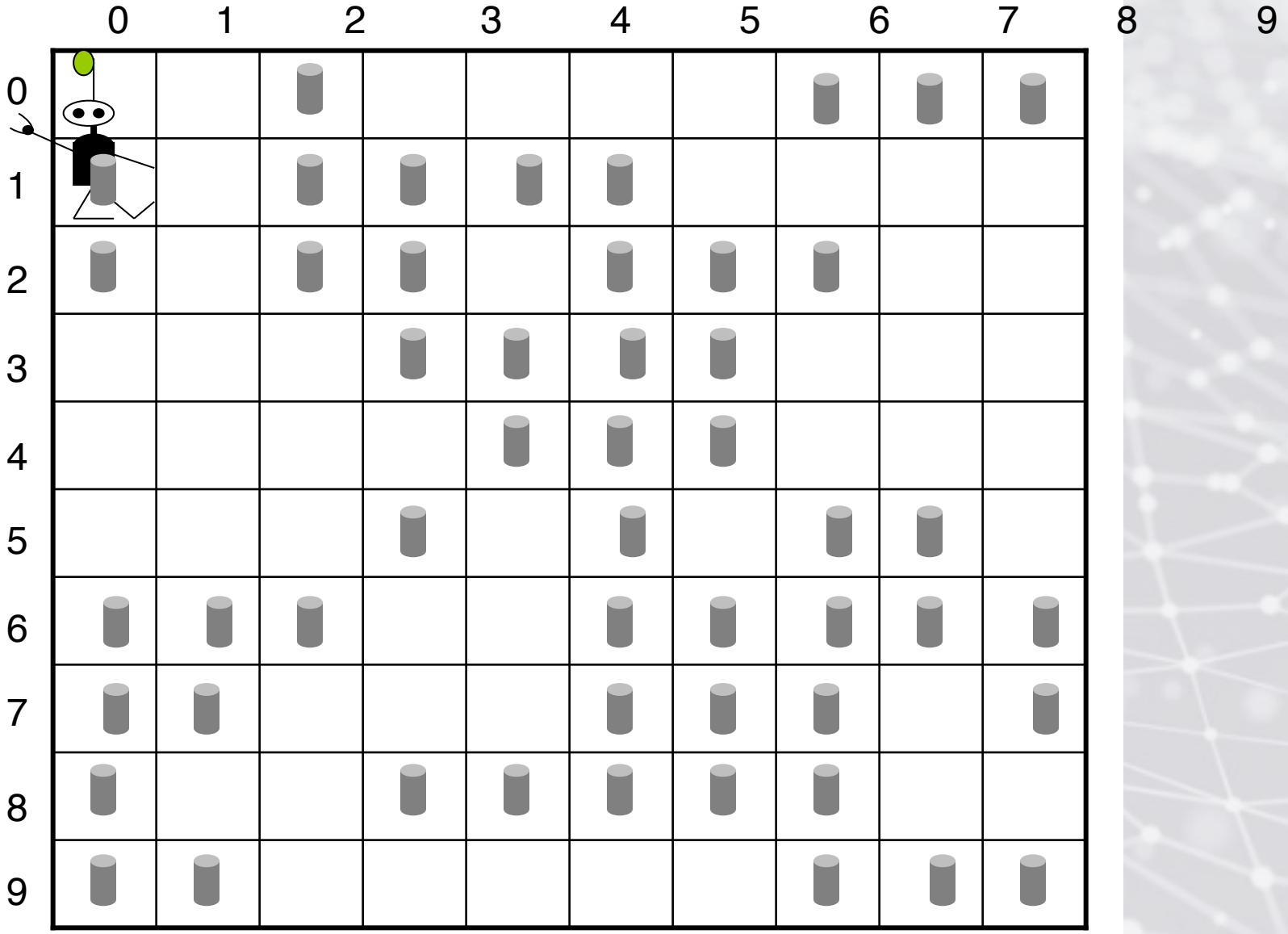
Time: 3 Score: -10



Time: 4 Score: −15



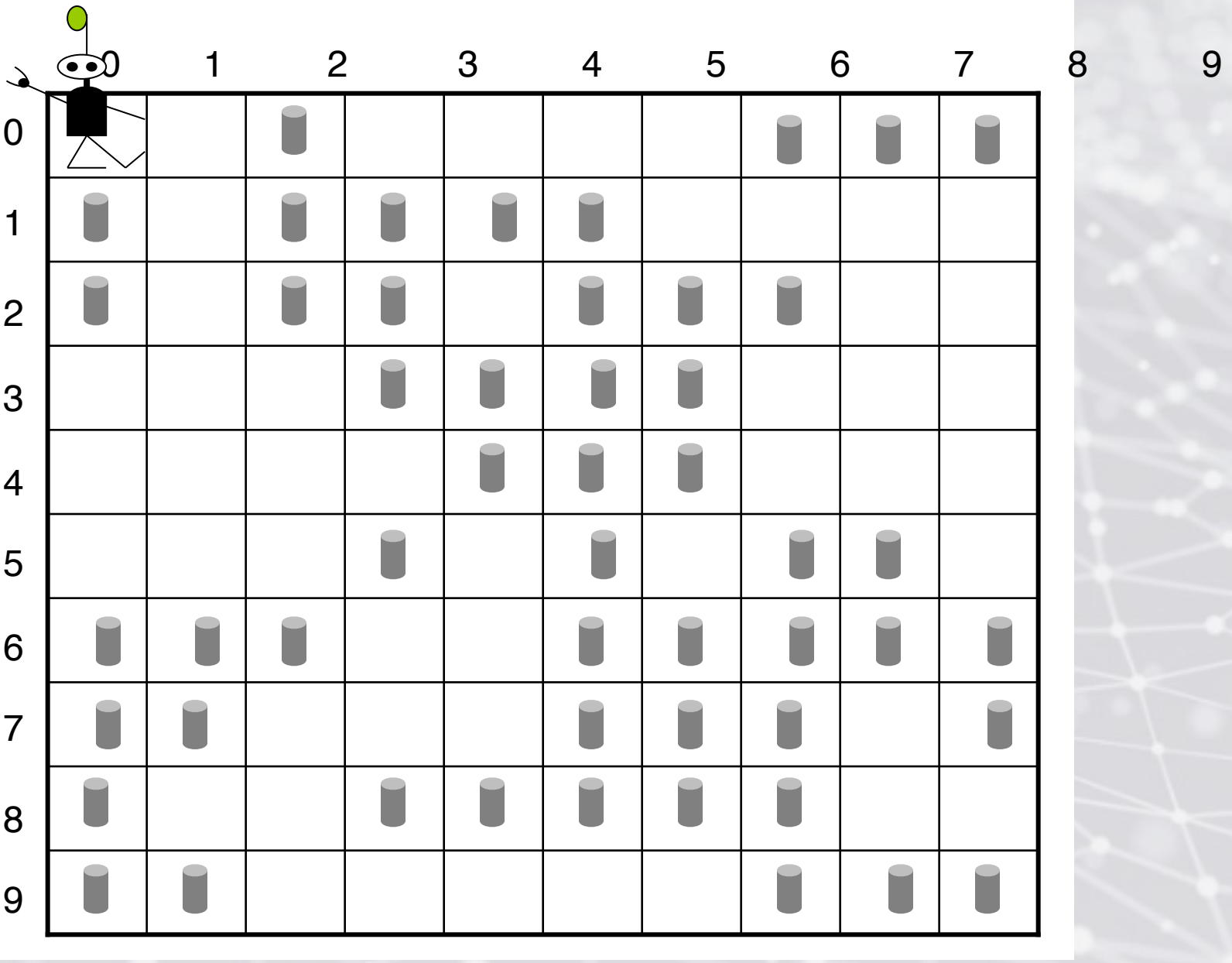
Time: 4 Score: −15



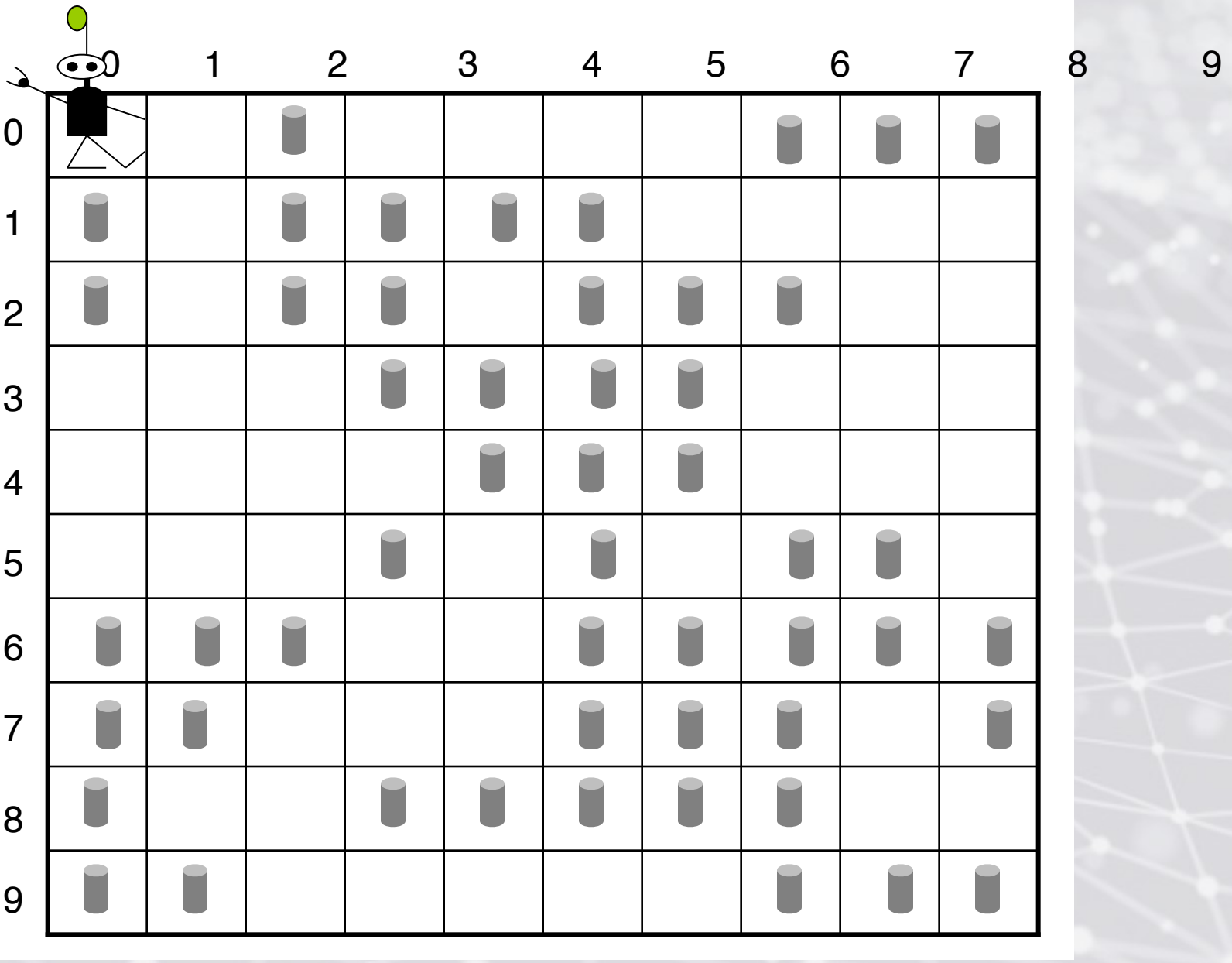
Generation 14

Best fitness = 1

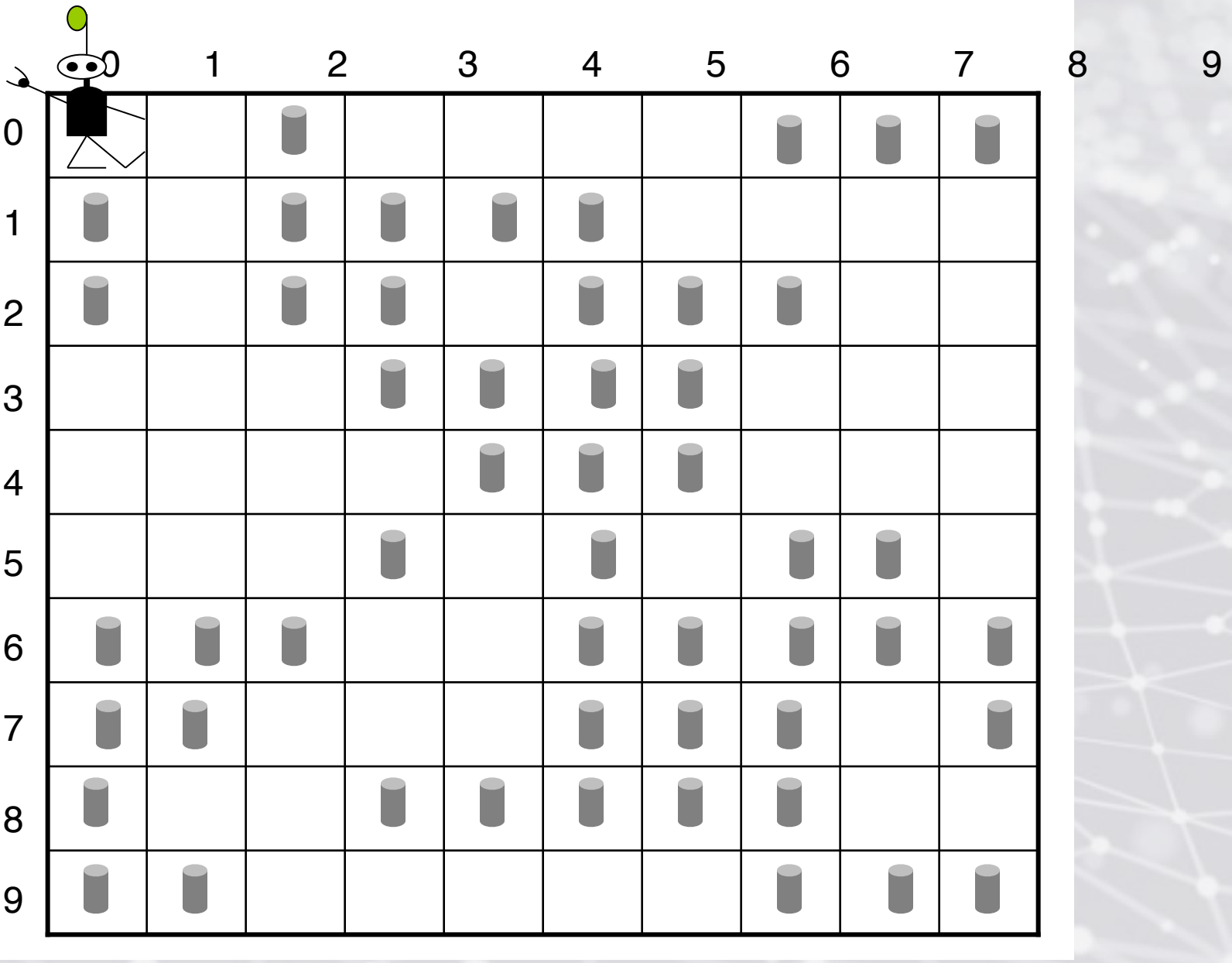
Time: 0 Score: 0



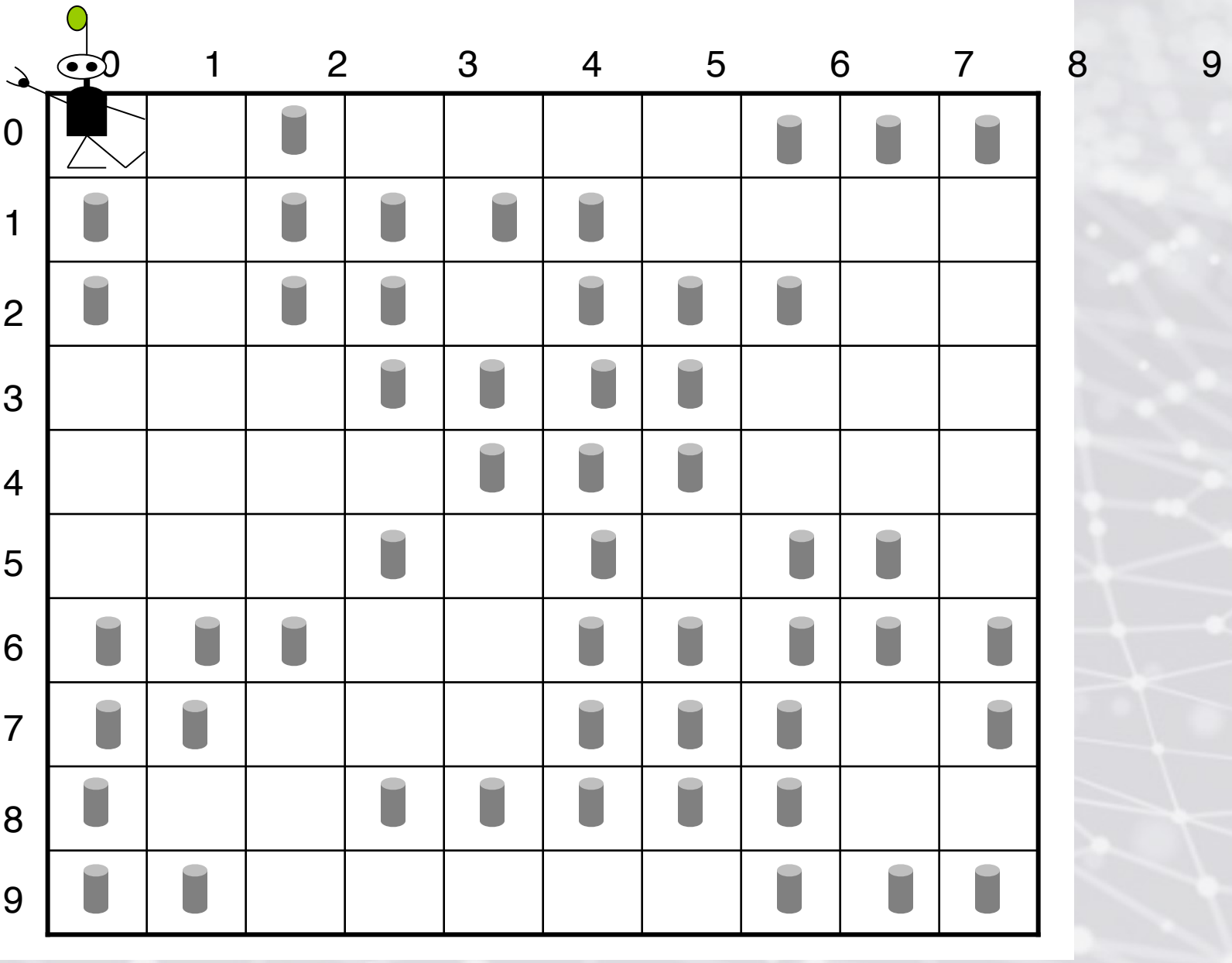
Time: 1 Score: 0



Time: 2 Score: 0



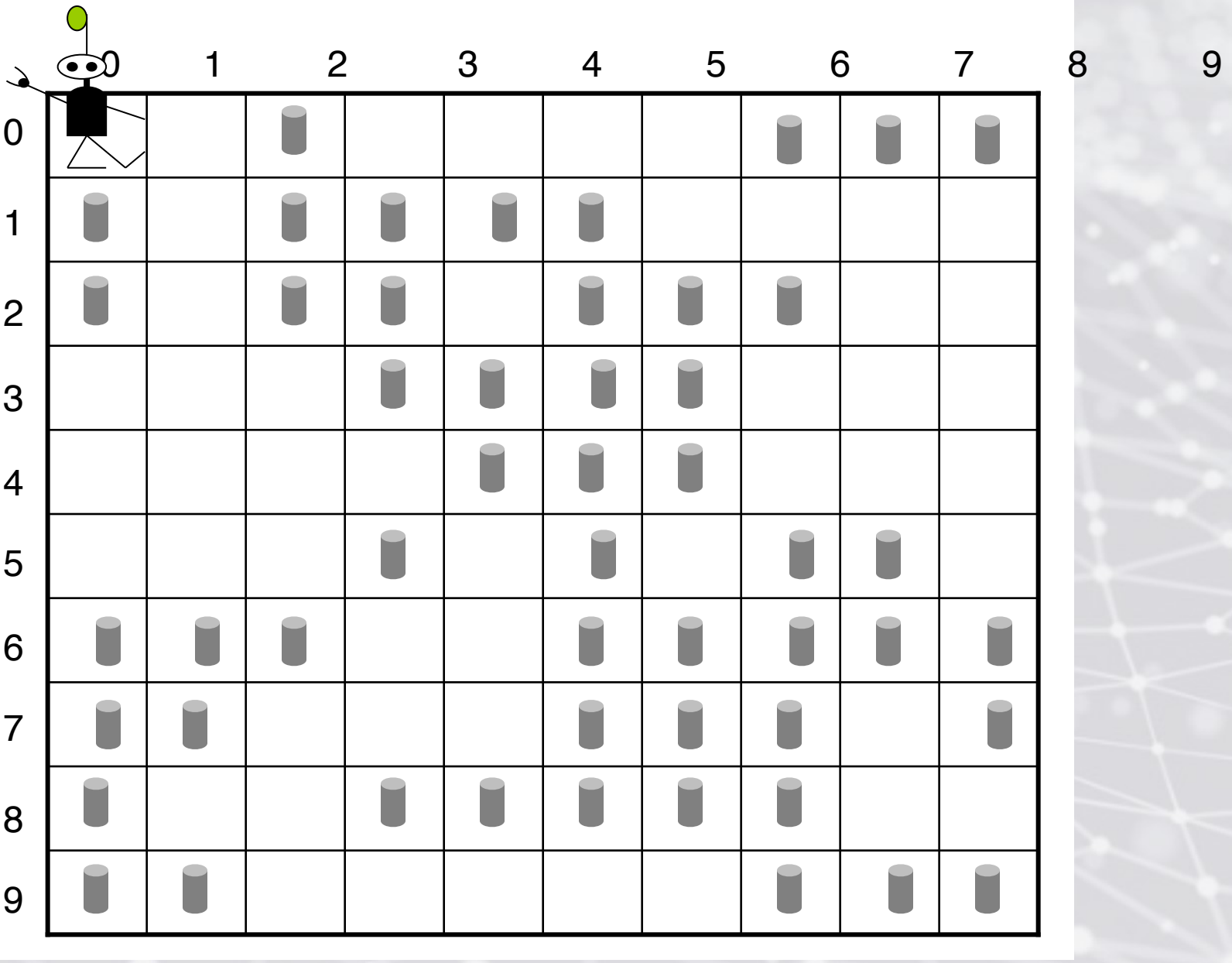
Time: 3 Score: 0



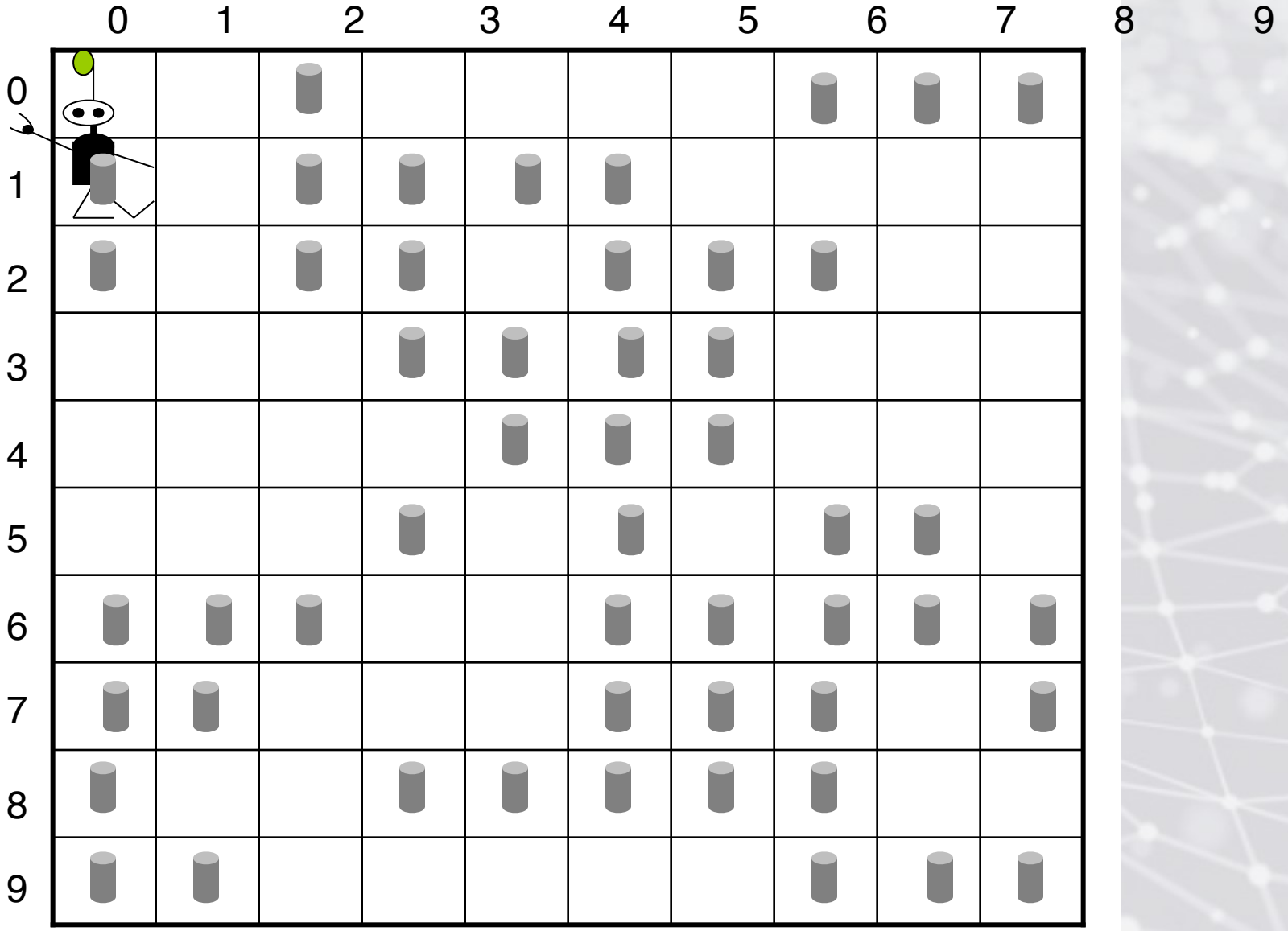
Generation 200

Fitness = 240

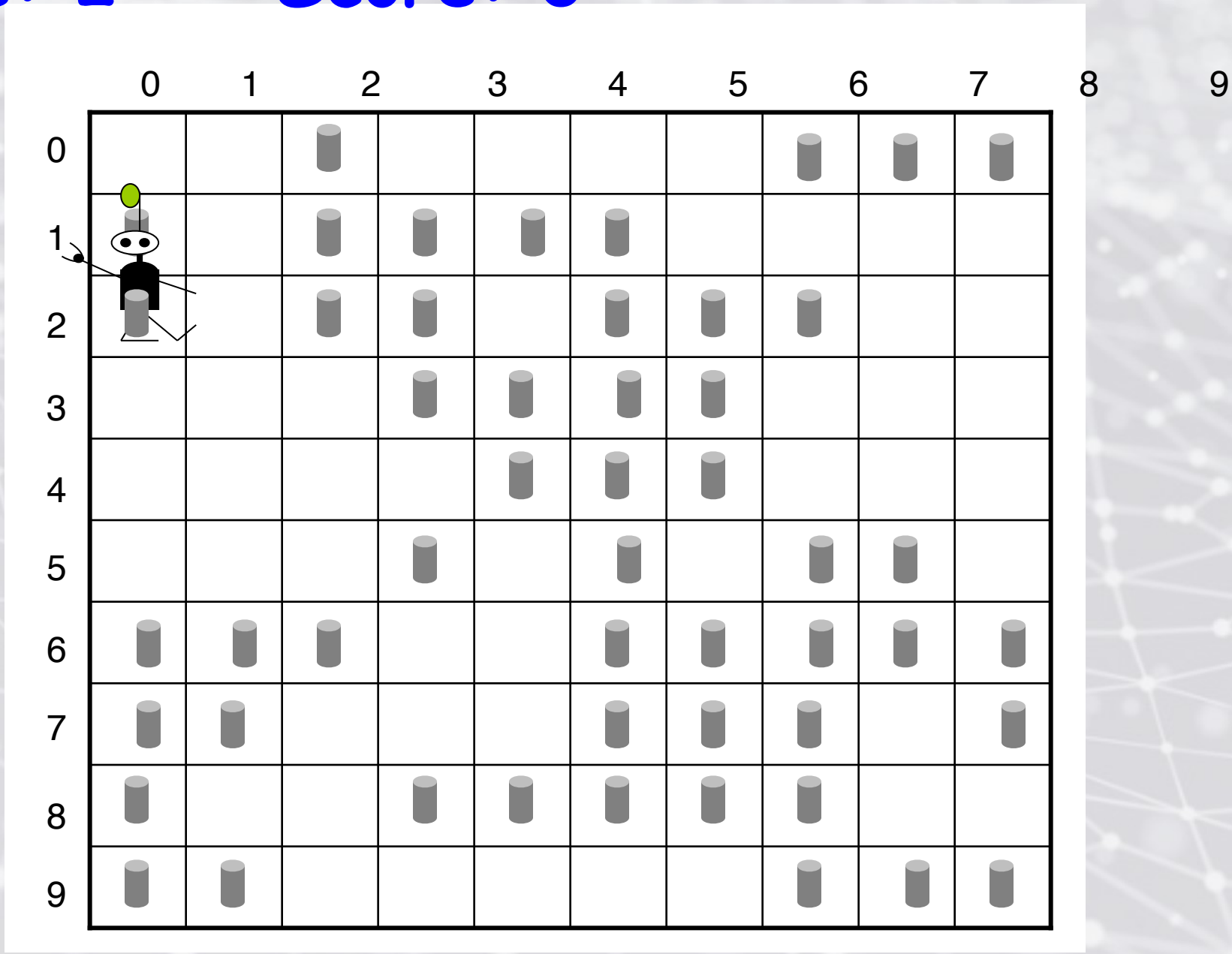
Time: 0 Score: 0



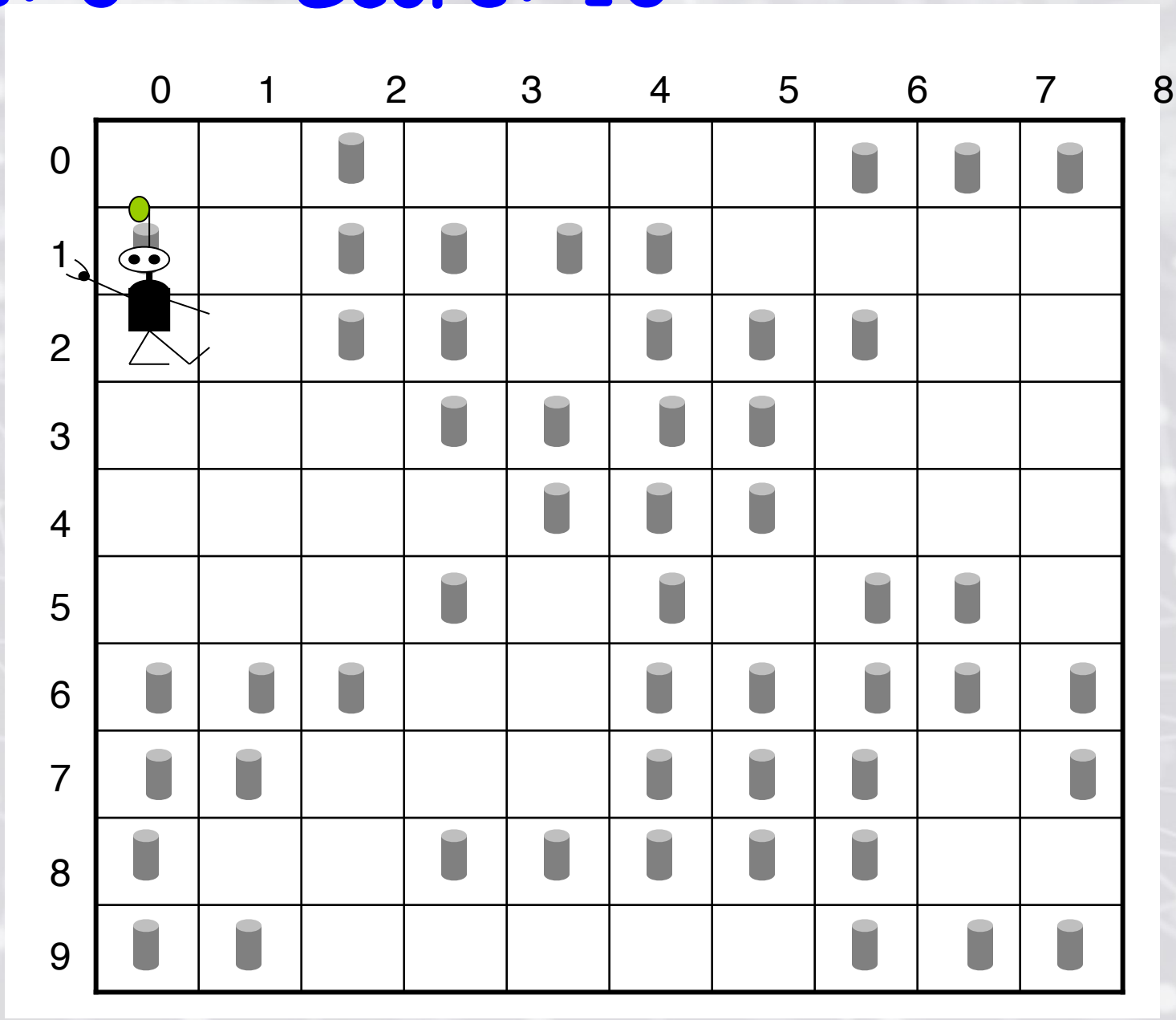
Time: 1 Score: 0



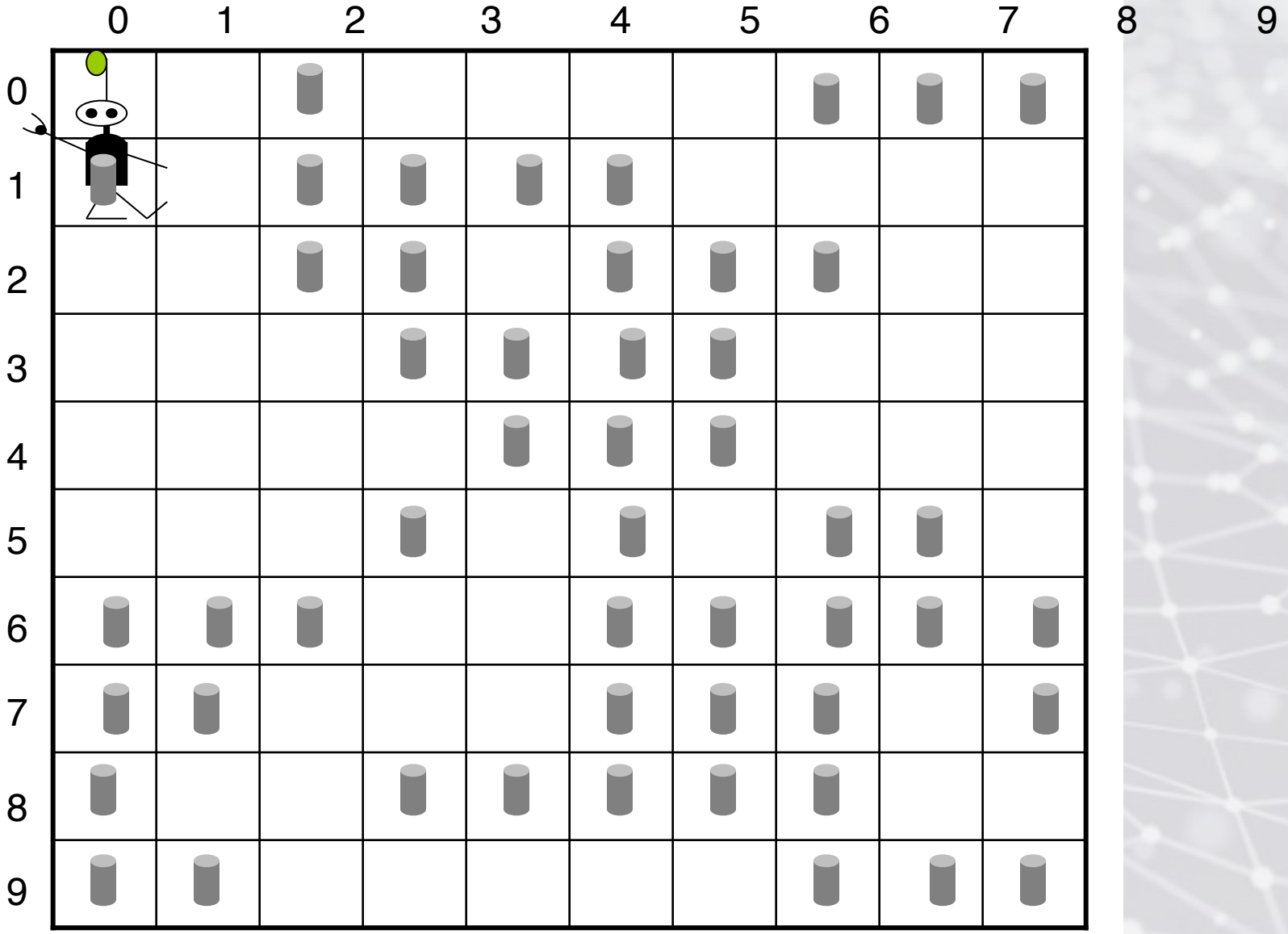
Time: 2 Score: 0



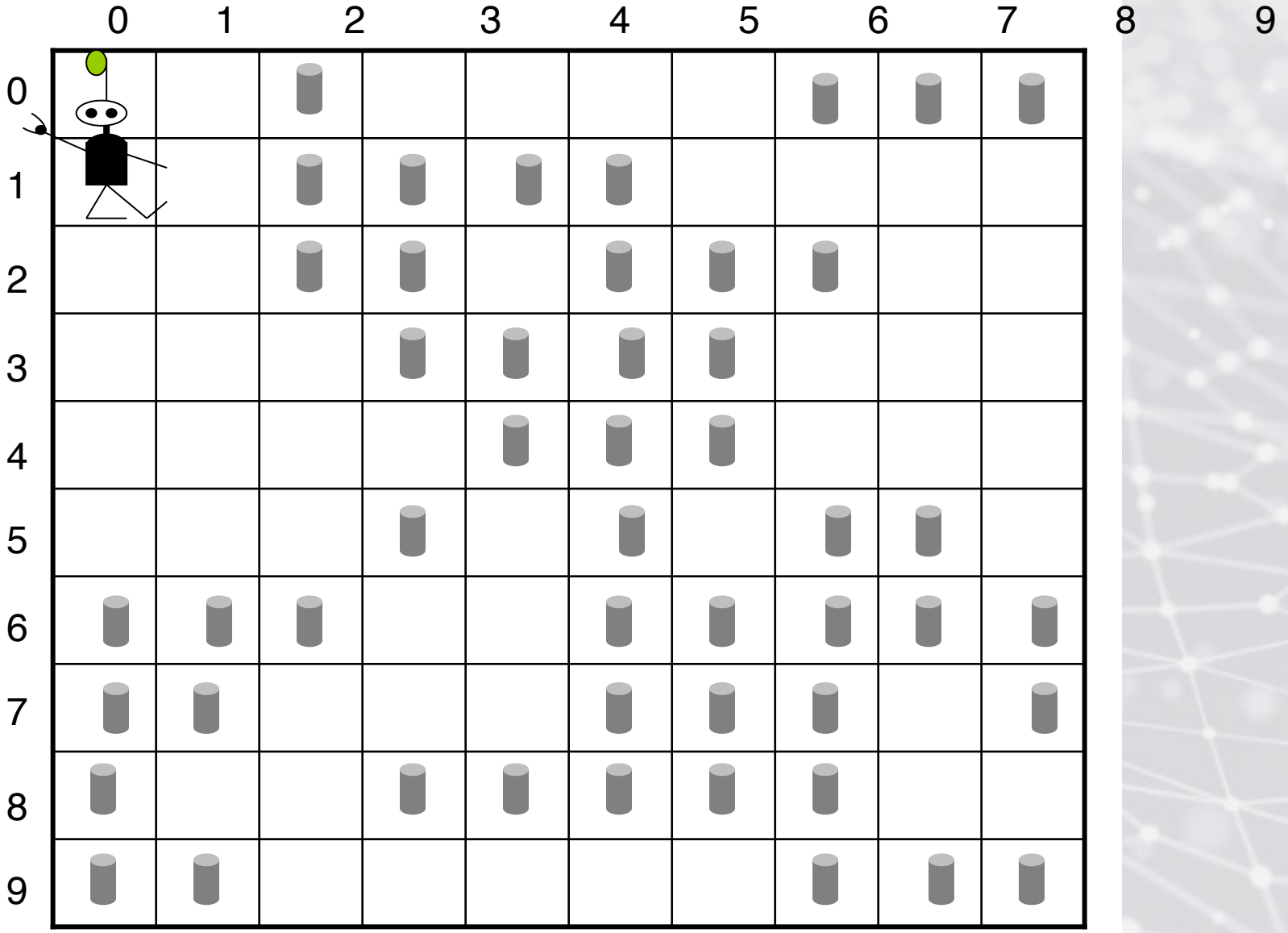
Time: 3 Score: 10



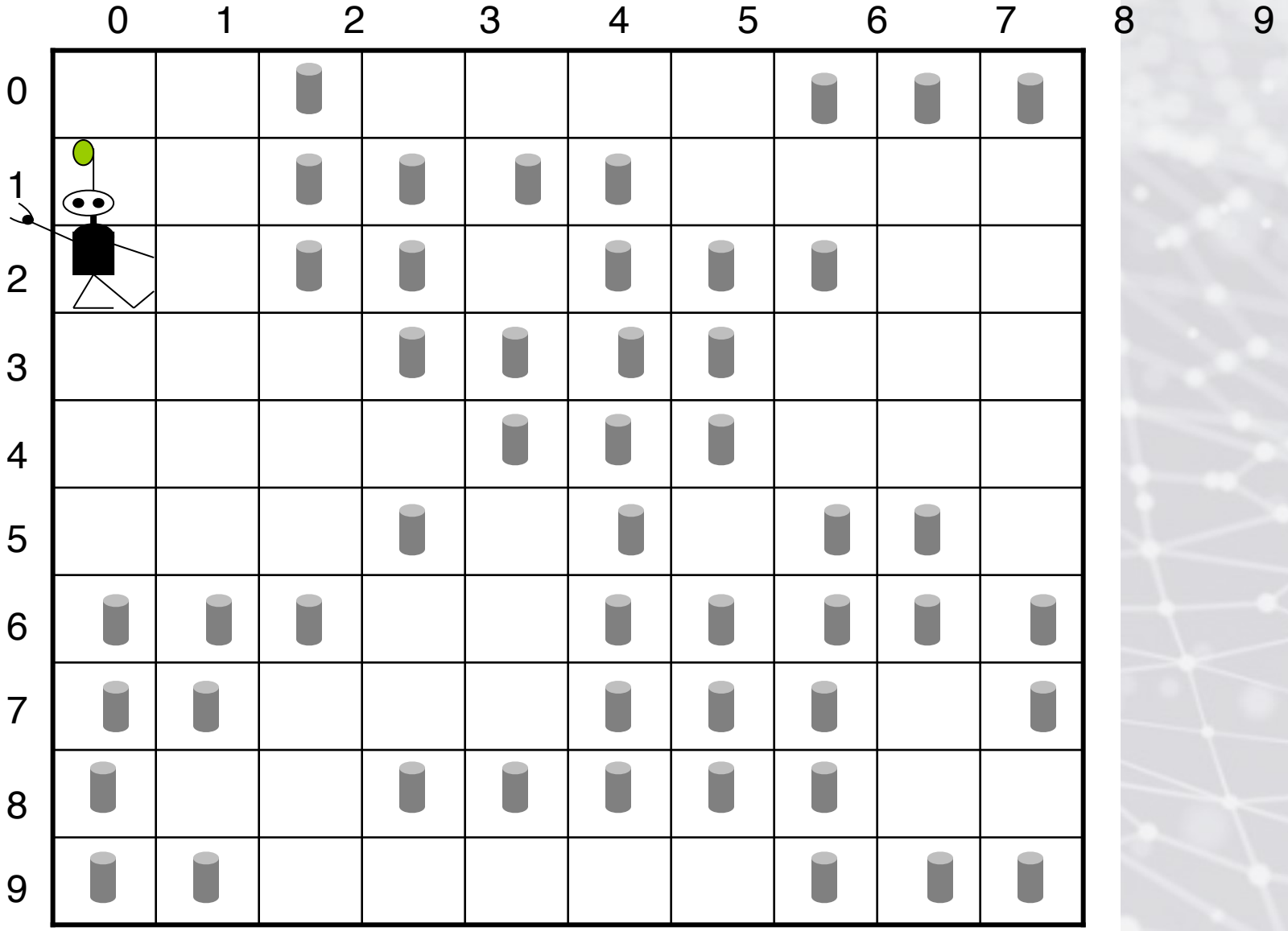
Time: 4 Score: 10



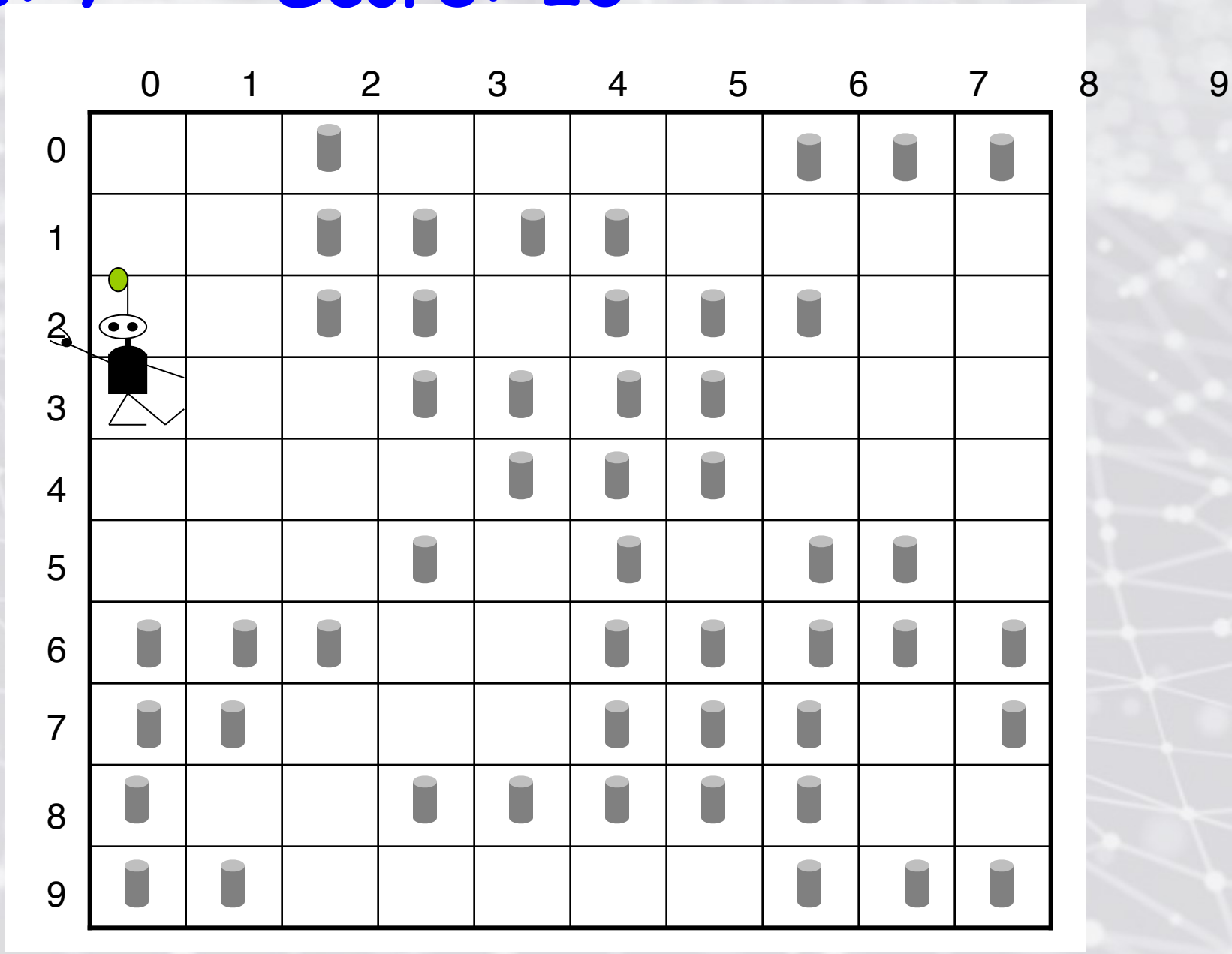
Time: 5 Score: 20



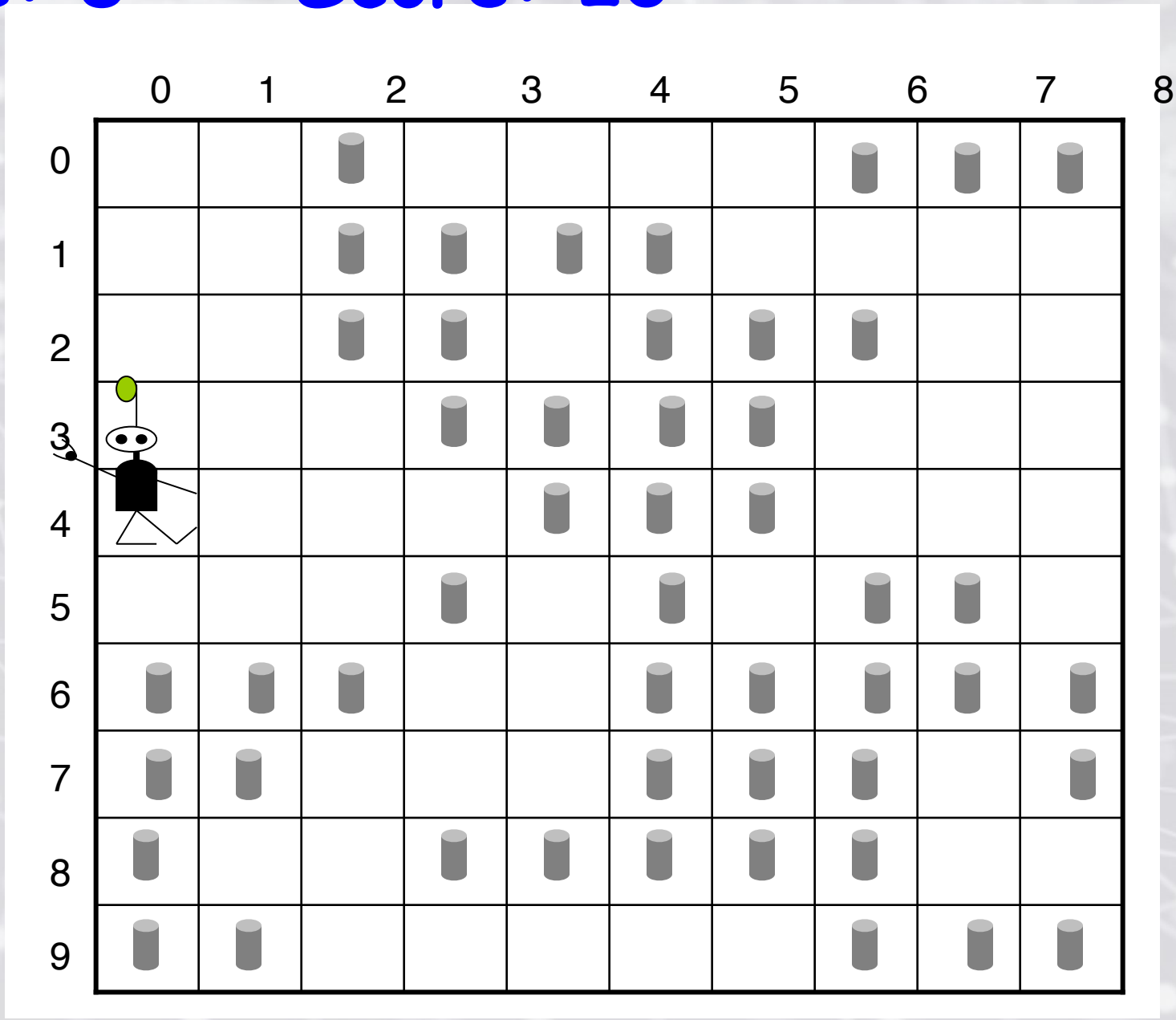
Time: 6 Score: 20



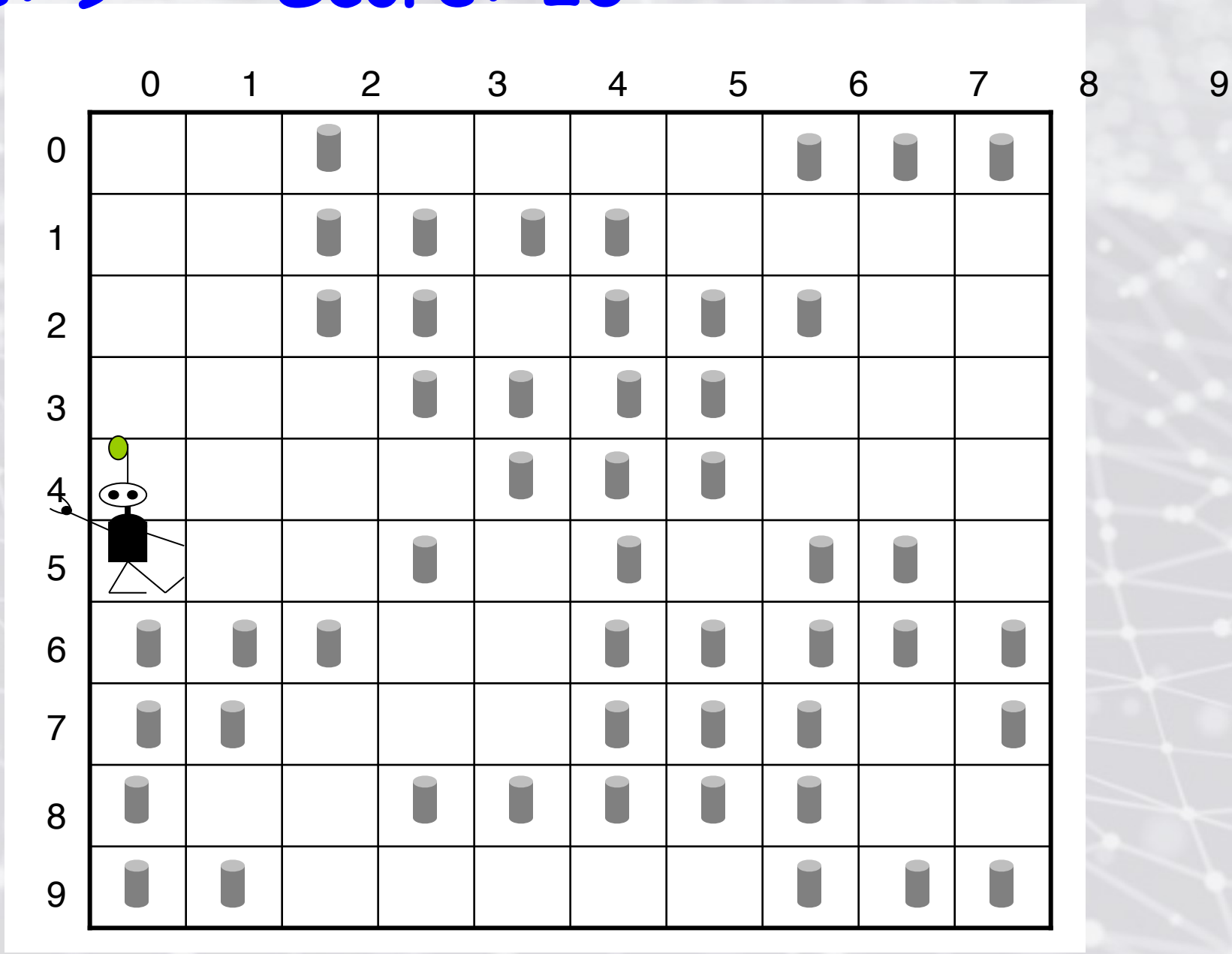
Time: 7 Score: 20



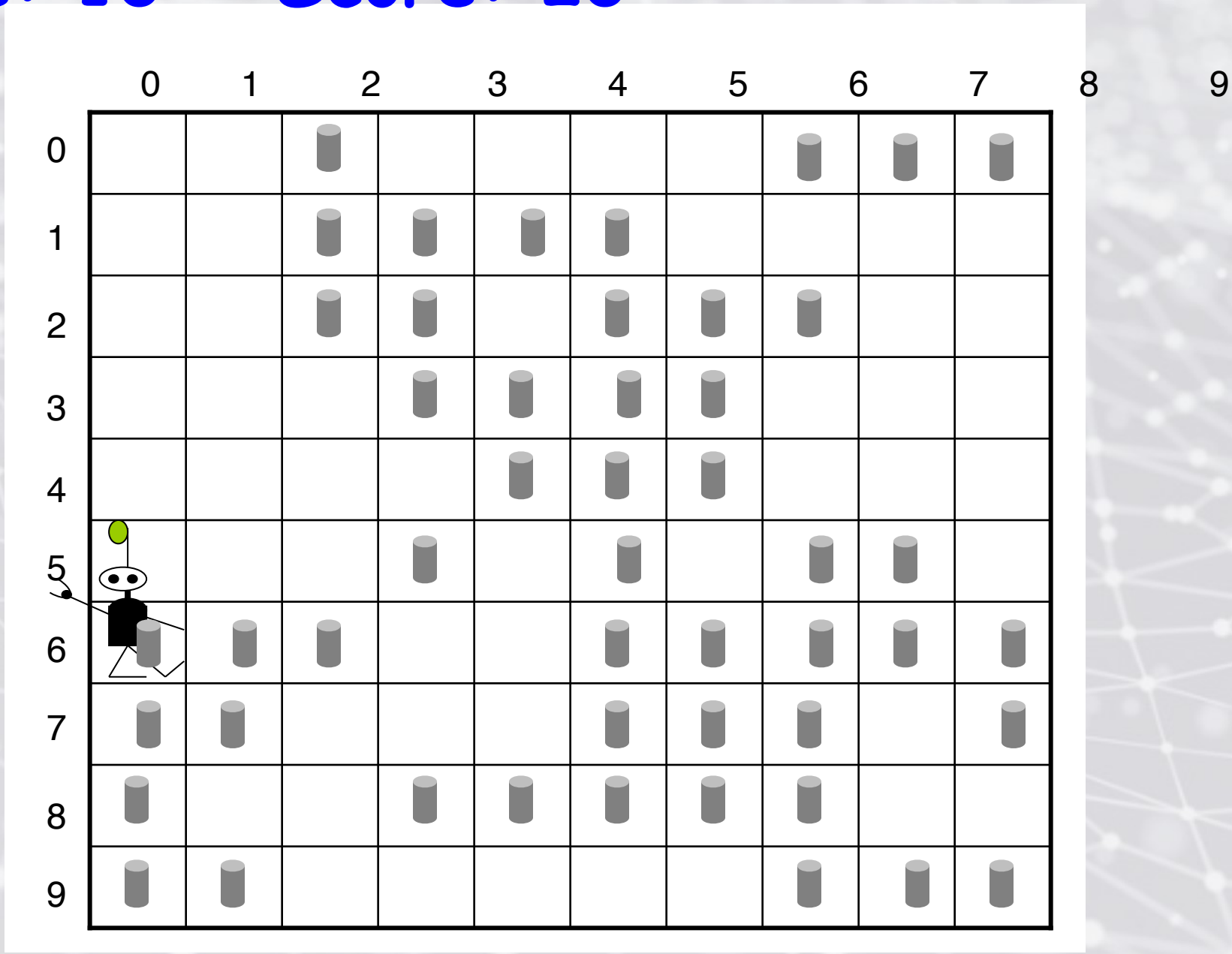
Time: 8 Score: 20



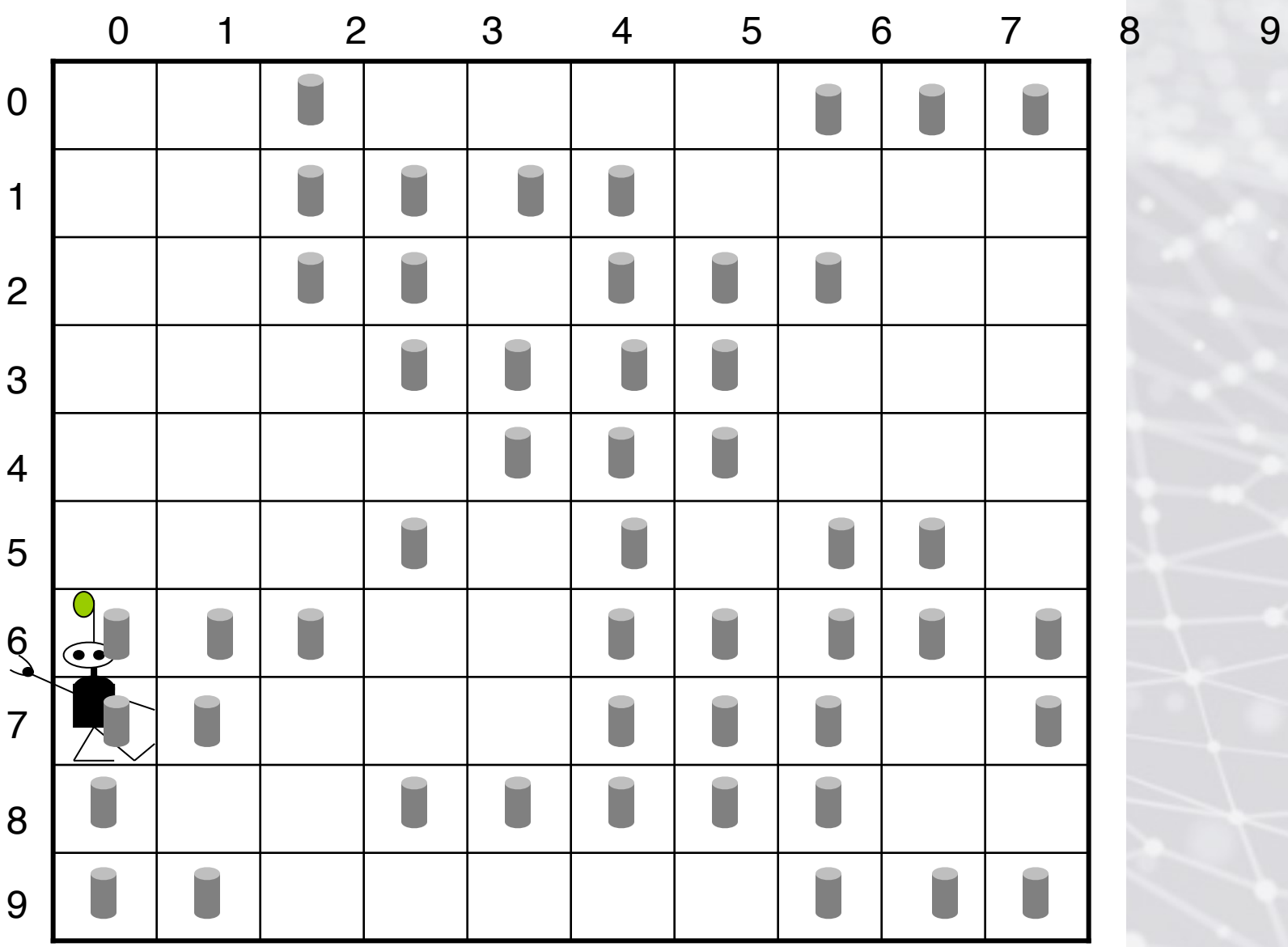
Time: 9 Score: 20



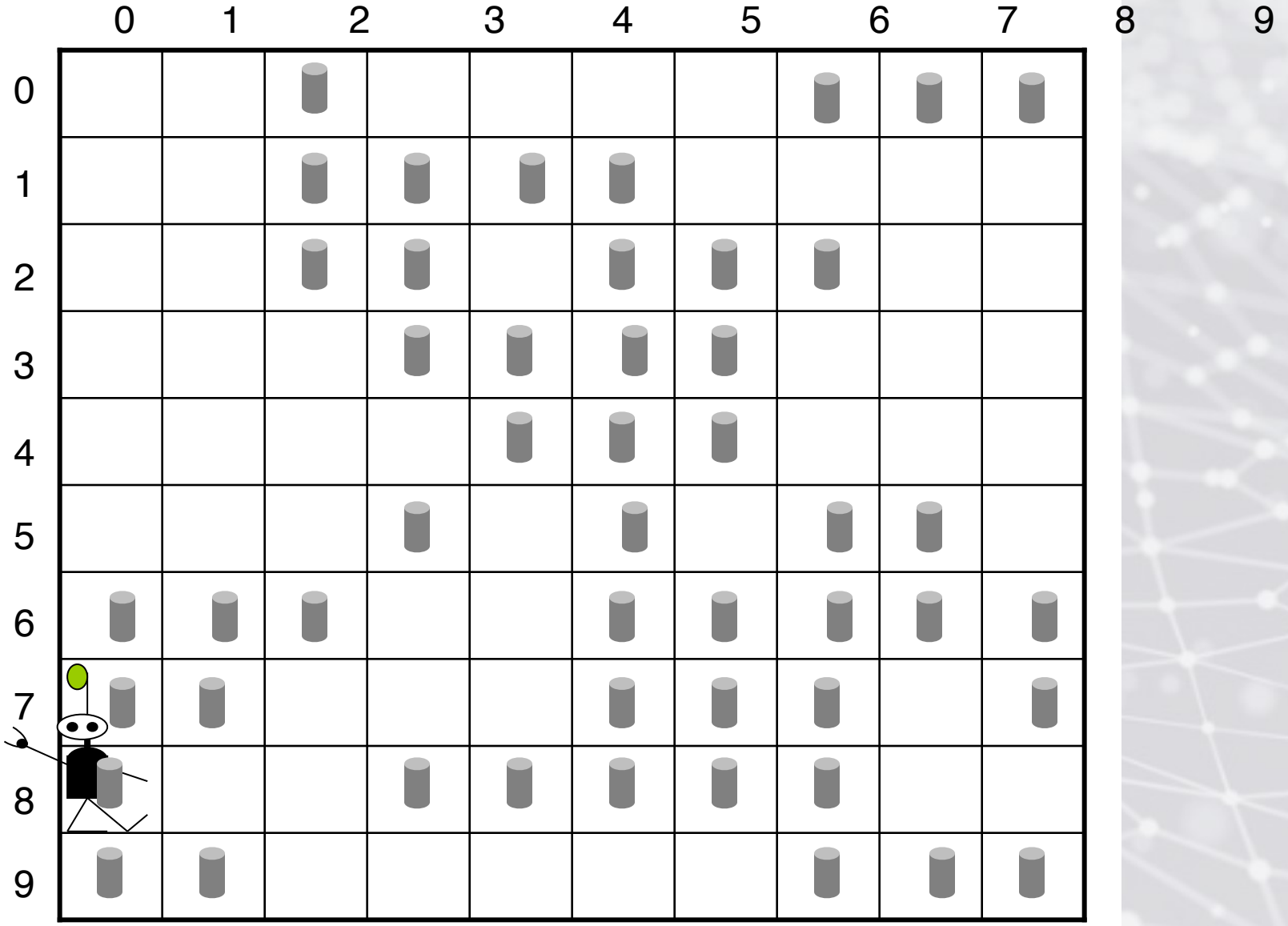
Time: 10 Score: 20



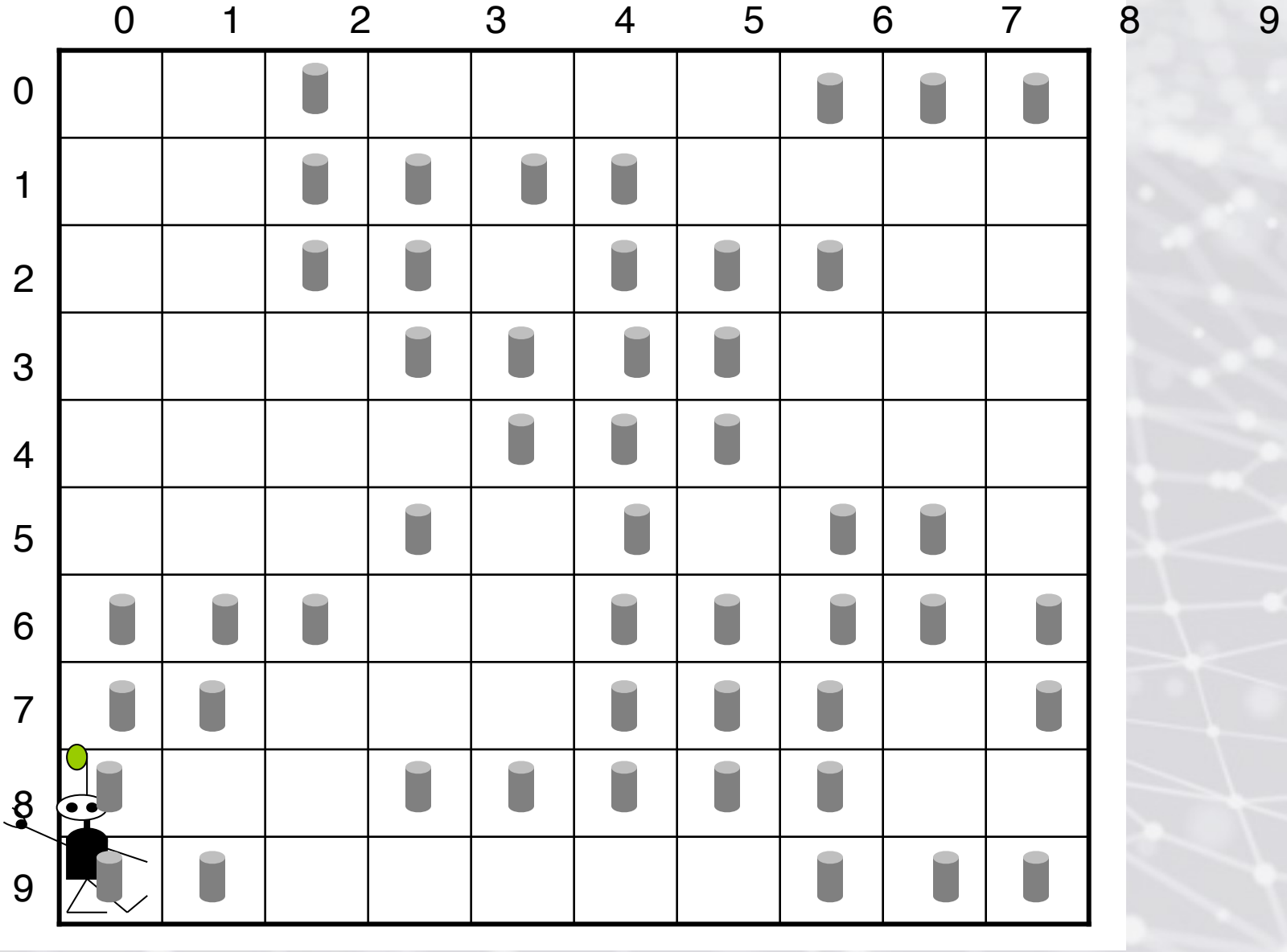
Time: 11 Score: 20



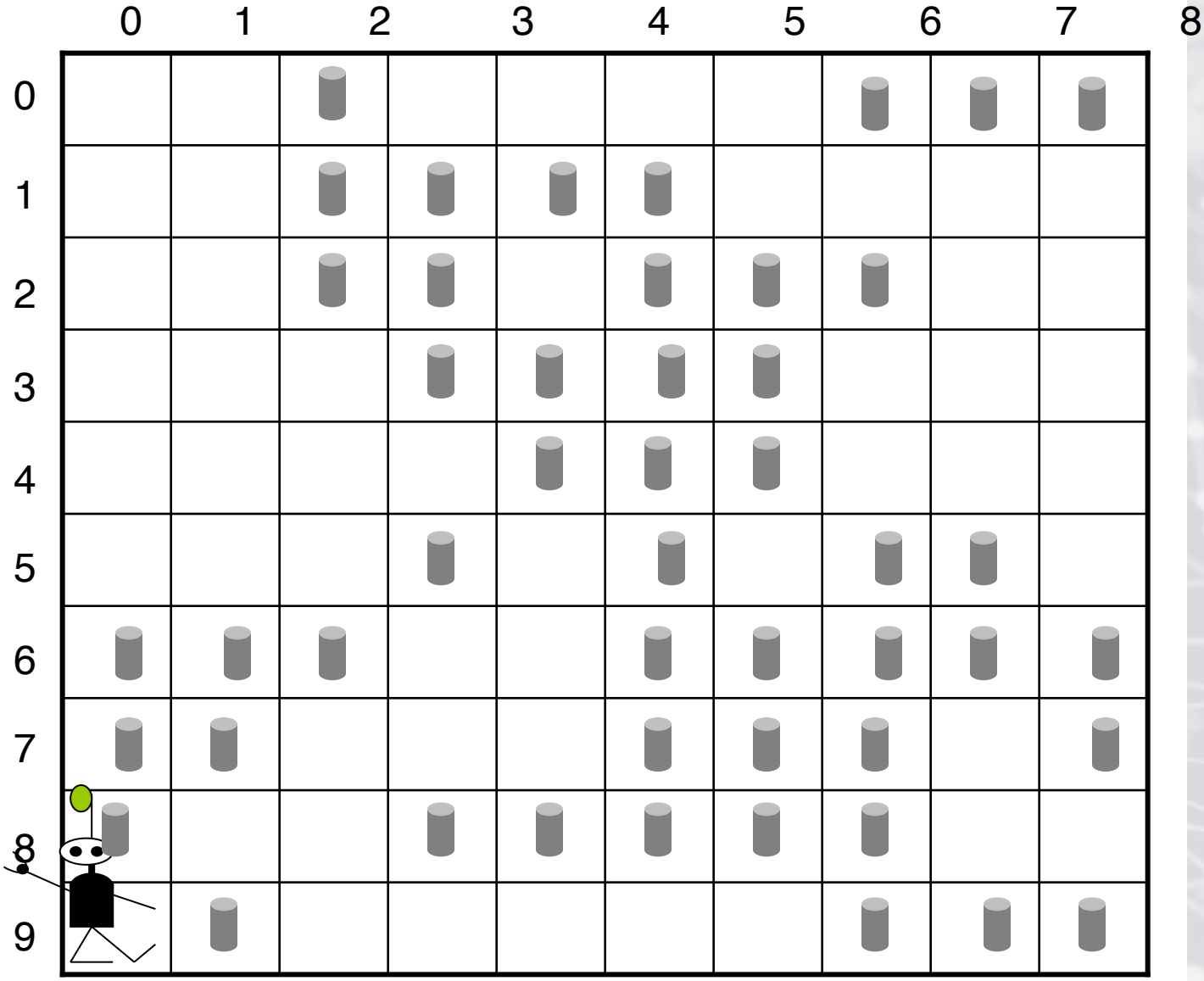
Time: 12 Score: 20



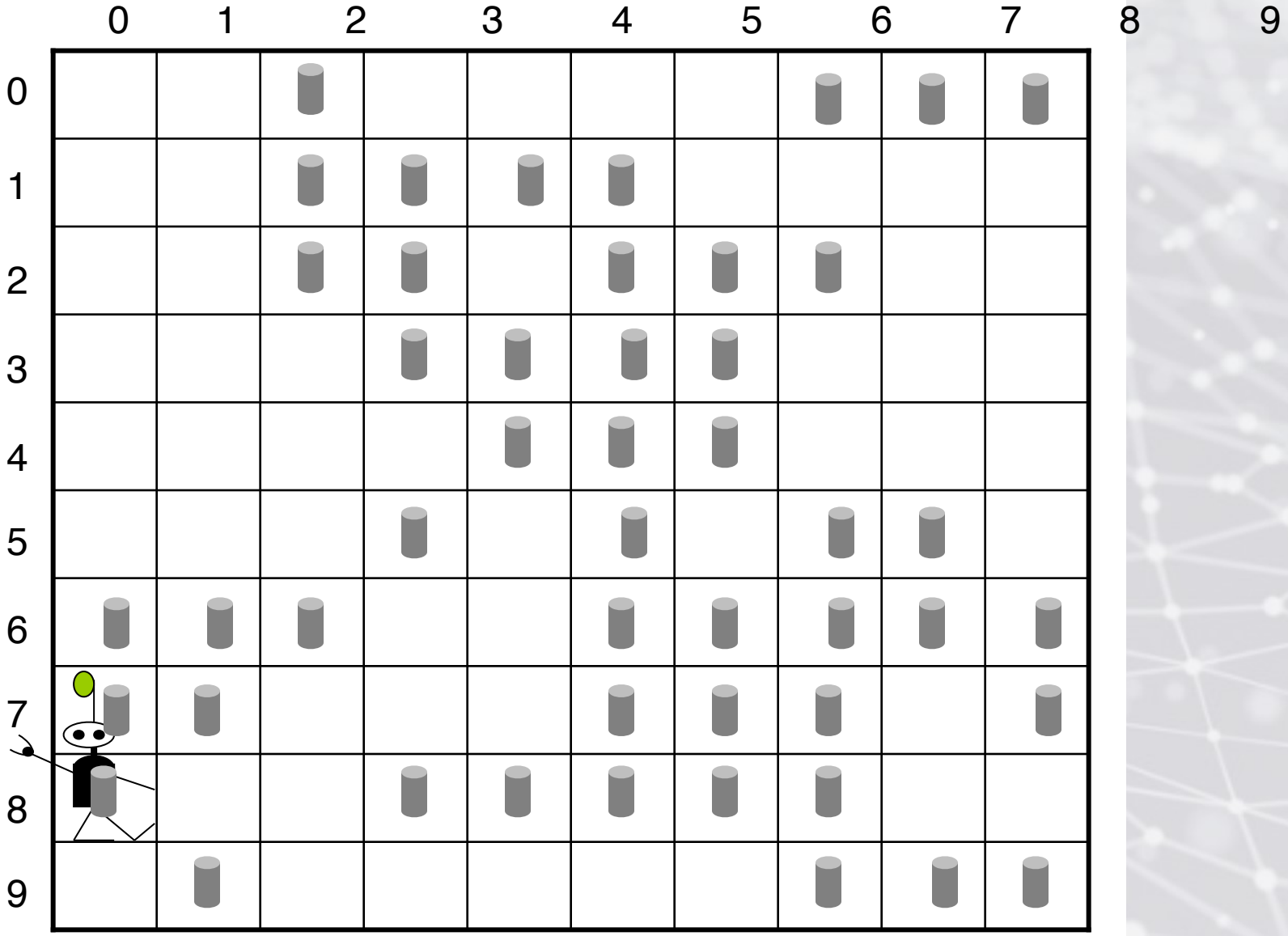
Time: 13 Score: 20



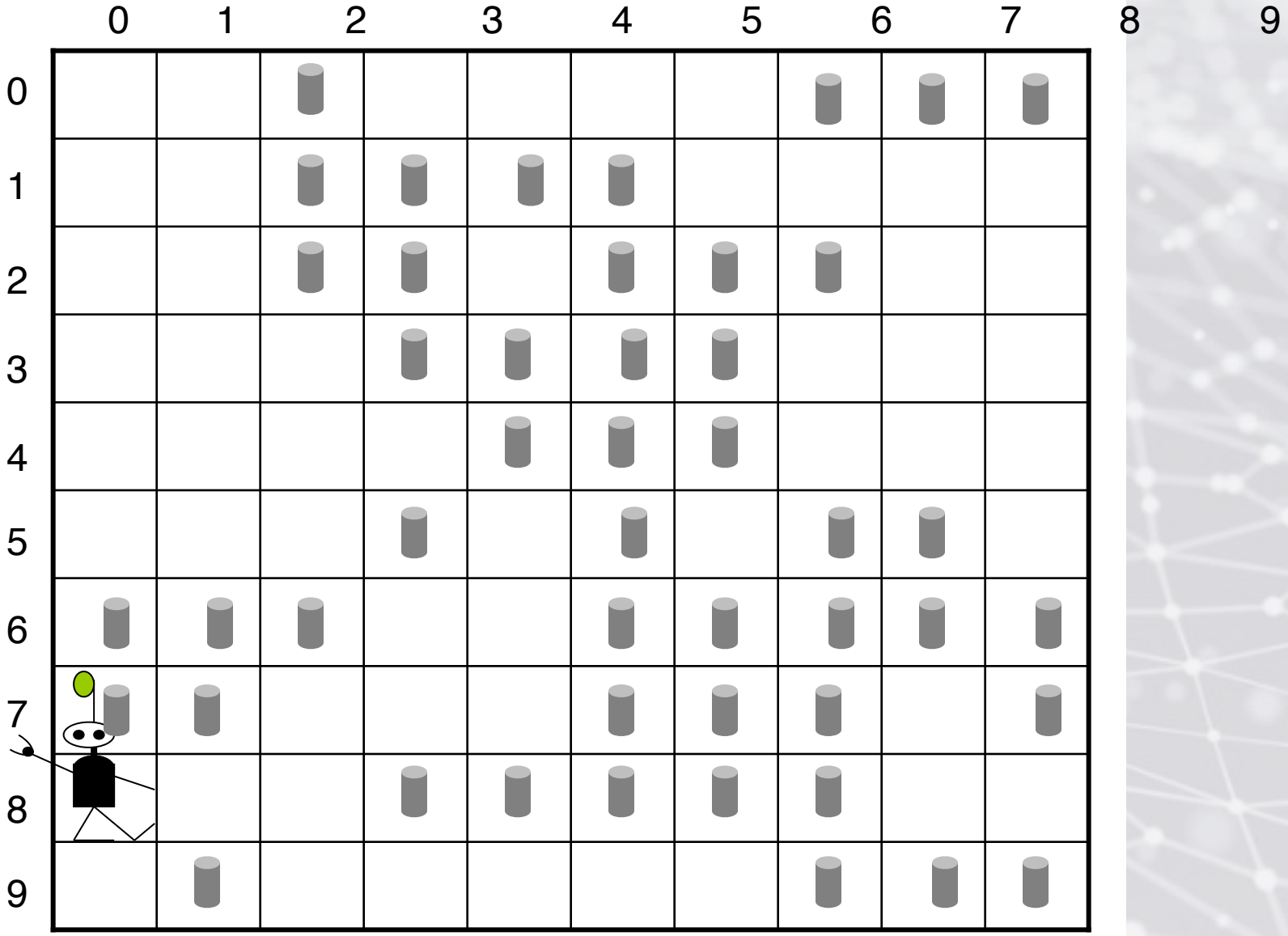
Time: 14 Score: 30



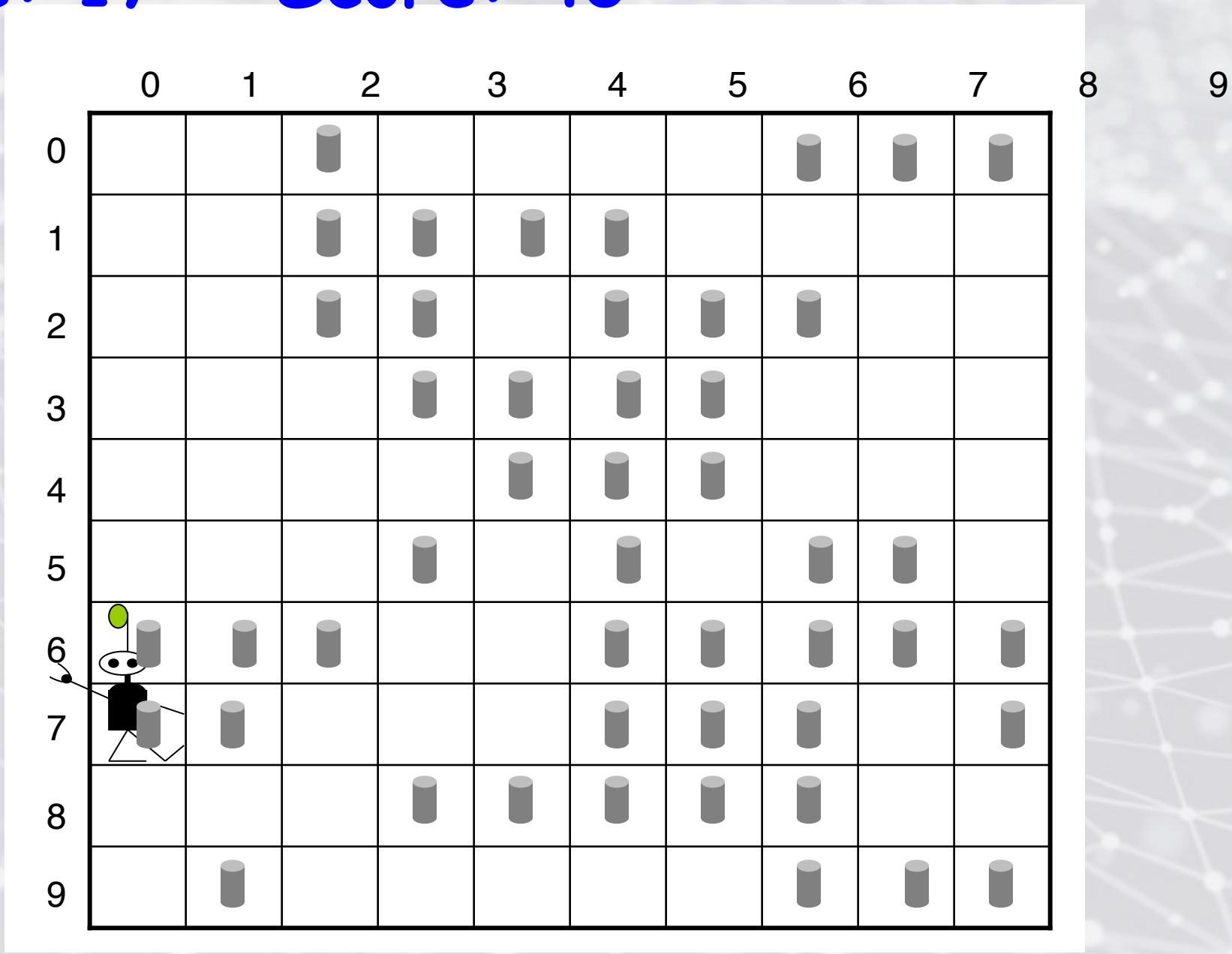
Time: 15 Score: 30



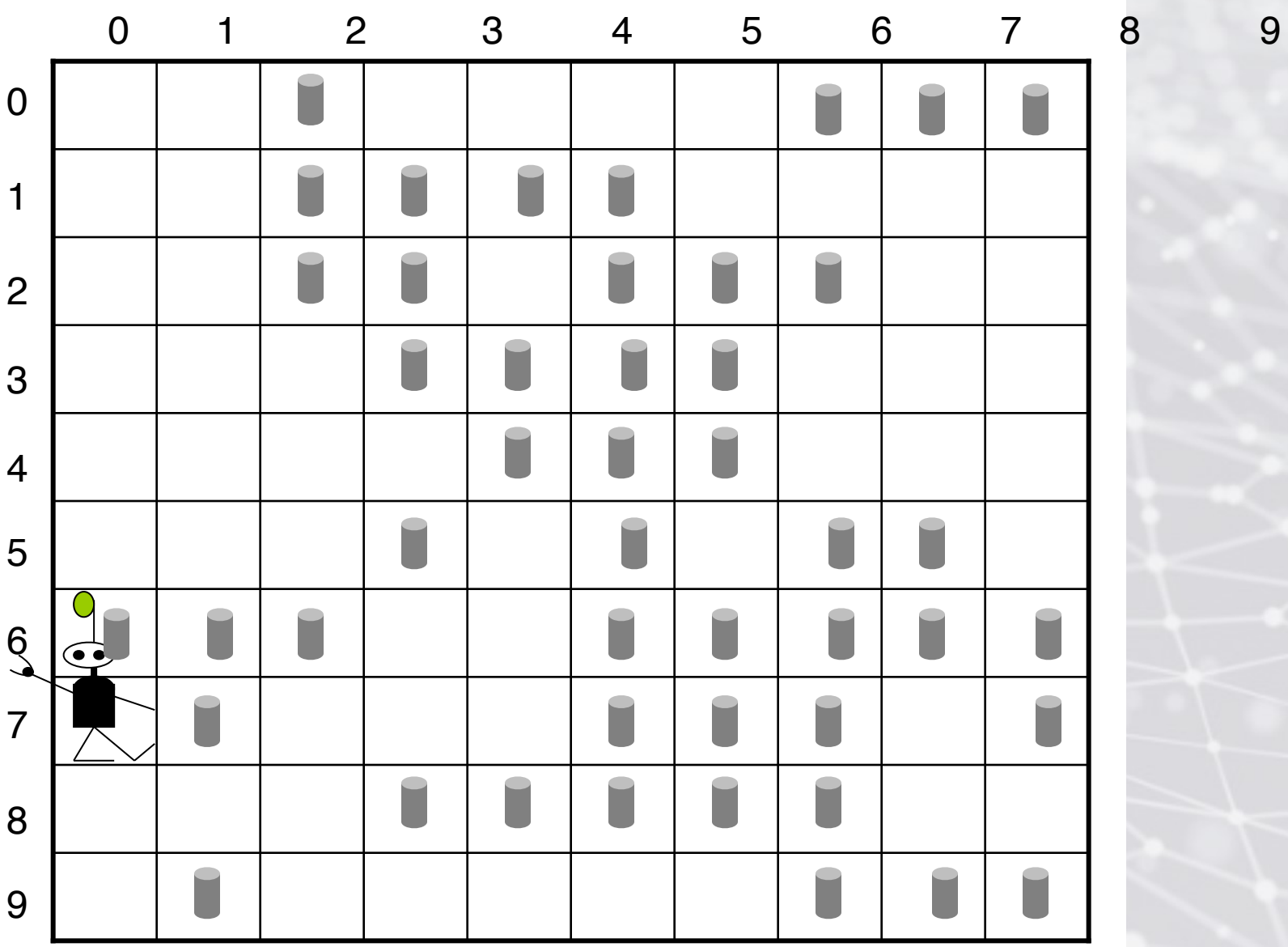
Time: 16 Score: 40



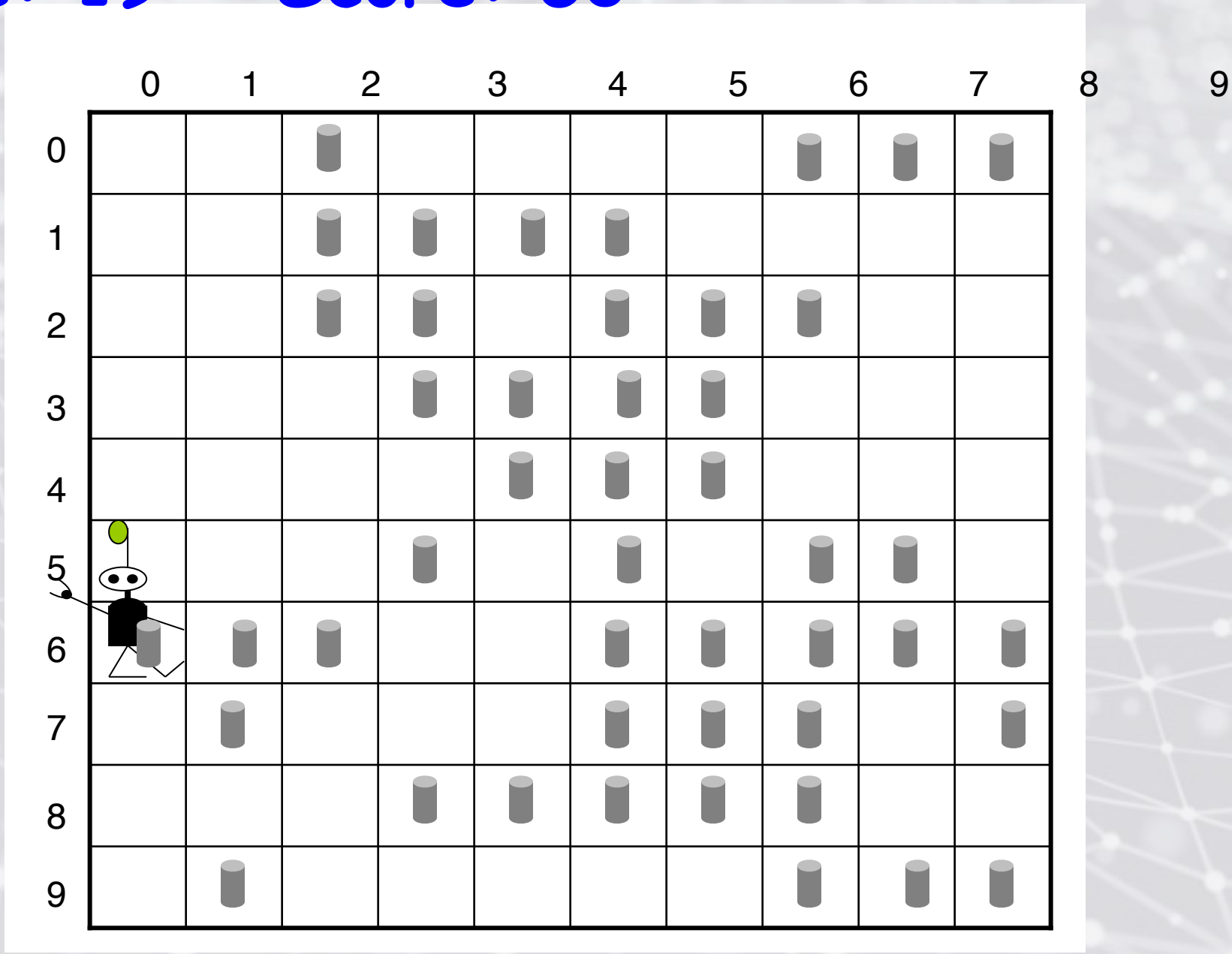
Time: 17 Score: 40



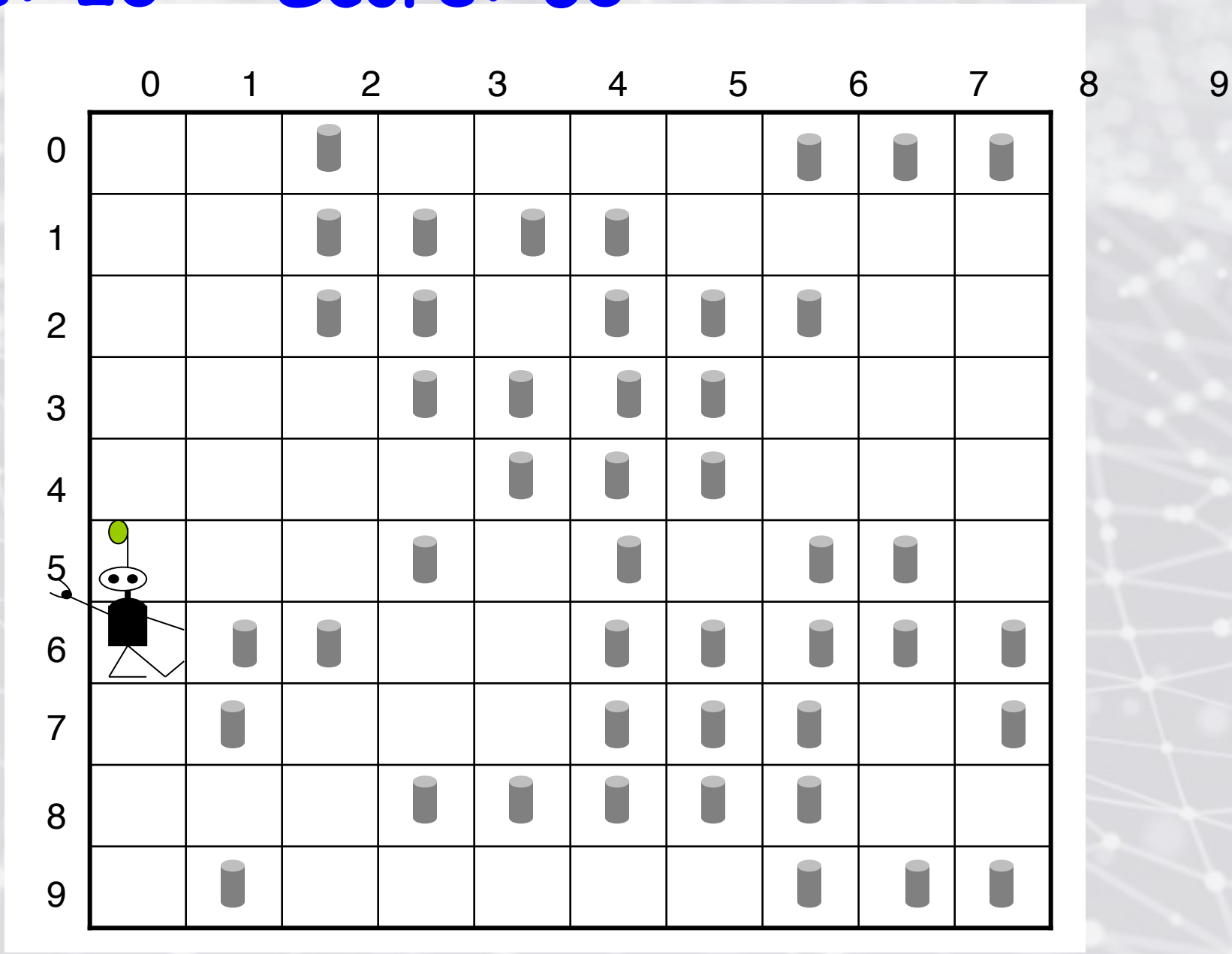
Time: 18 Score: 50



Time: 19 Score: 50



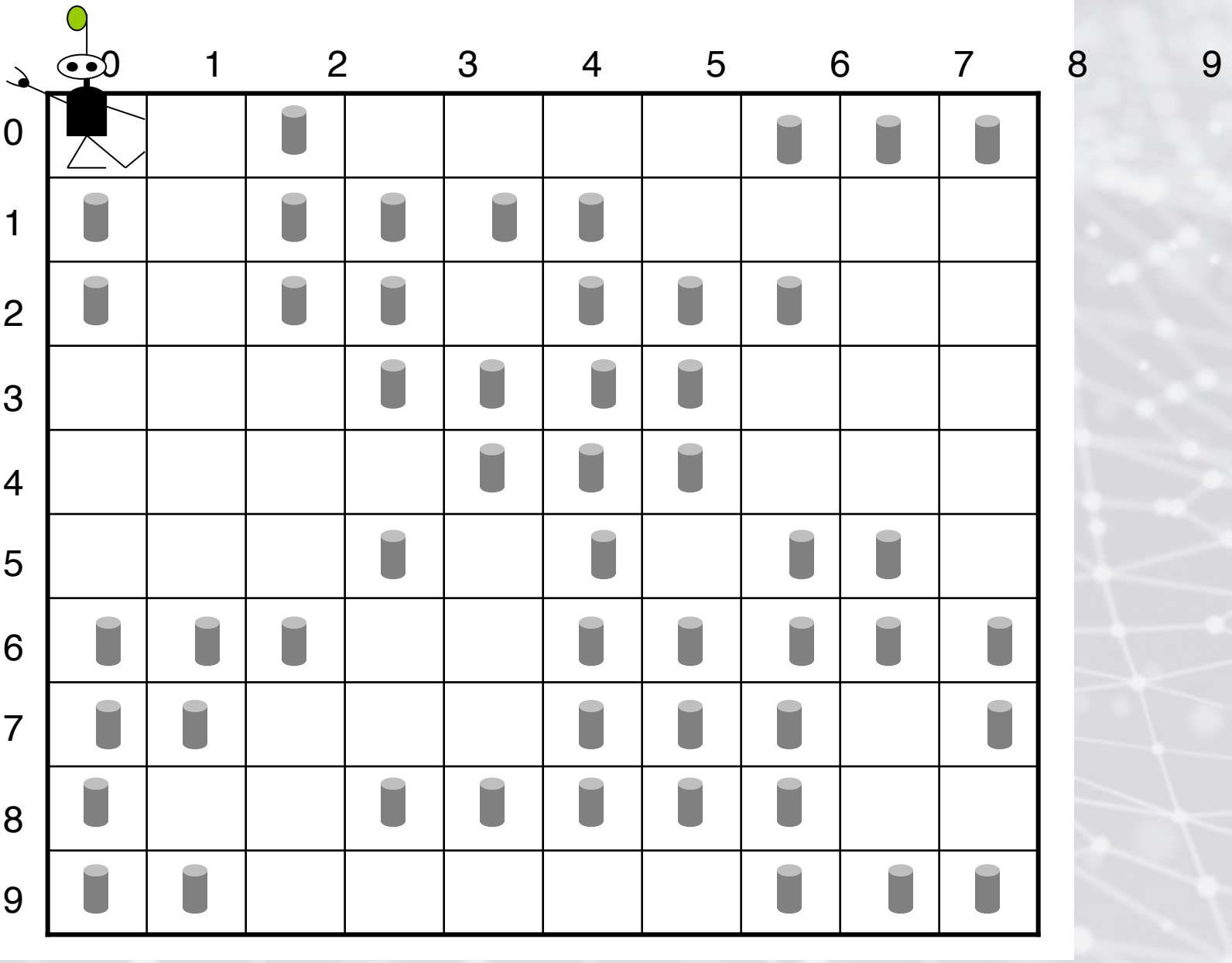
Time: 20 Score: 60



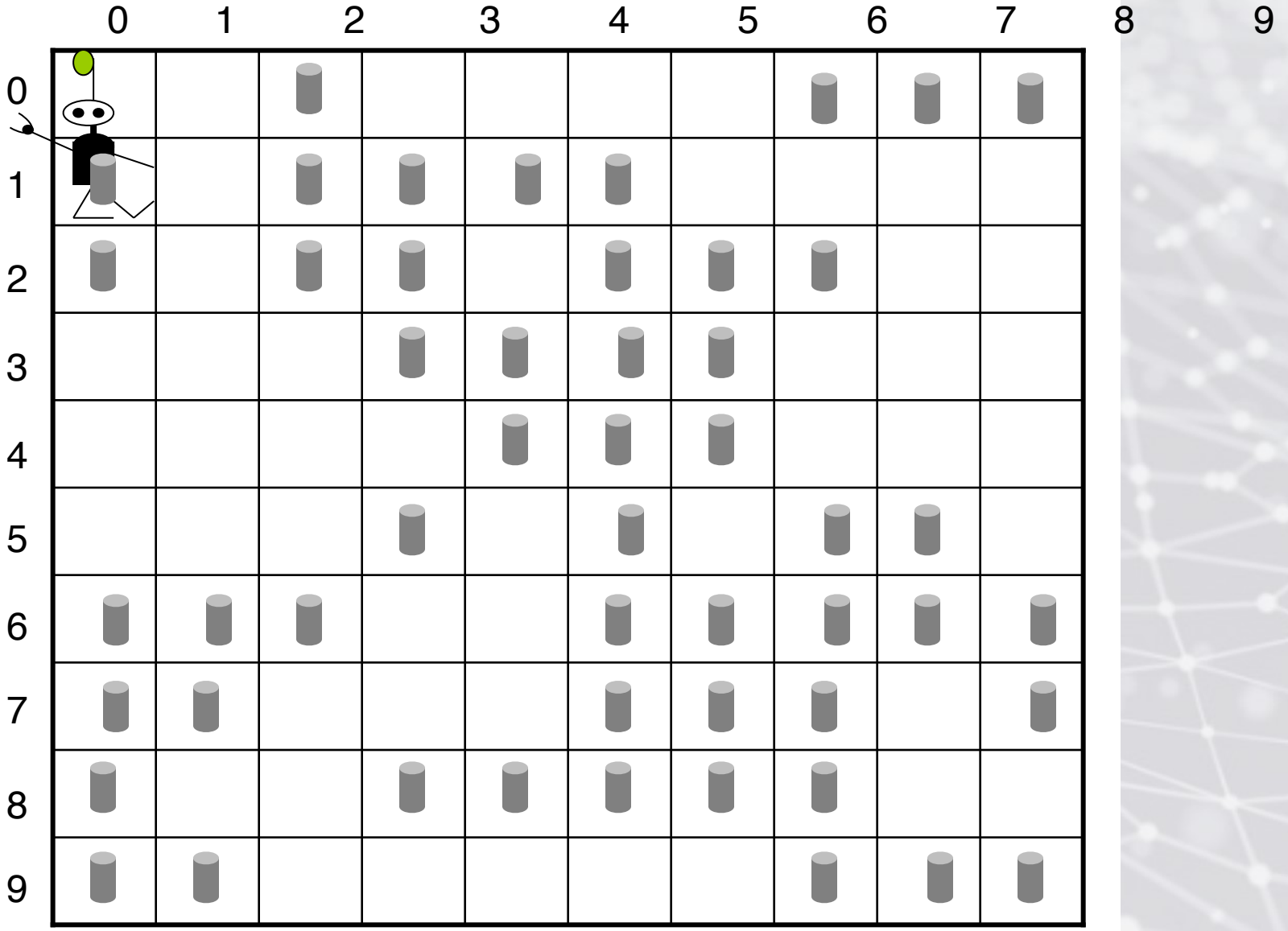
Generation 1000

Fitness = 492

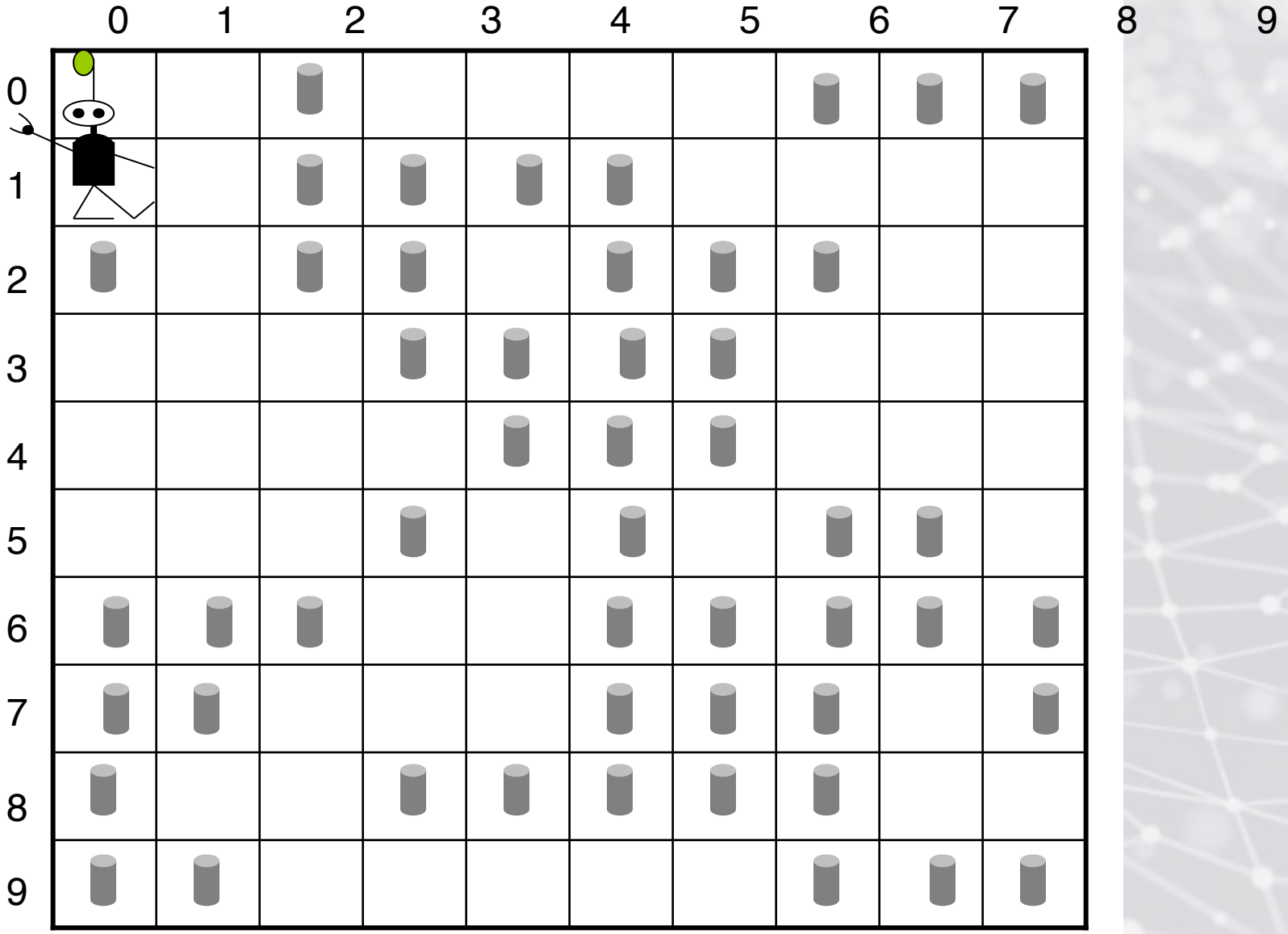
Time: 0 Score: 0



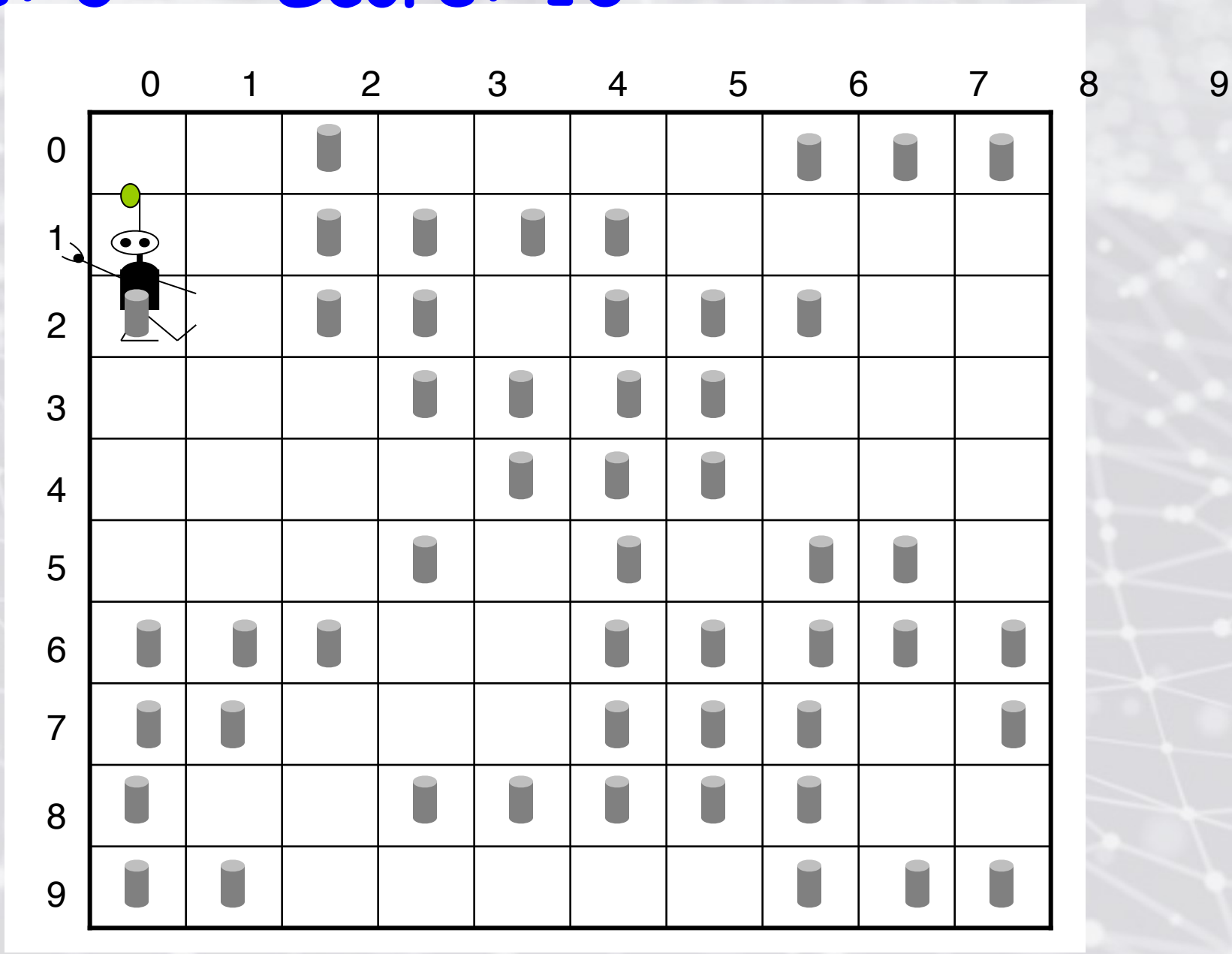
Time: 1 Score: 0



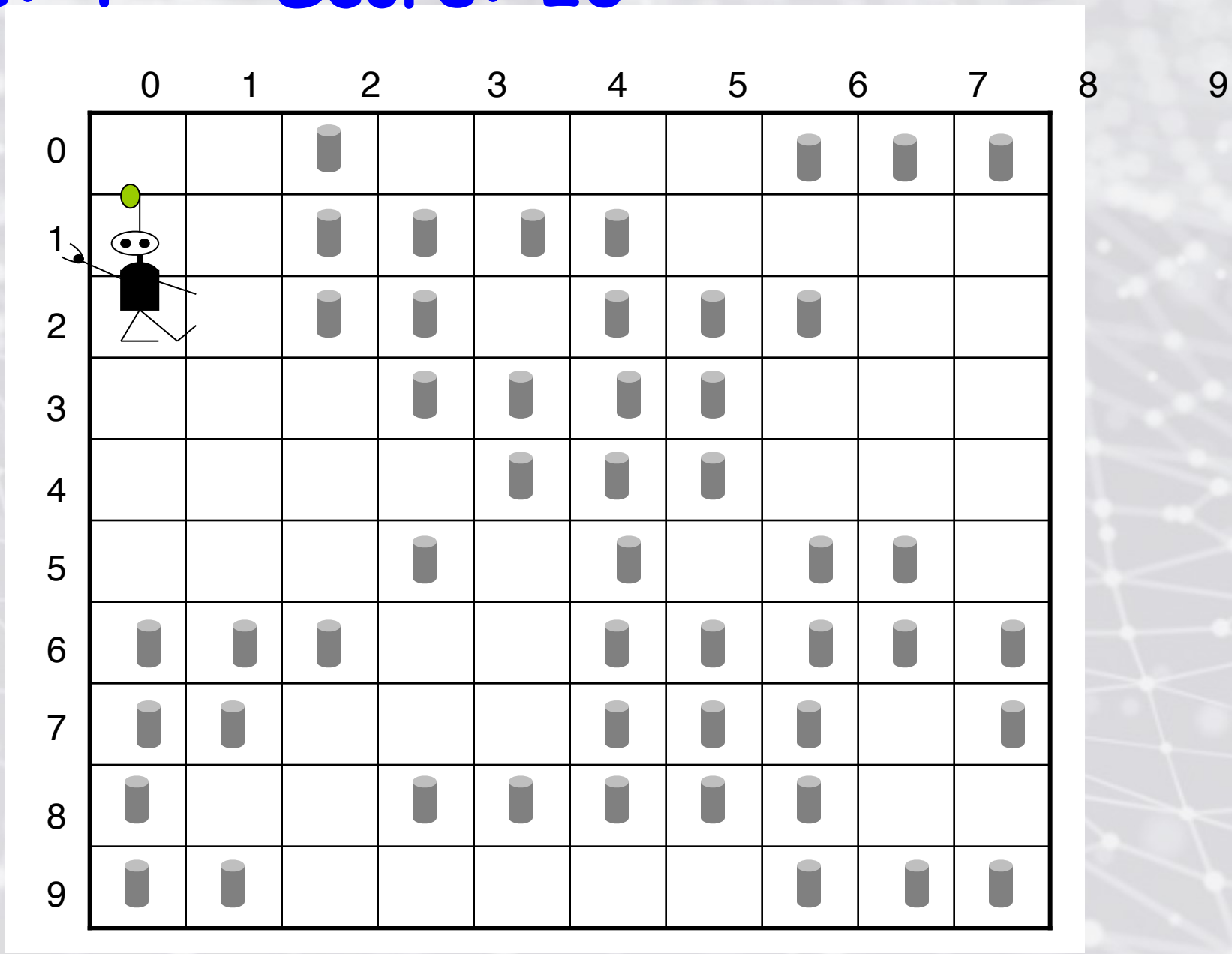
Time: 2 Score: 10



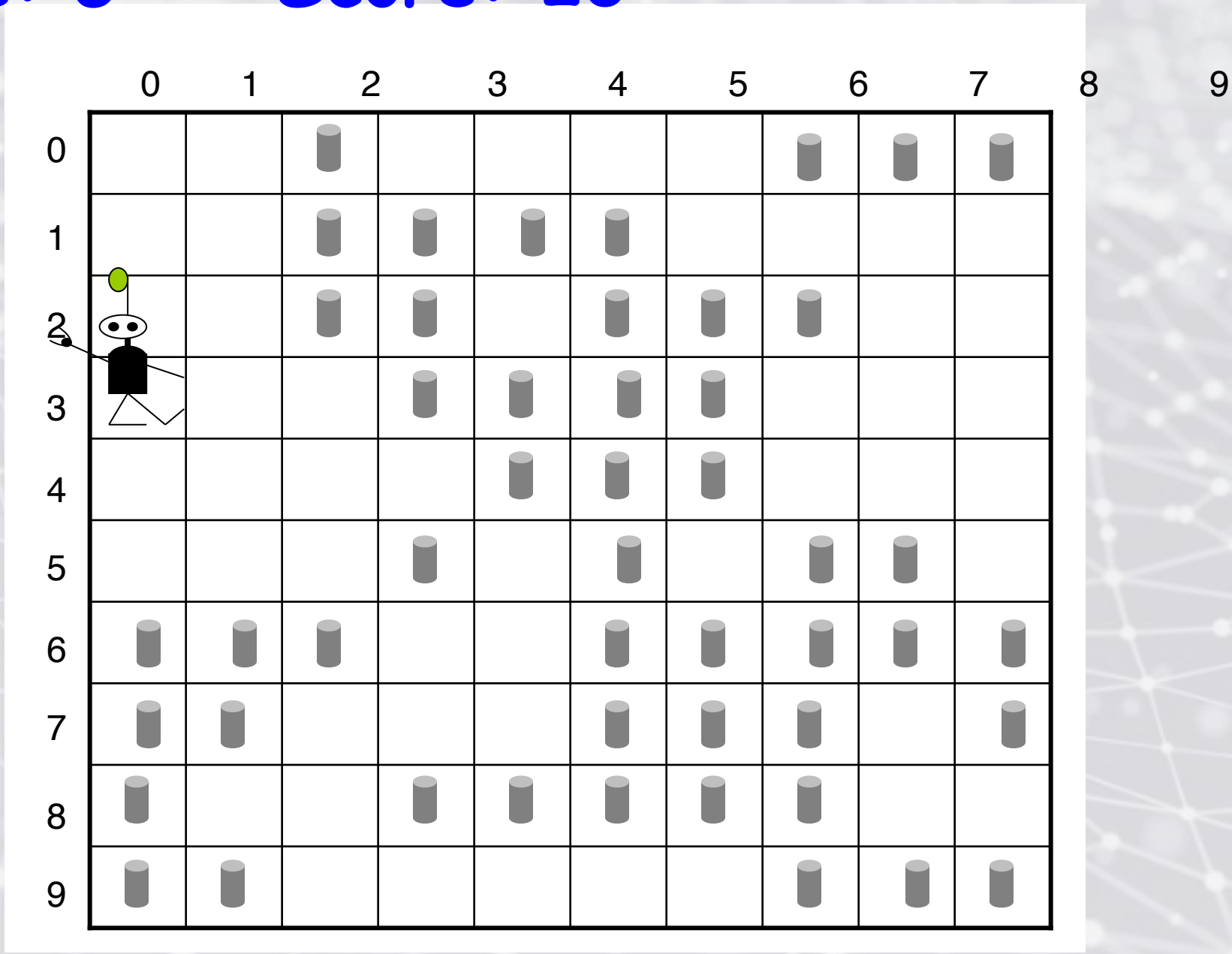
Time: 3 Score: 10



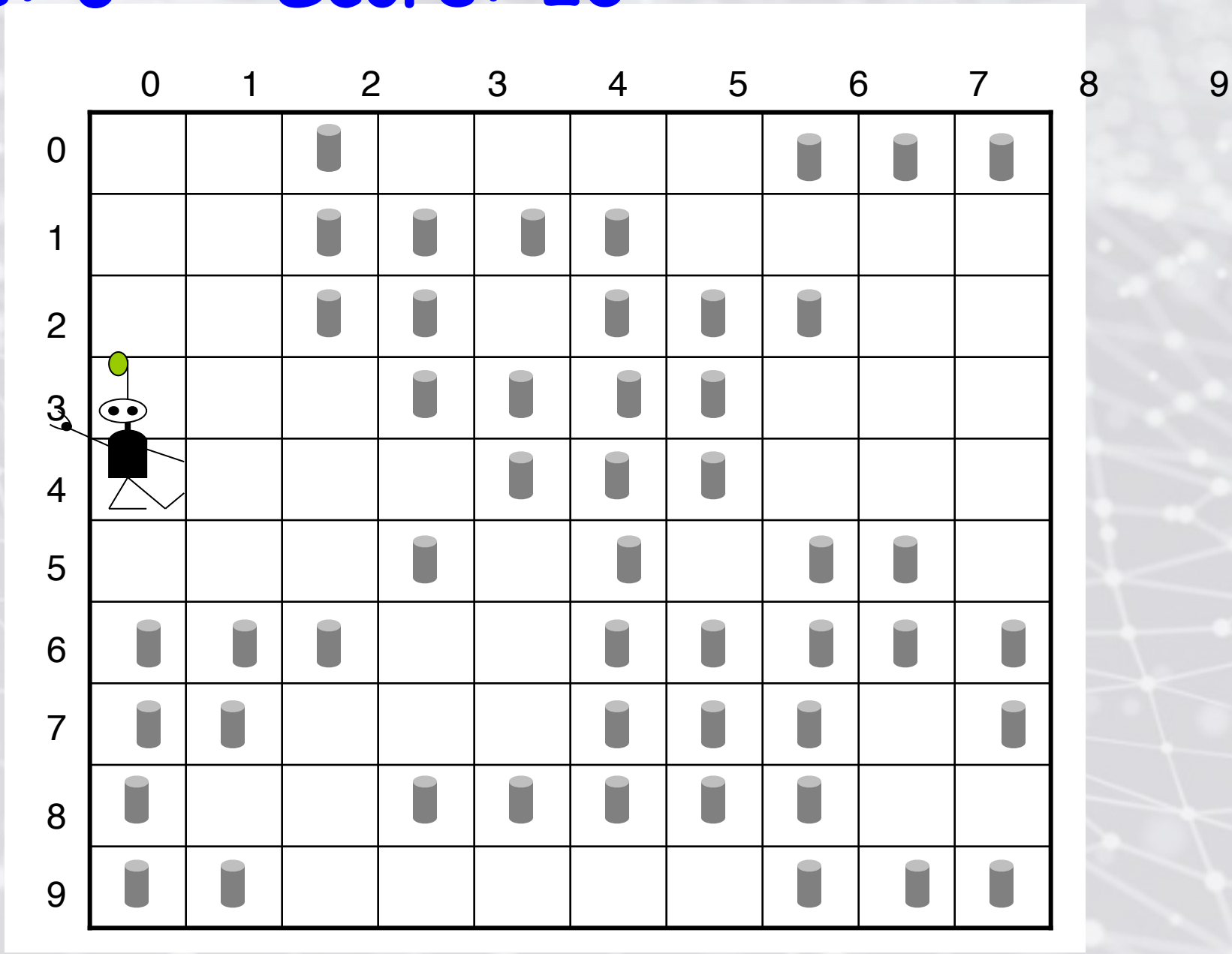
Time: 4 Score: 20



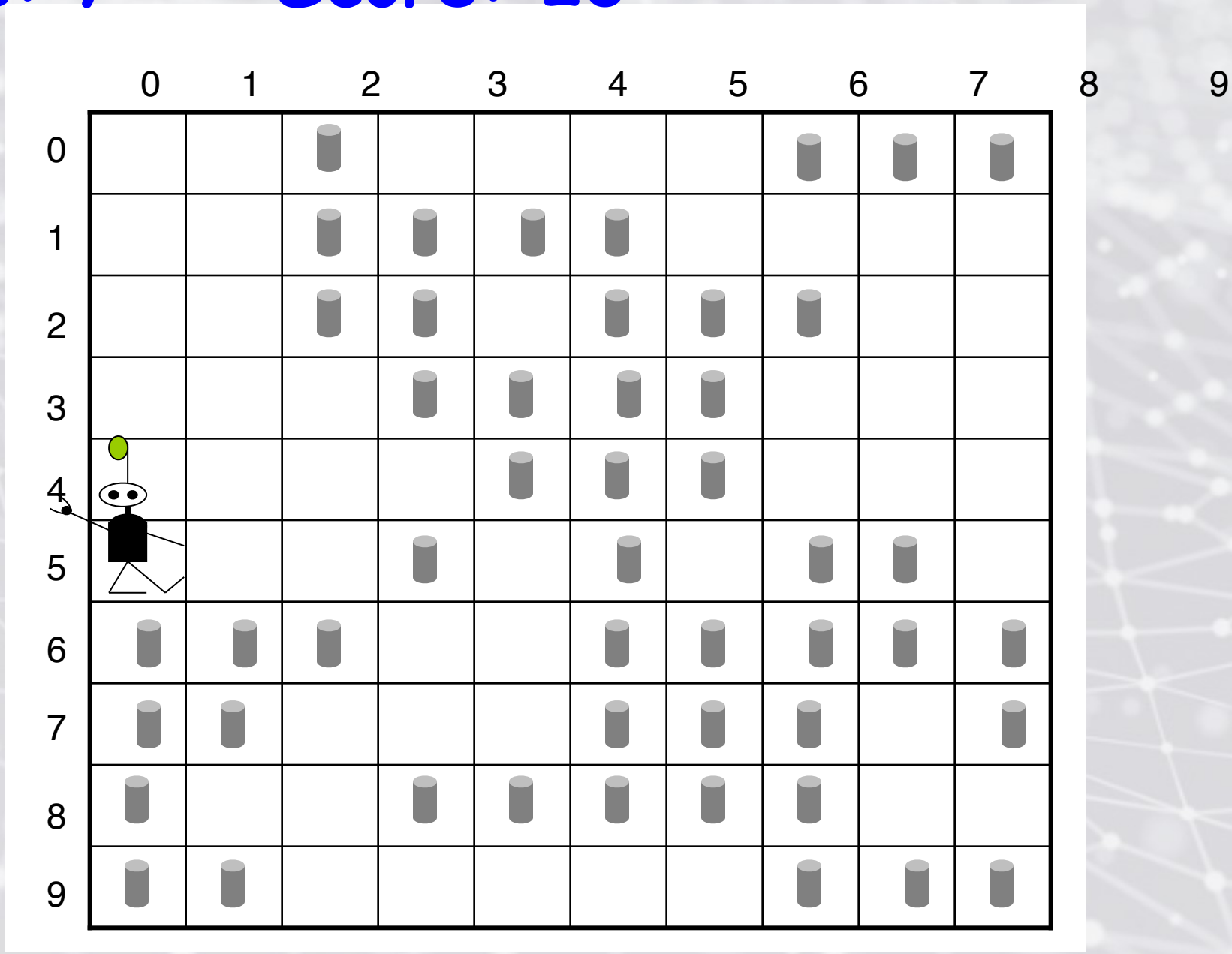
Time: 5 Score: 20



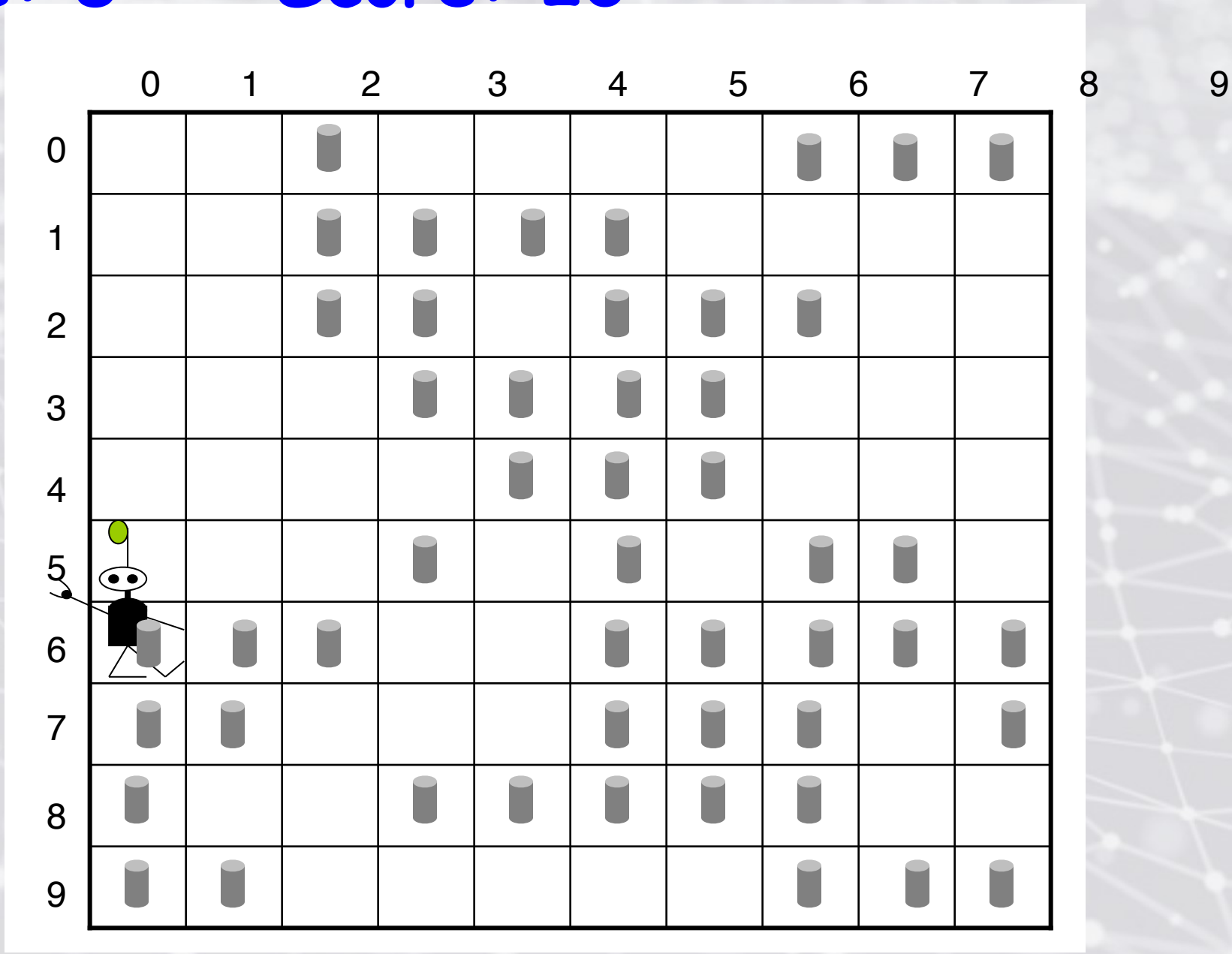
Time: 6 Score: 20



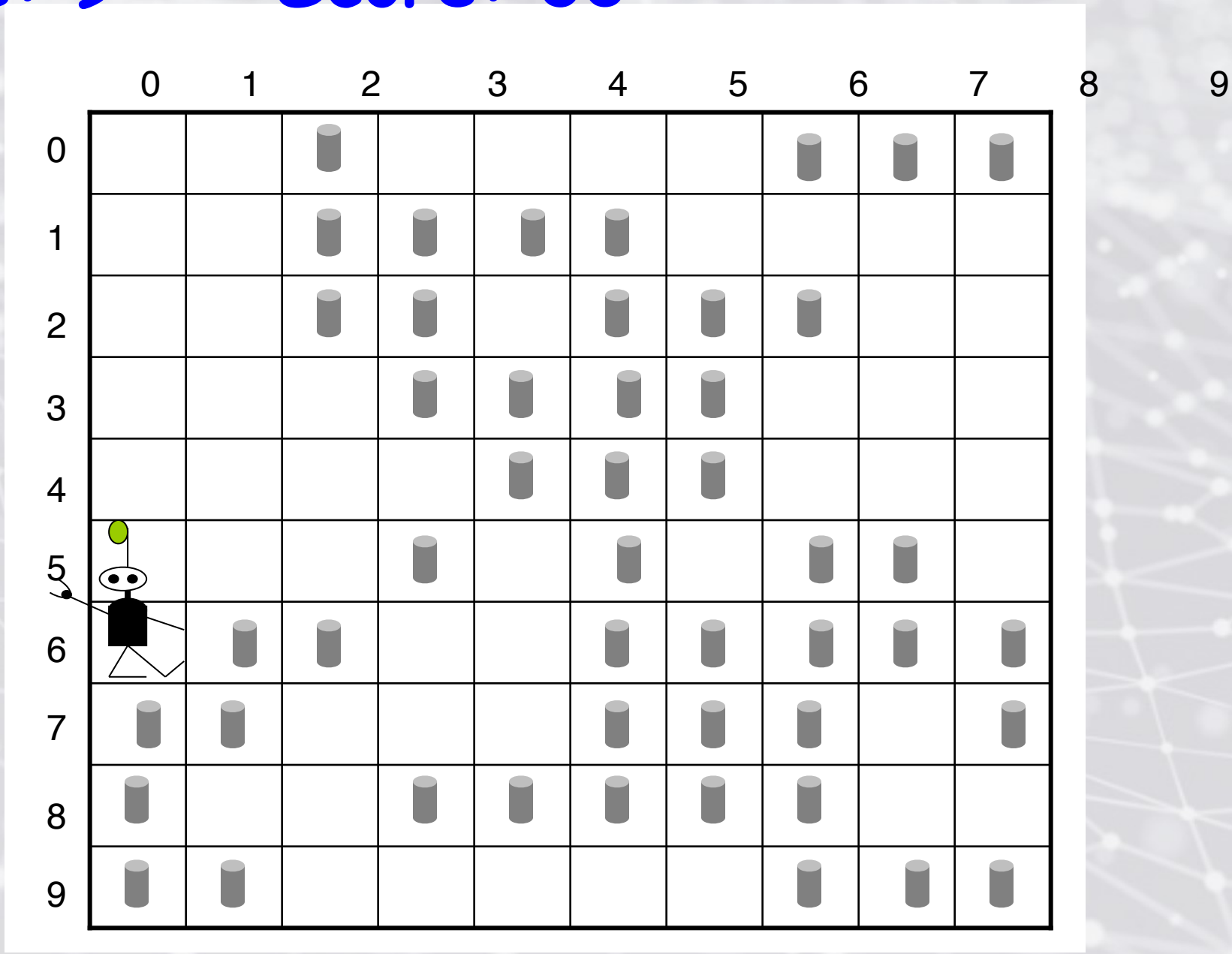
Time: 7 Score: 20



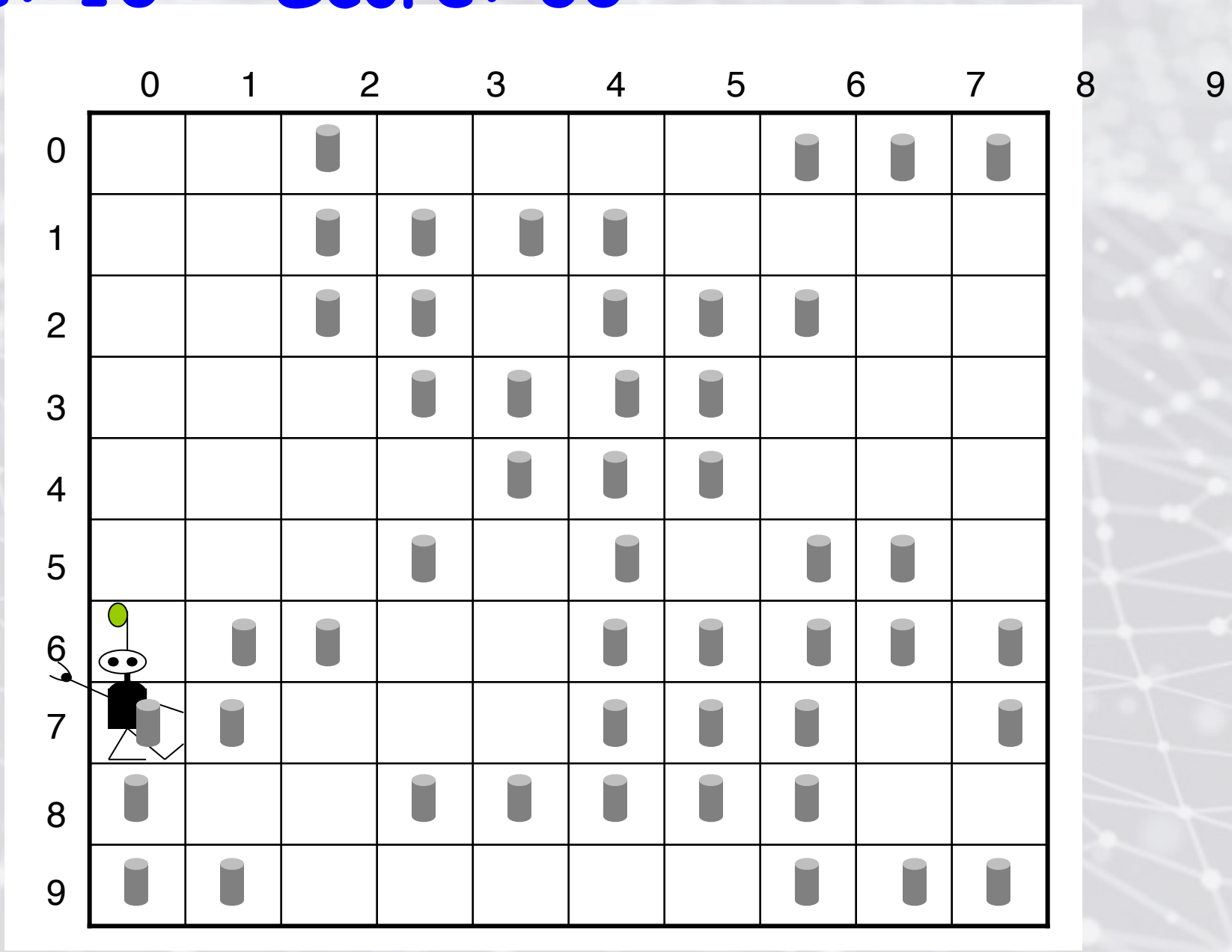
Time: 8 Score: 20



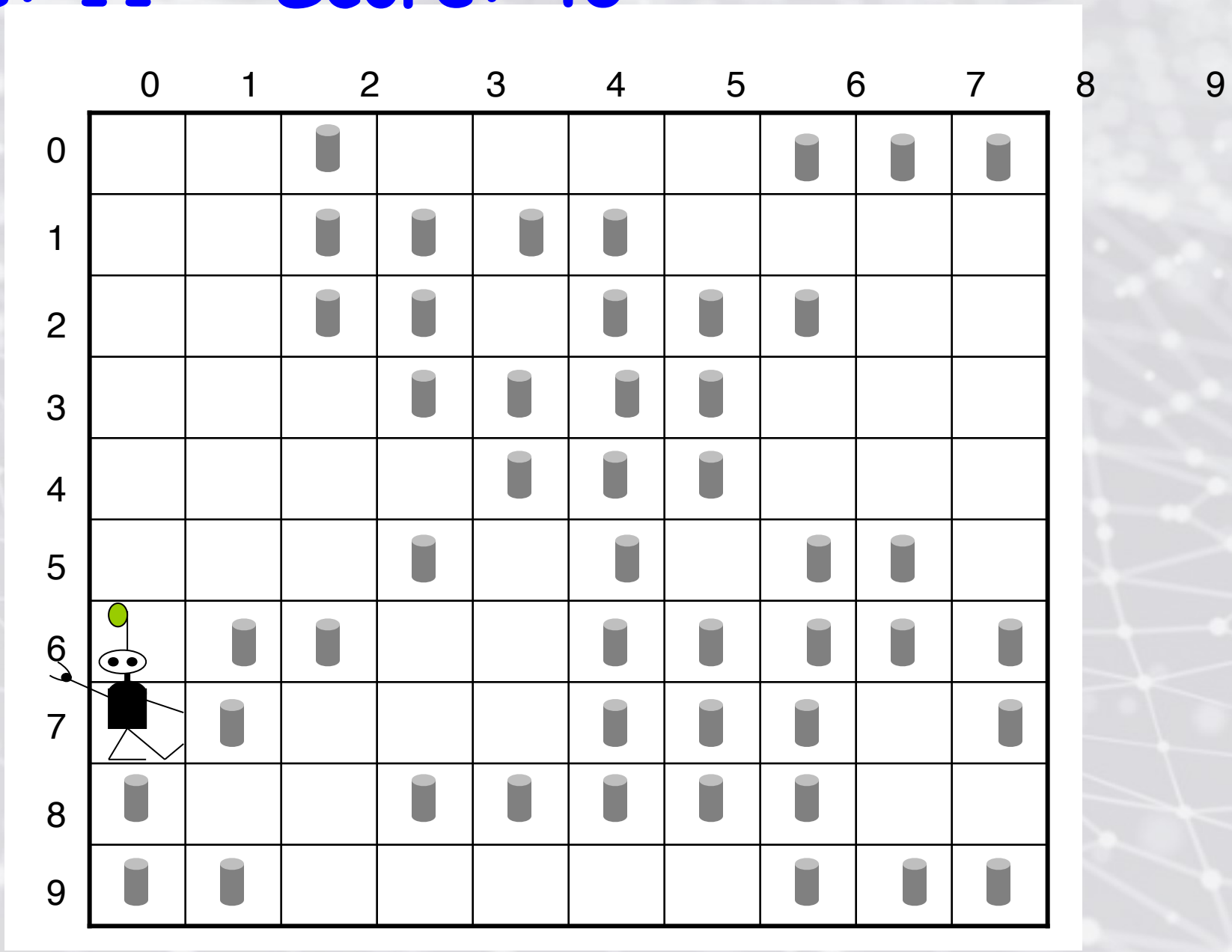
Time: 9 Score: 30



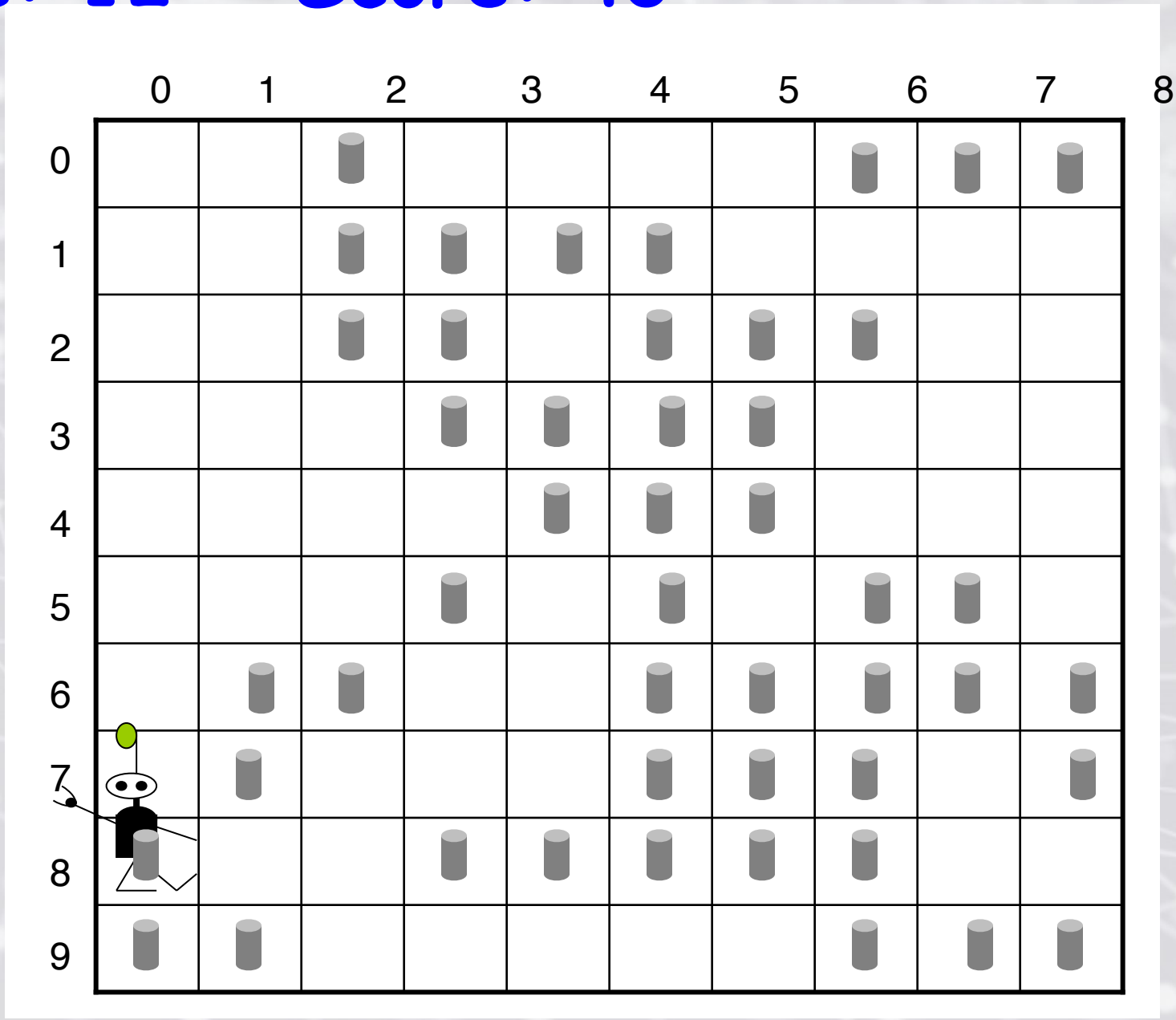
Time: 10 Score: 30



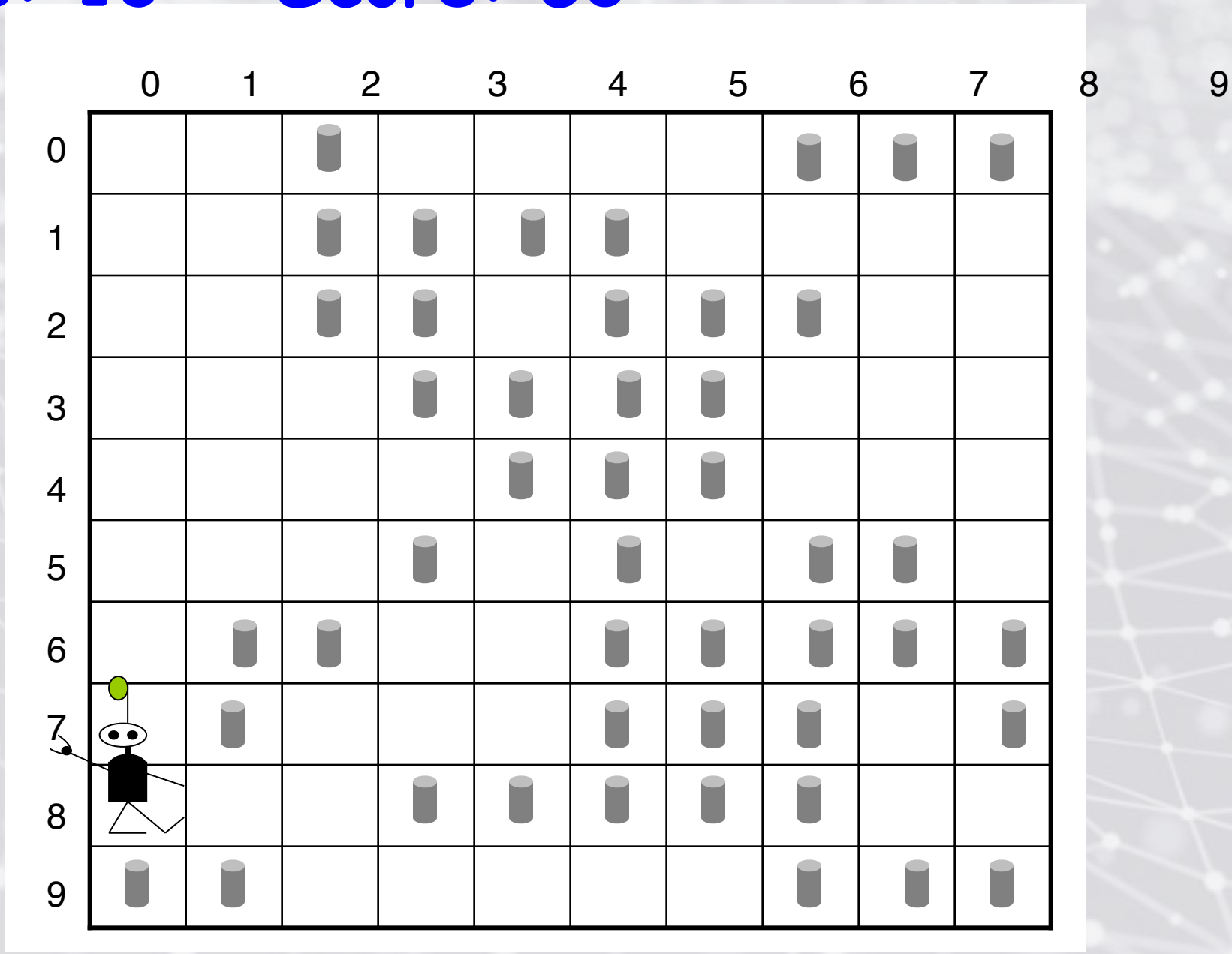
Time: 11 Score: 40



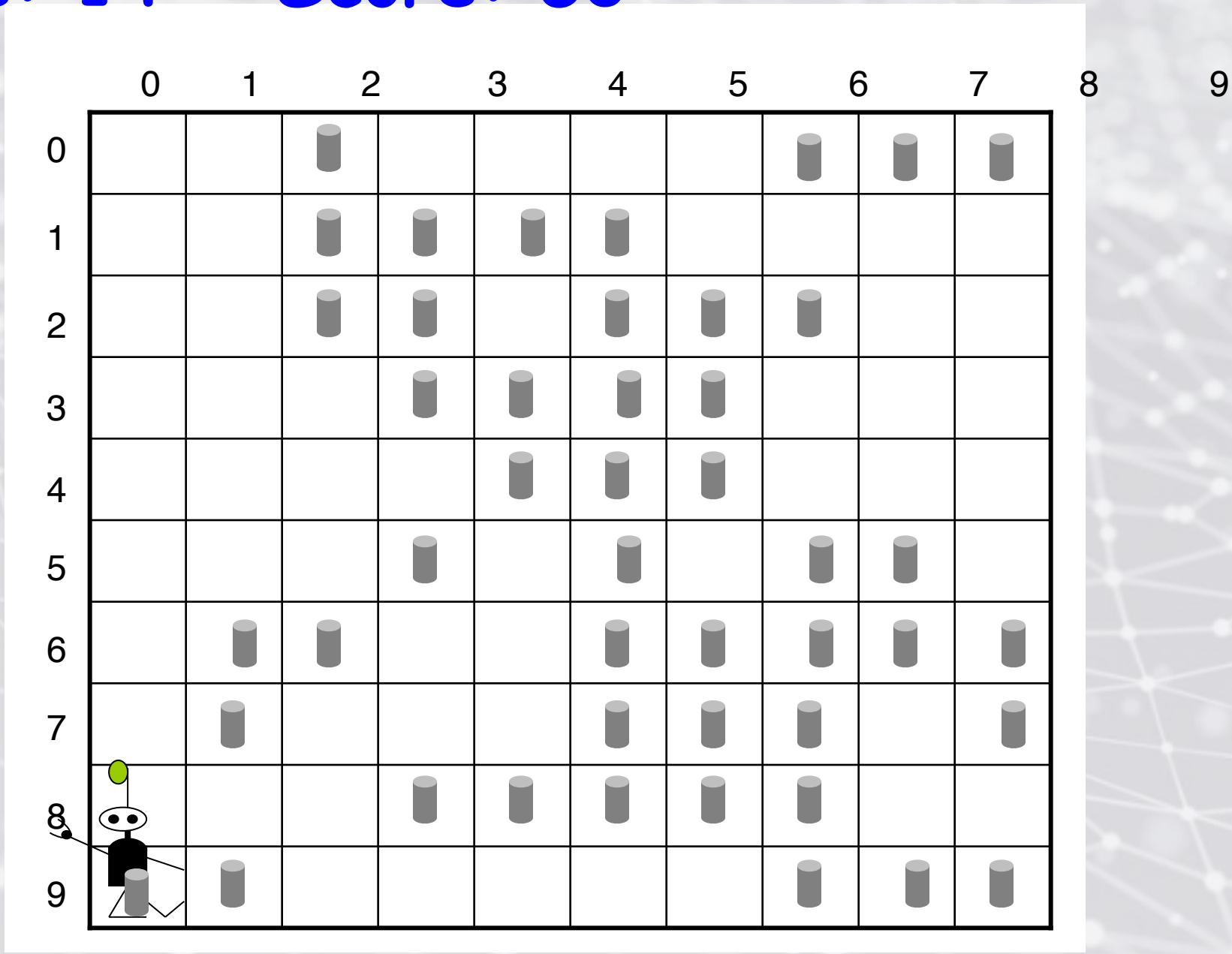
Time: 12 Score: 40



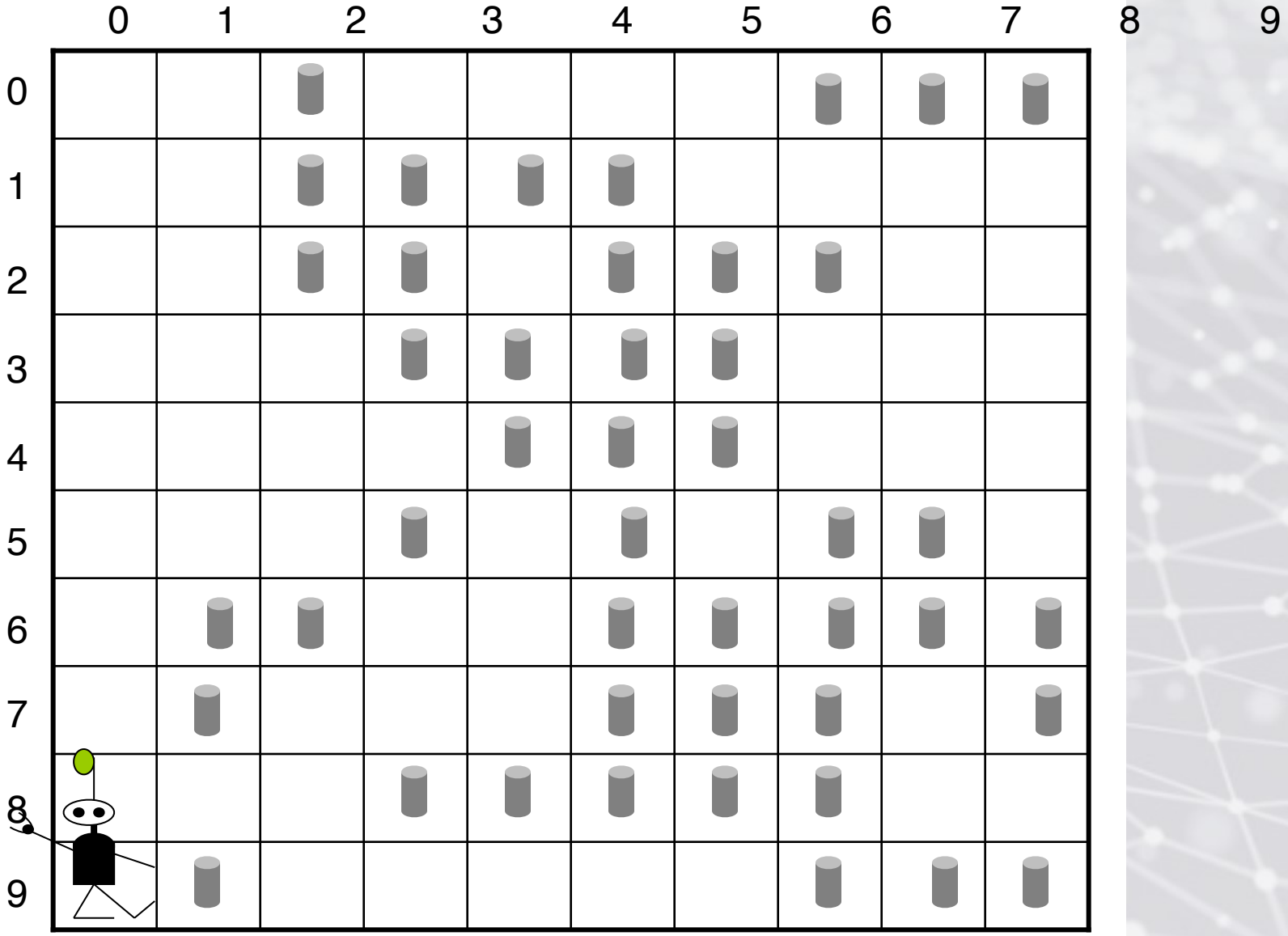
Time: 13 Score: 50



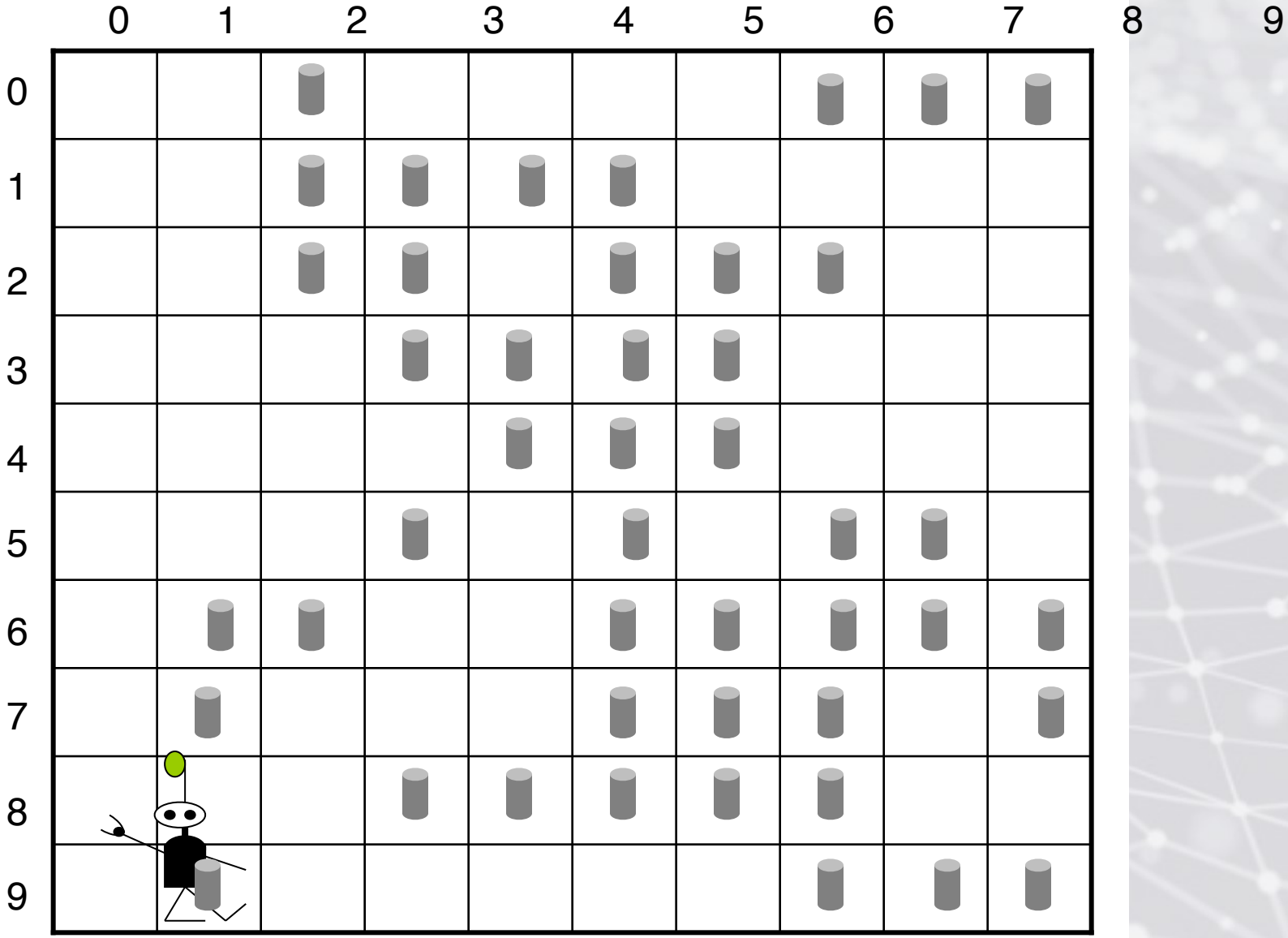
Time: 14 Score: 50



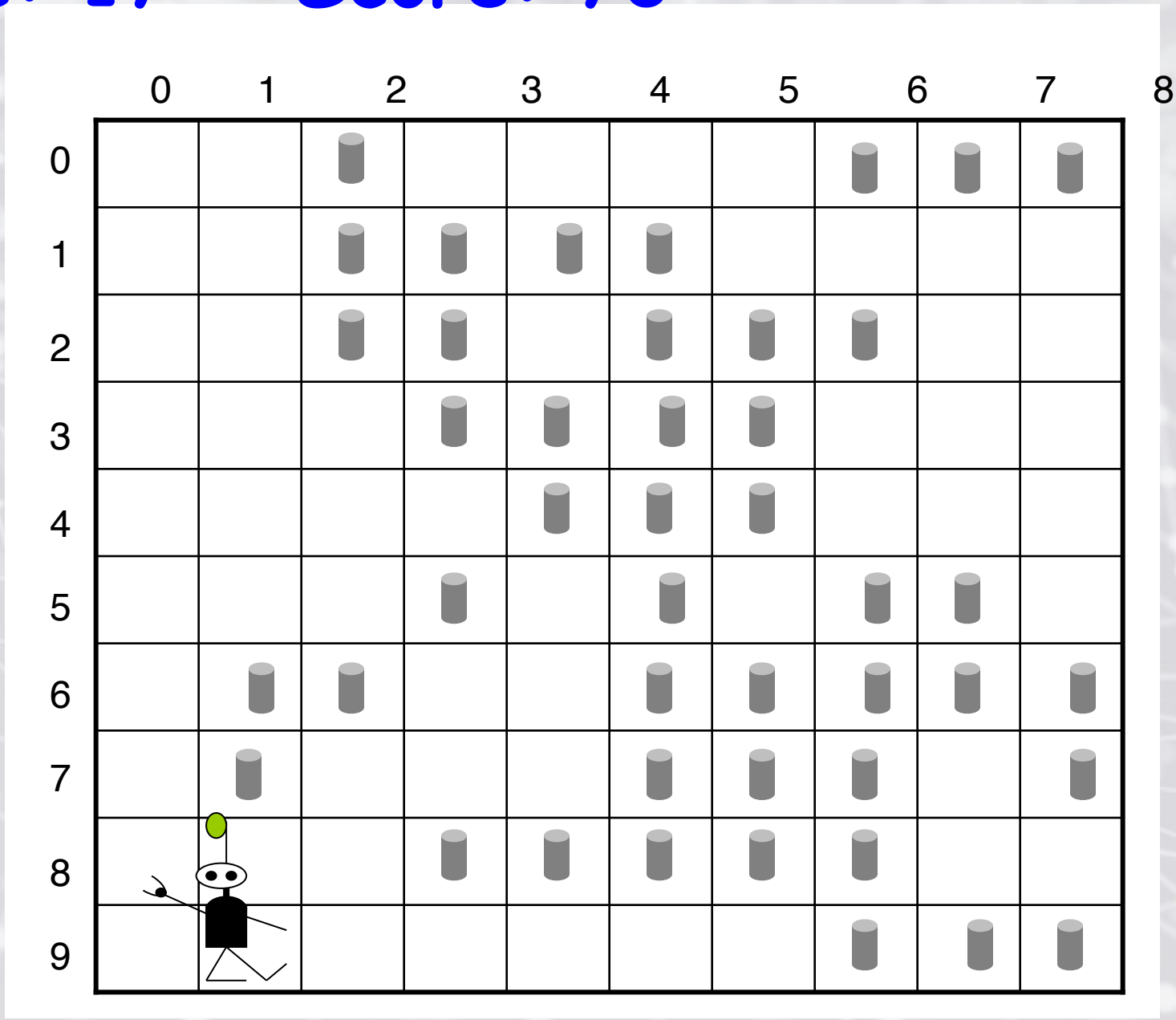
Time: 15 Score: 60



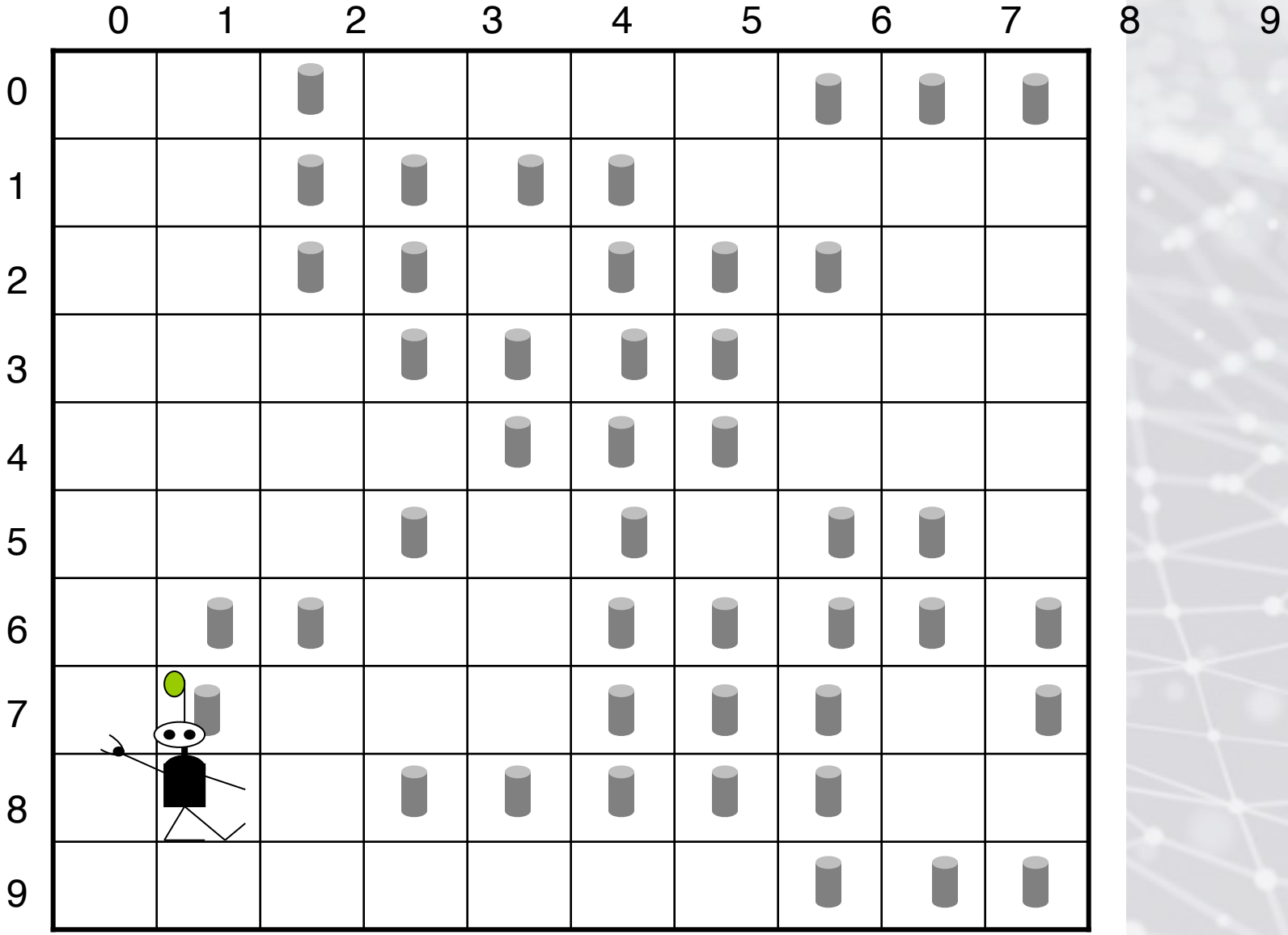
Time: 16 Score: 60



Time: 17 Score: 70

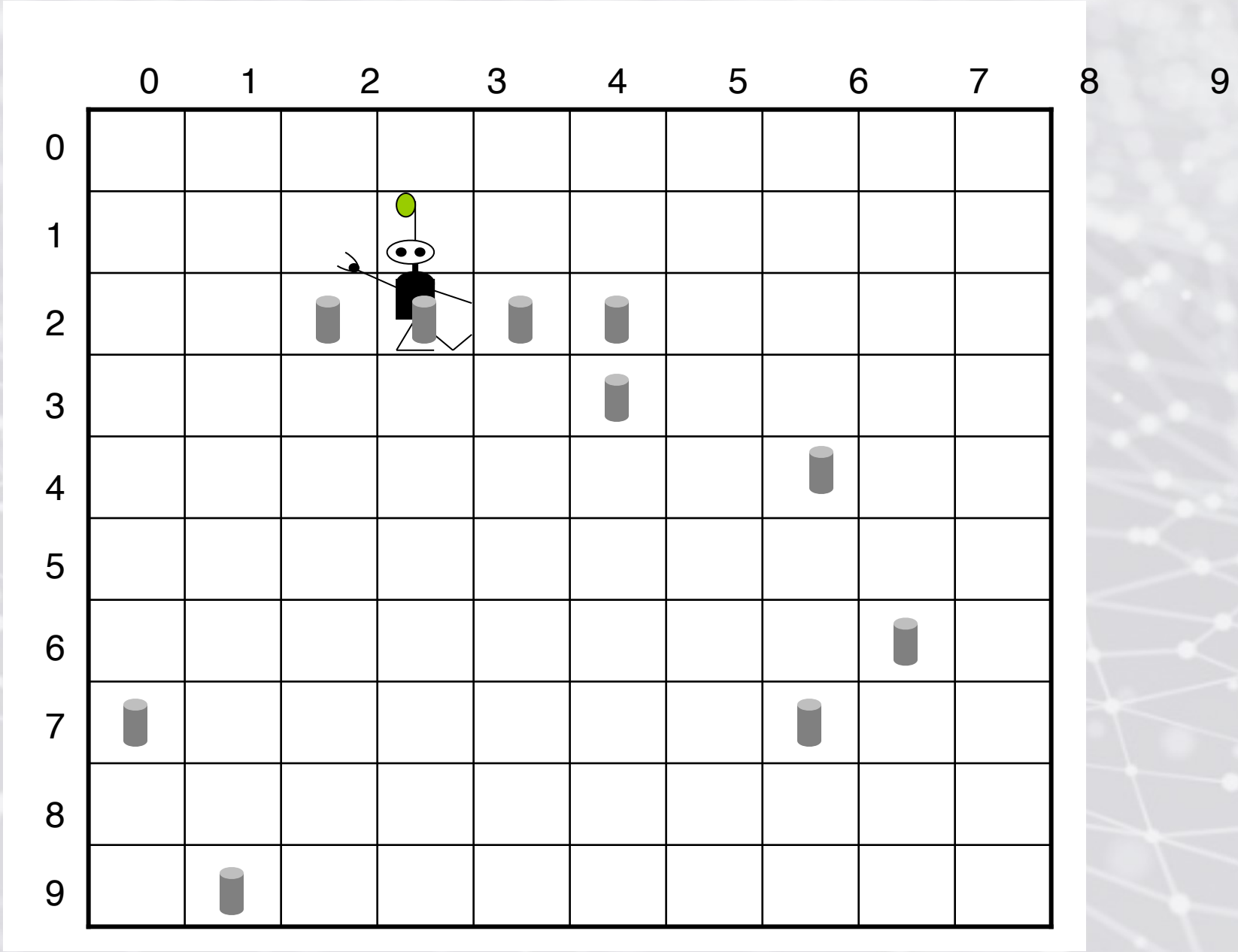


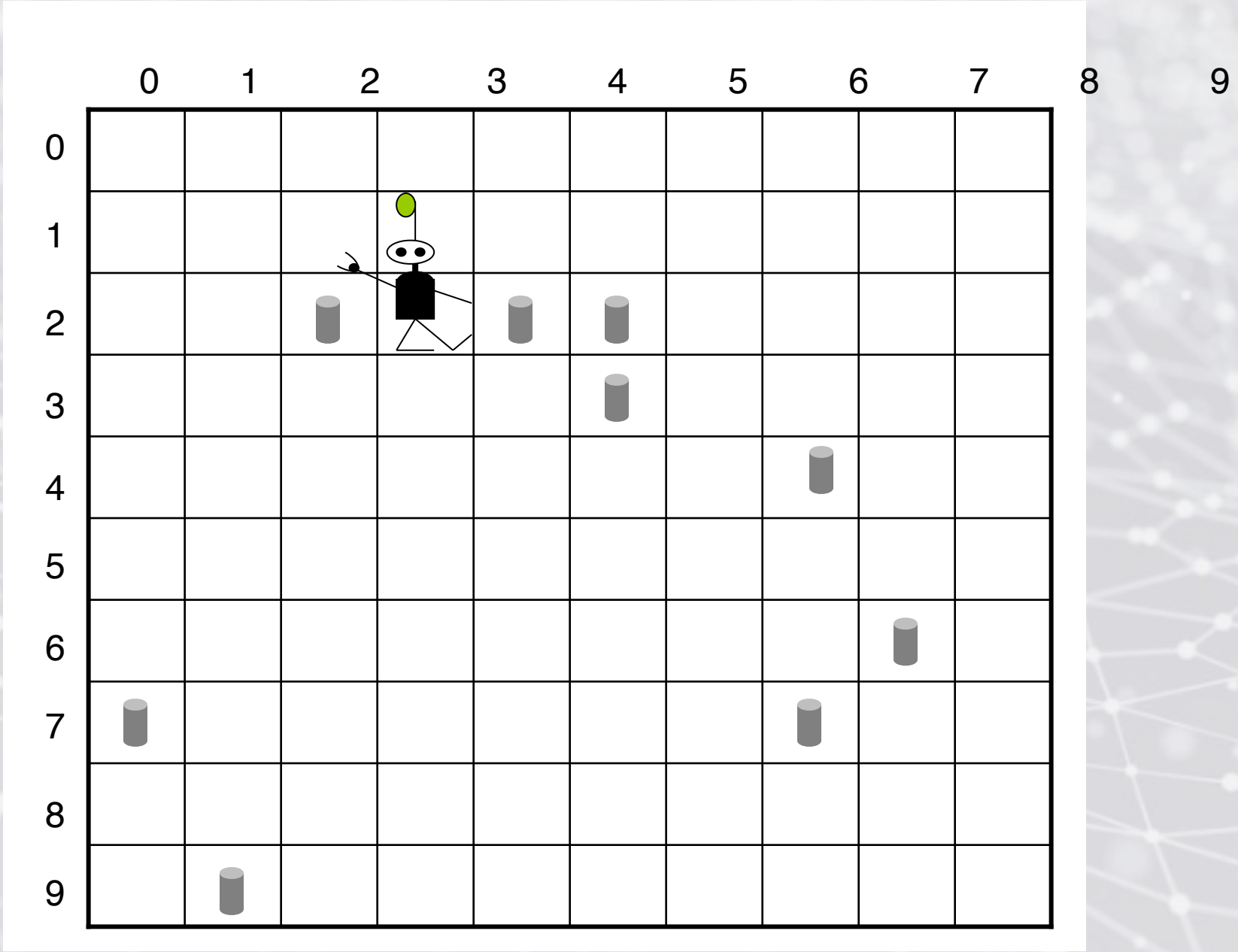
Time: 18 Score: 70

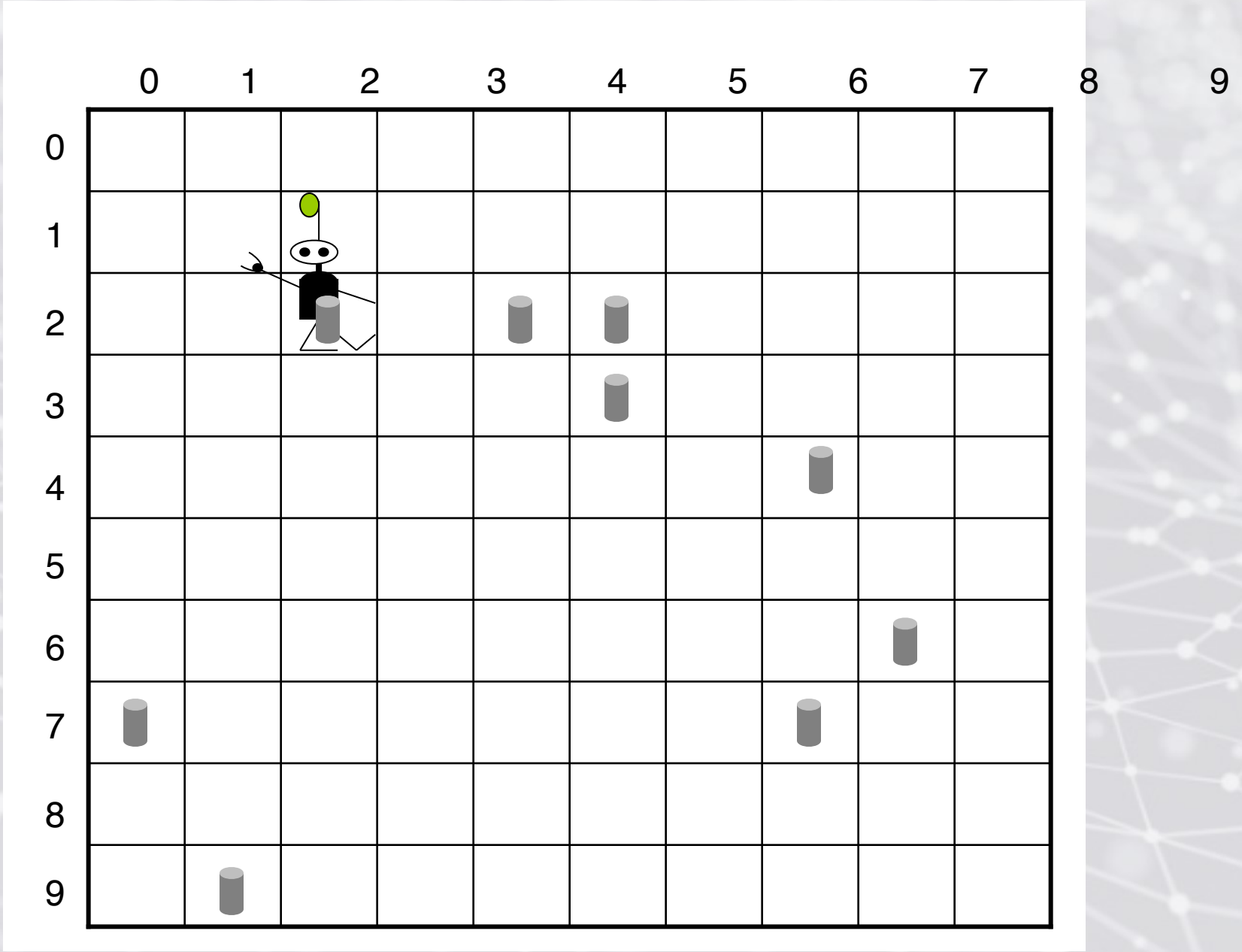


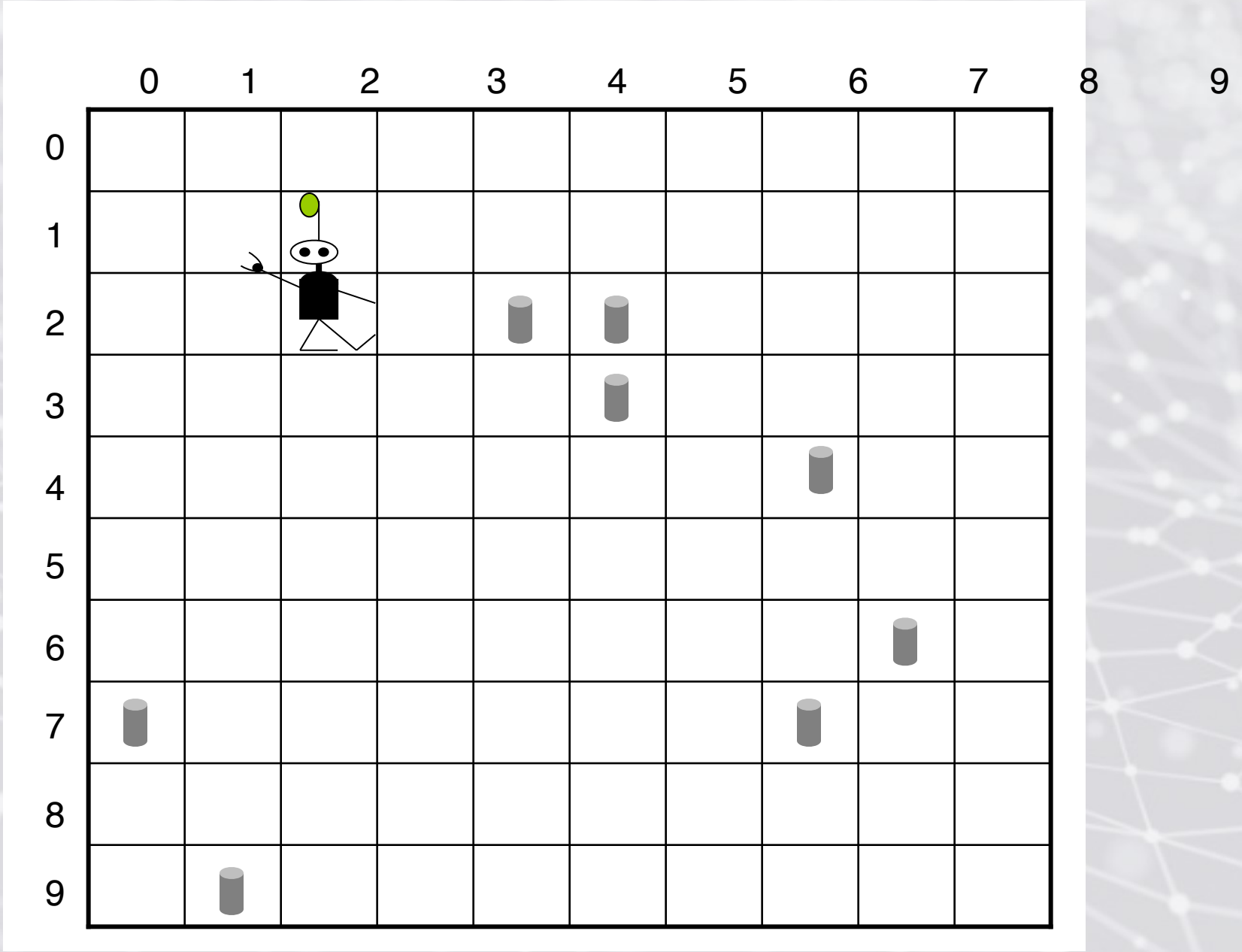
Why Did The GA's Strategy Outperform Mine?

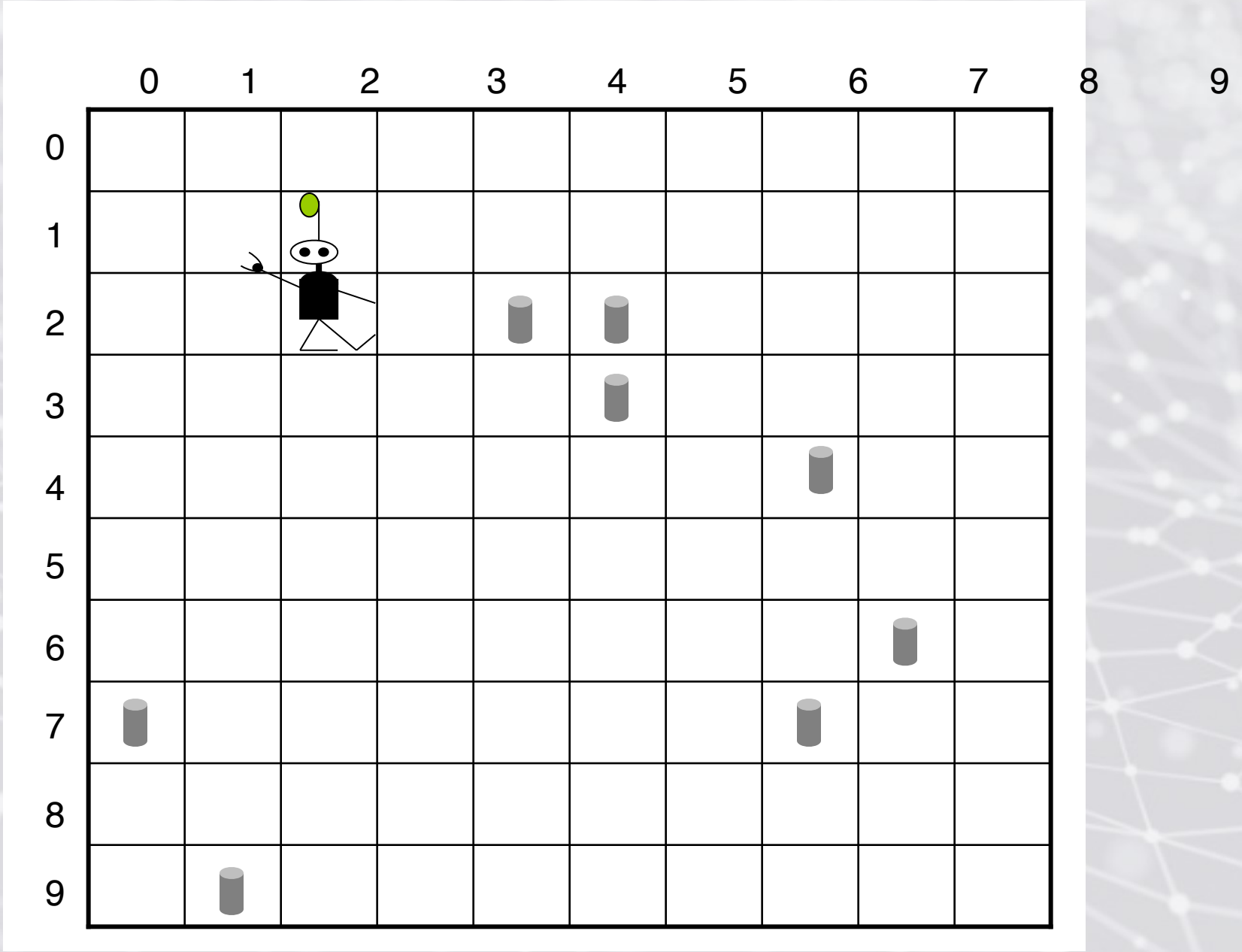
My Strategy


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

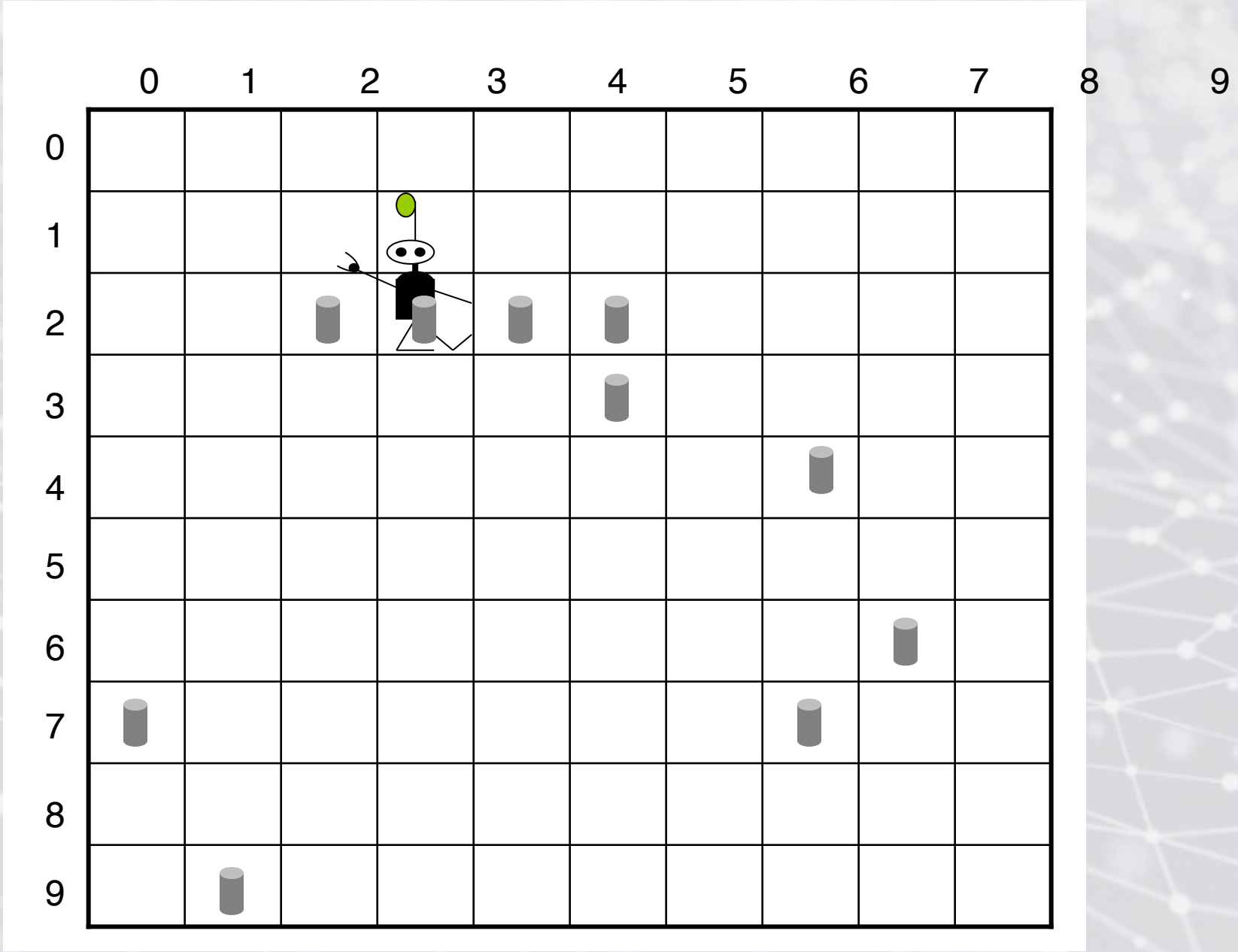

0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

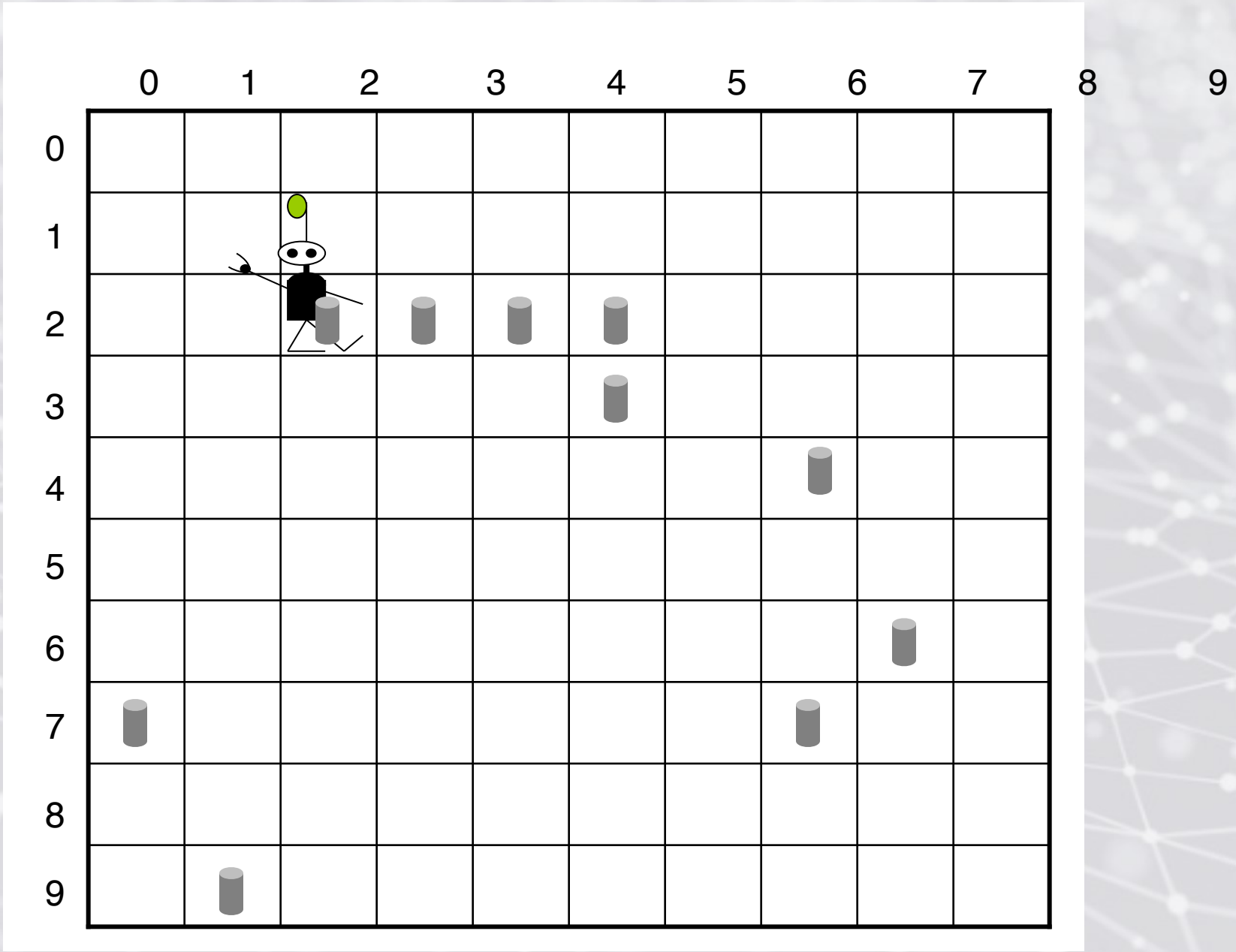

0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

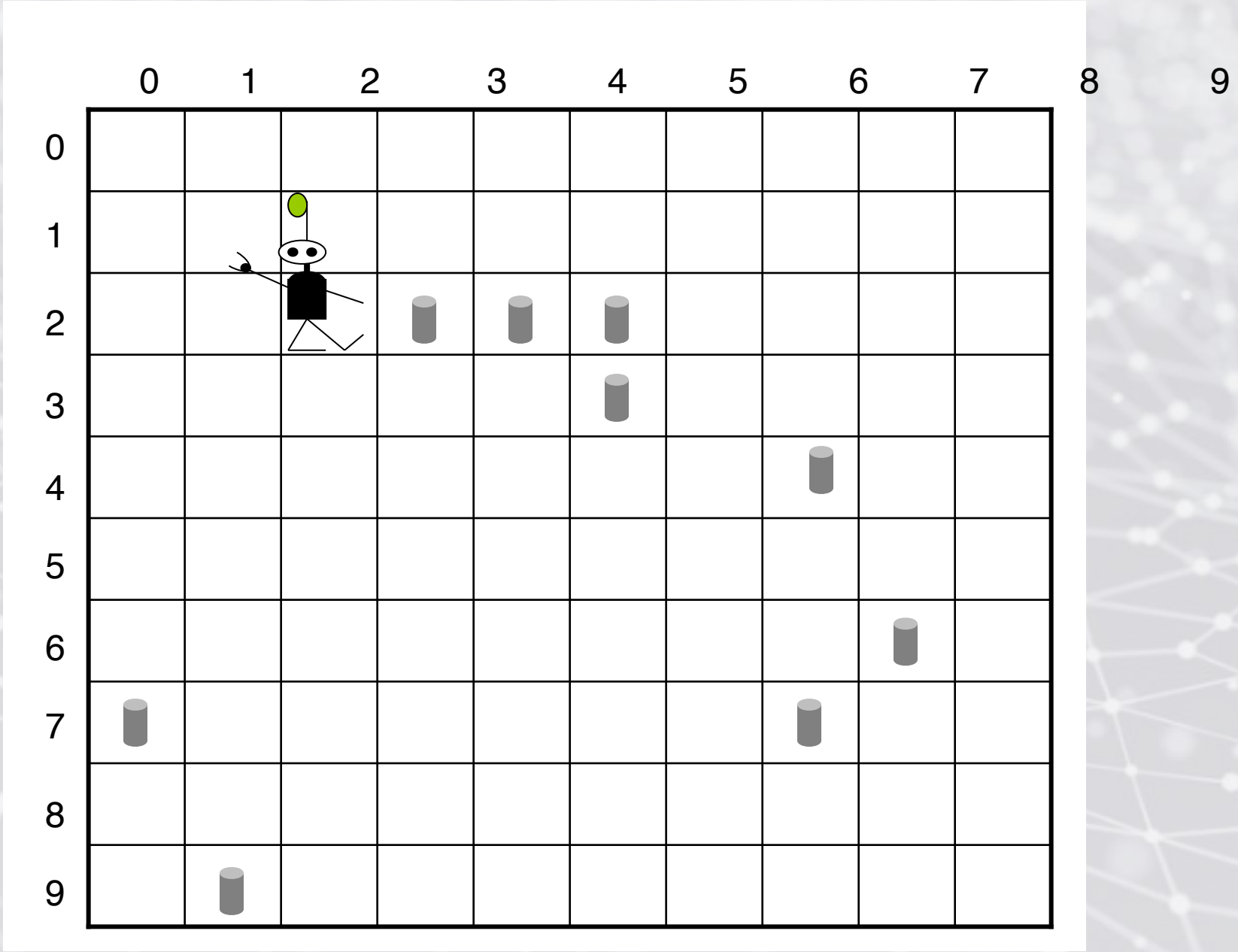

0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9

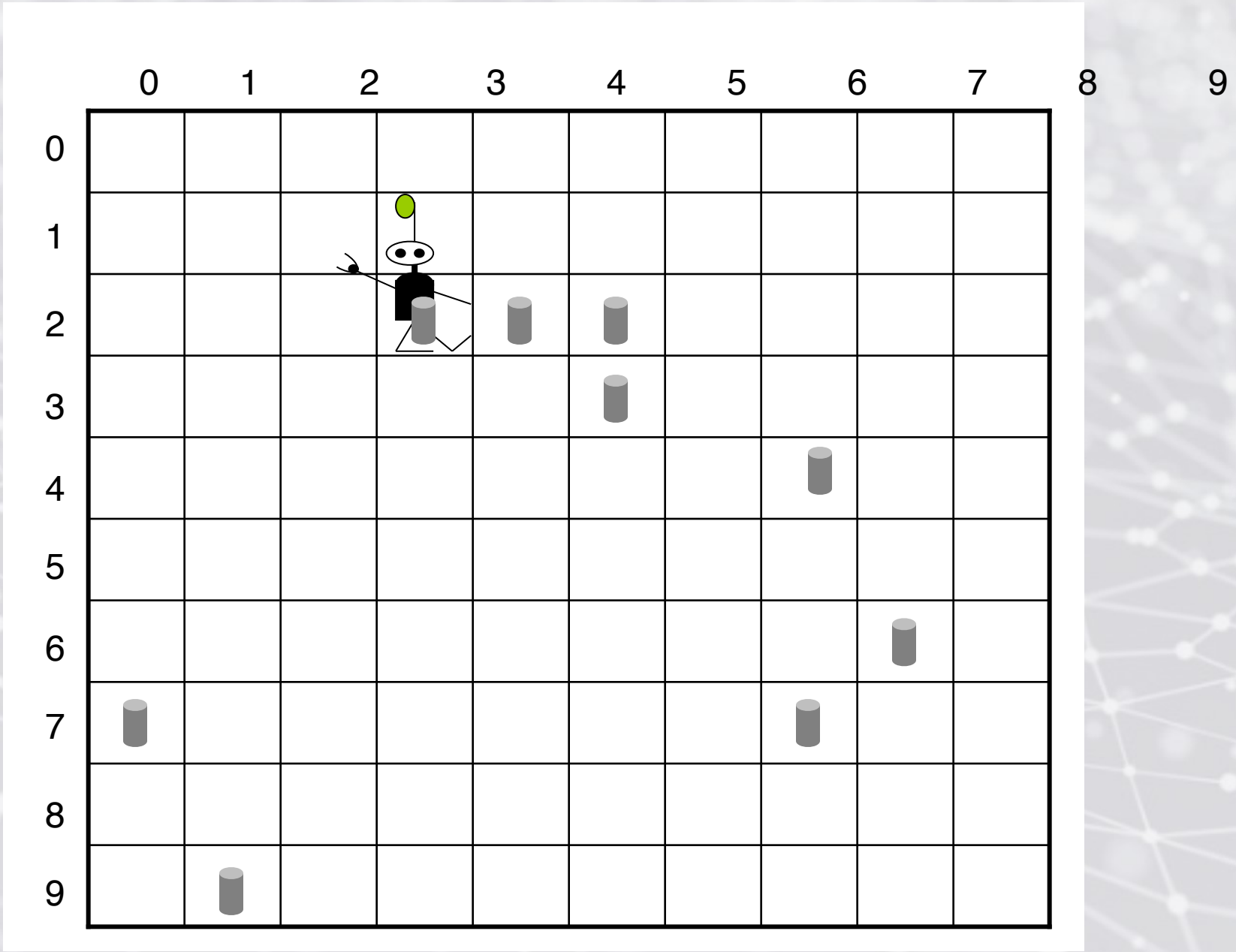

0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9

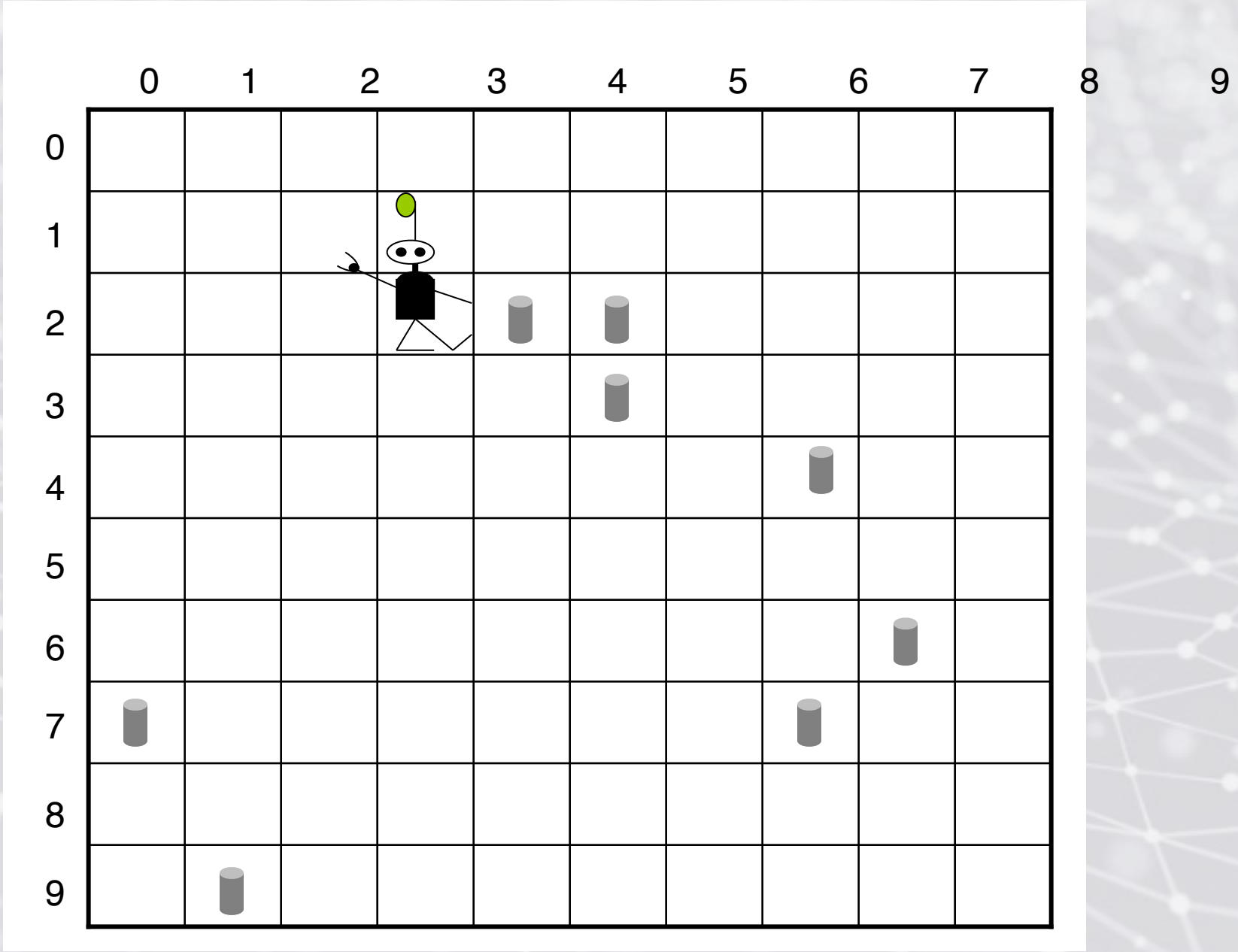
The GA's Evolved Strategy

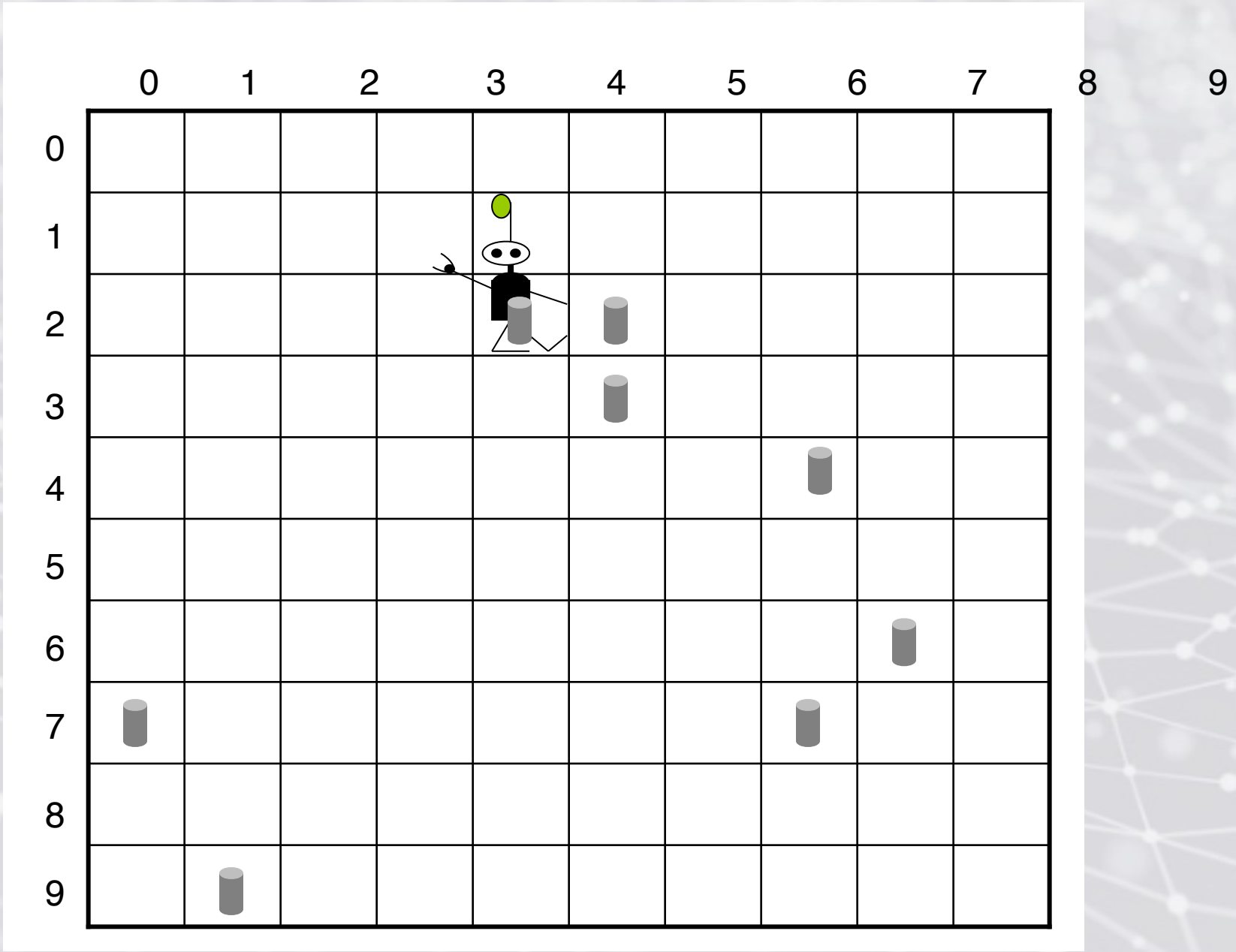

0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9

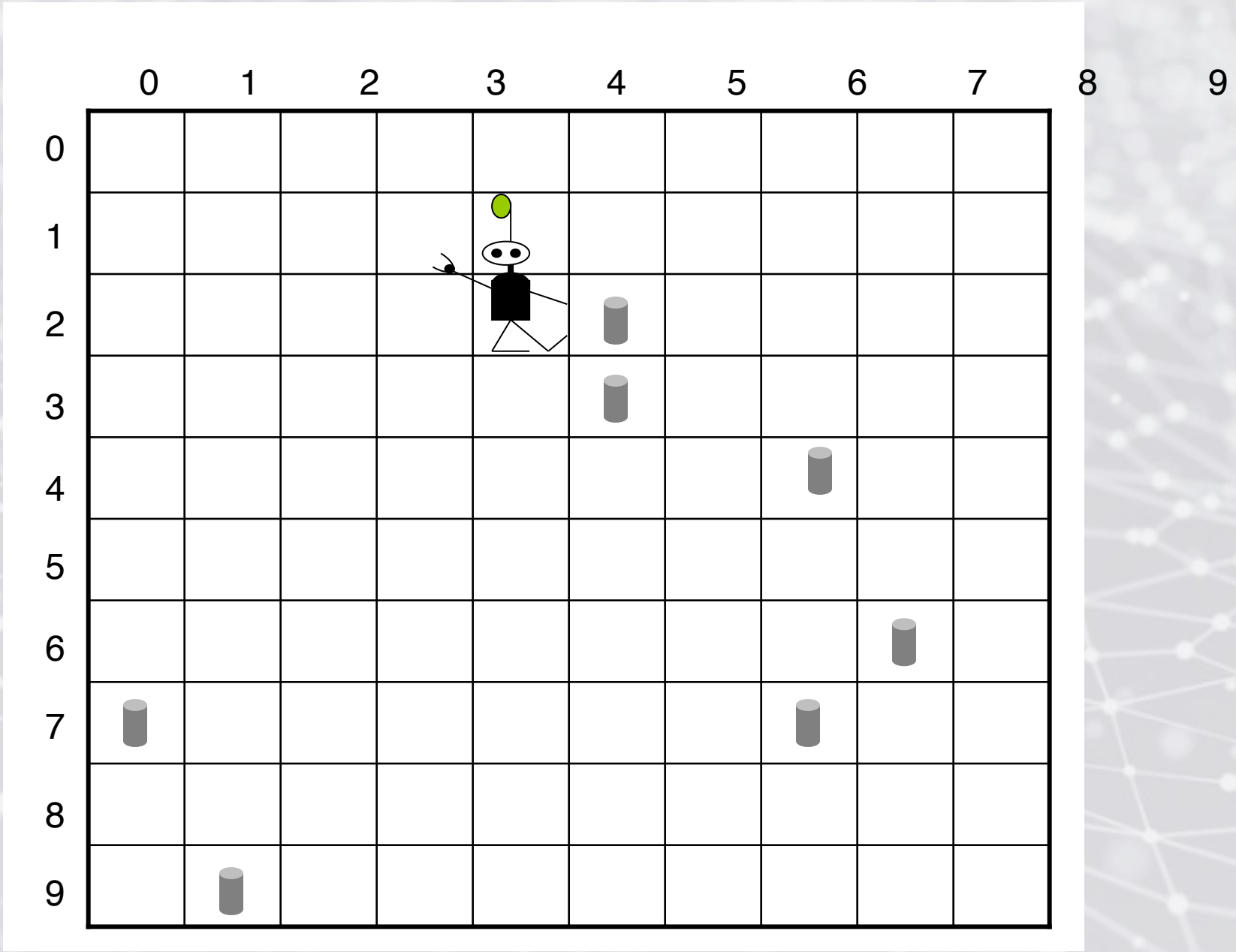

0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9

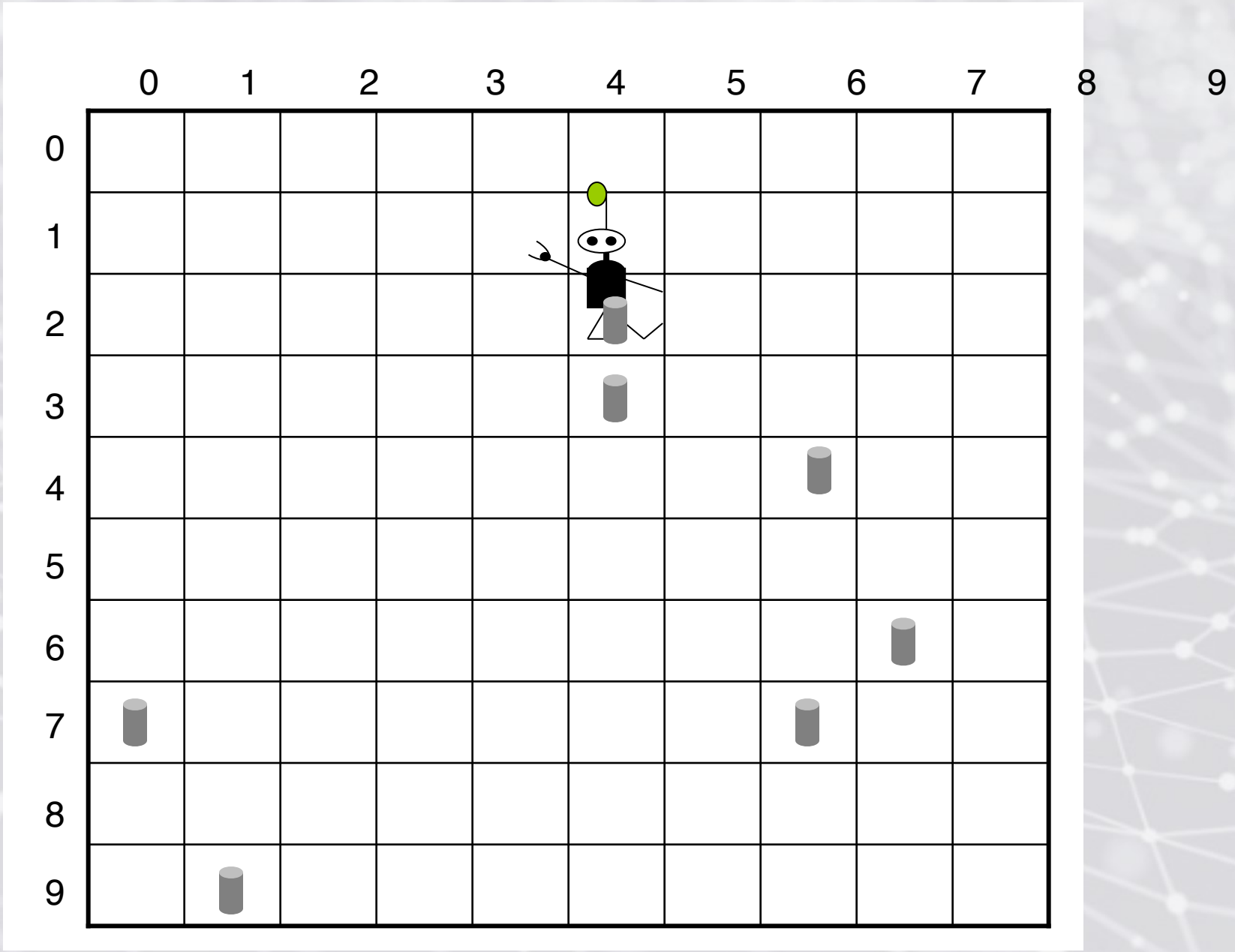

0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9

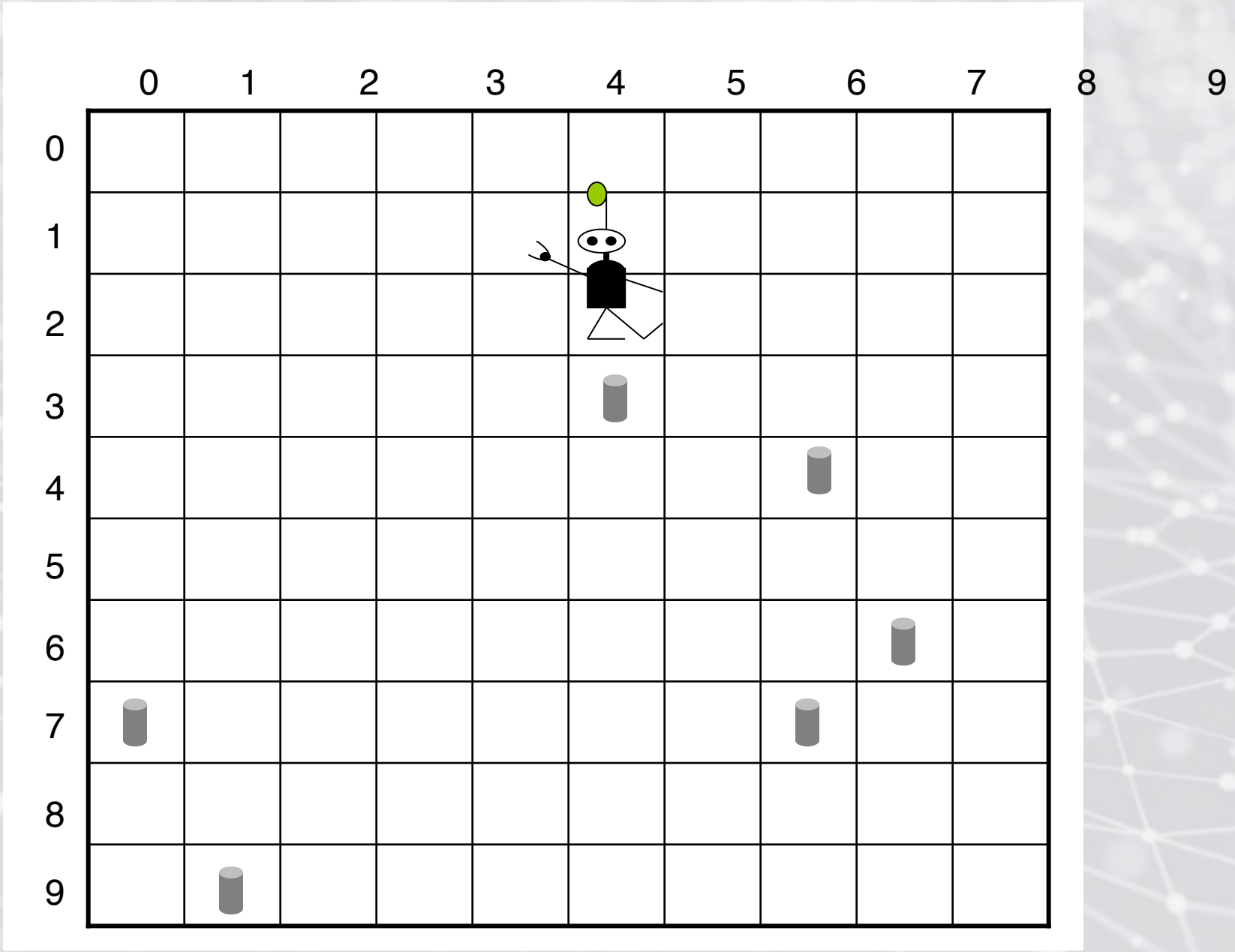

0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

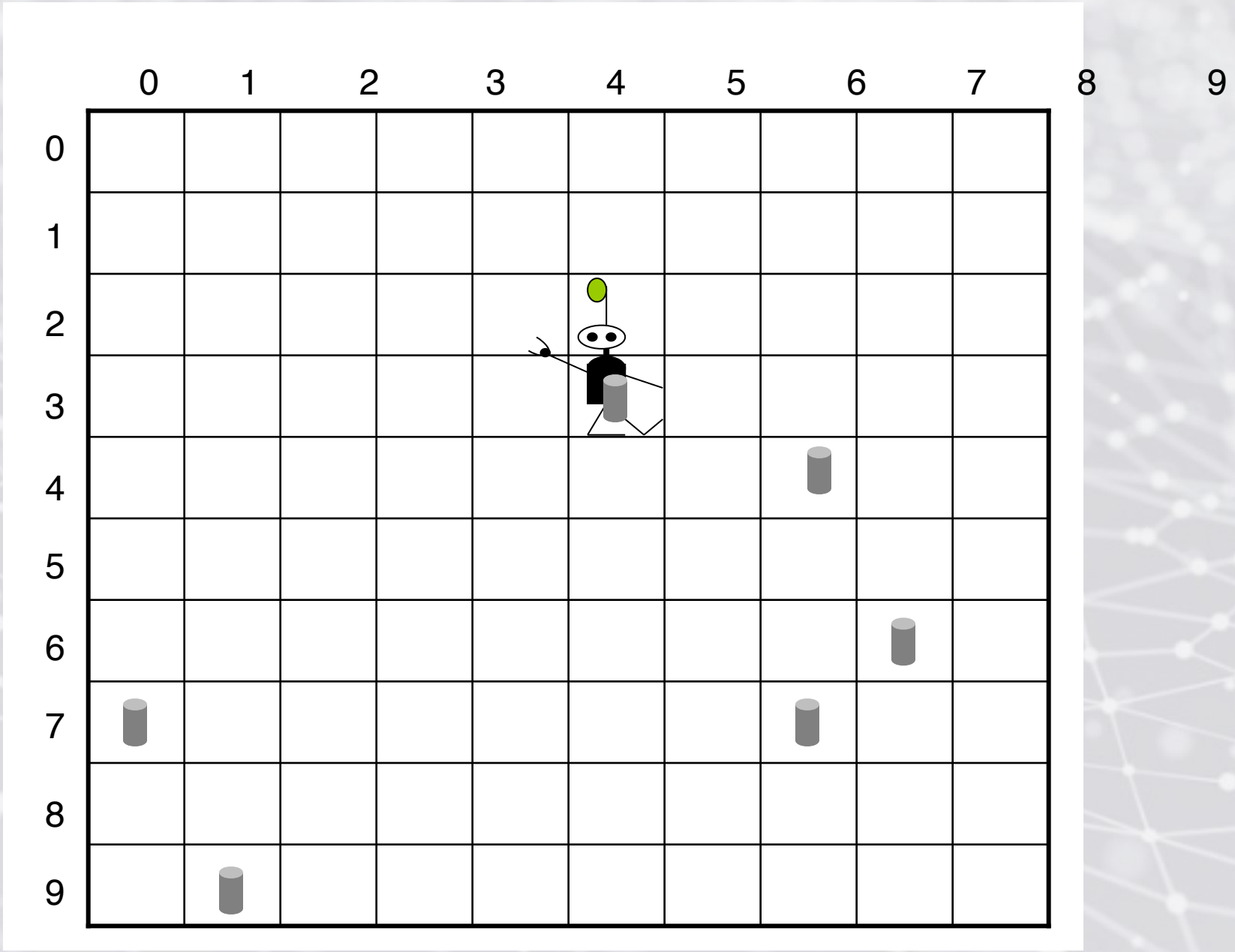

0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

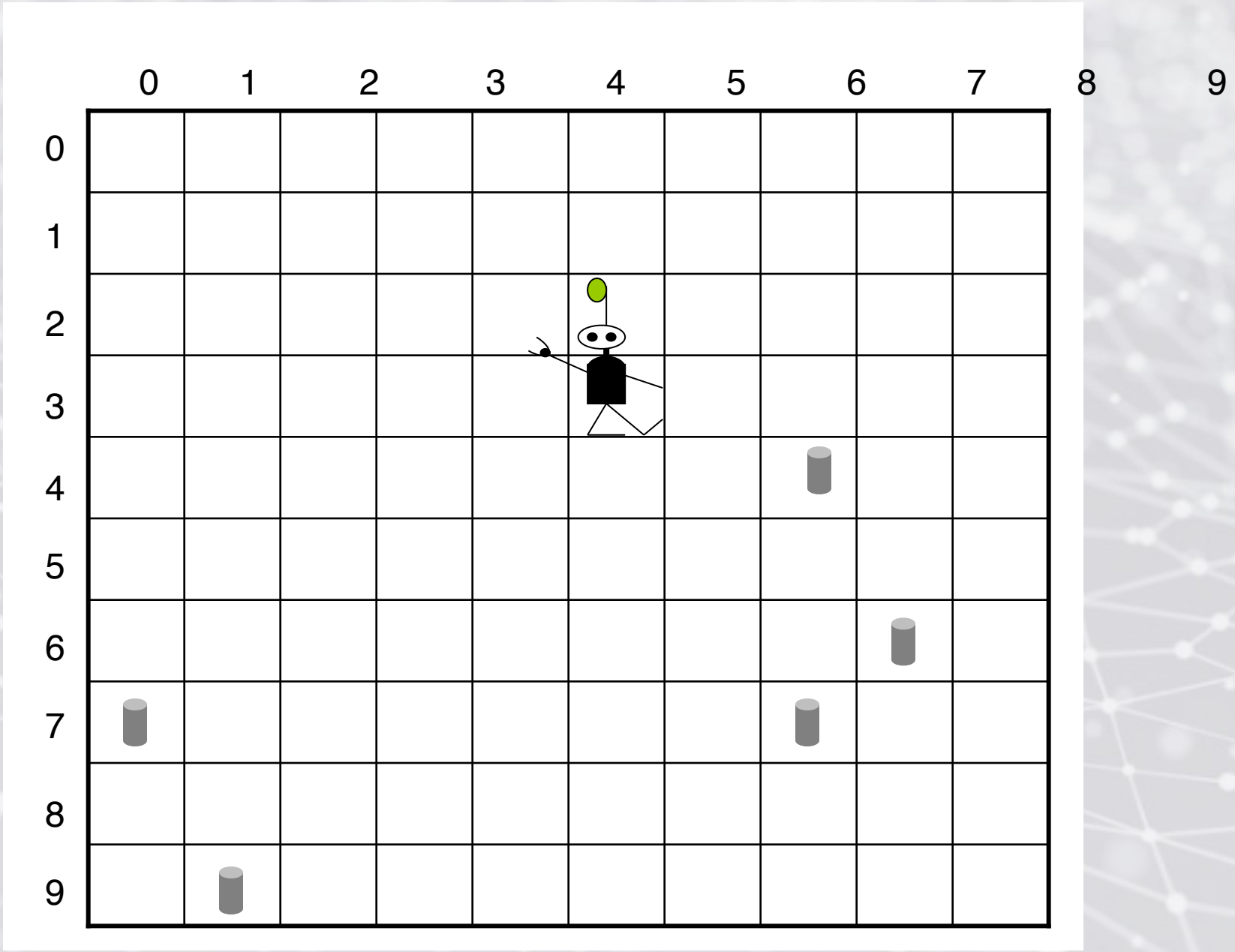

0 1 2 3 4 5 6 7 8 9
0
1
2
3
4
5
6
7
8
9


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

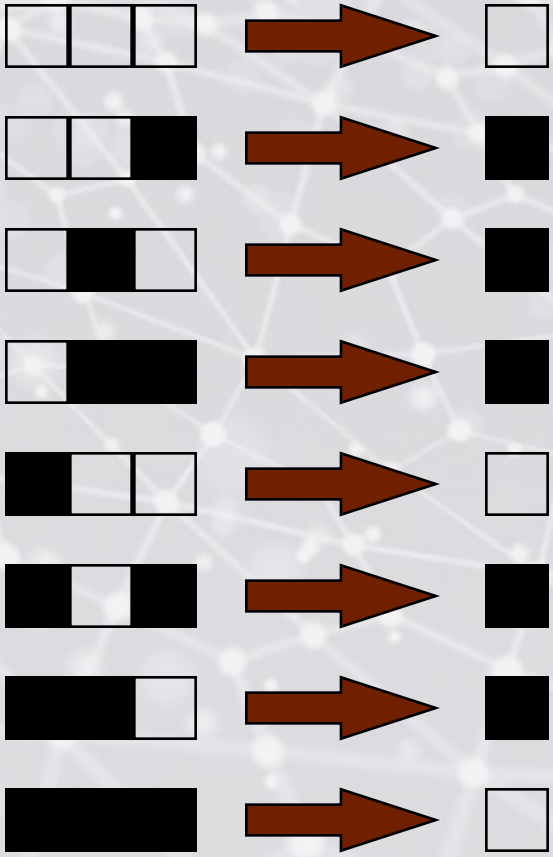
(Application to Cellular Automata
-- Bonus material)

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

One-dimensional cellular automata

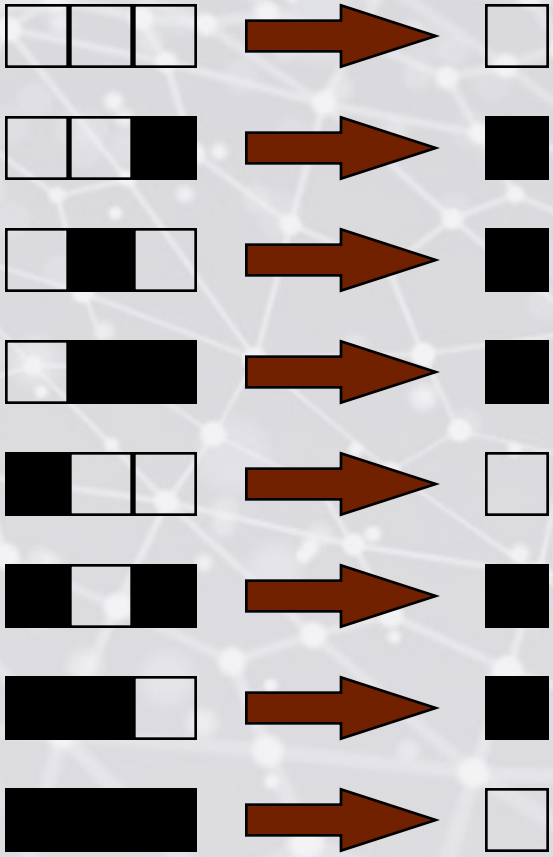
One-dimensional cellular automata

Rule:



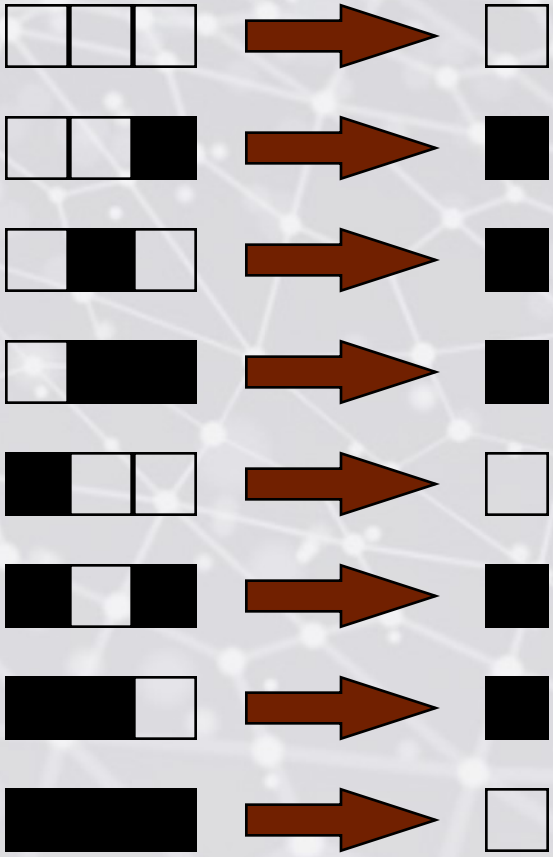
One-dimensional cellular automata

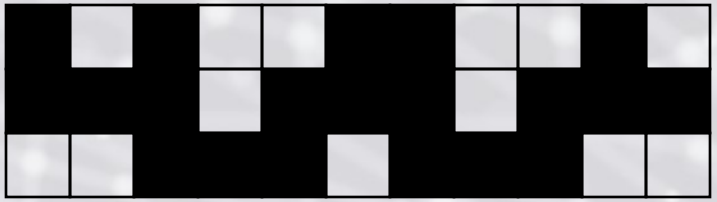
Rule:



One-dimensional cellular automata

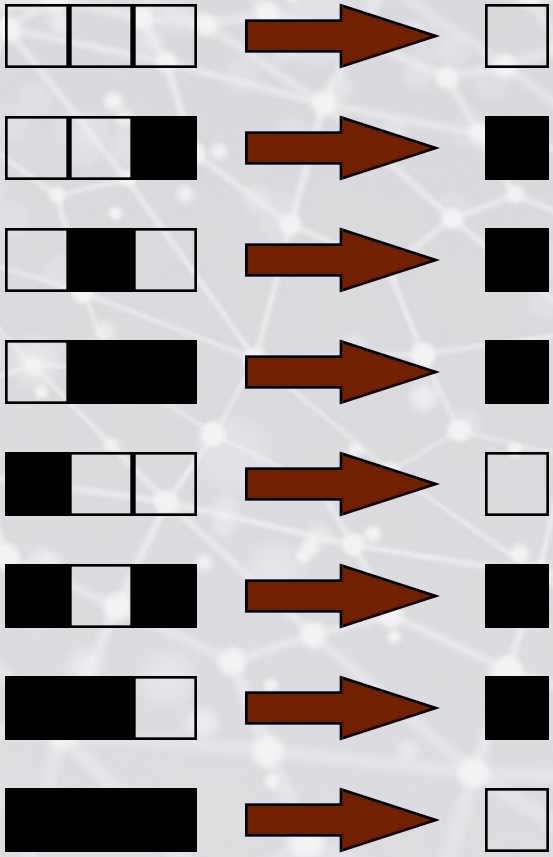
Rule:

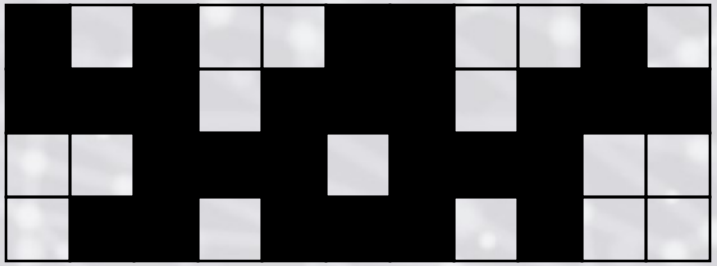




One-dimensional cellular automata

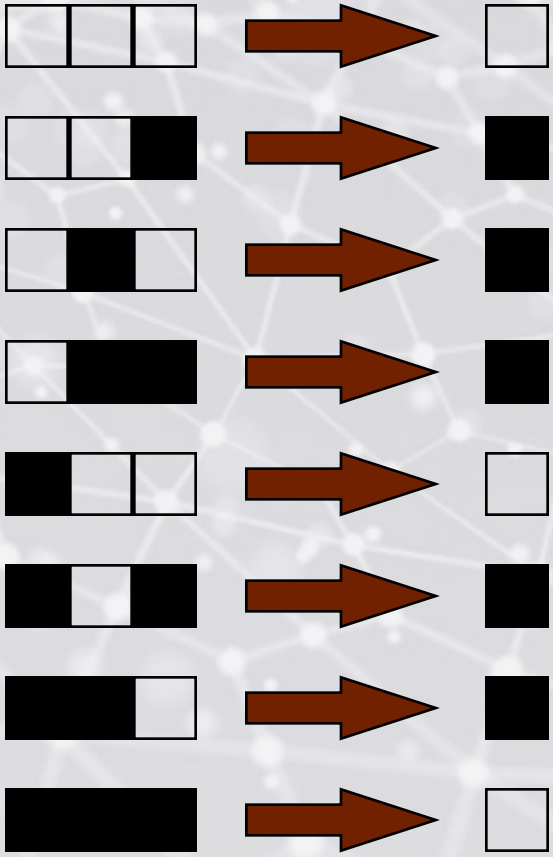
Rule:

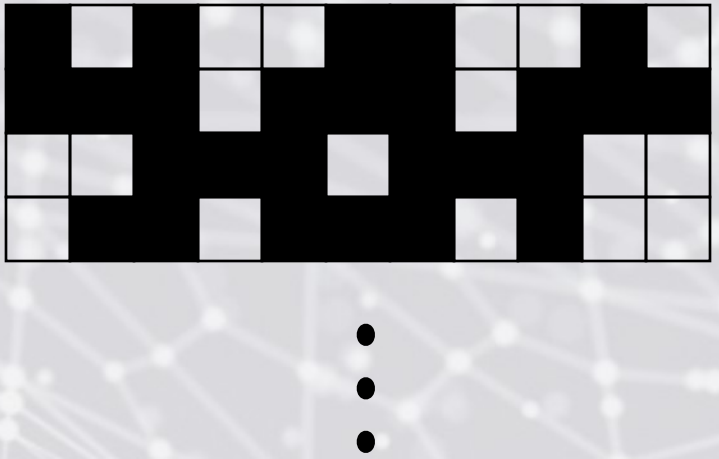




One-dimensional cellular automata

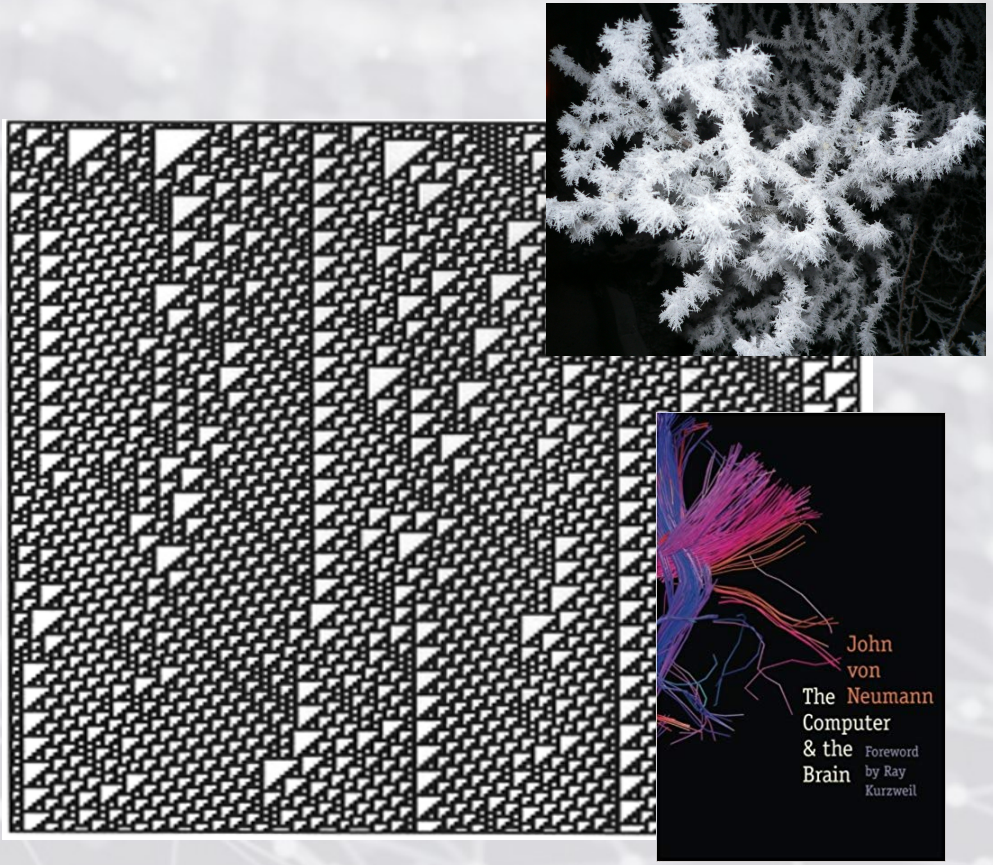
Rule:





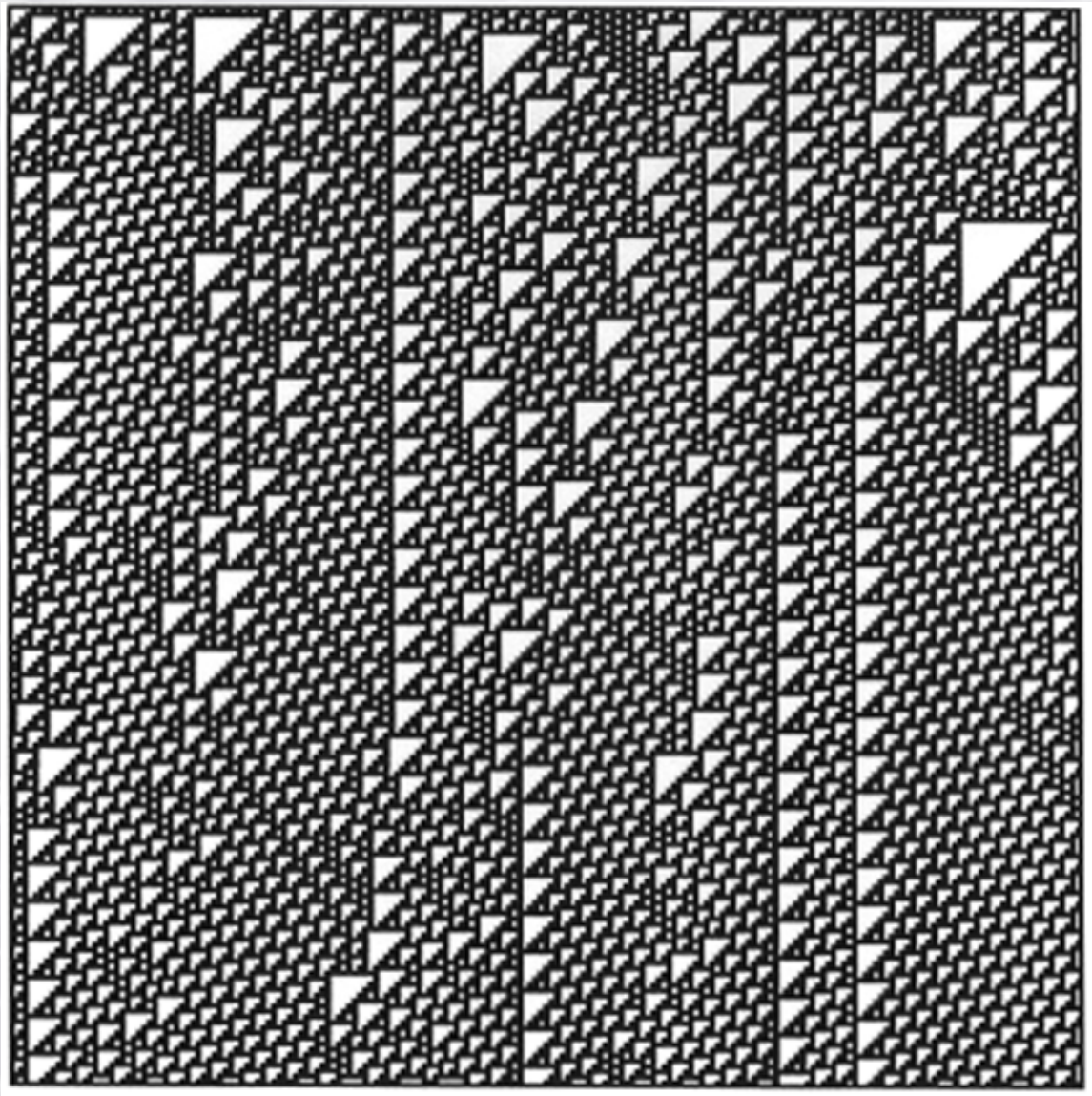
von Neumann

S. Ulam



(*) In the 1940s von Neumann defined an organism as constituting two traits: (1) capacity for reproduction, (2) it can simulate a Turing machine.

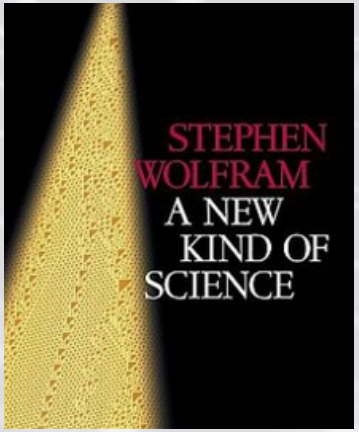
(*) While working at Los Alamos (with von Neumann), S. Ulam invented cellular automata as a model for self-replicating systems (based on the study of the growth of crystals).



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

Wolfram: A New Kind of Science

https://www.youtube.com/watch?v=60P7717-XOQ



Conway's Game of Life

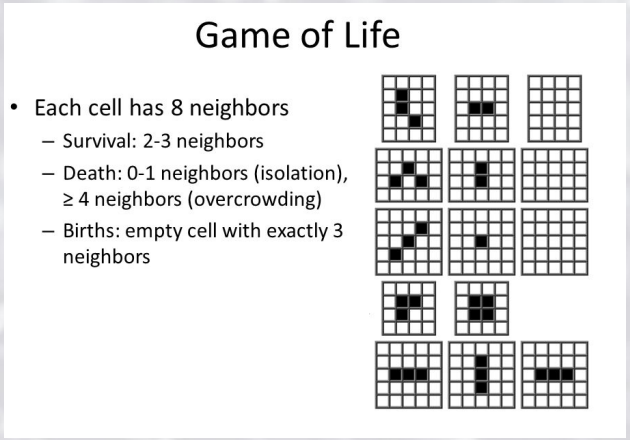
John Conway (Princeton) introduced a 2-D cellular automata, "Game of Life" in 1970.

It is a *zero-player game* (i.e. completely determined by initial configuration and rule set). One interacts with the Game of Life by creating an initial configuration and observing how it evolves, or, for advanced players, by creating patterns with particular properties.

- Every cell interacts with its eight "neighbors" which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

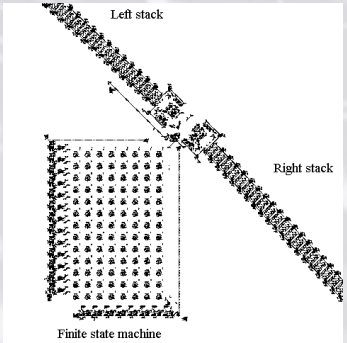
1. Any live cell with fewer than two live neighbors dies, as if by underpopulation.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Conway's Game of Life

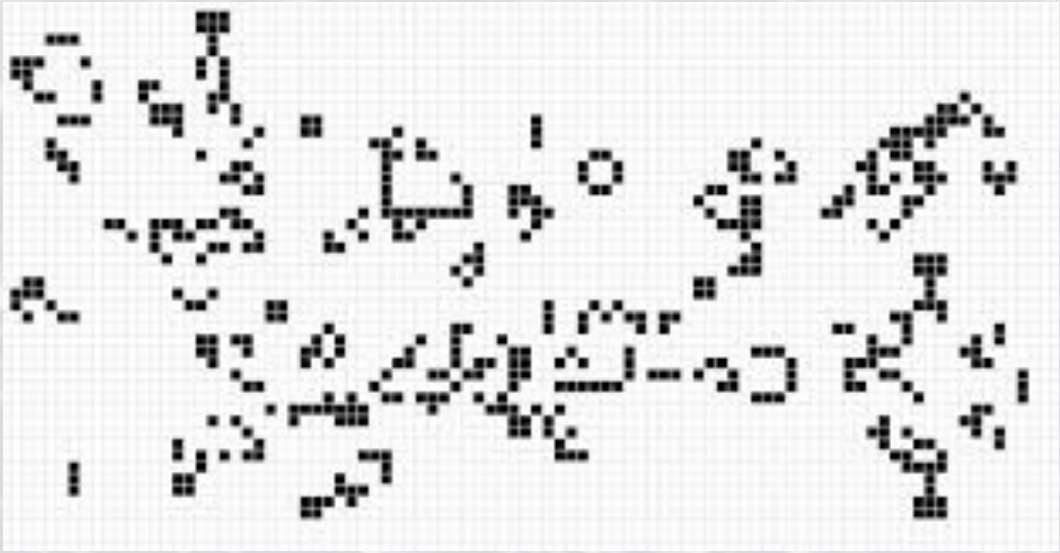


- *Game of Life* is **undecidable** (given an initial pattern and a later pattern, no such algorithm exists that can tell whether the later pattern is ever going to appear – this is a corollary to the **halting problem**).
- Game of Life includes a pattern that is equivalent to a Universal Turing Machine (UTM)
- Some patterns exist that remain chaotic forever. If this were not the case, one could progress the game sequentially until a non-chaotic pattern emerged, then compute whether a later pattern was going to appear.

http://pentadecathlon.com/lifeNews/2011/01/turing_machines_1.html



Conway's Game of Life



Some Demos:

https://bitstorm.org/gameoflife/

https://academo.org/demos/conways-game-of-life/

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

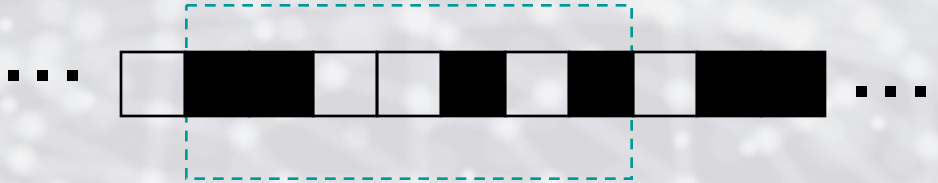
initial

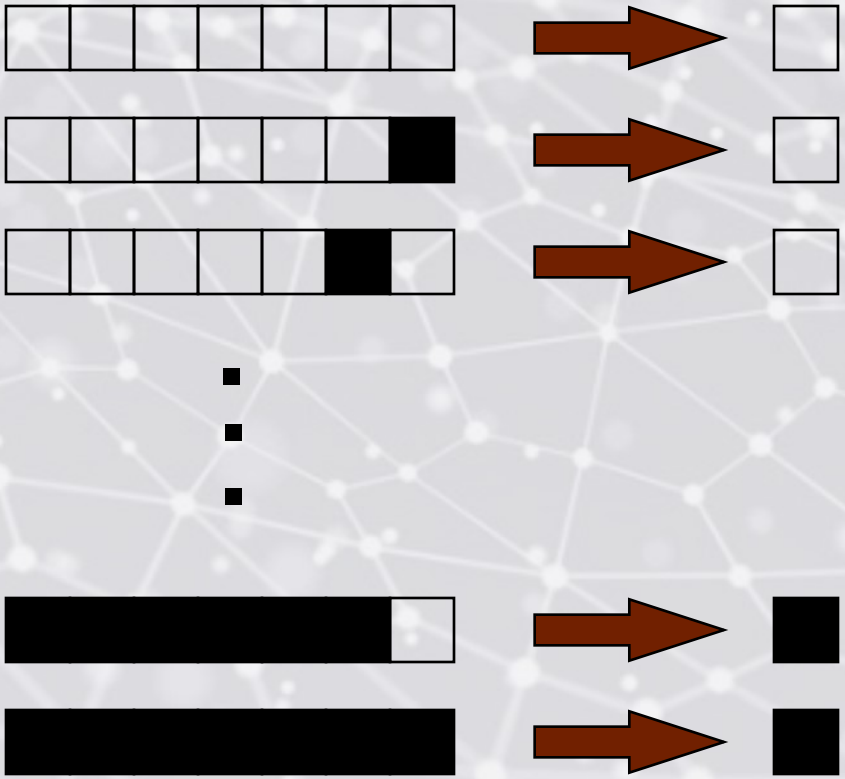
How to design a cellular automaton
that will do this?

final

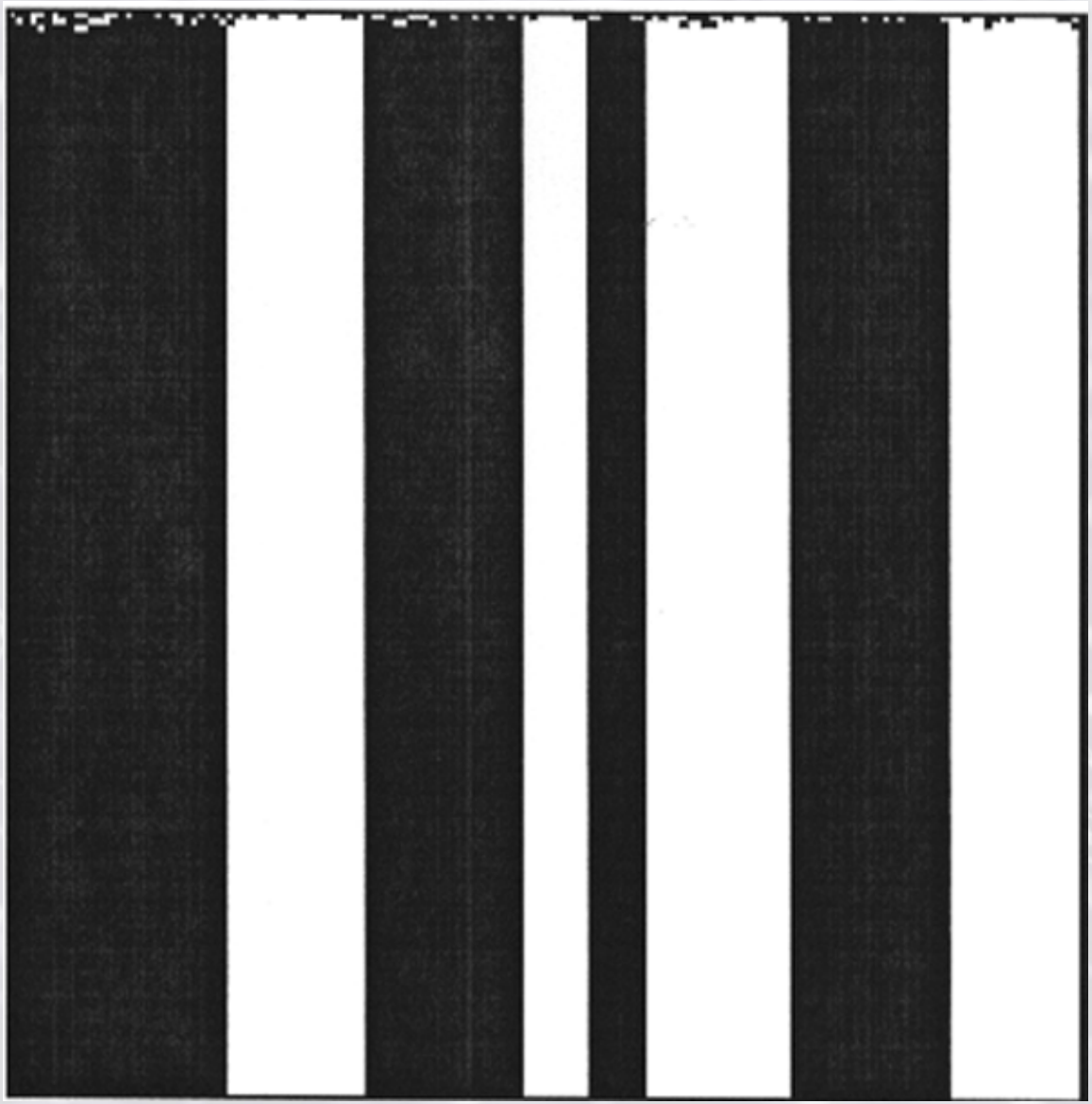
We used cellular automata with 6 neighbors for each cell:

Rule:





A candidate solution that does not work: Local majority voting

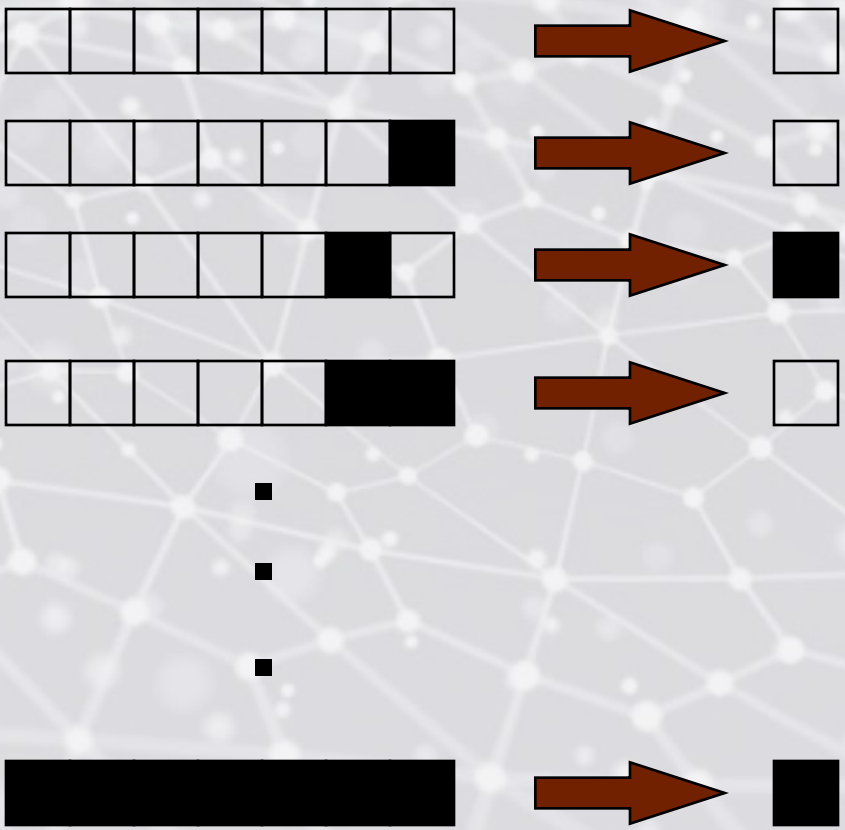


Evolving cellular automata with genetic algorithms

- 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 and crossovers.
- This process continues for many generations.

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

Rule table:



Chromosome

0

0

1

0

.

.

.

1

Create a random population of candidate cellular automata rules:

rule 1: 00100011000100101111000101001101110000...
rule 2: 00011001101010111111110000111010010 10...
rule 3: 11111000100101010000000111000100101 01...

.
.
.

rule 100: 00101110100000011111000001010010111 11...

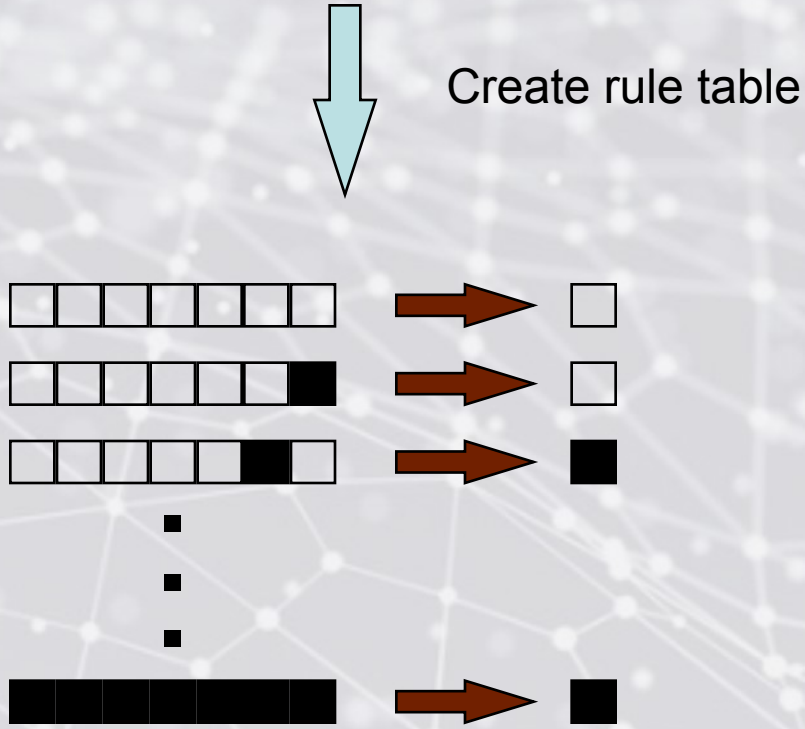
Calculating the Fitness of a Rule

- For each rule, create the corresponding cellular automaton. Run that cellular automaton on many initial configurations.

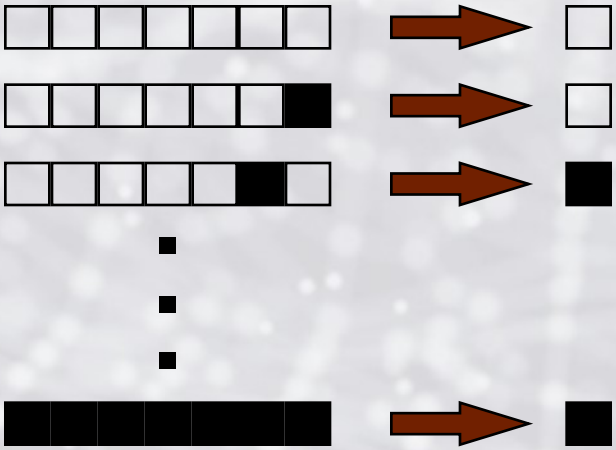
- Fitness of rule = fraction of correct classifications

For each cellular automaton rule in the population:

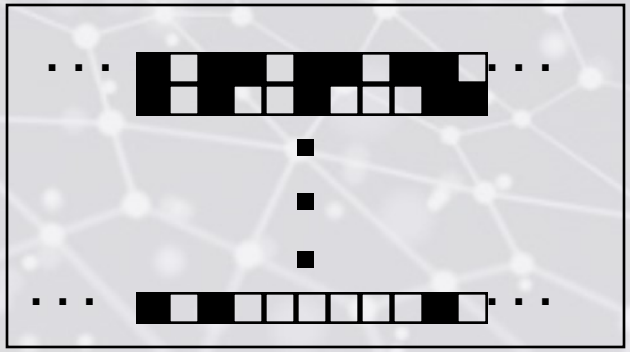
rule 1: 00100011000100101111000101001101110000...1



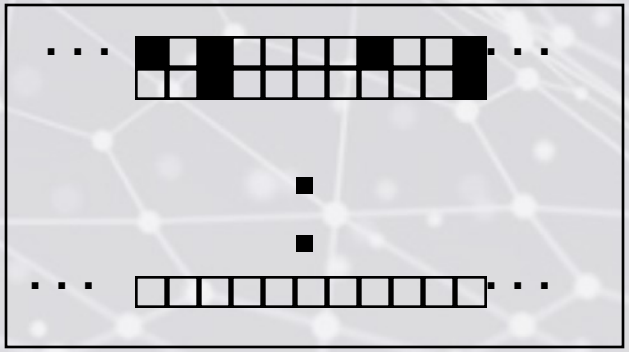
rule 1 rule table:



Run corresponding cellular automaton on many random initial lattice configurations



incorrect



correct

etc.

Fitness of rule = fraction of correct classifications

GA Population:

rule 1: 00100011000100101111000101001101110000... Fitness = 0.5
rule 2: 00011001101010111111110000111010010 10... Fitness = 0.2
rule 3: 11111000100101010000000111000100101 01... Fitness = 0.4

.
.
.

rule 100:00101111010000001111100000101001011111... Fitness = 0.0

Select fittest rules to reproduce themselves

rule 1: 00100011000100101111000101001101110000... Fitness = 0.5
rule 3: 11111000100101010000000111000100101 01... Fitness = 0.4

.
.
.

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 1000100101111000101001101110000...
rule 3: 1111100 0100101010000000111000100101010...

mutate

Children:

0010001010010101000000011100010010101...
1111100 1000100101111000101001101110000...

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 1000100101111000101001101110000...
rule 3: 1111100 0100101010000000111000100101010...

mutate

Children:

0010000010010101000000011100010010101...
1111100 1000100101111000101001101110000...

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 1000100101111000101001101110000...
rule 3: 1111100 0100101010000000111000100101010...

Children:

0010000010010101000000011100010010101...
1111100 1000100101111000101001101110000...

mutate

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 1000100101111000101001101110000...
rule 3: 1111100 0100101010000000111000100101010...

Children:

0010000010010101000000011100010010101...
1111100 1000100101111000101000101011100 0...

mutate

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 1000100101111000101001101110000...
rule 3: 1111100 0100101010000000111000100101010...

Children:

0010000010010101000000011100010010101...
1111100 1000100101111000101000010111000...

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 1000100101111000101001101110000...
rule 3: 1111100 0100101010000000111000100101010...

Children:

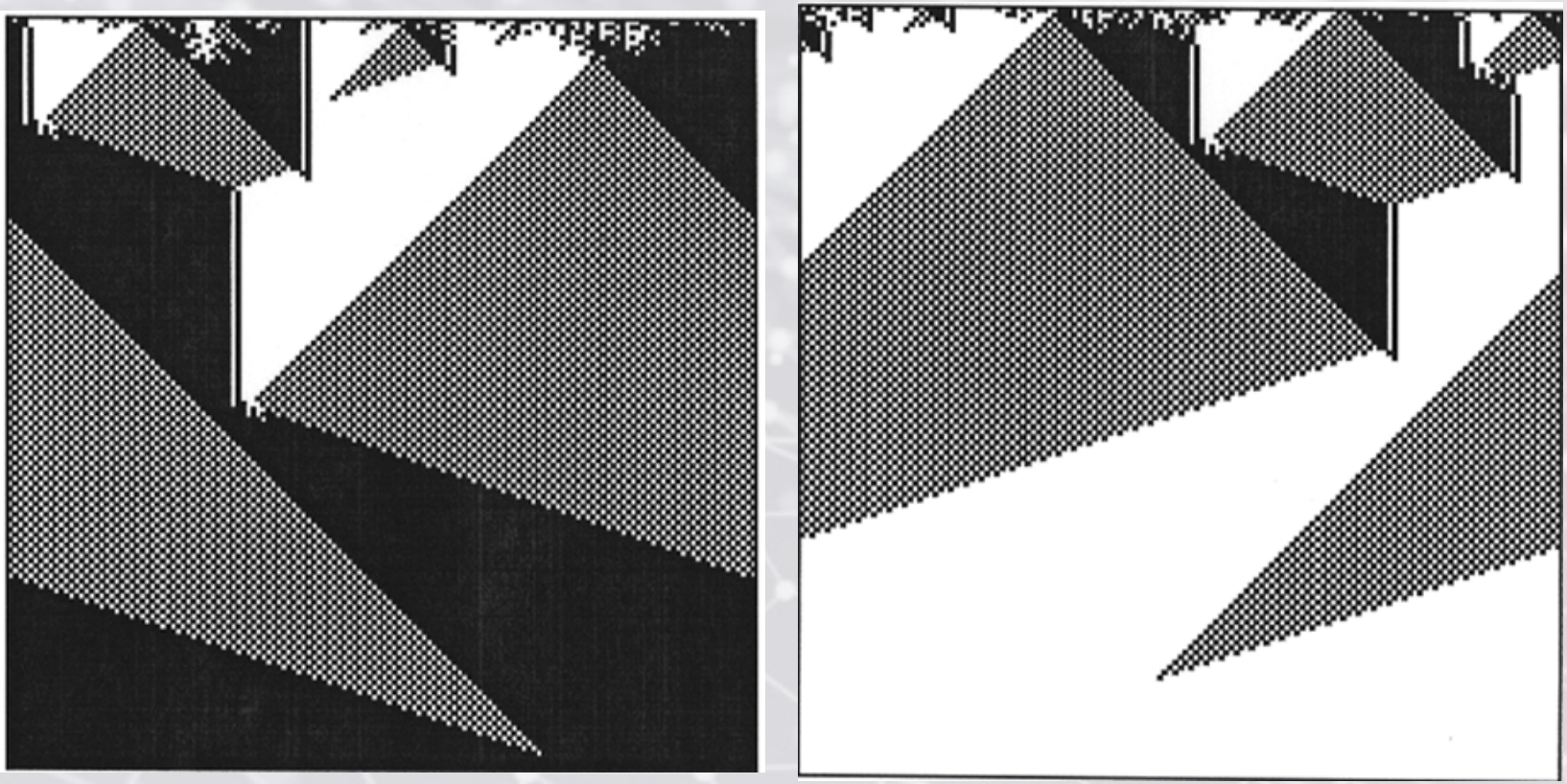
0010000010010101000000011100010010101...
1111100 1000100101111000101000101110000...

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

Keep iterating for many generations.

majority on

majority off

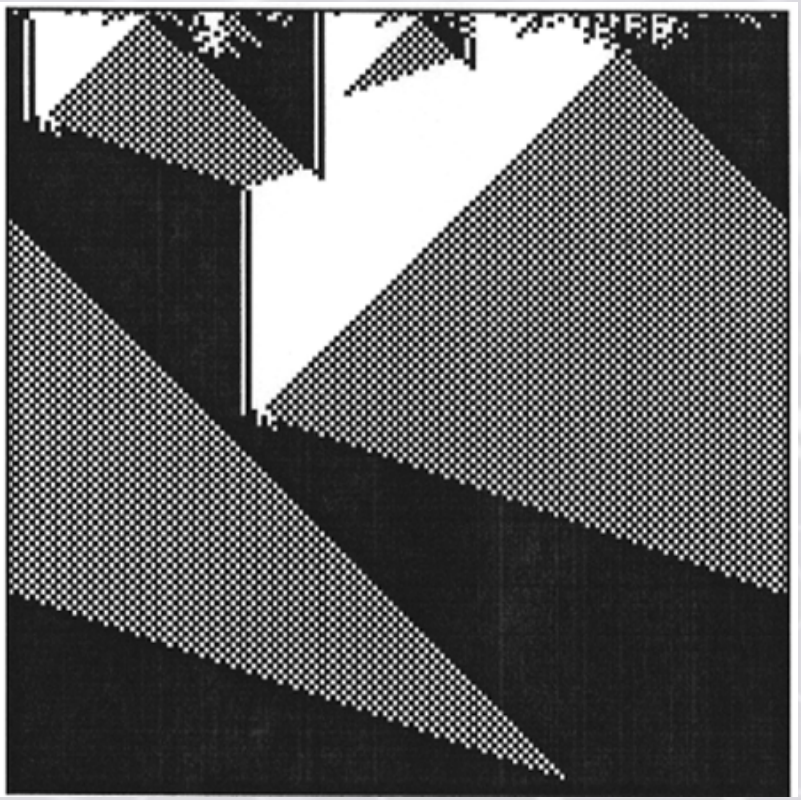


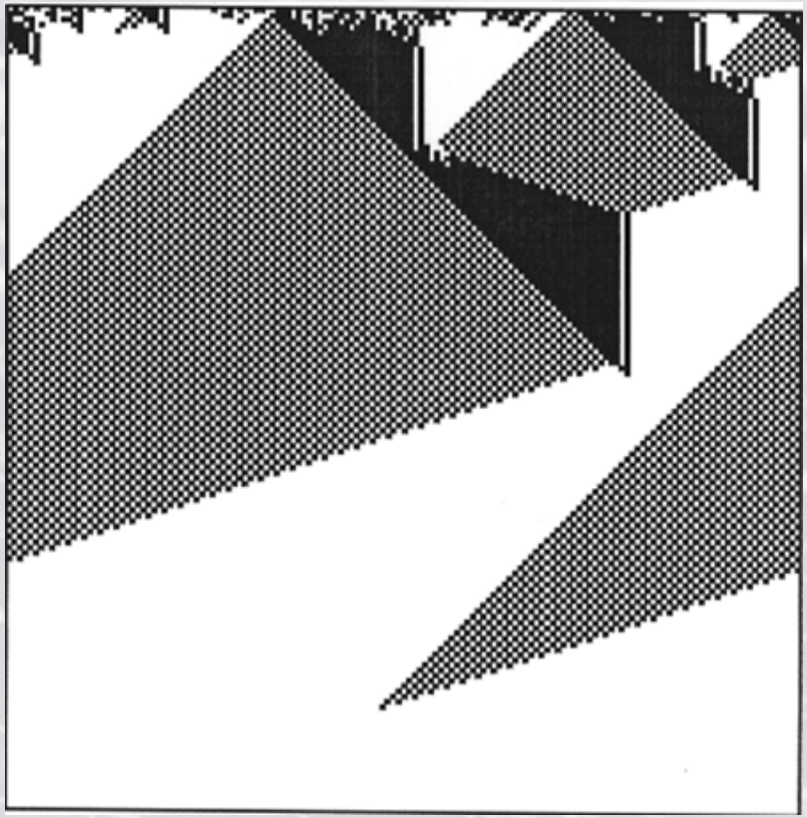
A cellular automaton evolved by
the genetic algorithm

How does it perform the computation?

majority on

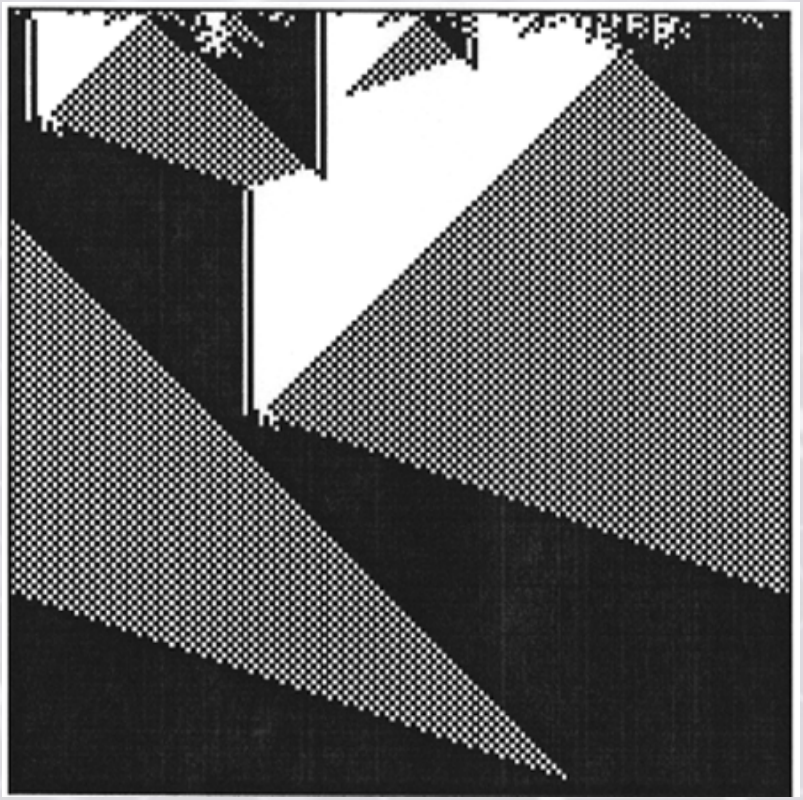
majority off



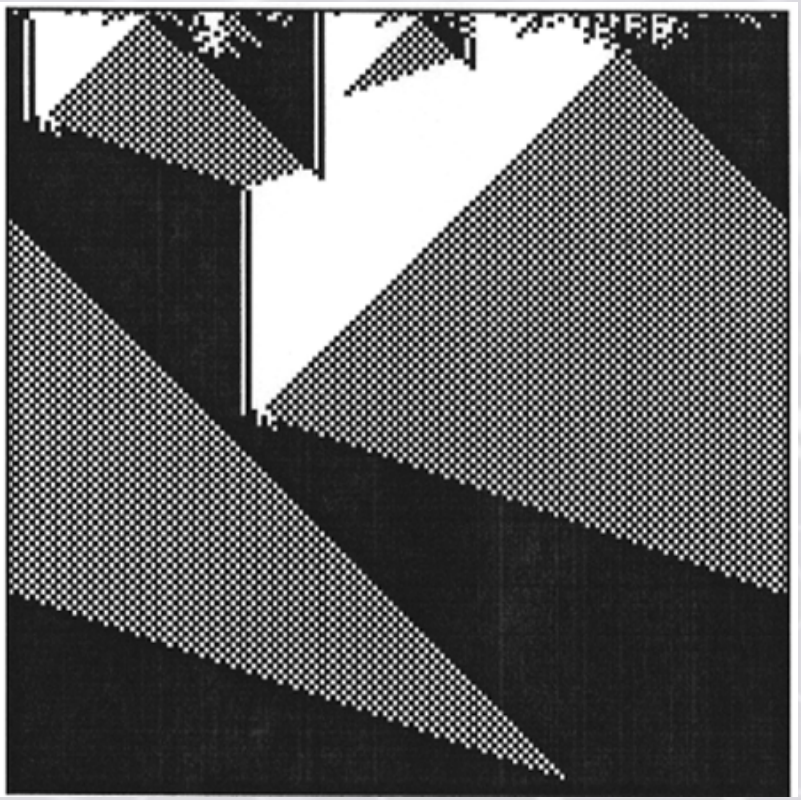


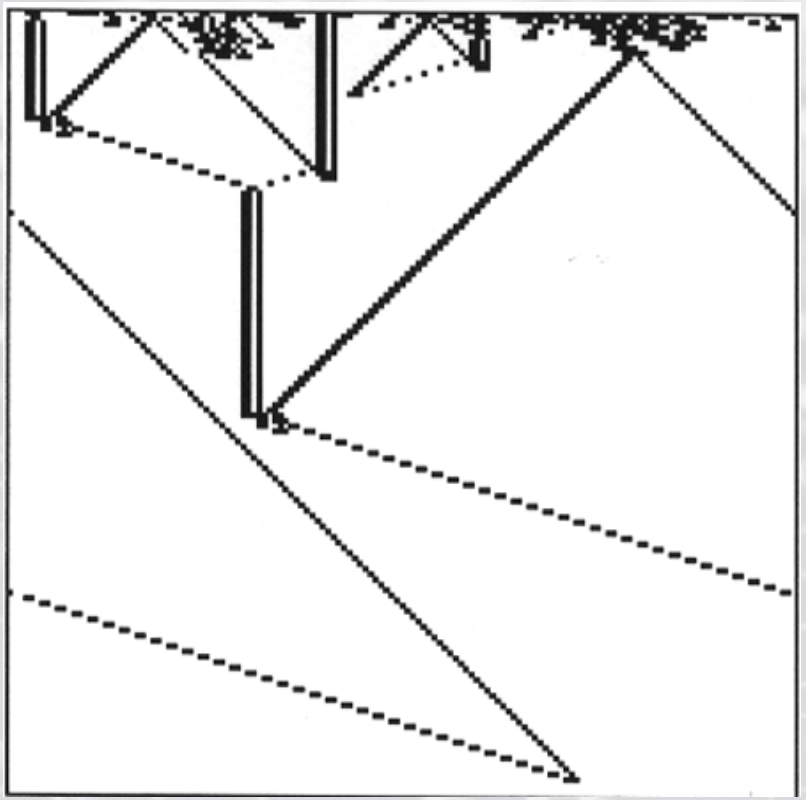
A cellular automaton evolved by the genetic algorithm

How do we describe information processing in complex systems?

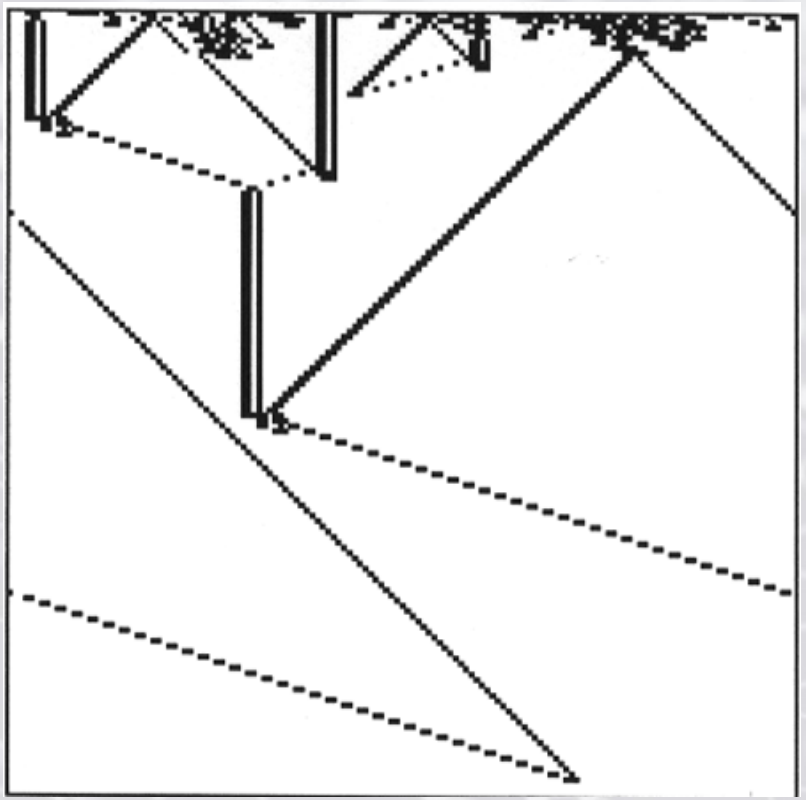


“simple patterns”:
black, white, checkerboard

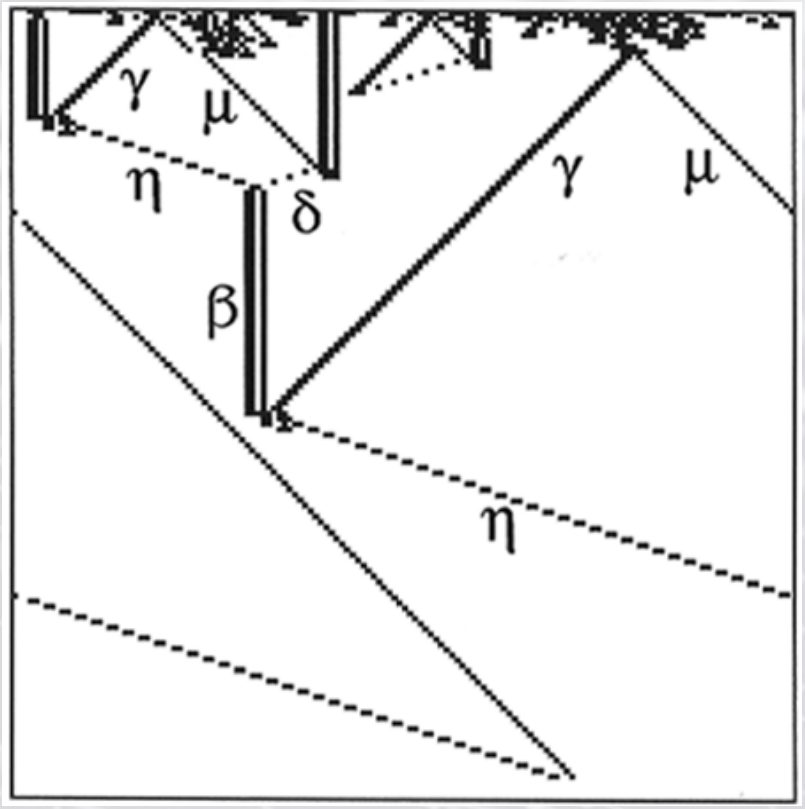




Simple patterns
filtered out



“particles”

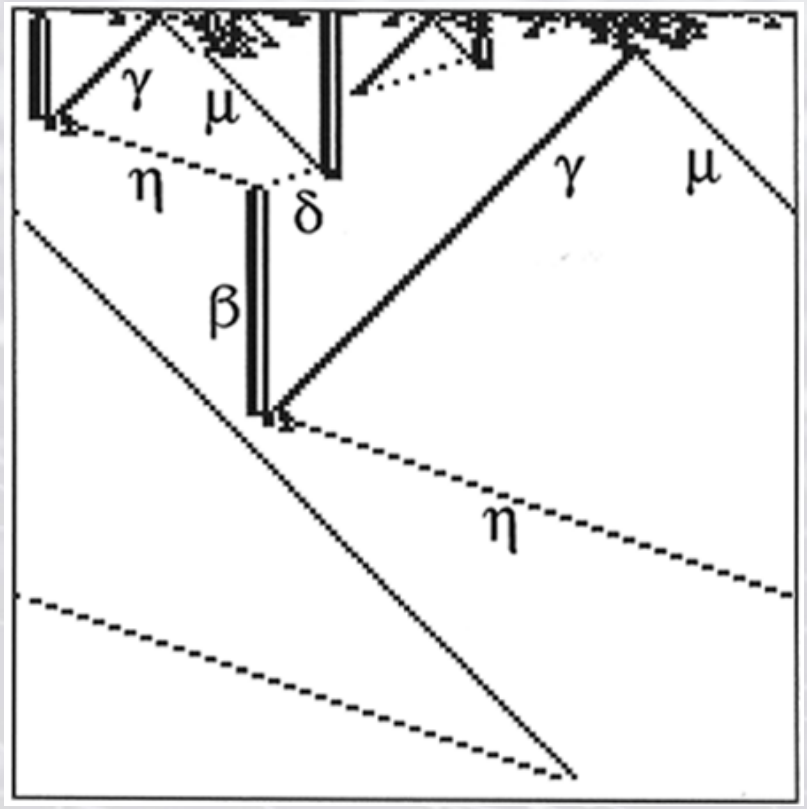


“particles”

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	$\alpha \to \gamma + \mu$
react	$\beta + \gamma \to \eta,\ \mu + \beta \to \delta,\ \eta + \delta \to \beta$
annihilate	$\eta + \mu \to \emptyset_1,\ \gamma + \delta \to \emptyset_0$



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

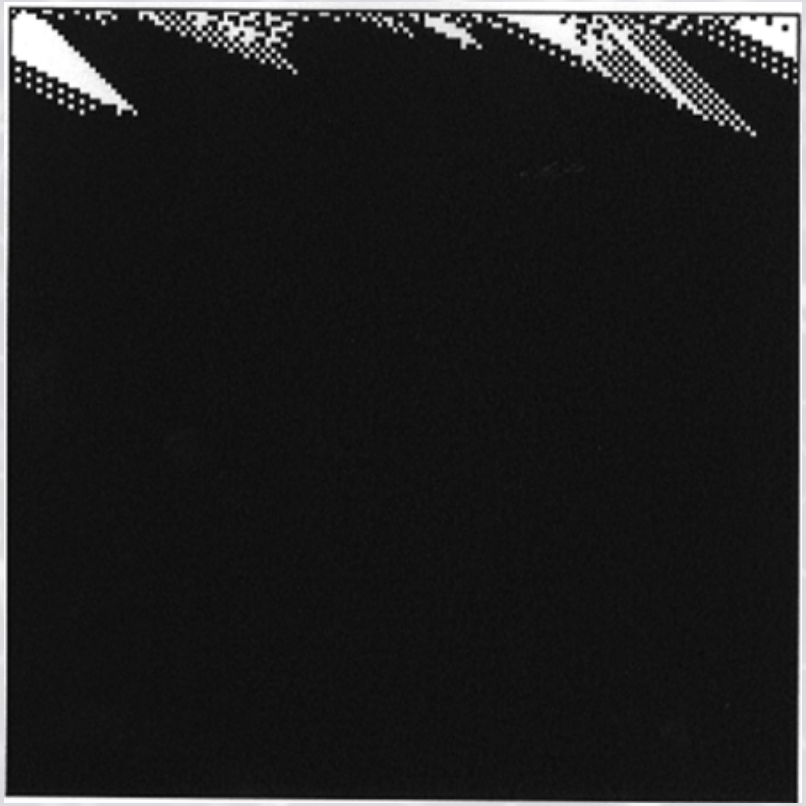
- Level of particles can explain:
  - Why one CA is fitter than another
  - What mistakes are made
  - How the GA produced the observed series of innovations

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

  ----> “Algorithmic” level

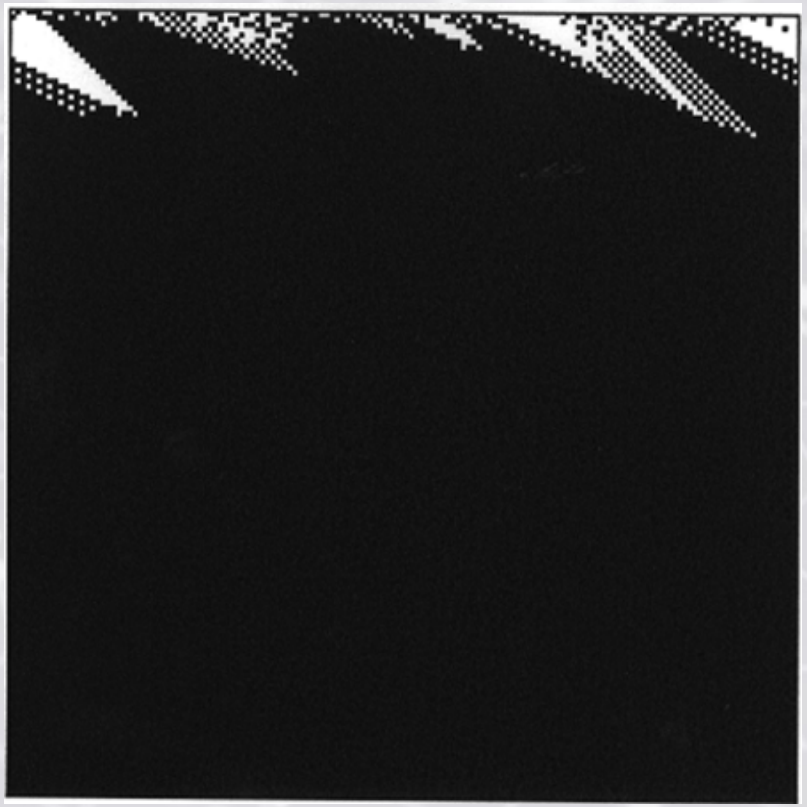
How the genetic algorithm evolved cellular automata

How the genetic algorithm evolved cellular automata



generation 8

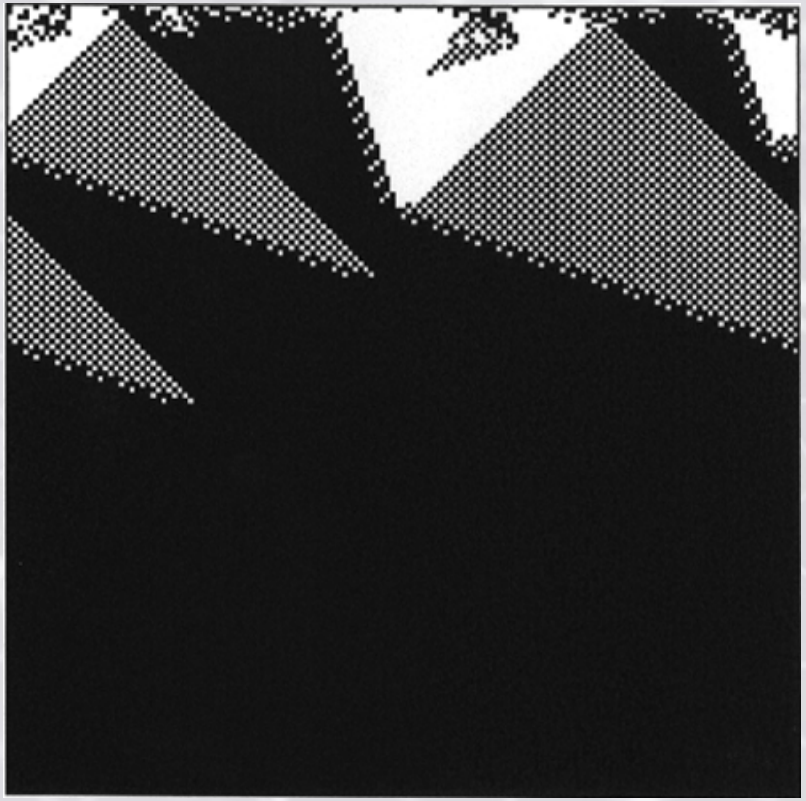
How the genetic algorithm evolved cellular automata



generation 8

generation 13

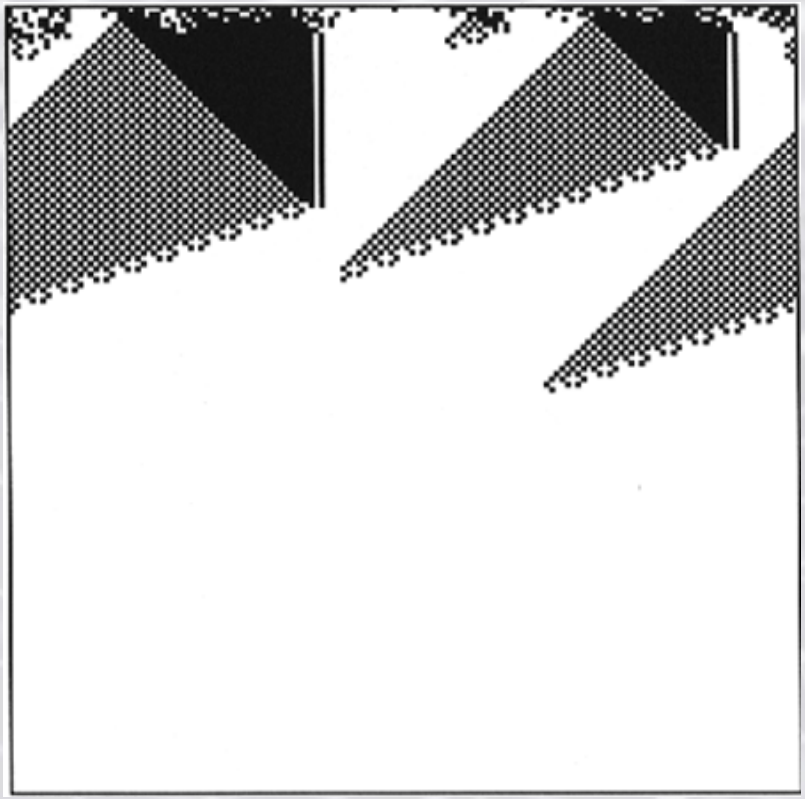
How the genetic algorithm evolved cellular automata



generation 17

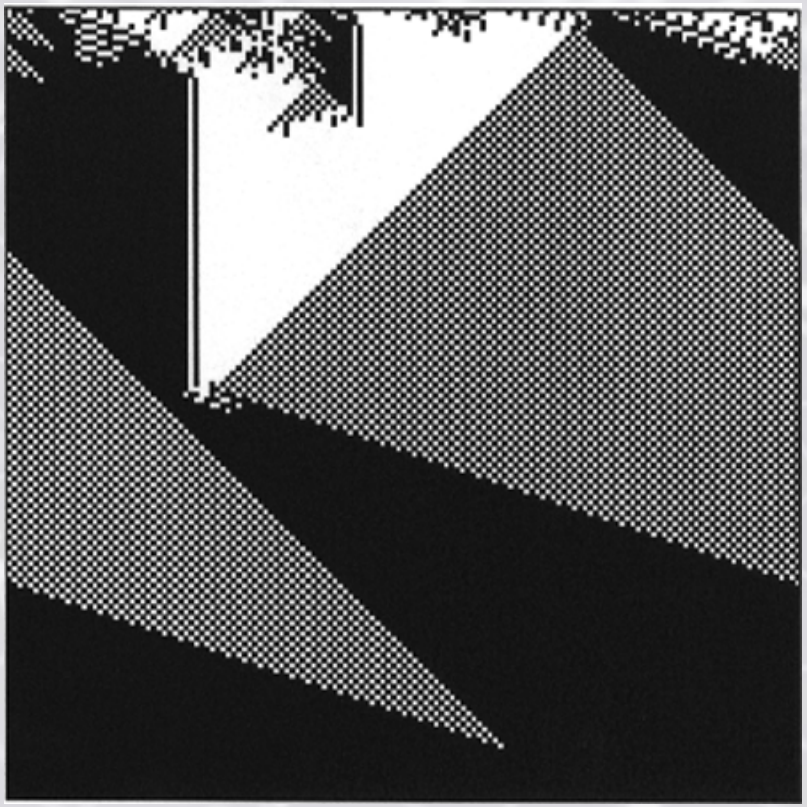
How the genetic algorithm evolved cellular automata

generation 17



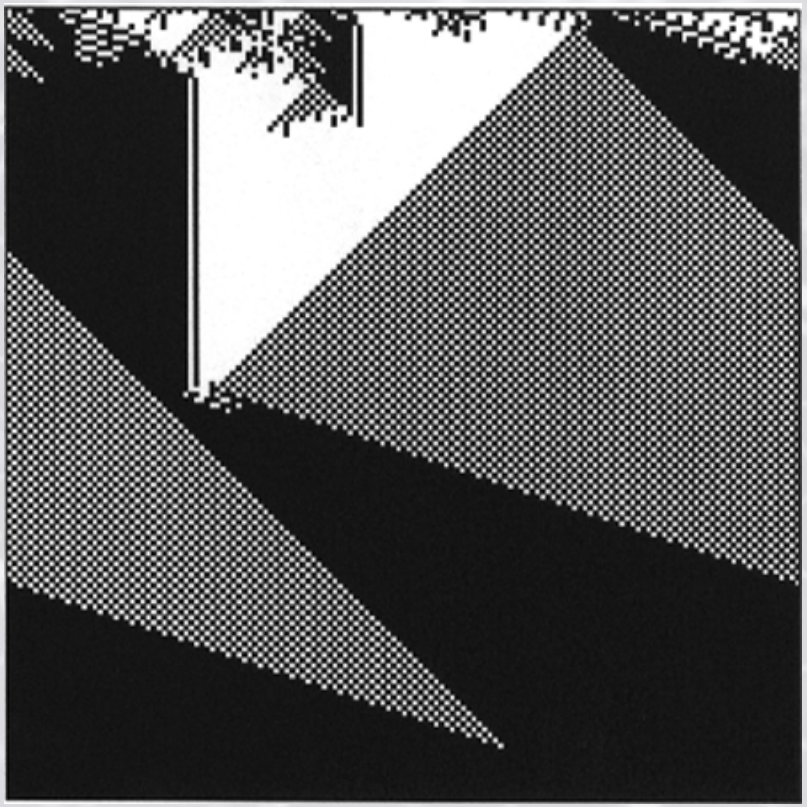
generation 18

How the genetic algorithm evolved cellular automata

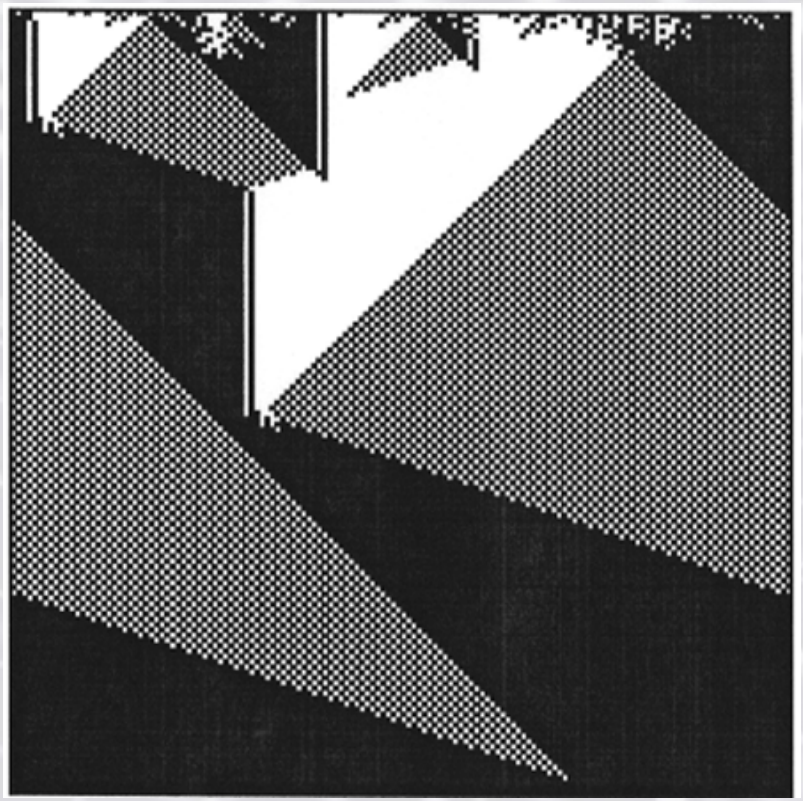


generation 33

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

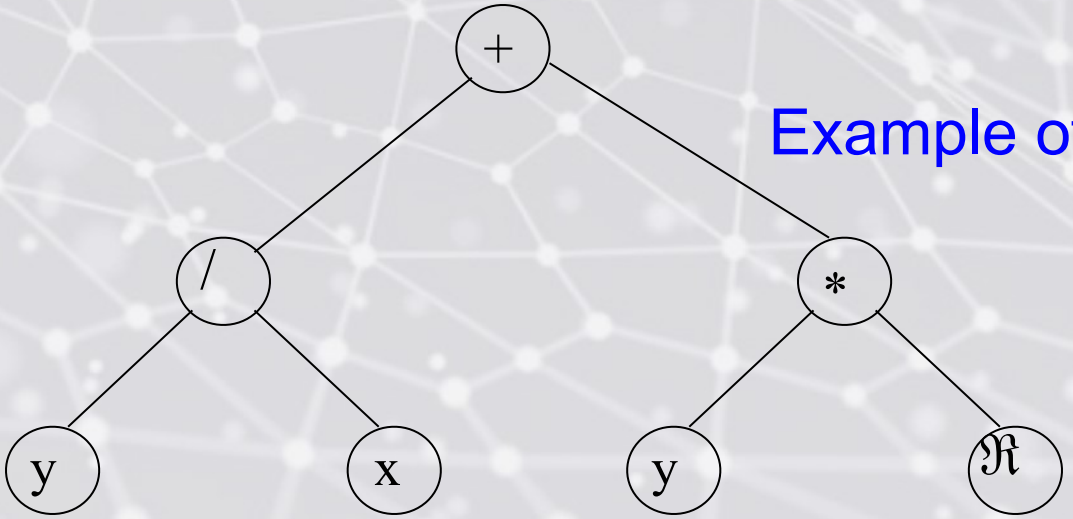
1. **Function induction:** 2D function-induction task (Pagie & Hogeweg, 1997)

   – Evolve function tree to approximate

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

– Hosts are candidate trees

- Function set: {+, -, ∗, %}
- Terminal set: {x, y, ℜ}



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

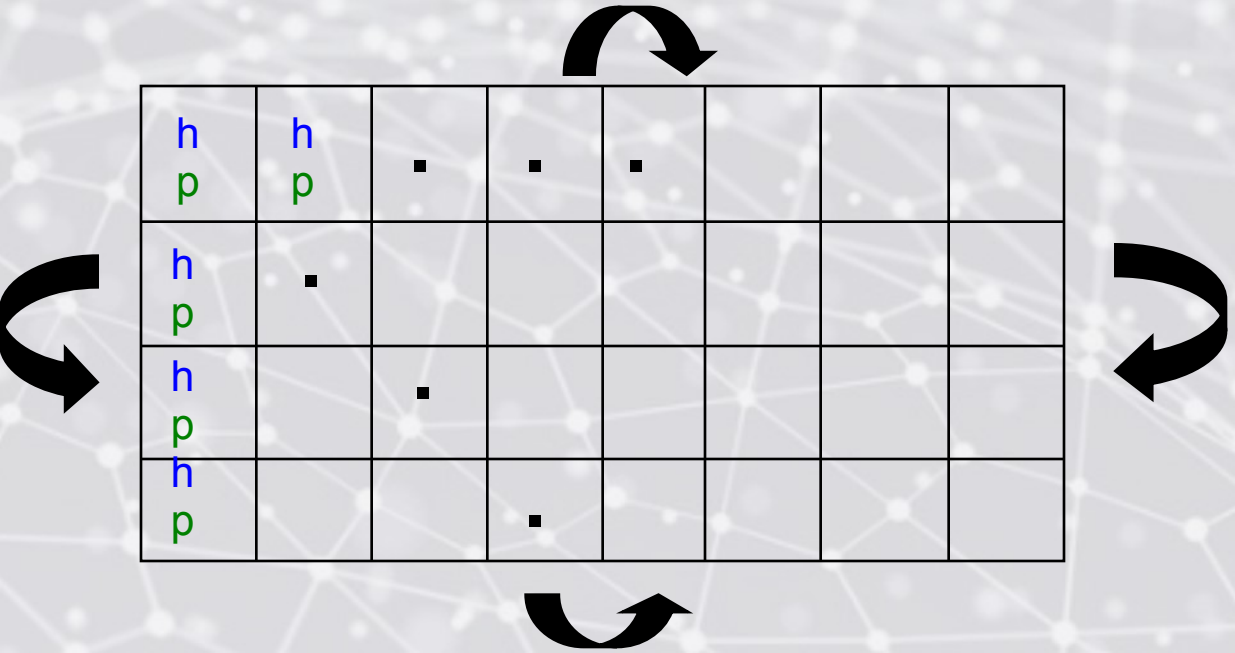
- Problem is to design 1D CA that classifies initial configurations (ICs) as “majority 1s” or “majority 0s”.

Spatial Coevolution

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

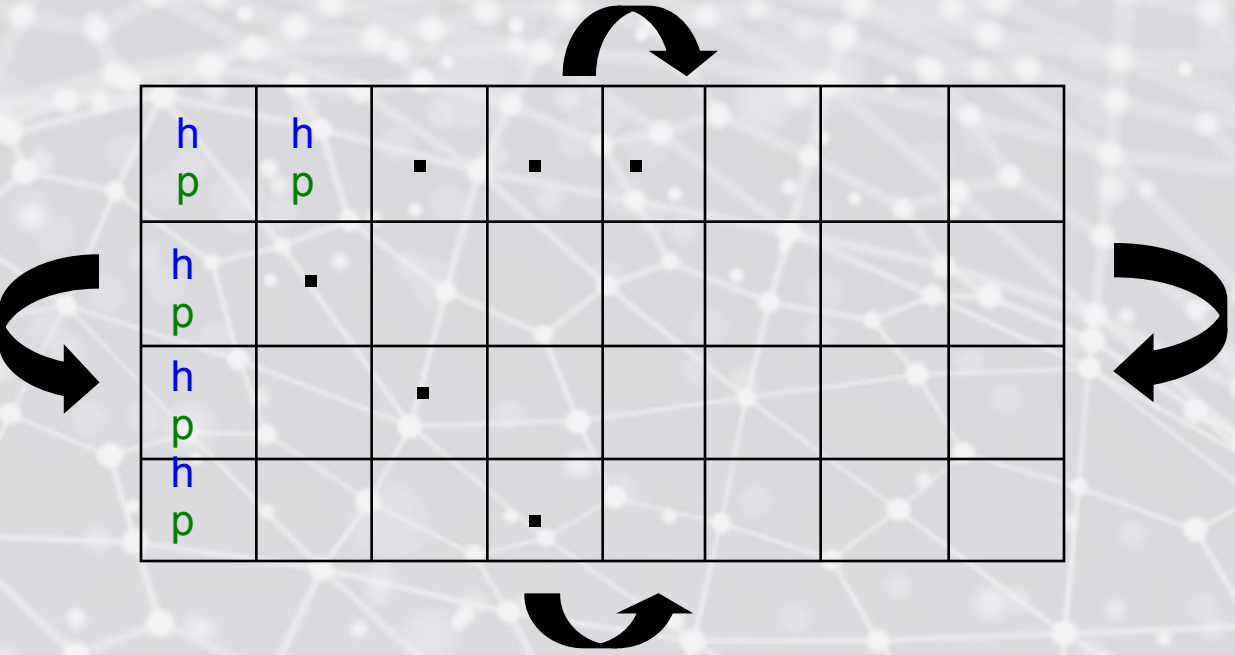
Spatial Coevolution

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



Spatial Coevolution

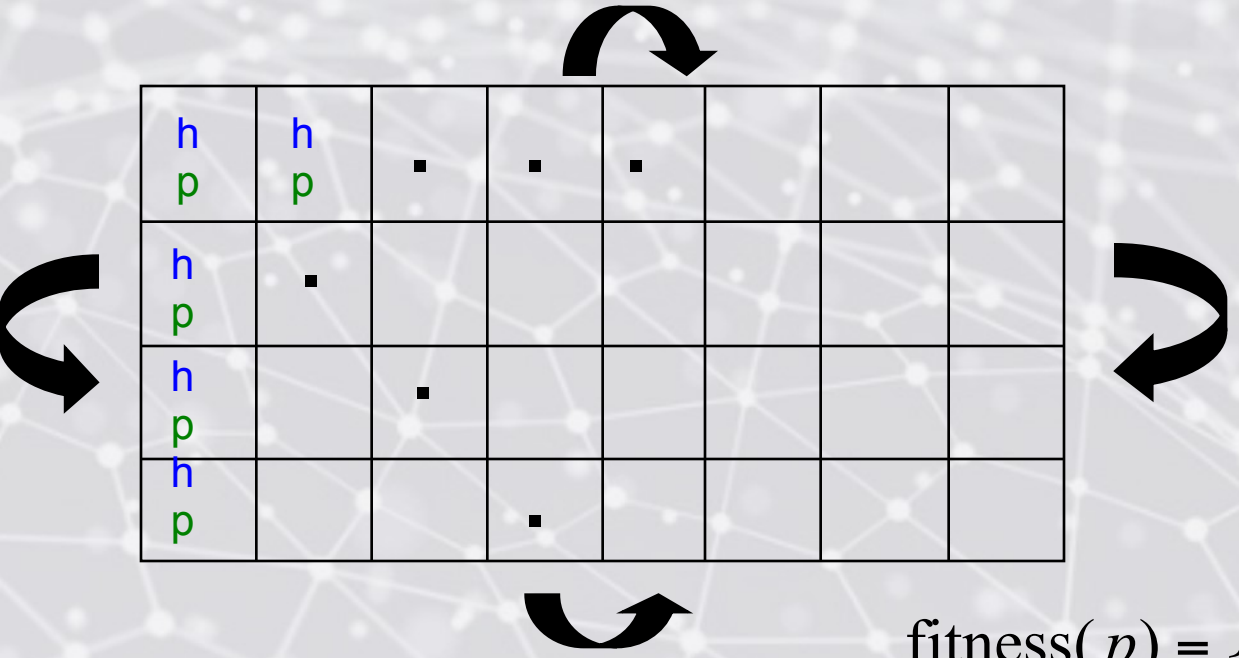
- 2D toroidal lattice with one host (*h*) and one parasite (*p*) per site



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

Spatial Coevolution

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

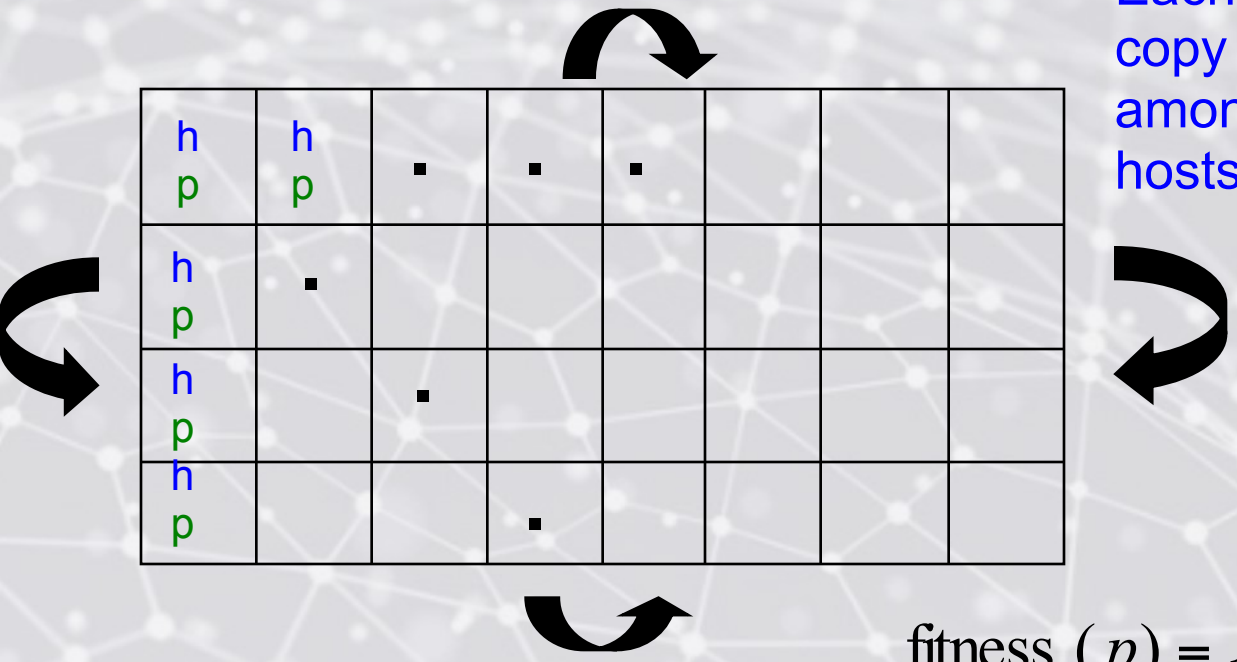


$$\text{fitness}(p) = \begin{cases} 0 & \text{if } h(\text{p}) \text{ is correct} \\ > 0 & \text{if } h(p) \text{ is not correct} \end{cases}$$

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

Spatial Coevolution

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



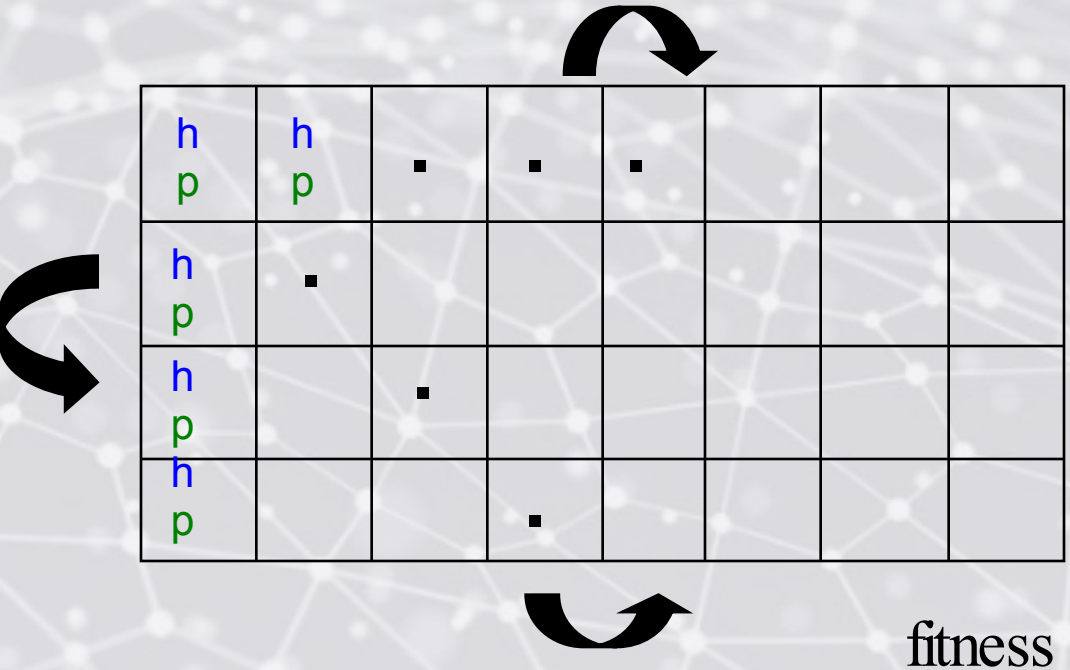
Each *h* is replaced by mutated copy of winner of tournament among itself and 8 neighboring hosts.

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

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

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.

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

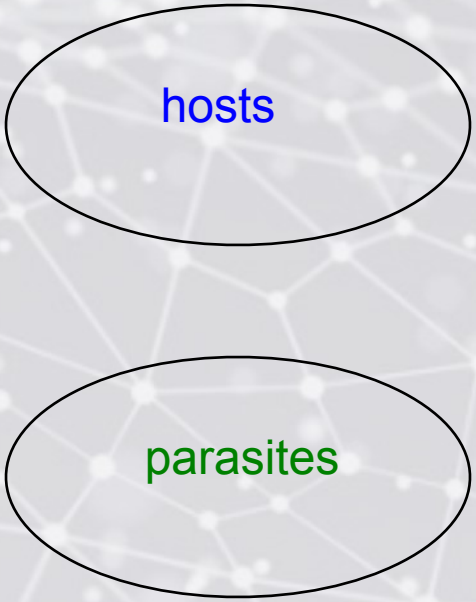
$$\text{fitness}(h) = \text{fraction of 9 neighborin g } p \text{ answered correctly}$$

Non-Spatial Coevolution

- No spatial distribution of host and parasite populations

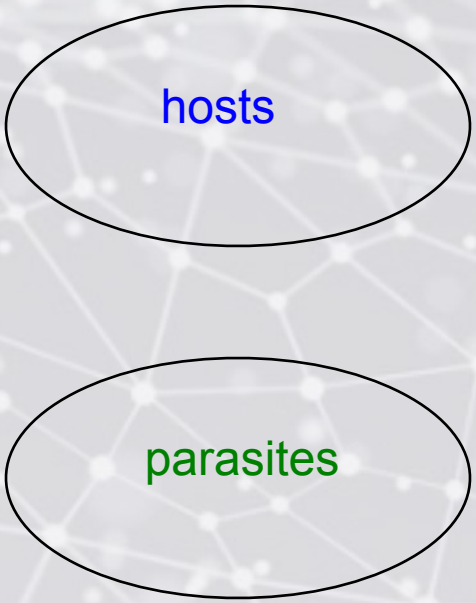
Non-Spatial Coevolution

- No spatial distribution of host and parasite populations



Non-Spatial Coevolution

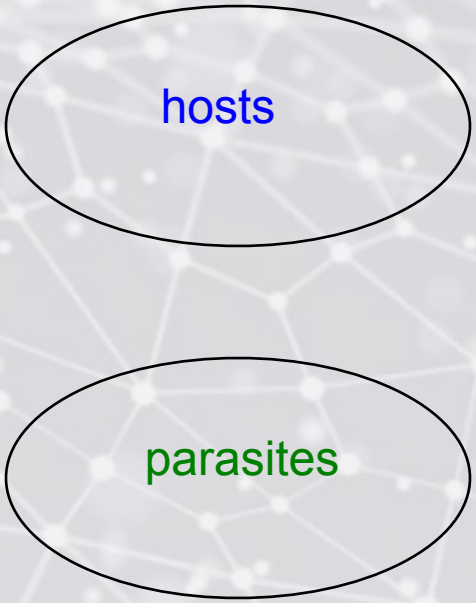
- No spatial distribution of host and parasite populations



fitness (h) = fraction of 9 *parasites p*
(randomly chosen from parasite population)
answered correctly

Non-Spatial Coevolution

- No spatial distribution of host and parasite populations



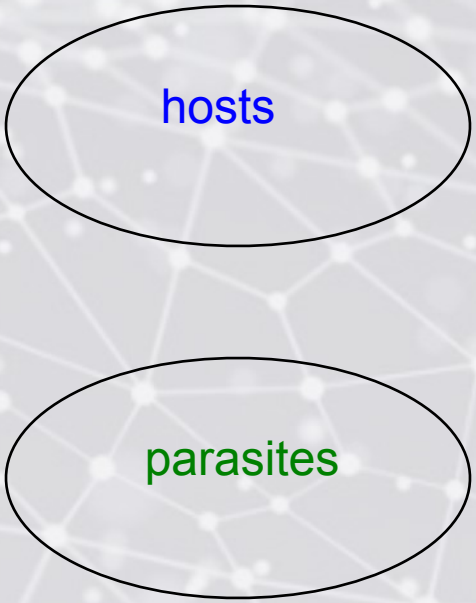
fitness (h) = fraction of 9 *parasites p* (randomly chosen from parasite population) answered correctly

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

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.

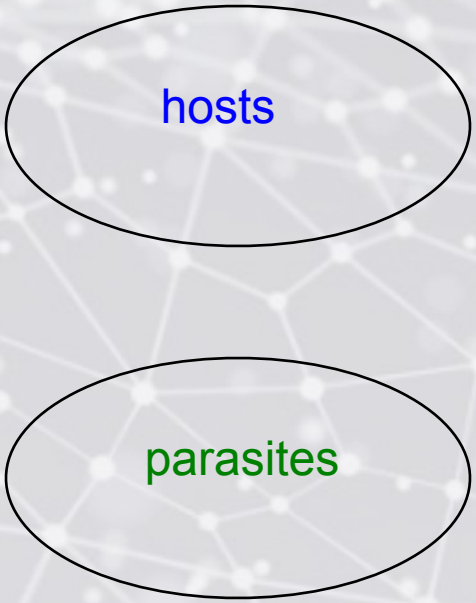
fitness (h) = fraction of 9 *parasites p* (randomly chosen from parasite population) answered correctly

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

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.

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

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

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

for host h randomly chosen from host population

- **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